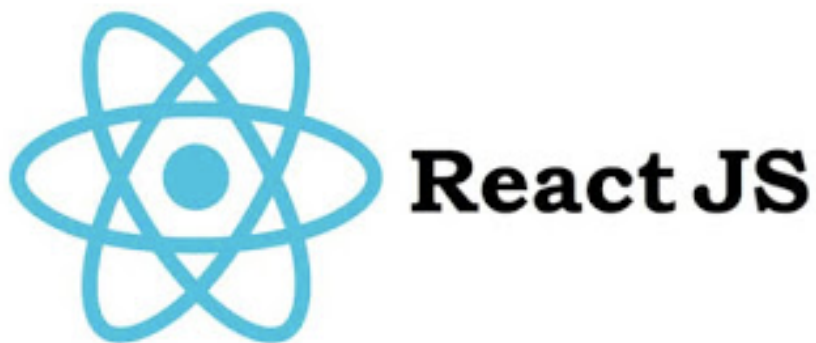


REACT JS WITH REDUX INTERVIEW **QUESTIONS FOR BEGINNERS TO** **EXPERIENCED**

1) What Is ReactJs?

React is an open source JavaScript front end UI library developed by Facebook for creating interactive, stateful & reusable UI components for web and mobile app. It is used by Facebook, Instagram and many more web apps. ReactJS is used for handling view layer for web and mobile applications. One of React's unique major points is that it perform not only on the client side, but also can be rendered on server side, and they can work together inter-operably.



2) Why ReactJs Is Used?

React is used to handle the view part of Mobile application and Web application.

3) How Is React different?

Basically, ReactJs is a limited library that is used for building the interactive UI components. Also used for building complex and stateful web as well as mobile apps. But some of the other JavaScript frameworks not perform the same.

4) When ReactJs Released?

March 2013.

5) What Is Current Stable Version Of ReactJs?

Version: 15.5

Release on: April 7, 2017

6) What is Repository URL of ReactJS?

<https://github.com/facebook/react>

7) Does ReactJs Use Html?

No, It uses JSX which is similar to HTML.

8) What are the features of React?

Major features of React are listed below:

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

9) How to Install and Create React App?

First, install react app creator for creating a react app easily:

```
1. On Windows
2. >> npm install create-react-app -g
3. OR
4. >> npm install -g create-react-app
5.
6. On Mac & Linux
7. >> sudo npm install create-react-app -g
8. OR
9. >> sudo npm install -g create-react-app
```

- Now, create a react app by type following command:

code [source](#)

```
1. >> create-react-app myFirstReactProject
```

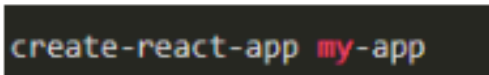
- Now, go to the project folder & start the app:

code [source](#)

```
1. >> npm start
```

10) 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.



```
create-react-app my-app
```

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 source maps.

11) How to specify a port to run a create-react-app based project?

```
If you don't want set environment variable, other option - modify scripts p  
>> "start": "react-scripts start"  
  
Linux and MacOS:  
>> "start": "PORT=3006 react-scripts start"  
OR  
>> "start": "export PORT=3006 react-scripts start"  
  
Windows:  
>> "start": "set PORT=3006 && react-scripts start"
```

12) What the major advantages of using ReactJs?

Following are the list of some major advantages of using ReactJs:

- **Performance** : it is really good performance because React Js uses virtual-DOM.
- **Rendering** : it's render server side and client side both as well.
- **Reusability**: React Js is all about components. So React js provides developers with the opportunity to have more time to use and create common abstractions, setting up the creation, distribution and consumption of isolated reusable parts.
- **JSX**: JSX makes it easy to read the code of your components.
- **Data Binding**: React Js uses one way data binding or uni directional data flow.

- **Testability:** it is easy to test, and integrate some tools like jest.
- **Maintainability:** it ensures readability and makes maintainability easier.
- **Better Combination Technologies:** React makes best use of HTML and JavaScript mixing them ideally and also incorporates CSS to provide your business with the best.
- **Integrate With Others Framework:** it's easy to integrate with other frameworks like Backbone.js, Meteor, Angular, etc.

13) What the major disadvantages of using ReactJs?

Following are the list of some major disadvantages of using ReactJs:

- It covers only 'View' part in MVC (Model-View-Controller).
- React Js is just a JavaScript library, Not a framework.
- It's library is very large and takes time to understand.
- it uses inline templating and JSX.

14) How is ReactJs different from AngularJS?

The first difference between both of them is their code dependency. ReactJS depends less to the code whereas AngularJS needs a lot of coding to be done. The packaging on React is quite strong as compared to the AngularJS. Another difference is React is equipped with Virtual Dom while the Angular has a Regular DOM. ReactJS is all about the components whereas AngularJS focus mainly on the Models, View as well as on Controllers. AngularJS was developed by Google while the ReactJS is the outcome of Facebook.

15) How is React different from 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

16) 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. Even though 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:

```
const element = (  
  <h1 className="greeting">  
    Hello, world!  
  </h1>  
);
```

Equivalent of the above using `React.createElement`

```
const element = React.createElement(  
  'h1',  
  {"className": "greeting"},  
  'Hello, world!'  
);
```

17) What Are The Advantages Of Using JSX?

- JSX is completely optional and its not mandatory, we don't need to use it in order to use React, but it has several advantages and a lot of nice features in JSX.
- JSX is always faster as it performs optimization while compiling code to vanilla JavaScript.
- JSX is also type-safe, means it is strictly typed and most of the errors can be caught during compilation of the JSX code to JavaScript.
- JSX always makes it easier and faster to write templates if we are familiar with HTML syntax.

18) Why can't browsers read JSX?

Browsers can only read JavaScript objects but JSX is 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.

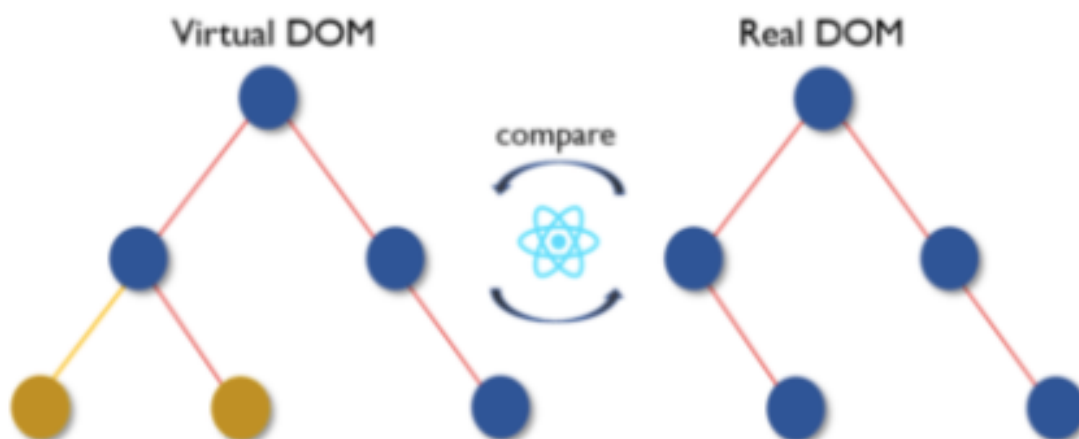
19) What do you understand by Virtual DOM? Explain its working?

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.

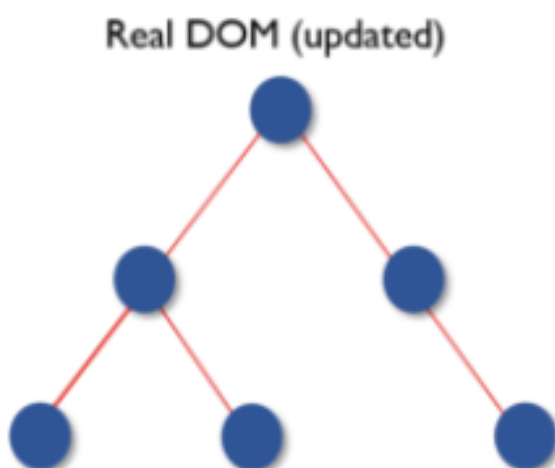
- Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation:



- Then the difference between the previous DOM representation and the new one is calculated:



- Once the calculations are done, the real DOM will be updated with only the things that have actually changed:



20) Differentiate between Real DOM and 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.

21) 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.

- Let's say there is a `<div>` somewhere in your HTML file:

```
<div id="root"></div>
```

- To render a React element into a root DOM node, pass both to `ReactDOM.render()`:

```
const element = <h1>Hello, world</h1>;  
ReactDOM.render(element, document.getElementById('root'));
```

22) What Are Components In ReactJs?

React encourages the idea of reusable components. They are widgets or other parts of a layout (a form, a button, or anything that can be marked up using HTML) that you can reuse multiple times in your web application. ReactJS enables us to create components by invoking the `React.createClass()` method features a `render()` method which is responsible for displaying the HTML code. When designing interfaces, we have to break down the individual design elements (buttons, form fields, layout components, etc.) into reusable components with well-defined interfaces. That way, the next time we need to build some UI, we can write much less code. This means faster development time, fewer bugs, and fewer bytes down the wire. There are mainly two types of Components:

- **Functional Component OR Stateless Component** - Only props, no state.

```
function Clock(props) {  
  return (  
    <div>  
      <h1>Hello, world!</h1>  
      <h2>It is {props.date.toLocaleTimeString()}.</h2>  
    </div>  
  );  
}
```

```
function tick() {  
  ReactDOM.render(  
    <Clock date={new Date()} />,  
    document.getElementById('root')  
  );  
}  
  
setInterval(tick, 1000);
```

- **Class Component OR Stateful Component** - Both props and state.

```
class Clock extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {date: new Date()};  
  }
```

```
render() {  
  return (  
    <div>  
      <h1>Hello, world!</h1>  
      <h2>It is {this.state.date.toLocaleTimeString()}.</h2>  
    </div>  
  );  
}  
}  
  
ReactDOM.render(  
  <Clock />,  
  document.getElementById('root')  
);
```

23) What is constructor in ReactJs?

The constructor for a React component is called before it is mounted. When implementing the constructor for a `React.Component` subclass, you should call `super(props)` before any other statement. Otherwise, `this.props` will be undefined in the constructor, which can lead to bugs.

```
constructor(props) {  
  super(props);  
  this.state = {  
    posts: [],  
    comments: []  
  };  
}
```

24) How To Embed Two Components In One Component?

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <Header/>
        <Content/>
      </div>
    );
  }
}

class Header extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
      </div>
    );
  }
}
```

```
class Content extends React.Component {
  render() {
    return (
      <div>
        <h2>Content</h2>
        <p>The content text!!!</p>
      </div>
    );
  }
}

export default App;
```

25) 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:

```
import React from 'react';

const Contacts = React.createClass({
  render() {
    return (
      <div></div>
    );
  }
});

export default Contacts;
```

- 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`:

```
import React from 'react';

class Contacts extends React.Component({
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div></div>
    );
  }
})

export default Contacts;
```

26) What is the difference between createElement and cloneElement?

- createElement is the thing that JSX gets transpiled to and is the thing that React uses to make React Elements (protest representations of some UI).
- CloneElement is utilized as a part of request to clone a component and pass it new props. They nailed the naming on these two.

27) 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. Example:

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.headerProp}</h1>
        <h2>{this.props.contentProp}</h2>
      </div>
    );
  }
}
export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App headerProp = "Header from props..." contentProp = "Content from props..." />, document.getElementById('app'));

export default App;
```

28) What is Default Props?

You can also set default property values directly on the component constructor instead of adding it to the **ReactDOM.render()** element. Example:

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.headerProp}</h1>
        <h2>{this.props.contentProp}</h2>
      </div>
    );
  }
}

App.defaultProps = {
  headerProp: "Header from props...",
  contentProp: "Content from props..."
}

export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App/>, document.getElementById('app'));
```

29) 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 behaviour. They are mutable unlike the props and create dynamic and interactive components. They are accessed via `this.state()`. Example:


```
import React from 'react';
class Button extends React.Component {
  constructor() {
    super();
    this.state = {
      count: 0,
    };
  }

  updateCount() {
    this.setState((prevState, props) => {
      return { count: prevState.count + 1 }
    });
  }

  render() {
    return (<button
      onClick={() => this.updateCount()}
      >
      Clicked {this.state.count} times
    </button>);
  }
}

export default Button;
```

30) 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')
);

```

31) How to Use State Correctly?

There are three things you should know about setState().

- **Do Not Modify State Directly** : use setState().

```
// Wrong
this.state.comment = 'Hello';
```

```
// Correct
this.setState({comment: 'Hello'});
```

- **State Updates May Be Asynchronous :** React may batch multiple `setState()` calls into a single update for performance. Because `this.props` and `this.state` may be updated asynchronously, you should not rely on their values for calculating the next state.

```
// Wrong
this.setState({
  counter: this.state.counter + this.props.increment,
});
```

```
// Correct
this.setState((state, props) => ({
  counter: state.counter + props.increment
}));
```

- **State Updates are Merged :** When you call `setState()`, React merges the object you provide into the current state. Your state may contain several independent variables but you can update them independently with separate `setState()` calls.

```

constructor(props) {
  super(props);
  this.state = {
    posts: [],
    comments: []
  };
}

```

```

componentDidMount() {
  fetchPosts().then(response => {
    this.setState({
      posts: response.posts
    });
  });

  fetchComments().then(response => {
    this.setState({
      comments: response.comments
    });
  });
}

```

32) Differentiate between stateful and stateless components?

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.

33) Difference between constructor and getInitialState ?

The difference between constructor and getInitialState is the difference between ES6 and ES5 itself. getInitialState is used with React.createClass and constructor is used with React.Component:

ES5

```
code source
1. var MyComponent = React.createClass({
2.   getInitialState() {
3.     return { /* initial state */ };
4.   },
5. });
```

ES6

```
code source
1. class MyComponent extends React.Component {
2.   constructor(props) {
3.     super(props);
4.     this.state = { /* initial state */ };
5.   }
6. }
```

34) 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.handleChange(e) } />
  );
}
```

35) Differentiate between states and 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

36) How the parent and child components exchange information?

This task is generally performed with the help of functions. Actually, there are several functions which are provided to both parent and child components. They simply make use of them through props. Their communication should be accurate and reliable. The need of same can be there anytime and therefore functions are considered for this task. They always make sure that information can be exchanged easily and in an efficient manner among the parent and child components.

37) How do you tell React to build in Production mode and what will that do?

Ordinarily you'd utilize Webpack's **DefinePlugin** strategy to set **NODE_ENV** to production. This will strip out things like propTypes approval and additional notices. Over that, it's likewise a smart thought to minify your code in light of the fact that React utilizes Uglify's dead-code elimination to strip out advancement just code and remarks, which will radically

diminish the measure of your package.

38) What do you understand with the term polling?

The server needs to be monitored to for updates with respect to time. The primary aim in most of the cases is to check whether novel comments are there or not. This process is basically considered as pooling. It checks for the updates approximately after every 5 seconds. It is possible to change this time period easily. Pooling help keeping an eye on the users and always make sure that no negative information is present on the servers. Actually, it can create issues related to several things and thus pooling is considered.

39) When would you use a Class Component over a Functional Component?

If your component has state or a lifecycle method(s), use a Class component. or else, use a Functional component.

40) How To Apply Validation On Props In ReactJs?

Properties validation is a useful way to force the correct usage of the components. This will help during development to avoid future bugs and problems, once the app becomes larger. It also makes the code more readable, since we can see how each component should be used.

- Example : we are creating **App** component with all the **props** that we need. **App.propTypes** is used for props validation. If some of the props aren't using the correct type that we assigned, we will get a console warning. After we specify validation patterns, we will set **App.defaultProps**.

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <h3>Array: {this.props.propArray}</h3>
        <h3>Bool: {this.props.propBool ? "True..." : "False..."}</h3>
        <h3>Func: {this.props.propFunc(3)}</h3>
        <h3>Number: {this.props.propNumber}</h3>
        <h3>String: {this.props.propString}</h3>
        <h3>Object: {this.props.propObject.objectName1}</h3>
        <h3>Object: {this.props.propObject.objectName2}</h3>
        <h3>Object: {this.props.propObject.objectName3}</h3>
      </div>
    );
  }
}
```

```
App.propTypes = {
  propArray: React.PropTypes.array.isRequired,
  propBool: React.PropTypes.bool.isRequired,
  propFunc: React.PropTypes.func,
  propNumber: React.PropTypes.number,
  propString: React.PropTypes.string,
  propObject: React.PropTypes.object
}
```

```
App.defaultProps = {
  propArray: [1,2,3,4,5],
  propBool: true,
  propFunc: function(e){return e},
  propNumber: 1,
  propString: "String value...",

  propObject: {
    objectName1: "objectValue1",
    objectName2: "objectValue2",
    objectName3: "objectValue3"
  }
}

export default App;
```



```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App/>, document.getElementById('app'));
```

41) Explain conditional rendering in React?

Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like if or the conditional operator to create elements representing the current state, and let React update the UI to match them.

- Consider these two components:

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

- We'll create a Greeting component that displays either of these components depending on whether a user is logged in:

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}
```

```
ReactDOM.render(  
  // Try changing to isLoggedIn={true}:  
  <Greeting isLoggedIn={false} />,  
  document.getElementById('root')  
);
```

42) Explain conditional rendering with element variables in React?

You can use variables to store elements. This can help you conditionally render a part of the component while the rest of the output doesn't change.

- Consider these two new components representing Logout and Login buttons:

```
function LoginButton(props) {  
  return (  
    <button onClick={props.onClick}>  
      Login  
    </button>  
  );  
}
```

```
function LogoutButton(props) {  
  return (  
    <button onClick={props.onClick}>  
      Logout  
    </button>  
  );  
}
```

- We will create a stateful component called LoginControl. It will render either <LoginButton /> or <LogoutButton /> depending on its current state. It will also render a <Greeting /> from the previous example:

```
class LoginControl extends React.Component {  
  constructor(props) {  
    super(props);  
    this.handleLoginClick = this.handleLoginClick.bind(this);  
    this.handleLogoutClick = this.handleLogoutClick.bind(this);  
    this.state = {isLoggedIn: false};  
  }  
  
  handleLoginClick() {  
    this.setState({isLoggedIn: true});  
  }  
}
```

```
  handleLogoutClick() {  
    this.setState({isLoggedIn: false});  
  }  
  
  render() {  
    const isLoggedIn = this.state.isLoggedIn;  
    let button;  
  
    if (isLoggedIn) {  
      button = <LogoutButton onClick={this.handleLogoutClick} />;  
    } else {  
      button = <LoginButton onClick={this.handleLoginClick} />;  
    }  
  }  
}
```

```
    return (  
      <div>  
        <Greeting isLoggedIn={isLoggedIn} />  
        {button}  
      </div>  
    );  
  }  
}  
  
ReactDOM.render(  
  <LoginControl />,  
  document.getElementById('root')  
);
```

43) Explain conditional rendering through inline if-else with conditional operator in React?

While declaring a variable and using an if statement is a fine way to conditionally render a component, sometimes you might want to use a shorter syntax. There are a way to inline conditions in JSX. Conditionally rendering elements inline is to use the JavaScript conditional operator condition ? true : false.

- In the example below, we use it to conditionally render a small block of text:

```
render() {  
  const isLoggedIn = this.state.isLoggedIn;  
  return (  
    <div>  
      The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.  
    </div>  
  );  
}
```

- It can also be used for larger expressions although it is less obvious what's going on:

```
render() {  
  const isLoggedIn = this.state.isLoggedIn;  
  return (  
    <div>  
      {isLoggedIn ? (  
        <LogoutButton onClick={this.handleLogoutClick} />  
      ) : (  
        <LoginButton onClick={this.handleLoginClick} />  
      )}  
    </div>  
  );  
}
```

44) What are the different phases of React component's lifecycle?

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

- **Initial Rendering Phase** : This is the phase when the component is about to start its life journey and make its way to the DOM.
- **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.
- **Unmounting Phase** : This is the final phase of a component's life cycle in which the component is destroyed and removed from the DOM.

45) Explain the lifecycle methods of React components in detail?

Some of the most important lifecycle methods are:

- **componentWillMount()** – Executed just before rendering takes place both on the client as well as server-side.
- **componentDidMount()** – Executed on the client side only after the first render.

- **componentWillReceiveProps()** – Invoked as soon as the props are received from the parent class and before another render is called.
- **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.
- **componentWillUpdate()** – Called just before rendering takes place in the DOM.
- **componentDidUpdate()** – Called immediately after rendering takes place.
- **componentWillUnmount()** – Called after the component is unmounted from the DOM. It is used to clear up the memory spaces.

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }
}
```

```
tick() {  
  this.setState({  
    date: new Date()  
  });  
}
```

```
render() {  
  return (  
    <div>  
      <h1>Hello, world!</h1>  
      <h2>It is {this.state.date.toLocaleTimeString()}.</h2>  
    </div>  
  );  
}  
  
ReactDOM.render(  
  <Clock />,  
  document.getElementById('root')  
);
```

46) 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:

- Events are named using camel case instead of just using the lowercase.
- 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.

47) How do you create an event in React?

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }
}
```

```
render() {
  return (
    <button onClick={this.handleClick}>
      {this.state.isToggleOn ? 'ON' : 'OFF'}
    </button>
  );
}

ReactDOM.render(
  <Toggle />,
  document.getElementById('root')
);
```

48) What Is Synthetic event 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.

49) What Is Child event in React?

When we need to update the **state** of the parent component from its child, we can create an

event handler (**updateState**) in the parent component and pass it as a prop (**updateStateProp**) to the child component where we can just call it. Example:

```
class App extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      data: 'Initial data...'
    }
    this.updateState = this.updateState.bind(this);
  };
  updateState() {
    this.setState({data: 'Data updated from the child component...'})
  }
}
```

```
  render() {
    return (
      <div>
        <Content myDataProp = {this.state.data}
          updateStateProp = {this.updateState}></Content>
      </div>
    );
  }
}
```

```
class Content extends React.Component {
  render() {
    return (
      <div>
        <button onClick = {this.props.updateStateProp}>CLICK</button>
        <h3>{this.props.myDataProp}</h3>
      </div>
    );
  }
}
export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App/>, document.getElementById('app'));
```

50) 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.

- **Example-1:**

```
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>
```

```
);
}
}
```

- **Example-2:**

```
class App extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      data: ''
    }
    this.updateState = this.updateState.bind(this);
    this.clearInput = this.clearInput.bind(this);
  };
  updateState(e) {
    this.setState({data: e.target.value});
  }
}
```

```
clearInput() {
  this.setState({data: ''});
  ReactDOM.findDOMNode(this.refs.myInput).focus();
}
render() {
  return (
    <div>
      <input value = {this.state.data} onChange = {this.updateState}
        ref = "myInput"></input>
      <button onClick = {this.clearInput}>CLEAR</button>
      <h4>{this.state.data}</h4>
    </div>
  );
}
}
export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App/>, document.getElementById('app'));
```

51) 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.

52) 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. Example:

```
//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>

    );
}
}

```

53) 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. Example:

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }
}
```

```
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.handleChange} />  
      </label>  
      <input type="submit" value="Submit" />  
    </form>  
  );  
}
```

54) How are handling multiple inputs with forms in React?

When you need to handle multiple controlled input elements, you can add a name attribute to each element and let the handler function choose what to do based on the value of event.target.name. Example:

```
class Reservation extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      isGoing: true,  
      numberOfGuests: 2  
    };  
  
    this.handleInputChange = this.handleInputChange.bind(this);  
  }  
}
```

```
handleInputChange(event) {  
  const target = event.target;  
  const value = target.type === 'checkbox' ? target.checked : target.value;  
  const name = target.name;  
  
  this.setState({  
    [name]: value  
  });  
}
```

```
render() {  
  return (  
    <form>  
      <label>  
        Is going:  
        <input  
          name="isGoing"  
          type="checkbox"  
          checked={this.state.isGoing}  
          onChange={this.handleInputChange} />  
        </label>  
      </form>  
    );  
}
```

```
      <br />  
      <label>  
        Number of guests:  
        <input  
          name="numberOfGuests"  
          type="number"  
          value={this.state.numberOfGuests}  
          onChange={this.handleInputChange} />  
      </label>  
    </form>  
  );  
}
```

55) Explain using forms from child component in React?

In the following example, we will see how to use forms from child component. **onChange** method will trigger state update that will be passed to the child

input **value** and rendered on the screen. A similar example is used in the Events chapter. Whenever we need to update state from child component, we need to pass the function that will handle updating (**updateState**) as a prop (**updateStateProp**). Example:

```
class App extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      data: 'Initial data...'
    }
    this.updateState = this.updateState.bind(this);
  };
  updateState(e) {
    this.setState({data: e.target.value});
  }
}
```

```
render() {
  return (
    <div>
      <Content myDataProp = {this.state.data}
        updateStateProp = {this.updateState}></Content>
    </div>
  );
}
```

```
class Content extends React.Component {
  render() {
    return (
      <div>
        <input type = "text" value = {this.props.myDataProp}
          onChange = {this.props.updateStateProp} />
        <h3>{this.props.myDataProp}</h3>
      </div>
    );
  }
}
export default App;
```



```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App/>, document.getElementById('app'));
```

56) What do you know about controlled and 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

57) What are Higher Order Components(HOC) in React?

- 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 behaviour from their input components. You can say that HOC are 'pure' components:

```
const NewComponent = (BaseComponent) => {
  // ... create new component from old one and update
  return UpdatedComponent
}
```

- A higher-order component is a function that takes a component as an argument and returns a component. This means that a HOC will always have a form similar to the follow:

```
1  import React from 'react';
2
3  const higherOrderComponent = (WrappedComponent) => {
4    class HOC extends React.Component {
5      render() {
6        return <WrappedComponent />;
7      }
8    }
9
10   return HOC;
11  };
```

- The higherOrderComponent is a function that takes a component called WrappedComponent as an argument. We create a new component called HOC which returns the <WrappedComponent/> from its render function. While this actually adds no functionality in the trivial example, it depicts the common pattern that every HOC function will follow. We can invoke the HOC as follows:

```
const SimpleHOC = higherOrderComponent(MyComponent);
```

58) Explain basic Higher Order Component(HOC)?

- Now we will extend our basic higher-order component pattern to inject data into our wrapped. Our team has actually figured out the secretToLife which turns out to be the number 42. Some of our components need to share this information, and we can create a HOC called withSecretToLife to pass it as a prop to our components:

```
import React from 'react';

const withSecretToLife = (WrappedComponent) => {
  class HOC extends React.Component {
    render() {
      return (
        <WrappedComponent
          {...this.props}
          secretToLife={42}
        />
      );
    }
  }

  return HOC;
};

export default withSecretToLife;
```

- Notice that this HOC is almost identical to our basic pattern. All we have done is add a prop `secretToLife={42}`, which allows the wrapped component to access the value by calling `this.props.secretToLife`. The other addition is that we spread the props passed to the component. This ensures that any other props that are passed to the wrapped component will be accessible via `this.props` in the same manner they would be called if the component was not passed through our higher-order component function:

```
import React from 'react';
import withSecretToLife from 'components/withSecretToLife';

const DisplayTheSecret = props => (
  <div>
    The secret to life is {props.secretToLife}.
  </div>
);

const WrappedComponent = withSecretToLife(DisplayTheSecret);

export default WrappedComponent;
```

- Our WrappedComponent, which is just an enhanced version of <DisplayTheSecret/>, will allow us to access secretToLife as a prop.

59) Explain practical Higher Order Component(HOC)?

- Now that we have a solid grasp on the fundamental pattern for HOC, we can build one that is practical for a real application. A higher-order component has access to all the default React API, including state and the lifecycle methods. The functionality of our withStorage HOC will be to save/load the state of a component, allowing us to quickly access and render it on a page load:

```
const withStorage = (WrappedComponent) => {
  class HOC extends React.Component {
    state = {
      localStorageAvailable: false,
    };

    componentDidMount() {
      this.checkLocalStorageExists();
    }

    checkLocalStorageExists() {
      const testKey = 'test';
```

```
    try {
      localStorage.setItem(testKey, testKey);
      localStorage.removeItem(testKey);
      this.setState({ localStorageAvailable: true });
    } catch(e) {
      this.setState({ localStorageAvailable: false });
    }
  }

  load = (key) => {
    if (this.state.localStorageAvailable) {
      return localStorage.getItem(key);
    }

    return null;
  }

  save = (key, data) => {
    if (this.state.localStorageAvailable) {
      localStorage.setItem(key, data);
    }
  }

  remove = (key) => {
    if (this.state.localStorageAvailable) {
      localStorage.removeItem(key);
    }
  }
}
```

```

    render() {
      return (
        <WrappedComponent
          load={this.load}
          save={this.save}
          remove={this.remove}
          {...this.props}
        />
      );
    }
  }

  return HOC;
}

export default withStorage;

```

- At the top of withStorage we have a single item in the component's state which tracks if localStorage is available in the given browser. We use the componentDidMount lifecycle hook which will check if localStorage exists in the checkLocalStorageExists function. Here it will test saving an item and set the state to true if it succeeds.
- We also add three functions to our HOC — load, save, and remove. These are used to directly access the localStorage API if it is available. Our three functions on the HOC are passed to our wrapped component to be consumed there.
- Now we will create a new component to be wrapped in our withStorageHOC. It will be used to display a user's username and favorite movie. However, the API call to get this information takes a very long time. We can also assume that these values will never change once set.
- To ensure we have a great user experience, we will make this API call only if the values haven't been saved. Then every time the user returns to the page, they can access the data immediately instead of waiting for our API to return:

```
import React from 'react';
import withStorage from 'components/withStorage';

class ComponentNeedingStorage extends React.Component {
  state = {
    username: '',
    favoriteMovie: '',
  }

  componentDidMount() {
    const username = this.props.load('username');
    const favoriteMovie = this.props.load('favoriteMovie');

    if (!username || !favoriteMovie) {
      // This will come from the parent component
      // and would be passed when we spread props {...this.props}
      this.props.reallyLongApiCall()
        .then((user) => {
          this.props.save('username', user.username) || '';
          this.props.save('favoriteMovie', user.favoriteMovie) || '';
          this.setState({
            username: user.username,
            favoriteMovie: user.favoriteMovie,
          });
        });
    } else {
      this.setState({ username, favoriteMovie })
    }
  }
}
```

```
render() {  
  const { username, favoriteMovie } = this.state;  
  
  if (!username || !favoriteMovie) {  
    return <div>Loading...</div>;  
  }  
  
  return (  
    <div>  
      My username is {username}, and I love to watch {favoriteMovie}.  
    </div>  
  )  
}  
  
const WrappedComponent = withStorage(ComponentNeedingStorage);  
  
export default WrappedComponent;
```

- Inside the componentDidMount of our wrapped component, we first try to access the username and favorite movie from localStorage. If the values do not exist, we make our expensive API call named this.props.reallyLongApiCall. Once this function returns, we save the username and favorite to localStorage and update the component's state to display them on the screen.

6o) What can you do with HOC?

Higher Order Component (HOC) can be used for -

- Code reusability.
- State abstraction and manipulation.
- Bootstrap abstraction.

- Props manipulation.
- Render High jacking.

Example:

```
var newData = {
  data: 'Data from HOC...',
}

var MyHOC = ComposedComponent => class extends React.Component {
  componentDidMount() {
    this.setState({
      data: newData.data
    });
  }
  render() {
    return <ComposedComponent {...this.props} {...this.state} />;
  }
}
```

```
};

class MyComponent extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.data}</h1>
      </div>
    )
  }
}

export default MyHOC(MyComponent);
```

61) 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.

62) What Are the methods called when Component is created or while inserting it to DOM?

- constructor() method.
- componentWillMount() method.
- render() method.
- componentDidMount() method.

63) What Are the methods called when State or Props of the Component is changed?

- componentWillReceiveProps().
- shouldComponentUpdate().
- componentWillUpdate().
- render().
- componentDidUpdate().

64) 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. Example:

```
class App extends React.Component {
  constructor() {
    super();

    this.state = {
      data:[
        {
          component: 'First...',
          id: 1
        },
        {
          component: 'Second...',
          id: 2
        },
        {
          component: 'Third...',
          id: 3
        }
      ]
    }
  }
}
```

```
render() {
  return (
    <div>
      <div>
        {this.state.data.map((dynamicComponent, i) => <Content
          key = {i} componentData = {dynamicComponent}/>)}
      </div>
    </div>
  );
}
```

```
class Content extends React.Component {
  render() {
    return (
      <div>
        <div>{this.props.componentData.component}</div>
        <div>{this.props.componentData.id}</div>
      </div>
    );
  }
}
export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App/>, document.getElementById('app'));
```

65) Explain extracting component with keys in React?

Keys only make sense in the context of the surrounding array. For example, if you extract a ListItem component, you should keep the key on the <ListItem /> elements in the array rather than on the element in the ListItem itself:

```
function ListItem(props) {
  // Correct! There is no need to specify the key here:
  return <li>{props.value}</li>;
}

function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    // Correct! Key should be specified inside the array.
    <ListItem key={number.toString()}
              value={number} />
  );
}
```

```
);  
return (  
  <ul>  
    {listItems}  
  </ul>  
);  
}  
  
const numbers = [1, 2, 3, 4, 5];  
ReactDOM.render(  
  <NumberList numbers={numbers} />,  
  document.getElementById('root')  
);
```

66) How can I make an AJAX call?

You can use any AJAX library you like with React. Some popular ones are Axios, jQuery AJAX, and the browser built-in `window.fetch`. Example:

```
fetch(url, options).then(function(response) {  
  // handle HTTP response  
}, function(error) {  
  // handle network error  
})
```

More comprehensive usage example:

```
fetch(url, {  
  method: "POST",  
  body: JSON.stringify(data),  
  headers: {  
    "Content-Type": "application/json"  
  },  
  credentials: "same-origin"  
}).then(function(response) {  
  response.status      //=> number 100–599  
  response.statusText  //=> String  
  response.headers     //=> Headers  
  response.url         //=> String  
  
  return response.text()  
}, function(error) {  
  error.message //=> String  
})
```

67) Where in the component lifecycle should I make an AJAX call?

You should populate data with AJAX calls in the `componentDidMount` lifecycle method. This is so you can use `setState` to update your component when the data is retrieved. The component below demonstrates how to make an AJAX call in `componentDidMount` to populate local component state.

- The example API returns a JSON object like this:

```
{
  "items": [
    { "id": 1, "name": "Apples", "price": "$2" },
    { "id": 2, "name": "Peaches", "price": "$5" }
  ]
}
```

- To fetch a JSON object like this:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      error: null,
      isLoading: false,
      items: []
    };
  }
}
```

```
componentDidMount() {
  fetch("https://api.example.com/items")
    .then(res => res.json())
    .then(
      (result) => {
        this.setState({
          isLoading: true,
          items: result.items
        });
      },
    ),
}
```

```
// Note: it's important to handle errors here
// instead of a catch() block so that we don't swallow
// exceptions from actual bugs in components.
(error) => {
  this.setState({
    isLoading: true,
    error
  });
}
)
}
```

```
render() {
  const { error, isLoading, items } = this.state;
  if (error) {
    return <div>Error: {error.message}</div>;
  } else if (!isLoading) {
    return <div>Loading...</div>;
  } else {
    return (
```

```
      <ul>
        {items.map(item => (
          <li key={item.name}>
            {item.name} {item.price}
          </li>
        ))}
      </ul>
    );
  }
}
```

68) Performing a get and post request through axios in ReactJs?

- Performing a GET request.


```
const axios = require('axios');

// Make a request for a user with a given ID
axios.get('/user?ID=12345')
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  })
  .then(function () {
    // always executed
  });
```

```
// Want to use async/await? Add the `async` keyword to your outer function/method.
async function getUser() {
  try {
    const response = await axios.get('/user?ID=12345');
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

- Performing a POST request.

```
axios.post('/user', {
  firstName: 'Fred',
  lastName: 'Flintstone'
})
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

Performing multiple concurrent requests

```
function getUserAccount() {  
  return axios.get('/user/12345');  
}  
  
function getUserPermissions() {  
  return axios.get('/user/12345/permissions');  
}  
  
axios.all([getUserAccount(), getUserPermissions()])  
  .then(axios.spread(function (acct, perms) {  
    // Both requests are now complete  
  }));
```

69) 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. Example is to set up routing for an app:

- **Install a React Router :** A simple way to install the **react-router** is to run the following code snippet in the **command prompt** window:

```
C:\Users\username\Desktop\reactApp>npm install react-router
```

- **Create Components :** In this step, we will create four components. The **App** component will be used as a tab menu. The other three components (**Home**), (**About**) and (**Contact**) are rendered once the route has changed:

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Router, Route, Link, browserHistory, IndexRoute } from 'react-router'

class App extends React.Component {
  render() {
    return (
      <div>
        <ul>
          <li>Home</li>
          <li>About</li>
          <li>Contact</li>
        </ul>
        {this.props.children}
      </div>
    )
  }
}
```

```
    </div>
  )
}
}
export default App;

class Home extends React.Component {
  render() {
    return (
      <div>
        <h1>Home...</h1>
      </div>
    )
  }
}
export default Home;
```

```
class About extends React.Component {
  render() {
    return (
      <div>
        <h1>About...</h1>
      </div>
    )
  }
}
export default About;
```

```
class Contact extends React.Component {
  render() {
    return (
      <div>
        <h1>Contact...</h1>
      </div>
    )
  }
}
export default Contact;
```

- **Add a Router :** Now, we will add routes to the app. Instead of rendering **App** element like in the previous example, this time the **Router** will be rendered. We will also set components for each route:

```
ReactDOM.render((
  <Router history = {browserHistory}>
    <Route path = "/" component = {App}>
      <IndexRoute component = {Home} />
      <Route path = "home" component = {Home} />
      <Route path = "about" component = {About} />
      <Route path = "contact" component = {Contact} />
    </Route>
  </Router>
), document.getElementById('app'))
```

70) 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.

71) 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>
```

72) List down the advantages of React Router?

Few advantages are:

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

73) How is React Router different from conventional 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

74) 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.

75) 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.

76) What are the three principles that Redux follows?

- **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.
- **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.
- **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.

77) 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.

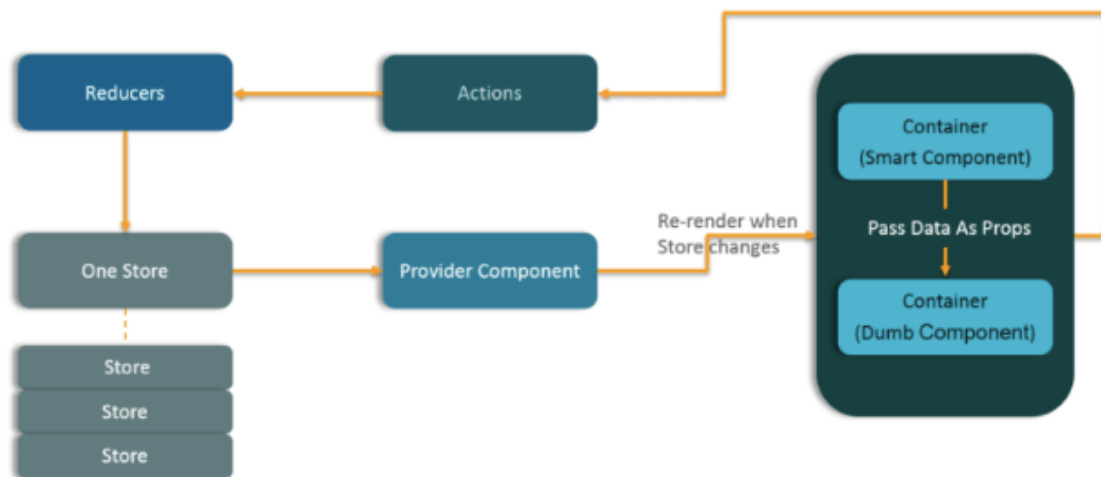


78) List down the components of Redux?

Redux is composed of the following components:

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

79) Show how the data flows through Redux?



80) 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:

```
/*
 * action types
 */

export const ADD_TODO = 'ADD_TODO'
export const TOGGLE_TODO = 'TOGGLE_TODO'
export const SET_VISIBILITY_FILTER = 'SET_VISIBILITY_FILTER'
```

```
/*
 * other constants
 */

export const VisibilityFilters = {
  SHOW_ALL: 'SHOW_ALL',
  SHOW_COMPLETED: 'SHOW_COMPLETED',
  SHOW_ACTIVE: 'SHOW_ACTIVE'
}
```



```
/*
 * action creators
 */

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

export function toggleTodo(index) {
  return { type: TOGGLE_TODO, index }
}

export function setVisibilityFilter(filter) {
  return { type: SET_VISIBILITY_FILTER, filter }
}
```

81) 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:

```
import { combineReducers } from 'redux'
import {
  ADD_TODO,
  TOGGLE_TODO,
  SET_VISIBILITY_FILTER,
  VisibilityFilters
} from './actions'
const { SHOW_ALL } = VisibilityFilters
```

```
function visibilityFilter(state = SHOW_ALL, action) {  
  switch (action.type) {  
    case SET_VISIBILITY_FILTER:  
      return action.filter  
    default:  
      return state  
  }  
}
```

```
function todos(state = [], action) {  
  switch (action.type) {  
    case ADD_TODO:  
      return [  
        ...state,  
        {  
          text: action.text,  
          completed: false  
        }  
      ]  
    }  
}
```

```
    case TOGGLE_TODO:  
      return state.map((todo, index) => {  
        if (index === action.index) {  
          return Object.assign({}, todo, {  
            completed: !todo.completed  
          })  
        }  
        return todo  
      })  
    default:  
      return state  
  }  
}
```

```
const todoApp = combineReducers({
  visibilityFilter,
  todos
})

export default todoApp
```

82) 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:

```
import { createStore } from 'redux'
import todoApp from './reducers'

const store = createStore(todoApp)
```

83) How is Redux different from Flux?

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

84) 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.

85) Functional programming concepts?

The various functional programming concepts used to structure Redux are listed below:

- Functions are treated as First class objects.
- Capable to pass functions in the format of arguments.

- Capable to control flow using, recursions, functions and arrays.
- helper functions such as reduce and map filter are used.
- allows linking functions together.
- The state doesn't change.
- Prioritise the order of executing the code is not really necessary.

86) Redux change of state?

For a release of an action, a change in state to an application is applied, this ensures an intent to change the state will be achieved. Example:

- The user clicks a button in the application.
- A function is called in the form of component
- So now an action gets dispatched by the relative container.
- This happens because the prop (which was just called in the container) is tied to an action dispatcher using mapDispatchToProps (in the container).
- Reducer on capturing the action it intern executes a function and this function returns a new state with specific changes.
- The state change is known by the container and modifies a specific prop in the component as a result of the mapStateToProps function.

87) What is the typical flow of data in a React + Redux app?

Call-back from UI component dispatches an action with a payload, these dispatched actions are intercepted and received by the reducers. this interception will generate a new application state. from here the actions will be propagated down through a hierarchy of

components from Redux store. The below diagram depicts the entity structure of a redux+react setup.

88) How different is React's ES6 syntax when compared to ES5?

Syntax has changed from ES5 to ES6 in following aspects:

i. require vs import

```

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

```

ii. export vs exports

```

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

```

iii. component and function

```

1 // ES5
2 var MyComponent = React.createClass({
3   render: function() {
4     return
5
6     <h3>Hello Edureka!</h3>
7
8
9
10
11   };
12   }
13 });
14
15 // ES6
16 class MyComponent extends React.Component {
17   render() {
18     return
19
20     <h3>Hello Edureka!</h3>
21
22
23
24
25   };
26

```

```

26 |   }
27 | }

```

iv. props

```

1  // ES5
2  var App = React.createClass({
3    propTypes: { name: React.PropTypes.string },
4    render: function() {
5      return
6
7
8    <h3>Hello, {this.props.name}!</h3>
9
10
11
12    ;
13    }
14  });
15
16  // ES6
17  class App extends React.Component {
18    render() {
19      return
20
21
22    <h3>Hello, {this.props.name}!</h3>
23
24
25
26    ;
27    }
28  }

```

v. state

```

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

```

```
21         super();
22         this.state = { name: 'world' };
23     }
24     render() {
25         return
26
27
28         <h3>Hello, {this.state.name}!</h3>
29
30
31
32     ;
33     }
34 }
```