Oct 16, 2019

Micro Frontends with Angular and single-spa

Matt Gold
Tech Lead at Kabbage

Kabbage

@mattbgold

ng update

ATL

DELTA

The Delta Hangar
Thursday, November 21
6 PM - 8:30 PM

Registration Link to come in Meetup Group

Kabbage

Oct 16, 2019

Micro Frontends with Angular and single-spa

+

Matt Gold
Tech Lead at Kabbage

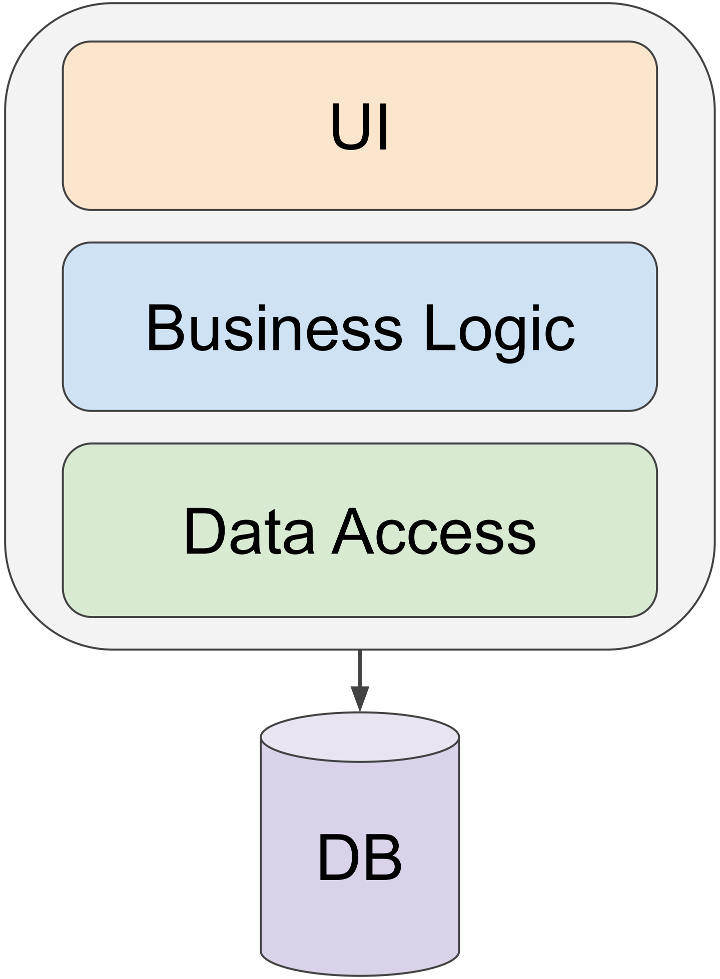
Kabbage

@mattbgold

How to architect a web application?

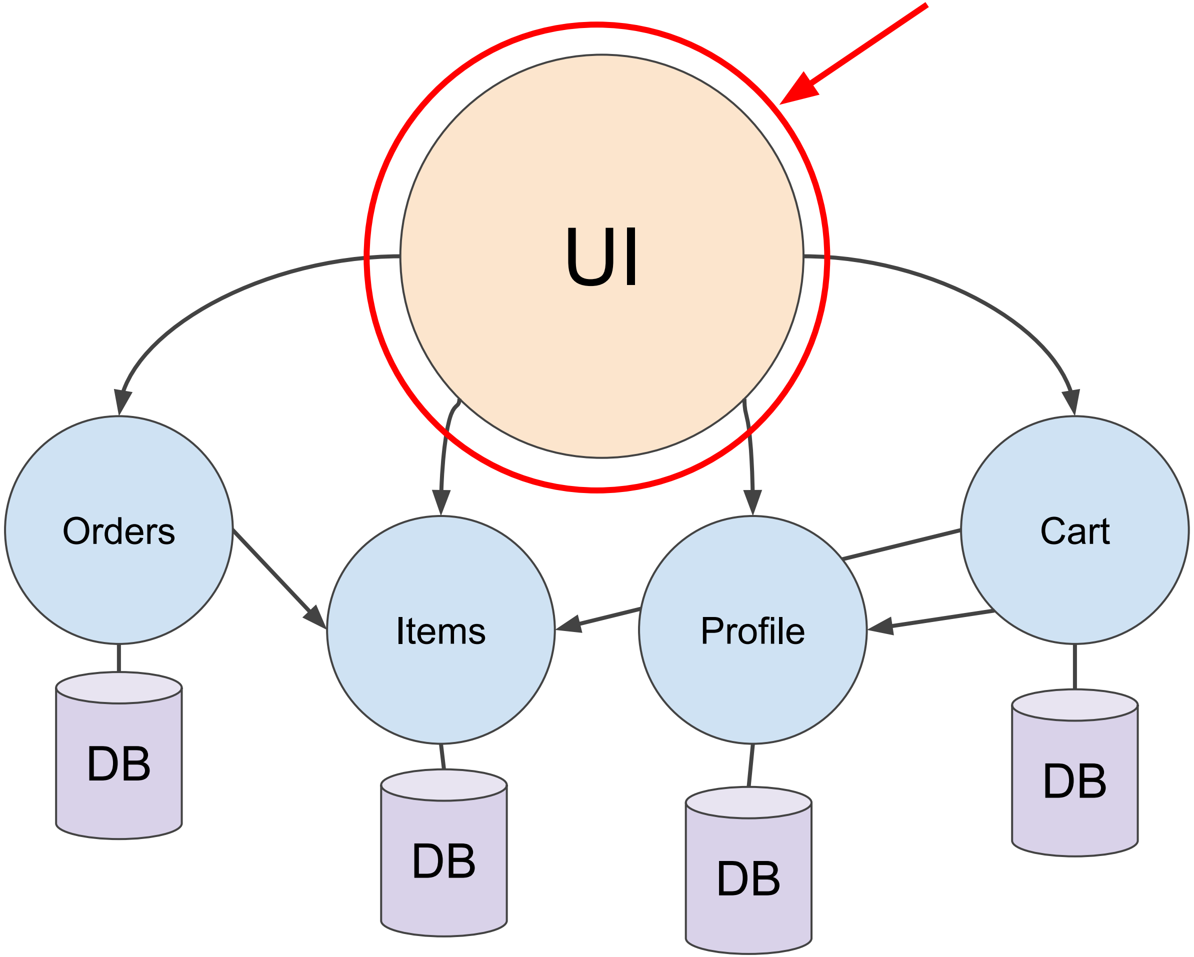
Monolith

- Tightly coupled
- Difficult to scale properly
- Reduced agility
- Difficult to maintain
- Locked into technology



Microservices

- Easier to maintain
- Scale Independently
- Release Fast & Often
- Team autonomy



UI

Wait, this sounds familiar...

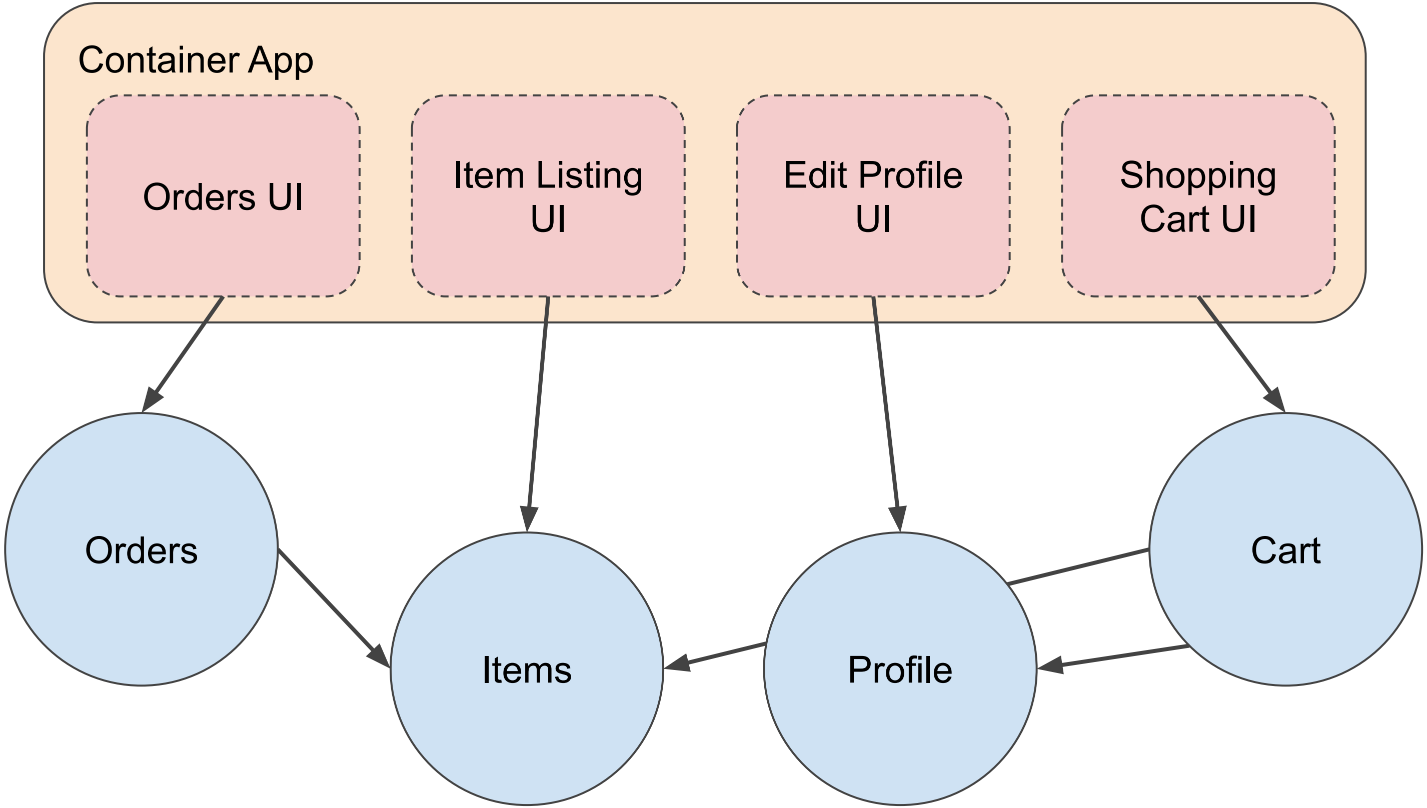
Haven't we solved these problems already?

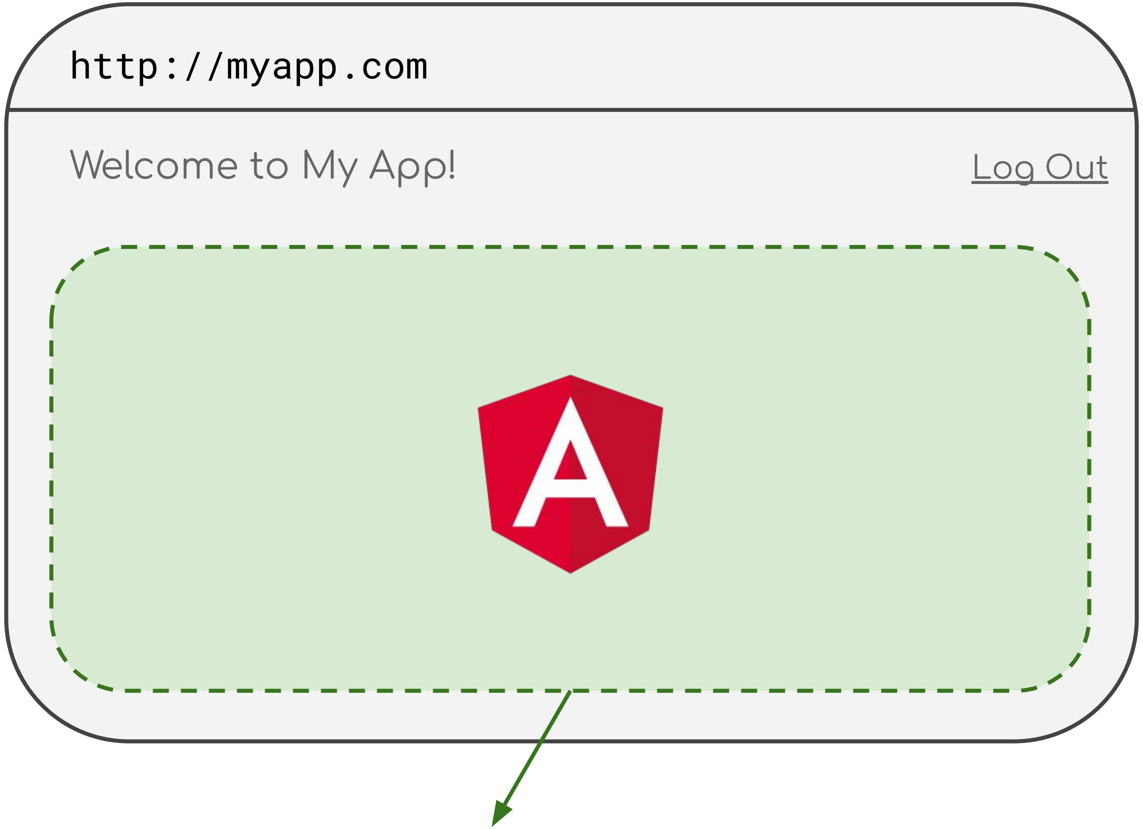
Introducing...

Micro Frontends

> *“An **architectural style** where independently deliverable frontend applications are composed into a greater whole”*

Micro Frontends



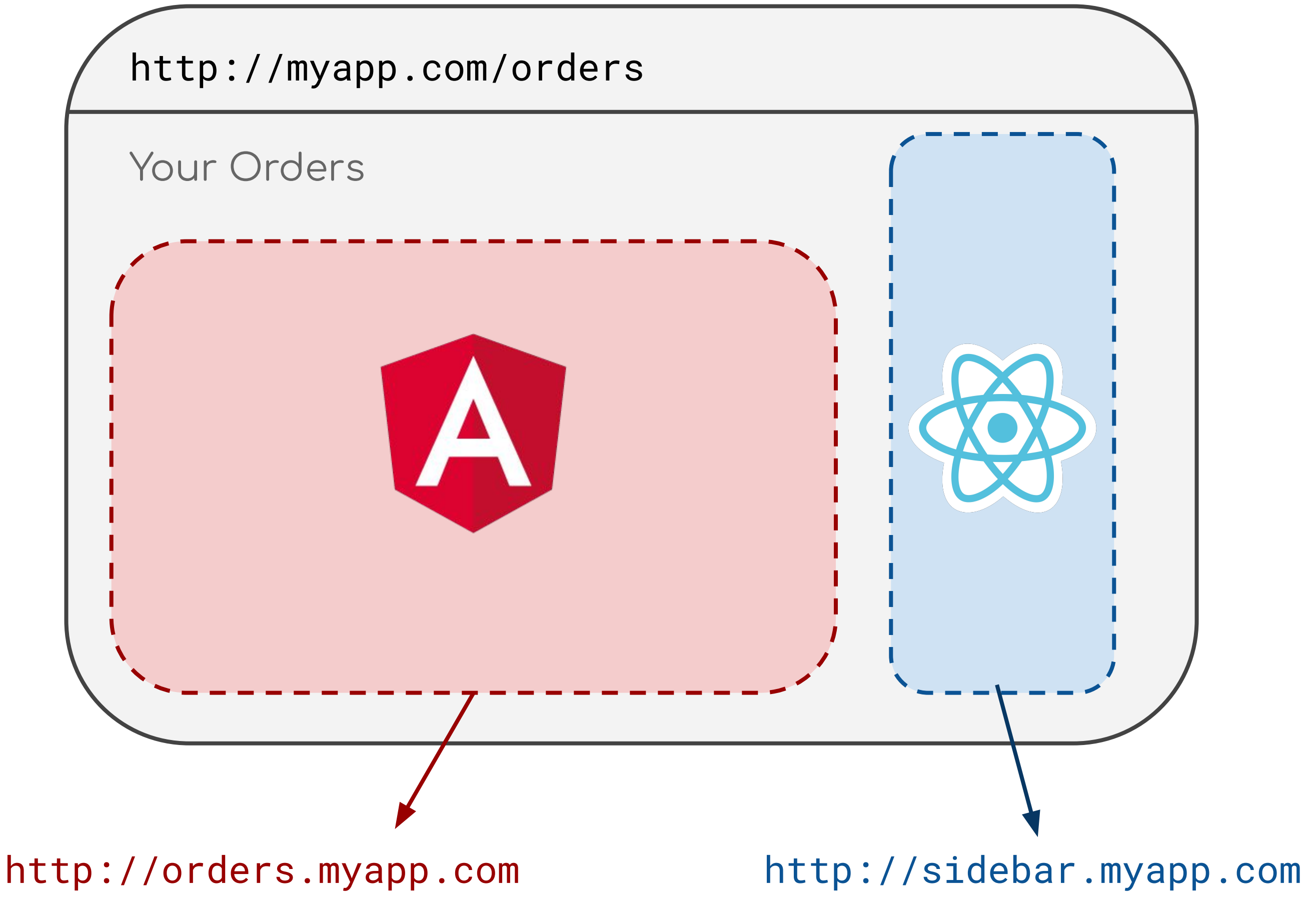


http://myapp.com

Welcome to My App!

Log Out

http://overview.myapp.com



http://myapp.com/orders

Your Orders

http://orders.myapp.com

http://sidebar.myapp.com

Micro Frontend Applications

Micro applications are **independently deliverable**

- Developed Independently
- Tested independently
- Deployed Independently

Sounds great...

But how do we implement this?

Micro Frontend Solutions

There are several options.

- Navigate between separate apps with a shared auth session.
- IFrames
- Web Components
- Runtime integration with JavaScript

single-spa

A framework for orchestrating micro frontends.

https://single-spa.js.org/

Single-spa application lifecycle

- `bootstrap()`
- `mount()`
- `unmount()`` ` ``

All single-spa applications must implement these three lifecycle methods. The framework handles the rest for you.

Single-spa Ecosystem

- `single-spa-angular`
- `single-spa-react`
- `single-spa-vue`
- `single-spa-angularjs`
- … (and many more)

Single-spa provides helper libraries for all popular front end frameworks.

These helpers implement the lifecycle methods for you, and provide other useful tools for building micro frontend apps.

Single-spa Concepts

- Applications
    - Lifecycle managed by single-spa activity function.
- Parcels
    - Lifecycle managed explicitly by your code.

Splitting Up Your Applications

- Monorepo
  - ~~😄 Easy to set up.~~
  - 😥 Tightly coupled. Deploy everything together.
- NPM Modules
  - ~~😐 Harder to set up.~~
  - 😐 Looser coupling, but root application needs to be frequently deployed.
- Dynamic Module Loading
  - ~~😥 Hardest to set up.~~
  - 😄 True decoupling. Fully independent deployments.

Options for Dynamically Loading Apps Into Single-Spa

- Plain old script tags!
  - `<script src=”https://my-app.com/main.js”></script>`
- Module Loaders!
  - SystemJS
  - RequireJS
  - jspm

- Supports all module types (ES6, CommonJS, etc),
- Polyfills **import-maps**

Import Maps

- Provides an abstraction over module paths. This **allows us to refer to our apps by name** in code, while dynamically configuring their remote location :

```
<script type="systemjs-importmap">{
  "imports": {
    "my-app": "https://my-app.com/main.js?version=1.5.2342"
  }
}</script>

...

System.import(`my-app`).then(...)
```

Common Myths

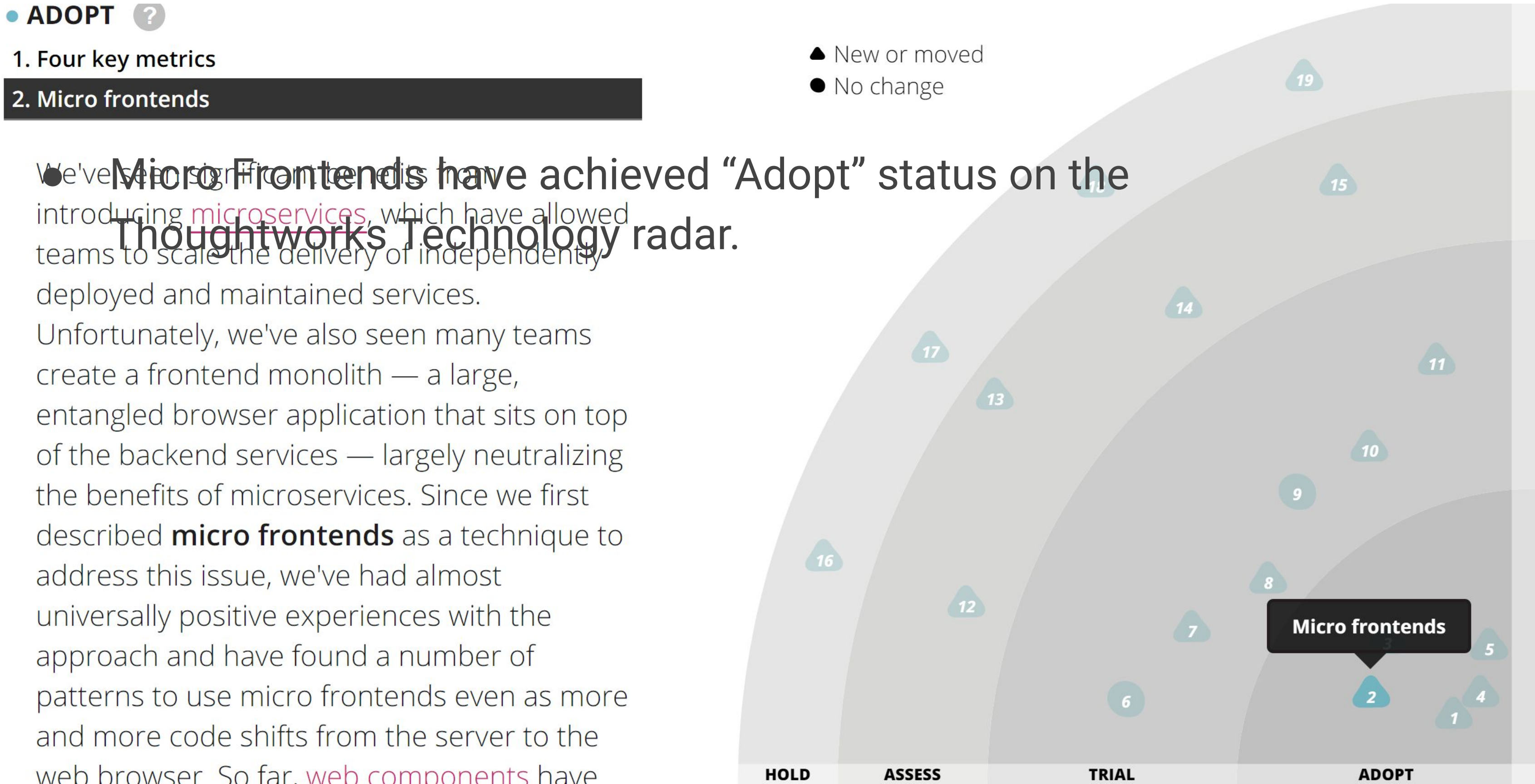
About Micro Frontends

“Micro frontends require framework libs to be downloaded again in each app, drastically increasing bundle size.”

- This is solvable, it just requires some extra setup:
  - Exclude shared dependencies from your bundles, e.g. using webpack externals.
  - Use global scope or import maps to provide the same dependencies at run time to all of your apps.

“Micro frontends are not a proven technique. It’s too early.”

- Micro Frontends have achieved “Adopt” status on the Thoughtworks Technology radar.



● ADOPT ?

1. Four key metrics

2. Micro frontends

▲ New or moved

● No change

We've seen significant benefits from introducing microservices, which have allowed teams to scale the delivery of independently deployed and maintained services. Unfortunately, we've also seen many teams create a frontend monolith — a large, entangled browser application that sits on top of the backend services — largely neutralizing the benefits of microservices. Since we first described **micro frontends** as a technique to address this issue, we've had almost universally positive experiences with the approach and have found a number of patterns to use micro frontends even as more and more code shifts from the server to the web browser. So far, web components have

“Micro frontends advocate using multiple frameworks and versions in the same application.”

- This is a notable and often-cited *capability* of micro frontends, but it is generally **not recommended** to do this.
  - It prevents us from sharing dependencies, and introduces more skillsets needed to maintain your application.
- There are use cases for this however, e.g. gradually migrating a large app from a legacy framework to a new framework.

demo

Building an Angular Micro Frontend

Next Steps

- Prefetch applications using resource hints. Take it even further with Guess.js
- Use webpack externals and import maps to remove shared dependencies from app bundles. (Huge savings)
- Deploy micro-app bundles to CDN under dynamic url and sync with import maps using remote configuration.
- Use NPM packages to share code and prevent duplication across micro-apps.
- Create a design system, or use an existing one like Material.

Downsides of Micro Frontends

- Challenging to set up initially.
- Complex infrastructure, just like with microservices.
  - Having dedicated DevOps helps.
- Local development and QA are more complex.
- Really needs a design system and infrastructure to easily share code and UI components.

Oct 16, 2019

Resources

- https://single-spa.js.org/docs.html
- https://martinfowler.com/articles/micro-frontends.html
- https://bit.ly/2tjZCng - Intro to module loaders.
- https://github.com/WICG/import-maps

Matt Gold
Tech Lead at
Kabbage

Animated Gifs by 100% soft (https://100soft.us/)

Kabbage

@mattbgold