**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA CÔNG NGHỆ PHẦN MỀM**





**Đồ án**

**Micro Front End**

**Giảng viên hướng dẫn :**

**Sinh viên thực hiện 1 : Trần Quang Huy**

**Mã sinh viên 1 : 21522169**

**Sinh viên thực hiện 2 : Ngô Hữu Thiên**

**Mã sinh viên 2 : 21522622**

**Lớp :**

**Tp HCM, tháng 11 năm 2024**

|  |  |
| --- | --- |
| **TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**  **KHOA CÔNG NGHỆ PHẦN MỀM**  **🙡★🙣** | **CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc lập – Tự do – Hạnh phúc**  **🙡★🙣** |

**NHIỆM VỤ ĐỒ ÁN**

**Họ và tên SV 1:** Trần Quang Huy MSSV: 21522169

**Họ và tên SV 2:** Ngô Hữu Thiên MSSV: 21522622

**Lớp:**

**Tên đề tài:**

**Giảng viên hướng dẫn:**

|  |  |
| --- | --- |
|  | *Tp.HCM, ngày tháng năm 2024*  GIẢNG VIÊN GIẢNG DẠY  *(Kí và ghi rõ họ tên)* |

LỜI CẢM ƠN

Sinh viên thực hiện

Trần Quang Huy

Ngô Hữu Thiên

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

|  |  |
| --- | --- |
|  | Tp.HCM, ngày … tháng 11 năm 2024  GVHD |

**MỤC LỤC**

Contents

[- Chương 1: TỔNG QUAN VỀ FRONTEND 1](#_Toc182463607)

[2.1. Thời kì desktop applications: 3](#_Toc182463608)

[2.2. Thời kì web applications: 4](#_Toc182463609)

[2.2.1. Thời kỳ JavaScript 5](#_Toc182463610)

[2.2.2. Thời kỳ của CSS: 7](#_Toc182463611)

[2.2.3. Cách mạng jQuery: 8](#_Toc182463612)

[2.3. Thời kỳ mobile applications: 9](#_Toc182463613)

[2.4. Thời kỳ cross platform applications: 11](#_Toc182463614)

[- Chương 2: CÁC KIẾN TRÚC PHỔ BIẾN 19](#_Toc182463615)

[1. Cross Platform vs Native: 19](#_Toc182463616)

[1.1 Khái niệm 19](#_Toc182463617)

[1.2 Cấu trúc mô hình Cross Platform vs Native 19](#_Toc182463618)

[1.3 Hoạt động của mô hình Cross Platform vs Native 19](#_Toc182463619)

[1.4 Lợi ích của mô hình Cross Platform vs Native 19](#_Toc182463620)

[1.5 Các ngôn ngữ và framework sử dụng mô hình Cross Platform vs Native 19](#_Toc182463621)

[1.6 Tổng kết 19](#_Toc182463622)

[2. SSR and Code Splitting: 19](#_Toc182463623)

[2.1 Khái niệm 19](#_Toc182463624)

[2.2 Cấu trúc của mô hình SSR and Code Splitting 20](#_Toc182463625)

[2.3 Hoạt động của mô hình SSR and Code Splitting 20](#_Toc182463626)

[2.5 Lợi ích của việc sử dụng mô hình SSR and Code Splitting 20](#_Toc182463627)

[2.6 Các ngôn ngữ và framework sử dụng mô hình SSR and Code Splitting 20](#_Toc182463628)

[2.7 Tổng kết 20](#_Toc182463629)

[3. Monolithic 20](#_Toc182463630)

[3.1 Khái niệm 20](#_Toc182463631)

[3.2 Cấu trúc của mô hình Monolithic 20](#_Toc182463632)

[3.3 Hoạt động của Monolithic 20](#_Toc182463633)

[3.4 Lợi ích của việc sử dụng mô hình Monolithic 20](#_Toc182463634)

[3.5 Các ngôn ngữ và framework sử dụng mô hình Monolithic 20](#_Toc182463635)

[3.6 Tổng kết 20](#_Toc182463636)

[4. Component-based 20](#_Toc182463637)

[4.1 Khái niệm 20](#_Toc182463638)

[4.2 Cấu trúc của mô hình component-based 20](#_Toc182463639)

[4.3 Hoạt động của mô hình component-based 22](#_Toc182463640)

[4.4 Lợi ích của việc sử dụng mô hình component-based 23](#_Toc182463641)

[4.5 Các ngôn ngữ và framework sử dụng mô hình component-based 23](#_Toc182463642)

[4.6 Tổng kết 24](#_Toc182463643)

[5. Microfrontend 24](#_Toc182463644)

[5.1 Khái niệm 24](#_Toc182463645)

[5.2 Cấu trúc hoạt động của mô hình microfrontend 24](#_Toc182463646)

[5.3 Hoạt động của mô hình microfrontend 24](#_Toc182463647)

[5.4 Lợi ích của việc sử dụng mô hình microfrontend 25](#_Toc182463648)

[5.5 Các công nghệ và framework sử dụng mô hình microfrontend 25](#_Toc182463649)

[5.6 Tổng kết 27](#_Toc182463650)

[- Chương 3: CÁC VẤN ĐỀ CƠ BẢN CỦA FRONTEND 28](#_Toc182463651)

[1. Hiển thị UI 28](#_Toc182463652)

[1.1. Định nghĩa 28](#_Toc182463653)

[1.2. Phương thức xử lý 28](#_Toc182463654)

[1.3. UI rendering trong React 28](#_Toc182463655)

[1.4 Tổng kết 29](#_Toc182463656)

[2. Tương tác UI 29](#_Toc182463657)

[2.1. Giải quyết vấn đề 29](#_Toc182463658)

[2.2. Tương tác trong React 29](#_Toc182463659)

[2.3. Tương tác trong Flutter 30](#_Toc182463660)

[3. Form validation 31](#_Toc182463661)

[3.1. Vấn đề 31](#_Toc182463662)

[3.2. Cách giải quyết 32](#_Toc182463663)

[3.3. Form Validation với React 33](#_Toc182463664)

[3.4. Tổng kết 33](#_Toc182463665)

[4. Tạo frontend động (dynamic form) 33](#_Toc182463666)

[4.1. Mô tả 33](#_Toc182463667)

[4.2. Công nghệ BEEKAI 34](#_Toc182463668)

[4.3. Công nghệ React-hook 35](#_Toc182463669)

[Chương 4: KHÁI QUÁT VỀ REACT 36](#_Toc182463670)

[1.1 Khái niệm: 36](#_Toc182463671)

[1.2 Lịch sử phát triển: 37](#_Toc182463672)

[1.3 Kiến trúc của hệ thống React Native: 37](#_Toc182463673)

[1.4 Ưu nhược điểm: 37](#_Toc182463674)

[1.4.1 Ưu điểm: 37](#_Toc182463675)

[1.4.2 Nhược điểm: 37](#_Toc182463676)

[1.5 Hiện trạng, tiềm năng 37](#_Toc182463677)

[o 1.6 Roadmap 37](#_Toc182463678)

[1.6.1 Điều kiện tiên quyết: 37](#_Toc182463679)

[1.6.2. Tài nguyên học tập: 37](#_Toc182463680)

[- Chương 5: MICROFRONTEND VỚI REACT 38](#_Toc182463681)

[o 1 Sử dụng Re.Pack: 38](#_Toc182463682)

[o 1.1 Khái niệm Re.Pack: 38](#_Toc182463683)

[- Chương 6: KẾT LUẬN 45](#_Toc182463684)

[o 1.1 Ưu điểm của đồ án: 45](#_Toc182463685)

[o 1.2 Hạn chế của đồ án: 45](#_Toc182463686)

[- TÀI LIỆU THAM KHẢO 47](#_Toc182463687)

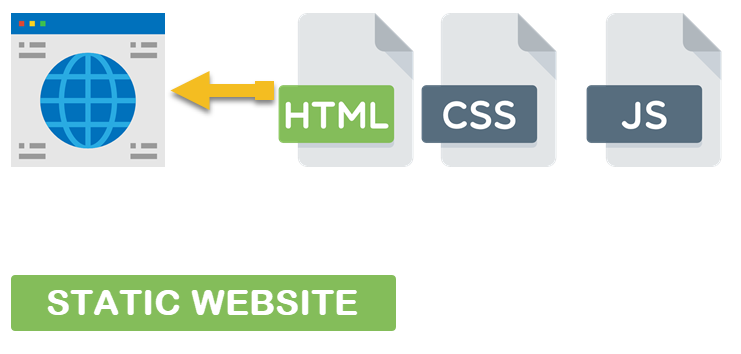
# Chương 1: TỔNG QUAN VỀ FRONTEND

1. **Frontend là gì:**

* Front-end là phần giao diện người dùng của một trang web hoặc ứng dụng di động mà người dùng có thể nhìn thấy và tương tác trực tiếp. Nó bao gồm tất cả các yếu tố trực quan như bố cục, màu sắc, phông chữ, hình ảnh, nút bấm, menu, v.v.
* Nói cách khác, front-end là "mặt tiền" của một trang web hoặc ứng dụng, là thứ đầu tiên mà người dùng nhìn thấy và trải nghiệm. Nó đóng vai trò quan trọng trong việc thu hút người dùng và tạo ấn tượng ban đầu tốt đẹp.
* Công việc chính của lập trình viên front-end là sử dụng các ngôn ngữ lập trình như HTML, CSS và JavaScript để xây dựng và duy trì giao diện người dùng. Họ cũng cần có kiến thức về các công nghệ web khác như frameworks, libraries và tools để tạo ra các trang web và ứng dụng web hiện đại, hiệu quả và tương thích với nhiều trình duyệt khác nhau.

1. **Lịch sử phát triển:**

* Lĩnh vực front-end đã trải qua nhiều giai đoạn phát triển từ khi ra đời cho đến nay, với sự ra đời của các công nghệ mới và sự thay đổi trong nhu cầu của người dùng. Dưới đây là một số mốc quan trọng trong lịch sử phát triển của front-end:
* Giai đoạn đầu (1990 - 1995):
* HTML là ngôn ngữ duy nhất được sử dụng để xây dựng trang web.
* Trang web chủ yếu là văn bản và hình ảnh tĩnh.
* Tương tác của người dùng bị hạn chế.



* Giai đoạn Web 1.0 (1995 - 2004):
* JavaScript được giới thiệu, cho phép tạo ra các trang web tương tác hơn.
* CSS được phát triển để tách biệt nội dung và giao diện của trang web.
* Các trình duyệt web bắt đầu hỗ trợ nhiều tính năng mới như frames và cookies.



* Giai đoạn Web 2.0 (2004 - 2010):
  + Sự ra đời của các ứng dụng web (web application) như Gmail, Facebook và YouTube.
  + AJAX (Asynchronous JavaScript and XML) được sử dụng để tạo ra các ứng dụng web động và tương tác hơn.
  + Các frameworks JavaScript đầu tiên như jQuery ra đời, giúp đơn giản hóa việc viết mã JavaScript.

A blue background with arrows pointing to a globe

Description automatically generated

* Giai đoạn Web 3.0 (2010 - nay):
  + Sự bùng nổ của các thiết bị di động như smartphone và tablet.
  + HTML5 và CSS3 được phát triển để hỗ trợ các tính năng mới cho thiết bị di động.
  + Các frameworks JavaScript hiện đại như React, Angular và Vue.js ra đời, giúp xây dựng các ứng dụng web phức tạp và hiệu quả hơn.
  + Mobile-first trở thành xu hướng thiết kế web, ưu tiên trải nghiệm người dùng trên thiết bị di động.

A blue background with icons and a person wearing virtual reality glasses

Description automatically generated

## Thời kì desktop applications:

* Thời kỳ Desktop Applications (Ứng dụng Máy Tính) bắt đầu từ những năm 1970 với sự ra đời của các máy tính cá nhân như Apple II, Commodore PET và IBM PC. Các ứng dụng máy tính thời kỳ này thường được viết bằng các ngôn ngữ lập trình như C, Pascal và BASIC, và được phân phối trên đĩa mềm hoặc CD-ROM.
* Đặc điểm nổi bật của thời kỳ Desktop Applications:
  + Tính độc lập: Ứng dụng máy tính chạy độc lập trên máy tính của người dùng, không cần kết nối internet.
  + Giao diện đơn giản: Giao diện người dùng (GUI) của các ứng dụng máy tính thời kỳ này thường đơn giản và dễ sử dụng.
  + Chức năng hạn chế: Các ứng dụng máy tính thời kỳ này thường có chức năng hạn chế so với các ứng dụng web hiện đại.
  + Cài đặt và gỡ cài đặt: Ứng dụng máy tính cần được cài đặt trên máy tính của người dùng trước khi sử dụng và có thể được gỡ cài đặt nếu không còn cần thiết.
* Lợi ích của Desktop Applications:
  + Hiệu suất cao: Ứng dụng máy tính thường có hiệu suất cao hơn so với các ứng dụng web, vì chúng không cần tải dữ liệu từ máy chủ.
  + Bảo mật: Ứng dụng máy tính được cài đặt trên máy tính của người dùng, giúp bảo vệ dữ liệu khỏi những kẻ tấn công bên ngoài.
  + Không phụ thuộc vào internet: Ứng dụng máy tính có thể hoạt động mà không cần kết nối internet.
* Tuy nhiên, Desktop Applications cũng có một số hạn chế:
  + Khả năng tương thích: Ứng dụng máy tính chỉ có thể hoạt động trên hệ điều hành và nền tảng cụ thể.
  + Cập nhật: Cập nhật ứng dụng máy tính thường phức tạp hơn so với cập nhật ứng dụng web.
  + Chi phí: Ứng dụng máy tính thường có giá cao hơn so với ứng dụng web.

## Thời kì web applications:

* Thời kỳ Web Applications (Ứng dụng Web) được đánh dấu bởi sự ra đời và phát triển mạnh mẽ của các ứng dụng web, thay thế cho các trang web tĩnh truyền thống. Nó bắt đầu từ khoảng đầu những năm 2000 và kéo dài đến nay, với nhiều giai đoạn phát triển và biến đổi.
* Một số mốc phát triển quan trong trong thời kỳ Web Applications:
  + Đầu những năm 2000: Xuất hiện các ứng dụng web đầu tiên như Gmail, Yahoo Mail, Hotmail.
  + Giữa những năm 2000: Phát triển mạnh mẽ của các ứng dụng web 2.0 như Facebook, Twitter, YouTube.
  + Cuối những năm 2000: Nổi lên các ứng dụng web di động (mobile web applications) cho phép người dùng truy cập trên điện thoại thông minh và máy tính bảng.
  + Hiện nay: Web Applications tiếp tục phát triển với tốc độ nhanh chóng, ứng dụng vào nhiều lĩnh vực như thương mại điện tử, giáo dục, y tế, v.v.
* Đặc điểm nổi bật của thời kỳ Web Applications:
  + Sự tương tác: Ứng dụng web cho phép người dùng tương tác trực tiếp với trang web, thay vì chỉ xem nội dung tĩnh.
  + Tính năng động: Ứng dụng web có thể cập nhật dữ liệu và thay đổi giao diện theo thời gian thực, mang lại trải nghiệm người dùng mượt mà và sống động hơn.
  + Tính phức tạp: Ứng dụng web có thể thực hiện nhiều chức năng phức tạp như xử lý dữ liệu, thanh toán trực tuyến, chat trực tiếp, v.v.
  + Khả năng tùy chỉnh: Ứng dụng web thường có thể được tùy chỉnh theo nhu cầu của người dùng, giúp nâng cao trải nghiệm người dùng.
* Lợi ích của Web Applications:
  + Tiết kiệm chi phí: Ứng dụng web không cần cài đặt phần mềm, có thể truy cập từ mọi nơi trên thế giới chỉ cần có kết nối internet.
  + Dễ dàng cập nhật: Ứng dụng web có thể được cập nhật dễ dàng và nhanh chóng mà không cần người dùng phải tải xuống phiên bản mới.
  + Khả năng mở rộng: Ứng dụng web có thể dễ dàng mở rộng để đáp ứng nhu cầu ngày càng tăng của người dùng.
  + Tính tương thích: Ứng dụng web có thể hoạt động trên nhiều trình duyệt web và thiết bị khác nhau.
* Tuy nhiên, Web Applications cũng có một số hạn chế:
  + Phụ thuộc vào kết nối internet: Ứng dụng web cần có kết nối internet để hoạt động.
  + Bảo mật: Ứng dụng web có thể dễ bị tấn công hơn so với phần mềm được cài đặt trên máy tính.
  + Hiệu suất: Hiệu suất của ứng dụng web có thể bị ảnh hưởng bởi tốc độ kết nối internet.

## Thời kỳ JavaScript

* Thời kỳ JavaScript bắt đầu từ khi ngôn ngữ lập trình JavaScript được phát triển vào năm 1995 bởi Brendan Eich tại Netscape. Kể từ đó, JavaScript đã trải qua nhiều giai đoạn phát triển và trở thành một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới, đóng vai trò quan trọng trong việc xây dựng các trang web và ứng dụng web hiện đại.
* Một số mốc quan trọng trong thời kỳ JavaScript:
  + 1995: Ra đời JavaScript, ban đầu được gọi là LiveScript.
  + 1997: JavaScript được ECMA International chuẩn hóa thành ECMAScript.
  + 1999: Internet Explorer hỗ trợ JavaScript.
  + 2005: XMLHttpRequest (XHR) được giới thiệu, cho phép JavaScript truy cập dữ liệu từ máy chủ mà không cần tải lại trang.
  + 2006: jQuery ra đời, giúp đơn giản hóa việc viết mã JavaScript.
  + 2008: Node.js ra đời, cho phép JavaScript chạy trên máy chủ.
  + 2010: HTML5 và CSS3 được phát triển, mở ra nhiều khả năng mới cho JavaScript.
  + 2013: React ra đời, trở thành một trong những thư viện JavaScript phổ biến nhất.
  + 2015: Angular ra đời, một framework JavaScript khác được ưa chuộng.
  + 2016: Vue.js ra đời, một framework JavaScript nhẹ và linh hoạt.
  + Hiện nay: JavaScript tiếp tục phát triển với tốc độ nhanh chóng, được sử dụng để xây dựng các ứng dụng web phức tạp, single-page applications (SPA), mobile applications và nhiều hơn nữa.
* Lý do JavaScript trở nên phổ biến:
  + Dễ học và sử dụng: JavaScript có cú pháp tương đối đơn giản và dễ học, ngay cả với những người mới bắt đầu.
  + Linh hoạt: JavaScript có thể được sử dụng để thực hiện nhiều tác vụ khác nhau, từ thao tác DOM (Document Object Model) đến xử lý dữ liệu và tạo hiệu ứng đồ họa.
  + Cộng đồng lớn: JavaScript có cộng đồng người dùng và nhà phát triển lớn, giúp bạn dễ dàng tìm kiếm sự trợ giúp và tài nguyên khi cần thiết.
  + Hỗ trợ đa nền tảng: JavaScript được hỗ trợ bởi tất cả các trình duyệt web phổ biến, cho phép bạn xây dựng các ứng dụng web có thể hoạt động trên nhiều thiết bị khác nhau.
* Tác động của JavaScript:
  + Thay đổi cách thức hoạt động của web: JavaScript đã biến các trang web trở nên năng động và tương tác hơn, cho phép người dùng thực hiện nhiều tác vụ mà không cần tải lại trang.
  + Mở ra cánh cửa cho các ứng dụng web phức tạp: JavaScript đã giúp tạo ra các ứng dụng web phức tạp như Gmail, Facebook, Netflix và nhiều hơn nữa.
  + Thúc đẩy sự phát triển của các công nghệ mới: JavaScript đã thúc đẩy sự phát triển của các công nghệ mới như Node.js, React, Angular và Vue.js, mở ra nhiều khả năng mới cho các nhà phát triển web.

## Thời kỳ của CSS:

* Thời kỳ CSS bắt đầu từ khi ngôn ngữ CSS được phát triển vào năm 1996 bởi Håkon Wium Lie tại Netscape. Kể từ đó, CSS đã trải qua nhiều giai đoạn phát triển và trở thành một ngôn ngữ quan trọng trong việc thiết kế giao diện người dùng (UI) cho các trang web và ứng dụng web.
* Các mốc quan trong trong thời kì CSS:
  + 1996: Ra đời CSS, ban đầu được gọi là Cascading Style Sheets.
  + 1997: CSS được W3C (World Wide Web Consortium) chuẩn hóa thành CSS1.
  + 1999: CSS2 được W3C phát hành, bổ sung nhiều tính năng mới cho CSS.
  + 2000s: CSS được sử dụng rộng rãi hơn trong thiết kế web, giúp tạo ra các trang web đẹp mắt và dễ sử dụng hơn.
  + 2009: CSS3 được W3C phát hành, giới thiệu nhiều tính năng mới như bố cục flexbox, grid layout, media queries và animations.
  + 2010s: CSS preprocessors như Sass và LESS được sử dụng phổ biến, giúp đơn giản hóa việc viết mã CSS.
  + Hiện nay: CSS tiếp tục phát triển với tốc độ nhanh chóng, với nhiều tính năng mới được bổ sung và các công cụ hỗ trợ ngày càng tiên tiến.
* Lý do CSS trở nên phổ biến:
  + Dễ học và sử dụng: CSS có cú pháp tương đối đơn giản và dễ học, ngay cả với những người mới bắt đầu.
  + Linh hoạt: CSS có thể được sử dụng để tạo ra nhiều kiểu dáng khác nhau cho các trang web, từ bố cục trang đến màu sắc, phông chữ và hiệu ứng.
  + Tiết kiệm thời gian: CSS giúp tiết kiệm thời gian cho các nhà phát triển web bằng cách cho phép họ định dạng nhiều trang web cùng một lúc.
  + Tăng hiệu suất: CSS giúp cải thiện hiệu suất trang web bằng cách giảm thiểu số lượng mã HTML cần thiết để định dạng trang.
* Tác động của CSS:
  + Thay đổi cách thức thiết kế web: CSS đã thay đổi cách thức thiết kế web bằng cách tách biệt nội dung và giao diện của trang web.
  + Tạo ra các trang web đẹp mắt và dễ sử dụng: CSS giúp tạo ra các trang web đẹp mắt và dễ sử dụng hơn, mang lại trải nghiệm tốt hơn cho người dùng.
  + Mở ra cánh cửa cho các thiết kế web phức tạp: CSS đã mở ra cánh cửa cho các thiết kế web phức tạp hơn, với nhiều hiệu ứng và tương tác hơn.

## Cách mạng jQuery:

* jQuery, ra đời vào năm 2006, đánh dấu một bước ngoặt trong lịch sử phát triển front-end, mang đến cuộc cách mạng cho cách thức xây dựng và tương tác với các trang web. Trước jQuery, việc viết mã JavaScript cho các tác vụ thông thường như thao tác DOM, xử lý sự kiện và AJAX có thể tốn thời gian và phức tạp. jQuery xuất hiện như một giải pháp đơn giản và hiệu quả, giúp đơn giản hóa việc viết mã JavaScript và nâng cao năng suất của các nhà phát triển web.
* Tác động của jQuery đối với việc phát triển frontend:
  + Đơn giản hóa mã JavaScript: jQuery cung cấp một bộ API phong phú với các hàm được viết sẵn, giúp các nhà phát triển thực hiện các tác vụ JavaScript thông thường một cách nhanh nhất.
  + Tăng tốc độ phát triển: Việc sử dụng jQuery giúp tiết kiệm thời gian viết mã, cho phép các nhà phát triển xây dựng trang web nhanh hơn và hiệu quả hơn.
  + Tăng tính tương thích: jQuery hỗ trợ nhiều trình duyệt web khác nhau, giúp đảm bảo website hoạt động tốt trên nhiều thiết bị và nền tảng.
  + Tạo ra cộng đồng lớn: jQuery sở hữu cộng đồng người dùng và nhà phát triển rộng lớn, cung cấp nguồn tài nguyên phong phú và hỗ trợ tích cực.
* Lợi ích nổi bật của jQuery:
  + Dễ học và sử dụng: Cú pháp jQuery đơn giản và dễ hiểu, giúp người mới bắt đầu dễ dàng tiếp cận và sử dụng.
  + Linh hoạt và mạnh mẽ: jQuery cung cấp nhiều hàm và phương thức để thực hiện đa dạng các tác vụ front-end.
  + Hỗ trợ đa dạng: jQuery hỗ trợ nhiều trình duyệt web và thư viện JavaScript khác, giúp dễ dàng tích hợp với các dự án hiện có.
  + Cộng đồng lớn và hoạt động: Cộng đồng jQuery luôn sẵn sàng hỗ trợ và chia sẻ kiến thức, giúp giải quyết vấn đề và nâng cao kỹ năng.
* Một số hạn chế của jQuery:
  + Kích thước: jQuery có kích thước tương đối lớn, có thể ảnh hưởng đến tốc độ tải trang.
  + Có thể dẫn đến mã lộn xộn: Việc sử dụng jQuery quá mức có thể dẫn đến mã JavaScript khó đọc và bảo trì.
  + Cập nhật ít thường xuyên: jQuery không được cập nhật thường xuyên như các thư viện JavaScript hiện đại khác.

## Thời kỳ mobile applications:

* Thời kỳ Mobile Applications (Ứng dụng Di động) đánh dấu một bước ngoặt trong lịch sử công nghệ, thay đổi hoàn toàn cách thức chúng ta sử dụng điện thoại thông minh và máy tính bảng. Từ những ứng dụng đơn giản ban đầu, ứng dụng di động đã phát triển thành những công cụ mạnh mẽ, đa dạng, đóng vai trò quan trọng trong mọi khía cạnh của cuộc sống.
* Lịch sử phát triển:
  + Khởi đầu:
    - Cuối những năm 1990s: Xuất hiện những ứng dụng di động đầu tiên trên nền tảng Java ME, Symbian OS và BlackBerry OS.
    - 2007: Apple ra mắt iPhone và App Store, mở ra kỷ nguyên mới cho ứng dụng di động iOS.
    - 2008: Google ra mắt Android và Google Play Store, thúc đẩy sự phát triển mạnh mẽ của ứng dụng di động đa nền tảng.
  + Phát triển bùng nổ:
    - 2010s: Nhu cầu sử dụng ứng dụng di động tăng cao, dẫn đến sự bùng nổ về số lượng và chất lượng ứng dụng.
    - Xuất hiện nhiều công nghệ mới như HTML5, CSS3 và JavaScript, giúp việc phát triển ứng dụng di động trở nên dễ dàng và hiệu quả hơn.
    - Các cửa hàng ứng dụng chính thức như App Store và Google Play Store trở thành trung tâm phân phối ứng dụng di động lớn nhất thế giới.
  + Hiện tại và tương lai:
    - Ứng dụng di động đã trở thành một phần không thể thiếu trong cuộc sống của chúng ta, với hàng tỷ người sử dụng trên toàn thế giới.
    - Trí tuệ nhân tạo (AI), Machine Learning (ML), Thực tế ảo (VR), Thực tế tăng cường (AR), Blockchain,... được ứng dụng rộng rãi vào ứng dụng di động, mang đến những trải nghiệm mới mẻ và đột phá.
    - Xu hướng phát triển tập trung vào ứng dụng di động đa nền tảng, bảo mật cao, thân thiện với môi trường và hỗ trợ người khuyết tật.
* Tác động to lớn:
  + Thay đổi cách thức kết nối: Ứng dụng di động giúp chúng ta dễ dàng kết nối với bạn bè, gia đình và đồng nghiệp mọi lúc mọi nơi.
  + Mở ra cơ hội kinh doanh mới: Ứng dụng di động tạo ra nhiều cơ hội kinh doanh mới cho các doanh nghiệp, giúp tiếp cận khách hàng tiềm năng và tăng doanh thu.
  + Cải thiện chất lượng cuộc sống: Ứng dụng di động giúp cải thiện chất lượng cuộc sống trong nhiều lĩnh vực như y tế, giáo dục, giao thông vận tải,...
  + Thúc đẩy đổi mới sáng tạo: Ứng dụng di động là nền tảng cho nhiều ý tưởng sáng tạo mới, thúc đẩy sự phát triển của khoa học kỹ thuật.
* Một số thách thức:
  + Bảo mật và quyền riêng tư: Việc sử dụng ứng dụng di động tiềm ẩn nguy cơ bị đánh cắp dữ liệu cá nhân và thông tin nhạy cảm.
  + Nghiện ứng dụng di động: Việc sử dụng ứng dụng di động quá mức có thể ảnh hưởng đến sức khỏe và tinh thần của người dùng.
  + Bất bình đẳng kỹ thuật số: Không phải ai cũng có thể tiếp cận và sử dụng ứng dụng di động do hạn chế về cơ sở hạ tầng và khả năng tài chính.

## Thời kỳ cross platform applications:

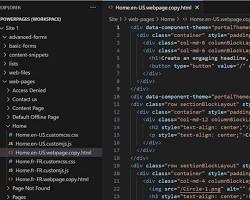
* Ứng dụng Cross-Platform (Ứng dụng đa nền tảng) đang ngày càng trở nên phổ biến trong thời đại ngày nay bởi những ưu điểm vượt trội so với các ứng dụng Native truyền thống. Dưới đây là một số lý do chính cho sự phát triển mạnh mẽ của xu hướng này:
  + Tiết kiệm chi phí và thời gian phát triển:
    - Việc phát triển một ứng dụng Native cho mỗi hệ điều hành riêng biệt đòi hỏi nhiều nguồn lực về nhân lực, tài chính và thời gian. Ứng dụng Cross-Platform giúp giải quyết vấn đề này bằng cách sử dụng một bộ mã duy nhất để xây dựng ứng dụng cho nhiều nền tảng khác nhau, từ đó tiết kiệm chi phí và thời gian phát triển đáng kể.
  + Mở rộng thị trường tiềm năng:
    - Thị trường di động ngày nay vô cùng đa dạng với nhiều hệ điều hành phổ biến như Android, iOS, Windows Phone, v.v. Việc phát triển ứng dụng Cross-Platform giúp doanh nghiệp tiếp cận được với lượng lớn người dùng trên nhiều nền tảng khác nhau, mở rộng thị trường tiềm năng và gia tăng cơ hội kinh doanh.
  + Dễ dàng bảo trì và cập nhật:
    - Việc quản lý và cập nhật một bộ mã duy nhất cho nhiều nền tảng sẽ dễ dàng và hiệu quả hơn so với việc quản lý nhiều phiên bản Native riêng biệt. Điều này giúp doanh nghiệp tiết kiệm thời gian và nguồn lực cho công tác bảo trì và cập nhật ứng dụng.
  + Nâng cao trải nghiệm người dùng:
    - Ứng dụng Cross-Platform được thiết kế để mang lại trải nghiệm người dùng nhất quán trên mọi nền tảng, đảm bảo sự mượt mà và tiện lợi cho người dùng bất kể họ sử dụng thiết bị nào.
  + Các công nghệ Cross-Platform tiên tiến:
    - Sự phát triển mạnh mẽ của các công nghệ Cross-Platform như React Native, Flutter, Xamarin, v.v. đã giúp việc xây dựng ứng dụng Cross-Platform trở nên dễ dàng và hiệu quả hơn bao giờ hết. Các công nghệ này cung cấp nhiều tính năng và khả năng mạnh mẽ, giúp các nhà phát triển tạo ra những ứng dụng đa nền tảng chất lượng cao.
  1. **Thời kì progressive web application**
* Progressive Web Applications (PWA) là một công nghệ web mới nổi đang nhanh chóng thu hút sự chú ý của giới lập trình viên và doanh nghiệp. PWA được xem là sự kết hợp hoàn hảo giữa website và ứng dụng di động, mang đến cho người dùng trải nghiệm mượt mà và tiện lợi như ứng dụng Native mà không cần tải xuống.
* Lý do cho sự bùng nổ của PWA:
  + Nâng cao trải nghiệm người dùng: PWA mang đến tốc độ tải trang nhanh chóng, khả năng hoạt động ngoại tuyến, khả năng đẩy thông báo và giao diện người dùng thân thiện, từ đó mang lại trải nghiệm mượt mà và tiện lợi cho người dùng.
  + Tiết kiệm chi phí phát triển: Do sử dụng công nghệ web nên PWA có thể được phát triển với chi phí thấp hơn so với ứng dụng Native.
  + Dễ dàng bảo trì và cập nhật: Việc quản lý và cập nhật một bộ mã duy nhất cho PWA sẽ dễ dàng và hiệu quả hơn so với việc quản lý nhiều phiên bản Native riêng biệt.
  + Tương thích với nhiều nền tảng: PWA có thể hoạt động trên mọi trình duyệt web và thiết bị di động, giúp doanh nghiệp tiếp cận được với lượng lớn người dùng tiềm năng.
  + SEO tốt hơn: PWA được tối ưu hóa cho công cụ tìm kiếm, giúp doanh nghiệp dễ dàng tiếp cận khách hàng tiềm năng hơn.
* Xu hướng tương lai:
  + Với những ưu điểm vượt trội, PWA được dự đoán sẽ trở thành xu hướng chủ đạo trong phát triển web trong tương lai. Các công nghệ liên quan đến PWA sẽ ngày càng được cải tiến và hoàn thiện, giúp việc xây dựng PWA trở nên dễ dàng, hiệu quả và tiết kiệm chi phí hơn. Doanh nghiệp sẽ ngày càng ưa chuộng sử dụng PWA để mang lại trải nghiệm tốt nhất cho người dùng và tăng cường khả năng cạnh tranh trên thị trường.
  1. **Thời kỳ của responsive design**
* Thiết kế Phản hồi (Responsive Design) đã và đang định hình thế giới web trong hơn một thập kỷ qua, trở thành tiêu chuẩn thiết yếu cho các trang web hiện đại. Xu hướng này mang đến những lợi ích to lớn cho cả người dùng và doanh nghiệp, dẫn đến sự áp dụng rộng rãi và ngày càng phổ biến.
* Lý do bùng nổ của Thiết kế Phản hồi:
  + Sự gia tăng truy cập web từ thiết bị di động: Nhu cầu truy cập web từ điện thoại thông minh và máy tính bảng không ngừng gia tăng, đòi hỏi các trang web phải thích ứng linh hoạt với nhiều kích thước màn hình khác nhau. Thiết kế Phản hồi đáp ứng hoàn hảo nhu cầu này, đảm bảo trải nghiệm người dùng nhất quán trên mọi thiết bị.
  + Tối ưu hóa trải nghiệm người dùng: Thiết kế Phản hồi chú trọng vào việc tạo ra giao diện dễ sử dụng, điều hướng mượt mà và hiển thị nội dung phù hợp trên mọi kích thước màn hình. Nhờ vậy, người dùng có thể truy cập và tương tác với trang web một cách dễ dàng và hiệu quả, bất kể họ sử dụng thiết bị nào.
  + Tiết kiệm chi phí và thời gian phát triển: Thay vì phải tạo ra các trang web riêng biệt cho từng thiết bị, Thiết kế Phản hồi cho phép xây dựng một trang web duy nhất có thể thích ứng với mọi kích thước màn hình. Điều này giúp tiết kiệm đáng kể chi phí và thời gian phát triển, quản lý và bảo trì trang web.
  + Tăng khả năng SEO: Google và các công cụ tìm kiếm khác ưu tiên các trang web thân thiện với thiết bị di động. Thiết kế Phản hồi giúp tối ưu hóa trang web cho SEO, nâng cao thứ hạng tìm kiếm và thu hút nhiều truy cập hơn.
  + Nâng cao hình ảnh thương hiệu: Thiết kế Phản hồi thể hiện sự chuyên nghiệp và uy tín của doanh nghiệp, mang đến hình ảnh thương hiệu nhất quán và ấn tượng trên mọi thiết bị.
* Xu hướng tương lai:

Thiết kế Phản hồi sẽ tiếp tục phát triển và hoàn thiện trong tương lai, song hành cùng sự phát triển của công nghệ web và nhu cầu ngày càng cao của người dùng. Một số xu hướng nổi bật bao gồm:

* + Thiết kế đáp ứng linh hoạt: Các trang web sẽ có khả năng thích ứng với nhiều kích thước màn hình hơn, từ màn hình lớn của máy tính để bàn đến màn hình gập và thiết bị đeo.
  + Trải nghiệm người dùng được cá nhân hóa: Thiết kế Phản hồi sẽ kết hợp với dữ liệu người dùng để cá nhân hóa trải nghiệm, cung cấp nội dung và chức năng phù hợp với từng cá nhân.
  + Sử dụng trí tuệ nhân tạo (AI): AI sẽ được ứng dụng để tự động tối ưu hóa giao diện và trải nghiệm người dùng cho từng thiết bị và ngữ cảnh truy cập.
  + Sự phát triển của các công nghệ web mới: Thiết kế Phản hồi sẽ tích hợp các công nghệ web mới như WebAssembly và Progressive Web Apps (PWA) để nâng cao hiệu suất và khả năng của trang web.

1. **Công nghệ frontend**
   1. **Các công cụ**

* Trình chỉnh sửa mã: Trình chỉnh sửa mã là cần thiết để viết mã HTML, CSS và JavaScript. Một số trình chỉnh sửa mã phổ biến bao gồm Visual Studio Code, Sublime Text và Atom.



* Công cụ gỡ lỗi: Công cụ gỡ lỗi cho phép bạn gỡ lỗi mã của mình và xác định các vấn đề. Một số công cụ gỡ lỗi phổ biến bao gồm Chrome DevTools, Firefox Developer Tools và Safari Web Inspector.

A screenshot of a computer

Description automatically generated

* Công cụ kiểm tra: Công cụ kiểm tra cho phép bạn kiểm tra trang web của mình trên các trình duyệt và thiết bị khác nhau. Một số công cụ kiểm tra phổ biến bao gồm BrowserStack, Sauce Labs và Selenium.
* Công cụ xử lý trước: Công cụ xử lý trước cho phép bạn xử lý mã của mình trước khi nó được trình duyệt tải. Điều này có thể hữu ích cho các tác vụ như tổng hợp LESS và SCSS thành CSS và nén tệp JavaScript. Một số công cụ xử lý trước phổ biến bao gồm Gulp, Grunt và Webpack.
* Công cụ quản lý gói: Công cụ quản lý gói cho phép bạn quản lý các phụ thuộc của dự án của mình. Một số công cụ quản lý gói phổ biến bao gồm npm và yarn.



* Công cụ kiểm soát phiên bản: Công cụ kiểm soát phiên bản cho phép bạn theo dõi thay đổi đối với mã của mình. Một số công cụ kiểm soát phiên bản phổ biến bao gồm Git và Mercurial.



* 1. **Các ngôn ngữ lập trình**
* HTML (HyperText Markup Language): Là ngôn ngữ nền tảng để tạo cấu trúc và nội dung cho trang web. HTML sử dụng các thẻ để định nghĩa các phần khác nhau của trang như tiêu đề, đoạn văn bản, hình ảnh, liên kết, v.v.



* CSS (Cascading Style Sheets): Được sử dụng để định dạng giao diện của trang web, bao gồm màu sắc, bố cục, kiểu chữ, v.v. CSS giúp bạn tạo ra các trang web đẹp mắt và thu hút người dùng.



* JavaScript: Mang đến khả năng tương tác cho trang web, cho phép bạn tạo các hiệu ứng động, xử lý sự kiện người dùng, thao tác với dữ liệu và giao tiếp với máy chủ. JavaScript đóng vai trò quan trọng trong việc tạo ra các trang web và ứng dụng web hiện đại, mượt mà và năng động.

A yellow sign with a black text

Description automatically generated

1. **Các frontend framework**
   1. **Định nghĩa**

* Frontend frameword là một bộ công cụ, thư viện và quy ước được thiết kế để hỗ trợ việc xây dựng các giao diện người dùng (UI) và ứng dụng web hiện đại. Framework front-end cung cấp một cấu trúc cơ bản và các tính năng sẵn có, giúp các nhà phát triển tiết kiệm thời gian và công sức, đồng thời tạo ra các ứng dụng web chất lượng cao, hiệu quả và dễ dàng bảo trì.
  1. **Các framework frontend phổ biến**
* React:
  + Được Facebook phát triển và ưa chuộng bởi sự linh hoạt, hiệu quả và dễ học.
  + Sử dụng mô hình component-based (dựa trên thành phần), giúp tái sử dụng code dễ dàng và bảo trì ứng dụng hiệu quả.
  + Cộng đồng người dùng lớn và nhiều tài liệu hướng dẫn, hỗ trợ đắc lực cho người mới bắt đầu.
  + Thư viện mở rộng phong phú, đáp ứng đa dạng nhu cầu phát triển.
* Vue.js:
  + Framework JavaScript nhẹ và dễ học, nổi tiếng với cú pháp đơn giản và trực quan.
  + Cung cấp nhiều tính năng mạnh mẽ để xây dựng các ứng dụng web hiện đại.
  + Tính linh hoạt cao, dễ dàng tích hợp với các thư viện và công cụ khác.
  + Cộng đồng người dùng đang phát triển nhanh chóng và có nhiều tài liệu hướng dẫn hữu ích.
* Angular:
  + Framework do Google phát triển, cung cấp bộ công cụ toàn diện để xây dựng ứng dụng web một trang (SPA).
  + Cấu trúc rõ ràng, ổn định và có khả năng mở rộng cao, phù hợp cho các dự án lớn và phức tạp.
  + Sử dụng TypeScript, giúp viết code an toàn và dễ bảo trì hơn.
  + Cộng đồng người dùng lớn và nhiều tài liệu hướng dẫn chi tiết.
* Bootstrap:
  + Framework CSS phổ biến để xây dựng giao diện người dùng responsive (phản hồi).
  + Cung cấp nhiều thành phần UI đã được tạo sẵn như nút, biểu mẫu, bảng, v.v., giúp dễ dàng tạo ra các trang web đẹp mắt và dễ sử dụng trên mọi thiết bị.
  + Dễ học và sử dụng, phù hợp cho cả người mới bắt đầu và lập trình viên front-end.
  + Cộng đồng người dùng lớn và nhiều tài liệu hướng dẫn chi tiết.
* Ngoài ra, còn có nhiều framework front-end khác cũng được sử dụng rộng rãi như:
  + jQuery: Thư viện JavaScript phổ biến để thao tác với DOM (Document Object Model), xử lý sự kiện và thực hiện các tác vụ AJAX.
  + Ember.js: Framework JavaScript mạnh mẽ để xây dựng các ứng dụng web một trang phức tạp.
  + Backbone.js: Framework JavaScript nhẹ và linh hoạt để xây dựng các ứng dụng web single-page (SPA).
  + Semantic UI: Framework CSS dựa trên Bootstrap, cung cấp nhiều thành phần UI đẹp mắt và dễ sử dụng.
  + Foundation: Framework CSS responsive (phản hồi) khác, cung cấp nhiều tính năng mạnh mẽ để xây dựng các giao diện người dùng hiện đại.

# Chương 2: CÁC KIẾN TRÚC PHỔ BIẾN

## Cross Platform vs Native:

### Khái niệm

### Cấu trúc mô hình Cross Platform vs Native

### Hoạt động của mô hình Cross Platform vs Native

### Lợi ích của mô hình Cross Platform vs Native

### Các ngôn ngữ và framework sử dụng mô hình Cross Platform vs Native

### 1.6 Tổng kết

-.

## SSR and Code Splitting:

### Khái niệm

* .

### Cấu trúc của mô hình SSR and Code Splitting

### Hoạt động của mô hình SSR and Code Splitting

### Lợi ích của việc sử dụng mô hình SSR and Code Splitting

### Các ngôn ngữ và framework sử dụng mô hình SSR and Code Splitting

### Tổng kết

## Monolithic

### Khái niệm

### Cấu trúc của mô hình Monolithic

### Hoạt động của Monolithic

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

### Các ngôn ngữ và framework sử dụng mô hình Monolithic

### Tổng kết

* MVVM là một mô hình thiết kế mạnh mẽ và linh hoạt, được sử dụng rộng rãi trong phát triển ứng dụng hiện đại. Việc áp dụng MVVM giúp tạo ra các ứng dụng dễ dàng bảo trì, mở rộng, kiểm tra và có hiệu suất cao.

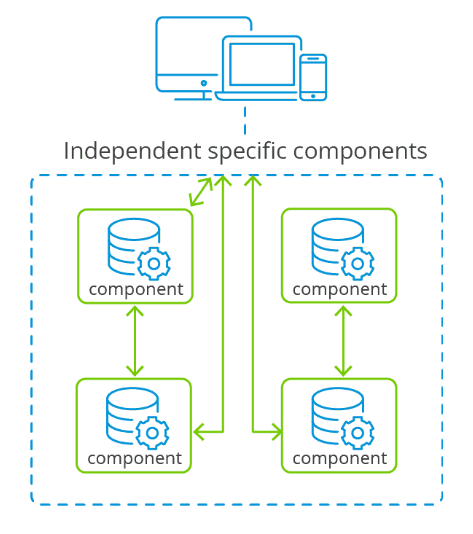
## Component-based

### 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

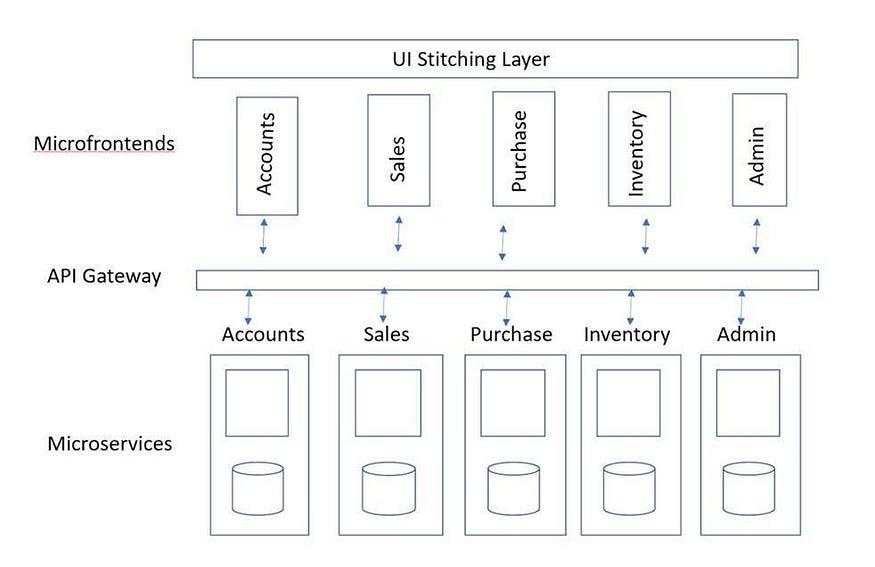
* Micro frontend là một kiến trúc ứng dụng web chia nhỏ ứng dụng thành các module hoặc chức năng riêng lẻ, được triển khai và quản lý độc lập. Giống như microservice ở phía backend, micro frontend mang lại nhiều lợi ích cho việc phát triển và bảo trì các ứng dụng web phức tạp.
* Kiến trúc micro frontend 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.

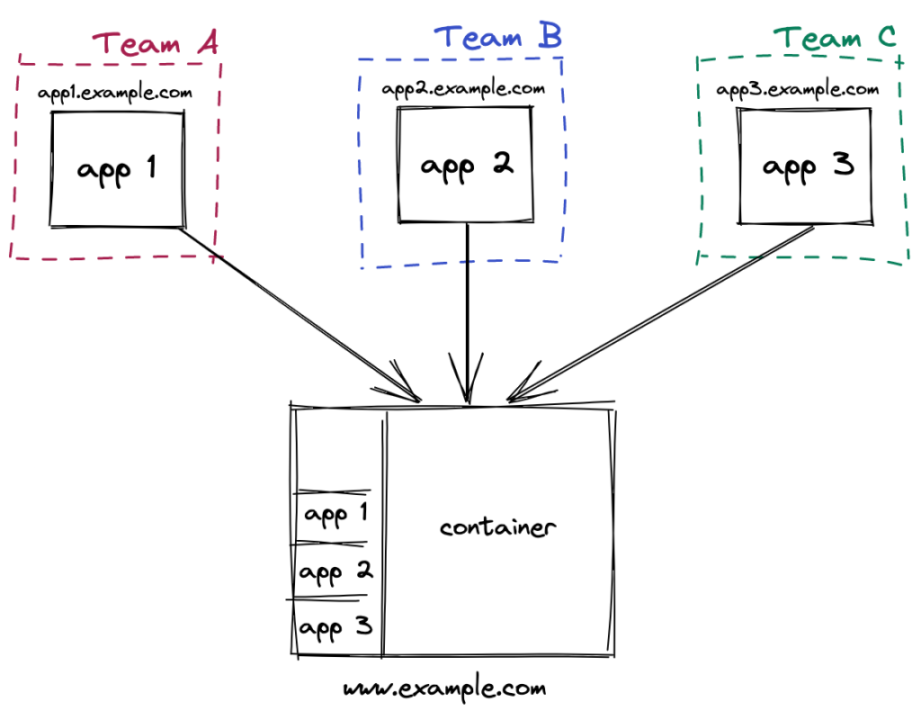


### Lợi ích của việc sử dụng mô hình microfrontend

* 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.

# Chương 3: CÁC VẤN ĐỀ CƠ BẢN CỦA FRONTEND

## 1. Hiển thị UI

## 1.1. Định nghĩa

* Hiển thị UI (viết tắt từ User Interface Display) là các yếu tố trực quan mà người dùng nhìn thấy và tương tác khi sử dụng một sản phẩm kỹ thuật số, chẳng hạn như trang web, ứng dụng di động hoặc phần mềm máy tính. Nó bao gồm tất cả các thành phần đồ họa, văn bản và các yếu tố tương tác mà người dùng sử dụng để thực hiện các tác vụ và điều hướng trong sản phẩm.

## 1.2. Phương thức xử lý

## 1.3. UI rendering trong React

* Component: Mọi thứ trong React đều là một component, bao gồm cả các phần tử HTML cơ bản. Mỗi component có một hàm render() trả về mô tả giao diện người dùng của nó.
* Virtual DOM: React sử dụng một cây DOM ảo để biểu diễn giao diện người dùng. Cây DOM ảo hiệu quả hơn nhiều so với DOM gốc vì nó chỉ cập nhật các phần của giao diện người dùng đã thay đổi thay vì render lại toàn bộ giao diện.
* Diffing: React sử dụng thuật toán diffing để so sánh cây DOM ảo trước đó với cây DOM ảo hiện tại. Thuật toán này xác định các phần của giao diện người dùng đã thay đổi và chỉ cập nhật các phần đó trong DOM gốc.
* Updating the DOM: Sau khi diffing, React cập nhật DOM gốc để phản ánh các thay đổi được xác định trong bước trước.
* React sử dụng một số kỹ thuật để tối ưu hóa quá trình rendering:
  + Batching: React sẽ gom các cập nhật trạng thái liên tiếp thành một lần cập nhật duy nhất để giảm thiểu số lần render.
  + Conditionally rendering: React cho phép bạn render các phần tử giao diện người dùng có điều kiện, nghĩa là chỉ render các phần tử cần thiết cho trạng thái hiện tại của ứng dụng.
  + PureComponent: React cung cấp lớp component PureComponent giúp tối ưu hóa hiệu suất rendering bằng cách chỉ render lại component khi các prop hoặc state của nó thay đổi.
* Một số khái niệm quan trọng khác liên quan đến UI rendering trong React:
  + State: State là dữ liệu cục bộ được lưu trữ trong mỗi component. Khi state của component thay đổi, component sẽ được render lại.
  + Props: Props là dữ liệu được truyền từ component cha sang component con. Props không ảnh hưởng trực tiếp đến việc rendering của component, nhưng chúng có thể được sử dụng trong hàm render() của component để tạo ra giao diện người dùng động.
  + Lifecycle methods: React cung cấp một số lifecycle methods cho phép bạn thực hiện các hành động nhất định tại các thời điểm khác nhau trong vòng đời của component, chẳng hạn như khi component được mounted hoặc unmounted.

## 1.4 Tổng kết

## 2. Tương tác UI

### 2.1. Giải quyết vấn đề

### 2.2. Tương tác trong React

* Dưới đây là một số khái niệm quan trọng về tương tác UI trong React:
  + Sự kiện: React sử dụng hệ thống sự kiện để theo dõi và phản hồi các hành động của người dùng. Các sự kiện phổ biến bao gồm nhấp chuột, cuộn, thay đổi kích thước và nhập văn bản.
  + Thành phần: Các thành phần React là các khối xây dựng cơ bản của giao diện người dùng. Mỗi thành phần có thể quản lý trạng thái và xử lý các sự kiện của riêng nó.
  + Hooks: React Hooks là các hàm đặc biệt cho phép bạn "móc" vào trạng thái và vòng đời của component mà không cần viết lớp. Hooks cung cấp một cách linh hoạt và gọn nhẹ để xử lý các tương tác UI.
  + Thư viện: Có nhiều thư viện React của bên thứ ba hỗ trợ việc xây dựng các tương tác UI phức tạp, chẳng hạn như react-router, react-motion và react-spring.
* Để tương tác UI hiệu quả trong React, bạn nên tuân thủ các nguyên tắc sau:
  + Rõ ràng và nhất quán: Thiết kế các tương tác UI rõ ràng, dễ hiểu và nhất quán trong suốt ứng dụng của bạn.
  + Phản hồi: Cung cấp phản hồi trực quan cho các hành động của người dùng để đảm bảo họ biết rằng hành động của họ đã được thực hiện.
  + Hiệu quả: Tối ưu hóa hiệu suất của các tương tác UI để mang lại trải nghiệm mượt mà và nhanh chóng.
  + Khả năng sử dụng: Đảm bảo rằng các tương tác UI của bạn có thể sử dụng được bởi tất cả mọi người, bất kể khả năng của họ.
  + Kiểm thử: Kiểm tra kỹ lưỡng các tương tác UI của bạn với người dùng thực tế để xác định và sửa lỗi.
* Một số ví dụ:
  + Điều hướng: React cung cấp các thư viện như react-router để tạo ra các hệ thống điều hướng mượt mà và dễ sử dụng.
  + Biểu mẫu: React cho phép bạn tạo các biểu mẫu trực quan và dễ tương tác bằng cách sử dụng các thành phần HTML và xử lý sự kiện.
  + Phản hồi động: Bạn có thể sử dụng Hooks và thư viện như react-motion và react-spring để tạo ra các hiệu ứng động và phản hồi mượt mà cho các hành động của người dùng.
  + Khả năng tiếp cận: React hỗ trợ các tính năng trợ năng để đảm bảo rằng ứng dụng của bạn có thể sử dụng được bởi tất cả mọi người, bất kể khả năng của họ.

### 2.3. Tương tác trong Flutter

* Dưới đây là một số khái niệm quan trọng về tương tác UI trong Flutter:
  + Gesture: Flutter sử dụng hệ thống gesture để theo dõi và phản hồi các hành động của người dùng trên màn hình cảm ứng. Các gesture phổ biến bao gồm chạm, vuốt, kéo, thả và phóng to thu nhỏ.
  + Widget: Widget là các khối xây dựng cơ bản của giao diện người dùng Flutter. Mỗi widget có thể quản lý trạng thái và xử lý các gesture của riêng nó.
  + State management: Flutter cung cấp các cơ chế quản lý trạng thái khác nhau để giúp bạn theo dõi và cập nhật dữ liệu UI một cách hiệu quả.
  + Animation: Flutter hỗ trợ animation mạnh mẽ để tạo ra các chuyển động mượt mà và giao diện người dùng hấp dẫn.
  + Accessibility: Flutter hỗ trợ các tính năng trợ năng để đảm bảo rằng ứng dụng của bạn có thể sử dụng được bởi tất cả mọi người, bất kể khả năng của họ.
* Để xây dựng các tương tác UI hiệu quả trong Flutter, bạn nên tuân thủ các nguyên tắc sau:
  + Rõ ràng và nhất quán: Thiết kế các tương tác UI rõ ràng, dễ hiểu và nhất quán trong suốt ứng dụng của bạn.
  + Phản hồi: Cung cấp phản hồi trực quan cho các gesture của người dùng để đảm bảo họ biết rằng hành động của họ đã được thực hiện.
  + Hiệu quả: Tối ưu hóa hiệu suất của các tương tác UI để mang lại trải nghiệm mượt mà và nhanh chóng.
  + Khả năng sử dụng: Đảm bảo rằng các tương tác UI của bạn có thể sử dụng được bởi tất cả mọi người, bất kể khả năng của họ.
  + Kiểm thử: Kiểm tra kỹ lưỡng các tương tác UI của bạn với người dùng thực tế để xác định và sửa lỗi.
* Một số ví dụ về tương tác UI phổ biến trong Flutter:
  + Điều hướng: Flutter cung cấp các widget như Navigator và MaterialPageRoute để tạo ra các hệ thống điều hướng mượt mà và dễ sử dụng.
  + Biểu mẫu: Flutter cho phép bạn tạo các biểu mẫu trực quan và dễ tương tác bằng cách sử dụng các widget như TextFormField và Form.
  + Danh sách: Flutter cung cấp widget ListView để hiển thị danh sách các item một cách hiệu quả và có thể cuộn được.
  + Chuyển động: Bạn có thể sử dụng các widget animation như AnimatedContainer và AnimatedOpacity để tạo ra các hiệu ứng chuyển động mượt mà và thu hút.

## 3. Form validation

### 3.1. Vấn đề

* Thông báo lỗi không rõ ràng:
  + Lỗi thiếu thông tin: Thông báo lỗi không chỉ ra cụ thể vấn đề gì sai với dữ liệu người dùng nhập vào, khiến họ khó sửa lỗi.
  + Ngôn ngữ khó hiểu: Thông báo lỗi sử dụng ngôn ngữ kỹ thuật hoặc không phù hợp với đối tượng người dùng, gây khó khăn trong việc hiểu và giải quyết lỗi.
  + Vị trí hiển thị không phù hợp: Thông báo lỗi xuất hiện ở vị trí không dễ nhìn thấy, khiến người dùng bỏ lỡ.
* Quy tắc validation quá phức tạp:
  + Yêu cầu quá nhiều thông tin không cần thiết từ người dùng, gây phiền hà và tốn thời gian.
  + Sử dụng các quy tắc validation không phù hợp với ngữ cảnh sử dụng, dẫn đến việc loại bỏ dữ liệu hợp lệ.
  + Quy tắc validation quá phức tạp và khó hiểu, gây khó khăn cho việc triển khai và bảo trì.
* Thiếu xử lý các trường hợp ngoại lệ:
  + Không xử lý các trường hợp dữ liệu đầu vào bất thường, dẫn đến lỗi và treo ứng dụng.
  + Không cung cấp hướng dẫn hoặc giải pháp cho người dùng khi gặp các trường hợp ngoại lệ.
* Khả năng sử dụng kém:
  + Giao diện form validation không thân thiện với người dùng, gây khó khăn trong việc sử dụng.
  + Thiếu các tính năng hỗ trợ như gợi ý, tự động hoàn thành, định dạng dữ liệu, v.v.
  + Phản hồi của form validation chậm hoặc không chính xác.

### 3.2. Cách giải quyết

* Thông báo lỗi không rõ ràng:
  + Giải pháp:
    - Cung cấp thông báo lỗi chi tiết, rõ ràng, giải thích cụ thể vấn đề và cách khắc phục.
    - Sử dụng ngôn ngữ đơn giản, dễ hiểu phù hợp với đối tượng người dùng.
    - Hiển thị thông báo lỗi gần với trường dữ liệu có lỗi để người dùng dễ dàng nhận biết.
* Quy tắc validation quá phức tạp:
  + Giải pháp:
    - Xác định rõ ràng mục đích sử dụng của form và chỉ yêu cầu những thông tin thực sự cần thiết.
    - Áp dụng các quy tắc validation phù hợp với loại dữ liệu và ngữ cảnh sử dụng.
    - Giữ cho quy tắc validation đơn giản, dễ hiểu và dễ dàng bảo trì.
* Thiếu xử lý các trường hợp ngoại lệ:
  + Giải pháp:
    - Lường trước và xử lý các trường hợp dữ liệu đầu vào bất thường.
    - Cung cấp thông báo lỗi và hướng dẫn cụ thể cho người dùng khi gặp các trường hợp ngoại lệ.
    - Cho phép người dùng bỏ qua hoặc nhập dữ liệu thay thế trong trường hợp ngoại lệ.
* Khả năng sử dụng kém:
  + Giải pháp:
    - Thiết kế giao diện form validation đơn giản, trực quan, dễ sử dụng.
    - Thêm các tính năng hỗ trợ như gợi ý, tự động hoàn thành, định dạng dữ liệu để giúp người dùng nhập dữ liệu nhanh chóng và chính xác.
    - Đảm bảo phản hồi của form validation nhanh chóng và chính xác.

### 3.3. Form Validation với React

* Một số phương pháp phổ biến để thực hiện form validation trong React:
* Sử dụng thư viện:
  + Nhiều thư viện React hỗ trợ form validation mạnh mẽ và dễ sử dụng, giúp bạn tiết kiệm thời gian và công sức phát triển. Một số thư viện phổ biến bao gồm:
  + Formik: Thư viện form validation phổ biến nhất cho React, cung cấp nhiều tính năng như xử lý lỗi, quản lý trạng thái, và định dạng dữ liệu.
  + React Hook Form: Thư viện nhẹ và linh hoạt, sử dụng Hooks để quản lý trạng thái và xử lý lỗi form validation.
  + Yup: Thư viện validation schema mạnh mẽ, cho phép bạn định nghĩa các quy tắc validation phức tạp một cách dễ dàng.
* Tự viết code validation:
  + Nếu bạn muốn kiểm soát hoàn toàn quy trình validation, bạn có thể tự viết code validation cho form của mình. Tuy nhiên, cách này đòi hỏi nhiều kiến thức về JavaScript và React, đồng thời tốn nhiều thời gian hơn so với sử dụng thư viện.
* Sử dụng các thành phần React tích hợp:
  + Một số thành phần React tích hợp sẵn chức năng validation cơ bản, như:
    - required prop: Yêu cầu người dùng phải nhập dữ liệu vào trường.
    - pattern prop: Xác định định dạng dữ liệu hợp lệ cho trường (ví dụ: email, số điện thoại).
    - min và max props: Xác định giá trị tối thiểu và tối đa cho trường số.

### 3.4. Tổng kết

## 4. Tạo frontend động (dynamic form)

### 4.1. Mô tả

* Dynamic Form hay còn gọi là Form Tự động, là một loại form có thể thay đổi cấu trúc và nội dung dựa trên các tương tác của người dùng hoặc dữ liệu được cung cấp. Nó cung cấp cho người dùng trải nghiệm linh hoạt và cá nhân hóa hơn so với các form truyền thống.
* Đặc điểm chính của Dynamic Form:
* Khả năng thay đổi: Cấu trúc và nội dung của form có thể thay đổi dựa trên các điều kiện nhất định, ví dụ như lựa chọn của người dùng hoặc dữ liệu nhập vào.
* Tính linh hoạt: Dynamic form có thể được sử dụng cho nhiều mục đích khác nhau, từ việc thu thập thông tin đơn giản đến việc tạo các quy trình phức tạp.
* Cá nhân hóa: Dynamic form có thể được cá nhân hóa cho từng người dùng dựa trên nhu cầu và thông tin của họ.
* Dễ sử dụng: Dynamic form thường có giao diện trực quan và dễ sử dụng, giúp người dùng dễ dàng cung cấp thông tin cần thiết.

### 4.2. Công nghệ BEEKAI

* BEEKAI là một trình tạo biểu mẫu trực tuyến miễn phí thế hệ mới giúp các doanh nghiệp và cá nhân tạo các ứng dụng biểu mẫu đẹp và chức năng mà không cần viết bất kỳ mã nào. Nó sử dụng giao diện biểu đồ quy trình công việc trực quan kéo thả để thiết kế biểu mẫu, giúp người dùng ở mọi cấp độ kỹ năng dễ dàng tạo các biểu mẫu phức tạp một cách dễ dàng.
* Tính năng chính của BEEKAI:
  + Trình tạo quy trình công việc trực quan: Tạo biểu mẫu bằng cách sử dụng biểu đồ quy trình công việc trực quan kéo thả, loại bỏ nhu cầu mã hóa.
  + Biểu mẫu có thể tùy chỉnh: Thiết kế biểu mẫu với nhiều yếu tố khác nhau, bao gồm trường văn bản, hộp kiểm, nút radio, menu thả xuống, v.v.
  + Logic điều kiện: Triển khai logic điều kiện để hiển thị hoặc ẩn các trường biểu mẫu dựa trên đầu vào của người dùng, tạo ra các biểu mẫu linh hoạt và được cá nhân hóa.
  + Xác thực dữ liệu: Đảm bảo độ chính xác của dữ liệu bằng cách áp dụng các quy tắc xác thực cho các trường biểu mẫu cụ thể.
  + Tích hợp: Tích hợp biểu mẫu BEEKAI với nhiều công cụ và nền tảng bên thứ ba khác nhau.
  + Thiết kế đáp ứng: Tạo biểu mẫu thích ứng liền mạch với các kích thước màn hình và thiết bị khác nhau.
  + Hợp tác: Hợp tác với các thành viên trong nhóm để thiết kế, xem xét và phê duyệt biểu mẫu.
* Lợi ích khi sử dụng BEEKAI:
  + Tiết kiệm thời gian và công sức: Loại bỏ nhu cầu mã hóa và đơn giản hóa quy trình tạo biểu mẫu.
  + Tạo biểu mẫu phức tạp dễ dàng: Xây dựng các biểu mẫu phức tạp với logic điều kiện và nhiều yếu tố biểu mẫu khác nhau mà không cần viết mã.
  + Nâng cao trải nghiệm người dùng: Thiết kế biểu mẫu thân thiện với người dùng và hấp dẫn phù hợp với các thiết bị khác nhau.
  + Thu thập dữ liệu chính xác: Đảm bảo tính toàn vẹn dữ liệu với các quy tắc xác thực dữ liệu và logic điều kiện.
  + Cải thiện hiệu quả quy trình làm việc: Tự động hóa việc thu thập dữ liệu và tích hợp biểu mẫu với các quy trình làm việc hiện có.
  + Giải pháp tiết kiệm chi phí: Sử dụng trình tạo biểu mẫu trực tuyến miễn phí mà không phát sinh thêm chi phí phần mềm.
* Trường hợp sử dụng cho BEEKAI:
  + Khảo sát khách hàng: Thu thập phản hồi và thông tin chi tiết từ khách hàng bằng cách sử dụng các cuộc khảo sát tương tác.
  + Đăng ký sự kiện: Quản lý đăng ký sự kiện và thu thập thông tin của người tham gia.
  + Tạo khách hàng tiềm năng: Thu thập khách hàng tiềm năng và chi tiết liên hệ cho khách hàng tiềm năng.
  + Ứng tuyển việc làm: Hợp lý hóa quy trình ứng tuyển việc làm và thu thập thông tin ứng viên.
  + Biểu mẫu nội bộ: Tạo biểu mẫu nội bộ cho việc tuyển dụng nhân viên, báo cáo chi phí và các quy trình kinh doanh khác.

### 4.3. Công nghệ React-hook

* React Hooks: Tăng cường sức mạnh cho các Component trong React
* React Hooks là một tính năng mới được giới thiệu trong React 16.8, mang đến cách tiếp cận mới để quản lý trạng thái và các hiệu ứng phụ (side effects) trong các component function.
* Lợi ích của React Hooks:
  + Sử dụng trạng thái và hiệu ứng phụ mà không cần lớp (class): React Hooks cho phép bạn sử dụng các tính năng cốt lõi của React như trạng thái và hiệu ứng phụ trong các component function thay vì phải viết các component class rườm rà.
  + Tái sử dụng logic: React Hooks khuyến khích việc viết logic tái sử dụng bằng cách cho phép bạn tách biệt logic trạng thái và hiệu ứng phụ khỏi component.
  + Dễ học và sử dụng: React Hooks sử dụng cú pháp JavaScript đơn giản và dễ hiểu, giúp bạn dễ dàng học hỏi và sử dụng.
  + Cải thiện hiệu suất: React Hooks có thể giúp cải thiện hiệu suất ứng dụng của bạn bằng cách tối ưu hóa cách quản lý trạng thái và hiệu ứng phụ.
* Các loại React Hooks phổ biến:
  + useState: Quản lý trạng thái cục bộ của một component.
  + useEffect: Thực hiện các hiệu ứng phụ, chẳng hạn như gọi API, đặt thời gian, hoặc đăng ký/hủy đăng ký sự kiện.
  + useContext: Chia sẻ trạng thái và logic giữa các component mà không cần truyền props qua từng cấp.
  + useReducer: Quản lý trạng thái phức tạp hơn với logic cập nhật trạng thái theo kiểu reducer.
  + useMemo: Ghi nhớ giá trị được tính toán để tránh tính toán lại không cần thiết.
  + useCallback: Ghi nhớ callback function để tránh tạo callback function mới mỗi lần render.

# Chương 4: KHÁI QUÁT VỀ REACT

## Khái niệm:

## Lịch sử phát triển:

## Kiến trúc của hệ thống React Native:

## Ưu nhược điểm:

### Ưu điểm:

### 1.4.2 Nhược điểm:

## 1.5 Hiện trạng, tiềm năng

## 1.6 Roadmap

### 1.6.1 Điều kiện tiên quyết:

### 1.6.2. Tài nguyên học tập:

# Chương 5: MICROFRONTEND VỚI REACT

## 1 Sử dụng Re.Pack:

## 1.1 Khái niệm Re.Pack:

* Re.Pack là một công cụ dựa trên Webpack giúp bạn xây dựng ứng dụng React Native với đầy đủ hỗ trợ hệ sinh thái Webpack. Nó được xem là thế hệ tiếp theo của một trình đóng gói dựa trên Webpack cho các ứng dụng React Native.
  1. **Một số khái niệm cần nắm khi sử dụng Re.Pack:** 
     1. **Một số khái niệm**

*Bảng 3: Glossary*

|  |  |
| --- | --- |
| Bundle | Một tập hợp code đã được xử lý (biên dịch, chuyển đổi), được đóng gói gọn lại thành một định dạng độc lập bởi một trình Bundler (ví dụ: Webpack, Rollup, v.v.). Nó bao gồm không chỉ code nguồn đã xử lý mà còn cả tất cả các phụ thuộc cần thiết và thường là các tài nguyên tĩnh. |
| Main bundle | Một dạng đặc biệt của Bộ thư viện (Bundle), có thể tự thực thi (độc lập). Nghĩa là việc chạy Bộ thư viện chính bên trong Máy ảo JavaScript sẽ khởi chạy ứng dụng của bạn. |
| Chunk | Một phiên bản nhẹ hơn của Bộ thư viện (Bundle), được thiết kế để nạp và sử dụng cùng với Bộ thư viện chính. Một khối (chunk) thường bị trì hoãn và được nạp theo nhu cầu bởi Bộ thư viện chính hoặc một khối khác. Các khối thường có thể chia sẻ và tái sử dụng ngầm các phụ thuộc với nhau và với Bộ thư viện chính. |
| Async chunk | Một dạng đặc biệt của Khối (Chunk), được nạp theo phương thức không đồng bộ. Trong ngữ cảnh của React Native, tất cả các Khối đều được nạp theo phương thức không đồng bộ, do đó Khối (Chunk) và Khối không đồng bộ (Async Chunk) có thể được sử dụng thay thế cho nhau. |
| Local chunk | Một Khối được lưu trữ cục bộ trên hệ thống tập tin (của thiết bị di động), đối lập với Khối từ xa (Remote Chunk). |
| Remote chunk | Một Khối được lưu trữ từ xa trên server, CDN hoặc bất kỳ vị trí mạng nào khác, đối lập với Khối cục bộ (Local Chunk). |
| Script | File tùy ý chứa mã có thể thực thi. Nó có thể là một Bộ thư viện (Bundle) được tạo bởi một Trình Bundler (ví dụ: Webpack, Rollup, v.v.) hoặc được tạo thủ công bằng tay. |
| Container | Một dạng đặc biệt của Bộ thư viện (Bundle) được tạo trong thiết lập Module Federation và được sử dụng bởi Bộ thư viện chính (Main Bundle). Các Container (Thùng chứa) cũng có thể sử dụng các Container khác và Khối (Chunk). |

* + 1. **Code splitting**
* Code Splitting là một kỹ thuật chia nhỏ code thành nhiều file khác nhau, có thể được nạp theo yêu cầu và song song.
* Công dụng:
  + Tối ưu kích thước ban đầu của ứng dụng và cải thiện hiệu suất khởi động: Bằng cách trì hoãn việc phân tích cú pháp (chỉ với JSC) và thực thi (JSC và Hermes) của code không quan trọng cho đến khi cần thiết.
  + Cung cấp nội dung và tính năng cho người dùng một cách linh hoạt dựa trên các yếu tố thời gian chạy: vai trò của người dùng, gói đăng ký, sở thích, v.v.
  + Đối với nhà phát triển và công ty: chia nhỏ và phân lập các phần của sản phẩm để cải thiện khả năng mở rộng và giảm thiểu sự phụ thuộc lẫn nhau giữa các thành phần.
* Code splitting là một trong những tính năng quan trọng nhất của Re.Pack. Nó dựa trên nền tảng của Webpack cũng như mô-đun gốc cho phép thực thi code bổ sung trên cùng một ngữ cảnh JavaScript (cùng một phiên bản React Native).
  + - 1. **Async chunks**
* Giả sử chúng ta đang xây dựng một ứng dụng học tập trực tuyến (E-Learning) với các chức năng dành riêng cho học sinh và giáo viên. Cả học sinh và giáo viên sẽ có giao diện người dùng (UI) và các tính năng khác nhau, vì vậy việc phân tách mã dành riêng cho học sinh khỏi mã dành riêng cho giáo viên là điều hợp lý. Đây chính là lúc Code Splitting (Phân tách Code) phát huy tác dụng. Chúng ta có thể sử dụng hàm dynamic import(...) cùng với React.lazy và React.Suspense để render có điều kiện phần học sinh và giáo viên dựa trên vai trò của người dùng. Mã dành cho học sinh và giáo viên sẽ được đặt trong một khối (chunk) không đồng bộ từ xa, do đó kích thước tải xuống ban đầu sẽ nhỏ hơn.
* Phía ‘học sinh’ sẽ có code như sau:

A computer code on a white background

Description automatically generated

* Và code phía giáo viên:

A computer screen shot of a computer code

Description automatically generated

* Và bây giờ khi ở trong component cha, nơi sẽ được triển khai code của cả hai phía:

A computer screen shot of a program code

Description automatically generated

* Ở giai đoạn này, toàn bộ mã được sử dụng bởi StudentSide.js sẽ được đóng gói vào student.chunk.bundle và mã của TeacherSide.js sẽ được đóng gói vào teacher.chunk.bundle. Tuy nhiên, trước khi chúng ta thực sự có thể render ứng dụng, cần thêm một bước nữa để ứng dụng có thể tìm và nạp các khối (chunk) này. Bước đó là sử dụng ScriptManager.shared.addResolver(...) để thiết lập bộ giải quyết (resolver) cho phép ứng dụng biết vị trí của các khối.

A computer screen shot of a computer code

Description automatically generated

* + - 1. **React Navigation**
* Khi sử dụng các khối không đồng bộ (Async Chunks), bạn có thể dễ dàng tích hợp các thành phần React thuộc các khối này bằng cách tạo một thành phần bao bọc (wrapper component) với React.Suspense và truyền nó như một thuộc tính component (component prop) cho một Screen (màn hình), ví dụ:

A computer screen shot of a program code

Description automatically generated

A computer screen shot of a program

Description automatically generated

* React Context (Ngữ cảnh React) cũng được truyền đến các khối (chunk), vì vậy bạn có thể sử dụng các hook useNavigation, useRoute và các hook khác bên trong các khối (ví dụ: bên trong các thành phần StudentSide hoặc TeacherSide) để truy cập dữ liệu hoặc tương tác với React Navigation.
  + 1. **Module Federation**
* Module Federation có nhiều điểm tương đồng với Code Splitting, nhưng nó cung cấp nhiều quyền kiểm soát hơn, linh hoạt hơn và khả năng mở rộng tốt hơn.
  + - 1. **Khái niệm:**
* Module Federation là một kiểu thiết kế phân rã ứng dụng thành các thành phần độc lập nhỏ hơn. Những thành phần này được gọi là container (vật chứa). Tương tự như microservices (vi mô dịch vụ) ở phía backend, Module Federation cho phép xây dựng các ứng dụng web với giao diện người dùng (front-end) phân tán. Các thành phần front-end phân tán này đôi khi được gọi là micro-frontends (vi mô front-end).

A diagram of a diagram

Description automatically generated

* + - 1. **Lợi ích của việc sử dụng Module Federation**
* Phân tách ứng dụng thành các micro-frontend độc lập: Bạn có thể chia nhỏ ứng dụng thành các khối chức năng riêng biệt, được gọi là micro-frontend. Điều này giúp cải thiện tính modular và khả năng bảo trì của ứng dụng.
* Tùy chỉnh cấu hình và quy trình xây dựng cho từng micro-frontend: Mỗi micro-frontend có thể được xây dựng và cấu hình riêng biệt, giúp tối ưu hóa hiệu suất và tính năng cho từng khối.
* Tải micro-frontend động theo yêu cầu: Chỉ những micro-frontend cần thiết mới được tải khi người dùng tương tác với chức năng tương ứng, giúp cải thiện thời gian tải ban đầu của ứng dụng.
* Sử dụng các phiên bản khác nhau của micro-frontend: Bạn có thể cập nhật từng micro-frontend độc lập mà không ảnh hưởng đến các phần khác của ứng dụng, tăng tính linh hoạt và khả năng cập nhật.
* Tích hợp micro-frontend từ bên ngoài: Bạn có thể tận dụng các micro-frontend đã được xây dựng sẵn từ các nguồn khác, giúp tiết kiệm thời gian phát triển và tận dụng các tính năng có sẵn.

#### **1.2.2.3 Những khó khăn khi sử dụng:**

* Trùng lặp phụ thuộc (dependency duplication): Nếu bạn sử dụng các phiên bản không tương thích của cùng một thư viện trong các micro-frontend/container khác nhau, có thể dẫn đến tình trạng trùng lặp phụ thuộc. Điều này gây lãng phí dung lượng và tiềm ẩn các lỗi xung đột.
* Phối hợp cấu hình Webpack: Cấu hình Webpack cho từng micro-frontend/container đòi hỏi sự phối hợp chặt chẽ. Nếu cấu hình không tương thích, các thành phần có thể không hoạt động chính xác.
* Triển khai phức tạp: Mỗi micro-frontend/container cần được triển khai riêng và sẵn sàng cho phía client (thường là thông qua internet). Điều này phức tạp hơn so với việc triển khai một ứng dụng monolithic.
* Quản lý phát hành phức tạp: Cần đảm bảo các micro-frontend/container độc lập càng nhiều càng tốt, tránh phụ thuộc lẫn nhau. Nếu có phụ thuộc, việc phát hành cần được đồng bộ để tránh lỗi.

# Chương 6: KẾT LUẬN

## 1.1 Ưu điểm của đồ án:

* Cung cấp cho bạn những kiến thức cơ bản về mô hình Micro Frontend, lợi ích, cách thức hoạt động và các công nghệ liên quan. Đồng thời, bạn đã được hướng dẫn cách xây dựng một ứng dụng demo sử dụng React Native để áp dụng mô hình Micro Frontend sử dụng Re.Pack.

## 1.2 Hạn chế của đồ án:

* Dự án chỉ tập trung vào giới thiệu các khái niệm cơ bản về Micro Frontend và ứng dụng demo cơ bản.
* Chưa đi sâu vào các chi tiết kỹ thuật phức tạp của việc xây dựng và triển khai các ứng dụng Micro Frontend quy mô lớn.
* Ít ví dụ thực tế về việc áp dụng Micro Frontend trong các dự án cụ thể**.**

**1.3 Hướng phát triển của đồ án:**

**1.3.1 Mở rộng phạm vi:**

* Cung cấp thêm ví dụ thực tế về việc áp dụng Micro Frontend với React Native trong các dự án với quy mô và độ phức tạp khác nhau.
* Giới thiệu các công nghệ và framework tiên tiến khác có thể được sử dụng cho Micro Frontend, chẳng hạn như Webpack, Module Federation, Single-spa, Bit, v.v.
* Thảo luận về các vấn đề thường gặp khi xây dựng và triển khai các ứng dụng Micro Frontend, và cách giải quyết các vấn đề đó.

**1.3.2. Đi sâu vào chi tiết kỹ thuật:**

* Cung cấp hướng dẫn chi tiết về cách xây dựng từng phần của ứng dụng demo, bao gồm cấu trúc ứng dụng, cách viết component, cách quản lý state, cách xử lý sự kiện, v.v.
* Giải thích các khái niệm kỹ thuật phức tạp liên quan đến Micro Frontend, chẳng hạn như microfrontend isolation, cross-origin resource sharing (CORS), event bubbling, v.v.
* Chia sẻ các mẹo và thủ thuật hữu ích để tối ưu hóa hiệu suất và bảo trì các ứng dụng Micro Frontend.

# TÀI LIỆU THAM KHẢO