[1. What are refs in ReactJS?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled1)

**Refs** are used for managing focus, selecting text and triggering animations. It also integrates with third-party libraries. Refs help in getting the reference to a DOM (Document Object Model) node. They return the node that we are referencing. Whenever we need DOM measurements, we can use refs.

[2. What is the purpose of render() function in ReactJS?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled2)

**render()** function is used to update the UI. For this, you have to create a new element and send it to ReactDOM.render(). React elements are immutable and once you create an element, you cannot change its attributes. Thus, elements are like a single frame and it depicts the UI at some point. ReactDOM.render() controls the content of the container node you pass and if there is any DOM element already present in it then it would be replaced when first called.

[3. What is the use of arrow function in ReactJS?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled3)

**Arrow functions** are extremely important for React operations. It prevents this bugs. Arrow functions make it easier to predict the behavior of this bugs when passed as callbacks. They don’t redefine the value of this within their function body. Hence, prevents bugs caused by the use of this within callbacks.

[4. How is ReactJS different from angular and VUE js?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled4)

**Angular Js** – developed by Google, angular is a typescript based JavaScript application framework. It is also known as Super-heroic JavaScript MVW Framework. It was developed with the motive to encounter the challenges of creating single page applications. There are several versions of angular such as Angular 2+, Angular 2 or ng2. Angular is the rewritten, mostly incompatible successor to AngularJS which means AngularJS is the oldest framework.

**React**– React was developed by Facebook in March 2013. It is a JavaScript library that is used for building user interfaces. React creates large web applications and also provides speed, scalability, and simplicity.

**Vue Js-** Launched in February 2014, Vue is the most famous and rapidly growing framework in JS field. Vue is an intuitive, fast and composable MVVM for building interactive interfaces. It is extremely adaptable and several JavaScript libraries make use of this. Vue is also a web application framework that helps in making advanced single page applications.

[5. How is flux different from redux?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled5)

Difference Between FLUX and Redux

|  |  |
| --- | --- |
| FLUX | REDUX |
| Flux is a container for application state and logic that are registered to callbacks. | Redux is a container for JavaScript apps. |
| It is an observer pattern that is modified to fit React. | It offers interesting features such as writing applications, testing in different environments such as a server, client, etc. |
| It is a fancy name given to observer pattern and Facebook has developed tools to aid the implementation of these patterns. | In redux, actions can be functions or promises. |
| Flux supports actions that are simple JavaScript objects. | Redux is the first choice for web developers because it offers live code editing. |

[6. What are the controlled components and uncontrolled components in ReactJS ?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled6)

**Controlled component** is more advisable to use as it is easier to implement forms in it. In this, form data are handled by React components. A controlled input accepts values as props and callbacks to change that value.

The **uncontrolled component** is a substitute for controlled components. Here form data is handled by DOM itself. In uncomfortable components, the ref can be used to get the form values from DOM.

[7. Explain the difference between functional and class components.](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled7)

**Functional components** are those components that returns React elements as a result. These are just simple old JavaScript functions. React 0.14 has given a new shortcut for creating simple stateless components that are known as functional components. These components make use of easy JavaScript functions.

**Class components** – most of the tech-savvy people are more familiar with class components as they have been around the corner for a longer time. These components make use of plain old javascript objects for creating pages, mixins, etc in an identical way. Using React’s create a class factory method, a literal is passed in defining the methods of a new component.

[8. What do you understand by mixin or higher order components in ReactJs ?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled8)

**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.

[9. What are Synthetic events?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled9)

SyntheticEvent is a cross-browser wrapper around browser’s native event. In React all of your event handlers will be passed instances of SyntheticEvent.The synthetic event works the same way as the event system of browsers, the only difference is that the same code will work across all browsers.

Below is a simple example of how to listen for a click event in react

import React, { Component } from 'react';

class ShowAlert extends Component {

showAlert() {

alert("Im an alert");

}

render() {

return (

<button onClick={this.showAlert}>show alert </button>

);

}

}

export default ShowAlert;

[10. What is the difference between Dom and virtual Dom in ReactJS ?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled10)

**DOM** is the acronym for Document Object Model. Dom is also called HTML DOM as it is an abstraction of structured code called HTML for web developers. Dom and HTML code are interrelated as the elements of HTML are known as nodes of DOM. It defines a  structure where users can create, alter, modify documents and the content present in it. So while HTML is a text, DOM is an in-memory representation of this text.

**Virtual DOM** is an abstraction of abstraction as it has been derived from HTML DOM. It is a representation of DOM objects like a lightweight copy. The virtual DOM was not invented by React, it is only used and provided for free.

[11. Enlist the advantages and disadvantages of ReactJS ?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled11)

**React.j**s is used by web developers for creating large web pages without reloading the entire page. It uses the data and can be changed over time.

**The following are the advantages of using React.js-**

1- React makes Search engine optimization (SEO)  easy.

2- It is very efficient as it ensures readability and easy maintenance.

3- It gives extraordinary developer tools to web developers and makes Java coding easier for them.

4- UI test cases.

**The following are the disadvantages of React-**

1- Some major configurations are required for integrating React js with traditional MVC framework such as substituting erb with React js.

2- It is a steep learning process for people who are new to web development world.

[12. What is ReactJS ?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled12)

React js is javascript based UI Library developed at Facebook, to create an interactive, stateful & reusable UI components. It is one of the most popular [javascript frameworks](https://www.onlineinterviewquestions.com/javascript/) that is created for handling the presentation layer for the web and mobile apps.

[13. List some advantages of ReactJS ?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled13)

Advantages of React Js

* **React.js is extremely efficient:** React.js creates its own virtual DOM where your components actually live. This approach gives you enormous flexibility and amazing gain in performance. React.js also calculates what are the changes needed to be made in DOM. This process of React.js avoids expensive DOM operations and make updates in a very client manner.
* **It makes writing Javascript easier:** React.js uses a special syntax called JSX, which allows you to mix HTML with Javascript. The user can drop a bit of HTML in the render function without having to concatenate strings, this is another fantastic thing. React.js turns those bits of HTML into functions with a special JSXTransformer.
* **It gives you out-of-the-box developer tools:** When you start your journey with React.js, do not forget to install official React.js chrome extension. It makes debugging your application much easier. After you install the extension, you will have a direct look into the virtual DOM as if you were browsing a regular DOM tree in the elements panel. Isn’t it pretty amazing!
* **It’s awesome for SEO:** One of the biggest problems with other Javascript frameworks is that they do not search engine friendly. Though there have been some improvements in this area, search engines generally have trouble reading Javascript heavy applications. React.js stands out from the crowd because you can run React.js on the server, and the virtual DOM will be rendered to the browser as a regular web page.
* **UI Test Cases:** It is extremely easy to write UI test cases because the virtual DOM system implemented entirely in JS.

Source: http://www.pro-tekconsulting.com/blog/advantages-disadvantages-of-react-js/

[14. What are Components in ReactJS ?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled14)

React Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

Below is ample component written in ES6 class to display a welcome message on the screen.

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name} </h1>;

}

}

const element = <Welcome name="Sara" />;

ReactDOM.render(

element,

document.getElementById('root')

);

source: https://facebook.github.io/react/docs/components-and-props.html

[15. What is JSX?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled15)

JSX is an XML/HTML-like syntax used by React that extends ECMAScript so that XML/HTML-like text can co-exist with JavaScript/React code. The syntax is intended to be used by preprocessors (i.e., transpilers like Babel) to transform HTML-like text found in JavaScript files into standard JavaScript objects that a JavaScript engine will parse.

Basically, by using JSX you can write concise HTML/XML-like structures (e.g., DOM like tree structures) in the same file as you write JavaScript code, then Babel will transform these expressions into actual JavaScript code. Unlike the past, instead of putting JavaScript into HTML, JSX allows us to put HTML into JavaScript.

By using JSX one can write the following JSX/JavaScript code:

var nav = (

<ul id="nav">

<li><a href="#">Home</li>

<li><a href="#">About

<li><a href=”#”>Clients

<li><a href=”#”>Contact Us

</ul> );

And Babel will transform it into this:

var nav = React.createElement(

"ul",

{ id: "nav" },

React.createElement(

"li",

null,

React.createElement(

"a",

{ href: "#" },

"Home"

)

),

React.createElement(

"li",

null,

React.createElement(

"a",

{ href: "#" },

"About"

)

),

React.createElement(

"li",

null,

React.createElement(

"a",

{ href: "#" },

"Clients"

)

),

React.createElement(

"li",

null,

React.createElement(

"a",

{ href: "#" },

"Contact Us"

)

)

);

source:https://www.reactenlightenment.com/react-jsx/5.1.html

[16. Explain life Cycle of React JS Component ?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled16)

React JS Component Lifecycle

Each component has several “lifecycle methods” that you can override to run code at particular times in the process. Methods prefixed with will are called right before something happens, and methods prefixed with did are called right after something happens.

Mounting

These methods are called when an instance of a component is being created and inserted into the DOM:

* constructor()
* componentWillMount()
* render()
* componentDidMount()

Updating

An update can be caused by changes to props or state. These methods are called when a component is being re-

rendered:

* componentWillReceiveProps()
* shouldComponentUpdate()
* componentWillUpdate()
* render()
* componentDidUpdate()

Unmounting

This method is called when a component is being removed from the DOM:

* componentWillUnmount()

Other APIs

Each component also provides some other APIs:

* setState()
* forceUpdate()

Class Properties

* defaultProps
* displayName

Instance Properties

* props
* state

Source: https://facebook.github.io/react/docs/react-component.html

[17. List some features of ReactJS ?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled17)

Undoubtedly today React is among of one the best JavaScript UI frameworks. It comes with a lot of features that helps programmers to create beautiful application easily, we have listed some of them below.

* It’s Adaptability
* Free and Open Source
* Decorators from ES7
* Server-side Communication
* Asynchronous Functions & Generators
* Flux Library
* Destructuring Assignments
* Usefulness of JSX

Also, Read [**React Native Interview questions**](https://www.onlineinterviewquestions.com/react-native-interview-questions/)

[18. How to use Events in ReactJS ?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled18)

Using Events is React js is very similar to handling event on DOM elements.The difference is only in syntax like.

1. The name of event is React js is always in camelCase.
2. With JSX you pass a function as the event handler, rather than a string.

Lets understand by an example:

// In HTML

<button onclick="activateAccount()">

Activate Account

</button>

//In React

<button onClick={activateAccount}>

Activate Account

</button>

Another difference is that in React js you cannot return false to prevent default behavior in React. You must call preventDefault explicitly. Read more from https://facebook.github.io/react/docs/handling-events.html

[19. What is flux in JavaScript?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled19)

Flux is an application architecture for creating data layers in JavaScript applications.It was designed at Facebook along with the React view library.Flux is not a framework or a library. It is simply a new kind of architecture that complements React and the concept of Unidirectional Data Flow.  
further reading https://scotch.io/tutorials/getting-to-know-flux-the-react-js-architecture

[20. What are refs in ReactJs ? When to use it.](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled20)

In React ref is used to store the reference of element or component returned by the component render() configuration function.Refs should be avoided in most cases, however, they can be useful when we need DOM measurements or to add methods to the components.

Refs can be used in the following cases

* Managing focus, text selection, or media playback.
* Triggering imperative animations.
* Integrating with third-party DOM libraries.

[21. What are stateless components in ReactJS?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled21)

Stateless components are components that don’t have any state. When something is stateless, it calculates its internal state but it never directly mutates it. For creating a stateless components No class and this keyword is needed. You can create a stateless components using plain functions or Es6 arrow function. Below is an example of stateless component in react.

//In Es6

const Pane = (props) =>

{props.children}

;

//In Es5

const Username = ({ username }) =>

The logged in user is: {username}

[22. What is the difference between State and props in React Js ?](https://www.onlineinterviewquestions.com/react-js-interview-questions/" \l "collapseUnfiled22)

**Props** are shorthand for properties.they are very similar to an argument is passed to a pure javascript function. Props of the component are passed from parent component which invokes component. During a component’s life cycle props should not change consider them as immutable.In React all props can be accessible with this.props.

import React from 'react';

class Welcome extends React.Component {

render() {

return <h1>Hello {this.props.name}</h1>;

}

}

const element = ;

**State** are used to create dynamic and interactive components in React.State is heart of react component that makes it alive and determines how a component renders & behaves.

// simple state example

import React from 'react';

class Button extends React.Component {

constructor() {

super();

this.state = {

count: 0,

};

}

updateCount() {

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

return { count: prevState.count + 1 }

});

}

render() {

return (<button

onClick={() => this.updateCount()}

>

Clicked {this.state.count} times

</button>);

}

}

export default Button;

**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.

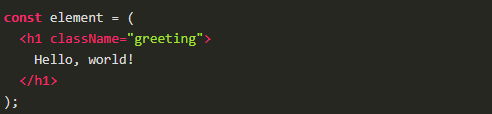
Whenever a ReactComponent is changing the state, diff algorithm in React runs and identifies what has changed. And then it updates the DOM with the results of diff. The point is - it’s done faster than it would be in the regular DOM.

**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.

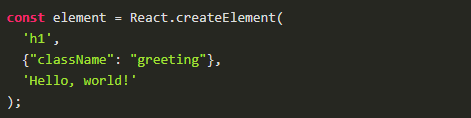
For example, below is the syntax for a basic element in React with JSX and its equivalent without it.

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



Equivalent of the above using React.createElement

https://miro.medium.com/max/60/1*CdjJk4B7TL5LbZFDLBX8sQ.png?q=20

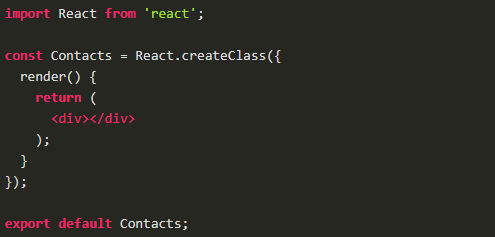


**Q3. What is React.createClass?**

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.

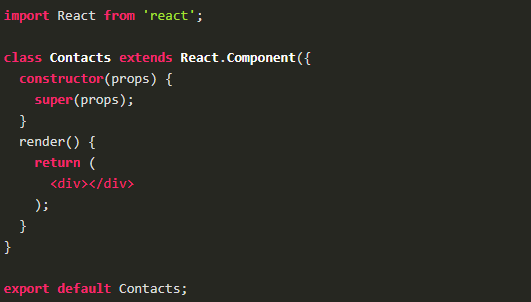
When using React’s createClass() method, we pass in an object as an argument. So we can write a component using createClass that looks like this:

https://miro.medium.com/max/60/1*q58HA3tryv9hpw2dxtpxmg.png?q=20



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.

https://miro.medium.com/max/60/1*zteV_2CbkJcRO1oHxL-lHQ.png?q=20



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 andReact?**

Prior to v0.14, all ReactDOM functionality was part of React. But later, Reactand 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.

https://miro.medium.com/max/60/1*2lHksigcP_3MFjKqCi7NBg.png?q=20



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.

https://miro.medium.com/max/60/1*a3zDRW805YMtAGVPuFZLMQ.png?q=20



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/521/1*2jC_2JPgxKTEN5bvp3SrUw.png

It includes everything we need to build a React app:

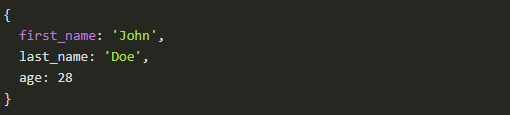
* React, JSX, ES6, and Flow syntax support.
* Language extras beyond ES6 like the object spread operator.
* Autoprefixed CSS, so you don’t need -webkit- or other prefixes.
* A fast interactive unit test runner with built-in support for coverage reporting.
* A live development server that warns about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

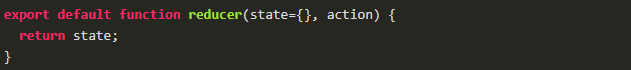


All of these data is retained by redux in a closure that redux calls a store . It also provides us a recipe of creating the said store, namely createStore(x).

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



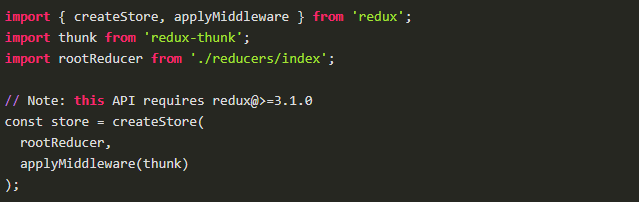
This store can only be updated by dispatching an action. Our App dispatches an action, it is passed into reducer; the reducer returns a fresh instance of the state; the store notifies our App and it can begin it's re render as required.

**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



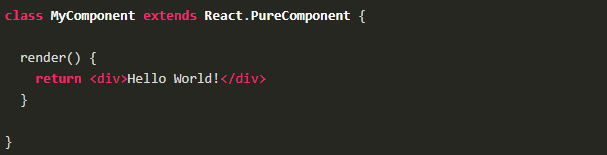
**Q12. What is PureComponent? When to use PureComponent over Component?**

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 viadispatch(action); Registers listeners via subscribe(listener); Handles unregistering of listeners via the function returned bysubscribe(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 withthis.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 tosetState and what is its purpose?**

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

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

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

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

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

With React Native it is possible to mimic the behavior of the native app in JavaScript and at the end, we will get platform specific code as the output. We may even mix the native code with the JavaScript if we need to optimize our application further.

# Q1. Why do we have both controlled and uncontrolled inputs?

## What interviewer wants to see: a solid understanding of a basic React concept.

Ok, this is more like a warm-up question. Which is even more important to give a full answer.

**A controlled input** accepts its current value as a prop, as well as a callback to change that value. It’s a “React way”:

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

**An uncontrolled input** stores its own state internally, using DOM API.

Here don’t provide value and onChange handler, but we use [ref](https://reactjs.org/docs/refs-and-the-dom.html)():

<input type="text" ref={this.textInput} />

And now you can access your input data as:

this.textInput.current.value

Your interviewer wants to hear more about details: are there any pros to using uncontrolled components, is there any performance difference?

Personally, I’ve never used uncontrolled inputs, but if you are learning React or you have to integrate React and non-React code then it might be necessary.

It’s nice to mention what your data (state) and UI (inputs) are always in sync with controlled input approach and it means you have to update the component’s state which triggers [React reconciliation process](https://reactjs.org/docs/reconciliation.html).

While with uncontrolled elements there is no need for that — you just keep the value inside the input DOM element.

# Q2. Why do we need a key property? Give an example when a bad key causes an error.

## What interviewer wants to see: some insight on how React works internally.

The answer is based on [the official React reconciliation process doc](https://reactjs.org/docs/reconciliation.html).

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](https://codesandbox.io/s/react-example-qfp5s). 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).

# Q3. How React event system is different from DOM events. Using stopImmediatePropagation().

## What interviewer wants to see: general knowledge of different JavaScript event models and everyday practical experience.

The best way to answer this question is to first shortly explain how DOM event concepts like bubbling and capturing work.

In short, with bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements.

With capturing, the event is first captured by the outermost element and propagated to the inner elements.

[Check this article for more info and demos](https://javascript.info/bubbling-and-capturing).

After that would be great to mention that React uses the **Event Delegation Pattern**— instead of assigning a handler to each of them — we put a single handler on their common ancestor and can access that element with event.target.

And then it’s quite obvious to explain how stopImmediatePropagation is different from the tradition stopPropagation method: it prevents other listeners of the same event from being called which a case for React because of the event delegation pattern.

# Q4. How to prevent components from re-rendering?

## What interviewer wants to see: more knowledge about React and you care about performance.

This is one of the most frequently asked questions. It’s worth to mention:

1. **shouldComponentUpdate()**— returns ‘true’ by default. You can override if you know which props have to trigger an update.
2. **PureComponents**— The difference between them is that [React.Component](https://reactjs.org/docs/react-api.html" \l "reactcomponent" \t "_blank) doesn’t implement [shouldComponentUpdat](https://reactjs.org/docs/react-component.html" \l "shouldcomponentupdate" \t "_blank)e method but React.PureComponentimplements it with a shallow prop and state comparison.
3. **React.memo** — The same as the previous one but it works with functional components.

And as a follow up: when to use Component instead of PureComponent?

You use PureComponents in 99% of cases in modern React. However, if you are working with Redux selectors, often you will need to explicitly specify the incoming prop changes to cancel the impending re-render to prevent UI thrashing. In this case, it’s appropriate to use a Component.

Check out this [deep reading](http://jamesknelson.com/should-i-use-shouldcomponentupdate/) if it’s all vague for you.

# Q5. Give an example of HOC.

## What interviewer wants to see: you are familiar with a well-known React pattern.

HOC — a higher-order component is a function that takes a component and returns a new component. It’s a technique for sharing code between React components.

Let’s say we have a simple Button component:

const Button = props => <button {...props}>Hello</button>;

We want to create a HOC for making a red border for our Button component. Let’s create it using our definition:

const withRedBorder = Component =>  
 (props) => (  
 <Component {...props} style={{ border: "1px solid red" }} />  
 );

And now we can create a RedButton component:

const RedButton = withRedBorder(Button);// and use it as:  
<RedButton />

Libraries that use HOC pattern: [Redux connect](https://react-redux.js.org/api/connect), [recompose](https://github.com/acdlite/recompose).

# Q6. What is the render props?

## What interviewer wants to see: you are familiar with a second well-known React pattern.

Render props — when a component takes a function that returns a React element and calls it instead of implementing its own render logic.

It’s another technique for sharing code between React components.

An example:

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

Libraries that use render props pattern: [React Router](https://reacttraining.com/react-router/web/api/Route/render-func), [Downshift](https://github.com/paypal/downshift).

**A follow up**: what about HOC vs render props. Which one is better, when to use? This is a quite controversial topic, in this [article](https://www.richardkotze.com/coding/hoc-vs-render-props-react) you can find some pros and cons.

# **Q7. React unit tests vs integration tests for components.**

## What interviewer wants to see: understanding about different approaches to testing components.

It’s worth to mention both [Enzyme](https://airbnb.io/enzyme/) and [react-testing-library](https://github.com/testing-library/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.

Check out this [neat video](https://www.youtube.com/watch?v=YQLn7ycfzEo) about react-testing-library.

# **Q8. What’s your favorite hook, how to implement it?**

## What interviewer wants to see: practical usage of hooks and understanding of how it works.

Despite hooks are still new API, a lot of people already use it in production and they expect you to know it.

Let’s recreate a useWindowSize — it’s easy-to-read hook and quite straightforward:

import { useState, useEffect } from 'react';

const useWindowSize = () => {  
 const getSize = () => ({  
 width: window.innerWidth,  
 height: window.innerHeight  
 });

const [size, setSize] = useState(getSize);

useEffect(() => {  
 const handleResize = () => setSize(getSize());  
 window.addEventListener('resize', handleResize);  
 return () => window.removeEventListener('resize', handleResize);  
 }, []); return size;  
}

Usage:

const App = () => {  
 const size = useWindowSize(); return (  
 <div>  
 {size.width}px / {size.height}px  
 </div>  
 );  
}

You might expect some questions like:

1. Is that necessary to call your function useWindowSize , what about just getWindowSize?

Yes, without it, we wouldn’t be able to automatically check for violations of [rules of Hooks](https://reactjs.org/docs/hooks-rules.html) because we couldn’t tell if a certain function contains calls to Hooks inside of it.

2. Will it work if we remove [] argument from useEffect?

Yes, but we’ll call useEffect hook on every render which may cause performance issues.

3. How React knows when to re-render App component if we handle window resizing in useWindowSize?

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

There is some portion of magic with React hooks. Check out [Why Do React Hooks Rely on Call Order?](https://overreacted.io/why-do-hooks-rely-on-call-order/) article for more info.

4. How to make this hook ready for Server-Side Rendering?

Something like this:

import { useState, useEffect } from 'react';

const useWindowSize = () => {  
 const isClient = typeof window === 'object';   
 const getSize = () => ({  
 width: isClient ? window.innerWidth : undefined,  
 height: isClient ? window.innerHeight : undefined  
 }); const [size, setSize] = useState(getSize); useEffect(() => {  
 const handleResize = () => setSize(getSize());  
 window.addEventListener('resize', handleResize);  
 return () => window.removeEventListener('resize', handleResize);  
 }, []); return size;  
}

# Q9. Context API. How much it’s re-render?

## What interviewer wants to see: arguably the most important thing in React - how to manage the data flow.

First of all, it’s worth to mention that unlike Redux, the context **doesn’t have its own state,**instead, it’s just a conduit and usually reads data from another component’s state.

Also, with Context API it’s quite easy to re-render much more than you need **even if you use PureComponent or React.memo**.

In the example below every time, we click on the button you will see a new message in the console:

import React, { useState, useContext } from "react";  
import ReactDOM from "react-dom";const ProfileContext = React.createContext();const App = () => {  
 const [val, setIncrement] = useState(0);  
 return (  
 <div>  
 <Profile userInfo={val} />  
 <button onClick={() => setIncrement(val + 1)}>Click</button>  
 <p>value: {val}</p>  
 </div>  
 );  
};class Profile extends React.PureComponent {  
 state = {  
 name: "Alex"  
 }; setName(name) {  
 this.setState({ name });  
 } render() {  
 const { name } = this.state;  
 const { setName } = this; return (  
 <ProfileContext.Provider  
 value={{  
 name,  
 setName  
 }}  
 >  
 <Logger />  
 </ProfileContext.Provider>  
 );  
 }  
}const Logger = React.memo(() => {  
 const { name } = useContext(ProfileContext);  
 console.log("Logger rerendering");  
 return null;  
});ReactDOM.render(<App />, document.querySelector("#root"));

In the example above our **Logger component is a pure component, but we still re-render it every time**. The reason for that is because we use the context here. Try to comment out the first line in the Logger component, now it won’t re-render.

This is because the context uses a reference identity to determine when to re-render. Let’s change our Profile component to make it work:

class Profile extends React.PureComponent {  
 state = {  
 name: "Alex",  
 setName(name) {  
 this.setState({ name });  
 }  
 }; render() {  
 return (  
 <ProfileContext.Provider value={this.state}>  
 <Logger />  
 </ProfileContext.Provider>  
 );  
 }  
}

Now it works as expected. Btw, if you know how to make it work with hooks — post it in the comment section!

# Q10. Migration from Class to Function Components.

## What interviewer wants to see: you have an idea of how to refactor ReactJS using hooks and you can demonstrate it.

With function components and hooks you can cover all class components use-cases, such as:

1. Class component state with useState hook
2. Class component lifecycle methods with useEffect hook
3. Bonus points: better abstraction with custom hooks

Let’s take a look at the example below and refactor it using hooks:

class App extends React.Component {  
 state = {  
 value: localStorage.getItem("info") || ""  
 }; componentDidUpdate() {  
 localStorage.setItem("info", this.state.value);  
 } onChange = event => {  
 this.setState({ value: event.target.value });  
 }; render() {  
 const { value } = this.state;  
 return (  
 <div>  
 <input value={value} type="text" onChange={this.onChange} />  
 <p>{value}</p>  
 </div>  
 );  
 }  
}

This is already quite compact, thanks to class fields declaration and [class arrow functions](https://javascriptweblog.wordpress.com/2015/11/02/of-classes-and-arrow-functions-a-cautionary-tale/).

The first iteration of refactoring replacing setState and componentDidUpdatemight look like this:

const App = () => {  
 const val = localStorage.getItem("info") || "";  
 const [value, setValue] = useState(val);  
 const onChange = event => setValue(event.target.value); useEffect(() => localStorage.setItem("info", value), [value]); return (  
 <div>  
 <input value={value} type="text" onChange={onChange} />  
 <p>{value}</p>  
 </div>  
 );  
};

Only 13 lines of code instead of 19 with classes with exactly the same functionality. But we can go one step further: we can easily extract and reuse a custom hook here:

const usePersistentStorage = key => {  
 const val = localStorage.getItem(key) || "";  
 const [value, setValue] = useState(val); useEffect(() => localStorage.setItem(key, value), [key, value]); return [value, setValue];  
};

And now we can use this in our App:

const App = () => {  
 const [value, setValue] = usePersistentStorage("info");  
 const onChange = event => setValue(event.target.value); return (  
 <div>  
 <input value={value} type="text" onChange={onChange} />  
 <p>{value}</p>  
 </div>  
 );  
};