# React JS-1

## Introduction

**What is React?**

> ➢ React is a **JavaScript** library and open source project created by **Facebook**
>
> ➢ React is used to build interactive **user interfaces (UI)** components.
>
> ➢ One of the most important aspects of React is the fact that you can create components, which are like custom, reusable HTML elements, to quickly and efficiently build user interfaces. React also streamlines how data is stored and handled, using state and props.
>
> ➢ Popular sites, such as **Instagram, Facebook, Uber, Codecademy** etc uses ReactJS.
>
> ➢ The user interface (UI) is the point of human-computer and communication in device. This can include display screens, keyboard, a mouse and the appearance of a desktop.

**History**

> ➢ React was first designed by **Jorden Walke**, a software engineer at Facebook.
>
> ➢ It was first deployed for Facebook News feed around 2011.
>
> ➢ In 2013, React was open sourced at JS conference.

**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.

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`**

**create-react-app myapp**

==**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 Folder**

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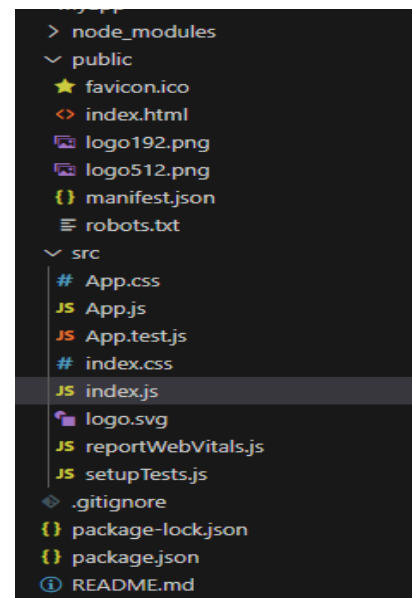
**Webpack** in react is a JavaScript module bundler that is commonly used with React to bundle and manage dependencies. It takes all of the individual JavaScript files and other assets in a project, such as images and CSS, and combines them into a single bundle that can be loaded by the browser.

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

1. **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.

2. **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.**

3. **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.

4. **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

5. **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 Folder**

  **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.

  1. **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. Ex – **Myapp.css**

  2. **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

  3. **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.

  4. *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.

  5. *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';

  6. *logo.svg*

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

  7. *reportWebVital.js*

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

  8. *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**

```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 program. 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. Use JSX fragment
<>...</>" or <div>....</div> etc.
 )
}
export default Ex1
```

## 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 **background-color** you use **backgroundColor**. 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 " " with { }.

```
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 and **"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>
  )};
                                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:
    a. Their names always begin with a capital letter.
    b. They return JSX markup.

**Example1**

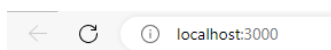**Build basic react app that display "Hello World" in browser.**

**We can do this in two ways.**

**Method-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.**

## Method – 2   Create file named Pr1.js as a component and import this component in App.js file.

**App.js**

```
import Pr1 from "./Pr1";
function App() {
  return (
    <div>
      <Pr1/> // It is component must start with capital letter.
    </div>
  );
}
export default App;
```

**Another File: Pr1.js**

```
import React from 'react'

 function Pr1() {
  return (
    <div>
      <h1>Hello World! </h1>
    </div>
  )
}
export default Pr1
```

**Note:** Always 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.

## Example 2

**Build basic react app with one h1 element in italic, Blue color with font-size 15px., one ordered list of 5 fruits, current time and current date with it.**

**App.js** (Method-1 Perform same using Method-2)

```
function App() {
  const date=new Date().toLocaleDateString()
  const time=new Date().toLocaleTimeString()
  return (
    <div>
      <h1 style = {{color:"blue", fontStyle: "italic", fontSize: "15px"}}>Task 1</h1>
      <p> List of fruits</p>
      {/* <ol> */}
       <ol style={{ textAlign: "center", display: "inline-block" }}>
         <li>Apple</li>
         <li>Lichi</li>
         <li>Kiwi</li>
         <li>Pineapple</li>
```

Prof. Priyen S. Patel

```
        <li>Strawberry</li>
      </ol>
      <h6>Current Date: {date}</h6>
      <h6>Current Time: {time}</h6>
    </div>
  ) }
export default App;
```

**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.

**\*\*Alternatively components can be created by using ES6 arrow function**

Ex1.js

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

# 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.

**Example:**

**Product Name and its price are declared as props in Pr2.js file and the value of props passed in app.js file. A variable can also be passed as props value.**

**In App.js** mention property for passing to specific file (here inside<pr2 property = value >

**In Pr2.js** call the value using: props.property (props is parameter pass in function)

**Step 1: Pass props to the component  In app.js file:**

```
import Pr2 from "./Pr2";

function App() {
 const xyz="Computer";
 return (
 <div>
 <Pr2 Name={xyz} Price="70000" />
 <Pr2 Name="Mobile" Price="20000" />
 </div>
 ) }
export default App;
```

**Step 2: Read props inside the component  In Pr2.js file**

**(*Using ES6 Arrow function):**

```
import React from 'react'
const Example = (props)=>{
 return(
 <div>
 <h1>{props.Name} : {props.Price}</h1>
 </div>
 );
}
export default Example
```

**Output:**

| |
|---|
| Computer : 70000 |
| Mobile : 20000 |

**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.**

**In App.js file:**

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

**In Example.js file:**

```
import React from 'react'

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>
```

---

# onSubmit event

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

\*\* Use the **preventDefault()** method on the event object to prevent a page refresh on form submit in React. It prevents the browser from issuing the default action which in the case of a form submission is to refresh the page.

preventDefault is used to prevent the default form behavior of submitting, reloading etc..

---

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

---

**Write react js script to display onSubmit event.**

---

**In Form.js**

```
import React from 'react';

function Form () {
  function handleSubmit (e) {
    e.preventDefault ();
    alert (' You clicked submit.');
  }
  return (
    <form onSubmit = {handleSubmit}>
```

---

```
        <input type="text"/>Enter name here
        <button type="submit">Submit</button>
    </form>
 );
}
export default Form;
```

**In App.js file:**

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

# 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"
  };
 const handleClick = () => {     //(Note that here ES 6 Arrow Function is used instead of js function)
     alert ('Welcome to LJU');
 }
 return (
  <div>
    <center>
     <button style = {mystyle} onClick={handleClick}>
        {/*<button 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
                                    OK
```
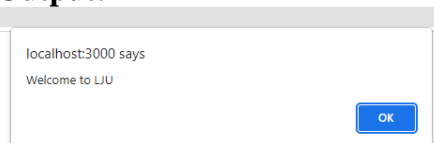
```
Click me
```

# Passing Arguments
To pass an argument to an event handler, use an arrow function.

**Example:**

```
Send "Goal!" as a parameter to the shoot function, using arrow function:
function Football() {
  const shoot = (a) => {
    alert(a);
  }
  return (
    <button onClick={() => shoot("Goal!")}>Take the shot!</button>
  );
}
```

# onChange event

An onChange event handler returns a Synthetic Event object which contains useful meta data such as the target input's id, name, and current value.

We can access the target input's value inside of the handleChange by accessing event.target.value.

**event.target** gives you the element that triggered the event. So, **event.target.value** retrieves the value of that element (an input field, in your example). In React, events are SynthenticEvent, a wrapper around the browser's native event. It has the same interface as the browser's native event, including stopPropagation() and preventDefault(), except the events work identically across all browsers.

- **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;
```
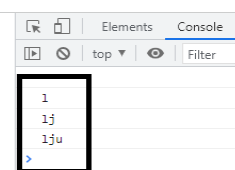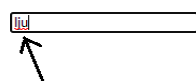
**App.js**

```
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).

**1) Example of DoubleClick event**

**Write react js script to display alert box with text "welcome to lju" only on double click.**
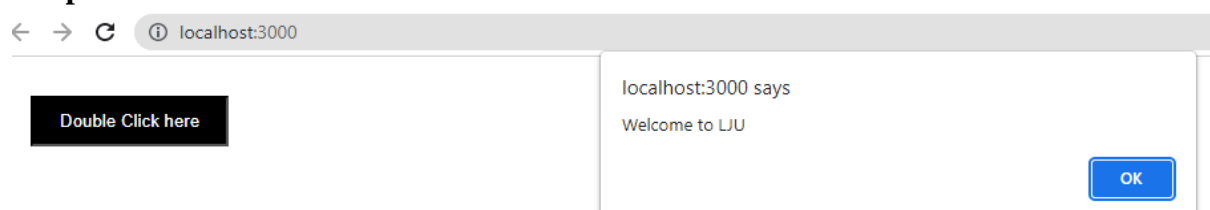
**Event3.js**

```
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:**

**2) Example for 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;
```

# Map and Filter

# Map

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

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.

**Syntax**
array.map(callback[, thisObject]);

Parameter Details
• callback − Function that produces an element of the new Array from an element
of the current one.
• thisObject − Object to use as this when executing callback.

Return Value
Returns the created array.

**Example-1 :**Create react 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:** We have an array of numbers and we want to multiply each of these numbers by 5. Create react app to display these multiplied numbers using map function.

**Map2.js**

```
import React from 'react';

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
```

**App.js**

```
import Map2 from "./Map1";

function App() {
 return (
   <div>
     <Map2/>
   </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-3:** 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:** (Call Arraymap.js in App.js)
**map function**
array values= A
array values= B
array values= C
array values= D
array values= E

**Example-4:** Create react app which displays images using map function.

**Product.js**

```
import React from 'react';
import img1 from "./img1.jpg" //import image from same folder
import img2 from "./img2.jpg" //import image from same folder
function Product() {
   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 Product
```

**App.js**

```
import Product from "./Product";

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

**Output: This will display two images.**

**Example-5:** Create react app which displays array elements which are greater than 3 Using map method. **[map with condition]**

```
function map() {
  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 map
```

**Output:**

Example of mapping with condition
**Array Elements= 4**

**Array Elements= 5**

**Array Elements= 6**

**Array Elements= 4**

# 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
```

**Alternate Way

```
     students.map((student,index) =>
     {
      return <li key={index}>{student.name}</li>
     })
```

# 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 1:** 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

**Example 2:** Write React code to filter out the number greater than 6 using Map method and Filter method to filter the array.

[**Discuss the difference between two method's output.**]

```
import React from 'react'
const ArrayMap_condition = () => {
 const arr1=[1,2,3,4,5,6,7,8,9]
 return (
 <div>
 <h1>Filtered Value</h1>
 {
   arr1.map((value)=>{                          (replace with arr1.filter and observe the o/p)
      if(value <=6){
        var arr1= <h1>array values= {value}</h1>
         return arr1
      }
   })
 }
 </div>
 )
}
export default ArrayMap_condition
```

**Output: Using ".map()" Method**

```
Filtered Value
array values= 1
array values= 2
array values= 3
array values= 4
array values= 5
array values= 6
```

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

**Output: Using ".filter()" Method**

```
Filtered Value

123456
```

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

# 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 Routing

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

Routing is a process in which a user is directed to different pages based on their action or request. ReactJS Router is mainly used for developing Single Page Web Applications. React Router is used to define multiple routes in the application. When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.

React Router is a standard library system built on top of the React and used to create routing in the React application using React Router Package. It provides the synchronous URL on the browser with data that will be displayed on the web page. It maintains the standard structure and behavior of the application and mainly used for developing single page web applications.

**React contains three different packages for routing.** These are:

1. **react-router**: It provides the core routing components and functions for the React Router applications.
2. **react-router-native**: It is used for mobile applications.
3. **react-router-dom**: It is used for web applications design.

**Routing structure**

1. **Store "BrowserRouter as Router"**: A <router> that uses the hash part of the URL to keep your UI in sync with the URL. It is the parent component that is used to store all of the other components.

2. **Routes**: To read a single component, wrap all routes inside the Routes component. Switch groups across multiple routes, iterate over them and find the first one that matches the path. Thus, the corresponding component of the path is rendered.
   An application can have multiple <Routes>. Routes are chosen based on the best match instead of being traversed in order.

3. **Route**: The route component will help us establish the link between the component's UI and the URL. To include the route in the application, add the below code to your app.js.
   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.

4. **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.

It is not possible to install react-router directly in your application. To use react routing, first, you need to install react-router-dom modules in your application. The below command is used to install react router dom.

**Add React Router**
To add React Router in your application, run below command in the terminal from the root directory of the application:
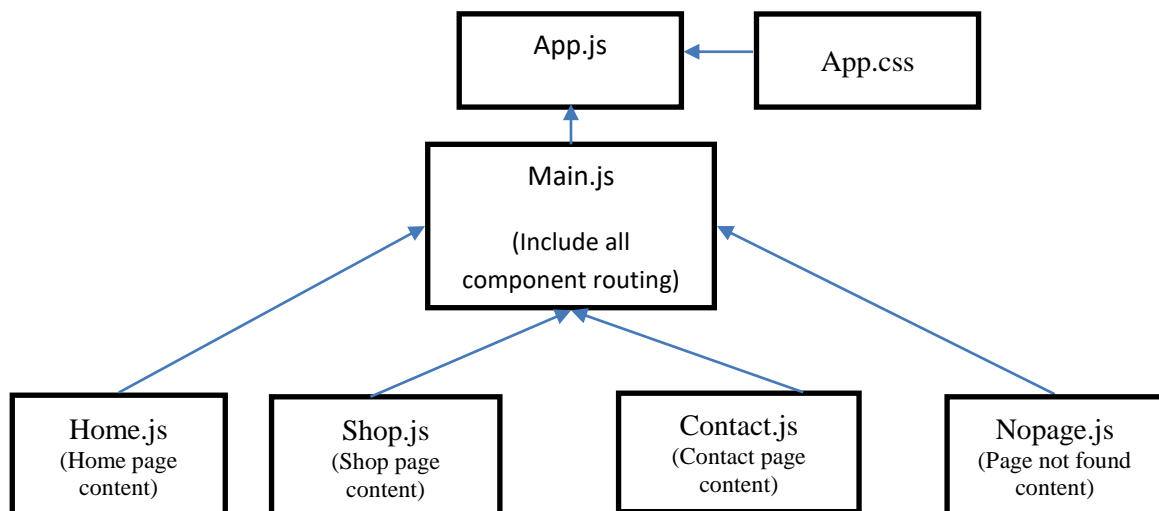
| **npm i react-router-dom** |
|---|

**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 **components** with several files:

**src\components\:**
- Home.js
- Shop.js
- Contact.js
- Nopage.js

**Example:**

Create react app to perform tasks as asked.

- First create files as asked below in component 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**

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

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

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

**components /Main.js**

```
import React from 'react';
import {BrowserRouter as Router,Route,Routes} from "react-router-dom";
import { 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.

**components/Home.js**

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

**components /Shop.js**

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

**components /Contact.js**

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

**components /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)**

```css
.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 tasks

**Task 1 :** Write a React code to prepare product page which contains Name, Image, Price and description of any three product. All data should be passed using props method. Use App. js to call Product.js only.

**Solution**

1. **Prepare Task.js in src folder which get the details from Product.js**

```
import React from 'react'
function Task(props) {
  return (
    <div>
      <h3>{props.name}</h3>
<img style={{heiight:400,width:300}} src={props.img1}/>
<br/>
<b>Price = {props.price}</b>
<p>Description = {props.disc}</p>
    </div>
  )
}
export default Task
```

2. **Product.js in src folder to pass data to Task.js**

```
import React from 'react'
import '../App.css';
import i1 from "./image/11.jpg" //(File in image folder in src folder ../)
import i2 from "./image/12.jpg"
import i3 from "./image/12p.jpg"
import Task from './Task';

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

 <div class="myclass">
  <Task name="Mobile" img1={i1} price="50000Rs" disc="Hello Mi note 11"/>
 </div>
 <div class="myclass">
```

```
  <Task name="Mobile" img1={i2} price="25000Rs" disc="Hello Mi note 12"/>
 </div>
 <div class="myclass">
  <Task name="Mobile" img1={i3} price="10000Rs" disc="Hello Mi note 12"/>
 </div>


    </div>
 </div>
 )}


export default Product
```

### 3. App.js is to call Product.js component.

```
import React from 'react'
import Product from './Product'

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

**\*\*Editorial class in App.css (Not mandatory)**

```
.App {
 text-align: center;
 display:flex;
 justify-content:space-evenly
}

 .myclass {
 text-align:center;
 background-color:azure;
 border:2px solid red;
  }
 .myclass:hover{background-color:burlywood; height:500px;width:600px}
```

**Task 2:** Create react app 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.map((person) => {
       if(person.age > 50){
         return(
          <li>
            {person.name}
          </li>
         )}
     })}
   </ul>
) }
export default M1

//Can also Map after applying Filter
 <ul>
  {
    people.filter((person) => (person.age > 50)).map((person)=>{
                                    return( <h3>{person.name}</h3> )})
  }
  </ul>
```

**Task 4:** Create react app to display name of students in h3 tag of CSE branch from given object.

**M2.js**

```
import React from 'react'
function M2() {
let students = [
```

```
{ id: "001", name: "A1", Branch: "CSE" },
{ id: "002", name: "A2", Branch: "CE" },
{ id: "003", name: "A3", Branch: "CSE" },
{ id: "004", name: "A4", Branch: "CSE" },
{ id: "005", name: "A5", Branch: "IT" }
]
return (
<div>
    {
students.filter((student) => (student.Branch === "CSE")).map((student)=>{
        return <h3>{student.name}</h3>
      }
    })
  }
</div>
) }
export default M2
```

**Task 5:** 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**

**Task 6:** Make a JS file named ex1.js, which contains two functions: one for addition and one for subtraction.
Pass these functions as component inside another function(ex1) in same file and  display addition and subtraction in browser.

**Ex1.js**

```
function Add() {
   return (
      <h1> Addition: {5+5} </h1>
   )
};
function Subtraction() {
   return(
      <div>
         <h1>Subtraction : {7-4}</h1>
      </div>
   )
};
function Ex1() {
   return(
      <div>
         <Add /> //Must be start with capital letter as we are using it as a component
         <Subtraction /> //Must be start with capital letter as we are using it as a component
      </div>
```

```
    )}
export default Ex1;
```

## App.js

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

**Task 7:**

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
```