**https://github.com/sudheerj/reactjs-interview-questions#when-to-use-a-class-component-over-a-function-component**

**Q1. How React works? How Virtual-DOM works in React?**

React creates a virtual DOM. When state changes in a component it firstly runs a “diffing” algorithm, which identifies what has changed in the virtual DOM. The second step is reconciliation, where it updates the DOM with the results of diff.

The HTML DOM is always tree-structured — which is allowed by the structure of HTML document. The DOM trees are huge nowadays because of large apps. Since we are more and more pushed towards dynamic web apps (Single Page Applications — SPAs), we need to modify the DOM tree incessantly and a lot. And this is a real performance and development pain.

The Virtual DOM is an abstraction of the HTML DOM. It is lightweight and detached from the browser-specific implementation details. It is not invented by React but it uses it and provides it for free. ReactElements lives in the virtual DOM. They make the basic nodes here. Once we defined the elements, ReactElements can be render into the "real" DOM.

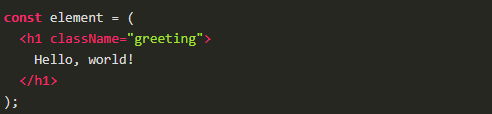
Whenever a ReactComponent is changing the state, diff algorithm in React runs and identifies what has changed. And then it updates the DOM with the results of diff. The point is - it’s done faster than it would be in the regular DOM.

**Q2. What is JSX?**

JSX is a syntax extension to JavaScript and comes with the full power of JavaScript. JSX produces React “elements”. You can embed any JavaScript expression in JSX by wrapping it in curly braces. After compilation, JSX expressions become regular JavaScript objects. This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions. Eventhough React does not require JSX, it is the recommended way of describing our UI in React app.

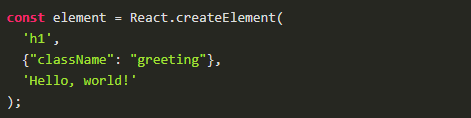
For example, below is the syntax for a basic element in React with JSX and its equivalent without it.

Image for post



Equivalent of the above using React.createElement

Image for post

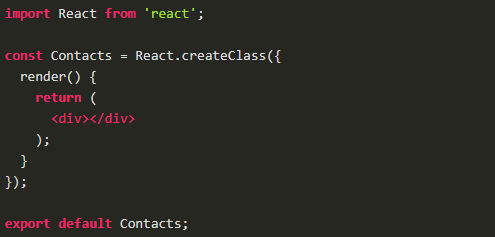


**Q3. What is React.createClass?**

React.createClass allows us to generate component "classes." But with ES6, React allows us to implement component classes that use ES6 JavaScript classes. The end result is the same -- we have a component class. But the style is different. And one is using a "custom" JavaScript class system (createClass) while the other is using a "native" JavaScript class system.

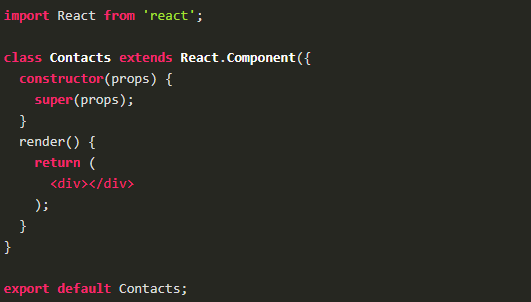
When using React’s createClass() method, we pass in an object as an argument. So we can write a component using createClass that looks like this:

Image for post



Using an ES6 class to write the same component is a little different. Instead of using a method from the react library, we extend an ES6 class that the library defines, Component.

Image for post



constructor() is a special function in a JavaScript class. JavaScript invokes constructor() whenever an object is created via a class.

**Q4. What is ReactDOM and what is the difference between ReactDOM and React?**

Prior to v0.14, all ReactDOM functionality was part of React. But later, React and ReactDOM were split into two different libraries.

As the name implies, ReactDOM is the glue between React and the DOM. Often, we will only use it for one single thing: mounting with ReactDOM. Another useful feature of ReactDOM is ReactDOM.findDOMNode() which we can use to gain direct access to a DOM element.

For everything else, there’s React. We use React to define and create our elements, for lifecycle hooks, etc. i.e. the guts of a React application.

**Q5. What are the differences between a class component and functional component?**

Class components allows us to use additional features such as local state and lifecycle hooks. Also, to enable our component to have direct access to our store and thus holds state.

When our component just receives props and renders them to the page, this is a ‘stateless component’, for which a pure function can be used. These are also called dumb components or presentational components.

From the previous question, we can say that our Booklist component is functional components and are stateless.

Image for post



On the other hand, the BookListContainer component is a class component.

**Q6. What is the difference between state and props?**

The state is a data structure that starts with a default value when a Component mounts. It may be mutated across time, mostly as a result of user events.

Props (short for properties) are a Component’s configuration. Props are how components talk to each other. They are received from above component and immutable as far as the Component receiving them is concerned. A Component cannot change its props, but it is responsible for putting together the props of its child Components. Props do not have to just be data — callback functions may be passed in as props.

There is also the case that we can have default props so that props are set even if a parent component doesn’t pass props down.

Image for post



Props and State do similar things but are used in different ways. The majority of our components will probably be stateless. Props are used to pass data from parent to child or by the component itself. They are immutable and thus will not be changed. State is used for mutable data, or data that will change. This is particularly useful for user input.

**Q7. What are controlled components?**

In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. When a user submits a form the values from the aforementioned elements are sent with the form. With React it works differently. The component containing the form will keep track of the value of the input in it's state and will re-render the component each time the callback function e.g. onChange is fired as the state will be updated. A form element whose value is controlled by React in this way is called a "controlled component".

With a controlled component, every state mutation will have an associated handler function. This makes it straightforward to modify or validate user input.

**Q8. What is a higher order component?**

A higher-order component (HOC) is an advanced technique in React for reusing component logic. HOCs are not part of the React API. They are a pattern that emerges from React’s compositional nature.

A higher-order component is a function that takes a component and returns a new component.

HOC’s allow you to reuse code, logic and bootstrap abstraction. HOCs are common in third-party React libraries. The most common is probably Redux’s connect function. Beyond simply sharing utility libraries and simple composition, HOCs are the best way to share behavior between React Components. If you find yourself writing a lot of code in different places that does the same thing, you may be able to refactor that code into a reusable HOC.

**Q9. What is create-react-app?**

create-react-app is the official CLI (Command Line Interface) for React to create React apps with no build configuration.

We don’t need to install or configure tools like Webpack or Babel. They are preconfigured and hidden so that we can focus on the code. We can install easily just like any other node modules. Then it is just one command to start the React project.

Image for post

Image for post

It includes everything we need to build a React app:

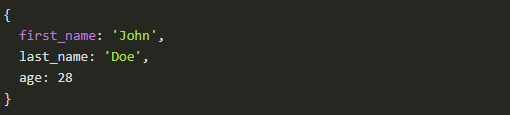
* React, JSX, ES6, and Flow syntax support.
* Language extras beyond ES6 like the object spread operator.
* Autoprefixed CSS, so you don’t need -webkit- or other prefixes.
* A fast interactive unit test runner with built-in support for coverage reporting.
* A live development server that warns about common mistakes.
* A build script to bundle JS, CSS, and images for production, with hashes and sourcemaps.

**Q10. What is Redux?**

The basic idea of Redux is that the entire application state is kept in a single store. The store is simply a javascript object. The only way to change the state is by firing actions from your application and then writing reducers for these actions that modify the state. The entire state transition is kept inside reducers and should not have any side-effects.

Redux is based on the idea that there should be only a single source of truth for your application state, be it UI state like which tab is active or Data state like the user profile details.

Image for post

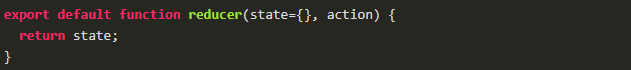


All of these data is retained by redux in a closure that redux calls a store . It also provides us a recipe of creating the said store, namely createStore(x).

The createStore function accepts another function, x as an argument. The passed in function is responsible for returning the state of the application at that point in time, which is then persisted in the store. This passed in function is known as the reducer.

This is a valid example reducer function:

Image for post



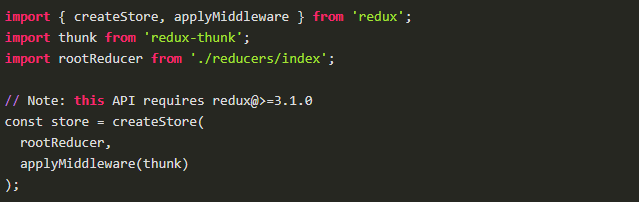
This store can only be updated by dispatching an action. Our App dispatches an action, it is passed into reducer; the reducer returns a fresh instance of the state; the store notifies our App and it can begin it's re render as required.

**Q11. What is Redux Thunk used for?**

Redux thunk is middleware that allows us to write action creators that return a function instead of an action. The thunk can then be used to delay the dispatch of an action if a certain condition is met. This allows us to handle the asyncronous dispatching of actions. The inner function receives the store methods dispatch and getState as parameters.

To enable Redux Thunk, we need to use applyMiddleware() as below

Image for post



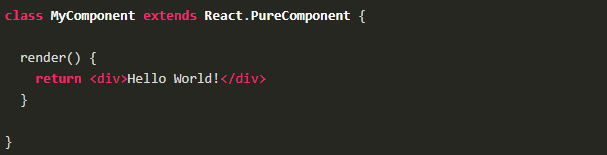
**Q12. What is PureComponent? When to use PureComponent over Component?**

PureComponent is exactly the same as Component except that it handles the shouldComponentUpdate method for us. When props or state changes, PureComponent will do a shallow comparison on both props and state. Component on the other hand won't compare current props and state to next out of the box. Thus, the component will re-render by default whenever shouldComponentUpdate is called.

When comparing previous props and state to next, a shallow comparison will check that primitives have the same value (eg, 1 equals 1 or that true equals true) and that the references are the same between more complex javascript values like objects and arrays.

It is good to prefer PureComponent over Component whenever we never mutate our objects.

Image for post



**Q13. How Virtual-DOM is more efficient than Dirty checking?**

In React, each of our components have a state. This state is like an observable. Essentially, React knows when to re-render the scene because it is able to observe when this data changes. Dirty checking is slower than observables because we must poll the data at a regular interval and check all of the values in the data structure recursively. By comparison, setting a value on the state will signal to a listener that some state has changed, so React can simply listen for change events on the state and queue up re-rendering.

The virtual DOM is used for efficient re-rendering of the DOM. This isn’t really related to dirty checking your data. We could re-render using a virtual DOM with or without dirty checking. In fact, the diff algorithm is a dirty checker itself.

We aim to re-render the virtual tree only when the state changes. So using an observable to check if the state has changed is an efficient way to prevent unnecessary re-renders, which would cause lots of unnecessary tree diffs. If nothing has changed, we do nothing.

**Q14. Is setState() is async? Why is setState() in React Async instead of Sync?**

setState() actions are asynchronous and are batched for performance gains. This is explained in documentation as below.

setState() does not immediately mutate this.state but creates a pending state transition. Accessing this.state after calling this method can potentially return the existing value. There is no guarantee of synchronous operation of calls to setState and calls may be batched for performance gains.

This is because setState alters the state and causes rerendering. This can be an expensive operation and making it synchronous might leave the browser unresponsive. Thus the setState calls are asynchronous as well as batched for better UI experience and performance.

**Q15. What is render() in React? And explain its purpose?**

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>, <div> etc. This function must be kept pure i.e., it must return the same result each time it is invoked.

**Q16. What are controlled and uncontrolled components in React?**

This relates to stateful DOM components (form elements) and the difference:

* A **Controlled Component** is one that takes its current value through props and notifies changes through callbacks like onChange. A parent component “controls” it by handling the callback and managing its own state and passing the new values as props to the controlled component. You could also call this a “dumb component”.
* A Uncontrolled Component is one that stores its own state internally, and you query the DOM using a ref to find its current value when you need it. This is a bit more like traditional HTML.

In most (or all) cases we should use controlled components.

**Q17. Explain the components of Redux.**

Redux is composed of the following components:

* **Action** — Actions are payloads of information that send data from our application to our store. They are the only source of information for the store. We send them to the store using store.dispatch(). Primarly, they are just an object describes what happened in our app.
* **Reducer** — Reducers specify how the application’s state changes in response to actions sent to the store. Remember that actions only describe what happened, but don’t describe how the application’s state changes. So this place determines how state will change to an action.
* **Store** — The Store is the object that brings Action and Reducer together. The store has the following responsibilities: Holds application state; Allows access to state via getState(); Allows state to be updated via dispatch(action); Registers listeners via subscribe(listener); Handles unregistering of listeners via the function returned by subscribe(listener).

It’s important to note that we’ll only have a single store in a Redux application. When we want to split your data handling logic, we’ll use reducer composition instead of many stores.

**Q18. What is React.cloneElement? And the difference with this.props.children?**

React.cloneElement clone and return a new React element using using the passed element as the starting point. The resulting element will have the original element's props with the new props merged in shallowly. New children will replace existing children. key and ref from the original element will be preserved.

React.cloneElement only works if our child is a single React element. For almost everything {this.props.children} is the better solution. Cloning is useful in some more advanced scenarios, where a parent send in an element and the child component needs to change some props on that element or add things like ref for accessing the actual DOM element.

**Q19. What is the second argument that can optionally be passed to setState and what is its purpose?**

A callback function which will be invoked when setState has finished and the component is re-rendered.

Since the setState is asynchronous, which is why it takes in a second callback function. With this function, we can do what we want immediately after state has been updated.

**Q20. What is the difference between React Native and React?**

React is a JavaScript library, supporting both front end web and being run on the server, for building user interfaces and web applications.

On the other hand, React Native is a mobile framework that compiles to native app components, allowing us to build native mobile applications (iOS, Android, and Windows) in JavaScript that allows us to use ReactJS to build our components, and implements ReactJS under the hood.

Edureka

**.  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:

1. It uses the **virtual DOM** instead of the real DOM.
2. It uses **server-side rendering**.
3. 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:

1. It increases the application’s performance
2. It can be conveniently used on the client as well as server side
3. Because of JSX, code’s readability increases
4. React is easy to integrate with other frameworks like Meteor, Angular, etc
5. Using React, writing UI test cases become extremely easy

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

Limitations of React are listed below:

1. React is just a library, not a full-blown framework
2. Its library is very large and takes time to understand
3. It can be little difficult for the novice programmers to understand
4. 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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | render(){      return(    <div>    <h1> Hello World from Edureka!!</h1>             </div>        );  } |

**[React with Redux Certification Training](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)**

* *[Instructor-led Sessions](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)*
* *[Assessments](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)*
* *[Assignments](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)*
* *[Lifetime Access](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)*

[Explore Curriculum](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)

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

A virtual DOM is a lightweight JavaScript object which originally is just the 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.  
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 following aspects:

1. require vs import

|  |  |
| --- | --- |
| 1  2  3  4  5 | // ES5  var React = require('react');    // ES6  import React from 'react'; |

1. export vs exports

|  |  |
| --- | --- |
| 1  2  3  4  5 | // ES5  module.exports = Component;    // ES6  export default Component; |

1. component and function

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | // ES5  var MyComponent = React.createClass({      render: function() {          return    <h3>Hello Edureka!</h3>  ;      }  });    // ES6  class MyComponent extends React.Component {      render() {          return    <h3>Hello Edureka!</h3>  ;      }  } |

1. props

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | // 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>  ;      }  } |

1. state

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

**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. What do you understand from “In React, everything is a component.”**

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. Explain 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>,<div>** 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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | 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().

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | //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:

1. *Initial Rendering Phase:* This is the phase when the component is about to start its life journey and make its way to the DOM.
2. *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.
3. *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:

1. ***componentWillMount()***–Executed just before rendering takes place both on the client as well as server-side.
2. ***componentDidMount()***–Executed on the client side only after the first render.
3. ***componentWillReceiveProps()***– Invoked as soon as the props are received from the parent class and before another render is called.
4. ***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.
5. ***componentWillUpdate()***– Called just before rendering takes place in the DOM.
6. ***componentDidUpdate()***–Called immediately after rendering takes place.
7. ***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:

1. Events are named using camel case instead of just using the lowercase.
2. 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?**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 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.

**Front End Web Development Training**

Next

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | //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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | 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.handleSubmit} />              </label>              <input type="submit" value="Submit" />          </form>        );  } |

**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 |

**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.

**React Redux – React Interview Questions**

**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

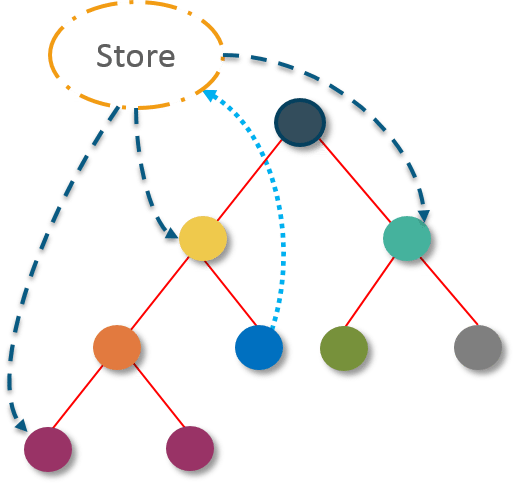
**35. Explain Flux.**

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 hottest 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?**

1. ***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.
2. ***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.
3. ***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:

1. **Action** – It’s an object that describes what happened.
2. **Reducer**–  It is a place to determine how the state will change.
3. **Store** – State/ Object tree of the entire application is saved in the Store.
4. **View** – Simply displays the data provided by the Store.

**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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 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 |

**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.

**React Router – React Interview Questions**

**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 **<div>** 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.

[[](https://www.edureka.co/reactjs-redux-certification-training)](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)

**[React with Redux Certification Training](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)**

[Weekday / Weekend BatchesSee Batch Details](https://www.edureka.co/reactjs-redux-certification-training" \t "_blank)

|  |  |
| --- | --- |
| 1  2  3  4  5 | <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:

1. 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>**).
2. No need to manually set History value: In React Router v4, all we need to do is wrap our routes within the **<BrowserRouter>** component.
3. 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.

**VIMP :Whare are HOC(Higher order functions)**

**VIMP :What is the significance of refs.**

Similarly to keys, refs are added as an attribute to a React.createElement() call, such as <li ref="someName"/>. The ref serves a different purpose, it provides us quick and simple access to the DOM Element represented by a React Element.

Refs can be either a string or a function. Using a string will tell React to automatically store the DOM Element as this.refs[refValue]. For example:

class List extends Component {

constructor(p){

super(p)

}

\_printValue(){

console.log(this.refs.someThing.value)

}

render() {

return <div onClick={e => this.\_printValue()}>

<p>test</p>

<input type="text" ref="someThing" />

<input type="text" ref="someThing" />

<input type="text" ref="someThing" />

<input type="text" ref="someThing" />

<input type="text" ref="someThing" />

</div>

}

}

**VIMP: Whar are the keys in React?**

Keys in React are used to identify unique VDOM Elements with their corresponding data driving the UI; having them helps React optimize rendering by recycling existing DOM elements. Let’s look at an example to portray this.

We have two <TwitterUser> Components being rendered to a page, drawn in decreasing order of followers:

-----------

| A - 103 |

-----------

-----------

| B - 92 |

-----------

Let’s say that B gets updated with 105 Twitter followers, so the app re-renders, and switches the ordering of A and B:

-----------

| B - 105 |

-----------

-----------

| A - 103 |

-----------

Without keys, React would primarily re-render both <TwitterUser> Elements in the DOM. It would re-use DOM elements, but React won’t **re-order** DOM Elements on the screen.

With keys, React would actually re-order the DOM elements, instead of rendering a lot of nested DOM changes. This can serve as a huge performance enhancement, especially if the DOM and VDOM/React Elements being used are costly to render.

Keys themselves should be a unique number or string; so if a React Component is the only child with its key, then React will repurpose the DOM Element represented by that key in future calls to render().

**What is PROP drilling?**

When building a React application, there is often the need for a deeply nested component to use data provided by another component that is much higher in the hierarchy.

Consider the following example components:

* <EditUsersPage />, which includes selectedUserAddress in its component state and renders a <User /> component
* <User />, which renders a <UserDetails /> component
* <UserDetails />, which renders a <UserAddress /> component
* A <UserAddress /> component that requires the selectedUserAddress property stored in the <EditUsersPage /> state

The simplest approach is to simply pass a selectedUserAddress prop from each component to the next in the hierarchy from the source component to the deeply nested component. This is called prop drilling.

The primary disadvantage of prop drilling is that components that should not otherwise be aware of the data—in this case <User /> and <UserDetails />—become unnecessarily complicated and are harder to maintain.

To avoid prop drilling, a common approach is to use React context. This allows a Provider component that supplies data to be defined, and allows nested components to consume context data via either a Consumer component or a useContext hook.

While context can be used directly for sharing global state, it is also possible to use context indirectly via a state management module, such as Redux.

**What is the StrictMode component?**

<StrictMode /> is a component included with React to provide additional visibility of potential issues in components. If the application is running in development mode, any issues are logged to the development console, but these warnings are not shown if the application is running in production mode.

Developers use <StrictMode /> to find problems such as deprecated lifecycle methods and legacy patterns, to ensure that all React components follow current best practices.

<StrictMode /> can be applied at any level of an application component hierarchy, which allows it to be adopted incrementally within a codebase.