React & Redux Interview Q & A…

**Q.1. How to create components in React?**

**Ans.** There are two possible ways to create a component.

**Functional 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>;

}

}

**Q.2. What are the difference between a class component and functional component?**

**Ans.**

**class Components**

* Class-based Components uses ES6 class syntax. It can make use of the lifecycle methods.
* Class components extend from React.Component.
* In here you have to use this keyword to access the props and functions that you declare inside the class components.

**Functional Components**

* Functional Components are simpler comparing to class-based functions.
* Functional Components mainly focuses on the UI of the application, not on the behavior.
* To be more precise these are basically render function in the class component.
* Functional Components can have state and mimic lifecycle events using Reach Hooks

**Q.3. What is difference between controlled vs uncontrolled component?**  
**Ans.**

* **Controlled Components**  
  In HTML, form elements such as <input />, <textarea />, and <select /> typically maintain their own state and update it based on user input. When a user submits a form, the values from the elements mentioned above are sent with the form. With React it works differently. The component containing the form will keep track of the value of the input in its state and will re-render the component each time the callback function, e.g., onChange is fired as the state will be updated. An input form element whose value is controlled by React in this way is called a "controlled component". You could also call this a "dumb component".
* **Uncontrolled Components**A Uncontrolled Component is one that stores its own state internally, and you query the DOM using a ref to find its current value when you need it. This is a bit more like traditional HTML.
* Example
* // Controlled:
* <input type="text" value={value} onChange={handleChange} />
* // Uncontrolled:
* <input type="text" defaultValue="foo" ref={inputRef} />
* // Use `inputRef.current.value` to read the current value of <input>

**Q.4. What is children?**

**Ans.** In JSX expressions that contain both an opening tag and a closing tag, the content between those tags is passed to components automatically as a special prop:

props.children

There are some 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 👶

const MainContainer = React.createClass({

render: function () {

return <div>{this.props.children}</div>;

},

});

ReactDOM.render(

<MainContainer>

<span>{'Hello'}</span>

<span>{'World'}</span>

</MainContainer>,

node,

);

**Q.5. What is prop drilling and how can you avoid it?**

**Ans.** While passing a prop from each component to the next in the hierarchy from the source component to the deeply nested component. This is called **prop drilling**.

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

**Q.6. What is Pure Component?**

**Ans.** 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.

**Q.7. Why should we not update the state directly?**

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

//Wrong ❌

this.state.message = 'Not Updated';

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: 'Updated' });

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

**Q.8. What is the purpose of callback function as an argument of setState()**

**Ans.** 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: 'Supi' }, () => console.log('The name has updated and component re-rendered'));

**Q.9. What are synthetic events in React?**

**Ans.** Synthetic Event 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.

**Q.10. What is "key" prop and what is the benefit of using it in arrays of elements 🗝?**

**Ans.** 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:

1. 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.
2. If you extract list item as separate component then apply *keys* on list component instead of li tag.
3. There will be a warning message in the console if the key prop is not present on list items.

**Q.11. Why are String Refs legacy?**

**Ans.** 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**.

1. 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.
2. 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.
3. 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.
4. 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 DataTable 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} />;

}

}

**Q.12. What is the difference between createElement and cloneElement?**

**Ans.** 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.

**Q.13. What is reconciliation?**

**Ans.** 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.

**Q.14. Is lazy function supports named exports?**

**Ans.** 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,

Example:

// FewComponents.js

export const SomeComponent = /\* ... \*/;

export const UnusedComponent = /\* ... \*/;

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

// IntermediateComponent.js

export { SomeComponent as default } from './FewComponents.js';

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

import React, { lazy } from 'react';

const SomeComponent = lazy(() => import('./IntermediateComponent.js'));

**Q.15. What are portals in React?**

**Ans.** 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.

**Q.16. What are stateless components?**

**Ans.** 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.

**Q.17. What are stateful components?**

**Ans.** 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

)

}

**Q.18. What is the impact of indexes as keys?**

**Ans.** 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} />)

}

**Q.19. How do you memoize a component?**

**Ans.** 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);

**Q.20. Why we need to pass a function to setState()?**

**Ans.** 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

**Q.21. Why should component names start with capital letter?**

**Ans.** 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 OneComponent extends Component {

// ...

}

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';

**What are the exceptions on React component naming?**

The component names should start with a uppercase letter but there are few exceptions on this convention. The lowercase tag names with a dot (property accessors) are still considered as valid component names.

For example the below tag can be compiled to a valid component,

render() {

return (

<obj.component /> // `React.createElement(obj.component)`

)

}

**Q.22. Can you force a component to re-render without calling setState?**

**Ans.** 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().

**Q.23. What is the difference between super() and super(props) in React usin ES6 classes?**

**Ans.** 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: 'Supi', ... }

}

}

**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.

**Q.24. Is it mandatory to define constructor for React component?**

**Ans.** 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.

**Q.25. What are default props?**

**Ans.** 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: 'blue',

};

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.

**Q.26. How to apply validation on props in React?**

**Ans.** 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:

1. PropTypes.number
2. PropTypes.string
3. PropTypes.array
4. PropTypes.object
5. PropTypes.func
6. PropTypes.node
7. PropTypes.element
8. PropTypes.bool
9. PropTypes.symbol
10. 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.

**Q.27. Why you can't update props in React?**

**Ans.** 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.

**Q.28. What are render props?**

**Ans.** 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.

**Q.29. What is Suspense component?**

**Ans.** 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.

Example

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

function MyComponent() {

return (

<div>

<Suspense fallback={<div>Loading...</div>}>

<OneComponent />

</Suspense>

</div>

);

}

As mentioned in the above code, Suspense is wrapped above the lazy component.

**Q.30. What is diffing algorithm?**

**Ans.** 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:

1. Two elements of different types will produce different trees.
2. The developer can hint at which child elements may be stable across different renders with a key prop.

**Q.31. How to re-render the view when the browser is resized?**

**Ans.** 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.updateDimensions);

}

updateDimensions() {

this.setState({ width: window.innerWidth, height: window.innerHeight });

}

render() {

return (

<span>

{this.state.width} x {this.state.height}

</span>

);

}

}

**Q.32. What is React memo function?**

**Ans.** 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 \*/

});

**Q.33. What is the methods order when component re-rendered?**

**Ans.** 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.

1. static getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

**Q.34. What are loadable components?**

**Ans.** 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

**Q.35. How to pretty print JSON with React?**

**Ans.** 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'));

**Q.36. What is render hijacking in react?**

**Ans.** 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.

**Q.37. How to use https instead of http in create-react-app?**

**Ans.** 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

**Q.38. How can we convert functional component to pure component?**

**Ans.** We can convert functional to pure component using React.memo.

Redux

**Q.1. What are reducers in redux?**

**Ans.** The reducer is a pure function that takes the previous state and an action, and returns the next state.

(previousState, action) =>newState

It's very important that the reducer stays *pure*. Things you should never do inside a reducer:

* Mutate its arguments;
* Perform side effects like API calls and routing transitions;
* Call non-pure functions, e.g. Date.now() or Math.random()

**Q.2. How is state changed in redux?**

**Ans.** 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. Instead, they express an intent to transform the state. Because all changes are centralized and happen one by one in a strict order, there are no subtle race conditions to watch out for. As actions are just plain objects, they can be logged, serialized, stored, and later replayed for debugging or testing purposes.

**Q.3. How Redux Form initialValues get updated from state?**

**Ans.** 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.

**Q.4. What is Redux Thunk?**

**Ans.** 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.

**Q.5. What is the difference between mapStateToProps() and mapDispatchToProps()?**

**Ans.**

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,

};

**Q.6. How to add multiple middlewares to Redux?**

**Ans.** You can use applyMiddleware where you can pass each piece of middleware as a new argument. So you just need to pass each piece of middleware you'd like. For example, you can add Redux Thunk and logger middlewares as an argument as below,

import { createStore, applyMiddleware } from 'redux'

const createStoreWithMiddleware = applyMiddleware(ReduxThunk, logger)(createStore);

**Q.7. What is React context vs React redux?**

**Ans.** 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. So you should feel much safer using Context via React Redux than directly because if it changes, the burden of updating the code will be on React Redux instead developer responsibility.

**Q.8. Why React uses className over class attribute?**

**Ans.** *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.

render() {

return <span className="menu navigation-menu">Menu</span>

}

**Q.9. What is Relay?**

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

**Q.10. How Relay is different from Redux?**

**Ans.** 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.

**Q.11. What is Combine Reducer?**

**Ans.** The combineReducers helper function turns an object whose values are different reducing functions into a single reducing function you can pass to createStore . The resulting reducer calls every child reducer, and gathers their results into a single state object.

React:

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

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

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

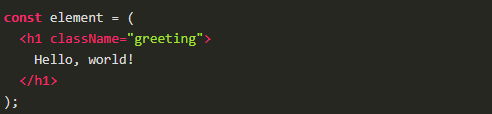
The Virtual DOM is an abstraction of the HTML DOM. It is lightweight and detached from the browser-specific implementation details. It is not invented by React but it uses it and provides it for free. ReactElements lives in the virtual DOM. They make the basic nodes here. Once we defined the elements,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

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

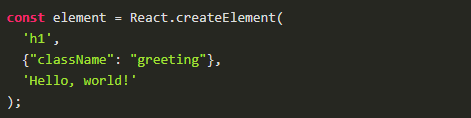
**Q2. What is JSX?**

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

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



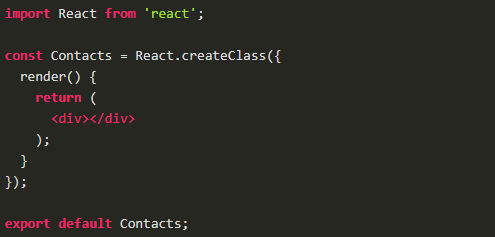
Equivalent of the above using React.createElement



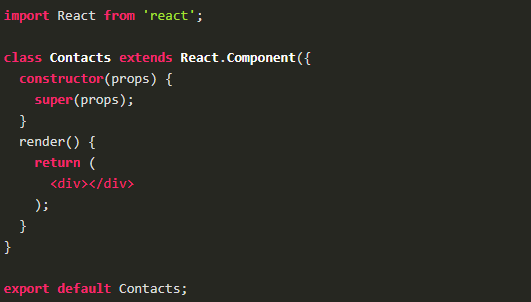
**Q3. What is React.createClass?**

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

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



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



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

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

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

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

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

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

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

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

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



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

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

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

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

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



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

**Q7. What are controlled components?**

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

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

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

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

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

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

**Q9. What is create-react-app?**

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

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

https://miro.medium.com/max/60/1*2jC_2JPgxKTEN5bvp3SrUw.png?q=20

https://miro.medium.com/max/651/1*2jC_2JPgxKTEN5bvp3SrUw.png

It includes everything we need to build a React app:

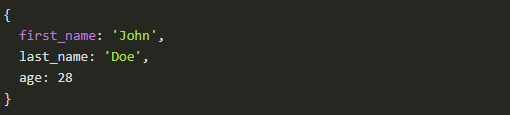
* React, JSX, ES6, and Flow syntax support.
* Language extras beyond ES6 like the object spread operator.
* Autoprefixed CSS, so you don’t need -webkit- or other prefixes.
* A fast interactive unit test runner with built-in support for coverage reporting.
* A live development server that warns about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

**Q10. What is Redux?**

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

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

https://miro.medium.com/max/60/1*UixQj8JaMzLmXOsHocWYeA.png?q=20

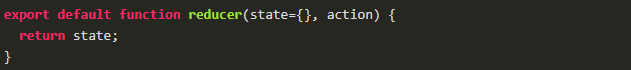


All of these data is retained by redux in a closure that redux calls a store . It also provides us a recipe of creating the said store, namely createStore(x).

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

This is a valid example reducer function:

https://miro.medium.com/max/60/1*ejuTaJ-mXMWnPVn8ItwFPA.png?q=20



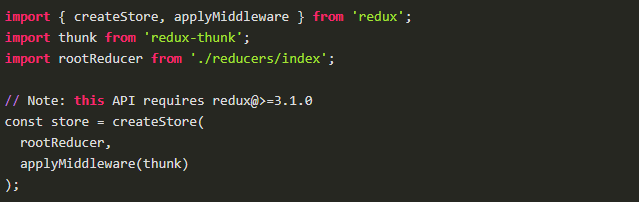
This store can only be updated by dispatching an action. Our App dispatches an action, it is passed into reducer; the reducer returns a fresh instance of the state; the store notifies our App and it can begin it's re render as required.

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

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

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

https://miro.medium.com/max/60/1*cCuCnVMum5Y453iFjYQKwQ.png?q=20



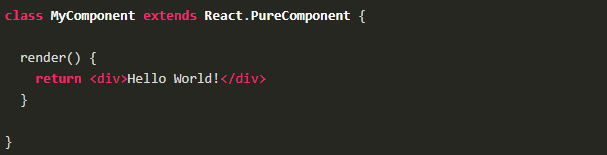
**Q12. What is PureComponent? When to use PureComponent over Component?**

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

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

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

https://miro.medium.com/max/60/1*iisL6VZlGGpO2W5wAKTb_g.png?q=20



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Redux is composed of the following components:

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

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

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

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

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

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

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

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

**1. How is Stateless component different from a Stateful component?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

The stateless component calculates the internal state of the component but does not have the authority to change state. There is no knowledge about the past, current, or future but receives props from the Stateful component, which are treated as a callback function.

**2. What is the difference between state and props?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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. ie:

* Props get passed to the component similar to function parameters.
* State is managed within the component similar to variables declared within a function.

**3. What can you do if the expression contains more than one line?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

In such a situation, enclosing the multi-line JSX expression is an option. If you are a first time user, it may seem awkward but later you can understand everything very easily. Many times it becomes necessary to avoid multi-lines to perform the task reliably and for getting the results as expected.

**4. What kind of information controls a segment in React?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

There are mainly two sorts of information that control a segment: State and Props.

* State: State information that will change, we need to utilize State.
* Props: Props are set by the parent and which are settled all through the lifetime of a part.

**5. Explain DOM Diffing in React.**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

The process of checking the difference between the new VDOM tree and the old VDOM tree is called "diffing". Diffing is accomplished by a heuristic O(n) algorithm. During this process, React will deduce the minimum number of steps needed to update the real DOM, eliminating unnecessary costly changes. This process is also referred to as reconciliation.

**6. What do you understand by the term polling?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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

**7. What’s the difference between an Element and a Component in React?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Simply put, a React element describes what you want to see on the screen. Not so simply put, a React element is an object representation of some UI.

A React component is a function or a class which optionally accepts input and returns a React element (typically via JSX which gets compiled to a createElement invocation).

**8. Why should we not call setState in componentWillUnmount?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

You should not call setState() in componentWillUnmount() because the component will never be re-rendered.

**9. What is the difference between React Native and React?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**10. What are React Hooks?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class. With Hooks, you can extract stateful logic from a component so it can be tested independently and reused. Hooks allow you to reuse stateful logic without changing your component hierarchy. This makes it easy to share Hooks among many components or with the community.

**11. When would you use a Class Component over a Functional Component?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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

**12. How to prevent a function from being called multiple times?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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:

* Throttling: Changes based on a time based frequency. For example, it can be used using \_.throttlelodash function.
* Debouncing: Publish changes after a period of inactivity. For example, it can be used using \_.debouncelodash function.
* RequestAnimationFrame throttling: Changes based on requestAnimationFrame. For example, it can be used using raf-schdlodash function.

**13. Describe how events are handled in React.**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

The event handlers in React will be passed instances of SyntheticEvent to solve cross-browser compatibility issues. As we mentioned earlier, SyntheticEvent is React’s cross-browser wrapper around the browser’s native event. The synthetic events have the same interface as the native ones but they work identically across all browsers.

However, React doesn’t actually attach events to the child nodes themselves. Instead, it uses a single event listener in order to listen to all events at the top level which. Not only is this great for the performance but it also means that React doesn’t have to keep track of the event listeners when updating the DOM.

**14. How to dispatch the data in-store?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

We can dispatch the data to another component which should be based on the action which stores the parent component.

**15. What do you understand by “Single source of truth”?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Single source of truth (SSOT) is the practice of structuring information models and associated data schema such that every data element is mastered (or edited) in only one place. Redux uses Store to store the entire state of the application at one location. So all the state of the component is stored in the store and the store itself receives updates. The single state tree makes tracking modifications simpler over time and debugging or inspecting the request.

**16. Why is switch keyword used in React Router v4?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Within the Switch component, Route and Redirect components are nested inside. Starting from the top Route/Redirect component in the Switch, to bottom Route/Redirect, each component is evaluated to be true or false based on whether or not the current URL in the browser matches the path/from prop on the Route/Redirect component. Switch is that it will only render the first matched child.

**17. How do you tell React to build in Production mode and what will that do?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

You set process.env.NODE\_ENV to production. When React in production mode, it’ll strip out any extra development features like warnings.

**18. How do you access imperative API of web components?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**19. What is the benefit of strict mode?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

It activates additional checks and warnings for its descendants. Note: Strict mode checks are run in development mode only; they do not impact the production build.

It is helpful in the following cases:

* Identifying components with unsafe lifecycle methods.
* Warning about legacy string ref API usage.
* Detecting unexpected side effects.
* Detecting legacy context API.
* Warning about deprecated findDOMNode usage.

**20. What is the difference between createElement and cloneElement?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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

**21. What are the features of ReactJS?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

The features of React JS are as follows:

1. React improves [SEO](https://breadnbeyond.com/video-marketing/seo-video/) performance: React boost the performance of the SEO to higher levels as a search engine faces the problem while reading JavaScript of high loaded applications.
2. React acts as a standard for mobile app development: It provides a transition process as an ideal solution for both mobile and web applications for building rich user interfaces.
3. React makes the process of writing components easier: Using React along with JSX will make you write components and code efficiently and clearly.
4. React increases efficiency: As the React boost the efficiency of components by reusing them. This is the reason why it is considered as an ideal feature of React. It is considered as the most reusable system component.
5. React ensures stable code: It ensures the stability of the code of an application by making use of downward dataflow.

**22. What are the rules one needs to follow regarding hooks?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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

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

**23. What are the conditions to safely use the index as a key?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

There are three conditions to make sure, it is safe use the index as a key.

* The list and items are static– they are not computed and do not change.
* The items in the list have no ids.
* The list is never reordered or filtered.

**24. Explain Presentational segment**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

A presentational part is a segment which allows you to renders HTML. The segment’s capacity is presentational in markup.

**25. What is Relay?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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

**26. Can we make changes inside child components?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Yes, we can make changes inside the child component in Props but not in the case of States.

**27. What is route based code splitting?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

A good place to start code splitting is with app routes. Break down an application into chunks per route, and then load that chunk when user navigate that route. Under the hood, webpack takes care of creating chunks and serve chunks to the user on demand.

We have to just create asyncComponent and import the desired component by using dynamic import() function.

**28. Explain the purpose of render() in React.**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**29. What is render hijacking?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**30. What are React Events?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Events are reactions (the library IS called React, right?) that are triggered by specific user actions like clicking on a UI button, hovering a mouse over a UI object, using keyboard commands with the UI, etc.

**31. What do you know about Flux?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Basically, Flux is a basic illustration that is helpful in maintaining the unidirectional data stream. It is meant to control construed data unique fragments to make them interface with that data without creating issues. Flux configuration is insipid; it's not specific to React applications, nor is it required to collect a React application. Flux is basically a straightforward idea, however in you have to exhibit a profound comprehension of its usage.

**32. What is ReactDOMServer?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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:

* renderToString()
* renderToStaticMarkup()

**33. How do the parent and child components exchange information?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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

**34. Where would you put AJAX calls in your React code?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

It is possible to use any AJAX library with React, such as Axios, jQuery AJAX, as well as the inbuilt browser window.fetch. Data with AJAX calls need to be added to the componentDidMount() lifecycle method. By doing this, it is possible to use the setState() method for updating component as soon as the data is retrieved.

**35. What is the use of Webpack?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Webpack in general is a module bundler, thanks to many plugins it provides although a lot more and it can be used to run tasks, clean build directories, check linting, handle typescript support, increase performance, provide chunking and a lot more.

**36. What is the difference between DOM and virtual DOM in React.js?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

DOM aka Document Object Model is an abstraction of structured code (HTML). Dom and HTML code are interrelated as the elements of HTML are known as nodes of DOM. It defines a structure where users modify the content present in the structure in any way they want (create, edit, alter, modify etc.). Basically, HTML is a text, DOM is an in-memory representation of this text.

Virtual DOM is a representation of DOM objects like a lightweight copy. It is used and provided for free by React.js

**37. Why do we need a Router to React?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

We need a Router to React so that we could define the multiple routes whenever the user types a particular URL. This way, the application of a particular router can be made when the URL matches the path defined inside the router.

**38. Why did you choose to work with react?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

This kind of question is less about rattling off facts related to JSX, Virtual DOMs, props, or state, and more about explaining to an employer why you have a professional interest in working with React JS. One of the main reasons this question (and your answer) is of interest to an interviewer is because it gives a sense of how you might explain the importance of using React to a non-technical client or stakeholder.

To answer, simply think of what drew you to React JS. It can be something as basic as the fact that React is easy to learn and start with, but allows plenty of room for growth over time (showing your willingness to learn new things and expand your knowledge as you go), or something as practical as the fact that there are so many job opportunities for React developers (showing you keep on top of the industry and are able to adapt as needed).

**39. List down the advantages of React Router.**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

* 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 () in which we enclose the specific child routes ().
* No need to manually set History value: In React Router v4, all we need to do is wrap our routes within the component.
* The packages are split: Three packages one each for Web, Native and Core. This supports the compact size of our application. It is easy to switch over based on a similar coding style.

**40. What is create-react-app?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

create-react-app is the official CLI (Command Line Interface) for React to create React apps with no build configuration. We don’t need to install or configure tools like Webpack or Babel. They are preconfigured and hidden so that we can focus on the code. We can install easily just like any other node modules.

**41. What are some of the major advantages to using react when building UIs?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Some of the major advantages of using React include:

* Increased application performance via the Virtual DOM model.
* Improved coding efficiency with JSX.
* The ability to reuse components across multiple projects.
* Flexibility and extensibility through add-on tools provided by React’sopen source community.

**42. Why are you not required to use inheritance?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**43. How is React different from Angular and VUE?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

The core difference between React and Vue is that React lacks any form of “abstraction”. It’s very much just straight JavaScript. This brings some of the drawbacks of of JS. If you’re a JS expert, React will give you more power. But if you are lacking the expertise, Vue will smooth some of the rough patches for you. It’s also worth noting that Vue doesn’t work with Arrow functions in the same way React does.

VUE.js was launched in 2014 and since then, it has been the most rapidly growing js framework. It is particularly useful for building intuitive interfaces while also being extremely adaptable. VUE is a web application framework that helps in making advanced single page applications.

Angular is a typescript based JavaScript application framework developed by Google, not a collection of libraries and it relies more on HTML than on JS. Despite the slowdown in recent years it’s actually used very widely for government and enterprise projects, which depend on a stable, well-established, and consistent ecosystem. It is also known as Super-heroic JavaScript MVW Framework. Its initial purpose was to encounter the challenges of creating single page apps. AngularJS is the oldest version of the Angular framework.

**44. What is the second argument that can optionally be passed to setState and what is its purpose?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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

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

**45. What are Higher Order Components(HOC)?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**46. What is suspense component?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Suspense is a component that wraps your own custom components. It lets your components communicate to React that they're waiting for some data to load before the component is rendered. It is important to note that Suspense is not a data fetching library like react-async, nor is it a way to manage state like Redux.

**47. Is it possible to display props on a parent component?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Yes, it is possible. The best way to perform this task is by using the spread operator. It can also be done with listing the properties but this is a complex process.

**48. Name 3 ways to create a component in React and its differences.**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

There are 3 main ways of creating a component. Extending from the Component class, extending from the PureComponent class or using a stateless function as component. The main difference is that we don’t have access to lifecycle methods in a stateless function nor to the state. PureComponent also implements componentShouldUpdate by default and provides a shallow compare for its props and state preventing unnecessary re-renders.

**49. Is setState() async? Why?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

setState() actions are indeed asynchronous. setState() doesn’t immediately mutate this.state. Instead, it creates a pending state transition. Accessing this.state after calling this method can potentially return the existing value. There is no guarantee of synchronous operation of calls to setState and calls may be batched for performance gains.

The reason behind is the way setState alters the state and causes rerendering. Making it synchronous might leave the browser unresponsive. That being said, thesetState calls are asynchronous as well as batched for better UI experience and performance.

**50. What is the point of renderToNodeStream method?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

It is used to render a React element to its initial HTML. Returns a Readable stream that outputs an HTML string. The HTML output by this stream is exactly equal to what ReactDOMServer.renderToString would return. You can use this method to generate HTML on the server and send the markup down on the initial request for faster page loads and to allow search engines to crawl your pages for SEO purposes.

**51. What is prop drilling and how can you avoid it?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

When building a React application, there is often the need for a deeply nested component to use data provided by another component that is much higher in the hierarchy. The simplest approach is to simply pass a prop from each component to the next in the hierarchy from the source component to the deeply nested component. This is called prop drilling. The primary disadvantage of prop drilling is that components that should not otherwise be aware of the data become unnecessarily complicated and are harder to maintain. To avoid prop drilling, a common approach is to use React context. This allows a Provider component that supplies data to be defined, and allows nested components to consume context data via either a Consumer component or a useContext hook.

**52. What do you understand by “In React, everything is a component.”**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Building blocks of the UI of a React application are components. These parts divide the entire UI into tiny parts that are autonomous and reusable. Then it becomes independent of each of these parts without affecting the remainder of the UI.

**53. How do you say that state updates are merged?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

State update are merged means that when you update only one key in the object state it will not affect the other keys. and key2 would not exist anymore in your state as it was not defined in your update. The merging affects key/value pair which are not included in your update.

**54. What is the behavior of uncaught errors in react 16?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**55. How does JSX prevent Injection Attacks?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

By default, 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. This helps prevent XSS (cross-site-scripting) attacks.

**56. What do you understand by mixin or higher order components in ReactJS?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**57. What do you understand by Props in React?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Prop is a contraction for Properties in React. These read-only components need to be kept immutable i.e. pure. Throughout the application, props are passed down from the parent components to the child components. In order to maintain the unidirectional data flow, a child component is restricted from sending a prop back to its parent component. This also helps in rendering the dynamically generated data.

**58. How is Virtual-DOM more efficient than Dirty checking?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

First thing to understand here is that in React, each component has a state which is observable. React knows when to re-render the scene because it is able to observe when this data changes. The observables are significantly faster than the Dirty checking because we don’t have to poll the data at a regular interval and check all of the values in the data structure recursively. By comparison, setting a value on the state will signal to a listener that some state has changed. In a situation like that, React can simply listen for change events on the state and queue up re-rendering. Long story short, the virtual DOM is more efficient than the Dirty checking simply because it prevents all the unnecessary re-renders. Re-rendering only occurs when the state changes.

**59. Is it mandatory to define constructor for React component?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**60. What is the difference between a controlled component and an uncontrolled component?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

A large part of React is this idea of having components control and manage their own state. What happens when we throw native HTML form elements (input, select, textarea, etc) into the mix? Should we have React be the “single source of truth” like we’re used to doing with React or should we allow that form data to live in the DOM like we’re used to typically doing with HTML form elements? These two questions are at the heart of controlled vs. uncontrolled components.

A controlled component is a component where React is in control and is the single source of truth for the form data. As you can see below, username doesn’t live in the DOM but instead lives in our component state. Whenever we want to update username, we call setState as we’re used to.

**61. What are the rules needs to follow for hooks?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

You need to follow two rules inorder to use hooks:

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

**62. React has something called a state. What is it and how it is used?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

States are the source of data for React components. In other words, they are objects responsible for determining components behavior and rendering. As such, they must be kept as simple as possible. Accessible by means of this.state(), state is mutable and creates dynamic and interactive components.

**63. What is JSX?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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

**64. Explain the use of Redux thunk?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Redux thunk acts as middleware which allows an individual to write action creators that return functions instead of actions. This is also used as a delay function in order to delay dispatch of an action if a certain condition is met. The two store methods getState() and dispatch() are provided as parameters to the inner function.

**65. Is setState() is async? Why is setState() in React Async instead of Sync?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

setState() actions are asynchronous and are batched for performance gains.

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

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

**66. What is the difference between async mode and concurrent mode?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Both refer to 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.

**67. What does shouldComponentUpdate do and why is it important?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

What shouldComponentUpdate does is it’s a lifecycle method that allows us to opt out of this reconciliation process for certain components (and their child components). Why would we ever want to do this? As mentioned above, “The end goal of reconciliation is to, in the most efficient way possible, update the UI based on new state”. If we know that a certain section of our UI isn’t going to change, there’s no reason to have React go through the trouble of trying to figure out if it should. By returning false from shouldComponentUpdate, React will assume that the current component, and all its child components, will stay the same as they currently are.

**68. Explain React Decorators**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Decorators in React help you take an existing Class component, or function of a Class component, and modify it, thereby allowing you to add extra capabilities, without having to mess with the existing codebase. Modification can be overriding the existing function completely, or just adding extra logic to it. In essence, decorators take one function, and return another function after spicing it up. React and decorators can go hand-in-hand due to presence of Higher Order Components , and Higher Order Functions.

**69. What are the different phases of React component’s lifecycle?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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

**70. What is the purpose of eslint plugin for hooks?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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, - Calls to Hooks are either inside a PascalCase function (assumed to be a component) or another useSomething function (assumed to be a custom Hook). - Hooks are called in the same order on every render.

**71. How would you debug an issue in react code? What debugging tools have you used?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Debugging as a crucial part of the development process. Before you start your next React JS job interview, make sure you have experience with the following industry standard debugging tools (and can explain how you’d use them):

* Linters (eslint, jslint)
* Debuggers (React Developer Tools)

**72. What are the lifecycle methods of ReactJS?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

* 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 eventListeners 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 rerender if component receives new prop.
* componentWillUpdate: Executed before re-rendering the component when there are pros & 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.

**73. How is ReactJs different from AngularJS?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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

**74. What would be two of the most significant drawbacks of React?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

* Integrating React with the MVC framework like Rails requires complex configuration.
* React requires the users to have knowledge about the integration of user interface into MVC framework.

**75. What are synthetic events in React?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**76. Can you force a React component to rerender without calling setState?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

In your component, you can call this.forceUpdate() to force a rerender.

**77. What is arrow function in React? How is it used?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**78. Mention the key benefits of Flux?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Applications that are built on Flux have components that can simply be tested. By simply updating the store, developers are able to manage and test any react component. It cut down the overall risk of data affection. All the applications are highly scalable and suffer no compatibility issues.

**79. Why are fragments better than container divs?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

* It’s a tiny bit faster and has less memory usage (no need to create an extra DOM node). This only has a real benefit on very large and/or deep trees, but application performance often suffers from death by a thousand cuts. This is one cut less.
* Some CSS mechanisms like Flexbox and CSS Grid have a special parent-child relationship, and adding divs in the middle makes it hard to keep the desired layout while extracting logical components.
* The DOM inspector is less cluttered.

**80. What are refs in React?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**81. Is it ref argument available for all functions or class components?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**82. In ReactJS, why there is a need to capitalize on the components?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

It is necessary because components are not the DOM element but they are constructors. If they are not capitalized, they can cause various issues and can confuse developers with several elements. At the same time, the problem of integration of some elements and commands can be there.

**83. What are the benefits of using typescript with reactjs?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Below are some of the benefits of using typescript with ReactJS:

* It is possible to use latest JavaScript features.
* Use of interfaces for complex type definitions.
* IDEs such as VS Code was made for TypeScript.
* Avoid bugs with the ease of readability and Validation.

**84. How would you structure a React application?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

This is an open question with many possible answers. The basic structure is usually module or feature based. We usually differentiate between UI and logic. There are many approaches to structure UI components with the most popular being atomic design. Data and business heavy applications use a more domain driven approach. The ideal combination for larger applications is having the domain logic separate and having the UI logic in an atomic structure. All this can be combined in features which are rendered on pages.

**85. What is the use of a super keyword in React?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

The super keyword helps you to access and call functions on an object’s parent.

**86. Why are String Refs considered legacy?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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

**87. When should you use the top-class elements for the function element?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

If your element does a stage or lifetime cycle, we should use top-class elements.

**88. What is the methods order when component is re-rendered?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

* static getDerivedStateFromProps()
* shouldComponentUpdate()
* render()
* getSnapshotBeforeUpdate()
* componentDidUpdate()

**89. Explain the Virtual DOM and its working.**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

A virtual DOM is a lightweight JS object. It is simply a copy of the real DOM. A virtual DOM is a node tree that lists various elements, their attributes, and content as objects and their properties.

The render() function in React is responsible for creating a node tree from the React components. This tree is then updated in response to the mutations resulting in the data model due to various actions made by the user or the system.

Virtual DOM operates in three simple steps:

* Step 1 – The entire UI is re-rendered in Virtual DOM representation as soon as there are some underlying data changes.
* Step 2 – Now, the difference between the previous DOM representation and the new one (resulted from underlying data changes) is calculated.
* Step 3 – After the calculations are successfully carried out, the real DOM is updated in line with only the things that actually underwent changes.

**90. Is it possible to nest JSX elements into other JSX elements?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

It is possible. The process is quite similar to that of nesting the HTML elements. However, there are certain things that are different in this. You must be familiar with the source and destination elements to perform this task simply.

**Advanced**

**1. How does React know when to re-render App component if we handle window resizing in useWindowSize?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

When you call setSize inside the custom hooks, React knows that this hook is used in App component and will re-render it.

**2. Explain Flexbox and its benefits**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Flexbox is a layout-ing method which solves many previous problems with handling layouts in CSS. It eliminates the need for different GRID libraries and using floats to position blocks on the site. It has a very intuitive api and gives a lot more control and flexibility.

**3. Why do we need a key property? Give an example when a bad key causes an error.**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

There are classic diffing algorithms with O(n³) time complexity, which might be used for creating a tree of React elements. But it means for displaying 1000 elements would require one billion comparisons.

Instead, React implements a heuristic O(n) algorithm with an assumption that the developer can hint at which child elements may be stable across different renders with a keyprop.

What about a bad key? Well, an index might be a very bad key if you decide to make your children removable. Check out this demo. Try to type something in the second input and then remove the first one. But you still can see the value in the second one, why so?

Because your keys are unstable. After removal, your third child with a key equals to 3, now has a key equals to 2. It’s not the same element for React now. And it will match it to the wrong DOM element, which previously had a key equals to 2 (which keeps the value we typed in a second input).

**4. React unit tests vs integration tests for components.**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

It’s worth mentioning both Enzyme and react-testing-library.

React testing library provides a clean and simple API which focuses on testing applications “as a user would”. This means an API returns HTML Elements rather than React Components with shallow rendering in Enzyme. It’s is a nice tool for writing integrational tests.

Enzyme is still a valid tool, it provides a more sophisticated API which gives you access to component’s props and internal state. It makes sense to create unit tests for components.

**5. What is windowing technique?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**6. Explain the positives and negatives of shallow rendering components in tests.**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

**Positives:**

* It is faster to shallow render a component than to fully render it. When a React project contains a large number of components, this performance difference can have a significant impact on the total time taken for unit tests to execute.
* Shallow rendering prevents testing outside the boundaries of the component being tested—a best practice of unit testing.

**Negatives:**

* Shallow rendering is less similar to real-world usage of a component as part of an application, so it may not catch certain problems. Take the example of a <House /> component that renders a <LivingRoom /> component. Within a real application, if the <LivingRoom /> component is broken and throws an error, then <House /> would fail to render. However, if the unit tests of <House /> only use shallow rendering, then this issue will not be identified unless <LivingRoom /> is also covered with unit tests.

**7. What are the problems of using render props with pure components?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**8. Do you know what the reconciliation algorithm is?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

It is the algorithm responsible for figuring out what changed between re-renders and how to update the actual DOM. It is basically a diffing algorithm. The latest addition of improvements on the core algorithm is called React Fiber.

**9. How to prevent components from re-rendering?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

* shouldComponentUpdate() — returns ‘true’ by default. You can override if you know which props have to trigger an update.
* PureComponents — The difference between them is that React.Component doesn’t implement shouldComponentUpdate method but React.PureComponentimplements it with a shallow prop and state comparison.
* React.memo — The same as the previous one but it works with functional components.

**10. How would you optimise the performance of a React application?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

The most expensive task in a React app is the update of the DOM. The basic optimisation is to reduce how many times a component re-renders. This can be achieved by using componentShouldUpdate, using PureComponent or memoization libraries like reselect. Reducing the size of the final JS file also helps improving performance and we can use dynamic imports and chunks for this.

**11. What are the drawbacks of MVW pattern?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

* DOM manipulation is very expensive which causes applications to behave slow and inefficient.
* Due to circular dependencies, a complicated model was created around models and views.
* Lot of data changes happens for collaborative applications(like Google Docs).
* No way to do undo (travel back in time) easily without adding so much extra code.

**Redux**

**1. What are the advantages of formik over redux form library?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Below are the main reasons to recommend formik over redux form library:

* The form state is inherently short-term and local, so tracking it in Redux (or any kind of Flux library) is unnecessary.
* Redux-Form calls your entire top-level Redux reducer multiple times ON EVERY SINGLE KEYSTROKE. This way it increases input latency for large apps.
* Redux-Form is 22.5 kB minified gzipped whereas Formik is 12.7 kB

**2. Can Redux only be used with React?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**3. How Relay is different from Redux?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**4. What are selectors? Why would you use reselect or a memoization library?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Selectors are functions which accept the state and return a portion of it while applying calculations, transformations, mappings, filtering etc. This way the logic of how to retrieve data for a specific view is encapsulated in a selector. Since many of the mentioned operations are expensive, when calling the selector again without state change, you want to skip the expensive operations as they will return the same results and hence the usage of reselect. Reselect will return the results from the Cashe in case arguments didn’t change.

**5. What is the mental model of redux-saga?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**6. What is an action in Redux?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

It is a function which returns an action object. The action-type and the action data are always stored in the action object. Actions can send data between the Store and the software application. All information retrieved by the Store is produced by the actions.

**7. Why are Redux state functions called reducers?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

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.

**8. What are the core principles of Redux?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

Redux follows three fundamental principles:

* 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.
* 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.
* 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.

**9. What are the downsides of Redux compared to Flux?**[**↑**](https://www.adaface.com/blog/react-interview-questions/#questionsindex)

* 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 assumes 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.
* 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.
* There is no nice Flow integration yet: Flux currently lets you do very impressive static type checks which Redux doesn't support yet.

#### 1. Benefits of Redux?

**Answer:**

* **Maintainability:** The maintenance of Redux becomes easier due to strict code structure and organization.
* **Organization:** code organization is very strict; hence the stability of the code is high, which intern increases the work to be much easier.
* **Server rendering:** This is useful, particularly to the preliminary render, which keeps up a better user experience or search engine optimization. The server-side created stores are forwarded to the client-side.
* **Developer tools:** It is Highly traceable, so changes in position and changes in the application; all such instances make the developers have a real-time experience.
* **Ease of testing:** The first rule of writing testable code is to write small functions that do only one thing and that are independent. Redux’s code is made of functions that used to be: small, pure, and isolated.

#### Q2. How Distinct from MVC and Flux?

**Answer:**  
As far as MVC structure is concerned, the data, presentation, and logical layers are well separated and handled. Change to an application even at a smaller position may involve many changes through the application. this happens because data flow exists bidirectional as far as MVC is concerned. Maintenance of MVC structures are hardly complex, and Debugging also expects a lot of experience for it.

Flux stands closely related to redux. A story based strategy allows capturing the changes applied to the application state, the event subscription, and the current state are connected by means of components. Call back payloads are broadcasted by means of Redux.

#### Q3. Functional programming concepts?

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

* Functions are treated as First-class objects.
* Capable of passing functions in the format of arguments.
* Capable of controlling flow using recursions, functions, and arrays.
* Helper functions such as reduce and map filters are used.
* Allows linking functions together.
* The state doesn’t change.
* Prioritize the order of executing the code is not really necessary.

#### Q4. Redux change of state?

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

**Example:**

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

#### Q5. Where can Redux be used?

**Answer:**  
Redux is majorly used in combination with reacting. It also has the ability to get used to other view libraries too. Some of the famous entities like AngularJS, Vue.js, and Meteor. It can get combined with Redux easily. This is a key reason for the popularity of Redux in its ecosystem. So many articles, tutorials, middleware, tools, and boilerplates are available.

### Part 2 – Redux Interview Questions (Advanced)

Let us now have a look at the advanced Interview Questions.

#### Q6. What is the typical flow of data in a React + Redux app?

**Answer:**  
Call-back from the UI component dispatches an action with a payload; these dispatched actions are intercepted and received by the reducers. This interception will generate a new application state. From here, the actions will be propagated down through a hierarchy of components from the Redux store. The below diagram depicts the entity structure of a redux+react setup.

#### Q7. What is store in redux?

**Answer:**  
The store holds the application state and supplies the helper methods for accessing the state.  
Register listeners and dispatch actions. There is only one Store while using Redux. The store is configured via the createStorefunction. The single store represents the entire state. R  
ducers return a state via action.

export function configureStore(initialState) {  
return createStore(rootReducer, initialState);  
}

The root reducer is a collection of all reducers in the application.

const rootReducer = combineReducers({  
donors: donorReducer,  
});

Let us move to the next Redux Interview Questions.

#### Q8. Explain Reducers in Redux?

**Answer:**  
The state of a store is updated by means of reducer functions. A stable collection of reducers form a store, and each of the stores maintains a separate state associated with itself. To update the array of donors, we should define a donor application.  
The reducer as follows.

export default function donorReducer(state = [], action) {  
switch (action.type) {  
case actionTypes.addDonor:  
return [...state, action.donor];  
default:  
return state;  
}  
}

The reducers receive the initial state and action. Based on the action type, it returns a new state for the store. The state maintained by reducers is immutable. The below-given reducer it holds the current state and action as an argument for it and then returns the next.

state:functionhandelingAuthentication(st, actn)  
{  
return \_.assign({}, st,  
{  
auth: actn.pyload  
});  
}

#### Q9. Redux workflow features?

**Answer:**

* **Reset:** Allow to reset the state of the store
* **Revert:** Rollback to the last committed state
* **Sweep:** All disabled actions that you might have fired by mistake will be removed
* **Commit:** Makes the current state the initial state

#### Q10. Explain action’s in Redux?

**Answer:**  
Actions in Redux are functions that return an action object. The action-type and the action data are packed in the action object. Which also allows a donor to be added to the system. Actions send data between the store and the application. The actions produce all information retrieved by the store.

export function addDonorAction(donor) {  
return {  
type: actionTypes.addDonor,  
donor,  
};  
}

* 1. **What are the core principles of Redux?**

There are three core principles that Redux follows:

* Single source of truth: The global state of your app is put away in an object tree inside a single store.
* The state is read-only: State can only be changed by emitting an action, an object that explains what has happened.
* Changes are made with pure functions: This is to define how the state tree is being transformed by the actions, you have to write pure reducers.
* **2. What is the difference between mapStateToProps() and mapDispatchToProps()?**

|  |  |
| --- | --- |
| **mapStateToProps()** | **mapDispatchToProps()** |
| It is a function that is used to provide the stored data to the component. | It is a function that is used to provide the action creators with props to the component. |
| All the results of mapStateToProps() should be the plain object that will later be merged into the component’s prop. | By mapDispatchToProps(), all the action creators are wrapped in the dispatcher call so that they may be called upon directly and will be merged into the component’s prop. |
| It is used to connect the redux state to the props of the react component. | It is used to connect redux actions to the react props. |

Q..Do you need to keepIs all component states in Redux store?

**Answer**

You need to keep your application state as small as possible. You don't have to push everything in there. Only do it makes a lot of sense to keep something there Or if it makes your life easier when using Dev Tools. But we shouldn't overload its importance too much.

Q.What is Redux DevTools?

**Answer**

**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.

Q.How to add multiple middlewares to Redux?

**Answer**

You can use applyMiddleware where you can pass each piece of middleware as a new argument. So you just need to pass each piece of middleware you’d like. For example, you can add Redux Thunk and logger middlewares as an argument as below,

## Q. What are the core principles of Redux?

Answer

Redux follows three fundamental principles: 1. **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. 2. **State is ready 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. 3. **Changes are made with pure functions:** To specify how the state tree is transformed by actions, you write pure reducers(Reducers are just pure functions that take the previous state and an action, and return the next state).

Q.What are the features of Redux DevTools?

**Answer**

Below are the major features of Redux devTools 1. Lets you inspect every state and action payload 2. Lets you go back in time by “cancelling” actions 3. If you change the reducer code, each “staged” action will be re-evaluated 4. If the reducers throw, you will see during which action this happened, and what the error was 5. With persistState() store enhancer, you can persist debug sessions across page reloads

Q.What is Redux Thunk?

**Answer**

**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.

Q.What is redux-saga?

**Answer**

**redux-saga** is a library that aims to make side effects (i.e. 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 as

npm install --save redux-saga

## Q. What is the difference between React context and React redux?

**Answer**

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. So you should feel much safer using context via React Redux than directly because if it changes, the burden of updating the code will be on React Redux instead developer responsibility.

Q.What are the differences between redux-saga and redux-thunk?

**Answer**

Both **Redux Thunk** and **Redux Saga** take care of dealing with side effects. In most of the scenarios, Thunk allows *Promises* to deal with them, whereas Saga uses *Generators*.

Thunk is simple to use and Promises are familiar to many developers, Saga/Generators are more powerful but you will need to learn them. But both the two middleware can coexists, so you can start with Thunks and introduce Sagas when/if you need them.

## Q. What are the differences between call and put in redux-saga?

**Answer**

Both **call** and **put** are effects creators functions.

* **call** function is used to create effect description, which instructs middleware to call the promise.
* **put** function creates effect, in 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 passed to `api.fetchUser` function.

// Instructing middleware to call promise, it resolved value 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

});

}

## Q. What is the proper way to access Redux store?

**Answer**

The best way to access your store through a component is using the connect() function. Actually 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) => {

return {

active: ownProps.filter === state.visibilityFilter

}

}

const mapDispatchToProps = (dispatch, ownProps) => {

return {

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 devs almost always recommend using connect over accessing the store directly (using context API).

class MyComponent {

someMethod() {

doSomethingWith(this.context.store);

}

}

## Q. Why are Redux state functions called as reducers?

**Answer**

**Reducers** always return the accumulation of the state (based on all previous and current actions) not only default values. Therefore, they act as a reducer of state. Each time a redux reducer is called, the state is passed in with the action (state, action). This state is then reduced (or accumulated) based on the action, and then the next state is returned. i.e, 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.

**3. Do you need to keep all component states in the Redux store?**

You do not need to push everything in the redux store as you have to keep your application state as small as possible. You should only do it if it makes a difference to you to keep something there or maybe helping you in making your life easier while using Dev Tools.

**5. What is an action in Redux?**

Actions are the plain JavaScript objects which contain a type field. Action can be considered as an event that can describe something that has happened in the application.Always remember actions should contain a small amount of information that is needed to mention what has happened.

**6. What is “store” in redux?**

The Redux “store” carries together all the states, reducers, and actions that create the app. The store has multiple responsibilities:

* It holds the state of the current application from inside
* With the help of store.getState(); it allows access to the current state.
* With the help of the store.dispatch(action); it allows the state to be updated.
* With the help of the store.subscriber(listener); it allows to register listener callbacks.

**Store Methods**

* getState()
* dispatch(action)
* subscribe(listener)
* replaceReducer(nextReducer)

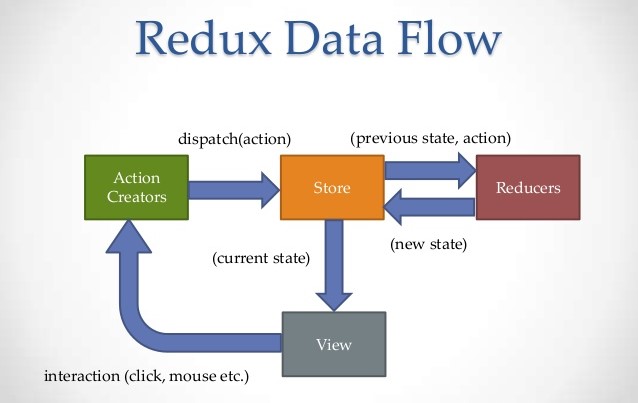
**9. How to access redux stores outside a react component?**

To access the redux stores outside a react component, you need to export the store from the module where it has been created with createStore.

**10. How to structure Redux top-level directories?**

All the applications have multiple top-level directories as mentioned below:

* Components: it is used for “dumb” React components that are unfamiliar with Redux.
* Containers: It is used for “smart” React components which are connected to the Redux.
* Actions: It is used for all the action creators, where the file name should be corresponding to the part of the app.
* Reducers: It is used for all the reducers where the file name is corresponding to the state key.
* Store: it is used for store initialization. This directory works best in small and mid-level size apps.



**12. What are reducers in redux?**

The reducers in redux are the pure functions that take the previous state and an action, and then it returns to the next state.  
(previousState, action) => newState

It is known as the reducer because they are the type of function that would pass to Array.prototype.reduce(reducer, ?initialValue). It is very essential to ensure that the reducer stays pure.

**Q.1. How to create components in React?**

**Ans.** There are two possible ways to create a component.

✅Functional 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>;

}

}

**Q.2. What are the difference between a class component and functional component?**

**Ans.**

✅Class Components

* Class-based Components uses ES6 class syntax. It can make use of the lifecycle methods.
* Class components extend from React.Component.
* In here you have to use this keyword to access the props and functions that you declare inside the class components.

✅Functional Components

* Functional Components are simpler comparing to class-based functions.
* Functional Components mainly focuses on the UI of the application, not on the behavior.
* To be more precise these are basically render function in the class component.
* Functional Components can have state and mimic lifecycle events using Reach Hooks

**Q.3. What is difference between controlled vs uncontrolled component?**

**Ans.**

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

✅Uncontrolled Components  
A Uncontrolled Component is one that stores its own state internally, and you query the DOM using a ref to find its current value when you need it. This is a bit more like traditional HTML.

Example

// Controlled:

<input type="text" value={value} onChange={handleChange} />

// Uncontrolled:

<input type="text" defaultValue="foo" ref={inputRef} />

// Use `inputRef.current.value` to read the current value of <input>

**Q.4. What is children?**

**Ans.** In JSX expressions that contain both an opening tag and a closing tag, the content between those tags is passed to components automatically as a special prop:

props.children

There are some 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 👶

const MainContainer = React.createClass({

render: function () {

return <div>{this.props.children}</div>;

},

});

ReactDOM.render(

<MainContainer>

<span>{'Hello'}</span>

<span>{'World'}</span>

</MainContainer>,

node,

);

**Q.5. What is prop drilling and how can you avoid it?**

**Ans.** While passing a prop from each component to the next in the hierarchy from the source component to the deeply nested component. This is called **prop drilling**.

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

**Q.6. What is Pure Component?**

**Ans.** 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.

**Q.7. Why should we not update the state directly?**

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

//Wrong ❌

this.state.message = 'Not Updated';

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: 'Updated' });

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

**Q.8. What is the purpose of callback function as an argument of setState()**

**Ans.** 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: 'Supi' }, () => console.log('The name has updated and component re-rendered'));

**Q.9. What are synthetic events in React?**

**Ans.** Synthetic Event 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.

**Q.10. What is "key" prop and what is the benefit of using it in arrays of elements 🗝?**

**Ans.** 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:

1. 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.
2. If you extract list item as separate component then apply *keys* on list component instead of li tag.
3. There will be a warning message in the console if the key prop is not present on list items.

**Q.11. Why are String Refs legacy?**

**Ans.** 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**.

1. 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.
2. 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.
3. 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.
4. 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 DataTable 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} />;

}

}

**Q.12. What is the difference between createElement and cloneElement?**

**Ans.** 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.

**Q.13. What is reconciliation?**

**Ans.** 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.

**Q.14. Is lazy function supports named exports?**

**Ans.** 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,

Example:

// FewComponents.js

export const SomeComponent = /\* ... \*/;

export const UnusedComponent = /\* ... \*/;

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

// IntermediateComponent.js

export { SomeComponent as default } from './FewComponents.js';

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

import React, { lazy } from 'react';

const SomeComponent = lazy(() => import('./IntermediateComponent.js'));

**Q.15. What are portals in React?**

**Ans.** 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.

**Q.16. What are stateless components?**

**Ans.** 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.

**Q.17. What are stateful components?**

**Ans.** 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

)

}

**Q.18. What is the impact of indexes as keys?**

**Ans.** 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} />)

}

**Q.19. How do you memoize a component?**

**Ans.** 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);

**Q.20. Why we need to pass a function to setState()?**

**Ans.** 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

**Q.21. Why should component names start with capital letter?**

**Ans.** 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 OneComponent extends Component {

// ...

}

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';

**What are the exceptions on React component naming?**

The component names should start with a uppercase letter but there are few exceptions on this convention. The lowercase tag names with a dot (property accessors) are still considered as valid component names.

For example the below tag can be compiled to a valid component,

render() {

return (

<obj.component /> // `React.createElement(obj.component)`

)

}

**Q.22. Can you force a component to re-render without calling setState?**

**Ans.** 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().

**Q.23. What is the difference between super() and super(props) in React usin ES6 classes?**

**Ans.** 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: 'Supi', ... }

}

}

**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.

**Q.24. Is it mandatory to define constructor for React component?**

**Ans.** 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.

**Q.25. What are default props?**

**Ans.** 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: 'blue',

};

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.

**Q.26. How to apply validation on props in React?**

**Ans.** 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:

1. PropTypes.number
2. PropTypes.string
3. PropTypes.array
4. PropTypes.object
5. PropTypes.func
6. PropTypes.node
7. PropTypes.element
8. PropTypes.bool
9. PropTypes.symbol
10. 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.

**Q.27. Why you can't update props in React?**

**Ans.** 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.

**Q.28. What are render props?**

**Ans.** 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.

**Q.29. What is Suspense component?**

**Ans.** 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.

Example

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

function MyComponent() {

return (

<div>

<Suspense fallback={<div>Loading...</div>}>

<OneComponent />

</Suspense>

</div>

);

}

As mentioned in the above code, Suspense is wrapped above the lazy component.

**Q.30. What is diffing algorithm?**

**Ans.** 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:

1. Two elements of different types will produce different trees.
2. The developer can hint at which child elements may be stable across different renders with a key prop.

**Q.31. How to re-render the view when the browser is resized?**

**Ans.** 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.updateDimensions);

}

updateDimensions() {

this.setState({ width: window.innerWidth, height: window.innerHeight });

}

render() {

return (

<span>

{this.state.width} x {this.state.height}

</span>

);

}

}

**Q.32. What is React memo function?**

**Ans.** 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 \*/

});

**Q.33. What is the methods order when component re-rendered?**

**Ans.** 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.

1. static getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

**Q.34. What are loadable components?**

**Ans.** 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

**Q.35. How to pretty print JSON with React?**

**Ans.** 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'));

**Q.36. What is render hijacking in react?**

**Ans.** 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.

**Q.37. How to use https instead of http in create-react-app?**

**Ans.** 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

**Q.38. How can we convert functional component to pure component?**

**Ans.** We can convert functional to pure component using React.memo.

**Redux Interview Questions 👩🏻‍💻**

[**Q.1. What are reducers in redux?**](https://dev.to/suprabhasupi/react-redux-interview-questions-with-answers-13ba#1-reducers)  
[**Q.2. How is state changed in redux?**](https://dev.to/suprabhasupi/react-redux-interview-questions-with-answers-13ba#2-state-changed-in-redux)  
[**Q.3. How Redux Form initialValues get updated from state?**](https://dev.to/suprabhasupi/react-redux-interview-questions-with-answers-13ba#3-initialvalues-updated-from-satte)  
[**Q.4. What is Redux Thunk?**](https://dev.to/suprabhasupi/react-redux-interview-questions-with-answers-13ba#4-redux-thunk)  
[**Q.5. What is the difference between mapStateToProps() and mapDispatchToProps()?**](https://dev.to/suprabhasupi/react-redux-interview-questions-with-answers-13ba#5-mapstatetoprops-vs-mapdispatchtoprops)  
[**Q.6. How to add multiple middlewares to Redux?**](https://dev.to/suprabhasupi/react-redux-interview-questions-with-answers-13ba#6-mutiple-middleware)  
[**Q.7. What is React context vs React redux?**](https://dev.to/suprabhasupi/react-redux-interview-questions-with-answers-13ba#7-react-context-vs-redux)  
[**Q.8. Why React uses className over class attribute?**](https://dev.to/suprabhasupi/react-redux-interview-questions-with-answers-13ba#8-redux-thunk)  
[**Q.9. What is Relay?**](https://dev.to/suprabhasupi/react-redux-interview-questions-with-answers-13ba#9-relay)  
[**Q.10. How Relay is different from Redux?**](https://dev.to/suprabhasupi/react-redux-interview-questions-with-answers-13ba#10-relay-different-from-redux)  
[**Q.11. What is Combine Reducer?**](https://dev.to/suprabhasupi/react-redux-interview-questions-with-answers-13ba#11-combine-reducer)

**Q.1. What are reducers in redux?**

**Ans.** The reducer is a pure function that takes the previous state and an action, and returns the next state.

(previousState, action) => newState

It's very important that the reducer stays *pure*. Things you should never do inside a reducer:

* Mutate its arguments;
* Perform side effects like API calls and routing transitions;
* Call non-pure functions, e.g. Date.now() or Math.random()

**Q.2. How is state changed in redux?**

**Ans.** 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. Instead, they express an intent to transform the state. Because all changes are centralized and happen one by one in a strict order, there are no subtle race conditions to watch out for. As actions are just plain objects, they can be logged, serialized, stored, and later replayed for debugging or testing purposes.

**Q.3. How Redux Form initialValues get updated from state?**

**Ans.** 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.

**Q.4. What is Redux Thunk?**

**Ans.** 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.

**Q.5. What is the difference between mapStateToProps() and mapDispatchToProps()?**

**Ans.**

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,

};

**Q.6. How to add multiple middlewares to Redux?**

**Ans.** You can use applyMiddleware where you can pass each piece of middleware as a new argument. So you just need to pass each piece of middleware you'd like. For example, you can add Redux Thunk and logger middlewares as an argument as below,

import { createStore, applyMiddleware } from 'redux'

const createStoreWithMiddleware = applyMiddleware(ReduxThunk, logger)(createStore);

**Q.7. What is React context vs React redux?**

**Ans.** 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. So you should feel much safer using Context via React Redux than directly because if it changes, the burden of updating the code will be on React Redux instead developer responsibility.

**Q.8. Why React uses className over class attribute?**

**Ans.** *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.

render() {

return <span className="menu navigation-menu">Menu</span>

}

**Q.9. What is Relay?**

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

**Q.10. How Relay is different from Redux?**

**Ans.** 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.

**Q.11. What is Combine Reducer?**

**Ans.** The combineReducers helper function turns an object whose values are different reducing functions into a single reducing function you can pass to createStore . The resulting reducer calls every child reducer, and gathers their results into a single state object.

### Q1: What is React?

Topic: **React**  
Difficulty: ⭐

React is an open-source JavaScript library created by Facebook for building complex, interactive UIs in web and mobile applications. React’s core purpose is to build UI components; it is often referred to as just the “V” (View) in an “MVC” architecture.

🔗 **Source:** [codementor.io](https://www.codementor.io/blog/5-essential-reactjs-interview-questions-du1084ym1)

### Q2: What is Redux?

Topic: **Redux**  
Difficulty: ⭐

**Redux** is a predictable state container for JavaScript apps based on the Flux design pattern. Redux can be used together with ReactJS, or with any other view library. It is tiny (about 2kB) and has no dependencies.

🔗 **Source:** [github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q3: What is virtual DOM?

Topic: **React**  
Difficulty: ⭐

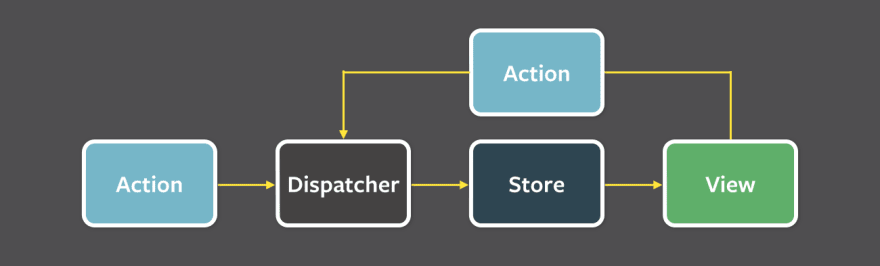
**The virtual DOM (VDOM)** is an in-memory representation of Real DOM. The representation of a UI is kept in memory and synced with the “real” DOM. It’s a step that happens between the render function being called and the displaying of elements on the screen. This entire process is called reconciliation.

🔗 **Source:** [github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q4: What is Flux?

Topic: **Redux**  
Difficulty: ⭐

**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 used this pattern internally when working with React The workflow between dispatcher, stores and views components with distinct inputs and outputs as follows:

[](https://res.cloudinary.com/practicaldev/image/fetch/s--7e-0f9Fb--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/github.com/sudheerj/reactjs-interview-questions/raw/master/images/flux.png)

🔗 **Source:** [github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q5: What are the advantages of ReactJS?

Topic: **React**  
Difficulty: ⭐

Below are the advantages of ReactJS:

1. Increases the application’s performance with Virtual DOM
2. JSX makes code is easy to read and write
3. It renders both on client and server side
4. Easy to integrate with other frameworks (Angular, BackboneJS) since it is only a view library
5. Easy to write UI Test cases and integration with tools such as JEST.

🔗 **Source:** [github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q6: What are the major features of ReactJS?

Topic: **React**  
Difficulty: ⭐

The major features of ReactJS are as follows,

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

🔗 **Source:** [https://github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q7: What is the difference between a Presentational component and a Container component?

Topic: **React**  
Difficulty: ⭐⭐

* **Presentational components** are concerned with how things look. They generally receive data and callbacks exclusively via props. These components rarely have their own state, but when they do it generally concerns UI state, as opposed to data state.
* **Container components** are more concerned with how things work. These components provide the data and behavior to presentational or other container components. They call Flux actions and provide these as callbacks to the presentational components. They are also often stateful as they serve as data sources.

🔗 **Source:** [github.com/Pau1fitz](https://github.com/Pau1fitz/react-interview)

### Q8: Describe how events are handled in React.

Topic: **React**  
Difficulty: ⭐⭐

In order to solve cross browser compatibility issues, your event handlers in React will be passed instances of SyntheticEvent, which is React’s cross-browser wrapper around the browser’s native event. These synthetic events have the same interface as native events you’re used to, except they work identically across all browsers.

What’s mildly interesting is that React doesn’t actually attach events to the child nodes themselves. React will listen to all events at the top level using a single event listener. This is good for performance and it also means that React doesn’t need to worry about keeping track of event listeners when updating the DOM.

🔗 **Source:** [tylermcginnis.com](https://tylermcginnis.com/react-interview-questions/)

### Q9: What is state in ReactJS?

Topic: **React**  
Difficulty: ⭐⭐

**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 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>

);

}

}

🔗 **Source:** [https://github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q10: What is the difference between state and props?

Topic: **React**  
Difficulty: ⭐⭐

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. i.e,

* Props get passed to the component similar to function parameters
* state is managed within the component similar to variables declared within a function.

🔗 **Source:** [https://github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q11: What are the limitations of ReactJS?

Topic: **React**  
Difficulty: ⭐⭐

Below are the list of limitations:

1. React is just a view library, not a full-blown framework
2. There is a learning curve for beginners who are new to web development.
3. Integrating React.js into a traditional MVC framework requires some additional configuration
4. The code complexity increases with inline templating and JSX.
5. Too many smaller components leading to over-engineering or boilerplate

🔗 **Source:** [github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q12: What’s the difference between an "Element" and a "Component" in React?

Topic: **React**  
Difficulty: ⭐⭐

Simply put, a React element describes what you want to see on the screen. Not so simply put, a React element is an object representation of some UI.

A React component is a function or a class which optionally accepts input and returns a React element (typically via JSX which gets transpiled to a createElement invocation).

🔗 **Source:** [medium.freecodecamp.org/](https://medium.freecodecamp.org/react-interview-questions-c8a319ed02bd)

### Q13: How is React different from AngularJS (1.x)?

Topic: **React**  
Difficulty: ⭐⭐

For example, AngularJS (1.x) approaches building an application by extending HTML markup and injecting various constructs (e.g. Directives, Controllers, Services) at runtime. As a result, AngularJS is very opinionated about the greater architecture of your application — these abstractions are certainly useful in some cases, but they come at the cost of flexibility.

By contrast, React focuses exclusively on the creation of components, and has few (if any) opinions about an application’s architecture. This allows a developer an incredible amount of flexibility in choosing the architecture they deem “best” — though it also places the responsibility of choosing (or building) those parts on the developer.

🔗 **Source:** [codementor.io](https://www.codementor.io/blog/5-essential-reactjs-interview-questions-du1084ym1)

### Q14: What are the downsides of Redux over Flux?

Topic: **Redux**  
Difficulty: ⭐⭐⭐

Instead of saying downsides we can say that there are few compromises of using Redux over Flux. Those are as follows:

1. **You will need to learn avoiding 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 your team to write non-mutative code.
2. **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 young but rich ecosystem. This means most packages are new ideas and haven't received the critical mass of usage yet
3. **There is no nice Flow integration yet:** Flux currently lets you do very impressive static type checks which Redux doesn't support yet.

🔗 **Source:** [github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q15: What are error boundaries in ReactJS (16)?

Topic: **React**  
Difficulty: ⭐⭐⭐

Error boundaries are React 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)

class ErrorBoundary extends React.Component {

constructor(props) {

super(props);

this.state = { hasError: false };

}

componentDidCatch(error, info) {

// Display fallback UI

this.setState({ hasError: true });

// You can also log the error to an error reporting service

logErrorToMyService(error, info);

}

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>

🔗 **Source:** [github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q16: Why ReactJS uses className over class attribute?

Topic: **React**  
Difficulty: ⭐⭐⭐

**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.

render() {

return <span className="menu navigation-menu">Menu</span>

}

🔗 **Source:** [github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q17: How to access redux store outside a react component?

Topic: **Redux**  
Difficulty: ⭐⭐⭐

Yes.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;

🔗 **Source:** [github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q18: What can you tell me about JSX?

Topic: **React**  
Difficulty: ⭐⭐⭐

When Facebook first released React to the world, they also introduced a new dialect of JavaScript called JSX that embeds raw HTML templates inside JavaScript code. JSX code by itself cannot be read by the browser; it must be transpiled into traditional JavaScript using tools like Babel and webpack. While many developers understandably have initial knee-jerk reactions against it, JSX (in tandem with ES2015) has become the defacto method of defining React components.

class MyComponent extends React.Component {

render() {

let props = this.props;

return (

<div className="my-component">

<a href={props.url}>{props.name}</a>

</div>

);

}

}

🔗 **Source:** [codementor.io](https://www.codementor.io/blog/5-essential-reactjs-interview-questions-du1084ym1)

### Q19: Why should not we update the state directly?

Topic: **React**  
Difficulty: ⭐⭐⭐

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:** The only place you can assign the state is constructor.

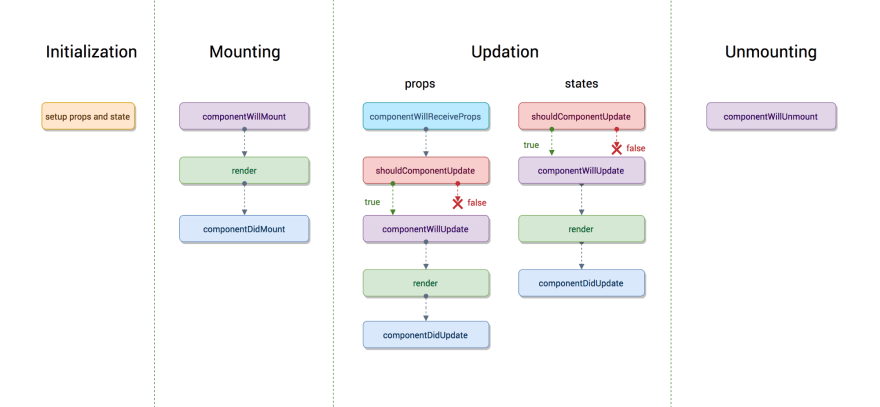
🔗 **Source:** [https://github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q20: What are the different phases of ReactJS component lifecycle?

Topic: **React**  
Difficulty: ⭐⭐⭐

There are four different phases of React component’s lifecycle:

1. **Initialization:** In this phase react component prepares setting up the initial state and default props.
2. **Mounting:** The react component is ready to mount in the browser DOM. This phase covers **componentWillMount** and **componentDidMount** lifecycle methods.
3. **Updating:** In this phase, the component get updated in two ways, sending the new props and updating the state. This phase covers **shouldComponentUpdate, componentWillUpdate and componentDidUpdate** lifecycle methods.
4. **Unmounting:** In this last phase, the component is not needed and get unmounted from the browser DOM. This phase include **componentWillUnmount** lifecycle method.

[](https://res.cloudinary.com/practicaldev/image/fetch/s--JddQdxAF--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https:/github.com/sudheerj/reactjs-interview-questions/raw/master/images/phases.png)

🔗 **Source:** [github.com/sudheerj](https://github.com/sudheerj/reactjs-interview-questions)

### Q21: Describe Flux vs MVC?

Topic: **React**  
Difficulty: ⭐⭐⭐⭐

Traditional MVC patterns have worked well for separating the concerns of data (Model), UI (View) and logic (Controller) — but MVC architectures frequently encounter two main problems:

* **Poorly defined data flow:** The cascading updates which occur across views often lead to a tangled web of events which is difficult to debug.
* **Lack of data integrity:** Model data can be mutated from anywhere, yielding unpredictable results across the UI.

With the Flux pattern complex UIs no longer suffer from cascading updates; any given React component will be able to reconstruct its state based on the data provided by the store. The Flux pattern also enforces data integrity by restricting direct access to the shared data.

🔗 **Source:** [codementor.io](https://www.codementor.io/blog/5-essential-reactjs-interview-questions-du1084ym1)

### Q22: Why would you use forceUpdate in a React component?

Topic: **React**  
Difficulty: ⭐⭐⭐⭐

In order to force a re-render if there is some condition React is not detecting that requires an update to the UI. Typically this should not be necessary to call.

🔗 **Source:** [github.com/WebPredict](https://github.com/WebPredict/react-interview-questions)

### Q23: What is wrong with this code?

Topic: **React**  
Difficulty: ⭐⭐⭐⭐

**Questions:**

What is wrong with this code?

this.setState((prevState, props) => {

return {

streak: prevState.streak + props.count

}

})

**Answer:**

Nothing is wrong with it. It’s rarely used and not well known, but you can also pass a function to setState that receives the previous state and props and returns a new state, just as we’re doing above. And not only is nothing wrong with it, but it’s also actively recommended if you’re setting state based on previous state.

🔗 **Source:** [tylermcginnis.com](https://tylermcginnis.com/react-interview-questions/)

### Q24: What is the difference between a controlled component and an uncontrolled component?

Topic: **React**  
Difficulty: ⭐⭐⭐⭐

* A **controlled component** is a component where React is in control and is the single source of truth for the form data.
* An **uncontrolled component** is where your form data is handled by the DOM, instead of inside your React component.

Though uncontrolled components are typically easier to implement since you just grab the value from the DOM using refs, it’s typically recommended that you favor controlled components over uncontrolled components. The main reasons for this are that controlled components support instant field validation, allow you to conditionally disable/enable buttons, enforce input formats, and are more “the React way”.

🔗 **Source:** [github.com/Pau1fitz](https://github.com/Pau1fitz/react-interview)

### Q25: Explain some difference between Flux and AngularJS (1.x) approach

Topic: **React**  
Difficulty: ⭐⭐⭐⭐⭐

UI components in AngularJS typically rely on some internal $scope to store their data. This data can be directly mutated from within the UI component or anything given access to $scope — a risky situation for any part of the component or greater application which relies on that data.

By contrast, the Flux pattern encourages the use of immutable data. Because the store is the central authority on all data, any mutations to that data must occur within the store. The risk of data pollution is greatly reduced.

🔗 **Source:** [codementor.io](https://www.codementor.io/blog/5-essential-reactjs-interview-questions-du1084ym1)

### Q26: What does Side effects mean in React? Provide some examples.

Topic: **React**  
Difficulty: ⭐⭐⭐⭐⭐

A **"side effect"** is anything that affects something outside the scope of the function being executed. These can be, say, a network request, which has your code communicating with a third party (and thus making the request, causing logs to be recorded, caches to be saved or updated, all sorts of effects that are outside the function.

There are more subtle side effects, too. Changing the value of a closure-scoped variable is a side effect. Pushing a new item onto an array that was passed in as an argument is a side effect. Functions that execute without side effects are called "pure" functions: they take in arguments, and they return values. Nothing else happens upon executing the function. This makes the easy to test, simple to reason about, and functions that meet this description have all sorts of useful properties when it comes to optimization or refactoring.

Pure functions are deterministic (meaning that, given an input, they always return the same output), but that doesn't mean that all impure functions have side effects. Generating a random value within a function makes it impure, but isn't a side effect, for example. React is all about pure functions, and asks that you keep several lifecycle methods pure.