1. What is React.Js ? Explain the Features of React.js

React is a popular open-source JavaScript library used for building user interfaces (UIs) and single-page applications (SPAs). It was developed by Facebook and (Jardan walke).

Features of React.js :

1. React is not a framework, it is javaScript UI libraray.
2. **Component-Based Architecture**: React is based on a component-based architecture, which means that the UI is broken down into smaller, reusable components that can be easily managed and updated.
3. **Virtual DOM:** React uses a virtual DOM, which is a lightweight representation of the actual DOM. This allows React to efficiently update the UI without having to re-render the entire page.
4. **JSX**: React uses JSX (JavaScript XML), which is a syntax extension that allows you to write HTML-like code in your JavaScript files. This makes it easier to write and read code, and also makes it easier to visualize the UI.
5. **One-way Data Binding**: React uses one-way data binding, which means that data flows in one direction from the parent component to the child components. This makes it easier to manage data and reduces the risk of data conflicts.
6. **Declarative Programming**: React uses a declarative programming style, which means that you describe what you want the UI to look like, and React takes care of the details of how to achieve that.
7. **High Performance:** React is designed to be highly performant, with a focus on minimizing the number of DOM updates and optimizing the rendering
8. **Large Community:** React has a large and active community of developers, which means that there are many resources and tools available to help you learn and use React effectively.
9. What is difference between Actual DOM and Virtual DOM?

Actual DOM : 1) Actual Dom (Document Object Model) It is a object representation of complete html page, as a tree-like structure.

2) Dom manipulation is very expensive.

3) There is too mach memeory westage.

4) The Actual DOM takes longer to render changes because it needs to update the layout and repaint the page.

Virtual DOM : 1) Virtual DOM is a lightweight and optimized copy of the Actual DOM. It is a JavaScript object that represents the structure of the Actual DOM.

1. Dom manipulation very easy.
2. No memory westage.
3. The Virtual DOM, on the other hand, is faster because it calculates the difference between the previous and current states and updates only the necessary elements.
4. What is component in react?

A component in React is a **self-contained and reusable block of code** that represents a part of a user interface.

There are 2 types of component in react :

1. Functional Component.
2. Class Component.

Functional Component : Functional components are simpler and more lightweight than class-based components, and they are used for rendering simple UI elements. They are defined as JavaScript functions that take in props (short for properties) as input and return JSX.

Class Component : Class-based components are more powerful and can have more complex behavior than functional components. They are defined as JavaScript classes that extend the React Component class.

Benefits of components :

1. Reusability
2. Modularity
3. Separation of Concerns.
4. Performance

4. How many types of component in react? Explain class component and functional component?

Same as Q.3

5. Explain the life cycle of class component (Mounting , Unmounting, Updating)

In React, class components have three main phases of the component lifecycle: mounting, updating, and unmounting. Each phase consists of a series of lifecycle methods that are automatically called by React at specific points during the component's lifetime.

Mounting: Mounting means putting elements into the DOM.

React has four built-in methods that gets called, in this order, when mounting a component:

1. constructor() - Initializes the component's state and binds methods to the component's context.
2. static getDerivedStateFromProps() - Updates the component's state based on changes to its props.
3. render() - Returns the component's JSX markup to be displayed on the page.
4. componentDidMount() - Executes after the component is mounted on the DOM, allowing access to DOM nodes and initialization of external libraries.

Updating:

1. getDerivedStateFromProps():- Updates the component's state based on changes to its props.
2. render() - Returns the updated JSX markup to be displayed on the page.
3. getSnapShortBeforeUpdate:
4. shouldComponentUpdate() - Determines if the component should be re-rendered based on changes to its state or props.
5. componentDidUpdate() - Executes after the component has been re-rendered and updates have been applied to the DOM.

Unmounting:

1. componentWillUnmount() - Executes before the component is removed from the DOM, allowing cleanup of any resources or event listeners used by the component.

6. How to implement life cycle phase (Mounting, Updating, Unmounting) using functional component.

In functional components, you can use the useEffect() hook to implement the lifecycle phases of mounting, updating, and unmounting.

1. **Mounting**: This phase is triggered when the component is first rendered to the DOM.

Example:

javascript

import React, { useEffect } from 'react';

function MyComponent(props) {

useEffect(() => {

// componentDidMount logic here

return () => {

// componentWillUnmount logic here

}

}, []);

return <p>My Component</p>;

}

In this example, the useEffect() hook is called with an empty dependency array, which means it will only run once, when the component is first mounted. The first function passed to useEffect() contains the logic for componentDidMount, and the function returned from useEffect() contains the logic for componentWillUnmount.

1. **Updating**: This phase is triggered when the component's state or props change.

Example:

javascript

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

function MyComponent(props) {

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

useEffect(() => {

// componentDidUpdate logic here

}, [count]);

const handleClick = () => {

setCount(count + 1);

}

return (

<div>

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

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

</div>

);

}

In this example, the useEffect() hook is called with a dependency array that includes count, which means it will run every time count changes. The function passed to useEffect() contains the logic for componentDidUpdate.

1. **Unmounting**: This phase is triggered when the component is removed from the DOM.

Example:

javascript

import React, { useEffect } from 'react';

function MyComponent(props) {

useEffect(() => {

return () => {

// componentWillUnmount logic here

}

}, []);

return <p>My Component</p>;

}

In this example, the useEffect() hook is called with an empty dependency array and returns a function that contains the logic for componentWillUnmount.

By using the useEffect() hook, you can implement the lifecycle phases of mounting, updating, and unmounting in functional components.

7. What do you mean by statefull and stateless component ?

**A stateful component** (also known as a class component) is a component that manages its own state through the this.state property. It can modify its own state by calling this.setState(). Stateful components are useful for managing complex state and performing data fetching and manipulation.

**A stateless component** (also known as a functional component) is a component that doesn't have its own state and is only dependent on the props it receives. Stateless components are useful for representing simple, presentational components that don't require complex logic or data fetching.

8. What is the difference between state and props.

State : 1) state is mutable.

2) state change can be asynchronous.

3) state can not be accessed by child component.

4) state cannot make component reusable.

5) state holds information about the component.

Props : 1) props are immutable.

2) props are read-only.

3) props can be accessed by child component.

4) props make components reusable.

5) props allow us to pass the data from one component to another component.

9. What is difference between an Element and components ?

**Element**:

* An element is the smallest building block in React applications.
* It describes what you want to see on the screen.
* Elements are plain JavaScript objects that represent a DOM node or a piece of UI.
* Elements are typically created using JSX syntax or React.createElement().
* Elements are immutable and cannot be modified. If you want to update the UI, you create a new element.

Ex:--- const element = <h1>Hello, world!</h1>;

**Component**:

* A component is a reusable, self-contained piece of UI.
* It is a JavaScript function or a class that returns an element or a tree of elements.

Ex:--- function Welcome(props) {

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

}

10. What happen when we call setState method.

When we call setState, we can update the state of a component. The setState method takes an object as an argument, which represents the new state values you want to set.

11. Can we re-render component without using setState

**Yes**, it is possible to re-render a component without using setState(). There are a few ways to achieve this:

1. **Props change:** If the component receives new props from its parent, it will re-render automatically.

Example:

javascript

function MyComponent(props) {

return <p>{props.text}</p>;

}

In this example, if the parent component updates the text prop, MyComponent will re-render with

1. **Force update:** You can force a component to re-render by calling the forceUpdate() method. This method is not recommended as it goes against React's unidirectional data flow and can cause unnecessary re-renders.

12. What is eager loading and lazy loading in react?

Same as Q.13

13. How we load the components eagrly and lazily?

Eager Loading:

1. During program initialization, all the required components or dependencies are loaded immediately.
2. Eager loading can help reduce latency during runtime because all the necessary resources are already available.
3. However, it can also increase the initial loading time and memory usage, especially if the program has a large number of components.

Ex:

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

import { HomeComponent } from './home.component';

import { AboutComponent } from './about.co.mponent';

@NgModule({

declarations: [AppComponent, HomeComponent, AboutComponent],

imports: [BrowserModule],

bootstrap: [AppComponent]

})

export class AppModule { }

Lazy Loading:

1. Components are loaded only when they are required for execution.
2. Lazy loading helps reduce the initial loading time and memory usage because only the necessary components are loaded upfront.
3. It can improve performance by deferring the loading of non-essential components until they are needed.
4. Lazy loading is especially beneficial in scenarios where not all components are frequently used or where loading all components upfront is resource-intensive.

Ex:

import React, { lazy, Suspense } from 'react';

const LazyComponent = lazy(() => import('./LazyComponent'));

function App() {

return (

<div>

<h1>My App</h1>

<Suspense fallback={<div>Loading...</div>}>

<LazyComponent />

</Suspense>

</div>

);

}

export default App;

14. What is error boundaries in react ?

Error Boundaries basically provide some sort of boundaries or checks on errors, They are React components that are used to handle JavaScript errors in their child component tree.

import React, { Component } from 'react';

class ErrorBoundary extends Component {

constructor(props) {

super(props);

this.state = { hasError: false };

}

static getDerivedStateFromError(error) {

return { hasError: true };

}

componentDidCatch(error, errorInfo) {

console.log(error, errorInfo);

}

render() {

if (this.state.hasError){

return <h1>Something went wrong.</h1>;

}

return this.props.children;

}

}

export default ErrorBoundary;

```

```

import React from 'react';

import ErrorBoundary from './ErrorBoundary';

function MyComponent() {

throw new Error('Error occurred in MyComponent.');

return (

<div>

<h1>My Component</h1>

<p>This is my component.</p>

</div>

);

}

function App() {

return (

<ErrorBoundary>

<MyComponent />

</ErrorBoundary>

);

}

export default App;

15. How to implement routing in react? What is use of BrowserRouter and Routes.

React Router is a standard library for routing in React. It enables navigation between views from different components in a React application, allows the browser URL to be changed, and keeps the UI in sync with the URL.

BrowserRouter : stores the current location in the browser's address bar using clean URLs and navigates using the browser's built-in history stack.

16. What is props driling in react?

the process of passing data from one component via several interconnected components to the component that needs it.

// Parent Component

function ParentComponent() {

const data = 'Hello, Prop Drilling!';

return (

<div>

<IntermediateComponent data={data} />

</div>

);

}

// Intermediate Component

function IntermediateComponent({ data }) {

return (

<div>

<ChildComponent data={data} />

</div>

);

}//////

// Child Component

function ChildComponent({ data }) {

return <div>{data}</div>;

}

17. What is pure component in react ? How create pure class component?

A pure component is a component that only re-renders when its props or state have changed. It helps improve performance by preventing unnecessary re-renders when the component's inputs haven't changed.

OR

If the previous value of state or props and the new value of state or props is the same, the component will not re-render itself. Since **Pure Components** restricts the re-rendering when there is no use of re-rendering of the component. Pure Components are Class Components which extends **React.PureComponent**.

import React from ‘react’;

export default class Test extends React.PureComponent{

   render(){

      return <h1>Welcome to GeeksforGeeks</h1>;

   }

}

18. How to use function component as pure component?

By using React.memo()

import React from 'react';

const MyComponent = ({ name }) => {

console.log('Rendering MyCompo nent');

return <div>Hello, {name}!</div>;

};

const PureMyComponent = React.memo(MyComponent);

const App = () => {

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

const increment = () => {

setCount(prevCount => prevCount + 1);

};

return (

<div>

<button onClick={increment}>Increment</button>

<PureMyComponent name="Alice" />

<PureMyComponent name="Bob" />

<PureMyComponent name="Charlie" />

<div>Count: {count}</div>

</div>

);

};

export default App;

19. What is Higher order component(HOC)

Higher-order components or HOC is the advanced method of reusing the component functionality logic. It simply takes the original component and returns the enhanced component.

Name.js:

import React from 'react'

const EnhancedComponent = (OriginalComponent) => {

    class NewComponent extends React.Component {

        // Logic here

        render() {

            // Pass the callable props to Original component

            return <OriginalComponent name="GeeksforGeeks"/>

        }

    }

    // Returns the new component

    return NewComponent

}

export default EnhancedComponent;

App.js

import React from "react";

import "./App.css"

import EnhancedComponent from './Name'

class App extends React.Component {

  render() {

    // Call the props from originalComponent

    return <h1>{this.props.name}</h1>

  }

}

// Passed the originalcomponent

export default EnhancedComponent(App);

20. What is protected/private route in react.js

Protected routes are routes that allow access to authorized users only. This means that users must first meet certain conditions before accessing that specific route. This is essential for securing certain routes or information. There are various ways you can use protected routes in React. One of them is using a higher-order component that wraps a protected route and checks if a user is authenticated before rendering a component.

21.What are react fragements?

Fragments allow you to group a list of children without adding extra nodes to the DOM.

22. What are the hooks in react.js ? Explain some hooks which used in your project?

This are the special function : - It is use with functional component only.

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.

Functional component+ Hooks = statefull component(class component)

When we use hook : 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.

23. Explain following

a. useState : useState is a built-in hook in React that allows functional components to manage state. . The useState hook takes an initial value as an argument and returns an array with two values: the current state value and a function to update the state.

b. UseLocation : this hook is return the location object that is used by react-router. This object represent the current url and is immutable.Whenever the url changes the useLocatiion hook return a newly updated location object.

c. useNavigate : this hook is used to perform programmatic navigation within your application. It return a navigate function that you can use to navigate to different routes.

d. useRef : this hook is used to create a mutable reference to an element or a value. It is primarily used to access and interact with DOM element directly. It returns mutable object.

e. UseMemo : useMemo hook is only run when one of its state is change. The react useMemo hook returns memoize value.

If we want to send the object as an argument into another component then we have to use usememo hook.

f. UseCallback : useCallback hook is only run when one of its dependency is change. The react useCallback returns memorize callback .

If we want to send the function as an argument into another component then we have to use useCallback hook.

g. UseDispatch : It is used to dispatch the action in our Application.

h. UseContext : if we want to use parent component data in child component then we have to use useContext hook.

If we want to pass the data by component tree without passing any props at every level so we use useContext hook.

24. in which life cycle method you would like to make a http call

In class components,we can make an API/HTTP/network call in the componentDidMount() method. This method is call once, immediately after the component is mounted and the DOM is ready.

25. Where to make a api/http/network call in functional components

In functional components, we can make an API/HTTP/network call in the useEffect hook. The useEffect hook is used to manage side effects in functional components, such as fetching data from an API, updating the DOM, or subscribing to events.

26. Write a code using useEffect to implement mounting, updating and unmounting in functional components.

In functional components, you can use the useEffect() hook to implement the lifecycle phases of mounting, updating, and unmounting.

1. **Mounting**: This phase is triggered when the component is first rendered to the DOM.

Example:

javascript

import React, { useEffect } from 'react';

function MyComponent(props) {

useEffect(() => {

// componentDidMount logic here

return () => {

// componentWillUnmount logic here

}

}, []);

return <p>My Component</p>;

}

In this example, the useEffect() hook is called with an empty dependency array, which means it will only run once, when the component is first mounted. The first function passed to useEffect() contains the logic for componentDidMount, and the function returned from useEffect() contains the logic for componentWillUnmount.

1. **Updating**: This phase is triggered when the component's state or props change.

Example:

javascript

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

function MyComponent(props) {

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

useEffect(() => {

// componentDidUpdate logic here

}, [count]);

const handleClick = () => {

setCount(count + 1);

}

return (

<div>

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

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

</div>

);

}

In this example, the useEffect() hook is called with a dependency array that includes count, which means it will run every time count changes. The function passed to useEffect() contains the logic for componentDidUpdate.

1. **Unmounting**: This phase is triggered when the component is removed from the DOM.

Example:

javascript

import React, {useEffect} from 'react';

function MyComponent(props) {

useEffect(() => {

return () => {

// componentWillUnmount logic here

}

}, []);

return <p>My Component</p>;

}

In this example, the useEffect() hook is called with an empty dependency array and returns a function that contains the logic for componentWillUnmount.

By using the useEffect() hook, you can implement the lifecycle phases of mounting, updating, and unmounting in functional components.

27. Can we update props ?

In React, props are read-only and cannot be directly updated by a child component. The parent component that owns the props can update them, and any changes to the props will trigger a re-render of the child component with the new prop values.

However, if a child component needs to change its behavior based on a prop value, it can manage that internally using state. The child component can create a state variable based on the prop value and update it using the useState hook or setState method (in the case of class components).

When the parent component updates the prop value, the child component will receive the new value in its props and can update its internal state accordingly to reflect the change

28. How to update parent state data from child component ?

To update the parent state from the children component, either we can use additional dependencies like Redux or we can use this simple method of passing the state of the parent to the children and handli it accordingly.

import React, { useState } from 'react';

import ChildComponent from './ChildComponent';

function ParentComponent() {

const [data, setData] = useState('');

const updateData = (newData) => {

setData(newData);

};

return (

<div>

<h2>Data in Parent Component: {data}</h2>

<ChildComponent updateData={updateData} />

</div>

);

}

----------------------------------------------------------

import React from 'react';

function ChildComponent({ updateData }) {

const handleButtonClick = () => {

// Update the parent's state by invoking the callback function

updateData('New Data from Child');

};

return (

<div>

<button onClick={handleButtonClick}>Update Parent State</button>

</div>

);

}

29. What is context api in react.js

Context API allows data to be passed through a component tree without having to pass props manually at every level.

import { createContext } from "react"

import B from "./B";

export const context = createContext()

export default function A) {

    return <>

        <context.Provider value={{ name: "payal" }}>

            <B></B>

        </context.Provider>

    </>

}

import C from "./C";

export default function B(){

    return <>

    <C/>

    </>

}

import { useContext } from "react";

import { context } from "./A";

export default function C() {

    const { name } = useContext(context);

    return <>

        <h1>{name}</h1>

    </>

}

30. What is redux? Explain the component of react (Store, Action, Reducer)?

Redux : redux is state managing library in javascript. It simply manages state in our application, In other word we can say that it is used to manage data in our application. It is used with library like React.

**Uses:** It makes easier to manage state and data of our application.

There are three principles of Redux :

1. **Single source of truth** : It helps to create universal apps. The state of our application is stored in one object tree called store. It means one app, one store.
2. **State is read-only(immutable)** : It means we can not change in state object and its properties directly.
3. **Changes are made with pure function(reducers)** : Reducers are pure function that take previous state and action and return a new state.

**Building Parts of redux:**

1. **Action**
2. **Reducers**
3. **Store**

**Actions:** Actions are a plain JavaScript object that contains information. Actions are the only source of information for the store. Actions have a type field that tells what kind of action to perform.

import { createSlice } from "@reduxjs/toolkit";

const slice = createSlice({

    name: 'seller',

    initialState: {

        currentSeller: null

    }

});

export default slice.reducer;

**Reducers:** As we already know, actions only tell what to do, but they don’t tell how to do, so reducers are the pure functions that take the current state and action and return the new state and tell the store how to do.

import { createSlice } from "@reduxjs/toolkit";

const slice = createSlice({

    name: 'seller',

    initialState: {

        currentSeller: null

    },

    reducers: {

        setSeller: (state, action) => {

            state.currentSeller = action.payload;

        },

Bbw2

        signOut: (state, action) => {

            state.currentSeller = null;

        }

    }

});

export const { setSeller, signOut } = slice.actions;

export default slice.reducer;

**Store:** The store is the object which holds the state of the application.  
**Functions associated with Store:** 

* createStore() – To create a store
* dispatch(action) -To change the state
* getState() – for getting current state of store.
* import { configureStore } from "@re.
  + duxjs/toolkit";
* import SellerSlice from "./sellerSignInSlice";
* const store = configureStore({
* reducer: {
* seller: SellerSlice,
* }
* })
* export default store

31. What is difference between useState and useReducer.

The useReducer hook handles more complex logic regarding the state updates. So if you're state is a single boolean, number, or string, Then it's obvious to use useState hook. However if your state is an object (example: person's information) or an array (example: array of products ) useReducer will be more appropriate to use.

32. What is call, bind, apply in js.

call() : call method is a method which we can use in different objects.

const person = {  
  firstName:"John",  
  lastName: "Doe",  
  fullName: function () {  
    return this.firstName + " " + this.lastName;  
  }  
}  
  
// This will return "John Doe":  
person.fullName();

bind() : with bind method an object can borrow a method from another object.

const person = {  
  firstName:"John",  
  lastName: "Doe",  
  fullName: function () {  
    return this.firstName + " " + this.lastName;  
  }  
}  
  
const member = {  
  firstName:"Hege",  
  lastName: "Nilsen",  
}  
  
let fullName = person.fullName.bind(member);

apply() : apply method is similar to call method. we can also use this method in different objects.

const person = {  
  fullName: function() {  
    return this.firstName + " " + this.lastName;  
  }  
}  
  
const person1 = {  
  firstName: "Mary",  
  lastName: "Doe"  
}  
  
// This will return "Mary Doe":  
person.fullName.apply(person1);

33. Difference between context api and redux?

useContext : 1) useContext hook is provide the way of pass data through component tree without passing props down through each nested component.

2) It is used to share data.

3) changes are made with the Context value.

4) We can change the state in it.

5) It is better to use with small applications.

6) It is easy to understand and requires less code.

7) It re-renders all components whenever there is any update in the provider’s value prop.

Syntax:

const Context = useContext(initialValue);

redux : 1) redux is a state managing library in javaScript.

2) It is used to manage data.

3) Changes are made with pure functions i.e. reducers.

4) The state is read-only. We cannot change them directly.

5) It is perfect for larger applications.

6) It is quite complex to understand.

7) It only re-render the updated components.

34. What is RTK ?

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

1. Too much code to configure the store.
2. Writing too much boilerplate code to dispatch actions and store the data in the reducer.
3. Extra packages like Redux-Thunk and Redux-Saga for doing asynchronous actions.

35. what is the use async thunk (createAsyncThunk) ?

Whenever we want to do asynchronous operations in redux then we have to use thunk (createAsyncThunk) method.

createAsyncThunk takes two arguments: a string that defines the name of the action, and a function that returns a Promise.

Thunk is used for fetching data from a API and storing response in Redux states which shortens and increases code clean-up.

import { createAsyncThunk, createSlice } from "@reduxjs/toolkit";

import axios from "axios";

export const fetchProduct = createAsyncThunk("fetchProduct", async () => {

    let response = await axios.get("http://localhost/product/viewproduct")

    console.log(response.data.products);

    return response.data.products;

});

const slice = createSlice({

    name: "Product",

    initialState: {

        Product: []

    }

})

36. What is memo in reactjs.

React Memo is a HOC that wrap a component to memoize rendered result and avoiding unnecessary renderings . This improve performance because it memorize result and skip rendering.

There are two ways you can wrap your component with React.memo() :

1. you can wrap the actual component without creating new variable to store the memorize component.

const myComponent = React.memo((props) => {

/\* render using props \*/

});

export default myComponent;

1. Another option is to create a new variable to store the memoized component and then export the new variable:

const myComponent = (props) => {

/\* render using props \*/

};

export const MemoizedComponent = React.memo(myComponent);

When we use UseMemo :

* Use React Memo if your component will render quite often.
* Use it when your component often renders with the **same props**. This happens to child components who are forced to re-render with the same props whenever the parent component renders.
* Use it in pure functional components alone. If you are using a class component, use the [React.PureComponent](https://reactjs.org/docs/react-api.html#reactpurecomponent).
* Use it if your component is big enough (contains a decent amount of UI elements) to have props equality check.

37. What is JSX ?

JSX (JavaScript XML) is an extention syntax used in React.

JSX allows us to write HTML in React.

JSX makes it easier to write and add HTML in React.

const element = <h1>Hello, JSX!</h1>;

38. What is reconciliation ?

Process of determine the changes between new virtual dom and previous virtual dom is called Diffing.

After diffing the process of made updation permanent in Actual Dom is called reconciliation.

39. What is controlled component and uncontrolled components ?

In React, components can be either "controlled" or "uncontrolled" depending on how they manage state :

**Controlled components** are components where the state is managed by the React component itself.

import { useState } from 'react';

function ControlledInput(props) {

const [value, setValue] = useState('');

const handleChange = (event) => {

setValue(event.target.value);

props.onChange(event.target.value);

};

return (

<input type="text" value={value} onChange={handleChange} />

);

}

export default ControlledInput;

In this example, the ControlledInput component manages its own state using the useState hook.

**Uncontrolled components** are components where the state is managed by the DOM itself.

import React from 'react';

function UncontrolledInput(props) {

const inputRef = React.useRef();

const handleClick = () => {

const value = inputRef.current.value;

props.onClick(value);

};

return (

<div>

<input type="text" ref={inputRef} />

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

</div>

);

}

export default UncontrolledInput;

In this example, the UncontrolledInput component uses a ref to get the current value of the input element when the user clicks the "Submit" button.

40. What does shouldComponentUpdate do and why is it important?

In react shouldcomponentUpdate method is one of the valuable method. This method is always return Boolean value and the execution of this method is always depends on true Or false. This method use to control unneccessory re-rendering. If shouldcomponentUpdate method retun true then the component will be re-render otherwise will not re-render.

Syntax :

ShouldComponentUpdate(nextProps, nextState){  
 return this.state.stateName ! = nextState.stateName;

}

41. Describe how events are handled in React.

42. What is the second argument that can optionally be passed to setState and what is its purpose?

43. List down the variation of setState() in reactjs ?

In React, the setState() method is used to update the state of a component,

Here are the variations of setState() :

1. Basic usage :  
   this.setState({key:value});
2. Functional update :

This.setState((prevState,props)=>({key:prevState.key+1}));

1. Async update :

this.setState({ key: value }, () => {

// callback function

});

44. what is props.childrens ?

In React props.children is a special prop that allow component to pass children element or component to another component as a part of their JSX markup. The prop.children prop is a reserved name and is used to represend content between the opening and closing tags of a component.

For example, suppose you have a component called Card that renders a styled box with a title and some content. You could use the props.children prop to pass in the content for the card:

function Card(props) {

return (

<div className="card">

<h2>{props.title}</h2>

<div className="content">

{props.children}

</div>

</div>

);

}

function App() {

return (

<Card title="My Card">

<p>This is the content for my card.</p>

</Card>

**);**

**}**

45. Does react use HTML ?

React is a JavaScript library for building user interfaces and it does not use HTML directly.

React uses JSX,which is syntax extention for javaScript that allow you to write Html-like syntax within your javascript code. JSX is not Html.

46. Tell me most signification drawback of reactjs?

47. Explaing React Router?

React Router is a standard library for routing in React. It enables navigation between views from differrent components in a React application, allows the browser URL to be changed, and keeps the UI in sync with the URL.

48. How modify each request and response (Interceptor) in react.js

Axios provides a way to intercept both the request and response before they are handled by your application. This allows you to modify the request or response headers, add authentication tokens, or perform other actions.

49. What is rest api ? How to call api from react ?

REST (Representational State Transfer) API is a type of web service that is used to create web applications that can communicate with each other using HTTP protocols. RESTful API is designed to be stateless, cacheable, and able to support a wide variety of data formats.

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

import axios from 'axios';

function App() {

const [data, setData] = useState([]);

useEffect(() => {

axios.get('https://api.example.com/data')

.then(response => setData(response.data))

.catch(error => console.error(error));

}, []);

return (

<div>

<h1>API Data</h1>

<ul>

{data.map(item => (

<li key={item.id}>{item.name}</li>

))}

</ul>

</div>

);

}

export default App;

50. How to send file in request from react ?

To send a file in a request from a React application, we can make use of the FormData API, which allows to send form data, including file uploads.

1. First, create a file input field in your component's JSX markup:

<input type="file" onChange={handleFileChange} />

2. we get the value of input field

const handleFileChange = (event) => {

const file = event.target.files[0];

// Do something with the fil e, such as storing it in component state

};

create a new FormData object and append the file to it:

const formData = new FormData();

formData.append('file', file);

axios.post('/upload', formData)

.then(response => {

// Handle the response

})

.catch(error => {

// Handle any errors

});

51. Why should we not update the state directly ?

In React, it's not recommended to update the state directly because it can lead to unexpected behavior and bugs in your application. Instead, we should use the setState method provided by React to update the state.

52. What is the purpose of callback function as an argument of setState().

In React, the setState() method is used to update the state of a component. This method is asynchronous, which means that the state may not be immediately updated when the setState() method is called. To handle this situation, setState() can take a callback function as an argument.

class MyComponent extends React.Component {

state = {

count: 0

};

handleClick = () => {

this.setState({ count: this.state.count + 1 }, () => {

console.log('Count is now:', this.state.count);

});

};

render() {

return (

<div>

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

<button onClick={this.handleClick}>Increment</button>

</div>

);

}

}

53. How to bind method and event handlers in jsx ?

There are 3 ways to bind method and event handler in JSX :

1) **Binding in Constructor :** In JavaScript classes, the methods are not bound by default. The same thing applies for React event handlers defined as class methods. Normally we bind them in constructor.

class Component extends React.Component {

constructor(props) {

super(props);

this.handleClick = this.handleClick.bind(this);

}

handleClick() {

// ...

}

}

2.**Public class fields syntax**: If you don't like to use bind approach then public class fields syntax can be used to correctly bind callbacks

handleClick = () => {

console.log('this is:', this);

};

<button onClick={this.handleClick}>{'Click me'}</button>

3.**Arrow functions in callbacks**: You can use arrow functions directly in the callbacks

<button onClick={(event) => this.handleClick(event)}>{'Click me'}</button>

54. What are syntheic event in reactjs ?

Synthetic events in React provide a consistent interface for event handling and improve performance by wrapping native browser events

import React from 'react';

function handleClick(event) {

event.preventDefault();

console.log('Button clicked!');

}

function MyButton() {

return (

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

);

}

export default MyButton;

55. What are inline conditional expression ?

Inline conditional expressions in React allow for conditional rendering of elements based on a certain condition, using a ternary-like syntax.

import React from 'react';

function MyComponent(props) {

const isLoggedIn = props.isLoggedIn;

return (

<div>

{isLoggedIn ? (

<h1>Welcome back!</h1>

) : (

<h1>Please log in.</h1>

)}

</div>

);

}

export default MyComponent;

56. What is key props and what is the benefit of using it in arrays of elements?

Key:-

A key is a special string attribute you should include when creating arrays of elements. Key prop helps React identify which items have changed, are added, or are removed

const todoItems = todos.map((todo) => <li key={todo.id}>{todo.text}</li>);

57. What is the use refs ?

In React, the useRef hook is used to create a mutable reference to a DOM element or a value that persists across component re-renders. This can be useful for accessing DOM nodes.

58. How to create refs?

import React, { useRef } from 'react';

function MyComponent() {

const inputRef = useRef(null);

function handleClick() {

let Name = inputRef.current.value;

}

return (

<div>

<input type="text" ref={inputRef} />

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

</div>

);

}

59. What are forward ref ?

The forwardRef method in React allows parent components to move down (or “forward”) refs to their children. ForwardRef gives a child component a reference to a DOM entity created by its parent component in React. This helps the child to read and modify the element from any location where it is used.

Syntax : React.forwardRef((props, ref) => {})

Ex : ---

Import React from “react”;

Function Parent(){  
const InputRef = useRef(null);

const handleClick = ()=>{  
 InputRef.current.focus();

}

Return <>

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

<Child ref={InputRef}/>

</>

}

Function Child = forwordRef((prop,ref)=>{

Return <>

<input ref={ref}/>

</>

})