Microfrontends using Module Federation

Creating a unified front-end experience using independently owned and deployed applications

Sales Deck

CrabFishTech - 2023

What is Microfrontend Architecture?

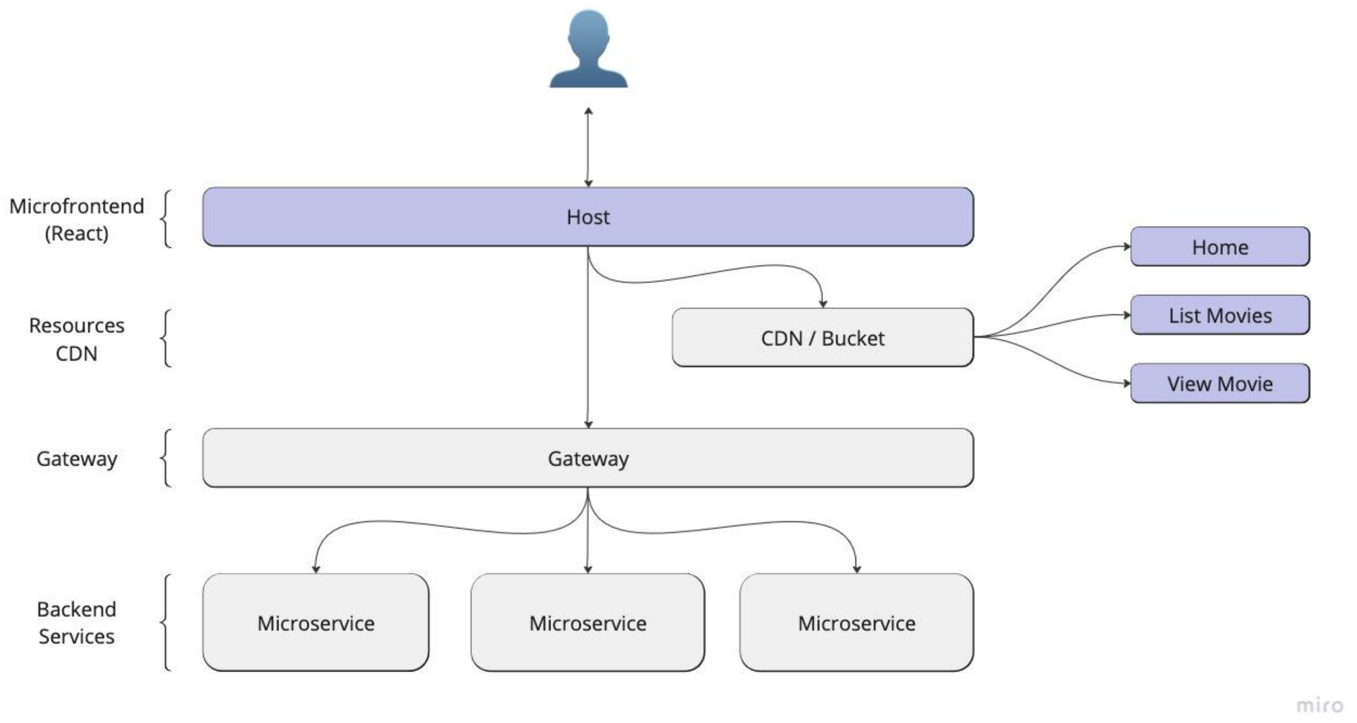
Microfrontend architecture is an approach to breaking up your front-end into a set of independently deployable, loosely coupled applications and/or components. These applications are then assembled together to act as a single user experience, once deployed.

Modern Web does this using a new technology called Module Federation. We have a single Host application which wraps up our single page applications represented as remotes.

What is Module Federation?

Module Federation aims to solve the sharing of modules in a distributed system, by shipping those critical shared pieces as macro or as micro as you would like. It does this by pulling them out of the the build pipeline and out of your apps.

Module Federation - Simplified



How does Module Federation make us better?

Module Federation Solves…

Module Federation is primarily designed to solve problems related to the development and maintenance of large-scale, complex web applications, especially those built using microfrontend architecture or distributed development teams. Here are some problems that this help addresses:

- Code Splitting and Lazy Loading: It allows for dynamic code splitting and lazy loading of JavaScript modules. This means that only the code needed for a specific part of the application is loaded when it's actually required, reducing the initial load time and improving performance.
- Cross-Team Collaboration: In larger projects or organizations, different teams may be responsible for different parts of an application. Module Federation enables teams to develop and maintain their modules independently while still integrating them into the overall application.
- Isolation and Sandboxing: Each microfrontend can run in its own isolated environment, reducing the risk of one module interfering with another. This isolation helps maintain the stability and integrity of the application.
- Runtime Module Resolution: Modules can be loaded dynamically at runtime, allowing for more flexible and adaptive applications. This is especially useful when dealing with user-specific customizations or extensions.
- Better Code Reusability: Module Federation encourages the creation of reusable components and modules, which can be shared across different microfrontends or projects.

The solutions we’ve seen so far…

A brief history of Microfrontends

Over the years, the term Microfrontend, has come in a bunch of different forms. I could touch on them all but overall you really just need to be aware of a handful. In the interest of easy to digest information, I will touch on only the top three microfrontend implementations that I personally have used.

- Hybrid: This is the common one most people think about when somebody mentioned Microfrontends. This is the pattern of ejecting individual components as js modules and combining them all in one page or application. These modules do not even need to be the same framework, you can mix and match angular, react, vue, and vanilla js.
- Distributed/Microsites: One of the less adopted patterns, that share similar structures such as Distributed Frontends. Without going into too much detail, this is the act of writing many small isolated standalone web applications and using infrastructure to tie them together to appear to be one single application experience. I am currently writing an article about this as well, but it has far more pitfalls to consider.
- Module Federation: Module Federation (MF) is a feature that allows for the dynamic loading of multiple versions of a module from multiple independent build systems. This allows for multiple systems to share code and be dynamically updated without having to rebuild the entire application. It also enables distributed teams and applications with different release cycles to share code without needing to wait for all systems to agree to and deploy a single shared version of a module.

Microsite Architecture / Distributed Frontends

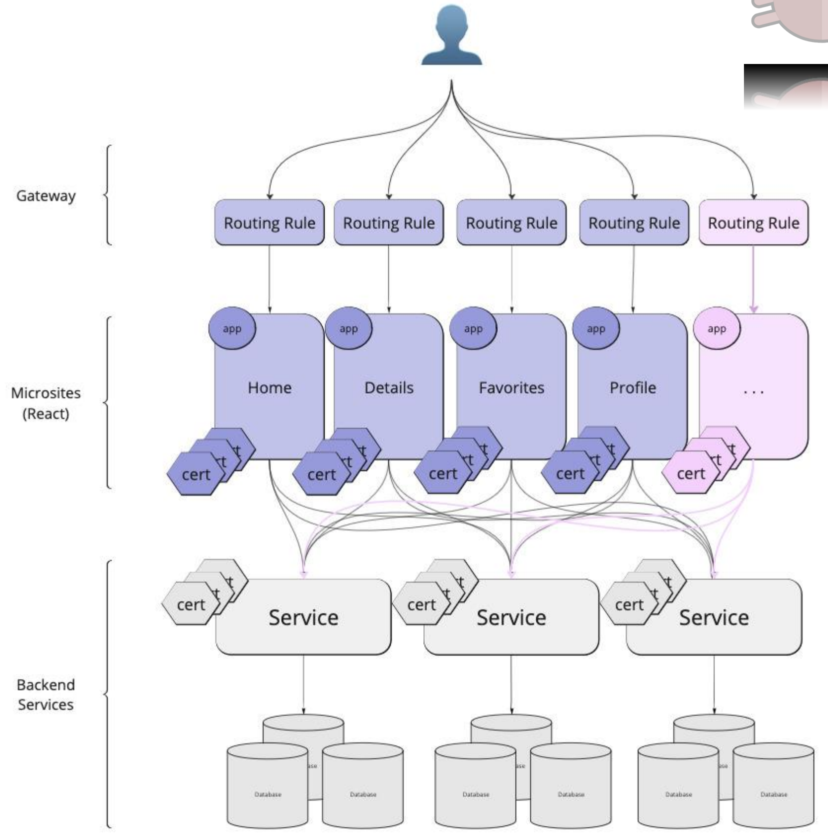
With the microsite architecture, or distributed frontends, we have many web applications deployed to independent infrastructure resources and all stitched together using routing to make it “feel” like a singular experience.

This pattern has a few positives:

- Independently deployable and owned pages or applications
- Language and Framework agnostic
- Each page can create their own experience

This pattern poses a few downsides too:

- Duplicative logic unless you have strict and relentless package management
- The risk of inconsistent design or common functionality
- Duplicative infrastructure and resources
- Higher resource costs
- Each page can create their own experience



Hybrid Microfrontends

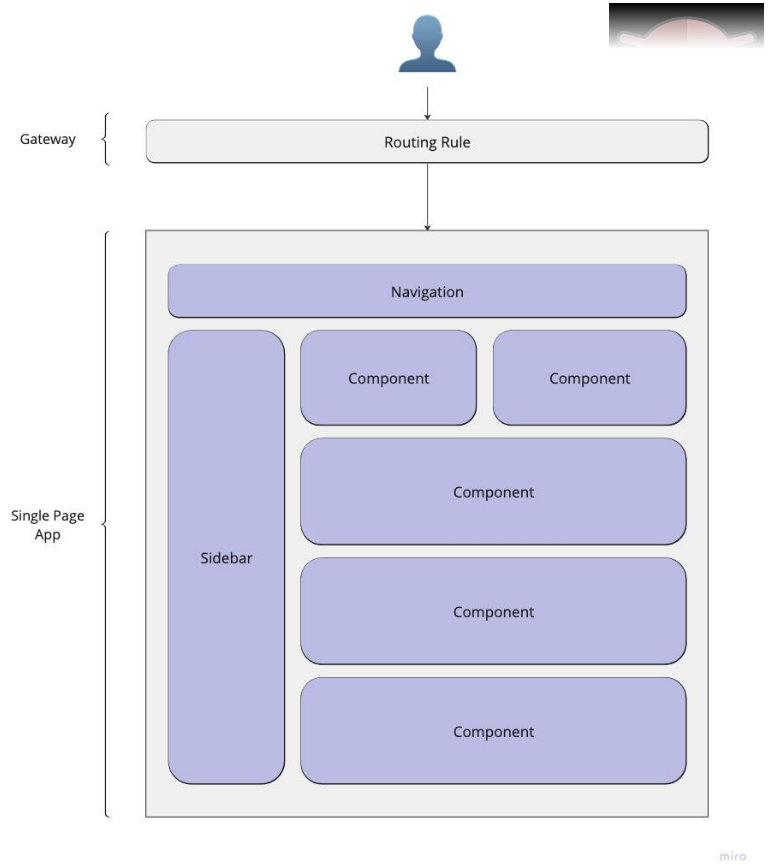
This is the common one most people think about when somebody mentioned Microfrontends. This is the pattern of ejecting individual components as js modules and combining them all in one page or application. These modules do not even need to be the same framework, you can mix and match angular, react, vue, and vanilla js.

This pattern has a few positives:

- Independently deployable and owned pages or applications
- Language and Framework agnostic
- Each component can be its own project in any framework

This pattern poses a few downsides too:

- Cannot share state across components
- Brittle page logic, if one component breaks, the whole page could be broken
- Duplicative logic unless you have strict and relentless package management
- The risk of inconsistent design or common functionality
- Multiple frameworks looks good on paper but are often quite different dev methodologies and rarely work together



Hosts? Remotes? What are these new terms?

Let’s talk terminology…

As you adopt Module Federation, there are a handful of terms you need to become comfortable with and where/when to use them.

- Hosts: A host is a simple parent application, which is put together during build-time or run-time from many smaller applications known as Remotes.
- Remotes: Remotes are easy, they are essentially single page applications, which are minified, bundled, and exported as javascript modules. They are wrapped in a Webpack shim, allowing a Host to know how to load and use them, and are essentially portable modules which can be used across Hosts or even by other Remotes!
- Modules: These are the parts that a Remote exposes for use inside a host or other remotes. This usually equates to components but not always!
- Shared: This is a collection of libraries/modules stored in a container at runtime, which are bundled together during deployment, so that any shared libs/modules that are used across remotes are only deployed once and used across the application.
- Federation: Federation refers to a system or approach where multiple independent modules or remotes are loaded in order to create a single experience while maintaining their autonomy and self-governance.

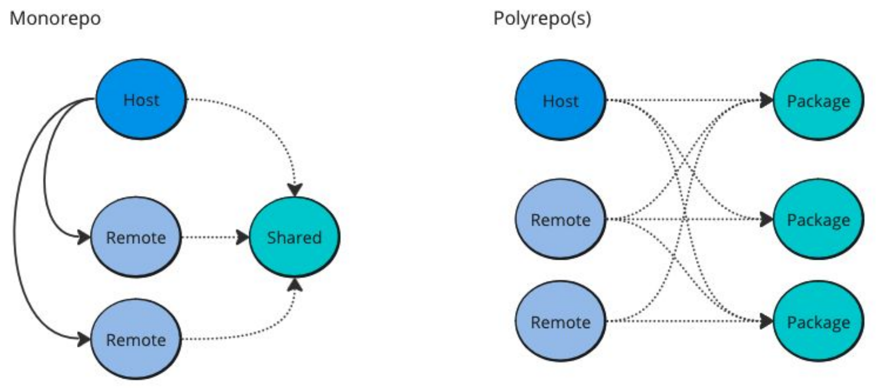
Monorepo or Polyrepo?

Monorepo or Polyrepo?

Honestly, this one is rather simple. Always start with a monorepo until you have your solution architecture, authentication, layout needs, and routing all figured out and configured.

This allows you to iterate quickly, avoid premature architecture decisions, determine your actual needs, onboard developers early.. And avoid expensive package versioning hell until you reach a "stable" point.

Once you feel confident in your solution, you may graduate to a Polyrepo and address your shared modules needs via packages.



If you want to learn more, I have an article:
https://medium.com/@cfryerdev/monorepo-vs-polyrepo-the-great-debate-7b71068e005c

The downsides to this new thing

Module Federation

While Module Federation offers numerous advantages, it also introduces certain challenges and considerations that developers need to be aware of:

- Complex Configuration: Setting up Module Federation can be complex, especially for those who are new to it. Configuring the webpack module federation plugin, managing shared modules, and dealing with various settings can be challenging.
- Debugging Complexity: Debugging can be more challenging in a microfrontend architecture, as each microfrontend may have its own development environment and debugging tools. Coordinating debugging efforts across different teams can be tricky.
- Dependency Management: Managing dependencies and ensuring that different microfrontends use compatible versions of shared libraries can be complex. Version conflicts or inconsistent dependencies can lead to runtime errors.
- Documentation and Communication: Documenting the architecture, dependencies, and communication between microfrontends is crucial to maintain a clear understanding of how the application works. Effective communication among teams is also essential.
- Complexity Trade-off: While Module Federation can simplify development in some aspects, it introduces its own level of complexity. Teams need to weigh the benefits against the added complexity to determine if it's the right solution for their project.

In conclusion, while Module Federation offers many benefits for building large and complex web applications, it also brings its own set of challenges related to configuration, runtime, testing, coordination, and complexity. These challenges can be managed with careful planning, documentation, and a clear understanding of the trade-offs involved.

Thank you!

Want to see some code? Go check out my github @ https://github.com/cfryerdev