TRƯỜNG ĐẠI HỌC CÔNG NGHỆ SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO THỰC TẬP TỐT NGHIỆP

Đề tài: Tìm hiểu Node.Js và framework Express.

Sinh viên thực hiện:

Vòng Quyền Minh, DH51703728

Nguyễn Hoàng Huy, DH51703482

Lớp:

D17\_TH07

Giảng viên hướng dẫn: GV.ThS Lê Triệu Ngọc Đức

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC TẬP TỐT NGHIỆP

Đề tài: Tìm hiểu Node.Js và framework Express

Sinh viên thực hiện:

Vòng Quyền Minh, DH51703728

Nguyễn Hoàng Huy, DH51703482

Lớp:

D17\_TH07

Giảng viên hướng dẫn: GV.ThS Lê Triệu Ngọc Đức

Phụ lục.

1. Giới thiệu chung về Node.js 1

2. Node.js frameworks và tools. 3

3. Lịch sử phát triển của Node.js 4

4. Cài đặt Node.js. 5

5. Đọc biến môi trường từ Node.js. 6

6. Tìm hiểu lập trình động bộ và lập trình bất đồng bộ. 6

7. Xử lý bất đồng bộ với Callback 8

8. Xử lý bất động bộ với Promise. 9

9. Xử lý bất đồng bộ với Async và Await trong Javascript. 11

10. Tìm hiểu về Node.js HTTP module. 13

10.1. Tạo request đến server. 14

11. Tìm hiểu về Node.js File System module. 16

11.1. Tạo file trong File System module. 17

11.2. Cập nhật file trong File System module. 18

11.3. Xóa file trong File System module. 19

11.4. Đổi tên file trong File System module. 19

12. Tìm hiểu về Node.js URL module. 19

13. Event module và đối tượng EventEmitter. 20

14. Giới thiệu Express framework trong Node.js. 21

14.1. Tổng quan về Express. 21

14.2. Cài đặt Express. 22

14.3. Cấu trúc của Express. 23

14.4. Cấu trúc của nodejs vô cùng đơn giản : 23

14.5. Tìm hiểu về Router trong Express. 24

14.6. Tìm hiểu về các method của Router. 24

14.7. Tìm hiểu về Middleware trong Expressjs 25

14.8. Application-Level middleware 25

14.9. Router-level middleware 27

14.10. Error-handling middleware 27

14.11. Built-in middleware 28

14.12. Third-party middleware 29

15. Xây dựng website thương mại điện tử sử dụng Node.js Express. 29

15.1. Giới thiệu chung. 29

15.2. Công nghệ khác. 29

15.3. Cơ sở dữ liệu. 31

15.4. Giao diện website. 32

15.5. Giao diện đăng nhập. 32

15.6. Giao diện đăng ký. 33

15.7. Giao diện giỏ hàng. 33

15.8. Giao diện xem đơn hàng đã đặt. 34

15.9. Source Code. 34

1. Giới thiệu chung về Node.js

Node.js là một môi trường runtime cho Javascript, mã nguồn mở và đa nền tảng. Node.js là một công cụ phổ biến cho hầu hết mọi loại project.

Node.js sử dụng Javascript V8, phần cốt lõi của Google Chrome, điều này giúp Node.js hoạt động vô cùng tốt.

Một ứng dụng Node.js chỉ thực thi trong một process duy nhất, mà không cần phải tạo ra một threads cho mỗi rrequest. Node.js cung cấp một tập bất đồng bộ I/O cơ bản trong thư viên chuẩn, thứ giúp Javascript có thể ngăn chặn code khỏi cơ chế blocking, thư viện viết cho Node.js được viết theo cơ chế non-blocking, điều này giúp cho cơ chế blocking không còn là mặc định nữa như trong các ngôn ngữ khác.

Khi Node.js thực hiện tác vụ I/O, như đọc dữ liệu từ network, truy cập database hoặc file hệ thống, thay vì thực hiện cơ chế blocking, mất công chở và tốn RAM, Node.js sẽ tiếp tục thực thị các công việc khác, khi có kết quả, phản hổi được trả về từ tác vụ I/O, Node.js sẽ quay trở lại tiến trình đó và thực thi công việc còn dang dở.

Điều này cho phép Node.js có thể xử lý đồng thời hàng ngìn kết nối chỉ trong một server mà không gặp bất cứ khó khăn nào. Node.js có một lợi thể độc nhất bởi vì có hàng triệu Front-end developer sử dụng Javascript và bây giờ họ có thể viết ở phía server-side mà không cần phải học một ngôn ngữ mới.

Dưới đây là một ví dụ cơ bản về Node.js, tên file app.js:



Dòng code đầu tiền thực hiện việc thêm module ‘http’.

Phương thức createServer() của ‘http’ giúp ta tạo và trả về một HTTP server.

Server được thiết lập chạy trên port và hostname được chỉ định. Khi server sẵn sàng, hàm callback được gọi, trong trường hợp của ví dụ callback sẽ xuất ra câu thông báo server đã được chạy trên địa chỉ mà ta thiết lập.

Bất cứ khi nào một request được gửi đến, sự kiện request sẽ được gọi, cung cấp cho ta 2 objects: request (http.IncommingMessage object) và response (http.ServerResponse object). 2 object này vô cùng quan trọng để ta có thể xử lý HTTP.

Object request sẽ cho phép ta truy cập vào chi tiết của request, bao gồm request data và request header trong ví dụ trên ta chưa sử dụng đến.

Object response cho phép ta trả dữ liệu về cho request đến.

Trong ví dụ trên ta đã dùng:

res.statusCode = 200;

Câu lệnh này gán gia trị 200 cho thuộc tính ‘statusCode’ đại diện cho response thành công.

Ngoài ra còn có:

Res.setHeader(“Content-Type”, “text/plain”)

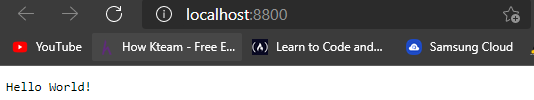
Res.end(“Hello World!\n”);

Giúp ta thiết lập loại nội dung mà response trả về trong ví dụ là ‘text/plain’ và kết thúc response cùng với nội dung trả về được truyền vào như là một tham số của phương thức end().

Để thực thị node.js file ta dùng cú pháp sau trong command line hoặc terminal của editor mà ta sử dụng:

‘node app.js’

Kết quả thu được khi truy cập <http://localhost:8800/>



1. Node.js frameworks và tools.

Node.js là một nền tảng cấp thấp. Để có thể đơn giản hóa và dễ sử dụng cho developer, có hàng nghìn thư viện được xây dựng cho Node.js bởi cộng đồng.

Một số thư viện được xây dựng và trờ nên vô cùng nổi tiếng, và dưới đây là danh sách một số thư viện đáng chú ý:

* AdonisJs: là framework MVC cho Node.js.
* Egg.js: dùng đề xây dựng ứng dụng và framework, giúp giảm thiểu chi phí phát triền và bảo trì.
* Express: Cung cấp công cụ vô cùng mạnh mẽ để tạo webserver.
* Fastify: cung cấp công cụ phát triển webserver, hiệu suất cao, dễ mở rộng.
* Gatsby: Phát triển dự trên React bởi GraphQL, công cụng để tạo website và app. Là một framework vô cùng tốt để build một portfolio hoặc blog hoặc cửa hàng thương mạng điện tử có lượng truy cập cao hoặc trang chủ cho công ty.
* Nest.js: là một framework viết bằng Typescript để xây dựng server-side app hiệu quả và dễ mở rộng. có thể sử dụng được Javascript và cả Typescript.
* Next.js: là React framework, có hộ trợ Typescript.
* Socket.io: là một công cụ giap tiếp real-time.

1. Lịch sử phát triển của Node.js

Javascript là một ngôn ngữ lập trình được sáng tạo bởi Netscape đóng vai trò như là một công cụ scripting để điều khiển web page trên trình duyệt của họ Netscape Navigator.

Một phần trong mô hình kinh doanh của Netscape là bán Web Server, thứ mà bao gồm một môi trường được gọi là Netscape LiveWire cho phép tạo các trang động bằng cách sử dụng Javascript. Tiếc thay là nó không quá thành công và server-side Javascript đã không mấy nối tiếng lúc bấy giờ cho đến khi gần đây, Node.js được giới thiệu,

Một trong những yếu tố dẫn đến thành công của Node.js đó chính là thời điểm mà Node.js được phát hành. Chỉ trong vài nắm trước đây, JS bắt đầu được biết đến như là một ngôn ngữ thật sự, và nhờ vào thời kỳ ứng dụng ‘Web 2.0’ (như Flicks, Gmail,…) những ứng dụng này đã cho thế giới thấy rằng thế nào là một trải nghiệm công nghệ hiện đại.

JS engines cụng trở nên tốt hơn khi các trình duyệt đua nhau phát hành và mong muốn đem đến cho người dùng một trải nghiệm mượt mà. Các đội phát triển trình duyệt thì cố gằng tìm ra cách để hộ trợ và tìm các cách đểb Js có thể chạy mượt và nhanh hơn. Engine mà Node.js sử dụng V8 cụng từ đó mà ngày càng phát triển và trở nên tốt hơn nhờ vào sự cạnh tranh này.

1. Cài đặt Node.js.

Tải bản cài đặt trực tiếp từ trang chủ https://nodejs.org/en/download/

Phiên bản mới nhất hiện tại là 15.12.0 và phiên bản khuyên dùng (LTS) là 14.16.0

Khi cài đặt node.js tiến trình cài đặt sẽ bao gồm luôn cả npm (Node package manager).

Kiến thức cần có trước khi sử dụng Node.js

Node.js khuyên người dùng nên có những kiến thức căn bản về Javascript bao gồm:

* Lexcial Structure.
* Expression
* Types
* Vaiables
* Functions
* this
* Arrow Functions
* Loops
* Scopes
* Arrays
* Template Literals
* Strict Mode
* Semicolons
* ECMAScript 6, 2016, 2017

Ngoài ra kiến thức về lập trình bất đồng bộ cũng là mấu chốt để sử dụng Node.js:

* Asynchronous programming and callbacks
* Timer
* Promises
* Async and Await
* Closures
* The Event Loo

1. Đọc biến môi trường từ Node.js.

Module ‘process’ của Node.js cung cấp thuộc tính ‘env’ cho phép ta truy cập và sử dụng các biến được thiết lập tại thời điểm mà ta chạy node.js.

Dưới đây là ví dụ để truy cập biến môi trường NODE\_ENV, biến này mặc định được thiết lập có giá trị là ‘development’

Ngoài ra, ‘process’ không buộc ta phải ‘require’ nó trước khi sử dụng.



1. Tìm hiểu lập trình động bộ và lập trình bất đồng bộ.

Synchronous có nghĩa là xử lý đồng bộ, chương trình sẽ chạy theo từng bước và chỉ khi nào bước 1 thực hiện xong thì mới nhảy sang bước 2, khi nào chương trình này chạy xong mới nhảy qua chương trình khác. Đây là nguyên tắc cơ bản trong lập trình mà bạn đã được học đó là khi biên dịch các đoạn mã thì trình biên dịch sẽ biên dịch theo thứ tự từ trên xuống dưới, từ trái qua phải và chỉ khi nào biên dịch xong dòng thứ nhât mới nhảy sang dòng thứ hai, điều này sẽ sinh ra một trạng thái ta hay gọi là trạng thái chờ. Ví dụ trong quy trình sản xuất dây chuyền công nghiệp được coi là một hệ thống xử lý đồng bộ.

Ngược lại với Synchronous thì Asynchronous là xử lý bất động bộ, nghĩa là chương trình có thể nhảy đi bỏ qua một bước nào đó, vì vậy Asynchronous được ví như một chương trình hoạt động không chặt chẽ và không có quy trình nên việc quản lý rất khó khăn. Nếu một hàm A phải bắt buộc chạy trước hàm B thì với Asynchronous sẽ không thể đảm bảo nguyên tắc này luôn đúng.

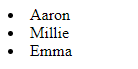
Để có thể hiểu rõ về lập trình đồng bộ và bất động bộ mời xem qua ví dụ sau. Trong ví dụ này, mục tiêu công việc của ta là thêm 1 user vào danh sách và đồng thời xuất danh sách các user lên trình duyệt (danh sách user bao gồm cả user vừa thêm).

Trước khi vào ví dụ, cần lưu ý là ví dụ này sẽ sử dụng hàm setTimeout() để mô phỏng độ trễ trong thực tế khi phải làm việc với database.

Ví dụ:



Khi vào trình duyệt ta sẽ nhận được kết quả như sau:



Tại sao user có tên ‘John’ lại không xuất hiện trong danh sách.

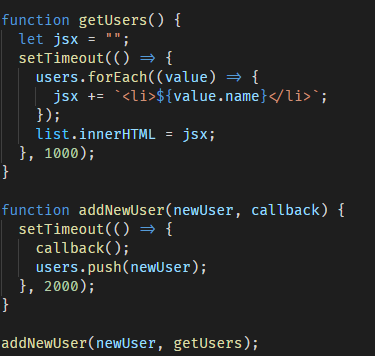
Lý dó là bởi, do độ trễ của “addNewUser” lớn hơn 1 giây so với “getUsers”, vì thế “getUsers” được thực thi trước, nên chỉ render ra 3 user ban đầu, nên dù ta có hoán đổi thứ tự gọi của 2 hàm kết quả vẫn như vậy. Dù không được render lên DOM nhưng ‘John’ vẫn được thêm vào mảng ‘users’. Vậy làm sao để giải quyết vấn đề này, làm sao để có thể vừa thêm mới một user và in danh sách toàn bộ user ban đầu và user mới lên trình duyệt? Để giải quyết vần đề này ta sẽ chuyển sang phần kế tiếp.

1. Xử lý bất đồng bộ với Callback

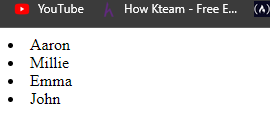
Tiếp tục từ ví dụ ở phần trước, để giải quyết vấn đề làm sao để có thể vừa thêm mới một user và in danh sách toàn bộ user ban đầu và user mới lên trình duyệt.

Ta sẽ sử dụng callback trong JS, được giới thiệu trong ES5.

Từ vị dụ trên ta sẽ sử dụng callback() trong hàm addNewUser, ta được:



Kết quả thu được:



Lý do là bởi, sau khi gọi “addNewUser”, delay sau 2s, thì callback (chính là function “getPost”) được gọi trước, nhưng delay của nó là 1 giây nên việc thêm “John” sẽ được thực thi trước sau nó callback được thực thi, vì thế ta nhận được kết quả trên.

1. Xử lý bất động bộ với Promise.

Để giải quyết bất đồng bộ, ngoài dùng callback như phần trước ta còn có thể dùng Promise trong JS.

Được giới thiệu trong ES5 và cho đến giờ nó đã phát triển, với ‘async’ và ‘await’ trong ES7

Promise là một đối tượng giúp ta giải quyết các vấn để bất động bộ trong Javascript.

Một Promise có 3 trạng thái:

* Pendding: promise đang chờ xử lý.
* Fullfilled: promise đã được thưc hiện/hoàn thành.
* Rejected: promise bị hủy bỏ/không được thực hiện.

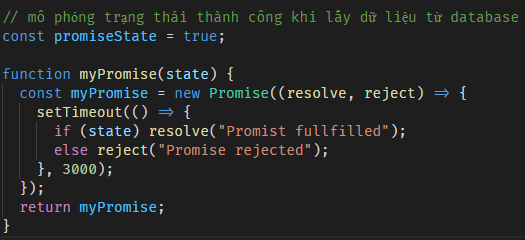
Khi một Promise được gọi, nó sẽ bắt đầu ở trạng thái Pending. Điều này nghĩa là hàm đang được gọi sẽ tiếp tục thực thi, trong khi Promise đang chở được xử lý cho đến khi nó được giải phóng bằng resolves, và trả về cho hàm dữ liệu mà nó request.

Đối tượng Promise nhận 1 tham số “executor” là 1 một function hoặc dùng Arrow function đều được, “executor” có 2 tham số là 2 function phản hồi khác nhau.

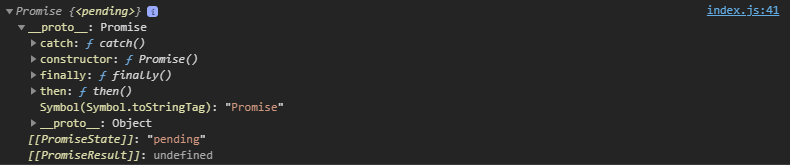
resolve : function này trả về một Promise có trạng thái “fullfilled”, sử dụng khi nếu điều kiện đưa ra thỏa mãn.

reject: function này trả về một Promise có trạng thái “rejected”, sử dụng khi nếu điều kiện hoặc xảy ra lỗi trong quá trình thực thi.

Ví dụ:



Khi ta gọi hàm myPromise(), và giả sử ta lấy thành công dữ liệu từ database, khi log ra console ta được kết quả như sau:



Như hình, ta có thể thấy, “PromiseState” đang ở trạng thái ‘pending’, điều này nghĩa là Promise đã có được kết quả nhưng vẫn đang chờ được xứ lý. Để xử lý và lấy kết quả từ hai phương thức resolve() và reject() trả về ta sử dụng callback trong 2 phương thức ‘then()’ và ‘catch()’, kết quả của ‘resolve’ sẽ được trả về trong callback của then() và ‘reject’ được trả về trong callback của catch().

Ví dụ:



Kết quả thu được:



Kết quả trả về “Promist fullfilled” vì resolve() được gọi trong Promise, đồng nghĩa là việc lấy dữ liệu database thành công.

1. Xử lý bất đồng bộ với Async và Await trong Javascript.

ES5 đã cung cấp cho ta đối tượng Promise để có thể xử lý bất động bộ trong JS, tuy nhiên việc vẫn sử dụng callback trong then() và catch() dẫn đến khó debug. Chính vì thế ES7 đã tạo ra async/await giải quyết dứt khoát vấn đề callback.

Để khai báo một function asynchronouse và sử dụng await, ta sẽ dùng từ khóa “async” trước function đó.

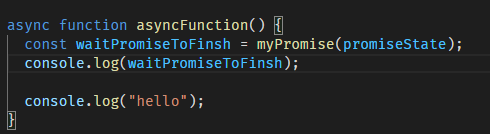
Async dùng đề xử lý bất đồng bộ trong Javascript, giúp cho code sẽ trông như là đồng bộ.

Await sẽ đảm bảo cho ta Promise của ta sẽ được chạy và trả về kết quả trước khi hàm async kết thúc.

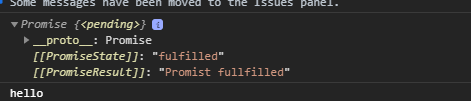
Lấy vị dụ từ phần Promise trước đó.

Đầu tiên ta có một function trả về một Promise, thời gian delay của hàm này sẽ là 3 giây (mô phỏng độ trễ khi gửi request và nhận respone từ database). Ngoài ra ta cũng dùng biến “promiseState” để đại diện cho trạng thái “respone” về có thành công hay không.

Ví dụ:



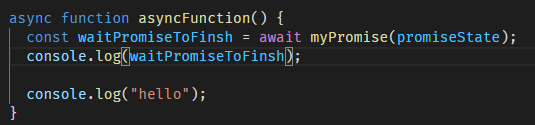
Kết quả nhận được:



Khác với Promise, nếu ta không dùng then() và catch() thì Promise của chúng ta sẽ mãi ở trạng thái ‘pending’ và ‘PromiseResult’ sẽ là ‘undefined’.

Nhưng khi sử dụng ‘async’ khai báo trước hàm, Promise được trả về lúc này có ‘promiseState’ là ‘fullfilled’ và ‘promiseResult’ là kết quả của ‘resolve()’ trả về. Tuy biết được được giá trị Promise trả về, nhưng ta không thể sử dụng được, bởi bản chất nó vẫn đang là một Promise và vẫn chưa được xử lý để lấy và sử dụng kết quả.

Để giải quyết vấn đề này, ta sử dụng từ khóa ‘await’ để xử lý Promise này, sử code ta được:



Kết quả nhận được:



Nhờ có async/await code của chúng ta sẽ trở nên dễ đọc hơn, tạo cảm giác như đang lập trình đồng bộ, dễ debug, kiểm tra kết quả.

1. Tìm hiểu về Node.js HTTP module.

Module HTTP trong Node.js, cho phép ra transfer data thông qua giao thức HTTP (Hyper Text Transfer Protocol).

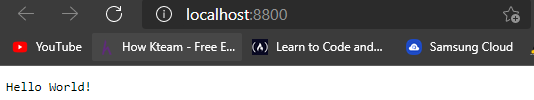


Sau khi require module http, ta có thể tạo một server mới thông qua phương thức createServer(), phương thức này nhận một callback có 2 tham số là req là biến chứa các thông tin được gửi đến server, res là biến dùng để phản hồi về cho request gửi đến.

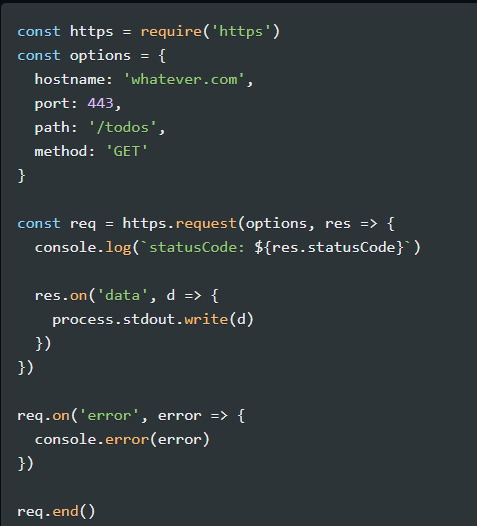
Sau khi đã tạo server, tiếp đến là sử dụng phương thức listen() để thiết lập cổng và host mà server sẽ được thực thi và truy cập.

Phương thức listen nhận 3 tham số: port, hosthostname, và callback function.

Nếu port và hostname chưa được sử dụng, nghĩa là server sẵn sàng có thể hoạt động, callback function sẽ được gọi, kết quả:



* 1. Tạo request đến server.



Trong ví dụ trên ta tạo một request theo phương thức get, đến server. Ta cũng có thể thay đổi phương thức, bẳng cách thay đổi thuốc tính ‘method’ trong opstions thành POST, DELETE, PUT, PATH.

Ví dụ: request phương thức post



1. Tìm hiểu về Node.js File System module.

Node.js file system module cho phép ta làm việc với file system trên máy tính.

Để sử dụng ta phải include moduel File System ‘fs’.



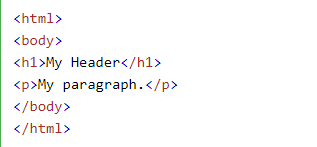
Các chức năng phổ biến ở File Sytem module:

* Read files
* Create files
* Update files
* Delete files
* Rename files
* Read Files.

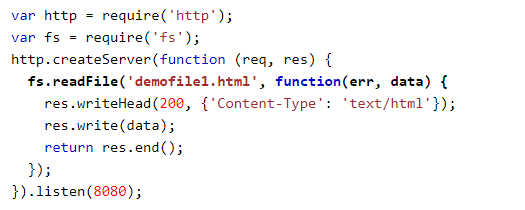
Phương thức fs.readFile(path, callback function(err, data)) được sử dụng để đọc file trên máy.

Ghi chú: biến data, chứa nội dung sau khi đọc file

Giả sử ta có file HTML sau:



Tạo file Node.js để đọc HTMl file, và trả về nội dung:



Khi truy cập vào http://localhost:8080/

Kết quả hiện ra chính là nội dung bên trong file HTML

* 1. Tạo file trong File System module.

File System module có những phương thức giúp ta tạo mới file:

* fs.appenFile()
* fs.open()
* fs.writeFile()

*fs.appendFiile(path, data[option, ], callback function(err))*

Phương thức này dùng để thêm, nối một nội dung cụ thể nào đó vào cuối file, nếu như file mục tiêu không tồn tại, thì nó sẽ tạo mới.

Ví dụ:



*fs.open(path, flag mode [string | number], callback function(err, file) )*

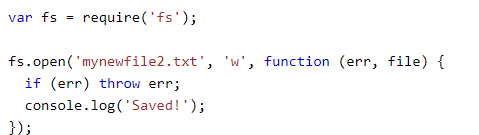
fs.open() có tham số thứ 2 chính là ‘flag’.

‘flag’ mode này có thể là string | number

Mặc định của nó là ‘r’ (read)

‘w’: là writing, nếu file không tồn tại, thì tạo mới một file rỗng.

Ví dụ:



*fs.wirteFile(path, data[option, ], callback function(err))*

phương thức này sẽ replace file hoặc nội dung của file đó nếu nó đã tồn tại trước đó, nếu như file không tồn tại thì sẽ tạo mới, và chứa nội dung wirte.

Ví dụ:

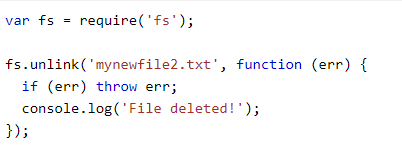


* 1. Cập nhật file trong File System module.

Các hàm dùng để update file

* Fs.appendFile(“file url”, “append content”, callback function(err))
* Fs.writeFile(‘file url’, ‘write content’, callback function(err))
  1. Xóa file trong File System module.

Để xóa file với File System module ta có thể sử dụng phương thức fs.unlink().



* 1. Đổi tên file trong File System module.

Để đổi tên file ta dùng phương thức fs.rename(path, String<new name>, callback function(err))

Nếu file không tồn tại, thì sẽ báo lỗi.

Ví dụ:



1. Tìm hiểu về Node.js URL module.

The URL module giúp ta tách địa chỉ web thành nhiều phần, giúp ta dễ nó trở nên readable. Tương tự như bao module khác, để sử dụng ta cần include nó vào.

var url = require('url');

Để tách đường dẫn web thành từng phần, ta sử dụng phương thức url.parse(path, parseQueryString[boolean]);

Mặc định của parseQueryString : false, nếu là true, thì trong Object trả về luôn luôn tồn tại một “query” property.

Ví dụ: sau khi tạo Server local trên port 8080.

var url = require("url");

var adr = "http://localhost:8080/default.htm?year=2017&month=february";

var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'

console.log(q.pathname); //returns '/default.html'

console.log(q.search); //returns '?year=2017&month=february'

var qdata = q.query; //returns an object: { year: 2017, month: 'february' }

console.log(qdata.month); //returns 'february'

ta có thể console.log(q) để xem thêm các properties khác.

1. Event module và đối tượng EventEmitter.

Node.js có xây dựng module tên là “Event”, cho phép ta, creat-, fire-, và for- cho các sự kiện.

Tượng tự, để sử dụng, trước hết ta phải include nó thông qua hàm require(“events”).

Ngoài ra, tất cả các event properties và phương thức đều là một thể hiện của EventEmitter object. Để có thể truy cập các properties và phương thức, ta có thể create EventEmitter object:

var events = require('events');

var eventEmitter = new events.EventEmitter();

Chúng ta có thể gán event handler cho chính events mà ta tạo ra với EventEmitter object.

Trong ví dụ dưới đây, có function sẽ được thực thi khi “talk” event được fired.

Để fire một event nào đó, ta sử dụng phương thức emit(“event name”);

Ví dụ:

var events = require("events");

var eventEmitter = new events.EventEmitter();

// hàm dùng để bắt sự kiện

var myEventHandler = function () {

  console.log("Did someone talking ?");

};

// sự kiện "talk" chính là event mà ta tự tạo ra. chứ không phải của hệ thống có sẵn

// chạy hàm myEventHandler khi sự kiện "talk" diễn ra

// tương tự như cách ta addEventListener trên trình duyệt

eventEmitter.on("talk", myEventHandler);

// fire (trigger) sự kiện "talk"

eventEmitter.emit("talk");

1. Giới thiệu Express framework trong Node.js.
   1. Tổng quan về Express.

Express là một frame work giành cho nodejs. Nó cung cấp cho chúng ta rất nhiều tính năng mạnh mẽ trên nền tảng Web cũng như trên các ứng dụng di động.

Về package hỗ trợ : Expressjs có vô số các package hỗ trợ nên các bạn không phải lo lắng.

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

Và hơn hết, các framework nổi tiếng của Nodejs hiện nay đều sử dụng Expressjs như một core function, chẳng hạn : SailsJS, MEAN,…

Express hỗ trợ các phương thức HTTP và midleware tạo ra một API vô cùng mạnh mẽ và dễ sử dụng .Có thể tổng hợp một số chức năng chính của Express như sau :

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

Để cài đặt Express vào dự án chúng ta , ta vào Teminal và gõ lệnh sau :

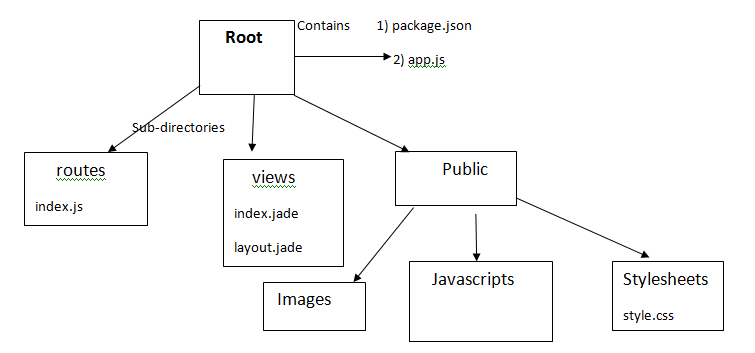
* $ mkdir express
* $ cd express
* npm install express –save

Với các câu lệnh trên sẽ lưu phần cài đặt trong thư mục node\_modules và tạo thư mục express đó. Chúng ta cần cài đặt thêm một số modules quan trọng đi cùng Express như sau :

* body-parser : đây là lớp trung gian , xử lý JSON , text và mã hóa URL
* cookie-parser : Chuyển đổi header của cookie và phân bố đến các req.cookies
* multer – xử lý phần multipart/form-data

Để cài đặt các module trên , lần lượt chạy các lệnh :

* $ npm install body-parser –save
* $ npm install cookie-parser –save
* $ npm install multer –save
  1. Cấu trúc của Express.



* 1. Cấu trúc của nodejs vô cùng đơn giản :
* Root
* App.js : chứa các thông tin về cấu hình , khai báo , định nghĩa,…để ứng dụng của thể chạy tốt.
* Package.json chứa các package cho ứng dụng chạy.
* Folder routes:chứa các route(tuyến đường) có trong ứng dụng
* Folder view : chứa các view/template cho ứng dụng
* Folder public chứa các file css, js ,images,…cho ứng dụng
  1. Tìm hiểu về Router trong Express.

Router là một Object nó là một instance riêng của middleware và routes. Chính vì nó là một instance của middleware và route nên nó có các chức năng của cả hai. Có thể gọi nó là một mini-application.

Các Application dùng Expressjs làm core đều có phần Router được tích hợp sẳn trong đó.

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

* 1. Tìm hiểu về các method của Router.

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

Lưu ý tên các method phải viết thường ( lowercase)

Ví Dụ :

router.get('/user/profile', function(req, res, next) {

res.send('user profile');

});

router.post('/'update/user/:id', function (res, req, next) {

res.send('Update user');

});

router.put('/update/posts/:id', function (req, res, next) {

res.send('Update post');

});

* 1. Tìm hiểu về Middleware trong Expressjs

Expressjs khi hoạt động, về cơ bản sẽ là một loạt các hàm Middleware được thực hiện liên tiếp nhau.Sau khi đã thiết lập, các request từ phía người dùng khi gửi lên Expressjs sẽ thực hiện lần lượt các hàm Middleware cho đến khi response cho người dùng. Các hàm này sẽ được quyền truy cập đến các đối tượng đại diện cho Request-req, response-res, hàm Middleware tiếp theo – Next, và đối tượng lỗi -err nếu cần thiết.

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

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

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

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

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

Khi khởi tạo một Web Application với ExpressJs , ta sẽ có một đối tượng đại diện cho Web App đó, thường được gán với biến app.Đối tượng này có thể khai báo các middleware thông báo các hàm : app.use() hoặc app.METHOD ( trong đó METHOD sẽ là các kiểu HTTP METHOD được ExpressJs hỗ trợ , dưới dạng tên là chữ viết thường, vd : app.get(), app.post().

Ví dụ :

var app = express()

app.use(function (req, res, next) {

console.log('Time:', Date.now())

next()

})

Ví dụ dưới đây dùng hàm use đến đường dẫn /user/:id. Hàm này sẽ được thực hiện mỗi khi request đến đường dẫn /user/:id bất kể phương thức nào (GET,POST,..):

app.use('/user/:id', function (req, res, next) {

console.log('Request Type:', req.method)

next()

})

Ví dụ cho một hàm được thực hiện mỗi khi truy cập đến đường dẫn /user/:id bằng phương thức GET :

app.get('/user/:id', function (req, res, next) {

res.send('USER')

})

Khi muốn gọi một loạt hàm middleware cho đường dẫn cụ thể, chúng ta có thể thực hiện như ví dụ dưới đây, bằng cách khai báo liên tiếp các tham số là các hàm sau tham số đường dẫn :

app.use('/user/:id', function (req, res, next) {

console.log('Request URL:', req.originalUrl)

next()

}, function (req, res, next) {

console.log('Request Type:', req.method)

next()

})

Hoặc chúng ta có thể tách ra thành 2 lần khai báo app.use, gọi là multiple routes, tuy nhiên ở các hàm phía trước cần gọi hàm next() khi kết thúc mỗi hàm, nếu không như ví dụ thì route thứ 2 sẽ không bao giờ được thực hiện do hàm thứ 2

Trong route thứ nhất không gọi đến hàm next() :

app.get('/user/:id', function (req, res, next) {

console.log('ID:', req.params.id)

next()

}, function (req, res, next) {

res.send('User Info')

})

// handler for the /user/:id path, which prints the user ID

app.get('/user/:id', function (req, res, next) {

res.end(req.params.id)

})

* 1. Router-level middleware

Các middleware này về chức năng không khác so với application-level middleware ở trên, tuy nhiên thay vì dùng biến app có thể gây nhầm lẫn với các thiết lập, phần router có thể không rõ ràng và khó phân biệt, ExpressJs cung cấp đối tượng router chuyên dùng để khai báo route bằng cách gọi hàm sau :

var router = express.Router()

* 1. Error-handling middleware

Đây là middleware phục vụ cho việc xử lý lỗi. Một số lưu ý là các hàm cho việc này luôn nhận bốn tham số ( err,req,res,next). Khi muốn khai báo một middleware cho việc xử lý lỗi, bạn cần tạo một hàm có 4 tham số đầu vào. Mặc dù bạn có thể không sử dụng đối tượng next, nhưng hàm vẫn còn format với 4 tham số như vậy. Nếu không ExpressJs sẽ không thể xác định đó là hàm xử lý lỗi, và sẽ không chạy khi có lỗi xảy ra, chỉ hoạt động giống như các middleware khác.

Ví dụ mô tả một hàm xử lý lỗi truyền cho client lỗi 500 khi có lỗi xảy ra từ sever :

app.use(function (err, req, res, next) {

console.error(err.stack)

res.status(500).send('Lỗi')

})

* 1. Built-in middleware

Kể từ phiên bản 4.x, ExpressJs đã không còn phụ thuộc vào thư viện Connect. Ngoài middleware express.satic, tất cả các hàm middleware khác đều đã được tách ra thành các modules riêng biệt. Điều này cung cấp cách tối ưu hóa và tùy chỉnh ứng dụng ExpressJs một cách linh hoạt nhất, giúp bạn tạo ra một ứng dụng Web Application phù hợp với nhu cầu, không bị thừa những thứ không cần thiết.

Chỉ có một built-in middleware duy nhất còn tồn tại trong ExpressJs là express.static, dựa trên thư viện Serve-static, được dùng để cung cấp các nội dung tĩnh trong trang Web, ví dụ như các trang HTML tĩnh , các file hình ảnh,css,js,…

Ví dụ dưới đây mô tả việc sử dụng express.static để tạo ra một thư mục có tên là public, người dùng có thể truy cập các file html và htm trong thư mục này :

var options = {

dotfiles: 'ignore',

etag: false,

extensions: ['htm', 'html'],

index: false,

maxAge: '1d',

redirect: false,

setHeaders: function (res, path, stat) {

res.set('x-timestamp', Date.now())

}

}

app.use(express.static('public', options))

Ngoài ra bạn có thể khai báo nhiều thư mục static trong web , ví dụ sau sẽ tạo ra 3 thư mục static :

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

app.use(express.static('uploads'))

app.use(express.static('files'))

* 1. Third-party middleware

Sử dụng Third-party sẽ giúp chúng ta thêm các chức năng cho Web App của mình mà không cần thiết mất nhiều công implement.

Chúng ta cần cài đặt module thông qua npm, sau đó khai báo sử dụng đối tượng app nếu dùng ở application-level, hoặc qua đối tượng router nếu dùng ở Router-level.

Ví dụ sẽ cài đặt và sử dụng middleware có tên là cookie-parser dùng để đọc cookies của request :

$ npm install cookie-parser

var express = require('express')

var app = express()

var cookieParser = require('cookie-parser')

// load the cookie-parsing middleware

app.use(cookieParser())

1. Xây dựng website thương mại điện tử sử dụng Node.js Express.
   1. Giới thiệu chung.

Xây dựng website thương mại điển tử xử dụng công nghệ Node.js framework Express. Website được xây dựng với các chức năng như: trình bày sản phẩm, tìm kiếm, đặt hàng.

* 1. Công nghệ khác.
* React.

React là thư viện JavaScript phổ biến nhất để xây dựng giao diện người dùng (UI). Nó cho tốc độ phản hồi tuyệt vời khi user nhập liệu bằng cách sử dụng phương pháp mới để render trang web.

Components của công cụ này được phát triển bởi Facebook. Nó được ra mắt như một công cụ JavaScript mã nguồn mở vào năm 2013.

* TailwindCss.

Tailwind là Framework CSS để phát triển UI nhanh chóng. Khi đề cập đến các Framework CSS, điều đầu tiên bạn nghĩ đến là Bootstrap, Material Design hoặc Bulma. Tuy nhiên, mặc dù Tailwind được coi là một Framework CSS, nó không phù hợp với triết lý tương tự như các Framework CSS khác. Tailwind chủ yếu là một khung công cụ tiện ích hay đúng hơn là một gói tiện ích. Tailwind không cung cấp chủ đề mặc định hoặc bất kỳ thành phần UI nào được định nghĩa trước.

Tailwind cung cấp các công cụ để phát triển nhanh chóng, đồng thời cho phép nhà phát triển maintain. Mặc dù không có giới hạn đối với trí tưởng tượng với Tailwind, thiết kế được cấu trúc theo cách để nhà phát triển có thể tạo một quy tắc tự áp đặt để tránh tăng size của CSS hoặc các thuộc tính lặp đi lặp lại.

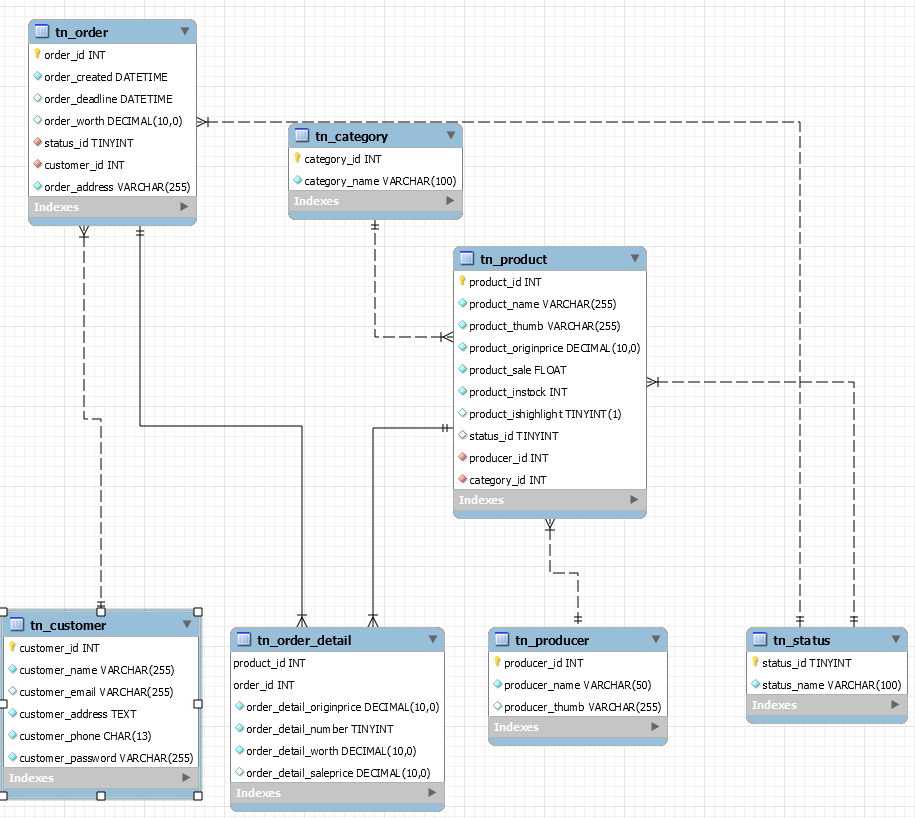
* Json Web Token.

JWT là một phương tiện đại diện cho các yêu cầu chuyển giao giữa hai bên Client – Server , các thông tin trong chuỗi JWT được định dạng bằng JSON . Trong đó chuỗi Token phải có 3 phần là header , phần payload và phần signature được ngăn bằng dấu “.”. JWT được sử dụng để:

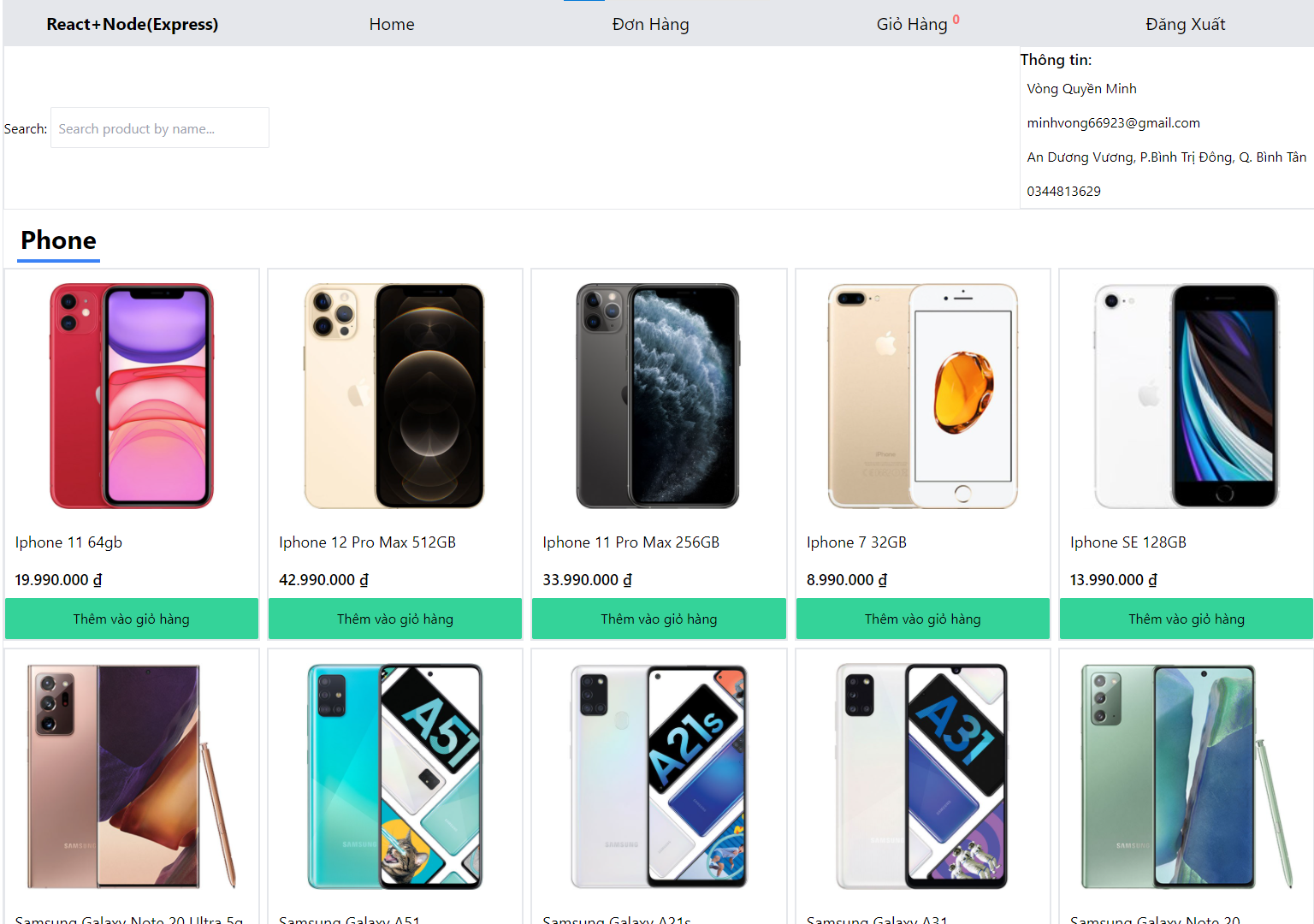
Authentication: Đây là trường hợp phổ biến nhất thường sử dụng JWT. Khi người dùng đã đăng nhập vào hệ thống thì những request tiếp theo từ phía người dùng sẽ chứa thêm mã JWT. Điều này cho phép người dùng được cấp quyền truy cập vào các url, service, và resource mà mã Token đó cho phép. Phương pháp này không bị ảnh hưởng bởi Cross-Origin Resource Sharing (CORS) do nó không sử dụng cookie.

Trao đổi thông tin: JSON Web Token là 1 cách thức khá hay để truyền thông tin an toàn giữa các thành viên với nhau, nhờ vào phần signature của nó. Phía người nhận có thể biết được người gửi là ai thông qua phần signature. Và chữ ký được tạo ra bằng việc kết hợp cả phần header, payload lại nên thông qua đó ta có thể xác nhận được chữ ký có bị giả mạo hay không.

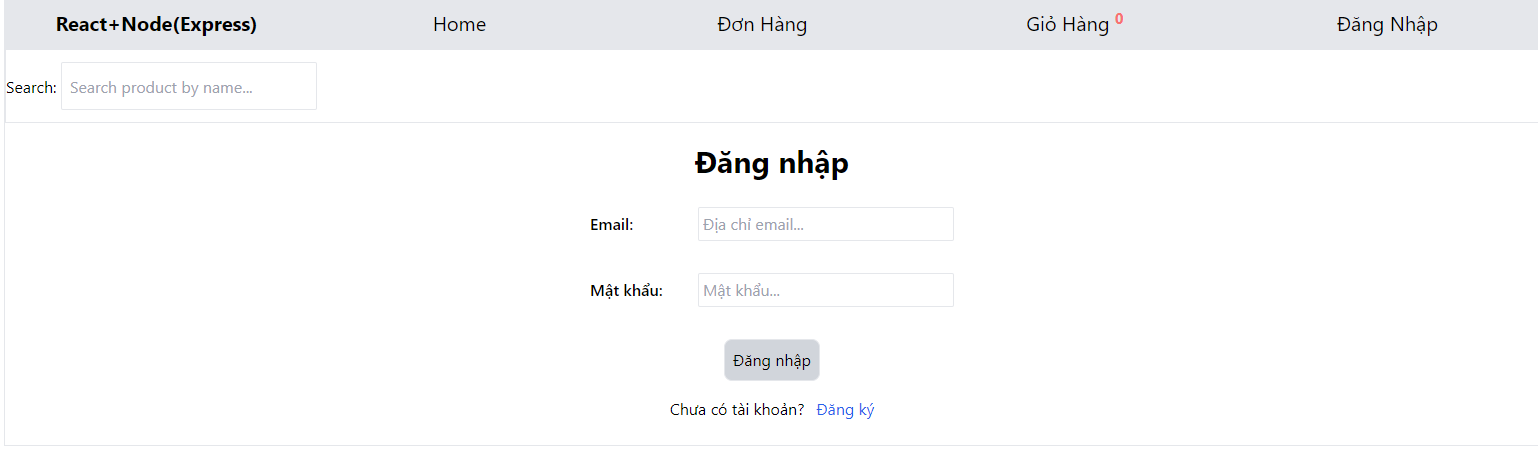
* 1. Cơ sở dữ liệu.



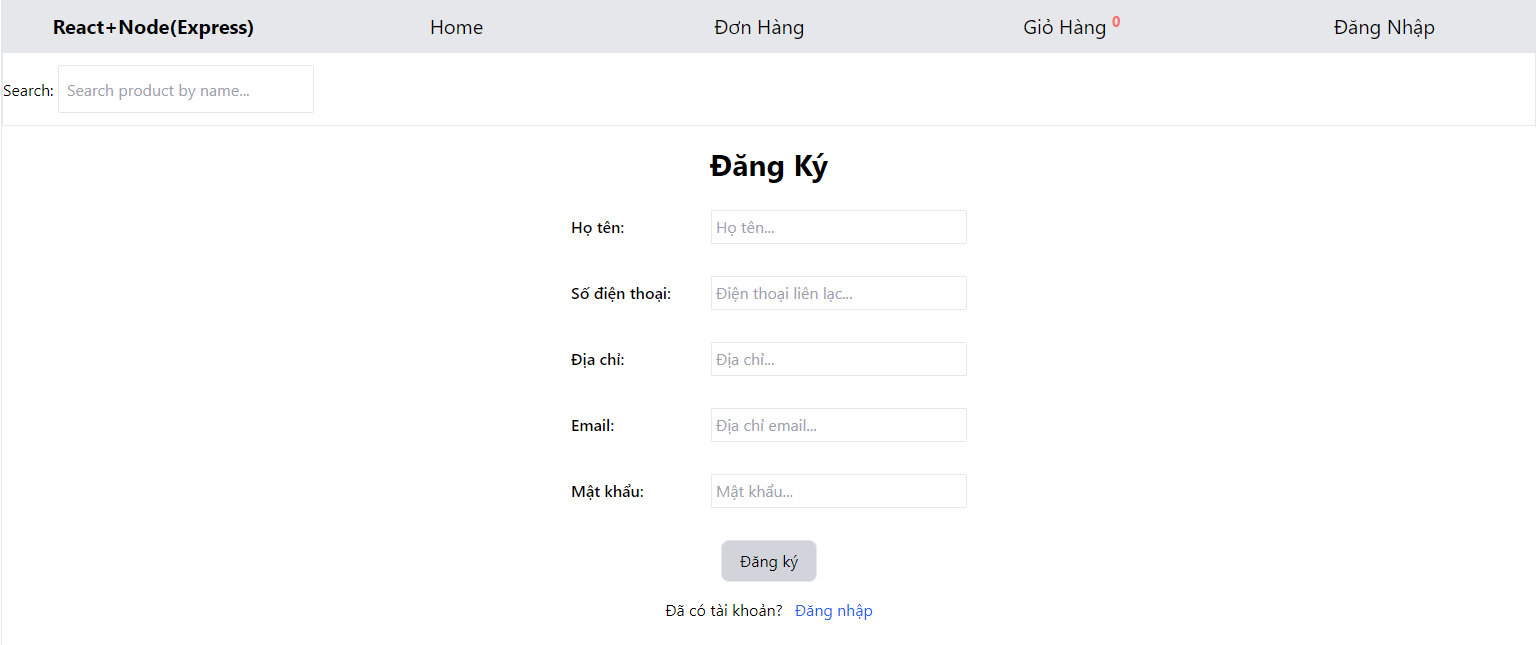
* 1. Giao diện website.



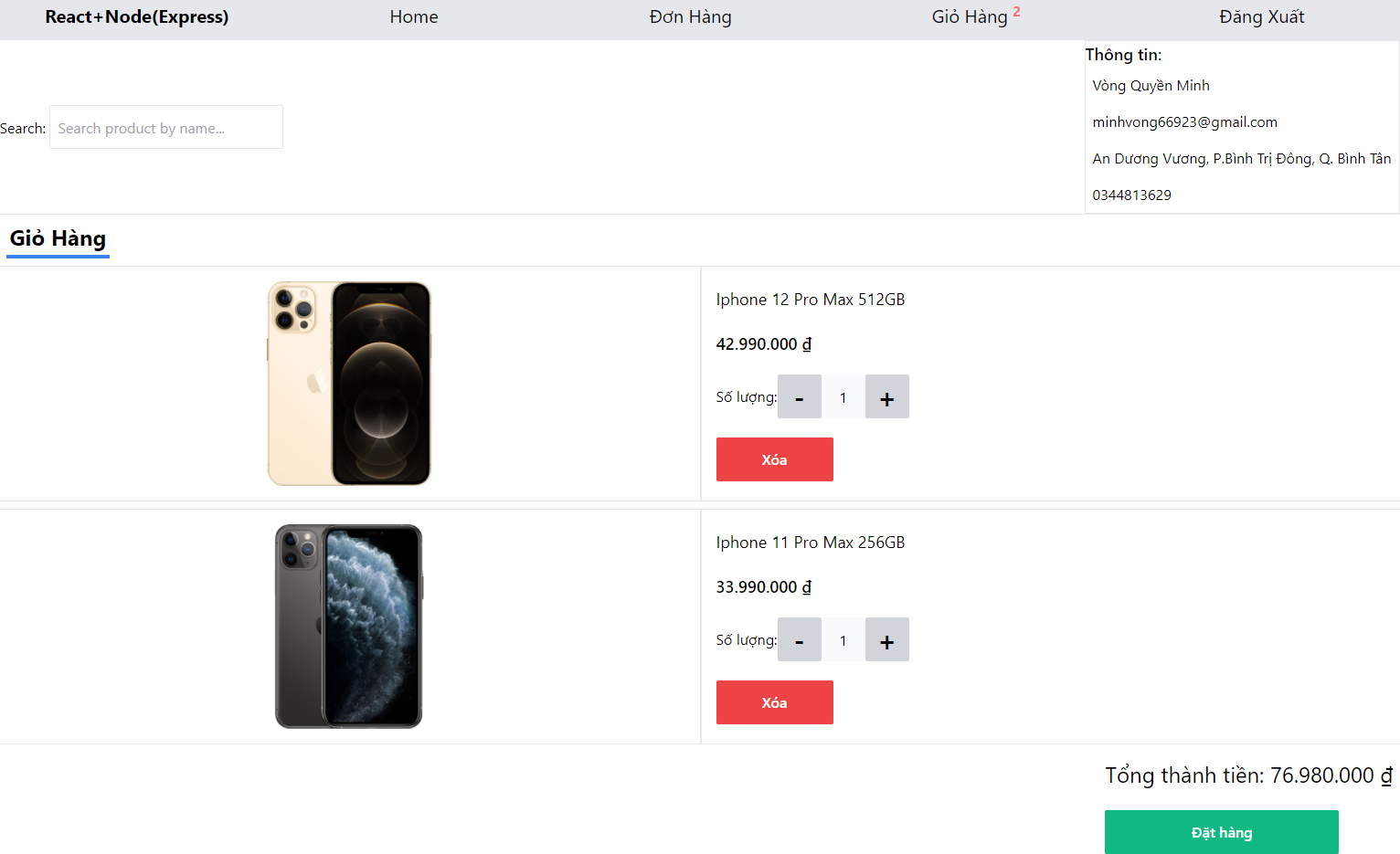
* 1. Giao diện đăng nhập.



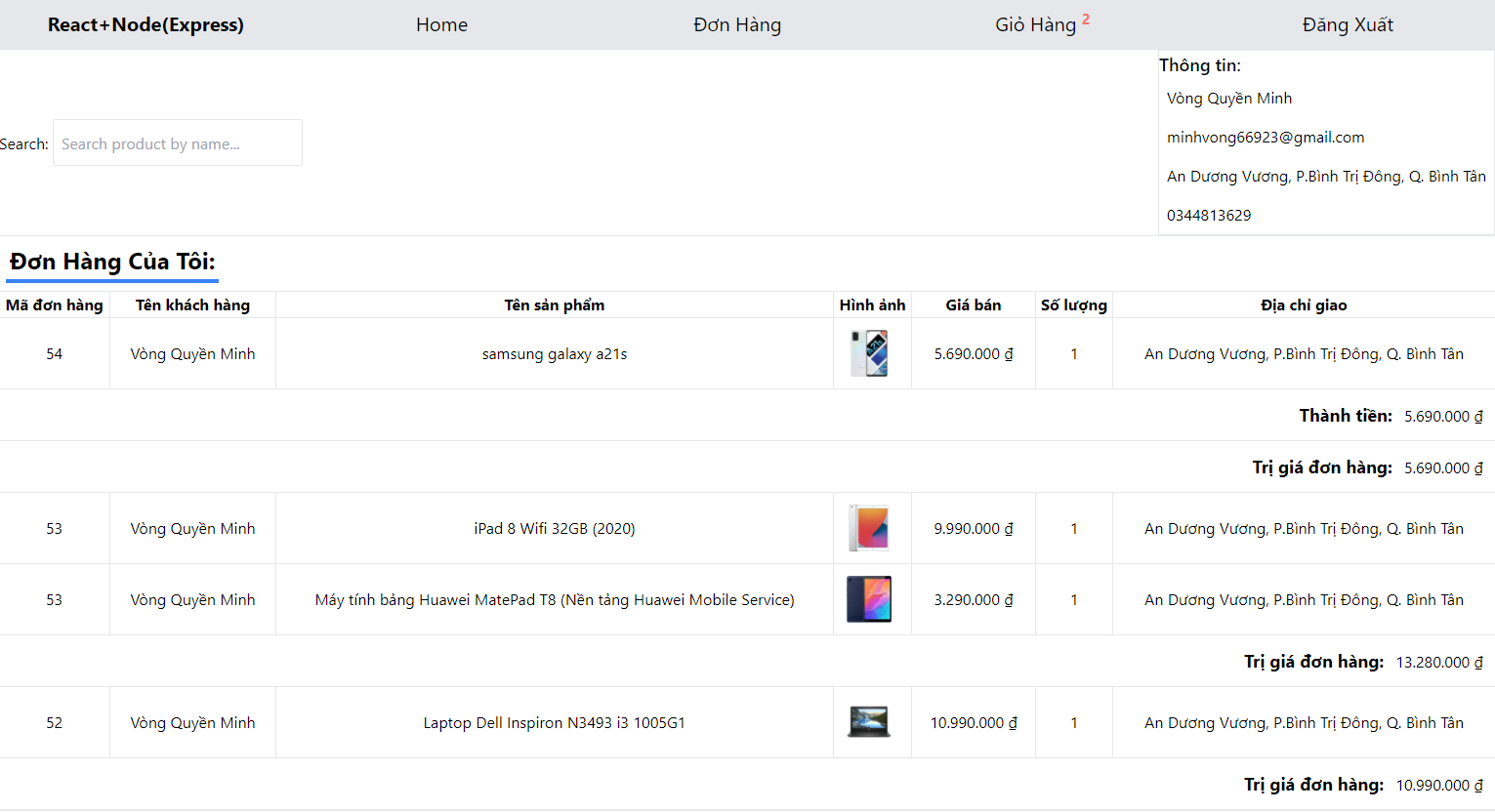
* 1. Giao diện đăng ký.



* 1. Giao diện giỏ hàng.



* 1. Giao diện xem đơn hàng đã đặt.



* 1. Source Code.

Github: <https://github.com/AaronVong/EcommerceApp-React-Express-Mysql>