

Mission of the New Technology Infrastructure

A fresh look into the existing tech stack.

Re-evaluate technical choices for web app development.

Goals

- Productivity
- Maturity
- Community

Imagine yourself as a **new hire**

How **easy** is it to adopt the tech stack?

How **fast** is it to develop a new feature?

How **sustainable** is it to maintain the tech stack over time?

How **flexiable** is it to customize the tech stack?

**The core of existing tech stack --
Angular**

Angular -- a framework

... and a heavily opinionated one.

Angular:

RxJS and Observables

Angular: RxJS

Angular HTTP client returns Observables.

RxJS is an implementation of Observables.

Observables are suitable for stream-like data.

... but making it framework core is not the best approach

Angular: RxJS

Require a paradigm shift to RFP (Reactive Functional Programming).

Over-complicate for most REST API calls.

RxJS' **104** operators vs. Promise's **7** methods

RxJs API: Operators (total 104 APIs)

<div>F</div> audit	<div>F</div> auditTime	<div>F</div> buffer
<div>F</div> bufferCount	<div>F</div> bufferTime	<div>F</div> bufferToggle
<div>F</div> bufferWhen	<div>F</div> catchError	<div>F</div> combineAll
<div>F</div> combineLatest	<div>F</div> concat	<div>F</div> concatAll
<div>F</div> concatMap	<div>F</div> concatMapTo	<div>F</div> count
<div>F</div> debounce	<div>F</div> debounceTime	<div>F</div> defaultIfEmpty
<div>F</div> delay	<div>F</div> delayWhen	<div>F</div> dematerialize
<div>F</div> distinct	<div>F</div> distinctUntilChanged	<div>F</div> distinctUntilKeyChanged
<div>F</div> elementAt	<div>F</div> endWith	<div>F</div> every
<div>F</div> exhaust	<div>F</div> exhaustMap	<div>F</div> expand
<div>F</div> filter	<div>F</div> finalize	<div>F</div> find
<div>F</div> findIndex	<div>F</div> first	<div>F</div> flatMap
<div>F</div> groupBy	<div>F</div> ignoreElements	<div>F</div> isEmpty
<div>F</div> last	<div>F</div> map	<div>F</div> mapTo
<div>F</div> materialize	<div>F</div> max	<div>F</div> merge
<div>F</div> mergeAll	<div>F</div> mergeMap	<div>F</div> mergeMapTo
<div>F</div> mergeScan	<div>F</div> min	<div>F</div> multicast
<div>F</div> observeOn	<div>F</div> onErrorResumeNext	<div>F</div> pairwise
<div>F</div> partition	<div>F</div> pluck	<div>F</div> publish
<div>F</div> publishBehavior	<div>F</div> publishLast	<div>F</div> publishReplay
<div>F</div> race	<div>F</div> reduce	<div>F</div> refCount
<div>F</div> repeat	<div>F</div> repeatWhen	<div>F</div> retry

Promise's 7 methods

`Promise.all()`

`Promise.prototype.catch()`

`Promise.prototype.finally()`

`Promise.prototype.then()`

`Promise.race()`

`Promise.reject()`

`Promise.resolve()`

... A lot of people I know have a hard time understanding Observables, Subjects, etc. All of that on top of general Angular complexity seem to be **very harsh on framework newcomers.**

We literally rejected interview candidates because they don't know Observable APIs. But RxJS is not as widely accepted in the JavaScript community as we thought.

Angular: RxJS

To learn more:

[Why isn't RxJS more popular?](#)

[RxJS is great. So why have I moved on?](#)

Back to our Goals

Again, imagine yourself as a new hire.

Productivity: Embrace simple and effective solutions.

Community: Use something widely supported in the community.

Maturity: Favor something battle-proven over a long period of time.

How can we get there?

Replace Angular's HTTP client with Promise based client (e.g., Axios).

Use RxJS observables only where they are appropriate.

Angular:

HTML Template

Angular template has its own DSL syntax:

```
<p>{{ name | slice:1:5 }}</p>
```

At first glance, you might think the DSL syntax is OK. But it could get unwieldy quickly.

Special DSL just for a loop

```
<li *ngFor="let item of items; let i = index; trackBy: trackByFn">  
  {{item.id}}  
</li>
```

Special DSL for if-else

```
<div *ngIf="someCondition; else falsyTemplate">
  <h1>Condition Passed!</h1>
</div>

<ng-template #falsyTemplate>
  <h1>Condition Failed!</h1>
</ng-template>
```

Angular 4 even touts “else” with *ngIf as a new feature.

How to use *ngIf else in Angular?



325



68

I'm using Angular and I want to use `*ngIf else` (available since version 4) in this example:

```
<div *ngIf="isValid">
  content here ...
</div>
```

```
<div *ngIf="!isValid">
  other content here...
</div>
```

asked 1 year, 5 months ago

viewed 360,146 times

active 17 days ago

How can I achieve the same behavior with `ngIf else` ?

angular

if-statement

angular-template

share improve this question

edited Dec 16 '17 at 21:39



Lazar Ljubenović

6,611 ● 19 ● 45

When developer's fluency in basic **if-else** statements is in question, what do you think how productive they will code?

Angular: HTML Template

Angular is template driven.

Templates all invent their own DSLs which are not standard anywhere else.

...and they are often counter-productive to work with.

How do we improve?

No need for HTML DSL to mimic JavaScript code.

Re-use what JavaScript developers already know -- JavaScript.

For example, instead of writing this:

```
<div *ngIf="someCondition; else falsyTemplate">
  <h1>Condition Passed!</h1>
</div>

<ng-template #falsyTemplate>
  <h1>Condition Failed!</h1>
</ng-template>
```

switch to JavaScript ternary operator

```
<div>
  {someCondition ? <h1>Condition Passed!</h1> : <h1>Condition Failed!</h1>}
</div>
```

Angular	Plain JavaScript
*ngForOf	Array.map
*ngIf	if-else statement, or ternary operator
filter pipe	plain function

The root problem is -- Angular tries to use HTML syntax to mimic JavaScript code. But the invented DSL is often verbose and cumbersome to work with.

All these smalls could add up to the potential loss of daily productivity.

Recap: how do we improve?

Once more, imagine yourself as a new hire.

Maturity/Productivity: Use standard JavaScript code instead of non-standard DSLs. Let JavaScript deal with logics and code path. Do what JavaScript is best at.

Angular:

Tool Chain

Angular: Ng CLI

Ng CLI is Angular's recommended building tools.

Widely adopted for its out-of-box functionalities.

... however, it is **over-abstracted** and **inflexible**.

Angular CLI and Webpack

Webpack is the centerpiece of the building process.

It governs code splitting, minimization, optimization, post processing, transformation, bundling, etc.

It is very important for developers to have **full control of webpack configuration**.

However, in Angular webpack configuration is generated and encapsulated.

...where in React and Vue, developers have **direct access** to webpack configuration.



larssn commented on May 22 • edited ▼



Intro

We dearly need a lightweight way to customize the internal webpack config file, so advanced users can tweak it, and still use the CLI, without having to eject. The current approach of having to eject causes a huge amount of overhead (also for the CLI team it seems). In NG5, we personally had 4 different fully complete webpack config files, to satisfy our dev environment, which was getting out of hand.

Since you can't eject in NG6, you currently have people rewriting `node_modules` files, to do some simple customizations. See <https://gist.github.com/niespodd/1fa82da6f8c901d1c33d2fcbb762947d>

Angular's `ng eject` command can be used to "eject" webpack configuration.

... but even this hacky feature is **disabled in Angular 6**.

Angular 6 and Webpack

Angular 6 will introduce *a new custom builder system* which is an abstraction layer on top of many build tools including webpack.

However, this system is **not ready**. And there is no announcement of when it will be ready.



CarissaThomas commented on Jun 6



Is there a way to manually edit a webpack.config file in Angular 6? If ng eject no longer works, how can I access the Webpack.config file? I'm currently having issues with URLs and dots on the Webpack dev server and I need to access the webpack.config so that I can properly implement the disableDotRule. Honestly, if Angular 6 was released without these kinds of issues in consideration, I question the decision making on this one.



sarunint commented on Jun 7

Contributor



@CarissaThomas Unfortunately, at this stage it is impossible. But it will be possible soon, since version 6 of the CLI supports custom builders. But its API is not stable for custom implementation yet. So, soon...

Undocumented breaking change

Yet, the `ng eject` is already disabled, even before the new build system is released.

And this critical breaking change is not even documented.

<> Code

🔔 Issues 1,421

🔗 Pull requests 58

📖 Wiki

📊 Insights

ng eject not available at Angular 6.0.0 #10618

🔔 Open


andygup opened this issue on May 3 · 21 comments



andygup commented on May 3 • edited ▼



This is currently an undocumented breaking change for those migrating from 5 to 6, ng eject is temporarily disabled at v6.0.0.



**There is no
migration path**

When a breaking change was introduced in React, React owners created a tool to **automatically migrate** the affected source code.

<https://github.com/reactjs/react-codemod>

Note: this tool comes from React's official GitHub.

Angular 6 new build system

Even when the new builder system is ready in an unknown future, I doubt why we need yet another build system to abstract most already well known build tools, including Webpack.

Most of the build tools' documentation are **more complete** than Angular CLI's.

Most of the build tools have **faster innovation cycles** than Angular CLI's.

This over abstraction is **unnecessary** and even **dangerous**.



biiiipy commented on Jun 19



Yes, at this point Angular CLI is dangerously bad to rely on. If at some point some dependency breaks or you need some customization that angular CLI doesn't support, then you are **screwed**. And CLI should help you do hard, repetitive, standardized work, not lock you in. That's why I don't recommend **anybody**, apart from quick demos, to use CLI in their projects.

Also, "Eject" feature is kind of nice, but I think it shows the fundamental flaw with Angular CLI - it's either CLI or plain webpack. It doesn't enhance existing webpack pipeline, but tries to replace it with it's own high level construct, and that creates a lot of friction, lock-in, unnecessary compatibility problems etc, and in the end defeats it's own purpose.



19



12

A UI framework should not try to dictate the building process and tools.

The tail should not wag the dog.

YUI, jQuery, and even the infamous Struts didn't dictate how we build web assets.

In a sense, Angular is worse than AngularJS when it comes to framework lock-in.

Our business requires high-level of customization and scalability.

Having full control of our building process is essential to create highly customizable and scalable web apps.

How can we improve?

Back to the principle:

Maturity: No over abstraction.

Productivity: Control the build tools directly.

Community: Rely on more complete documentation of individual build tools.

Which means

Use *Webpack* directly for bundling and optimization.

Use *Lerna* to manage mono repo.

Use *npm* or *yarn* in place of Angular's *ng cli*.

... etc.

Page Frame

An umbrella component that centralize common UI aspects such as:

Page header, footer, error pages, debug footer, confirm ID, loading spinner, nav bar, etc.

Often heard complaints

- Performance
- Feels too heavy and app doesn't have enough control

Proposal

Re-evaluate dumb vs. smart components.

Clear separation of data and presentation.

Expose services to apps, and give apps more control.

Other areas

fe-business-lib - Make the service lib UI agnostic

GraphQL

Text resolution

Settings and configs

Tech stack exploration

The methodology

From a perspective of an outsider, use an alternative UI library or framework as a probe or **forcing function** to "question" the existing tech stack, and provide a better solution.

The golden question

Without the existing tech stack, starting from a clean slate, what kind of new tech stack shall we create for better web app development as of today?

What the alternative UI library will be?



It is not about Angular vs. React.

It is about Angular vs. Something better than Angular and its ecosystem
...in terms of Productivity, Maturity, and Community.

Recap on Productivity

A dev team's productivity and the quality of its work are largely **determined by its junior engineers.**

Having a tech stack that has **lower entry barriers** and less chance for errors will definitely boost the team's productivity and quality.

React and Productivity

Low learning curve - the core takes 2 days to master.

Much less core concept - No DI, decorators, pipes, injectors, module declarations, nor special DSL.

Unopinionated - No deep coupling with high-learning-curve libraries at its core.

Usually take less code to achieve the same output.

Newcomer friendly.

Recap on Maturity

A framework or library can only be as solid as its foundation.

React and Maturity

No experimental features, e.g. decorators.

No non-standard module system, nor awkward DSLs.

Mature tool chains.

Smooth version upgrade with migration tools and guidance.

No radical breaking changes without migration path.

Misc: React and Quality

Test coverage for HTML and even CSS styles.

```
1  FontWeight > should set correct font-weight
2
3  expect(value).toMatchSnapshot()
4
5  Received value does not match stored snapshot 1.
6
7  - Snapshot
8  + Received
9
10 @@ -1,7 +1,7 @@
11     .co {
12 -   font-weight: 300;
13 +   font-weight: 200;
14     }
15
16 <div
17   className="c0"
18 >
```

HTML, CSS, and JavaScript -- all 3 main building blocks of web apps can now be validated in React.

Community

Developer preference matters

In GitHub, React has **109,545** stars and 6,244 watches while Angular has **39,862** and 3,174 watches respectively.

React Router has **32,176** stars, which is not included in the React count listed above.

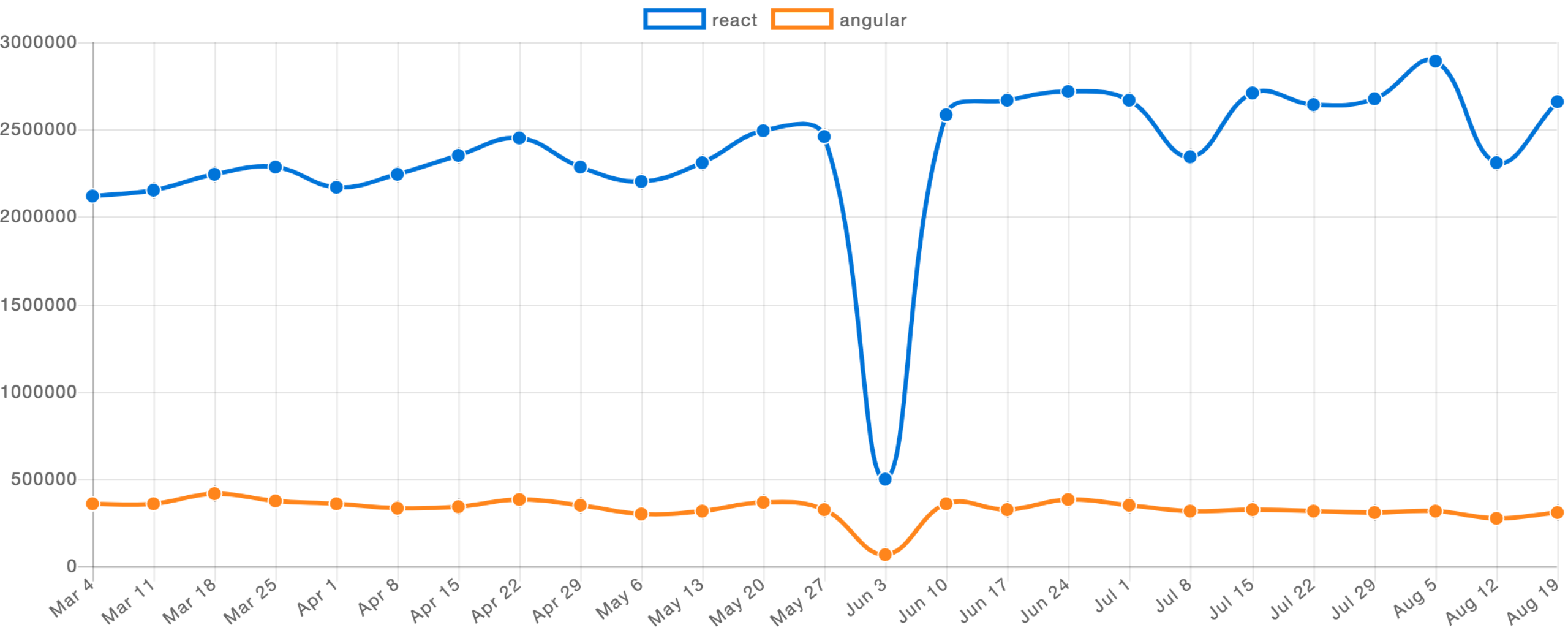
State of JavaScript 2017



- I've never heard of it
- I've HEARD of it and am **Not** interested
- I've HEARD of it and **WOULD** like to learn it
- I've USED it before, and would **NOT** use it again
- I've USED it before, and **WOULD** use it again

NPM downloads

Number of npm loads in the past 6 months



Market demand

A 2018 survey of more than 60,000 worldwide open positions that require a specific knowledge of a certain framework.

(source: [indeed.com](https://www.indeed.com))

Vue.js

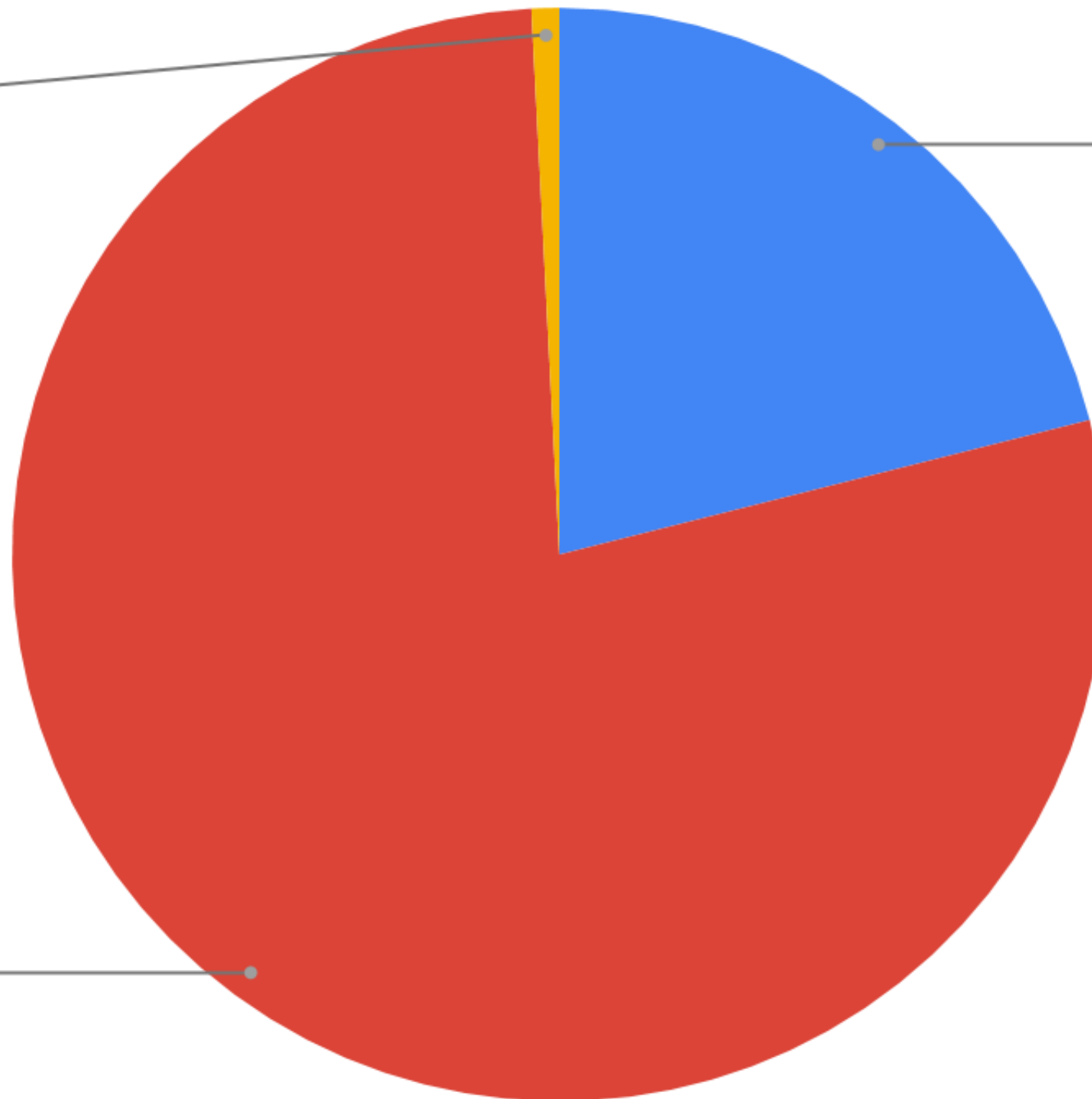
0.8%

Angular

21.0%

React

78.1%



What do all these charts mean to us?

Larger talent pool for hiring and mind sharing.

More active community support.

But the thing that really matters:

More sources to ctrl+c/ctrl+v code from Stackoverflow ;)



**By the way,
this entire slide deck is created in
React**