React Interview Questions

1. Differentiate between Real DOM and Virtual DOM.

Real DOM vs Virtual DOM	
Real DOM	**Virtual DOM**
1. It updates slow.	1. It updates faster.
2. Can directly update HTML.	2. Can't directly update HTML.
3. Creates a new DOM if element updates.	3. Updates the JSX if element updates.
4. DOM manipulation is very expensive.	4. DOM manipulation is very easy.
5. Too much of memory wastage.	5. No memory wastage.

2. What is React?

- React is a front-end JavaScript library developed by Facebook in 2011.
- It follows the component based approach which helps in building reusable UI components.
- It is used for developing complex and interactive web and mobile UI.
- Even though it was open-sourced only in 2015, it has one of the largest communities supporting it.

3. What are the features of React?

Major features of React are listed below:

i. It uses the **virtual DOM** instead of the real DOM.
ii. It uses **server-side rendering**.
iii. It follows **uni-directional data flow** or data binding.

4. List some of the major advantages of React.

Some of the major advantages of React are:

i. It increases the application's performance
ii. It can be conveniently used on the client as well as server side
iii. Because of JSX, code's readability increases
iv. React is easy to integrate with other frameworks like Meteor, Angular, etc
v. Using React, writing UI test cases become extremely easy

5. What are the limitations of React?

Limitations of React are listed below:

i. React is just a library, not a full-blown framework
ii. Its library is very large and takes time to understand
iii. It can be little difficult for the novice programmers to understand
iv. Coding gets complex as it uses inline templating and JSX

6. What is JSX?

JSX is a shorthand for JavaScript XML. This is a type of file used by React which utilizes the expressiveness of JavaScript along with HTML like template syntax. This makes the HTML file really easy to understand. This file makes applications robust and boosts its performance. Below is an example of JSX:

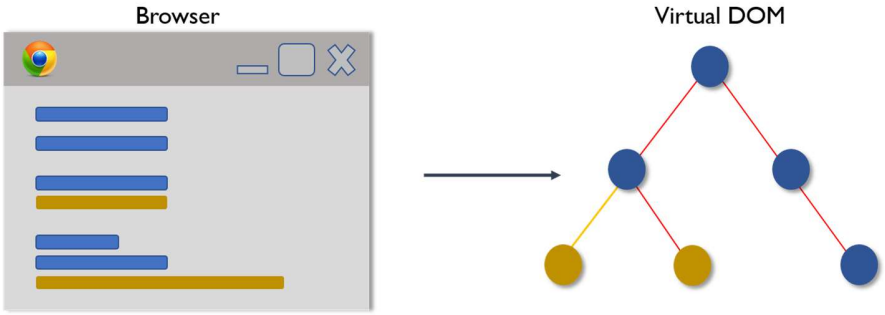
```
render(){
    return(

<div>

<h1> Hello World from Edureka!!</h1>

        </div>

    );
}
```

7. What do you understand by Virtual DOM? Explain its works.

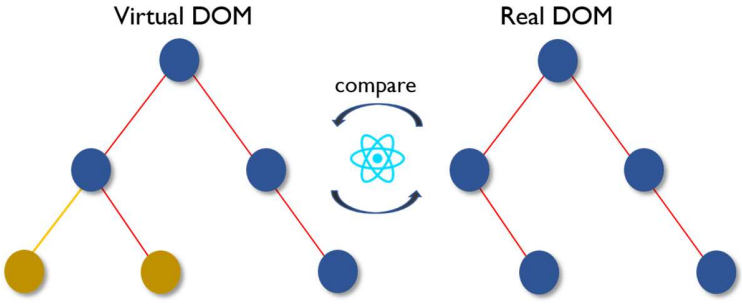
A virtual DOM is a lightweight JavaScript object which originally is just a copy of the real DOM. It is a node tree that lists the elements, their attributes and content as Objects and their properties. React's render function creates a node tree out of the React components. It then updates this tree in response to the mutations in the data model which is caused by various actions done by the user or by the system. Check out this Web developer course online to learn more about react.

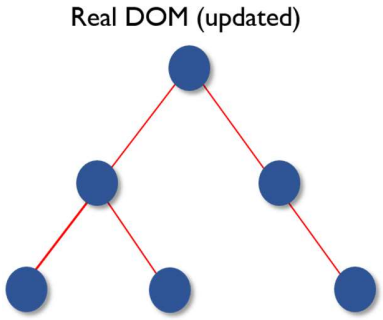
This Virtual DOM works in three simple steps.

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. 

8. Why can't browsers read JSX?

Browsers can only read JavaScript objects but JSX in not a regular JavaScript object. Thus to enable a browser to read JSX, first, we need to transform JSX file into a JavaScript object using JSX transformers like Babel and then pass it to the browser.

9. How different is React's ES6 syntax when compared to ES5?

Syntax has changed from ES5 to ES6 in the following aspects:

i. require vs import

```
// ES5
var React = require('react');

// ES6
import React from 'react';
```

ii. export vs exports

```
// ES5
module.exports = Component;

// ES6
export default Component;
```

iii. component and function

```
// ES5
var MyComponent = React.createClass({
    render: function() {
        return

<h3>Hello Edureka!</h3>
;
    }
});

// ES6
class MyComponent extends React.Component {
    render() {
        return

<h3>Hello Edureka!</h3>
;
    }
}
```

iv. props

```
// ES5
var App = React.createClass({
    propTypes: { name: React.PropTypes.string },
    render: function() {
        return

<h3>Hello, {this.props.name}!</h3>
;
    }
});

// ES6
class App extends React.Component {
    render() {
        return

<h3>Hello, {this.props.name}!</h3>
```

```
18;
19     }
20}
```

v. state

```
1 // ES5
2 var App = React.createClass({
3     getInitialState: function() {
4         return { name: 'world' };
5     },
6     render: function() {
7         return
8
9 <h3>Hello, {this.state.name}!</h3>
10;
11     }
12});
13
14// ES6
15class App extends React.Component {
16     constructor() {
17         super();
18         this.state = { name: 'world' };
19     }
20     render() {
21         return
22
23<h3>Hello, {this.state.name}!</h3>
24;
25     }
26}
```

10. How is React different from Angular?

React vs Angular		
TOPIC	**REACT**	**ANGULAR**
1. ARCHITECTURE	Only the View of MVC	Complete MVC

2. RENDERING	Server-side rendering	Client-side rendering
3. DOM	Uses virtual DOM	Uses real DOM
4. DATA BINDING	One-way data binding	Two-way data binding
5. DEBUGGING	Compile time debugging	Runtime debugging
6. AUTHOR	Facebook	Google

React Components – React Interview Questions

11. "In React, everything is a component." Explain.

Components are the building blocks of a React application's UI. These components split up the entire UI into small independent and reusable pieces. Then it renders each of these components independent of each other without affecting the rest of the UI.

12. What is the purpose of render() in React.

Each React component must have a **render()** mandatorily. It returns a single React element which is the representation of the native DOM component. If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as **<form>, <group>, ** etc. This function must be kept pure i.e., it must return the same result each time it is invoked.

13. How can you embed two or more components into one?

We can embed components into one in the following way:

```
class MyComponent extends React.Component{
    render(){
        return(

<div>

<h1>Hello</h1>

                <Header/>
```

```
            </div>

        );
    }
}
class Header extends React.Component{
    render(){
        return

<h1>Header Component</h1>

   };
}
ReactDOM.render(
    <MyComponent/>, document.getElementById('content')
);
```

14. What is Props?

Props is the shorthand for Properties in React. They are read-only components which must be kept pure i.e. immutable. They are always passed down from the parent to the child components throughout the application. A child component can never send a prop back to the parent component. This help in maintaining the unidirectional data flow and are generally used to render the dynamically generated data.

15. What is a state in React and how is it used?

States are the heart of React components. States are the source of data and must be kept as simple as possible. Basically, states are the objects which determine components rendering and behavior. They are mutable unlike the props and create dynamic and interactive components. They are accessed via **this.state().**

16. Differentiate between states and props.

States vs Props

Conditions	State	Props
1. Receive initial value from parent component	Yes	Yes

2. Parent component can change value	No	Yes
3. Set default values inside component	Yes	Yes
4. Changes inside component	Yes	No
5. Set initial value for child components	Yes	Yes
6. Changes inside child components	No	Yes

17. How can you update the state of a component?

State of a component can be updated using this.setState().

```
class MyComponent extends React.Component {
    constructor() {
        super();
        this.state = {
            name: 'Maxx',
            id: '101'
        }
    }
    render()
        {
            setTimeout(()=>{this.setState({name:'Jaeha', id:'222'})},2000)
            return (

<div>

<h1>Hello {this.state.name}</h1>

<h2>Your Id is {this.state.id}</h2>

                    </div>

            );
        }
    }
ReactDOM.render(
    <MyComponent/>, document.getElementById('content')
);
```

18. What is arrow function in React? How is it used?

Arrow functions are more of brief syntax for writing the function expression. They are also called *'fat arrow'* (**=>**) the functions. These functions allow to bind the context of the components properly since in ES6 auto binding is not available by default. Arrow functions are mostly useful while working with the higher order functions.

```
//General way
render() {
    return(
        <MyInput onChange={this.handleChange.bind(this) } />
    );
}
//With Arrow Function
render() {
    return(
        <MyInput onChange={ (e) => this.handleOnChange(e) } />
    );
}
```

19. Differentiate between stateful and stateless components.

Stateful vs Stateless	
Stateful Component	**Stateless Component**
1. Stores info about component's state change in memory	1. Calculates the internal state of the components
2. Have authority to change state	2. Do not have the authority to change state
3. Contains the knowledge of past, current and possible future changes in state	3. Contains no knowledge of past, current and possible future state changes
4. Stateless components notify them about the requirement of the state change, then they send down the props to them.	4. They receive the props from the Stateful components and treat them as callback functions.

20. What are the different phases of React component's lifecycle?

There are three different phases of React component's lifecycle:

i. *Initial Rendering Phase:* This is the phase when the component is about to start its life journey and make its way to the DOM.
ii. *Updating Phase:* Once the component gets added to the DOM, it can potentially update and re-render only when a prop or state change occurs. That happens only in this phase.
iii. *Unmounting Phase:* This is the final phase of a component's life cycle in which the component is destroyed and removed from the DOM.

21. Explain the lifecycle methods of React components in detail.

Some of the most important lifecycle methods are:

i. ***componentWillMount()*** – Executed just before rendering takes place both on the client as well as server-side.
ii. ***componentDidMount()*** – Executed on the client side only after the first render.
iii. ***componentWillReceiveProps()*** – Invoked as soon as the props are received from the parent class and before another render is called.
iv. ***shouldComponentUpdate()*** – Returns true or false value based on certain conditions. If you want your component to update, return **true** else return **false**. By default, it returns false.
v. ***componentWillUpdate()*** – Called just before rendering takes place in the DOM.
vi. ***componentDidUpdate()*** – Called immediately after rendering takes place.
vii. ***componentWillUnmount()*** – Called after the component is unmounted from the DOM. It is used to clear up the memory spaces.

22. What is an event in React?

In React, events are the triggered reactions to specific actions like mouse hover, mouse click, key press, etc. Handling these events are similar to handling events in DOM elements. But there are some syntactical differences like:

i. Events are named using camel case instead of just using the lowercase.
ii. Events are passed as functions instead of strings.

The event argument contains a set of properties, which are specific to an event. Each event type contains its own properties and behavior which can be accessed via its event handler only.

23. How do you create an event in React?

```
class Display extends React.Component({
    show(evt) {
        // code
    },
    render() {
        // Render the div with an onClick prop (value is a function)
        return (

<div onClick={this.show}>Click Me!</div>

        );
    }
});
```

24. What are synthetic events in React?

Synthetic events are the objects which act as a cross-browser wrapper around the browser's native event. They combine the behavior of different browsers into one API. This is done to make sure that the events show consistent properties across different browsers.

25. What do you understand by refs in React?

Refs is the short hand for References in React. It is an attribute which helps to store a reference to a particular React element or component, which will be returned by the components render configuration function. It is used to return references to a particular element or component returned by render(). They come in handy when we need DOM measurements or to add methods to the components.

```
class ReferenceDemo extends React.Component{
     display() {
         const name = this.inputDemo.value;
         document.getElementById('disp').innerHTML = name;
```

```
     }
render() {
    return(

<div>
            Name: <input type="text" ref={input => this.inputDemo = input} />
            <button name="Click" onClick={this.display}>Click</button>

<h2>Hello <span id="disp"></span> !!!</h2>

      </div>
    );
   }
}
```

26. List some of the cases when you should use Refs.

Following are the cases when refs should be used:

- When you need to manage focus, select text or media playback
- To trigger imperative animations
- Integrate with third-party DOM libraries

27. How do you modularize code in React?

We can modularize code by using the export and import properties. They help in writing the components separately in different files.

```
//ChildComponent.jsx
export default class ChildComponent extends React.Component {
    render() {
        return(

<div>

<h1>This is a child component</h1>

          </div>

        );
```

```
    }
}

//ParentComponent.jsx
import ChildComponent from './childcomponent.js';
class ParentComponent extends React.Component {
    render() {
        return(

<div>
                <App />
            </div>

        );
    }
}
```

28. How are forms created in React?

React forms are similar to HTML forms. But in React, the state is contained in the state property of the component and is only updated via setState(). Thus the elements can't directly update their state and their submission is handled by a JavaScript function. This function has full access to the data that is entered by the user into a form.

```
handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
}

render() {
    return (

<form onSubmit={this.handleSubmit}>
            <label>
                Name:
                <input type="text" value={this.state.value} onChange={this.handl
            </label>
            <input type="submit" value="Submit" />
        </form>
```

```
16
17      );
18}
```

29. What do you know about controlled and uncontrolled components?

Controlled vs Uncontrolled Components	
Controlled Components	**Uncontrolled Components**
1. They do not maintain their own state	1. They maintain their own state
2. Data is controlled by the parent component	2. Data is controlled by the DOM
3. They take in the current values through props and then notify the changes via callbacks	3. Refs are used to get their current values

React Interview Questions

30. What are Higher Order Components(HOC)?

Higher Order Component is an advanced way of reusing the component logic. Basically, it's a pattern that is derived from React's compositional nature. HOC are custom components which wrap another component within it. They can accept any dynamically provided child component but they won't modify or copy any behavior from their input components. You can say that HOC are 'pure' components.

31. What can you do with HOC?

HOC can be used for many tasks like:

- Code reuse, logic and bootstrap abstraction
- Render High jacking
- State abstraction and manipulation
- Props manipulation

32. What are Pure Components?

Pure components are the simplest and fastest components which can be written. They can replace any component which only has a **render().** These components enhance the simplicity of the code and performance of the application.

33. What is the significance of keys in React?

Keys are used for identifying unique Virtual DOM Elements with their corresponding data driving the UI. They help React to optimize the rendering by recycling all the existing elements in the DOM. These keys must be a unique number or string, using which React just reorders the elements instead of re-rendering them. This leads to increase in application's performance.

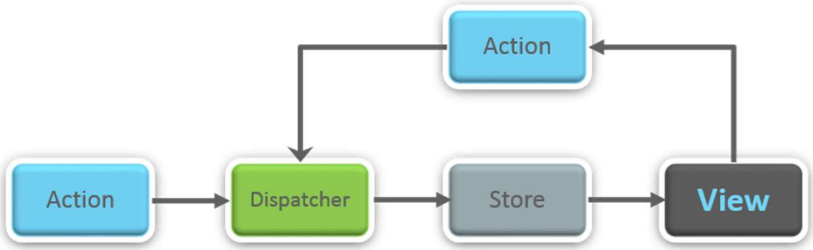
34. What were the major problems with MVC framework?

Following are some of the major problems with MVC framework:

- DOM manipulation was very expensive
- Applications were slow and inefficient
- There was huge memory wastage
- Because of circular dependencies, a complicated model was created around models and views

35. Explain Flux.

Flux is an architectural pattern which enforces the uni-directional data flow. It controls derived data and enables communication between multiple components using a central Store which has authority for all data. Any update in data throughout the application must occur here only. Flux provides stability to the application and reduces run-time errors.



36. What is Redux?

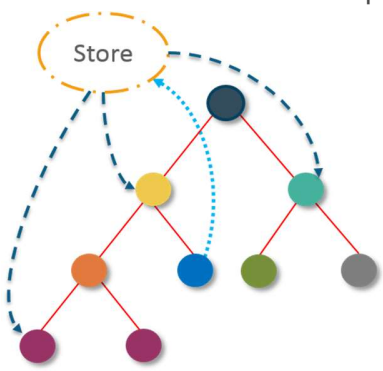
Redux is one of the most trending libraries for front-end development in today's marketplace. It is a predictable state container for JavaScript applications and is used for the entire applications state management. Applications developed with Redux are easy to test and can run in different environments showing consistent behavior.

37. What are the three principles that Redux follows?

i. ***Single source of truth:*** The state of the entire application is stored in an object/ state tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.

ii. ***State is read-only:*** The only way to change the state is to trigger an action. An action is a plain JS object describing the change. Just like state is the minimal representation of data, the action is the minimal representation of the change to that data.

iii. ***Changes are made with pure functions:*** In order to specify how the state tree is transformed by actions, you need pure functions. Pure functions are those whose return value depends solely on the values of their arguments.



38. What do you understand by "Single source of truth"?

Redux uses 'Store' for storing the application's entire state at one place. So all the component's state are stored in the Store and they receive updates from the Store itself. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.

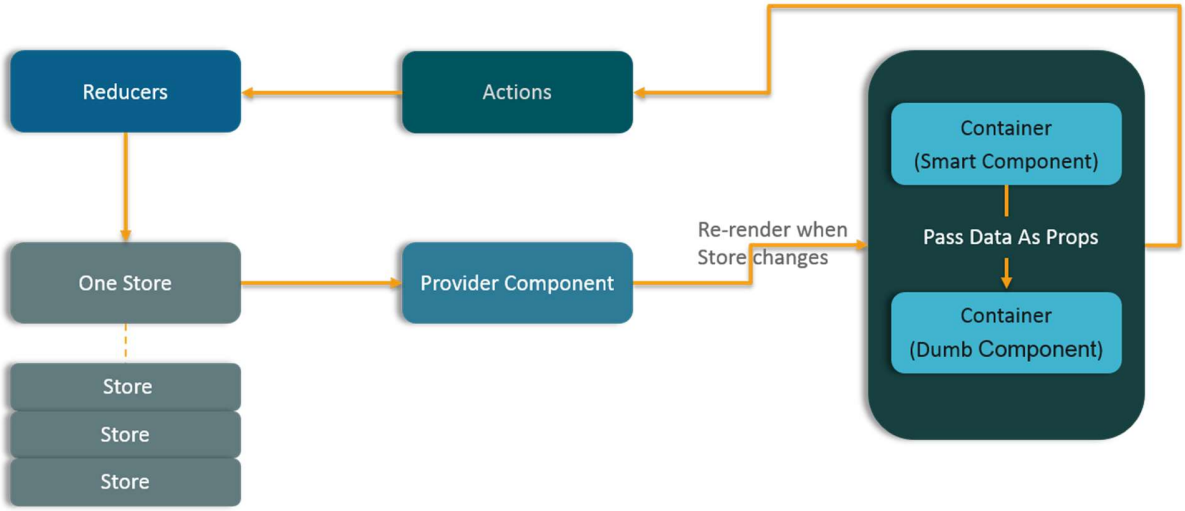
39. List down the components of Redux.

Redux is composed of the following components:

i. **Action** – It's an object that describes what happened.
ii. **Reducer** – It is a place to determine how the state will change.
iii. **Store** – State/ Object tree of the entire application is saved in the Store.
iv. **View** – Simply displays the data provided by the Store.

In case you are facing any challenges with these React interview questions, please comment on your problems in the section below.

40. Show how the data flows through Redux?



41. How are Actions defined in Redux?

Actions in React must have a type property that indicates the type of ACTION being performed. They must be defined as a String constant and you can add more properties to it as well. In Redux, actions are created using the functions called Action Creators. Below is an example of Action and Action Creator:

```
function addTodo(text) {
        return {
                type: ADD_TODO,
                 text
    }
}
```

42. Explain the role of Reducer.

Reducers are pure functions which specify how the application's state changes in response to an ACTION. Reducers work by taking in the previous state and action, and then it returns a new state. It determines what sort of update needs to be done based on the type of the action, and then returns new values. It returns the previous state as it is, if no work needs to be done.

43. What is the significance of Store in Redux?

A store is a JavaScript object which can hold the application's state and provide a few helper methods to access the state, dispatch actions and register listeners. The entire state/ object tree of an application is saved in a single store. As a result of this, Redux is very simple and predictable. We can pass middleware to the store to handle the processing of data as well as to keep a log of various actions that change the state of stores. All the actions return a new state via reducers.

44. How is Redux different from Flux?

Flux vs Redux	
Flux	**Redux**
1. The Store contains state and change logic	1. Store and change logic are separate
2. There are multiple stores	2. There is only one store
3. All the stores are disconnected and flat	3. Single store with hierarchical reducers
4. Has singleton dispatcher	4. No concept of dispatcher
5. React components subscribe to the store	5. Container components utilize connect
6. State is mutable	6. State is immutable

In case you are facing any challenges with these React interview questions, please comment on your problems in the section below.

React Interview Questions

45. What are the advantages of Redux?

Advantages of Redux are listed below:

- **Predictability of outcome –** Since there is always one source of truth, i.e. the store, there is no confusion about how to sync the current state with actions and other parts of the application.
- **Maintainability –** The code becomes easier to maintain with a predictable outcome and strict structure.
- **Server-side rendering –** You just need to pass the store created on the server, to the client side. This is very useful for initial render and provides a better user experience as it optimizes the application performance.
- **Developer tools –** From actions to state changes, developers can track everything going on in the application in real time.
- **Community and ecosystem –** Redux has a huge community behind it which makes it even more captivating to use. A large community of talented individuals contribute to the betterment of the library and develop various applications with it.
- **Ease of testing –** Redux's code is mostly functions which are small, pure and isolated. This makes the code testable and independent.
- **Organization –** Redux is precise about how code should be organized, this makes the code more consistent and easier when a team works with it

46. What is React Router?

React Router is a powerful routing library built on top of React, which helps in adding new screens and flows to the application. This keeps the URL in sync with data that's being displayed on the web page. It maintains a standardized structure and behavior and is used for developing single page web applications. React Router has a simple API.

47. Why is switch keyword used in React Router v4?

Although a ** ** is used to encapsulate multiple routes inside the Router. The 'switch' keyword is used when you want to display only a single route to be rendered amongst the several defined routes. The **<switch>** tag when in use matches the

typed URL with the defined routes in sequential order. When the first match is found, it renders the specified route. Thereby bypassing the remaining routes.

48. Why do we need a Router in React?

A Router is used to define multiple routes and when a user types a specific URL, if this URL matches the path of any 'route' defined inside the router, then the user is redirected to that particular route. So basically, we need to add a Router library to our app that allows creating multiple routes with each leading to us a unique view.

```
<switch>
    <route exact path='/' component={Home}/>
    <route path='/posts/:id' component={Newpost}/>
    <route path='/posts'   component={Post}/>
</switch>
```

49. List down the advantages of React Router.

Few advantages are:

i. Just like how React is based on components, in React Router v4, the API is *'All About Components'*. A Router can be visualized as a single root component (**<BrowserRouter>**) in which we enclose the specific child routes (**<route>**).
ii. No need to manually set History value: In React Router v4, all we need to do is wrap our routes within the **<BrowserRouter>** component.
iii. The packages are split: Three packages one each for Web, Native and Core. This supports the compact size of our application. It is easy to switch over based on a similar coding style.

50. How is React Router different from conventional routing?

Conventional Routing vs React Routing

Topic	Conventional Routing	React Routing
PAGES INVOLVED	Each view corresponds to a new file	Only single HTML page is involved

URL CHANGES	A HTTP request is sent to a server and corresponding HTML page is received	Only the History attribute is changed
FEEL	User actually navigates across different pages for each view	User is duped thinking he is navigating across different pages

51. What is the difference between a controlled and uncontrolled component in React?

React interview questions – Controlled and uncontrolled components

In React, a controlled component is a component that has its state controlled by the parent component. The parent component passes the state as props to the controlled component and also handles any changes to the state via callback functions. The controlled component only renders the received props and does not have its own state.

An uncontrolled component, on the other hand, maintains its own internal state and updates it using DOM events. The component directly updates the DOM and does not rely on the parent component to pass and update the state.

An example of a controlled component is a form input that receives its value from the parent component as a prop and updates the parent component's state via a callback function when the input is changed. An uncontrolled component would be a form input that maintains its own internal state and updates the value directly when the input is changed, without the need for a callback function.

In general, controlled components are considered to be more predictable and easier to debug than uncontrolled components. They also make it easier to implement complex validation and error handling.

52. How do you handle forms in React?

Handling forms in React can be done in a few different ways, but the most common approach is to create a controlled component for the form and its inputs. A controlled component is a component that has its state controlled by the parent component. The parent component passes the state as props to the controlled component and also handles any changes to the state via callback functions.

Here is an example of how to handle a simple form with two input fields (username and password) in a controlled component:

1. First, define the initial state of the form in the parent component's constructor. For example:

```
</pre>
constructor(props) {
super(props);
this.state = {
  username: '' ,
  password: '',
 };
}
<pre>
```

2. Next, create callback functions for each input field that updates the corresponding state property when the input value changes. For example:

```
</span>
handleUsernameChange = (event) =
this.setState({username: event.target.value});
}
handlePasswordChange = (event) => {
this.setState({password: event.target.value});
}
```

3. Pass the state properties and callback functions as props to the controlled form component. For example:

```
<Form
```

```
username={this.state.username}
password={this.state.password}
handleUsernameChange={this.handleUsernameChange}
handlePasswordChange={this.handlePasswordChange}
/>
```

4. In the controlled form component, use the passed-in props to set the value and onChange attributes of each input field. For example:

```
<input
type="text"
value={props.username}
onChange={props.handleUsernameChange}
/>
<input
type="password"
value={props.password}
onChange={props.handlePasswordChange}
/>
```

1. Finally, in the parent component's form submit callback function, you can access the form data from the component's state and handle the form submission as necessary.

Alternatively, you can use third-party libraries such as Formik, or the new hooks in react, useState and useEffect, to handle forms in a more efficient way.

It's important to note that when you are creating forms in React, you should also validate the input values, and display appropriate error messages to users.

53. Explain the concept of a Higher Order Component (HOC) in React.

In React, a Higher Order Component (HOC) design is used to reuse component logic. It is a function that accepts a component as an argument and outputs a new component that extends the capabilities of the input component. Without having to write duplicate code, HOCs can be used to add shared features, like authentication and data retrieval, to various components. All of the original component's props as well as any extra props supplied to the HOC are transferred to the wrapped

component. HOCs are a potent method for composing and enhancing pre-existing components without changing their original source code.

54. What is the purpose of the mapStateToProps function in Redux?

The mapStateToProps function in Redux is a way for a component to access the current state of the store and update its props accordingly. It does this by taking the current state of the store as an argument and returning an object that maps the state to the props of the component. The returned object is then passed to the component as props, allowing it to access the state and re-render when the state changes. This function is typically defined as a separate function outside of the component and is passed as an argument to the connect function, which is used to connect the component to the store.

55. How do you handle routing in a React application?

In a React application, routing is typically handled using a library such as React Router. React Router allows you to define specific routes for different parts of your application and map them to specific components. When the user navigates to a specific route, the corresponding component is displayed on the page.

For example, you could have a route for the homepage that maps to a “Home” component and a route for a user’s profile that maps to a “Profile” component. When the user navigates to the “/” route, the Home component would be displayed, and when they navigate to the “/profile” route, the Profile component would be displayed.

To use React Router in a React application, you’ll need to install it, import it into your application, and define your routes and the components they map to. Here’s an example of how you might set up React Router in a simple React application:

```
1 import { BrowserRouter as Router, Route } from "react-router-dom";
2 import Home from “./components/Home";
3 import profile from “./components/Profile”;
```

```
function App() {
return (
<Router>
   <Route exact path="/" component={Home} />
<Route path="/profile"” component={Profile} />
</Router>
);
}
```

In this example, the Router component is used to wrap the entire application and the Route component is used to define the specific routes and the components they map to. The exact prop is used to ensure that only the exact path is matched and not any subpaths.

React interview questions – Server-side and client-side rendering

56. Explain the difference between server-side rendering and client-side rendering in React.

In a React application, there are two main ways to render the components: server-side rendering (SSR) and client-side rendering (CSR).

Server-side rendering (SSR) is when the initial render of a React application is done on the server. The server generates the HTML for the initial state of the application and sends it to the browser. When the JavaScript bundle loads, React takes over and the application continues to function as a SPA (Single-Page Application) on the client side.

This approach has a few benefits such as:

Improved performance for search engines and users on slow connections

Faster time-to-first-byte

Better accessibility for users who have JavaScript disabled

Client-side rendering (CSR) is when the React application is rendered entirely in the browser, using JavaScript. The browser requests the JavaScript bundle from the server and then renders the components on the client side. This approach has the

benefit of faster load times for users on fast connections and a more responsive user interface.

In general, CSR is the simpler option to implement and more popular, but SSR is a good choice for certain use cases, such as when SEO is a primary concern, or when the app is targeting users on slow internet connections.

It is also worth noting that, it is possible to have a hybrid approach between SSR and CSR which is called isomorphic or universal rendering. This approach allows to leverage the benefits of both SSR and CSR.

57. How do you handle data persistence in a React application?

In a React application, data persistence can be handled using a variety of methods, including:

1. Local storage: This allows you to store key-value pairs in the browser's local storage, which can be retrieved even after the user closes the browser or restarts their device.
2. Cookies: Cookies are small pieces of data that are stored in the user's browser and can be accessed by the website on subsequent visits.
3. IndexedDB: It's a low-level API for client-side storage of large amounts of structured data, including files/blobs.
4. Web SQL Database: This is a deprecated technology for storing data in a client-side database using SQL.
5. Server-side storage: You can also store data on a remote server using an API or a database such as MySQL, MongoDB, etc.
6. Redux or Mobx: State management libraries like Redux or Mobx can be used to manage and persist application state across different components and sessions.

Which one to use depends on your specific use case and requirements.

58. What is the difference between a stateless component and a stateful component in React?

In React, a component can be either stateless or stateful. The main difference between the two is how they manage and update their data.

A stateless component, also known as a “dumb” or “presentational” component, is a component that does not maintain its own internal state. It receives data and callbacks through props (short for properties) and only renders the UI based on those props. Stateless components are typically used for simple, presentational elements that don’t need to handle any complex logic or internal state updates. They are simple functions that take props and return JSX.

A stateful component, also known as a “smart” or “container” component, is a component that maintains its own internal state. It can handle internal state updates and side effects, and may also manage the state of other child components. Stateful components are typically used for more complex elements that need to handle user interactions, API calls, or other logic. They are class components that extend React.Component.

In general, it is recommended to use stateless components as much as possible to keep the application simple and easy to understand. Stateful components should only be used when it is necessary to manage state or handle complex logic.

59. Explain the concept of a Pure Component in React.

A “pure component” in React is a component that updates only when its properties or state have changed. In contrast, a “non-pure component” re-renders each time the parent component re-renders, regardless of whether its props or state have changed. Pure components are more productive since they do not needlessly re-render. By extending React, a component in React can be made pure. React is substituted by PureComponent. Component. This prompts the shouldComponentUpdate method, which decides whether or not to re-render, to provide an automatic shallow comparison of the component’s props and state.

60. How do you handle optimization in a large React application?

There are several techniques that can be used to optimize a large React application:

Use the React Developer Tools to identify and fix performance bottlenecks. The React Developer Tools allow you to track the performance of individual components and identify which components are causing the most re-renders.

Use the shouldComponentUpdate lifecycle method to prevent unnecessary re-renders. This method allows you to control when a component should update based on its props and state.

Use PureComponent and memo instead of Components. These are more efficient alternatives to React.Component that only re-render when props or state have changed.

Use the useEffect hook to handle side effects. This hook allows you to run side effects, such as network requests, after a component has rendered.

Use the useMemo hook to memoize expensive calculations. This hook allows you to cache the results of expensive calculations and only recalculate them when the inputs have changed.

Lazy loading: Lazy loading is a technique where you only load the components that are needed for the current view. This can greatly improve the performance of your application.

Code splitting: Code splitting is a technique where you split your application into smaller chunks of code that are loaded on demand. This can greatly improve the performance of your application.

Optimize the loading time of your application by using techniques like code minification, compression, and caching.

It's also important to keep in mind that performance optimization is an ongoing process and you should regularly check and optimize your application as it grows.

61. What is the purpose of the combineReducers function in Redux?

The combineReducers function in Redux is used to combine multiple individual reducers into a single root reducer. In a Redux application, the state is managed by a single store and each piece of the state is managed by a specific reducer. The combineReducers function takes an object whose keys correspond to the keys in the state, and whose values are the individual reducers that will manage those parts of the state.

The combineReducers function is used to compose the different reducers that handle different parts of the state into a single root reducer. This root reducer is then passed to the createStore function to create the Redux store.

The combineReducers function is also useful for structuring and organizing your code in a more modular way, as it allows you to separate the logic for different parts of the state into different files and functions.

The combineReducers function is not mandatory to use, but it makes it easier to split the application state and the reducer functions that handle it in a more modular way and also it helps to avoid name collision if you have multiple reducer functions that handle a specific part of the state.

62. How do you handle error handling in a React application?

A React application can handle errors in a few different ways. To handle problems that happen during rendering, one typical solution is to utilise a try-catch block within a component's lifecycle functions, such as componentDidCatch. Use the Error Boundaries feature to design a component that detects errors that happen inside its child components as an alternative strategy. In addition, you may manage errors that

happen inside a functional component by combining the useEffect hook with a try-catch block.

There are several ways that an error can be handled in a React application. Using a try-catch block in a component's lifecycle routines, like componentDidCatch, is a common way to manage rendering-related issues. An alternate approach is to construct a component that detects problems that occur inside its child components using the Error Boundaries feature. Additionally, you may control errors that occur inside a functional component by combining a try-catch block with the useEffect hook.

63. What is the difference between a smart component and a dumb component in React?

Smart Component	Dumb Component
Has state and logic	Has no state or logic
Can communicate with other components through props and callbacks	Can only receive props and emit events
Can make API calls or perform complex logic	Can only display data passed to it
Typically class-based components	Typically functional components

64. Explain the concept of a Render Prop in React.

A render prop in React is a technique for conveying component logic. Instead of using a component's props to communicate data and behaviour, a render prop is a function that a component utilises to select what to render. The "provider" component makes the render prop available, but the "consumer" component is the one that uses it. With this approach, component flexibility and reuse are enhanced.

65. How do you handle testing in a React application?

React Testing Library is a great tool for testing React components. It's a set of helpers that let you test React components without relying on their implementation details. This approach makes refactoring a breeze and also nudges you towards best practices for accessibility. With components, the distinction between a "unit" and "integration" test can be blurry. If you're testing a form, should its test also test the buttons inside of it? Or should a button component have its own test suite? Should refactoring a button ever break the form test? Jest is a JavaScript test runner that lets you access the DOM via jsdom, which is an approximation of how the browser works. Jest provides a great iteration speed combined with powerful features like mocking modules and timers so you can have more control over how the code executes. It's a great tool for running tests on React apps. You can also use a tool like BrowserStack's Real Device Cloud to run end-to-end tests on real devices. Cross browser compatibility testing can also be done with a tool like BrowserStack Live.

66. What are the different ways to pass data between components in React?

Data is passed as a property on the element when utilising props to communicate between components. For instance, you can do the following to send a user object from a parent component to a child component: The props object, props.user, can then be used in the child component to access the data. Data can be passed via the component tree using context rather than having to go through each level of the tree. You must establish a context object containing a Provider and a Consumer in order to use context.

The components that require the data are served by the Provider, and the components that need to access the data are served by the Consumer. Data that is local to a component can be stored in a state, which allows for the tracking of data across time. Through the useState Hook, state in a component can be obtained. The current state value and a function to update the state value are the two items of an array that the useState Hook returns after receiving an initial value.

67. Explain the concept of a Portal in React.

A React component can be rendered outside of the DOM hierarchy of its parent component using React Portals. As an example, you could render components in a modal dialogue box, hover card, loader, or popup message, which would be in a "different place" than their parent component. The ReactDOM.createPortal() method, which accepts a React element (child) and a DOM element as inputs, is used to create React portals (container). The container is the DOM element that the child component should be rendered into, and the child component is any renderable React child, such as an element, string, or fragment. For items that must appear above all other elements, such as profile hovercards, modal dialogue boxes, and tooltips, portals are frequently utilised.

68. How do you handle performance optimization in a React application?

There are several ways to optimize the performance of a React application, including:

- Using the shouldComponentUpdate lifecycle method to prevent unnecessary re-renders of components.
- Using React's built-in PureComponent or implementing a custom shouldComponentUpdate method to optimize performance for functional components.
- Using the React developer tools to profile the application and identify performance bottlenecks.
- Using the React.memo method for functional components
- Using React's Context API instead of props drilling.
- Using the useEffect hook to handle side effects in functional components.
- Using the useCallback and useMemo hooks to prevent unnecessary re-renders and improve performance.
- Lazy loading of components and code splitting.
- Minimizing the number of DOM updates by using the key prop when rendering a list of items.
- Using the useReducer hook to manage state updates instead of useState
- Using a virtualized list library like react-virtualized, react-window etc.
- It's always a good idea to test performance with real-world use cases and user interactions before and after making any optimization.

React interview questions – functional components and class components

69. What is the difference between a functional component and a class component in React?

In React, a functional component is a plain JavaScript function that takes in props and returns a React element. A class component is a JavaScript class that extends React.Component and has a render method that returns a React element.

One key difference between the two is that a class component can have local state and lifecycle methods, while a functional component cannot. However, starting with React 16.8, functional components can also have a state using hooks.

Functional components are considered simpler, easier to understand and test, and have better performance than class components. Class components are useful when you need to use lifecycle methods or the local state.

70. Explain the concept of a Context in React.

In React, context is a way to share data that is considered “global” for a component tree. It allows you to pass data through the component tree without having to pass props down manually at every level.

A component that needs to access the context data can consume it by using the useContext hook or the Consumer component. To make the context available to a component, a parent component needs to provide it using the Provider component.

Context is often used for data that is required by many components in an application, such as the currently authenticated user, the current locale, or the theme.

It should be noted that context should be used sparingly, as it can make your components more difficult to reason about and test. If possible, it’s better to pass props down the component tree manually.

71. How do you handle asynchronous data loading in a React application?

In a React application, asynchronous data loading can be handled using a technique called “lifting state up”. This involves moving the state that manages the loading and error state of the data to a common ancestor component, and passing down the necessary callbacks and state through props to the components that need to use the data.

One popular way to handle async data loading is to use the useEffect hook in combination with fetch or a library like axios to load data in a component after it has been rendered. The useEffect hook allows you to synchronize a component with an external system, such as a server, by running a side effect (the data loading) after the component has rendered. The hook takes a callback function that contains the effect, and an array of dependencies.

Another way is to use a library like redux-thunk or redux-saga to handle the async request and store the data in the store/state. These libraries provide an easy way to handle async actions and keep the component state clean.

In either case, it’s important to keep an eye on the component’s state and update it properly with the loaded data.

A simple example of loading data asynchronously in a React component using useEffect and fetch:

```
1 import { useState, useEffect } from ‘react’;
2 function MyComponent() {
3 const [data, setData] = usestate(null);
4 const [error, setError] = usestate(null);
5 const [loading, setloading] = usestate(true);
6 useEffect(() => {
7 async function fetchData() {
8 try {
9 const response = await fetch('https://example.com/data’);
10const json = await response.json();
11setData(json);
12setLoading(false);
13} catch (error) {
14setError(error) {
```

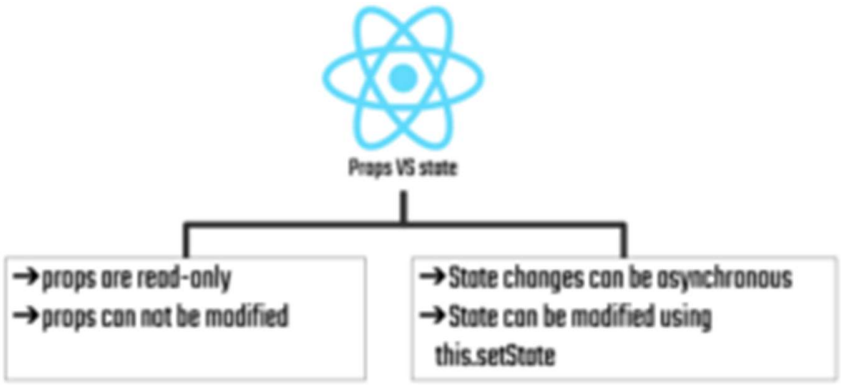
```
setLoading(fase);
    }
}
fetchData();
}, []);
if (loading) {
return <p>Loading...</p>;
}
```

```
if (error) {
return <p>Error: {error.message}</p>;
}
return <p>Data: {JSON.stringify(data)}</p>;
}
```

It's important to note that this is a simple example, and in a real-world application you may need to handle more complex cases such as pagination, caching and handling different types of errors.

72. What is the difference between state and props in React?
State and Props are both concepts in React that are used to store and manipulate data within a React component. The main difference between the two is that State is used to store and manage the data that is local and specific to a component, while Props are used to pass data from a parent component to its child components.

State is considered to be dynamic, meaning that it can change over time as a result of user interactions or other events. On the other hand, Props are considered to be static and cannot be changed by the child component. Instead, the parent component is responsible for updating the value of its Props and passing the updated value to the child component.



In summary, State is used to manage the internal state of a component, while Props are used to pass data from a parent component to its child components.

73. Explain the concept of a Hook in React.

During an interview, you can explain Hooks in the following way:

“Hooks are a new feature in React that allows us to add state and other React features to functional components. They were introduced in React 16.8 and have since become a popular way to manage state and side effects in functional components. Hooks are named functions that start with the word use and allow us to reuse stateful logic across components without having to write a class component. For example, the useState Hook allows us to add state to a functional component and the useEffect Hook lets us perform side effects like data fetching or updating the document title. Hooks make our code more reusable, easier to understand, and easier to test.”

React interview questions – localization in react

74. How do you handle localization in a React application? Handling localization in a React application typically involves creating translated versions of your text content and displaying the appropriate version based on the user’s preferred language.

One way to handle localization in a React application is to use a library such as react-i18next. This library provides a set of tools for internationalization and localization, including the ability to define translation keys and their corresponding translations, as well as providing a way to switch between languages at runtime.

To use react-i18next in your React application, you would install it using npm and then configure it in your index.js file. After that, you can use the useTranslation Hook to access the translations in your components.

Here's how you can explain localization in a React application in an interview:

"Localization in a React application involves creating translated versions of text content and displaying the appropriate version based on the user's preferred language. To handle localization in a React application, I would use a library such as react-i18next, which provides a set of tools for internationalization and localization. With this library, I would define translation keys and their corresponding translations and provide a way to switch between languages at runtime. In my components, I would use the useTranslation Hook to access the translations and display the appropriate version of the text content."

75. What is the difference between a static component and a dynamic component in React?

In React, a static component is a component that is defined with a fixed set of properties or attributes and does not change during its lifecycle. A static component is defined using a simple JavaScript function that returns a tree of elements that represent the component's UI.

A dynamic component, on the other hand, is a component that can change its properties, state or behavior based on interactions with the user or events that occur within the application. A dynamic component is typically defined using a class component or a functional component with the useState or useEffect Hooks.

Here's an example of a static component:

```
function welcome(props) {
return <h1>Hello, {props.name}</h1>;
}
```

76. Explain the concept of a Renderless Component in React.

A Renderless Component in React is a component that doesn't render any HTML elements to the DOM, but instead exposes data and methods to other components through props and callbacks. The purpose of a renderless component is to encapsulate logic that can be reused across multiple components and keep the component tree lean and flexible. The other components that consume the logic provided by a renderless component can then render the HTML elements they need based on the information and functionality they receive from the renderless component. This approach separates the logic and presentation concerns, making the code easier to maintain and test.

77. How do you handle server-side rendering in a React application?

Server-side rendering (SSR) in React involves rendering your React components on the server and sending the resulting HTML to the client. This provides a number of benefits, including improved performance and search engine optimization (SEO). To implement SSR in a React application, you can use a library like Next.js or Razzle, which provide an easy-to-use framework for handling SSR. Alternatively, you can use the ReactDOMServer API to manually render your components on the server. The key steps in the process are:

1. Setting up your server to handle incoming requests and render the appropriate components.
2. Rendering the components on the server using ReactDOMServer.renderToString or ReactDOMServer.renderToStaticMarkup.
3. Sending the resulting HTML to the client as part of the response.
4. Hydrating the components on the client so that they can be interactively controlled by the user.

It's worth noting that SSR comes with some trade-offs and additional complexity, so it's important to carefully consider whether it's the right choice for your application.

78. What is the difference between a presentational component and a container component in React?

In React, a presentational component (also known as a dumb component) is a component that focuses on UI (user interface) and presentation of the data, while a container component (also known as a smart component) is a component that focuses on how the data is being managed and provides the data for the presentational components.

A presentational component is typically written as a functional component, receiving data as props and returning a view, while a container component is typically written as a class component, handling data management and state changes, and passing down the data to the presentational components as props.

The separation of concerns between the two types of components allows for better code organization, maintenance, and testing.

79. Explain the concept of a Custom Hook in React.

A Custom Hook in React is a JavaScript function that lets you extract state logic and behavior out of a component, and reuse it across multiple components. Custom Hooks allow you to abstract away state and behavior that is common across your application into a reusable piece of code.

Custom Hooks are named with the prefix use (e.g. useForm, useFetch), and can call other Hooks as well as your own custom Hooks. They have the same rules as Hooks and can only be called at the top level of your component or your own custom Hooks.

Custom Hooks can receive arguments and return values, just like a regular function, but they also have the ability to manage state and perform side-effects. By abstracting state and behavior into a Custom Hook, you can improve the readability and maintainability of your code.

Examples of things you can build with Custom Hooks include:

- Data fetching

- Managing state updates
- Handling form submissions
- Implementing animations and transitions
- And many more.

Using Custom Hooks can make your components cleaner, more reusable, and easier to test, which makes them a powerful tool in your React toolkit.

80. How do you handle accessibility in a React application?

Handling accessibility in a React application involves making sure that your application can be used by as many people as possible, including those with disabilities. This can be achieved through various techniques, including:

1. Semantic HTML: Use semantic HTML elements, such as <button>, <nav>, and <header>, to clearly define the structure and purpose of your content.
2. Accessible Props: Use accessible props, such as aria-label, role, and tabIndex, to provide additional information to assistive technologies, such as screen readers.
3. Keyboard Navigation: Ensure that all functionality can be accessed using a keyboard, and that keyboard focus is managed correctly.
4. Color Contrast: Make sure that the contrast between the text and the background is high enough to be readable by people with color blindness or low vision.
5. Alternative Text: Provide alternative text for images, videos, and other non-text elements to ensure that information is accessible to screen reader users.
6. Screen Reader Testing: Test your application with screen readers and other assistive technologies to identify and fix any accessibility issues.

It is important to note that accessibility is a continuous process and should be considered throughout the development of your React application. The use of tools, such as linting rules and accessibility testing tools, can also help ensure that your application is accessible.

81. What is the difference between a reducer and an action in Redux?

In Redux, a reducer and an action are two different but related concepts.

An action is a plain JavaScript object that describes the change that should be made to the state of the application. It has a type property that defines the type of action being performed, and a payload property that provides any additional data needed to perform the action. Actions are dispatched from the application to the Redux store, which then passes the action to the reducers.

A reducer is a pure function that takes the current state of the application and an action, and returns the next state of the application. The reducer is responsible for handling the actions and updating the state accordingly. It should not perform any side-effects, such as making API calls, but should instead only return the next state.

In summary, actions describe what should change, while reducers define how the state should change in response to the actions.

82. Explain the concept of a Higher Order Component (HOC) in React and when to use it.

A Higher Order Component (HOC) in React is a function that takes a component as an argument and returns a new component with additional props. The purpose of a HOC is to reuse logic across multiple components. An HOC is not a “part” of React, it’s a pattern in React for reusing component logic.

Use a HOC when you need to:

- Share common logic between multiple components, such as data fetching or authorization.
- Abstract state and behavior that can be reused across your application, into a reusable HOC.
- Render a component within another component and pass props to the wrapped component.

Examples of HOCs include the withRouter HOC from react-router and the connect HOC from react-redux.

83. How do you handle data validation in a React application?

Data validation in a React application can be handled in a variety of ways, including:

1. PropTypes: React provides a built-in library called PropTypes that allows you to specify the expected data types for your component's props. PropTypes will validate the props passed to your component at runtime, and will log warnings in the browser console if any props are of the wrong type.
2. Custom validation functions: You can write custom validation functions to check the validity of your data. These functions can be called inside your component, and can be used to set error messages or update the state to indicate invalid data.
3. Third-party libraries: There are several third-party libraries available for data validation in React, such as yup, joi, or zod. These libraries provide a more powerful and flexible way to validate data, and often provide a more user-friendly way to report errors.

Regardless of the method you choose, it's important to handle data validation in your React application to ensure that the data being processed is in the correct format and meets the required constraints. Validation helps to catch potential errors early in the development process and prevent bugs from affecting the end-user experience.

84. What is the difference between a synchronous action and an asynchronous action in Redux?

In Redux, an action is a plain JavaScript object that describes the change in the state of the application. Actions can be either synchronous or asynchronous.

A synchronous action is an action that is dispatched and immediately processed by the Redux store. The store updates the state, and the updated state is immediately available for consumption by the components.

An asynchronous action, on the other hand, is an action that is dispatched but takes some time to complete. Asynchronous actions are typically used when performing network requests or doing other operations that take time. These actions cannot be immediately processed by the Redux store, so they require additional logic to handle their completion.

In a Redux application, asynchronous actions are often handled using middleware, such as redux-thunk or redux-saga, which allow for the dispatching of actions that represent the start and completion of asynchronous operations. These middleware provide a way to handle the asynchrony of the operation and ensure that the state is updated appropriately once the operation is complete.

85. Explain the concept of a Virtual DOM in React.

A Virtual DOM (Document Object Model) is a lightweight in-memory representation of the actual DOM in a web page. In React, the Virtual DOM acts as an intermediary between the React component's render output and the browser's DOM.

When a React component's state changes, React updates the Virtual DOM, instead of directly updating the actual DOM. This is more efficient because updating the Virtual DOM is faster than updating the actual DOM, as it can calculate the difference between the previous and current render output, and only update the parts that have changed.

React then takes the updated Virtual DOM and uses it to update the actual DOM, minimizing the amount of work that needs to be done in the actual DOM and improving the overall performance of the application.

In summary, the Virtual DOM in React acts as an optimization to increase the speed and efficiency of updates to the user interface.

86. How do you handle browser compatibility in a React application?

To handle browser compatibility in a React application, you can use various techniques such as:

1. Polyfills: To support older browsers, you can use polyfills, which are JavaScript libraries that emulate missing features in older browsers.
2. Browser detection: You can use libraries like browser-detect to detect the user's browser and its version, and adjust your code accordingly.
3. Feature detection: Instead of relying on browser detection, you can use feature detection to check if a specific feature is supported by the user's browser before using it.
4. CSS Reset: You can use CSS resets like normalize.css to make sure that all browsers display the styles in a consistent way.
5. Testing: Regular testing in different browsers and devices is essential to catch any compatibility issues early in the development process.

By using these techniques, you can ensure that your React application runs smoothly across different browsers and devices.

87. What is the difference between a stateful component and a stateless component in React?

In React, a stateful component, also known as a "smart" or "container" component, is a component that maintains its own internal state, typically via the useState or this.state hooks. It may also manage data that is passed down to it as props from other components, and it may use lifecycle methods, such as componentDidMount, to fetch data or perform other side effects.

On the other hand, a stateless component, also known as a "dumb" or "presentational" component, is a component that only receives data via props and does not maintain its own internal state. It simply renders the data it receives in a visually appealing way and does not manage or manipulate it in any way. These components are considered "pure" because they are only concerned with the rendering of the data and do not have side effects.

The key difference between the two is the way they manage and manipulate data. Stateful components have their own internal state and are responsible for managing

and updating it, whereas stateless components simply receive data via props and render it without any data manipulation.

88. Explain the concept of a Thunk in Redux.

A Thunk in Redux is a function that returns another function instead of a plain action object. It's used to perform asynchronous operations and dispatch multiple actions. Thunks allow you to write action creators that return a function instead of an action. This can be useful for performing asynchronous operations, such as API calls, and dispatching multiple actions, such as one to indicate that the API call has started and another to indicate that it has finished. The inner function receives the store's dispatch method as an argument, which can be used to dispatch actions at any point in the future. Thunks are typically implemented using a middleware, such as the redux-thunk middleware.

89. How do you handle security in a React application?

Handling security in a React application involves multiple steps, including:

Input validation: Validate all user inputs on the client and server side to prevent any malicious data from being processed.

Authenticating and authorizing users: Use a secure authentication mechanism such as JSON Web Tokens (JWT) to ensure that only authorized users can access sensitive data.

Storing sensitive data securely: Do not store sensitive information such as passwords and credit card numbers in local storage, use encrypted storage instead.

Implementing HTTPS: Use HTTPS to ensure secure communication between the client and server and protect against network attacks such as man-in-the-middle attacks.

Keeping dependencies up-to-date: Regularly update React and its dependencies to patch any known security vulnerabilities.

Using Content Security Policy (CSP): Implement a Content Security Policy (CSP) to restrict the types of resources that can be loaded in a React application and prevent cross-site scripting (XSS) attacks.

Regular security audits: Conduct regular security audits to identify and address potential security issues in a timely manner.

90. What is the difference between a function component and a class component in React?

In React, there are two main types of components: function components and class components.

Function Components, also known as “stateless” or “functional” components, are JavaScript functions that accept props as input and return React elements as output. They are simple, easy to understand and test, and are usually used for presentational components that don’t have their own state or lifecycle methods.

Class Components, on the other hand, are JavaScript classes that extend the React.Component base class. They are used for creating components that have a state, or need to access lifecycle methods such as componentDidMount or shouldComponentUpdate. Class components are more complex than function components, but provide more advanced features.

In summary, the main difference between function and class components in React is that function components are simpler, more straightforward, and easier to understand, while class components are more powerful and provide more advanced features, but are also more complex.

91. Explain the concept of a Provider in React-Redux.

The “Provider” in React-Redux is a higher-order component that wraps your React application and provides it with the ability to access the Redux store. It allows you to pass the store down to your components using context, without having to manually pass it down as props through every level of the component tree.

By using the Provider, you ensure that all of your components can subscribe to the store and dispatch actions to modify its state. In other words, the Provider acts as a bridge between your React components and your Redux store, making the store accessible to all components in your application.

92. How do you handle code splitting in a React application?

Code splitting in React can be handled using the following approaches:

Dynamic Imports: Dynamic imports allow you to load a component lazily only when it is needed. This is done using the import() syntax and provides a way to split code into smaller chunks that can be loaded on demand.

```
import React, { Suspense } from 'react’;
const LazyComponent = React.lazy(() => import('./LazyComponent'));
function App() {
return (
<div>
<Suspense fallback={<div>Loading...</div>}>
<LazyComponent />
</Suspense>
</div>
);
}
```

```
import React, { lazy, Suspense } from 'react’;
import { Route } from 'react-router-dom’;
const Home = lazy(() => import('./Home'));
const About = lazy(() => import('./About'));
function App() {
return (
<div>
<Suspense fallback={<div>Loading...</div>}>
<Route exact path="/" component={Home} />
<Route path="/about" component={About} />
</Suspense>
</div>
);
}
```

Webpack Bundle Analyzer: This is a tool that provides a visual representation of the code and its size. You can use this tool to identify the large chunks of code that can be split into smaller chunks and loaded lazily.

By using these approaches, you can effectively handle code splitting in a React application and improve its performance by reducing the initial loading time and only loading the required code on demand.

93. What is the difference between a connected component and a component in React-Redux?

Component	Connected Component (Higher Order Component)
Definition	A plain React component that receives props and returns a tree of React elements.
Usage	Used to display UI elements, manage local component state and pass props to child components.
Example	A button, a form, a card, etc.

Connected components are higher-order components that are wrapped around plain components to provide them access to the Redux store. Connected components are used to access the state of the store and dispatch actions, whereas plain components are used to manage UI elements and local component state.

94. Explain the concept of a Sagas in Redux.

A Saga in Redux is a way to manage side effects (e.g. asynchronous operations like data fetching and impure operations like accessing the browser cache) in a Redux application. It is implemented as a middleware using generator functions in JavaScript and runs in the background, separate from the main thread of your application, watching for actions dispatched to the store. When a specific action is detected, the Saga can perform various tasks and trigger additional actions as needed, updating the store based on the results of the asynchronous operations. The key benefit of using Sagas is that they make it easier to reason about, test, and manage the flow of data in your application.

95. How do you handle code optimization in a large React application?

Handling code optimization in a large React application can be achieved through several approaches:

1. Code splitting: This allows you to split your code into smaller chunks that can be loaded on demand, reducing the initial load time of your application.
2. Lazy loading: Lazy loading allows you to load components only when they are required, reducing the amount of code that needs to be loaded and parsed at startup.
3. Use of a bundler such as Webpack: A bundler can help you optimize your code by reducing the size of your JavaScript files, combining multiple files into one, and more.
4. Use of caching: You can cache the data and components that are frequently used in your application to avoid fetching the same data over and over.
5. Use of efficient algorithms and data structures: In order to keep your application fast, it's important to use algorithms and data structures that are optimized for performance.
6. Regular performance monitoring and profiling: Regular performance monitoring and profiling can help you identify performance bottlenecks and areas for improvement in your code.
7. Use of optimization techniques such as memoization: By using techniques such as memoization, you can reduce the number of unnecessary re-renders and computations in your application, improving its overall performance.

96. What is the difference between a React component and a React element?

A React component is a JavaScript class or function that returns a React element. It is a reusable piece of UI that describes a part of the user interface.

A React element, on the other hand, is a plain JavaScript object that represents a DOM node. It is an immutable representation of a DOM node, which can be created using React.createElement or JSX.

In short, a component is a blueprint for creating elements, and an element is an instance of a component.

97. Explain the concept of a Middleware in Redux.

In Redux, a middleware is a software component that sits between the store and the action dispatching process to add additional functionality, such as logging, crash reporting, handling asynchronous actions, etc. It allows you to extend the store's behavior without modifying the store itself. Middlewares are applied using the applyMiddleware method and can be composed together to achieve a desired behavior. When an action is dispatched, it passes through each middleware in the order they were composed, giving the middleware an opportunity to interact with the action before it reaches the store. This provides a way to manipulate actions and state, and to perform complex actions that can span multiple actions.

98. How do you handle internationalization in a React application?

Handling internationalization (i18n) in a React application involves adapting the user interface and content of the application to meet the language and cultural requirements of different locales.

There are several libraries and techniques that can be used to implement internationalization in a React application, including:

1. react-intl: A popular library for internationalizing React applications. It provides components for formatting dates, numbers, and strings, as well as handling pluralization and message extraction.
2. Context API: React's Context API can be used to store the current locale and make it available to the components that need it. The locale can be changed dynamically to switch the language of the application.
3. Custom hooks: Custom hooks can be written to encapsulate the logic for formatting and retrieving messages, and to make it easier to use in components.

Here's an example of how the react-intl library can be used to implement internationalization in a React application:

```
import React from 'react';
import { FormattedMessage, useIntl } from ‘react-intl';
function MyComponent() {
const intl = useintl();
return (
<div>
<p>
   <FormattedMessage id="greeting" defaultMessage="Hello, World!" />
</p>
<p>
{int1.formatDate(new Date(), {
weekday: 'long’,
year: ‘numeric’,
month: 'long’,
day: ‘numeric’,
})}
</p>
</div>
);
}
```

In this example, the useIntl hook is used to access the intl object, which provides internationalization functions like formatDate. The FormattedMessage component is used to display a localized message with the ID greeting.

Implementing internationalization in a React application can greatly improve the user experience for users who speak different languages and are located in different regions. It’s an important consideration for any application that aims to have a global reach.

99. What is the difference between a React component and a React class?

When it comes to an interview, it’s important to understand the difference between React class components and functional components. This knowledge can demonstrate your understanding of React and its components, and it may also show your ability to write efficient and maintainable code.

When asked about class components, you can highlight that they are defined as JavaScript classes that extend the React.Component class, have a render method, and can have additional lifecycle methods and state. You can also provide a simple example to show your understanding of class components.

When asked about functional components, you can emphasize that they are defined as plain JavaScript functions that return the component's JSX markup, and that they can use state and other React features with hooks. You can also give an example to show how to write a functional component that achieves the same functionality as a class component.

Finally, you can explain the trade-offs between class components and functional components, such as that functional components are generally simpler and easier to read, while class components offer more features and flexibility. Showing your ability to weigh the pros and cons of each approach can demonstrate your critical thinking skills and ability to write maintainable code.

100. Explain the concept of a Memoization in React.

In React, memoization is a technique used to optimize the performance of a component by avoiding unnecessary re-renders. It involves caching the results of a component's render so that if the inputs (props) to the component do not change, the cached result can be reused, instead of re-computing the result.

React provides a built-in hook called useMemo for implementing memoization. useMemo takes a function and an array of dependencies as arguments and returns a memoized value. The function is re-executed only if one or more of the dependencies have changed.

Here is an example of how useMemo can be used to memoize an expensive calculation:

```
1 import React, { useMemo } from 'react';
2 function MyComponent({ data }) {
```

```
const memoizedvalue = useMemo(() => {
// Do some expensive calculation
let result = 0;
for (let i = 0; i < data.length; i++) {
result += data[i];
}
return result;
}, [data]);
return <div>The result is {memoizedvalue}</div>;
}
```

In this example, useMemo is used to memoize the result of the expensive calculation performed on data. The calculation will only be re-executed if the value of data changes.

Memoization can greatly improve the performance of a React application by avoiding unnecessary re-renders and re-calculations, especially when dealing with complex or large data structures.