React is an indispensable part of the present-day JavaScript frontend web development scenario. For those looking to work with JS on the frontend, having adequacy in React is a must.

React stood among the [leading JavaScript frameworks in 2018](https://hackr.io/blog/10-best-javascript-frameworks-2018) and probably will continue to do so in 2019. Though regarded as a front-end framework, React is actually a front-end library.

In fact, the whole idea of React as a web development framework is so good that it finished charting as one of the leading web development frameworks in 2018.

Thus, React is one of the most important JavaScript frameworks that any JS aspirant must know about. Here are 20 React Interview questions that you should have answers to while facing your next frontend web developer interview:

**Question**: **What is React? What are some of its standouts?**  
**Answer**: React is a frontend JavaScript library. It was developed by Facebook in 2011. It enhances application performance while allowing for working on both client-side and server-side.

Writing UI test cases is simple with React, which is also easy to integrate with Angular, Meteor, and other popular JS frameworks. Here are some of the major standouts of React:

* Excellent for developing complex and interactive web and mobile UI
* Follows the component-based approach and helps in building reusable UI components
* Features one of the largest community support
* Makes use of the virtual DOM instead of the real DOM
* Relies on server-side rendering
* Supports unidirectional data flow or data binding

**Question**: **React has something called a state. What is it and how it is used?**  
**Answer**: States are the source of data for React components. In other words, they are objects responsible for determining components behavior and rendering. As such, they must be kept as simple as possible.

Accessible by means of this.state(), state is mutable and create dynamic and interactive components. Use of a state can be visualized by the following code snippet:

class App extends React.Component {

constructor() {

super();

this.state={

foo: 'bar'

}

}

**Question**: **Why do we use render() in React?**  
**Answer**: In React, each component must have a render() function. It returns a single React element, which is, in fact, the representation of the native DOM component.

When there is a need for more than one HTML element to be rendered, we group them in one enclosing tag, which can be <div>, <form>, <group>, or some other tag. There is a need for the render() function to return the same result each time it is invoked i.e. it needs to be kept pure.

**Question**: **Draw a diagram showing how data flows through Redux.**  
**Answer**:

**Question**: **Explain major differences between the ES5 and ES6 syntax with relevant examples.**  
**Answer**: The syntax has witnessed a great change from ES5 to ES6. Important differences among the two releases of ECMAScript are:

* **Require vs. Import – The require used in ES5 is now replaced with import.**var React = require(‘react’); //is now replaced with  
  import React from ‘react’; //in ES6
* **Export vs. Exports –** Instead of exports, now export is used.export default Component; // replaces  
  module.exports = Component; // in ES6
* **Component and Function –** The use of component and function has also changed from ES5 to ES6.  
  In ES5:

var MyComponent = React.createClass({

render: function() {

return

<h3>Hello World!</h3>

;

}

});

In ES6:

class MyComponent extends React.Component {

render() {

return

<h3>Hello World!</h3>

;

}

}

* **Props – Rules for using props has also changed from ES5 to ES6**

In ES5:

var App = React.createClass({

propTypes: { name: React.PropTypes.string },

render: function() {

return

<h3>Hello, {this.props.name}!</h3>

;

}

});

In ES6:

class App extends React.Component {

render() {

return

<h3>Hello, {this.props.name}!</h3>

;

}

}

* **State – Using state has also been tweaked for ES6.**

In ES5:

var App = React.createClass({

getInitialState: function() {

return { name: 'world' };

},

render: function() {

return

<h3>Hello, {this.state.name}!</h3>

;

}

});

In ES6:

class App extends React.Component {

constructor() {

super();

this.state = { name: 'world' };

}

render() {

return

<h3>Hello, {this.state.name}!</h3>

;

}

}

**Question**: **Explain the Virtual DOM and its working.**  
**Answer**: A virtual DOM is a lightweight JS object. It is simply a copy of the real DOM. A virtual DOM is a node tree that lists various elements, their attributes, and content as objects and their properties.

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

The Virtual DOM operates in three simple steps:

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

**Question**: **How does the Real DOM differ from the Virtual DOM?**  
**Answer**:

* DOM Manipulation – Real DOM supports a very expensive DOM manipulation. Virtual DOM, on the contrary, has an inexpensive DOM manipulation.
* Element Update – Real DOM creates a new DOM when an element updates. Virtual DOM doesn’t do so in such a case. Instead, it updates the JSX.
* Memory Wastage – Real DOM causes a lot of memory wastage while there is no memory wastage for Virtual DOM.
* Update Speed – Real DOM updates slowly. On the other end, the virtual DOM updates faster.
* Updating HTML – Real DOM can directly update HTML, while virtual DOM can’t update HTML directly.

**Question: Explain various lifecycle methods of React components.**  
**Answer**:

* componentDidMount() – Executes on the client side after the first render
* componentDidUpdate() – Called immediately after rendering takes place in the DOM
* componentWillMount() – Executes immediately before rendering starts on both the client-side and the server-side
* componentWillReceiveProps() – Invokes when props are received from the parent class and before another render is called
* componentWillUnmount() – Used to clear up the memory space. Called right after the component is unmounted from the DOM
* componentWillUpdate() – Called immediately before rendering takes place in the DOM
* shouldComponentUpdate() – Returns either true or false. Though false by default, needs to be set to return true if the component needs to be updated

**Question**: **Explain JSX with a code example. Why can’t browsers read it?**  
**Answer**: JSX is a contraction of the JavaScript XML. It uses the expressiveness of JavaScript for making the HTML code easily understandable. JSX files make applications robust while boosting their performance. A code example of JSX is:

render(){

return(

<div>

<h1> React learning made better by Hackr.io!!</h1>

</div>

);

}

JSX isn’t a regular JS object. The inability of browsers in reading JSX is due to the fact that browsers can only read regular JS objects.

In order to enable a web browser for reading the JSX file, it needs to be transformed into a regular JavaScript object. For this, JSX transformers, like [Babel](https://babeljs.io/), are used.

**Question**: **Give a code example to demonstrate embedding two or more components into one.**  
**Answer**:

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')

);

**Question**: **Give a code example to modularize code in React.**  
**Answer**: In order to modularize code in React, export and import properties are used. They assist in writing the components distinctly in different files:

export default class ChildComponent extends React.Component {

render() {

return(

<div>

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

</div>

);

}

}

import ChildComponent from './childcomponent.js';

class ParentComponent extends React.Component {

render() {

return(

<div>

<ChildComponent />

</div>

);

}

}

**Question**: **How does the React Router differ from conventional routing?**  
**Answer**:

* Changes in the URL – A HTTP request is sent to a server for receiving a corresponding HTML page in conventional routing. React routing necessitates only for a change in the History attribute.
* Navigation – In conventional routing, the user actually navigates across different web pages for each individual view. In React routing, however, the users feel like they are navigating across distinct webpages while in actuality they aren’t.
* Pages – Whereas in React routing only a single HTML page is involved, each view corresponds to a new file in conventional routing.

**Question**: **How does the state differ from props in React?**  
**Answer**:

* Changes inside child components are possible with props but not with state
* Changes inside component aren’t possible with props but with state
* Props allow for parent component to change value, state doesn’t

**Question**: **How will you distinguish Redux from Flux?**  
**Answer**:

* **Components – React components subscribe to the store in flux whereas in redux, container components utilize connect**
* Dispatcher – There is no dispatcher in redux. On the other hand, flux has a singleton dispatcher
* Number of Stores – While flux has several stores, there is only a single store for redux
* State – It is mutable for flux but immutable for redux
* Store – In flux, the store contains state as well as change logic. Contrary to this, the store in redux is separate from the change logic
* Store Type – All stores in flux are disconnected and flat. This is not the case with redux, where there is a single store with hierarchical reducers

**Question**: **How would you create a form in React?**  
**Answer:** React forms are identical to HTML forms. However, the state is contained in the state property of the component in React and is updateable only via the setState() method.

Therefore, the elements in a React form can’t directly update their state. Their submission is handled by a JS function, which has full access to the data entered into the form by a user.

Following code demonstrates creating a form in React:

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>

);

}

**Question: What are the advantages of using Redux?**  
**Answer:**

* **Better Code Organization** – Redux is precise in terms of how the code needs to be organized. This results in a consistent code workable for any development team
* **Developer Tools** – Allow developers to track each and everything, ranging from actions to state changes, happening in the application in real time
* **Easy Testing** – Redux code is mainly composed of functions that are isolated, pure, and small. Hence, testing is much easy and simple
* **Large-scale Community** – Redux is backed by a mammoth community. It contributes to an ever-growing and refined library and ready-to-use applications
* **Maintainability** – Thanks to a predictable outcome and strict structure, the code is easier to maintain.
* **Output Predictability** – There is no confusion about syncing the current state with actions as well as other parts of the application as there is only a [single source of truth](https://stackoverflow.com/questions/47182888/what-does-the-single-source-of-truth-mean), which is the store
* **Server–side Rendering** – There is a need of only passing the store created on the server-side to the client-side. In addition to this being useful for initial render, it also offers a better user experience because it optimizes the application performance

**Question**: **What do you understand by Props in React?**  
**Answer:** Props is a contraction for Properties in React. These read-only components need to be kept immutable i.e. pure. Throughout the application, props are passed down from the parent components to the child components.

In order to maintain the unidirectional data flow, a child component is restricted from sending a prop back to its parent component. This also helps in rendering the dynamically generated data.

**Question**: **Where would you put AJAX calls in your React code?**  
**Answer:** It is possible to use any AJAX library with React, such as Axios, jQuery AJAX, as well as the inbuilt browser window.fetch.

Data with AJAX calls need to be added to the componentDidMount() lifecycle method. By doing this, it is possible to use the setState() method for updating component as soon as the data is retrieved.

**Question**: **Write a sample code to update the state of a component in React?**  
**Answer**: State of a component in React is updated with this.setState() as demonstrated in the following code example:

class MyComponent extends React.Component {

constructor() {

super();

this.state = {

name: 'Akhil',

id: '101'

}

}

render()

{

setTimeout(()=>{this.setState({name:'Bhadwal', 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')

);

**Question**: **You must’ve heard that “In React, everything is a component.” What do you understand from the statement?**  
**Answer**: The building blocks of a React application’s UI are called components. Any app UI created using React is divisible into a number of small independent and reusable pieces, known as components.  
React renders each of the components independent of each other. Hence, there is no effect of rendering a component on the rest of the app UI.  
**All Done!**

[](https://hackr.io/tutorials/learn-react?ref=blog-post-banner)

That sums up the list of the important 20 React Interview Questions. Other than robust technical know-how, you need to have a clear and confident presentation. Ensure to keep up your demeanor and mannerism while giving out answers to the interviewer.

If you still want to enhance your React understanding and skill, here are some opportune [React tutorials](https://hackr.io/tutorials/learn-react?ref=blog) recommended by the community.

<https://hackr.io/blog/react-interview-questions>