SOA - micro frontend research

tags: `uni`, `soa`, `micro-frontend`

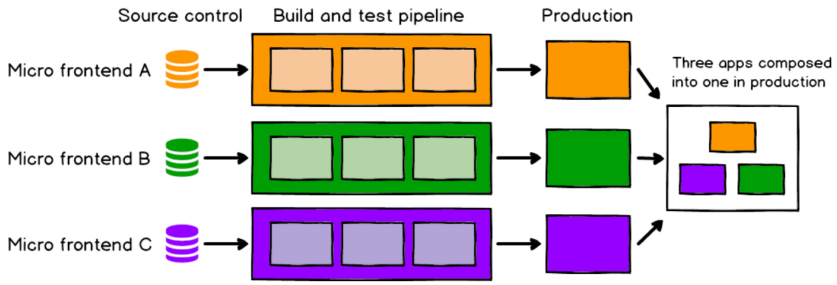
members:

- *Đinh Quốc Trung - 19020025*
- *Nguyễn Minh Quân - 19020019*
- *Lê Vũ Quang - 19020020*

---

1. What is micro frontend ?

- `microfrontend` là một cách tiếp cận dựa trên tư tưởng của `microservice`. Đấy chính là thay vì có một codebase chúng ta sẽ dùng nhiều codebase với các công nghệ có thể khác nhau, và trên từng codebase sẽ chỉ quan tâm đến một tính năng cụ thể.
- Có thể xem một ứng dụng web là một bộ kết hợp của nhiều tính năng, mỗi một tính năng như vậy được quản lý bởi một team.



2. Benefits

- **Library development:** Các team khác nhau sẽ thử và triển khai thành phần mà họ chịu trách nhiệm, chính vì vậy nó sẽ dễ dàng hơn cho việc tái sử dụng mã nguồn cho các project khác. Điều này thúc đẩy cho việc xây dựng một hệ thống thư viện riêng của công ty gồm các thành phần web và ứng dụng web.
- **Flexibility and variation:** Lập trình viên có thể chọn framework tuỳ thích (`react`, `vue`, `angular` …), hoặc bất cứ loại ngôn ngữ lập trình nào mà họ muốn dùng để phát triển thành phần frontend mà họ chịu trách nhiệm. Điều này sẽ giúp cho các lập trình viên thoải mái và có thể làm việc với hiệu suất tối đa.
- **Speed:** Nếu dự án có đủ tài nguyên để cung cấp đủ số team frontend, thì ứng dụng có thể được tạo ra một cách nhanh chóng với các nhóm làm

việc trên các thành phần của riêng họ, và họ sẽ chịu trách nhiệm về việc thử nghiệm, triển khai của thành phần đó.

- **Scalability:** Nếu dự án phức tạp, nó sẽ cần được scale dần theo thời gian. Và nếu công ty đủ nguồn lực cho nhiều team, thì các team này có thể làm việc và phát triển giao diện người dùng của riêng họ, dẫn đến việc mở rộng ứng dụng nhanh chóng hơn và dễ dàng quản lý hơn.
- **Developer experience:** Việc học code cho một microfrontend sẽ dễ dàng hơn so với việc học code một hệ thống đơn khối. Đồng thời lập trình viên có thể chọn ngôn ngữ mà họ quen thuộc khi sử dụng trên thành phần đấy. Điều này cho phép các doanh nghiệp tiếp cận với đội ngữ nhân viên tiềm năng hơn, có khả năng làm việc trên nhiều ứng dụng, vì họ không cần chỉ dựa vào những thành viên có kinh nghiệm nhất. Điều này còn có thể giúp ích cho việc sử dụng ngân sách.
- **Independent deployment and testing:** Các nhóm chỉ triển khai thành phần mà họ đã phát triển. Bởi vì các giao diện người dùng là độc lập, nên việc triển khai và các thay đổi chỉ có thể ảnh hưởng đến dịch vụ cụ thể đó mà không gây ra các thay đổi cho toàn bộ hệ thống. Điều này cũng có nghĩa là các nhóm chỉ cần chịu trách nhiệm kiểm tra và giám sát một khía cạnh của tổng thể, tránh việc toàn bộ dự án phải được theo dõi và kiểm tra cùng một lúc.

3. Drawbacks

- **Resources:** Để những ưu điểm trên của kiến trúc micro frontend được thực hiện, dự án phải có quyền truy cập vào nhiều nguồn lực. Nếu một doanh nghiệp không có đủ quy mô hoặc nguồn lực cho các dự án micro frontend, thì microservices thực sự có thể trở nên cồng kềnh và đòi hỏi khắt khe hơn là sử dụng monolith. Một nhóm sẽ chịu trách nhiệm về việc tạo, thử nghiệm và triển khai nhiều mô-đun, có thể trên nhiều ngôn ngữ, thay vì chỉ tham gia vào một codebase. Đây không phải là cách sử dụng thời gian hiệu quả. Nếu có các nhóm riêng biệt, thì microservices mới có thể được xem xét.
- **Complexity:** Có một nguy cơ khi sử dụng microfrontend là việc các lập trình viên có thể sử dụng quá nhiều thành phần trong dự án của họ theo thời gian, gây ra nhiều vấn đề ở giai đoạn triển khai và thử nghiệm. Giải quyết điều này đòi hỏi một giai đoạn lập kế hoạch chi tiết và phức tạp để quyết định số lượng và mức độ chính xác của các thành phần sẽ được sử dụng. Tương tự như vậy, mỗi nhóm nhà phát triển sẽ yêu cầu một bộ KPI chính xác để tuân theo.
- **Payload:** Microservice được sử dụng có thể dẫn đến nhu cầu nhân bản các dependencies chung. Ví dụ như nếu một microfrontend yêu cầu một chương trình cụ thể phải được cài đặt để nó hoạt động, thì phía người dùng cũng được yêu cầu tải một bản sao tương tự. Mà có thể có nhiều thành phần yêu cầu như vậy, dẫn đến người dùng phải tải về nhiều thành phần giống nhau.

4. Integration

a. Build time integration

- Là phương pháp coi các ứng dụng như một package và ứng dụng chính sẽ thêm các ứng dụng con như một thư viện. Ví dụ:

```
{
  "name": "@micro-frontends/container",
  "version": "1.0.0",
  "description": "Micro frontends demo",
  "dependencies": {
    "@micro-frontends/products": "^1.2.3",
    "@micro-frontends/checkout": "^4.5.6",
    "@micro-frontends/user-profile": "^7.8.9"
  }
}
```

- Hạn chế:
  - Chúng ta sẽ phải dịch lại các ứng dụng chính và release lại mỗi khi các ứng dụng con có thay đổi
  - Không có sự đồng bộ chức năng giữa các ứng dụng chính nếu chúng ta bỏ sót quá trình đồng bộ version của ứng dụng con (Đây cũng có thể là một điểm tốt nếu như chúng ta không muốn nâng cấp chức năng ở một trang nào đó)
  - Phụ thuộc các dependencies với nhau
    - Nếu project `@micro-frontends/container` sử dụng React và `@micro-frontends/products` cũng sử dụng React thì sẽ bị trùng lặp thư viện và tăng dung lượng khi tải trang web (vấn đề này đã đề cập ở phần *3.Drawbacks*)
    - Nếu project `@micro-frontends/container` sử dụng React và `@micro-frontends/products` sử dụng chung React với project chính thì sẽ bị phụ thuộc vào version của project chính.

b. Runtime integration via iframes

- Iframe là phương pháp giúp các tài liệu HTML có thể được nhúng vào bên trong một tài liệu HTML khác. Về bản chất, iframe giúp dễ dàng tạo một trang từ các trang con độc lập với nhau. Chúng cũng cung cấp một mức độ tách biệt tốt về styling và các biến toàn cục không can thiệp vào nhau.

```
<html>
  <head>
    <title>Micro frontends</title>
  </head>
  <body>
    <h1>Welcome to Micro frontends</h1>
```

```
    <iframe id="micro-frontend-container"></iframe>

    <script type="text/javascript">
      const microFrontendsByRoute = {
        '/': 'https://micro-frontends.tuando.net/demo/react-example',
        '/products': 'https://micro-frontends.tuando.net/demo/react-example/products'
      };

      const iframe = document.getElementById('micro-frontend-container');
      iframe.src = microFrontendsByRoute[window.location.pathname];
    </script>
  </body>
</html>
```

- Ưu điểm:
  - Không bị ảnh hưởng bởi styles (CSS) giữa các trang chính và trang trong iframe
- Hạn chế:
  - Phải tải lại toàn bộ trang khi thay đổi đường dẫn
  - Khó khăn trong việc giao tiếp giữa các chức năng

c. Runtime integration via Javascript

- Cách tiếp cận này là việc chúng ta khai báo các global function hỗ trợ render các chức năng ở các dự án con. Sau đó dự án chính sẽ gắn các *script bundle file* của các dự án con, lúc đấy nếu muốn hiển thị chức năng nào thì chỉ việc gọi chức năng đó thôi. Ví dụ:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

window.renderProducts = (containerId, history) => {
    ReactDOM.render(
        <App history={history} />,
        document.getElementById(containerId),
    );
};
```

d. Runtime integration via Web Components

- Cách tiếp cận này cho phép chúng ta khai báo một *HTML Custom Element*, ví dụ như ta khai báo một *HTML Custom Element* `<web-components-products></web-components-products>` thì chỗ nào muốn sử dụng ta chỉ cần chèn đoạn code `<web-components-products></web-components-products>` để dùng.

- Ưu điểm:
  - Không bị phụ thuộc dependencies giữa các dự án với nhau (ví dụ: khác version React giữa các dự án)
  - Vì cho phép tạo một HTML Custom Element nên ta có thể gắn thẻ HTML Custom này vào bất cứ đoạn code HTML nào, không quan trọng dự án đó đang sử dụng frontend framework nào
  - Hỗ trợ Shadow DOM: cho phép style css tồn tại độc lập, không ảnh hưởng giữa các dự án với nhau
  - Có thể phát triển theo hướng package (publish lên một registry) mà không cần phải có domain host cho dự án. Từ đó làm đơn giản hoá việc quản lý các version release.
- Hạn chế:
  - Không thể chia sẻ tài nguyên giữa các dự án với nhau (ví dụ: sử dụng chung thư viện React)

e. Module Federation Webpack 5

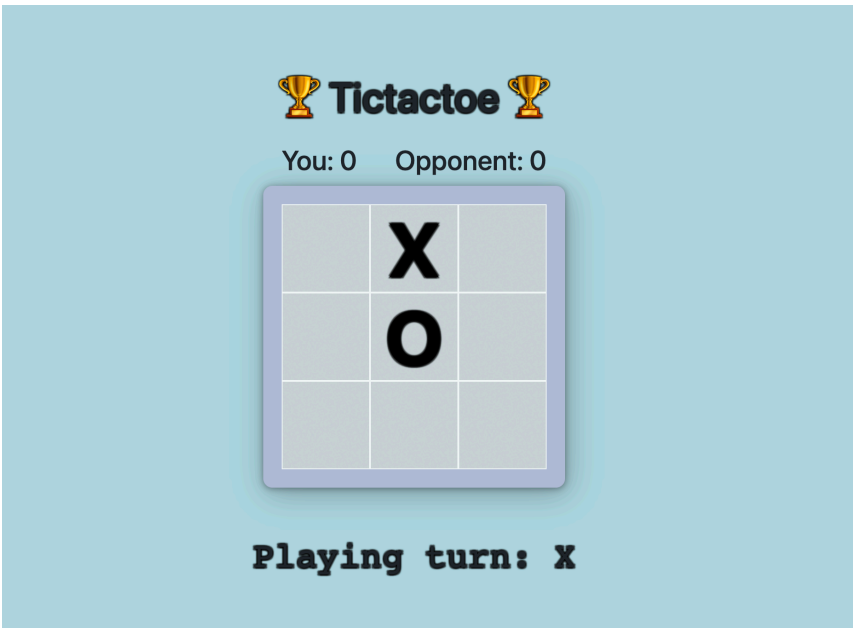
- Module Federation là một tính năng của Webpack 5. Nó cho phép chúng ta cấu hình để một ứng dụng có thể dynamic load code từ một ứng dụng khác.
  - *Ví dụ:* chúng ta có 2 ứng dụng được phát triển độc lập A và B, ứng dụng B là một phần nhỏ chức năng của ứng dụng A. Module Federation sẽ cho phép ta nhúng ứng dụng B vào ứng dụng A và chia sẻ tài nguyên giữa chúng.
- Ưu điểm:
  - Có thể chia sẻ tài nguyên giữa các dự án. Ví dụ dự án A sử dụng React 16.x và dự án B cũng sử dụng React 16.x thì khi tải module B sẽ không cần phải tải thêm React một lần nữa, nếu 2 version khác nhau thì nó sẽ tự động tải thêm version React còn thiếu.
  - Giao tiếp giữa các dự án đơn giản, có thể sử dụng chung một Redux store giữa các dự án với nhau
- Hạn chế:
  - Các dự án phải sử dụng Module Federation của Webpack 5.x
  - Buộc các dự án phải có các static domain để tải các bundle file tương ứng. Vì các chức năng Module Federation chỉ hỗ trợ cấu hình tải các file từ một remote url

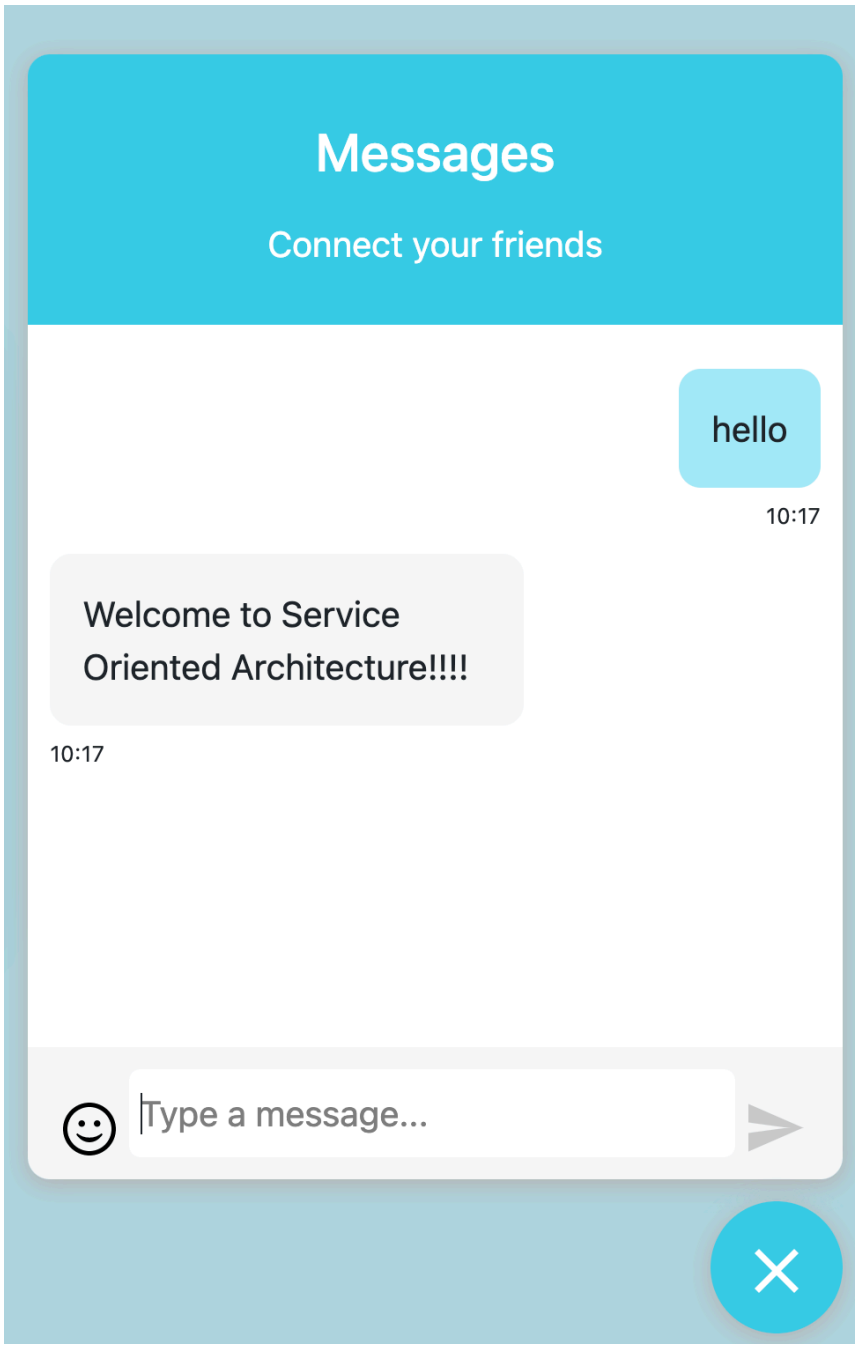
5. Demo - Tic tac toe

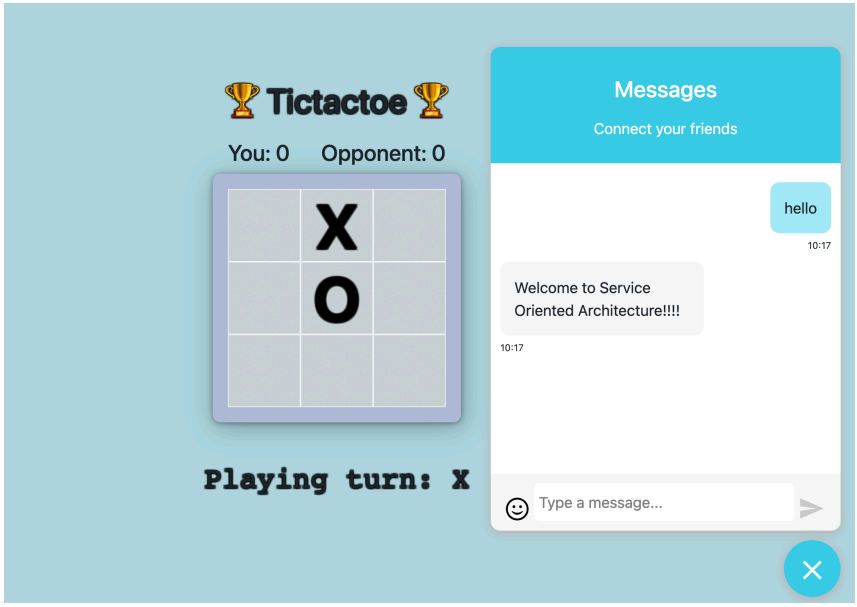
- Github: https://github.com/minhquanym/MicroFrontendDemo?fbclid=IwAR0aJgzSPYmfwGbFgqWilB
DdkudsYs_lOE
- Sử dụng **Module Federation Webpack** để triển khai microfrontend gồm 3 components:
  - Game **Tictactoe**: Vue
  - Khung **Chatbox**: React
  - **Container**: React

- Mỗi component sẽ được triển khai riêng biệt. **Container** sẽ gửi request HTTP đến các component con để thực hiện việc render chúng mỗi khi cần đến.
- Như vậy, việc phát triển các thành phần được tách biệt hoàn toàn, kể cả framework khi ta đã sử dụng **Vue** để triển khai game và **React** để triển khai chatbox

Ảnh demo

- 

- 
Messages
Connect your friends
hello
10:17
Welcome to Service Oriented Architecture!!!!
10:17
Type a message...

- 
Tictactoe
You: 0
Opponent: 0
X
O
Playing turn: X
Messages
Connect your friends
hello
10:17
Welcome to Service Oriented Architecture!!!!
10:17
Type a message...