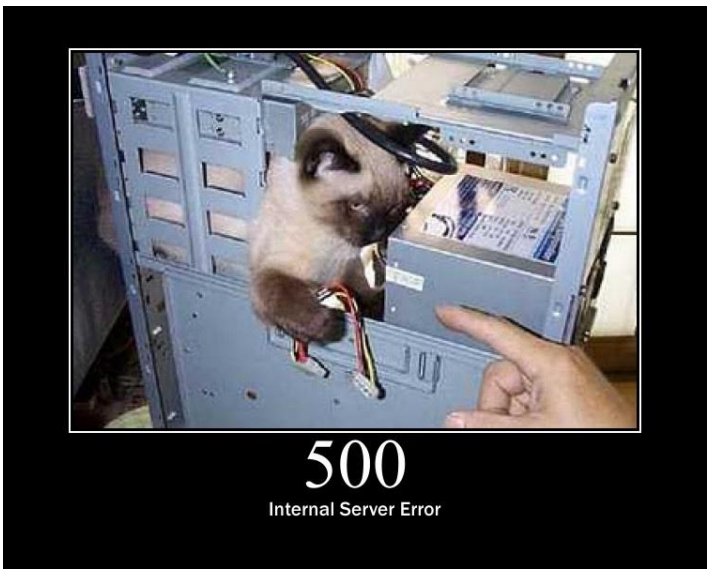
A Comparative Assessment of Angular and Vue.js



Index

Angular and Vue are solutions to managing the development of client side applications.

Angular is considered a platform, where the vendor API includes features like routing, form validation and other solutions.

Vue is a view library, based on the templating style. It's recently become more platform-like by including a custom state management solution and routing.

Both Vue and Angular are extremely powerful and similar in their approach to view management.
In this document, I aim to compare the differences of the two, from both a technical and high level perspective.

Benefits / Use Case

- Vue
    - Easy to get started/learn
    - Applications are composable very quickly
    - Flexible
    - Gives you the ability to put together your own solution, typescript, scss is optional
    - Great for putting together a project quickly
- Angular
    - Robust and highly structured
    - Batteries included – Validation, HTTP, CSS pre-processors, singletons (services)
    - It's opinionated making it scalable, maintainable and simplifies the process of finding staff who know how to work within the platform.
    - Flexible, Angular can be used as just a view layer so if you want to use your own HTTP solution, you don't need to use the Angular tools
    - High performance, lots of documentation/tutorials/stack overflow activity and because the API hasn't changed, answers for older versions of Angular are relevant today

Vue is evolving rapidly, new features are published frequently. In the 4 years of operation, they have had 3 major releases, with a 4th currently in beta. Each major release had breaking changes .

Angular is a robust product that places itself as a platform to provide slow, dependable upgrades. Long term support for the API allows teams to migrate to newer APIs over years while still being able to update their existing codebase, receiving the benefits of new performance and security updates, without requiring any code rewrites.

Ultimately, the choice of the two products prompt the following questions:

- Do we value the ability to rapidly prototype a project?
- Do we value a fast paced, evolving framework, or a dependable API where staff can grow experienced within?
- Do we care about a market validated/tested platform?
- Do we value the shelf life of the knowledge staff acquire?
- Do we value project testability?
- Do we value rolling our own solutions or using vendor maintained solutions (batteries)?
- Do we value upgrading the codebase frequently for new features, or stable LTS releases with long notice periods for API deprecations?
- Do we value employees with experience in an uniform, opinionated environment, or who have worked within a specific iteration of Vue?

Inception

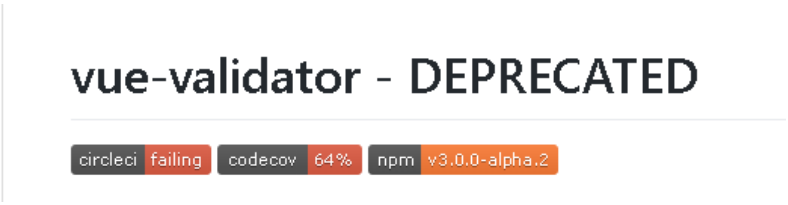
- Vue
  - Based on AngularJS 1.x, aimed at being a better version of Angular 1
- Angular
  - Learned from the mistakes of Angular 1, analysed the market, emerging standards and JS ecosystem (React, Web components, Redux)

Age

- Vue
  - 4 years
- Angular
  - 2 years
  - 8 Years if you count AngularJS

Changes

- Vue
  - 3 main releases 4 years
  - Breaking changes in main releases
  - Evolving best practices
  - Official/popular packages are community driven and have been deprecated at the whim of the developers, being replaced with other community packages



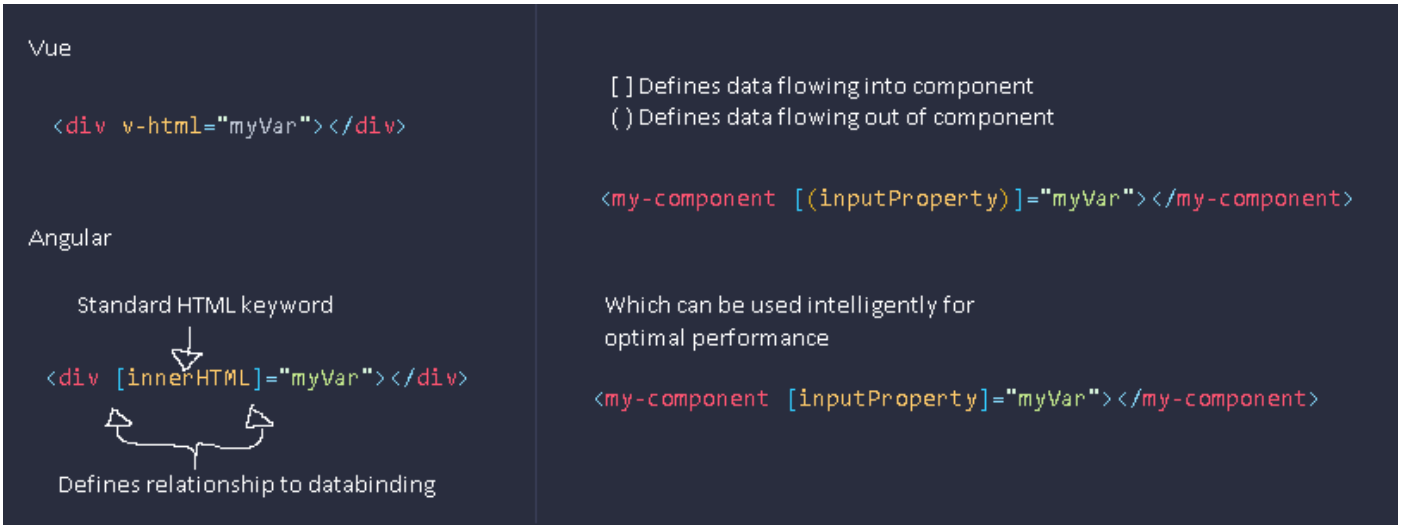
  - Vue 3 is coming with breaking changes
  - Vue CLI has two versions, with the recommended one being in beta, serving Vue 2.5 with different best practices to the current one which is the 5th listing down on google.
- Angular
  - Nearly no breaking changes since RC (2 years)
  - Sem-versioning with a major update every 6 months
  - LTS supported for 1 year and deprecated changes are held for 2 years

Updates

- Vue
  - Vue 3 is coming out
  - No notice of deprecated changes, can get stuck on V2.5 with no road to upgrade.
- Angular
  - Angular updates every 6 months with performance and feature updates.
  - The API stays the same, and any deprecated changes you're notified for 2 years before removal

Syntax

- Vue
  - Mostly the same
  - Uses custom syntax for bindings
  - Components are semantic, declared using ES5 syntax
- Angular
  - Mostly the same
  - Javascript syntax for bindings (innerHTML, value, classList)
  - Components are classes



State Management

- Vue
  - Vuex

  - Mutates state
  - Custom tooling
  - Does not like immutable changes
- Angular
  - Redux

  - Years of redux community tooling
  - Immutable state
  - Practices developed over time by both the React and Angular communities
  - Uses the official redux tools used by hundreds of thousands of developers
  - Redux time travel allows for highly testable solutions, allowing QA to send a state to a developer to help recreate a bug,
  - State history allows the production client to self-report a bug when an error occurs, sending back the steps the client took to get to the error, allowing developers to recreate it.
  - Importing state allows for rapid development (no need to log in, click a menu item, scroll, etc, to get to the place you're working on)

Components

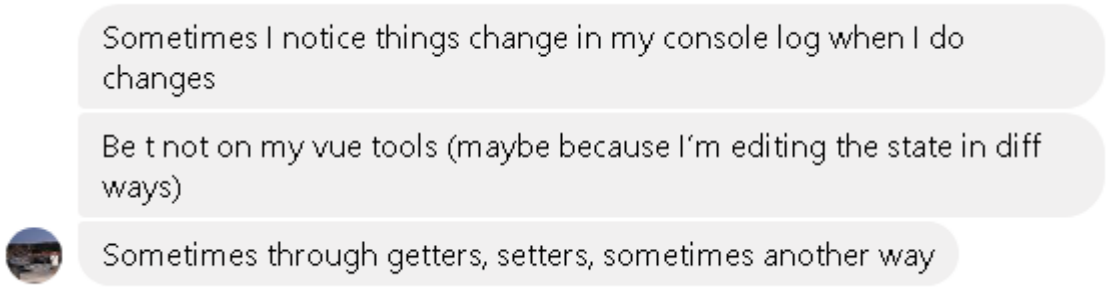
- Vue
    - Single file components with limited support for file splitting
    - Single file support seems to have issues with *eslint* and *vue-loader*, possibly fixed in Vue 3 or the new CLI (must be prepared to upgrade).
    - Looks like web components
    - Not actual web components and don't have a host component
    - Semantic labelling requires enforced standards on component labels
- Angular
    - Full support for single and multi file components
    - Actually compiles to native web components
    - Platform independent – With Angular components being native web components, your Angular component library can be extracted (using the CLI ng packagr) and used in Vue, React, Vanilla, whatever.

Code splitting

Both support code splitting and hot loading

Hot Reload / Tooling

- Vue
    - Hot reload doesn't clear console between reloads leaving old errors in console, requiring hard refresh
    - Poor error messages
    - My personal experience with the Vue tools has been pretty average. They aren't very informative and have lots of "gotchya"s. You ignore them mostly, but these things have never happened using the Angular / Redux tools.
    - The following is a conversation snippet from a friend of mine in new Zealand who has been having a really tough time using Vue for a large application.



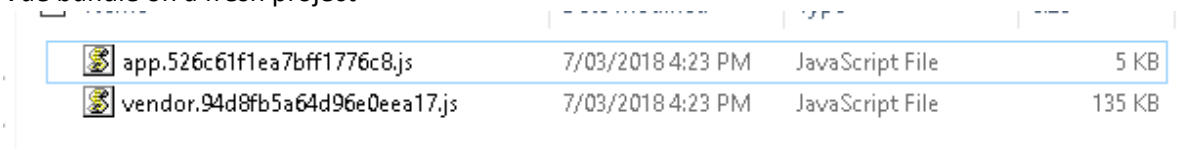
- Angular
    - Proper hot reload, where reloads clear the console however only reload the module edited for quick compilation.
    - Descriptive error messages

Server Side Rendering

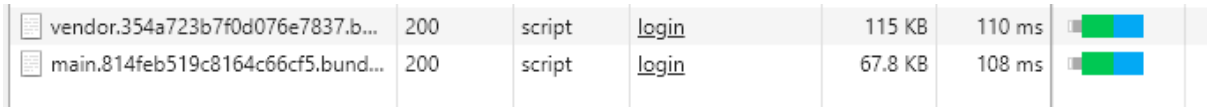
- Vue
  - Third party project Nuxt.js
- Angular
  - Angular core project Angular Universal

Bundle sizes *(Look at vendor.js – that's the platform's code)*

- Vue (135kb)
  - Vue bundle on a fresh project



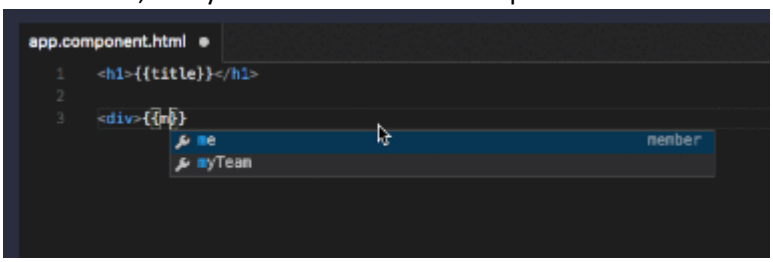
- Angular (115kb)
  - This is an old Angular 2 project of mine and included most of the base packages



  - Angular 6 will have smaller bundle sizes

Features

- Vue
  - Templating engine
  - Routing solution
  - Webpack 3
  - ES6 support, updates to ES7/ES8 will require project upgrades.
- Angular
  - Templating engine
  - Routing solution
  - Webpack 4
  - Seamless ES8 support, with updates to the ES standard being part of the non-breaking release
  - Optional type support
  - Built in singleton pattern (services)
  - Intellisense, everywhere – even in the template



Security

- Vue
    - Does not escape/sanitize by default
- Angular (better)
    - Sanitizes anything placed on the DOM, requiring a manual and very deliberate security override to add unsafe HTML

Backing

- Vue
    - Vue was created by Evan You after working for Google using AngularJS in a number of projects. He later summed up his thought process, "I figured, what if I could just extract the part that I really liked about Angular and build something really lightweight without all the extra concepts involved?"[6] Vue was originally released in February 2014. - Wikipedia
    - Evan You was referring to Angular 1.xx
    - It's still mostly developed by Evan, however it's more community driven now
- Angular
    - The platform is developed by Google in conjunction with Microsoft.
    - Active development team resourced by Google, with community packages worth their salt absorbed by Google, resourced and elevated to core supported libraries. (Angular Universal)

Same Component, one written in Vue and one written in Angular using the single file syntax (no Typescript syntax)

```
<template>
<div class="namespace-myvalue-host">
    {{ myValue }}
</div>
</template>

<style scoped>
.namespace-myvalue-host {
    color: red
}
</style>

<script>
export default {
    name: 'zip-button',
    props: {
        myValue: undefined
    },
    data () {
        return { };
    }
};
</script>
```

```
import { Component, Input } from '@angular/core'

const Template = `
    {{ myValue }}
`


const Styles = `
    :host {
        color: red
    }
`


@Component({
    selector: 'app-fab',
    template: Template,
    styles: [Styles]
})
export class MyComponent {
      @Input() private myValue = ''
}
```

Code Challenge

The following is invalid Vue code, try to figure out why, you may use google – the answer is on the next page.

```
<template>
<div class="namespace-myvalue-host">
    <ul>
        <li
            v-for="(item, key) in myValue"
            :key='key'
            >{{ item }}
        </li>
    </ul>
</div>
</template>

<script>
export default {
    name: 'namespace-myvalue',
    props: {
        myValue: []
    },
    data () {
        return { };
    }
};
</script>
```

Usage: (you can use a variable from the component's controller rather than an inline array)

```
<namespace-myvalue
    :myValue="['one', 'two']"
    >Sign In
</namespace-myvalue>
```

The compiler's output:

```
 DONE  Compiled successfully in 578ms

 I  Your application is running here: http://localhost:8081
```

Chrome Devtools

```
▶[Vue warn]: Invalid prop: type check failed for prop "myValue". Expected , got
Array.

found in

---> <ZipButton> at src\components\buttons\button\button.component.vue
       <SignInView> at src\views\sign-in\sign-in.view.vue
         <AppComponent> at src\app.component.vue
           <Root>
```

Answer

An input property MUST be initialised to (undefined)
If you don't initialise it, Vue will fail silently, if you initialise it to a value, Vue will fail silently.
You must assign a value in the component start lifecycle hook, if the value requires assignment (for example it has a default value)

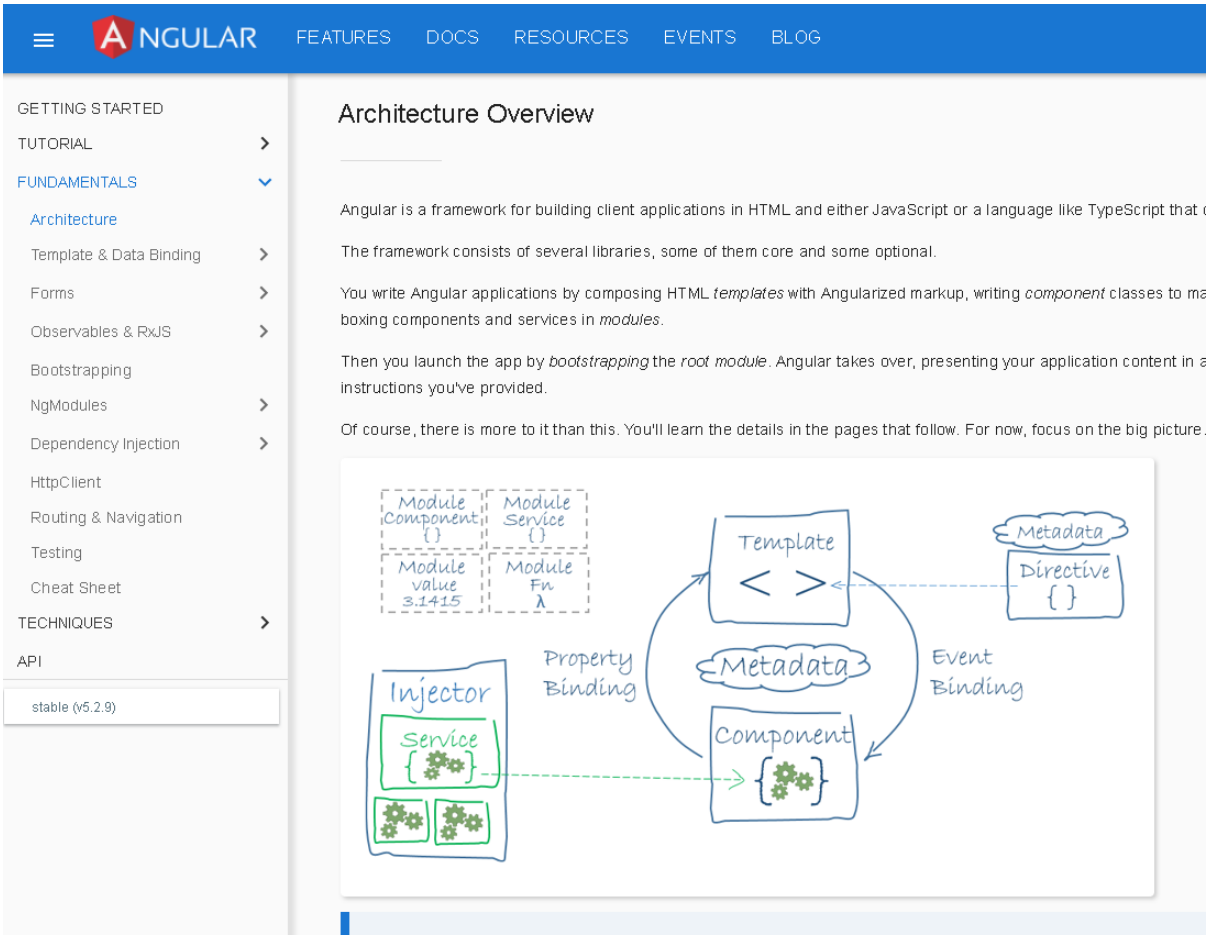
```
<template>
<div class="namespace-myvalue-host">
    <ul>
        <li
            v-for="(item, key) in myValue"
            :key='key'
            >{{ item }}
        </li>
    </ul>
</div>
</template>

<script>
export default {
    name: 'zip-button',
    props: {
        myValue: undefined
    },
    data () {
        return { };
    }
};
</script>
```

While this is in the documentation as the last part of the Component Usage under the section of Prop Validation – it is hidden away under a subject matter "Prop Validation" where it talks about ensuring types, with understanding how to set the default value being a consequence of reading the information on another topic in that section – type validation.

Which leads me to my last point.

Documentation

- Vue
  - Rich
  - All documentation on a single page
- Angular
  - Comprehensive
  - Has deprecation notices on every API
  - Tutorials
  - Videos
  - Test projects
  - In browser examples hosted on https://stackblitz.com/



Recommendation

Vue is great, Angular is great, in fact both Vue and Angular are almost identical in both syntax and ease of use – however Vue covers a smaller scope and doesn't have the support, long term consistency, testability, community (employees) or performance of Angular.

Vue also has lots of "gotchyas" which are constantly changing and moving based on the development cycle – however because there is no easy migration path, your staff will need to focus their experience in using a specific version of Vue.

I'm highly invested in this company, the product and would like the use technologies that most aligns with our long-term goals.

From my research, it's my personal opinion that Angular is more suited to our use case. I believe the stability of the Angular API leads itself to securing a longer shelf life for the product.

I believe that finding quality staff will be easier with Angular.

I believe that the skills our current staff will learn using Angular will grow them more professionally, than Vue – mainly because Vue isn't popular in production.

Relating to that, as a financial technology company, I feel it's a wise move to not take a gamble on a technology that has the potential to drop support suddenly or require a high amount of effort to upgrade APIs in order to get security or performance benefits as we don't want another case where we are stuck on an old version with no upgrade path.

If you have any comments or questions or just want to discuss please get back to me.