How to break your monolith frontend app to more manageable micro frontends

Nipun S Bhasarkar
Web app development CoP
InfoCepts Technologies Pvt. Ltd

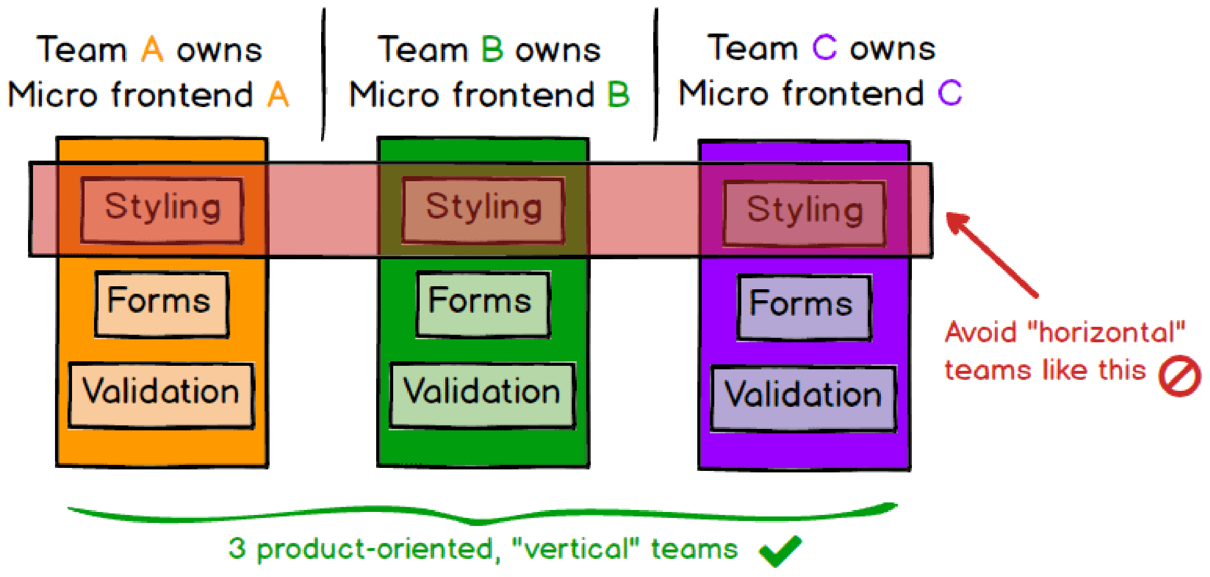
Problem Statement:

In today's world if an application want's to be competitive it should cater to all the needs of its customers and that too in a short span of time. This is not possible if your application is a huge monolith. This creates a bottleneck and makes it hard to scale development and that the teams keep stepping on each other's toes instead being freely able to work independently, with the tech stack of their choice and their own build and deployment pipeline. Instead of working separately from each other, the developers are bound to the same codebase. This makes development inefficient and hard to scale.

The idea of micro frontends is to build different frontends in decoupled codebases, which can be released through separated pipelines and are stitched together to create a cohesive application that feels unified to the user. Micro frontend as an idea has been wandering around the internet since 2015 and it caught pace since late 2018.

How Is It Done?

Micro frontends follow the same principle as microservices on the backend. Each application lives independently and has a well-defined objective. When we breakdown a monolith like this, it is also possible to breakdown into specialized teams.


Team A owns Micro frontend A
Team B owns Micro frontend B
Team C owns Micro frontend C
Styling
Forms
Validation
Styling
Forms
Validation
Styling
Forms
Validation
Avoid "horizontal" teams like this
3 product-oriented, "vertical" teams

Credit: https://martinfowler.com/

Using this approach also helps to integrate with the microservices built by the backend team. We can pick the frontend developers that build the micro frontend application A, with the backend developers that build the microservice of the application A. So now we have a full team whose only objective is to code the application A!

Advantages

A micro frontend architecture can deliver the following advantages:

1. Very easy to change the technology stack, since each application will naturally contain less code and will not interfere in other apps
2. Fast maintenance: as each application has only one concern. A bug can be easily spotted and fixed
3. Fast deployment: it's easier and faster to build and deploy little applications
4. Easy scaling: each application has its own scale requirement, so we can easily provide different environments.

Disadvantages

Every decision has its disadvantages, and it's not different with micro frontends:

1. Applications using shared libraries needs to be handled specially, so that the browser won't download same library multiple times
2. Code redundancy: some code may be repeated on each application. We surely could write a helper application that others use, but that would create a tight coupling betwee n them
3. Architectural complexity: it's way easier to manage a single monolith than multiple applications. To overcome this, there must be some automation and a lot of documentation

Ways to Build Micro Frontends

To breakdown a frontend monolith application we have the following techniques:

- ❖ One application can represent on page / module of the monolith
- ❖ Iframes: This is an old approach where we can have a single page with multiple inner applications, each one in a different iframe.
- ❖ Webpack module federation: This feature has been recently launched which aims to bundle different applications using WebPack. Here we can define how apps depends on each other and common libraries can be shared.
- ❖ Web Components: each application can be represented by a custom HTML element that could be orchestrated by a host application

Use cases

Case 1

Connected Systems – Nielsen: Connected Systems is a collection of different analytics applications coupled together by a similar look and feel. Each application is independently developed, deployment and maintained by its own team.

The advantage was new features and bug fixes were getting released bi-monthly, without much dependency on other applications.

Case 2

Common Data Selector: Each of the analytics applications had their own data selection experience and the UI and functionalities were similar across all their data selectors. There came a need to standardize this data selection experience. CDS was created as a SaaS standalone application which can be integrated into other analytics applications using iFrames and communication via post messages.

The advantage CDS brought to the table was each analytics app team didn't have to worry about their data selector anymore. This new data selector bought uniformity across all apps, uniform features and user experience was a standard. CDS features were configurable and could be turned on / off based on a configuration. Moreover, once deployed the new features and bug fixes were readily available across all the apps without the apps making any code change or deploying the new version of their app.

This saved at least $500,000 in terms of development and maintenance.

Learnings

- Breaking down monoliths and implementing micro frontends proved to be a game changer in terms of speedy development and release of new features and bug fixes.
- Small development teams working on individual code bases and fixed responsibility performed well.
- The iFrame approach is still used in some places, but the webpack module federation is the way to move ahead with micro frontends.

Conclusion

As frontend codebases continue to get more complex over the years, we see a growing need for more scalable architectures. We need to be able to draw clear boundaries that establish the right levels of coupling and cohesion between technical and domain entities. We should be able to scale software delivery across independent, autonomous teams.