React JS

Introduction

- ✓ The React.js is an open-source JavaScript framework and library developed by Facebook. It's used for building interactive user interfaces and web applications quickly and efficiently.
- ✓ Primary role of React in an application is to handle the view layer of that application just like the V in a model-view-controller (MVC) pattern by providing the best and most efficient rendering execution.
- ✓ Before starting we should have a basic understanding of HTML, CSS, DOM, ES6, Node.js and npm.
- ✓ Because of its ability to create fast, efficient, and scalable web applications, React has gained stability and popularity. Thousands of web applications use it today, from well-established companies to new start-ups. Some of the popular examples are as under:

Facebook, Instagram, Netflix, Reddit, Uber, Airbnb, The New York Times, Khan Academy, Codecademy, WhatsApp Web

History

- ✓ React was created by Jordan Walke, a software engineer at Meta, who initially developed a prototype called "F-Bolt", later renaming it to "FaxJS".
- ✓ This early version is documented in Jordan Walke's GitHub repository. Influences for the project included XHP, an HTML component library for PHP.
- ✓ React was first deployed on Facebook's News Feed in 2011 and subsequently integrated into Instagram in 2012.
- ✓ In May 2013, at JSConf US, the project was officially open-sourced.

About React

- ✓ **Component based approach:** A component is one of the core building blocks of React. In other words, we can say that every application you will develop in react will be made up of pieces called components. Components make the task of building UIs much easier.

- ✓ **Uses a declarative approach:** Declarative programming is a programming paradigm that expresses the logic of a computation without describing its control flow.
- ✓ DOM (Document Object Model) updates are handled gracefully.
- ✓ Reusable code.

Setup of React JS

- **Step 1:** Install Node.js installer for windows.

- **Step 2:** Open command prompt or terminal in VS code to check whether it is completely installed or not.

```
node -v
```

If the installation went well it will give you the version you have installed

- **Step 3:** Now Create a new folder where you want to make your react app using the below command:

```
mkdir react
```

***Note:** The **react** in the above command is the name of the folder and can be anything.*

Move inside the same folder using the below command:

```
cd react
```

- **Step 4:** Now, go inside **folder(react)**.

- ✓ The **npm** stands for **Node Package Manager** and it is the default package manager for **Node.js.** It gets installed into the system with the **installation of node.js.**
- ✓ The **npx** stands for **Node Package Execute** and it comes with the npm, when you installed npm above 5.2.0 version then automatically npx will be installed. It is an npm package runner that can execute any package that you want from the npm registry without even installing that package. Using the NPX package runner to execute a package can also help reduce the pollution of installing lots of packages on your machine.

In npx you can create a react app without installing the package:

`` `npx create-react-app myApp` ``

This command is required in every app's life cycle only once.

Run below command if not able to create a react app using above command.

`` `npm install -g create-react-app` ``

By creating a react app folder structure will be looked as shown in next topic called "Folder Structure" image.

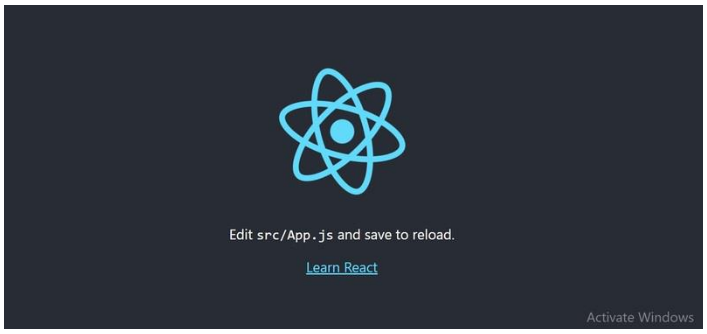
- **Step 5:** Now, go to **myApp**(Created react app) folder

```
cd myApp
```

- **Step 6:** To start your app run the below command:

```
npm start
```

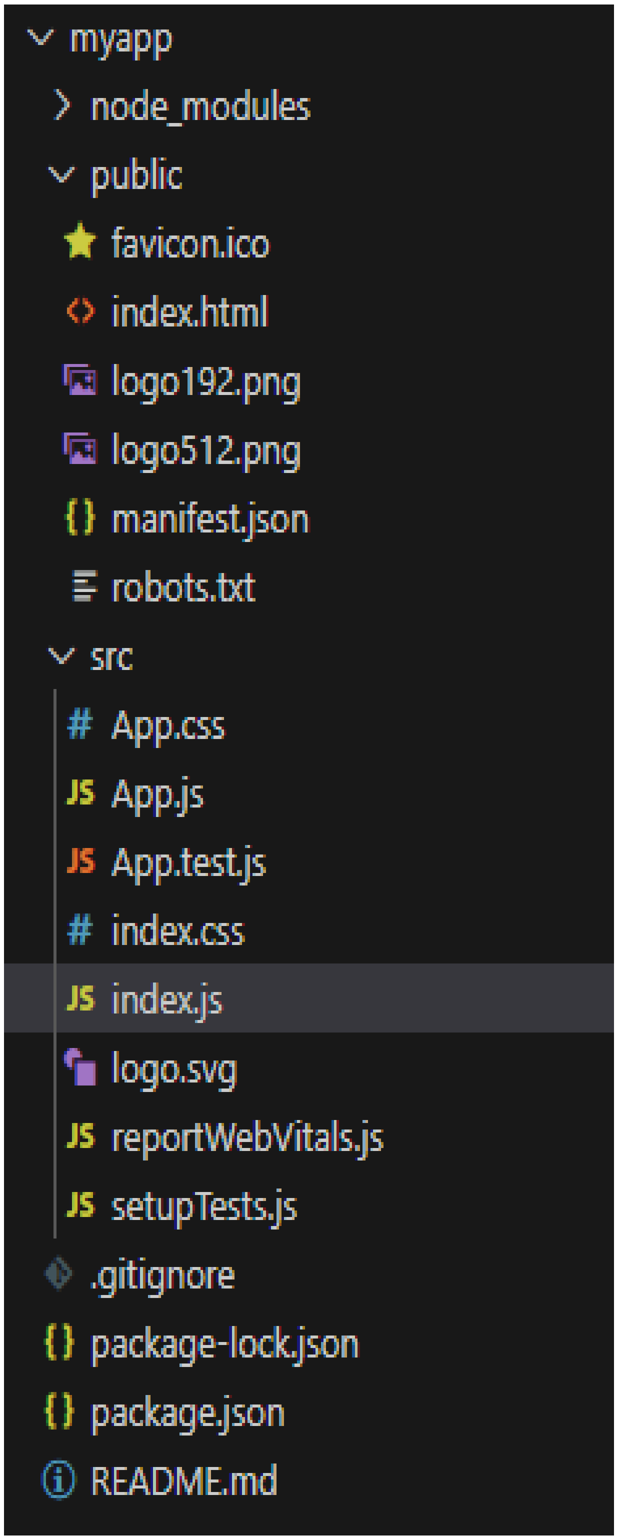
Once you run the above command, a new tab will open in your browser showing React logo as shown below:



Edit src/App.js and save to reload.

Folder Structure

Once a react app gets created, The folder structure looks as below.



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

node_modules

- ✓ In this folder, you will get various folders of all the required dependencies & packages that may be used for building your react app. For example – Webpack, Babel, JSX, Jest & more.
- ✓ You not need to modify the node_module.
- ✓ It is already configured with the react app.

Public

- ✓ The public folder contains static files such as index.html, javascript library files, images, and other assets, etc. which you don't want to be processed by **webpack**.
- ✓ Files in this folder are copied and pasted as they are directly into the build folder. Only files inside the `public` folder can be referenced from the HTML.

Webpack in react is a JavaScript module bundler that is commonly used with React to bundle and manage dependencies. It takes all of the individual JavaScript files and other assets in a project, such as images and CSS, and combines them into a single bundle that can be loaded by the browser.

- ✓ If you put assets in the public folder and you have to give their reference in your project, then you will have to use a special variable that is called **PUBLIC_URL**.
- ✓ A file that remains in the public folder, will be accessible by **%PUBLIC_URL%**.

For example –

```
<link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
```

- ✓ When you run the npm build command to deploy your project, create-react-app will convert %PUBLIC_URL% to the right absolute path of your application. So that it can work well if you use host/client-side routing at a non-root URL
- ✓ *favicon.ico*
- ✓ This the default react icon that always remains in the public folder. you can also put here your own project icon but the icon extension must be .ico and the icon name may be anything.
- ✓ You can remove favicon.ico when you place a new favicon for your project/website.
- ✓ When you open your app in the web browser, you will see an icon in the tab on the left side. It is the symbol of your application. So, you should not leave it.

index.html

- ✓ This is the index file that displays when the react app opens in the web browser. It contains the HTML template of the react application.
- ✓ index.html file is the root file of the react app. Everything will be rendered through it on the front end. So, Don't try to change & remove this file from the public folder.

Note – index.html must exist in the public folder and you must not delete it otherwise you will get an error.

logo192.png & logo512.png

- ✓ These are the logos of react js. It is placed just for the initial view of react app. you can remove/leave it depends on you.

manifest.json

- ✓ manifest.json provides the metadata like short_name, name & icons in the form of JSON for a react application. It may be installed on the mobile or desktop. So that you can directly open the react application with its installed favicon.

- ✓ Due to the manifest.json file, users get a notification to install react application on their mobile or desktop.
- ✓ You must not remove manifest.json but you can modify JSON values according to your project

robots.txt

- ✓ **The robot.txt file is given just for SEO purposes.** As a developer, you need not do anything with this file. This file is not related to development.

src

- ✓ **In the src folder, You can put all the js, CSS, images, components file & other assets of your projects.**

- ✓ By default, we get the following files that are necessary to understand their usages. you can create your own files according to these files for developing your projects.

App.css

- ✓ **App.css file contains a default CSS code and import into the App.js file.** It is also global, you can import another file. You can create your own CSS file like App.css but make sure that its name must start with the uppercase letter and.

- ✓ for example – **Myapp.css**

App.js

  - ✓ **App.js is a parent component file of your react app. It is imported into the index.js file to render content/HTML in the root element that remains in public/HTML.**
  - ✓ You can also create your own component file according to App.js but make sure that its extension must be .js and its name must start with an uppercase letter (recommended).
  - ✓ for an example – Myapp.js.

App.test.js

- ✓ **App.test.js gives us a testing environment**. Basically, it's written code to protect the react application to be crashed.

- ✓ We also need not modify & remove this file from the react application.

index.css

index.css file contains some default css code for index.js. You can modify/add some new CSS code according to your project design pattern.

index.js

- ✓ **index.js file is an entry point of react app**. Means that all the component renders through this file to the index.html.
- ✓ Basically, your application executes properly with the help of index.js. Even all the js files of components are imported in this file.
- ✓ for example – **As App.js file is imported with using import App from './App'** .
- ✓ If you want to add your own module then you also have to import your own Myapp.js file using the import Myapp from './Myapp' in index.js file;

logo.svg

This is the default logo of react js. You can remove it and place your project logo.

reportWebVital.js

reportWebVital.js is related to the speed of your application. You also need not to do anything with this file.

setupTest.js

In this file, @testing-library/jest-dom is imported. You need not modify and remove it from the application

.gitignore

.gitignore file is used to ignore those files that have not to be pushed to the git.

By default, dependencies, testing folders/files are defined in the .gitignore. When you push your app to the git, these folders/files will not be pushed.

package-lock.json

package-lock.json file maintains a version of installed dependencies.

package.json

All the dependencies are defined in this file. It maintains which dependencies are necessary for our application

README.md

In this file, Some instructions are written to configure and set up the react application. Even you can also write more instructions for your project that will help the developer to configure it easily.

Index.html > index.js > App.js file will be called.

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
      <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
       <!-- Notice the use of %PUBLIC_URL% in the tags above. It will be replaced with the URL
of the `public` folder during the build. Only files inside the `public` folder can be referenced
from the HTML. Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
work correctly both with client-side routing and a non-root public URL  -->

    <!-- manifest.json provides metadata used when your web app is installed on a user's
mobile device or desktop. -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>React App</title>
  </head>

 <body>
   <noscript>You need to enable JavaScript to run this app.</noscript>
   <div id="root"></div>
   <!-- This HTML file is a template. If you open it directly in the browser, you will see an
empty page. You can add webfonts, meta tags, or analytics to this file. The build step will
place the bundled scripts into the <body> tag.  -->
  </body>
</html>
```

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.

reportWebVitals();
```

React.StrictMode is a tool that highlights potential issues in a programme. It works by encapsulating a portion of your full application as a component. StrictMode does not render any visible elements in the DOM in development mode, but it enables checks and gives warnings.

React render

React renders HTML to the web page by using a function called **createRoot()** and its method **render()**.

The createRoot Function

The createRoot() function takes one argument, an HTML element.
The purpose of the function is to define the HTML element where a React component should be displayed.

The render Method

The render() method is then called to define the React component that should be rendered.

When building web applications in React, you use two packages—**react and react-dom.**
The react package holds the react source for components, state, props and all the code that is react.
The **react-dom** package as the name implies is the glue between React and the DOM. Often, you will only use it for one single thing: mounting your application to the index.html file with **ReactDOM.render()**.

Why separate them?
The reason React and ReactDOM were split into two libraries was due to the arrival of **React Native** (A react platform for mobile development).

React components are such a great way to organize UI that it has now spread to mobile to react is used in web and in mobile. react-dom is used only in web apps.

Note: Previously we had to **import React** because the JSX is converted into regular Javascript that use react's React.createElement method.
But, React has introduced a new JSX transform with the release of React 17 which automatically transforms JSX without using React.createElement. This allows us to not import React, however, **you'll need to import React to use Hooks and other exports that React provides. But if you have a simple component, you no longer need to import React**. All the JSX conversion is handled by React without you having to import or add anything.

App.js

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

function App() {
  return (
    <div className="App">
```

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

export default App;
```

JSX (JavaScript XML)

- ✓ JSX is a syntax extension for JavaScript that lets you write HTML-like markup inside a JavaScript file.
- ✓ Each React component is a JavaScript function that may contain some markup that React renders into the browser.
- ✓ React components use a syntax extension called JSX to represent that markup.
- ✓ JSX looks a lot like HTML, but it is a bit stricter and can display dynamic information.

Simple HTML code

```
<h1>LJU students</h1>
<ul>
   <li>CSE</li>
   <li>IT</li>
   <li>CE</li>
</ul>
```

And if we want to put HTML code into our component:

```
function Ex1() {
  return (
   // If you copy and paste above code as it is here, it will not work and give an error: 
“Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment 
<>...</>? “
  )
}
export default Ex1
```

As we must have to follow the rules of JSX

The Rules of JSX

1. Return a single root element

To return multiple elements from a component, wrap them with a single parent tag.
For example, you can use a <div>:

```
<div>
  <h1>LJU students</h1>
<ul>
   <li>CSE</li>
   <li>IT</li>
  <li>CE</li>
</ul>
</div>
```

If you don't want to add an extra <div> to your markup, you can write **<> and </>** instead if "<div>" tag

This empty tag is called a **Fragment**. Fragments let you group things without leaving any trace in the browser HTML tree.

2. **Close all the tags**

JSX requires tags to be explicitly closed: self-closing tags like <img> must become <img />, and wrapping tags like "<li>oranges" must be written as <li>oranges</li>.

3. **camelCase are required**

- ✓ JSX turns into JavaScript and attributes written in JSX become keys of JavaScript objects.
- ✓ In your own components, you will often want to read those attributes into variables.
- ✓ But JavaScript has limitations on variable names. For example, their names can't contain dashes or be reserved words like **class**.
- ✓ This is why, in React, many HTML and SVG attributes are written in camelCase. For example, instead of **stroke-width** you use **strokeWidth**.
- ✓ Since **class** is a reserved word, in React you write **className** instead.

```
<img  src=" "  alt=" "  className="photo" />
```

4. **Passing expression**

If you want to dynamically specify the src or alt text in img tag. You could use a value from JavaScript by replacing " and " with { and }

```
Import pic from "./img.jpg"

function Avatar() {
  const description = 'test image';
  return (
    <img
      className="pic"
      src={pic}
      alt={description}
    />
  );
}
```

```
export default Avatar;
```

Note: The **className="pic"**, which specifies an **"pic" CSS class name** that applies css to the image
src={pic} that reads the value of **pic which is imported**. That's because **curly braces** let you work with JavaScript right there in your markup!

You can only use curly braces in two ways inside JSX:

As text directly inside a JSX tag: **<h1>{name}'s To Do List</h1>** works, but <{tag}> Test's To Do List</{tag}> will not work.

As attributes immediately following the **= sign: src={pic}** will read the avatar variable, but src="{pic}" will pass the string "{pic}".

5. Using "double curlies": CSS and other objects in JSX

- ✓ In addition to strings, numbers, and other JavaScript expressions, you can even pass objects in JSX.
- ✓ You may see this with inline CSS styles in JSX.
- ✓ React does not require you to use inline styles (CSS classes work great for most cases). But when you need an inline style, you pass an object to the style attribute:

```
function Subtraction() {
   return(
      <div>
         <h1 style={{backgroundColor:'red',color:'#fff'}}>Subtraction : {7-4}</h1>
       </div>
   )
};
```

OR

```
function Subtraction() {
  var mystyle = {backgroundColor:'red',color:'#fff'};
   return(
      <div>
         <h1 style={mystyle}>Subtraction : {7-4}</h1>
       </div>
   )
};
```

JSX Comments

To write comments in React (JSX), we need to wrap **them in curly braces**.

```
Function comment() {
 Return(
    {/* this works */ }
 )
}
```

The curly braces tell the JSX parser to parse the code inside as JavaScript, and not a string.

Since the contents inside are parsed as JavaScript, this enables us to also do multi-line or single-line comments:

```
function comment(){
 return (
 <>
 {
  /*
    mult-line
    test
  */
 }
 {
    // single-line test
 }
 </>
)}
```

In the case of a single-line comment, You cannot have the ending bracket in the same line, because that will break everything.

React Components

- ✓ A react component is a JavaScript function that you can sprinkle with markup
- ✓ React lets you create components, reusable UI elements for your app.
- ✓ In a React app, every piece of UI is a component.
- ✓ React components are regular JavaScript functions except:
    - Their names always begin with a capital letter.
    - They return JSX markup.

Example1

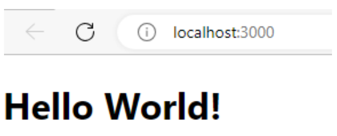
Build basic react app that display "Hello World" in browser.

1) Make changes in existing component App.js as shown below.

App.js

```
function App() {
  return (
    <div>
      <h1>Hello World!</h1>
    </div>
  );
}
export default App;
```

This App.js file is imported as a component in index.js file as we have shown above.


localhost:3000

Hello World!

This App.js file can also be considered as a component. As we are exporting it and importing in the file index.js. We can reuse this component just by importing it.

App.js is the default component file. If you don't want to make changes in this component file then create your own component file i.e. Myapp.js and make changes in it. And call your component file in index.js.

Myapp.js

```
function Myapp() {
  return (
    <div>
      <h1>Hello World!</h1>
    </div>
  );
}
export default Myapp;
```

This Myapp.js file is imported as a component in index.js file.

2) **Create file named ex1.js as a component and import this component in App.js or Myapp.js file.**

Myapp.js

```
import Ex1 from "./Ex1";
function Myapp() {
  return (
    <div>
      <Ex1/>
    </div>
  );
}
export default Myapp;
```

Ex1.js

```
function Ex1() {
   return(
       <div>
           <h1>Hello World!</h1>
       </div>
   )
}
export default Ex1;
```

This App.js or Myapp.js file is imported as a component in index.js file as we have shown above.

Note: Please try to follow last method of importing component to App.js or Myapp.js file. In this file just import component and call this component as shown in above example.

Create component using arrow function

Ex1.js

```
const Ex1=()=> {
   return(
       <div>
           <h1>Hello World!</h1>
       </div>
   )
}
export default Ex1;
```

Example

Write React code to render a component with the following data:

- **A heading in italics, blue color, and font-size 25px.**
- **An image.**
- **An ordered list of 3 fruits that start with the letter "A".**
- **The current time and current date in red color, centered.**

App1.js

```
import img1 from “img1.png”

function App1() {
const date=new Date().toLocaleDateString()
const time=new Date().toLocaleTimeString()
return (
   <div>
     <h1 style = {{color:"blue", fontStyle: "italic", fontSize: "25px"}}>Hello, Good Morning!</h1>
     <h3> List of fruits</h3>
        <ol type=”A”>
          <li>Apple</li>
         <li>Lichi</li>
         <li>Kiwi</li>
        </ol>
        <img src={img1} alt=”image”/>
        <h6>Current Date: {date}</h6>
        <h6>Current Time: {time}</h6>
   </div>
) }
export default App1;
```

Note: Styling info must be written inside two sets of curly braces {{}}. If dashed property name is used like background-color, font-size, then we have to use camel case names of properties like backgroundColor, fontSize.

****new Date().toLocaleDateString()**: The toLocaleDateString() method returns the date (not the time) of a date object as a string, using locale conventions.

****new Date().toLocaleTimeString()**: The toLocaleTimeString() method returns the time portion of a date object as a string, using locale conventions.

Map and Filter

Map

- ✓ In React, the Map method used to traverse and display a list of similar objects of a component.
- ✓ Often, we find ourselves needing to take an array and modify every element in it. Use .map() whenever you need to update data inside an array (by mapping over it!).
- ✓ A map is not the feature of React. Instead, it is the standard JavaScript function that could be called on any array.
- ✓ The map() method creates a new array by calling a provided function on every element in the calling array.

Example-1

Write React code to render a component to display all array elements in h2 tag using map function.

Map1.js

```
import React from 'react';
function Map1() {
   const arr=[1,2,3,4,5];
   return (
   <div>
      <h1>Example of mapping</h1>
      {
         arr.map((value)=>
         {
            return <h2>Array Element= {value}</h2>
         })
      }
   </div>
   )
}
export default Map1
```

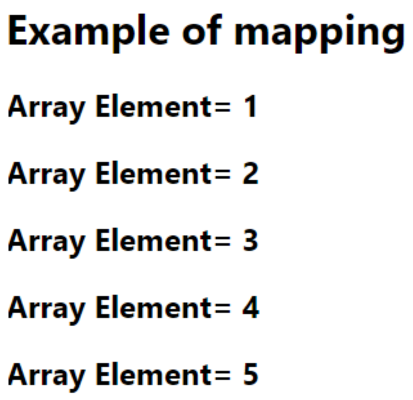
Myapp.js

```
import Map1 from "./Map1";
function Myapp() {
  return (
   <div>
```

```
    <Map1/>
   </div>
  );
  }
export default Myapp;
```

This Myapp.js file is imported as a component in index.js file

Output:



```
Example of mapping

Array Element= 1

Array Element= 2

Array Element= 3

Array Element= 4

Array Element= 5
```

Example-2

Write React code to render a component having an array of strings and convert it in Uppercase using map method.

Arraymap.js

```
const Arraymap = () => {
 const arr=["a","b","c","d","e"];
 return (
 <div>
 <h1>map function</h1>
 { arr.map((value)=> {
      return <p>array values= {value.toUpperCase()}</p>
   })
 }
 </div>
 ) }
export default Arraymap
```

Output:

```
map function

array values= A
array values= B
array values= C
array values= D
array values= E
```

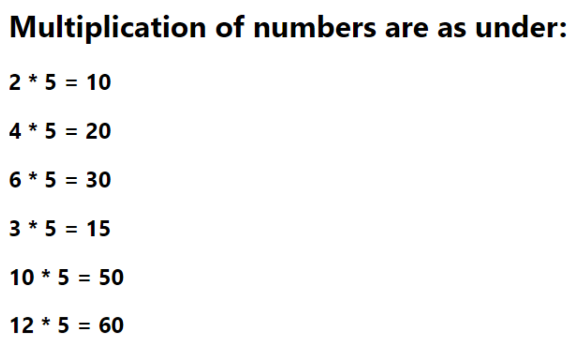
Example-3

We have an array of numbers and we want to multiply each of these numbers by 5. Write React code to render a component to display these multiplied numbers using map function.

Map2.js

```
function Map2() {
   let arr = [2, 4, 6, 3, 10, 12]
   return (
   <div>
      <h1>Multiplication of numbers are as under: </h1>
      {
         arr.map((value)=>
         {
            return <h2>{value} * 5 = {value * 5}</h2>
         })
         }
   </div>
   )
}
export default Map2
```

Output:



```
Multiplication of numbers are as under:

2 * 5 = 10

4 * 5 = 20

6 * 5 = 30

3 * 5 = 15

10 * 5 = 50

12 * 5 = 60
```

Example-4

Write React code to render a component which displays images using map function.

Map3.js

```
import React from 'react';
import img1 from "./img1.png" //import image from same folder
import img2 from "./img2.png" //import image from same folder
function Map3() {
   const images=[{id:2,pic:img1},{id:2,pic:img2}];
  return (
   <div>
      {images.map((val) => {
         return <img src={val.pic} heigth="200px" width="200px" alt="logo" />
      })}
   </div>
  )
}
export default Map3
```

Output: This will display two images.

Example: map with condition

Write React code to render a component which displays array elements which are greater than 3.

Map4.js

```
function Map4() {
   const arr=[1,2,3,4,5,3,6,4,3,1];
   return (
   <div>
      <h1>Example of mapping with condition</h1>
      {
         arr.map((value)=>
         {
            if(value>3){
            return (<h2 >Array Elements= {value}</h2>)
            }
         })
      }
   </div>
   )
}
export default Map4
```

List and Keys

- ✓ Lists are very useful when it comes to developing the UI of any website.
- ✓ Lists are mainly used for displaying menus on a website, for example, the navbar menu. In regular JavaScript, we can use arrays for creating lists.
- ✓ A “key” is a special string attribute you need to include when creating lists of elements in React.
- ✓ If **lists** do not include the **key** attribute, then it will give below warning. The warning says that each of the list items in our unordered list should have a unique key.

```
Warning: Each child in an array or iterator
should have a unique "key" prop
```

Keys are used in React to identify which items in the list are changed, updated, or deleted. In other words, we can say that keys are used to give an identity to the elements in the lists. It is recommended to use a string as a key that uniquely identifies the items in the list.

List.js

```
import React from "react";
 function List() {
   const students = [
     {id: 1, name: 'ABC'},
     {id: 2, name: 'XYZ'},
     {id: 3, name: 'PQR'}
   ];
   return(
     <ul>
     {
       students.map((student) =>
       {
         return <li key={student.id.toString()}>{student.name}</li>
       })
     }
     </ul>
   )
  }
  export default List
```

OR

List.js

```
import React from "react";
 function List() {
    const students = [
      {id: 1, name: 'ABC'},
      {id: 2, name: 'XYZ'},
      {id: 3, name: 'PQR'}
    ];
    return(
      <ul>
      {
        students.map((student,index) =>
        {
          return <li key={index}>{student.name}</li>
        })
      }
      </ul>
    )
  }
  export default List
```

Filter

- ✓ filter() loops through data, and filters out data that doesn't match the criteria that we set.
- ✓ So, It's the process of looping through an array and including or excluding elements inside that array based on a condition that you provide.
- ✓ It is also built in JavaScript function.

Write React component to skip digit “3” from an array and display all remaining digits of the array.

Filt1.js

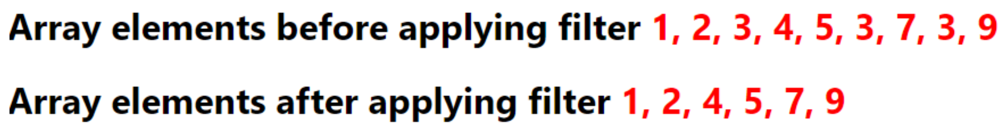
```
import React from 'react';
function Filt1() {
   const arr=[1,2,3,4,5,3,7,3,9];
   const newarr = arr.filter((num)=>
   {
      if(num===3){
         return false;
      }
      else{
         return true;
      }
   });
   var arr1 = arr.join(", "); //join each element of array by “,”
   var arr2 = newarr.join(", "); //join each element of array by “,”
   return (
   <div>
      <h1>Array elements before applying filter <span style={{color:"red"}}> {arr1} </span>
</h1>
      <h1>Array elements after applying filter <span style={{color:"red"}}> {arr2} </span>
</h1>
   </div>
   )
}
export default Filt1
```

OR

Filt1.js

```
function Filt1() {
   var arr=[1,2,3,4,5,3,7,3,9];
   return (
      <>

      <h1>Array elements before applying filter <span style={{color:"red"}}> {arr.join(", ")}
</span> </h1>
      <h1>Array elements after applying filter
      {
         arr.filter((num)=>(num!=3)).map((num)=> {return <span style={{color:"red"}}>
{num}, </span>})

      }
      </h1>
      </>
   )
}
export default Filt1
```

Output:



```
Array elements before applying filter 1, 2, 3, 4, 5, 3, 7, 3, 9

Array elements after applying filter 1, 2, 4, 5, 7, 9
```

To understand difference of output using map and filter refer below examples.

Example:

Write React code to filter out the numbers greater 6 using map/filter function.

Check difference in output using different methods.

Using Only map method

```
const ArrayMap_condition = () => {
 const arr1=[1,2,3,4,5,6,7,8,9]
 return (
 <div>
 <h1>using map/filter function</h1>
{
   arr1.map((value)=>{
   if(value <=6){
        return <h1>array values= {value}</h1>
   }
   })
}
 </div>
 )
}
export default ArrayMap_condition
```

Output using map function:

```
using map/filter function
array values= 1
array values= 2
array values= 3
array values= 4
array values= 5
array values= 6
```

Cons: The map method will still iterate over all elements, but only those that meet the condition will render something. Others will effectively render undefined.

OR

Using Only filter method

```
const ArrayMap_condition = () => {
 const arr1=[1,2,3,4,5,6,7,8,9]
 return (
 <div>
 <h1>using map/filter function</h1>
{
   arr1.filter((value)=>{
   if(value <=6){
         return <h1>array values= {value}</h1>
      }
   })
}
 </div>
 )
}
export default ArrayMap_condition
```

Output using filter function:

```
using map/filter function
123456
```

Filtering First: arr1.filter filters the array to only include values less than or equal to 6 and displayed filtered values to without applying <h1> element.(returns only values)

OR

Using map and filter methods

```
const ArrayMap_condition = () => {
 const arr1=[1,2,3,4,5,6,7,8,9]
 return (
 <div>
 <h1>using map/filter function</h1>
{
arr1.filter((value)=>value <=6).map((value)=>{ return <h1>array values= {value}</h1>})
}
 </div>
 )
}
export default ArrayMap_condition
```

Output using map and filter function:

```
using map/filter function
array values= 1
array values= 2
array values= 3
array values= 4
array values= 5
array values= 6
```

- ✓ **Filtering First**: arr1.filter((value) => value <= 6) filters the array to only include values less than or equal to 6.
- ✓ **Mapping with Keys**: **map((value)=>{ return <h1>array values= {value}</h1>})** maps the filtered values to <h1> elements.

React Props

- ✓ Props stand for "Properties." It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.
- ✓ React components use props to communicate with each other. Each component can pass some information to other components by giving them props.
- ✓ Props are similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.

Let's understand how to pass and read data using props by below example.

Step 1: Pass props to the component

Prop1.js

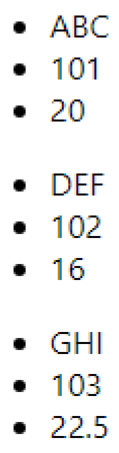
```
import Prop2 from "./ Prop2";

function Prop1 () {
var n = “ABC”;
  return (
    <div>
      <Prop2 name={n} rollnum="101" marks="20" />
      <Prop2 name="DEF" rollnum="102" marks="16" />
      <Prop2 name="GHI" rollnum="103" marks="22.5" />
    </div>
  );
}
export default Prop1;
```

Step 2: Read props inside the component

Prop2.js

```
function Prop2(props){
    return(
    <div>
        <ul>
            <li>{props.name}</li>
            <li>{props.rollnum}</li>
            <li>{props.marks}</li>
        </ul>
    </div>
    );
}
export default Prop2;
```

Output:



```
• ABC
• 101
• 20

• DEF
• 102
• 16

• GHI
• 103
• 22.5
```

Example:

Write a React code to print car's brand name and its model name which are passed as props using JSON.

Ex2.js

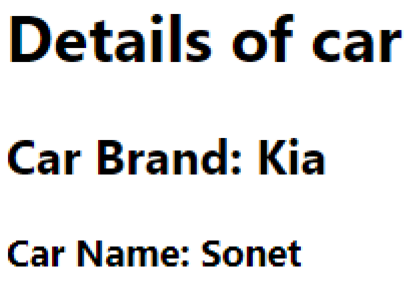
```
import Ex3 from "./Ex3";
function Ex2() {
 const carInfo = { brand: "Kia", name: "Sonet" };
 return (
   <div>
     <h1>Details of car</h1>
     <Ex3 car={ carInfo }/>
   </div>
 );
 }
export default Ex2;
```

Ex3.js

```
import React from 'react';
function Ex3(props) {
   return(
       <>
       <h2>Car Brand: { props.car.brand } </h2>
```

```
        <h3>Car Name: { props.car.name }</h3>
        </>
    );
}
export default Ex3;
```

Output:



Details of car

Car Brand: Kia

Car Name: Sonet

Example

Write a program using ReactJS in which you've to create two variable names -Student_name and University_name, these both values should be passed to another component names_Details where these values are printed using props.

Example1.js

```
import Example from "./Example";
function Example1 () {
const Details = {Student_name: "abc", University_name: "LJU"};
return (
<div>
<Example data ={ Details }/>
</div>
) }
export default Example1
```

Example.js

```
function Example(props) {
return(
<h2> My name is {props.data.Student_name}.
I am a student of { props.data.University_name } University !</h2>
) }
export default Example
```

Output:

```
My name is abc. I am a student of LJU University !
```

React Events

To handle events with React elements is very similar to handle events with DOM events. There are some syntax differences as below:
React events are named using camelCase, rather than lowercase as we used to do in HTML.

In HTML:
```
<button onclick="demofunction ()">
  LJ University
</button>
```

In React:
```
<button onClick = { demoFunction }>
  LJ University
</button>
```

Another difference is that we cannot **return false** to prevent default behavior in React. We must call **preventDefault** explicitly.
preventDefault is used to prevent the default form behavior of submitting.

In HTML
```
<form onsubmit="console.log (' You clicked submit. '); return false">
  <button type="submit">Submit</button>
</form>
```

In React
```
function Form () {
  function handleSubmit (e) {
    e.preventDefault ();
    alert (' You clicked submit.');
  }

  return (
    <form onSubmit = {handleSubmit}>
      <button type="submit">Submit</button>
    </form>
  );
}
```

1) Example of onclick event

Write react js script to display alert box with text “Welcome to LJU” by clicking on button.

Event1.js

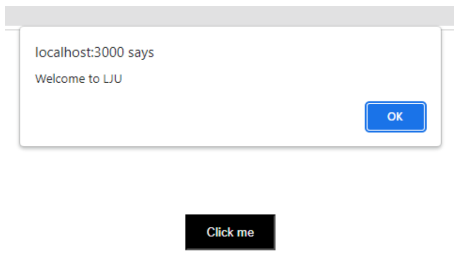
```
import React from 'react';
function Event1() {
  const mystyle = {
      color : "white",
      backgroundColor: "#000000",
      padding: "10px 20px",
      margin: "200px"
    };
  Function handleClick () {
        alert ('Welcome to LJU');
  }
  return (
    <div>
      <center>
        <button style = {mystyle} onClick={handleClick}>
          Click me
        </button>
      </center>
    </div>
  );
}
export default Event1;
```

App.js

```
import Event1 from "./Event1"
function App() {
  return (
    <div>
      <Event1/>
    </div>
```

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

Output:


localhost:3000 says

Welcome to LJU

---

2) Example of onchange event

Write react js script to display values in console while changing it in text box.

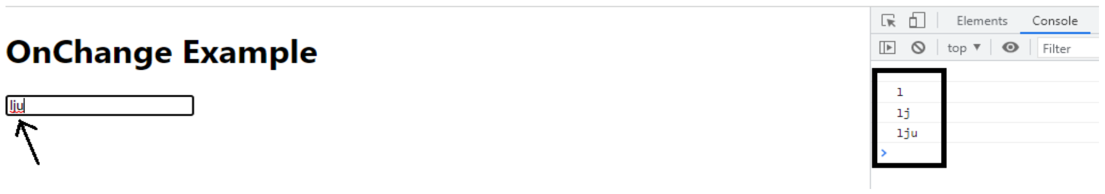
Event2.js

```
import React from 'react';
function Event2() {
 function handleChange(event) {
   console.log (event.target.value);
 }
 return (
   <input type=”text” name="firstName" onChange={handleChange} />
 );
}
export default Event2;
```

App.js

```
import Event2 from "./Event2"
function App() {
 return (
   <div>
     <Event2/>
   </div>
 );
 }
export default App;
```

Output:



```
1
lj
lju
```

event.target gives the element that triggered the event.
So, event.target.value retrieves the value of that element (an input field, in above example).

3) Example of DoubleClick event

Write react js script to display alert box with text "welcome to lju" only on double click.

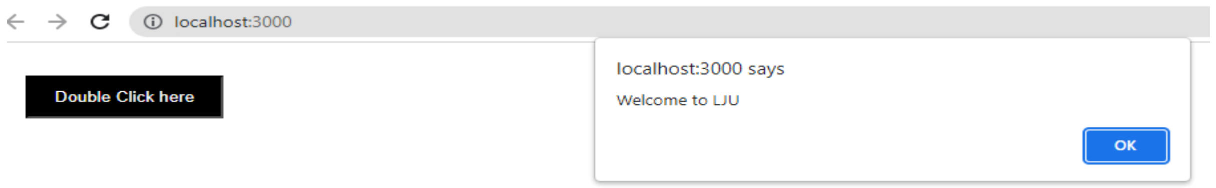
Event3.js

```
import React from 'react';
function Event3() {
  const mystyle = {
      color : "white",
      backgroundColor : "#000000",
      padding: "10px 20px",
      margin:"30px"
    };

         const doubleClickHandler = (event) => {
                  alert("Welcome to LJU");
         }
         return (
                  <>
       <button style={mystyle} onDoubleClick = {doubleClickHandler}>Double Click
here</button>
                  </>
         );
}
export default Event3;
```

App.js

```
import Event3 from "./Event3"
function App() {
 return (
  <div>
   <Event3/>
  </div>
 );
 }
export default App;
```

Output:



localhost:3000 says

Welcome to LJU

4) Example of onSubmit event

Write react js script to display alert box with text “You clicked submit.” **only on submitting form.**

Event4.js

```
function Event4() {
 function handleSubmit (e) {
  e.preventDefault ();
  alert (' You clicked submit.');
 }

 return (
  <form onSubmit = {handleSubmit}>
   <button type="submit">Submit</button>
  </form>
 );
}export default Event4;
```

App.js

```
import Event3 from "./Event4"
function App() {
 return (
  <div>
   <Event4/>
  </div>
  );
  }
export default App;
```

React Routing

Create React App doesn't include page routing. React Router is the most popular solution for page routing.

Add React Router

To add React Router in your application, run below command in the terminal from the root directory of the application:

```
npm i react-router-dom
```

Folder Structure

To create an application with multiple page routes, let's first start with the file structure. Within the **src** folder, we'll create a folder named **routing** with several files:

src\routing\:

- Home.js
- Shop.js
- Contact.js
- Nopage.js

Each file will contain a very basic React component.

✓ **BrowserRouter:** It is used to keep your UI in sync with the URL. It is the parent component that is used to store all of the other components.

✓ **Routes:** An application can have multiple <Routes>. Routes are chosen based on the best match instead of being traversed in order.

✓ **Route:** It is used to define and render component based on the specified path. It will accept components and render to define what should be rendered.

✓ **Link:** Link component is used to create links to different routes and implement navigation around the application. It works like HTML anchor tag.

Note: Do not use anchor tags instead of <Link> components because using anchor tags would not allow applications to remain Single Page Application (SPA). HTML anchor tag would trigger a page reload or page refresh when clicked.

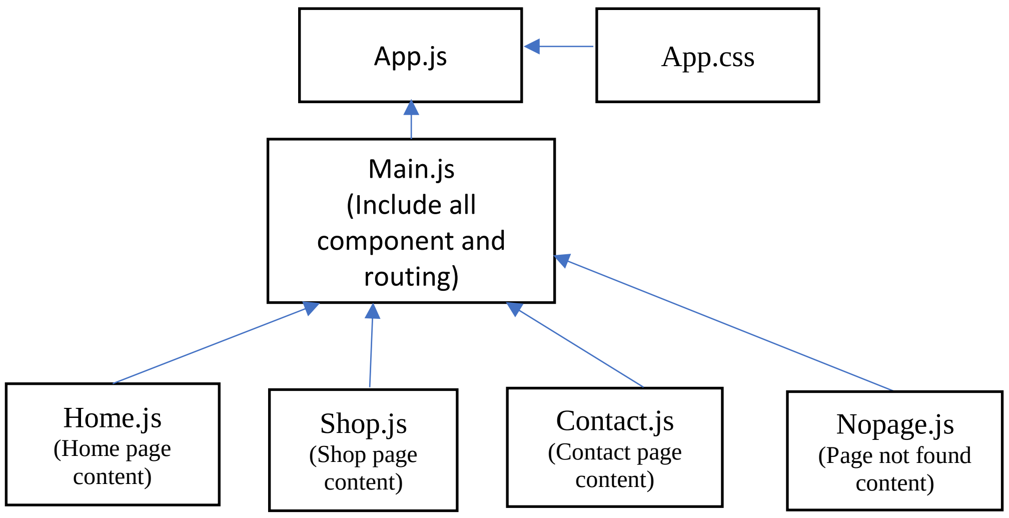
Example:

- Create react app to perform tasks as asked.

    First create files as asked below in routing folder

1. Home.js - for the home page content
2. Shop.js - for the shop page content
3. Contact.js - for the contact page content
4. Nopage.js - for the page other than mentioned links

- Create Main.js file which contains Links for Home, Shop and Product page. Also, add functionality of page routing.Finally call Main.js in App.js.


App.js
App.css
Main.js (Include all component and routing)
Home.js (Home page content)
Shop.js (Shop page content)
Contact.js (Contact page content)
Nopage.js (Page not found content)

App.js

```
import './App.css';
import Main from "./routing/Main.js";

function App() {
  return (
    <div>
      <Main/>
    </div>
  );
  }
export default App;
```

(Below all files are inside routing folder in src folder.)

routing/Main.js

```
import React from 'react';
import {BrowserRouter as Router,Route,Routes,Link} from "react-router-dom";
import Home from './Home';
import Contact from './Contact'
import Shop from './Shop'
import Nopage from './Nopage';
function Main() {
return (
<div>
   <Router>
     <div className='main-route'>
          <ul>
            <li><Link to="/">Home</Link></li>
            <li><Link to="/shop">Shop</Link></li>
            <li><Link to="/contact">Contact</Link></li>
          </ul>
      </div>
      <Routes>
         <Route path="/" element={<Home/>}/>
         <Route path="Contact" element={<Contact/>}/>
         <Route path="Shop" element={<Shop/>}/>
         <Route path="*" element={<Nopage/>}/>
      </Routes>
   </Router>

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

Setting the path to * will act as a catch all undefined URLs and display 404 error page.

routing/Home.js

```
import React from 'react'
function Home(){
return (
<div>
<h1>Home page</h1>
</div>
)
}
export default Home
```

routing/Shop.js

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

```
function Shop(){
return (
<div>
<h1>Shop page</h1>
</div>
)
}
export default Shop
```

routing/Contact.js

```
import React from 'react'
function Contact() {
return (
<div>
<h1>Contact Detail</h1>
</div>
)
}
export default Contact
```

routing/Nopage.js

```
import React from 'react'
function Nopage() {
return (
<div>
<h1>404 Page not Found</h1>

</div>
)
}
export default Nopage
```

App.css **(It is Not compulsory to add this file in this example. Added css for the reference only)**

```
.main-route ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #000;
  margin-bottom: 50px;
}
.main-route li {
  float: left;
}
.main-route li a {
  display: block;
  color: white;
  text-align: center;
  padding: 20px;
  text-decoration: none;
}
.main-route h1{ color: red; text-align: center;
```

Miscellaneous Examples

Create react component to perform the tasks as asked below.
Add array of 5 objects with properties Name and Age. Check if age is greater than 50 then display the person name of who are greater than 50 age.

M1.js

Using only map function to fulfil condition and to display values.

```
import React from 'react'
function M1()  {
const people = [
   {
      name: 'ABC',
      age: 31,
   },
   {
      name: 'XYZ',
      age: 55,
   },
   {
      name: 'PQR',
      age: 36,
   },
   {
      name: 'EFG',
      age: 69,
   },
   {
      name: 'DEF',
      age: 34,
   }
];
return (
   <ul>
   {
      people.filter((p) => (p.age > 50)).map((p)=>{return( <h3>{p.name}</h3> )})
   }
   </ul>
) }
export default M1
```

Example:2: Create react app to display students of CSE branch.

M2.js

```
import React from 'react'
function M2() {
let students = [
{ id: "001", name: "N1", Branch: "CSE" },
{ id: "002", name: "N2", Branch: "CE" },
{ id: "003", name: "N3", Branch: "CSE" },
{ id: "004", name: "N4", Branch: "CSE" },
{ id: "005", name: "N5", Branch: "IT" }
]
return (
<div>
students.filter((student) => (student.Branch === "CSE")).map((student)=>{return(
<h3>{student.name}</h3> )})
</div>
)
}
export default M2
```

Example3: Create react app to pass product image, name and price as properties from one component to another component. Add an array of objects with pic, name and price properties of 2 products. Display Image name and price of the products in browser using map method.

P.js (Pass the data)

```
import P1 from "./P1";
import img1 from "./img1.png"
import img2 from "./img2.png"
function P(){
   const prod=[
   {
      pic:img1,
      name:"Product1",
      price:3000
   },
   {
      pic:img2,
      name:"Product2",
      price:3000
```

```
    }
]
return(
    <div>
        <P1 info={prod}/>
    </div>
)
}
export default P
```

P1.js (Read the data)

```
import React from "react";
function P1(prop){
    return(
        <>
        {
        prop.info.map((pr)=>{
            return (
            <div>
            <img src={pr.pic} alt="No Image" />
            <h2>{pr.name}</h2>
            <h3>{pr.price}</h3>
            </div>
            )
        })
    }
    </>
    )
}
export default P1
```

Import P component in App.js file

Example4:

Create a component to perform the tasks as described below:

1. Add a text field and a submit button.

- While changing the value in the text field, display it below the form.

- Display this text field value in an alert box upon submitting it.

2. Add a button to perform click and double-click event tasks.

- On click event, display message in h3 tag "You clicked once".

- On double-click event, display message in h3 tag "You clicked twice".

- Display these message should be displayed below the button.

```
function Map1() {
   const arr=[1,2,3,4,5];
   function handleSubmit (e) {
      e.preventDefault ();
      alert (document.getElementById('uname').value);
     }
     function handleclick(){
      document.getElementById('test1').innerHTML = “You clicked once”
   }
   function handledoubleclick(){
      document.getElementById('test1').innerHTML = “You clicked twice”
   }
   function handleChange(event) {
      document.getElementById('test').innerHTML =event.target.value;
   }
   return (
   <div>
      <form onSubmit = {handleSubmit}>
         <input type="text" id="uname" onChange={handleChange}></input>
         <input type="submit"/>
      </form>

     <h1 id="test">On change event</h1>

     <button style={{backgroundColor:'black',padding:"20px",color:"white"}}
onClick={handleclick} onDoubleClick={handledoubleclick}>Click</button>
     <h1 id="test1">Click/DoubleClick event</h1>
   </div>
   )
}

export default Map1
```

Hooks in react js

Hooks were added to React in version 16.8.

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

There are 3 rules for hooks:

- Hooks can only be called inside React function components (Only call hooks from React functions" — don't call hooks from plain JavaScript functions so that stateful logic stays with the component.)
- Hooks can only be called at the top level of a component and Hooks cannot be conditional (don't call hooks from inside loops, conditions, or nested statements so that the hooks are called in the same order each render.)

You must import Hooks from react.
Suppose, we are using the **useState** Hook to keep track of the application state. Then we have to import it as below.

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

1. useState

The React useState Hook allows us to track state in a function component.
State generally refers to data or properties that need to be tracking in an application.
Initialize useState
We initialize our state by calling useState in our function component.
useState accepts an initial state and returns two values:
- o The current state.
- o A function that updates the state.

Syntax:

```
Const [current state, function to update state] =useState (initial state)
```

To import useState

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

Example:

Write a program to build React app having a button which increase count by 1 while clicking it.

US1.js

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

function US1() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  function handleCount() { setCount(count+1) }

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={handleCount}>
        Click me
      </button>
    </div>
  );
}
export default US1
```

Example:

Create a program to build React app having buttons to increment and decrement the number by clicking that respective button. Also, increment of the number should be performed only if number is less than 10 and decrement of the number should be performed if number is greater than 0.

US.js

```
import { useState } from "react";
function US () {
    const [num,setnum]=useState(0)
    function increment(){
      if(num<10){
          setnum(num+1);
      }else{
          return false;
      }
  }
  function decrement(){
      if(num > 0){
          setnum(num-1);
      }else{
          return false;
      }
      }
  return (
      <div>
          <button onClick={increment}>Increment</button>
          <button onClick={decrement}>Decrement</button>
          <h1> {num} </h1>
      </div>
  )
}
export default US
```

Example:

Write a program to build React app to perform the tasks as asked below.

- **Add three buttons "Change Text", "Change Color", "Hide/Show".**
- **Add heading "LJ University" in red color(initial) and also add "React Js Hooks" text in h2 tag.**
- **By clicking on "Change text" button text should be changed to "Welcome students" and vice versa.**
- **By clicking on "Change Color" button change color of text to "blue" and vice versa. This color change should be performed while double clicking on the button.**

- **Initially button text should be “Hide”. While clicking on it the button text should be changed to “Show” and text “React Js Hooks” will not be shown.**

US1.js

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

function US1(){
  //useState to Change text
  const [name,setName] = useState("LJ University");
  //useState to Change Color
  const [textColor,setcolor] = useState("Red");
  //useState to Show Hide text
  const [hideText,setHide]=useState("React Js Hooks");
  const[buttontext,setButtontext]= useState("Hide")

  // Function to show and hide text and also button text
  function showhide() {
    if(buttontext==="Hide")
    {
      setButtontext("Show");
      setHide("")
    }
    else{
      setButtontext("Hide");
      setHide("React Js Hooks")
    }
  };
  // function to change text value
  function changeName(){
    if(name === "LJ University"){
    setName("Welcome Students")
    }else{
      setName("LJ University")
    }
  }
  // Function to change color of the text
  function changeColor(){
    if(textColor === 'red'){
    setcolor("blue")
    }else{
      setcolor("red")
    }
  }
return(
  <div>
    <button onClick={changeName}>Change Text</button>
    <button onDoubleClick={changeColor}>Change Color</button>
    <button onClick = {showhide}> {buttontext}</button>

    <h1 style={{color:textColor}}>{name}</h1>
    <h2>{hideText}</h2>
  </div>
)}
export default US1
```

Example:

Write a program to build React app having a button which changes image by clicking it.

US2.js

```
import { useState } from 'react';
import img1 from "./img1.png";
import img2 from "./img2.png";
function US2 () {
  const [myImage,setImage]=useState(img1);
  function changeImage () {
    if(myImage === img1){
      setImage(img2)
    }else{
         setImage(img1)
      }
   }
   return (
      <div>
         <img src={myImage} heigth="200px" width="200px" alt="logo" />
         <button onClick={changeImage}>Change Image</button>
      </div>
      ) }
export default US2
```

Example:

Write React component having a button and image. By clicking on button, image changes randomly from a given array of images.

US3.js

```
import {useState} from "react";
import img1 from "./img1.jpg"
import img2 from "./img2.jpg"
import img3 from "./img3.png"
import img4 from "./img4.jpg"
import img5 from "./img5.jpg"

function US3()
{
   const arr = [img1,img2,img3,img4,img5]
   const [myimage,setimage] = useState(arr[0]);
   function changeImage {
      const randomIndex = Math.floor(Math.random() * arr.length);
      setimage(arr[randomIndex]);
     };

     return (
      <div className="App">
```

```
        <header className="App-header">
          <h1>Random Image Generator</h1>
          <img src={myimage} alt="Random" width="500" height="500"/>
          <button onClick={changeImage}>Change Image</button>
        </header>
      </div>
    );

}
export default US3
```

Example:

Create a React component that manages multiple form input fields using a single state object and displays the values in real-time

US4.js

```
import { useState } from 'react'
function US4() {
   const[data,setdata]=useState({});

   function handleChange(e) {
      const { name, value } = e.target;
      setdata({...data,[name]: value});
    };
  return (
   <div>
     <div><input    type="text"    name="firstName"    onChange={handleChange}
placeholder='First Name'/></div>
      <div><input    type="text"    name="lastName"    onChange={handleChange}
placeholder='Last Name'/></div>
      <h1>First Name: {data.firstName} Lastname: {data.lastName}</h1>
   </div>
  ) }
export default US4
```

Example:

Write a react component for todo list.

- **Add 1 input field and button and by clicking on button display entered task on the same page.**
- **Also, add delete button with each added task to delete the task.**

Todo.js

```
import {useState} from 'react'

function Todo() {
   const[Task, setTask]= useState("");
   const[Todolist, setTodoList]=useState([]);
   function handleChange(event) {
      setTask(event.target.value);
   }
   // To add task
   function addTask(event) {
      setTodoList([...Todolist,Task]);
   };
   //To add delete functionality
   function deleteTask(taskName){
      setTodoList(
         Todolist.filter((task)=>{
            if (task!==taskName){
               return true;
            } else{
               return false;
            }
         })
      )
   }
   return (
   <div>
   <h1> Enter Task </h1>
   <input  onChange={handleChange}/>
     <button  onClick={addTask}> Add Task </button>

     {Todolist.map((task)=>{
       return(
         <div>
           <h1> {task}</h1>
           <button onClick={() => deleteTask(task)}>Delete</button>
         </div>
       );
     })}
   </div>
  );
}
export default Todo
```

Create react app which takes user defined inputs number 1 and number 2 and perform addition, subtraction, multiplication, division of the numbers. (Use useState hook)

US5.js

```
      import  { useState } from 'react'
      function US5() {
        const[data,setdata]=useState({});
        const[result,setresult]=useState();

        const handleChange = (e) => {
          const { name, value } = e.target;
          setdata({...data,[name]: value});
         };

         function addition(){
          setresult(parseInt(data.num1) + parseInt(data.num2))
         }
         function sub(){
                setresult(parseInt(data.num1) - parseInt(data.num2))
              }
              function mult(){
                setresult(parseInt(data.num1) * parseInt(data.num2))
              }
              function division(){
                setresult(parseInt(data.num1) / parseInt(data.num2))
              }

          return (
            <div>
               <div><input type="number" name="num1"  onChange={handleChange}
placeholder='First Name'/></div>
                <div><input type="number" name="num2"  onChange={handleChange}
placeholder='Last Name'/></div>
                <button onClick={addition}>addition</button>
                <button onClick={sub}>Subtraction</button>
                <button onClick={mult}>Multiplication</button>
                <button onClick={division}>Division</button>
                <h1> {result}</h1>
            </div>
          ) }
        export default US5
```

2. useReducer

The useReducer Hook is the better alternative to the useState hook and is generally more preferred over the useState hook when you have complex state-building logic or when the next state value depends upon its previous value or when the components are needed to be optimized.

You can add a reducer to your component using the useReducer hook. Import the useReducer method from the library like this:

```
import { useReducer } from 'react'
```

The useReducer method gives you a state variable and a dispatch method to make state changes. You can define state in the following way:

```
const [state, dispatch] = useReducer(reducerFunction, initialValue)
```

The reducer function contains your state logic. You can choose which state logic to call using the dispatch function. The state can also have some initial value similar to the useState hook.

Let's understand using example

The useReducer(reducer, initialState) hook accepts 2 arguments: the reducer function and the initial state. The hook then returns an array of 2 items: the current state and the dispatch function.

UR.js

```
import React, { useReducer } from 'react';
const initialstate = 0;
function reducer(state,action){
  if(action.type==='increment'){
    return state+1;
  }
}
function UR() {
 const [state, dispatch] = useReducer(reducer, initialstate);
 return (
  <button onClick={()=> dispatch({type:"increment"})}>
    Click me ({state})
  </button>
 );
}
export default UR
```

A. Initial state

The initial state is the value the state is initialized with. Here initialized with 0.

B. Reducer function

The reducer is a pure function that accepts 2 parameters: **the current state and an action object.** Depending on the action object, the reducer function must update the state in an immutable manner, and return the new state.

C. Action object

An action object is an object that describes how to update the state.
Typically, the action object has a property **type** — a string describing what kind of state update the reducer must do.

If the action object must carry some useful information to be used by the reducer, then you can add additional properties to the action object. **Here we have defined type "increment".**

D. Dispatch function

The dispatch is a special function that dispatches an action object.
The dispatch function is created for you by the useReducer() hook:

```
const [state, dispatch] = useReducer(reducer, initialState);
```

Whenever you want to update the state (usually from an event handler or after completing a fetch request), you simply call the dispatch function with the appropriate action object: dispatch(actionObject).
Like as below in above example

```
   <button onClick={()=> dispatch({type:"increment"})}>
```

In simpler terms, dispatching means a request to update the state.

What Is a Reducer and Why do You Need It?

Let's take an example of a To-Do app. This app involves adding, deleting, and updating items in the todo list. The update operation itself may involve updating the item or marking it as complete.

When you implement a todo list, you'll have a state variable todoList and make state updates to perform each operation. However, these state updates may appear at different places, sometimes not even inside the component.
To make your code more readable, you can move all your state updates into a single function that can exist outside your component. While performing the required operations, your component just has to call a single method and select the operation it wants to perform.

The function which contains all your state updates is called the reducer. This is because you are reducing the state logic into a separate function. The method you call to perform the operations is the dispatch method.

Example:

Write react component to increase value by 5 while clicking on button. Initialize value with 20. Use useReducer hook to perform the task.

UR1.js

```
import React, { useReducer } from 'react'
function reducer(state,action){
   return state+action;
}
function UR1 () {
   const [state,dispatch]=useReducer(reducer ,20);
  return (
   <div align="center">
      <h1 align="center">{state}</h1>
      <button onClick={()=>dispatch(5)}>Add</button>
   </div>
  )
}
export default UR1
```

Example:

Create react js app to increase value by 1 while clicking on button “Increment” and decrease value by 1 while clicking on button “Decrement”. Initialize value with 0. Use useReducer hook to perform the task.

Usereducer.js

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

const initialState=0;
function reducer(state,action){
   if(action.type==='increment'){
      return state+1;
   }
   if(action.type==='decrement'){
      return state-1;
   }
}
function Usereducer(){
   const[state,dispatch] = useReducer(reducer,initialState);
   return(
      <>
         <h1>{state}</h1>
         <button onClick={()=> dispatch({type:"increment"})}> Increment </button>
         <button onClick={()=>dispatch({type:"decrement"})}> Decrement </button>
      </>
   )
}
export default Usereducer
```

3. useContext

Context provides a way to pass data or state through the component tree without having to pass props down manually through each nested component. It is designed to share data that can be considered as global data for a tree of React components.

How to use the context

Using the context in React requires 3 simple steps:

- creating the context,
- providing the context,
- consuming the context.

A. Creating the context

The built-in function createContext(default) creates a context instance:

```
import { createContext } from 'react';
const Fname = createContext('Default Value'); //default value is optional
```

The function accepts one optional argument: the default value.

B. Providing the context

Context.Provider component available on the context instance is used to provide the context to its child components, no matter how deep they are.

To set the value of context use the **value prop**

```
< Fname.Provider value="Test" />
```

Main.js

```
import React,{ createContext } from 'react';
import Comp from './Comp'
const Fname = createContext();
function Main() {
 return (
  < Fname.Provider value='ABC'>
   < Comp /> // component in which consuming the context
  </ Fname.Provider>
 );
}
export default Main
export {Fname}
```

Again, what's important here is that all the components that'd like later to consume the context have to be wrapped inside the provider component.
If you want to change the context value, simply update the value prop.

C. Consuming the context

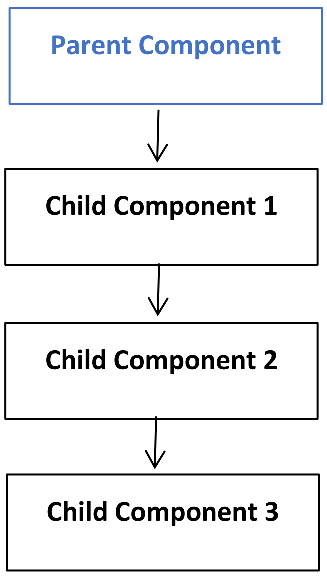
Consuming the context can be performed by the useContext(Context) React hook:

Comp.js

```
Import React,{ useContext } from 'react';
import { Fname } from './Main;

function Comp () {
  const value = useContext(Fname);
  return <span>{value}</span>;
}
Export default Comp
```

The hook returns the value of the context: value = useContext(Context). The hook also makes sure to re-render the component when the context value changes.

Suppose, we have one parent component and we want to access its data in child component 3. Then, we can use createContext for creating a context and usecontext to use the context.



Example:

Write a reactJS program to perform the tasks as asked below.

- **Create one main file (parent file) name PC.js and other 2 component files C1.js and C2.js**
- **Pass First name and Last name from PC.js file to C2.js file. And display Welcome ABC XYZ (suppose firstname is ABC and Last name is XYZ) in browser.**

PC.js

```
import React, { createContext } from "react"
import C1 from "./C1"
// To create context for firstname and lastname
const Fname = createContext();
const Lname = createContext();

function PC(){
   return (
   <>
      <Fname.Provider value="ABC">  // to provide the value which should be passed
         <Lname.Provider value="XYZ">
            <C1/>
         </Lname.Provider>
      </Fname.Provider>
   </>
   )
}
export default PC
export {Fname,Lname} //Each component must have one default export, so the context should be exported using {} as object.
```

C1.js

```
import React from "react"
import C2 from "./C2"
function C1(){
   return ( <C2/>)
}
```

```
export default C1
```

C2.js

```
import React, { useContext } from "react"
import { Fname,Lname } from "./PC"
function C2(){
   const fn = useContext(Fname)
   const ln = useContext(Lname)
   return (
   <h1>Welcome {fn} {ln}</h1>
   )
}
export default C2
```

App.js <- PC.js <- C1.js <- C2.js

So, First Name and Last Name are defined in PC.js component and are used directly in C2.js component without passing data to all other hierarchical components.

Example:

Write a reactJS program to perform the tasks as asked below.

- **Create one main file (parent file) name Comp.js and other 3 component files Comp1.js, Comp2.js, Comp3.js.**
- **Pass Number1 and Number 2 from Comp.js file to Comp3.js file. Calculate multiplication of the numbers using useContext.**

Comp.js

```
import React, { createContext } from "react"
import Comp1 from "./Comp1"
const Num1 = createContext();
const Num2 = createContext();
function Comp(){
   return (
   <>
      <Num1.Provider value="20">
```

```
          <Num2.Provider value="5">
             <Comp1/>
          </Num2.Provider>
       </Num1.Provider>
    </>
    )
}
export default Comp
export {Num1,Num2}
```

Comp1.js

```
import React from "react"
import Comp2 from "./Comp2"

function Comp1(){
   return ( <Comp2/> )
}
export default Comp1
```

Comp2.js

```
import React from "react"
import Comp3 from "./Comp3"

function Comp2(){
   return (
         <Comp3/>
   )
}
export default Comp2
```

Comp3.js

```
import React, { useContext } from "react"
import { Num1,Num2 } from "./Comp"

function Comp3(){
   const num1 = useContext(Num1)
   const num2 = useContext(Num2)
```

```
  return (
  <h1>Multiplication of numbers in component-3: {num1 * num2}</h1>
  )
}

export default Comp3
```

App.js <- Comp.js <- Comp1.js <- Comp2.js <- Comp3.js

So, Number 1 and number 2 are defined in Comp.js component and are used directly in Comp3.js component without passing data to all other hierarchical components.

Example:

Use multiple contexts in a React application by creating and consuming them across different components.
Comp1.js: Creates a context for CSS styling and provides it to Comp2.
Comp2.js: Creates a context for a string value ("Students") and provides it to Comp3.
Comp3.js: Consumes both contexts and displays a message with the provided styles and string.

Comp1.js

```
import { createContext } from "react"
import Comp2 from "./Comp2"
const CC = createContext();
const mycss={backgroundColor:'yellow',color:'red',fontSize:"45px"}
function Comp(){
  return (
  <>
    <CC.Provider value={mycss}>
        <Comp2/>
    </CC.Provider>
  </>
  )
}
export default Comp
export {CC}
```

Comp2.js

```
import { createContext } from "react"
import Comp3 from "./Comp3"
const CC1 = createContext();


function Comp(){
  return (
    <CC1.Provider value="Students">
    <Comp3/>
    </CC1.Provider>
   )
}
export default Comp
export {CC1}
```

Comp3.js

```
import { useContext } from "react"
import { CC } from "./Comp1"
import { CC1 } from "./Comp2"
function Comp3(){
  const mycss = useContext(CC)
  const data = useContext(CC1)
  return (
  <h1 style={mycss}>Welcome to useContext tutorial {data}</h1>
  )
}
export default Comp3
```

4. useEffect

The useEffect Hook allows you to perform side effects in your components.
Some examples of side effects are: fetching data, directly updating the DOM, and timers.
useEffect accepts two arguments. The second argument is optional.

```
useEffect(<function>, <dependency>)
```

To import useEffect

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

Below is the example to understand concept of empty array and array with value of useEffect

Write react js app to perform the tasks as asked below.

- **Add two buttons and increment count by one with each click.**
- **Display alert as an effect on specified conditions.**
  - **Effect will be triggered only when page rendered for the 1st time. (empty array)**
  - **Effect will be triggered every time the button A is clicked. (array with value)**
  - **When the Page is Rendered for the First Time and On Every Update/Event Triggered**

UE1.js

```
import {useState,useEffect } from 'react'
function UE1() {
   const[count,setcount]=useState(0);
   const[calculation,setcal]=useState(0);
// when the page is rendered for first time  and also when Button A(count) clicked.
    useEffect(()=>
    {
      alert("Clicked")
    },[count]);
// only once when the page is rendered
```

```
  useEffect(()=>
    {
      /alert("Clicked")
    },[]);
// when the page is rendered for first time and also on every update/event
triggered
     useEffect(()=>
    {
      alert("Clicked")
    });

    const changeCount=()=>{
      setcount(count+1);
    }
    const changeCalc=()=>{
      setcal(calculation+1);
    }
  return (
   <div>
      <button onClick={changeCount}>Button A {count}</button><br/>
      <button onClick={changeCalc}>Button B {calculation}</button>
   </div>
  ) }
export default UE1
```

Explanation of useEffect Usage

When the Page is Rendered for the First Time and When Button A (Count) is Clicked:

```
useEffect(() => {
   alert("Clicked");
}, [count]);
```

This useEffect runs whenever the count state changes, including the initial render. So, every time Button A is clicked and count is updated, the effect runs, showing the alert "Clicked".

When the Page is Rendered for the First Time Only:

```
useEffect(() => {
    alert("Clicked");
}, []);
```

This useEffect with an empty dependency array runs only once when the component mounts. It won't run again unless the component is unmounted and remounted.

When the Page is Rendered for the First Time and On Every Update/Event Triggered:

```
useEffect(() => {
    alert("Clicked");
});
```

This useEffect runs on every render, including the initial render and any subsequent updates. This is because there is no dependency array, so it runs every time the component renders.

If your alert is showing twice initially, it is likely because of React's Strict Mode. In development mode, React's Strict Mode intentionally invokes certain lifecycle methods (like useEffect) twice to help identify potential issues.

Explanation

React Strict Mode is a tool for highlighting potential problems in an application. It does so by intentionally invoking certain methods, such as component mounts and updates, twice. This behavior is designed to help developers find bugs related to side effects that might be dependent on certain lifecycle methods.

How to Check if Strict Mode is the Cause

You can check if this is the issue by looking at your index.js file where the React application is initialized. If you see React.StrictMode wrapping your application, that's the reason for the double invocation.

Write a ReactJS script to create a digital clock running continuously. (useEffect)
UE2.js

```
import { useState, useEffect } from 'react';
function UE2() {
  const [date, setDate] = useState(new Date());
  useEffect(() => {
    setInterval(() => {
      setDate(new Date());
    }, 1000)
  }, [])

  return (
    <h1>
      Time using Localtimestring - {date.toLocaleTimeString()}<br/>
      Hour-{date.getHours()}:Min-{date.getMinutes()}:Sec-{date.getSeconds()}
    </h1>
  );
}
export default UE2;
```

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.
- ✓ In React, form data is usually handled by the components. When the data is handled by the components, all the data is stored in the component state.
- ✓ You can control changes by adding event handlers in the onChange attribute and that event handler will be used to update the state of the variable.

Adding Forms in React You add a form with React like any other element:

```
function MyForm()
{ return(
<form>
<label>Enter your name:
<input type="text" /> </label>
</form>
) }
```

Handling Forms Handling forms is about how you handle the data when it changes value or gets submitted.

In React, form data is usually handled by the components. When the data is handled by the components, all the data is stored in the component state.
You can control changes by adding event handlers in the onChange attribute.
We can use the useState Hook to keep track of each inputs.

```
import { useState } from 'react';
function MyForm() {
const [name, setName] = useState("");
return (
<form>
<label>Enter your name:
<input type="text" value={name} onChange={(e) => setName(e.target.value)} />
</label>
</form>
)
}
```

Submitting Forms

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

```
import { useState } from 'react';
function MyForm() {
const [name, setName] = useState("");
function handleSubmit (event)  {
        event.preventDefault();
        alert(`The name you entered was: ${name}`)
}
return (
<form onSubmit={handleSubmit}>
<label>Enter your name:
<input type="text" value={name} onChange={(e) => setName(e.target.value)} />
</label>
<input type="submit" />
</form>
)}
```

Form Fields

❖ **Textarea**

The textarea element in React is slightly different from ordinary HTML. In React the value of a textarea is placed in a value attribute. We'll use the useState Hook to mange the value of the textarea:

```
import { useState } from 'react';
function MyForm() {
const [txtarea, setTextarea] = useState(
"The content of a textarea goes in the value attribute"
);
Function handleChange(event) {
    setTextarea(event.target.value)
}
return (
<form>
<textarea value={txtarea} onChange={handleChange} />
</form>
)
}
```

❖ **Select**

A drop down list or a select box, in React is also a bit different from HTML.
In React, the selected value is defined with a value attribute on the select tag:

```
function MyForm() {
const [myCar, setMyCar] = useState("Volvo");
function handleChange(event) {
setMyCar(event.target.value)
}
return (
<form>
<select value={myCar} onChange={handleChange}>
<option value="Ford">Ford</option>
<option value="Volvo">Volvo</option>
<option value="Fiat">Fiat</option>
</select>
</form>
)
}
```

❖ **Radio Button**

Example:- Create a React Form to select any of pizza size using radio button.

```
import React from 'react';
import { useState } from 'react';
function Myform() {
    const [s, setSize] = useState('Medium');
    function onOptionChange(e)  {
     setSize(e.target.value);
};
return (
<div>
<h3>Select Pizza Size</h3>
<form>
<input type='radio' name='ps' value='Regular' checked={s === 'Regular'}
onChange={onOptionChange} />
<label>Regular</label>

<input type='radio' name='ps' value='Medium' checked={s === 'Medium'}
onChange={onOptionChange} />
<label>Medium</label>

<input type='radio' name='ps' value='Large' checked={s === 'Large'}
onChange={onOptionChange} />
<label>Large</label>
</form>
<p>
  Selected size <strong>{s}</strong>
</p>
</div>
);}
export default Myform
```

Create react app which contains form with following fields.

- **First Name(Input type text)**
- **Email(Input type email)**
- **Password(Input type password)**
- **Confirm Password(Input type password)**
- **Message (Textarea)**
- **Gender(Radio Button)**
- **City (Dropdown)**

Display submitted values in alert box. (Using useState Hook)

Form1.js

```
import { useState } from 'react'

function Form1() {
   const[formdata,setformdata]=useState({});
   function handlechange(event) {
      const {name,value} = event.target;
      setformdata({...formdata, [name]: value})
   }
   function handlesubmit(e){
      e.preventDefault();
      alert("Your form has been submitted.\nName: " + formdata.fname +
"\nEmail: " + formdata.eid + "\nCity: "+ formdata.city +"\nGender:
"+formdata.gender)
   }
   return (
   <div>
      <form onSubmit={handlesubmit}>

         <label>First Name:</label>
         <input type="text" name="fname" onChange={handlechange} /><br/>

         <label>Email Id:</label>
         <input type="email" name="eid" onChange={handlechange} /><br/>
```

```
          <label>Password:</label>
          <input type="password" name="pass" onChange={handlechange}
required/><br/>

          <label>Confirm Password:</label>
          <input type="password" name="cpass" onChange={handlechange} /><br/>

          <label>Message : </label>
          <textarea name="msg" onChange={handlechange} /><br/>

          <select onChange={handlechange} name='city'>
             <option value="Ahmedabad">Ahmedabad</option>
             <option value="Rajkot">Rajkot</option>
          </select>

          <input type="radio" name="gender" value="Male"
onChange={handlechange} />Male
          <input type="radio" name="gender" value="Female"
onChange={handlechange}/>Female

       <button type="submit">Submit</button> <br/>
       </form>
    </div>
  )
}
export default Form1
```

Create react app which contains form with following fields.

- **Email(Input type email)**
- **Password(Input type password)**
- **Confirm Password(Input type password)**
- **Add validation using regex to validate email id and password (Must contain at least 8 characters and must contain at least 1 uppercase, 1 lowercase and 1 digit).**
- **Also values of password and confirm password must be same.**

Display email in alert box. (Using useState Hook)

Form2.js

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

function Form2() {
   const[formdata,setformdata]=useState({});
   function handlechange(event){
      const {name,value} = event.target;
      setformdata({...formdata, [name]: value})
   }
   function handlesubmit(e)
   {
      e.preventDefault();
      const emailregex=/^([a-zA-Z0-9._-]+@[a-zA-Z]+\.[a-zA-Z]{2,4}$)/;
      const passregex=/(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z]).{8,}/;
      if(!emailregex.test(formdata.eid))
      {
         alert("Please enter a valid Email ID");
      }
      else if(!passregex.test(formdata.pass))
      {
         alert("Password must contain atleast 8 characters ( Atleast 1 digit, 1
lowercase & 1 uppercase alphabets )");
      }
      else if(formdata.pass !== formdata.cpass){
         alert("password and confirm password must be same");
      }
      else{
        alert("Your form has been submitted.\nEmail: " + formdata.eid )
      }
    }
    return (
    <div>
      <form onSubmit={handlesubmit}>

        <label>Email Id:</label>
        <input type="email" name="eid" onChange={handlechange} /><br/>

        <label>Password:</label>
        <input type="password" name="pass" onChange={handlechange}/><br/>

        <label>Confirm Password:</label>
        <input type="password" name="cpass" onChange={handlechange} /><br/>

      <button type="submit">Submit</button> <br/>
      </form>

    </div>
  )
}
export default Form2
```

Example:
Create react app which contains form with fields Name, Email Id, Password and Confirm Password. When the form submitted the values of password and confirm password fields must be same else it will give an error message in alert box. If form submitted successfully then display entered name and email id in alert box.

Form2.js

```
import React, { useState } from 'react'
function Form2(){
  const[formdata,setformdata]=useState({});
  const handlechange = (event) => {
    const name = event.target.name;
    const value = event.target.value;
    setformdata({...formdata, [name]: value})
  }
  const handlesubmit=(e)=>
  {
    e.preventDefault();
    if(formdata.pass !== formdata.cpass){
      alert("Values of Password and Confirm password must be same")
    }else{
      alert("Welcome "+formdata.fname+"\n Your Email id is: "+ formdata.eid)
    }
  }
  return (
  <div>
    <form className="form-data" onSubmit={handlesubmit}>

      <label>Name:</label>
      <input type="text" name="fname" onChange={handlechange} /><br/>

      <label>Email Id:</label>
      <input type="email" name="eid" onChange={handlechange}
required/><br/>

      <label>Password :</label>
      <input type="password"  name="pass" onChange={handlechange}
required/><br/>

      <label>Confirm Password :</label>
      <input type="password"  name="cpass" onChange={handlechange} /><br/>

    <button type="submit">Submit</button> <br/>
    </form>
```

```

   </div>
  )
}
export default Form2
```

Axios

What is Axios?

- It is a library which is used to make requests to an API, return data from the API, and then do things with that data in our React application.
- Axios is a very popular HTTP client, which allows us to make HTTP requests from the browser.

Syntax:

```
const getPostsData = () => {
axios
.get("API URL")
.then(data => console.log(data.data))
.catch(error => console.log(error));
};
getPostsData();
```

Installing Axios

• In order to use Axios with React, we need to install Axios.
It does not come as a native JavaScript API, so that's why we have to manually import into our project.

• Open up a new terminal window, move to your project’s root directory, and run any of the following commands to add Axios to your project.

```
npm install axios
```

To perform this request when the component mounts, we use ‘useEffect’ hook. This involves

- importing Axios (**import axios from "axios"**;)
- using the .get() method to make a GET request to your endpoint,
- using a **.then()** callback to get back all of the response data.

What if there's an error while making a request? For example, you might pass along the wrong data, make a request to the wrong endpoint, or have a network error. In this case, instead of executing the .then() callback, Axios will throw an error and run the **.catch()** callback function. In this function, we are taking the error data and putting it in state to alert our user about the error. So if we have an error, we will display that error message.

Program: Create a react app to display images by requesting API using Axios.

```
import React,{ useState,useEffect } from 'react'
import axios from "axios";
const Randomimage = () =>
{
   const[myimg,setimg]=useState("");

   useEffect(() => {
      setInterval(() => {
         axios
         .get('https://dog.ceo/api/breeds/image/random')
         .then((response)=>{console.log(response.data);setimg(response.data);})
         .catch((error)=>{console.error(error);})
      }, 2000)
   },[])

   return(
      <div>
         <img src={myimg.message} alt='image' height={300} width={300}/>
      </div>
   )
}
export default Randomimage
```

- useEffect: This hook runs side effects in functional components.
- The empty dependency array [] means the effect runs only once after the initial render.
- setInterval: Sets up a function to run every 2 seconds (2000 milliseconds).
- axios.get('https://dog.ceo/api/breeds/image/random'): Makes a GET request to the Dog CEO API to fetch a random dog image.

- then(): Executes when the API call is successful. The response data, which contains the image URL, is logged to the console and stored in the myimg state.
- catch(): Executes if there's an error with the API call, logging the error to the console.

Write React component that fetches a random joke from the API and displays it when the "Generate Joke" button is clicked.

```
import React,{ useState,useEffect } from 'react'
import axios from "axios";
const Randomjokeapi = () =>
{
   const[joke,setJoke]=useState('');

   function fetchJoke(){
      axios
      .get("https://official-joke-api.appspot.com/random_joke")
      .then((response)=>{setJoke(response.data);})
      .catch((error)=>{console.error(error);})
   }
    useEffect(fetchJoke,[])

   return(
      <div>
      <h1>{joke.setup}</h1>
      <h3>{joke.punchline}</h3>
      <button onClick={fetchJoke} >Generate Joke </button>
      </div>
   )
}
export default Randomjokeapi
```

https://official-joke-api.appspot.com/random_joke link contains object as shown below. We have used setup and punchline properties for our webpage.

```
{"type":"general","setup":"What has ears but cannot hear?","punchline":"A field
of corn.","id":238}
```

- joke: to store the joke data, specifically the setup and punchline properties.
- fetchjoke uses axios.get to fetch a random joke from the API.
- On success, updates the joke state with the setup and punchline from the fetched data.
- On failure, logs the error to the console and updates the error state with a user-friendly message.