ReactJS

# Why React?

## History

2011 – created by Facebook

2012 – Used on Instagram

2013 – Open sourced

Some people dismissed it because of mixup of markup and logic in single file. But over time, many people embraced it.

2014 – embraced by many large companies

2015 – React Native released for native mobile applications

2016 April – React 15 released

Today – Over 30K components are in production at Facebook

Full time dev staff for regular releases, bug fixes and documentation.

Used by many in Fortune 500 companies

Let’s explore six key reasons for choosing React.

1. Flexibility
2. Developer experience
3. Corporate investment
4. Community support
5. Performance
6. Testability
7. Flexibility : Learn once, write everywhere.

Once you learn ReactJs, you can build UI for huge variety of platforms. It’s more flexible than opinionated frameworks unlike Angular and Ember.

Where can we use React? – Started for Web applications. Now we can develop static websites using Gatsby and Phenomic, truly native mobile apps using React Native, installable desktop using Electron to run on Mac and Windows, also supports Server rendering using NextJS and finally can create Virtual Reality using ReactVR.

React Renderers

React is highly versatile because the renderer is separate from React iself. For web apps use react-dom, for mobile use react-native and for VR environments use react-vr renderer. There are over dozens of renderers including WebGL, CLI, PDF and word documents, canvas.

React-dom provides a simple function called ReactDomServer.renderToString() that renders your component to string of HTML for server side rendering. There are multiple libraries for SSR : NextJS, Gtasby and Phenomic, even for static websites. Since React is a lightweight library you can use it with existing apps too.

Facebook used React to slowly replace its php application. Since React is used by Facebook, its supported by multiple browsers including latest version of IE.

1. Developer experience

You’ll rarely use the docs. There are few APIs to learn.

For example:

import React from ‘react’;

function HelloWorld(props) {  
 return <div>Hello {props.name} </div>  
}

A simple hello world component can be rendered like this. The function receives an object and render its name property encapsulated within a div tag.

You can also declare a React component using a standard JavaScript class.

import React from ‘react’;

class HelloWorld extends React.Component {  
 return() {  
 return <div>Hello {props.name} </div>  
 }  
}

The HTML sitting inside JavaScript is called JSX.

JSX compiles to JS.

For example:

<h1 color=”red”>Heading here</h1>

Is compiled to:

React.createElement(“h1”, {color: “red”}, “Heading here”)

First parameter is the element tag, second parameter is an object with all attributes for the element and then third parameter is the markup that should sit inside the element.

It’s recommended to use JSX because for someone familiar with HTML, it’s easier to read.

Traditional frameworks like Amber, Vue and Angular seek to enhance the power of HTML by inventing their own syntax for example : looping.

“JS” in HTML

Angular : <div \*ngFor=”let user of users”>

Vue : <div v-for=”user in users”>

Ember: {{ #each user in users }}

React went the opposite route. Instead of trying to make HTML powerful, React handles HTML in JavaScript. You don’t need to learn new framework specific keywords, conditionals, looping, you just use JavaScript. JavaScript already has a function for looping called map().

“HTML” in JS

React : { users.map(createUser) }

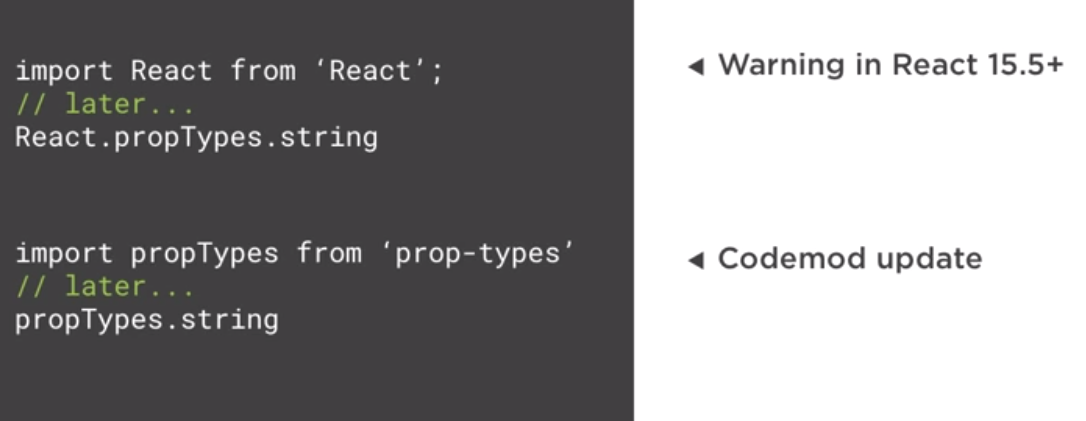
Traditional libraries put fake JavaScript in HTML and React puts fake HTML in JavaScript. React encourages you to get better at JavaScript and in doing so, get better at React.

Use create-react-app to create a new React application and npm start to serve your application. In React, each component is isolated and while developing, each change is applied instantly in the browser. If you make a mistake, you get a detailed error message in the browser and IDE. If you need to debug the code, just put ‘debugger;’.

You can use Visual Studio Code or online code editors such as Code Sandbox, GitPod etc for developing your app.

1. Corporate Investment

Facebook is deeply committed to React. Although React is open sourced, four of the top contributors of React are full time Facebook employees. And Facebook dev team maintains document for each release of React and plans for future. Because of Facebook’s deep existing commitment to React, when breaking changes occur in React, Facebook provides a codemod that automates the change. Codemod is command line tool that you can point at your code to automate the changes. So you can automatically update older React components to the latest specifications.



For example, in React 15.5, propTypes was moved to a different module and all the projects that were built with older versions of React started getting errors after upgrading. Using codemod, developers were able to update their code as per the changes of new version of React. We can rely on React because Facebook must rely on codemods to update the code that they create.

1. Community

Since 2013, React’s popularity has steadily grown to over 75K stars on Github. This makes React one of the most popular library on Github. Today it has over 1K contributors. React components are being downloaded 1.5 million times every single week. On Stackshare, where companies list down the technology stack that they use, over 5K companies have reported using React.

Companies using React : Apple, Adobe, Microsoft, Amazon, Twitter, BBC, Tesla, Netflix, Airbnb, Dropbox, Slack, Reddit, PayPal.

Microsoft open sourced their React library for Office UI.

Google has their own Material UI for React.

Deep community investments has led to wide variety of mature related projects. For routing, use React Router. Do you want to handle complex data flows, consider Redux and Mobx. Automated testing with Jest also from Facebook. Want a library for RESTFul API calls, check out GraphQL. Want server side rendering, try NextJS.

1. Performance

React team recognized that JavaScript is fast but it’s the DOM that makes it slow. They realized that updating the DOM is expensive so they found that updating the DOM in efficient way would help in performance.

Before React most libraries would unintelligently update the DOM to reflect the new state. This often led to redrawing a significant portion of the page even when only a minor change had occurred. In contrast, React monitors the values of each component’s state. When a component’s state changes, React compares the existing DOM state to what the new DOM looks like and then determines the least expensive way to update the DOM. Benefits of this approach: It avoids layout thrashing, which is when a browser has to calculate the position of each element when DOM element changes. And being efficient is important in a world of mobile devices so conserving battery and CPU usage is a concern. This also enables React’s simple programming model, when data changes, React effectively updates the DOM and there is nothing extra that you have to do. The comparison happens in memory so it’s fast.

Today many libraries use similar approach but React’s performance remains competitive. React provides many performance optimizations but they are rarely necessary.

ReactJS with React DOM only weight 35K and there are other React alternatives available such as Preact and Inferno that provide light weight library but they miss out on some features of React to help keep the size down.

1. Testability

Final reason is testability. Testing the front end is hard. React’s design is very friendly to automated testing.

Traditional UI tests vs React

1. It’s a hassle to setup. You have to carefully wire together multiple open source projects to get it to work. With React, testing is already configured for you out of the box.
2. Traditional UI tests require a browser. React uses Node to quickly test the app in memory.
3. Browser based UI tests are slow. In memory tests are faster.
4. Browser based UI tests are more integration oriented whereas React allows us to write Reliable Deterministic unit tests focused on single component in isolation.
5. Browser based tests are time consuming and hard to maintain. React tests can be written quickly in Jest and Enzyme.

With React, the vast majority of your tests contain pure functions. A pure function always returns the same output for the same input. It has no side effects. React’s component style makes it trivial to test your app.

For example:

function HelloWorld(props) {  
 return <div>Hello {props.name} </div>  
}

If we always pass the word “World” to this function then the output is always going to be Hello World. This is reliable, deterministic and has no side effects. It relies upon no global state. This design can scale upto highly complex UIs. Any one of following testing frameworks can work with React : Mocha, Jasmine, Tape, QUnit, Ava, Jest. For React, Jest is most popular and it’s created by Facebook. Every time you hit save during test runs, the automated tests are re-ran.

# Tradeoffs

Let’s consider six key tradeoffs when designing React.

1. Framework vs Library.
2. Concise vs Explicit
3. Template-centric vs JavaScript-centric.
4. Separate template vs Single-file component
5. Standard vs non-standard.
6. Community backed vs Corporate backed
7. Framework vs Library

Competitors like Angular and Ember are framework whereas React is a library. Here’s few advantages to choosing a framework. A framework contains more opinions. This reduces design fatigues and has less setup overhead. Frameworks can help enforce maintain consistency.

However React’s library approach, react is significantly lighter and you can use React in existing applications. You can use your React component to gradually update your application. This is why Facebook is slowly migrating its server side rendered PHP code to React. React allows you to only pull in the features that you need for your project. React can be used as an optional framework.

Angular comes with many features bundled such as Testing, HTTP library, Routing, Internationalization, Animation, Form validation and CLI. In contrast, since React is a feature composed library, you select the features you need and you add them in. Since React is popular, there are so many options for features you need.

1. Concise vs Explicit

React trades conciseness for predictability and explicitness. You spend a little more time wiring things together so that helps things not fall apart.

Example:

Frameworks like Angular and Knockout has two-way binding as a way to avoid typing by automatically keeping form inputs in sync with underline data. This was popular until React came. It was popular because it required less code. React supports one-way binding. It requires a little more code. With React, you declare an explicit change handler and you reference it on your input. This extra work has some benefits, you can transform and validate inputs before updating states and do some performance optimization as desired. This makes it easy to debug and handle when an error occurs. Although React helped repopularize one way binding, Angular and other frameworks have embraced it as well. You can use libraries in React for two way binding but it’s not recommended. Also you don’t need to declare separate change handler for each input. There are simpler patterns for centralizing your change handlers. So in real world, the amount of code you write in React is not substantially large because you have typically single change handler per component.

1. Template-centric vs JavaScript centric

Angular, Vue and Ember seek to make HTML more powerful by inventing their own unique syntax for writing code in HTML. React takes the opposite approach and utilizes the power of JavaScript to handle HTML. This makes React elegant. Let’s consider a simple Conditional which shows a tag when isAdmin is true.

Angular : <h1 \*ngIf=”isAdmin”>Hi Admin</h1>

Vue : <h1 v-if=”isAdmin”>Hi Admin</h1>

Ember: <h1>{{ if isAdmin ‘Hi Admin’}}</h1>

Here, Angular conditional is written inside a string and the ngIf directive must be prefixed with asterisk. Similar for Vue and Ember.

React : { isAdmin && <h1>Hi Admin</h1>}

With React, we can use JavaScript’s logical && operator. The right hand side only runs when left hand side is true. Since the code is plain JavaScript, we get autocomplete support as we type the code and if we type something invalid, we get error messages.

Now let’s consider a loop in each technology.

Angular: <div \*ngFor=”let user of users”>{{ user.name}}</div>

Vue: <div v-for=”user in users”>{{ user.name }} </div>

Ember: {{ #each users as |user| }}  
<div>{{ user.name }}</div>  
{{ /each }}

With Angular, you say ngFor then use a syntax that looks like JavaScript but that’s declared inside a string. Similar for Vue. With Ember, you use Ember’s pound each helper which is lengthy.

React : users.map(user => <div>{ user.name } </div>)

With React, you use JavaScript’s built in map() that takes an arrow function which displays the user’s name. So it’s preferable because the syntax is plain JavaScript.

Finally let’s see how each handles clicking a button.

Angular: <button (click)=”delete()”>Delete</button>

Vue: <button v-on:click=”delete”>Delete</button>

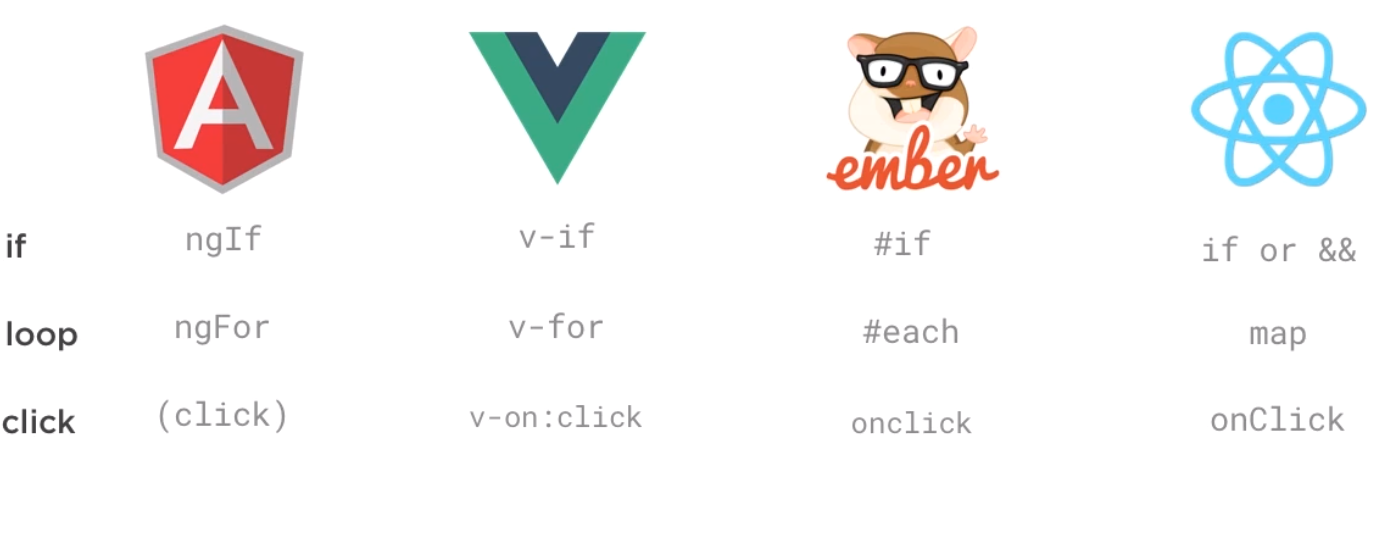
Ember: <button onclick={{ action ‘delete’ }}>Delete</button>

With Angular, you have parenthesis around the events, and unlike traditional event handlers, you also have to put parenthesis after the event handler method. This wouldn’t work if it were real JavaScript. With Vue, you put v-on before the events. With Ember, you specify plain onclick but inside you Ember specific convention to wrap the method in string with action keyword.

React: <button (click)=”delete()”>Delete</button>

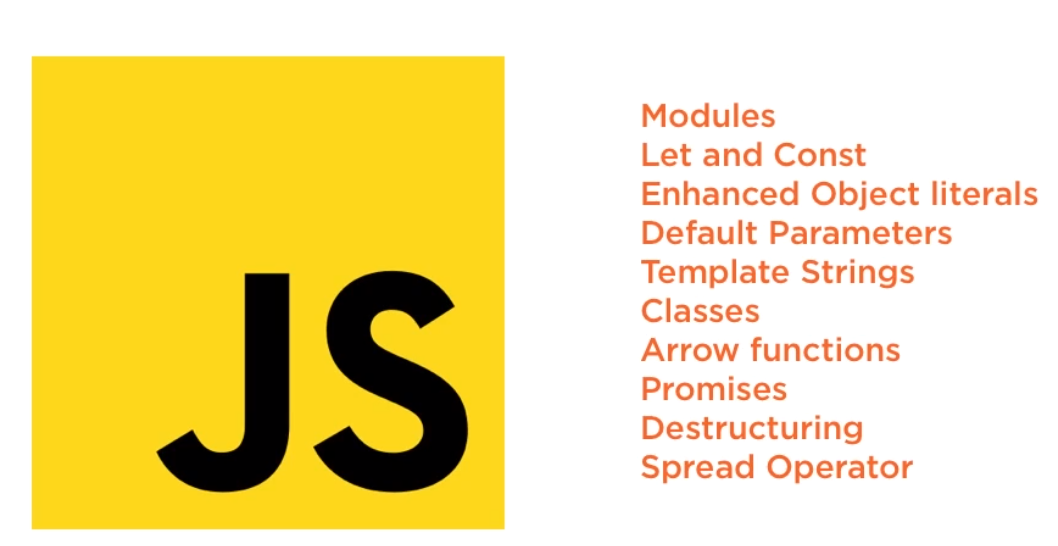
Finally in React, you use native click handler name but its camel case since JSX uses JavaScript casing rules, otherwise the only unique syntax is you specify the function name inside single braces.

If you know JavaScript, then you can easily write conditionals, loops and event handlers in React. This is why React’s API is so small.



Let’s see the benefit of each approach.

Template-centric : It requires little JS knowledge. Template languages provide a streamlined API for performing core functions. You focus on enhancing the template with framework specific syntax. These syntaxes are useful for avoiding confusion with JS binding and this keyword. Template languages is preferable because of principle called Rule of least power. It’s counter-intuitive but less powerful languages can be preferable because they can protect from misuse by only allowing to perform a small set of prescribed operations. For example, Angular’s template syntax only supports JavaScript syntax of comparison. Compared to that, React has little framework specific syntax because you spend more time learning syntax of the framework which JavaScript already supports. React has fewer concepts to learn. Most of the code is JavaScript in React. React’s code is easy to read and it improves JavaScript learnings.



In order to learn React, you need to get better at JavaScript because this means your skills transfer to all JavaScript code even if you use framework or library other than React.

# Why not React?