MICRO FRONTEND

**What are MICRO FRONTENDS?**

Currently, microservices are used to separate the different components of an application, and they work perfectly well in the backend.

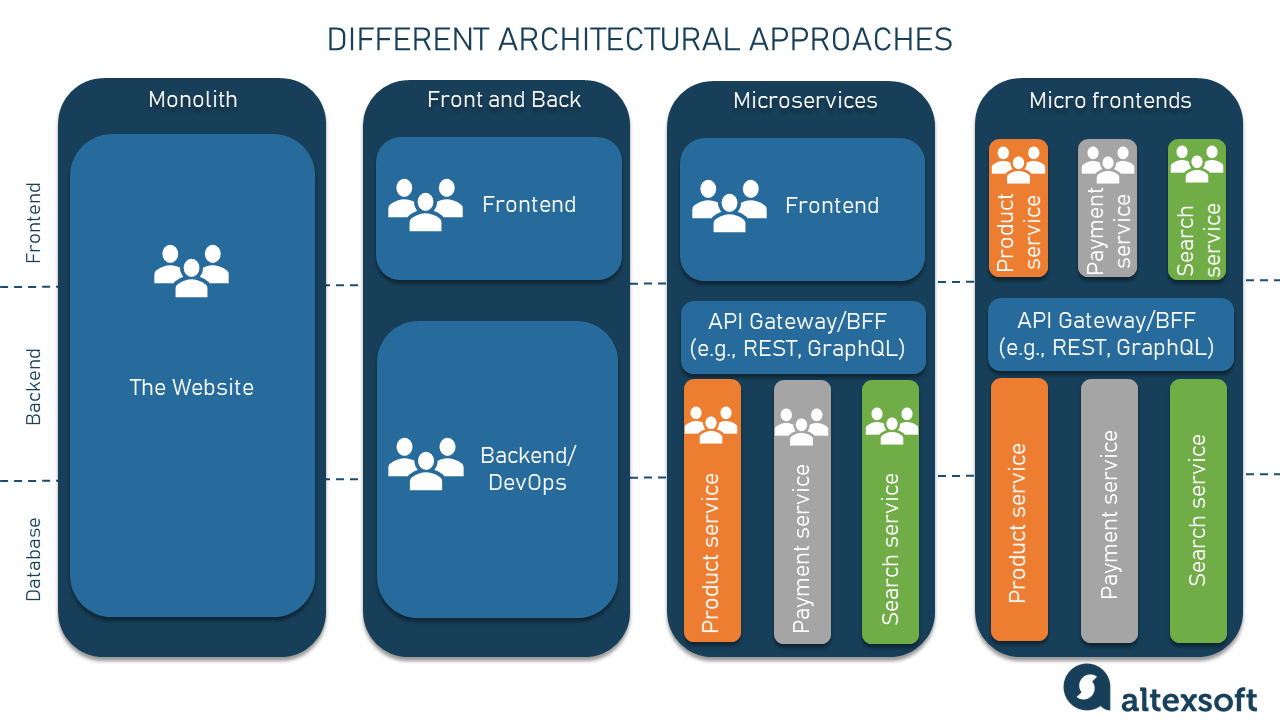
Thus, each application function is handled as an independent service, avoiding bottlenecks in the database.

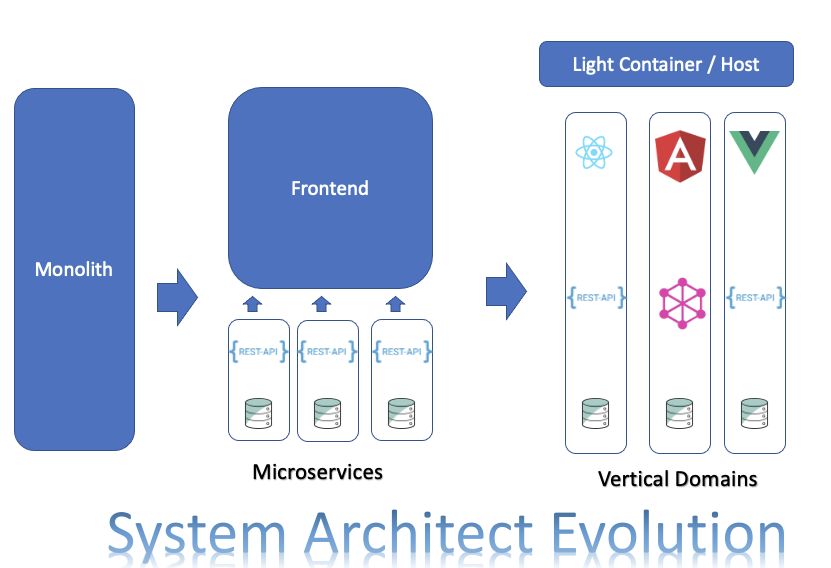
Micro frontends have a similar concept to microservices, i.e., they are the extension of the microservice concept to the frontend world.

The micro frontend is a type of architecture where a web application is divided into different modules or individual functions, implemented autonomously, allowing frontend teams the same level of flexibility and speed that microservices provide to backend teams.

**How do micro frontends work?**

A micro frontend can be a complete page or also specific fragments of the page, which can be used by other teams to add somewhere in the page they are developing. Unlike reusable components, micro frontends can be implemented independently as individual projects.





Types of microfront-ends

Monorepository:

This refers to applications in which all projects will ultimately be found within the same repository.

Multirepository:

You have the projects in different repositories. This implies having the projects more isolated, with an independent dependency system for each one.

Metarepository:

This refers to the strategy that seeks to adopt the two previous types. Thus, in this type there will be multiple repositories, but in addition there will be one where all of them will be integrated.

Each of these types has its advantages and disadvantages, and each has different ways of reducing their disadvantages.

Advantages of micro frontend architecture

* **Independent implementations**: With micro frontend architecture, the functions of a monolithic application are separated to become different sub-applications that are implemented independently by different work teams, instead of implementing all functions at once.
* **Higher performance**: In a monolithic application, the entire application must always be loaded, even if not all of its parts are used, making its initial load time high. With the micro frontend architecture, a specific module of the application is loaded on demand, thus reducing the initial loading time.
* **Faster implementation**: As each function is implemented independently, the development speed is significantly increased. To correct or add something within a function, the entire application does not have to be executed.
* **Reliability**: If one of the modules fails, the others will continue to function. This is very important for a large application, because it has multiple independent functions.

**What is Module Federation?**

Module federation allows a JavaScript application to dynamically load code from another application — in the process, sharing dependencies, if an application consuming a federated module does not have a dependency needed by the federated code — Webpack will download the missing dependency from that federated build origin.

## Why Is Webpack Module Federation Important?

Webpack Module Federation is a powerful feature revolutionizing JavaScript architecture and the micro-frontend approach. It can provide many benefits to your application system if appropriately utilized.

Before the introduction of Module Federation, code sharing tended not to be a smooth process. Micro-frontend implementation was also getting more complex.

**Module Federation: Terminology**

Following are some important terms you may need to familiarize yourself with when using Webpack Module Federation:

* **Host**: The Webpack build initialized first during a page build is the host. The host application contains typical features from a SPA or SSR application that boots and renders the components the user would see first.
* **Remote**: Remote is another Webpack build from which the host can consume a part. It can strictly be either a remote or a host. The major functionality of a remote is to expose modules to be consumed.
* plugins: [
* new container.ModuleFederationPlugin({
* remotes: {
* app\_two: 'app\_two',
* }
* })
* ]
* **Bidirectional host**: Bidirectional hosts can consume other applications or be consumed. So, this Webpack build or bundle can act as either a **remote** or a **host** at runtime.
* **Vendor Federation**: Vendor Federation is an important feature that aids in solving a significant performance issue in micro-frontend architecture. Regardless of where the modules are loaded from, it enables all or part of a remote or host’s npm modules to share at runtime declaratively.

**Configuration Options**

A build can both consume modules from other builds and provide modules to other builds at runtime, thanks to the **ModuleFederationPlugin**. You need to be familiar with several key configuration options, as Module Federation is configuration-based:

* **library**: The **library** contains a **name** (name of the library) and **type** options. It helps decide how the exposed code is retrieved and stored.
* **name**: The **name** configuration option uniquely identifies the exposed container.
* plugins: [
* new container.ModuleFederationPlugin({
* name: 'she1',
* })
* ]
* };
* **filename**: This determines the filename of the output bundle in Module Federation.
* plugins: [
* new container.ModuleFederationPlugin({
* filename: 'she1/remoteShop.js'
* })
* ]
* **exposes**: Through Module Federation, you can also share file types in addition to the modules. This configuration option depicts the path to the files or modules exposed by the container.
* **shared**: This is an essential configuration option, allowing you to share libraries on which the exposed module depends to run.

For More Information Options:

<https://webpack.js.org/plugins/module-federation-plugin/>

**Benefits of Webpack Module Federation**

* Using the **shared** option**minimizes dependency duplication**, as the remotes depend on the host’s dependencies. If the host lacks a dependency, the remote downloads its dependency only when necessary.
* **Server-side rendering is possible**, as Module Federation can work in any environment.
* **Enhances build performance**, as Module Federation supports the micro-frontend approach, allowing different teams to work simultaneously on a larger application by building and deploying independent, split projects.
* Module Federation **supports lazy loading bundles** to load modules only when necessary, resulting in better web performance.
* Module Federation **manages a dependency graph** for shared dependencies. It helps download necessary dependencies even when there is an issue like a network failure.
* Improves both **user experience** and **developer experience**.

**Micro frontends benefits**

1. **Scaling to multiple teams**

Unlike a traditional monolithic frontend, micro frontends allow several teams to work independently on different pieces of software.

The original work team is divided into several independent teams, where everyone can express themselves to the best of their ability, with each team making its own decision on architecture, coding style and testing, depending on each particular task.

2. **Faster development and deployment**

Micro frontends greatly improve the frontend development process, because instead of having a single team dealing with all the communication load that occurs with a monolithic frontend, there are now smaller autonomous teams, working at the same time, on the implementation of different functions.

3. **Adoption of different technological stacks**

Each of the micro frontends can be implemented using different technology stacks, as they are autonomous pieces of software, representing a great strength.

4. **Web applications that are easy to maintain**

Monolithic applications are difficult to maintain, especially when they are destined to grow. In contrast, the micro frontend architecture is made up of small parts, making them very easy to test and maintain.

5. **Continuous updates**

With the implementation of microservices, the flexibility to update backend functions without affecting other parts of the code increased. However, a monolithic frontend remained.

6. **Smaller, more manageable code**

By their very nature, micro frontends have a significantly smaller amount of code than monolithic frontends. This code is much more precise and easier to manage, so developers are less likely to make mistakes due to its complexity and can work faster and with less effort.

7. **Reuse of functionality**

Micro frontends are especially beneficial when developing several applications with common workflow requirements. By using micro frontends, companies can take the commonality of functionality, saving time and effort when creating new applications.

Best practices with micro frontends

**Find the right size for micro frontends**

This is similar to thinking about how to size microservices. If they are too small, the application will be too fragmented, but if they are too large, the application will be too loosely connected. Therefore, a balance must be sought.

**Do not create too many micro frontends**

If too many micro frontends are created, the application may become excessively fragmented, creating components with no real value. There is a lot of literature that analyzes the correct size of micro frontends and what should or should not be built in a microservice.

**Single Page Application (SPA)**

Each micro frontend should itself be a single page application. This system of micro frontends can be easily created using a meta-framework for single-page applications, such as Vue, ReactJS, Angular, Svelte, etc., which allows multiple development frameworks to be combined into a single page, without the need to update it.

**Communication**

The different micro frontends must communicate with each other through a standard event bus. Thus, each module will work independently in its own framework, which will handle incoming and outgoing events.

**Component libraries**

Depending on the technological stack used by the main application, its modules must be organized in libraries, which will be "required" by the main application, thus being conformed by different components.

**Any Doubts?**

## 

## Thank You

Abhishek Negi Inn-304