

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO THỰC TẬP TỐT NGHIỆP

Tên đề tài:

THIẾT KẾ VÀ XÂY DỰNG WEBSITE CỬA HÀNG THỜI TRANG

Giảng viên hướng dẫn: ThS. Vũ Đình Long

Sinh viên thực hiện:

19H1120035: Đào Văn Thương

19H1120054: Nguyễn Minh Nhật

Thành phố Hồ Chí Minh, năm 2022

Lời cảm ơn

Lời đầu tiên em xin chân thành cảm ơn các thầy, cô trong khoa Công nghệ thông tin, trường Đại học Giao thông vận tải TP.HCM đã tạo điều kiện thuận lợi cho chúng em trong quá trình học tập tại trường cũng như trong thời gian thực hiện đồ án Thực tập tốt nghiệp. Đặc biệt em muốn gửi lời cảm ơn đến Thạc sĩ Vũ Đình Long – giảng viên hướng dẫn trực tiếp trong quá trình làm đồ án của chúng em.

Chúng em đã cố gắng với tất cả nỗ lực của bản thân để hoàn thành dự án, nhưng do thời gian có hạn và kinh nghiệm bản thân còn hạn chế nên đồ án không tránh khỏi những thiếu sót. Chúng em kính mong nhận được sự góp ý từ quý thầy cô, bạn bè để có thể nâng cao kiến thức của bản thân và hoàn thiện đồ án được tốt hơn.

Chúng em xin chân thành cảm ơn!

Lời cam đoan

Nhóm chúng em xin cam đoan đề tài: “Thiết kế và xây dựng website cửa hàng thời trang” là do bản thân nhóm chúng em tự nghiên cứu thực hiện với sự hỗ trợ của giảng viên hướng dẫn và tham khảo từ các tài liệu, giáo trình liên quan đến đề tài và không có sự sao chép từ các nguồn khác. Nhóm chúng em xin hoàn toàn chịu trách nhiệm về tính trung thực của đề tài này.

Thứ tư, ngày 23 tháng 11 năm 2022

Sinh viên thực hiện

Đào Văn Thương

Nguyễn Minh Nhật

MỤC LỤC

CHƯƠNG 1. TỔNG QUAN VỀ ĐỀ TÀI.....	1
1.1. Lí do chọn đề tài	1
1.2. Mục đích - nhiệm vụ nghiên cứu.....	1
1.3. Phương pháp nghiên cứu.....	1
1.4. Công nghệ sử dụng.....	2
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT	3
2.1. Giới thiệu ReactJS.....	3
2.1.1. <i>Giới thiệu về JavaScript(JS).....</i>	<i>3</i>
2.1.2. <i>Khái niệm SPA (Single Page Application).....</i>	<i>3</i>
2.1.3. <i>Cấu trúc của một SPA và nguyên lý hoạt động.....</i>	<i>4</i>
2.1.4. <i>Ưu điểm của SPA.....</i>	<i>5</i>
2.1.5. <i>Nhược điểm của SPA.....</i>	<i>5</i>
2.1.6. <i>ReactJS là gì?.....</i>	<i>6</i>
2.1.7. <i>Những khái niệm cơ bản của ReactJS.....</i>	<i>6</i>
2.1.8. <i>Ưu điểm và nhược điểm của ReactJS.....</i>	<i>11</i>
2.2. Giới thiệu về TypeScript(TS)	13
2.2.1. <i>TypeScript là gì?.....</i>	<i>13</i>
2.2.2. <i>Tại sao nên sử dụng TypeScript?</i>	<i>14</i>
2.2.3. <i>Cơ bản về TypeScript</i>	<i>14</i>
2.2.4. <i>Ưu điểm của TypeScript</i>	<i>18</i>
2.3. Giới thiệu Redux, Redux Toolkit và RTK query	19
2.3.1. <i>Khái niệm Redux.....</i>	<i>19</i>
2.3.2. <i>Lợi ích của Redux.....</i>	<i>19</i>
2.3.3. <i>Cấu trúc của Redux</i>	<i>20</i>
2.3.4. <i>Nguyên lý vận hành</i>	<i>21</i>
2.3.5. <i>Redux hoạt động như thế nào ?.....</i>	<i>22</i>

2.3.6.	<i>Redux Thunk là gì ?</i>	25
2.3.7.	<i>Redux Persist là gì ?</i>	25
2.3.8.	<i>Redux-toolkit (RTK)</i>	25
2.3.9.	<i>RTK bao gồm những gì ?</i>	25
2.3.10.	<i>Khái niệm về RTK query</i>	29
2.4.	Giới thiệu về SASS/SCSS và CSS Module	33
2.4.1.	<i>CSS Preprocessor là gì ?</i>	33
2.4.2.	<i>SASS là gì ?</i>	33
2.4.3.	<i>Ưu điểm của SASS</i>	34
2.4.4.	<i>Cách thức hoạt động của SASS là gì ?</i>	34
2.4.5.	<i>SCSS là gì ?</i>	34
2.4.8.	<i>CSS Module là gì ?</i>	41
2.4.9.	<i>Tại sao chúng ta nên sử dụng CSS Module ?</i>	44
2.5.	Giới thiệu NodeJS	44
2.5.1.	<i>Khái niệm NodeJS</i>	44
2.5.2.	<i>Các tính năng của NodeJS</i>	44
2.5.3.	<i>Ứng dụng của NodeJS</i>	45
2.5.4.	<i>Server Side Rendering(SSR) và Client Side Rendering(CSR)</i>	45
2.5.5.	<i>Cấu trúc NodeJS</i>	49
2.5.6.	<i>Node Package Manager</i>	52
2.5.7.	<i>Ưu điểm và nhược điểm của NodeJS</i>	53
2.6.	Giới thiệu ExpressJS	53
2.6.1.	<i>Khái niệm ExpressJS</i>	53
2.6.2.	<i>Cấu trúc ExpressJS</i>	54
2.6.3.	<i>Router trong ExpressJS</i>	54
2.6.4.	<i>Middleware trong ExpressJS</i>	56

2.6.5.	<i>JSON Web Token (JWT)</i>	57
2.7.	Giới thiệu MongoDB & mongoose	58
2.7.1.	<i>Khái niệm MongoDB</i>	58
2.7.2.	<i>Khái niệm mongoose</i>	58
2.7.3.	<i>So sánh Relational database management system và MongoDB</i>	59
2.7.4.	<i>Kiểu dữ liệu của MongoDB</i>	59
2.7.5.	<i>Cách thức hoạt động</i>	61
2.7.6.	<i>Ưu điểm và nhược điểm của MongoDB</i>	61
2.8.	Giới thiệu SocketIO	63
2.8.1.	<i>Khái niệm về SocketIO</i>	63
2.8.2.	<i>Cơ chế hoạt động của SocketIO</i>	63
2.8.3.	<i>Điểm nổi bật của SocketIO</i>	66
2.9.	API (Application Program Interface)	66
2.9.1.	<i>Khái niệm về API</i>	66
2.9.2.	<i>WebAPI</i>	67
2.9.3.	<i>RESTfulAPI</i>	67
2.10.	Giới thiệu Axios	69
2.10.1.	<i>Khái niệm về Axios</i>	69
2.10.2.	<i>Các tính năng của Axios</i>	70
2.10.3.	<i>Ưu điểm của Axios</i>	70
CHƯƠNG 3.	PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG	71
3.1.	Biểu đồ phân cấp chức năng	71
3.1.1.	<i>Biểu đồ phân cấp chức năng mua hàng</i>	71
3.1.2.	<i>Biểu đồ phân cấp chức năng bán hàng</i>	71
3.2.	Biểu đồ Use-Case	71
3.3.	Biểu đồ quan hệ thực thể ERD	73

CHƯƠNG 4. TRIỂN KHAI ỨNG DỤNG	75
4.1. Triển khai Frontend.....	75
4.1.1. <i>Giao diện trang chủ.....</i>	<i>75</i>
4.1.2. <i>Giao diện trang đăng nhập.....</i>	<i>76</i>
4.1.3. <i>Giao diện trang đăng ký tài khoản.....</i>	<i>76</i>
4.1.4. <i>Giao diện sản phẩm của từng đối tượng</i>	<i>78</i>
4.1.5. <i>Giao diện Layout category - navigation.....</i>	<i>87</i>
4.1.6. <i>Giao diện trang tất cả sản phẩm.....</i>	<i>87</i>
4.1.7. <i>Giao diện chi tiết sản phẩm.....</i>	<i>89</i>
4.1.8. <i>Giao diện trang thông tin cá nhân</i>	<i>92</i>
4.1.9. <i>Giao diện trang đơn hàng</i>	<i>93</i>
4.1.10. <i>Giao diện danh sách địa chỉ giao hàng.....</i>	<i>96</i>
4.1.11. <i>Giao diện đã xem gần đây.....</i>	<i>98</i>
4.1.12. <i>Giao diện giỏ hàng</i>	<i>98</i>
4.1.13. <i>Giao diện danh sách sản phẩm yêu thích.....</i>	<i>99</i>
4.1.14. <i>Giao diện tìm kiếm sản phẩm</i>	<i>100</i>
4.1.15. <i>Giao diện trang đặt hàng</i>	<i>101</i>
4.1.16. <i>Giao diện nhắn tin.....</i>	<i>102</i>
4.2. Triển khai Backend.....	104
4.2.1. <i>Xây dựng các models.....</i>	<i>104</i>
4.2.2. <i>Xây dựng các routes</i>	<i>113</i>
CHƯƠNG 5. TỔNG KẾT	130
5.1. Kết luận	130
5.2. Những khó khăn gặp phải	131
5.3. Hướng phát triển	131

CHƯƠNG 1. TỔNG QUAN VỀ ĐỀ TÀI

1.1. Lí do chọn đề tài

Như chúng ta cũng thấy trong thị trường hiện nay thì việc cạnh tranh về kinh doanh ngày càng trở nên quyết liệt và hầu hết những nhà kinh doanh, những công ty lớn đều rất chú tâm đến việc làm thoả mãn khách hàng một cách tốt nhất.

So với kinh doanh truyền thống thì Thương mại điện tử chi phí thấp hơn, hiệu quả đạt cao hơn. Hơn thế nữa, với lợi thế của công nghệ Internet nên việc truyền tải thông tin về sản phẩm nhanh chóng, thuận tiện. Kết hợp với bộ phận giao hàng tận nơi, là thông qua bưu điện và ngân hàng để thanh toán tiền, càng tăng thêm thuận lợi để loại hình này phát triển.

Do đó, với sự ra đời các website bán hàng qua mạng, mọi người có thể mua mọi thứ hàng hoá mọi lúc mọi nơi mà không cần phải tới tận nơi để mua. Trên thế giới có rất nhiều website bán hàng trực tuyến vẫn chưa được phổ biến rộng rãi. Và khái niệm thương mại điện tử còn khá xa lạ. Trước thực tế đó nhóm em đã chọn đề tài: *Xây dựng website bán hàng trực tuyến* để nghiên cứu và học tập cùng với lợi thế hiện tại là sau đợt dịch COVID, các cửa hàng – thương hiệu đánh mạnh vào việc mua sắm online vì vậy phát triển một website về ngành hàng thời trang hiện đã và đang rất có chỗ đứng trong thị trường Việt Nam nói chung và quốc tế nói riêng.

1.2. Mục đích - nhiệm vụ nghiên cứu

Đề tài được nghiên cứu dựa trên cơ sở lý thuyết các công nghệ cũng như việc tìm hiểu quá trình xây dựng một sản phẩm, một hệ thống công nghệ nhằm nâng cao trải nghiệm của dùng và việc quản lý của các chủ cửa hàng cũng trở nên nhanh và tiện lợi hơn rất nhiều.

Ngoài ra nhóm còn nghiên cứu về giao diện người dùng cũng như trải nghiệm người dùng để có thể tăng mức trải nghiệm của người dùng lên cao nhất có thể.

1.3. Phương pháp nghiên cứu

- Về mặt lý thuyết:

- Tìm hiểu kỹ thuật lập trình, cách thức hoạt động và các đối tượng trong một ứng dụng.
- Phân tích, thiết kế được một hệ thống website bán hàng.
- Hiểu được cách thức hoạt động của Client – Server

- Về mặt lập trình:

- Sử dụng draw.io để vẽ các biểu đồ cần thiết cho việc thiết kế hệ thống.

- Sử dụng các ngôn ngữ lập trình và IDE phù hợp để tiến hành lập trình Front-end và Back-end cũng như cơ sở dữ liệu.
- *Về mặt hoạt động:*
 - Phần mềm có thể thực hiện đủ các chức năng cơ bản của một website bán hàng online (vd: xem, xoá, sửa giỏ hàng, đăng nhập, đăng xuất, chat với cửa hàng, ...)

1.4. Công nghệ sử dụng

- Front-end: Sử dụng framework ReactJS, Redux Toolkit, RTK query và SocketIO.
 - Back-end: Sử dụng ExpressJS & NodeJS và SocketIO.
 - Database: Sử dụng MongoDB để quản trị cơ sở dữ liệu.
- ⇒ Cả Front-end và Back-end được viết bằng TypeScript.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Giới thiệu ReactJS

2.1.1. Giới thiệu về JavaScript(JS)

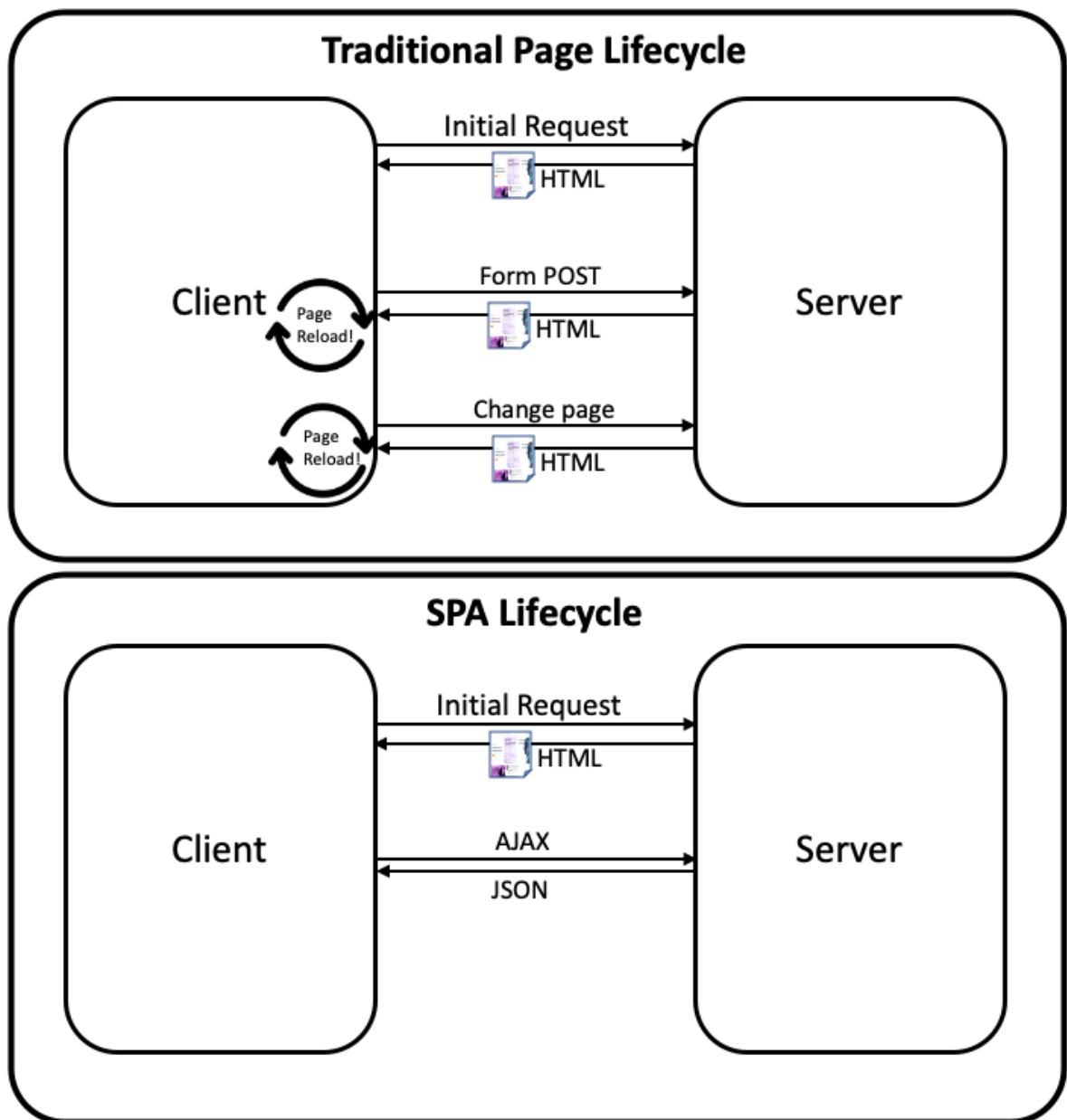
JavaScript chính là một ngôn ngữ lập trình web rất phổ biến ngày nay. JavaScript được tích hợp đồng thời nhúng vào HTML để hỗ trợ cho website trở nên sống động hơn. Chúng cũng đóng vai trò tương tự như một phần của website, cho phép Client-Side Script từ người dùng tương tự máy chủ (NodeJS) để tạo ra những website động. Đã có khá nhiều framework ra đời và được viết bằng loại ngôn ngữ này. Từ front-end cho đến back-end thì bất cứ nơi nào cũng có sự xuất hiện của JavaScript.



Hình 2.1.1. Javascript là ngôn ngữ lập trình được sử dụng khá phổ biến

2.1.2. Khái niệm SPA (Single Page Application)

Single Page Application (hay còn được biết đến với cái tên viết tắt **SPA**) là một ứng dụng web hay một website mà ở đó tất cả các thao tác của người dùng chỉ diễn ra trên một trang duy nhất, tất cả các cấu trúc của trang web (HTML) được tải sẵn một lần và sẽ không tải lại ngay cả khi chuyển trang.



Hình 2.1.2. Mô tả về Traditional Page Lifecycle và SPA Lifecycle

Một vài website nổi tiếng như: Gmail, Facebook, Youtube, Twitter, ... đều trang sử dụng SPA để tạo nên những trải nghiệm mang tính chiều sâu và chiều rộng cho người dùng.

2.1.3. Cấu trúc của một SPA và nguyên lý hoạt động

Có rất nhiều nội dung được lặp lại trên đa số các website. Một số thành phần vẫn được giữ nguyên khi người dùng điều hướng trang web (header, footer, logo, navigation bar...), một số thì chỉ nằm ở những vị trí cố định (filter bar, banner), và còn rất nhiều layout và template được lặp lại. SPA sử dụng hiệu quả những sự lặp lại như này và nhờ vào việc linh động viết lại những thành phần cần thiết (có thay đổi) trong trang hiện tại thay vì tải lại toàn bộ một trang mới từ Server, SPA loại bỏ sự ngắt quãng khi người dùng đang trải

nghiệm những trang nối tiếp nhau, tạo cho người dùng cảm giác như đang sử dụng một ứng dụng trên desktop.

Ví dụ: màn hình website cần hiển thị bức tranh có một căn nhà và cái cây. Theo truyền thống, multi-page website sẽ chuẩn bị sẵn HTML cho toàn bộ bức tranh ấy trên Server, sau đó gửi trang HTML đến browser của client.

Trong khi đó, SPA sẽ gửi từng thành phần (cái cây, ngôi nhà, mặt trời), màu sắc (data và content) của trang web và những chỉ dẫn để kết nối những thành phần đó với nhau đến client, sau đó từ phía client sẽ dựa trên những chỉ dẫn đó mà render nên một trang web hoàn chỉnh.

Bằng cách truyền thống hay cách của SPA thì phía client đều thấy cùng trang web như nhau, nhưng SPA sẽ cho thấy sự khác biệt về tốc độ khi click vào một button next, filter 1 kết quả, mở 1 email...

2.1.4. Ưu điểm của SPA

- *Phát triển 1 lần cho tất cả các ứng dụng máy khách (Web, mobile, IOT...).*

Đối với các doanh nghiệp, việc lập trình một website Single Page Application sẽ tiết kiệm thời gian, chi phí rất nhiều khi doanh nghiệp có thể sử dụng các API backend cho cả website lẫn thiết bị di động và ứng dụng di động nếu có.

Lúc này, luồng thông tin được sắp xếp hợp lý hơn và giúp cho việc phát triển ứng dụng di động trở nên dễ dàng hơn nhiều.

Nếu doanh nghiệp phát triển các ứng dụng mua sắm Thương mại điện tử hay mạng xã hội, Single Page Application sẽ giúp doanh nghiệp tối ưu tốc độ tải. Điều này sẽ giúp gia tăng giá trị về mặt kinh tế khách hàng ở lại ứng dụng lâu hơn và họ sẽ có xu hướng sử dụng sản phẩm của bạn, góp phần trực tiếp làm tăng tỷ lệ chuyển đổi.

- *Single Page Application có tiềm năng phát triển trong tương lai.*

Single Page Application có tiềm năng phát triển rất lớn trong tương lai khi các thiết bị di động lên ngôi và hầu hết các “ông lớn” công nghệ đều đang tập trung phát triển Single Page Application để tối ưu hóa trải nghiệm người dùng.

2.1.5. Nhược điểm của SPA

Single Page Application không phải là một mô hình hoàn hảo. Mô hình này vẫn có một số nhược điểm khá bất lợi như:

- Không thể sử dụng các thủ thuật SEO cao cấp.
- Không phù hợp với các lập trình viên thiếu kinh nghiệm frontend.

- Chia tách 2 team frontend và backend sẽ làm tăng chi phí (quản lý, giao tiếp, xử lý xung đột,...).
- Gia tăng khối lượng công việc.

2.1.6. *ReactJS là gì?*

ReactJS là một thư viện Javascript có tính hiệu quả và linh hoạt để xây dựng các thành phần giao diện người dùng (UI) có thể sử dụng lại. ReactJS giúp phân chia các UI phức tạp thành các thành phần nhỏ (được gọi là *component*). Nó được tạo ra bởi Jordan Walke, một kỹ sư phần mềm tại Facebook. ReactJS ban đầu được phát triển và duy trì bởi Facebook và sau đó được sử dụng trong các sản phẩm của mình như WhatsApp & Instagram.

ReactJS được dùng để xây dựng các ứng dụng Single Page Application (SPA). Một trong những điểm hấp dẫn của ReactJS là nó không chỉ được xây dựng bên phía client mà còn sử dụng được bên phía server.

2.1.7. *Những khái niệm cơ bản của ReactJS*

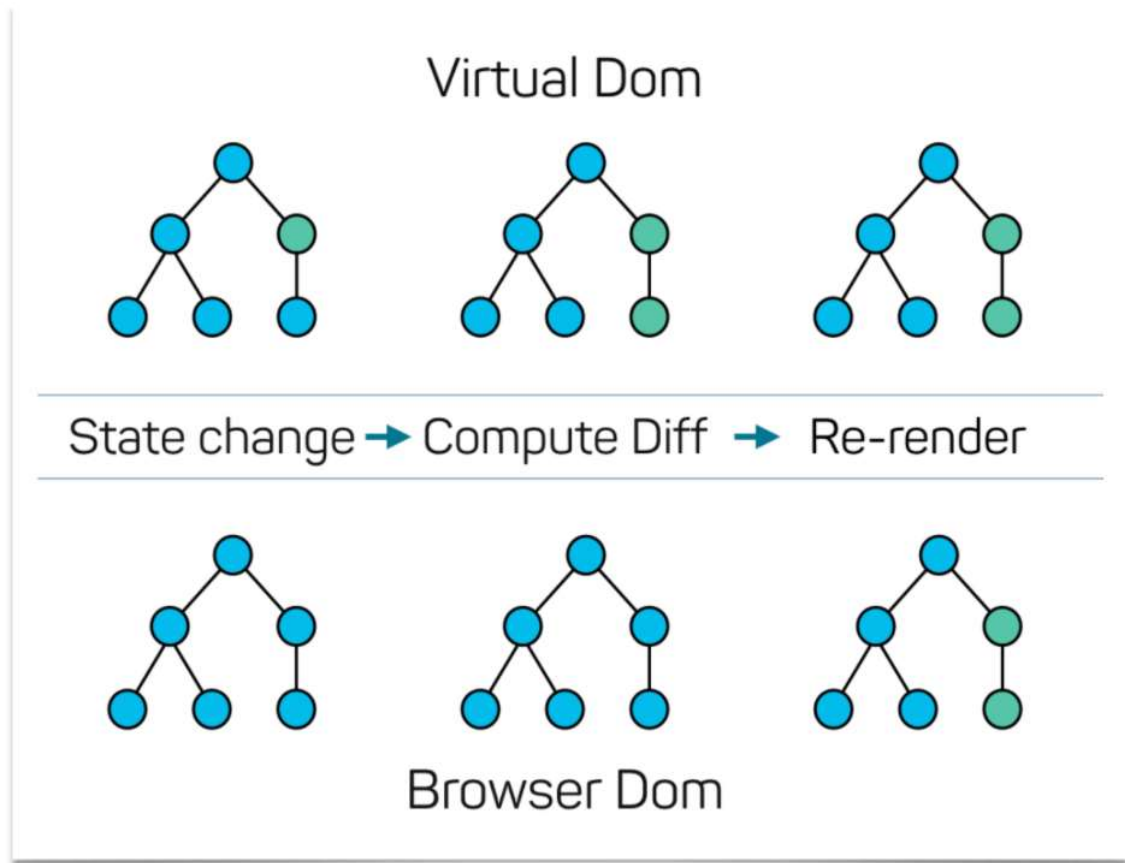
Khi bắt đầu làm quen với ReactJS, chúng ta nên làm quen với các khái niệm cơ bản của nó trước, bởi các khái niệm này sẽ đi cùng với chúng ta trong suốt quá trình học tập và làm việc với ReactJS sau này.

➤ **Virtual DOM**

Để hiểu rõ khái niệm về Virtual DOM, chúng ta cùng nhau tìm hiểu về DOM trước. DOM là một Document Object Model và là một cấu trúc trừu tượng của text. Các đoạn mã HTML được gọi là HTML DOM. Mỗi elements trong HTML là các nodes của DOM đó.

Tại sao có DOM rồi lại cần Virtual DOM (DOM ảo)? Khi chúng ta làm việc với một DOM, khi một nodes thay đổi thì tất cả các nodes cũng phải thay đổi theo. Giả sử, chúng ta có một danh sách gồm 10 items, nếu chúng ta thay đổi 1 item thì DOM cũng thay đổi 9 items còn lại về trạng thái ban đầu của nó.

Điều này là không cần thiết, mặc dù tốc độ xử lý của DOM khá nhanh nhưng đối với các ứng dụng SPA việc thay đổi các DOM này là liên tục nên nó sẽ xảy ra khá chậm và không khả thi khi xây dựng ứng dụng lớn. Lúc này Virtual DOM sẽ được dùng để thay thế. Nó được xây dựng dựa trên DOM thật, có một vài thuộc tính của DOM thật nhưng khi thay đổi Virtual DOM sẽ không thực hiện thay đổi trên màn hình giống như DOM thật.



Khi chúng ta thực hiện render một JSX element, mỗi Virtual DOM object sẽ được cập nhật, khi Virtual DOM được cập nhật, ReactJS sẽ so sánh Virtual DOM với Virtual DOM trước đó để kiểm tra trước khi thực hiện cập nhật và sau đó sẽ cập nhật trên một phần của DOM thật. Thay đổi DOM thật sẽ được hiển thị ra màn hình.

Quay lại ví dụ bên trên, thì lúc này khi chúng ta sử dụng Virtual DOM thì nó chỉ cập nhật duy nhất 1 item, lúc này tài nguyên sẽ được tiết kiệm cũng như tốc độ xử lý cũng nhanh hơn rất nhiều.

➤ JSX

JSX là viết tắt của Javascript XML, nó cho phép chúng ta viết các đoạn mã HTML trong React một cách dễ dàng và có cấu trúc hơn. Về cú pháp cũng gần tương tự như HTML, giả sử ta có 1 đoạn mã HTML như sau:

```
1 | <p class="text">Đây là đoạn mã HTML</p>
```

Thì trong JSX thì sẽ được viết như thế này:

```
1 | <p className="text">Đây là đoạn mã HTML</p>
```

Chỉ cần thay `class` thành `className` là xong

➤ Components

Khi bạn làm việc với một dự án lớn, UI có độ phức tạp cao chia các thành phần khác nhau. Việc chia nhỏ các thành phần trong UI là một điều cần thiết, các phần nhỏ này được gọi là các components, cho phép render các đoạn mã HTML,... Trong ReactJS cách viết components được chia thành 2 loại:

- Class components
- Function components

```
1 //Function component
2 function Clock(props) {
3   return (
4     <div>
5       <h1>Hello, world!</h1>
6     </div>
7   );
8 }
```

```
1 //Class component
2 class Clock extends React.Component {
3   render() {
4     return (
5       <div>
6         <h1>Hello, world!</h1>
7       </div>
8     );
9   }
10 }
```

Mỗi loại sẽ có ưu và nhược điểm khác nhau. Sự khác biệt rõ nhất là cú pháp. Một functional component thực tế chỉ là một hàm Javascript đơn giản chấp nhận các props như là một argument và trả về một phần tử React.

Một class component yêu cầu chúng ta cần kế thừa từ phần `React.Component` và tạo một function `render` trả về một phần tử React. Điều này sẽ đòi hỏi chúng ta phải code nhiều hơn nhưng sẽ cung cấp một số lợi ích mà chúng ta sẽ thấy sau này.

➤ Props

Chúng ta cần sử dụng props khi sử dụng cùng một Component nhưng với những thông số khác nhau.

Ví dụ chúng ta có một component in ra chữ “Xin chào”, nhưng vì muốn tái sử dụng để in ra “Xin chào bạn”, sử dụng props:

```

// Functional
const Hello = (props) => {
  return (
    <div className="App">
      Hello {props.name}
    </div>
  );
}

// Class component
class Hello extends React.Component {
  render() {
    return (
      <div className="App">
        Hello {this.props.name}
      </div>
    );
  }
}

// Sử dụng
<Hello name="You" />

```

Với props trong Class component được xem như giá trị truyền vào cho hàm khởi tạo class. Còn props trong Function Component thì được xem như là giá trị truyền vào hàm pure function khi định nghĩa component.

➤ **State**

Trước khi phiên bản React 16.8 ra đời thì chúng ta gần như không thể nào sử dụng state trong functional. Tuy nhiên phiên bản react 16.8 xuất hiện với một thuật ngữ hoàn toàn mới đó là hooks. Với việc xuất hiện hook thì việc sử dụng state trong functional hoàn toàn dễ dàng.

State trong Class Component được định nghĩa như sau:


```
import React, { Component } from 'react';

class TestComponent extends Component {
  constructor(props) {
    super(props);
    // khởi tạo giá trị state
    this.state = { isLoading: false };
  }

  render() {
    return <div>TestComponent</div>;
  }
}
```

Khi muốn thay đổi giá trị state, bạn gọi phương thức `setState` của component:

```
this.setState((state) => ({ isLoading: true }));
```

State trong Function Component được định nghĩa như sau:

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

export function TestComponent(props) {
  // giá trị khởi tạo state được truyền vào trong useState hook
  const [state, setState] = useState({ isLoading: false });

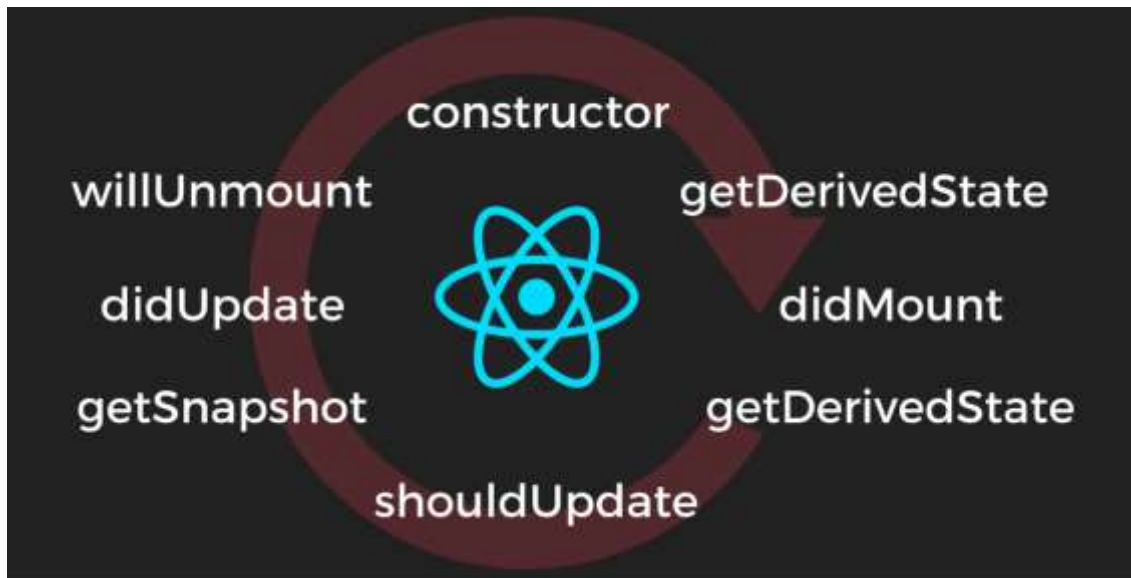
  return <div>TestComponent</div>;
}
```

Hàm **`useState`** trả về giá trị của component state trong biến state và hàm `setState`. Khi muốn thay đổi giá trị của state thì bạn có thể gọi hàm `setState`.

```
setState({ isLoading: true });
```

➤ React Lifecycle

React Lifecycle là một vòng đời của component, khi chúng ta tiến hành render một component thì ReactJS thực hiện nhiều tiến trình khác nhau, các tiến trình này được lặp đi lặp lại đối với các component.



Giả sử khi một component được gọi trước tiên nó sẽ cài đặt props và state, sau đó tiến hành mounting, update, unmounting,... việc tham gia vào quá trình này bạn cần sử dụng đến các hàm hỗ trợ của lifecycle.

2.1.8. Ưu điểm và nhược điểm của ReactJS

➤ Ưu điểm

- ReactJS cực kì hiệu quả: ReactJS tạo ra cho chính nó DOM ảo – nơi mà các component thực sự tồn tại trên đó. Điều này sẽ giúp cải thiện hiệu suất rất nhiều. ReactJS cũng tính toán những thay đổi nào cần cập nhật lên DOM và chỉ thực hiện chúng. Điều này giúp ReactJS tránh những thao tác cần trên DOM mà nhiều chi phí. Chúng ta có thể viết một ví dụ đơn giản về ReactJS như sau:

```

<title>Hello React</title>
<script src="https://fb.me/react-0.13.2.js"></script>
<script src="https://fb.me/JSXTransformer-0.13.2.js"></script>
<script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
<div id="content"></div>
<script type="text/jsx">
  var Hoge = React.createClass({
    getInitialState(){
      return{
        style:{
          color: "#ccc",
          width: 200,
          height: 100
        }
      };
    },
    onChange(){
      var style = _.clone(this.state.style);
      style.color = "#ddd";
      this.setState({ style: style});
    },
    render(){
      return(
        <div style={this.state.style} onClick={this.onChange}>xxx</div>
      );
    }
  })
  React.render(
    <Hoge />,
    document.getElementById('content')
  );
</script>

```

Ở ví dụ trên, chúng ta đã định nghĩa một component Hoge, bằng method `React.createClass (...)`, sau đó render ra View:

```
React.render(, document.getElementById('content'));
```

Khi đó ta có thể viết các methods, các functions tác động lên component Hoge (thay đổi Model data), View sẽ lập tức được cập nhật cho dù ta không phải động chạm trực tiếp gì vào phần tử DOM trên View.

- ReactJS giúp việc viết các đoạn code JS dễ dàng hơn: Nó dùng cú pháp đặc biệt là JSX (Javascript mở rộng) cho phép ta trộn giữa code HTML và Javascript. Ta có thể thêm vào các đoạn HTML vào trong hàm render mà không cần phải nối chuỗi. Đây là đặc tính thú vị của ReactJS. Nó sẽ chuyển đổi các đoạn HTML thành các hàm khởi tạo đối tượng HTML bằng bộ biến đổi JSX.

- Nó có nhiều công cụ phát triển: Khi bạn bắt đầu ReactJS, đừng quên cài đặt ứng dụng mở rộng của Chrome dành cho ReactJS. Nó giúp bạn debug code dễ dàng hơn. Sau khi bạn cài đặt ứng dụng này, bạn sẽ có cái nhìn trực tiếp vào Virtual DOM như thể bạn đang xem cây DOM thông thường.

- Render tầng server: Một trong những vấn đề với các ứng dụng đơn trang là tối ưu SEO và thời gian tải trang. Nếu tất cả việc xây dựng và hiển thị trang đều thực hiện ở client, thì người dùng sẽ phải chờ cho trang được khởi tạo và hiển thị lên. Điều này thực tế là chậm. Hoặc nếu giả sử người dùng vô hiệu hóa Javascript thì sao? ReactJS là một thư viện component, nó có thể vừa render ở ngoài trình duyệt sử dụng DOM và cũng có thể render bằng các chuỗi HTML mà server trả về.

- Làm việc với vấn đề test giao diện: Nó cực kì dễ để viết các test case giao diện vì Virtual DOM được cài đặt hoàn toàn bằng JS.

- Hiệu năng cao đối với các ứng dụng có dữ liệu thay đổi liên tục, dễ dàng cho bảo trì và sửa lỗi.

➤ **Nhược điểm**

- ReactJS chỉ phục vụ cho tầng View. React chỉ là View Library nó không phải là một MVC framework như những framework khác. Đây chỉ là thư viện của Facebook giúp render ra phần view. Vì thế React sẽ không có phần Model và Controller, mà phải kết hợp với các thư viện khác. React cũng sẽ không có two-way binding hay là Ajax.

- Tích hợp ReactJS vào các framework MVC truyền thống yêu cầu cần phải cấu hình lại.

- React khá nặng nếu so với các framework khác, React có kích thước tương đương với Angular (Khoảng 35kb so với 39kb của Angular). Trong khi đó Angular là một framework hoàn chỉnh.

- Khó tiếp cận cho người mới học Web.

2.2. Giới thiệu về TypeScript(TS)

2.2.1. TypeScript là gì?

TypeScript là một dự án mã nguồn mở được phát triển bởi Microsoft, nó có thể được coi là một phiên bản nâng cao của Javascript bởi việc bổ sung tùy chọn kiểu tĩnh và lớp hướng đối tượng mà điều này không có ở Javascript. TypeScript có thể sử dụng để phát triển các ứng dụng chạy ở client-side (ReactJS) và server-side (NodeJS).

TypeScript sử dụng tất cả các tính năng của ECMAScript 2015 (ES6) như classes, modules. Không dừng lại ở đó nếu như ECMAScript 2017 ra đời thì mình tin chắc rằng TypeScript cũng sẽ nâng cấp phiên bản của mình lên để sử dụng mọi kỹ thuật mới nhất từ ECMAScript. Thực ra TypeScript không phải ra đời đầu tiên mà trước đây cũng có một số thư viện như CoffeeScript và Dart được phát triển bởi Google, tuy nhiên điểm yếu là hai thư viện này sử dụng cú pháp mới hoàn toàn, điều này khác hoàn toàn với TypeScript, vì vậy tuy ra đời sau nhưng TypeScript vẫn đang nhận được sự đón nhận từ các lập trình viên.

2.2.2. Tại sao nên sử dụng TypeScript?

- **Dễ phát triển dự án lớn:** Với việc sử dụng các kỹ thuật mới nhất và lập trình hướng đối tượng nên TypeScript giúp chúng ta phát triển các dự án lớn một cách dễ dàng.
- **Nhiều Framework lựa chọn:** Hiện nay các Javascript Framework đã dần khuyến khích nên sử dụng TypeScript để phát triển, ví dụ như AngularJS 2.0 và Ionic 2.0.
- **Hỗ trợ các tính năng của Javascript phiên bản mới nhất:** TypeScript luôn đảm bảo việc sử dụng đầy đủ các kỹ thuật mới nhất của Javascript, ví dụ như version hiện tại là ECMAScript 2015 (ES6).
- **Là mã nguồn mở:** TypeScript là một mã nguồn mở nên bạn hoàn toàn có thể sử dụng mà không mất phí, bên cạnh đó còn được cộng đồng hỗ trợ.
- **TypeScript là Javascript:** Bản chất của TypeScript là biên dịch tạo ra các đoạn mã javascript nên bạn có thể chạy bất kỳ ở đâu miễn ở đó có hỗ trợ biên dịch Javascript. Ngoài ra bạn có thể sử dụng trộn lẫn cú pháp của Javascript vào bên trong TypeScript, điều này giúp các lập trình viên tiếp cận TypeScript dễ dàng hơn.

2.2.3. Cơ bản về TypeScript

➤ Basic Types

Trong TypeScript chia làm 7 loại cơ bản, bao gồm: boolean, number, string, array, enum, any, void. Khi khai báo ta sẽ sử dụng cấu trúc như sau:

```
var tên_biến : kiểu_trả_về = giá_trị_biến;
```

- Boolean:

```
var isDone: boolean = true;
```

- String:

```
var name: boolean = "nguyen thi A";
```

- Number:

```
var height: number = 8;
```

- Array: có 2 kiểu khai báo tương đương với nhau trong TypeScript.

```
1: var list: boolean[] = [true, false];  
2: var isDone: Array<boolean> = [true, false];
```

- Enum: khi khai báo enum một cách thông thường các phần tử sẽ được đánh số từ 0 tăng dần.

```
enum Color{Red, Green, Blue};  
var c: Color = Color.Green  
var colorName = Color[1] // kết quả sẽ là Green
```

- Any: Any là một kiểu mà bạn có thể gán bất kỳ kiểu nào cho nó.

```
var notSure: any = 4;  
notSure = "maybe a string instead";  
notSure = false; // khai báo này hoàn toàn được chấp nhận.  
                // nếu notSure ban đầu khai báo và number thì  
                // tại đây chắc chắn sẽ có lỗi  
  
// nếu sử dụng var list:number[] thì  
// tất cả các phần tử trong list sẽ phải là kiểu number  
var list:any[] = [1, true, "free"];  
list[1] = 100;
```

- Void: Cũng giống như any nhưng void được sử dụng là đầu ra của hàm.

```
function warnUser(): void {  
    alert("This is my warning message");  
}
```

➤ Function

Cũng giống như Javascript, TypeScript có 2 cách khai báo function.

```
//Named function
function add(x, y) {
    return x+y;
}

//Anonymous function
var myAdd = function(x, y) { return x+y; };
```

Nhưng khi khai báo function TypeScript còn hỗ trợ việc khai báo với các kiểu trả ra của function và cũng như kiểu đầu vào của dữ liệu.

```
function add(x: number, y: number): number {
    return x+y;
}

var myAdd = function(x: number, y: number): number { return x+y; };
```

Không những thế khi sử dụng TypeScript ta có thể khai báo giá trị mặc định của đầu vào ngay khi khai báo function.

```
function buildName(firstName: string, lastName = "Smith") {
    return firstName + " " + lastName;
}

//làm việc hoàn toàn OK. buildName("bob") = "bob Smith"
var result1 = buildName("Bob");
//error, too many parameters
var result2 = buildName("Bob", "Adams", "Sr.");
//ah, just right
var result3 = buildName("Bob", "Adams");
```

Không dừng lại ở đó TypeScript còn hỗ trợ việc bỏ qua nhập một hoặc vài đầu vào.

```
function buildName(firstName: string, lastName?: string) {
    if (lastName)
        return firstName + " " + lastName;
    else
        return firstName;
}

var result1 = buildName("Bob"); //làm việc hoàn toàn OK. buildName("bob") = "bob"
var result2 = buildName("Bob", "Adams", "Sr."); //error, too many parameters
var result3 = buildName("Bob", "Adams"); //ah, just right
```

ở ví dụ trên việc khai báo lastName? làm cho việc nhập lastName có thể không cần nữa và kết quả trả ra chỉ là undefined mà thôi. TypeScript còn có một cách làm việc với các param đầu vào có tên gọi “Rest Parameters”.

```
function buildName(firstName: string, ...restOfName: string[]) {
    return firstName + " " + restOfName.join(" ");
}

var employeeName = buildName("Joseph", "Samuel", "Lucas", "MacKinzie");
// => "Joseph Samuel Lucas MacKinzie"
```

ở trên firstName sẽ là bắt buộc phải nhập, còn các tham số còn lại sẽ được gộp chung vào một biến array.

➤ Class

```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

var greeter = new Greeter("world");
```

Hàm constructor sẽ được chạy ngay khi khởi tạo class mới, ở ví dụ trên khi khai báo **new Greeter(“world”)** thì việc đầu tiên sẽ là chạy hàm constructor gán message “world”

vào biến `greeting` của class. Hơn nữa, cũng giống như các ngôn ngữ lập trình hướng đối tượng khác, chúng ta cũng có thể dễ dàng sử dụng kế thừa trong TypeScript.

```
class Animal {
    name: string;
    constructor(theName: string) {
        this.name = theName;
    }

    move(meters: number = 0): string {
        return this.name + " moved " + meters + "m.";
    }
}

class Snake extends Animal {
    constructor(name: string) {
        super(name);
    }

    move(meters = 5): string {
        return super.move(meters);
    }
}

var ranLuc = new Snake("Ran Luc");
ranLuc.move(); // = Ran Luc moved 5 m.
ranLuc.move(34); // = Ran Luc moved 34 m.
```

Ở đây chúng ta sẽ có class `Animal` bao gồm có biến `name` chứa tên và function `move` chỉ sự di chuyển của động vật đó. Class `Snake` kế thừa lại class `Animal`. Trong hàm khởi tạo của class `Snake` ta đã sử dụng `super(name)`, việc này chính là việc gọi tới hàm constructor của class cha. Tương tự tại hàm `move` việc gọi `super.move(meters)` chính là gọi tới hàm `move` của class cha (`Animal`).

2.2.4. Ưu điểm của TypeScript

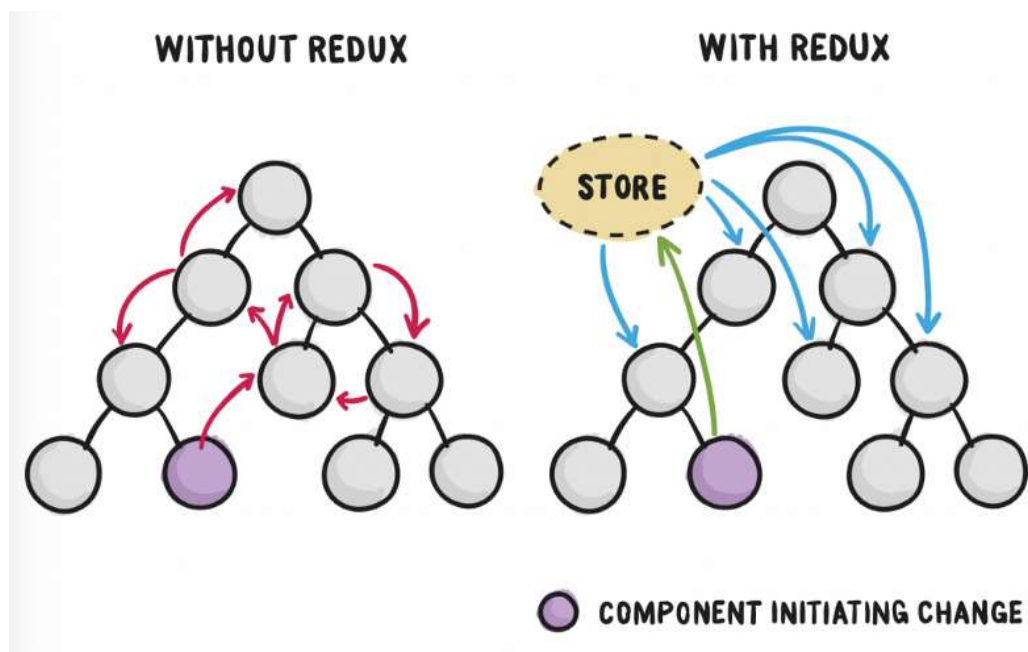
- Dễ dàng hơn trong phát triển dự án lớn, được hỗ trợ bởi các Javascript Framework lớn.
- Hầu hết các cú pháp hướng đối tượng đều được hỗ trợ bởi TypeScript như kế thừa, đóng gói, constructor, abstract, interface, implement, override, ...

- Cách tổ chức code rõ ràng hơn, hỗ trợ cơ chế giúp kiến trúc hệ thống code hướng module, hỗ trợ namespace, giúp xây dựng các hệ thống lớn nơi mà nhiều lập trình viên có thể làm việc cùng nhau một cách dễ dàng hơn.
- Hỗ trợ các tính năng mới nhất của Javascript. TypeScript luôn đảm bảo việc sử dụng đầy đủ các kỹ thuật mới nhất của Javascript, ví dụ như version hiện tại là ECMAScript 2015 (ES6).
- Một lợi thế của TypeScript nữa là mã nguồn mở vì vậy nó miễn phí và có cộng đồng hỗ trợ rất lớn.
- Với static typing, code viết bằng TypeScript dễ dự đoán và debug hơn.

2.3. Giới thiệu Redux, Redux Toolkit và RTK query

2.3.1. Khái niệm Redux

Redux là một thư viện JavaScript có tác dụng tạo ra một lớp quản lý mọi trạng thái của ứng dụng. Hay nói cách khác Redux là một predictable state management tool (công cụ quản lý cao cấp) cho các ứng dụng được viết bằng ngôn ngữ lập trình JavaScript và còn được gọi là ReduxJS.

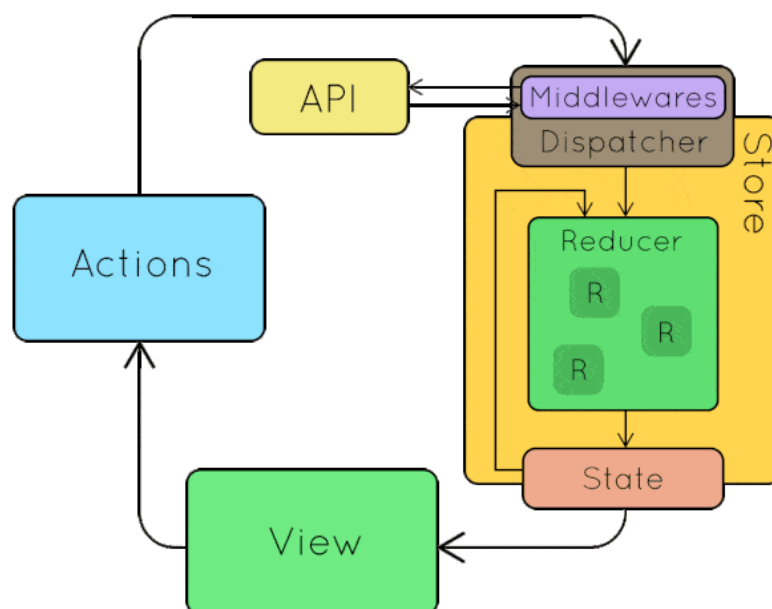


2.3.2. Lợi ích của Redux

Redux được xem là một thư viện cho phép bạn có thể quản lý tốt các ứng dụng có trong Javascript đồng thời giúp ứng dụng trở nên đơn giản và dễ bảo trì. Ngoài ra, Redux được sử dụng phổ biến bởi những lợi ích sau:

- **Hỗ trợ dự đoán trạng thái state:** Redux là phần mềm hỗ trợ dự đoán và quản lý trạng thái state. Nếu cả state và action được chuyển về reducer thì state sẽ luôn ở một trạng thái và không có dấu hiệu thay đổi. Điều này giúp người dùng có khả năng thực hiện các nhiệm vụ phức tạp như hoàn tác redo đồng thời luân chuyển linh hoạt giữa các state để đánh giá được mức độ hiệu quả trong thời điểm thực tế.
- **Khả năng bảo trì:** Redux với hệ thống code cực kỳ nghiêm ngặt giúp Redux có thể được bảo trì một cách dễ dàng hơn. Điều này giúp người dùng có khả năng tách biệt logic các nghiệp vụ ra khỏi sơ đồ thành phần,
- **Gỡ lỗi dễ dàng:** Redux cho phép người dùng có thể dễ dàng gỡ lỗi cho ứng dụng bằng cách lưu lại những state và action để thực hiện việc nhận diện các trường hợp lỗi mạng, lỗi mã hoá và một vài lỗi khác trong quá trình triển khai chương trình. Việc gỡ lỗi thường mất nhiều thời gian và phức tạp nhưng với Redux Devtool thì việc này đối với người dùng sẽ trở nên dễ dàng hơn.
- **Hiệu suất tốt:** Redux với khả năng tối ưu hoá hiệu suất giúp các thành phần được kết nối với người dùng dễ dàng và hiển thị ngay khi cần thiết.
- **Tính bền bỉ:** Tính năng này của Redux giúp người dùng có thể giữ state trong bộ nhớ cục bộ của ứng dụng và khôi phục dễ dàng.

2.3.3. Cấu trúc của Redux



Redux học hỏi kiến trúc của Flux đã ra đời trước đó nhưng lược bỏ đi sự phức tạp không cần thiết vì vậy cấu trúc cơ bản của React Redux chỉ có 4 thành phần như sau:

- **Action:** Hiểu đơn giản đây là nơi được tạo ra để lưu giữ và mang các thông tin hoặc dữ liệu từ nhiều nguồn khác nhau, có thể đến từ người dùng, máy chủ, API call, form submission,... gửi tới Store. Hệ thống dữ liệu này là một sự kiện mô tả đầy đủ những gì xảy ra, như hành động của người dùng (nhấp chuột, copy, tải,...), thời gian xảy ra hành động, hành động diễn ra ở đâu,...

- **Reducer:** Là một hàm thuần túy để xác định State đã thay đổi như thế nào từ đó trả về một trạng thái mới từ trạng thái ban đầu. Có nghĩa là nếu Action không mô tả rõ ràng trạng thái nào của response đã thay đổi và thay đổi như thế nào thì Reducer sẽ đảm nhiệm việc này. Reducer xác định trạng thái hiện tại của ứng dụng thay đổi như thế nào để đáp ứng với dữ liệu của Action được gửi đến Store.

- **Store:** Là nơi đóng vai trò quan trọng nhất thực hiện nhiệm vụ kích hoạt các Action đã được thực hiện phải sử dụng đến các phần tử dispatcher sau đó gửi đến Reducer. Store cũng là chương trình duy nhất tại Redux hỗ trợ việc lưu trữ, quản lý State, cho phép truy cập State qua getState, update State qua dispatch (action), đăng ký listener qua subscribe (listener). Store còn cho phép người dùng có thể tiếp tục truy cập và can thiệp vào những chương trình đã được lưu thông qua những phương pháp hỗ trợ gồm cập nhật, đăng ký hoặc hủy.

- **View:** Là nơi hiển thị các dữ liệu được cung cấp bởi Store

2.3.4. Nguyên lý vận hành

Quá trình xây dựng Redux được ứng dụng 3 nguyên lý vận hành cơ bản:

- *Sử dụng một nguồn dữ liệu tin cậy duy nhất*
 - Quá trình hoạt động của Redux phụ thuộc vào nhiều nguồn dữ liệu khác nhau như từ máy chủ, thao tác người dùng,... Việc có nhiều nguồn dữ liệu như vậy khiến ứng dụng khó kiểm soát hết được.
 - Để giải quyết vấn đề Redux đã đưa ra giải pháp bằng cách mọi nguồn dữ liệu đều được xử lý và tập hợp thành một nguồn dữ liệu duy nhất. Hệ thống state của toàn bộ ứng dụng được chứa trong một object tree và cây này nằm trong Store duy nhất.
- *Xây dựng trạng thái chỉ được phép đọc.*
 - Cách duy nhất để người dùng có thể thay đổi State của ứng dụng là phát một Action, tức một object mô tả tất cả những gì xảy ra. Trạng thái của Redux chỉ là một đối tượng và nó chỉ có thể thay đổi chỉ khi xuất hiện một sự kiện. ngoài ra thì không được phép thay đổi trực tiếp.

- *Chỉ thay đổi bằng hàm thuần túy*

○ Để chỉ ra cách mà State được biến đổi bởi Action, người dùng sử dụng các pure function được gọi là Reducer. Thông qua hàm thuần túy bạn có thể thực hiện việc thay đổi trạng thái của ứng dụng. Cụ thể, dữ liệu của các sự kiện và trạng thái hiện tại đưa vào sẽ được hàm xử lý và trả về trạng thái tiếp theo.

2.3.5. Redux hoạt động như thế nào ?

Cách Redux hoạt động rất đơn giản. Có một “store” trung tâm chứa toàn bộ trạng thái của ứng dụng. Mỗi thành phần có thể truy cập trạng thái được lưu trữ mà không phải gửi từ thành phần này sang thành phần khác.

Có ba phần xây dựng: actions, store và reducers. Nói ngắn gọn về cách hoạt động của từng lại. Điều này rất quan trọng vì chúng giúp chúng ta hiểu được lợi ích của Redux và cách sử dụng nó.

➤ Actions trong Redux

Nói một cách đơn giản, **action** là sự kiện. Chúng là cách duy nhất bạn có thể gửi dữ liệu từ ứng dụng của mình đến “store” Redux. Dữ liệu có thể là từ các tương tác của người dùng, các lệnh gọi API hoặc là gửi form.

Các hành động được gửi bằng phương thức **store.dispatch()**. Các hành động là các đối tượng Javascript đơn giản và chúng phải có thuộc tính loại để chỉ ra loại hành động sẽ được thực hiện. Họ cũng phải có một “payload” có chứa thông tin cần được xử lý bằng hành động. Hành động được tạo thông qua Action Creator.

Dưới đây, một ví dụ về hành động có thể được thực hiện trong quá trình đăng nhập trong ứng dụng:

```
{
  type: "LOGIN",
  payload: {
    username: "foo",
    password: "bar"
  }
}
```

Dưới đây là một ví dụ về Action Creator.

```
const setLoginStatus = (name, password) => {  
  return {  
    type: "LOGIN",  
    payload: {  
      username: "foo",  
      password: "bar"  
    }  
  }  
}
```

Như đã giải thích trước đó, action phải chứa thuộc tính và sau đó thành phần khác sẽ được “payload” lưu trữ.

➤ Reducers trong Redux

Reducers là các hàm thuần túy lấy trạng thái hiện tại của ứng dụng, thực hiện một hành động và trả về trạng thái mới. Các trạng thái này được lưu trữ dưới dạng đối tượng và chúng xác định trạng thái của ứng dụng thay đổi như thế nào để đáp ứng với hành động được gửi đến “store”.

Nó dựa trên hàm “reduce” trong Javascript, trong đó một giá trị được tính từ nhiều giá trị sau khi thực hiện chức năng gọi lại.

Ví dụ:

```
const LoginComponent = (state = initialState, action) => {
  switch (action.type) {

    // This reducer handles any action with type "LOGIN"
    case "LOGIN":
      return state.map(user => {
        if (user.username !== action.username) {
          return user;
        }

        if (user.password == action.password) {
          return {
            ...user,
            login_status: "LOGGED IN"
          }
        }
      });
    default:
      return state;
  }
};
```

Reducers lấy trạng thái trước của ứng dụng và trả về trạng thái mới dựa trên hành động được truyền cho nó.

Vì là các hàm thuần túy, chúng không thay đổi dữ liệu trong đối tượng được truyền cho chúng hoặc thực hiện bất kỳ tác dụng phụ nào trong ứng dụng. Cho cùng một đối tượng, chúng phải luôn tạo ra cùng một kết quả.

➤ Store trong Redux

Các “store” giữ trạng thái ứng dụng. Chỉ có một “store” trong bất kỳ ứng dụng Redux nào. Bạn có thể truy cập trạng thái được lưu trữ, cập nhật trạng thái và đăng ký hoặc hủy đăng ký “listeners” thông qua các phương thức trợ giúp.

Tạo store cho việc đăng nhập

```
const store = createStore(LoginComponent);
```

Các hành động được thực hiện trên trạng thái luôn trả về một trạng thái mới. Vì vậy, quản lý trạng thái là rất dễ dàng và có thể dự đoán.

Với Redux, có một trạng thái chung trong store và mỗi thành phần có quyền truy cập vào trạng thái. Điều này giúp loại bỏ sự cần thiết phải liên tục chuyển trạng thái từ thành phần này sang thành phần khác.

2.3.6. *Redux Thunk* là gì ?

Khi nhắc đến Redux thì không thể không nhớ đến Redux Thunk. Và định nghĩa này đã được các chuyên gia nhận định như sau: Redux Thunk là một Middleware có thể cho phép người dùng viết các Action trả về một function. Thay vì phải sử dụng một plain javascript object bằng cách trì hoãn quá trình đưa action đến reducer. Ngoài ra, Redux Thunk còn được sử dụng nhằm mục đích xử lý các logic bất đồng bộ phức tạp. Những đồng bộ này cần truy cập đến store hoặc lấy dữ liệu như Ajax request.

2.3.7. *Redux Persist* là gì ?

Redux Persist là dạng gói tự động hóa cho quy trình duy trì trạng thái từ cửa hàng Redux của bạn đến với bộ nhớ của thiết bị cục bộ. Ví dụ như: Redux Persist có nhiệm vụ thực hiện tái tạo cửa hàng Redux trong các lần khởi chạy ứng dụng. Các tiện ích này sẽ giúp người dùng giảm thiểu công việc cũng như quá trình để duy trì được dữ liệu trên thiết bị. Có thể là mã thông báo xác thực hoặc các cài đặt tài khoản.

Việc sử dụng Redux Persist sẽ giúp bạn thực hiện công việc hoàn toàn tự động mà chỉ cần lượng nhỏ bản ghi sẵn cho quá trình khởi tạo. Gói tự động này hoạt động rất hiệu quả và sở hữu nhiều bộ giảm được thiết kế vô cùng tốt. Ngoài ra, nó còn có thể giúp cho các bộ giảm bớt chi tiết khi cần thiết hoặc khi ứng dụng đang ngày càng phức tạp. Nhờ vậy mà quy trình quản lý cửa hàng Redux sẽ trở nên đơn giản mà hiệu quả hơn.

2.3.8. *Redux-toolkit (RTK)*

RTK (Redux-toolkit) là một thư viện giúp mình viết Redux tốt hơn, dễ hơn và đơn giản hơn (tiêu chuẩn để viết Redux).

Ba vấn đề làm nền tảng ra đời RTK:

- Cấu hình của Redux quá phức tạp.
- Phải thêm nhiều packages để Redux làm bất cứ điều gì hữu ích.
- Redux yêu cầu quá nhiều mã soạn sẵn.

2.3.9. *RTK bao gồm những gì ?*

➤ **configureStore()**

- Có sẵn Redux DevTools.
- Có sẵn Redux-Thunk để thực hiện async actions.

Khi chưa có Redux Toolkit:

```
// Khi chưa có Redux Toolkit
// store.js
import { createStore, applyMiddleware, compose } from 'redux';
import thunkMiddleware from 'redux-thunk';
import rootReducer from './reducers';
// Enable to use redux dev tool in development mode
const composeEnhancers = 'development' === process.env.NODE_ENV
  ? (window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose)
  : compose;
// Use redux-thunk as a redux middleware
const enhancer = composeEnhancers(applyMiddleware(thunkMiddleware));
const store = createStore(rootReducer, {}, enhancer);
export default store;
```

Khi đã có Redux Toolkit:

```
// Khi đã có redux toolkit 🥳
// store.js
import { configureStore } from '@reduxjs/toolkit'
import rootReducer from './reducers'
const store = configureStore({ reducer: rootReducer })
```

➤ **createReducer()**

Khi không có Redux Toolkit:

```
// Không có Redux Toolkit
function counterReducer(state = 0, action) {
  switch (action.type) {
    case 'increment':
      return state + action.payload
    case 'decrement':
      return state - action.payload
    default:
      return state
  }
}
```

Khi sử dụng Redux Toolkit:

```
// Có Redux Toolkit
// - Mỗi key là một case
// - Không cần handle default case
const counterReducer = createReducer(0, {
  increment: (state, action) => state + action.payload,
  decrement: (state, action) => state - action.payload
})
```

Một điểm hay nữa là reducer có thể mutate data trực tiếp. Bản chất bên dưới họ sử dụng thư viện Immerjs.

```
// Một điểm hay nữa là reducer có thể mutate data trực tiếp.
// Bản chất bên dưới họ sử dụng thư viện Immerjs
const todoReducer = createReducer([], {
  addTodo: (state, action) => {
    // 1. Có thể mutate data trực tiếp 🎨
    state.push(action.payload)
  },
  removeTodo: (state, action) => {
    // 2. Hoặc phải trả về state mới
    // CHỨ KO ĐƯỢC cả 1 và 2 nha 😊
    const newState = [...state];
    newState.splice(action.payload, 1);
    return newState;
  }
})
```

➤ createAction()

```
// Không có redux toolkit
const INCREMENT = 'counter/increment'
function increment(amount) {
  return {
    type: INCREMENT,
    payload: amount
  }
}
const action = increment(3)
// { type: 'counter/increment', payload: 3 }
```

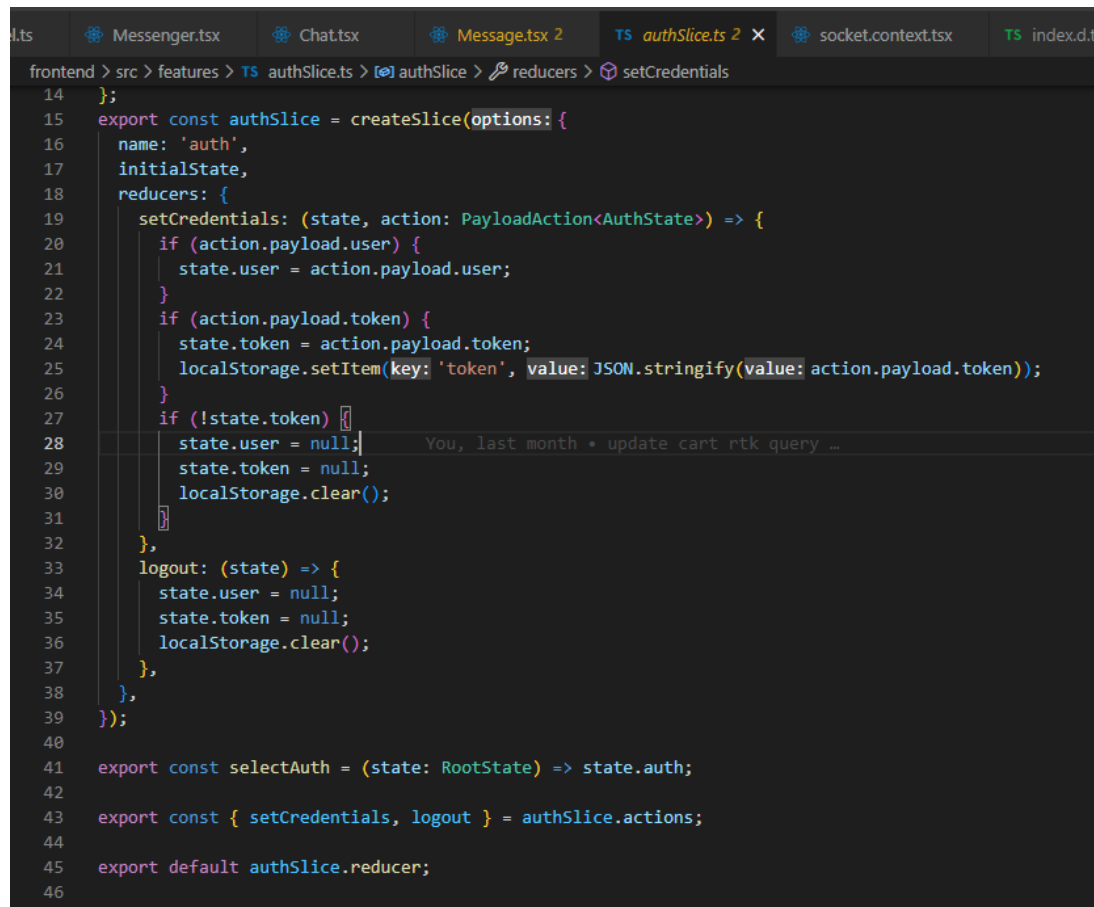
```

// Có redux toolkit
const increment = createAction('counter/increment')
const action = increment(3)
// returns { type: 'counter/increment', payload: 3 }
console.log(increment.toString())
// 'counter/increment'

```

➤ createSlice()

Phần này sẽ bao gồm cả 3 phần trên, **createSlice()** thường sử dụng, vì nó gọn và còn bao gồm cả **createReducer()** và **createAction()**.



```

14  };
15  export const authSlice = createSlice(options: {
16    name: 'auth',
17    initialState,
18    reducers: {
19      setCredentials: (state, action: PayloadAction<AuthState>) => {
20        if (action.payload.user) {
21          state.user = action.payload.user;
22        }
23        if (action.payload.token) {
24          state.token = action.payload.token;
25          localStorage.setItem(key: 'token', value: JSON.stringify(value: action.payload.token));
26        }
27        if (!state.token) {
28          state.user = null;
29          state.token = null;
30          localStorage.clear();
31        }
32      },
33      logout: (state) => {
34        state.user = null;
35        state.token = null;
36        localStorage.clear();
37      },
38    },
39  });
40
41  export const selectAuth = (state: RootState) => state.auth;
42
43  export const { setCredentials, logout } = authSlice.actions;
44
45  export default authSlice.reducer;
46

```

```

frontend > src > app > TS store.ts > persistConfig
5
6 import { persistReducer } from 'redux-persist'; 5.8k (gzipped: 2k)
7 import storage from 'redux-persist/lib/storage'; 1.5k (gzipped: 635)
8 import { cartsApi } from '../services/cartsApi';
9 import cartReducer from '../features/cartSlice';
10 import { ordersApi } from '../services/ordersApi';
11 const persistConfig = {
12   key: 'root',
13   version: 1,
14   storage,
15 };
16
17 const rootReducer = combineReducers({ reducers: {
18   auth: authReducer,
19   cart: cartReducer,
20   [authApi.reducerPath]: authApi.reducer,
21   [cartsApi.reducerPath]: cartsApi.reducer,
22   [ordersApi.reducerPath]: ordersApi.reducer,
23 }});
24
25 const persistedReducer = persistReducer({ config: persistConfig, baseReducer: rootReducer });
26 export const store = configureStore({ options: {
27   reducer: persistedReducer,
28   middleware: (getDefaultMiddleware) =>
29     getDefaultMiddleware({ serializableCheck: false }).concat([
30       authApi.middleware,
31       cartsApi.middleware,
32       ordersApi.middleware,
33     ]),
34 });
35
36 export type AppDispatch = typeof store.dispatch;
37 export type RootState = ReturnType<typeof store.getState>;
38 setupListeners(store.dispatch);

```

2.3.10. Khái niệm về RTK query

RTK query là một addon trong bộ thư viện Redux Toolkit. Nó giúp chúng ta thực hiện data fetching một cách đơn giản hơn thay vì sử dụng `createAsyncThunk` để thực hiện async action. Chú ý RTK query là dùng để query (kết nối API), chứ không phải dùng để code async trong Redux thay cho `createAsyncThunk`.

RTK Query được bao gồm trong quá trình cài đặt gói Redux Toolkit. Nó được có sẵn thông qua một trong hai điểm nhập dưới đây:

```

import { createApi } from '@reduxjs/toolkit/query'

/* React-specific entry point that automatically generates
   hooks corresponding to the defined endpoints */
import { createApi } from '@reduxjs/toolkit/query/react'

```

RTK Query bao gồm các API:

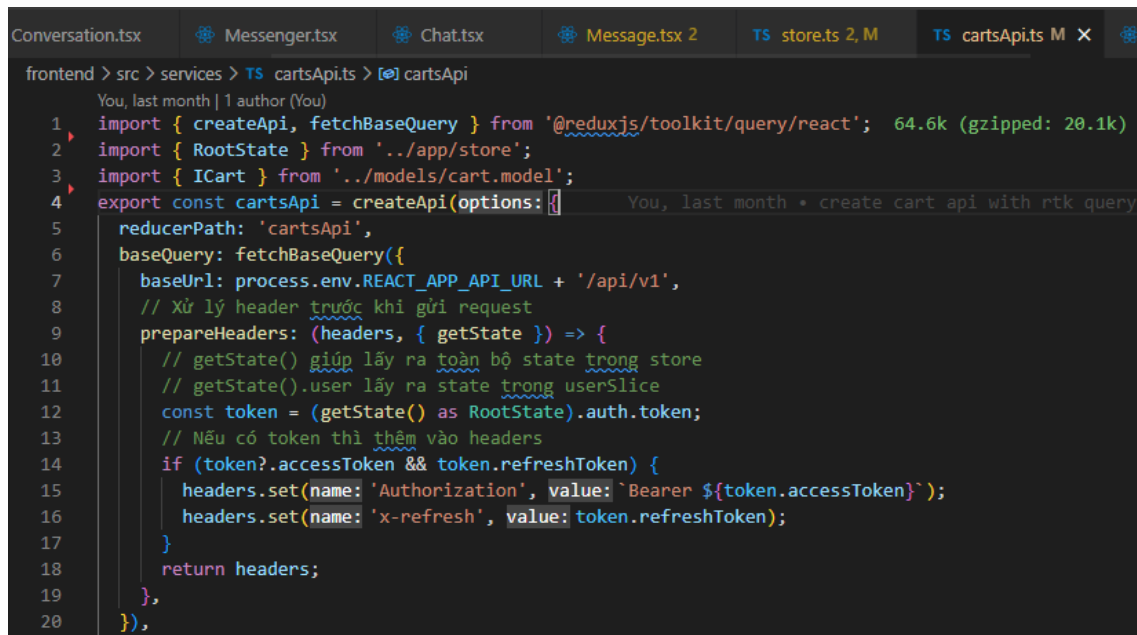
- **createApi()**: là chức năng cốt lõi của RTK Query. Nó cho phép xác định một tập hợp các endpoints mô tả cách truy xuất dữ liệu từ một loại endpoints, bao gồm cả cấu hình về cách tìm nạp và chuyển đổi dữ liệu đó. Trong hầu hết các trường hợp, nên sử dụng tính năng này một lần cho mỗi ứng dụng.

- **fetchBaseQuery()**: Một trình bao bọc nhỏ xung quanh tìm nạp nhằm mục đích đơn giản hóa các yêu cầu. Dự định là *baseQuery* được đề xuất sẽ được sử dụng trong *createApi* cho phần lớn người dùng.

- **<ApiProvider>**: Có thể được sử dụng làm Provider nếu chưa có Redux store.

- **setupListeners()**: Một tiện ích được sử dụng để kích hoạt các hành vi *refetchOnMount* và *refetchOnReconnect*.

Đầu tiên chúng ta tạo 1 file mới, tương tự như tạo một slice, và file này dùng để khai báo các lệnh gọi API. Ví dụ tạo 1 file *cartApi.ts* với nội dung như sau:



```
frontend > src > services > TS cartsApi.ts > [x] cartsApi
You, last month | 1 author (You)
1 import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react'; 64.6k (gzipped: 20.1k)
2 import { RootState } from '../app/store';
3 import { ICart } from '../models/cart.model';
4 export const cartsApi = createApi(options: { You, last month * create cart api with rtk query
5   reducerPath: 'cartsApi',
6   baseQuery: fetchBaseQuery({
7     baseUrl: process.env.REACT_APP_API_URL + '/api/v1',
8     // Xử lý header trước khi gửi request
9     prepareHeaders: (headers, { getState }) => {
10       // getState() giúp lấy ra toàn bộ state trong store
11       // getState().user lấy ra state trong userSlice
12       const token = (getState() as RootState).auth.token;
13       // Nếu có token thì thêm vào headers
14       if (token?.accessToken && token.refreshToken) {
15         headers.set(name: 'Authorization', value: `Bearer ${token.accessToken}`);
16         headers.set(name: 'x-refresh', value: token.refreshToken);
17       }
18       return headers;
19     },
20   }),
```

```

21 tagTypes: ['Cart'],
22 endpoints: (builder) => ({
23   getMyCart: builder.query<ICart, {}>({definition: {
24     query: () => {
25       return {
26         url: '/cart/me',
27         method: 'GET',
28         // credentials: 'include',
29       };
30     },
31     providesTags: ['Cart'],
32   }),
33   addItemToCart: builder.mutation<
34     { message: string; data: ICart },
35     { product: string; color: string; size: string | number }
36   >({definition: {
37     query: (body) => {
38       return {
39         url: '/cart/add-to-cart',
40         method: 'POST',
41         body,
42       };
43     },
44     invalidatesTags: ['Cart'],
45   }),
46   removeItemFromCart: builder.mutation<ICart, string>({definition: {
47     query: (cartItemId) => {
48       return {
49         url: `cart/remove-item-from-cart/${cartItemId}`,
50         method: 'PUT',
51       };
52     },
53     invalidatesTags: ['Cart'],
54   }),

```

```

55   updateQuantityCart: builder.mutation<ICart, { cartItemId: string; quantity: number }>({definition: {
56     query: ({ cartItemId, quantity }) => {
57       return {
58         url: `cart/${cartItemId}`,
59         method: 'PUT',
60         body: { quantity },
61       };
62     },
63     invalidatesTags: ['Cart'],
64   }),
65   }),
66 });
67
68 export const {
69   useGetMyCartQuery,
70   useAddItemToCartMutation,
71   useRemoveItemFromCartMutation,
72   useUpdateQuantityCartMutation,
73 } = cartsApi;
74

```

Nhúng API này vào trong store như một Slice:

```

frontend > src > app > TS store.ts > [x] persistConfig
6 import { persistReducer } from 'redux-persist'; 5.8k (gzipped: 2k)
7 import storage from 'redux-persist/lib/storage'; 1.5k (gzipped: 635)
8 import { cartsApi } from '../services/cartsApi';
9 import cartReducer from '../features/cartSlice';
10 import { ordersApi } from '../services/ordersApi';
11 const persistConfig = {
12   key: 'root',
13   version: 1,
14   storage,
15 };
16
17 const rootReducer = combineReducers(reducers: {
18   auth: authReducer,
19   cart: cartReducer,
20   [authApi.reducerPath]: authApi.reducer,
21   [cartsApi.reducerPath]: cartsApi.reducer,
22   [ordersApi.reducerPath]: ordersApi.reducer,
23 });
24
25 const persistedReducer = persistReducer(config: persistConfig, baseReducer: rootReducer);
26 export const store = configureStore(options: {
27   reducer: persistedReducer,
28   middleware: (getDefaultMiddleware) =>
29     getDefaultMiddleware(options: { serializableCheck: false }).concat(items: [
30       authApi.middleware,
31       cartsApi.middleware,
32       ordersApi.middleware,
33     ]),
34 });
35
36 export type AppDispatch = typeof store.dispatch;
37 export type RootState = ReturnType<typeof store.getState>;
38 setupListeners(store.dispatch);

```

Sau khi cấu hình xong, chúng ta có thể thêm các endpoint để thực hiện request đến API. Endpoint trong RTK Query phân làm 2 loại:

- Query: Dùng để lấy dữ liệu (có thể lưu cache).
- Mutation: Dùng để cập nhật dữ liệu (validate cache).

Với request addItemToCart thì sẽ dùng loại mutation và sử dụng mutation để gọi API:

```

77 const [addItemToCart, { isLoading: isLoadingAddToCart }] = useAddItemToCartMutation();
78 const { refetch } = useGetMyProfileQuery(arg: {});

156 const handleAddToCart = async () => {
157   if (product && defaultColor && defaultSize) {
158     if (user) {
159       await addItemToCart(arg: {
160         product: product._id,
161         color: defaultColor.colorName,
162         size: defaultSize.size,
163       });
164     }
165     dispatch(
166       thunkAction: addToCart(payload: {
167         product: product,
168         color: defaultColor.colorName,
169         size: defaultSize.size,
170         quantity: 1,
171         image: defaultColor.images[0],
172       }),
173     );
174   } else {
175     toast.warn(content: 'Vui lòng chọn màu sắc và kích cỡ !');
176   }

```

2.4. Giới thiệu về SASS/SCSS và CSS Module

2.4.1. CSS Preprocessor là gì ?

CSS Preprocessor là một chương trình cho phép khởi tạo CSS từ cú pháp của bộ tiền xử lý (pre-processor). Hầu hết các CSS Preprocessor sẽ thêm một số tính năng không có sẵn trong CSS thuần túy, chẳng hạn như mixin, nesting selector, inheritance selector,... Các tính năng này giúp cấu trúc của CSS dễ đọc và dễ bảo trì hơn.

Để sử dụng một CSS Preprocessor, người dùng phải cài đặt CSS compiler trên web server hoặc dùng CSS Preprocessor để biên dịch môi trường dev, sau đó upload file CSS đã được biên dịch lên web server.

Một số CSS Preprocessor phổ biến gồm có: SASS, SCSS, LESS, Stylus và PostCSS. Về cơ bản thì SASS và SCSS là giống nhau, chỉ khác về cách viết.



2.4.2. SASS là gì ?

SASS (Syntactically Awesome Style Sheets) là một tiện ích mở rộng của CSS. Các ngôn ngữ style sheet kiểm soát vị trí và cách thức văn bản hiển thị trên webpage, từ kích thước và màu sắc khung cho đến vị trí của menu.

CSS được sử dụng hầu hết các trang web, thiết kế để giúp các developer viết lệnh hiển thị văn bản trên màn hình. CSS không được xây dựng để làm việc với các biến số hay thực hiện nhiều tác vụ phức tạp. SASS/SCSS chính là giải pháp giải quyết những khuyết điểm của CSS, giúp các developer tiết kiệm thời gian và công sức cho những dự án của mình.

2.4.3. Ưu điểm của SASS

- Thứ nhất, code SASS được tổ chức tốt hơn so với CSS. Về mặt lý thuyết, CSS và cả SASS đều có khả năng giống nhau, nhưng SASS có khối lượng code nhỏ hơn. Do đó, code của SASS dễ đọc và hiểu hơn, đặc biệt là trong những dự án web với nhiều developer khác nhau.
- SASS có cú pháp tương đối giống với CSS nên việc học sẽ không có nhiều khó khăn.
- Có khả năng tái sử dụng. SASS cho phép tái sử dụng các biến và đoạn code nhiều lần, giúp các developer tiết kiệm được nhiều thời gian và giảm thiểu bug trong code. Bên cạnh đó, việc thay đổi style cũng nhanh chóng và đơn giản.
- SASS có độ ổn định cao. Được ra mắt lần đầu vào năm 2006, SASS được hỗ trợ bởi nhiều developer kinh nghiệm đến từ các công ty công nghệ lớn nên sẽ được cập nhật và duy trì liên tục.

2.4.4. Cách thức hoạt động của SASS là gì ?

SASS là một ngôn ngữ preprocessor được biên dịch sang CSS. Các ngôn ngữ này nhận dữ liệu đầu vào và chuyển thành một input khác để dùng cho những chương trình khác. Do đó, bản chất của việc chạy code SASS là chuyển đổi code sang CSS. Output này sau đó sẽ được sử dụng bởi trình duyệt web vì các trình duyệt chỉ có khả năng đọc code CSS.

2.4.5. SCSS là gì ?

SCSS (Sassy Cascading Style Sheets) là một ngôn ngữ tiền xử lý được biên dịch thành CSS, với phần mở rộng file có dạng .scss. SCSS cho phép thêm các tính năng bổ sung vào CSS, bao gồm các biến, nesting,... Các tính năng bổ sung có thể giúp việc viết SCSS đơn giản và nhanh chóng hơn nhiều so với CSS tiêu chuẩn. SCSS có thể sử dụng code và hàm của CSS, tuân theo cú pháp của CSS và hỗ trợ mọi tính năng có trong SASS.

2.4.6. Ưu điểm và nhược điểm của SCSS

➤ Ưu điểm

- SCSS cho phép viết code gọn gàng, nhanh chóng hơn trong cấu trúc chương trình.
- SCSS cung cấp tính năng nesting, do đó developer có thể sử dụng cú pháp lồng nhau và nhiều hàm hữu ích, trong đó có cả các thao tác liên quan đến màu hay dùng hàm toán học,...

- Cung cấp các biến để tái sử dụng các giá trị trong CSS.
- Tương thích được với mọi phiên bản CSS.

➤ **Nhược điểm**

- Debug: Các pre-processor có một giai đoạn biên dịch, do đó code CSS trở nên vô nghĩa trong quá trình debug. Ngoài ra việc debug cũng trong SCSS cũng khó hơn rất nhiều.
- File CSS lớn: Dù source file có kích thước không đáng kể thì file CSS vẫn sẽ có kích thước lớn hơn nhiều.
- Ngoài ra, việc sử dụng SCSS cũng có thể làm vô hiệu hóa một số inspector tích hợp sẵn trong trình duyệt.

2.4.7. Các tính năng cơ bản của SCSS

➤ **Xếp chồng (Nested Rules)**

Đây là một trong những tính năng được sử dụng thường xuyên nhất của SCSS. Ví dụ:

```
<div class="container">
  <div class="row">
    <div class="navbar col-14">
      <a class="brand">TopDev</a>
      <ul class="menu">
        <li><a href="#">Menu 1</a></li>
        <li><a href="#">Menu 2</a></li>
      </ul>
    </div>
  </div>
</div>
```

Giả sử ta chỉ cần CSS cho thẻ ul với class menu thì trong CSS ta viết như sau:

```
.navbar ul.menu {
  list-style: none;
}
```

Nếu muốn CSS cho thẻ li trong ul thì:

```
.navbar ul.menu li {
  padding: 3px;
}
```

Sau đó nếu muốn thêm tiếp CSS cho thẻ a trong li, thì ta cần phải lặp đi lặp lại tên parent tag (class hoặc id) của thẻ muốn CSS. Tuy nhiên việc này mất rất nhiều thời gian, do đó ta có thể dùng tính năng Nested Rules của SASS.

Ví dụ:

```

.navbar {
  ul.menu {
    list-style: none;

    li {
      padding: 3px;

      a {
        text-decoration: none;
      }
    }
  }
}

```

Sau khi compile ra CSS thuần:

```

.navbar ul.menu {
  list-style: none;
}
.navbar ul.menu li {
  padding: 3px;
}
.navbar ul.menu li a {
  text-decoration: none;
}

```

➤ **Biến (variable)**

Sử dụng biến với SCSS không quá phức tạp, ta chỉ cần đặt tên cho biến (bắt đầu bằng \$).

Biến chứa các giá trị ta có thể dùng nhiều lần, chẳng hạn như mã màu, font hay kiểu chữ.

```

$RedColor = #fff;

.navbar {
  ul.menu {
    list-style: none;

    li {
      padding: 3px;

      a {
        text-decoration: none;
        color: $Redcolor
      }
    }
  }
}

```

➤ Mixin rule

Mixin cho phép tạo các hàm được dùng trong SCSS và người dùng có thể truyền thêm các tham số nếu cần. Mixin là một cơ chế khá phổ biến trong SASS, mang nhiều thuộc tính được quy ước trong một mix nào đó rồi @include vào một thành phần bất kỳ mà không cần viết lại thuộc tính.

```
@mixin colorVsStyle {  
    color: #f06;  
    font-style: italic;  
}  
  
.navbar {  
    ul.menu {  
        list-style: none;  
  
        li {  
            padding: 5px;  
  
            a {  
                text-decoration: none;  
                @include colorVsStyle;  
            }  
        }  
    }  
}
```

Nếu không muốn color luôn nhận giá trị #f06 thì có thể truyền thuộc tính vào mix như một tham số:

```

@mixin colorVsStyle($color, $fontStyle) {
  color: $color;
  font-style: $fontStyle;
}

.navbar {
  ul.menu {
    list-style: none;

    li {
      padding: 5px;

      a {
        text-decoration: none;
        @include colorVsStyle(#000, italic);
      }
    }
  }
}

```

➤ Kế thừa (Extend)

Để sử dụng ta chỉ cần định nghĩa một class, rồi thêm @extend với những tag cần:

```

.title-box {
  color: ##2EFEC8;
  text-shadow: 0px 0px 10px #6E6E6E;
  display: inline-block;
  text-transform: uppercase;
}

.navbar {
  ul.menu {
    list-style: none;

    li {
      padding: 4px;

      a {
        text-decoration: none;
        @extend .title-box;
      }
    }
  }
}

```

➤ Import

Cú pháp import rất hữu dụng và thường xuyên được sử dụng trong các dự án, tương tự với việc require hay thêm include file vào những file khác trong PHP.

Giả sử ta có một trang index gồm header, body và footer. Thay vì sử dụng CSS để cho tất cả vào một file style.css thì ta chỉ cần thực hiện các bước sau với SASS:

1. Tạo 1 file `_header.scss` để CSS riêng cho header.
2. `_body.scss` để CSS riêng cho body.
3. `_footer.scss` để CSS riêng cho footer.
4. `@import` 3 file trên vào `style.css`.

`_header.scss`

```

#header {
  // code sass
}

```

`_body.scss`

```

#body {
  // code sass
}

```

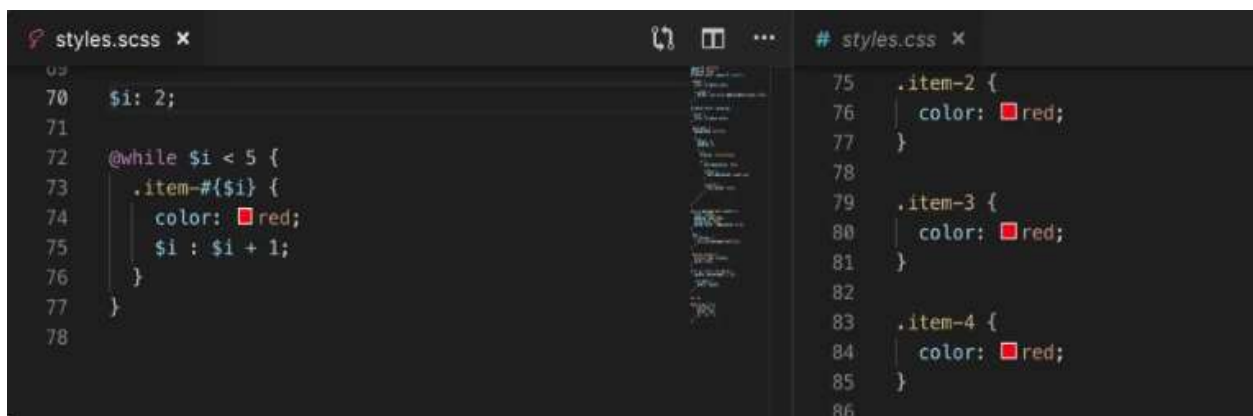
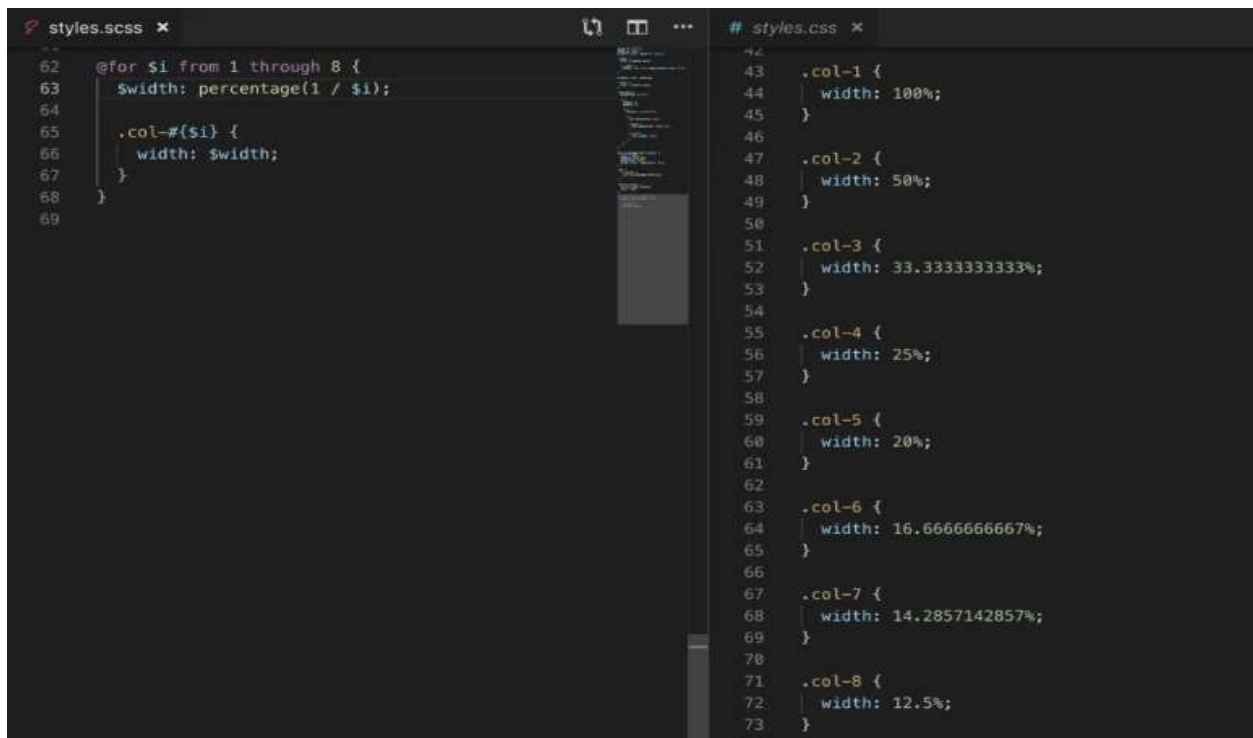
_footer.scss

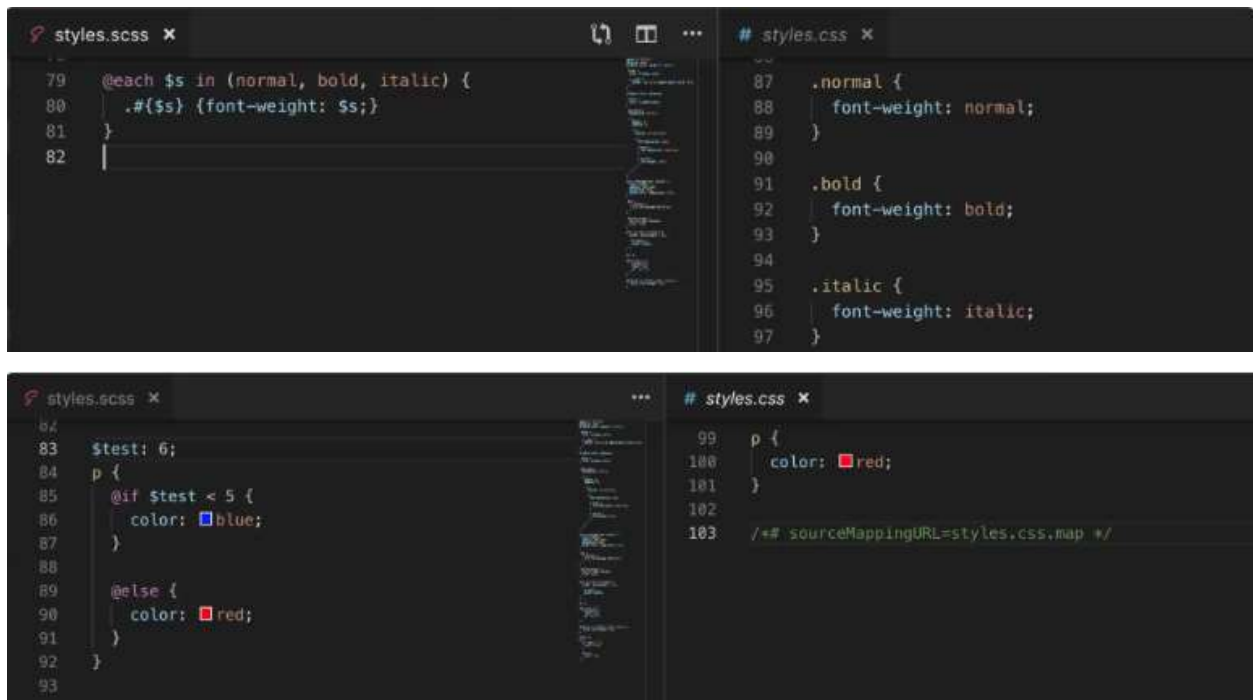
```
#footer {  
    // code sass  
}
```

_style.scss

```
@import 'header';  
@import 'body';  
@import 'footer';  
  
// code sass
```

➤ Vòng lặp

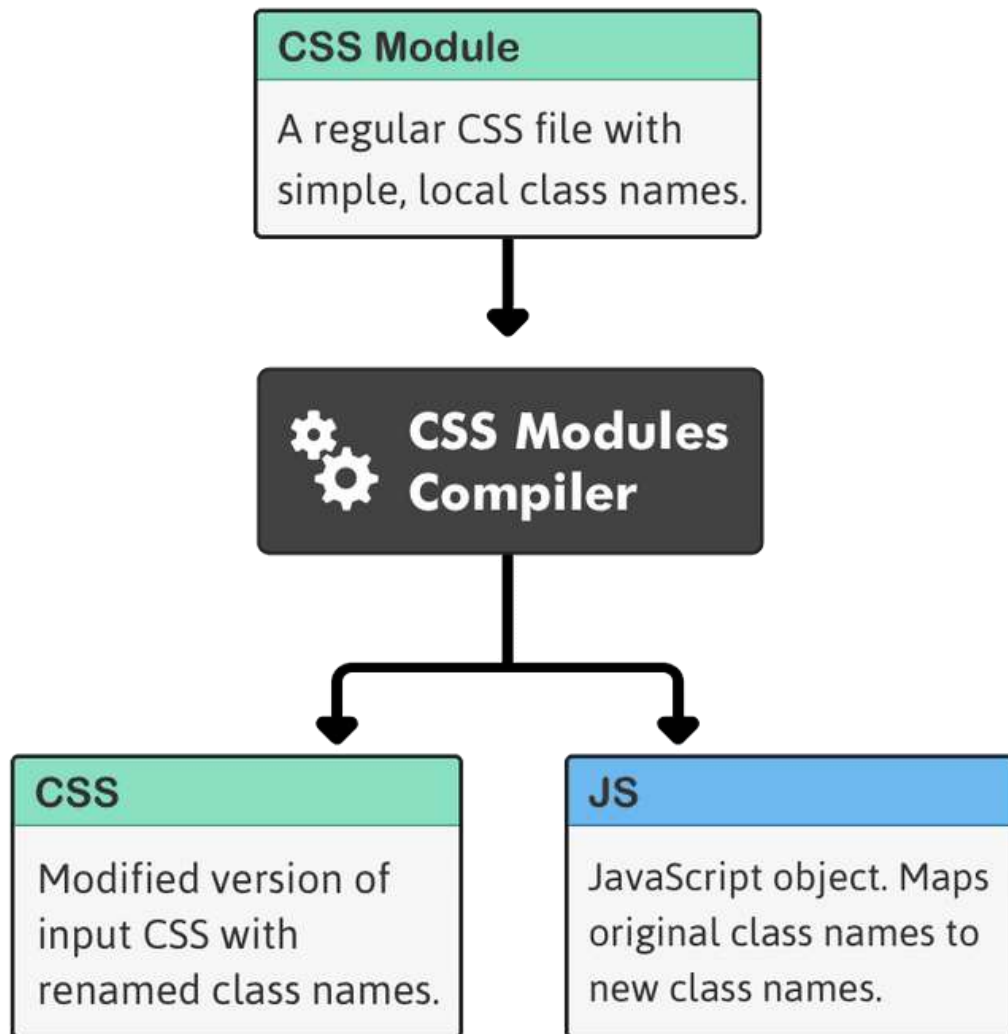




2.4.8. CSS Module là gì ?

CSS Module là những file css bao gồm tất cả các **class names** và **animation names**.

Vì vậy, cũng gần giống như một số ngôn ngữ CSS mở rộng như SASS hay SCSS, CSS Module không thể thực thi trực tiếp trên trình duyệt mà cần thông qua các trình biên dịch (Webpack hoặc Browserify).



Bây giờ ta xét một ví dụ cụ thể cho dễ hiểu. Đầu tiên ta cần 1 file HTML và CSS thông thường như sau:

```
<h1 class="title">An example heading</h1>
```

```
.title {  
  background-color: red;  
}
```

Khi áp dụng đoạn code trên, chúng ta sẽ có được thẻ h1 màu đỏ, browser sẽ hiểu cả 2 file và không cần xử lý gì cả.

Nhưng CSS Module thì khác, thay vì viết code HTML, chúng ta cần viết code bằng Javascript, ví dụ như sau:

```
import styles from "./styles.css";

element.innerHTML =
  `

# 


```

style.css

```
.title {
  background-color: red;
}
```

Trong quá trình build, trình biên dịch sẽ duyệt tìm qua file “./styles.css”. Và sau đó sẽ tìm qua file Javascript mà chúng ta đã viết. Tạo class “./title” có thể truy cập qua {styles.title}. Tiếp đó compiler sẽ xử lý để tạo 2 file HTML và CSS mới, với một chuỗi ký tự mới thay thế cho cả HTML và CSS Selector.

```
<h1 class="_styles__title_309571057">
  An example heading
</h1>
```

```
._styles__title_309571057 {
  background-color: red;
}
```

Thuộc tính class và selector “.title” đã hoàn toàn biến mất, được thay thế hoàn toàn bằng một chuỗi mới. File CSS ban đầu chúng ta không hề được sử dụng trên browser.



2.4.9. Tại sao chúng ta nên sử dụng CSS Module ?

CSS Module đảm bảo rằng tất cả style cho một component:

- Chỉ tồn tại ở 1 nơi.
- Chỉ được sử dụng cho riêng component đó mà không sử dụng ở bất kỳ chỗ nào khác.
- Thêm vào đó bất kỳ component nào đều có một sự phụ thuộc, ví dụ:

```
import buttons from './buttons.css';
import padding from './padding.css';

element.innerHTML = `<div class="${buttons.red} ${padding.large}">`;
```

- Với cách này chúng ta khắc phục được vấn đề phạm vi global trong css.
- Với CSS Module và khái niệm phạm vi local theo mặc định, chúng ta sẽ tránh được vấn đề này.

2.5. Giới thiệu NodeJS

2.5.1. Khái niệm NodeJS

NodeJS là một nền tảng (Platform) phát triển độc lập được xây dựng trên V8 JavaScript Engine – trình thông dịch thực thi mã JavaScript giúp chúng ta có thể xây dựng được các ứng dụng web.

NodeJS có thể chạy trên nhiều nền tảng hệ điều hành khác nhau từ Window cho tới Linux, OS X nên đó cũng là một lợi thế. NodeJS cung cấp các thư viện phong phú ở dạng Javascript Module khác nhau giúp đơn giản hóa việc lập trình và giảm thời gian ở mức thấp nhất.

Ý tưởng chính của NodeJS là sử dụng non-blocking, hướng sự vào ra dữ liệu thông qua các tác vụ thời gian thực một cách nhanh chóng. NodeJS đưa ra là sử dụng luồng đơn (Single-Threaded), kết hợp với non-blocking I/O để thực thi các request, cho phép hỗ trợ hàng chục ngàn kết nối đồng thời. Do đó, NodeJS có khả năng mở rộng nhanh chóng, khả năng xử lý một số lượng lớn các kết nối đồng thời bằng thông lượng cao.

2.5.2. Các tính năng của NodeJS

➤ Lập trình hướng sự kiện và không đồng bộ

Toàn bộ API trong thư viện NodeJS đều không đồng bộ, hay không bị chặn. Về cơ bản điều này có nghĩa là một server sử dụng NodeJS sẽ không bao giờ chờ một API trả về dữ

liệu. Server sẽ chuyển sang API kế tiếp sau khi gọi API đó và cơ chế thông báo của Events trong NodeJS giúp server nhận được phản hồi từ lần gọi API trước.

➤ **Cực kì nhanh chóng**

Được xây dựng trên Công cụ JavaScript V8 của Google Chrome, thư viện NodeJS có khả năng xử lý mã vô cùng nhanh.

➤ **Đơn luồng (Single Thread) nhưng có khả năng mở rộng cao**

NodeJS sử dụng một mô hình luồng đơn với vòng lặp sự kiện. Cơ chế event cho phép máy chủ phản hồi non-blocking và cũng cho phép khả năng mở rộng cao hơn so với các server truyền thống hỗ trợ giới hạn các thread để xử lý yêu cầu. NodeJS sử dụng một chương trình đơn luồng, cùng một chương trình có thể cung cấp dịch vụ cho một số lượng yêu cầu lớn hơn so với các máy chủ truyền thống như Apache HTTP Server.

➤ **Không có buffer**

Các ứng dụng NodeJS không có vùng nhớ tạm thời (buffer) cho bất kỳ dữ liệu nào. Các ứng dụng này chỉ đơn giản xuất dữ liệu theo khối.

➤ **License**

NodeJS được phát hành theo giấy phép MIT.

2.5.3. Ứng dụng của NodeJS

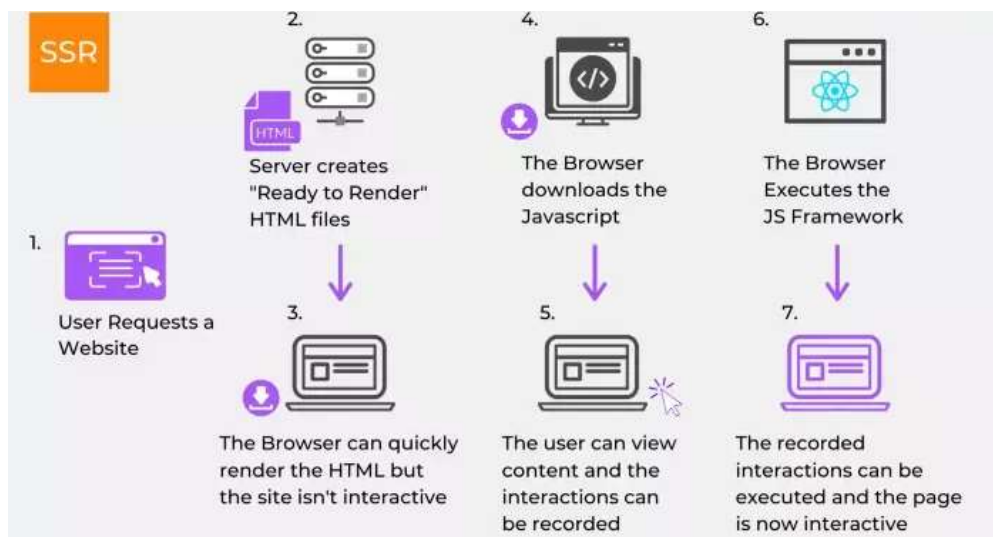
- **Hệ thống Notification:** Giống như Facebook hay Twitter.
- **WebSocket Server:** Các máy chủ web socket như là Online Chat, Game Server,...
- **Fast File Upload Client:** Các chương trình upload file tốc độ cao.
- **Ad Server:** Các máy chủ quảng cáo.
- **Cloud Services:** Các dịch vụ đám mây.
- **RESTful API:** Những ứng dụng mà được sử dụng cho các ứng dụng khác thông qua API.
- **Any Real-time Data Application:** Bất kỳ một ứng dụng nào có yêu cầu về tốc độ thời gian thực.
- **Ứng dụng Single Page Application (SPA):** Những ứng dụng này thường request rất nhiều đến server thông qua AJAX.
- **Ứng dụng truy vấn tới NoSQL database:** Như MongoDB, CouchDB,...
- **Ứng dụng CLI:** Các công cụ sử dụng command-line.

2.5.4. Server Side Rendering(SSR) và Client Side Rendering(CSR)

➤ **Server-Side Rendering (SSR)**

Server-Side Rendering hay SSR là cách thông thường cho việc render trang web ở trình duyệt. Như các bước mô tả bên dưới cách truyền thống để rendering nội dung web như các bước dưới đây:

- Người dùng gửi một yêu cầu tới website (Thông thường thông qua trình duyệt).
- Phía server kiểm tra và chuẩn bị nội dung HTML sau khi đã đi qua một lượt các script có trong trang web.
- Các đoạn HTML đã được biên dịch được gửi tới trình duyệt của người dùng cho việc render.
- Trình duyệt tải về HTML và làm các trang có thể nhìn thấy với người dùng.
- Trình duyệt sau đó tải về Javascript(JS) và tiến hành thực thi JS, nó làm cho trang web có thể tương tác.



Trong quá trình này, tất cả các gánh nặng của việc lấy nội dung động, chuyển chúng thành HTML, gửi chúng tới trình duyệt đều ở phía server. Do đó, quá trình này được gọi là Server side rendering(SSR).

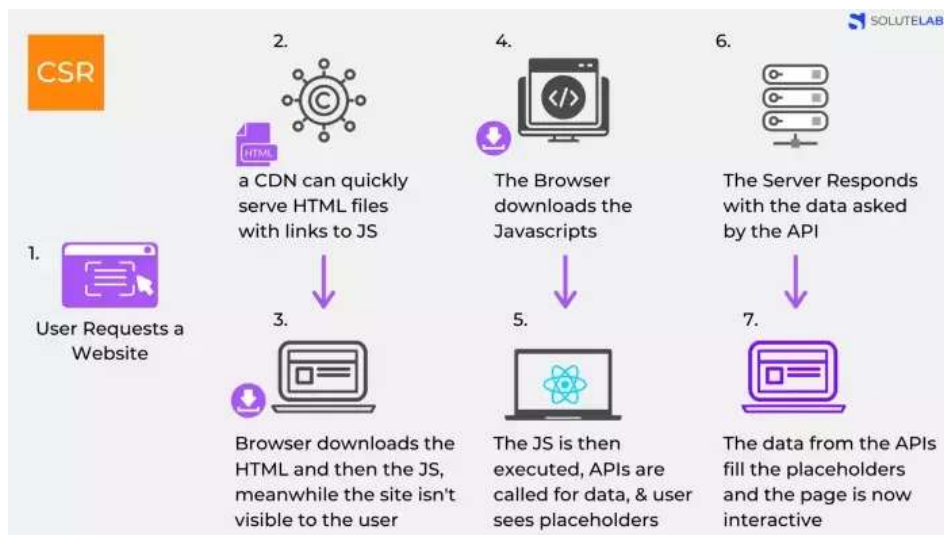
Việc chịu trách nhiệm cho việc render HTML hoàn thiện dẫn tới server tốn bộ nhớ và tài nguyên để xử lý. Do đó, server side rendering làm tăng thời gian tải trang khi so sánh với các trang tĩnh (các trang web không render các nội dung động).

➤ Client-Side Rendering (CSR)

Client-Side Rendering là một cách tiếp cận khác về việc làm thế nào một trang web được xử lý để hiển thị trên trình duyệt. Ở CSR, gánh nặng về việc biên dịch nội dung, sinh ra HTML được chuyển tới phía trình duyệt người dùng.

Cách tiếp cận này được tiếp sức mạnh từ các framework Javascript và các thư viện. Luồng chính của một trang web render trong trường hợp Client-Side Rendering như sau:

- Người dùng gửi request tới website.
- Thay vì một server, một con CDN có thể được sử dụng để gửi HTML, CSS và các file hỗ trợ cho người dùng.
- Trình duyệt tải HTML và JS trong khi nhìn thấy một biểu tượng loading.
- Sau khi trình duyệt lấy JS về, nó sẽ tạo các yêu cầu API thông qua Ajax và lấy về các nội dung động và xử lý chúng để render ra nội dung cuối cùng.
- Sau khi server phản hồi, nội dung cuối cùng sẽ được render sử dụng quá trình xử lý DOM trên trình duyệt người dùng.



Bởi vì quá trình liên quan đến việc fetching và xử lý dữ liệu ở phía client nên quá trình này được gọi là client-side rendering.

➤ So sánh SSR và CSR

- Thời gian tải trang lần đầu

Thời gian tải trang lần đầu là thời gian trung bình khi người dùng tải trang web của bạn lần đầu tiên. Ở lần tải đầu tiên ở CSR, trình duyệt tải HTML, CSS, và tất cả các script sau đó biên dịch HTML thành nội dung có thể sử dụng trên trình duyệt.

Khoảng thời gian này thường nhiều hơn là việc lấy về một đoạn HTML đã được biên dịch và các script tương ứng. Do đó SSR sẽ tốn ít thời gian hơn cho việc tải trang lần đầu.

- Lần thứ hai và các lần tải tiếp theo

Thời gian tải trang lần thứ hai là thời gian trung bình cho việc di chuyển từ trang này sang trang khác. Trong hoàn cảnh này, bởi vì các đoạn script cần thiết đã được load trong

CSR, thời gian tải là ít hơn với CSR. Nó sẽ không gửi request tới server trừ khi Javascript cần được tải.

Với SSR, một vòng lặp đầy đủ như lần tải đầu được lặp lại. Điều này có nghĩa đã có sự thay đổi lớn của CSR với SSR từ lần tải trang thứ hai.

- *Ảnh hưởng tới caching*

Caching đã trở thành thứ cần thiết ngày nay. Để tăng tốc các ứng dụng web nặng, mọi trình duyệt, cũng như mọi web server đều triển khai các cơ chế caching để tăng tốc ứng dụng. Điều này sẽ cải thiện thời gian tải toàn bộ của CSR cũng như SSR. Tuy nhiên, có một lợi ích mà chỉ có CSR mới có.

Ở CSR, cũng như việc các tải các module là không cần thiết, ứng dụng CSR có thể hoạt động mà không cần tới Internet (trừ khi bạn gửi yêu cầu lấy data). Khi đã được load, ứng dụng không cần thiết gửi các yêu cầu tới server lần nào nữa. Điều này làm cho các ứng dụng web khi được chuyển hướng sẽ giống như một ứng dụng desktop.

Ở SSR, yêu cầu tới server luôn được gửi đi. Do đó thời gian tải trang là cao hơn so với CSR. Việc caching đã cải thiện nội dung render cho SSR cũng như script được lấy ra từ cache.

- *Ảnh hưởng tới SEO*

Với một trang web doanh nghiệp, tối ưu nó cho các công cụ tìm kiếm là điều cần thiết. Các máy tìm kiếm đọc và hiểu trang web của bạn sử dụng các bot tự động gọi là các crawlers. Các crawlers thường quan tâm đến các metadata của trang web hơn là nội dung thực sự. Do đó, điều quan trọng cần chú ý là trang web của bạn cần có các metadata chuẩn SEO cho các máy tìm kiếm.

Với CSR, nội dung trang web được sinh ra tự động nhờ Javascript. Do đó việc thay đổi metadata từ trang này sang trang web sẽ phải tiến hành bằng Javascript. Trong quá khứ, các máy tìm kiếm thường không thích chạy Javascript trong khi crawlers đang quét trang web. Tuy nhiên Google đang chấp thuận chạy Javascript, xu hướng đang thay đổi.

Với CSR, bạn cần tận dụng và tốn công hơn cho việc đảm bảo metadata thay đổi từ trang này sang trang khác. Chúng ta có thể dùng các plugin như React Helmet cho ReactJS để làm điều này.

Với SSR, trang web hoàn chỉnh đã được biên dịch với đúng các metadata và gửi tới trình duyệt. Điều này đảm bảo metadata của trang web luôn luôn chính xác bất kể crawlers

có cho phép Javascript sử dụng hay không. Điều này dẫn tới SSR là cách tiếp cận dễ hơn cho SEO so với CSR.

2.5.5. Cấu trúc NodeJS

➤ Module

- Module giống như các thư viện JavaScript sử dụng trong ứng dụng Node.js application bao gồm một bộ các chức năng. Để đưa một module vào ứng dụng Node.js thì phải sử dụng hàm `require()` function with the với dấu ngoặc đơn chứa tên module.

- Node.js có rất nhiều module cung cấp các chức năng cơ bản cần thiết cho ứng dụng web. Ví dụ như các module trong bảng sau:

Core Modules	Description
http	Includes classes, methods and events to create Node.js http server
util	Includes utility functions useful for developers
fs	Includes events, classes, and methods to deal with file I/O operations
url	Includes methods for URL parsing
querystring	Includes methods to work with query string
stream	Includes methods to handle streaming data
zlib	Includes methods to compress or decompress files

➤ Console

Bảng điều khiển cung cấp phương thức gỡ lỗi tương tự như bảng điều khiển cơ bản của JavaScript trên các trình duyệt internet. Nó sẽ in các thông báo ra stdout và stderr.

➤ Cluster

Node.js được xây dựng dựa trên ý tưởng lập trình đơn luồng. Cluster là một module cho phép đa luồng bằng cách tạo ra các quy trình con có chung cổng máy chủ và chạy đồng thời.

Including cluster module in the application

```
var cluster = require('cluster');  
  
if (cluster.isWorker) {  
  console.log('Child thread');  
} else {  
  console.log('Parent thread');  
  cluster.fork();  
  cluster.fork();  
}
```

Creating child threads by using fork() method

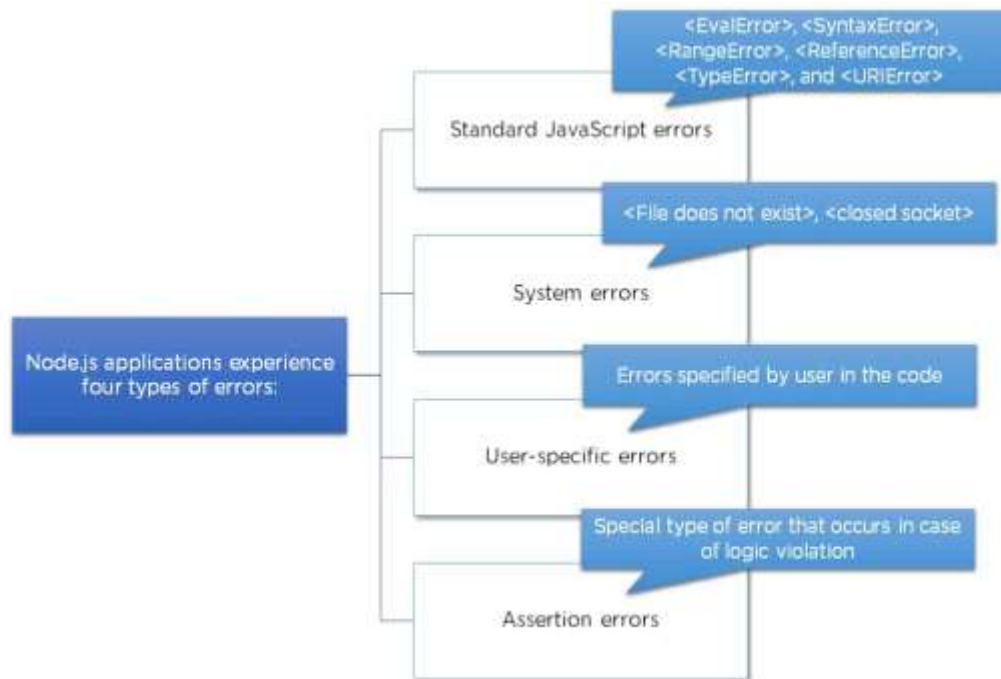
➤ Global

Biến toàn cục trong Node.js tồn tại trong tất cả các module. Những biến này bao gồm các hàm, module, string,... Một số biến toàn cục trong Node.js có thể kể đến trong bảng sau:

Global Objects	Description
__dirname	Specifies the name of the directory that contains the code of application
__filename	Specifies the filename of the code
exports	A reference to the module.exports, shorter to type
module	A reference to the current module
require	Used to import modules, local files, and also JSON

➤ Error Handling

Các ứng dụng Node.js gặp 4 loại lỗi sau.



Lỗi trong Node.js được xử lý qua các exception.

➤ Streaming

Stream là các đối tượng cho phép bạn đọc và viết dữ liệu một cách liên tục. Có 4 loại stream như sau:

- Readable: Là loại stream mà từ đó dữ liệu có thể đọc được.
- Writable: Là loại stream mà dữ liệu có thể được viết lên đó.
- Duplex: Là loại stream có thể đọc và viết được.
- Transform: Là loại stream có thể thao tác dữ liệu trong khi nó đang được đọc hoặc viết.
- Buffer: Bộ đệm là một module cho phép xử lý các stream chỉ chứa các dữ liệu dạng nhị phân. Một bộ đệm trống với độ dài là '10' được tạo ra bởi phương thức sau:

```
var buf = Buffer.alloc(10);
```

➤ Domain

Domain module sẽ chặn các lỗi chưa được xử lý. Hai phương thức được dùng để chặn lỗi đó là:

➤ Internal Binding

Error emitter thực thi code trong phương thức run.

➤ External Binding:

Error emitter được thêm thẳng vào domain qua phương thức add của nó

- *DNS*

○ DNS module được sử dụng để kết nối đến một máy chủ DNS và thực hiện phân giải tên miền sử dụng phương thức sau:

```
dns.resolve()
```

○ DNS module cũng được sử dụng để thực hiện phân giải tên miền mà không cần kết nối mạng bằng phương thức sau:

```
dns.lookup()
```

- *Debugger*

○ Node.js có chức năng gỡ lỗi có thể được sử dụng với một client gỡ lỗi được tích hợp sẵn. Trình gỡ lỗi của Node.js không có quá nhiều tính năng nhưng nó hỗ trợ các chức năng kiểm tra code cơ bản. Trình gỡ lỗi có thể được sử dụng trong bảng lệnh bằng cách sử dụng từ khoá 'inspect' phía trước tên của file JavaScript.

2.5.6. *Node Package Manager*

Khi thảo luận về NodeJS thì một điều chắc chắn không nên bỏ qua là xây dựng package quản lý sử dụng các công cụ NPM mà mặc định với mọi cài đặt NodeJS. Ý tưởng của mô-đun NPM là khá tương tự như Ruby-Gems: một tập hợp các hàm có sẵn để có thể sử dụng được, thành phần tái sử dụng, tập hợp các cài đặt dễ dàng thông qua kho lưu trữ trực tuyến với các phiên bản quản lý khác nhau.

Danh sách các mô-đun có thể tìm trên web NPM package hoặc có thể truy cập bằng cách sử dụng công cụ NPM CLI sẽ tự động cài đặt NodeJS.

Một số các module NPM phổ biến nhất hiện nay là:

- **express:** Express.js, một Sinatra-inspired web framework khá phát triển của NodeJS, chứa rất nhiều các ứng dụng chuẩn của NodeJS ngày nay.

- **connect:** Connect là một mở rộng của HTTP server framework cho NodeJS, cung cấp một bộ sưu tập của hiệu suất cao “plusgin” được biết đến như là trung gian, phục vụ như một nền tảng cơ sở cho Express.

- **socket.io and sockjs:** Hai thành phần Server-side websockets components nổi tiếng nhất hiện nay.

- **Jade:** Một trong những engines mẫu, lấy cảm hứng từ HAML, một phân mặc định trong ExpressJS.

2.5.7. Ưu điểm và nhược điểm của NodeJS

➤ Ưu điểm

- Có tốc độ xử lý nhanh nhờ cơ chế xử lý bất đồng bộ (non-blocking).
- Giúp bạn dễ dàng mở rộng khi có nhu cầu phát triển website.
- Nhận và xử lý nhiều kết nối chỉ với một single-thread. Nhờ đó, hệ thống xử lý sẽ sử dụng ít lượng RAM nhất và giúp quá trình xử NodeJS lý nhanh hơn rất nhiều.
- Có khả năng xử lý nhiều request cùng một lúc trong thời gian ngắn nhất.
- Có khả năng xử lý hàng ngàn process cho hiệu suất đạt mức tối ưu nhất.
- Phù hợp để xây dựng những ứng dụng thời gian thực như các ứng dụng chat, mạng xã hội ...

➤ Nhược điểm

- NodeJS gây hao tốn tài nguyên và thời gian. NodeJS được viết bằng C++ và JavaScript nên khi xử lý cần phải trải qua một quá trình biên dịch. Nếu cần xử lý những ứng dụng tốn tài nguyên CPU thì không nên sử dụng NodeJS.
- NodeJS so với các ngôn ngữ khác như PHP, Ruby và Python sẽ không có sự chênh lệch quá nhiều. NodeJS có thể sẽ phù hợp với việc phát triển ứng dụng mới. Tuy nhiên khi xây dựng và triển khai dự án quan trọng thì NodeJS không phải là sự lựa chọn hoàn hảo nhất.

2.6. Giới thiệu ExpressJS

2.6.1. Khái niệm ExpressJS

Express là một framework dành cho NodeJS. Nó cung cấp cho chúng ta rất nhiều tính năng mạnh mẽ trên nền tảng web cũng như trên các ứng dụng di động. Express hỗ trợ các phương thức HTTP và middleware tạo ra một API vô cùng mạnh mẽ và dễ sử dụng. Có thể tổng hợp một số chức năng chính của express như sau:

- Thiết lập các lớp trung gian để trả về các HTTP request.
- Định nghĩa các router cho phép sử dụng với các hành động khác nhau dựa trên phương thức HTTP và URL.
- Cho phép trả về các trang HTML dựa vào các tham số.

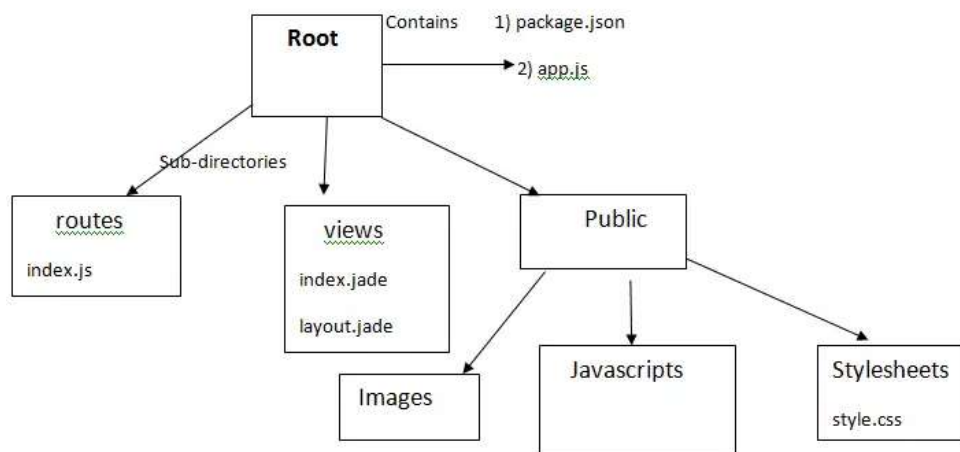
ExpressJS hay còn được viết là Express js, Express.js. Đây là một framework mã nguồn mở miễn phí cho NodeJS. ExpressJS được sử dụng trong thiết kế và xây dựng các ứng dụng web một cách đơn giản và nhanh chóng.

ExpressJS là một framework nhỏ nhưng linh hoạt được xây dựng trên nền tảng của NodeJS. Nó cung cấp tính năng mạnh mẽ để phát triển web hoặc mobile.

Về các package hỗ trợ: ExpressJS có vô số các package hỗ trợ nên các bạn không phải lo lắng khi làm việc với Framework này.

Về performance: Express cung cấp thêm về các tính năng (feature) để lập trình tốt hơn. Chứ không làm giảm tốc độ của NodeJS.

2.6.2. Cấu trúc ExpressJS



- Root:
- “app.js” chứa các thông tin về cấu hình, khai báo, các định nghĩa,... để ứng dụng của chúng ta chạy ok.
- “package.json” chứa các package cho ứng dụng chạy.
- Folder “routes”: chứa các route có trong ứng dụng
- Folder “view”: chứa view/template cho ứng dụng
- Folder “public”: chứa các file css, js, images,...cho ứng dụng

2.6.3. Router trong ExpressJS

➤ Khái niệm

Router là một Object (khác với Routing), nó là một instance riêng của **middleware** và **routes**. Chính vì nó là một **instance** của middleware và route nên nó có các chức năng của cả hai. Chúng ta có thể gọi nó là một mini-application.

Các Application dùng ExpressJS làm core đều có phần Router được tích hợp sẵn trong đó.

Router hoạt động như một middleware nên ta có thể dùng nó như một **arguments**. Hoặc dùng nó như một **arguments** cho route khác.

Ví dụ:

```
// invoked for any requests passed to this router
router.use(function(req, res, next) {
  // .. some logic here .. like any other middleware
  next();
});

// will handle any request that ends in /events
// depends on where the router is "use()"d
router.get('/events', function(req, res, next) {
  // ..
});
```

Chúng ta cũng có thể sử dụng Router để chia route. Chẳng hạn:

```
app.use('/calendar', router);
```

➤ Tìm hiểu router.METHOD()

Router.METHOD() cung cấp cho chúng ta chức năng routing trong ExpressJS. Cụ thể **METHOD()** ở đây là các HTTP method mà chúng ta thường xuyên sử dụng. Chẳng hạn **GET, POST, PUT, PATCH, DELETE,...**

Ví dụ:

```
router.get('/user/profile', function(req, res, next) {
  res.send('user profile');
});
```

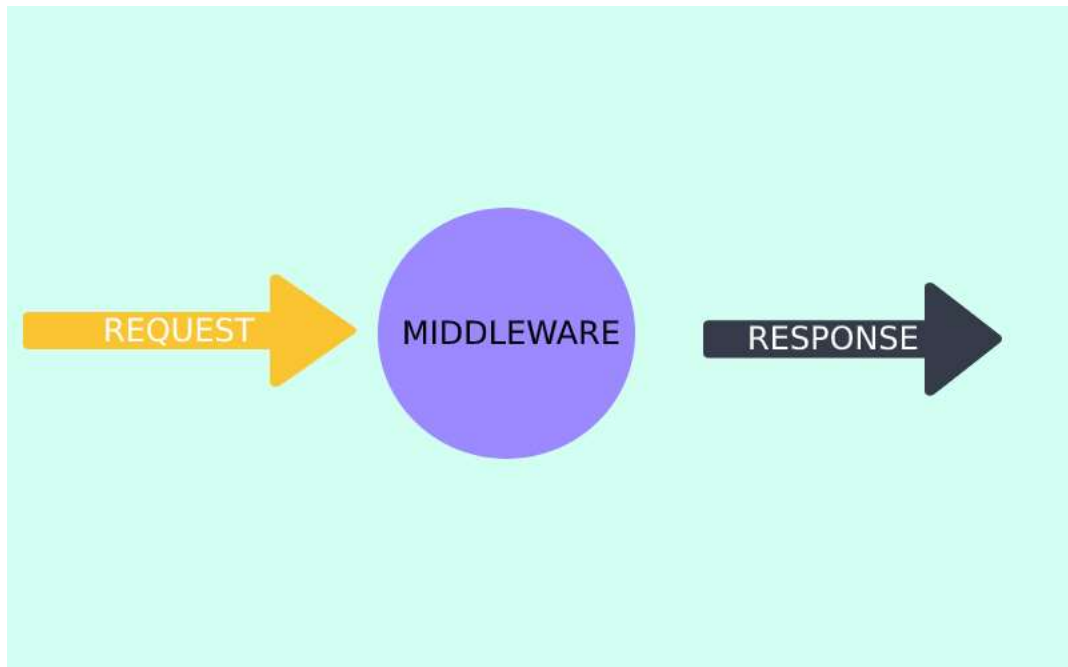
```
router.post('/update/user/:id', function (res, req, next) {
  res.send('Update user');
});
```

```
router.put('/update/posts/:id', function (req, res, next) {  
  res.send('Update post');  
});
```

2.6.4. Middleware trong ExpressJS

Middleware là một phần rất quan trọng trong ExpressJS nó giúp chúng ta có thể lọc và ngăn chặn các request xấu trong quá trình chạy của ứng dụng. Middleware là các hàm khác nhau được gọi bởi [Router Express] trước khi các request hoàn tất.

Các hàm middleware có thể thực thi ở đầu, giữa, hoặc cuối vòng đời của một request. Trong stack các middleware function luôn được thêm vào theo thứ tự mà chúng ta mong muốn ngay ban đầu.



Middleware thường sử dụng để thực hiện các tác vụ như: `parseJSON`, `parseCookie`, kiểm tra đăng nhập, thậm chí còn được sử dụng để viết các module một cách nhanh chóng.

➤ Middleware Function trong ExpressJS

Các **middleware function** là các hàm có quyền truy cập vào các đối tượng như *request* (*req*), *response* (*res*), *next()* trong chu kỳ của ứng dụng. **next()** là một hàm trong Route Express, khi được gọi hàm *next()* sẽ thực thi các middleware tiếp theo. Các middle function có thể thực hiện các chức năng như:

- Thay đổi các đối tượng như *request* (*req*), *response* (*res*).
- Kết thúc vòng đời của một request.

- Gọi các middleware trong stack.

Ví dụ, để khởi tạo một middleware function chúng ta sử dụng:

```
1 //middleware function
2 var myLogger = function(req, res, next) {
3   console.log("Middleware")
4   next()
5 }
6 //Khai báo sử dụng middleware
7 app.use(myLogger)
```

Khi khởi tạo *middleware* bằng cách này thì bất cứ các request nào được gửi đến sẽ phải thông qua *middleware*.

2.6.5. JSON Web Token (JWT)

JWT là viết tắt của JSON Web Token là 1 tiêu chuẩn mở (RFC 7519) định nghĩa cách thức truyền tin an toàn giữa các thành viên bằng 1 đối tượng JSON, nó an toàn vì nó có chứa một chữ ký số. Cấu trúc của một JWT bao gồm 3 phần:

- **Header:** Header bao gồm hai phần chính: loại token (mặc định là JWT – Thông tin này cho biết đây là một Token JWT) và thuật toán đã dùng để mã hóa (HMAC SHA256 – HS256 hoặc RSA).
- **Payload:** Phần payload sẽ chứa các thông tin mình muốn đặt trong chuỗi Token như username, userId, author,...
- **Signature:** Chữ ký Signature trong JWT là một chuỗi được mã hóa bởi header, payload cùng với một chuỗi bí mật gọi là secret key nó được bảo mật ở phía server.
- Dưới đây là 1 JSON Web Token, các thành phần **Header**, **Payload**, **Signature** tương ứng cách nhau bởi dấu “.”

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

➤ Tại sao cần sử dụng JWT

Khi user truy cập và login vào trang web thành công thì phía server sẽ tạo và lưu một session chứa thông tin của người dùng đang đăng nhập và trả lại cho client session ID để truy cập cho những request sau. Từ những request tiếp theo thì user sẽ gửi kèm theo session id đó, server sẽ tìm kiếm session id đó trong session log để biết được user đang đăng nhập là user nào. Phương thức này hoạt động tốt khi có một server duy nhất, vì thế khi những ứng dụng web sử dụng nhiều server thì nó sẽ có một số hạn chế.

Khi user thực hiện đăng nhập và được định hướng đến server 1 thì thông tin đăng nhập của user sẽ được lưu trữ ở server 1. Khi người dùng gửi request tiếp theo và được định hướng đến server 2 thì ở server 2 lúc này sẽ không có thông tin của user vì thông tin đó đang lưu ở server 1. Thì đó JWT sẽ giúp để giải quyết được vấn đề này thay vì lưu trữ thông tin ở server và trả về một session id thì bây giờ chúng ta sẽ trả về thông tin người dùng dưới dạng token, khi request được gửi đi thì token đó cũng được gửi đi theo và được xác thực ở phía server. Khi đó những server có chung secret key thì đều có thể dùng secret key để verify request từ client gửi lên.

2.7. Giới thiệu MongoDB & mongoose

2.7.1. Khái niệm MongoDB

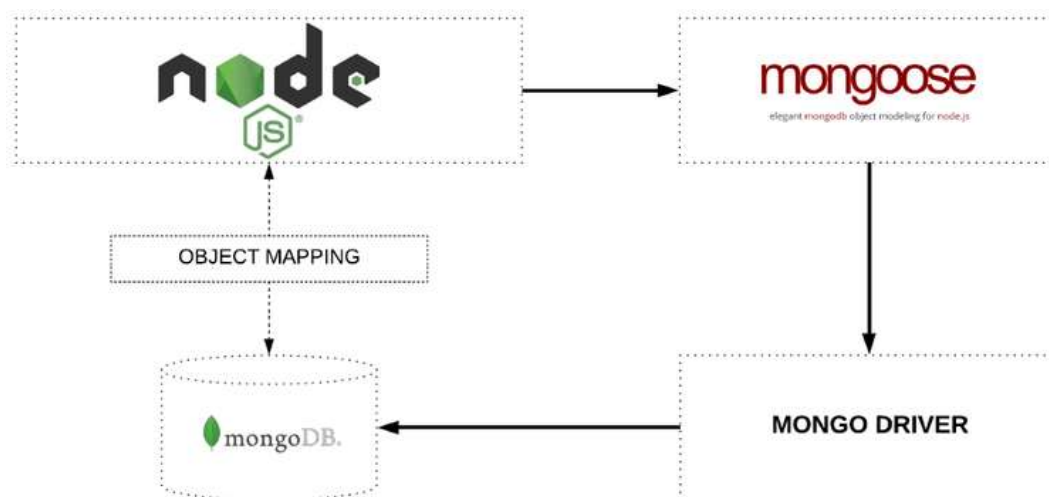
MongoDB là một cơ sở dữ liệu mã nguồn mở và là cơ sở dữ liệu NoSQL hàng đầu, được hàng triệu người sử dụng, MongoDB được viết bằng C++.

MongoDB còn là một cơ sở dữ liệu đa nền tảng, hoạt động trên các khái niệm Collection và Document, nó cung cấp hiệu suất cao, tính khả dụng cao và khả năng mở rộng dễ dàng.

Vì MongoDB là một cơ sở dữ liệu NoSQL nên có thể lưu trữ các JSON trong đó, cấu trúc của các tài liệu này có thể thay đổi vì nó không bắt buộc như các cơ sở dữ liệu SQL. Đây là một trong những lợi thế của việc sử dụng NoSQL vì nó tăng tốc độ phát triển ứng dụng và giảm sự phức tạp của việc triển khai.

2.7.2. Khái niệm mongoose

Mongoose là một thư viện mô hình hoá đối tượng (Object Data Model – ODM) cho MongoDB và NodeJS. Nó quản lý mối quan hệ giữa dữ liệu, cung cấp sự xác nhận giản đồ và được sử dụng để dịch giữa các đối tượng trong mã và biểu diễn các đối tượng trong MongoDB.



2.7.3. So sánh Relational database management system và MongoDB

Các mô hình dữ liệu khác nhau có thể cho phép tạo ra tính linh hoạt cao hơn nhiều so với cấu trúc cứng nhắc do cơ sở dữ liệu quan hệ (RDBMS) áp đặt. Do tính linh hoạt của chúng, cơ sở dữ liệu NoSQL nổi tiếng là lựa chọn tốt hơn để lưu trữ dữ liệu bản cấu trúc và phi cấu trúc, trái ngược với dữ liệu có cấu trúc chuẩn hoá theo yêu cầu của cơ sở dữ liệu quan hệ. Vì cơ sở dữ liệu NoSQL của MongoDB không đi kèm với một lược đồ xác định trước, nên thường tùy thuộc vào các nhà phát triển hoặc quản trị viên cơ sở dữ liệu để xác định cách dữ liệu nên được tổ chức và truy cập sao cho phù hợp nhất với ứng dụng của họ.

RDBMS	MongoDB
<ul style="list-style-type: none"> Dữ liệu có cấu trúc và tổ chức Sử dụng ngôn ngữ SQL để truy vấn dữ liệu Dữ liệu và các mối quan hệ của nó được lưu trữ trong các bảng riêng biệt Có tính chặt chẽ 	<ul style="list-style-type: none"> Không sử dụng SQL Không khai báo ngôn ngữ truy vấn dữ liệu Không định nghĩa schema Có 1 số nhóm dạng: Key-Value Storage, Column Store, Document Store, Graph Databases Dữ liệu phi cấu trúc và không thể đoán trước Ưu tiên cho hiệu năng cao, tính sẵn sàng cao và khả năng mở rộng

2.7.4. Kiểu dữ liệu của MongoDB

Type	Number	Alias
Double	1	“double”
String	2	“string”
Object	3	“object”
Array	4	“array”
Binary data	5	“binData”
Undefined	6	“undefined”
ObjectId	7	“objectId”
Boolean	8	“bool”
Date	9	“date”
Null	10	“null”
Regular Expression	11	“regex”
DBPointer	12	“dbPointer”
JavaScript	13	“javascript”
Symbol	14	“symbol”
JavaScript (with scope)	15	“javascriptWithScope”
32-bit interger	16	“int”
Timestamp	17	“timestamp”
64-bit Interger	18	“long”
Decimal128	19	“decimal”
Min key	-1	“minkey”
Max key	127	“maxkey”

- **Chuỗi:** Đây là kiểu dữ liệu được sử dụng phổ biến nhất để lưu giữ dữ liệu. Chuỗi trong MongoDB phải là UTF-8 hợp lệ.
- **Số nguyên:** Kiểu dữ liệu này được sử dụng để lưu một giá trị số. Số nguyên có thể là 32 bit hoặc 64 bit phụ thuộc vào Server của bạn.
- **Boolean:** Kiểu dữ liệu này được sử dụng để lưu giữ một giá trị Boolean (true/false).
- **Double:** Kiểu dữ liệu này được sử dụng để lưu các giá trị số thực dấu chấm động.
- **Min/Max keys:** Kiểu dữ liệu này được sử dụng để so sánh một giá trị với các phần tử BSON thấp nhất và cao nhất.

- **Mảng:** Kiểu dữ liệu này được sử dụng để lưu giữ các mảng hoặc danh sách hoặc nhiều giá trị vào trong một key.
- **Timestamp:** Đánh dấu thời điểm một Document được sửa đổi hoặc được thêm vào.
- **Object:** Kiểu dữ liệu này được sử dụng cho các Document được nhúng vào.
- **Null:** Kiểu dữ liệu này được sử dụng để lưu một giá trị Null.
- **Symbol:** Kiểu dữ liệu này được sử dụng giống như một chuỗi.
- **Date:** Kiểu dữ liệu này được sử dụng để lưu giữ date và time hiện tại trong định dạng UNIX time.
- **Object ID:** Kiểu dữ liệu này được sử dụng để lưu giữ ID của Document.
- **Binary data:** Kiểu dữ liệu này được sử dụng để lưu giữ dữ liệu nhị phân.
- **Code:** Kiểu dữ liệu này được sử dụng để lưu giữ JavaScript code vào trong Document.
- **Regular expression:** Kiểu dữ liệu này được sử dụng để lưu giữ Regular Expression.

2.7.5. Cách thức hoạt động

MongoDB hoạt động dưới một tiến trình ngầm service, luôn mở một cổng (mặc định là 27017) để lắng nghe các yêu cầu truy vấn, thao tác từ các ứng dụng gửi vào sau đó mới tiến hành xử lý.

Mỗi một bản ghi của MongoDB được tự động gắn thêm một field có tên “_id” thuộc kiểu dữ liệu ObjectId mà nó quy định để xác định được tính duy nhất của bản ghi này so với bản ghi khác, cũng như phục vụ các thao tác tìm kiếm và truy vấn thông tin về sau. Trường dữ liệu “_id” luôn được tự động đánh chỉ mục để tốc độ truy vấn thông tin đạt hiệu suất cao nhất.

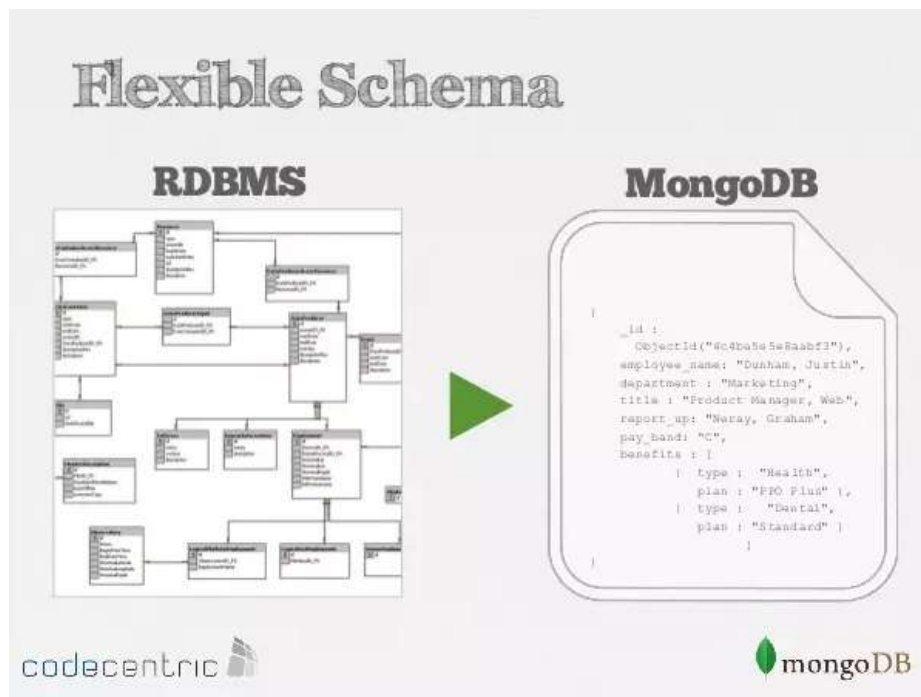
Mỗi khi có một truy vấn dữ liệu, bản ghi được cache (ghi đệm) lên bộ nhớ Ram, để phục vụ lượt truy vấn sau diễn ra nhanh hơn mà không cần phải đọc từ ổ cứng.

Khi có yêu cầu thêm/xóa/sửa bản ghi, để đảm bảo hiệu suất của ứng dụng mặc định MongoDB sẽ chưa cập nhập xuống ổ cứng ngay mà sau 60 giây MongoDB mới thực hiện ghi toàn bộ dữ liệu thay đổi từ RAM xuống ổ cứng.

2.7.6. Ưu điểm và nhược điểm của MongoDB

➤ Ưu điểm

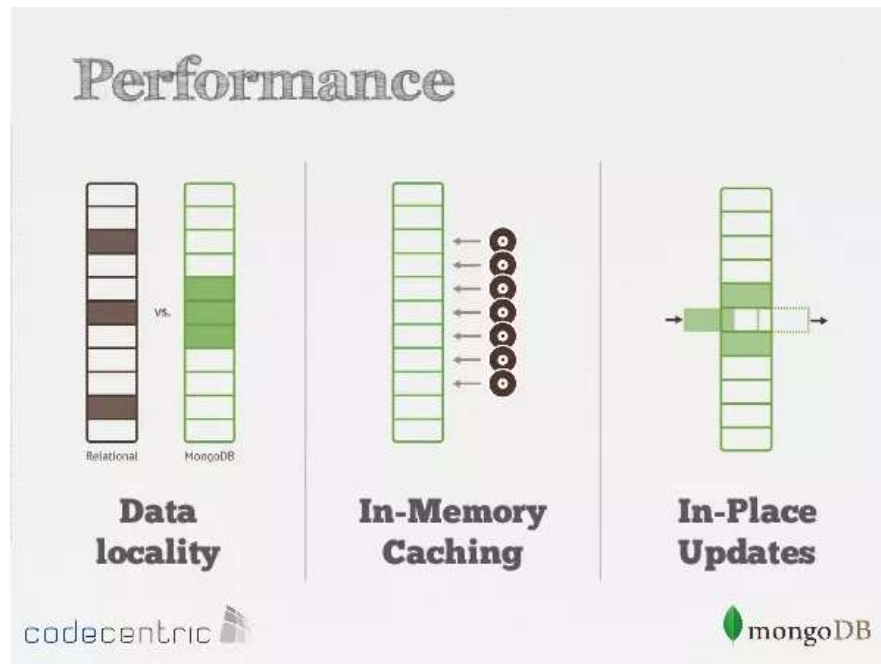
- Ít schema hơn vì schema được sinh ra là để nhóm các đối tượng vào 1 cụm, dễ quản lý.



- Cấu trúc của một đối tượng rõ ràng.
- Không có các join phức tạp.
- Khả năng mở rộng cực lớn: việc mở rộng dữ liệu mà không phải lo đến các vấn đề như khoá ngoại, khoá chính, kiểm tra ràng buộc,... MongoDB cho phép thực hiện replication và sharding nên việc mở rộng cũng thuận lợi hơn.



- Sử dụng bộ nhớ trong để lưu giữ của số làm việc cho phép truy cập dữ liệu nhanh hơn. Việc cập nhật được thực hiện nhanh gọn nhờ update tại chỗ.



➤ **Nhược điểm**

- Dữ liệu được caching, lấy RAM làm trọng tâm hoạt động vì vậy khi hoạt động yêu cầu một bộ nhớ RAM lớn.
- Như cách thức hoạt động đã nói, mọi thay đổi về dữ liệu mặc định đều chưa được ghi xuống ổ cứng ngay lập tức vì vậy khả năng bị mất dữ liệu từ nguyên nhân mất điện đột xuất là rất cao.

2.8. Giới thiệu SocketIO

2.8.1. Khái niệm về SocketIO

Socket.io là một framework của Node.js cho phép xây dựng các ứng dụng web chạy thời gian thực (realtime). Là thư viện Javascript, nó cho phép giao tiếp theo hai hướng giữa client và server. Socket.io hoạt động như một thư viện phía client đang chạy trong trình duyệt và như là một thư viện phía server cho node.js. Framework cho phép thực hiện đồng thời thời gian thực cho việc cộng tác và trao đổi dữ liệu. Hơn nữa, các tính năng chính của nó bao gồm xử lý I/O không đồng bộ, luồng nhị phân, nhắn tin tức thời và hơn thế nữa.

2.8.2. Cơ chế hoạt động của SocketIO

➤ **Khai báo sử dụng SocketIO**

Cơ chế hoạt động của một ứng dụng realtime đó là thông qua server để lắng nghe (listen) data và truyền data về các máy client. Vì vậy cần cài khai báo sử dụng socketio ở cả phía server và client.

Code khai báo sử dụng SocketIO ở server:

```
// build server, khai báo sử dụng socket io
var express = require("express");
var app = express();
app.use(express.static("public"));
app.set("view engine", "ejs");
app.set("views", "./views");

var server = require("http").Server(app);
var io = require("socket.io")(server);
server.listen(3000);
```

Code khai báo sử dụng SocketIO ở phía client:

```
<html>
  <head>
    <title>Demo Socketio - Homepage</title>
    <script src="jquery.js"></script>
    <script src="socket.io/socket.io.js"></script>

    <script>
      var socket = io("http://localhost:3000");
    </script>
  </head>

  <body>
  </body>
</html>
```

➤ Cơ chế lắng nghe, truyền dữ liệu của SocketIO

Để lắng nghe data, chúng ta sử dụng câu lệnh `socket.on()`, để phát dữ liệu thì sử dụng lệnh `socket.emit()`.

Ví dụ, client gửi 1 đoạn chat đi, thì khi đó ở phía server cần viết code để nhận dữ liệu đoạn code đó và truyền dữ liệu chat đó đi đến các server khác. Đồng thời ở phía client cũng cần viết code để gửi và nhận dữ liệu từ server.

Code phía server:

```

api > src > ts socket.ts > socket > io.on() callback > socket.on() callback
46 function socket({ io }: { io: Server }) {
47   io.on(ev: EVENTS.connection, listener: (socket: Socket) => {
48     socket.on(ev: EVENTS.CLIENT.ADD_USER, listener: (userId: string) => {
49       addUser(userId, socketId: socket.id);
50       io.emit(ev: EVENTS.SERVER.GET_USERS, ...args: users);
51     });
52     // server received message
53     socket.on(
54       ev: EVENTS.CLIENT.SEND_MESSAGE,
55       listener: ({
56         message,
57         receiverId,
58       }: {
59         message: IMessageResponse;
60         receiverId: string;
61       }) => {
62         // You, 5 days ago + update socket up!
63         console.log(message: { message, receiverId });
64         const receiveUsers = getUsers(userId: receiverId);
65         const senders = getUsers(userId: message.sender._id);
66         const socketIdReceivers = receiveUsers.map(
67           callbackfn: (socketUser) => socketUser.socketId
68         );
69         const socketIdsSenders = senders.map(
70           callbackfn: (socketUser) => socketUser.socketId
71         );
72         io.to(room: [...socketIdReceivers, ...socketIdsSenders]).emit(
73           ev: EVENTS.SERVER.GET_MESSAGE,
74           ...args: {
75             message,
76             receiverId,
77           }
78         );
79       }
80     );
81   });
82 }

```

Code phía client:

```

useEffect(effect: () => {
  socket.on(ev: config.socketEvents.SERVER.GET_MESSAGE, listener: ({ message }: { message: IMessage }) => {
    message.sender._id !== user?._id && setReceiver(value: message.sender);
    setMessages(value: (prev) => {
      const lastPrevMessage = prev[prev.length - 1];
      if (
        lastPrevMessage &&
        // user?._id !== message.sender._id &&
        lastPrevMessage.conversation === message.conversation &&
        lastPrevMessage._id !== message._id
      ) {
        // You, 5 days ago + update chatbox
        return [...prev, message];
      }
      return prev;
    });
  });
}, deps: []);

```

```

const message: IMessage = await messageApi.create(body: createMessage);
if (message) {
  // setMessages((prev) => [...prev, message]);
  socket.emit(ev: config.socketEvents.CLIENT.SEND_MESSAGE, ...args: {
    message,
    receiverId: receiver?._id,
  });
  setNewMessage(value: '');
  setFileImages(value: []);
  messageInputRef.current?.focus();
}

```


socket.on và socket.emit có parameter thứ nhất là tên đường truyền. Tên đường truyền có thể là chuỗi bất kỳ, tuy nhiên để truyền và nhận dữ liệu của chung 1 đường truyền thì tên đường truyền phải giống nhau.

Ví dụ: client gửi data bằng đường truyền có tên là “sendMessage”, thì để nhận được data của đường truyền này thì server cũng phải báo tên đường truyền là “sendMessage” trong lệnh socket.on.

2.8.3. Điểm nổi bật của SocketIO

➤ Khả năng bảo mật

Ngay khi Socket.io xuất hiện thì nó có thể tự động tạo ra những kết nối dạng bảo mật như:

- Thực hiện Proxy và cân bằng cho tải.
- Tạo tường lửa cá nhân và các phần mềm chống lại virus.

Socket.io thực hiện xây dựng dựa vào Engine.IO như sau: nó sẽ khởi chạy theo phương thức long-polling để tự động kết nối, rồi dùng những phương thức giao tiếp khác để giao tiếp được tốt hơn.

➤ Kết nối lại tự động và phát hiện tình trạng ngắt kết nối

Khi chạy, nếu như client bị ngắt kết nối thì nó có thể kết nối tự động cho đến khi server xác nhận đã được phản hồi. Đây là tính năng cho phép thiết bị có thể tự động kết nối lại mãi mãi cho đến khi server phản hồi. Socket.io còn cung cấp cho bạn những event có thể phát hiện ngắt kết nối giữa client và server.

➤ Hỗ trợ nhị phân

Socket.io hỗ trợ bạn một số kiểu mã nhị phân:

- ArrayBuffer cùng Blob nằm trên trình duyệt.
- ArrayBuffer cùng Buffer nằm trong Node.js.

➤ Hỗ trợ tạo phòng và kênh

Socket.io cho phép người dùng tạo các kênh riêng biệt, từ đó tạo ra những mối quan hệ riêng giữa các phần tương tự như module riêng lẻ hoặc dựa vào một số quyền khác nhau. Ngoài ra, nó còn hỗ trợ chúng ta tạo các phòng khác nhau cùng những clients được tham gia vào những phòng khác.

2.9. API (Application Program Interface)

2.9.1. Khái niệm về API

API là viết tắt của Application Programming Interface – phương thức trung gian kết nối các ứng dụng và thư viện với nhau. API cung cấp khả năng truy xuất đến một tập hợp các hàm hay dùng, từ đó có thể trao đổi dữ liệu giữa các ứng dụng.

API là cơ chế cho phép 2 thành phần phần mềm giao tiếp với nhau bằng một tập hợp các các định nghĩa và giao thức.

Kiến trúc API thường được giải thích dưới dạng máy chủ và máy khách. Ứng dụng gửi yêu cầu được gọi là máy khách, còn ứng dụng gửi phản hồi được gọi là máy chủ.

Các đặc điểm nổi bật của API:

- API sử dụng mã nguồn mở, dùng được với mọi client hỗ trợ: XML, JSON/
- API có khả năng đáp ứng đầy đủ các thành phần HTTP: URL, request/response headers, caching, versioning, content format,...
- Mô hình web API dùng để hỗ trợ MVC như: unit test, injection, ioc container, model binder, action result, filter, routing, controller. Ngoài ra, nó cũng hỗ trợ RESTful đầy đủ các phương thức như GET, POST, PUT, DELETE dữ liệu.
- Được đánh giá là một trong những kiểu kiến trúc hỗ trợ tốt nhất cho các thiết bị có lượng băng thông bị giới hạn như smartphone, tablet,...

2.9.2. WebAPI

Web API là một phương thức dùng để cho phép các ứng dụng khác nhau có thể giao tiếp, trao đổi dữ liệu qua lại. Dữ liệu được Web API trả lại thường ở dạng JSON hoặc XML thông qua giao thức HTTP hoặc HTTPS.

2.9.3. RESTfulAPI

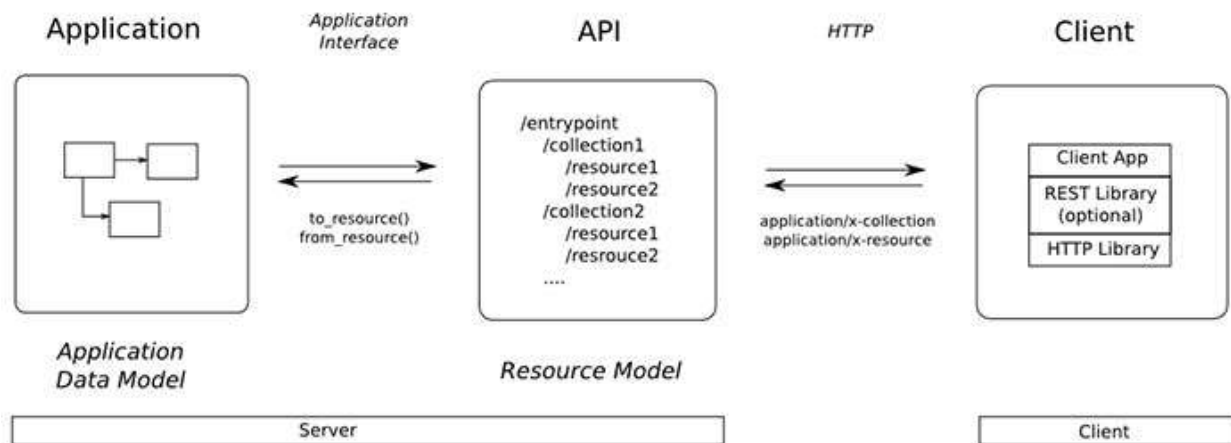
➤ **RESTful API là gì ?**

RESTFUL API (hay REST API) là một giao diện lập trình ứng dụng (API hay web API) tuân theo các ràng buộc của kiểu kiến trúc REST, cho phép tương tác với các dịch vụ web RESTful. Hay nói đơn giản, RESTful API là một tiêu chuẩn được dùng trong việc thiết kế API dành cho các ứng dụng web (thiết kế Web Services) để hỗ trợ cho việc quản lý các resource. REST là viết tắt của cụm từ Representational State Transfer (ứng dụng chuyển đổi cấu trúc dữ liệu), được tạo ra lần đầu bởi nhà khoa học máy tính Roy Fielding ở những năm 2000.

➤ **Cách thức hoạt động của RESTful API**

RESTful API chia một transaction (giao dịch) ra thành nhiều module nhỏ, mỗi module giải quyết một phần cơ bản của transaction. Việc này giúp tăng tính linh hoạt nhưng đòi

khi lại tương đối khó khăn cho các developer khi muốn thiết kế REST API từ đầu. Hiện tại có khá nhiều công ty cung cấp model cho các developer sử dụng, trong đó phổ biến nhất gồm có Amazon S3, CDMI hay OpenStack Swift.



Một RESTful API sử dụng các câu lệnh để lấy tài nguyên, trạng thái của tài nguyên ở bất kỳ timestamp nào được gọi là một biểu diễn của tài nguyên đó. Các phương thức HTTP mà RESTful API sử dụng được xác định bởi giao thức RFC 2616:

- **GET:** Trả về một tài nguyên.
- **PUT:** Thay đổi trạng thái hoặc cập nhật tài nguyên (có thể là đối tượng, file hay block).

- **POST:** Tạo tài nguyên.
- **DELETE:** Xóa một tài nguyên.

Một số định dạng dữ liệu được RESTful API hỗ trợ:

- application/json.
- application/xml.
- application/x-wbx+xml.
- application/x-www-form-urlencoded.
- multipart/form-data.

➤ Status code

Khi request một API, có một số status code thường gặp như sau:

- 200 OK: Trả về thành công cho những phương thức GET, PUT, PATCH hoặc DELETE.

- 201 Created: Trả về khi tạo thành công một Resource.
- 204 No Content: Trả về khi xóa thành công Resource.

- 304 Not Modified: Client có thể sử dụng dữ liệu cache.
- 400 Bad Request: Request không hợp lệ.
- 401 Unauthorized: Request cần phải có xác thực.
- 403 Forbidden: Bị từ chối không cho phép.
- 404 Not Found: Không tìm thấy resource từ URL.
- 405 Method Not Allowed: Phương thức không cho phép với user hiện tại.
- 410 Gone: Resource không còn tồn tại, version cũ không còn hỗ trợ.
- 415 Unsupported Media Type: Không hỗ trợ kiểu Resource này.
- 422 Unprocessable Entity: Dữ liệu không được xác thực.
- 429 Too Many Requests: Request bị từ chối do bị giới hạn.

➤ **Ưu điểm của RESTful API**

- Dễ hiểu, dễ học, đơn giản.
- Cho phép tổ chức các ứng dụng phức tạp, dễ dàng sử dụng tài nguyên.
- Quản lý tải cao nhờ HTTP proxy server và cache.
- Các client mới có thể dễ dàng làm việc trên những ứng dụng khác.
- Cho phép sử dụng các lệnh gọi thủ tục HTTP tiêu chuẩn để truy xuất dữ liệu và request.

- RESTful API dựa trên code và có thể sử dụng nó để đồng bộ hoá dữ liệu bằng website.

- Cung cấp các định dạng linh hoạt bằng cách tuần tự hoá (serialize) dữ liệu ở dạng XML hay JSON.

- Cho phép sử dụng các giao thức OAuth để xác thực request REST.

➤ **Nhược điểm của RESTful API**

- **Không có trạng thái:** Hầu hết các ứng dụng web đều yêu cầu cơ chế stateful (có trạng thái). Giả sử ta cần mua một website có cơ chế giỏ hàng, khi đó ta cần biết số lượng sản phẩm có trong giỏ hàng trước khi thực hiện thanh toán. Việc duy trì trạng thái này là nhiệm vụ của phía client, do đó ứng dụng client có thể cồng kềnh và khó bảo trì hơn.

- **Bảo mật:** REST có thể phù hợp với các URL public, nhưng không phải là một lựa chọn tốt nếu cần truyền dữ liệu nhạy cảm giữa client và server.

2.10. Giới thiệu Axios

2.10.1. Khái niệm về Axios

Axios là một thư viện HTTP Client dựa trên Promise dùng để hỗ trợ cho việc xây dựng các ứng dụng API từ đơn giản đến phức tạp và có thể được sử dụng cả ở trình duyệt hay NodeJS. Ở phía server thì nó sử dụng native module HTTP trong NodeJS, còn ở phía client (trình duyệt) thì nó sử dụng XMLHttpRequest.

2.10.2. Các tính năng của Axios

- Tạo request từ trình duyệt bằng XMLHttpRequest.
- Tạo request từ node.js bằng http.
- Hỗ trợ Promise API.
- Đón chặn request và response.
- Biến đổi dữ liệu request và response.
- Bỏ request.
- Tự động chuyển đổi cho dữ liệu JSON.
- Hỗ trợ phía client bảo vệ chống lại XSRF.

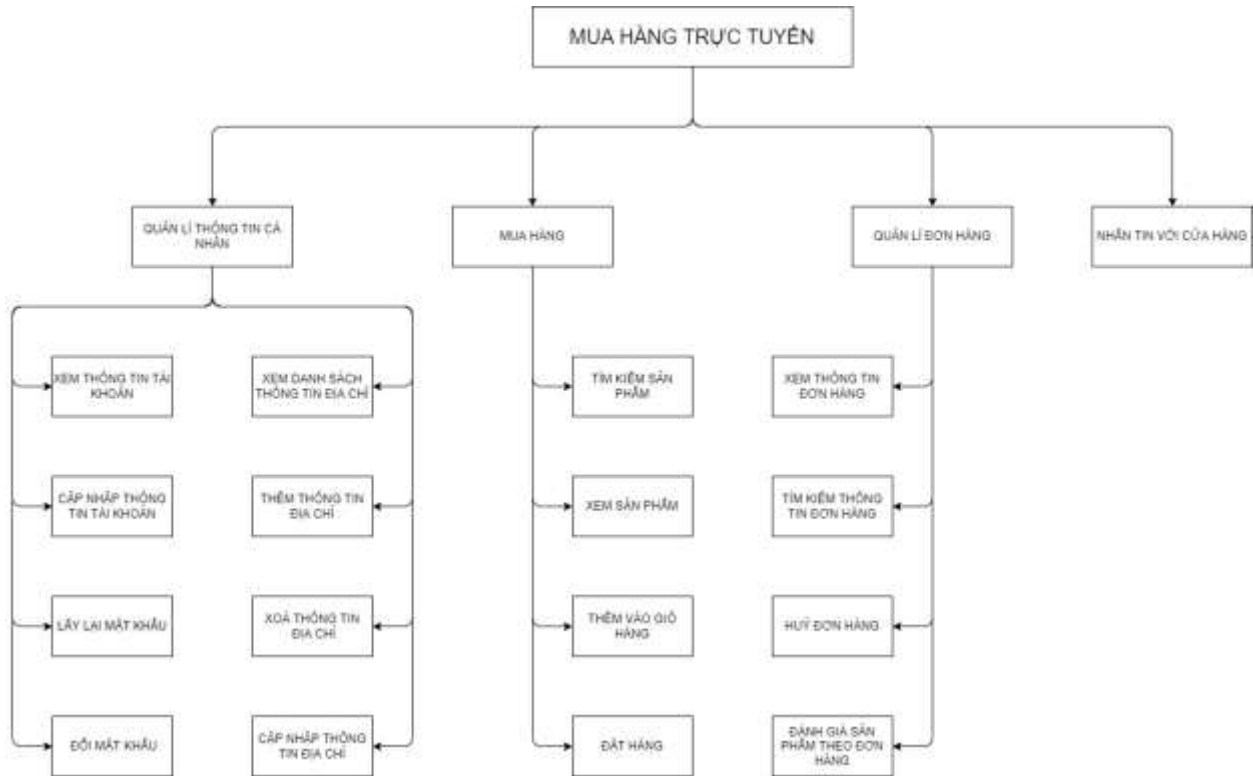
2.10.3. Ưu điểm của Axios

Axios xây dựng dựa trên nền tảng Promise do đó nó kế thừa các ưu điểm của Promise. Cho phép thực hiện các hook (intercept) ngay khi gửi request và nhận response. Cho phép hủy yêu cầu, đây là một chức năng mà các thư viện khác không có. Axios là HTTP Client giúp xây dựng các ứng dụng kết nối từ nhiều nguồn dữ liệu. Axios là phần công cụ giúp lấy dữ liệu dễ dàng cho các framework như Vue.js, **React.js**, Angular... xây dựng các ứng dụng front-end linh động.

CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

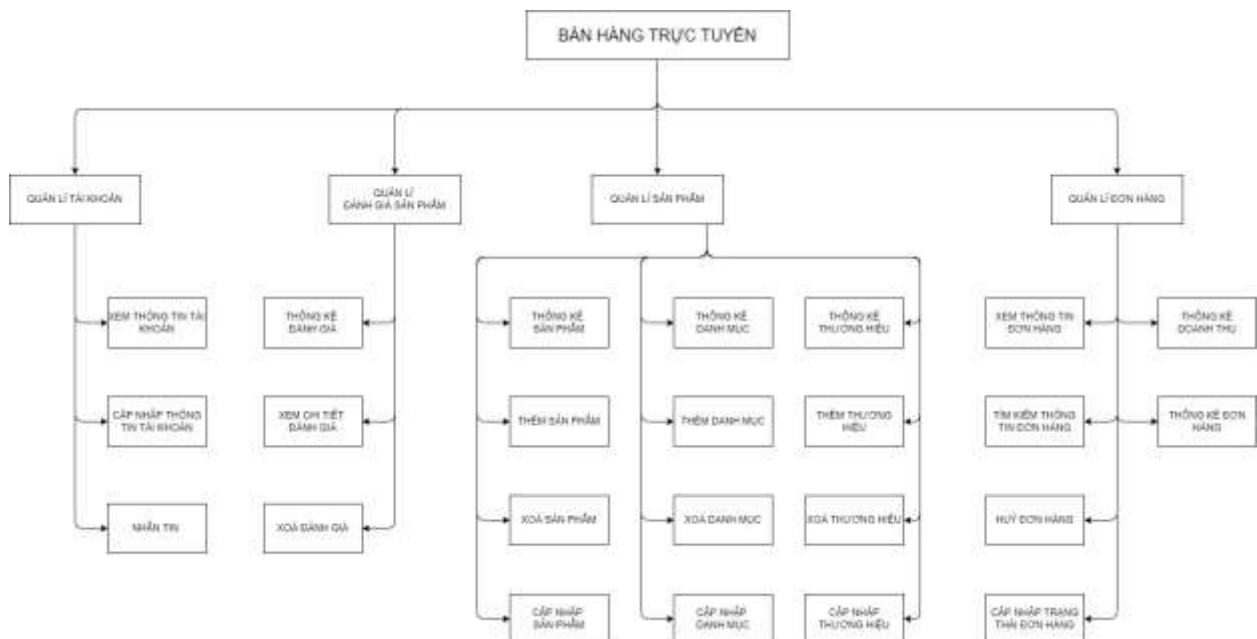
3.1. Biểu đồ phân cấp chức năng

3.1.1. Biểu đồ phân cấp chức năng mua hàng



Hình 3.1.1. Biểu đồ phân cấp chức năng mua hàng cho khách hàng

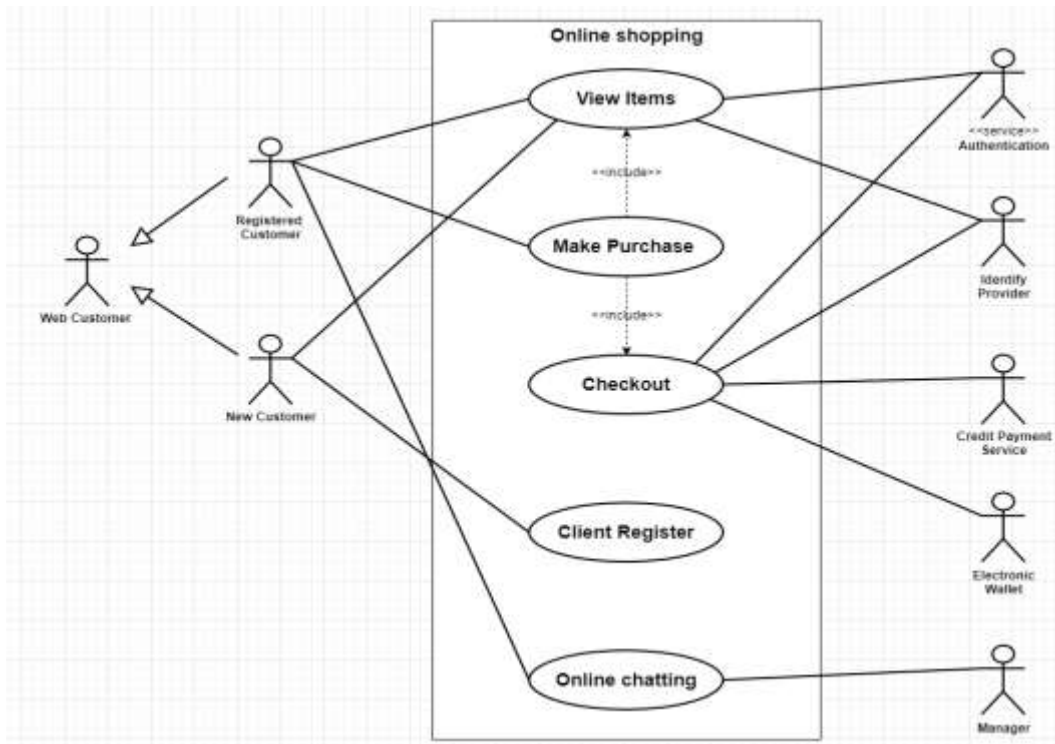
3.1.2. Biểu đồ phân cấp chức năng bán hàng



Hình 3.1.2. Biểu đồ phân cấp chức năng bán hàng của quản trị viên

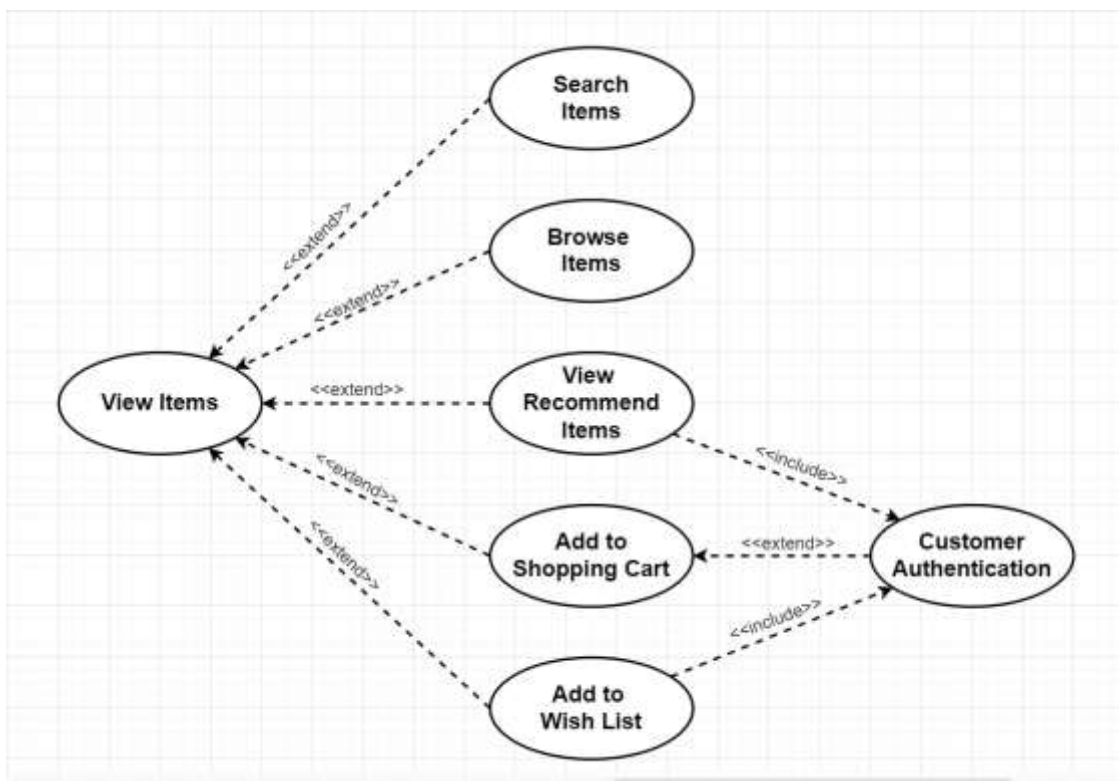
3.2. Biểu đồ Use-Case

- Biểu đồ Use-Case tổng quan



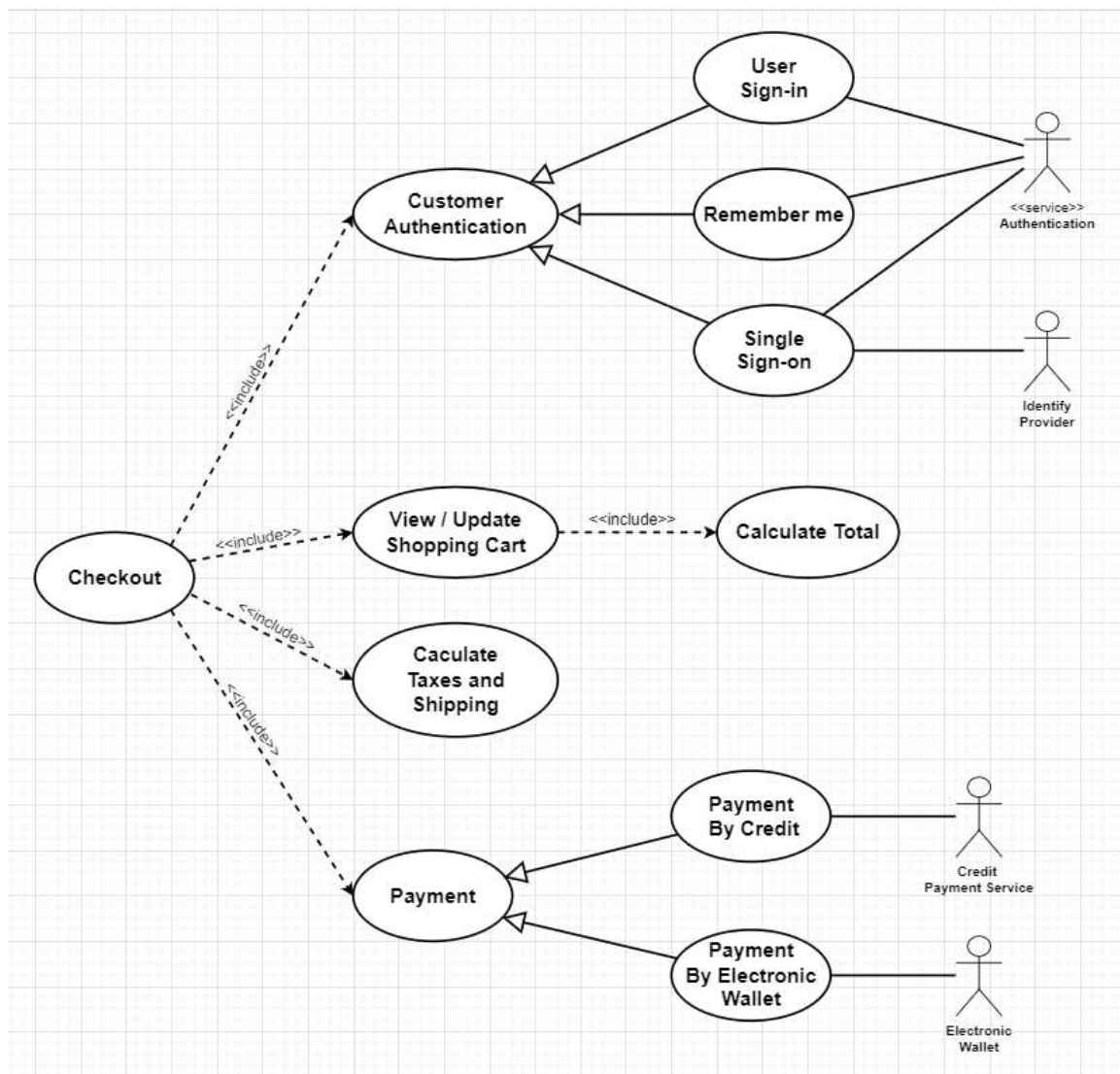
Hình 3.1. Biểu đồ use-case tổng hợp

- Biểu đồ Use-Case mở rộng của “View Items”



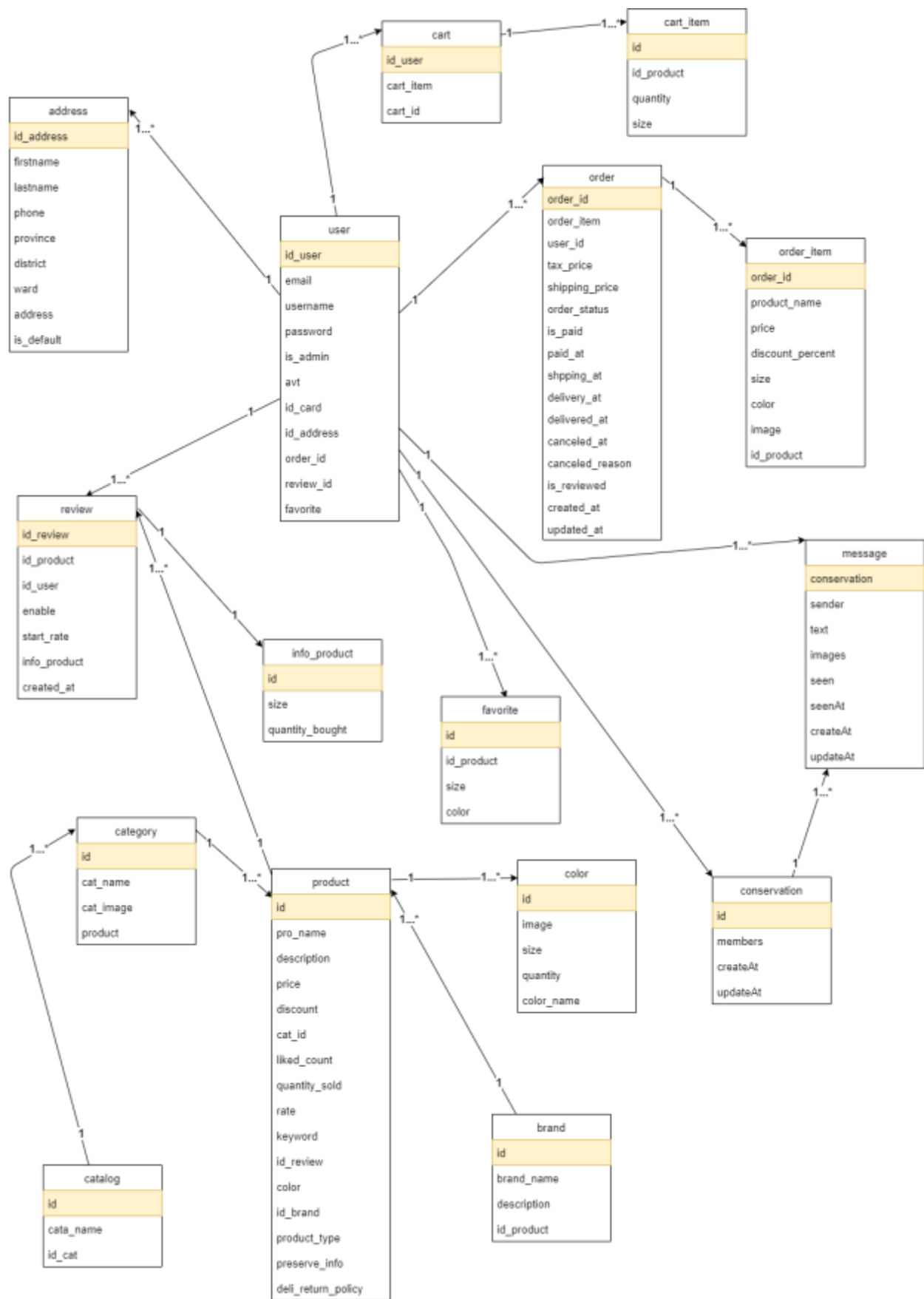
Hình 3.2. Biểu đồ Use-Case “View Items”

- Biểu đồ Use-Case mở rộng của “Checkout”



Hình 3.3. Biểu đồ Use-Case “Checkout”

3.3. Biểu đồ quan hệ thực thể ERD

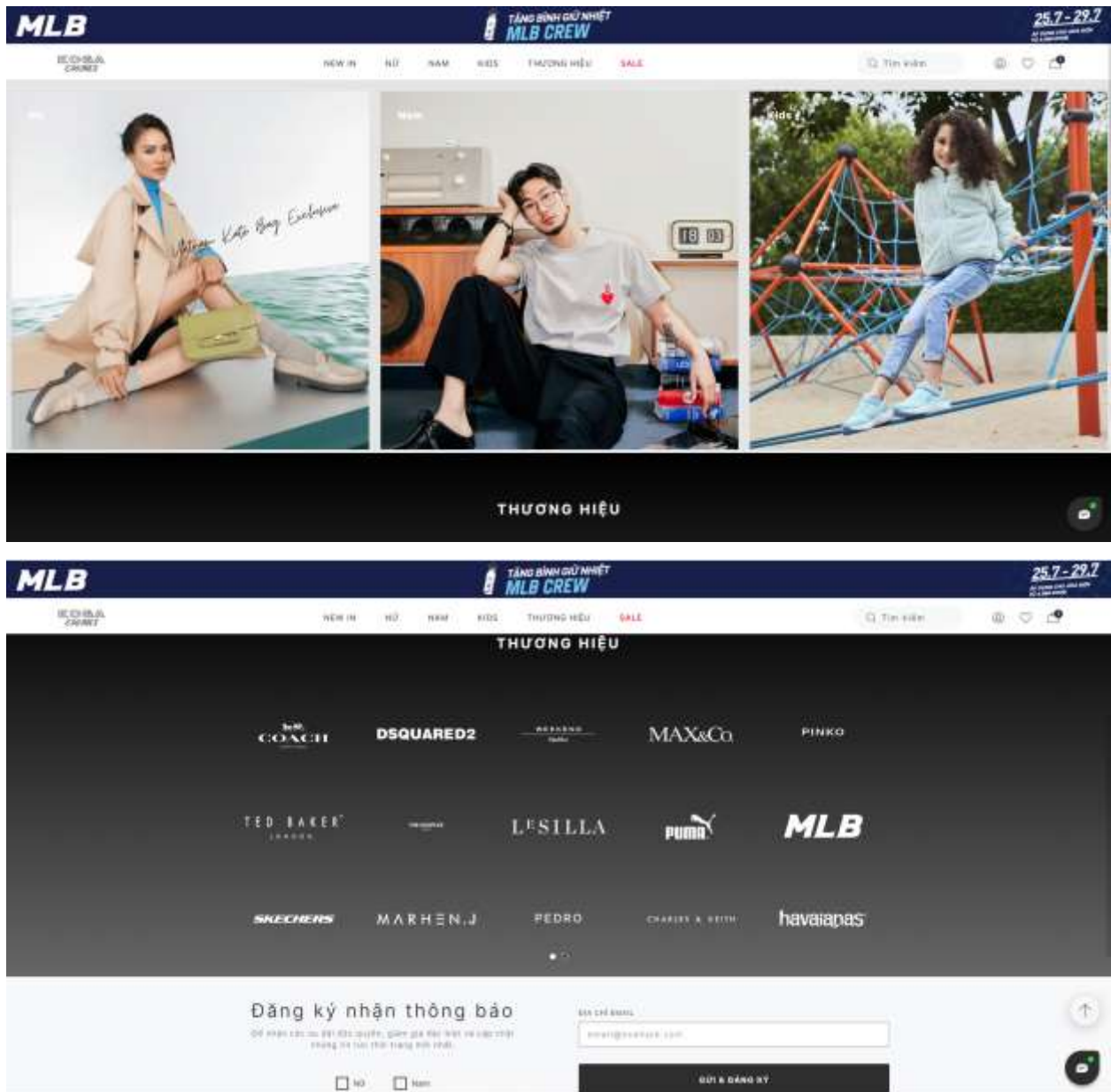


Hình 3.4. Biểu đồ quan hệ thực thể - ERD

CHƯƠNG 4. TRIỂN KHAI ỨNG DỤNG

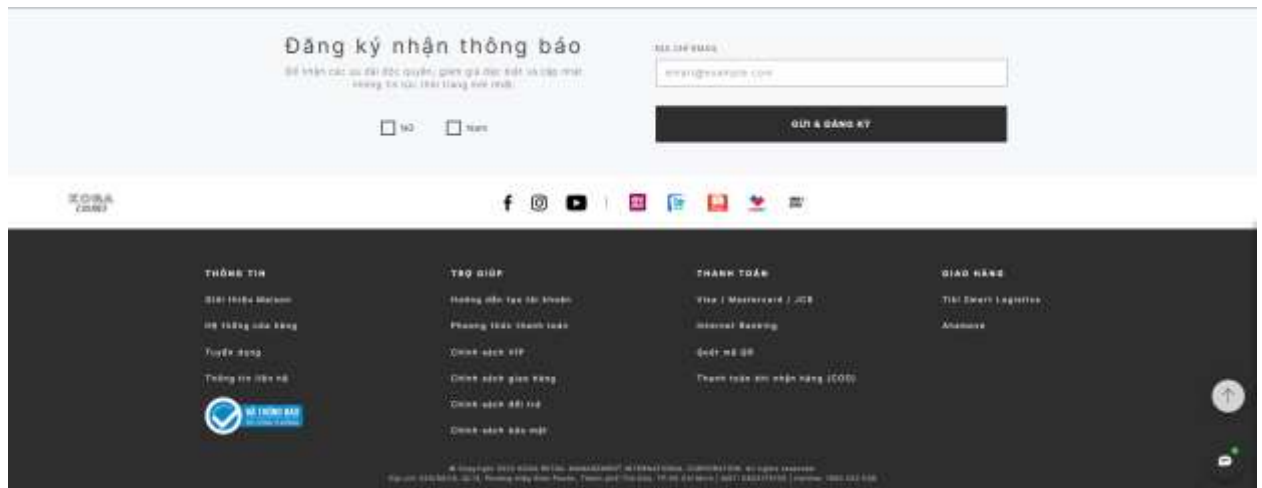
4.1. Triển khai Frontend

4.1.1. Giao diện trang chủ



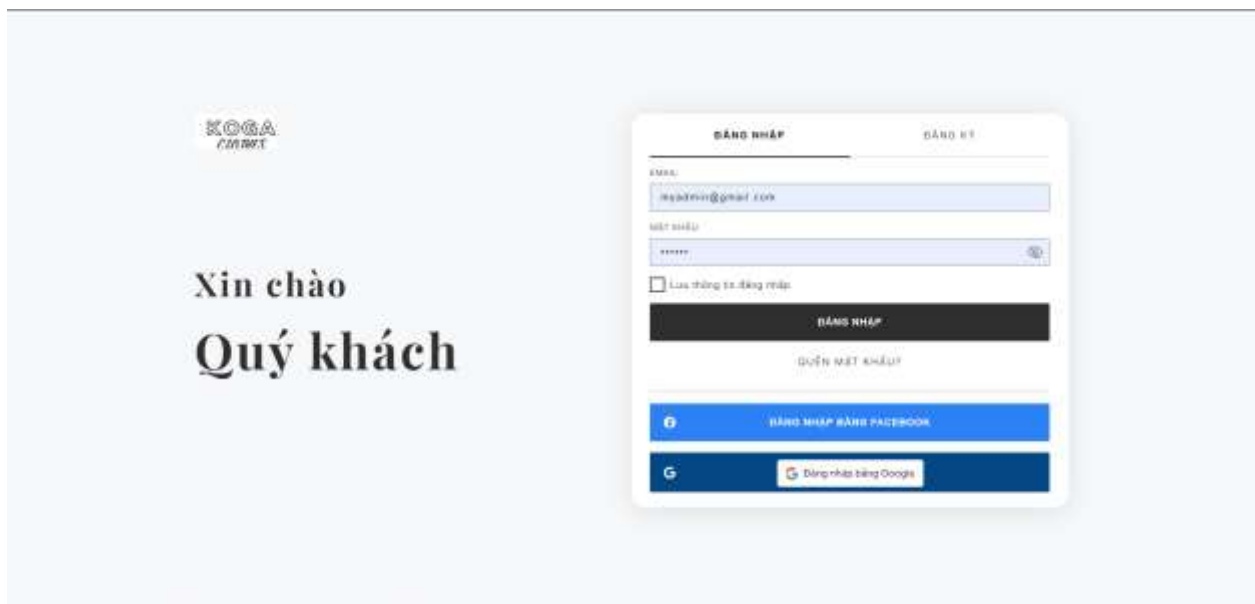
Hình 4.1.1. Giao diện trang chủ

Người dùng khi đăng nhập vào trang chủ sẽ thấy được các thành phần của trang chủ như: Slider để trình chiếu những ứng phẩm, Layout giới thiệu các nhãn hàng, thương hiệu, Layout đăng ký nhận thông báo và cuối cùng là Footer sẽ hiển thị những thông tin chi tiết về website.



Hình 4.1.2. Giao diện Footer

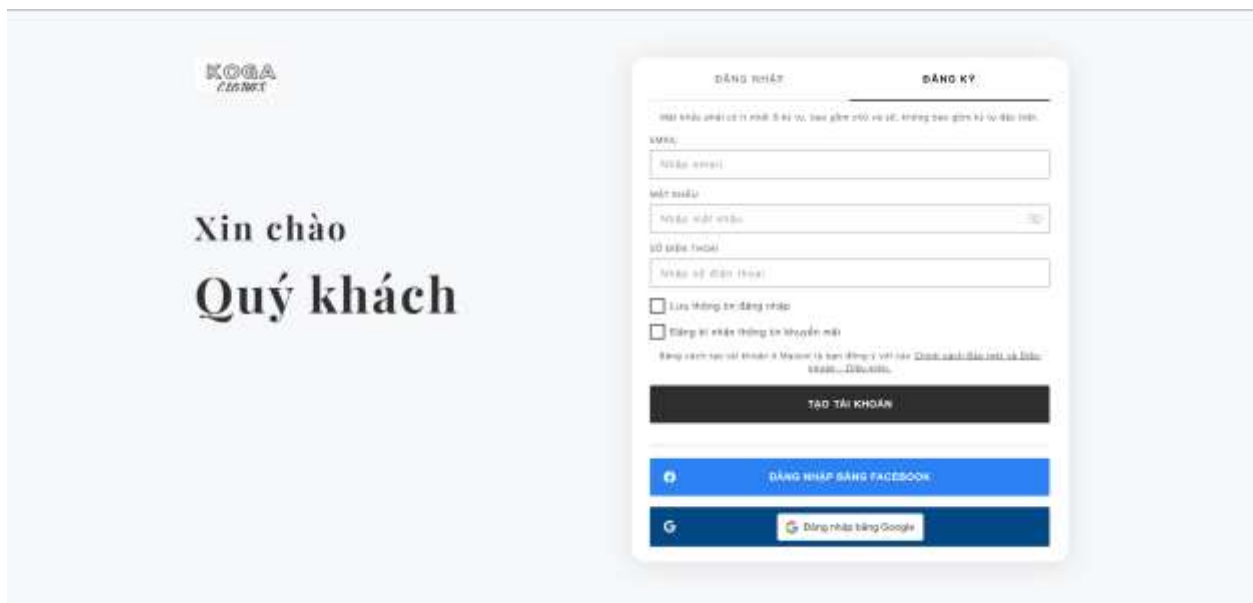
4.1.2. Giao diện trang đăng nhập



Hình 4.1.3. Giao diện trang đăng nhập

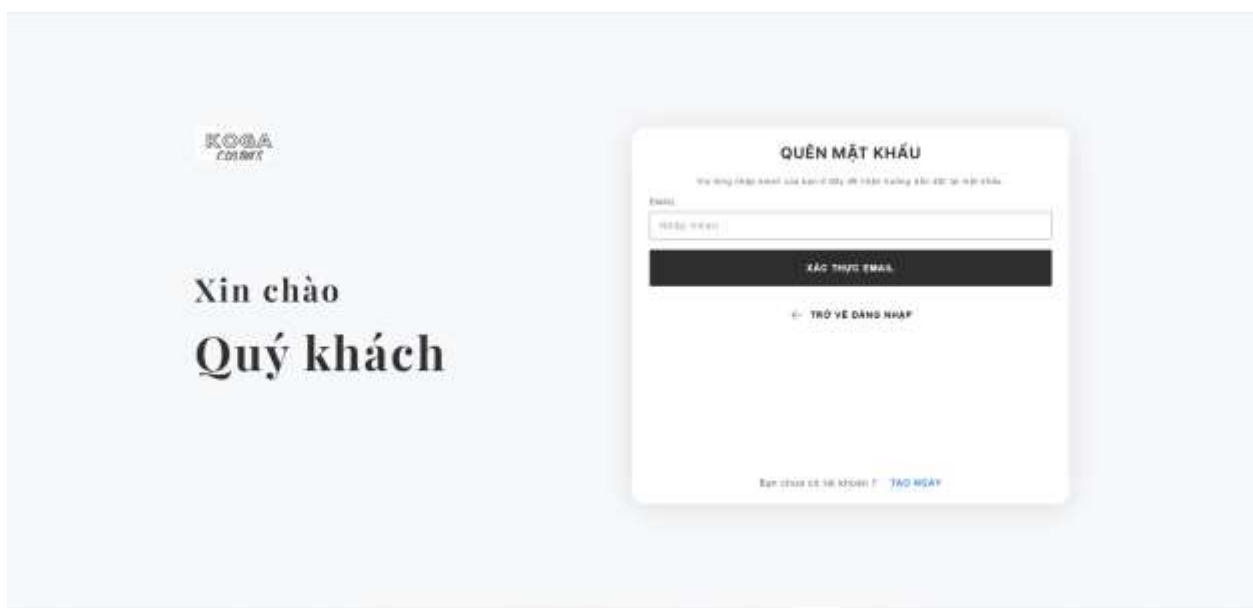
Giao diện đăng nhập được thiết kế tối giản, người dùng có thể nhập 2 trường thông tin email và mật khẩu để đăng nhập hoặc các button đăng nhập bằng Facebook hoặc đăng nhập bằng Google, ngoài ra có thể đăng ký khi click vào button Đăng ký và tìm lại mật khẩu.

4.1.3. Giao diện trang đăng ký tài khoản



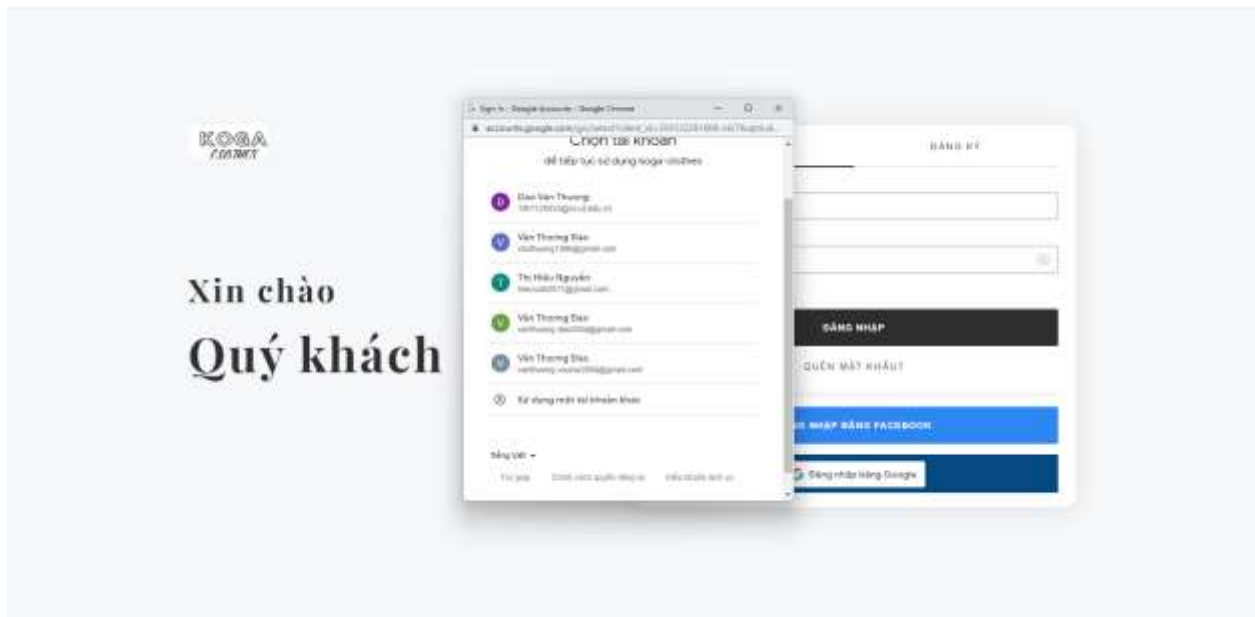
Hình 4.1.4. Giao diện trang đăng ký tài khoản

Tại đây người dùng cần nhập các trường thông tin cần thiết để đăng ký tài khoản như: email, mật khẩu và số điện thoại. Ngoài ra chức năng lưu thông tin mật khẩu và đăng ký nhận thông báo cũng được tích hợp vào đây.



Hình 4.1.5. Giao diện trang lấy lại mật khẩu.

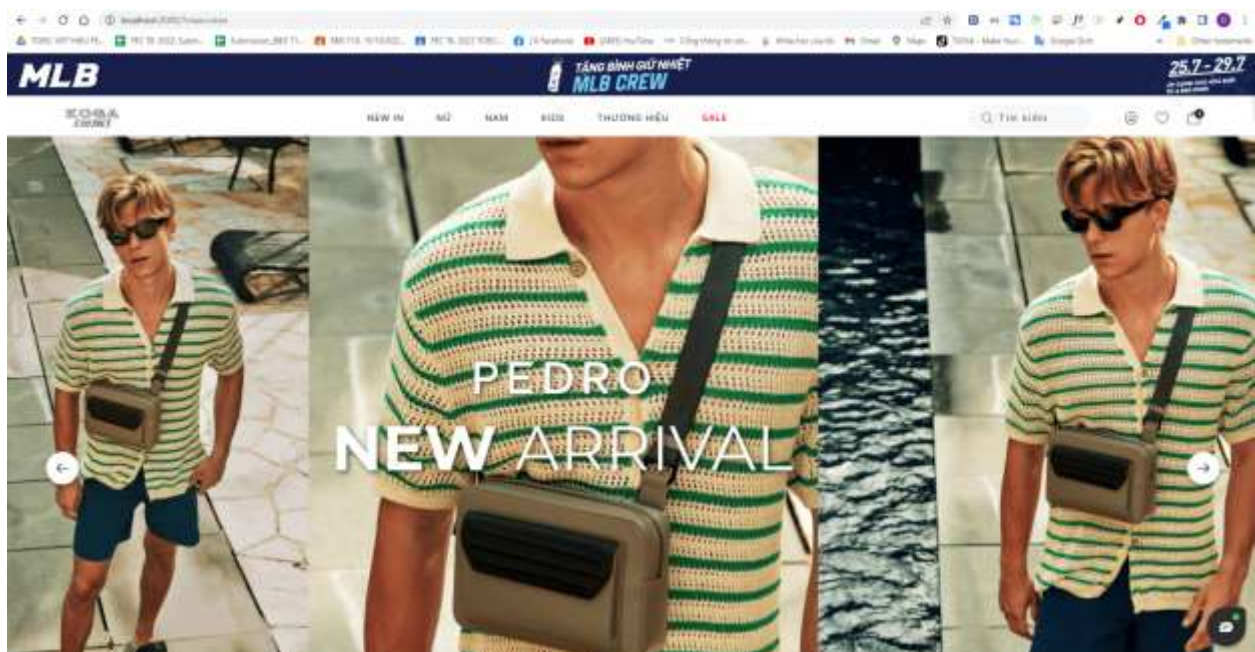
Tại đây nếu người dùng quên mật khẩu, thì chỉ cần nhập địa chỉ email đã tạo tài khoản trước đó và bấm “XÁC THỰC EMAIL”. Mật khẩu mới sẽ gửi về email của họ.



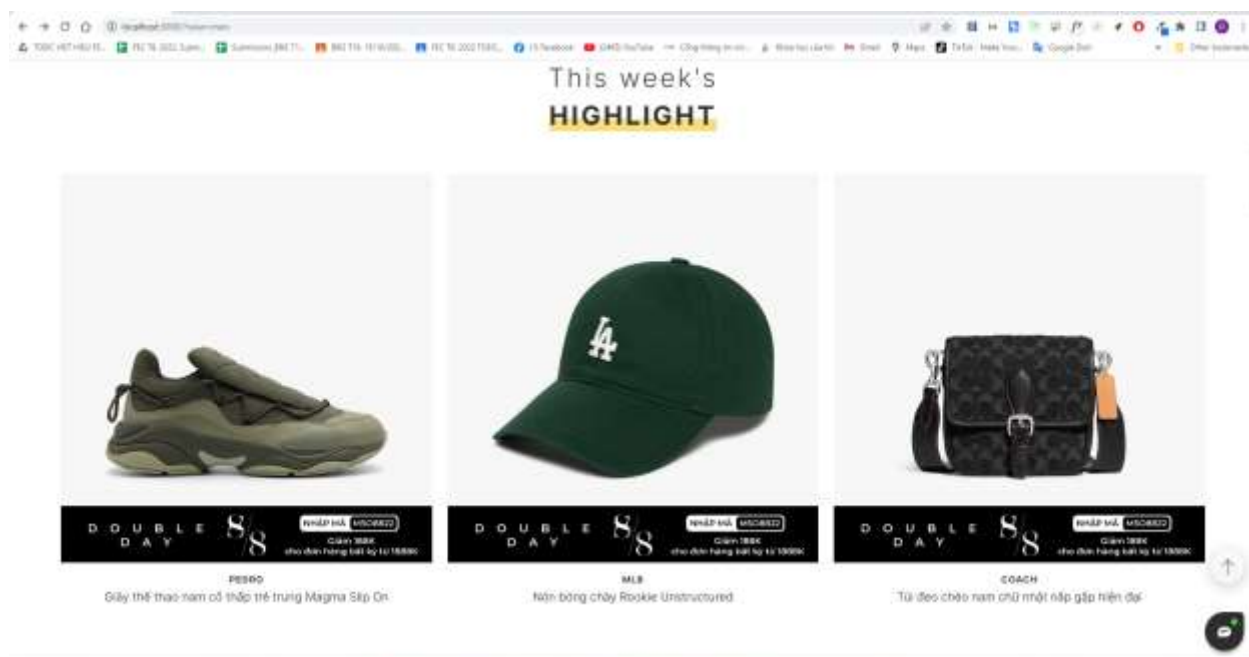
Hình 4.1.6. Giao diện đăng nhập bằng tài khoản Google

4.1.4. Giao diện sản phẩm của từng đối tượng

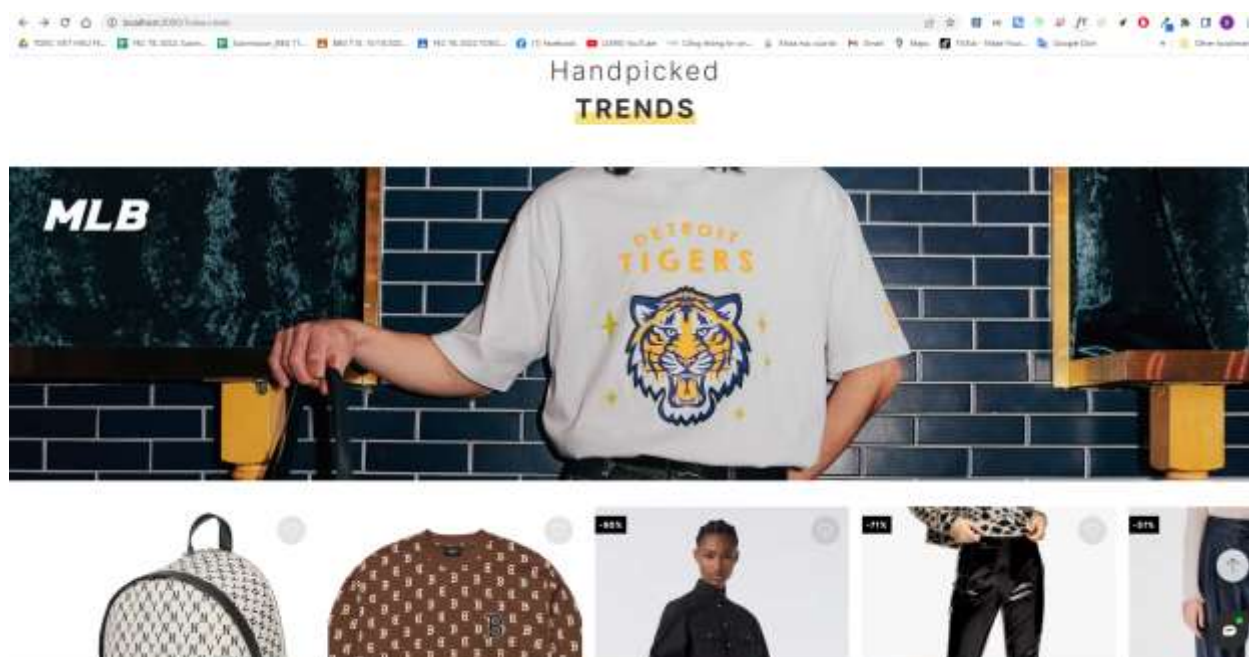
➤ Giao diện view dành cho Nam

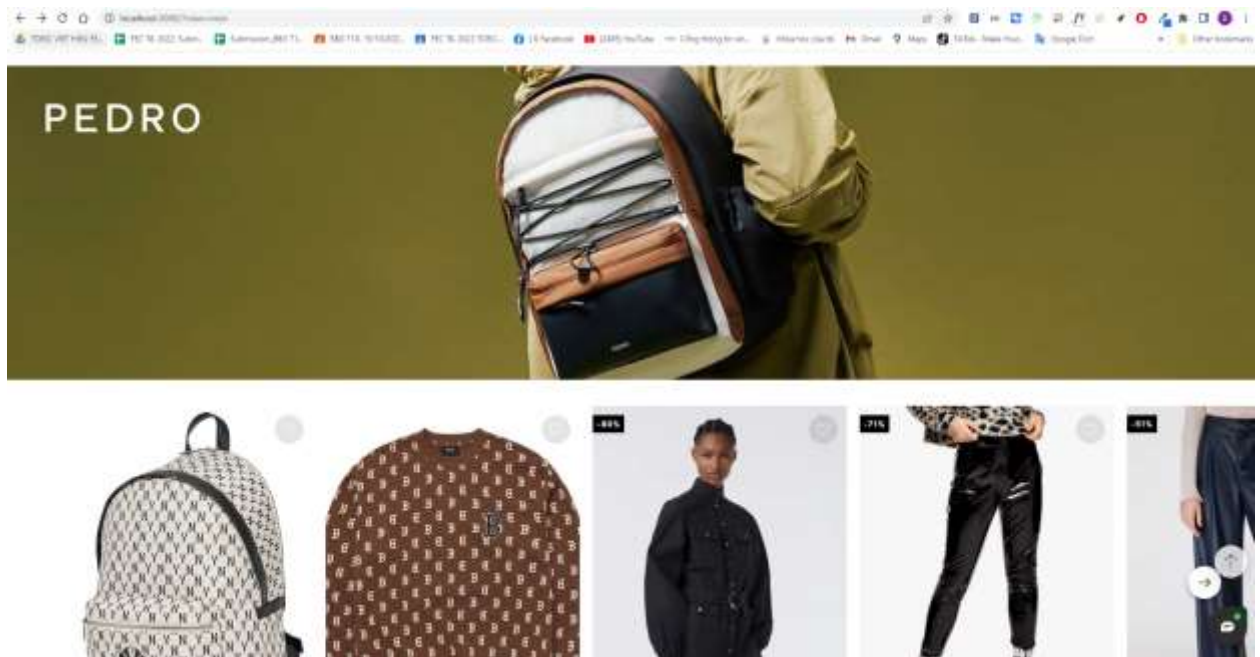


Hình 4.1.7. Giao diện Slide cho view Nam

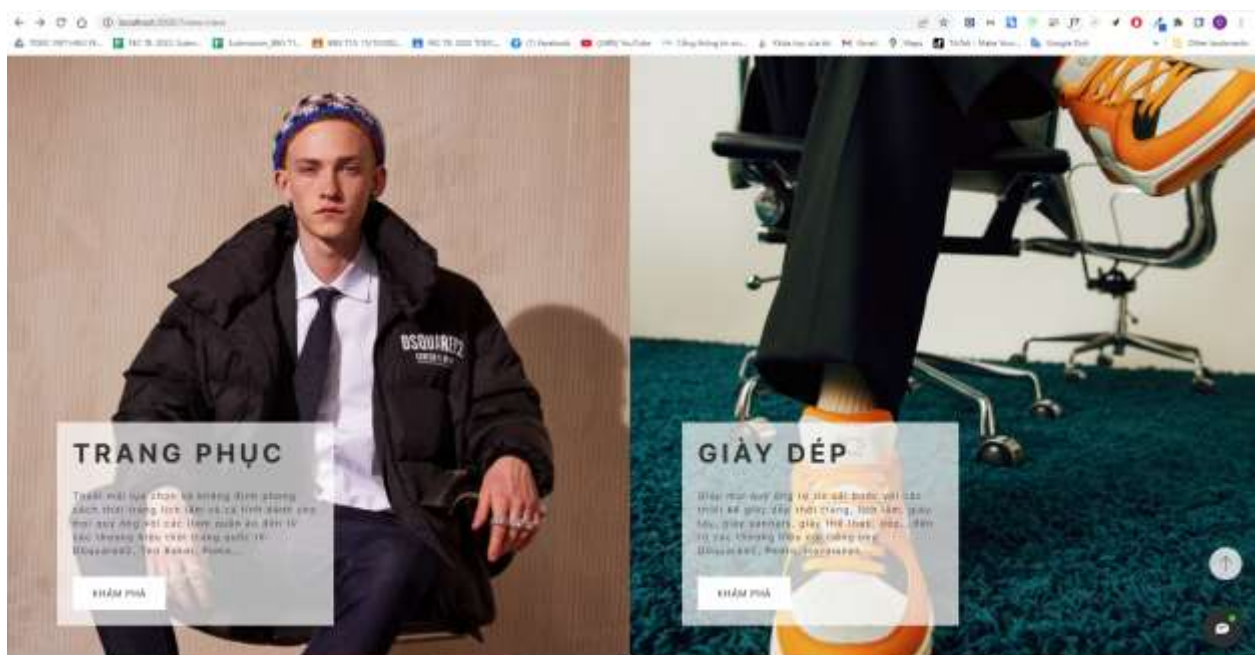


Hình 4.1.7. Giao diện sản phẩm nổi bật trong tuần dành cho nam

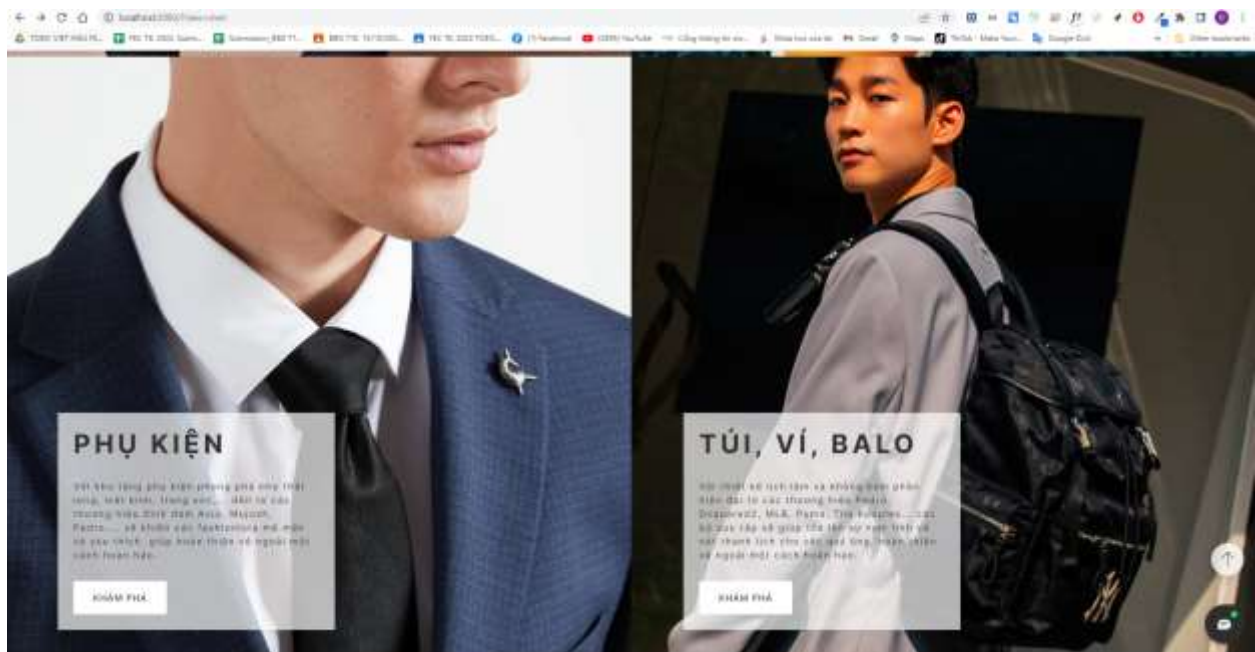




Hình 4.1.8. Giao diện sản phẩm được lựa chọn nhiều nhất theo Thương hiệu.

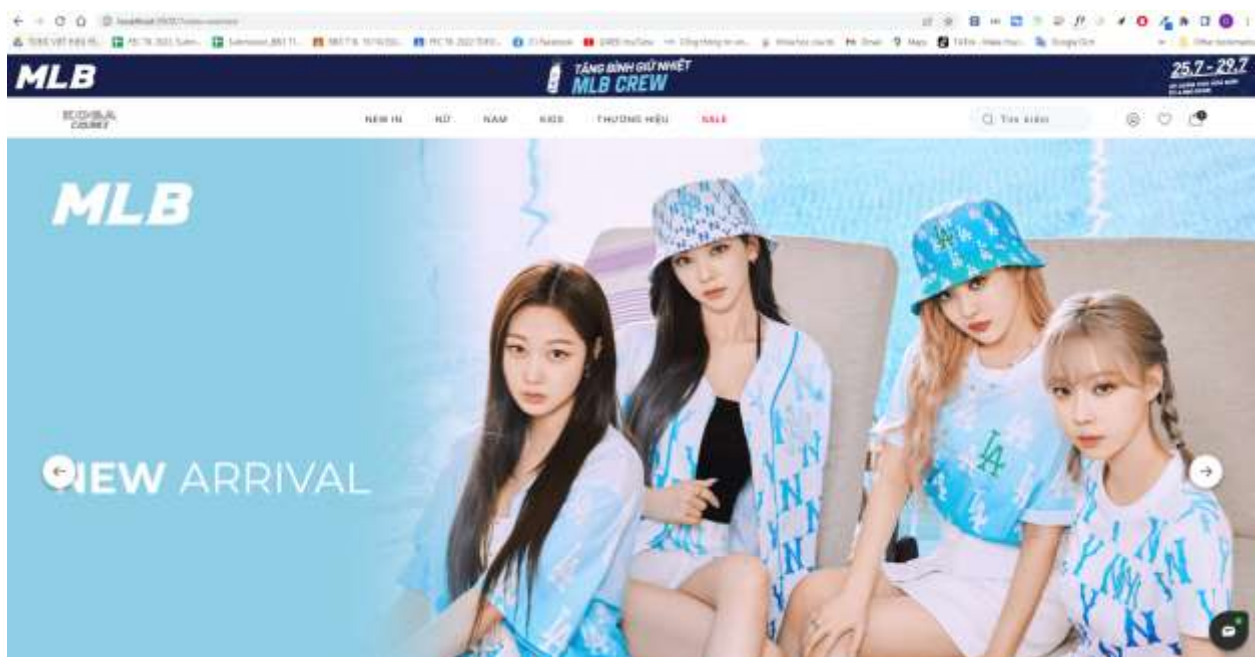


Hình 4.1.9. Giao diện mục lục sản phẩm dành cho nam.

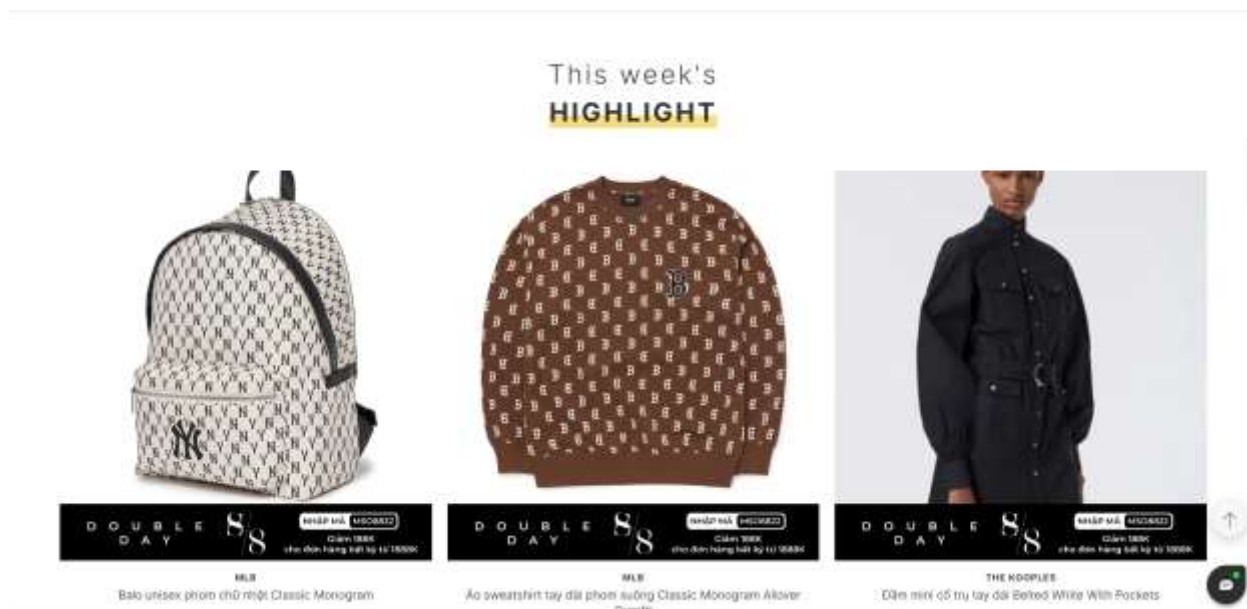
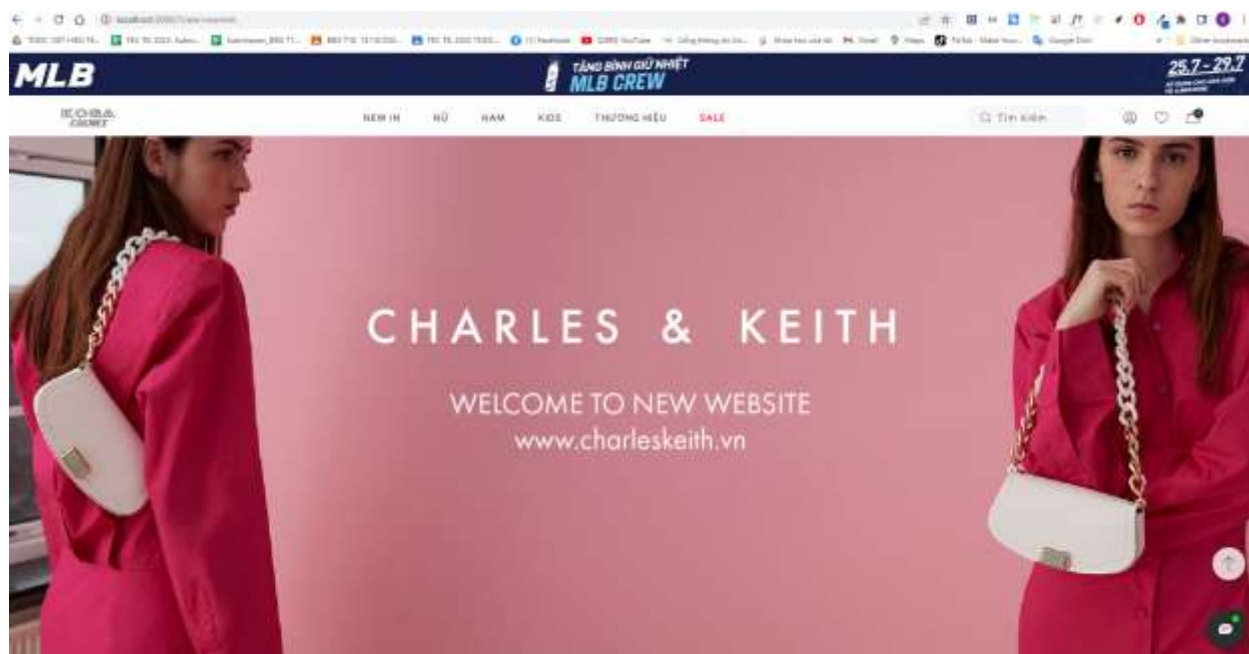


Hình 4.1.10. Giao diện mục lục sản phẩm dành cho nam.

➤ Giao diện view dành cho Nữ

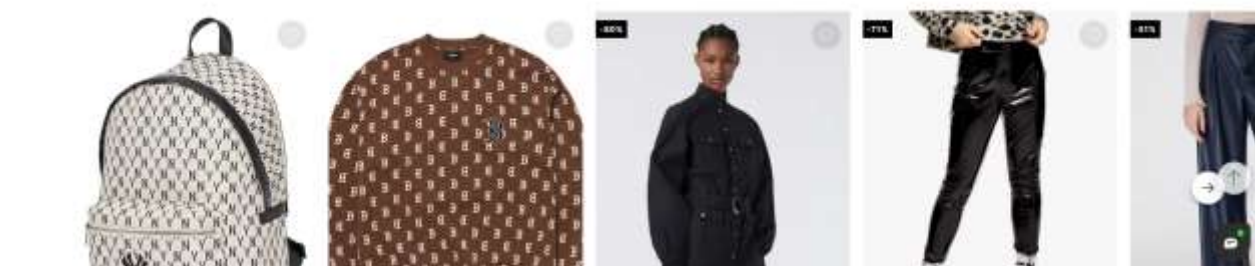
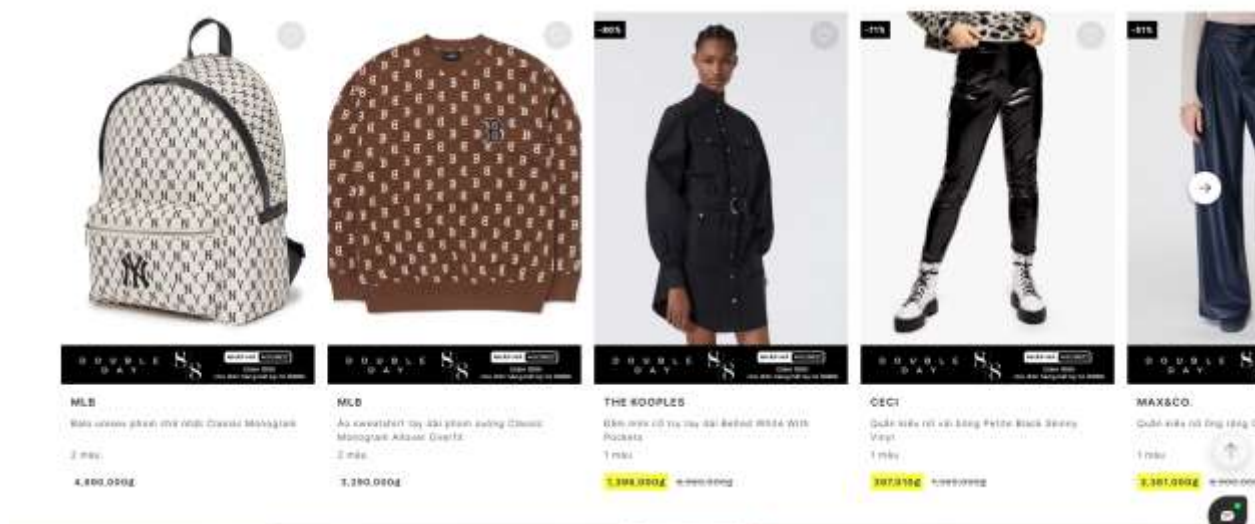


Hình 4.1.11. Giao diện Slide cho view Nữ



Hình 4.1.12. Giao diện sản phẩm nổi bật trong tuần dành cho nữ





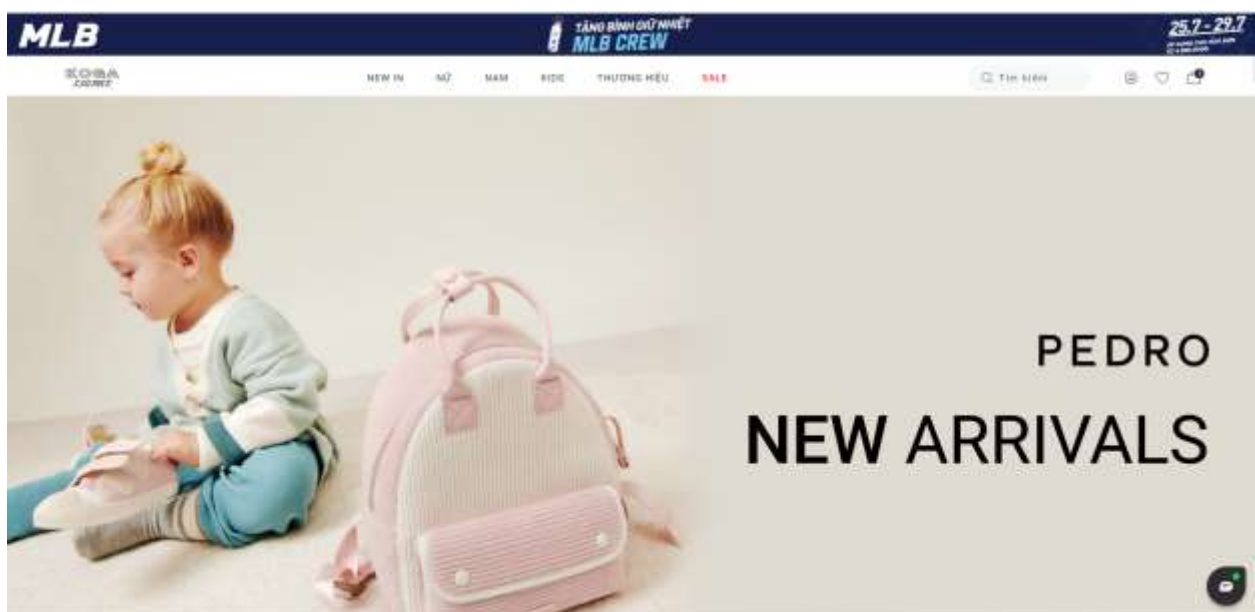
Hình 4.1.13. Giao diện sản phẩm được lựa chọn nhiều nhất theo Thương hiệu.



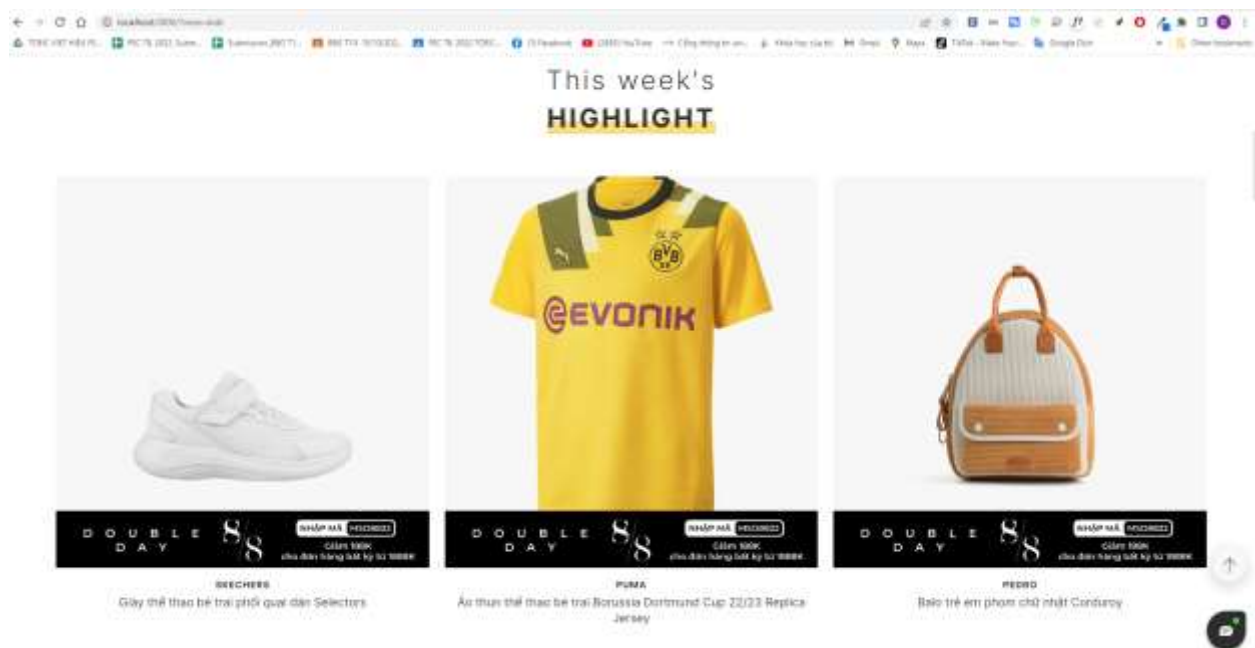


Hình 4.1.14. Giao diện mục lục sản phẩm dành cho nữ.

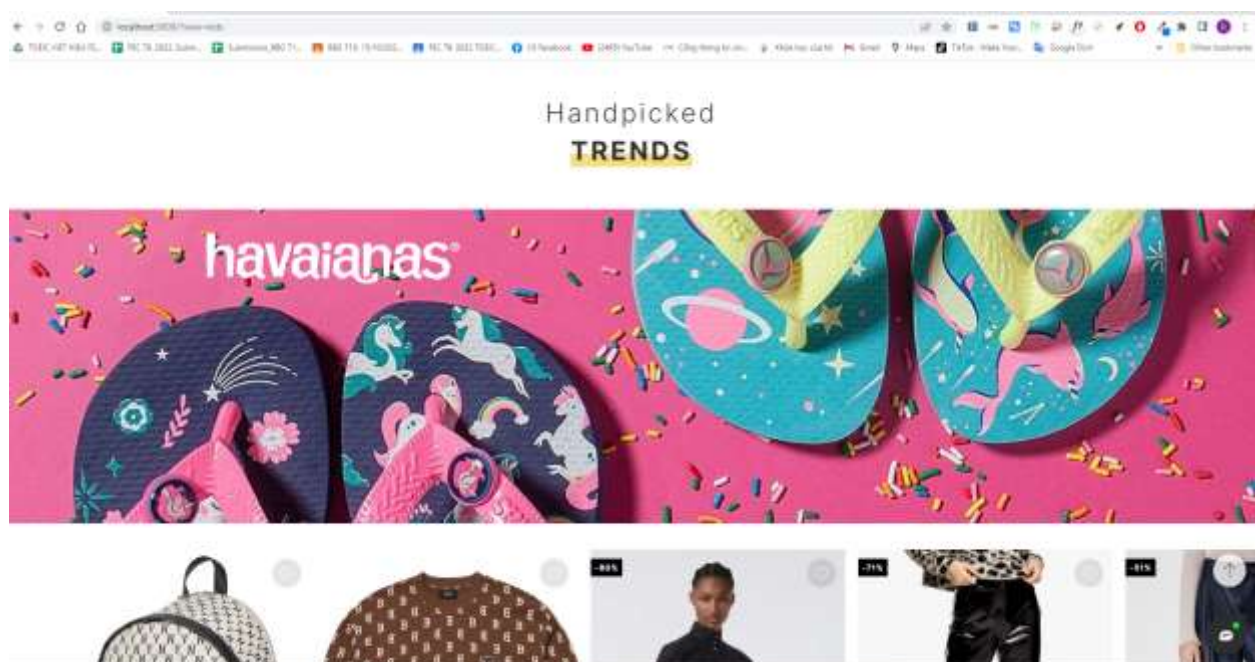
➤ **Giao diện view dành cho Kid**



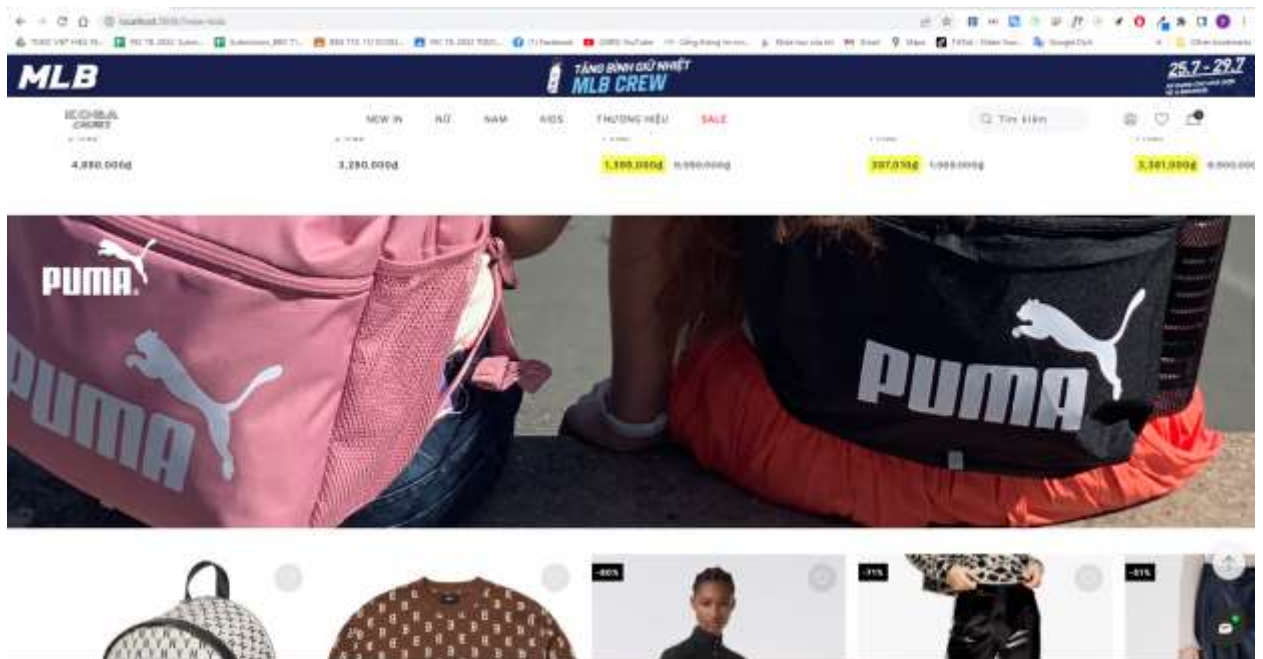
Hình 4.1.15. Giao diện Slide cho view Kid



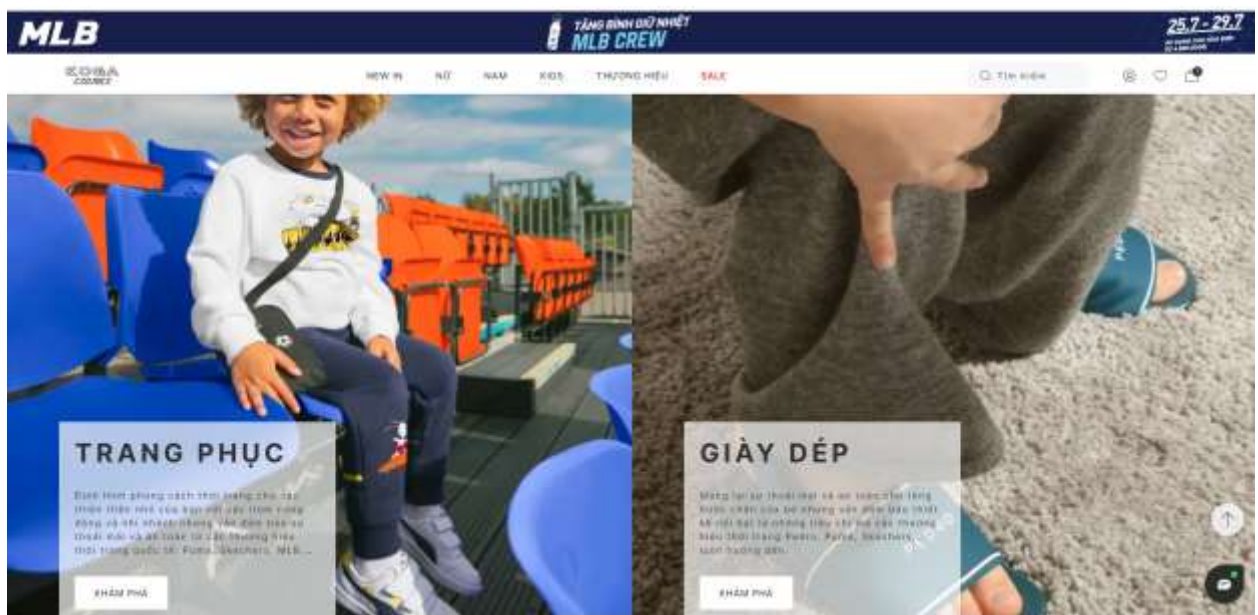
Hình 4.1.16. Giao diện sản phẩm nổi bật trong tuần dành cho kids

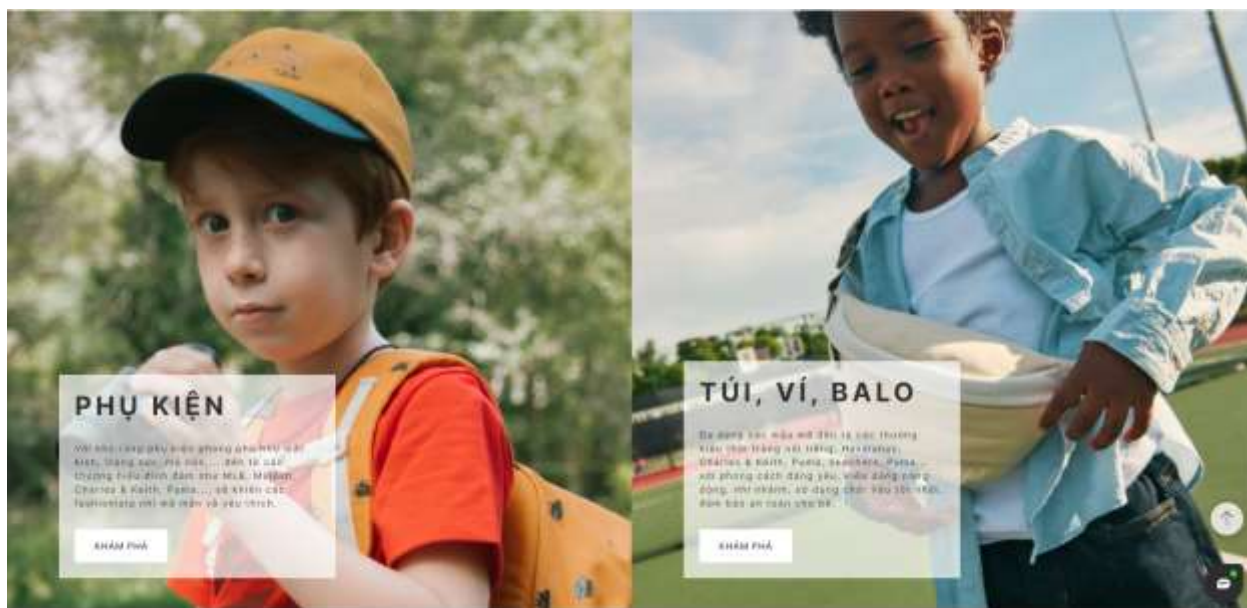


Hình 4.1.18. Giao diện sản phẩm được lựa chọn nhiều nhất theo Thương hiệu.



Hình 4.1.19. Giao diện sản phẩm được lựa chọn nhiều nhất theo Thương hiệu.





Hình 4.1.20. Giao diện mục lục sản phẩm dành cho kid.

4.1.5. Giao diện Layout category - navigation



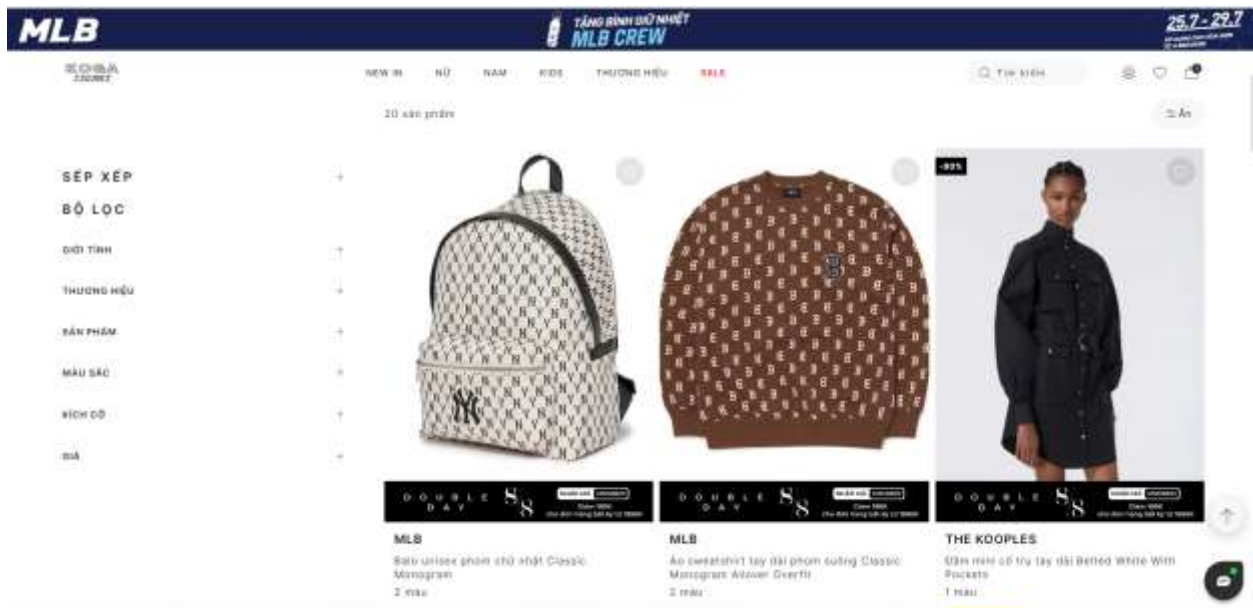
Hình 4.1.21. Giao diện trang Layout category - navigation

Ở tất cả các trang đều có thể truy cập Category bằng cách hover vào thanh Navigation bar, hệ thống sẽ xổ xuống một category bao gồm tất cả danh mục sản phẩm.

Ở thanh Navigation bar này còn có 3 button:

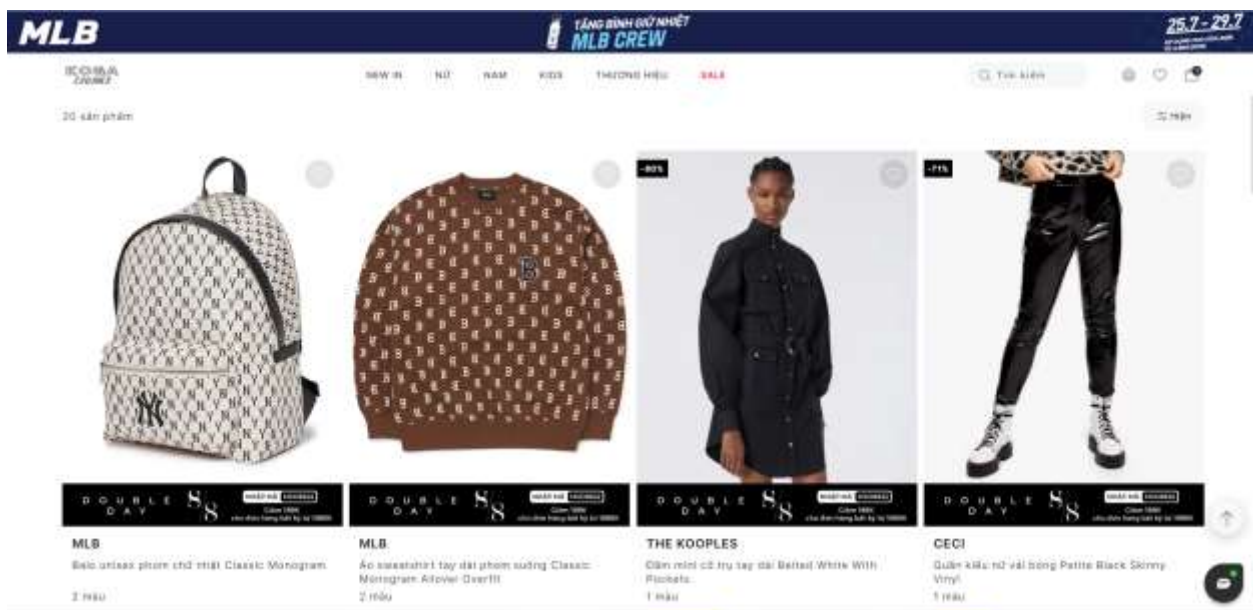
- Thông tin cá nhân.
- Danh sách sản phẩm yêu thích.
- Giỏ hàng.

4.1.6. Giao diện trang tất cả sản phẩm



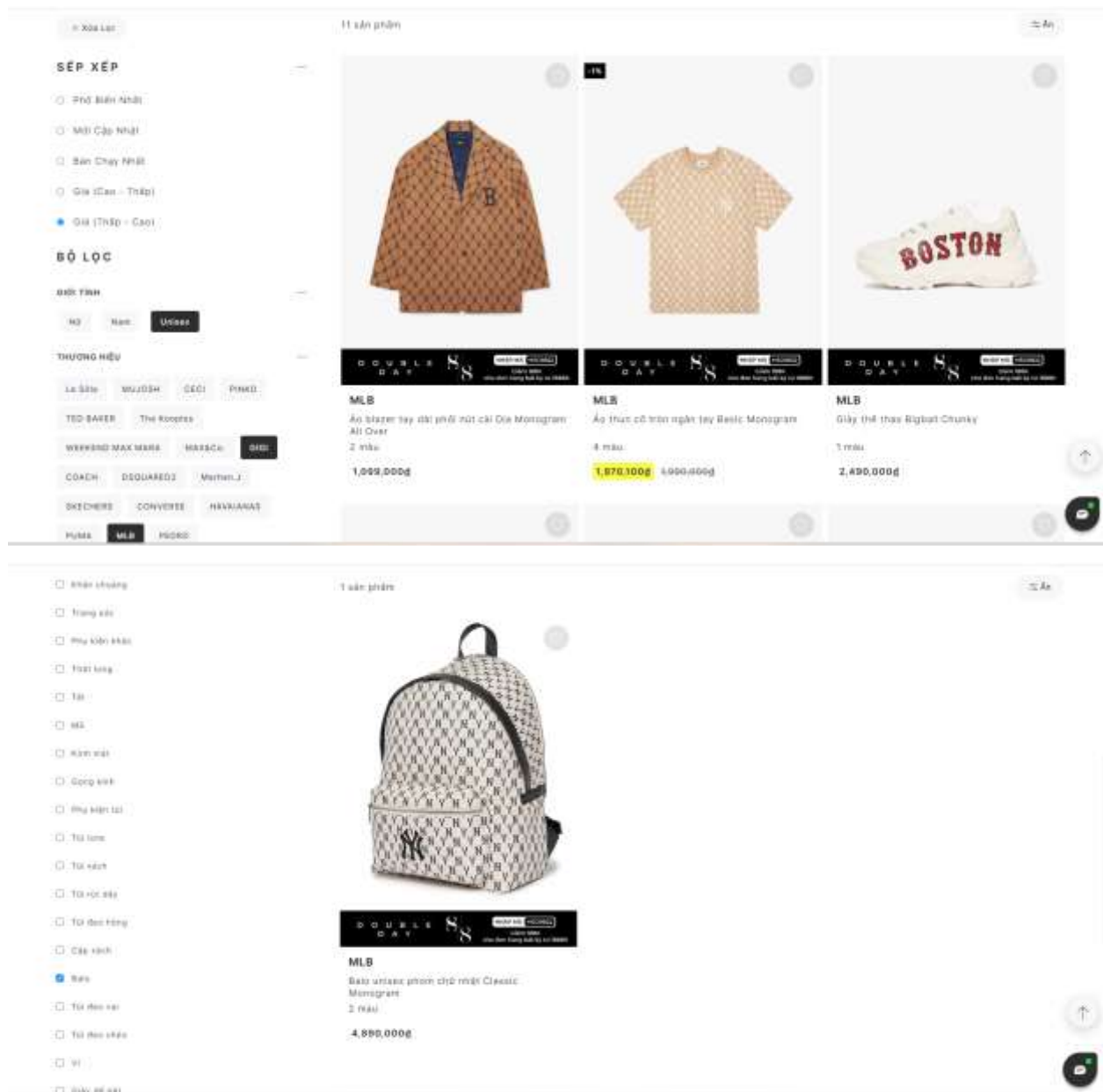
Hình 4.1.22. Giao diện trang tất cả sản phẩm đang hiện filter

Giao diện trang tổng sản phẩm sẽ liệt kê ra tất cả sản phẩm có trong cửa hàng. Bên phải còn có một bảng filter dùng để lọc các sản phẩm cần tìm. Góc phải trên sẽ có button có thể ẩn tính năng filter này.



Hình 4.1.23. Giao diện trang tất cả sản phẩm đã ẩn filter

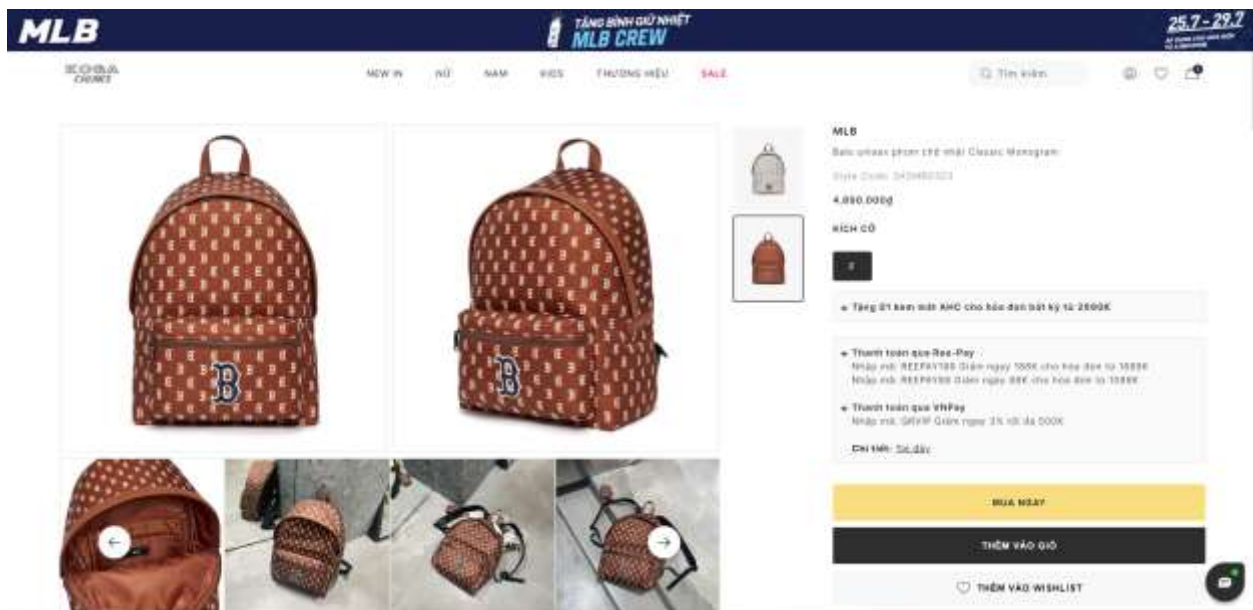
Giao diện trang tổng sản phẩm khi đã tắt tính năng filter, danh sách sản phẩm sẽ được nâng thành bốn cột sản phẩm thay vì ba như lúc bật tính năng filter.



Hình 4.1.24. Giao diện trang tất cả sản phẩm đã filter.

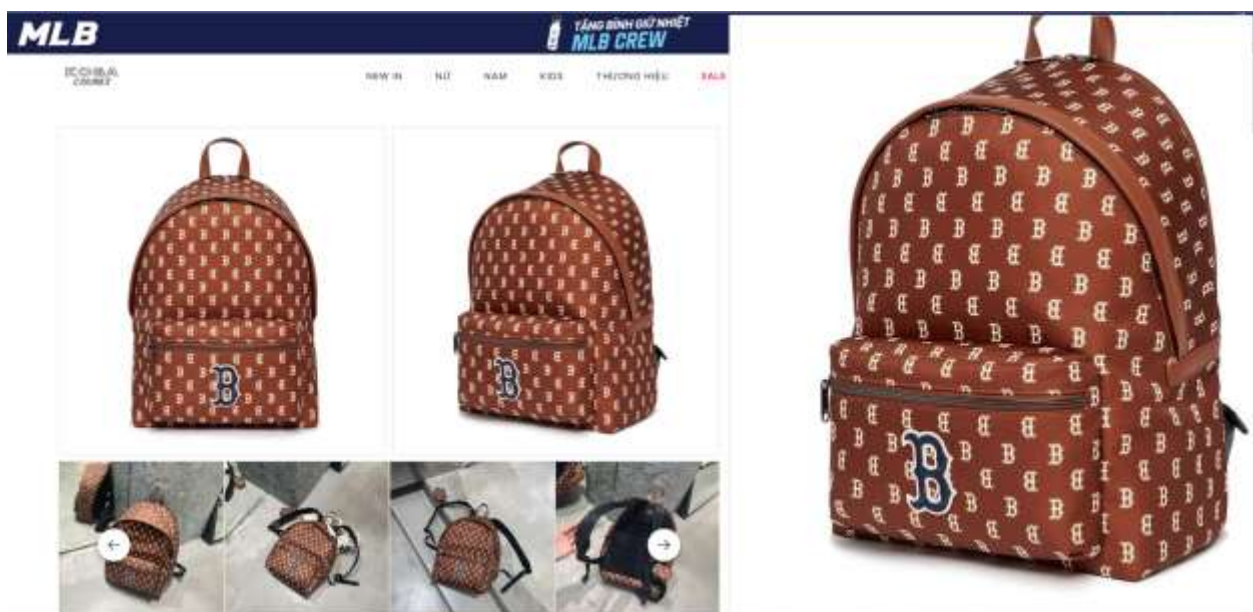
Giao diện trang tất cả sản phẩm đã được sắp xếp theo từ giá thấp đến cao và được lọc bởi giới tính, thương hiệu và theo danh mục sản phẩm. Phía trên bên trái có nút “Xóa Lọc” để xóa tất cả bộ lọc.

4.1.7. Giao diện chi tiết sản phẩm



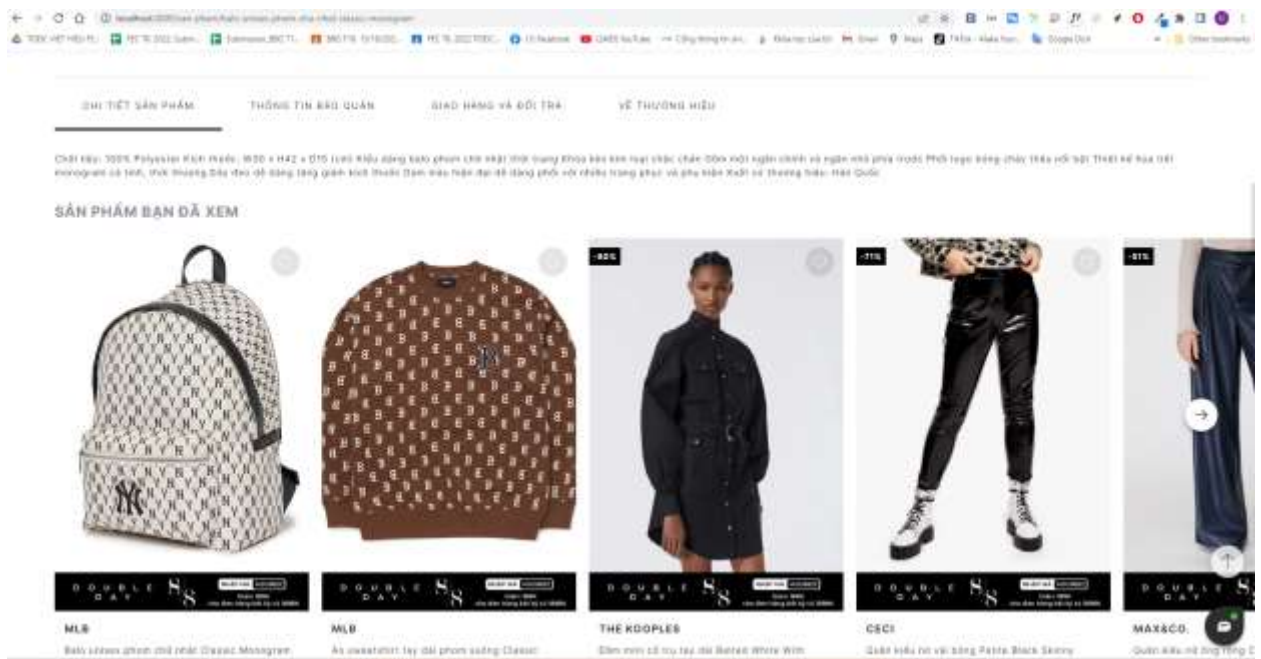
Hình 4.1.25. Giao diện trang chi tiết sản phẩm.

Giao diện trang chi tiết sản phẩm sẽ hiển thị thông tin sản phẩm đã chọn, ngoài ra còn có nút “MUA NGAY”, “THÊM VÀO GIỎ”, “THÊM VÀO WISHLIST”. Bên dưới sẽ có một panel để xem các thông tin như “chi tiết sản phẩm”, “thông tin bảo quản”, “giao hàng và đổi trả” cũng như giới thiệu “về thương hiệu” của sản phẩm đó.

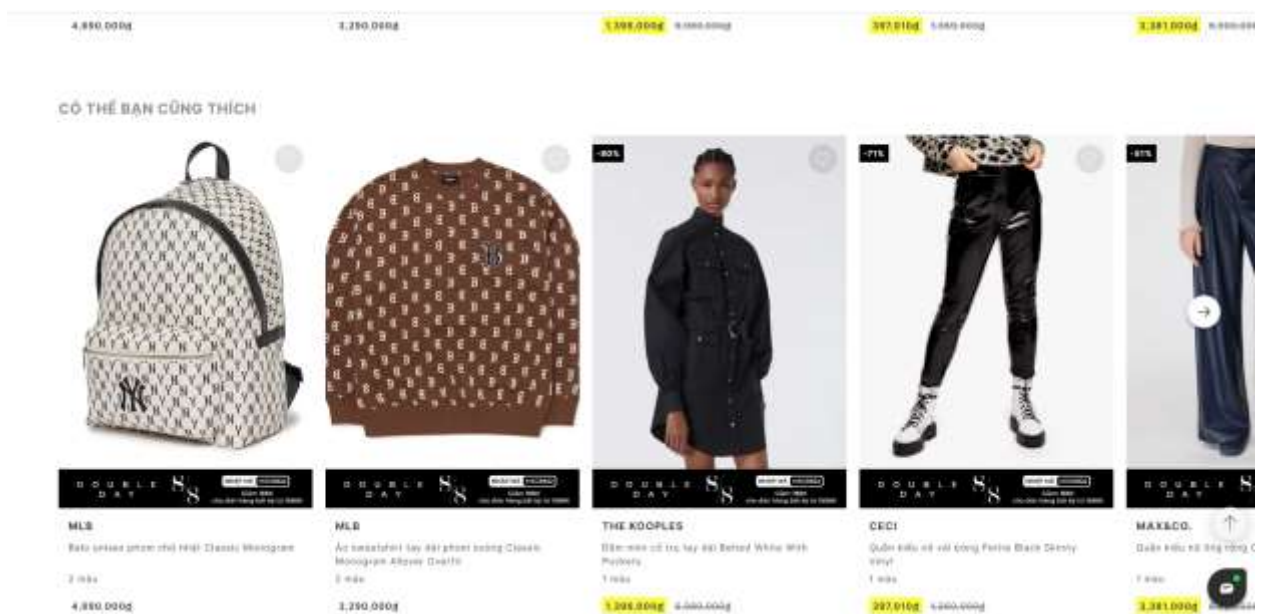


Hình 4.1.25. Giao diện trang chi tiết sản phẩm khi hover vào ảnh.

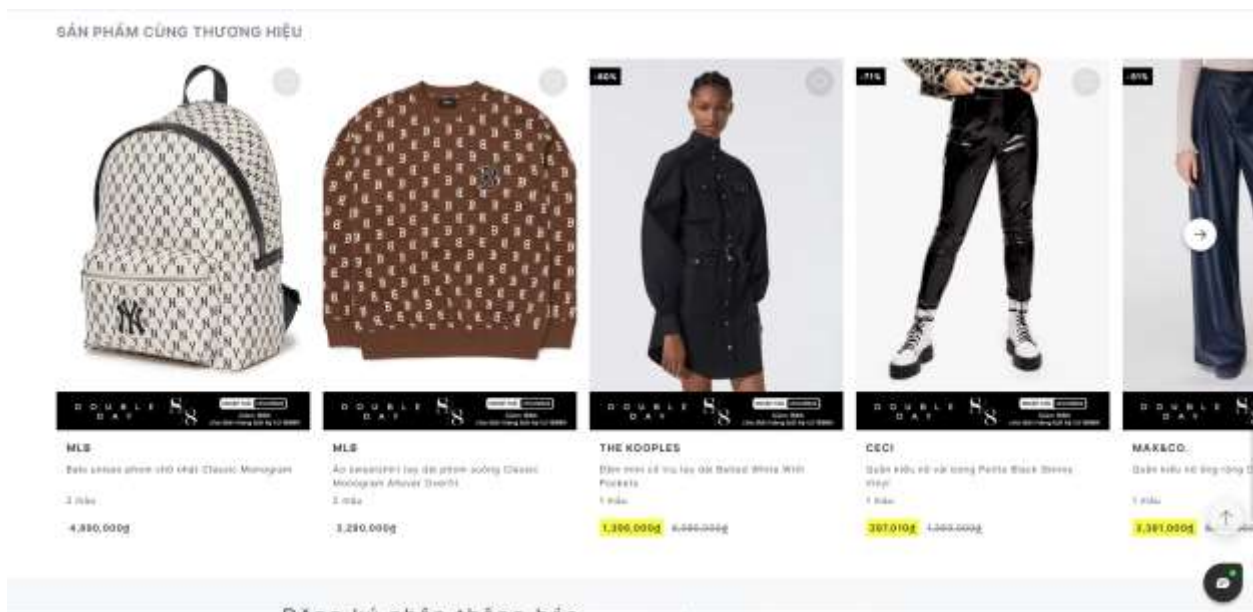
Khi người dùng hover con trỏ chuột vào những hình ảnh của sản phẩm thì sẽ zoom to hình ảnh đó ở bên khung ảnh bên phải.



Hình 4.1.26. Giao diện trang chi tiết sản phẩm.



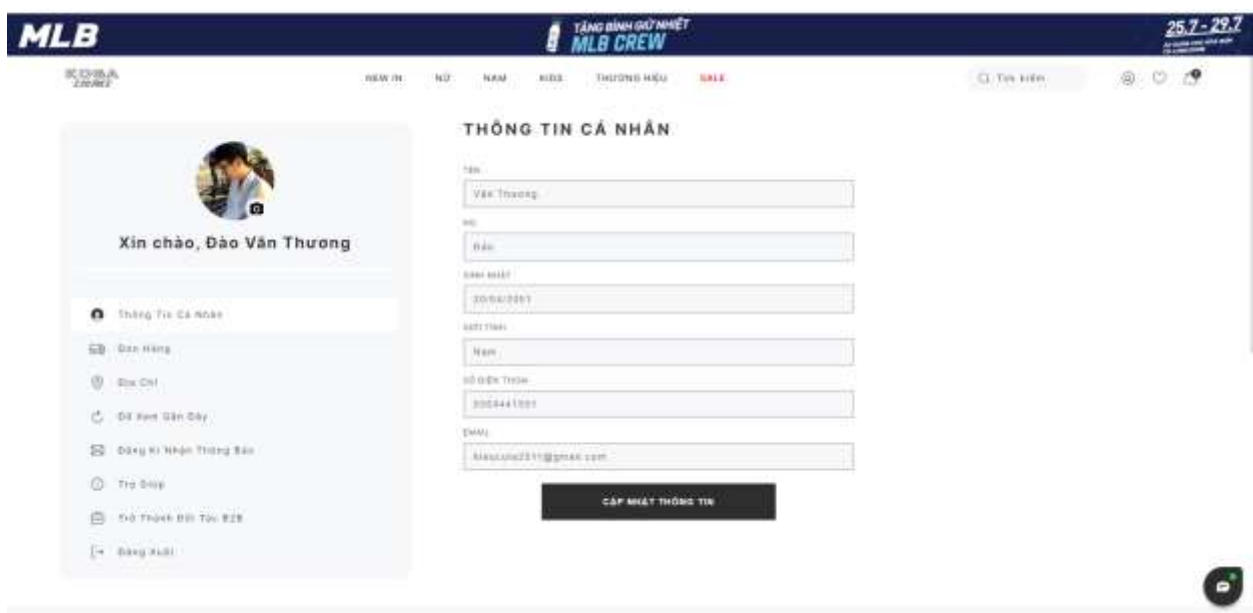
Hình 4.1.27. Giao diện trang chi tiết sản phẩm.



Hình 4.1.28. Giao diện trang chi tiết sản phẩm.

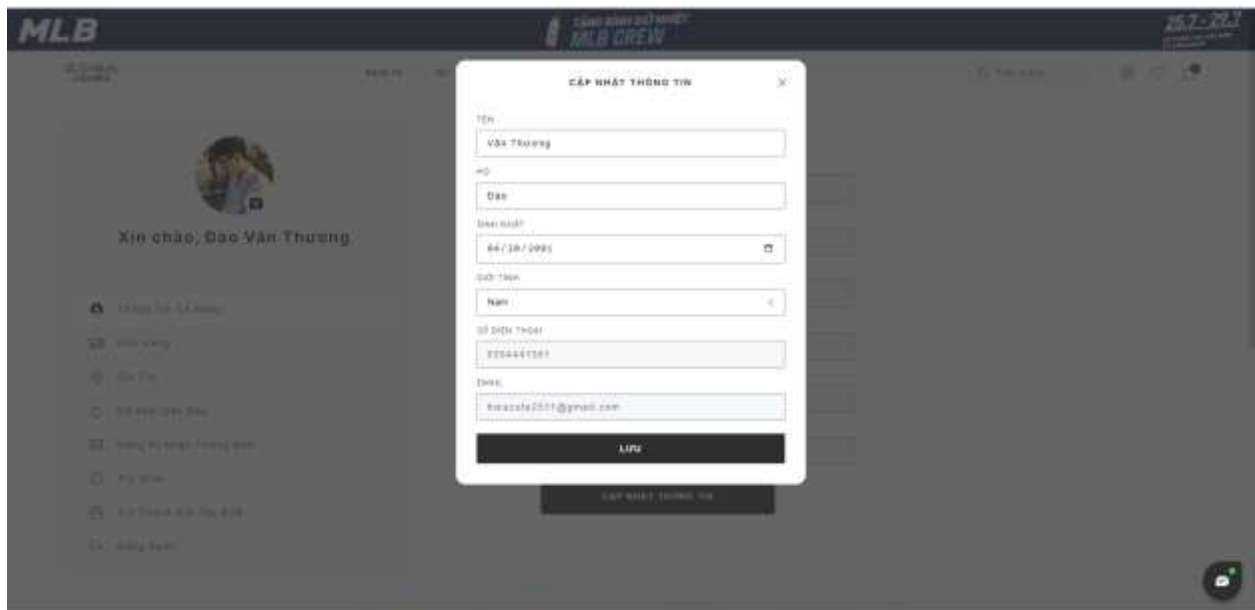
Để nâng cấp trải nghiệm người dùng hệ thống có thêm các thành phần bên dưới như: “Sản phẩm đã xem”, “Sản phẩm bạn yêu thích” và “Sản phẩm cùng thương hiệu” để người dùng có thể thuận tiện hơn khi muốn xem thêm sản phẩm khác.

4.1.8. Giao diện trang thông tin cá nhân



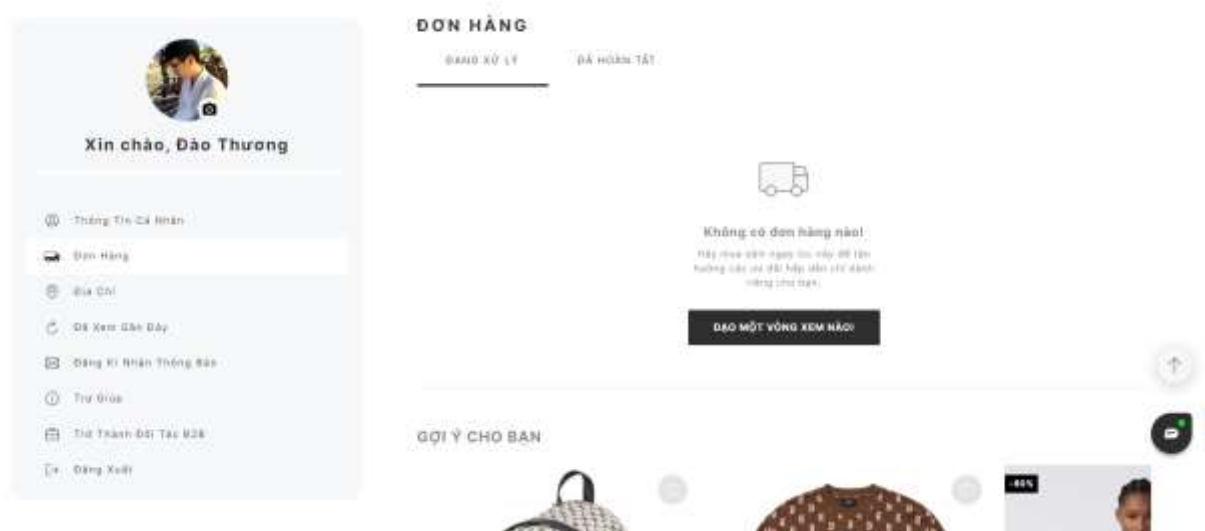
Hình 4.1.29. Giao diện trang thông tin cá nhân.

Trang thông tin cá nhân sẽ hiển thị hình ảnh đại diện kèm theo các thông tin cá nhân mà hệ thống ghi nhận được vào bên phải.

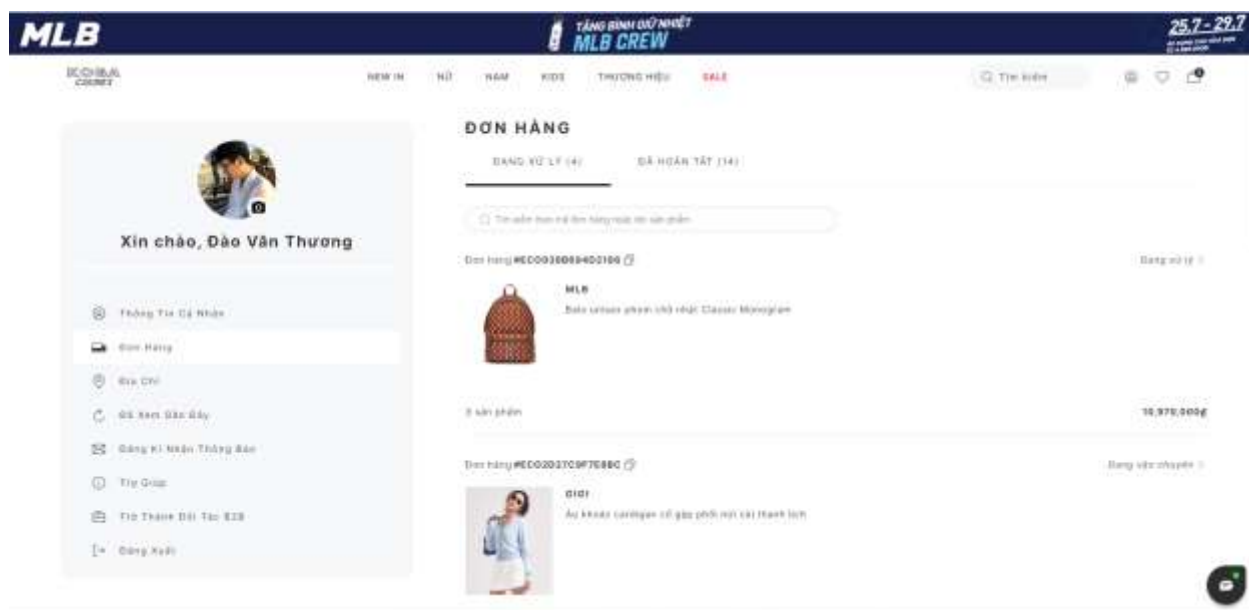


Hình 4.1.30. Giao diện trang thông tin cá nhân – cập nhật thông tin.

4.1.9. Giao diện trang đơn hàng

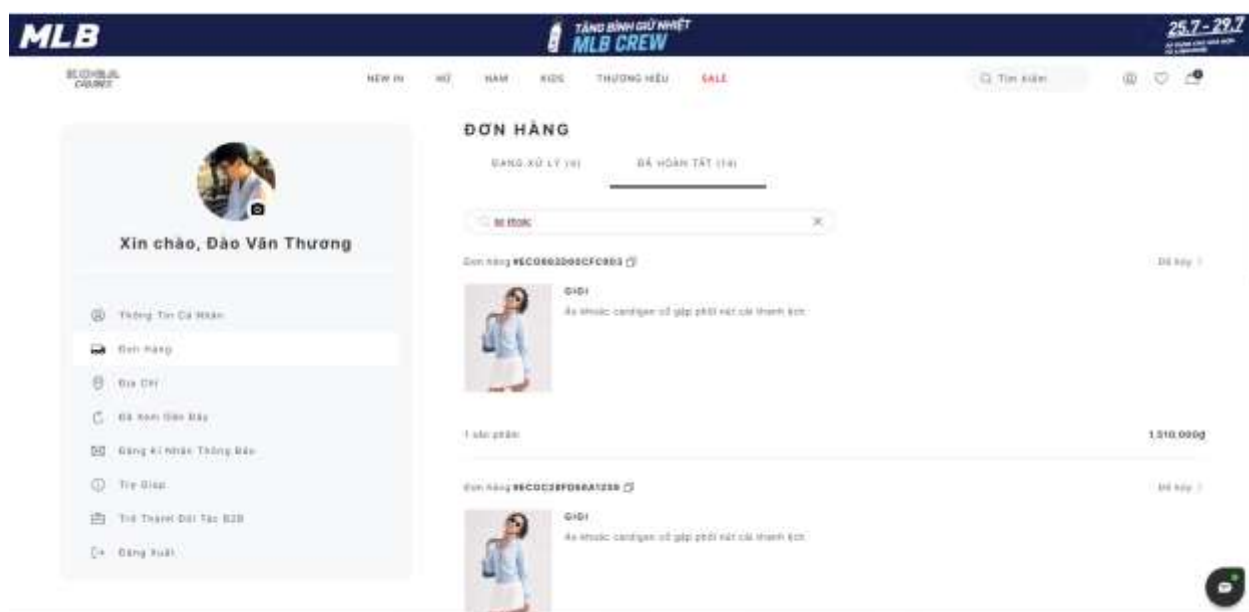


Hình 4.1.31. Giao diện trang đơn hàng khi người dùng chưa có đơn hàng nào.

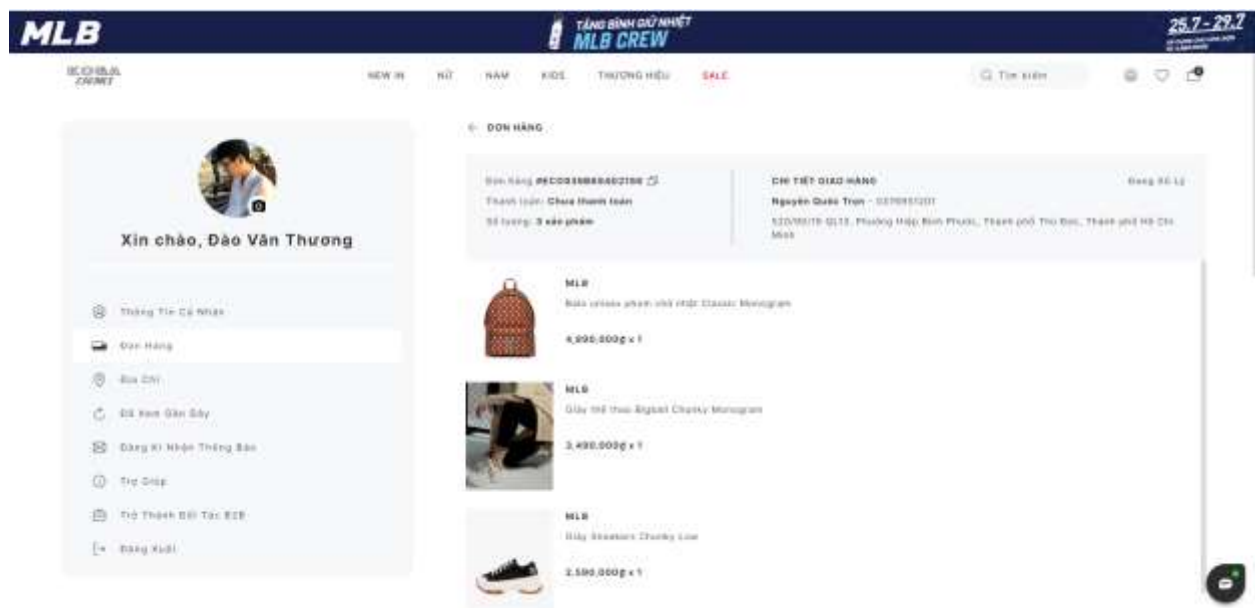


Hình 4.1.32. Giao diện trang đơn hàng.

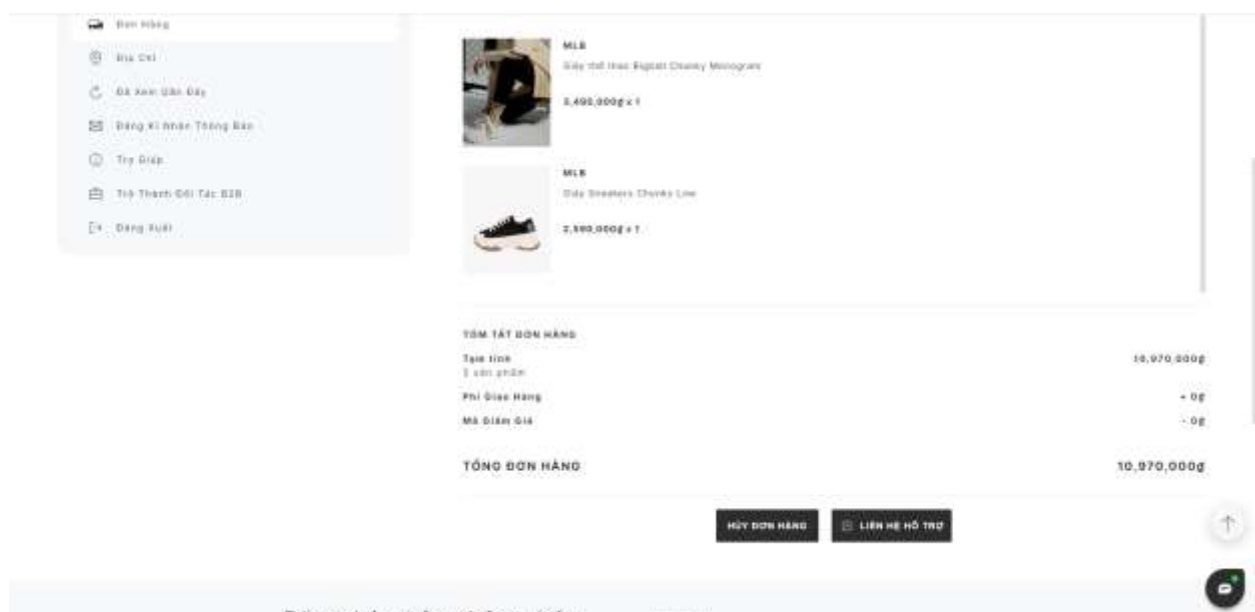
Giao diện đơn hàng sẽ hiển thị các đơn hàng đã đặt, trang đơn hàng sẽ được chia thành hai tab là: “Đang xử lý” và “Đã xử lý” để người dùng có thể dễ dàng theo dõi lịch sử đơn hàng của mình. Ngoài ra để thu hút thêm lượt xem và mua sản phẩm, trang sẽ được thêm một tab “gợi ý cho bạn” để người dùng có thể dễ dàng click vào.



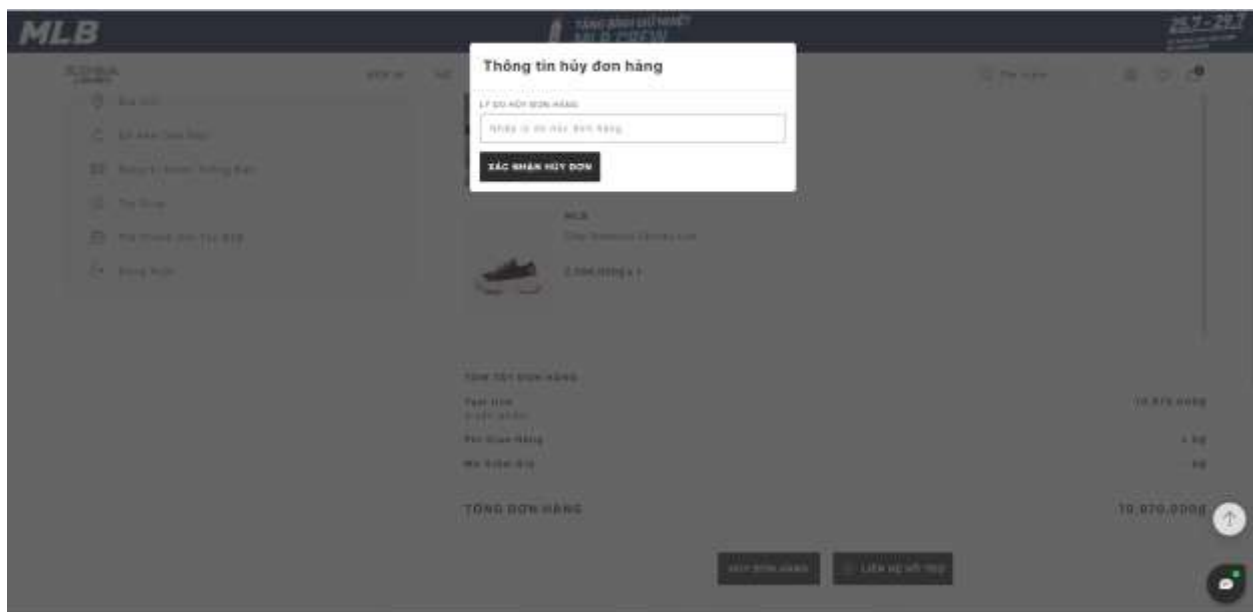
Hình 4.1.33. Giao diện trang đơn hàng – tìm kiếm đơn hàng.



Hình 4.1.34. Giao diện trang chi tiết đơn hàng.

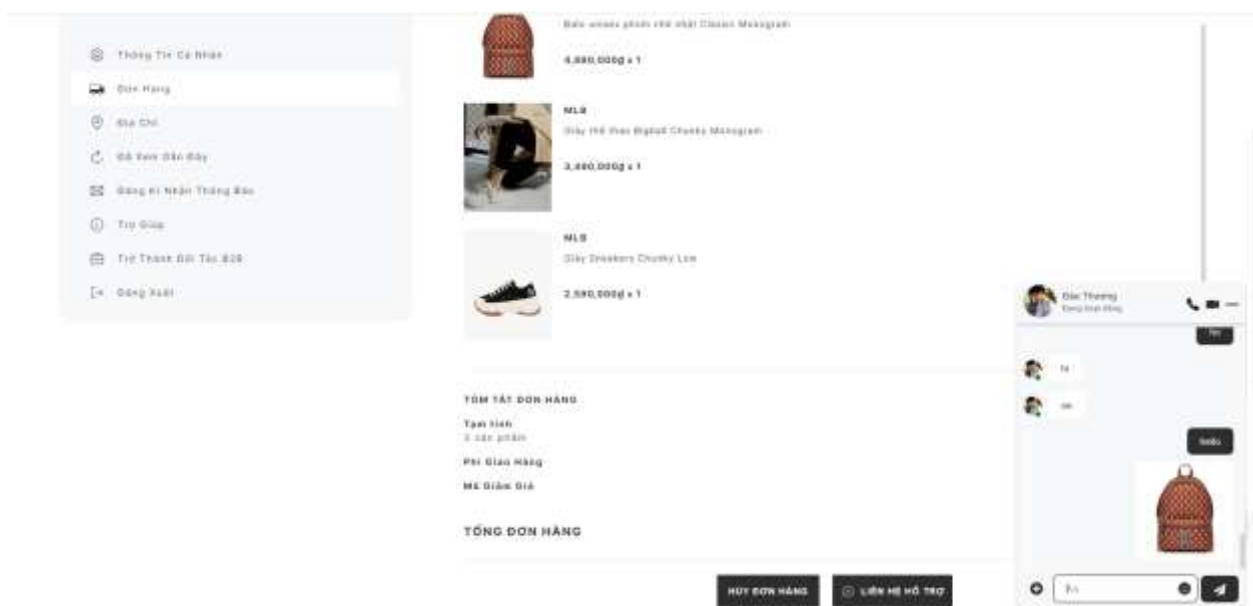


Hình 4.1.35. Giao diện trang chi tiết đơn hàng.



Hình 4.1.36. Giao diện hủy đơn hàng.

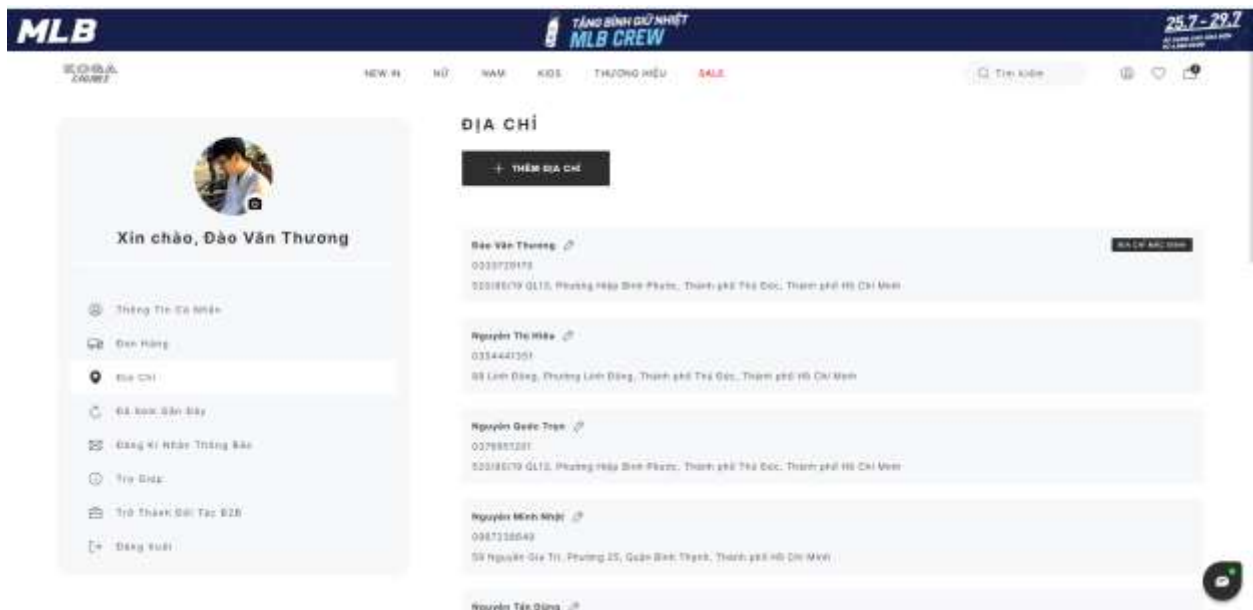
Khi đơn đặt hàng đang ở trạng thái “đang xử lý” thì người dùng mới có thể hủy được đơn hàng.



Hình 4.1.37. Giao diện chatbox được hiển thị.

Nếu người dùng muốn liên hệ quản trị viên để được hỗ trợ thì có thể click vào button “LIÊN HỆ HỖ TRỢ” hoặc click vào button “CHAT” ở bên phải góc dưới màn hình.

4.1.10. Giao diện danh sách địa chỉ giao hàng



Hình 4.1.38. Giao diện danh sách địa chỉ giao hàng.



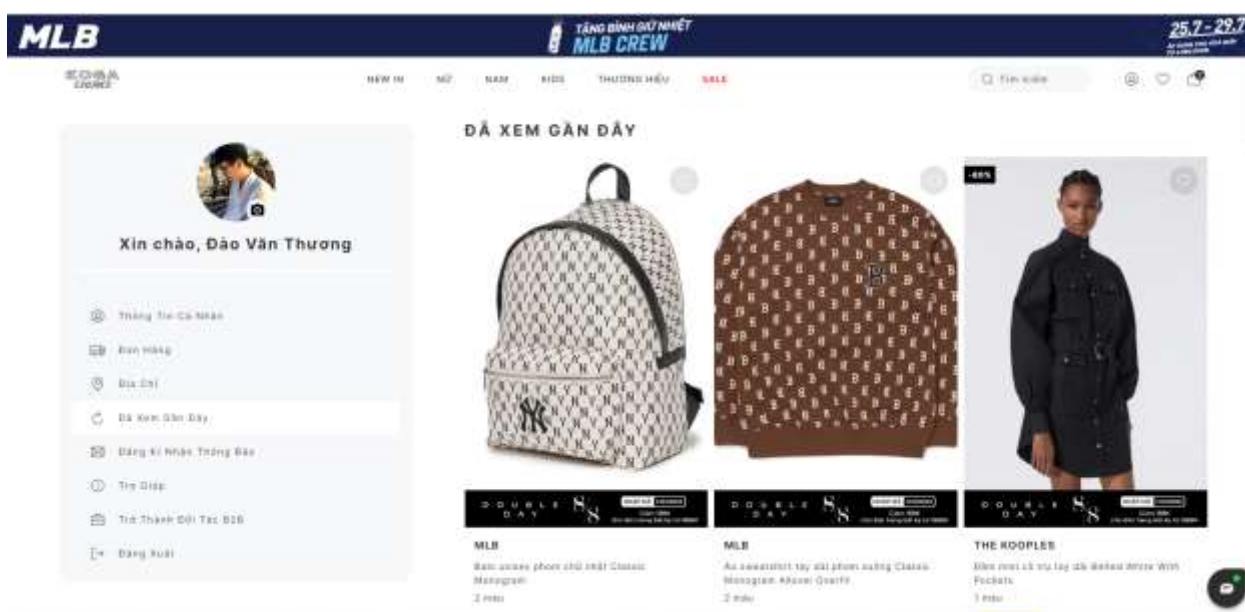
Hình 4.1.39. Giao diện thêm mới địa chỉ giao hàng.

Khi người dùng muốn thêm địa chỉ giao hàng mới, thì có thể click vào button “Thêm Địa Chỉ” để thêm mới.



Hình 4.1.40. Giao diện cập nhật địa chỉ giao hàng.

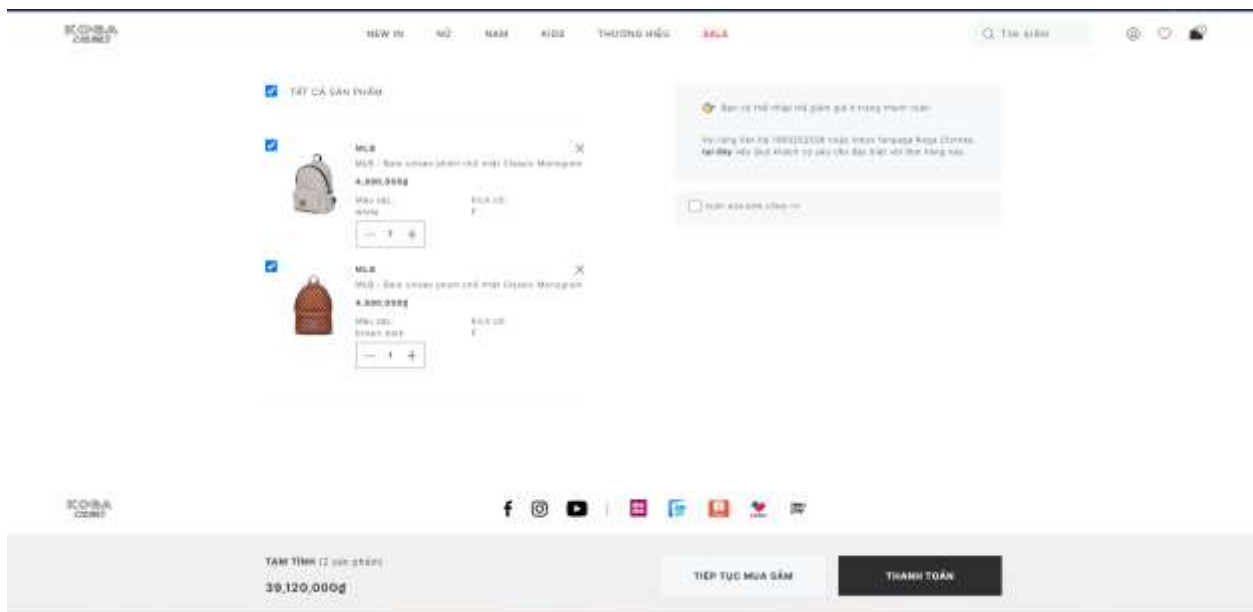
4.1.11. Giao diện đã xem gần đây



Hình 4.1.41. Giao diện các sản phẩm đã xem gần đây.

Phần giao diện này sẽ hiển thị một danh sách các sản phẩm đã xem, trong trường hợp có quá nhiều sản phẩm người dùng muốn xem lại sản phẩm đã xem nhưng không tìm được thì có thể dùng chức năng này để tìm lại một cách dễ dàng.

4.1.12. Giao diện giỏ hàng



Hình 4.1.42. Giao diện trang giỏ hàng.

Tại đây người dùng có thể xem tất cả sản phẩm đã bỏ vào giỏ hàng, ngoài ra còn có một khung view tổng giá trị sản phẩm mà người dùng tick chọn, góc trên sẽ có button để chọn tất cả sản phẩm có trong giỏ.

Phía dưới hiển thị giá tiền các sản phẩm đã tick kèm theo hai nút “Tiếp tục mua sắm” và tiến hành “Thanh toán” để tối ưu trải nghiệm của người dùng.

4.1.13. Giao diện danh sách sản phẩm yêu thích



Hình 4.1.43. Giao diện danh sách các sản phẩm đã yêu thích.

Sau khi click vào button “yêu thích” sẽ hiện ra một cửa sổ danh sách các sản phẩm đã thêm vào mục yêu thích, danh sách này sẽ bao gồm các thông tin cơ bản của sản phẩm đã yêu thích. Ngoài ra button “Chuyển tất cả vào giỏ hàng” đc thêm vào danh sách này để

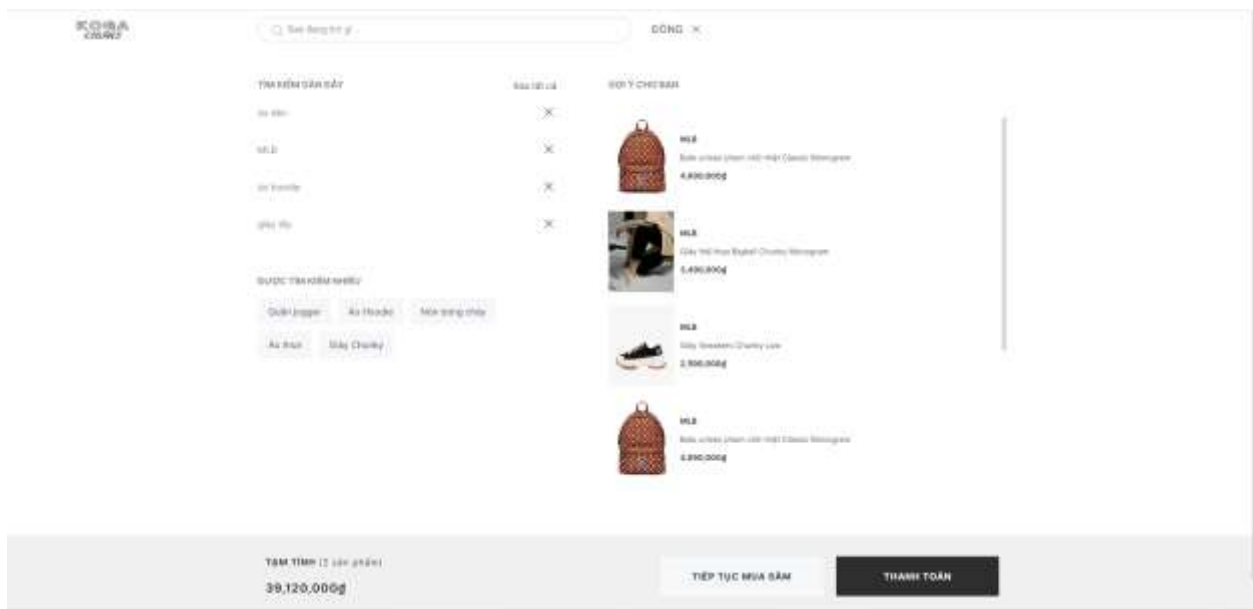
người dùng có thể dễ dàng thanh toán sau khi cân nhắc các sản phẩm trong danh sách yêu thích này.



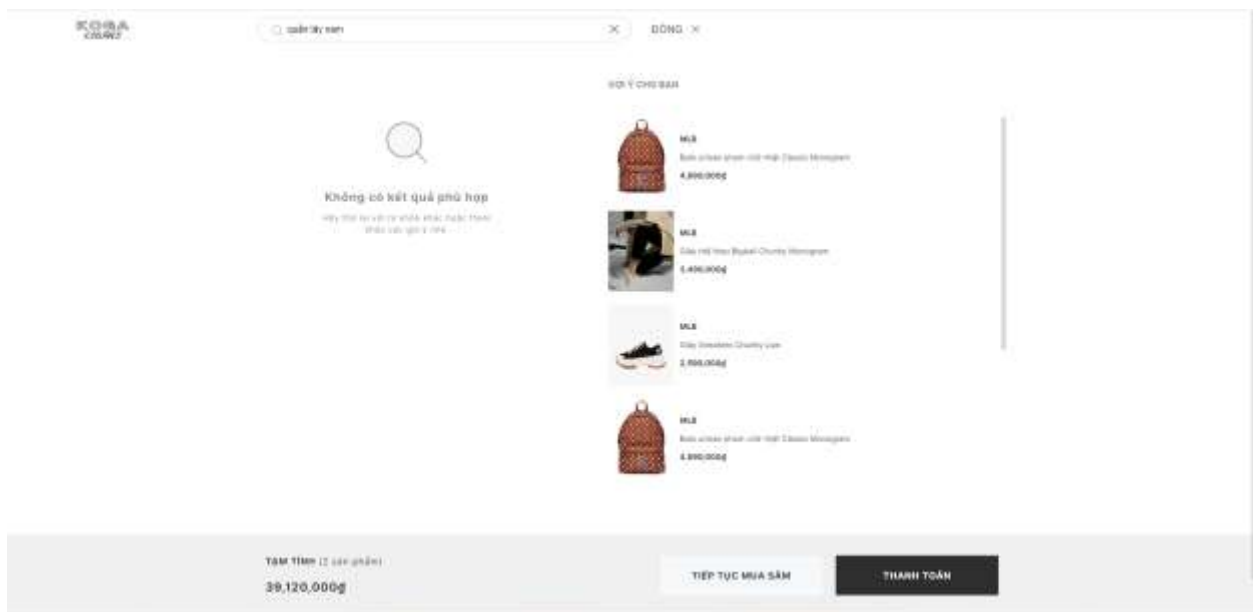
Hình 4.1.44. Giao diện danh sách các sản phẩm đã yêu thích.

Khi người dùng muốn xóa sản phẩm khỏi danh sách yêu thích thì sẽ có 2 lựa chọn “Chuyển Vào Giỏ Hàng” hoặc “Xóa”.

4.1.14. Giao diện tìm kiếm sản phẩm



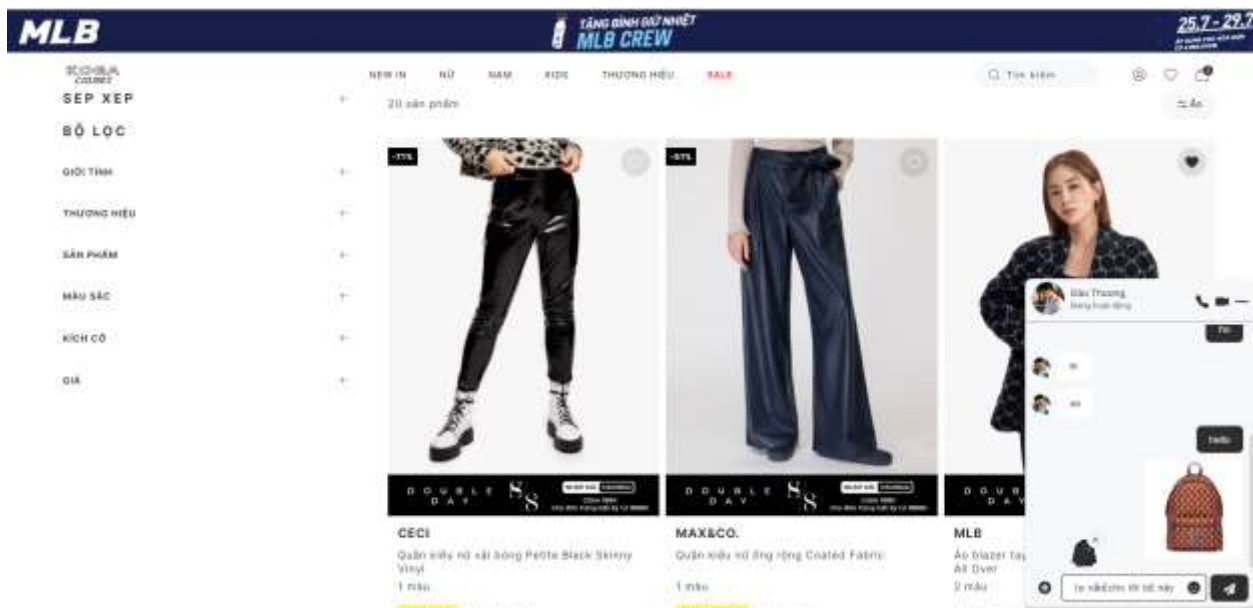
Hình 4.1.45. Giao diện tìm kiếm sản phẩm.



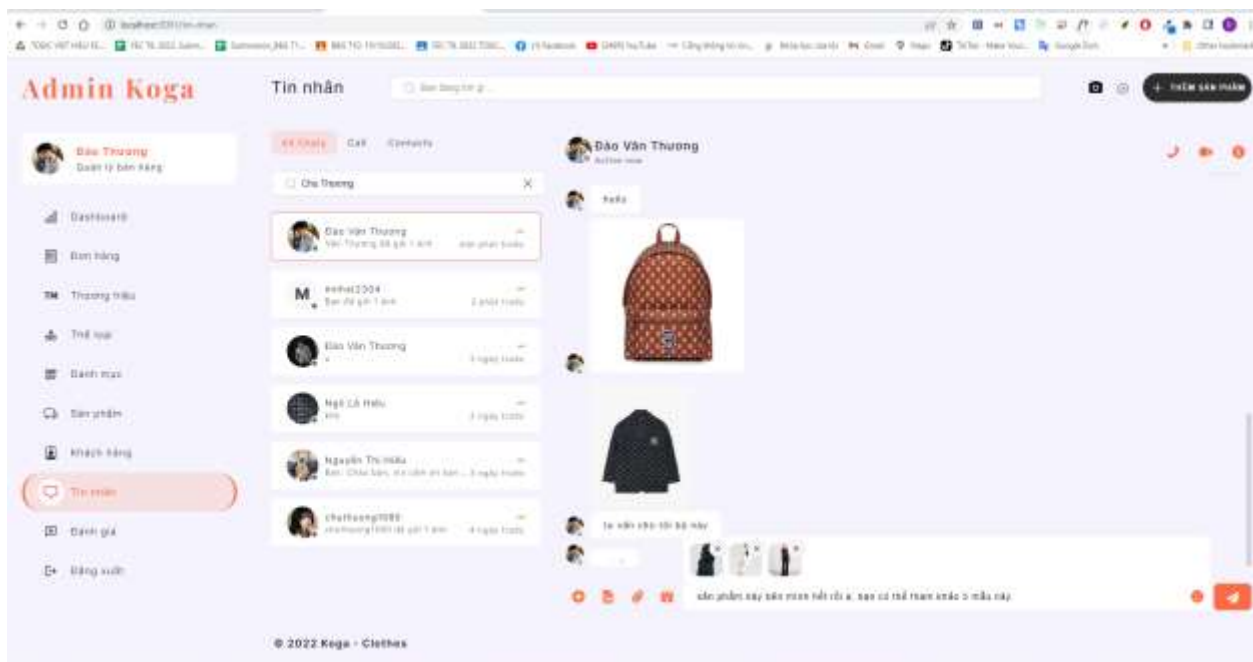
Hình 4.1.46. Giao diện tìm kiếm sản phẩm khi không có kết quả.

4.1.15. Giao diện trang đặt hàng

Hình 4.1.47. Giao diện trang đặt hàng.

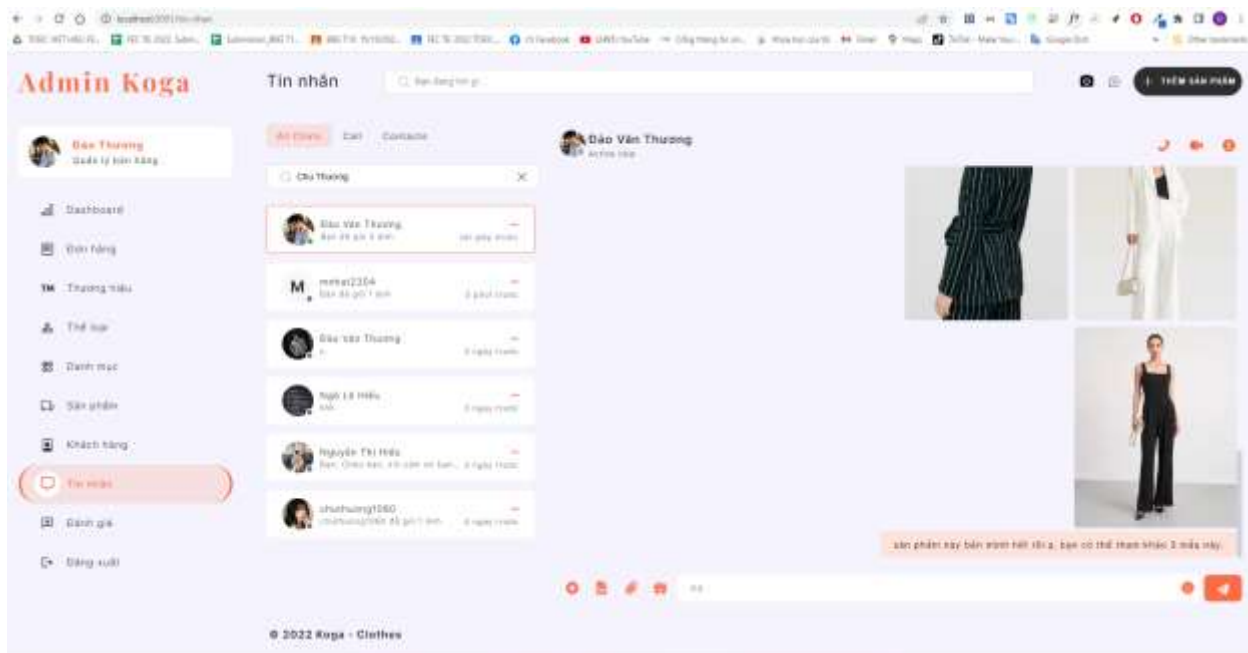


Hình 4.1.50. Giao diện nhắn tin bên khách hàng.



Hình 4.1.51. Giao diện nhắn tin bên quản trị viên.

Phần nhắn tin này, người dùng có thể liên hệ với quản trị viên hoặc người bán để được hỗ trợ. Biểu tượng loading bên trong tin nhắn cho biết người dùng đang nhập tin nhắn để gửi tới quản trị viên.



Hình 4.1.52. Giao diện nhắn tin bên quản trị viên.

4.2. Triển khai Backend

4.2.1. Xây dựng các models

Bảng 4.2.1. Model User

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	email	String	Tên đăng nhập, là thuộc tính bắt buộc và là duy nhất.
2	username	String	Tên đăng nhập, là thuộc tính bắt buộc và là duy nhất.
3	password	String	Mật khẩu là thuộc tính bắt buộc.
4	isAdmin	Boolean	Phân quyền user có phải là quản trị viên hay không, giá trị mặc định là “false”.
5	avatar	String	Là ảnh đại diện của người dùng.
6	firstName	String	Là họ và tên lót của người dùng.
7	lastName	String	Là tên của người dùng.
8	phone	String	Là số điện thoại của người dùng.
9	gender	String	Là giới tính của người dùng.
10	dateOfBirth	String	Là ngày sinh của người dùng.
11	cart	ObjectId	Là một ObjectId tham chiếu đến đối tượng “Cart”.

12	orders	Array	Là một mảng chứa các ObjectId tham chiếu đến các đối tượng “Order”.
13	reviews	Array	Là một mảng chứa các ObjectID tham chiếu đến các đối tượng “Review”.
14	favorites	Array	Là một mảng các đối tượng “Favorite”.
15	addresses	Array	Là một mảng các đối tượng “Address”.
16	loggedOut	Boolean	Cho biết người dùng này đã đăng xuất hay chưa, giá trị mặc định là “true”
17	loggedOutAt	Date	Thời gian người dùng đăng xuất khỏi trang web.
18	createdAt	Date	Thời gian tài khoản được tạo.
19	updatedAt	Date	Thời gian tài khoản được cập nhật.

Bảng 4.2.2. Model Address

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	firstName	String	Là họ và tên lót của người nhận đơn hàng.
2	lastName	String	Là tên của người nhận đơn hàng.
3	phone	String	Là số điện thoại để liên hệ nhận hàng.
4	province	String	Là tỉnh hoặc thành phố của địa chỉ.
5	district	String	Là quận hoặc huyện của địa chỉ.
6	ward	String	Là phường hoặc xã của địa chỉ.
7	address	String	Là địa chỉ cụ thể để giao hàng.
8	isDefault	Boolean	Là địa chỉ mặc định khi đặt hàng, mặc định là “false”.
9	createdAt	Date	Thời gian tạo địa chỉ.
10	updatedAt	Date	Thời gian địa chỉ được cập nhật.

Bảng 4.2.3. Model Favorite

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
-----	----------------	--------------	-------

1	product	ObjectId	Là 1 ObjectId tham chiếu đến đối tượng “Product”.
2	size	String	Là thông tin kích cỡ của sản phẩm được yêu thích.
3	color	String	Là thông tin màu sắc của sản phẩm được yêu thích.
4	quantity	Number	Là số lượng sản phẩm, mặc định là 1.

Bảng 4.2.4. Model Session

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	user	ObjectId	Là 1 ObjectId tham chiếu đến đối tượng “User”.
2	valid	Boolean	Sau khi user đăng nhập vào thì sẽ tạo ra 1 đối tượng Session, và giá trị là “true”. Khi user đăng xuất khỏi trang web thì sẽ cập nhật lại Session này là “false”.
3	userAgent	String	Là tác nhân người dùng. Cho biết thông tin về trình duyệt và hệ điều hành.
4	createdAt	Date	Thời gian user đăng nhập vào hệ thống.
5	updatedAt	Date	Thời gian user đăng xuất khỏi hệ thống.

Bảng 4.2.5. Model Catalog

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	name	String	Là tên của mục lục sản phẩm, bắt buộc và là duy nhất.
2	categories	Array	Là một mảng các ObjectId ánh xạ đến các đối tượng “Category”.
3	slug	String	Là một thuật ngữ dùng để mô tả tên “Catalog” trên URL.
5	createdAt	Date	Thời gian tạo catalog.

6	updatedAt	Date	Thời gian cập nhật catalog.
---	-----------	------	-----------------------------

Bảng 4.2.6. Model Category

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	name	String	Là tên của danh mục sản phẩm, bắt buộc và là duy nhất.
2	products	Array	Là một mảng các ObjectId ánh xạ đến các đối tượng “Product”.
3	catalog	ObjectId	Là một ObjectId ánh xạ đến đối tượng “Catalog”.
4	gender	Array	Là một mảng chứa các giới tính dành cho danh mục này.
5	slug	String	Là một thuật ngữ dùng để mô tả tên “Category” trên URL.
6	createdAt	Date	Thời gian tạo danh mục.
7	updatedAt	Date	Thời gian cập nhật danh mục.

Bảng 4.2.7. Model Brand

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	name	String	Là tên của Thương hiệu, bắt buộc và là duy nhất.
2	products	Array	Là một mảng các ObjectId ánh xạ đến các đối tượng “Product”.
3	logo	String	Là hình ảnh logo của Thương hiệu.
4	image	String	Là hình ảnh nổi bật của Thương hiệu.
4	history	String	Là lịch sử hình thành của Thương hiệu.
5	slug	String	Là một thuật ngữ dùng để mô tả tên “Brand” trên URL.
6	createdAt	Date	Thời gian tạo Thương hiệu.

7	updatedAt	Date	Thời gian cập nhật Thương hiệu.
---	-----------	------	---------------------------------

Bảng 4.2.8. Model Product

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	name	String	Là tên của sản phẩm, bắt buộc và là duy nhất.
2	description	String	Là mô tả về sản phẩm.
3	price	Number	Là giá của sản phẩm.
4	discount	Number	Là số phần trăm giảm giá cho sản phẩm.
5	category	ObjectId	Là một ObjectId tham chiếu đến đối tượng “Category”.
6	likeCount	Number	Là số lượng yêu thích sản phẩm.
7	quantitySold	Number	Là số sản phẩm đã bán.
8	favorites	Array	Là một mảng các đối tượng “User”.
9	rate	Number	Là số sao trung bình của sản phẩm đã được đánh giá.
10	keywords	Array	Là một mảng chứa những keyword để người dùng có thể dễ dàng tìm kiếm.
11	reviews	Array	Là một mảng chứa các ObjectId đánh giá tham chiếu đến các đối tượng “Review”.
12	colors	Array	Là một mảng các đối tượng “Color”.
13	brand	ObjectId	Là một ObjectId của Thương hiệu tham chiếu đến đối tượng “Brand”.
14	gender	String	Là giới tính được dành cho sản phẩm này.
15	preserveInformation	String	Là thông tin bảo quản của sản phẩm.
16	deliveryReturnPolicy	String	Là chính sách đổi trả của sản phẩm.
5	slug	String	Là một thuật ngữ dùng để mô tả tên “Product” trên URL.
6	createdAt	Date	Thời gian tạo sản phẩm.

7	updatedAt	Date	Thời gian cập nhật sản phẩm.
---	-----------	------	------------------------------

Bảng 4.2.8. Model Color

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	images	Array	Là một mảng chứa các hình ảnh chính của sản phẩm.
2	imageSmall	String	Là hình ảnh đặc trưng của sản phẩm.
3	imageMedium	String	Là hình ảnh tượng trưng cho màu của sản phẩm.
4	colorName	String	Là tên màu sắc của sản phẩm.
5	sizes	Array	Là một mảng chứa các thông tin kích cỡ và số lượng tồn của sản phẩm theo màu này.

Bảng 4.2.9. Model Cart

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	user	ObjectId	Là một ObjectId tham chiếu đến đối tượng “User”.
2	cartItems	Array	Là một mảng các đối tượng “CartItem”
3	createdAt	Date	Thời gian tạo giỏ hàng.
4	updatedAt	Date	Thời gian cập nhật giỏ hàng.

Bảng 4.2.10. Model CartItem

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	product	ObjectId	Là một ObjectId tham chiếu đến đối tượng “Product”.
2	quantity	Number	Là số lượng sản phẩm của Item trong Giỏ hàng.

3	size	String	Là kích cỡ sản phẩm của Item trong Giỏ hàng.
4	color	String	Là tên màu sắc của sản phẩm trong CartItem
5	image	Array	Là một mảng chứa các thông tin kích cỡ và số lượng tồn của sản phẩm theo màu này.
6	createdAt	Date	Thời gian thêm CartItem.
7	updatedAt	Date	Thời gian cập nhật CartItem.

Bảng 4.2.11. Model Order

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	orderId	String	Là ID của đơn hàng.
2	user	ObjectId	Là một ObjectId của User tham chiếu đến đối tượng “User”.
3	orderItems	Array	Là một mảng các đối tượng “OrderItem”
4	deliveryInformation	Object	Là một đối tượng “Address” chứa thông tin giao hàng của đơn hàng.
5	taxPrice	Array	Là một mảng chứa các thông tin kích cỡ và số lượng tồn của sản phẩm theo màu này.
6	shippingPrice	Number	Là phí vận chuyển đơn hàng.
7	orderStatus	String	Là trạng thái của đơn hàng.
8	totalPrice	Number	Là tổng giá tiền của đơn hàng.
9	provisionalPrice	Number	Là tổng giá tiền của các sản phẩm trong đơn hàng
10	isPaid	Boolean	Là “true” nếu người dùng đã thanh toán.
11	paidAt	Date	Thời gian thanh toán đơn hàng.
12	shippingAt	Date	Thời gian đơn hàng được vận chuyển.
13	deliveryAt	Date	Thời gian đơn hàng đang được giao.

14	deliveredAt	Date	Thời gian đơn hàng đã được giao cho khách hàng.
15	canceledAt	Date	Thời gian đơn hàng bị hủy.
16	canceledReason	String	Lý do hủy đơn hàng.
17	isEvaluated	Boolean	Là “true” nếu đơn hàng đã được đánh giá.
18	createdAt	Date	Thời gian đơn hàng được tạo
19	updatedAt	Date	Thời gian đơn hàng được cập nhật.

Bảng 4.2.12. Model OrderItem

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	product	ObjectId	Là một ObjectId tham chiếu đến đối tượng “Product”.
2	brandName	String	Là tên của Thương hiệu.
3	image	String	Là hình ảnh chính của sản phẩm.
4	quantity	Number	Là số lượng sản phẩm.
5	size	String	Là kích cỡ sản phẩm.
6	color	String	Là màu sắc của sản phẩm.
7	discount	Number	Giá trị giảm giá của sản phẩm.
8	price	Number	Giá của sản phẩm.
9	name	String	Tên sản phẩm

Bảng 4.2.13. Model Review

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	user	ObjectId	Là một ObjectId tham chiếu đến đối tượng “User”.
2	product	ObjectId	Là một mảng các đối tượng “Product”

3	orderedProductDetail	Object	Là một đối tượng chứa các thuộc tính về sản phẩm đã đánh giá (size, color, quantity).
4	content	String	Nội dung đánh giá
5	star	Number	Số lượng sao được đánh giá, nhỏ nhất là 1 và lớn nhất là 5.
3	createdAt	Date	Thời gian tạo đánh giá.
4	updatedAt	Date	Thời gian cập nhật đánh giá.

Bảng 4.2.14. Model Conversation

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	members	Array	Là một mảng các đối tượng “User” trong cuộc hội thoại, ở trong phạm vi này, chỉ có khách hàng nhắn tin với quản trị viên nên mảng này sẽ có 2 đối tượng.
3	createdAt	Date	Thời gian tạo cuộc hội thoại.
4	updatedAt	Date	Thời gian cập nhật cuộc hội thoại.

Bảng 4.2.15. Model Message

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	conversation	ObjectId	Là một ObjectId tham chiếu đến đối tượng “Conversation”.
2	sender	ObjectId	Là người gửi tin nhắn, là một ObjectId tham chiếu đến đối tượng “User”.
3	text	String	Là nội dung tin nhắn.
4	images	Array	Là một mảng chứa tất cả các hình ảnh của tin nhắn (nếu có).
5	seen	Boolean	Là trạng thái đã xem tin nhắn, mặc định là “false”.
6	seenAt	Date	Thời gian đã xem tin nhắn

3	createdAt	Date	Thời gian tạo tin nhắn.
4	updatedAt	Date	Thời gian cập nhật tin nhắn.

4.2.2. Xây dựng các routes

➤ Xây dựng route user

- **POST /api/v1/users/register** : Đăng ký tài khoản.

```
// * REGISTER USER
// POST /api/v1/users/register
router.post(
  "/users/register",
  validateRequest(createUserSchema),
  createUserHandler
);
```

- **POST /api/v1/users/login**: Đăng nhập.

```
// * LOGIN USER
// POST /api/v1/users/login
router.post(
  "/users/login",
  validateRequest(createUserSessionSchema),
  createUserSessionHandler
);
```

- **POST /api/v1/auth/google**: Đăng nhập với tài khoản Google.

```
// * LOGIN WITH GOOGLE
// POST /api/v1/auth/google
router.post("/auth/google", googleLoginHandler);
```

- **DELETE /api/v1/users/logout**: Đăng xuất.

```
// * LOGOUT
// DELETE /api/v1/users/logout
router.delete("/users/logout", requiresUser,
  invalidateUserSessionHandler);
```


- **POST /api/v1/password/forgot:** Lấy lại mật khẩu.

```
// * FORGOT PASSWORD
// POST /api/v1/password/forgot
router.post(
  "/password/forgot",
  validateRequest(createForgotPassword),
  forgotPasswordHandler
);
```

- **POST /api/v1/password/update:** Thay đổi mật khẩu.

```
// * CHANGE PASSWORD
// POST /api/v1/password/update
router.post(
  "/password/update",
  [requiresUser, validateRequest(createChangePasswordSchema)],
  changePasswordHandler
);
```

- **GET /api/v1/me:** Lấy thông tin cá nhân.

```
// * GET PROFILE
// GET /api/v1/me
router.get("/me", requiresUser, getProfileHandler);
```

- **PUT /api/v1/me/update:** Cập nhật thông tin cá nhân.

```
// * UPDATE PROFILE
// PUT /api/v1/me/update
router.put(
  "/me/update",
  [requiresUser, validateRequest(updateUserSchema)],
  updateProfileHandler
);
```

- **POST /api/v1/me/addresses/add:** Thêm thông tin địa chỉ giao hàng.

```
// * ADD ADDRESS
// POST /api/v1/me/addresses/add
```

```
router.post(
  "/me/addresses/add",
  [requiresUser, validateRequest(addAddressSchema)],
  addAddressHandler
);
```

- **PUT /api/v1/me/addresses/:addressId:** Cập nhật địa chỉ giao hàng.

```
// * UPDATE ADDRESS
// POST /api/v1/me/addresses/:addressId
router.put(
  "/me/addresses/:addressId",
  [requiresUser, validateRequest(updateAddressSchema)],
  updateAddressHandler
);
```

- **DELETE /api/v1/me/addresses/:addressId:** Xóa địa chỉ giao hàng.

```
// * DELETE ADDRESS
// DELETE /api/v1/me/addresses/:addressId
router.delete(
  "/me/addresses/:addressId",
  [requiresUser, validateRequest(updateAddressSchema)],
  deleteAddressHandler
);
```

- **GET /api/v1/sessions:** Lấy tất cả session đang tồn tại.

```
// * GET THE USER'S SESSIONS ---- ADMIN
// GET /api/v1/sessions
router.get("/sessions", requiresAdmin, getUserSessionHandler);
```

- **GET /api/v1/admin/users:** Lấy tất cả user.

```
// * GET ALL USER ---- ADMIN
// GET /api/v1/admin/users
router.get("/admin/users", requiresAdmin, getAllUserHandler);
```

- **GET /api/v1/admin/user/:userId:** Lấy thông tin user theo userID.

```
// * GET A USER ---- ADMIN
// GET /api/v1/admin/user/:userId
router.get("/admin/user/:userId", requiresAdmin, getUserHandler);
```

- **PUT /api/v1/admin/user/:userId:** Cập nhật thông tin user theo userID.

```
// * UPDATE USER ROLE ----- ADMIN
// PUT /api/v1/admin/user/:userId
router.put("/admin/user/:userId", requiresAdmin, updateUserRoleHandler);
```

- **DELETE /api/v1/admin/user/:userId:** Xóa thông tin user theo userID.

```
// * DELETE USER ----- ADMIN
// DELETE /api/v1/admin/user/:userId
router.delete("/admin/user/:userId", requiresAdmin, deleteUserHandler);
```

➤ Xây dựng route Catalog

- **POST /api/v1/admin/catalog/new:** Tạo mục lục sản phẩm bởi Admin.

```
// * CREATE CATALOG
// POST /api/v1/admin/catalog/new
router.post(
  "/admin/catalog/new",
  [requiresAdmin, validateRequest(createCatalogSchema)],
  createCatalogHandler
);
```

- **GET /api/v1/catalogs:** Lấy thông tin tất cả mục lục sản phẩm.

```
// * GET ALL CATALOGS
// GET /api/v1/catalogs
router.get("/catalogs", getAllCatalogHandler);
```

- **GET /api/v1/catalog/:catalogId:** Lấy thông tin chi tiết của catalog.

```
// * GET CATALOG DETAILS
// GET /api/v1/catalog/:catalogId
router.get(
  "/catalog/:catalogId",
```

```

    validateRequest(getCatalogSchema),
    getCatalogHandler
  );

```

- **PUT /api/v1/admin/catalog/:catalogId:** Cập nhật catalog.

```

// * UPDATE CATALOG
// PUT /api/v1/admin/catalog/:catalogId
router.put(
  "/admin/catalog/:catalogId",
  [requiresAdmin, validateRequest(updateCatalogSchema)],
  updateCatalogHandler
);

```

- **DELETE /api/v1/admin/catalog/:catalogId:** Xóa catalog.

```

// * DELETE CATALOG
// DELETE /api/v1/admin/catalog/:catalogId
router.delete(
  "/admin/catalog/:catalogId",
  [requiresAdmin, validateRequest(deleteCatalogSchema)],
  deleteCatalogHandler
);

```

➤ Xây dựng route Category

- **GET api/v1/categories:** Lấy tất cả danh mục sản phẩm.

```

// * GET ALL CATEGORY
// GET api/v1/categories
router.get("/categories", getAllCategoryHandler);

```

- **GET api/v1/category/:categoryId:** Lấy thông tin chi tiết của danh mục sản phẩm.

```

// * GET CATEGORY DETAILS
// GET api/v1/category/:categoryId
router.get(
  "/category/:categoryId",
  validateRequest(getCategorySchema),
  getCategoryHandler
);

```

- **POST api/v1/admin/category/new:** Tạo mới danh mục sản phẩm bởi Admin.

```
// * CREATE CATEGORY --- ADMIN
// POST api/v1/admin/category/new
router.post(
  "/admin/category/new",
  [requiresAdmin, validateRequest(createCategorySchema)],
  createCategoryHandler
);
```

- **PUT api/v1/admin/category/:categoryId:** Cập nhật danh mục sản phẩm.

```
// * UPDATE CATEGORY --- ADMIN
// PUT api/v1/admin/category/:categoryId
router.put(
  "/admin/category/:categoryId",
  [requiresAdmin, validateRequest(updateCategorySchema)],
  updateCategoryHandler
);
```

- **DELETE api/v1/admin/category/:categoryId:** Xóa danh mục sản phẩm.

```
// * DELETE CATEGORY --- ADMIN
// DELETE api/v1/admin/category/:categoryId
router.delete(
  "/admin/category/:categoryId",
  [requiresAdmin, validateRequest(deleteCategorySchema)],
  deleteCategoryHandler
);
```

➤ Xây dựng route Brand

- **POST /api/v1/admin/brand/new:** Tạo mới một Thương hiệu.

```
// * CREATE BRAND ---- ADMIN
// POST /api/v1/admin/brand/new
router.post(
  "/admin/brand/new",
  [requiresAdmin, validateRequest(createBrandSchema)],
  createBrandHandler
);
```

```
createBrandHandler  
);
```

- **GET /api/v1/brands:** Lấy tất cả thông tin Thương hiệu.

```
// * GET ALL BRAND  
// GET /api/v1/brands  
router.get("/brands", getAllBrandHandler);
```

- **GET /api/v1/brand/:brandId:** Lấy thông tin chi tiết của Thương hiệu.

```
// * GET BRAND DETAILS  
// GET /api/v1/brand/:brandId  
router.get("/brand/:brandId", validateRequest(getBrandSchema),  
getBrandHandler);
```

- **PUT /api/v1/admin/brand/:brandId:** Cập nhật thông tin Thương hiệu.

```
// * UPDATE BRAND  
// PUT /api/v1/admin/brand/:brandId  
router.put(  
  "/admin/brand/:brandId",  
  [requiresAdmin, validateRequest(updateBrandSchema)],  
  updateBrandHandler  
);
```

- **DELETE /api/v1/admin/brand/:brandId:** Xóa thương hiệu.

```
// * DELETE BRAND  
// DELETE /api/v1/admin/brand/:brandId  
router.delete(  
  "/admin/brand/:brandId",  
  [requiresAdmin, validateRequest(deleteBrandSchema)],  
  deleteBrandHandler  
);
```

➤ Xây dựng route Product

- **POST /api/v1/admin/product/new:** Tạo mới sản phẩm.

```
// * CREATE PRODUCT ---- ADMIN
```

```
// POST /api/v1/admin/product/new
router.post(
  "/admin/product/new",
  [requiresAdmin, validateRequest(createProductSchema)],
  createProductHandler
);
```

- **GET /api/v1/products:** Lấy tất cả thông tin sản phẩm.

```
// * GET ALL PRODUCTS
// GET /api/v1/products
router.get("/products", getAllProductHandler);
```

- **GET /api/v1/product/:productId:** Lấy thông tin sản phẩm theo ID sản phẩm.

```
// * GET PRODUCT DETAILS BY ID
// GET /api/v1/product/:productId
router.get(
  "/product/:productId",
  validateRequest(getProductSchema),
  getProductHandler);
```

- **GET /api/v1/product/slug/:slug:** Lấy thông tin sản phẩm theo slug.

```
// * GET PRODUCT DETAILS BY SLUG
// GET /api/v1/product/slug/:slug
router.get("/product/slug/:slug", getProductBySlugHandler);
```

- **PUT /api/v1/admin/product/:productId:** Cập nhật sản phẩm.

```
// * UPDATE PRODUCT ---- ADMIN
// PUT /api/v1/admin/product/:productId
router.put(
  "/admin/product/:productId",
  [requiresAdmin, validateRequest(updateProductSchema)],
  updateProductHandler
);
```

- **PUT /api/v1/products/favorite/add/:productId:** Thêm sản phẩm yêu thích.

```
// * ADD FAVORITE PRODUCTS
```

```
// PUT /api/v1/products/favorite/add/:productId
router.put(
  "/products/favorite/add/:productId",
  [requiresUser, validateRequest(updateProductSchema)],
  addFavoriteHandler
);
```

- **PUT /api/v1/products/favorite/remove/:productId:** Bỏ yêu thích sản phẩm.

```
// * REMOVE FAVORITE PRODUCTS
// PUT /api/v1/products/favorite/remove/:productId
router.put(
  "/products/favorite/remove/:productId",
  [requiresUser, validateRequest(updateProductSchema)],
  removeFavoriteHandler
);
```

- **PATCH /api/v1/admin/product/restore/:productId:** Khôi phục sản phẩm.

```
// * RESTORE PRODUCT
// PATCH /api/v1/admin/product/restore/:productId
router.patch(
  "/admin/product/restore/:productId",
  [requiresAdmin, validateRequest(restoreProductSchema)],
  restoreProductHandler
);
```

- **DELETE /api/v1/admin/product/:productId:** Xóa sản phẩm

```
// * SOFT DELETE PRODUCT
// DELETE /api/v1/admin/product/:productId
router.delete(
  "/admin/product/:productId",
  [requiresAdmin, validateRequest(deleteProductSchema)],
  deleteProductHandler
);
```

- **DELETE /api/v1/admin/product/force/:productId:** Xóa vĩnh viễn sản phẩm.


```
// * DELETE PRODUCT
// DELETE /api/v1/admin/product/force/:productId
router.delete(
  "/admin/product/force/:productId",
  [requiresAdmin, validateRequest(deleteProductSchema)],
  forceDestroyProductHandler
);
```

➤ Xây dựng route Cart

- **GET /api/v1/carts:** Lấy tất cả các giỏ hàng trong Database.

```
// * GET ALL CART
// GET /api/v1/carts
router.get("/carts", getAllCartHandler);
```

- **GET /api/v1/cart/me:** Lấy thông tin giỏ hàng.

```
// * GET MY CART
// GET /api/v1/cart/me
router.get("/cart/me", requiresUser, getMyCartHandler);
```

- **GET /api/v1/cart/:cartId:** Lấy thông tin giỏ hàng.

```
// * GET A CART
// GET /api/v1/cart/:cartId
router.get("/cart/:cartId", getCartHandler);
```

- **POST /api/v1/cart/add-to-cart:** Thêm item vào giỏ hàng.

```
// * ADD ITEM TO CART
// POST /api/v1/cart/add-to-cart
router.post(
  "/cart/add-to-cart",
  [requiresUser, validateRequest(addItemToCartSchema)],
  addItemToCartHandler
);
```

- **PUT /api/v1/cart/:cartItemId:** Cập nhật số lượng item trong giỏ hàng.

```
// * UPDATE CART --- update quantity
// PUT /api/v1/cart/:cartItemId
router.put(
  "/cart/:cartItemId",
  [requiresUser, validateRequest(updateCartSchema)],
  updateCartHandler
);
```

- **PUT /api/v1/cart/remove-item-from-cart/:cartItemId:** Xóa item khỏi giỏ hàng.

```
// * REMOVE ITEM FROM CART
// PUT /api/v1/cart/remove-item-from-cart/:cartItemId
router.put(
  "/cart/remove-item-from-cart/:cartItemId",
  [requiresUser, validateRequest(removeItemFromCartSchema)],
  removeItemFromCartHandler
);
```

- **DELETE /api/v1/cart/:cartItemId:** Xóa giỏ hàng

```
// * DELETE CART
// DELETE /api/v1/cart/:cartItemId
router.delete(
  "/cart/:cartId",
  [requiresAdmin, validateRequest(deleteCartSchema)],
  deleteCartHandler
);
```

➤ Xây dựng route Order

- **POST /api/v1/order/new:** Tạo mới đơn hàng.

```
// * CREATE NEW ORDER
// POST /api/v1/order/new
router.post(
  "/order/new",
  [requiresUser, validateRequest(createOrderSchema)],
  createOrderHandler
);
```

- **GET /api/v1/order/me:** Lấy tất cả các đơn hàng của user.

```
// * GET MY ORDER
// GET /api/v1/order/me
router.get("/order/me", requiresUser, getMyOrderHandler);
```

- **GET /api/v1/order/:orderId:** Lấy thông tin chi tiết của đơn hàng theo orderId.

```
// * GET ORDER
// GET /api/v1/order/:orderId
router.get(
  "/order/:orderId",
  [requiresUser, validateRequest(getOrderSchema)],
  getOrderHandler
);
```

- **GET /api/v1/admin/orders:** Lấy tất cả đơn hàng.

```
// * GET ALL ORDER
// GET /api/v1/admin/orders
router.get("/admin/orders", requiresAdmin, getAllOrderHandler);
```

- **PUT /api/v1/admin/order/:orderId:** Cập nhật trạng thái đơn hàng bởi Admin.

```
// * UPDATE ORDER STATUS
// PUT /api/v1/admin/order/:orderId
router.put(
  "/admin/order/:orderId",
  [requiresAdmin, validateRequest(updateOrderSchema)],
  updateStatusOrderHandler
);
```

- **PUT /api/v1/order/cancel/:orderId:** Hủy đơn đặt hàng.

```
// * CANCEL ORDER
// PUT /api/v1/order/cancel/:orderId
router.put(
  "/order/cancel/:orderId",
  [requiresUser, validateRequest(cancelOrderSchema)],
  cancelOrderHandler
);
```

```
);
```

- **DELETE /api/v1/admin/order/:orderId:** Xóa đơn hàng.

```
// * DELETE ORDER
// DELETE /api/v1/admin/order/:orderId
router.delete(
  "/admin/order/:orderId",
  [requiresAdmin, validateRequest(deleteOrderSchema)],
  deleteOrderHandler
);
```

➤ Xây dựng route Review

- **POST /api/v1/review/new:** Tạo mới bài đánh giá sản phẩm.

```
// * CREATE NEW REVIEW
// POST /api/v1/review/new
router.post(
  "/review/new",
  [requiresUser, validateRequest(createReviewSchema)],
  createReviewHandler
);
```

- **GET /api/v1/reviews:** Lấy tất cả bài đánh giá sản phẩm.

```
// * GET ALL REVIEW
// GET /api/v1/reviews
router.get("/reviews", getAllReviewHandler);
```

- **GET /api/v1/review/:reviewId:** Lấy thông tin chi tiết bài đánh giá sản phẩm.

```
// * GET REVIEW
// GET /api/v1/review/:reviewId
router.get(
  "/review/:reviewId",
  validateRequest(getReviewSchema),
  getReviewHandler
);
```

- **GET /api/v1/reviews/:productId:** Lấy tất cả đánh giá sản phẩm theo productId.

```
// * GET ALL REVIEW BY PRODUCT
// GET /api/v1/reviews/:productId
router.get(
  "/reviews/:productId",
  validateRequest(getAllReviewByProductSchema),
  getAllReviewByProduct
);
```

- **PUT /api/v1/review/:reviewId:** Cập nhật đánh giá.

```
// * UPDATE REVIEW
// PUT /api/v1/review/:reviewId
router.put(
  "/review/:reviewId",
  [requiresUser, validateRequest(updateReviewSchema)],
  updateReviewHandler
);
```

- **PATCH /api/v1/review/restore/:reviewId:** Khôi phục đánh giá sản phẩm.

```
// * RESTORE REVIEW
// PATCH /api/v1/review/restore/:reviewId
router.patch(
  "/review/restore/:reviewId",
  [requiresUser, validateRequest(updateReviewSchema)],
  restoreReviewHandler
);
```

- **DELETE /api/v1/review/:reviewId:** Xóa đánh giá sản phẩm

```
// * SOFT DELETE REVIEW
// DELETE /api/v1/review/:reviewId
router.delete(
  "/review/:reviewId",
  [requiresUser, validateRequest(deleteReviewSchema)],
  destroyReviewHandler
);
```

```
);
```

- **DELETE /api/v1/review/force/:reviewId:** Xóa vĩnh viễn đánh giá sản phẩm

```
// * FORCE DELETE REVIEW
// DELETE /api/v1/review/force/:reviewId
router.delete(
  "/review/force/:reviewId",
  [requiresUser, validateRequest(deleteReviewSchema)],
  forceDestroyReviewHandler
);
```

➤ Xây dựng route Conversation

- **POST /api/v1/conversation/new:** Tạo mới một cuộc trò chuyện.

```
// * CREATE CONVERSATION
// POST /api/v1/conversation/new
router.post(
  "/conversations/new",
  [requiresUser, validateRequest(createConversationSchema)],
  createConversationHandler
);
```

- **GET /api/v1/conversations:** Lấy tất cả các cuộc trò chuyện của tôi.

```
// * GET MY CONVERSATION
// GET /api/v1/conversations
router.get("/conversations", requiresUser, getConversationHandler);
```

- **PUT /api/v1/conversations/:conversationId:** Cập nhật cuộc trò chuyện.

```
// * UPDATE CONVERSATION
// PUT /api/v1/conversations/:conversationId
router.put(
  "/conversations/:conversationId",
  [requiresUser, validateRequest(updateConversationSchema)],
  updateConversationHandler
);
```

➤ **Xây dựng route Message**

- **POST /api/v1/messages:** Tạo mới 1 tin nhắn

```
// * CREATE NEW MESSAGE
// POST /api/v1/messages
router.post(
  "/messages",
  [requiresUser, validateRequest(createMessageSchema)],
  createMessageHandler
);
```

- **GET /api/v1/messages/:conversationId:** Lấy tất cả tin nhắn của cuộc trò chuyện.

```
// * GET MESSAGES FROM CONVERSATION
// GET /api/v1/messages/:conversationId
router.get(
  "/messages/:conversationId",
  [requiresUser, validateRequest(getMessageSchema)],
  getMessageHandler
);
```

- **GET /api/v1/messages/latest/:conversationId:** Lấy tin nhắn mới nhất.

```
// * GET MESSAGE LATEST FROM CONVERSATION
// GET /api/v1/messages/latest/:conversationId
router.get(
  "/messages/latest/:conversationId",
  [requiresUser, validateRequest(getMessageSchema)],
  getMessageLatestHandler
);
```

- **PUT /api/v1/messages/seen/:conversationId/:receiverId:** Cập nhật trạng thái đã xem của tin nhắn.

```
router.put(
  "/messages/seen/:conversationId/:receiverId",
  [requiresUser, validateRequest(updateMessageSchema)],
  updateMessageHandler
);
```

➤ **Xây dựng route Upload hình ảnh**

- **POST /api/v1/upload/avatar:** Dùng để upload hình ảnh avatar và sẽ trả về thông tin file đã upload, sau đó lấy tên file cập nhật cho thuộc tính avatar của user.

```
router.post(
  "/upload/avatar",
  requiresUser,
  upload.single(FieldName.AVATAR),
  uploadSingleHandler
);
```

- **POST /api/v1/upload/brand:** Tương tự như upload avatar, dùng để upload các hình ảnh của Thương hiệu.

```
router.post(
  "/upload/brands",
  requiresAdmin,
  upload.single(FieldName.IMAGE_BRAND),
  uploadSingleHandler
);
```

- **POST /api/v1/upload/products:** Upload hình ảnh sản phẩm, gồm 3 field.

```
router.post(
  "/upload/products",
  requiresAdmin,
  upload.fields([
    { name: FieldName.IMAGE_LARGE, maxCount: 50 },
    { name: FieldName.IMAGE_SMALL, maxCount: 1 },
    { name: FieldName.IMAGE_MEDIUM, maxCount: 1 },
  ]),
  uploadMultiHandler
);
```

- **POST /api/v1/upload/messages:** Upload hình ảnh tin nhắn.

```
router.post(
  "/upload/messages",
  requiresUser,
```



```
upload.fields([ { name: fieldName.IMAGE_MESSAGES, maxCount: 10 } ]),
uploadMultiHandler
);
```

- **POST /api/v1/upload/catalogs:** Upload hình ảnh mục lục sản phẩm.

```
router.post(
  "/upload/catalogs",

  requiresAdmin,
  upload.fields([
    { name: fieldName.IMAGE_MEN, maxCount: 1 },
    { name: fieldName.IMAGE_WOMAN, maxCount: 1 },
    { name: fieldName.IMAGE_KID, maxCount: 1 },
  ]),
  uploadCatalogHandler
);
```

CHƯƠNG 5. TỔNG KẾT

5.1. Kết luận

Sau 2 tháng thực hiện dự án, mặc dù có nhiều khó khăn gặp phải nhưng chúng em đã cố hết sức để tìm hiểu và vượt qua khó khăn cũng như giải quyết được các vấn đề, cuối cùng dự án của chúng em cũng được hoàn tất.

Nhóm chúng em đã nghiên cứu, áp dụng được các công nghệ vào xây dựng một website trên nền tảng kiến trúc Client-Server. Thông qua việc tìm hiểu sâu được cách hoạt động và vận hành của hệ thống đã giúp chúng em làm chủ được công nghệ, trao dồi được kiến thức và các kỹ năng trong việc lập trình cũng như thiết kế hệ thống.

Áp dụng được các công nghệ tốt vào đồ án như React, Redux Toolkit để xử lý các thao tác trên website giúp hệ thống trở nên nhanh hiện đại hơn và quá trình coding cũng dễ dàng hơn.

Có cơ hội tìm hiểu và áp dụng NodeJS, ExpressJS, MongoDB vào dự án đầu tay giúp chúng em có những trải nghiệm tuyệt vời. Tuy nhiên, do thiếu kinh nghiệm và chưa có

kiến thức chuyên sâu về các công nghệ nên quá trình sử dụng còn khá sai sót, hạn chế, chưa tối ưu.

Bên cạnh phát triển dự án, khả năng phân tích, nghiên cứu cũng như khả năng tự học, đọc tài liệu của chúng em được cải thiện rất nhiều.

5.2. Những khó khăn gặp phải

- Áp lực về thời gian do chưa có nhiều kinh nghiệm.
- Khả năng phân bổ thời gian chưa được tối ưu.
- Thiếu kỹ năng quản lý code.
- Chưa hoàn thành dự án hoàn hảo theo ý tưởng nghĩ ra.

5.3. Hướng phát triển

Mục tiêu ngắn hạn: Tối ưu hệ thống

- Tối ưu được giao diện giúp người dùng dễ dàng sử dụng hơn.
- Phân chia các component một cách hợp lý.
- Nâng cấp được các API để hoàn thiện hơn hệ thống.
- Giao diện admin phải hoàn chỉnh hơn, có nhiều chức năng hơn.

Mục tiêu dài hạn: Nâng cấp hệ thống

- Xây dựng được ứng dụng trên thiết bị di động đa nền tảng với công nghệ React Native.
- Nâng cấp website thành Web 3.0.
- Có thể liên kết được với các bên thứ 3 để hệ thống được biết đến nhiều hơn.