**React.js**

* **What is reactjs?**

React.js is a **JavaScript library developed by Facebook**, primarily used for building **user interfaces (UIs) for web applications**.

It allows developers to create reusable, self-contained UI components, making complex application interfaces easier to build, maintain, and scale.

**Features**

**Component-Based Architecture:**

The UI is broken down into reusable, modular components.

**Virtual DOM:** A lightweight representation of the real DOM(Document Object Module), enabling efficient updates and rendering.

**Declarative Syntax**: Allows developers to declare what they want the UI to look like, and React takes care of updating it.

**Unidirectional Data Flow:** Data flows in a single direction, making state management more predictable and debugging easier.

**NOTE:-React.js creates a Single Page Application (SPA)**, we mean that React enables the development of web applications that load a **single HTML page initially** and then dynamically update content without fully reloading the page.

* **COMMANDS IN REACT.JS**

**npx create-react-app my-app :** Create a new react application

**npm start :** Runs the application in development mode on a local server (default http://localhost:3000).

**npm install :** Installs all dependencies listed in package.json.

* **What is Single Page Application(SPA)?**

A **Single Page Application (SPA)** is a web application that loads a single HTML page initially and uses JavaScript to update the content dynamically as users interact with the app. Instead of navigating between different HTML pages, the SPA only changes the specific parts of the page that need updating, which provides a seamless user experience.

* **When to Use React.js?**

**Building Single-Page Applications (SPAs)**: React is excellent for SPAs where only parts of the page update rather than reloading the entire page. This creates a smooth, app-like experience for users.

**Developing Dynamic Web Applications**: If your app has interactive features (like social feeds, dashboards, or complex forms), React’s state management and component architecture make it ideal.

**Frequent Updates or Changing Data**: React’s Virtual DOM and component reusability make it efficient to update and manage dynamic data or data that changes frequently.

* **What is jsx**?

JSX (JavaScript XML), is a syntax extension for JavaScript often used with React. It allows you to write HTML-like code within JavaScript, making it easier to build and structure user interfaces.

* **Components in Reactjs?**

It has two components:

* Functional Components and
* Class Components

**Functional Components:**

Functional components are JavaScript functions that accept props as an argument and return React elements.

**EX:**

import React from 'react';

import ReactDOM from 'react-dom/client';

const root=ReactDOM.createRoot(document.getElementById('root'));

function Header(){

let menu= [

    'demo', 'Welocome to ReactJs', 'About ReactJs'

];

    return(

    <header>

        <ul>

            <li>{menu[0]}</li>

            <li>{menu[1]}</li>

            <li>{menu[2]}</li>

        </ul>

    </header>

    );

}

  root.render(<Header/>

    );

**Class Components:**

Class components are ES6 classes that extend **React.Component** and have a **render()** method, which returns React elements.

EX:

import React from 'react';

import ReactDOM from 'react-dom/client';

const root=ReactDOM.createRoot(document.getElementById('root'));

class Demo extends **React.Component**{

**render(){**

        let demoDetails=[

            'ReactDemo', 'Welcome to ReactJs','About ReactJs'

        ]

        return (

                <header>

                    <ul>

                        <li>{demoDetails[0]}</li>

                        <li>{demoDetails[1]}</li>

                        <li>{demoDetails[2]}</li>

</ul>

                </header>

                )

    }

}

root.render(<Header/>

    );

* **Import and Exports**

**It was introduced in js ES6 version**

In React, **imports** and **exports** are used to manage and organize code by separating it into **modules or files.** This allows code to be reused across different parts of an application, keeping it modular and organized**.**

**Imports:**

The purpose of the import statement is to bring in code from other files or modules into the current file, which helps to keep code **organized, modular, and reusable.**

**Exports:**

Types of Exports in Reactjs are:

**Named Exports :**

Named exports allow multiple exports and must use curly braces in import.

When importing, you must use the exact names enclosed in **curly braces { }.**

**EX:**

**App.js**

Named Export

export let **emp**={

name:'Silpa',

sal:20000,

  addr:'nellore'

}

**Index.js**

import React from 'react';

import ReactDOM from 'react-dom/client';

**Here we are importing the named export in curly braces{emp}**

import **{emp}** from './App'

const root=ReactDOM.createRoot(document.getElementById('root'));

function Header(){

console.log(emp)

let menu= [

    'demo', 'Welocome to ReactJs', 'About ReactJs'

];

    return(

    <header>

        <ul>

            <li>{menu[0]}</li>

            <li>{menu[1]}</li>

            <li>{menu[2]}</li>

            </ul>

    </header>

    );

}

    root.render(<Header/>

    );

**Default Exports:**

Default exports allow **only one default** export per file and do not require curly braces in import.

**EX:**

**App.js**

let x=12

export default x

**Index.js**

import React from 'react';

import ReactDOM from 'react-dom/client';

Default export

import **x** from './App'

const root=ReactDOM.createRoot(document.getElementById('root'));

function Header(){

console.log(x)

}

root.render(<Header/>

    );

* **Props in ReactJs**

The main purpose of props(Properties) in reactjs, we can pass some information from Parent Component to Child Component.

Props are **immutable**.

Props are basically used at the run time.

**By using Functional component:**

Once these are defined we can get those values in the component using parameters based on the functional component. So directly we can display those values in components using props. parameter name.

**Ex:**

**App.js 🡪Parent Component**

import User from './User'

function App(){

    return(

        < > **🡪fragments**

        <User name='Silpa' age='22' Address='hyd'/>

        <User name='Janu' age='3' Address='Chennai'/>

         </>

        )

}

export default App

**User.js ---🡪Child component**

function User(props){

    console.log(props)

    return(

        <div>

        <h1>Name:{props.name}</h1>

        <h2>age:{props.age}</h2>

        <h3>Address:{props.Address}</h3>

        </div>

    )

}

**o/p:**

**Name:Silpa**

**age:22**

**Address:hyd**

**Name:Janu**

**age:3**

**Address:Chennai**

**By using Class Component:**

In class components directly we can get through the constructor or use this. props. parameter name, but constructors are not mandatory.

**App.js -🡪parent Component**

import User from './User'

function App(){

    return(

        <>

        <User name='Silpa' age='22' Address='hyd'/>

        <User name='Janu' age='3' Address='Chennai'/>

         </>

        )

}

export default App

**User.js -🡪Child Component**

import React from "react"

class User extends React.component{

    render(){

        return(

                    <div>

                    <h1>Name:{this.props.name}</h1>

                    <h2>age:{this.props.age}</h2>

                    <h3>Address:{this.props.Address}</h3>

                    </div>

                )

    }

}

export default User

**o/p:**

**Name:Silpa**

**age:22**

**Address:hyd**

**Name:Janu**

**age:3**

**Address:Chennai**

* **How to use map( ) method with props in React**

When we try to call the Same component with different values it creates a lot of time and increases the lines of code.

To overcome this issue we have to prefer the **Map** method.

By using that key react library will rerender the elements fastly