

React JS

Introduction

- The React.js framework 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

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.

The command → `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:

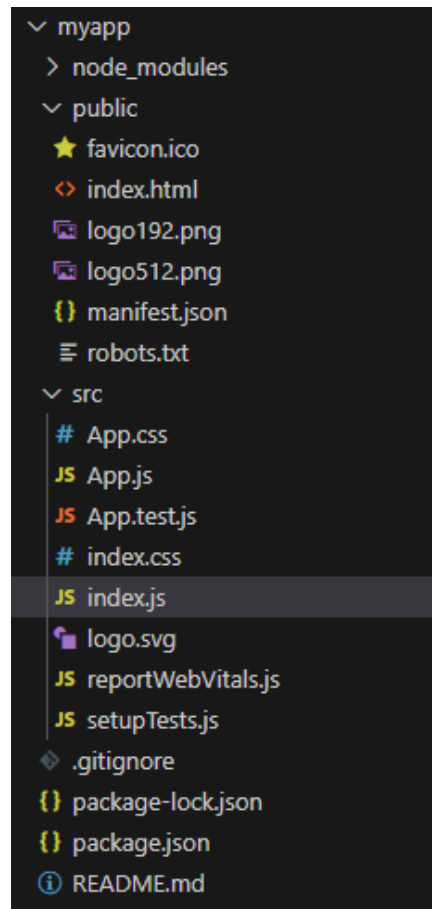
`npm start`

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



Folder Structure

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



- **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 is 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.

for 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'** . you also have to import your own **Myapp.js** file using the **import Myapp from './Myapp'**;

- *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 > by default 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 of `"<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 `` must become ``, and wrapping tags like `"oranges"` must be written as `oranges`.

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.

```

```

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 `}`

```
function Avatar() {
  const pic = 'img.png';
  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 the JavaScript variable called **pic**. 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>
  )
};

```

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.

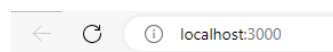
We can do this in two ways.

1) Make changes in App.js file as shown below.

App.js

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

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



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.

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

App.js

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

Ex1.js

```
import React from 'react';
function Ex1() {
  return(
    <div>
      <h1>Hello World!</h1>
    </div>
  )
}
export default Ex1;
```

Note: Please follow 2nd method of importing component to App.js file. In this file just import component and call this component as shown in above example.

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

App.js

```
import Ex2 from './Ex2';

function App() {
  var n = "ABC";
  return (
    <div>
      <Ex2 name={n} rollnum="101" marks="20" />
      <Ex2 name="DEF" rollnum="102" marks="16" />
      <Ex2 name="GHI" rollnum="103" marks="22.5" />
    </div>
  );
}

export default App;
```

Step 2: Read props inside the component

Ex2.js

```
import React from 'react';
const Ex2 = (props)=>{
  return(
    <div>
      <ul>
        <li>{props.name}</li>
        <li>{props.rollnum}</li>
      </ul>
    </div>
  );
}
```

```
        <li>{props.marks}</li>
      </ul>
    </div>
  );
}
export default Ex2;
```

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.

App.js

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

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

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>
  );
}

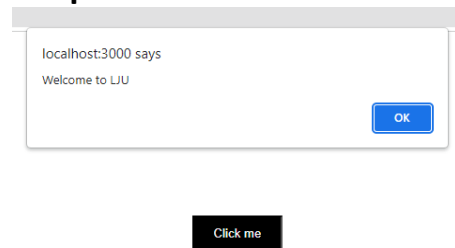
```

```
export default Event1;
```

App.js

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

Output:



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} />
  );
}
```

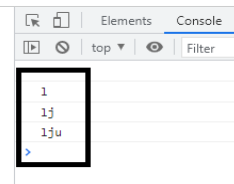
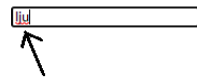
```
export default Event2;
```

App.js

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

Output:

OnChange Example



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;

```

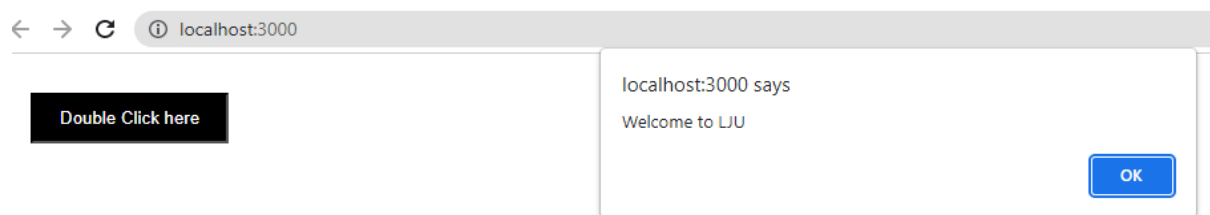
App.js

```

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

```

Output:



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

Create react js app 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
```

App.js

```
import Map1 from "./Map1";

function App() {
  return (
    <div>
      <Map1/>
    </div>
  );
}
export default App;
```

Output:

Example of mapping

Array Element= 1

Array Element= 2

Array Element= 3

Array Element= 4

Array Element= 5

Example-2

Write a program to create ReactJS application having an array of strings and convert it in Uppercase using MAP method.

In Arraymap.js file:

```
import React from 'react'
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. Create react js app to display these multiplied numbers using map function.

Map1.js

```

import React from 'react';
function Map1() {
  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 Map1

```

App.js

```

import Map1 from "./Map1";

function App() {
  return (
    <div>
      <Map1/>
    </div>
  );
}
export default App;

```

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

Create react app which displays images using map function.

Map2.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 Map2() {
  const images=[{
    id:1,
    pic:img1,
  },
  {
    id:1,
    pic:img2,
  },
];
return (
  <div>
    {images.map((val) => {
      return <img src={val.pic} heigth="200px" width="200px" alt="logo" />
    })}
  </div>
)
}
export default Map2
```

App.js

```
import Map2 from "./Map2";

function App() {
  return (
    <div>
      <Map2/>
    </div>
  );
}
export default App;
```

Output: This will display two images.

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

```

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.

Example, in the below case,

Create we have apply filter to skip digit “3” from the 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
```

App.js

```
import Filt1 from "./Filt1";

function App() {
  return (
    <div>
      <Filt1/>
    </div>
  );
}
export default App;
```

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

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 -D 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.

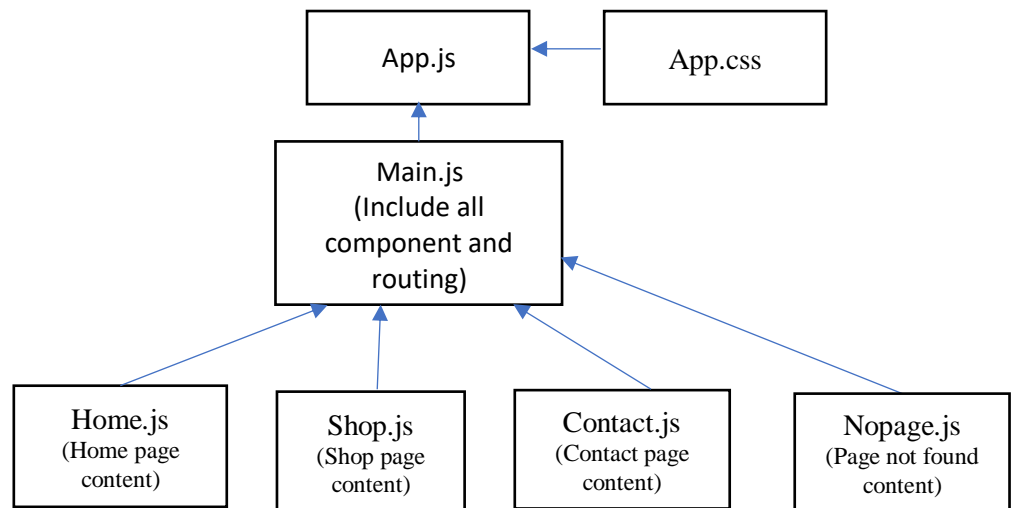
- ✓ **Link:** Link component is used to create links to different routes and implement navigation around the application. It works like HTML anchor tag.

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.

- Now, create files as asked below:
 - **Home.js** - for the home page content
 - **Shop.js** - for the shop page content
 - **Contact.js** - for the contact page content
 - **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.



App.js

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

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

```

    </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 Page from './Page';
import Home from './Home';
import Contact from './Contact';
import Shop from './Shop';
import Nopage from './Nopage';
function Main() {
  return (
    <div class="main-route">
      <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;}
```