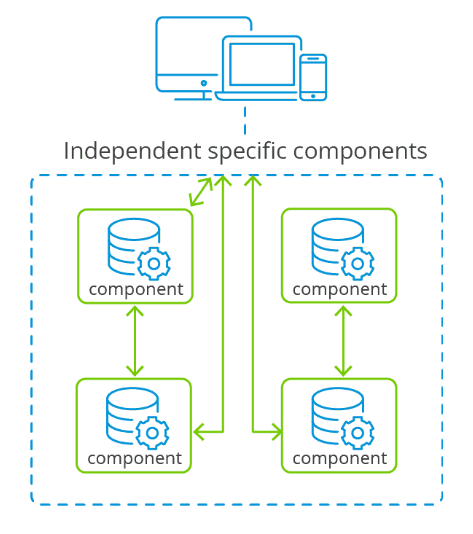
## Component-based

### Khái niệm

* Mô hình Component-based (Phát triển phần mềm dựa trên thành phần) hay còn gọi là Component-based software engineering (CBSE) là một tập hợp các phương pháp để xây dựng hệ thống phần mềm từ các phần có thể tái sử dụng. Những phần này, được gọi là thành phần phần mềm, được đóng gói với chức năng và dữ liệu của chúng được ẩn khỏi thế giới bên ngoài. Chúng tương tác với nhau thông qua các giao diện được định nghĩa rõ ràng.

### Cấu trúc của mô hình component-based

* Thành phần (Component):
  + Là đơn vị cơ bản của mô hình Component-based.
  + Chứa một tập hợp các chức năng và dữ liệu được đóng gói và ẩn khỏi thế giới bên ngoài.
  + Giao tiếp với các thành phần khác thông qua các giao diện được định nghĩa rõ ràng.
  + Có thể được tái sử dụng trong nhiều ứng dụng hoặc hệ thống khác nhau.
* Giao diện (Interface):
  + Định nghĩa các phương thức và thuộc tính mà một thành phần cung cấp cho các thành phần khác.
  + Giúp các thành phần giao tiếp với nhau mà không cần biết chi tiết về cách thức hoạt động bên trong của nhau.
  + Thúc đẩy sự tách biệt trách nhiệm và giúp dễ dàng thay thế các thành phần.
* Môi trường thi hành (Execution Environment):
  + Cung cấp môi trường cho các thành phần để chạy và tương tác với nhau.
  + Quản lý việc tạo, tải và hủy các thành phần.
  + Cung cấp các dịch vụ chung như quản lý bộ nhớ, quản lý tài nguyên và xử lý lỗi.
* Khung phần mềm (Component Framework):
  + Cung cấp một tập hợp các công cụ và dịch vụ để hỗ trợ phát triển và triển khai các ứng dụng Component-based.
  + Giúp đơn giản hóa việc phát triển, quản lý và tích hợp các thành phần.
* Mối quan hệ giữa các thành phần:
  + Các thành phần có thể được kết nối với nhau theo nhiều cách khác nhau, bao gồm:
    - Liên kết tĩnh: Các thành phần được kết nối với nhau tại thời điểm biên dịch.
    - Liên kết động: Các thành phần được kết nối với nhau tại thời điểm chạy.
    - Liên kết thông qua giao diện: Các thành phần giao tiếp với nhau thông qua các giao diện được định nghĩa rõ ràng.
    - Liên kết thông qua sự phụ thuộc: Các thành phần phụ thuộc vào các thành phần khác để hoạt động.



### Hoạt động của mô hình component-based

* Phân chia hệ thống thành các thành phần:
  + Bước đầu tiên là phân chia hệ thống phần mềm thành các thành phần nhỏ hơn, độc lập và có thể tái sử dụng.
  + Mỗi thành phần nên có trách nhiệm rõ ràng và được đóng gói với chức năng và dữ liệu riêng.
  + Việc phân chia thành phần hiệu quả sẽ giúp tăng khả năng tái sử dụng, giảm thiểu phụ thuộc và cải thiện tính mô-đun của hệ thống.
* Xác định giao diện cho mỗi thành phần:
  + Mỗi thành phần cần có một giao diện rõ ràng để định nghĩa các phương thức và thuộc tính mà nó cung cấp cho các thành phần khác.
  + Giao diện giúp các thành phần giao tiếp với nhau mà không cần biết chi tiết về cách thức hoạt động bên trong của nhau.
  + Việc xác định giao diện rõ ràng sẽ giúp tăng khả năng tương thích và giảm thiểu(coupling) giữa các thành phần.
* Triển khai các thành phần:
  + Mỗi thành phần có thể được triển khai bằng ngôn ngữ lập trình, công nghệ và khung phần mềm riêng biệt.
  + Việc triển khai các thành phần cần tuân theo các giao diện đã được xác định trước đó.
  + Có thể sử dụng các công cụ và framework Component-based để hỗ trợ việc triển khai và quản lý các thành phần.
* Tích hợp các thành phần:
  + Các thành phần được tích hợp với nhau thông qua các giao diện của chúng.
  + Môi trường thi hành sẽ quản lý việc tạo, tải và hủy các thành phần, cũng như cung cấp các dịch vụ chung như quản lý bộ nhớ, quản lý tài nguyên và xử lý lỗi.
  + Việc tích hợp các thành phần cần đảm bảo rằng chúng tương thích với nhau và có thể giao tiếp hiệu quả.

### Lợi ích của việc sử dụng mô hình component-based

* Tăng khả năng tái sử dụng: Các thành phần có thể được tái sử dụng trong các ứng dụng khác nhau, giúp giảm thời gian và chi phí phát triển.
* Cải thiện khả năng bảo trì: Liên kết lỏng lẻo và phân chia trách nhiệm khiến các thành phần dễ hiểu, sửa đổi và gỡ lỗi hơn.
* Phát triển nhanh hơn: Bằng cách sử dụng các thành phần được xây dựng sẵn, các nhà phát triển có thể tập trung vào các khía cạnh độc đáo của ứng dụng của họ.
* Giảm thiểu rủi ro: Các thành phần có thể tái sử dụng có khả năng đã được kiểm tra và gỡ lỗi rộng rãi, giảm thiểu nguy cơ lỗi trong ứng dụng cuối cùng.

### Các ngôn ngữ và framework sử dụng mô hình component-based

* Ngôn ngữ:
  + JavaScript:
    - React: thư viện JavaScript phổ biến nhất cho phát triển giao diện người dùng component-based.
    - Vue.js: framework JavaScript nhẹ và linh hoạt cho phát triển ứng dụng web component-based.
    - Angular: framework JavaScript mạnh mẽ và đầy đủ tính năng cho phát triển ứng dụng web component-based.
  + Java:
    - Spring: framework Java phổ biến cung cấp hỗ trợ mạnh mẽ cho lập trình component-based.
    - OSGi: nền tảng Java cho phát triển phần mềm component-based.
  + C#:
    - NET Framework: framework phần mềm của Microsoft cung cấp lớp nền cho phát triển component-based.
    - Unity: engine trò chơi phổ biến sử dụng kiến trúc component-based.
* Framework:
  + React Native: framework JavaScript để xây dựng ứng dụng di động native bằng React.
  + NativeScript: framework JavaScript để xây dựng ứng dụng di động native bằng JavaScript, TypeScript, Vue.js hoặc Angular.
  + Flutter: framework UI di động cross-platform từ Google, sử dụng Dart để xây dựng ứng dụng gốc.
  + Xamarin: framework cross-platform từ Microsoft để xây dựng ứng dụng di động native bằng C#.
  + Apache Camel: framework mã nguồn mở cho tích hợp ứng dụng enterprise, sử dụng mô hình component-based để kết nối các hệ thống khác nhau.

### Tổng kết

* Nhìn chung, mô hình Component-based là một cách tiếp cận mạnh mẽ để xây dựng các hệ thống phần mềm phức tạp. Nó có thể cải thiện khả năng tái sử dụng, khả năng bảo trì và tốc độ phát triển. Tuy nhiên, điều quan trọng là phải xem xét những hạn chế tiềm ẩn và đảm bảo rằng bạn có nguồn lực và chuyên môn để phát triển và quản lý các thành phần có thể tái sử dụng một cách hiệu quả.

## Microfrontend

### Khái niệm

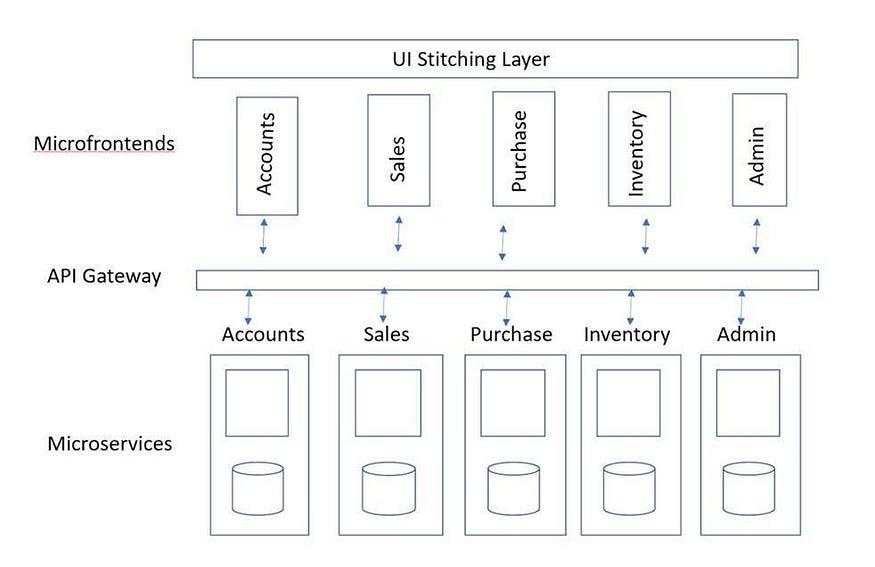
* Micro frontend là một kiến trúc ứng dụng web chia nhỏ ứng dụng thành các module hoặc chức năng riêng lẻ, được triển khai và quản lý độc lập. Giống như microservice ở phía backend, micro frontend mang lại nhiều lợi ích cho việc phát triển và bảo trì các ứng dụng web phức tạp.
* Kiến trúc micro frontend mở rộng các khái niệm của microservice sang thế giới frontend. Micro frontend là một kiểu kiến trúc trong đó một ứng dụng web được chia thành các module hoặc chức năng riêng lẻ khác nhau, được triển khai tự chủ, cho phép các nhóm frontend có cùng mức độ linh hoạt và tốc độ mà microservice cung cấp cho các nhóm backend.

### Cấu trúc hoạt động của mô hình microfrontend

* Mô hình Microfrontend hoạt động dựa trên việc chia nhỏ ứng dụng web thành các module hoặc chức năng riêng biệt, được gọi là Microfrontend. Mỗi Microfrontend được phát triển và triển khai độc lập, có thể sử dụng các công nghệ và framework khác nhau.

### Hoạt động của mô hình microfrontend

* Microfrontend: Mỗi Microfrontend là một ứng dụng web riêng lẻ, có thể được xây dựng bằng các công nghệ web khác nhau như HTML, CSS, JavaScript.
* Shell: Shell là ứng dụng web chính, chịu trách nhiệm tải và hiển thị các Microfrontend. Shell thường được xây dựng bằng một framework JavaScript như React, Angular hoặc Vue.js.
* Bộ định tuyến: Bộ định tuyến chịu trách nhiệm xác định Microfrontend nào cần được tải cho mỗi URL. Bộ định tuyến có thể được thực hiện bằng các thư viện JavaScript như Single-Spa, Bit hoặc Module Federation.
* Giao tiếp: Các Microfrontend có thể giao tiếp với nhau thông qua các phương thức như HTTP hoặc WebSocket. Giao tiếp này cho phép các Microfrontend chia sẻ dữ liệu và chức năng với nhau.

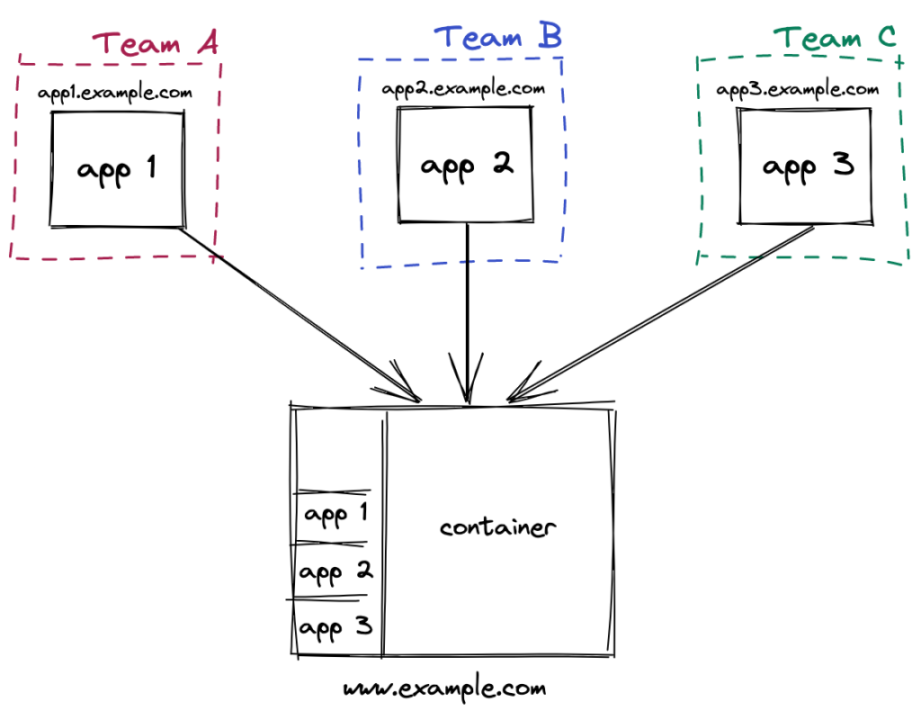


### Lợi ích của việc sử dụng mô hình microfrontend

* Tăng tốc độ phát triển: Các nhóm có thể làm việc song song trên các Microfrontend khác nhau, giúp đẩy nhanh tốc độ phát triển ứng dụng.
* Cải thiện khả năng bảo trì: Việc chia nhỏ ứng dụng thành các module nhỏ hơn giúp việc bảo trì và gỡ lỗi dễ dàng hơn.
* Tăng tính linh hoạt: Các Microfrontend có thể được dễ dàng thay đổi hoặc cập nhật mà không ảnh hưởng đến các phần khác của ứng dụng.
* Khả năng mở rộng: Microfrontend có thể dễ dàng được mở rộng bằng cách thêm các Microfrontend mới.

### Các công nghệ và framework sử dụng mô hình microfrontend

* React: Thư viện JavaScript phổ biến do Facebook phát triển, được sử dụng rộng rãi để xây dựng các giao diện người dùng (UI) component-based (dựa trên thành phần). React giúp tạo ra các UI linh hoạt, hiệu quả và dễ dàng bảo trì.
* Vue.js: Thư viện JavaScript nhẹ và linh hoạt khác để xây dựng giao diện người dùng. Vue.js dễ học và sử dụng, đồng thời cung cấp nhiều tính năng mạnh mẽ để phát triển các ứng dụng web hiện đại.
* Angular: Framework JavaScript do Google phát triển, cung cấp một bộ công cụ toàn diện để xây dựng các ứng dụng web một trang (SPA). Angular nổi tiếng với cấu trúc rõ ràng, tính ổn định và khả năng mở rộng cao, phù hợp cho các dự án lớn và phức tạp.
* Webpack module bundler:
  + Công cụ phổ biến để đóng gói các module JavaScript và tài nguyên tĩnh (static assets) thành các tập tin JavaScript tối ưu hóa cho trình duyệt. Webpack giúp đơn giản hóa việc quản lý các module trong các ứng dụng microfrontend phức tạp.
* Module federation:
  + Tính năng của Webpack giúp chia sẻ các module JavaScript giữa các ứng dụng microfrontend đang chạy trên cùng một trang web. Module federation giúp tối ưu hóa việc sử dụng mã và giảm thời gian tải trang.



* Single-spa:
  + Thư viện JavaScript giúp quản lý lifecycle (vòng đời) của các ứng dụng microfrontend trong một ứng dụng web duy nhất. Single-spa cung cấp các API để tải, khởi động và ngắt kết nối các microfrontend một cách hiệu quả.

A diagram of a web page

Description automatically generated

### Tổng kết

* Microfrontend là một mô hình phát triển web ngày càng phổ biến vì nó mang lại nhiều lợi ích cho các ứng dụng web phức tạp. Tuy nhiên, việc áp dụng microfrontend cũng có thể không thể thực hiện được, chẳng hạn như việc quản lý sự phức tạp của nhiều microfrontend và đảm bảo sự tương thích giữa các microfrontend.