REACT JS

Interview Questions

Contents

1. 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.

2. What are the features of React?

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

3. List some of the major advantages of React.

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

4. What are the limitations of React?

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

5. What is JSX?

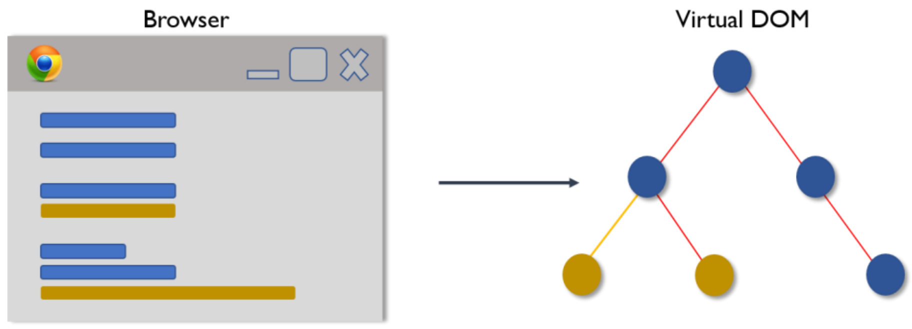
JSX is a shorthand for JavaScript XML. This is a type of file used by React which utilizes the expressiveness of JavaScript along with HTML like template syntax. This makes the HTML file really easy to understand. This file makes applications robust and boosts its performance. Below is an example of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

```
render(){

return(

<div>

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

</div>

);
```

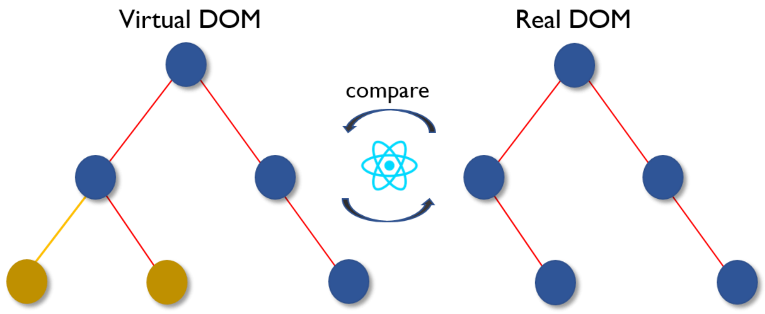
6. What do you understand by Virtual DOM? Explain its working.

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.

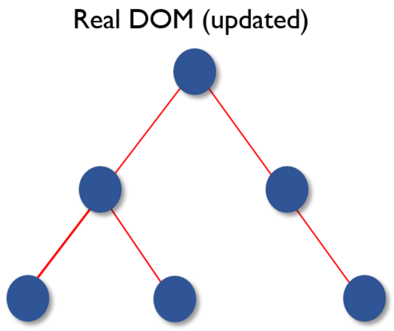
1. Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.



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.



7. 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.

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:

i. require vs import

```
// ES5
var React = require('react');
 // ES6
import React from 'react';
```

ii. export vs export

```
// ES5
module.exports = Component;
 // ES6
export default Component;
```

iii. component and function

```
// ES5
var MyComponent = React.createClass({
    render: function() {
        return
<h3>Hello Edureka!</h3>
;}});
// ES6
class MyComponent extends React.Component {
    render() {
        return
<h3>Hello Edureka!</h3>
;}}
```

iv. props

```
// ES5
var App = React.createClass({
    propTypes: { name: React.PropTypes.string
},
    render: function() {
        return
<h3>Hello, {this.props.name}!</h3>
;
    }
});

// ES6
class App extends React.Component {
    render() {
        return
 <h3>Hello, {this.props.name}!</h3>
;
 }
}
```

v. state

```
// 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?

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

11. “In React, everything is a component.” Explain.

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

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

Each React component must have a **render()** mandatorily. It returns a single React element which is the representation of the native DOM component. If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as **<form>, <group>,<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:

```
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')
);
```

PROPS AND STATE QUESTIONS

14. What is Props?

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

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

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

16. Differentiate between states and props.

States vs Props

Conditions	State	Props
1. Receive initial value from parent component	Yes	Yes
2. Parent component can change value	No	Yes
3. Set default values inside component	Yes	Yes
4. Changes inside component	Yes	No
5. Set initial value for child components	Yes	Yes
6. Changes inside child components	No	Yes

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

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

```
class MyComponent extends React.Component {
    constructor() {
        super();
        this.state = {
            name: 'Maxx',
            id: '101'
        }}
    render() {
  setTimeout(()=>{this.setState({name:'Jaeha', id:'222'})},2000)
   return (
      <div>
        <h1>Hello {this.state.name}</h1>
       <h2>Your Id is {this.state.id}</h2>
     </div>
         );}}
ReactDOM.render(
    <MyComponent/>, document.getElementById('content'));
```

18. What are the differences between state and props?

	State	**Props**
Use	Holds information about the components	Allows to pass data from one component to other components as an argument
Mutability	Is mutable	Are immutable

Read-Only	Can be changed	Are read-only
Child components	Child components cannot access	Child component can access
Stateless components	Cannot have state	Can have props

19. What is prop drilling in React?

Sometimes while developing React applications, there is a need to pass data from a component that is higher in the hierarchy to a component that is deeply nested.
To pass data between such components, we pass props from a source component, and keep passing the prop to the next component in the hierarchy till we reach the deeply nested component.
The disadvantage of using prop drilling is that the components that should otherwise be not aware of the data have access to the data.

REACT LIFECYCLE QUESTIONS

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

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

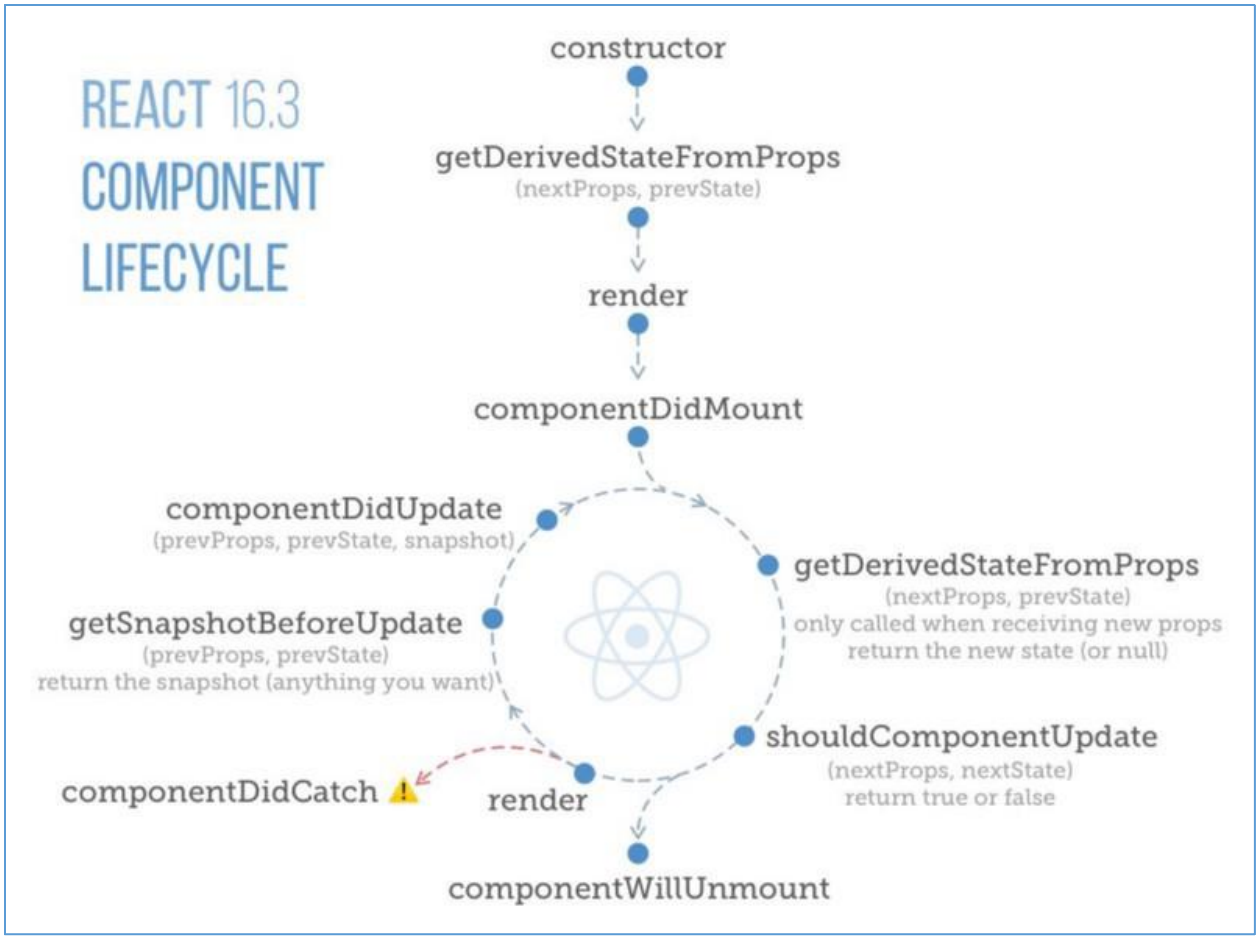
i. *Initial Rendering Phase:* This is the phase when the component is about to start its life journey and make its way to the DOM.
ii. *Updating Phase:* Once the component gets added to the DOM, it can potentially update and re-render only when a prop or state change occurs. That happens only in this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

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

Some of the most important lifecycle methods are: **https://reactjs.org/docs/react-component.html**

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

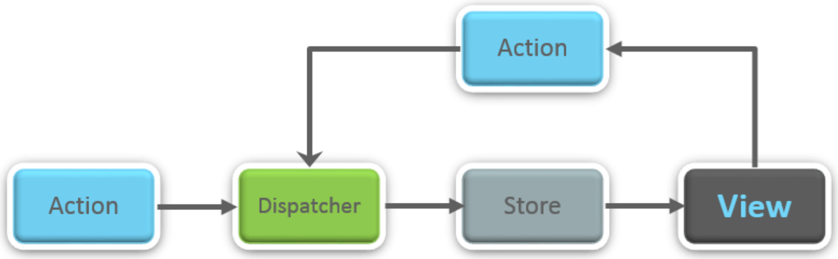
iii. ***componentWillReceiveProps()*** – Invoked as soon as the props are received from the parent class and before another render is called.
iv. ***shouldComponentUpdate()*** – Returns true or false value based on certain conditions. If you want your component to update, return **true** else return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



REDUX QUESTIONS

22. 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.



23. What is Redux?

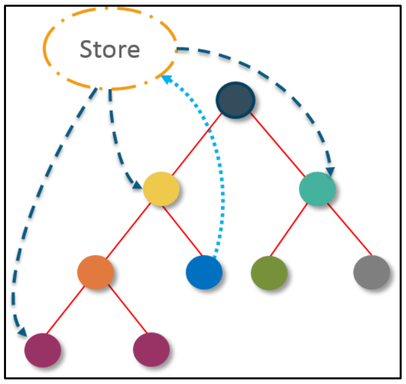
Redux is one of the most trending libraries for front-end development in today's marketplace. It is a predictable state container for JavaScript applications and is used for the entire applications state management. Applications developed with Redux are easy to test and can run in different environments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

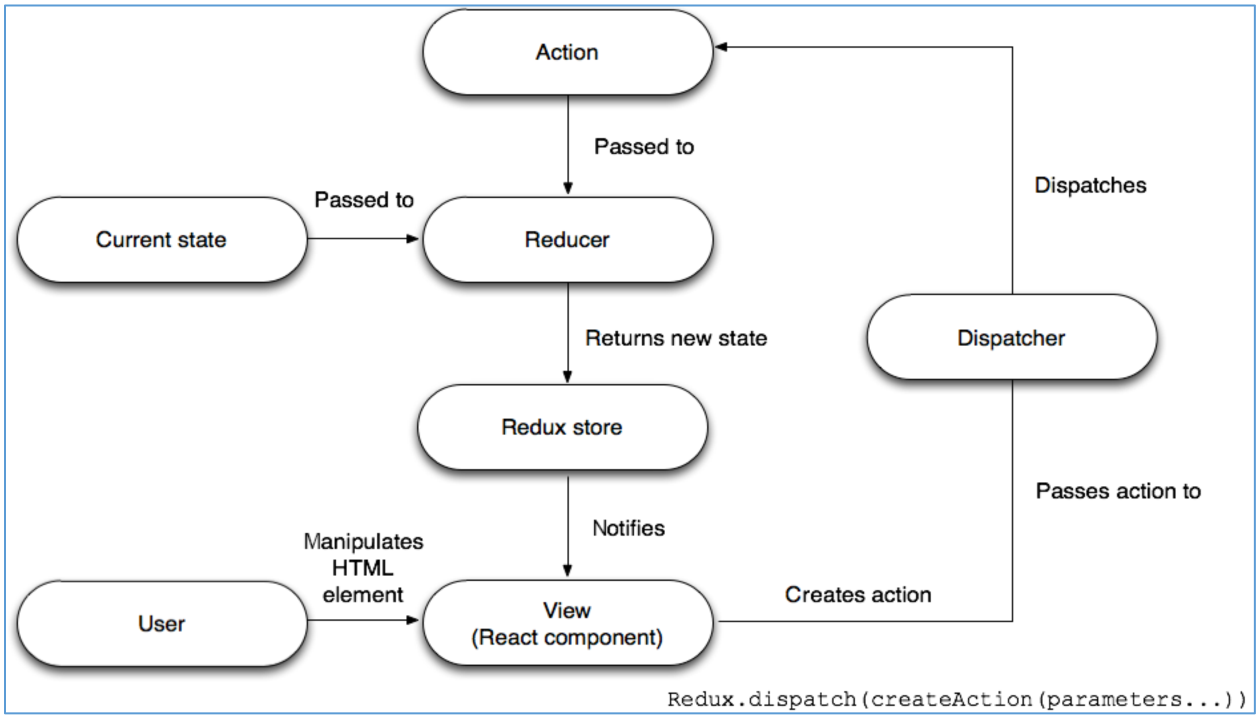
24. What are the three principles that Redux follows?

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

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

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





25. What do you understand by “Single source of truth”?

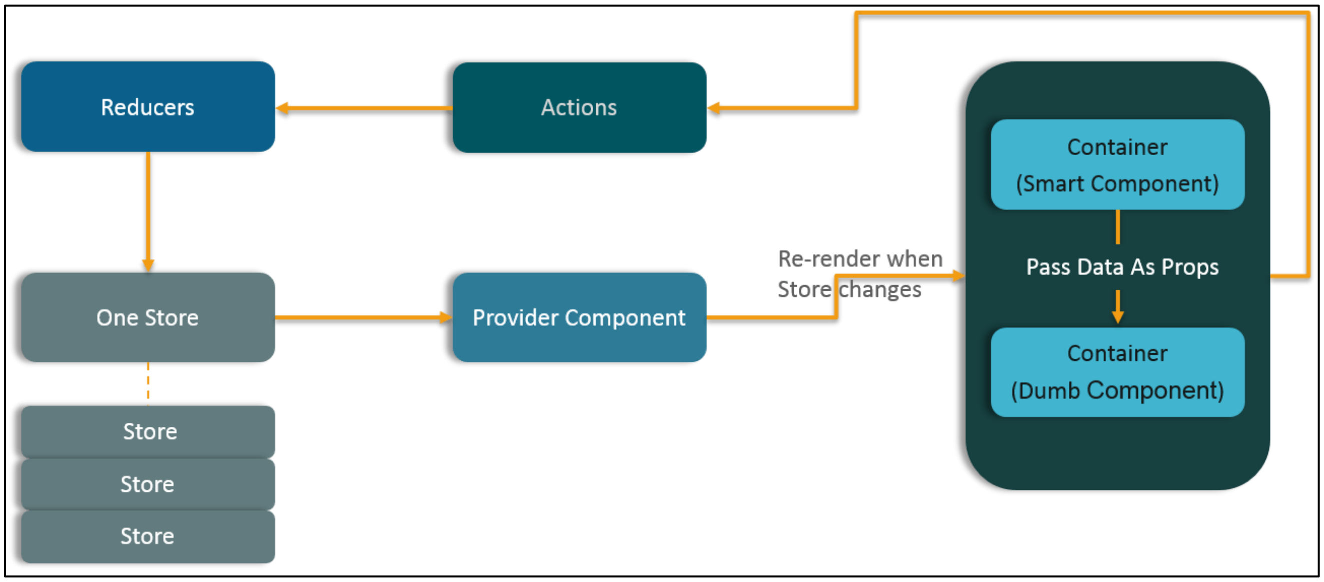
Redux uses ‘Store’ for storing the application’s entire state at one place. So all the component’s state are stored in the Store and they receive updates from the Store itself. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.

26. List down the components of Redux.

Redux is composed of the following components:

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

27. Show how the data flows through Redux?



28. How are Actions defined in Redux?

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

```
function addTodo(text) {

       return{

                type: ADD_TODO,

                 text

    }

}
```

29. 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.

30. 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.

31. 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

32. 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 QUESTIONS

33. 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.

34. 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.

35. 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=&rsquo;/&rsquo;&nbsp;component={Home}/>

    <route path=&rsquo;/posts/:id&rsquo; component={Newpost}/>

    <route path=&rsquo;/posts&rsquo;&nbsp;&nbsp; component={Post}/>

</switch>
```

36. List down the advantages of React Router.

Few advantages are:

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

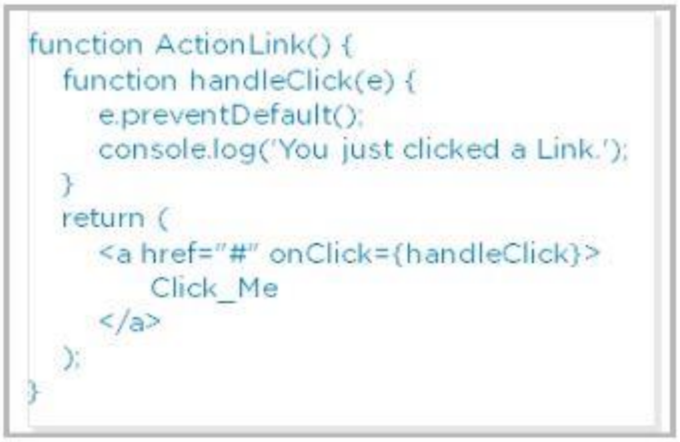
37. How is React Router different from conventional routing?

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

REACT EVENTS QUESTIONS

38. What are synthetic events in React?

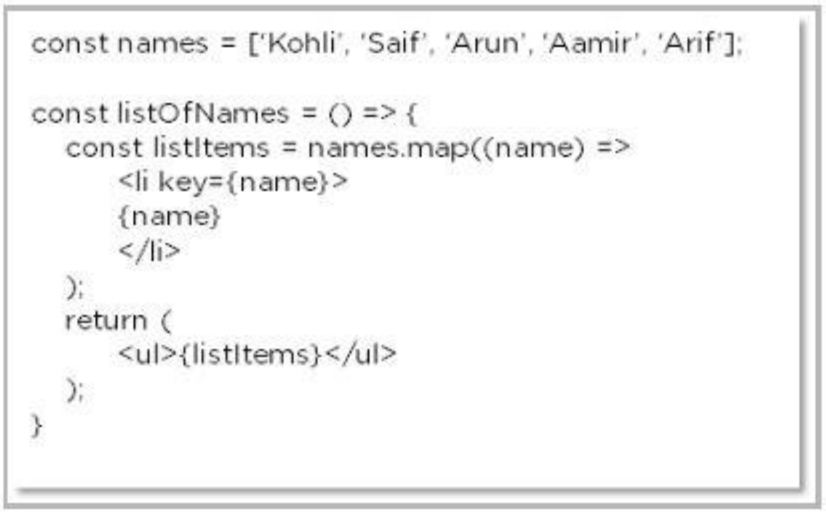
- Synthetic events combine the response of different browser's native events into one API, ensuring that the events are consistent across different browsers.
- The application is consistent regardless of the browser it is running in. Here, preventDefault is a synthetic event.



```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('You just clicked a Link.');
  }
  return (
    <a href="#" onClick={handleClick}>
       Click_Me
    </a>
  );
}
```

39. Explain how lists work in React

- We create lists in React as we do in regular JavaScript. Lists display data in an ordered format
- The traversal of lists is done using the map() function



```
const names = ['Kohli', 'Saif', 'Arun', 'Aamir', 'Arif'];

const listOfNames = () => {
  const listItems = names.map((name) =>
      <li key={name}>
      {name}
      </li>
  );
  return (
      <ul>{listItems}</ul>
  );
}
```

40. Why is there a need for using keys in Lists?

Keys are very important in lists for the following reasons:

- A key is a unique identifier and it is used to identify which items have changed, been updated or deleted from the lists
- It also helps to determine which components need to be re-rendered instead of re-rendering all the components every time. Therefore, it increases performance, as only the updated components are re-rendered

41. What is an event in React?

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

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

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

42. How do you create an event in React?

```
class Display extends React.Component({

    show(evt) {

        // code

    },

    render() {

        // Render the div with an onClick prop (value is a function)

        return (

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

        );

    }

});
```

43. What are synthetic events in React?

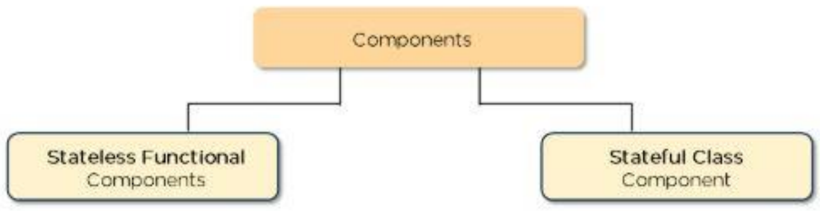
Synthetic events are the objects which act as a cross-browser wrapper around the browser’s native event. They combine the behavior of different browsers into one API. This is done to make sure that the events show consistent properties across different browsers.

COMPONENT QUESTIONS

44. What are the components in React?

Components are the building blocks of any React application, and a single app usually consists of multiple components. A component is essentially a piece of the user interface. It splits the user interface into independent, reusable parts that can be processed separately.

There are two types of components in React:

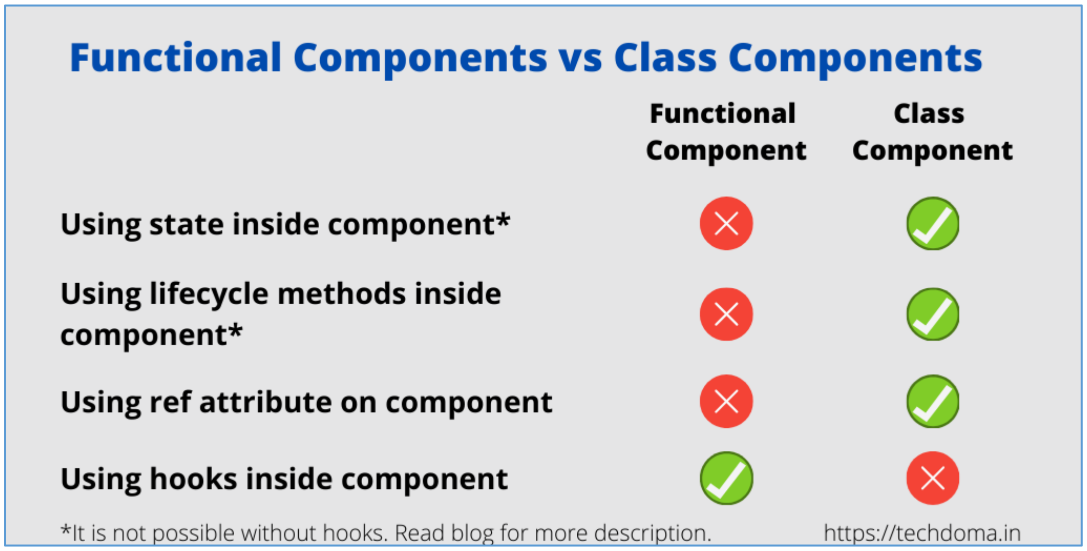


- **Functional Components**: These types of components have no state of their own and only contain render methods, and therefore are also called stateless components. They may derive data from other components as props (properties).

```
function Greeting(props) {

  return <h1>Welcome to {props.name}</h1>;

}
```

- **Class Components**: These types of components can hold and manage their own state and have a separate render method to return JSX on the screen. They are also called Stateful components as they can have a state.

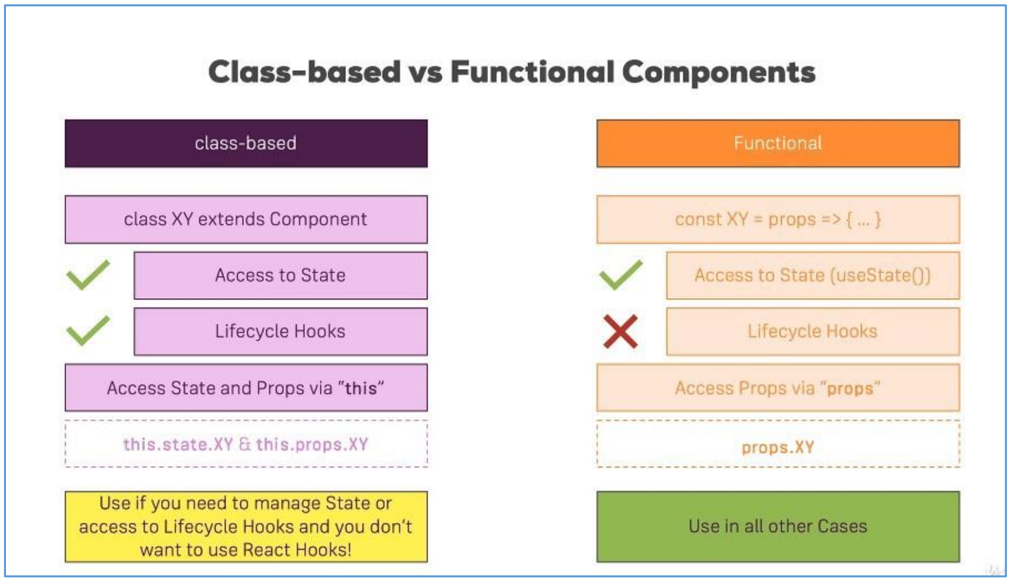
```
class Greeting extends React.Component {

  render() {

    return <h1>Welcome to {this.props.name}</h1>; }}
```



Functional Components vs Class Components

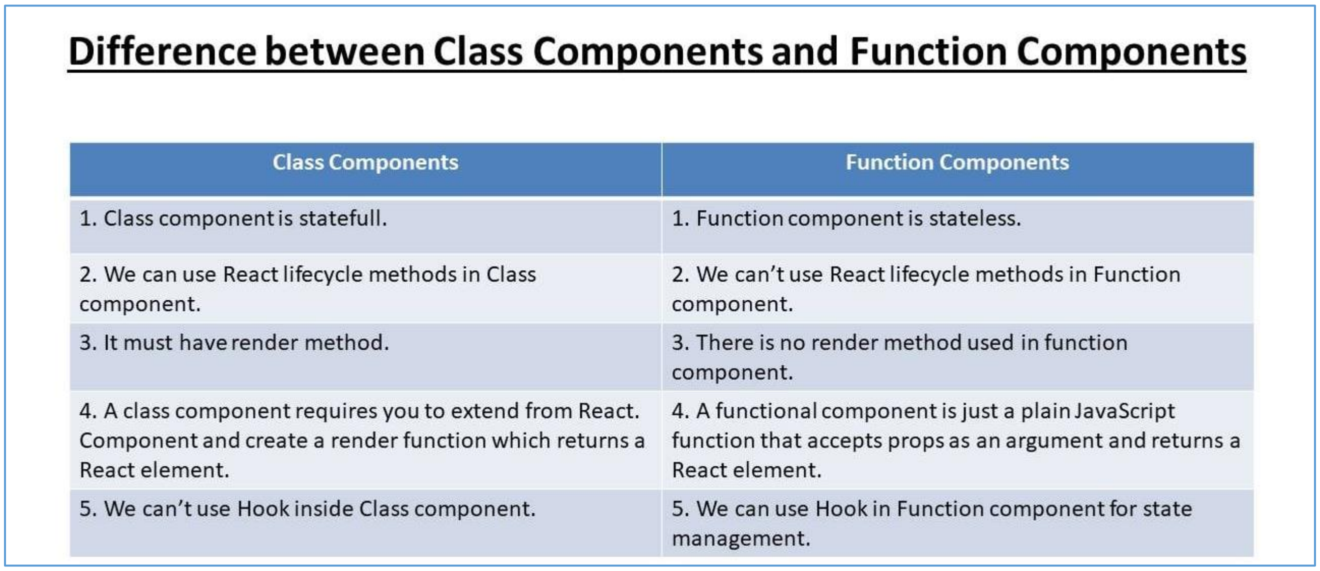
	Functional Component	Class Component
Using state inside component*	✗	✓
Using lifecycle methods inside component*	✗	✓
Using ref attribute on component	✗	✓
Using hooks inside component	✓	✗

*It is not possible without hooks. Read blog for more description. https://techdoma.in



Class-based vs Functional Components

class-based	Functional
class XY extends Component	const XY = props => { ... }
✓ Access to State	✓ Access to State (useState())
✓ Lifecycle Hooks	✗ Lifecycle Hooks
Access State and Props via "**this**"	Access Props via "**props**"
this.state.XY & this.props.XY	props.XY
Use if you need to manage State or access to Lifecycle Hooks and you don't want to use React Hooks!	Use in all other Cases



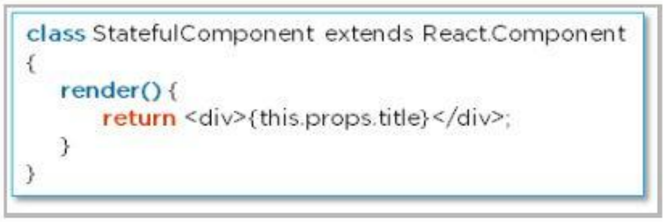
Difference between Class Components and Function Components

Class Components	Function Components
1. Class component is statefull.	1. Function component is stateless.
2. We can use React lifecycle methods in Class component.	2. We can't use React lifecycle methods in Function component.
3. It must have render method.	3. There is no render method used in function component.
4. A class component requires you to extend from React. Component and create a render function which returns a React element.	4. A functional component is just a plain JavaScript function that accepts props as an argument and returns a React element.
5. We can't use Hook inside Class component.	5. We can use Hook in Function component for state management.

45. What are the differences between class and functional components?

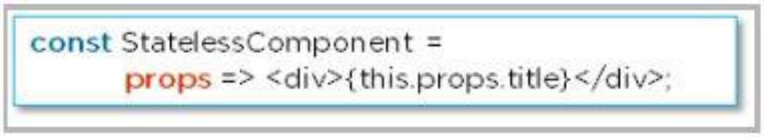
	Class Components	Functional Components
State	Can hold or manage state	Cannot hold or manage state
Simplicity	Complex as compared to the stateless component	Simple and easy to understand
Lifecycle methods	Can work with all lifecycle methods	Does not work with any lifecycle method
Reusability	Can be reused	Cannot be reused

- Class components example:



```
class StatefulComponent extends React.Component
{
    render() {
        return <div>{this.props.title}</div>;
    }
}
```

- Functional components example:



```
const StatelessComponent =
        props => <div>{this.props.title}</div>;
```

46. 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

47. 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.

48. 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

49. 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.

50. 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.

GENERAL QUESTIONS

51. Explain Strict Mode in React.

StrictMode is a tool added in the version 16.3 of React to highlight potential problems in an application. It performs additional checks on the application.

```
function App() {
  return (
    <React.StrictMode>
      <div classname="App">
        <Header/>
        <div>
          Page Content
        </div>
        <Footer/>
      </div>
    </React.StrictMode>
  );
}
```

To enable StrictMode, <React.StrictMode> tags need to be added inside the application:

```
import React from "react";
import ReactDOM from "react-dom";

import App from "./App";

const rootElement = document.getElementById("root");
ReactDOM.render(
 <React.StrictMode>
   <App />
 </React.StrictMode>,
 rootElement
);
```

StrictMode currently helps with the following issues:

- Identifying components with unsafe lifecycle methods
  Certain lifecycle methods are unsafe to use in asynchronous react applications. With the use of third-party libraries it becomes difficult to ensure that certain lifecycle methods are not used.
  StrictMode helps in providing us a warning if any of the class components uses an unsafe lifecycle method.

- Warning about the usage of legacy string API
  If one is using an older version of React, callback ref is the recommended way to manage refs instead of using the string refs. StrictMode gives a warning if we are using string refs to manage refs.

- Warning about the usage of findDOMNode
  Previously, findDOMNode( ) method was used to search the tree of a DOM node. This method is deprecated in React. Hence, the StrictMode gives us a warning about the usage of this method.
- Warning about the usage of legacy context API (because the API is error-prone)

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

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

```
//General way

render() {

    return(

        <MyInput
onChange={this.handleChange.bind(this) } />

    );}

//With Arrow Function

render() {

    return(

        <MyInput onChange={ (e) =>
this.handleOnChange(e) } />

    );}
```

53. What do you understand by refs in React?

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

```
class ReferenceDemo extends React.Component{


     display() {

         const name = this.inputDemo.value;

         document.getElementById('disp').innerHTML = name;

     }

render() {

    return(

         <div>

            Name: <input type="text" ref={input => this.inputDemo = input} />

            <button name="Click" onClick={this.display}>Click</button>

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

       </div>

    );

   }

 }
```

54. 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

55. How do you modularize code in React?

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

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

<div>

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

           </div>

        );
    }
}


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

<div>
                <App />
            </div>

        );
    }
}
```

56. How are forms created in React?

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

```
handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
}
render() {
    return (
<form onSubmit={this.handleSubmit}>
            <label>
                Name:
                <input type="text" value={this.state.value}
onChange={this.handleSubmit} />
            </label>
            <input type="submit" value="Submit" />
        </form>
    );
}
```

57. 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.

58. 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

59. What is the purpose of using super constructor with props argument?

A child class constructor cannot make use of **this** reference until super() method has been called. The same applies for ES6 sub-classes as well. The main reason of passing props parameter to super() call is to access this.props in your child constructors.

60. What are Fragments?

You can use fragment keyword to group a list of children components without using any extra nodes to the DOM.

61. Name a few techniques to optimize React app performance.

There are many ways through which one can optimize the performance of a React app, let's have a look at some of them:

Using useMemo( ) -

It is a React hook that is used for caching CPU-Expensive functions.
Sometimes in a React app, a CPU-Expensive function gets called repeatedly due to re-renders of a component, which can lead to slow rendering.
useMemo( ) hook can be used to cache such functions. By using useMemo( ), the CPU-Expensive function gets called only when it is needed.

Using React.PureComponent -

It is a base component class that checks state and props of a component to know whether the component should be updated.
Instead of using the simple React.Component, we can use React.PureComponent to reduce the re-renders of a component unnecessarily.

Maintaining State Colocation -

This is a process of moving the state as close to where you need it as possible.
Sometimes in React app, we have a lot of unnecessary states inside the parent component which makes the code less readable and harder to maintain. Not to forget, having many states inside a single component leads to unnecessary re-renders for the component.
It is better to shift states which are less valuable to the parent component, to a separate component.

Lazy Loading -

It is a technique used to reduce the load time of a React app. Lazy loading helps reduce the risk of web app performances to minimal.

62. What are error boundaries?

Introduced in the version 16 of React, Error boundaries provide a way for us to catch errors that occur in the render phase.

What is an error boundary?

Any component which uses one of the following lifecycle methods, is considered an error boundary.
In what places can an error boundary detect an error?

- Render phase
- Inside a lifecycle method
- Inside the constructor

Without using error boundaries:

```
class CounterComponent extends React.Component{
 constructor(props){
   super(props);
   this.state = {
     counterValue: 0
   }
   this.incrementCounter = this.incrementCounter.bind(this);
 }

 incrementCounter(){
   this.setState(prevState => counterValue = prevState+1);
 }

 render(){
   if(this.state.counter === 2){
     throw new Error('Crashed');
   }

   return(
     <div>
       <button onClick={this.incrementCounter}>Increment Value</button>
       <p>Value of counter: {this.state.counterValue}</p>
     </div>
   )
 }
}
```

In the code above, when the counterValue equals to 2, we throw an error inside the render method.
When we are not using the error boundary, instead of seeing an error, we see a blank page.
Since any error inside the render method, leads to unmounting of the component.
To display an error that occurs inside the render method, we use error boundaries.

With error boundaries:
As mentioned above, error boundary is a component using one or both of the following methods:

static getDerivedStateFromError and componentDidCatch.
Let's create an error boundary to handle errors in render phase:

```
class ErrorBoundary extends React.Component {
 constructor(props) {
   super(props);
   this.state = { hasError: false };
 }

 static getDerivedStateFromError(error) {
   return { hasError: true };
 }
  componentDidCatch(error, errorInfo) {
   logErrorToMyService(error, errorInfo);
 }

 render() {
   if (this.state.hasError) {
     return <h4>Something went wrong</h4>
   }
   return this.props.children;
 }
}
```

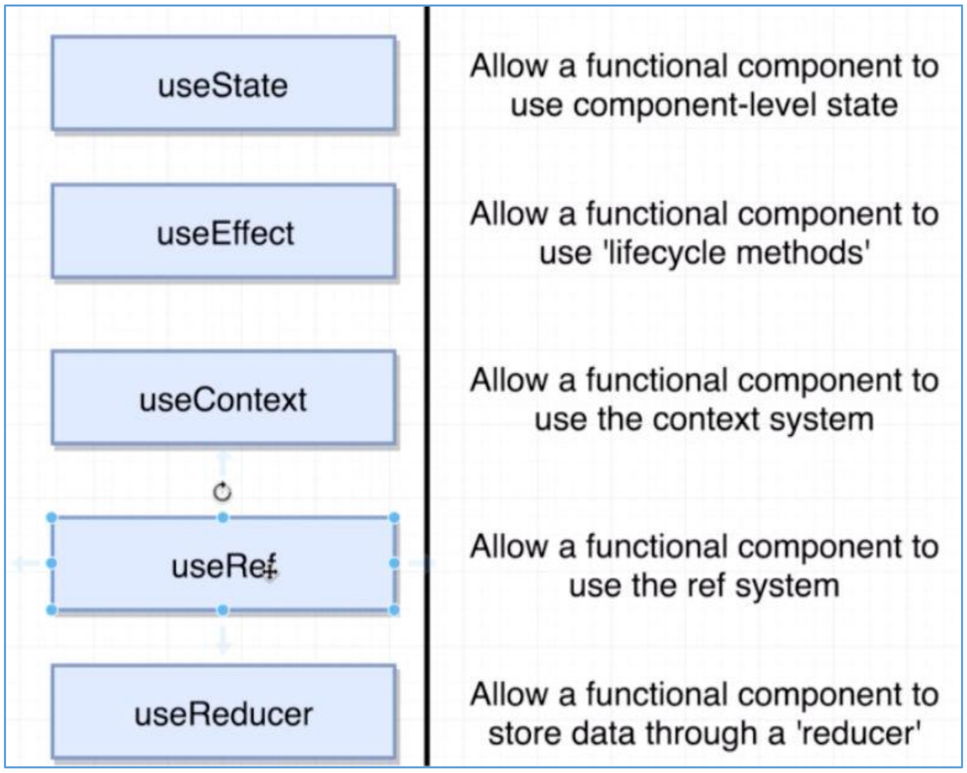
In the code above, **getDerivedStateFromError** function renders the fallback UI interface when the render method has an error.
componentDidCatch logs the error information to an error tracking service.
Now with error boundary, we can render the CounterComponent in the following way:

```
<ErrorBoundary>
  <CounterComponent/>
</ErrorBoundary>
```

REACT HOOKS QUESTIONS

63. What are Hooks?

Hooks are functions that let us “hook into” React state and lifecycle features from **a functional component**.
React Hooks **cannot** be used in class components. They let us write components without class.


useState
Allow a functional component to use component-level state
useEffect
Allow a functional component to use 'lifecycle methods'
useContext
Allow a functional component to use the context system
useRef
Allow a functional component to use the ref system
useReducer
Allow a functional component to store data through a 'reducer'

64. Why were Hooks introduced in React?

React hooks were introduced in the 16.8 version of React.
Previously, functional components were called stateless components. Only class components were used for state management and lifecycle methods.
The need to change a functional component to a class component, whenever state management or lifecycle methods were to be used, led to the development of Hooks.

65. Why should I use hooks?

- Hooks allow for simply reuse of stateful logic without layering abstractions like HoCs and Render Props (See the question about code reuse)
- Fully opt-in and backward compatible
- Hooks make complicated components easier to understand by grouping related code together into functions
- Hooks allow you to avoid Class components which introduce unnecessary complications

66. What is the useState hook?

- useState is a Hook that lets you add React state to function components
- useState like all Hooks is a function
- Argument: the initial state
- Returns: Pair containing the current state and a function to update it.

67. What is the useEffect hook?

- useEffect lets you perform side effects in function components
- useEffect is triggered after a render
- useEffect is like the combination

  of componentDidMount, componentDidUpdateand componentWillUnmount
- Arguments: function to be called, and an array for React to check for changes in order to render

68. What is the useReducer hook?

- useReducer is an alternative to useState that is used when there is complex state logic that that involves multiple sub-values or when the next state depends on the previous one
- 3 Arguments: Reducer function, initial State Object, and a function to initialize the state lazily
- Reducer function takes the current state and action variable and returns the next state
- Returns: a pair containing the current state, and a dispatch function for dispatching an action.
- Works similarly to Redux

69. How do does the performance of using hooks compare to that of classes?

- Hooks avoid a lot of the overhead that is present with classes such as the creation of instances and binding of events

- Hooks result in smaller component trees because it avoids the nesting present in HoCs and render props resulting in less work for React to do.

70. What Are The 2 Rules You Must Follow While Using Hooks?

There are 2 rules which are imposed while you are coding with Hooks:

- React Hooks should only be called at the Top Level. They shouldn't be called inside loops, nested functions or conditions.
- Hooks can only be called from React Function Components.

71. What is the difference between setState() and useEffect() in React?

Simply put, setState() allows programmers to set the state of functional components whereas the useEffect() method allows programmers to use lifecycle methods to React.

72. Is it possible to use both classes and Hooks in my React Code?

While Hooks cannot be a part of classes, you can use both within your React Web applications as you wish. Existing React applications will continue to work fine and there are no plans of removing classes from React in the near future. However, Hooks offer greater flexibility and easier code understanding therefore your company or organization should quickly transition to it.

73. Do Hooks work with static typing?

Static typing is a process of checking code during compile time to ensure that all variables are statically typed. Hooks are primarily functions and have been designed in a way to ensure that all attributes are statically typed. You can also use the React API with custom Hooks if you want to enforce stricter static typing within your code.

74. What functionalities that are provided by classes and are not covered by Hooks yet?

The following methods have not been introduced in Hooks yet:

1. getSnapshotBeforeUpdate
2. getDerivedStateFromError
3. componentDidCatch

These are uncommon lifecycles, however. Furthermore, Hooks may not be compatible with certain third-party libraries.

CONTEXT API QUESTIONS

75. What is the purpose of context API in react?

If a child component at nth level require a property from a parent component at any level, the information needs to be passed one level by level through props. In an application with lot of nested components, it is difficult. Context API helps to directly send an information from a parent component to a child component at any level.