

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA CÔNG NGHỆ PHẦN MỀM



BÁO CÁO ĐỒ ÁN
SEMINAR VỀ CÁC VẤN ĐỀ HIỆN ĐẠI CỦA
CÔNG NGHỆ PHẦN MỀM

ĐỀ TÀI

**ÁP DỤNG KIẾN TRÚC MICROFRONTEND VÀO
ỨNG DỤNG ĐẶT PHÒNG KHÁCH SẠN BẰNG
WEBPACK MODULE FEDERATION**

Giảng viên phụ trách: ThS. Đinh Nguyễn Anh Dũng

Sinh viên thực hiện:

Trần Tuấn Kiệt – 21522266
Trần Thị Tuyết Mai – 21520340

Mục lục

Chương 1. Lý thuyết.....	7
I. Giới thiệu.....	7
1. Đặt vấn đề	7
2. Mục tiêu của báo cáo	7
3. Phạm vi nghiên cứu	7
II. Tổng quan về kiến trúc Microfrontend.....	7
1. Phạm vi nghiên cứu	7
2. Lịch sử phát triển	9
3. So sánh Microfrontend với kiến trúc truyền thống.....	9
4. Các thành phần chính.....	9
5. Các phương pháp tích hợp	10
6. Ví dụ	12
7. Các loại tích hợp chính	14
III. Áp dụng Webpack Module Federation xây dựng Microfrontend cho ứng dụng đặt phòng khách sạn.....	16
1. Sơ lược về Webpack	16
2. Webpack development server.....	17
3. Cài đặt Module Federation	19
4. Hiểu hơn về Module Federation	22
5. Hiểu hơn về các tùy chọn cấu hình.....	25
IV. Quy trình phát triển	27
1. Chia sẻ các phụ thuộc giữa các ứng dụng (sub-project)	27
2. Yêu cầu dẫn đến lựa chọn kiến trúc.....	29
3. Tại sao phải sử dụng mount function để render UI?.....	36
V. Lợi ích của Microfrontend trong phát triển ứng dụng.....	38
1. Triển khai nhanh hơn và quản lý bản phát hành tốt hơn	38
2. Nhiều nhóm có trách nhiệm khác nhau	38
3. Tự do công nghệ	39
4. Dễ dàng mở rộng quy mô	39
5. Triển khai liên tục	39
Chương 2. Ứng dụng Microfrontend trong lĩnh vực đặt phòng khách sạn .	40
I. Yêu cầu chức năng	40
II. Yêu cầu phi chức năng.....	42

III. Use case:	44
1. Use case tổng quan:	44
2. Đặc tả Use case:	44
IV. Các module microfrontend chính:	49
1. Team Quản lý đặt phòng.....	49
2. Team Thanh toán	50
3. Team Quản lý người dùng	50
4. Team Đánh giá và Xếp hạng	50
5. Team Marketing.....	50
V. Một số ví dụ trùu tượng hoá các microfrontend trên trang.	50
1. Trang đăng nhập/đăng ký:	52
2. Trang chủ:	53
3. Trang chi tiết.....	54
4. Trang Booking:.....	55
5. Trang Thanh toán:.....	55
VI. Thực hiện hoá các microfrontend lên trên trang	56
1. Trang Đăng nhập/Đăng ký:	56
2. Trang Hồ sơ người dùng:.....	57
3. Trang chủ:	58
4. Trang Chi tiết khách sạn:	59
Chương 3. Tổng kết	60
I. Thách thức khi áp dụng Microfrontend	60
1. Kích thước tải trọng	60
2. Giao tiếp giữa các trang	60
3. Sự khác biệt về thiết kế.....	61
4. Độ phức tạp về mặt vận hành	61
II. Thực tiễn và ví dụ thành công	61
1. Các công ty đã áp dụng Microfrontend	61
2. Nghiên cứu điển hình về ứng dụng đặt phòng khách sạn (Case Study)62	62
III. Tóm tắt các điểm chính	62
Chương 4. Kế hoạch thực hiện	63
Tài liệu tham khảo	65

Mục lục hình ảnh

Hình 1.1: Kiến trúc micro-fontend	8
Hình 1.2: Mỗi micro frontend được triển khai một cách độc lập	8
Hình 1.3: Kiến trúc monolithic và micro-fontend	9
Hình 1.4: Thanh trượt các cách tiếp cận tích hợp microfrontend	10
Hình 1.5: Giải sử 2 thành phần trong thương mại điện tử	13
Hình 1.6: Mô hình hoá microfrontend cho 2 thành phần trên	13
Hình 1.7: Mô hình tổng quan của 2 microfrontend trên	14
Hình 1.8: Các loại tích hợp chính – Tích hợp build-time	15
Hình 1.9: Các loại tích hợp chính – Tích hợp runtime	15
Hình 1.10: Cách thức hoạt động của Webpack	17
Hình 1.11: Cách thức hoạt động của Webpack development server	17
Hình 1.12: Bundle có trên trình duyệt sau khi đi qua Webpack development server	18
Hình 1.13: Webpack development server chia nhỏ bundle thành các chunk	18
Hình 1.14: Các bước cài đặt Module Federation	19
Hình 1.15: Cách sử dụng Module Federation trong Webpack với hai ứng dụng Products và Container	20
Hình 1.16: Quy trình đóng gói của Webpack khi sử dụng Module Federation cho ứng dụng Products	22
Hình 1.17: Quá trình sử dụng Webpack để gộp và đóng gói các tệp Javascript từ các thành phần khác nhau	23
Hình 1.18: Quá trình tải và thực thi các tệp JavaScript giữa hai Webpack Dev Server khác nhau	24
Hình 1.19: Tuỳ chọn cấu hình đối với Host	25
Hình 1.20: Tuỳ chọn cấu hình đối với Remote	25
Hình 1.21: Sự phân biệt giữa các alias	26
Hình 1.22: Quy trình phát triển dự án microfrontend	27
Hình 1.23: Chia sẻ phụ thuộc giữa các ứng dụng với nhau	27
Hình 1.24: Quy trình chia sẻ module	28
Hình 1.25: Các phiên bản phụ thuộc ở các remote khác nhau	29
Hình 1.26: Các phiên bản phụ thuộc ở các remote khác nhau về phiên bản nhỏ (minor version)	29
Hình 1.27: Giao diện mẫu minh họa yêu cầu dẫn đến lựa chọn kiến trúc - Tổng quan	29
Hình 1.28: Nhiệm vụ của từng microfrontend	30
Hình 1.29: : Nhiệm vụ của từng microfrontend một cách tổng quan	30

Hình 1.30: Lưu ý quan trọng khi sử dụng microfrontend	31
Hình 1.31: Một số trường hợp khách quan được đặt ra khi sử dụng microfrontend	31
Hình 1.32: Những tiêu chí của việc không có sự liên kết giữa các dự án con với nhau	32
Hình 1.33: Nguy cơ tiềm ẩn khi có sự liên kết giữa các dự án con với nhau.....	33
Hình 1.34: Sự khác biệt về framework/thư viện	33
Hình 1.35: Minh họa không có sự kết hợp giữa container và các ứng dụng con	34
Hình 1.36: Minh họa kiểm soát phiên bản (monorepo so với separate) không nên có bất kỳ tác động nào đến toàn bộ dự án.....	35
Hình 1.37: Minh họa container phải có khả năng quyết định luôn sử dụng phiên bản mới nhất của microfrontend hoặc chỉ định một phiên bản cụ thể.....	36
Hình 1.38: Minh họa lí do không nên sử dụng Component liên Component cho từng microfrontend	37
Hình 1.39: Minh họa có sự thay đổi thư viện sẽ dẫn đến lỗi.....	37
Hình 1.40: Minh họa lí do nên sử dụng hàm mount.....	38
Hình 2.1: Use case tổng quan của ứng dụng đặt phòng khách sạn	44
Hình 2.2: Chú thích khung hình ảnh cho từng microfrontend	51
Hình 2.3: Wireframe trang đăng nhập/đăng ký	52
Hình 2.4: Wireframe trang chủ.....	53
Hình 2.5: Wireframe trang chi tiết khách sạn.....	54
Hình 2.6: Wire frame trang đặt phòng.....	55
Hình 2.7: Wireframe trang thanh toán.....	55
Hình 2.8: Wireframe trang thanh toán thành công	56
Hình 2.9: Minh họa giao diện trang Đăng nhập/Dăng ký và micofrontend	56
Hình 2.10: Minh họa giao diện trang Hồ sơ người dùng và micofrontend	57
Hình 2.11: Minh họa giao diện trang Trang chủ và micofrontend.....	58
Hình 2.12: Minh họa giao diện trang Chi tiết khách sạn và micofrontend	59
Hình 3.1: Biểu đồ biểu diễn mức độ phức tạp khi phát triển dự án so với tiến độ dự án	60

Mục lục bảng

Bảng 2.1: Yêu cầu chức năng của ứng dụng đặt phòng khách sạn	40
Bảng 2.2: Yêu cầu phi chức năng của ứng dụng đặt phòng khách sạn	42
Bảng 2.3: Use case đăng ký tài khoản.....	44
Bảng 2.4: Use case đăng nhập.....	45
Bảng 2.5: Use case tìm kiếm khách sạn	46
Bảng 2.6: Use case xem chi tiết khách sạn.....	46
Bảng 2.7: Use case đặt phòng khách sạn.....	47
Bảng 2.8: Use case thanh toán.....	47
Bảng 2.9: Use case quản lý đặt phòng.....	48
Bảng 2.10: Use case đánh giá khách sạn	49
Bảng 4.1: Kế hoạch thực hiện	63

Chương 1. Lý thuyết

I. Giới thiệu

1. Đặt vấn đề

Trong thời đại công nghệ số hiện nay, nhu cầu sử dụng các ứng dụng đặt phòng khách sạn ngày càng gia tăng, kéo theo sự cạnh tranh khốc liệt trong ngành du lịch và khách sạn. Việc phát triển các ứng dụng này không chỉ đơn thuần là cung cấp dịch vụ đặt phòng, mà còn phải đảm bảo trải nghiệm người dùng tốt nhất, tốc độ tải trang nhanh và khả năng mở rộng linh hoạt. Trong bối cảnh đó, kiến trúc Microfrontend đã nổi lên như một giải pháp tối ưu giúp các nhà phát triển ứng dụng giải quyết những thách thức này.

2. Mục tiêu của báo cáo

Mục tiêu của báo cáo này là trình bày rõ ràng về kiến trúc Microfrontend, các lợi ích mà nó mang lại trong phát triển ứng dụng đặt phòng khách sạn, cũng như các thách thức và giải pháp khi áp dụng kiến trúc này. Báo cáo cũng sẽ xem xét một số ví dụ thực tiễn từ các công ty đã thành công trong việc triển khai Microfrontend trong ứng dụng của họ.

3. Phạm vi nghiên cứu

Báo cáo sẽ tập trung vào việc phân tích và đánh giá kiến trúc Microfrontend trong bối cảnh phát triển ứng dụng đặt phòng khách sạn. Các vấn đề liên quan đến quá trình phát triển, triển khai, bảo trì và nâng cấp ứng dụng cũng sẽ được đề cập. Đồng thời, báo cáo cũng sẽ trình bày những xu hướng mới trong lĩnh vực công nghệ phần mềm có liên quan đến Microfrontend và ứng dụng đặt phòng khách sạn.

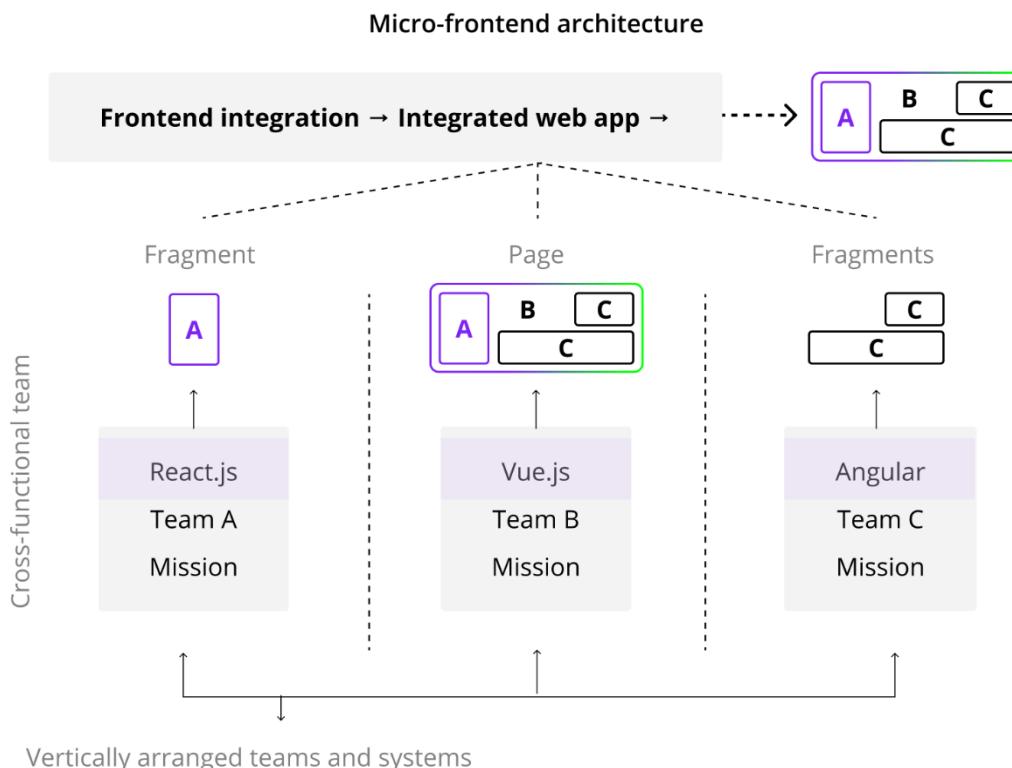
II. Tổng quan về kiến trúc Microfrontend

1. Phạm vi nghiên cứu

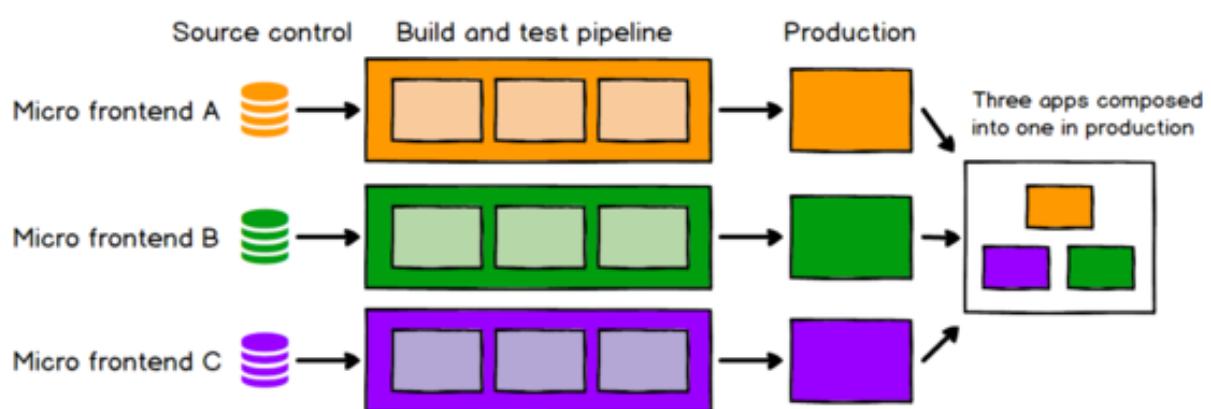
Microfrontend là một phong cách kiến trúc của phát triển web frontend, trong đó một ứng dụng được chia thành các tính năng và được phân phối độc lập. Mỗi microfrontend thường đại diện cho một tính năng hoặc một phần của giao diện người dùng, cho phép các nhóm phát triển làm việc song song mà không bị ảnh hưởng lẫn nhau. Điều này được thực hiện để cải thiện chất lượng phân phối và hiệu quả của các nhóm chịu trách nhiệm về code frontend.

Mỗi microfrontend có thể có kho lưu trữ source code, dependencies, automation tests động và pipeline riêng. Mỗi microfrontend rất có thể thuộc sở

hữu của một nhóm frontend duy nhất và được phát triển, thử nghiệm và triển khai độc lập với các nhiều tính năng, giúp tăng hiệu quả của quy trình phát triển.



Hình 1.1: Kiến trúc micro-frontend



Hình 1.2: Mỗi micro frontend được triển khai một cách độc lập

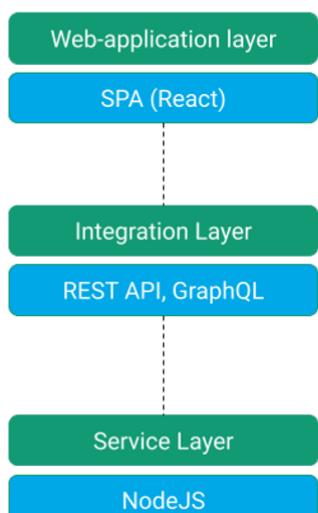
2. Lịch sử phát triển

Thuật ngữ “microfrontend” mới được giới thiệu khá gần đây – vào năm 2016, nó được Michael Geers sử dụng và vào cuối năm đó, nó xuất hiện trong ThoughtWorks Technology Radar. Tuy nhiên, các nhà phát triển frontend đã sử dụng phương pháp tiếp cận kiến trúc này trong một thời gian khá dài để hưởng lợi từ tính mô-đun, triển khai độc lập và khả năng mở rộng trong phát triển frontend.

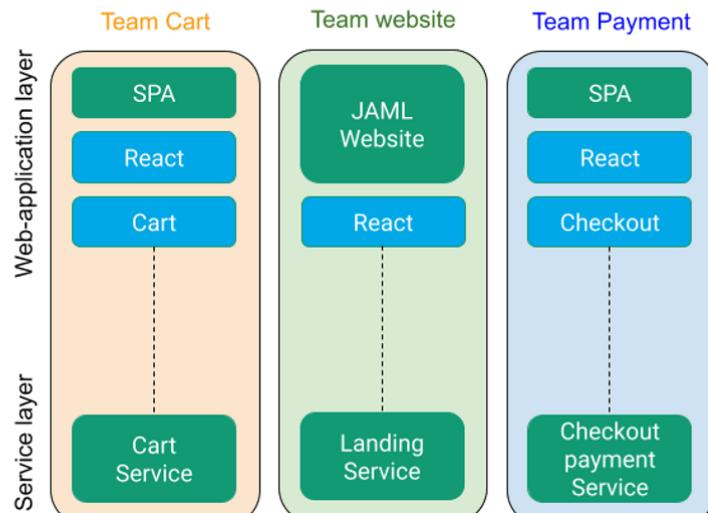
3. So sánh Microfrontend với kiến trúc truyền thống

- Kiến trúc truyền thống: Thường sử dụng một ứng dụng đơn nhất, trong đó tất cả các phần của ứng dụng được phát triển và triển khai cùng nhau. Điều này có thể dẫn đến việc khó khăn trong việc bảo trì, cập nhật và mở rộng ứng dụng.
- Microfrontend: Mỗi phần của ứng dụng được phát triển độc lập và có thể sử dụng công nghệ khác nhau. Điều này cho phép các nhóm phát triển tự do chọn công nghệ phù hợp nhất cho từng tính năng mà không làm ảnh hưởng đến các phần khác của ứng dụng.

Monolithic



Micro Frontends



Hình 1.3: Kiến trúc monolithic và micro-frontend

4. Các thành phần chính

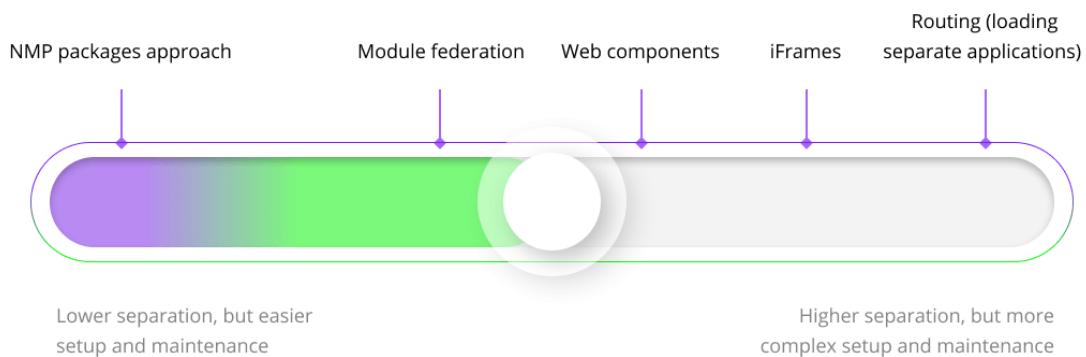
Microfrontend có thể được phân chia thành các thành phần sau:

- Tính năng (Feature): Mỗi microfrontend có thể đại diện cho một tính năng hoặc một phần của giao diện người dùng, như thanh tìm kiếm, giỏ hàng, hoặc trang chi tiết khách sạn.

- Giao tiếp: Các microfrontend cần có cách để giao tiếp với nhau, thường thông qua các API hoặc Event.
- Quản lý trạng thái (State Management): Do mỗi microfrontend có thể quản lý trạng thái của riêng nó, việc đồng bộ hóa trạng thái giữa các phần là rất quan trọng để đảm bảo trải nghiệm người dùng liên tục.
- Triển khai: Mỗi microfrontend có thể được triển khai độc lập, giúp cho việc cập nhật hoặc sửa lỗi trở nên dễ dàng và nhanh chóng hơn.

5. Các phương pháp tích hợp

Các cách tiếp cận tích hợp microfrontend 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 microfrontend hơn. Các cách tiếp cận này có thiết lập dễ dàng hơn nhiều – chúng yêu cầu một phiên bản framework microfrontend và một phiên bản thư viện. Giả sử ta di chuyển thanh trượt sang một bên khác có nhiều sự tách biệt hơn. Trong trường hợp đó, ứng dụng sẽ có độ phức tạp thiết lập và chi phí bảo trì lớn hơn, nhưng tất cả các lợi ích microfrontend đều có thể “tỏa sáng” hoàn toàn.



Hình 1.4: Thanh trượt các cách tiếp cận tích hợp microfrontend

Hãy cùng xem xét một số phương pháp tiếp cận phổ biến nhất.

a. Định tuyến: tải các ứng dụng riêng biệt

Ý tưởng đằng sau cách tiếp cận này là ứng dụng được tách biệt dựa trên định tuyến và mỗi tuyến load một ứng dụng hoàn toàn độc lập, bao gồm các package bên trong. Trong trường hợp này, cơ sở hạ tầng hoạt động như một shell và chuyển hướng giữa các ứng dụng xảy ra thông qua ví dụ như API gateway hoặc CDN.

Định tuyến có thể được tiếp cận bằng các liên kết HTML. Trong trường hợp như vậy, trang sẽ tải lại trang trên mỗi trang mỗi khi chuyển hướng đến một liên kết khác. Không có khả năng chia sẻ dữ liệu với cách tiếp cận này, chỉ có các query parameters, session storage hoặc data storage trong cơ sở dữ liệu.

Hãy cùng xem cách thức hoạt động theo góc nhìn của người dùng. Khi người dùng điều hướng đến một URL cụ thể trong ứng dụng, trung tâm ứng dụng định tuyến sẽ nhận được yêu cầu. Nó xác định microfrontend nào sẽ xử lý yêu cầu dựa trên URL. Microfrontend đã chọn sẽ được tải và hiển thị cho người dùng, trong khi các microfrontend khác vẫn ẩn.

Nếu cần phải hiển thị một trang mà không cần tải lại, có thể áp dụng một shared shell application hoặc một meta-framework như single-spa. Shell application bao gồm HTML, CSS và JavaScript tối thiểu. Người dùng sẽ thấy một trang được hiển thị tĩnh ngay lập tức ngay cả khi dữ liệu được yêu cầu vẫn đang chờ xử lý từ máy chủ.

Tất cả các tương tác và bản cập nhật đều được tải mà không cần tải chúng từ máy chủ, do đó trang không được làm mới theo yêu cầu của mỗi người dùng.

b. Phương pháp tiếp cận gói NPM

Mỗi microfrontend được phát hành dưới dạng một gói riêng lẻ với phương pháp này (NPM – Node Package Manager). Sau đó, ứng dụng shell sẽ liệt kê các gói này và kết hợp chúng vào ứng dụng tổng thể. Phương pháp này có lợi thế là thiết lập khá đơn giản. Tuy nhiên, nó đi kèm với một số nhược điểm nhất định, chẳng hạn như yêu cầu đối với các phiên bản phụ thuộc đơn lẻ và không thể phát hành microfrontend mà không có bản phát hành shell, điều này có thể tạo ra sự phụ thuộc giữa các nhóm và cản trở các bản cập nhật.

c. Microfrontend dựa trên iframe

Phương pháp iframe nhúng từng microfrontend vào iframe của riêng nó và một ứng dụng trang đơn (SPA – Single Page Application) hoạt động như một container. Về cơ bản, đây 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 phương pháp tương đối cũ để xây dựng kiến trúc microfrontend. Mặc dù nó cung cấp mức độ tách biệt tốt, nhưng cần phải giải quyết nhiều vấn đề để nó hoạt động (ví dụ: bảo mật, chia sẻ dữ liệu).

d. WebComponents

Các thành phần web ngụ ý xây dựng từng microfrontend như một component riêng biệt có thể được triển khai độc lập dưới dạng tệp .js. Ứng dụng tải và hiển thị chúng trong các placeholders được tạo riêng trong bộ cục. Các Web Components cho trình duyệt biết khi nào và ở đâu để tạo component.

Nhược điểm của cách tiếp cận này là mỗi Web Component đều lớn và cần phải đóng gói phiên bản framework của nó. Vì vậy, mặc dù ta có thể tách nhiều component tùy ý thì hiệu suất phải được theo dõi cẩn thận.

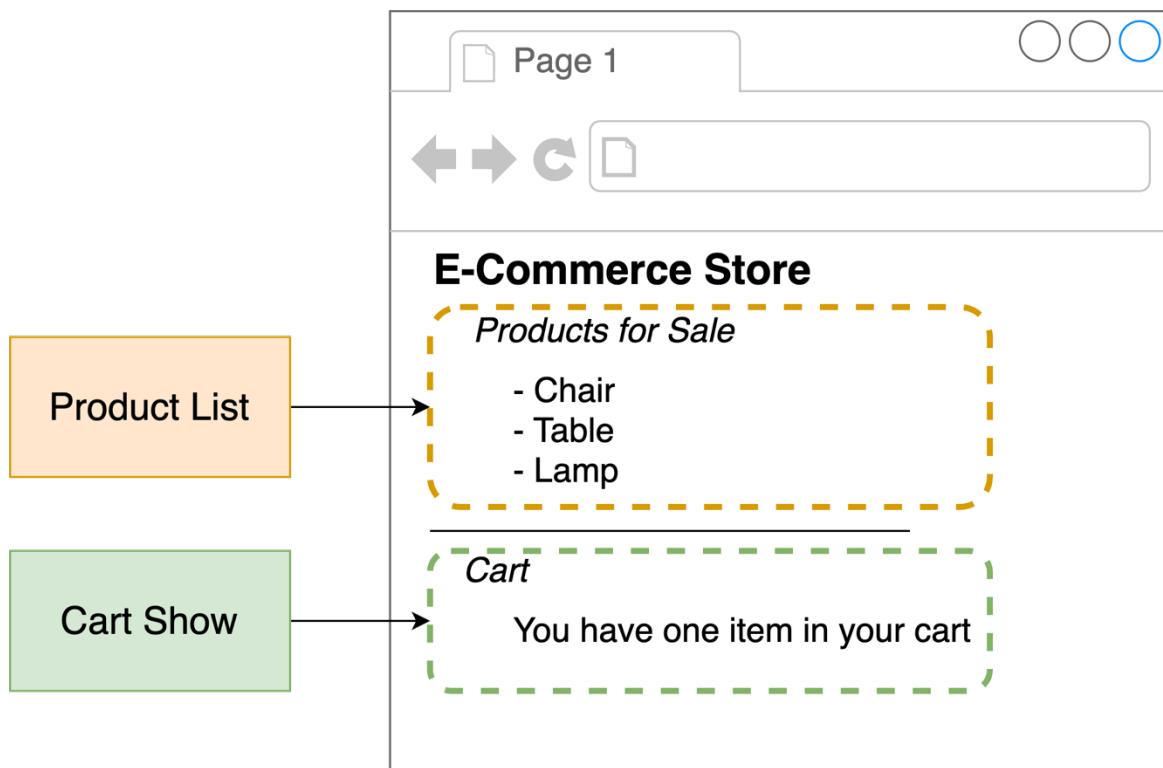
e. Module Federation

Module Federation là một cách tiếp cận khác giúp làm cho tất cả các thành phần microfrontend trông giống như một ứng dụng. Trên thanh trượt của các mức độ phân tách khác nhau được sử dụng trong microfrontend, module federation có thể được đặt ở đâu đó ở giữa. Module federation được phát hành cùng với Webpack 5 (2020), vì vậy nó hiện đại và có thể được các khuôn khổ khác nhau áp dụng.

Module Federation cho phép tải microfrontend khi chạy vào ứng dụng shell mà không phụ thuộc vào thời gian build. Điều này đặc biệt hữu ích khi tùy chỉnh các trang của ta cho các client cụ thể. Trang web của ta có thể có một mô-đun “thanh toán” được tải từ một gói “chung” cho hầu hết các client và một mô-đun “tùy chỉnh” (thậm chí có thể do nhóm phát triển của client xây dựng) cho những client khác. Module federation cho phép ta thay đổi cấu hình microfrontend của mình khi chạy.

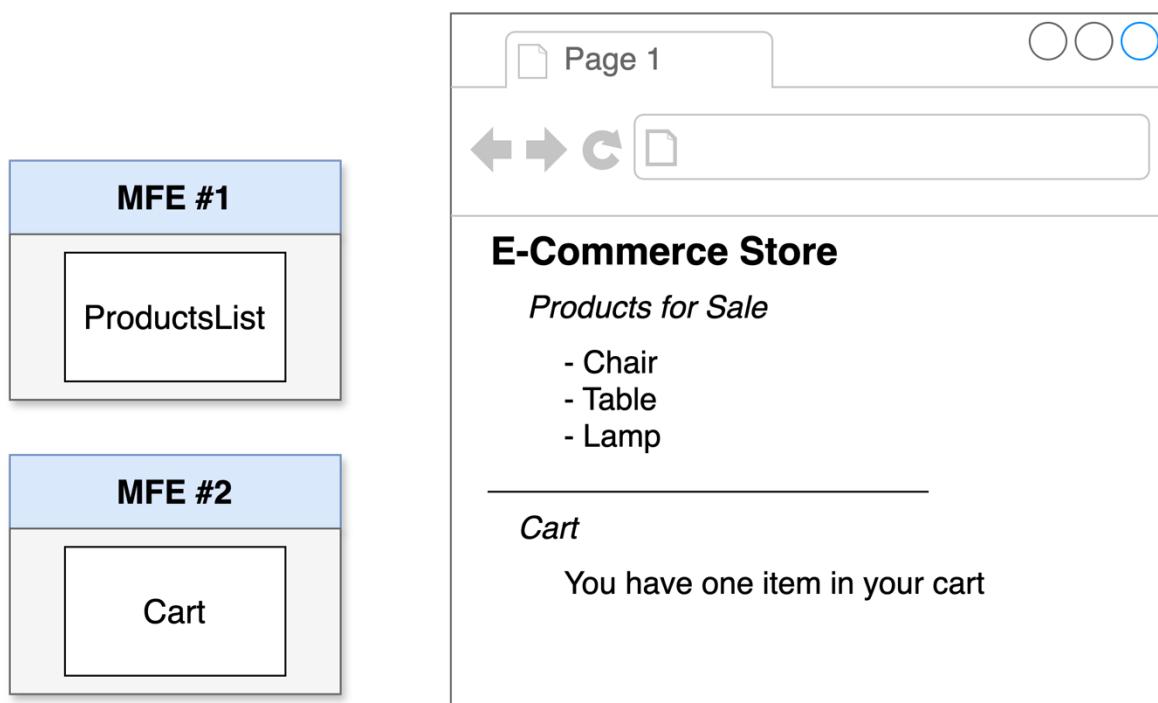
6. Ví dụ

Giả sử chúng ta có một trang thương mại điện tử với 2 thành phần là danh sách sản phẩm và giỏ hàng:



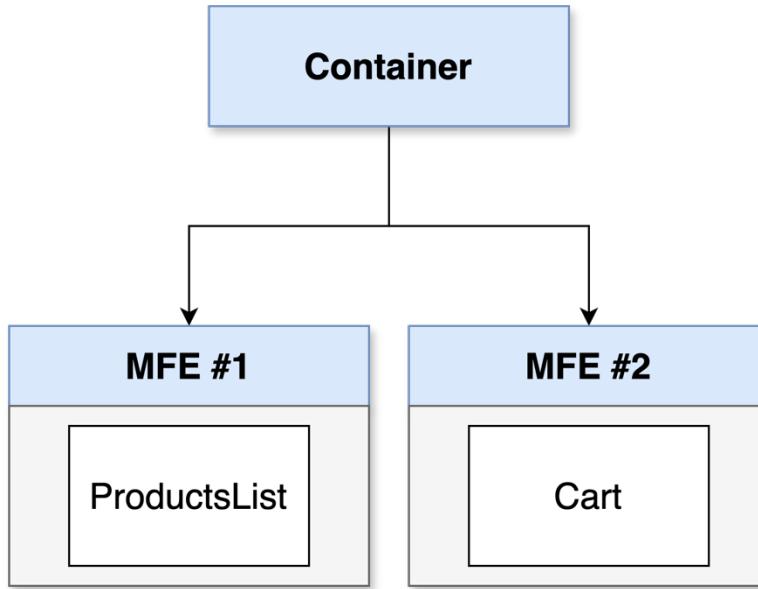
Hình 1.5: Giả sử 2 thành phần trong thương mại điện tử

Ta sẽ chia ra được 2 microfrontend cho 2 thành phần này:



Hình 1.6: Mô hình hoá microfrontend cho 2 thành phần trên

Nhìn một cách tổng quan, nó sẽ được biểu diễn như sau:



Hình 1.7: Mô hình tổng quan của 2 microfrontend trên

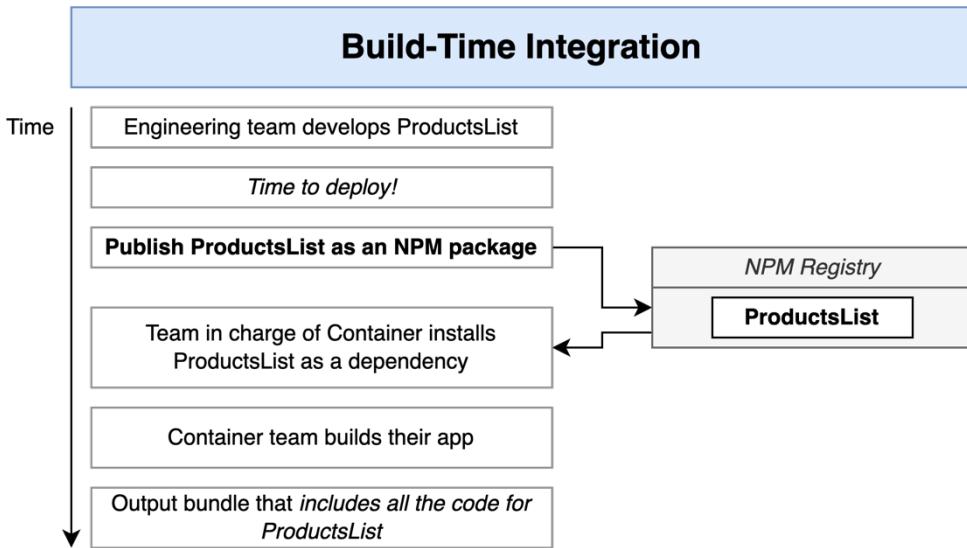
Nhưng vấn đề đặt ra là: **Container có thể truy cập vào mã nguồn trong MFE #1 và #2 khi nào và bằng cách nào?**

Sẽ có 3 phương pháp chính để Container có thể truy cập vào các MFE này, và nó được gọi là các loại tích hợp. Sẽ được làm rõ ở phần tiếp theo.

7. Các loại tích hợp chính

a. Build-time integration

Trước khi Container được tải trong trình duyệt, nó sẽ được truy cập vào mã nguồn ProductsList.

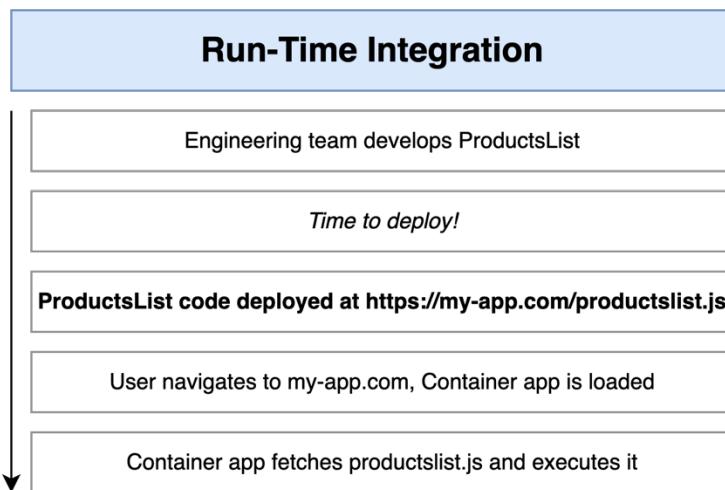


Hình 1.8: Các loại tích hợp chính – Tích hợp build-time

Với cách tiếp cận này, sẽ có lợi ích là dễ cài đặt và dễ hiểu, nhưng Container phải được triển khai lại mỗi khi `ProductsList` được cập nhật và có ý định (cám dỗ) kết hợp chặt chẽ Container + `ProductsList` với nhau.

b. Run-time integration

Sau khi Container được tải trong trình duyệt, nó sẽ được truy cập vào mã nguồn `ProductsList`.



Hình 1.9: Các loại tích hợp chính – Tích hợp runtime

Với cách tiếp cận này, `ProductList` có thể được độc lập bất cứ lúc nào và có thể triển khai nhiều phiên bản khác nhau của `ProductsList` và Container có thể

quyết định sử dụng phiên bản nào, nhưng công cụ và thiết lập phức tạp hơn nhiều.

c. Server integration

Trong khi gửi JS xuống để tải Container, máy chủ quyết định có bao gồm nguồn ProductsList hay không.

d. Lưu ý

Trong phạm vi đề tài này, chúng ta sẽ chỉ tập trung vào Run-time Integration sử dụng Webpack Module Federation.

III. Áp dụng Webpack Module Federation xây dựng Microfrontend cho ứng dụng đặt phòng khách sạn

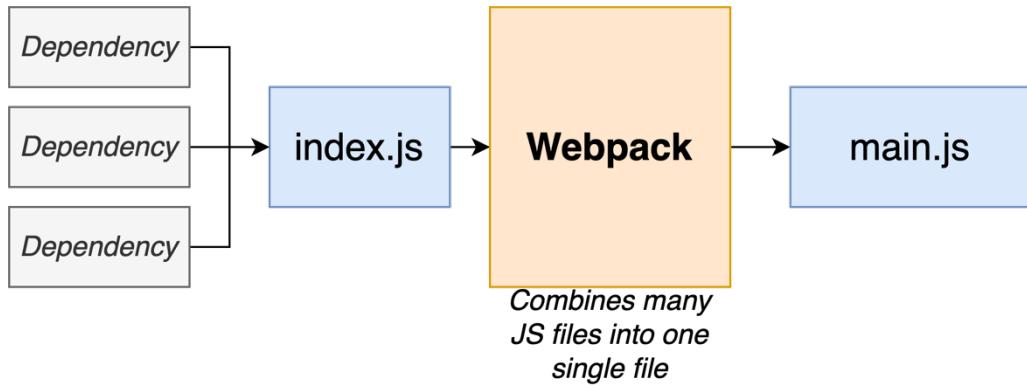
1. Sơ lược về Webpack

Webpack là một trình đóng gói mô-đun mã nguồn mở và miễn phí cho JavaScript. Nó được tạo chủ yếu cho JavaScript, nhưng nó có thể chuyển đổi các tài sản giao diện người dùng như HTML, CSS và hình ảnh nếu các trình tải tương ứng được bao gồm. Webpack lấy các mô-đun có phụ thuộc và tạo các tài sản tĩnh biểu diễn các mô-đun đó.

Webpack lấy các phụ thuộc và tạo biểu đồ phụ thuộc cho phép các nhà phát triển web sử dụng phương pháp tiếp cận mô-đun cho mục đích phát triển ứng dụng web của họ. Nó có thể được sử dụng từ dòng lệnh hoặc có thể được định cấu hình bằng tệp cấu hình có tên là webpack.config.js. Tệp này định nghĩa các quy tắc, plugin, v.v. cho một dự án. (Webpack có khả năng mở rộng cao thông qua các quy tắc cho phép các nhà phát triển viết các tác vụ tùy chỉnh mà họ muốn thực hiện khi đóng gói các tệp lại với nhau.)

Node.js là bắt buộc để sử dụng Webpack.

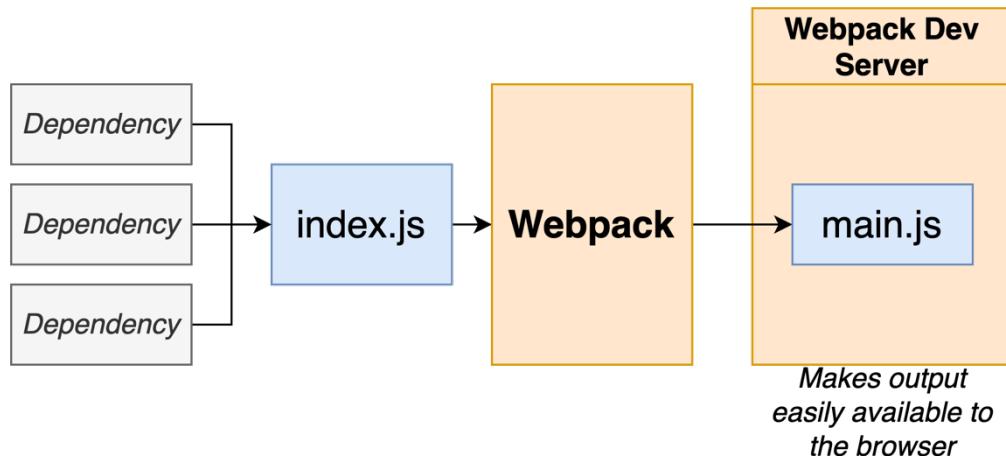
Webpack cung cấp mã theo yêu cầu bằng cách sử dụng biệt danh là **code-splitting**. Hai kỹ thuật tương tự được Webpack hỗ trợ khi nói đến dynamic code splitting. Phương pháp đầu tiên và được khuyến nghị là sử dụng cú pháp import() tuân thủ đề xuất ECMAScript cho các lần nhập động. Phương pháp cũ dành riêng cho Webpack là sử dụng require.ensure.



Hình 1.10: Cách thức hoạt động của Webpack

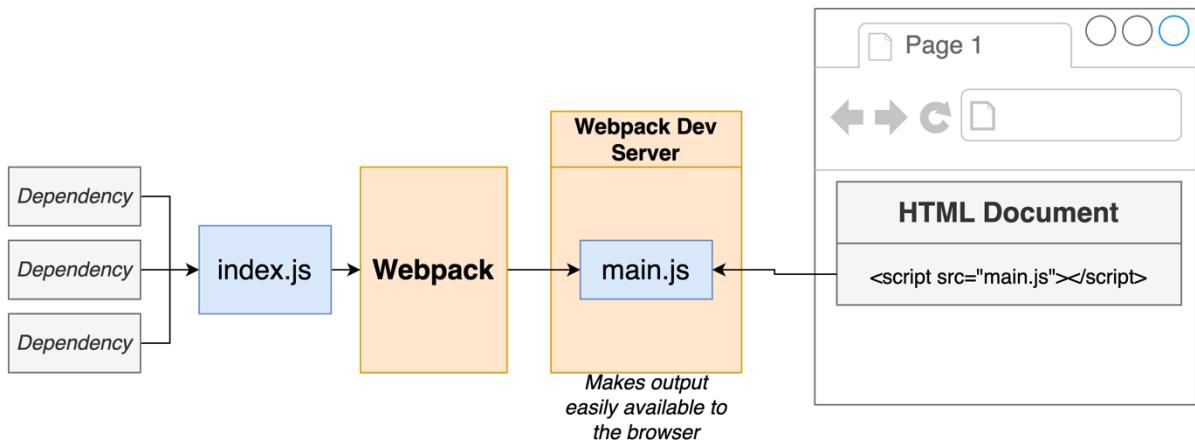
2. Webpack development server

Webpack cũng cung cấp một máy chủ phát triển tích hợp, webpack-dev-server, có thể được sử dụng như một máy chủ HTTP để phục vụ các tệp trong khi phát triển. Nó cũng cung cấp khả năng sử dụng hot module replacement (HMR), cập nhật mã trên trang web mà không yêu cầu nhà phát triển tải lại trang.



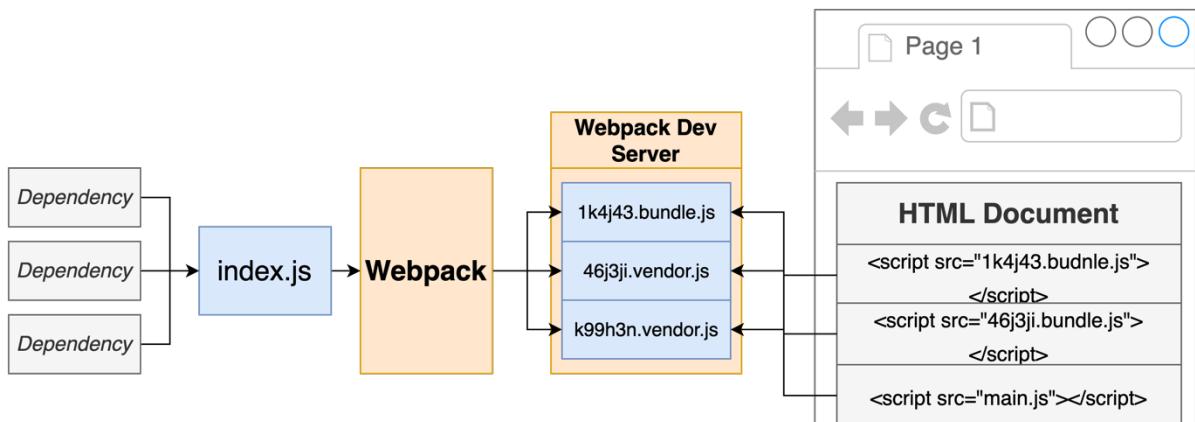
Hình 1.11: Cách thức hoạt động của Webpack development server

Sau khi build webpack dev server sẽ tạo ra một file main.js (main bundle).



Hình 1.12: Bundle có trên trình duyệt sau khi đi qua Webpack development server

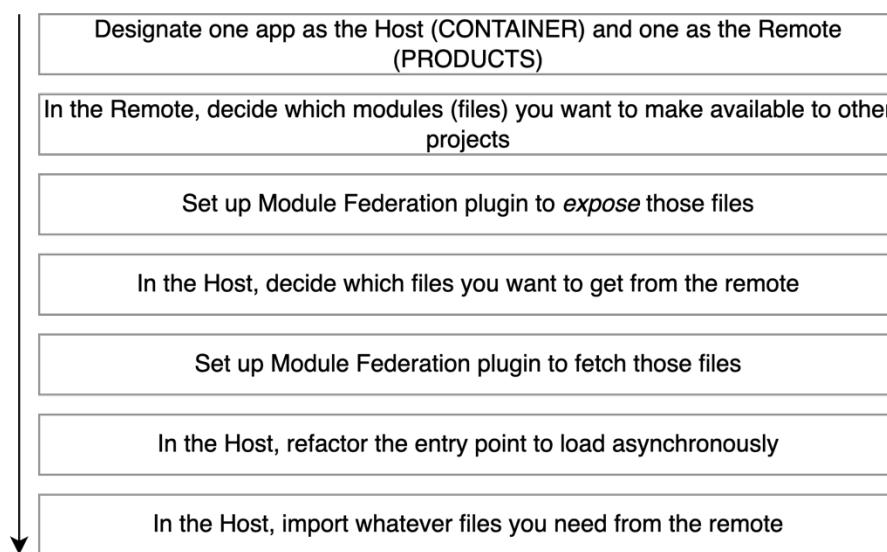
Các phần tử HTML có thể được xử lý logic thông qua các file js đã được build (bundle) này.



Hình 1.13: Webpack development server chia nhỏ bundle thành các chunk

3. Cài đặt Module Federation

a. Các bước để cài đặt:



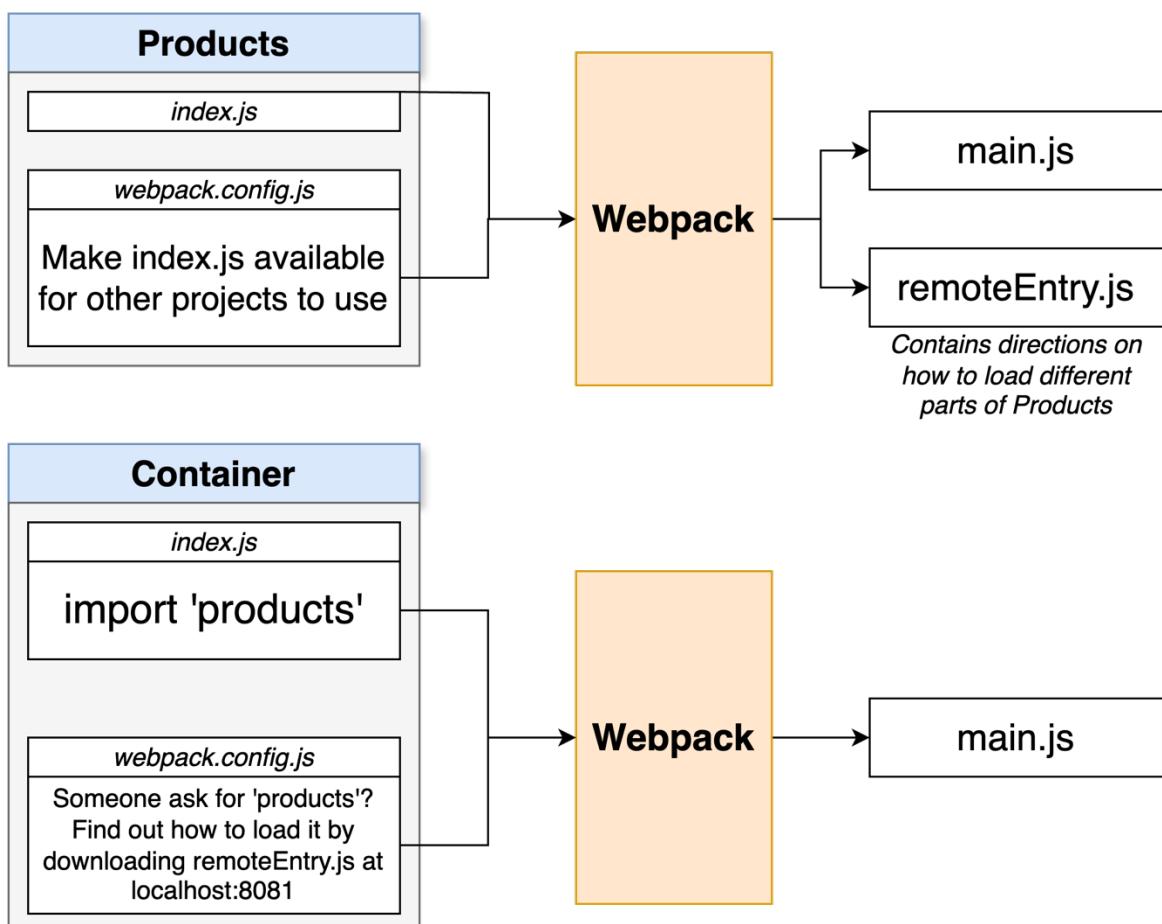
Hình 1.14: Các bước cài đặt Module Federation

- Chỉ định Host (CONTAINER) và Remote (PRODUCTS): Chọn một ứng dụng sẽ đóng vai trò là "Host" - ứng dụng chính chịu trách nhiệm tải các module từ ứng dụng khác, và một ứng dụng khác sẽ là "Remote" - ứng dụng cung cấp các module mà Host có thể sử dụng.
- Xác định các module trong Remote: Trong ứng dụng Remote, ta cần xác định rõ các module (tệp JavaScript hoặc các thành phần) mà ta muốn chia sẻ, cho phép các ứng dụng khác truy cập và sử dụng.
- Thiết lập Module Federation trong Remote: Cấu hình plugin Module Federation trong ứng dụng Remote để "xuất" các module mà ta đã xác định. Điều này sẽ làm cho các module có sẵn để các ứng dụng khác có thể truy cập vào.
- Xác định các module cần dùng trong Host: Trong ứng dụng Host, quyết định rõ ràng các module mà ta muốn lấy từ ứng dụng Remote. Điều này có thể bao gồm các thành phần, thư viện, hoặc bất kỳ phần mã nào ta cần sử dụng từ Remote.
- Cấu hình Module Federation trong Host: Thiết lập plugin Module Federation trong ứng dụng Host để "lấy" các module từ ứng dụng Remote. Plugin này sẽ quản lý việc liên kết và tải các module từ Remote vào Host.

- Điều chỉnh cách tải trong Host để bất đồng bộ: Trong ứng dụng Host, cần cấu hình lại điểm nhập (entry point) để tải các module từ Remote theo cách bất đồng bộ (asynchronously). Điều này giúp tối ưu hiệu suất và đảm bảo các module được tải khi cần thiết mà không làm chậm ứng dụng.
- Nhập các module từ Remote vào Host: Khi đã cấu hình xong, ta có thể nhập các module từ ứng dụng Remote vào ứng dụng Host như thể chúng là các phần của ứng dụng chính. Điều này giúp Host sử dụng trực tiếp các thành phần hoặc chức năng từ Remote mà không cần sao chép mã.

b. Cách sử dụng Module Federation trong Webpack với hai ứng dụng:

"Products" (Remote) và "Container" (Host):



Hình 1.15: Cách sử dụng Module Federation trong Webpack với hai ứng dụng Products và Container

Products (Remote):

- `index.js`: Đây là file chính mà ta muốn cung cấp cho các dự án khác để sử dụng.

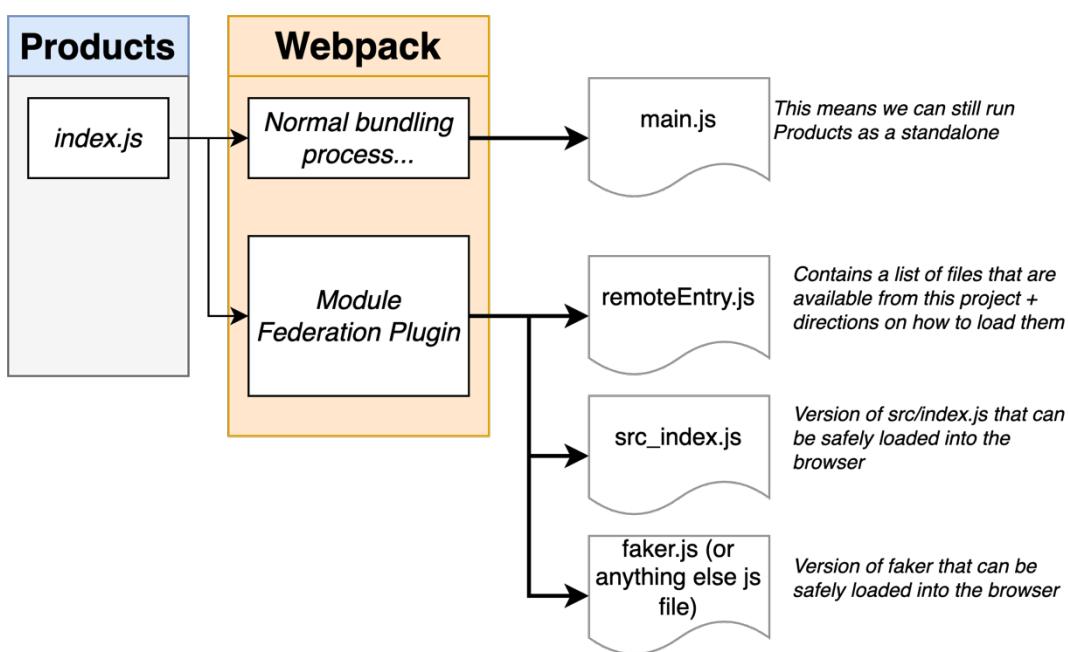
- webpack.config.js: Cấu hình này sẽ thiết lập Module Federation để làm cho index.js khả dụng cho các ứng dụng khác. Cấu hình này sẽ xuất một file remoteEntry.js, chứa thông tin cần thiết để ứng dụng khác (Container) có thể tải các module từ đây.
- Kết quả: Sau khi chạy Webpack, ta sẽ có hai file: main.js (mã chính của ứng dụng) và remoteEntry.js (chứa các thông tin cần thiết cho việc chia sẻ module).

Container (Host):

- index.js: Đây là nơi mà ta muốn nhập và sử dụng các module từ ứng dụng "Products".
- webpack.config.js: Cấu hình này sẽ thiết lập Module Federation để nhập các module từ Products. Nó chứa các thông tin để tìm và tải remoteEntry.js từ "Products" thông qua một URL (ví dụ: localhost:8081). Khi cần sử dụng products, ứng dụng sẽ tự động tải module từ remoteEntry.js.
- Kết quả: Sau khi chạy Webpack, ta sẽ có file main.js chứa mã của ứng dụng Container, cùng với cơ chế tải các module từ Remote.

4. Hiểu hơn về Module Federation

a. Quy trình đóng gói (bundling) của Webpack khi sử dụng Module Federation cho ứng dụng "Products" (Remote):



Hình 1.16: Quy trình đóng gói của Webpack khi sử dụng Module Federation cho ứng dụng Products

Products (Remote):

- `index.js`: Đây là file nguồn chính mà ta muốn chia sẻ cho các ứng dụng khác thông qua Module Federation.

Webpack:

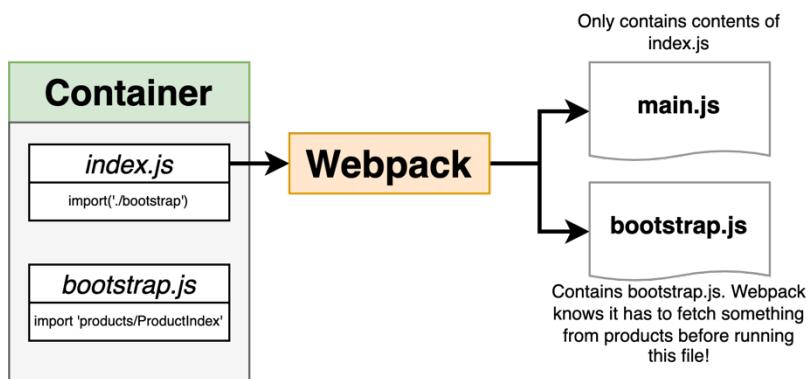
- **Normal bundling process**: Webpack thực hiện quá trình đóng gói bình thường cho toàn bộ mã nguồn của ứng dụng. Điều này sẽ tạo ra một file bundle chính, ví dụ như `main.js`.
- **Module Federation Plugin**: Plugin này được thêm vào cấu hình Webpack để làm cho một số module có thể được chia sẻ và truy cập từ các ứng dụng khác. Nó tạo ra một file đặc biệt, ví dụ như `remoteEntry.js`, chứa thông tin về các module mà ứng dụng "Products" cung cấp.

Kết quả sau khi chạy Webpack:

- `main.js`: File chính chứa mã đã được đóng gói của ứng dụng.

- remoteEntry.js: File chứa metadata và các thông tin cần thiết để các ứng dụng khác có thể tải các module từ "Products". Đây là file quan trọng nhất trong cơ chế chia sẻ module.
- src_index.js, faker.js (hoặc bất kỳ file JavaScript nào khác): Các file này đại diện cho các phân mảnh khác mà "Products" có thể sử dụng hoặc chia sẻ. Tất cả sẽ được quản lý bởi Webpack.

b. Quá trình sử dụng Webpack để gộp và đóng gói các tệp JavaScript từ các thành phần khác nhau



Hình 1.17: Quá trình sử dụng Webpack để gộp và đóng gói các tệp Javascript từ các thành phần khác nhau

Container: Đại diện cho thư mục chứa các tệp mã nguồn chính, gồm hai tệp:

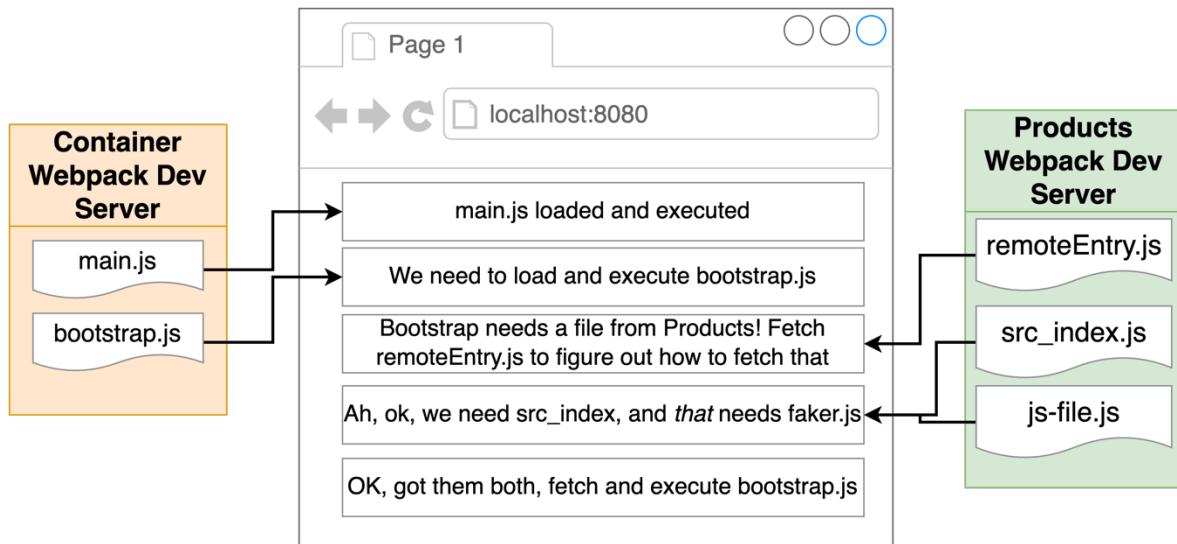
- index.js: Tệp này nhập (import) các module khác, chẳng hạn như ./bootstrap.
- bootstrap.js: Tệp này nhập thêm một module khác, ví dụ như products/ProductIndex.

Webpack: Công cụ này được sử dụng để đóng gói (bundle) các tệp JavaScript từ Container thành các tệp độc lập:

- main.js
- bootstrap.js

Cả hai tệp này sau đó có thể được sử dụng bởi ứng dụng web, giúp tối ưu hóa quá trình quản lý mã nguồn và tăng hiệu suất khi triển khai.

c. Quá trình tải và thực thi các tệp JavaScript giữa hai Webpack Dev Server khác nhau



Hình 1.18: Quá trình tải và thực thi các tệp JavaScript giữa hai Webpack Dev Server khác nhau

Container Webpack Dev Server (bên trái) chứa hai tệp:

- `main.js`
- `bootstrap.js`

Products Webpack Dev Server (bên phải) chứa ba tệp:

- `remoteEntry.js`
- `src_index.js`
- `js-file.js`

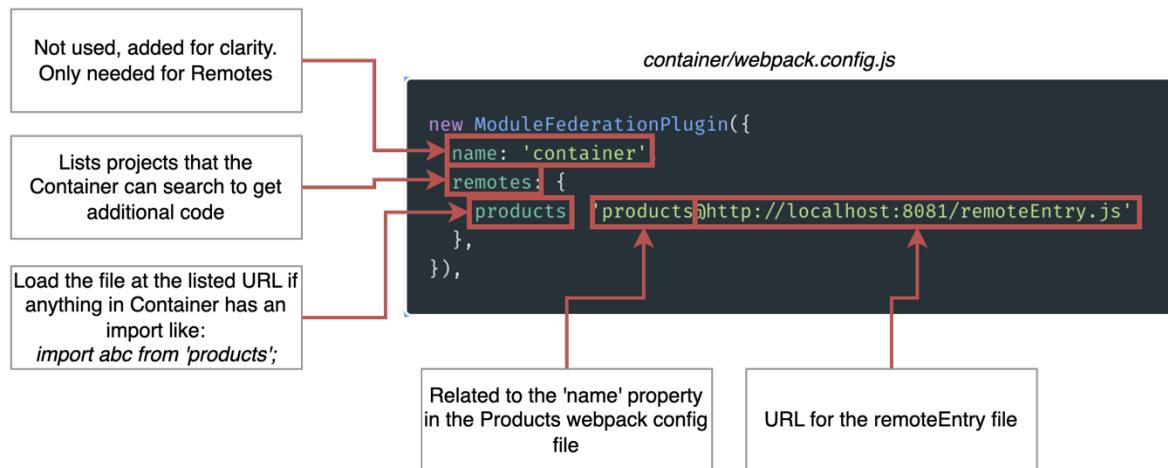
Quá trình này diễn ra như sau:

- Trình duyệt truy cập trang web tại `localhost:8080`.
- `main.js` được tải và thực thi trước.
- Sau đó, `main.js` yêu cầu tải và thực thi `bootstrap.js`.
- Tuy nhiên, `bootstrap.js` cần một tệp từ Products. Do đó, nó gửi yêu cầu tải `remoteEntry.js` từ Products Webpack Dev Server để biết cách tải các tệp khác.
- Sau khi `remoteEntry.js` được tải, nó cho biết rằng cần tải `src_index.js`, và `src_index.js` cũng yêu cầu thêm `faker.js`.
- Cuối cùng, khi tất cả các tệp cần thiết đã được tải, `bootstrap.js` được thực thi.

Quá trình này là một ví dụ về việc chia sẻ module giữa các hệ thống khác nhau trong một ứng dụng sử dụng Webpack Module Federation, giúp quản lý và chia sẻ các phần của mã nguồn giữa các ứng dụng hoặc dự án độc lập.

5. Hiểu hơn về các tùy chọn cấu hình

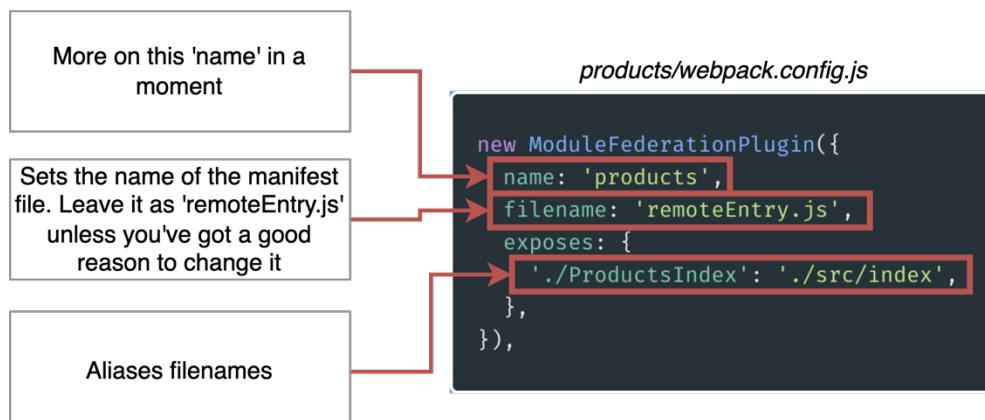
a. Đối với Host:



Hình 1.19: Tùy chọn cấu hình đối với Host

- Không cần phải quá quan tâm về name.
- remotes: Danh sách dự án mà Host có thể tìm để lấy mã.
- remotes.products: tải tệp ở URL được liệt kê nếu mọi thứ trong Host có một import kiểu `import abc from 'products'`. Cái tên products này chính là name của Remote.

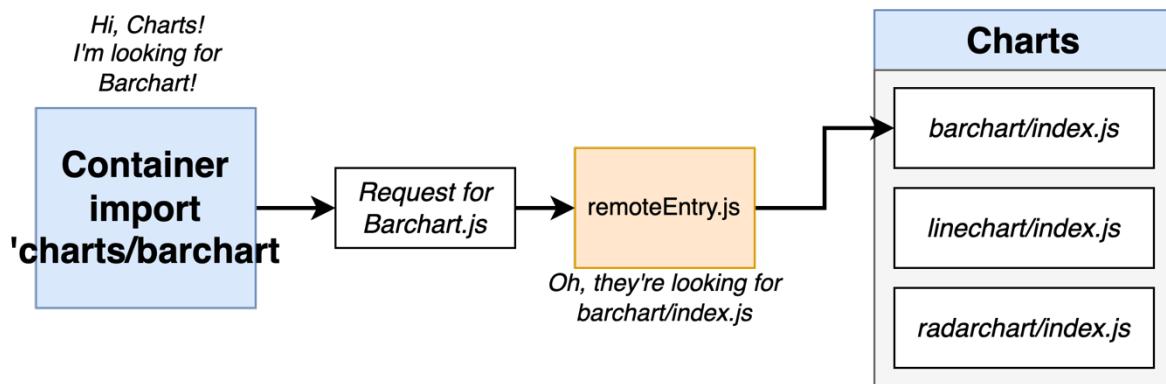
b. Đối với Remote:



Hình 1.20: Tùy chọn cấu hình đối với Remote

- name: đây là một thuộc tính vô cùng quan trọng vì nó quyết định tên remote mà cần được liệt kê ở Host.
- filename: đây là tên file manifest. Mặc định nên để là remoteEntry.js. Đây sẽ là file mà host sẽ cần phải liệt kê và tìm tới.
- exposes: liệt kê danh sách các thành phần cần được lấy ra trong filename. Và đặt cho chúng những alias phù hợp.

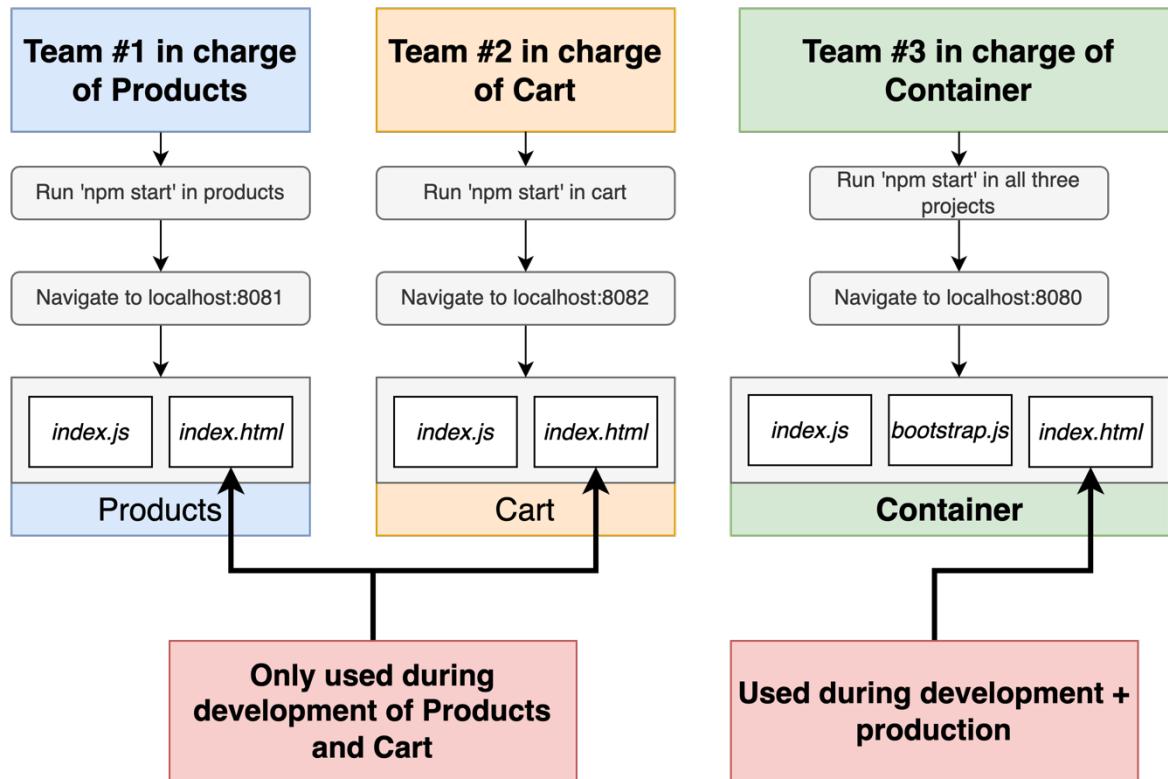
Tại sao nên có sự phân biệt giữa các alias?



Hình 1.21: Sự phân biệt giữa các alias

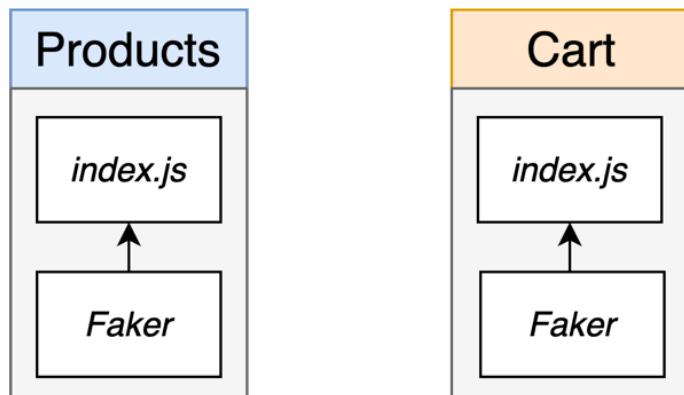
Tạo ra sự phân biệt đó để Host có thể tìm kiếm đúng và chính xác thành phần cần sử dụng từ Remote.

IV. Quy trình phát triển



Hình 1.22: Quy trình phát triển dự án microfrontend

1. Chia sẻ các phụ thuộc giữa các ứng dụng (sub-project)

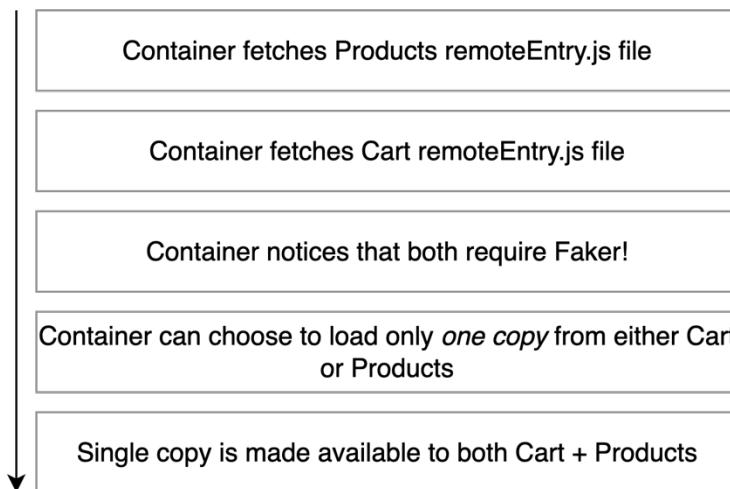


Both Products + Cart need the faker module

Hình 1.23: Chia sẻ phụ thuộc giữa các ứng dụng với nhau

Trong quá trình phát triển, giữa các team sẽ cài các phụ thuộc cho app của mình và sẽ xảy ra trường hợp các phụ thuộc giống nhau (về tên và phiên bản). Việc để trình duyệt tải cả các phụ thuộc tuy giống nhau nhưng mỗi phụ thuộc là một lần tải này sẽ làm giảm tốc độ tải trang (hiệu suất) và tốn tài nguyên. Chính vì thế cần Webpack cung cấp một cơ chế để chia sẻ các phụ thuộc này với nhau.

Quy trình chia sẻ module sẽ được diễn ra như sau:

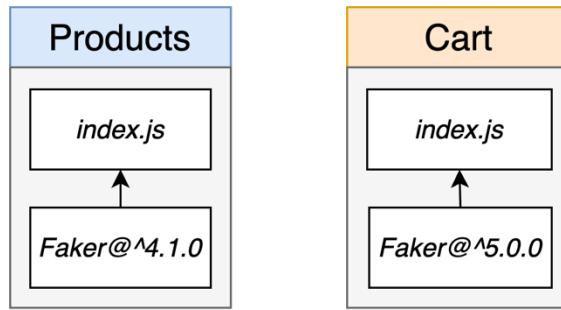


Hình 1.24: Quy trình chia sẻ module

- Host sẽ fetch remoteEntry.js file từ các Remote.
- Host nhận ra các Remote dùng một loại phụ thuộc giống nhau.
- Host có thể chọn để tải một bản sao chép từ phụ thuộc đó của các Remote (hoặc).
- Bản sao chép này sẽ được làm cho có sẵn ở cả các Remote.

Vậy nếu các phụ thuộc tuy giống nhau về tên, nhưng có sự phân biệt về version thì sao? Nó sẽ có các vấn đề xảy ra như sau:

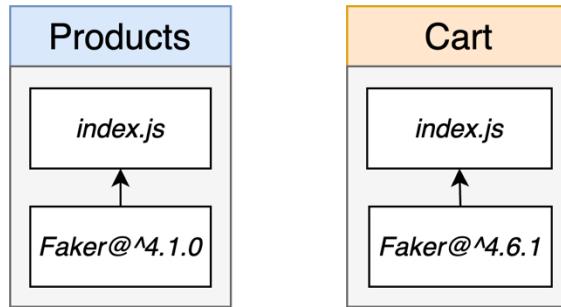
- Các phiên bản phụ thuộc ở các Remote khác nhau:



Hình 1.25: Các phiên bản phụ thuộc ở các remote khác nhau

Trong trường hợp này, Webpack sẽ tự động load cả 2 module của 2 phụ thuộc có 2 phiên bản khác nhau này.

- Các phiên bản phụ thuộc ở các Remote khác nhau về phiên bản nhỏ:



Hình 1.26: Các phiên bản phụ thuộc ở các remote khác nhau về phiên bản nhỏ (minor version)

Trong trường hợp này, Webpack vẫn sẽ load một bản copy của bản phụ thuộc đó duy nhất.

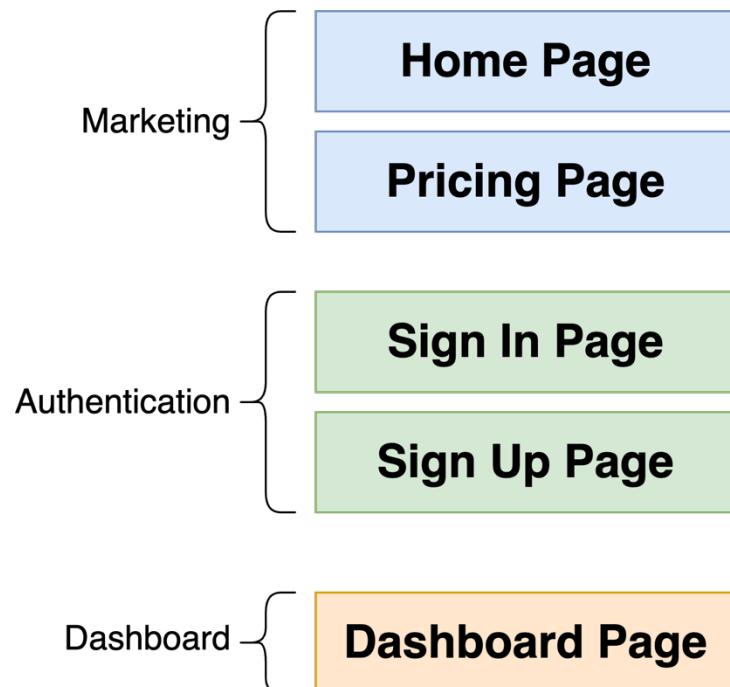
2. Yêu cầu dẫn đến lựa chọn kiến trúc

Giả sử ta có giao diện như sau:



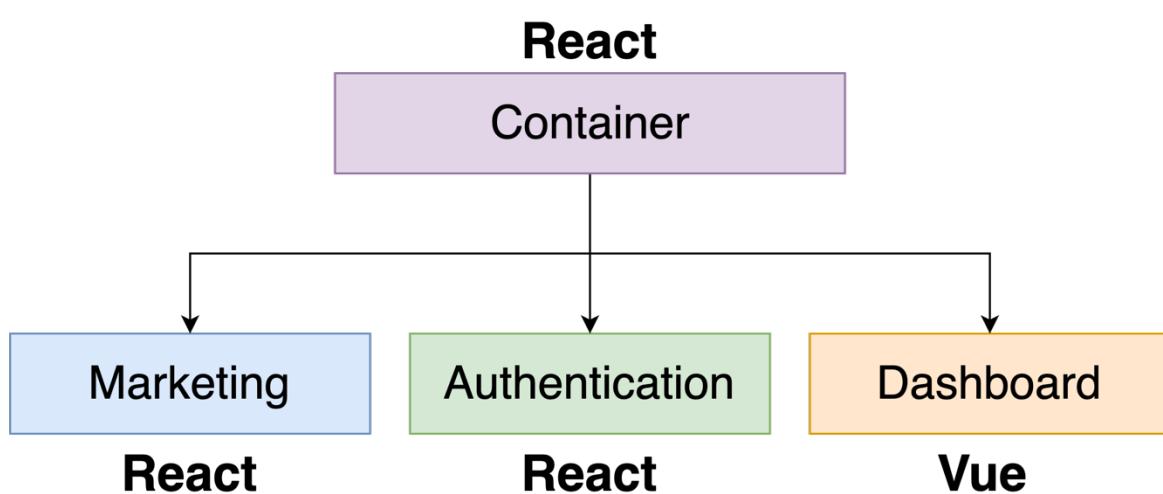
Hình 1.27: Giao diện mẫu minh họa yêu cầu dẫn đến lựa chọn kiến trúc - Tổng quan

Giao diện sẽ được chia thành 3 microfrontend chính:



Hình 1.28: Nhiệm vụ của từng microfrontend

Một cách tổng quan:



Hình 1.29: : Nhiệm vụ của từng microfrontend một cách tổng quan

Huge Disclaimer

Some blog posts, articles, videos, etc will tell you to do things differently



The architecture for this project is determined by its *requirements*



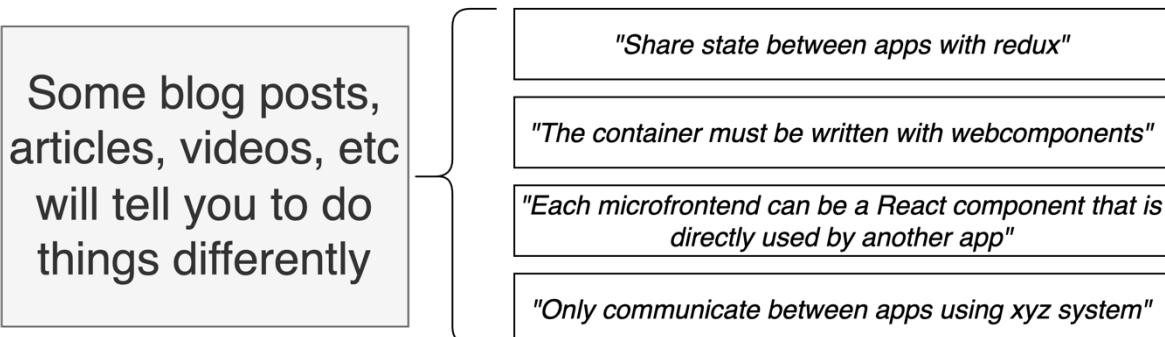
You need to think about the requirements of your app to decide if this architecture will work for you

Hình 1.30: Lưu ý quan trọng khi sử dụng microfrontend

Những lưu ý quan trọng khi sử dụng microfrontend:

- Một số bài viết của blog, videos, v.v... sẽ cho ta biết cách làm mọi thứ khác biệt.
- Kiến trúc cho dự án này được xác định bởi chính yêu cầu của dự án.
- Ta cần nghĩ về những yêu cầu của ứng dụng để xác định xem nếu như kiến trúc này sẽ hoạt động một cách hiệu quả nhất.

Examples



Hình 1.31: Một số trường hợp khách quan được đặt ra khi sử dụng microfrontend

Ví dụ:

- Chia sẻ trạng thái giữa các ứng dụng với redux.
- Container phải được viết bằng việc sử dụng web components.
- Mỗi microfrontend có thể là một React component mà trực tiếp được sử dụng bởi microfrontend khác.
- Chỉ giao tiếp giữa các ứng dụng bằng một hệ thống nào đó.

a. Không có sự liên kết giữa các dự án con

Inflexible Requirement #1

Zero coupling between child projects

No importing of functions/objects/classes/etc

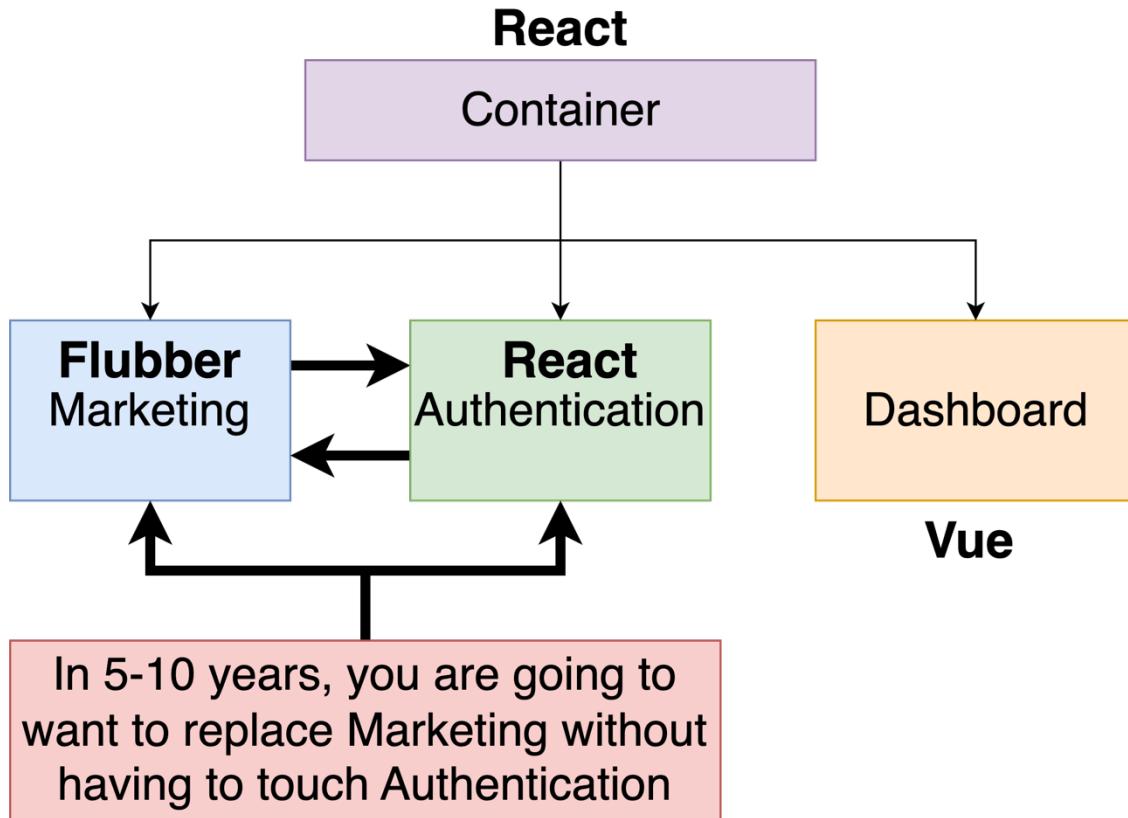
No shared state

Shared libraries through MF is ok

Hình 1.32: Những tiêu chí của việc không có sự liên kết giữa các dự án con với nhau

Yêu cầu không thể thay đổi đầu tiên – Không có sự ghép nối giữa các ứng dụng con:

- Không có sự truy xuất của các hàm, đối tượng, lớp, v.v...
- Không chia sẻ trạng thái.
- Chia sẻ thư viện giữa các microfrontend thì vẫn ổn.



Hình 1.33: Nguy cơ tiềm ẩn khi có sự liên kết giữa các dự án con với nhau

Trong 5-10 năm tới, ta sẽ có ý định muốn thay đổi Marketing mà không can thiệp vào Authentication.



Hình 1.34: Sự khác biệt về framework/thư viện

Và có khi chúng ta cũng sẽ sau cùng quên đi cách mà React - tổng quát hơn là bất cứ thư viện/framework nào đó hoạt động.

b. Gần như không có sự kết hợp giữa container và các ứng dụng con

Inflexible Requirement #2

Near-zero coupling between container and child apps

Container shouldn't assume that a child is using a particular framework

Any necessary communication done with callbacks or simple events

Hình 1.35: Minh họa không có sự kết hợp giữa container và các ứng dụng con

Yêu cầu không thể thay đổi thứ hai – Không có sự ghép nối giữa container và các ứng dụng con:

- Container không nên cho rằng là ứng dụng con sẽ sử dụng một framework cụ thể.
- Bất kỳ sự giao tiếp cấp thiết nào hoàn thành với callbacks hoặc sự kiện đơn giản.

c. CSS từ một dự án không nên ảnh hưởng đến dự án khác

Trong nhiều trường hợp, các dự án khi gộp lại với nhau sẽ xảy ra xung đột về các CSS Class, Selector,... Điều này sẽ gây ra khá nhiều vấn đề vì gần như việc đặt tên các Class, Selector là không thể đoán trước được. Chính vì vậy, có một số giải pháp và đặc biệt là giải pháp sử dụng CSS module để có thể đặt tên tuy giống nhau về mặt lý thuyết nhưng khi triển khai lên giao diện thì các Class, Selector đó sẽ không hề giống nhau, chúng sẽ được “mã hóa” một cách độc lập với các Class, Selector khác.

d. Kiểm soát phiên bản (monorepo so với separate) không nên có bất kỳ tác động nào đến toàn bộ dự án

Inflexible Requirement #4

Version control (monorepo vs separate) shouldn't have any impact on the overall project

Some people want to use monorepos

Some people want to keep everything in a separate repo

Hình 1.36: Minh họa kiểm soát phiên bản (monorepo so với separate) không nên có bất kỳ tác động nào đến toàn bộ dự án

Đối với monorepo:

- Toàn bộ các microfrontend được lưu trữ trong một kho chung.
- Dễ dàng đảm bảo sự nhất quán về phiên bản của các thư viện dùng chung.
- Quy trình build/test/deploy có thể được đồng bộ hóa hơn.
- Nhưng nó làm tăng độ phức tạp khi nhiều đội cùng làm việc trong một kho lớn và nó có thể ảnh hưởng đến tốc độ phát triển nếu các quy trình không được tối ưu.

Đối với separate repo:

- Dễ dàng triển khai độc lập các module mà không cần đợi các phần khác của hệ thống.
- Đội ngũ phát triển có thể làm việc trên các module khác nhau mà không bị phụ thuộc vào các thay đổi từ đội khác.
- Nhưng sự quản lý phụ thuộc giữa các module có thể phức tạp hơn và quá trình đồng bộ hóa các bản cập nhật có thể khó khăn nếu không có quy trình CI/CD rõ ràng.

e. Container phải có khả năng quyết định luôn sử dụng phiên bản mới nhất của microfrontend hoặc chỉ định một phiên bản cụ thể

Inflexible Requirement #5

Container should be able to decide to always use the latest version of a microfrontend **or** specify a specific version

(1) Container will always use the latest version of a child app
(doesn't require a redeploy of container)

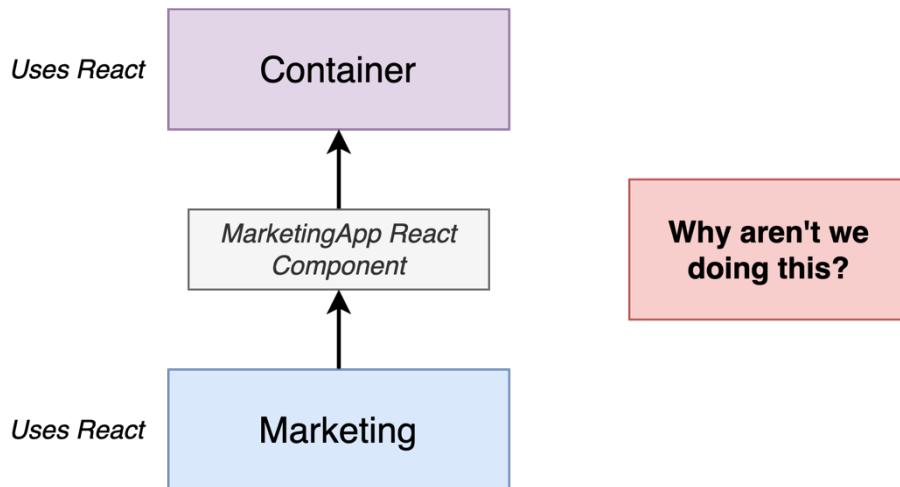
(2) Container can specify exactly what version of a child it wants to use (requires a redeploy to change)

Hình 1.37: Minh họa container phải có khả năng quyết định luôn sử dụng phiên bản mới nhất của microfrontend hoặc chỉ định một phiên bản cụ thể

- Container sẽ luôn sử dụng phiên bản mới nhất của ứng dụng con và nó không bắt buộc phải triển khai lại container.
 - o Áp dụng dynamic loading qua Module Federation hoặc CDN.
 - o Đảm bảo các microfrontend tương thích với container qua kiểm tra backward compatibility.
- Container có thể cụ thể chính xác phiên bản nào của ứng dụng con mà nó muốn để sử dụng và nó bắt buộc phải triển khai để áp dụng thay đổi/cập nhật.
 - o Đặt phiên bản cố định trong cấu hình (ví dụ: URL của module hoặc tag trong package).
 - o Hữu ích khi cần kiểm soát chặt chẽ hoặc rollback.

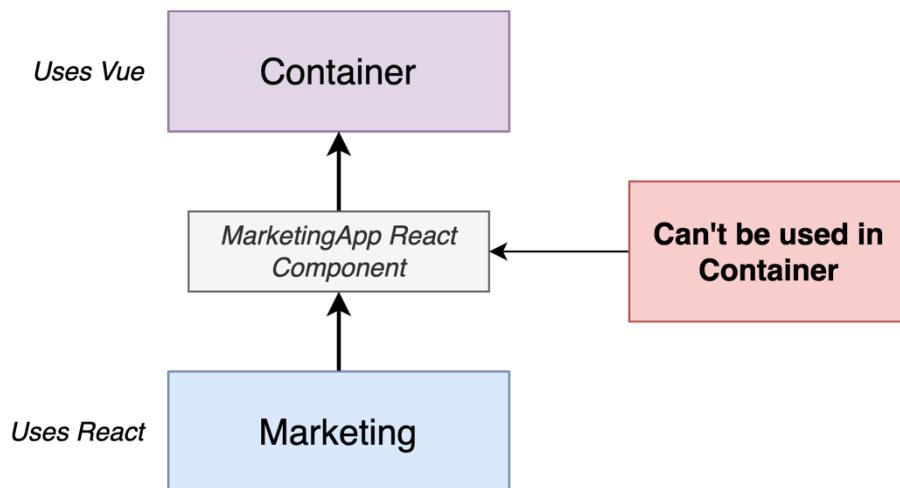
3. Tại sao phải sử dụng mount function để render UI?

Giả sử ta Container sử dụng React và microfrontend Marketing cũng sử dụng React:



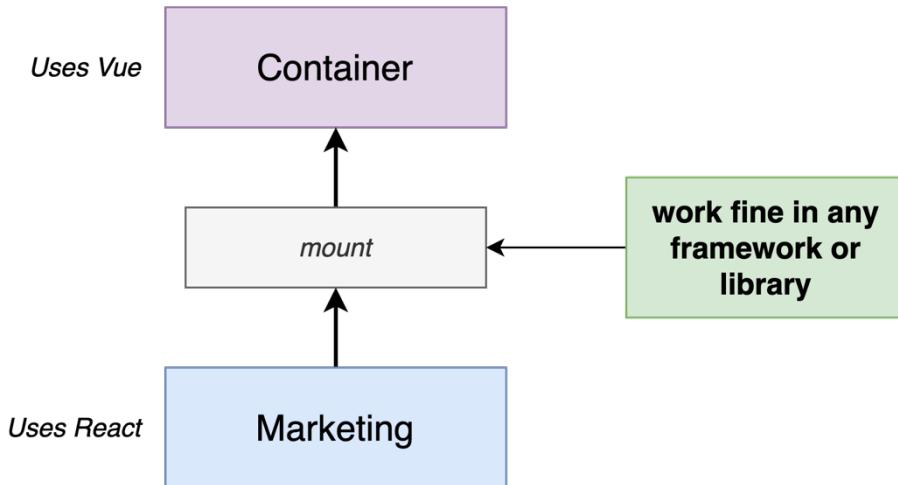
Hình 1.38: Minh họa lí do không nên sử dụng Component liên Component cho từng microfrontend

Lúc này, có sự tương thích giữa 2 thư viện (framework) với nhau, cho nên ta có thể expose microfrontend Marketing như một React Component. Nhưng điều này đã vi phạm với yêu cầu lựa chọn kiến trúc #2, nghĩa là sau này nếu như hoặc Marketing đổi framework, hoặc Container đổi framework, sẽ dẫn đến việc chỉnh sửa source code rất nhiều (xung đột với mục đích chọn kiến trúc) vì không có sự tương thích giữa các framework.



Hình 1.39: Minh họa có sự thay đổi thư viện sẽ dẫn đến lỗi

Chính vì thế, thay vì ta expose microfrontend như những component, thì ta sẽ expose nó như những mount function để có thể được sử dụng ở bất cứ đâu và tương thích với bất kỳ framework nào.



Hình 1.40: Minh họa lí do nên sử dụng hàm mount

Mount function sẽ chọn phần tử (HTML DOM) thông qua id/class ở trang web và render lên màn hình thông qua logic được định sẵn trong nó.

V.Lợi ích của Microfrontend trong phát triển ứng dụng

Chia khối đơn thành các mô-đun được ghép nối lỏng lẻo là một nhiệm vụ phức tạp và chỉ có ý nghĩa khi các giao diện microfrontend giải quyết được các vấn đề mà chúng được cho là giải quyết. Nếu ta không gặp phải những vấn đề này, thì việc trải qua rắc rối khi áp dụng phương pháp giao diện microfrontend là không có ý nghĩa gì.

Sau đây là những lợi ích mà phương pháp giao diện microfrontend có thể mang lại:

1. Triển khai nhanh hơn và quản lý bản phát hành tốt hơn

Trái ngược với giao diện đơn khối (monolith), trong đó một nhóm duy nhất chịu trách nhiệm triển khai tất cả các chức năng và tính năng mới, giao diện microfrontend cho phép các nhóm nhỏ hơn, tự chủ làm việc đồng thời để triển khai nhiều chức năng và tính năng khác nhau. Điều này giúp giảm đáng kể thời gian phát triển và đẩy nhanh quá trình phát hành.

2. Nhiều nhóm có trách nhiệm khác nhau

Mỗi nhóm chịu trách nhiệm về giao diện microfrontend của mình, là giao diện độc lập. Họ có thể xây dựng, thử nghiệm, triển khai và cập nhật giao diện này một cách độc lập. Với kiến trúc khối đơn, nếu Nhóm A đã sẵn sàng phát hành và Nhóm B cần thời gian để chuẩn bị, Nhóm A phải đợi. Rào cản này đã

được loại bỏ trong phương pháp giao diện microfrontend, vì mỗi nhóm có quyền tự do làm những gì họ chọn và không phụ thuộc vào người khác.

3. Tự do công nghệ

Mỗi microfrontend có thể được triển khai trên các ngăn xếp công nghệ khác nhau vì chúng là các phần mềm độc lập. Do đó, các nhóm làm việc trên chúng có thể tự do lựa chọn ngăn xếp công nghệ của riêng mình, dựa trên chuyên môn và kinh nghiệm của họ.

4. Dễ dàng mở rộng quy mô

Microfrontend còn mang đến một lợi thế khác cho các nhà phát triển. Vì mỗi tính năng có thể được mở rộng quy mô độc lập, nên toàn bộ quy trình trở nên hiệu quả hơn về mặt chi phí và thời gian so với monolith.

5. Triển khai liên tục

Bằng cách chia nhỏ frontend monolith thành các thành phần nhỏ hơn, độc lập, microfrontend cho phép cập nhật gia tăng mà không ảnh hưởng đến toàn bộ ứng dụng. Chúng cũng giúp dễ dàng khôi phục một thành phần duy nhất về phiên bản trước đó, có thể cải thiện sự cộng tác của nhóm bằng cách giảm tình trạng tắc nghẽn và tăng khả năng mở rộng quy mô, cùng nhiều lợi thế khác.

Chương 2. Ứng dụng Microfrontend trong lĩnh vực đặt phòng khách sạn

I. Yêu cầu chức năng

Bảng 2.1: Yêu cầu chức năng của ứng dụng đặt phòng khách sạn

Nhóm	Chức năng	Mô tả	Tác nhân
Quản lý khách sạn và phòng	Thêm, chỉnh sửa, xóa khách sạn	Cho phép thêm mới, cập nhật thông tin và xóa khách sạn khỏi hệ thống.	Admin
	Quản lý thông tin phòng	Quản lý thông tin từng loại phòng của khách sạn: loại phòng, giá, trạng thái, số lượng khách tối đa, hình ảnh.	
	Quản lý tiện ích khách sạn	Quản lý các tiện ích của khách sạn như hồ bơi, Wi-Fi, bãi đỗ xe, dịch vụ phòng, v.v.	
	Quản lý khuyến mãi	Thêm, cập nhật và xóa các chương trình khuyến mãi cho từng khách sạn, bao gồm chiết khấu, thời gian hiệu lực, và mô tả khuyến mãi.	
Tìm kiếm và khám phá	Tìm kiếm khách sạn	Tìm kiếm khách sạn theo địa điểm, ngày, và bộ lọc như giá, loại phòng, xếp hạng và tiện ích.	Khách hàng
	Gợi ý khách sạn và điểm đến	Đề xuất khách sạn và điểm đến theo xu hướng hoặc dựa trên vị trí của người dùng (nếu được phép).	
	Xem chi tiết khách sạn	Hiển thị thông tin chi tiết khách sạn, bao gồm mô tả, ảnh, tiện ích, loại phòng và giá.	
	Khám phá các điểm đến	Khám phá các điểm đến phổ biến và nội dung gợi ý liên quan (như địa danh, hoạt động du lịch).	
Đặt phòng	Lựa chọn phòng và ngày	Chọn loại phòng, số lượng phòng, ngày nhận và trả phòng, hiển thị chi phí dự kiến.	Khách hàng

	Xác nhận đặt phòng	Xác nhận thông tin đặt phòng và chuyển sang bước thanh toán.	
	Quản lý đặt phòng	Theo dõi và quản lý trạng thái của từng đặt phòng của khách hàng.	Khách hàng, Admin
	Thay đổi/hủy đặt phòng	Cho phép thay đổi hoặc hủy đặt phòng trước ngày nhận phòng theo chính sách.	Khách hàng
Thanh toán	Tính toán tổng chi phí	Tính tổng tiền bao gồm khuyến mãi và thuế.	Hệ thống
	Phương thức thanh toán	Hỗ trợ các phương thức thanh toán như thẻ tín dụng, ví điện tử và chuyển khoản.	Khách hàng
	Xác nhận thanh toán	Xác nhận trạng thái giao dịch thành công hoặc thất bại.	Hệ thống
	Quản lý hoàn tiền	Hỗ trợ hoàn tiền trong trường hợp hủy đặt phòng hoặc yêu cầu khác.	Admin
Đánh giá và xếp hạng	Đánh giá và xếp hạng	Cho phép khách hàng để lại đánh giá và xếp hạng khách sạn sau khi hoàn thành chuyến đi.	Khách hàng
	Hiển thị đánh giá	Hiển thị đánh giá của các khách hàng khác, bao gồm xếp hạng, bình luận.	
	Quản lý đánh giá	Kiểm duyệt các đánh giá để đảm bảo tính trung thực và phù hợp.	Admin
Quản lý người dùng	Đăng ký và đăng nhập	Cho phép người dùng tạo tài khoản mới và đăng nhập vào hệ thống.	Khách hàng
	Cập nhật thông tin cá nhân	Khách hàng có thể cập nhật thông tin như tên, email, số điện thoại.	
	Khôi phục mật khẩu	Hỗ trợ khôi phục mật khẩu trong trường hợp quên.	
	Quản lý lịch sử đặt phòng	Hiển thị lịch sử đặt phòng của người dùng trong tài khoản cá nhân.	

Quản lý tài khoản admin	Báo cáo và thống kê	Theo dõi báo cáo tổng quan: tổng số đặt phòng, doanh thu, khách sạn nổi bật.	Admin
Tương tác người dùng	Thông báo	Gửi thông báo cho người dùng về trạng thái đặt phòng, khuyến mãi và các gợi ý điểm đến.	Hệ thống, Admin
	Danh sách yêu thích	Cho phép người dùng lưu các khách sạn yêu thích để tham khảo sau.	Khách hàng

II. Yêu cầu phi chức năng

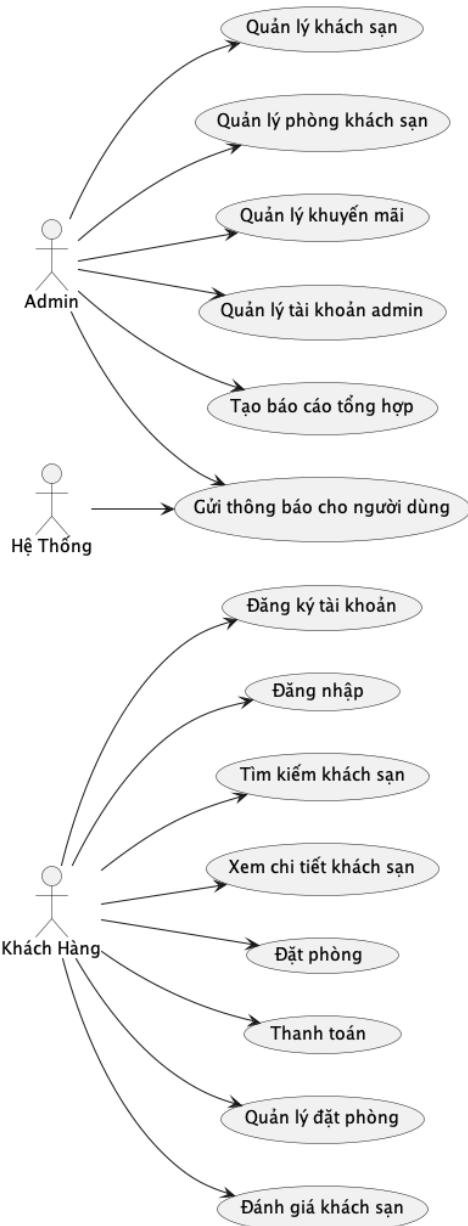
Bảng 2.2: Yêu cầu phi chức năng của ứng dụng đặt phòng khách sạn

Yêu cầu phi chức năng	Mô tả
Hiệu năng	Hệ thống phải đáp ứng ít nhất 1000 yêu cầu đồng thời.
	Thời gian phản hồi dưới 3 giây cho các thao tác chính (tìm kiếm, đặt phòng, thanh toán).
	Thời gian tải trang không quá 2 giây trong điều kiện mạng bình thường.
Khả năng mở rộng	Hệ thống phải dễ dàng mở rộng để hỗ trợ thêm nhiều khách sạn, địa điểm và người dùng.
	Cơ sở hạ tầng có thể mở rộng khi nhu cầu lưu trữ và lưu lượng truy cập tăng lên.
Bảo mật	Mã hóa mọi thông tin nhạy cảm (mật khẩu, thông tin thanh toán) trước khi lưu trữ.
	Tuân thủ chuẩn bảo mật PCI-DSS cho thanh toán, hỗ trợ xác thực hai yếu tố (2FA) cho Admin.
	Đảm bảo hệ thống không dễ bị tấn công bởi SQL Injection, XSS, CSRF, v.v.

Tính sẵn sàng và tin cậy	Hệ thống phải có tính khả dụng cao (99.9% uptime), đảm bảo không gián đoạn trong giờ cao điểm.
	Cơ chế sao lưu và phục hồi dữ liệu để không mất dữ liệu khi có sự cố.
Khả năng duy trì và dễ quản lý	Thiết kế dễ bảo trì với mã nguồn và cơ sở dữ liệu có cấu trúc rõ ràng.
	Hệ thống log chi tiết hỗ trợ theo dõi và xử lý lỗi nhanh chóng.
	Giao diện quản trị trực quan, dễ sử dụng cho Admin và dễ dàng tùy chỉnh khi mở rộng.
Khả năng tương thích	Hoạt động tốt trên nhiều trình duyệt như Chrome, Firefox, Safari, và Edge.
	Hỗ trợ trên các nền tảng và thiết bị khác nhau, bao gồm máy tính và thiết bị di động.
Khả năng phục hồi	Khôi phục nhanh chóng sau các sự cố, tối đa trong 15 phút.
	Sao lưu định kỳ các thành phần quan trọng và khôi phục mà không gây ảnh hưởng đến người dùng.
Khả năng giám sát và báo cáo	Hệ thống giám sát phát hiện sự cố trong thời gian thực và thông báo cho đội ngũ quản trị.
	Cung cấp báo cáo chi tiết về tình trạng hoạt động, hiệu suất, doanh thu và các chỉ số quan trọng cho Admin.
Trải nghiệm người dùng (UX)	Giao diện thân thiện, dễ điều hướng, phù hợp cho người dùng mới và dễ tiếp cận.
	Các quy trình chính (tìm kiếm, đặt phòng, thanh toán) được thiết kế đơn giản, giúp người dùng hoàn thành các thao tác nhanh chóng và dễ dàng.

III. Use case:

1. Use case tổng quan:



Hình 2.1: Use case tổng quan của ứng dụng đặt phòng khách sạn

2. Đặc tả Use case:

a. Đăng ký:

Bảng 2.3: Use case đăng ký tài khoản

Yếu tố	Chi tiết
Tên Use Case	Đăng ký tài khoản

Mô tả	Khách hàng tạo tài khoản để sử dụng các tính năng của hệ thống.
Tác nhân	Khách hàng
Tiến trình	<ol style="list-style-type: none"> 1. Khách hàng nhập thông tin cá nhân (tên, email, mật khẩu, số điện thoại). 2. Hệ thống xác nhận và tạo tài khoản mới. 3. Khách hàng nhận email xác nhận đăng ký.
Điều kiện tiên quyết	Khách hàng chưa có tài khoản
Điều kiện kết thúc	Tài khoản đã được tạo và email xác nhận gửi cho khách hàng.
Kết quả mong đợi	Tạo tài khoản thành công và khách hàng nhận được email xác nhận.

b. Đăng nhập:

Bảng 2.4: Use case đăng nhập

Yếu tố	Chi tiết
Tên Use Case	Đăng nhập
Mô tả	Khách hàng đăng nhập vào hệ thống để truy cập các dịch vụ như đặt phòng, xem lịch sử đặt phòng.
Tác nhân	Khách hàng
Tiến trình	<ol style="list-style-type: none"> 1. Khách hàng nhập thông tin đăng nhập (email, mật khẩu). 2. Hệ thống xác thực thông tin và cho phép đăng nhập nếu đúng.
Điều kiện tiên quyết	Khách hàng đã có tài khoản
Điều kiện kết thúc	Khách hàng được đăng nhập vào hệ thống.

Kết quả mong đợi	Khách hàng đăng nhập thành công và vào được trang chủ hệ thống.
-------------------------	---

c. Tìm kiếm khách sạn:

Bảng 2.5: Use case tìm kiếm khách sạn

Yếu tố	Chi tiết
Tên Use Case	Tìm kiếm khách sạn
Mô tả	Khách hàng tìm kiếm khách sạn theo các tiêu chí như địa điểm, ngày, giá, tiện ích, v.v.
Tác nhân	Khách hàng
Tiến trình	<ol style="list-style-type: none"> Khách hàng nhập thông tin tìm kiếm (địa điểm, ngày check-in, check-out, bộ lọc). Hệ thống hiển thị kết quả tìm kiếm khách sạn phù hợp.
Điều kiện tiên quyết	Không có yêu cầu đặc biệt
Điều kiện kết thúc	Kết quả tìm kiếm khách sạn hiển thị.
Kết quả mong đợi	Khách hàng nhận được danh sách khách sạn phù hợp với yêu cầu tìm kiếm.

d. Xem chi tiết khách sạn:

Bảng 2.6: Use case xem chi tiết khách sạn

Yếu tố	Chi tiết
Tên Use Case	Xem chi tiết khách sạn
Mô tả	Khách hàng xem thông tin chi tiết về khách sạn bao gồm mô tả, tiện ích, giá phòng, ảnh, v.v.
Tác nhân	Khách hàng
Tiến trình	<ol style="list-style-type: none"> Khách hàng chọn khách sạn trong kết quả tìm kiếm. Hệ thống hiển thị thông tin chi tiết về khách sạn và các loại phòng.

Điều kiện tiên quyết	Khách sạn đã được tìm thấy
Điều kiện kết thúc	Thông tin chi tiết của khách sạn được hiển thị.
Kết quả mong đợi	Khách hàng thấy thông tin chi tiết về khách sạn.

e. Đặt phòng:

Bảng 2.7: Use case đặt phòng khách sạn

Yếu tố	Chi tiết
Tên Use Case	Đặt phòng
Mô tả	Khách hàng chọn phòng và thực hiện đặt phòng.
Tác nhân	Khách hàng
Tiến trình	<ol style="list-style-type: none"> Khách hàng chọn loại phòng, số lượng phòng, ngày check-in/check-out. Hệ thống tính toán giá tổng và hiển thị. Khách hàng xác nhận đặt phòng.
Điều kiện tiên quyết	Khách hàng đã chọn khách sạn và phòng
Điều kiện kết thúc	Đặt phòng đã được xác nhận và thông tin gửi qua email.
Kết quả mong đợi	Khách hàng hoàn tất việc đặt phòng và nhận thông báo xác nhận.

f. Thanh toán:

Bảng 2.8: Use case thanh toán

Yếu tố	Chi tiết
Tên Use Case	Thanh toán
Mô tả	Khách hàng thực hiện thanh toán cho đặt phòng đã chọn.

Tác nhân	Khách hàng
Tiến trình	<p>1. Khách hàng chọn phương thức thanh toán (thẻ tín dụng, ví điện tử, chuyển khoản).</p> <p>2. Hệ thống xử lý giao dịch và xác nhận thanh toán thành công.</p>
Điều kiện tiên quyết	Đặt phòng đã được xác nhận và hiển thị giá
Điều kiện kết thúc	Thanh toán đã được xử lý thành công và khách hàng nhận thông báo.
Kết quả mong đợi	Khách hàng thanh toán thành công và nhận được thông báo xác nhận thanh toán.

g. Quản lý đặt phòng:

Bảng 2.9: Use case quản lý đặt phòng

Yếu tố	Chi tiết
Tên Use Case	Quản lý đặt phòng (Xem và hủy bỏ)
Mô tả	Khách hàng có thể xem và hủy bỏ các đặt phòng trong tài khoản của mình.
Tác nhân	Khách hàng
Tiến trình	<p>1. Khách hàng vào mục "Lịch sử đặt phòng" và xem thông tin đặt phòng.</p> <p>2. Nếu cần, khách hàng hủy đặt phòng trước ngày nhận phòng.</p>
Điều kiện tiên quyết	Khách hàng đã có các đặt phòng trong tài khoản
Điều kiện kết thúc	Đặt phòng đã được hủy và xác nhận qua email.
Kết quả mong đợi	Khách hàng thành công trong việc xem và hủy đặt phòng.

h. Đánh giá khách sạn:

Bảng 2.10: Use case đánh giá khách sạn

Yếu tố	Chi tiết
Tên Use Case	Đánh giá khách sạn
Mô tả	Khách hàng để lại đánh giá và xếp hạng cho khách sạn sau khi sử dụng dịch vụ.
Tác nhân	Khách hàng
Tiến trình	<ol style="list-style-type: none"> Khách hàng vào trang khách sạn và chọn "Đánh giá". Khách hàng nhập nội dung đánh giá và chọn số sao (xếp hạng). Hệ thống lưu lại đánh giá.
Điều kiện tiên quyết	Khách hàng đã hoàn thành chuyến đi và đặt phòng
Điều kiện kết thúc	Đánh giá của khách hàng được lưu và hiển thị.
Kết quả mong đợi	Đánh giá và xếp hạng của khách hàng được ghi nhận và hiển thị trên trang khách sạn.

IV. Các module microfrontend chính:

1. Team Quản lý đặt phòng

- Module: Booking
- Chức năng:
 - Tìm kiếm khách sạn
 - Hiển thị danh sách khách sạn phù hợp (với bộ lọc)
 - Chi tiết phòng và giá cả
 - Chọn ngày và số lượng phòng, khách
 - Xác nhận đặt phòng
 - Gợi ý khách sạn dựa trên vị trí, xu hướng du lịch hoặc sở thích của người dùng
 - Cung cấp các địa điểm du lịch, điểm đến phổ biến/thịnh hành

- Cung cấp các loại chỗ nghỉ khác nhau (Khách sạn, căn hộ, resort,...)
- Cung cấp các khách sạn đang có ưu đãi

2. Team Thanh toán

- Module: Payment
- Chức năng:
 - Xác nhận thông tin thanh toán
 - Lựa chọn phương thức thanh toán (thẻ tín dụng, ví điện tử,...)
 - Xử lý giao dịch thanh toán
 - Hiển thị trạng thái giao dịch (thành công/thất bại)

3. Team Quản lý người dùng

- Module: User Profile
- Chức năng:
 - Đăng ký/Đăng nhập
 - Quản lý thông tin tài khoản
 - Hiển thị và quản lý lịch sử đặt phòng
 - Cập nhật thông tin cá nhân (email, số điện thoại,...)

4. Team Đánh giá và Xếp hạng

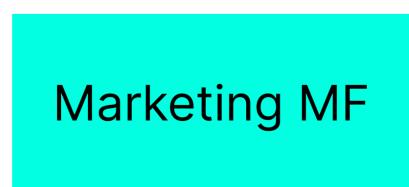
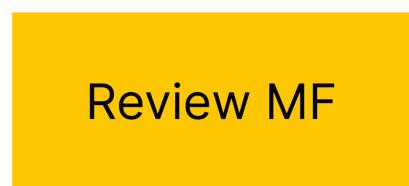
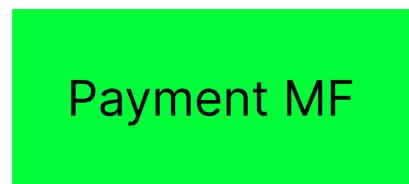
- Module: Review
- Chức năng:
 - Hiển thị đánh giá khách sạn từ người dùng
 - Cho phép người dùng thêm đánh giá và xếp hạng sau khi hoàn tất chuyến đi
 - Tính năng lọc đánh giá theo mức xếp hạng

5. Team Marketing

- Module: Marketing
- Chức năng:
 - Hiển thị các hình ảnh quảng bá trang web
 - Hiển thị các giá trị mà trang web cũng như doanh nghiệp mang lại

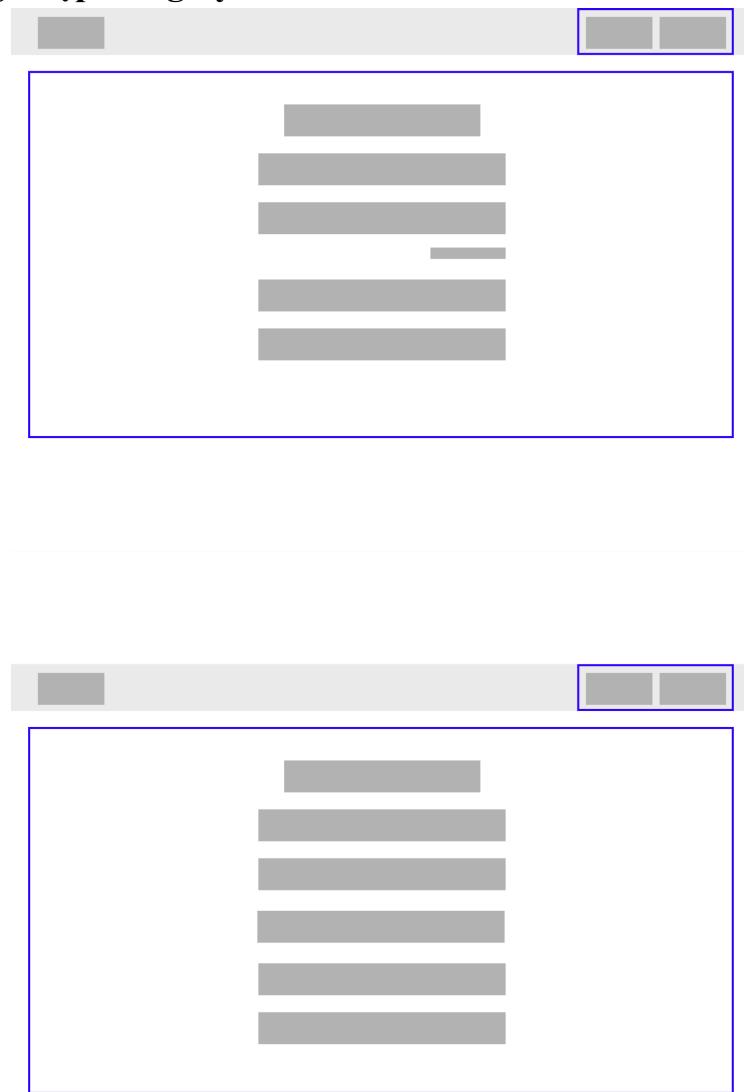
V. Một số ví dụ trừu tượng hóa các microfrontend trên trang

Chú thích hình ảnh:



Hình 2.2: Chú thích khung hình ảnh cho từng microfrontend

1. Trang đăng nhập/đăng ký:



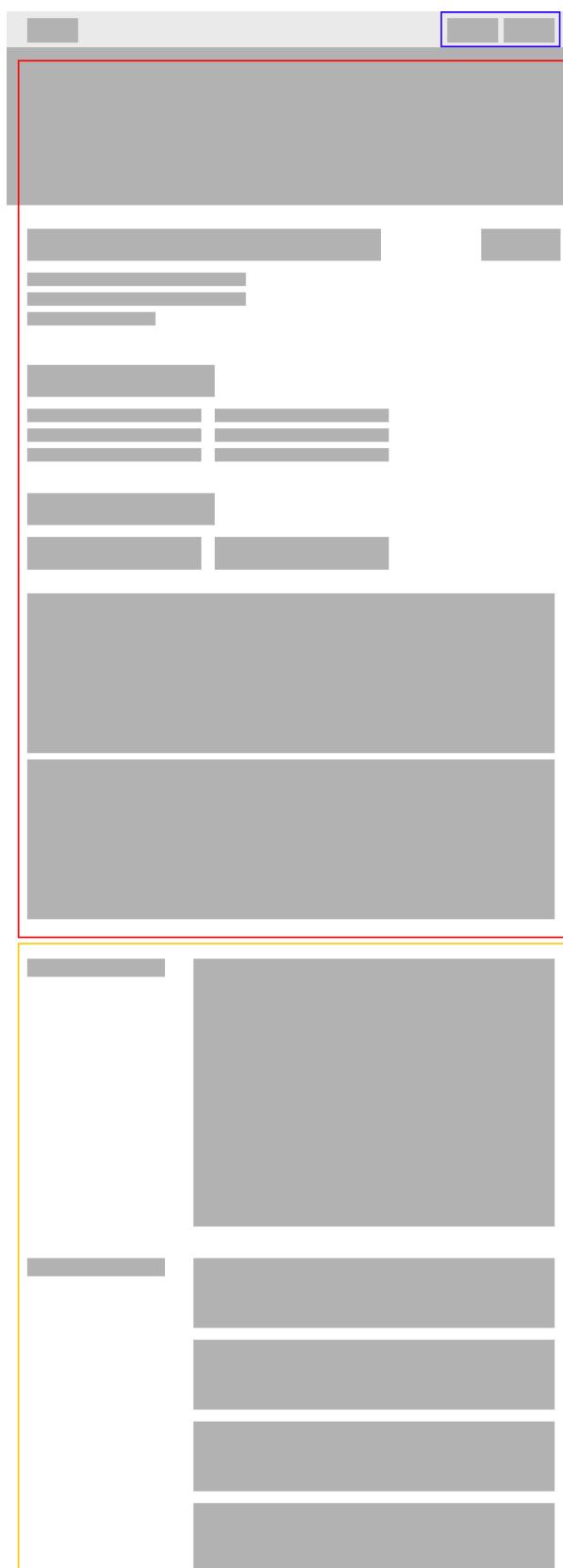
Hình 2.3: Wireframe trang đăng nhập/đăng ký

2. Trang chủ:



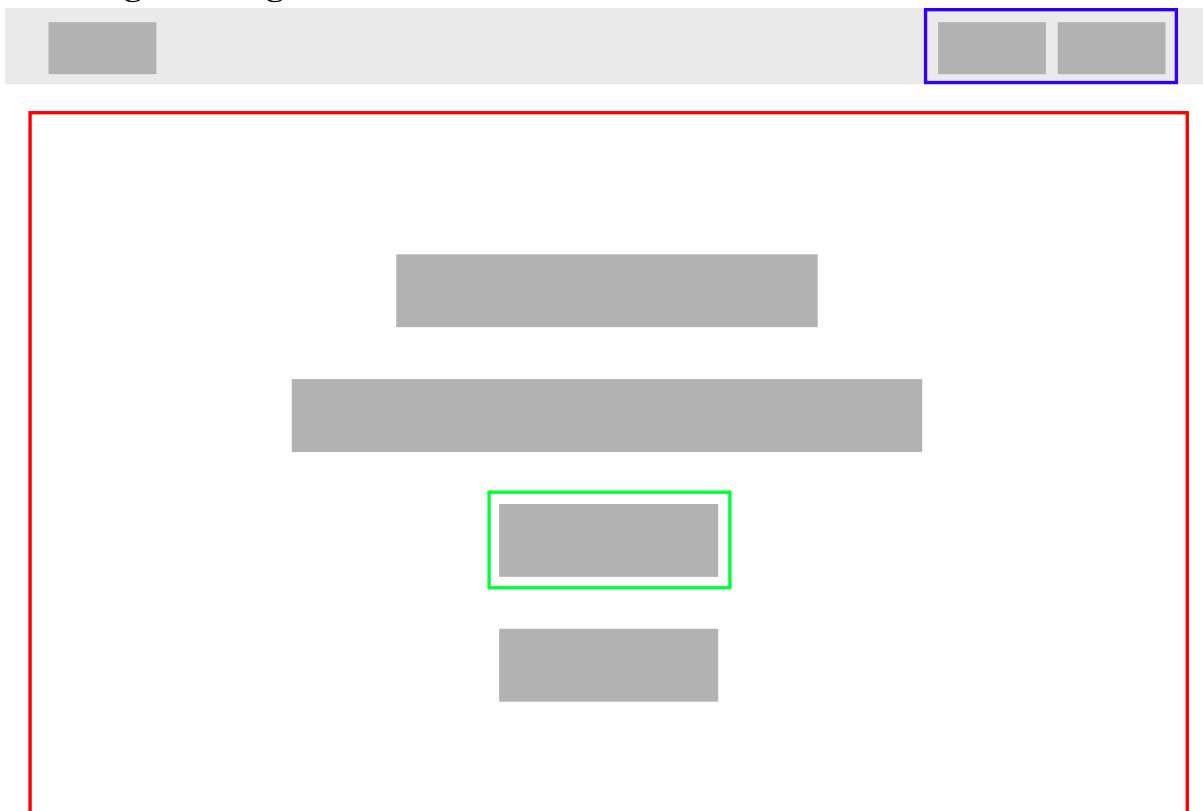
Hình 2.4: Wireframe trang chủ

3. Trang chi tiết



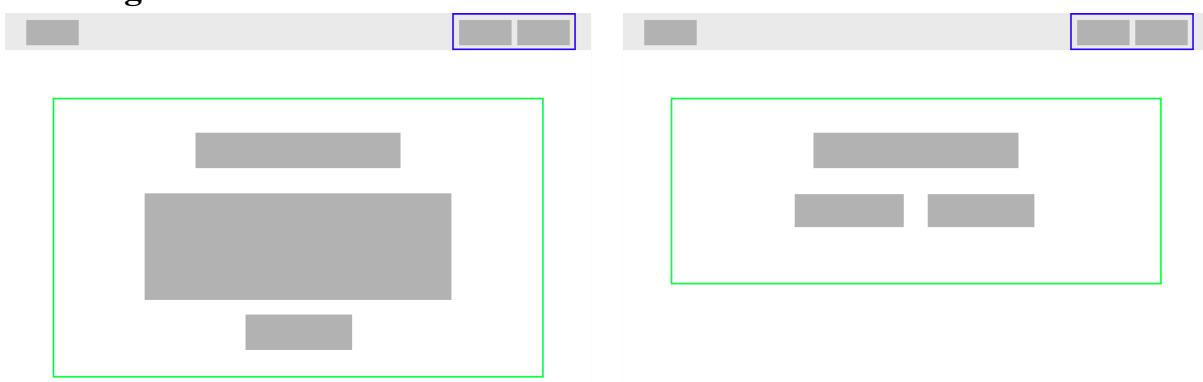
Hình 2.5: Wireframe trang chi tiết khách sạn

4. Trang Booking:

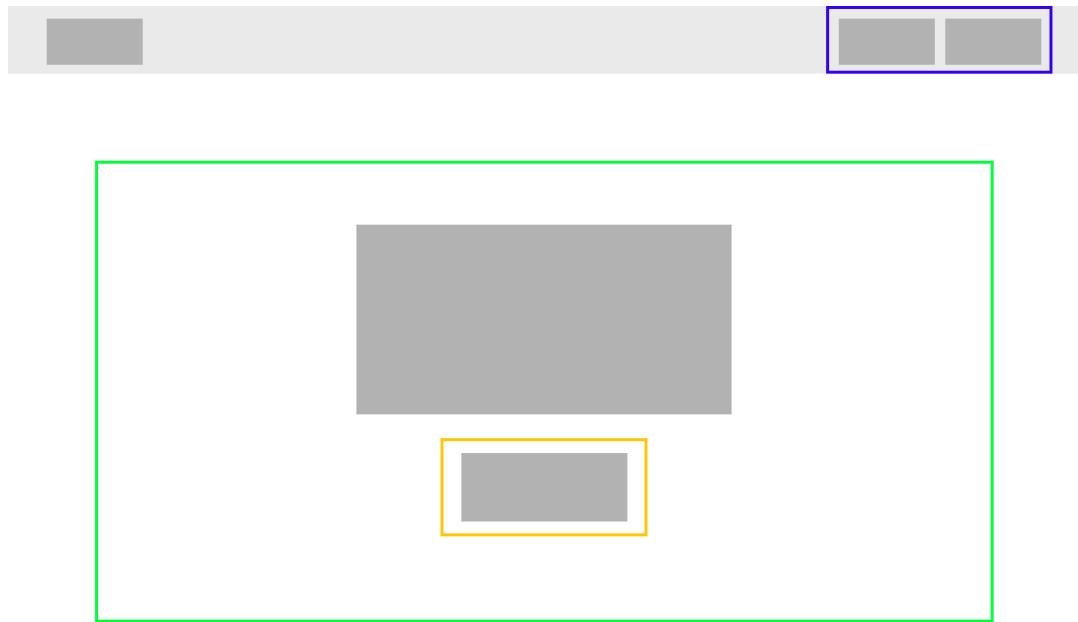


Hình 2.6: Wireframe trang đặt phòng

5. Trang Thanh toán:



Hình 2.7: Wireframe trang thanh toán



Hình 2.8: Wireframe trang thanh toán thành công

VI. Thực hiện hoá các microfrontend lên trên trang

1. Trang Đăng nhập/Đăng ký:

The image shows two side-by-side screenshots of a web application's login and registration pages, both titled "Booking".

Login Page (Left):

- Header:** Displays "Đăng nhập" and "Đăng ký" buttons.
- Form Fields:** Includes input fields for "Email/SĐT" and "Mật khẩu".
- Buttons:** A large green "Đăng nhập" button, a "hoặc" link, and a "Đăng nhập với Google" button.
- Text:** A link "Bạn chưa có tài khoản? Đăng ký ngay" at the bottom.

Registration Page (Right):

- Header:** Displays "Đăng nhập" and "Đăng ký" buttons.
- Form Fields:** Includes input fields for "Tên", "Email", "SĐT", "Mật khẩu", and "Xác thực mật khẩu".
- Buttons:** A large green "Đăng ký" button, a "hoặc" link, and a "Đăng ký với Google" button.
- Text:** A link "Bạn đã có tài khoản? Đăng nhập ngay" at the bottom.

Hình 2.9: Minh họa giao diện trang Đăng nhập/Đăng ký và microfrontend

Các microfrontend được áp dụng: User Profile

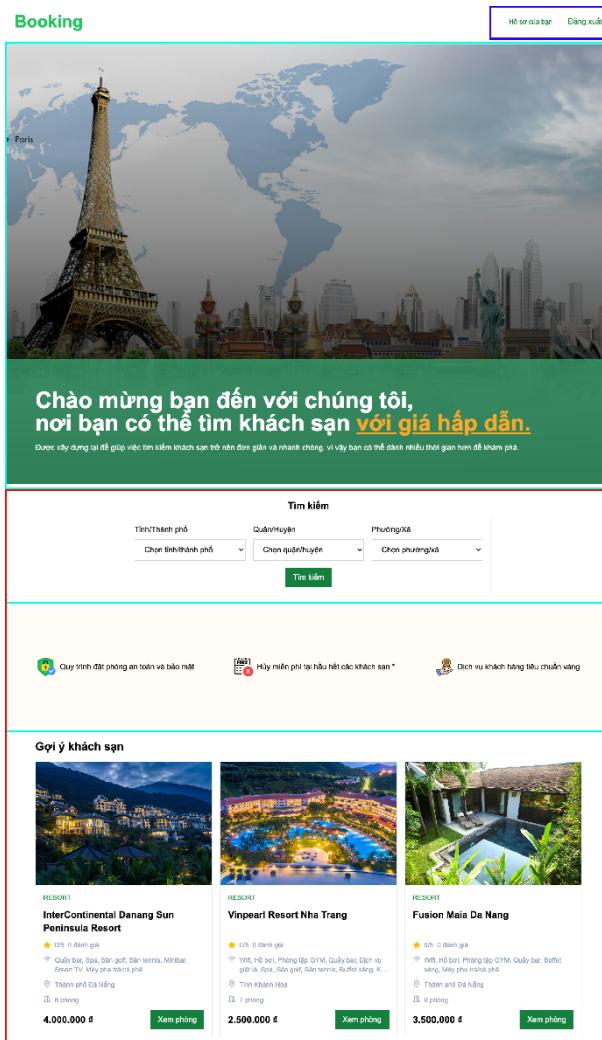
2. Trang Hồ sơ người dùng:

The screenshot displays two views of a user profile page. The left view shows a list of bookings with a yellow box highlighting the 'Đánh giá' (Review) button for each entry. The right view shows a detailed booking entry with a yellow box highlighting the 'Đánh giá khách sạn' (Hotel Review) form. Both views include tabs for 'Thông tin hồ sơ' (Profile info), 'Lịch sử đặt phòng' (Booking history), and other account-related sections.

Hình 2.10: Minh họa giao diện trang Hồ sơ người dùng và microfrontend

Các microfrontend được áp dụng: User Profile, Review, Booking.

3. Trang chủ:



Hình 2.11: Minh họa giao diện trang Trang chủ và microfrontend

Các microfrontend được áp dụng: User Profile, Marketing, Booking.

4. Trang Chi tiết khách sạn:

The screenshot displays a travel booking interface for the **InterContinental Danang Sun Peninsula Resort**. At the top, there's a green "Booking" button and a red-bordered "Hỗ trợ của bạn | Đăng xuất" (Help | Log Out) button. Below the header is a large image of the resort's buildings at night.

[RESORT] InterContinental Danang Sun Peninsula Resort

Đánh giá
④ A20, Phường An Hải Bắc, Quận Sơn Trà, Thành phố Đà Nẵng
④ Quầy bar, Spa, Sân golf, Sân tennis, Minibar, Smart TV, Máy pha trà/cà phê

Mô tả
Khu nghỉ dưỡng nằm trên đồi với tầm nhìn toàn cảnh biển tuyệt đẹp

Phòng

TÊN PHÒNG	LỐI PHÒNG	GIA	SỨC CHẤT	BẢNG PHÒNG
Phòng cho 4 người có Ban Công	quadruple	6.300.000 ₫	4	<button>Thêm</button>
Phòng cho 3 người có Ban Công	triple	4.900.000 ₫	3	<button>Thêm</button>
Phòng Deluxe Giường Đôi Có Ban Công	studio	3.900.000 ₫	2	<button>Thêm</button>
Phòng cho 3 người	triple	3.500.000 ₫	3	<button>Thêm</button>
Phòng Deluxe Giường Đôi	studio	2.500.000 ₫	2	<button>Thêm</button>
Phòng cho 4 người	quadruple	5.500.000 ₫	4	<button>Thêm</button>
Phòng Deluxe Giường Đôi Ban Công	triple	5.000.000 ₫	4	<button>Thêm</button>
Phòng Deluxe Giường Đôi	triple	4.500.000 ₫	4	<button>Thêm</button>

Đánh giá
Không có đánh giá nào

Khách sạn liên quan

- RESORT**
Vinpearl Resort Nha Trang
④ 05 0 đánh giá
④ Vinpearl Hồ bơi, Phòng tập Gym, Quầy bar, Dịch vụ giặt là, Spa, Sân golf, Sân tennis, Buffet sáng, K...
④ Tiện ích nhà
④ phòng
2.500.000 ₫ **Xem phòng**
- RESORT**
Fusion Maia Da Nang
④ 05 0 đánh giá
④ WiFi, Hồ bơi, Phòng tập Gym, Quầy bar,uffet sáng, K...
④ Tiện ích nhà
④ phòng
3.500.000 ₫ **Xem phòng**
- BUỔI HỘI**
Hotel de l'Opera Hanoi
④ 05 0 đánh giá
④ WiFi, Quầy bar, Minibar, Buffet sáng, Máy pha cà phê, Phòng họp, Hồ bơi
④ Tiện ích nhà
④ phòng
1.800.000 ₫ **Xem phòng**

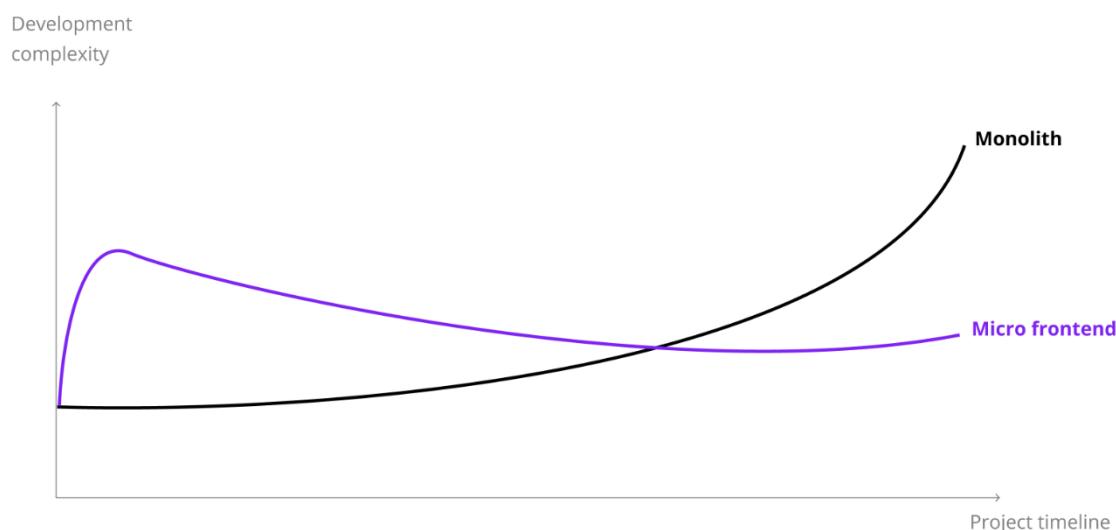
Hình 2.12: Minh họa giao diện trang Chi tiết khách sạn và microfrontend

Các microfrontend được áp dụng: User Profile, Booking, Review, Payment.

Chương 3.Tổng kết

I. Thách thức khi áp dụng Microfrontend

Microfrontend có nhiều lợi ích, vì vậy, thật dễ dàng để cho rằng chúng luôn là giải pháp tốt nhất cho kiến trúc frontend của developer. Trên thực tế, điều này không đúng. Phương pháp microfrontend không có khả năng giải quyết tất cả các vấn đề của developer. Do đó, khi quyết định có nên áp dụng phương pháp microfrontend hay không, điều cần thiết là phải giải quyết các thách thức của nó trước.



Hình 3.1: Biểu đồ biểu diễn mức độ phức tạp khi phát triển dự án so với tiến độ dự án

1. Kích thước tải trọng

Microfrontend trùng lặp các phụ thuộc, các framework và thư viện, làm tăng kích thước tải trọng của ứng dụng web. Mặc dù yếu tố này là một thách thức đáng kể, nhưng trang web trên microfrontend sẽ tải xuống nhanh hơn so với trên kiến trúc monolith.

Một số giải pháp bao gồm việc sử dụng tài nguyên cẩn thận, lựa chọn phụ thuộc tỉ mỉ và tách biệt cẩn thận các trang ít khi sử dụng.

2. Giao tiếp giữa các trang

Giao tiếp giữa các microfrontend rất khó triển khai và duy trì nhưng có những trường hợp các trang riêng biệt cần giao tiếp ở mức tối thiểu với nhau. Một phần của ứng dụng cần được máy chủ hoặc các microfrontend khác thông

báo về tương tác của người dùng và cần thay đổi, làm mới hoặc kích hoạt hành động.

3. Sự khác biệt về thiết kế

Nếu các nhóm riêng biệt làm việc trên mỗi microfrontend, họ có thể không thấy được toàn cảnh của ứng dụng. Do đó, các trang của trang web có thể trông giống như được tạo thành từ nhiều bản vá, không nhất quán về phong cách và UX/UI.

Ngoài ra, còn có nhiều cách để giải quyết vấn đề này – tạo các component chung, hướng dẫn về style mà tất cả các nhóm có thể tham khảo hoặc chỉ cần giao tiếp với nhau.

4. Độ phức tạp về mặt vận hành

Cuối cùng, việc áp dụng microfrontend vào giao diện người dùng gây ra độ phức tạp về mặt vận hành vì cách tiếp cận kiến trúc giao diện người dùng như vậy đòi hỏi nhiều kho lưu trữ, công cụ, nhiều pipelines xây dựng/triển khai và cơ sở hạ tầng phức tạp hơn để tất cả có thể hoạt động cùng nhau.

II. Thực tiễn và ví dụ thành công

1. Các công ty đã áp dụng Microfrontend

- Âm nhạc & Phương tiện: Spotify sử dụng microfrontend cho một nền tảng linh hoạt, theo mô-đun, cho phép phát triển nhanh nhẹn và cung cấp tính năng nhanh chóng.
- Bán lẻ: IKEA và Zalando khai thác microfrontend để mô-đun hóa nền tảng trực tuyến của họ, đảm bảo khả năng bảo trì mã và phát hành tính năng nhanh hơn.
- Tài chính: American Express và PayPal sử dụng microfrontend để nâng cao khả năng mở rộng và khả năng bảo trì của các ứng dụng web của họ.
- Thực phẩm & Đồ uống: Starbucks áp dụng microfrontend để hợp lý hóa nền tảng đặt hàng trực tuyến của mình, đảm bảo trải nghiệm người dùng mượt mà trong bối cảnh cơ sở mã ngày càng phát triển.
- Công nghệ: Upwork, HelloFresh và LambdaTest thúc đẩy kiến trúc web có thể mở rộng và bảo trì với microfrontend, thúc đẩy tính tự chủ của nhóm và chu kỳ phát triển nhanh hơn.

2. Nghiên cứu điển hình về ứng dụng đặt phòng khách sạn (Case Study)

Ví dụ về một ứng dụng đặt phòng khách sạn: Một công ty khởi nghiệp trong lĩnh vực du lịch đã áp dụng kiến trúc Microfrontend cho ứng dụng đặt phòng khách sạn của mình. Dưới đây là một số điểm nổi bật trong quy trình phát triển:

- Chia thành các microfrontend: Công ty này đã chia ứng dụng thành các microfrontend cho các tính năng chính như tìm kiếm, đặt phòng, thanh toán, và đánh giá. Mỗi nhóm phát triển phụ trách một microfrontend cụ thể, giúp tăng tốc độ phát triển và giảm thiểu sự phụ thuộc lẫn nhau.
- Sử dụng API để giao tiếp: Mỗi microfrontend được thiết kế để giao tiếp qua API, cho phép chúng dễ dàng tương tác và chia sẻ dữ liệu, chẳng hạn như thông tin về phòng đã đặt hoặc đánh giá của người dùng.
- Triển khai độc lập: Khi cần cập nhật hoặc sửa lỗi, các nhóm có thể triển khai các microfrontend mà không cần dừng toàn bộ ứng dụng, giúp tiết kiệm thời gian và đảm bảo tính liên tục trong dịch vụ.
- Cải thiện trải nghiệm người dùng: Kết quả là, ứng dụng đã cải thiện đáng kể về tốc độ và tính năng, giúp người dùng dễ dàng tìm kiếm và đặt phòng hơn. Đánh giá từ người dùng cũng tăng lên do trải nghiệm mượt mà và khả năng truy cập nhanh chóng đến các tính năng quan trọng.

III. Tóm tắt các điểm chính

Trong báo cáo này, chúng ta đã khám phá kiến trúc Microfrontend và cách áp dụng nó vào phát triển ứng dụng đặt phòng khách sạn. Microfrontend mang lại nhiều lợi ích như tính linh hoạt, khả năng mở rộng, quản lý đội ngũ phát triển, và tối ưu hóa trải nghiệm người dùng. Việc chia nhỏ ứng dụng thành các microfrontend giúp các nhóm phát triển hoạt động độc lập, cho phép họ nhanh chóng triển khai và cập nhật các tính năng mà không ảnh hưởng đến toàn bộ hệ thống.

Chúng ta cũng đã xem xét các yếu tố cần cân nhắc khi áp dụng Microfrontend, từ việc xác định các tính năng cho đến cách tổ chức nhóm phát triển và quản lý trạng thái. Các ví dụ thành công từ các công ty như Zalando và Spotify chứng minh tính hiệu quả của kiến trúc này trong việc phát triển các ứng dụng lớn và phức tạp.

Chương 4. Kế hoạch thực hiện

Bảng 4.1: Kế hoạch thực hiện

Giai đoạn	Tuần	Ngày	Công việc	Thành viên thực hiện
Lên kế hoạch	1, 2, 3, 4	30/09 – 20/10	- Tìm hiểu các nội dung liên quan đến đề tài - Lập bảng kế hoạch sắp tới - Chuẩn bị dàn ý nội dung	Kiệt, Mai
Đặc tả yêu cầu	5	21/10 – 26/10	- Tổng hợp chức năng và use case - Xây dựng các sơ đồ use case và đặc tả sơ đồ use case	Kiệt, Mai
Thiết kế	6	28/10 - 03/11	- Thiết kế giao diện	Mai
Xây dựng Backend	6, 7	28/10 – 10/11	- Xây dựng Backend cho toàn bộ ứng dụng	Kiệt
Xây dựng ứng dụng	8	11/11 – 17/11	Xây dựng các Microfrontend: - Booking	Mai
	9	18/11 – 25/11	Xây dựng các Microfrontend: - User Profile - Marketing	Kiệt
	10, 11	26/11 – 08/12	Xây dựng các Microfrontend: - Payment - Review	Kiệt
Triển khai ứng dụng	12	09/12 – 13/12	- Testing	Mai

	12	14/12 – 15/12	- Triển khai ứng dụng lên AWS S3 và AWS CloudFront	Kiệt
Báo cáo	13	16/12 – 23/12	- Viết báo cáo và hoàn thiện, chỉnh sửa (nếu có) - Chuẩn bị Slide thuyết trình	Mai, Kiệt

Tài liệu tham khảo

- [1] S. Grider, "Microfrontends with React: A Complete Developer's Guide," 12 2024. [Online]. Available: <https://www.udemy.com/course/microfrontend-course>.
- [2] OpenSource, "Companies already using micro frontends and why," 11 04 2024. [Online]. Available: <https://dev.to/buildwebcrumbs/companies-already-using-micro-frontends-and-why-o37>.
- [3] Wikipedia, "Webpack," 06 08 2024. [Online]. Available: <https://en.wikipedia.org/wiki/Webpack>.
- [4] M. Anser, "Micro Frontend Architecture," 18 10 2021. [Online]. Available: <https://levelup.gitconnected.com/micro-frontend-architecture-794442e9b325>.
- [5] P. Tkhir, "Micro Frontend Architecture: What, Why, and How to Use It," 14 11 2023. [Online]. Available: <https://euristiq.com/micro-frontend-architecture/>.
- [6] M. Geers, "Micro Frontends: Building a modern webapp with multiple teams," 25 03 2017. [Online]. Available: <https://speakerdeck.com/naltatis/micro-frontends-building-a-modern-webapp-with-multiple-teams>.