1. What is jsx?

Java script xml extension. Is a java script function return normal html.

const root = ReactDOM.createRoot(

document.getElementById('root') as HTMLElement

);

root.render(<App/>);

{} -> it tells javaScript expression. Boolean will not render.

1. CSS files as module?

File name with “App.module.css” .

.name{

color:”red”

}

Import styles from “./App.module.css”;

<div className={style.name}></div>

const isGreen = false;

style={{color:isGreen ? ”green”:”red”}}

style={{backgroundColor:’red’}}

1. List and Keys rendering

Key ={key} we need to use in iteration.

<ul>

{

data.map((res)=> (<li key={res.id}>{res.name}</li>))

}

</ul>

react-icons

1. Functional components

Rfce -> functional components to create.

Function Main (){

return(

<>

<h4>Hello world</h4

</>

)

}

export default Main;

1. React Events

Const handleClick = (eve)=>{

}

onDoubleClick

<button onClick={ handleClick }></button>

1. Reactjs Props and prop Drilling

If props are not coming from parent we can set default props.

Component\_Name.defaultProps = {

title:’default title’

}

1. Reactjs Forms

<form>

<label htmlFor=”addItem”>Title</label>

<input autofocus id=” addItem” onChange={(e)=> setIteam(e.target.value)}>

</form>

1. React useState

Const [name,setName] = useState(‘name’); // name and function.

setName(‘test1’);

{name} // test1

1. Reactjs useEffect Hook

useEffect is asynchronous.

useEffect(()=>{

// render always for every change occur.

});

useEffect(()=>{

// render once at installation (load) time.

},[])

useEffect(()=>{

// render if there is dependency.

},[dependency])

useEffect(()=>{

return () => {

connection.disconnect(); to unsubscribe.

};

},[])

1. Fetch Data from API in React Js

useEffect(()=>{

const fetchItems= async ()=>{

try{

const response = await fetch(api\_url);

if(!response.ok) throw Error(‘Did not receive data’);

const listItems = await response.json();

// here we need to set items in ‘state’.

} catch(error){

Console.log(error);

} finally {

Console.log(‘done’);

}

})

(async ()=> await fetchItems())();

},[]);

1. React Routing

npm I react-router-dom -S

index.js

import {BrowserRouter as Router, Route} from ‘react-router-dom’;

<Router>

<Route path=”/” component={App} />

</Router>

App.js

Import Header, Footer, Home, Nav

Import {Route, Switch, useHistory} from ‘react-router-dom’;

<div>

<Header />

<nav />

<Switch>

<Route exact path=”/” component={Home} />

<Route exact path=”/posts” component={NewPost} />

<Route exact path=”/posts/:id” component={PostPage} />

<Route exact path=”/about” component={About} />

<Route path=”\*” component={Missing} />

</Switch>

</div>

Import {link} from ‘react-router-dom’;

<ul>

<li> <Link to=”/”>Home</Link></li>

<li> <Link to=”/posts”>Post</Link></li>

<li> <Link to={`/post/${post.id}}`>Single Post</Link></li>

</ul>

Import {Route, Switch, useHistory, useParams} from ‘react-router-dom’;

const {id} = useParams();

const history = useHistory();

history.push(‘/’);

1. Reactjs Axios Api Request

npm install axios -S.

api.js:-

import axios from ‘axios’;

export default axios.create({

baseUrl: ‘url’

});

1. Custom hooks

useWindow.js

import {useState, useEffect} from ‘react’;

const useWindowSize = () =>{

const [windowSize, setWindowSize] = useState({

width:undefined,

height:undefined

})

useEffect (()=>{

const handleResize = () =>{

setWindowSize({

width:window.innerWidth,

height:window.innerHeigth

})

}

handleResize();

window.addEventListener(‘resize’, handleResize);

return ()=> {

window.removeEventListener(“resize”, handleResize);

}

},[])

return windowSize;

}

Const {width, height} = useWindowSize();

1. Context Api

DataContext.js

Import {createContext, useState, useEffect} from “react”;

Const DataContext = createContect({});

export const DataProvider = ({childern}) =>{

return{

<DataContext.Provider value ={{width:200}}>

{{childern}}

</DataContext.Provider>

}

}

In app.js

<div>

< DataProvider>

// here routing will come.

< / DataProvider>

</div>

In any component

Import {useContect} form ‘react’;

Import DataContext from ‘file path’;

Const {width} = useContext(DataContext);

1. useCallback

It returns memorise function it will not call function.

Const [num1] = useState(4);

Const [num2] = useState(5);

Const sum = useCallback(()=> num1 + num2, [num1,num2]);

Const [result, setResult] = useState(0);

useEffect(()=>{

console.log(‘sum is’, sum);

setResult(sum()); // with out useCallback it calls multiple times.

},[sum])

1. useMemo

when ever we update state it re-render component.

Returns memorizes value from function.

Const fib = useMemo(()=> function(val for state), [val]);

useEffect(()=>{

console.log(‘sum is’, sum);

setResult(sum()); // with out useCallback it calls multiple times.

},[sum])

1. useRef

we can use more than one useref. value stays same every render. It not rerender.

Like set focus.

const renders = useRef(0);

render.current++;

{render.current}

1. useReducer

without useState hook we can manage our state using useReducer.

Here we use ‘dispatch’.

Const [state, dispatch] = useReducer(reducer, {count:0});

Const reducer = (state, action) =>{

Switch (action.type){

Case ‘increment’:

Return {count: state.count + 1}

Case ‘decrement’:

Return {count: state.count - 1}

Default:

Throw new Error();

}

}

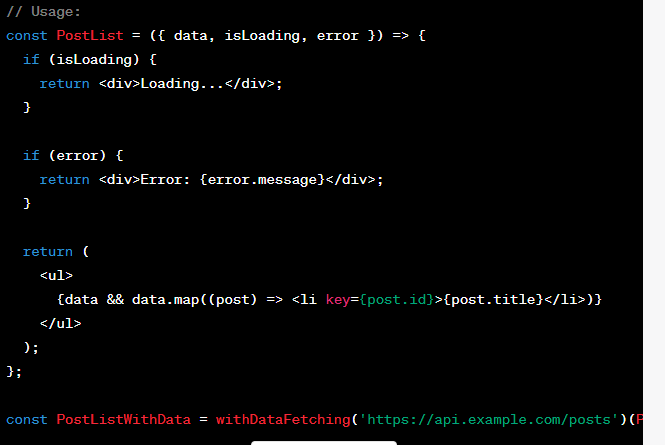
In event we use. dispatch({type:’increment’});

1. What is ReactJS? How does it differ from other JavaScript frameworks?
   * ReactJS is a JavaScript library used for building user interfaces. It differs from other frameworks by introducing a virtual DOM, which allows efficient updates and rendering of components. React also promotes a component-based architecture, making it easy to reuse and maintain UI elements.

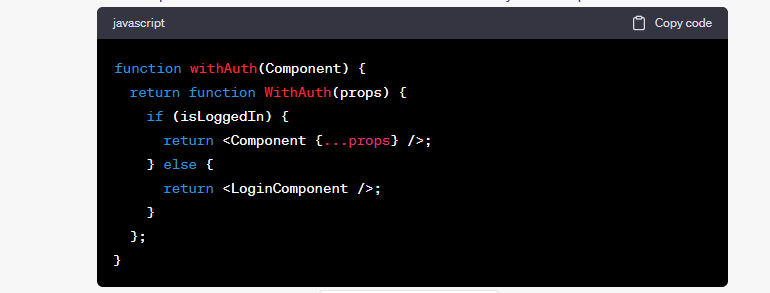
Key features of ReactJS include:

* Virtual DOM for efficient rendering.
* Component-based architecture for reusability.
* Unidirectional data flow with props and state.
* React hooks for managing state and side effects.
* JSX syntax for writing HTML-like code in JavaScript.
* React Router for client-side routing.
* React Native for building mobile applications.

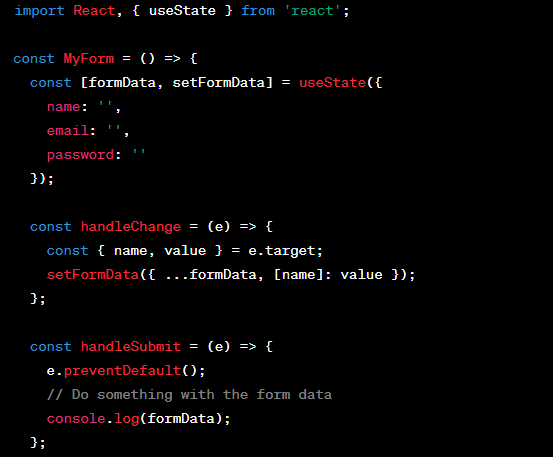
1. Explain the concept of Virtual DOM in React and how it helps in performance optimization.
   * Virtual DOM is a lightweight representation of the actual DOM in memory. When changes occur in the component's state, react calculates the difference between the previous and updated virtual DOM and applies only the necessary updates to the actual DOM. This approach minimizes direct manipulations of the DOM, resulting in improved performance.
2. How does React handle component communication?
   * React handles component communication through props and callbacks. Props allow data to be passed from parent to child components, while callbacks enable child components to communicate with their parent components by invoking functions passed as props.
3. What are React hooks? Explain some commonly used hooks.
   * React hooks are functions introduced in React 16.8 to allow functional components to have state and use lifecycle methods. Some commonly used hooks are:
     1. useState: Allows functional components to manage state.
     2. useEffect: Handles side effects and lifecycle methods.
     3. useContext: Accesses the context in functional components.
     4. useRef: Provides a mutable ref object for storing values across renders.
     5. useCallback: Returns a memoized callback function for performance optimization.
4. How would you optimize a React application for better performance?
   * Here are some techniques to optimize a React application's performance:
     1. Implement code splitting and lazy loading to reduce initial bundle size and load only the required components.
     2. Use React.memo or shouldComponentUpdate to prevent unnecessary re-renders.
     3. Optimize the use of React hooks to minimize unnecessary state updates.
     4. Use the production build of React to enable additional performance optimizations.
     5. Implement memoization techniques like memo or useMemo to cache expensive calculations.
     6. Implement virtualization techniques like windowing or infinite scrolling for efficient rendering of large lists.
     7. Use a performance profiling tool like React Profiler or Chrome DevTools to identify and optimize performance bottlenecks.
5. Explain the concept of state in React and the difference between state and props.
   * State in React represents the internal data of a component. It is mutable and managed by the component itself. Changes to state trigger re-rendering of the component.
   * Props, on the other hand, are immutable and passed from parent to child components. They provide a way for components to receive data and communicate with each other. Props cannot be modified by the receiving component.
6. What are higher-order components (HOCs) in React? Provide an example of their usage.
   * Higher-order components (HOCs) are functions that take a component as input and return a new component with additional functionality. They allow code reuse and enable cross-cutting concerns such as authentication, logging, or data fetching to be applied to multiple components.
   * Example: An HOC that adds authentication functionality to a component:







1. How would you handle forms in React? Explain controlled and uncontrolled components.
   * Controlled components Controlled components are React components in which form data is handled by the component's state. The component's state is responsible for storing and updating the form values.
   * Uncontrolled components Uncontrolled components allow form elements to manage their own state internally, without being directly controlled by React's state. In this approach, you typically use the **ref** attribute to access the form values.

 A picture containing text, screenshot, font

Description automatically generated

A picture containing text, screenshot, font, design

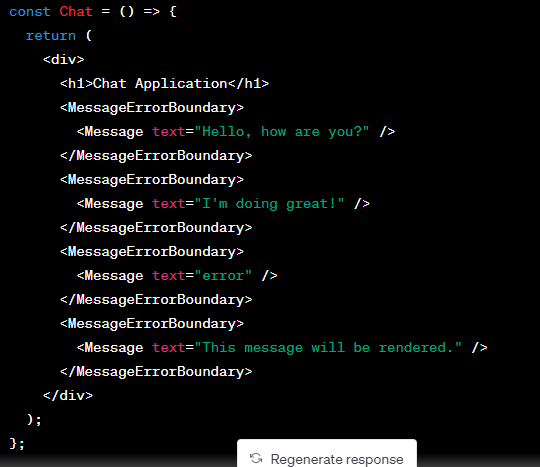
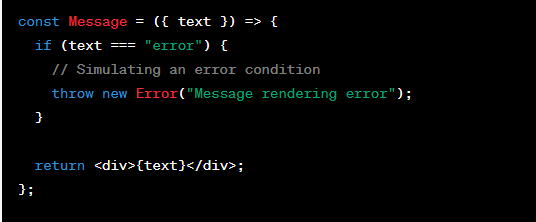
Description automatically generated

1. What is the significance of keys in React? Why are they important when rendering lists?

* Keys in React are used to uniquely identify elements in a list. They help React efficiently update and re-render the list when items are added, removed, or reordered. Keys provide a way for React to keep track of individual components within a list and optimize the rendering process.

1. How do you handle error boundaries in React?

* Error boundaries in React are components that catch JavaScript errors during rendering, in lifecycle methods, or in the constructor of the whole tree below them. To handle errors, you can create an error boundary component by implementing the **componentDidCatch** lifecycle method. This method allows you to handle the error and display a fallback UI to the user.

A picture containing text, electronics, screenshot, software

Description automatically generated