[Top 40 React Interview Questions and Answers for 2023](https://www.knowledgehut.com/interview-questions/react)

=================

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

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

**2. What are controlled and uncontrolled components?**

In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself. To write an uncontrolled component, instead of writing an event handler for every state update, you can use a ref to get form values from the DOM.

**3. What is Lifting State Up in React?**

In React, sharing state is accomplished by moving it up to the closest common ancestor of the components that need it. This is called “lifting state up”.

**4. What are the lifecycle methods of React?**

The few lifecycle methods of React

The **render**() method is the most used lifecycle method and handles the rendering of the component to the UI.

The **componentDidMount**() is called as soon as the component is mounted and ready. This is a good place to initiate API calls, if you need to load data from a remote endpoint.

The **componentDidUpdate**()is invoked as soon as the updating happens. The method is updating the DOM in response to prop or state changes.

The **componentWillUnmount**() is called just before the component is unmounted and destroyed.

**5. What are portals in React?**

Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component. The first argument ( child ) is any renderable React child, such as an element, string, or fragment.

**6. What is the use of react-dom package?**

The react-DOM package provides DOM-specific methods that can be used at the top level of the app and as an escape hatch to get outside of the React model if we need to. Most of the components should not need to use this module.

**7. How to use InnerHtml in React?**

This rule applies when innerHTML prop for a React DOM element is used. InnerHTML prop is risky because it is easy to expose the users to a cross-site scripting (XSS) attack. React provides dangerouslySetInnerHTML as a replacement for innerHTML prop to remind you that it is dangerous.

**8. How events are different in React?**

Handling events with React elements is very similar to handling events on DOM elements. There are some syntax differences: React events are named using **camelCase**, rather than lowercase. With JSX you pass a function as the event handler, rather than a string.

**9. How you use decorators in React?**

Decorators in React help to take an existing Class component, or function of a Class component, and modify it, thereby allowing adding 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.

**10. How to enable production mode in React?**

* Building your app for production
* Enable Debugging over USB-Most Android devices can only install and run apps downloaded from Google Play, by default.
* Plug in your device via USB-Let's now set up an Android device to run our React Native projects.
* Run your app

**15. What is strict mode in React?**

Strict Mode is a tool for highlighting potential problems in an application. Like Fragment. Strict Mode does not render any visible UI. It activates additional checks and warnings for its descendants.

**16. What are React Mixins?**

Mixins extend the class they are mixed into, while HOCs compose classes and return new classes. Mixins can set State in components since they are going to be part of that component. HOCs should not set State since they are only functions that take a class and return another class, they communicate through props instead.

================

1. You should not limit yourself with React js interview questions and answers. Do not forget to ask about the fundamentals of programming, JavaScript, front-end development, network data transfer, security, and code testing.

2. Your questions should be ordered from the simplest to the most difficult ones.

Now, I will show you the 36 React interview questions that I recommend asking. They are split into seven blocks:

1. Programming
2. JavaScript
3. Front-End development
4. React
5. Network data transfer
6. Security
7. Code testing

Under each block, I will tell why *it is essential for the developer to know this topic*. Under each question, I will explain what answer you should expect.

## React Basics — React Interview Questions

### **1. What Is React? What Types of React Components Are You Familiar With?**

React is “a free and open-source front-end JavaScript library for building user interfaces based on UI components. React can be used as a base in developing SPAs or mobile apps. Besides, React is maintained by Meta (previously known as Facebook).

React has four types of components: **functional components, pure components, class components, and higher-order components.** Functional components take in props and return JSX. Pure components are the simplest and fastest — they don’t depend on the state of variables from the outer scope. Class components consist of a class and can use all the major React functions. Higher-order components are advanced components, and they return one or more components depending on the array data.

### **2. What Is The Difference Between Real DOM and Virtual DOM?**

DOM stands for Document Object Model, and it is a mechanism for programs to access and update the structure, style, and contents of a document. Real DOM, also called browser DOM or HTML DOM, treats a document as a tree with its elements being nodes. Yet, real DOM is very inefficient and hard to manage. Virtual DOM is an abstract form of real DOM, a more lightweight one. All ReactJS reviews mention the fact that React uses virtual DOM as its ultimate advantage.

### **3. What types of hooks are there in React? How to handle the lifecycle inside the hooks properly?**

Hooks are the functional components, such as useEffect, useState, useRef, useCallback. To handle a lifecycle, you use the useEffect hook.

### **4. How can you create high-order components in React?**

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

**Reference:**

<https://www.freecodecamp.org/news/higher-order-components-in-react/#:~:text=Higher%2Dorder%20components%20(HOCs),that%20wraps%20the%20original%20component>.

### **5. What is the concept of keys in React, and why cannot we use indexes of the map as keys for dynamic lists?**

Keys are the properties given to every element in an array to give them a unique ID and identify which items have changed (added/removed/re-ordered). Keys have to be static and unique. Meanwhile, a map is an iterative function. If we used map indexes as keys for dynamic lists, then the entire list would be rerendered once a change in a single element occurred.

### **6. How events are handled in React?**

There are many different types of events: keyboard, mouse, touch, clipboard, form events, and a lot more. The process of handling events in React resembles event handling in plain HTML/JS. Yet, there are differences: React events names use **camel case**, not a lowercase, and in JSX, you pass a function as the event handler, not a string.

The main problem is that different browsers handle events differently — what can be rendered in Google Chrome won’t necessarily be rendered in Internet Explorer. React solves this problem. It introduces synthetic events, which emulate the work of a native event in every browser.

### **7. What is the difference between controlled and uncontrolled components?**

Controlled components are bound to values, and to introduce any changes, event-based callbacks should be used. The **input in controlled components is handled by React, not by DOM, the mutable state is kept in the state property**, and there are functions that control the data passed down to the component. Meanwhile, **uncontrolled** elements are very similar to the conventional HTML inputs, and **the form data is handled by DOM. This data has its own state, which is updated with each change in input value.**

### **8. How does context API work in React?**

Usually, variables have to be passed from parent nodes to child nodes, which may be a cumbersome solution. Context is a way to share this data without passing it explicitly through each level of a tree. Context passes the data from a parent to every child of it, which is marked as a context consumer. In other words, a parent is a context provider, and each child that agrees to accept changes in context is a context consumer. For example, context is often used to set UI themes, which are used in most, if not all, parts of the app.

## **Programming Basics — React Interview Questions**

### **9. What programming paradigms are you familiar with?**

There are many programming paradigms, such as imperative, declarative, structured, procedural, event-driven, flow-driven, and others. But the two major paradigms are Object-Oriented Programming (OOP) and Functional Programming (FP).

### **10. What are the fundamental principles of Functional Programming?**

There are seven essential principles of Functional Programming:

* Immutability — Once a value is assigned to something, this value shouldn’t change.
* Disciplined state — You should avoid shared, mutable states.
* Referential transparency — Once you replace a function call with its return value, the behavior of the app shouldn’t change.
* Pure functions — A function always has to return the same output if given the same inputs, and it should have no side-effects.
* First-class functions — Functions can be passed as arguments, returned as values, stored in data structures, and assigned to variables.
* Higher-order functions — Higher-order functions should take functions as arguments and returns a function as its output.
* Type systems — The compiler helps you avoid common development mistakes and errors.

### **11. What is idempotence and how it is connected with FP?**

Idempotence is a concept used in mathematical logic and computer science. Simply saying, it means that the same operation should always return the same output if provided the same input. Its relation with the Functional Programming rests in the fact that the concept of idempotence reflects one of the core principles of FP, that is, Pure functions.

### **12. What is currying, side effect, and pure function?**

**Currying** is an advanced technique of working with functions, which doesn’t call but transforms the function. It translates a function from callable as f(a, b, c) into callable as f(a)(b)(c).

**Side-effect** is any change in the application state, which modifies some state variable value outside its local environment.

**Pure function** is a function that (1) always returns the same output if given the same inputs and (2) has no side effects.

### 13. How do you understand the concepts of MVC, MVP, MVVM?

* MVC is a software development pattern, which stands for Model, View, Controller.
* MVP is a software development pattern, which stands for Model, View, Presenter.
* MVVM MVC is a software development pattern, which stands for Model, View, View Model.

Models contain all the application data. Views display visual elements and controls on the user interface. Controller takes inputs and converts it into commands for Model or View. Presenter acts as a middleman between Model and View — it takes data from Model and formats it to be displayed by View. A View Model transforms data from Model into values that View can display.

### 14. What is the difference between primitives and non-primitives?

There are five primitive data types in JavaScript: number, string, boolean, undefined, and null, but there is one non-primitive data type: object. Primitive data types are stored by value, while non-primitive data types are stored by reference. When creating a primitive variable, you create a potential new address, but when you create a non-primitive object, you create a pointer to that object.

### 15. What is closure, and what is it used for?

Closure is a mechanism that connects a function and a reference to its outer scope. Created at the moment of function creation, a closure provides access to the outer scope from the function’s inner scope. A closure controls what is and what is not in the scope of a certain function. It can be used in event handlers, callbacks, currying, and other functional programming patterns. Most importantly, closures are often used for object data privacy.

### 16. What is the difference between a function and a plain object?

A plain object is a set of key/value pairs wrapped in {}, which is perfect for storing simple data sets. A function is a Function object; it has parameters, which are called arguments and passed to the function by value, and a return statement, which specifies the value that the function returns.

### 17. How does hoisting work in JavaScript, and what is the order of hoisting?

Hoisting is a JavaScript mechanism, which moves variables and function declarations to the top of their scope before code is executed. As for the order of precedence, variable assignments go first, function declarations go second, and variable declarations go third.

### 18. What is the difference in usage of callback, promise, and async/await?

Callbacks are the functions that take time to produce a result. These functions, however, prove ineffective when there are callbacks nested inside other callbacks. Promises are callbacks that have three states: resolved, rejected, or pending. They are more effective than callbacks. Yet, async/await is the most convenient way of dealing with asynchronous code. Once called, an async function returns a Promise. If it returns a value, the Promise is resolved with this value. If it throws an exception or a value, the Promise is rejected with the thrown value. An await expression located inside of an async function pauses the execution of it and waits for the Promise to be resolved, then, resumes the execution, and returns the resolved value.

### 19. How does the inheritance model in JavaScript differ from that in other languages?

JavaScript doesn’t support multiple inheritances. To inherit property values at run time, JavaScript searches through the prototype chain of an object to find a value. Each object has a single associated prototype, so JavaScript can inherit dynamically only from one prototype chain.

### 20. What is an SPA? What are the major pros and cons of it?

SPA stands for a single-page application, and it is a web solution type that dynamically rewrites the current web page with new data from the client as contrasted to the default method when the browser loads entire new pages.

As for the advantages of SPAs, they are reactive, user-friendly, fast to load, easy to add advanced features, and more resource-efficient in terms of bandwidth usage. At the same time, they use a lot of browser resources and don’t perform quite well in terms of search engine optimization.

### 21. What is server-side rendering, and what problems does it solve?

Server-side rendering is a method that renders the React components on the server and brings HTML content as an output. It is the opposite of client-side rendering, where a browser downloads a minimal HTML page, renders the JavaScript code, and fills it out with the content. Server-side rendering solves the problem with search engine crawlers not understanding JavaScript, which is crucial for SEO results. Also, it can improve application performance, which is important for content-heavy websites.

### 22. What is CORS?

CORS stands for cross-origin resource sharing, and it uses additional HTTP headers, which make a browser grant a web app, which is running at one origin, access to some certain resources from a different origin. A cross-origin HTTP request is a request of resources from a domain, protocol, or port different from the origin domain, protocol, or port of the web app. For the sake of data security, browsers restrict cross-origin HTTP requests if they are initiated from scripts.

### 23. How do you create and manage user sessions properly? What is JWT?

[User sessions](https://docs.oracle.com/cd/E19683-01/817-2172-10/dwsessn.html) stand for the series of user interactions with the app, which are tracked by the server. They are used to maintain user-specific states, that is, persistent objects and authenticated user identities. JWT stands for JSON Web Token. It is an open standard for access tokens creation, which is based on the JSON format. Usually, it is used to transfer data for authentication in client-server apps. To keep users logged in when they navigate through different screens, JWT is generated on a server and sent to a client, where it is transformed into a cookie. This cookie keeps the user logged in. To drop a session, you use a logout method when JWT is dropped from cookies.

## **Network Data Transfer Basics — React Interview Questions**

### 24. What are the parts of an HTTP request?

An HTTP request message consists of the following parts: a request type, a series of HTTP headers or header fields, and a request body if needed.

### 25. What is the difference between HTTP2 and HTTP?

HTTP, which stands for Hypertext Transfer Protocol, is the foundation of data communication for the World Wide Web. HTTP has existed since the very dawn of the Internet, but five years ago, it was revolutionized, and HTTP2 was introduced. The goals of HTTP2 include a protocol negotiation mechanism, improved page load speed, compressed request headers, request multiplexing, request pipelining, Head-of-line blocking, etc..

### 26. What is REST and RESTful?

REST stands for a representational state transfer. It is a software architecture style, which introduces a set of constraints that should be used when creating web services. A web service that complies with REST standards is called RESTful.

### 27. What is a websocket, and how does it work?

Websocket is a computer communications protocol, which provides full-duplex communication channels over a single Transmission Control Protocol connection. This advanced technology enables bidirectional communication between a server and a browser — you can send messages to the server and get event-driven responses with no need to poll the server for a reply. A websocket starts as a normal HTTP request and response. Within this HTTP request-response chain, the client requests a websocket connection to be opened and awaits for the server to respond. If the server’s response is positive, a websocket connection between a server and a client is established. Data flows with the help if a basic framed message protocol, which is closed once both the server and the client decide to end the websocket connection.

### 28. What is a cookie, and what types of cookies are you familiar with?

A cookie is a small chunk of data, which is sent by a website and stored by the user’s browser. This data is used for a website to remember stateful information or to record users’ browsing activity. As for the types of cookies, there are session cookies, permanent cookies, same-site cookies, HTTP-only cookies, etc.

### 29. What is the difference between cookies and local storage?

Cookies and local storage are designed to serve different purposes: cookies are mostly read by the server-side, and local storage is read by the client-side. Also, local storage can store data of large sizes than cookies.

## **Security Basics — React Interview Questions**

### 30. What types of security attacks on the front-end are you aware of?

Major security concerns for a front-end developer are SQL injections, cross-site request forgeries, and cross-site scripting attacks.

### 31. What is brute force and how to avoid it? Have you heard about “salt and pepper” terms?

When somebody conducts a brute-force attack, they submit numerous passwords with the hope to eventually guess correctly. This attack can be used to break into a closed-access system or account or to decipher encrypted data. It is often used when the attacker sees no other weaknesses that they may tackle to breach the system. The simplest way to prevent brute-force attacks is to limit the number of failed login attempts. Using Captcha or two-factor authentication may also help.

As for salt and pepper, it is a password hashing terminology.

### 32. How can you protect your app from XSS?

To conduct a cross-site scripting attack, one injects a piece of malicious code, which runs a client-side script, into a legitimate web page. Once a user opens the infected website, the user’s browser downloads the script. To prevent an XSS attack, you should encode all variable strings before they will be displayed on the web page. In other words, you should convert every potentially dangerous character to an HTML entity. Also, you should limit input by types: a user can type only numbers into a number field and so on.

### 33. What do you know about CSRF?

CSRF stands for Cross-Site Request Forgery — this security attack forces a user to perform unwanted actions on a website. By doing a certain action, users can leak data, change the session state, or manipulate their own account without being aware of it. The malicious web request usually includes proper URL parameters, cookies, and other web data, so the server doesn’t recognize a forgery. By trusting an authorized user, the server executes the action they performed without asking to confirm it.

## Code Testing Basics — React Interview Questions

### 34. What is the difference between TDD and BDD?

TDD stands for test-driven development, and BDD stands for behavior-driven development. In both TDD and BDD, tests are written before development. Both TDD and BDD aim to prevent bugs, and both involve detailed documentation. Yet, the implementation of the two is different. In TDD, a developer writes an automated test case based on the project specifications. The development continues until this test is executed successfully. In BDD, a developer writes given-when-then expressions, which resemble tests but are written in simple English. Thus, TDD is aimed to test the feature implementation, and BDD tests feature behavior. Tests in TDD are written in a programming language, and tests in BDD are written in English.

### 35. What types of testing are you familiar with, and what is the purpose of each?

There are many different types of testing: manual and automated, functional and non-functional, scripted, exploratory, and adhoc. Among the functional types, there are unit, integration, system, smoke, regression, and acceptance testing. Among the non-functional testing types, there are UI/UX, compatibility, API, performance, security, and accessibility testing. A developer must know about these types: unit, functional, integration, end-to-end (e2e), and regression testing.

### 36. How to manage e2e tests from the front-end perspective? What tools would you use for it?

To manage end-to-end tests from the front-end perspective, you can use such tools as Enzyme, Jest and jasmine, testcafe/puppeteer/nightwatch.

================

1. What is the React Virtual DOM?

Virtual DOM is a concept where a virtual representation of the real DOM is kept inside the memory and is synced with the actual DOM by a library such as ReactDOM.

The virtual DOM is an object that represents the real DOM in the memory. Since DOM updates are an integral part of any web app but are the costliest operation in the world of frontend, the virtual DOM is utilized to check for parts of the app that need to be updated & update only those parts, thus significantly boosting performance.

2. Why do we need to transpile React code?

React code is written in JSX, but no browser can execute JSX directly as they are built to read-only regular JavaScript.

Thus we require to use tools like Babel to transpile JSX to JavaScript so that the browser can execute it.

3. What is the significance of keys in React?

Keys in React is used to identify unique VDOM Elements with their corresponding data driving the UI; having them helps React optimize rendering by recycling existing DOM elements.

Key helps React identify which items have changed, are added, or are removed, enabling it to reuse already existing DOM elements, thus providing a performance boost.

For example:

const Todos = ({ todos }) => {

return (

<div>

{todos.map((todo) => (

<li>{todo.text}</li>

))}

</div>

);

};

This would cause new DOM Elements to be created everytime todos change, but adding the key prop (<li key={todo.id}>{todo.text}</li>) would result in "dragging" around the DOM Elements inside the ul tag & updating only the necessary lis.

4. What is the significance of refs in React?

Refs are variables that allow you to persist data between renders, just like state variables, but unlike state variables, updating refs does NOT cause the component to re-render.

Refs are usually used to, but not restricted to, store reference to DOM elements.

5. What are the most common approaches for styling a React application?

**CSS Classes**

React allows class names to be specified for a component like class names are set for a DOM element in HTML. When developers first start using React after developing traditional web applications, they often opt for CSS classes as they are already familiar with the approach.

**Inline CSS**

Styling React elements using inline CSS allows styles to be completely scoped to an element. However, certain styling features are not available with inline styles. For example, the styling of pseudo-classes like :hover.

**Pre-processors (Sass, Stylus, and Less)**

Pre-processors are often used on React projects. This is because, like CSS, they are well understood by developers and are often already in use if React is being integrated into a legacy application.

**CSS-in-JS Modules** (Styled Components, Emotion, and Styled-jsx)

CSS-in-JS modules are a popular option for styling React applications because they integrate closely with React components. For example, they allow styles to change based on React props at runtime. Also, by default, most of these systems scope all styles to the respective component being styled.

6. What are some of the performance optimization strategies for React?

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

**useCallback** can be used to obtain a similar result.

**Lazy loading** is a technique used to reduce the load time of a React app. It helps reduce the risk of web app performances to a minimum, by loading up the components as the user navigates through the app.

7. What is prop drilling and how to avoid it?

Sometimes while developing React applications, there is a need to pass data from a component that is higher in the hierarchy to a component that is deeply nested. To pass data between such components, we pass props from a source component and keep passing the prop to the next component in the hierarchy till we reach the deeply nested component.

The disadvantage of using prop drilling is that the components that should otherwise be not aware of the data have access to the data, moreover, the code becomes harder to maintain.

Prop drilling can be avoided using the Context API or some form of State Management library.

8. What is the StrictMode component and why would you use it?

<StrictMode /> is a component included with React to provide additional visibility of potential issues in components. Suppose the application is running in development mode. In that case, any issues are logged to the development console, but these warnings are not shown if the application is running in production mode.

Developers use <StrictMode /> to find problems such as deprecated lifecycle methods and legacy patterns, to ensure that all React components follow current best practices.

<StrictMode /> can be applied at any level of an application component hierarchy, which allows it to be adopted incrementally within a codebase.

9. What are synthetic events in React?

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

const Component = () => {

const handleClick = (e) => {

e.preventDefault(); // synthetic event

console.log("link clicked");

};

return <a onClick={(e) => handleClick}>Click me</a>;

};

10. Why is it not advisable to update state directly, but use the setState call?

The conventional way to update state is to use the setState call. Without using it, the user would still be able to modify the state, but it would not update the DOM to reflect the new state.

const Component = () => {

const [count, setCount] = useState(0);

// let [count, setCount] = React.useState(0);

const handleClickUpdate = () => {

setCount((c) => c + 1);

// count = count + 1; // will not update the DOM

};

return (

<>

{count}

<button onClick={handleClickUpdate}>Click me</button>

</>

);

};

11. What are portals in React?

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

const Portal = ({ children }) => {

// NOTE: it is advisable to create a new DOM node for the portal

const portalRoot = document.getElementById("portal-root");

return ReactDOM.createPortal(children, portalRoot);

};

====================

1. Explain the Virtual DOM, and a pragmatic overview of how React renders it to the DOM.

The Virtual DOM is an interesting concept; it’s a complex idea that boils down into a much simpler algorithm.

In React, if we create simple ES6 class and print it out, we have a function (as all functions can be used as a constructor in JavaScript):

const app = () => {

let React = react,

{Component} = React,

DOM = reactDom

class Comments extends Component {

constructor(props){ super(props) }

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

}

console.log(Comments)

}

require('react', 'react-dom').then(app)

The console.log(Comments) gives us something that looks like this (after compiled by Babel from ES6 to ES5):

function Comments(props) {

\_classCallCheck(this, Comments);

return \_possibleConstructorReturn(this, Object.getPrototypeOf(Comments).call(this, props));

}

When we write something to draw a React Component to the screen, we might have something like the following:

DOM.render(<Comments />, document.body)

The JSX gets transpiled into ES5 by Babel as well:

DOM.render(React.createElement(Comments, null), document.body);

We can see that <Comments /> is transpiled directly into React.createElement(Comments, null). This is where we can see what a Virtual DOM object actually *is*: a plain JavaScript Object that represents the tag to be rendered onto the screen.

Let’s inspect the output of React.createElement():

console.log(<div/>)

// or

console.log(React.createElement('div', null))

This gives us:

{"type":"div","key":null,"ref":null,"props":{},"\_owner":null,"\_store":{}}

See how the type is a string? DOM.render({...}) gets this object above and looks at the type, and decides whether or not to reuse an existing <div> element on the DOM or create a new <div> and append it.

The Virtual DOM is not a simple Object – it is a recursive structure. For example, if we add two elements beneath the <div/>:

console.log(<div><span/><button/></div>)

// or

console.log(React.createElement(

'div',

null,

React.createElement('span', null),

React.createElement('button', null)

))

What we get is a nested Object-tree:

{

"type":"div",

"key":null,

"ref":null,

"props":{

"children": [

{"type":"span","key":null,"ref":null,"props":{}},

{"type":"button","key":null,"ref":null,"props":{}}

]

}

}

This is why, in a React Component’s code, we can access the child and ancestor elements via this.props.children. What React will do is walk down a very deep tree of nested Objects (depending on your UI complexity), each sitting in their parent element’s children.

One thing to note is that the type so far has just been a string. When a React Element is made from a custom Component (like Comments above), the type is a function:

console.log(<Comments />)

// or

console.log(React.createElement(Comments, null))

gives us:

{

"key":null,

"ref":null,

"props":{},

“type”: function Component() { ... }

}

You can play around with a web version of this code [at Matthew Keas’ github](https://goo.gl/HZZMjv).

2.Explain the standard JavaScript toolchain, transpilation (via Babel or other compilers), JSX, and these items’ significance in recent development. What sort of tools might you use in the build steps to optimize the compiled output React code?

The bleeding edge JavaScript toolchain can seem quite complex, and it’s very important to feel confident in the toolchain and to have a mental picture of how the pieces fit together. There are a couple primary pillars in the JavaScript toolchain: Dependency Management, Linting, Style-checking, Transpilation, and Compilation, Minification, Source-Mapping.

Typically, we use build tools like Gulp, Watchify/Browserify, Broccoli, or Webpack to *watch the filesystem* for file events (like when you add or edit a file). After this occurs, the build tool is configured to carry out a group of *sequential or parallel tasks*.

This part is the most complex piece, and is the center of the development process.

The rest of the tools belong in that group of sequential or parallel tasks:

* Style linting - typically a linter like JSCS is used to ensure the source code is following a certain structure and style
* Dependency Management - for JavaScript projects, most people use other packages from npm; some plugins exist for build systems (e.g. Webpack) and compilers (e.g. Babel) that allow automatic installation of packages being imported or require()‘d
* Transpilation - a specific sub-genre of compilation, transpilation involves compiling code from one source version to another, only to a similar runtime level (e.g. ES6 to ES5)
* Compilation - specifically separate from transpiling ES6 and JSX to ES5, is the act of including assets, processing CSS files as JSON, or other mechanisms that can load and inject external assets and code into a file. In addition, there are all sorts of build steps that can analyze your code and even optimize it for you.
* Minification and Compression - typically part of – but not exclusively controlled by – compilation, is the act of minifying and compressing a JS file into fewer and/or smaller files
* Source-Mapping - another optional part of compilation is building source maps, which help identify the line in the original source code that corresponds with the line in the output code (i.e. where an error occurred)

For React, there are specific build tool plugins, such as the [babel-react-optimize presets](https://github.com/thejameskyle/babel-react-optimize) that involves compiling code into a format that optimizes React, such as automatically compiling any React.createElement() calls into a JavaScript Object that inlines right into the source code:

class MyComponent extends React.Component {

render() {

return (

<div className={this.props.className}> <span>Hello World</span> </div>

);

}

}

becomes

class MyComponent extends React.Component {

render() {

return (

\_jsx('div', { className: this.props.className }, void 0,

\_jsx('span', {}, void 0, 'Hello World')

)

);

}

}

3. How would you create Higher Order Components (HOCs) in React?

Higher Order Components (HOCs) are the coined term for a custom Component that accepts dynamically provided children. For example, let’s make <LazyLoad /> Component that takes child image tags as children, waits until the <LazyLoad /> Component is scrolled into view, and then loads the images they point to in the background (before rendering them to the DOM).

An HOC accepts children via props:

DOM.render(

<LazyLoad>

<img src="https://media.giphy.com/media/HhvUpQhBWMtwc/200.gif"/>

<img src="https://media2.giphy.com/media/3oEduUDvycvu3GYkdG/200w.gif"/>

<img src="https://media0.giphy.com/media/142UITjG5GjIRi/200w.gif" />

</LazyLoad>,

document.body)

Creating an HOC means handling this.props.children in the Component’s code:

interactive example can be found at <https://goo.gl/ns0B6j>

class LazyLoad extends Component {

constructor(p){

super(p)

this.state = { loaded:0 }

this.\_scroll = this.\_scroll.bind(this)

}

\_scroll(){

let el = DOM.findDOMNode(this)

let {top} = el.getBoundingClientRect()

let viewportHeight = Math.max(document.documentElement.clientHeight, window.innerHeight || 0)

if(top < (viewportHeight + this.props.top)) {

window.removeEventListener('scroll', this.\_scroll)

this.setState({loaded:1})

}

}

componentDidMount(){

window.addEventListener('scroll', this.\_scroll)

this.\_scroll()

}

componentWillUnmount(){

window.removeEventListener('scroll', this.\_scroll)

}

render(){

let {children} = this.props,

{loaded} = this.state

return <div>

{loaded && children}

</div>

}

}

LazyLoad.defaultProps = { top: 100 }

Noting a few things about this code:

1. We set up initial state (this.state = {loaded: 0}) in the constructor(). This will be set to 1 when the parent container is scrolled into view.
2. The render() returns the props.children as child elements when this occurs. Extract the src by using ES6 destructuring, where {props:{src}} creates a variable src with the appropriate value.
3. We used a single componentDidMount() lifecycle method. This is used because on mount, we’d like the component to check if the HOC is visible.
4. The largest function of our component, \_scroll(), grabs the HOC Component’s DOM element with DOM.findDOMNode() and then gets the elements position. This position is compared to the height of the browser window, and if it is less than 100px from the bottom, then the scroll listener is removed and loaded is set to 1.

This technique is called HOC (Higher Order Component) because we pass in elements as this.props.children when we nest those elements inside the container component:

<HOC>

<div>some</div>

<span>children</span>

<Props/>

</HOC>

All of these nested elements (which can be custom components) are nested under <HOC/>, thus HOC’s code will be able to access them as this.props.children.

4. What is the significance of keys in React?

Keys in React are used to identify unique VDOM Elements with their corresponding data driving the UI; having them helps React optimize rendering by recycling existing DOM elements. Let’s look at an example to portray this.

We have two <TwitterUser> Components being rendered to a page, drawn in decreasing order of followers:

-----------

| A - 103 |

-----------

-----------

| B - 92 |

-----------

Let’s say that B gets updated with 105 Twitter followers, so the app re-renders, and switches the ordering of A and B:

-----------

| B - 105 |

-----------

-----------

| A - 103 |

-----------

Without keys, React would primarily re-render both <TwitterUser> Elements in the DOM. It would re-use DOM elements, but React won’t re-order DOM Elements on the screen.

With keys, React would actually re-order the DOM elements, instead of rendering a lot of nested DOM changes. This can serve as a huge performance enhancement, especially if the DOM and VDOM/React Elements being used are costly to render.

Keys themselves should be a unique number or string; so if a React Component is the only child with its key, then React will repurpose the DOM Element represented by that key in future calls to render().

Let’s demonstrate this with a simple list of todos rendered with React:

Interactive code sample available on [Matthew Keas’ github](https://goo.gl/fpAvSc).

class List extends Component {

constructor(p){

super(p)

this.state = { items: Array(5).fill(1).map((x,i) => ({id:i})) }

}

componentDidMount(){

const random = (a,b) => Math.random() < .5 ? -1 : 1

setInterval(() => {

this.setState({ items: this.state.items.sort(random) }) }, 20)

}

render() {

let {items} = this.state

return <ul>

{items.map(item => <li key={item.id}>{item.id}</li>)}

</ul>

}

}

DOM.render(<List />, document.body)

The setInterval() occurring on mount reorders the items array in this.state every 20ms. Computationally, if React is reordering the items in state, then it would manipulate the DOM elements themselves instead of “dragging” them around between positions in the <ul>.

It is worth noting here that if you render a homogenous array of children – such as the <li>’s above – React will actually console.warn() you of the potential issue, giving you a stack trace and line number to debug from. You won’t have to worry about React quietly breaking.

**5. What is the significance of refs in React?**

Similarly to keys, refs are added as an attribute to a React.createElement() call, such as <li ref="someName"/>. The ref serves a different purpose, it provides us quick and simple access to the DOM Element represented by a React Element.

Refs can be either a string or a function. Using a string will tell React to automatically store the DOM Element as this.refs[refValue]. For example:

class List extends Component {

constructor(p){ super(p) }

\_printValue(){ console.log(this.refs.someThing.value) }

render() {

return <div onClick={e => this.\_printValue()}>

<p>test</p>

<input type="text" ref="someThing" />

</div>

}

}

DOM.render(<List />, document.body)

this.refs.someThing inside componentDidUpdate() used to refer to a special identifier that we could use with React.findDOMNode(refObject) – which would provide us with the DOM node that exists on the DOM at this very specific instance in time. Now, React automatically attaches the DOM node to the ref, meaning that this.refs.someThing will directly point to a DOM Element instance.

Additionally, a ref can be a function that takes a single input. This is a more dynamic means for you assign and store the DOM nodes as variables in your code. For example:

class List extends Component {

constructor(p){ super(p) }

\_printValue(){

console.log(this.myTextInput.value)

}

render() {

return <div onClick={e => this.\_printValue()}>

<p>test</p>

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

</div>

}

}

DOM.render(<List />, document.body)

6. [Legacy projects only, < circa 2016] In a general overview, how might React Router and its techniques differ from more traditional JavaScript routers like Backbone’s Router?

“Traditional” routers like the ever-popular [Backbone.Router](http://backbonejs.org/#Router) establish a predefined set of routes, in which each route defines a series of actions to take when a route is triggered. When combining Backbone.Router with React, you may have to mount and unmount React Components when the route changes:

var MyRouter = Backbone.Router.extend({

routes: {

'home': 'showHome',

'search/:q': 'showSearch',

'\*default': 'show404'

},

showHome(){

DOM.unmountComponentAtNode(document.body)

DOM.render(<Home />, document.body)

},

showSearch(q){

DOM.unmountComponentAtNode(document.body)

DOM.render(<Search query={q} />, document.body)

},

show404(){

DOM.unmountComponentAtNode(document.body)

DOM.render(<Error />, document.body)

}

})

The router exists externally of the React Components, and the VDOM has to mount and unmount potentially frequently, introducing a possible slew of problems. React Router focuses on not just “single-level” routing, but enables - nay, *empowers* - the creation of HOCs that can “decide for themselves” what to render within them.

This is where the advanced HOC implementations can really help simplify a seemingly complex notion. Let’s look at using a tiny router to assess some of the beauty of embedding application routers inside React HOCs. Here, we define a Component that wraps it’s own routing mechanism (router() not provided here, see [universal-utils](https://github.com/matthiasak/universal-utils/blob/master/src/router-alt.js)):

// router(routesObject, callback) --> when a route event occurs, we invoke callback() with

// the React Element and the props passed via the route params

class Router extends Component {

constructor(...a){

super(...a)

let p = this.props

this.state = {

routes: p.routes || {},

default: p.default || '/'

}

this.router = router(this.state.routes, (el, props) => {

this.current = el

})

this.router.trigger(this.state.default)

}

render(){

return this.current()

}

}

DOM.render(<Router routes={{

'/': () => <Home/>,

'/search/:q': ({q}) => <Search query={q} />,

'\*': () => <Error />

}}/>, document.body)

This Router Component opts for parsing the routes object passed into this.props instead of reading over an array of React Components passed as this.props.children. React Router opts for the latter technique. Need proof? Take a look at this example code provided by [React Router’s docs](https://github.com/reactjs/react-router):

DOM.render(

<Router history={browserHistory}>

<Route path="/" component={App}>

<Route path="about" component={About}/>

<Route path="users" component={Users}>

<Route path="/user/:userId" component={User}/>

</Route>

<Route path="\*" component={NoMatch}/>

</Route>

</Router>

, document.body)

A <Router /> Component has one or more <Route /> Components as items in this.props.children, and <Route />s can have sub-<Route />s. React Router’s code recursively walks down the tree of children until there are no more to process, allowing the developer to recursively declare routes in a structure that encapsulates sub-routes, instead of having to implement a Backbone-esque flat list of routes (i.e. "/", "/about", "/users", "/users/:id", etc).

7. Why do class methods need to be bound to a class instance, and how can you avoid the need for binding?

In JavaScript, the value of this changes depending on the current context. Within React class component methods, developers normally expect this to refer to the current instance of a component, so it is necessary to bind these methods to the instance. Normally this is done in the constructor—for example:

class SubmitButton extends React.Component {

constructor(props) {

super(props);

this.state = {

isFormSubmitted: false

};

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

}

handleSubmit() {

this.setState({

isFormSubmitted: true

});

}

render() {

return (

<button onClick={this.handleSubmit}>Submit</button>

)

}

}

There are several common approaches used to avoid this binding:

### 1. Define Your Event Handler as an Inline Arrow Function

For example:

class SubmitButton extends React.Component {

constructor(props) {

super(props);

this.state = {

isFormSubmitted: false

};

}

render() {

return (

<button onClick={() => {

this.setState({ isFormSubmitted: true });

}}>Submit</button>

)

}

}

Using an arrow function like this works because arrow functions do not have their own this context. Instead, this will refer to the context in which the arrow function was defined—in this case, the current instance of SubmitButton.

### 2. Define Your Event Handler as an Arrow Function Assigned to a Class Field

class SubmitButton extends React.Component {

state = { isFormSubmitted: false }

handleSubmit = () => {

this.setState({ isFormSubmitted: true });

}

render() {

return (

<button onClick={this.handleSubmit}>Submit</button>

)

}

}

### 3. Use a Function Component with Hooks

Using the hooks functionality in React it is possible to use state without using this, which simplifies component implementation and unit testing.

For example:

const SubmitButton = () => {

const [isFormSubmitted, setIsFormSubmitted] = useState(false);

return (

<button onClick={() => { setIsFormSubmitted(true); }}>Submit</button>

)

};

8. Explain the positives and negatives of shallow rendering components in tests.

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.

9. If you wanted a component to perform an action only once when the component initially rendered—e.g., make a web analytics call—how would you achieve this with a class component? And how would you achieve it with a function component?

### Using a Class Component

The componentDidMount() lifecycle hook can be used with class components:

class Homepage extends React.Component {

componentDidMount() {

trackPageView('Homepage');

}

render() {

return <div>Homepage</div>;

}

}

Any actions defined within a componentDidMount() lifecycle hook are called only once when the component is first mounted.

### Using a Function Component

The useEffect() hook can be used with function components:

const Homepage = () => {

useEffect(() => {

trackPageView('Homepage');

}, []);

return <div>Homepage</div>;

};

The useEffect() hook is more flexible than the lifecycle methods used for class components. It receives two parameters:

1. The first parameter it takes is a callback function to be executed.
2. The optional second parameter it takes is an array containing any variables that are to be tracked.

The value passed as the second argument controls when the callback is executed:

* If the second parameter is undefined, the callback is executed every time that the component is rendered.
* If the second parameter contains an array of variables, then the callback will be executed as part of the first render cycle and will be executed again each time an item in the array is modified.
* If the second parameter contains an empty array, the callback will be executed only once as part of the first render cycle. The example above shows how passing an empty array can result in similar behaviour to the componentDidMount() hook within a function component.

10. What are the most common approaches for styling a React application?

### CSS Classes

React allows class names to be specified for a component, like class names are specified for a DOM element in HTML.

When developers first start using React after developing traditional web applications, they often use CSS classes for styling because they are already familiar with the approach.

### Inline CSS

Styling React elements using inline CSS allows styles to be completely scoped to an element using a well-understood, standard approach. However, there are certain styling features that are not available with inline styles. For example, the styling of :hover pseudo-classes.

Pre-processors Such as Sass, Stylus, and Less

Pre-processors are often used on React projects. This is because, like CSS, they are well understood by developers and are often already in use if React is being integrated into a legacy application.

### CSS-in-JS Modules Such as Styled Components, Emotion, and Styled-jsx

CSS-in-JS modules are a popular option for styling React applications because they integrate closely with React components. For example, they allow styles to change based on React props at runtime. Also, by default, most of these systems scope all styles to the respective component being styled.

11. If you were working on a React application that was rendering a page very slowly, how would you go about investigating and fixing the issue?

If a performance issue such as slow rendering is seen within a React app, the first step is to use the Profiler tool provided within the React Developer Tools browser plugin, which is available for Google Chrome and Mozilla Firefox. The Profiler tool allows developers to find components that take a long time to render or are rendering more frequently than necessary.

One of the most common issues in React applications is when components re-render unnecessarily. There are two tools provided by React that are helpful in these situations:

* React.memo(): This prevents unnecessary re-rendering of function components
* PureComponent: This prevents unnecessary re-rendering of class components

Both of these tools rely on a shallow comparison of the props passed into the component—if the props have not changed, then the component will not re-render. While both tools are very useful, the shallow comparison brings with it an additional performance penalty, so both can have a negative performance impact if used incorrectly. By using the React Profiler, performance can be measured before and after using these tools to ensure that performance is actually improved by making a given change.

12. At a high level, what is the virtual DOM (VDOM) and how does React use it to render to the DOM?

The VDOM is a programming concept, providing a critical part of the React architecture. Rather than interacting directly with the DOM, changes are instead first rendered to the VDOM—a lightweight representation of the target state of the DOM.

Changes made to the VDOM are batched together to avoid unnecessary frequent changes to the DOM. Each time these batched changes are persisted to the DOM, React creates a diff between the current representation and the previous representation persisted to the DOM, then applies the diff to the DOM.

This abstraction layer for the DOM provides a simple interface for developers while allowing React to update the DOM in an efficient and performant manner.

13. What is prop drilling and how can you avoid it?

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.

Consider the following example components:

* <EditUsersPage />, which includes selectedUserAddress in its component state and renders a <User /> component
* <User />, which renders a <UserDetails /> component
* <UserDetails />, which renders a <UserAddress /> component
* A <UserAddress /> component that requires the selectedUserAddress property stored in the <EditUsersPage /> state

The simplest approach is to simply pass a selectedUserAddress 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—in this case <User /> and <UserDetails />—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.

While context can be used directly for sharing global state, it is also possible to use context indirectly via a state management module, such as Redux.

14. What is the StrictMode component and why would you use it?

<StrictMode /> is a component included with React to provide additional visibility of potential issues in components. If the application is running in development mode, any issues are logged to the development console, but these warnings are not shown if the application is running in production mode.

Developers use <StrictMode /> to find problems such as deprecated lifecycle methods and legacy patterns, to ensure that all React components follow current best practices.

<StrictMode /> can be applied at any level of an application component hierarchy, which allows it to be adopted incrementally within a codebase.

15. What is the key architectural difference between a JavaScript library such as React and a JavaScript framework such as Angular? How would that impact the decision for a project to use one versus the other?

React enables developers to render a user interface. To create a full front-end application, developers need other pieces, such as state management tools like Redux.

Like React, Angular enables developers to render a user interface, but it is a “batteries included” framework that includes prescriptive, opinionated solutions to common requirements like state management.

While there are many other considerations when comparing React and Angular specifically, this key architectural difference means that:

* Using a library such as React can give a project a greater ability to evolve parts of the system—again for example, state management—over time, when new solutions are created by the open source community.
* Using a framework such as Angular can make it easier for developers to get started and can also simplify maintenance.

16. How can automated tooling be used to improve the accessibility of a React application?

There are two main categories of automated tools that can be used to identify accessibility issues:

### Static Analysis Tools

Linting tools like ESLint can be used with plugins such as eslint-plugin-jsx-a11y to analyse React projects at a component level. Static analysis tools run very quickly, so they bring a good benefit at a low cost.

### Browser Tools

Browser accessibility tools such as aXe and Google Lighthouse perform automated accessibility at the app level. This can discover more real-world issues, because a browser is used to simulate the way that a user interacts with a website. It is possible for many of these tools to run in a continuous integration environment such as Travis or Jenkins. Since these tools take longer to execute, many developers just run these tools within their local browser on an occasional basis, such as when reaching project milestones.

17. [Legacy projects only: React < 16.8] What are pure functional Components?

Traditional React Components as we have seen thus far are creating a class with class Example extends React.Component or React.createClass(). These create stateful components if we ever set the state (i.e. this.setState(), getInitialState(), or this.state = {} inside a constructor()).

If we have no intention for a Component to need state, or to need lifecycle methods, we can actually write Components with a pure function, hence the term “pure functional Component”:

function Date(props){

let {msg="The date is:"} = props

let now = new Date()

return <div>

<span>{msg}</span>

<time>{now.toLocaleDateString()}</time>

</div>

}

This function that returns a React Element can be used whereever we see fit:

DOM.render(<div><Date msg="Today is"/><div>)

You might notice that <Date/> also takes a prop – we can still pass information into the Component.

18. How might React handle or restrict Props to certain types, or require certain Props to exist?

You may recall a previous example that looked like the following (some parts of the code left out):

class LazyLoad extends Component {

constructor(p){

super(p)

this.state = { loaded:0 }

}

render(){

let {children} = this.props,

{loaded} = this.state

return <div>

{loaded && children}

</div>

}

}

When rendering the <LazyLoad/>, we can pass in props (i.e. <LazyLoad top={0}/>). Props are essentially inputs or values being passed down to one Component from the parent rendering context, and the code that passes the props to the element may not be compliant with your code. For example, top here seems to be just a number, but would I be able to verify that the prop is in-fact a number before my component is rendered? It’s certainly possible to write this code in each and every Component that uses props. However, React provides us a much simpler and shorter solution: Prop Types.

let p = React.PropTypes

LazyLoad.PropTypes = {

top: p.number

}

When using React’s non-minified development version (i.e. when building and testing in development), React will throw an error to alert you of any instances where a Prop is either missing or the wrong type. Above, top should always be a number.

We can make top a *required* prop by adding:

let p = React.PropTypes

LazyLoad.PropTypes = {

top: p.number.isRequired

}

PropTypes can be used [to test Props for any kind of value](https://reactjs.org/docs/typechecking-with-proptypes.html). Here’s a few quick type-checkers React has for JavaScript’s built-in types:

* React.PropTypes.array,
* React.PropTypes.bool,
* React.PropTypes.func,
* React.PropTypes.number,
* React.PropTypes.object,
* React.PropTypes.string,
* React.PropTypes.symbol,

We can also test that props are React and DOM types:

* React.PropTypes.node,
* React.PropTypes.element,

And we have the ability to test more complex types, such as “shapes”, “instances of”, or “collections of”:

* React.PropTypes.instanceOf(Message),
* React.PropTypes.oneOf(['News', 'Photos']),
* React.PropTypes.oneOfType([ React.PropTypes.string, React.PropTypes.number, React.PropTypes.instanceOf(Message)])
* React.PropTypes.arrayOf(React.PropTypes.number),
* React.PropTypes.shape({ color: React.PropTypes.string, fontSize: React.PropTypes.number })

Use these PropTypes to produce errors and track down bugs. When used effectively, PropTypes will prevent your team from losing too much time in the debugging and documentation process, ensuring stricter standards and understanding of your growing library of Components.

19. [Legacy projects only: React < 15.5] Compare and contrast creating React Components in ES5 and ES2015 (also known as ES6). What are the advantages and disadvantages of using one or the other? Include notes about default props, initial state, PropTypes, and DisplayName.

Creating React Components the ES5 way involves using the React.createClass() method:

var Comments = React.createClass({

displayName: 'Comments',

getInitialState: function(){

return {comments: []}

},

getDefaultProps: function(){

return {some\_object: {a:1, b:2, c:3}}

},

\_handleClick: function(){

alert('hello world!')

},

render: function(){

return <div>

There are {this.state.comments.length} comments

<button onClick={this.\_handleClick}>click me!</button>

</div>

}

})

This Comments Component can now be rendered either inside another React Component or directly in the call to ReactDOM.render():

ReactDOM.render(<Comments />, document.querySelector('.app'))

ES5 Components have some particular qualities, which we’ll note:

1. Like the above example, to set the state to an initial value, create the getInitialState() function on the Component. What it returns will be the initial state for a Component when rendered.
2. Additionally, you can set the default props for the component to have a certain value with the getDefaultProps() method on the ES5 version.
3. The displayName is used in debugging and error reporting by React. If you use JSX, then the displayName is automatically filled out.
4. For some, it is common practice to denote a custom method added to a React Component by prefixing it with an underscore, hence \_handleClick. \_handleClick is passed as the onClick callback for a button in the code above. We can’t do this so easily in the ES6 API of React, because the ES5 version has *autobinding*, but the ES6 does not. Let’s take a look at what autobinding provides:

Auto-binding

Consider the following piece of code:

var thing = {

name: 'jen',

speak: function(){ console.log(this.name) }

}

window.addEventListener('keyup', thing.speak)

Invoking thing.speak() in the console will log "jen", but pressing a key will log undefined because the *context* of the callback is the global object. The browser’s global object – window – becomes this inside the speak() function, so this.name becomes window.name, which is undefined.

React in ES5 automatically does autobinding, effectively doing the following:

window.addEventListener('keyup', thing.speak.bind(thing))

*Autobinding* automatically binds our functions to the React Component instance so that passing the function by reference in the render() works seamlessly.

Creating React Components the ES6 way works a little differently, favoring the ES6 class ... extends ... syntax, and no autobinding feature:

class Comments extends React.Component {

constructor(props){

super(props)

this.state = {comments: []}

}

\_handleClick(){

alert('hello world!')

}

render(){

return <div>

There are {this.state.comments.length} comments

<button onClick={() => this.\_handleClick}>click me!</button>

</div>

}

}

Comments.defaultProps = {a:1, b:2, c:3}

Comments.displayName = 'Comments'

1. Notice that in ES6, we have a constructor() that we use to set the initial state,
2. We can add default props and a display name as properties of the new class created, and
3. The render() method, which works as normal, but we’ve had to alter how we pass in the callback function. This current approach (<button onClick={() => this.\_handleClick}>click me!</button>) will create a new function each time the component is re-rendered; so if it becomes a performance bottleneck, you can always bind manually and store the callback:

class Comments extends React.Component {

constructor(...args) {

super(...args);

this.state = { toggledOn: false };

this.\_handleClick = this.\_handleClick.bind(this);

}

\_handleClick() {

this.setState(prevState => ({ toggledOn: !prevState.toggledOn });

}

render() {

return <button onClick={this.\_handleClick}> { this.state.toggledOn ? 'ON' : 'OFF' } </button>

}

}

Or with class fields syntax:

class Comments extends React.Component {

state = { toggledOn: false };

\_handleClick = () => {

this.setState(prevState => ({ toggledOn: !prevState.toggledOn }));

};

render() {

return <button onClick={this.\_handleClick}> {this.state.toggleOn ? 'ON' : 'OFF' </button>

}

}

Many React utility libraries on npm provide a single function to bind all handlers in the constructor, just like React does.

20. [Legacy projects only: React < 15.5] Compare and contrast incorporating mixins and enforcing modularity in React Components. (extend, createClass and mixins, HOCs) Why would you use these techniques, and what are the drawbacks of each?

Modularity is – in effect – something partially done with intention while coding, and partially done when refactoring afterwards.

Let’s first paint a scenario which we’ll model using each method above. Imagine we have three React Components: onScrollable, Loadable, and Loggable.

* an onScrollable Component will listen to the window.onscroll event, and use a logging mechanism to record it
* a Loadable Component will not render until one or more async requests have finished loading, and will use a logging mechanism to record when this occurs
* a Loggable Component provides a logging mechanism, be it a console, a [Winston Node.js logging setup](https://github.com/winstonjs/winston) on our own server, or some 3rd party logging service that records logs via JSON requests

First, let’s model this with React’s ES5 API and [mixins](https://facebook.github.io/react/docs/reusable-components.html#mixins).

Interactive code sample at [Matthew Keas’ github](https://goo.gl/kSIJe0).

var onKeypress = {

componentDidMount(){

this.onpress && window.addEventListener('keyup', this.onpress)

},

componentWillUnmount(){

this.onpress && window.removeEventListener('keyup', this.onpress)

}

}

var Loadable = {

componentDidMount(){

if(this.load){

this.setState({loaded: false})

Promise.all([].concat(this.load))

.then(() =>

this.setState({loaded: true}))

}

}

}

var Loggable = {

log(...args) {

alert(args)

}

}

var Example = React.createClass({

mixins: [Loggable, Loadable, onKeypress],

componentWillMount(){

this.onpress = (e) => this.log(e.which, 'pressed!')

this.load = [new Promise((res,rej) => setTimeout(res, 3000))]

this.log = (...args) => console.log(...args)

},

getInitialState(){

return {}

},

render() {

if(!this.state.loaded)

return <div>loading...</div>

return <div>test</div>

}

})

DOM.render(<Example />, document.body)

Let’s note a few things about the above code:

1. There are three POJOs (Plain ol’ JS Objects) created, which hold lifecycle and/or custom methods.
2. When creating the Example Component, we add mixins: [Loggable, Loadable, onKeypress], meaning that any functions from all three objects are included in the Example class.
3. Both onKeypress and Loadable add a componentDidMount(), but this doesn’t mean the latter cancels out the prior. In fact, all componentDidMount() functions from each mixin will be invoked when the event occurs. The same is true for all lifecycle methods added to mixins. This way, both the onKeypress and Loadable mixins will work simultaneously!

Mixins are possible, but not built-in to React’s ES6 API. However, the ES6 API makes it easier to create a custom Component that extends another custom Component.

So our Components’ prototype chains would look like the following:

[Example] --- extends ---> [Loggable] --- extends ---> [Loadable] --- extends ---> [onKeypress]

This would result from Components written as such:

class onKeypress {}

class Loadable extends onKeypress {}

class Loggable extends Loadable {}

class Example extends Loggable {}

Creating anonymous classes would help here, because then Loggable would not have to extend Loadable *and* onKeypress.

class Example extends (class a extends Loggable extends ...) { }

With a mixin() function, this could look more like:

class Example extends mixin(Loggable, Loadable, onKeypress) { }

Let’s try to write mixin() by building a chain of anonymous classes that extend Loggable, Loadable, and onKeypress:

const mixin = (...classes) =>

classes.reduce((a,v) => {

return (class temp extends a)

}, (class temp {}))

There’s a caveat, though – if Loadable extends onKeypress and both implement componentDidMount(), Loadable’s version will be lower on the prototype chain, which means the function from onKeypress will never be invoked.

The takeaway here is that the mixin pattern isn’t easily implemented by relying only on the ES6 extends approach. Let’s try to implement mixin() again, but build a more robust function:

const mixin = (...classes) => {

class \_mixin {}

let proto = \_mixin.prototype

classes.map(({prototype:p}) => {

Object.getOwnPropertyNames(p).map(key => {

let oldFn = proto[key] || (() => {})

proto[key] = (...args) => {

oldFn(...args)

return p[key](...args)

}

})

})

return \_mixin

}

This new mixin() implementation maps over each class, and cascades function calls from a parent class’s componentDidMount() alongside the child’s componentDidMount().

There are similar implementations of mixin() available on npm, using packages like [react-mixin](https://www.npmjs.com/package/react-mixin) and [es6-react-mixins](https://www.npmjs.com/package/es6-react-mixins).

We use mixin() from above like so:

interactive code sample available at <https://goo.gl/VnQ21R>

class A {

componentDidMount(){ console.log(1) }

}

class B {

componentDidMount(){ console.log(2) }

}

class C extends mixin(A,B) {

componentDidMount(...p){

super.componentDidMount(...p)

console.log(3)

}

}

let c = new C()

c.componentDidMount() // logs 1, 2, 3

Recently, React provided support for – and documented its preference of – React Components declared with ES6 classes. ES6 classes allow us to create component heirarchies with less code, however this makes it more difficult to create a single Component that inherits properties from several mixins, instead forcing us to create prototype chains.

21.[Legacy projects only: React < 16] Compare and contrast the various React Component lifecycle methods. How might understanding these help build certain interfaces/features?

There are several React lifecycle methods that help us manage the asynchronous and non-determinate nature of a Component during it’s lifetime in an app – we need provided methods to help us handle when a component is created, rendered, updates, or removed from the DOM.

Let’s first classify and define the life-cycle methods:

The “Will’s” - invoked right before the event represented occurs.

* componentWillMount() - Invoked once, both on the client and server, immediately before the initial rendering occurs. If you call setState within this method, render() will see the updated state and will be executed only once despite the state change.
* componentWillReceiveProps(object nextProps) - Invoked when a component is receiving new props. This method is not called for the initial render. Calling this.setState() within this function will not trigger an additional render. One common mistake is for code executed during this lifecycle method to assume that props have changed.
* componentWillUnmount() - Invoked immediately before a component is unmounted from the DOM. Perform any necessary cleanup in this method, such as invalidating timers or cleaning up any DOM elements that were created in componentDidMount.
* componentWillUpdate(object nextProps, object nextState) - Invoked immediately before rendering when new props or state are being received. This method is not called for the initial render.

The “Did’s”

* componentDidMount() - Invoked once, only on the client (not on the server), immediately after the initial rendering occurs. At this point in the lifecycle, you can access any refs to your children (e.g., to access the underlying DOM representation). The componentDidMount() method of child components is invoked before that of the parent component.
* componentDidUpdate(object prevProps, object prevState) - Invoked immediately after the component’s updates are flushed to the DOM. This method is not called for the initial render. Use this as an opportunity to operate on the DOM when the component has been updated.

The “Should’s”

* shouldComponentUpdate(object nextState, object nextProps) - Invoked before rendering when new props or state are being received. This method is not called for the initial render or when forceUpdate() is used. Use this as an opportunity to return false when you’re certain that the transition to the new props and state will not require a component update.

Having a strong understanding of how these fit together – and how setState() or forceUpdate() affect the lifecycle – will help the conscious React developer build robust UIs.

### 1. What is React?

React is a front-end and open-source JavaScript library which is useful in developing user interfaces specifically for applications with a single page. It is helpful in building complex and reusable user interface(UI) components of mobile and web applications as it follows the component-based approach.

The important features of React are:

* It supports server-side rendering.
* It will make use of the virtual DOM rather than real DOM (Data Object Model) as RealDOM manipulations are expensive.
* It follows unidirectional data binding or data flow.
* It uses reusable or composable UI components for developing the view.

### 2. What are the advantages of using React?

MVC is generally abbreviated as Model View Controller.

* Use of Virtual DOM to improve efficiency: React uses virtual DOM to render the view. As the name suggests, virtual DOM is a virtual representation of the real DOM. Each time the data changes in a react app, a new virtual DOM gets created. Creating a virtual DOM is much faster than rendering the UI inside the browser. Therefore, with the use of virtual DOM, the efficiency of the app improves.
* Gentle learning curve: React has a gentle learning curve when compared to frameworks like Angular. Anyone with little knowledge of javascript can start building web applications using React.
* SEO friendly: React allows developers to develop engaging user interfaces that can be easily navigated in various search engines. It also allows server-side rendering, which boosts the SEO of an app.
* Reusable components: React uses component-based architecture for developing applications. Components are independent and reusable bits of code. These components can be shared across various applications having similar functionality. The re-use of components increases the pace of development.
* Huge ecosystem of libraries to choose from: React provides you with the freedom to choose the tools, libraries, and architecture for developing an application based on your requirement.

### 3. What are the limitations of React?

The few limitations of React are as given below:

* React is not a full-blown framework as it is only a library.
* The components of React are numerous and will take time to fully grasp the benefits of all.
* It might be difficult for beginner programmers to understand React.
* Coding might become complex as it will make use of inline templating and JSX.

### 4. What is useState() in React?

The useState() is a built-in React Hook that allows you for having state variables in functional components. It should be used when the DOM has something that is dynamically manipulating/controlling.

In the below-given example code, The useState(0) will return a tuple where the count is the first parameter that represents the counter’s current state and the second parameter setCounter method will allow us to update the state of the counter.

const [count, setCounter] = useState(0);

const [otherStuffs, setOtherStuffs] = useState(...);

...

const setCount = () => {

setCounter(count + 1);

setOtherStuffs(...);

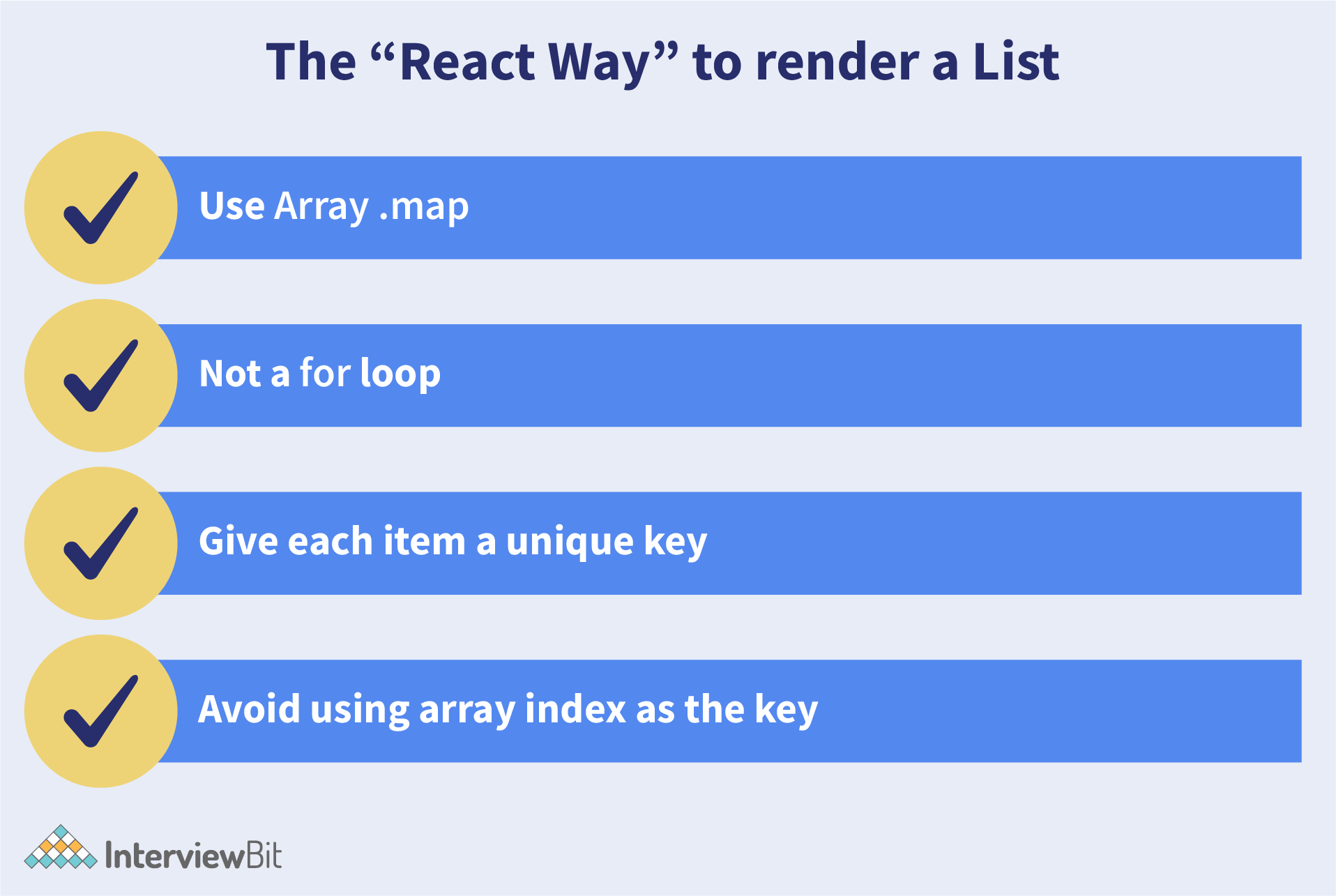
...

};

We can make use of setCounter() method for updating the state of count anywhere. In this example, we are using setCounter() inside the setCount function where various other things can also be done. The idea with the usage of hooks is that we will be able to keep our code more functional and avoid class-based components if they are not required.

### 5. What are keys in React?

A key is a special string attribute that needs to be included when using lists of elements.



Example of a list using key -

const ids = [1,2,3,4,5];

const listElements = ids.map((id)=>{

return(<li key={id.toString()}> {id}</li>)

})

Importance of keys -

* Keys help react identify which elements were added, changed or removed.
* Keys should be given to array elements for providing a unique identity for each element.
* Without keys, React does not understand the order or uniqueness of each element.
* With keys, React has an idea of which particular element was deleted, edited, and added.
* Keys are generally used for displaying a list of data coming from an API.

\*\*\*Note- Keys used within arrays should be unique among siblings. They need not be globally unique.

### 6. What is JSX?

JSX stands for JavaScript XML. It allows us to write HTML inside JavaScript and place them in the DOM without using functions like appendChild( ) or createElement( ).

As stated in the official docs of React, JSX provides syntactic sugar for React.createElement( ) function.

Note- We can create react applications without using JSX as well.

Let’s understand how JSX works:

Without using JSX, we would have to create an element by the following process:

const text = React.createElement('p', {}, 'This is a text');

const container = React.createElement('div','{}',text );

ReactDOM.render(container,rootElement);

Using JSX, the above code can be simplified:

const container = (<div> <p>This is a text</p></div>);

ReactDOM.render(container,rootElement);

As one can see in the code above, we are directly using HTML inside JavaScript.

### 7. What are the differences between functional and class components?

Before the introduction of Hooks in React, functional components were called stateless components and were behind class components on a feature basis. After the introduction of Hooks, functional components are equivalent to class components.

Although functional components are the new trend, the react team insists on keeping class components in React. Therefore, it is important to know how these components differ.

On the following basis let’s compare functional and class components:

* Declaration

Functional components are nothing but JavaScript functions and therefore can be declared using an arrow function or the function keyword:

function card(props){

return( <div className="main-container"> <h2>Title of the card</h2> </div> )

}

const card = (props) =>{

return( <div className="main-container"> <h2>Title of the card</h2> </div> )

}

Class components, on the other hand, are declared using the ES6 class:

class Card extends React.Component{

constructor(props){

super(props);

}

render(){

return( <div className="main-container"> <h2>Title of the card</h2> </div> )

}

}

* Handling props

Let’s render the following component with props and analyze how functional and class components handle props:

<Student Info name="Vivek" rollNumber="23" />

In functional components, the handling of props is pretty straightforward. Any prop provided as an argument to a functional component can be directly used inside HTML elements:

function StudentInfo(props){

return(

<div className="main">

<h2>{props.name}</h2>

<h4>{props.rollNumber}</h4>

</div>

)

}

In the case of class components, props are handled in a different way:

class StudentInfo extends React.Component{

constructor(props){

super(props);

}

render(){

return(

<div className="main">

<h2>{this.props.name}</h2>

<h4>{this.props.rollNumber}</h4>

</div>

)

}

}

As we can see in the code above, this keyword is used in the case of class components.

* Handling state

Functional components use React hooks to handle state. It uses the useState hook to set the state of a variable inside the component:

function ClassRoom(props){

let [studentsCount,setStudentsCount] = useState(0);

const addStudent = () => {

setStudentsCount(++studentsCount);

}

return(

<div>

<p>Number of students in class room: {studentsCount}</p>

<button onClick={addStudent}>Add Student</button>

</div>

)

}

Since useState hook returns an array of two items, the first item contains the current state, and the second item is a function used to update the state.

In the code above, using array destructuring we have set the variable name to studentsCount with a current value of “0” and setStudentsCount is the function that is used to update the state.

For reading the state, we can see from the code above, the variable name can be directly used to read the current state of the variable. We cannot use React Hooks inside class components, therefore state handling is done very differently in a class component:

Let’s take the same above example and convert it into a class component:

class ClassRoom extends React.Component{

constructor(props){

super(props);

this.state = {studentsCount : 0};

this.addStudent = this.addStudent.bind(this);

}

addStudent(){

this.setState((prevState)=>{

return {studentsCount: prevState.studentsCount++}

});

}

render(){

return(

<div>

<p>Number of students in class room: {this.state.studentsCount}</p>

<button onClick={this.addStudent}>Add Student</button>

</div>

)

}

}

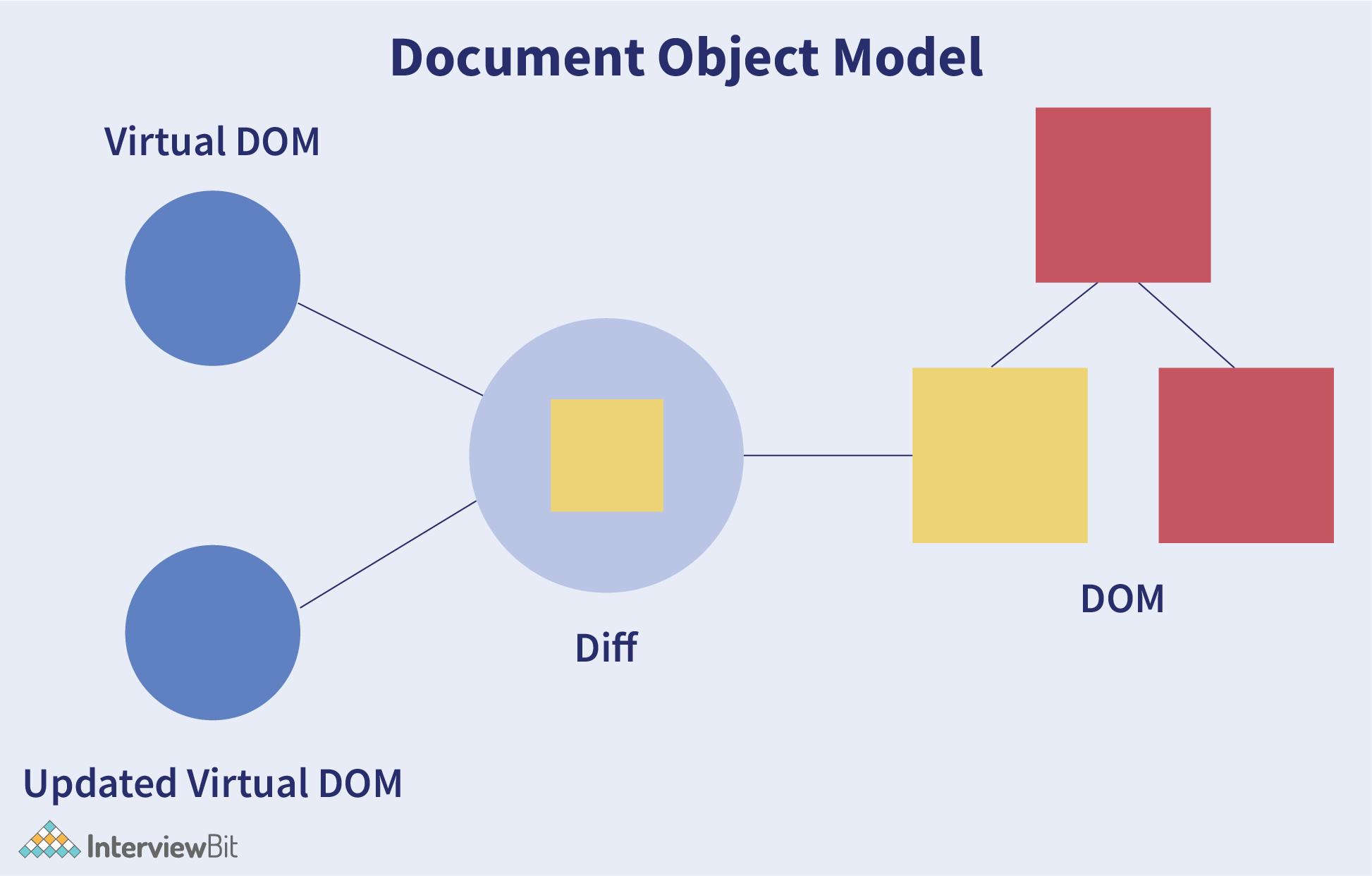
In the code above, we see we are using this.state to add the variable studentsCount and setting the value to “0”.

For reading the state, we are using this.state.studentsCount.

For updating the state, we need to first bind the addStudent function to this. Only then, we will be able to use the setState function which is used to update the state.

### 8. What is the virtual DOM? How does react use the virtual DOM to render the UI?

As stated by the react team, virtual DOM is a concept where a virtual representation of the real DOM is kept inside the memory and is synced with the real DOM by a library such as ReactDOM.



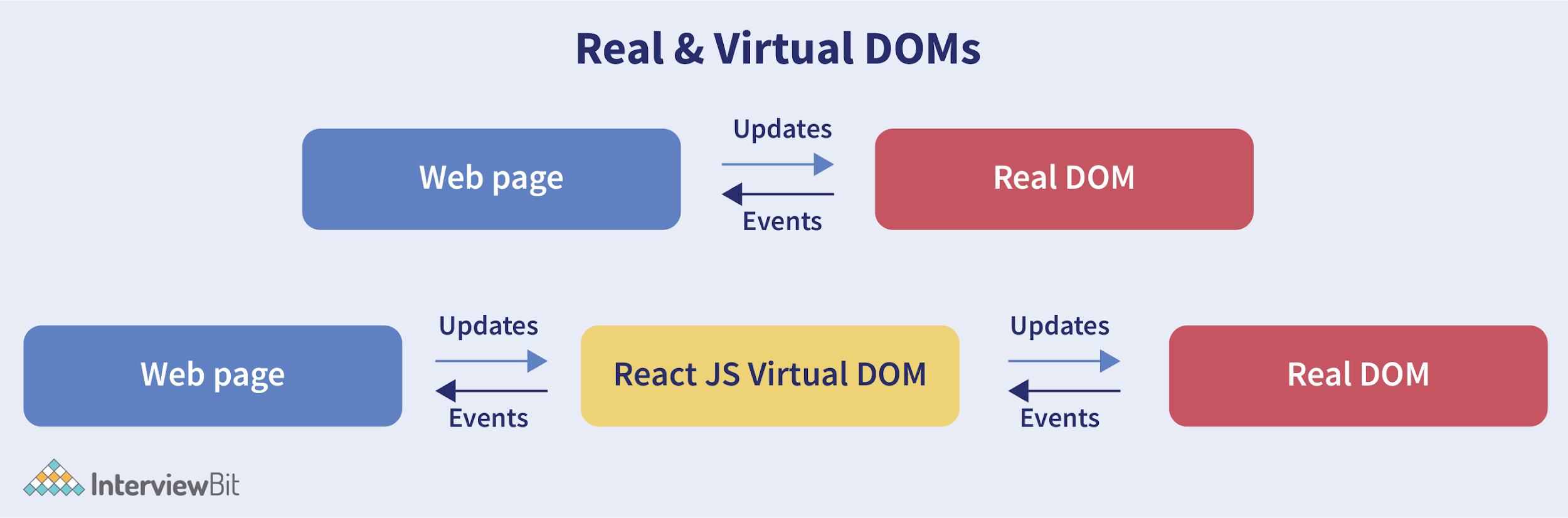
Why was virtual DOM introduced?

DOM manipulation is an integral part of any web application, but DOM manipulation is quite slow when compared to other operations in JavaScript. The efficiency of the application gets affected when several DOM manipulations are being done. Most JavaScript frameworks update the entire DOM even when a small part of the DOM changes.

For example, consider a list that is being rendered inside the DOM. If one of the items in the list changes, the entire list gets rendered again instead of just rendering the item that was changed/updated. This is called inefficient updating.

To address the problem of inefficient updating, the react team introduced the concept of virtual DOM.

How does it work?



For every DOM object, there is a corresponding virtual DOM object(copy), which has the same properties. The main difference between the real DOM object and the virtual DOM object is that any changes in the virtual DOM object will not reflect on the screen directly. Consider a virtual DOM object as a blueprint of the real DOM object. Whenever a JSX element gets rendered, every virtual DOM object gets updated.

\*\*Note- One may think updating every virtual DOM object might be inefficient, but that’s not the case. Updating the virtual DOM is much faster than updating the real DOM since we are just updating the blueprint of the real DOM.

React uses two virtual DOMs to render the user interface. One of them is used to store the current state of the objects and the other to store the previous state of the objects. Whenever the virtual DOM gets updated, react compares the two virtual DOMs and gets to know about which virtual DOM objects were updated. After knowing which objects were updated, react renders only those objects inside the real DOM instead of rendering the complete real DOM. This way, with the use of virtual DOM, react solves the problem of inefficient updating.

### 9. What are the differences between controlled and uncontrolled components?

Controlled and uncontrolled components are just different approaches to handling input from elements in react.

| Feature | Uncontrolled | Controlled | Name attrs |
| --- | --- | --- | --- |
| One-time value retrieval (e.g. on submit) | ✔️ | ✔️ | ✔️ |
| Validating on submit | ✔️ | ✔️ | ✔️ |
| Field-level Validation | ❌ | ✔️ | ✔️ |
| Conditionally disabling submit button | ❌ | ✔️ | ✔️ |
| Enforcing input format | ❌ | ✔️ | ✔️ |
| several inputs for one piece of data | ❌ | ✔️ | ✔️ |
| dynamic inputs | ❌ | ✔️ | 🤔 |

* Controlled component: In a controlled component, the value of the input element is controlled by React. We store the state of the input element inside the code, and by using event-based callbacks, any changes made to the input element will be reflected in the code as well.

When a user enters data inside the input element of a controlled component, onChange function gets triggered and inside the code, we check whether the value entered is valid or invalid. If the value is valid, we change the state and re-render the input element with the new value.

Example of a controlled component:

function FormValidation(props) {

let [inputValue, setInputValue] = useState("");

let updateInput = e => {

setInputValue(e.target.value);

};

return (

<div>

<form>

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

</form>

</div>

);

}

As one can see in the code above, the value of the input element is determined by the state of the inputValue variable. Any changes made to the input element is handled by the updateInput function.

* Uncontrolled component: In an uncontrolled component, the value of the input element is handled by the DOM itself. Input elements inside uncontrolled components work just like normal HTML input form elements.

The state of the input element is handled by the DOM. Whenever the value of the input element is changed, event-based callbacks are not called. Basically, react does not perform any action when there are changes made to the input element.

Whenever use enters data inside the input field, the updated data is shown directly. To access the value of the input element, we can use ref.

Example of an uncontrolled component:

function FormValidation(props) {

let inputValue = React.createRef();

let handleSubmit = e => {

alert(`Input value: ${inputValue.current.value}`);

e.preventDefault();

};

return (

<div>

<form onSubmit={handleSubmit}>

<input type="text" ref={inputValue} />

<button type="submit">Submit</button>

</form>

</div>

);

}

As one can see in the code above, we are not using onChange function to govern the changes made to the input element. Instead, we are using ref to access the value of the input element.

### 10. What are props in React?

The props in React are the inputs to a component of React. They can be single-valued or objects having a set of values that will be passed to components of React during creation by using a naming convention that almost looks similar to HTML-tag attributes. We can say that props are the data passed from a parent component into a child component.

The main purpose of props is to provide different component functionalities such as:

* Passing custom data to the React component.
* Using through this.props.reactProp inside render() method of the component.
* Triggering state changes.

For example, consider we are creating an element with reactProp property as given below:

<Element reactProp = "1" />  
This reactProp name will be considered as a property attached to the native props object of React which already exists on each component created with the help of React library: props.reactProp;.

### 11. Explain React state and props.

| Props | State |
| --- | --- |
| Immutable | Owned by its component |
| Has better performance | Locally scoped |
| Can be passed to child components | Writeable/Mutable |
|  | has setState() method to modify properties |
|  | Changes to state can be asynchronous |
|  | can only be passed as props |

* React State  
  Every component in react has a built-in state object, which contains all the property values that belong to that component.  
  In other words, the state object controls the behaviour of a component. Any change in the property values of the state object leads to the re-rendering of the component.

Note- State object is not available in functional components but, we can use React Hooks to add state to a functional component.

How to declare a state object?

*Example:*

class Car extends React.Component{

constructor(props){

super(props);

this.state = { brand: "BMW", color: "black" }

}

}

How to use and update the state object?

class Car extends React.Component {

constructor(props) {

super(props);

this.state = { brand: "BMW", color: "Black" };

}

changeColor() {

this.setState(prevState => {

return { color: "Red" };

});

}

render() {

return (

<div>

<button onClick={() => this.changeColor()}>Change Color</button>

<p>{this.state.color}</p>

</div>

);

}

}

As one can see in the code above, we can use the state by calling this.state.propertyName and we can change the state object property using setState method.

* React Props

Every React component accepts a single object argument called props (which stands for “properties”). These props can be passed to a component using HTML attributes and the component accepts these props as an argument.

Using props, we can pass data from one component to another.

*Passing props to a component:*

While rendering a component, we can pass the props as an HTML attribute:

<Car brand="Mercedes"/>

The component receives the props:

*In Class component:*

class Car extends React.Component {

constructor(props) {

super(props);

this.state = { brand: this.props.brand, color: "Black" };

}

}

*In Functional component:*

function Car(props) {

let [brand, setBrand] = useState(props.brand);

}

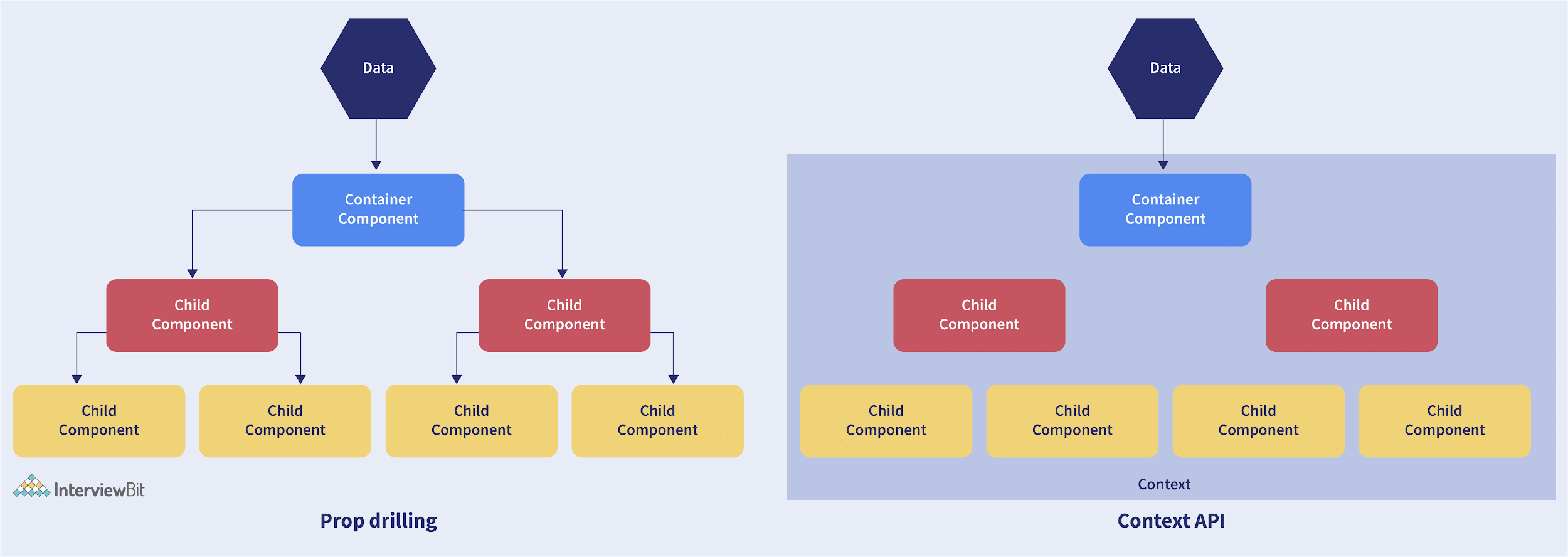
Note- Props are read-only. They cannot be manipulated or changed inside a component.

### 12. Explain about types of side effects in React component.

There are two types of side effects in React component. They are:

* Effects without Cleanup: This side effect will be used in useEffect which does not restrict the browser from screen update. It also improves the responsiveness of an application. A few common examples are network requests, Logging, manual DOM mutations, etc.
* Effects with Cleanup: Some of the Hook effects will require the cleanup after updating of DOM is done. For example, if you want to set up an external data source subscription, it requires cleaning up the memory else there might be a problem of memory leak. It is a known fact that React will carry out the cleanup of memory when the unmounting of components happens. But the effects will run for each render() method rather than for any specific method. Thus we can say that, before execution of the effects succeeding time the React will also cleanup effects from the preceding render.

### 13. What is prop drilling in React?



Sometimes while developing React applications, there is a need to pass data from a component that is higher in the hierarchy to a component that is deeply nested. To pass data between such components, we pass props from a source component and keep passing the prop to the next component in the hierarchy till we reach the deeply nested component.

The disadvantage of using prop drilling is that the components that should otherwise be not aware of the data have access to the data.

### 14. What are error boundaries?

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

* What is an error boundary?

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

1. Render phase
2. Inside a lifecycle method
3. Inside the constructor

Without using error boundaries:

class CounterComponent extends React.Component{

constructor(props){

super(props);

this.state = { counterValue: 0 }

this.incrementCounter = this.incrementCounter.bind(this);

}

incrementCounter(){

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

}

render(){

if(this.state.counter === 2){

throw new Error('Crashed');

}

return(

<div>

<button onClick={this.incrementCounter}>Increment Value</button>

<p>Value of counter: {this.state.counterValue}</p>

</div>

)

}

}

In the code above, when the counterValue equals 2, we throw an error inside the render method.

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

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

Let’s create an error boundary to handle errors in the render phase:

class ErrorBoundary extends React.Component {

constructor(props) {

super(props);

this.state = { hasError: false };

}

static getDerivedStateFromError(error) {

return { hasError: true };

}

componentDidCatch(error, errorInfo) {

logErrorToMyService(error, errorInfo);

}

render() {

if (this.state.hasError) {

return <h4>Something went wrong</h4>

}

return this.props.children;

}

}

In the code above, getDerivedStateFromError function renders the fallback UI interface when the render method has an error.

componentDidCatch logs the error information to an error tracking service.

Now with the error boundary, we can render the CounterComponent in the following way:

<ErrorBoundary>

<CounterComponent/>

</ErrorBoundary>

### 15. What is React Hooks?

React Hooks are the built-in functions that permit developers for using the state and lifecycle methods within React components. These are newly added features made available in React 16.8 version. Each lifecycle of a component is having 3 phases which include mount, unmount, and update. Along with that, components have properties and states. Hooks will allow using these methods by developers for improving the reuse of code with higher flexibility navigating the component tree.

Using Hook, all features of React can be used without writing class components. *For example*, before React version 16.8, it required a class component for managing the state of a component. But now using the useState hook, we can keep the state in a functional component.

### 16. Explain React Hooks.

What are Hooks? Hooks are functions that let us “hook into” React state and lifecycle features from a functional component.

React Hooks cannot be used in class components. They let us write components without class.

Why were Hooks introduced in React?

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

*Example of a hook:* useState hook:

In functional components, the useState hook lets us define a state for a component:

function Person(props) {

// We are declaring a state variable called name.

// setName is a function to update/change the value of name

let [name, setName] = useState('');

}

The state variable “name” can be directly used inside the HTML.

### 17. What are the rules that must be followed while using React Hooks?

There are 2 rules which must be followed while you code with Hooks:

* React Hooks must be called only at the top level. It is not allowed to call them inside the nested functions, loops, or conditions.
* It is allowed to call the Hooks only from the React Function Components.

### 18. What is the use of useEffect React Hooks?

The useEffect React Hook is used for performing the side effects in functional components. With the help of useEffect, you will inform React that your component requires something to be done after rendering the component or after a state change. The function you have passed(can be referred to as “effect”) will be remembered by React and call afterwards the performance of DOM updates is over. Using this, we can perform various calculations such as data fetching, setting up document title, manipulating DOM directly, etc, that don’t target the output value. The useEffect hook will run by default after the first render and also after each update of the component. React will guarantee that the DOM will be updated by the time when the effect has run by it.

The useEffect React Hook will accept 2 arguments: useEffect(callback,[dependencies]);

Where the first argument callback represents the function having the logic of side-effect and it will be immediately executed after changes were being pushed to DOM. The second argument dependencies represent an optional array of dependencies. The useEffect() will execute the callback only if there is a change in dependencies in between renderings.

Example:

import { useEffect } from 'react';

function WelcomeGreetings({ name }) {

const msg = `Hi, ${name}!`; // Calculates output

useEffect(() => {

document.title = `Welcome to you ${name}`; // Side-effect!

}, [name]);

return <div>{msg}</div>; // Calculates output

}

The above code will update the document title which is considered to be a side-effect as it will not calculate the component output directly. That is why updating of document title has been placed in a callback and provided to useEffect().

Consider you don’t want to execute document title update each time on rendering of WelcomeGreetings component and you want it to be executed only when the name prop changes then you need to supply name as a dependency to useEffect(callback, [name]).

### 19. Why do React Hooks make use of refs?

Earlier, refs were only limited to class components but now it can also be accessible in function components through the useRef Hook in React.

The refs are used for:

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

### 20. What are Custom Hooks?

A Custom Hook is a function in Javascript whose name begins with ‘use’ and which calls other hooks. It is a part of React v16.8 hook update and permits you for reusing the stateful logic without any need for component hierarchy restructuring.

In almost all of the cases, custom hooks are considered to be sufficient for replacing render props and HoCs (Higher-Order components) and reducing the amount of nesting required. Custom Hooks will allow you for avoiding multiple layers of abstraction or wrapper hell that might come along with Render Props and HoCs.

The disadvantage of Custom Hooks is it cannot be used inside of the classes.

================

1. What is ReactJS?

react\_-\_a\_javascript\_library\_for\_building\_user\_interfaces-1 35+ Basic & Advanced ReactJS Interview Questions

A: This is one of the first ReactJS interview questions. You need to be able to explain that React is a JavaScript-based UI library. It was developed at Facebook with the purpose of creating interactive and reusable UI components.

React is currently one of the most popular JavaScript frameworks used for handling the presentation layer for web and mobile apps.

2. Name some advantages of React.

A: One of the basic React.js interview questions. You should be able to name some of React’s biggest advantages such as:

High efficiency: React is known for its flexibility and efficiency because of its own virtual DOM that it creates. It automatically calculates the changes needed to be made in DOM thus avoiding expensive DOM options and making regular updates whenever needed.

Easier JavaScript writing: Thanks to the JSX syntax which allows you to mix HTML with JavaScript, writing is much easier with React. The user can simply drop a bit of HTML in the render function without having to concatenate strings and React then turns those HTML bits into functions with its JSXTransformer.

A variety of developer tools: in order to get the most out of React.js, you should install its official browser extension. It comes in quite handy, especially for debugging the app. It gives you a direct look into the virtual DOM as if you were browsing a regular DOM tree in the elements panel.

Great for SEO: Most of the JavaScript frameworks aren’t particularly search engine friendly, but that is not the case with React. Thanks to its unique virtual DOM that is rendered to the browser as a regular page, the search engines will not encounter the same issues as with reading other JavaScript-heavy apps.

UI Test Cases: It is very easy to write UI test cases thanks to the virtual DOM in JavaScript.

3. What is JSX?

introducing\_jsx\_-\_react-1 35+ Basic & Advanced ReactJS Interview Questions

A: JSX is an XML/HTML-like syntax used by React for extending ECMAScript in order for the XML/HTML-like text to coexist 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.

4. What are the disadvantages of React.js?

A: When talking about React questions, one should be well aware of the biggest disadvantages of this framework. Let’s take a look at some you should name.

Disadvantages:

Integration with traditional MVC framework requires major configurations

The steep learning curve for the beginners

What that means in practice is that you can write concise HTML/XML-like structures in the same file as you write JavaScript code. These structures will then be automatically transformed into JavaScript code.

5. What is the difference between createElement and cloneElement?

A: This is one of the quite logical React interview questions and answers. Basically, as the name suggests, createElement is what React uses to create React Elements, cloneElement, on the other hand, is used to clone an element and pass it new props. Might seem quite obvious, but it is still one of the top 50 interview questions and answers.

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

A: When preparing for your React js interview, you need to know that a callback function is invoked when setState has finished and the component is re-rendered. However, keep in mind that setState is asynchronous and that is the reason why it requires a second callback function. However, rather than relying solely on the callback function, one should use another lifecycle method.

7. What is the difference between state and props?

A: This is one of the ReactJS interview questions and answers for experienced as well as for beginners.

Long story short, 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, on the other hand, are a component’s configuration. Think of them as a way for the components to communicate with each other. They are received from the above component and are immutable as far as the component receiving them is concerned. A component isn’t able to change its props, but it has to put together the props of its child components. While the props are mostly data, other things can be passed as them as well, for example, callback functions.

Props and states do similar things but are used in different ways. The main difference is that props are immutable and cannot be changed while the state is used for mutable data or data that will change.

8. What is Redux?

read\_me\_-\_redux-1 35+ Basic & Advanced ReactJS Interview Questions

A: You need to know a thing or two about React-Redux when preparing for an interview. With that in mind, we have composed several Redux interview questions so let’s start with the most basic one.

The basic idea of Redux is that the entire application state is kept in a single store which is simply a JavaScript object. If you want to change the state, you need to fire actions from your application and then write reducers for these actions that modify the state. The entire state transition is kept inside reducers and isn’t supposed to have any side effects.

The general idea behind Redux is that there should be only a single source of truth for your application state, such as the UI state (which tab is active) or the data state (the user profile details).

All of the data is retained in the Redux store. The createStore function is used to create mentioned stores. This function accepts another function as an argument. The passed in function (also known as the reducer) is responsible for returning the state of the application at a certain point in time which is then persisted in the store.

9. What is the difference between React Native and React?

A: This is another one of the common React interview questions. It all starts with the fact that React is a JavaScript library and it supports both front-end web and being run on the server and it is commonly used for building user interfaces and web applications.

React Native, however, is a mobile framework that compiles to native app components. It allows us to build native mobile applications for Windows, Android, and iOS in JavaScript while we can use ReactJS to build our components.

With React Native, we can mimic the behavior of the native app in JavaScript and get a platform-specific code as the output. It is also possible to mix the native code with the JavaScript if we need to optimize our application further.

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

A: We have mentioned the single source of truth many time earlier in this article but would you know how to explain what it actually means? Well, you should because this is one of the React interview questions that are pretty basic.

The single source of truth refers to the store used for storing the app’s entire state at one place. The benefits of the single source of truth include the facts that all the components stored there receive updates from the store itself, it is easier to keep track of their changes, as well as debug and inspect the application.

11. What is React Router?

A: React Router is a powerful routing library built on top of React. It is highly useful for adding new screens and flows to the application while keeping the URL in sync with data displayed on the web page. The React Router has a simple AP and it maintains a standardized behavior and structure for developing single page web apps.

12. Why is switch keyword used in React Router v4?

A: <div> is used to encapsulate multiple routes inside the Router. The switch keyword, however, is used when you want to display only a single route to be rendered amongst the several defined routes. The <switch> tag matches the typed URL with the defined routes in sequential order. When the first match is found, it renders the specified route bypassing all the other routes.

13. Why do we need a Router in React?

A: A Router is particularly useful for creating and defining multiple routes. When you type a specific URL that matches a path of any route defined inside the router, you will be redirected to that particular route. Basically, you will bypass all the other routes and get to the one you need.

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

A: 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.

15. What is render() in React? Explain its purpose.

A: We should start by explaining that each React component must have a render() because 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 and return the same result each time it is invoked.

Intermediate ReactJS Interview Questions

16. What is Flux in JavaScript?

A: This is another one of the common interview questions on React js. You should know that Flux is an application architecture for creating data layers in JavaScript apps. It is neither a framework nor a library, but a type of architecture that complements the concept of Unidirectional Data Flow.

17. What are refs in React? When to use them?

A: Refs are used for storing the reference of element or component returned by the component render() configuration function. In most cases, they should be avoided although there are some instances when using them is useful, such as:

Managing focus, text selection, or media playback.

Triggering imperative animations.

Integrating with third-party DOM libraries.

18. What is the difference between DOM and virtual DOM in React.js?

A: 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

19. What are the controlled components and uncontrolled components in React?

A: When answering React interview questions, you should know that the form data is handled by the React components.

A controlled input accepts values as props and callbacks to change that value. The uncontrolled component, on the other hand, is a substitute for controlled components. In these cases, DOM itself handles the form data.

20. What’s the difference between an Element and a Component in React?

A: React element is an object representation of some UI. Basically, it describes what you want to see on the screen. A React component, on the other hand, is a function or a class that optionally accepts input and returns a React element. This is also one of the common interview questions on ReactJS.

21. Explain the difference between functional and class components.

A: The components that return React elements as a result are called functional components. They are basically just simple JavaScript functions. They, however, haven’t been around for long. In fact, they have been introduced with React 0.14.

Class components, on the other hand, have been around for quite some time. They use plain Java objects for creating pages. With the React’s create-a-class-factory method, a literal is passed in defining the methods of a new component.

22. How is React different from Angular and VUE?

A: 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.

In the end, they all have something to offer and being tribal about it is pointless and self-destructive.

23. What is the use of the arrow function in React?

A: The arrow function is something you should be familiar when preparing for React interview questions because this function is very important for React operations. It allows you to predict the behavior of bugs when passed as callbacks hence it prevents bugs caused by this altogether.

[activecampaign form=25]

24. List some of the cases when you should use Refs.

A: Refs should be used in the following cases:

When you need to manage focus, select text or media playback

To trigger imperative animations

Integrate with third-party DOM libraries

25. What are Pure Components?

A: Pure components are the simplest and fastest components which can replace any component with only a render(). Pure components enhance the simplicity of the code and performance of the application. Consider learning more about them when preparing for React native interview questions.

26. What is the significance of keys in React?

A: When you want to identify the unique virtual DOM elements with their corresponding data driving the UI – that is where the keys come in. By recycling all the existing elements in the DOM, the keys help React optimize the rendering. They allow React to reorder the elements instead of re-rendering them, which increases the app’s performance. Definitely add the keys on your list of ReactJS interview questions and answers.

27. How do you tell React to build in Production mode and what will it do?

A: Typically, you would use the Webpack’s DefinePlugin method to set NODE\_ENV to production. This would strip out things like propType validation and extra warnings. It would be smart to minify your code as well because React uses Uglify’s dead-code elimination to strip out development only code and comments, which will significantly reduce the size of your bundle.

28. Describe how events are handled in React.

A: 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.

29. What are the advantages of Redux?

A: Speaking of React Redux interview questions, one should definitely be able to name its most important advantages. This is one of the Redux interview questions and answers for experienced as well as for the beginners.

So let’s take a look at some of the advantages of ReactJS with Redux:

Predictability of outcome – With only one source of truth, i.e. the store, there is no confusion about how to sync the current state with actions and other parts of the application.

Maintainability – The code becomes easier to maintain with a predictable outcome and strict structure.

Server-side rendering – You just need to pass the store created on the server to the client side. Not only is this good for the initial render but it also provides a better user experience as it optimizes the app performance.

Developer tools – From actions to state changes, developers can track everything going on in the application in real time.

Community and ecosystem – Anyone who has been new to something at some point in life (haven’t we all?) knows the importance of a solid community and its support. Luckily, Redux has a huge community of talented individuals that constantly contribute to the betterment of the library and develop various apps with it.

Ease of testing – Redux’s code is easily testable and independent, mostly because the functions are small, pure, and isolated.

Organization – Redux is precise about how code should be organized which makes the code more consistent and easier when a team works on it.

30. Is setState() async? Why?

A: 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, the setState calls are asynchronous as well as batched for better UI experience and performance. Keep this in mind as this is definitely among the most popular 50 interview questions and answers when it comes to React.

31. Explain the components of Redux.

A: This is another common Redux React question. You should know that Redux is composed of the following components:

Action — The action is the only source of information that sends data from our application to our store. Actions are sent the store using store.dispatch().

Reducer — Reducers specify how the app’s state changes in response to actions sent to the store. Since the actions don’t show the application’s state changes, this place determines how the 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 the 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).

That being said, keep in mind that there is only a single store in a Redux application. When we want to split the data handling logic, we need to use the reducer composition instead of many stores.

32. What is Context?

Context enables passing data through the component tree without having to pass props down manually at every level. Which means you can deep-nest items without issue.

It is primarily used when some data needs to be accessible by many components at different nesting levels.

Advanced ReactJS Interview Questions

33. Explain mixin or higher order components (HOC) in React.

A: HOC in React are functions that take and return components. Basically, it reuses component logic. That being said, you should also know that HOC isn’t exactly a part of React API. They are functions that loop over and apply functions to every element in an array. Keep that in mind when answering React js questions.

34. What are synthetic events in React?

A: When talking about React js interview questions and answers, we should mention the synthetic events. They act as a cross-browser wrapper around the browser’s native event thus combining the behavior of different browsers into one API. Their purpose is to ensure that the events show consistent properties across different browsers.

35. What can you do with HOC?

A: You can use HOC for:

Code reuse, logic and bootstrap abstraction

Render hijacking

State abstraction and manipulation

Props manipulation

36. What is Redux Thunk used for?

A: Speaking of ReactJS Redux, you should also be familiar with Redux Thunk used to write action creators that return a function instead of an action. In addition to that, it can be used to delay the dispatch of an action if a certain condition is met. This allows us to handle the asynchronous dispatching of actions. The inner function receives the store methods dispatch and getState as parameters.

37. Can you discuss your experience with React chart libraries and how you have utilized them in your previous projects?

A: In my previous projects as a React developer, I have extensively used React chart libraries like Chart.js, Recharts, and Victory. These libraries have allowed me to create dynamic and visually appealing charts, including line, bar, and pie charts.

I have also implemented features such as tooltips, zooming, and responsive design to enhance the user experience. Overall, my experience with React chart libraries has enabled me to deliver data-driven solutions efficiently.

==========================================

1. How do you create refs in React?
2. How often does the React *useState* update? Why?
3. Name three advantages of using React Hooks.
4. Name two advantages of using React.js.
5. Outline the different stages of the React.js lifecycle.
6. Explain why class methods should be bound to class instances.
7. What happens if you attempt to update the state directly?
8. Explain whether Hooks replace higher-order components.
9. Name one advantage of using Flux.

## 5 crucial common advanced React.js interview questions and answers to look for

Here are five key common advanced React.js interview questions from the section above and the answers you should look for from your candidates.

### 1. How do you create refs in React?

Can your candidates explain that they must use *React.createRef()* to create refs in React.js? They should also understand that the ref is often assigned to an instance property when constructing a component.

### 2. Name two advantages of using React.js.

Candidates may mention several advantages of using React.js when responding to this question. For example, they may explain that the library helps them build high-quality user interfaces or that it permits them to write custom components.

### 3. Name one advantage of using Flux.

Your candidates may start by explaining that Flux is a JavaScript architecture that operates on a unidirectional data flow. But do they know Flux offers many advantages for projects with dynamic data? For example, can they explain that Flux helps ensure dynamic data is updated effectively?

### 4. Name three advantages of using React Hooks.

Three advantages of using React Hooks are that they enable developers to:

* Enhance component tree readability
* Share logic among different components
* Effectively handle the setup of side effects

### 5. How often does the React *useState* update? Why?

Since developers use *useState* to enhance performance by creating queues, React doesn’t update changes immediately. Candidates should know that *useState* doesn’t implement changes to the *state* object directly; instead, the updates occur asynchronously.

## 8 advanced behavioral React.js interview questions

This section includes eight advanced behavioral React.js interview questions you can ask to find out which methods your candidates use when working on React.js projects.



1. Which method would you use to handle events in React?
2. In which situation would you use refs in React?
3. Why would you use super constructors with props arguments?
4. How would you use validation on props?
5. Which method would you use to add attributes to components conditionally?
6. What methods would you use to check and improve slow app rendering in React?
7. In which situation would you use *useMemo()* in React?
8. How would you avoid binding in React?

## 5 crucial advanced behavioral React.js interview questions and answers to look for

Here are five of the most important advanced behavioral React.js interview questions from the section above and the answers you should look for from your candidates.

### 1. In which situation would you use refs in React?

Advanced candidates should understand that they can use React refs to access a DOM element. They may also explain that they would use refs to access an element they have created to change a child component’s value.

### 2. In which situation would you use *useMemo()* in React?

Do your candidates know that developers can use *useMemo()* to cache a variable’s value along with dependency lists? Can they explain that they would use *useMemo()* to help them avoid unnecessary re-renders? They may also say that *useMemo()* can be useful in situations where there are high processing amounts.

### 3. Why would you use super constructors with props arguments?

Candidates may explain that they pass props to super constructors to access and use *this.props* in the constructor. They may mention that when they implement a *constructor()* function within a React component, they use *super()* to call the parent constructor.

### 4. How would you avoid binding in React?

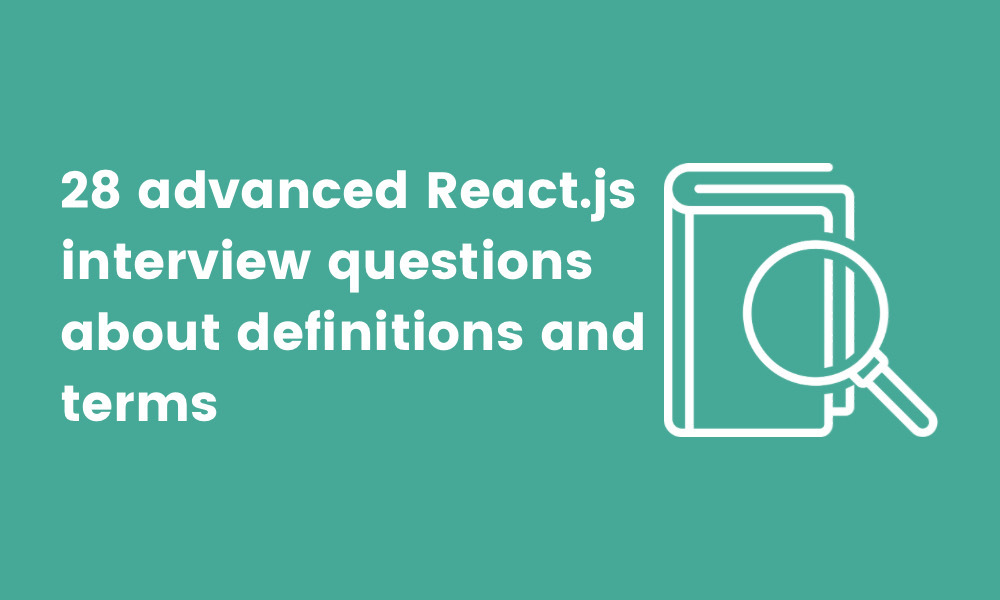
Candidates who have advanced React skills should be aware that they can use arrow functions in class properties to avoid binding in React. They may mention that class properties are a new feature and, to use them, a developer must enable transform-class-properties.

### 5. Which method would you use to handle events in React?

Can your candidates explain that to handle events in React, they would name them using camelCase (instead of lowercase)? They should be able to also explain that when they use JSX, they pass a function as the event handler (instead of a string).

## 28 advanced React.js interview questions about definitions and terms

Ask your candidates these advanced React.js interview questions about definitions and terms to discover if they have the knowledge to use the library efficiently.



1. Explain what a higher-order component is.
2. Explain what a mounted component is.
3. Explain what *useState* is.
4. Explain what an event is in React.
5. What is a class component?
6. What is a component?
7. What is the difference between class and functional components?
8. What is a *state* object?
9. What is a *props* object?
10. How are *state* objects different from *props* objects?
11. Explain what MVC architecture is.
12. Name an architectural difference between React and Angular.
13. Explain what a controlled component is.
14. Explain what an uncontrolled component is.
15. How are controlled and uncontrolled components different?
16. Explain what React Hooks are.
17. Explain what three dots mean in React.
18. What are package managers in React.js?
19. Explain what prop drilling is.
20. Explain what *StrictMode* is.
21. Explain what the Shadow DOM is.
22. Explain what the virtual DOM is.
23. How is the Shadow DOM different from the virtual DOM?
24. Explain what the React.js lifecycle methods are.
25. Explain what a pure function is.
26. Explain what JSX is.
27. Explain what Flux architecture is.
28. What are bundlers in React.js?

## 5 crucial advanced React.js interview questions and answers related to definitions and terms

Here are five vital advanced React.js interview questions about definitions and terms, along with answers you should listen for from candidates.

### 1. Explain what React Hooks are.

Hooks are a feature that was introduced in React 16.8. They enable you to use state and other React features without having to write a class. They’re called Hooks because they allow you to “hook” into React features. One example is the *useState* Hook, which allows you to add React state to function components.

### 2. What is a *state* object?

A *state* object is a plain JavaScript object that developers use in React to show information on a component’s current properties. Developers can manage the *state* object in the component. Changing the *state* object causes the component to re-render.

### 3. What is a *props* object?

A *props* object takes the form of an ordinary object. Candidates should be able to explain that *props* objects abide by immutable properties, meaning that a component cannot change its own *props* object.

### 4. What is a class component?

A class component is a simple class that consists of several functions. It accepts props as arguments and returns React elements. Developers must create render functions to use class components and receive React elements.

### 5. What is a component?

A component is a reusable piece of code that developers can use to return React elements that will be rendered to a page. Components can return other components, as well as [arrays](https://www.testgorilla.com/test-library/programming-skills-tests/python-coding-test-working-with-arrays/), numbers, and strings.

================

## React Interview Questions for Experienced

### 1. Explain Strict Mode in React.

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

function App() {

return (

<React.StrictMode>

<div classname="App">

<Header/> <div> Page Content </div> <Footer/>

</div>

</React.StrictMode>

);

}

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

import React from "react";

import ReactDOM from "react-dom";

import App from "./App";

const rootElement = document.getElementById("root");

ReactDOM.render(

<React.StrictMode> <App /></React.StrictMode>, rootElement

);

StrictMode currently helps with the following issues:

* Identifying components with unsafe lifecycle methods:
  + Certain lifecycle methods are unsafe to use in asynchronous react applications. With the use of third-party libraries, it becomes difficult to ensure that certain lifecycle methods are not used.
  + StrictMode helps in providing us with a warning if any of the class components use an unsafe lifecycle method.
* Warning about the usage of legacy string API:
  + If one is using an older version of React, callback ref is the recommended way to manage refs instead of using the string refs. StrictMode gives a warning if we are using string refs to manage refs.
* Warning about the usage of findDOMNode:
  + Previously, findDOMNode( ) method was used to search the tree of a DOM node. This method is deprecated in React. Hence, the StrictMode gives us a warning about the usage of this method.
* Warning about the usage of legacy context API (because the API is error-prone).

### 2. How to prevent re-renders in React?

* Reason for re-renders in React:
  + Re-rendering of a component and its child components occur when props or the state of the component has been changed.
  + Re-rendering components that are not updated, affects the performance of an application.
* How to prevent re-rendering:

Consider the following components:

class Parent extends React.Component {

state = { messageDisplayed: false };

componentDidMount() {

this.setState({ messageDisplayed: true });

}

render() {

console.log("Parent is getting rendered");

return ( <div className="App"> <Message /> </div> );

}

}

class Message extends React.Component {

constructor(props) {

super(props);

this.state = { message: "Hello, this is vivek" };

}

render() {

console.log("Message is getting rendered");

return ( <div> <p>{this.state.message}</p> </div> );

}

}

* The Parent component is the parent component and the Message is the child component. Any change in the parent component will lead to re-rendering of the child component as well. To prevent the re-rendering of child components, we use the shouldComponentUpdate( ) method:

\*\*Note- Use shouldComponentUpdate( ) method only when you are sure that it’s a static component.

class Message extends React.Component {

constructor(props) {

super(props);

this.state = { message: "Hello, this is vivek" };

}

shouldComponentUpdate() {

console.log("Does not get rendered");

return false;

}

render() {

console.log("Message is getting rendered");

return ( <div> <p>{this.state.message}</p> </div> );

}

}

As one can see in the code above, we have returned false from the shouldComponentUpdate( ) method, which prevents the child component from re-rendering.

### 3. What are the different ways to style a React component?

There are many different ways through which one can style a React component. Some of the ways are :

* Inline Styling: We can directly style an element using inline style attributes. Make sure the value of style is a JavaScript object:

class RandomComponent extends React.Component {

render() {

return (

<div>

<h3 style={{ color: "Yellow" }}>This is a heading</h3>

<p style={{ fontSize: "32px" }}>This is a paragraph</p>

</div>

);

}

}

* Using JavaScript object: We can create a separate JavaScript object and set the desired style properties. This object can be used as the value of the inline style attribute.

class RandomComponent extends React.Component {

paragraphStyles = { color: "Red", fontSize: "32px" };

headingStyles = { color: "blue", fontSize: "48px" };

render() {

return (

<div>

<h3 style={this.headingStyles}>This is a heading</h3>

<p style={this.paragraphStyles}>This is a paragraph</p>

</div>

);

}

}

* CSS Stylesheet: We can create a separate CSS file and write all the styles for the component inside that file. This file needs to be imported inside the component file.

import './RandomComponent.css';

class RandomComponent extends React.Component {

render() {

return (

<div>

<h3 className="heading">This is a heading</h3>

<p className="paragraph">This is a paragraph</p>

</div>

);

}

}

* CSS Modules: We can create a separate CSS module and import this module inside our component. Create a file with “.module.css”‘ extension, styles.module.css:

.paragraph{

color:"red";

border:1px solid black;

}

We can import this file inside the component and use it:

import styles from './styles.module.css';

class RandomComponent extends React.Component {

render() {

return (

<div>

<h3 className="heading">This is a heading</h3>

<p className={styles.paragraph} >This is a paragraph</p>

</div>

);

}

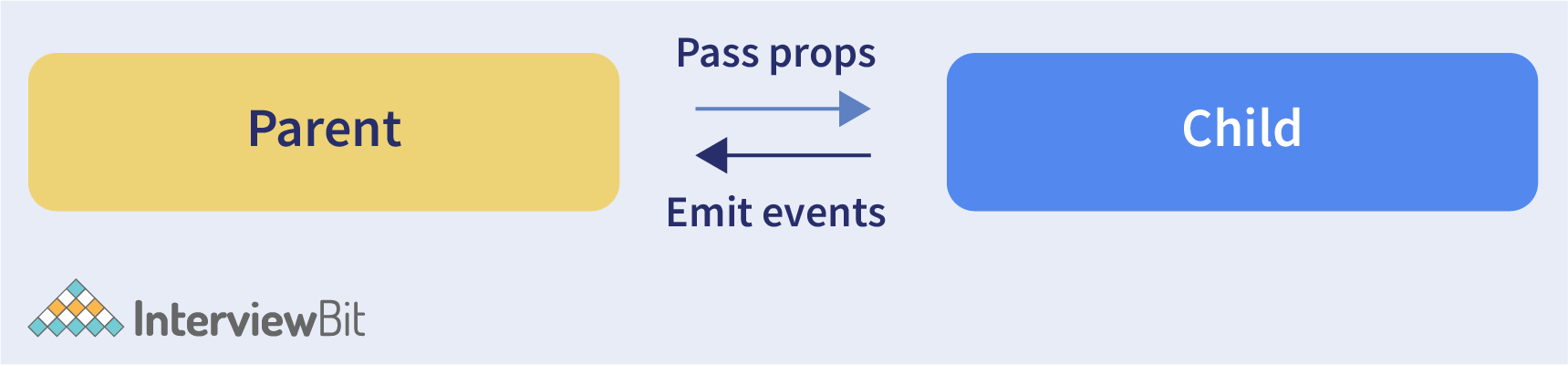
}

### 4. Name a few techniques to optimize React app performance.

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

* Using useMemo( ) -
  + It is a React hook that is used for caching CPU-Expensive functions.
  + Sometimes in a React app, a CPU-Expensive function gets called repeatedly due to re-renders of a component, which can lead to slow rendering.  
    useMemo( ) hook can be used to cache such functions. By using useMemo( ), the CPU-Expensive function gets called only when it is needed.
* Using React.PureComponent -
  + It is a base component class that checks the state and props of a component to know whether the component should be updated.
  + Instead of using the simple React.Component, we can use React.PureComponent to reduce the re-renders of a component unnecessarily.
* Maintaining State Colocation -
  + This is a process of moving the state as close to where you need it as possible.
  + Sometimes in React app, we have a lot of unnecessary states inside the parent component which makes the code less readable and harder to maintain. Not to forget, having many states inside a single component leads to unnecessary re-renders for the component.
  + It is better to shift states which are less valuable to the parent component, to a separate component.
* Lazy Loading -
  + It is a technique used to reduce the load time of a React app. Lazy loading helps reduce the risk of web app performances to a minimum.

### 5. How to pass data between react components?



Parent Component to Child Component (using props)

With the help of props, we can send data from a parent to a child component.

How do we do this?

Consider the following Parent Component:

import ChildComponent from "./Child";

function ParentComponent(props) {

let [counter, setCounter] = useState(0);

let increment = () => setCounter(++counter);

return (

<div>

<button onClick={increment}>Increment Counter</button>

<ChildComponent counterValue={counter} />

</div>

);

}

As one can see in the code above, we are rendering the child component inside the parent component, by providing a prop called counterValue. The value of the counter is being passed from the parent to the child component.

We can use the data passed by the parent component in the following way:

function ChildComponent(props) {

return (

<div> <p>Value of counter: {props.counterValue}</p> </div>

);

}

We use the props.counterValue to display the data passed on by the parent component.

Child Component to Parent Component (using callbacks)

This one is a bit tricky. We follow the steps below:

* Create a callback in the parent component which takes in the data needed as a parameter.
* Pass this callback as a prop to the child component.
* Send data from the child component using the callback.

We are considering the same example above but in this case, we are going to pass the updated counterValue from child to parent.

Step1 and Step2: Create a callback in the parent component, pass this callback as a prop.

function ParentComponent(props) {

let [counter, setCounter] = useState(0);

let callback = valueFromChild => setCounter(valueFromChild);

return (

<div>

<p>Value of counter: {counter}</p>

<ChildComponent callbackFunc={callback} counterValue={counter} />

</div>

);

}

As one can see in the code above, we created a function called callback which takes in the data received from the child component as a parameter.

Next, we passed the function callback as a prop to the child component.

Step3: Pass data from the child to the parent component.

function ChildComponent(props) {

let childCounterValue = props.counterValue;

return (

<div>

<button onClick={() => props.callbackFunc(++childCounterValue)}> Increment Counter </button>

</div>

);

}

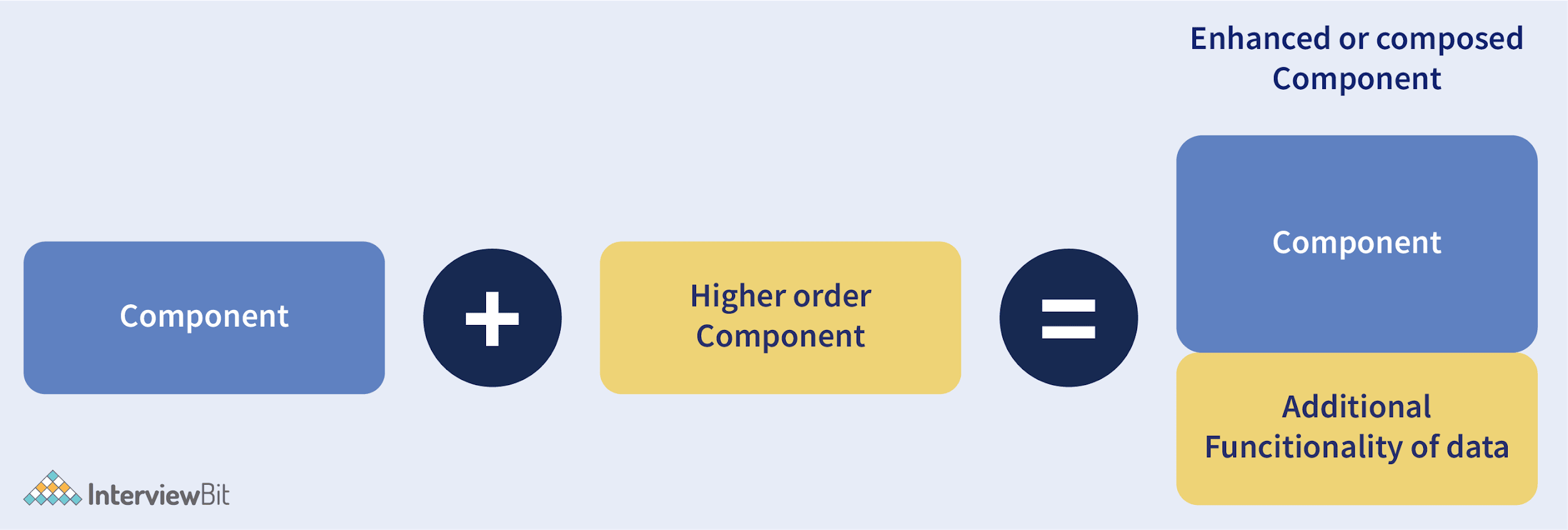
In the code above, we have used the props.counterValue and set it to a variable called childCounterValue.

Next, on button click, we pass the incremented childCounterValue to the props.callbackFunc.

This way, we can pass data from the child to the parent component.

### 6. What are Higher Order Components?

Simply put, Higher-Order Component(HOC) is a function that takes in a component and returns a new component.



When do we need a Higher Order Component?

While developing React applications, we might develop components that are quite similar to each other with minute differences. In most cases, developing similar components might not be an issue but, while developing larger applications we need to keep our code DRY, therefore, we want an abstraction that allows us to define this logic in a single place and share it across components. HOC allows us to create that abstraction.

Example of a HOC:

Consider the following components having similar functionality. The following component displays the list of articles:

// "GlobalDataSource" is some global data source

class ArticlesList extends React.Component {

constructor(props) {

super(props);

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

this.state = {

articles: GlobalDataSource.getArticles(),

};

}

componentDidMount() {

// Listens to the changes added

GlobalDataSource.addChangeListener(this.handleChange);

}

componentWillUnmount() {

// Listens to the changes removed

GlobalDataSource.removeChangeListener(this.handleChange);

}

handleChange() {

// States gets Update whenver data source changes

this.setState({

articles: GlobalDataSource.getArticles(),

});

}

render() {

return (

<div>

{this.state.articles.map((article) => (

<ArticleData article={article} key={article.id} />

))}

</div>

);

}

}

The following component displays the list of users:

// "GlobalDataSource" is some global data source

class UsersList extends React.Component {

constructor(props) {

super(props);

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

this.state = {

users: GlobalDataSource.getUsers(),

};

}

componentDidMount() {

// Listens to the changes added

GlobalDataSource.addChangeListener(this.handleChange);

}

componentWillUnmount() {

// Listens to the changes removed

GlobalDataSource.removeChangeListener(this.handleChange);

}

handleChange() {

// States gets Update whenver data source changes

this.setState({

users: GlobalDataSource.getUsers(),

});

}

render() {

return (

<div>

{this.state.users.map((user) => (

<UserData user={user} key={user.id} />

))}

</div>

);

}

}

Notice the above components, both have similar functionality but, they are calling different methods to an API endpoint.

Let’s create a Higher Order Component to create an abstraction:

// Higher Order Component which takes a component

// as input and returns another component

// "GlobalDataSource" is some global data source

function HOC(WrappedComponent, selectData) {

return class extends React.Component {

constructor(props) {

super(props);

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

this.state = {

data: selectData(GlobalDataSource, props),

};

}

componentDidMount() {

// Listens to the changes added

GlobalDataSource.addChangeListener(this.handleChange);

}

componentWillUnmount() {

// Listens to the changes removed

GlobalDataSource.removeChangeListener(this.handleChange);

}

handleChange() {

this.setState({

data: selectData(GlobalDataSource, this.props),

});

}

render() {

// Rendering the wrapped component with the latest data data

return <WrappedComponent data={this.state.data} {...this.props} />;

}

};

}

We know HOC is a function that takes in a component and returns a component.

In the code above, we have created a function called HOC which returns a component and performs functionality that can be shared across both the ArticlesList component and UsersList Component.

The second parameter in the HOC function is the function that calls the method on the API endpoint.

We have reduced the duplicated code of the componentDidUpdate and componentDidMount functions.

Using the concept of Higher-Order Components, we can now render the ArticlesList and UsersList components in the following way:

const ArticlesListWithHOC = HOC(ArticlesList, (GlobalDataSource) => GlobalDataSource.getArticles());

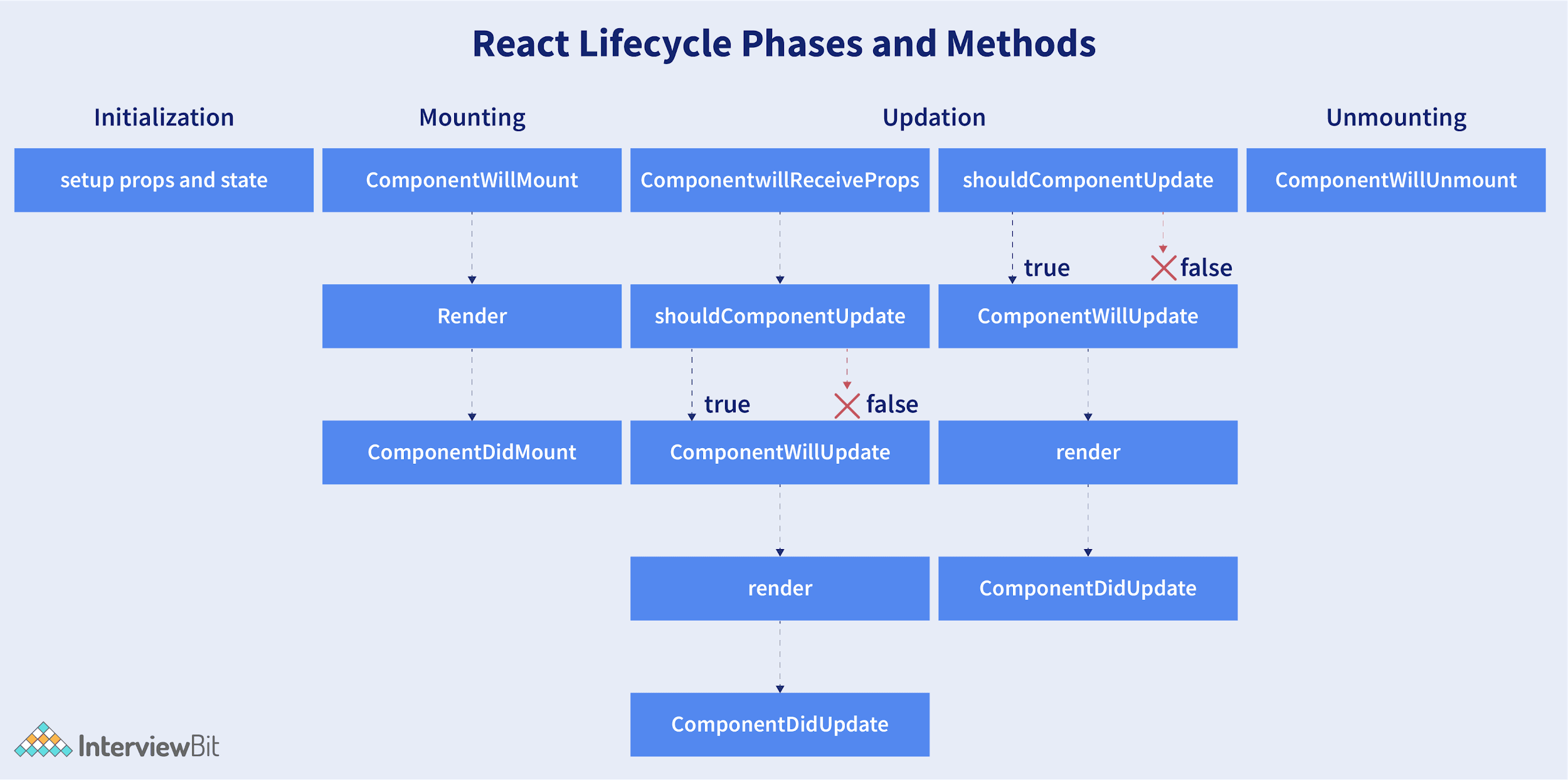
const UsersListWithHOC = HOC(UsersList, (GlobalDataSource) => GlobalDataSource.getUsers());

Remember, we are not trying to change the functionality of each component, we are trying to share a single functionality across multiple components using HOC.

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

There are four different phases in the lifecycle of React component. They are:

* Initialization: During this phase, React component will prepare by setting up the default props and initial state for the upcoming tough journey.
* Mounting: Mounting refers to putting the elements into the browser DOM. Since React uses VirtualDOM, the entire browser DOM which has been currently rendered would not be refreshed. This phase includes the lifecycle methods componentWillMount and componentDidMount.
* Updating: In this phase, a component will be updated when there is a change in the state or props of a component. This phase will have lifecycle methods like componentWillUpdate, shouldComponentUpdate, render, and componentDidUpdate.
* Unmounting: In this last phase of the component lifecycle, the component will be removed from the DOM or will be unmounted from the browser DOM. This phase will have the lifecycle method named componentWillUnmount.



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

React lifecycle hooks will have the methods that will be automatically called at different phases in the component lifecycle and thus it provides good control over what happens at the invoked point. It provides the power to effectively control and manipulate what goes on throughout the component lifecycle.

For example, if you are developing the YouTube application, then the application will make use of a network for buffering the videos and it consumes the power of the battery (assume only these two). After playing the video if the user switches to any other application, then you should make sure that the resources like network and battery are being used most efficiently. You can stop or pause the video buffering which in turn stops the battery and network usage when the user switches to another application after video play.

So we can say that the developer will be able to produce a quality application with the help of lifecycle methods and it also helps developers to make sure to plan what and how to do it at different points of birth, growth, or death of user interfaces.

The various lifecycle methods are:

* constructor(): This method will be called when the component is initiated before anything has been done. It helps to set up the initial state and initial values.
* getDerivedStateFromProps(): This method will be called just before element(s) rendering in the DOM. It helps to set up the state object depending on the initial props. The getDerivedStateFromProps() method will have a state as an argument and it returns an object that made changes to the state. This will be the first method to be called on an updating of a component.
* render(): This method will output or re-render the HTML to the DOM with new changes. The render() method is an essential method and will be called always while the remaining methods are optional and will be called only if they are defined.
* componentDidMount(): This method will be called after the rendering of the component. Using this method, you can run statements that need the component to be already kept in the DOM.
* shouldComponentUpdate(): The Boolean value will be returned by this method which will specify whether React should proceed further with the rendering or not. The default value for this method will be True.
* getSnapshotBeforeUpdate(): This method will provide access for the props as well as for the state before the update. It is possible to check the previously present value before the update, even after the update.
* componentDidUpdate(): This method will be called after the component has been updated in the DOM.
* componentWillUnmount(): This method will be called when the component removal from the DOM is about to happen.

### 9. Does React Hook work with static typing?

Static typing refers to the process of code check during the time of compilation for ensuring all variables will be statically typed. React Hooks are functions that are designed to make sure about all attributes must be statically typed. For enforcing stricter static typing within our code, we can make use of the React API with custom Hooks.

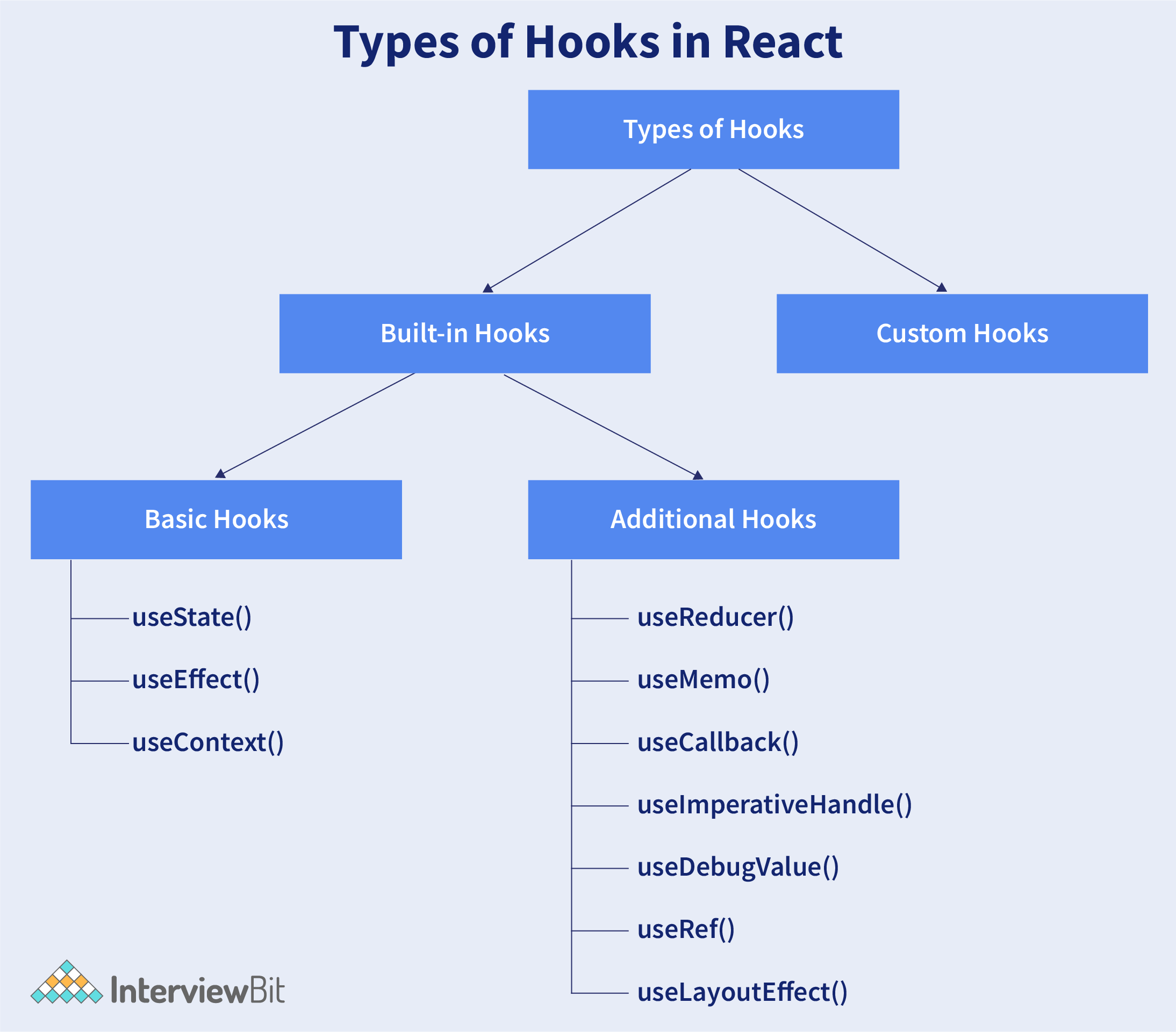
### 10. Explain about types of Hooks in React.

There are two types of Hooks in React. They are:

1. Built-in Hooks: The built-in Hooks are divided into 2 parts as given below:

* Basic Hooks:
  + useState(): This functional component is used to set and retrieve the state.
  + useEffect(): It enables for performing the side effects in the functional components.
  + useContext(): It is used for creating common data that is to be accessed by the components hierarchy without having to pass the props down to each level.
* Additional Hooks:
  + useReducer() : It is used when there is a complex state logic that is having several sub-values or when the upcoming state is dependent on the previous state. It will also enable you to optimization of component performance that will trigger deeper updates as it is permitted to pass the dispatch down instead of callbacks.
  + useMemo() : This will be used for recomputing the memoized value when there is a change in one of the dependencies. This optimization will help for avoiding expensive calculations on each render.
  + useCallback() : This is useful while passing callbacks into the optimized child components and depends on the equality of reference for the prevention of unneeded renders.
  + useImperativeHandle(): It will enable modifying the instance that will be passed with the ref object.
  + useDebugValue(): It is used for displaying a label for custom hooks in React DevTools.
  + useRef() : It will permit creating a reference to the DOM element directly within the functional component.
  + useLayoutEffect(): It is used for the reading layout from the DOM and re-rendering synchronously.

2. Custom Hooks: A custom Hook is basically a function of JavaScript. The Custom Hook working is similar to a regular function. The “use” at the beginning of the Custom Hook Name is required for React to understand that this is a custom Hook and also it will describe that this specific function follows the rules of Hooks. Moreover, developing custom Hooks will enable you for extracting component logic from within reusable functions.



### 11. Differentiate React Hooks vs Classes.

| React Hooks | Classes |
| --- | --- |
| It is used in functional components of React. | It is used in class-based components of React. |
| It will not require a declaration of any kind of constructor. | It is necessary to declare the constructor inside the class component. |
| It does not require the use of this keyword in state declaration or modification. | Keyword this will be used in state declaration (this.state) and in modification (this.setState()). |
| It is easier to use because of the useState functionality. | No specific function is available for helping us to access the state and its corresponding setState variable. |
| React Hooks can be helpful in implementing Redux and context API. | Because of the long setup of state declarations, class states are generally not preferred. |

### 12. How does the performance of using Hooks will differ in comparison with the classes?

* React Hooks will avoid a lot of overheads such as the instance creation, binding of events, etc., that are present with classes.
* Hooks in React will result in smaller component trees since they will be avoiding the nesting that exists in HOCs (Higher Order Components) and will render props which result in less amount of work to be done by React.

### 13. Do Hooks cover all the functionalities provided by the classes?

Our goal is for Hooks to cover all the functionalities for classes at its earliest. There are no Hook equivalents for the following methods that are not introduced in Hooks yet:

* getSnapshotBeforeUpdate()
* getDerivedStateFromError()
* componentDidCatch()

Since it is an early time for Hooks, few third-party libraries may not be compatible with Hooks at present, but they will be added soon.

### 14. What is React Router?

React Router refers to the standard library used for routing in React. It permits us for building a single-page web application in React with navigation without even refreshing the page when the user navigates. It also allows to change the browser URL and will keep the user interface in sync with the URL. React Router will make use of the component structure for calling the components, using which appropriate information can be shown. Since React is a component-based framework, it’s not necessary to include and use this package. Any other compatible routing library would also work with React.

The major components of React Router are given below:

* BrowserRouter: It is a router implementation that will make use of the HTML5 history API (pushState, popstate, and event replaceState) for keeping your UI to be in sync with the URL. It is the parent component useful in storing all other components.
* Routes: It is a newer component that has been introduced in the React v6 and an upgrade of the component.
* Route: It is considered to be a conditionally shown component and some UI will be rendered by this whenever there is a match between its path and the current URL.
* Link: It is useful in creating links to various routes and implementing navigation all over the application. It works similarly to the anchor tag in HTML.

### 15. Can React Hook replaces Redux?

The React Hook cannot be considered as a replacement for Redux (It is an open-source, JavaScript library useful in managing the application state) when it comes to the management of the global application state tree in large complex applications, even though the React will provide a useReducer hook that manages state transitions similar to Redux. Redux is very useful at a lower level of component hierarchy to handle the pieces of a state which are dependent on each other, instead of a declaration of multiple useState hooks.

In commercial web applications which is larger, the complexity will be high, so using only React Hook may not be sufficient. Few developers will try to tackle the challenge with the help of React Hooks and others will combine React Hooks with the Redux.

### 16. Explain conditional rendering in React.

Conditional rendering refers to the dynamic output of user interface markups based on a condition state. It works in the same way as JavaScript conditions. Using conditional rendering, it is possible to toggle specific application functions, API data rendering, hide or show elements, decide permission levels, authentication handling, and so on.

There are different approaches for implementing conditional rendering in React. Some of them are:

* Using if-else conditional logic which is suitable for smaller as well as for medium-sized applications
* Using ternary operators, which takes away some amount of complication from if-else statements
* Using element variables, which will enable us to write cleaner code.

### 17. Explain how to create a simple React Hooks example program.

I will assume that you are having some coding knowledge about JavaScript and have installed Node on your system for creating a below given React Hook program. An installation of Node comes along with the command-line tools: npm and npx, where npm is useful to install the packages into a project and npx is useful in running commands of Node from the command line. The npx looks in the current project folder for checking whether a command has been installed there. When the command is not available on your computer, the npx will look in the npmjs.com repository, then the latest version of the command script will be loaded and will run without locally installing it. This feature is useful in creating a skeleton React application within a few key presses.

Open the Terminal inside the folder of your choice, and run the following command:

npx create-react-app react-items-with-hooks

Here, the create-react-app is an app initializer created by Facebook, to help with the easy and quick creation of React application, providing options to customize it while creating the application? The above command will create a new folder named react-items-with-hooks and it will be initialized with a basic React application. Now, you will be able to open the project in your favourite IDE. You can see an src folder inside the project along with the main application component App.js. This file is having a single function App() which will return an element and it will make use of an extended JavaScript syntax(JSX) for defining the component.

JSX will permit you for writing HTML-style template syntax directly into the JavaScript file. This mixture of JavaScript and HTML will be converted by React toolchain into pure JavaScript that will render the HTML element.

It is possible to define your own React components by writing a function that will return a JSX element. You can try this by creating a new file src/SearchItem.jsand put the following code into it.

import React from 'react';

export function SearchItem() {

return (

<div>

<div className="search-input"> <input type="text" placeholder="SearchItem"/> </div>

<h1 className="h1">Search Results</h1>

<div className="items">

<table>

<thead>

<tr>

<th className="itemname-col">Item Name</th>

<th className="price-col">Price</th>

<th className="quantity-col">Quantity</th>

</tr>

</thead>

<tbody></tbody>

</table>

</div>

</div>

);

}

This is all about how you can create a component. It will only display the empty table and doesn’t do anything. But you will be able to use the Search component in the application. Open the file src/App.js and add the import statement given below to the top of the file.

import { SearchItem } from './SearchItem';

Now, from the logo.svg, import will be removed and then contents of returned value in the function App() will be replaced with the following code:

<div className="App">

<header> Items with Hooks </header>

<SearchItem/>

</div>

You can notice that the element <SearchItem/> has been used just similar to an HTML element. The JSX syntax will enable for including the components in this approach directly within the JavaScript code. Your application can be tested by running the below-given command in your terminal.

npm start

This command will compile your application and open your default browser into [http://localhost:4000](http://localhost:4000/). This command can be kept on running when code development is in progress to make sure that the application is up-to-date, and also this browser page will be reloaded each time you modify and save the code.

This application will work finely, but it doesn’t look nice as it doesn’t react to any input from the user. You can make it more interactive by adding a state with React Hooks, adding authentication, etc.

### 18. How to create a switching component for displaying different pages?

A switching component refers to a component that will render one of the multiple components. We should use an object for mapping prop values to components.

A below-given example will show you how to display different pages based on page prop using switching component:

import HomePage from './HomePage'

import AboutPage from './AboutPage'

import FacilitiesPage from './FacilitiesPage'

import ContactPage from './ContactPage'

import HelpPage from './HelpPage'

const PAGES = {

home: HomePage,

about: AboutPage,

facilities: FacilitiesPage,

contact: ContactPage

help: HelpPage

}

const Page = (props) => {

const Handler = PAGES[props.page] || HelpPage

return <Handler {...props} />

}

// The PAGES object keys can be used in the prop types for catching errors during dev-time.

Page.propTypes = {

page: PropTypes.oneOf(Object.keys(PAGES)).isRequired

}

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

It is possible to listen to the resize event in componentDidMount() and then update the width and height dimensions. It requires the removal of the event listener in the componentWillUnmount() method.

Using the below-given code, we can render the view when the browser is resized.

class WindowSizeDimensions extends React.Component {

constructor(props){

super(props);

this.updateDimension = this.updateDimension.bind(this);

}

componentWillMount() {

this.updateDimension()

}

componentDidMount() {

window.addEventListener('resize', this.updateDimension)

}

componentWillUnmount() {

window.removeEventListener('resize', this.updateDimension)

}

updateDimension() {

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

}

render() {

return <span>{this.state.width} x {this.state.height}</span>

}

}

### 20. How to pass data between sibling components using React router?

Passing data between sibling components of React is possible using React Router with the help of history.push and match.params.

In the code given below, we have a Parent component AppDemo.js and have two Child Components HomePage and AboutPage. Everything is kept inside a Router by using React-router Route. It is also having a route for /about/{params} where we will pass the data.

import React, { Component } from ‘react’;

class AppDemo extends Component {

render() {

return (

<Router>

<div className="AppDemo">

<ul>

<li> <NavLink to="/" activeStyle={{ color:'blue' }}>Home</NavLink> </li>

<li> <NavLink to="/about" activeStyle={{ color:'blue' }}>About </NavLink> </li>

</ul>

<Route path="/about/:aboutId" component={AboutPage} />

<Route path="/about" component={AboutPage} />

<Route path="/" component={HomePage} />

</div>

</Router>

);

}

}

export default AppDemo;

The HomePage is a functional component with a button. On button click, we are using props.history.push(‘/about/’ + data) to programmatically navigate into /about/data.

export default function HomePage(props) {

const handleClick = (data) => {

props.history.push('/about/' + data);

}

return (

<div> <button onClick={() => handleClick('DemoButton')}>To About</button> </div>

)

}

Also, the functional component AboutPage will obtain the data passed by props.match.params.aboutId.

export default function AboutPage(props) {

if(!props.match.params.aboutId) {

return <div>No Data Yet</div>

}

return (

<div>

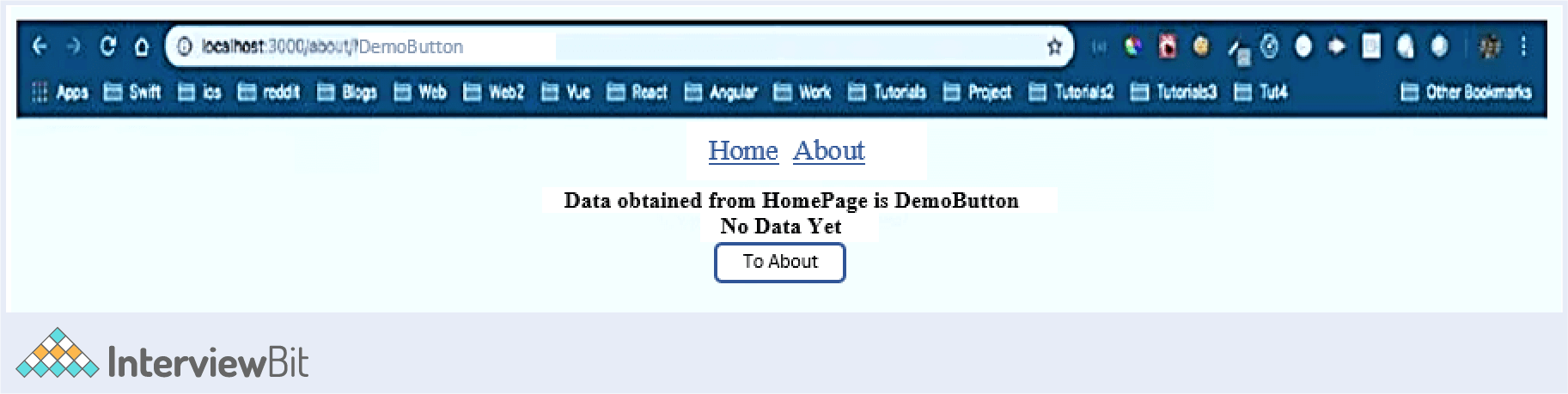
{`Data obtained from HomePage is ${props.match.params.aboutId}`}

</div>

)

}

After button click in the HomePage the page will look like below:



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

The react-router package will provide the component <Redirect> in React Router. Rendering of a <Redirect> component will navigate to a newer location. In the history stack, the current location will be overridden by the new location just like the server-side redirects.

import React, { Component } from 'react'

import { Redirect } from 'react-router'

export default class LoginDemoComponent extends Component {

render() {

if (this.state.isLoggedIn === true) {

return <Redirect to="/your/redirect/page" />

} else {

return <div>{'Please complete login'}</div>

}

}

}

## 

22. What is React or ReactJS?

Component-based Javascript library

Javascript framework

Javascript file

None of the above

23. Which of the following comes under the advantages of React?

Integration with other frameworks (like BackboneJS, Angular, etc.) becomes easier because it is only a view library

Increases the performance of an application using Virtual DOM

Can render both on server and client side

================

### 1. How to create components in React?

*Components form the fundamental building blocks of a React application. The candidate should be able to explain the two approaches to creating a component in React along with their differences and applications.*

React provides two ways to create a component – function components and class components.

1. Function Components is the simplest way to create a component. It uses pure JavaScript functions that accept props and returns a React element.

function Welcome({ message }) {

return <h1>{`Hello, ${message}`}</h1>

}

1. Class Components, on the other hand, uses ES6 class to define a component. The same function component can be rewritten as the following class component:

class Welcome extends React.Component {

render() {

return <h1>{`Hello, ${this.props.message}`}</h1>

}

}

### 2. What are props in React?

*Props are one of the most basic ways of passing data between components in React. The candidate should be able to explain what props are and how they’re used to communicate between different components. Don’t forget to provide a couple of examples to demonstrate its usage in a real application.*

When building an application, you often end up building a component tree with a hierarchy of components. Oftentimes, you will need to pass data between the components as you go down the tree. These are where props come into play.

Props are the component’s inputs. They can be in the form of a single value or an object containing a set of values that are passed from a parent component to a child component.

The snippet below shows how a prop is passed from the parent:

<Welcome name={'Hello'}>

The props can then be accessed via the child component’s props property:

*// for function components*

function Welcome(props) {

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

}

*// for class components*

class Welcome extends React.Component {

render() {

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

}

}

### 3. What are states in React?

*A likely follow-up to the previous question, still in the realm of managing data inside a component. Understanding the concept of state in React is vital to being able to use components effectively. Make sure your candidate is able to explain what states are and how they are used in a React component.*

State is an object that contains information local to the component that may change over the lifetime of the component.The snippet below shows a state object containing the property message which is then rendered in the h1 tag.

class Welcome extends React.Component {

constructor(props) {

super(props)

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

}

render() {

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

}

}

### 4. What is context in React?

### *Context is a data-sharing mechanism in React. You might ask this question to gauge the candidate’s knowledge of how components can pass and share data. The candidate should be able to explain what context is and their ideal use cases.*

Context is an alternate way to pass data through the component tree without passing props manually at every level. Context is designed to share data that is considered “global” within a tree of components.

For example, if you have a component tree with the following structure:

Component A

|

Component B

|

Component C

|

Component D

Passing data from component A to component D using props would require the data to be passed from A to B to C and finally to D. Context provides a way to share values like these between components to avoid the above scenario.

A common use case of context is application theming, language, and user information which are accessed from various parts of the application. The example below shows how user information is set and accessed by different components in the application.

*// Create a context for the current user (with '' as the default).*

const UserContext = React.createContext('');

class App extends React.Component {

render() {

*// Use a Provider to pass the current user to the tree below. Any component can read it, no matter how deep it is. In this example, we're passing 'Mike' as the current value.*

return (

<UserContext.Provider value='Mike'> <Toolbar /> </UserContext.Provider>

);

}

}

*// A component in the middle doesn't have to*

*// pass the user information down explicitly.*

function Toolbar() {

return (

<div> <Greeting /> </div>

);

}

class Greeting extends React.Component {

*// Assign a contextType to read the current user context.*

*// React will find the closest theme Provider above and use its value.*

static contextType = UserContext;

render() {

return <h1>{this.context}</h1>; *// will display "Mike" in an h1 tag*

}

}

### 5. How do you conditionally render components?

*Conditionally rendering a component is a common requirement in most applications. For example, a component might need to be rendered only if a certain set of data is available in your application’s state. Your candidate should explain the different approaches available and how to use them.*

JSX doesn’t render false or undefined. This behavior lets us use conditional short-circuiting to render parts of your component using the && notation.

For example, a component that only displays the p tag when the name property exists can be written as:

function Welcome(props) {

return (

<div>

{props.name && <p>{props.name}</p>}

</div>

);

}

A ternary operator can also be used to create an if-else condition. Using the same example above, we could add a fallback string that displays “Name not available” when name is undefined.

function Welcome(props) {

return (

<div>

{props.name ? <p>{props.name}</p>} : <p>Name not available</p>

</div>

);

}

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

*Handling events are a common requirement when building applications. These events could be DOM events, such as clicks or custom events that a component emits. A candidate should be able to explain how events work along with the various ways to listen to the emitted events.*

React provides three approaches to listen to emitted events:

1. Callback with arrow functions – Bind the event to an arrow function that calls the desired function.

class Greeting extends React.Component {

handleClick() {

console.log('Clicked');

}

render() {

return <button onClick={() => this.handleClick()}>Click Me</button>;

}

}

1. Bind in constructor – Event handlers defined as class methods in React aren’t bound by default. We could manually bind them in the constructor to allow them to be called from the template.

class Greeting extends React.Component {

constructor(props) {

super(props);

this.handleClick = this.handleClick.bind(this);

}

handleClick() { console.log('Clicked'); }

render() {

return <button onClick={this.handleClick}>Click Me</button>;

}

}

1. Public class fields syntax – We can avoid the manual binding of the method defining the method using the arrow function. Arrow functions don’t have their own this keyword causing the this keyword to be bound lexically.

class Greeting extends React.Component {

handleClick = () => {

console.log('Clicked');

}

render() {

return <button onClick={this.handleClick}>Click Me</button>;

}

}

### 7. What is a SyntheticEvent in React?

*SyntheticEvent provides a convenient abstraction over browser events. This abstraction streamlines the interaction with browser events from our React application. You might ask this question to gain insight into the candidate’s level of understanding of events and how they work in React.*

SyntheticEvent is a cross-browser wrapper around the browser’s native event. It has the same API as its browser’s native counterpart, including methods like stopPropagation() and preventDefault(). However, unlike the browser’s native events, SyntheticEvent works identically across all browsers.

### 8. How to conditionally apply class attributes?

*Similar to hiding and displaying an element, sometimes, we also need to conditionally apply styling to our element. Typically, styles are applied via classes instead of directly setting it on the element. You might ask this question to understand the candidate’s familiarity with using conditionals in React and how they can be applied to class attributes.*

React lets you include conditionals such as && and ternary operators to conditionally display parts of your component. The same concept can be applied to classNames.

The snippet below uses a ternary operator to add the disabled class when the disabled prop is true and the default class if it’s false.

<div className={'btn-panel ' + (this.props.disabled ? 'disabled' : 'default')}>

### 9. What are refs in React?

*Sometimes accessing the DOM directly is required. Understanding the recommended approach to accessing the DOM could save you from unexpected behaviors in your application. The candidate should be able to explain the purpose of refs, how refs are used, along with when they should be used.*

Refs provide a way to access DOM nodes or React elements created in the render method.

Refs act as an escape hatch to the regular React dataflow of passing props down a component tree. They are used to update the child component (including both React components and DOM elements) without passing in new props.

The following snippet shows how refs are used to set the focus on the input element when a button is clicked.

class CustomTextInput extends React.Component {

constructor(props) {

super(props); *// create a ref to store the textInput DOM element*

this.textInput = React.createRef();

this.focusTextInput = this.focusTextInput.bind(this);

}

focusTextInput() {

*// Access the textInput using "current" and set the focus using the input's DOM API*

this.textInput.current.focus();

}

render() {

return (

<div>

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

<input type="button" value="Focus" onClick={this.focusTextInput} />

</div>

);

}

}

A few use cases for refs include:

* managing focus and text selection
* triggering imperative animations
* integrating with third-party DOM libraries

### 10. What are the four stages a React component goes through?

Or, asked another way: *What happens during the lifecycle of a React component?*

One of the most valuable parts of React is its component lifecycle — so understanding exactly how components function over time is instrumental in building a maintainable application.

Each React component has its own lifecycle. Each stage of the lifecycle invokes a series of methods allowing us to perform specific tasks at a specific stage of the component’s lifecycle. Understanding the different stages within a component’s lifecycle enables us to use them effectively. The candidate should be able to iterate what the four stages are and what happens in each stage.

A React Component goes through the following four stages:

1. Initialization – the component is constructed with the given props and default state
2. Mounting – rendering the JSX returned by the render method
3. Updating – when the state of a component is updated and the application is repainted
4. Unmounting – the final stage of the component lifecycle where the component is removed from the DOM

### 11. What is React? How is it different from other JS frameworks?

*Although this sounds like a relatively simple question, it’s asking the candidate to state an informed opinion about React and any competing alternatives. In short, this question is designed to test a candidate’s knowledge about the JavaScript ecosystem at large while also pressing for specifics on what makes React unique.*

Let’s look at each part of the answer separately.

#### What is React?

React is an open-source JavaScript library created by Facebook for building complex, interactive UIs in web and mobile applications.

The key point in this answer is that React’s core purpose is to build UI components; it is often referred to as just the “V” (View) in an “MVC” architecture. Therefore it has no opinions on the other pieces of your technology stack and can be seamlessly integrated into any application.

#### How is React different?

The answer to this question will likely vary depending on the candidate’s personal experiences. The important thing is to listen for real-life examples provided and opinions on whether or not the candidate prefers React and why.

Because React is a small library focused on building UI components, it is necessarily different from many other JavaScript frameworks.

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 in many situations, 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 incredible flexibility in choosing the architecture they deem “best” — though it also places the responsibility of choosing (or building) those parts on the developer.

“*I recently migrated an application originally written in Angular to React, and one of the things I loved most was…*“

Be prepared to ask some follow-up questions as well, such as:

* Under what circumstances would you choose React over another technology? For example, *React vs Angular* or *React vs Vue*.
* If React only focuses on a small part of building UI components, can you explain some pitfalls one might encounter when developing a large application?
* If you were rewriting an Angular application in React, how much code could you expect to re-use?

### 12. What can you tell me about JSX?

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.

Asking questions about JSX tests whether or not the candidate can state an informed opinion towards JSX and defend it based on personal experience. Let’s cover some of the basic talking points.

Adopting JSX allows the developer to simultaneously adopt ES2015 — giving immediate access to some wonderful syntactic sugar.

ES2015 introduced a variety of new features to JavaScript that makes writing large applications far easier than ever before: classes, block scoping via let, and the new spread operator are just a small portion of the additions.

import AnotherClass from './AnotherClass';

class MyComponent extends React.Component {

render() {

let props = this.props;

return (

<div className="my-component">

<AnotherClass {...props} />

</div>

);

}

}

But while ES2015 is becoming more and more widespread, it still is far from widely supported by the major browsers — so tools like Babel or webpack are needed to convert everything into legacy ES5 code.

### 13. What are stateless components?

If React components are essentially state machines that generate UI markup, then what are stateless components?

Stateless components (a flavor of “reusable” components) are nothing more than pure functions that render DOM-based solely on the properties provided to them.

const StatelessCmp = props => {

return (

<div className="my-stateless-component"> {props.name}: {props.birthday} </div>

);

};

*// ---*

ReactDOM.render(

<StatelessCmp name="Art" birthday="10/01/1980" />,

document.getElementById('main')

);

This component has no need for any internal state — let alone a constructor or lifecycle handlers. The component’s output is purely a function of the properties provided to it.

* When should you use a class component over a function component?
* What are the differences between stateless and stateful components?
* What are the differences between controlled and uncontrolled components?
* How to apply validation on props in React?
* How to listen to state changes?
* How do you lift state up in React?
* How to set the focus of an input element on page load?
* What are forward refs?
* What is the difference between HTML and React event handling?
* What are the lifecycle methods in React?

## Intermediate React Interview Questions and Answers

The following set of questions should test a candidate’s mid-level knowledge of React and some of its core features.

### 1. How to create elements in a loop in React?

*Applications usually involve displaying a dynamic set of components based on an array or object. The candidate should be able to explain what a v-for directive is along with their applications.*

React lets us use JavaScript’s map function to loop through an array or object – rendering the returned template for each entry.

The following code shows how the items array is used to generate a set of ItemComponent:

<div>

{ items.map(item =>

<ItemComponent key={item.id} item={item}/>

)}

</div>

### 2. How do you update the state object in React?

*Keeping your views and your logic in sync is an important aspect of every application. This is equivalent to updating the state object and making sure the component is updated accordingly. The candidate should be able to explain the recommended approach when working with the state object and how to avoid the component from getting out of sync with the state.*

React provides utility functions to manage your component’s local state. The setState function is the recommended approach to update the state object as React does the heavy lifting for you.

The setState function merges the new object with the current state. The updated state then triggers a component re-render automatically, ensuring that your views are always up-to-date with the latest state changes in your component.

The following example shows how the user object is updated using the setState function:

class Profile extends React.Component {

constructor() {

super();

this.state = { name: 'Mike' };

}

updateName() {

this.setState({ name: 'Bob' });

}

render() {

return ( <div> <h1>{this.state.name}</h1> </div> );

}

}

### 3. What are Higher-order components?

*Higher-order components (HOC) is a pattern derived from React’s compositional nature. You might ask this question to better understand the candidate’s familiarity with React’s component system. The candidate should be able to explain what HOCs are conceptually and what problem it solves.*

HOC is a function that accepts a component and returns a new component. These components are also referred to as pure components as they accept dynamic child component but doesn’t modify or copy any of the child components.

Use cases of HOCs include:

* Code reuse and logic abstraction
* State abstraction and manipulation
* Props manipulation
* Render hijacking

### 4. What is the purpose of the useMemo hook?

*Optimization of processes is a necessity as applications grow and begins to perform heavier tasks. Memoization is an optimization technique that helps speed up expensive function calls using cached results. Understanding optimization techniques is important to keep your code fast and efficient. The candidate should be able to explain what memoization is and also how to use React’s useMemo hook to optimize your application.*

Memoization is an optimization technique that speeds up your code by storing the results of expensive function calls and reusing the stored result when the same input occurs again.

Functional components use the useMemo hook to memoize expensive functions. As a result, these functions are only called when the input changes instead of on every render.

The following code shows a memoized function using the useMemo hook. useMemo will only recompute the memoized value when one of the dependencies (a or b) has changed.

const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);

### 5. What are Render Props?

*Understanding different techniques to share code between components helps make an informed decision on the best approach*. The candidate should have a basic understanding of what render props are, *along with their ideal use cases.*

Render prop is a technique to share code between components by passing a function (the function should return an element) as the component’s props. The component with the render prop can then use the element from the passed-in function in its render function to compose a new component.

The code below shows how a function is passed in as a prop to the DataProvider component:

<DataProvider render={data => (

<h1>Hello {data.target}</h1>

)}/>

### 6. Are you familiar with Flux?

Flux is an architectural pattern that enforces unidirectional data flow — its core purpose is to control derived data so that multiple components can interact with that data without risking pollution.

The Flux pattern is generic; it’s not specific to React applications, nor is it required to build a React app. However, Flux is commonly used by React developers because React components are declarative — the rendered UI (View) is simply a function of state (Store data).

Flux is relatively simple in concept, but in a technical interview, it’s important that the developer demonstrates a deep understanding of its implementation. Let’s cover of the important few discussion points.

#### Description of Flux

In the Flux pattern, the Store is the central authority for all data; any mutations to the data must occur within the store. Changes to the Store data are subsequently broadcast to subscribing Views via events. Views then update themselves based on the new state of received data.

To request changes to any Store data, Actions may be fired. These Actions are controlled by a central Dispatcher; Actions may not occur simultaneously, ensuring that a Store only mutates data once per Action.

The strict unidirectional flow of this Flux pattern enforces data stability, reducing data-related runtime errors throughout an application.

#### Flux vs MVC

Traditional MVC patterns have worked well for separating the concerns of data (Model), UI (View) and logic (Controller) — but many web developers have discovered limitations with that approach as applications grow in size. Specifically, 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.

During a technical interview, one should discuss the differences between the Flux and MVC design patterns within the context of a specific example:

*For example, imagine we have a “master/detail” UI in which the user can select a record from a list (master view) and edit it using an auto-populated form (detail view).*

*With an MVC architecture, the data contained within the Model is shared between both the master and detail Views. Each of these views might have its own Controller delegating updates between the Model and the View. At any point the data contained within the Model might be updated — and it’s difficult to know where exactly that change occurred. Did it happen in one of the Views sharing that Model, or in one of the Controllers? Because the Model’s data can be mutated by any actor in the application, the risk of data pollution in complex UIs is greater than we’d like.*

*With a Flux architecture, the Store data is similarly shared between multiple Views. However this data can’t be directly mutated — all of the requests to update the data must pass through the Action > Dispatcher chain first, eliminating the risk of random data pollution. When updates are made to the data, it’s now much easier to locate the code requesting those changes.*

#### Difference with AngularJS (1.x)

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.

#### Testing

One of the most valuable aspects of applications built on Flux is that their components become incredibly easy to test. Developers can recreate and test the state of any React component by simply updating the store — direct interactions with the UI (with tools like Selenium) are no longer necessary in many cases.

#### Popular Flux Libraries

While Flux is a general pattern for enforcing data flow through an application, there exist many implementations from which to choose. There are nuances between each implementation, as well as specific pros and cons to consider. The candidate should provide examples of real-world experience with using Flux.

### More React Intermediate Interview Questions to Practice

Before we wrap this section up, here are a few other React intermediate interview questions you might want to ask your candidates:

* What are pure components?
* What is the difference between setState and replaceState methods?
* How do you update the state with values that depend on the current state?
* How do you force a component to re-render without calling setState?
* How do you set dynamic key names in the state object?
* How to re-render the view when the viewport size changes?
* What is the purpose of the useEffect hook?
* What is the purpose of the useRefs hook?
* Does React Hook work with static typing?
* How do you build a custom hook?

And one last intermediate React question asked at interviews:

“Explain this code.”

## Advanced React Interview Questions

The following set of questions should test the candidate’s advanced knowledge of React and some of its widely used features.

### 1. What are portals in React?

*Most applications contain modals, toasts, and other elements that require escaping the default DOM hierarchy. Portals provide a convenient way to render components in a separate DOM node. The candidate should be able to explain how portals work and their ideal use cases.*

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

A portal is created by calling ReactDom‘s createPortal function.

ReactDOM.createPortal(content, containerElement);

The createPortal function accepts two parameters:

* content – a valid renderable React element
* containerElement – a valid DOM element to append the content

### 2. What are error boundaries?

*A JavaScript error in one part of the app shouldn’t break the entire app. Error boundary is React’s solution to this common problem. The candidate should be able to explain an error boundary and use it to handle errors in your application.*

Error boundaries are React components that catch JavaScript errors anywhere in its child component tree. The Error boundary component can then log those errors and display fallback UI instead of crashing the entire component tree. You can think of error boundaries as a catch block for components.

The snippet below shows a simple ErrorBoundary component that displays a fallback h1 and logs the error.

class ErrorBoundary extends React.Component {

constructor(props) {

super(props);

this.state = { hasError: false };

}

static getDerivedStateFromError(error) {

return { hasError: true };

}

componentDidCatch(error, errorInfo) {

console.log(error, errorInfo);

}

render() {

if (this.state.hasError) { *// fallback UI*

return <h1>Something went wrong.</h1>;

}

return this.props.children;

}

}

The ErrorBoundary component can then be used to catch all errors in the Child component as follows:

<ErrorBoundary>

<Child />

</ErrorBoundary>

### 3. What is React Profiler and what is it used for?

*Optimizations are a key part of every application. Knowing which tools to use can be a huge productivity boost. This is a more advanced question that you might ask to understand the candidate’s ability to diagnose and optimize a React application.*

React Profiler is a means to measure the cost of rendering in a React application. The purpose of this component is to help developers identify parts of the application that are slow and may benefit from further optimizations.

The Profiler can be added anywhere in the component tree to measure its rendering cost. For example, the code below shows how the Profiler component is used to measure the rendering cost of the Navigation component and its descendants:

render(

<App>

<Profiler id="Navigation" onRender={callback}>

<Navigation {...props} />

</Profiler>

<Main {...props} />

</App>

)

### 4. What is StrictMode in React?

*Catching potential problems is a huge benefit when building applications. React has a built-in component to help developers do just that. You might ask your candidate this question to better understand the depth of their knowledge of more advanced React components. The candidate should be able to explain what the component does, its usage, and its benefits.*

StrictMode is a tool to highlight potential problems in an application. Although StrictMode is used as a component, it doesn’t create a visible UI in the DOM. It only enables additional checks for its descendants.

You can enable strict mode for any part of your application by wrapping the components inside the React.StrictMode component. The following code shows how strict mode checks are run on the BlogContent component and its descendants. The Header component won’t be checked as it is outside of the React.StrictMode component.

import React from 'react';

function Application() {

return (

<div>

<Header />

<React.StrictMode>

<div> <BlogContent /> </div>

</React.StrictMode>

</div>

);

}

Benefits of using StrictMode include:

* Identifying components with unsafe lifecycles
* Warning about legacy string ref API usage
* Warning about deprecated findDOMNode usage
* Detecting unexpected side effects
* Detecting legacy context API
* Ensuring reusable state

### 5. What are React Fragments used for?

Sometimes, we need to return multiple React elements from a single parent. This only works when the elements are wrapped under a single generic HTML element like a div. Fragments allow us to return multiple elements directly without the additional wrapper element.

React fragments are special features that allow creating a group of children elements or components without creating an actual node in the template. Fragments are denoted by an open empty tag (<>) and a closed empty tag (</>).

The code below shows how a React component could return two td elements without wrapping them inside a div container.

function Columns() {

return (

<>

<td>Column 1</td>

<td>Column 2</td>

</>

);

}

How to prevent re-renders in React?

* How do you modularize code in a React project?
* What is React Router?
* How does the React Router differ from conventional routing?
* How do you access query parameters using React Router?
* What is Redux?
* How does data flow in a React-Redux application?
* When do you use React Context vs Redux?
* How do you pass data from a child component to its parent?
* How do you use decorators in React?

## 1. What is React?

[React](https://reactjs.org/) is a "JavaScript library for building user interfaces" which was developed by Facebook in 2011.

It’s the V in the MVC (Model - View -Controller), so it is rather an open-source UI library than a framework.

## 2. What are the advantages of React?

* Good performance: due to VDOM, see [#17](https://mokkapps.de/blog/my-top-react-interview-questions#7-what-is-the-virtual-dom).
* Easy to learn: with basic JavaScript knowledge you can start building applications. Frameworks like Angular require more knowledge about other technologies and patterns like RxJS, TypeScript, and Dependency Injection.
* One-way data flow: this flow is also called "parent to child" or "top to bottom" and prevents errors and facilitates debugging.
* Reusable components: Re-using React components in other parts of the code or even in different projects can be done with little or no changes.
* Huge community: The community supplies a ton of libraries that can be used to build React applications.
* It is very popular among developers.

## [3. What are the disadvantages of React?](https://mokkapps.de/blog/my-top-react-interview-questions#_3-what-are-the-disadvantages-of-react)

* As React provides only the View part of the MVC model you mostly will rely on other technologies in your projects as well. Therefore, every React project might look quite different.
* Some people think that JSX is too difficult to grasp and too complex.
* Often poor documentation for React and its libraries.

## [4. What is JSX?](https://mokkapps.de/blog/my-top-react-interview-questions#_4-what-is-jsx)

JSX (JavaScript XML) allows us to write HTML inside JavaScript. The [official docs](https://reactjs.org/docs/introducing-jsx.html) describe it as "syntax extension to JavaScript".

React recommends using JSX, but it is also possible to create applications [without using JSX](https://reactjs.org/docs/react-without-jsx.html) at all.

A simple JSX example:

const element = <h1>Hello, world!</h1>

## [5. How to pass data between components?](https://mokkapps.de/blog/my-top-react-interview-questions#_5-how-to-pass-data-between-components)

1. Use props to pass data from parent to child.
2. Use callbacks to pass data from child to parent.
3. Use any of the following methods to pass data among siblings:
   * Integrating the methods mentioned above.
   * Using [Redux](https://redux.js.org/).
   * Utilizing [React's Context API](https://reactjs.org/docs/context.html#api).

## [6. What are the differences between functional and class components?](https://mokkapps.de/blog/my-top-react-interview-questions#_6-what-are-the-differences-between-functional-and-class-components)

[Hooks](https://reactjs.org/docs/hooks-intro.html) were introduced in React 16.8. In previous versions, functional components were called stateless components and did not provide the same features as class components (e.g., accessing state). Hooks enable functional components to have the same features as class components. There are no plans to remove class components from React.

So let's take a look at the differences:

### [Declaration & Props](https://mokkapps.de/blog/my-top-react-interview-questions#declaration-props)

#### [Functional Component](https://mokkapps.de/blog/my-top-react-interview-questions#functional-component)

Functional components are JavaScript functions and therefore can be declared using an arrow function or the function keyword. Props are simply function arguments and can be directly used inside JSX:

const Card = (props) => {

return <h2>Title: {props.title}</h2>

}

function Card(props) {

return <h2>Title: {props.title}</h2>

}

#### [Class component](https://mokkapps.de/blog/my-top-react-interview-questions#class-component)

Class components are declared using the ES6 class keyword. Props need to be accessed using the this keyword:

class Card extends React.Component {

constructor(props) {

*super*(props)

}

render() { return <h2>Title: {*this*.props.title}</h2> }

}

### [Handling state](https://mokkapps.de/blog/my-top-react-interview-questions#handling-state)

#### [Functional components](https://mokkapps.de/blog/my-top-react-interview-questions#functional-components)

In functional components we need to use the useState hook to be able to handle state:

const Counter = (props) => {

const [counter, setCounter] = useState(0)

const increment = () => {

setCounter(++counter)

}

return (

<div>

<p>Count: {counter}</p>

<button *onClick*={increment}>Increment Counter</button>

</div>

)

}

#### [Class components](https://mokkapps.de/blog/my-top-react-interview-questions#class-components)

It's not possible to use React Hooks inside class components, therefore state handling is done differently in a class component:

class Counter extends React.Component {

constructor(props) {

*super*(props)

*this*.state = { counter: 0 }

*this*.increment = *this*.increment.bind(*this*)

}

increment() {

*this*.setState((prevState) => {

return { counter: prevState.counter + 1 }

})

}

render() {

return (

<div>

<p>Count: {*this*.state.counter}</p>

<button *onClick*={*this*.increment}>Increment Counter</button>

</div>

)

}

}

## [7. What is the Virtual DOM?](https://mokkapps.de/blog/my-top-react-interview-questions#_7-what-is-the-virtual-dom)

The [Virtual DOM (VDOM)](https://reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom) is a lightweight JavaScript object and it contains a copy of the real DOM.

| Real DOM | Virtual DOM |
| --- | --- |
| Slow & expensive DOM manipulation | Fast & inexpensive DOM manipulation |
| Allows direct updates from HTML | It cannot be used to update HTML directly |
| Wastes too much memory | Less memory consumption |

## [8. Is the Shadow DOM the same as the Virtual DOM?](https://mokkapps.de/blog/my-top-react-interview-questions#_8-is-the-shadow-dom-the-same-as-the-virtual-dom)

No, they are different. The [Shadow DOM](https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM) is a browser technology designed primarily for scoping variables and CSS in web components.

The virtual DOM is a concept implemented by libraries in JavaScript on top of browser APIs.

## [9. What is "React Fiber"?](https://mokkapps.de/blog/my-top-react-interview-questions#_9-what-is-react-fiber)

Fiber is the new reconciliation engine in React 16.

Its headline feature is incremental rendering: the ability to split rendering work into chunks and spread it out over multiple frames.

## [10. How does state differ from props?](https://mokkapps.de/blog/my-top-react-interview-questions#_10-how-does-state-differ-from-props)

Both props and state are plain JavaScript objects.

Props (short for "properties") is an object of arbitrary inputs that are passed to a component by its parent component.

State are variables that are initialized and managed by the component and change over the lifetime of a specific instance of this component.

[This article from Kent C. Dodds](https://kentcdodds.com/blog/props-vs-state) provides a more detailed explanation.

## [11. What are the differences between controlled and uncontrolled components?](https://mokkapps.de/blog/my-top-react-interview-questions#_11-what-are-the-differences-between-controlled-and-uncontrolled-components)

The value of an input element in a controlled React component is controlled by React.

The value of an input element in an uncontrolled React component is controlled by the DOM.

## [12. What are the different lifecycle methods in React?](https://mokkapps.de/blog/my-top-react-interview-questions#_12-what-are-the-different-lifecycle-methods-in-react)

React class components provide these lifecycle methods:

* componentDidMount(): Runs after the component output has been rendered to the DOM.
* componentDidUpdate(): Runs immediately after updating occurs.
* componentWillUnmount(): Runs before the component is unmounted from the DOM and is used to clear up the memory space.

There exist some other [rarely used](https://reactjs.org/docs/react-component.html#rarely-used-lifecycle-methods) and [legacy](https://reactjs.org/docs/react-component.html#legacy-lifecycle-methods) lifecycle methods.

Hooks are used in functional components instead of the above-mentioned lifecycle methods. The Effect Hook useEffect adds, for example, the ability to perform side effects and provides the same functionality as componentDidMount, componentDidUpdate, and componentWillUnmount.

## [13. How can you improve your React app's performance?](https://mokkapps.de/blog/my-top-react-interview-questions#_13-how-can-you-improve-your-react-apps-performance)

* Use [React.PureComponent](https://reactjs.org/docs/react-api.html#reactpurecomponent) which is a base class like React.Component but it provides in some cases a performance boost if its render() function renders the same result given the same props and state.
* Use [useMemo Hook](https://reactjs.org/docs/hooks-reference.html#usememo) to memoize functions that perform expensive calculations on every render. It will only recompute the memoized value if one of the dependencies (that are passed to the Hook) has changed.
* State colocation is a process that moves the state as close to where you need it. Some React applications have a lot of unnecessary state in their parent component which makes the code harder to maintain and leads to a lot of unnecessary re-renders. [This article](https://kentcdodds.com/blog/state-colocation-will-make-your-react-app-faster) provides a detailed explanation about state colocation.
* Lazy load your components to reduce the load time of your application. React [Suspense](https://reactjs.org/docs/react-api.html#suspense) can be used to lazy load components.

## [14. What are keys in React?](https://mokkapps.de/blog/my-top-react-interview-questions#_14-what-are-keys-in-react)

React needs keys to be able to identify which elements were changed, added, or removed. Each item in an array needs to have a key that provides a stable identity.

It's not recommended to use indexes for keys if the order of items may change as it can have a negative impact on the performance and may cause state issues. React will use indexes as keys if you do not assign an explicit key to list items.

## [15. What are Higher Order Components?](https://mokkapps.de/blog/my-top-react-interview-questions#_15-what-are-higher-order-components)

A [higher-order component (HOC)](https://reactjs.org/docs/higher-order-components.html#use-hocs-for-cross-cutting-concerns) is a function that takes a component and returns a new component.

They are an advanced technique in React for reusing component logic and they are not part of the React API, per se. They are a pattern that emerges from React’s compositional nature:

const EnhancedComponent = higherOrderComponent(WrappedComponent)

Whereas a component transforms props into UI, a higher-order component transforms a component into another component.

## [16. What are error boundaries?](https://mokkapps.de/blog/my-top-react-interview-questions#_16-what-are-error-boundaries)

React 16 introduced a new concept of an “error boundary”.

[Error boundaries](https://reactjs.org/docs/error-boundaries.html#gatsby-focus-wrapper) 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. Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

## [17. Why Hooks were introduced?](https://mokkapps.de/blog/my-top-react-interview-questions#_17-why-hooks-were-introduced)

Hooks solve a wide variety of seemingly unconnected problems in React that were encountered by Facebook over five years of writing and maintaining tens of thousands of components:

* Hooks allow you to reuse stateful logic without changing your component hierarchy.
* Hooks let you split one component into smaller functions based on what pieces are related (such as setting up a subscription or fetching data).
* Hooks let you use more of React’s features without classes.
* It removed the complexity of dealing with the this keyword inside class components.

## [18. What is the purpose of useEffect hook?](https://mokkapps.de/blog/my-top-react-interview-questions#_18-what-is-the-purpose-of-useeffect-hook)

The [Effect hook](https://reactjs.org/docs/hooks-reference.html#useeffect) lets us perform side effects in functional components. It helps us to avoid redundant code in different lifecycle methods of a class component. It helps to group related code.

## [19. What are synthetic events in React?](https://mokkapps.de/blog/my-top-react-interview-questions#_19-what-are-synthetic-events-in-react)

[SyntheticEvent](https://reactjs.org/docs/events.html) is a cross-browser wrapper around the browser's native event. It has the same API as the browser's native event, including stopPropagation() and `preventDefault(), except the events work identically across all browsers.

## [20. What is the use of refs?](https://mokkapps.de/blog/my-top-react-interview-questions#_20-what-is-the-use-of-refs)

A [Ref](https://reactjs.org/docs/glossary.html#refs) is a special attribute that can be attached to any component. It can be an object created by React.createRef(), a callback function or a string (in legacy API).

To get direct access to a DOM element or component instance you can use ref attribute as a callback function. The function receives the underlying DOM element or class instance (depending on the type of element) as its argument.

In most cases, refs should be used sparingly.

Q1. What are some key benefits of using ReactJS?

This is a common advanced ReactJS interview question asked in technical interviews. Some key benefits of using ReactJS include:

ReactJS is the best framework for front-end development in Java as it is packed with a host of features.

ReactJS uses a Virtual DOM, thereby enhancing the performance of applications built using the framework.

React employs a component-based architecture and allows components to be reused with great ease.

React has a vast list of tools and libraries, providing impressive depth and flexibility.

Q2. What is the Virtual DOM in React?

Virtual DOM is a feature through which a virtual version of the real DOM is placed in the memory. Using the ReactDOM library, the virtual and real DOMs are synced whenever required.

Q3. Which are the different Lifecycle phases in ReactJS?

This is another important advanced ReactJS interview question asked in technical interviews. Components in ReactJS fundamentally go through three Lifecycle Phases.

Unmounting

Mounting

Updating

Q4. Which are the various Lifecycle Methods in the Mounting Lifecycle Phase?

The various methods in the mounting Lifecycle Phase include:

Render

ComponentWillMount

ComponentDidMount

Q5. Which are the different Lifecycle methods in the Updating Phase?

The lifecycle methods in the Updating Phase include:

ComponentWillReceiveProps

Render

ComponentDidUpdate

ShouldComponentUpdate

ComponentWillUpdate

Q6. What is an Event in ReactJS?

Events are actions that are performed by the user or computer. Actions such as clicking and pressing keys are examples of Events in ReactJS. Unlike in HTML, where events are in lowercase, events in React are in CamelCase.

Q7. What are some prime features of the ReactJS framework?

This again is a frequently asked ReactJS interview question asked in technical interviews.

The prime features of the ReactJS framework include:

ReactJs allows for Data Binding, where developers can choose what elements users can access and what elements are hidden. The security of applications created is also foolproof because of this feature.

ReactJS has a unidirectional data flow.

ReactJs employs Virtual DOM instead of RealDOM.

React mainly uses Server-side rendering.

Q8. What is Redux?

Redux is a popularly-used library for front-end development. It is loaded with a spate of features that help accomplish advanced front-end tasks using UI components. Applications built using Redux are easy to test and can seamlessly run on multiple framework environments.

Q9. What are some advantages of using Flux?

Flux offers a bunch of advantages, the most important ones being:

It ensures uniformity of code execution, thereby preventing Runtime errors.

It helps to perform updates more effectively through unidirectional data flow, thereby improving the efficiency of the application.

Q10. What are some functions that can be performed using HOC?

Using HOC, you can -

Render Hijacking

Manipulate and Reuse code

Manipulate Props

Take a look at some ReactJS Interview Questions and Answers here.

Sample Advanced ReactJS Interview Questions for Practice

In this section, we’ll look at some additional ReactJS interview questions that you can practice for your upcoming interview.

Why do we use Keys in ReactJS lists?

How can the transversal of lists be achieved in ReactJS?

What do you understand about components in React?

Explain the two types of Components in ReactJS?

What are Synthetic events in React?

What are Pure Components in ReactJS?

What are some important features of Redux?

What do you understand about Flux in React?

Why are Routers used in ReactJS?

What are some disadvantages associated with the ReactJS framework?

Why can’t browsers read JSX? How can browsers be made to read JSX?

What do you understand about Higher-Order Components in ReactJS?

What do you understand by References in ReactJS?

What are Props in ReactJS?

How do you modularize code in the ReactJS framework?

What is the use of Webpack in React?

What is Babel in ReactJS?

What is Prop-drilling in ReactJS?

What are Error boundaries in ReactJS?

What do you understand about the Strict mode in React?

These above advanced ReactJS interview questions will help you understand the type of questions asked in your Java developer interview, especially if you are a front-end developer.

FAQs on Advanced ReactJS Interview Questions

Q1. Do you need to be good with ReactJS for backend developer roles?

As ReactJs is primarily an open-source front-end framework used for building interface applications, you don’t need to know your way around ReactJS if you’re a backend developer.

Q2. What main concepts should you prepare to answer advanced ReactJS interview questions?

Advanced ReactJS interview questions are mostly around Redux, Components, Flux, Synthetic Events, Virtual DOMs, and Security.

Q4. What is Lazy Loading in ReactJS?

Lazy Loading is a feature where buckets of code are bundled together and deployed simultaneously after each block of code is executed.

Q5. What is the use of JSX in React?

JSX in React is a JavaScript XML file that helps write HTML code in the React framework.

### 1. What is the difference between Virtual DOM and Real DOM?

| Virtual DOM | Real DOM |
| --- | --- |
| Changes can be made easily | Changes can be expensive |
| Minimal memory wastage | High demand for memory and more wastage |
| JSX element is updated if the element exists | Creates a new DOM every time an element gets updated |
| Cannot update HTML directly | Able to directly manipulate HTML |
| Faster updates | Slow updates |

### 2. What is React?

React is a widely used JavaScript library that was launched in 2011. It was created by developers at Facebook, and it is primarily used for front-end development. React uses the component-based approach, which ensures that you build components that possess high reusability.

React is well known for developing and designing complex mobile user interfaces and web applications.

### 3. What is the meaning of Virtual DOM?

A virtual DOM is a simple JavaScript object that is the exact copy of the corresponding real DOM. It can be considered a node tree that consists of elements, their attributes, and other properties. Using the [render() in React](https://intellipaat.com/blog/what-is-render-in-reactjs/), it creates a node tree and updates it based on the changes that occur in the data model. These changes are usually triggered by users or the actions caused by the system.

Next up among these React interview questions, you need to take a look at some of the important features that it offers.

### 4. What are some of the important features of React?

React has multiple [features](https://intellipaat.com/blog/react-js-features/) that are used for unique purposes. The important ones are as mentioned below:

* React makes use of a single-direction data flow model.
* It deals with complete server-side data processing and handling.
* React uses a Virtual DOM, which has many advantages of its own.

### 5. What is the meaning of JSX?

JSX is the abbreviation for JavaScript XML. It is a file that is used in React to bring out the essence of JavaScript to React and use it for its advantages.

It even includes bringing out HTML and the easy syntax of JavaScript. This ensures that the resulting HTML file will have high readability, thereby relatively increasing the performance of the application.

Consider the following example of a JSX:

render(){

return(

<div>

<h1> Hello Intellipaat learners!</h1>

</div>

);

}

### 6. Can browsers read a JSX file?

No, browsers cannot read JSX files directly. It can only read the objects provided by JavaScript. Now, to make a browser read a JSX file, it has to be transformed to a JavaScript object using JSX transformers, and only then it can be fed into the browser for further use in the pipeline.

### 7. Why is React widely used today?

React provides users with an ample number of advantages when building an application. Some of them are as follows:

* With React, UI testing becomes very easy.
* React can integrate with Angular and other frameworks easily.
* The high readability index ensures easy understanding.
* React can be used for both client-side and server-side requirements.
* It boosts application performance and overall efficiency.

### 8. Are there any disadvantages to using React?

There are some limitations when using React as mentioned below:

* Writing code is complicated as it uses JSX and inline template formatting.
* Beginners might find it tough to cope with its syntaxes and methods.
* The library contains a huge repository of information, which might be overwhelming.
* React is a simple library and not a complete framework hence calls for dependencies.

### 9. Differentiate between Angular and React.

The difference between [Angular and React](https://intellipaat.com/blog/angular-vs-react/) is as follows:

| Comparison Factor | Angular | React |
| --- | --- | --- |
| Created by | Google | Facebook |
| DOM | Real DOM | Virtual DOM |
| Render Support | Client-side | Server-side |
| Architecture | Full MVC support | Only the view aspect of MVC |
| Data Binding | Two-way binding | Unidirectional binding |

*Go through our* [*ReactJS vs React Native*](https://intellipaat.com/blog/reactjs-vs-react-native-difference/) *blog and get your confusion clear.*

### 10. What is the meaning of the component-based architecture of React?

In React, components are foundations used to build user interfaces for applications. With the component-based system in place, all of the individual entities become completely reusable and independent of each other. This means that rendering the application is easy and not dependent on the other components of the UI.

### 11. How does rendering work in React?

Rendering is an important aspect of React, as every single component must be rendered. This is done using the render() function. Once the function is called, it returns an element that represents a DOM component.

It is also possible to render more than one HTML element at a time by enclosing the HTML tags and passing them through the render function.

### 12. What are states in React?

States form is one of the vital aspects of React. It is considered a source of data or objects that control aspects such as component behavior and rendering. In React, states are used to easily create dynamic and interactive components.

*Check out our blog on* [*ReactJS Tutorial*](https://intellipaat.com/blog/reactjs-tutorial/) *to learn more about ReactJS.*

### 13. What are props in React?

Props are the shorthand name given to properties in React. Props are read-only components that are immutable in nature. In an application, props follow a hierarchy that is passed down from parent to child components. However, the reverse is not supported. This is done to ensure that there is only a single directional flow of data at all times.

### 14. What is the use of an arrow function in React?

An arrow function is used to write an expression in React. It allows users to manually bind components easily. The functionality of arrow functions can be very useful when you are working with higher-order functions, in particular.

Consider the following example:

//The usual way

render() {

return(

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

);

}

//Making use of the arrow function

render() {

return(

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

);

}

### 15. What is a higher-order component in React?

Higher-order components (HOCs) are a widely used technique in React for applying concepts that involve component reusability logic. They are not a [React Native Firebase](https://intellipaat.com/blog/react-native-firebase/) part of the React API and allow users to easily reuse the code and bootstrap abstraction.

HOCs are also used to allow the simple sharing of behaviors across all of the components in React, adding more efficiency and functionality to the application.

### 16. What is the meaning of create-react-app in React?

The create-react-app in React is a simple command-line interface (CLI) that is used in the creation of React applications, which have no build configuration.

All tools are pre-configured when using the CLI, and this allows users to focus on the code more than on dependencies to develop the application.

The following syntax is used to start a simple project in React:

Create-react-app my-app

*Enroll in Intellipaat’s* [*Full Stack Developer Certification*](https://intellipaat.com/full-stack-web-developer-mean-stack-certification-training/) *to master front-end and back-end technologies.*

### 17. What are some of the advantages of using create-react-app in React?

Making use of create-react-app is advantageous in the following way:

* Support for JSX, ES6, and flow statements
* Already built and ready auto-prefixed CSS
* Fast interactive testing components
* Live development servers that help in debugging
* Scripts to handle JSS, CSS, and other files

Next up on these React Redux interview questions, you need to understand the meaning of Redux.

### 18. What is the meaning of Redux?

Redux is used to store the state of the application in a single entity. This simple entity is usually a JavaScript object. Changing states can be done by pushing out actions from the application and writing corresponding objects for them that are used to modify the states.

For example:

{

first\_name: ‘John’,

last\_name : ‘Kelly’,

age: 25

}

All of the data is retained by Redux (also called a store).

### 19. What is the difference between Props and States?

| Condition | Props | States |
| --- | --- | --- |
| Changes in child components | Yes | No |
| Parent component changing values | Yes | No |
| Changes inside components | No | Yes |

## Intermediate React Interview Questions

### 20. What are the three phases of a component life cycle in React?

The following are the three phases of a component’s life cycle:

* Initial rendering: This is the phase that involves the beginning of the journey of the component to the DOM.
* Update: Here, the component can be updated and rendered again if required after it gets added to the DOM.
* Unmounting: The final phase involves the destruction of the component and its eventual removal from the DOM.

### 21. What are events in React?

Whenever there are actions performed in React, such as hovering the mouse or pressing a key on the keyboard, these actions trigger events. Events then perform set activities as a response to these triggers. Handling an event in React is very similar to that in the DOM architecture.

### 22. How are events created in React?

Events can be created very easily in React as shown here:

class Display extends React.Component({

show(evt) {

// Code inside

},

render() {

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

return (

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

);

}

});

### 23. How is routing in React different from Conventional routing?

Differences between conventional routing and the [routing in React](https://intellipaat.com/blog/routing-in-react-js/) can be shown using the following aspects:

* Pages: Each view is considered as a new file in conventional routing while it is considered as a single HTML entity in React.
* Navigation: In conventional routing, users have to move across web pages for viewing. In React, the views are not refreshed as objects are re-issued to create new views.

### 24. Differentiate between Flux and Redux in React.

| Comparison Factor | Flux | Redux |
| --- | --- | --- |
| Components | Components connected to Flux in React | Container components directly connect |
| Dispatcher | Has a dispatcher | No dispatcher |
| Number of Stores | Single store | Multiple stores |
| State | Mutable state | Immutable state |
| Storage | Contains state and logic | State and logic are separate |

### 25. Can AJAX be used with React?

Yes, any AJAX library, such as [Axios](https://intellipaat.com/blog/axios-in-react/) and jQuery AJAX, can be used with React easily. One important thing is to maintain the states of the components, and here too, the props are passed from the parents to the child components.

Child components still cannot send back props to parents, and this factor greatly increases rendering efficiency when dynamic data is considered.

*If you are looking forward to becoming proficient in Angular.js, then make sure to check out Intellipaat’s latest offerings for the* [*Angular JS Course*](https://intellipaat.com/angular-training/)*.*

### 26. What is the meaning of synthetic events in React?

Synthetic events in React are objects that act as cross-browser wrappers, allowing for the use of native events. This is done to ensure that a variety of browsers can run the API and that the event contains all properties.

### 27. What are stateful components in React?

Stateful components are entities that store the changes that happen and place them into the memory. Here, the state can be changed, alongside storing information such as past, current, and future changes.

### 28. What are refs in React?

‘Refs’ is short for references in React. Refs are used to store a reference to a single React element or React component. This is later returned using the render function.

They are mainly used in the following scenarios:

* To initiate imperative animations
* To join third-party DOM libraries
* To manage focus and apply media playback

### 29. What are controlled components in React?

Controlled components in React refer to the components that have the ability to maintain their state. The data is completely controlled by the parent component, and the current value is fetched by making use of props. This is done to notify users of any changes that occur when using callbacks.

### 30. Why is a router required in React?

A router is very much necessary in React as it is used to manage multiple routes whenever a user types in a URL. If the route is present in the router for that corresponding URL, then the user is taken to the particular route.

To do this, the router library needs to be added in React. It can be done using the following syntax:

<switch>

<route exact path=’/’ component={Home}/>

<route path=’/posts/:id’ component={Newpost}/>

<route path=’/posts’ component={Post}/>

</switch>

### 31. What are the components of Redux in React?

Redux consists of four main components as shown below:

* Action: An object that describes the call
* Reducer: The state change storage unit
* Store: The state and object tree storage
* View: Displays data provided by the store

*Pursuing your career in Web Development? Here is the professional* [*React JS Course in Bangalore*](https://intellipaat.com/react-js-training-bangalore/) *provided by Intellipaat.*

### 32. What are the advantages of using Redux?

There are many advantages of Redux, and some of them are as given below:

| Organized Approach | Redux requires code to be organized, thereby making it consistent and easy to work with |
| --- | --- |
| Testing Ability | Redux functions are small and isolated, making the code more independent and testable |
| Tools | Developers can track actions and all of the tools in React using Redux easily |
| Community | Redux has a larger community, helping users with efficient and easy-to-use libraries |

### 33. What are the disadvantages of using MVC in React?

Among a plethora of advantages of using MVC in React, there are minor problems as stated below:

* A lot of memory waste occurs
* DOM manipulation costs a lot.
* The application becomes slow.
* Lots of dependencies are created.
* The complexity of models increases.

Next up among these ReactJS interview questions is understanding pure components.

### 34. What are pure components in React?

Pure components are singular entities that are written in React. They are fast and simple to write and have the ability to replace a component that has only the render() function. This is done to ensure that the performance of the application is good and that the code is kept simple at the same time.

### 35. What are higher-order components (HOCs) used for?

HOCs are used for a variety of tasks such as:

* Manipulation of props
* State manipulation and abstraction
* Render highjacking
* Code reusing
* Bootstrap abstraction

### 36. What are the keys in React?

Keys are used in React to check all items and to track changes actively. They are used to directly check if an item has been added or removed from a list.

Consider the following syntax:

function List ({ todos }) {

return (

<ul>

{todos.map(({ task, id} ) => <li key={id}>{task}</li>}

</ul>

)

}

### 37. Differentiate between a controlled component and an uncontrolled component in React.

A controlled component, as the name suggests, is a component over which React has complete control. It is the singular point of data for the forms.

An uncontrolled component is one where the form data gets handled by DOM and not the React component. This is usually done using refs in React.

*To learn more about React JS, Enroll now in our* [*React certification course*](https://intellipaat.com/react-js-certification-training-course/) *to gain in-depth knowledge.*

### 38. How can you tell React to build in production mode?

React can be coded to directly build into production by setting the process.env.NODE\_ENV variable to production.

Note: When React is in production, warnings, and other development features are not shown.

### 39. What is the difference between cloneElement and createElement in React?

In React, cloneElement is primarily used to clone an element and pass it to new props directly. Whereas, createElement is the entity that JSX gets compiled into. This is also used to create [elements in React](https://intellipaat.com/blog/react-native-elements/).

Next up on this top React interview questions and answers blog, take a look at the use of the second argument.

### 40. What is the use of the second argument that is passed to setState? Is it optional?

When setState is finished, a callback function is invoked, and the components get re-rendered in React.

Yes, the setState function in programming allows for an optional argument, typically a callback function, since it operates asynchronously. However, it is generally considered best practice to utilize alternative lifecycle methods instead of relying solely on this approach.

### 41. Is there a way to avoid the requirement of binding when using React?

Yes, there are two main ways you can use to avoid the need for binding. They are as follows:

* Defining the Event Handler as an Inline Arrow function:

class SubmitButton extends React.Component {

constructor(props) {

super(props);

this.state = {

isFormSubmitted: false

};

}

render() {

return (

<button onClick={() => {

this.setState({ isFormSubmitted: true });

}}>Submit</button>

)

}

}

* Using a function component along with Hooks:

const SubmitButton = () => {

const [isFormSubmitted, setIsFormSubmitted] = useState(false);

return (

<button onClick={() => {

setIsFormSubmitted(true);

}}>Submit</button>

)

};

Also, the Event Handler can be defined as an Arrow function, which is eventually assigned to a Class Field to obtain similar results.

### 42. What is the StrictMode component used in React?

The StrictMode component, when used, would benefit users immensely while creating new codebases to understand the components being used.

However, it can fit well in debugging as well because it will help solve the problem faster when it is wrapped with other components, which could be causing the problem.

Next up on these interview questions on React JS, you have to understand how to speed up rendering.

### 43. What would you do if your React application is rendering slowly?

The cause of slow rendering in React is mostly because of the number of re-render operations, which are sometimes unnecessary. There are two main tools provided by it to help users here:

* memo(): This is used to prevent all of the unnecessary re-rendering carried out by the function components.
* PureComponent: This is used to ensure that the unnecessary re-rendering of class components is avoided.

### 44. Can you conditionally add attributes to components in React?

Yes, there is a way in which you can add attributes to a React component when certain conditions are met.

React has the ability to omit an attribute if the value passed to it is not true.

Consider the following example:

var condition = true;

var component = (

<div

value="foo"

{ ...( condition && { disabled: true } ) } />

);

### 45. Why are props passed to the super() function in React?

Props gets passed onto the super() function if a user wishes to access this.props in the constructor.

Consider the following example:

class MyComponent extends React.Component {

constructor(props) {

super(props)

console.log(this.props)

// -> { icon: 'home', … }

}

}

### 46. What is the difference between using getInitialState and constructors in React?

When using ES6, users must initialize the state in the constructor and the getInitialState method is defined. This is done using React.createClass as shown in the below example:

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { /\* initial state \*/ };

}

}

This above piece of code is equivalent to the following:

var MyComponent = React.createClass({

getInitialState() {

return { /\* initial state \*/ };

},

});

### 47. What are the predefined prop types present in React?

There are five main predefined prop types in React. They are as follows:

1. PropTypes.bool
2. PropTypes.func
3. PropTypes.node
4. PropTypes.number
5. PropTypes.string

The propTypes can be defined for the user component as shown below:

import PropTypes from 'prop-types';

class User extends React.Component {

render() {

return (

<h1>Welcome, {this.props.name}</h1>

<h2>Age, {this.props.age}

);

}

}

User.propTypes = {

name: PropTypes.string.isRequired,

age: PropTypes.number.isRequired

};

### 48. What is React Fiber?

React Fiber is a new engine in React. It is the core implementation core algorithm in React 16.

The main goal of React Fiber is to ensure that there are incremental rendering facilities for the virtual DOM. This increases efficiency when rendering animations, gestures, etc., and also helps in assigning priority to updates based on the requirement, thereby increasing overall efficiency.

### 49. What are Hooks in React?

Hooks are used to make use of the state and other features without having to explicitly write a class. Hooks were added to the React version, v16.8. The stateful logic can be easily extracted from a component, along with testing and reusing it. All of this is done without making any changes to the component hierarchy.

==================

https://github.com/sudheerj/reactjs-interview-questions#what-is-the-difference-between-shadow-dom-and-virtual-dom

What is React?

React(aka React.js or ReactJS) is an open-source front-end JavaScript library that is used for building composable user interfaces, especially for single-page applications. It is used for handling view layer for web and mobile apps based on components in a declarative approach.

React was created by Jordan Walke, a software engineer working for Facebook. React was first deployed on Facebook's News Feed in 2011 and on Instagram in 2012.

What is the history behind React evolution?

The history of ReactJS started in 2010 with the creation of XHP. XHP is a PHP extension which improved the syntax of the language such that XML document fragments become valid PHP expressions and the primary purpose was used to create custom and reusable HTML elements.

The main principle of this extension was to make front-end code easier to understand and to help avoid cross-site scripting attacks. The project was successful to prevent the malicious content submitted by the scrubbing user.

But there was a different problem with XHP in which dynamic web applications require many roundtrips to the server, and XHP did not solve this problem. Also, the whole UI was re-rendered for small change in the application. Later, the initial prototype of React is created with the name FaxJ by Jordan inspired from XHP. Finally after sometime React has been introduced as a new library into JavaScript world.

Note: JSX comes from the idea of XHP

What are the major features of React?

The major features of React are:

* Uses JSX syntax, a syntax extension of JS that allows developers to write HTML in their JS code.
* It uses Virtual DOM instead of Real DOM considering that Real DOM manipulations are expensive.
* Supports server-side rendering which is useful for Search Engine Optimizations(SEO).
* Follows Unidirectional or one-way data flow or data binding.
* Uses reusable/composable UI components to develop the view.

What is JSX?

JSX stands for JavaScript XML and it is an XML-like syntax extension to ECMAScript. Basically it just provides the syntactic sugar for the React.createElement(type, props, ...children) function, giving us expressiveness of JavaScript along with HTML like template syntax.

In the example below, the text inside <h1> tag is returned as JavaScript function to the render function.

export default function App() {

return ( <h1 className="greeting">{"Hello, this is a JSX Code!"}</h1> );

}

If you don't use JSX syntax then the respective JavaScript code should be written as below,

import { createElement } from 'react';

export default function App() {

return createElement( 'h1', { className: 'greeting' }, 'Hello, this is a JSX Code!' );

}

Note: JSX is stricter than HTML

What is the difference between Element and Component?

An Element is a plain object describing what you want to appear on the screen in terms of the DOM nodes or other components. Elements can contain other Elements in their props. Creating a React element is cheap. Once an element is created, it cannot be mutated.

The JavaScript representation(Without JSX) of React Element would be as follows:

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

and this element can be simiplified using JSX

<div id="login-btn">Login</div>

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

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

Finally, this element renders to the DOM using ReactDOM.render().

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

const Button = ({ handleLogin }) => (

<div id={"login-btn"} onClick={handleLogin}> Login </div>

);

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

const Button = ({ handleLogin }) =>

React.createElement( "div", { id: "login-btn", onClick: handleLogin }, "Login" );

How to create components in React?

Components are the building blocks of creating User Interfaces(UI) in React. There are two possible ways to create a component.

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

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 a class component:

class Greeting extends React.Component {

render() { return <h1>{`Hello, ${this.props.message}`}</h1>; }

}

When to use a Class Component over a Function Component?

After the addition of Hooks(i.e. React 16.8 onwards) it is always recommended to use Function components over Class components in React. Because you could use state, lifecycle methods and other features that were only available in class component present in function component too.

But even there are two reasons to use Class components over Function components.

If you need a React functionality whose Function component equivalent is not present yet, like Error Boundaries.

In older versions, If the component needs state or lifecycle methods then you need to use class component.

Note: You can also use reusable react error boundary third-party component without writing any class. i.e, No need to use class components for Error boundaries.

The usage of Error boundaries from the above library is quite straight forward.

"use client";

import { ErrorBoundary } from "react-error-boundary";

<ErrorBoundary fallback={<div>Something went wrong</div>}>

<ExampleApplication />

</ErrorBoundary>

What are Pure Components?

Pure components are the components which render the same output for the same state and props. In function components, you can achieve these pure components through memoized React.memo() API wrapping around the component. This API prevents unnecessary re-renders by comparing the previous props and new props using shallow comparison. So it will be helpful for performance optimizations.

But at the same time, it won't compare the previous state with the current state because function component itself prevents the unnecessary rendering by default when you set the same state again.

The syntactic representation of memoized components looks like below,

const MemoizedComponent = memo(SomeComponent, arePropsEqual?);

Below is the example of how child component(i.e., EmployeeProfile) prevents re-renders for the same props passed by parent component(i.e.,EmployeeRegForm).

import { memo, useState } from 'react';

const EmployeeProfile = memo(function EmployeeProfile({ name, email }) {

return (<> <p>Name:{name}</p> <p>Email: {email}</p> </>);

});

export default function EmployeeRegForm() {

const [name, setName] = useState('');

const [email, setEmail] = useState('');

return (

<>

<label> Name: <input value={name} onChange={e => setName(e.target.value)} /> </label>

<label> Email: <input value={email} onChange={e => setEmail(e.target.value)} /> </label>

<hr/>

<EmployeeProfile name={name}/>

</>

);

}

In the above code, the email prop has not been passed to child component. So there won't be any re-renders for email prop change.

In class components, the components extending React.PureComponent instead of React.Component become the pure components. When props or state changes, PureComponent will do a shallow comparison on both props and state by invoking shouldComponentUpdate() lifecycle method.

Note: React.memo() is a higher-order component.

What is state in React?

State of a component is an object that holds some information that may change over the lifetime of the component. The important point is whenever the state object changes, the component re-renders. It is always recommended to make our state as simple as possible and minimize the number of stateful components.

state

Let's take an example of User component with message state. Here, useState hook has been used to add state to the User component and it returns an array with current state and function to update it.

import React, { useState } from "react";

function User() {

const [message, setMessage] = useState("Welcome to React world");

return ( <div> <h1>{message}</h1> </div> );

}

State is similar to props, but it is private and fully controlled by the component ,i.e., it is not accessible to any other component till the owner component decides to pass it.

What are props in React?

Props are inputs to components. They are single values or objects containing a set of values that are passed to components on creation similar to HTML-tag attributes. Here, the data is passed down from a parent component to a child component.

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

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

For example, let us create an element with reactProp property:

<Element reactProp={"1"} />

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

props.reactProp

For example, the usage of props in function component looks like below:

import React from "react";

import ReactDOM from "react-dom";

const ChildComponent = (props) => {

return ( <div> <p>{props.name}</p> <p>{props.age}</p> </div> );

};

const ParentComponent = () => {

return (

<div>

<ChildComponent name="John" age="30" /> <ChildComponent name="Mary" age="25" />

</div>

);

};

The properties from props object can be accessed directly using destructing feature from ES6 (ECMAScript 2015). The above child component can be simplified like below.

const ChildComponent = ({name, age}) => {

return ( <div> <p>{name}</p> <p>{age}</p> </div> );

};

What is the difference between state and props?

In React, both state and props are plain JavaScript objects and used to manage the data of a component, but they are used in different ways and have different characteristics. state is managed by the component itself and can be updated using the setState() function. Unlike props, state can be modified by the component and is used to manage the internal state of the component. Changes in the state trigger a re-render of the component and its children. props (short for "properties") are passed to a component by its parent component and are read-only, meaning that they cannot be modified by the component itself. props can be used to configure the behavior of a component and to pass data between components.

Why should we not update the state directly?

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

this.state.message = "Hello world"; //Wrong

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

this.setState({ message: "Hello World" }); //Correct

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

What is the purpose of callback function as an argument of setState()?

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

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

setState({ name: "John" }, () => console.log("The name has updated and component re-rendered") );

What is the difference between HTML and React event handling?

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

In HTML, the event name usually represents in lowercase as a convention:

<button onclick="activateLasers()"></button>

Whereas in React it follows camelCase convention:

<button onClick={activateLasers}>

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

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

Whereas in React you must call preventDefault() explicitly:

function handleClick(event) {

event.preventDefault(); console.log("The link was clicked.");

}

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

How to bind methods or event handlers in JSX callbacks?

There are 3 possible ways to achieve this in class components:

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

class User extends Component {

constructor(props) {

super(props); this.handleClick = this.handleClick.bind(this);

}

handleClick() { console.log("SingOut triggered"); }

render() { return <button onClick={this.handleClick}>SingOut</button>; }

}

Public class fields syntax: If you don't like to use bind approach then public class fields syntax can be used to correctly bind callbacks. The Create React App eanables this syntax by default.

handleClick = () => { console.log("SingOut triggered", this); };

<button onClick={this.handleClick}>SingOut</button>

Arrow functions in callbacks: It is possible to use arrow functions directly in the callbacks.

handleClick() { console.log('SingOut triggered'); }

render() {

return <button onClick={() => this.handleClick()}>SignOut</button>;

}

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

How to pass a parameter to an event handler or callback?

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

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

This is an equivalent to calling .bind:

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

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

<button onClick={this.handleClick(id)} />;

handleClick = (id) => () => { console.log("Hello, your ticket number is", id); };

What are synthetic events in React?

SyntheticEvent is a cross-browser wrapper around the browser's native event. Its API is same as the browser's native event, including stopPropagation() and preventDefault(), except the events work identically across all browsers. The native events can be accessed directly from synthetic events using nativeEvent attribute.

Let's take an example of BookStore title search component with the ability to get all native event properties

function BookStore() {

handleTitleChange(e) {

console.log('The new title is:', e.target.value);

const nativeEvent = e.nativeEvent; // 'e' represents synthetic event

console.log(nativeEvent);

e.stopPropogation();

e.preventDefault();

}

return <input name="title" onChange={handleTitleChange} />

}

What are inline conditional expressions?

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

<h1>Hello!</h1>;

{

messages.length > 0 && !isLogin ? ( <h2>You have {messages.length} unread messages.</h2> ) : (

<h2>You don't have unread messages.</h2>

);

}

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

A key is a special attribute you should include when mapping over arrays to render data. Key prop helps React identify which items have changed, are added, or are removed.

Keys should be unique among its siblings. 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*: 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.

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

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

The key attribute accepts either string or number and internally convert it as string type.

What is the use of refs?

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

How to create refs?

There are two approaches

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

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.myRef = React.createRef();

}

render() { return <div ref={this.myRef} />; }

}

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

class SearchBar extends Component {

constructor(props) {

super(props);

this.txtSearch = null;

this.state = { term: "" };

this.setInputSearchRef = (e) => {

this.txtSearch = e;

};

}

onInputChange(event) {

this.setState({ term: this.txtSearch.value });

}

render() {

return (

<input value={this.state.term} onChange={this.onInputChange.bind(this)}

ref={this.setInputSearchRef} />

);

}

}

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

What are forward refs?

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

const ButtonElement = React.forwardRef((props, ref) => (

<button ref={ref} className="CustomButton"> {props.children} </button>

));

// Create ref to the DOM button:

const ref = React.createRef();

<ButtonElement ref={ref}>{"Forward Ref"}</ButtonElement>;

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

It is preferred to use callback refs over findDOMNode() API. Because findDOMNode() prevents certain improvements in React in the future.

The legacy approach of using findDOMNode:

class MyComponent extends Component {

componentDidMount() {

findDOMNode(this).scrollIntoView();

}

render() { return <div />; }

}

The recommended approach is:

class MyComponent extends Component {

constructor(props) {

super(props);

this.node = createRef();

}

componentDidMount() {

this.node.current.scrollIntoView();

}

render() {

return <div ref={this.node} />;

}

}

Why are String Refs legacy?

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

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

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

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

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

class MyComponent extends Component {

renderRow = (index) => {

// This won't work. Ref will get attached to 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} />

);

}

}

What is Virtual DOM?

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

How Virtual DOM works?

The Virtual DOM works in three simple steps.

* Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.
* Then the difference between the previous DOM representation and the new one is calculated.
* Once the calculations are done, the real DOM will be updated with only the things that have actually changed.

What is the difference between Shadow DOM and Virtual DOM?

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

What is React Fiber?

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

What is the main goal of React Fiber?

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

Its main goals are:

* Ability to split interruptible work in chunks.
* Ability to prioritize, rebase and reuse work in progress.
* Ability to yield back and forth between parents and children to support layout in React.
* Ability to return multiple elements from render().
* Better support for error boundaries.

What are controlled components?

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

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

handleChange(event) {

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

}

What are uncontrolled components?

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

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

class UserProfile extends React.Component {

constructor(props) {

super(props);

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

this.input = React.createRef();

}

handleSubmit(event) {

alert("A name was submitted: " + this.input.current.value);

event.preventDefault();

}

render() {

return (

<form onSubmit={this.handleSubmit}>

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

<input type="submit" value="Submit" />

</form>

);

}

}

In most cases, it's recommend to use controlled components to implement forms. In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself.

What is the difference between createElement and cloneElement?

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

What is Lifting State Up in React?

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

What are the different phases of component lifecycle?

The component lifecycle has three distinct lifecycle phases:

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

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

Unmounting: In this last phase, the component is not needed and gets unmounted from the browser DOM. This phase includes componentWillUnmount() lifecycle method.

It's worth mentioning that React internally has a concept of phases when applying changes to the DOM. They are separated as follows

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

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

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

React 16.3+ Phases (or an interactive version)

phases 16.4+

Before React 16.3

phases 16.2

What are the lifecycle methods of React?

Before React 16.3

componentWillMount: Executed before rendering and is used for App level configuration in your root component.

componentDidMount: Executed after first rendering and here all AJAX requests, DOM or state updates, and set up event listeners should occur.

componentWillReceiveProps: Executed when particular prop updates to trigger state transitions.

shouldComponentUpdate: Determines if the component will be updated or not. By default it returns true. If you are sure that the component doesn't need to render after state or props are updated, you can return false value. It is a great place to improve performance as it allows you to prevent a re-render if component receives new prop.

componentWillUpdate: Executed before re-rendering the component when there are props & state changes confirmed by shouldComponentUpdate() which returns true.

componentDidUpdate: Mostly it is used to update the DOM in response to prop or state changes.

componentWillUnmount: It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component.

React 16.3+

getDerivedStateFromProps: Invoked right before calling render() and is invoked on every render. This exists for rare use cases where you need a derived state. Worth reading if you need derived state.

componentDidMount: Executed after first rendering and where all AJAX requests, DOM or state updates, and set up event listeners should occur.

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 the state or props are updated, you can return a false value. It is a great place to improve performance as it allows you to prevent a re-render if component receives a new prop.

getSnapshotBeforeUpdate: Executed right before rendered output is committed to the DOM. Any value returned by this will be passed into componentDidUpdate(). This is useful to capture information from the DOM i.e. scroll position.

componentDidUpdate: Mostly it is used to update the DOM in response to prop or state changes. This will not fire if shouldComponentUpdate() returns false.

componentWillUnmount It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component.

What are Higher-Order Components?

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

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

const EnhancedComponent = higherOrderComponent(WrappedComponent);

HOC can be used for many use cases:

* Code reuse, logic and bootstrap abstraction.
* Render hijacking.
* State abstraction and manipulation.
* Props manipulation.

How to create props proxy for HOC component?

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

function HOC(WrappedComponent) {

return class Test extends Component {

render() {

const newProps = {

title: "New Header",

footer: false,

showFeatureX: false,

showFeatureY: true,

};

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

}

};

}

What is context?

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

For example, authenticated users, locale preferences, UI themes need to be accessed in the application by many components.

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

What is children prop?

Children is a prop (this.props.children) that allows you to pass components as data to other components, just like any other prop you use. Component tree put between component's opening and closing tag will be passed to that component as children prop.

There are several methods available in the React API to work with this prop. These include React.Children.map, React.Children.forEach, React.Children.count, React.Children.only, React.Children.toArray.

A simple usage of children prop looks as below,

const MyDiv = React.createClass({

render: function () { return <div>{this.props.children}</div>; },

});

ReactDOM.render(

<MyDiv>

<span>{"Hello"}</span>

<span>{"World"}</span>

</MyDiv>,

node

);

How to write comments in React?

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

Single-line comments:

<div>

{/\* Single-line comments(In vanilla JavaScript, the single-line comments are represented by double slash(//)) \*/}

{`Welcome ${user}, let's play React`}

</div>

Multi-line comments:

<div>

{/\* Multi-line comments for more than

one line \*/}

{`Welcome ${user}, let's play React`}

</div>

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

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

Passing props:

class MyComponent extends React.Component {

constructor(props) {

super(props);

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

}

}

Not passing props:

class MyComponent extends React.Component {

constructor(props) {

super();

console.log(this.props); // prints undefined // but props parameter is still available

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

}

render() { // no difference outside constructor

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

}

}

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

What is reconciliation?

Reconciliation is the process through which React updates the Browser DOM and makes React work faster. React use a diffing algorithm so that component updates are predictable and faster. React would first calculate the difference between the real DOM and the copy of DOM (Virtual DOM) when there's an update of components. React stores a copy of Browser DOM which is called Virtual DOM. When we make changes or add data, React creates a new Virtual DOM and compares it with the previous one. This comparison is done by Diffing Algorithm. Now React compares the Virtual DOM with Real DOM. It finds out the changed nodes and updates only the changed nodes in Real DOM leaving the rest nodes as it is. This process is called Reconciliation.

How to set state with a dynamic key name?

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

handleInputChange(event) {

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

}

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

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

render() { // Wrong: handleClick is called instead of passed as a reference!

return <button onClick={this.handleClick()}>{'Click Me'}</button>

}

Instead, pass the function itself without parenthesis:

render() { // Correct: handleClick is passed as a reference!

return <button onClick={this.handleClick}>{'Click Me'}</button>

}

Is lazy function supports named exports?

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

// MoreComponents.js

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

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

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

// IntermediateComponent.js

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

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

import React, { lazy } from "react";

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

Why React uses className over class attribute?

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

render() {

return <span className={'menu navigation-menu'}>{'Menu'}</span>

}

What are fragments?

It's a common pattern or practice in React for a component to return multiple elements. Fragments let you group a list of children without adding extra nodes to the DOM. You need to use either or a shorter syntax having empty tag (<></>).

Below is the example of how to use fragment inside Story component.

function Story({title, description, date}) {

return (

<Fragment>

<h2>{title}</h2>

<p>{description}</p>

<p>{date}</p>

</Fragment>

);

}

It is also possible to render list of fragments inside a loop with the mandatory key attribute supplied.

function StoryBook() {

return stories.map(story =>

<Fragment key={ story.id}>

<h2>{story.title}</h2>

<p>{story.description}</p>

<p>{story.date}</p>

</Fragment>

);

}

Usually you don't need to use until unless there is a need of key attribute. The usage of shorter syntax looks like below.

function Story({title, description, date}) {

return (

<>

<h2>{title}</h2>

<p>{description}</p>

<p>{date}</p>

</>

);

}

Why fragments are better than container divs?

Below are the list of reasons to prefer fragments over container DOM elements,

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

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

The DOM Inspector is less cluttered.

What are portals in React?

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

ReactDOM.createPortal(child, container);

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

What are stateless components?

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

What are stateful components?

If the behavior of a component is dependent on the state of the component then it can be termed as stateful component. These stateful components are either function components with hooks or class components.

Let's take an example of function stateful component which update the state based on click event,

import React, {useState} from 'react';

const App = (props) => {

const [count, setCount] = useState(0);

handleIncrement() { setCount(count+1); }

return (

<>

<button onClick={handleIncrement}>Increment</button>

<span>Counter: {count}</span>

</>

)

}

How to apply validation on props in React?

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

The set of predefined prop types:

PropTypes.number PropTypes.string PropTypes.array PropTypes.object PropTypes.func PropTypes.node

PropTypes.element PropTypes.bool PropTypes.symbol 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.

The Equivalent Functional Component

import React from "react";

import PropTypes from "prop-types";

function User({ name, age }) {

return (

<>

<h1>{`Welcome, ${name}`}</h1>

<h2>{`Age, ${age}`}</h2>

</>

);

}

User.propTypes = { name: PropTypes.string.isRequired, age: PropTypes.number.isRequired,};

What are the advantages of React?

Below are the list of main advantages of React,

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

What are the limitations of React?

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

What are error boundaries in React v16?

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

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

class ErrorBoundary extends React.Component {

constructor(props) {

super(props);

this.state = { hasError: false };

}

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

logErrorToMyService(error, info);

}

static getDerivedStateFromError(error) { // Update state so the next render will show the fallback UI.

return { hasError: true };

}

render() {

if (this.state.hasError) { // You can render any custom fallback UI

return <h1>{"Something went wrong."}</h1>;

}

return this.props.children;

}

}

After that use it as a regular component:

<ErrorBoundary> <MyWidget /> </ErrorBoundary>

How are error boundaries handled in React v15?

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

What are the recommended ways for static type checking?

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

What is the use of react-dom package?

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

render()

hydrate()

unmountComponentAtNode()

findDOMNode()

createPortal()

What is the purpose of render method of react-dom?

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

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

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

What is ReactDOMServer?

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

renderToString()

renderToStaticMarkup()

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

// using Express

import { renderToString } from "react-dom/server";

import MyPage from "./MyPage";

app.get("/", (req, res) => {

res.write( "<!DOCTYPE html><html><head><title>My Page</title></head><body>" );

res.write('<div id="content">');

res.write(renderToString(<MyPage />));

res.write("</div></body></html>");

res.end();

});

How to use innerHTML in React?

The dangerouslySetInnerHTML attribute is React's replacement for using innerHTML in the browser DOM. Just like innerHTML, it is risky to use this attribute considering cross-site scripting (XSS) attacks. You just need to pass a \_\_html object as key and HTML text as value.

In this example MyComponent uses dangerouslySetInnerHTML attribute for setting HTML markup:

function createMarkup() {

return { \_\_html: "First &middot; Second" };

}

function MyComponent() {

return <div dangerouslySetInnerHTML={createMarkup()} />;

}

How to use styles in React?

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

| const divStyle = { color: "blue", backgroundImage: "url(" + imgUrl + ")", };  function HelloWorldComponent() { return <div style={divStyle}>Hello World!</div>; } |
| --- |

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

How events are different in React?

Handling events in React elements has some syntactic differences:

React event handlers are named using camelCase, rather than lowercase.

With JSX you pass a function as the event handler, rather than a string.

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

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

What is the impact of indexes as keys?

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

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

{

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

}

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

{

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

}

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

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

componentDidMount() {

axios.get(`api/todos`)

.then((result) => { this.setState({ messages: [...result.data] }) })

}

What will happen if you use props in initial state?

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

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

class MyComponent extends React.Component {

constructor(props) {

super(props);

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

}

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

}

Using props inside render method will update the value:

class MyComponent extends React.Component {

constructor(props) {

super(props); this.state = { record: [], };

}

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

}

How do you conditionally render components?

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

const MyComponent = ({ name, address }) => (

<div> <h2>{name}</h2> {address && <p>{address}</p>} </div>

);

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

const MyComponent = ({ name, address }) => (

<div>

<h2>{name}</h2> {address ? <p>{address}</p> : <p>{"Address is not available"}</p>}

</div>

);

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

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

For example,

const ComponentA = () => ( <ComponentB isDisplay={true} className={"componentStyle"} /> );

const ComponentB = ({ isDisplay, ...domProps }) => ( <div {...domProps}>{"ComponentB"}</div> );

How you use decorators in React?

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

@setTitle("Profile")

class Profile extends React.Component { //.... }

/\*

title is a string that will be set as a document title

WrappedComponent is what our decorator will receive when put directly above a component class as seen in the example above

\*/

const setTitle = (title) => (WrappedComponent) => {

return class extends React.Component {

componentDidMount() { document.title = title; }

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

};

};

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

How do you memoize a component?

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

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

import moize from "moize";

import Component from "./components/Component"; // this module exports a non-memoized component

const MemoizedFoo = moize.react(Component);

const Consumer = () => {

<div>

{"I will memoize the following entry:"} <MemoizedFoo />

</div>;

};

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

const MemoComponent = React.memo(function MemoComponent(props) {

/\* render using props \*/

});

OR;

export default React.memo(MyFunctionComponent);

How you implement Server Side Rendering or SSR?

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

import ReactDOMServer from "react-dom/server";

import App from "./App";

ReactDOMServer.renderToString(<App />);

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

How to enable production mode in React?

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

What is CRA and its benefits?

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

Let's create Todo App using CRA:

# Installation

$ npm install -g create-react-app

# Create new project

$ create-react-app todo-app

$ cd todo-app

# Build, test and run

$ npm run build

$ npm run test

$ npm start

It includes everything we need to build a React app:

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.

What is the lifecycle methods order in mounting?

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

constructor()

static getDerivedStateFromProps()

render()

componentDidMount()

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

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

componentWillMount()

componentWillReceiveProps()

componentWillUpdate()

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

What is the purpose of getDerivedStateFromProps() lifecycle method?

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

class MyComponent extends React.Component {

static getDerivedStateFromProps(props, state) { // ...

}

}

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

What is the purpose of getSnapshotBeforeUpdate() lifecycle method?

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

class MyComponent extends React.Component {

getSnapshotBeforeUpdate(prevProps, prevState) { // ...

}

}

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

Do Hooks replace render props and higher order components?

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

What is the recommended way for naming components?

It is recommended to name the component by reference instead of using displayName.

Using displayName for naming component:

export default React.createClass({ displayName: "TodoApp", // … });

The recommended approach:

export default class TodoApp extends React.Component { // … }

also

const TodoApp = () => { //… };

export default TodoApp;

What is the recommended ordering of methods in component class?

Recommended ordering of methods from mounting to render stage:

static methods

constructor()

getChildContext()

componentWillMount()

componentDidMount()

componentWillReceiveProps()

shouldComponentUpdate()

componentWillUpdate()

componentDidUpdate()

componentWillUnmount()

click handlers or event handlers like onClickSubmit() or onChangeDescription()

getter methods for render like getSelectReason() or getFooterContent()

optional render methods like renderNavigation() or renderProfilePicture()

render()

What is a switching component?

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

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

import HomePage from "./HomePage";

import AboutPage from "./AboutPage";

import ServicesPage from "./ServicesPage";

import ContactPage from "./ContactPage";

const PAGES = {

home: HomePage,

about: AboutPage,

services: ServicesPage,

contact: ContactPage,

};

const Page = (props) => {

const Handler = PAGES[props.page] || ContactPage;

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

};

// The keys of the PAGES object can be used in the prop types to catch dev-time errors.

Page.propTypes = {

page: PropTypes.oneOf(Object.keys(PAGES)).isRequired,

};

Why we need to pass a function to setState()?

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

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

// assuming this.state.count === 0

this.setState({ count: this.state.count + 1 });

this.setState({ count: this.state.count + 1 });

this.setState({ count: this.state.count + 1 });

// this.state.count === 1, not 3

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

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

count: prevState.count + props.increment,

}));

// this.state.count === 3 as expected

(OR)

Why function is preferred over object for setState()?

React may batch multiple setState() calls into a single update for performance. Because this.props and this.state may be updated asynchronously, you should not rely on their values for calculating the next state.

This counter example will fail to update as expected:

// Wrong

this.setState({

counter: this.state.counter + this.props.increment,

});

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

// Correct

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

What are React Mixins?

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

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

const PureRenderMixin = require("react-addons-pure-render-mixin");

const Button = React.createClass({

mixins: [PureRenderMixin], // ...

});

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

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

if (this.isMounted()) {

this.setState({...})

}

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

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

What are the Pointer Events supported in React?

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

The following event types are now available in React DOM:

onPointerDown

onPointerMove

onPointerUp

onPointerCancel

onGotPointerCapture

onLostPointerCapture

onPointerEnter

onPointerLeave

onPointerOver

onPointerOut

Why should component names start with capital letter?

If you are rendering your component using JSX, the name of that component has to begin with a capital letter otherwise React will throw an error as an unrecognized tag. This convention is because only HTML elements and SVG tags can begin with a lowercase letter.

class SomeComponent extends Component { // Code goes here }

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

class myComponent extends Component {

render() { return <div />; }

}

export default myComponent;

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

import MyComponent from "./myComponent";

What are the exceptions on React component naming?

The component names should start with an uppercase letter but there are few exceptions to 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)`

)

}

Are custom DOM attributes supported in React v16?

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

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

<div mycustomattribute={"something"} />

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

<div />

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

<div mycustomattribute="something" />

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

What is the difference between constructor and getInitialState?

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

Using ES6 classes:

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { /\* initial state \*/ };

}

}

Using React.createClass():

const MyComponent = React.createClass({

getInitialState() {

return { /\* initial state \*/ };

},

});

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

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

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

component.forceUpdate(callback);

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

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

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

Using super(props):

class MyComponent extends React.Component {

constructor(props) {

super(props);

console.log(this.props); // { name: 'John', ... }

}

}

Using super():

class MyComponent extends React.Component {

constructor(props) {

super(); console.log(this.props); // undefined

}

}

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

How to loop inside JSX?

You can simply use Array.prototype.map with ES6 arrow function syntax.

For example, the items array of objects is mapped into an array of components:

<tbody>

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

</tbody>

But you can't iterate using for loop:

<tbody>

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

</tbody>

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

How do you access props in attribute quotes?

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

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

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

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

Using template strings will also work:

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

What is React proptype array with shape?

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

ReactComponent.propTypes = {

arrayWithShape: React.PropTypes.arrayOf(

React.PropTypes.shape({

color: React.PropTypes.string.isRequired, fontSize: React.PropTypes.number.isRequired,

})

).isRequired,

};

How to conditionally apply class attributes?

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

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

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

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

Template strings will also work:

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

What is the difference between React and ReactDOM?

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

Why ReactDOM is separated from React?

The React team worked on extracting all DOM-related features into a separate library called ReactDOM. React v0.14 is the first release in which the libraries are split. By looking at some of the packages, react-native, react-art, react-canvas, and react-three, it has become clear that the beauty and essence of React has nothing to do with browsers or the DOM.

To build more environments that React can render to, React team planned to split the main React package into two: react and react-dom. This paves the way to writing components that can be shared between the web version of React and React Native.

How to use React label element?

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

<label for={'user'}>{'User'}</label>

<input type={'text'} id={'user'} />

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

<label htmlFor={'user'}>{'User'}</label>

<input type={'text'} id={'user'} />

How to combine multiple inline style objects?

You can use spread operator in regular React:

<button style={{ ...styles.panel.button, ...styles.panel.submitButton }}>

{"Submit"}

</button>

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

<button style={[styles.panel.button, styles.panel.submitButton]}>

{"Submit"}

</button>

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

You can use the useState hook to manage the width and height state variables, and the useEffect hook to add and remove the resize event listener. The [] dependency array passed to useEffect ensures that the effect only runs once (on mount) and not on every re-render.

import React, { useState, useEffect } from "react";

function WindowDimensions() {

const [dimensions, setDimensions] = useState({

width: window.innerWidth, height: window.innerHeight,

});

useEffect(() => {

function handleResize() {

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

}

window.addEventListener("resize", handleResize);

return () => window.removeEventListener("resize", handleResize);

}, []);

return ( <span> {dimensions.width} x {dimensions.height} </span> );

}

Using Class Component

What is the difference between setState() and replaceState() methods?

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

How to listen to state changes?

The componentDidUpdate lifecycle method will be called when state changes. You can compare provided state and props values with current state and props to determine if something meaningful changed.

componentDidUpdate(object prevProps, object prevState)

Note: The previous releases of ReactJS also uses componentWillUpdate(object nextProps, object nextState) for state changes. It has been deprecated in latest releases.

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

The better approach is to use Array.prototype.filter() method.

For example, let's create a removeItem() method for updating the state.

removeItem(index) {

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

}

Is it possible to use React without rendering HTML?

It is possible. Below are the possible options:

render() { return false }

render() { return true }

render() { return null }

React version >=16.0.0:

render() { return [] }

render() { return "" }

React version >=16.2.0:

render() { return <React.Fragment></React.Fragment>}

render() { return <></>}

React version >=18.0.0:

render() { return undefined}

How to pretty print JSON with React?

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

const data = { name: "John", age: 42 };

class User extends React.Component {

render() { return <pre>{JSON.stringify(data, null, 2)}</pre>; }

}

React.render(<User />, document.getElementById("container"));

Why you can't update props in React?

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

How to focus an input element on page load?

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

class App extends React.Component {

componentDidMount() { this.nameInput.focus(); }

render() {

return (

<div>

<input defaultValue={"Won't focus"} />

<input ref={(input) => (this.nameInput = input)} defaultValue={"Will focus"} />

</div>

);

}

}

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

Also in Functional component (react 16.08 and above)

import React, { useEffect, useRef } from "react";

const App = () => {

const inputElRef = useRef(null);

useEffect(() => { inputElRef.current.focus(); }, []);

return (

<div>

<input defaultValue={"Won't focus"} />

<input ref={inputElRef} defaultValue={"Will focus"} />

</div>

);

};

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

What are the possible ways of updating objects in state?

Calling setState() with an object to merge with state:

Using Object.assign() to create a copy of the object:

const user = Object.assign({}, this.state.user, { age: 42 });

this.setState({ user });

Using spread operator:

const user = { ...this.state.user, age: 42 };

this.setState({ user });

Calling setState() with a function:

this.setState((prevState) => ({

user: { ...prevState.user, age: 42, },

}));

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

You can use React.version to get the version.

const REACT\_VERSION = React.version;

ReactDOM.render(

<div>{`React version: ${REACT\_VERSION}`}</div>,

document.getElementById("app")

);

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

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

Manual import from core-js:

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

import "core-js/fn/array/find";

import "core-js/fn/array/includes";

import "core-js/fn/number/is-nan";

Using Polyfill service:

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

<script src="https://cdn.polyfill.io/v2/polyfill.min.js?features=default,Array.prototype.includes"></script>

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

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

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

"scripts": {

"start": "set HTTPS=true && react-scripts start"

}

or just run set HTTPS=true && npm start

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

Create a file called .env in the project root and write the import path:

NODE\_PATH=src/app

After that restart the development server. Now you should be able to import anything inside src/app without relative paths.

How to add Google Analytics for React Router?

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

history.listen(function (location) {

window.ga("set", "page", location.pathname + location.search);

window.ga("send", "pageview", location.pathname + location.search);

});

How to update a component every second?

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

componentDidMount() {

this.interval = setInterval(() => this.setState({ time: Date.now() }), 1000)

}

componentWillUnmount() { clearInterval(this.interval) }

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

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

<div

style={{

transform: "rotate(90deg)", WebkitTransform: "rotate(90deg)", // note the capital 'W' here

msTransform: "rotate(90deg)", // 'ms' is the only lowercase vendor prefix

}}

/>

How to import and export components using React and ES6?

You should use default for exporting the components

import React from "react";

import User from "user";

export default class MyProfile extends React.Component {

render() { return <User type="customer">//...</User>; }

}

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

Why is a component constructor called only once?

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

How to define constants in React?

You can use ES7 static field to define constant.

class MyComponent extends React.Component {

static DEFAULT\_PAGINATION = 10;

}

How to programmatically trigger click event in React?

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

This can be done in two steps:

Create ref in render method:

<input ref={(input) => (this.inputElement = input)} />

Apply click event in your event handler:

this.inputElement.click();

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

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

What are the common folder structures for React?

There are two common practices for React project file structure.

Grouping by features or routes:

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

common/

├─ Avatar.js

├─ Avatar.css

├─ APIUtils.js

└─ APIUtils.test.js

feed/

├─ index.js

├─ Feed.js

├─ Feed.css

├─ FeedStory.js

├─ FeedStory.test.js

└─ FeedAPI.js

profile/

├─ index.js

├─ Profile.js

├─ ProfileHeader.js

├─ ProfileHeader.css

└─ ProfileAPI.js

Grouping by file type:

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

api/

├─ APIUtils.js

├─ APIUtils.test.js

├─ ProfileAPI.js

└─ UserAPI.js

components/

├─ Avatar.js

├─ Avatar.css

├─ Feed.js

├─ Feed.css

├─ FeedStory.js

├─ FeedStory.test.js

├─ Profile.js

├─ ProfileHeader.js

└─ ProfileHeader.css

What are the popular packages for animation?

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

What is the benefit of styles modules?

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

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

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

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

And then imported individually in other components:

import { space, colors } from "./styles";

What are the popular React-specific linters?

ESLint is a popular JavaScript linter. There are plugins available that analyse specific code styles. One of the most common for React is an npm package called eslint-plugin-react. By default, it will check a number of best practices, with rules checking things from keys in iterators to a complete set of prop types.

Another popular plugin is eslint-plugin-jsx-a11y, which will help fix common issues with accessibility. As JSX offers slightly different syntax to regular HTML, issues with alt text and tabindex, for example, will not be picked up by regular plugins.

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

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

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

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { employees: [], error: null, };

}

componentDidMount() {

fetch("https://api.example.com/items")

.then((res) => res.json())

.then(

(result) => { this.setState({ employees: result.employees, }); },

(error) => { this.setState({ error }); }

);

}

render() {

const { error, employees } = this.state;

if (error) {

return <div>Error: {error.message}</div>;

} else {

return (

<ul>

{employees.map((employee) => (

<li key={employee.name}>

{employee.name}-{employee.experience}

</li>

))}

</ul>

);

}

}

}

What are render props?

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

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

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

What is React Router?

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

How React Router is different from history library?

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

What are the <Router> components of React Router v4?

React Router v4 provides below 3 <Router> components:

<BrowserRouter>

<HashRouter>

<MemoryRouter>

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

What is the purpose of push() and replace() methods of history?

A history instance has two methods for navigation purpose.

push()

replace()

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

How do you programmatically navigate using React Router v4?

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

Using the withRouter() higher-order function:

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

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

const Button = withRouter(({ history }) => (

<button type="button" onClick={() => { history.push("/new-location"); }} >

{"Click Me!"}

</button>

));

Using <Route> component and render props pattern:

The <Route> component passes the same props as withRouter(), so you will be able to access the history methods through the history prop.

import { Route } from "react-router-dom";

const Button = () => (

<Route

render={({ history }) => (

<button type="button" onClick={() => { history.push("/new-location"); }} >

{"Click Me!"}

</button>

)}

/>

);

Using context:

This option is not recommended and treated as unstable API.

const Button = (props, context) => (

<button type="button" onClick={() => { context.history.push("/new-location"); }} >

{"Click Me!"}

</button>

);

Button.contextTypes = {

history: React.PropTypes.shape({ push: React.PropTypes.func.isRequired, }),

};

How to get query parameters in React Router v4?

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

const queryString = require("query-string");

const parsed = queryString.parse(props.location.search);

You can also use URLSearchParams if you want something native:

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

const foo = params.get("name");

You should use a polyfill for IE11.

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

You have to wrap your Route's in a <Switch> block because <Switch> is unique in that it renders a route exclusively.

At first you need to add Switch to your imports:

import { Switch, Router, Route } from "react-router";

Then define the routes within <Switch> block:

<Router>

<Switch>

<Route {/\* ... \*/} />

<Route {/\* ... \*/} />

</Switch>

</Router>

How to pass params to history.push method in React Router v4?

While navigating you can pass props to the history object:

this.props.history.push({

pathname: "/template", search: "?name=sudheer", state: { detail: response.data },

});

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

How to implement default or NotFound page?

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

<Switch>

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

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

<Route component={NotFound} />

</Switch>

How to get history on React Router v4?

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

Create a module that exports a history object and import this module across the project.

For example, create history.js file:

import { createBrowserHistory } from "history";

export default createBrowserHistory({

/\* pass a configuration object here if needed \*/

});

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

import { Router } from "react-router-dom";

import history from "./history";

import App from "./App";

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

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

// some-other-file.js

import history from "./history";

history.push("/go-here");

How to perform automatic redirect after login?

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

import React, { Component } from "react";

import { Redirect } from "react-router";

export default class LoginComponent extends Component {

render() {

if (this.state.isLoggedIn === true) {

return <Redirect to="/your/redirect/page" />;

} else {

return <div>{"Login Please"}</div>;

}

}

}

What is React Intl?

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

What are the main features of React Intl?

* Display numbers with separators.
* Display dates and times correctly.
* Display dates relative to "now".
* Pluralize labels in strings.
* Support for 150+ languages.
* Runs in the browser and Node.
* Built on standards.

What are the two ways of formatting in React Intl?

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

Using react components:

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

Using an API:

const messages = defineMessages({

accountMessage: {

id: "account", defaultMessage: "The amount is less than minimum balance.",

},

});

formatMessage(messages.accountMessage);

How to use <FormattedMessage> as placeholder using React Intl?

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

import React from "react";

import { injectIntl, intlShape } from "react-intl";

const MyComponent = ({ intl }) => {

const placeholder = intl.formatMessage({ id: "messageId" });

return <input placeholder={placeholder} />;

};

MyComponent.propTypes = { intl: intlShape.isRequired, };

export default injectIntl(MyComponent);

How to access current locale with React Intl?

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

import { injectIntl, intlShape } from "react-intl";

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

MyComponent.propTypes = { intl: intlShape.isRequired, };

export default injectIntl(MyComponent);

How to format date using React Intl?

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

import { injectIntl, intlShape } from "react-intl";

const stringDate = this.props.intl.formatDate(date, {

year: "numeric",

month: "numeric",

day: "numeric",

});

const MyComponent = ({ intl }) => (

<div>{`The formatted date is ${stringDate}`}</div>

);

MyComponent.propTypes = { intl: intlShape.isRequired, };

export default injectIntl(MyComponent);

What is Shallow Renderer in React testing?

Shallow rendering is useful for writing unit test cases in React. It lets you render a component one level deep and assert facts about what its render method returns, without worrying about the behavior of child components, which are not instantiated or rendered.

For example, if you have the following component:

function MyComponent() {

return (

<div>

<span className={"heading"}>{"Title"}</span>

<span className={"description"}>{"Description"}</span>

</div>

);

}

Then you can assert as follows:

import ShallowRenderer from "react-test-renderer/shallow";

// in your test

const renderer = new ShallowRenderer();

renderer.render(<MyComponent />);

const result = renderer.getRenderOutput();

expect(result.type).toBe("div");

expect(result.props.children).toEqual([

<span className={"heading"}>{"Title"}</span>,

<span className={"description"}>{"Description"}</span>,

]);

What is TestRenderer package in React?

This package provides a renderer that can be used to render components to pure JavaScript objects, without depending on the DOM or a native mobile environment. This package makes it easy to grab a snapshot of the platform view hierarchy (similar to a DOM tree) rendered by a ReactDOM or React Native without using a browser or jsdom.

import TestRenderer from "react-test-renderer";

const Link = ({ page, children }) => <a href={page}>{children}</a>;

const testRenderer = TestRenderer.create(

<Link page={"https://www.facebook.com/"}>{"Facebook"}</Link>

);

console.log(testRenderer.toJSON());

// {

// type: 'a',

// props: { href: 'https://www.facebook.com/' },

// children: [ 'Facebook' ]

// }

What is the purpose of ReactTestUtils package?

ReactTestUtils are provided in the with-addons package and allow you to perform actions against a simulated DOM for the purpose of unit testing.

What is Jest?

Jest is a JavaScript unit testing framework created by Facebook based on Jasmine and provides automated mock creation and a jsdom environment. It's often used for testing components.

What are the advantages of Jest over Jasmine?

There are couple of advantages compared to Jasmine:

* Automatically finds tests to execute in your source code.
* Automatically mocks dependencies when running your tests.
* Allows you to test asynchronous code synchronously.
* Runs your tests with a fake DOM implementation (via jsdom) so that your tests can be run on the command line.
* Runs tests in parallel processes so that they finish sooner.

Give a simple example of Jest test case

Let's write a test for a function that adds two numbers in sum.js file:

const sum = (a, b) => a + b;

export default sum;

Create a file named sum.test.js which contains actual test:

import sum from "./sum";

test("adds 1 + 2 to equal 3", () => { expect(sum(1, 2)).toBe(3); });

And then add the following section to your package.json:

{ "scripts": { "test": "jest" } }

Finally, run yarn test or npm test and Jest will print a result:

$ yarn test

PASS ./sum.test.js

✓ adds 1 + 2 to equal 3 (2ms)

What is flux?

Flux is an application design paradigm used as a replacement for the more traditional MVC pattern. It is not a framework or a library but a new kind of architecture that complements React and the concept of Unidirectional Data Flow. Facebook uses this pattern internally when working with React.

What is Redux?

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

What are the core principles of Redux?

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.

What are the downsides of Redux compared to Flux?

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

You will need to learn to avoid mutations: Flux is un-opinionated about mutating data, but Redux doesn't like mutations and many packages complementary to Redux assume you never mutate the state. You can enforce this with dev-only packages like redux-immutable-state-invariant, Immutable.js, or instructing your team to write non-mutating code.

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.

What is the difference between mapStateToProps() and mapDispatchToProps()?

mapStateToProps() is a utility which helps your component get updated state (which is updated by some other components):

const mapStateToProps = (state) => {

return { todos: getVisibleTodos(state.todos, state.visibilityFilter), };

};

mapDispatchToProps() is a utility which will help your component to fire an action event (dispatching action which may cause change of application state):

const mapDispatchToProps = (dispatch) => {

return {

onTodoClick: (id) => { dispatch(toggleTodo(id)); },

};

};

It is recommended to always use the “object shorthand” form for the mapDispatchToProps.

Redux wraps 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,

};

Can I dispatch an action in reducer?

Dispatching an action within a reducer is an anti-pattern. Your reducer should be without side effects, simply digesting the action payload and returning a new state object. Adding listeners and dispatching actions within the reducer can lead to chained actions and other side effects.

How to access Redux store outside a component?

You just need to export the store from the module where it created with createStore(). Also, it shouldn't pollute the global window object.

store = createStore(myReducer);

export default store;

What are the drawbacks of MVW pattern?

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.

Are there any similarities between Redux and RxJS?

These libraries are very different for very different purposes, but there are some vague similarities.

Redux is a tool for managing state throughout the application. It is usually used as an architecture for UIs. Think of it as an alternative to (half of) Angular. RxJS is a reactive programming library. It is usually used as a tool to accomplish asynchronous tasks in JavaScript. Think of it as an alternative to Promises. Redux uses the Reactive paradigm because the Store is reactive. The Store observes actions from a distance, and changes itself. RxJS also uses the Reactive paradigm, but instead of being an architecture, it gives you basic building blocks, Observables, to accomplish this pattern.

How to dispatch an action on load?

You can dispatch an action in componentDidMount() method and in render() method you can verify the data.

class App extends Component {

componentDidMount() { this.props.fetchData(); }

render() {

return this.props.isLoaded ? ( <div>{"Loaded"}</div> ) : ( <div>{"Not Loaded"}</div> );

}

}

const mapStateToProps = (state) => ({ isLoaded: state.isLoaded, });

const mapDispatchToProps = { fetchData };

export default connect(mapStateToProps, mapDispatchToProps)(App);

How to use connect() from React Redux?

You need to follow two steps to use your store in your container:

Use mapStateToProps(): It maps the state variables from your store to the props that you specify.

Connect the above props to your container: The object returned by the mapStateToProps function is connected to the container. You can import connect() from react-redux.

import React from "react";

import { connect } from "react-redux";

class App extends React.Component {

render() { return <div>{this.props.containerData}</div>; }

}

function mapStateToProps(state) { return { containerData: state.data }; }

export default connect(mapStateToProps)(App);

How to reset state in Redux?

You need to write a root reducer in your application which delegate handling the action to the reducer generated by combineReducers().

For example, let us take rootReducer() to return the initial state after USER\_LOGOUT action. As we know, reducers are supposed to return the initial state when they are called with undefined as the first argument, no matter the action.

const appReducer = combineReducers({ /\* your app's top-level reducers \*/ });

const rootReducer = (state, action) => {

if (action.type === "USER\_LOGOUT") { state = undefined; }

return appReducer(state, action);

};

In case of using redux-persist, you may also need to clean your storage. redux-persist keeps a copy of your state in a storage engine. First, you need to import the appropriate storage engine and then, to parse the state before setting it to undefined and clean each storage state key.

const appReducer = combineReducers({ /\* your app's top-level reducers \*/ });

const rootReducer = (state, action) => {

if (action.type === "USER\_LOGOUT") {

Object.keys(state).forEach((key) => { storage.removeItem(`persist:${key}`); });

state = undefined;

}

return appReducer(state, action);

};

Whats the purpose of at symbol in the Redux connect decorator?

The @ symbol is in fact a JavaScript expression used to signify decorators. Decorators make it possible to annotate and modify classes and properties at design time.

Let's take an example setting up Redux without and with a decorator.

Without decorator:

import React from "react";

import \* as actionCreators from "./actionCreators";

import { bindActionCreators } from "redux";

import { connect } from "react-redux";

function mapStateToProps(state) {

return { todos: state.todos };

}

function mapDispatchToProps(dispatch) {

return { actions: bindActionCreators(actionCreators, dispatch) };

}

class MyApp extends React.Component {

// ...define your main app here

}

export default connect(mapStateToProps, mapDispatchToProps)(MyApp);

With decorator:

import React from "react";

import \* as actionCreators from "./actionCreators";

import { bindActionCreators } from "redux";

import { connect } from "react-redux";

function mapStateToProps(state) {

return { todos: state.todos };

}

function mapDispatchToProps(dispatch) {

return { actions: bindActionCreators(actionCreators, dispatch) };

}

@connect(mapStateToProps, mapDispatchToProps)

export default class MyApp extends React.Component {

// ...define your main app here

}

The above examples are almost similar except the usage of decorator. The decorator syntax isn't built into any JavaScript runtimes yet, and is still experimental and subject to change. You can use babel for the decorators support.

What is the difference between React context and React Redux?

You can use Context in your application directly and is going to be great for passing down data to deeply nested components which what it was designed for.

Whereas Redux is much more powerful and provides a large number of features that the Context API doesn't provide. Also, React Redux uses context internally but it doesn't expose this fact in the public API.

Why are Redux state functions called reducers?

Reducers always return the accumulation of the state (based on all previous and current actions). Therefore, they act as a reducer of state. Each time a Redux reducer is called, the state and action are passed as parameters. This state is then reduced (or accumulated) based on the action, and then the next state is returned. You could reduce a collection of actions and an initial state (of the store) on which to perform these actions to get the resulting final state.

How to make AJAX request in Redux?

You can use redux-thunk middleware which allows you to define async actions.

Let's take an example of fetching specific account as an AJAX call using fetch API:

export function fetchAccount(id) {

return (dispatch) => {

dispatch(setLoadingAccountState()); // Show a loading spinner

fetch(`/account/${id}`, (response) => {

dispatch(doneFetchingAccount()); // Hide loading spinner

if (response.status === 200) {

dispatch(setAccount(response.json)); // Use a normal function to set the received state

} else {

dispatch(someError);

}

});

};

}

function setAccount(data) {

return { type: "SET\_Account", data: data };

}

Should I keep all component's state in Redux store?

Keep your data in the Redux store, and the UI related state internally in the component.

What is the proper way to access Redux store?

The best way to access your store in a component is to use the connect() function, that creates a new component that wraps around your existing one. This pattern is called Higher-Order Components, and is generally the preferred way of extending a component's functionality in React. This allows you to map state and action creators to your component, and have them passed in automatically as your store updates.

Let's take an example of <FilterLink> component using connect:

import { connect } from "react-redux";

import { setVisibilityFilter } from "../actions";

import Link from "../components/Link";

const mapStateToProps = (state, ownProps) => ({

active: ownProps.filter === state.visibilityFilter,

});

const mapDispatchToProps = (dispatch, ownProps) => ({

onClick: () => dispatch(setVisibilityFilter(ownProps.filter)),

});

const FilterLink = connect(mapStateToProps, mapDispatchToProps)(Link);

export default FilterLink;

Due to it having quite a few performance optimizations and generally being less likely to cause bugs, the Redux developers almost always recommend using connect() over accessing the store directly (using context API).

class MyComponent {

someMethod() {

doSomethingWith(this.context.store);

}

}

What is the difference between component and container in React Redux?

Component is a class or function component that describes the presentational part of your application.

Container is an informal term for a component that is connected to a Redux store. Containers subscribe to Redux state updates and dispatch actions, and they usually don't render DOM elements; they delegate rendering to presentational child components.

What is the purpose of the constants in Redux?

Constants allows you to easily find all usages of that specific functionality across the project when you use an IDE. It also prevents you from introducing silly bugs caused by typos – in which case, you will get a ReferenceError immediately.

Normally we will save them in a single file (constants.js or actionTypes.js).

export const ADD\_TODO = "ADD\_TODO";

export const DELETE\_TODO = "DELETE\_TODO";

export const EDIT\_TODO = "EDIT\_TODO";

export const COMPLETE\_TODO = "COMPLETE\_TODO";

export const COMPLETE\_ALL = "COMPLETE\_ALL";

export const CLEAR\_COMPLETED = "CLEAR\_COMPLETED";

In Redux, you use them in two places:

During action creation:

Let's take actions.js:

import { ADD\_TODO } from "./actionTypes";

export function addTodo(text) {

return { type: ADD\_TODO, text };

}

In reducers:

Let's create reducer.js:

import { ADD\_TODO } from "./actionTypes";

export default (state = [], action) => {

switch (action.type) {

case ADD\_TODO:

return [ ...state, { text: action.text, completed: false, }, ];

default: return state;

}

};

What are the different ways to write mapDispatchToProps()?

There are a few ways of binding action creators to dispatch() in mapDispatchToProps().

Below are the possible options:

const mapDispatchToProps = (dispatch) => ({

action: () => dispatch(action()),

});

const mapDispatchToProps = (dispatch) => ({

action: bindActionCreators(action, dispatch),

});

const mapDispatchToProps = { action };

The third option is just a shorthand for the first one.

What is the use of the ownProps parameter in mapStateToProps() and mapDispatchToProps()?

If the ownProps parameter is specified, React Redux will pass the props that were passed to the component into your connect functions. So, if you use a connected component:

import ConnectedComponent from "./containers/ConnectedComponent";

<ConnectedComponent user={"john"} />;

The ownProps inside your mapStateToProps() and mapDispatchToProps() functions will be an object:

{

user: "john";

}

You can use this object to decide what to return from those functions.

How to structure Redux top level directories?

Most of the applications has several top-level directories as below:

Components: Used for dumb components unaware of Redux.

Containers: Used for smart components connected to Redux.

Actions: Used for all action creators, where file names correspond to part of the app.

Reducers: Used for all reducers, where files name correspond to state key.

Store: Used for store initialization.

This structure works well for small and medium size apps.

What is redux-saga?

redux-saga is a library that aims to make side effects (asynchronous things like data fetching and impure things like accessing the browser cache) in React/Redux applications easier and better.

It is available in NPM:

$ npm install --save redux-saga

What is the mental model of redux-saga?

Saga is like a separate thread in your application, that's solely responsible for side effects. redux-saga is a redux middleware, which means this thread can be started, paused and cancelled from the main application with normal Redux actions, it has access to the full Redux application state and it can dispatch Redux actions as well.

What are the differences between call() and put() in redux-saga?

Both call() and put() are effect creator functions. call() function is used to create effect description, which instructs middleware to call the promise. put() function creates an effect, which instructs middleware to dispatch an action to the store.

Let's take example of how these effects work for fetching particular user data.

function\* fetchUserSaga(action) {

// `call` function accepts rest arguments, which will be 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,

});

}

What is Redux Thunk?

Redux Thunk middleware allows you to write action creators that return a function instead of an action. The thunk can be used to delay the dispatch of an action, or to dispatch only if a certain condition is met. The inner function receives the store methods dispatch() and getState() as parameters.

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

Both Redux Thunk and Redux Saga take care of dealing with side effects. In most of the scenarios, Thunk uses Promises to deal with them, whereas Saga uses Generators. Thunk is simple to use and Promises are familiar to many developers, Sagas/Generators are more powerful but you will need to learn them. But both middleware can coexist, so you can start with Thunks and introduce Sagas when/if you need them.

What is Redux DevTools?

Redux DevTools is a live-editing time travel environment for Redux with hot reloading, action replay, and customizable UI. If you don't want to bother with installing Redux DevTools and integrating it into your project, consider using Redux DevTools Extension for Chrome and Firefox.

What are the features of Redux DevTools?

Some of the main features of Redux DevTools are below,

Lets you inspect every state and action payload.

Lets you go back in time by cancelling actions.

If you change the reducer code, each staged action will be re-evaluated.

If the reducers throw, you will see during which action this happened, and what the error was.

With persistState() store enhancer, you can persist debug sessions across page reloads.

What are Redux selectors and why to use them?

Selectors are functions that take Redux state as an argument and return some data to pass to the component.

For example, to get user details from the state:

const getUserData = (state) => state.user.data;

These selectors have two main benefits,

The selector can compute derived data, allowing Redux to store the minimal possible state

The selector is not recomputed unless one of its arguments changes

What is Redux Form?

Redux Form works with React and Redux to enable a form in React to use Redux to store all of its state. Redux Form can be used with raw HTML5 inputs, but it also works very well with common UI frameworks like Material UI, React Widgets and React Bootstrap.

What are the main features of Redux Form?

Some of the main features of Redux Form are:

Field values persistence via Redux store.

Validation (sync/async) and submission.

Formatting, parsing and normalization of field values.

How to add multiple middlewares to Redux?

You can use applyMiddleware().

For example, you can add redux-thunk and logger passing them as arguments to applyMiddleware():

import { createStore, applyMiddleware } from "redux";

const createStoreWithMiddleware = applyMiddleware(

ReduxThunk, logger)(createStore);

How to set initial state in Redux?

You need to pass initial state as second argument to createStore:

const rootReducer = combineReducers({

todos: todos, visibilityFilter: visibilityFilter,

});

const initialState = { todos: [{ id: 123, name: "example", completed: false }],};

const store = createStore(rootReducer, initialState);

How Relay is different from Redux?

Relay is similar to Redux in that they both use a single store. The main difference is that relay only manages state originated from the server, and all access to the state is used via GraphQL queries (for reading data) and mutations (for changing data). Relay caches the data for you and optimizes data fetching for you, by fetching only changed data and nothing more.

What is an action in Redux?

Actions are plain JavaScript objects or payloads of information that send data from your application to your store. They are the only source of information for the store. Actions must have a type property that indicates the type of action being performed.

For example, let's take an action which represents adding a new todo item:

{ type: ADD\_TODO, text: 'Add todo item' }

What is the difference between React Native and React?

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

React Native is a mobile framework that compiles to native app components, allowing you to build native mobile applications (iOS, Android, and Windows) in JavaScript that allows you to use React to build your components, and implements React under the hood.

How to test React Native apps?

React Native can be tested only in mobile simulators like iOS and Android. You can run the app in your mobile using expo app (https://expo.io) Where it syncs using QR code, your mobile and computer should be in same wireless network.

How to do logging in React Native?

You can use console.log, console.warn, etc. As of React Native v0.29 you can simply run the following to see logs in the console:

$ react-native log-ios

$ react-native log-android

How to debug your React Native?

* Run your application in the iOS simulator.
* Press Command + D and a webpage should open up at http://localhost:8081/debugger-ui.
* Enable Pause On Caught Exceptions for a better debugging experience.
* Press Command + Option + I to open the Chrome Developer tools, or open it via View -> Developer -> Developer Tools.
* You should now be able to debug as you normally would.
* React supported libraries & Integration

What is reselect and how it works?

Reselect is a selector library (for Redux) which uses memoization concept. It was originally written to compute derived data from Redux-like applications state, but it can't be tied to any architecture or library.

Reselect keeps a copy of the last inputs/outputs of the last call, and recomputes the result only if one of the inputs changes. If the the same inputs are provided twice in a row, Reselect returns the cached output. It's memoization and cache are fully customizable.

What is Flow?

Flow is a static type checker designed to find type errors in JavaScript. Flow types can express much more fine-grained distinctions than traditional type systems. For example, Flow helps you catch errors involving null, unlike most type systems.

What is the difference between Flow and PropTypes?

Flow is a static analysis tool (static checker) which uses a superset of the language, allowing you to add type annotations to all of your code and catch an entire class of bugs at compile time.

PropTypes is a basic type checker (runtime checker) which has been patched onto React. It can't check anything other than the types of the props being passed to a given component. If you want more flexible typechecking for your entire project Flow/TypeScript are appropriate choices.

How to use Font Awesome icons in React?

The below steps followed to include Font Awesome in React:

Install font-awesome:

$ npm install --save font-awesome

Import font-awesome in your index.js file:

import "font-awesome/css/font-awesome.min.css";

Add Font Awesome classes in className:

render() { return <div><i className={'fa fa-spinner'} /></div> }

What is React Dev Tools?

React Developer Tools let you inspect the component hierarchy, including component props and state. It exists both as a browser extension (for Chrome and Firefox), and as a standalone app (works with other environments including Safari, IE, and React Native).

The official extensions available for different browsers or environments.

Chrome extension

Firefox extension

Standalone app (Safari, React Native, etc)

Why is DevTools not loading in Chrome for local files?

If you opened a local HTML file in your browser (file://...) then you must first open Chrome Extensions and check Allow access to file URLs.

How to use Polymer in React?

You need to follow below steps to use Polymer in React,

Create a Polymer element:

<link rel="import" href="../../bower\_components/polymer/polymer.html" />;

Polymer({

is: "calendar-element", ready: function () { this.textContent = "I am a calendar"; },

});

Create the Polymer component HTML tag by importing it in a HTML document, e.g. import it in the index.html of your React application:

<link rel="import" href="./src/polymer-components/calendar-element.html"/>

Use that element in the JSX file:

import React from "react";

class MyComponent extends React.Component {

render() { return <calendar-element />; }

}

export default MyComponent;

What are the advantages of React over Vue.js?

* Gives more flexibility in large apps developing.
* Easier to test.
* Suitable for mobile apps creating.
* More information and solutions available.

Note: The above list of advantages are purely opinionated and it vary based on the professional experience. But they are helpful as base parameters.

Why React tab is not showing up in DevTools?

When the page loads, React DevTools sets a global named \_\_REACT\_DEVTOOLS\_GLOBAL\_HOOK\_\_, then React communicates with that hook during initialization. If the website is not using React or if React fails to communicate with DevTools then it won't show up the tab.

What are Styled Components?

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

Give an example of Styled Components?

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

import React from "react";

import styled from "styled-components";

// Create a <Title> component that renders an <h1> which is centered, red and sized at 1.5em

const Title = styled.h1` font-size: 1.5em; text-align: center; color: palevioletred;`;

// Create a <Wrapper> component that renders a <section> with some padding and a papayawhip background

const Wrapper = styled.section` padding: 4em; background: papayawhip;`;

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

<Wrapper>

<Title>{"Lets start first styled component!"}</Title>

</Wrapper>

What is Relay?

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

How to use TypeScript in create-react-app application?

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

npx create-react-app my-app --typescript

# or

yarn create react-app my-app --typescript

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

Now the project layout should look like the following:

my-app/

├─ .gitignore

├─ images.d.ts

├─ node\_modules/

├─ public/

├─ src/

│ └─ ...

├─ package.json

├─ tsconfig.json

├─ tsconfig.prod.json

├─ tsconfig.test.json

└─ tslint.json

Miscellaneous

What are the main features of Reselect library?

Let's see the main features of Reselect library,

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

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

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

Give an example of Reselect usage?

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

import { createSelector } from "reselect";

const shopItemsSelector = (state) => state.shop.items;

const taxPercentSelector = (state) => state.shop.taxPercent;

const subtotalSelector = createSelector(shopItemsSelector, (items) =>

items.reduce((acc, item) => acc + item.value, 0)

);

const taxSelector = createSelector(

subtotalSelector, taxPercentSelector, (subtotal, taxPercent) => subtotal \* (taxPercent / 100)

);

export const totalSelector = createSelector(

subtotalSelector, taxSelector, (subtotal, tax) => ({ total: subtotal + tax })

);

let exampleState = {

shop: {

taxPercent: 8,

items: [ { name: "apple", value: 1.2 }, { name: "orange", value: 0.95 }, ],

},

};

console.log(subtotalSelector(exampleState)); // 2.15

console.log(taxSelector(exampleState)); // 0.172

console.log(totalSelector(exampleState)); // { total: 2.322 }

Does the statics object work with ES6 classes in React?

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

someComponent = React.createClass({

statics: { someMethod: function () { // .. }, },

});

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

class Component extends React.Component {

static propTypes = { // … };

static someMethod() { // … }

}

or writing them outside class as below,

class Component extends React.Component {

....

}

Component.propTypes = {...}

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

Can Redux only be used with React?

Redux can be used as a data store for any UI layer. The most common usage is with React and React Native, but there are bindings available for Angular, Angular 2, Vue, Mithril, and more. Redux simply provides a subscription mechanism which can be used by any other code.

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

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

How Redux Form initialValues get updated from state?

You need to add enableReinitialize : true setting.

const InitializeFromStateForm = reduxForm({

form: "initializeFromState", enableReinitialize: true,

})(UserEdit);

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

How React PropTypes allow different types for one prop?

You can use oneOfType() method of PropTypes.

For example, the height property can be defined with either string or number type as below:

Component.propTypes = {

size: PropTypes.oneOfType([PropTypes.string, PropTypes.number]),

};

Can I import an SVG file as react component?

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

import { ReactComponent as Logo } from "./logo.svg";

const App = () => (

<div> {/\* Logo is an actual react component \*/} <Logo /> </div>

);

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

Why are inline ref callbacks or functions not recommended?

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

class UserForm extends Component {

handleSubmit = () => {

console.log("Input Value is: ", this.input.value);

};

render() {

return (

<form onSubmit={this.handleSubmit}>

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

Access DOM input in handle submit

<button type="submit">Submit</button>

</form>

);

}

}

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

class UserForm extends Component {

handleSubmit = () => {

console.log("Input Value is: ", this.input.value);

};

setSearchInput = (input) => {

this.input = input;

};

render() {

return (

<form onSubmit={this.handleSubmit}>

<input type="text" ref={this.setSearchInput} /> // Access DOM input

in handle submit

<button type="submit">Submit</button>

</form>

);

}

}

Note: In React v16.3,

What is render hijacking in react?

The concept of render hijacking is the ability to control what a component will output from another component. It 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 enable hijacking, but by using HOC you make your component behave differently.

What are HOC factory implementations?

There are two main ways of implementing HOCs in React.

Props Proxy (PP) and Inheritance Inversion (II).

But they follow different approaches for manipulating the WrappedComponent.

Props Proxy

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

function ppHOC(WrappedComponent) {

return class PP extends React.Component {

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

};

}

Inheritance Inversion

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

function iiHOC(WrappedComponent) {

return class Enhancer extends WrappedComponent {

render() { return super.render(); }

};

}

How to pass numbers to React component?

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

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

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

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

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

Do other parts of the application care about this data?

Do you need to be able to create further derived data based on this original data?

Is the same data being used to drive multiple components?

Is there value to you in being able to restore this state to a given point in time (ie, time travel debugging)?

Do you want to cache the data (i.e, use what's in state if it's already there instead of re-requesting it)?

What is the purpose of registerServiceWorker in React?

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

import React from "react";

import ReactDOM from "react-dom";

import App from "./App";

import registerServiceWorker from "./registerServiceWorker";

ReactDOM.render(<App />, document.getElementById("root"));

registerServiceWorker();

What is React memo function?

Class components can be restricted from re-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 \*/

});

What is React lazy function?

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

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

function MyComponent() {

return ( <div> <OtherComponent /> </div> );

}

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

How to prevent unnecessary updates using setState?

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

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

getUserProfile = (user) => {

const latestAddress = user.address;

this.setState((state) => {

if (state.address === latestAddress) { return null; }

else { return { title: latestAddress }; }

});

};

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

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

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

const ReactJSDevs = () => {

return [

<li key="1">John</li>,

<li key="2">Jackie</li>,

<li key="3">Jordan</li>,

];

};

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

const JSDevs = () => {

return (

<ul>

<li>Brad</li>

<li>Brodge</li>

<ReactJSDevs />

<li>Brandon</li>

</ul>

);

};

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

render() { return 'Welcome to ReactJS questions'; }

// Number

render() { return 2018; }

How to use class field declarations syntax in React classes?

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

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

class Counter extends Component {

state = { value: 0 };

handleIncrement = () => {

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

};

handleDecrement = () => {

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

};

render() {

return (

<div>

{this.state.value}

<button onClick={this.handleIncrement}>+</button>

<button onClick={this.handleDecrement}>-</button>

</div>

);

}

}

What are hooks?

Hooks is a special JavaScript function that allows you use state and other React features without writing a class. This pattern has been introduced as a new feature in React 16.8 and helped to isolate the stateful logic from the components.

Let's see an example of useState hook:

import { useState } from "react";

function Example() {

// Declare a new state variable, which we'll call "count"

const [count, setCount] = useState(0);

return (

<>

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

</>

);

}

Note: Hooks can be used inside an existing function component without rewriting the component.

What rules need to be followed for hooks?

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

Call Hooks only at the top level of your react functions: 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: You shouldn’t call Hooks from regular JavaScript functions. Instead, you should call them from either function components or custom hooks.

The eslint plugin named eslint-plugin-react-hooks can be used to enforce these two rules.

How to ensure hooks followed the rules in your project?

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

npm install eslint-plugin-react-hooks@next

And apply the below config in your ESLint config file,

// Your ESLint configuration

{

"plugins": [ // ...

"react-hooks"

],

"rules": { // ...

"react-hooks/rules-of-hooks": "error"

}

}

For example, the linter enforce proper naming convention for hooks. If you rename your custom hooks which as prefix "use" to something else then linter won't allow you to call built-in hooks such as useState, useEffect etc inside of your custom hook anymore.

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

What are the differences between Flux and Redux?

| Flux | Redux |
| --- | --- |
| State is mutable | State is immutable |
| The Store contains both state and change logic | The Store and change logic are separate |
| There are multiple stores exist | There is only one store exist |
| All the stores are disconnected and flat | Single store with hierarchical reducers |
| It has a singleton dispatcher | There is no concept of dispatcher |
| React components subscribe to the store | Container components uses connect function |

What are the benefits of React Router V4?

Below are the main benefits of React Router V4 module,

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

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

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

Can you describe about componentDidCatch lifecycle method signature?

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

error: - The error object which was thrown

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

The method structure would be as follows

componentDidCatch(error, info);

In which scenarios error boundaries do not catch errors?

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

Inside Event handlers

Asynchronous code using setTimeout or requestAnimationFrame callbacks

During Server side rendering

When errors thrown in the error boundary code itself

Why do you not need error boundaries for event handlers?

Error boundaries do not catch errors inside event handlers.

React doesn’t need error boundaries to recover from errors in event handlers. Unlike the render method and lifecycle methods, the event handlers don’t happen during rendering. So if they throw, React still knows what to display on the screen.

If you need to catch an error inside an event handler, use the regular JavaScript try / catch statement:

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { error: null };

this.handleClick = this.handleClick.bind(this);

}

handleClick() {

try {

// Do something that could throw

} catch (error) {

this.setState({ error });

}

}

render() {

if (this.state.error) { return <h1>Caught an error.</h1>; }

return <button onClick={this.handleClick}>Click Me</button>;

}

}

Note that the above example is demonstrating regular JavaScript behavior and doesn’t use error boundaries.

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

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

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

try { showButton(); }

catch (error) { // ...}

Whereas error boundaries wrap declarative code as below,

<ErrorBoundary>

<MyComponent />

</ErrorBoundary>

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

What is the behavior of uncaught errors in react 16?

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

What is the proper placement for error boundaries?

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

You can wrap top-level route components to display a generic error message for the entire application.

You can also wrap individual components in an error boundary to protect them from crashing the rest of the application.

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

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

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

stacktrace

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

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

What are the possible return types of render method?

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

React elements: Elements that instruct React to render a DOM node. It includes html elements such as <div/> and user defined elements.

Arrays and fragments: Return multiple elements to render as Arrays and Fragments to wrap multiple elements

Portals: Render children into a different DOM subtree.

String and numbers: Render both Strings and Numbers as text nodes in the DOM

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

What is the main purpose of constructor?

The constructor is mainly used for two purposes,

To initialize local state by assigning object to this.state

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

constructor(props) {

super(props); // Don't call this.setState() here!

this.state = { counter: 0 };

this.handleClick = this.handleClick.bind(this);

}

Is it mandatory to define constructor for React component?

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

What are default props?

The defaultProps can be defined as a property on the component to set the default values for the props. These default props are used when props not supplied(i.e., undefined props), but not for null props. That means, If you provide null value then it remains null value.

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

function MyButton { // … }

MyButton.defaultProps = { color: "red", };

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

render() { return <MyButton /> ; // props.color will contain red value }

Why should not call setState in componentWillUnmount?

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

What is the purpose of getDerivedStateFromError?

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

The signature of the lifecycle method is as follows,

static getDerivedStateFromError(error)

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

class ErrorBoundary extends React.Component {

constructor(props) {

super(props);

this.state = { hasError: false };

}

static getDerivedStateFromError(error) {

// Update state so the next render will show the fallback UI.

return { hasError: true };

}

render() {

if (this.state.hasError) { // You can render any custom fallback UI

return <h1>Something went wrong.</h1>;

}

return this.props.children;

}

}

What is the methods order when component re-rendered?

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

static getDerivedStateFromProps()

shouldComponentUpdate()

render()

getSnapshotBeforeUpdate()

componentDidUpdate()

What are the methods invoked during error handling?

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

static getDerivedStateFromError()

componentDidCatch()

What is the purpose of displayName class property?

The displayName string is used in debugging messages. Usually, you don’t need to set it explicitly because it’s inferred from the name of the function or class that defines the component. You might want to set it explicitly if you want to display a different name for debugging purposes or when you create a higher-order component.

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

function withSubscription(WrappedComponent) {

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

}

WithSubscription.displayName = `WithSubscription(${getDisplayName( WrappedComponent )})`;

return WithSubscription;

}

function getDisplayName(WrappedComponent) {

return ( WrappedComponent.displayName || WrappedComponent.name || "Component" );

}

What is the browser support for react applications?

React supports all popular browsers, including Internet Explorer 9 and above, although some polyfills are required for older browsers such as IE 9 and IE 10. If you use es5-shim and es5-sham polyfill then it even support old browsers that doesn't support ES5 methods.

What is the purpose of unmountComponentAtNode method?

This method is available from react-dom package and it removes a mounted React component from the DOM and clean up its event handlers and state. If no component was mounted in the container, calling this function does nothing. Returns true if a component was unmounted and false if there was no component to unmount.

The method signature would be as follows,

ReactDOM.unmountComponentAtNode(container);

What is code-splitting?

Code-Splitting is a feature supported by bundlers like Webpack and Browserify which can create multiple bundles that can be dynamically loaded at runtime. The react project supports code splitting via dynamic import() feature.

For example, in the below code snippets, it will make moduleA.js and all its unique dependencies as a separate chunk that only loads after the user clicks the 'Load' button. moduleA.js

const moduleA = "Hello";

export { moduleA };

App.js

import React, { Component } from "react";

class App extends Component {

handleClick = () => {

import("./moduleA")

.then(({ moduleA }) => { // Use moduleA })

.catch((err) => { // Handle failure });

};

render() {

return ( <div> <button onClick={this.handleClick}>Load</button> </div> );

}

}

export default App;

What are Keyed Fragments?

The Fragments declared with the explicit <React.Fragment> syntax may have keys. The general use case is mapping a collection to an array of fragments as below,

function Glossary(props) {

return (

<dl>

{props.items.map((item) => (

// Without the `key`, React will fire a key warning

<React.Fragment key={item.id}>

<dt>{item.term}</dt>

<dd>{item.description}</dd>

</React.Fragment>

))}

</dl>

);

}

Note: key is the only attribute that can be passed to Fragment. In the future, there might be a support for additional attributes, such as event handlers.

Does React support all HTML attributes?

As of React 16, both standard or custom DOM attributes are fully supported. Since React components often take both custom and DOM-related props, React uses the camelCase convention just like the DOM APIs.

Let us take few props with respect to standard HTML attributes,

<div tabIndex="-1" /> // Just like node.tabIndex DOM API

<div className="Button" /> // Just like node.className DOM API

<input readOnly={true} /> // Just like node.readOnly DOM API

These props work similarly to the corresponding HTML attributes, with the exception of the special cases. It also support all SVG attributes.

What are the limitations with HOCs?

Higher-order components come with a few caveats apart from its benefits. Below are the few listed in an order,

Don’t use HOCs inside the render method: It is not recommended to apply a HOC to a component within the render method of a component.

render() {

// A new version of EnhancedComponent is created on every render

// EnhancedComponent1 !== EnhancedComponent2

const EnhancedComponent = enhance(MyComponent);

// That causes the entire subtree to unmount/remount each time!

return <EnhancedComponent />;

}

The above code impacts on performance by remounting a component that causes the state of that component and all of its children to be lost. Instead, apply HOCs outside the component definition so that the resulting component is created only once.

Static methods must be copied over: When you apply a HOC to a component the new component does not have any of the static methods of the original component

// Define a static method

WrappedComponent.staticMethod = function () { /\*...\*/ };

// Now apply a HOC

const EnhancedComponent = enhance(WrappedComponent);

// The enhanced component has no static method

typeof EnhancedComponent.staticMethod === "undefined"; // true

You can overcome this by copying the methods onto the container before returning it,

function enhance(WrappedComponent) {

class Enhance extends React.Component { /\*...\*/ }

// Must know exactly which method(s) to copy :(

Enhance.staticMethod = WrappedComponent.staticMethod;

return Enhance;

}

Refs aren’t passed through: For HOCs you need to pass through all props to the wrapped component but this does not work for refs. This is because ref is not really a prop similar to key. In this case you need to use the React.forwardRef API

How to debug forwardRefs in DevTools?

React.forwardRef accepts a render function as parameter and DevTools uses this function to determine what to display for the ref forwarding component.

For example, If you don't name the render function or not using displayName property then it will appear as ”ForwardRef” in the DevTools,

const WrappedComponent = React.forwardRef((props, ref) => {

return <LogProps {...props} forwardedRef={ref} />;

});

But If you name the render function then it will appear as ”ForwardRef(myFunction)”

const WrappedComponent = React.forwardRef(function myFunction(props, ref) {

return <LogProps {...props} forwardedRef={ref} />;

});

As an alternative, You can also set displayName property for forwardRef function,

function logProps(Component) {

class LogProps extends React.Component { // ... }

function forwardRef(props, ref) {

return <LogProps {...props} forwardedRef={ref} />;

}

// Give this component a more helpful display name in DevTools.

// e.g. "ForwardRef(logProps(MyComponent))"

const name = Component.displayName || Component.name;

forwardRef.displayName = `logProps(${name})`;

return React.forwardRef(forwardRef);

}

When component props defaults to true?

If you pass no value for a prop, it defaults to true. This behavior is available so that it matches the behavior of HTML.

For example, below expressions are equivalent,

<MyInput autocomplete />

<MyInput autocomplete={true} />

Note: It is not recommended to use this approach because it can be confused with the ES6 object shorthand (example, {name} which is short for {name: name})

What is NextJS and major features of it?

Next.js is a popular and lightweight framework for static and server‑rendered applications built with React. It also provides styling and routing solutions. Below are the major features provided by NextJS,

Server-rendered by default

Automatic code splitting for faster page loads

Simple client-side routing (page based)

Webpack-based dev environment which supports (HMR)

Able to implement with Express or any other Node.js HTTP server

Customizable with your own Babel and Webpack configurations

How do you pass an event handler to a component?

You can pass event handlers and other functions as props to child components. It can be used in child component as below,

<button onClick="{this.handleClick}"></button>

Is it good to use arrow functions in render methods?

Yes, You can use. It is often the easiest way to pass parameters to callback functions. But you need to optimize the performance while using it.

class Foo extends Component {

handleClick() { console.log("Click happened"); }

render() { return <button onClick={() => this.handleClick()}>Click Me</button>; }

}

Note: Using an arrow function in render method creates a new function each time the component renders, which may have performance implications

How to prevent a function from being called multiple times?

If you use an event handler such as onClick or onScroll and want to prevent the callback from being fired too quickly, then you can limit the rate at which callback is executed. This can be achieved in the below possible ways,

Throttling: Changes based on a time based frequency. For example, it can be used using \_.throttle lodash function

Debouncing: Publish changes after a period of inactivity. For example, it can be used using \_.debounce lodash function

RequestAnimationFrame throttling: Changes based on requestAnimationFrame. For example, it can be used using raf-schd lodash function

How JSX prevents Injection Attacks?

React DOM escapes any values embedded in JSX before rendering them. Thus it ensures that you can never inject anything that’s not explicitly written in your application. Everything is converted to a string before being rendered.

For example, you can embed user input as below,

const name = response.potentiallyMaliciousInput;

const element = <h1>{name}</h1>;

This way you can prevent XSS(Cross-site-scripting) attacks in the application.

How do you update rendered elements?

You can update UI(represented by rendered element) by passing the newly created element to ReactDOM's render method.

For example, lets take a ticking clock example, where it updates the time by calling render method multiple times,

function tick() {

const element = (

<div>

<h1>Hello, world!</h1>

<h2>It is {new Date().toLocaleTimeString()}.</h2>

</div>

);

ReactDOM.render(element, document.getElementById("root"));

}

setInterval(tick, 1000);

How do you say that props are readonly?

When you declare a component as a function or a class, it must never modify its own props.

Let us take a below capital function,

function capital(amount, interest) {

return amount + interest;

}

The above function is called “pure” because it does not attempt to change their inputs, and always return the same result for the same inputs. Hence, React has a single rule saying "All React components must act like pure functions with respect to their props."

How do you say that state updates are merged?

When you call setState() in the component, React merges the object you provide into the current state.

For example, let us take a facebook user with posts and comments details as state variables,

constructor(props) {

super(props);

this.state = { posts: [], comments: [] };

}

Now you can update them independently with separate setState() calls as below,

componentDidMount() {

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

fetchComments().then(response => {

this.setState({ comments: response.comments });

});

}

As mentioned in the above code snippets, this.setState({comments}) updates only comments variable without modifying or replacing posts variable.

How do you pass arguments to an event handler?

During iterations or loops, it is common to pass an extra parameter to an event handler. This can be achieved through arrow functions or bind method.

Let us take an example of user details updated in a grid,

<button onClick={(e) => this.updateUser(userId, e)}>Update User details</button>

<button onClick={this.updateUser.bind(this, userId)}>Update User details</button>

In the both approaches, the synthetic argument e is passed as a second argument. You need to pass it explicitly for arrow functions and it will be passed automatically for bind method.

How to prevent component from rendering?

You can prevent component from rendering by returning null based on specific condition. This way it can conditionally render component.

function Greeting(props) {

if (!props.loggedIn) { return null; }

return <div className="greeting">welcome, {props.name}</div>;

}

class User extends React.Component {

constructor(props) {

super(props);

this.state = {loggedIn: false, name: 'John'};

}

render() {

return (

<div>

//Prevent component render if it is not loggedIn

<Greeting loggedIn={this.state.loggedIn} />

<UserDetails name={this.state.name}>

</div>

);

}

In the above example, the greeting component skips its rendering section by applying condition and returning null value.

What are the conditions to safely use the index as a key?

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

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.

Should keys be globally unique?

The keys used within arrays should be unique among their siblings but they don’t need to be globally unique. i.e, You can use the same keys with two different arrays.

For example, the below Book component uses two arrays with different arrays,

function Book(props) {

const index = (

<ul> {props.pages.map((page) => ( <li key={page.id}>{page.title}</li> ))} </ul>

);

const content = props.pages.map((page) => (

<div key={page.id}>

<h3>{page.title}</h3> <p>{page.content}</p> <p>{page.pageNumber}</p>

</div>

));

return ( <div> {index} <hr /> {content} </div> );

}

What is the popular choice for form handling?

Formik is a form library for react which provides solutions such as validation, keeping track of the visited fields, and handling form submission.

In detail, You can categorize them as follows,

Getting values in and out of form state

Validation and error messages

Handling form submission

It is used to create a scalable, performant, form helper with a minimal API to solve annoying stuff.

What are the advantages of formik over redux form library?

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

Why are you not required to use inheritance?

In React, it is recommended to use composition over 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 explicitly and safely. Whereas, If you want to reuse non-UI functionality between components, it is suggested to extract it into a separate JavaScript module. Later components import it and use that function, object, or class, without extending it.

Can I use web components in react application?

Yes, you can use web components in a react application. Even though many developers won't use this combination, it may require especially if you are using third-party UI components that are written using Web Components.

For example, let us use Vaadin date picker web component as below,

import React, { Component } from "react";

import "./App.css";

import "@vaadin/vaadin-date-picker";

class App extends Component {

render() {

return (

<div className="App">

<vaadin-date-picker label="When were you born?"></vaadin-date-picker>

</div>

);

}

}

export default App;

What is dynamic import?

You can achieve code-splitting in your app using dynamic import.

Let's take an example of addition,

Normal Import

import { add } from "./math";

console.log(add(10, 20));

Dynamic Import

import("./math").then((math) => { console.log(math.add(10, 20)); });

What are loadable components?

If you want to do code-splitting in a server rendered app, it is recommend to use Loadable Components because React.lazy and Suspense is not yet available for server-side rendering. Loadable lets you render a dynamic import as a regular component.

Lets take an example,

import loadable from "@loadable/component";

const OtherComponent = loadable(() => import("./OtherComponent"));

function MyComponent() {

return ( <div> <OtherComponent /> </div> );

}

Now OtherComponent will be loaded in a separated bundle

What is suspense component?

If the module containing the dynamic import is not yet loaded by the time parent component renders, you must show some fallback content while you’re waiting for it to load using a loading indicator. This can be done using Suspense component.

For example, the below code uses suspense component,

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

function MyComponent() {

return (

<div>

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

</div>

);

}

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

What is route based code splitting?

One of the best place to do code splitting is with routes. The entire page is going to re-render at once so users are unlikely to interact with other elements in the page at the same time. Due to this, the user experience won't be disturbed.

Let us take an example of route based website using libraries like React Router with React.lazy,

import { BrowserRouter as Router, Route, Switch } from "react-router-dom";

import React, { Suspense, lazy } from "react";

const Home = lazy(() => import("./routes/Home"));

const About = lazy(() => import("./routes/About"));

const App = () => (

<Router>

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

<Switch>

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

<Route path="/about" component={About} />

</Switch>

</Suspense>

</Router>

);

In the above code, the code splitting will happen at each route level.

Give an example on How to use context?

Context is designed to share data that can be considered global for a tree of React components.

For example, in the code below lets manually thread through a “theme” prop in order to style the Button component.

//Lets create a context with a default theme value "luna"

const ThemeContext = React.createContext("luna");

// Create App component where it uses provider to pass theme value in the tree

class App extends React.Component {

render() {

return ( <ThemeContext.Provider value="nova"> <Toolbar /> </ThemeContext.Provider> );

}

}

// A middle component where you don't need to pass theme prop anymore

function Toolbar(props) {

return ( <div> <ThemedButton /> </div> );

}

// Lets read theme value in the button component to use

class ThemedButton extends React.Component {

static contextType = ThemeContext;

render() { return <Button theme={this.context} />; }

}

What is the purpose of default value in context?

The defaultValue argument is only used when a component does not have a matching Provider above it in the tree. This can be helpful for testing components in isolation without wrapping them.

Below code snippet provides default theme value as Luna.

const MyContext = React.createContext(defaultValue);

How do you use contextType?

ContextType is used to consume the context object. The contextType property can be used in two ways,

contextType as property of class: The contextType property on a class can be assigned a Context object created by React.createContext(). After that, you can consume the nearest current value of that Context type using this.context in any of the lifecycle methods and render function.

Lets assign contextType property on MyClass as below,

class MyClass extends React.Component {

componentDidMount() {

let value = this.context; /\* perform a side-effect at mount using the value of MyContext \*/

}

componentDidUpdate() {

let value = this.context; /\* ... \*/

}

componentWillUnmount() {

let value = this.context; /\* ... \*/

}

render() {

let value = this.context; /\* render something based on the value of MyContext \*/

}

}

MyClass.contextType = MyContext;

Static field You can use a static class field to initialize your contextType using public class field syntax.

class MyClass extends React.Component {

static contextType = MyContext;

render() {

let value = this.context; /\* render something based on the value \*/

}

}

What is a consumer?

A Consumer is a React component that subscribes to context changes. It requires a function as a child which receives current context value as argument and returns a react node. The value argument passed to the function will be equal to the value prop of the closest Provider for this context above in the tree.

Lets take a simple example,

<MyContext.Consumer>

{value => /\* render something based on the context value \*/}

</MyContext.Consumer>

How do you solve performance corner cases while using context?

The context uses reference identity to determine when to re-render, there are some gotchas that could trigger unintentional renders in consumers when a provider’s parent re-renders.

For example, the code below will re-render all consumers every time the Provider re-renders because a new object is always created for value.

class App extends React.Component {

render() {

return ( <Provider value={{ something: "something" }}> <Toolbar /> </Provider> );

}

}

This can be solved by lifting up the value to parent state,

class App extends React.Component {

constructor(props) {

super(props);

this.state = { value: { something: "something" }, };

}

render() {

return ( <Provider value={this.state.value}> <Toolbar /> </Provider> );

}

}

What is the purpose of forward ref in HOCs?

Refs will not get passed through because ref is not a prop. It is handled differently by React just like key. If you add a ref to a HOC, the ref will refer to the outermost container component, not the wrapped component. In this case, you can use Forward Ref API. For example, we can explicitly forward refs to the inner FancyButton component using the React.forwardRef API.

The below HOC logs all props,

function logProps(Component) {

class LogProps extends React.Component {

componentDidUpdate(prevProps) {

console.log("old props:", prevProps);

console.log("new props:", this.props);

}

render() {

const { forwardedRef, ...rest } = this.props;

// Assign the custom prop "forwardedRef" as a ref

return <Component ref={forwardedRef} {...rest} />;

}

}

return React.forwardRef((props, ref) => {

return <LogProps {...props} forwardedRef={ref} />;

});

}

Let's use this HOC to log all props that get passed to our “fancy button” component,

class FancyButton extends React.Component {

focus() { // ... }

// …

}

export default logProps(FancyButton);

Now let's create a ref and pass it to FancyButton component. In this case, you can set focus to button element.

import FancyButton from "./FancyButton";

const ref = React.createRef();

ref.current.focus();

<FancyButton label="Click Me" handleClick={handleClick} ref={ref} />;

Is ref argument available for all functions or class components?

Regular function or class components don’t receive the ref argument, and ref is not available in props either. The second ref argument only exists when you define a component with React.forwardRef call.

Why do you need additional care for component libraries while using forward refs?

When you start using forwardRef in a component library, you should treat it as a breaking change and release a new major version of your library. This is because your library likely has a different behavior such as what refs get assigned to, and what types are exported. These changes can break apps and other libraries that depend on the old behavior.

How to create react class components without ES6?

If you don’t use ES6 then you may need to use the create-react-class module instead. For default props, you need to define getDefaultProps() as a function on the passed object. Whereas for initial state, you have to provide a separate getInitialState method that returns the initial state.

var Greeting = createReactClass({

getDefaultProps: function () { return { name: "Jhohn", }; },

getInitialState: function () { return { message: this.props.message }; },

handleClick: function () { console.log(this.state.message); },

render: function () { return <h1>Hello, {this.props.name}</h1>; },

});

Note: If you use createReactClass then auto binding is available for all methods. i.e, You don't need to use .bind(this) with in constructor for event handlers.

Is it possible to use react without JSX?

Yes, JSX is not mandatory for using React. Actually it is convenient when you don’t want to set up compilation in your build environment. Each JSX element is just syntactic sugar for calling React.createElement(component, props, ...children).

For example, let us take a greeting example with JSX,

class Greeting extends React.Component {

render() { return <div>Hello {this.props.message}</div>; }

}

ReactDOM.render( <Greeting message="World" />, document.getElementById("root") );

You can write the same code without JSX as below,

class Greeting extends React.Component {

render() { return React.createElement("div", null, `Hello ${this.props.message}`); }

}

ReactDOM.render(

React.createElement(Greeting, { message: "World" }, null), document.getElementById("root")

);

What is a diffing algorithm?

React needs to use algorithms to find out how to efficiently update the UI to match the most recent tree. The diffing algorithms is generating the minimum number of operations to transform one tree into another. However, the algorithms have a complexity in the order of O(n³) where n is the number of elements in the tree.

In this case, 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:

Two elements of different types will produce different trees.

The developer can hint at which child elements may be stable across different renders with a key prop.

What are the rules covered by the diffing algorithm?

When diffing two trees, React first compares the two root elements. The behavior is different depending on the types of the root elements. It covers the below rules during reconciliation algorithm,

Elements Of Different Types: Whenever the root elements have different types, React will tear down the old tree and build the new tree from scratch. For example, elements to , or from to of different types lead a full rebuild.

DOM Elements Of The Same Type: When comparing two React DOM elements of the same type, React looks at the attributes of both, keeps the same underlying DOM node, and only updates the changed attributes. Lets take an example with same DOM elements except className attribute,

<div className="show" title="ReactJS" />

<div className="hide" title="ReactJS" />

Component Elements Of The Same Type: When a component updates, the instance stays the same, so that state is maintained across renders. React updates the props of the underlying component instance to match the new element, and calls componentWillReceiveProps() and componentWillUpdate() on the underlying instance. After that, the render() method is called and the diff algorithm recurses on the previous result and the new result.

Recursing On Children: when recursing on the children of a DOM node, React just iterates over both lists of children at the same time and generates a mutation whenever there’s a difference. For example, when adding an element at the end of the children, converting between these two trees works well.

<ul>

<li>first</li> <li>second</li>

</ul>

<ul>

<li>first</li> <li>second</li>

<li>third</li>

</ul>

Handling keys: React supports a key attribute. When children have keys, React uses the key to match children in the original tree with children in the subsequent tree. For example, adding a key can make the tree conversion efficient,

<ul>

<li key="2015">Duke</li> <li key="2016">Villanova</li>

</ul>

<ul>

<li key="2014">Connecticut</li> <li key="2015">Duke</li> <li key="2016">Villanova</li>

</ul>

When do you need to use refs?

There are few use cases to go for refs,

Managing focus, text selection, or media playback.

Triggering imperative animations.

Integrating with third-party DOM libraries.

Must prop be named as render for render props?

Even though the pattern named render props, you don’t have to use a prop named render to use this pattern. i.e, Any prop that is a function that a component uses to know what to render is technically a “render prop”. Lets take an example with the children prop for render props,

<Mouse children={(mouse) => ( <p> The mouse position is {mouse.x}, {mouse.y} </p> )} />

Actually children prop doesn’t need to be named in the list of “attributes” in JSX element. Instead, you can keep it directly inside element,

<Mouse> {(mouse) => ( <p> The mouse position is {mouse.x}, {mouse.y} </p> )} </Mouse>

While using this above technique(without any name), explicitly state that children should be a function in your propTypes.

Mouse.propTypes = { children: PropTypes.func.isRequired, };

What are the problems of using render props with pure components?

If you create a function inside a render method, it negates the purpose of pure component. Because the shallow prop comparison will always return false for new props, and each render in this case will generate a new value for the render prop. You can solve this issue by defining the render function as instance method.

How do you create HOC using render props?

You can implement most higher-order components (HOC) using a regular component with a render prop. For example, if you would prefer to have a withMouse HOC instead of a component, you could easily create one using a regular with a render prop.

function withMouse(Component) {

return class extends React.Component {

render() {

return ( <Mouse render={(mouse) => <Component {...this.props} mouse={mouse} />} /> );

}

};

}

This way render props gives the flexibility of using either pattern.

What is windowing technique?

Windowing is a technique that only renders a small subset of your rows at any given time, and can dramatically reduce the time it takes to re-render the components as well as the number of DOM nodes created. If your application renders long lists of data then this technique is recommended. Both react-window and react-virtualized are popular windowing libraries which provides several reusable components for displaying lists, grids, and tabular data.

How do you print falsy values in JSX?

The falsy values such as false, null, undefined, and true are valid children but they don't render anything. If you still want to display them then you need to convert it to string. Let's take an example on how to convert to a string,

<div>My JavaScript variable is {String(myVariable)}.</div>

What is the typical use case of portals?

React portals are very useful when a parent component has overflow: hidden or has properties that affect the stacking context (e.g. z-index, position, opacity) and you need to visually “break out” of its container.

For example, dialogs, global message notifications, hovercards, and tooltips.

How do you set default value for uncontrolled component?

In React, the value attribute on form elements will override the value in the DOM. With an uncontrolled component, you might want React to specify the initial value, but leave subsequent updates uncontrolled. To handle this case, you can specify a defaultValue attribute instead of value.

render() {

return (

<form onSubmit={this.handleSubmit}>

<label> User Name: <input defaultValue="John" type="text" ref={this.input} />

</label> <input type="submit" value="Submit" />

</form>

);

}

The same applies for select and textArea inputs. But you need to use defaultChecked for checkbox and radio inputs.

What is your favorite React stack?

Even though the tech stack varies from developer to developer, the most popular stack is used in react boilerplate project code. It mainly uses Redux and redux-saga for state management and asynchronous side-effects, react-router for routing purpose, styled-components for styling react components, axios for invoking REST api, and other supported stack such as webpack, reselect, ESNext, Babel. You can clone the project https://github.com/react-boilerplate/react-boilerplate and start working on any new react project.

How to add Bootstrap to a react application?

Bootstrap can be added to your React app in a three possible ways,

Using the Bootstrap CDN: This is the easiest way to add bootstrap. Add both bootstrap CSS and JS resources in a head tag.

Bootstrap as Dependency: If you are using a build tool or a module bundler such as Webpack, then this is the preferred option for adding Bootstrap to your React application

npm install bootstrap

React Bootstrap Package: In this case, you can add Bootstrap to our React app is by using a package that has rebuilt Bootstrap components to work particularly as React components. Below packages are popular in this category,

react-bootstrap

reactstrap

Can you list down top websites or applications using react as front end framework?

Below are the top 10 websites using React as their front-end framework,

>>> Facebook >>> Uber

>>> Instagram >>> WhatsApp

>>> Khan Academy >>> Airbnb

>>> Dropbox >>> Flipboard

>>> Netflix >>> PayPal

Is it recommended to use CSS In JS technique in React?

React does not have any opinion about how styles are defined but if you are a beginner then good starting point is to define your styles in a separate \*.css file as usual and refer to them using className. This functionality is not part of React but came from third-party libraries. But If you want to try a different approach(CSS-In-JS) then styled-components library is a good option.

Do I need to rewrite all my class components with hooks?

No. But you can try Hooks in a few components(or new components) without rewriting any existing code. Because there are no plans to remove classes in ReactJS.

How to fetch data with React Hooks?

The effect hook called useEffect can be used to fetch data from an API and to set the data in the local state of the component with the useState hook’s update function.

Here is an example of fetching a list of react articles from an API using fetch.

import React from "react";

function App() {

const [data, setData] = React.useState({ hits: [] });

React.useEffect(() => {

fetch("http://hn.algolia.com/api/v1/search?query=react")

.then(response => response.json())

.then(data => setData(data))

}, []);

return (

<ul>

{data.hits.map((item) => ( <li key={item.objectID}> <a href={item.url}>{item.title}</a> </li> ))}

</ul>

);

}

export default App;

A popular way to simplify this is by using the library axios.

We provided an empty array as second argument to the useEffect hook to avoid activating it on component updates. This way, it only fetches on component mount.

Is Hooks cover all use cases for classes?

Hooks doesn't cover all use cases of classes but there is a plan to add them soon. Currently there are no Hook equivalents to the uncommon getSnapshotBeforeUpdate and componentDidCatch lifecycles yet.

What is the stable release for hooks support?

React includes a stable implementation of React Hooks in 16.8 release for below packages

React DOM

React DOM Server

React Test Renderer

React Shallow Renderer

Why do we use array destructuring (square brackets notation) in useState?

When we declare a state variable with useState, it returns a pair — an array with two items. The first item is the current value, and the second is a function that updates the value. Using [0] and [1] to access them is a bit confusing because they have a specific meaning. This is why we use array destructuring instead.

For example, the array index access would look as follows:

var userStateVariable = useState("userProfile"); // Returns an array pair

var user = userStateVariable[0]; // Access first item

var setUser = userStateVariable[1]; // Access second item

Whereas with array destructuring the variables can be accessed as follows:

const [user, setUser] = useState("userProfile");

What are the sources used for introducing hooks?

Hooks got the ideas from several different sources. Below are some of them,

* Previous experiments with functional APIs in the react-future repository
* Community experiments with render prop APIs such as Reactions Component
* State variables and state cells in DisplayScript.
* Subscriptions in Rx.
* Reducer components in ReasonReact.

How do you access imperative API of web components?

Web Components often expose an imperative API to implement its functions. You will need to use a ref to interact with the DOM node directly if you want to access imperative API of a web component. But if you are using third-party Web Components, the best solution is to write a React component that behaves as a wrapper for your Web Component.

What is formik?

Formik is a small react form library that helps you with the three major problems,

* Getting values in and out of form state
* Validation and error messages
* Handling form submission

What are typical middleware choices for handling asynchronous calls in Redux?

Some of the popular middleware choices for handling asynchronous calls in Redux eco system are Redux Thunk, Redux Promise, Redux Saga.

Do browsers understand JSX code?

No, browsers can't understand JSX code. You need a transpiler to convert your JSX to regular Javascript that browsers can understand. The most widely used transpiler right now is Babel.

Describe about data flow in react?

React implements one-way reactive data flow using props which reduce boilerplate and is easier to understand than traditional two-way data binding.

What is react scripts?

The react-scripts package is a set of scripts from the create-react-app starter pack which helps you kick off projects without configuring. The react-scripts start command sets up the development environment and starts a server, as well as hot module reloading.

What are the features of create react app?

Below are the list of some of the features provided by create react app.

React, JSX, ES6, Typescript and Flow syntax support.

Autoprefixed CSS

CSS Reset/Normalize

A live development server

A fast interactive unit test runner with built-in support for coverage reporting

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

An offline-first service worker and a web app manifest, meeting all the Progressive Web App criteria.

What is the purpose of renderToNodeStream method?

The ReactDOMServer#renderToNodeStream method is used to generate HTML on the server and send the markup down on the initial request for faster page loads. It also helps search engines to crawl your pages easily for SEO purposes. Note: Remember this method is not available in the browser but only server.

What is MobX?

MobX is a simple, scalable and battle tested state management solution for applying functional reactive programming (TFRP). For reactJs application, you need to install below packages,

npm install mobx --save

npm install mobx-react --save

What are the differences between Redux and MobX?

| Topic | Redux | MobX |
| --- | --- | --- |
| Definition | It is a javascript library for managing the application state | It is a library for reactively managing the state of your applications |
| Programming | It is mainly written in ES6 | It is written in JavaScript(ES5) |
| Data Store | There is only one large store exist for data storage | There is more than one store for storage |
| Usage | Mainly used for large and complex applications | Used for simple applications |
| Performance | Need to be improved | Provides better performance |
| How it stores | Uses JS Object to store | Uses observable to store the data |

Should I learn ES6 before learning ReactJS?

No, you don’t have to learn es2015/es6 to learn react. But you may find many resources or React ecosystem uses ES6 extensively. Let's see some of the frequently used ES6 features,

Destructuring: To get props and use them in a component

// in es 5

var someData = this.props.someData;

var dispatch = this.props.dispatch;

// in es6

const { someData, dispatch } = this.props;

Spread operator: Helps in passing props down into a component

// in es 5

<SomeComponent someData={this.props.someData} dispatch={this.props.dispatch} />

// in es6

<SomeComponent {...this.props} />

Arrow functions: Makes compact syntax

var users = usersList.map(function (user) { return <li>{user.name}</li>; }); // es 5

const users = usersList.map((user) => <li>{user.name}</li>); // es 6

What is Concurrent Rendering?

The Concurrent rendering makes React apps to be more responsive by rendering component trees without blocking the main UI thread. It allows React to interrupt a long-running render to handle a high-priority event. i.e, When you enabled concurrent Mode, React will keep an eye on other tasks that need to be done, and if there's something with a higher priority it will pause what it is currently rendering and let the other task finish first. You can enable this in two ways,

// 1. Part of an app by wrapping with ConcurrentMode

<React.unstable\_ConcurrentMode>

<Something />

</React.unstable\_ConcurrentMode>;

// 2. Whole app using createRoot

ReactDOM.unstable\_createRoot(domNode).render(<App />);

What is the difference between async mode and concurrent mode?

Both refers the same thing. Previously concurrent Mode being referred to as "Async Mode" by React team. The name has been changed to highlight React’s ability to perform work on different priority levels. So it avoids the confusion from other approaches to Async Rendering.

Can I use javascript urls in react16.9?

Yes, you can use javascript: URLs but it will log a warning in the console. Because URLs starting with javascript: are dangerous by including unsanitized output in a tag like <a href> and create a security hole.

const companyProfile = { website: "javascript: alert('Your website is hacked')", };

// It will log a warning

<a href={companyProfile.website}>More details</a>;

Remember that the future versions will throw an error for javascript URLs.

What is the purpose of eslint plugin for hooks?

The ESLint plugin enforces rules of Hooks to avoid bugs. It assumes that any function starting with ”use” and a capital letter right after it is a Hook. In particular, the rule enforces that,

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.

What is the difference between Imperative and Declarative in React?

Imagine a simple UI component, such as a "Like" button. When you tap it, it turns blue if it was previously grey, and grey if it was previously blue.

The imperative way of doing this would be:

if (user.likes()) {

if (hasBlue()) { removeBlue(); addGrey(); }

else { removeGrey(); addBlue(); }

}

Basically, you have to check what is currently on the screen and handle all the changes necessary to redraw it with the current state, including undoing the changes from the previous state. You can imagine how complex this could be in a real-world scenario.

In contrast, the declarative approach would be:

if (this.state.liked) { return <blueLike />; }

else { return <greyLike />; }

Because the declarative approach separates concerns, this part of it only needs to handle how the UI should look in a specific state, and is therefore much simpler to understand.

What are the benefits of using typescript with reactjs?

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

How do you make sure that user remains authenticated on page refresh while using Context API State Management?

When a user logs in and reload, to persist the state generally we add the load user action in the useEffect hooks in the main App.js. While using Redux, loadUser action can be easily accessed.

App.js

import { loadUser } from "../actions/auth";

store.dispatch(loadUser());

But while using Context API, to access context in App.js, wrap the AuthState in index.js so that App.js can access the auth context. Now whenever the page reloads, no matter what route you are on, the user will be authenticated as loadUser action will be triggered on each re-render.

index.js

import React from "react";

import ReactDOM from "react-dom";

import App from "./App";

import AuthState from "./context/auth/AuthState";

ReactDOM.render(

<React.StrictMode>

<AuthState> <App /> </AuthState>

</React.StrictMode>,

document.getElementById("root")

);

App.js

const authContext = useContext(AuthContext);

const { loadUser } = authContext;

useEffect(() => { loadUser();}, []);

const loadUser = async () => {

const token = sessionStorage.getItem("token");

if (!token) { dispatch({ type: ERROR, }); }

setAuthToken(token);

try {

const res = await axios("/api/auth");

dispatch({ type: USER\_LOADED, payload: res.data.data, });

} catch (err) { console.error(err); }

};

What are the benefits of new JSX transform?

There are three major benefits of new JSX transform,

It is possible to use JSX without importing React packages

The compiled output might improve the bundle size in a small amount

The future improvements provides the flexibility to reduce the number of concepts to learn React.

How is the new JSX transform different from old transform??

The new JSX transform doesn’t require React to be in scope. i.e, You don't need to import React package for simple scenarios.

Let's take an example to look at the main differences between the old and the new transform,

Old Transform:

import React from "react";

function App() { return <h1>Good morning!!</h1>; }

Now JSX transform convert the above code into regular JavaScript as below,

import React from "react";

function App() {

return React.createElement("h1", null, "Good morning!!");

}

New Transform:

The new JSX transform doesn't require any React imports

function App() { return <h1>Good morning!!</h1>; }

Under the hood JSX transform compiles to below code

import { jsx as \_jsx } from "react/jsx-runtime";

function App() {

return \_jsx("h1", { children: "Good morning!!" });

}

Note: You still need to import React to use Hooks.

How do you get redux scaffolding using create-react-app?

Redux team has provided official redux+js or redux+typescript templates for create-react-app project. The generated project setup includes,

Redux Toolkit and React-Redux dependencies

Create and configure Redux store

React-Redux <Provider> passing the store to React components

Small "counter" example to demo how to add redux logic and React-Redux hooks API to interact with the store from components The below commands need to be executed along with template option as below,

Javascript template:

npx create-react-app my-app --template redux

Typescript template:

npx create-react-app my-app --template redux-typescript

What are React Server components?

React Server Component is a way to write React component that gets rendered in the server-side with the purpose of improving React app performance. These components allow us to load components from the backend.

Note: React Server Components is still under development and not recommended for production yet.

What is prop drilling?

Prop Drilling is the process by which you pass data from one component of the React Component tree to another by going through other components that do not need the data but only help in passing it around.

What is state mutation and how to prevent it?

State mutation happens when you try to update the state of a component without actually using setState function. This can happen when you are trying to do some computations using a state variable and unknowingly save the result in the same state variable. This is the main reason why it is advised to return new instances of state variables from the reducers by using Object.assign({}, ...) or spread syntax.

This can cause unknown issues in the UI as the value of the state variable got updated without telling React to check what all components were being affected from this update and it can cause UI bugs.

Ex:

class A extends React.component {

constructor(props) {

super(props);

this.state = { loading: false }

}

componentDidMount() {

let { loading } = this.state;

loading = (() => true)(); // Trying to perform an operation and directly saving in a state variable

}

How to prevent it: Make sure your state variables are immutable by either enforcing immutability by using plugins like Immutable.js, always using setState to make updates, and returning new instances in reducers when sending updated state values.

What is the difference between useState and useRef hook?

useState causes components to re-render after state updates whereas useRef doesn’t cause a component to re-render when the value or state changes. Essentially, useRef is like a “box” that can hold a mutable value in its (.current) property.

useState allows us to update the state inside components. While useRef allows referencing DOM elements.

What is a wrapper component?

A wrapper in React is a component that wraps or surrounds another component or group of components. It can be used for a variety of purposes such as adding additional functionality, styling, or layout to the wrapped components.

For example, consider a simple component that displays a message:

const Message = ({ text }) => {

return <p>{text}</p>;

};

We can create a wrapper component that will add a border to the message component:

const MessageWrapper = (props) => {

return (

<div style={{ border: "1px solid black" }}>

<Message {...props} />

</div>

);

};

Now we can use the MessageWrapper component instead of the Message component and the message will be displayed with a border:

<MessageWrapper text="Hello World" />

Wrapper component can also accept its own props and pass them down to the wrapped component, for example, we can create a wrapper component that will add a title to the message component:

const MessageWrapperWithTitle = ({title, ...props}) => {

return (

<div>

<h3>{title}</h3>

<Message {...props} />

</div>

);

};

Now we can use the MessageWrapperWithTitle component and pass title props:

<MessageWrapperWithTitle title="My Message" text="Hello World" />

This way, the wrapper component can add additional functionality, styling, or layout to the wrapped component while keeping the wrapped component simple and reusable.

What are the differences between useEffect and useLayoutEffect hooks?

useEffect and useLayoutEffect are both React hooks that can be used to synchronize a component with an external system, such as a browser API or a third-party library. However, there are some key differences between the two:

Timing: useEffect runs after the browser has finished painting, while useLayoutEffect runs synchronously before the browser paints. This means that useLayoutEffect can be used to measure and update layout in a way that feels more synchronous to the user.

Browser Paint: useEffect allows browser to paint the changes before running the effect, hence it may cause some visual flicker. useLayoutEffect synchronously runs the effect before browser paints and hence it will avoid visual flicker.

Execution Order: The order in which multiple useEffect hooks are executed is determined by React and may not be predictable. However, the order in which multiple useLayoutEffect hooks are executed is determined by the order in which they were called.

Error handling: useEffect has a built-in mechanism for handling errors that occur during the execution of the effect, so that it does not crash the entire application. useLayoutEffect does not have this mechanism, and errors that occur during the execution of the effect will crash the entire application.

In general, it's recommended to use useEffect as much as possible, because it is more performant and less prone to errors. useLayoutEffect should only be used when you need to measure or update layout, and you can't achieve the same result using useEffect.

What are the differences between Functional and Class Components?

There are two different ways to create components in ReactJS. The main differences are listed down as below,

1. Syntax:

The classs components uses ES6 classes to create the components. It uses render function to display the HTML content in the webpage.

The syntax for class component looks like as below.

class App extends Reacts.Component {

render(){ return <h1>This is a class component</h1>} }

Note: The Pascal Case is the recommended approach to provide naming to a component.

Functional component has been improved over the years with some added features like Hooks. Here is a syntax for functional component.

function App(){

return <div className="App">

<h1>Hello, I'm a function component</h1>

</div>

}

2. State:

State contains information or data about a component which may change over time.

In class component, you can update the state when a user interacts with it or server updates the data using the setState() method. The initial state is going to be assigned in the Constructor( ) method using the the this.state object and it is possible to different data types in the this.state object such as string, boolean, numbers, etc. A simple example showing how we use the setState() and constructor()

class App extends Component {

constructor() {

super();

this.state = { message: "This is a class component", };

}

updateMessage() {

this.setState({ message: "Updating the class component", });

}

render() {

return (

<>

<h1>{this.state.message}</h1>

<button onClick={() => { this.updateMessage(); }}> Click!! </button>

</>

);

}

}

You not use state in functional components because it was only supported in class components. But over the years hooks have been implemented in functional component which enable to use state in functional component too.

The useState() hook can used to implement state in funcitonal component. It returns an array with two items: the first item is current state and the next one is a function (setState) that updates the value of the current state.

Let's see an example to demonstrate the state in functional components,

function App() {

const [message, setMessage] = useState("This is a functional component");

const updateMessage = () => {

setMessage("Updating the functional component");

};

return (

<div className="App">

<h1>{message} </h1> <button onClick={updateMessage}>Click me!!</button>

</div>

);

}

4. Props:

Props are referred to as "properties". The props are passed into react component just like arguments passed to a function. In otherwords, they are similar to HTML attributes.

The props are accessible in child class component using this.props as shown in below example,

class Child extends React.Component {

render() {

return <h1> This is a functional component and component name is {this.props.name} </h1>;

}

}

class Parent extends React.Component {

render() {

return (

<div className="Parent">

<Child name="First child component" /> <Child name="Second child component" />

</div>

);

}

}

Props in functional components are similar to that of the class components but the difference is the absence of 'this' keyword.

function Child(props) {

return <h1>This is a child component and the component name is{props.name}</h1>;

}

function Parent() {

return (

<div className="Parent">

<Child name="First child component" /> <Child name="Second child component" />

</div>

);

}

What is strict mode in React?

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

```jsx harmony

import React from "react";

function ExampleApplication() {

return (

<div>

<Header />

<React.StrictMode>

<div> <ComponentOne /> <ComponentTwo /> </div>

</React.StrictMode>

<Header />

</div>

);

}

```

In the example above, the \_strict mode\_ checks apply to `<ComponentOne>` and `<ComponentTwo>` components only. i.e., Part of the application only.

What is the benefit of strict mode?

The will be helpful in the below cases,

Whenever the component

* 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

Why does strict mode render twice in React?

StrictMode renders components twice in development mode(not production) in order to detect any problems with your code and warn you about those problems. This is used to detect accidental side effects in the render phase. If you used create-react-app development tool then it automatically enables StrictMode by default.

ReactDOM.render(

<React.StrictMode> {App} </React.StrictMode>,

document.getElementById('root')

);

If you want to disable this behavior then you can remove strict mode.

ReactDOM.render( {App}, document.getElementById('root'));

To detect side effects the following functions are invoked twice:

Class component constructor, render, and shouldComponentUpdate methods

Class component static getDerivedStateFromProps method

Function component bodies

State updater functions

Functions passed to useState, useMemo, or useReducer (any Hook)

============

1. What are the features of React?

JSX react

JSX: JSX is a syntax extension to JavaScript. It is used with React to describe what the user interface should look like. By using JSX, we can write HTML structures in the same file that contains JavaScript code.

Components

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

Virtual DOM

Virtual DOM: React keeps a lightweight representation of the real DOM in the memory, and that is known as the virtual DOM. When the state of an object changes, virtual DOM changes only that object in the real DOM, rather than updating all the objects.

Data binding

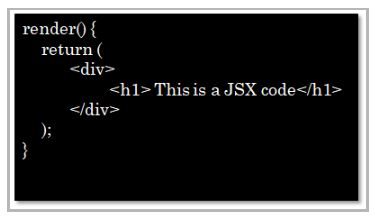
One-way data-binding: React’s one-way data binding keeps everything modular and fast. A unidirectional data flow means that when designing a React app, you often nest child components within parent components.

High-performance

High performance: React updates only those components that have changed, rather than updating all the components at once. This results in much faster web applications.

### 2. What is JSX?

JSX is a syntax extension of JavaScript. It is used with React to describe what the user interface should look like. By using JSX, we can write HTML structures in the same file that contains JavaScript code.



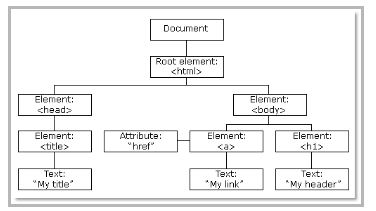
### 3. Can web browsers read JSX directly?

* Web browsers cannot read JSX directly. This is because they are built to only read regular JS objects and JSX is not a regular JavaScript object
* For a web browser to read a JSX file, the file needs to be transformed into a regular JavaScript object. For this, we use Babel

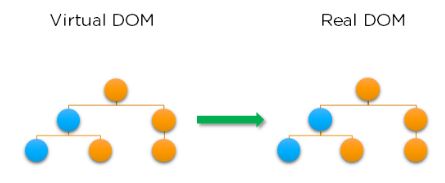


### 4. What is the virtual DOM?

DOM stands for Document Object Model. The DOM represents an HTML document with a logical tree structure. Each branch of the tree ends in a node, and each node contains objects.



React keeps a lightweight representation of the real DOM in the memory, and that is known as the virtual DOM. When the state of an object changes, the virtual DOM changes only that object in the real DOM, rather than updating all the objects. The following are some of the most frequently asked react interview questions.



5. Why use React instead of other frameworks, like Angular?

Dynamic

Easy creation of dynamic applications: React makes it easier to create dynamic web applications because it provides less coding and provides more functionality, whereas, with JavaScript applications, code tends to get complex very quickly.

Improved

Improved performance: React uses virtual DOM, which makes web applications perform faster. Virtual DOM compares its previous state and updates only those components in the real DOM, whose states have changed, rather than updating all the components — like conventional web applications.

Reusable

Reusable components: Components are the building blocks of any React application, and a single app usually consists of multiple components. These components have their own logic and controls, and they can be reused through the application, which, in turn, dramatically reduces the development time of an application.

Data flow

Unidirectional data flow: React follows a unidirectional data flow. This means that when designing a React app, we often nest child components within parent components. And since the data flows in a single direction, it becomes easier to debug errors and know where the problem occurs in an application at the moment.

Dedicated

Dedicated tools for easy debugging: Facebook has released a chrome extension that we can use to debug React applications. This makes the process of debugging React to web applications faster and easier.

### 8. What is an event in React?

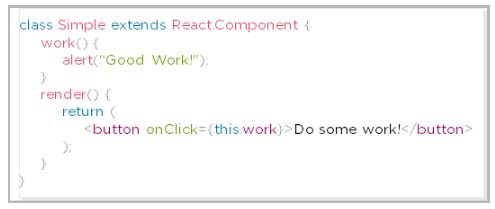
An event is an action that a user or system may trigger, such as pressing a key, a mouse click, etc.

* React events are named using camelCase, rather than lowercase in HTML.
* With JSX, you pass a function as the event handler, rather than a string in HTML.

<Button onPress={lightItUp} />

### 9. How do you create an event in React?

A React event can be created by doing the following:



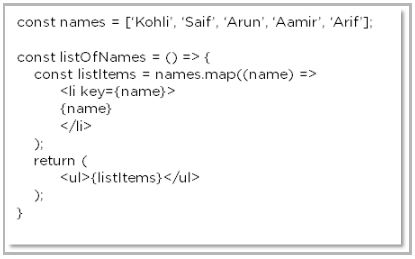
### 10. What are synthetic events in React?

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



### 11. Explain how lists work in React

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



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

Keys are very important in lists for the following reasons:

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

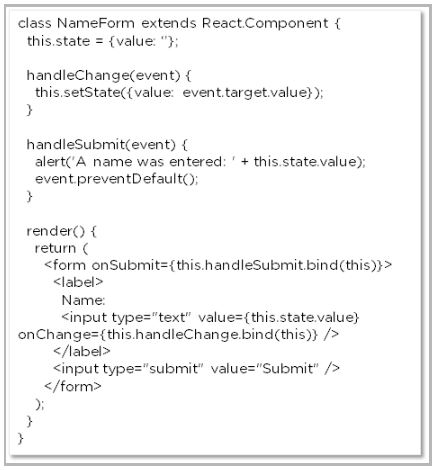
### 13. What are forms in React?

React employs forms to enable users to interact with web applications.

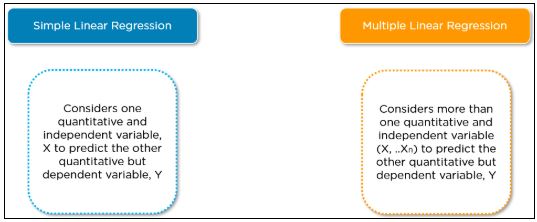
* Using forms, users can interact with the application and enter the required information whenever needed. Form contain certain elements, such as text fields, buttons, checkboxes, radio buttons, etc
* Forms are used for many different tasks such as user authentication, searching, filtering, indexing, etc

### 14. How do you create forms in React?

We create forms in React by doing the following:



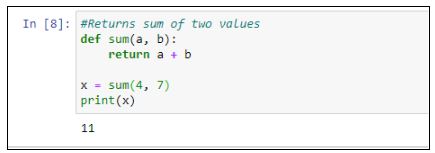
The above code will yield an input field with the label Name and a submit button. It will also alert the user when the submit button is pressed.



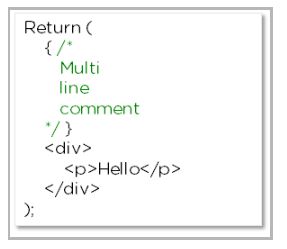
### 15. How do you write comments in React?

There are basically two ways in which we can write comments:

* Single-line comments

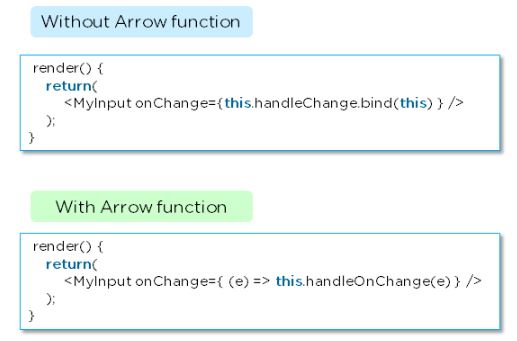


* Multi-line comments



### 16. What is an arrow function and how is it used in React?

* An arrow function is a short way of writing a function to React.
* It is unnecessary to bind ‘this’ inside the constructor when using an arrow function. This prevents bugs caused by the use of ‘this’ in React callbacks.



### 7. How is React different from React Native?

|  | React | React Native |
| --- | --- | --- |
| Release | 2013 | 2015 |
| Platform | Web | Mobile – Android, iOS |
| HTML | Yes | No |
| CSS | Yes | No |
| Prerequisites | JavaScript, HTML, CSS | React.js |

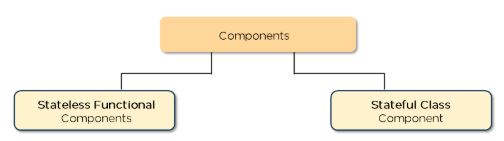
### 18. How is React different from Angular?

|  | Angular | React |
| --- | --- | --- |
| Author | Google | Facebook |
| Architecture | Complete MVC | View layer of MVC |
| DOM | Real DOM | Virtual DOM |
| Data-Binding | Bi-directional | Uni-directional |
| Rendering | Client-Side | Server-Side |
| Performance | Comparatively slow | Faster due to Virtual DOM |

### 19. What are the components in React?

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

There are two types of components in React:



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

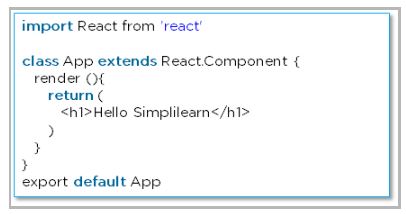
| function Greeting(props) {  return <h1>Welcome to {props.name}</h1>;  } |
| --- |

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

| class Greeting extends React.Component {  render() {  return <h1>Welcome to {this.props.name}</h1>;  }  } |
| --- |

### 20. What is the use of render() in React?

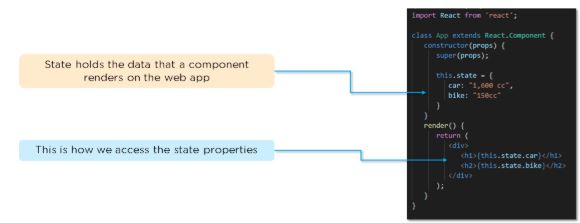
* It is required for each component to have a render() function. This function returns the HTML, which is to be displayed in the component.
* If you need to render more than one element, all of the elements must be inside one parent tag like <div>, <form>.



### 21. What is a state in React?

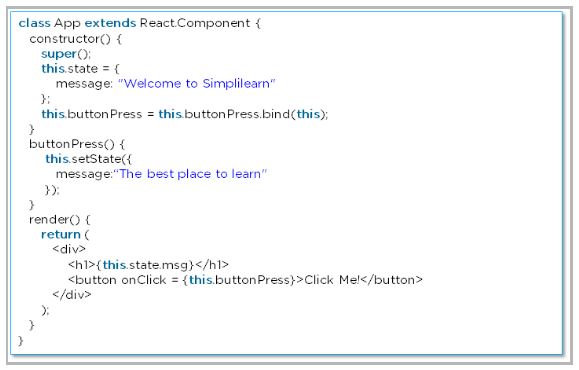
* The state is a built-in React object that is used to contain data or information about the component. The state in a component can change over time, and whenever it changes, the component re-renders.
* The change in state can happen as a response to user action or system-generated events. It determines the behavior of the component and how it will render.

### 22. How do you implement state in React?



### 23. How do you update the state of a component?

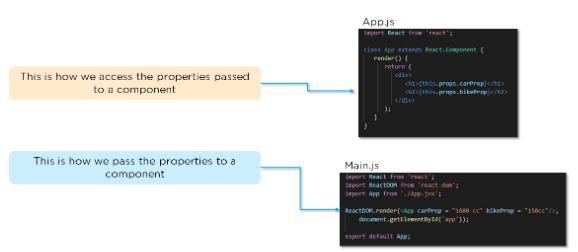
We can update the state of a component by using the built-in ‘setState()’ method:



### 24. What are props in React?

* [Props](https://www.simplilearn.com/tutorials/reactjs-tutorial/react-props) are short for Properties. It is a React built-in object that stores the value of attributes of a tag and works similarly to HTML attributes.
* Props provide a way to pass data from one component to another component. Props are passed to the component in the same way as arguments are passed in a function.

### 25. How do you pass props between components?



### 26. What are the differences between state and props?

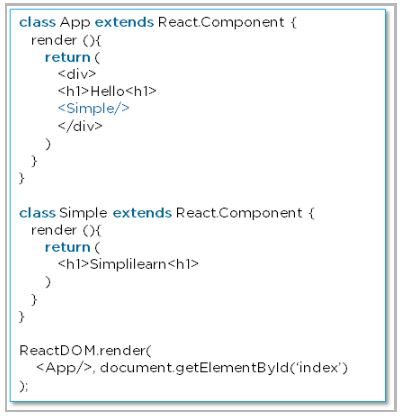
|  | State | Props |
| --- | --- | --- |
| Use | Holds information about the components | Allows to pass data from one component to other components as an argument |
| Mutability | Is mutable | Are immutable |
| Read-Only | Can be changed | Are read-only |
| Child components | Child components cannot access | Child component can access |
| Stateless components | Cannot have state | Can have props |

### 27. What is a higher-order component in React?

A higher-order component acts as a container for other components. This helps to keep components simple and enables re-usability. They are generally used when multiple components have to use a common logic.

### 28. How can you embed two or more components into one?

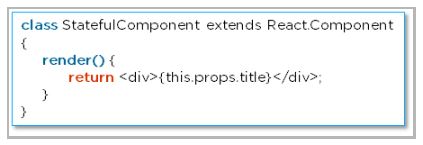
We can embed two or more components into one using this method:



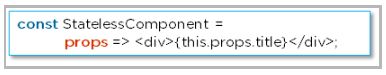
### 29. What are the differences between class and functional components?

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

* Class components example:



* Functional components example:



### 30. Explain the lifecycle methods of components.

* getInitialState(): This is executed before the creation of the component.
* componentDidMount(): Is executed when the component gets rendered and placed on the DOM.
* shouldComponentUpdate(): Is invoked when a component determines changes to the DOM and returns a “true” or “false” value based on certain conditions.
* componentDidUpdate(): Is invoked immediately after rendering takes place.
* componentWillUnmount(): Is invoked immediately before a component is destroyed and unmounted permanently.

So far, if you have any doubts about the above React interview questions and answers, please ask your questions in the section below.

### 31. What is Redux?

[Redux](https://www.simplilearn.com/tutorials/reactjs-tutorial/react-with-redux) is an open-source, JavaScript library used to manage the application state. React uses Redux to build the user interface. It is a predictable state container for JavaScript applications and is used for the entire application’s state management.

### 32. What are the components of Redux?

* Store: Holds the state of the application.
* Action: The source information for the store.
* Reducer: Specifies how the application's state changes in response to actions sent to the store.



### 33. What is the Flux?

* Flux is the application architecture that Facebook uses for building web applications. It is a method of handling complex data inside a client-side application and manages how data flows in a React application.



* There is a single source of data (the store) and triggering certain actions is the only way way to update them.The actions call the dispatcher, and then the store is triggered and updated with their own data accordingly.



* When a dispatch has been triggered, and the store updates, it will emit a change event that the views can rerender accordingly.



### 34. How is Redux different from Flux?

| SN | Redux | Flux |
| --- | --- | --- |
| 1. | Redux is an open-source JavaScript library used to manage application State | Flux is an architecture and not a framework or library |
| 2. | Store’s state is immutable | Store’s state is mutable |
| 3. | Can only have a single-store | Can have multiple stores |
| 4. | Uses the concept of reducer | Uses the concept of the dispatcher |

### 35. What is React Router?

React Router is a routing library built on top of React, which is used to create routes in a React application. This is one of the most frequently asked react interview questions.

### 36. Why do we need to React Router?

* It maintains consistent structure and behavior and is used to develop single-page web applications.
* Enables multiple views in a single application by defining multiple routes in the React application.

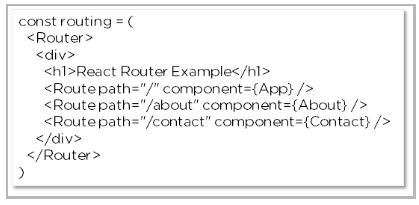
### 37. How is React routing different from conventional routing?

| SN | React Routing | Conventional routing |
| --- | --- | --- |
| 1. | Single HTML page | Each view is a new HTML file |
| 2. | The user navigates multiple views in the same file | The user navigates multiple files for each view |
| 3. | The page does not refresh since it is a single file | The page refreshes every time user navigates |
| 4. | Improved performance | Slower performance |

### 38. How do you implement React routing?

We can implement routing in our React application using this method:

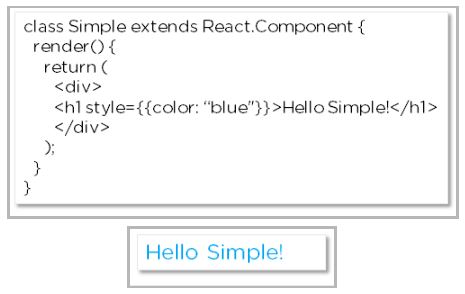
Considering we have the components App, About, and Contact in our application:



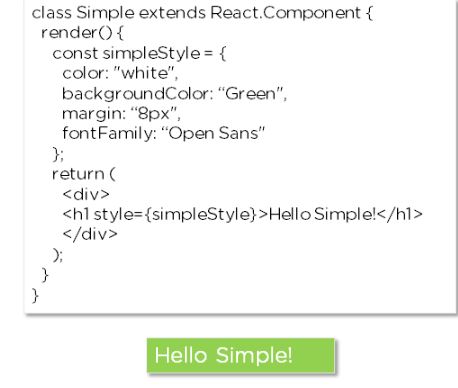
### 39. How do you style React components?

There are several ways in which we can style React components:

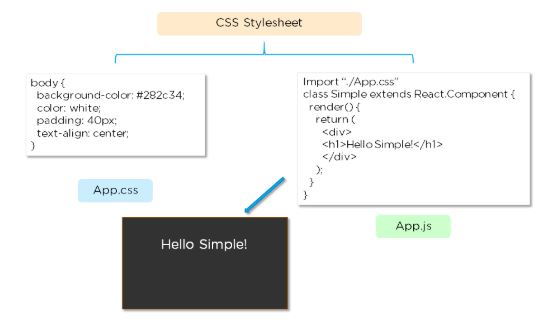
* Inline Styling



* JavaScript Object



* CSS Stylesheet



### 40. Explain the use of CSS modules in React.

* The CSS module file is created with the .module.css extension
* The CSS inside a module file is available only for the component that imported it, so there are no naming conflicts while styling the components.



### ==============

### 1. Differentiate between Real DOM and Virtual DOM.

| Real DOM | Virtual DOM |
| --- | --- |
| 1. It updates slow. | 1. It updates faster. |
| 2. Can directly update HTML. | 2. Can’t directly update HTML. |
| 3. Creates a new DOM if element updates. | 3. Updates the JSX if element updates. |
| 4. DOM manipulation is very expensive. | 4. DOM manipulation is very easy. |
| 5. Too much of memory wastage. | 5. No memory wastage. |

### 

### 2. What is React?

* React is a front-end JavaScript library developed by Facebook in 2011.
* It follows the component based approach which helps in building reusable UI components.
* It is used for developing complex and interactive web and mobile UI.
* Even though it was open-sourced only in 2015, it has one of the largest communities supporting it.

### 3. What are the features of React?

Major features of React are listed below:

1. It uses the virtual DOM instead of the real DOM.
2. It uses server-side rendering.
3. It follows unidirectional data flow or data binding.

### 4. List some of the major advantages of React.

Some of the major advantages of React are:

1. It increases the application’s performance
2. It can be conveniently used on the client as well as server side
3. Because of JSX, code’s readability increases
4. React is easy to integrate with other frameworks like Meteor, Angular, etc
5. Using React, writing UI test cases become extremely easy

### 5. What are the limitations of React?

Limitations of React are listed below:

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

### 6. What is JSX?

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

render(){

return(

<div>

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

</div>

);

}

7. What do you understand by Virtual DOM? Explain its works.

A virtual DOM is a lightweight JavaScript object which originally is just a copy of the real DOM. It is a node tree that lists the elements, their attributes and content as Objects and their properties. React’s render function creates a node tree out of the React components. It then updates this tree in response to the mutations in the data model which is caused by various actions done by the user or by the system. Check out this [Web developer course online](https://www.edureka.co/masters-program/full-stack-developer-training) to learn more about react.

This Virtual DOM works in three simple steps.

1. Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.
2. Then the difference between the previous DOM representation and the new one is calculated.
3. Once the calculations are done, the real DOM will be updated with only the things that have actually changed. 

### 8. Why can’t browsers read JSX?

Browsers can only read JavaScript objects but JSX in not a regular JavaScript object. Thus to enable a browser to read JSX, first, we need to transform JSX file into a JavaScript object using JSX transformers like Babel and then pass it to the browser.

### 9. How different is React’s ES6 syntax when compared to ES5?

require vs import

var React = require('react'); // ES5

import React from 'react'; // ES6

===========

export vs exports  
module.exports = Component; // ES5

export default Component; // ES6

component and function

// ES5

var MyComponent = React.createClass({

render: function() { return <h3>Hello Edureka!</h3>; }

});

// ES6

class MyComponent extends React.Component {

render() { return <h3>Hello Edureka!</h3>; }

}

Props

// ES5

var App = React.createClass({

propTypes: { name: React.PropTypes.string },

render: function() { return <h3>Hello, {this.props.name}!</h3>; }

});

// ES6

class App extends React.Component {

render() { return <h3>Hello, {this.props.name}!</h3>; }

}

State

|  | // ES5  var App = React.createClass({  getInitialState: function() {  return { name: 'world' };  },  render: function() { return <h3>Hello, {this.state.name}!</h3>; }  });  // ES6  class App extends React.Component {  constructor() {  super();  this.state = { name: 'world' };  }  render() { return <h3>Hello, {this.state.name}!</h3>; }  } |
| --- | --- |

### 10. How is React different from Angular?

| TOPIC | REACT | ANGULAR |
| --- | --- | --- |
| *1. ARCHITECTURE* | Only the View of MVC | Complete MVC |
| *2. RENDERING* | Server-side rendering | Client-side rendering |
| *3. DOM* | Uses virtual DOM | Uses real DOM |
| *4. DATA BINDING* | One-way data binding | Two-way data binding |
| *5. DEBUGGING* | Compile time debugging | Runtime debugging |
| *6. AUTHOR* | Facebook | Google |

## 

## 

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

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

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

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

### 13. How can you embed two or more components into one?

We can embed components into one in the following way:

|  | class MyComponent extends React.Component{  render(){  return(  <div>  <h1>Hello</h1>  <Header/>  </div>  );  }  }  class Header extends React.Component{  render(){  return <h1>Header Component</h1>  };  }  ReactDOM.render(  <MyComponent/>, document.getElementById('content')  ); |
| --- | --- |

### 14. What is Props?

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

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

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

### 16. Differentiate between states and props.

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

### 

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

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

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

### 18. What is arrow function in React? How is it used?

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

|  | //General way  render() { return(<MyInput onChange={this.handleChange.bind(this) } /> ); }  //With Arrow Function  render() { return( <MyInput onChange={ (e) => this.handleOnChange(e) } /> ); } |
| --- | --- |

### 19. Differentiate between stateful and stateless components.

| Stateful Component | Stateless Component |
| --- | --- |
| 1. Stores info about component’s state change in memory | 1. Calculates the internal state of the components |
| 2. Have authority to change state | 2. Do not have the authority to change state |
| 3. Contains the knowledge of past, current and possible future changes in state | 3. Contains no knowledge of past, current and possible future state changes |
| 4. Stateless components notify them about the requirement of the state change, then they send down the props to them. | 4. They receive the props from the Stateful components and treat them as callback functions. |

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

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

1. *Initial Rendering Phase:* This is the phase when the component is about to start its life journey and make its way to the DOM.
2. *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.
3. *Unmounting Phase:* This is the final phase of a component’s life cycle in which the component is destroyed and removed from the DOM.

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

Some of the most important lifecycle methods are:

1. *componentWillMount()* – Executed just before rendering takes place both on the client as well as server-side.
2. *componentDidMount()* – Executed on the client side only after the first render.
3. *componentWillReceiveProps()* – Invoked as soon as the props are received from the parent class and before another render is called.
4. *shouldComponentUpdate()* – Returns true or false value based on certain conditions. If you want your component to update, return true else return false. By default, it returns false.
5. *componentWillUpdate()* – Called just before rendering takes place in the DOM.
6. *componentDidUpdate()* – Called immediately after rendering takes place.
7. *componentWillUnmount()* – Called after the component is unmounted from the DOM. It is used to clear up the memory spaces.

### 22. What is an event in React?

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

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

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

### 23. How do you create an event in React?

|  | class Display extends React.Component({  show(evt) { // code  },  render() { // Render the div with an onClick prop (value is a function)  return ( <div onClick={this.show}>Click Me!</div> );  }  }); |
| --- | --- |

### 24. What are synthetic events in React?

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

### 25. What do you understand by refs in React?

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

|  | class ReferenceDemo extends React.Component{  display() {  const name = this.inputDemo.value;  document.getElementById('disp').innerHTML = name;  }  render() {  return(  <div>  Name: <input type="text" ref={input => this.inputDemo = input} />  <button name="Click" onClick={this.display}>Click</button>  <h2>Hello <span id="disp"></span> !!!</h2>  </div>  );  }  } |
| --- | --- |

### 26. List some of the cases when you should use Refs.

Following are the cases when refs should be used:

* When you need to manage focus, select text or media playback
* To trigger imperative animations
* Integrate with third-party DOM libraries

### 27. How do you modularize code in React?

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

|  | //ChildComponent.jsx  export default class ChildComponent extends React.Component {  render() {  return( <div> <h1>This is a child component</h1> </div> );  }  }    //ParentComponent.jsx  import ChildComponent from './childcomponent.js';  class ParentComponent extends React.Component {  render() {  return( <div> <App /> </div> );  }  } |
| --- | --- |

### 28. How are forms created in React?

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

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

### 29. What do you know about controlled and uncontrolled components?

| Controlled Components | Uncontrolled Components |
| --- | --- |
| 1. They do not maintain their own state | 1. They maintain their own state |
| 2. Data is controlled by the parent component | 2. Data is controlled by the DOM |
| 3. They take in the current values through props and then notify the changes via callbacks | 3. Refs are used to get their current values |

## React Interview Questions

### 30. What are Higher Order Components(HOC)?

Higher Order Component is an advanced way of reusing the component logic. Basically, it’s a pattern that is derived from React’s compositional nature. HOC are custom components which wrap another component within it. They can accept any dynamically provided child component but they won’t modify or copy any behavior from their input components. You can say that HOC are ‘pure’ components.

### 31. What can you do with HOC?

HOC can be used for many tasks like:

* Code reuse, logic and bootstrap abstraction
* Render High jacking
* State abstraction and manipulation
* Props manipulation

### 32. What are Pure Components?

*Pure* components are the simplest and fastest components which can be written. They can replace any component which only has a render(). These components enhance the simplicity of the code and performance of the application.

### 33. What is the significance of keys in React?

Keys are used for identifying unique Virtual DOM Elements with their corresponding data driving the UI. They help React to optimize the rendering by recycling all the existing elements in the DOM. These keys must be a unique number or string, using which React just reorders the elements instead of re-rendering them. This leads to increase in application’s performance.

### 34. What were the major problems with MVC framework?

Following are some of the major problems with MVC framework:

* DOM manipulation was very expensive
* Applications were slow and inefficient
* There was huge memory wastage
* Because of circular dependencies, a complicated model was created around models and views

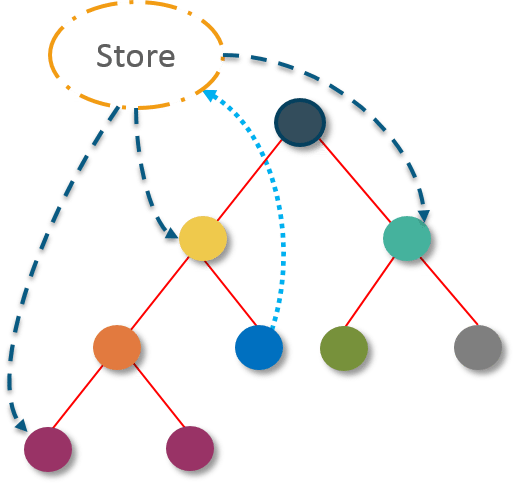
### 35. Explain Flux.

Flux is an architectural pattern which enforces the uni-directional data flow. It controls derived data and enables communication between multiple components using a central Store which has authority for all data. Any update in data throughout the application must occur here only. Flux provides stability to the application and reduces run-time errors.

### 36. What is Redux?

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

### 37. What are the three principles that Redux follows?

1. *Single source of truth:* The state of the entire application is stored in an object/ state tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the application.
2. *State is read-only:* The only way to change the state is to trigger an action. An action is a plain JS object describing the change. Just like state is the minimal representation of data, the action is the minimal representation of the change to that data.
3. *Changes are made with pure functions:* In order to specify how the state tree is transformed by actions, you need pure functions. Pure functions are those whose return value depends solely on the values of their arguments.

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

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

### 39. List down the components of Redux.

Redux is composed of the following components:

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

In case you are facing any challenges with these React interview questions, please comment on your problems in the section below.

### 40. Show how the data flows through Redux?



### 41. How are Actions defined in Redux?

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

|  | function addTodo(text) {  return {  type: ADD\_TODO,  text  }  } |
| --- | --- |

### 42. Explain the role of Reducer.

Reducers are pure functions which specify how the application’s state changes in response to an ACTION. Reducers work by taking in the previous state and action, and then it returns a new state. It determines what sort of update needs to be done based on the type of the action, and then returns new values. It returns the previous state as it is, if no work needs to be done.

### 43. What is the significance of the Store in Redux?

A store is a JavaScript object which can hold the application’s state and provide a few helper methods to access the state, dispatch actions and register listeners. The entire state/ object tree of an application is saved in a single store. As a result of this, Redux is very simple and predictable. We can pass middleware to the store to handle the processing of data as well as to keep a log of various actions that change the state of stores. All the actions return a new state via reducers.

### 44. How is Redux different from Flux?

| Flux | Redux |
| --- | --- |
| 1. The Store contains state and change logic | 1. Store and change logic are separate |
| 2. There are multiple stores | 2. There is only one store |
| 3. All the stores are disconnected and flat | 3. Single store with hierarchical reducers |
| 4. Has singleton dispatcher | 4. No concept of dispatcher |
| 5. React components subscribe to the store | 5. Container components utilize connect |
| 6. State is mutable | 6. State is immutable |

In case you are facing any challenges with these React interview questions, please comment on your problems in the section below.

### 45. What are the advantages of Redux?

Advantages of Redux are listed below:

* Predictability of outcome – Since there is always one source of truth, i.e. the store, there is no confusion about how to sync the current state with actions and other parts of the application.
* Maintainability – The code becomes easier to maintain with a predictable outcome and strict structure.
* Server-side rendering – You just need to pass the store created on the server, to the client side. This is very useful for initial render and provides a better user experience as it optimizes the application performance.
* Developer tools – From actions to state changes, developers can track everything going on in the application in real time.
* Community and ecosystem – Redux has a huge community behind it which makes it even more captivating to use. A large community of talented individuals contribute to the betterment of the library and develop various applications with it.
* Ease of testing – Redux’s code is mostly functions which are small, pure and isolated. This makes the code testable and independent.
* Organization – Redux is precise about how code should be organized, this makes the code more consistent and easier when a team works with it

### 46. What is React Router?

React Router is a powerful routing library built on top of React, which helps in adding new screens and flows to the application. This keeps the URL in sync with data that’s being displayed on the web page. It maintains a standardized structure and behavior and is used for developing single page web applications. React Router has a simple API.

### 47. Why is switch keyword used in React Router v4?

Although a <div> is used to encapsulate multiple routes inside the Router. The ‘switch’ keyword is used when you want to display only a single route to be rendered amongst the several defined routes. The <switch> tag when in use matches the typed URL with the defined routes in sequential order. When the first match is found, it renders the specified route. Thereby bypassing the remaining routes.

### 48. Why do we need a Router in React?

A Router is used to define multiple routes and when a user types a specific URL, if this URL matches the path of any ‘route’ defined inside the router, then the user is redirected to that particular route. So basically, we need to add a Router library to our app that allows creating multiple routes with each leading to us a unique view.

|  | <switch>  <route exact path=&rsquo;/&rsquo;&nbsp;component={Home}/>  <route path=&rsquo;/posts/:id&rsquo; component={Newpost}/>  <route path=&rsquo;/posts&rsquo;&nbsp;&nbsp; component={Post}/>  </switch> |
| --- | --- |

### 49. List down the advantages of React Router.

Few advantages are:

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

### 50. How is React Router different from conventional routing?

| Topic | Conventional Routing | React Routing |
| --- | --- | --- |
| PAGES INVOLVED | Each view corresponds to a new file | Only single HTML page is involved |
| URL CHANGES | A HTTP request is sent to a server and corresponding HTML page is received | Only the History attribute is changed |
| FEEL | User actually navigates across different pages for each view | User is duped thinking he is navigating  across different pages |

51. What is the difference between a controlled and uncontrolled component in React?

In React, a controlled component is a component that has its state controlled by the parent component. The parent component passes the state as props to the controlled component and also handles any changes to the state via callback functions. The controlled component only renders the received props and does not have its own state.

An uncontrolled component, on the other hand, maintains its own internal state and updates it using DOM events. The component directly updates the DOM and does not rely on the parent component to pass and update the state.

An example of a controlled component is a form input that receives its value from the parent component as a prop and updates the parent component’s state via a callback function when the input is changed. An uncontrolled component would be a form input that maintains its own internal state and updates the value directly when the input is changed, without the need for a callback function.

In general, controlled components are considered to be more predictable and easier to debug than uncontrolled components. They also make it easier to implement complex validation and error handling.

52. How do you handle forms in React?

Handling forms in React can be done in a few different ways, but the most common approach is to create a controlled component for the form and its inputs. A controlled component is a component that has its state controlled by the parent component. The parent component passes the state as props to the controlled component and also handles any changes to the state via callback functions.

Here is an example of how to handle a simple form with two input fields (username and password) in a controlled component:

1. First, define the initial state of the form in the parent component’s constructor. For example:

|  | </pre>  constructor(props) {  super(props);  this.state = { username: '' , password: '', };  }  <pre> |
| --- | --- |

1. Next, create callback functions for each input field that updates the corresponding state property when the input value changes. For example:

|  | </span>  handleUsernameChange = (event) =  this.setState({username: event.target.value});  }  handlePasswordChange = (event) => {  this.setState({password: event.target.value});  } |
| --- | --- |

3. Pass the state properties and callback functions as props to the controlled form component. For example:

|  | <Form  username={this.state.username}  password={this.state.password}  handleUsernameChange={this.handleUsernameChange}  handlePasswordChange={this.handlePasswordChange}  /> |
| --- | --- |

1. In the controlled form component, use the passed-in props to set the value and onChange attributes of each input field. For example:

|  | <input type="text" value={props.username} onChange={props.handleUsernameChange} />  <input type="password" value={props.password} onChange={props.handlePasswordChange} /> |
| --- | --- |

1. Finally, in the parent component’s form submit callback function, you can access the form data from the component’s state and handle the form submission as necessary.

Alternatively, you can use third-party libraries such as Formik, or the new hooks in react, useState and useEffect, to handle forms in a more efficient way.

It’s important to note that when you are creating forms in React, you should also validate the input values, and display appropriate error messages to users.

53. Explain the concept of a Higher Order Component (HOC) in React.

In React, a Higher Order Component (HOC) design is used to reuse component logic. It is a function that accepts a component as an argument and outputs a new component that extends the capabilities of the input component. Without having to write duplicate code, HOCs can be used to add shared features, like authentication and data retrieval, to various components. All of the original component’s props as well as any extra props supplied to the HOC are transferred to the wrapped component. HOCs are a potent method for composing and enhancing pre-existing components without changing their original source code.

54. What is the purpose of the mapStateToProps function in Redux?

The mapStateToProps function in Redux is a way for a component to access the current state of the store and update its props accordingly. It does this by taking the current state of the store as an argument and returning an object that maps the state to the props of the component. The returned object is then passed to the component as props, allowing it to access the state and re-render when the state changes. This function is typically defined as a separate function outside of the component and is passed as an argument to the connect function, which is used to connect the component to the store.

55. How do you handle routing in a React application?

In a React application, routing is typically handled using a library such as React Router. React Router allows you to define specific routes for different parts of your application and map them to specific components. When the user navigates to a specific route, the corresponding component is displayed on the page.

For example, you could have a route for the homepage that maps to a “Home” component and a route for a user’s profile that maps to a “Profile” component. When the user navigates to the “/” route, the Home component would be displayed, and when they navigate to the “/profile” route, the Profile component would be displayed.

To use React Router in a React application, you’ll need to install it, import it into your application, and define your routes and the components they map to. Here’s an example of how you might set up React Router in a simple React application:

|  | import { BrowserRouter as Router, Route } from "react-router-dom";  import Home from “./components/Home";  import profile from “./components/Profile”;  function App() {  return (  <Router>  <Route exact path="/" component={Home} />  <Route path="/profile"” component={Profile} />  </Router>  );  } |
| --- | --- |

In this example, the Router component is used to wrap the entire application and the Route component is used to define the specific routes and the components they map to. The exact prop is used to ensure that only the exact path is matched and not any subpaths.

56. Explain the difference between server-side rendering and client-side rendering in React.

In a React application, there are two main ways to render the components: server-side rendering (SSR) and client-side rendering (CSR).

Server-side rendering (SSR) is when the initial render of a React application is done on the server. The server generates the HTML for the initial state of the application and sends it to the browser. When the JavaScript bundle loads, React takes over and the application continues to function as a SPA (Single-Page Application) on the client side.

This approach has a few benefits such as:

Improved performance for search engines and users on slow connections

Faster time-to-first-byte

Better accessibility for users who have JavaScript disabled

Client-side rendering (CSR) is when the React application is rendered entirely in the browser, using JavaScript. The browser requests the JavaScript bundle from the server and then renders the components on the client side. This approach has the benefit of faster load times for users on fast connections and a more responsive user interface.

In general, CSR is the simpler option to implement and more popular, but SSR is a good choice for certain use cases, such as when SEO is a primary concern, or when the app is targeting users on slow internet connections.

It is also worth noting that, it is possible to have a hybrid approach between SSR and CSR which is called isomorphic or universal rendering. This approach allows to leverage the benefits of both SSR and CSR.

57. How do you handle data persistence in a React application?

In a React application, data persistence can be handled using a variety of methods, including:

1. Local storage: This allows you to store key-value pairs in the browser’s local storage, which can be retrieved even after the user closes the browser or restarts their device.
2. Cookies: Cookies are small pieces of data that are stored in the user’s browser and can be accessed by the website on subsequent visits.
3. IndexedDB: It’s a low-level API for client-side storage of large amounts of structured data, including files/blobs.
4. Web SQL Database: This is a deprecated technology for storing data in a client-side database using SQL.
5. Server-side storage: You can also store data on a remote server using an API or a database such as MySQL, MongoDB, etc.
6. Redux or Mobx: State management libraries like Redux or Mobx can be used to manage and persist application state across different components and sessions.

Which one to use depends on your specific use case and requirements.

58. What is the difference between a stateless component and a stateful component in React?

In React, a component can be either stateless or stateful. The main difference between the two is how they manage and update their data.

A stateless component, also known as a “dumb” or “presentational” component, is a component that does not maintain its own internal state. It receives data and callbacks through props (short for properties) and only renders the UI based on those props. Stateless components are typically used for simple, presentational elements that don’t need to handle any complex logic or internal state updates. They are simple functions that take props and return JSX.

A stateful component, also known as a “smart” or “container” component, is a component that maintains its own internal state. It can handle internal state updates and side effects, and may also manage the state of other child components. Stateful components are typically used for more complex elements that need to handle user interactions, API calls, or other logic. They are class components that extend React.Component.

In general, it is recommended to use stateless components as much as possible to keep the application simple and easy to understand. Stateful components should only be used when it is necessary to manage state or handle complex logic.

59. Explain the concept of a Pure Component in React.

A “pure component” in React is a component that updates only when its properties or state have changed. In contrast, a “non-pure component” re-renders each time the parent component re-renders, regardless of whether its props or state have changed. Pure components are more productive since they do not needlessly re-render. By extending React, a component in React can be made pure. React is substituted by PureComponent. Component. This prompts the shouldComponentUpdate method, which decides whether or not to re-render, to provide an automatic shallow comparison of the component’s props and state.

60. How do you handle optimization in a large React application?

There are several techniques that can be used to optimize a large React application:

Use the React Developer Tools to identify and fix performance bottlenecks. The React Developer Tools allow you to track the performance of individual components and identify which components are causing the most re-renders.

Use the shouldComponentUpdate lifecycle method to prevent unnecessary re-renders. This method allows you to control when a component should update based on its props and state.

Use PureComponent and memo instead of Components. These are more efficient alternatives to React.Component that only re-render when props or state have changed.

Use the useEffect hook to handle side effects. This hook allows you to run side effects, such as network requests, after a component has rendered.

Use the useMemo hook to memoize expensive calculations. This hook allows you to cache the results of expensive calculations and only recalculate them when the inputs have changed.

Lazy loading: Lazy loading is a technique where you only load the components that are needed for the current view. This can greatly improve the performance of your application.

Code splitting: Code splitting is a technique where you split your application into smaller chunks of code that are loaded on demand. This can greatly improve the performance of your application.

Optimize the loading time of your application by using techniques like code minification, compression, and caching.

It’s also important to keep in mind that performance optimization is an ongoing process and you should regularly check and optimize your application as it grows.

61. What is the purpose of the combineReducers function in Redux?

The combineReducers function in Redux is used to combine multiple individual reducers into a single root reducer. In a Redux application, the state is managed by a single store and each piece of the state is managed by a specific reducer. The combineReducers function takes an object whose keys correspond to the keys in the state, and whose values are the individual reducers that will manage those parts of the state.

The combineReducers function is used to compose the different reducers that handle different parts of the state into a single root reducer. This root reducer is then passed to the createStore function to create the Redux store.

The combineReducers function is also useful for structuring and organizing your code in a more modular way, as it allows you to separate the logic for different parts of the state into different files and functions.

The combineReducers function is not mandatory to use, but it makes it easier to split the application state and the reducer functions that handle it in a more modular way and also it helps to avoid name collision if you have multiple reducer functions that handle a specific part of the state.

62. How do you handle error handling in a React application?

A React application can handle errors in a few different ways. To handle problems that happen during rendering, one typical solution is to utilise a try-catch block within a component’s lifecycle functions, such as componentDidCatch. Use the Error Boundaries feature to design a component that detects errors that happen inside its child components as an alternative strategy. In addition, you may manage errors that happen inside a functional component by combining the useEffect hook with a try-catch block.

There are several ways that an error can be handled in a React application. Using a try-catch block in a component’s lifecycle routines, like componentDidCatch, is a common way to manage rendering-related issues. An alternate approach is to construct a component that detects problems that occur inside its child components using the Error Boundaries feature. Additionally, you may control errors that occur inside a functional component by combining a try-catch block with the useEffect hook.

63. What is the difference between a smart component and a dumb component in React?

| Smart Component | Dumb Component |
| --- | --- |
| Has state and logic | Has no state or logic |
| Can communicate with other components through props and callbacks | Can only receive props and emit events |
| Can make API calls or perform complex logic | Can only display data passed to it |
| Typically class-based components | Typically functional components |

64. Explain the concept of a Render Prop in React.

A render prop in React is a technique for conveying component logic. Instead of using a component’s props to communicate data and behaviour, a render prop is a function that a component utilises to select what to render. The “provider” component makes the render prop available, but the “consumer” component is the one that uses it. With this approach, component flexibility and reuse are enhanced.

65. How do you handle testing in a React application?

React Testing Library is a great tool for testing React components. It’s a set of helpers that let you test React components without relying on their implementation details. This approach makes refactoring a breeze and also nudges you towards best practices for accessibility. With components, the distinction between a “unit” and “integration” test can be blurry. If you’re testing a form, should its test also test the buttons inside of it? Or should a button component have its own test suite? Should refactoring a button ever break the form test? Jest is a JavaScript test runner that lets you access the DOM via jsdom, which is an approximation of how the browser works. Jest provides a great iteration speed combined with powerful features like mocking modules and timers so you can have more control over how the code executes. It’s a great tool for running tests on React apps. You can also use a tool like BrowserStack’s Real Device Cloud to run end-to-end tests on real devices. Cross browser compatibility testing can also be done with a tool like BrowserStack Live.

66. What are the different ways to pass data between components in React?

Data is passed as a property on the element when utilising props to communicate between components. For instance, you can do the following to send a user object from a parent component to a child component: The props object, props.user, can then be used in the child component to access the data. Data can be passed via the component tree using context rather than having to go through each level of the tree. You must establish a context object containing a Provider and a Consumer in order to use context.

The components that require the data are served by the Provider, and the components that need to access the data are served by the Consumer. Data that is local to a component can be stored in a state, which allows for the tracking of data across time. Through the useState Hook, state in a component can be obtained. The current state value and a function to update the state value are the two items of an array that the useState Hook returns after receiving an initial value.

67. Explain the concept of a Portal in React.

A React component can be rendered outside of the DOM hierarchy of its parent component using React Portals. As an example, you could render components in a modal dialogue box, hover card, loader, or popup message, which would be in a “different place” than their parent component. The ReactDOM.createPortal() method, which accepts a React element (child) and a DOM element as inputs, is used to create React portals (container). The container is the DOM element that the child component should be rendered into, and the child component is any renderable React child, such as an element, string, or fragment. For items that must appear above all other elements, such as profile hovercards, modal dialogue boxes, and tooltips, portals are frequently utilised.

68. How do you handle performance optimization in a React application?

There are several ways to optimize the performance of a React application, including:

* Using the shouldComponentUpdate lifecycle method to prevent unnecessary re-renders of components.
* Using React’s built-in PureComponent or implementing a custom shouldComponentUpdate method to optimize performance for functional components.
* Using the React developer tools to profile the application and identify performance bottlenecks.
* Using the React.memo method for functional components
* Using React’s Context API instead of props drilling.
* Using the useEffect hook to handle side effects in functional components.
* Using the useCallback and useMemo hooks to prevent unnecessary re-renders and improve performance.
* Lazy loading of components and code splitting.
* Minimizing the number of DOM updates by using the key prop when rendering a list of items.
* Using the useReducer hook to manage state updates instead of useState
* Using a virtualized list library like react-virtualized, react-window etc.
* It’s always a good idea to test performance with real-world use cases and user interactions before and after making any optimization.

69. What is the difference between a functional component and a class component in React?

In React, a functional component is a plain JavaScript function that takes in props and returns a React element. A class component is a JavaScript class that extends React.Component and has a render method that returns a React element.

One key difference between the two is that a class component can have local state and lifecycle methods, while a functional component cannot. However, starting with React 16.8, functional components can also have a state using hooks.

Functional components are considered simpler, easier to understand and test, and have better performance than class components. Class components are useful when you need to use lifecycle methods or the local state.

70. Explain the concept of a Context in React.

In React, context is a way to share data that is considered “global” for a component tree. It allows you to pass data through the component tree without having to pass props down manually at every level.

A component that needs to access the context data can consume it by using the useContext hook or the Consumer component. To make the context available to a component, a parent component needs to provide it using the Provider component.

Context is often used for data that is required by many components in an application, such as the currently authenticated user, the current locale, or the theme.

It should be noted that context should be used sparingly, as it can make your components more difficult to reason about and test. If possible, it’s better to pass props down the component tree manually.

71. How do you handle asynchronous data loading in a React application?

In a React application, asynchronous data loading can be handled using a technique called “lifting state up”. This involves moving the state that manages the loading and error state of the data to a common ancestor component, and passing down the necessary callbacks and state through props to the components that need to use the data.

One popular way to handle async data loading is to use the useEffect hook in combination with fetch or a library like axios to load data in a component after it has been rendered. The useEffect hook allows you to synchronize a component with an external system, such as a server, by running a side effect (the data loading) after the component has rendered. The hook takes a callback function that contains the effect, and an array of dependencies.

Another way is to use a library like redux-thunk or redux-saga to handle the async request and store the data in the store/state. These libraries provide an easy way to handle async actions and keep the component state clean.

In either case, it’s important to keep an eye on the component’s state and update it properly with the loaded data.

A simple example of loading data asynchronously in a React component using useEffect and fetch:

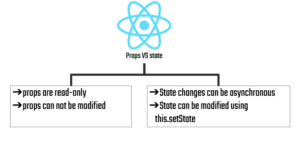
|  | import { useState, useEffect } from ‘react’;  function MyComponent() {  const [data, setData] = usestate(null);  const [error, setError] = usestate(null);  const [loading, setloading] = usestate(true);  useEffect(() => {  async function fetchData() {  try {  const response = await fetch('https://example.com/data’);  const json = await response.json();  setData(json);  setLoading(false);  } catch (error) {  setError(error) {  setLoading(fase);  }  }  fetchData();  }, []);  if (loading) {  return <p>Loading...</p>;  } |
| --- | --- |

|  | if (error) {  return <p>Error: {error.message}</p>;  }  return <p>Data: {JSON.stringify(data)}</p>;  } |
| --- | --- |

It’s important to note that this is a simple example, and in a real-world application you may need to handle more complex cases such as pagination, caching and handling different types of errors.

72. What is the difference between state and props in React?  
State and Props are both concepts in React that are used to store and manipulate data within a React component. The main difference between the two is that State is used to store and manage the data that is local and specific to a component, while Props are used to pass data from a parent component to its child components.

State is considered to be dynamic, meaning that it can change over time as a result of user interactions or other events. On the other hand, Props are considered to be static and cannot be changed by the child component. Instead, the parent component is responsible for updating the value of its Props and passing the updated value to the child component.



In summary, State is used to manage the internal state of a component, while Props are used to pass data from a parent component to its child components.

73. Explain the concept of a Hook in React.  
  
“Hooks are a new feature in React that allows us to add state and other React features to functional components. They were introduced in React 16.8 and have since become a popular way to manage state and side effects in functional components. Hooks are named functions that start with the word use and allow us to reuse stateful logic across components without having to write a class component. For example, the useState Hook allows us to add state to a functional component and the useEffect Hook lets us perform side effects like data fetching or updating the document title. Hooks make our code more reusable, easier to understand, and easier to test.”

74. How do you handle localization in a React application?  
Handling localization in a React application typically involves creating translated versions of your text content and displaying the appropriate version based on the user’s preferred language.

One way to handle localization in a React application is to use a library such as react-i18next. This library provides a set of tools for internationalization and localization, including the ability to define translation keys and their corresponding translations, as well as providing a way to switch between languages at runtime.

To use react-i18next in your React application, you would install it using npm and then configure it in your index.js file. After that, you can use the useTranslation Hook to access the translations in your components.

Here’s how you can explain localization in a React application in an interview:

“Localization in a React application involves creating translated versions of text content and displaying the appropriate version based on the user’s preferred language. To handle localization in a React application, I would use a library such as react-i18next, which provides a set of tools for internationalization and localization. With this library, I would define translation keys and their corresponding translations and provide a way to switch between languages at runtime. In my components, I would use the useTranslation Hook to access the translations and display the appropriate version of the text content.”

75. What is the difference between a static component and a dynamic component in React?  
In React, a static component is a component that is defined with a fixed set of properties or attributes and does not change during its lifecycle. A static component is defined using a simple JavaScript function that returns a tree of elements that represent the component’s UI.

A dynamic component, on the other hand, is a component that can change its properties, state or behavior based on interactions with the user or events that occur within the application. A dynamic component is typically defined using a class component or a functional component with the useState or useEffect Hooks.

Here’s an example of a static component:

|  | function welcome(props) {  return <h1>Hello, {props.name}</h1>;  } |
| --- | --- |

76. Explain the concept of a Renderless Component in React.

A Renderless Component in React is a component that doesn’t render any HTML elements to the DOM, but instead exposes data and methods to other components through props and callbacks. The purpose of a renderless component is to encapsulate logic that can be reused across multiple components and keep the component tree lean and flexible. The other components that consume the logic provided by a renderless component can then render the HTML elements they need based on the information and functionality they receive from the renderless component. This approach separates the logic and presentation concerns, making the code easier to maintain and test.

77. How do you handle server-side rendering in a React application?

Server-side rendering (SSR) in React involves rendering your React components on the server and sending the resulting HTML to the client. This provides a number of benefits, including improved performance and search engine optimization (SEO). To implement SSR in a React application, you can use a library like Next.js or Razzle, which provide an easy-to-use framework for handling SSR. Alternatively, you can use the ReactDOMServer API to manually render your components on the server. The key steps in the process are:

1. Setting up your server to handle incoming requests and render the appropriate components.
2. Rendering the components on the server using ReactDOMServer.renderToString or ReactDOMServer.renderToStaticMarkup.
3. Sending the resulting HTML to the client as part of the response.
4. Hydrating the components on the client so that they can be interactively controlled by the user.

It’s worth noting that SSR comes with some trade-offs and additional complexity, so it’s important to carefully consider whether it’s the right choice for your application.

78. What is the difference between a presentational component and a container component in React?

In React, a presentational component (also known as a dumb component) is a component that focuses on UI (user interface) and presentation of the data, while a container component (also known as a smart component) is a component that focuses on how the data is being managed and provides the data for the presentational components.

A presentational component is typically written as a functional component, receiving data as props and returning a view, while a container component is typically written as a class component, handling data management and state changes, and passing down the data to the presentational components as props.

The separation of concerns between the two types of components allows for better code organization, maintenance, and testing.

79. Explain the concept of a Custom Hook in React.

A Custom Hook in React is a JavaScript function that lets you extract state logic and behavior out of a component, and reuse it across multiple components. Custom Hooks allow you to abstract away state and behavior that is common across your application into a reusable piece of code.

Custom Hooks are named with the prefix use (e.g. useForm, useFetch), and can call other Hooks as well as your own custom Hooks. They have the same rules as Hooks and can only be called at the top level of your component or your own custom Hooks.

Custom Hooks can receive arguments and return values, just like a regular function, but they also have the ability to manage state and perform side-effects. By abstracting state and behavior into a Custom Hook, you can improve the readability and maintainability of your code.

Examples of things you can build with Custom Hooks include:

* Data fetching
* Managing state updates
* Handling form submissions
* Implementing animations and transitions
* And many more.

Using Custom Hooks can make your components cleaner, more reusable, and easier to test, which makes them a powerful tool in your React toolkit.

80. How do you handle accessibility in a React application?

Handling accessibility in a React application involves making sure that your application can be used by as many people as possible, including those with disabilities. This can be achieved through various techniques, including:

1. Semantic HTML: Use semantic HTML elements, such as <button>, <nav>, and <header>, to clearly define the structure and purpose of your content.
2. Accessible Props: Use accessible props, such as aria-label, role, and tabIndex, to provide additional information to assistive technologies, such as screen readers.
3. Keyboard Navigation: Ensure that all functionality can be accessed using a keyboard, and that keyboard focus is managed correctly.
4. Color Contrast: Make sure that the contrast between the text and the background is high enough to be readable by people with color blindness or low vision.
5. Alternative Text: Provide alternative text for images, videos, and other non-text elements to ensure that information is accessible to screen reader users.
6. Screen Reader Testing: Test your application with screen readers and other assistive technologies to identify and fix any accessibility issues.

It is important to note that accessibility is a continuous process and should be considered throughout the development of your React application. The use of tools, such as linting rules and accessibility testing tools, can also help ensure that your application is accessible.

81. What is the difference between a reducer and an action in Redux?

In Redux, a reducer and an action are two different but related concepts.

An action is a plain JavaScript object that describes the change that should be made to the state of the application. It has a type property that defines the type of action being performed, and a payload property that provides any additional data needed to perform the action. Actions are dispatched from the application to the Redux store, which then passes the action to the reducers.

A reducer is a pure function that takes the current state of the application and an action, and returns the next state of the application. The reducer is responsible for handling the actions and updating the state accordingly. It should not perform any side-effects, such as making API calls, but should instead only return the next state.

In summary, actions describe what should change, while reducers define how the state should change in response to the actions.

82. Explain the concept of a Higher Order Component (HOC) in React and when to use it.

A Higher Order Component (HOC) in React is a function that takes a component as an argument and returns a new component with additional props. The purpose of a HOC is to reuse logic across multiple components. An HOC is not a “part” of React, it’s a pattern in React for reusing component logic.

Use a HOC when you need to:

* Share common logic between multiple components, such as data fetching or authorization.
* Abstract state and behavior that can be reused across your application, into a reusable HOC.
* Render a component within another component and pass props to the wrapped component.

Examples of HOCs include the withRouter HOC from react-router and the connect HOC from react-redux.

83. How do you handle data validation in a React application?

Data validation in a React application can be handled in a variety of ways, including:

1. PropTypes: React provides a built-in library called PropTypes that allows you to specify the expected data types for your component’s props. PropTypes will validate the props passed to your component at runtime, and will log warnings in the browser console if any props are of the wrong type.
2. Custom validation functions: You can write custom validation functions to check the validity of your data. These functions can be called inside your component, and can be used to set error messages or update the state to indicate invalid data.
3. Third-party libraries: There are several third-party libraries available for data validation in React, such as yup, joi, or zod. These libraries provide a more powerful and flexible way to validate data, and often provide a more user-friendly way to report errors.

Regardless of the method you choose, it’s important to handle data validation in your React application to ensure that the data being processed is in the correct format and meets the required constraints. Validation helps to catch potential errors early in the development process and prevent bugs from affecting the end-user experience.

84. What is the difference between a synchronous action and an asynchronous action in Redux?

In Redux, an action is a plain JavaScript object that describes the change in the state of the application. Actions can be either synchronous or asynchronous.

A synchronous action is an action that is dispatched and immediately processed by the Redux store. The store updates the state, and the updated state is immediately available for consumption by the components.

An asynchronous action, on the other hand, is an action that is dispatched but takes some time to complete. Asynchronous actions are typically used when performing network requests or doing other operations that take time. These actions cannot be immediately processed by the Redux store, so they require additional logic to handle their completion.

In a Redux application, asynchronous actions are often handled using middleware, such as redux-thunk or redux-saga, which allow for the dispatching of actions that represent the start and completion of asynchronous operations. These middleware provide a way to handle the asynchrony of the operation and ensure that the state is updated appropriately once the operation is complete.

85. Explain the concept of a Virtual DOM in React.

A Virtual DOM (Document Object Model) is a lightweight in-memory representation of the actual DOM in a web page. In React, the Virtual DOM acts as an intermediary between the React component’s render output and the browser’s DOM.

When a React component’s state changes, React updates the Virtual DOM, instead of directly updating the actual DOM. This is more efficient because updating the Virtual DOM is faster than updating the actual DOM, as it can calculate the difference between the previous and current render output, and only update the parts that have changed.

React then takes the updated Virtual DOM and uses it to update the actual DOM, minimizing the amount of work that needs to be done in the actual DOM and improving the overall performance of the application.

In summary, the Virtual DOM in React acts as an optimization to increase the speed and efficiency of updates to the user interface.

86. How do you handle browser compatibility in a React application?

To handle browser compatibility in a React application, you can use various techniques such as:

1. Polyfills: To support older browsers, you can use polyfills, which are JavaScript libraries that emulate missing features in older browsers.
2. Browser detection: You can use libraries like browser-detect to detect the user’s browser and its version, and adjust your code accordingly.
3. Feature detection: Instead of relying on browser detection, you can use feature detection to check if a specific feature is supported by the user’s browser before using it.
4. CSS Reset: You can use CSS resets like normalize.css to make sure that all browsers display the styles in a consistent way.
5. Testing: Regular testing in different browsers and devices is essential to catch any compatibility issues early in the development process.

By using these techniques, you can ensure that your React application runs smoothly across different browsers and devices.

87. What is the difference between a stateful component and a stateless component in React?

In React, a stateful component, also known as a “smart” or “container” component, is a component that maintains its own internal state, typically via the useState or this.state hooks. It may also manage data that is passed down to it as props from other components, and it may use lifecycle methods, such as componentDidMount, to fetch data or perform other side effects.

On the other hand, a stateless component, also known as a “dumb” or “presentational” component, is a component that only receives data via props and does not maintain its own internal state. It simply renders the data it receives in a visually appealing way and does not manage or manipulate it in any way. These components are considered “pure” because they are only concerned with the rendering of the data and do not have side effects.

The key difference between the two is the way they manage and manipulate data. Stateful components have their own internal state and are responsible for managing and updating it, whereas stateless components simply receive data via props and render it without any data manipulation.

88. Explain the concept of a Thunk in Redux.

A Thunk in Redux is a function that returns another function instead of a plain action object. It’s used to perform asynchronous operations and dispatch multiple actions. Thunks allow you to write action creators that return a function instead of an action. This can be useful for performing asynchronous operations, such as API calls, and dispatching multiple actions, such as one to indicate that the API call has started and another to indicate that it has finished. The inner function receives the store’s dispatch method as an argument, which can be used to dispatch actions at any point in the future. Thunks are typically implemented using a middleware, such as the redux-thunk middleware.

89. How do you handle security in a React application?

Handling security in a React application involves multiple steps, including:

Input validation: Validate all user inputs on the client and server side to prevent any malicious data from being processed.

Authenticating and authorizing users: Use a secure authentication mechanism such as JSON Web Tokens (JWT) to ensure that only authorized users can access sensitive data.

Storing sensitive data securely: Do not store sensitive information such as passwords and credit card numbers in local storage, use encrypted storage instead.

Implementing HTTPS: Use HTTPS to ensure secure communication between the client and server and protect against network attacks such as man-in-the-middle attacks.

Keeping dependencies up-to-date: Regularly update React and its dependencies to patch any known security vulnerabilities.

Using Content Security Policy (CSP): Implement a Content Security Policy (CSP) to restrict the types of resources that can be loaded in a React application and prevent cross-site scripting (XSS) attacks.

Regular security audits: Conduct regular security audits to identify and address potential security issues in a timely manner.

90. What is the difference between a function component and a class component in React?

In React, there are two main types of components: function components and class components.

Function Components, also known as “stateless” or “functional” components, are JavaScript functions that accept props as input and return React elements as output. They are simple, easy to understand and test, and are usually used for presentational components that don’t have their own state or lifecycle methods.

Class Components, on the other hand, are JavaScript classes that extend the React.Component base class. They are used for creating components that have a state, or need to access lifecycle methods such as componentDidMount or shouldComponentUpdate. Class components are more complex than function components, but provide more advanced features.

In summary, the main difference between function and class components in React is that function components are simpler, more straightforward, and easier to understand, while class components are more powerful and provide more advanced features, but are also more complex.

91. Explain the concept of a Provider in React-Redux.

The “Provider” in React-Redux is a higher-order component that wraps your React application and provides it with the ability to access the Redux store. It allows you to pass the store down to your components using context, without having to manually pass it down as props through every level of the component tree.

By using the Provider, you ensure that all of your components can subscribe to the store and dispatch actions to modify its state. In other words, the Provider acts as a bridge between your React components and your Redux store, making the store accessible to all components in your application.

92. How do you handle code splitting in a React application?

Code splitting in React can be handled using the following approaches:

Dynamic Imports: Dynamic imports allow you to load a component lazily only when it is needed. This is done using the import() syntax and provides a way to split code into smaller chunks that can be loaded on demand.

|  | import React, { Suspense } from 'react’;  const LazyComponent = React.lazy(() => import('./LazyComponent'));  function App() {  return (  <div>  <Suspense fallback={<div>Loading...</div>}>  <LazyComponent />  </Suspense>  </div>  );  } |
| --- | --- |

|  | import React, { lazy, Suspense } from 'react’;  import { Route } from 'react-router-dom’;  const Home = lazy(() => import('./Home'));  const About = lazy(() => import('./About'));  function App() {  return (  <div>  <Suspense fallback={<div>Loading...</div>}>  <Route exact path="/" component={Home} />  <Route path="/about" component={About} />  </Suspense>  </div>  );  } |
| --- | --- |

Webpack Bundle Analyzer: This is a tool that provides a visual representation of the code and its size. You can use this tool to identify the large chunks of code that can be split into smaller chunks and loaded lazily.

By using these approaches, you can effectively handle code splitting in a React application and improve its performance by reducing the initial loading time and only loading the required code on demand.

93. What is the difference between a connected component and a component in React-Redux?

| Component | Connected Component (Higher Order Component) |
| --- | --- |
| Definition | A plain React component that receives props and returns a tree of React elements. |
| Usage | Used to display UI elements, manage local component state and pass props to child components. |
| Example | A button, a form, a card, etc. |

Connected components are higher-order components that are wrapped around plain components to provide them access to the Redux store. Connected components are used to access the state of the store and dispatch actions, whereas plain components are used to manage UI elements and local component state.

94. Explain the concept of a Sagas in Redux.

A Saga in Redux is a way to manage side effects (e.g. asynchronous operations like data fetching and impure operations like accessing the browser cache) in a Redux application. It is implemented as a middleware using generator functions in JavaScript and runs in the background, separate from the main thread of your application, watching for actions dispatched to the store. When a specific action is detected, the Saga can perform various tasks and trigger additional actions as needed, updating the store based on the results of the asynchronous operations. The key benefit of using Sagas is that they make it easier to reason about, test, and manage the flow of data in your application.

95. How do you handle code optimization in a large React application?

Handling code optimization in a large React application can be achieved through several approaches:

1. Code splitting: This allows you to split your code into smaller chunks that can be loaded on demand, reducing the initial load time of your application.
2. Lazy loading: Lazy loading allows you to load components only when they are required, reducing the amount of code that needs to be loaded and parsed at startup.
3. Use of a bundler such as Webpack: A bundler can help you optimize your code by reducing the size of your JavaScript files, combining multiple files into one, and more.
4. Use of caching: You can cache the data and components that are frequently used in your application to avoid fetching the same data over and over.
5. Use of efficient algorithms and data structures: In order to keep your application fast, it’s important to use algorithms and data structures that are optimized for performance.
6. Regular performance monitoring and profiling: Regular performance monitoring and profiling can help you identify performance bottlenecks and areas for improvement in your code.
7. Use of optimization techniques such as memoization: By using techniques such as memoization, you can reduce the number of unnecessary re-renders and computations in your application, improving its overall performance.

96. What is the difference between a React component and a React element?

A React component is a JavaScript class or function that returns a React element. It is a reusable piece of UI that describes a part of the user interface.

A React element, on the other hand, is a plain JavaScript object that represents a DOM node. It is an immutable representation of a DOM node, which can be created using React.createElement or JSX.

In short, a component is a blueprint for creating elements, and an element is an instance of a component.

97. Explain the concept of a Middleware in Redux.

In Redux, a middleware is a software component that sits between the store and the action dispatching process to add additional functionality, such as logging, crash reporting, handling asynchronous actions, etc. It allows you to extend the store’s behavior without modifying the store itself. Middlewares are applied using the applyMiddleware method and can be composed together to achieve a desired behavior. When an action is dispatched, it passes through each middleware in the order they were composed, giving the middleware an opportunity to interact with the action before it reaches the store. This provides a way to manipulate actions and state, and to perform complex actions that can span multiple actions.

98. How do you handle internationalization in a React application?

Handling internationalization (i18n) in a React application involves adapting the user interface and content of the application to meet the language and cultural requirements of different locales.

There are several libraries and techniques that can be used to implement internationalization in a React application, including:

1. react-intl: A popular library for internationalizing React applications. It provides components for formatting dates, numbers, and strings, as well as handling pluralization and message extraction.
2. Context API: React’s Context API can be used to store the current locale and make it available to the components that need it. The locale can be changed dynamically to switch the language of the application.
3. Custom hooks: Custom hooks can be written to encapsulate the logic for formatting and retrieving messages, and to make it easier to use in components.

Here’s an example of how the react-intl library can be used to implement internationalization in a React application:

|  | import React from 'react';  import { FormattedMessage, useIntl } from ‘react-intl';  function MyComponent() {  const intl = useintl();  return (  <div>  <p>  <FormattedMessage id="greeting" defaultMessage="Hello, World!" />  </p>  <p>  {int1.formatDate(new Date(), {  weekday: 'long’,  year: ‘numeric’,  month: 'long’,  day: ‘numeric’,  })}  </p>  </div>  );  } |
| --- | --- |

In this example, the useIntl hook is used to access the intl object, which provides internationalization functions like formatDate. The FormattedMessage component is used to display a localized message with the ID greeting.

Implementing internationalization in a React application can greatly improve the user experience for users who speak different languages and are located in different regions. It’s an important consideration for any application that aims to have a global reach.

99. What is the difference between a React component and a React class?

When it comes to an interview, it’s important to understand the difference between React class components and functional components. This knowledge can demonstrate your understanding of React and its components, and it may also show your ability to write efficient and maintainable code.

When asked about class components, you can highlight that they are defined as JavaScript classes that extend the React.Component class, have a render method, and can have additional lifecycle methods and state. You can also provide a simple example to show your understanding of class components.

When asked about functional components, you can emphasize that they are defined as plain JavaScript functions that return the component’s JSX markup, and that they can use state and other React features with hooks. You can also give an example to show how to write a functional component that achieves the same functionality as a class component.

Finally, you can explain the trade-offs between class components and functional components, such as that functional components are generally simpler and easier to read, while class components offer more features and flexibility. Showing your ability to weigh the pros and cons of each approach can demonstrate your critical thinking skills and ability to write maintainable code.

100. Explain the concept of a Memoization in React.

In React, memoization is a technique used to optimize the performance of a component by avoiding unnecessary re-renders. It involves caching the results of a component’s render so that if the inputs (props) to the component do not change, the cached result can be reused, instead of re-computing the result.

React provides a built-in hook called useMemo for implementing memoization. useMemo takes a function and an array of dependencies as arguments and returns a memoized value. The function is re-executed only if one or more of the dependencies have changed.

Here is an example of how useMemo can be used to memoize an expensive calculation:

|  | import React, { useMemo } from 'react’;  function MyComponent({ data }) {  const memoizedvalue = useMemo(() => {  // Do some expensive calculation  let result = 0;  for (let i = 0; i < data.length; i++) {  result += data[i];  }  return result;  }, [data]);  return <div>The result is {memoizedvalue}</div>;  } |
| --- | --- |

In this example, useMemo is used to memoize the result of the expensive calculation performed on data. The calculation will only be re-executed if the value of data changes.

Memoization can greatly improve the performance of a React application by avoiding unnecessary re-renders and re-calculations, especially when dealing with complex or large data structures.

101. How do you handle events in React?

When building a React application, it’s common to need to respond to user actions such as button clicks, form submissions, and other events. In React, this is achieved through the use of event handlers. An event handler is a callback function that is attached to an element in the UI, and it’s executed when a specified event occurs.

For example, to handle a click event on a button, you would define a function in your React component that updates the component’s state, and then attach that function to the button as an onClick event handler.

|  | class MyComponent extends React.Component {  constructor(props) {  super(props);  this.state = {count: 0};  }  handleclick() {  this.setstate({count: this.state.count + 1});  }  render() {  return (  <div>  <p>You clicked {this.state.count} times</p>  <button onClick={() => this.handleClick()}> Click me </button>  </div>  );  } |
| --- | --- |

In this example, when the button is clicked, the handleClick function is called and the component’s state is updated. This causes a re-render of the component, and the displayed count is updated accordingly.

Event handlers are a key part of React’s event handling system and are used to add interactivity to your application.

Quizzing-application-300x175.png

=========

1. What is one of the core types in React?

A Component is one of the core building blocks of React. React allows us to pass information to components using things called props (short for properties). Because React comprises several components, props make it possible to share the same data across the components that need them.

2. What do you mean by the state?

The state is an instance of React Component Class can be defined as an object of a set of observable properties that control the behavior of the component. In other words, the State of a component is an object that holds some information that may change over the lifetime of the component.

3. What is redux?

Redux is a knowable state container built for JavaScript apps. It allows managing the application state, and it stands to be open source. Applications that run consistently help run in different environments (client, server, and native) and are easy to test.

4. Is it possible to display props on a parent component?

First pass the data from the child to the parent, as an argument into a callback from the parent. Set this incoming parameter as a state on the parent component, and then pass it as a prop to the other child (see above example). The sibling can then use the data as a prop.

5. In ReactJS, why there is a need to capitalize on the components?

Capitalized types indicate that the JSX tag is referring to a React component. These tags get compiled into a direct reference to the named variable, so if you use the JSX Foo/ expression, Foo must be in scope.

6. What do you know about synthetic events?

Synthetic events are the wrapper that React uses to standardize event functionality across browsers. It is important to remember that events are not part of core Java script. Instead, they come from each browser's own Java script API — meaning that how browsers handle events will differ.

7. Explain DOM diffing?

Once React knows which virtual DOM objects have changed, then React updates only those objects, in the real DOM. This makes the performance far better when compared to manipulating the real DOM directly. This makes React standout as a high performance JavaScript library.

8. Is it possible to nest JSX elements into other JSX elements?

It is possible to nest JSX elements. 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.

9. List some of the major advantages of React.

· Easy to Learn, Easy to Use.

· Reusable Components, Great Developer Tools.

· The Virtual DOM, the Ecosystem.

· It's Easier to Write with JSX.

10. What are the limitations of React?

React Technology accelerates so fast so that it cannot make proper documentation of the project. So, the developer tries to write the instruction on its own. React focus on the view part of MVC i.e. UI of the web application.

11. What is JSX?

JSX is an XML/HTML-like syntax used by React that extends ECMA Script so that XML/HTML-like text can co-exist with JavaScript/React code. JSX allows us to put HTML into JavaScript.

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

The virtual DOM (VDOM) is a programming concept where an ideal, or “virtual”, representation of a UI is kept in memory and synced with the “real” DOM by a library such as React DOM. Since “virtual DOM” is more of a pattern than a specific technology, people sometimes say it to mean different things.

13. Why can’t browsers read JSX?

Browsers cannot read JSX because there is no inherent implementation for the browser engines to read and understand it. You can use label to transform your JSX into native JavaScript and HTML which browser can understand.

14. How is React different from Angular?

Angular JS is a structural framework for developing dynamic web apps, whereas React is a JavaScript library that allows you to build UI components. Angular JS is based on MVC (Model View Controller) whereas React is based on Virtual DOM. Angular is based on Typescript and React is based on JavaScript.

15. What do you understand from “In React, everything is a component.”

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

16. Explain the purpose of render() in React.

render() function is used to update the UI. For this, you have to create a new element and send it to React DOM. 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.

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

The state is an instance of React Component Class can be defined as an object of a set of observable properties that control the behavior of the component. JS import React, { Component} from 'react'; class Example extends Component { constructor(props) { super(props); } render() { return ( ... ) } } Adding our state object is easy enough. Inside the constructor, after super(props); , just add this. state and set it equal to an empty object.  
  
18. What are the different phases of React component’s lifecycle?

A React Component can go through four stages of its life as follows.

· Initialization: This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.

· Mounting: Mounting is the stage of rendering the JSX returned by the render method itself.

· Updating: Updating is the stage when the state of a component is updated and the application is repainted.

· Unmounting: As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

19. What is an event in React?

An event is an action that could be triggered as a result of the user action or system generated event. For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.

20. How do you create an event in React?

Handling events with React elements is very similar to handling events on DOM elements. There are some syntax differences: React events are named using camelCase, rather than lowercase. With JSX you pass a function as the event handler, rather than a string.

==============

1. Can you list the main differences between React and React Native?

Platform: React is a library for building user interfaces in web applications, whereas React Native is a framework for building native mobile applications for iOS, Android, and other platforms.

Language: React uses JavaScript, while React Native uses a combination of JavaScript and a subset of CSS called Flexbox for styling.

Components: Both React and React Native use components as the building blocks of user interfaces, but the components in React are designed for web browsers, while the components in React Native are designed for mobile devices and can include native UI elements such as View, Text, Image, etc.

Styling: React uses CSS for styling, while React Native uses a subset of CSS called Flexbox for styling, along with some additional styling options specific to mobile applications.

UI/UX: React is designed to create web-based user interfaces, while React Native is designed to create mobile user interfaces that look and feel like native applications, providing a more native-like user experience.

Performance: React Native uses native UI components, which are rendered directly by the device's GPU. This makes it more efficient in terms of performance compared to React, which relies on a virtual DOM to update the UI.

Platform-specific APIs: React Native provides access to native APIs of the underlying mobile platform, such as the camera, geolocation, etc., allowing developers to create more sophisticated and feature-rich mobile applications compared to React, which primarily focuses on web-based UI.

Development workflow: React Native requires a different development setup compared to React, as it involves building, compiling, and running native code on mobile devices, whereas React can be developed and run in a web browser.

Code Reusability: One of the main advantages of React Native is its ability to reuse a significant portion of code between different platforms (iOS, Android, etc.), which can result in faster development and easier maintenance. Conversely, React components are typically designed specifically for web browsers and may not be easily reusable on other platforms.

Ecosystem: While React has a mature ecosystem with a large number of libraries, tools, and resources available for web development, React Native has its own ecosystem tailored for mobile app development, which includes libraries, components, and tools specific to mobile platforms.

These are some of the key differences between React and React Native. While they share similarities, they are designed for different platforms and have their own strengths and use cases. Developers should choose the one that best fits their specific requirements and target platform.

2. How can animation for React and React Native be implemented? Do both have the same paths or are there differences?

The main difference in implementing animations between React and React Native is in how they handle rendering and updating of components.

In React, which is used for building web applications, animations are typically implemented using CSS transitions or animations. CSS animations are processed by the browser's rendering engine, which updates the visual display of components based on changes in CSS properties. React updates the DOM based on the changes in component state or props, and the CSS animations are applied accordingly. The DOM (Document Object Model) is a tree-like structure that represents the structure of HTML elements in the web page.

In React Native, which is used for building mobile applications, animations are implemented using a different approach that leverages the native capabilities of the underlying platform. React Native uses a bridge to communicate between the JavaScript code and the native components rendered on the screen. When an animation is triggered in React Native, it is processed natively on the device without involving the DOM or CSS. This allows for more efficient and smoother animations, as they are directly handled by the native rendering engine of the platform (e.g., iOS or Android) and take advantage of the platform's hardware acceleration.

Another key difference is that React Native uses a different set of components and styling approaches compared to React. React Native provides a set of pre-built native components that are optimized for mobile applications, such as View, Text, and Image, which have different behavior and styling compared to the HTML elements used in React.

Additionally, React Native uses Flexbox for layout, which is similar to CSS Flexbox but has some differences in syntax and behavior.

Overall, while React and React Native share similar concepts for implementing animations, they differ in the underlying mechanisms and approaches due to the platform-specific nature of web and mobile development. It's important to understand these differences when implementing animations in React or React Native to ensure optimal performance and a smooth user experience.

For React, which is used for building web applications, there are several libraries and approaches available for implementing animations.

Some popular libraries include:

React Spring: A physics-based animation library that provides a simple and intuitive API for animating components in React. It supports various types of animations, such as transitions, keyframes, and parallax effects.

React Transition Group: A library that allows you to define transition animations for components as they enter or exit the DOM. It provides a way to add CSS classes to components during transitions, which can be used to define animations using CSS animations or transitions.

CSS Animations and Transitions: You can also implement animations directly using CSS animations or transitions in React by applying CSS classes to components or using inline styles. This approach gives you more control over the animation details but requires manual handling of animation states and timing.

For React Native, which is used for building mobile applications, there are some differences due to the use of native components and the absence of a DOM.

Some popular libraries for animations in React Native include:

React Native Animated: A built-in module in React Native that provides a declarative API for creating animations. It uses a similar syntax to CSS animations and transitions but is optimized for performance on mobile devices.

Lottie React Native: A library that allows you to use Adobe After Effects animations in React Native. It provides a way to animate complex animations using JSON files created in After Effects.

React Native Reanimated: A library that allows you to create complex animations and gestures using a declarative API. It provides a high-performance way to implement animations in React Native and is particularly useful for animations with complex logic and interactions.

3. You have a web application developed with React. The client wants to adapt this app for Android and iOS using React Native. List the code-related differences you will encounter.

Platform-specific components: React Native has its own set of components that are different from the ones used in React. For example, in React Native, you'll use View instead of div, Text instead of span, and Touchable components instead of regular HTML buttons or links.

Styling: Styling in React Native uses a different syntax than web-based CSS. Instead of using CSS classes and styles, you'll use a subset of CSS called Flexbox to define layouts in React Native. Additionally, some styles that are available in web-based CSS may not be available in React Native and vice versa.

Navigation: Navigation in React Native is different from web-based navigation. React Native uses its own navigation libraries, such as react-navigation, which provides a set of navigation components and APIs specifically designed for mobile apps, including stack, tab, and drawer navigation.

Native modules: React Native allows you to write native modules in platform-specific languages (Java/Kotlin for Android, Objective-C/Swift for iOS) to interact with native functionalities that are not available through JavaScript. This means you may need to write native modules for certain features, such as accessing the device's camera or handling push notifications.

Platform-specific code: While a majority of the code in React Native is shared between Android and iOS, there may be cases where you need to write platform-specific code — for example, when handling permissions, dealing with platform-specific UI patterns, or implementing platform-specific APIs.

Performance: Performance considerations may be different in React Native compared to web-based React. React Native apps need to render UI components natively on the device, which can have performance implications, especially when dealing with complex UI and animations. Optimizing performance in React Native may require different techniques than those used in web-based React apps.

Testing: Testing in React Native may require different tools and approaches compared to testing web-based React apps. For example, you may need to use native testing frameworks and emulators/simulators for Android and iOS, as well as specific testing libraries for React Native components and navigation.

Deployment: Deploying a React Native app involves different processes compared to deploying a web-based React app. For example, you'll need to build separate binary files for Android (APK) and iOS (IPA) platforms and submit them to their respective app stores (Google Play Store for Android, Apple App Store for iOS), following their guidelines and requirements.

Device APIs: While React Native provides a large set of cross-platform APIs, there may be cases where you need to access platform-specific APIs or features that are not available in React Native. In such cases, you may need to write native modules or use third-party libraries to access those APIs.

Development environment: Setting up the development environment for React Native may require additional steps compared to web-based React. For example, you may need to install platform-specific dependencies, such as Java Development Kit (JDK) for Android and Xcode for iOS, as well as configure emulators/simulators for testing.

These are some of the key differences you may encounter when developing a mobile app using React Native compared to a web application developed with React. While React Native allows code reuse and cross-platform development, it's important to consider the platform-specific differences and requirements to ensure a smooth transition from web to mobile development.

4. Convince the customer to switch from native mobile app development to cross-platform one with React Native.

As a team that specializes in React Native development, there are several ways you can effectively convince a client to use cross-platform technology for their mobile application, even if they initially express a preference for native development. Here are some potential strategies:

Cost-effectiveness: One of the key advantages of using cross-platform technology like React Native is that it allows for code-sharing across multiple platforms, such as iOS and Android. This can result in significant cost savings compared to developing separate native applications for each platform. You can explain to the client that by using React Native, they can achieve cost-effectiveness in their mobile app development, as they will only need to maintain a single codebase for both platforms, reducing development time and costs.

Faster development: React Native enables faster development cycles, as it allows for hot-reloading, which means developers can see the changes in the app in real time without having to recompile the entire application. This can lead to quicker iterations and faster deployment of updates and improvements to the app. You can highlight to the client that by using React Native, their app can be developed and updated more quickly, giving them a competitive edge in the market.

Wide platform support: React Native has a large and active community of developers, and it is constantly updated and improved by Facebook and the community. It also supports a wide range of platforms, including iOS, Android, and even web and desktop platforms. This means that by using React Native, the client can have their app running on multiple platforms with a single codebase, which can greatly extend their app's reach and user base.

Native-like performance: React Native uses a bridge to communicate between JavaScript code and native modules, which allows it to achieve performance close to that of native apps. While there may be some performance differences compared to fully native apps, the performance gap has significantly reduced over the years, and React Native apps can provide a smooth and responsive user experience. You can reassure the client that React Native apps can deliver a high-quality, native-like performance that meets the expectations of their users.

Skilled development team: If your team specializes in React Native development, you can highlight the expertise and experience of your developers in building cross-platform applications using React Native. You can showcase your team's portfolio of successful React Native projects, demonstrate their skills and expertise in working with the technology, and provide testimonials or references from satisfied clients. This can instill confidence in the client that your team has the necessary expertise to deliver a high-quality, cross-platform mobile application.

Flexibility and future-proofing: Cross-platform technologies like React Native offer flexibility in terms of code reusability and adaptability to changing technology trends. By using React Native, the client can future-proof their application, as it allows for easier updates and modifications to adapt to evolving user needs and technological advancements. You can explain to the client that using React Native will provide them with the flexibility to make changes and improvements to their app in the future, all without having to rewrite the entire codebase.

Support and community: React Native has a large and supportive community of developers, with extensive documentation, forums, and resources available for support. By using React Native, the client can leverage this community and avail themselves of the vast knowledge base and support available. This can significantly aid in resolving issues, addressing challenges, and ensuring a successful mobile app development project.

In summary, by highlighting React Native’s cost-effectiveness, faster development cycles, wide platform support, native-like performance, skilled development team, flexibility, and community support, you can effectively convince a client to consider using React Native or other cross-platform technologies for their mobile app development, even if they initially express a preference for native development.

5. What difficulties might a junior React developer face when switching to React Native?

Switching from React to React Native can be an exciting opportunity for a junior developer, but it also comes with its own set of challenges. Here are some potential difficulties your junior React developer may face when transitioning to React Native:

Platform-specific knowledge: React Native allows for building mobile applications for both iOS and Android platforms. However, it requires developers to have an understanding of platform-specific concepts and APIs, such as navigation, styling, and performance optimization, which may differ from React. Your junior developer may need to learn and adapt to these platform-specific nuances.

Native module development: React Native allows developers to write platform-specific native modules using native code (Java/Kotlin for Android, Objective-C/Swift for iOS) to interact with device features like the camera, sensors, or other native functionality. Your junior developer may need to learn how to write and integrate native modules, which can involve dealing with platform-specific languages and tools.

UI/UX considerations: While React and React Native share similar concepts for building user interfaces, there are differences in how components are styled and rendered in React Native due to the constraints of mobile platforms. Your junior developer may need to learn new UI/UX patterns, such as using Flexbox for layout, handling touch events, and optimizing performance for mobile devices.

Device testing and debugging: Unlike web development, React Native involves building apps for physical devices, which may have different hardware capabilities and performance characteristics. Your junior developer may need to learn how to test and debug on physical devices, understand performance optimizations, and handle device-specific issues such as screen sizes, touch gestures, and orientation changes.

Third-party libraries: React Native has a rich ecosystem of third-party libraries, but not all React libraries are compatible with React Native. Your junior developer may need to find and learn new libraries specifically designed for React Native, which may have different APIs and usage patterns compared to React.

Project setup and configuration: Setting up a React Native project involves configuring the development environment, installing native dependencies, and managing project dependencies. This process can be different from setting up a React project and may require additional setup steps, such as installing Android Studio or Xcode, configuring emulators/simulators, and linking native modules.

Performance considerations: Mobile apps have stricter performance requirements compared to web apps due to limited resources on mobile devices. Your junior developer may need to understand and optimize performance considerations in React Native, such as reducing UI re-renders, optimizing image loading, and handling memory efficiently.

It's important to provide your junior developer with the proper support, resources, and guidance during their transition to React Native. Encouraging them to learn from official documentation, online tutorials, and seeking help from experienced React Native developers can help them overcome these challenges and become proficient in mobile development with React Native.

6. You need to develop a project within 3 months. Convince your team to use the Expo framework.

Faster development: Expo provides a set of pre-built UI components, libraries, and tools that can significantly speed up the development process. It offers a wide range of built-in functionality, such as camera access, push notifications, and geolocation, which can save time and effort in writing custom code.

Simplified workflow: Expo provides an all-in-one development environment that includes not only the development tooling but also a cloud-based build service for compiling your app in the cloud. This makes it easier to set up and manage your development environment, reducing the need for complex configurations and installations.

Better developer experience: Expo offers a wide range of developer-friendly features, such as hot module replacement (HMR) for fast code updates, a comprehensive set of debugging tools, and an easy-to-use interface for managing dependencies. This can improve the overall development experience and help developers be more productive.

Robust community and ecosystem: Expo has a large and active community of developers, which means you can find plenty of resources, tutorials, and support online. Expo also has a rich ecosystem of third-party libraries and integrations, which can further enhance the functionality and capabilities of your app.

Over-the-Air (OTA) updates: One of the unique features of Expo is its ability to deliver OTA updates to your app without requiring users to download and install a new version from the app store. This can be a significant advantage in terms of quickly delivering bug fixes, improvements, and new features to your users without the need for app store reviews and approvals.

Seamless publishing process: Expo provides a simplified and streamlined process for publishing your app to app stores. It automates many of the manual steps required for publishing, such as generating app icons, managing certificates, and handling app store metadata, making it easier and more efficient to publish your app to both the Apple App Store and Google Play Store.

These are some compelling arguments that you can use to highlight the advantages of using the Expo framework for your project. It's important to evaluate your specific project requirements and discuss things with your team to determine if Expo is the right fit for your needs.

7. How can you develop a secure mobile banking application using React Native?

Use HTTPS: Ensure that all communication between the mobile app and the backend server is encrypted using HTTPS (TLS/SSL) to protect data in transit from interception or eavesdropping.

Secure authentication: Implement secure authentication practices, such as using multi-factor authentication (MFA), employing strong password policies, and having token-based authentication. Avoid storing sensitive authentication data, such as passwords, locally on the device.

Secure data storage: Use secure storage options provided by React Native, such as AsyncStorage, Keychain (iOS), and SharedPreferences (Android) to store sensitive data such as user credentials, tokens, and other such information securely on the device.

Code and library review: Regularly review and update the codebase and third-party libraries used in your React Native app to ensure they are free from known security vulnerabilities.

Input validation: Implement proper input validation techniques to prevent attacks such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

Secure communication: Use secure communication protocols, such as SSL/TLS, for all network requests made from the mobile app to the backend server to protect against man-in-the-middle (MITM) attacks.

Offline data protection: Implement encryption and other security measures to protect sensitive data stored locally on the device, especially in offline scenarios where data may be cached or stored for later use.

Use libraries with good security practices: Utilize well-known libraries and packages with a strong track record of security practices and regular updates. Avoid using outdated or deprecated libraries that may have known security vulnerabilities.

Regular security testing: Conduct regular security testing, including penetration testing and vulnerability assessments, to identify and address potential security weaknesses in your mobile app.

Keep up with security best practices: Stay updated with the latest security best practices and guidelines for mobile app development, including those specific to React Native, and follow security recommendations from the React Native community, as well as industry standards and regulations.

Remember, security is an ongoing process, and it's important to continually monitor and update your mobile banking application to address any emerging security threats or vulnerabilities.

8. What factors should be considered when choosing between React Native and Flutter for specific projects?

When choosing between React Native and Flutter for specific projects, consider the following factors:

Project requirements: Consider the specific requirements of your project, such as the complexity of the UI, performance needs, and integration with existing technologies. React Native is a good choice if you want to leverage existing web development skills and have a large developer community, while Flutter may be better for projects that require high-performance UI and smooth animations.

Team skill-set: Evaluate the skill set of your development team. If your team is already proficient in JavaScript, React Native may be a more natural fit, as it uses JavaScript and popular web development concepts. On the other hand, if your team has experience with Dart or object-oriented programming, Flutter may be easier to adopt.

Platform support: Consider the platforms you need to target. React Native is primarily focused on building mobile apps for iOS and Android, while Flutter also supports building apps for web, desktop, and embedded devices. If you need to target multiple platforms, Flutter may provide a more unified development experience.

Development speed: React Native allows for faster development cycles, as it supports hot reloading, allowing developers to see changes in real time. Flutter also has a hot reload feature, but it may require a longer build time. Consider the development speed requirements of your project and the preferences of your team.

Community and ecosystem: Both React Native and Flutter have active communities and ecosystems with extensive libraries, tools, and documentation. Consider the availability and quality of third-party libraries and resources for the specific features you need in your project.

Maintenance and support: Consider the long-term maintenance and support requirements of your project. React Native has been around for longer and has a larger community, which means it may have more mature libraries and resources, but it also has a higher chance of facing breaking changes in the future. As a younger framework, Flutter may have a smaller community but benefits from the support of Google, which provides regular updates and improvements.

Ultimately, the choice between React Native and Flutter should be based on the specific needs and requirements of your project, the skill set of your team, and the preferences of your stakeholders. Both frameworks have their strengths and weaknesses, so it's important to carefully evaluate the factors that are most relevant to your project before making a decision.

9. Choose between Vue, React, and Angular for a medium-sized e-commerce project that will last for 8 months.

When choosing a suitable framework, consider the following factors:

Familiarity and expertise of the development team: Consider the experience and skill set of your development team. If your team is already familiar with a particular framework, it might be more efficient and productive to stick with that framework. For example, if your team has experience with React, it may make sense to choose React for your project to leverage their existing expertise.

Project requirements and complexity: Consider the specific requirements and complexity of your e-commerce project. Different frameworks have different features, ecosystems, and architectural patterns that may be better suited for specific use cases. For example, if you need a high degree of flexibility and customization, Vue might be a good choice due to its incrementally-adoptable nature. If you need a highly efficient and scalable solution with a robust structure and tooling, Angular might be suitable. If you need a lightweight and fast solution, React’s component-based architecture might be a good fit.

Community and ecosystem: Consider the size and activity of the community and ecosystem around each framework. A larger community generally means better community support, extensive documentation, and a wide range of third-party libraries and plugins, which can be helpful during development. React and Angular have larger communities compared to Vue, but Vue has been gaining popularity rapidly and has a growing ecosystem as well.

Development speed and ease of use: Consider the development speed and ease of use of each framework. Vue is known for its simplicity and ease of use, making it a good choice for smaller projects or teams with limited experience. React is known for its flexibility and fast development cycles, while Angular has a steeper learning curve but offers a highly structured and scalable approach to building large applications.

Long-term maintainability and stability: Consider the long-term maintainability and stability of the framework. Choose a framework that has a stable release history, active community support, and regular updates to ensure that it will be maintained and supported in the long run.

Ultimately, the best choice between Vue, React, and Angular for your e-commerce project will depend on the specific requirements, expertise of your team, and your project's long-term goals. It may be beneficial to involve your development team in the decision-making process and conduct a thorough evaluation of each framework based on your project's needs.

10. Why can’t you use .map() to display a large list in either React or React Native?

In React and React Native, rendering a large list of items efficiently is an important performance consideration. The conventional approach of using .map() to render a large list of items may not be optimal because it can cause performance issues, such as slow rendering and decreased user experience due to the overhead of creating and managing a large number of React elements.

The main difference between rendering a large list in React and React Native is the underlying rendering engine. React renders to the web using the browser's DOM, while React Native renders to native UI components on mobile devices. This difference in rendering engines leads to differences in how large lists are efficiently rendered in both React and React Native.

In React, when rendering a large list, you would typically use techniques such as virtualization or windowing, where only a subset of the list is rendered at any given time, and the DOM elements are recycled as the user scrolls. Libraries such as react-virtualized or react-window can be used to implement virtualization in React.

In React Native, virtualization is built into the core rendering engine, so you don't need to use additional libraries. React Native provides components like FlatList or VirtualizedList that automatically handle rendering only the visible items in the list and recycle the underlying native UI components as the user scrolls. This allows for the efficient rendering of large lists in React Native without the need for additional optimizations.

In summary, while you can use .map() to render a list of items in React and React Native, it may not be the most efficient approach for large lists. Instead, using techniques such as virtualization or utilizing built-in components like FlatList or VirtualizedList in React Native can greatly improve performance and provide a smooth user experience when rendering large lists.

11. How would you design a secure flow that allows users to preview a product before downloading the application and authenticating?

When a user clicks on the link, they will be redirected to different pages of your React web app depending on the platform.

You should use Firebase Dynamic links to implement this feature. The Firebase link should usually contain:

PC Preview Page link (in your React app) with product ID as the parameter

Mobile Preview Page links (in your React app) with product ID as the parameter

Package name (of Android app) and app identifier (of iOS app) to redirect the user to Play Market/App Store later

If the user clicks on the link using a PC:

The PC Preview Page page should display a product preview (you can get data from the Node.js Express server by product ID) and a QR code with the same link. Why the same? When a user scans the QR code:

If the application is already installed, your React Native app will intercept (you need to set it up) and process the link with a product ID from the QR code. Thus, the user will receive the product at the current step.

If the app is not installed, the user will be redirected to the Mobile Preview page.

If the user clicks on the link using a phone (or was redirected there after scanning the QR code):

Mobile Preview Page should display a product preview (you can get data from Node.js Express server by product ID as well) and a button like: «Get the ‘product’ in the ‘app name’».

When the user clicks on the button, the product ID should be copied to the clipboard (I'll explain why later).

After clicking, the user should be redirected to the Play Market/App Store depending on the OS. To implement this, you should use the same Firebase Dynamic Link again.

BUT: Remove the links redirecting to the Mobile Preview Page, as this link already contains the parameters redirecting the user to the Play Market/App Store (using the package name and app identifier). This action will provide the correct process of Deferred Deep Linking.

NOTICE: If the application is already installed, it will intercept the link and the user will receive the product at the current step.

The user installs and opens the app:

The behavior for getting a product ID for Android and iOS will be slightly different:

Android: After first launch, given that the application was installed after clicking on the Firebase Dynamic Link, the Deferred Deep Linking mechanism will work. We will get the effect as if the application was previously installed and running because it intercepted the link. This can help you get the product ID from the parameters of the intercepted link.

iOS: After first launch, we should get the product ID from the clipboard because the Deferred Deep Linking mechanism does not work for iOS now. (Perhaps this requires very fine configuration for the native side, but as far as I know, it doesn't work for iOS now.) However, the application will ask the user if they grant access to the data in the clipboard, which may cause some inconvenience.

Once the user is logged in, you can start getting the product process using the product ID. Of course, logging in and getting the product by the ID endpoints must be available on your Node.js Express server.

12. Is it necessary to use Typescript in React/React Native for every project?

The decision to use TypeScript in React or React Native projects depends on several factors, including the size and complexity of the project, the team's familiarity with TypeScript, and the specific requirements and constraints of the project. While TypeScript can be beneficial in many projects, it may not always be necessary or make sense in every situation.

TypeScript is a statically typed superset of JavaScript that adds optional static typing to the language. It provides features such as type annotations, interfaces, and type inference that can help catch potential type-related errors during development, improving code quality and reducing bugs.

In larger and more complex projects, TypeScript can offer several advantages. For instance, TypeScript can provide better code navigation and autocompletion in IDEs, as well as enhanced type-checking during compilation, which can help identify type-related issues early in the development process. This can lead to improved maintainability, scalability, and overall code quality in larger projects, where managing dependencies, collaborating with multiple team members, and maintaining a large codebase can be more challenging.

Additionally, TypeScript can help improve team productivity, as it can provide better documentation and self-documenting code through its type annotations and interfaces. This can help team members better understand the structure and usage of the code components.

However, in smaller projects with simple requirements, TypeScript may not always be necessary. The decision to use TypeScript should be carefully considered based on the specific needs and goals of the project, as well as the experience and familiarity of the development team with TypeScript. If the team is not familiar with TypeScript or if the project has limited complexity and size, it may be more efficient to stick with plain JavaScript.

Ultimately, the decision to use TypeScript in React or React Native projects should be based on a thorough assessment of the project's requirements, the team's familiarity with TypeScript, and other relevant factors rather than being applied as a blanket rule for every project.

13. React uses SPA — that is, everything works within a single page. How does it work in React Native?

In React Native, the concept of Single Page Application (SPA) is slightly different compared to traditional web applications built with React.

In React Native, each screen or view in the mobile app is represented by a separate component, which is similar to how different pages are represented by separate components in React. When you navigate from one screen to another, you are actually navigating between different components or views rather than reloading an entire page.

React Native uses a navigation library, such as React Navigation, to handle the navigation between different screens in the mobile app. React Navigation provides different types of navigators, such as Stack Navigator, Tab Navigator, Drawer Navigator, etc. that you can use to define the navigation structure of your app.

When you navigate from one screen to another using React Navigation, the content within the current screen's component is unmounted and the component of the next screen is mounted, similar to how React works. This means that the state and props of the current screen's component will be preserved, while the component of the next screen will be rendered with its own state and props.

In other words, in React Native, it's more similar to the content changing within a single screen rather than creating a new Activity or replacing an Activity as in native Android development. React Native allows you to define the UI components and navigation structure in a declarative way using JavaScript and React, and it then uses the underlying native components and views to render the user interface on the respective platform (such as iOS or Android). This is different from creating a new Activity in native Android development, as the navigation and screen management are done within the same JavaScript runtime and do not involve creating separate activities with their own life cycles.

This means that all application code is executed inside one JavaScript runtime, while the native side is only responsible for displaying native components on the screen in which the entire application is running. Transition between screens occurs without creating separate activities or fragments on Android or UIViewController on iOS. Instead, the content of the current screen is replaced with the new content defined in the new screen component, and this happens natively, providing high performance and smooth transitions between screens.

In React Native, on the native side, the entire application works within a single screen, which is created at the start of the application and remains active during the entire lifecycle of the application. This screen contains a frame inside which React Native components are rendered, and all of the app's user interface, including screen rendering and navigation controls, happens inside this screen.

14. What features do React and React Native provide for handling user interaction? Are there differences in their use?

React and React Native are popular JavaScript frameworks for building user interfaces. While React is used for building web applications, React Native is used for building mobile applications for iOS, Android, and other platforms. Both React and React Native provide features and tools for handling gestures, events, and user interactions, but there are some differences in their implementation due to the different natures of web and mobile environments.

React for web development:

Event handling: React provides a synthetic event system that normalizes event handling across different browsers. You can attach event handlers to DOM elements using React's event handling syntax, such as onClick, onKeyPress, etc. React also supports custom events.

Gestures: React does not have built-in support for handling gestures, such as swipe, pinch, etc. However, you can use third-party libraries like React Gesture or React-DND to handle gestures in React applications.

React Native:

Touch handling: React Native provides a built-in touch event system for handling touch events, such as onPress, onLongPress, onSwipe, etc. These events are similar to the ones used in React for web but are designed specifically for touch-based interactions on mobile devices.

Gestures: React Native has built-in support for handling gestures, such as swipe, pinch, etc., using the PanResponder API. You can use PanResponder to define gesture recognition logic and attach it to components to handle gestures.

Animated library: React Native also provides the Animated library, which allows you to create complex animations and interactions using declarative syntax. It provides support for animations, transitions, and gestures, making it easy to create interactive user interfaces.

Native module integration: React Native allows you to interact with native modules written in Java (for Android) or Objective-C/Swift (for iOS), which gives you the flexibility to handle complex interactions that require direct access to the native APIs of the mobile platform.

In summary, while React and React Native share some similarities in handling events and user interactions, React Native provides additional features specific to mobile environments, such as touch handling, built-in gesture recognition, and integration with native modules, making it well-suited for building mobile applications with rich and interactive user interfaces.

15. What are the performance and optimization differences between React and React Native for mobile development?

Performance differences:

Rendering: React renders components to the web using the browser's DOM, while React Native renders components to a native view hierarchy, which allows it to achieve better performance compared to React for mobile applications.

UI updates: React uses a virtual DOM (a lightweight in-memory representation of the DOM) to optimize UI updates, while React Native uses a bridge to communicate between JavaScript and the native components, which introduces some performance overhead.

Animations: React Native has built-in support for animations using the Animated API, which is optimized for mobile devices and can deliver smooth animations. In React, animations are typically done using CSS animations or third-party libraries, which may not perform as well on mobile devices.

Optimization differences:

Styling: In React, styling is typically done using CSS, while in React Native, styling is done using a combination of JavaScript and a subset of CSS called Flexbox. This means that styling in React Native is more tightly integrated with the component code, which can help with code organization and optimization.

Bundling: React Native uses a different bundling mechanism compared to React, as it needs to package JavaScript code along with native components for different platforms. This can affect the size of the final app binary and may require additional optimization techniques, such as code splitting or dynamic imports, to reduce the bundle size.

Platform-specific code: React Native allows for writing platform-specific code, which can be used to optimize performance for specific platforms. For example, by using native modules in React Native, developers can write performance-critical code in native languages like Java or Objective-C/Swift, which can lead to performance optimizations.

In summary, React and React Native have some performance and optimization differences due to the differences in their rendering, UI updates, animations, styling, bundling, and platform-specific code. React Native is optimized for mobile development and provides native-like performance, while React is optimized for web development. However, the choice between React and React Native depends on the specific requirements of the project, such as the target platform, performance expectations, and the development team's expertise.

What is ES6 feature

small coding on how let,var,const works javascript set ,

closure (closure with different let scenarios)

difference between function and class component

use of sort destructure to insert element in array

how to remove duplicate in array

how browser router is used path with exact and how it works

custom hooks webpack

how to deploy react

how useEffect works during that lifecycle

lifecycle arrow function

knowledge on latest react version

Js basics that handles array and objects and have some recap on filter,find,map js basics

es6 basics

React basics like lifecycle and hook concepts

Q1. How do you avoid re-rendering of a component? With useEffect second parameter, should ComponentUpdate

Q2. What are refs? How will you use it for getting input value? Explain with code.

Q3. What are the ways to handle Errors in React?

Q4. What is Lazy Loading, Suspense. How do they work?

Q5. How componentWillUnmount works?

Q6. Explain useState, useEffect Hooks.

How do you avoid re-rendering of a component? With useEffect second parameter, should ComponentUpdate

What are the ways to handle Errors in React?

How to optimize React App? Important Question

What is Stateless and Stateful component?

How to redirect to login page through React Router if the user has not logged in and trying to go to another page through URL.

REACT(Virtual DOM, class vs functions, lifecycle methods and hooks, work flow on a button click, ROUTER, Context, HOC),

REDUX(create store, apply middleware, combine reducers, pure functions, redux saga/thunk, also folder structure for redux ),

REACT-REDUX(connect(), mapstatetoprops, mapdispatchtoprops, useselector and usedispatch).

\*\*\*\*\*\*\*\*\*\*\*\*

Photon I Q & A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

useMemo, usecallback

Different lifecycle hooks

useEffect - different options available

How to pass from grand parent to grand child without props and redux) - using React Context API

Synchreon

\*\*\*\*\*\*\*\*\*\*\*

1. What is Virtual Dom
2. How rendering is happening in Virtual Dom, how it detects and rendering changed section alone
3. Controlled and uncontrolled components
4. useEffect - different scenarios
5. How to start and stop timer? Which logic to be used?
6. Why we use Context API and what is the difference between ContextAPI and Redux?

1. Context API prompts a re-render on each update of the state and **re-renders all components** regardless. Redux however, only **re-renders the updated components**. This can be monitored on the console as there's a log in each component.

2. You can’t use it to manage the state of your application in a single store and it is not as flexible as Redux.

# **Comparing Redux & Context API**

| **Context API** | **Redux** |
| --- | --- |
| Built-in tool that ships with React | Additional installation Required, driving up the final bundle size |
| Requires minimal Setup | Requires extensive setup to integrate it with a React Application |
| Specifically designed for static data, that is not often refreshed or updated | Works like a charm with both static and dynamic data |
| Adding new contexts requires creation from scratch | Easily extendible due to the ease of adding new data/actions after the initial setup |
| Debugging can be hard in highly nested React Component Structure even with Dev Tool | Incredibly powerful Redux Dev Tools to ease debugging |
| UI logic and State Management Logic are in the same component | Better code organization with separate UI logic and State Management Logic |

1. React application is lagging. How to improve the performance?

<https://www.tiny.cloud/blog/react-performance-optimization/>

<https://www.simform.com/blog/react-performance/>

1. What is Profiler in React?

<https://react.dev/reference/react/Profiler>

Also, another Chrome performance testing using Dev tools Profiler

1. React Unit testing

* Jest. Jest is the most popular testing framework created and maintained by Facebook. ...
* Mocha. Mocha is another popular testing framework for Javascript developers. ...
* Chai. ...
* Jasmine. ...
* Karma. …

<https://www.digitalocean.com/community/tutorials/how-to-test-a-react-app-with-jest-and-react-testing-library>

1. React Fiber
2. What is the use of forwardRef:

<https://react.dev/reference/react/forwardRef>

1. Lazy loading in React - https://react.dev/reference/react/lazy