**React**

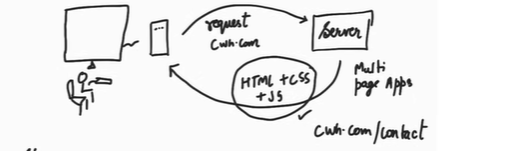
**What is React?**

React is a JavaScript library created by Facebook and React is a User Interface (UI) library React is a tool for building UI components

## What is Babel?

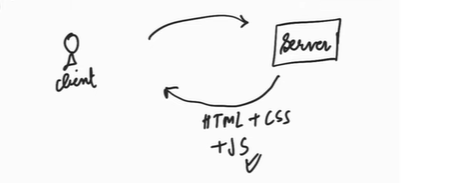
## Babel is a JavaScript compiler that can translate markup or programming languages into JavaScript.Babel is available for different conversions. React uses Babel to convert JSX into JavaScript.

**Difference between single page application or non single page application**



In non single page application

(Suppose we have a web site and we send the request for login page of that webside from server , server return the html, css and js file for login page if I request for another page again server return all three file means our we web page reload again and again for every request)



But in single page application

(Suppose we have a web site and we send the request for login page of that webside from server , server return the html, css and js file for login page then javascript is responsible for rendering every page not relode the website again and again)

1. **Components**

* Functional Components
* Class Components
* JSX(JavaScript XML)

|  |  |
| --- | --- |
| **[Functional Components](https://www.geeksforgeeks.org/reactjs-functional-components/)** | **[Class Components](https://www.geeksforgeeks.org/reactjs-class-based-components/)** |
| A functional component is just a plain JavaScript pure function that accepts props as an argument and returns a React element(JSX).  const Car=()=> {    return <h2>Hi, I am also a Car!</h2>;  } | A class component requires you to extend from React. Component and create a render function that returns a React element.  class Car extends React.Component {    render() {      return <h2>Hi, I am a Car!</h2>;    }} |
| There is no render method used in functional components. It simply return JSX  const FunctionalComponent = () => {    return (        <h3>Functional Component : </h3>    )  }  export default FunctionalComponent; | It must have the render() method for returning JSX  import React, { Component } from "react";  class ClassComponent extends React.Component {    render() {      return (        <h3>Class Component : </h3>      )    }  }  export default ClassComponent; |
| Perversely function components are mainly responsible for rendering UI. So it is known as Stateless components as they simply accept data and display them It can not implement logic and state. But This problem is solved with the help of a special ReactJS concept called “hooks”. | Class components are responsible for rendering UI. As well as implement logic and state. So it is known as Stateful components |
| React lifecycle methods (for example, componentDidMount) cannot be used in functional components | React lifecycle methods can be used inside class components (for example, componentDidMount). |

* JSX(JavaScript XML)
* JSX(JavaScript XML)

**What is JSX ?**

JSX stands for JavaScript XML. JSX is a syntax extension for JavaScript that lets you write HTML inside a JavaScript file.

### **Note**

### JSX and React are two separate things. They’re often used together, but you can [use them independently](https://reactjs.org/blog/2020/09/22/introducing-the-new-jsx-transform.html" \l "whats-a-jsx-transform" \t "https://react.dev/learn/_blank) of each other. JSX is a syntax extension, while React is a JavaScript library.

**The Rules of JSX**

**Return a single root element**

To return multiple elements from a component, **wrap them with a single parent tag. Like**  <div> If you don’t want to add an extra <div> to your markup, you can use Fragmentation <> and </> instead of <div>:

### **Close all the tags :-** JSX requires tags to be explicitly closed: self-closing tags like <img> must become <img />.

**what is JSX Converter in react ?**

JSX Converter in React is a tool or process used to convert JSX code into regular JavaScript code. However, web browsers cannot directly understand JSX code; they only understand plain JavaScript. So, JSX Converter comes into play to translate JSX code into JavaScript code that browsers can interpret.

// JSX Code

const element = <h1>Hello, world!</h1>;

// After converting JSX Code using JSX converter

const element = React.createElement('h1', null, 'Hello, world!');

1. **Props (Properties)**

* **Passing / Accessing Props**
* **Default Props**
* **Props Types**

You use props to pass data and values from one component to another to get dynamic and unique outputs.

import React from 'react'

import ReactDOM from 'react-dom/client'

import App from './App.jsx'

import './index.css'

import Destructring from './Destructring.jsx'

ReactDOM.createRoot(document.getElementById('root')).render(

Here we pass props into the App.js component

    <App  Name="Anurag Roy" Age="18" Address="Bhagalpur"/>

    {/\* We can also pass props as child  \*/}

<p> Right now Anurag facing some problems</p>

    <Destructring

      img="https://ng.jumia.is/unsafe/fit-in/300x300/filters:fill(white)/product/82/6142201/1.jpg?2933"

      name="Cyxus"

      desc="Non-Slip Fitness Leisure Running Sneakers"

      price="$29" />

)

import React from 'react'

How to Access and Use Props in React

The component receives props as a function parameter. It uses the value of props by

defining the parameter as props objects.

function App(props) { //Here App component receives props as a function parameter

console.log(props)

console.table(props)

  return (

    <>

   //it uses the value of props by defining the parameter as props objects

    <h1>Hellow my name is {props.Name} </h1>

    <p>{props.children}</p>

    <h1>and i am {props.Age}year old </h1>

    <h1>and i am from {props.Address} </h1>

    </>

  )

}

export default App

Destructuring Props in React

import React from 'react';

We can also Destructure the props because we know that whatever we pass data in props through parents components props receive all data in object

function Destructring({ img, name, desc, price}) {

    //First Step: Destructuring within the body of the function

    // const {name, desc, price,img}=props

    // console.log(props)

  return (

    <div>

      <div style={{width:"400px", height:"400px ", border:'2px solid black', textAlign:"center"}}>

        <h2>{name}</h2>

        <img src={img} alt=""  style={{width:"300px", height:"150px "}}/>

        <p>{desc}</p>

        <h1>{price}</h1>

      </div>

    </div>

  );

}

export default Destructring;

* **Default Props**
* **Props Types**

import React from 'react';

import PropTypes from 'prop-types'

function PropsType(props) {

  return (

    <div>

    <h1>Hellow my name is {props.Name} </h1>

    <p>{props.children}</p>

    <h1>and i am {props.Age}year old </h1>

    <h1>and i am from {props.Address} </h1>

    </div>

  );

}

// We can also set the type of props means whatever we pass the data as a props

this data belongs to particular type or not

PropsType.propTypes = {

    Name: PropsType.string

  }

// Specifies the default values for props:

PropTypes.defaultProps = {

    name: 'Anurag'

  };

export default PropsType;

1. **Life-cycle Method(Functional Components)**

In React, components have a life-cycle method that consists of three main phases like Mounting Updating and Unmounting Phase. And each phase also contain a set of lifecycle method

These methods allow you to control the component's behavior and perform specific actions at different stages of its lifecycle.

**Note :- we can achieve lifecycle method in functional component through hooks.**

**Phases of lifecycle method**

* Mounting phases
* Updating phases
* Unmounting phases
* Mounting phases

The mounting phase refers to the state when a component is being created and inserted into the DOM. During the mounting phase four built-in lifecycle methods that gets called.

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

1.Constructor()

Constructor() method is called when component is first created and before the loading the page. If you want to initialized the state and binding method we can do that in constructor() method.

1. componentDidMount()

This lifecycle method called when our component render first time

1. render()

This render() method is used to render the JSX and that gives actually outputs the HTML to the DOM.

1. **Hooks (Functional Components)**

* **useState Hooks**
* **useEffect Hooks**
* **useeMemo Hooks**
* **useCallback Hooks**
* **useRef Hooks**

hooks are the method we use to implement logic and functionalities in our components. we can also achieve lifecycle method in functional component through hooks.

( History of Hooks => Perversely function components are mainly responsible for rendering UI. So it is known as Stateless components as they simply accept data and display them It can not implement logic and state. But This problem is solved with the help of a special ReactJS concept called “hooks”.)

* **useState Hooks**

useState() takes a single argument that determines the initial value of the state. This argument can be a string, a number, an array, an object, or any other JavaScript data type. useState() returns an array containing two items. The first item is the current value of the state; the second item is a function that can be used to update the state.

* **useEffect Hooks**

The useEffect in ReactJS is used to handle the side effects such as fetching data and updating DOM(we cant update DOM directly in react we update DOM through virtual DOM but can update through useEffect hook). This hook runs on every render but there is also a way of using a dependency array using which we can control the effect of rendering. it is also capable of handling componentDidMount(), componentDidUpdate(), and componentWillUnmount() life-cycle methods of class-based components into the functional components.

## Controlling side effects in useEffect :

1.  1. If you want to run useEffect on every render do not need to pass any dependency

useEffect(()->{  
 // Example Code  
})

### ****For componentDidMount****

2.  If you  want to run useEffect only once on the first render pass any empty array in the dependency.

useEffect(()->{  
 // Example Code  
}, [] )

### ****For componentDidUpdate****

3.  If you  want to run  useEffect on change of a particular state. Pass the state or props in the dependency array.

useEffect(()->{  
 // Example Code  
}, [props, state] )

### ****For****[componentWillUnmount](https://www.geeksforgeeks.org/reactjs-componentwillunmount-method/" \t "https://www.geeksforgeeks.org/reactjs-useeffect-hook/_blank)

### useEffect(()=>{ return()=>{ //You can add your code for unmounting phase of component console.log("Functional Component Removed ") } },[]) //Write all the code of unmounting phase only inside the callback function

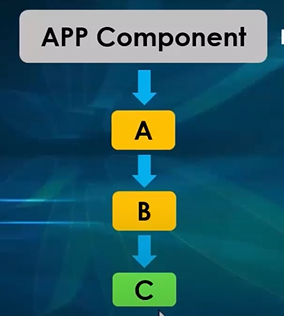
* **useContext Hooks**

**Props Drilling.**

**Scenario -1**



Here App is a Parents Components and A is a Child components I want to pass data from parents to child we can do with the help of props easily



**Scenario -2**

Here App Component is parent component A is Child component and A has also child B and B has also child C component I want to pass data from parents to child C we can do with the help of props . first pass data from APP to A and again pass data A to B and then pass data B to C components I.e called props Drilling.

Now the disadvantage of props drilling is if I want to pass data from parent to child C but we have to pass A and B also. We can solve this problem with the help of useContext hook. It is used to receive the data globally.

**React Context ( Context API)**

It is a way to manage state globally.

Without create context we can not share state of useState hooks or any variable or object data globally but with the help of Context API we can share everything globally.

How to create Context API

Step -1 : First Create Context using createContext()

Step-2 : Then use provider for providing data

Step 3 : Then Recive data using useContext()

App Componanet

import React, { createContext } from 'react'

import ComponentA from './components/ComponetA'

import './App.css'

// if i want to send data from App components  to components C directly so we use useContext

// create context

export const nameContext = createContext()

export const AgeContext=createContext()

function App() {

  return (

    <>

      <nameContext.Provider value={"Anurag"}>

        <AgeContext.Provider value={23}>

        <ComponentA />

        </AgeContext.Provider>

      </nameContext.Provider>

    </>

  )

}

export default App

Component - C

import React, { useContext } from 'react';

import { AgeContext, nameContext } from '../App';

function ComponentC() {

   const myName= useContext(nameContext)

   const myAge=useContext(AgeContext)

  return (

    <div>

      <h1>hello i am componet C</h1>

      <h1>My name is {myName}</h1>

      <h1>And i am {myAge} year old</h1>

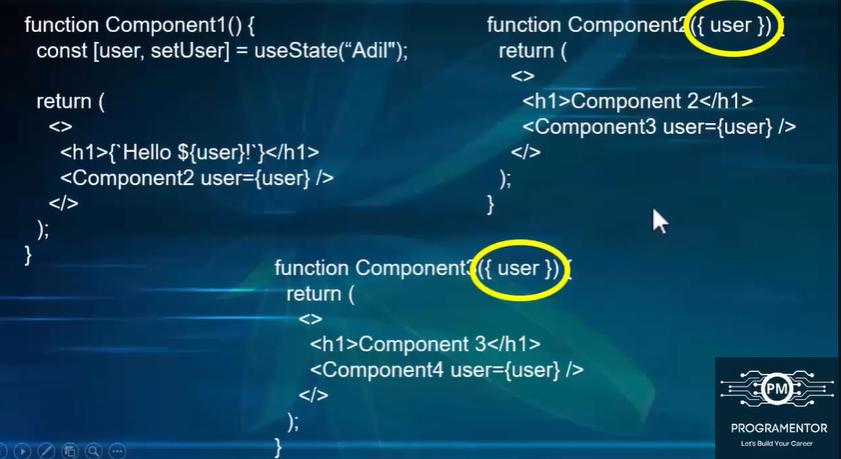
    </div>

  );

}

export default ComponentC;

If you want to pass state without Context , we will need to pass state as a props through each nested components. This is called props drilling



**[What is Memoization?](https://www.geeksforgeeks.org/what-is-memoization-a-complete-tutorial/" \l "def" \t "https://www.geeksforgeeks.org/what-is-memoization-a-complete-tutorial/_blank)**

**Memoization is an optimization technique the can be used to reduce the time consuming calculations by saving previous input to something called cache memory and returning the result from it.**

**Example**

Suppose you write a function for finding the factorial of a number once that function is executed and you got a result of that function after some time you need to get the result of the same function again and again in the same program. In that case same function execute again and again and take so much time for executing the same function. If you want to save the time which take to execute same function again and again so we can store the result of first time execution of function in cache memory once we store the result in cache memory. And then if I call the came function again in that case we don’t need to execute our function simply it return the result from cache memory which is saved from first execution of function. I.e called Memoization.

1. **Event Handling and Form**

* **Handling Events in Components**
* **Handling From Data**

1. **Routing**

In traditional websites, the browser requests a document from a web server, downloads and evaluates CSS and JavaScript assets, and renders the HTML sent from the server. When the user clicks a link, it starts the process all over again for a new page.

Client side routing allows your app to update the URL from a link click without making another request for another document from the server. Instead, your app can immediately render some new UI and make data requests with fetch to update the page with new information.

This enables faster user experiences because the browser doesn't need to request an entirely new document or re-evaluate CSS and JavaScript assets for the next page. It also enables more dynamic user experiences with things like animation.

* Functional Components:

These are simply JavaScript functions. We can create a functional component in React by writing a JavaScript function. These functions may or may not receive data as parameters. In the functional Components, the return value is the JSX code to render to the DOM tree.

**Note**

## Problem with using functional components

They do not have access to dedicated state variables like **class-based components**. This problem is solved with the help of a special ReactJS concept called “hooks”. ReactJS has access to a special hook called [useState()](https://www.geeksforgeeks.org/reactjs-usestate-hook/" \t "https://www.geeksforgeeks.org/reactjs-functional-components/_blank)**.**The **useState()** is used to initialize only one state variable to multiple state variables. The first value returned is the initial value of the state variable, while the second value returned is a reference to the function that updates it.