REACT NOTES

Syllabus

Fundamentals

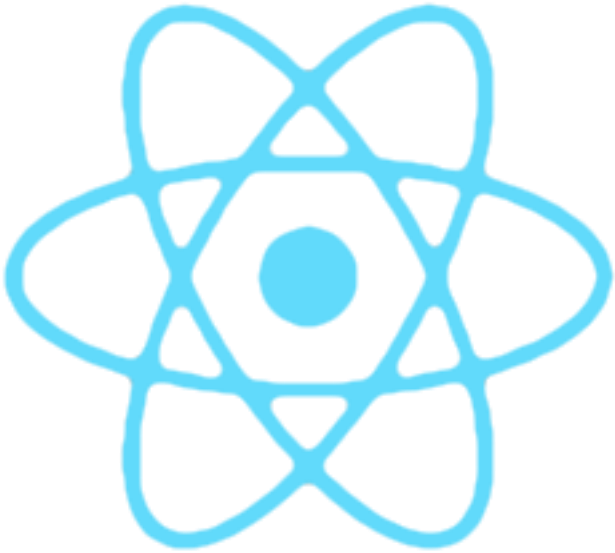
- Introduction
- Implementation
- React app structure
- Components
- functional Components
- class Components
- JSX and babble
- React fragments
- Formatting in react
- Dynamic values
- Adding styles in react
- State management and Side effects
- Event Handling in react js
- Conditional rendering in react
- Mutiple Components
- Displaying Multiple data
- Props
- Fecthing the data in react - fetch() and axios()

Advance

- JSON server (alt of Back end server)
- Error handling
- React Routing
- Route parameters
- Public Routing
- Protected routing
- handling 404 error page
- HTTP methods
- Form handling

Hooks

- useState()
- useEffect()
- useRef()
- useCallback()
- useMemo()
- useReducer()



React

React Introduction

React JS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library responsible only for the view layer of the application. It was created by Jordan Walke, who was a software engineer at Facebook. It was initially developed and maintained by Facebook and was later used in its products like WhatsApp & Instagram. Facebook developed React JS in 2011 in its newsfeed section, but it was released to the public in the month of May 2013.

A ReactJS application is made up of multiple components, each component responsible for outputting a small, reusable piece of HTML code. The components are the heart of all React applications. These Components can be nested with other components to allow complex applications to be built of simple building blocks. ReactJS uses virtual DOM based mechanism to fill data in HTML DOM. The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.

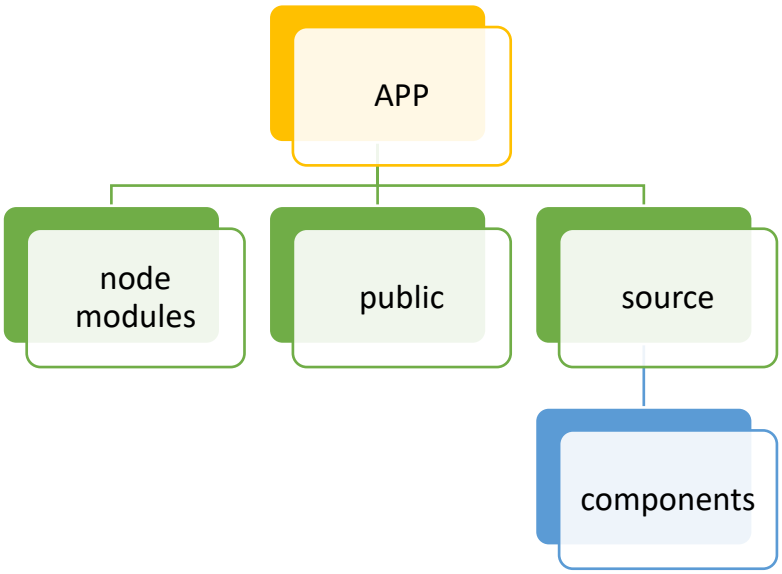
React App Installation

We can install and execute some packages by using npm (node package manager) and npx(node package Extension).The command npm is used to download JavaScript packages from Node Package Manager, and npx is used to execute JavaScript packages downloaded this way

To create react app - "npm create-react-app appname"

To run react app - "npm run start" or "npm start"

Application Structure



Components

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. You can see a UI broken down into multiple individual pieces called components and work on them independently and merge them all in a parent component which will be your final UI.

1) functional Component

If we return a template using a function, this is the most widely used type of component in development mode because of its advantages.

```
function Home()
{
    return(
        <div>
            <h1>This is Home component</h1>
        </div>
    )
}

export default Home
```

2) Class Components

If we return a template using a class with render , now a days class components are being ignored because of it's disadvantages.

```
import { Component } from "react"

class Home extends Component
{
    render()
    {
        return(
            <div>
                <h1>This is Home component</h1>
            </div>
        )
    }
}

export default Home
```

JSX and babbel

JSX is a combination of JS + XML which will allows programmers to write a HTML template inside the JavaScript.

- we can write predefined tags and also custom tags.
- JSX is a special Js value but our run time environment(node js) cannot understand these values.
- Babble as a transpiler which will converts the special JS values to Normal JS values.

React fragments

- In js we know our functions can return a signle value.
- But in react js if we try to return Multiple JSX template , it will throw an error.
- We can resolve it by using fragments in react.
- Recat fragments can be any scementic or JSX fragment (<> </>)

```
1   function Home()
2   {
3       return(
4           <h1>This is Home component</h1>
5           <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
6               Temporibus, molestiae.</p>
7           <button> click button </button>
8       )
9   }
10
11  export default Home
```

In this above picture , we are trying to return multiple jsx structure but it is leading to syntax error , so we need to enclose the multiple jsx with any semantic tag or react fragment <> </> as shown in below image.

```jsx
function Home()
{
    return(
        <div>
            <h1>This is Home component</h1>
            <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
                Temporibus, molestiae.</p>
            <button> click button </button>
        </div>
    )
}

export default Home
```

Formatting in react

React is a combination of HTML (jsx) and JS so we have format the code, when it is nested.

-> <HTML> { JS CODE } <HTML>

-> JS CODE (<HTML>) JS CODE

Dynamic values

In React Js we can access the dynamic values with the help of curly braces

Only numbers and string datatype values can be printed.

```jsx
function Home()
{
    let val = "hello world";

    return(
        <div>
            <h1> { val } </h1>
        </div>
    )
}

export default Home
```

Adding styles in react

Inline - In react inline css should be added as a object properties because it is a JSX(special js value)

```
function Home()
{
    let stylObj = { color : "red" , backgroundColor : "black" }


    return(
        <div style={ stylObj }>
            <h1> { val } </h1>
        </div>
    )
}

export default Home
```

External – we can create a external css document or we can add the styles in one global css document (index.css)

NOTE

HTML class attribute should be used as className to differentiate

with Js class keyword.

Event handling in react Js

Event handling in react can be done by using event listeners as an attributes.

1st case: when events are 0 augmented just pass the function reference to the event listeners.

ex : <button onClick={ function reference }> click </button>

2nd case: when events are augmented we take a help of call backs and then invoke our functions inside the call back function.

ex : <button onClick={ ()=>{ Fname(args) } }> click </button>

```
1   function Home()
2   {
3       let a = ()=>{
4           console.log("this is zero argumented function");
5       }
6
7       let b = (val)=>{
8           console.log("this isargumented function");
9       }
10
11      return(
12          <div>
13              <button onClick={a}> btn1  </button>
14              <button onClick={()=>{ b(10) }}> btn2  </button>
15          </div>
16      )
17  }
18
19  export default Home
```

Displaying multiple data in React

While building react applications , we might face difficulty to display multiple jsx template for multiple data, in that case we can take the help of array map method to return the multiple jsx templates based on the number of values we have in array.

Syntax : array.map((element)=>{ return (<JSX/>) })

```
1   import { useState } from "react"
2
3   function Home()
4   {
5       let[names , setNames] = useState(["alex" , "scott" , "king" , "james" , "raj"])
6
7       return(
8           <div>
9               {
10                  names.map((name)=>{
11                      return(<div key={name}>
12                          <h1>{name}</h1>
13                      </div>)
14                      })
15              }
16          </div>
17      )
18  }
19  export default Home
```

While returning the multiple jsx template from map method we need to give a unique key property value for every element.

The main purpose of keys is to help React differentiate and distinguish elements from each other, increasing its performance when diffing between the virtual and real DOM.

Conditional Rendering

In React we might need to render something based upon some condition , in that case we can use conditional rendering.

Conitional rendering can be done in two ways.

1) Using Logical ampersand

   When we need to render one template based on a condition.

```
1   import { useState } from "react"
2
3   function Home()
4   {
5       let[price , setPrice] = useState(0)
6
7       return(
8           <div>
9               {price==0 && <h1>No Balance in the account</h1>}
10          </div>
11      )
12  }
13  export default Home
```

2) Using Ternary operator

   When we need to render any one out of two template based on a condition.

```
1   import { useState } from "react"
2
3   function Home()
4   {
5       let[age , setAge] = useState(23)
6
7       return(
8       <div>
9       {
10          age>=18 ? <h1>You can vote</h1> : <h1>You cannot vote</h1>
11      }
12      </div>
13      )
14  }
15  export default Home
```

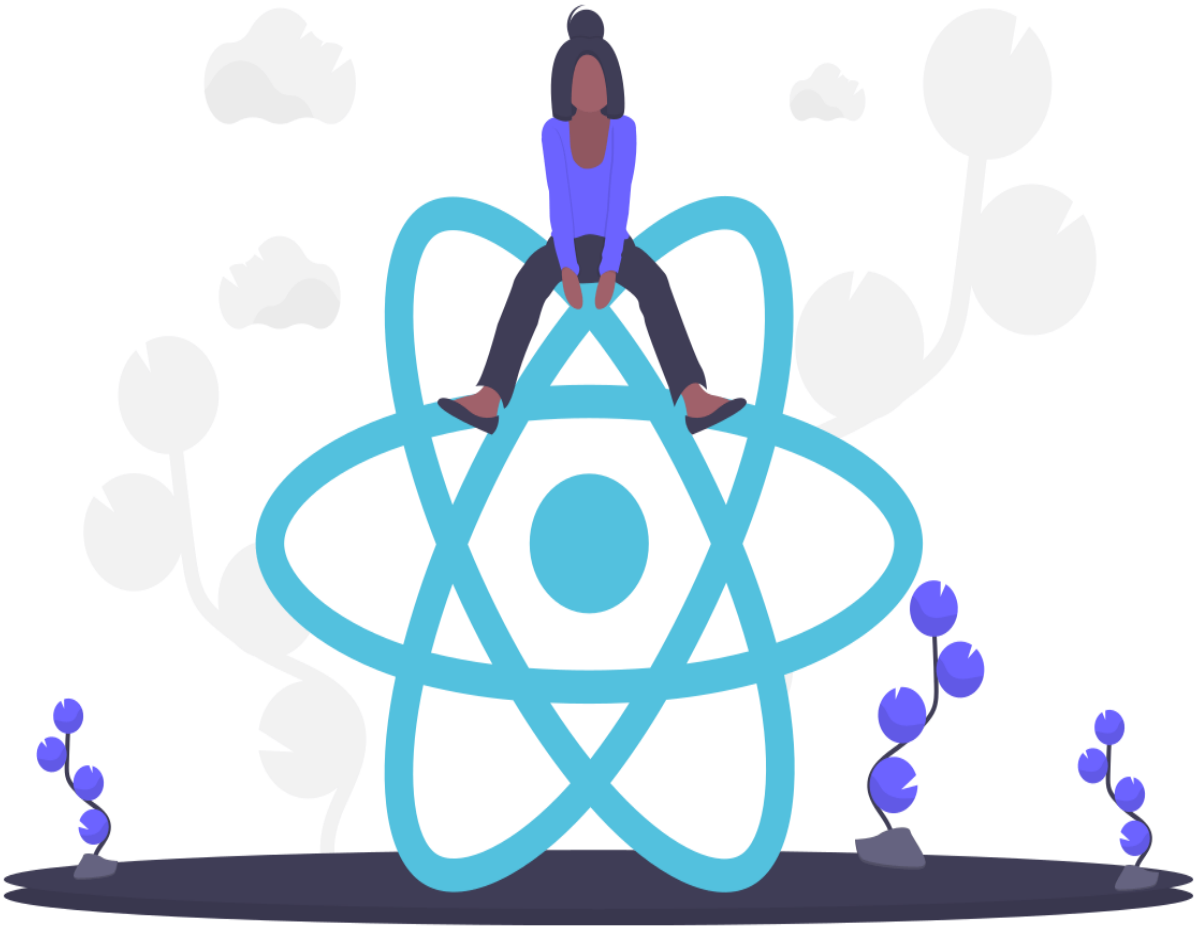
Props

In React Js we can send the data from parent component to child component with the help of props concept.

- Props are arguments passed into React components.
- Props are passed to components via HTML attributes.

```
1   import { useState } from "react"
2   import Child from "./Child";
3
4   function Parent()
5   {
6       let[count , setCount] = useState(100);
7
8       return(
9           <div>
10              <Child count={count}/>
11          </div>
12      )
13  }
14  export default Parent
15
```

```
1   function Child({count})
2   {
3       return(
4           <div>
5               <h1> {count} </h1>
6           </div>
7       )
8   }
9   export default Child
10
```



HOOKS

React Hooks are a function type that allows you to hook into React state and lifecycle features. This was introduced with React 16.8 update and has become a must-needed part of any React application since then.

State management in React

In react we use a dynamic value, and whenever that dynamic is updated the current component should be re-render then only we can see that updated value in UI.

1) useState()

useState hook ca be used to acheve the state managment.

useState will accept a single value and returns an array of two elements.

```
1   import { useState } from "react"
2
3   function Home()
4   {
5       let[price , setPrice] = useState(100)
6
7       return(
8       <div>
9           <h1> {price} </h1>
10          <button onClick={()=>{ setPrice(price+100) }}>
11              incrase price
12          </button>
13      </div>
14      )
15  }
16  export default Home
```

Side Effects in React

In react whenever the component is rendered or re-rendered or removed from the DOM tree we might need to executed some JS statements (side effects for the component) , in this case we use useeffect react hook.

useEffect()

useEffect hook can be used as life cycle methods in functional component.

```
function Home()
{
    let[price , setPrice] = useState(100);

    useEffect( ()=>{
        console.log("All types of renders")
    } )

    useEffect( ()=>{
        console.log("only Initial renders")
    } , [] )

    useEffect( ()=>{
        console.log("Initial renders and re-render of specified dependency")
    } )

    return(
    <div>
        <h1> {price} </h1>
        <button onClick={()=>{ setPrice(price+100) }}>
            incrase price
        </button>
    </div>
    )
}
export default Home
```

useEffect will accept callback and dependency array.

1) If only callback – works for all types renders.
2) If empty dependency - works for only initial render.
3) If any dependency – works for initial render and re-renders of dependency

Replacement of DOM method

In react application to handle the form data we can use dom get methods , but instead if we take a help of useRef we can replace dom methods.

useRef()

useref should be declared , assign and then access it by using current property.

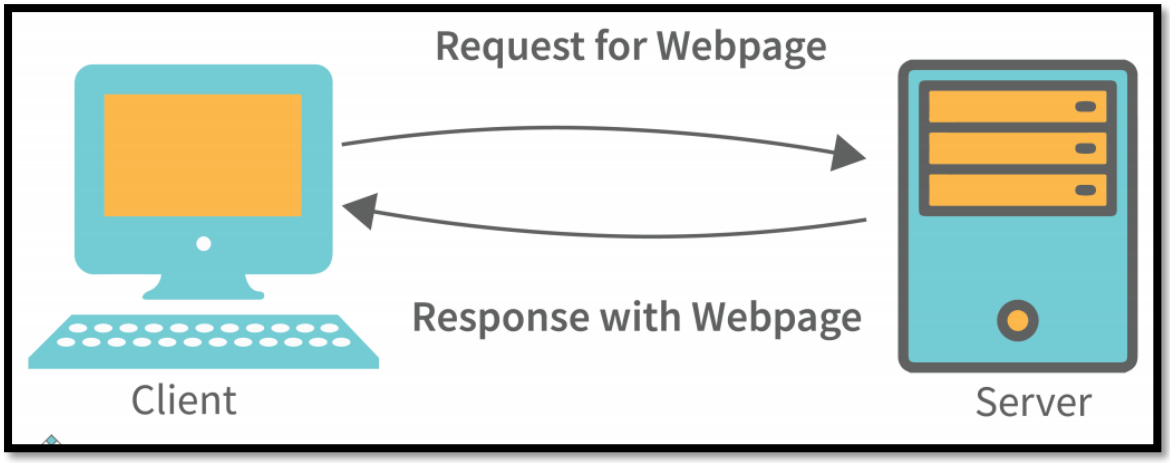
```
import { useRef } from "react";

function Home()
{
    let inpRef = useRef();   1) declare the reference

    let printVal = ()=>{
        console.log(inpRef.current.value); 3) access the element using
    }                                          reference.current property

    return(
    <div>
        <input type="text" ref={inpRef} /> 2) assign the reference with same name

        <button onClick={printVal}>
            access input tag
        </button>
    </div>
    )
}
export default Home
```

ADVANCE CONCEPTS OF REACT JS

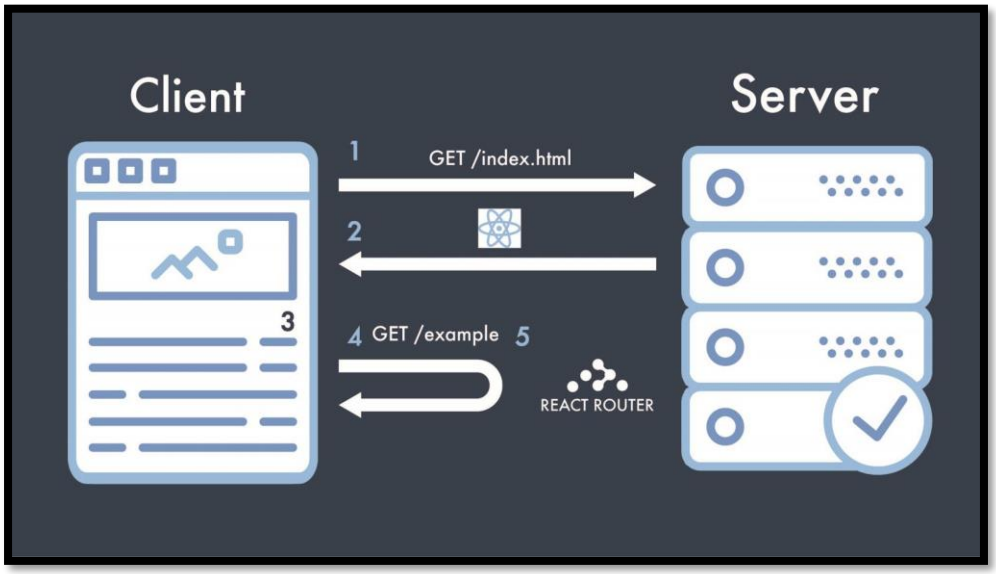
React Routing

Using React library we can develop **single page application** , which means that our entire front end application is just a single HTML page and then changes in content is done within the same page instead of getting a different HTML page.

But in **convectional routing** if user asks a request then browser sends a http request every time to server and server then send back a new html page as a response.



React Router is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.



Need of React Router

React Router plays an important role to display multiple views in a single page application. Without React Router, it is not possible to display multiple views in React applications. Most of the social media websites like Facebook, Instagram uses React Router for rendering multiple views.

React Router Installation

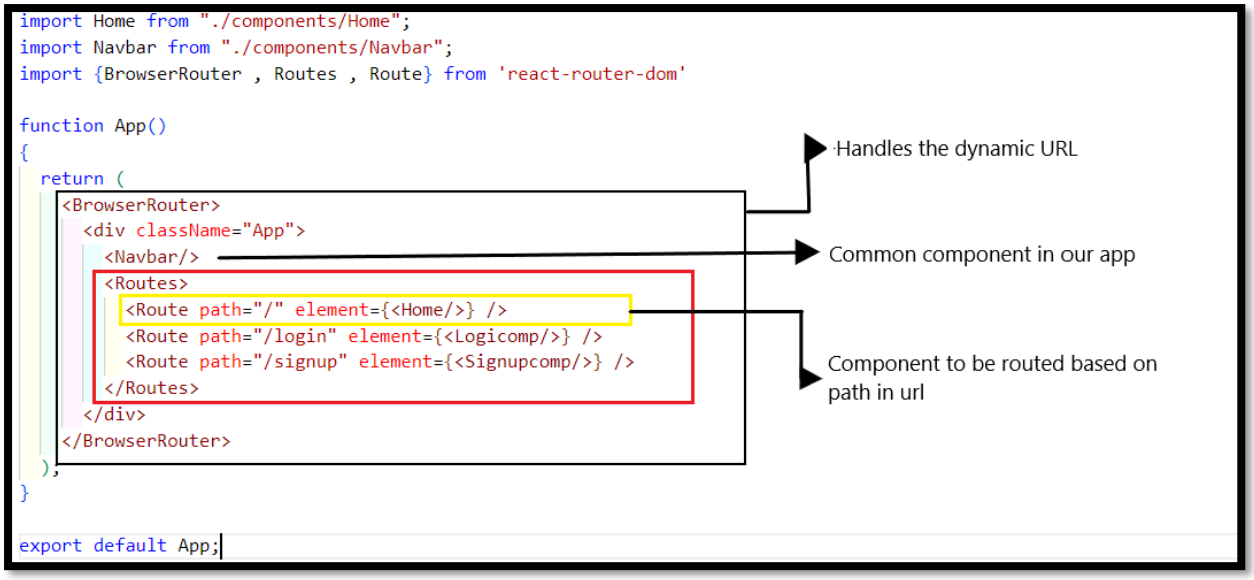
Create React App doesn't include page routing. React Router is the most popular solution. To use react routing, first, you need to install react-router-dom modules in your application. The below command is used to install react router dom.

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

BrowserRouter : It is used for handling the dynamic URL.

Routes : acts as a container/parent for all the individual route that will be created in our app.

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 and to property to create a path in URL

```
import { Link } from "react-router-dom";

function Navbar()
{
    return(
    <nav>
        <Link to="/"><H1>LOGO</H1></Link>

        <Link to="/login">Login</Link>
        <Link to="/signup">Signup</Link>
    </nav>
    )
}
export default Navbar
```

Router Parameter

Route parameters are named URL segments that are used to capture the values specified at their position in the URL. The captured values are populated in the req params object, with the name of the route parameter specified in the path as their respective keys.

Step1 : Adding the dynamic paths for the link.

```
import { Link } from "react-router-dom";

function Navbar()
{
    let userId = 123;

    return(
    <nav>
        <Link to={`/login/${userId}`}>Login</Link>
    </nav>
    )
}
export default Navbar
```

Step12 : saving the dynamic paths value from the url link. As a route parameter.

```
import Logicomp from "./components/Logicomp";
import {BrowserRouter , Routes , Route} from 'react-router-dom'

function App()
{
  return (
    <BrowserRouter>
      <div className="App">
        <Navbar/>
        <Routes>
          <Route path="/login/:userId" element={<Logicomp/>} />
        </Routes>
      </div>
    </BrowserRouter>
  );
}

export default App;
```

Step1 : using the value saved in route parameter by using useParam hook.

```
import { useRef } from "react-router-dom"

function Logicomp()
{
    let {userId} = useRef();
    return(
        <div>
            <h1> {userId} </h1>
        </div>
    )
}
export default Logicomp;
```