

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CẦN THƠ
TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**



**NIÊN LUẬN
NGÀNH CÔNG NGHỆ THÔNG TIN**

**Đề tài
XÂY DỰNG HỆ THỐNG
MẠNG XÃ HỘI**

Sinh viên: Lê Dương Trí

MSSV: B1910320

Khóa: K45

Cần Thơ, 05/2023

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CẦN THƠ
TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN**



**NIÊN LUẬN
NGÀNH CÔNG NGHỆ THÔNG TIN**

**Đề tài
XÂY DỰNG HỆ THỐNG
MẠNG XÃ HỘI**

**Giảng viên hướng dẫn:
Th.S. Lê Huỳnh Quốc Bảo**

**Sinh viên thực hiện:
Họ tên: Lê Dương Trí
MSSV: B1910320
Khóa: 45**

Cần Thơ, 05/2023

LỜI CẢM ƠN

Lời đầu tiên tôi xin chân thành cảm ơn Ban lãnh đạo trường Đại học Cần Thơ, quý thầy cô Trường Công nghệ thông tin và Truyền thông đã dẫn dắt, truyền đạt những kinh nghiệm, kiến thức quý báu trong suốt thời gian học tập, nghiên cứu và rèn luyện tại trường.

Tôi xin gửi lời cảm ơn chân thành đến Th.S. Lê Huỳnh Quốc Bảo, người đã tận tình hướng dẫn, giúp đỡ và tạo điều kiện tốt cho tôi có thể hoàn thành tốt niên luận này.

Do giới hạn kiến thức và khả năng lý luận của bản thân còn nhiều thiếu sót và hạn chế, kính mong sự chỉ dẫn và đóng góp của các Thầy, Cô để bài luận văn của tôi được hoàn thiện hơn.

Xin chân thành cảm ơn!

Cần Thơ, ngày ... tháng 05 năm 2023

Sinh viên thực hiện

Lê Dương Trí

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

Cần Thơ, ngày ... tháng ... năm 2023

Giảng viên hướng dẫn

Th.S. Lê Huỳnh Quốc Bảo

MỤC LỤC

MỤC LỤC.....	i
DANH MỤC HÌNH	v
DANH MỤC BẢNG.....	vi
DANH MỤC THUẬT NGỮ VÀ TỪ VIẾT TẮT	vii
TÓM TẮT	ix
ABSTRACT	x
PHẦN 1. GIỚI THIỆU	1
I. ĐẶT VẤN ĐỀ.....	1
II. LỊCH SỬ GIẢI QUYẾT VẤN ĐỀ	1
III. MỤC TIÊU ĐỀ TÀI	1
IV. ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU	1
1. Đối tượng nghiên cứu	1
2. Phạm vi nghiên cứu.....	2
V. PHƯƠNG PHÁP NGHIÊN CỨU	2
VI. NỘI DUNG NGHIÊN CỨU	2
VII. BỐ CỤC NIÊN LUẬN	3
1. Phần giới thiệu	3
2. Phần nội dung.....	3
3. Phần kết luận.....	3
PHẦN 2. NỘI DUNG	4
CHƯƠNG I. CƠ SỞ LÝ THUYẾT	4
I. VueJS.....	4
1. Template Syntax	4
1.1. Interpolation	4
1.2. Directives.....	6
1.3. Cú pháp rút gọn v-bind và v-on	7
2. Computed property và watcher	8
2.1. Computed property.....	8

2.2. Watcher	11
3. Binding, Render, Xử lý sự kiện	13
3.1. Binding	13
3.2. Render	16
3.3. Xử lý sự kiện	19
4. Components	20
4.1. Tổ chức component	20
4.2. Truyền dữ liệu xuống component con bằng prop	21
4.3. Tái sử dụng component	21
5. Lifecycle Hooks	22
5.1. Đăng ký Lifecycle Hooks	22
5.2. Sở đồ Lifecycle	23
II. NodeJS	26
1. Blocking I/O và Nonblocking I/O	26
2. Event Loop	27
3. Synchronous và Asynchronous	27
4. Callback	28
5. Module	28
6. NPM – Node Package Manager	28
III. Thư viện Axios	29
IV. Phương thức xác thực JWT	29
V. ExpressJS framework	30
1. Route parameters	31
2. Response methods	31
VI. Middleware trong ExpressJS	31
VII. API và RESTful API	32
VIII. Cơ sở dữ liệu MongoDB	34
IX. Socket.io	34
CHƯƠNG II. Nội dung	36

I. Mô tả đề tài.....	36
II. Các chức năng chính của hệ thống.....	37
III. use case diagram	38
IV. Thiết kế cơ sở dữ liệu Mongodb.....	38
1. Mô tả bảng dữ liệu	38
1.1. Bảng User	38
1.2. Bảng Post.....	39
1.3. Bảng Comment.....	40
1.4. Bảng News	40
1.5. Bảng Conversation	41
1.6. Bảng Message.....	41
2. Mô hình dữ liệu mức quan hệ	41
V. thiết kế các chức năng.....	42
1. Đăng ký và Đăng nhập.....	42
2. Cập nhật hồ sơ.....	43
3. Đăng bài viết	43
4. Tương tác bài đăng.....	44
4.1. Yêu thích bài đăng.....	44
4.2. Bình luận bài đăng.....	44
5. Kết bạn	45
6. Nhắn tin.....	46
CHƯƠNG III. kết quả nghiên cứu	47
I. giao diện Đăng ký và Đăng nhập	47
1. Đăng nhập	47
2. Đăng ký	47
II. giao diện Xem hồ sơ	48
III. giao diện Cập nhật hồ sơ.....	49
IV. giao diện Đăng bài viết.....	50
V. giao diện Tương tác bài đăng.....	51

VI. giao diện Kết bạn.....	52
VII. giao diện Nhắn tin	52
PHẦN 3. KẾT LUẬN và hướng phát triển.....	53
I. kết luận	53
1. Kết quả đạt được	53
2. Hạn chế.....	53
II. HƯỚNG PHÁT TRIỂN.....	53
TÀI LIỆU THAM KHẢO.....	54

DANH MỤC HÌNH

Hình 1. Tổ chức dưới dạng một cây component.....	20
Hình 2. Sơ đồ Lifecycle của một cá thể	23
Hình 3. Mô hình kiến trúc tổng quan về kiến trúc NodeJS và ExpressJS sử dụng JWT	30
Hình 4. Mô hình kiến trúc thành phần RESTful API.....	33
Hình 5. Socket.IO và các ứng dụng thời gian thực	35
Hình 6. Sơ đồ chức năng.....	37
Hình 7. Mô hình dữ liệu mức quan hệ MongoDB	41
Hình 8. Lưu đồ chức năng đăng nhập và đăng ký	42
Hình 9. Lưu đồ chức năng cập nhật hồ sơ	43
Hình 10. Lưu đồ chức năng đăng bài viết.....	43
Hình 11. Lưu đồ chức năng yêu thích bài đăng.....	44
Hình 12. Lưu đồ chức năng bình luận.....	44
Hình 13. Lưu đồ chức năng kết bạn.....	45
Hình 14. Lưu đồ chức năng nhắn tin.....	46
Hình 15. Trang đăng nhập.....	47
Hình 16. Trang đăng ký	47
Hình 17. Trang bài đăng của người dùng.....	48
Hình 18. Trang hồ sơ của người dùng.....	48
Hình 19. Chỉnh sửa thông tin hồ sơ	49
Hình 20. Chỉnh sửa thông tin liên hệ	49
Hình 21. Trang chủ với những bài đăng	50
Hình 22. Trang đăng bài viết.....	50
Hình 23. Trang chi tiết bài đăng – phần thông tin bài đăng.....	51
Hình 24. Trang chi tiết bài đăng – phần bình luận.....	51
Hình 25. Trang tìm kiếm và kết bạn	52
Hình 26. Trang hội thoại với bạn bè	52

DANH MỤC BẢNG

Bảng 1. Một số phương thức (response methods) hỗ trợ thường sử dụng nhất.....	31
Bảng 2. Các tác vụ cơ bản của REST dựa trên phương thức HTTP.....	33
Bảng 3. Bảng CSDL User	39
Bảng 4. Bảng CSDL Post.....	40
Bảng 5. Bảng CSDL Comment.....	40
Bảng 6. Bảng CSDL News.....	40
Bảng 7. Bảng CSDL Conversation	41
Bảng 8. Bảng CSDL Message.....	41

DANH MỤC THUẬT NGỮ VÀ TỪ VIẾT TẮT

STT	Từ viết tắt	Từ viết đầy đủ	Ý nghĩa
1	CRUD	Create – Read – Update - Delete	Thêm – Đọc – Sửa - Xóa
2	ES6	ECMAScript 6	Phiên bản mới nhất chuẩn ECMAScript
3	HTTP	Hyper Text Transfer Protocol	Giao thức Truyền tải Siêu Văn Bản
4	REST	Representational State Transfer	Một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng
5	API	Application Programming Interface	Giao diện lập trình ứng dụng
6	JWT	Json Web Token	Một phương tiện đại diện cho các yêu cầu chuyển giao giữa hai bên Client - Server
7	HTML	Hypertext Markup Language	Ngôn ngữ lập trình dùng để xây dựng và cấu trúc lại các thành phần có trong Website
8	NoSQL	None-Relational SQL	NoSQL được phát triển trên Javascript Framework với kiểu dữ liệu là JSON và dạng dữ liệu theo kiểu key và value
9	JSON	JavaScript Object Notation	Một kiểu dữ liệu mở trong Javascript
10	XML	Extensible Markup Language	XML có chức năng truyền dữ liệu và mô tả nhiều loại dữ liệu khác nhau. Tác dụng chính của XML là đơn giản hóa việc chia sẻ dữ liệu giữa các nền tảng và các hệ thống

			được kết nối thông qua mạng Internet.
11	CSS	Cascading Style Sheets	Được dùng để miêu tả cách trình bày các tài liệu viết bằng ngôn ngữ HTML và XHTML. Ngoài ra ngôn ngữ định kiểu theo tầng cũng có thể dùng cho XML, SVG, XUL.
12	DOM	Document Object Model	Mô hình Các Đối tượng Tài liệu
13	UI	User Interface	Giao diện người dùng
14	SFC	Single File Components	SFC là một định dạng tệp đặc biệt cho phép chúng tôi đóng gói mẫu, logic và kiểu dáng của thành phần Vue trong một tệp
15	SPA	Single-Page Applications	SPA là một ứng dụng web hoặc trang web tương tác với người dùng bằng cách tự động viết lại trang web hiện tại bằng dữ liệu mới từ máy chủ web, thay vì phương pháp mặc định của trình duyệt web tải toàn bộ trang mới. Mục tiêu là chuyển đổi nhanh hơn để làm cho trang web giống một ứng dụng gốc hơn.

TÓM TẮT

Mạng xã hội là một nền tảng trực tuyến với các mô hình, tính năng, cách sử dụng khác nhau, giúp mọi người dễ dàng truy cập và kết nối. Đây là nơi mọi người có thể giao lưu, gặp gỡ, chia sẻ thông tin, hình ảnh, video,...

Mạng xã hội cho phép người dùng chia sẻ câu chuyện, bài viết, ý tưởng cá nhân, đăng ảnh, video, đồng thời thông báo về hoạt động, sự kiện trên mạng hoặc trong thế giới thực. Mạng xã hội giúp người dùng kết nối với những người sống ở những vùng đất khác nhau hoặc trên toàn thế giới.

Đề tài “Xây dựng hệ thống mạng xã hội” sử dụng nền tảng NodeJS, framework VueJS 3 được xây dựng với những chức năng cần thiết của một mạng xã hội cần có. Đề tài được xây dựng trên nền tảng NodeJS và framework VueJS, được quản lý bởi cơ sở dữ liệu MongoDB và công cụ lập trình VSCode. Hệ thống giúp người dùng có thể tương tác với người dùng khác bằng cách đăng bài viết, thích hoặc bình luận bài viết và có thể nhắn tin,...

ABSTRACT

Social network is an online platform with different models, features, usage, making it easy for people to access and connect. This is a place where people can exchange, meet, share information, photos, videos, etc.

Social networks allow users to share stories, articles, personal ideas, post photos and videos, and announce activities and events online or in the real world. Social networks help users connect with people living in different lands or around the world.

The topic "Building a social network system" using the NodeJS platform, the VueJS 3 framework is built with the necessary functions of a required social network. The project is built on NodeJS platform and VueJS framework, managed by MongoDB database and VSCode programming tool. The system helps users to interact with other users by posting articles, liking or commenting on articles and can message, etc.

PHẦN 1. GIỚI THIỆU

I. ĐẶT VẤN ĐỀ

Trong cuộc sống bận rộn ngày nay, việc gặp mặt trực tiếp để liên lạc, trao đổi, chia sẻ giữa bạn bè, gia đình,... gặp phải trở ngại. Cũng như sau đại dịch Covid-19 việc gặp mặt trực tiếp nhau lại càng khó khăn.

Cùng với quá trình toàn cầu hóa và sự phát triển của công nghệ thông tin, mạng internet trên thế giới nói chung và Việt Nam nói riêng ngày càng phát triển mạnh mẽ. Sự tham gia của các cá nhân trên mạng ngày càng tích cực và nhu cầu chia sẻ thông tin, kết nối bạn bè là nhu cầu thiết yếu thúc đẩy sự ra đời và phát triển của các mạng xã hội.

Hiện nay, tình hình phát triển của các mạng xã hội cũng rất khả quan. Số lượng người truy cập và đăng ký thành viên ở mạng xã hội ngày càng tăng. Điển hình như một số website mạng xã hội: Facebook, Twitter, Reddit,... Tuy nhiên, dù đã có rất nhiều các trang mạng xã hội như thế nhưng nhu cầu của người dùng vẫn rất cao. Việc xây dựng website mạng xã hội giúp người dùng có nhiều sự lựa chọn phù hợp là việc làm cần thiết.

II. LỊCH SỬ GIẢI QUYẾT VẤN ĐỀ

Lĩnh vực mạng xã hội hiện nay đang rất phát triển, nhiều website lớn được xây dựng để phục vụ cho nhu cầu giao lưu, kết bạn. Đã và đang mang lại lượng người dùng cực lớn cho các nhà phát triển mạng xã hội web và ứng dụng.

Chính vì vậy, việc xây dựng website Life để chạy theo xu hướng của hiện đại, cũng là để tiếp cận tới nhiều người dùng hơn trên toàn thế giới nói chung và toàn Việt Nam nói riêng.

III. MỤC TIÊU ĐỀ TÀI

Với hệ thống mạng xã hội Life, mọi người có thể liên lạc, trao đổi, chia sẻ thông tin, trạng thái, hình ảnh,... cho bạn bè và gia đình một cách nhanh chóng và dễ dàng. Ngoài ra, còn giúp mở rộng các mối quan hệ như tìm kiếm, kết bạn với mọi người. Loại bỏ những trở ngại về địa lý và thời gian.

IV. ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU

1. Đối tượng nghiên cứu

Đối tượng nghiên cứu chính của đề tài là sử dụng VueJS và ExpressJS (NodeJS) để xây dựng website. Ngoài ra, cũng sẽ nghiên cứu thêm về việc xây dựng một kiến

trúc hệ thống ứng dụng sử dụng máy chủ dữ liệu (API server) và công cụ quản lý (Web client). Đối tượng nghiên cứu đầy đủ sẽ là một hệ thống gồm 2 thành phần:

- **REST API server:** xây dựng một server cung cấp các thao tác tạo, thêm, sửa, xóa (CRUD) dữ liệu của hệ thống thông qua HTTP request.
- **Web client:** xây dựng một ứng dụng web giao tiếp với server thông qua REST API, nhằm mục đích quản lý dữ liệu cốt lõi của hệ thống.

2. Phạm vi nghiên cứu

Phạm vi nghiên cứu của đề tài chủ yếu là làm việc với VueJS để tạo ra website mạng xã hội, sử dụng Axios để giao tiếp với API server. Phía Backend sẽ sử dụng NodeJS với framework ExpressJS để xây dựng nhanh hệ thống RESTful API.

V. PHƯƠNG PHÁP NGHIÊN CỨU

Nội dung nghiên cứu khá phổ biến, do đó phương pháp nghiên cứu chủ yếu là tìm hiểu thông qua tài liệu trang chủ của các thư viện, thông qua các video trên Youtube và làm các ví dụ trực tiếp. Từ đó, sẽ vận dụng các kiến thức đã tiếp thu được vào thực hiện đề tài. Tương ứng với mỗi thành phần của hệ thống sẽ có những phương pháp nghiên cứu khác nhau: API server: nghiên cứu cơ bản về NodeJS với framework Express để xây dựng một hệ thống RESTful và hệ quản trị cơ sở dữ liệu MongoDB. Web client: nghiên cứu cơ bản về ứng dụng framework VueJS và thư viện axios.

VI. NỘI DUNG NGHIÊN CỨU

STT	Đối tượng	Nội dung
1	API và RESTful API	Tìm hiểu về API và RESTful API
2	NodeJS – ExpressJS	Tìm hiểu về NodeJS với ExpressJS
3	MongoDB	Tìm hiểu hệ quản trị cơ sở dữ liệu MongoDB, cấu hình MongoDB trong NodeJS
4	VueJS	Tìm hiểu về framework Vue 3
5	Axios	Tìm hiểu về Axios Trao đổi với API server thông qua axios
6	JWT	Tìm hiểu xác thực và phân quyền người dùng
7	Socket.io	Tìm hiểu về Socket.io

8	Middleware	Tìm hiểu về quản lý các request và response của người dùng hệ thống
---	------------	---

VII. BỐ CỤC NIÊN LUẬN

Bố cục niên luận gồm có 3 phần chính như sau:

1. Phần giới thiệu

Bao gồm các nội dung:

- Đặt vấn đề
- Lịch sử giải quyết vấn đề
- Mục tiêu đề tài
- Đối tượng và phạm vi nghiên cứu
- Phương pháp nghiên cứu
- Nội dung nghiên cứu
- Bố cục của niên luận

2. Phần nội dung

Bao gồm các chương sau:

- Chương 1. Cơ sở lý thuyết
- Chương 2. Nội dung
- Chương 3. Kết quả nghiên cứu

3. Phần kết luận

Bao gồm các nội dung:

- Kết luận
- Hướng phát triển

PHẦN 2. NỘI DUNG

CHƯƠNG I. CƠ SỞ LÝ THUYẾT

I. VUEJS

VueJS gọi tắt là Vue, là một framework linh động dùng để xây dựng UI. Khác với các framework nguyên khối (monolithic), Vue được thiết kế từ đầu theo hướng cho phép và khuyến khích việc phát triển ứng dụng theo từng bước. Khi phát triển lớp giao diện (view layer), người dùng chỉ cần dùng thư viện lõi (core library) của Vue, vốn rất dễ học và tích hợp với các thư viện hoặc dự án có sẵn. Cùng lúc đó, nếu kết hợp với những kỹ thuật hiện đại như SFC và các thư viện hỗ trợ, Vue cũng đáp ứng được dễ dàng nhu cầu xây dựng những ứng dụng đơn trang (SPA) với độ phức tạp cao hơn nhiều.

1. Template Syntax

Vue sử dụng cú pháp mẫu dựa trên HTML cho phép bạn liên kết khai báo DOM được hiển thị với dữ liệu của phiên bản thành phần cơ bản. Tất cả các mẫu Vue đều là HTML hợp lệ về mặt cú pháp có thể được phân tích cú pháp bằng các trình duyệt và trình phân tích cú pháp HTML tuân thủ thông số kỹ thuật.

Về cơ bản, Vue biên dịch các mẫu thành mã JavaScript được tối ưu hóa cao. Kết hợp với hệ thống phản ứng, Vue có thể tìm ra số lượng thành phần tối thiểu để kết xuất lại một cách thông minh và áp dụng số lượng thao tác DOM tối thiểu khi trạng thái ứng dụng thay đổi.

Nếu bạn đã quen thuộc với các khái niệm Virtual DOM và thích sức mạnh thô của JavaScript, thì bạn cũng có thể viết trực tiếp các hàm kết xuất thay vì các mẫu, với sự hỗ trợ JSX tùy chọn. Tuy nhiên, xin lưu ý rằng chúng không được hưởng mức độ tối ưu hóa thời gian biên dịch giống như các mẫu.

1.1. Interpolation

- **Interpolation - Text**

Hình thức ràng buộc dữ liệu cơ bản nhất là nội suy văn bản (text interpolation) sử dụng cú pháp “mustache” (“ria mép” – hai dấu ngoặc nhọn):

```
<span>Thông điệp: {{ msg }}</span>
```

HTML

Thẻ mustache sẽ được thay thế bằng giá trị của thuộc tính **msg** trên object data tương ứng, và cũng sẽ được cập nhật bất cứ khi nào thuộc tính này thay đổi.

- **HTML thuần túy**

Cú pháp mustache sẽ thông dịch dữ liệu ra thành văn bản thuần túy (plain text), nghĩa là các kí tự HTML đặc biệt như `<&”` sẽ được mã hóa. Để xuất ra HTML thuần túy, bạn sẽ cần đến directive **v-html**.

```
<p>Sử dụng cú pháp mustache: {{ rawHtml }}</p>
<p>Sử dụng directive v-html: <span v-html="rawHtml"></span></p>
```

HTML

Nội dung của thẻ **span** sẽ được thay thế bằng giá trị của thuộc tính **rawHtml** dưới dạng HTML thuần túy - tất cả các ràng buộc dữ liệu sẽ không được xử lí. Lưu ý rằng bạn không thể dùng **v-html** để viết template partial, vì Vue không phải là một template engine dựa trên chuỗi. Thay vào đó, hãy dùng component cho mục đích biên soạn và tái sử dụng UI.

- **Các thuộc tính HTML**

Cú pháp mustache không dùng được bên trong các thuộc tính HTML. Thay vào đó, bạn hãy dùng **directive v-bind**:

```
<div v-bind:id="dynamicId"></div>
```

HTML

Directive này cũng hoạt động với các thuộc tính boolean như **disabled** và **selected** - các thuộc tính này sẽ được bỏ đi khi biểu thức được tính toán trả về kết quả sai (false):

```
<button v-bind:disabled="isButtonDisabled">Hòn Vọng Phu</button>
```

HTML

- **Sử dụng các biểu thức JavaScript**

Cho đến nay chúng ta chỉ mới bind vào các khóa thuộc tính đơn giản trong template. Tuy nhiên, thật ra Vue hỗ trợ sức mạnh toàn diện của các biểu thức JavaScript bên trong toàn bộ các ràng buộc dữ liệu (data binding):

```
{{ number + 1 }}
{{ ok ? 'YES' : 'NO' }}
{{ message.split('').reverse().join('') }}
<div v-bind:id="'list-' + id"></div>
```

HTML

Các biểu thức này sẽ được tính toán dưới dạng JavaScript trong scope của đối tượng Vue hiện hành. Một hạn chế ở đây là mỗi ràng buộc chỉ có thể chứa một biểu thức đơn lẻ, vì thế các trường hợp sau sẽ không hoạt động:

```
<!-- đây là một khai báo, không phải biểu thức: -->
{{ var a = 1 }}

<!--
  các lệnh quản lí luồng (flow control) cũng sẽ không hoạt động,
  thay vào đó bạn hãy dùng toán tử ba ngôi (ternary operator):
-->
{{ if (ok) { return message } }}
```

1.2. Directives

Directive là các thuộc tính đặc biệt với prefix (tiếp đầu ngữ) **v-**. Giá trị của thuộc tính directive phải là một biểu thức JavaScript đơn lẻ (ngoại trừ **v-for** mà chúng ta sẽ đề cập sau). Nhiệm vụ của một directive là áp dụng các hiệu ứng phụ vào DOM khi giá trị của biểu thức thay đổi. Hãy xem lại ví dụ chúng ta đã thấy trong phần giới thiệu:

```
<p v-if="seen">Thoát ẩn thoát hiện</p>
```

Ở đây, directive **v-if** sẽ thêm vào hoặc bỏ đi phần tử **<p>** dựa trên tính đúng sai của giá trị của biểu thức **seen**.

- **Tham số**

Một số directive có thể nhận vào một tham số, đánh dấu bằng một dấu hai chấm theo sau tên của directive. Ví dụ, directive **v-bind** được dùng để cập nhật động một thuộc tính HTML:

```
<a v-bind:href="url"> ... </a>
```

Ở đây **href** là tham số hướng dẫn directive **v-bind** ràng buộc thuộc tính **href** vào giá trị của biểu thức **url**.

Một ví dụ khác là directive **v-on**. Directive này lắng nghe các sự kiện DOM:

```
<a v-on:click="doSomething"> ... </a>
```

Tham số ở đây là tên của sự kiện để lắng nghe (click). Chúng ta cũng sẽ bàn sâu về quản lý sự kiện sau.

- **Modifier**

Modifier là các hậu tố (postfix) đặc biệt được đánh dấu bằng một dấu chấm, chỉ rõ rằng một directive phải được ràng buộc theo một cách đặc biệt nào đó. Ví dụ, modifier **.prevent** hướng dẫn directive **v-on** gọi **event.preventDefault()** khi sự kiện được kích hoạt:

```
<form v-on:submit.prevent="onSubmit"> ... </form>
```

HTML

Sau này bạn sẽ gặp thêm các ví dụ khác về modifier cho **v-on** và **v-model**, khi chúng ta bàn đến các tính năng này.

1.3. Cú pháp rút gọn v-bind và v-on

Prefix **v-** đóng vai trò gợi ý trực quan để nhận ra các thuộc tính riêng của Vue trong template. Điều này có ích khi bạn sử dụng Vue vào các dự án có sẵn, tuy nhiên đối với các directive được dùng thường xuyên thì **v-** có thể trông hơi rườm rà. Thêm vào đó **v-** trở nên kém quan trọng hơn khi bạn xây dựng các ứng dụng một trang, trong đó Vue quản lý toàn bộ các template. Vì thế Vue cung cấp dạng rút gọn (shorthand) đặc biệt cho hai trong số các directive được dùng nhiều nhất, **v-bind** và **v-on**:

- **v-bind**

```
<!-- cú pháp đầy đủ -->
<a v-bind:href="url"> ... </a>

<!-- cú pháp rút gọn: dùng dấu hai chấm -->
<a :href="url"> ... </a>
```

HTML

- **v-on**

```
<!-- cú pháp đầy đủ -->
<a v-on:click="doSomething"> ... </a>

<!-- cú pháp rút gọn: dùng kí tự @ -->
<a @click="doSomething"> ... </a>
```

HTML

Tuy nhìn có vẻ khác với HTML thông thường, **:** và **@** là các kí tự hợp lệ cho các thuộc tính HTML, và các trình duyệt hỗ trợ Vue đều có thể parse được hai kí tự

này. Thêm vào đó, các directive không xuất hiện trong code HTML được render ra. Cú pháp rút gọn là hoàn toàn không bắt buộc, nhưng có lẽ là bạn sẽ thích dùng sau khi biết thêm về cách dùng của chúng.

2. Computed property và watcher

2.1. Computed property

Viết biểu thức trực tiếp trong template rất tiện, nhưng chỉ dành cho những biểu thức có tính toán đơn giản. Những biểu thức phức tạp được viết theo cách đó sẽ khiến template cồng kềnh và khó bảo trì. Ví dụ:

```
<div id="example">
  {{ message.split('').reverse().join('') }}
</div>
```

Đến đây, template không còn đơn giản và mang tính khai báo (declarative) nữa. Bạn sẽ phải mất chút thời gian thì mới nhận ra được **message** đã bị đảo ngược. Càng tệ hơn khi bạn sử dụng biến **message** đảo ngược này nhiều lần trong code.

Đó là lí do tại sao đối với bất kì logic nào phức tạp, bạn nên sử dụng **computed property**.

- Ví dụ cơ bản

```
<div id="example">
  <p>Thông điệp ban đầu: "{{ message }}"</p>
  <p>Thông điệp bị đảo ngược bằng tính toán (computed): "{{ reversedMessage }}"</p>
</div>
```

```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'người đông bến đợi thuyền xuôi ngược'
  },
  computed: {
    // một computed getter
    reversedMessage: function () {
      // `this` trỏ tới đối tượng vm
      return this.message.split(' ').reverse().join(' ')
    }
  }
})
```

Kết quả là:

Thông điệp ban đầu: "người đông bến đợi thuyền xuôi ngược"
Thông điệp bị đảo ngược (computed): "ngược xuôi thuyền đợi bến
đông người"

- **Computed caching và method**

Bạn có lẽ đã nhận ra chúng ta cũng có thể đạt được cùng một kết quả bằng cách sử dụng một phương thức:

```
<p>Thông điệp bị đảo ngược: "{{ reverseMessage() }}"</p>
```

HTML

```
// trong component
methods: {
  reverseMessage: function () {
    return this.message.split(' ').reverse().join(' ')
  }
}
```

JS

Thay vì sử dụng computed property, chúng ta cũng có thể dùng một phương thức thay thế. Nếu xét về kết quả cuối cùng thì hai cách tiếp cận này thật ra chỉ là một. Tuy nhiên, sự khác biệt ở đây là *computed property được cache lại dựa vào những những thành phần phụ thuộc (dependency)*. Một computed property chỉ được tính toán lại khi những thành phần phụ thuộc của chúng thay đổi. Điều này có nghĩa: miễn là giá trị của **message** không thay đổi, thì những truy cập tới computed **reversedMessage** sẽ ngay lập tức trả về kết quả được tính toán trước đó mà không phải chạy lại hàm một lần nữa.

Điều này cũng có nghĩa là computed property dưới đây sẽ không bao giờ cập nhật, bởi vì **Data.now()** không phải là một thành phần phụ thuộc phản ứng (reactive dependency):

```
computed: {  
  now: function () {  
    return Date.now()  
  }  
}
```

Để so sánh, một phương thức luôn được gọi khi có một sự kiện render lại (re-render) xảy ra.

- **Computed và watched**

Vue cung cấp một cách khái quát hơn để quan sát và phản ứng (react) lại những thay đổi trên dữ liệu: **watch property**. Khi bạn có một số dữ liệu cần được thay đổi dựa trên những dữ liệu khác, bạn rất dễ lạm dụng **watch** - nhất là nếu bạn có nền tảng về AngularJS. Tuy nhiên, thường thì bạn nên dùng **computed** thay vì **watch**. Hãy xem ví dụ sau:

```
<div id="demo">{{ fullName }}</div>
```

```
var vm = new Vue({  
  el: '#demo',  
  data: {  
    firstName: 'Trần',  
    lastName: 'Lập',  
    fullName: 'Trần Lập'  
  },  
  watch: {  
    firstName: function (val) {  
      this.fullName = val + ' ' + this.lastName  
    },  
    lastName: function (val) {  
      this.fullName = this.firstName + ' ' + val  
    }  
  }  
})
```

Đoạn code phía trên theo hướng mệnh lệnh và lặp lại. Hãy so sánh với phiên bản dùng computed property:


```
JS
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Evan',
    lastName: 'You'
  },
  computed: {
    fullName: function () {
      return this.firstName + ' ' + this.lastName
    }
  }
})
```

- **Computed Setter**

Những computed property mặc định chỉ có getter, nhưng bạn cũng có thể cung cấp setter nếu cần thiết:

```
JS
// ...
computed: {
  fullName: {
    // getter
    get: function () {
      return this.firstName + ' ' + this.lastName
    },
    // setter
    set: function (newValue) {
      var names = newValue.split(' ')
      this.firstName = names[0]
      this.lastName = names[names.length - 1]
    }
  }
}
// ...
```

Bây giờ, khi bạn gán **vm.fullName = 'John Doe'**, thì setter sẽ được gọi, **vm.firstName** và **vm.lastName** sẽ được cập nhật tương ứng.

2.2. Watcher

Computed property thích hợp cho hầu hết các trường hợp, nhưng cũng có lúc cần tới những watcher tùy biến. Đó là lí do tại sao Vue cung cấp một cách khái quát hơn để phản ứng lại với việc thay đổi dữ liệu trong **watch**. Cách sử dụng này rất hữu ích khi bạn muốn thực hiện những tính toán không đồng bộ và tốn kém liên quan đến việc thay đổi dữ liệu.

Ví dụ:

```
<div id="watch-example">
  <p>
    Hãy hỏi một câu hỏi yes/no:
    <input v-model="question">
  </p>
  <p>{{ answer }}</p>
</div>
```

```
<script src="https://cdn.jsdelivr.net/npm/axios@0.12.0/dist/axios.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/lodash@4.13.1/lodash.min.js"></script>
<script>
var watchExampleVM = new Vue({
  el: '#watch-example',
  data: {
    question: '',
    answer: 'Không thể trả lời nếu bạn chưa đặt câu hỏi!'
  },
  watch: {
    // bất cứ lúc nào câu hỏi thay đổi, hàm bên dưới sẽ chạy
    question: function (newQuestion, oldQuestion) {
      this.answer = 'Đang chờ bạn đặt xong câu hỏi...'
      this.getAnswer()
    }
  },
  methods: {
    // _.debounce là một hàm do Lodash cung cấp
    // Để tìm hiểu rõ hơn cách hoạt động của hàm này,
    // bạn có thể truy cập: https://lodash.com/docs#debounce
    getAnswer: _.debounce(
      function () {
        if (this.question.indexOf('?') === -1) {
          this.answer = 'Câu hỏi thì thường chứa một dấu "?" ;-)'
          return
        }
        this.answer = 'Đang suy nghĩ...'
        var vm = this
        axios.get('https://yesno.wtf/api')
          .then(function (response) {
            vm.answer = _.capitalize(response.data.answer)
          })
      }
    )
  }
})
```

```
    })  
  },  
  // Đây là thời gian (đơn vị mili giây) chúng ta đợi người dùng dừng gõ.  
  500  
)  
}  
})  
</script>
```

Trong trường hợp này, sử dụng **watch** cho phép chúng ta thực hiện những tính toán không đồng bộ (ví dụ: truy cập tới một API), giới hạn việc chúng ta thường xuyên thực hiện tính toán đó và gán trạng thái trung gian cho tới khi chúng ta có được kết quả cuối cùng. Nếu dùng computed property bạn sẽ không làm được những chuyện này.

3. Binding, Render, Xử lý sự kiện

3.1. Binding

3.1.1. Binding class trong HTML

- Sử dụng cú pháp Object

Chúng ta có thể truyền một đối tượng vào **:class** (viết tắt của **v-bind:class**) để tự động chuyển đổi giữa các lớp:

```
<div :class="{ active: isActive }"></div>
```

template

Cú pháp như trên nghĩa là class **active** sẽ được áp dụng tùy theo tính đúng sai (truthiness) của thuộc tính dữ liệu **isActive**.

Bạn có thể bật tắt nhiều class bằng cách dùng nhiều field (trường) trong object. Thêm vào đó, directive **:class** và thuộc tính **class** thông thường có thể được dùng cùng lúc với nhau. Nếu chúng ta có template sau:

```
<div  
  class="static"  
  :class="{ active: isActive, 'text-danger': hasError }"  
></div>
```

template

và dữ liệu truyền vào như thế này:

```
data() {  
  return {  
    isActive: true,  
    hasError: false  
  }  
}
```

thì kết quả render sẽ là:

```
<div class="static active"></div>
```

Khi giá trị **isActive** hoặc **hasError** thay đổi, danh sách class sẽ được cập nhật tương ứng. Ví dụ, nếu **hasError** trở thành **true**, danh sách class sẽ trở thành **"static active text-danger"**.

- **Sử dụng cú pháp mảng**

Chúng ta có thể truyền một mảng vào **:class** để áp dụng một danh sách class:

```
data() {  
  return {  
    activeClass: 'active',  
    errorClass: 'text-danger'  
  }  
}
```

```
<div :class="[activeClass, errorClass]"></div>
```

sẽ render ra kết quả sau:

```
<div class="active text-danger"></div>
```

Nếu muốn bật tắt theo điều kiện một class trong danh sách, bạn có thể dùng một toán tử ba ngôi (ternary expression):

```
<div :class="[isActive ? activeClass : '', errorClass]"></div>
```

Đoạn code trên sẽ luôn luôn áp dụng class **errorClass**, nhưng chỉ áp dụng **activeClass** khi **isActive** mang giá trị đúng.

Cách làm này có thể hơi dài dòng nếu bạn có nhiều class theo điều kiện. Do đó, bạn có thể dùng cú pháp object bên trong cú pháp mảng, như sau:

```
<div :class="{ active: isActive }, errorClass}"></div>
```

template

- **Sử dụng với các component**

Khi sử dụng thuộc tính **class** trên một component tùy biến, những class được liệt kê trong thuộc tính này sẽ được thêm vào phần tử root của component đó. Những class có sẵn trên phần tử này sẽ được giữ nguyên.

Ví dụ, nếu bạn khai báo một component với một số class có sẵn như sau:

```
Vue.component('my-component', {  
  template: '<p class="foo bar"></p>  
})
```

JS

sau đó khi dùng component này bạn lại khai báo một số class khác:

```
<my-component class="baz qux"></my-component>
```

HTML

thì kết quả HTML tạo thành sẽ là:

```
<p class="foo bar baz qux"></p>
```

HTML

Binding cho class cũng vậy:

```
<my-component v-bind:class="{ active: isActive }"></my-component>
```

HTML

Khi **isActive** mang giá trị đúng, kết quả HTML sẽ là:

```
<p class="foo bar active"></p>
```

HTML

3.1.2. Binding cho inline style

- **Sử dụng cú pháp Object**

Cú pháp object của **v-bind:style** rất đơn giản - trông giống như CSS thông thường, chỉ khác ở chỗ nó là một object JavaScript. Bạn có thể dùng camelCase hoặc kebab-case (đặt trong dấu nháy nếu là kebab-case) đối với tên thuộc tính CSS:

```
<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

HTML

```
data: {
  activeColor: 'red',
  fontSize: 30
}
```

JS

Thông thường thì ta nên bind vào một object dành riêng cho style để template được gọn gàng hơn:

```
<div v-bind:style="styleObject"></div>
```

HTML

```
data: {
  styleObject: {
    color: 'red',
    fontSize: '13px'
  }
}
```

JS

Một lần nữa, cú pháp object thường được dùng kết hợp với các computed property trả về object.

- **Sử dụng cú pháp mảng**

Cú pháp mảng của **v-bind:style** giúp bạn áp dụng nhiều object style cho cùng một phần tử web:

```
<div v-bind:style="[baseStyles, overridingStyles]"></div>
```

HTML

3.2. Render

3.2.1. Render theo điều kiện

- **v-if**

Lệnh v-if được sử dụng để hiển thị một khối theo điều kiện. Khối sẽ chỉ được hiển thị nếu biểu thức của lệnh trả về giá trị trung thực.

```
<h1 v-if="ok">Mọi việc ổn cả</h1>
```

HTML

- **Nhóm điều kiện với v-if trên thẻ <template>**

Vì là một directive, **v-if** phải được dùng trên một phần tử đơn lẻ (single element) như **<p>** hoặc **<div>**. Nếu chúng ta muốn bật tắt một nhóm các phần tử thì sao? Chỉ cần dùng **v-if** trên một phần tử **<template>** với vai trò wrap (bọc) các phần tử lại thành một nhóm. Kết quả render cuối cùng sẽ không có phần tử **<template>** này.

```
<template v-if="ok">
  <h1>Anh chàng chần lộn</h1>
  <p>Ơ này, Augustin thân mến ơi</p>
  <p>Mọi việc đều như ý, như ý, như ý</p>
</template>
```

HTML

- **v-else**

Ta có thể dùng directive **v-else** để chỉ định một khối “else” cho **v-if**:

```
<div v-if="Math.random() > 0.5">
  Tài
</div>
<div v-else>
  Xiu
</div>
```

HTML

Để được coi là hợp lệ, phần tử có **v-else** phải theo ngay sau một phần tử **v-if** hoặc **v-else-if**.

3.2.2. Render danh sách

- **v-for**

Chúng ta có thể dùng directive **v-for** để render một danh sách các item dựa trên một mảng. Directive **v-for** đòi hỏi một cú pháp đặc biệt dưới dạng **item in items**, trong đó **items** là mảng dữ liệu nguồn và **item** trỏ đến phần tử mảng đang được duyệt đến:

```
<ul id="example-1">
  <li v-for="item in items">
    {{ item.name }}
  </li>
</ul>
```

HTML

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { name: 'Cà phê' },
      { name: 'Trà đặc' },
      { name: 'Bò húc' }
    ]
  }
})
```

Kết quả:

- Cà phê
- Trà đặc
- Bò húc

- **v-for dùng trong thẻ <template>**

Tương tự với **v-if**, bạn có thể dùng **v-for** trên một thẻ **<template>** để render một lúc nhiều phần tử. Ví dụ:

```
<ul>
  <template v-for="item in items">
    <li>{{ item.msg }}</li>
    <li class="divider"></li>
  </template>
</ul>
```

- **v-for dùng với v-if**

Khi được dùng trên cùng một node, **v-for** có độ ưu tiên cao hơn **v-if**, có nghĩa là **v-if** sẽ được thực thi một cách riêng biệt trên mỗi vòng lặp của **v-for**. Điều này có thể có ích khi bạn muốn render cho chỉ một số **item**, như trong ví dụ sau:

```
<li v-for="todo in todos" v-if="!todo.isComplete">
  {{ todo }}
</li>
```


3.3. Xử lý sự kiện

Chúng ta có thể sử dụng chỉ thị **v-on** mà chúng ta thường rút ngắn thành ký hiệu **@** để lắng nghe các sự kiện DOM và chạy một số JavaScript khi chúng được kích hoạt. Cách sử dụng sẽ là **v-on:click="handler"** hoặc với phím tắt, **@click="handler"**.

Giá trị xử lý có thể là một trong những điều sau đây:

- **Inline handlers:** JavaScript nội tuyến sẽ được thực thi khi sự kiện được kích hoạt (tương tự như thuộc tính onclick gốc).
- **Method handlers:** Tên thuộc tính hoặc đường dẫn dẫn đến một phương thức được xác định trên thành phần.

3.3.1. Inline handlers

Inline handlers thường được sử dụng trong các trường hợp đơn giản, ví dụ:

```
data() {  
  return {  
    count: 0  
  }  
}
```

js

```
<button @click="count++">Add 1</button>  
<p>Count is: {{ count }}</p>
```

template

3.3.2. Method handlers

Tuy nhiên, logic cho nhiều trình xử lý sự kiện sẽ phức tạp hơn và có thể không khả thi với trình xử lý nội tuyến. Đó là lý do tại sao **v-on** cũng có thể chấp nhận tên hoặc đường dẫn của phương thức thành phần mà bạn muốn gọi.

Ví dụ:

```
data() {  
  return {  
    name: 'Vue.js'  
  }  
},  
methods: {  
  greet(event) {  
    // `this` inside methods points to the current active instance  
    alert(`Hello ${this.name}!`)  
    // `event` is the native DOM event  
    if (event) {  
      alert(event.target.tagName)  
    }  
  }  
}
```



```
<!-- `greet` is the name of the method defined above -->  
<button @click="greet">Greet</button>
```

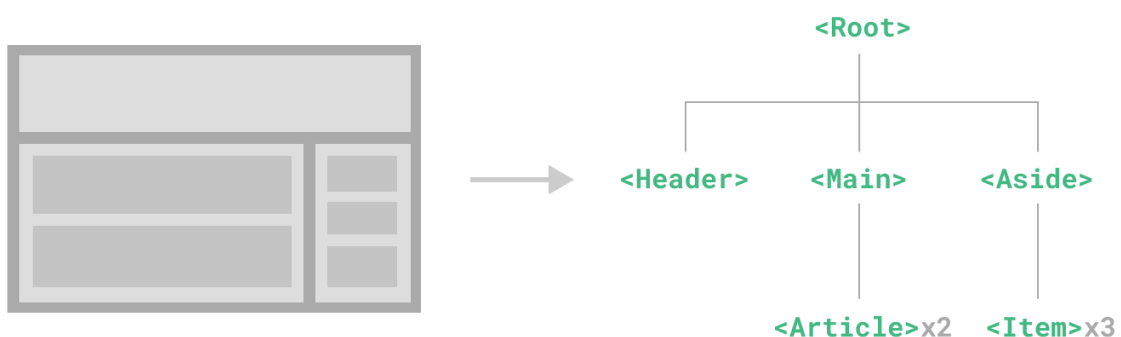
template

Trình xử lý phương thức tự động nhận đối tượng Sự kiện DOM gốc kích hoạt nó - trong ví dụ trên, chúng ta có thể truy cập phần tử gửi sự kiện qua `event.target.tagName`.

4. Components

4.1. Tổ chức component

Một ứng dụng thường được tổ chức dưới dạng một cây component lồng nhau:



Hình 1. Tổ chức dưới dạng một cây component

Ví dụ, bạn có thể có các component cho header, sidebar, khu vực nội dung, mỗi component này lại chứa các component dành cho trình đơn, blog post, vân vân.

Để có thể được sử dụng trong các template, component phải được đăng kí. Có hai cách đăng kí component: toàn cục và cục bộ. Trên đây chúng ta chỉ mới đăng kí component ở cấp toàn cục với **Vue.component**:

```
Vue.component('my-component-name', {  
  // ... tùy chọn ...  
})
```

JS

Component đăng kí ở cấp toàn cục có thể được dùng trong template của bất kì đối tượng Vue gốc (**new Vue**) nào được tạo ra sau đó – và trong tất cả các component con trên cây component của đối tượng đó.

4.2. Truyền dữ liệu xuống component con bằng prop

Prop là các thuộc tính tùy chỉnh mà bạn có thể đăng kí trên một component. Khi một giá trị được truyền vào một prop, nó trở thành một “_prop_erty” của đối tượng component đó. Để truyền tựa đề (**title**) vào component bài viết (**blog-post**), chúng ta sử dụng tùy chọn **props**:

```
Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>'
})
```

JS

Một component có thể có bao nhiêu prop tùy ý, và prop có thể nhận bất kì giá trị gì. Trong template trên đây, bạn có thể thấy là chúng ta có thể truy xuất giá trị này trên đối tượng component, giống như với **data**.

4.3. Tái sử dụng component

Bạn có thể tái sử dụng component bao nhiêu lần tùy ý:

```
<div id="components-demo">
  <button-counter></button-counter>
  <button-counter></button-counter>
  <button-counter></button-counter>
</div>
```

HTML

Bạn đã bấm 0 lần. Bạn đã bấm 0 lần. Bạn đã bấm 0 lần.

Bạn đã bấm 2 lần. Bạn đã bấm 0 lần. Bạn đã bấm 1 lần.

Đề ý là khi bấm các nút trên đây, mỗi nút giữ một giá trị **count** riêng hoàn toàn tách biệt. Điều này là vì mỗi khi bạn dùng một component, một **đối tượng** của component đó được tạo mới.

- **Data phải là một hàm**

Bạn có thể cũng đã để ý thấy rằng khi định nghĩa component **<button-counter>**, chúng ta không truyền thẳng một object vào **data** như thế này:

```
data: {  
  count: 0  
}
```

Thay vào đó, tùy chọn **data** của component phải là một hàm. Bằng cách này, mỗi đối tượng của component có thể duy trì một bản sao riêng biệt của đối tượng data được trả về:

```
data: function () {  
  return {  
    count: 0  
  }  
}
```

Nếu Vue không có quy tắc này, bấm một nút sẽ ảnh hưởng đến dữ liệu của **toàn bộ các đối tượng khác**.

5. Lifecycle Hooks

5.1. Đăng ký Lifecycle Hooks

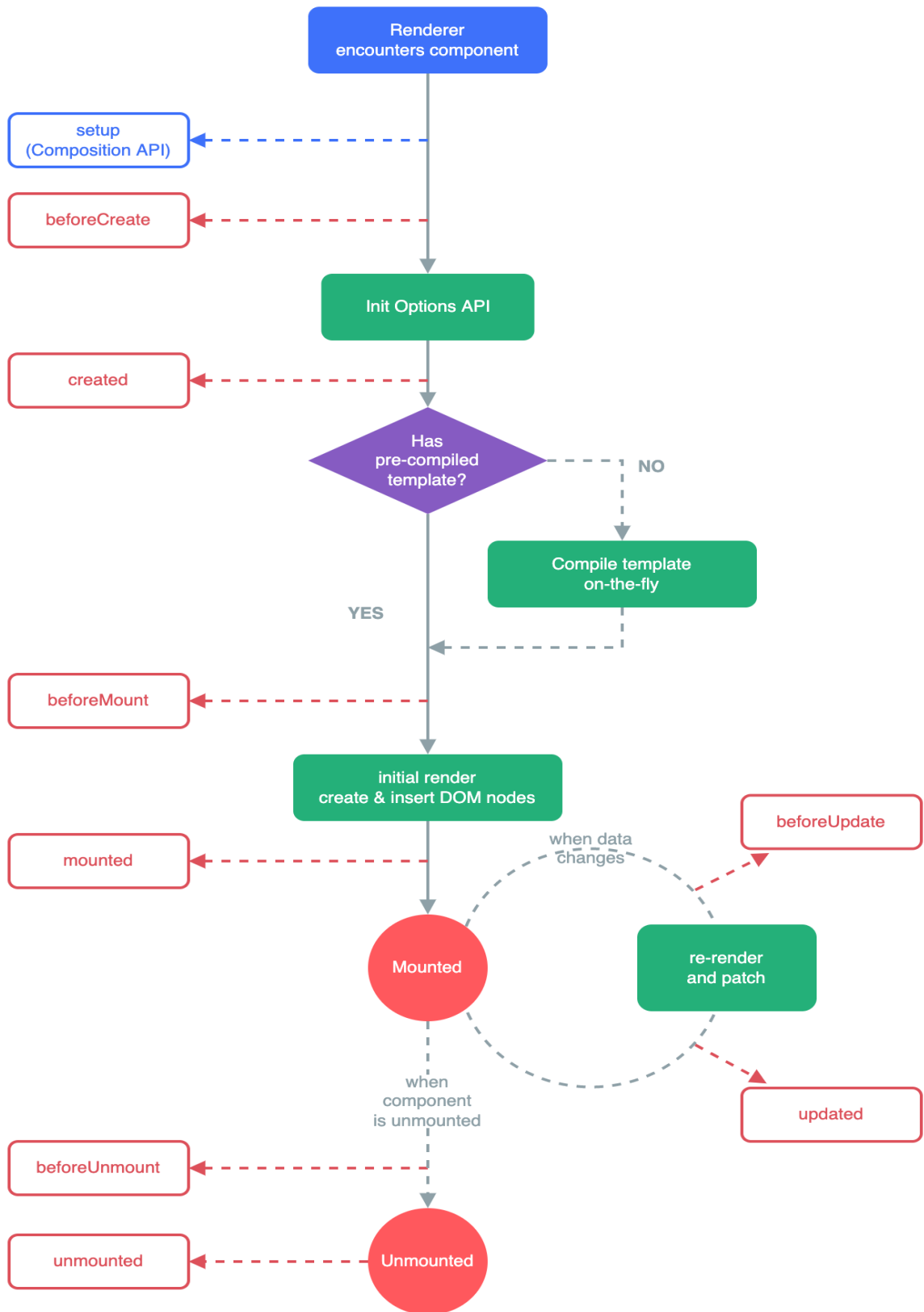
Ví dụ: mounted hook có thể được sử dụng để chạy mã sau khi thành phần kết thúc quá trình hiển thị ban đầu và tạo các nút DOM:

```
export default {  
  mounted() {  
    console.log(`the component is now mounted.`)  
  }  
}
```

Ngoài ra còn có các hook khác sẽ được gọi ở các giai đoạn khác nhau trong vòng đời của phiên bản, trong đó thường được sử dụng nhất là mount, update và unmount.

Tất cả các móc vòng đời được gọi với bối cảnh **this** của chúng trở đến phiên bản hoạt động hiện tại đang gọi nó. Lưu ý rằng điều này có nghĩa là bạn nên tránh sử dụng các hàm mũi tên khi khai báo móc vòng đời, vì bạn sẽ không thể truy cập phiên bản thành phần thông qua **this** nếu bạn làm như vậy.

5.2. Sơ đồ Lifecycle



Hình 2. Sơ đồ Lifecycle của một cá thể

5.2.1. beforeCreate và created

- **beforeCreate()**

Vì hook này là thứ khởi tạo tất cả dữ liệu và sự kiện phản ứng của dữ liệu. nên beforeCreate() không có quyền truy cập vào bất kì dữ liệu và sự kiện phản ứng nào của một thành phần.

```
<script>
export default {
  name: "Lifecycle",
  data () {
    return {
      message: "hello"
    }
  },
  beforeCreate() {
    console.log(this.message);
  }
}
</script>
```

Ở ví dụ này output trả về là `undefined` vì dữ liệu chưa được khai báo và chưa được gán vào trong component chính vì thế bạn cũng không thể gọi được các methods trong phương thức này. Sử dụng hook này rất hữu ích khi cần một số loại lệnh xử lý logic/API không cần gán cho dữ liệu bởi vì khi gán dữ liệu sau hook này nó sẽ tự mất.

- **created()**

Ở trong hook này thì có thể truy cập được dữ liệu bình thường vì dữ liệu đã được khai báo và phản ứng nếu có sự thay đổi chúng ta cùng xem sự thay đổi nhé :

```
export default {
  name: "Lifecycle",
  data () {
    return {
      message: "hello",
      create: '',
    }
  },
  beforeCreate() {
    this.message = "xin chao"
    this.message2 = "xin chao";
  },
  created() {
    console.log(this.message);
    console.log(this.message2);
    this.create = "xin chao"
  },
}
</script>
```

Để chứng minh cho ví dụ mà tớ đã nêu ở **beforeCreate()** thì tớ đã khai báo để cho dễ hiểu. Ở đây output sẽ trả về hello ứng với `this.message` và xin chao ứng với `this.message2`. Sử dụng phương thức này rất hữu ích khi xử lý việc đọc/ghi dữ liệu.

5.2.2. **beforeMount** và **mounted**

- **beforeMount()**

Được gọi trước khi DOM được hiển thị và gán cho component, trong phần này truy cập đến các phần tử DOM vẫn sẽ báo lỗi.

- **mounted ()**

Được gọi sau khi component được hiển thị lần đầu tiên, ở đây các element đã được hiển thị và khai báo cho phép truy cập DOM.

5.2.3. **beforeUpdate** và **update**

- **beforeUpdate()**

Khi dữ liệu được thay đổi, trước khi component rendered lại ở đây sẽ là nơi tốt nhất để cập nhật DOM theo cách thủ công trước bất kỳ thay đổi xảy ra.

BeforeUpdate có thể được sử dụng để theo dõi số lượng chỉnh sửa được thực hiện đối với một thành phần hoặc thậm chí theo dõi các hành động để tính năng hoàn tất.

- **update()**

Ở đây phương thức `updated` sẽ được gọi khi DOM đã được cập nhật. Đối với từng trường hợp sử dụng, chúng ta có thể muốn xem xét sử dụng trình theo để phát hiện dữ liệu thay đổi này. Đây là hook để chúng ta theo dõi tất cả các giá trị và giá trị mới đã thay đổi.

5.2.4. `beforeUnmount` và `unMounted`

- `beforeUnmount()`

Ở đây trước khi Component được hủy bỏ, ở trong giai đoạn này tất cả các thành phần vẫn hoạt động bình thường và chưa bị phá hủy vì thế chúng được sử dụng để quản lý tài nguyên các thành phần.

- `unMounted()`

Ở thời điểm này hầu hết các thành phần đã bị phá hủy. Nên không làm được gì nhiều và đây là giai đoạn cuối cùng của vòng đời.

II. NODEJS

NodeJS là một nền tảng được xây dựng trên V8 JavaScript Engine – trình thông dịch thực thi mã JavaScript, giúp xây dựng các ứng dụng web một cách đơn giản và dễ dàng mở rộng.

NodeJS được phát triển bởi Ryan Dahl vào năm 2009 và có thể chạy trên nhiều hệ điều hành khác nhau: OS X, Microsoft Windows, Linus.

1. Blocking I/O và Nonblocking I/O

I/O là quá trình giao tiếp (lấy dữ liệu vào, trả dữ liệu ra) giữa một hệ thống thông tin và môi trường bên ngoài. Với CPU, thậm chí mọi giao tiếp dữ liệu với bên ngoài cấu trúc chip như việc nhập/xuất dữ liệu với memory (RAM) cũng là tác vụ I/O.

Blocking I/O: Yêu cầu thực thi một IO operation, sau khi hoàn thành thì trả kết quả lại. Process/Thread gọi bị block cho đến khi có kết quả trả về hoặc xảy ra ngoại lệ.

Nonblocking I/O: Yêu cầu thực thi IO operation và trả về ngay lập tức (timeout = 0). Nếu operation chưa sẵn sàng để thực hiện thì thử lại sau. Tương đương với kiểm tra IO operation có sẵn sàng ngay hay không, nếu có thì thực hiện và trả về, nếu không thì thông báo thử lại sau.

2. Event Loop

NodeJS là một ứng dụng đơn luồng (Single Thread), nó hoạt động phía trên một nền tảng được viết bởi C++, nền tảng này sử dụng đa luồng (Multi Threads) để thực hiện đồng thời các nhiệm vụ.

Mỗi yêu cầu (request) từ phía người dùng được NodeJS coi là một sự kiện (event), chúng được đặt vào một Event Queue (Hàng đợi sự kiện). NodeJS sử dụng quy tắc FIFO (First In First Out), điều này có nghĩa là những yêu cầu đến trước sẽ được xử lý trước.

Event Loop là một vòng lặp vô tận, nó sẽ chuyển các yêu cầu sang Thread Pool (Bể chứa các luồng), đồng thời mỗi yêu cầu sẽ được đăng ký một hàm Callback. Khi một yêu cầu được xử lý xong, hàm Callback tương ứng sẽ được gọi thực thi.

Thread Pool là một chương trình viết bằng ngôn ngữ C++, nó hỗ trợ đa luồng (Multi Threads), chính vì vậy tại đây các yêu cầu sẽ được xử lý trên các luồng khác nhau. NodeJS cũng hỗ trợ đa tiến trình (Multi Processes), điều này có nghĩa là chúng có thể được thực thi trên các lõi (Core) khác nhau.

Khi yêu cầu được xử lý xong, NodeJS sẽ gọi hàm Callback (Đã được đăng ký cho yêu cầu này) để thực thi nó.

Nếu mỗi kết nối tới Server đều mở ra một luồng (Thread) sẽ rất tốn bộ nhớ. Điều này đã được chứng thực khi so sánh Apache và Nginx (Hai Web Server triển khai các ứng dụng PHP). Apache đã tiêu tốn bộ nhớ hơn rất nhiều so với Nginx.

NodeJS giống với Nginx là chúng chỉ sử dụng một luồng đơn (Single Thread) để đón tiếp các kết nối từ phía người dùng, và coi mỗi yêu cầu của người dùng là một sự kiện.

Các hoạt động I/O rất tốn tài nguyên của hệ thống, vì vậy NodeJS quản lý chặt chẽ việc sử dụng các hoạt động I/O. Vì vậy chỉ cần sử dụng Callback khi bạn thực thi các nhiệm vụ liên quan với I/O.

3. Synchronous và Asynchronous

Synchronous: Các sự kiện diễn ra theo thứ tự. Một sự kiện chỉ được bắt đầu khi sự kiện trước kết thúc.

Asynchronous: Không theo thứ tự, các hoạt động có thể xảy ra đồng thời hoặc chỉ ít, mặc dù các hoạt động bắt đầu theo thứ tự nhưng kết thúc thì không. Một hành động có thể bắt đầu (và thậm chí kết thúc) trước khi hành động trước đó hoàn thành.

4. Callback

Callback là một hàm sẽ được thực hiện sau khi một hàm khác đã thực hiện xong.

Trong JavaScript, hàm là đối tượng. Do đó, các hàm có thể lấy các hàm làm đối số và có thể được trả về bởi các hàm khác. Các hàm thực hiện điều này được gọi là higher – order function (Hàm bậc cao hơn). Bất kỳ hàm nào được truyền dưới dạng đối số được gọi là hàm Callback.

JavaScript là một ngôn ngữ lập trình hướng sự kiện và bất đồng bộ nên callback function đóng vai trò rất quan trọng.

ES6 định nghĩa 3 state cho một lời gọi hàm không đồng bộ:

- Pending: hàm đang được thực hiện và chưa trả về kết quả. Trong lúc này, nếu cố tình console.log biến kết quả sẽ nhận được output <pending>.
- Fulfilled: hàm được thực hiện xong – thành công và trả về kết quả.
- Rejected: hàm đã thực hiện xong - không thành công. Thường thì sẽ bắt exception tại bước này.

Máy chủ NodeJS có thể nhận rất nhiều yêu cầu (request) từ rất nhiều người dùng. Vì vậy để nâng cao khả năng phục vụ, tất cả các API của NodeJS được thiết kế hỗ trợ Callback

5. Module

NodeJS sử dụng kiến trúc Module để đơn giản hóa việc tạo ra các ứng dụng phức tạp. Module là giống như các thư viện trong C, C#, Java,... Mỗi module chứa một tập các hàm chức năng có liên quan đến một “đối tượng” của Module.

Các module được giữ tách biệt riêng với nhau, tách riêng với code base, khi nào cần sử dụng những cái nào thì gọi chúng ra và kết hợp lại với nhau tùy logic xử lý của bạn.

6. NPM – Node Package Manager

NPM viết tắt của Node Package Manager là một công cụ tạo và quản lý các thư viện lập trình JavaScript cho NodeJS. Trong cộng đồng JavaScript, các lập trình viên chia sẻ hàng trăm nghìn các thư viện với các đoạn code đã thực hiện sẵn một chức năng nào đó. Nó giúp cho các dự án mới tránh phải viết lại các thành phần cơ bản, các thư viện lập trình hay thậm chí các framework

Dựa theo chức năng mà chia package ra làm 2 loại, đó là Simple dependencies và Development dependencies.

- Simple dependencies là những package bắt buộc phải có trong quá trình chạy sản phẩm. Khi cài đặt Simple dependencies, NPM sẽ tự động cài đặt tất cả các dependencies cần thiết
- Development dependencies là những package bắt buộc khi phát triển cũng như phát hành sản phẩm. Khi cài đặt Development dependencies, NPM sẽ chỉ cài đặt các dependencies cần thiết.

III. THƯ VIỆN AXIOS

Axios là một thư viện HTTP client dựa trên Promise dùng để gửi các request HTTP bất đồng bộ đến REST endpoint để sử dụng các dịch vụ CRUD.

Promise là một cơ chế trong JavaScript giúp bạn thực thi các tác dụng bất đồng bộ mà không rơi vào callback hell hay pyramid of doom, là tình trạng các hàm callback lồng vào nhau ở quá nhiều tầng, thay vào đó là then – catch. Hiểu đơn giản là khi thực hiện một tác vụ bất đồng bộ, sau khi thực hiện thành công thì “then” sẽ được gọi, ngược lại khi có lỗi phát sinh thì “catch” sẽ được gọi. Tương tự như Promise, axios cũng có thể sử dụng theo kiểu gửi request lồng nhau.

Ngoài ra, axios còn hỗ trợ interceptor dùng để thực hiện một số công việc khác trước khi gửi request hoặc ngay khi nhận được response. Ví dụ như tạo mới access token khi nhận được response thông báo lỗi “401 —Unauthorized”.

IV. PHƯƠNG THỨC XÁC THỰC JWT

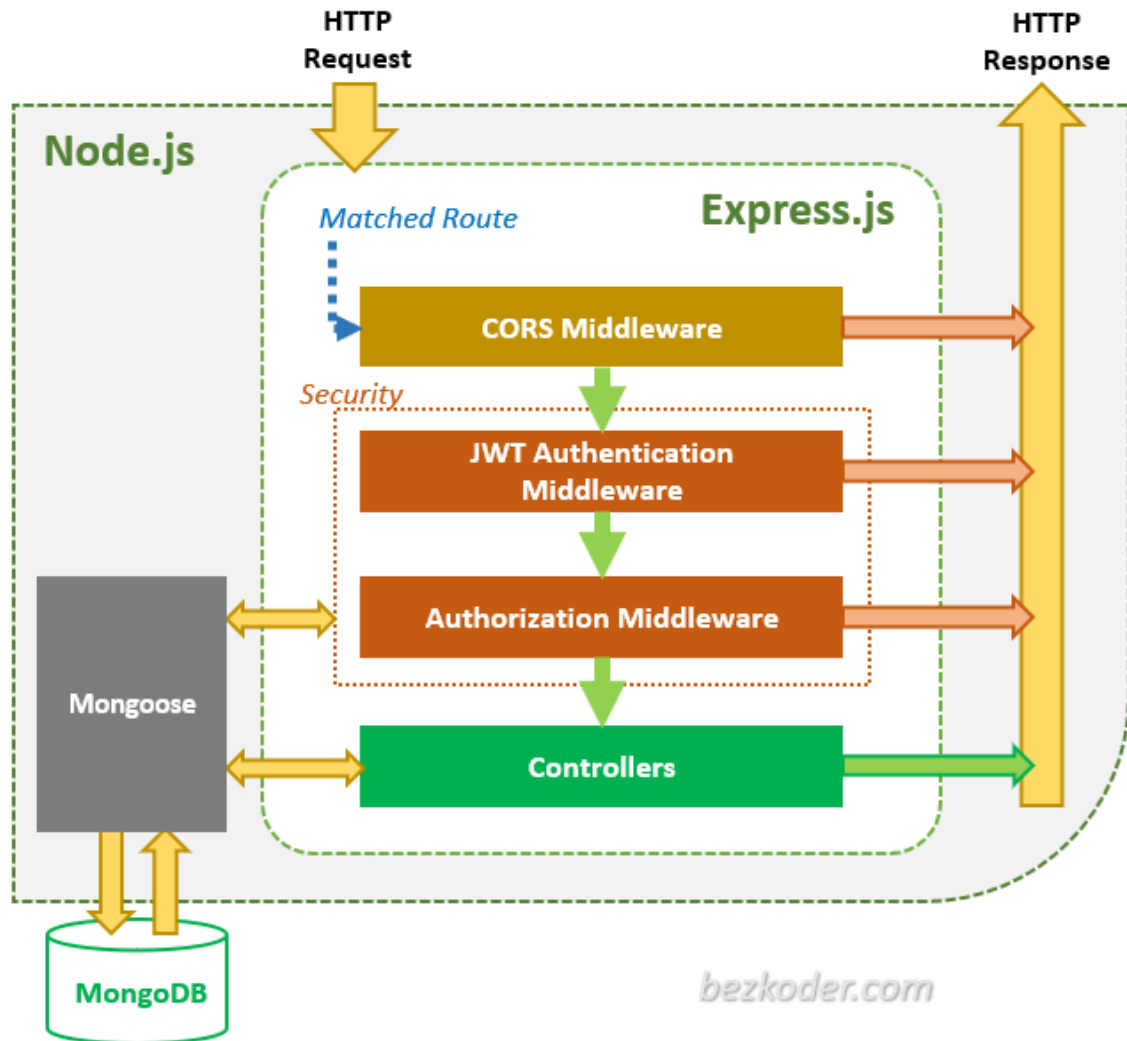
JWT được viết tắt từ JSON Web Token, đây là một phương tiện đại diện cho các bên yêu cầu chuyển giao giữa server và client. JWT cũng được xem như một tiêu chuẩn mở nhằm xác minh thông tin an toàn giữa các bên Client —Server dưới dạng JSON object.

Các chuỗi thông tin dạng JSON sẽ tiến hành mã hóa để trở thành một chuỗi ký tự lộn xộn, không có trật tự nhất định và rất khó hiểu khi nhìn vào. Thông tin này có thể được xác minh và tin cậy vì nó được ký điện tử – digitally signed. JWT có thể được ký bằng cách sử dụng một secret hoặc cặp public/private key.

- **Lợi ích của việc sử dụng JWT:**

Ủy quyền: Đây là trường hợp nên sử dụng JWT. Khi người dùng đã đăng nhập, mỗi request tiếp theo được gửi từ Client sẽ bao gồm JWT, cho phép người dùng access vào routes, service và resources được phép với token đó. Single Sign ON là tính năng sử dụng JWT rộng rãi hiện nay, vì chi phí thấp và dễ dàng sử dụng trên các domains khác nhau.

Trao đổi thông tin: JSON Web Token là một cách tốt để truyền thông tin an toàn giữa các bên Client và Server. Ví dụ, sử dụng các cặp public/private key, bạn có thể biết chắc người gửi. Ngoài ra, vì được xác định dựa vào header và payload, bạn cũng có thể xác minh rằng nội dung chưa bị giả mạo.



Hình 3. Mô hình kiến trúc tổng quan về kiến trúc NodeJS và ExpressJS sử dụng JWT

V. EXPRESSJS FRAMEWORK

ExpressJS là một framework được xây dựng trên nền tảng của NodeJS. Nó cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile. ExpressJS hỗ trợ các method HTTP và middleware tạo ra API vô cùng mạnh mẽ và dễ dàng sử dụng.

- Định tuyến (Routing)

Routing trong NodeJS là một khái niệm nói đến việc xác định ứng dụng sẽ đáp ứng như thế nào khi người dùng tạo một request đến một endpoint (Điểm cuối) cụ

thể nào đó. Điểm cuối đó thường là một URI hoặc một đường dẫn (Path) với một Request method (GET, POST, PUT,...) cụ thể.

- Cấu trúc định tuyến cơ bản: `app.METHOD (Path, Handler...)` Trong đó:

App: là một instance của `express`

METHOD: là một HTTP Method Express hỗ trợ rất nhiều loại HTTP methods khác nhau, bao gồm: `get`, `post`, `put`, `patch`, `delete`, `head`, `options`, `trace`, `copy`, `lock`, ...

Path: là đường dẫn trên máy chủ Route path có thể là một chuỗi thông thường (String) hoặc là một chuỗi có ký hiệu biểu thức chính quy (string patterns) hoặc là một biểu thức chính quy (regular expressions).

Handler: là một hàm function sẽ thực thi khi một route được trùng khớp. Đơn giản là một hoặc nhiều function sẽ được gọi khi một route trùng khớp để đáp ứng một yêu cầu nào đó. Lưu ý các handler sẽ được gọi đúng theo thứ tự truyền vào.

1. Route parameters

Route parameters là những vị trí trên URL được đánh dấu bằng cách đặt tên, mục đích là để lấy ra các giá trị tương ứng. Tất cả các giá trị đối số sẽ được đặt vào đối tượng request trong thuộc tính `params`. Với tên thuộc tính trùng khớp với từ khóa được xác định trên URL.

2. Response methods

Sau việc tiếp nhận và xử lý, thì việc tiếp theo đó là đáp ứng (response). Trong `express` định nghĩa sẵn một số phương thức hỗ trợ hay dùng nhất là:

Tên phương thức	Ý nghĩa
<code>res.json()</code>	Trả về một dữ liệu dạng JSON
<code>res.redirect()</code>	Chuyển hướng đến một đường dẫn nào đó
<code>res.render()</code>	Trả về một view template
<code>res.send()</code>	Gửi dữ liệu dạng text

Bảng 1. Một số phương thức (response methods) hỗ trợ thường sử dụng nhất

VI. MIDDLEWARE TRONG EXPRESSJS

ExpressJS khi hoạt động sẽ là một loạt các hàm Middleware được thực hiện liên tiếp nhau. Sau khi đã thiết lập, các request từ phía người dùng khi gửi lên **ExpressJS** sẽ thực hiện lần lượt qua các hàm **Middleware** cho đến khi trả về response cho người dùng. Các hàm này sẽ được quyền truy cập đến các đối tượng đại

diện cho **Request** – **req**, **Response** – **res**, hàm Middleware tiếp theo – **next**, và đối tượng lỗi – **err** nếu cần thiết.

Một hàm **Middleware** sau khi hoạt động xong, nếu chưa phải là cuối cùng trong chuỗi các hàm cần thực hiện, sẽ cần gọi lệnh **next()** để chuyển sang hàm tiếp theo, bằng không xử lý sẽ bị treo tại hàm đó.

Các chức năng mà middleware có thể thực hiện trong **ExpressJS** sẽ bao gồm :

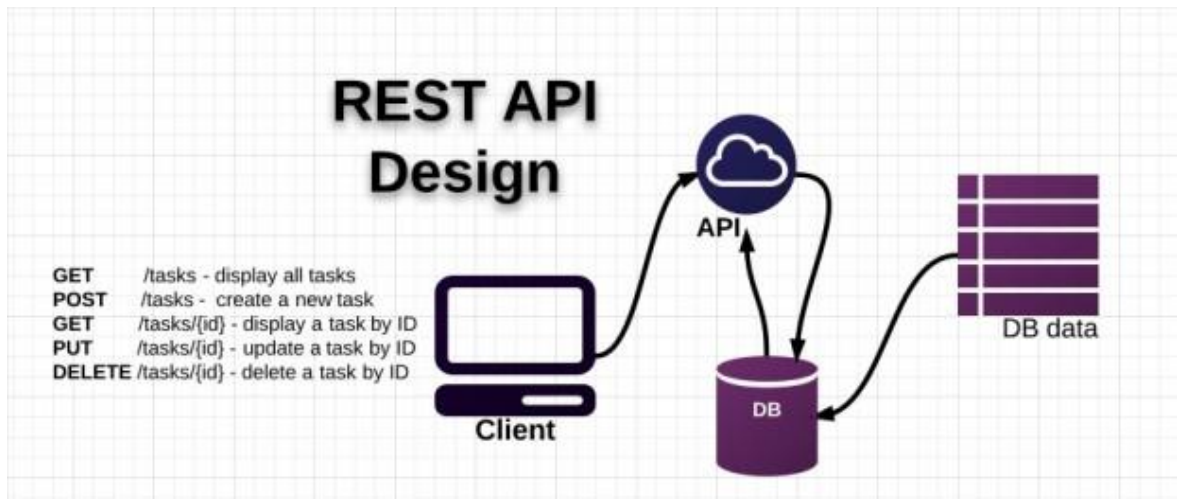
- Thực hiện bất cứ đoạn code nào
- Thay đổi các đối tượng request và response
- Kết thúc một quá trình request-response
- Gọi hàm middleware tiếp theo trong stack

Trong Express, có 5 kiểu middleware có thể sử dụng :

- Application-level middleware (middleware cấp ứng dụng)
- Router-level middleware (middleware cấp điều hướng – router)
- Error-handling middleware (middleware xử lý lỗi)
- Built-in middleware (middleware sẵn có)
- Third-party middleware (middleware của bên thứ ba)

VII. API VÀ RESTFUL API

API là viết tắt của Application Programming Interface (Giao diện lập trình ứng dụng) là một tập hợp các định nghĩa, giao thức, và công cụ chương trình con để xây dựng để xây dựng phần mềm và ứng dụng. Hiểu đơn giản API là một trung gian phần mềm cho phép hai ứng dụng giao tiếp với nhau. API có thể trả về dữ liệu mà bạn cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như JSON hay XML. Với API, bạn có thể tiếp cận, truy xuất dữ liệu từ máy chủ thể hiện chúng trên ứng dụng phần mềm hoặc website của mình một cách dễ dàng hơn. Tính tới nay, API đã phát triển với nhiều loại ứng dụng và phần mềm khác nhau. Thế hệ mới nhất của web/ app API có thể ứng dụng được ở mọi hệ thống từ cơ sở dữ liệu, hệ điều hành, hệ thống nền web, thư viện hay thậm chí là phần cứng máy tính.



Hình 4. Mô hình kiến trúc thành phần RESTful API

REST (Representational State Transfer) là một kiến trúc phần mềm bao gồm các quy tắc để tạo ra dịch vụ web (web service). Một webservice tuân thủ theo kiến trúc REST thì gọi là RESTful webservice. Webservice này sử dụng giao thức HTTP để triển khai kiến trúc web. Như vậy, RESTful API chính là kiến trúc thiết kế API tuân thủ theo kiến trúc REST thông qua các phương thức của HTTP (GET, POST, PUT, DELETE,...). Tương ứng với mỗi phương thức HTTP sẽ thực hiện những tác vụ tương ứng:

Phương thức HTTP	Tác vụ
GET (Read)	Lấy dữ liệu
POST (Create)	Tạo mới dữ liệu
PUT (Update)	Cập nhật, thay thế dữ liệu
DELETE (Delete)	Xóa dữ liệu

Bảng 2. Các tác vụ cơ bản của REST dựa trên phương thức HTTP

Các tác vụ đọc, tạo, cập nhật, xóa được gọi là CRUD service (Create, Read, Update, Delete). Mỗi tác vụ trên phải được gọi thông qua địa chỉ URI (Uniform Resource Identifier) kèm theo phương thức và payload (có thể có hoặc không, thường là định dạng XML hoặc JSON).

RESTful API sử dụng giao thức stateless (là một giao thức truyền thông không sử dụng session) và theo tiêu chuẩn nên hệ thống sẽ nhanh, đáng tin cậy và có thể mở rộng dễ dàng. Thông thường, RESTful API sẽ xác thực người dùng khi gửi yêu cầu đối với những tác vụ nguy hiểm như cập nhật hoặc xóa dữ liệu hoặc chỉ cho phép đối với người quản trị

VIII. CƠ SỞ DỮ LIỆU MONGODB

MongoDB là một hệ thống cơ sở dữ liệu NoSQL mã nguồn mở có kiến trúc hướng tài liệu (document-oriented), được thiết kế để lưu trữ một lượng lớn dữ liệu và cho phép bạn làm việc với dữ liệu đó một cách hiệu quả. Hệ thống cơ sở dữ liệu NoSQL cung cấp một giải pháp thay thế cho cơ sở dữ liệu quan hệ truyền thống sử dụng SQL (Ngôn ngữ truy vấn có cấu trúc). Trong MongoDB, dữ liệu được lưu trữ trong các document sử dụng cấu trúc giống JSON để biểu diễn và tương tác với dữ liệu

MongoDB hỗ trợ trên nhiều hệ điều hành: Windows, Linus, MacOS,...

Cơ sở dữ liệu MongoDB được phát triển và quản lý bởi MongoDB.Inc, được phát hành lần đầu vào năm 2009. Nó cũng hỗ trợ driver cho tất cả các ngôn ngữ phổ biến như C, C++, C# và .Net, Go, Java, NodeJS, Perl, PHP, Python,... Ngày nay, có rất nhiều công ty đã sử dụng MongoDB như Facebook, Nokia, eBay, Adobe, Google, ... để lưu trữ lượng lớn dữ liệu của họ.

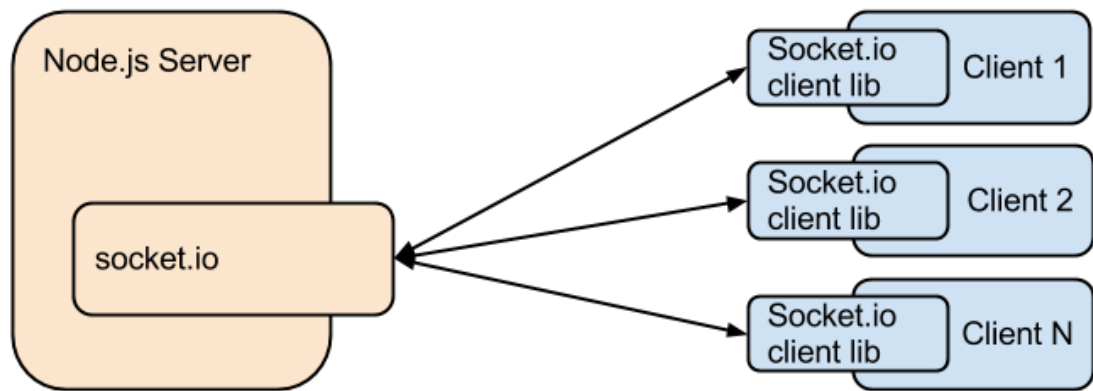
IX. SOCKET.IO

Socket.io là công cụ kết nối mở cho phép máy chủ và máy khách giao tiếp hai chiều với nhau theo thời gian thực. Khi máy chủ có Socket.io và máy khách có gói Socket.io trong trình duyệt thì việc liên kết sẽ được thực hiện

Socket.io là một module của NodeJs. Được xây dựng nhằm mục đích tạo ra real time NodeJS application. Socket.io cung cấp cho lập trình viên các đặc trưng như event, room và tự động phục hồi lại kết nối.

Khi chúng ta include Socket.io module vào trong ứng dụng của mình nó sẽ cung cấp cho chúng ta hai object đó là: socket server quản lý functionality phía server và socket client điều khiển functionality phía client.

Khi client muốn kết nối tới Socket.io server, nó sẽ gửi cho server một “handshake HTTP request”. Server sẽ phân tích request đó với những thông tin cần thiết trong suốt quá trình kết nối. Nó sẽ tìm cấu hình của middleware mà đã được đăng ký với server và thực thi chúng trước khi đưa ra sự kiện kết nối. Khi kết nối thành công thì connection event listener được thực thi, tạo ra một instance mới của socket có thể coi như định danh của client mà mỗi một client kết nối tới sẽ có 1 định danh. Các bạn có thể thấy rõ khi xem hình dưới đây



Hình 5. Socket.IO và các ứng dụng thời gian thực

Một module khác của Node.js là LightStreamer-adapter cũng có tạo các kết nối từ client tới server nhưng không trực tiếp mà thông qua LightStreamer Server, đó là các máy chủ theo thời gian thực và nằm ngoài tiến trình của Node.js Server

CHƯƠNG II. NỘI DUNG

I. MÔ TẢ ĐỀ TÀI

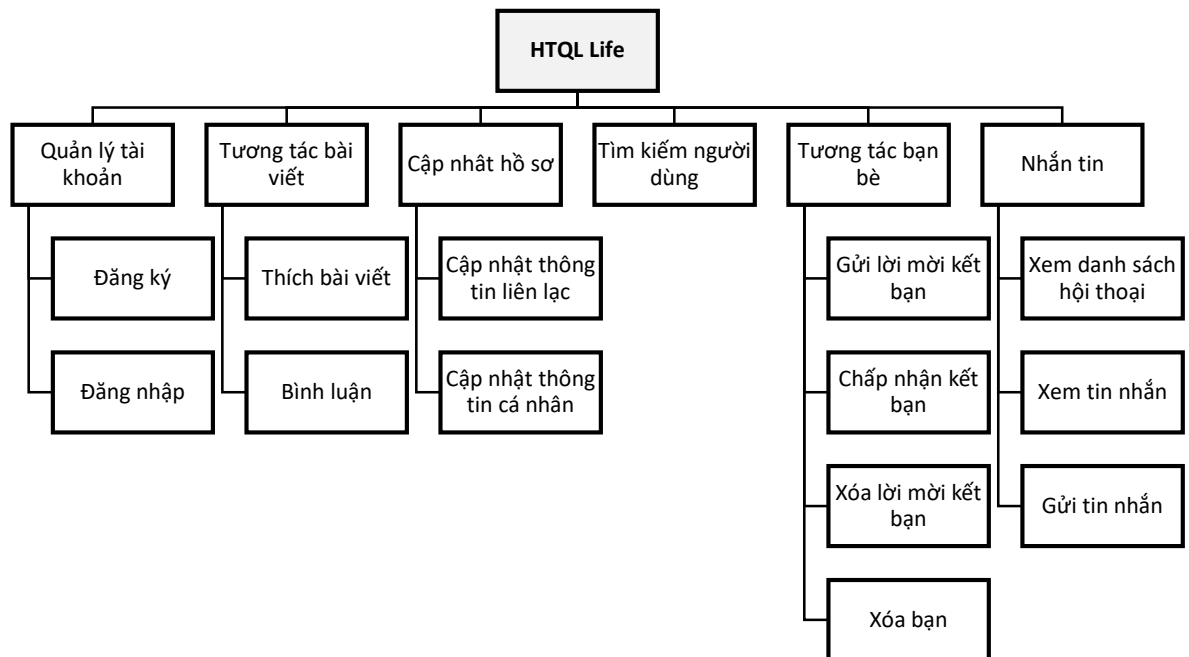
Hệ thống mạng xã hội Life là một nền tảng trực tuyến, nơi mà mọi người có thể xây dựng các mối quan hệ ảo với những người có chung tính cách, sở thích, ... hoặc với cả những người có mối quan hệ ngoài đời thực như bạn bè, gia đình.

Khi truy cập vào hệ thống, người dùng có thể dễ dàng đăng ký cho mình một tài khoản, chỉ sau một vài thao tác cung cấp thông tin cá nhân cơ bản. Sau khi đăng nhập thành công vào hệ thống bằng tài khoản vừa tạo, người dùng có thể cập nhật lại một số thông tin cá nhân để hoàn thiện trang cá nhân nếu muốn. Hoặc người dùng có thể ngay lập tức, tìm kiếm các người dùng khác qua tên, để gửi lời mời kết bạn với nhau. Bạn bè của nhau sẽ có thể nhắn tin trực tiếp qua lại với nhau một cách dễ dàng, nhanh chóng.

Một chức năng không thể thiếu của mạng xã hội là đăng bài viết. Sau khi đăng nhập, người dùng có thể đăng bài viết (có thể đính kèm ảnh). Ở trang chủ của hệ thống, người dùng sẽ có thể xem được các bài đăng của bạn bè. Người dùng có thể để lại tương tác cho các bài viết, thông qua hành động “Yêu thích” hoặc “Bình luận”.

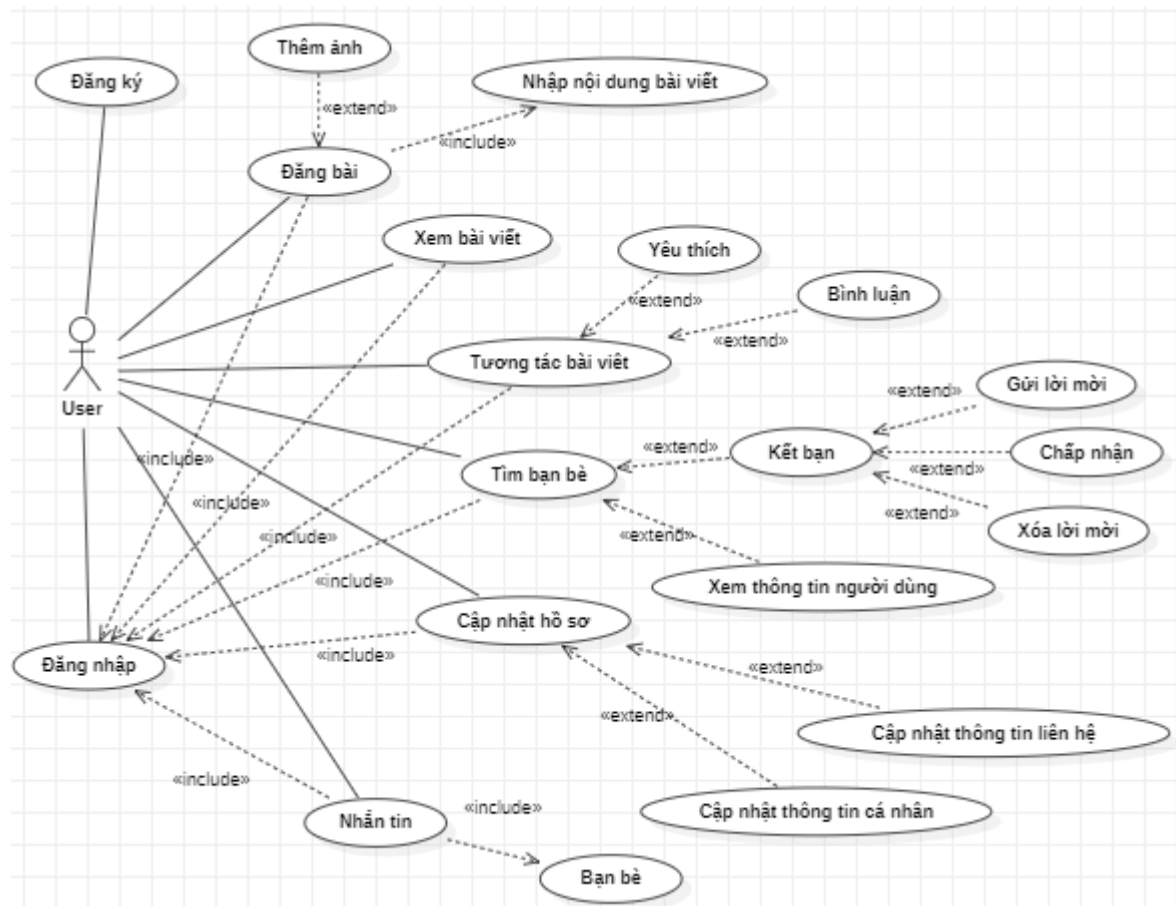
Thông qua hệ thống mạng xã hội Life, giúp cho mọi người có thể dễ dàng giữ liên lạc với bạn bè, người thân. Cũng như giúp mở rộng mối quan hệ, giao lưu, kết bạn một cách dễ dàng, nhanh chóng

II. CÁC CHỨC NĂNG CHÍNH CỦA HỆ THỐNG



Hình 6. Sơ đồ chức năng

III. USE CASE DIAGRAM



Người dùng khách (không có tài khoản): có thể đăng ký tài khoản để bắt đầu sử dụng các chức năng dành cho thành viên.

Người dùng thành viên (có tài khoản): sau khi đăng nhập có thể thực hiện các hành động như tìm kiếm bạn bè, kết bạn để xem bài đăng của người khác, đăng bài viết của cá nhân, yêu thích và hủy yêu thích bài đăng, bình luận bài đăng. Chỉnh sửa hồ sơ cá nhân và có thể chat cùng với bạn bè.

IV. THIẾT KẾ CƠ SỞ DỮ LIỆU MONGODB

1. Mô tả bảng dữ liệu

1.1. Bảng User

STT	Tên thuộc tính	Kiểu dữ liệu	Khóa	Diễn giải
1	_id	ObjectId	PK	Id người dùng
2	username	String		Tài khoản

3	password	String		Mật khẩu
4	fullname	String		Tên đầy đủ
5	firstname	String		Họ
6	lastname	String		Tên
7	gender	Boolean		Giới tính (true = Nam, false = Nữ)
8	email	String		Địa chỉ email
9	phone	String		Số điện thoại
10	date_birth	Date		Ngày sinh
11	places_lived	String		Nơi sinh sống
12	intro	String		Giới thiệu
13	friends_list	Array		Danh sách Id người dùng là đã kết bạn
14	avatar	Object		Chứa đường dẫn và tên ảnh
15	avatar_data	String		Đường dẫn ảnh đến cloud
16	avatar_name	String		Tên ảnh trong cloud

Bảng 3. Bảng CSDL User

1.2. Bảng Post

STT	Tên thuộc tính	Kiểu dữ liệu	Khóa	Diễn giải
1	_id	ObjectId	PK	Id bài đăng
2	title	String		Tiêu đề bài đăng
3	content	String		Nội dung bài đăng
4	image	Object		Chứa đường dẫn và tên ảnh
5	img_data	String		Đường dẫn ảnh đến cloud
6	img_name	String		Tên ảnh trong cloud
7	changed	Boolean		Có sự thay đổi bài đăng hay không

8	favorites_list	Array		Danh sách Id người dùng đã yêu thích bài đăng
9	_uid	String	FK	Id người dùng

Bảng 4. Bảng CSDL Post

1.3. Bảng Comment

STT	Tên thuộc tính	Kiểu dữ liệu	Khóa	Diễn giải
1	_id	ObjectId	PK	Id bình luận
2	content	String		Nội dung bình luận
3	date_created	Date		Ngày tạo
4	changed	Boolean		Có sự thay đổi bình luận hay không
5	_uid	String	FK	Id người dùng
6	_pid	String	FK	Id bài đăng

Bảng 5. Bảng CSDL Comment

1.4. Bảng News

STT	Tên thuộc tính	Kiểu dữ liệu	Khóa	Diễn giải
1	_id	ObjectId	PK	Id tin tức
2	_receiveUid	String	FK	Id người nhận
3	_sendUid	String	FK	Id người gửi
4	type	Object		Chứa kiểu tin tức
5	add_friend	Boolean		Kiểu tin tức kết bạn
6	new_comment	Boolean		Kiểu tin tức có bình luận mới trong bài đăng của người dùng
7	readed	Boolean		Đã đọc hay chưa
8	confirm	Boolean		Đã xác nhận hay chưa

Bảng 6. Bảng CSDL News

1.5. Bảng Conversation

Tên thuộc tính	Kiểu dữ liệu	Khóa	Diễn giải
_id	ObjectId	PK	Id cuộc hội thoại
members	Array	FK	Chứa danh sách Id người dùng
createdAt	Date		Ngày tạo

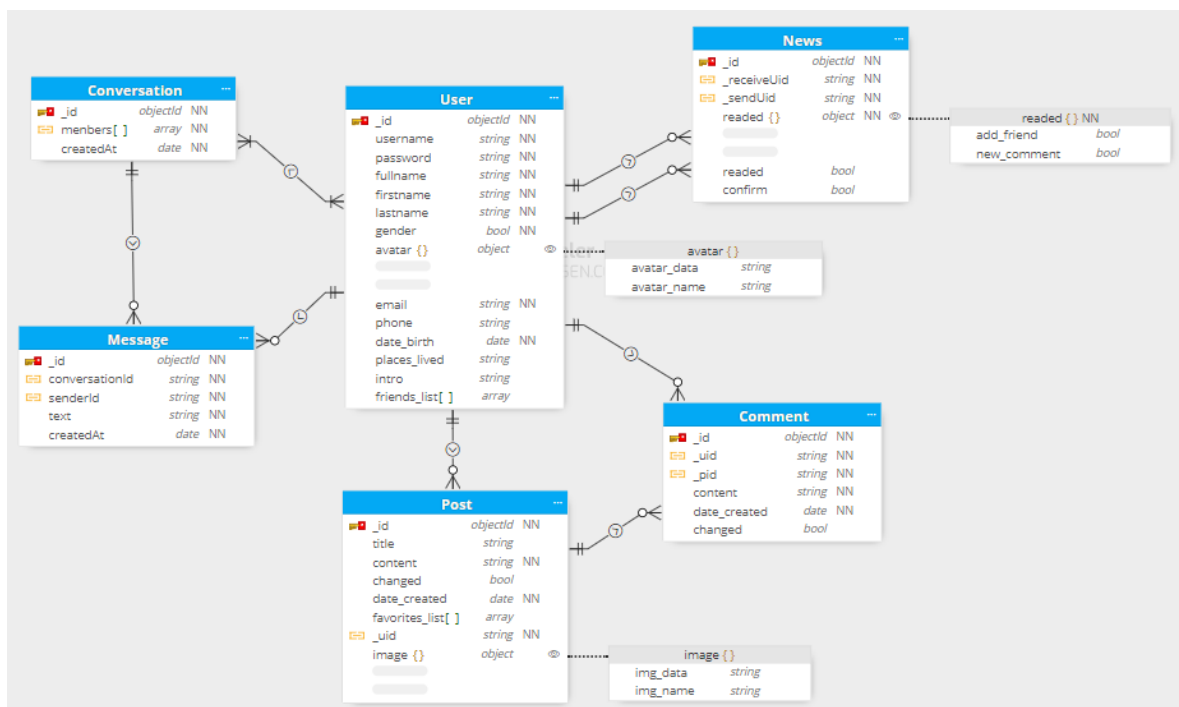
Bảng 7. Bảng CSDL Conversation

1.6. Bảng Message

Tên thuộc tính	Kiểu dữ liệu	Khóa	Diễn giải
_id	ObjectId	PK	Id tin nhắn
conversationId	String	FK	Id cuộc hội thoại
senderId	String	FK	Id người gửi
text	String		Nội dung tin nhắn
createdAt	Date		Ngày tạo

Bảng 8. Bảng CSDL Message

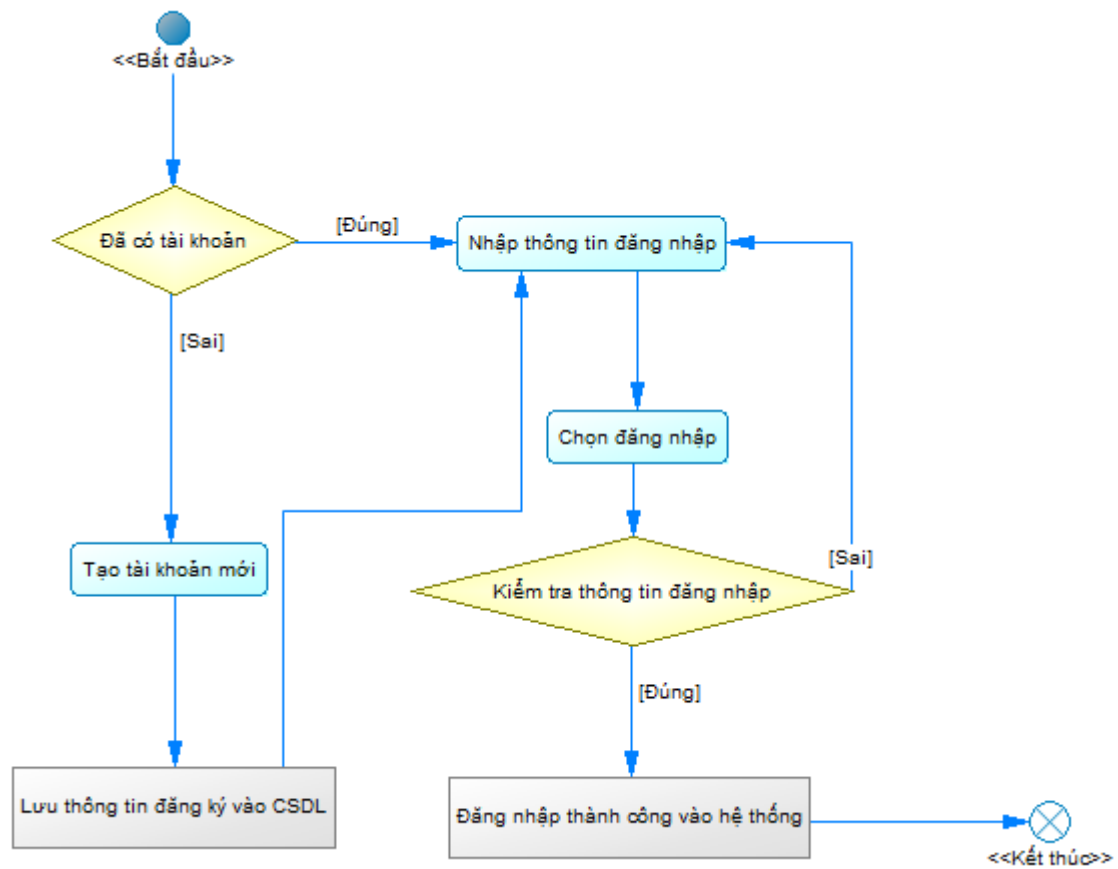
2. Mô hình dữ liệu mức quan hệ



Hình 7. Mô hình dữ liệu mức quan hệ MongoDB

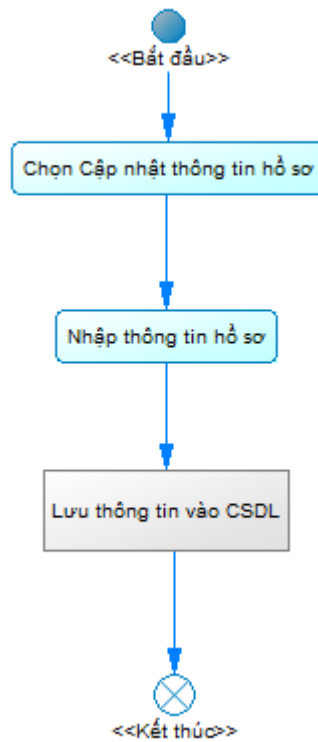
V. THIẾT KẾ CÁC CHỨC NĂNG

1. Đăng ký và Đăng nhập



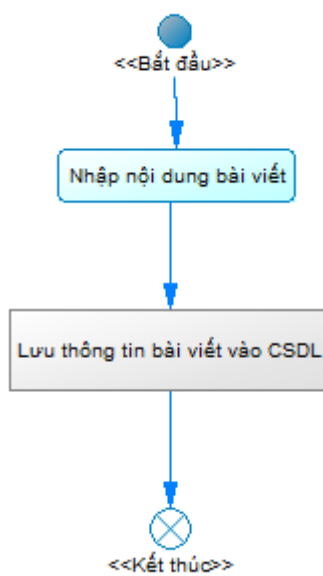
Hình 8. Lưu đồ chức năng đăng nhập và đăng ký

2. Cập nhật hồ sơ



Hình 9. Lưu đồ chức năng cập nhật hồ sơ

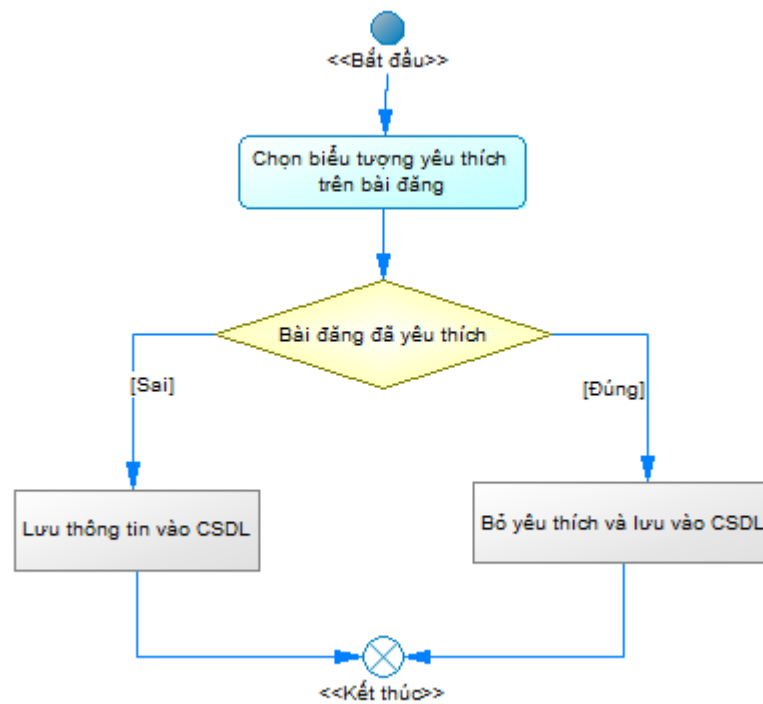
3. Đăng bài viết



Hình 10. Lưu đồ chức năng đăng bài viết

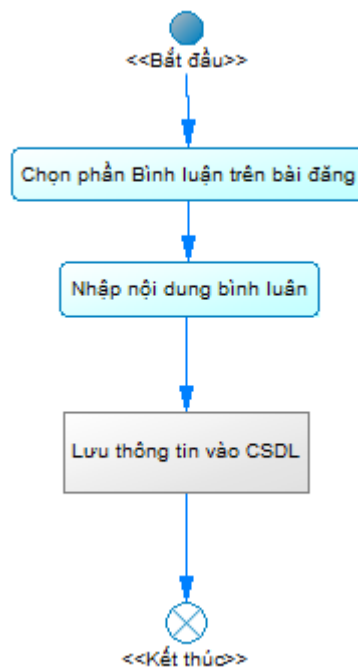
4. Tương tác bài đăng

4.1. Yêu thích bài đăng



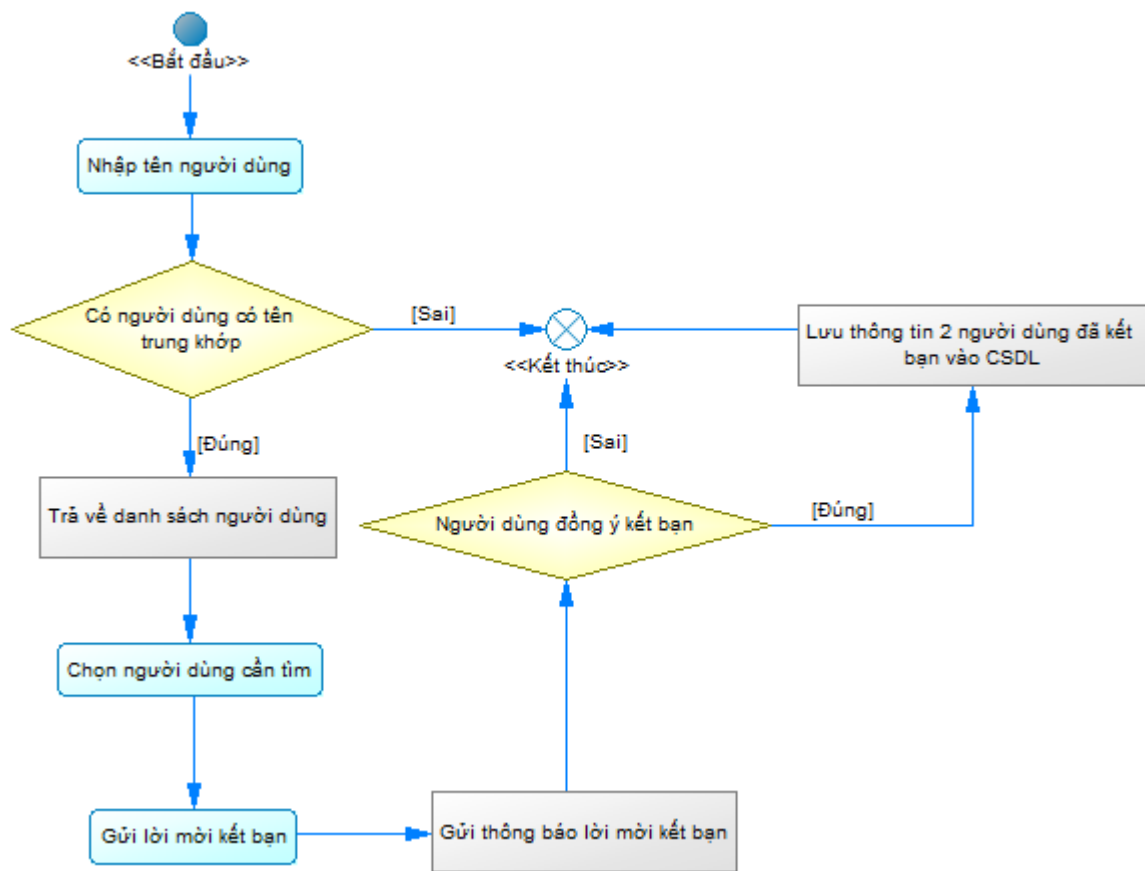
Hình 11. Lưu đồ chức năng yêu thích bài đăng

4.2. Bình luận bài đăng



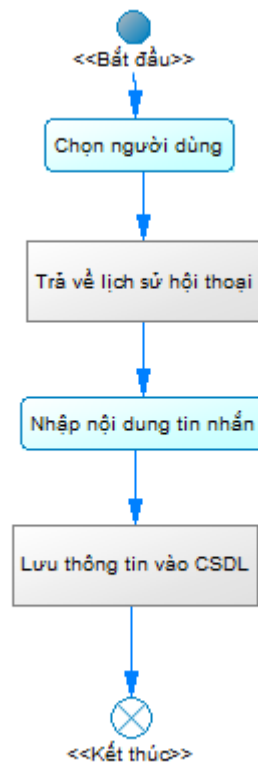
Hình 12. Lưu đồ chức năng bình luận

5. Kết bạn



Hình 13. Lưu đồ chức năng kết bạn

6. Nhắn tin

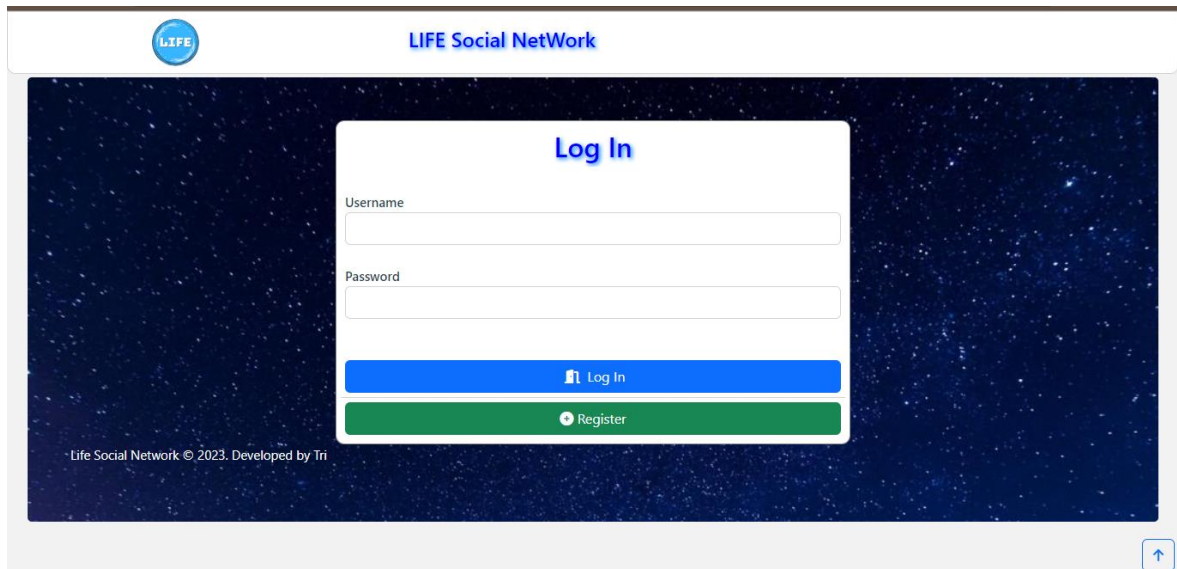


Hình 14. Lưu đồ chức năng nhắn tin

CHƯƠNG III. KẾT QUẢ NGHIÊN CỨU

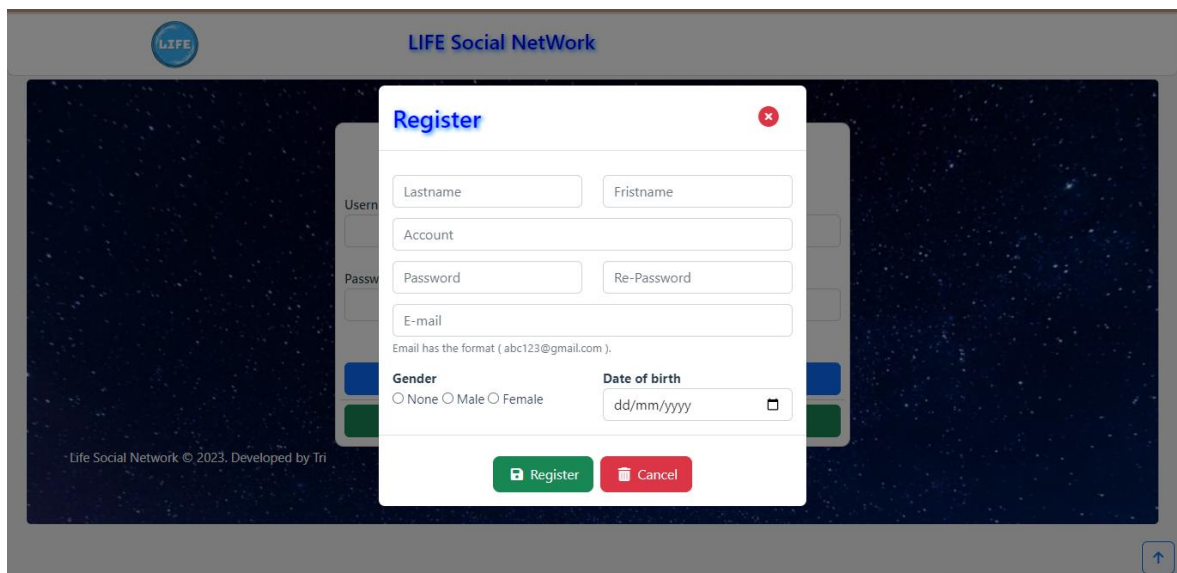
I. GIAO DIỆN ĐĂNG KÝ VÀ ĐĂNG NHẬP

1. Đăng nhập



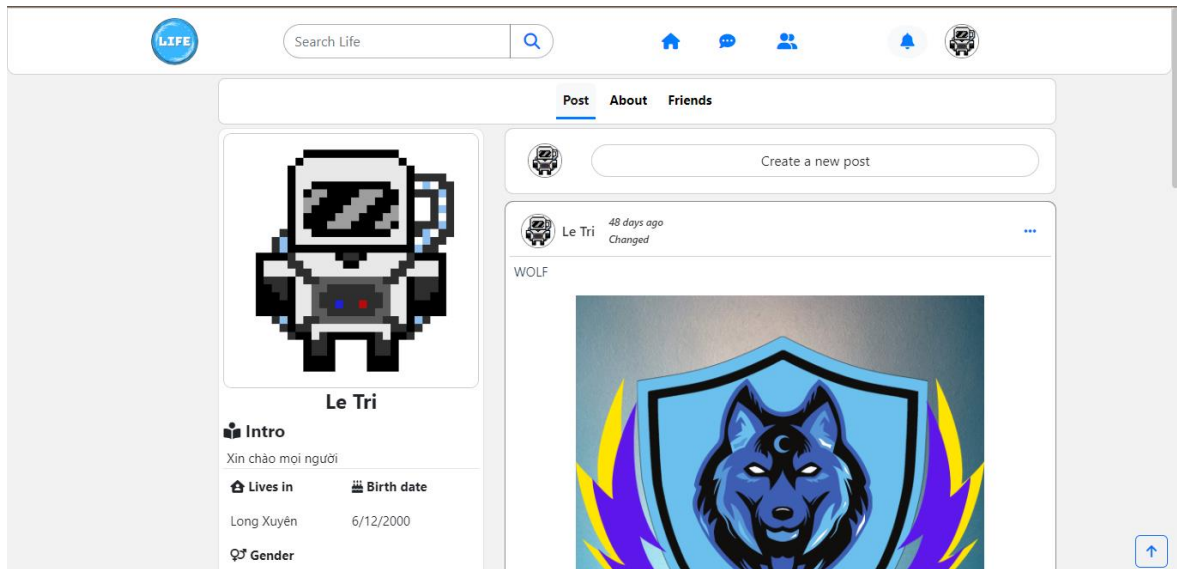
Hình 15. Trang đăng nhập

2. Đăng ký

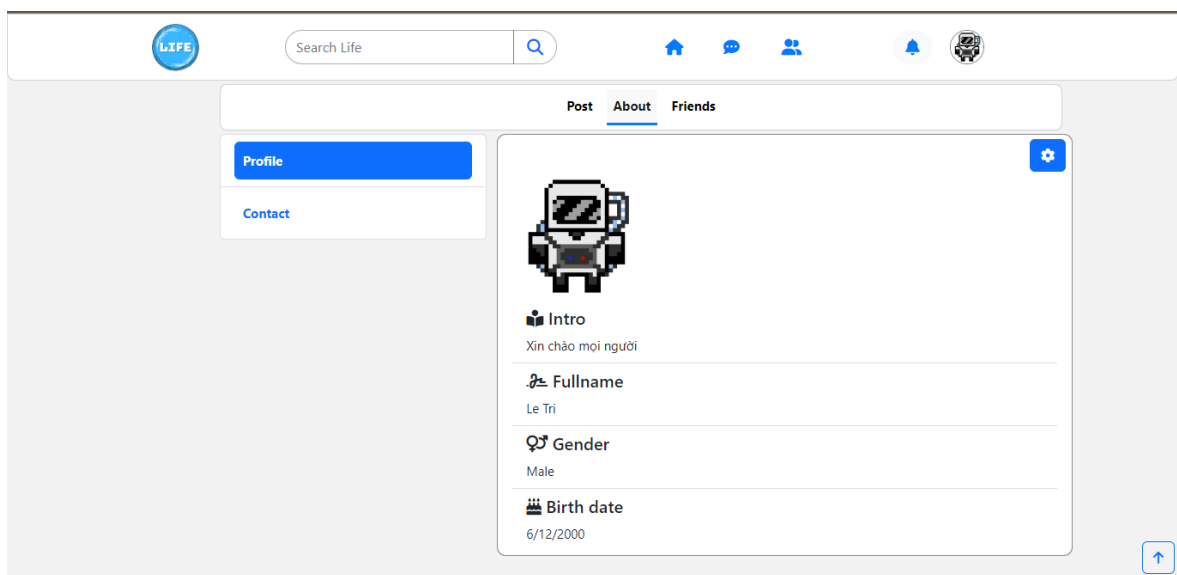


Hình 16. Trang đăng ký

II. GIAO DIỆN XEM HỒ SƠ



Hình 17. Trang bài đăng của người dùng



Hình 18. Trang hồ sơ của người dùng

III. GIAO DIỆN CẬP NHẬT HỒ SƠ

The screenshot shows a web application interface with a sidebar containing 'Profile' and 'Contact' tabs. The 'Edit Profile' modal is open, displaying the following fields:

- Lastname:** Text input with value 'Le'.
- Firstname:** Text input with value 'Tri'.
- Gender:** Radio buttons for 'None', 'Male', and 'Female'.
- Date of birth:** Text input with value '06/12/2000' and a calendar icon.
- Intro:** Text area with value 'Xin chào mọi người'.

At the bottom of the modal are two buttons: 'Save changes' (green) and 'Cancel' (red).

Hình 19. Chỉnh sửa thông tin hồ sơ

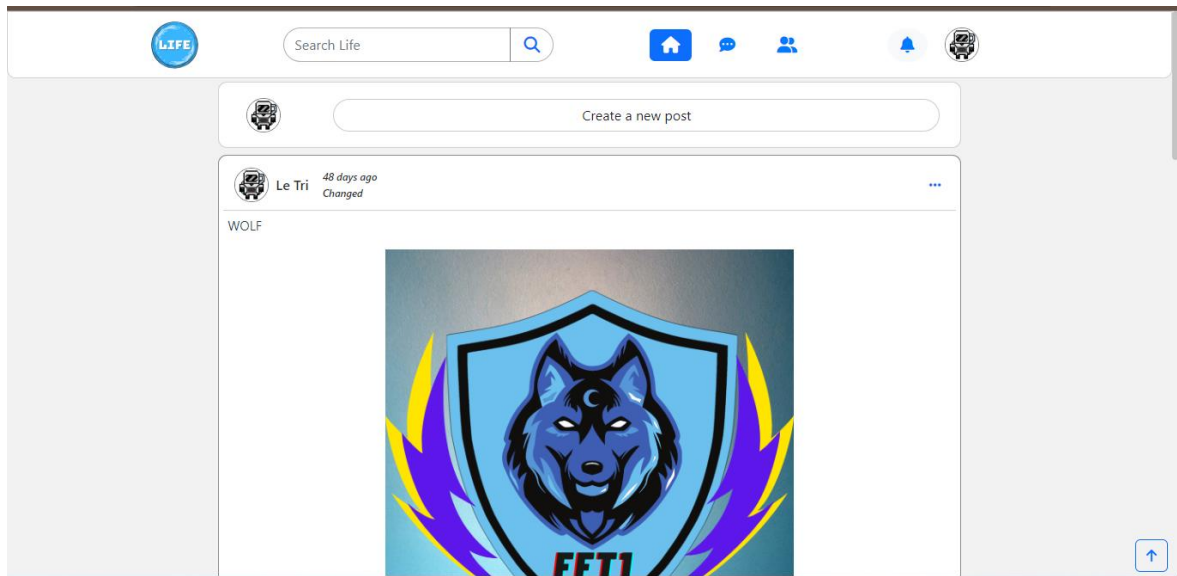
The screenshot shows the same web application interface, but with the 'Edit Contact' modal open. The 'Contact' tab is selected in the sidebar. The modal contains the following fields:

- Address:** Text input with value 'Long Xuyên'.
- Email:** Text input with value 'tri123@gmail.com'. Below the input is a hint: 'Email has the format (abc123@gmail.com).'.
- Phone:** Text input with value '0937425230'. Below the input is a hint: 'Phone number starts (09|03|07|08|05).'.

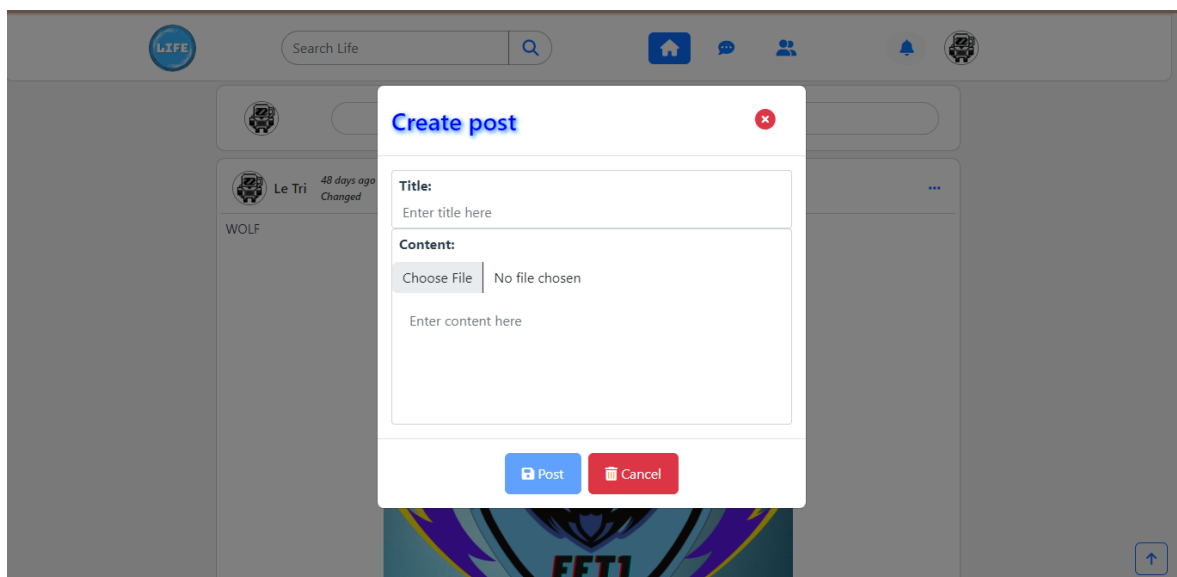
At the bottom of the modal are two buttons: 'Save changes' (green) and 'Cancel' (red).

Hình 20. Chỉnh sửa thông tin liên hệ

IV. GIAO DIỆN ĐĂNG BÀI VIẾT

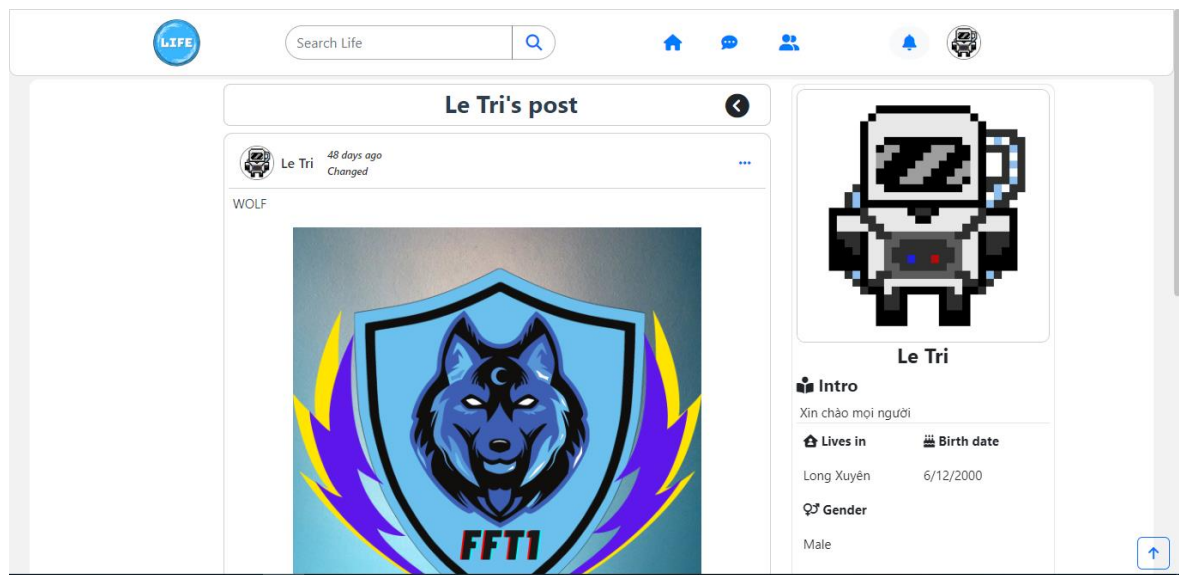


Hình 21. Trang chủ với những bài đăng

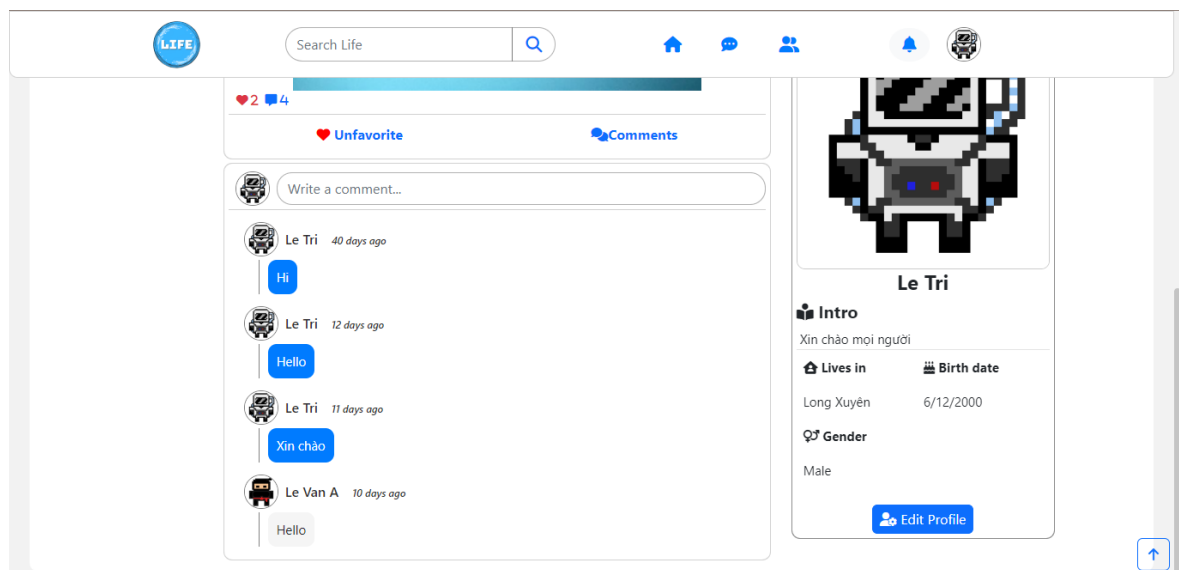


Hình 22. Trang đăng bài viết

V. GIAO DIỆN TƯƠNG TÁC BÀI ĐĂNG

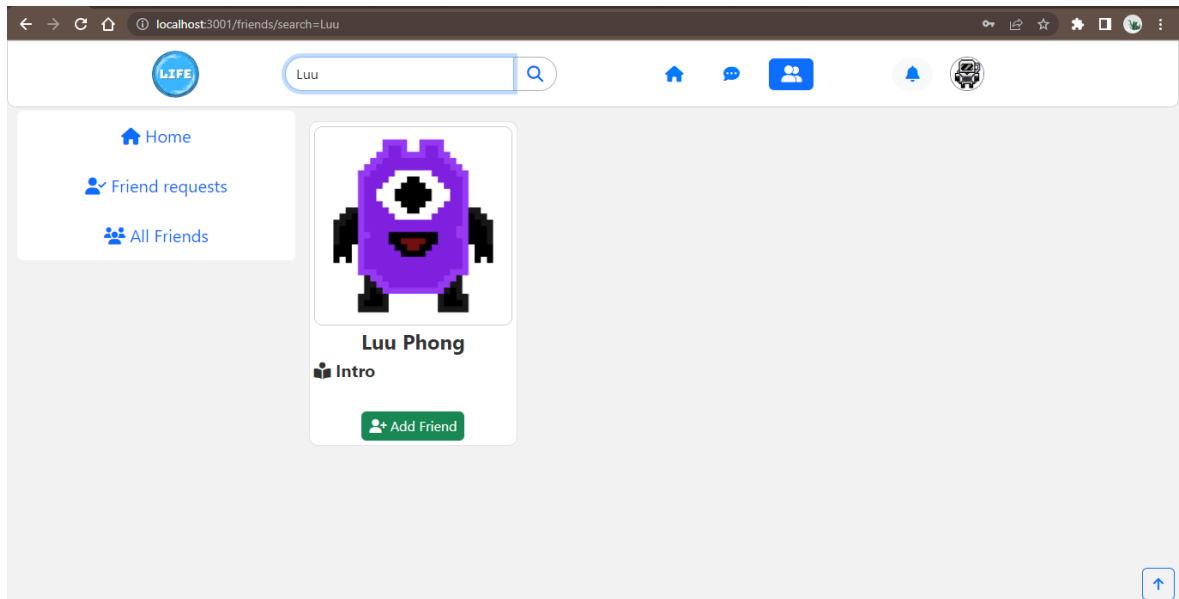


Hình 23. Trang chi tiết bài đăng – phần thông tin bài đăng



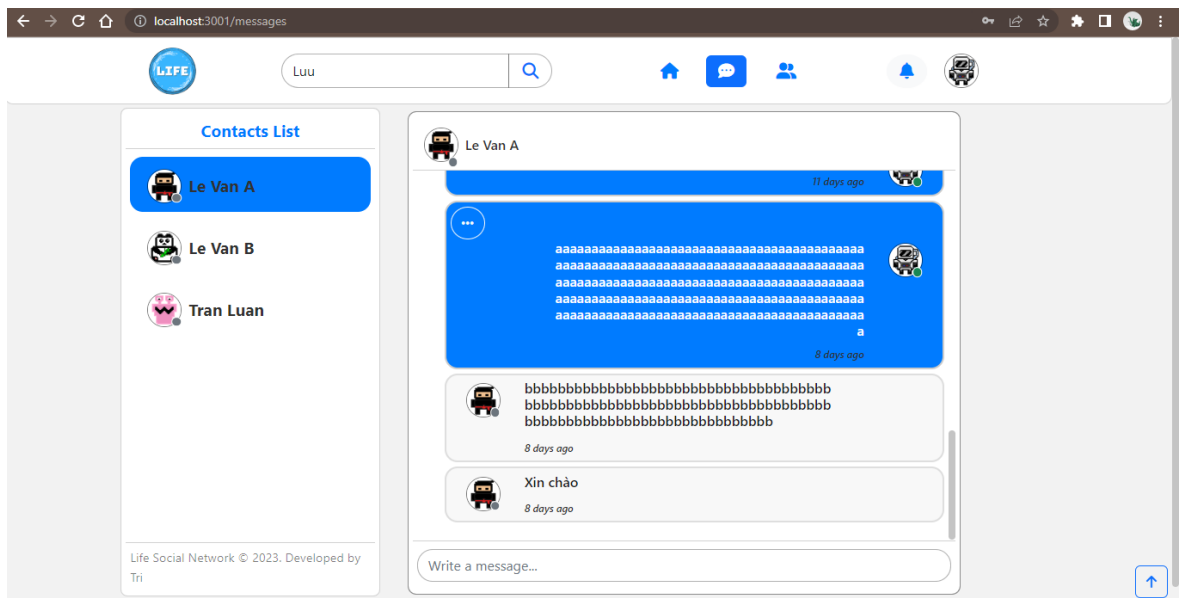
Hình 24. Trang chi tiết bài đăng – phần bình luận

VI. GIAO DIỆN KẾT BẠN



Hình 25. Trang tìm kiếm và kết bạn

VII. GIAO DIỆN NHẮN TIN



Hình 26. Trang hội thoại với bạn bè

PHẦN 3. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

I. KẾT LUẬN

1. Kết quả đạt được

Mạng xã hội gần như có những tính năng nên có, đạt được mục đích ban đầu mong muốn:

- Thiết kế thành công hệ thống trên nền tảng web
- Xây dựng được một số chức năng của hệ thống mạng xã hội
- Hệ thống có thể chạy trên nhiều trình duyệt
- Giúp mọi người có cơ hội để tiếp cận nhau hơn và chia sẻ những khoảnh khắc cùng nhau

2. Hạn chế

Giao diện Web còn đơn giản, còn thiếu nhiều chức năng như chia sẻ video hay các loại file khác ví dụ word, excel,... chưa có phần trả lời bình luận và một số tính năng hoạt động chưa thật sự mượt mà. Cấu trúc cơ sở dữ liệu chưa được chắc chắn và đơn giản, an toàn bảo mật của hệ thống chưa được cao.

II. HƯỚNG PHÁT TRIỂN

Phát triển thêm nhiều tính năng, cải thiện hiệu năng hoạt động của web tốt hơn, cải thiện độ mượt mà của các chức năng và logic dữ liệu hơn,...

TÀI LIỆU THAM KHẢO

- [1] VueJS [online]. Available: [Vue.js - The Progressive JavaScript Framework | Vue.js \(vuejs.org\)](https://vuejs.org/)
- [2] NodeJS [online]. Available: [Node.js \(nodejs.org\)](https://nodejs.org/)
- [3] ExpressJS [online]. Available: [Express - Node.js web application framework \(expressjs.com\)](https://expressjs.com/)
- [4] Axios [online]. Available: [Bắt đầu | Tài liệu Axios \(axios-http.com\) - https://axios-http.com/vi/docs/intro](https://axios-http.com/vi/docs/intro)
- [5] Json Web Token [online]. Available: [JSON Web Token Introduction - jwt.io](https://jwt.io)
- [6] API & RESTful API [online]. Available: [RESTful API là gì? Cách thiết kế RESTful API | TopDev \(https://topdev.vn/blog/restful-api-la-gi/\)](https://topdev.vn/blog/restful-api-la-gi/)
- [7] MongoDB [online]. Available: [TÌM HIỂU VỀ MONGODB \(viblo.asia\) - https://viblo.asia/p/tim-hieu-ve-mongodb-4P856ajGIY3](https://viblo.asia/p/tim-hieu-ve-mongodb-4P856ajGIY3)
- [8] Socket.io [online]. Available: [Cơ bản về Socketio \(viblo.asia\) - https://viblo.asia/p/co-ban-ve-socketio-bJzKm0kY59N](https://viblo.asia/p/co-ban-ve-socketio-bJzKm0kY59N)
- [9] Middleware trong ExpressJS [online]. Available: [Tìm hiểu về middleware trong ExpressJS \(viblo.asia\) - https://viblo.asia/p/tim-hieu-ve-middleware-trong-expressjs-gVQelwaaGZJ](https://viblo.asia/p/tim-hieu-ve-middleware-trong-expressjs-gVQelwaaGZJ)