KHOA KỸ THUẬT VÀ CÔNG NGHỆ

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

****

THỰC TẬP ĐỒ ÁN CHUYÊN NGÀNH

**HỌC KỲ I, NĂM HỌC 2024-2025**

**XÂY DỰNG WEBSITE BÁN CÂY CẢNH VỚI MÃ NGUỒN MỞ**

*Giáo viên hướng dẫn:*

ThS. Nguyễn Ngọc Đan Thanh

*Sinh viên thực hiện:*

Họ tên: Võ Thị Diểm

MSSV: 110121013

Lớp: DA21TTB

***Trà Vinh, tháng 12 năm 2024***

KHOA KỸ THUẬT VÀ CÔNG NGHỆ

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

****

THỰC TẬP ĐỒ ÁN CƠ SỞ NGÀNH

**HỌC KỲ I, NĂM HỌC 2024-2025**

**XÂY DỰNG WEBSITE BÁN CÂY CẢNH VỚI MÃ NGUỒN MỞ**

*Giáo viên hướng dẫn:*ThS. Nguyễn Ngọc Đan Thanh

*Sinh viên thực hiện:*

Họ tên: Võ Thị Diểm

MSSV: 110121013

Lớp: DA21TTB

***Trà Vinh, tháng 12 năm 2024***

++

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

1. **Quá trình thực hiện:**

* Nghiêm túc, báo cáo tiến độ theo lịch
* Có khả năng nghiên cứu độc lập

1. **Về báo cáo**

* Đúng mẫu
* Đầy đủ các nội dung theo đề cương

1. **Về chương trình**

* Chương trình đáp ứng theo yêu cầu đã phân tích
* Dữ liệu minh họa phong phú

1. **Kết luận:** Đạt mức tốt

*Trà Vinh, ngày ….. tháng …… năm ……*

**Giáo viên hướng dẫn**

*(Ký tên và ghi rõ họ tên)*

**NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG**

*Trà Vinh, ngày ….. tháng …… năm ……*

**Thành viên hội đồng**

*(Ký tên và ghi rõ họ tên)*

**LỜI CẢM ƠN**

Đầu tiên, tôi xin gửi lời cảm ơn chân thành đến Khoa Kỹ thuật và Công nghệ đã tạo điều kiện thuận lợi cho tôi trong suốt quá trình học tập và hoàn thành đồ án thực tập cơ sở ngành. Tiếp theo, tôi xin bày tỏ lòng biết ơn sâu sắc đến cô Nguyễn Ngọc Đan Thanh đã dày công truyền đạt kiến thức và hướng dẫn tôi trong quá trình làm bài.

Tôi đã cố gắng vận dụng những kiến thức đã học được trong các học kỳ qua để hoàn thành đồ án. Kết quả đạt được là cả quá trình nổ lực của tôi, tuy nhiên vẫn không tránh khỏi những thiếu sót trong quá trình nghiên cứu và trình bày. Rất kính mong sự góp ý của quý thầy (cô) để đồ án thực tập cơ sở ngành của tôi được hoàn thiện hơn.

Xin trân trọng cảm ơn!

**Sinh viên thực hiện**

**MỤC LỤC**

[CHƯƠNG 1. TỔNG QUAN 1](#_Toc179962037)

[CHƯƠNG 2. NGHIÊN CỨU LÝ THUYẾT 2](#_Toc179962038)

[2.1 Giới thiệu các frontend framework 2](#_Toc179962039)

[2.1.1 Bootstrap 2](#_Toc179962040)

[2.1.2 Angular 2](#_Toc179962041)

[2.1.3 W3.CSS 2](#_Toc179962042)

[2.1.4 Vue.js 2](#_Toc179962043)

[2.2 Giới thiệu về các backend framework 3](#_Toc179962044)

[2.2.1 Express.js (Node.js) 3](#_Toc179962045)

[2.2.2 Django (Python) 3](#_Toc179962046)

[2.2.3 Spring Boot(Java) 3](#_Toc179962047)

[2.3 Giới thiệu về Tailwind CSS 3](#_Toc179962048)

[2.3.1 Tổng quan Tailwind CSS 3](#_Toc179962049)

[2.3.2 Cách cài đặt Tailwind 4](#_Toc179962050)

[2.3.3 Các class tiện ích của Tailwind 7](#_Toc179962051)

[2.4 Giới thiệu về React 25](#_Toc179962052)

[2.4.1 Tổng quan về React 25](#_Toc179962053)

[2.4.2 Cách cài đặt React 26](#_Toc179962054)

[2.4.3 Cấu trúc của React 30](#_Toc179962055)

[2.4.4 Các khái niệm cơ bản trong React 32](#_Toc179962056)

[2.4.5 Hooks 40](#_Toc179962057)

[2.4.6 Axios 46](#_Toc179962058)

[2.5 Giới thiệu về Laravel 47](#_Toc179962059)

[2.5.1 Tổng quan về Laravel 47](#_Toc179962060)

[2.5.2 Ưu điểm của Laravel 47](#_Toc179962061)

[2.5.3 Cách cài đặt Laravel 48](#_Toc179962062)

[2.5.4 Cấu trúc và thư mục trong Laravel 51](#_Toc179962063)

[2.5.5 Mô hình MVC trong Laravel 53](#_Toc179962064)

[2.5.6 Các thành phần của Laravel 54](#_Toc179962065)

[2.6 Giới thiệu về Postman 68](#_Toc179962066)

[2.6.1 API là gì? Vì sao phải test API? 68](#_Toc179962067)

[2.6.2 Postman là gì? 68](#_Toc179962068)

[2.6.3 Ưu, nhược điểm của Postman 68](#_Toc179962069)

[2.6.4 Các thành phần trong Postman 69](#_Toc179962070)

[2.6.5 Cách tạo Request 71](#_Toc179962071)

[CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU 73](#_Toc179962072)

[3.1 Mô tả bài toán 73](#_Toc179962073)

[3.2 Đặc tả các yêu cầu chức năng 73](#_Toc179962074)

[3.2.1 Yêu cầu chức năng 73](#_Toc179962075)

[3.2.2 Yêu cầu phi chức năng 73](#_Toc179962076)

[3.3 Thiết kế dữ liệu 73](#_Toc179962077)

[3.3.1 Lược đồ cơ sở dữ liệu 73](#_Toc179962078)

[3.3.2 Danh sách các thực thể 73](#_Toc179962079)

[3.3.3 Chi tiết các thực thể 73](#_Toc179962080)

[3.4 Thiết kế giao diện 73](#_Toc179962081)

[3.4.1 Sơ đồ website 73](#_Toc179962082)

[3.4.2 Giao diện website 73](#_Toc179962083)

[CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU 73](#_Toc179962084)

[4.1 Dữ liệu thử nghiệm 73](#_Toc179962085)

[4.2 Kết quả thử nghiệm 73](#_Toc179962086)

[CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 73](#_Toc179962087)

[5.1 Kết luận 73](#_Toc179962088)

[5.2 Hướng phát triển 73](#_Toc179962089)

[DANH MỤC TÀI LIỆU THAM KHẢO 74](#_Toc179962090)

[PHỤ LỤC 75](#_Toc179962091)

**DANH MỤC HÌNH ẢNH - BẢNG BIỂU**

[Hình 2.1 Kết quả chia cột theo colums-3 7](#_Toc179962092)

[Hình 2.2 Kết quả hiển thị block và inline-block 8](#_Toc179962093)

[Hình 2.3 Kết quả sử dụng positon relative và absolute 9](#_Toc179962094)

[Hình 2.4 Kết quả sử dụng flex-wrap 11](#_Toc179962095)

[Hình 2.5 Kết quả sử dụng grid 12](#_Toc179962096)

[Hình 2.6 Kết quả sử dụng padding 12](#_Toc179962097)

[Hình 2.7 Kết quả sử dụng margin 13](#_Toc179962098)

[Hình 2.8 Kết quả các kích thước chiều rộng 15](#_Toc179962099)

[Hình 2.9 Kết quả thiết lập chiều rộng theo phần trăm 17](#_Toc179962100)

[Hình 2.10 Kết quả thiết lập chiều cao cố định 18](#_Toc179962101)

[Hình 2.11 Kết quả sử dụng một số lớp tiện ích trong Typography 20](#_Toc179962102)

[Hình 2.12 Kết quả sử dụng màu nền 21](#_Toc179962103)

[Hình 2.13 Kết quả tạo đường viền 22](#_Toc179962104)

[Hình 2.14 Kết quả tạo viền đỏ 22](#_Toc179962105)

[Hình 2.15 Kết quả tạo kiểu viền 23](#_Toc179962106)

[Hình 2.16 Kết quả chạy React 27](#_Toc179962107)

[Hình 2.17 Kết quả của trước/sau khi nhấn nút, ngược lại 35](#_Toc179962108)

[Hình 2.18 Kết quả cài đặt composer thành công 48](#_Toc179962109)

[Hình 2.19 Kết quả chạy serve thành công 49](#_Toc179962110)

[Hình 2.20 Kết quả hiển thị giao diện Laravel 50](#_Toc179962111)

[Hình 2.21 Kết quả tạo middleware 56](#_Toc179962112)

[Hình 2.22 Kết quả tạo PageController 57](#_Toc179962113)

[Hình 2.23 Kết quả chạy logic trong controller thông qua route 58](#_Toc179962114)

[Hình 2.24 Kết quả hiển thị từ view 60](#_Toc179962115)

[Hình 2.25 Lấy địa chỉ API dùng phương thức GET 71](#_Toc179962116)

[Hình 2.26 Kết quả lấy dữ liệu thành công 72](#_Toc179962117)

[Hình 2.27 Tạo request POST 72](#_Toc179962118)

[Bảng 3.1 Danh sách thực thể 32](#_Toc154839936)

[Bảng 3.2 Chi tiết thực thể nguoidung 33](#_Toc154839937)

[Bảng 3.3 Chi tiết thực thể sanpham 33](#_Toc154839938)

[Bảng 3.4 Chi tiết thực thể danhmuc 33](#_Toc154839939)

[Bảng 3.5 Chi tiết thực thể hinhanh 34](#_Toc154839940)

[Bảng 4.1 Dữ liệu người dùng 37](#_Toc154839941)

[Bảng 4.2 Dữ liệu danh mục 37](#_Toc154839942)

[Bảng 4.3 Dữ liệu sản phẩm 37](#_Toc154839943)

[Bảng 4.4 Dữ liệu hình ảnh 38](#_Toc154839944)

**TÓM TẮT ĐỒ ÁN CHUYÊN NGÀNH**

Vấn đề nghiên cứu: Trong việc phát triển một website bán hàng, các yếu tố như giao diện người dùng trực quan, tốc độ tải trang nhanh, khả năng tương thích trên nhiều thiết bị và quản lý dữ liệu hiệu quả là những yếu tố quyết định sự thành công của trang web.

Hướng tiếp cận: Quá trình nghiên cứu sẽ phân tích các câu lệnh cơ bản và các tính năng chính của ReactJS, TailwindCSS, và Laravel trong việc xây dựng website thương mại điện tử. Cụ thể, nghiên cứu sẽ tập trung vào cách ReactJS giúp tạo ra các giao diện người dùng động, quản lý trạng thái sản phẩm và giỏ hàng, đồng thời cải thiện trải nghiệm người dùng. Song song với đó, TailwindCSS sẽ được sử dụng để tùy chỉnh và tối ưu hóa giao diện cho các thiết bị di động và máy tính. Laravel sẽ đóng vai trò xử lý backend, bao gồm quản lý cơ sở dữ liệu sản phẩm, thực hiện các giao dịch mua bán, và đảm bảo an toàn thông tin khách hàng..

Cách giải quyết vấn đề: Sẽ bao gồm việc tham khảo các tài liệu hướng dẫn chính thức của ReactJS, TailwindCSS, và Laravel, đồng thời áp dụng chúng vào thực tế phát triển website bán hàng. Qua các ví dụ thực tiễn, sẽ tiến hành tích hợp ReactJS để quản lý giỏ hàng, TailwindCSS để tùy chỉnh giao diện sản phẩm, và Laravel để xử lý đơn hàng và quản lý cơ sở dữ liệu sản phẩm. Bằng cách kết hợp ba công nghệ này, sẽ tạo ra một website có khả năng tương tác tốt, bảo mật cao và tối ưu cho người dùng.

Một số kết quả đạt được: Sau khi nghiên cứu và thực hành, một số kết quả đáng chú ý sẽ được đạt được:

* Tạo giao diện động, dễ sử dụng cho người mua sắm, tối ưu hóa cho nhiều thiết bị khác nhau.
* Quản lý giỏ hàng và trạng thái người dùng hiệu quả bằng ReactJS.
* Tùy chỉnh và tối ưu hóa giao diện sản phẩm một cách linh hoạt bằng TailwindCSS.
* Sử dụng Laravel để quản lý dữ liệu sản phẩm, xử lý đơn hàng, và bảo mật thông tin khách hàng, giúp website hoạt động mượt mà và an toàn.

**MỞ ĐẦU**

Trong thời đại công nghệ số phát triển nhanh chóng, việc mua sắm trực tuyến đã trở thành một phần không thể thiếu trong cuộc sống hàng ngày, thay đổi hoàn toàn thói quen mua sắm của con người. Thương mại điện tử không chỉ giúp người tiêu dùng dễ dàng tiếp cận sản phẩm từ khắp nơi trên thế giới, mà còn tạo cơ hội lớn cho các doanh nghiệp và cá nhân xây dựng thương hiệu của mình trên nền tảng số. Các website bán hàng trực tuyến không chỉ cần đáp ứng được nhu cầu của khách hàng về sự tiện lợi, mà còn phải đảm bảo tốc độ tải trang nhanh, giao diện trực quan, và khả năng tương thích trên nhiều thiết bị. Những yếu tố này đều có ảnh hưởng trực tiếp đến trải nghiệm người dùng và quyết định thành bại của một trang web bán hàng.

Việc xây dựng một website thương mại điện tử hiện đại đòi hỏi sự kết hợp của nhiều yếu tố về công nghệ và trải nghiệm người dùng. Không chỉ cần một giao diện đẹp, mà còn phải tối ưu về hiệu suất và bảo mật, giúp khách hàng có những trải nghiệm mua sắm mượt mà và an toàn. Những yêu cầu này đang thúc đẩy các nhà phát triển tìm đến các công nghệ hiện đại như ReactJS, Laravel và TailwindCSS, nhằm mang lại sự linh hoạt, tốc độ và trải nghiệm tối ưu cho người dùng.

# 

# TỔNG QUAN

Trong thời đại công nghệ số, việc xây dựng một website thương mại điện tử cho việc bán cây cảnh không chỉ đơn thuần là tạo ra một nền tảng mua bán trực tuyến. Để thành công, các website này cần phải đáp ứng nhiều yếu tố quan trọng nhằm cải thiện trải nghiệm của người dùng và thu hút khách hàng yêu thích cây cảnh.

Đầu tiên, tốc độ tải trang đóng vai trò quyết định, người tiêu dùng thường có xu hướng rời khỏi trang web nếu nó mất quá nhiều thời gian để tải, đặc biệt khi họ đang tìm kiếm cây cảnh đẹp mắt và đa dạng chủng loại. Thứ hai, giao diện người dùng (UI) cần được thiết kế trực quan, dễ sử dụng, và thể hiện phong cách gần gũi với thiên nhiên, giúp người dùng dễ dàng tìm kiếm và chọn lựa các loại cây phù hợp với sở thích và không gian của họ.

Ngoài ra, website cần tối ưu hóa khả năng tương thích với nhiều thiết bị, từ máy tính để bàn đến điện thoại di động, nhằm đảm bảo trải nghiệm nhất quán và mượt mà trên mọi kích thước màn hình. Điều này đặc biệt quan trọng vì người dùng có thể truy cập và xem các sản phẩm bất cứ lúc nào, bất cứ đâu. Giao diện cần được tối ưu hóa để hiển thị hình ảnh cây cảnh rõ ràng, sắc nét và hấp dẫn, giúp khách hàng hình dung tốt hơn về sản phẩm.

Một yếu tố không thể thiếu là cung cấp thông tin chi tiết về từng loại cây, bao gồm cách chăm sóc, môi trường phù hợp, và lợi ích của cây cảnh đối với sức khỏe và phong thủy. Điều này không chỉ tăng giá trị nội dung cho website mà còn giúp người dùng có thêm kiến thức, từ đó tạo niềm tin và thúc đẩy họ quyết định mua hàng. Các tính năng như lọc sản phẩm theo kích thước, màu sắc và giá cả cũng là yếu tố cần thiết, giúp khách hàng tìm được sản phẩm ưng ý một cách nhanh chóng và tiện lợi.

Dưới đây là một số website thương mại điện tử phổ biến trong lĩnh vực bán cây cảnh.

1. **Cây Xinh**
   * **Ưu điểm**: Website cung cấp hình ảnh đẹp, thông tin chi tiết và phân loại cây rõ ràng, giúp người dùng dễ dàng lựa chọn.
   * **Hạn chế**: Website Cây Xinh gặp một số hạn chế về tốc độ xử lý thêm sản phẩm vào giỏ hàng chậm không có chức năng xem nhanh sản phẩm, chỉ thêm vào giỏ hàng 1 lần khi ở trang chủ. [1]
2. **Sài Gòn Hoa**
   * **Ưu điểm**: Sài Gòn Hoa có danh mục sản phẩm đa dạng, từ cây cảnh văn phòng đến các loại hoa trang trí và website có nhiều bài viết hướng dẫn chăm sóc cây, cung cấp kiến thức hữu ích cho người mới.
   * **Hạn chế**: Thiết kế giao diện có phần cũ, thiếu sự hiện đại, khi ở trang chủ muốn thêm vào giỏ hàng phải chọn vào sản phẩm nó chuyển đến trang chi tiết sản phẩm mới cho phép thêm vào giỏ hàng. Khả năng lọc sản phẩm cũng khá hạn chế, đôi khi gây khó khăn khi người dùng muốn tìm cây theo kích thước hoặc giá. [2]
3. **Cây Cảnh Hà Nội**
   * **Ưu điểm**: Website này có nhiều chủng loại cây và giá cả hợp lý, cũng như cung cấp dịch vụ chăm sóc cây tận nhà, tạo sự tiện lợi cho người dùng.
   * **Hạn chế**: Giao diện website khá phức tạp, thông tin về sản phẩm không được tổ chức gọn gàng, gây khó khăn cho người dùng khi tìm kiếm cây theo từng danh mục. Tốc độ tải trang cũng khá chậm, ảnh hưởng đến trải nghiệm mua sắm trực tuyến. [3]
4. **Sapo Garden**
   * **Ưu điểm**: Sapo Garden có giao diện đẹp và chuyên nghiệp. Nó cung cấp thông tin chi tiết về sản phẩm với hình ảnh rõ ràng. Ngoài ra, website cũng có các bộ lọc tìm kiếm giúp người dùng tìm cây dễ dàng và nhanh chóng.
   * **Hạn chế**: Thêm vào giỏ hàng bị tải lại trang khiến cho người dùng trải nghiệm không được mượt. [4]

Nhìn chung, mặc dù các website này đều có những ưu điểm nổi bật và mang lại nhiều lợi ích cho người dùng, nhưng các hạn chế về tốc độ tải trang, giao diện chưa tối ưu, thiếu hệ thống đánh giá từ khách hàng và tính năng hỗ trợ trực tuyến làm giảm tính trải nghiệm và sự tiện lợi cho khách hàng. Để giải quyết các hạn chế này, tôi đã lựa chọn và kết hợp những công nghệ tiên tiến nhằm xây dựng một website hoàn thiện, tối ưu hơn cho người dùng.

Trước hết, **TailwindCSS** là một framework CSS tiện lợi và mạnh mẽ, cung cấp các lớp tiện ích giúp tùy chỉnh giao diện một cách linh hoạt và nhanh chóng. Với TailwindCSS, có thể tạo ra một giao diện đẹp mắt, hiện đại và tương thích với mọi kích thước màn hình mà không cần viết mã CSS phức tạp.

Bên cạnh đó, **ReactJS** giúp frontend của website trở nên linh hoạt và phản hồi nhanh chóng, mang lại trải nghiệm mượt mà cho người dùng. Việc sử dụng ReactJS cho phép các thao tác trên trang, như tìm kiếm và thêm sản phẩm vào giỏ hàng, diễn ra ngay lập tức mà không cần tải lại trang, giúp nâng cao trải nghiệm mua hàng.

Cuối cùng, **Laravel** là một framework PHP mạnh mẽ, cung cấp khả năng quản lý cơ sở dữ liệu, xác thực người dùng và xử lý API hiệu quả, giúp xây dựng backend một cách bảo mật và dễ dàng. Laravel hỗ trợ tốt cho việc xử lý các yêu cầu của người dùng và bảo mật thông tin.

Từ những lợi ích mà TailwindCSS, ReactJS và Laravel đem lại tôi đã quyết định chọn đề tài “ Xây dựng website bán cây cảnh với mã nguồn mở ” để thực hiện đồ án chuyên ngành của mình.

**Tài liệu tham khảo**

|  |  |
| --- | --- |
| [1] | "Cây xinh," [Online]. Available: https://cayxinh.com.vn/. [Accessed 9 Nov 2024]. |
| [2] | "Sài Gòn Hoa," [Online]. Available: https://saigonhoa.com/. [Accessed 9 11 2024]. |
| [3] | "Cây cảnh Hà Nội," [Online]. Available: https://caycanhhanoi.com/. [Accessed 9 Nov 2024]. |
| [4] | "Sapo Garden," [Online]. Available: https://themes.sapo.vn/demo/template-garden. [Accessed 9 Nov 2024]. |
|  |
|  |

# NGHIÊN CỨU LÝ THUYẾT

## Giới thiệu các frontend framework

### Bootstrap

Bootstrap là một framework HTML, CSS, JavaScript mã nguồn mở miễn phí, giúp cho quá trình xây dựng giao diện trở nên dễ dàng và nhanh chóng. Bootstrap cung cấp các thành phần được thiết kế sẵn có như: các nút, navbar, slider, các thông báo, cảnh báo, form, modal,.... Ngoài ra, còn hỗ trợ responsive tương thích với nhiều loại thiết bị khác nhau như điện thoại, máy tính, tablet,…giúp cho các lập tình viên tốn ít thời gian và công sức thiết kế giao diện, tạo ra các trang web thân thiện với người dùng.

### Angular

Angular là một JS framework mã nguồn mở được viết bằng TypeScript và phát triển bởi Google.Angular được dùng để thiết kế giao diện người dùng (front-end) của các ứng dụng web. Angular có một cấu trúc mạnh mẽ nó hỗ trợ xây dựng các ứng dụng web đơn trang (Single-Page Applications - SPA), giúp tạo giao diện người dùng tương tác và linh hoạt, hỗ trợ kiểm thử và bảo trì ứng dụng một cách dễ dàng và nhanh chóng.

### W3.CSS

W3.CSS là một thư viện CSS dùng để thiết kế giao diện website, nó rất là nhỏ gọn, có tốc độ nhanh và dễ sử dụng so với các framework khác và được phát triển bởi World Wide Web Consortium (W3C). Ngoài việc nhỏ gọn, dễ sử dụng W3.CSS còn có tính năng phản hồi và có sự tương thích với nhiều loại thiết bị có kích thước màn hình khác nhau, giúp tạo ra giao diện đẹp dễ sử dụng. W3.CSS có cung cấp các thành phần được xây dựng sẵn như: nút, thanh điều hướng, bảng, lưới,… giúp cho việc xây dựng và thiết website được thuận tiện, dễ dàng và nhanh chóng.

### Vue.js

Vue.js là một framework JavaScript mã nguồn mở được sử dụng để xây dựng các giao diện người dùng web tương tác cho ứng dụng. Vue.js cung cấp các thành phần tái sử dụng và cung cấp các tính năng để quản lý trạng thái ứng dụng một cách hiệu quả. Vue.js hỗ trợ two-way data binding, nghĩa là khi dữ liệu trong model thay đổi, giao diện người dùng cũng tự động thay đổi và ngược lại. Điều này rất hữu ích khi làm việc với các form hoặc dữ liệu động, nó thể tích hợp vào các ứng dụng web hiện có hoặc được sử dụng để xây dựng các ứng dụng web độc lập. Vue.js cũng được đánh giá cao về hiệu suất và khả năng tương thích với các thư viện và framework khác, dễ học, dễ tiếp cận.

## Giới thiệu về các backend framework

### Express.js (Node.js)

Express.js là một framework web phổ biến cho Node.js, thiết kế đơn giản nhưng cực kỳ linh hoạt. Nó cung cấp các công cụ tối thiểu cho việc xây dựng các API và ứng dụng web mà không áp đặt nhiều quy định hoặc mô hình lập trình cụ thể. Express.js được sử dụng chủ yếu cho các ứng dụng API RESTful, dịch vụ microservices, và ứng dụng thời gian thực như chat hoặc game.

### Django (Python)

Django là một framework web cao cấp, được phát triển với triết lý "batteries-included" (bao gồm tất cả mọi thứ cần thiết) và bảo mật cao. Django tuân theo mô hình MVT (Model-View-Template), giúp phát triển ứng dụng web nhanh chóng và hiệu quả. Django thường được sử dụng cho các hệ thống web lớn, các ứng dụng quản lý nội dung, các nền tảng thương mại điện tử, và các dịch vụ SaaS (Software-as-a-Service).

### Spring Boot(Java)

Spring Boot là một framework được xây dựng trên nền tảng Spring (một framework phổ biến trong hệ sinh thái Java), giúp đơn giản hóa quá trình phát triển các ứng dụng Java bằng cách cung cấp các tính năng mạnh mẽ và tự động hóa nhiều cấu hình phức tạp. Spring Boot cho phép bạn tạo các ứng dụng độc lập, có thể chạy ngay mà không cần cấu hình thủ công phức tạp như trong các dự án Java truyền thống.

## Giới thiệu về Tailwind CSS

### Tổng quan Tailwind CSS

Tailwind CSS là một framework CSS dạng utility-first được thiết kế để giúp bạn xây dựng giao diện người dùng một cách nhanh chóng và linh hoạt. Thay vì cung cấp các thành phần giao diện sẵn có như trong Bootstrap, Tailwind CSS cung cấp hàng loạt các class tiện ích nhỏ mà bạn có thể sử dụng để trực tiếp tùy chỉnh các yếu tố của giao diện.

Các đặc điểm chính của Tailwind CSS:

* **Utility-first (Tiện ích trước):** Thay vì sử dụng các thành phần giao diện sẵn có như button hoặc form với thiết kế định sẵn, Tailwind CSS cho phép bạn xây dựng giao diện hoàn toàn từ đầu bằng cách kết hợp các class nhỏ như text-center, bg-blue-500, p-4, mt-2,... Mỗi class điều chỉnh một thuộc tính CSS cụ thể (ví dụ: màu sắc, khoảng cách, kích thước).
* **Không có các thành phần UI sẵn có:** Tailwind không cung cấp các component giao diện như nút bấm hoặc navbar có sẵn. Thay vào đó, bạn có thể tạo ra chúng từ các class tiện ích.
* **Tùy biến mạnh mẽ:**Tailwind CSS có thể dễ dàng tùy biến thông qua file cấu hình tailwind.config.js. Bạn có thể thay đổi màu sắc, font chữ, kích thước và khoảng cách theo ý thích của mình.
* **Responsive design dễ dàng:** Tailwind CSS cung cấp các class để tạo giao diện responsive dễ dàng, ví dụ: sm:p-4, md:p-6, lg:p-8. Bạn có thể điều chỉnh giao diện dựa trên kích thước màn hình mà không cần viết media queries thủ công.
* **JIT (Just-In-Time) Mode:** JIT giúp Tailwind CSS chỉ tạo ra các class mà bạn thực sự sử dụng trong dự án, giúp giảm dung lượng file CSS một cách đáng kể.

### Cách cài đặt Tailwind

Để sử dụng Tailwind CSS có 4 cách để cài đặt tùy thuộc vào người sử dụng muốn chọn cách nào phù hợp với môi trường phát triển của dự án.

#### Cài đặt thông qua CDN

Phương pháp đơn giản nhất để bắt đầu sử dụng Tailwind CSS chỉ cần thêm link liên kết CDN vào trong phần header của file HTML cho phép sử dụng Tailwind ngay lập tức mà không cần cấu hình gì thêm.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Ví dụ về Tailwind</title>

<link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">

</head>

<body>

<div class="bg-green-500 text-white">

Chào mừng đến với Tailwind CSS

</div>

</body>

</html>

#### Cài đặt thông qua npm

Nếu người dùng đang làm việc trên một dự án lớn hơn hoặc muốn tùy chỉnh Tailwind CSS thì cài đặt thông qua npm là sự lựa chọn tốt nhất.

Các bước thực hiện cài đặt:

Bước 1: Tạo dự án

mkdir my-project

cd my-project

npm init -y

Bước 2: Cài đặt Tailwind CSS

npm install tailwindcss

Bước 3: Tạo file cấu hình

npx tailwindcss init

Bước 4: Tạo file CSS và import Tailwind. Để thực hiện bước này cần tạo file CSS và thêm vào các dòng dưới đây.

@tailwind base;

@tailwind components;

@tailwind utilities;

Bước 5: Biên dịch CSS

npx tailwindcss -i ./styles.css -o ./dist/output.css --watch

Bước 6: Thêm CSS vào HTML

<link href="dist/output.css" rel="stylesheet">

#### Cài đặt qua PostCSS

Nếu người dùng đang sử dụng PostCSS trong dự án có thể thêm Tailwind như một plugin.

Bước 1: Cài đặt các package cần thiết

npm install tailwindcss postcss autoprefixer

Bước 2: Tạo file cấu hình PostCSS, để thực hiện bước này cần tạo file postcss.config.js và thêm vào cấu hình dưới đây.

module.exports = {

plugins: {

tailwindcss: {},

autoprefixer: {},

},

}

#### Tích hợp vào các framework JavaScript

Tailwind CSS có thể dễ dàng tích hợp vào các framework JavaScript như React, Vue hoặc Angular. Dưới đây là ví dụ cho React.

Bước 1: Cài đặt Tailwind CSS

npm install tailwindcss

Bước 2: Tạo file cấu hình và cấu hình như trên

Bước 3: Import Tailwind CSS vào file chính .Ví dụ: index.js hoặc App.js

import './styles.css';

### Các class tiện ích của Tailwind

#### Container

Class container được sử dụng để tạo một phần tử với chiều rộng tối đa được xác định sẵn cho phép điều chỉnh giao diện của trang web theo kích thước màn hình khác nhau. Khi kích thước màn hình nhỏ hơn breakpoint đã chỉ định, các class sẽ không được áp dụng.

Các breakpoints mặc định trong Tailwind CSS bao gồm:

* **sm**: 640px (max-width:640px)
* **md**: 768px (max-width:768px)
* **lg**: 1024px (max-width:1024px)
* **xl**: 1280px (max-width:1280px)
* **2xl**: 1536px (max-width:1536px)

#### Columns

Là tiện ích dùng để kiểm soát số lượng cột trong một phần tử, được viết dưới dạng class colums-{size} trong đó {size} là số cột tùy chỉnh, có thể chia theo số lượng cột(‘columns-2’, ‘columns-3’,..) hoặc chiều rộng cột(‘columns-xs’, ‘columns-sm’,..)

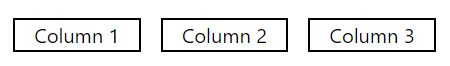
<div class="columns-3 mt-8 mx-3 ">

<p class="border-2 border-black text-center">Column 1</p>

<p class="border-2 border-black text-center">Column 2</p>

<p class="border-2 border-black text-center">Column 3</p>

</div>



Hình 2.1 Kết quả chia cột theo colums-3

#### Display

Trong Tailwind CSS, display gồm các tiện ích kiểm soát cách hiển thị một phần tử trên trang. Các tiện ích này cung cấp khả năng kiểm soát các thuộc tính CSS phổ biến, chẳng hạn như block, inline-block, flex, grid,…

Một số hiển thị mà tiện ích display có sẵn trong Tailwind:

* block: Phần tử hoạt động giống như phần tử khối.
* inline: Phần tử hoạt động giống như một phần tử nội tuyến.
* inline-block: Phần tử hoạt động giống như một phần tử khối nội tuyến.
* flex: Phần tử hoạt động như một vùng chứa flex.
* inline-flex: Phần tử hoạt động như một container flex nội tuyến.
* grid: Phần tử hoạt động như một hộp chứa dạng lưới.
* inline-grid: Phần tử hoạt động như một vùng chứa lưới nội tuyến.
* table: Phần tử hoạt động giống như một bảng.
* hidden: Phần tử bị ẩn khỏi trang.

<!-- Block Display (takes full width) -->

<div class="block bg-blue-100 p-4 mb-4">

<p>This is a block element. It takes the full width of the container.</p>

</div>

<!-- Inline-Block Display (takes only the necessary width) -->

<div class="inline-block bg-green-100 p-4">

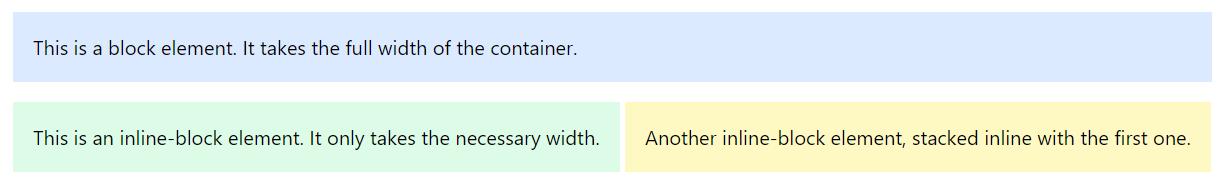
<p>This is an inline-block element. It only takes the necessary width.</p>

</div>

<div class="inline-block bg-yellow-100 p-4">

<p>Another inline-block element, stacked inline with the first one.</p>

</div>



Hình 2.2 Kết quả hiển thị block và inline-block

#### Position

Các tiện ích position cho phép định vị vị trí các phần tử trong trang web. Có thể sử dụng các lớp này để đặt các phần tử ở vị trí cụ thể dựa trên các thuộc tính CSS như static, relative, absolute, fixed và sticky.

<!-- Phần tử cha với vị trí relative -->

<div class="relative bg-blue-100 p-20 border border-blue-300 mb-4">

    <h2 class="text-xl text-center font-bold">Relative parent</h2>

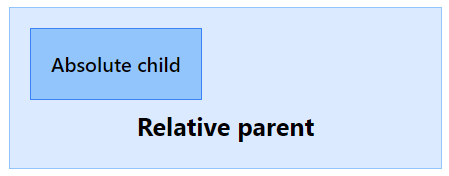
    <!-- Phần tử con với vị trí absolute -->

    <div class="absolute top-4 left-4 bg-blue-300 p-4 border border-blue-500">

        <h3 class="font-semibold">Absolute child</h3>

    </div>

</div>



Hình 2.3 Kết quả sử dụng positon relative và absolute

#### Flexbox

**Flexbox** (Flexible Box Layout) là một hệ thống bố cục trong CSS, được thiết kế để phân chia và căn chỉnh các phần tử một cách dễ dàng trong một container. Mục đích của Flexbox là cung cấp một cách sắp xếp phần tử linh hoạt, đặc biệt hữu ích khi bạn phải căn chỉnh và phân phối không gian giữa các phần tử trong một bố cục có kích thước không xác định thay đổi theo kích thước màn hình.

Một số thuộc tính của flexbox:

**Flex basis:** xác định kích thước cơ bản của một phần tử trước khi không gian còn lại được phân chia. Nó tương tự như width, nhưng linh hoạt hơn khi kết hợp với Flexbox. Giá trị có thể là một giá trị cụ thể (ví dụ: 200px, 50%) hoặc auto (mặc định, phần tử tự động điều chỉnh theo nội dung).

**Direction:** xác định hướng mà các phần tử con (flex items) sẽ được sắp xếp bên trong flex container.

* flex-row: Các phần tử con được sắp xếp theo hàng (mặc định).
* flex-col: Các phần tử con được sắp xếp theo cột.

**Justify Content:** sắp xếp các phần tử theo chiều ngang

* justify-start: Sắp xếp các phần tử con bắt đầu từ đầu.
* justify-center: Sắp xếp các phần tử con ở giữa.
* justify-end: Sắp xếp các phần tử con ở cuối.
* justify-between: Phân bố các phần tử con với khoảng cách đều.
* justify-around: Phân bố các phần tử con với khoảng cách đều, nhưng có khoảng trống bên cạnh các phần tử đầu và cuối.

**Align Items**: sắp xếp các phần tử theo chiều dọc

* items-start: Căn chỉnh các phần tử con ở đầu.
* items-center: Căn chỉnh các phần tử con ở giữa.
* items-end: Căn chỉnh các phần tử con ở dưới.
* items-stretch: Kéo dài các phần tử con để lấp đầy không gian.

**Flex Wrap:** Cho phép các phần tử con tự động xuống hàng khi kích thước chiều rộng không đủ.

* flex-wrap: Cho phép các phần tử con bọc vào hàng mới nếu không đủ không gian.
* flex-nowrap: Không bọc phần tử con (mặc định).
* flex-wrap-reverse: Bọc các phần tử con theo chiều ngược lại.

<div class="flex flex-wrap justify-between bg-gray-200 p-4">

<div class="bg-blue-300 p-4 m-2 flex-grow">Item 1</div>

<div class="bg-green-300 p-4 m-2 flex-grow">Item 2</div>

<div class="bg-red-300 p-4 m-2 flex-grow">Item 3</div>

<div class="bg-yellow-300 p-4 m-2 flex-grow">Item 4</div>

<div class="bg-purple-300 p-4 m-2 flex-grow">Item 5</div>

</div>



Hình 2.4 Kết quả sử dụng flex-wrap

#### Grid

Grid là một hệ thống bố cục mạnh mẽ cho phép tạo ra các bố cục phức tạp bằng cách sử dụng các hàng và cột. Tailwind CSS cung cấp các lớp tiện ích để làm việc với grid một cách dễ dàng và nhanh chóng.

Dưới đây là một số thuộc tính chính có thể sử dụng với Grid trong Tailwind CSS:

1. **grid**: Chuyển đổi một phần tử thành grid container.
2. **grid-cols-{n}**: Xác định số lượng cột trong grid (ví dụ: grid-cols-3 cho 3 cột).
3. **grid-rows-{n}**: Xác định số lượng hàng trong grid.
4. **col-span-{n}**: Xác định số lượng cột mà một phần tử sẽ chiếm (ví dụ: col-span-2 cho 2 cột).
5. **row-span-{n}**: Xác định số lượng hàng mà một phần tử sẽ chiếm.
6. **gap-{size}**: Đặt khoảng cách giữa các hàng và cột trong grid.

<div class="grid grid-cols-3 gap-4 bg-gray-200 p-4">

<div class="bg-blue-300 p-4">Item 1</div>

<div class="bg-green-300 p-4 col-span-2">Item 2 (Chiếm 2 cột)</div>

<div class="bg-red-300 p-4">Item 3</div>

<div class="bg-yellow-300 p-4">Item 4</div>

<div class="bg-purple-300 p-4">Item 5</div>

</div>



Hình 2.5 Kết quả sử dụng grid

#### Spacing

**Padding (Đệm)**

p-{size}: Thiết lập padding cho tất cả các phía (trên, dưới, trái, phải).

pt-{size}: Thiết lập padding cho phía trên.

pr-{size}: Thiết lập padding cho phía bên phải.

pb-{size}: Thiết lập padding cho phía dưới.

pl-{size}: Thiết lập padding cho phía bên trái.

px-{size}: Thiết lập padding cho hai bên trái và phải.

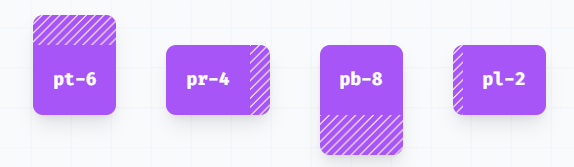
py-{size}: Thiết lập padding cho hai bên trên và dưới.

<div class="pt-6 ...">pt-6</div>

<div class="pr-4 ...">pr-4</div>

<div class="pb-8 ...">pb-8</div>

<div class="pl-2 ...">pl-2</div>



Hình 2.6 Kết quả sử dụng padding

**Margin (Lề)**

m-{size}: Thiết lập margin cho tất cả các phía.

mt-{size}: Thiết lập margin cho phía trên.

mr-{size}: Thiết lập margin cho phía bên phải.

mb-{size}: Thiết lập margin cho phía dưới.

ml-{size}: Thiết lập margin cho phía bên trái.

mx-{size}: Thiết lập margin cho hai bên trái và phải.

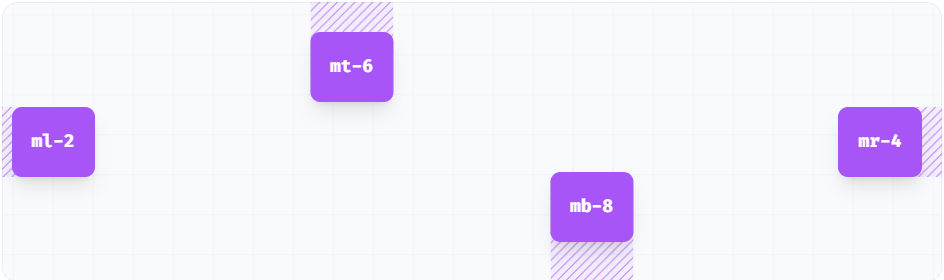
my-{size}: Thiết lập margin cho hai bên trên và dưới.

<div class="mt-6 ...">mt-6</div>

<div class="mr-4 ...">mr-4</div>

<div class="mb-8 ...">mb-8</div>

<div class="ml-2 ...">ml-2</div>



Hình 2.7 Kết quả sử dụng margin

#### Sizing

Sizing trong Tailwind CSS bao gồm các lớp tiện ích để xác định chiều rộng (width), chiều cao (height), và các kích thước khác của phần tử. Các lớp này giúp dễ dàng điều chỉnh và kiểm soát kích thước phần tử một cách nhanh chóng và linh hoạt.

**Chiều rộng (Width)**

**Chiều rộng cố định** được đặt bằng cách sử dụng các lớp như w-{size}, với các giá trị cụ thể như px, rem, hoặc các kích thước mặc định của Tailwind.

các lớp phổ biến như:

* w-1: 0.25rem (4px)
* w-4: 1rem (16px)
* w-8: 2rem (32px)
* w-16: 4rem (64px)
* w-24: 6rem (96px)
* w-32: 8rem (128px)
* w-64: 16rem (256px)
* w-96: 24rem (384px)

<div class="w-96 bg-blue-300 rounded text-center mb-4">w-96 (384px width)</div>

<div class="w-80 bg-green-300 rounded text-center mb-4">w-80 (320px width)</div>

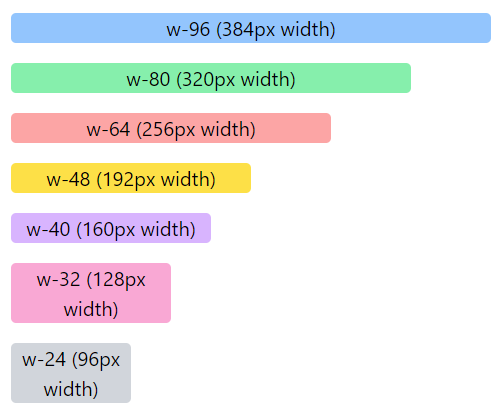
<div class="w-64 bg-red-300 rounded text-center mb-4">w-64 (256px width)</div>

<div class="w-48 bg-yellow-300 rounded text-center mb-4">w-48 (192px width)</div>

<div class="w-40 bg-purple-300 rounded text-center mb-4">w-40 (160px width)</div>

<div class="w-32 bg-pink-300 rounded text-center mb-4">w-32 (128px width)</div>

<div class="w-24 bg-gray-300 rounded text-center mb-4">w-24 (96px width)</div>



Hình 2.8 Kết quả các kích thước chiều rộng

**Chiều rộng phần trăm:** xác định chiều rộng của phần tử dựa trên phần trăm chiều rộng của phần tử cha. Các lớp trong Tailwind như w-1/2, w-1/3, w-1/4,... sẽ phân chia chiều rộng theo tỷ lệ phần trăm.

Các lớp phổ biến:

* w-1/2: Chiều rộng 50% của phần tử cha.
* w-1/3: Chiều rộng 33.33% của phần tử cha.
* w-2/3: Chiều rộng 66.67% của phần tử cha.
* w-1/4: Chiều rộng 25% của phần tử cha.
* w-full: Chiều rộng 100% của phần tử cha.

<!-- Flex with w-1/2 -->

<div class="flex mb-4 gap-3">

<div class="w-1/2 bg-blue-300 p-2 rounded">w-1/2</div>

<div class="w-1/2 bg-green-300 p-2 rounded">w-1/2</div>

</div>

<!-- Flex with w-2/5 and w-3/5 -->

<div class="flex mb-4 gap-3">

<div class="w-2/5 bg-red-300 p-2 rounded">w-2/5</div>

<div class="w-3/5 bg-yellow-300 p-2 rounded">w-3/5</div>

</div>

<!-- Flex with w-1/3 and w-2/3 -->

<div class="flex mb-4 gap-3">

<div class="w-1/3 bg-purple-300 p-2 rounded">w-1/3</div>

<div class="w-2/3 bg-pink-300 p-2 rounded">w-2/3</div>

</div>

<!-- Flex with w-1/4 and w-3/4 -->

<div class="flex mb-4 gap-3">

<div class="w-1/4 bg-gray-300 p-2 rounded">w-1/4</div>

<div class="w-3/4 bg-indigo-300 p-2 rounded">w-3/4</div>

</div>

<!-- Flex with w-1/5 and w-4/5 -->

<div class="flex mb-4 gap-3">

<div class="w-1/5 bg-orange-300 p-2 rounded">w-1/5</div>

<div class="w-4/5 bg-teal-300 p-2 rounded">w-4/5</div>

</div>

<!-- Flex with w-1/6 and w-5/6 -->

<div class="flex mb-4 gap-3">

<div class="w-1/6 bg-blue-500 p-2 rounded">w-1/6</div>

<div class="w-5/6 bg-green-500 p-2 rounded">w-5/6</div>

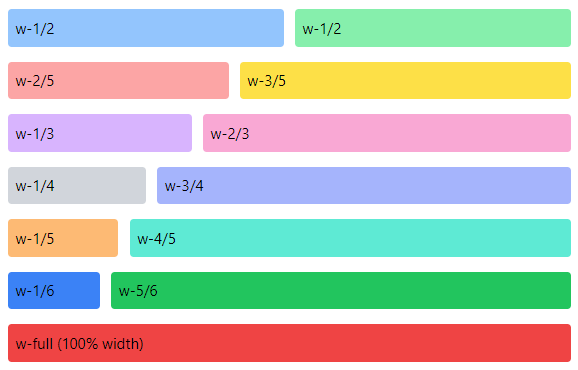
</div>

<!-- Full width element -->

<div class="w-full bg-red-500 p-2 rounded">

w-full (100% width)

</div>



Hình 2.9 Kết quả thiết lập chiều rộng theo phần trăm

**Chiều cao (Height)**

**Chiều cao cố định**: Các lớp h-{size} trong Tailwind CSS cho phép đặt chiều cao cố định cho các phần tử bằng đơn vị rem hoặc các kích thước đã được định nghĩa sẵn.

Các lớp phổ biến:

* h-4: 1rem (16px)
* h-8: 2rem (32px)
* h-16: 4rem (64px)
* h-24: 6rem (96px)
* h-32: 8rem (128px)
* h-48: 12rem (192px)
* h-64: 16rem (256px)
* h-96: 24rem (384px)
* h-auto: Chiều cao tự động, phụ thuộc vào nội dung.

<div class="flex flex-row items-end space-y-4 gap-3">

<div class="h-96 w-12 bg-blue-500 flex items-center justify-center text-white">h-96</div>

<div class="h-80 w-12 bg-blue-500 flex items-center justify-center text-white">h-80</div>

<div class="h-64 w-12 bg-blue-500 flex items-center justify-center text-white">h-64</div>

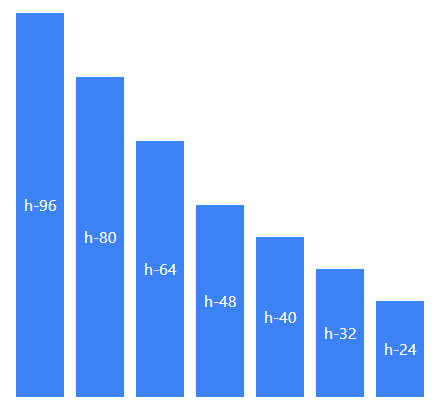
<div class="h-48 w-12 bg-blue-500 flex items-center justify-center text-white">h-48</div>

<div class="h-40 w-12 bg-blue-500 flex items-center justify-center text-white">h-40</div>

<div class="h-32 w-12 bg-blue-500 flex items-center justify-center text-white">h-32</div>

<div class="h-24 w-12 bg-blue-500 flex items-center justify-center text-white">h-24</div>

</div>

****

Hình 2.10 Kết quả thiết lập chiều cao cố định

**Chiều cao phần trăm**: đặt chiều cao của phần tử theo phần trăm bằng cách sử dụng các lớp như h-1/2, h-full, v.v.

Các lớp phổ biến:

* h-1/2: Chiều cao 50% so với phần tử cha.
* h-full: Chiều cao 100% của phần tử cha.
* h-screen: Chiều cao 100% của màn hình hiển thị (viewport height).

**Chiều cao tùy chỉnh**: sử dụng lớp min-h hoặc max-h để đặt chiều cao tối thiểu hoặc tối đa cho phần tử.

Các lớp phổ biến:

* min-h-0: Chiều cao tối thiểu là 0.
* min-h-full: Chiều cao tối thiểu là 100% của phần tử cha.
* max-h-full: Chiều cao tối đa là 100% của phần tử cha.

#### Typography

**Font Family**: Sử dụng lớp font-{family} để thay đổi kiểu font.

* font-sans: Font sans-serif.
* font-serif: Font serif.
* font-mono: Font monospace.

**Font Weight**: Thay đổi độ dày của font.

* font-thin: Độ dày 100.
* font-extralight: Độ dày 200.
* font-light: Độ dày 300.
* font-normal: Độ dày 400 (mặc định).
* font-medium: Độ dày 500.
* font-semibold: Độ dày 600.
* font-bold: Độ dày 700.
* font-extrabold: Độ dày 800.
* font-black: Độ dày 900.

**Font Size**: Sử dụng lớp text-{size} để điều chỉnh kích thước chữ.

* text-xs: Kích thước chữ nhỏ (0.75rem).
* text-sm: Kích thước chữ nhỏ hơn (0.875rem).
* text-base: Kích thước chữ cơ bản (1rem).
* text-lg: Kích thước chữ lớn (1.125rem).
* text-xl: Kích thước chữ rất lớn (1.25rem).
* text-2xl: Kích thước chữ lớn hơn nữa (1.5rem).
* text-3xl: Kích thước chữ rất lớn (1.875rem).

**Font Style**: Sử dụng lớp italic để làm cho chữ nghiêng và not-italic để bỏ nghiêng.

**Text Color**: Thay đổi màu chữ bằng cách sử dụng các lớp màu như text-red-500, text-blue-600, v.v.

<h1 class="text-3xl font-bold text-blue-700 mb-4">Tiêu đề Chính</h1>

    <p class="text-base font-normal text-gray-800 mb-2">Đây là một đoạn văn bản với font chữ cơ bản.</p>

    <p class="text-lg font-semibold text-gray-600 mb-2">Đoạn văn bản này có độ dày chữ trung bình.</p>

    <p class="text-sm font-light text-gray-500 italic">Đoạn văn bản này là nghiêng và có độ dày nhẹ.</p>

    <p class="text-xl font-bold text-red-500">Đoạn văn bản này có chữ đậm và màu đỏ.</p>



Hình 2.11 Kết quả sử dụng một số lớp tiện ích trong Typography

#### Backgrounds

**Background Color**: Sử dụng lớp bg-{color} để đặt màu nền. Tailwind CSS có sẵn rất nhiều màu với các mức độ khác nhau.

Ví dụ về một số màu

bg-red-500: Màu nền đỏ.

bg-blue-600: Màu nền xanh.

bg-green-400: Màu nền xanh lá.

bg-gray-800: Màu nền xám đậm.

bg-yellow-300: Màu nền vàng.

   <button class="bg-red-500 p-2 rounded text-white font-semibold">bg-red-500</button>

    <button class="bg-green-500 p-2 rounded text-white font-semibold">bg-green-500</button>

    <button class="bg-blue-500 p-2 rounded text-white font-semibold">bg-blue-500</button>

    <button class="bg-yellow-500 p-2 rounded text-white font-semibold">bg-yellow-500</button>

    <button class="bg-indigo-500 p-2 rounded text-white font-semibold">bg-indigo-500</button>

    <button class="bg-purple-500 p-2 rounded text-white font-semibold">bg-purple-500</button>



Hình 2.12 Kết quả sử dụng màu nền

**Background Size**: Bạn có thể điều chỉnh kích thước hình ảnh nền với lớp bg-cover, bg-contain, hoặc bg-auto.

bg-cover: Hình ảnh nền sẽ bao phủ toàn bộ phần tử.

bg-contain: Hình ảnh nền sẽ co giãn để vừa với kích thước phần tử.

**Background Position**: Điều chỉnh vị trí của hình ảnh nền với các lớp như:

bg-center: Căn giữa hình nền.

bg-top, bg-bottom: Đặt hình nền ở trên hoặc dưới.

**Background Repeat**: Điều chỉnh việc lặp lại hình ảnh nền với lớp bg-repeat, bg-no-repeat, hoặc bg-repeat-x/y (lặp theo trục X hoặc Y).

#### Borders

Trong **Tailwind CSS**, thuộc tính **border** được quản lý thông qua nhiều lớp tiện ích khác nhau như độ dày, màu sắc, kiểu viền và bo góc.

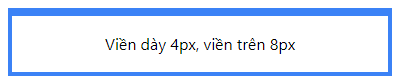
**Border Width (độ dày viền)**

* border: Tạo viền mặc định với độ dày 1px.
* border-2, border-4, border-8: Độ dày của viền lần lượt là 2px, 4px, 8px.
* border-t, border-b, border-l, border-r: Tạo viền riêng cho từng cạnh (trên, dưới, trái, phải).

<div class="border-4 border-t-8 border-blue-500 p-4">

Viền dày 4px, viền trên 8px

</div>

****

Hình 2.13 Kết quả tạo đường viền

**Border Color(màu sắc viền)**

border-{color}: Tạo viền với màu xác định.

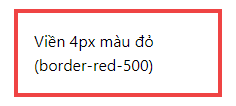
Ví dụ border-red-500, border-blue-600, border-gray-300.

Có thể chọn từ bảng màu có sẵn trong Tailwind với các sắc độ khác nhau.

<div class="border-4 border-red-500 p-4">

Viền 4px màu đỏ (border-red-500)

</div>



Hình 2.14 Kết quả tạo viền đỏ

**Border Style (kiểu viền)**

* border-solid: Viền nét liền (mặc định).
* border-dashed: Viền nét đứt.
* border-dotted: Viền chấm.

<!-- Viền nét liền (solid) -->

<button class="border-solid border-2 border-indigo-600 p-4">

Viền nét liền (solid)

</button>

<!-- Viền nét đứt (dashed) -->

<button class="border-dashed border-2 border-indigo-600 p-4">

Viền nét đứt (dashed)

</button>

<!-- Viền chấm (dotted) -->

<button class="border-dotted border-2 border-indigo-600 p-4">

Viền chấm (dotted)

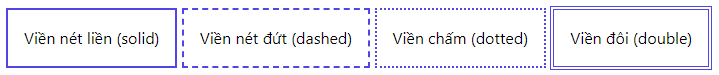
</button>

<!-- Viền đôi (double) -->

<button class="border-double border-4 border-indigo-600 p-4">

Viền đôi (double)

</button>



Hình 2.15 Kết quả tạo kiểu viền

#### Table

TailWind cung cấp 2 lớp tiện ích cho bố cục bảng:

table-auto: Điều chỉnh bố cục của bảng dựa trên nội dung. Đây là mặc định.

table-fixed: Buộc bảng có bố cục cố định, chiều rộng các cột sẽ bằng nhau, văn bản có thể bị cắt ngắn nếu quá dài.

<table class="table-fixed min-w-full border border-gray-400">

<thead>

<tr class="bg-gray-200">

<th class="w-1/3 px-4 py-2">Tên</th>

<th class="w-1/3 px-4 py-2">Tuổi</th>

<th class="w-1/3 px-4 py-2">Địa điểm</th>

</tr>

</thead>

<tbody>

<tr>

<td class="border px-4 py-2">John Doe</td>

<td class="border px-4 py-2">32</td>

<td class="border px-4 py-2">New York</td>

</tr>

</tbody>

</table>

**Caption Side (Vị trí tiêu đề bảng)**

Xác định vị trí của tiêu đề bảng (caption) so với bảng.

<table class="min-w-full table-auto">

<caption class="text-lg font-semibold my-2" style="caption-side: top;">Tiêu đề bảng</caption>

<thead>

<tr>

<th class="px-4 py-2">Tên</th>

<th class="px-4 py-2">Tuổi</th>

<th class="px-4 py-2">Địa điểm</th>

</tr>

</thead>

<tbody>

<tr>

<td class="border px-4 py-2">John Doe</td>

<td class="border px-4 py-2">32</td>

<td class="border px-4 py-2">New York</td>

</tr>

</tbody>

</table>

## Giới thiệu về React

### Tổng quan về React

**React** (ReactJS) là một thư viện [JavaScript](https://200lab.io/blog/javascript-la-gi/) mã nguồn mở, được dùng để xây dựng giao diện người dùng (frontend) cho web. React chỉ tập trung vào phần hiển thị giao diện (view), chứ không can thiệp vào cách sắp xếp logic nghiệp vụ hoặc cấu trúc ứng dụng

Các tính năng nổi bật của Reactjs:

**Component:** React chia giao diện người dùng thành các thành phần (component) nhỏ, độc lập và có thể tái sử dụng. Mỗi component đại diện cho một phần của giao diện và chúng có thể chứa HTML (JSX) và logic JavaScript để tạo nên các chức năng cụ thể.

**Virtual DOM:** ReactJS sử dụng Virtual DOM để tối ưu hóa hiệu suất của ứng dụng. Virtual DOM là một bản sao của DOM được lưu trữ trong bộ nhớ và được cập nhật một cách nhanh chóng khi có thay đổi, giúp tăng tốc độ và hiệu suất của ứng dụng.

**JSX (JavaScript XML):** JSX là một ngôn ngữ lập trình phân biệt được sử dụng trong ReactJS để mô tả các thành phần UI. JSX kết hợp HTML và JavaScript, giúp cho việc viết mã dễ hiểu và dễ bảo trì hơn.

**State và Props:** ReactJS cho phép quản lý trạng thái của các thành phần UI thông qua State và Props

**State** là dữ liệu nội bộ của một component, có thể thay đổi theo thời gian và ảnh hưởng đến giao diện.

**Props** (viết tắt của "Properties") là các tham số được truyền từ component cha xuống component con. Props không thay đổi trong suốt vòng đời của component đó.

**Hỗ trợ đa nền tảng:** ReactJS không chỉ được sử dụng để phát triển ứng dụng web, mà còn được sử dụng để phát triển ứng dụng di động với React Native. Sử dụng React Native, các nhà phát triển có thể xây dựng ứng dụng di động cho cả iOS và Android sử dụng cùng một mã nguồn.

**Redux:** Redux là một thư viện quản lý trạng thái cho các ứng dụng ReactJS. Nó giúp quản lý trạng thái của ứng dụng một cách chính xác và dễ dàng, đồng thời giúp tăng tính linh hoạt và khả năng mở rộng của ứng dụng.

**React Hooks:** Từ React 16.8, Hooks cho phép sử dụng state và các tính năng khác của React trong các function component mà không cần sử dụng class component.

### Cách cài đặt React

**Cài đặt trực tiếp React từ tag <script>**

Cách nhanh nhất để bắt đầu học React là viết React trực tiếp vào tệp HTML.

Tạo file index.html với nội dung như sau:

<!DOCTYPE html>

<html>

<head>

<script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>

<script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>

<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

</head>

<body>

<div id="mydiv"></div>

<script type="text/babel">

function Hello() {

return <h1>Welcome to Reactjs</h1>;

}

const container = document.getElementById('mydiv');

const root = ReactDOM.createRoot(container);

root.render(<Hello />)

</script>

</body>

</html>

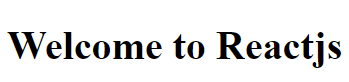
**react.development.js và react-dom.development.js**: Đây là hai file thư viện cốt lõi của React và ReactDOM, cần thiết để chạy ứng dụng React.

**babel.min.js:** Thư viện này giúp trình duyệt hiểu được cú pháp JSX (JavaScript XML), cho phép bạn viết các thành phần React như HTML trực tiếp trong JavaScript.

**JSX:** Trong đoạn mã <h1>Hello World!</h1>, bạn sử dụng JSX để tạo giao diện. Đây là cú pháp rất quen thuộc của React, giúp viết mã dễ dàng hơn.

**root.render:** Đây là cách React render nội dung vào phần tử DOM, trong trường hợp này là <div id="mydiv"></div>

Khi click chạy file index.html, trình duyệt sẽ cho ta kết quả sau:



Hình 2.16 Kết quả chạy React

**Cài đặt React bằng lệnh**

Bước 1: Cài đặt Node.js và npm

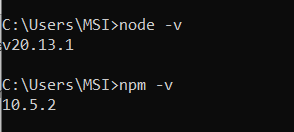
Trước tiên, cần cài đặt Node.js vì ReactJS sử dụng npm (Node Package Manager) để quản lý các gói và thư viện.

Tải Node.js từ trang chủ: https://nodejs.org

Sau khi cài đặt, kiểm tra phiên bản Node.js và npm trong terminal/cmd:

node -v

npm -v

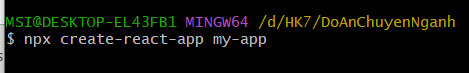


Bước 2: Tạo dự án React bằng Create React App

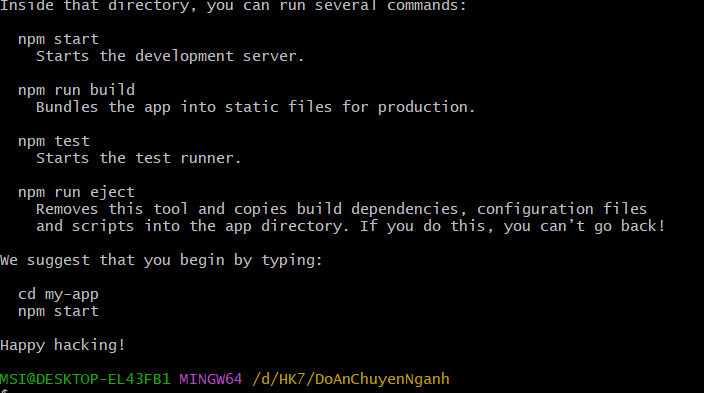
Create React App là công cụ dễ sử dụng nhất để tạo dự án React.

Mở terminal hoặc command prompt.

Chạy lệnh sau để tạo một dự án React:

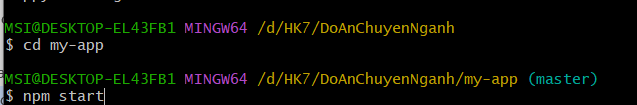


Thay my-app bằng tên dự án, khi tạo thành công sẽ hiển thị thông báo Happy hacking!.

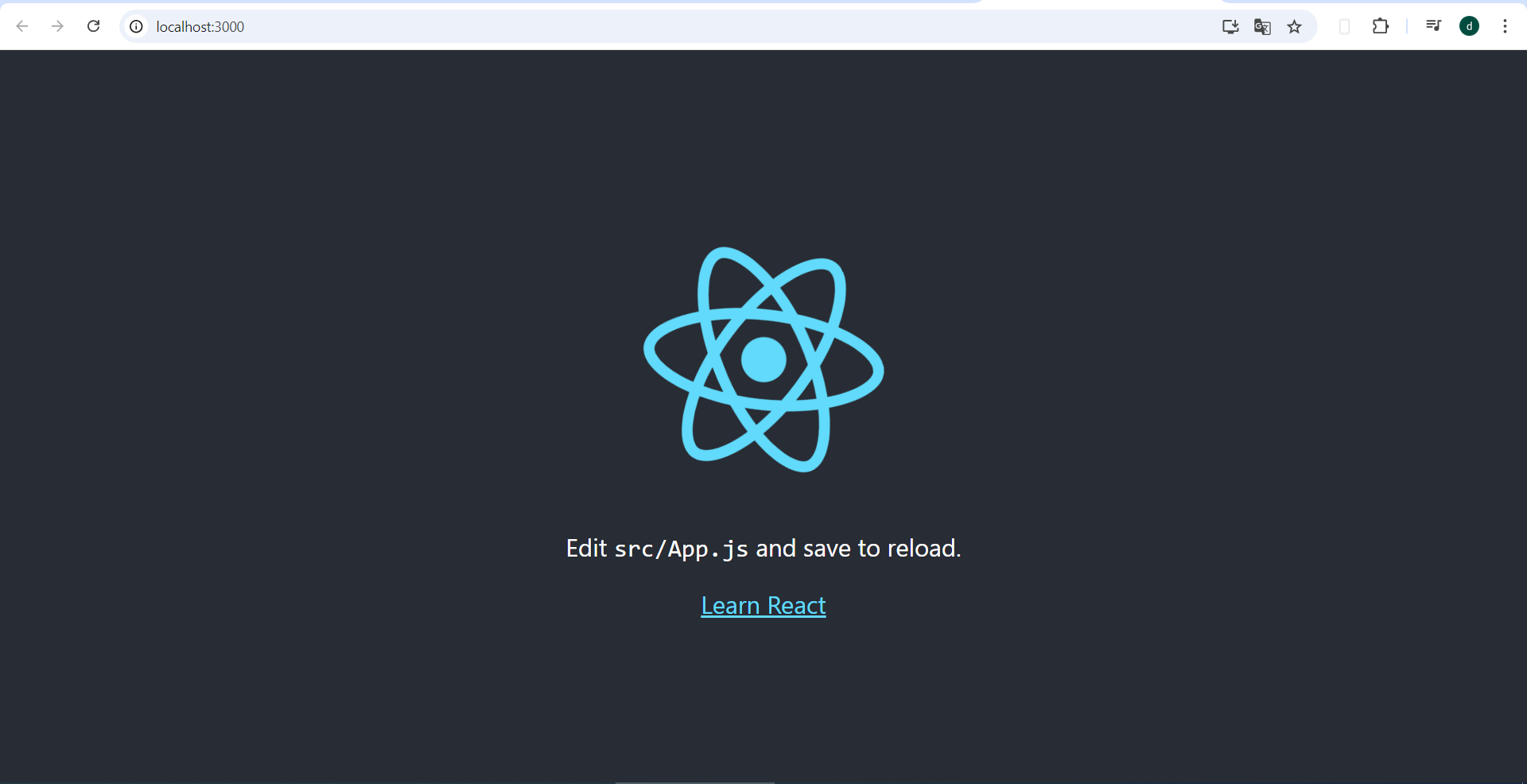


Bước 3: Chạy ứng dụng React

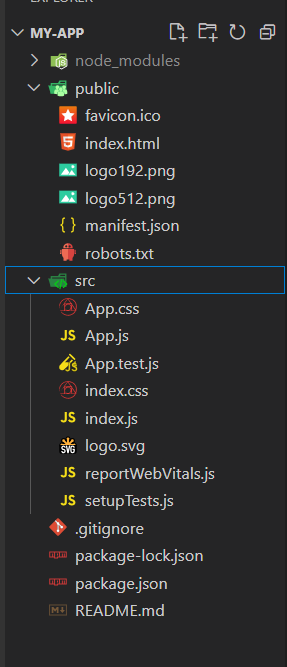
Sau khi tạo dự án thành công, thì ta sẽ truy cập vào dự án và chạy ứng dụng



Ứng dụng sẽ được chạy trên trình duyệt tại địa chỉ <http://localhost:3000>.



### Cấu trúc của React



Cấu trúc này bao gồm các thư mục và tệp quan trọng cho một ứng dụng React:

**1. node\_modules/**

Thư mục này chứa tất cả các thư viện và dependencies mà dự án cần, được quản lý bởi npm.

**2. public/**

Thư mục này chứa các tài nguyên công cộng của dự án, bao gồm:

favicon.ico: Icon nhỏ hiển thị trên tab trình duyệt.

index.html: Tệp HTML chính của ứng dụng. Đây là nơi mà toàn bộ ứng dụng React sẽ được "render" vào. React sẽ gắn kết ứng dụng vào thẻ <div id="root"></div> trong tệp này.

logo192.png & logo512.png: Các tệp logo cho ứng dụng.

manifest.json: Cung cấp thông tin về ứng dụng, đặc biệt hữu ích khi bạn muốn ứng dụng của mình hoạt động như một Progressive Web App (PWA).

robots.txt: Quy định cách các công cụ tìm kiếm thu thập thông tin từ trang web.

**3. src/**

Thư mục này chứa mã nguồn chính của ứng dụng, bao gồm:

App.css: Tệp CSS cho component App.

App.js: Component chính của ứng dụng, nơi logic chính của giao diện được viết.

App.test.js: Tệp dùng để kiểm thử component App bằng Jest.

index.css: Tệp CSS chung cho toàn bộ ứng dụng.

index.js: Điểm vào chính của ứng dụng React. Tại đây, ReactDOM.render() được sử dụng để render ứng dụng vào thẻ root trong index.html.

logo.svg: Logo dưới định dạng SVG.

reportWebVitals.js: Tệp này được sử dụng để đo lường hiệu suất của ứng dụng.

setupTests.js: Cấu hình kiểm thử với Jest.

**4. Các tệp khác**

.gitignore: Xác định các tệp và thư mục sẽ bị bỏ qua bởi Git.

package.json: Tệp định nghĩa thông tin về dự án và các dependencies cần thiết.

package-lock.json: Quản lý các phiên bản chính xác của dependencies.

README.md: Tệp hướng dẫn hoặc thông tin về dự án.

### Các khái niệm cơ bản trong React

#### Components

Có 2 thành phần chính trong components là class components và function components.

**Class components**: Trước khi React 16.8 giới thiệu hooks, class component là cách duy nhất để sử dụng state và các phương thức vòng đời.

Class component sử dụng cú pháp của các lớp JavaScript và cần phải kế thừa từ React.Component.

class Info extends React.Component {

render() {

return <h1>Name:{this.props.name}</h1>;

}

}

**Function components:** trả về HTML và hoạt động gần giống với thành phần Class, nhưng thành phần Function có thể được viết bằng ít mã hơn, dễ hiểu.

function Info(props) {

return <h1>Name:{props.name}</h1>;

}

**Components in Components**: là component bên trong một component khác, cho phép tổ chức các phần của giao diện người dùng thành các thành phần độc lập và dễ quản lý hơn.

// Component nhỏ để hiển thị một người dùng

function UserCard(props) {

return (

<div className="user-card">

<h2>{props.name}</h2>

<p>Email: {props.email}</p>

</div>

);

}

// Component lớn hơn để hiển thị danh sách người dùng

function UserList() {

const users = [

{ name: "Vo Thi Diem", email: "vodiem3002@gmail.com" },

{ name: "Pham Thuy Hang", email: "thuyhang2003@gmail.com" },

{ name: "Diep Tu Nhu", email: " tunhu2003@gmail.com " }

];

return (

<div className="user-list">

{users.map((user, index) => (

<UserCard key={index} name={user.name} email={user.email} />

))}

</div>

);

}

#### Props

**Props:** là các tham số đầu vào của component, được truyền từ component cha đến component con. Props là không thể thay đổi và được dùng để truyền dữ liệu hoặc hàm từ cha xuống con.

Mỗi lần props thay đổi, component con sẽ tự động re-render để cập nhật giao diện.

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

function App() {

return <Welcome name="Alice" />;

}

#### Xử lý Events

Xử ý Events trong React khá giống với cách xử lý sự kiện trong HTML thông thường, nhưng có một số khác biệt nhỏ. React sử dụng các **synthetic events** (sự kiện tổng hợp), giúp đảm bảo tính tương thích của các sự kiện trên mọi trình duyệt.

Ví dụ về cách viết xử lý sự kiện của React.

function showMessage() {

console.log('Hello, this is a console message!');

}

function App() {

return <button onClick={showMessage}>Show Message</button>;

}

export default App;

#### Conditional

**Sử dụng If**

Có thể sử dụng câu lệnh if để kiểm tra điều kiện và trả về giao diện khác nhau dựa trên điều kiện đó.

Ví dụ: Chuyển Đổi Giữa Chế Độ Ngày và Đêm

import React, { useState } from 'react';

function App() {

const [isDay, setIsDay] = useState(true); // Trạng thái để kiểm tra chế độ

const toggleMode = () => {

setIsDay(prevMode => !prevMode); // Chuyển đổi giữa chế độ ngày và đêm

};

return (

<div style={{

backgroundColor: isDay ? '#ffffff' : '#333333',

color: isDay ? '#000000' : '#ffffff',

height: '100vh',

display: 'flex',

justifyContent: 'center',

alignItems: 'center',

flexDirection: 'column'

}}>

<h1>{isDay ? 'Good Morning!' : 'Good Night!'}</h1>

<button onClick={toggleMode}>

Switch to {isDay ? 'Night' : 'Day'} Mode

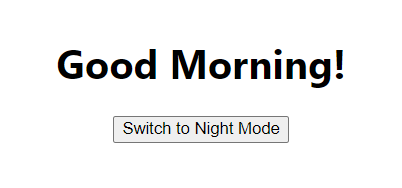
</button>

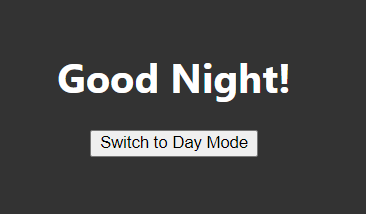
</div>

);

}

export default App;





Hình 2.17 Kết quả của trước/sau khi nhấn nút, ngược lại

**Toán tử logic &&:** làmột cách tiện lợi để thực hiện conditional rendering (kết xuất có điều kiện). Khi sử dụng toán tử &&, nếu điều kiện phía trước là true, React sẽ hiển thị phần tử bên phải. Nếu điều kiện là false, React sẽ bỏ qua phần tử bên phải và không kết xuất nó.

Cách viết: { condition && <Component /> }

Trong đó:

* Nếu condition là true, thì <Component /> sẽ được hiển thị.
* Nếu condition là false, không có gì được hiển thị

Ví dụ về ẩn/hiện nút đăng xuất khi người dùng đã đăng nhập

function App() {

const isLoggedIn = true; // Thay đổi thành false để thử nghiệm

return (

<div>

<h1>Dashboard</h1>

{isLoggedIn && <button>Log out</button>}

</div>

);

}

export default App;

**Toán tử 3 ngôi:** là một cách gọn gàng và hiệu quả để thực hiện điều kiện trong JavaScript và React. Toán tử này có cú pháp đơn giản hơn so với câu lệnh if...else và rất hữu ích khi cần đưa ra một lựa chọn giữa hai kết quả.

Cú pháp: condition ? expressionIfTrue : expressionIfFalse;

Trong đó:

* condition: Biểu thức điều kiện (nếu điều kiện này đúng thì sẽ thực hiện kết quả đầu tiên).
* expressionIfTrue: Biểu thức sẽ được thực hiện nếu điều kiện đúng.
* expressionIfFalse: Biểu thức sẽ được thực hiện nếu điều kiện sai.

**Ví dụ:** hiển thị nội dung dựa trên trạng thái đăng nhập của người dùng

function App() {

const isLoggedIn = true; // Thay đổi thành false để kiểm tra kết quả khác

return (

<div>

<h1>{isLoggedIn ? "Welcome back!" : "Please log in."}</h1>

</div>

);

}

export default App;

#### Lists và Keys

Trong React, danh sách (lists) cho phép bạn kết xuất một tập hợp các phần tử từ một mảng dữ liệu. Điều này rất hữu ích khi bạn cần hiển thị các mục giống nhau, chẳng hạn như sản phẩm, người dùng, hoặc bài viết.

Phương thức map(): Sử dụng map() để lặp qua mảng và tạo ra một danh sách các phần tử.

**Cấu trúc cơ bản:**

const items = ['Item 1', 'Item 2', 'Item 3'];

function ItemList() {

return (

<ul>

{items.map((item, index) => (

<li key={index}>{item}</li>

))}

</ul>

);

}

export default ItemList;

Trong ví dụ trên, mỗi phần tử trong mảng items được kết xuất thành một phần tử <li>

**Tại sao cần key trong danh sách?**

Khi bạn kết xuất danh sách các phần tử, React yêu cầu mỗi phần tử có một thuộc tính key duy nhất. Điều này giúp React xác định phần tử nào đã thay đổi, được thêm vào hoặc bị xóa, từ đó tối ưu hóa quá trình cập nhật DOM.

**Tại sao key quan trọng?**

- Hiệu suất: Việc sử dụng key giúp React xác định các phần tử thay đổi và tối ưu hóa việc tái kết xuất.

- Tránh lỗi: Không có key hoặc sử dụng key không duy nhất có thể dẫn đến hành vi không mong muốn, chẳng hạn như việc giữ lại trạng thái của các phần tử sai.

**Cách chọn key**

- Sử dụng giá trị duy nhất: Nếu mỗi phần tử trong danh sách có một ID duy nhất (như trong cơ sở dữ liệu), hãy sử dụng giá trị đó làm key.

- Tránh sử dụng chỉ số mảng: Sử dụng chỉ số mảng (như trong ví dụ trên) có thể dẫn đến lỗi khi danh sách được thay đổi. Ví dụ, nếu bạn thêm hoặc xóa một mục, các chỉ số sẽ thay đổi, dẫn đến cập nhật không chính xác.

Ví dụ với key

const users = [

{ id: 1, name: 'John' },

{ id: 2, name: 'Jane' },

{ id: 3, name: 'Tom' }

];

function UserList() {

return (

<ul>

{users.map((user) => (

<li key={user.id}>{user.name}</li>

))}

</ul>

);

}

export default UserList;

#### Router

Router được sử dụng để chuyển hướng giữa các trang web với nhau trong ứng dụng web. Để sử dụng cần cài đặt React Router vào trong dự án.

npm install react-router-dom

**Cấu trúc cơ bản của Router**

import React from 'react';

import { BrowserRouter as Router, Route, Switch, Link } from 'react-router-dom';

function Home() {

return <h2>Home Page</h2>;

}

function About() {

return <h2>About Page</h2>;

}

function NotFound() {

return <h2>404 Not Found</h2>;

}

function App() {

return (

<Router>

<nav>

<ul>

<li>

<Link to="/">Home</Link>

</li>

<li>

<Link to="/about">About</Link>

</li>

</ul>

</nav>

<Switch>

<Route path="/" exact component={Home} />

<Route path="/about" component={About} />

<Route component={NotFound} />

</Switch>

</Router>

);

}

export default App;

**BrowserRouter:** Đây là Router chính của ứng dụng. Nó sử dụng API của trình duyệt để điều hướng giữa các trang.

**Route:** Đây là thành phần để định nghĩa các route. Bạn có thể chỉ định path và thành phần tương ứng sẽ được hiển thị khi URL trùng với path.

**Link:** Thành phần này được sử dụng để điều hướng đến các route khác mà không cần tải lại trang.

**Switch:** Chỉ định rằng chỉ có một trong các route con được hiển thị tại một thời điểm. Nó sẽ kiểm tra từng route và render route đầu tiên khớp với URL.

### Hooks

**Hooks** trong React cho phép bạn sử dụng state và các tính năng khác của React trong các function components, giúp việc quản lý trạng thái và các side effect trở nên đơn giản và dễ hiểu hơn. Trước khi có hooks, state và các lifecycle methods chỉ có thể được sử dụng trong class components. Hooks đã ra mắt từ phiên bản React 16.8 và mang lại một cách tiếp cận hiện đại hơn.

#### UseState

UseState là hook được sử dụng để khai báo và quản lý state trong function component. Nó trả về một cặp giá trị: giá trị hiện tại của state và một hàm để cập nhật giá trị đó.

Cú pháp: **const [state, setState] = useState(initialState);**

**Ví dụ:**

import React, { useState } from 'react';

function Counter() {

const [count, setCount] = useState(0);

return (

<div>

<p>You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

export default Counter;

useState(0) khởi tạo state count với giá trị ban đầu là 0. Khi người dùng nhấn nút, setCount được gọi để tăng giá trị của count.

#### useEffect

useEffect là hook cho phép bạn thực hiện các side effects trong function component, như gọi API, cập nhật DOM, hoặc đăng ký các event. Nó tương đương với các phương thức componentDidMount, componentDidUpdate, và componentWillUnmount trong class component.

Cú pháp:

**useEffect(() => {**

**// Side effect**

**return () => {**

**// Cleanup (tùy chọn)**

**};**

**}, [dependencies]);**

**Ví dụ:**

import React, { useState, useEffect } from 'react';

function Timer() {

const [seconds, setSeconds] = useState(0);

useEffect(() => {

const interval = setInterval(() => {

setSeconds(seconds => seconds + 1);

}, 1000);

return () => clearInterval(interval); // Cleanup khi component unmount

}, []);

return <div>Timer: {seconds} seconds</div>;

}

export default Timer;

Trong ví dụ này, useEffect khởi tạo một setInterval khi component được render lần đầu tiên. Khi component unmount, clearInterval sẽ được gọi để dọn dẹp interval.

#### useContext

useContext cho phép bạn sử dụng context mà không cần phải sử dụng các thành phần trung gian (props drilling). Nó giúp truyền dữ liệu từ component cha xuống các component con ở nhiều cấp độ mà không cần truyền qua từng cấp.

Ví dụ:

import React, { useContext } from 'react';

const ThemeContext = React.createContext('light');

function ThemedButton() {

const theme = useContext(ThemeContext);

return <button style={{ background: theme === 'dark' ? '#333' : '#FFF' }}>I am styled by {theme} theme</button>;

}

function App() {

return (

<ThemeContext.Provider value="dark">

<ThemedButton />

</ThemeContext.Provider>

);

}

export default App;

useContext(ThemeContext) giúp component ThemedButton truy cập trực tiếp vào giá trị context ("dark") mà không cần truyền qua props.

#### useReducer

useReducer là hook thường được sử dụng để quản lý state phức tạp hơn, đặc biệt khi nhiều biến state có liên quan đến nhau. Nó giống với mô hình reducer trong Redux.

Cú pháp: **const [state, dispatch] = useReducer(reducer, initialState);**

import React, { useReducer } from 'react';

const initialState = { count: 0 };

function reducer(state, action) {

switch (action.type) {

case 'increment':

return { count: state.count + 1 };

case 'decrement':

return { count: state.count - 1 };

default:

return state;

}

}

function Counter() {

const [state, dispatch] = useReducer(reducer, initialState);

return (

<div>

<p>Count: {state.count}</p>

<button onClick={() => dispatch({ type: 'increment' })}>+</button>

<button onClick={() => dispatch({ type: 'decrement' })}>-</button>

</div>

);

}

export default Counter;

useReducer giúp quản lý logic tăng và giảm giá trị của state count dựa trên các hành động được dispatch.

#### useRef

useRef cung cấp một cách để giữ tham chiếu đến một phần tử DOM hoặc lưu trữ một giá trị nào đó mà không cần render lại component.

import React, { useRef } from 'react';

function FocusInput() {

const inputRef = useRef(null);

const handleClick = () => {

inputRef.current.focus();

};

return (

<div>

<input ref={inputRef} type="text" />

<button onClick={handleClick}>Focus the input</button>

</div>

);

}

export default FocusInput;

#### useMemo

useMemo giúp tối ưu hiệu suất bằng cách chỉ tính toán lại giá trị khi một trong các dependencies thay đổi. Nó hữu ích khi bạn có các tính toán tốn kém cần tránh lặp lại không cần thiết.

import React, { useState, useMemo } from 'react';

function ExpensiveComponent({ num }) {

const computeFactorial = n => {

console.log('Computing factorial...');

return n <= 0 ? 1 : n \* computeFactorial(n - 1);

};

const factorial = useMemo(() => computeFactorial(num), [num]);

return <div>Factorial of {num}: {factorial}</div>;

}

export default ExpensiveComponent;

#### useCallback

useCallback trả về một phiên bản memoized của một callback function, giúp tránh việc tạo lại hàm mỗi khi component render, rất hữu ích khi bạn truyền các hàm vào component con để tối ưu hóa hiệu suất.

Ví dụ:

import React, { useState, useCallback } from 'react';

function Button({ handleClick }) {

return <button onClick={handleClick}>Click me</button>;

}

function App() {

const [count, setCount] = useState(0);

const increment = useCallback(() => {

setCount(prevCount => prevCount + 1);

}, []);

return (

<div>

<p>Count: {count}</p>

<Button handleClick={increment} />

</div>

);

}

export default App;

### Axios

**Axios** là một thư viện HTTP client dành cho JavaScript, cho phép bạn thực hiện các yêu cầu HTTP dễ dàng. Nó rất phổ biến trong các ứng dụng React, Vue, và Node.js do tính năng mạnh mẽ và cú pháp dễ sử dụng.

**Đặc điểm của Axios**

Dễ sử dụng: Cú pháp đơn giản và dễ hiểu, giúp bạn thực hiện các yêu cầu HTTP nhanh chóng.

Hỗ trợ Promise: Axios trả về một Promise, cho phép bạn sử dụng các phương thức .then() và .catch() để xử lý kết quả.

Tự động chuyển đổi dữ liệu: Axios tự động chuyển đổi dữ liệu phản hồi thành định dạng JSON, giúp bạn không cần gọi thêm phương thức như .json() trong Fetch.

Bắt lỗi tự động: Axios tự động bắt lỗi cho các yêu cầu HTTP không thành công (như 404 hoặc 500), giúp bạn dễ dàng xử lý lỗi.

Hỗ trợ cấu hình: Bạn có thể cấu hình các yêu cầu, như cài đặt headers, timeout, và nhiều tùy chọn khác.

Hỗ trợ hủy yêu cầu: Bạn có thể hủy một yêu cầu đang được thực hiện.

Cài đặt và sử dụng Axios

Để sử dụng Axios trong dự án của bạn, bạn cần cài đặt nó qua npm.

npm install axios

Dưới đây là một số ví dụ về cách sử dụng Axios để thực hiện các yêu cầu HTTP.

Ví dụ 1: Gửi yêu cầu GET

import axios from 'axios';

axios.get('https://jsonplaceholder.typicode.com/users')

.then(response => {

console.log(response.data); // In ra dữ liệu trả về từ server

})

.catch(error => {

console.error('Error fetching data:', error); // Xử lý lỗi

});

Ví dụ 2: Gửi yêu cầu POST

import axios from 'axios';

const newUser = {

name: 'John Doe',

email: 'john.doe@example.com'

};

axios.post('https://jsonplaceholder.typicode.com/users', newUser)

.then(response => {

console.log('User created:', response.data);

})

.catch(error => {

console.error('Error creating user:', error);

});

## Giới thiệu về Laravel

### Tổng quan về Laravel

**Laravel** là một framework PHP mã nguồn mở được thiết kế để phát triển các ứng dụng web. Được ra mắt lần đầu vào năm 2011 bởi Taylor Otwell, Laravel đã nhanh chóng trở thành một trong những framework PHP phổ biến nhất nhờ vào tính năng mạnh mẽ, dễ sử dụng và cấu trúc rõ ràng.

### Ưu điểm của Laravel

**MVC Architecture**: Laravel sử dụng mô hình MVC (Model-View-Controller), giúp tách biệt logic ứng dụng, giao diện người dùng và dữ liệu, từ đó làm cho mã nguồn dễ bảo trì hơn.

**Routing**: Laravel cung cấp một hệ thống routing đơn giản và mạnh mẽ, cho phép bạn dễ dàng xác định các URL và các hành động tương ứng mà ứng dụng sẽ thực hiện.

**Eloquent ORM**: Laravel đi kèm với Eloquent, một hệ thống Object-Relational Mapping (ORM) mạnh mẽ, giúp tương tác với cơ sở dữ liệu dễ dàng hơn thông qua các mô hình PHP.

**Blade Templating**: Laravel sử dụng Blade, một engine template mạnh mẽ giúp bạn dễ dàng xây dựng giao diện người dùng với các cấu trúc điều kiện và vòng lặp.

**Migration và Seeding**: Laravel hỗ trợ quản lý cơ sở dữ liệu thông qua các migration, cho phép bạn dễ dàng thay đổi cấu trúc cơ sở dữ liệu mà không mất dữ liệu. Seeding cho phép bạn dễ dàng tạo dữ liệu mẫu cho cơ sở dữ liệu.

**Middleware**: Middleware là các lớp lọc được sử dụng để kiểm tra các yêu cầu HTTP trước khi chúng đến controller. Điều này rất hữu ích để thực hiện xác thực và quản lý quyền truy cập.

**Security**: Laravel có nhiều tính năng bảo mật tích hợp, bao gồm mã hóa mật khẩu, bảo vệ CSRF (Cross-Site Request Forgery) và XSS (Cross-Site Scripting).

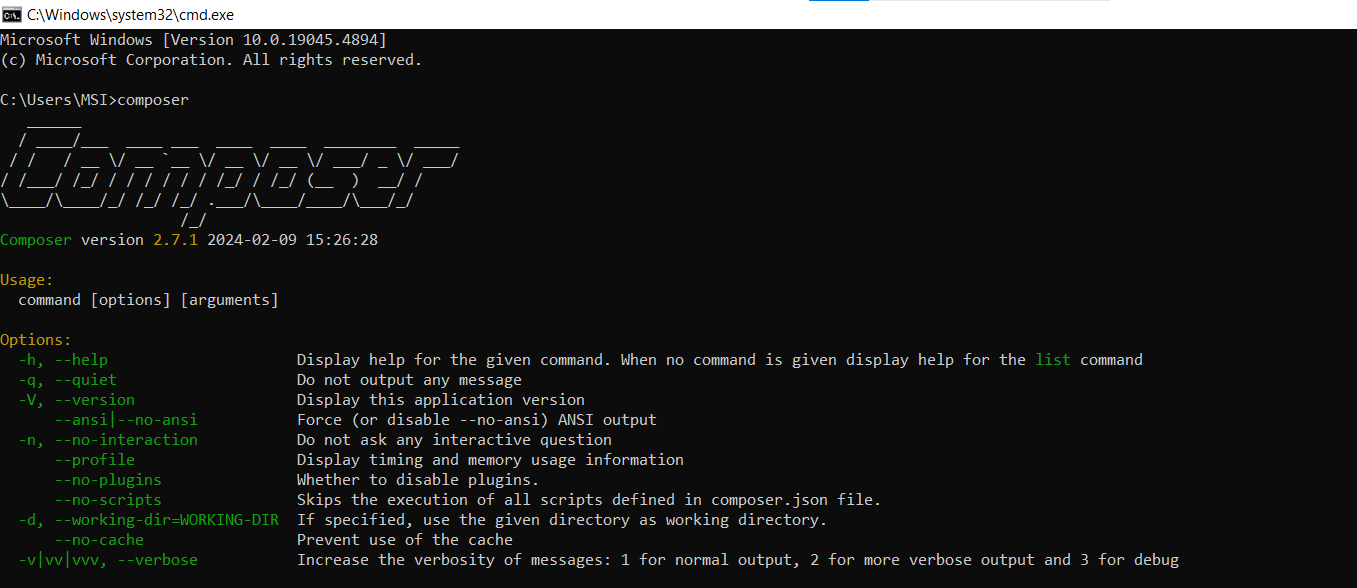
### Cách cài đặt Laravel

**Cài đặt laravel thông qua Composer**

Bước 1: Tải composer

Bước 2: Click vào file vừa tải và chọn Next liên tiếp cuối cùng chọn Install để tiến hành cài đặt. Khi cài đặt xong chọn Finish để kết thúc.

Bước 3: Mở cmd của windows lên gõ composer. Nếu cài đặt thành công sẽ hiển thị kết quả như hình dưới đây.



Hình 2.18 Kết quả cài đặt composer thành công

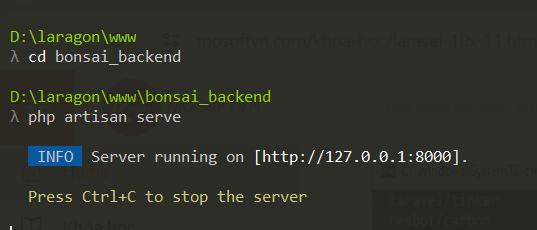
**Cài đặt Laravel**

Tạo 1 thư mục chứa dự án sau đó bật cmd và gõ câu lệnh sau để tạo dự án laravel

composer create-project laravel/laravel:^10.0 example-app

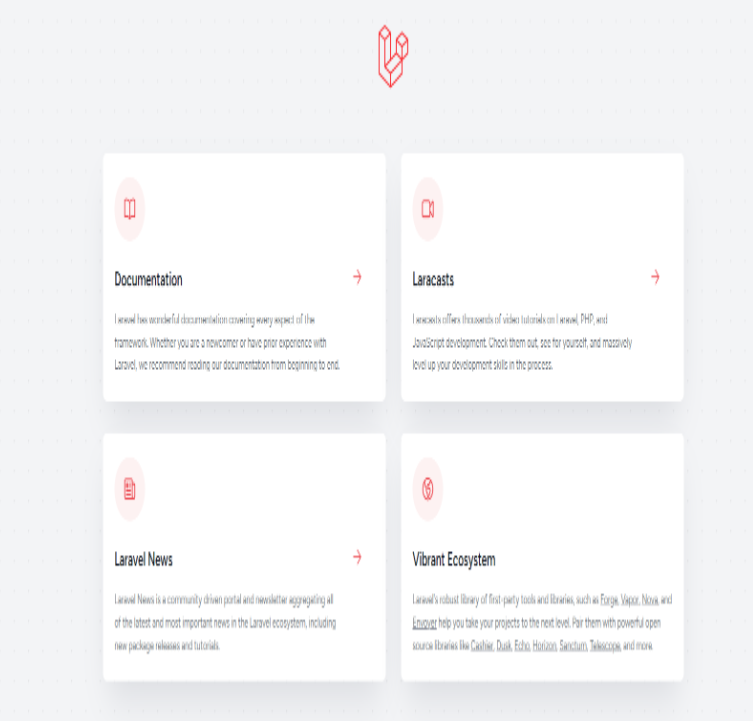
Sau khi cài đặt thành công, tiến hành chạy dự án bằng lệnh:

php artisan serve



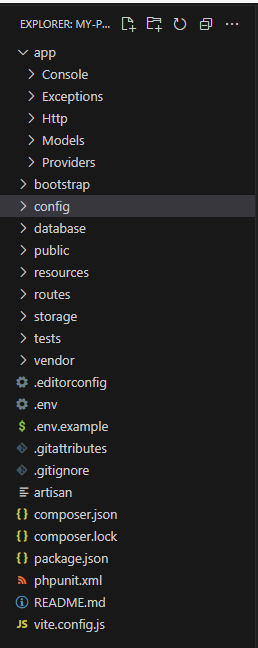
Hình 2.19 Kết quả chạy serve thành công

Sau đó vào trình duyệt gõ localhost:8000 ta sẽ nhận được kết quả như hình bên dưới.



Hình 2.20 Kết quả hiển thị giao diện Laravel

### Cấu trúc và thư mục trong Laravel



Mô tả về các thư mục chính trong một dự án Laravel:

**1. app/**

* Chứa toàn bộ mã nguồn của ứng dụng, đây là thư mục quan trọng nhất.
* Sub-thư mục chính:
  + Console/: Chứa các lệnh Artisan do người dùng định nghĩa.
  + Exceptions/: Chứa các file xử lý ngoại lệ của ứng dụng.
  + Http/: Chứa các controller, middleware và request. Đây là nơi điều phối các request từ người dùng.
    - Controllers/: Chứa các controller quản lý logic xử lý request.
    - Middleware/: Chứa các middleware cho request trước khi đến controller.
  + Models/: Chứa các model của ứng dụng.
  + Providers/: Chứa các service providers dùng để đăng ký dịch vụ trong Laravel.

**2. bootstrap/**

Chứa các file dùng để khởi động ứng dụng và các cấu hình cho Laravel. File app.php trong thư mục này sẽ khởi động Laravel framework.

**3. config/**

Chứa các file cấu hình của ứng dụng (ví dụ: database.php, app.php, mail.php,...). Bạn có thể tuỳ chỉnh các cấu hình của ứng dụng tại đây.

**4. database/**

Chứa các file liên quan đến cơ sở dữ liệu, bao gồm:

* + migrations/: Chứa các file migration để tạo và quản lý cấu trúc bảng trong cơ sở dữ liệu.
  + factories/: Dùng để định nghĩa các model factory, tạo dữ liệu giả cho ứng dụng.
  + seeders/: Chứa các class seeder để nhập dữ liệu mẫu vào cơ sở dữ liệu.

**5. public/**

Là thư mục root của ứng dụng web, chứa các file có thể truy cập từ bên ngoài, như file JavaScript, CSS, hình ảnh, và file index.php khởi động ứng dụng.

**6. resources/**

Chứa các file liên quan đến giao diện của ứng dụng.

* + views/: Chứa các view (file Blade) của ứng dụng.
  + lang/: Chứa các file ngôn ngữ phục vụ việc đa ngôn ngữ cho ứng dụng.
  + js/, css/, sass/: Chứa các file frontend như JavaScript, CSS, hoặc SASS.

**7. routes/**

Chứa các file định nghĩa route của ứng dụng.

* + web.php: Định nghĩa route cho ứng dụng web (HTTP routes).
  + api.php: Định nghĩa các route cho API (không có session).
  + console.php: Định nghĩa các route cho các lệnh Artisan.
  + channels.php: Định nghĩa các kênh broadcast cho ứng dụng.

**8. storage/**

Chứa các file liên quan đến session, cache, logs, và file tải lên.

* + app/: Dành cho các file người dùng tải lên.
  + framework/: Chứa các file do framework tạo ra (cache, session, views compiled).
  + logs/: Chứa các file log của ứng dụng.

**9. tests/**

Chứa các file liên quan đến việc kiểm thử (test) ứng dụng. Laravel cung cấp PHPUnit để viết các unit test.

**10. vendor/**

Chứa các thư viện của bên thứ ba và các gói cài đặt thông qua Composer.

**11. artisan**

Đây là file thực thi chính của Artisan, Laravel's command-line interface (CLI). Có thể chạy các lệnh Artisan từ đây.

**12. composer.json**

File chứa các thông tin về gói Composer và các phụ thuộc của dự án.

### Mô hình MVC trong Laravel

Mô hình MVC (Model-View-Controller) là một kiến trúc phần mềm phổ biến được sử dụng trong Laravel. Nó tách rời các thành phần của ứng dụng thành ba phần chính, giúp quản lý và mở rộng ứng dụng dễ dàng hơn.

Controller: Đảm bảo việc gửi nhận request giữa các view và model, các controller nằm trong thư mục app/Http/Controllers​

View: Đảm nhận việc hiển thị dữ liệu gửi xuống từ controller, các file nằm trong resources/views

Model: Đảm nhận việc máp, ánh xạ các bảng trong CSDL và các thao tác với CSDL, Các models được đặt trong thư mục app/Models​

**Luồng hoạt động trong MVC Laravel:**

1. Người dùng gửi một yêu cầu (request) từ giao diện web (như truy cập một trang hoặc gửi biểu mẫu).
2. Yêu cầu được gửi đến router của Laravel, xác định controller phù hợp để xử lý yêu cầu.
3. Controller nhận request, gọi các phương thức tương ứng trong model để thao tác với cơ sở dữ liệu nếu cần.
4. Sau khi xử lý xong, controller gửi dữ liệu tới view để hiển thị.
5. View nhận dữ liệu từ controller và trả về kết quả hiển thị cho người dùng.

**Ưu điểm của mô hình MVC trong Laravel:**

* **Phân tách rõ ràng:** Mỗi thành phần (model, view, controller) có trách nhiệm riêng, giúp mã nguồn dễ quản lý, bảo trì, và mở rộng.
* **Tái sử dụng mã:** Tái sử dụng code trở nên dễ dàng hơn khi tách biệt rõ giữa logic xử lý và giao diện.
* **Quy mô lớn:** MVC phù hợp cho các dự án lớn khi có nhiều thành viên làm việc trên các phần khác nhau của ứng dụng.

### Các thành phần của Laravel

#### Routing

**Routing** là cách hệ thống xác định tuyến đường (URL) mà người dùng truy cập và chỉ định các phương thức hoặc controller để xử lý các yêu cầu đó.

Các route trong Laravel được định nghĩa trong các file thuộc thư mục routes/:

* web.php: Được dùng cho các route liên quan đến giao diện người dùng (web).
* api.php: Được dùng cho các route liên quan đến API (thường trả về dữ liệu JSON). Nếu khai báo trong api.php thì đường dẫn phải thêm tiền tố api ở phía trước.

Các phương thức HTTP request

Route::get($uri, $callback);

Route::post($uri, $callback);

Route::put($uri, $callback);

Route::patch($uri, $callback);

Route::delete($uri, $callback);

Trong đó:

* $uri là path mà bạn muốn route xử lí.
* $callback là một callback, callable hoặc một array chứa thông tin controller và phương thức được gọi đến.

Ví dụ dưới đây là cách viết route gọi đến function index trong Controller

Route::get(‘/’,[Controller::class,’index’])

**Route Parameters:** là route có truyền một tham số

Ví dụ:

Route::get('/user/{id}', function (string $id) {

return 'User '.$id;

});

#### Middleware

Middleware là một cơ chế được sử dụng để lọc các yêu cầu HTTP vào ứng dụng. Nó hoạt động như một lớp trung gian giữa yêu cầu đến và phản hồi trả về. Middleware có thể thực hiện các hành động như xác thực, kiểm tra quyền truy cập, lọc dữ liệu hoặc thay đổi yêu cầu trước khi nó được chuyển đến controller.

Một số tính năng của Middleware:

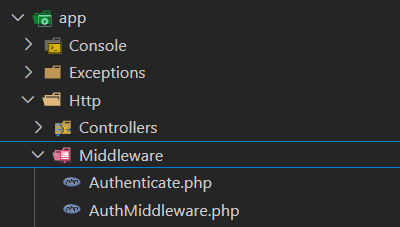
* Xác thực người dùng: Middleware có thể kiểm tra xem người dùng đã đăng nhập hay chưa trước khi cho phép truy cập vào một tuyến đường (route).
* Xử lý quyền hạn: Middleware có thể kiểm tra quyền hạn của người dùng để xác định xem họ có quyền truy cập vào một tài nguyên cụ thể hay không.
* Thay đổi yêu cầu hoặc phản hồi: Middleware có thể sửa đổi yêu cầu trước khi gửi tới controller hoặc sửa đổi phản hồi trước khi trả về cho client.
* Bảo vệ khỏi các cuộc tấn công bảo mật: Middleware cũng có thể được sử dụng để bảo vệ ứng dụng khỏi các cuộc tấn công như CSRF hoặc XSS.

**Tạo Middleware**

Để tạo Middleware dùng lệnh:

php artisan make:middleware AuthMiddleware

Sau khi tạo, middleware mới sẽ được lưu trong thư mục app/Http/Middleware.



Hình 2.21 Kết quả tạo middleware

Để sử dụng middleware, bạn cần khai báo nó trong app/Http/Kernel.php

Ví dụ:

 protected $middlewareAliases = [

        'auth' => \App\Http\Middleware\AuthMiddleware::class,

Sau khi đăng ký middleware, có thể áp dụng nó vào các route trong file routes/web.php

Route::get('profile', function () {

// code

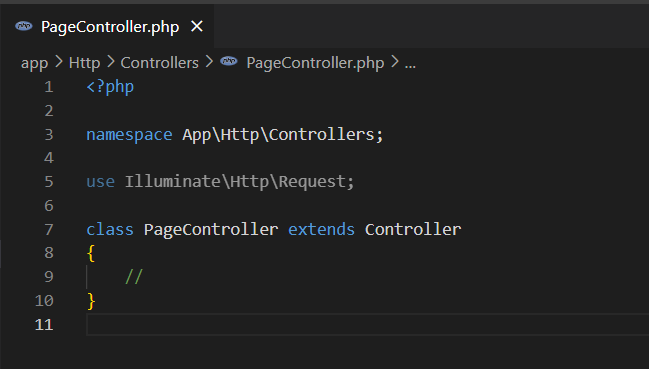
})->middleware('checkAge');

#### Controllers

Để tạo Controller tạo nhập lệnh:

php artisan make:controller PageController

Trong đó PageController là tên Controller có thể thay đổi tên cho phù hợp.



Hình 2.22 Kết quả tạo PageController

Ví dụ: viết hàm index cho PageController để hiển thị ra chữ “Hello World”

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PageController extends Controller

{

    function index() {

        return "Hello World";

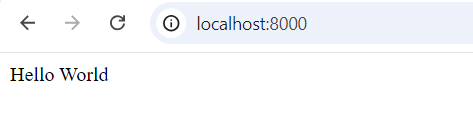
    }

}

Sau đó vào web.php viết đường dẫn với phương thức GET gọi đến index trong PageController.

Route::get("/",[PageController::class,"index"]);

Sau đó chạy serve bằng lệnh: **php artisan serve**



Hình 2.23 Kết quả chạy logic trong controller thông qua route

#### Requests

**Request** là một phần quan trọng trong việc xử lý các yêu cầu HTTP. Nó đại diện cho thông tin mà client (trình duyệt, ứng dụng di động, v.v.) gửi đến server. Request chứa dữ liệu như thông tin form, file upload, cookies, headers, và các tham số của URL.

Laravel cung cấp lớp Illuminate\Http\Request, dễ dàng truy xuất các thông tin trong một yêu cầu HTTP. Có thể lấy dữ liệu từ request trong controller, middleware hoặc bất kỳ nơi nào trong ứng dụng.

Ví dụ: Sử dụng đối tượng Request để lấy dữ liệu từ form.

use Illuminate\Http\Request;

class UserController extends Controller

{

public function store(Request $request)

{

// Lấy tất cả dữ liệu từ form

$data = $request->all();

// Lấy giá trị cụ thể từ request

$name = $request->input('name');

$email = $request->input('email');

// Lấy dữ liệu từ query string (URL)

$page = $request->query('page');

}

}

#### Views

Chịu trách nhiệm hiển thị giao diện người dùng. View thường chứa mã HTML kèm theo cú pháp của Blade để hiển thị dữ liệu từ controller. Tất cả các file view được lưu trữ trong thư mục resources/views/. Có thể tạo các file Blade với đuôi .blade.php

Ví dụ: Tạo một file với tên [home.blade.php](http://home.blade.php)

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Home</title>

</head>

<body>

    <h1>Welcome to the {{$name}}</h1>

</body>

</html>

Sau đó sẽ hiển thị View trong Controller

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PageController extends Controller

{

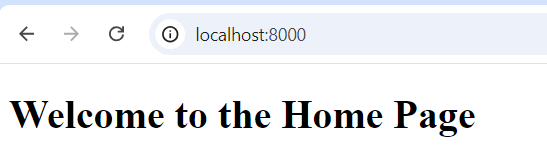
    function index() {

        $name = "Home Page";

        return view('home')->with('name', $name);

    }

}



Hình 2.24 Kết quả hiển thị từ view

#### Blade Templates

Blade Templates là công cụ template engine mạnh mẽ và linh hoạt của Laravel, cho phép xây dựng giao diện người dùng một cách dễ dàng. Blade cung cấp cú pháp đơn giản để nhúng logic vào trong các file HTML, đồng thời hỗ trợ việc kế thừa template và tái sử dụng các thành phần UI.

Blade template có phần mở rộng file là .blade.php. Laravel sẽ tự động biên dịch các file này sang PHP và hiển thị nội dung của chúng khi bạn truy cập trang web.

Ví dụ: tạo một file Blade là [home.blade.php](http://home.blade.php) trong thư mục resources/views

<!-- resources/views/home.blade.php -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Welcome</title>

</head>

<body>

<h1>Hello, {{ $name }}</h1>

</body>

</html>

**Cú pháp Blade cơ bản**

**Echo dữ liệu**: Để hiển thị dữ liệu, sử dụng cú pháp {{ ... }}. Blade sẽ tự động escape dữ liệu để tránh lỗi XSS (Cross-Site Scripting).

Ví dụ: {{$name}}

**Câu điều kiện**: Sử dụng các câu lệnh điều kiện như @if, @elseif, @else, và @endif.

Ví dụ:

@if($age >= 18)

<p>You are an adult.</p>

@else

<p>You are not an adult yet.</p>

@endif

**Vòng lặp**: Có thể sử dụng các câu lệnh vòng lặp như @for, @foreach, @while.

Ví dụ:

@foreach($users as $user)

<p>{{ $user->name }}</p>

@endforeach

**Kế thừa Template**: Blade hỗ trợ kế thừa và chia sẻ các thành phần UI giữa các view khác nhau thông qua hai từ khóa chính là @extends và @section.

**Tạo layout (master template)**

Đầu tiên, có thể tạo một layout chính mà các trang con sẽ kế thừa. Ví dụ, tạo một file layout.blade.php:

<!-- resources/views/layout.blade.php -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@yield('title')</title>

</head>

<body>

<header>

<h1>My Website</h1>

</header>

<div class="container">

@yield('content')

</div>

<footer>

<p>© 2024 My Website</p>

</footer>

</body>

</html>

@yield('title'): Là một placeholder, nơi trang con có thể chèn nội dung.

@yield('content'): Là nơi trang con chèn nội dung chính của trang.

**Sử dụng layout trong các trang con**

Tiếp theo, tạo một trang con và kế thừa từ layout đã tạo bằng cách sử dụng @extends và @section.

<!-- resources/views/home.blade.php -->

@extends('layout')

@section('title', 'Home Page')

@section('content')

<h2>Welcome to the Home Page</h2>

<p>This is the content of the homepage.</p>

@endsection

Ở đây, @section('title', 'Home Page') định nghĩa tiêu đề của trang và @section('content') định nghĩa nội dung chính của trang.

**Truyền biến vào view:**Từ controller, có thể truyền biến vào view bằng cách sử dụng hàm view().

public function index() {

$name = "John Doe";

return view('home', ['name' => $name]);

}

Trong Blade, sẽ truy xuất biến $name:

<p>Hello, {{ $name }}</p>

#### Eloquent ORM

Model được xây dựng dựa trên Eloquent ORM (Object-Relational Mapping), đây là hệ thống ORM của Laravel giúp tương tác với cơ sở dữ liệu một cách dễ dàng và trực quan bằng cách sử dụng các đối tượng thay vì các truy vấn SQL thuần túy.

Để tạo model dùng lệnh:

php artisan make:model Category

Sau đó kết nối với bảng trong cơ sở dữ liệu

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Category extends Model

{

    use HasFactory;

    protected $table = 'categories';

    public function products()

    {

        return $this->hasMany(Product::class);

    }

}

**Eloquent Relationship(mối quan hệ)**

Eloquent hỗ trợ các loại quan hệ giữa các bảng thông qua các phương thức đơn giản như:

**One-to-One(kết nối 1-1)**

Ví dụ, một người dùng có một hồ sơ cá nhân.

php

public function profile()

{

return $this->hasOne(Profile::class);

}

**One-to-Many(kết nối 1-nhiều)**

Ví dụ, một danh mục có nhiều sản phẩm.

php

public function products()

{

return $this->hasMany(Product::class);

}

**Many-to-Many(kết nối nhiều-nhiều)**

Ví dụ, một sinh viên có thể học nhiều khóa học và một khóa học có nhiều sinh viên.

php

public function courses()

{

return $this->belongsToMany(Course::class);

}

#### Database Migrations

Migration quản lý phiên bản cơ sở dữ liệu giống như hệ thống quản lý phiên bản mã nguồn. Nó cho phép bạn xây dựng, thay đổi cấu trúc bảng, cột, chỉ mục của cơ sở dữ liệu một cách dễ dàng và có thể theo dõi được.

Để tạo bảng dùng lệnh:

php artisan make:migration create\_categories\_table

Sau khi tạo xong thì tạo các thuộc tính cho bảng.

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration

{

    /\*\*

     \* Run the migrations.

     \*/

    public function up(): void

    {

        Schema::create('categories', function (Blueprint $table) {

            $table->increments('id');

            $table->string('name');

            $table->timestamps();

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*/

    public function down(): void

    {

        Schema::dropIfExists('categories');

    }

};

Để chạy bảng dùng lệnh

php artisan migrate

Để kiểm tra trạng thái của bảng dùng lệnh:

php artisan migrate:status

Trong trường hợp muốn thực thi rollback migrate data về phiên bản trước đó có thể dùng lệnh:

php artisan migrate:rollback

Để cập bảng sử dụng cú pháp tương tự như tạo bảng và chỉ thêm thuộc tính cần tạo thêm

Ví dụ thêm thuộc tính status cho bảng Category

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

Schema::table('categories', function (Blueprint $table) {

$table->integer('status');

});

Để đổi tên bảng thực hiện như sau:

use Illuminate\Support\Facades\Schema;

Schema::rename($from, $to);

Trong đó:

* $from là tên bảng ban đầu
* $to là tên bảng mới

Để xóa bảng thực hiện như sau:

Schema::drop('table name');

Các phương thức kiểu dữ liệu thường dùng trong migration​:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tên phương thức**​ | **Giải thích**​ | **Tên phương thức**​ | **Giải thích**​ |
| **Id()**​ | Tạo cột id tự động tăng​ | **longText()**​ | Tạo cột kiểu longtext​ |
| **bigIncrements()**​ | Tạo cột kiểu bigint tự động tăng​ | **dateTime()**​ | Tạo cột kiểu dữ liệu datetime​ |
| **bigInteger()**​ | Tạo cột bigint​ | **date()**​ | Tạo cột kiểu date​ |
| **boolean)()**​ | Tạo cột kiểu bit​ | **softDeletes()**​ | Tạo cột kiểu date​ |
| **char()**​ | Tạo cột kiểu char​ | **time()**​ | Tạo cột kiểu time​ |
| **tinyInteger()**​ | Tạo cột tinyint​ | **timestamp()**​ | Tạo cột kiểu timestamp​ |
| **integer()**​ | Tạo cột kiểu int​ | **timestamps()**​ | Tạo 2 cột created\_at, updated\_at​ |
| **foreignId()**​ | Tạo cột int unsigned (không âm)​ | **text()**​ | Tạo cột kiểu text​ |
| **increments()**​ | Tạo cột id tự động tang, không âm​ | **mediumText()**​ | Tạo cột kiểu medium text​ |
| **float()**​ | Kiểu float​ | **string()**​ | Tạo cột kiểu varchar​ |

#### Database Seeding

Seeders là các lớp dùng để tạo và nạp dữ liệu mẫu (fake data) hoặc dữ liệu mặc định vào cơ sở dữ liệu. Seeders được sử dụng trong quá trình phát triển và kiểm thử để có dữ liệu ngay lập tức mà không cần phải nhập liệu thủ công.

Để tạo Seeder dùng lệnh:

php artisan make:seeder UserSeeder

Ví dụ:

<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

use Illuminate\Support\Facades\DB;

use Illuminate\Support\Facades\Hash;

use Faker\Factory as Faker;

class UserSeeder extends Seeder

{

public function run()

{

$faker = Faker::create();

for ($i = 0; $i < 10; $i++) {

DB::table('users')->insert([

'name' => $faker->name,

'email' => $faker->unique()->safeEmail,

'password' => Hash::make('password'),

'address' => $faker->address,

'phone' => $faker->phoneNumber,

'created\_at' => now(),

'updated\_at' => now(),

]);

}

}

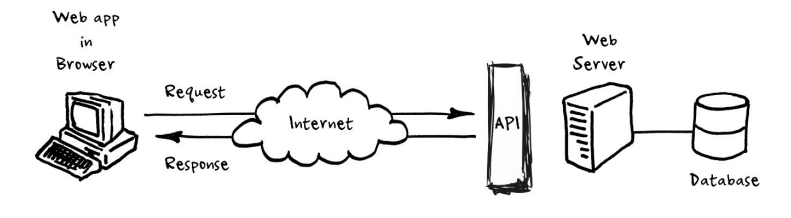
}

Để thực thi dùng lệnh :

php artisan db:seed --class=UserSeeder

## Giới thiệu về Postman

### API là gì? Vì sao phải test API?



API (Application Programming Interface) là một tập hợp các quy tắc và giao thức cho phép các phần mềm giao tiếp với nhau. Nó đóng vai trò như một cầu nối giữa các ứng dụng, cho phép chúng trao đổi dữ liệu và thực hiện các chức năng mà không cần biết chi tiết cách thức hoạt động bên trong của nhau.

**Vì sao phải test API?**

Trong quá trình triển khai dự án, phần server và client làm độc lập với nhau nên có nhiều chỗ client chưa làm xong, không thể chờ client làm xong để test được dữ liệu mà test API bằng công cụ khác luôn –> Lúc này việc test hoàn toàn không phụ thuộc gì vào client.

Kể cả khi client làm xong rồi, nếu mình test trên client mà thấy lỗi liên quan đến logic và dữ liệu thì cũng cần test thêm cả API để biết chính xác là server sai hay client sai –> fix lỗi sẽ nhanh hơn.

### Postman là gì?

Postman là một công cụ phổ biến dùng để phát triển và kiểm thử API. Ban đầu, nó được phát triển như một tiện ích mở rộng của Google Chrome, nhưng hiện tại đã trở thành một ứng dụng độc lập với nhiều tính năng mạnh mẽ, hỗ trợ lập trình viên và tester dễ dàng gửi yêu cầu HTTP và nhận phản hồi từ API trong quá trình phát triển.

### Ưu, nhược điểm của Postman

**Ưu điểm của Postman**

Dễ sử dụng: Giao diện người dùng thân thiện và dễ hiểu giúp người dùng mới có thể làm quen nhanh chóng.

Tính năng phong phú: Postman hỗ trợ nhiều tính năng như gửi yêu cầu HTTP, kiểm thử tự động, mock server, và tạo tài liệu API tự động.

Quản lý Collection: Cho phép tổ chức các yêu cầu thành collection, dễ dàng chia sẻ và tái sử dụng.

Hỗ trợ kiểm thử tự động: Cho phép viết script kiểm thử để tự động kiểm tra các phản hồi từ API, giúp phát hiện lỗi nhanh chóng.

**Nhược điểm của Postman**

Hiệu suất: Khi làm việc với các collection lớn hoặc khi số lượng yêu cầu nhiều, Postman có thể bị chậm hoặc không phản hồi tốt.

Phiên bản miễn phí có hạn chế: Mặc dù Postman có phiên bản miễn phí, nhưng một số tính năng cao cấp (như tích hợp nâng cao và giám sát API) chỉ có trong phiên bản trả phí.

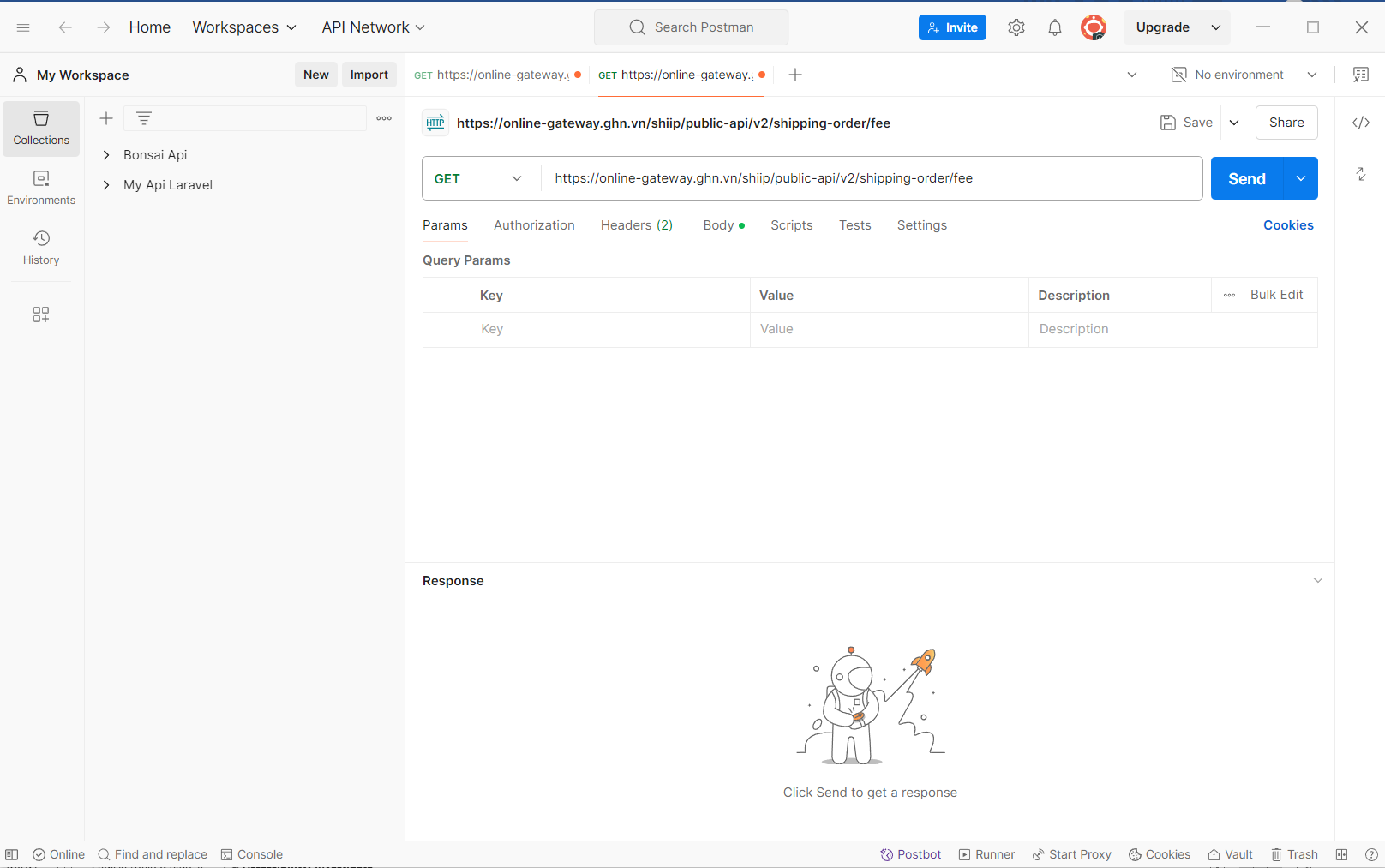
Yêu cầu kết nối internet: Một số tính năng của Postman, đặc biệt là hợp tác nhóm và chia sẻ tài liệu, yêu cầu kết nối internet.

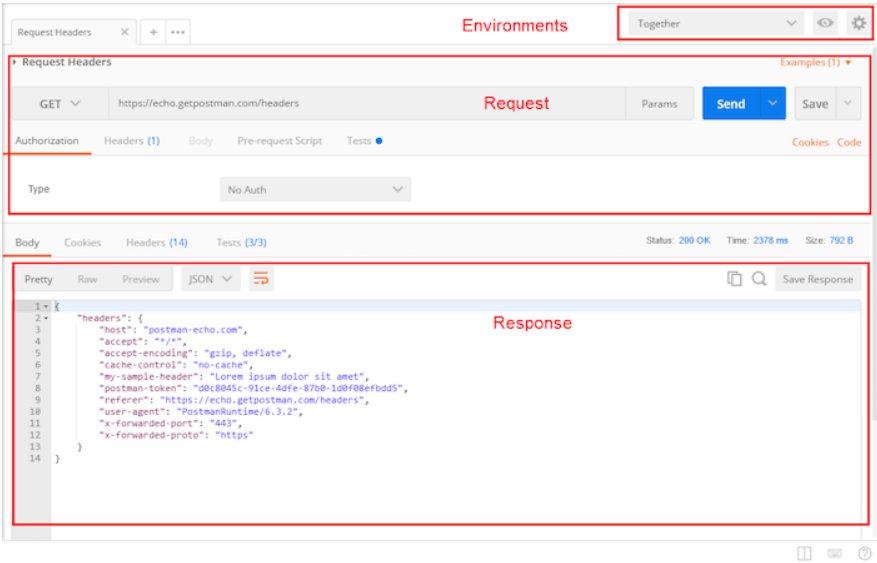
Quản lý biến phức tạp: Việc quản lý nhiều biến và môi trường có thể trở nên phức tạp, đặc biệt với những người dùng mới.

Khó khăn trong việc kiểm thử API phức tạp: Đối với các API phức tạp, việc thiết lập và viết kịch bản kiểm thử có thể trở nên khó khăn và yêu cầu kiến thức chuyên sâu.

### Các thành phần trong Postman

Giao diện của Postman





Trong phần này gồm có 3 thành phần chính:

Environments: Chứa các thông tin môi trường. Ví dụ: mình làm 1 dự án nhưng có 3 môi trường khác nhau: dev, staging và product. Có phần này, mình có thể nhanh chóng đổi sang môi trường cần test mà không phải mất công đổi URL của từng request. (Cái này sẽ được nói rõ hơn ở những bài sau)

Request: Phần chứa các thông tin chính của API.

Response: Chứa các thông tin trả về sau khi Send Request.

### Cách tạo Request

Khi làm việc với API, chúng ta chỉ làm việc với 2 dạng API chính là GET và POST.

GET: Yêu cầu server đưa lại resource: Hãy tưởng tượng ra cái cảnh vào fb, tay vuốt new

feeds.

POST: Yêu cầu server cho tạo ra 1 resource mới. Ví dụ: đăng ký 1 chuyến đi ở GrabBike.

Và một request gồm có 4 thành phần:

1. URL

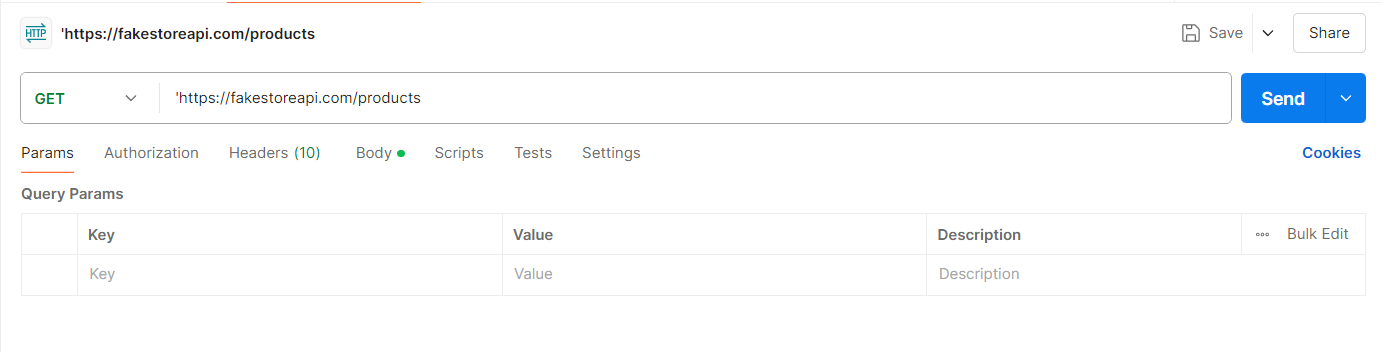
2. Method

3. Headers

4. Body

#### Tạo request GET

Đây là trang tôi lấy dữ liệu API mẫu: https://fakestoreapi.com/docs



Hình 2.25 Lấy địa chỉ API dùng phương thức GET

URL: https://fakestoreapi.com/products

2. Method: GET

3. Headers: Không cần điền gì cả

4. Body: Phương thức GET không có body,

Khi nhấn nút send sẽ thông báo 200 OK nghĩa là lấy dữ liệu thành công



Hình 2.26 Kết quả lấy dữ liệu thành công

Thông tin trả về sẽ có mấy điểm cần quan tâm:

1. Định dạng dữ liệu trả về: thông thường là json và nên để chế độ Pretty để cho dễ nhìn.

2. Nội dung dữ liệu: Đây là phần cần phải kiểm tra.

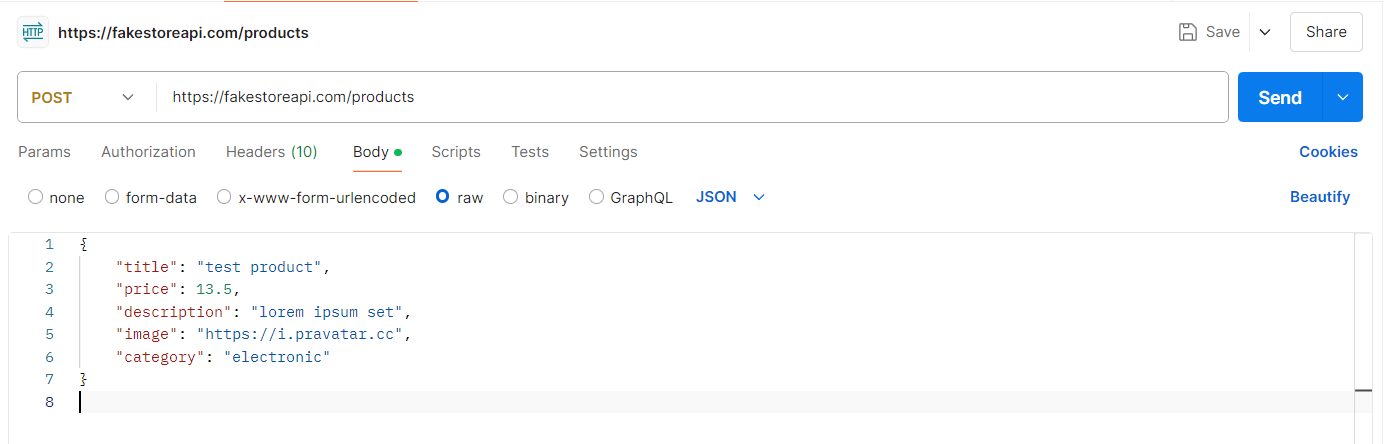
– So sánh với cái Sample Response ở API docs để xem *c*ấ*u trúc* trả về đã đúng hay chưa.

– Value của từng key đã đúng chưa, so sánh với nội dung trong DB

3. Trạng thái của API (status) và thời gian trả về.

#### Tạo request POST

Tương tự như GET nhưng cần phải truyền key, value cho phần Body chọn raw và chuyển đổi GET thành POST.



Hình 2.27 Tạo request POST

# HIỆN THỰC HÓA NGHIÊN CỨU

## Mô tả bài toán

## Đặc tả các yêu cầu chức năng

### Yêu cầu chức năng

### Yêu cầu phi chức năng

## Thiết kế dữ liệu

### Lược đồ cơ sở dữ liệu

#### Mô hình thực thể kết hợp

#### Mô hình vật lý

### Danh sách các thực thể

### Chi tiết các thực thể

## Thiết kế giao diện

### Sơ đồ website

### Giao diện website

# KẾT QUẢ NGHIÊN CỨU

## Dữ liệu thử nghiệm

## Kết quả thử nghiệm

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

## Hướng phát triển

# DANH MỤC TÀI LIỆU THAM KHẢO

# PHỤ LỤC

**Hướng dẫn cài đặt laragon**