The front-end framework AoA

27 April 2018

Executive Summary

Due to the recent announcement that AngularJS (Angular 1.x) will no longer be supported at the beginning of 2021, the Enterprise Data Dissemination Environment (EDDE) technical team identified a programmatic risk for CEDSCI and recommends that the platform execute a migration strategy to a supported javascript (js) framework. The team conducted a marketplace analysis of leading javascript frameworks and found that Angular (Angular 2.x), ReactJS, and Vue.js were the most suitable frameworks based on EDDE technical requirements. The team then built "Hello World" applications as a means to further test the frameworks, weighing both technical and operational benefits of each framework. The team concluded that Vue.js is the most appropriate replacement framework. ReactJS was identified as a strong secondary alternative to Vue.js while Angular had characteristics that were less than ideal.

Table of Contents

        * Performance
    - React
        * Development Paradigm, Tooling, and Lifecycle
        * Learning curve
        * Components
        * Templates
        * CSS management
        * Performance
    - Vue
        * Development Paradigm
        * Learning curve
        * Components
        * Templates
        * CSS Management
        * Performance
- Hello World
    - Process
        * Application requirements
            · Angular
            · React
            · Vue
        * Results
- Decision
    - Expected Level of Effort
- References

Background

On January 26th, 2018 the maintainers of AngularJS announced that version 1.7 will be the team's final significant release, and that AngularJS will enter a 3 year Long Term Support (LTS) period, after which time support for the framework will now longer be provided; all development for AngularJS 1.x will cease, and migration strategies to Angular (version 2.x) or other comparable js frameworks should be developed for AngularJS (1.x) systems. No new systems should be developed using AngularJS.

AngularJS is a javascript framework that assists in developing and maintaining web applications, and is the framework currently used by EDDE. One important point to note is that the EDDE front-end currently uses AngularJS version 1.5, and therefore would not benefit from the issuance of a LTS period without first migrating to version 1.7. Consequently for EDDE as the primary enterprise dissemination service as of June 2019 and a Decennial system responsible for the dissemination of Decennial data in 2021, the announcement of a final AngularJS release and the beginning of a LTS period represents a programmatic risk

for CEDSCI, and an operational risk to the enterprise as it compromises the maintainability of the EDDE platform.

Risk

Should the EDDE platform not migrate to a supported JS framework, then the EDDE UI will be required to actively maintain the AngularJS framework during the initial release of 2020 data and into the foreseeable future.

Mitigation strategies

Two potential mitigation strategies are as follows: (1) migrate to an actively maintained front-end framework (Angular or other mainline alternatives); or (2) continue to support AngularJS past the LTS period through direct contract with Google (AngularJS sponsor) or another third-party vendor.

The first mitigation strategy consists of either migrating the EDDE front-end to Angular (which is the direct successor of AngularJS) or a competing javascript framework. Due to the proliferation of modern javascript front-end frameworks, several viable open-source options exist in the marketplace. The level of effort required to migrate from AngularJS to Angular is comparable to migrating to other mainline JS frameworks, and - importantly - a significantly larger effort than upgrading from AngularJS 1.5 to Angular 1.7. As a result, the CEDSCI program found it appropriate to consider additional JS frameworks alternatives in the mitigation strategy.

The second mitigation strategy serves to extend the inevitable solution of migrating from AngularJS and should be executed only under the circumstances that any viable migration strategy compromises the CEDSCI program's milestones and obligations to June 2019 production and Decennial operations. Ultimately, the imperative questions facing the CEDSCI program is what javascript framework should be used to replace AngularJS, and can a migration occur prior to June 2019.

Timing

As a Decennial system, the EDDE platform is included in 2020 activities that include the 2020 Test Readiness Review and Operational Readiness Review schedules, it is critical that risk mitigation strategies be employed

Methodology

To identify the most suitable javascript framework available for EDDE usage, CEDSCI staff executed a three-phase approach: 1. Conduct a market analysis of mainline javascript frameworks to identify the most appropriate replacements available today; 2. Develop 'Hello World' applications that establish the time, tooling, and packages required to instantiate an baseline application, ensuring specific EDDE requirements such as componentization is demonstrated; and 3. Produce a series of final recommendations for CEDSCI leadership and oversight entities.

The market analysis section summarizes certain criteria about three JavaScript frameworks: Angular 5.x, React 16.x, and Vue 2.x. It is a research document by nature, aggregating information gleaned from public sources.

The Hello World section documents the process of standing up a minimal 'hello world' application using each of the three js frameworks. The processs relies solely on each of the framework's documentation resources, and only deviates from their specific instructions through the addition of specific capabilities required by EDDE, namely routing and state management.

The recommendations sections synthesizes the prior sections individual summaries, and delivers a final framework recommendation and a migration strategy for the EDDE platform.

Team

The EDDE technical team is comprised of a combination of federal and contracting staff who are contributing to the CEDSCI program:

- Brian Barenbaum
- Romir Campbell
- Brian Campo
- David MacCormack
- Ray Vargas
- Richie Wang
- Zach Whitman

Market Analysis

There are dozens of web application JavaScript frameworks available (1,2,3), that each have different benefits and drawbacks given the EDDE use case. Given the sheer number of potential frameworks, the EDDE technical team removed js frameworks that were either deemed too new, experimental, or lacking a dedicated

development team. This initial preselection left the following frameworks for initial consideration:

- Angular
- Backbone
- Elm
- Ember
- Inferno
- Polymer
- Preact
- React
- Vue

Following the preselection, the EDDE team concluded that a second round down-select was necessary, and that the top three most popular frameworks should be given further consideration. The decision to focus on these three was driven by a number of factors, but the primary drivers were: 1) popularity; and 2) industry adoption. Though there's no single definitive metric to measure popularity and adoption a survey of community writing and discussions seems to indicate that Angular (5.x) and React (16.x) are roughly equally popular and Vue (2.x) is less so, but gaining very quickly (4). All three libraries are open source and released under a 2-clause MIT license.

The table below summarizes some of information with details in following sections.

Angular	React	Vue
Corporate lead	Google	Facebook
GitHub - watchers	3,007	5,761
GitHub - stars	35,272	93,928
GitHub - forks	8,601	17,703
GitHub – contributors	622	1,177
NPM downloads last week	663,117	2,417,516
Indeed.com jobs in DC metro area	3,000	1,000
Programming paradigm	Imperative, Reactive	Functional
Primary language	TypeScript	HTML+JavaScript
Prescriptive	High	Low
Learning curve	Steep	Moderate
HTML data binding	Two-way	One-way
Framework tooling	Required	Recommended

Angular

Angular is a TypeScript-based, open-source front-end web application platform led by the Angular Team at Google and by a community of individuals and

corporations. Angular is a complete rewrite from the same team that built AngularJS. While the current EDDE user interface is written in AngularJS, the level of effort required to move to Angular 5 is comparable to that of any other framework.

Development Paradigm, Tooling, and Lifecycle

Angular-based software (components and applications) are written in TypeScript. This brings both advantages and disadvantages.

- **Advantages:** compile-time support for traditional OOP features including classes (inheritance, encapsulation, etc) static type checking, and generics.
- **Disadvantages:** a significant investment to integrate and maintain tooling in the development lifecycle, both on developer's workstations/VDI and server-side.

Angular is prescriptive with respect to routing and state management (5,6). Though community alternatives exist (it is possible to use Redux with Angular), by convention there is an Angular-prescribed method. This has the benefit that functionality like deep linking may be available by default (depending on requirements), however, it is less flexible and that may be an issue when trying to achieve custom behavior or address performance problems.

Learning Curve

One thing that seems to be generally accepted is that Angular has a very steep learning curve (4,7,8). One potential reason for this is that Angular has much in common with an application development platform and for all intents and purposes, developers must use Angular's tooling in order to produce a component and/or application.

Components

By convention, developers develop Angular components and then build an application by tying those components together. Based on internal discussions with the EDDE technical team, Angular does support creating standalone components which can be consumed by other Census or 3rd party developers in HTML via a "script" element or import instruction without any runtime dependency on Angular or its toolchain.

Templates

In Angular each component serves the role of controller and viewmodel, and templates serve as the view. Templates are interpolated HTML and use the defacto standard mustache-style syntax (9).

CSS Management

By convention Angular encourages developers use component-scoped CSS, such that styling is tied to each component (10). However, it does support global styling as well. Because templates are a superset of HTML you can simply include a "" element. Angular relies on webpack to manage global styling (11). It does not appear to prescribe a preprocessor but angular-cli does support three: SASS, LESS, and STYLUS.

Performance

The load time performance of an Angular application is highly dependent on its size. Based on my reading, as long as the tools are used correctly a simple Angular 5 application can be reasonably small –on the order of 100KB (12,13). This is on par with React.

Based on my reading good runtime performance of an Angular application is highly dependent on the developer understanding Angular internals and tuning code appropriately (14).

React

React is a JavaScript XML (JSX) based open-source framework for the development of single-page web applications created by Jordan Walke, a software engineer at Facebook.

Development Paradigm, Tooling, and Lifecycle

React-based components and applications are developed in JSX. JSX is basically just JavaScript with embedded HTML fragments. Components override a class "render" method in order to return the component's view (an HTML fragment). React manages a virtual DOM – an in-memory data structure which makes it appear as though the entire page (DOM) is re-rendered whenever anything changes. However, behind the scenes React is doing "diff" and only updating the parts that have actually changed. Unlike Angular, React is not prescriptive. It does not prefer specific routing and state management, and a number of community solutions exist . This has the advantage of choice but the

disadvantage that a different group of developers other than the framework itself maintains the (effectively required) functionality. There is an NPM managed "CLI" application for React developed by a 3rd party (15).

Learning curve

There doesn't seem to be a consensus, and I don't see any good way of objectively evaluating it, but my interpretation of what I read is that React has a shallower learning curve than Angular. React is, at its core, just a JavaScript library. A developer can simply include it, create a class with a render method, and let React take over executing callbacks as necessary. A developer needs to understand events handling, state management, routing, etc. in order to implement an application, of course. However, the basic idea is very much like a traditional framework (with callbacks).

Components

Developers create React components by extending "React.Component" and implementing a render method. An application is built by composition (16). Because this is implemented at runtime, by definition, other Census developers or 3rd party developers would have to also include React via an HTML "script" element or import statement in order to use CEDSCI/EDDE components.

Templates

React does not offer templates in the traditional sense (as Angular and Vue do). In React the UI of a component is always expressed as a render function. This is very powerful but it can lead to increased complexity (e.g. more code to accomplish the same thing).

CSS management

React supports component-scoped CSS and CSS is usually implemented by putting it inside JavaScript (so called CSS-in-JS). If you want to build React components that can be reused by others then CSS-in-JS is mandatory (17). JavaScript/runtime application of CSS is very powerful but comes with increased runtime size and complexity.

For an application, one could reasonably store CSS externally and import it. That would allow the use of 3rd party preprocessors such as SASS (17).

Performance

The load time performance of a React-based application is highly dependent on its size, of course. The production revision of React (minified and gzipped) is 42KB (18).

Though it may require developer optimization, such as the use of immutable data structures, runtime performance is on par with Angular and Vue (19).

Vue

Vue.js (or just Vue) is an open-source JavaScript framework for building user interfaces. Integration into projects that use other JavaScript libraries is relatively easy because it is designed to be incrementally adoptable. Vue was created by Evan You, a former Google employee, based on his experience with AngularJS.

While Vue does not have a corporate entity driving its development it is used by a number of well-known companies (20) including: Netflix, Adobe, Alibaba, Nintendo (21), and GitLab (22).

Development Paradigm

Vue-based applications and components are developed in HTML and JavaScript. Vue components may either be instantiated directly in JavaScript or by making use of custom HTML elements (23). Vue uses a Virtual DOM to manage the view/display state (much like React).

Vue takes a middle-ground approach to prescribing routing and state management solutions. You can use anything you want (even roll your own) but the Vue developers have official routing (24) and state management (25) implementations.

With Vue, the command line tools are optional though recommended (26) to help facilitate using it with webpack and other popular tools. Vue-cli is a JavaScript application installable via NPM or YARN.

Learning curve

Vue has a reputation for being easy to learn (27,28,29,30).

Components

Developers create Vue components in JavaScript by extending "Vue.component()" and passing in an declarative Object which defines the component. An application is built by composition. Vue does not make a distinction between a component and a container (as React does); however, it may be useful to impose one

ourselves (31). Because component definition is implemented at runtime other Census developers or 3rd party developers would have to also include Vue in order to use CEDSCI/EDDE components.

Templates

Vue uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying Vue instance's data (32). All Vue templates are valid HTML. Under the hood, Vue compiles the templates into virtual DOM render functions. Combined with the reactivity system, Vue is able to calculate the minimal number of components to re-render and apply the minimal amount of DOM manipulations when the app state changes; this is very similar to React. In Vue, you can use the template syntax or write render functions using JSX.

CSS Management

Vue doesn't prescribe a specific CSS management strategy, however, it does recommend some (33). Like Angular, it has support for component-scoped CSS in order to localize management when applicable. In addition, it works well with preprocessors.

Performance

The load time performance of a Vue-based application is highly dependent on its size, of course. The production revision of Vue (minified and gzipped) is 31KB (34).

Runtime performance is on par with Angular and React (19). Unlike React, Vue tracks Virtual DOM dependences automatically (35) freeing developers from doing so.

Hello World

As part of the EDDE team's assessment of Angular, React, and Vue, the team built 'hello world' application instances using each framework as a means to better determine their suitability to EDDE.

Process

The 'hello world' applications were built using only the official documentation sections of each frameworks' website. All three frameworks recommended using

some type of command line interface (CLI) that appear to be officially supported and in various states of development.

For Angular, the angular-cli was used and is supported by Ng developers. React documentation recommended using the create-react-app CLI, which is currently officially maintained. For Vue, the vue-cli was recommended in the docs and appears to be supported by Vue developers, although is currently in beta.

Application requirements

All applications were built following the recommended defaults, and included one parent and one nested child component. The applications also include each framework's routing and state management libraries to better reflect the bare minimum requirements necessary for EDDE. All framework specific media or assets were removed for consistency and to improve comparability between applications. Further, production build were generated and can be found in /dist for ng and vue and /build in react. All applications required the use of Node.js and NPM or Yarn.

The applications may all be found in the review's sub-directories. The installation processes and package listings are below.

Angular

ng install docs

```
npm install -g @angular/cli
ng new my-app
cd my-app
npm install @ngrx/core @ngrx/store  --save
ng serve --open
```

ng package.json

One important detail is that Angular 6 was released days after the Angular 5 hello world app was built. Angular is on a 6-month update cycle so given this cadence, rebuilding an angular 6 application for the purposes of this test was not warranted.

React

reactjs install docs

```
npm install -g create-react-app
create-react-app my-app
cd my-app
yarn add react-redux react-router-dom redux
npm start
```

reactjs package.json

Vue

vue docs

```
npm install -g @vue/cli
vue create my-project
npm install vue-router --save
npm install vuex --save
```

vue package.json

Results

The results of the tests show that the level of effort required in building all three applications were straightforward and well documented, likely a testament to the frameworks' respective CLIs, level of maturity, ecosystem, and community engagement. Deviating modestly from the initial 'hello world' guidance to include routing and state management also proved to be a relatively consistent experience as well.

The general metrics of the frameworks were - likewise - comparable. In terms of the initial and repeated payload sizes, all frameworks performed well once all builds were production-ready.

Framework	No Cache	Reload w/o cache	reload w/ cache
react	365 KB	3.3 KB	0 KB
vue	106 KB	106 KB	2.6 KB
ng	314 KB	314 KB	799 B

All three were well documented in terms of standing up minimal apps. However and incidental reading of each showed that Vue's may have been slightly more robust as it allowed the user to quickly toggle between versions. It should be noted that this distinction is relatively minor and only a contributing factor.

The most immediate differences between Angular and the other two frameworks is: (1) the use of Typescript; and (2) its prescriptive nature.

On the first point, Typescript excels in catching issues early, and in the context of a larger development team, could yield relatively substantial time savings in tracking down annoying bugs. That said, there is the implicit cost of using a javascript superset rather than vanilla javascript, and in today's market, not every front-end developer currently working with the Census Bureau have project experience with it. Therefore, there would be an initial cost in ensuring that all team members were adequately trained before committing code. That said, due

to the construction of components in Angular, it is possible to segment effort to html, css, and typescript which allows teams with varying level of comfort with frontend development to be immediately productive.

On the second point, Angular is prescriptive and consequently less flexible - some would say inflexible. The number of tooling decisions needed is greatly reduced, which *can* free up time for development. However because the decisions have been removed by the architecture and design team means that certain compromises must be accepted if and when the prescribed solution is not optimized to the use case.

Unlike Angular, both Vue and React are based on the virtual DOM model and could be more aptly be described as libraries than actual frameworks. Consequently they are considerably lighter weight and take far less to instantiate bare bone applications.

The most noticeable distinction between React and Vue is the use of JSX and the use of templates. Vue structures components by binding attributes to HTML while React launches the DOM in js, or JSX to be more specific. Although JSX is - at its core - a combination of HTML and js, it does have modest learning curve and combines assets into a single file which can incumber team members who need to modify html and js independently.

In short, all three were highly capable, easy to stand up and modify to the required minimal use case, and performant. The more notable differences between the frameworks/libraries are their flexibility and learning curve. From the hello world effort, it was clear that Vue remained closest to vanilla js, html, and css and as a result, has the lowest learning curve out of the three. Angular, due to its prescriptive nature and use of typescript, is likely the hardest to pick up. React is moderately more challenging than Vue.

Framework	Learning curve
Angular	High
React	Medium
Vue	Low

From both the landscape analysis and the hello world effort, it is clear that an initial decision between an angular or non-angular approach should be made.

Decision

The team met over a series of meetings to discuss the results of the landscape analysis and the hello world effort. After careful review of the findings, the team then came to a unanimous recommendation.

The team's concluded that any of these frameworks *could* be used to implement the CEDSCI uses cases: CEDSCI/EDDE UI, Microdata Access Tool, and VisART. However in narrowing down the options, the first decision point made was Angular vs non-Angular, with the primary question being: does the benefits of adopting the Angular ecosystem outweigh the costs/risks. Creating a maintainable, performant Angular application or component requires not just a JavaScript expert but an Angular expert and EDDE would be forced to hire/contract with developers who know Typescript, Angular's tools, and idioms.

Furthermore, because of angular's relative inflexibilty and EDDE's broadening scope, it is critical that a premium be placed on flexibilty. The EDDE team does not believe that the productivity boost in limiting options is an acceptable trade-off to the potential compromises the team would be forced to accept.

After the team agreed to focus on the non-Angular options, the points of distinction came to the rapidity of team adoption and the practical implications of adopting each of these frameworks. In the end, the deciding factor between Vue and React came down to the learning curve and the potential capacity for the typical Census developer to quickly contribute to a performant and capable application. For that reason, the team unanimously agreed to recommend Vue as the primary solution, React as a close second alternative, and Angular as an acceptable third.

Expected Level of Effort

References

1. Various. List of JavaScript libraries. s.l. : https://en.wikipedia.org/wiki/List_of_JavaScript_libraries#Web-application_related_(MVC,_MVVM). 2. Shilova, Margarita. JavaScript frameworks you should know. s.l. : https://apiumhub.com/tech-blog-barcelona/top-javascript-frameworks/. 3. Various. Comparison of JavaScript frameworks. s.l. : https://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks. 4. Hannah, John. The Ultimate Guide to JavaScript Frameworks. s.l. : https://javascriptreport.com/the-ultimate-guide-to-javascript-frameworks/. 5. Medium. Angular 2 vs React: The Ultimate Dance Off. s.l. : https://medium.com/javascript-scene/angular-2-vs-react-the-ultimate-dance-off-60e7dfbc379c, 2016. 6. Cabot Technology. A Comparative Study of Progressive JS Frameworks: Angular.js & Vue.js. s.l. : https://hackernoon.com/a-comparative-study-of-progressive-js-frameworks-angular-js-vue-js-4ebda3904f8f, 2017. 7. Reddit. Steep learning curve for Angular 2. s.l. : https://www.reddit.com/r/Angular2/comments/5hlw5p/steep_learning_curve_for_angular_2/. 8. Alexsoft. The Good and the Bad of Angular Development. s.l. : https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/. 9. Angular.io. Angular Template Syntax. s.l. : https://angular.io/guide/template-syntax. 10. Blog, Angular.io.

The State of CSS in Angular. s.l. : https://blog.angular.io/the-state-of-css-in-angular-4a52d4bd2700. 11. Angular In Depth. This is how angular-cli/webpack delivers your CSS styles to the client. s.l. : https://blog.angularindepth.com/this-is-how-angular-cli-webpack-delivers-your-css-styles-to-the-client-d4adf15c4975, 2018. 12. VueJS.org. Comparison with Other Frameworks. s.l. : https://vuejs.org/v2/guide/comparison.html. 13. Reddit. My experience today re: Angular vs. Vue bundle size. s.l. : https://www.reddit.com/r/Angular2/comments/7e2upr/my_experience_today_re_angular_vs_vue_bundle_s 2017. 14. Spears, Paul. Angular Runtime Performance Guide. s.l. : https://blog.oasisdigital.com/2017/angular-runtime-performance-guide/, 2017. 15. Abramov, Dan. Create Apps with No Configuration. s.l. : https://reactjs.org/blog/2016/07/22/create-apps-with-no-configuration.html, 2016. 16. ReactJS.org. Composition vs Inheritance. s.l. : https://reactjs.org/docs/composition-vs-inheritance.html. 17. Menubar.io. 5 Ways To Style React Components. s.l. : https://www.menubar.io/5-ways-to-style-react-components, 2108. 18. Restuta. Sizes of JS frameworks, just minified + minified and gzipped. s.l. : https://gist.github.com/Restuta/cda69e50a853aa64912d, 2017. 19. Stefankrause. Results for js web frameworks benchmark – round 7. s.l. : http://www.stefankrause.net/js-frameworks-benchmark7/table.html, 2018. 20. Netguru.co. 13 Top Companies That Have Trusted Vue.js – Examples of Applications. s.l. : https://www.netguru.co/blog/13-top-companies-that-have-trusted-vue.js-examples-of-applications, 2018. 21. Sloboda Studio. Should You Choose VueJS Over React? s.l. : https://sloboda-studio.com/blog/should-you-choose-vuejs-over-react/, 2018. 22. GitLab. Why We Chose Vue.js. s.l. : https://about.gitlab.com/2016/10/20/why-we-chose-vue/, 2016. 23. Vue. Components Basics. s.l. : https://vuejs.org/v2/guide/components.html, 2018. 24. —. Routing. s.l. : https://vuejs.org/v2/guide/routing.html, 2018. 25. —. State Management. s.l. : https://vuejs.org/v2/guide/state-management.html, 2018. 26. —. Announcing vue-cli. s.l. : https://vuejs.org/2015/12/28/vue-cli/, 2015. 27. Reddit. Between React and Vue.js, which is easier to learn? s.l. : https://www.reddit.com/r/javascript/comments/45gwer/between_react_and_vuejs_which_is_easier_to_learn 2016. 28. —. Why is Vue so easy? s.l. : https://www.reddit.com/r/vuejs/comments/774dn7/why_is_vue_so_ 2017. 29. Medium. Top Tutorials To Learn Vue Js For Beginners. s.l. : https://medium.com/quick-code/top-tutorials-to-learn-vue-js-for-beginners-6c693e41091d, 2018. 30. Gore, Anthony. Vue.js is easier to learn than jQuery. s.l. : https://medium.com/js-dojo/vue-js-is-easier-to-learn-than-jquery-abbbb9c12cf8, 2016. 31. Balk, Alex. Structure a Vue.js App from Containers and Components. s.l. : https://www.outbrain.com/techblog/2017/12/structure-a-vue-js-app-from-containers-and-components/, 2017. 32. Various. Vue.js. s.l. : https://en.wikipedia.org/wiki/Vue.js, 2018. 33. Vue. CSS Management. s.l. : https://ssr.vuejs.org/en/css.html, 2018. 34. —. Installation. s.l. : https://vuejs.org/v2/guide/installation.html, 2018. 35. Kemp, Jonathan. Get More Done with Vue.js: Vue and Performance. s.l. : https://jonkemp.com/javascript/get-done-vue-js-vue-performance.html, 2017. 36. State Of JS. Front-end Frameworks – Results. s.l. :

https://stateofjs.com/2017/front-end/results, 2017.