# Crack Your
## React Interview

**By-Sahosoft Solution**

# About The Book

This book is based on React Interview Questions which are mainly asked in most of the interviews. We are pleased to say that we got the idea of this book while looking for the perfect solutions for our React-related queries. Unfortunately, in most cases, we couldn't find enough information or satisfactory solutions. Often, we had to search through multiple websites, tutorials, and links to collect enough information to resolve our queries and after that we used to compile for further convenience. After facing this type of situation, we planned to write a book which covers almost every React-based interview question in addition to covering almost all the topics of Angular.

We think this book will help you to prepare in a better way towards cracking your interviews. This is the reason behind the title of book, i.e., **"Crack Your React Interview"**.

We believe that you also believe that reading multiple questions related to a subject is a great way to learn and revise the concept. It helps to enhance the knowledge in a great helpful way. In this book, we tried to cover most of the important concepts of each version of Angular in the form of interview questions and answers with examples.

Although we have tried to make this book as accurate as possible but if there is something that is not required or you find an error in the book, please let us know.

# About The Authors

**Chandan Kumar Singh**

Chandan Kumar Singh has an experience of 12+ years in developing enterprise-level applications using: C#, ASP.NET4.0 with MVC, Angular 2, Angular 4+, React JS, Node JS, Mongo DB, Express JS , LINQ, SQL Server, AJAX, XML, JavaScript, and jQuery; and analyzing the business requirements and translating it to technical specifications.

He holds a vast experience in all the phases of software development life cycle (SDLC) including requirements gathering, analysis, design, reviews, coding, testing, debugging, documenting, unit and integration testing.

Chandan is a good communicator with a bunch of interpersonal skills that help him in leveraging technical, business acumen to communicate effectively with client executives and their respective teams and recognizing project business needs and presenting solutions.

**Ajeet Kumar Singh**

I have been working for 10 years in software developing field. I have designed and developed applications using C#, ASP.NET4.0 with MVC, LINQ, SQL Server, AngularJS, Angular (version 2+), React JS, Node JS, Mongo DB, Express JS etc.
Experience in all the phases of software development life cycle (SDLC): including requirements gathering, analysis, design, reviews, coding, testing, debugging, documenting, unit and integration testing.
I love to do work with client-side framework like Angular, Angular (ver. 2+), React, Vue, etc. and learn all the new technologies also with the expertise to grasp new concepts quickly and utilize the same in a productive manner.

## 1. What is React?

React is an **open-source frontend JavaScript library** which is used for building user interfaces especially for single page applications. It is used for handling view layer for web and mobile apps. React was created by Jordan Walke, a software engineer working for Facebook. React was first deployed on Facebook's News Feed in 2011 and on Instagram in 2012.

## 2. What are the major features of React?

The major features of React are:

- o It uses **VirtualDOM** instead of RealDOM considering that RealDOM manipulations are expensive.
- o Supports **server-side rendering**.
- o Follows **Unidirectional** data flow or data binding.
- o Uses **reusable/composable** UI components to develop the view.

## 3. What is JSX?

*JSX* is a XML-like syntax extension to ECMAScript (the acronym stands for *JavaScript XML*). Basically it just provides syntactic sugar for the `React.createElement()` function, giving us expressiveness of JavaScript along with HTML like template syntax.
In the example below text inside <h1> tag is returned as JavaScript function to the render function.

```
class App extends React.Component {
    render() {
      return(
        <div>
          <h1>{'Welcome to React world!'}</h1>
        </div>
      )
    }
```

```
    }
```

## 4. What is the difference between Element and Component?

An *Element* is a plain object describing what you want to appear on the screen in terms of the DOM nodes or other components. *Elements* can contain other *Elements* in their props. Creating a React element is cheap. Once an element is created, it is never mutated.

The object representation of React Element would be as follows:

```
const element = React.createElement(
  'div',
  {id: 'login-btn'},
  'Login'
)
```

The above `React.createElement()` function returns an object:

```
{
  type: 'div',
  props: {
    children: 'Login',
    id: 'login-btn'
  }
}
```

And finally it renders to the DOM using `ReactDOM.render()`:

```
<div id='login-btn'>Login</div>
```

Whereas a **component** can be declared in several different ways. It can be a class with a `render()` method. Alternatively, in simple cases, it can be defined as a function. In either case, it takes props as an input, and returns a JSX tree as the output:

```
const Button = ({ onLogin }) =>
  <div id={'login-btn'} onClick={onLogin}>Login</div>
```

Then JSX gets transpiled to a `React.createElement()` function tree:

```
const Button = ({ onLogin }) => React.createElement(
    'div',
    { id: 'login-btn', onClick: onLogin },
    'Login'
  )
```

## 5. How to create components in React?

There are two possible ways to create a component.

**Function Components:** This is the simplest way to create a component. Those are pure JavaScript functions that accept props object as first parameter and return React elements:

```
function Greeting({ message }) {
    return <h1>{`Hello, ${message}`}</h1>
}
```

**Class Components:** You can also use ES6 class to define a component. The above function component can be written as:

```
class Greeting extends React.Component {
    render() {
        return <h1>{`Hello, ${this.props.message}`}</h1>
    }
}
```

## 6. When to use a Class Component over a Function Component?

If the component needs state or lifecycle methods then use class component otherwise use function component. However, from React 16.8 with the addition of Hooks, you could use state , lifecycle methods and other features that were only available in class component right in your function component.

## 7. What are Pure Components?

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

## 8. What is state in React?

*State* of a component is an object that holds some information that may change over the lifetime of the component. We should always try to make our state as simple as possible and minimize the number of stateful components.

Let's create an user component with message state,

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

      this.state = {
        message: 'Welcome to React world'
      }
    }

    render() {
      return (
        <div>
          <h1>{this.state.message}</h1>
        </div>
      )
    }
  }
```



state is used for internal
communication inside a Component

State is similar to props, but it is private and fully controlled by the component. i.e, It is not accessible to any component other than the one that owns and sets it.

## 9. What are props in React?

*Props* are inputs to components. They are single values or objects containing a set of values that are passed to components on creation using a naming convention similar to HTML-tag attributes. They are data passed down from a parent component to a child component.

The primary purpose of props in React is to provide following component functionality:

- ✓ Pass custom data to your component.
- ✓ Trigger state changes.
- ✓ Use via `this.props.reactProp` inside component's `render()` method.

For example, let us create an element with `reactProp` property:
```
<Element reactProp={'1'} />
```

This `reactProp` (or whatever you came up with) name then becomes a property attached to React's native props object which originally already exists on all components created using React library.
```
props.reactProp
```

## 10.     What is the difference between state and props?

Both *props* and *state* are plain JavaScript objects. While both of them hold information that influences the output of render, they are different in their functionality with respect to component. Props get passed to the component similar to function parameters whereas state is managed within the component similar to variables declared within a function.

## 11.     Why should we not update the state directly?

If you try to update state directly then it won't re-render the component.

```
//Wrong
this.state.message = 'Hello world'
```

Instead use `setState()` method. It schedules an update to a component's state object. When state changes, the component responds by re-rendering.

```
//Correct
this.setState({ message: 'Hello World' })
```

**Note:** You can directly assign to the state object either in *constructor* or using latest javascript's class field declaration syntax.

## 12. What is the purpose of callback function as an argument of `setState()`?

The callback function is invoked when setState finished and the component gets rendered. Since `setState()` is **asynchronous** the callback function is used for any post action.

**Note:** It is recommended to use lifecycle method rather than this callback function.

```
setState({ name: 'John' }, () => console.log('The name has upda
ted and component re-rendered'));
```

## 13. What is the difference between HTML and React event handling?

Below are some of the main differences between HTML and React event handling,

i.   In HTML, the event name should be in *lowercase*:

```
<button onclick='activateLasers()'/>
```

Whereas in React it follows *camelCase* convention:

```
<button onClick='activateLasers()'/>
```

ii.  In HTML, you can return `false` to prevent default behavior:

```
<a href='#' onclick='console.log("The link was clicked.");
 return false;' />
```

Whereas in React you must call preventDefault() explicitly:

```
function handleClick(event) {
    event.preventDefault()
    console.log('The link was clicked.')
}
```

iii.  In HTML, you need to invoke the function by **appending()** Whereas in react you should not **append()** with the function name. (refer "**activateLasers**" function in the first point for example)

## 14.  How to bind methods or event handlers in JSX callbacks?

There are 3 possible ways to achieve this:

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

```
class Component extends React.Component {
    constructor(props) {
        super(props)
        this.handleClick = this.handleClick.bind(this)
    }

    handleClick() {
        // ...
    }
}
```

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

```
handleClick = () => {
    console.log('this is:', this)
}
<button onClick={this.handleClick}>
    {'Click me'}
```

```
</button>
```

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

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

**Note:** If the callback is passed as prop to child components, those components might do an extra re-rendering. In those cases, it is preferred to go with **.bind()** or *public class fields syntax* approach considering performance.

## 15.    How to pass a parameter to an event handler or callback?

You can use an *arrow function* to wrap around an *event handler* and pass parameters:

```
<button onClick={() => this.handleClick(id)} />
```
This is an equivalent to calling .bind:

```
<button onClick={this.handleClick.bind(this, id)} />
```

Apart from these two approaches, you can also pass arguments to a function which is defined as arrow function

```
<button onClick={this.handleClick(id)} />
handleClick = (id) => () => {
    console.log("Hello, your ticket number is", id)
};
```

## 16.    What are synthetic events in React?

SyntheticEvent is a cross-browser wrapper around the browser's native event. It's API is same as the browser's native event, including **stopPropagation()** and **preventDefault()**, except the events work identically across all browsers.

## 17.    What are inline conditional expressions?

You can use either *if statements* or *ternary expressions* which are available from JS to conditionally render expressions. Apart from these approaches, you can also embed any expressions in JSX by wrapping them in curly braces and then followed by JS logical operator **&&**.

```
<h1>Hello!</h1>
{

    messages.length > 0 && !isLogin ?
      <h2>
          You have {messages.length} unread messages.
      </h2>
        :
        <h2>
            You don't have unread messages.
        </h2>
}
```

## 18.     What is "key" prop and what is the benefit of using it in arrays of elements?

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

Most often we use ID from our data as *key*:

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

When you don't have stable IDs for rendered items, you may use the item *index* as a *key* as a last resort:

```
const todoItems = todos.map((todo, index) =>
    <li key={index}>
        {todo.text}
    </li>
)
```

**Note:**

i. Using *indexes* for *keys* is **not recommended** if the order of items may change. This can negatively impact performance and may cause issues with component state.

ii. If you extract list item as separate component then apply *keys* on list component instead of `li` tag.

iii. There will be a warning message in the console if the `key` prop is not present on list items.

## 19.    What is the use of refs?

The *ref* is used to return a reference to the element. They *should be avoided* in most cases, however, they can be useful when you need a direct access to the DOM element or an instance of a component.

## 20.    How to create refs?

There are two approaches

This is a recently added approach. *Refs* are created using `React.createRef()` method and attached to React elements via the `ref` attribute. In order to use *refs* throughout the component, just assign the *ref* to the instance property within constructor.

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props)
        this.myRef = React.createRef()
    }
    render() {
        return <div ref={this.myRef} />
    }
}
```

You can also use ref callbacks approach regardless of React version. For example, the search bar component's input element accessed as follows,

```jsx
class SearchBar extends Component {
    constructor(props) {
        super(props);
        this.txtSearch = null;
        this.state = { term: '' };
        this.setInputSearchRef = e => {
            this.txtSearch = e;
        }
    }
    onInputChange(event) {
    this.setState({ term: this.txtSearch.value });
  }
    render() {
        return (
        <input
            value={this.state.term}
            onChange={this.onInputChange.bind(this)}
            ref={this.setInputSearchRef} />
        );
    }
}
```

You can also use *refs* in function components using **closures**. **Note**: You can also use inline ref callbacks even though it is not a recommended approach

## 21.    What are forward refs?

*Ref forwarding* is a feature that lets some components take a *ref* they receive, and pass it further down to a child.

```jsx
const ButtonElement = React.forwardRef((props, ref) => (
    <button ref={ref} className="CustomButton">
        {props.children}
    </button>
));

// Create ref to the DOM button:
const ref = React.createRef();
<ButtonElement ref={ref}>{'Forward Ref'}</ButtonElement>
```

## 22. Which is preferred option with in callback refs and findDOMNode()?

It is preferred to use *callback refs* over **findDOMNode()** API.
Because **findDOMNode()** prevents certain improvements in React in the future.
The **legacy** approach of using findDOMNode:

```
class MyComponent extends Component {
    componentDidMount() {
        findDOMNode(this).scrollIntoView()
    }

    render() {
        return <div />
    }
}
```

The recommended approach is:

```
class MyComponent extends Component {
    constructor(props) {
        super(props);
        this.node = createRef();
    }
    componentDidMount() {
        this.node.current.scrollIntoView();
    }

    render() {
        return <div ref={this.node} />
    }
}
```

## 23. Why are String Refs legacy?

If you worked with React before, you might be familiar with an older API where the ref attribute is a string, like **ref={'textInput'}**, and the DOM node is accessed as **this.refs.textInput**. We advise against it because string refs have below issues, and are considered legacy. String refs were **removed in React v16**.

They *force React to keep track of currently executing component*. This is problematic because it makes react module stateful, and thus causes weird errors when react module is duplicated in the bundle.

**They are *not composable* —** if a library puts a ref on the passed child, the user can't put another ref on it. Callback refs are perfectly composable.

They *don't work with static analysis* like Flow. Flow can't guess the magic that framework does to make the string ref appear on **this.refs**, as well as its type (which could be different). Callback refs are friendlier to static analysis.

It doesn't work as most people would expect with the "render callback" pattern (e.g. )

```
class MyComponent extends Component {
    renderRow = (index) => {
        // This won't work. Ref will get attached to DataTab
le rather than MyComponent:
        return <input ref={'input-' + index} />;

        // This would work though! Callback refs are awesome
.
return <input ref={input => this['input- + index] = input}/>;
    }

    render() {
        return <DataTable data={this.props.data} renderRow={
this.renderRow} />
    }
}
```

## 24. What is Virtual DOM?

The ***Virtual DOM*** (**VDOM**) is an in-memory representation of *Real DOM*. The representation of a UI is kept in memory and synced with the "**real**" **DOM**. It's a step that happens between the render function being called and the displaying of elements on the screen. This entire process is called *reconciliation*.
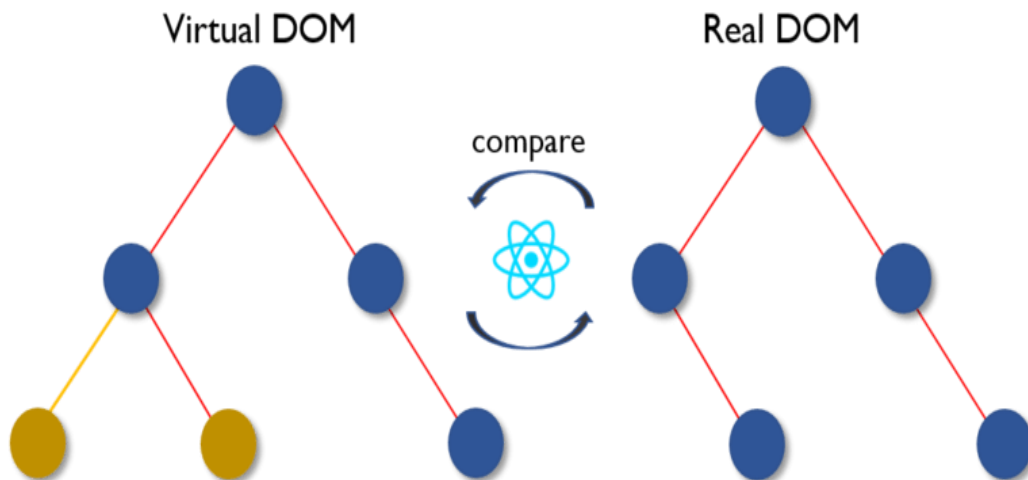
## 25. How Virtual DOM works?

The ***Virtual DOM*** works in three simple steps.

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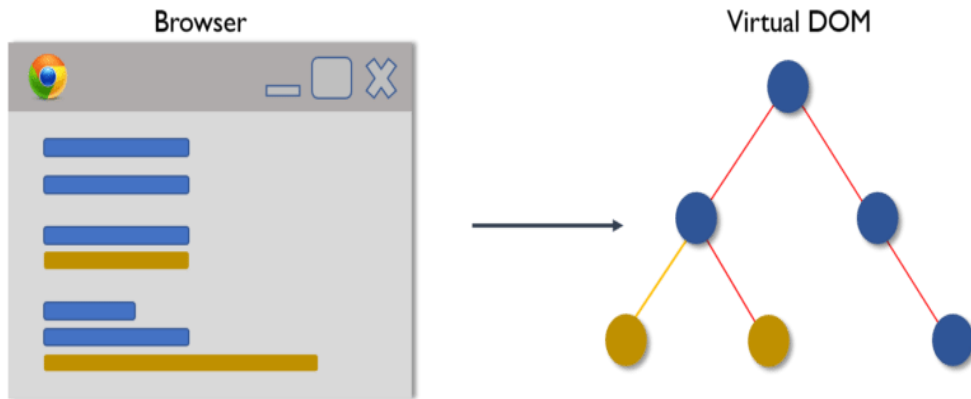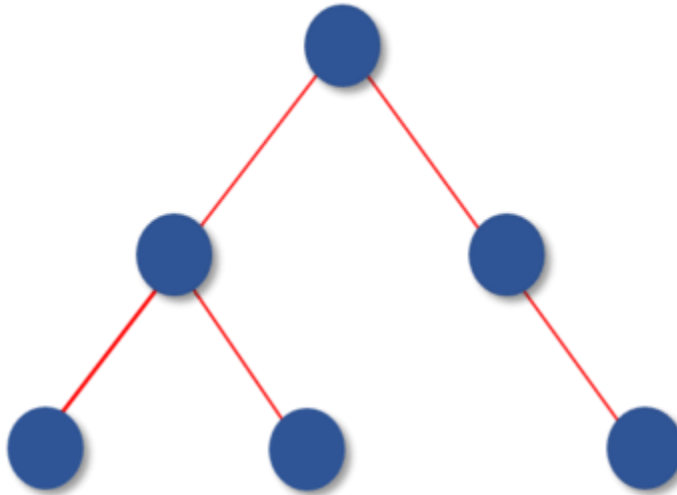


    ii.    Then the difference between the previous DOM representation and the new one is calculated.



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

## Real DOM (updated)



**26.** **What is the difference between Shadow DOM and Virtual DOM?**

The ***Shadow DOM*** is a browser technology designed primarily for scoping variables and CSS in *web components*. The ***Virtual DOM*** is a concept implemented by libraries in JavaScript on top of browser APIs.

**27.** **What is React Fiber?**

**Fiber** is the new ***reconciliation* engine** or **reimplementation** of core algorithm in **React v16**. The goal of React Fiber is to increase its suitability for areas like animation, layout, gestures, ability to pause, abort, or reuse work and assign priority to different types of updates; and new concurrency primitives.

**28.** **What is the main goal of React Fiber?**

The goal of *React Fiber* is to increase its suitability for areas like animation, layout, and gestures. Its headline feature is **incremental rendering**: the ability to split rendering work into chunks and spread it out over multiple frames.

**29.** **What are controlled components?**

A component that controls the input elements within the forms on subsequent user input is called **Controlled Component**, i.e, every state mutation will have an associated handler function.

For example, to write all the names in uppercase letters, we use handleChange as below,

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

## 30.      What are uncontrolled components?

The **Uncontrolled Components** are the ones that store their own state internally, and you query the DOM using a ref to find its current value when you need it. This is a bit more like traditional HTML.

In the below UserProfile component, the name input is accessed using ref.

```
class UserProfile extends React.Component {
    constructor(props) {
        super(props)
        this.handleSubmit = this.handleSubmit.bind(this)
        this.input = React.createRef()
    }

    handleSubmit(event) {
        alert('A name was submitted: ' + this.input.current.val
ue)
        event.preventDefault()
    }

    render() {
        return (
            <form onSubmit={this.handleSubmit}>
                <label>
                    {'Name:'}
                    <input type="text" ref={this.input} />
                </label>
                <input type="submit" value="Submit" />
            </form>
        );
```

```
        }
}
```

In most cases, it's recommend to use controlled components to implement forms.

## 31.  What is the difference between createElement and cloneElement?

JSX elements will be transpiled to `React.createElement()` functions to create React elements which are going to be used for the object representation of UI. Whereas `cloneElement` is used to clone an element and pass it new props.

## 32.  What is Lifting State Up in React?

When several components need to share the same changing data then it is recommended to *lift the shared state up* to their closest common ancestor. That means if two child components share the same data from its parent, then move the state to parent instead of maintaining local state in both of the child components.

## 33.  What are the different phases of component lifecycle?

The component lifecycle has three distinct lifecycle phases:

i.  **Mounting:** The component is ready to mount in the browser DOM. This phase covers initialization from `constructor()`, `getDerivedStateFromProps()`, `render()`, and `componentDidMount()` lifecycle methods.

ii.  **Updating:** In this phase, the component get updated in two ways, sending the new props and updating the state either from `setState()` or `forceUpdate()`. This phase covers `getDerivedStateFromProps()`, `shouldComponentUpdate()`, `render()`, `getSnapshotBeforeUpdate()` and `componentDidUpdate()` lifecycle methods.

iii. **Unmounting:** In this last phase, the component is not needed and get unmounted from the browser DOM. This phase includes `componentWillUnmount()` lifecycle method.
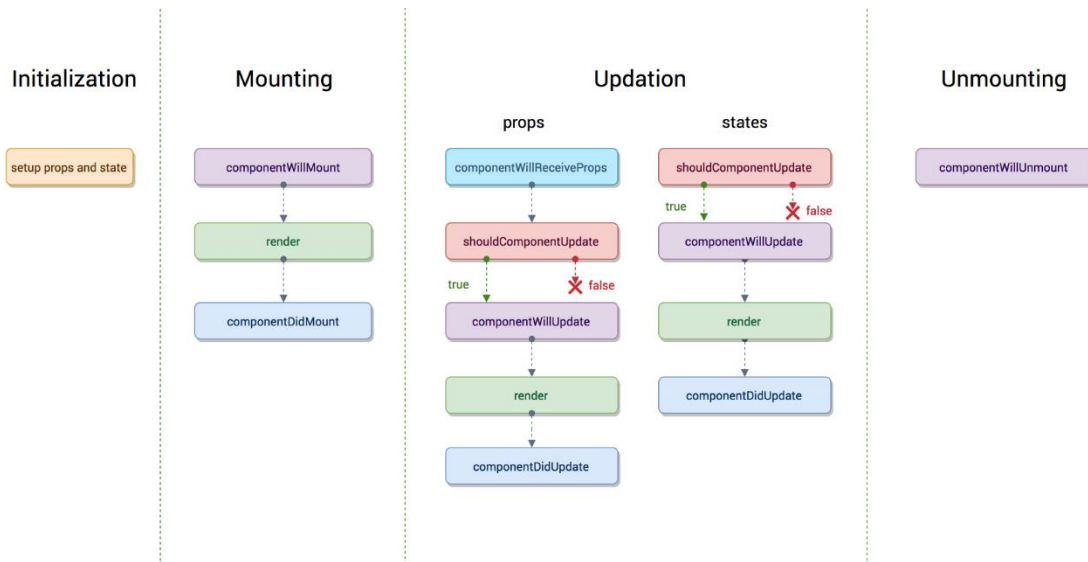
It's worth mentioning that React internally has a concept of phases when applying changes to the DOM. They are separated as follows

iv. **Render** The component will render without any side-effects. This applies for Pure components and in this phase, React can pause, abort, or restart the render.

v. **Pre-commit** Before the component actually applies the changes to the DOM, there is a moment that allows React to read from the DOM through the `getSnapshotBeforeUpdate()`.

vi. **Commit** React works with the DOM and executes the final lifecycles respectively `componentDidMount()` for mounting, `componentDidUpdate()` for updating, and `componentWillUnmount()` for unmounting.

React 16.3+ Phases (or an interactive version)



Before React 16.3

| Initialization | Mounting | Updation | | Unmounting |
|---|---|---|---|---|

**props** **states**

setup props and state — componentWillMount → render → componentDidMount

componentWillReceiveProps → shouldComponentUpdate → (true) componentWillUpdate → render → componentDidUpdate ; (false) ✗

shouldComponentUpdate → (true) componentWillUpdate → render → componentDidUpdate ; (false) ✗
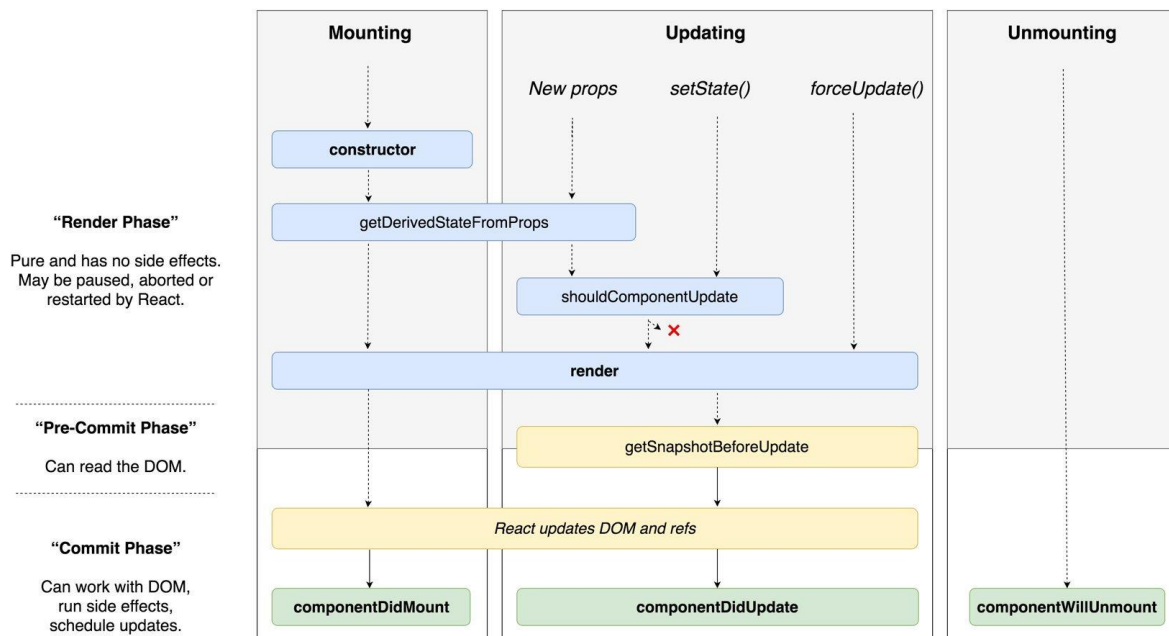
componentWillUnmount

## 34. What are the lifecycle methods of React?

Before React 16.3

- **componentWillMount:** Executed before rendering and is used for App level configuration in your root component.
- **componentDidMount:** Executed after first rendering and here all AJAX requests, DOM or state updates, and set up event listeners should occur.
- **componentWillReceiveProps:** Executed when particular prop updates to trigger state transitions.
- **shouldComponentUpdate:** Determines if the component will be updated or not. By default it returns `true`. If you are sure that the component doesn't need to render after state or props are updated, you can return false value. It is a great place to improve performance as it allows you to prevent a re-render if component receives new prop.
- **componentWillUpdate:** Executed before re-rendering the component when there are props & state changes confirmed by `shouldComponentUpdate()` which returns true.
- **componentDidUpdate:** Mostly it is used to update the DOM in response to prop or state changes.
- **componentWillUnmount:** It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component.

React 16.3+

- o **getDerivedStateFromProps:** Invoked right before calling `render()` and is invoked on *every* render. This exists for rare use cases where you need derived state. Worth reading if you need derived state.
- o **componentDidMount:** Executed after first rendering and here all AJAX requests, DOM or state updates, and set up event listeners should occur.
- o **shouldComponentUpdate:** Determines if the component will be updated or not. By default it returns `true`. If you are sure that the component doesn't need to render after state or props are updated, you can return false value. It is a great place to improve performance as it allows you to prevent a re-render if component receives new prop.
- o **getSnapshotBeforeUpdate:** Executed right before rendered output is committed to the DOM. Any value returned by this will be passed into `componentDidUpdate()`. This is useful to capture information from the DOM i.e. scroll position.
- o **componentDidUpdate:** Mostly it is used to update the DOM in response to prop or state changes. This will not fire if `shouldComponentUpdate()` returns `false`.
- o **componentWillUnmount** It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component.

## 35.     What are Higher-Order Components?

A *higher-order component* (*HOC*) is a function that takes a component and returns a new component. Basically, it's a pattern that is derived from React's compositional nature.

We call them **pure components** because they can accept any dynamically provided child component but they won't modify or copy any behavior from their input components.

```
const EnhancedComponent =
higherOrderComponent(WrappedComponent)
```

HOC can be used for many use cases:

- ❖ Code reuse, logic and bootstrap abstraction.
- ❖ Render hijacking.

❖ State abstraction and manipulation.

❖ Props manipulation.

## 36. How to create props proxy for HOC component?

You can add/edit props passed to the component using *props proxy* pattern like this:

```javascript
function HOC(WrappedComponent) {
    return class Test extends Component {
        render() {
            const newProps = {
                title: 'New Header',
                footer: false,
                showFeatureX: false,
                showFeatureY: true
            }

            return <WrappedComponent {...this.props} {...newProps} />
        }
    }
}
```

## 37. What is context?

*Context* provides a way to pass data through the component tree without having to pass props down manually at every level.

For example, authenticated user, locale preference, UI theme need to be accessed in the application by many components.

```javascript
const {Provider, Consumer} =
            React.createContext(defaultValue)
```

## 38. What is children prop?

*Children* is a prop (**this.props.children**) that allow you to pass components as data to other components, just like any other prop you use. Component

tree put between component's opening and closing tag will be passed to that component as `children` prop.

There are a number of methods available in the React API to work with this prop.

These include **React.Children.map, React.Children.forEach, React.Children.count, React.Children.only, React.Children.toArray**.

A simple usage of children prop looks as below,

```jsx
const MyDiv = React.createClass({
    render: function () {
        return <div>{this.props.children}</div>
    }
})

ReactDOM.render(
    <MyDiv>
        <span>{'Hello'}</span>
        <span>{'World'}</span>
    </MyDiv>,
    node
)
```

## 39.     How to write comments in React?

The comments in **React/JSX** are similar to JavaScript Multiline comments but are wrapped in curly braces.

**Single-line comments:**

```jsx
<div>
    {/* Single-line comments(In vanilla JavaScript, the single-line comments are represented by double slash(//)) */}
    {`Welcome ${user}, let's play React`}
</div>
```

**Multi-line comments:**

```jsx
<div>
    {/* Multi-line comments for more than
    one line */}
    {`Welcome ${user}, let's play React`}
</div>
```

## 40. 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.

**Passing props:**

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

        console.log(this.props)
        // prints { name: 'John', age: 42 }
    }
}
```

**Not passing props:**

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

        console.log(this.props)
        // prints undefined

        // but props parameter is still available
        console.log(props)
        // prints { name: 'John', age: 42 }
    }

    render() {
        // no difference outside constructor
        console.log(this.props)
        // prints { name: 'John', age: 42 }
    }
}
```

The above code snippets reveals that `this.props` is different only within the constructor. It would be the same outside the constructor.

## 41. What is reconciliation?

When a component's props or state change, React decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React will update the DOM. This process is called *reconciliation*.

## 42. How to set state with a dynamic key name?

If you are using **ES6** or the Babel transpiler to transform your JSX code then you can accomplish this with *computed property names*.

```
handleInputChange(event) {
this.setState({ [event.target.id]: event.target.value })
}
```

## 43. What would be the common mistake of function being called every time the component renders?

You need to make sure that function is not being called while passing the function as a parameter.

```
render() {
    // Wrong: handleClick is called instead of passed as a refe
rence!
return <button onClick={this.handleClick()}>{'Click Me'}</butto
n>
}
```

Instead, pass the function itself without parenthesis:

```
render() {
    // Correct: handleClick is passed as a reference!
    return <button onClick={this.handleClick}>{'Click Me'}</but
ton>
    }
```

## 44. Is lazy function supports named exports?

No, currently **React.lazy** function supports default exports only. If you would like to import modules which are named exports, you can create an intermediate module that reexports it as the default. It also ensures that tree shaking keeps working and don't pull unused components. Let's take a component file which exports multiple named components,

```
// MoreComponents.js
export const SomeComponent = /* ... */;
export const UnusedComponent = /* ... */;
```

and reexport MoreComponents.js components in an intermediate file IntermediateComponent.js

```
// IntermediateComponent.js
export { SomeComponent as default } from "./MoreComponents.js";
```

Now you can import the module using lazy function as below,

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

const SomeComponent = lazy(() => import("./IntermediateComponen
t.js"));
```

## 45.    Why React uses `className` over `class` attribute?

`class` is a keyword in JavaScript, and JSX is an extension of JavaScript. That's the principal reason why React uses `className` instead of `class`. Pass a string as the **className** prop.

```
render() {
    return <span className={'menu navigation-
menu'}>{'Menu'}</span>
  }
```

## 46.    What are fragments?

It's common pattern in React which is used for a component to return multiple elements. *Fragments* let you group a list of children without adding extra nodes to the DOM.

```
render() {
    return (
        <React.Fragment>
            <ChildA />
            <ChildB />
            <ChildC />
        </React.Fragment>
    )
}
```

There is also a *shorter syntax*, but it's not supported in many tools:

```
render() {
    return (
        <>
            <ChildA />
            <ChildB />
            <ChildC />
        </>
    )
}
```

## 47.     Why fragments are better than container divs?

Below are the list of reasons,

❖ Fragments are a bit faster and use less memory by not creating an extra DOM node. This only has a real benefit on very large and deep trees.

❖ Some CSS mechanisms like *Flexbox* and *CSS Grid* have a special parent-child relationships, and adding divs in the middle makes it hard to keep the desired layout.

❖ The DOM Inspector is less cluttered.

## 48.     What are portals in React?

*Portal* is a recommended way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.

```
ReactDOM.createPortal(child, container)
```

The first argument is any render-able React child, such as an element, string, or fragment. The second argument is a DOM element.

### 49.    What are stateless components?

If the behaviour is independent of its **state** then it can be a **stateless component**. You can use either a function or a class for creating stateless components. But unless you need to use a **lifecycle hook** in your components, you should go for function components. There are a lot of benefits if you decide to use function components here; they are easy to write, understand, and test, a little faster, and you can avoid the **this** keyword altogether.

### 50.    What are stateful components?

If the behaviour of a component is dependent on the *state* of the component then it can be termed as stateful component. These *stateful components* are always *class components* and have a state that gets initialized in the `constructor`.

```
class App extends Component {
    constructor(props) {
        super(props)
        this.state = { count: 0 }
    }

    render() {
        // ...
    }
}
```

**React 16.8 Update:**

Hooks let you use state and other React features without writing classes.

*The Equivalent Functional Component*

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

const App = (props) => {
    const [count, setCount] = useState(0);
```

```
    return (
    // JSX
  )
}
```

## 51.    How to apply validation on props in React?

When the application is running in *development mode*, React will automatically check all props that we set on components to make sure they have *correct type*. If the type is incorrect, React will generate warning messages in the console. It's disabled in *production mode* due to performance impact. The mandatory props are defined with `isRequired`.

The set of predefined prop types:

   i.    `PropTypes.number`
   ii.   `PropTypes.string`
  iii.   `PropTypes.array`
  iv.   `PropTypes.object`
   v.    `PropTypes.func`
  vi.   `PropTypes.node`
  vii.  `PropTypes.element`
 viii.  `PropTypes.bool`
  ix.   `PropTypes.symbol`
   x.    `PropTypes.any`

We can define `propTypes` for `User` component as below:

```
import React from 'react'
import PropTypes from 'prop-types'

class User extends React.Component {
    static propTypes = {
        name: PropTypes.string.isRequired,
        age: PropTypes.number.isRequired
    }

    render() {
        return (
            <>
```

```
                <h1>{`Welcome, ${this.props.name}`}</h1>
                <h2>{`Age, ${this.props.age}`}</h2>
            </>
        )
    }
}
```

**Note:** In React v15.5 PropTypes were moved from React.PropTypes to prop-types library.

## 52.     What are the advantages of React?

Below are the list of main advantages of React,

i.      Increases the application's performance with *Virtual DOM*.
ii.     JSX makes code easy to read and write.
iii.    It renders both on client and server side (*SSR*).
iv.     Easy to integrate with frameworks (Angular, Backbone) since it is only a view library.
v.      Easy to write unit and integration tests with tools such as Jest.

## 53.     What are the limitations of React?

Apart from the advantages, there are few limitations of React too,

i.      React is just a view library, not a full framework.
ii.     There is a learning curve for beginners who are new to web development.
iii.    Integrating React into a traditional MVC framework requires some additional configuration.
iv.     The code complexity increases with inline templating and JSX.
v.      Too many smaller components leading to over engineering or boilerplate.

## 54.     What are error boundaries in React v16?

*Error boundaries* are components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed.

A class component becomes an error boundary if it defines a new lifecycle method called
**componentDidCatch(error, info)**
or
**static getDerivedStateFromError() :**

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

    componentDidCatch(error, info) {
        // You can also log the error to an error reporting service
        logErrorToMyService(error, info)
    }

    static getDerivedStateFromError(error) {
        // Update state so the next render will show the fallback UI.
        return { hasError: true };
    }

    render() {
        if (this.state.hasError) {
            // You can render any custom fallback UI
            return <h1>{'Something went wrong.'}</h1>
        }
        return this.props.children
    }
}
```

After that use it as a regular component:

```
<ErrorBoundary>
  <MyWidget />
</ErrorBoundary>
```

## 55.     How error boundaries handled in React v15?

**React v15** provided very basic support for *error boundaries* using **unstable_handleError** method. It has been renamed to **componentDidCatch** in React v16.

## 56.     What are the recommended ways for static type checking?

Normally we use *PropTypes library* (React.PropTypes moved to a prop-types package since React v15.5) for *type checking* in the React applications. For large code bases, it is recommended to use *static type checkers* such as Flow or TypeScript, that perform type checking at compile time and provide auto-completion features.

## 57.     What is the use of `react-dom` package?

The `react-dom` package provides *DOM-specific methods* that can be used at the top level of your app. Most of the components are not required to use this module. Some of the methods of this package are:

    i.   render()
   ii.   hydrate()
  iii.   unmountComponentAtNode()
   iv.   findDOMNode()
    v.   createPortal()

## 58.     What is the purpose of render method of `react-dom`?

This method is used to render a React element into the DOM in the supplied container and return a reference to the component. If the React element was previously rendered into container, it will perform an update on it and only mutate the DOM as necessary to reflect the latest changes.

**ReactDOM.render(element, container[, callback])**

If the optional callback is provided, it will be executed after the component is rendered or updated.

## 59.    What is ReactDOMServer?

The `ReactDOMServer` object enables you to render components to static markup (typically used on node server). This object is mainly used for *server-side rendering* (SSR). The following methods can be used in both the server and browser environments:

i.   `renderToString()`
ii.  `renderToStaticMarkup()`

For example, you generally run a Node-based web server like Express, Hapi, or Koa, and you call `renderToString` to render your root component to a string, which you then send as response.

```
// using Express
import { renderToString } from 'react-dom/server'
import MyPage from './MyPage'

app.get('/', (req, res) => {
    res.write('<!DOCTYPE html><html><head><title>My Page</title></head><body>')
    res.write('<div id="content">')
    res.write(renderToString(<MyPage />))
    res.write('</div></body></html>')
    res.end()
})
```

## 60.    How to use innerHTML in React?

The `dangerouslySetInnerHTML` attribute is React's replacement for using `innerHTML` in the browser DOM. Just like `innerHTML`, it is risky to use this attribute considering cross-site scripting (XSS) attacks. You just need to pass a `__html` object as key and HTML text as value.
In this example MyComponent uses `dangerouslySetInnerHTML` attribute for setting HTML markup:

```
function createMarkup() {
    return { __html: 'First &middot; Second' }
  }
```

```
function MyComponent() {
    return <div dangerouslySetInnerHTML={createMarkup()} />
  }
```

## 61. How to use styles in React?

The `style` attribute accepts a JavaScript object with camelCased properties rather than a CSS string. This is consistent with the DOM style JavaScript property, is more efficient, and prevents XSS security holes.

```
const divStyle = {
    color: 'blue',
    backgroundImage: 'url(' + imgUrl + ')'
};

function HelloWorldComponent() {
    return <div style={divStyle}>Hello World!</div>
}
```

Style keys are camelCased in order to be consistent with accessing the properties on DOM nodes in JavaScript (e.g. **node.style.backgroundImage**).

## 62. How events are different in React?

Handling events in React elements has some syntactic differences:

   i.    React event handlers are named using camelCase, rather than lowercase.
   ii.   With JSX you pass a function as the event handler, rather than a string.

## 63. What will happen if you use setState() in constructor?

When you use **setState()**, then apart from assigning to the object state React also re-renders the component and all its children. You would get error like this: *Can only update a mounted or mounting component.* So we need to use **this.state** to initialize variables inside constructor.

## 64. What is the impact of indexes as keys?

Keys should be stable, predictable, and unique so that React can keep track of elements.

In the below code snippet each element's key will be based on ordering, rather than tied to the data that is being represented. This limits the optimizations that React can do.

```
{
    todos.map((todo, index) =>
        <Todo
            {...todo}
            key={index}
        />
    )
}
```

If you use element data for unique key, assuming todo.id is unique to this list and stable, React would be able to reorder elements without needing to reevaluate them as much.

```
{
    todos.map((todo) =>
        <Todo {...todo}
            key={todo.id} />
    )
}
```

**65.    Is it good to use setState() in componentWillMount() method?**

Yes, it is safe to use **setState()** inside **componentWillMount()** method. But at the same it is recommended to avoid async initialization in **componentWillMount()** lifecycle method. **componentWillMount()** is invoked immediately before mounting occurs. It is called before **render()**, therefore setting state in this method will not trigger a re-render. Avoid introducing any side-effects or subscriptions in this method. We need to make sure async calls for component initialization happened in **componentDidMount()** instead of **componentWillMount()**.

```
componentDidMount() {
    axios.get(`api/todos`)
        .then((result) => {
            this.setState({
                messages: [...result.data]
```

```
        })
    })
}
```

## 66.    What will happen if you use props in initial state?

If the props on the component are changed without the component being refreshed, the new prop value will never be displayed because the constructor function will never update the current state of the component. The initialization of state from props only runs when the component is first created.

The below component won't display the updated input value:

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

        this.state = {
            records: [],
            inputValue: this.props.inputValue
        };
    }

    render() {
        return <div>{this.state.inputValue}</div>
    }
}
```

Using props inside render method will update the value:

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

        this.state = {
            record: []
        }
    }

    render() {
        return <div>{this.props.inputValue}</div>
    }
}
```

## 67. How do you conditionally render components?

In some cases you want to render different components depending on some state. JSX does not render `false` or `undefined`, so you can use conditional *short-circuiting* to render a given part of your component only if a certain condition is true.

```jsx
const MyComponent = ({ name, address }) => (
    <div>
        <h2>{name}</h2>
        {address &&
            <p>{address}</p>
        }
    </div>
)
```

If you need an **if-else** condition then use ***ternary operator***.

```jsx
const MyComponent = ({ name, address }) => (
    <div>
        <h2>{name}</h2>
        {address
            ? <p>{address}</p>
            : <p>{'Address is not available'}</p>
        }
    </div>
)
```

## 68. Why we need to be careful when spreading props on DOM elements?

When we ***spread props*** we run into the risk of adding unknown HTML attributes, which is a bad practice. Instead we can use prop destructuring with **...rest operator**, so it will add only required props.

**For example,**

```jsx
const ComponentA = () =>
    <ComponentB isDisplay={true} className={'componentStyle'} /
>
```

```
const ComponentB = ({ isDisplay, ...domProps }) =>
    <div {...domProps}>{'ComponentB'}</div>
```

### 69.     How you use decorators in React?

You can *decorate* your *class* components, which is the same as passing the component into a function. **Decorators** are flexible and readable way of modifying component functionality.

```
@setTitle('Profile')
class Profile extends React.Component {
    //....
}
```

```
/*
  title is a string that will be set as a document title
  WrappedComponent is what our decorator will receive when
  put directly above a component class as seen in the example a
bove
*/
const setTitle = (title) => (WrappedComponent) => {
    return class extends React.Component {
        componentDidMount() {
            document.title = title
        }

        render() {
            return <WrappedComponent {...this.props} />
        }
    }
}
```

**Note:** Decorators are a feature that didn't make it into ES7, but are currently a *stage 2 proposal.*

## 70.     How do you memoize a component?

There are memoize libraries available which can be used on function components.

For example **moize** library can memoize the component in another component.

```
import moize from 'moize'
import Component from './components/Component'
// this module exports a non-memoized component

const MemoizedFoo = moize.react(Component)

const Consumer = () => {
    <div>
        {'I will memoize the following entry:'}
        <MemoizedFoo />
    </div>
}
```

**Update:** Since React v16.6.0, we have a `React.memo`. It provides a higher order component which memoizes component unless the props change. To use it, simply wrap the component using React.memo before you use it.

```
const MemoComponent = React.memo(function MemoComponent(props)
{
    /* render using props */
});
OR
export default React.memo(MyFunctionComponent);
```

## 71.     How you implement Server Side Rendering or SSR?

React is already equipped to handle rendering on Node servers. A special version of the DOM renderer is available, which follows the same pattern as on the client side.

```
import ReactDOMServer from 'react-dom/server'
import App from './App'

ReactDOMServer.renderToString(<App />)
```

This method will output the regular HTML as a string, which can be then placed inside a page body as part of the server response. On the client side, React detects the pre-rendered content and seamlessly picks up where it left off.

## 72.     How to enable production mode in React?

You should use Webpack's `DefinePlugin` method to set `NODE_ENV` to `production`, by which it strip out things like propType validation and extra warnings. Apart from this, if you minify the code, for example, Uglify's dead-code elimination to strip out development only code and comments, it will drastically reduce the size of your bundle.

## 73.     What is CRA and its benefits?

The `create-react-app` CLI tool allows you to quickly create & run React applications with no configuration step.

Let's create Todo App using *CRA*:

```
# Installation
$ npm install -g create-react-app

# Create new project
$ create-react-app todo-app
$ cd todo-app

# Build, test and run
$ npm run build
$ npm run test
$ npm start
```

It includes everything we need to build a React app:

i.    React, JSX, ES6, and Flow syntax support.
ii.   Language extras beyond ES6 like the object spread operator.
iii.  Autoprefixed CSS, so you don't need -webkit- or other prefixes.
iv.   A fast interactive unit test runner with built-in support for coverage reporting.

v.   A live development server that warns about common mistakes.

vi.  A build script to bundle JS, CSS, and images for production, with hashes and sourcemaps.

## 74.   What is the lifecycle methods order in mounting?

The lifecycle methods are called in the following order when an instance of a component is being created and inserted into the DOM.

i.   `constructor()`
ii.  `static getDerivedStateFromProps()`
iii. `render()`
iv.  `componentDidMount()`

## 75.   What are the lifecycle methods going to be deprecated in React v16?

The following lifecycle methods going to be unsafe coding practices and will be more problematic with async rendering.

i.   `componentWillMount()`
ii.  `componentWillReceiveProps()`
iii. `componentWillUpdate()`

Starting with React v16.3 these methods are aliased with `UNSAFE_` prefix, and the unprefixed version will be removed in React v17.

## 76.   What is the purpose of `getDerivedStateFromProps()` lifecycle method?
The new static **getDerivedStateFromProps()** lifecycle method is invoked after a component is instantiated as well as before it is re-rendered. It can return an object to update state, or `null` to indicate that the new props do not require any state updates.

```
class MyComponent extends React.Component {
    static getDerivedStateFromProps(props, state) {
      // ...
    }
```

```
}
```

This lifecycle method along with **componentDidUpdate()** covers all the use cases of **componentWillReceiveProps()**.

### 77.     What is the purpose of getSnapshotBeforeUpdate() **lifecycle method?**

The new **getSnapshotBeforeUpdate()** lifecycle method is called right before DOM updates. The return value from this method will be passed as the third parameter to **componentDidUpdate()**.

```
class MyComponent extends React.Component {
    getSnapshotBeforeUpdate(prevProps, prevState) {
        // ...
    }
}
```

This lifecycle method along with **componentDidUpdate()** covers all the use cases of **componentWillUpdate()**.

### 78.     Do Hooks replace render props and higher order components?

Both render props and higher-order components render only a single child but in most of the cases Hooks are a simpler way to serve this by reducing nesting in your tree.

### 79.     What is the recommended way for naming components?

It is recommended to name the component by reference instead of using displayName.
Using displayName for naming component:

```
export default React.createClass({
    displayName: 'TodoApp',
    // ...
})
```

The **recommended** approach:

```
export default class TodoApp extends React.Component {
```

```
    // ...
}
```

## 80. What is the recommended ordering of methods in component class?

*Recommended* ordering of methods from *mounting* to *render stage*:

   i.    `static methods`
   ii.   `constructor()`
  iii.  `getChildContext()`
  iv.  `componentWillMount()`
   v.   `componentDidMount()`
  vi.  `componentWillReceiveProps()`
 vii.  `shouldComponentUpdate()`
viii.  `componentWillUpdate()`
  ix.  `componentDidUpdate()`
   x.   `componentWillUnmount()`
  xi.  click handlers or event handlers
      like `onClickSubmit()` or `onChangeDescription()`
 xii.  getter methods for render
      like `getSelectReason()` or `getFooterContent()`
xiii.  optional render methods
      like `renderNavigation()` or `renderProfilePicture()`
xiv.  `render()`

## 81. What is a switching component?

A *switching component* is a component that renders one of many components. We need to use object to map prop values to components.

For example, a switching component to display different pages based on `page` prop:

```
import HomePage from './HomePage'
import AboutPage from './AboutPage'
import ServicesPage from './ServicesPage'
import ContactPage from './ContactPage'
```

```
const PAGES = {
    home: HomePage,
    about: AboutPage,
    services: ServicesPage,
    contact: ContactPage
}

const Page = (props) => {
    const Handler = PAGES[props.page] || ContactPage

    return <Handler {...props} />
}

// The keys of the PAGES object can be used in the prop types t
o catch dev-time errors.
Page.propTypes = {
 page: PropTypes.oneOf(Object.keys(PAGES)).isRequired
}
```

## 82.    Why we need to pass a function to setState()?

The reason behind for this is that **setState()** is an asynchronous operation. React batches state changes for performance reasons, so the state may not change immediately after setState() is called. That means you should not rely on the current state when calling setState() since you can't be sure what that state will be. The solution is to pass a function to setState(), with the previous state as an argument. By doing this you can avoid issues with the user getting the old state value on access due to the asynchronous nature of setState().

Let's say the initial count value is zero. After three consecutive increment operations, the value is going to be incremented only by one.

```
// assuming this.state.count === 0
this.setState({ count: this.state.count + 1 })
this.setState({ count: this.state.count + 1 })
this.setState({ count: this.state.count + 1 })
// this.state.count === 1, not 3
```

If we pass a function to setState(), the count gets incremented correctly.

```
this.setState((prevState, props) => ({
    count: prevState.count + props.increment
}));
  // this.state.count === 3 as expected
```

**(OR)**

## Why function is preferred over object for setState()?

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.

This counter example will fail to update as expected:

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

The preferred approach is to call `setState()` with function rather than object. That function will receive the previous state as the first argument, and the props at the time the update is applied as the second argument.

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

## 83.    What is strict mode in React?

`React.StrictMode` is a useful component for highlighting potential problems in an application. Just like `<Fragment>`, `<StrictMode>` does not render any extra DOM elements. It activates additional checks and warnings for its descendants. These checks apply for *development mode* only.

```
import React from 'react'

function ExampleApplication() {
    return (
        <div>
            <Header />
```

```
            <React.StrictMode>
                <div>
                    <ComponentOne />
                    <ComponentTwo />
                </div>
            </React.StrictMode>
            <Footer />
        </div>
    )
}
```

In the example above, the *strict mode* checks apply
to **<ComponentOne>** and **<ComponentTwo>** components only.

## 84.     What are React Mixins?

*Mixins* are a way to totally separate components to have a common
functionality. Mixins **should not be used** and can be replaced with *higher-
order components* or *decorators*.

One of the most commonly used mixins is PureRenderMixin. You might be
using it in some components to prevent unnecessary re-renders when the
props and state are shallowly equal to the previous props and state:

```
const PureRenderMixin = require('react-addons-pure-render-mixin')

const Button = React.createClass({
    mixins: [PureRenderMixin],
    // ...
});
```

## 85.     Why is isMounted() an anti-pattern and what is the proper solution?

The primary use case for isMounted() is to avoid calling setState() after a
component has been unmounted, because it will emit a warning.

```
if (this.isMounted()) {
    this.setState({ ...})
}
```

Checking **isMounted()** before calling **setState()** does eliminate the warning, but it also defeats the purpose of the warning. Using **isMounted()** is a code smell because the only reason you would check is because you think you might be holding a reference after the component has unmounted.

An optimal solution would be to find places where **setState()** might be called after a component has unmounted, and fix them. Such situations most commonly occur due to callbacks, when a component is waiting for some data and gets unmounted before the data arrives. Ideally, any callbacks should be canceled in **componentWillUnmount(),** prior to unmounting.

## 86.    What are the Pointer Events supported in React?

*Pointer Events* provide a unified way of handling all input events. In the old days we had a mouse and respective event listeners to handle them but nowadays we have many devices which don't correlate to having a mouse, like phones with touch surface or pens. We need to remember that these events will only work in browsers that support the *Pointer Events* specification.

The following event types are now available in *React DOM*:

   i.   onPointerDown
  ii.   onPointerMove
 iii.   onPointerUp
 iv.   onPointerCancel
  v.   onGotPointerCapture
 vi.   onLostPointerCapture
 vii.   onPointerEnter
viii.   onPointerLeave
 ix.   onPointerOver
  x.   onPointerOut

## 87.    Why should component names start with capital letter?

If you are rendering your component using JSX, the name of that component has to begin with a capital letter otherwise React will throw an error as

unrecognized tag. This convention is because only HTML elements and SVG tags can begin with a lowercase letter.

```
class SomeComponent extends Component {
    // Code goes here
}
```

You can define component class which name starts with lowercase letter, but when it's imported it should have capital letter. Here lowercase is fine:

```
class myComponent extends Component {
    render() {
        return <div />
    }
}

export default myComponent
```

While when imported in another file it should start with capital letter:

```
import MyComponent from './MyComponent'
```

## 88.    Are custom DOM attributes supported in React v16?

Yes. In the past, React used to ignore unknown DOM attributes. If you wrote JSX with an attribute that React doesn't recognize, React would just skip it.

For example, let's take a look at the below attribute:

```
<div mycustomattribute={'something'} />
```

Would render an empty div to the DOM with React v15:

```
<div />
```

In React v16 any unknown attributes will end up in the DOM:

```
<div mycustomattribute='something' />
```

This is useful for supplying browser-specific non-standard attributes, trying new DOM APIs, and integrating with opinionated third-party libraries.

### 89. What is the difference between constructor and getInitialState?

You should initialize state in the constructor when using ES6 classes, and **getInitialState()** method when using **React.createClass()**.

**Using ES6 classes:**

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props)
        this.state = { /* initial state */ }
    }
}
```

**Using React.createClass():**

```
const MyComponent = React.createClass({
    getInitialState() {
        return { /* initial state */ }
    }
})
```

**Note:** React.createClass() is deprecated and removed in React v16. Use plain JavaScript classes instead.

### 90. Can you force a component to re-render without calling setState?

By default, when your component's state or props change, your component will re-render. If your render() method depends on some other data, you can tell React that the component needs re-rendering by calling forceUpdate(). component.forceUpdate(callback)

It is recommended to avoid all uses of forceUpdate() and only read from this.props and this.state in render().

### 91. What is the difference between super() and super(props) in React using ES6 classes?

When you want to access this.props in constructor() then you should pass props to super() method.

**Using `super(props)`:**

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props)
        console.log(this.props) // { name: 'John', ... }
    }
}
```

**Using `super()`:**

```
class MyComponent extends React.Component {
    constructor(props) {
        super()
        console.log(this.props) // undefined
    }
}
```

Outside `constructor()` both will display same value for `this.props`.

## 92.    How to loop inside JSX?

You can simply use `Array.prototype.map` with ES6 *arrow function* syntax. For example, the `items` array of objects is mapped into an array of components:

```
<tbody>
    {items.map(item => <SomeComponent key={item.id}
      name={item.name} />)}
</tbody>
```

But you can't iterate using `for` loop:

```
<tbody>
    for (let i = 0; i < items.length) {
        <SomeComponent key={items[i].id}
          name={items[i].name} />
    }
</tbody>
```

This is because JSX tags are transpiled into *function calls*, and you can't use statements inside expressions. This may change thanks to do expressions which are *stage 1 proposal*.

## 93.  How do you access props in attribute quotes?

React (or JSX) doesn't support variable interpolation inside an attribute value. The below representation won't work:

```
<img className='image' src='images/{this.props.image}' />
```

But you can put any JS expression inside curly braces as the entire attribute value. So the below expression works:

```
<img className='image' src={'images/' + this.props.image} />
```

Using *template strings* will also work:

```
<img className='image' src={`images/${this.props.image}`} />
```

## 94.  What is React proptype array with shape?

If you want to pass an array of objects to a component with a particular shape then use **React.PropTypes.shape()** as an argument to **React.PropTypes.arrayOf().**

```
ReactComponent.propTypes = {
    arrayWithShape: React.PropTypes.arrayOf(React.PropTypes.sha
pe({
        color: React.PropTypes.string.isRequired,
        fontSize: React.PropTypes.number.isRequired
    })).isRequired
}
```

## 95.  How to conditionally apply class attributes?

You shouldn't use curly braces inside quotes because it is going to be evaluated as a string.

```
<div className="btn-
panel {this.props.visible ? 'show' : 'hidden'}">
```

Instead you need to move curly braces outside (don't forget to include spaces between class names):

```
<div className={'btn-
panel ' + (this.props.visible ? 'show' : 'hidden')}>
```

*Template strings* will also work:

```
<div className={'btn-
panel ' + (this.props.visible ? 'show' : 'hidden')}>
```

## 96.    What is the difference between React and ReactDOM?

The react package
contains **React.createElement()**, **React.Component**, **React.Children**, and other helpers related to elements and component classes. You can think of these as the isomorphic or universal helpers that you need to build components.
The **react-dom** package contains **ReactDOM.render()**, and in **react-dom/server** we have *server-side rendering* support
with **ReactDOMServer.renderToString()** and **ReactDOMServer.renderToStaticMarkup()**.

## 97.    Why ReactDOM is separated from React?

The React team worked on extracting all DOM-related features into a separate library called *ReactDOM*. React v0.14 is the first release in which the libraries are split. By looking at some of the **packages**, **react-native**, **react-art**, **react-canvas**, and **react-three**, it has become clear that the beauty and essence of React has nothing to do with browsers or the DOM.
To build more environments that React can render to, React team planned to split the main React package into two: react and react-dom. This paves the way to writing components that can be shared between the web version of React and React Native.

## 98.    How to use React label element?

If you try to render a **`<label>`** element bound to a text input using the standard `for` attribute, then it produces HTML missing that attribute and prints a warning to the console.

```
<label for={'user'}>{'User'}</label>
<input type={'text'} id={'user'} />
```

Since `for` is a reserved keyword in JavaScript, use `htmlFor` instead.

```
<label htmlFor={'user'}>{'User'}</label>
<input type={'text'} id={'user'} />
```

## 99.     How to combine multiple inline style objects?

You can use *spread operator* in regular React:

```
<button style={{ ...styles.panel.button, ...styles.panel.submit
Button }}>{'Submit'}</button>
```

If you're using React Native then you can use the array notation:

```
<button style={[styles.panel.button, styles.panel.submitButton]
}>{'Submit'}</button>
```

## 100.     How to re-render the view when the browser is resized?

You can listen to the `resize` event in `componentDidMount()` and then update the dimensions (`width` and `height`). You should remove the listener in `componentWillUnmount()` method.

```
class WindowDimensions extends React.Component {
    constructor(props) {
        super(props);
        this.updateDimensions = this.updateDimensions.bind(this
);
    }

    componentWillMount() {
        this.updateDimensions()
    }
```

```
    componentDidMount() {
        window.addEventListener('resize', this.updateDimensions
)
    }

    componentWillUnmount() {
        window.removeEventListener('resize', this.updateDimensi
ons)
    }

    updateDimensions() {
        this.setState({ width: window.innerWidth, height: windo
w.innerHeight })
    }

    render() {
        return <span>{this.state.width} x {this.state.height}</
span>
    }
}
```

## 101.    What is the difference

**between** `setState()` **and** `replaceState()` **methods?**
When you use **setState()** the current and previous states are
merged. **replaceState()** throws out the current state, and replaces it with
only what you provide. Usually **setState()** is used unless you really need to
remove all previous keys for some reason. You can also set state
to **false/null** in **setState()** instead of using **replaceState()**.

## 102.    How to listen to state changes?

The `componentDidUpdate` lifecycle method will be called when state changes.
You can compare provided state and props values with current state and props
to determine if something meaningful changed.
**componentDidUpdate(object prevProps, object prevState)**
**Note:** The previous releases of ReactJS also
uses **componentWillUpdate(object nextProps, object nextState)** for
state changes. It has been deprecated in latest releases.

### 103. What is the recommended approach of removing an array element in React state?

The better approach is to use **Array.prototype.filter()** method.
For example, let's create a **removeItem()** method for updating the state.

```
removeItem(index) {
    this.setState({
        data: this.state.data.filter((item, i) => i !== index)
    })
}
```

### 104. Is it possible to use React without rendering HTML?

It is possible with latest version (>=16.2). Below are the possible options:

```
render() {
    return false
}
render() {
    return null
}
render() {
    return []
}
render() {
    return <React.Fragment></React.Fragment>
}
render() {
    return <></>
}
```

Returning undefined won't work.

### 105. How to pretty print JSON with React?

We can use **<pre>** tag so that the formatting of the **JSON.stringify()** is retained:

```
const data = { name: 'John', age: 42 }

class User extends React.Component {
    render() {
        return (
            <pre>
                {JSON.stringify(data, null, 2)}
            </pre>
        )
    }
}

React.render(<User />, document.getElementById('container'))
```

## 106.    Why you can't update props in React?

The React philosophy is that props should be *immutable* and *top-down*. This
means that a parent can send any prop values to a child, but the child can't
modify received props.

## 107.    How to focus an input element on page load?

You can do it by creating *ref* for `input` element and using it
in `componentDidMount()`:

```
class App extends React.Component {
    componentDidMount() {
        this.nameInput.focus()
    }

    render() {
        return (
            <div>
                <input
                    defaultValue={'Won\'t focus'}
                />
                <input
                    ref={(input) => this.nameInput = input}
                    defaultValue={'Will focus'}
                />
```

```
            </div>
        )
    }
}

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

## 108.    What are the possible ways of updating objects in state?

**Calling setState() with an object to merge with state:**
Using Object.assign() to create a copy of the object:

```
const user = Object.assign({}, this.state.user, { age: 4
2 })
this.setState({ user })
```

Using *spread operator*:

```
const user = { ...this.state.user, age: 42 }
this.setState({ user })
```

**Calling setState() with a function:**

```
this.setState(prevState => ({
    user: {
        ...prevState.user,
        age: 42
    }
}));
```

## 109.    Why function is preferred over object for setState()?

React may batch multiple setState() calls into a single update for performance. Because this.props and this.statemay be updated asynchronously, you should not rely on their values for calculating the next state. This counter example will fail to update as expected:

```
// Wrong
```

```
this.setState({ counter: this.state.counter + this.props.increm
ent, })
```

The preferred approach is to call setState() with function rather than object. That function will receive the previous state as the first argument, and the props at the time the update is applied as the second argument.

```
// Correct
this.setState((prevState, props) => ({ counter: prevState.count
er + props.increment }))
```

## 110. How can we find the version of React at runtime in the browser?

You can use React.version to get the version.
```
const REACT_VERSION = React.version

ReactDOM.render(
  <div>{`React version: ${REACT_VERSION}`}</div>,
  document.getElementById('app')
)
```

## 111. What are the approaches to include polyfills in your create-react-app?

There are approaches to include polyfills in create-react-app,

   i. **Manual import from core-js:**
Create a file called (something like) polyfills.js and import it into root index.js file. Run npm install core-js or yarn add core-js and import your specific required features.

```
import 'core-js/fn/array/find'
import 'core-js/fn/array/includes'
import 'core-js/fn/number/is-nan'
```

   ii. **Using Polyfill service:**

Use the polyfill.io CDN to retrieve custom, browser-specific polyfills by adding this line to index.html:

```
<script src='https://cdn.polyfill.io/v2/polyfill.min.js?fe
atures=default,Array.prototype.includes'></script>
```

In the above script we had to explicitly request
the **Array.prototype.includes** feature as it is not included in the
default feature set.

## 112.    How to use https instead of http in create-react-app?

You just need to use HTTPS=true configuration. You can edit
your package.json scripts section:

```
"scripts": {
    "start": "set HTTPS=true && react-scripts start"
  }
```

or just run set HTTPS=true && npm start

## 113.    How to avoid using relative path imports in create-react-app?

Create a file called .env in the project root and write the import path:
**NODE_PATH=src/app**
After that restart the development server. Now you should be able to import
anything inside src/app without relative paths.

## 114.    How to add Google Analytics for React Router?

Add a listener on the history object to record each page view:

```
history.listen(function (location) {
    window.ga('set', 'page', location.pathname + location.searc
h)
    window.ga('send', 'pageview', location.pathname + location.
search)
})
```

## 115.    How to update a component every second?

You need to use `setInterval()` to trigger the change, but you also need to clear the timer when the component unmounts to prevent errors and memory leaks.

```
componentDidMount() {
    this.interval = setInterval(() => this.setState({ time: Dat
e.now() }), 1000)
}

componentWillUnmount() {
    clearInterval(this.interval)
}
```

## 116. How do you apply vendor prefixes to inline styles in React?

React *does not* apply *vendor prefixes* automatically. You need to add vendor prefixes manually.

```
<div style={{
    transform: 'rotate(90deg)',
    WebkitTransform: 'rotate(90deg)', // note the capital 'W' h
ere
    msTransform: 'rotate(90deg)' // 'ms' is the only lowercase
vendor prefix
}} />
```

## 117. How to import and export components using React and ES6?

You should use default for exporting the components

```
import React from 'react'
import User from 'user'

export default class MyProfile extends React.Component {
    render() {
        return (
            <User type="customer">
                    //...
            </User>
        )
    }
```

```
}
```

With the export specifier, the MyProfile is going to be the member and exported to this module and the same can be imported without mentioning the name in other components.

## 118.   Why React component names must begin with a capital letter?

In JSX, lowercase tag names are considered to be HTML tags. However, capitalized and lowercase tag names with a dot (property accessors) aren't. . **<component />** compiles to **React.createElement('component')** (i.e, HTML tag)

I.    **<obj.component />** compiles to **React.createElement(obj.component)**

ii. **<Component />** compiles to **React.createElement(Component)**

## 119.   Why is a component constructor called only once?

React's *reconciliation* algorithm assumes that without any information to the contrary, if a custom component appears in the same place on subsequent renders, it's the same component as before, so reuses the previous instance rather than creating a new one.

## 120.   How to define constants in React?

You can use ES7 `static` field to define constant.

```
class MyComponent extends React.Component {
    static DEFAULT_PAGINATION = 10
}
```

*Static fields* are part of the *Class Fields* stage 3 proposal.

## 121.   How to programmatically trigger click event in React?

You could use the ref prop to acquire a reference to the underlying `HTMLInputElement` object through a callback, store the reference as a class property, then use that reference to later trigger a click from your event handlers using the `HTMLElement.click` method.

This can be done in two steps:

    i.    Create ref in render method:

```
<input ref={input => this.inputElement = input} />
```

    ii.    Apply click event in your event handler:

```
this.inputElement.click()
```

## 122.    Is it possible to use async/await in plain React?

If you want to use `async/await` in React, you will need *Babel* and **transform-async-to-generator** plugin. React Native ships with Babel and a set of transforms.

## 123.    What are the common folder structures for React?

There are two common practices for React project file structure.

    i.    **Grouping by features or routes:**

One common way to structure projects is locate CSS, JS, and tests together, grouped by feature or route.

```
common/
├── Avatar.js
├── Avatar.css
├── APIUtils.js
└── APIUtils.test.js
feed/
├── index.js
├── Feed.js
├── Feed.css
├── FeedStory.js
├── FeedStory.test.js
└── FeedAPI.js
```

```
profile/
├── index.js
├── Profile.js
├── ProfileHeader.js
├── ProfileHeader.css
└── ProfileAPI.js
```

ii. **Grouping by file type:**

Another popular way to structure projects is to group similar files together.

```
api/
├── APIUtils.js
├── APIUtils.test.js
├── ProfileAPI.js
└── UserAPI.js
components/
├── Avatar.js
├── Avatar.css
├── Feed.js
├── Feed.css
├── FeedStory.js
├── FeedStory.test.js
├── Profile.js
├── ProfileHeader.js
└── ProfileHeader.css
```

## 124. What are the popular packages for animation?

*React Transition Group* and *React Motion* are popular animation packages in React ecosystem.

## 125. What is the benefit of styles modules?

It is recommended to avoid hard coding style values in components. Any values that are likely to be used across different UI components should be extracted into their own modules.

For example, these styles could be extracted into a separate component:

```
export const colors = {
    white,
    black,
    blue
}

export const space = [
    0,
    8,
    16,
    32,
    64
]
```

And then imported individually in other components:

```
import { space, colors } from './styles'
```

## 126.  What are the popular React-specific linters?

ESLint is a popular JavaScript linter. There are plugins available that analyse specific code styles. One of the most common for React is an npm package called **eslint-plugin-react**. By default, it will check a number of best practices, with rules checking things from keys in iterators to a complete set of prop types.
Another popular plugin is **eslint-plugin-jsx-a11y**, which will help fix common issues with accessibility. As JSX offers slightly different syntax to regular HTML, issues with `alt` text and `tabindex`, for example, will not be picked up by regular plugins.

## 127.  How to make AJAX call and in which component lifecycle methods should I make an AJAX call?

You can use **AJAX** libraries such as **Axios**, **jQuery AJAX**, and the browser built-in **fetch**. You should fetch data in the **componentDidMount()** lifecycle method. This is so you can use **setState()** to update your component when the data is retrieved.

For example, the employees list fetched from API and set local state:

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

    componentDidMount() {
        fetch('https://api.example.com/items')
            .then(res => res.json())
            .then(
                (result) => {
                    this.setState({
                        employees: result.employees
                    })
                },
                (error) => {
                    this.setState({ error })
                }
            )
    }

    render() {
        const { error, employees } = this.state
        if (error) {
            return <div>Error: {error.message}</div>;
        } else {
            return (
                <ul>
                    {employees.map(employee => (
                        <li key={employee.name}>
                            {employee.name}-
{employee.experience}
                        </li>
                    ))}
                </ul>
            )
        }
    }
}
```

## 128.    What are render props?

**Render Props** is a simple technique for sharing code between components using a prop whose value is a function. The below component uses render prop which returns a React element.

```
<DataProvider render={data => (
    <h1>{`Hello ${data.target}`}</h1>
)} />
```

Libraries such as React Router and DownShift are using this pattern.

## 129.    What is React Router?

React Router is a powerful routing library built on top of React that helps you add new screens and flows to your application incredibly quickly, all while keeping the URL in sync with what's being displayed on the page.

## 130.    How React Router is different from history library?

React Router is a wrapper around the `history` library which handles interaction with the browser's `window.history` with its browser and hash histories. It also provides memory history which is useful for environments that don't have global history, such as mobile app development (**React Native**) and unit testing with Node.

## 131.    What are the `<Router>` components of React Router v4?
React Router v4 provides below 3 <Router> components:

  i.   <BrowserRouter>
  ii.  <HashRouter>
  iii. <MemoryRouter>

The above components will create *browser*, *hash*, and *memory* history instances. React Router v4 makes the properties and methods of the `history` instance associated with your router available through the context in the `router` object.

## 132.    What is the purpose of `push()` and `replace()` methods of `history`?

A history instance has two methods for navigation purpose.

   i.   `push()`
  ii.   `replace()`

If you think of the history as an array of visited locations, `push()` will add a new location to the array and `replace()` will replace the current location in the array with the new one.

## 133. How do you programmatically navigate using React Router v4?

There are three different ways to achieve programmatic routing/navigation within components.

   i.   **Using the `withRouter()` higher-order function:**
The `withRouter()` higher-order function will inject the history object as a prop of the component. This object
provides `push()` and `replace()` methods to avoid the usage of context.

```
import { withRouter } from 'react-router-dom'
// this also works with 'react-router-native'

const Button = withRouter(({ history }) => (
    <button
        type='button'
        onClick={() => { history.push('/new-location') }}
    >
        {'Click Me!'}
    </button>
))
```

  ii.   **Using `<Route>` component and render props pattern:**
The **`<Route>`** component passes the same props as **`withRouter()`,** so you will be able to access the history methods through the history prop.

```
import { Route } from 'react-router-dom'

const Button = () => (
    <Route render={({ history }) => (
        <button
            type='button'
```

```
            onClick={() => { history.push('/new-
location') }}
        >
            {'Click Me!'}
        </button>
    )} />
)
```

iii. **Using context:**

This option is not recommended and treated as unstable API.

```
const Button = (props, context) => (
    <button
        type='button'
        onClick={() => {
            context.history.push('/new-location')
        }}
    >
        {'Click Me!'}
    </button>
)

Button.contextTypes = {
    history: React.PropTypes.shape({
        push: React.PropTypes.func.isRequired
    })
}
```

## 134.     How to get query parameters in React Router v4?

The ability to parse query strings was taken out of React Router v4 because there have been user requests over the years to support different implementation. So the decision has been given to users to choose the implementation they like. The recommended approach is to use query strings library.

```
const queryString = require('query-string');
const parsed = queryString.parse(props.location.search);
```

You can also use URLSearchParams if you want something native:

```
const params = new URLSearchParams(props.location.search)
```

```
const foo = params.get('name')
```

You should use a *polyfill* for IE11.

## 135. Why you get "Router may have only one child element" warning?

You have to wrap your Route's in a **<Switch>** block because **<Switch>** is unique in that it renders a route exclusively.
At first you need to add Switch to your imports:

```
import { Switch, Router, Route } from 'react-router'
```

Then define the routes within <Switch> block:

```
<Router>
    <Switch>
        <Route {/* ... */} />
        <Route {/* ... */} />
    </Switch>
</Router>
```

## 136. How to pass params to `history.push` method in React Router v4?
While navigating you can pass props to the `history` object:

```
this.props.history.push({
    pathname: '/template',
    search: '?name=sudheer',
    state: { detail: response.data }
})
```

The `search` property is used to pass query params in **push()** method.

## 137. How to implement *default* or *NotFound* page?

A **<Switch>** renders the first child **<Route>** that matches. A **<Route>** with no path always matches. So you just need to simply drop path attribute as below

```
<Switch>
    <Route exact path="/" component={Home} />
```

```
    <Route path="/user" component={User} />
    <Route component={NotFound} />
</Switch>
```

## 138.    How to get history on React Router v4?

Below are the list of steps to get history object on React Router v4,

Create a module that exports a `history` object and import this module across the project.
For example, create `history.js` file:

```
import { createBrowserHistory } from 'history'

export default createBrowserHistory({
    /* pass a configuration object here if needed */
})
```

You should use the `<Router>` component instead of built-in routers. Imported the above `history.js` inside `index.js` file:

```
import { Router } from 'react-router-dom'
import history from './history'
import App from './App'

ReactDOM.render((
    <Router history={history}>
        <App />
    </Router>
), holder)
```

You can also use push method of `history` object similar to built-in history object:

```
// some-other-file.js
import history from './history'

history.push('/go-here')
```

## 139.    How to perform automatic redirect after login?

The **react-router** package provides **&lt;Redirect&gt;** component in React Router. Rendering a **&lt;Redirect&gt;** will navigate to a new location. Like server-side redirects, the new location will override the current location in the history stack.

```
import React, { Component } from 'react'
import { Redirect } from 'react-router'

export default class LoginComponent extends Component {
    render() {
        if (this.state.isLoggedIn === true) {
            return <Redirect to="/your/redirect/page" />
        } else {
            return <div>{'Login Please'}</div>
        }
    }
}
```

## 140. What is React Intl?

The **React Intl** library makes internalization in React straightforward, with off-the-shelf components and an API that can handle everything from formatting strings, dates, and numbers, to pluralization. React Intl is part of *FormatJS* which provides bindings to React via its components and API.

## 141. What are the main features of React Intl?

Below are the main features of React Intl,

   i.     Display numbers with separators.
   ii.    Display dates and times correctly.
   iii.   Display dates relative to "now".
   iv.   Pluralize labels in strings.
   v.    Support for 150+ languages.
   vi.   Runs in the browser and Node.
   vii.  Built on standards.

## 142. What are the two ways of formatting in React Intl?

The library provides two ways to format strings, numbers, and dates:

i. **Using react components:**

```
<FormattedMessage
    id={'account'}
    defaultMessage={'The amount is less than minimum balance
.'}
/>
```

ii. **Using an API:**

```
const messages = defineMessages({
    accountMessage: {
        id: 'account',
        defaultMessage: 'The amount is less than minimum bal
ance.',
    }
})

formatMessage(messages.accountMessage)
```

## 143. How to use `<FormattedMessage>` as placeholder using React Intl?

The **`<Formatted... />`** components from **react-intl** return elements, not plain text, so they can't be used for placeholders, alt text, etc. In that case, you should use lower level API **formatMessage()**. You can inject the `intl` object into your component using **injectIntl()** higher-order component and then format the message using `formatMessage()` available on that object.

```
import React from 'react'
import { injectIntl, intlShape } from 'react-intl'

const MyComponent = ({ intl }) => {
    const placeholder = intl.formatMessage({ id: 'messageId' })
    return <input placeholder={placeholder} />
}

MyComponent.propTypes = {
    intl: intlShape.isRequired
```

```
}
export default injectIntl(MyComponent)
```

## 144.    How to access current locale with React Intl?

You can get the current locale in any component of your application using **injectIntl()**:

```
import { injectIntl, intlShape } from 'react-intl'

const MyComponent = ({ intl }) => (
   <div>{`The current locale is ${intl.locale}`}</div>
)

MyComponent.propTypes = {
    intl: intlShape.isRequired
}

export default injectIntl(MyComponent)
```

## 145.    How to format date using React Intl?

The **injectIntl()** higher-order component will give you access to the **formatDate()** method via the props in your component. The method is used internally by instances of FormattedDate and it returns the string representation of the formatted date.

```
import { injectIntl, intlShape } from 'react-intl'

const stringDate = this.props.intl.formatDate(date, {
    year: 'numeric',
    month: 'numeric',
    day: 'numeric'
})

const MyComponent = ({ intl }) => (
    <div>{`The formatted date is ${stringDate}`}</div>
)
```

```
MyComponent.propTypes = {
    intl: intlShape.isRequired
}

export default injectIntl(MyComponent)
```

## 146.    What are Styled Components?

**styled-components** is a JavaScript library for styling React applications. It removes the mapping between styles and components, and lets you write actual CSS augmented with JavaScript.

## 147.    Give an example of Styled Components?

Lets create **<Title>** and **<Wrapper>** components with specific styles for each.

```
import React from 'react'
import styled from 'styled-components'

// Create a <Title> component that renders an <h1> which is cen
tered, red and sized at 1.5em
const Title = styled.h1`
  font-size: 1.5em;
  text-align: center;
  color: palevioletred;
`

// Create a <Wrapper> component that renders a <section> with s
ome padding and a papayawhip background
const Wrapper = styled.section`
  padding: 4em;
  background: papayawhip;
`
```

These two variables, `Title` and `Wrapper`, are now components that you can render just like any other react component.

```
<Wrapper>
<Title>{'Lets start first styled component!'}</Title>
</Wrapper>
```

## 148. What is Relay?

Relay is a JavaScript framework for providing a data layer and client-server communication to web applications using the React view layer.

## 149. How to use TypeScript in `create-react-app` application?

Starting from **react-scripts@2.1.0** or higher, there is a built-in support for typescript. i.e, **create-react-app** now supports typescript natively. You can just pass --typescript option as below
```
npx create-react-app my-app --typescript
```

# or

```
yarn create react-app my-app --typescript
```

But for lower versions of react scripts, just supply **--scripts-version** option as **react-scripts-ts** while you create a new project. **react-scripts-ts** is a set of adjustments to take the standard `create-react-app` project pipeline and bring TypeScript into the mix.

Now the project layout should look like the following:

```
my-app/
├─ .gitignore
├─ images.d.ts
├─ node_modules/
├─ public/
├─ src/
│  └─ ...
├─ package.json
├─ tsconfig.json
├─ tsconfig.prod.json
├─ tsconfig.test.json
└─ tslint.json
```

## 150. What are the main features of Reselect library?

Let's see the main features of Reselect library,

i.   Selectors can compute derived data, allowing Redux to store the minimal possible state.

ii.  Selectors are efficient. A selector is not recomputed unless one of its arguments changes.

iii. Selectors are composable. They can be used as input to other selectors.

## 151.   Give an example of Reselect usage?

Let's take calculations and different amounts of a shipment order with the simplified usage of Reselect:

```javascript
import { createSelector } from 'reselect'

const shopItemsSelector = state => state.shop.items
const taxPercentSelector = state => state.shop.taxPercent

const subtotalSelector = createSelector(
    shopItemsSelector,
    items => items.reduce((acc, item) => acc + item.value, 0)
)

const taxSelector = createSelector(
    subtotalSelector,
    taxPercentSelector,
    (subtotal, taxPercent) => subtotal * (taxPercent / 100)
)

export const totalSelector = createSelector(
    subtotalSelector,
    taxSelector,
    (subtotal, tax) => ({ total: subtotal + tax })
)

let exampleState = {
    shop: {
        taxPercent: 8,
        items: [
            { name: 'apple', value: 1.20 },
            { name: 'orange', value: 0.95 },
        ]
    }
}
```

```
console.log(subtotalSelector(exampleState)) // 2.15
console.log(taxSelector(exampleState))      // 0.172
console.log(totalSelector(exampleState))    // { total: 2.322 }
```

## 152. Does the statics object work with ES6 classes in React?

No, statics only works with **React.createClass()**:

```
someComponent = React.createClass({
    statics: {
        someMethod: function () {
            // ..
        }
    }
})
```

But you can write statics inside ES6+ classes as below,

```
class Component extends React.Component {
    static propTypes = {
        // ...
    }

    static someMethod() {
        // ...
    }
}
```

or writing them outside class as below,

```
class Component extends React.Component {
    ....
}

Component.propTypes = { ...}
Component.someMethod = function () {....}
```

## 153. Can Redux only be used with React?

Redux can be used as a data store for any UI layer. The most common usage is with React and React Native, but there are bindings available for Angular,

Angular 2, Vue, Mithril, and more. Redux simply provides a subscription mechanism which can be used by any other code.

## 154.    Do you need to have a particular build tool to use Redux?

Redux is originally written in ES6 and transpiled for production into ES5 with Webpack and Babel. You should be able to use it regardless of your JavaScript build process. Redux also offers a UMD build that can be used directly without any build process at all.

## 155.    How Redux Form `initialValues` get updated from state?

You need to add enableReinitialize : true setting.

```
const InitializeFromStateForm = reduxForm({
    form: 'initializeFromState',
    enableReinitialize: true
})(UserEdit)
```

If your `initialValues` prop gets updated, your form will update too.

## 156.    How React PropTypes allow different types for one prop?

You can use **oneOfType()** method of PropTypes.
For example, the height property can be defined with
either string or number type as below:

```
Component.PropTypes = {
    size: PropTypes.oneOfType([
        PropTypes.string,
        PropTypes.number
    ])
}
```

## 157.    Can I import an SVG file as react component?

You can import SVG directly as component instead of loading it as a file. This feature is available with `react-scripts@2.0.0` and higher.

```
import { ReactComponent as Logo } from './logo.svg'

const App = () => (
    <div>
        {/* Logo is an actual react component */}
        <Logo />
    </div>
)
```

**Note**: Don't forget about the curly braces in the import.

## 158. Why are inline ref callbacks or functions not recommended?

If the ref callback is defined as an inline function, it will get called twice during updates, first with null and then again with the DOM element. This is because a new instance of the function is created with each render, so React needs to clear the old ref and set up the new one.

```
class UserForm extends Component {
    handleSubmit = () => {
        console.log("Input Value is: ", this.input.value)
    }


    render() {
        return (
            <form onSubmit={this.handleSubmit}>
                <input
                    type='text'
                    ref={(input) => this.input = input} /> // Access DOM input in handle submit
                <button type='submit'>Submit</button>
            </form>
        )
    }
}
```

But our expectation is for the ref callback to get called once, when the component mounts. One quick fix is to use the ES7 class property syntax to define the function

```jsx
class UserForm extends Component {
    handleSubmit = () => {
        console.log("Input Value is: ", this.input.value)
    }

    setSearchInput = (input) => {
        this.input = input
    }

    render() {
        return (
            <form onSubmit={this.handleSubmit}>
                <input
                    type='text'
                    ref={this.setSearchInput} /> // Access DOM
input in handle submit
                <button type='submit'>Submit</button>
            </form>
        )
    }
}
```

## 159.    What is render hijacking in react?

The concept of render hijacking is the ability to control what a component will output from another component. It actually means that you decorate your component by wrapping it into a Higher-Order component. By wrapping you can inject additional props or make other changes, which can cause changing logic of rendering. It does not actually enables hijacking, but by using HOC you make your component behave in different way.

## 160.    What are HOC factory implementations?

There are two main ways of implementing HOCs in React.

    i.    Props Proxy (PP) and

ii.   Inheritance Inversion (II).

But they follow different approaches for manipulating the *WrappedComponent*.

**Props Proxy**

In this approach, the render method of the HOC returns a React Element of the type of the WrappedComponent. We also pass through the props that the HOC receives, hence the name **Props Proxy**.

```
function ppHOC(WrappedComponent) {
    return class PP extends React.Component {
        render() {
            return <WrappedComponent {...this.props} />
        }
    }
}
```

**Inheritance Inversion**

In this approach, the returned HOC class (Enhancer) extends the WrappedComponent. It is called Inheritance Inversion because instead of the WrappedComponent extending some Enhancer class, it is passively extended by the Enhancer. In this way the relationship between them seems **inverse**.

```
function iiHOC(WrappedComponent) {
    return class Enhancer extends WrappedComponent {
        render() {
            return super.render()
        }
    }
}
```

## 161.   How to pass numbers to React component?

You should be passing the numbers via curly **braces({})** where as strings in quotes

```
React.render(<User age={30} department={"IT"} />, document.getE
lementById('container'));
```

## 162. Do I need to keep all my state into Redux? Should I ever use react internal state?

It is up to developer decision. i.e, It is developer job to determine what kinds of state make up your application, and where each piece of state should live. Some users prefer to keep every single piece of data in Redux, to maintain a fully serializable and controlled version of their application at all times. Others prefer to keep non-critical or UI state, such as "is this dropdown currently open", inside a component's internal state.

Below are the thumb rules to determine what kind of data should be put into Redux

i. Do other parts of the application care about this data?
ii. Do you need to be able to create further derived data based on this original data?
iii. Is the same data being used to drive multiple components?
iv. Is there value to you in being able to restore this state to a given point in time (ie, time travel debugging)?
v. Do you want to cache the data (i.e, use what's in state if it's already there instead of re-requesting it)?

## 163. What is the purpose of registerServiceWorker in React?

React creates a service worker for you without any configuration by default. The service worker is a web API that helps you cache your assets and other files so that when the user is offline or on slow network, he/she can still see results on the screen, as such, it helps you build a better user experience, that's what you should know about service worker's for now. It's all about adding offline capabilities to your site.

```
import React from 'react';

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

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

## 164.    What is React memo function?

Class components can be restricted from rendering when their input props are the same using **PureComponent or shouldComponentUpdate**. Now you can do the same with function components by wrapping them in **React.memo**.

```
const MyComponent = React.memo(function MyComponent(props) {
    /* only rerenders if props change */
});
```

## 165.    What is React lazy function?

The `React.lazy` function lets you render an dynamic import as a regular component. It will automatically load the bundle containing the OtherComponent when the component gets rendered. This must return a Promise which resolves to a module with a default export containing a React component.

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {
    return (
        <div>
            <OtherComponent />
        </div>
    );
}
```

**Note:** React.lazy and Suspense is not yet available for server-side rendering. If you want to do code-splitting in a server rendered app, we still recommend React Loadable.

## 166.    How to prevent unnecessary updates using setState?

You can compare current value of the state with an existing state value and decide whether to rerender the page or not. If the values are same then you need to return **null** to stop re-rendering otherwise return the latest state value.

For example, the user profile information is conditionally rendered as follows,

```
getUserProfile = user => {
    const latestAddress = user.address;
    this.setState(state => {
        if (state.address === latestAddress) {
            return null;
        } else {
            return { title: latestAddress };
        }
    });
};
```

## 167. How do you render Array, Strings and Numbers in React 16 Version?

**Arrays**: Unlike older releases, you don't need to make sure **render** method return a single element in React16. You are able to return multiple sibling elements without a wrapping element by returning an array.

For example, let us take the below list of developers,

```
const ReactJSDevs = () => {
    return [
        <li key="1">John</li>,
        <li key="2">Jackie</li>,
        <li key="3">Jordan</li>
    ];
}
```

You can also merge this array of items in another array component.

```
const JSDevs = () => {
    return (
        <ul>
            <li>Brad</li>
            <li>Brodge</li>
            <ReactJSDevs />
            <li>Brandon</li>
```

```
            </ul>
    );
}
```

**Strings and Numbers:** You can also return string and number type from the render method.

```
render() {
    return 'Welcome to ReactJS questions';
}
// Number
render() {
    return 2018;
}
```

### 168.    How to use class field declarations syntax in React classes?

React Class Components can be made much more concise using the class field declarations. You can initialize local state without using the constructor and declare class methods by using arrow functions without the extra need to bind them.

Let's take a counter example to demonstrate class field declarations for state without using constructor and methods without binding,

```
class Counter extends Component {
    state = { value: 0 };

    handleIncrement = () => {
        this.setState(prevState => ({
            value: prevState.value + 1
        }));
    };

    handleDecrement = () => {
        this.setState(prevState => ({
            value: prevState.value - 1
        }));
    };

    render() {
        return (
            <div>
```

```
                {this.state.value}

                <button onClick={this.handleIncrement}>+</butto
n>
                <button onClick={this.handleDecrement}>-
</button>
            </div>
        )
    }
}
```

## 169.    What are hooks?

Hooks is a new feature(React 16.8) that lets you use state and other React features without writing a class.

Let's see an example of useState hook example,

```
import { useState } from 'react';

function Example() {
    // Declare a new state variable, which we'll call "count"
    const [count, setCount] = useState(0);

    return (
        <div>
            <p>You clicked {count} times</p>
            <button onClick={() => setCount(count + 1)}>
                Click me
        </button>
            </div>
    );
}
```

## 170.    What are the rules needs to follow for hooks?

You need to follow two rules in order to use hooks,

i.    Call Hooks only at the top level of your react functions. i.e, You shouldn't call Hooks inside loops, conditions, or nested functions. This will ensure

that Hooks are called in the same order each time a component renders and it preserves the state of Hooks between multiple useState and useEffect calls.

ii. Call Hooks from React Functions only. i.e, You shouldn't call Hooks from regular JavaScript functions.

## 171. How to ensure hooks followed the rules in your project?

React team released an ESLint plugin called **eslint-plugin-react-hooks** that enforces these two rules. You can add this plugin to your project using the below command,

```
npm install eslint-plugin-react-hooks@next
```

And apply the below config in your ESLint config file,

```
// Your ESLint configuration
{
    "plugins": [
        // ...
        "react-hooks"
    ],
        "rules": {
        // ...
        "react-hooks/rules-of-hooks": "error"
    }
}
```

**Note:** This plugin is intended to use in Create React App by default.

## 172. What are the differences between Flux and Redux?

Below are the major differences between Flux and Redux

| Flux | Redux |
| --- | --- |
| State is mutable | State is immutable |
| The Store contains both state and change | The Store and change logic are separate |

| Flux | Redux |
|---|---|
| logic | |
| There are multiple stores exist | There is only one store exist |
| All the stores are disconnected and flat | Single store with hierarchical reducers |
| It has a singleton dispatcher | There is no concept of dispatcher |
| React components subscribe to the store | Container components uses connect function |

## 173. What are the benefits of React Router V4?

Below are the main benefits of React Router V4 module,

    i.   In React Router v4(version 4), the API is completely about components. A router can be visualized as a single component(`<BrowserRouter>`) which wraps specific child router components(`<Route>`).

    ii.   You don't need to manually set history. The router module will take care history by wrapping routes with `<BrowserRouter>` component.

    iii.   The application size is reduced by adding only the specific router module(Web, core, or native)

## 174. Can you describe about componentDidCatch lifecycle method signature?

The **componentDidCatch** lifecycle method is invoked after an error has been thrown by a descendant component. The method receives two parameters,

    i.   error: - The error object which was thrown

    ii.   info: - An object with a componentStack key contains the information about which component threw the error.

The method structure would be as follows

```
componentDidCatch(error, info)
```

### 175.   In which scenarios error boundaries do not catch errors?

Below are the cases in which error boundaries doesn't work,

i.   Inside Event handlers

ii.   Asynchronous code using **setTimeout or requestAnimationFrame** callbacks

iii.   During Server side rendering

iv.   When errors thrown in the error boundary code itself

### 176.   Why do not you need error boundaries for event handlers?

Error boundaries do not catch errors inside event handlers. Event handlers don't happened or invoked during rendering time unlike render method or lifecycle methods. So React knows how to recover these kind of errors in event handlers. If still you need to catch an error inside event handler, use the regular JavaScript try / catch statement as below

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props);
        this.state = { error: null };
    }

    handleClick = () => {
        try {
            // Do something that could throw
        } catch (error) {
            this.setState({ error });
        }
    }

    render() {
        if (this.state.error) {
            return <h1>Caught an error.</h1>
        }
        return <div onClick={this.handleClick}>Click Me</div>
```

```
        }
}
```

The above code is catching the error using vanilla javascript try/catch block instead of error boundaries.

## 177. What is the difference between try catch block and error boundaries?

Try catch block works with imperative code whereas error boundaries are meant for declarative code to render on the screen.

For example, the try catch block used for below imperative code

```
try {
    showButton();
} catch (error) {
    // ...
}
```

Whereas error boundaries wrap declarative code as below,

```
<ErrorBoundary>
    <MyComponent />
</ErrorBoundary>
```

So if an error occurs in a **componentDidUpdate** method caused by a **setState** somewhere deep in the tree, it will still correctly propagate to the closest error boundary.

## 178. What is the behavior of uncaught errors in react 16?

In React 16, errors that were not caught by any error boundary will result in unmounting of the whole React component tree. The reason behind this decision is that it is worse to leave corrupted UI in place than to completely remove it. For example, it is worse for a payments app to display a wrong amount than to render nothing.

## 179. What is the proper placement for error boundaries?

The granularity of error boundaries usage is up to the developer based on project needs. You can follow either of these approaches,

i. You can wrap top-level route components to display a generic error message for the entire application.
ii. You can also wrap individual components in an error boundary to protect them from crashing the rest of the application.

## 180. What is the benefit of component stack trace from error boundary?

Apart from error messages and javascript stack, React16 will display the component stack trace with file names and line numbers using error boundary concept.

For example, BuggyCounter component displays the component stack trace as below,

```
▶React caught an error thrown by BuggyCounter. You should fix this error in your code.    react-dom.development.js:7708
React will try to recreate this component tree from scratch using the error boundary you provided, ErrorBoundary.

Error: I crashed!

The error is located at:
    in BuggyCounter (at App.js:26)
    in ErrorBoundary (at App.js:21)
    in div (at App.js:8)
    in App (at index.js:5)
```

## 181. What is the required method to be defined for a class component?

The `render()` method is the only required method in a class component. i.e, All methods other than render method are optional for a class component.

## 182. What are the possible return types of render method?

Below are the list of following types used and return from render method,

i. **React elements:** Elements that instruct React to render a DOM node. It includes html elements such as `<div/>` and user defined elements.
ii. **Arrays and fragments:** Return multiple elements to render as Arrays and Fragments to wrap multiple elements
iii. **Portals:** Render children into a different DOM subtree.
iv. **String and numbers:** Render both Strings and Numbers as text nodes in the DOM

v. **Booleans or null:** Doesn't render anything but these types are used to conditionally render content.

## 183. What is the main purpose of constructor?

The constructor is mainly used for two purposes,

i. To initialize local state by assigning object to this.state

ii. For binding event handler methods to the instance For example, the below code covers both the above cases,

```
constructor(props) {
    super(props);
    // Don't call this.setState() here!
    this.state = { counter: 0 };
    this.handleClick = this.handleClick.bind(this);
}
```

## 184. Is it mandatory to define constructor for React component?

No, it is not mandatory. i.e, If you don't initialize state and you don't bind methods, you don't need to implement a constructor for your React component.

## 185. What are default props?

The defaultProps are defined as a property on the component class to set the default props for the class. This is used for undefined props, but not for null props.

For example, let us create color default prop for the button component,

```
class MyButton extends React.Component {
    // ...
}

MyButton.defaultProps = {
    color: 'red'
```

```
    };
```

If props.color is not provided then it will set the default value to 'red'. i.e,
Whenever you try to access the color prop it uses default value

```
render() {
    return <MyButton />;
            // props.color will be set to red
}
```

**Note:** If you provide null value then it remains null value.

## 186.    Why should not call setState in componentWillUnmount?

You should not call **setState()** in componentWillUnmount() because once a
component instance is unmounted, it will never be mounted again.

## 187.    What is the purpose of getDerivedStateFromError?

This lifecycle method is invoked after an error has been thrown by a
descendant component. It receives the error that was thrown as a parameter
and should return a value to update state.

The signature of the lifecycle method is as follows,

**static getDerivedStateFromError(error)**

Let us take error boundary use case with the above lifecycle method for
demonistration purpose,

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

    static getDerivedStateFromError(error) {
        // Update state so the next render will show the fallba
ck UI.
        return { hasError: true };
    }
```

```
render() {
    if (this.state.hasError) {
        // You can render any custom fallback UI
        return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

## 188.    What is the methods order when component re-rendered?

An update can be caused by changes to props or state. The below methods are called in the following order when a component is being re-rendered.

   i.    static getDerivedStateFromProps()
   ii.   shouldComponentUpdate()
   iii.  render()
   iv.  getSnapshotBeforeUpdate()
   v.   componentDidUpdate()

## 189.    What are the methods invoked during error handling?

Below methods are called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component.

   i.    static getDerivedStateFromError()
   ii.   componentDidCatch()

## 190.    What is the purpose of displayName class property?

The displayName string is used in debugging messages. Usually, you don't need to set it explicitly because it's inferred from the name of the function or class that defines the component. You might want to set it explicitly if you

want to display a different name for debugging purposes or when you create a higher-order component.

For example, To ease debugging, choose a display name that communicates that it's the result of a withSubscription HOC.

```
function withSubscription(WrappedComponent) {

    class WithSubscription extends React.Component {/* ... */ }

    WithSubscription.displayName = `WithSubscription(${getDispl
ayName(WrappedComponent)})`;
    return WithSubscription;
}

function getDisplayName(WrappedComponent) {
    return WrappedComponent.displayName || WrappedComponent.nam
e || 'Component';
}
```

## 191.    What is the browser support for react applications?

React supports all popular browsers, including Internet Explorer 9 and above, although some polyfills are required for older browsers such as IE 9 and IE 10. If you use **es5-shim and es5-sham** polyfill then it even support old browsers that doesn't support ES5 methods.

## 192.    What is the purpose of unmountComponentAtNode method?

This method is available from react-dom package and it removes a mounted React component from the DOM and clean up its event handlers and state. If no component was mounted in the container, calling this function does nothing. Returns true if a component was unmounted and false if there was no component to unmount.

The method signature would be as follows,

```
ReactDOM.unmountComponentAtNode(container)
```

## 193. What is code-splitting?

Code-Splitting is a feature supported by bundlers like Webpack and Browserify which can create multiple bundles that can be dynamically loaded at runtime. The react project supports code splitting via dynamic import() feature.

For example, in the below code snippets, it will make moduleA.js and all its unique dependencies as a separate chunk that only loads after the user clicks the 'Load' button. **moduleA.js**

```
const moduleA = 'Hello';

export { moduleA };
```

**App.js**

```
import React, { Component } from 'react';

class App extends Component {
    handleClick = () => {
        import('./moduleA')
            .then(({ moduleA }) => {
                // Use moduleA
            })
            .catch(err => {
                // Handle failure
            });
    };

    render() {
        return (
            <div>
                <button onClick={this.handleClick}>Load</button>
            </div>
        );
    }
}

export default App;
```

## 194. What is the benefit of strict mode?

The will be helpful in the below cases

  i.   Identifying components with **unsafe lifecycle methods**.
 ii.   Warning about **legacy string ref** API usage.
iii.   Detecting unexpected **side effects**.
 iv.   Detecting **legacy context** API.
  v.   Warning about deprecated findDOMNode usage

## 195.    What are Keyed Fragments?

The Fragments declared with the explicit <React.Fragment> syntax may have keys. The general use case is mapping a collection to an array of fragments as below,

```
function Glossary(props) {
    return (
        <dl>
            {props.items.map(item => (
                // Without the `key`, React will fire a key warning
                <React.Fragment key={item.id}>
                    <dt>{item.term}</dt>
                    <dd>{item.description}</dd>
                </React.Fragment>
            ))}
        </dl>
    );
}
```

**Note:** key is the only attribute that can be passed to Fragment. In the future, there might be a support for additional attributes, such as event handlers.

## 196.    Does React support all HTML attributes?

As of React 16, both standard or custom DOM attributes are fully supported. Since React components often take both custom and DOM-related props, React uses the camelCase convention just like the DOM APIs.

Let us take few props with respect to standard HTML attributes,

```
<div tabIndex="-1" />      // Just like node.tabIndex DOM API
<div className="Button" /> // Just like node.className DOM API
<input readOnly={true} />  // Just like node.readOnly DOM API
```

These props work similarly to the corresponding HTML attributes, with the exception of the special cases. It also support all SVG attributes.

## 197.   What are the limitations with HOCs?

Higher-order components come with a few caveats apart from its benefits. Below are the few listed in an order,

**Don't use HOCs inside the render method:** It is not recommended to apply a HOC to a component within the render method of a component.

```
render() {
    // A new version of EnhancedComponent is created on ever
y render
    // EnhancedComponent1 !== EnhancedComponent2
    const EnhancedComponent = enhance(MyComponent);
    // That causes the entire subtree to unmount/remount eac
h time!
    return <EnhancedComponent />;
}
```

The above code impact performance by remounting a component that causes the state of that component and all of its children to be lost. Instead, apply HOCs outside the component definition so that the resulting component is created only once.

**Static methods must be copied over:** When you apply a HOC to a component the new component does not have any of the static methods of the original component

```
// Define a static method
WrappedComponent.staticMethod = function () {/*...*/ }
// Now apply a HOC
const EnhancedComponent = enhance(WrappedComponent);

// The enhanced component has no static method
```

```
typeof EnhancedComponent.staticMethod === 'undefined' // tru
e
```

You can overcome this by copying the methods onto the container before returning it,

```
function enhance(WrappedComponent) {
    class Enhance extends React.Component {/*...*/ }
    // Must know exactly which method(s) to copy :(
    Enhance.staticMethod = WrappedComponent.staticMethod;
    return Enhance;
}
```

**Refs aren't passed through:** For HOCs you need to pass through all props to the wrapped component but this does not work for refs. This is because ref is not really a prop similar to key. In this case you need to use the React.forwardRef API

## 198.    How to debug forwardRefs in DevTools?

**React.forwardRef** accepts a render function as parameter and DevTools uses this function to determine what to display for the ref forwarding component.

For example, If you don't name the render function or not using displayName property then it will appear as "ForwardRef" in the DevTools,

```
const WrappedComponent = React.forwardRef((props, ref) => {
    return <LogProps {...props} forwardedRef={ref} />;
});
```

But If you name the render function then it will appear as **"ForwardRef(myFunction)"**

```
const WrappedComponent = React.forwardRef(
    function myFunction(props, ref) {
        return <LogProps {...props} forwardedRef={ref} />;
    }
);
```

As an alternative, You can also set displayName property for forwardRef function,

```
function logProps(Component) {
```

```
    class LogProps extends React.Component {
        // ...
    }

    function forwardRef(props, ref) {
        return <LogProps {...props} forwardedRef={ref} />;
    }

    // Give this component a more helpful display name in DevTo
ols.
    // e.g. "ForwardRef(logProps(MyComponent))"
    const name = Component.displayName || Component.name;
    forwardRef.displayName = `logProps(${name})`;

    return React.forwardRef(forwardRef);
}
```

## 199. When component props defaults to true?

If you pass no value for a prop, it defaults to true. This behavior is available so that it matches the behavior of HTML.

For example, below expressions are equivalent,

```
<MyInput autocomplete />

<MyInput autocomplete={true} />
```

**Note:** It is not recommended to use this approach because it can be confused with the ES6 object shorthand (example, {name} which is short for {name: name})

## 200. What is NextJS and major features of it?

Next.js is a popular and lightweight framework for static and server-rendered applications built with React. It also provides styling and routing solutions. Below are the major features provided by NextJS,

i.   Server-rendered by default
ii.  Automatic code splitting for faster page loads

iii.  Simple client-side routing (page based)

iv.  Webpack-based dev environment which supports (HMR)

v.  Able to implement with Express or any other Node.js HTTP server

vi.  Customizable with your own Babel and Webpack configurations

## 201.  How do you pass an event handler to a component?

You can pass event handlers and other functions as props to child components. It can be used in child component as below,

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

## 202.  Is it good to use arrow functions in render methods?

Yes, You can use. It is often the easiest way to pass parameters to callback functions. But you need to optimize the performance while using it.

```
class Foo extends Component {
    handleClick() {
      console.log('Click happened');
    }
    render() {
      return <button onClick={() => this.handleClick()}>Click M
e</button>;
    }
  }
```

**Note:** Using an arrow function in render method creates a new function each time the component renders, which may have performance implications

## 203.  How to prevent a function from being called multiple times?

If you use an event handler such as **onClick or onScroll** and want to prevent the callback from being fired too quickly, then you can limit the rate at which callback is executed. This can be achieved in the below possible ways,

i.  **Throttling:** Changes based on a time based frequency. For example, it can be used using _.throttle lodash function

ii. **Debouncing:** Publish changes after a period of inactivity. For example, it can be used using _.debounce lodash function

iii. **RequestAnimationFrame throttling:** Changes based on requestAnimationFrame. For example, it can be used using raf-schd lodash function

## 204. How JSX prevents Injection Attacks?

React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that's not explicitly written in your application. Everything is converted to a string before being rendered.

For example, you can embed user input as below,

```
const name = response.potentiallyMaliciousInput;
const element = <h1>{name}</h1>;
```

This way you can prevent XSS(Cross-site-scripting) attacks in the application.

## 205. How do you update rendered elements?

You can update UI(represented by rendered element) by passing the newly created element to ReactDOM's render method.

For example, lets take a ticking clock example, where it updates the time by calling render method multiple times,

```
function tick() {
    const element = (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {new Date().toLocaleTimeString()}.</h2>
      </div>
    );
    ReactDOM.render(element, document.getElementById('root'));
  }

  setInterval(tick, 1000);
```

## 206.    How do you say that props are read only?

When you declare a component as a function or a class, it must never modify its own props.

Let us take a below capital function,

```
function capital(amount, interest) {
    return amount + interest;
 }
```

The above function is called "pure" because it does not attempt to change their inputs, and always return the same result for the same inputs. Hence, React has a single rule saying "All React components must act like pure functions with respect to their props."

## 207.    How do you say that state updates are merged?

When you call setState() in the component, React merges the object you provide into the current state.

For example, let us take a facebook user with posts and comments details as state variables,

```
  constructor(props) {

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

Now you can update them independently with separate `setState()` calls as below,

```
  componentDidMount() {

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

    fetchComments().then(response => {
```

```
        this.setState({
            comments: response.comments
        });
    });
}
```

As mentioned in the above code snippets, **this.setState({comments})** updates only comments variable without modifying or replacing posts variable.

## 208.    How do you pass arguments to an event handler?

During iterations or loops, it is common to pass an extra parameter to an event handler. This can be achieved through arrow functions or bind method.

Let us take an example of user details updated in a grid,

```
<button onClick={(e) => this.updateUser(userId, e)}>Update User
 details</button>

<button onClick={this.updateUser.bind(this, userId)}>Update Use
r details</button>
```

In both the approaches, the synthetic argument e is passed as a second argument. You need to pass it explicitly for arrow functions and it forwarded automatically for bind method.

## 209.    How to prevent component from rendering?

You can prevent component from rendering by returning null based on specific condition. This way it can conditionally render component.

```
function Greeting(props) {
    if (!props.loggedIn) {
        return null;
    }

    return (
        <div className="greeting">
            welcome, {props.name}
        </div>
    );
```

```
}
class User extends React.Component {
    constructor(props) {
        super(props);
        this.state = { loggedIn: false, name: 'John' };
    }

    render() {
        return (
            <div>
                        //Prevent component render if it is not lo
ggedIn
                <Greeting loggedIn={this.state.loggedIn} />
                <UserDetails name={this.state.name}>
            </div>
        );
    }
```

In the above example, the greeting component skips its rendering section by applying condition and returning null value.

## 210.    What are the conditions to safely use the index as a key?

There are three conditions to make sure, it is safe use the index as a key.

i.    The list and items are static– they are not computed and do not change
ii.   The items in the list have no ids
iii.  The list is never reordered or filtered.

## 211.    Is it keys should be globally unique?

Keys used within arrays should be unique among their siblings but they don't need to be globally unique. i.e, You can use the same keys with two different arrays.

For example, the below book component uses two arrays with different arrays,

```
function Book(props) {
    const index = (
```

```
        <ul>
            {props.pages.map((page) =>
                <li key={page.id}>
                    {page.title}
                </li>
            )}
        </ul>
    );
    const content = props.pages.map((page) =>
        <div key={page.id}>
            <h3>{page.title}</h3>
            <p>{page.content}</p>
            <p>{page.pageNumber}</p>
        </div>
    );
    return (
        <div>
            {index}
            <hr />
            {content}
        </div>
    );
}
```

## 212.    What is the popular choice for form handling?

Formik is a form library for react which provides solutions such as validation, keeping track of the visited fields, and handling form submission.

In detail, You can categorize them as follows,

  i.   Getting values in and out of form state
 ii.   Validation and error messages
iii.   Handling form submission

It is used to create a scalable, performant, form helper with a minimal API to solve annoying stuff.

## 213.    What are the advantages of formik over redux form library?

Below are the main reasons to recommend formik over redux form library,

    i.    The form state is inherently short-term and local, so tracking it in Redux (or any kind of Flux library) is unnecessary.

    ii.    Redux-Form calls your entire top-level Redux reducer multiple times ON EVERY SINGLE KEYSTROKE. This way it increases input latency for large apps.

    iii.    Redux-Form is 22.5 kB minified gzipped whereas Formik is 12.7 kB

## 214.    Why do you not required to use inheritance?

In React, it is recommend using composition instead of inheritance to reuse code between components. Both Props and composition give you all the flexibility you need to customize a component's look and behavior in an explicit and safe way. Whereas, If you want to reuse non-UI functionality between components, it is suggested to extracting it into a separate JavaScript module. Later components import it and use that function, object, or a class, without extending it.

## 215.    Can I use web components in react application?

Yes, you can use web components in a react application. Even though many developers won't use this combination, it may require especially if you are using third-party UI components that are written using Web Components.

For example, let us use `Vaadin` date picker web component as below,

```
import React, { Component } from 'react';
import './App.css';
import '@vaadin/vaadin-date-picker';
class App extends Component {
    render() {
        return (
            <div className="App">
                <vaadin-date-
picker label="When were you born?"></vaadin-date-picker>
            </div>
```

```
        );
    }
}
}
export default App;
```

## 216.  What is dynamic import?

The dynamic import() syntax is a ECMAScript proposal not currently part of the language standard. It is expected to be accepted in the near future. You can achieve code-splitting into your app using dynamic import.

Let's take an example of addition,

  i.  **Normal Import**

```
import { add } from './math';
console.log(add(10, 20));
```

  ii.  **Dynamic Import**

```
import("./math").then(math => {
    console.log(math.add(10, 20));
});
```

## 217.  What are loadable components?

If you want to do code-splitting in a server rendered app, it is recommend to use Loadable Components because React.lazy and Suspense is not yet available for server-side rendering. Loadable lets you render a dynamic import as a regular component.

Lets take an example,

```
import loadable from '@loadable/component'

const OtherComponent = loadable(() => import('./OtherComponent'
))

function MyComponent() {
    return (
```

```
        <div>
            <OtherComponent />
        </div>
    )
}
```

Now OtherComponent will be loaded in a separated bundle

### 218.    What is suspense component?

If the module containing the dynamic import is not yet loaded by the time parent component renders, you must show some fallback content while you're waiting for it to load using a loading indicator. This can be done using **Suspense** component.

For example, the below code uses suspense component,

```
const OtherComponent = React.lazy(() => import('./OtherComponen
t'));

function MyComponent() {
    return (
        <div>
            <Suspense fallback={<div>Loading...</div>}>
                <OtherComponent />
            </Suspense>
        </div>
    );
}
```

As mentioned in the above code, Suspense is wrapped above the lazy component.

### 219.    What is route based code splitting?

One of the best place to do code splitting is with routes. The entire page is going to re-render at once so users are unlikely to interact with other elements in the page at the same time. Due to this, the user experience won't be disturbed.

Let us take an example of route based website using libraries like React Router with React.lazy,

```
import { BrowserRouter as Router, Route, Switch } from 'react-
router-dom';
import React, { Suspense, lazy } from 'react';

const Home = lazy(() => import('./routes/Home'));
const About = lazy(() => import('./routes/About'));

const App = () => (
    <Router>
        <Suspense fallback={<div>Loading...</div>}>
            <Switch>
                <Route exact path="/" component={Home} />
                <Route path="/about" component={About} />
            </Switch>
        </Suspense>
    </Router>
);
```

In the above code, the code splitting will happen at each route level.

## 220.    Give an example on How to use context?

**Context** is designed to share data that can be considered **global** for a tree of React components.

For example, in the code below lets manually thread through a "theme" prop in order to style the Button component.

```
//Lets create a context with a default theme value "luna"
const ThemeContext = React.createContext('luna');
// Create App component where it uses provider to pass theme va
lue in the tree
class App extends React.Component {
    render() {
        return (
            <ThemeContext.Provider value="nova">
                <Toolbar />
            </ThemeContext.Provider>
```

```
        );
    }
}
// A middle component where you don't need to pass theme prop a
nymore
function Toolbar(props) {
    return (
        <div>
            <ThemedButton />
        </div>
    );
}
// Lets read theme value in the button component to use
class ThemedButton extends React.Component {
    static contextType = ThemeContext;
    render() {
        return <Button theme={this.context} />;
    }
}
```

### 221.    What is the purpose of default value in context?

The defaultValue argument is only used when a component does not have a matching Provider above it in the tree. This can be helpful for testing components in isolation without wrapping them.

Below code snippet provides default theme value as Luna.

```
const MyContext = React.createContext(defaultValue);
```

### 222.    How do you use contextType?

ContextType is used to consume the context object. The contextType property can be used in two ways,

i.   **contextType as property of class:** The contextType property on a class can be assigned a Context object created by React.createContext(). After that, you can consume the nearest current value of that Context type using this.context in any of the lifecycle methods and render function.

Lets assign contextType property on MyClass as below,

```
class MyClass extends React.Component {
    componentDidMount() {
        let value = this.context;
        /* perform a side-
effect at mount using the value of MyContext */
    }
    componentDidUpdate() {
        let value = this.context;
        /* ... */
    }
    componentWillUnmount() {
        let value = this.context;
        /* ... */
    }
    render() {
        let value = this.context;
        /* render something based on the value of MyContex
t */
    }
}
MyClass.contextType = MyContext;
```

**Static field** You can use a static class field to initialize your contextType using public class field syntax.

```
class MyClass extends React.Component {
    static contextType = MyContext;
    render() {
        let value = this.context;
        /* render something based on the value */
    }
}
```

## 223.    What is a consumer?

A Consumer is a React component that subscribes to context changes. It requires a function as a child which receives current context value as argument and returns a react node. The value argument passed to the function will be equal to the value prop of the closest Provider for this context above in the tree.

Lets take a simple example,

```
<MyContext.Consumer>
  {value => /* render something based on the context value */}
</MyContext.Consumer>
```

## 224.  How do you solve performance corner cases while using context?

The context uses reference identity to determine when to re-render, there are some gotchas that could trigger unintentional renders in consumers when a provider's parent re-renders.

For example, the code below will re-render all consumers every time the Provider re-renders because a new object is always created for value.

```
class App extends React.Component {
    render() {
        return (
            <Provider value={{ something: 'something' }}>
                <Toolbar />
            </Provider>
        );
    }
}
```

This can be solved by lifting up the value to parent state,

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

    render() {
        return (
            <Provider value={this.state.value}>
                <Toolbar />
            </Provider>
        );
    }
}
```

### 225. What is the purpose of forward ref in HOCs?

Refs will not get passed through because ref is not a prop. It handled differently by React just like **key**. If you add a ref to a HOC, the ref will refer to the outermost container component, not the wrapped component. In this case, you can use Forward Ref API. For example, we can explicitly forward refs to the inner FancyButton component using the React.forwardRef API.

The below HOC logs all props,

```javascript
function logProps(Component) {
    class LogProps extends React.Component {
        componentDidUpdate(prevProps) {
            console.log('old props:', prevProps);
            console.log('new props:', this.props);
        }

        render() {
            const { forwardedRef, ...rest } = this.props;

            // Assign the custom prop "forwardedRef" as a ref
            return <Component ref={forwardedRef} {...rest} />;
        }
    }

    return React.forwardRef((props, ref) => {
        return <LogProps {...props} forwardedRef={ref} />;
    });
}
```

Let's use this HOC to log all props that get passed to our "fancy button" component,

```javascript
class FancyButton extends React.Component {
    focus() {
        // ...
    }

    // ...
}
export default logProps(FancyButton);
```

```
```

Now lets create a ref and pass it to FancyButton component. In this case, you can set focus to button element.

```javascript
import FancyButton from './FancyButton';

const ref = React.createRef();
ref.current.focus();
<FancyButton
    label="Click Me"
    handleClick={handleClick}
    ref={ref}
/>;
```

## 226.    Is it ref argument available for all functions or class components?

Regular function or class components don't receive the ref argument, and ref is not available in props either. The second ref argument only exists when you define a component with React.forwardRef call.

## 227.    Why do you need additional care for component libraries while using forward refs?

When you start using forwardRef in a component library, you should treat it as a breaking change and release a new major version of your library. This is because your library likely has a different behavior such as what refs get assigned to, and what types are exported. These changes can break apps and other libraries that depend on the old behavior.

## 228.    How to create react class components without ES6?

If you don't use ES6 then you may need to use the create-react-class module instead. For default props, you need to define getDefaultProps() as a function

on the passed object. Whereas for initial state, you have to provide a separate getInitialState method that returns the initial state.

```
var Greeting = createReactClass({
    getDefaultProps: function () {
        return {
            name: 'Jhohn'
        };
    },
    getInitialState: function () {
        return { message: this.props.message };
    },
    handleClick: function () {
        console.log(this.state.message);
    },
    render: function () {
        return <h1>Hello, {this.props.name}</h1>;
    }
});
```

**Note:** If you use createReactClass then auto binding is available for all methods. i.e, You don't need to use `.bind(this)` with in constructor for event handlers.

### 229.    Is it possible to use react without JSX?

Yes, JSX is not mandatory for using React. Actually it is convenient when you don't want to set up compilation in your build environment. Each JSX element is just syntactic sugar for calling `React.createElement(component, props, ...children)`.

For example, let us take a greeting example with JSX,

```
class Greeting extends React.Component {
    render() {
        return <div>Hello {this.props.message}</div>;
    }
}

ReactDOM.render(
    <Greeting message="World" />,
    document.getElementById('root')
);
```

You can write the same code without JSX as below,

```
class Greeting extends React.Component {
    render() {
        return React.createElement('div', null, `Hello ${this.p
rops.message}`);
    }
}

ReactDOM.render(
    React.createElement(Greeting, { message: 'World' }, null),
    document.getElementById('root')
);
```

## 230. What is diffing algorithm?

React needs to use algorithms to find out how to efficiently update the UI to match the most recent tree. The diffing algorithms is generating the minimum number of operations to transform one tree into another. However, the algorithms have a complexity in the order of O(n3) where n is the number of elements in the tree.

In this case, for displaying 1000 elements would require in the order of one billion comparisons. This is far too expensive. Instead, React implements a heuristic O(n) algorithm based on two assumptions:

   i.  Two elements of different types will produce different trees.
   ii. The developer can hint at which child elements may be stable across different renders with a key prop.

## 231. What are the rules covered by diffing algorithm?

When diffing two trees, React first compares the two root elements. The behavior is different depending on the types of the root elements. It covers the below rules during reconciliation algorithm,

**Elements Of Different Types:** Whenever the root elements have different types, React will tear down the old tree and build the new tree from scratch.

For example, elements to      , or from to of different types lead a full rebuild.

**DOM Elements Of The Same Type:** When comparing two React DOM elements of the same type, React looks at the attributes of both, keeps the same underlying DOM node, and only updates the changed attributes. Lets take an example with same DOM elements except className attribute,

```
<div className="show" title="ReactJS" />

<div className="hide" title="ReactJS" />
```

**Component Elements Of The Same Type:** When a component updates, the instance stays the same, so that state is maintained across renders. React updates the props of the underlying component instance to match the new element, and calls componentWillReceiveProps() and componentWillUpdate() on the underlying instance. After that, the render() method is called and the diff algorithm recurses on the previous result and the new result.

**Recursing On Children:** when recursing on the children of a DOM node, React just iterates over both lists of children at the same time and generates a mutation whenever there's a difference. For example, when adding an element at the end of the children, converting between these two trees works well.

```
<ul>
  <li>first</li>
  <li>second</li>
</ul>

<ul>
  <li>first</li>
  <li>second</li>
  <li>third</li>
</ul>
```

**Handling keys:** React supports a key attribute. When children have keys, React uses the key to match children in the original tree with children in the subsequent tree. For example, adding a key can make the tree conversion efficient,

```
<ul>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>

<ul>
  <li key="2014">Connecticut</li>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>
```

## 232.    When do you need to use refs?

There are few use cases to go for refs,

    i.   Managing focus, text selection, or media playback.

   ii.   Triggering imperative animations.

  iii.   Integrating with third-party DOM libraries.

## 233.    Is it prop must be named as render for render props?

Even though the pattern named render props, you don't have to use a prop named render to use this pattern. i.e, Any prop that is a function that a component uses to know what to render is technically a "render prop". Lets take an example with the children prop for render props,

```
<Mouse children={mouse => (
  <p>The mouse position is {mouse.x}, {mouse.y}</p>
)}/>
```

Actually children prop doesn't need to be named in the list of "attributes" in JSX element. Instead, you can keep it directly inside element,

```
<Mouse>
    {mouse => (
        <p>The mouse position is {mouse.x}, {mouse.y}</p>
    )}
</Mouse>
```

While using this above technique(without any name), explicitly state that children should be a function in your propTypes.

```
Mouse.propTypes = {
    children: PropTypes.func.isRequired
};
```

## 234.    What are the problems of using render props with pure components?

If you create a function inside a render method, it negates the purpose of pure component. Because the shallow prop comparison will always return false for new props, and each render in this case will generate a new value for the render prop. You can solve this issue by defining the render function as instance method.

## 235.    How do you create HOC using render props?

You can implement most higher-order components (HOC) using a regular component with a render prop. For example, if you would prefer to have a withMouse HOC instead of a component, you could easily create one using a regular with a render prop.

```
function withMouse(Component) {
    return class extends React.Component {
        render() {
            return (
                <Mouse render={mouse => (
                    <Component {...this.props} mouse={mouse} />
                )} />
            );
        }
    }
}
```

This way render props gives the flexibility of using either pattern.

## 236.    What is windowing technique?

Windowing is a technique that only renders a small subset of your rows at any given time, and can dramatically reduce the time it takes to re-render the components as well as the number of DOM nodes created. If your application renders long lists of data then this technique is recommended. Both react-window and react-virtualized are popular windowing libraries which provides several reusable components for displaying lists, grids, and tabular data.

### 237.    How do you print falsy values in JSX?

The falsy values such as false, null, undefined, and true are valid children but they don't render anything. If you still want to display them then you need to convert it to string. Let's take an example on how to convert to a string,

```
<div>
  My JavaScript variable is {String(myVariable)}.
</div>
```

### 238.    What is the typical use case of portals?

React portals are very useful when a parent component has overflow: hidden or has properties that affect the stacking context(z-index,position,opacity etc styles) and you need to visually "break out" of its container.

For example, dialogs, global message notifications, hovercards, and tooltips.

### 239.    How do you set default value for uncontrolled component?

In React, the value attribute on form elements will override the value in the DOM. With an uncontrolled component, you might want React to specify the initial value, but leave subsequent updates uncontrolled. To handle this case, you can specify a **defaultValue** attribute instead of **value**.

```
render() {
    return (
        <form onSubmit={this.handleSubmit}>
            <label>
                User Name:
                <input
```

```
                    defaultValue="John"
                    type="text"
                    ref={this.input} />
            </label>
            <input type="submit" value="Submit" />
        </form>
    );
}
```

The same applies for `select` and `textArea` inputs. But you need to use **defaultChecked** for `checkbox` and `radio` inputs.

## 240.    What is your favorite React stack?

Even though the tech stack varies from developer to developer, the most popular stack is used in react boilerplate project code. It mainly uses Redux and redux-saga for state management and asynchronous side-effects, react-router for routing purpose, styled-components for styling react components, axios for invoking REST api, and other supported stack such as webpack, reselect, ESNext, Babel.

## 241.    What is the difference between Real DOM and Virtual DOM?

Below are the main differences between Real DOM and Virtual DOM,

| Real DOM | Virtual DOM |
|---|---|
| Updates are slow | Updates are fast |
| DOM manipulation is very expensive. | DOM manipulation is very easy |
| You can update HTML directly. | You Can't directly update HTML |
| It causes too much of memory wastage | There is no memory wastage |
| Creates a new DOM if element updates | It updates the JSX if element update |

## 242. How to add Bootstrap to a react application?

Bootstrap can be added to your React app in a three possible ways,

i. Using the Bootstrap CDN: This is the easiest way to add bootstrap. Add both bootstrap CSS and JS resources in a head tag.

ii. Bootstrap as Dependency: If you are using a build tool or a module bundler such as Webpack, then this is the preferred option for adding Bootstrap to your React application

```
npm install bootstrap
```

iii. React Bootstrap Package: In this case, you can add Bootstrap to our React app is by using a package that has rebuilt Bootstrap components to work particularly as React components. Below packages are popular in this category,
   a. react-bootstrap
   b. reactstrap

## 243. Can you list down top websites or applications using react as front end framework?

Below are the `top 10 websites` using React as their front-end framework,

i. Facebook
ii. Uber
iii. Instagram
iv. WhatsApp
v. Khan Academy
vi. Airbnb
vii. Dropbox
viii. Flipboard
ix. Netflix
x. PayPal

### 244. Is it recommended to use CSS In JS technique in React?

React does not have any opinion about how styles are defined but if you are a beginner then good starting point is to define your styles in a separate *.css file as usual and refer to them using className. This functionality is not part of React but came from third-party libraries. But If you want to try a different approach(CSS-In-JS) then styled-components library is a good option.

### 245. Do I need to rewrite all my class components with hooks?

No. But you can try Hooks in a few components(or new components) without rewriting any existing code. Because there are no plans to remove classes in ReactJS.

### 246. How to fetch data with React Hooks?

The effect hook called `useEffect` is used to fetch the data with axios from the API and to set the data in the local state of the component with the state hook's update function.

Let's take an example in which it fetches list of react articles from the API

```javascript
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function App() {
    const [data, setData] = useState({ hits: [] });

    useEffect(async () => {
        const result = await axios(
            'http://hn.algolia.com/api/v1/search?query=react',
        );

        setData(result.data);
    }, []);

    return (
        <ul>
            {data.hits.map(item => (
```

```
                <li key={item.objectID}>
                    <a href={item.url}>{item.title}</a>
                </li>
            ))}
        </ul>
    );
}

export default App;
```

Remember we provided an empty array as second argument to the effect hook to avoid activating it on component updates but only for the mounting of the component. i.e, It fetches only for component mount.

## 247.   Is Hooks cover all use cases for classes?

Hooks doesn't cover all use cases of classes but there is a plan to add them soon. Currently there are no Hook equivalents to the uncommon **getSnapshotBeforeUpdate** and **componentDidCatch** lifecycles yet.

## 248.   What is the stable release for hooks support?

React includes a stable implementation of React Hooks in 16.8 release for below packages

   i.    React DOM
   ii.   React DOM Server
   iii.  React Test Renderer
   iv.   React Shallow Renderer

## 249.   Why do we use array destructuring (square brackets notation) in useState?

When we declare a state variable with useState, it returns a pair — an array with two items. The first item is the current value, and the second is a function

that updates the value. Using [0] and [1] to access them is a bit confusing because they have a specific meaning. This is why we use array destructuring instead.

For example, the array index access would look as follows:

```
var userStateVariable = useState('userProfile'); // Returns an array pair

var user = userStateVariable[0]; // Access first item
var setUser = userStateVariable[1]; // Access second item
```
Whereas with array destructuring the variables can be accessed as follows:
```
const [user, setUser] = useState('userProfile');
```

## 250.   What are the sources used for introducing hooks?

Hooks got the ideas from several different sources. Below are some of them,

i.   Previous experiments with functional APIs in the react-future repository

ii.   Community experiments with render prop APIs such as Reactions Component

iii.   State variables and state cells in DisplayScript.

iv.   Subscriptions in Rx.

v.   Reducer components in ReasonReact.

## 251.   How do you access imperative API of web components?

Web Components often expose an imperative API to implement its functions. You will need to use a **ref** to interact with the DOM node directly if you want to access imperative API of a web component. But if you are using third-party Web Components, the best solution is to write a React component that behaves as a **wrapper** for your Web Component.

## 252.   What is formik?

Formik is a small react form library that helps you with the three major problems,

    i.    Getting values in and out of form state

    ii.    Validation and error messages

    iii.    Handling form submission

## 253.    What are typical middleware choices for handling asynchronous calls in Redux?

Some of the popular middleware choices for handling asynchronous calls in Redux eco system are `Redux Thunk, Redux Promise, Redux Saga`.

## 254.    Do browsers understand JSX code?

No, browsers can't understand JSX code. You need a transpiler to convert your JSX to regular Javascript that browsers can understand. The most widely used transpiler right now is Babel.

## 255.    Describe about data flow in react?

React implements one-way reactive data flow using props which reduce boilerplate and is easier to understand than traditional two-way data binding.

## 256.    What is react scripts?

The `react-scripts` package is a set of scripts from the create-react-app starter pack which helps you kick off projects without configuring. The `react-scripts start` command sets up the development environment and starts a server, as well as hot module reloading.

## 257.    What are the features of create react app?

Below are the list of some of the features provided by create react app.

   i.   React, JSX, ES6, Typescript and Flow syntax support.

  ii.   Autoprefixed CSS

 iii.   CSS Reset/Normalize

 iv.   A live development server

  v.   A fast interactive unit test runner with built-in support for coverage reporting

 vi.   A build script to bundle JS, CSS, and images for production, with hashes and sourcemaps

vii.   An offline-first service worker and a web app manifest, meeting all the Progressive Web App criteria.

## 258.    What is the purpose of renderToNodeStream method?

The `ReactDOMServer#renderToNodeStream` method is used to generate HTML on the server and send the markup down on the initial request for faster page loads. It also helps search engines to crawl your pages easily for SEO purposes. **Note:** Remember this method is not available in the browser but only server.

## 259.    What is MobX?

MobX is a simple, scalable and battle tested state management solution for applying functional reactive programming (TFRP). For reactJs application, you need to install below packages,

```
npm install mobx --save
npm install mobx-react --save
```

## 260.    What are the differences between Redux and MobX?

Below are the main differences between Redux and MobX,

| Topic | Redux | MobX |
|---|---|---|
| Definition | It is a javascript library for managing the application state | It is a library for reactively managing the state of your applications |
| Programming | It is mainly written in ES6 | It is written in JavaScript(ES5) |
| Data Store | There is only one large store exist for data storage | There is more than one store for storage |
| Usage | Mainly used for large and complex applications | Used for simple applications |
| Performance | Need to be improved | Provides better performance |
| How it stores | Uses JS Object to store | Uses observable to store the data |

## 261. Should I learn ES6 before learning ReactJS?

No, you don't have to learn es2015/es6 to learn react. But you may find many resources or React ecosystem uses ES6 extensively. Let's see some of the frequently used ES6 features,

**Destructuring:** To get props and use them in a component

```
// in es 5
var someData = this.props.someData
var dispatch = this.props.dispatch

// in es6
const { someData, dispatch } = this.props
```
Spread operator: Helps in passing props down into a component

```
// in es 5
<SomeComponent someData={this.props.someData} dispatch={this
.props.dispatch} />

// in es6
<SomeComponent {...this.props} />
```

```
Arrow functions: Makes compact syntax
// es 5
var users = usersList.map(function (user) {
    return <li>{user.name}</li>
})
// es 6
const users = usersList.map(user => <li>{user.name}</li>);
```

## 262.    What is Concurrent Rendering?

The Concurrent rendering makes React apps to be more responsive by rendering component trees without blocking the main UI thread. It allows React to interrupt a long-running render to handle a high-priority event. i.e, When you enabled concurrent Mode, React will keep an eye on other tasks that need to be done, and if there's something with a higher priority it will pause what it is currently rendering and let the other task finish first. You can enable this in two ways,

```
// 1. Part of an app by wrapping with ConcurrentMode
<React.unstable_ConcurrentMode>
  <Something />
</React.unstable_ConcurrentMode>

// 2. Whole app using createRoot
ReactDOM.unstable_createRoot(domNode).render(<App />);
```

## 263.    What is the difference between async mode and concurrent mode?

Both refers the same thing. Previously concurrent Mode being referred to as "Async Mode" by React team. The name has been changed to highlight React's ability to perform work on different priority levels. So it avoids the confusion from other approaches to Async Rendering.

## 264.    Can I use javascript urls in react16.9?

Yes, you can use javascript: URLs but it will log a warning in the console. Because URLs starting with javascript: are dangerous by including unsanitized output in a tag like `<a href>` and create a security hole.

```
const companyProfile = {
    website: "javascript: alert('Your website is hacked')",
  };
  // It will log a warning
  <a href={companyProfile.website}>More details</a>
```

Remember that the future versions will throw an error for javascript URLs.

## 265.    What is the purpose of eslint plugin for hooks?

The ESLint plugin enforces rules of Hooks to avoid bugs. It assumes that any function starting with "use" and a capital letter right after it is a Hook. In particular, the rule enforces that,

   i.   Calls to Hooks are either inside a PascalCase function (assumed to be a component) or another useSomething function (assumed to be a custom Hook).
   ii.  Hooks are called in the same order on every render.

## 266.    What is the difference between Imperative and Declarative in React?

Imagine a simple UI component, such as a "Like" button. When you tap it, it turns blue if it was previously grey, and grey if it was previously blue.

The imperative way of doing this would be:

```
if( user.likes() ) {
    if( hasBlue() ) {
        removeBlue();
        addGrey();
    } else {
        removeGrey();
        addBlue();
    }
}
```

Basically, you have to check what is currently on the screen and handle all the changes necessary to redraw it with the current state, including undoing the changes from the previous state. You can imagine how complex this could be in a real-world scenario.

In contrast, the declarative approach would be:

```jsx
if( this.state.liked ) {
    return <blueLike />;
} else {
    return <greyLike />;
}
```

Because the declarative approach separates concerns, this part of it only needs to handle how the UI should look in a sepecific state, and is therefore much simpler to understand.

## 267. What are the benefits of using typescript with reactjs?

Below are some of the benefits of using typescript with Reactjs,

i.   It is possible to use latest JavaScript features
ii.  Use of interfaces for complex type definitions
iii. IDEs such as VS Code was made for TypeScript
iv.  Avoid bugs with the ease of readability and Validation

## 268. How do you make sure that user remains authenticated on page refresh while using Context API State Management?

When a user logs in and reload, to persist the state generally we add the load user action in the useEffect hooks in the main App.js. While using Redux, loadUser action can be easily accessed.

**App.js**

```jsx
import {loadUser}  from '../actions/auth';
store.dispatch(loadUser());
```

- But while using **Context API**, to access context in App.js, wrap the AuthState in index.js so that App.js can access the auth context. Now whenever the page reloads, no matter what route you are on, the user will be authenticated as **loadUser** action will be triggered on each re-render.

**index.js**

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import AuthState from './context/auth/AuthState'

ReactDOM.render(
    <React.StrictMode>
        <AuthState>
            <App />
        </AuthState>
    </React.StrictMode>,
    document.getElementById('root')
);
```

**App.js**

```
   const authContext = useContext(AuthContext);


const { loadUser } = authContext;

useEffect(() => {
  loadUser();
},[])
```

**loadUser**

```
    const loadUser = async () => {

    const token = sessionStorage.getItem('token');

    if(!token){
        dispatch({
            type: ERROR
        })
    }
    setAuthToken(token);

    try {
        const res = await axios('/api/auth');
```

```
      dispatch({
          type: USER_LOADED,
          payload: res.data.data
      })

   } catch (err) {
      console.error(err);
   }
}
```

## 269. What are the benefits of new JSX transform?

There are three major benefits of new JSX transform,

i. It is possible to use JSX without importing React packages
ii. The compiled output might improve the bundle size in a small amount
iii. The future improvements provides the flexibility to reduce the number of concepts to learn React.

## 270. How does new JSX transform different from old transform?

The new JSX transform doesn't require React to be in scope. i.e, You don't need to import React package for simple scenarios.

Let's take an example to look at the main differences between the old and the new transform,

**Old Transform:**

```
import React from 'react';

function App() {
   return <h1>Good morning!!</h1>;
}
```

Now JSX transform convert the above code into regular JavaScript as below,

```
import React from 'react';

function App() {
   return React.createElement('h1', null, 'Good morning!!');
}
```

**New Transform:**

The new JSX transform doesn't require any React imports

```
function App() {
    return <h1>Good morning!!</h1>;
  }
```

Under the hood JSX transform compiles to below code

```
import {jsx as _jsx} from 'react/jsx-runtime';

function App() {
  return _jsx('h1', { children: 'Good morning!!' });
}
```

**Note:** You still need to import React to use Hooks.

## 271.    Why is it necessary to capitalize component names?

 It is necessary because components are not DOM elements, they are constructors. Also, in JSX lowercase tag names are referring to HTML elements, not components.

## 272.    What is the difference between createElement()    and cloneElement() methods?

 In JSX the React element is transpiled to React.createElement() which represents an UI element. Whereas React.cloneElement() is used in order to clone an element and pass it new props.

## 273.    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. |

## 274. What are the features of React?

Major features of React are listed below:
iv. It uses the virtual DOM instead of the real DOM.
v. It uses server-side rendering.
vi. It follows uni-directional data flow or data binding.

## 275. 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.

## 276. 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.

**277.    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 exports
// ES5
module.exports = Component;
// ES6
export default Component;

iii. component and function
// ES5
 var MyComponent = React.createClass({ render: function() { return <h3>Hello Sahosoft!</h3>
} }

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

## 278.     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 |

## 279.     What do you understand from "In React, everything is a component."

Components are the building blocks of a React application's UI. These components split up the entire UI into small independent and reusable

pieces. Then it renders each of these components independent of each other without affecting the rest of the UI.

### 280. 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.

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

We can embed components into one in the following way: 1

```
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')
);
```

### 282. 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.

### 283. 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 viathis.state().

### 284. 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 |

### 285. 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('conte
nt'));
```

## 286.     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(th
is)} />); }

//With Arrow
function render() { return (<MyInput onChange={(e) => this.hand
leOnChange(e)} />); }
```

## 287.     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. |

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

There are three different phases of React component's lifecycle:
vii. Initial Rendering Phase: This is the phase when the component is about to start its life journey and make its way to the DOM.
viii. 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.
ix. Unmounting Phase: This is the final phase of a component's life cycle in which the component is destroyed and removed from the DOM.

**289.** **Explain the lifecycle methods of React components in detail.**

Some of the most important lifecycle methods are:
x. **componentWillMount()** – Executed just before rendering takes place both on the client as well as server-side.
xi. **componentDidMount()** – Executed on the client side only after the first render.
xii. **componentWillReceiveProps()** – Invoked as soon as the props are received from the parent class and before another render is called.
xiii. **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.
xiv. **componentWillUpdate()** – Called just before rendering takes place in the DOM.
xv. **componentDidUpdate()** – Called immediately after rendering takes place.

xvi. **componentWillUnmount()** – Called after the component is unmounted from the DOM. It is used to clear up the memory spaces.

**290.    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. ii. 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.

291.    How do you create an event in React?

```
class Display extends React.Component({
    show(evt) { // code },
        render() { // Render the div with an onClick prop (valu
e is a function) return (
            <div onClick={this.show}>Click Me!</div> ); }
    });
```

**292.    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.

**293.    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.

```jsx
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>
        );
    }
}
```

**294.** **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.

**295.** **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.

```jsx
//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>);
    }
}
```

**296.      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} onChang
e={this.handleSubmit} />
        </label>
            <input type="submit" value="Submit" />
        </form>
    );
}
```

**297.      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 |

## 298.      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.

## 299.      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

## 300.      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.

## 301.      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.

## 302. 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

## 303. 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.

## 304. 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.

## 305. What are the three principles that Redux follows?

xvii. **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.

xviii. **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.

xix. **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.

## 306. 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.

## 307. 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.

## 308. Show how the data flows through Redux?
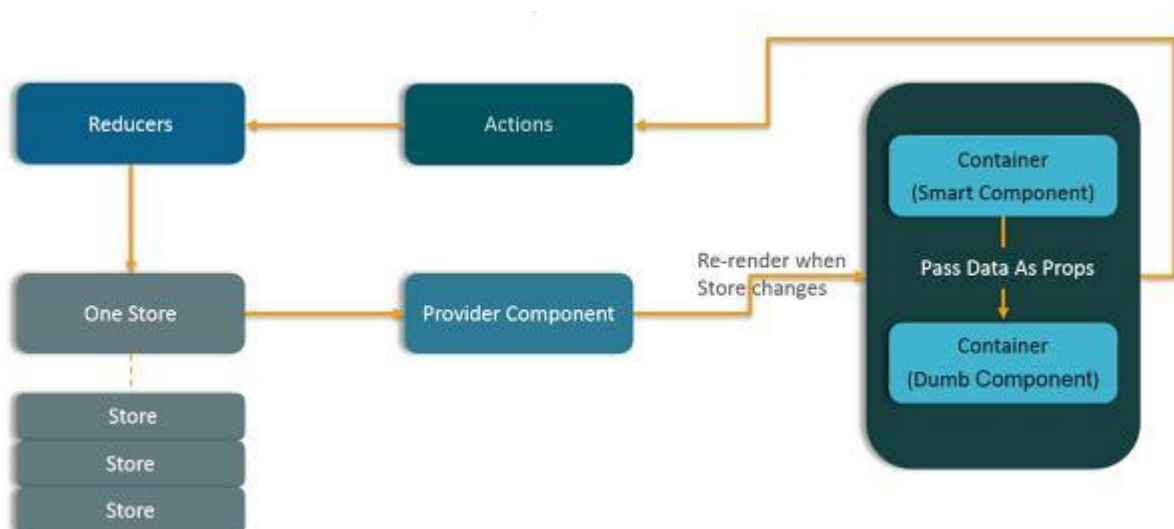
**309.    How are Actions defined in Redux?**

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
    }
}
```

**310.    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.

**311.    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.

**312.    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 |

**313.      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.

**314.      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

behaviour and is used for developing single page web applications. React Router has a simple API.

### 315.    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.

### 316.    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>
```

### 317.    List down the advantages of React Router.

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

**318.** **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 |

**319.** **Why Reactjs Is Used?**

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

**320.** **Does Reactjs Use Html?**

No, It uses JSX which is simiar to HTM.

**321.** **When Reactjs Released?**

March 2013

**322.** **What Is Current Stable Version Of Reactjs?**

Version: 15.5 Release on: April 7, 2017

**323.** **What Are The Life Cycle Of Reactjs?**

Initialization State/Property Updates Destruction

**324.      What Are The Feature Of Reactjs?**

JSX: JSX is JavaScript syntax extension. Components : React is all about components. One direction flow: React implements one way data flow which makes it easy to reason about your app

**325.      What Are The Advantages Of Reactjs?**

React uses virtual DOM which is JavaScript object. This will improve apps performance It can be used on client and server side Component and Data patterns improve readability. Can be used with other framework also.

**326.      How To Embed Two Components In One Component?**

```jsx
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>
        );
    }
}
```

**327.      What Are The Advantages Of Using Reactjs?**

Advantages of ReactJS:
React uses virtual DOM which is JavaScript object. This improves application performance as JavaScript virtual DOM is faster than the regular DOM.

React can be used on client and as well as server side too.

Using React increases readability and makes maintainability easier. Component, Data patterns improves readability and thus makes it easier for manitaing larger apps.

React can be used with any other framework (Backbone.js, Angular.js) as it is only a view layer. React's JSX makes it easier to read the code of our component.

It's really very easy to see the layout. How components are interacting, plugged and combined with each other in app.

## 328.    What Are The Limitations Of Reactjs?

Limitations of ReactJS:

React is only for view layer of the app so we still need the help of other technologies to get a complete tooling set for development.

React is using inline templating and JSX.

This can seem awkward to some developers. The library of react is too large. Learning curve for ReactJS may be steep.

## 329.    How To Use Forms In Reactjs?

In React's virtual DOM, HTML Input element presents an interesting problem.

With the others DOM environment, we can render the input or textarea and thus allows the browser maintain its state that is (its value).

we can then get and set the value implicitly with the DOM API.

In HTML, form elements such as <input>, <textarea>, and <select> itself maintain their own state and update its state based on the input provided by user.

In React, components' mutable state is handled by the state property and is only updated by **setState().** HTML <input> and <textarea> components use the value attribute. HTML <input> checkbox and radio components, checked attribute is used. <option> (within <select>) components, selected attribute is used for select box.

## 330.    How To Use Events In Reactjs?

React identifies every events so that it must have common and consistent behavior across all the browsers.

Normally, in normal JavaScript or other frameworks, the onchange event is triggered after we have typed something into a Textfield and then "exited out of it".

In ReactJS we cannot do it in this way. The explanation is typical and non-trivial: *"<input type="text" value="dataValue"> renders an input textbox initialized with the value, "dataValue".

When the user changes the input in text field, the node's value property will update and change. However, node.getAttribute('value') will still return the value used at initialization time that is dataValue.

**Form Events:**
onChange: onChange event watches input changes and update state accordingly. onInput: It is triggered on input data onSubmit: It is triggered on submit button.

**Mouse Events:**
onClick: OnClick of any components event is triggered on. onDoubleClick: onDoubleClick of any components event is triggered on. onMouseMove: onMouseMove of any components, panel event is triggered on. onMouseOver: onMouseOver of any components, panel, divs event is triggered on.

**Touch Events:**
onTouchCancel: This event is for canceling an events. onTouchEnd: Time Duration attached to touch of a screen. onTouchMove: Move during touch device . onTouchStart: On touching a device event is generated.

**331.     Give An Example Of Using Events?**

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

var StepCounter = React.createClass(
    {
        getInitialState: function () {
            return { counter: this.props.initialCounter };
        },

        handleClick: function () {
            this.setState({ counter: this.state.counter + 1 });
        },
```

```
        render: function () {
            return <div onClick={this.handleClick}>
                OnClick Event, Click Here: {this.state.counter}
</div>;
        }
    });
ReactDOM.render(< StepCounter initialCounter={7} />, document.g
etElementById('content'));
```

### 332. Explain Various Flux Elements Including Action, Dispatcher, Store And View?

Flux can be better explained by defining its individual components: **Actions** – They are helper methods that facilitate passing data to the Dispatcher.
**Dispatcher** – It is Central hub of app, it receives actions and broadcasts payloads to registered callbacks.
**Stores** – It is said to be Containers for application state & logic that have callbacks registered to the dispatcher. Every store maintains particular state and it will update when it is needed. It wakes up on a relevant dispatch to retrieve the requested data. It is accomplished by registering with the dispatcher when constructed. They are similar to model in a traditional MVC (Model View Controller), but they manage the state of many objects — it does not represent a single record of data like ORM models do. **Controller Views** – React Components grabs the state from Stores and pass it down through props to child components to view to render application.

### 333. What Is Flux Concept In Reactjs?

Flux is the architecture of an application that Facebook uses for developing client side web applications. Facebook uses internally when working with React. It is not a framework or a library. This is simply a new technique that complements React and the concept of Unidirectional Data Flow.
Facebook dispatcher library is a sort of global pub/sub handler technique which broadcasts payloads to registered callbacks.

### 334. Give an Example Of Both Stateless And Stateful Components With Source Code?

Stateless and Stateful components Stateless: When a component is "stateless", it calculates state is calculated internally but it directly never mutates it.

With the same inputs, it will always produce the same output. It means it has no knowledge of the past, current or future state changes.

```javascript
var React = require('react');
var Header = React.createClass({
    render: function () {
        return (<img src={this.props.imageSource} />);
    }
}
);
ReactDOM.render(<Header imageSource="myImage.png" />, document.
body);
```

**Stateful** : When a component is "stateful", it is a central point that stores every information in memory about the app/component's state, do has the ability to change it.

It has knowledge of past, current and potential future state changes. Stateful component change the state, using **this.setState** method.

```javascript
var React = require('react');

var Header = React.createClass({
    getInitialState: function () {
        return { imageSource: "header.png" };
    },

    changeImage: function () {
        this.setState({ imageSource: "changeheader.png" });
    },

    render: function () {
        return (<img src={this.state.imageSource} onClick={this
.changeImage.bind(this)} />);
    }
});

module.exports = Header;
```

**335.** **Explain Basic Code Snippet Of Jsx With The Help Of A Practical Example?**

Your browsers does not understand JSX code natively, we need to convert it to JavaScript first which can be understand by our browsers. We have aplugin which handles including Babel 5's in-browser ES6 and JSX transformer called browser.js.

Babel will understand and recognize JSX code in <script type="text/babel"></script> tags and transform/convert it to normal JavaScript code. In case of production we will need to pre-compile our JSX code into JS before deploying to production environment so that our app renders faster.

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <title>My First React JSX Example</title>
</head>

<body>
    <div id="hello-world"></div>
    <script src="https://fb.me/react-15.0.0.js"></script>
    <script src="https://fb.me/react-dom-15.0.0.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
 core/5.8.34/browser.min.js"></script>
    <script
       type="text/babel"> var HelloWorld = React.createClass({ render: f
unction () { return (<p>Hello, World</p>) } }); ReactDOM.render(<HelloWor
ld />, document.getElementById('hello-world'));</script>
</body>

</html>
```

**336.** **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.

### 337. What Is Reactjs-jsx?

JSX (JavaScript XML), lets us to build DOM nodes with HTML-like syntax. JSX is a preprocessor step which adds XML syntax to JavaScript. Like XML, JSX tags have a tag name, attributes, and children JSX also has the same.
If an attribute/property value is enclosed in quotes(""), the value is said to be string. Otherwise, wrap the value in braces and the value is the enclosed JavaScript expression. We can represent JSX as <HelloWorld/>.

### 338. 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.

### 339. How To Apply Validation On Props In Reactjs?

When the application is running in development mode, React will automatically check for all props that we set on components to make sure they must right correct and right data type.
For instance, if we say a component has a Message prop which is a string and is required, React will automatically check and warn if it gets invalid string or number or boolean objects.

For performance reasons this check is only done on dev environments and on production it is disabled so that rendering of objects is done in fast manner . Warning messages are generated easily using a set of predefined options such as:

**React.PropTypes.string**
**React.PropTypes.number**
**React.PropTypes.func**
**React.PropTypes.node**
**React.PropTypes.bool**

## 340.     What Are State And Props In Reactjs?

State is the place where the data comes from. We must follow approach to make our state as simple as possible and minimize number of stateful components. For example, ten components that need data from the state, we should create one container component that will keep the state for all of them.
The state starts with a default value and when a Component mounts and then suffers from mutations in time (basically generated from user events). A Component manages its own state internally, but—besides setting an initial state—has no business fiddling with the stateof its children.
You could say the state is private.

```jsx
import React from 'react';
import ReactDOM from 'react-dom';

var StepCounter = React.createClass({
    getInitialState: function () {
        return { counter: this.props.initialCount };
    },

    handleClick: function () {
        this.setState({ counter: this.state.counter + 1 });
    },

    render: function () {
        return <div onClick={this.handleClick}>{this.state.counter}</div>;
    }
});
```

```
ReactDOM.render(< StepCounter initialCount={7} />, document.get
ElementById('content'));
```

**Props**: They are immutable, this is why container component should define state that can be updated and changed. It is used to pass data down from our view controller(our top level component). When we need immutable data in our component we can just add props to reactDOM.render() function.

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

class PropsApp extends React.Component {
    render() {
        return (
            <div>
                <h1>{this.props.headerProperty}</h1>
                <h2>{this.props.contentProperty}</h2>
            </div>
        );
    }
}
ReactDOM.render(
    <PropsApp headerProperty="Header from props..." contentProp
erty="Content from props..." />,
    document.getElementById('app'));
```

### 341. What Is The Difference Between The State And Props In Reactjs?

**Props**: Passes in from parent component.<PropsApp headerProperty = "Header from props…" contentProperty = "Content from props…"/>
These properties are being read by PropsApp component and sent to ReactDOM View.

**State**: Created inside component by getInitialState. this. state reads the property of component and update its value it by this. setState() method and then returns to ReactDOM view.State is private within the component.

### 342. List some advantages of ReactJS ?

Advantages of React Js

- React.js is extremely efficient: React.js creates its own virtual DOM where your components actually live. This approach gives you enormous flexibility and amazing gain in performance. React.js also calculates what are the changes needed to be made in DOM. This process of React.js avoids expensive DOM operations and make updates in a very client manner.
- It makes writing Javascript easier: React.js uses a special syntax called JSX, which allows you to mix HTML with Javascript. The user can drop a bit of HTML in the render function without having to concatenate strings, this is another fantastic thing. React.js turns those bits of HTML into functions with a special JSXTransformer.
- It gives you out-of-the-box developer tools: When you start your journey with React.js, do not forget to install official React.js chrome extension. It makes debugging your application much easier. After you install the extension, you will have a direct look into the virtual DOM as if you were browsing a regular DOM tree in the elements panel. Isn't it pretty amazing!
- It's awesome for SEO: One of the biggest problems with other Javascript frameworks is that they do not search engine friendly. Though there have been some improvements in this area, search engines generally have trouble reading Javascript heavy applications. React.js stands out from the crowd because you can run React.js on the server, and the virtual DOM will be rendered to the browser as a regular web page.
- UI Test Cases: It is extremely easy to write UI test cases because the virtual DOM system implemented entirely in JS.

## 343.    What are Components in ReactJS?

React Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.
Below is ample component written in ES6 class to display a welcome message on the screen.

```
class Welcome extends React.Component {
```

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

const element = <Welcome name="Sara" />;
ReactDOM.render(element, document.getElementById('root'));
```

## 344.    What is JSX?

JSX is an XML/HTML-like syntax used by React that extends ECMAScript so that XML/HTML-like text can co-exist with JavaScript/React code.

The syntax is intended to be used by preprocessors (i.e., transpilers like Babel) to transform HTML-like text found in JavaScript files into standard JavaScript objects that a JavaScript engine will parse.

Basically, by using JSX you can write concise HTML/XML-like structures (e.g., DOM like tree structures) in the same file as you write JavaScript code, then Babel will transform these expressions into actual JavaScript code.
Unlike the past, instead of putting JavaScript into HTML, JSX allows us to put HTML into JavaScript.

By using JSX one can write the following JSX/JavaScript code:

```
var nav = (
    <ul id="nav">
        <li><a href="#">Home</li>
        <li><a href="#">About </a></li>
        <li><a href="#">Clients </a></li>
        <li><a href="#">Contact Us</a> </li>
</ul> );
```

And Babel will transform it into this:

```
var nav = React.createElement("ul", { id: "nav" },
    React.createElement("li", null, React.createElement("a", {
href: "#" }, "Home")),
```

```
    React.createElement("li", null, React.createElement("a", {
href: "#" }, "About")),
    React.createElement("li", null, React.createElement("a", {
href: "#" }, "Clients")),
    React.createElement("li", null, React.createElement("a", {
href: "#" }, "Contact Us")));
```

**345.    Explain life Cycle of React JS Component ?**

React JS Component Lifecycle Each component has several "lifecycle methods" that you can override to run code at particular times in the process. Methods prefixed with will are called right before something happens, and methods prefixed with did are called right after something happens.

**Mounting**

These methods are called when an instance of a component is being created and inserted into the DOM:

- constructor()
- componentWillMount()
- render()
- componentDidMount()

**Updating**

An update can be caused by changes to props or state. These methods are called when a component is being re-

**rendered**:

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

**Unmounting**

This method is called when a component is being removed from the DOM:

- componentWillUnmount()

**Other APIs**

Each component also provides some other APIs:

- setState()

- forceUpdate()

**Class Properties**

- defaultProps

- displayName

**Instance Properties**

- props
- state

### 346.    List some features of ReactJS?

Undoubtedly today React is among of one the best JavaScript UI frameworks. It comes with a lot of features that helps programmers to create beautiful application easily, we have listed some of them below.
- It's Adaptability
- Free and Open Source
- Decorators from ES7
- Server-side Communication
- Asynchronous Functions & Generators
- Flux Library
- Destructuring Assignments
- Usefulness of JSX

### 347.    How to use Events in ReactJS?

Using Events is React js is very similar to handling event on DOM elements.The difference is only in syntax like.

1. The name of event is React js is always in camelCase.

2. With JSX you pass a function as the event handler, rather than a string.

Let's understand by an example:

// In HTML

**<button onclick="activateAccount()"> Activate Account </button>**

//In React
**<button onClick={activateAccount}> Activate Account </button>**

Another difference is that in React js you cannot return false to prevent default behavior in React. You must call preventDefault explicitly.

## 348.      What is flux in JavaScript?

Flux is an application architecture for creating data layers in JavaScript applications. It was designed at Facebook along with the React view library. Flux is not a framework or a library. It is simply a new kind of architecture that complements React and the concept of Unidirectional Data Flow.

## 349.      What are refs in React? When to use it.

In React ref is used to store the reference of element or component returned by the component render() configuration function.Refs should be avoided in most cases, however, they can be useful when we need DOM measurements or to add methods to the components.
Refs can be used in the following cases
   • Managing focus, text selection, or media playback.
   • Triggering imperative animations.
   • Integrating with third-party DOM libraries.

## 350.      What are stateless components in React?

Stateless components are components that don't have any state. When something is stateless, it calculates its internal state but it never directly mutates it. For creating a stateless component No class and this keyword is needed. You can create a stateless components using plain functions or Es6 arrow function. Below is an example of stateless component in react.
//In Es6
 **const Pane = (props) => {props.children} ;**

//In Es5

**const Username = ({ username }) => The logged in user is: {username}**

351.     What is the difference between State and props in ReactJs?

**Props** are shorthand for properties. they are very similar to an argument is passed to a pure javascript function. Props of the component are passed from parent component which invokes component.

During a component's life cycle props should not change consider them as immutable. In React all props can be accessible with this.props.

```
import React from 'react';
class Welcome extends React.Component {
    render() {
        return <h1>Hello {this.props.name}</h1>;
    }
}
const element = ;
```

**State** are used to create dynamic and interactive components in React.State is heart of react component that makes it alive and determines how a component renders & behaves. // simple state example

```
import React from 'react';

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

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

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

```
    }
}

export default Button;
```

352.    What are Synthetic events?

SyntheticEvent is a cross-browser wrapper around browser's native event. In React all of your event handlers will be passed instances of SyntheticEvent.
The synthetic event works the same way as the event system of browsers, the only difference is that the same code will work across all browsers. Below is a simple example of how to listen for a click event in react

```
import React, { Component } from 'react';

class ShowAlert extends Component {
    showAlert() { alert("Im an alert"); }

    render() {
        return (
            <button onClick={this.showAlert}>show alert </button>);
    }
}

export default ShowAlert;
```

353.    **What is the difference between Dom and virtual Dom in React js?**

**DOM** is the acronym for Document Object Model. Dom is also called HTML DOM as it is an abstraction of structured code called HTML for web developers. Dom and HTML code are interrelated as the elements of HTML are known as nodes of DOM. It defines a structure where users can create, alter, modify documents and the content present in it. So while HTML is a text, DOM is an in-memory representation of this text.

**Virtual DOM** is an abstraction of abstraction as it has been derived from HTML DOM. It is a representation of DOM objects like a lightweight copy. The virtual DOM was not invented by React, it is only used and provided for free.

354.     Enlist the advantages and disadvantages of React js?

React.js is used by web developers for creating large web pages without reloading the entire page. It uses the data and can be changed over time.
The following are the advantages of using React.js-
1- React makes Search engine optimization (SEO) easy.
2- It is very efficient as it ensures readability and easy maintenance.
3- It gives extraordinary developer tools to web developers and makes Java coding easier for them.
4- UI test cases.

The following are the disadvantages of React
1- Some major configurations are required for integrating React js with traditional MVC framework such as substituting erb with React js.
2- It is a steep learning process for people who are new to web development world.

**355.     What are the controlled components and uncontrolled components in React?**

**Controlled component** is more advisable to use as it is easier to implement forms in it. In this, form data are handled by React components. A controlled input accepts values as props and callbacks to change that value.
The **uncontrolled component** is a substitute for controlled components. Here form data is handled by DOM itself. In uncomfortable components, the ref can be used to get the form values from DOM.

**356.     Explain the difference between functional and class components.**

**Functional components** are those components that returns React elements as a result. These are just simple old JavaScript functions. React 0.14 has given a new shortcut for creating simple stateless components that are known as functional components. These components make use of easy JavaScript functions.

**Class components** – most of the tech-savvy people are more familiar with class components as they have been around the corner for a longer time. These components make use of plain old java objects for creating pages, mixins, etc in an identical way. Using React's create a class factory method, a literal is passed in defining the methods of a new component.

**357.     What do you understand by mixin or higher order components in React?**

**Higher order components (HOC)** is a function that takes component as well as returns a component. It is a modern technique in React that reuses the component logic. However, Higher order components are not a part of React API, per se. These are patterns that emerge from React's compositional nature. In other words, HOC's are functions that loop over and applies a function to every element in an array.

358.     How is React different from angular and VUE js?

**Angular Js** – developed by Google, angular is a typescript based JavaScript application framework. It is also known as Super-heroic JavaScript MVW Framework. It was developed with the motive to encounter the challenges of creating single page applications. There are several versions of angular such as Angular 2+, Angular 2 or ng2. Angular is the rewritten, mostly incompatible successor to AngularJS which means AngularJS is the oldest framework.

**React**– React was developed by Facebook in March 2013. It is a JavaScript library that is used for building user interfaces. React creates large web applications and also provides speed, scalability, and simplicity.

**Vue Js**- Launched in February 2014, Vue is the most famous and rapidly growing framework in JS field. Vue is an intuitive, fast and composable MVVM for building interactive interfaces. It is extremely adaptable and several JavaScript libraries make use of this. Vue is also a web application framework that helps in making advanced single page applications.

**359.     Open source front-end JavaScript library for generation Single Page Applications and Stateful and Stateless?**

React

**360.      Can you bind JavaScript with XML in which front-end applications using which scripting language?**

JSX (JavaScript and XML) in React

**361.      Display or Create DOM properties in Reacts ?**

Element

**362.      Which function is used to display a DOM?**

ReactDOM.render()

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

A Youngster class constructor can't make use of this orientation until super () method has been called. The same applies for ES6 sub classes. The reason of passing props parameter to super () call is to access this props in your child constructors.

**364.      What is reconciliation?**

When a components props or state change, React decides whether an real DOM update is necessary by comparing the newly returned element with the before rendered one. When they not equal, React will inform the DOM. This process is called reconciliation.

**365.      Why React uses class Name over class attribute?**

Class is the keyword in JavaScript,JSX is an extension of JavaScript. That is the principal reason why React uses class Name instead of class. Pass a string as the class Name prop.

**366.      What are fragments?**

Its mutual pattern in React which is used for a component to return multiple elements. Fragments let group a list of children without adding extra nodes to the DOM.

### 367.      What are portals in React?

Portal is a suggested way to render children into a DOM node that exists outside the DOM order of the parent component.

### 368.      What are stateless components?

If behavior is independent of its state then it can be a displaced component. You can use either function or class for creating stateless components. But unless we need to use a lifecycle hook in our components, we should go for function components. There are a lot of benefits if decide to use function components here, they are easy to write, understand, and test, a little faster, and Every one can avoid the this keyword altogether.

### 369.      What are stateful components?

If the behavior of a component is reliant on the state of the component then it can be termed as stateful component. These stateful components are always class components and have a state that gets modified in the constructor.

### 370.      What is the use of react-dom package?

The react-dom package provides DOM-specific methods that can be used at the highest level of our app. Most of components are not required to use this module. Some of the methods of this package are:
render()
hydrate()
unmountComponentAtNode()
findDOMNode()
createPortal()

### 371.      What is React DOM Server?

The React DOM Server object enables us to render components to static markup . This object is mostly used for server-side rendering (SSR). The below methods can be used in both the server and browser environments:
renderToString()

renderToStaticMarkup

**372.      What will happen if you use set State() in constructor?**

When you use set State(), then apart from assigning to the object state React also re renders the component and all its children. We would get error like this: Can only update a mounted or mounting component. Hence we need to use this. stateto initialize variables inside constructor.

**373.      What will happen if you use props in initial state?**

If the props on the component are changed without the component being revitalized, the new prop value will never be displayed because the constructor function will never update the current state of the component. The starting of state from props only runs when the component is first created.

**374.      What is the lifecycle methods order in mounting?**

The lifecycle methods are called in the below order when an instance of a component is being created and inserted into the DOM. constructor() static getDerivedStateFromProps() render() componentDidMount()

**375.      How to re-render the view when the browser is resized?**

We can listen to the resize event in component Did Mount() and then update the dimensions (width and height). we should remove the listener in component Will Unmounts() method.

**376.      What is the difference between setState() and replaceState() methods?**

When we use setState() the current and previous states are merged. replaceState() throws out the current state, and replaces it with only what we provide. Usually setState() is used unless we really need to remove all previous keys for some reason. we can also set state to false/null in setState() instead of using replaceState().

**377.      How can find the version of React at runtime in the browser?**

we can use React.version to get the version. const REACT_VERSION = React.version ReactDOM.render( <div>{`React version: ${REACT_VERSION}`}</div>, document.getElementById('app'))

**378.      How to use https instead of http in create-react-app?**

We just need to use HTTPS=true configuration. we can edit our package.json scripts section: "scripts": { "start": "set HTTPS=true && react-scripts start"}

### 379.   Why is a component constructor called only once?

React's appeasement algorithm assumes that without any information to the contrary, if a custom component appears in the same place on consequent renders, it's the same component as before, so reuses the previous instance moderately than making a new one.

### 380.   What are the popular packages for animation?

React Motion and React Transition Group are popular animation packages in React ecosystem.

### 381.   What is React Router?

React Router is a authoritative routing library built on top of React that helps you add new screens and flows to your application incredibly quickly, all while keeping the URL in sync with what's being displayed on the page.

### 382.   What is the purpose of push() and replace() methods of history?

History instance has two methods for navigation purpose. push() replace() If we think of the history as an array of visited locations, push() will add a new location to the array and replace() will replace the current location in the array with the new one.

### 383.   How to pass params to history.push method in React Router v4?

While navigating we can pass props to the history object:
this.props.history.push({pathname:'/template',search:'?name=sudheer',state: { detail:response.data }})

### 384.   What is the purpose of ReactTestUtils package?

ReactTestUtils are providing in the with-addons package and allowto perform actions against a simulated DOM for the purpose of unit testing.

### 385.   What Are The Limitations Of Reactjs?

React is for view layer of the app so we still need the help of others to get a complete tooling set for development. It Is using parallel tempting and JSX. It can seem difficult

to some developers. The library is (React) too large. Its (React) Learning curve may be steep.

### 386.    What Is Flux Concept In Reactjs

It is the architecture of an application that Facebook uses for developing client side web applications. Facebook uses within when working with React. React is not a library or a framework. This is just a new technique that complements React and the idea of Uni-directional Data Flow.

### 387.    What is Jest?

Jest – JavaScript unit testing framework created by Facebook based on Jasmine and provides automated mock creation and a jsdom environment. It's often used for testing components.

### 388.    What is flux?

Flux is designed paradigm used as an alternate for the more traditional MVC pattern. It is not a framework or a library but a new kind of architecture that complements React and the concept of Unidirectional Data Flow. Facebook use this pattern inside when working with respond.

### 389.    What is Redux?

Redux is an expected state container for JavaScript apps based on the Flux design pattern. Redux can be used mutually with React, or with any other view library. It is a little size (about 2kB) and has no dependency.

### 390.    What is the difference between React context and React Redux?

We can use Context in our application directly and is going to be great for passing down data to acutely nested components which what it was designed for. Whereas Redux is much more potent and provides a large number of skin textures that the Context API doesn't provide. Also, React Redux uses context internally but it doesn't render this fact in the public API.

### 391.    What is the difference between React Native and React?

React -JavaScript library, supporting both front-end web and being run on the server, for building user interfaces and web applications. React Native – mobile framework that compiles to native app components, allowing you to build native mobile

applications (iOS, Android, and Windows) in JavaScript that allows you to use React to build your components, and implements React under the hood.

## 392. What is JSX?

JSX is a XML like syntax addition to ECMAScript (the acronym stands for JavaScript XML). mainly it provides syntactic sugar for the React.createElement() function, giving us articulacy of JavaScript along with HTML like template syntax.

In the example below text inside <h1> tag return as JavaScript function to the render function.

```
class App extends React.Component {
    render() {
        return (<div><h1>{'Welcome to React world!'}</h1></div>
)
    }
}
```

## 393. What is the difference between Element and Component?

Element – plain object describing what we want to appear on the screen in terms of the DOM nodes / other components. Elements can contain other Elements in their props.

Creating a React element is cheap. Once an element is created, it is never mutated. The object representation of React Element as follows:

**Const element=React.createElement('div', {id:'login-btn'},'Login')**

The above React.createElement() function returns an object: {type: 'div',props: {children: 'Login',id: 'login-btn' }}

And last renders to the DOM using ReactDOM.
render(): <divid='login-btn'>Login</div> component – declared in several ways.

Component can class with a render() method.

Alternatively it can be defined as a function. Component takes props as an input, and returns an JSX tree as the output:

Const Button= ({ onLogin }) =><div id={'login-btn'} onClick={onLogin} />

Then JSX gets transpiled to React.createElement() function tree:

const Button= ({ onLogin }) =>React.createElement('div',{ id:'login-btn', onClick:onLogin },'Login')

### 394. How to create components in React?

There are 2 ways to create a component. Function Components:This is the one way to create a component.
Function components are JavaScript functions that accept props object as first parameter and return React elements:

```
functionGreeting({ message }) { return < h1 > {`Hello, ${message}`}</h1 > ?}
```

Class Components: we can also use ES6 class to define a component.The above function component can be written as below:

```
class Greeting extends React.Component {
    render() { vii.return < h1 > {`Hello, ${this.props.message}` }</h1 >
 }}
```

### 395. Characterize ReactJs and Its highlights.

ReactJs is a front-end JavaScript library and gives a part based methodology that causes fashioners to manufacture reusable segments. Respond has following recorded highlights:
• React utilizes virtual DOM as opposed to genuine DOM,
• React utilizes Server-side rendering,
• Uni-directional information stream or information restricting is utilized in React.

### 396. For what reason would it be advisable for us to utilize React, tell its points of interest as well?

Respond is a standout amongst the most utilized libraries and has following recorded worthwhile highlights:

- Application execution is expanded by React
- It can be utilized on customer andserver-sidee
- Code coherence is expanded due to JSX
- The client can without much of a stretch coordinate React with different systems like Angular, Meteor, and so forth.
- UI test cases can be effectively composed by utilizing React

### 397.    Enroll React constraints on the off chance that you know any?

In spite of the fact that React is a well-known apparatus however has a few impediments that are recorded underneath: • React isn't a structure yet a lot of libraries. • Designers set aside the opportunity to comprehend this library as it is substantial in size • For fledgling software engineers, it very well may be hard to comprehend this library • Inline templating and JSX makes the coding to some degree complex

### 398.    Clarify Virtual DOM of React and how can it function?

Virtual DOM is only it is only a duplicate of genuine DOM. In the hub tree of Virtual DOM, all components, qualities, and substance are recorded as articles and its properties. Through render work, a hub tree is made by the React segments. After transformation, the tree gets refreshed and it might occur because of framework client's activity.

### 399.    Separate among React and Angular.

Respond and Angular are two of the most prominent libraries for JavaScript and let us have a speedy take a gander at contrasts between them:
- Angular can actualize finish MVC, while through React we can just execute View of MVC.
- Angular gives customer side rendering while React give server-side rendering.
- Angular can give two-way information official while React can give just a single way information authoritative.
- Angular is given by the Google and React is given by the Facebook.
- Angular utilizes run-time troubleshooting and React utilizes aggregate time investigating.
- Angular utilizes Real Dom while React utilizes virtual DOM.

### 400.    What is the importance of the announcement 'In React, everything is part'?

The building squares of the React application UI are called parts. Whole UI is separated into a few parts that are littler in size and are reusable also. At that point every one of these parts is rendered freely without influencing rest of the UI.

### 401.  Characterize Props.

In React, Props is utilized for Properties. These are permanent parts of React that must be kept unadulterated. These parts are constantly passed to the youngster segments from parent segments and can't be sent back to the parent segment. It makes the information stream unidirectional and powerfully created information is rendered through this idea.

### 402.  Think about props and state.

Following are a couple of recognizable focuses for a brisk correlation of prop and state:
• Both Props and State get beginning an incentive from the parent part
• In State, the parent segment can change the esteem, while if there should be an occurrence of props it isn't so.
• Default qualities can be set in both state and prop
• Components can be changed from inside if there should be an occurrence of State while this is impossible in Props
• The starting worth can be set for tyke segment in both of the cases

### 403.  By what method can the segment state be refreshed?

For this, we can utilize this.setState() work in the accompanying way:

```
class newComponent extands React.Component {

    Constructor(){
        Super();
        this.state = { name: 'Maxx', id: '101' }
    }

    render() {
```

```
      setTimeout(() => { this.setState({ name: 'Denis', id: '
222' }) }, 2000)

      return(<div> Welcome {this.state, name} What is id {thi
s.state.id} </div>);
    }
}


ReatDOM.render(<newComponent />, document.getElementById('conte
nt'))
```

**404.       Think about stateful and stateless segments.**

Following are the fundamental impressive contrasts among stateful and stateless segments:
• In the two cases, the data about the difference in conditions of the segment is put away
• Stateless parts have the specialist to change the satiate while stateful segments don't have such expert
• Stateless segments store the data about past, present, and future states, while stateful segments don't store such data about the parts.

**405.       What are the periods of the lifecycle of React's segment?**

In the lifecycle of React's segment, there are following three stages:
• Initial Phase: Initial stage is the stage when the segments begin a mind-blowing voyage to advance toward DOM
• Updating Phase: At when parts are added to DOM and refresh and rendered when a state or prop changes happen.
• Unmounting Phase: This is considered last period of the part in which the segment gets annihilated.

**406.       What are the techniques for React segment lifecycle?**

In the whole lifecycle of a React part, the accompanying strategies are utilized to achieve the capacities:
   ❖ **componentWillMount()** – On customer and server side this capacity gets executed just before the rendering
   ❖ **componentDidMount()** – After first render it gets executed on the customer side

- ❖ **componentWillReceiveProps()** – This capacity is summoned when the props are gotten from the parent class and another render isn't being called.
- ❖ **shouldComponentUpdate()** – This Boolean capacity returns genuine or false according to circumstance like in the event that the part should be refreshed, genuine is returned else false is returned
- ❖ **thecomponentWillUpdate()** – It is called when rendering isn't being called
- ❖ **componentDidUpdate()** – It is called soon after when render work is called
- ❖ **componentwillUnmount()** – When a part gets un-mounted from DOM then this capacity is called.

## 407.      What is the state in react js?

it holds the information about the component and the state is using only inside the component. A state is mutable. By default, a component has no state.

## 408.      What are props in react js?

Props mean properties, It's a way of passing data from parent to child and we can say that props are just a communication channel between components. Always moving from parent to child component. It's immutable and read-only.

## 409.      What is different b/w super() and super(props)?

Only one reason for using props parameter in the super keyword. When you want to access this.props inside the constructor then you need to pass the props parameter in the super keyword and no difference outside the constructor.

## 410.      What is the super keyword in react?

The super keyword is used to access & call functions on an object's parent.

## 411.      Real Disadvantages of utilizing React.

- ❖ It is covering for 'View' layer in MVC (Model-View-Controller).
- ❖ React is only a JavaScript library, Not a system.
- ❖ Its library is vast and sets aside opportunity to get it.
- ❖ it utilizes inline templating and JSX.

## 412.      What are the periods of the lifecycle of React's part?

In the lifecycle of React's part, there are following three stages:

- ❖ Initial Phase: Initial stage is the stage when the parts begin an incredible voyage to advance toward DOM
- ❖ Updating Phase: At when parts are added to DOM and refresh and rendered when a state or prop changes happen.
- ❖ Unmounting Phase: This is known as the last period of the segment in which the part gets wrecked.

### 413. What are the strategies for React part lifecycle?

In the whole lifecycle of a React part, the accompanying strategies are utilized to achieve the capacities:

- ❖ **componentWillMount()** – On customer and server side this capacity gets executed just before the rendering
- ❖ **componentDidMount()** – After first render it gets executed on the customer side
- ❖ **componentWillReceiveProps()** – This capacity is summoned when the props are gotten from the parent class and another render isn't being called.
- ❖ **shouldComponentUpdate()** – This Boolean capacity returns genuine or false according to circumstance like on the off chance that the segment should be refreshed, genuine is returned else false is returned
- ❖ **thecomponentWillUpdate()** – It is called when rendering isn't being called
- ❖ **componentDidUpdate()** – It is called soon after when render work is called
- ❖ segments **willUnmount()** – When a segment gets un-mounted from DOM then this capacity is called.

### 414. What is yield catchphrase in JavaScript ?

The yield catchphrase is utilized to delay and resume a generator work. That is known as yield catchphrase.

### 415. What are the means to Setup React Environment?

1. Above all else Make beyond any doubt you have to introduce late form of Node.js. You simply need Node >= 6 on your machine.
2. Introduce hub in your framework (you can download from here
https://nodejs.org/en/download/)
3. Pick your task organizer and open cmd direction.
4. Write in direction :- npm introduce make respond application
5. After entire process again compose the :- make respond application your venture name

6. Proceed to the venture area utilizing direction :- disc your-venture name

7. Presently you can run the respond application utilizing direction (Start the improvement server) :- npm begin

8. On the off chance that you need to make packages the application into static records for generation :- num run manufacture

npm introduce – g make respond application

make respond application your-application name

cd your-application name

npm begin

You may likewise utilize npx

npx make respond application your-application name

cd your-application name

npm begin

### 416.      What is Presentational segment

A presentational part is a segment that renders HTML as it were. The segment's capacity is presentational markup.

In term of Redux application, a presentational part does not connect with the Redux store. The presentational segment acknowledges props from a compartment part. The compartment segment indicates the information a presentational segment should render. The compartment part has individualized conduct.

### 417.      What are the Component uses and features?

1. Components provide strong encapsulation for reusable components.

2. Components provide a strong declarative library that keeps the DOM in sync with your data.

### 418.      What are the kinds of information that control a segment in React?

There are two sorts of information that control a part: State and Props in responding. Props are set by the parent and they are settled all through the lifetime of a part. State information that will change, we need to utilize state.

### 419.      Clarify Components lifecycle in ReactJS?

**Constructor(or getInitialState)** :- Constructor is utilized to set beginning state for segment.

**componentWillMount**:- componentWillMount strategy will be called before part render to a program and furthermore it is customer side and also server-side rendering.

**Render**:- Render strategy you will see part on Browser and this technique ought to be unadulterated. You can not refresh state in this strategy

**componentDidMount**:- This technique will be called directly after the render strategy. refreshing state in this strategy will re-render part. It's a customer side rendering.

**componentWillReceiveProps**:- this technique is called at whatever point segment gets new props. shouldComponentUpdate:- In this technique, you can check whether re-rendering the part is fundamental or not. return false on the off chance that you would prefer not to re-render part.

a segment will **refresh**:- this strategy will be called after shouldComponentUpdate(only in the event that it returns genuine). Also, before rendering the render strategy.

**render**:- In this technique refreshed segment will be rendered to screen. with new data(or changes).

**componentDidUpdate**:- componentDidUpdate will be called after render technique. a segment willUnmount:- componentWillUnmount crushed and expelled from the DOM and just a single lifecycle strategy is dynamic.

### 420.        What are the center standards of Redux?

Redux pursues three key standards:

❖ **Single wellspring of truth:** The condition of your entire application is put away in a protest tree inside a solitary store. The single state tree makes it less demanding to monitor changes after some time and troubleshoot or assess the application.

❖ **A state is perused just**: The best way to change the state is to produce an activity, a question portraying what occurred. This guarantees neither the perspectives nor the system callbacks will ever compose straightforwardly to the state.

❖ **Changes are made with unadulterated capacities**: To determine how the state tree is changed by activities, you compose reducers. Reducers are simply unadulterated capacities that make the past state and move as parameters and restore the following state.

### 421.        What are the drawbacks of Redux contrasted with Flux?

Rather than saying drawbacks we can state that there are few bargains of utilizing Redux over Flux. Those are as per the following:

❖ **You should figure out how to maintain a strategic distance from transformations:** Flux is un-obstinate about changing information, however Redux doesn't care for transformations and numerous bundles correlative to Redux accept you never change the state. You can uphold this with dev-just bundles like redux unchanging state-invariant, Immutable.js, or educating your group to compose non-changing code.

❖ **You will need to deliberately pick your bundles:** While Flux expressly doesn't attempt to take care of issues, for example, fix/re-try, industriousness, or structures, Redux has expansion focuses, for example, middleware and store enhancers, and it has generated a rich environment.

❖ **There is no pleasant Flow incorporation yet:** Flux as of now gives you a chance to do extremely amazing static sort checks which Redux doesn't bolster yet.

## 422. Would i be able to dispatch an activity in reducer?

Dispatching an activity inside a reducer is an enemy of example. Your reducer ought to be without reactions, just processing the activity payload and restoring another state protest. Including audience members and dispatching activities inside the reducer can prompt binded activities and opposite symptoms.

## 423. How to get to Redux store outside a part?

Truly. You simply need to send out the store from the module where it made with createStore(). Additionally, it shouldn't dirty the worldwide window question. store = createStore(myReducer)

## 424. What are the drawbacks of MVW pattern?

vii. The DOM manipulation is very expensive which causes applications behaves slowly and inefficient.
viii. Due to circular dependencies, a complicated model was created around models and views.
1. A lot of data changes happens for collaborative applications(like Google Docs).
2. No way to do undo (travel back in time) easily without adding so much extra code.

## 425. Are there any similitudes among Redux and RxJS?

These libraries are altogether different for altogether different purposes, yet there are some unclear similitudes. Redux is a device for overseeing state all through the application.

It is typically utilized as a design for UIs. Consider it an option in contrast to (half of) Angular. RxJS is a responsive programming library. It is typically utilized as an apparatus to achieve offbeat undertakings in JavaScript.

Consider it an option in contrast to Promises. Redux utilizations the Reactive worldview on the grounds that the Store is responsive. The Store watches activities from a separation and changes itself.

RxJS additionally utilizes the Reactive worldview, yet as opposed to being an engineer, it gives you fundamental building squares, Observables, to achieve this example.

### 426.     How to utilize interface() from React Redux?

You have to pursue two stages to utilize your store in your compartment: Use mapStateToProps(): It maps the state factors from your store to the props that you determine.

Interface the above props to your compartment: The protest returned by the mapStateToProps work is associated with the holder. You can import interface() from respond redux.

import React from 'respond' import { associate } from 'respond redux'

class App broadens React.Component {
render() { return <div>{this.props.containerData}</div> } }

work mapStateToProps(state) { return { containerData: state.data } } send out default connect(mapStateToProps)(App)

### 427.     What is the contrast between React setting and React Redux?

You can utilize Context in your application specifically and will be extraordinary for going down information to profoundly settled parts which what it was intended for. Though Redux is substantially more incredible and gives an extensive number of highlights that the Context API doesn't give. Likewise, React Redux utilizations setting inside yet it doesn't uncover this reality in the general population API.

### 428.     For what reason are Redux state capacities called reducers?

Reducers dependably restore the amassing of the state (in view of all past and current activities). Accordingly, they go about as a reducer of state. Each time a Redux reducer is called, the state and activity are passed as parameters.

This state is then lessened (or aggregated) in view of the activity, and afterward the following state is returned. You could diminish a gathering of activities and an underlying state (of the store) on which to play out these activities to get the subsequent last state.

### 429. Real Advantages of utilizing React.

• Testability :- it is anything but difficult to test, and incorporate a few devices like quip.

• Maintainability :- it guarantees clarity and makes practicality simpler.

• Performance :- it is great execution since it's utilization virtual-dom.

• Rendering :- it's render server-side and customer side also.

• Code Reusability :- React about segments. So reactJS furnishes designers with the chance to have more opportunity to utilize and make normal reflections, setting up the creation, circulation and utilization of segregated reusable parts.

• Better Combination Technologies :- React makes best utilization of HTML and JavaScript blending them in a perfect world and furthermore consolidates CSS to furnish your business with the best.

• Integrate With Others Framework :- it's anything but difficult to incorporate with different systems like Backbone.js, Meteor, Angular, and so forth.

• JSX:- JSX makes it simple to peruse the code of your segments. It is likewise extremely simple to see the design, or how segments are stopped/joined with one another.

• Data Binding :- React utilizes one-way information official or uni-directional information stream .

### 430. What is state in respond js ?

it holds the data about the part and state is utilizing just inside the segment. State is alterable. As a matter of course segment has no state

### 431. What is props in respond js ?

Props implies properties, It's a method for passing information from parent to kid and we can state that props are only a correspondence channel between parts. Continually moving from parent to kid part. It's a permanent and perused as it were.

### 432. What is ReactJS Components?

ReactJS segments split the UI into free reusable pieces that is called segment. parts resemble JavaScript capacities.

### 433. What's the distinction b/w an Element and a Component in React?

Fundamentally, a React part depicts what you have to see on the screen. Not all that fundamentally, a React component is a challenge depiction of some UI.
A React segment is a capacity or a class which then again recognizes information and returns a React segment (usually by methods for JSX which gets transpiled to a create Element summon)

### 434. What is a three explanations for the accomplishment of ReactJS?

• ReactJS is an innovation that can be trusted for complex undertakings. • Developers don't stress over the bugs since it generally guarantees mistake free results it offers versatile applications.
• It is quick innovation and can basically be trusted for quality results.

### 435. What are the center standards of Redux?

Redux pursues three crucial standards:
• Single wellspring of truth: The condition of your entire application is put away in a protest tree inside a solitary store. The single state tree makes it less demanding to monitor changes after some time and troubleshoot or examine the application.
• State is perused just: The best way to change the state is to produce an activity, a protest portraying what occurred. This guarantees neither the perspectives nor the system callbacks will ever compose specifically to the state.
• Changes are made with unadulterated capacities: To determine how the state tree is changed by activities, you compose reducers. Reducers are simply unadulterated capacities that take the past state and an activity as parameters, and restore the following state.

### 436. Characterize State in React and the manner in which it is utilized in React.

States are essentially information source in React and are kept as basic as could reasonably be expected. Through states, the rendering and conduct of the parts are distinguished. In contrast to props, they are changeable and can be utilized to make intuitive and dynamic segments. For the most part, this.state() work is utilized to get to them.

### 437. How is React different from Angular?

React vs Angular

| TOPIC | REACT | ANGULAR |
|---|---|---|
| ARCHITECTURE | Only the View of MVC | Complete MVC |
| RENDERING | Server side rendering | Client side rendering |
| DOM | Uses virtual DOM | Uses real DOM |
| DATA BINDING | One-way data binding | Two-way data binding |
| DEBUGGING | Compile time debugging | Run time debugging |
| AUTHOR | Facebook | Google |

### 438. Why can't browsers read JSX?

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.

## 439. What is Shallow Renderer in React testing?

*Shallow rendering* is useful for writing unit test cases in React. It lets you render a component *one level deep* and assert facts about what its render method returns, without worrying about the behavior of child components, which are not instantiated or rendered.

For example, if you have the following component:

```
function MyComponent() {
    return (
        <div>
            <span className={'heading'}>{'Title'}</span>
            <span className={'description'}>{'Description'}</span>
        </div>
    )
}
```

Then you can assert as follows:

```
import ShallowRenderer from 'react-test-renderer/shallow'
```

```
// in your test
const renderer = new ShallowRenderer()
renderer.render(<MyComponent />)

const result = renderer.getRenderOutput()

expect(result.type).toBe('div')
expect(result.props.children).toEqual([
    <span className={'heading'}>{'Title'}</span>,
    <span className={'description'}>{'Description'}</span>
])
```

## 440.    What is `TestRenderer` package in React?

This package provides a renderer that can be used to render components to pure JavaScript objects, without depending on the DOM or a native mobile environment. This package makes it easy to grab a snapshot of the platform view hierarchy (similar to a DOM tree) rendered by a ReactDOM or React Native without using a browser or `jsdom`.

```
import TestRenderer from 'react-test-renderer'

const Link = ({ page, children }) => <a href={page}>{children}<
/a>

const testRenderer = TestRenderer.create(
    <Link page={'https://www.facebook.com/'}>{'Facebook'}</Link
>
)

console.log(testRenderer.toJSON())
```

## 441.    What is the purpose of ReactTestUtils package?

*ReactTestUtils* are provided in the `with-addons` package and allow you to perform actions against a simulated DOM for the purpose of unit testing.

## 442.    What is Jest?

*Jest* is a JavaScript unit testing framework created by Facebook based on Jasmine and provides automated mock creation and a `jsdom` environment. It's often used for testing components.

## 443.    What are the advantages of Jest over Jasmine?

There are couple of advantages compared to Jasmine:

- o  Automatically finds tests to execute in your source code.
- o  Automatically mocks dependencies when running your tests.
- o  Allows you to test asynchronous code synchronously.
- o  Runs your tests with a fake DOM implementation (via `jsdom`) so that your tests can be run on the command line.
- o  Runs tests in parallel processes so that they finish sooner.

## 444.    Give a simple example of Jest test case

Let's write a test for a function that adds two numbers in `sum.js` file:

```
const sum = (a, b) => a + b

export default sum
```

Create a file named `sum.test.js` which contains actual test:

```
import sum from './sum'

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3)
})
```

And then add the following section to your `package.json`:

```
{
  "scripts": {
    "test": "jest"
  }
}
```

Finally, run `yarn test` or `npm test` and Jest will print a result:

```
$ yarn test
PASS ./sum.test.js
✓ adds 1 + 2 to equal 3 (2ms)
```

## 445.    What is flux?

*Flux* is an *application design paradigm* used as a replacement for the more traditional MVC pattern. It is not a framework or a library but a new kind of architecture that complements React and the concept of Unidirectional Data Flow. Facebook uses this pattern internally when working with React.

The workflow between dispatcher, stores and views components with distinct inputs and outputs as follows:
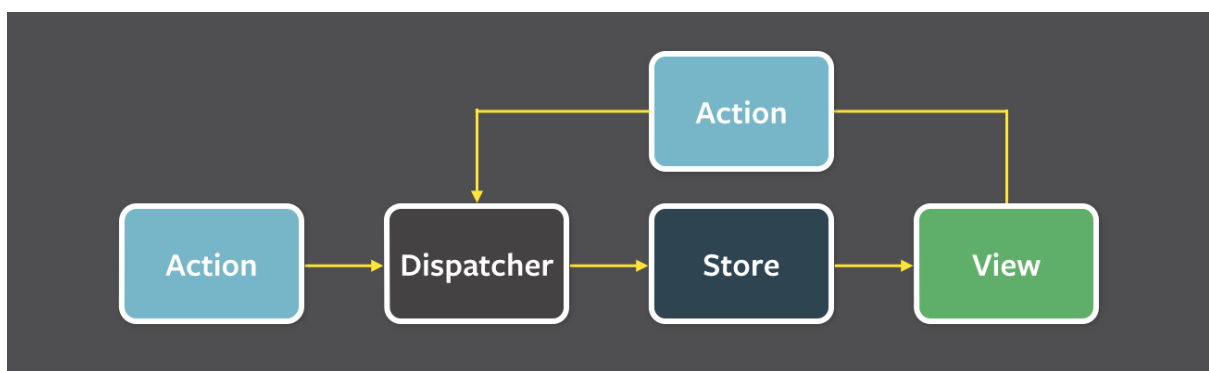


## 446.    What is Redux?

*Redux* is a predictable state container for JavaScript apps based on the *Flux design pattern*. Redux can be used together with React, or with any other view library. It is tiny (about 2kB) and has no dependencies.

## 447.    What are the core principles of Redux?

Redux follows three fundamental principles:

i.    **Single source of truth:** The state of your whole application is stored in an object 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 emit an action, an object describing what happened. This ensures that neither the views nor the network callbacks will ever write directly to the state.

iii. **Changes are made with pure functions:** To specify how the state tree is transformed by actions, you write reducers. Reducers are just pure functions that take the previous state and an action as parameters, and return the next state.

## 448.   What are the downsides of Redux compared to Flux?

Instead of saying downsides we can say that there are few compromises of using Redux over Flux. Those are as follows:

i. **You will need to learn to avoid mutations:** Flux is un-opinionated about mutating data, but Redux doesn't like mutations and many packages complementary to Redux assume you never mutate the state. You can enforce this with dev-only packages like `redux-immutable-state-invariant`, Immutable.js, or instructing your team to write non-mutating code.

ii. **You're going to have to carefully pick your packages:** While Flux explicitly doesn't try to solve problems such as undo/redo, persistence, or forms, Redux has extension points such as middleware and store enhancers, and it has spawned a rich ecosystem.

iii. **There is no nice Flow integration yet:** Flux currently lets you do very impressive static type checks which Redux doesn't support yet.

## 449.   What is the difference between `mapStateToProps()` and `mapDispatchToProps()`?

**mapStateToProps()** is a utility which helps your component get updated state (which is updated by some other components):

```
const mapStateToProps = (state) => {
  return {
    todos: getVisibleTodos(state.todos, state.visibilityFilter)
  }
}
```

**mapDispatchToProps()** is a utility which will help your component to fire an action event (dispatching action which may cause change of application state):

```
const mapDispatchToProps = (dispatch) => {
  return {
    onTodoClick: (id) => {
      dispatch(toggleTodo(id))
    }
  }
}
```

Recommend always using the "object shorthand" form for the mapDispatchToProps

Redux wrap it in another function that looks like (...args) => dispatch(onTodoClick(...args)), and pass that wrapper function as a prop to your component.

```
const mapDispatchToProps = ({
  onTodoClick
})
```

## 450.  Can I dispatch an action in reducer?

Dispatching an action within a reducer is an **anti-pattern**. Your reducer should be *without side effects*, simply digesting the action payload and returning a new state object. Adding listeners and dispatching actions within the reducer can lead to chained actions and other side effects.

## 451.  How to access Redux store outside a component?

You just need to export the store from the module where it created with createStore(). Also, it shouldn't pollute the global window object.
store = createStore(myReducer)

export default store

## 452.  What are the drawbacks of MVW pattern?

i. DOM manipulation is very expensive which causes applications to behave slow and inefficient.

ii. Due to circular dependencies, a complicated model was created around models and views.

iii. Lot of data changes happens for collaborative applications(like Google Docs).

iv. No way to do undo (travel back in time) easily without adding so much extra code.

### 453. Are there any similarities between Redux and RxJS?

These libraries are very different for very different purposes, but there are some vague similarities.

Redux is a tool for managing state throughout the application. It is usually used as an architecture for UIs. Think of it as an alternative to (half of) Angular. RxJS is a reactive programming library. It is usually used as a tool to accomplish asynchronous tasks in JavaScript. Think of it as an alternative to Promises. Redux uses the Reactive paradigm because the Store is reactive. The Store observes actions from a distance, and changes itself. RxJS also uses the Reactive paradigm, but instead of being an architecture, it gives you basic building blocks, Observables, to accomplish this pattern.

### 454. How to dispatch an action on load?

You can dispatch an action in `componentDidMount()` method and in `render()` method you can verify the data.

```
class App extends Component {
    componentDidMount() {
        this.props.fetchData()
    }

    render() {
        return this.props.isLoaded
            ? <div>{'Loaded'}</div>
            : <div>{'Not Loaded'}</div>
```

```
    }
}

const mapStateToProps = (state) => ({
    isLoaded: state.isLoaded
})

const mapDispatchToProps = { fetchData }

export default connect(mapStateToProps, mapDispatchToProps)(App
)
```

## 455. How to use connect() from React Redux?

You need to follow two steps to use your store in your container:

**Use mapStateToProps():** It maps the state variables from your store to the props that you specify.
**Connect the above props to your container:** The object returned by the mapStateToProps function is connected to the container. You can import connect() from react-redux.

```
import React from 'react'
import { connect } from 'react-redux'

class App extends React.Component {
    render() {
        return <div>{this.props.containerData}</div>
    }
}

function mapStateToProps(state) {
    return { containerData: state.data }
}

export default connect(mapStateToProps)(App)
```

## 456. How to reset state in Redux?

You need to write a *root reducer* in your application which delegate handling the action to the reducer generated by `combineReducers()`.
For example, let us take `rootReducer()` to return the initial state after `USER_LOGOUT` action. As we know, reducers are supposed to return the initial state when they are called with `undefined` as the first argument, no matter the action.

```javascript
const appReducer = combineReducers({
    /* your app's top-level reducers */
})

const rootReducer = (state, action) => {
    if (action.type === 'USER_LOGOUT') {
        state = undefined
    }

    return appReducer(state, action)
}
```

In case of using `redux-persist`, you may also need to clean your storage. `redux-persist` keeps a copy of your state in a storage engine. First, you need to import the appropriate storage engine and then, to parse the state before setting it to undefined and clean each storage state key.

```javascript
const appReducer = combineReducers({
    /* your app's top-level reducers */
})

const rootReducer = (state, action) => {
    if (action.type === 'USER_LOGOUT') {
        Object.keys(state).forEach(key => {
            storage.removeItem(`persist:${key}`)
        })

        state = undefined
    }

    return appReducer(state, action)
}
```

**457.** **Whats the purpose of** at **symbol in the Redux connect decorator?**

The **@** symbol is in fact a JavaScript expression used to signify decorators. *Decorators* make it possible to annotate and modify classes and properties at design time.

Let's take an example setting up Redux without and with a decorator.

- **Without decorator:**

```
import React from 'react'
import * as actionCreators from './actionCreators'
import { bindActionCreators } from 'redux'
import { connect } from 'react-redux'

function mapStateToProps(state) {
    return { todos: state.todos }
}

function mapDispatchToProps(dispatch) {
    return { actions: bindActionCreators(actionCreators, dispatch) }
}

class MyApp extends React.Component {
    // ...define your main app here
}

export default connect(mapStateToProps, mapDispatchToProps)(MyApp)
```

- **With decorator:**

```
import React from 'react'
import * as actionCreators from './actionCreators'
import { bindActionCreators } from 'redux'
import { connect } from 'react-redux'

function mapStateToProps(state) {
    return { todos: state.todos }
}
```

```
function mapDispatchToProps(dispatch) {
    return { actions: bindActionCreators(actionCreators, d
ispatch) }
}

@connect(mapStateToProps, mapDispatchToProps)
export default class MyApp extends React.Component {
    // ...define your main app here
}
```

The above examples are almost similar except the usage of decorator. The decorator syntax isn't built into any JavaScript runtimes yet, and is still experimental and subject to change. You can use babel for the decorators support.

## 458.    What is the difference between React context and React Redux?

You can use **Context** in your application directly and is going to be great for passing down data to deeply nested components which what it was designed for.

Whereas **Redux** is much more powerful and provides a large number of features that the Context API doesn't provide. Also, React Redux uses context internally but it doesn't expose this fact in the public API.

## 459.    Why are Redux state functions called reducers?

Reducers always return the accumulation of the state (based on all previous and current actions). Therefore, they act as a reducer of state. Each time a Redux reducer is called, the state and action are passed as parameters. This state is then reduced (or accumulated) based on the action, and then the next state is returned. You could *reduce* a collection of actions and an initial state (of the store) on which to perform these actions to get the resulting final state.

## 460.    How to make AJAX request in Redux?

You can use `redux-thunk` middleware which allows you to define async actions.

Let's take an example of fetching specific account as an AJAX call using *fetch API*:

```javascript
export function fetchAccount(id) {
    return dispatch => {
        dispatch(setLoadingAccountState()) // Show a loading sp
inner
        fetch(`/account/${id}`, (response) => {
            dispatch(doneFetchingAccount()) // Hide loading spi
nner
            if (response.status === 200) {
                dispatch(setAccount(response.json)) // Use a no
rmal function to set the received state
            } else {
                dispatch(someError)
            }
        })
    }
}

function setAccount(data) {
    return { type: 'SET_Account', data: data }
}
```

## 461.    Should I keep all component's state in Redux store?

Keep your data in the Redux store, and the UI related state internally in the component.

## 462.    What is the proper way to access Redux store?

The best way to access your store in a component is to use the `connect()` function, that creates a new component that wraps around your existing one. This pattern is called *Higher-Order Components*, and is generally the preferred way of extending a component's functionality in React. This allows you to map state and action creators to your component, and have them passed in automatically as your store updates.

Let's take an example of `<FilterLink>` component using connect:

```
import { connect } from 'react-redux'
import { setVisibilityFilter } from '../actions'
import Link from '../components/Link'

const mapStateToProps = (state, ownProps) => ({
    active: ownProps.filter === state.visibilityFilter
})

const mapDispatchToProps = (dispatch, ownProps) => ({
    onClick: () => dispatch(setVisibilityFilter(ownProps.filter
))
})

const FilterLink = connect(
    mapStateToProps,
    mapDispatchToProps
)(Link)

export default FilterLink
```

Due to it having quite a few performance optimizations and generally being less likely to cause bugs, the Redux developers almost always recommend using `connect()` over accessing the store directly (using context API).

```
class MyComponent {
    someMethod() {
        doSomethingWith(this.context.store)
    }
}
```

## 463.    What is the difference between component and container in React Redux?

**Component** is a class or function component that describes the presentational part of your application.

**Container** is an informal term for a component that is connected to a Redux store. Containers *subscribe* to Redux state updates and *dispatch* actions, and

they usually don't render DOM elements; they delegate rendering to presentational child components.

## 464. What is the purpose of the constants in Redux?

Constants allows you to easily find all usages of that specific functionality across the project when you use an IDE. It also prevents you from introducing silly bugs caused by typos – in which case, you will get
a `ReferenceError` immediately.
Normally we will save them in a single file
(`constants.js` or `actionTypes.js`).

```js
export const ADD_TODO = 'ADD_TODO'
export const DELETE_TODO = 'DELETE_TODO'
export const EDIT_TODO = 'EDIT_TODO'
export const COMPLETE_TODO = 'COMPLETE_TODO'
export const COMPLETE_ALL = 'COMPLETE_ALL'
export const CLEAR_COMPLETED = 'CLEAR_COMPLETED'
```

In Redux, you use them in two places:

i. **During action creation:**

Let's take `actions.js`:
```js
import { ADD_TODO } from './actionTypes';

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

ii. **In reducers:**

Let's create `reducer.js`:
```js
import { ADD_TODO } from './actionTypes'

export default (state = [], action) => {
    switch (action.type) {
        case ADD_TODO:
            return [
                ...state,
                {
                    text: action.text,
```

```
                        completed: false
                }
            ];
        default:
            return state
    }
}
```

## 465. What are the different ways to write `mapDispatchToProps()`?

There are a few ways of binding *action creators* to `dispatch()` in `mapDispatchToProps()`.

Below are the possible options:

```
const mapDispatchToProps = (dispatch) => ({
    action: () => dispatch(action())
})
const mapDispatchToProps = (dispatch) => ({
    action: bindActionCreators(action, dispatch)
})
const mapDispatchToProps = { action }
```

The third option is just a shorthand for the first one.

## 466. What is the use of the `ownProps` parameter in `mapStateToProps()` and `mapDispatchToProps()`?

If the `ownProps` parameter is specified, React Redux will pass the props that were passed to the component into your *connect* functions. So, if you use a connected component:

```
import ConnectedComponent from './containers/ConnectedComponent';

<ConnectedComponent user={'john'} />
```

The `ownProps` inside your `mapStateToProps()` and `mapDispatchToProps()` functions will be an object:
`{ user: 'john' }`

You can use this object to decide what to return from those functions.

## 467. How to structure Redux top level directories?

Most of the applications has several top-level directories as below:

    i.   **Components**: Used for *dumb* components unaware of Redux.

    ii.   **Containers**: Used for *smart* components connected to Redux.

    iii.   **Actions**: Used for all action creators, where file names correspond to part of the app.

    iv.   **Reducers**: Used for all reducers, where files name correspond to state key.

    v.   **Store**: Used for store initialization.

This structure works well for small and medium size apps.

## 468. What is redux-saga?

`redux-saga` is a library that aims to make side effects (asynchronous things like data fetching and impure things like accessing the browser cache) in React/Redux applications easier and better.

It is available in NPM:

```
$ npm install --save redux-saga
```

## 469. What is the mental model of redux-saga?

*Saga* is like a separate thread in your application, that's solely responsible for side effects. `redux-saga` is a redux *middleware*, which means this thread can be started, paused and cancelled from the main application with normal Redux actions, it has access to the full Redux application state and it can dispatch Redux actions as well.

## 470. What are the differences between `call()` and `put()` in redux-saga?

Both `call()` and `put()` are effect creator functions. `call()` function is used to create effect description, which instructs middleware to call the promise. `put()` function creates an effect, which instructs middleware to dispatch an action to the store.

Let's take example of how these effects work for fetching particular user data.

```
function* fetchUserSaga(action) {
    // `call` function accepts rest arguments, which will be pa
ssed to `api.fetchUser` function.
    // Instructing middleware to call promise, it resolved valu
e will be assigned to `userData` variable
    const userData = yield call(api.fetchUser, action.userId)

    // Instructing middleware to dispatch corresponding action.
    yield put({
        type: 'FETCH_USER_SUCCESS',
        userData
    })
}
```

## 471.    What is Redux Thunk?

*Redux Thunk* middleware allows you to write action creators that return a function instead of an action. The thunk can be used to delay the dispatch of an action, or to dispatch only if a certain condition is met. The inner function receives the store methods `dispatch()` and `getState()` as parameters.

## 472.    What are the differences between `redux-saga` and `redux-thunk`?

Both *Redux Thunk* and *Redux Saga* take care of dealing with side effects. In most of the scenarios, Thunk uses *Promises* to deal with them, whereas Saga uses *Generators*. Thunk is simple to use and Promises are familiar to many developers, Sagas/Generators are more powerful but you will need to learn them. But both middleware can coexist, so you can start with Thunks and introduce Sagas when/if you need them.

## 473.    What is Redux DevTools?

*Redux DevTools* is a live-editing time travel environment for Redux with hot reloading, action replay, and customizable UI. If you don't want to bother with installing Redux DevTools and integrating it into your project, consider using Redux DevTools Extension for Chrome and Firefox.

## 474. What are the features of Redux DevTools?

Some of the main features of Redux DevTools are below,

i. Lets you inspect every state and action payload.
ii. Lets you go back in time by *cancelling* actions.
iii. If you change the reducer code, each *staged* action will be re-evaluated.
iv. If the reducers throw, you will see during which action this happened, and what the error was.
v. With `persistState()` store enhancer, you can persist debug sessions across page reloads.

## 475. What are Redux selectors and why to use them?

*Selectors* are functions that take Redux state as an argument and return some data to pass to the component.

For example, to get user details from the state:

```
const getUserData = state => state.user.data
```

These selectors have two main benefits,

i. The selector can compute derived data, allowing Redux to store the minimal possible state
ii. The selector is not recomputed unless one of its arguments changes

## 476. What is Redux Form?

*Redux Form* works with React and Redux to enable a form in React to use Redux to store all of its state. Redux Form can be used with raw HTML5 inputs,

but it also works very well with common UI frameworks like Material UI, React Widgets and React Bootstrap.

## 477. What are the main features of Redux Form?

Some of the main features of Redux Form are:

    i.    Field values persistence via Redux store.
    ii.    Validation (sync/async) and submission.
    iii.    Formatting, parsing and normalization of field values.

## 478. How to add multiple middlewares to Redux?

You can use `applyMiddleware()`.
For example, you can add `redux-thunk` and `logger` passing them as arguments to `applyMiddleware()`:

```
import { createStore, applyMiddleware } from 'redux'
const createStoreWithMiddleware =
    applyMiddleware(ReduxThunk, logger)(createStore)
```

## 479. How to set initial state in Redux?

You need to pass initial state as second argument to createStore:

```
const rootReducer = combineReducers({
    todos: todos,
    visibilityFilter: visibilityFilter
})

const initialState = {
    todos: [{ id: 123, name: 'example', completed: false }]
}

const store = createStore(
    rootReducer,
    initialState
```

### 480.     How Relay is different from Redux?

Relay is similar to Redux in that they both use a single store. The main difference is that relay only manages state originated from the server, and all access to the state is used via *GraphQL* queries (for reading data) and mutations (for changing data). Relay caches the data for you and optimizes data fetching for you, by fetching only changed data and nothing more.

### 481.     What is an action in Redux?

*Actions* are plain JavaScript objects or payloads of information that send data from your application to your store. They are the only source of information for the store. Actions must have a type property that indicates the type of action being performed.

For example, let's take an action which represents adding a new todo item:

```
{
    type: ADD_TODO,
    text: 'Add todo item'
}
```

### 482.     What is the difference between React Native and React?

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

**React Native** is a mobile framework that compiles to native app components, allowing you to build native mobile applications (iOS, Android, and Windows) in JavaScript that allows you to use React to build your components, and implements React under the hood.

### 483.     How to test React Native apps?

React Native can be tested only in mobile simulators like iOS and Android. You can run the app in your mobile using expo app (https://expo.io) Where it syncs using QR code, your mobile and computer should be in same wireless network.

### 484.   How to do logging in React Native?

You can use `console.log`, `console.warn`, etc. As of React Native v0.29 you can simply run the following to see logs in the console:
```
$ react-native log-ios
$ react-native log-android
```

### 485.   How to debug your React Native?

Follow the below steps to debug React Native app:

  i.   Run your application in the iOS simulator.
  ii.   Press `Command + D` and a webpage should open up at `http://localhost:8081/debugger-ui`.
  iii.   Enable *Pause On Caught Exceptions* for a better debugging experience.
  iv.   Press `Command + Option + I` to open the Chrome Developer tools, or open it via `View -> Developer -> Developer Tools`.
  v.   You should now be able to debug as you normally would.

### 486.   What is reselect and how it works?

*Reselect* is a **selector library** (for Redux) which uses *memoization* concept. It was originally written to compute derived data from Redux-like applications state, but it can't be tied to any architecture or library.

Reselect keeps a copy of the last inputs/outputs of the last call, and recomputes the result only if one of the inputs changes. If the the same inputs are provided twice in a row, Reselect returns the cached output. It's memoization and cache are fully customizable.

**487.    What is Flow?**

*Flow* is a *static type checker* designed to find type errors in JavaScript. Flow types can express much more fine-grained distinctions than traditional type systems. For example, Flow helps you catch errors involving `null`, unlike most type systems.

**488.    What is the difference between Flow and PropTypes?**

Flow is a *static analysis tool* (static checker) which uses a superset of the language, allowing you to add type annotations to all of your code and catch an entire class of bugs at compile time.

PropTypes is a *basic type checker* (runtime checker) which has been patched onto React. It can't check anything other than the types of the props being passed to a given component. If you want more flexible typechecking for your entire project Flow/TypeScript are appropriate choices.

**489.    How to use Font Awesome icons in React?**

The below steps followed to include Font Awesome in React:

i.   Install `font-awesome`:
`$ npm install --save font-awesome`

ii.  Import `font-awesome` in your `index.js` file:
`import 'font-awesome/css/font-awesome.min.css'`

iii. Add Font Awesome classes in `className`:

iv.  `render() {`

v.   `return <div><i className={'fa fa-spinner'} /></div>`
`}`

**490.    What is React Dev Tools?**

*React Developer Tools* let you inspect the component hierarchy, including component props and state. It exists both as a browser extension (for Chrome and Firefox), and as a standalone app (works with other environments including Safari, IE, and React Native).

The official extensions available for different browsers or environments.

   i.     **Chrome extension**

   ii.     **Firefox extension**

   iii.     **Standalone app** (Safari, React Native, etc)

## 491.    Why is DevTools not loading in Chrome for local files?

If you opened a local HTML file in your browser (`file://...`) then you must first open *Chrome Extensions* and check `Allow access to file URLs`.

## 492.    How to use Polymer in React?

You need to follow below steps to use Polymer in React,

Create a Polymer element:

```
<link rel='import' href='../../bower_components/polymer/polymer.html' />
Polymer({
    is: 'calender-element',
    ready: function () {
        this.textContent = 'I am a calender'
    }
})
```

Create the Polymer component HTML tag by importing it in a HTML document, e.g. import it in the `index.html` of your React application:

```
<link rel='import' href='./src/polymer-components/calender-element.html'></link>
```

Use that element in the JSX file:

```
import React from 'react'

class MyComponent extends React.Component {
    render() {
        return (
            <calender-element />
```

```
        )
    }
}
export default MyComponent
```

### 493.    What are the advantages of React over Vue.js?

React has the following advantages over Vue.js:

    i.    Gives more flexibility in large apps developing.
   ii.    Easier to test.
  iii.    Suitable for mobile apps creating.
  iv.    More information and solutions available.

**Note:** The above list of advantages are purely opinionated and it vary based on the professional experience. But they are helpful as base parameters.

### 494.    What is the difference between React and Angular?

Let's see the difference between React and Angular in a table format.

| React | Angular |
| --- | --- |
| React is a library and has only the View layer | Angular is a framework and has complete MVC functionality |
| React handles rendering on the server side | AngularJS renders only on the client side but Angular 2 and above renders on the server side |
| React uses JSX that looks like HTML in JS which can be confusing | Angular follows the template approach for HTML, which makes code shorter and easy to understand |
| React Native, which is a React type to build mobile applications are faster and more stable | Ionic, Angular's mobile native app is relatively less stable and slower |

|   React   |   Angular   |
|-----------|-------------|
| In React, data flows only in one way and hence debugging is easy | In Angular, data flows both way i.e it has two-way data binding between children and parent and hence debugging is often difficult |

**Note:** The above list of differences are purely opinionated and it vary based on the professional experience. But they are helpful as base parameters.

## 495.    Why React tab is not showing up in DevTools?

When the page loads, *React DevTools* sets a global named `__REACT_DEVTOOLS_GLOBAL_HOOK__`, then React communicates with that hook during initialization. If the website is not using React or if React fails to communicate with DevTools then it won't show up the tab.