Front-end Frameworks comparison

A dicussion of JS frameworks for non-expert colleagues

In my role as Web Architect, I was asked to compare JavaScript front-end frameworks. I wanted to help my team select one framework that for new projects, in different teams, environments and customer budget ranges.

The criteria I chose was not necessarily only technical, but meant to help our company adapt to a new market of technologies, which had stalled in the recent months due to a case of organizational “analysis paralysis”.

I wanted to select a single framework, or at least provide a way to decide, so we could share knowledge and experts, collaborate more, and create more “synergy” by using the same base technology.

I presented my findings to colleagues and management during a workshop. Below are my ideas:

## The competitors

There are three main competitors for the position of “popular” JS frameworks. While there is not a single “winner”, we should select one that can help us achieve our business goals in most cases.

Three are obvious JS competitors: Angular, React, VueJS.

### Angular

Angular, currently version 5, is the “new” Angular framework, with versions 2+ (it needs to be distinguished from the “old” AngularJS framework with versions 1.x). After some attemps and experimentation on my part, I found an [upgrade](https://angular.io/guide/upgrade) that is [not trivial](https://angular.io/api/core/NgZone) to move from AngularJS 1.x to Angular 2+, which meant many rewrites would be required to move older projects.

Angular receives most of its support and financing from Google, and Google teams make most of the steering of the project.

The framework is designed to be a “batteries included” solution, Angular makes many choices for the developer and comes with many of its functionalities out-of-the-box. This can help make different projects more standardized.

Anecdotally, seems to me it has less buzz in the media than other JS frameworks and tools, its “community” is mostly composed of internal users which are sponsored to support the framework.

Google, apparently, didn’t eat its own dog food. Their main products still use other libraries (e.g. Gmail and Google Analytics use Closure).

### React

When using the metaphor of “the cathedral and the bazaar”, clearly React is a bustling bazaar of libraries, plugins and ideas.

The framework creators have not pushed a deliberate way to build a “React” app, which helps keep development flexible. The downside is that this may introduce some fatigue, developers must spend a lot of energy choosing, changing and adapting when starting a new project, the flexibility comes at a cost.

Facebook, the original creator of React, switched from the confusing “BSD + Patents” license to the MIT license to the relief of corporate users, which were concerned that the patents of some of their products might be lost to Facebook be introducing React to their stack.

Large applications built in React might suffer from tight coupling, so solutions such as Redux and functional programming are used to solve them. This comes at the cost of using higher level of abstractions when building functionality that could be more simple.

### VueJS

Vue 2.0 was released in September 2016, which is the stable development version.

Unlike the two previous competitors, a large corporation does not fund it, and it is partly crowdfunded. The community offers grassroots donations support to it.

It is fast growing as an alternative tool globally, however, since its creator, Evan You, is from China, it is also popular and relevant inside the country, which can be an advantage when finding development talent there.

Additionally as an example of partnerships of VueJS, Laravel, one of the popular new PHP frameworks bundles VueJS by default. Out of the box, they use of VueJS to do part of the presentational logic.

VueJS does not offer a complete landscape solution for mobile apps. Unlike React and Angular, a fractioned mobile offer might represent more difficulty when creating mobile apps (there is no obvious equivalent of VueJS Native).

As an advantage, the framework can be used to “progressively” rewrite applications built in older generation frameworks. It can be used to refactor older “jQuery mess” type of web interfaces.

**I also want to mention 2 technologies that may help our development efforts in the near future.**

### Honorable Mention: GraphQL

In [GraphQL](http://graphql.org/) the API definition is not based on REST, it instead, defines APIs via “Queries” and “Schemas” of objects travelling through the network which are composed of several basic types and its compositions: (Object, Int, Boolean, …)

Once an “Schema” is agreed upon it can be used to generate programmatically client and server code.

It uses two main kinds of query, which are use to fulfill get role of the HTTP verbs called: via ExecuteQuery (replacing GET) and ExecuteMutation (replacing POST, PUT, DELETE).

GraphQL helps in solving endless discussions around how to map relationships between objects, which in REST is never obvious choice (see [n+1 problem](https://restfulapi.net/rest-api-n-1-problem/)). This helps building more straightforward front-end code that decides which entity and fields to show and in relation to which other entities. As applications grow more complex, the number relationships between data will grow increasingly making the API difficult to use.

GraphQL also allows for a gradual “evolution” of APIs, via deprecations, which is a weak point of most REST+JSON setups.

*Why mention GraphQL here?* Deciding to use GraphQL has an impact in the way we design our front-end (and back-end). It challenges some assumptions about how “state” gets maintained in the front-end applications.

### Honorable Mention: Storybook

[Storybook](https://github.com/storybooks/storybook) is a way to do separate component testing and development outside of the application.

It can be a powerful tool when separating concerns of “components” and “modules” of our front-end end application.

Storybook allows testing and using components independently from each other and from the larger application. Any business user or tester is able to check the look and feel of the app pieces before it’s integrated and built.

*Why mention it here?* Storybook is compatible with the major frameworks discussed here. Regardless of which framework is chosen, making stateless means making them reusable. This has an impact on how teams might approach building their UIs and how projects are organized, specially on complex front-end projects.

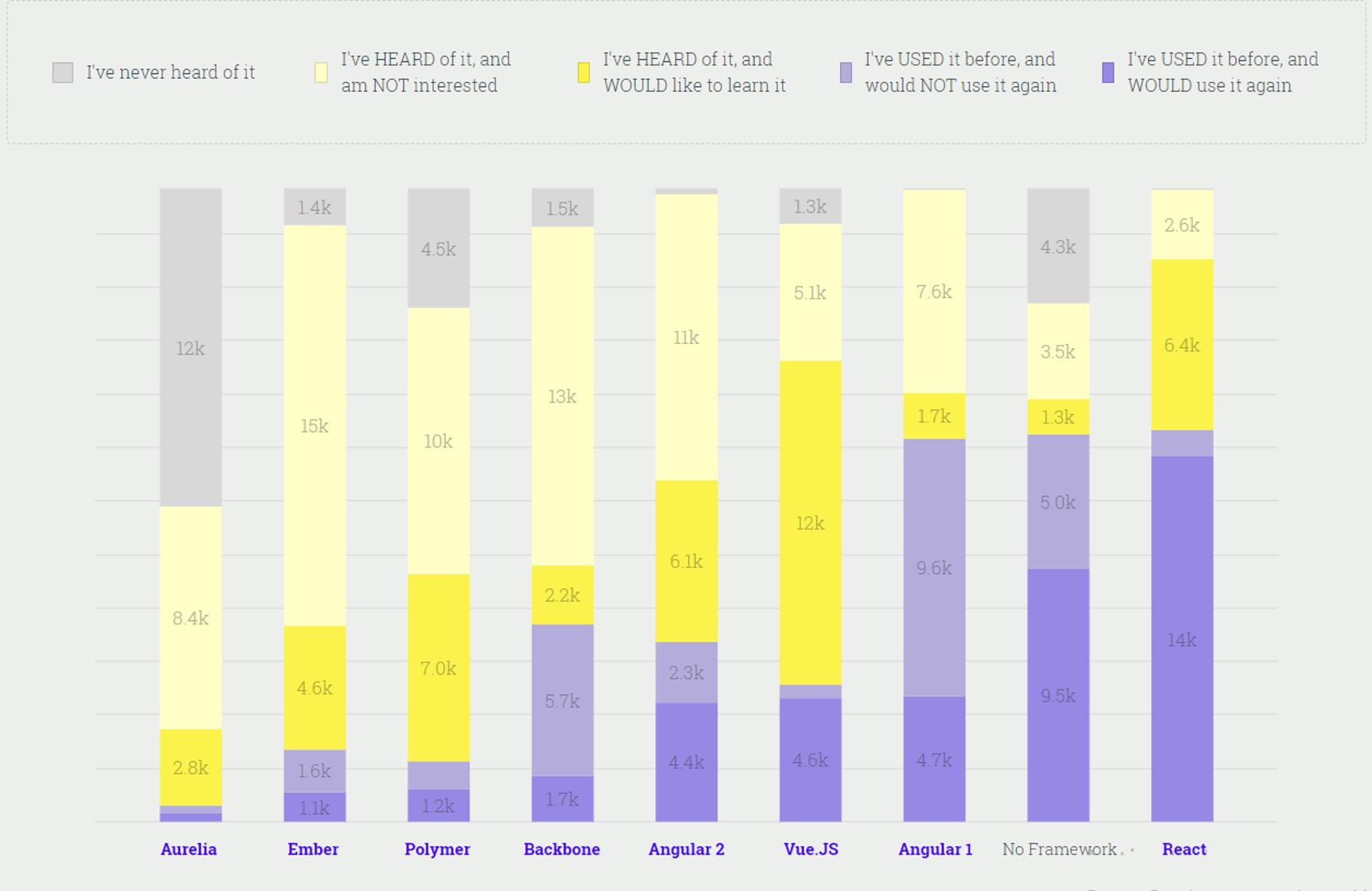
## JavaScript in 2017

The 2017 state of JS gave me insights into the market and usage trends for the frameworks we are discussing.

While I do not normally use the number of stars and questions on StackOverflow as an indicator of which tools I should use it does have major impact in the corporate world: **hiring**.

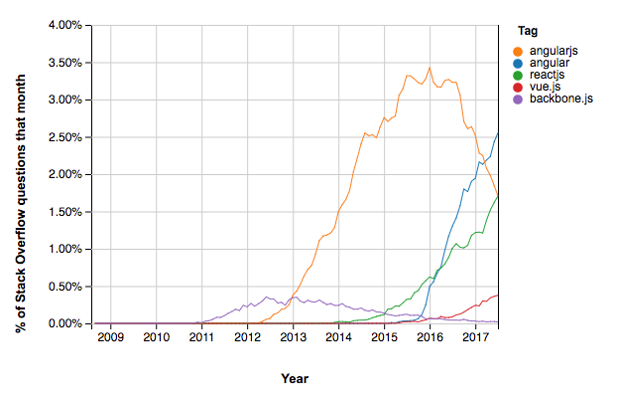
When building teams of developers in a fast paced environment, such as China (with a high turnover rate and short deadlines), skills of each individual contributor should align to the needs of the project at hand as fast as possible, and popular frameworks allow shorter training periods. This means, more popular frameworks can have a bigger pool of candidates that can work on them.

Below a figure of the usage and intention of usage of different frameworks globally:



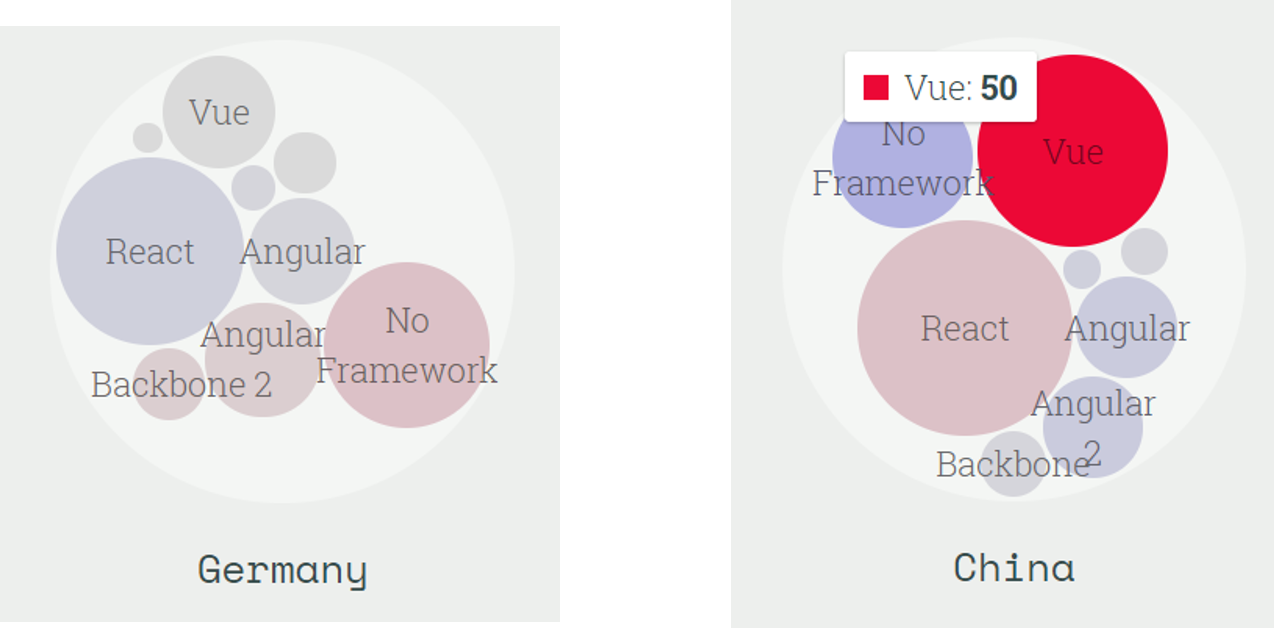
JS Framework Popularity contest

VueJS and Angular 2+ are interesting for developers’s new projects, and React is established as the most-used framework. I see a similar trend in StackOverflow:



JS Framework Popularity contest

I wanted to analyze China and Germany markets, and the differences are interesting:



JS Framework Popularity contest

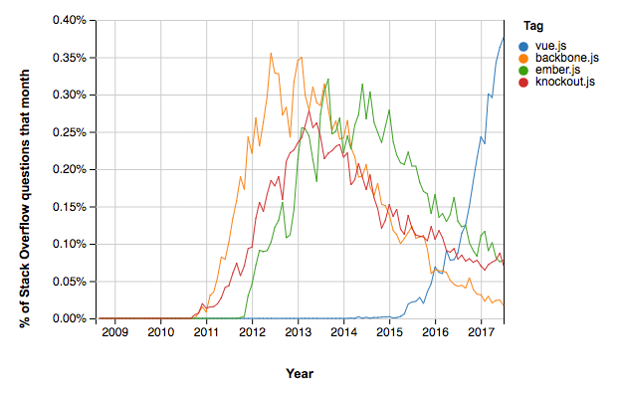
While Germany has a similar result as other global trends, China, the home of VueJS, has a preference for it (note: the data for China is smaller could be more biased).

This means that when building new front-end applications with teams in China, VueJS might have some advantages.

## Suggestions based on JS history

Historically, popular frameworks have had a 2-year life cycle to later get “abandoned” or stuck as framework “zombies”. We cannot easily predict the popularity of a given framework.

I suggest planning for a 2-4 years application life cycle, this is the history lesson of older JavaScript frameworks.



JS Framework Popularity in the past

So this leads me to some conclusions:

* Don’t build UI monoliths: different business scenarios and roles might call for different UI approaches.
  + For example, a back-office can be built using classical MVC templates using VueJS, while the client facing application is built using SPA with Angular.
* Different teams will require different tooling and life cycle. Depending on the expected lifetime of a project, you may choose a more established framework over a new one.
* Select a good tool for your job market, China and Europe have different talent markets: hire accordingly.

**Note:** At the end of discussion, we opted for a general guideline of using VueJS for most new projects. VueJS provided an attractive choice for new hires, buy-in of the China market and overall “love” of the global community.