**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 you to state an informed opinion about React, as well as any competing alternatives. In short, this question is designed to test your knowledge about the JavaScript ecosystem at large while also pressing you for specifics on what makes React unique.

To answer this question, let’s take each part 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 your own personal experiences. The important thing is to speak directly to your own experience — cite some examples and be prepared to offer opinions on whether or not you prefer 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 you demonstrate a deep understanding of React but also position yourself as a strong candidate by highlighting your experience.

Be prepared to be asked some follow-up questions as well, such as:

* *Under what circumstances would you choose React over (AngularJS, etc)?*
* *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

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.

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);

}

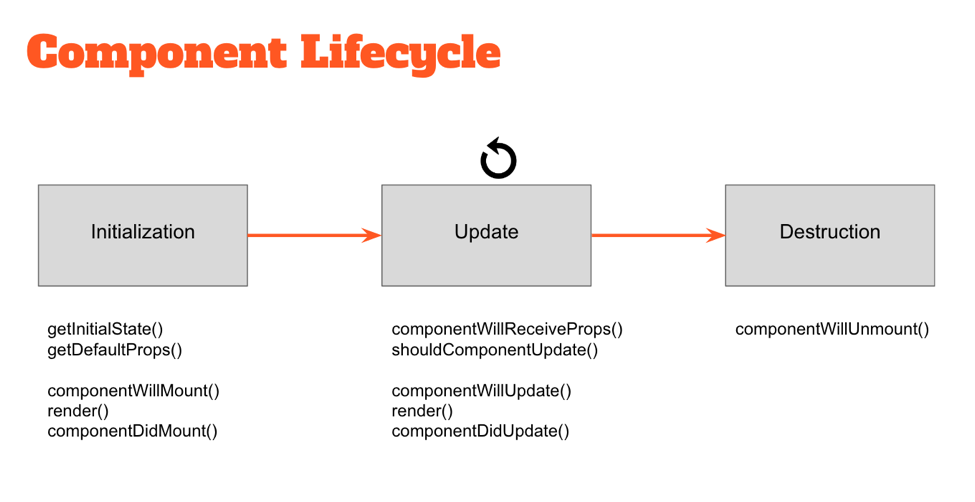
onResizeHandler() {

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;

}

}

**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>

);

}

}

If you are asked this question during an interview, it is ultimately asking you to 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) 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 you’ll need to use a tool like Babel or webpack to convert everything into legacy ES5 code.

If you have built a React application using JSX and ES2015, be sure to speak about some specific pros or cons you encountered:

*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…*

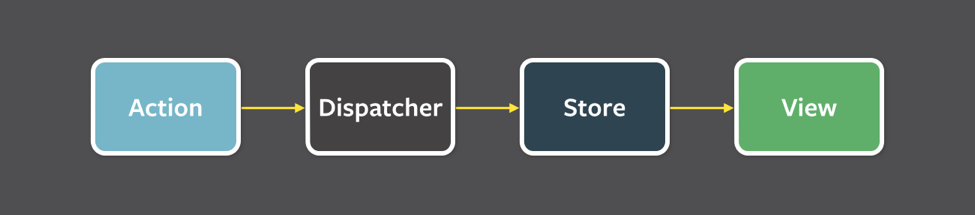
*On the other hand, I could do without the hassle of configuring webpack and Babel. Our team ran into issues with…*

Also be sure to review the React docs on [JSX Gotchas](https://facebook.github.io/react/docs/jsx-gotchas.html) before your interview!

**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, you’ll need to demonstrate 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, it would be great to 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. In a technical interview, you should be prepared to discuss any real-world experience you have using Flux.

For example, you 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”)

);

As you can see, 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, [every technical interview will ask you to look at (and probably write) some code](http://skillcrush.com/2016/03/29/rock-your-next-whiteboard-test/). Take a look at the code below and it will help you [be more prepared for a coding challenge](https://www.codementor.io/learn-programming/9-essential-tips-tackle-coding-challenge):

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?

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

constructor(props) {

super(props);

// ...

}

* 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 we might 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, we would 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>

### Q1: What is ReactJS? What are the advantages of using ReactJS and how it’s different from other JavaScript Frameworks? What are its limitations?

React is an open source JavaScript front end UI library developed by Facebook  for creating interactive, stateful & reusable UI components for web and mobile app. It is used by Facebook, Instagram and many more web apps.

ReactJS is used for handling view layer for web and mobile applications. One of React’s unique major points is that  it perform not only on the client side, but also can be rendered on server side, and they can work together inter-operably.

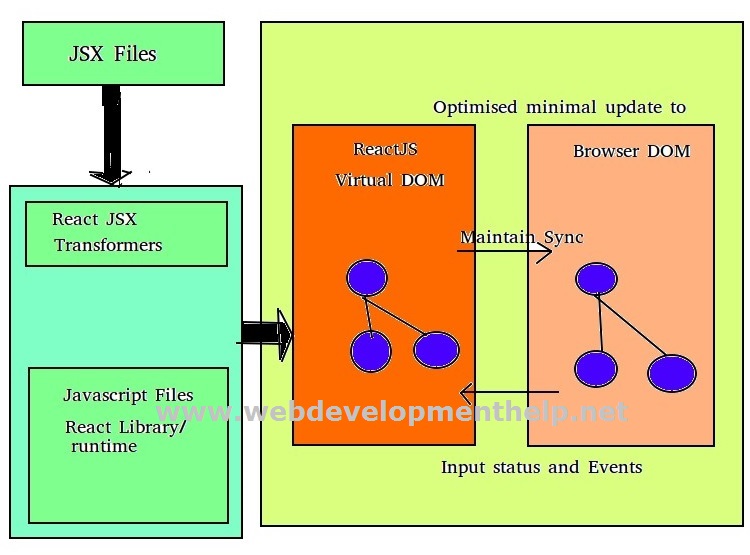
#### Advantages of ReactJS:

React uses virtual DOM which is JavaScript object. This improves application performance as JavaScript virtual DOM is faster than the regular DOM.

* React can be used on client and as well as server side too.
* Using React increases readability and makes maintainability easier. Component, Data patterns improves readability and thus makes it easier for manitaing larger apps.
* React can be used with any other framework (Backbone.js, Angular.js) as it is only a view layer.
* React’s JSX makes it easier to read the code of our component. It’s really very easy to see the layout. How components are interacting, plugged and combined with each other in app.

#### Limitations of ReactJS:

* React is only for view layer of the app so we still need the help of other technologies to get a complete tooling set for development.
* React is using inline templating and JSX. This can seem awkward to some developers.
* The library of react  is too  large.
* Learning curve  for ReactJS may be steep.



### Q2: Please explain step by step approach on how to setup environment for ReactJS?

***Q3: What is ReactJS-JSX? What are the advantages of using JSX? Explain basic code snippet of JSX with the help of a practical example?***

**JSX (JavaScript XML)**, lets us to build DOM nodes with HTML-like syntax. JSX is a preprocessor step which adds XML syntax to JavaScript.

Like XML, JSX tags have a tag name, attributes, and children JSX also has the same. If an attribute/property value is enclosed in quotes(“”), the value is said to be string. Otherwise, wrap the value in braces and the value is the enclosed JavaScript expression. We can represent JSX as <**HelloWorld**/>.

JSX is completely optional and its not mandatory, we don’t need to use it in order to use React, but it has several advantages  and a lot of nice features in JSX.

* JSX is always faster as it performs optimization while compiling code to vanilla JavaScript.
* JSX is also type-safe, means it is strictly typed  and most of the errors can be caught during compilation of the JSX code to JavaScript.
* JSX always makes it easier and faster to write templates if we are familiar with HTML syntax.

Our browsers does not understand JSX code natively, we need to convert it to JavaScript first which can be understand by our browsers. We have aplugin which handles including Babel 5’s in-browser ES6 and JSX transformer called browser.js.

Babel will understand and recognize JSX code in <script type=”text/babel”></script> tags and transform/convert it to normal JavaScript code.

In case of production we will need to pre-compile our JSX code into JS before deploying to production environment so that our app renders faster.

***How to apply validation on Props in ReactJS? Apply validation in previously developed example in above question?***

When the application is running in development mode, React will automatically check  for all props that we set on components to make sure they must right correct and right data type.

For instance, if we say a component has a Message prop which is a string and is required, React will automatically check and warn  if it gets invalid string or number or boolean objects. For performance reasons this check is only done on dev environments  and on production it is disabled so that rendering of objects is done in fast manner .

Warning messages are generated   easily  using a set of predefined options such as:

* React.PropTypes.string
* React.PropTypes.number
* React.PropTypes.func
* React.PropTypes.node
* React.PropTypes.bool

### Q9: How to set up routing in ReactJS. Explain with the help of step by step approach?

Routing is the  key features of web applications (and even other platforms) could not be left out in React. We can develop full featured single page applications (SPA) with React, one of the key feature is routing.

We use library called**React-Router** for routing in reactJS. React goodness is  to keep things as simple as possible and that is why the core library exactly does what React is about, components. Routing, DOM rendering and other logics are abstracted to a different library.

*What happens when you call****setState****?*

The first thing React will do when setState is called is merge the object you passed into setState into the current state of the component. This will kick off a process called reconciliation. The end goal of reconciliation is to, in the most efficient way possible, update the UI based on this new state. To do this, React will construct a new tree of React elements (which you can think of as an object representation of your UI). Once it has this tree, in order to figure out how the UI should change in response to the new state, React will diff this new tree against the previous element tree. By doing this, React will then know the exact changes which occurred, and by knowing exactly what changes occurred, will able to minimize its footprint on the UI by only making updates where absolutely necessary.

*When would you use a****Class Component****over a****Functional Component****?*

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

*What are****refs****in React and why are they important?*