React Day – 02

Components in React serve as independent and reusable code blocks for UI elements. They represent different parts of a web page and contain both **structure** and **behavior.** They are similar to **JavaScript functions** and make creating and managing complex user interfaces easier by breaking them down into smaller, reusable pieces.

What are React Components?

React Components are the building block of React Application. They are the reusable code blocks containing logics and and UI elements. They have the same purpose as **JavaScript functions** and return **HTML**. Components make the task of building UI much easier.

Types of Components in React

In React, we mainly have two types of components:

- **Functional Components**
- **Class based Components**

Functional components are just like JavaScript functions that accept properties and return a React element.

```
function welcome() {

    return <h1>Hello, World</h1>;

}
```

Class components are a little more complex than the functional components. A class component can show **inheritance** and access data of other components.

```
class Welcome extends Component {
```

```
    render() {
        return <h1>Hello, World!</h1>;
    }
}
```

Core React Concepts

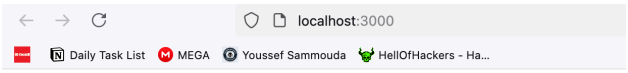
- **Props:** Components can receive data from parent components through props, enabling you to pass information and customize component behavior.
- **State:** Components can manage their internal state using the useState hook. This state dictates the component's appearance and behavior, and updates trigger re-renders.
- **Lifecycle Methods:** React provides lifecycle methods like componentDidMount and componentDidUpdate that allow you to perform actions at specific stages of a component's lifecycle.
- **Conditional Rendering:** Control what gets displayed on the screen based on certain conditions using conditional statements within JSX.

Props

- **Props(short for properties)** are a way to pass data from the parent component to the child component.
- Props are like **function arguments**, you can use them as attributes in components.

```
function Header(props){
    return(<div>
        <h1 >{props.text}</h1>
    </div>);
}
export default Header;
```

```
    <Header text="Hello"/>
    <Header text="Hello 2"/>
```



Hello

Hello 2

Passing information from one component to another

```
function Header(){
    return(
    <div>
        <NavBar name="home"/>
        <NavBar name="about"/>
    </div>);
}
function NavBar(props){
    return(
    <div>
        <h3>{props.name}</h3>
    </div>);
}

export default Header;
```

```
  <React.StrictMode>
    <Header/>
  </React.StrictMode>
```

Class Component Based Props

```
class Temp extends React.Component{
  render(){
    return(<div>
      <h1>{this.props.name}</h1>
      <h1>{this.props.age}</h1>
    </div>)
  }
}
  const root = ReactDOM.createRoot(
  document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Header/>
    <Temp name="raju" age="21"/>
```

```
  </React.StrictMode>
);
```

ReactJS State

The State is a way to store and manage the information or data while creating a React Application. The state is a **JavaScript object** that contains the real-time data or information on the webpage.

the State of a component is an object that holds some information that may change over the lifetime of the component.

Creating State Object

Creating a state is essential to building dynamic and interactive components.

We can create a state object within the **constructor** of the class component.

Approaches To Implementing State in React

There are two ways to implement state in React Components based upon the type of component

- Using the *this.state* object (Class Component)
- Using the *useState hook* (Functional Components)

Example using this.state

```
class MyComponent extends React.Component {
  constructor(){
    super();
    this.state={
      message:"Welcome Visitor"
    }
  }
  changeMessage(){
```

```
    this.setState({message:'Thank You'})
  }
    render() {
        return (
            <div>
                <h1>{this.state.message}</h1>
                <button onClick={()=>
                this.changeMessage()}>Subscribe</button>
            </div>
        );
    }
}
```

Counter Game Example:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import Header from './Header'
import Form from './Form'
import Footer from './Footer';

class MyComponent extends React.Component {
  constructor(){
    super();
    this.state={
      message:0
    }
  }
  changeMessage(){
    this.setState({message:this.state.message+1})
  }
  changeMessage2(){
    this.setState({message:this.state.message-1})
  }
    render() {
        return (
            <div>
                <h1>Count:{this.state.message}</h1>
                <button onClick={()=>
                this.changeMessage()}>Increase</button>
                 <button onClick={()=>
                this.changeMessage2()}>Decrease</button>
            </div>
        );
    }
}
```

```
  const root = ReactDOM.createRoot(
  document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Header/>
    <MyComponent/>
  </React.StrictMode>
);
```

Approaches To Implementing State in React

There are two ways to implement state in React Components based upon the type of component

- Using the *this.state object* (Class Component)
- Using the *useState hook* (Functional Components)

UseState Example Change Names

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom/client';
const root =document.getElementById('root')//root id from
index.html
const VirtualRoot = ReactDOM.createRoot(root);
//virtual Root( V DOM)
function Component(){
  const array=["Ravi","Raju","Ram","Rahul","Raki"]
  const [Name,Change]= useState(0)

  function ChangeName(){
    Change(Name+1)
  }
  return(
    <div>
      <h1>Name: {array[Name]}</h1>
        <button onClick={ChangeName}>Click me</button>
    </div>
  )
}
VirtualRoot.render(
  <div>
    <Component/>
  </div>
)
```

Implement state using react hooks Example:

```
const MyComponent =()=>{
  const [count,setCount]=useState(0);
  const Increase=()=>{
    setCount(count+1);
  }
  return(
    <div><h1>Count : {count}</h1>
    <button onClick={Increase}>Increase</button></div>
  )
}
```

2nd way using arrow function in onclock

```
const MyComponent =()=>{
  const [count,setCount]=useState(0);

  return(
    <div><h1>Count : {count}</h1>
    <button onClick={()=>setCount((count)=>count+1)}>Increase</
button></div>
  )
}
```

React Hooks

In React, Hooks are reusable functions that provide access to state in React Applications. allow developers to hook into the state and other React features without having to write a class.

Types of React Hooks

The_Built-in React Hooks are:

- **State Hooks**
- **Context Hooks**
- **Ref Hooks**
- **Effect Hooks**
- **Performance Hooks**
- **Resource Hooks**

1. React State Hooks

State Hooks stores and provide access to the information. To add state in Components we use:

- **useState Hook**: useState Hooks provides state variable with direct update access.
- useReducer Hook: useReducer Hook provides a state variable along with the update logic in reducer function.

Use Reducer Hook

```
import { useReducer } from 'react';

const initialValue =0
const reducer =(state,action)=>{
  switch(action){
    case "add":
      return state+1;
    case "sub":
      return state-1;
    case "clear":
      return 0;
    default:
      return "error";
  }
}
const MyComponent = ()=>{
  const [count,action]= useReducer(reducer,initialValue);
  return(
  <div>
    <h2>{count}</h2>
    <button onClick={()=>action("add")}>Add</button>
    <button onClick={()=>action("sub")}>sub</button>
    <button onClick={()=>action("clear")}>Clear</button>
  </div>)
}
```

ReactJS useEffect Hook

The useEffect Hook allows us to perform side effects on the components. fetching data, directly updating the DOM and timers

are some side effects. It is called every time any state if the dependency array is modified or updated.

React useEffect Hook ShortHand for:

- **FUNCTION:** contains the code to be executed when useEffect triggers.
- **DEPENDENCY:** is an optional parameter, useEffect triggers when the given dependency is changed.

```
//useEffect(<FUNCTION>, <DEPENDECY>)
```

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

const HookCounterOne=()=> {
  const [count, setCount] = useState(0);
 //useEffect(<FUNCTION>, <DEPENDECY>)
  useEffect(
    () => {document.title = `You clicked ${count} times`;}
    ,[count]);

  return (
      <div>
          <button onClick={() =>
             setCount((count) => count + 1)}>
              Click {count} times
          </button>
      </div>
  );
}
```

React Router

React Router, is your essential tool for building single-page applications (SPAs). Imagine users effortlessly transitioning between sections, experiencing your website like a fluid app and React Router makes it possible, paving the way for a delightful user experience and a modern web presence. A React website shouldn’t mean a Large page reloads every time users navigate.

Steps to Use React Router

Step 1: Initialize a react project. Check this post for setting up React app

Step 2: Install react-router in your application write the following command in your terminal

npm i react-router-dom, npm i react-router-dom

Step 3: Importing React Router

import { BrowserRouter, Routes, Route } from "react-router-dom";

React Router Components

The Main Components of React Router are:

- **BrowserRouter:** BrowserRouter is a router implementation that uses the HTML5 history API(pushState, replaceState, and the popstate event) to keep your UI in sync with the URL. It is the parent component that is used to store all of the other components.
- **Routes:** It's a new component introduced in the v6 and an upgrade of the component. The main advantages of Routes over Switch are:
  - Relative s and s
  - Routes are chosen based on the best match instead of being traversed in order.
- **Route:** Route is the conditionally shown component that renders some UI when its path matches the current URL.
- **Link:** The link component is used to create links to different routes and implement navigation around the application. It works like an HTML anchor tag.

Example

Index.js

```
import React, { useEffect, useReducer, useState } from 'react';
import ReactDOM from 'react-dom/client';
import Header from './components/Header';
import Login from './components/Login'
import {BrowserRouter as Router,
  Routes,
  Link,
  Route} from 'react-router-dom'
import Register from './components/Register';
const root =document.getElementById('root')//root id from
index.html
```

```
const VirtualRoot = ReactDOM.createRoot(root);


//virtual Root( V DOM)
VirtualRoot.render(

  <Router>
  <div>
    <Header/>
  </div>
  <Routes>
      <Route path="/login" element={<Login/>} />
      <Route path="/register" element={<Register/>} />
    </Routes>
  </Router>
)
```

Header.js

```
import '../style.css'
import {Link} from 'react-router-dom'
export default function Header(){
    return(
            <div class="header">
                <h3>FlipKart</h3>
                <ul>
                    <li>
                        <Link to="/">Home</Link>
                    </li>
                    <li><Link to="/login">Login</Link></li>
                    <li><Link to="/register">Register</Link></li>
                </ul>
            </div>


    )
}
```