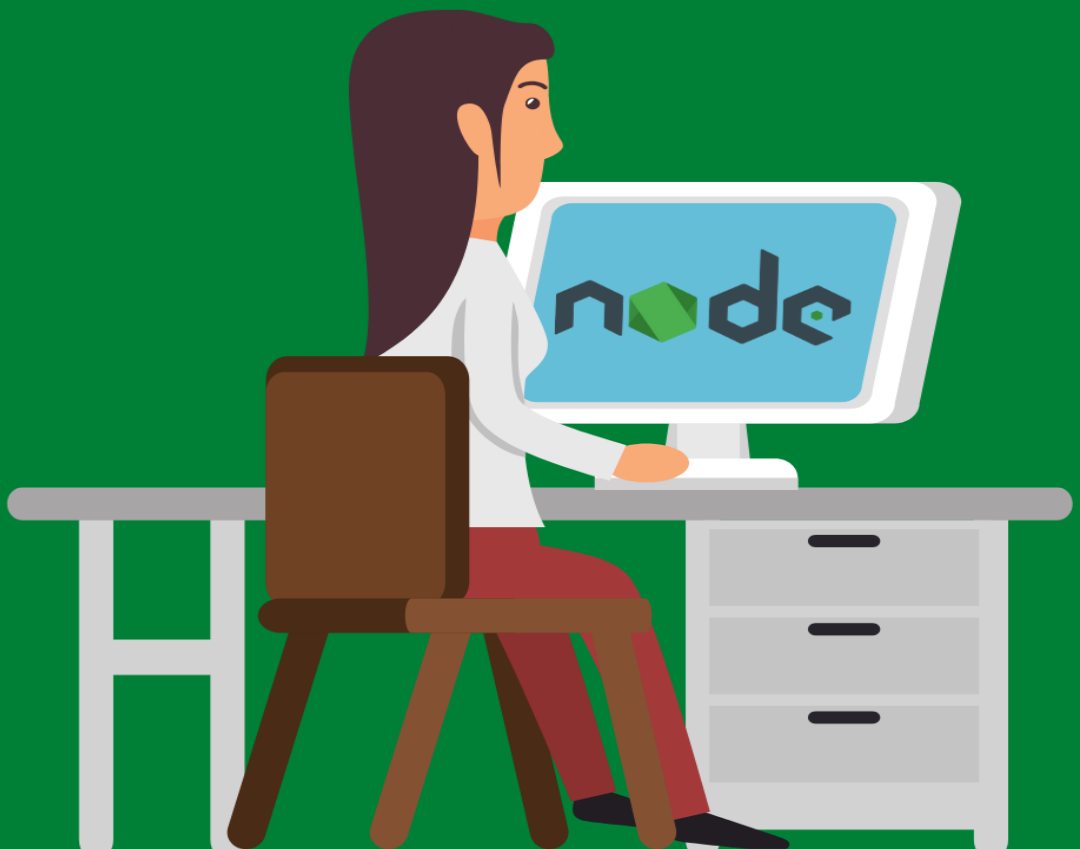
TỰ XÂY DỰNG ỨNG DỤNG TỪ A-Z

LẬP TRÌNH NODE.JS THẬT ĐƠN GIẢN

BY VNTALKING.COM



MỤC LỤC

Lời nói đầu

Cách học đúng cách

Yêu cầu trình độ

Liên hệ với tác giả

Giới thiệu

Giới thiệu ứng dụng blog sẽ xây dựng

Node.js là gì?

Cài đặt Node.js

Tạo server Nodejs đầu tiên

Hiểu hơn về request và response

Giới thiệu về NPM và Express

Cài đặt Custom Package với NPM

Giới thiệu Express

Xử lý request với Express

Bất đồng bộ với Call Back

Trả về một file html cho client

Trả về static resource (image, css, js...) cho client

Tổng kết

Bước đầu tạo web app với Express

Tải web template

Tự động khởi động server mỗi khi thay đổi mã nguồn

Npm start

Tạo thư mục public chứa tệp static

Tạo Page routes

Tổng kết

Templating Engine

Giới thiệu Template engine

Layout

Tổng kết

Giới thiệu MongoDB

- Kiến trúc của MongoDB
- Cài đặt MongoDB
- Kết nối và quản lý MongoDB với Robo 3T
- Cài đặt Mongoose
- Kết nối MongoDB từ Node.js
- Định nghĩa Model
- Tạo các action CRUD với Mongoose model
  - Lấy dữ liệu từ MongoDB
  - Update document
  - Xóa một document
- Tổng kết

Ứng dụng MongoDB vào dự án

- Lưu dữ liệu bài Post vào Database
- Hiển thị danh sách các bài Post
- Hiển thị dữ liệu động với Template engine
- Hiển thị nội dung một Post
- Thêm Fields và Schema
- Tổng kết

Tạo tính năng upload ảnh với Express

- Tổng kết

Tìm hiểu Express Middleware

- Middleware tùy chỉnh
- Tạo và đăng ký Validation middleware
- Tổng kết

Refactoring theo mô hình MVC

- Giới thiệu mô hình MVC
- Tiến hành Refactoring
- Tổng kết

Tạo tính năng đăng ký thành viên

- User Model
- Controller xử lý đăng ký user

Mã hóa mật khẩu

Mongoose Validation

Tạo tính năng đăng nhập

Tổng kết

Xác thực với Express Sessions

Implementing User Sessions

Protect một Pages nào đó với Authentication Middleware

User Logout

Tạo trang 404

Tổng kết

Triển khai web app lên server thật

Cài đặt server

1. Cài đặt NGINX và Git
2. Cài đặt NodeJS và mongoDB
3. Cài đặt PM2

Đưa sourcecode Node.js lên VPS

Quản lý ứng dụng Node.js bằng PM2

Kết nối domain vào vps

Cấu hình Nginx Reverse Proxy Server

Chào tạm biệt

Tài liệu tham khảo

Lời nói đầu

Node.js đang trở thành một xu hướng của giới lập trình back-end. Có rất nhiều ứng dụng lớn của các đại gia đang sử dụng Nodejs. Có thể kể tên như: Paypal, Netflix, LinkedIn...

Mục tiêu của cuốn sách này đó là giúp các bạn bước vào thế giới của Node.js một cách vững chắc nhất. Tức là bạn sẽ hiểu rõ được bản chất, cách xây dựng ứng dụng Nodejs một cách bài bản nhất. Thật không quá khi gọi là "vũ trụ Node.js ". "Vũ trụ Node.js " bắt nguồn từ viên gạch Javascript.

Để khám phá "Vũ trụ Node.js " một cách trơn tru, cuốn sách này sẽ giúp các bạn tìm hiểu "tam trụ" cơ bản của Node.js, đó là Node.js, ExpressJS, và MongoDB.

Mỗi phần trong cuốn sách này sẽ được trình bày thẳng vào vấn đề, kiến thức trọng tâm để tránh mất thời gian vàng ngọc của bạn.

Sau khi bạn đọc xong cuốn sách này, bạn sẽ đủ kỹ năng để tự mình xây dựng một web app bằng Nodejs và triển khai nó trên Internet.

Cách học đúng cách

Cuốn sách này mình chia nhỏ nội dung thành 12 phần, mỗi phần sẽ giới thiệu một chủ đề riêng biệt. Mục đích là để bạn có thể chủ động lịch học, không bị dồn nén quá nhiều.

Với mỗi phần lý thuyết, mình đều có ví dụ minh họa và code luôn vào dự án. Vì vậy, cách học tốt nhất vẫn là vừa học vừa thực hành. Bạn nên **tự mình viết lại từng dòng code** và chạy nó. Đừng copy cả đoạn code trong sách, điều này sẽ làm hạn chế khả năng viết code của bạn, cũng như làm bạn nhiều khi không hiểu vì sao code bị lỗi.

> *Nhớ nhé, đọc đến đâu, tự viết code đến đó, tự build và kiểm tra đoạn code đó chạy đúng không!*

Yêu cầu trình độ

Cuốn sách này mình xây dựng từ những kiến thức nền tảng Node.js từ cơ bản nhất. Nên không cần bạn phải có kiến thức về Node.js.

Tuy nhiên, vì Node.js được xây dựng trên ngôn ngữ Javascript nên sẽ tốt hơn nếu bạn đã có kiến thức căn bản về Javascript. Ngoài ra, bạn cũng cần chút hiểu biết về HTML và CSS để dựng giao diện web.

Liên hệ với tác giả

Nếu có bất kỳ vấn đề gì trong quá trình học, code bị lỗi hoặc không hiểu, các bạn có thể liên hệ với mình qua một trong những hình thức dưới đây:

- Website: https://vntalking.com
- Fanpage: https://facebook.com/vntalking
- Email: support@vntalking.com
- Github: https://github.com/vntalking/nodejs-express-mongodb-co-ban

PHẦN 1
Giới thiệu

Node.js là một JavaScript runtime để chạy Javascript phía server. Có rất nhiều công ty đã sử dụng Node.js để xây dựng cho ứng dụng của họ. Có thể kể một số tên tuổi đình đám như: WalMart, LinkedIn, PayPal, YouTube, Yahoo!, Amazon.com, Netflix, eBay và Reddit.

Trong cuốn sách này, chúng ta sẽ học Node.js kết hợp với Express và mongoDB để xây dựng một blog từ đầu, từ con số 0. Trong quá trình đọc và thực hành theo sách, bạn sẽ hiểu và tự mình xây dựng một ứng dụng riêng từ những kỹ thuật trong cuốn sách này.

Chúng ta sẽ cùng nhau tìm hiểu một loạt những kỹ thuật như xác thực người dùng, validate dữ liệu, bất đồng bộ trong Javascript, Express, MongoDB và template engine.v.v...

Giới thiệu ứng dụng blog sẽ xây dựng

Như mình đã giới thiệu ở trên, để việc học có hiệu quả, thay vì chỉ có những đoạn code mẫu ngắn và không liên quan tới nhau, chúng ta sẽ vừa học vừa xây dựng một ứng dụng hoàn chỉnh. Tiêu chí là "*học đến đâu, ứng dụng đến đó*".

Đây là giao diện của ứng dụng:

Hình 1.1: Template giao diện dùng trong sách

Với trang blog này, người dùng có thể đăng ký tài khoản mới. Sau khi đăng ký xong thì họ có thể về trang chủ, đăng nhập vào blog. Thanh navigation bar sẽ hiển thị những menu khác nhau tùy thuộc vào trạng thái người đã đăng nhập rồi hay chưa. Để xây dựng giao diện, chúng ta sẽ sử dụng EJS template engine.

Sau khi người dùng đăng nhập, thành navigator bar sẽ hiển thị text "LogOut" để họ có thể đăng xuất nếu cần. Ngoài ra, còn thêm một menu "new post" để họ có thể tạo bài viết mới, upload ảnh lên blog. Sau khi họ tạo bài viết xong thì quay trở lại trang chủ, blog sẽ hiển thị danh sách các bài viết đã published.

Thông qua việc thực hành xây dựng ứng dụng blog này, bạn sẽ nắm chắc được kiến thức về Node.js, kết hợp với express và MongoDB.

Node.js là gì?

Trước khi tìm hiểu khái niệm Node.js là gì, bạn cần phải hiểu cơ chế Internet hoạt động như thế nào đã. Khi một người dùng mở trình duyệt, cô ấy vào một website như vntalking.com chẳng hạn. Như vậy là cô ấy đã tạo request tới server, lúc này cô ấy/trình duyệt được coi là một client. Khi client tạo request một server và server phản hồi với nội dung của trang web mà client yêu cầu.

Có một số ngôn ngữ lập trình được thiết kế để viết các ứng dụng cho server như PHP, Ruby, Python, ASP, Java... Nếu trước kia, Javascript được thiết kế để chạy trên các trình duyệt, cung cấp thêm khả năng tương tác của trang web với người dùng. Ví dụ như các menu có hiệu ứng dropdown, hay hiệu ứng tuyết rơi...

Nhưng mọi chuyện đã thay đổi vào năm 2009, khi Node.js ra đời, sử dụng dụng V8 engine làm bộ chạy các mã Javascript. Giờ đây, Javascript đã vượt ra khỏi khuôn khổ của trình duyệt, và cho phép nó chạy trên các server. Như vậy, ngoài các ngôn ngữ dành riêng cho server như PHP, Golang, JAVA... thì Javascript đã trở thành một lựa chọn sáng giá khác.

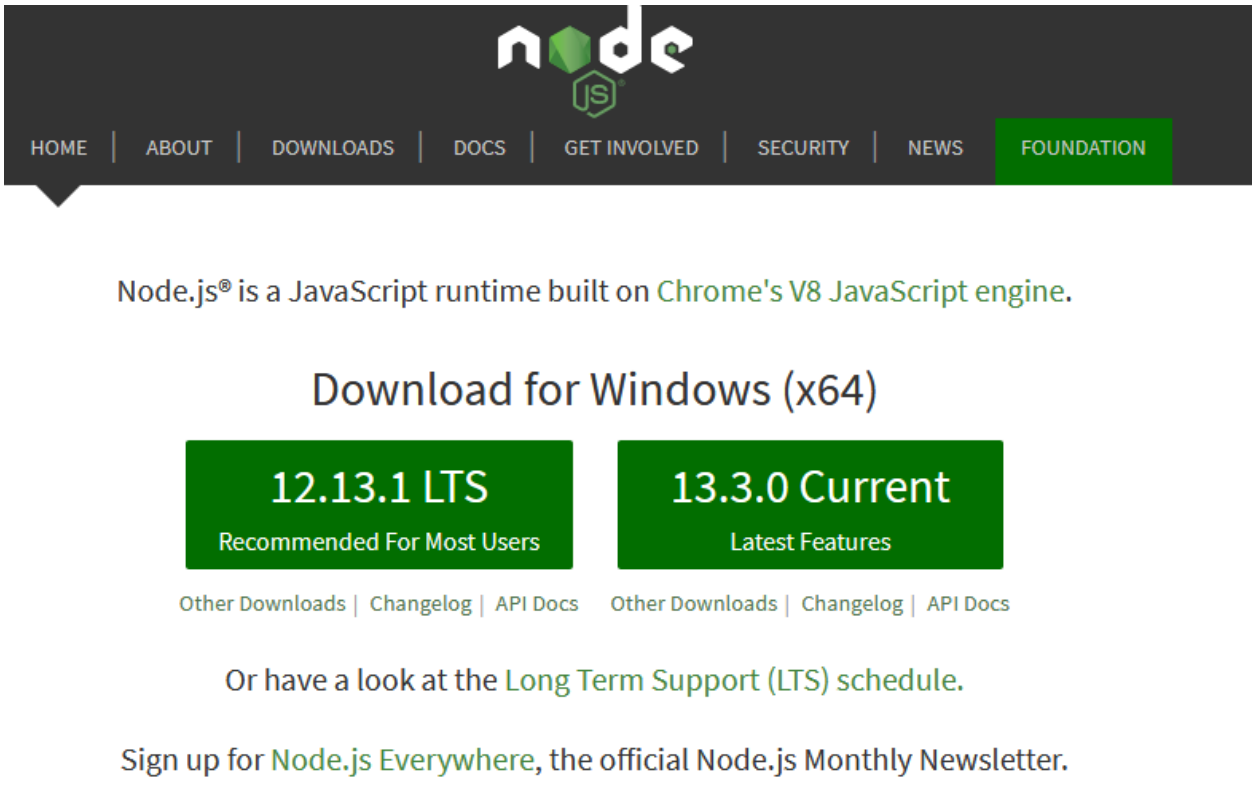
Những lợi ích mà Node.js mang lại cho bạn:

- Đầu tiên, V8 JavaScript engine là một Javascript engine mạnh mẽ, được sử dụng trong trình duyệt Chrome của Google. Điều này sẽ làm ứng dụng của bạn có tốc độ rất nhanh.
- Node.js khuyến khích viết mã kiểu asynchronous (bất đồng bộ) để cải thiện tốc độ ứng dụng, tránh được những vấn đề phát sinh của việc sử dụng đa luồng.
- Thứ 3 đó là Javascript là một ngôn ngữ rất phổ biến, do vậy bạn sẽ thừa hưởng rất nhiều thư viện hay ho mà lại miễn phí.

- Cuối cùng là do Node.js cũng sử dụng javascript, do vậy bạn sẽ tận dụng được những kiến thức đã có từ trước, khi bạn viết ứng dụng dùng Javascript trên trình duyệt. Giờ đây, thay vì phải tìm hiểu thêm một ngôn ngữ mới, bạn chỉ cần biết một mình Javascript là đủ full stack rồi.

Cài đặt Node.js

Để cài đặt Node.js, bạn vào trang chủ nodejs.org (hình 1.4) và tải phiên bản tương ứng với hệ điều hành trên máy tính của bạn.



Hình 1.2: Download node.js

Việc cài đặt diễn ra cũng đơn giản. Nếu máy bạn dùng window thì cài đặt như mọi phần mềm khác thôi. Bạn có thể tham khảo chi tiết các cài đặt chi tiết tại đây.

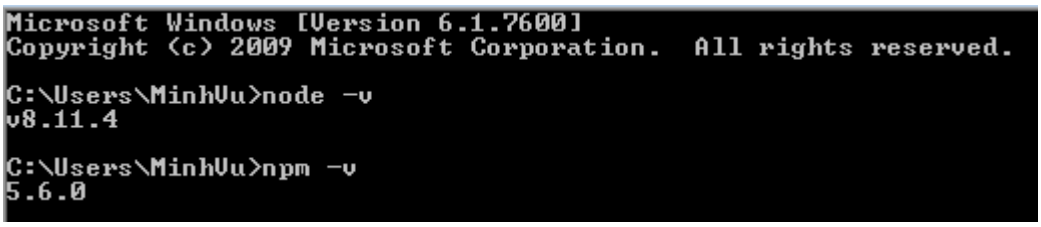
Sau khi cài xong, bạn có thể kiểm tra version bằng lệnh sau:

```
node -v
```

Ngoài ra, khi cài đặt Node.js, bạn sẽ được khuyến mãi cả NPM nữa.

```
npm -v
```

Kết quả hiện ra như hình 1.5 là được.



Hình 1.3: kiểm tra phiên bản node.js

Tạo server Nodejs đầu tiên

Chúng ta sẽ tạo một server đơn giản nhất bằng Node.js để các bạn dễ hiểu hơn về cách một client tạo request tới server và server phản hồi lại request đó như thế nào.

Bạn có thể sử dụng bất kỳ trình soạn thảo code để viết code. Như mình thì vẫn đề xuất các bạn sử dụng Visual Studio Code. Giờ mình sẽ tạo một thư mục và dùng Visual Studio Code (từ giờ mình sẽ viết tắt là VS) để mở thư mục ấy. Các bạn tạo mới một file Javascript, đặt tên là index.js. Dưới đây là đoạn code để tạo một http server đơn giản:

```
const http = require('http')

const server = http.createServer((req, res) => {
    console.log(req.url)
    res.end('VNTALKING: Xin chào Node.js')
})
server.listen(3000)
```

Giải thích code

```
const http = require('http')
```

Hàm require có tác dụng là import một module vào tệp bạn đang xử lý. Hàm này trong Node.js giống như hàm import hay include trong các ngôn ngữ khác. Tham số đầu vào của hàm require là tên của module được định nghĩa bằng từ khóa export và trả về package đó.

Quay lại đoạn code trên, mình require một module có tên là http và gán nó cho biến cũng có tên là http (mình đặt trùng để cho dễ nhận biết biến này là package nào thôi, chứ bạn có thể đặt biến bất kỳ tên nào bạn muốn).

http là một module được tích hợp sẵn trong Node.js, dùng để cung cấp các phương thức tương tác với server như GET, POST, UPDATE...

```
const server = http.createServer(...)
```

Đoạn code này giúp chúng ta khởi tạo một server. Hàm này có tham số là một hàm số với hai tham số: request và response.

```
const server = http.createServer((req, res) => {
    console.log(req.url)
    res.end('VNTALKING: Xin chào Node.js')
})
```

Thực ra hàm được truyền vào *createServer()* cũng không phải là một khái niệm mới mẻ đâu. Đó chính là callback. Hàm callback này sẽ được gọi khi việc khởi tạo server hoàn thành. Trong đó có hai tham số, req (request) là đối tượng mà nó nhận được từ client (trình duyệt chẳng hạn), còn res (response) là đối tượng mà server sẽ trả về cho trình duyệt. Chúng ta có thể làm bất cứ điều gì với hai đối tượng này. Như trong ví dụ trên, mình đơn giản chỉ là in ra log đối tượng req và trả lại trình duyệt một dòng thông báo: "*VNTALKING: Xin chào Node.js*".

```
server.listen(3000)
```

Mỗi một server đều phải lắng một port bất kỳ trong dải từ 1 ->65535. Bạn có thể tùy chọn thoải mái. Tất nhiên nên trừ những port thông dụng đang được sử dụng bởi hệ thống như: 80, 22, 23, 25... Nếu bạn hỏi port là gì thì mình giải thích đơn giản là port là một gateway kết nối ra ngoài hoặc giữa các ứng dụng trên server được sử dụng bởi một ứng dụng cụ thể. Nếu nhiều ứng dụng cùng chạy trên server thì mỗi ứng dụng sẽ sử dụng một port khác nhau.

Trong ví dụ này, ứng dụng sẽ sử dụng port 3000 và mọi request tới port 3000 đều sẽ trả về dòng thông báo "*VNTALKING: Xin chào Node.js* "

Cách chạy ứng dụng

Để thực thi ứng dụng, chúng ta ta mở cửa sổ lệnh (trong ubuntu gọi là terminal, còn trong window gọi command Prompt), di chuyển con trỏ tới thư mục mã nguồn

```
Microsoft Windows [Version 6.1.7600]

Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

E:\github\nodejs-express-mongodb-co-ban>node index.js
```

Giờ bạn mở trình duyệt và truy cập vào địa chỉ *http://localhost:3000* để xem kết quả.

Bởi vì chúng ta đang chạy ứng dụng trên chính máy tính của mình nên chỉ có thể truy cập qua localhost. Phần cuối cuốn sách, mình sẽ hướng dẫn các bạn triển khai ứng dụng để publish lên internet, và bất kỳ ai, bất kỳ đâu cũng có thể truy cập được.

Hiểu hơn về request và response

Ứng dụng của chúng ta hiện tại mới chỉ phản hồi dòng chữ "*VNTALKING: Xin chào Node.js* " cho client, cho dù đằng sau URL:localhost:3000/xxx là gì đi nữa. Để có những xử lý phản hồi khác nhau cho những URL khác nhau, chúng ta cần có thể xử lý đơn giản như sau:

```
const server = http.createServer((req, res) => {
if (req.url === '/about')
        res.end('The about page')
elseif (req.url === '/contact')
        res.end('The contact page')
elseif (req.url === '/')
        res.end('The home page')
else {
        res.writeHead(404)
        res.end('page not found')
    }
})

server.listen(3000)
```

Giờ bạn chạy lại server và thử trên trình duyệt những URL sau:

- http://localhost:3000
- http://localhost:3000/about
- http://localhost:3000/contact

Tuy nhiên, đây chỉ là mình làm ví dụ đơn giản để bạn hiểu hơn về request và response thôi. Còn sau này, với dự án thực tế thì người ta không dùng kiểu *if-else* thế này đâu. Lúc đó bạn sẽ gặp khái niệm Router. Sử dụng router để điều hướng trang web.

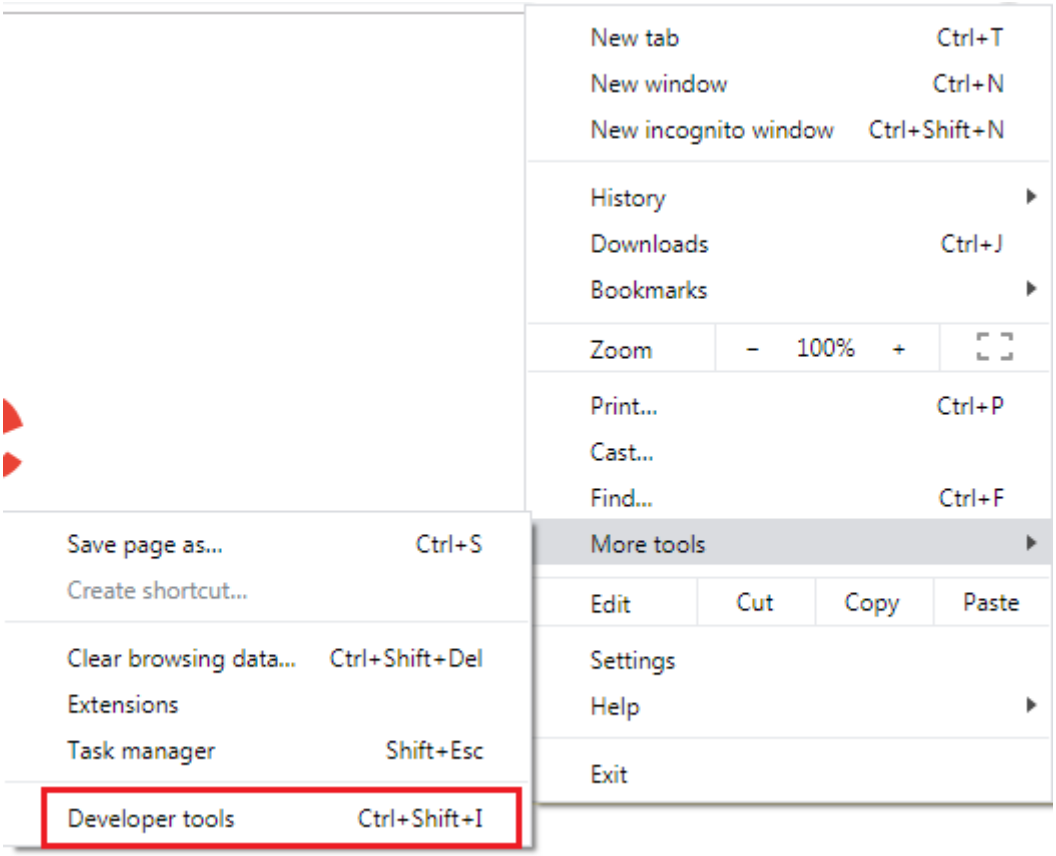
Tất nhiên, chung ta sẽ tìm hiểu router ở phần sau của cuốn sách.

Mình muốn rõ hơn về đoạn code này:

```
res.writeHead(404)
```

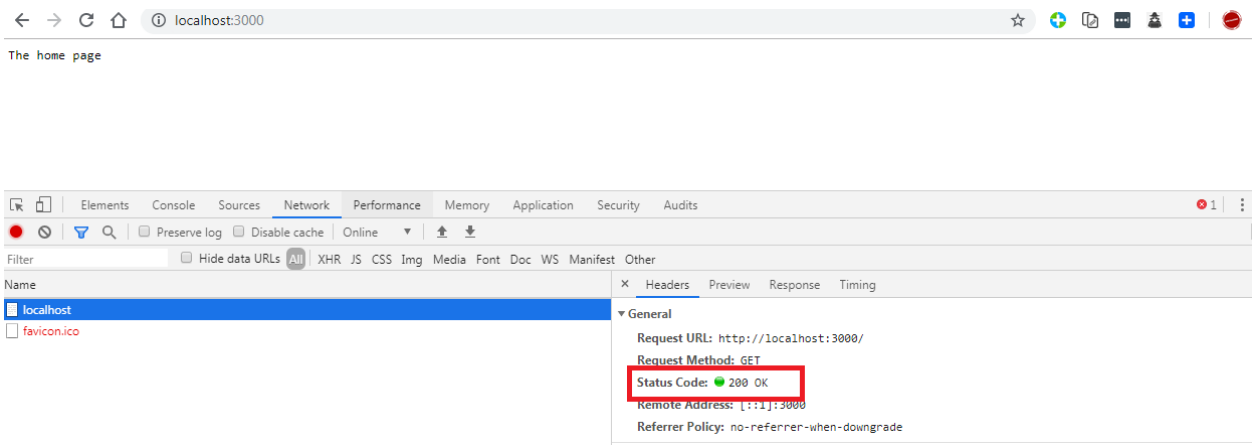
Khi có một request tới server và server phản hồi lại request đó. Với request valid, server sẽ phản hồi mã *200-OK*. Còn nếu gặp lỗi thì tùy vào kiểu lỗi mà trả về mã lỗi tương ứng. Mã lỗi được quy định theo chuẩn của HTTP. Các bạn có thể tham khảo ở đây.

Các bạn có thể kiểm tra mã trả về từ server bằng cách: Mở trình duyệt (Chrome chẳng hạn), tìm đến Developer tools như hình vẽ. Sau đó chọn tab Network



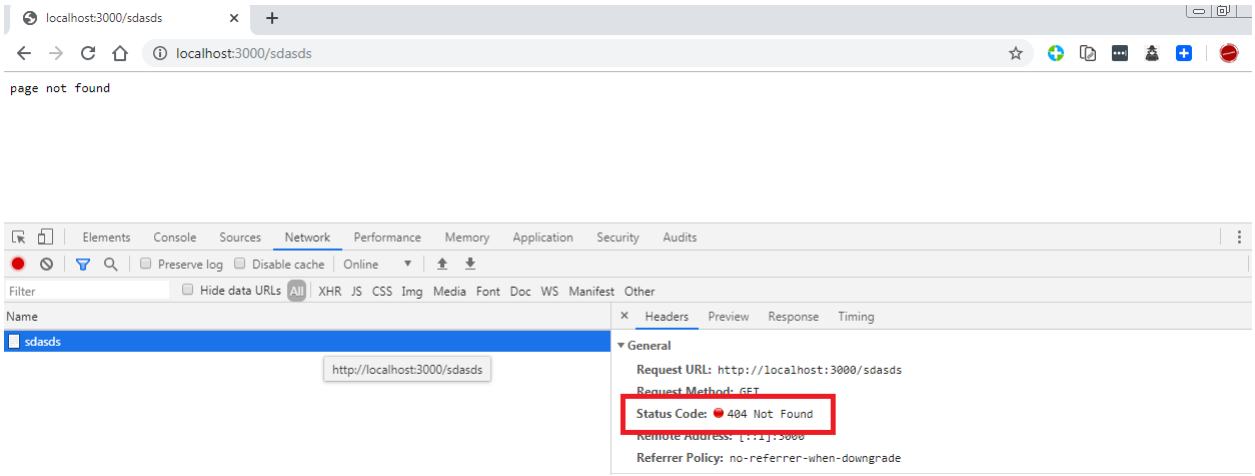
Hình 1.4: Developer tools trong trình duyệt Chrome

Sau đó bạn truy cập vào một URL chẳng hạn: *http://localhost:3000*



Hình 1.5: Kiểm tra status code 200 - OK

Còn nếu bạn truy cập một URL không tồn tại, như đoạn code trên thì sẽ trả về lỗi 404



Hình 1.6: Status 404 - Not Found

Tóm lại, với đoạn code trên, chúng ta đang viết một hàm Javascript đơn lẻ để lắng nghe những request từ client như browser, mobile hay bất kể client nào khác sử dụng API. Chúng ta gọi hàm này là một request handler (xử lý request). Cứ khi có request đến, hàm này sẽ tìm kiếm và xử lý rồi trả kết quả về cho client.

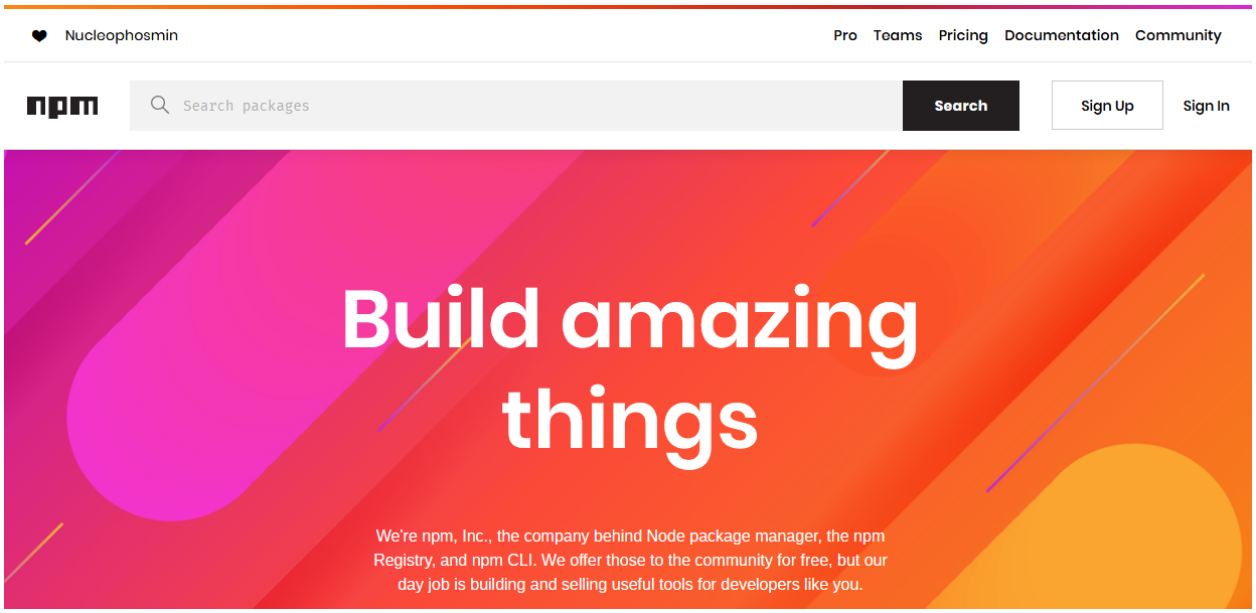
Tất cả ứng dụng Node.js đều hoạt động theo cơ chế như vậy. Một single request handler để xử lý tất cả các request và phản hồi các request đó. Với các ứng dụng nhỏ thì điều này có vẻ đơn giản. Nhưng với ứng dụng lớn, phải xử lý rất nhiều request, chưa kể để có kết quả phản hồi thì cũng mất nhiều thời gian như: render HTML, upload/download ảnh... Để giải quyết những vấn đề này chúng ta cần sự trợ giúp của công cụ. Phần tiếp theo của cuốn sách, chúng ta sẽ cùng nhau xem ExpressJS giải quyết vấn đề này như thế nào.

PHẦN 2

Giới thiệu về NPM và Express

Trong đoạn code minh họa của phần 1, chúng ta hoàn toàn sử dụng những package build -in của Node.js, ví dụ: http, fs... Trong khi đó, có rất nhiều packages được phát triển bởi các nhà phát triển bên thứ ba mà bạn có thể sử dụng cho dự án của bạn. Những packages này được lưu trữ trên website npmjs.com (hình 2.1). Trang web npmjs.com có thể hiểu như một market các packages, nơi mà bạn có thể tha hồ lựa chọn package nào phù hợp với dự án để sử dụng.

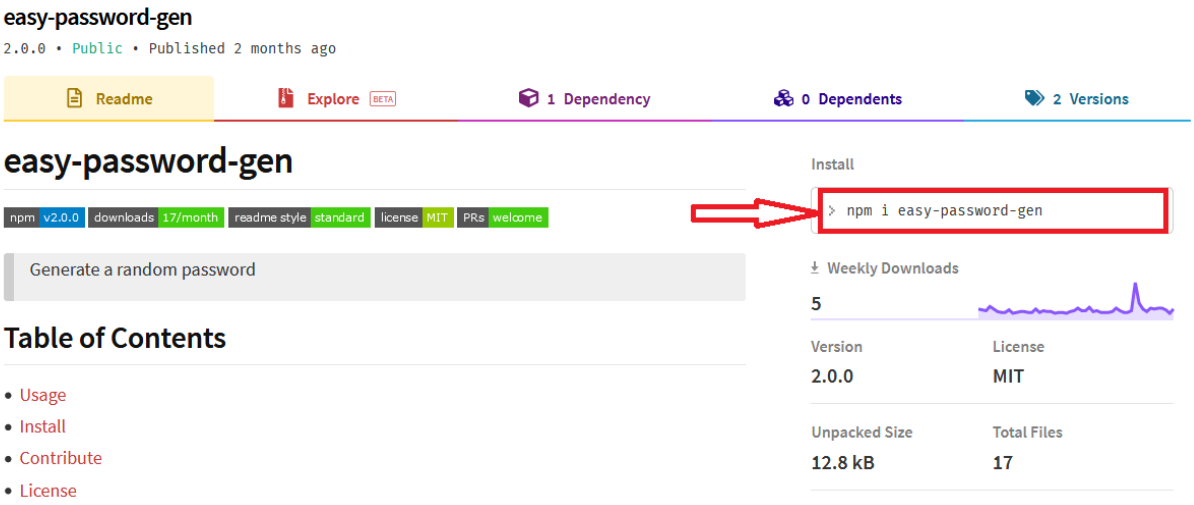
Còn NPM (Node Package Manager) là phần mềm được cài đặt cùng với Node.js, nó được dùng để quản lý các packages mà bạn download từ npmjs.com.



Hình 2.1: giao diện trang web npmjs.com

Cài đặt Custom Package với NPM

Để cài đặt một package từ npmjs.com, bạn tìm kiếm package từ hộp search, sau khi chọn được một package có vẻ ưng ý. Bạn chọn package đó và nhìn các thông tin một package như hình bên dưới.



Hình 2.2: Trang thông tin một module trên npmjs.com

Ở đây nhìn vào thông tin cách cài đặt, họ có ghi rõ lệnh cài đặt package này bằng npm: Ví dụ lệnh cài đặt package easy-password-gen như hình 2.2

```
npm i easy-password-gen
```

Trong cuốn sách này, mình giới thiệu với các bạn một package quan trọng và cũng rất phổ biến trong thế giới Node.js, đó là ExpressJS. Bởi vì cú pháp API của Node.js có thể dài dòng, khó hiểu và giới hạn về tính năng, nên người ta bắt đầu nghĩ tới một framework giúp họ đơn giản hóa quá trình viết ứng dụng.

ExpressJS là một Framework nhỏ, nhưng linh hoạt được xây dựng trên nền tảng của Nodejs. Nó cung cấp các tính năng mạnh mẽ để phát triển ứng dụng web. Chúng ta sẽ thấy ExpressJS làm đơn giản hóa các API của Node.js, cách tổ chức dự án theo mô hình MVC với middleware và routing. Ngoài ra còn hàng tá những hàm Utils rất hữu ích để xử lý HTTP và render HTML.

Để cài đặt ExpressJS, các bạn gõ:

```
npm install express
```

Trước khi bạn có thể chạy câu lệnh cài đặt express như trên, dự án của bạn cần phải tạo trước *package.json*. Đây là file cấu hình của dự án, lưu trữ tất cả các thông tin về dự án như: tên dự án, version, các packages được sử dụng trong dự án... Nếu bạn chạy lệnh npm install express mà không có package.json thì sẽ gặp lỗi: "no such file or directory, open … package.json".

Để generate ra file *package.json*, bạn di chuyển con trỏ vào thư mục dự án, gõ lệnh:

```
npm init
```

Sau đó bạn điền các thông tin như trong hướng dẫn của trình thuật sĩ. Sau khi hoàn thành, bạn gõ "Yes" để xác nhận tạo *package.json*

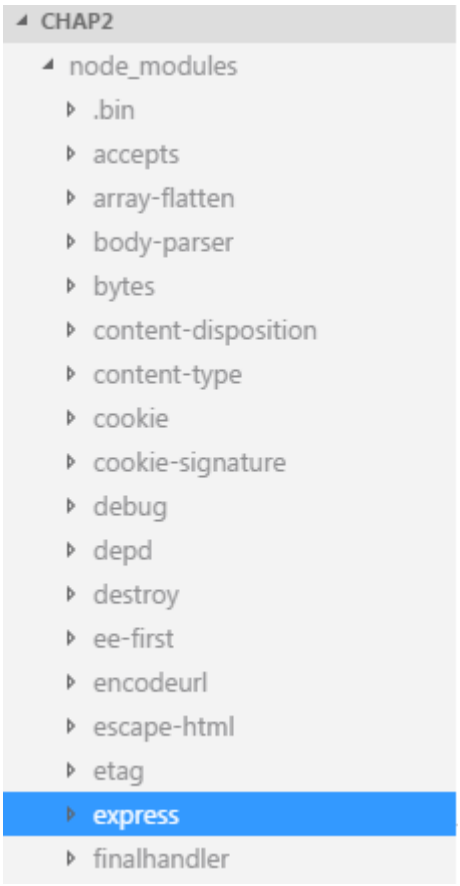
Ví dụ đây là *package.json* mà mình tạo cho dự án cho phần 2 của cuốn sách này.

```
{
      "name": "chap2",
      "version": "1.0.0",
      "description": "Code minh họa cho chap 2",
      "main": "index.js",
      "scripts": {
      "test": "echo \"Error: no test specified\" && exit 1"
        },
      "author": "VNTALKING.COM",
      "license": "ISC"
}
```

Tiếp theo, bạn gõ lệnh cài đặt express: *npm install express*. Khi việc cài đặt kết thúc, bạn sẽ thấy *package.json* thêm một dependencies là *express* như bên dưới.

```
{
      "name": "chap2",
      "version": "1.0.0",
      "description": "Code minh họa cho chap 2",
      "main": "index.js",
      "scripts": {
            "test": "echo \"Error: no test specified\" && exit 1"
        },
      "author": "VNTALKING.COM",
      "license": "ISC",
      "dependencies": {
            "express": "^4.17.1"
        }
}
```

Thẻ *dependencies* sẽ chứa tất cả tên của các packages cùng với version lúc cài đặt. Tại thời điểm của cuốn sách này thì version mới nhất của express là 4.17.1. Ngoài ra, toàn bộ mã nguồn của package express sau khi được cài đặt sẽ được lưu tại thư mục *node_modules* (Hình 2.3)



Hình 2.3: nơi chứa mã nguồn expressJS trong dự án

Bạn mở thư mục *node_modules* thì thấy ngoài thư mục có tên express thì còn có rất nhiều thư mục khác nữa. Mặc dù lúc trước bạn mới chỉ cài mỗi một mình ExpressJS. Nguyên nhân là do ExpressJS có sử dụng các dependencies khác, nên khi cài đặt ExpressJS thì npm cũng sẽ tải và cài đặt chúng luôn. Để biết được express có những dependencies nào thì bạn lại vào package của ExpressJS là sẽ thấy.

Tuy nhiên, bạn không nên sửa bất kì dòng code trong thư mục *node_modules*. Vì đây là mã nguồn gốc của package, bạn sẽ không biết được sẽ phát sinh lỗi gì nếu sửa chúng đâu.

Giới thiệu Express

Ở phần này, mình sẽ giới thiệu nhiều hơn về ExpressJS, tại sao nó lại giúp phát triển ứng dụng Node.js đơn giản và nhanh hơn.

Đầu tiên, chúng ta sẽ import *express* vào ứng dụng bằng cách thêm đoạn code sau vào *index.js*:

```
const express = require('express')
```

Để có thể so sánh và kết luận tại sao ExpressJS lại giúp phát triển ứng dụng nhanh hơn, chúng ta sẽ viết lại những gì ở phần 1 cuốn sách, nhưng với sự trợ giúp của ExpressJS.

```
const http = require('http');

const server = http.createServer((request, response) => {
    console.log("xin chao VNTALKING.COM");
})

server.listen(3000)
```

Nhưng với ExpressJS thì bạn chỉ cần như sau:

```
const express = require('express')
const app = express()
app.listen(3000, () =>{
      console.log("App listening on port 3000")
})
```

Bạn thấy đấy, chúng ta không cần phải import thêm bất cứ package nào cả, cũng không cần phải viết đoạn xử lý request hay response nữa. Về bản chất thì ExpressJS lo làm giúp việc import các package cần thiết cho chúng ta rồi, chúng ta không cần phải làm nữa. Tuy nhiên, nếu chỉ đơn giản là rút gọn code thì ExpressJS lại quá bình thường phải không? ExpressJS không đơn giản như vậy đâu.

Xử lý request với Express

ExpressJS cho phép xử lý linh hoạt hơn với các request "GET" hay "POST" từ trình duyệt. Dưới đây là minh họa cách ứng dụng trả về một đối tượng JSON khi nhận được một request:

```
const express = require('express')
const app = express()
app.listen(3000, () =>{
      console.log("App listening on port 3000")
})

app.get("/", (request, response) =>{
    response.json({
        name: "Duong Anh Son",
        website: "VNTALKING.COM"
    })
})
```

Trong đoạn code trên, bạn có thể thấy là ExpressJS cung cấp nhiều lựa chọn hơn để bạn có thể phản hồi lại những request từ trình duyệt. Ví dụ, chúng ta trả về đối tượng JSON cho trình duyệt bằng hàm *response.json()*. Nhờ điều này mà chúng ta dễ dàng xây dựng các REST API với Node.js.

Ngoài ra, chúng ta có thể định nghĩa những routes cụ thể, ví dụ như:

```
app.get("/about", (request, response) =>{
    response.json({
        name: "Duong Anh Son",
        website: "VNTALKING.COM"
    })
})
```

Cái này người ta gọi là Routing. Tức hiểu nôm na là chúng ta quy ước phần xử lý với một URL cụ thể nào đó. Như ở phần 1, để xử lý những URL riêng biệt, chúng ta phải nhờ đến *if-else* trong một hàm xử lý request lớn.

```
const server = http.createServer((req, res) => {
if (req.url === '/about')
        res.end('The about page')
else if (req.url === '/contact')
        res.end('The contact page')
else if (req.url === '/')
        res.end('The home page')
else {
        res.writeHead(404)
        res.end('page not found')
    }
})
```

Với ExpressJS thì bạn có thể tách riêng biệt cho từng URL riêng biệt. Điều này giúp tăng khả năng module hóa hay khả năng bảo trì dự án.

Với đoạn code thì express có thể tách như sau:

```
app.get("/about", (request, response) =>{
    res.send('The about page')
})

app.get("/contact", (req, res) =>{
    res.send('The contact page')
})

app.get("/contact", (req, res) =>{
```

```
    res.send('The contact page')
})

app.get("/", (req, res) =>{
    res.send('The home page')
})

app.get('*', function(req, res){
    res.header(404)
    res.send('page not found')
});
```

Bất đồng bộ với Call Back

Trong những đoạn code trên, bạn đã bắt gặp việc sử dụng các hàm call back. Call Back là một khái niệm vô cùng quan trọng trong Node.js, được sử dụng để xử lý các hàm bất đồng bộ.

Đồng bộ tức là các hàm viết sau sẽ đợi hàm viết trước hoàn thành. Ví dụ như trong PHP thì các hàm sẽ lần lượt thực hiện:

```
Task 1 -> Task 2 -> Task 3 -> Task 4 -> Completion
```

Như vậy, nếu task 1 thực hiện quá lâu hoặc bị block vì lý do nào đó thì hàm task sẽ không thể thực hiện. Với Node.js thì hơi khác, nó cho phép các hàm được thực hiện bất đồng bộ, tức là hàm sau không cần phải đợi hàm trước thực hiện xong. Tất cả sẽ được cho vào một queue, cái nào xong trước thì thông báo.

Ví dụ đoạn code sau trong Node.js

```
//Task1
app.get('/', (req, res) => {
      // query database
})
//Task2
app.get('/about', (req, res) => {
    res.sendFile(path.resolve(__dirname, 'about.html'))
})
```

Khi bạn chạy chương trình, cả 2 task đều sẽ được thực hiện cùng một thời điểm. Nếu task 1 mà cần nhiều thời gian, mà task 2 lại đơn giản chỉ gửi một file tĩnh cho trình duyệt thì khả năng task sẽ thực hiện xong trước. Cả 2 task cùng bắt đầu một thời điểm nhưng không có nghĩa chúng sẽ thực hiện đồng thời. Khi một task cần thời gian thì task sẽ được thực hiện. Và khi một task thực hiện xong thì sử dụng call back để thông báo cho chương trình biết.

Về lý thuyết thì chương trình bất đồng bộ sẽ chạy nhanh hơn vì tận dụng được thời gian nhàn rỗi của CPU.

Trả về một file html cho client

Để phàn hồi lại client một file html trong ExpressJS, bạn sử dụng *sendFile* api. Cách làm như sau:

```
// called when request to '/about' comes in
app.get('/about', (req, res) => {
      res.sendFile(path.resolve(__dirname, 'about.html'))
})

//called when request to '/contact' comes
app.get('/contact', (req, res) => {
      res.sendFile(path.resolve(__dirname, 'contact.html'))
})
```

Như ví dụ trên, máy chủ node.js sẽ trả về cho client tệp about.html khi nó request tới */about*

Trong các ví dụ vừa qua mình sử dụng hàm app.get('...') để xử lý các yêu cầu HTTP GET. GET là loại request được quy ước dùng để truy cập vào server để lấy thông tin, tài nguyên, hình ảnh... mà không thay đổi nội dung phía server. Ngoài GET ra, HTTP còn định nghĩa các loại request khác như POST, DELETE, UPDATE. Chúng ta sẽ tìm hiểu chúng dần dần trong các phần sau của cuốn sách.

Trả về static resource (image, css, js...) cho client

Với các ứng dụng web, việc phải sử dụng image, css, js là điều gần như bắt buộc để định dạng, trang trí giao diện cho ứng dụng. Nhưng mặc định thì tất cả các tài nguyên trên máy chủ đều được bảo mật, nếu không có quyền thì client không thể truy cập được, hoặc ít nhất phải truy cập qua các router.

Có một giải pháp phổ biến là chúng ta sẽ đưa hết các file static (image, css, js...) vào một thư mục, gọi là public. Thư mục public có thể truy cập từ bất kỳ đâu, ai cũng có thể truy cập được, chỉ biết đường dẫn của nó.

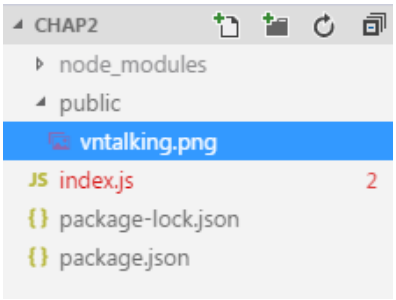
Trong Express, bạn dễ dàng định nghĩa một thư mục public như thế.

```
const express = require('express')
const app = express()
const path = require('path')
```

```
app.use(express.static('public'))

app.listen(3000, () => {
    console.log("App listening on port 3000")
})
```

Giờ làm ví dụ minh họa, mình đưa một tệp ảnh vntalking.png vào thư mục public như dưới đây.



Hình 2.4: Thư mục public dùng để chứa static resource

Giờ bạn có thể truy cập trực tiếp trên trình duyệt *http://localhost:3000/vntalking.png* . Bạn thấy đấy, chúng ta không cần phải định nghĩa router để truy cập vào file ảnh đó.

Bạn có thể sử dụng các tệp trong thư mục public trong các file html mà không phải thông qua router.

```
//about.html
<h1>Đây là màn hình thông tin về VNTALKING.COM</h1>
<img src="/vntalking.png" style="width: 30%"/>
```

Và thử vào trình duyệt kiểm tra thử.

Hình 2.5: Minh họa truy cập static resource

Cám ơn bạn đã quan tâm tới sách học lập trình Node.js thật đơn giản.

Mời bạn truy cập VNTALKING.COM để đặt mua bản sách đầy đủ nhé. Đặc biệt ưu đãi cho các bạn học sinh - sinh viên.