

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



BÁO CÁO THỰC TẬP TỐT NGHIỆP

TÌM HIỂU FRAMEWORK NODE.JS VÀ ANGULAR
DÙNG XÂY DỰNG WEBSITE TIN TỨC MINH HỌA

NGÀNH: CÔNG NGHỆ THÔNG TIN
CHUYÊN NGÀNH: CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN
Giảng viên: Trần Đức Doanh

SINH VIÊN THỰC HIỆN:
Đào Tùng Lâm; MSSV: 1551120097

TP. HỒ CHÍ MINH – 18/5/2020

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

TP.HCM, ngày ... tháng ... năm 2020

Giáo viên hướng dẫn

This image shows a full page of white paper with horizontal dotted lines. The lines are evenly spaced and run across the width of the page, providing a guide for handwriting practice. There are no margins, text, or other markings on the page.

Giáo viên phản biện

LỜI CAM ĐOAN

Tôi tên Lương Công Cường và Đào Tùng Lâm xin cam đoan tự thực hiện Thực Tập Tốt Nghiệp, không sao chép Thực Tập Tốt Nghiệp khác dưới bất kỳ hình thức nào, các số liệu trích dẫn trong Thực Tập Tốt Nghiệp là trung thực, các số liệu, thông tin sử dụng trong bài Báo cáo Thực Tập Tốt Nghiệp này được thu thập từ nguồn thực tế, trên các sách báo khoa học chuyên ngành (có trích dẫn đầy đủ và theo đúng quy định). Nội dung trong báo cáo này do kinh nghiệm của bản thân được rút ra từ quá trình nghiên cứu tìm hiểu và thực tế KHÔNG SAO CHÉP từ các nguồn tài liệu, báo cáo khác.

Nếu có bất kỳ sai sót nào, Chúng tôi xin hoàn toàn chịu trách nhiệm theo quy định của Nhà Trường và Pháp luật.

MỤC LỤC

DANH MỤC CÁC TỪ VIẾT TẮT	8
DANH MỤC HÌNH ẢNH.....	8
DANH MỤC BẢNG	9
LỜI MỞ ĐẦU	10
PHẦN I: TÌM HIỂU VÀ CÀI ĐẶT MÔI TRƯỜNG.....	12
CHƯƠNG 1: GIỚI THIỆU VỀ NODEJS	12
1.1 Giới thiệu về NodeJS.....	12
1.1.1 NodeJS là gì và tại sao nên học NodeJS.	12
1.1.2 Ứng dụng của NodeJS.....	13
1.1.3 Hệ sinh thái của NodeJS.....	13
1.1.4. Cách thiết lập NodeJS.	14
1.2. Tầm quan trọng của JavaScript.....	15
1.2.1. Giới thiệu lại các khái niệm quan trọng của JS.	15
1.2.2. Tính cần thiết của ES6-7.	16
1.2.3. Phiên bản ECMAScript và NodeJS	17
1.2.4. Công Cụ hỗ trợ lập trình.....	18
CHƯƠNG 2: ỨNG DỤNG CỦA NODEJS	19
2.1. Chạy ứng dụng NodeJS với terminal	19
2.1.1. Viết code tạo server.....	19
2.1.2. Chạy server NodeJS.	19
2.2. Node package manager(npm) và NodeJS module system	20
2.2.1. Node package manager(npm)	20
2.2.2. NodeJS module system.	22
2.3. NodeJS core packages.	24
2.4. Xử lý yêu cầu API cho ứng dụng NodeJS.....	25
2.5. Xử lý file trong ứng dụng NodeJS.	25
2.5.1 Mở một File trong Node.js	25
2.5.2 Lấy thông tin File trong Node.js.....	27
2.5.3 Ghi dữ liệu vào File trong Node.js.....	28
2.5.4 Đọc dữ liệu từ File trong Node.js	29

2.5.5 Đóng File trong Node.js	29
2.5.6 Truncate một File trong Node.js	29
2.5.7. Xóa File trong Node.js	30
2.5.8 Tạo thư mục trong Node.js	30
2.5.9 Đọc thư mục trong Node.js.....	30
2.5.10 Xóa thư mục trong Node.js.....	31
2.6. Khái niệm về Callback	31
2.6.1. Callback là gì?.....	31
2.6.2. Ví dụ Node JS Callback	32
CHƯƠNG 3: TEMPLATE ENGINE HANDLEBARS	35
3.1 Giới thiệu	35
3.2 Khi nào nên sử dụng handlebars.js và tại sao nên sử dụng nó	35
3.3. Handlebars hoạt động như thế nào.....	36
CHƯƠNG 4: EXPRESS.JS VÀ API	38
4.1 Tạo server HTTP đơn giản với NodeJS	38
4.2 Giới thiệu về Express.	38
4.3 Render phản hồi HTML bằng chuỗi mẫu.....	39
4.4 Tạo các điểm cuối API restful.	40
4.5 Sử dụng postman để kiểm tra API.	46
CHƯƠNG 5: MONGODB	50
5.1 Giới thiệu về MongoDB và cách thiết lập MongoDB cục bộ.	50
5.2 Thao tác dữ liệu MongoDB.	51
PHẦN II: XÂY DỰNG WEBSITE	57
CHƯƠNG 6: XÂY DỰNG WEBSITE DEMO	57
6.1 Giới thiệu web demo Website Tin Tức	57
6.2 Nền tảng và công cụ phát triển	57
6.3 Chức năng.....	58
6.3.2 Danh sách các Actor.....	58
6.3.3. Danh sách các Use-case	59
6.4 Kết quả thực nghiệm	60
6.4.1 Tổ chức	60

6.4.1.1 Database	60
6.4.1.2 Tạo routes	64
6.4.1.3 Tạo controller.....	65
6.4.2 Thực hiện	68
6.4.2.1 Trang chủ.....	68
6.4.2.2 Trang đăng ký thành viên.....	69
6.4.2.3 Trang đăng nhập	69
6.4.2.4 Trang liên hệ.....	69
6.4.2.5 Trang xem chi tiết bài viết.....	70
6.4.2.6 Trang xem thông tin thành viên đang bài	71
6.4.2.7 Trang quản trị của Admin	71
6.4.2.8 Trang quản lý bài viết của thành viên.....	72
KẾT LUẬN.....	73
1. Kết quả đạt được	73
2. Hướng phát triển	73

DANH MỤC CÁC TỪ VIẾT TẮT

API: Application Program Interface

AJAX: Asynchronous Javascript and XML

CLI: Command Line Interface

ES6: ECMAScript 6

E4X: ECMAScript for XML

HTTP: Hypertext Transfer Protocol

HTML: Hypertext Markup Language

MSI: Microsoft Installer

Npm: Node package manager

PM: Postman

JS: javascript

FD: File Descriptor

SPA: Single Page Application

SQL: Structure Query Language

XML: Extensible Markup Language

DANH MỤC HÌNH ẢNH

Hình 1: Ví dụ callback

Hình 2: Kết quả chạy tập tin blocking-example.js

Hình 3: Kết quả chạy tập tin non-blocking-example.js

Hình 4: Danh sách kiểm thử và kết quả

Hình 5: Kết quả chạy test sử dụng Postman collection runner

Hình 6: Sơ đồ Use-case

Hình 7.1: Trang chủ

Hình 7.2: Trang đăng ký thành viên

Hình 7.3: Trang đăng nhập

Hình 7.4: Trang liên hệ

Hình 7.5: Trang xem chi tiết bài viết

Hình 7.6: Thông tin profile thành viên

Hình 7.7: Trang quản trị của Admin

Hình 7.8: Trang quản lý bài viết của thành viên

Hình 7.9: Một số chức năng của thành viên

DANH MỤC BẢNG

Bảng1: Một số mô-đun cốt lõi quan trọng trong Node.js.

Bảng 2: Các Flag được sử dụng cho hoạt động Đọc/Ghi file trong Node.js

Bảng 3: Một số phương thức hữu ích có thể được sử dụng để kiểm tra kiểu file

Bảng 4: Các tiêu chuẩn để thiết kế API

Bảng 5: Danh sách các tùy chọn

Bảng 6: Danh sách các Actor

Bảng 7: Danh sách các Use-case

LỜI MỞ ĐẦU

1. Tính cấp thiết của đề tài

Công nghệ thông tin càng ngày càng phát triển. Nhu cầu lao động đang tăng cao, thị trường nguồn lao động chưa đủ chất lượng. Cần một nguồn lực lớn, Lập trình web luôn trong tình trạng thiếu lao động tay nghề cao.

Nhu cầu thiết kế website đang tăng trưởng với tốc độ rất nhanh, đặc biệt đối với các doanh nghiệp thì việc sở hữu một website chuyên nghiệp lại có ảnh hưởng không nhỏ đến việc khẳng định uy tín cũng như quảng bá hình ảnh của mình trên thị trường.

Tốc độ truy cập là một trong những yếu tố quan trọng giúp website được đánh giá cao và thu hút được nhiều lượt truy cập. Một trang web load nhanh sẽ tạo được hứng thú với người dùng và khả năng họ ghé thăm lần sau sẽ cao hơn. Ngoài ra, tốc độ cũng ảnh hưởng đến thứ hạng trên bộ máy tìm kiếm và quyết định một phần kết quả SEO web. Vì vậy lựa chọn công nghệ để thiết kế web có tốc độ xử lý rất cần thiết trong cuộc sống hiện nay.

Nodejs là một nền tảng (Platform) phát triển độc lập được xây dựng ở trên Javascript Runtime của Chrome mà chúng ta có thể xây dựng được các ứng dụng mạng một cách nhanh chóng và dễ dàng mở rộng. Phần Core bên dưới của Nodejs được viết hầu hết bằng C++ nên cho tốc độ xử lý và hiệu năng khá cao. Nodejs tạo ra được các ứng dụng có tốc độ xử lý nhanh, realtime thời gian thực. Nodejs áp dụng cho các sản phẩm có lượng truy cập lớn, cần mở rộng nhanh, cần đổi mới công nghệ, hoặc tạo ra các dự án Startup nhanh nhất có thể.

Bất kỳ ai cũng có thể phát triển ứng dụng Node.js bằng cách sử dụng JavaScript và có thể chạy trên Microsoft Windows, Linux hoặc OS X.

2. Mục đích nghiên cứu

Trải qua quá trình làm việc và học tập tại trường bốn năm với những kinh nghiệm lập trình, bài báo cáo với mục đích: Từ những cơ sở lý luận đã học được trên trường và thông qua quá trình tìm hiểu công nghệ NodeJS từ đó đánh giá được ưu và nhược điểm của NodeJs so với các công nghệ khác như PHP, ASP.NET... đúc kết được những kinh nghiệm khi nào nên chọn công nghệ nào phù hợp với mục đích nhu cầu thiết kế web.

Xây dựng thành công website tin tức ba gồm các chức năng như:

- Khách hàng vẫn lai có thể xem các bài viết tin tức nhanh nhất
- Khách hàng có thể đăng ký trở thành một thành viên để có thể đăng bài viết của mình lên trang web
- Admin có chức năng quản lý các thành viên và các bài viết của họ

3. Nhiệm vụ nghiên cứu

Sau khi nghiên cứu cần xây dựng WebSite demo

Thời gian kết thúc nghiên cứu: từ 04/03/2020 đến 20/05/2020

4. Phương pháp nghiên cứu

Nghiên cứu được thực hiện bằng các phương pháp sau: Tìm hiểu thông qua đọc các nguồn sách, báo, tài liệu online liên quan đến công nghệ NodeJS, và các nguồn khác. Thực hành code trên phần mềm Visual Studio Code, MongoDB Compass.

5. Kết cấu của TTTN

Ngoài trang phụ bìa; mục lục; danh mục các từ viết tắt, lời mở đầu; kết luận; tài liệu tham khảo và phụ lục; bài báo cáo được chia làm 6 chương:

Chương 1: Giới thiệu về NodeJs

Chương 2: Ứng dụng của NodeJs

Chương 3: Template Engine Handlebars

Chương 4: Express.js và API

Chương 5: MongoDB

Chương 6: Xây dựng WebSite Demo

PHẦN I: TÌM HIỂU VÀ CÀI ĐẶT MÔI TRƯỜNG

CHƯƠNG 1: GIỚI THIỆU VỀ NODEJS

1.1 Giới thiệu về NodeJS.

1.1.1 NodeJS là gì và tại sao nên học NodeJS.

Node.js là một mã nguồn mở, một môi trường cho các máy chủ và ứng dụng mạng.

Node.js sử dụng Google V8 JavaScript engine để thực thi mã, và một tỷ lệ lớn các mô-đun cơ bản được viết bằng JavaScript. Các ứng dụng node.js thì được viết bằng JavaScript.

Node.js chứa một thư viện built-in cho phép các ứng dụng hoạt động như một Webserver mà không cần phần mềm như Nginx, Apache HTTP Server hoặc IIS.

Node.js cung cấp kiến trúc hướng sự kiện (event-driven) và non-blocking I/O API, tối ưu hóa thông lượng của ứng dụng và có khả năng mở rộng cao

Mọi hàm trong Node.js là không đồng bộ (asynchronous). Do đó, các tác vụ đều được xử lý và thực thi ở chế độ nền (background processing)

Còn đối với những bạn sinh viên (như mình) thì mình nghĩ một số lí do sau đây sẽ làm bạn không còn phân vân nên hay không nên học nữa:

- Node.js là phần nâng cao hơn của javascript. Cho nên rất phù hợp với người mới bắt đầu với lập trình, đã học qua frontend (html, css, js). Các bạn cần có kiến thức cơ bản về javascript.

- Có thể sử dụng 1 ngôn ngữ lập trình (javascript) vừa để xử lý giao diện (frontend) vừa xử lý logic (back-end), nên có 1 sự kết nối trong việc học, việc đọc và sửa code (chung cú pháp)

- Node.js cực kì dễ cài đặt và sử dụng. Bạn không cần thêm 1 phần mềm nào chạy server khác (như Glassfish/tomcat cho JavaWeb, Xampp cho php). Gần như tất cả mọi thứ chỉ trong 1 lần setup.
- Node.js miễn phí và chạy trên nhiều platform (windows, linux, Mac OS, ..)
- Node.js có 1 cộng đồng rất nhiều người học với số lượng built-in module (các package hỗ trợ) nhiều, cài đặt bằng npm.
- Học Node.js để viết những website thực tế, theo mình trực quan và hứng thú hơn rất nhiều so với việc viết code console trên lớp. Chưa kể lại với thời gian viết cho ra sản phẩm nhanh.

1.1.2 Ứng dụng của NodeJS.

Node.js thường được sử dụng để:

- Phát triển ứng dụng Real-time: mạng xã hội, chat, ...
- Ứng dụng Single Page Application (SPA): bởi 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: đây là các công cụ sử dụng command-line.

1.1.3 Hệ sinh thái của NodeJS.

Hệ sinh thái module npm mở (open source), ai cũng có thể truy cập và tải về miễn phí, cũng như tạo ra và upload module của riêng họ. Một số module npm phổ biến:

- **Express:** expressjs, một framework phát triển web dựa trên Sinatra, nền tảng chuẩn cho rất nhiều ứng dụng Node.js bây giờ.
- **Connect:** Connect là một framework server HTTP mở rộng, cung cấp một bộ các plugins “hiệu năng cao” như middleware, server, đồng thời cũng là nền tảng cho Express.

- **Socket.io** và **sockjs**: thành phần server-side của hai websocket thông dụng nhất hiện nay.
- **Jade**: một trong những template engine phổ biến (chuẩn viết html), dựa trên HTML, nền tảng của Expressjs.
- **Mongo** và **mongojs**: gói MongoDB cung cấp những APIs cho hệ cơ sở dữ liệu đối tượng MongoDB trong Node.js
- **Redis**: thư viện người dùng.
- **CoffeeScript**: trình biên dịch CoffeeScript cho phép những nhà phát triển viết nên chương trình Node.js bằng Coffee (uống cà phê là viết được Node.js <3)
- **Underscore** (lodash, lazy): thư viện cung cấp rất nhiều tiện ích, được sử dụng phổ biến nhất trong Javascript, được gói gọn trong Node.js, với cách thực thi khác biệt, hứa hẹn sẽ mang đến hiệu năng tốt hơn.
- **Forever**: rõ ràng đây là tiện ích thông dụng nhất, giúp cho mã Node chạy một cách liên tục.

Danh sách trên đây chỉ mang tính giới thiệu. Thực tế còn rất nhiều các module mạnh mẽ khác nữa.

1.1.4. Cách thiết lập NodeJS.

- Download Nodejs

Download phiên bản mới nhất của Node.js từ trang Node.js Downloads. Tại thời điểm bài hướng dẫn này, các phiên bản mới nhất tương ứng với các hệ điều hành như sau:

OS	Archive name
Windows	node-v0.12.0-x64.msi
Linux	node-v0.12.0-linux-x86.tar.gz
Mac	node-v0.12.0-darwin-x86.tar.gz
SunOS	node-v0.12.0-sunos-x86.tar.gz

- Cài đặt Node.js trên UNIX/Linux/Mac OS X, và SunOS

Dựa vào các kiến trúc của các hệ điều hành, tải và giải nén node-v0.12.0-osname.tar.gz trong thư mục /tm và sau cùng là giải nén vào thư mục /usr/local/nodejs. Ví dụ:

```
$ sudo apt-get install nodejs
```

Thêm /usr/local/nodejs/bin vào biến môi trường PATH

OS	Kết quả
Linux	export PATH=/usr/local/nodejs/bin
Mac	export PATH=\$PATH:/usr/local/nodejs/bin
FreeBSD	export PATH=\$PATH:/usr/local/nodejs/bin

- Cài đặt Node.js trên Windows

Sử dụng file MSI và theo các hướng dẫn hiện lên khi cài đặt Node.js. Mặc định, bộ cài đặt Node.js được lưu trữ tại C:\Program Files\nodejs .

- Xác nhận quá trình cài đặt Node.js có thành công không

Tạo một file js với tên main.js trên máy tính (Windows hoặc Linux) với dòng code dưới đây:

```
/* Hello, World! Vi du kiem tra cai dat node.js */ console.log("Hello, World!")
```

Bây giờ thực hiện main.js sử dụng trình biên dịch Node.js để xem các kết quả:

```
$ node main.js
```

Nếu mọi thứ thành công, sau khi biên dịch sẽ cho kết quả:

```
Hello, World!
```

1.2. Tầm quan trọng của JavaScript

1.2.1. Giới thiệu lại các khái niệm quan trọng của JS.

Phạm vi trong ngôn ngữ lập trình thể hiện mức độ truy cập và vòng đời của các biến hay tham số. Scope có thể được xác định trên globally hoặc locally. Scope là một trong những khái niệm quan trọng đối với developer vì nó sẽ giúp bạn tránh khỏi gặp bug khi đặt tên biến trùng và quản lý bộ nhớ, giá trị của biến hoặc tham số tại mỗi thời điểm hay ngữ cảnh(context) mà nó đang tham chiếu đến => viết code nhanh hơn dễ

bảo trì hơn. Vòng đời của một biến trong javascript: Vòng đời của một biến sẽ được bắt đầu khi nó được khai báo Biến local sẽ bị xóa đi khi kết thúc một function Biến global sẽ bị xóa đi khi đóng browser(hoặc tab), lưu ý là biến sẽ duy trì trong một trình duyệt web, các biến toàn cục sẽ bị xóa khi bạn đóng cửa sổ trình duyệt (hoặc tab), nhưng vẫn có sẵn cho các trang mới được tải vào cùng một cửa sổ.

Trong javascript ta có thể phân loại scope ra thành 3 loại chính như sau: global scope, local scope.

Global scope Biến được khai báo ở ngoài function => biến toàn cục Các biến toàn cục có thể được truy cập bởi tất cả các đoạn script trong webpage

Local scope Biến local chỉ có thể được truy cập trong các function mà nó được khai báo

Khi làm việc với các function asynchronous do đó chương trình sẽ nhảy bước, do đó để lấy kết quả của hàm ajax, ta phải truyền cho nó 1 callback. Sau khi hàm AJAX lấy được kết quả, nó sẽ gọi hàm callback với kết quả thu được => không có quy trình, khó khăn trong việc kiểm soát, thiếu tính chặt chẽ. Giả sử bài toán đặt ra ta cần gọi 2 API để lấy dữ liệu, và giao diện chỉ được sinh ra khi có đủ kết quả từ 2 API.

1.2.2. Tính cần thiết của ES6-7.

- Có 10 tính cần thiết cần phải nhắc đến:

- + Default Parameters in ES6
- + Template Literals in ES6
- + Multi-line String in ES6
- + Destructuring Assignment in ES6
- + Enhanced Object Literals in ES6
- + Arrow Function in ES6
- + Promises in ES6
- + Block-Scoped Constructs Let and Const
- + Classes in ES6
- + Modules in ES6

1.2.3. Phiên bản ECMAScript và NodeJS

- Thời điểm ECMAScript mới được phát hành, bản thân Netscape cũng tiếp tục hiện thực JavaScript và có phiên bản riêng. Cụ thể, phiên bản ES đầu tiên tương đương với JavaScript 1.3. ECMAScript 2 chỉ là phiên bản hiệu chỉnh để tương thích với tiêu chuẩn ISO/IEC 16262. ECMAScript 3 là phiên bản có một số cải tiến đáng lưu ý như RegularExpression, try/catch... và tương đương với JavaScript 1.5 của Netscape Navigator 6.0, Firefox 1.0 và JScript 5.5 của Internet Explorer 5.5.

ECMAScript 4 với đặc tả nổi bật là class, khai báo kiểu, generator, iterator, và E4X (ECMAScript for XML) đã có sự không đồng thuận giữa hai tổ chức có thị phần trình duyệt lớn nhất lúc bấy giờ là Microsoft và Netscape/Mozilla, dẫn đến việc phiên bản này bị loại bỏ. Tuy nhiên vẫn có một hiện thực khá nổi tiếng sử dụng đặc tả này, đó chính là ActionScript 3 dành cho Flash.

ECMAScript 5 là phiên bản thay thế ES4, là sự thỏa hiệp của Brendan Eich (lúc này đã là CTO của Mozilla, tổ chức phi lợi nhuận thành lập bởi Netscape và chịu trách nhiệm phát triển trình duyệt nguồn mở Firefox) với các bên phản đối ES4 (trong đó có Yahoo, Microsoft và Google). Những cải tiến trong ES5 không nhiều và vốn ban đầu được Microsoft đưa ra dưới phiên bản đề nghị là ES3.1. Một số thay đổi đáng chú ý trong ES5: strict mode, JSON, các phương thức của Object và Array, getter & setter... ES5 được chính thức phát hành vào tháng 12/2009, tương đương với JavaScript 1.8.5 (Firefox 4, và là phiên bản JavaScript độc lập cuối cùng) và chỉ được hỗ trợ hoàn toàn kể từ Internet Explorer 9. Giai đoạn này cũng đánh dấu sự xuất hiện của Node.js, nền tảng server side thành công nhất của JavaScript.

Phiên bản tiếp theo sau ECMAScript 5 mất một khoảng thời gian dài để hoàn tất. Lúc này, Firefox đã chiếm được thị phần đáng kể, IE vẫn chiếm đa số nhưng đang mất dần thị phần và Chrome là anh lính mới nhưng phát triển rất nhanh. JavaScript nhận được sự quan tâm lớn từ xu hướng phát triển web app nặng về phía Front End với cách tiếp cận AJAX. Tiếp theo sau đó là sự nở rộ của các JavaScript frameworks.

Các công ty nắm thị phần trình duyệt đa số đã đạt được sự đồng thuận cao nên dự án cho phiên bản ES tiếp theo được đặt tên là ECMAScript Harmony. ECMAScript

Harmony cũng đánh dấu quy trình làm việc mới của tiểu ban ECMA TC39 (Technical Committee 39) với các đặc tả mới được tiến hành hoàn thiện theo module và chuẩn hóa qua 5 bước (tương tự quy trình của W3C cho các module của HTML & CSS).

ECMAScript 6 (ECMAScript 2015), Các đặc tả của ES6 được khóa sổ vào tháng 6 2015 và phiên bản ES được đổi từ số thứ tự thành năm phát hành, ECMAScript 2015. Đây là phiên bản có rất nhiều cải tiến về cú pháp, khiến cho JavaScript gần như lột xác hoàn toàn.

Một số đặc tả mới đáng chú ý:

- class và public method
- block scope và từ khóa let, const
- arrow function
- template string
- generator và iterator
- Promise
- destructuring assignment
- module (còn gọi là ECMAScript module / esm)

1.2.4. Công Cụ hỗ trợ lập trình.

Khi học lập trình Nodejs thì các bạn có thể sử dụng IDE hoặc Editor. Nếu sử dụng IDE thì bạn có thể sử dụng IDE tốt nhất cho Node chính là Webstorm của JetBrains, tuy nhiên công cụ này là trả phí và giá khá là cao nên mình không khuyến khích sử dụng nếu bạn là sinh viên hoặc là người mới nghiên cứu về Node JS. Bạn có thể tải bản dùng thử 30 ngày của Webstorm nếu muốn thử trải nghiệm.

Cá nhân mình thì rất thích và đang sử dụng Sublime text. Ngoài ra còn có một công cụ khác miễn phí nữa chính là Visual Studio Code của Microsoft, đây cũng là công cụ rất được ưa thích. Và Editor cuối mình giới thiệu chính là Atom

CHƯƠNG 2: ỨNG DỤNG CỦA NODEJS

2.1. Chạy ứng dụng NodeJS với terminal

2.1.1. Viết code tạo server.

Tạo 1 file server.js ở đâu bạn thích, chúng ta sẽ để ở ngoài Desktop\Node với nội dung như sau:

```
1.  var http = require('http');
2.  http.createServer(function (req, res){
3.      res.writeHead(200, {'Content-Type': 'text/plain'});
4.      res.end();
5.  }).listen(8080, '127.0.0.1');
6.  console.log('Server running at http://127.0.0.1:8080/');
```

Bạn có thể dùng notepad để code, thường thì dùng **Notepad++** sẽ thích hơn.

Ngoài ra để lập trình với Node, bạn nên dùng các IDE mạnh hơn như **Sublime Text**, và đặc biệt gần đây là **Visual Studio Code**. Trong toàn bộ các bài viết sau này của mình, sẽ dùng trên **Visual Studio Code** nhé.

2.1.2. Chạy server NodeJS.

Mở Cmd command [Windows -> Run hoặc phím nóng Windows + R] (Đối với OSX, Linux, Unix, Ubuntu... thì là mở terminal)

Trở đến thư mục chứa file vừa tạo:

```
$cd [Đường dẫn đến thư mục chứa tệp tin server.js]
```

Sau đó nhập lệnh tạo Server bằng cú pháp :

```
$node server.js
```

Nếu thành công thì trên Cmd sẽ hiện lên dòng chữ "Server running at <http://127.0.0.1:8080/>"

Để kiểm chứng hãy vào :

- <http://localhost:8080/>

- <http://127.0.0.1:8080/>

2.2. Node package manager(npm) và NodeJS module system

2.2.1. Node package manager(npm)

Node package manager(npm) cung 2 chức năng chính:

Các kho lưu trữ trực tuyến cho node.js packages/modules có thể tìm kiếm trên search.nodejs.org

Tiện ích dòng lệnh để cài đặt các gói Node.js, quản lý phiên bản và quản lý phụ thuộc các gói Node.js

NPM đi kèm với các bản cài đặt Node.js sau phiên bản v0.6.3. Để xác minh tương tự, mở cmd và gõ lệnh sau và xem kết quả:

```
$ npm -version
```

Nếu bạn đang chạy một phiên bản NPM cũ thì việc cập nhật nó lên phiên bản mới nhất là khá dễ dàng. Chỉ cần sử dụng lệnh sau từ root:

```
$ sudo npm install npm -g
```

Cài đặt các modules bằng NPM

- Có một cú pháp đơn giản để cài đặt bất kỳ module Node.js nào:

```
$ npm install <Module Name>
```

- Ví dụ: dưới đây là lệnh để cài đặt mô-đun khung web Node.js nổi tiếng có tên express

```
$ npm install express
```

- Bây giờ bạn có thể sử dụng mô-đun này trong tệp js của bạn như sau:

```
var express = require('express');
```

Cài đặt toàn cục và cục bộ

- Theo mặc định, NPM cài đặt bất kỳ phụ thuộc nào trong chế độ cục bộ. Ở đây chế độ cục bộ đề cập đến việc cài đặt gói trong thư mục node_modules nằm trong thư mục có ứng dụng Node. Các gói được triển khai cục bộ có thể truy cập thông qua phương thức `request()`. Ví dụ, khi chúng ta cài đặt mô-đun express, nó đã tạo thư mục node_modules trong thư mục hiện tại nơi nó đã cài đặt mô-đun express.

- Ngoài ra, bạn có thể sử dụng lệnh `npm ls` để liệt kê tất cả các mô-đun được cài đặt cục bộ.

- Các gói / phụ thuộc được cài đặt toàn cầu được lưu trữ trong thư mục hệ thống. Các phụ thuộc như vậy có thể được sử dụng trong hàm CLI (Giao diện dòng lệnh) của bất kỳ node.js nào nhưng không thể được nhập trực tiếp bằng cách sử dụng `require()` trong ứng dụng Node. Bây giờ hãy thử cài đặt mô-đun `express` bằng cách cài đặt toàn cục.

```
$ npm install express -g
```

- Điều này sẽ tạo ra một kết quả tương tự nhưng mô-đun sẽ được cài đặt trên toàn cục. Ở đây, dòng đầu tiên hiển thị phiên bản mô-đun và vị trí nơi nó được cài đặt.

- Bạn có thể sử dụng lệnh sau để kiểm tra tất cả các mô-đun được cài đặt trên toàn cục.

```
$ npm ls -g
```

Sử dụng package.json

- `package.json` có trong thư mục gốc của bất kỳ ứng dụng / mô-đun Node nào và được sử dụng để xác định các thuộc tính của gói. Hãy mở `package.json` của `express` hiện tại trong thư mục “`node_modules/express/`”

Gỡ cài đặt Mô-đun

- Sử dụng lệnh sau để gỡ cài đặt mô-đun `Node.js`.

```
$ npm uninstall express
```

- Khi NPM gỡ cài đặt gói, bạn có thể xác minh gói đó bằng cách xem nội dung của thư mục `/node_modules/` hoặc nhập lệnh sau.

```
$ npm ls
```

Cập nhật một mô-đun.

- Cập nhật `package.json` và thay đổi phiên bản của phụ thuộc sẽ được cập nhật và chạy lệnh sau.

```
$ npm update express
```

Tìm kiếm một mô-đun

- Tìm kiếm tên gói bằng NPM

\$ npm search express

Tạo một mô-đun

- Tạo một mô-đun yêu cầu gói.json được tạo. Hãy tạo **package.json** bằng NPM, sẽ tạo khung cơ bản của **package.json**.

\$ npm init

- Bạn sẽ cần cung cấp tất cả các thông tin cần thiết về mô-đun của bạn. Bạn có thể nhận trợ giúp từ tệp **package.json** đã đề cập ở trên để hiểu ý nghĩa của các thông tin khác nhau được yêu cầu. Khi **package.json** được tạo, hãy sử dụng lệnh sau để tự đăng ký với trang lưu trữ NPM bằng địa chỉ email hợp lệ.

\$ npm adduser

- Bây giờ là lúc để xuất bản mô-đun của bạn

\$ npm publish

- Nếu mọi thứ đều ổn với mô-đun của bạn, thì nó sẽ được xuất bản trong kho lưu trữ và có thể truy cập để cài đặt bằng NPM như bất kỳ mô-đun Node.js nào khác.

2.2.2. NodeJS module system.

Các module trong NodeJS giống như các thư viện trong JavaScript. Nó là một tập hợp các chức năng bạn muốn đưa vào ứng dụng của bạn.

Tích hợp modules

- **Node.js** có một bộ các mô-đun tích hợp mà bạn có thể sử dụng mà không cần cài đặt thêm.

Include Modules

- Để include Modules, ta sử dụng hàm **require()** với tên của modules đó, ví dụ:

1. **var** http = require('http');

- Bây giờ ứng dụng của bạn có quyền truy cập vào mô-đun **HTTP** và có thể tạo máy chủ:

```
1. http.createServer(function (req, res) {  
2.   res.writeHead(200, {'Content-Type': 'text/html'});  
3.   res.end('Hello World!');  
4. }).listen(8080);
```

Tạo Modules của riêng bạn

- Bạn có thể tạo các mô-đun của riêng bạn và dễ dàng đưa chúng vào các ứng dụng của bạn.

- Ví dụ sau đây tạo ra một mô-đun trả về một đối tượng ngày và thời gian:

```
1. exports.myDateTime = function () {  
2.   return Date();  
3. };
```

- Sử dụng từ khóa **exports** để làm cho các thuộc tính và phương thức có sẵn bên ngoài tệp mô-đun.

- Lưu mã ở trên trong một tệp có tên "**myfirstmodule.js**"

Include Modules của riêng bạn

- Bây giờ bạn có thể include và sử dụng mô-đun trong bất kỳ tệp Node.js nào của mình.

- ví dụ:

```
1. var http = require('http');  
2. var dt = require('./myfirstmodule');  
3.  
4. http.createServer(function (req, res) {  
5.   res.writeHead(200, {'Content-Type': 'text/html'});  
6.   res.write("The date and time are currently: " + dt.myDateTime());  
7.   res.end();  
8. }).listen(8080);
```

- Lưu ý rằng chúng tôi sử dụng **./** để định vị mô-đun, điều đó có nghĩa là mô-đun nằm trong cùng thư mục với tệp **Node.js**.

2.3. NodeJS core packages.

Node.js là một khung trọng lượng nhẹ. Các mô-đun cốt lõi bao gồm các chức năng tối thiểu của Node.js. Các mô-đun lõi này được biên dịch vào phân phối nhị phân của nó và tự động tải khi quá trình Node.js bắt đầu. Tuy nhiên, trước tiên bạn cần nhập mô-đun lõi để sử dụng nó trong ứng dụng của mình.

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods, and events to work with file I/O.
util	util module includes utility functions useful for programmers.

Bảng 1: Một số mô-đun cốt lõi quan trọng trong Node.js.

Loading các mô-đun lõi

- Để sử dụng các mô-đun NPM hoặc lõi Node.js, trước tiên bạn cần nhập nó bằng hàm `require()` như hiển thị bên dưới.

1. `var module = require('module_name');`

- Theo cú pháp trên, chỉ định tên mô-đun trong hàm `require()`. Hàm Yêu cầu sẽ trả về một đối tượng, hàm, thuộc tính hoặc bất kỳ loại JavaScript nào khác, tùy thuộc vào những gì mô-đun được chỉ định trả về.

- Ví dụ sau minh họa cách sử dụng mô-đun `http` của Node.js để tạo máy chủ web.


```
1. var http = require('http');
2. var server = http.createServer(function(req, res){
3.   //write code here
4. });
5. server.listen(5000);
```

- Trong ví dụ trên, hàm request () trả về một đối tượng vì mô-đun http trả về chức năng của nó dưới dạng đối tượng, sau đó bạn có thể sử dụng các thuộc tính và phương thức của nó bằng cách sử dụng ký hiệu dấu chấm, ví dụ http.createServer ().

- Theo cách này, bạn có thể tải và sử dụng các mô-đun lõi Node.js trong ứng dụng của mình. Chúng tôi sẽ sử dụng các mô-đun cốt lõi trong quá trình làm đồ án.

2.4. Xử lý yêu cầu API cho ứng dụng NodeJS.

Sử dụng phương thức HTTP và các API Route.

- Hãy tưởng tượng rằng bạn đang xây dựng một RESTful API Node.js để tạo, cập nhật, gọi thông tin hay xóa người dùng. Với các tính năng đó HTTP đã cung cấp một bộ đầy đủ các phương thức: POST, PUT, GET, PATCH và DELETE.

- Cách tối ưu nhất là các API route của bạn chỉ nên sử dụng danh từ như là các định danh tài nguyên. Các route khi đó sẽ trông như thế này:

POST /user hay PUT /user/:id để tạo người dùng mới.

GET /user để lấy danh sách người dùng.

GET /user/:id để lấy thông tin của một người dùng.

PATCH /user/:id để sửa một bản ghi người dùng đã có.

DELETE /user/:id để xóa một người dùng.

2.5. Xử lý file trong ứng dụng NodeJS.

2.5.1 Mở một File trong Node.js

- Để mở một file trong chế độ không đồng bộ, bạn sử dụng phương thức `open()` có cú pháp:

```
1. fs.open(path, flags[, mode], callback)
```

- Tham số

`path` - Đây là một chuỗi biểu diễn tên file cũng như đường dẫn tới file đó.

flags - Biểu diễn hành vi của file được mở. Tất cả các giá trị có thể sẽ được trình bày trong bảng dưới đây.

mode - Thiết lập chế độ cho file, các chế độ này chỉ được thiết lập khi file đã được tạo. Giá trị mặc định là 0666, tức là readable và writeable.

callback - Hàm callback nhận hai tham số, ví dụ (err, fd).

Các Flag được sử dụng cho hoạt động Đọc/Ghi file trong Node.js

lag	Miêu tả
	Mở file để đọc. Xuất hiện Exception nếu file không tồn tại.
+	Mở file để đọc và ghi. Xuất hiện Exception nếu file không tồn tại.
s	Mở file để đọc trong chế độ đồng bộ.
s+	Mở file để đọc và ghi, báo cho Hệ điều hành mở nó trong chế độ đồng bộ.
	Mở file để ghi. Nếu file không tồn tại, nó sẽ tạo file mới.
x	Giống 'w' nhưng hoạt động này thất bại nếu file không tồn tại (tức là nó không tạo file mới).
+	Mở file để đọc và ghi. Nếu file không tồn tại, nó sẽ tạo file mới.
x+	Giống 'w+' nhưng hoạt động này thất bại nếu file không tồn tại
	Mở file để append. File sẽ được tạo nếu nó không tồn tại.

x	Giống 'a' nhưng hoạt động này thất bại nếu file không tồn tại.
+	Mở file để đọc và append. File sẽ được tạo nếu nó không tồn tại.
x+	Giống 'a+' nhưng hoạt động này thất bại nếu file không tồn tại.

Bảng 2: Các Flag được sử dụng cho hoạt động Đọc/Ghi file trong Node.js

Ví dụ:

- Ví dụ sau minh họa cách mở một file để đọc và ghi. Đầu tiên bạn tạo main.js có nội dung như dưới đây. Nội dung file khá giống ví dụ trên, bạn chú ý vào phần flag đã sử dụng ở đây.

```

1. var fs = require("fs");
2. // Hoat dong mo File theo cach thuc khong dong bo
3. console.log("Chuan bi mo File hien tai!");
4. fs.open('input.txt', 'r+', function(err, fd) {
5.     if (err) {
6.         return console.error(err);
7.     }
8.     console.log("File duoc mo thanh cong!");
9. });

```

- Chạy main.js để xem kết quả:

```
$ node main.js
```

- Kiểm tra kết quả:

```

1. Chuan bi mo File hien tai!
2. File duoc mo thanh cong!

```

2.5.2 Lấy thông tin File trong Node.js

- Để lấy thông tin về một file trong Node.js, bạn sử dụng phương thức stat() của fs Module có cú pháp:

```
fs.stat(path, callback)
```

- Trong đó:

path - Đây là một chuỗi biểu diễn tên file cũng như đường dẫn tới file đó.

callback - Là hàm callback nhận hai tham số (err, stats), trong đó stats là một đối tượng của fs.Stats được in ra như trong ví dụ sau.

- Ngoài các thuộc tính quan trọng được in ra như trong ví dụ sau, lớp fs.Stats còn có một số phương thức hữu ích có thể được sử dụng để kiểm tra kiểu file. Đó là:

Phương thức	Miêu tả
stats.isFile()	Trả về true nếu đó là một file
stats.isDirectory()	Trả về true nếu đó là một thư mục
stats.isBlockDevice()	Trả về true nếu đó là một Block Device.
stats.isCharacterDevice()	Trả về true nếu đó là một Character Device.
stats.isSymbolicLink()	Trả về true nếu đó là một Symbolic Link.
stats.isFIFO()	Trả về true nếu đó là một kiểu FIFO.
stats.isSocket()	Trả về true nếu đó là một kiểu Socket.

Bảng 3: Một số phương thức hữu ích có thể được sử dụng để kiểm tra kiểu file

2.5.3 Ghi dữ liệu vào File trong Node.js

- Để ghi dữ liệu vào File trong Node.js, bạn có thể sử dụng phương thức writeFile() của fs Module như sau:

fs.writeFile(filename, data[, options], callback)

- Phương thức này sẽ ghi đè nếu file đã tồn tại.

- Trong đó:

path - Đây là một chuỗi biểu diễn tên file cũng như đường dẫn tới file đó.

data - Dữ liệu dạng String hoặc Buffer để ghi vào File.

options - Tham số này là một đối tượng giữ {encoding, mode, flag}. Theo mặc định, mã hóa là utf8, mode là giá trị 0666 và flag là 'w'

callback - Hàm callback nhận một tham số là err và được sử dụng để trả về một lỗi nếu xảy ra bất kỳ lỗi nào trong hoạt động ghi

2.5.4 Đọc dữ liệu từ File trong Node.js

- Để đọc dữ liệu từ một File, bạn sử dụng phương thức read() có cú pháp sau:

fs.read(fd, buffer, offset, length, position, callback)

- Phương thức này sẽ sử dụng tham số fd (viết tắt của File Descriptor) để đọc file. Nếu bạn muốn đọc file bởi sử dụng trực tiếp tên file thì bạn nên sử dụng phương thức khác.

- Trong đó:

fd - Là viết tắt của file descriptor được trả về bởi phương thức fs.open().

buffer - Đây là Buffer, là nơi dữ liệu được ghi vào.

offset - Đây là offset trong Buffer để dữ liệu bắt đầu ghi từ vị trí đó.

length - Một số nguyên xác định số byte để đọc.

position - Một số nguyên xác định nơi bắt đầu đọc từ trong file. Nếu vị trí là null, dữ liệu sẽ được đọc từ vị trí hiện tại của file.

callback - Một hàm callback nhận ba tham số, có dạng (err, bytesRead, buffer).

2.5.5 Đóng File trong Node.js

- Để đóng một file sau khi đã mở, bạn sử dụng phương thức close() có cú pháp:

fs.close(fd, callback)

- Trong đó:

fd - Là viết tắt của file descriptor được trả về bởi phương thức fs.open().

callback - Hàm callback nhận một tham số để xử lý trường hợp nếu có exception.

2.5.6 Truncate một File trong Node.js

- Để truncate một file đã mở, bạn sử dụng phương thức ftruncate() có cú pháp:

`fs.ftruncate(fd, len, callback)`

- Trong đó:

`fd` - Là viết tắt của file descriptor được trả về bởi phương thức `fs.open()`.

`len` - Là độ dài của file sau khi đã được truncate.

`callback` - Hàm callback nhận một tham số để xử lý trường hợp nếu có exception

2.5.7. Xóa File trong Node.js

- Để xóa một file trong Node.js, bạn sử dụng phương thức `unlink()` có cú pháp:

`fs.unlink(path, callback)`

- Trong đó:

`path` - Là tên file hoặc tên đường dẫn trỏ đến file.

`callback` - Hàm callback nhận một tham số để xử lý trường hợp nếu có exception.

2.5.8 Tạo thư mục trong Node.js

- Để tạo một thư mục trong Node.js, bạn sử dụng phương thức `mkdir()` có cú pháp:

`fs.mkdir(path[, mode], callback)`

- Trong đó:

`path` - Là tên thư mục bao gồm đường dẫn trỏ tới thư mục đó.

`mode` - Chế độ xác định các quyền cho phép khi truy cập thư mục. Giá trị mặc định là 0777.

`callback` - Hàm callback nhận một tham số để xử lý trường hợp nếu có exception.

2.5.9 Đọc thư mục trong Node.js

- Để đọc thư mục trong Node.js, bạn sử dụng phương thức `readdir()` có cú pháp:

`fs.readdir(path, callback)`

- Trong đó:

`path` - Là tên thư mục bao gồm đường dẫn trỏ tới thư mục đó.

`callback` - Hàm callback nhận hai tham số, dạng (err, files) trong đó files là một mảng chứa các tên file trong thư mục.

2.5.10 Xóa thư mục trong Node.js

- Để xóa một thư mục trong Node.js, bạn sử dụng phương thức `rmdir()` có cú pháp:

`fs.rmdir(path, callback)`

- Trong đó:

`path` - Là tên thư mục bao gồm đường dẫn tới thư mục đó.

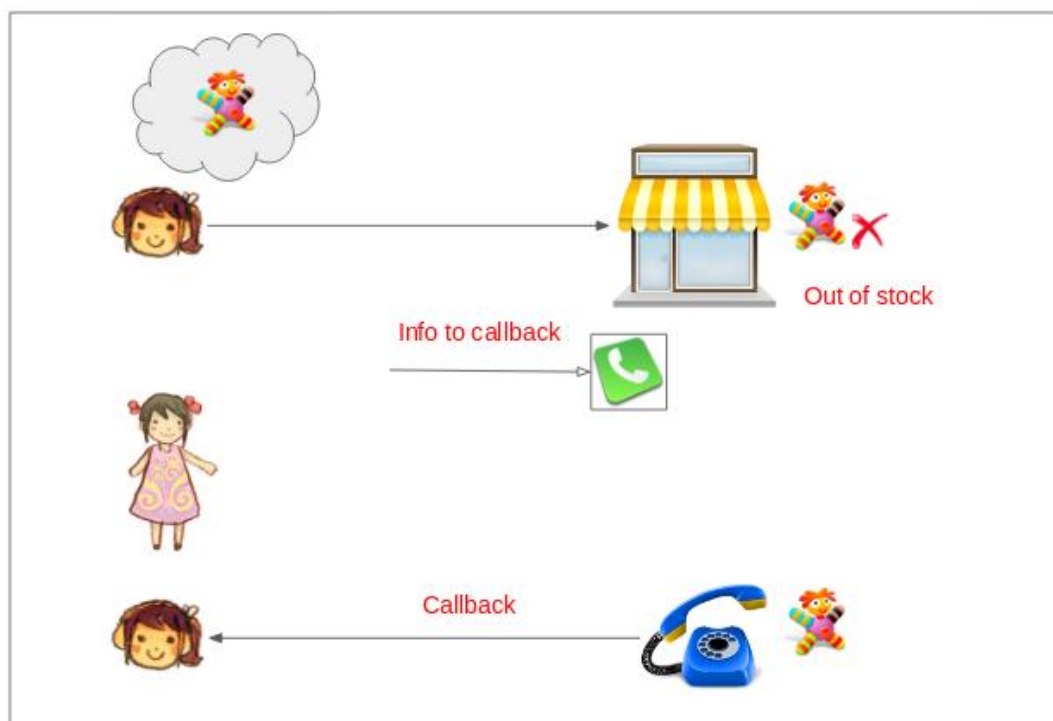
`callback` - Hàm callback nhận một tham số để xử lý trường hợp nếu có exception.

2.6. Khái niệm về Callback

2.6.1. Callback là gì?

Để giải thích Callback là gì chúng ta hãy xem một tình huống như sau:

Bạn tới một cửa hàng để mua một món đồ mà bạn yêu thích, nhân viên cửa hàng nói với bạn rằng hiện tại món đồ đó đã hết, bạn để lại số điện thoại và yêu cầu họ gọi lại ngay sau khi có hàng. Sau đó bạn có thể đi chơi hoặc làm một công việc nào đó và không cần quan tâm tới cửa hàng đó nữa, cho tới khi bạn nhận được điện thoại thông báo của hàng đã có món đồ mà bạn yêu thích.



Hình 1: ví dụ callback

Máy chủ NodeJS có thể nhận rất nhiều các 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. "callback" là một hàm (function), nó sẽ được gọi khi NodeJs hoàn thành một tác vụ (task) cụ thể.

2.6.2. Ví dụ Node JS Callback

Trong NodeJS các API được thiết kế để hỗ trợ Callback. Giả sử rằng bạn đang viết một chương trình để đọc 2 tập tin. Để làm việc này bạn sử dụng module **fs**, nó cung cấp cho bạn 2 hàm để đọc file là `readFile` và `readFileSync`. Chúng ta sẽ tìm hiểu sự khác biệt giữa 2 hàm này.

Blocking

`readFileSync` là một hàm đọc file một cách đồng bộ (synchronous), chính vì vậy trong khi hàm này đang thực thi nó sẽ chặn (block) chương trình thực thi các dòng code tiếp theo.

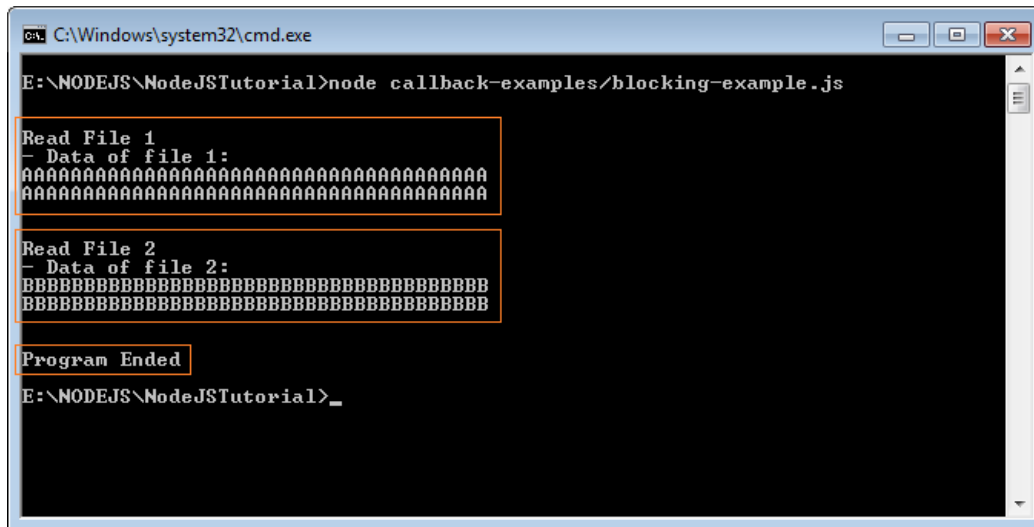
blocking-example.js

```
1. var fs = require("fs");
2.
3. // -----> Read file 1:
4. console.log("\n");
5. console.log("Read File 1");
6.
7. var data1 = fs.readFileSync('C:\\test\\file1.txt');
8.
9. console.log("- Data of file 1: ");
10. console.log(data1.toString());
11.
12.
13. // -----> Read file 2:
14. console.log("\n");
15. console.log("Read File 2");
16.
17. var data2 = fs.readFileSync('C:\\test\\file2.txt');
18. console.log("- Data of file 2: ");
19. console.log(data2.toString());
20.
21.
22. console.log("\n");
```



```
23. console.log("Program Ended");
```

Thực thi tập tin blocking-example.js và đây là kết quả mà bạn nhận được:



```
C:\Windows\system32\cmd.exe
E:\NODEJS\NodeJSTutorial>node callback-examples/blocking-example.js

Read File 1
- Data of file 1:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Read File 2
- Data of file 2:
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

Program Ended
E:\NODEJS\NodeJSTutorial>_
```

Hình 2: kết quả chạy tập tin blocking-example.js

Non Blocking:

Nên sử dụng hàm **readFile** để đạt được hiệu suất tốt hơn cho chương trình. Hàm này đọc file một cách "không đồng bộ" (asynchronous), nó "không chặn" (non block) chương trình thực thi các dòng code tiếp theo, nói cách khác là chương trình không đợi hàm này hoàn thành. Nhưng khi hàm này thực hiện xong nhiệm vụ của nó, nó sẽ gọi tới hàm Callback.

non-blocking-example.js

```
1. var fs = require("fs");
2.
3. // A Callback function!
4. function read Finished File1(err, data) {
5.   if (err) console.log(err);
6.   console.log("- Data of file 1: ");
7.   console.log(data.toString());
8. }
9.
```

```

10. // A Callback function!
11. function readFinishedFile2(err, data) {
12.   if (err) console.log(err);
13.   console.log("- Data of file 2: ");
14.   console.log(data.toString());
15. }
16.
17. // -----> Read file 1:
18. console.log("\n");
19. console.log("Read File 1");
20.
21. fs.readFile('C:\\test\\file1.txt', readFinishedFile1);
22.
23. // -----> Read file 2:
24. console.log("\n");
25. console.log("Read File 2");
26.
27. fs.readFile('C:\\test\\file2.txt', readFinishedFile2);
28.
29. console.log("\n");
30. console.log("Program Ended \n");

```

Chạy tập tin non-blocking-example.js và đây là kết quả mà bạn nhận được:

```

C:\Windows\system32\cmd.exe
E:\NODEJS\NodeJSTutorial>node callback-examples/non-blocking-example.js

Read File 1
Read File 2
Program Ended
- Data of file 1:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
- Data of file 2:
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
E:\NODEJS\NodeJSTutorial>_

```

Hình 3: kết quả chạy tập tin non-blocking-example.js

CHƯƠNG 3: TEMPLATE ENGINE HANDLEBARS

3.1 Giới thiệu

Handlebars.js là một công cụ phía máy khách (mặc dù nó cũng có thể được sử dụng trên máy chủ) công cụ tạo khuôn template cho JavaScript

Handlebars.js được viết bằng JavaScript, Handlebars.js là trình biên dịch nhận bất kỳ biểu thức HTML và biên dịch chúng thành hàm JavaScript, Hàm JavaScript này nhận một tham số, một đối tượng, dữ liệu của bạn, và nó trả về một chuỗi có các giá trị HTML và thuộc tính của đối tượng được chèn vào HTML.

3.2 Khi nào nên sử dụng handlebars.js và tại sao nên sử dụng nó

Khi nào nên sử dụng công cụ tạo template JavaScript?

Chúng ta nên sử dụng một JavaScript templating engine handlebars.js khi:

Bạn sử dụng một javascript front-end framework như: backbone.js, ember.js, jquery,.... Hầu hết các framework javascript đều dựa vào templating engines

View trong ứng dụng của bạn sẽ được cập nhật thường xuyên, đặc biệt là kết quả của các thay đổi đối với dữ liệu từ server thông qua API REST hoặc từ dữ liệu trên máy khách

Ứng dụng của bạn có nhiều tương tác

Bạn đang phát triển một ứng dụng web có nhiều lượt truy cập

Bạn muốn quản lý dễ dàng nội dung của mình

Tại sao nên sử dụng handlebars.js

Nó là một trong những templating engines được sử dụng phổ biến nhất

Nó là một phần mở rộng của ngôn ngữ Mustache Javascript templating, nó thay thế Mustache.js Những lý do bạn nên sử dụng Handlebars.js như sau:

Handlebars là một trong những công cụ tiên tiến nhất, giàu tính năng và phổ biến trong tất cả Javascript templating engines.

Handlebars là templating engine không có logic, có nghĩa là có rất ít hoặc không có logic trong các template HTML mà bạn sử dụng.

Nó giữ cho các trang HTML của bạn đơn giản, sạch sẽ và sẽ tách biệt khỏi các tệp javascript dựa trên logic và handlebars phục vụ tốt mục đích này Tóm lại, học

Handlebars.js ngày từ bây giờ là một sự đầu tư và là một sự lựa chọn sáng suốt giúp bạn lập trình hiệu quả hơn bây giờ và bạn sẽ dễ dàng thích nghi với các JS frameworks

3.3. Handlebars hoạt động như thế nào

Như đã nói trong phần giới thiệu: Handlebars.js là trình biên dịch được xây dựng bằng javascript, lấy bất kỳ biểu thức HTML và handlebars nào và biên dịch chúng thành hàm javascript. Hàm javascript này sau đó nhận một tham số, một đối tượng, dữ liệu của bạn và nó trả về một chuỗi HTML với các giá trị của thuộc tính đối tượng được chèn vào HTML.

3 Bộ phận chính của Handlebars templating:

Biểu thức của Handlebars.js:

Một biểu thức handlebars đơn giản được viết như thế này (trong đó, nội dung của content có thể là một biến hoặc một hàm trợ giúp với các thông số hoặc không có tham số)

1. `{{ content }}`

hoặc như thế này nếu biểu thức nằm trong khối:

1. `{{#each}}`

2. HTML content and other Handlebars expression go here.

3. `{{/each}}`

Với biểu thức handlebars HTML. Biến `customerName` là thuộc tính sẽ được truy xuất bởi hàm `handlebars.compile`:

1. `Name: {{ customerName }}`

Dưới đây là ví dụ về một thẻ script Handlebars:

`<div> Name: {{ headerTitle }} </div>`

Biểu thức của Handlebars.js

Phần chính thứ 2 trong Handlebars templating là dữ liệu bạn muốn hiển thị trên trang. Bạn chuyển một dữ liệu đối tượng tới hàm handlebars, đối tượng dữ liệu được gọi là bối cảnh. Và đối tượng này có thể bao gồm tất cả các mảng, chuỗi, số, các đối tượng khác hoặc kết hợp tất cả đối tượng này. Nếu đối tượng dữ liệu có một mảng các đối tượng, bạn có thể sử dụng hàm Handlebars mỗi vòng lặp để lặp lại mảng và bối cảnh hiện tại được đặt cho từng mục trong mảng. Dưới đây là các ví dụ về việc thiết lập đối tượng dữ liệu và cách lặp nó trong handlebars:

- Đối tượng dữ liệu với mảng đối tượng:

```
1. //đối tượng customers có một mảng các đối tượng mà chúng ta sẽ chuyển tới han  
   dlebars:  
2. var theData = {customers:[{firstName:"Michael", lastName:"Alexander", age:2  
   0}, {firstName:"John", lastName:"Allen", age:29}]};
```

- Bạn có thể sử dụng vòng lặp để lặp lại đối tượng của customers như thế này:

```
1. {{#each customers}}  
2.   <li> {{ firstName }} {{ lastName }} </li>  
3. {{/each}}
```

hoặc nếu bạn chuyển đối tượng customers dưới dạng một mảng các đối tượng, chúng ta có thể sử dụng câu lệnh khối như thế này:

```
1. {{#customers}}  
2.   <li> {{ firstName }} {{ lastName }} </li>  
3. {{/customers}}
```

đối tượng dữ liệu kiểu string:

```
1. var theData = {headerTitle:"Shop Page", weekDay:"Wednesday"};
```

Hàm biên dịch: Đoạn mã cuối cùng mà chúng ta cần cho việc tạo template của handlebars xử lý 2 bước sau:

1. Biên dịch template với hàm handlebars.

2. Sau đó, sử dụng hàm được biên dịch đó để gọi đối tượng dữ liệu được truyền cho nó (nó lấy một đối tượng dữ liệu làm tham số duy nhất của nó). Và điều này sẽ trả về một chuỗi HTML với các giá trị đối tượng được chèn vào HTML.

Tóm lại: Hàm `Handlebars.compile` lấy mẫu làm tham số và nó trả về hàm JavaScript. Sau đó, chúng tôi sử dụng hàm được biên dịch này để thực thi đối tượng dữ liệu và trả về một chuỗi có HTML và các giá trị đối tượng được thêm vào. Sau đó chúng ta có thể chèn chuỗi vào trang HTML.

CHƯƠNG 4: EXPRESS.JS VÀ API

4.1 Tạo server HTTP đơn giản với NodeJS

Tạo 1 file `server.js` ở đâu bạn thích, chúng ta sẽ để ở ngoài `Desktop\Node` với nội dung như sau:

```
1. var http = require('http');
2. http.createServer(function (req, res){
3.   res.writeHead(200, {'Content-Type': 'text/plain'});
4.   res.end('http://yournodejs.blogspot.com/');
5. }).listen(8080, '127.0.0.1');
6. console.log('Server running at http://127.0.0.1:8080/');
```

Vào CMD và trở đến thư mục chứa file vừa tạo:

`$cd [Đường dẫn đến thư mục chứa tệp tin server.js]`

Sau đó nhập lệnh tạo Server bằng cú pháp :

`$node server.js`

Nếu thành công thì trên Cmd sẽ hiện lên dòng chữ "`Server running at http://127.0.0.1:8080/`"

Để kiểm chứng hãy vào :

- `http://localhost:8080/`

- `http://127.0.0.1:8080/`

4.2 Giới thiệu về Express.

Express là một framework nhỏ và tiện ích để xây dựng các ứng dụng web, cung cấp một lượng lớn của tính năng mạnh mẽ để phát triển các ứng dụng web và mobile.

Nó rất dễ dàng để phát triển các ứng dụng nhanh dựa trên Node.js cho các ứng dụng Web. Dưới đây là các tính năng cơ bản của Express framework.

Cho phép thiết lập các lớp trung gian để trả về các HTTP request.

Định nghĩa bảng routing có thể được 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 truyền vào đến template.

Về Cài đặt Express Framework.

- Đầu tiên, cài đặt Express framework sử dụng npm như sau:

```
$ npm install express --save
```

- Lệnh trên lưu phần cài đặt trong thư mục node_modules và tạo thư mục express bên trong thư mục đó. Dưới đây là các thành phần module quan trọng được cài đặt cùng với express:

body-parser - Đây là một lớp trung gian node.js để xử lý JSON, dữ liệu thô, text và mã hóa URL.

cookie-parser - Chuyển đổi header của Cookie và phân bổ đến các req.cookies

multer - Đây là một thành phần trung gian trong node.js để xử lý phần multipart/form-data.

4.3 Render phản hồi HTML bằng chuỗi mẫu.

Tạo 1 file **app.js** ở đâu bạn thích, chúng ta sẽ để ở ngoài Desktop\Node với nội dung như sau:

```
1. var http = require('http');
2. http.createServer(function (req, res){
3.   res.writeHead(200, {'Content-Type': 'text/plain'});
4.   res.end('http://yournodejs.blogspot.com/');
5. }).listen(8080, '127.0.0.1');
6. console.log('Server running at http://127.0.0.1:8080/');
```

Sau đó nhập lệnh chạy bằng cú pháp :

```
$ node app.js
```

Sau khi nhập lệnh xong chúng ta mở trình duyệt kiểm tra với url:

```
Localhost:8080
```

4.4 Tạo các điểm cuối API restful.

Giả sử chúng ta có một cơ sở dữ liệu dựa trên JSON chứa thông tin về User, tên file là users.json

```
1. {  
2.   "user1" : {  
3.     "name" : "huong",  
4.     "password" : "password1",  
5.     "profession" : "sinhvien",  
6.     "id": 1  
7.   },  
8.   "user2" : {  
9.     "name" : "manh",  
10.    "password" : "password2",  
11.    "profession" : "giangvien",  
12.    "id": 2  
13.  },  
14.  "user3" : {  
15.    "name" : "tuyen",  
16.    "password" : "password3",  
17.    "profession" : "laptrinhvien",  
18.    "id": 3  
19.  }  
20. }
```


Dựa vào các thông tin cơ bản này, chúng ta sẽ cung cấp các RESTful API sau đây:

Stt	URI	Phương thức HTTP	POST body	Kết quả
1	listUsers	GET	empty	Hiển thị danh sách user
2	addUser	POST	JSON String	Thêm một user mới
3	deleteUser	DELETE	JSON String	Xóa một user hiện tại.
4	:id	GET	empty	Hiển thị chi tiết một user

Bảng 4: Các tiêu chuẩn để thiết kế API

Liệt kê các User

- Chúng ta cùng triển khai RESTful API đầu tiên có tên listUsers bởi sử dụng đoạn code sau đây:

```

1. var express = require('express');
2. var app = express();
3. var fs = require("fs");
4. app.get('/listUsers', function (req, res) {
5.   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
6.     console.log( data );
7.     res.end( data );
8.   });
9. })
10. var server = app.listen(8081, function () {
11.   var host = server.address().address
12.   var port = server.address().port
13.   console.log("Ung dung Node.js đang lang nghe tai dia chi: http://%s:%s", host,
    port)

```

```
14. ))
```

- Bây giờ thử truy cập API đã được định nghĩa trên bởi sử dụng

<http://127.0.0.1:8081/listUsers> trên máy tính local. Nó sẽ cho ra kết quả sau đây:

```
1. {  
2.   "user1" : {  
3.     "name" : "huong",  
4.     "password" : "password1",  
5.     "profession" : "sinhvien",  
6.     "id": 1  
7.   },  
8.   "user2" : {  
9.     "name" : "manh",  
10.    "password" : "password2",  
11.    "profession" : "giangvien",  
12.    "id": 2  
13.  },  
14.  "user3" : {  
15.    "name" : "tuyen",  
16.    "password" : "password3",  
17.    "profession" : "laptrinhvien",  
18.    "id": 3  
19.  }  
20. }
```

Thêm User mới:

- API sau chỉ ra cách thêm một User mới vào danh sách. Dưới đây là thông tin của User mới:

```
1. user = {  
2.   "user4" : {  
3.     "name" : "hoang",  
4.     "password" : "password4",  
5.     "profession" : "sinhvien",  
6.     "id": 4  
7.   }  
8. }
```

- Bạn có thể sử dụng Ajax để thực hiện việc này, nhưng để đơn giản chúng ta sẽ hard code ở đây. Dưới đây là phương thức addUser API để thêm một user mới trong cơ sở dữ liệu.

```
1. var express = require('express');
2. var app = express();
3. var fs = require("fs");
4. var user = {
5.   "user4" : {
6.     "name" : "hoang",
7.     "password" : "password4",
8.     "profession" : "sinhvien",
9.     "id": 4
10.  }
11. }
12. app.get('/addUser', function (req, res) {
13.   // Đầu tiên, đọc tất cả các User đang tồn tại.
14.   fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
15.     data = JSON.parse( data );
16.     data["user4"] = user["user4"];
17.     console.log( data );
18.     res.end( JSON.stringify(data));
19.   });
20. })
21. var server = app.listen(8081, function () {
22.   var host = server.address().address
23.   var port = server.address().port
24.   console.log("Ung dụng Node.js đang lắng nghe tại địa chỉ: http://%s:%s", host, port)
25.
26. })
```

- Bây giờ thử truy cập API trên bởi sử dụng <http://127.0.0.1:8081/addUsers> trên máy tính local. Kết quả sẽ được hiện ra như sau:

```
1. { user1: { name: 'huong',
2.   password: 'password1',
3.   profession: 'sinhvien',
4.   id: 1
5. },
6. user2: { name: 'manh',
```

```

7.     password: 'password2',
8.     profession: 'giangvien',
9.     id: 2
10.  },
11. user3: { name: 'tuyen',
12.     password: 'password3',
13.     profession: 'laptrinhvien',
14.     id: 3
15.  },
16. user4: { name: 'hoang',
17.     password: 'password4',
18.     profession: 'sinhvien',
19.     id: 4
20.  }
21. }

```

Hiển thị thông tin của User

- Bây giờ cùng triển khai một API mà gọi đến userID để hiển thị chi tiết thông tin User tương ứng.

```

1.  var express = require('express');
2.  var app = express();
3.  var fs = require("fs");
4.
5.  app.get('/:id', function (req, res) {
6.    // Đầu tiên, đọc tất cả các User đang tồn tại.
7.    fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
8.      users = JSON.parse( data );
9.      var user = users["user" + req.params.id]
10.     console.log( user );
11.     res.end( JSON.stringify(user));
12.   });
13. })
14.
15. var server = app.listen(8081, function () {
16.
17.   var host = server.address().address
18.   var port = server.address().port
19.   console.log("Ứng dụng Node.js đang lắng nghe tại địa chỉ: http://%s:%s", host,
    port)
20.
21. })

```

- Tiếp đó, bạn gọi service trên bởi sử dụng địa chỉ <http://127.0.0.1:8081/2> trên máy tính local. Kết quả sẽ như sau:

```
1. {  
2.   "name":"manh",  
3.   "password":"password2",  
4.   "profession":"giangvien",  
5.   "id":2  
6. }
```

Xóa User

- API này tương tự như addUser API, tại đây bạn có thể nhận một dữ liệu đầu vào thông qua req.body và sau đó dựa vào userID để xóa User đó khỏi Database. Để đơn giản, giả sử chúng ta xóa user có ID là 2.

```
1. var express = require('express');  
2. var app = express();  
3. var fs = require("fs");  
4.  
5. var id = 2;  
6.  
7. app.get('/deleteUser', function (req, res) {  
8.  
9.   // Đầu tiên, đọc tất cả các User đang tồn tại.  
10.  fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {  
11.    data = JSON.parse( data );  
12.    delete data["user" + 2];  
13.  
14.    console.log( data );  
15.    res.end( JSON.stringify(data));  
16.  });  
17. })  
18. var server = app.listen(8081, function () {  
19.  
20.  var host = server.address().address  
21.  var port = server.address().port  
22.  console.log("Ung dụng Node.js đang lắng nghe tại địa chỉ: http://%s:%s", host,  
    port)  
23. })
```

Gọi service trên bởi sử dụng <http://127.0.0.1:8081/deleteUser> trên máy local .
Nó sẽ cho ra kết quả sau đây:

```
1. { user1:
2.   { name: 'huong',
3.     password: 'password1',
4.     profession: 'sinhvien',
5.     id: 1 },
6.   user3:
7.     { name: 'tuyen',
8.       password: 'password3',
9.       profession: 'laptrinhvien',
10.      id: 3 }
11. }
```

4.5 Sử dụng postman để kiểm tra API.

Writing tests

- Trong Postman, chúng ta có thể viết test riêng của mình bằng Javascript. Hiện tại có thể viết test một cách rất là ngắn gọn và hữu ích là sử dụng PM API mới.

- Dưới đây là một số hàm cơ bản thường sử dụng:

pm.test() là một hàm được sử dụng để viết các đặc tả kiểm thử. Cách viết này cho phép ta đặt tên kiểm thử chính xác và đảm bảo rằng phần còn lại của kịch bản không bị chặn trong trường hợp có lỗi.

Ví dụ:

```
1. // example using pm.response.to.have
2. pm.test("response is ok", function () {
3.   pm.response.to.have.status(200);
4. });
5.
6. // example using pm.expect()
7. pm.test("environment to be production", function () {
8.   pm.expect(pm.environment.get("env")).to.equal("production");
9. });
10.
11. // example using response assertions
12. pm.test("response should be okay to process", function () {
13.   pm.response.to.not.be.error;
14.   pm.response.to.have.jsonBody("");
```

```

15. pm.response.to.not.have.jsonBody("error");
16. });
17.
18. // example using pm.response.to.be*
19. pm.test("response must be valid and have a body", function () {
20.     // assert that the status code is 200
21.     pm.response.to.be.ok; // info, success, redirection, clientError, serverError, a
        re other variants
22.     // assert that the response has a valid JSON body
23.     pm.response.to.be.withBody;
24.     pm.response.to.be.json; // this assertion also checks if a body exists, so the a
        bove check is not needed
25. });

```

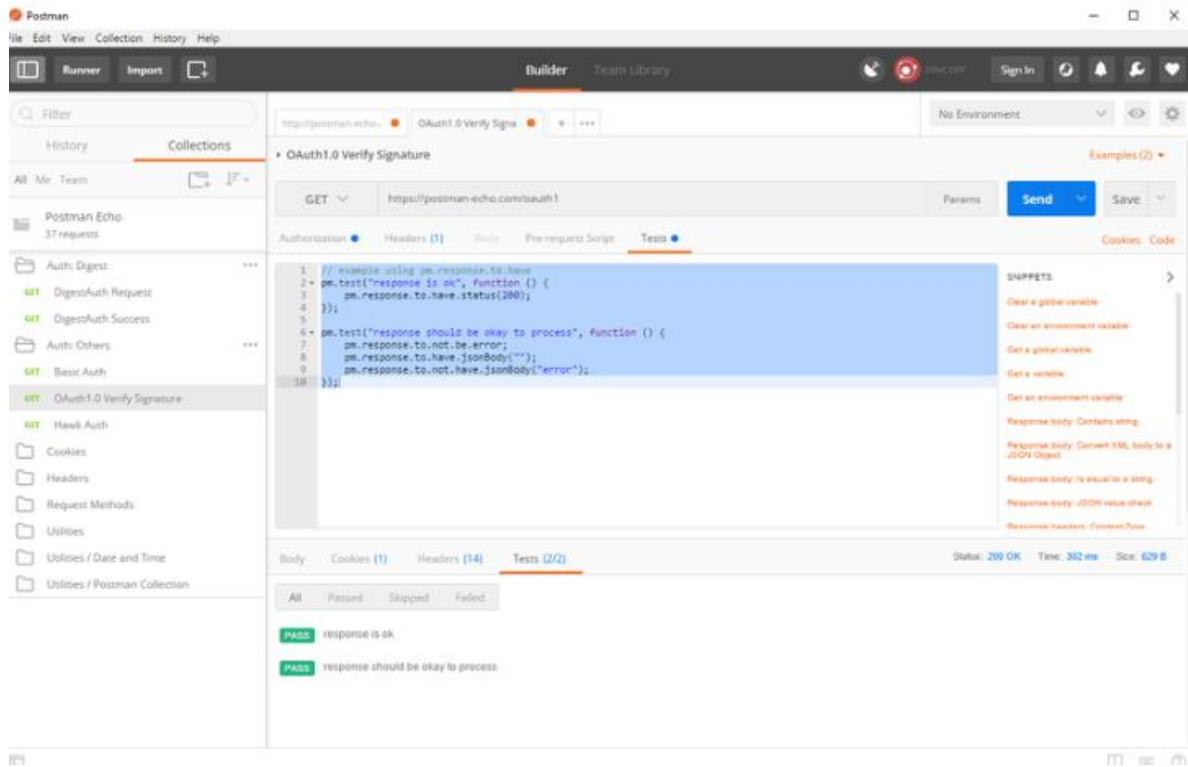
- pm.expect() là hàm được xây dựng dựa trên thư viện kiểm thử của Javascript. Tương tự, nó có thể viết kịch bản kiểm thử một cách dễ dàng và có thể giải quyết khẳng định của dữ liệu từ một phản hồi hoặc biến.

- Đối tượng `pm.response.to.be.*` cung cấp các dấu ngoặc đơn cho các kiểm tra dựa trên sử dụng phản hồi thường xuyên.

Test results

- Ở bước trên, sử dụng các hàm là ta có thể tự viết cho mình các kiểm thử. Tuy nhiên, làm thế nào để ta biết được nó có đi qua hay không?

- Sau khi chạy các yêu cầu với kiểm thử, chúng ta sẽ đi tới tab Tests để xem phản hồi kết quả trả về. Ở đây, mình sẽ thấy danh sách các kiểm thử và kết quả tương ứng là passed hay failed.



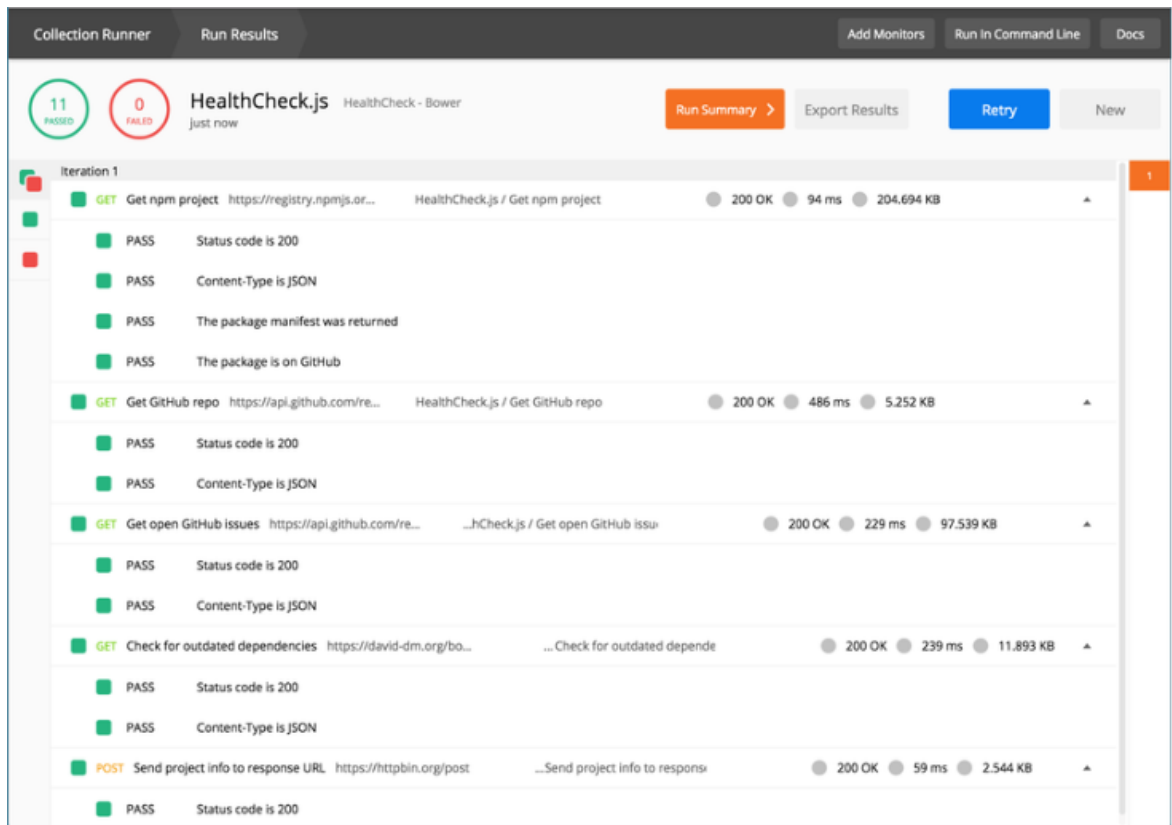
Hình 4: Danh sách kiểm thử và kết quả

Running tests

Có nhiều cách để chạy test sử dụng Postman, dưới đây là hai cách thông dụng hay dùng:

- Collection runner:

Trường hợp chạy test sử dụng Postman collection runner, chúng ta có thể view được kết quả trong thời gian thực:



Hình 5: Kết quả chạy test sử dụng Postman collection runner

Newman

- Việc chạy sử dụng Collection runner là một cách khá hay. Tuy nhiên nó vẫn mang tính thủ công. Để việc chạy các kiểm thử trở nên tự động, chúng ta có thể dùng Newman CLI.

- Ở đây mình sẽ trình bày về tích hợp dòng lệnh với Newman.

- Newman cho phép chạy và kiểm tra trực tiếp từ dòng lệnh. Nó được xây dựng để dễ dàng tích hợp với máy chủ và xây dựng hệ thống và nó được xây dựng dựa trên Node.js.

- Để chạy Newman , trước tiên cần cài Node.js

- Sau khi cài Node.js xong , chỉ cần chạy dòng lệnh sau trên Command line để cài đặt Newman:

`npm install -g newman`

CHƯƠNG 5: MONGODB

5.1 Giới thiệu về MongoDB và cách thiết lập MongoDB cục bộ.

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++.

Ngoài ra, MongoDB 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.

Khái niệm Collection

Collection là một nhóm các Document trong MongoDB. Nó tương đương như một bảng trong RDBMS. Do đó, một Collection tồn tại bên trong một cơ sở dữ liệu duy nhất. Các Collection không có ràng buộc Relationship như các hệ quản trị cơ sở dữ liệu khác nên việc truy xuất rất nhanh, chính vì thế mỗi collection có thể chứa nhiều thể loại khác nhau không giống như table trong hệ quản trị mysql là các field cố định. Các Document bên trong một Collection có thể có nhiều trường khác nhau. Đặc biệt, tất cả các Document trong một Collection là tương tự nhau hoặc với cùng mục đích liên quan.

Khái niệm Document

Một Document trong MongoDB, có cấu trúc tương tự như kiểu dữ liệu JSON, là một tập hợp các cặp key-value. Các Document có schema động, nghĩa là Document trong cùng một Collection không cần thiết phải có cùng một tập hợp các trường hoặc cấu trúc giống nhau, và các trường chung trong Document của một Collection có thể giữ các kiểu dữ liệu khác nhau.

Cấu trúc Document đơn giản

- Ví dụ dưới đây minh họa cấu trúc Document của một Blog site với một cặp key-value phân biệt bởi dấu phẩy.

```
1. {  
2.   _id: ObjectId(7df78ad8902c)  
3.   title: 'MongoDB Overview',  
4.   description: 'MongoDB is no sql database',  
5.   by: 'tutorials point',
```

```

6.   url: 'http://www.tutorialspoint.com',
7.   tags: ['mongodb', 'database', 'NoSQL'],
8.   likes: 100,
9.   comments: [
10.    {
11.      user: 'user1',
12.      message: 'My first comment',
13.      dateCreated: new Date(2011,1,20,2,15),
14.      like: 0
15.    },
16.    {
17.      user: 'user2',
18.      message: 'My second comments',
19.      dateCreated: new Date(2011,1,25,7,45),
20.      like: 5
21.    }
22.  ]
23. }

```

- Ở đây, `_id` là một số thập lục phân 12 byte để đảm bảo tính duy nhất của mỗi Document. Bạn có thể cung cấp `_id` trong khi chèn vào Document. Nếu bạn không cung cấp, thì MongoDB sẽ cung cấp một id duy nhất cho mỗi Document. Trong 12 byte này, 4 byte đầu là cho Timestamp hiện tại, 3 byte tiếp theo cho ID của thiết bị, 2 byte tiếp là process id của MongoDB Server và 3 byte còn lại là giá trị có thể tăng.

5.2 Thao tác dữ liệu MongoDB.

Tạo Database trong MongoDB

- Lệnh `use DATABASE_NAME` trong MongoDB được sử dụng để tạo cơ sở dữ liệu. Lệnh này sẽ tạo một cơ sở dữ liệu mới, nếu nó chưa tồn tại, nếu không thì, lệnh này sẽ trả về cơ sở dữ liệu đang tồn tại.

- Cú pháp cơ bản của lệnh `use DATABASE_NAME` là như sau:

`use DATABASE_NAME`

Xóa Database trong MongoDB

- Lệnh `db.dropDatabase()` trong MongoDB được sử dụng để xóa một cơ sở dữ liệu đang tồn tại.

- Cú pháp cơ bản của lệnh `dropDatabase()` là như sau:

`db.dropDatabase()`

- Lệnh này sẽ xóa cơ sở dữ liệu đã chọn. Nếu bạn không chọn bất kỳ cơ sở dữ liệu nào, thì nó sẽ xóa cơ sở dữ liệu mặc định test.

Tạo Collection trong MongoDB

- Phương thức `db.createCollection(name, options)` trong MongoDB được sử dụng để tạo Collection.

- Cú pháp cơ bản của lệnh `createCollection()` trong MongoDB như sau:

`db.createCollection(name, options)`

- Trong lệnh trên, name là tên của Collection. Options là một Document và được sử dụng để xác định cấu hình cho Collection

- Tham số options là tùy ý, vì thế bạn chỉ cần xác định tên của Collection. Dưới đây là danh sách các tùy chọn bạn có thể sử dụng:

Trường	Kiểu	Miêu tả
capped	Boolean	(Tùy ý) Nếu true, kích hoạt một Capped Collection. Đây là một Collection có kích cỡ cố định mà tự động ghi đè các entry cũ nhất khi nó tiếp cận kích cỡ tối đa. Nếu bạn xác định là true, thì bạn cũng cần xác định tham số size
autoIndexID	Boolean	(Tùy ý) Nếu true, tự động tạo chỉ mục trên các trường <code>_id</code> . Giá trị mặc định là false
size	Số	(Tùy ý) Xác định kích cỡ tối đa (giá trị byte) cho một Capped Collection. Nếu tham số capped là true, thì bạn cũng cần xác định trường này
max	Số	(Tùy ý) Xác định số Document tối đa được cho phép trong một Capped Collection

Bảng 5: Danh sách các tùy chọn

- Trong khi thực hiện việc chèn dữ liệu vào Document, đầu tiên MongoDB kiểm tra trường size của Capped Collection, sau đó nó kiểm tra trường max.

Xóa Collection trong MongoDB

- Phương thức `db.collection.drop()` trong MongoDB được sử dụng để xóa một Collection từ cơ sở dữ liệu.

- Cú pháp cơ bản của lệnh `drop()` là như sau:

```
db.COLLECTION_NAME.drop()
```

Chèn Document trong MongoDB

- Để chèn dữ liệu vào trong Collection trong MongoDB, bạn cần sử dụng phương thức `insert()` hoặc `save()`.

- Cú pháp cơ bản của lệnh `insert()` như sau:

```
>db.COLLECTION_NAME.insert(document)
```

Truy vấn Document trong MongoDB

- Cú pháp cơ bản của phương thức `find()` là như sau:

```
>db.COLLECTION_NAME.find()
```

- Phương thức `find()` sẽ hiển thị tất cả Document ở dạng không có cấu trúc (hiển thị không theo cấu trúc nào).

- Phương thức `pretty()` trong MongoDB

- Để hiển thị các kết quả theo một cách đã được định dạng, bạn có thể sử dụng phương thức `pretty()`.

```
>db.mycol.find().pretty()
```

Cập nhật Document trong MongoDB

- Phương thức `update()` hoặc `save()` trong MongoDB được sử dụng để cập nhật Document vào trong một Collection. Phương thức `update()` cập nhật các giá trị trong Document đang tồn tại trong khi phương thức `save()` thay thế Document đang tồn tại với Document đã truyền trong phương thức `save()` đó.

- Phương thức `update()` cập nhật các giá trị trong Document đang tồn tại.

- Cú pháp cơ bản của phương thức `update()` là như sau:

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA,  
UPDATED_DATA)
```

- Phương thức `save()` thay thế Document đang tồn tại với Document mới đã được truyền trong phương thức `save()` này.

- Cú pháp cơ bản của phương thức `save()` như sau:

```
>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

Xóa Document trong MongoDB

- Phương thức `remove()` trong MongoDB được sử dụng để xóa Document từ Collection. Phương thức `remove()` nhận hai tham số. Tham số đầu tiên deletion criteria xác định Document để xóa, và tham số thứ hai là `justOne`.

deletion criteria : (Tùy ý) Xác định Document để xóa.

justOne : (Tùy ý) Nếu được thiết lập là `true` hoặc `1`, thì chỉ xóa một Document.

- Cú pháp cơ bản của phương thức `remove()` là như sau:

```
>db.COLLECTION_NAME.remove(DELETION_CRITTERIA)
```

- Nếu có nhiều bản ghi và bạn chỉ muốn xóa bản ghi đầu tiên, thì thiết lập tham số `justOne` trong phương thức `remove()`.

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
```

Projection trong MongoDB

- Trong MongoDB, ý nghĩa của projection là chỉ chọn dữ liệu cần thiết thay vì chọn toàn bộ dữ liệu của một Document. Nếu một Document có 5 trường và bạn chỉ cần 3 trường, thì bạn chỉ nên chọn 3 trường từ Document đó.

- Phương thức `find()` trong MongoDB, đã được giải thích trong Truy vấn Document, chấp nhận tham số tùy ý thứ hai mà là danh sách các trường bạn muốn lấy. Trong MongoDB, khi bạn thực thi phương thức `find()`, thì nó hiển thị tất cả các trường của một Document. Để giới hạn điều này, bạn cần thiết lập danh sách các trường với giá trị `1` hoặc `0`. Giá trị `1` được sử dụng để hiển thị trường, trong khi `0` được sử dụng để ẩn trường.

- Cú pháp cơ bản của phương thức `find()` với projection là như sau:

```
>db.COLLECTION_NAME.find({}, {KEY:1})
```

Giới hạn bản ghi trong MongoDB

- Để giới hạn các bản ghi trong MongoDB, bạn cần sử dụng phương thức `limit()`. Phương thức `limit()` nhận một tham số ở dạng kiểu số, là số Document mà bạn muốn hiển thị.

- Cú pháp cơ bản của phương thức `limit()` là như sau:

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

- Ngoài phương thức `limit()`, trong MongoDB có một phương thức khác là `skip()` cũng nhận một tham số ở dạng số và được sử dụng để nhảy qua số Document đã xác định.

- Cú pháp cơ bản của phương thức `skip()` là như sau:

```
>db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```

Sắp xếp bản ghi trong MongoDB

- Để sắp xếp các Document trong MongoDB, bạn cần sử dụng phương thức `sort()`. Phương thức `sort()` nhận một Document chứa danh sách các trường cùng với thứ tự sắp xếp của chúng. Để xác định thứ tự sắp xếp, 1 và -1 được sử dụng. 1 được sử dụng cho thứ tự tăng dần, trong khi -1 được sử dụng cho thứ tự giảm dần.

- Cú pháp cơ bản của phương thức `sort()` là như sau:

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

Chỉ mục (Index) trong MongoDB

- Chỉ mục (Index) hỗ trợ việc phân giải các truy vấn hiệu quả hơn. Nếu không có chỉ mục, MongoDB phải quét qua mọi Document của một Collection để chọn các Document mà kết nối với lệnh truy vấn. Việc quét này có thể không hiệu quả và yêu cầu MongoDB xử lý một số lượng lớn dữ liệu.

- Chỉ mục (Index) là các cấu trúc dữ liệu đặc biệt, lưu giữ một phần nhỏ của tập hợp dữ liệu, giúp việc "vọc" vào Collection một cách dễ dàng hơn. Chỉ mục lưu giữ giá trị của một trường cụ thể hoặc tập hợp các trường, được sắp xếp bởi giá trị của trường như đã được xác định trong chỉ mục.

- Để tạo một chỉ mục, bạn cần sử dụng phương thức `ensureIndex()` của MongoDB.

- Cú pháp cơ bản của phương thức `ensureIndex()` là như sau:

```
>db.COLLECTION_NAME.ensureIndex({KEY:1})
```

- Ở đây, key là tên của trường mà bạn muốn tạo chỉ mục và 1 là cho thứ tự tăng dần. Để tạo chỉ mục theo thứ tự giảm dần, bạn cần sử dụng -1.

Aggregation trong MongoDB

- Aggregation có thể hiểu là sự tập hợp. Các Aggregation operation xử lý các bản ghi dữ liệu và trả về kết quả đã được tính toán. Các phép toán tập hợp nhóm các giá trị từ nhiều Document lại với nhau, và có thể thực hiện nhiều phép toán đa dạng trên dữ liệu đã được nhóm đó để trả về một kết quả duy nhất. Trong SQL, count(*) và GROUP BY là tương đương với Aggregation trong MongoDB.

- Với Aggregation trong MongoDB, bạn nên sử dụng phương thức `aggregate()`.

- Cú pháp cơ bản của phương thức `aggregate()` là như sau:

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Tạo Backup trong MongoDB

- Để tạo Backup của cơ sở dữ liệu trong MongoDB, bạn nên sử dụng lệnh `mongodump`. Lệnh này sẽ dump tất cả dữ liệu của Server vào trong dump directory. Có nhiều tùy chọn có sẵn từ đó bạn có thể giới hạn lượng dữ liệu hoặc tạo backup của Remote Server.

- Cú pháp cơ bản của lệnh `mongodump` trong MongoDB là:

```
>mongodump
```

- Để phục hồi dữ liệu đã sao lưu trong MongoDB, bạn sử dụng lệnh `mongorestore`. Lệnh này phục hồi tất cả dữ liệu từ thư mục sao lưu.

- Cú pháp cơ bản của lệnh `mongorestore` là:

```
>mongorestore
```


PHẦN II: XÂY DỰNG WEBSITE

CHƯƠNG 6: XÂY DỰNG WEBSITE DEMO

6.1 Giới thiệu web demo Website Tin Tức

“Tin tức tích lũy trong mỗi chúng ta được gọi là kiến thức, tài sản quý giá nhất của con người”

Với sự bùng nổ thông tin trên internet, vai trò của các trang thông tin điện tử trực tuyến càng trở nên quan trọng. Khác với báo chí truyền thông có giới hạn thời gian cập nhật tin tức, các tờ báo trực tuyến đã cung cấp được sự tiện lợi trong việc cập nhật và phát hành thông tin. Về phía người dùng, họ có thể xem thông tin mọi lúc mọi nơi. Về phía những người cung cấp thông tin, các nhà báo, họ có thể dễ dàng cập nhật những tin tức mới nhất, thời sự nhất. Do đó việc sử dụng các trang thông tin trực tuyến luôn là điều cần thiết hiện nay nhằm đáp ứng nhu cầu cập nhật thông tin của mỗi người.

Tin tức là những việc đã xảy ra dù tốt dù xấu, để giúp con người biết những chuyện xung quanh và trên Thế giới. Ngày nay nhờ thông tin truyền thông nhanh, cho nên bất cứ chuyện gì vừa xảy ra ở đâu trên thế giới thì ta đều có thể biết ngay, nhờ đó mà có thể học được nhiều cái hay cũng như tránh được những chuyện xấu xảy ra, như các trận sóng Thần, bão táp, núi lửa sắp đến, các chất độc hại trong thức ăn.... giúp con người biết trước mà tránh khỏi các nguy hiểm sắp đến.

Tin tức vô cùng quan trọng nó cho người ta tri thức và là cơ sở để người ta tiến hành mọi việc lớn nhỏ. Khi có Internet, tin tức càng quan trọng vì tốc độ lan truyền nhanh ảnh hưởng ngay tức thì trên diện rộng.

6.2 Nền tảng và công cụ phát triển

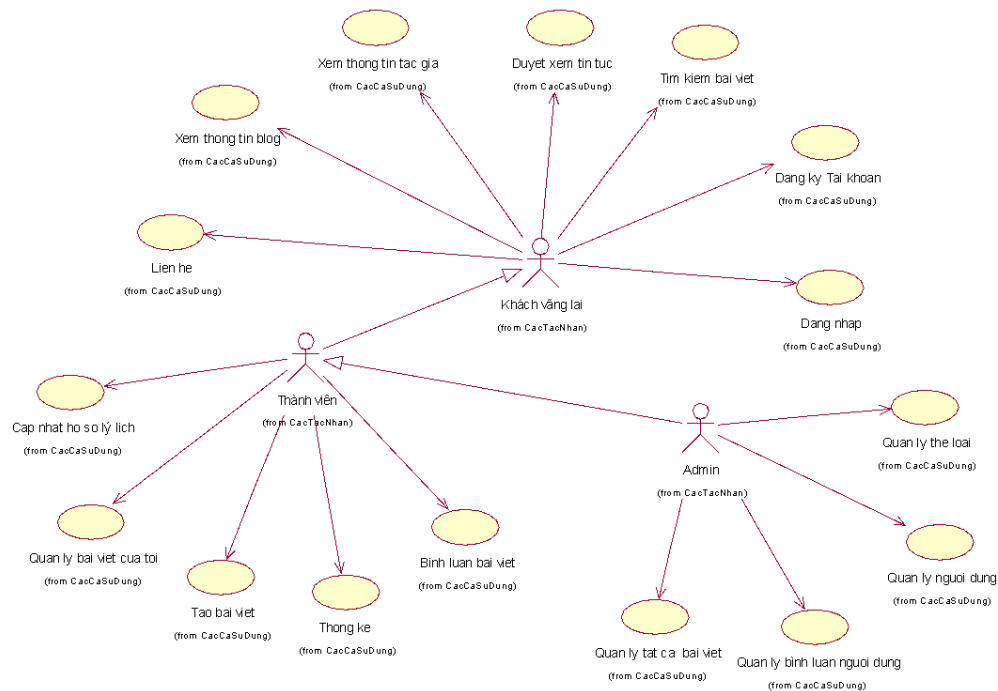
Hoạt động trên nền tảng web. HTML5 + CSS3 + framework bootstrap 4 + ngôn ngữ lập trình Javascript và môi trường Nodejs, Express framework + hệ quản trị CSDL MongoDB

Công cụ phát triển:

- Visual Studio Code Version 1.33
- MongoDB Compass.
- Postman

6.3 Chức năng

6.3.1 Sơ đồ Use-case tổng quan



Hình 5: Sơ đồ Use-case

6.3.2 Danh sách các Actor

STT	Actor	Ý nghĩa
1	Khách vãng lai	Khách, người dùng khi chưa đăng nhập vào hệ thống
2	Thành viên	Thành viên, người dùng đã có tài khoản và đã đăng nhập vào hệ thống
3	Admin	Người quản trị hệ thống

Bảng 6: Danh sách các Actor

6.3.3. Danh sách các Use-case

STT	Use-case	Ý nghĩa
1	Dang nhap	Đăng nhập vào hệ thống
2	Dang ky tai khoan	Đăng ký thành viên
3	Tim kiem bai viet	Tìm kiếm bài viết
4	Duyet xem tin tuc	Duyệt xem bài viết
5	Xem thong tin tac gia	Xem thông tin người đăng bài
6	Xem thong tin blog	Xem thông tin website
7	Lien he	Liên hệ ý kiến với người quản trị
8	Cap nhat ho so ly lich	Cập nhật thông tin thành viên
9	Quan ly bai viet cua toi	Quản lý bài viết của thành viên
10	Tao bai viet	Đăng bài viết mới
11	Thong ke	Thống kê số bài viết
12	Binh luan bai viet	Cho phép thành viên bình luận bài viết
13	Quan ly tat ca bai viet	Người quản trị được phép quản lý tất cả bài viết của thành viên
14	Quan ly binh luan nguoi dung	Quản trị viên được phép quản lý bình luận của thành viên
15	Quan ly nguoi dung	Quản trị viên được phép quản lý tài khoản thành viên
16	Quan ly the loai	Quản lý thể loại người quản trị có thể thêm sửa xóa

Bảng 7: Danh sách các Use-case

6.4 Kết quả thực nghiệm

6.4.1 Tổ chức

6.4.1.1 Database

Gồm các collections:

Category:

```
1. const mongoose = require('mongoose');
2. const URLSlugs = require('mongoose-url-slugs');
3.
4. const Schema = mongoose.Schema;
5.
6. const CategorySchema = new Schema({
7.   name: {
8.     type: String,
9.     required: true,
10.  },
11.  slug: {
12.    type: String
13.  },
14.  date: {
15.    type: Date,
16.    default: Date.now()
17.  }
18. }, {usePushEach: true})
19.
20.
21. CategorySchema.plugin(URLSlugs('name', {field: 'slug'}))
22. module.exports = mongoose.model('categories', CategorySchema)
```

Comment:

```
1. const mongoose = require('mongoose');
2. const moment = require('moment')
3.
4. const Schema = mongoose.Schema;
5.
6. const CommentSchema = new Schema({
7.   user: {
8.     type: Schema.Types.ObjectId,
9.     ref: 'users'
10.  },
11.  body: {
12.    type: String,
13.    required: true,
14.  },
15.  approveComment: {
16.    type: Boolean,
17.    default: true
18.  },
19.  date: {
20.    type: Date,
21.    default: moment()
22.  }
23. })
24. })
25.
26. module.exports = mongoose.model('comments', CommentSchema)
```

Post:

```
1. const PostSchema = new Schema({
2.   user : {
3.     type: Schema.Types.ObjectId,
4.     ref: 'users'
5.   },
6.   category : {
7.     type: Schema.Types.ObjectId,
8.     ref: 'categories'
9.   },
10.  title: {
11.    type: String,
12.    required: true,
13.    text: true
14.  },
15.  status: {
16.    type: String,
17.    default: 'public',
18.  },
19.  allowComments: {
20.    type: Boolean,
21.    required: true,
22.  },
23.  body: {
24.    type: String,
25.    required: true,
26.    text: true
27.  },
28.  file: {
29.    type: String,
30.    text: true
31.  },
32.  date: {
33.    type: Date,
34.    default: moment()
35.  },
36.  slug: {
37.    type: String,
38.    text: true
39.  },
40.  comments: [{
41.    type: Schema.Types.ObjectId,
42.    ref: 'comments'
```

```
43.   }]  
44. }, {usePushEach: true})  
45. PostSchema.plugin(URLSlugs('title', {field: 'slug'}))  
46. module.exports = mongoose.model('posts', PostSchema)
```

User:

```
1.  const mongoose = require('mongoose');  
2.  
3.  const Schema = mongoose.Schema;  
4.  
5.  const UserSchema = new Schema({  
6.    firstName: {  
7.      type: String,  
8.      required: true,  
9.    },  
10.   lastName: {  
11.     type: String,  
12.     required: true,  
13.   },  
14.   email: {  
15.     type: String,  
16.     required: true,  
17.   },  
18.   password: {  
19.     type: String,  
20.     required: true,  
21.   },  
22.   occupations: {  
23.     type: String,  
24.     required: true,  
25.   },  
26.   file: {  
27.     type: String,  
28.   },  
29.   isAdmin: {  
30.     type: Boolean,  
31.     required: false,  
32.   }  
33. })  
34. })  
35. UserSchema.methods.testMethod = function(){  
36. }  
37. module.exports = mongoose.model('users', UserSchema)
```

6.4.1.2 Tạo routes

Router (/categories và /categories/:id và /categories/create và /categories/edit:id)

```
1. const express = require('express');
2. const adminController = require('../controllers/admin/categories')
3. const {userAuthenticated, admin} = require('../helpers/authentication')
4. const router = express.Router();
5.
6. router.all('/', [userAuthenticated],(req, res, next) => {
7.   req.app.locals.layout = 'admin'
8.   next()
9. })
10.
11. router.route('/')
12.   .get(admin, adminController.index)
13.
14. router.route('/:id')
15.   .delete(admin, adminController.delete)
16.
17. router.route('/create')
18.   .post(admin, adminController.createCategory)
19.
20. router.route('/edit/:id')
21.   .get(admin, adminController.edit)
22.   .put(admin, adminController.put)
23.
24. module.exports = router
```

Router (/comments, và /comments/:id /comments/approve-comment)

```
1. const express = require('express');
2. const adminController = require('../controllers/admin/comments')
3. const {userAuthenticated, admin} = require('../helpers/authentication')
4. const router = express.Router();
5.
6. router.all('/', [userAuthenticated], (req, res, next) => {
7.   req.app.locals.layout = 'admin'
8.   next()
9. })
10. router.route('/')
11.   .get(admin, adminController.getComments)
12.   .post(adminController.postComments)
13. router.route('/:id')
```



```

14. .get(adminController.getCommentsId)
15. .delete(adminController.deleteComments)
16. router.route('/approve-comment')
17. .post(adminController.postApproveComments)
18. module.exports = router

```

6.4.1.3 Tạo controller

Category:

```

1. const Category = require('../models/category')
2. const { isEmpty, uploadDir } = require('../helpers/upload-helper')
3.
4. module.exports = {
5.   index : (req, res) =>{
6.     Category.find({}).then(categories =>{
7.       res.render('admin/categories/index', {categories: categories})
8.     })
9.
10.  },
11.  createCategory: (req, res) =>{
12.    const newCategory = new Category({
13.      name: req.body.name
14.    })
15.    newCategory.save().then(savedCategory =>{
16.      res.redirect('/admin/categories')
17.    })
18.  },
19.  edit : (req, res) =>{
20.    Category.findOne({_id : req.params.id})
21.      .then(category => {
22.        res.render('admin/categories/edit', {category: category})
23.      })
24.  },
25.  put : (req, res) =>{
26.    Category.findOne({_id : req.params.id})
27.      .then(category => {
28.        category.name = req.body.name;
29.        category.save().then(save => {
30.          res.redirect('/admin/categories')
31.        })
32.      })
33.  },
34.  delete: (req, res) =>{

```

```

35.     Category.remove({_id : req.params.id}).then(result => {
36.         res.redirect('/admin/categories')
37.     })
38.
39. }
40.
41. }

```

Comments:

```

1.  const Comment = require('../models/Comment')
2.  const Post = require('../models/post')
3.
4.  module.exports = {
5.      deleteComments: (req, res) => {
6.          Comment.remove({ _id: req.params.id }).then(deleteIt => {
7.              Post.findOneAndUpdate({ comments: req.params.id }, { $pull: { comments: req.params.id } }, (err, data) => {
8.                  if (err) console.log(err);
9.
10.                 res.redirect('/admin/comments')
11.             })
12.         });
13.     },
14.     getComments: (req, res) => {
15.         const perPage = 10;
16.         const page = req.query.page || 1;
17.
18.         Comment.find({})
19.             .populate('user')
20.             .sort({ "date": -1 })
21.             .skip((perPage * page) - perPage)
22.             .limit(perPage)
23.             .then(comments => {
24.                 Comment.find({})
25.                 .countDocuments().then(postCount => {
26.                     if (postCount === 0) {
27.                         postCount = 10
28.                     }
29.                     res.render('admin/comments',
30.                         {
31.                             comments: comments,
32.                             current: parseInt(page),
33.                             pages: Math.ceil(postCount / perPage),

```

```

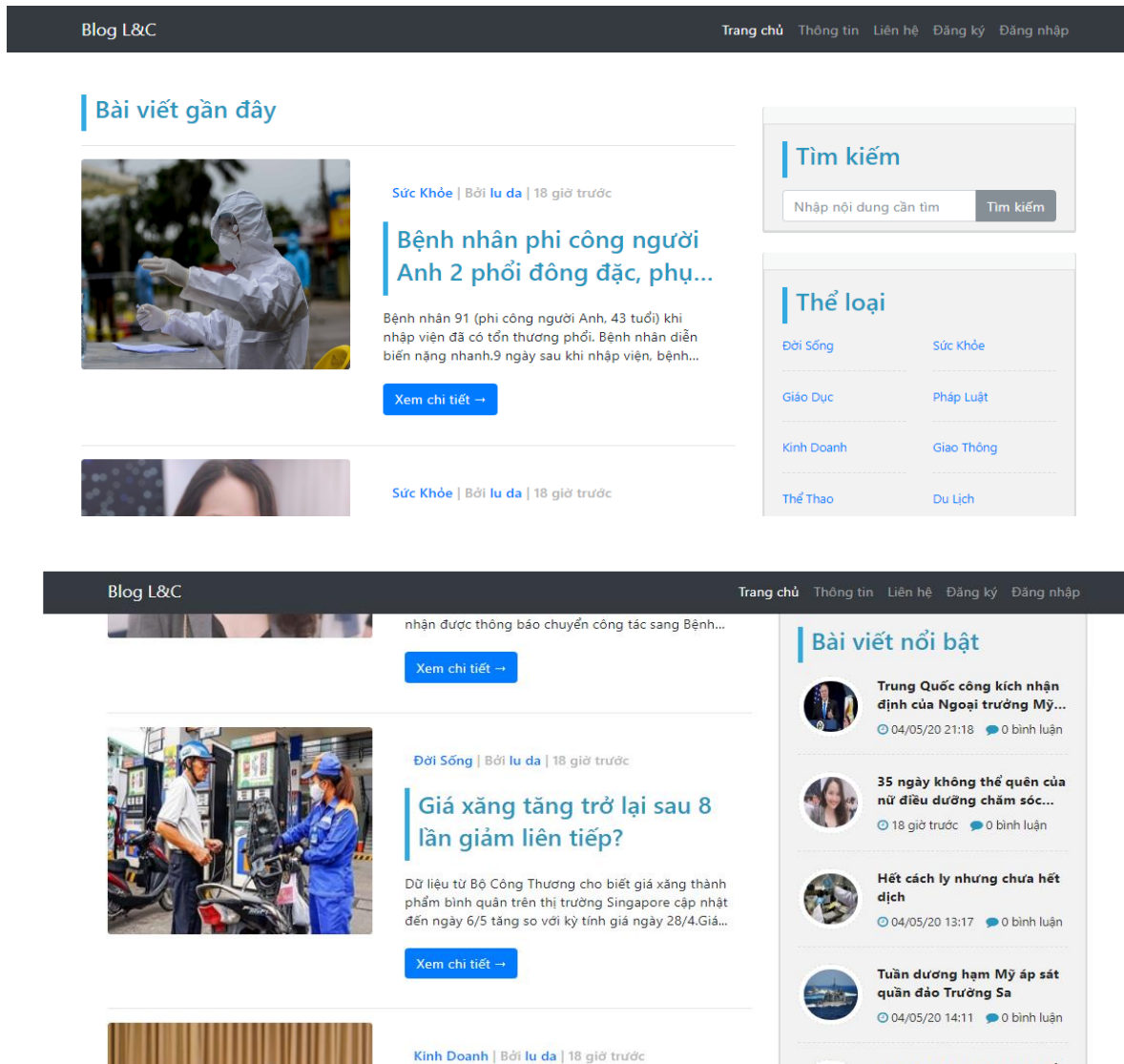
34.         })
35.     })
36. })
37. },
38. getCommentsId: (req, res) => {
39.     Post.find({ comments: req.params.id })
40.     .then(post => {
41.         let slug = post[0].slug;
42.         res.redirect(`/post/${slug}`)
43.     })
44. },
45. postComments: (req, res) => {
46.     let error = "";
47.     let comments_content = "";
48.     Post.findOne({ _id: req.body.id }).then(post => {
49.         if (!req.body.body) {
50.
51.             req.flash('success_message', 'Vui lòng nhập bình luận')
52.             return res.redirect(`/post/${post.slug}`)
53.
54.         } else {
55.             const newComment = new Comment({
56.                 user: req.session.userId,
57.                 body: req.body.body
58.             })
59.             post.comments.push(newComment);
60.             post.save().then(savedPost => {
61.                 newComment.save().then(savedComment => {
62.                     res.redirect(`/post/${post.slug}`)
63.
64.                 })
65.             })
66.         }
67.
68.
69.     })
70. },
71. postApproveComments: (req, res) => {
72.     Comment.findByIdAndUpdate(req.body.id, { $set: { approveComment: req.body.approveComment } }, (err, result) => {
73.         if (err) return err;
74.         res.send(result);
75.
76.     })

```

77.
78. }
79. }

6.4.2 Thực hiện

6.4.2.1 Trang chủ



Hình 7.1: Trang chủ

6.4.2.2 Trang đăng ký thành viên

Blog L&C

[Trang chủ](#) [Thông tin](#) [Liên hệ](#) [Đăng ký](#) [Đăng nhập](#)

Đăng ký tài khoản

Họ

Nhập họ

Tên

Nhập tên

Địa chỉ Email

daotunglamht@gmail.com

Nghề nghiệp

Nhập nghề nghiệp

Mật khẩu

Xác nhận

Xác nhận mật khẩu

Đăng ký

[Trang đăng nhập](#)
[Trang chủ](#)

Hình 7.2: Trang đăng ký thành viên

6.4.2.3 Trang đăng nhập

Blog L&C

[Trang chủ](#) [Thông tin](#) [Liên hệ](#) [Đăng ký](#) [Đăng nhập](#)

Đăng nhập

Email

daotunglamht@gmail.com

Mật khẩu

☐ Lưu đăng nhập

Đăng nhập

[Đăng ký một tài khoản](#)
[Quên mật khẩu](#)
[Trang chủ](#)

Hình 7.3: Trang đăng nhập

6.4.2.4 Trang liên hệ

Liên hệ với chúng tôi

"Nếu bạn có câu hỏi hay ý kiến? Xin đừng ngần ngại liên hệ trực tiếp với chúng tôi. Nhóm chúng tôi sẽ trả lời câu hỏi của bạn!"


Tên

Địa chỉ Email


Chủ đề

Tin nhắn

Gửi


1/22 Lê Đức Thọ , Phường 7 quận
Gò Vấp, Thành Phố Hồ Chí Minh


0362220343
0369854539


daotunglamht1@gmail.com
luongcongcuong@gmail.com

Hình 7.4: Trang liên hệ

6.4.2.5 Trang xem chi tiết bài viết

Blog L&C

Trang chủThông tinLiên hệĐăng kýĐăng nhập



Giá xăng tăng trở lại sau 8 lần giảm liên tiếp?

🕒 19 giờ trước | bởi [lu da](#)

Dữ liệu từ Bộ Công Thương cho biết giá xăng thành phẩm bình quân trên thị trường Singapore cập nhật đến ngày 6/5 tăng so với kỳ tính giá ngày 28/4.

Giá xăng RON 92 dùng để pha chế xăng E5 RON 92 trung bình 23,12 USD/thùng, xăng RON 95 là 24,85 USD/thùng, cùng tăng khoảng 26% so với kỳ trước. Giá dầu trung bình không biến động mạnh.

10 KỶ ĐIỀU CHỈNH GIÁ XĂNG GẦN NHẤT

● E5 RON 92

● RON 95

Tìm kiếm

Nhập nội dung cần tìm

Tìm kiếm

Thể loại

[Đời Sống](#)

[Sức Khỏe](#)

[Giáo Dục](#)

[Pháp Luật](#)

[Kinh Doanh](#)

[Giao Thông](#)

[Thể Thao](#)

[Du Lịch](#)

[Quản Sự](#)


[Game](#)

Hình 7.5: Trang xem chi tiết bài viết

6.4.2.6 Trang xem thông tin thành viên đang bài

Blog L&C

Trang chủ Quản lý bài viết Thông tin Liên hệ



lu da
sinh viên

Serie A trở lại từ ngày 13/6

14/05/20 15:13 0 bình luận

NỔNG: Sập công trình kinh hoàng ở Đồng Nai, 10 người tử vong

14/05/20 15:13 0 bình luận

Đồng Nhi ra MV tặng mẹ

14/05/20 10:53 0 bình luận

Bệnh nhân phi công người Anh 2 phổi đông đặc, phụ thuộc hoàn toàn vào ECMO

12/05/20 15:33 0 bình luận

35 ngày không thể quên của nữ điều dưỡng chăm sóc bệnh nhân Covid-19

12/05/20 15:33 0 bình luận

Giá xăng tăng trở lại sau 8 lần giảm liên tiếp?

12/05/20 15:33 0 bình luận

EVN: Giảm gần 1.000 tỷ đồng tiền điện cho khách hàng

12/05/20 15:33 0 bình luận

Hình 7.6: Thông tin profile thành viên

6.4.2.7 Trang quản trị của Admin



Hình 7.7: Trang quản trị của Admin

6.4.2.8 Trang quản lý bài viết của thành viên



Hình 7.8: Trang quản lý bài viết của thành viên

Trang chủ									
<div> Dashboard Thông tin cá nhân Đăng bài </div>									
Tất cả bài viết									
STT	IMAGE	Tiêu đề	Trạng thái	Thể loại	Cho phép bình luận	Xem bài viết	Thời gian		
0		Serie A trở lại từ ngày 13/6	public	Thể Thao	true	Xem	14/05/20 15:13	Sửa	Xóa
1		NÓNG: Sập công trình kinh hoàng ở Đồng Nai, 10 người tử vong	public	Pháp Luật	true	Xem	14/05/20 15:13	Sửa	Xóa
2		Đồng Nhi ra MV tặng mẹ	public	Đời Sống	true	Xem	14/05/20 10:53	Sửa	Xóa
3		Bệnh nhân phi công người Anh 2 phổi đông đặc, phụ thuộc hoàn toàn vào ECMO	public	Sức Khỏe	true	Xem	12/05/20 15:33	Sửa	Xóa
4		35 ngày không thể quên của nữ điều dưỡng chăm sóc bệnh nhân Covid-19	public	Sức Khỏe	true	Xem	12/05/20 15:33	Sửa	Xóa
5		Giá xăng tăng trở lại sau 8 lần giảm liên tiếp?	public	Đời Sống	true	Xem	12/05/20 15:33	Sửa	Xóa

Hình 7.9: Một số chức năng của thành viên

KẾT LUẬN

1. Kết quả đạt được

Hai tháng, một khoảng thời gian không dài, nhưng với sự chỉ bảo và hướng dẫn của thầy **TRẦN ĐỨC DOANH** cùng với sự nỗ lực làm việc của nhóm thực hiện đề án, đề tài “tìm hiểu framework node.js và angular dùng xây dựng website tin tức minh họa ” của chúng em đã được hoàn thành.

Với những thuận lợi và khó khăn trong quá trình tìm hiểu NodeJs, website demo về cơ bản đã hoàn thành nhưng không tránh phần sai sót. Tuy nhiên, chúng em đã rất nỗ lực và đã hoàn thành được những nội dung chính sau:

- Tìm hiểu cơ bản về công nghệ NodeJs
- Vận dụng ngôn ngữ HTML, Javascript, NodeJs, MongoDB và công cụ Visual Studio Code, MongoDB Compass, Postman vào việc xây dựng website tin tức.
- Phần giao diện người dùng: giao diện thân thiện, cho phép bạn đọc dễ dàng xem tin, tìm kiếm tin, đăng bài, ý kiến cho mỗi tin.
- Phần quản trị: Đã xây dựng được hệ thống quản lý dữ liệu của website, giúp cho những người quản trị dễ dàng quản lý thông tin, dữ liệu, xem, thêm, xóa , cập nhật dữ liệu cho website.

2. Hướng phát triển

Chúng em sẽ tiếp tục tìm hiểu sâu hơn về công nghệ Nodejs và tiếp tục phát triển website demo Tin tức. Hiểu được tầm quan trọng của tin tức, chúng em luôn muốn hoàn thiện website tin tức của mình tốt hơn, phân tích kỹ hơn các yêu cầu và xây dựng giao diện hoàn thiện hơn, có tính chuyên nghiệp hơn.

TÀI LIỆU THAM KHẢO

1. Express - Node.js web application framework, <https://expressjs.com/> 10.3.2020
2. API Reference Documentation, <https://developer.mozilla.org/vi/docs/Learn> 03.2020
3. MongoDB Compass, <https://docs.mongodb.com/compass/master/> 04.2020
4. Một cái nhìn tổng quan nhất về Nodejs, <https://viblo.asia/p/mot-cai-nhin-tong-quan-nhat-ve-nodejs-Ljy5VeJ3lra> 03.2020
5. Node.js là gì và tại sao tôi nên học lập trình Node.js?
<https://techmaster.vn/posts/33428/nodejs-la-gi-va-tai-sao-toi-nen-hoc-lap-trinh-nodejs>
03.2020