React Interview Question

**1.What is React?\\**

React(React.js or ReactJS) is an **open-source front-end JavaScript library** that is **used for building user interfaces or UI component,** especially for **single-page applications.**

It is used for handling view layer for web and mobile apps based on components in a declarative approach.

React was created by [Jordan Walke](https://github.com/jordwalke), a software engineer working for Facebook. React was first deployed on Facebook's News Feed in 2011 and on Instagram in 2012.

**What is Single Page Application?**

A Single Page Application (SPA) is a web application that loads a single HTML page and dynamically updates the content as the user interacts with the app, without requiring a full page reload.

This approach enhances user experience by providing a more fluid and responsive interface, often using JavaScript frameworks like React, Angular, or Vue.js.

**2.What are the major features of React?**

The major features of React are:

* Uses **JSX** syntax, a syntax extension of JS that allows developers to write HTML in their JS code.
* It uses **Virtual DOM** instead of Real DOM considering that Real DOM manipulations are expensive.
* Supports **server-side rendering** which is useful for Search Engine Optimizations(SEO).
* Follows **Unidirectional or one-way** data flow or data binding.
* Uses **reusable/composable** UI components to develop the view.

**3. What are the limitations of React?**

The few limitations of React are as given below:

* **Not a Full Framework**: React is a library focused on UI, meaning you may need to integrate additional libraries for routing, state management, and other functionalities, which can lead to fragmentation. The components of React are numerous and will take time to fully grasp the benefits of all.
* **Using JSX** Coding might become complex as it will make use of inline templating and JSX.
* **Boilerplate Code**: Setting up a React project often requires a lot of boilerplate code, especially when integrating with state management libraries (like Redux) or routing.
* **State Management**: Managing state can become complex in larger applications, and while there are solutions like Redux and MobX, they add additional complexity.

### 4. What is JSX?

JSX stands for JavaScript XML.

It allows us to write HTML inside JavaScript and place them in the DOM without using functions like appendChild( ) or createElement( ).

Basically it just provides the syntactic sugar for the React.createElement(type, props, ...children) function, so we don’t need to write all these function just use JSX.

Note- We can create react applications without using JSX as well.

JSX is stricter than HTML.

Let’s understand **how JSX works**:

Without using JSX, we would have to create an element by the following process:

const text = React.createElement('p', {}, 'This is a text');

const container = React.createElement('div','{}',text );

ReactDOM.render(container,rootElement);

**Using JSX**, the above code can be simplified:

const container = (

    <div>

      <p>This is a text</p>

    </div>

    );

    ReactDOM.render(container,rootElement);

As one can see in the code above, we are directly using HTML inside JavaScript.

**How browser understand JSX?**

With the help of Bable react understand JSX.

**What is Babel?**

Babel is a very famous **transpiler** that basically allows us to use future JavaScript in today’s browsers.

In simple words, it can convert the latest version of JavaScript code into the one that the browser understands.

Transpiler is a tool that is used to convert source code into another source code that is of the same level.

The latest standard version that JavaScript follows is ES2020 which is not fully supported by all browsers hence we make use of a tool such as ‘babel’ so that we can convert it into the code that today’s browser understands.

## **Using Babel with React**

We use Babel with React to transpile the JSX code into simple React functions that can be understood by browsers. Using this way we can assure that our JSX code can work in almost any browser. This combination is widely used in modern-day web development.

### 5. What is the difference between Element and Component?

### An Element is a plain object and are the smallest building blocks of React apps.

### An element describes what you want to appear on the screen.

### React elements are cheap to create.

The JavaScript representation(Without JSX) of React Element would be as follows:

const element = React.createElement("div", { id: "login-btn" }, "Login");

and this element can be simiplified using JSX

<div id="login-btn">Login</div>

<div id="login-btn">Login</div>

The above React.createElement() function returns an object as below:

{

  type: 'div',

  props: {

    children: 'Login',

    id: 'login-btn'

  }

}

Finally, this element renders to the DOM using ReactDOM.render().

To render a React element, first pass the DOM element to [ReactDOM.createRoot()](https://legacy.reactjs.org/docs/react-dom-client.html#createroot), then pass the React element to root.render():

<div id="root"></div>:

const root = ReactDOM.createRoot(

    document.getElementById('root')

  );

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

  root.render(element);

**Component**

A **Component** is one of the core building blocks of React that manage state and behavior.

In other words, we can say that every application you will develop in React will be made up of pieces called components.

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.

It can be a class with a render() method or it can be defined as a function. In either case, it takes props as an input, and returns a JSX tree as the output.

const Button = ({ handleLogin }) => (

    <div id={"login-btn"} onClick={handleLogin}>

      Login

    </div>

  );

## **6. Types of Components in React JS**

Components are the building blocks of creating User Interfaces(UI) in React. There are two possible ways to create a/ component.

**1. Function Components:**

This is the simplest way to create a component. Those are pure JavaScript functions.  It is a pure function that takes in props and returns JSX elements.

Unlike class components, functional components do not have their own state or lifecycle methods.

**Characteristics of Functional Components**

Here are some key characteristics of functional components:

* **Pure Functions**: Functional components are pure functions, meaning they do not have any side effects and always return the same output given the same inputs.
* **No State**: Functional components do not have their own state. If you need to manage state, you can use the **useState** hook.
* **No Lifecycle Methods**: Functional components do not have lifecycle methods like **componentDidMount** or **componentWillUnmount**. Instead, you can use the **useEffect** hook to handle side effects.
* **No this Context**: Functional components do not have a **this** context, so you don't need to worry about binding functions to the component.

**Without props**

function demoComponent() {

    return (<h1>

                Welcome Message!

            </h1>);

}

**With Props**

function Welcome(props) {

    return <h1>Hello, {props.name}</h1>;

  }

**1. Class Components:**

We can use JavaScript ES6 classes to create class-based components in React.

In React, a class component is a type of component that is defined as a class. It is a way to create reusable UI components that can manage their own state and lifecycle.

The class components are a little more complex than the functional components.

The functional components are not aware of the other components in your program whereas the class components can work with each other.

We can pass data from one class component to another class component.

**Characteristics of Class Components**

Here are some key characteristics of class components:

* **State**: Class components can have their own state, which is an object that stores data that can change over time.
* **Lifecycle Methods**: Class components have lifecycle methods, such as **componentDidMount** and **componentWillUnmount**, that allow you to execute code at specific points in the component's life cycle.
* **this Context**: Class components have a **this** context, which refers to the component instance.
* **Inheritance**: Class components can inherit behavior from parent classes using inheritance.

class Democomponent extends React.Component {

    render() {

        return <h1>Welcome Message!</h1>;

    }

}



**Lifecycle Methods in Class Components**

Here are some common lifecycle methods in class components:

* **constructor**: Called when the component is created.
* **componentDidMount**: Called when the component is mounted to the DOM.
* **componentDidUpdate**: Called when the component's props or state change.
* **componentWillUnmount**: Called when the component is unmounted from the DOM.
* **render**: Called when the component needs to render its JSX elements.

## Functional Components vs. Class Components

Now that we have seen what functional and class components are, let’s look at some differences between them.

|  |  |  |
| --- | --- | --- |
|  | **Functional Components** | **Class Components** |
| **Syntax** | Functional components are written as a JavaScript function. | Class components are written as a JavaScript class. |
| **State and Lifecycle Methods** | Functional components do not have a state or lifecycle methods. | Class components have a state and can implement lifecycle methods like componentDidMount and componentDidUpdate. |
| **Performance** | Faster as they do not have state and lifecycle, react needs to do less work to render these components. | Slower as they have state and lifecycle, react needs to do comparatively more work to render these components. |
| **Code Length** | Functional components tend to be shorter and more concise | Class components require the boilerplate code, such as a constructor method and the use of “this” to access props and state. |
| **Usage of “this”** | Functional components do not use “this” at all, which makes them easier to understand for beginners. | Class components use the “this” keyword is used to refer to the current instance of the component which can be confusing for new developers. |
| **Use Of Constructor** | There is no use of Constructor in functional component | We can use constructor in class component. |
| **Render Method** | There is no render method in functional Component. | Render method is compulsory. |

### 7. What are Pure Components?

PureComponent is similar to [Component](https://react.dev/reference/react/Component) but it skips re-renders for same props and state and render the same output for the same state and props.

## **Pure Components Key Points:**

* **Shallow Comparison:**
  + Pure components perform a shallow comparison of the props and states. If the objects are passed as props or state have the same references, a re-render is prevented.
* **Performance Optimization:**
  + Pure components can provide performance optimizations by preventing unnecessary re-renders when the data is same and hasn’t modified.
* **ShouldComponentUpdate:**
  + Pure components automatically implement the [shouldComponentUpdate()](https://www.geeksforgeeks.org/reactjs-shouldcomponentupdate-method/) method with a shallow prop and state comparison. This method returns false if the props and state haven’t changed.
* **Easier Debugging:** Pure Components can make it easier to debug your application by reducing the number of unnecessary re-renders.

**Class Component**

import React from 'react';

class PercentageStat extends React.PureComponent {

  render() {

    const { label, score = 0, total = Math.max(1, score) } = this.props;

    return (

      <div>

        <h6>{ label }</h6>

        <span>{ Math.round(score / total \* 100) }%</span>

      </div>

    )

  }

}

export default PercentageStat;

**Functional Component**

import { memo, useState } from 'react';

const EmployeeProfile = memo(

function EmployeeProfile({ name, email }) {

  return (<>

        <p>Name:{name}</p>

        <p>Email: {email}</p>

        </>);

}

);

export default function EmployeeRegForm() {

  const [name, setName] = useState('');

  const [email, setEmail] = useState('');

  return (

    <>

      <label>

        Name: <input value={name} onChange={e => setName(e.target.value)} />

      </label>

      <label>

        Email: <input value={email} onChange={e => setEmail(e.target.value)} />

      </label>

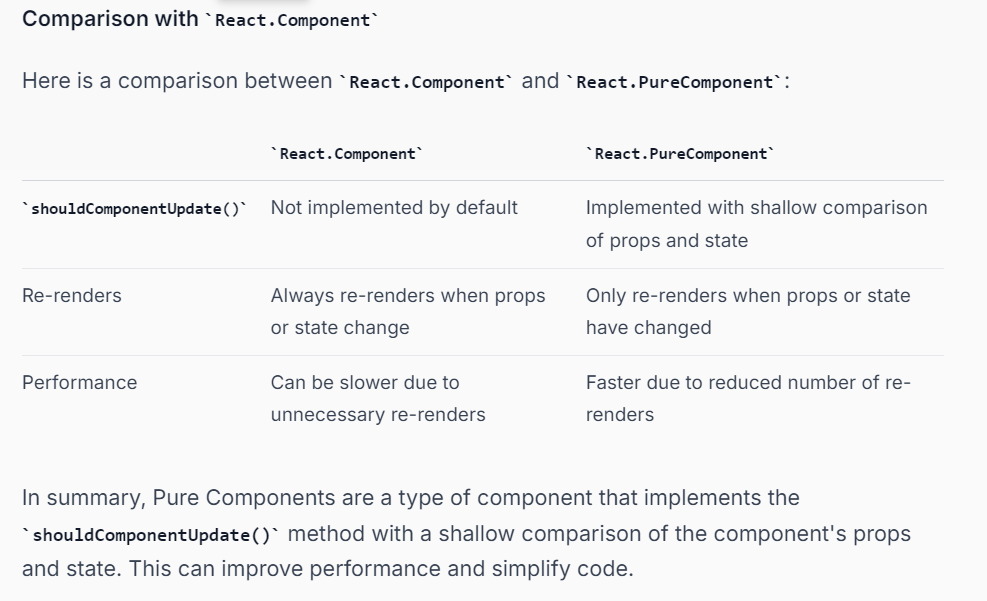
      <hr/>

      <EmployeeProfile name={name} email={email}/>

    </>

  );

}



### 8. What is state in React?

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

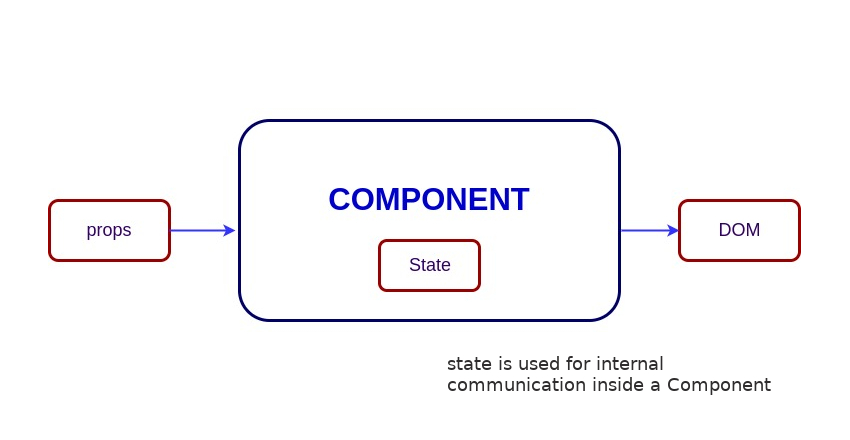
The important point is whenever the state object changes, the component re-renders.

**Types of State**

There are two types of state in React:

* **Local State**: This is the state that is stored within a component. Local state is used to store data that is specific to a component.
* **Global State**: This is the state that is shared across multiple components. Global state is used to store data that is shared across the application.

Note :



Functional Component Example

import React, { useState } from "react";

function User() {

  const [message, setMessage] = useState("Welcome to React world");

  return (

    <div>

      <h1>{message}</h1>

    </div>

  );

}

Class Component Example

import React from 'react';

class User extends React.Component {

  constructor(props) {

    super(props);

    this.state = {

      message: "Welcome to React world",

    };

  }

  render() {

    return (

      <div>

        <h1>{this.state.message}</h1>

      </div>

    );

  }

}

### 9. What are props in React?

Props stand for "**Properties**."

They are **read-only** components.

It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.

It gives a way to pass data from one component to other components.

It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.

Functional Component Example

import React from "react";

import ReactDOM from "react-dom";

const ChildComponent = (props) => {

  return (

    <div>

      <p>{props.name}</p>

      <p>{props.age}</p>

    </div>

  );

};

const ParentComponent = () => {

  return (

    <div>

      <ChildComponent name="John" age="30" />

      <ChildComponent name="Mary" age="25" />

    </div>

  );

};

Class Component Example

App.js

import React, { Component } from 'react';

class App extends React.Component {

   render() {

      return (

          <div>

            <h1> Welcome to { this.props.name } </h1>

            <p> <h4> Javatpoint is one of the best Java training institute in Noida, Delhi, Gurugram, Ghaziabad and Faridabad. </h4> </p>

          </div>

      );

   }

}

export default App;

Main.js

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App.js';

ReactDOM.render(<App name = "JavaTpoint!!" />, document.getElementById('app'));

## **10. Differences between props and state**

| **PROPS** | **STATE** |
| --- | --- |
| The Data is passed from one component to another. | The Data is passed within the component only. |
| It is Immutable (cannot be modified). | It is Mutable ( can be modified). |
| Props can be used with class and functional components. | The state can be used only with the state components/class component (Before 16.0). |
| Props are read-only. | The state is both read and write. |

# **11. Why should we not update the state directly?**

* If you try to update state directly then it won't re-render the component.
* Instead use setState() method.
* It schedules an update to a component's state object. When state changes, the component responds by re-rendering.

## **12. Differences between Real DOM and Virtual DOM?**

**Real DOM :**

The Real DOM, also known as the actual DOM, is the browser’s representation of a web page’s HTML structure.

When a user interacts with a web page, such as clicking a button or filling out a form, the browser updates the Real DOM to reflect the changes.

The browser then re-renders the page to display the updated HTML.

The changes and updates to the DOM are fast because of its tree-like structure but re-rendering whole documents makes the DOM Slow.

All UI components need to be re-rendered for every DOM update**.**

**Virtual DOM :**

A virtual DOM is a lightweight JavaScript object which originally is just a copy of the real DOM

Virtual DOM makes the performance faster, not because the processing itself is done in less time. The reason is the amount of changed only that element which is changed by user not entire DOM.

This Virtual DOM works in three simple steps.

1. Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.
2. Then the difference between the previous DOM representation and the,new one is calculated.
3. Once the calculations are done, the real DOM will be updated with only the things that have actually changed.
4. Patch: React creates a patch, which is a set of instructions to update the real DOM.
5. Update Real DOM: React applies the patch to the real DOM, updating only the elements that have changed.



# **13.Controlled vs Uncontrolled Components**

## **Controlled Components**

In React, Controlled Components are those which are controlled or handled by the component’s state.

Controlled Component always takes an input or a form.

It takes its current value through props and makes changes through callbacks like onClick, onChange, etc.

// FileName - App.js

import { useState } from "react";

import "./App.css";

function App() {

    const [name, setName] = useState(""); // Create a state to control value

    function handleSubmit() {

        alert(`Name: ${name}`);

    }

    return (

        <div className="App">

            <h1 className="geeks">GeeksForGeeks</h1>

            <h3>Controlled Component</h3>

            <form onSubmit={handleSubmit}>

                <label>Name:</label>

                <input

                    name="name"

                    value={name}

                    onChange={(e) =>

                        setName(e.target.value)

                    }

                />

                <button type="submit">Submit</button>

            </form>

        </div>

    );

}

export default App;

## **Un-Controlled Components**

Uncontrolled Components are the components that are not controlled by the React state and are handled by the DOM (Document Object Model).

DOM means we can controlled by getElementByID(), querySelector() etc or by “useRef”

So in order to access any value that has been entered we take the help of refs.

// FileName - App.js

import React, { useRef } from "react";

import "./App.css";

function App() {

    const inputRef = useRef(null);

    function handleSubmit() {

        alert(`Name: ${inputRef.current.value}`);

    }

    return (

        <div className="App">

            <h1 className="geeks">GeeksForGeeks</h1>

            <h3>Uncontrolled Component</h3>

            <form onSubmit={handleSubmit}>

                <label>Name :</label>

                <input

                    type="text"

                    name="name"

                    ref={inputRef}

                />

                <button type="submit">Submit</button>

            </form>

        </div>

    );

}

export default App;

|  |  |  |
| --- | --- | --- |
| **SN** | **Controlled** | **Uncontrolled** |
| **1.** | It does not maintain its internal state. | It maintains its internal states. |
| **2.** | Here, data is controlled by the State of component. | Here, data is controlled by the DOM itself using ref. |
| **3.** | It accepts its current value as a prop. | It uses a ref for their current values. |
| **4.** | It allows validation control. | It does not allow validation control. |
| **5.** | It has better control over the form elements and data. | It has limited control over the form elements and data. |

# **14. React Component Life-Cycle**

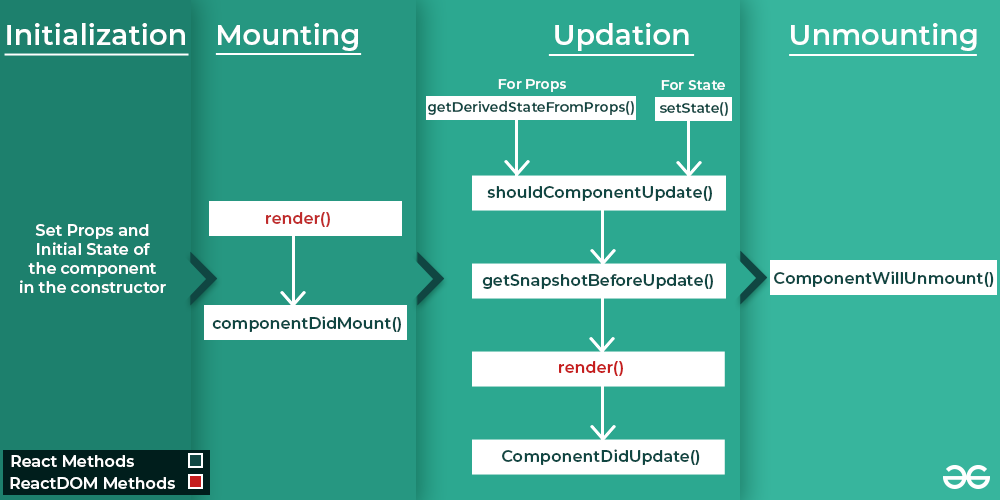
In React, components have a lifecycle that consists of different phases.

Each phase has a set of lifecycle methods that are called at specific points in the component's lifecycle.

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

 The lifecycle of the component is divided into **four phases**. They are:

1. Initial Phase
2. Mounting Phase
3. Updating Phase
4. Unmounting Phase



**Initial Phase**

It is the **birth** phase of the lifecycle of a ReactJS component.

In this phase, a component contains the default Props and initial State.

These default properties are done in the constructor of a component.

The initial phase only occurs once and consists of the following methods.

* **getDefaultProps()**  
  It is used to specify the default value of this.props. It is invoked before the creation of the component or any props from the parent is passed into it.
* **getInitialState()**  
  It is used to specify the default value of this.state. It is invoked before the creation of the component.

## **2. Mounting Phase**

In this phase, the instance of a component is created and inserted into the DOM.

It consists of the following methods.

* **componentWillMount()- Depricated**  
  This is invoked immediately before a component gets rendered into the DOM. In the case, when you call **setState()** inside this method, the component will not **re-render**.
* **componentDidMount()**  
  This is invoked immediately after a component gets rendered and placed on the DOM. Now, you can do any DOM querying operations. It is use at the time of api call.

**Qusetion**

**A: Can we update the state inside the render?**

You can not set state inside render function because it will cause side effect.

What exactly happens is that each time you update state react calls render function, so if you will update state inside render function then it will stuck inside \*\*infinite loop\*\*.

import {Component} from 'react';

class App extends Component {

  constructor(){

    super();

    this.state={

      user:"user1"

    }

  }

  render(){

      this.setState({ user: "user2" })

  return (

    <>

      <h1>welcome</h1>

    </>

  );

  }

}

export default App;

* **render()**  
  This method is defined in each and every class component.

It is responsible for returning a single root **HTML node** element.

If you don't want to render anything, you can return a **null** or **false** value.

If any change has done in component the render methods re-render automatic.

**Question**

**Can we call render method inside render method?**

one class component will have single render function and we cannot call render function inside render function. It will throw syntax error.

## **3. Updating Phase**

* It is the next phase of the lifecycle of a react component.
* Here, we get new **Props** and change **State**.
* The main aim of this phase is to ensure that the component is displaying the latest version of itself.
* Unlike the Birth or Death phase, this phase repeats again and again.

**b. componentDidUpdate()**

It is invoked immediately after the component updating occurs.

In this method, you can put any code inside this which you want to execute once the updating occurs. This method is not invoked for the initial render.

Good for operations that need to happen after a render, like fetching new data based on prop changes.

We always put if condition to update the state inside this otherwise if run in infinite loop which give error.

Note: componentDidUpdate will not be invoked.If shouldComponentUpdate return false.

1. **shouldComponentUpdate()**  
   It is invoked when a component decides any changes/updation to the DOM.

It allows you to control the component's behavior of updating itself.

If this method returns true, the component will update.

Otherwise, the component will skip the updating because it return false.

**Question**

**Which method calls first either componentDidUpdate or shouldComponentupdate**

**4. Unmounting**

This phase occurs when a component is being removed from the DOM.

Invoked just before the component is unmounted and destroyed. This is where you should clean up any resources (e.g., timers, subscriptions).

**componentWillUnmount():**

* **Clear timers**
* **Unsubscribe** from WebSocket, event listeners, or API
* **Abort** pending network requests
* **Cancel animations**

\

**A screen shot of a computer program

Description automatically generated**

**React Hooks**

Hooks are the new feature introduced in the React 16.8 version.

It allows you to use state and other React features without writing a class.

Hooks are the functions which "hook into" React state and lifecycle features from function components.

It does not work inside classes.

## **When to use a Hooks**

If you write a function component, and then you want to add some state to it, previously you do this by converting it to a class. But, now you can do it by using a Hook inside the existing function component.

## **Rules of Hooks**

### 1. Only call Hooks at the top level

### 2. Only call Hooks from React functions

### Example

import React, { useState } from 'react';

import ReactDOM from 'react-dom/client';

function App() {

const [click, setClick] = useState(0); //----------------------> Hooks On top of the function and call inside react function

    // using array destructuring here

    // to assign initial value 0

    // to click and a reference to the function

    // that updates click to setClick

    return (

        <div>

            <p>You clicked {click} times</p>

            <button onClick={() => setClick(click + 1)}>

                Click me

            </button>

        </div>

    );

}

export default App;

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

root.render(

<React.StrictMode>

    <App />

</React.StrictMode>

);

### 1. What is useState hook in React?

It is a hook which is used to create and update the state values.

One use of useState() can only be used to declare one state variable.

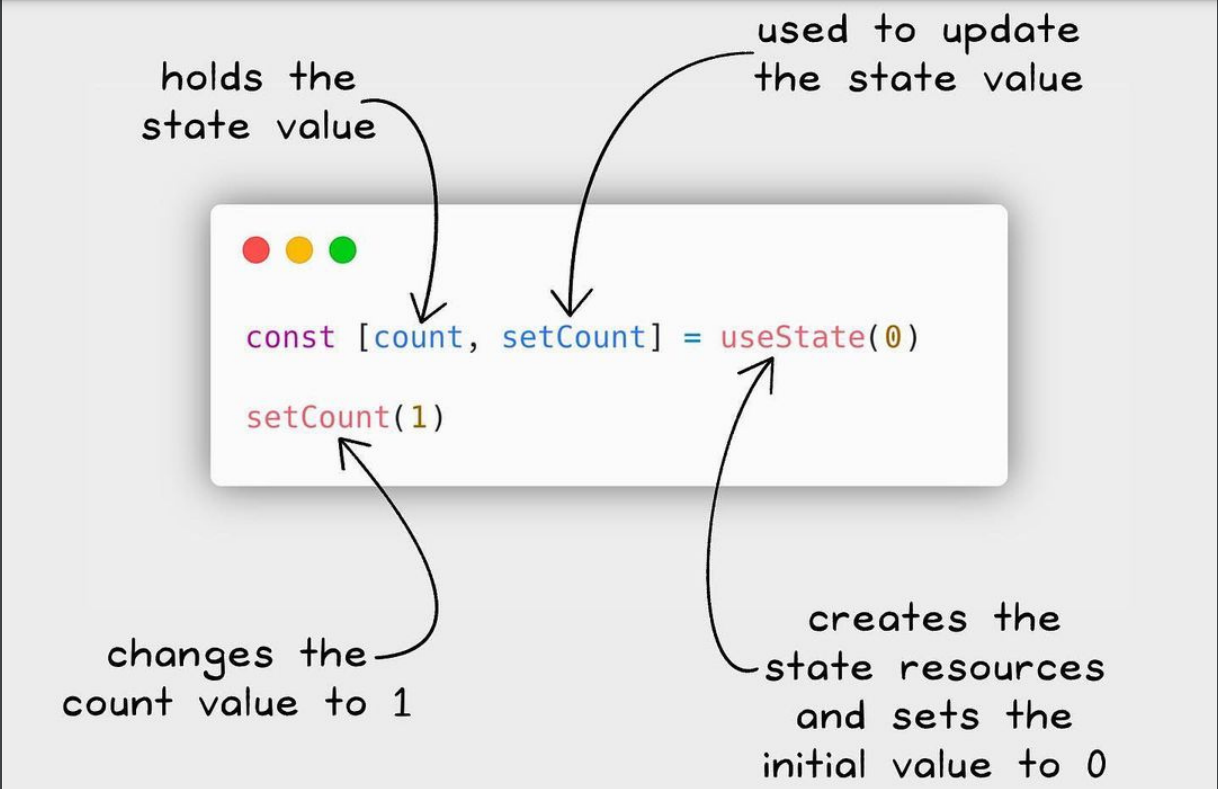
**How to use**

**To use it first Import on the top**

import { useState } from "react";

### Structure of useState hook

* This hooks takes some initial state and returns two value.
* First value contains the state and the second value is a function which updates the state.
* The value passed in useState will be treated as the default value

****

**Example**

import React, { useState } from 'react'; // Declear the useState

function App() {

    const [click, setClick] = useState(0); // Define on the top and set initial value 0

    return (

        <div>

            <p>You clicked {click} times</p> // Show the state value

            <button onClick={() => setClick(click + 1)}>  //Update the state of the hook

                Click me

            </button>

        </div>

    );

}

export default App;

## **2. What is useEffect hook ?**

The useEffect hook in React is used to handle the side effects in functional React Component .

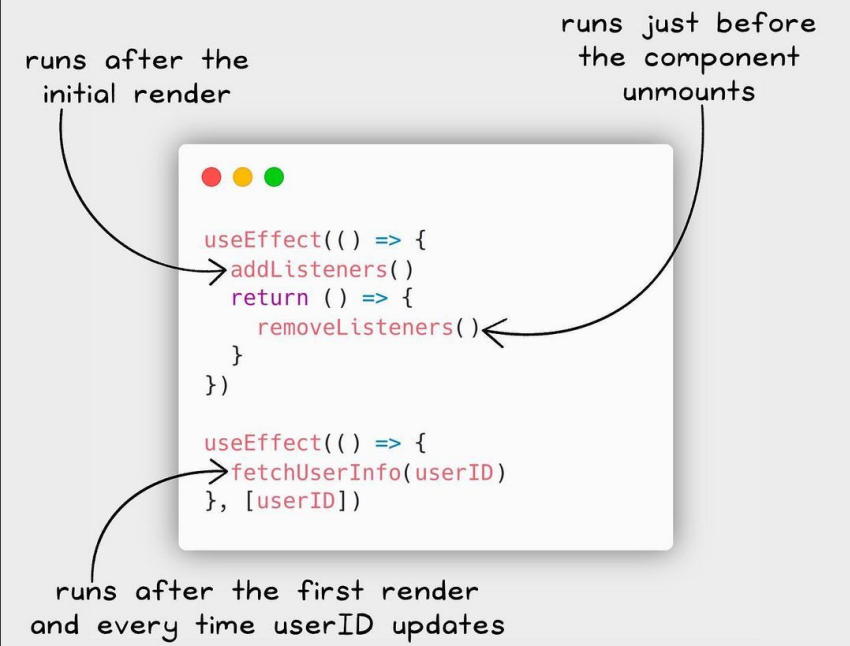
Side effects can include data fetching, subscriptions, or manually changing the DOM.

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

In other words, Effects Hooks are equivalent to componentDidMount(), componentDidUpdate(), and componentWillUnmount() lifecycle methods.

Side effects have common features which the most web applications need to perform, such as:

* Updating the DOM,
* Fetching and consuming data from a server API,
* Setting up a subscription, etc.



To import the useEffect hook, write the following code at the top level of your component

import { useEffect } from "react"

**Example**

import { useState, useEffect } from "react";

function HookCounterOne() {

    const [count, setCount] = useState(0);

    useEffect(() => {

        document.title = `You clicked ${count} times`;

    }, [count]); // [] array dependency

    return (

        <div>

            <button onClick={() => setCount((prevCount) => prevCount + 1)}>

                Click {count} times{" "}

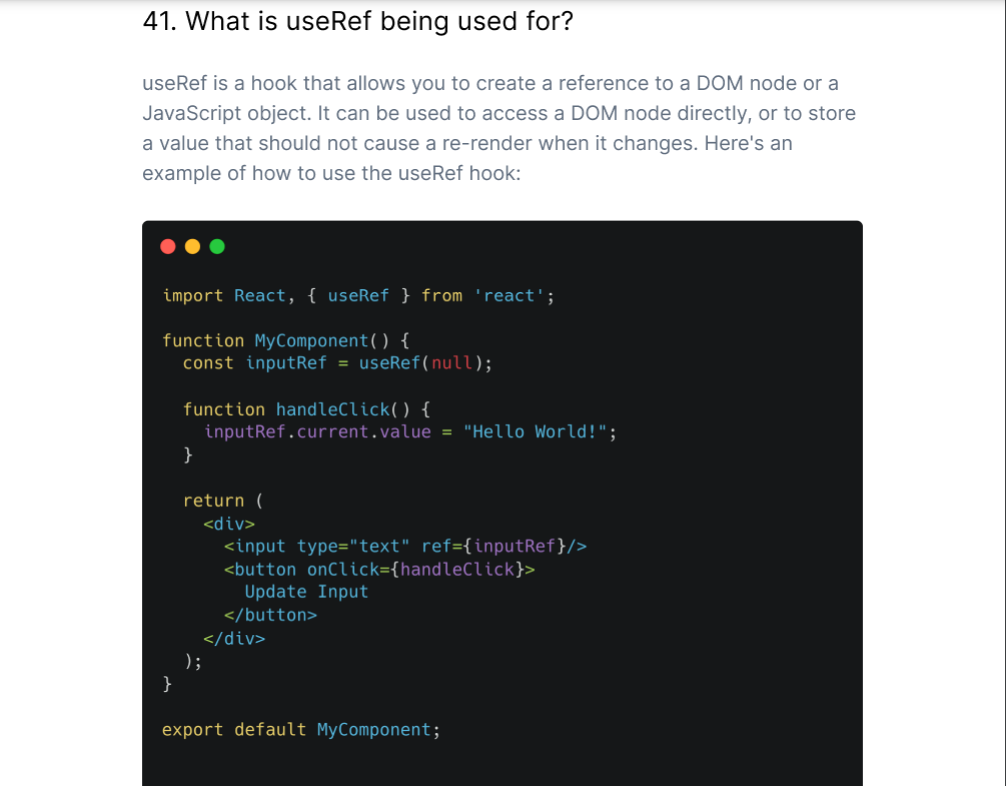
            </button>

        </div>

    );

}

export default HookCounterOne;



**useReducer**

The useReducer Hook is similar to the useState Hook.

It allows for custom state logic

The reducer function contains your custom state logic and the initialState can be a simple value but generally will contain an object.

The useReducer Hook returns the current state and a dispatch method.

import React, { useReducer } from 'react';

// Define the reducer function

const reducer = (state, action) => {

switch (action.type) {

case 'increment':

return { count: state.count + 1 };

case 'decrement':

return { count: state.count - 1 };

case 'reset':

return { count: 0 };

default:

return state;

}

};

const Counter = () => {

// Initial state

const initialState = { count: 0 };

// Use the reducer

const [state, dispatch] = useReducer(reducer, initialState);

return (

<div>

<p>Count: {state.count}</p>

<button onClick={() => dispatch({ type: 'increment' })}>Increment</button>

<button onClick={() => dispatch({ type: 'decrement' })}>Decrement</button>

<button onClick={() => dispatch({ type: 'reset' })}>Reset</button>

</div>

);

};

export default Counter;

**Explanation**

1. **Reducer Function**: The reducer function defines how the state changes based on the action. It takes the current state and an action object, and returns the new state based on the action type.
2. **Initial State**: You define your initial state, in this case, { count: 0 }.
3. **Using useReducer**: The useReducer hook returns the current state and the dispatch function, which you can use to send actions to the reducer.
4. **Dispatching Actions**: You can dispatch actions by calling dispatch with an action object that has a type. This updates the state based on the logic defined in the reducer.

**useMemo Hook in React**

The useMemo Hook returns a memoized value and prevents the application from unnecessary re-renders.

**Why do we need useMemo?**

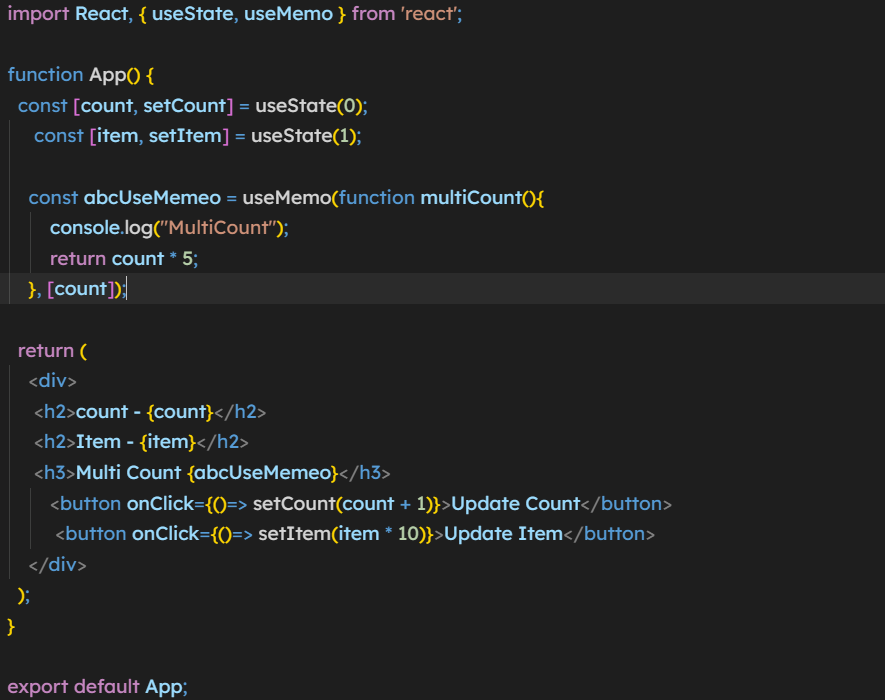
When a component re-renders, all calculations and function calls are executed again. This can lead to performance issues, especially if the calculations are expensive or if the component renders frequently.

**How does useMemo work?**

**useMemo** takes two arguments:

1. A function that returns a value (the computation).
2. An array of dependencies that determines when to recompute the value.

If the dependencies haven't changed since the last render, **useMemo** returns the cached value instead of recalculating it.



Explanation :

In this example if we don’t wrapped the multiCount function in useMemo and call directly on that time it re-render again an again due to this a performance issue comes in application.

But after wrapping it into the useMemo hooks it call only when count calls because I pass count in array of useMemo.

**When to Use useMemo**

1. **Expensive Calculations**: If you have a calculation that takes a significant amount of time, wrapping it in useMemo can prevent unnecessary recalculations.
2. **Referential Equality**: When passing objects or arrays to child components, useMemo can help maintain referential equality, preventing unnecessary re-renders.

**Additional Hooks**

* useReducer
* useCallback
* useMemo
* useImperativeHandle
* useLayoutEffect
* useDebugValue

### What is custom hook?

We know that hooks like useState, useEffect are reusable components.

Somtimes we make components that we have to reuse again and again in the application.

In this case we can convert the component to hooks by using “use” keword.

* So custom hook is nothing but a javascript function.
* The name of custom Hook starts with "use" which can call other Hooks.
* A custom Hook is just like a regular function, and the word "use" in the beginning tells that this function follows the rules of Hooks.

### ****Need for Custom Hooks****

The main reason for which you should be using Custom hooks is to maintain the concept of [DRY](https://www.geeksforgeeks.org/7-common-programming-principles-that-every-developer-must-follow/)(Don’t Repeat Yourself) in your React apps.

For example, suppose you have some logic that makes use of some built-in hooks and you need to use the logic in multiple functional components. So, the easier way to do it is creating a separate function that wraps the logic inside it and then call it from those components.Here, the separate function you created is the custom hook.

import React, { useState, useEffect } from 'react';

const useDocumentTitle = title => {  // Create Custom Hook

  useEffect(() => {

    document.title = title;

  }, [title])

}

function CustomCounter() {

  const [count, setCount] = useState(0);

  const incrementCount = () => setCount(count + 1);

  useDocumentTitle(`You clicked ${count} times`);

  // useEffect(() => {

  //   document.title = `You clicked ${count} times`

  // });

  return (

    <div>

      <p>You clicked {count} times</p>

      <button onClick={incrementCount}>Click me</button>

    </div>

  )

}

export default CustomCounter;

## [**Higher-Order Function**](https://www.geeksforgeeks.org/higher-order-functions-currying/)

A higher-order component (HOC) is a function that takes a component as an argument and returns a new component.

Basically, it's a pattern that is derived from React's compositional nature.

We call them **pure components** because they can accept any dynamically provided child component but they won't modify or copy any behavior from their input components.

HOC can be used for many use cases:

1. **Code reuse**: HOCs enable you to reuse code across multiple components without duplicating logic.
2. **Decoupling**: HOCs help decouple components from specific functionality, making it easier to test and maintain individual components.
3. **Abstraction**: HOCs provide a way to abstract away complex logic and expose a simpler interface to components.
4. Render hijacking.
5. State abstraction and manipulation.
6. Props manipulation.

import './App.css';

import React, { useRef, useState } from 'react'

function App() {

  return (

    <div className="App">

      <h1>HOC </h1>

      <HOCRed cmp={Counter} /> // take another component

      <HOCGreen cmp={Counter} />

      <HOCBlue cmp={Counter} />

    </div>

  );

}

// Creating high order function

function HOCRed(props)

{

// Return new component

  return <h2 style={{backgroundColor:'red',width:100}}>Red<props.cmp /></h2>

}

function HOCGreen(props)

{

  return <h2 style={{backgroundColor:'green',width:100}}>Grren<props.cmp /></h2>

}

function HOCBlue(props)

{

  return <h2 style={{backgroundColor:'blue',width:100}}>blue <props.cmp /></h2>

}

function Counter()

{

  const [count,setCount]=useState(0)

  return<div>

    <h3>{count}</h3>

    <button onClick={()=>setCount(count+1)}>Update</button>

  </div>

}

export default App;

## **Context API**

#### **1. What is Context API?**

The Context API is a feature in React that allows you to share state and props across the entire component tree without having to pass them explicitly through every level of the component hierarchy.

It provides a way to manage global state and helps avoid "prop drilling," where props are passed down through many layers of components.

Context API is a React built-in, and is a way to manage state that is global to the application.

### 2.Why is context API used?

Context API solves the problem of prop drilling in React.

Prop Drilling occurs when data is to be passed between multiple layers before finally sending it to the required component. This makes the application slower.

This problem is solved by Context API as it creates global variables to be used throughout the application without any middle components involved.It is also easier to use than React Redux

### ****3. Working of Context API****

* To work with Context API we need React.createContext.
* It has two properties Provider and Consumer.
* The Provider component provides the context to the Consumer component.
* The Consumer component then uses that context to render the appropriate UI.

### 4. Benefits of Context API over React Redux

* In Redux we have to manipulate or update multiple files to add even a single feature but in Context it can be done in much lesser lines of code
* One way data binding in React is maintained using Context whereas Redux violates it.
* Multiple stores/contexts can be created using Context whereas Redux creates just a single store

#### **5. When should you use Reducers vs Context API?**

Reducers should be used when you need to manage complex state logic, while Context API is best used for simple state management.

#### **6. What is the process used by Context API to update values in a context object?**

The process used by Context API to update values in a context object is called “reconciliation.”

This process works by taking the new values that have been set for a context object and compare them to the old values. If there are any differences, the Context API will then update the context object accordingly.

#### **7. What languages can be used to write Context API components?**

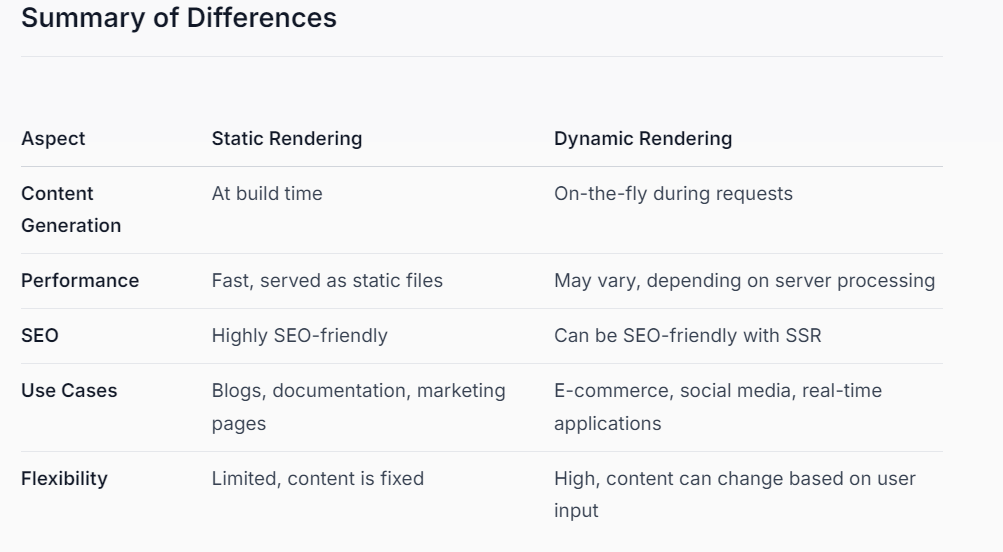
Context API components can be written in any language that can be compiled to JavaScript. This includes languages like TypeScript, Babel, and Flow.

#### **8. What’s the difference between static and dynamic rendering?**

**Definition**: Static rendering involves generating HTML content at build time. The resulting HTML files are served directly to the user without any server-side processing during a request.

#### **Definition**: Dynamic rendering generates HTML content on-the-fly, typically in response to user interactions or requests. This can involve server-side rendering (SSR) or client-side rendering (CSR).

On-the-fly (it means that the content is created and sent to the browser in real-time, rather than being pre-generated and served as static HTML.)



#### **9. What happens if we try to read data from a Context object before initializing it?**

If you try to read data from a Context object before initializing it, you will get an error. This is because the Context object needs to be initialized with data before it can be used.

SearchContext.js

import { createContext, useState } from "react";

export const SearchContext = createContext(""); //Create the context

export const SearchContextProvider = ({children})=>{   //Create the Provider

    const [searchVal,setSearchVal] = useState("");

    console.log("SearchVal=>",searchVal);

    return (

        <SearchContext.Provider value={{searchVal,setSearchVal}}>     // send the value

            {children}

        </SearchContext.Provider>

    )

}

**ProductMain.jsx**

import { useContext } from 'react';    // import useContext

import { SearchContext } from '../../context/SearchContext.js';      // import the createContext variable name

const ProductsPageMain = () => {

    const {searchVal} = useContext(SearchContext);       // use the context and get value

}

# **React Router**

#### **1. What is React Router?**

React Router is a popular library for managing routing in React applications.

It enables developers to create single-page applications (SPAs) with navigation capabilities, allowing users to move between different views or components without refreshing the entire page.

**How to Install React Router Dom liberary?**

npm install react-router-dom

**There are 3 terms which we use in router**

**1: BrowserRouter**

* A high-level component that wraps your application and enables routing functionality.
* It helps to connect our application with browser URL.

**2: Router**

* Routes are chosen based on the best match instead of being traversed in order.

**3:** **Route**

* Route is the conditionally shown component that renders some UI when its path matches the current URL.

**2. Client Side Routing**

Client-side routing in React refers to the process of managing the application's URL and rendering the corresponding components or views without requiring a full page reload.

This is achieved by using a router library or component that updates the URL in the browser's address bar and renders the new component.

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.

Client side routing is enabled by creating a Router and linking/submitting to pages with Link and <Form>:

****

**In line 16 we use Link with the help of this we can do client side rendering.**

**3.** **Nested routing**

Nested routing allows you to render **sub-routes** in your application. It can be understood in the below example.

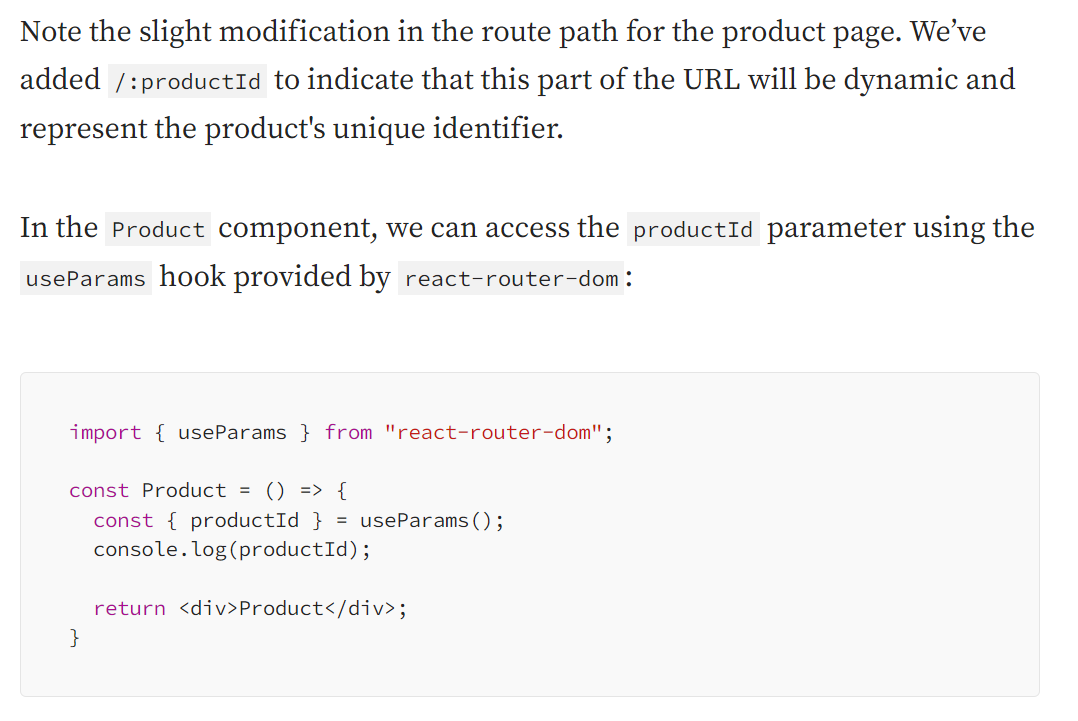
#### **4. What is nested routing?**

* Nested routing is when you have routes within other routes.
* This can be useful if you want to create a hierarchy of pages, for example if you have a blog with different categories and you want each category to have its own page with a list of posts.
* To do this, you would create a route for each category, and then nest the route for the posts within that.
* Nested routing is useful when you need to render multiple components or views within a single route.

**5. Dynamic Routing / URL**

Dynamic routing is a powerful technique used in react to handle navigation and rendering of pages based on changing parameters.

****

****

1. **Difference b/w Link, NavLink and Outlet ?**

**Link:**

* Purpose: The Link component is used to create navigational links to different routes in your application.
* Usage: It allows users to navigate to different URLs without reloading the page.

When we use **Link**, we can think of it as the React version of the **<a>** tag in HTML, which allows us to create links to other pages.

**Link** receives a **to** attribute which is the URL to which the link refers.

This attribute can be a string representing a route, or an object containing a route and state.

**A computer screen with text on it

Description automatically generated**

**NavLink:**

* Purpose: The NavLink component is similar to Link but provides additional functionality for styling active links.
* Usage: It allows you to apply styles or classes to links based on whether they are active (i.e., the current route matches the link).

For example, with NavLink, we have the ability to know whether our link is in an "active" or "pending" state.

We can adapt these states to our needs in order to display the link status differently. For example, we can change our CSS styles according to the link status.

****

Also, we can change our CSS styles according to the link state by using the **style** attribute.

For example:

****

**OUTLET**

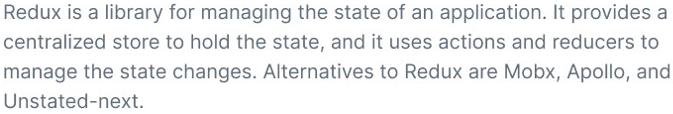
* **Purpose**: The **Outlet** component is used for rendering child routes in nested routing.
* **Usage**: It acts as a placeholder for rendering the components of child routes defined in a parent route.

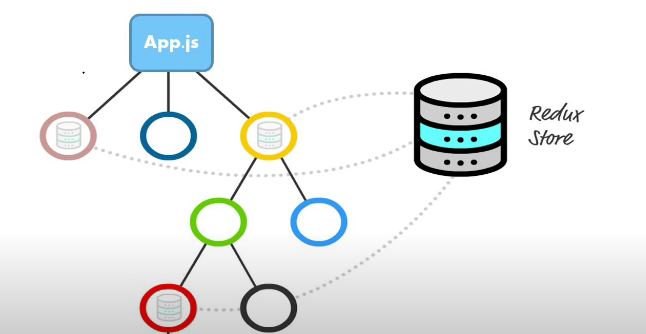
A computer screen shot of a black background

Description automatically generated

**REDUX**

**1. What is Redux?**

****

****

**2. What are the main component of redux?**

**There are 3 main component in redux**

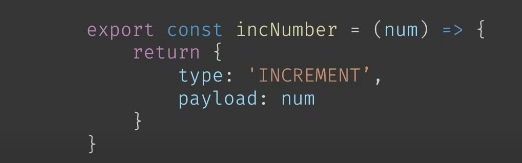
**a. Action:**

****

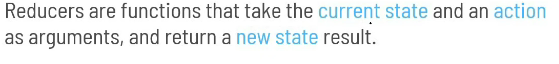
**Who Creates the Action?**

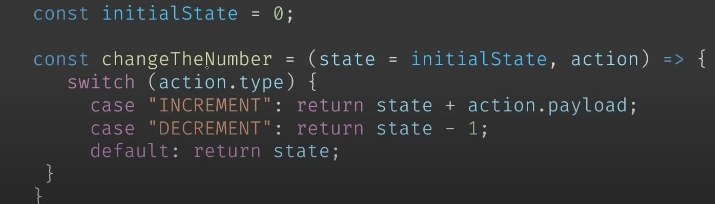
It is create by Pure Funciton.

It is reusable, portable and easy to test.

****

**b. Reducer**

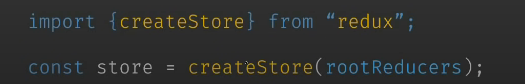
****

****

**c. Store**

 The Redux store is a centralized store that holds the entire state of a web application.

Every Redux store has a single root reducer.

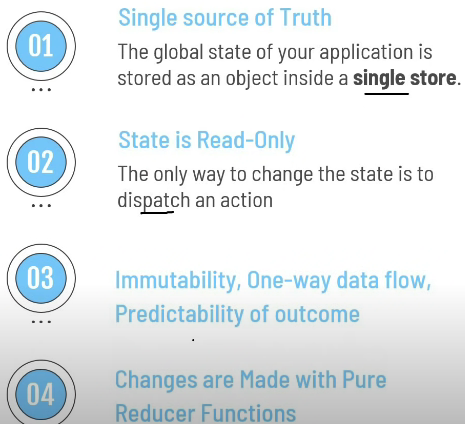
****

**3. What is Dispatch in redux?**

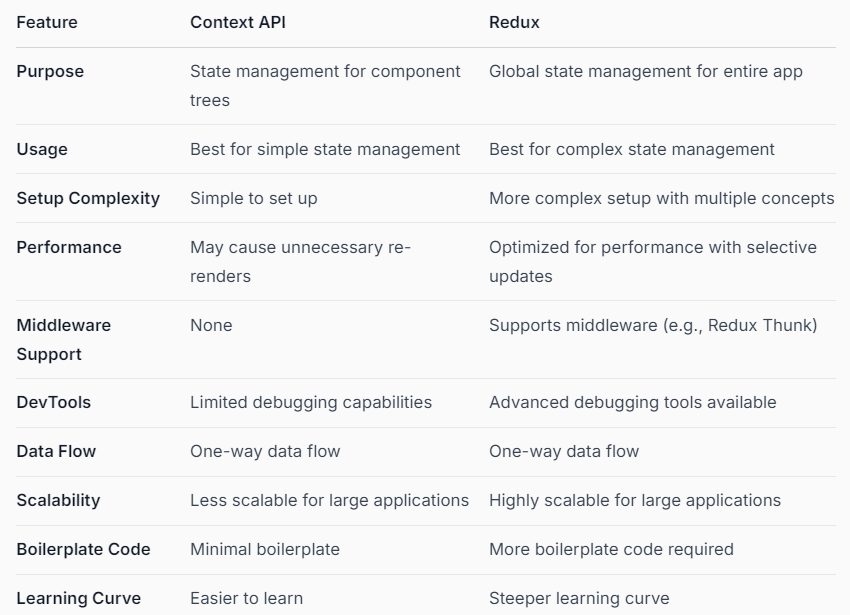
The dispatch function accepts an object that represents the type of action we want to execute when it is called.

Basically, it sends the type of action to the reducer function to perform its job, which, of course, is updating the state.

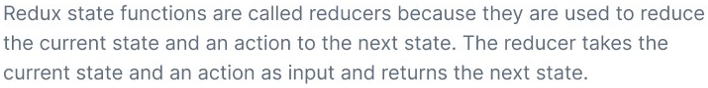
**4. Principle Of Redux**

****

**5. Difference between Redux and Context API?**

****

**6. Why Redux state function call Reducer?**

****

**Install Redux and React-Redux**

npm install redux react-redux

**7. What is Saga?**

Redux-Saga is a library that allows you to handle side effects in a Redux application by using generator function.

It handles async data in redux like APIs Data.

We can call it a middleware.

Here we can write Async operation like time taken logic and Api calls.

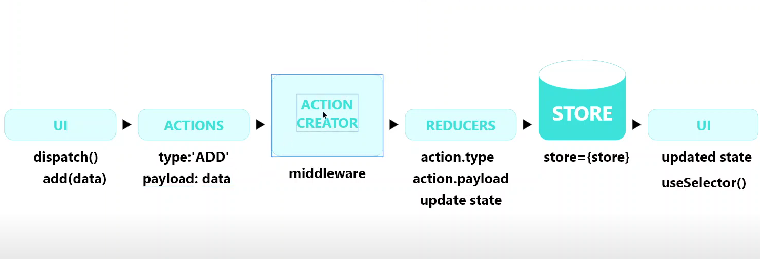
**8. What is Thunk?**

It is a middleware that allow you to return a function , rather than a action, within Redux.

The purpose of middleware is to intercept an action before reaches to reducer.

Means, instead of sending action to reducer we send first to Action creator which is our middleware it return a function not action.

The middleware can do any think with action but it can’t change the state value.



**Redux Toolkit**

**1. What is Redux ToolKit? And why we use it instead of using redux?**

Redux Toolkit is used for writing redux code but in a more concise way. Redux Toolkit (RTK) solves problems that most of the developer’s face who used redux in a react application.

RTK abstracts the basic redux code and provides us boilerplates that enable us to write redux code in less lines of code.

**Issues with basic Redux:**

* Configuring a Redux store is too complicated
* Have to add a lot of packages to build a large-scale application
* Redux requires too much boilerplate code which makes it cumbersome to write efficient and clean code.

**Reasons for preferring RTK(Redux ToolKit):**

* Redux ToolKit solves various issues by providing a hook-based implementation of Redux
* RTK gives the ability to write mutable state updates in the reducers.
* It also eliminates the use of extra coding by providing boilerplates.
* RTK also has the feature of RTK query which eliminates the use of Thunks and makes the query processing faster
* It also provides automatic support for Redux Dev-tools Extension and for immer.js library which is a great tool to deal with immutable objects.
* You can also use the various predefined functions of Redux Toolkit which not only speeds up the process but also saves time.

**Dependencies Included with RTK(Redux ToolKit):**

* immer
* redux
* redux-thunk
* reselect

#### **To install RTK(Redux ToolKit) in the existing project use the command**

# NPM  
npm install @reduxjs/toolkit

#### **To create a new project with RTK use the command:**

npx create-react-app my-app --template redux

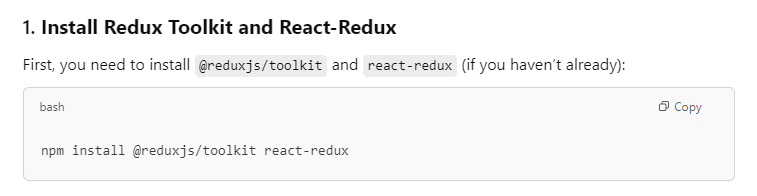
**Important features provided with RTK:**

* Automatic support for Redux-thunk, Redux DevTools Extension, and default middleware is provided by **configureStore()**function
* Support for the immer library that allows writing the immutable code mutably is provided by **createReducer()** utility.
* create action and create Reducer functions are replaced with a single function called **createSlice()** function.
* **createAsyncThunk()** that takes Redux strings as arguments and returns a Promise.

#### **Comparing Store and Reducers**

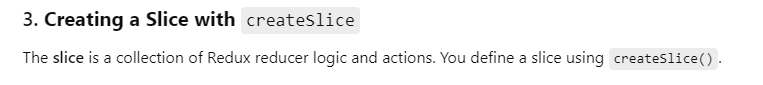
|  |  |
| --- | --- |
| **Redux Reducer** | **RTK Reducer** |
| const addHandler=(state=0,action)=>{  **if**(action.type==='ADD')  {  **return** state+1;  }  **return** state;  }    const store = createStore(addHandler);    store.dispatch({type:'ADD'}); | const add = createAction('ADD');    const addHandler = createReducer(0, {    [add]: state => state + 1  })    const store = configureStore({    reducer: addHandler  })    store.dispatch(add()); |

**Step-by-Step Setup of Redux Toolkit in a React Application**

****

**A screenshot of a computer program

Description automatically generated**

****

**A screen shot of a computer program

Description automatically generated**

**Extra Questions**

**What is React Fragment?**

In React, fragments is a way to group multiple elements without adding extra nodes to the DOM.

They allow you to return multiple elements from a component without wrapping them in a div or any other HTML element, which can help keep the DOM tree clean and improve performance.

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**What are the synthetic events in react?**

In React, synthetic events are a cross-browser wrapper around the native events.

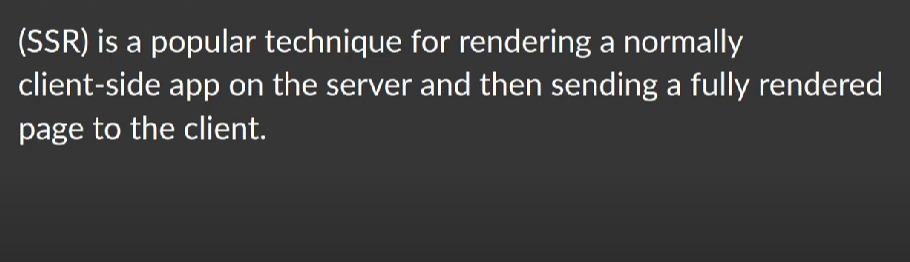
They provide a consistent interface for handling events in React applications, ensuring that events behave the same way across different browsers.

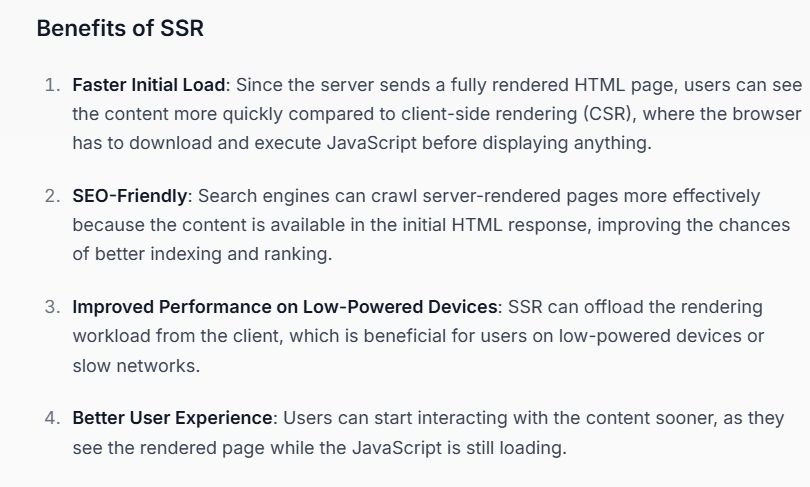
**Key Points about Synthetic Events:**

* Unified Interface: When you write code to handle events, you want it to work the same way for all users, regardless of their browser. Synthetic events ensure that you don't have to change your code for different browsers.
* Event Properties: Synthetic events have the same properties as native events, such as target, currentTarget, preventDefault(), and stopPropagation(). This allows developers to use familiar event handling patterns



**Server Side Rendering**

****

****

**A black and orange text

Description automatically generated**

**Q. What is ReactDOM?**

ReactDOM is a package that provides methods to interact with the DOM (Document Object Model) in a web browser.

It is a critical part of the React library because it acts as the bridge between React components (written in JavaScript) and the browser's DOM (the structure of HTML elements).

