

**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM**

**Khóa Luận Tốt Nghiệp**

Đề tài : Handmade Instructor

# MỤC LỤC

[MỤC LỤC 1](#_Toc528749021)

[DANH MỤC CÁC HÌNH VẼ 3](#_Toc528749022)

[DANH MỤC CÁC BẢNG BIỂU 3](#_Toc528749023)

[LỜI MỞ ĐẦU 4](#_Toc528749024)

[CHƯƠNG 1 : GIỚI THIỆU 5](#_Toc528749025)

[1.1 Tổng quan 5](#_Toc528749026)

[1.2 Mục tiêu đề tài 5](#_Toc528749027)

[1.3 Phạm vi đề tài 6](#_Toc528749028)

[1.4 Mô tả yêu cầu chức năng 7](#_Toc528749029)

[CHƯƠNG 2 : CƠ SỞ LÝ THUYẾT 8](#_Toc528749030)

[2.1 Nodejs Express framework: 8](#_Toc528749031)

[2.2 Nodejs 8](#_Toc528749032)

[2.3 RESTful API 10](#_Toc528749033)

[2.4 JSON Web Token 13](#_Toc528749034)

[2.5 Multer 20](#_Toc528749035)

[CHƯƠNG 3 : PHÂN TÍCH 30](#_Toc528749036)

[3.1 Tiêu đề mục 30](#_Toc528749037)

[3.1.1 Tiêu đề tiểu mục 30](#_Toc528749038)

[3.1.2 Tiêu đề tiểu mục 30](#_Toc528749039)

[3.2 Tiêu đề mục 30](#_Toc528749040)

[3.2.1 Tiêu đề tiểu mục 30](#_Toc528749041)

[3.2.2 Tiêu đề tiểu mục 30](#_Toc528749042)

[CHƯƠNG 4 : THIẾT KẾ VÀ HIỆN THỰC 31](#_Toc528749043)

[4.1 Tiêu đề mục 31](#_Toc528749044)

[4.1.1 Tiêu đề tiểu mục 31](#_Toc528749045)

[4.1.2 Tiêu đề tiểu mục 31](#_Toc528749046)

[4.2 Tiêu đề mục 31](#_Toc528749047)

[4.2.1 Tiêu đề tiểu mục 31](#_Toc528749048)

[4.2.2 Tiêu đề tiểu mục 31](#_Toc528749049)

[CHƯƠNG 5 : KẾT LUẬN 32](#_Toc528749050)

[5.1 Kết quả đạt được 32](#_Toc528749051)

[5.2 Hạn chế của đồ án 32](#_Toc528749052)

[5.3 Hướng phát triển 32](#_Toc528749053)

[TÀI LIỆU THAM KHẢO 32](#_Toc528749054)

[PHỤ LỤC 33](#_Toc528749055)

# DANH MỤC CÁC HÌNH VẼ

[Hình 1‑1 Thao tác cập nhật mục lục **Error! Bookmark not defined.**](#_Toc262311533)

[Hình 1‑2 Cách chèn nhãn cho hình **Error! Bookmark not defined.**](#_Toc262311534)

[Hình 1‑3 Cách tạo một nhãn mới **Error! Bookmark not defined.**](#_Toc262311535)

[Hình 1‑4 Cách tham chiếu đến một nhãn **Error! Bookmark not defined.**](#_Toc262311536)

# DANH MỤC CÁC BẢNG BIỂU

[Bảng 1‑1 Tên bảng **Error! Bookmark not defined.**](#_Toc262311537)

# LỜI MỞ ĐẦU

1. Tổng quan tình hình nghiên cứu thuộc lĩnh vực của đề tài

Trong phần này, sinh viên cần tìm hiểu để biết được tình hình nghiên cứu thuộc lĩnh vực của đề tài hoặc những thông tin liên quan trên thế giới.

Sinh viên cần tìm hiểu để biết được tình hình nghiên cứu thuộc lĩnh vực của đề tài hoặc những thông tin liên quan trong nước (hoặc những đề tài liên quan mà những khóa trước đã thực hiện).

Liệt kê công trình sinh viên nghiên cứu khoa học có liên quan (nếu có): Tên công trình nghiên cứu, do ai thực hiện, thời gian nào.

Mục này sẽ giúp sáng tỏ vấn đề nghiên cứu trong đề tài, nó cho biết đề tài là mới hoặc trùng lặp hay không, nếu trùng lặp thì phải nêu lên những hạn chế của đề tài cũ so với đề tài mình sẽ thực hiện hoặc đưa ra giải pháp mới cho tình huống cũ.

2. Ý nghĩa khoa học và thực tiễn của đề tài

Ý nghĩa khoa học của đề tài?

Ý nghĩa thực tiễn của đề tài? (mục đích ứng dụng, ứng dụng kết quả trong thực tiễn, ứng dụng kết quả phục vụ các nghiên cứu khác)

Mục này sẽ là lý do để sinh viên lựa chọn đề tài và chính là cơ sở khoa học của đề tài.

# : GIỚI THIỆU

## Tổng quan

Handmade gần đây trở thành một từ khóa phổ biến trong giới trẻ. Thầy cô có thể bắt gặp handmade ở khắp mọi nơi từ món vật dụng rất nhỏ như dây buộc tóc, vòng tay, những chiếc ví đến những vật dụng có giá trị hơn như các món mỹ phẩm, nước hoa hoặc đồ nội thất. Nếu có dịp đi các hội chợ handmade, thầy/cô có thể chiêm ngưỡng rất nhiều các sản phẩm handmade với rất nhiều các chất liệu thể hiện sự sáng tạo rất độc đáo của giới trẻ. Có thể thầy/cô thắc mắc thực sự handmade là gì và lý do vì sao nó đang trở nên là một trào lưu thịnh hành như vậy không?

Handmade nói chung là vật được làm bằng tay, không sử dụng máy móc như các sản phẩm gia dụng được sản xuất hàng loạt như hiện nay. Một sản phẩm handmade thực sự phải được qua các công đoạn bằng tay một cách tỉ mỉ, thể hiện sự sáng tạo thông qua việc chọn lựa, kết hợp các chất liệu thô với nhau.

Hiện nay handmade đang dần trở nên thành một trào lưu, hấp dẫn nhiều người và giá thành sản phẩm luôn ở mức cao vì nó là thế giới của sự sáng tạo, sự đam mê của những người luôn muốn tạo ra những món đồ thật “độc”, “lạ”, “đẹp” .. và hơn hết là thể hiện được những dấu ấn riêng của bản thân. Chỉ cần vài món nguyên vật liệu rẻ tiền ngoài chợ cộng thêm óc sáng tạo, nghệ thuật và kiên nhẫn, mọi người đều có thể tự tạo ra một chiếc vòng tay, dây đeo cổ và không lo đụng hàng ☺ .

Chính vì thế việc có một môi trường xã hội đảm bảo cho mọi người có thể giao lưu học hỏi với nhau là điều không thể thiếu, vì số người bắt đầu làm các sản phẩm handmade đang dần gia tăng, đặc biệt là trong năm 2018 này. Vậy ý tưởng của đề tài là nhằm tới việc tạo ra một môi trường web, một máy chủ cung cấp các dịch vụ để mọi người có thể tham gia vào xã hội, một cộng đồng đam mê cũng như các tác giả với những sản phẩm của riêng mình. Mọi người có thể cùng nhau chia sẻ, hướng dẫn cách làm và học hỏi lẫn nhau.

## Mục tiêu đề tài

Mục tiêu đầu tiên : Đề tài -Handmade Instructor- phải có khả năng tối thiểu là cung cấp được nội dung bài đăng tương ứng với mục khách hàng muốn. Nếu khách hàng muốn “Trang trí” nó phải trả về được những bài hướng dẫn làm những món trang trí và những mục khác cũng như vậy.

Mục tiêu thứ hai : Đề tài phải đảm bảo được người dùng hoàn toàn có khả năng bình luận trên bài đăng của người khác.

Mục tiêu thứ ba : Đề tài phải cung cấp trang riêng cho mỗi người dùng đăng ký trang web, những người đăng ký đều có thể viết bài hướng dẫn, xem lại những bài mình đã share, xem hồ sơ công khai của những người dùng mình đang theo dõi hoặc đang được theo dõi, xem các nội dung được bình luận trên bài của mình hay cũng có thể xóa đi bài mình đã viết.

Mục tiêu thứ tư : Chức năng không thể thiếu trong các trang web là khả năng bảo mật và xác thực người dùng. Đề tài phải có khả năng kiểm tra thông tin đăng nhập, xác thực người dùng và cũng cho phép người dùng đăng kí tài khoản.

Mục tiêu thứ năm : Nội dung, dữ liệu trang web phải hoàn toàn được lưu trữ trên đám mây. Các nơi lưu trữ được cân nhắc và sẽ sử dụng :

1. DynamoDB của AWS
2. MongoDB Atlas
3. Simple Storage Service của AWS

Mục tiêu thứ sáu : Đảm bảo việc truy xuất dữ liệu thông qua các đám mây

## Phạm vi đề tài

Nhóm đã thống nhất sẽ sử dụng Express cho đề tài. Là một framework cho nodejs, nó cung cấp nhiều tính năng mạnh mẽ cho nền tảng web. Dễ dàng sử dụng, code ngắn gọn và nó phù hợp với các đề tài sử dụng nodejs cho việc phát triển web.

Một trong những nội dung cần nghiên cứu là việc routing của trang web. Express có hỗ trợ các phương thức HTTP và middleware tạo ra một API rất mạnh mẽ và sử dụng dễ dàng hơn. Điều này đã góp cho nhóm một phần nào sự thuận tiện.

Vấn đề thứ hai là JsonWebToken(JWT), nó là một tiêu chuẩn mở định nghĩa cách thức truyền tin an toàn giữa các thành viên bằng một đối tượng JSON. Thông tin này có thể được xác thực và đánh dấu tin cậy nhờ vào chữ ký của nó. Phần chữ ký của JWT sẽ được mã hóa lại bằng HMAC hoặc RSA. Có thể nói đơn giản rằng với mỗi yêu cầu login (với username và password đúng) thì nó sẽ trả về token cho phép người dùng vào hệ thống, và nếu không đúng username hay pass thì nó sẽ không gửi trả về bất kì token nào, khi không có token, bạn không đủ tin cậy vào hệ thống.

Vấn đề thứ ba là cân nhắc việc sử dụng amazon web service, cụ thể là dynamodb, simple storage service (S3) và đặc biệt là s3 vì việc sử dụng nó không cẩn thận sẽ gây rủi ro tài chính cho chính khoản tiền trong tài khoản của bạn. Đã có nhiều trường hợp phản hồi rằng EC2 tự khởi động, thực chất nó có liên quan đến s3 về việc khởi động máy chủ ảo để có thể dùng trực tiếp nội dung media của s3 nếu có (không chắc về điều này).

Vấn đề thứ tư là các thuật toán lấy dữ liệu thời tiết vẫn đang được phát triển và nghiên cứu để nó có thể lấy được dữ liệu vị trí người dùng và lấy dữ liệu thời tiết tương ứng.

## Mô tả yêu cầu chức năng

Sau khi phân tích yêu cầu chức năng, nhóm đưa ra các yêu cầu chức năng sau:

* + Khi truy cập vào server , người dùng phải thấy được trang chủ của hệ thống bao gồm tất cả các bài viết.
  + .Khi chọn vào một mục menu trên trang chủ, người dùng phải xem được danh sách các bài viết theo mục định sẵn
  + Người dùng có khả năng đang nhập vào tài khoản của mình nếu có tài khoản.
  + .Người dùng có thể đăng kí tài khoản trang web để có thể sử dụng trong việc đăng bài hướng dẫn chia sẻ và các công việc khác mà người dùng có thể làm được như:
* Theo dõi bài viết của người dùng khác
* Xóa bài viết
* Tìm kiếm bài viêt
* Xem các bình luận trên bài viết của mình
* Xem tất cả các bài mình đã đăng.

# : CƠ SỞ LÝ THUYẾT

## Nodejs Express framework:

Về cơ bản express 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 web hoặc mobile. Express có vô vàn các package hỗ trợ đồng thời cũng hỗ trợ hiệu suất và cung cấp them các tính nang để có thể lập trình tốt hơn cho các nhà phát triển mà không làm giảm tốc độ của NodeJS.

Về khía cạnh cơ bản, Express có một số ưu điểm vượt trội như:

* + Express hỗ trợ chúng ta phát triển ứng dụng theo mô hình MVC.
  + Cho phép định nghĩa các middleware.
  + Định nghĩa rõ ràng các request methods trong route.
  + Hỗ trợ mạnh về REST API.
  + Và còn nhiều ưu điểm nữa, nhưng tạm thời những ưu thế kia được team cân nhắc và áp dụng vào trong suốt quá trình phát triển ứng dụng web server.

Do vốn dĩ Express không hề được tích hợp sẵn vào trong nodejs nên muốn sử dụng nó chúng ta phải tải nó về mới sử dụng được. Và để cài đặt nó chúng ta có thể sử dụng cú pháp của npm (node package management).

npm install express --save

Và sau khi cài đặt xong thì trong thư mục project sẽ có thêm folder node\_modules – đây là thư mục chứa các package download từ npm.

## Nodejs

Là javascript phía server, được sử dụng để phát triển các server web với tốc độ truy vấn cao. Sử dụng Google V8 Javascript engine để thực thi mã và một tỷ lệ lớn các module cơ bản được phát triển bằng javascript.

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

Nodejs 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 nodejs đều không đồng bộ (asynchronous). Do đó các tác vụ đều được xử lý và thực thi ở chế độ nền (background processing).

**Nhược điểm:**

1. *Ứng dụng nặng tốn tài nguyên. Nếu bạn cần xử lý các ứng dụng tốn tài nguyên như encoding video, convert file, decoding encryption ... hoặc các ứng dụng tương tự như vậy thì không nên dùng nodejs. Bởi vì Nodejs được viết bằng C++ và javascript nên phải thông qua thêm một trình biên dịch của nodejs nên sẽ lâu hơn 1 chút.*
2. *Nodejs và ngôn ngữ khác như PHP, Ruby, .NET, ... thì việc cuối cùng là phát triển các app web. Nodejs cũng chỉ là ngôn ngữ mới sơ khai như các ngôn ngữ lập trình khác. Vậy nên bạn đừng hy vọng nó sẽ hơn những ngôn ngữ kia nhưng có thể chắc chắn rằng bạn sẽ phát triển được một web app bằng nodejs như mong đợi.*

**Ưu điểm:**

1. *Nhận và xử lý nhiều kết nối chỉ với một single-thread. Điều này giúp cho hệ thống ngốn ít RAM nhất có thể và chạy nhanh vì nó không phải tạo thread mới cho mỗi truy vấn giống PHP. Ngoài ra, tận dụng ưu điểmnon-blocking I/O của Javascript mà nodejs tận dụng tối đâ tài nguyên của server mà không tạo ra độ trễ (như PHP ☺ ).*
2. *JSON APIs với cơ chế event-driven, non-blocking I/O và mô hình kết hợp với javascript là sự lựa chọn tuyệt vời cho các dịch vụ Webs làm bằng JSON.*
3. *Hoàn toàn phù hợp với những app trên một trang (như Gmail) vì với khả năng xử lý nhiều request/s đồng thời phản hồi nhanh. Nếu các ứng dụng mà không muốn tải lại trang và nhận được một lượng lớn các request từ người dùng cần sự hoạt động nhanh để thể hiện sự chuyên nghiệp thì nodejs sẽ là lựa chọn của bạn.*
4. *Shelling tools unix Nodejs sẽ tận dụng tối đa Unix để hoạt động - ??? Có thể hiểu là Nodejs có hoàn toàn khả năng xử lý hàng nghìn process và thả ra một luồng khiến cho hiệu suất hoạt động đạt mức tối đa nhất và tuyệt vời nhất.*
5. *Streaming Data (Luồng dữ liệu) Các web thông thường gửi HTTP requesst và nhận phản hồi lại (Luồng dữ liệu). Gỉa sử sẽ cần một hàm xử lý một luồng dữ liệu cực lớn, Nodejs sẽ xây dựng các Proxy phân vùng các luồng dữ liệu để đảm bảo tối đa hoạt động cho các luồng dữ liệu khác.*
6. *Ứng dụng web sử dụng thời gian thực (Real-time applications) như ứng dụng chat, các dịch vụ mạng xã hội như Facebook, Twitter, ...*

## RESTful API

Có thể nói Nodejs thực sự tỏa sáng một cách chói lóa trong việc xây dựng RESTful API (json) .Gần như không có ngôn ngữ nào xử lý JSON dễ dàng hơn javascript, hơn nữa nodejs xử lý non-blocking nên chẳng còn gì thích hợp hơn nodejs trong trường hợp xây dựng ứng dụng restful api.

Một cách cơ bản và khái quát hóa mà nói thì RESTful API sử dụng các yêu cầu HTTP để thực hiện bốn hoạt động được gọi là CRUD (Create Read Update Delete).

Một điều lưu ý là không có khác biệt lớn giữa REST hay RESTful. REST là viết tắt của cụm từ Representational State Transfer và các ứng dụng sử dụng kiểu thiết kế REST thì được gọi là RESTful (hiểu nôm na thì –ful là tiếp vị ngữ giống như beauty và beautiful). Tất nhiên có thể sử dụng thuật ngữ REST thay cho RESTful và ngược lại.



Để hiểu rõ hơn chúng ta sẽ đi qua phần **Resource**

**Resource**

Việc quản lý tài nguyên là một phần quan trọng và chiếm phần lớn trong việc phát triển website. Trong đó tài nguyên của các trang web khác nhau có thể sẽ khác nhau.Với các trang mạng xã hội như facebook thì resource thường là danh sách người dùng (user hoặc account), danh sách các bài viết, etc..... Đối với các trang web bán hàng thì thường là danh sách sản phẩm, danh sách người bán, danh sách khách hàng,....

Việc quản lý resource của một website bao gồm 4 tác vụ chính:

* Tạo mới một resource (create)
* Lấy thông tin một resource (read)
* Cập nhật một resource (update)
* Xóa một resosurce (delete)

Có rất nhiều cách khác nhau để xây dựng một trang web thực hiện cả 4 tác vụ trên. Ví dụ, với một trang blog chạy dưới tên miền là<http://MikuBlog.wordpress.com> thì để xem một bài viết với ID là 123 thì có thể làm theo một trong các cách sau:

* + Gửi một request tới URL <http://MikuBlog.wordpress.com/posts?id=123> với HTTP method là GET.
  + Gửi một request tới URL <http://MikuBlog.wordpress.com/posts/123> với HTTP method là GET.
  + Gửi một request tới URL <http://MikuBlog.wordpress.com/action=view_post&id=123> với HTTP method là GET.
  + Gửi một request tới URL <http://MikuBlog.wordpress.com/view_post&id=123> với HTTP method là GET.
  + .....

Và sau một thời gian nữa thì người ta đã thống nhất ra các tiêu chuẩn khác nhau để thực hiện việc quản lý tài nguyên. Các tiêu chuẩn này (hay còn được gọi là WEB API hoặc HTTP API) quy định một cách thống nhất việc quản lý các tài nguyên của website và RESTful là một trong số đó.

Vậy RESTful API là một tiêu chuẩn dùng trong việc thiết kế các API trong các ứng dụng web để quản lý các tài nguyên.

Trọng tâm của REST quy định cách sử dụng các HTTP method (như GET, POST, PUT, DELETE ... ) và cách định dạng các URL cho ứng dụng web để quản lý các tài nguyên. Ví dụ với một trang blog quản lý các bài viết chúng ta có các URL đi với các method như sau:

* + URL tạo bài viết: <http://MikuBlog.wordpress.com/posts>. Tương ứng với http method là POST.
  + URL đọc bài viết với ID là 123: <http://MikuBlog.wordpress.com/posts/123>. Tương ứng với http method là GET.
  + URL cập nhật bài viết với ID là 123: <http://MikuBlog.wordpress.com/posts/123>. Tương ứng với http method là PUT.
  + URL xóa bài viết với ID là 123: <http://MikuBlog.wordpress.com/posts/123>. Tương ứng với http method là DELETE.

Với các ứng dụng web được thiết kế sử dụng RESTful, người phát triển có thể dễ dàng biết được URL và http method để quản lý một tài nguyên.Bạn cũng cần lưu ý bản thân RESTful không quy định logic code ứng dụng và RESTful cũng không giới hạn bởi ngôn ngữ lập trình ứng dụng.Bất kì ngôn ngữ lập trình (hoặc framework) nào cũng có thể áp dụng RESTful trong việc thiết kế API cho ứng dụng web.

## JSON Web Token

JSON Web Token (JWT)là một tiêu chuẩn mở (RFC 7519) định nghĩa cách thức truyền tin an toàn giữa các thành viên bằng một đối tượng JSON.Thông tin này có thể được xác thực và đánh dấu tin cậy nhờ vào “chữ ký” của nó. Phần chữ ký của JWT sẽ được mã hóa bằng HMAC hoặc RSA.

Đặc điểm nổi bật của JWT:

1. Kích thước nhỏ: JWT có thể được truyền thông qua URL hoặc qua giao thức POST hay nhét vào bên trong phần HTTP Header. Kích thước nhỏ hơn ứng với việc truyền tải thông tin nhanh hơn. Dưới đây là cách thức truyền token vào trong HTTP Header sử dụng bảng mẫu Bearer (ví dụ):

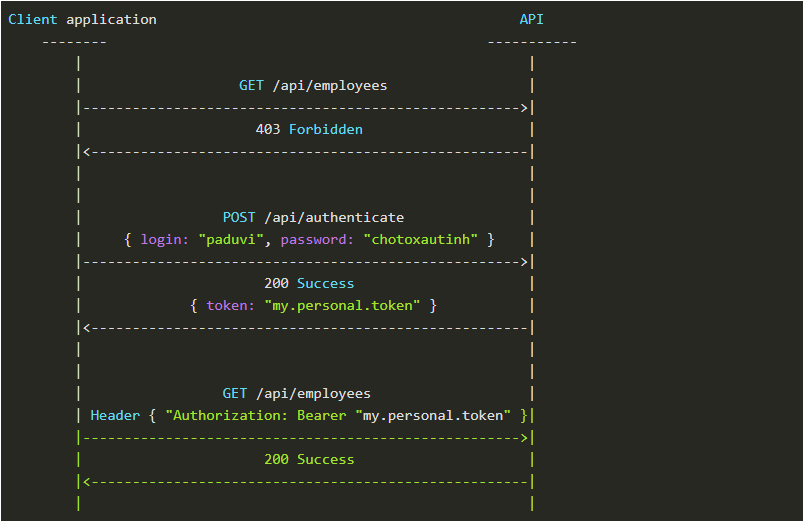
Authorization: Bearer <token>

1. Khép kín: phần Payload chứa toàn bộ những thông tin mà chúng ta cần tới, ví dụ như thông tin của người dùng (thay vì phải truy vấn cơ sở dữ liệu nhiều lần).

***Trường hợp sử dụng JSON Web Token :***

* **Authentication**: Đây là kịch bản muôn đời của json web token. Một khi người dùng đăng nhập vào hệ thống thì những request tiếp theo từ phía người dùng sẽ chứa thêm mã JWT, cho phép người dùng có quyền truy cập vào các đường dẫn, dịch vụ và tài nguyên mà cần có sự cho phép nếu có mã Token đó.Và phương pháp này không dùng cookie !!

**VD:**



* **Trao đổi thông tin**: JSON Web Token là một cách thức không hề tồi ☺ để truyền tin giữa các thành viên với nhau, nhờ vào phần “chữ ký” của nó. Phía người nhận có thể biết được người gửi là ai thông qua phần chữ ký.Ngoài ra chữ ký được tạo ra bằng việc kết hợp cả phần header và phần payload nên người ta có thể xác nhận được chữ ký có bị giả mạo hay không.

**Sơ lược cấu trúc của JSON Web Token:**

JSON Web Token bao gồm ba phần và tách nhau bởi dấu chấm (.):

1. Header
2. Payload
3. Signature (chữ ký)

Dạng tổng quát của JWT : xxxxx.yyyyy.zzzzz

1. **Về phần Header**

Phần header dùng để khai báo kiểu chữ ký và thuật toán mã hóa sẽ dùng cho cái token cảu chúng ta.

Ví dụ cho phần header :

{

“alg” : “HS256”,

“typ” : “JWT”

}

Đoạn header này khai báo rằng đối tượng được mã hóa là một JWT (Để phân biệt với JWS hay JWE) và chữ ký của nó sử dụng thuật toán mã hóa HMAC SHA-256.

Đoạn header này sẽ được mã hóa base64url và ta thu được phần đầu tiên của JWT:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

Chú ý rằng code ở phía trên là **base64url** chứ không phải là **base64**. Về cơ bản hai cái này tương tự nhau nhưng base64url vẫn có những sự khác biệt là :

* Không thêm = vào
* Các ký tự ***+*** và / sẽ được thay thế bằng – và \_

Chúng ta có thể tự triển khai phát triển một thuật toán mã hóa base64url. Dưới dây là một thuật toán mô phỏng bằng Javascript:

function base64url (source) {

//Encode in classical base64

encodedSource = CryptoJS.enc.Base64.stringify (source);

//Remove padding equal characters

encodedSource = encodedSource.replace ( /=+$/ , ‘’ );

//Replace characters according to base64url specifications

encodedSource = encodedSource.replace ( /\+/g , ‘-‘ );

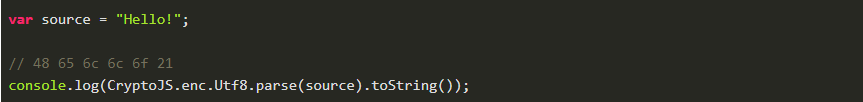
encodedSource = encodedSource.replace ( /\//g , ‘\_’ );

return encodedSource;

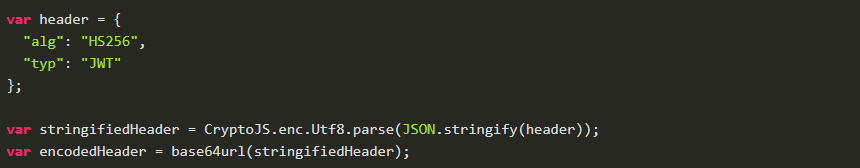
}

Ở đoạn code trên chúng ta đã sử dụng thư viện CryptoJS để có thể mã hóa base64 rồi sau đó loại bỏ các ký tự = và thay thế các ký tự + / đi.

Để có thể sử dụng được hàm trên, đầu vào của hàm cần là một mảng byte ở định dạng UTF-8. Ta có thể chuyển đổi từ xâu ký tự sang mảng byte bằng một hàm khác cũng được cung cấp bởi CryptoJS:



Và cuối cùng ta đã thu được phần đầu tiên của JWT:



1. **Về phần Payload**

Về phần thứ hai của token đó là Payload, nơi chứa các nội dung của thông tin. Thông tin truyền đi có thể là mô tả của một thực thể (ví dụ như người dùng) hoặc cũng có thể là các thông số bổ sung thêm cho phần Header. Nhìn chung, chúng được chia làm 3 loại: Reserved, Public và Private.

1. **Reserved:** là những thông tin đã được quy định trong IANA JSON Web Token Claims registry. Chúng bao gồm:

* **iss (issuer)** : tổ chức phát hành token
* **sub (subject) :** chủ đề của token
* **aud (audience)** : đối tượng sử dụng token
* **exp (expired time)** : thời điểm mà token sẽ hết hạn
* **nbf (not before time)** : token sẽ chưa hợp lệ trước thời điểm này
* **iat (issued at)** : thời điểm token được phát hành, tính theo UNIX time
* **iji** : JWT ID

**(**Chú ý rằng các khóa của claims đều chỉ dài 3 ký tự vì mục đích giảm kích thước của token**)**

1. **Public:** Khóa nên được quy định ở IANA JSON Web Token Registry hoặc là một URL có chức không gian tên không bị trùng lặp.

Ví dụ:

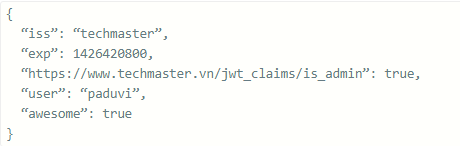


1. **Private:** Phần thông tin thêm dùng để truyền qua giữa các máy thành viên.

Ví dụ:



**Ví dụ cho phần Payload:**

****

Mã hóa base64url ta thu được phần thứ hai của token:

eyJpc3MiOiJ0ZWNobWFzdGVyIiwiZXhwIjoxNDI2NDIwODAwLCJodHRwczovL3d3dy50ZWNobWFzdGVyLnZuL2p3dF9jbGFpbXMvaXNfYWRtaW4iOnRydWUsInVzZXIiOiJwYWR1dmkiLCJhd2Vzb21lIjp0cnVlfQ

1. **Về phần Signature:**

Phần chữ ký được tạo bằng cách kết hợp hai phần header và payload rồi mã hóa nó lại bằng một giải thuật encode nào đó, càng phức tạp càng tốt, ví dụ như HMAC SHA-256

Rồi ta sẽ thu được phần cuối của token:

Cuối cùng khi ta đã có đủ ba phần thì ta sẽ đặt chúng vào cùng với nhau và ta sẽ có JWT là:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0ZWNobWFzdGVyIiwiZXhwIjoxNDI2NDIwODAwLCJodHRwczovL3d3dy50ZWNobWFzdGVyLnZuL2p3dF9jbGFpbXMvaXNfYWRtaW4iOnRydWUsInVzZXIiOiJwYWR1dmkiLCJhd2Vzb21lIjp0cnVlfQ.uL7nEjM7ihbQe7l01rmQCtGYoKyb4VyabWqX8PZKdt4

Và đây là mẫu code javascript triển khai toàn bộ công việc trên:

**var header = {**

"alg":"HS256",

"typ": "JWT"

};

**var** stringifiedHeader = CryptoJS.enc.Utf8.parse(JSON.stringify(header));

**var** encodedHeader = base64url(stringifiedHeader);

**var** data = {

"iss": "techmaster",

"exp": 1426420800,

"https://www.techmaster.vn/jwt\_claims/is\_admin": **true**,

"user": "paduvi",

"awesome": **true**

};

**var** stringifiedData = CryptoJS.enc.Utf8.parse(JSON.stringify(data));

**var** encodedData = base64url(stringifiedData);

**var** token = encodedHeader + "." + encodedData;

**var** secret = "My very confidential secret!";

**var** signature = CryptoJS.HmacSHA256(token, secret);

signature = base64url(signature);

**var** signedToken = token + "." + signature;

Code mẫu trên theo nguồn : <https://techmaster.vn/posts/33959/khai-niem-ve-json-web-token>

## Multer

Một gói khá hữu ích của nodejs được tung ra phiên bản đầu tiên vào năm 2013. Là một middleware kiểm soát việc đăng tải dữ liệu kiểu multipart/form-data.

Cách cài đặt khá đơn giản, sử dụng dòng lệnh npm như sau :

$ npm install –save multer

Cách dùng :

Multer thêm đối tượng *body* vàđối tượng *file* hoặc đối tượng *files* đến đối tượng *request*. Đối tượng *request* sẽ chứa các giá trị của các input của form html, đối tượng *file* hoặc đối tượng *files* sẽ chứa các file được tải lên thông qua form.

Ví dụ:

Chú ý đừng quên mục enctype=”multipart/form-data” trong cái form

<form **action**="/profile" **method**="post" **enctype**="multipart/form-data">

  <input **type**="file" **name**="avatar" />

</form>

Và trong file server:

var express **=** require('express')

var multer  **=** require('multer')

var upload **=** multer({ dest**:** 'uploads/' })

var app **=** express()

app.post('/profile', upload.single('avatar'), function (req, res, next) {

*// req.file is the `avatar` file*

*// req.body will hold the text fields, if there were any*

})

app.post('/photos/upload', upload.array('photos', 12), function (req, res, next) {

*// req.files is array of `photos` files*

*// req.body will contain the text fields, if there were any*

})

var cpUpload **=** upload.fields([{ name**:** 'avatar', maxCount**:** 1 }, { name**:** 'gallery', maxCount**:** 8 }])

app.post('/cool-profile', cpUpload, function (req, res, next) {

*// req.files is an object (String -> Array) where fieldname is the key, and the value is array of files*

*// e.g.* *:  req.files['avatar'][0] -> File*

*//  req.files['gallery'] -> Array*

*// req.body will contain the text fields, if there were any*

})

Trong trường hợp cần kiểm soát form multipart mà chỉ có chữ, không có dữ liệu media thì nên sử dụng phương thức .none():

Ví dụ:

var express **=** require('express')

var app **=** express()

var multer  **=** require('multer')

var upload **=** multer()

app.post('/profile', upload.none(), function (req, res, next) {

*// req.body contains the text fields*

})

**Về API của multer**:

**-Thông tin file**

Mỗi một file được upload sẽ có những thông tin sau:

|  |  |  |
| --- | --- | --- |
| Key | Description | Note |
| fieldname | Tên cụ thể của trường trong form |  |
| originalname | Tên file trên máy tính của người dùng |  |
| encoding | Mã hóa loại file |  |
| mimetype | mime type of the file | (nhóm chưa hiểu rõ về loại thông tin này) |
| size | Kích cỡ của file theo đơn vị bytes |  |
| destination | Folder lưu file được tải lên | DiskStorage |
| filename | Tên của file khi được lưu vào mục destination ở trên | DiskStorage |
| path | Đường dẫn đầy đủ đến nơi file được lưu trữ | DiskStorage |
| buffer | Buffer của toàn bộ file | MemoryStorage |

.**Mục tùy chọn của multer**

Multer chấp nhận đối tượng tùy chọn, thuộc tính cơ bản nhất là *dest (trong destination)* , thuộc tính này sẽ cho multer biết nơi trong project dùng để lưu trữ file upload.Trong trường hợp quên cung cấp, file được tải lên sẽ giữ trong bộ nhớ và không bao giờ được ghi ra ổ đĩa.

Theo mặc định thì multer sẽ đặt lại tên cho các file được tải lên để tránh mâu thuẫn như trùng tên.Hàm đặt lại tên có thể được tùy chỉnh theo nhu cầu cá nhân của người phát triển.

Các mục tùy chọn sau có thể được dùng trong multer:

|  |  |
| --- | --- |
| Key | Description |
| *dest* or *storage* | Định nghĩa nơi lưu trữ file |
| *fileFilter* | Hàm kiểm soát những file được chấp nhận |
| *limits* | Giới hạn dữ liệu được tải lên |
| *preservePath* | Giữ nguyên đường dẫn của file thay vì để tên của file |

Trong các web app thông thường, chỉ có tùy chọn *dest* là có thể phải bắt buộc.Được cấu hình theo cách sau (thông thường):

var upload **=** multer({ dest**:** 'uploads/' })

Nếu như nhà phát triển muốn kiểm soát toàn bộ dữ liệu được tải lên, họ phải dùng tùy chọn *storage* thay vì *dest*.

Các hàm phổ biến:

#### .single(fieldname)

-Chấp nhận file tải lên duy nhất, nghĩa là khi form có tới hai input dạng file nhưng nó chỉ lấy một trường tải lên duy nhất. File đơn này sẽ được lưu trữ trong req.file.

#### .array(fieldname[, maxCount])

-Chấp nhận một mảng các files, miễn cùng tên trường fieldname trong form. Ùy chọn này sẽ gây ra lỗi nếu như số lượng file tải lên vượt quá biên *maxCount* được nêu trên. Mảng các file này sẽ được lưu trữ trong req.files.

#### .fields(fields)

-Chấp nhận một chuỗi các files, cụ thể là các trường. Đối tượng với mảng các file này sẽ được lưu trữ trong req.files.

[

{ name: 'avatar', maxCount: 1 },

{ name: 'gallery', maxCount: 8 }

]

#### .none()

-Chấp nhận duy nhất dữ liệu text.Nếu như có bất kì file được tải lên nào thì sẽ gây ra lỗi “LIMIT\_UNEXPECTED\_FILE” .

#### .any()

-Chấp nhận tất cả các file được cung cấp đến.Một mảng các file này sẽ được lưu trữ trong req.files.

**Warning : Hãy chắc chắn rằng nhà phát triển phải luôn kiểm soát được các file được tải lên bởi người dùng. Đừng thêm multer vào project với vị trí là một global middleware nếu không muốn người dùng độc hại tải lên file gì đó và được truyền thẳng vào route. Chỉ nên dùng hàm .any() trong route một khi đã kiểm soát được các file được tải lên.**

.**Mục lưu trữ của multer**

* **DiskStorage**

Engine này cho bạn toàn quyền kiểm soát để lưu trữ file vào ổ đĩa.

Code mẫu bên dưới:

var storage = multer.diskStorage({

destination: function (req, file, cb) {

cb(null, '/tmp/my-uploads')

},

filename: function (req, file, cb) {

cb(null, file.fieldname + '-' + Date.now())

}

})

var upload = multer({ storage: storage })

Code mẫu bên trên sử dụng hai tùy chọn có sẵn là *destination* và *filename* . Cả hai tùy chọn đều là hàm xác định hay định nghĩa vị trí lưu file lên khi được tải lên.

-Trong khi *destination* được sử dụng để xác định hay định nghĩa trong thư mục nào các file tải lên sẽ được lưu trữ. Vụ này cũng có thể được cung cấp bởi dạng chuỗi như (vd: ‘/MikuBest/uploads’). Nếu *destination* không được định nghĩa, thì thư mục lưu mặc định của hệ điều hành đang dùng sẽ định nghĩa là folder temporary.

Lưu ý: Nhà phát triển chịu trách nhiệm hoàn toàn trong việc tạo chỉ mục khi cung cấp *destination* trong hàm. Khi đưa chuỗi (chuỗi đường dẫn của folder), multer sẽ đảm bảo rằng các đường dẫn folder *destination* đó đã được tạo.

-Khác với *destination*, *filename* được sử dụng để xác định xem file sẽ được đặt tên gì trong cái folder *destination*. Nếu tên không được định nghĩa bởi người phát triển thì mỗi file được up lên sẽ được “biên chế” cho một các tên ngẫu nhiên, đương nhiên tên sẽ không bao gồm tên mở rộng của file như *.png* hay *.mp4* .

Lưu ý: Multer sẽ không nối thêm tên mở rộng, người phát triển nên cung cấp hoàn toàn đẩy đủ tên file bao gồm các phần mở rộng.

* **MemoryStorage**

Engine này lưu trữ các file trong bộ nhớ dưới dạng đối tượng Buffer và nó không có tùy chọn thêm nào.

var storage = multer.memoryStorage()

var upload = multer({ storage: storage })

Khi sử dụng memorystorage. Các file sẽ chứa trường gọi là buffer, nó sẽ chứa toàn bộ nội dung file.

**WARNING**: Khi tải lên file dung lượng lớn hoặc file có dung lượng nhỏ với số lượng quá nhiều có thể khiến cho tài nguyên bộ nhớ máy tính lúc đó bị cạn kiệt và mất nhiều thời gian phản hồi.

* **Limits**

Đây là đối tượng sẽ chỉ định giới hạn kích cỡ của file và hiển nhiên đối tượng này là một tùy chọn.

Các thông số sau là có sẵn trong limits

|  |  |  |
| --- | --- | --- |
| Key | Description | Default |
| fieldNameSize | Kích cỡ tên file tối đa | 100 bytes |
| fieldSize | Kích cỡ giá trị file tối đa | 1 MB |
| fields | Số lượng tối đa các trường không dùng để up file | Infinity |
| fileSize | Dành cho form multipart, kích cỡ file tối đa (trong đơn vị bytes) | Infinity |
| files | Dành cho form multipart, số lượng các trường cho phép tải file | Infinity |
| parts | Dành cho form multipart, số lượng tối đa của các bộ phận (fields + files) | Infinity |
| headerPairs | Dành cho form multipart, số lượng tối đa của các khóa header sang các cặp giá trị để chuyển đổi | 2000 |

Xác định giới hạn sẽ giúp cho website của bạn có thể chống lại được kiểu tấn công từ chối dịch vụ.

* **fileFilter**

Thêm cái này vào hàm upload để kiểm soát được file nào nên được tiếp tục đưa lên và file nào phải bỏ đi. Cái hàm sẽ trông như thế này:

function fileFilter (req, file, cb) {

// The function should call `cb` with a boolean

// to indicate if the file should be accepted

// To reject this file pass `false`, like so:

cb(null, false)

// To accept the file pass `true`, like so:

cb(null, true)

// You can always pass an error if something goes wrong:

cb(new Error('I don\'t have a clue!'))

}

* **Error handling**

Khi gặp phải lỗi thì Multer sẽ ủy thác cái lỗi này lại cho Express. Người phát triển có thể làm màu hay có thể nói là họ có thể thiết kế một trang hay ho để hiển thị lỗi. Nếu muốn họ có thể tùy chọn và sử dụng tiêu chuẩn của Express.

Nếu họ muốn bắt một cái lỗi cụ thể từ phía Multer, họ có thể tự gọi hàm middleware. Cũng vậy, nếu họ muốn mắt duy nhất lỗi multer, họ có thể dùng *The Multer errors,* họ có thể dùng lớp MulterErrorđể đính kèm đối tượng multer vào chính nó (ví dụ: err instanceof multer.MulterError).

Code mẫu phía dưới:

var multer = require('multer')

var upload = multer().single('avatar')

app.post('/profile', function (req, res) {

upload(req, res, function (err) {

if (err instanceof multer.MulterError) {

// A Multer error occurred when uploading.

} else if (err) {

// An unknown error occurred when uploading.

}

// Everything went fine.

})

})

## Gói Body-parser của Express

Để hiểu rõ hơn về “quyền năng” của gói này, chúng ta nên đi qua những vấn đề mà team đã gặp phải .....

Trong những ngày đầu viết web server cho đồ án, đã có trường hợp gặp phải như là không tài nào bắt được dữ liệu từ bên form gửi sang. Do vẫn ảnh hưởng tư tưởng từ bên Nodejs thuần, tức là bạn chỉ cần dùng parameter *request* và gọi một field nào đó từ form bên client từ url nhận được. Thế nhưng nếu bạn dùng kĩ thuật đó tại Express thì những gì bạn đạt được chỉ là *undefined.*

Đoạn code ví dụ:

-Phía server:

app.post(\'/\', function(req, res) {

console.log(req.body.email);

});

-Phía client:

<form *class=*"login-form" *action=*"signup" *method=*"post" *name=*"form-login">  
 *<!-- username -->* <div *class=*"username">  
 <label><span *class=*"entypo-user" /></label>  
 <input *type=*"email" *placeholder=*"Email" *name=*"email" *id=*"email"/>  
 </div>  
 *<!-- password -->* <div *class=*"password">  
 <label><span *class=*"entypo-lock" /></label>  
 <input *type=*"password" *placeholder=*"Password" *name=*"password" *id=*"password"/>  
 </div>  
 *<!-- confirm password -->* <div *class=*"password">  
 <label><span *class=*"entypo-lock" /></label>  
 <input *type=*"password" *placeholder=*"Confirm password" *name=*"password1" *id=*"password1" *onkeyup=*"CheckConfirmPass()"/>  
 <script>  
 *function* CheckConfirmPass() {  
 *let* first\_pass = document.getElementById("password").value;  
 *let* second\_pass = document.getElementById("password1").value;  
 *let* email = document.getElementById("email").value;  
  
 document.getElementById("submit").disabled = email === "" && first\_pass === "" && second\_pass === "";  
 document.getElementById("submit").disabled = first\_pass !== second\_pass;  
 }  
 </script>  
 </div><input *type=*"submit" *value=*"Join us" *class=*"btn" *id=*"submit" *disabled*/>  
 <p>  
 Already have an account? <a *href=*"/login" *class=*"link" *style=*"margin-left: 10px;">Sign in now <span *class=*"entypo-right-thin" /></a>  
 </p>  
</form>

Và kết quả trả về thực sự là *undefined* mặc dù đã làm theo đúng trình tự.

-Điều tiếp theo sau đó là cố gắng kiểm tra object của *req*. Cố gắng tìm kiếm sự tồn tại của bất kì data nào trong đó nhưng object của req thật sự quá lớn và khó hiểu nữa nên từ bỏ. Và sau khi tìm kiếm sự trợ giúp từ stackoverflow, tôi đã tìm ra một giải pháp là dùng gói body-parser của express.

-Trên hết để sử dụng được body-parser , người phát triển cần dùng đoạn code như sau:

app.use(bodyParser.json());

Để hiểu cách thức hoạt động của chúng, chúng ta phải hiểu được cách thức hoạt động của middleware trong Express ra sao. Cụ thể, khi *app.use()*  được sử dụng cùng với các đối số cơ bản như *req* và *res*:

Code ví dụ:

app.use(function(req, res) {

// make somethings

});

* Function trên sẽ hoạt động với mọi lượt request
* Function này sẽ hoạt động như một middleware

Nếu như ai đó muốn middleware trên chỉ hoạt động với một request cụ thể nào đó thì người đó chỉ cần thêm đường dẫn cụ thể vào.

Code ví dụ:

app.use(\'/test\', cb);

Với đoạn code trên, middleware chỉ hoạt động khi đường dẫn match với */test*

Có thể hiểu rằng body-parser hoạt động bằng cách nó sẽ trả về một function và chính function đó sẽ được dùng làm đối số cho app.use, nó hoạt động giống như bất kì middleware nào khác.

Ví dụ:

var cb = bodyParser.json();

app.use(cb);

Chức năng lắng nghe trên *req.on(’data’)* và xây dựng *req.body* từ các đoạn dữ liệu mà nó nhận được.

Nhưng có một số lưu ý. Về cơ bản, có nhiều cách khác nhau để định dạng dữ liệu khi người dùng POST dữ liệu đến server:

* application/x-www-form-urlencoded
* multipart/form-data
* application/json
* application/xml
* and maybe some others

Tóm tắt lại, bodyParser phải phân tích dữ liệu mỗi loại một cách khác nhau. ***bodyParser*** hỗ trợ các function khác nhau để làm điều này.

// for parsing application/json

app.use(bodyParser.json());

// for parsing application/x-www-form-urlencoded

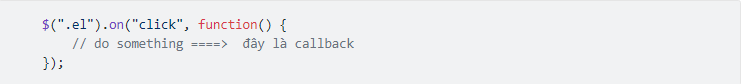
app.use(bodyParser.urlencoded({ extended: true }));

// for parsing multipart/form-data

app.use(multer());

Vậy bằng việc sử dụng bodyParser bạn hoàn toàn có thể lấy được dữ liệu post từ client.

## CallBack trong javascript

Callback là một đoạn code được truyền vào như một tham số của một hàm và chờ để được gọi vào thực thi. Javascript là một ngôn ngữ lập trình hướng sự kiện và bất đồng bộ nên callback function đóng vai trò rất quan trọng, bạn sẽ truyền một callback function vào các sự kiện và xử lý bất đồng bộ đó. Một ví dụ quen thuộc mà hầu như ai cũng đã từng sử dụng mà chỉ có điều là chúng ta không để ý tới. Ví dụ dưới đây trong việc bắt một sự kiện onClick cho một element nào đó:

Để dễ hiểu hơn, hàm trên chúng ta thấy đã truyền một function như một parameter cho hàm *on*.

Thời điểm mà thực thi đoạn code truyền vào phụ thuộc vào hàm đó đươc định nghĩa như thế nào.Để được minh bạch vấn đề trên, chúng ta đi đến ví dụ tiếp theo là về hàm *SayHello()*

function *sayHello*(name, callback){

let myName = name.toUpperCase() + “, Hello”;

return callback(myname);

}

let result = sayHello(“Vũ”, function (arg) {

return arg;

});

alert(result);

Có thể kết luận rằng thời điểm mà callback được thực thi là thời điểm sau khi biến myName được thực hiện và khi chúng ta gọi *sayHello* trả về sẽ alert với nội dung là “ Vũ, Hello”.

**Một số lưu ý khi sử dụng callback:**

* **Callback phải là một function, bạn phải truyền vào một function hoặc nếu bạn truyền loại dữ liệu khác thì bạn chỉ nhận lỗi “Callback must be a function” mà thôi (tin đi, gặp bug dạng này miết)**
* **Từ khóa this: Trong hàm callback phải luôn cận trọng với từ khóa this. Vì hàm callback chỉ là một hàm bình thường nên khi sử dụng nó trong hàm thì nó sẽ hiểu *this* lúc này chính là đối tượng *Window*.**

## Promise của Javascript (chỉ có trong phiên bản ECMAScript 6)

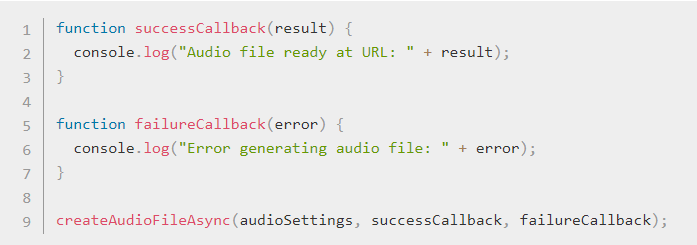
Chúng ta đều biết javascript là ngôn ngữ bất đồng bộ, vì cơ bản, Nodejs là javascript phía server. Chính do đặc tính bất đồng bộ mà server web được phát triển bởi nodejs được chạy và thực thi rất nhanh.

Và cũng do tính bất đồng bộ mà không ít khó khăn đã xảy ra trong giai đoạn phát triển server. Khó khăn thường gặp nhất có lẽ là ***CallbackHell,*** điều sẽ xảy ra khi mà trong một chức năng có quá nhiều hàm callback chồng chéo lên nhau, gây không ít phiền nhiễu, phức tạp hóa thuật toán và rối rắm mọi thứ lên.

Thật may mắn khi Promise được tạo ra để khắc phục tình trạng này. Về cơ bản Promise là một đối tượng đại diện cho sự hoàn thành( thành công hay thất bại của một tiến trình đồng bộ). Vâng, Promise giúp đoạn code javascript của chúng ta dễ nhìn hơn (khi mà nhìn chúng thật thân thuộc như giống như các ngôn ngữ lập trình đồng bộ yêu thích cảu hầu hết chúng ta ).

Chúng ta có thể hiểu rằng promise là một đối tượng trả về mà bạn gắn callback vào nó thay vì truyền callback vào trong một hàm.

Sau đây là một đoạn code đơn giản về việc tạo file âm thanh bất đồng bộ - *createAudioFileAsync()*



Thay vì sử dụng một đoạn code lằng nhằng như trên - các hàm riêng lẻ sẽ trả về kết quả tương ứng mà bạn gọi vào trong hàm chính tương ứng với mỗi vị trí hàm callback, tượng trưng cho sự thất bại hay thành công của hàm chính.

Chúng ta có thể viết lại hàm trên nhưng chúng ta sẽ thử thêm Promise vào:

Hoặc thậm chí nếu bạn muốn màu mè:



Vậy cách tiếp cận này – dùng Promise – sẽ được lợi gì ?

Sau đây là những ưu điểm cảu việc dùng Promise mà nhóm không thể không cân nhắc tới:

* **SỰ ĐẢM BẢO**

1. Callback sẽ không bao giờ được gọi đến trước khi hoàn tất sự kiện javascript trước đó (điều này đảm bảo việc giống với lập trình đồng bộ).
2. Callback được thêm vào từ khóa *then()* sau khi tiến trình bất đồng bộ đã hoàn thành mà bạn vấn muốn gọi thêm hàm nào đó, và vẫn theo nguyên tắc ở trên.
3. Nếu sử dụng nhiều từ khóa *then()* thì mỗi hàm callback sẽ được gọi theo chế độ ưu tiên queue, lần lượt cứ theo thứ tự mà chúng được thêm vào.

* **GỌI NỐI**
* **GỌI NỐI SAU HÀM *CATCH***
* **XỬ LÝ LỖI TẬP TRUNG**
* **BỐ CỤC ĐOẠN CODE**
* **VẤN ĐỀ THỜI GIAN**
* **VIỆC LỒNG CODE**

## Async/Await

# : PHÂN TÍCH

Chương này sinh viên trình bày quá trình phân tích bài toán hoặc hệ thống từ việc thu thập thông tin, thu nhận yêu cầu ban đầu. Sinh viên có thể trình bày các lược đồ (UML, ERD, …)

## Tiêu đề mục

### Tiêu đề tiểu mục

Nội dung tiểu mục

### Tiêu đề tiểu mục

Nội dung tiểu mục

## Tiêu đề mục

### Tiêu đề tiểu mục

Nội dung tiểu mục

### Tiêu đề tiểu mục

Nội dung tiểu mục

# : THIẾT KẾ VÀ HIỆN THỰC

Chương này sinh viên trình bày quá trình thiết kế, thử nghiệm và hiện thực hệ thống hoặc bài toán.

## Tiêu đề mục

### Tiêu đề tiểu mục

Nội dung tiểu mục

### Tiêu đề tiểu mục

Nội dung tiểu mục

## Tiêu đề mục

### Tiêu đề tiểu mục

Nội dung tiểu mục

### Tiêu đề tiểu mục

Nội dung tiểu mục

# : KẾT LUẬN

## Kết quả đạt được

Trình bày các chức năng đã hoàn thành trong đồ án hoặc kết quả nghiên cứu.

## Hạn chế của đồ án

Trình bày những hạn chế, công việc chưa hoàn tất trong đồ án

## Hướng phát triển

Trình bày các định hướng phát triển cho hệ thống hoặc hướng nghiên cứu trong tương lai.

# 

# TÀI LIỆU THAM KHẢO

Các tài liệu Tiếng Việt

1. Họ và Tên tác giả (Thứ tự theo Tên). Tên sách. Tên nhà xuất bản, năm xuất bản.

Các tài liệu Tiếng Anh

1. Amy Apon. Lecture for Cluster and Grid Computing. University of Arkansas, 2004.

Các tài liệu từ Internet

1. Website: [www.cengageasia.com](http://www.cengageasia.com)
2. ...

# PHỤ LỤC