**Kiến trúc Micro-frontend: Chia nhỏ ứng dụng web quy mô lớn**

Kiến trúc Micro-frontend là một mô hình phổ biến để xây dựng các ứng dụng web quy mô lớn. Ý tưởng đằng sau nó là chia nhỏ một ứng dụng duy nhất thành các phần nhỏ hơn, độc lập, hay còn gọi là micro-frontend, có thể được phát triển, kiểm thử và triển khai độc lập. Mặc dù cách tiếp cận này mang lại một số lợi ích, nhưng nó cũng đặt ra những thách thức mới về giao tiếp giữa các micro-frontend.

Micro-frontend là một phong cách kiến trúc của phát triển web frontend, trong đó một ứng dụng được chia thành các tính năng – micro-frontend – và được triển khai độc lập. Điều này được thực hiện để cải thiện chất lượng và hiệu quả của các nhóm chịu trách nhiệm về mã frontend.

Mỗi micro-frontend có thể có kho mã nguồn, tập hợp các phụ thuộc, các test case tự động và pipeline triển khai riêng. Mỗi micro-frontend thường được sở hữu bởi một nhóm frontend duy nhất và được phát triển, kiểm thử và triển khai độc lập với các tính năng khác, tăng hiệu quả của quá trình phát triển.

A diagram of a web application

Description automatically generated

**How micro frontend works: integration approaches**

Hãy bắt đầu bằng cách xem xét kiến trúc monolith. Nó được đặt tên theo một khối đá rắn là một thực thể duy nhất. Trong phát triển phần mềm, kiến trúc monolith đề cập đến một ứng dụng có thể được quản lý bởi một nhóm với cơ sở dữ liệu, backend và frontend được chia sẻ.

Khi các ứng dụng trở nên phức tạp hơn, các nhà phát triển đã thấy lợi ích của việc tách biệt phát triển backend và frontend. Sau đó, họ áp dụng cách tiếp cận microservices vào backend để làm cho ứng dụng có khả năng mở rộng hơn, dẫn đến các dịch vụ liên kết lỏng lẻo có thể được triển khai độc lập.

Bước tiếp theo hợp lý là áp dụng cách tiếp cận tương tự cho frontend, vì frontend monolith trở thành một điểm nghẽn cho phát triển nhanh chóng. Thay đổi frontend monolith ảnh hưởng đến toàn bộ ứng dụng, tăng thời gian biên dịch.

Ngoài ra, các vấn đề về sở hữu trách nhiệm phát sinh nếu nhiều nhóm làm việc trên một ứng dụng có kiến trúc monolith.

Ví dụ: hai nhóm phát hành nhiều tính năng quan trọng lên sản xuất, nhưng sau khi phát hành, thống kê hiệu suất giảm 30%. Cả hai nhóm đều cho rằng mã của họ hoàn hảo, vậy ai chịu trách nhiệm về sự giảm hiệu suất? Nếu mọi người đều chịu trách nhiệm, thì không ai chịu trách nhiệm cả.

Đó là lý do tại sao việc tạo ra kiến trúc micro-frontend có ý nghĩa. Một ứng dụng được chia thành các phần biệt lập, tự quản lý và độc lập, sau đó được kết hợp lại với nhau để trông giống như một trang web duy nhất.

Core ideas của microfontend

**Segmented parts, thicker domain boundaries(**Các phần được phân đoạn, ranh giới miền dày hơn): mỗi tính năng tồn tại trong một micro-frontend riêng biệt.

**Autonomous teams(**Các nhóm tự trị): không cần phải phối hợp các bản phát hành với các nhóm phát triển khác.

**Loosely coupled app(**Ứng dụng lỏng lẻo): tất cả các thành phần có thể được xây dựng, triển khai, mở rộng và thậm chí thất bại độc lập.

**Shell:** kết hợp các micro-frontend lại với nhau thành một thứ thống nhất trên một trang duy nhất.

Các cách tiếp cận tích hợp micro-frontend khác nhau được mô tả tốt nhất là một thanh trượt. Ở một bên của thanh trượt, ứng dụng ít tách biệt hơn và do đó có ít lợi ích của micro-frontend hơn. Để bảo vệ nó, các cách tiếp cận này có thiết lập dễ dàng hơn nhiều - chúng yêu cầu một phiên bản khung công tác micro-frontend và một phiên bản thư viện. Giả sử bạn di chuyển thanh trượt sang phía bên kia với sự tách biệt nhiều hơn. Trong trường hợp đó, ứng dụng kết thúc với độ phức tạp thiết lập lớn hơn và chi phí bảo trì, nhưng tất cả các lợi ích của micro-frontend có thể tỏa sáng đầy đủ.

A green and white progress bar

Description automatically generated

**+ Routing: loading separate applications**

Ý tưởng đằng sau cách tiếp cận này là ứng dụng được tách riêng dựa trên định tuyến và mỗi tuyến đường tải một ứng dụng hoàn toàn độc lập, bao gồm cả các gói khung bên trong. Trong trường hợp này, cơ sở hạ tầng hoạt động như một vỏ bọc và chuyển hướng giữa các ứng dụng xảy ra thông qua ví dụ: cổng API hoặc CDN.

Định tuyến hoạt động cho tích hợp cấp độ trang khi các nhóm khác nhau sở hữu các trang của ứng dụng và mỗi micro-frontend được coi là một ứng dụng trang đơn.

Định tuyến có thể đạt được với các liên kết HTML. Trong trường hợp đó, trang tải lại trên mỗi trang. Không có khả năng chia sẻ dữ liệu với cách tiếp cận này, chỉ có các tham số truy vấn, bộ nhớ phiên hoặc lưu trữ dữ liệu trong cơ sở dữ liệu.

**+ WebComponents**

**Web components** ngụ ý việc xây dựng mỗi micro-frontend như một thành phần độc lập có thể được triển khai độc lập dưới dạng một tệp .js. Ứng dụng tải và hiển thị chúng vào các placeholder được tạo đặc biệt trong layout. Web components thông báo cho trình duyệt khi nào và ở đâu để tạo một thành phần.

Nhược điểm của cách tiếp cận này là mỗi web component khá lớn và cần đóng gói phiên bản framework của nó. Vì vậy, mặc dù bạn có thể tách biệt nhiều thành phần như bạn muốn, hiệu suất cần được theo dõi cẩn thận.

**+ Module Federation**

**Module Federation** là một cách tiếp cận khác giúp tất cả các thành phần micro-frontend trông giống như một ứng dụng duy nhất. Trên thanh trượt các mức độ tách biệt khác nhau được sử dụng trong micro-frontend, module federation có thể được đặt ở đâu đó ở giữa. Module federation được phát hành với Webpack 5 (2020), vì vậy nó hiện đại và có thể được áp dụng bởi các framework khác nhau.

**Module Federation** cho phép tải các micro-frontend tại thời gian chạy vào một ứng dụng shell mà không cần bất kỳ phụ thuộc thời gian xây dựng nào (chỉ cần xác định 'hợp đồng' tích hợp). Điều này đặc biệt hữu ích khi tùy chỉnh các trang của bạn cho các khách hàng cụ thể. Trang web của bạn có thể có một mô-đun 'thanh toán' được tải từ một gói 'chung' cho hầu hết khách hàng và một mô-đun 'tùy chỉnh' (có thể thậm chí được xây dựng bởi nhóm phát triển của khách hàng) cho những người khác. Module federation cho phép bạn thay đổi cấu hình micro-frontend của mình tại thời gian chạy.

**+ Micro-frontend dựa trên iframe**

Cách tiếp cận iframe nhúng mỗi micro-frontend vào iframe riêng của nó, và một ứng dụng trang đơn (SPA) đóng vai trò là container. Về cơ bản, đó là một tài liệu HTML được đặt bên trong một tài liệu HTML khác. SPA điều phối giao tiếp giữa các iframe.

Iframe là một cách tiếp cận tương đối cũ để xây dựng kiến trúc micro-frontend. Mặc dù nó cung cấp mức độ tách biệt tốt, nhưng nhiều vấn đề cần được giải quyết để nó hoạt động (ví dụ: bảo mật, chia sẻ dữ liệu).

Khi nào nên sử dụng?

**+ Dự án của bạn có quy mô trung bình đến lớn**

Áp dụng cách tiếp cận micro-frontend là lý tưởng khi xử lý nhiều nhóm làm việc trên các tính năng riêng biệt. Kiến trúc này cung cấp khả năng mở rộng tốt hơn và cho phép triển khai tính năng nhanh hơn, mà không có nguy cơ ảnh hưởng đến toàn bộ trang web.

**+ Năng suất là ưu tiên hàng đầu của bạn**

Các nhóm độc lập làm việc riêng biệt trên các micro-frontend có thể nâng cao năng suất tổng thể. Với các nhóm làm việc trên các tính năng khác nhau cùng một lúc, mà không ảnh hưởng đến quy trình của nhau, phát triển ứng dụng có thể được hoàn thành một cách có tổ chức và hiệu quả hơn.

**Lợi ích của microfrontend:**

+ Triển khai nhanh hơn và quản lý phát hành tốt hơn

+ Tự do Công nghệ

+ Dễ dàng mở rộng

**Nhược điểm:**

\* Mặc dù micro-frontend mang lại nhiều lợi ích, nhưng không phải lúc nào nó cũng là giải pháp tốt nhất cho kiến trúc frontend của bạn. Thực tế, cách tiếp cận micro-frontend không thể giải quyết tất cả các vấn đề của bạn. Do đó, khi quyết định có nên sử dụng micro-frontend hay không, điều quan trọng là phải giải quyết các thách thức của nó.

+ **Payload size:** Micro-frontend nhân đôi các dependencies và đóng gói các framework và thư viện, làm tăng kích thước tải trọng của ứng dụng web. Mặc dù đây là một thách thức đáng kể, nhưng trang web trên micro-frontend sẽ tải nhanh hơn so với kiến trúc monolith.

Một số giải pháp một phần bao gồm việc lưu trữ cẩn thận các tài nguyên, lựa chọn kỹ lưỡng các phụ thuộc và tách biệt cẩn thận các trang ít được sử dụng.

+ **Cross pages communication:**

Giao tiếp giữa các micro-frontend khó triển khai và bảo trì, nhưng có những trường hợp các trang riêng biệt cần giao tiếp với nhau ở mức độ tối thiểu. Một phần của ứng dụng cần được thông báo bởi host hoặc các micro-frontend khác về tương tác của người dùng và rằng nó cần thay đổi, làm mới hoặc kích hoạt một hành động.

+ **Operational complexity**

Cuối cùng, việc áp dụng micro-frontend mang lại sự phức tạp về vận hành vì cách tiếp cận kiến trúc frontend này đòi hỏi nhiều kho lưu trữ, công cụ, nhiều pipeline xây dựng/triển khai hơn và cơ sở hạ tầng phức tạp hơn để tất cả hoạt động cùng nhau.

**LOGIC BEHIND THE SCENES?**

một số cách khác nhau để chia sẻ trạng thái và mã giữa các micro-frontend, đồng thời nhấn mạnh ưu điểm và nhược điểm của từng phương pháp. Chúng ta cũng sẽ đề cập đến các kỹ thuật tối ưu hóa liên quan đến nhiều micro-frontend để đảm bảo giao tiếp tối ưu.

**Sharing state and code between micro-frontend:**

**Xét useCase: “A Catalog and A Cart.”**

Trong thế giới monolith, mỗi domain nằm trong cùng một App, điều này giúp dễ dàng truyền dữ liệu từ domain này sang domain khác. Các framework khác nhau cung cấp cách riêng để truyền dữ liệu giữa các ranh giới domain này. Một trong những cách phổ biến nhất là duy trì dữ liệu ở một lớp cao hơn và truyền nó đến các ranh giới khác thông qua props.

Giờ đây, điều này không còn đơn giản khi toàn bộ ứng dụng được chia thành nhiều ứng dụng nhỏ hơn chạy độc lập và được nhúng trong một ứng dụng container duy nhất, còn được gọi là Micro Frontends. Vì vậy, kiến trúc với toàn bộ danh sách lợi ích của nó cũng đi kèm với một tập hợp các phức tạp và một trong số đó là Giao tiếp giữa các micro-frontend.

1. Dùng props

* Đây là kỹ thuật cơ bản nhất để giao tiếp chéo giữa các micro-frontend, trong đó ứng dụng container duy trì trạng thái và truyền nó đến các micro-frontend cần thiết.
* Lấy ví dụ về trang thương mại điện tử, micro-frontend Catalog nhận một callback để thêm sản phẩm vào giỏ hàng, chịu trách nhiệm tăng tổng số lượng và truyền số lượng đến micro-frontend Cart.

A screen shot of a computer code

Description automatically generated

Ưu điểm:

* Đây là một trong những kỹ thuật truyền dữ liệu rất nổi tiếng trong kiến trúc dựa trên thành phần.
* Hầu hết các framework đều hỗ trợ cách này.
* Người ta luôn có thể sử dụng các cấu trúc framework để tránh các vấn đề phát sinh từ prop, ví dụ: React Context, v.v.

Nhược điểm:

* Tăng cường sự liên kết giữa các micro-frontend và ứng dụng container.
* Khó đạt được nếu hai micro-frontend không sử dụng cùng một framework.
* Nó có thể ảnh hưởng đến hiệu suất tổng thể của ứng dụng vì nhiều lớp không mong muốn sẽ được hiển thị lại với mỗi thay đổi trạng thái.

1. Sử dụng Platform Storage Apis

* Trong kỹ thuật này, chúng ta có thể tận dụng các API lưu trữ tích hợp sẵn của nền tảng như Local Storage trong trình duyệt và Async Storage trong các giải pháp đa nền tảng như React Native cho các micro-frontend di động.
* Chúng ta có thể tạo một thư viện tiện ích lưu trữ đơn giản, cung cấp setter và getter từ các API lưu trữ. Giờ đây, thay vì khiến các micro-frontend giao tiếp thông qua ứng dụng container, mỗi micro-frontend khác nhau có thể đặt và đọc dữ liệu trực tiếp bằng cách sử dụng.
* Trong ví dụ dưới đây, Catalog và Cart, cả hai micro-frontend đều sử dụng thư viện tiện ích lưu trữ. Micro-frontend Catalog đặt số lượng sản phẩm được thêm mới nhất vào bộ nhớ, sau đó được micro-frontend Cart đọc.

A screenshot of a computer program

Description automatically generated

Ưu điểm:

* Có sẵn cho cả trình duyệt và thiết bị di động. Local storage cho trình duyệt và Async storage cho ứng dụng di động.
* Ít sự liên kết hơn so với truyền props giữa App và micro-frontend nhưng khó gỡ lỗi micro-frontend nào đang đặt dữ liệu.

Nhược điểm:

* Không phải là giải pháp khả mở rộng cho các ứng dụng lớn hơn. Nhưng có thể được sử dụng cho một tập hợp dữ liệu nhỏ. Luôn tốt khi đặt tên không gian dữ liệu vào bộ nhớ của nền tảng theo tên ứng dụng để tránh sự mơ hồ.
* Không phải là kỹ thuật an toàn để lưu trữ dữ liệu bảo mật.

1. Sử dụng Custom Events

* Kỹ thuật này phù hợp hơn với các web micro-frontend và là một kỹ thuật khả mở rộng hơn cho các micro-frontend thời gian chạy. Ý tưởng chính ở đây là tận dụng API sự kiện tùy chỉnh tích hợp sẵn của trình duyệt để xuất bản các sự kiện với dữ liệu từ một micro-frontend và các micro-frontend khác đăng ký các sự kiện để nhận dữ liệu. Kỹ thuật này gần nhất với kiến trúc điều khiển sự kiện trong thế giới microservices.
* Trong ví dụ dưới đây, Catalog và Cart, cả hai micro-frontend đều sử dụng thư viện sự kiện. Micro-frontend Catalog tạo ra một sự kiện ADD\_TO\_CART trong thanh ghi sự kiện và sau đó xuất bản cùng một sự kiện đó. Micro-frontend Cart đăng ký cùng một sự kiện và nhận được số lượng sản phẩm cập nhật trong giỏ hàng thông qua dữ liệu sự kiện. Chúng ta cũng cần đảm bảo rằng các sự kiện đã đăng ký được hủy đăng ký khi thành phần được tháo gỡ.

A screenshot of a computer program

Description automatically generated

Ưu điểm:

* Giải pháp tích hợp sẵn trong nền tảng trình duyệt.
* Rất gần với kiến trúc sự kiện bất đồng bộ trong thế giới microservices. Dễ hiểu đối với cả các nhà phát triển backend.
* Chi phí thiết lập cao nhưng dễ mở rộng.
* Xây dựng một cơ chế chung mà tất cả các nhóm micro-frontend có thể theo dõi.

Nhược điểm:

* Không thể đạt được trong trường hợp của micro-frontend di động.

1. T

**Giới hạn về framework, library SSR?**

Với project được build bằng các framework/library khác như Svelte, NextJS, Preact, hay server-side rendering,... Thì liệu có giới hạn là gì:

* Giới hạn của nó không chỉ ở mỗi microfrontend, thực tế về mặt kĩ thuật thì bất kì cái gì miễn nó là Module thì đều áp dụng được hết, Module ở đây có thể là Javascript module hay là cả CSS/SCSS/LESS,....
* Tất nhiên với mỗi framework/library khác nhau hay dùng các tool khác nhau (Vite, Rollup,...) thì đôi khi ta cần cấu hình khác đi chút.

**Vậy là mỗi microfrontend cần deploy ở 1 domain riêng?**

* mỗi MFE chúng cần được host ở đâu đó, và có thể được truy cập từ App shell. Nhưng bởi vì mỗi MFE được build ra static files (remoteEntry.js, loader,...) nên ta hoàn toàn có thể dùng 1 domain để share cho cả app-shell và toàn bộ các MFE (dùng các đường dẫn có prefix khác nhau chẳng hạn)

**Hot Module Replacement**

https://webpack.js.org/concepts/hot-module-replacement/