Comparison of Frontend JavaScript Frameworks

Introduction

* Hi there! My name is James… excited to be here :)

React vs. Angular vs. Vue

* In this talk we'll be comparing React, Angular, and Vue
* We've got 30 minutes, so let's get down to business… cut to the chase… whatever you want to say…

Tell me about yourself

* Who is already using one of these frameworks?
* Who has been using React for over a year? Angular? Vue?
* Which of these are you most interested in learning something about?

Goals

* The goal of this talk isn't to tell you everything that you need to know about these frameworks in order to be successful
  * It wouldn't be possible to do that for just one of these frameworks let along three
* The goal is to give you an impression of what it'd be like to work with each of these frameworks so that you can decide which you may want to take a closer look at
  * Or if you're already using a framework, whether or not it'd make sense for you to take a closer look at a different framework

Out of Scope

There's so much to talk about... something's got to go!

Routing
State Management
Universal Rendering
Mobile Development

We’re not going to have time to talk any of these things

What is a Frontend Framework?

* What is a frontend framework?
* What are React, Angular, and Vue?

React

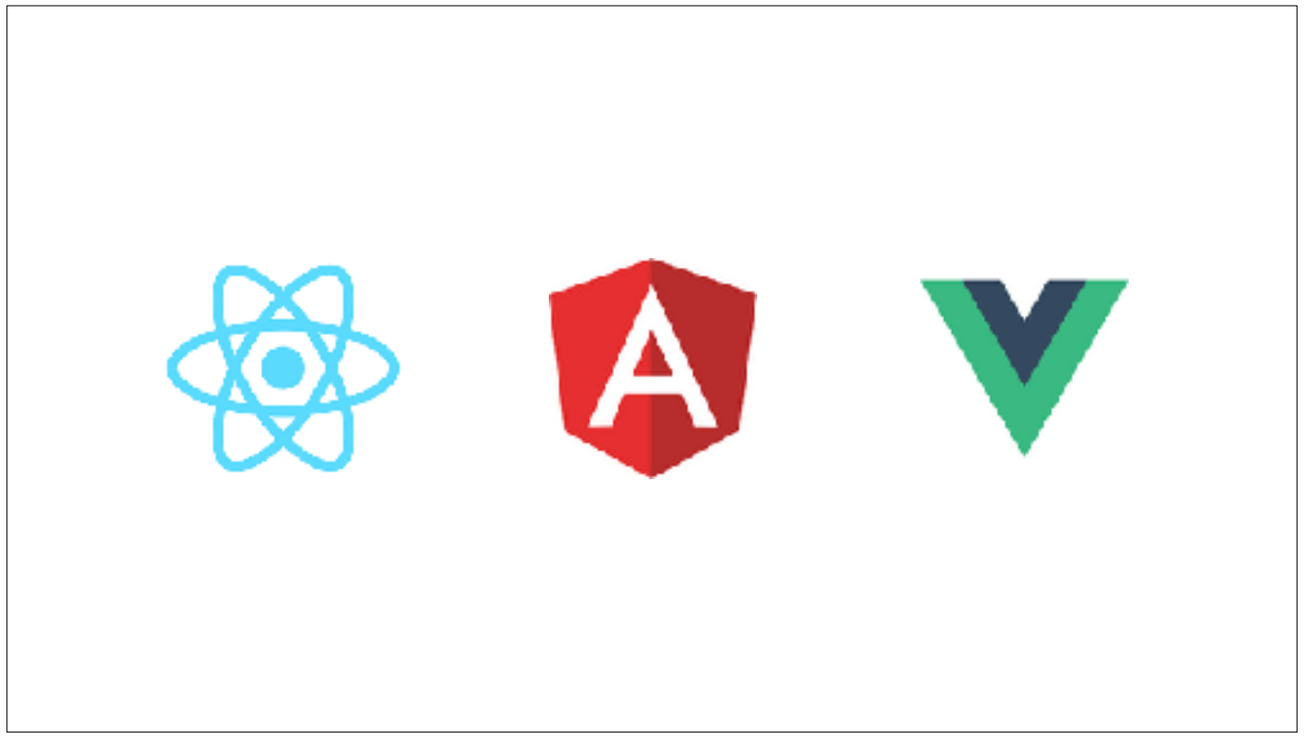
A JavaScript library for building user interfaces

Angular

Learn one way to build applications and reuse your code
and abilities to build apps for any deployment target

Vue

A progressive framework for building user interfaces



* React and Vue are at their core more focused on building UIs
* Angular is more focused on building applications
* If you extend React and Vue with other libraries (as most devs do), they also can be used to build applications

How Do They Work?

* We're not going to spend a lot of time on this, all three have the following high level concepts in common:

Component abstraction

Some type of change detection

Some type of DOM update optimization

Front end build

* Component abstraction
  * Hand-waving
* Some type of change detection
  * Hand-waving
* Some type of DOM update optimization
  * Hand-waving
* Front end build
  * Transpilation or compilation (ala Babel or TypeScript)
  * Bundling (ala webpack)
  * Hand-waving

Components

* It's all about components now
* We'll start with comparing how you get started with each, but we'll primarily be comparing how each handles component development

Creating Projects

create-react-app

```
> npx create-react-app my-app
```

Getting Started

* create-react-app
* No installation necessary!
  * You CAN install it globally if you'd like, but the docs just recommend using npx

[User Guide](https://github.com/facebook/create-react-app/blob/master/packages/react-scripts/template/README.md)

Project Creation

```
npx create-react-app my-app
```

Getting Started

* Angular CLI
* Install it globally to give you access to the `ng` command

```
npm install -g @angular/cli
```

https://cli.angular.io/

Project Creation

```
ng new my-app
```

Vue CDN
“Downgraded”

```
<script
  src="https://unpkg.com/vue@2.5.13/dist/vue.js">
</script>
```

Getting Started

* Include Vue via a CDN and a `<script>` tag
* No build process

Project Creation

* Add an HTML file and one or more JS files

Vue CLI
“Upgraded”

```
> npm install -g @vue/cli
> vue create my-app
```

Getting Started

* Vue CLI
* Install it globally to give you access to the `vue` command

```
npm install -g @vue/cli
```

https://github.com/vuejs/vue-cli/blob/dev/docs/README.md

Project Creation

```
vue create my-app
```

Comparison

```
> npx create-react-app my-app
> ng new my-app
> vue create my-app
```

* All three frameworks have a very similar project creation process
* The Vue CLI offers an interesting interactive mode to select the features you want your project to include
* The Angular CLI can be used throughout development (it's not just used for project creation)
  * Component and service creation
  * New `update` and `add` commands can install packages and run migration and installation scripts
  * Workspaces can contain multiple application and library projects

Development Workflow

* All of the CLI options offer a similar experience
* Browse to the root of the project and start the dev server...

```
npm start
ng serve
npm run serve
```

...and edit your app and review the change in the browser :)

With "downgraded" Vue projects, you're on your own (no build process, no webpack dev server)

Production Builds

* All of the CLI options offer a similar experience

```
npm run build
ng build --prod
npm run build
```

* React generates files into a `build` folder
* Angular generates files into a `dist` folder
* Vue generates files into a `dist` folder

(Basic) Component Development

* Templating
* Properties
* Events

Switch to markdown file…

Highlights and Opinions

Pros

- Relatively simple to learn
- Elegant in its design
- Focus on functional programming
- Broad community support
- Approach encourages you to decompose your app into smaller and smaller components

Cons

- Can feel a bit constraining or foreign at times
- Many ways to accomplish the same thing (i.e. less opinionated)
- You can't leverage native HTML

Pros

- Capable CLI
- All inclusive and opinionated
- Leverages native HTML and CSS
- For a longtime C# dev, coding in TypeScript feels comfortable (and safe)

Cons

- Relatively verbose
- Can feel heavy handed or overly complex at times
- Difficult to getted started without the assistance of the CLI

Pros

- Downgraded mode is brilliant (and more importantly, it's fun!)
- Less opinionated (especially when compared with Angular)
- Leverages native HTML and CSS

Cons

- Less community support (at this time)
- No "one way" to setup projects

Making a Choice

How do you decide on which to use?

* What does your dev team look like?
* What kind of app are you building?
* What features are most important to you?

Performance

Is one framework the clear leader in performance?

http://www.stefankrause.net/js-frameworks-benchmark7/table.html

Performance, in most cases, is probably as not as much of a differentiating factor as it maybe once used to be

Other Considerations

- CLIs
- Forms
- Universal or Server-side Rendering
- Mobile Dev
- Docs
- Learning Resources
- Tooling Support
- Popularity
- Support
- Community

Wrap Up

Don’t Over Think It

* You can be successful with any of these frameworks
  * Arguing over which of these is better is a little like arguing over which ice cream flavor is "better"

Shared Concepts

- SPAs
- Components
- Templating and Binding
- Virtual DOM
- Data Flow
- State Management
- Backend APIs
- Routing
- Language Abstractions
- Testing
- Front End Builds

* What are the conceptual things that you need to know in order to be successful with any of these frameworks?
* As you learn and gain skill with these concepts you can apply them to any framework, current or future

Conclusion

* Things will continue to change and evolve
* Picking the "right" framework is important, but it's more important to focus on the core concepts and skills that front end devs need to have in order to be successful
  * That will also equip you for whatever comes next... because you better believe that something new is right around the corner
* Don't worry too much about it... just make a choice and do your best :)

Thanks!

James Churchill

Treehouse: https://teamtreehouse.com/jameschurchill
Twitter: @SmashDev
GitHub: smashdevcode