

# 5 Essential React.js Interview Questions and Answers

Published Jul 14, 2016 Last updated Nov 24, 2017



In the 2017 developer survey (<https://insights.stackoverflow.com/survey/2017>), Stack Overflow noted that React is still among the the most popular JavaScript libraries to date. React has exploded in popularity because its simple and declarative API produces highly-performant applications — and that momentum only continues to grow.

If you're looking to build a robust web application, chances are that React may be a good fit for you. Once you're ready to hire a React developer, here are essential interview questions to ask and some advanced concepts to know.

## A Word on Technical Interviews

Dec 17, 2015


**37 Java Interview Questions and Answers (/blog/java-interview-sample-questions-answers-du107xs23)**




Feb 8, 2016

**29 AngularJS Interview Questions and Answers (/blog/angularjs-interview-questions-sample-answers-du1081n7p)**

Nov 10, 2017 **FEATURED POST**

**40+ Upwork Alternatives: Best Freelance Websites for 2018 (/blog/40-upwork-alternatives-316o841kxm)**

Before we dive right into the questions, it needs to be said that technical interviews are notorious for gotcha-style questions and irrelevant whiteboarding exercises. This article avoids that interview style entirely — rather, I outlined five general (yet vital) concepts which I believe  any seasoned React developer should know.

 Over the years, I have been in countless interviews as both the applicant  and the conductor. My experience has taught me that the best  candidates for an engineering role are those who can **articulate intelligent opinions** and **defend them using examples from their own experience**. Pair-programming relevant examples as a follow-up to discussion would be my preferred interview format, but we will stick to the Q&A portion for this article.

## Question #1: What is React? How is it different from other JS frameworks?

Although this sounds like a relatively simple question, it's really asking the candidate to state an informed opinion about React, as well as 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 (<https://www.codementor.io/javascript/tutorial/should-you-build-your-web-application-with-javascript-mvc-frameworks>).

## 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 than a lot of 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 an incredible amount of flexibility in choosing the architecture they deem "best" — though it also places the responsibility of choosing (or building) those parts on the developer.*

*I recently migrated an application originally written in AngularJS to React, and one of the things I loved most was...*

By comparing and contrasting React with another library, not only can the candidate demonstrate a deep understanding of React, but also position themselves as a potentially strong candidate.

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* (<https://www.codementor.io/codementorteam/react-vs-angular-2-comparison-beginners-guide-lvz5710ha>) or *React vs Vue* (<https://www.codementor.io/vuejsdevelopers/react-or-vue-which-javascript-ui-library-should-you-be-using-6hri3num4>).
- 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 AngularJS application in React, how much code could you expect to re-use?


## Question #2: What happens during the lifecycle of a React component?

One of the most valuable parts of React is its component lifecycle (<https://facebook.github.io/react/docs/component-specs.html>) — so understanding exactly how components function over time is instrumental in building a maintainable application.


### High-Level Component Lifecycle


At the highest level, React components have lifecycle events that fall into three general categories:


1. Initialization
2. State/Property Updates
3. Destruction





**Hire  
Freelance  
Developers  
In Any Tech  
Stack**

 **Front-end** >  
(/front-end-developers)

 **Back-end** >  
(/back-end-developers)

 **Full-stack** >  
(/full-stack-developers)

 **Mobile** >  
(/mobile-app-developers)

 **DevOps** >  
(/devops-developers)

Every React component defines these events as a mechanism for managing its properties, state, and rendered output. Some of these events only happen once, others happen more frequently; understanding these three general categories should help you clearly visualize when certain logic needs to be applied.



**Blockchain** >

(/blockchain-developers)

**See more categories**  
(/developers)

For example, a component may need to add event listeners to the DOM when it first mounts. However, it should probably remove those event listeners when the component unmounts from the DOM so that irrelevant processing does not occur.

```
class MyComponent extends React.Component {  
  // when the component is added to the DOM...  
  componentDidMount() {  
    window.addEventListener('resize', this.onResizeHandler);  
  }  
  
  // when the component is removed from the DOM...  
  componentWillUnmount() {  
    window.removeEventListener('resize', this.onResizeHandler);  
  }  
  
  onResizeHandler() {  
    console.log('The window has been resized!');  
  }  
}
```

## Low-Level Component Lifecycle



Within these three general buckets exist a number of specific lifecycle hooks — essentially abstract methods — that can be utilized by any React component to more accurately manage updates. Understanding how and when these hooks fire is key to building stable components and will enable you to control the rendering process (improving performance).

Take a look at the diagram above. The events under “Initialization” only happen when a component is first initialized or added to the DOM. Similarly, the events under “Destruction” only happen once (when the component is removed from the DOM). However, the events under “Update” happen every time the properties or state of the component change.

For example, components will automatically re-render themselves any time their properties or state change. However, in some cases a component might not need to update — so preventing the component from re-rendering might improve the performance of our application.



```
class MyComponent extends React.Component {  
  // only re-render if the ID has changed!  
  shouldComponentUpdate(nextProps, nextState) {  
    return nextProps.id === this.props.id;  
  }  
}
```



Find React Developers

(<https://www.codementor.io/reactjs-developers>)

***Find top React developers today.***

*CodementorX will find you the best engineers for your project.*

## Question #3: 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.

```
class MyComponent extends React.Component {  
  render() {  
    let props = this.props;  
  
    return (  
      <div className="my-component">  
        <a href={props.url}>{props.name}</a>  
      </div>  
    )  
  }  
}
```



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.

## Key Talking Points

*Developers do not have to use JSX (and ES2015) to write an application in React.*

This is certainly true. Having said that, many React developers prefer to use JSX as its syntax is far more declarative and reduces overall code complexity. Facebook certainly encourages it in all of their documentation!

*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 (<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>), block scoping via let (<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>), and the new



spread ([https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread\\_operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_operator)) 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.

Candidates that have built a React application using JSX and ES2015 can speak about some specific pros or cons encountered, such as:

*Although it took me some time to get used to the JSX and ES2015 syntax, I discovered how much I really enjoyed using it. Specifically, I'm a big fan of...*

*On the other hand, I could do without the hassle of configuring webpack and Babel. Our team ran into issues with...*

The React docs on JSX Gotchas (<https://facebook.github.io/react/docs/jsx-gotchas.html>) may be good to know/review.

## Question #4: 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 (<http://www.seleniumhq.org/projects/webdriver/>)) 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 from. 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.

For example, the candidate might discuss:

- Redux (<http://redux.js.org/>): perhaps the most popular Flux library today.
- Alt.js (<http://alt.js.org/>): another popular library for managing data in React applications.

## Question #5: 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 output of the component is purely a function of the properties provided to it.

## Bonus Question: Explain this Code

As I mentioned at the beginning of this article, technical interviews (<http://skillcrush.com/2016/03/29/rock-your-next-whiteboard-test/>) may also include time where the developer is asked to look at (and probably write) some code. Take a look at the code below:

```

class MyComponent extends React.Component {
  constructor(props) {
    // set the default internal state
    this.state = {
      clicks: 0
    };
  }

  componentDidMount() {
    this.refs.myComponentDiv.addEventListener('click', this.clickHandler);
  }

  componentWillUnmount() {
    this.refs.myComponentDiv.removeEventListener('click', this.clickHandler);
  }

  clickHandler() {
    this.setState({
      clicks: this.clicks + 1
    });
  }

  render() {
    let children = this.props.children;

    return (
      <div className="my-component" ref="myComponentDiv">
        <h2>My Component ({this.state.clicks} clicks)</h2>
        <h3>{this.props.headerText}</h3>
        {children}
      </div>
    );
  }
}

```

Given the code defined above, can you identify two problems?

1. The constructor does not pass its props to the super class. It should include the following line:

```

constructor(props) {
  super(props);
  // ...
}

```



2. The event listener (when assigned via `addEventListener()`) is not properly scoped because ES2015 doesn't provide autobinding (<https://facebook.github.io/react/docs/reusable-components.html#no-autobinding>). Therefore the developer can re-assign `clickHandler` in the constructor to include the correct binding to this:

```

constructor(props) {
  super(props);
  this.clickHandler = this.clickHandler.bind(this);
  // ...
}

```

Can you explain what the output of this class actually does? How would you use it in an application?

This class creates a `<div />` element and attaches a click listener to it. The content of this component includes a `<h2 />` element that updates every time the user clicks on the parent `<div />`, as well as an `<h3 />` element containing a provided title and whatever child elements were passed to it.

To use this class, the candidate should import it into another class and use it like this:



```
<MyComponent headerText="A list of paragraph tags">
  <p>First child.</p>
  <p>Any other <span>number</span> of children...</p>
</MyComponent>
```



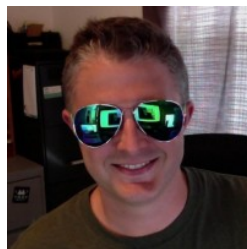
## Conclusion

Interviewing a React developer involves much more than just testing for React knowledge — you should also ask questions about JavaScript (<https://www.codementor.io/nihantanu/21-essential-javascript-tech-interview-practice-questions-answers-du107p62z>) and about other nuances more closely related to the project or job in question.

This article attempted to cover some basic talking points that would demonstrate whether or not a developer has adequate understanding of React and its core concepts. I hope you find it useful — good luck!

---

## Author Bio



**Arthur Kay** (<https://www.codementor.io/arthurakay>)

Arthur Kay has been working with the Web since the late 1990s, when GeoCities and scrolling marquees were all the rage. Arthur graduated from Loyola University Chicago and now works as a Senior Software Engineer with a core focus on JavaScript.



---

Like this article? Share it with your friends!



## **Codementor Team (/codementorteam)**

On-Demand Marketplace for Software Developers

Our team is obsessed with learning about new technologies. We post about development learning, step-by-step guides, technical tutorials, as well as Codementor community announcements to help keep you up-to-date.

Dec 17, 2015 

## 37 Java Interview Questions and Answers (/blog/java-interview-sample-questions-answers-du107xs23)

Here's a comprehensive list of the top technical interview questions to ask Java developers. Get insights on how to answer these questions from Java experts.

**Read more** (/blog/java-interview-sample-questions-answers-du107xs23)



Feb 8, 2016

## 29 AngularJS Interview Questions and Answers (/blog/angularjs-interview-questions-sample-answers-du1081n7p)

Here are the top interview questions that AngularJS developers should know, whether you want to hire an AngularJS developer or just need interview practice.

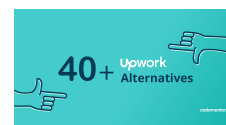
**Read more** (/blog/angularjs-interview-questions-sample-answers-du1081n7p)



Nov 10, 2017 **FEATURED POST**

## 40+ Upwork Alternatives: Best Freelance Websites for 2018 (/blog/40-upwork-alternatives-316o841kmx)

Freelance talent sourcing giants like Upwork aren't the only options to find the remote workers you need. Check out this list of 40 Upwork alternatives.



[Read more \(/blog/40-upwork-alternatives-316o841kmx\)](/blog/40-upwork-alternatives-316o841kmx)



CodementorX has top developers available for hire and freelance jobs. Check out our React developers (/reactjs-developers)!



### Tony Guerrero

I have been in the coding education sector for about 4 years now and I love helping students understand

React   Node.js

JavaScript   Ruby

**HIRE NOW**

[\(/toneloke/profile\)](/toneloke/profile)



### Matt Tanguay-Carel

I do consulting and web development. I've worn many hats but these days I tend to help startups and coach

React   Ruby

Ruby on Rails

**HIRE NOW**

[\(/matstc/profile\)](/matstc/profile)



### Michael Gutierrez

I help startups and larger companies build their modern applications and products. Experience

React   React

Redux   JavaScript

**HIRE NOW**

[\(/iamgutz/profile\)](/iamgutz/profile)

# Are you a top freelance developer? Join CodementorX!

APPLY AS A DEVELOPER (/freelancer/apply)

HIRE A DEVELOPER (https://hire.codementor.io)




18 Comments

Codementor

 Login ▾

 Recommend 1

 Share

Sort by Newest ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name



**kanika** • 5 months ago

great blog

^ | ▾ • Reply • Share ›

**Mitesh Dave** • 6 months ago

Please add Render prop pattern and Fiber Reconciliation

^ | ▾ • Reply • Share ›

**Dardo Ariel Landa** • 6 months ago

Thank you very much !

1 ^ | ▾ • Reply • Share ›

**Karun Kamal** • 9 months ago

Superb article! especially the example in the end. Thank you

1 ^ | ▾ • Reply • Share ›

**Polo Rith** • 9 months ago

Yes, appreciated!

^ | ▾ • Reply • Share ›

**Amánda Koster** • 9 months ago

I thank you!

^ | v • Reply • Share ›

**Shyam** • a year ago

Hi **@Arthur Kay**



Correct me if I am wrong



In example of question 2



```
shouldComponentUpdate(nextProps, nextState) {  
  return nextProps.id === this.props.id;  
}
```



Should comparison statement be : return `nextProps.id !== this.props.id`;

1 ^ | v • Reply • Share ›



**Steve Rogers** ➔ Shyam • 4 months ago

You could also use PureComponent as well if you're using it only for shallow comparison.

^ | v • Reply • Share ›



**Eze** ➔ Shyam • a year ago

You're right! As it is, it will be re-render all the time.

1 ^ | v • Reply • Share ›

**David Corwin** • a year ago

There is also a typo (an extra closing curly) in `<h2>My Component ({this.state.clicks} clicks)</h2>`, so really this code does nothing! :)

1 ^ | v • Reply • Share ›

**Kelson** • 2 years ago

Also, ref strings are deprecated in favor of callback refs:

<https://facebook.github.io/...>

1 ^ | v • Reply • Share ›

**Arthur Kay** ➔ Kelson • 2 years ago

You're right, though back when I originally wrote this article they hadn't yet been deprecated. I really need to do a follow-up post on this subject as there are now a few things in here that aren't the greatest interview questions for React in 2017.

^ | v • Reply • Share ›

**mcarel** ➔ Arthur Kay • 4 months ago

Nice article, but you should fix all the minor issues people in the comment have identified (e.g. the extra `}` in the `h2` tag, not using `'ref'` in favor of `'onClick'`, the condition in `'shouldComponentUpdate'`).

That would make an even nicer article! :)

^ | v • Reply • Share ›

**Mikhail Novikov** • 2 years ago

```
clickHandler() {  
  this.setState({  
    clicks: this.clicks + 1  
  });  
}
```



this.clicks = undefined, as result state.clicks = NaN



Its mistake, must be this.state.clicks + 1

1 ^ | v • Reply • Share ›



**sh4dow83** ➔ Mikhail Novikov • 2 years ago

Indeed, I was just going to post that as well. :)

^ | v • Reply • Share ›



**Vinay Babu** ➔ sh4dow83 • 7 months ago

But, state updates in react are async by default. In that case we can't expect the clicks property on state is not the correct way to use it(Because it might give you inconsistent value). Instead we shall use previous state which is a default input parameter for setState function.

please let me know if i am wrong.

^ | v • Reply • Share ›

**Jeff Bellsey** • 2 years ago

I would suggest that a bigger problem than the unbound click handler is: why is the interviewer installing a click handler using refs, using the window's event system manually? React provides an elegant event handling system (e.g., "onClick") that should always be used when possible. Refs are an escape hatch, and rarely need to be used.

The Mystery of the Unbound This is not a React-specific issue. (In fact, neither is the issue with super.) Although, if you wanted to test for this particular gotcha, I'd probably recommend using the React event system: `<div onclick="{this.unboundHandler}">`, where the unbound handler references an instance property like `this.state`, which will throw an error.

4 ^ | v • Reply • Share ›

**Arthur Kay** ➔ Jeff Bellsey • 2 years ago

Jeff,

You're right... I picked a contrived example to illustrate a different problem (something not necessarily specific to React). Then again, the point of all of these questions is asking the interviewee to state informed opinions and defend them -- and it looks like you've passed!

^ | v • Reply • Share ›



## TOP DEVELOPERS

Ruby on Rails Developer (/ruby-on-rails-developers)

Python Developer (/python-developers)

PHP Developer (/php-developers)

Elixir Developer (/elixir-developers)

Node.js Developer (/nodejs-developers)

Django Developer (/django-developers)

Go Developer (/go-developers)

Fullstack Developer (/full-stack-developers)

JavaScript Developer (/javascript-developers)

React Developer (/reactjs-developers)

AngularJS Developer (/angularjs-developers)

Vue.js Developer (/vue-developers)

HTML/CSS (/html\_css-developers)

Web Developer (/web-developers)

Frontend Developer (/frontend-developers)

Swift Developer (/swift-developers)

iOS Developer (/ios-developers)

Android Developer (/android-developers)

React Native Developer (/reactnative-developers)

Ionic Developer (/ionic-developers)

App Developer (/app-developers)

Mobile Developer (/mobile-developers)

## PRODUCTS

### Codementor

Find a mentor to help you in real time (/)

### CodementorX

Hire world-class freelance developers for your team (<https://hire.codementor.io>)

### Community

Share insights, exchange ideas, and learn from fellow developers (/community)

## COMPANY

Apply as a Developer (/freelancer/apply)



