Jump Start Sketch

by Daniel Schwarz

- **Product Manager:** Simon Mackie
- **English Editor:** Ralph Mason
- **Technical Editor:** Darin Dimitrov
- **Cover Designer:** Alex Walker

Notice of Rights

Notice of Liability

Trademark Notice

Published by SitePoint Pty. Ltd.

48 Cambridge Street Collingwood
VIC Australia 3066
Web: www.sitepoint.com
Email: books@sitepoint.com

ISBN 978-0-9943469-6-4 (print)

ISBN 978-0-9943470-2-2 (ebook)
Printed and bound in the United States of America

About Daniel Schwarz

Danielhttps://mrdaniels.ch is a designer, writer, and now author. He’s also a digital nomad, travelling the world with his beloved wife and earning his bucks writing about various design-related topics, and his

other beloved: Sketch. He founded Airwalk Studioshttp://airwalk-studios.com, which is working on books and magazines for designers and digital nomads. He and his wife both run the company from Airbnb's and local cafés.

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals. Visit http://www.sitepoint.com/ to access our blogs, books, newsletters, articles, and community forums. You'll find a stack of information on JavaScript, PHP, Ruby, mobile development, design, and more.

Table of Contents

Working with Data in React: Properties & State

By Eric Greene

This article is part of a web development series from Microsoft. Thank you for supporting the partners who make SitePoint possible.

Managing data is essential to any application. Orchestrating the flow of data through the user interface (UI) of an application can be challenging. Often, today's web applications have complex UIs such that modifying the data in one area of the UI can directly and indirectly affect other areas of the UI. Two-way data binding via Knockout.js and Angular.js are popular solutions to this problem.

For some applications (especially with a simple data flow), two-way binding can be a sufficient and quick solution. For more complex applications, two-way data binding can prove to be insufficient and a hindrance to effective UI design. React does not solve the larger problem of application data flow (although Flux does), but it does solve the problem of data flow within a single component.

Within the context of a single component, React solves both the problem of data flow, as well as updating the UI to reflect the results of the data flow. The second problem of UI updates is solved using a pattern named Reconciliation which involves innovative ideas such as a Virtual DOM. The next article will examine Reconciliation in detail. This article is focused on the first problem of data flow, and the kinds of data React uses within its components.

Kinds of Component Data

Data within React Components is stored as either properties or state.

Properties are the input values to the component. They are used when rendering the component and initializing state (discussed shortly). After instantiating the component, properties should be considered immutable. Property values can only be set when instantiating the component, then when the component is re-rendered in the DOM, React will compare between the old and new property values to determine

what DOM updates are required.

Here is a demonstration of setting the property values and updating the DOM in consideration of updated property values.

See the Pen [React.js Property Update Demo] by SitePoint ([@SitePoint]3) on [CodePen].

State data can be changed by the component and is usually wired into the component's event handlers. Typically, updating the state triggers React components re-render themselves. Before a component is initialized, its state must be initialized. The initialized values can include constant values, as well as, property values (as mentioned above).

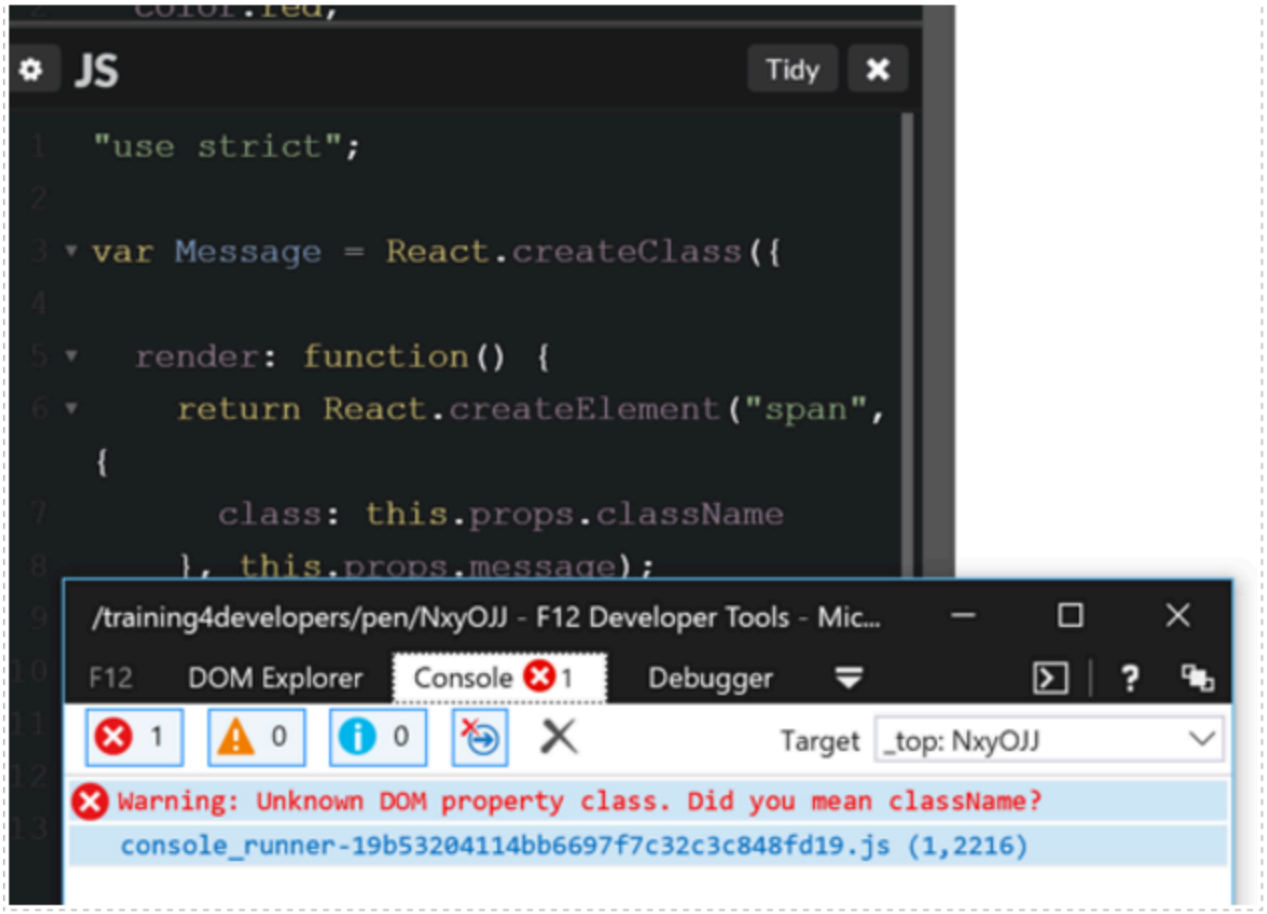
In comparison with frameworks such as Angular.js, properties can be thought of as one-way bound data, and state as two-way bound data. This is not a perfect analogy since Angular.js uses one kind of data object which is used two different ways, and React is using two data objects, each with their specific usages.

Properties

My previous [React article] covered the syntax to specify and access properties. The article explored the use of JavaScript and JSX with static as well as dynamic properties in various code demonstrations. Expanding on the earlier exploration, let's look at some interesting details regarding working with properties.

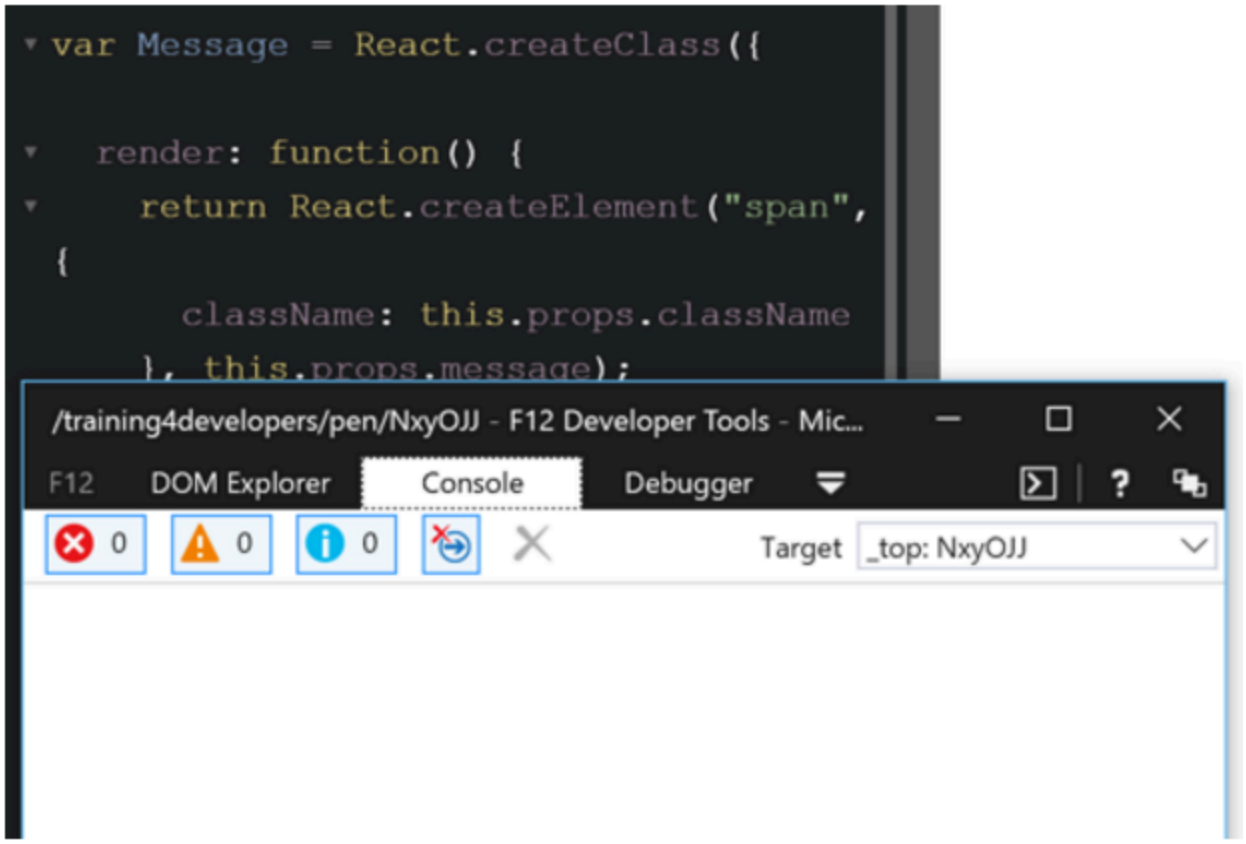
When adding a CSS class name to a component, the property name `className` must be used, rather than `class` must be used. React requires this because ES2015 identifies the word `class` as a reserved keyword and is used for defining objects. To avoid confusion with this keyword, the property name `className` is used. If a property named `class` is used, React will display a helpful console message informing the developer that the property name needs to be changed to `className`.

Observe the incorrect `class` property name, and the helpful warning message displayed in the Microsoft Edge console window.



Incorrect class property name

Changing the `class` property to `className`, results in the warning message not being displayed.



class property changed to ClassName

When the property name is changed from `class` to `className` the warning message does not appear. See below for the complete CodePen demonstration.

See the Pen [React.js Class Property Demo](#) by SitePoint ([@SitePoint]9) on [CodePen](#).

In addition to property names such as `className`, React properties have other interesting aspects. For example, mutating component properties is an anti-pattern. Properties can be set when instantiating a component, but should not be mutated afterwards. This includes changing properties after instantiating the component, as well as after rendering it. Mutating values within a component are considered state, and tracked with the `state` property rather than the `props` property.

In the following code sample, `SomeComponent` is instantiated with `createElement`, and then the property values are manipulated afterwards.

JavaScript:

```
var someComponent = React.createElement(SomeComponent);

someComponent.props.prop1 = "some value";
someComponent.props.prop2 = "some value";
```

JSX:

```
var someComponent = &lt;SomeComponent /&gt;;

someComponent.props.prop1 = "some value";
someComponent.props.prop2 = "some value";
```

Manipulating the `props` of the instantiated component could result in errors that would be hard to trace. Also, changing the properties does not trigger an update to the component, resulting in the component and the properties could be out of sync.

Instead, properties should be set as part of the component instantiation process, as shown below.

JavaScript:

```
var someComponent = React.createElement(SomeComponent, {
  prop1: "some value",
  prop2: "some value"
});
```

JSX:

```
var someComponent = <somecomponent prop1="some value" prop2="some value">
```

The component can then be re-rendered at which point React will perform its Reconciliation process to determine how the new property values affect the DOM. Then, the DOM is updated with the changes.

See the first CodePen demonstration at the top of this article for a demonstration of the DOM updates.

State

State represents data that is changed by a component, usually through interaction with the user. To facilitate this change, event handlers are registered for the appropriate DOM elements. When the events occur, the updated values are retrieved from the DOM, and notify the component of the new state. Before the component can utilize state, the state must be initialized via the `getInitialState` function. Typically, the `getInitialState` function initializes the state using static values, passed in properties, or another data store.

```
var Message = React.createClass({

  getInitialState: function() {
    return { message: this.props.message };
  },
```

Once the state is initialized, the state values can be used like property values when rendering the component. To capture the user interactions which update the state, event handlers are registered. To keep the React components self-contained, event handler function objects can be passed in as properties or defined directly on the component object definition itself.

See the Pen [React.js State Update Demo] by SitePoint ([@SitePoint]3) on [CodePen].

One of the benefits of React is that standard HTML events are used. Included with standard HTML events is the standard HTML Event object. Learning special event libraries, event handlers, or custom event objects is not needed. Because modern browsers are largely compatible, intermediary cross-browser libraries such as jQuery are not needed.

To handle the state changes, the `setState` function is used to set the new value on the appropriate state properties. Calling this function causes the component to re-render itself.

As shown below in the Visual Studio Code editor, the `setState` function is called from the `_messageChange` event handler.

state-updates.jsx blog-post-2/src/www/js

```
1  (function() {
2
3    "use strict";
4
5    var Message = React.createClass({
6
7      getInitialState: function() {
8        return { message: this.props.message };
9      },
10
11     _messageChange: function(e) {
12       this.setState({ message: e.target.value });
13     },
14
```

Visual Studio code editor

Recommended Courses

Wes Bos

A step-by-step training course to get you building real world React.js + Firebase apps and website components in a couple of afternoons. Use coupon code **'SITEPOINT'** at checkout to get **25% off**.

Conclusion

React components provide two mechanisms for working with data: properties and state. Dividing data between immutable properties and mutable state more clearly identifies the role of each kind of data, and the component's relationship to it. In general, properties are preferred because they simplify the flow of data. State is useful for capturing data updates resulting from user interactions and other UI events.

The relationship between properties and state facilitate the flow of data through a component. Properties can be used to initialize state, and state values can be used to set properties when instantiating and rendering a component. New values from user interaction are captured via state, and then used to update the properties.

The larger flow of data within an application is accomplished via a pattern named Flux.

This article is part of the web development series from Microsoft tech evangelists and DevelopIntelligence on practical JavaScript learning, open source projects, and interoperability best practices including Microsoft Edge browser and the new EdgeHTML rendering engine. DevelopIntelligence offers JavaScript Training and React Training Courses through appendTo, their front-end focused blog and course site.

We encourage you to test across browsers and devices including Microsoft Edge — the default browser for Windows 10 — with free tools on dev.microsoftedge.com, including status.microsoftedge.com, a portal for the latest implementation status and future roadmap for interoperable web platform features in

Microsoft Edge and other browsers, including Internet Explorer. Also, visit the Edge blog to stay updated and informed from Microsoft developers and experts.

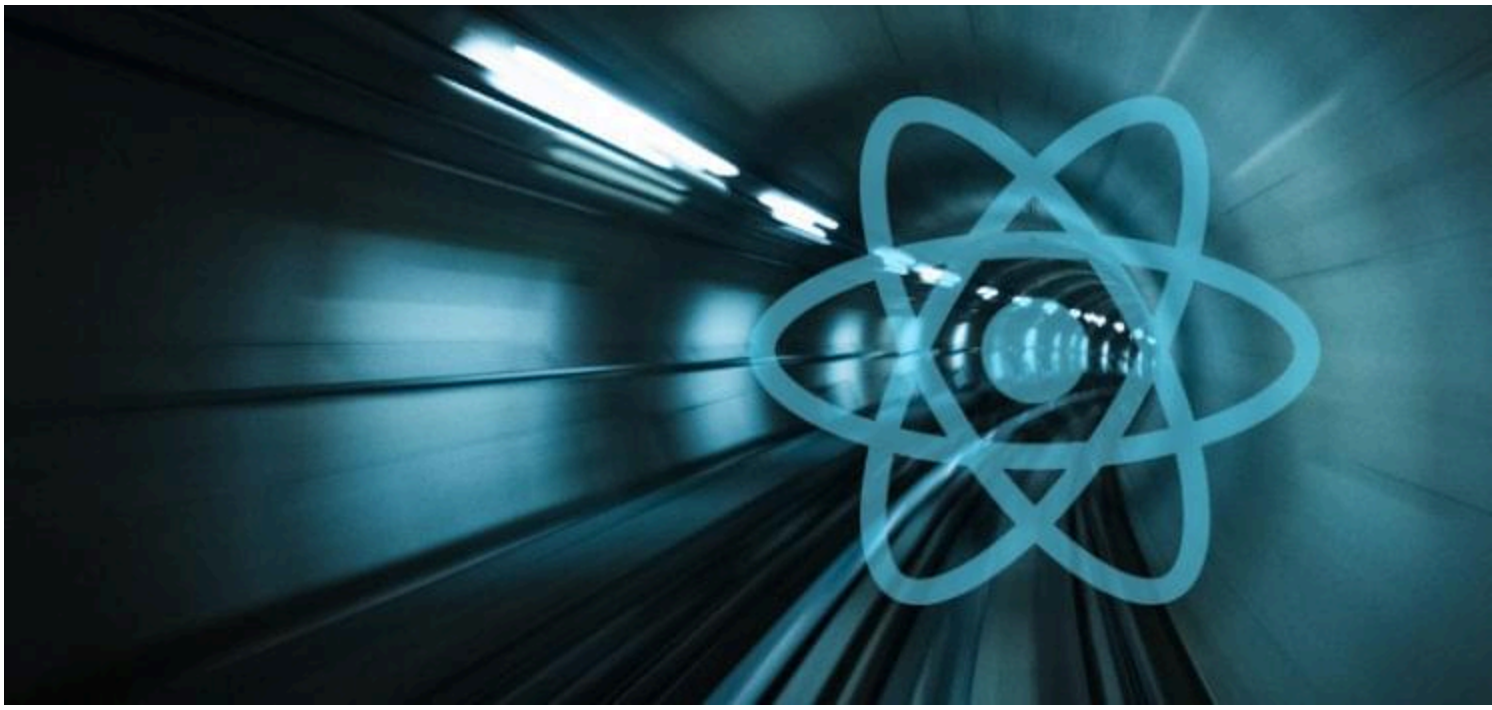
Eric Greene is a professional software developer specializing in HTML, CSS, and JavaScript technologies. Right now he is focused on Node.js, React, GraphQL, Relay, Angular (1 and 2), Backbone, and jQuery. He has been developing software and delivering training classes for nearly 19 years. He holds the MCSD Certification for ASP.Net Web Applications, and is a Microsoft Certified Trainer. Eric has worked with companies of all sizes in the insurance, nuclear engineering, publishing and communications industries. Among his many professional endeavors, Eric is a Technical Instructor at DevelopIntelligence.

Getting React Projects Ready Fast with Pre-configured Builds

By Pavels Jelisejevs

Starting a new React project nowadays is not as simple as we'd like it to be. Instead of instantly diving into the code and bringing your application to life, you have to spend time configuring the right build tools to set up a local development environment, unit testing, and a production build. But there are projects where all you need is a simple setup to get things running quickly and with minimal effort.

Create React App provides just that. It's a CLI tool from Facebook that allows you to generate a new React project and use a pre-configured Webpack build for development. Using it, you'll never have to look at the Webpack config again.



Getting React Projects Ready Fast with Pre-configured Builds

How Does Create React App Work?

Create React App is a standalone tool that should be installed globally via npm, and called each time you need to create a new project:

```
npm install -g create-react-app
```

To create a new project, run:

```
create-react-app react-app
```

Create React App will set up the following project structure:

```
.
├── .gitignore
├── README.md
├── package.json
├── node_modules
├── public
│   ├── favicon.ico
│   └── index.html
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
```

```
    ├── index.css
    ├── index.js
    └── logo.svg
```

It will also add a `react-scripts` package to your project that will contain all of the configuration and build scripts. In other words, your project depends `react-scripts`, not on `create-react-app` itself. Once the installation is complete, you can start working on your project.

Starting a Local Development Server

The first thing you'll need is a local development environment. Running `npm start` will fire up a Webpack development server with a watcher that will automatically reload the application once you change something. Hot reloading, however, is only supported for styles.

The application will be generated with a number of features built-in.

ES6 and ES7

The application comes with its own Babel preset, babel-preset-react-app, to support a set of ES6 and ES7 features. It even supports some of the newer features like async/await, and import/export statements. However, certain features, like decorators, have been intentionally left out.

Asset import

You can also import CSS files from your JS modules that allow you to bundle styles that are only relevant for the modules that you ship. The same thing can be done for images and fonts.

ESLint

During development, your code will also be run through ESLint, a static code analyzer that will help you spot errors during development.

Environment variables

You can use Node environment variables to inject values into your code at built-time. React-scripts will automatically look for any environment variables starting with `REACT_APP_` and make them available under the global `process.env`. These variables can be in a `.env` file for convenience:

```
REACT_APP_BACKEND=http://my-api.com
REACT_APP_BACKEND_USER=root
```

You can then reference them in your code:

```
fetch({process.env.REACT_APP_SECRET_CODE}/endpoint)
```

Proxying to a backend

If your application will be working with a remote backend, you might need to be able to proxy requests during local development to bypass CORS. This can be set up by adding a proxy field to your `package.json` file:

```
"proxy": "http://localhost:4000",
```

This way, the server will forward any request that doesn't point to a static file the given address.

Running Unit Tests

Executing `npm test` will run tests using Jest and start a watcher to re-run them whenever you change something:

```
PASS  src/App.test.js
  ✓ renders without crashing (7ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.123s, estimated 1s
Ran all test suites.

Watch Usage
 › Press p to filter by a filename regex pattern.
 › Press q to quit watch mode.
 › Press Enter to trigger a test run.
```

Jest is a test runner also developed by Facebook as an alternative to Mocha or Karma. It runs the tests on a Node environment instead of a real browser, but provides some of the browser-specific globals using jsdom.

Jest also comes integrated with your VCS and by default will only run tests on files changed since your last commit. For more on this, refer to "How to Test React Components Using Jest".

Creating a Production Bundle

When you finally have something you deploy, you can create a production bundle using `npm run build`. This will generate an optimized build of your application, ready to be deployed to your environment. The generated artifacts will be placed in the build folder:

```
.
├── asset-manifest.json
├── favicon.ico
├── index.html
```

```
└── static
    ├── css
    │   ├── main.9a0fe4f1.css
    │   └── main.9a0fe4f1.css.map
    ├── js
    │   ├── main.3b7bfee7.js
    │   └── main.3b7bfee7.js.map
    └── media
        └── logo.5d5d9eef.svg
```

The JavaScript and CSS code will be minified, and CSS will additionally be run through Autoprefixer to enable better cross-browser compatibility.

Deployment

React-scripts provides a way to deploy your application to GitHub pages by simply adding a homepage property to `package.json`. There's also a separate Heroku build pack.

Opting Out

If at some point you feel that the features provided are no longer enough for your project, you can always opt out of using react-scripts by running `npm run eject`. This will copy the Webpack configuration and build scripts from `react-scripts` into your project and remove the dependency. After that, you're free to modify the configuration however you see fit.

Recommended Courses

Wes Bos

A step-by-step training course to get you building real world React.js + Firebase apps and website components in a couple of afternoons. Use coupon code **'SITEPOINT'** at checkout to get **25% off**.

In Conclusion

If you're looking to start a new React project look no further. Create React App will allow you to quickly start working on your application instead of writing yet another Webpack config.

Have you given it a try yet? What did you think? Let me know in the comments!

This post was peer reviewed by Joan Yin. Thanks to all of SitePoint's peer reviewers for making SitePoint content the best it can be!

Pavels is a software developer from Riga, Latvia, with a keen interest for everything web-related. His interests range from back-end to front-end development, as well as analysis and automation. If you have something to discuss, you can always reach him via Facebook or LinkedIn.

React vs Angular: An In-depth Comparison

By Pavels Jelisejevs

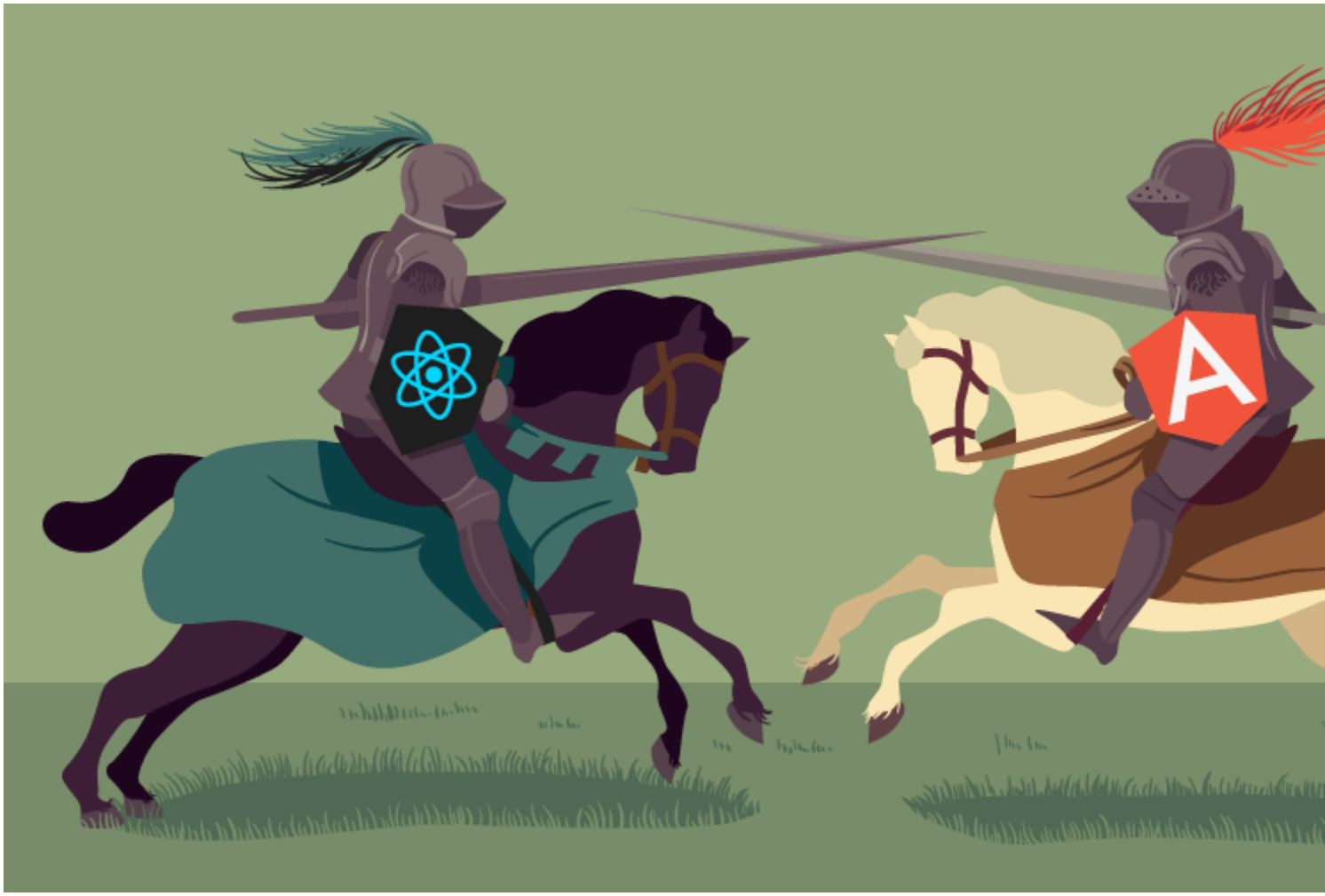
This article is included in our anthology, Modern JavaScript. If you want everything in one place to get up to speed on modern JavaScript, sign up for SitePoint Premium and download all our JavaScript books and courses for just $5.

Should I choose Angular or React? Today's bipolar landscape of JavaScript frameworks has left many developers struggling to pick a side in this debate. Whether you're a newcomer trying to figure out where to start, a freelancer picking a framework for your next project, or an enterprise-grade architect planning a strategic vision for your company, you're likely to benefit from having an educated view on this topic.

To save you some time, let me tell you something up front: this article won't give a clear answer on which framework is better. But neither will hundreds of other articles with similar titles. I can't tell you that, because the answer depends on a wide range of factors which make a particular technology more or less suitable for your environment and use case.

Since we can't answer the question directly, we'll attempt something else. We'll compare Angular (2+, not the old AngularJS) and React, to demonstrate how you can approach the problem of comparing any two frameworks in a structured manner on your own and tailor it to your environment. You know, the old "teach a man to fish" approach. That way, when both are replaced by a BetterFramework.js in a

year's time, you'll be able to re-create the same train of thought once more.



Two knights jousting, with React and Angular logos on their shields

Where to Start?

Before you pick any tool, you need to answer two simple questions: "Is this a good tool per se?" and "Will it work well for my use case?" Neither of them mean anything on their own, so you always need to keep both of them in mind. All right, the questions might not be that simple, so we'll try to break them down into smaller ones.

Questions on the tool itself:

- How mature is it and who's behind it?
- What kind of features does it have?
- What architecture, development paradigms, and patterns does it employ?
- What is the ecosystem around it?

Questions for self-reflection:

- Will I and my colleagues be able to learn this tool with ease?
- Does is fit well with my project?
- What is the developer experience like?

Using this set of questions you can start your assessment of any tool and we'll base our comparison of React and Angular on them as well.

There's another thing we need to take into account. Strictly speaking, it's not exactly fair to compare Angular to React, since Angular is a full-blown, feature-rich framework, while React just a UI component library. To even the odds, we'll talk about React in conjunction with some of the libraries often used with it.

Maturity

An important part of being a skilled developer is being able to keep the balance between established, time-proven approaches and evaluating new bleeding-edge tech. As a general rule, you should be careful when adopting tools that haven't yet matured due to certain risks:

- The tool may be buggy and unstable.
- It might be unexpectedly abandoned by the vendor.
- There might not be a large knowledge base or community available in case you need help.

Both React and Angular come from good families, so it seems that we can be confident in this regard.

React

React is developed and maintained by Facebook and used in their own products, including Instagram and WhatsApp. It has been around for roughly three and a half years now, so it's not exactly new. It's also one of the most popular projects on GitHub, with about 74,000 stars at the time of writing. Sounds good to me.

Angular

Angular (version 2 and above) has been around less then React, but if you count in the history of its predecessor, AngularJS, the picture evens out. It's maintained by Google and used in AdWords and Google Fiber. Since AdWords is one of the key projects in Google, it is clear they have made a big bet on it and is unlikely to disappear anytime soon.

Features

Like I mentioned earlier, Angular has more features out of the box than React. This can be both a good and a bad thing, depending on how you look at it.

Both frameworks share some key features in common: components, data binding, and platform-agnostic rendering.

Angular

Angular provides a lot of the features required for a modern web application out of the box. Some of the standard features are:

- Dependency injection
- Templates, based on an extended version of HTML
- Routing, provided by @angular/router
- Ajax requests by @angular/http
- @angular/forms for building forms
- Component CSS encapsulation
- XSS protection
- Utilities for unit-testing components.

Having all of these features available out of the box is highly convenient when you don't want to spend time picking the libraries yourself. However, it also means that you're stuck with some of them, even if you don't need them. And replacing them will usually require additional effort. For instance, we believe that for small projects having a DI system creates more overhead than benefit, considering it can be effectively replaced by imports.

React

With React, you're starting off with a more minimalistic approach. If we're looking at just React, here's what we have:

- No dependency injection
- Instead of classic templates it has JSX, an XML-like language built on top of JavaScript
- XSS protection
- Utilities for unit-testing components.

Not much. And this can be a good thing. It means that you have the freedom to choose whatever additional libraries to add based on your needs. The bad thing is that you actually have to make those choices yourself. Some of the popular libraries that are often used together with React are:

We've found the freedom of choosing your own libraries liberating. This gives us the ability to tailor our stack to particular requirements of each project, and we didn't find the cost of learning new libraries that high.

Languages, Paradigms, and Patterns

Taking a step back from the features of each framework, let's see what kind higher-level concepts are popular with both frameworks.

React

There are several important things that come to mind when thinking about React: JSX, Flow, and Redux.

JSX

JSX is a controversial topic for many developers: some enjoy it, and others think that it's a huge step back. Instead of following a classical approach of separating markup and logic, React decided to combine them within components using an XML-like language that allows you to write markup directly in your JavaScript code.

While the merits of mixing markup with JavaScript might be debatable, it has an indisputable benefit: static analysis. If you make an error in your JSX markup, the compiler will emit an error instead of continuing in silence. This helps by instantly catching typos and other silly errors.

Flow

Flow is a type-checking tool for JavaScript also developed by Facebook. It can parse code and check for common type errors such as implicit casting or null dereferencing.

Unlike TypeScript, which has a similar purpose, it does not require you to migrate to a new language and annotate your code for type checking to work. In Flow, type annotations are optional and can be used to provide additional hints to the analyzer. This makes Flow a good option if you would like to use static code analysis, but would like to avoid having to rewrite your existing code.

Redux

Redux is a library that helps manage state changes in a clear manner. It was inspired by Flux, but with some simplifications. The key idea of Redux is that the whole state of the application is represented by a single object, which is mutated by functions called reducers. Reducers themselves are pure functions and are implemented separately from the components. This enables better separation of concerns and testability.

If you're working on a simple project, then introducing Redux might be an over complication, but for medium- and large-scale projects, it's a solid choice. The library has become so popular that there are projects implementing it in Angular as well.

All three features can greatly improve your developer experience: JSX and Flow allow you to quickly spot places with potential errors, and Redux will help achieve a clear structure for your project.

Angular

Angular has a few interesting things up its sleeve as well, namely TypeScript and RxJS.

TypeScript

TypeScript is a new language built on top of JavaScript and developed by Microsoft. It's a superset of JavaScript ES2015 and includes features from newer versions of the language. You can use it instead of Babel to write state of the art JavaScript. It also features an extremely powerful typing system that can statically analyze your code by using a combination of annotations and type inference.

There's also a more subtle benefit. TypeScript has been heavily influenced by Java and .NET, so if your developers have a background in one of these languages, they are likely to find TypeScript easier to learn than plain JavaScript (notice how we switched from the tool to your personal environment). Although Angular has been the first major framework to actively adopt TypeScript, it's also possible to use it together with React.

RxJS

RxJS is a reactive programming library that allows for more flexible handling of asynchronous operations and events. It's a combination of the Observer and Iterator patterns blended together with functional programming. RxJS allows you to treat anything as a continuous stream of values and perform various operations on it such as mapping, filtering, splitting or merging.

The library has been adopted by Angular in their HTTP module as well for some internal use. When you perform an HTTP request, it returns an Observable instead of the usual Promise. Although this library is extremely powerful, it's also quite complex. To master it, you'll need to know your way around different types of Observables, Subjects, as well as around a hundred methods and operators. Yikes, that seems to be a bit excessive just to make HTTP requests!

RxJS is useful in cases when you work a lot with continuous data streams such as web sockets, however, it seems overly complex for anything else. Anyway, when working with Angular you'll need to learn it at least on a basic level.

We've found TypeScript to be a great tool for improving the maintainability of our projects, especially those with a large code base or complex domain/business logic. Code written in TypeScript is more descriptive and easier to follow. Since TypeScript has been adopted by Angular, we hope to see even more projects using it. RxJS, on the other hand, seems only to be beneficial in certain cases and should be adopted with care. Otherwise, it can bring unwanted complexity to your project.

Ecosystem

The great thing about open source frameworks is the number of tools created around them. Sometimes, these tools are even more helpful than the framework itself. Let's have a look at some of the most popular tools and libraries associated with each framework.

Angular

Angular CLI

A popular trend with modern frameworks is having a CLI tool that helps you bootstrap your project without having to configure the build yourself. Angular has Angular CLI for that. It allows you to generate and run a project with just a couple of commands. All of the scripts responsible for building the application, starting a development server and running tests are hidden away from you in `node_modules`. You can also use it to generate new code during development. This makes setting up new projects a breeze.

Ionic 2

Ionic 2 is a new version of the popular framework for developing hybrid mobile applications. It provides a Cordova container that is nicely integrated with Angular 2, and a pretty material component library. Using it, you can easily set up and build a mobile application. If you prefer a hybrid app over a native one, this is a good choice.

Material design components

If you're a fan of material design, you'll be happy to hear that there's a Material component library available for Angular. Currently, it's still at an early stage and slightly raw but it has received lots of contributions recently, so we might hope for things to improve soon.

Angular universal

Angular universal is a seed project that can be used for creating projects with support for server-side rendering.

@ngrx/store

[@ngrx/store]20 is a state management library for Angular inspired by Redux, being based on state mutated by pure reducers. Its integration with RxJS allows you to utilize the push change detection strategy for better performance.

> There are plenty of other libraries and tools available in the Awesome Angular list.

React

Create React App

Create React App is a CLI utility for React to quickly set up new projects. Similar to Angular CLI it allows you to generate a new project, start a development server and create a bundle. It uses Jest, a relatively new test runner from Facebook, for unit testing, which has some nice features of its own. It also supports flexible application profiling using environment variables, backend proxies for local development, Flow, and other features. Check out this brief introduction to Create React App for more information.

React Native

React Native is a platform developed by Facebook for creating native mobile applications using React. Unlike Ionic, which produces a hybrid application, React Native produces a truly native UI. It provides a set of standard React components which are bound to their native counterparts. It also allows you to create your own components and bind them to native code written in Objective-C, Java or Swift.

Material UI

There's a material design component library available for React as well. Compared to Angular's version, this one is more mature and has a wider range of components available.

Next.js

Next.js is a framework for the server-side rendering of React applications. It provides a flexible way to completely or partially render your application on the server, return the result to the client and continue in the browser. It tries to make the complex task of creating universal applications as simple as possible so the set up is designed to be as simple as possible with a minimal amount of new primitives and requirements for the structure of your project.

MobX

MobX is an alternative library for managing the state of an application. Instead of keeping the state in a single immutable store, like Redux does, it encourages you to store only the minimal required state and derive the rest from it. It provides a set of decorators to define observables and observers and introduce reactive logic to your state.

Storybook

Storybook is a component development environment for React. It allows you to quickly set up a separate application to showcase your components. On top of that, it provides numerous add-ons to document, develop, test and design your components. We've found it to be extremely useful to be able to develop components independently from the rest of the application. You can learn more about Storybook from a previous article.

> There are plenty of other libraries and tools available in the Awesome React list.

Adoption, Learning Curve and Development Experience

An important criterion for choosing a new technology is how easy it is to learn. Of course, the answer depends on a wide range of factors such as your previous experience and a general familiarity with the related concepts and patterns. However, we can still try to assess the number of new things you'll need to learn to get started with a given framework. Now, if we assume that you already know ES6+, build tools and all of that, let's see what else you'll need to understand.

React

With react the first thing you'll encounter is JSX. It does seem awkward to write for some developers, however, it doesn't add that much complexity; Just expressions, which are actually JavaScript, and a special HTML-like syntax. You'll also need to learn how to write components, use props for configuration and manage internal state. You don't need to learn any new logical structures or loops since all of this is plain JavaScript.

The official tutorial is an excellent place to start learning React. Once you're done with that, get familiar with the router. The react router v4 might be slightly complex and unconventional, but nothing to worry about. Using Redux will require a paradigm shift to learn how to accomplish already familiar tasks in a manner suggested by the library. The free Getting Started with Redux video course can quickly introduce you to the core concepts. Depending on the size and the complexity of your project you'll need to find and learn some additional libraries and this might be the tricky part, but after that everything should be smooth sailing.

Recommended Courses

Wes Bos

A step-by-step training course to get you building real world React.js + Firebase apps and website components in a couple of afternoons. Use coupon code **'SITEPOINT'** at checkout to get **25% off**.

We were genuinely surprised at how easy it was to get started using React. Even people with a backend development background and limited experience in frontend development were able to catch up quickly. The error messages you might encounter along the way are usually clear and provide explanations on how to resolve the underlying problem. The hardest part may be finding the right libraries for all of the required capabilities, but structuring and developing an application is remarkably simple.

Angular

Learning Angular will introduce you to more new concepts than React. First of all, you'll need to get comfortable with TypeScript. For developers with experience in statically typed languages such as Java or .NET this might be easier to understand than JavaScript, but for pure JavaScript developers, this might require some effort.

Recommended Courses

Todd Motto

The ultimate resource to learn Angular and its ecosystem. Use coupon code **'SITEPOINT'** at checkout to get **25% off**.

The framework itself is rich in topics to learn, starting from basic ones such as modules, dependency injection, decorators, components, services, pipes, templates, and directives, to more advanced topics such as change detection, zones, AoT compilation, and Rx.js. These are all covered in the documentation. Rx.js is a heavy topic on its own and is described in much detail on the official website. While relatively easy to use on a basic level it gets more complicated when moving on to advanced topics.

All in all, we noticed that the entry barrier for Angular is higher than for React. The sheer number of new concepts is confusing to newcomers. And even after you've started, the experience might be a bit rough since you need to keep in mind things like Rx.js subscription management, change detection performance and bananas in a box (yes, this is an actual advice from the documentation). We often encountered error messages that are too cryptic to understand, so we had to google them and pray for an exact match.

It might seem that we favor React here, and we definitely do. We've had experience onboarding new developers to both Angular and React projects of comparable size and complexity and somehow with React it always went smoother. But, like I said earlier, this depends on a broad range of factors and might work differently for you.

Putting it Into Context

You might have already noted that each framework has its own set of capabilities, both with their good and bad sides. But this analysis has been done outside of any particular context and thus doesn't provide

an answer on which framework should you choose. To decide on that, you'll need to review it from a perspective of your project. This is something you'll need to do on your own.

To get started, try answering these questions about your project and when you do, match the answers against what you've learned about the two frameworks. This list might not be complete, but should be enough to get you started:

1. How big is the project?
2. How long is it going to be maintained for?
3. Is all of the functionality clearly defined in advance or are you expected to be flexible?
4. If all of the features are already defined, what capabilities do you need?
5. Are the domain model and business logic complex?
6. What platforms are you targeting? Web, mobile, desktop?
7. Do you need server-side rendering? Is SEO important?
8. Will you be handling a lot of real-time event streams?
9. How big is your team?
10. How experienced are your developers and what is their background?
11. Are there any ready-made component libraries that you would like to use?

If you're starting a big project and you would like to minimize the risk of making a bad choice, consider creating a proof-of-concept product first. Pick some of the key features of the projects and try to implement them in a simplistic manner using one of the frameworks. PoCs usually don't take a lot if time to build, but they'll give you some valuable personal experience on working with the framework and allow you to validate the key technical requirements. If you're satisfied with the results, you can continue with full-blown development. If not, failing fast will save you lot of headaches in the long run.

One Framework to Rule Them All?

Once you've picked a framework for one project, you'll get tempted to use the exact same tech stack for your upcoming projects. Don't. Even though it's a good idea to keep your tech stack consistent, don't blindly use the same approach every time. Before starting each project, take a moment to answer the same questions once more. Maybe for the next project, the answers will be different or the landscape will change. Also, if you have the luxury of doing a small project with a non-familiar tech stack, go for it. Such experiments will provide you with invaluable experience. Keep your mind open and learn from your mistakes. At some point, a certain technology will just feel natural and *right*.

This article was peer reviewed by Jurgen Van de Moere and Joan Yin. Thanks to all of SitePoint's peer reviewers for making SitePoint content the best it can be!

Pavels is a software developer from Riga, Latvia, with a keen interest for everything web-related. His interests range from back-end to front-end development, as well as analysis and automation. If you have

something to discuss, you can always reach him via Facebook or LinkedIn.