# What is React?

1. React is one of the most powerful, widely used frontend frameworks.
2. https://reactjs.org/docs/react-api.html#reactmemo

# Why React

1. Reusability of components
2. Well supported documents due to its popularity
3. React is not opinionated, which means that it won’t force you to follow any specific design patterns, project organizational structure, or logic. It’s all up to you.
4. Smaller learning curve, especially when you already have a good grasp of JavaScript and HTML from our previous lessons.

# React Life Cycle

1. **For componentDidMount**

useEffect(() => {

// Your code here

}, []);

1. **For componentDidUpdate**

useEffect(() => {

// Your code here

}, [yourDependency]

);

1. **For componentWillUnmount**

useEffect(() => {

// componentWillUnmount

return () => {

// Your code here

}

}, [yourDependency]

);

# Context:

* React Context is a way to manage state globally.
  + 1. Const MyContext = React.CreateContext()
    2. <MyContext.Provider value = {user}> </MyContext.Provider>
    3. Const user = React.useContext(MyContext)
* React context allows us to pass down and use (consume) data in whatever component we need in our React app without using props.
* Context provides a way to pass data through the component tree without having to pass props down manually at every level.

# Fragment

render only takes one component as an argument so to bind multiple components without any extra tags we use fragment

# Controlled and Uncontrolled Component

1. This relates to stateful DOM components (form elements) and the React docs explain the difference:
2. A Controlled Component is one that takes its current value through props and notifies changes through callbacks like onChange. A parent component "controls" it by handling the callback and managing its own state and passing the new values as props to the controlled component. You could also call this a "dumb component".
3. An Uncontrolled Component is one that stores its own state internally, and you query the DOM using a ref to find its current value when you need it. This is a bit more like traditional HTML.

# What is react-router?

1. import {useHistory} from ‘react-router’
2. const history = useHistory()
3. history.push(“/error”)

# Redux

# React.Memo

* React.memo is a higher order component.

# shouldComponentUpdate() in functional component

* function MyComponent(){

returns <div>Hello World</div>

}

function isEqual (prevProps,nextprops){

// return true if no need to render again

// return false for re-render

Export default React.memo(MyComponent,isEqual)

}

# Advantages of functional component over class component

1. Easier understanding
2. Class components are converted in functional components by react so it is reducing a step to convert so it much more faster

# useEffect

* useEffect(()=>{},[]) = > this will work as componentdidmount
* useEffect(()=>{},) = > hit in useEffect evertime there is a update

# useContext with react-router

1. <https://dev.to/ilizette/understanding-usecontext-in-react-26gf>

const ThemeContext = React.createContext(themes.light);

function App() {

return (

<ThemeContext.Provider value={themes.dark}>

<Toolbar />

</ThemeContext.Provider>

);

}

function Toolbar(props) {

return (

<div>

<ThemedButton />

</div>

);

}

function ThemedButton() {

const theme = useContext(ThemeContext);

return (

<button style={{ background: theme.background, color: theme.foreground }}>

I am styled by theme context!

</button>

);

}

# Pure Functions in React

* export default React.memo(MyComponent)

# Pure Components

1. A React component is considered pure if it renders the same output for the same state and props. For this type of class component, React provides the PureComponent base class. Class components that extend the React.PureComponent class are treated as pure components.
2. 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;

# Virtual DOM

1. React uses Virtual DOM exists which is like a lightweight copy of the actual DOM(a virtual representation of the DOM).
2. How Virtual DOM actually make things faster: When anything new is added to the application, a virtual DOM is created and it is represented as a tree. Each element in the application is a node in this tree. So, whenever there is a change in the state of any element, a new Virtual DOM tree is created. This new Virtual DOM tree is then compared with the previous Virtual DOM tree and make a note of the changes. After this, it finds the best possible ways to make these changes to the real DOM. Now only the updated elements will get rendered on the page again.

# DOM:

* DOM is an API for the HTML, and we use languages like JavaScript or frameworks like React, Vue.js to access and manipulate the HTML using the corresponding DOM objects.

# Lazy loading in react

1. https://www.loginradius.com/blog/engineering/lazy-loading-in-react/#:~:text=In%20essence%2C%20lazy%20loading%20means,it%20at%20the%20same%20time.
2. lazy loading means that a component or a part of code must get loaded when it is required. It is also referred to as code splitting and data fetching.
3. In situations where we know that certain code/features will not be accessible to all the users or the user does not access it frequently, it is best to load them when the user requests for it. This improves user experience as well as initial loading time.
4. import React, { Suspense } from "react";
5. const Customer = React.lazy(() => import("./Customer.js"));
6. const Admin = React.lazy(() => import("./Admin.js"));
7. //Instead of regular import statements, we will use the above approach for lazy loading
8. export default (props) => {
9. if (props.user === "admin") {
10. return (
11. // fallback component is rendered until our main component is loaded
12. <Suspense fallback={<div>Loading</div>}>
13. <Admin />
14. </Suspense>
15. );
16. } else if (props.user === "customer") {
17. return (
18. <Suspense fallback={<div>Loading</div>}>
19. <Customer />
20. </Suspense>
21. );
22. } else {
23. return <div> Invalid User </div>;
24. }
25. };

# What is SPA?

1. A SPA application is a single page that continuously interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server.
2. When you send a request to visit a web page, the browser sends a request to the server and gets an HTML file in return. With a SPA, the server only sends an HTML file on the first request; it sends data known as JSON on subsequent requests.
3. Quick Loading Time
4. Seamless User Experience
5. Ease in Building Feature-rich Apps
6. Uses Less Bandwidth

# When to choose functional components and class components?

1. If the project is bit more complex, if we want more control over the state management, I will choose class-based components
2. There are not exact functional hooks for every class component life cycle.
3. I would choose class-based component if I had to implement redux.
4. I will choose functional based component if the project is less complex.

# When do we use SPA and Multiple Page Application?

# Why should we use React?

// Advantages of React over other spa applications

1. Learning curve is much easier for new/junior developers if JS is already known
2. React is flexible, component-based and view only.
3. Angular is fixed and complex, component-based and MVC

# What is SSR?

## Used for SSR

1. SEO – Search Engine Optimization
   1. Search Engine Crawlers doesn’t understand/render Javascript yet
2. Improved performance
   1. The application performance depends on the server’s resources and user’s network speed

## Cons of Rendering React on the Server

1. SSR can improve performance if your application is small. But it can also degrade performance if it is heavy.
2. It increases response time (and it can be worse if the server is busy).
3. It increases response size, which means the page takes longer to load.
4. It increases the complexity of the application.

## How does it Work? — (4 Simple Steps)

1. Create a fresh Redux Store on every request.
2. Optionally dispatch some actions.
3. Get the state out of the Store and perform SSR.
4. Send the state obtained in the previous step along with the response.

# Hooks

# Custom Hooks

import React from "react";

import copy from "copy-to-clipboard";

export default function useCopyToClipboard(resetInterval = null) {

const [isCopied, setCopied] = React.useState(false);

function handleCopy(text) {

if (typeof text === "string" || typeof text == "number") {

copy(text.toString());

setCopied(true);

} else {

setCopied(false);

console.error(

`Cannot copy typeof ${typeof text} to clipboard, must be a string or number.`

);

}

}

return [isCopied, handleCopy];

}

import React from "react";

import ClipboardIcon from "../svg/ClipboardIcon";

import SuccessIcon from "../svg/SuccessIcon";

import useCopyToClipboard from "../utils/useCopyToClipboard";

function CopyButton({ code }) {

const [isCopied, handleCopy] = useCopyToClipboard();

return (

<button onClick={() => handleCopy(code)}>

{isCopied ? <SuccessIcon /> : <ClipboardIcon />}

</button>

);

}

# useRef

# The useRef Hook allows you to persist values between renders.

# It can be used to store a mutable value that does not cause a re-render when updated.

# It can be used to access a DOM element directly.

# HOC

# What is render prop

# Forms ReactJs

# What is doctype in html?

# Middleware. Importance of middleware in redux

# Redux thunk use case

# Jest – Testing React