

**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM**

**Khóa Luận Tốt Nghiệp**

Đề tài : Handmade Instructor

# MỤC LỤC

[MỤC LỤC 1](#_Toc532436353)

[LỜI MỞ ĐẦU 3](#_Toc532436354)

[CHƯƠNG 1 : GIỚI THIỆU 4](#_Toc532436355)

[1.1 Tổng quan 4](#_Toc532436356)

[1.2 Mục tiêu đề tài 4](#_Toc532436357)

[1.3 Phạm vi đề tài 5](#_Toc532436358)

[1.4 Mô tả yêu cầu chức năng 6](#_Toc532436359)

[CHƯƠNG 2 : CƠ SỞ LÝ THUYẾT 7](#_Toc532436360)

[2.1 Nodejs Express framework: 7](#_Toc532436361)

[2.2 Nodejs 7](#_Toc532436362)

[2.3 JSON Web Token 9](#_Toc532436363)

[2.4 Multer 16](#_Toc532436364)

[2.5 Gói Body-parser của Express 25](#_Toc532436365)

[2.6 CallBack trong javascript 29](#_Toc532436366)

[2.7 Promise của Javascript (chỉ có trong phiên bản ECMAScript 6) 30](#_Toc532436367)

[2.8 Async/Await 36](#_Toc532436368)

[2.9 Socket.io 37](#_Toc532436369)

[2.10 Amazon Simple Storage Service (Amazon S3) 41](#_Toc532436370)

[2.11 Amazon DynamoDB 44](#_Toc532436371)

[CHƯƠNG 3 : PHÂN TÍCH 46](#_Toc532436372)

[3.1 Đặc tả use case cho từng chức năng 48](#_Toc532436373)

[3.1.1 UCID: UC1 48](#_Toc532436374)

[3.1.2 UCID: UC2 49](#_Toc532436375)

[3.1.3 UCID: UC3 50](#_Toc532436376)

[3.1.4 UCID: UC4 51](#_Toc532436377)

[3.1.5 UCID: UC5 52](#_Toc532436378)

[3.1.6 UCID: UC6 53](#_Toc532436379)

[3.1.7 UCID: UC7 54](#_Toc532436380)

[3.1.8 UCID: UC8 55](#_Toc532436381)

[3.1.9 UCID: UC9 56](#_Toc532436382)

[3.1.10 UCID: UC10 58](#_Toc532436383)

[CHƯƠNG 4 : THIẾT KẾ VÀ HIỆN THỰC 61](#_Toc532436384)

[4.1 Sơ đồ activity 61](#_Toc532436385)

[4.1.1 Sơ đồ activity cho UC1 61](#_Toc532436386)

[4.1.2 Sơ đồ activity cho UC2 61](#_Toc532436387)

[4.1.3 Sơ đồ activity cho UC3 63](#_Toc532436388)

[4.1.4 Sơ đồ activity cho UC4 64](#_Toc532436389)

[4.1.5 Sơ đồ activity cho UC5 65](#_Toc532436390)

[4.1.6 Sơ đồ activity cho UC6 66](#_Toc532436391)

[4.1.7 Sơ đồ activity cho UC7 67](#_Toc532436392)

[4.1.8 Sơ đồ activity cho UC8 68](#_Toc532436393)

[4.1.9 Sơ đồ activity cho UC9 69](#_Toc532436394)

[4.1.10 Sơ đồ activity cho UC10 70](#_Toc532436395)

[4.2 Sơ đồ sequence 71](#_Toc532436396)

[4.2.1 Sơ đồ sequence cho UC1 71](#_Toc532436397)

[4.2.2 Sơ đồ sequence cho UC2 72](#_Toc532436398)

[4.2.3 Sơ đồ sequence cho UC3 73](#_Toc532436399)

[4.2.4 Sơ đồ sequence cho UC4 74](#_Toc532436400)

[4.2.5 Sơ đồ sequence cho UC5 75](#_Toc532436401)

[4.2.6 Sơ đồ sequence cho UC6 76](#_Toc532436402)

[4.2.7 Sơ đồ sequence cho UC7 77](#_Toc532436403)

[4.2.8 Sơ đồ sequence cho UC8 78](#_Toc532436404)

[4.2.9 Sơ đồ sequence cho UC9 79](#_Toc532436405)

[4.2.10 Sơ đồ sequence cho UC10 80](#_Toc532436406)

[CHƯƠNG 5 : KẾT LUẬN 81](#_Toc532436407)

[5.1 Kết quả đạt được 81](#_Toc532436408)

[5.2 Hạn chế của đồ án 81](#_Toc532436409)

[5.3 Hướng phát triển 82](#_Toc532436410)

[TÀI LIỆU THAM KHẢO 83](#_Toc532436411)

[PHỤ LỤC 84](#_Toc532436412)

# 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:
* Xóa bài viết
* 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, ...*

## 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):

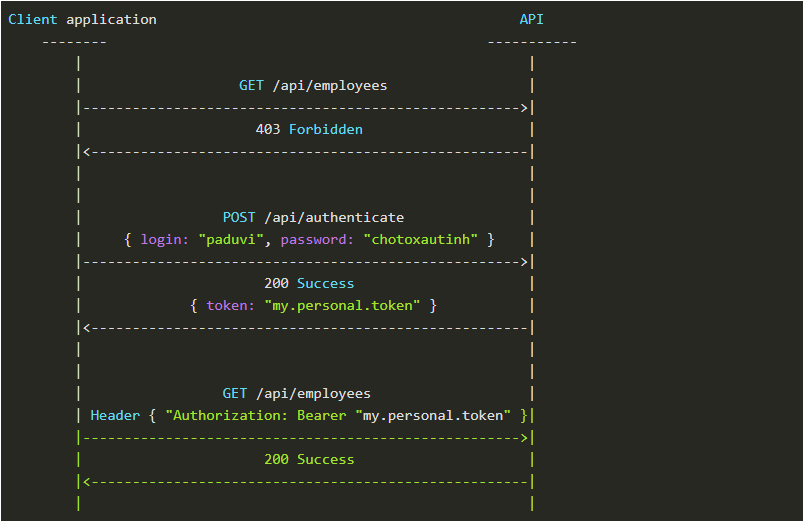
Authorization: Bearer <token>

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 , ‘-‘ );

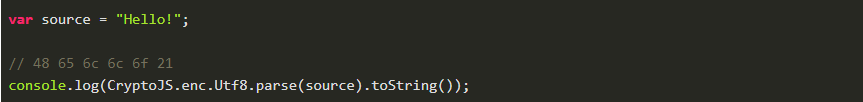
encodedSource = encodedSource.replace ( /\//g , ‘\_’ );

return encodedSource;

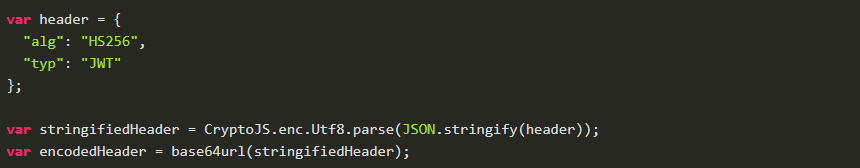
}

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:

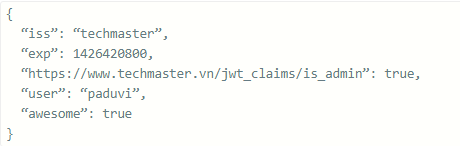


1. **Private:** Phần thông tin thêm dùng để truyền qua giữa các máy thành viên.

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

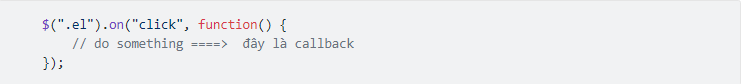
app.use(bodyParser.urlencoded({ extended: true }));

// 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)

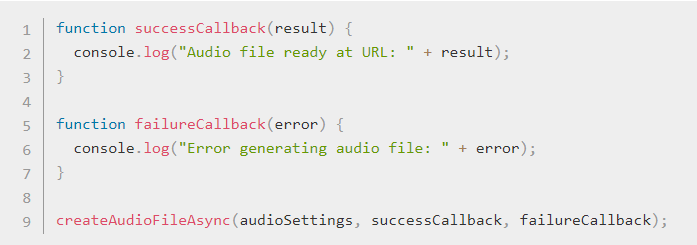
Chúng ta đều biết javascript là ngôn ngữ bất đồng bộ, vì cơ bản, Nodejs là javascript phía server. Chính do đặc tính bất đồng bộ mà server web được phát triển bởi nodejs được chạy và thực thi rất nhanh.

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**

Thật ra trong suốt quá trình phát triển ứng dụng web, nhóm luôn có một nhu cầu về việc có thể thực thi hai - hay nhiều hơn nữa – các tiến trình liên tiếp nhau, caisua bắt đầu ngay khi cái trước hoàn tất, có được kết quả trả về và sau đó được sử dụng ngay với tư cách là một tham số cho tiến trình tiếp theo. May mắn thya vụ này được hỗ trỡ trong Promise, vì một trong những đặc điểm thú vị của promise là khả năng gọi nối – một chuỗi Promise.

Cách hoạt động của nó chủ yếu dựa vào từ khóa *then(),* trả về một promise mới, khác với ban đầu:



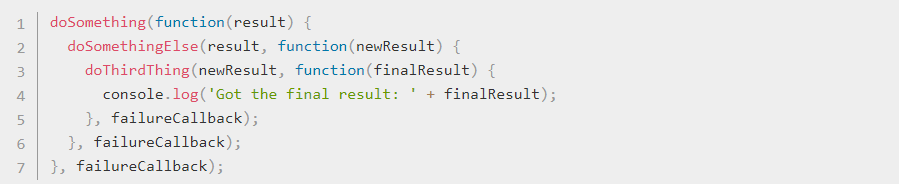
Hoặc:

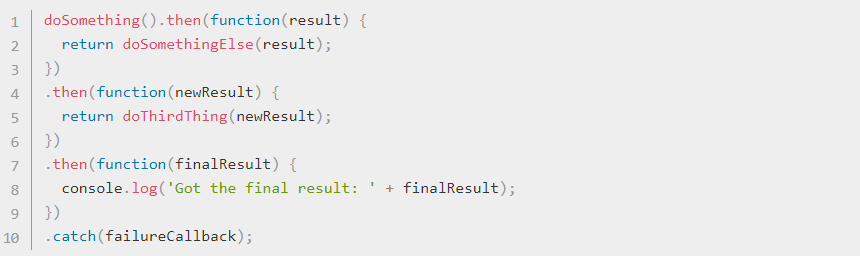


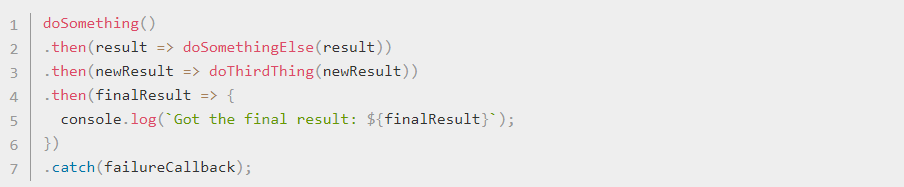
Promise thứ hai ở đây (promise2) ngoài tư cách là đã hoàn thành và đang giữ kết quả của tiến trình trước (promise), còn đang đại diện cho cả *successCallback* hoặc *failureCallback* mà chúng ta đưa vào, hai hàm *successCallback, failureCallback* ngoài ra cũng có thể là các hàm bất đồng bộ khác trả về promise. Trong trường hợp đó, bất kì callback nào được thêm vào promise2 cũng sẽ được xếp phía sau promise, trả về một trong hai *successCallback* hoặc *failureCallback.*

Tóm lại, mối promise đại diện cho việc hoàn tất của một bước bất đồng bộ trong chuỗi.

Trước khi có promise, kết quả của việc thực hiện một chuỗi hàng loạt các tiến trình bất đồng bộ trong một hàm được gọi với cái tên nhẹ nhàng trìu mến là “thảm họa” kim tự tháp callback hay ***CallbackHell***.



Thay vì như vậy, bằng một cách “văn minh” hơn, chúng ta có thể gắn các callback vào các promise trả về, tạo thành một chuỗi các promise.

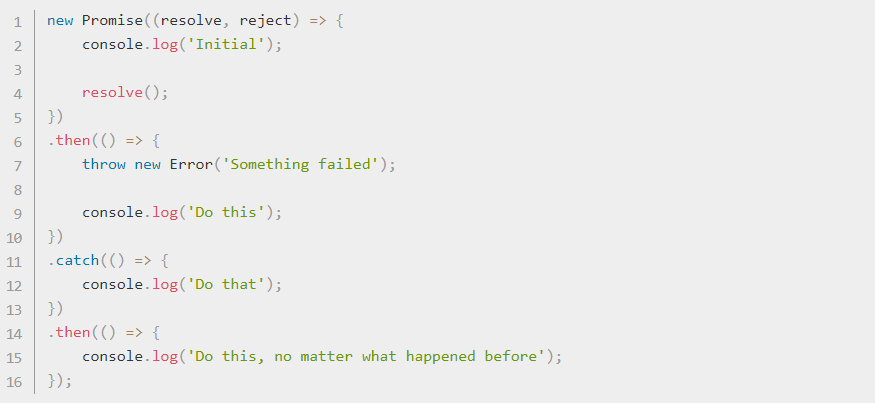
**Với đoạn code trên, thật ra *catch(failureCallBack)* là cách viết gọn của *then(null, failureCallback).* Chắc hẳn chúng ta vẫn còn nhớ cách viết code lambda trong C# - một cách code tắt -,hoàn toàn được hỗ trợ trong javascript nhé:

***Lưu ý***: Hãy luôn trả về kết quả hoặc callback sẽ không thể nào nhận được kết quả từ promise trước đó.

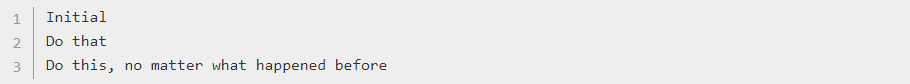
* **GỌI NỐI SAU HÀM *CATCH***

Điều này thật ra cho phép chúng ta phát triển thêm các đoạn code mặc dù đã có lỗi xảy ra ở một đoạn code catch phía trên. Chúng ta chỉ đơn thuần là tiếp tục gọi thêm từ khóa *then()* sau hàm bắt lỗi *catch()*.

Ví dụ:



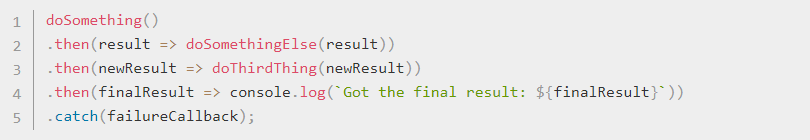
Kết quả của đoạn code trên:



**Ghi chú:** Thật ra dòng kết quả “Do this” đã không in ra bởi vì Error “Something failed” đã xảy ra trước và gây lỗi trong promise.

* **XỬ LÝ LỖI TẬP TRUNG**

Có lẽ chúng ta đã đi qua phần ***Chaining (gọi nối)*** ở phía trên, đây là phần lỗi phải nói alf nó ngốn rất nhiều thời gian của nhóm, có lẽ vì nó rối như mạng nhện ☺. Trong ví dụ của chaining, có thể thấy đến tận ba lần hàm callback lồng nhau (chính xác mà nói thì là 3 hàm failurecallback) đã được sử dụng. Thay vì vậy, đặc tính xử lý lỗi tập trung này cho phép chúng ta chỉ cần khai báo một lần vào cuối cái chuỗi promise:



Về căn bản, một chuỗi promise sẽ tự động dừng lại nếu có lỗi xảy ra và truy xuống những hàng phía dưới để tỉm và gọi hàm xử lý lỗi catch. Có vẻ nó khá giống với ***try catch*** của code đồng bộ.

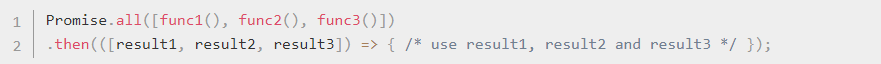
Promise có thể giúp giải quyết một hạn chế căn bản của kim tự tháp callback (thường gọi với cái tên trìu mến là ***CallbackHell***) đó là cho phép bắt được mọi loại lỗi, từ throw Error cho đến lỗi syntax. Khá quan trọng và cần thiết cho việc phối hợp xử lý các hàm đồng bộ.

* **BỐ CỤC ĐOẠN CODE**

Theo như tên Promise, trong tiếng việt có nghĩa là lời hứa, mà lời hứa thì có cả khả năng sẽ làm hay thất hứa ☺. Vậy trong code ta có Promise.resolve() tức là thành công hoặc ngược lại ta có Promise.reject() tức thất bại, không thể làm tròn nhiệm vụ, đây là hai hàm gọi tắt để tự động tạo các promise giải quyết hay từ chối (nếu có lỗi) tương ứng. Tiện dụng mọi lúc ☺.

Ngoài ra Promise.all() và Promise.race() là hai công cụ đắc lực để chạy các tác vụ bất đồng bộ song song.

Chúng ta có thể bắt đầu các tác vụ một cách song song và chờ chúng kết thúc như thế này:



## Async/Await

Là một tính năng của javascript giúp chúng ta làm việc được với các hàm bất đồng bộ theo cách dễ quản lý hơn và dễ hiểu code (khi xem lại). Nó được xây dựng dựa trên Promise. Trong đó:

Async – Khai báo một hàm bất đồng bộ

***Async function Foo(){***

***//doing some stuff here***

***...***

***}***

* Với Async, nó sẽ tự động biến bổi một hàm javascript thông thường thành một hàm Promise.
* Khi gọi tới hàm async nó sẽ xử lý mọi thứ và được trả về kết quả trong hàm của nó.
* Async cho phép sử dụng await.’

Await – tạm dừng việc thực hiện các hàm async

Ví dụ lấy lại đoạn code trên:

***Async function Foo(){***

***//doing some stuff here***

***let content = await CallSomeFunction();***

***.... // chờ đợi cho đến khi câu lệnh trên kết thúc***

***}***

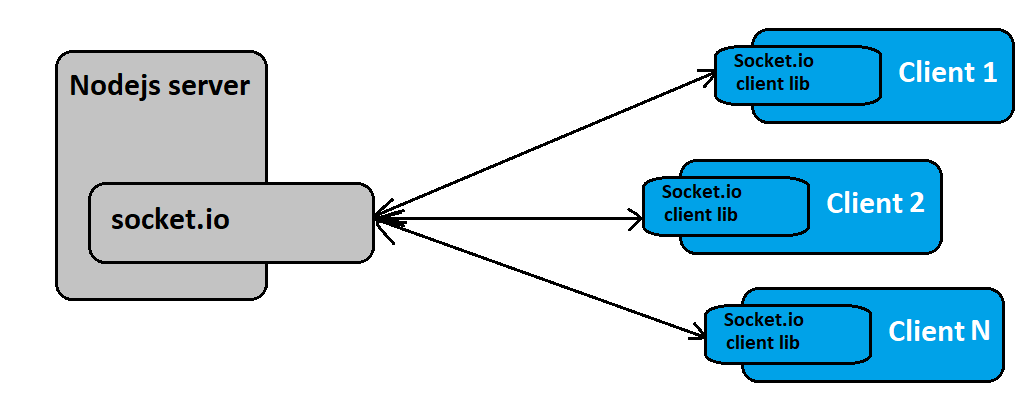
* Khi được đặt trước một Promise, nó sẽ đợi cho đến khi Promise kết thúc và trả về kết quả.
* Await chỉ làm việc với Promise, nó không hoạt động với callbacks.
* Và nó chỉ được sử dụng trong các hàm đã được khai báo async.

## Socket.io

Khi xây dựng và phát triển đến tính năng bình luận, em đã nghĩ nó nhất thiết phải có hỗ trợ việc xử lý thời gian thực. Khi một người comment thì người khác phải thấy ngay. Và rõ là socket.io xử lý vấn đề này khá tốt.

Socket.io được xây dựng nhằm mục đích tạo ra ứng dụng nodejs thời gian thực. Nó cung cấp cho lập trình viên các đặc trưng như event, room và có thể tự động phục hồi lại kết nối.

Khi include vào trong project, socket.io sẽ cung cấp cho chúng ta hai đối tượng, một dùng ở phía server và một được dùng phía client.

Khi người dùng kết nối tới socket.io server, phía client sẽ gửi cho server một “handshake HTTP request”. Server sẽ đem đi phân tích mấy cái request đó với những thông tin cần thiết trong suốt quá trình kết nối. Phía client sẽ tìm cấu hình của middleware mà đã được đăng ký với server và thực thi chúng trước khi đưa ra sự kiện kết nối. Khi kết nối thành công thì sự kiện lắng nghe kết nối được thực hiện, tạo ra một phiên bản mới của socket mà có thể được xem là định danh của client mà mỗi một client kết nối tới sẽ có một định danh.

Vể cơ bản, để sử dụng socket.io, ta chỉ cẩn cài đặt thông qua npm:

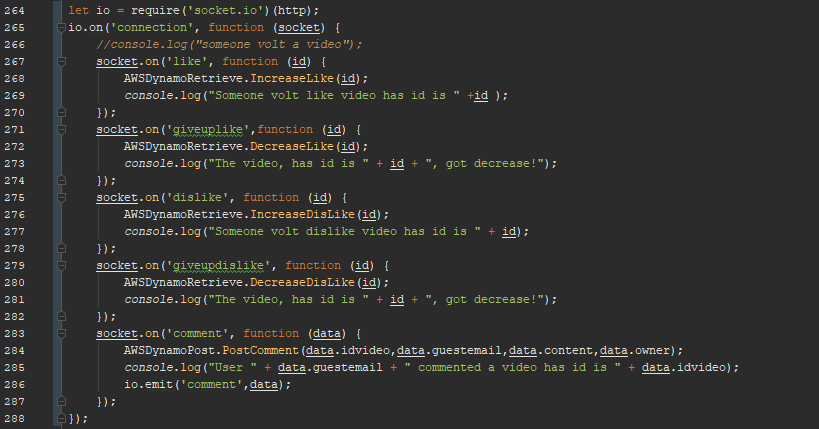


Tiếp theo là cài đặt phía server:

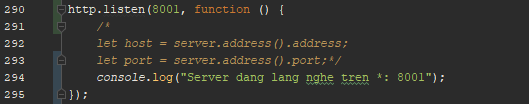
Ta sẽ gọi thư viện vào:



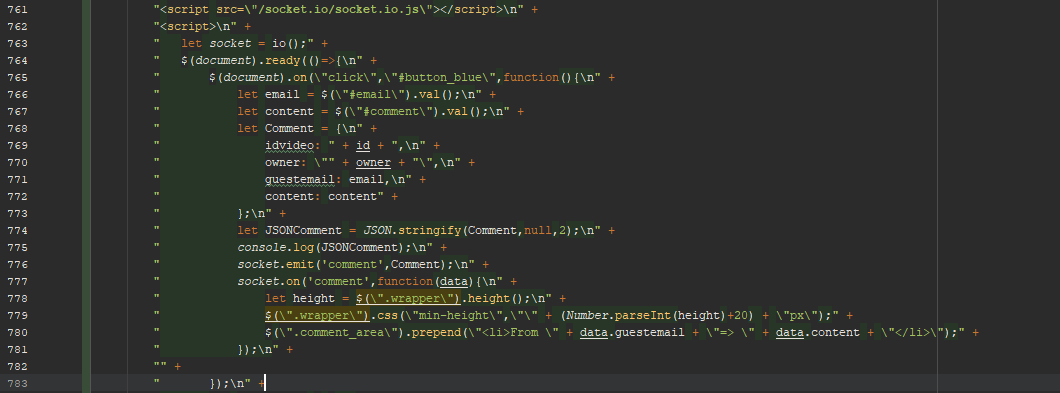
Và viết code xử lý request tương ứng từ phía client, để ý chỗ này, ta sẽ quay

lại:

Và tiếp đến là hàm khởi tạo server:



Sau đây là code phía client:

Ở dòng đầu ta khai báo để lấy thư viện socket.io về:

<script src=\"/socket.io/socket.io.js\"></script>

Sau đó, ta bắt đầu gọi hàm ***io()***

"<script>\n" +  
" let *socket* = io();" +  
" $(*document*).ready(()=>{\n" +  
" $(*document*).on(\"click\",\"#button\_blue\",function(){\n" +  
" let email = $(\"#email\").val();\n" +  
" let content = $(\"#comment\").val();\n" +  
" let Comment = {\n" +  
" idvideo: " + id + ",\n" +  
" owner: \"" + owner + "\",\n" +  
" guestemail: email,\n" +  
" content: content" +  
" };\n" +  
" let JSONComment = *JSON*.stringify(Comment,null,2);\n" +  
" *console*.log(JSONComment);\n" +  
" *socket*.emit('comment',Comment);\n" +  
" *socket*.on('comment',function(data){\n" +  
" let height = $(\".wrapper\").height();\n" +  
" $(\".wrapper\").css(\"min-height\",\"\" + (*Number*.parseInt(height)+20) + \"px\");" +  
" $(\".comment\_area\").prepend(\"<li>From \" + data.guestemail + \"=> \" + data.content + \"</li>\");" +  
" });\n" +  
"" +  
" });\n"

Khi nhấn nút comment, hàm sẽ bắt đầu lấy các dữ liệu từ form và cho vào đối tượng Comment đã được tạo sẵn rồi sẽ gửi đối tượng đó thông qua ***socket.emit.***

***.Socket.emit(Tên kênh,dữ liệu) ??***

***Socket.emit()*** là một hàm dùng để gửi dữ liệu đi

Ở đây có hai tham số:

* Tham số đầu tiên là kênh gửi, dạng chuỗi nhé. Chúng ta sẽ định nghĩa tên kênh, lưu ý rằng tên kênh phải giống nhau giữa phía client và server cho mỗi một hành động.
* Tham số thứ hai là dữ liệu ta sẽ gửi về server

Như đã nói, chúng ta sẽ quay lại phần phía server:

let io = require('socket.io')(http);  
io.on('connection', function (socket) {   
 *//console.log("someone volt a video");* socket.on('like', function (id) {  
 AWSDynamoRetrieve.IncreaseLike(id);  
 *console*.log("Someone volt like video has id is " +id );  
 });  
 socket.on('giveuplike',function (id) {  
 AWSDynamoRetrieve.DecreaseLike(id);  
 *console*.log("The video, has id is " + id + ", got decrease!");  
 });  
 socket.on('dislike', function (id) {  
 AWSDynamoRetrieve.IncreaseDisLike(id);  
 *console*.log("Someone volt dislike video has id is " + id);  
 });  
 socket.on('giveupdislike', function (id) {  
 AWSDynamoRetrieve.DecreaseDisLike(id);  
 *console*.log("The video, has id is " + id + ", got decrease!");  
 });  
 socket.on('comment', function (data) {  
 AWSDynamoPost.PostComment(data.idvideo,data.guestemail,data.content,data.owner);  
 *console*.log("User " + data.guestemail + " commented a video has id is " + data.idvideo);  
 io.emit('comment',data);  
 });  
});

Chúng ta để ý thấy có hàm ***socket.on()*** , đây là hàm sẽ nhận dữ liệu từ ***socket.emit()*** , nó sẽ lấy dữ liệu từ kênh tương ứng, trùng kênh với nhau.

Ở phía client, ta có:

*socket*.emit('comment',Comment);

Tức là gửi dữ liệu Comment qua kênh comment

Và rồi ở phía server, ta có:

socket.on('comment', function (data) {  
 AWSDynamoPost.PostComment(data.idvideo,data.guestemail,data.content,data.owner);  
 *console*.log("User " + data.guestemail + " commented a video has id is " + data.idvideo);  
 io.emit('comment',data);  
});

Tức là phía server sẽ nhận được dữ liệu Comment từ phía client và truyền vào hàm callback để xử lý việc đăng comment lên amazon web service.

## Amazon Simple Storage Service (Amazon S3)

Từ việc phát triển web server lưu trữ video dẫn tới nhu cầu trong việc lưu trữ một lượng lớn dữ liệu media, và hiển nhiên nó cần được trích xuất từ bất cứ đâu, bất cứ thiết bị nào ngay cả thiết bị dành trong việc phát triển Internet of things đồng thời phải được lưu trữ một cách đơn giản và bảo mật. Ứng viên tốt nhất trong trường hợp này có lẽ là Amazon Simple Storage Service.

Web server lưu trữ video của nhóm rõ là cần những cái ích sau đây từ việc dùng amazon s3:

1. Sao lưu và phục hồi
   * + Amazon S3 cung cấp điểm đến cực kỳ ổn định, quy mô linh hoạt và bảo mật để nhóm có thể sao lưu và lưu trữ dữ liệu. Nhóm hoàn toàn có thể sử dụng khả năng tạo phiên bản của s3 để đem lại lá chắn bảo vệ hoàn hảo cho dữ liệu. Với tận bốn lớp lưu trữ khác nhau để giúp nhóm tối ưu hóa chi phí và hiệu năng nhất có thể mà vẫn có thể đáp ứng được thời điểm và thời gian khôi phục.
2. Lưu trữ dữ liệu
   * + Cung cấp rất nhiều lớp lưu trữ dữ liệu nhằm đáp ứng nhu cầu lưu trữ một cách “chuẩn” nhất có thể của người dùng. Thậm chí có thể đáp ứng nhu cầu lưu trữ của các ngành pháp quy hay lưu trữ một cách chủ động cho các tổ chức cần truy cập nhanh và không thường xuyên vào dữ liệu được lưu trữ.
3. Phục hồi được sau thảm họa
   * + Có các cơ sở hạ tầng trên toàn cầu, tất cả đều có độ bền và bảo mật cao cảu amazon s3 đem đến giải pháp phục hồi sau thảm họa mạnh mẽ, được thiết kế đem đến khả năng bảo vệ dữ liệu vượt trội.
     + Cross-Region Replication (CRR) tự động sao chép mọi đối tượng s3 đến bộ chứa đích nằm trên khu vực AWS khác.

Để mà có thể ứng dụng tính lưu trữ của s3 vào project, về mặt lý thuyết có thể sử dụng đoạn mã sau:

Function upload (req, res) {

var file = req.files.file;

var s3bucket = new AWS.S3({params: {Bucket: 'mybucketname'}});

s3bucket.createBucket(function() {

var params = {

Key: file.name,

Body: file

};

s3bucket.upload(params, function(err, data) {

console.log("PRINT FILE:", file);

if (err) {

console.log('ERROR MSG: ', err);

} else {

console.log('Successfully uploaded data');

}

});

});

};

Thật chất để mà lưu được file media, nhóm đã kết hợp với cả gói urlencodeParser – đã được nói ở mục [trên](#_Gói_Body-parser_của) và cả gói multer – được phát triển riêng trên cộng đồng github, xem lại ở [đây](#_Multer).

Đoạn mã thật sự trong việc ứng dụng việc lưu trữ của project được thể hiện dưới đây và nó khá là dài dòng:

let multer = require('multer');  
let bodyParser = require('body-parser');

let urlencodedParser = bodyParser.urlencoded({ extended : true });

var storage = multer.diskStorage({  
 destination: function (req,file,cb) {  
 cb(null,'./api/data/')  
 },  
 filename: function (req,file,cb) {  
 cb(null,*Date*.now() + file.originalname);  
 }  
});  
  
const upload = multer({  
 storage: storage  
});

// ý tưởng chính ở đây là dùng multer để báo cho ide biết là ta sẽ tạo một

//folder trong project dùng cho việc lưu trữ tạm file media

//sau khi đăng lên s3 thành công thì xóa nó đi để giải phóng dung lượng

app.post("/Dangbai", urlencodedParser, upload.any(), function (req,res,next) {  
 fs.readdir("./api/data/", (err,files) =>{  
 const videotargetPath = path.join(\_\_dirname, "./api/data/" + files[0]);  
 const imagetargetPath = path.join(\_\_dirname, "./api/data/" + files[1]);  
 let video\_file\_location = ""; *// tạo biến để lưu địa chỉ của file được lưu trong s3* let image\_file\_location = "";  
 *console*.log(videotargetPath);  
 *console*.log(imagetargetPath);  
 AWS.config.update({  
 region:'us-east-1',

endpoint:'http://s3.amazonaws.com',  
 "accessKeyId": config.accesskeyid, "secretAccessKey": config.secretkey  
 });  
*//create s3 service object* s3 = new AWS.S3({ apiVersion : '2006-03-01' });  
*//call s3 to retrieve upload file to specified bucket* let uploadParams1 = {Bucket: "handmadevideos001", Key: '', Body: '',ACL:'public-read-write'};  
 let uploadParams2 = {Bucket: "handmadevideos001", Key: '', Body: '',ACL:'public-read-write'};  
 let videofileStream = fs.createReadStream(videotargetPath);  
 let imagefileStream = fs.createReadStream(imagetargetPath);  
  
 uploadParams1.Body = videofileStream;*//nội dung file* uploadParams1.Key = "videos/" + path.basename(videotargetPath);*//đường dẫn folder lưu file trong s3* uploadParams2.Body = imagefileStream;  
 uploadParams2.Key = "images/" + path.basename(imagetargetPath);  
*//call s3 to retrieve upload file to specified bucket* s3.upload( uploadParams1, function (err, data) {  
 if (err)  
 *console*.log("Error", err);  
 else{  
 video\_file\_location = data.Location;  
 *console*.log("Upload success", video\_file\_location);  
 *//unlink để xóa đi file vừa add vào hệ thống sau khi up lên cloud* fs.unlink(videotargetPath, function (err) {  
 if (err)  
 return *console*.log(err);  
 else  
 return *console*.log("Remove successfully");  
 });  
 AWS.config.update({  
 region:'us-east-1',  
 endpoint:'http://dynamodb.us-east-1.amazonaws.com',  
 "accessKeyId": config.accesskeyid, "secretAccessKey": config.secretkey  
 });  
 let docClient = new AWS.DynamoDB.DocumentClient();  
 let table = "handmadevideo01";  
 let email = req.body.email;  
 let urlVideo = video\_file\_location;  
 let image = image\_file\_location;  
 let like\_count = 0;  
 let dislike\_count = 0;  
 let title = req.body.video\_title;  
 let summary = req.body.video\_description;  
 let tags = req.body.video\_tags;  
 let params0 = {  
 TableName: table,  
 Select: 'COUNT'  
 };  
 docClient.scan(params0, (err,data) => {  
 if (err)  
 *console*.log("Please view some json errors:" + *JSON*.stringify(err,null,2));  
 else {  
 let params = {  
 TableName : table,  
 Item:{  
 "id" : *Number*.parseInt(data.Count + 1),  
 "email" : email,  
 "urlVideo" : urlVideo,

"image" : image,  
 "like\_count": like\_count,  
 "dislike\_count": dislike\_count,  
 "title" : title,  
 "summary" : summary,  
 "tags" : tags  
 }  
  
 };  
 docClient.put(params, function (err, data) {  
 if (err)  
 *console*.log("Unable to add a new item.Please review some json errors : ",*JSON*.stringify(err,null,2));  
 else  
 *console*.log("Added item:",*JSON*.stringify(data,null,2));  
 });  
 res.redirect("/editorrender?email=" + email);  
 }  
 });  
 }  
 });  
  
 s3.upload( uploadParams2, function (err,data) {  
 if (err)  
 *console*.log("Error", err);  
 else{  
 image\_file\_location = data.Location;  
 *console*.log("Upload success", image\_file\_location);  
 fs.unlink(imagetargetPath, function (err) {  
 if (err)  
 return *console*.log(err);  
 else  
 return *console*.log("Remove successfully");  
 });  
 }  
 });  
 });  
  
});

## Amazon DynamoDB

# : PHÂN TÍCH

Từ ý tưởng tạo một web server chủ yếu cho người dung chia sẻ video hướng dẫn, một web server đơn giản mang tính chất giải trí cao. Từ các tính năng như:

-Với tư cách người dùng thông thường

* Đăng tải video
* Like video
* Bình luận video
* Theo dõi theo chủ đề
* Đăng ký

-Với tư cách người đăng bài

* Xem lại tất cả các bài đã đăng xem các bình luận
* Xem các bình luận mà người ta bình luận trên bài của mình
* Gỡ bỏ các bài đăng
* Đăng bài
* Đăng nhập

Có thể thấy việc chúng ta lưu trữ các dữ liệu media số lượng lớn là tất yếu. Về cơ bản sẽ sử dụng Dynamodb của Amazon web service để giải quyết vấn đề này. Có thể nhận ra được đối tượng video có thể được thiết kế như sau đây là phù hợp và tiện lợi để giải quyết vấn đề trên:

|  |  |
| --- | --- |
| Tên bảng: handmadevideo01 | |
| Tên thuộc tính | Kiểu dữ liệu và ý nghĩa |
| id | Number, mã của mỗi video được đăng tải. |
| dislike\_count | Number, số lượng lượt không thích của mỗi video. |
| email | String, địa chỉ email của người đăng bài. |
| HandleName | String, tên tài khoản của người đăng. |
| image | String, địa chỉ hình ảnh được lưu trong s3. |
| like\_count | Number, số lượng lượt thích của mỗi video. |
| summary | String, tóm tắt nội dung video. |
| tags | String, tên loại của video |
| title | String, tên tiêu đề của video |
| urlVideo | String, địa chỉ đường dẫn của video bên s3. |

Và cũng thuận tiện trong việc lấy bình luận cho mỗi video tương ứng, bảng Bình luận sẽ được thiết kế như sau đây:

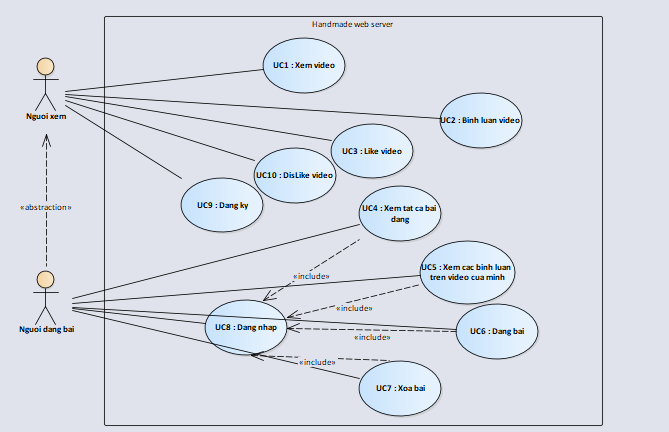
|  |  |
| --- | --- |
| Tên bảng: Comments | |
| Tên thuộc tính | Kiểu dữ liệu và ý nghĩa |
| IDcontent | Number, mã của mỗi bình luận. |
| guestemail | String, địa chỉ email của người bình luận. |
| content | String, nội dung bình luận. |
| idvideo | Number, mã của video được bình luận tương ứng. |
| writer | String, địa chỉ email của người đăng video đó. |

Về việc đăng ký hay đăng nhập, thông tin cũng sẽ được lưu tại Dynamodb, bảng tài khoản sẽ được thiết kế như sau:

|  |  |
| --- | --- |
| Tên bảng: Account | |
| Tên thuộc tính | Kiểu dữ liệu và ý nghĩa |
| EMAIL\_ADDRESS | String, địa chỉ email của người dùng đăng ký. |
| PASS | String, mật khẩu tương ứng của người dùng đăng ký. |

Từ những nhận định và những yêu cầu hệ thống trên, có thể khái quát hóa hệ thống thành sơ đồ Use case sau đây:

**SƠ ĐỒ USE CASE TỔNG QUÁT**



## Đặc tả use case cho từng chức năng

### UCID: UC1

|  |  |  |
| --- | --- | --- |
| Tên use case | Xem video | |
| Tác nhân | Người xem | |
| Mục đích | Cho người dùng xem video | |
| Mô tả | Sau khi click vào video, hệ thống sẽ lấy video đó trong dynamodb và trả về cho người dùng | |
| Tiền điều kiện |  | |
| Hậu điều kiện | Trả về đúng video người xem muốn | |
| Luồng sự kiện chính | Người xem, người đăng bài | Hệ thống |
|  | 1. Chọn vào video muốn xem |  |
|  |  | 1. Lấy thông tin từ phía client |
|  |  | 1. Render về đoạn HTML chứa nội dung video |
| Luồng sự kiện thay thế |  | |

### UCID: UC2

|  |  |  |
| --- | --- | --- |
| Tên use case | Bình luận video | |
| Tác nhân | Người xem | |
| Mục đích | Người xem video có thể để lại bình luận | |
| Mô tả | Chức năng này giúp cho người xem video hoàn toàn có khả năng để lại suy nghĩ của mình về video. | |
| Tiền điều kiện | [UC1](#_UCID:_UC1) | |
| Hậu điều kiện | Bình luận được hiển thị trong phần bình luận | |
| Luồng sự kiện chính | Người xem, người đăng bài | Hệ thống |
|  | 1. Nhập bình luận |  |
|  | 1. Chọn đăng bình luận |  |
|  |  | 1. Lấy dữ liệu post từ phía client |
|  |  | 1. Lưu vào dynamodb |
|  |  | 1. Hiển thị cho người dùng khác |
| Luồng sự kiện thay thế |  | |

### UCID: UC3

|  |  |  |
| --- | --- | --- |
| Tên use case | Like video | |
| Tác nhân | Người xem | |
| Mục đích | Cho người xem đánh giá video bằng lượt like. | |
| Mô tả | Người xem nhấn vào biểu tượng like thì sẽ tăng số lượng like lên. | |
| Tiền điều kiện | [UC1](#_UCID:_UC1) | |
| Hậu điều kiện | Lượt like tăng lên 1 | |
| Luồng sự kiện chính | Người xem | Hệ thống |
|  | 1. Nhấn vào biểu tượng like |  |
|  |  | 1. Tăng tạm thời lượt like lên 1 |
|  |  | 1. Cập nhật lượt like của video lên 1 đơn vị trong dynamodb |
|  |  | 1. Cập nhật lại lượt like cho các người dùng khác |
| Luồng sự kiện thay thế | Luồng sự kiện thay thế a1 bắt đầu tại bước 1 khi người xem nhấn thêm lần nữa | |
|  | Người xem | Hệ thống |
|  |  | 1. Giảm tạm thời lượt like xuống 1 |
|  |  | 1. Cập nhật lượt like video xuống 1 đơn vị trong dynamodb |
|  |  | 1. Cập nhật lại lượt like cho các người dùng khác |

### UCID: UC4

|  |  |  |
| --- | --- | --- |
| Tên use case | Xem tất cả bài đăng | |
| Tác nhân | Người đăng | |
| Mục đích | Giúp người đăng bài cụ thể được xem lại danh sách những bài mình đã đăng. | |
| Mô tả | Chức năng này sẽ lấy toàn bộ các bài đăng mà người ấy đã đăng và hiển thị lại dưới dạng danh sách. | |
| Tiền điều kiện | [UC8](#_UCID:_UC8Đăng_nhập) | |
| Hậu điều kiện | Trả về danh sách các bài đã đăng của người đăng. | |
| Luồng sự kiện chính | Người đăng | Hệ thống |
|  | 1. Chọn vào đường dẫn menu Danh sách bài đăng |  |
|  |  | 1. Trả về danh sách bài đăng của người đó |
| Luồng sự kiện thay thế | Luồng sự kiện thay thế a1 bắt đầu tại bước số 1 khi mà không lấy được danh sách bài đăng của người đó | |
|  | Người đăng | Hệ thống |
|  |  | 1. Trả về null |

### UCID: UC5

|  |  |  |
| --- | --- | --- |
| Tên use case | Xem tất cả bình luận trên video của mình | |
| Tác nhân | Người đăng | |
| Mục đích | Giúp cho người đăng xem các bình luận trên các bài viết của mình. | |
| Mô tả | Sau khi click vào, hệ thống sẽ trả về các bài đăng kèm theo các bình luận trên mỗi bài đăng đó. | |
| Tiền điều kiện | [UC8](#_UCID:_UC8Đăng_nhập) | |
| Hậu điều kiện | Trả về được danh sách các bài đăng kèm theo các bài đăng trên mỗi video ấy. | |
| Luồng sự kiện chính | Người đăng | Hệ thống |
|  | 1. Chọn vào đường dẫn xem các bình luận |  |
|  |  | 1. Trả về danh sách các bài đăng và kèm theo các bình luận trên mỗi video. |
| Luồng sự kiện thay thế | Luồng sự kiện thay thế a1 bắt đầu tại bước số 1 khi mà không có bài đăng của người đó. | |
|  | Người đăng | Hệ thống |
|  |  | 1. Trả về null. |

### UCID: UC6

|  |  |  |
| --- | --- | --- |
| Tên use case | Đăng bài | |
| Tác nhân | Người đăng | |
| Mục đích | Giúp người dùng có thể đăng video của mình | |
| Mô tả | Sau khi click vào đường dẫn Đăng bài, người dùng có thể cung cấp các thông số nội dung của video và có thể đăng bài của mình. | |
| Tiền điều kiện | [UC8](#_UCID:_UC8Đăng_nhập) | |
| Hậu điều kiện | Nội dung video được kiểm tra và lưu trữ lên đám mây sau ít phút sau đó. | |
| Luồng sự kiện chính | Người đăng | Hệ thống |
|  | 1. Chọn vào đường dẫn Đăng bài |  |
|  |  | 1. Trả về form đăng bài |
|  | 1. Cung cấp đầy đủ nội dung video |  |
|  | 1. Chọn đăng bài |  |
|  |  | 1. Kiểm tra dữ liệu |
|  |  | 1. Lưu vào cơ sở dữ liệu |
|  |  | 1. Tải lại trang |
| Luồng sự kiện thay thế | Luồng sự kiện thay thế a1 bắt đầu tại bước số 5 khi hệ thống nhận thấy một trường dữ liệu bị thiếu hoặc sai kiểu dữ liệu. | |
|  | Người đăng | Hệ thống |
|  |  | 6)Yêu cầu người đăng chỉnh sửa hay bổ sung thông tin đó. |

### UCID: UC7

|  |  |  |
| --- | --- | --- |
| Tên use case | Xóa bài | |
| Tác nhân | Người đăng bài | |
| Mục đích | Giúp người đăng bài gỡ bỏ video đăng lộn hoặc muốn gỡ video nào đó. | |
| Mô tả | Người đăng sẽ chọn vào đường dẫn Gỡ bài/Xóa bài và từ danh sách họ có thể chọn bài cần xóa và nhấn xóa. | |
| Tiền điều kiện | [UC8](#_UCID:_UC8Đăng_nhập), có bài đăng của mình. | |
| Hậu điều kiện | Bài đăng đó gỡ xuống khỏi dynamodb và simple storage service (s3) thành công. | |
| Luồng sự kiện chính | Người đăng | Hệ thống |
|  | 1. Chọn vào đường dẫn xóa bài |  |
|  |  | 1. Trả về danh sách bài đăng |
|  | 1. Chọn xóa bài muốn xóa |  |
|  |  | 1. Xóa khỏi hệ thống cơ sở dữ liệu đám mây. |
|  |  | 1. Thông báo cho người dùng khác là người đăng đã xóa bài đăng với tên video tương ứng. |
| Luồng sự kiện thay thế |  | |

### UCID: UC8

|  |  |  |
| --- | --- | --- |
| Tên use case | Đăng nhập | |
| Tác nhân | Người đăng | |
| Mục đích | Giúp cho người đăng có thể đăng nhập vào hệ thống. | |
| Mô tả | Người đăng vào trang chính của web và click vào login. Từ đó, họ cung cấp email và mật khẩu để vào trang chính của họ. | |
| Tiền điều kiện | [UC9](#_UCID:_UC9Đăng_ký) | |
| Hậu điều kiện | Đăng nhập được vào hệ thống | |
| Luồng sự kiện chính | Người đăng | Hệ thống |
|  | 1. Từ trang chính chọn login |  |
|  |  | 1. Trả về trang login |
|  | 1. Cung cấp email và mật khẩu |  |
|  |  | 1. Kiểm tra hợp lệ |
|  |  | 1. Trả về trang chính dành cho người đăng bài. |
| Luồng sự kiện thay thế | Luồng sự kiện a1 bắt đầu tại bước số 4 khi email hoặc mật khẩu nhập sai hoặc bỏ sót | |
|  | Người đăng | Hệ thống |
|  |  | 5) Trả về lại trang login |

### UCID: UC9

|  |  |  |
| --- | --- | --- |
| Tên use case | Đăng ký | |
| Tác nhân | Người xem | |
| Mục đích | Trờ thành thành viên để có thể đăng bài | |
| Mô tả | Từ trang chủ người xem có thể chọn vào phần login, tích vào phần đăng ký và cung cấp thông tin để có thể đăng bài sau này. | |
| Tiền điều kiện |  | |
| Hậu điều kiện | Tài khoản được lưu vào đám mây, người dùng hoàn toàn có thể truy cập được vào tài khoản của mình. | |
| Luồng sự kiện chính | Người xem | Hệ thống |
|  | 1. Chọn login |  |
|  |  | 1. Trả về trang login |
|  | 1. Click vào đường dẫn đăng ký |  |
|  |  | 1. Chuyển thành form đăng ký |
|  | 1. Cung cấp thông tin |  |
|  | 1. Chọn đăng ký |  |
|  |  | 1. Kiểm tra hợp lệ dữ liệu |
|  |  | 1. Lưu vào thông tin tài khoản vào đám mây |
|  |  | 1. Trả về trang login |
| Luồng sự kiện thay thế | Luồng sự kiện thay thế a1 bắt đầu tại bước số 7 khi hệ thống nhận thấy dữ liệu bị thiếu hoặc không hợp lệ | |
|  | Người xem | Hệ thống |
|  |  | 1. Yêu cầu người dùng cung cấp lại hoặc chỉnh sửa thông tin |

### UCID: UC10

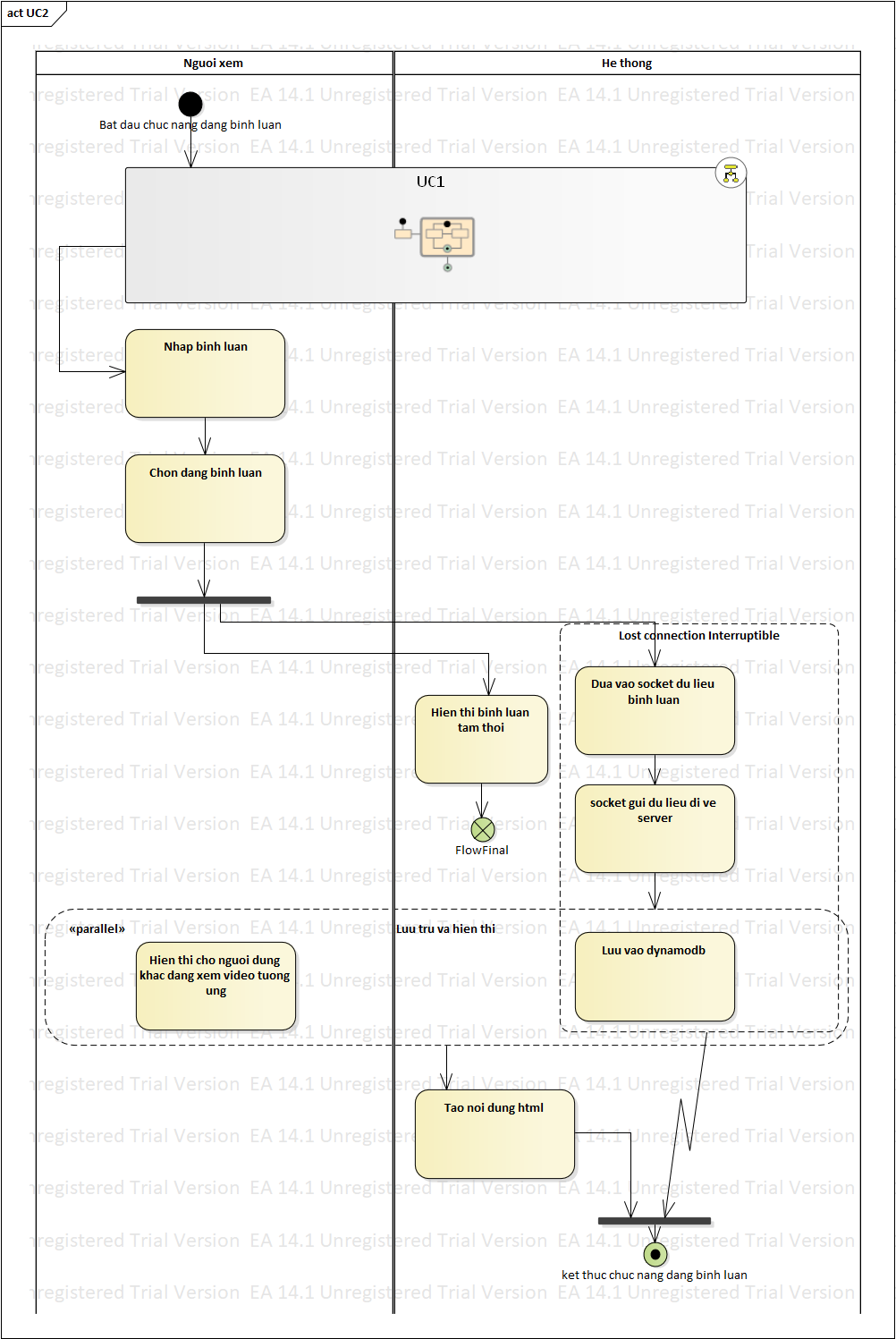
|  |  |  |
| --- | --- | --- |
| Tên use case | Dislike video | |
| Tác nhân | Người xem, người đăng bài | |
| Mục đích | Đánh giá chất lượng video không tốt | |
| Mô tả | Bằng việc chọn vào biểu tượng không thích, người xem có thể đánh giá lượt không thích của video. | |
| Tiền điều kiện | [UC1](#_UCID:_UC1) | |
| Hậu điều kiện | Lượt dislike tăng lên 1 đơn vị | |
| Luồng sự kiện chính | Người xem | Hệ thống |
|  | 1. Nhấn vào biểu tượng dislike |  |
|  |  | 1. Tăng tạm thời lượt dislike lên 1 |
|  |  | 1. Cập nhật lượt dislike của video lên 1 đơn vị trong dynamodb |
|  |  | 1. Cập nhật lại lượt dislike cho các người dùng khác |
| Luồng sự kiện thay thế | Luồng sự kiện thay thế a1 bắt đầu tại bước 1 khi người xem nhấn thêm lần nữa | |
|  | Người xem | Hệ thống |
|  |  | 1. Giảm tạm thời lượt dislike xuống 1 |
|  |  | 1. Cập nhật lượt dislike video xuống 1 đơn vị trong dynamodb |
|  |  | 1. Cập nhật lại lượt dislike cho các người dùng khác |

# : THIẾT KẾ VÀ HIỆN THỰC

## Sơ đồ activity

### Sơ đồ activity cho [UC1](#_UCID:_UC1)

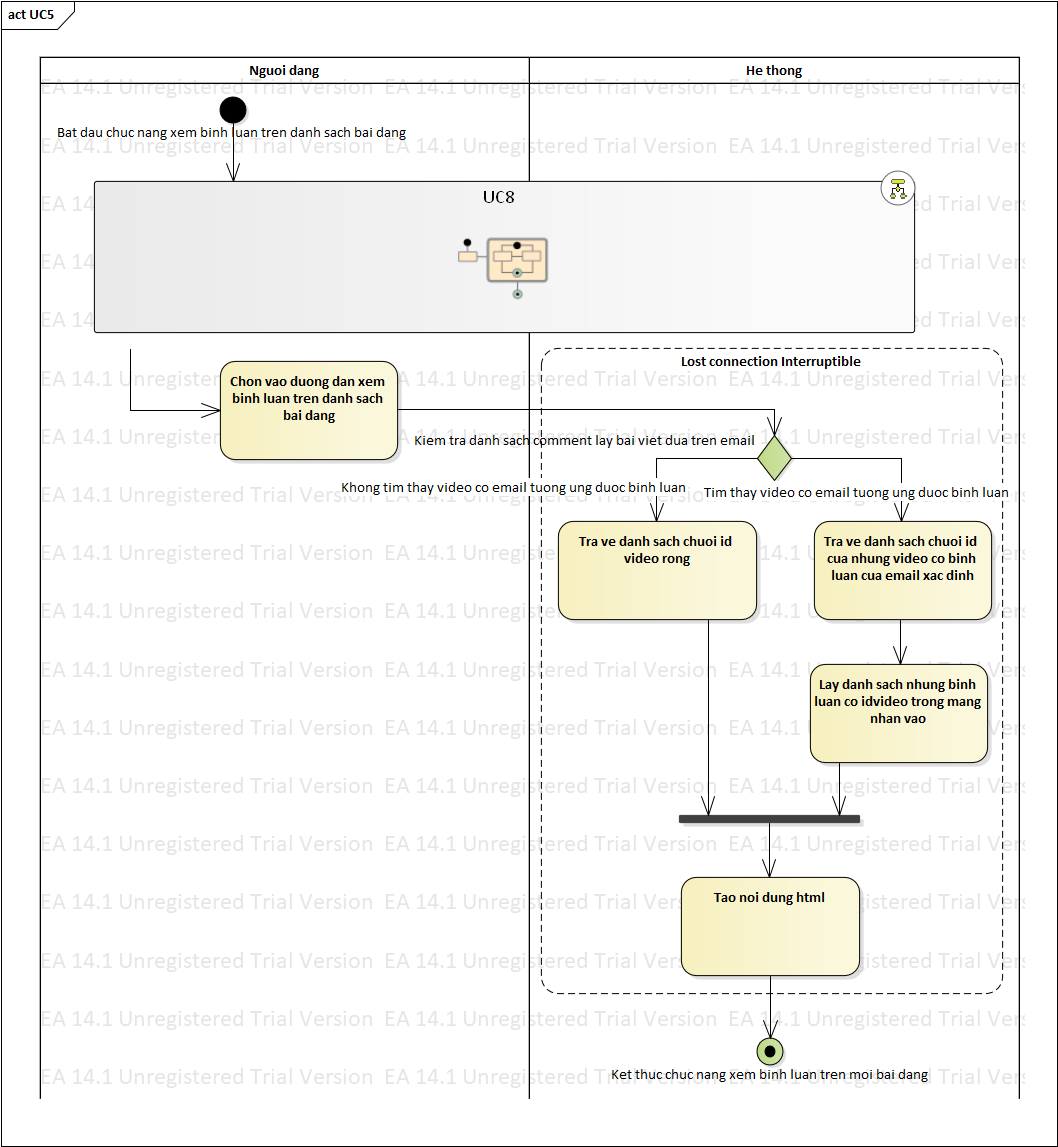
### Sơ đồ activity cho [UC2](#_UCID:_UC2)



### Sơ đồ activity cho [UC3](#_UCID:_UC3)

### Sơ đồ activity cho [UC4](#_UCID:_UC4)

### Sơ đồ activity cho [UC5](#_UCID:_UC5)



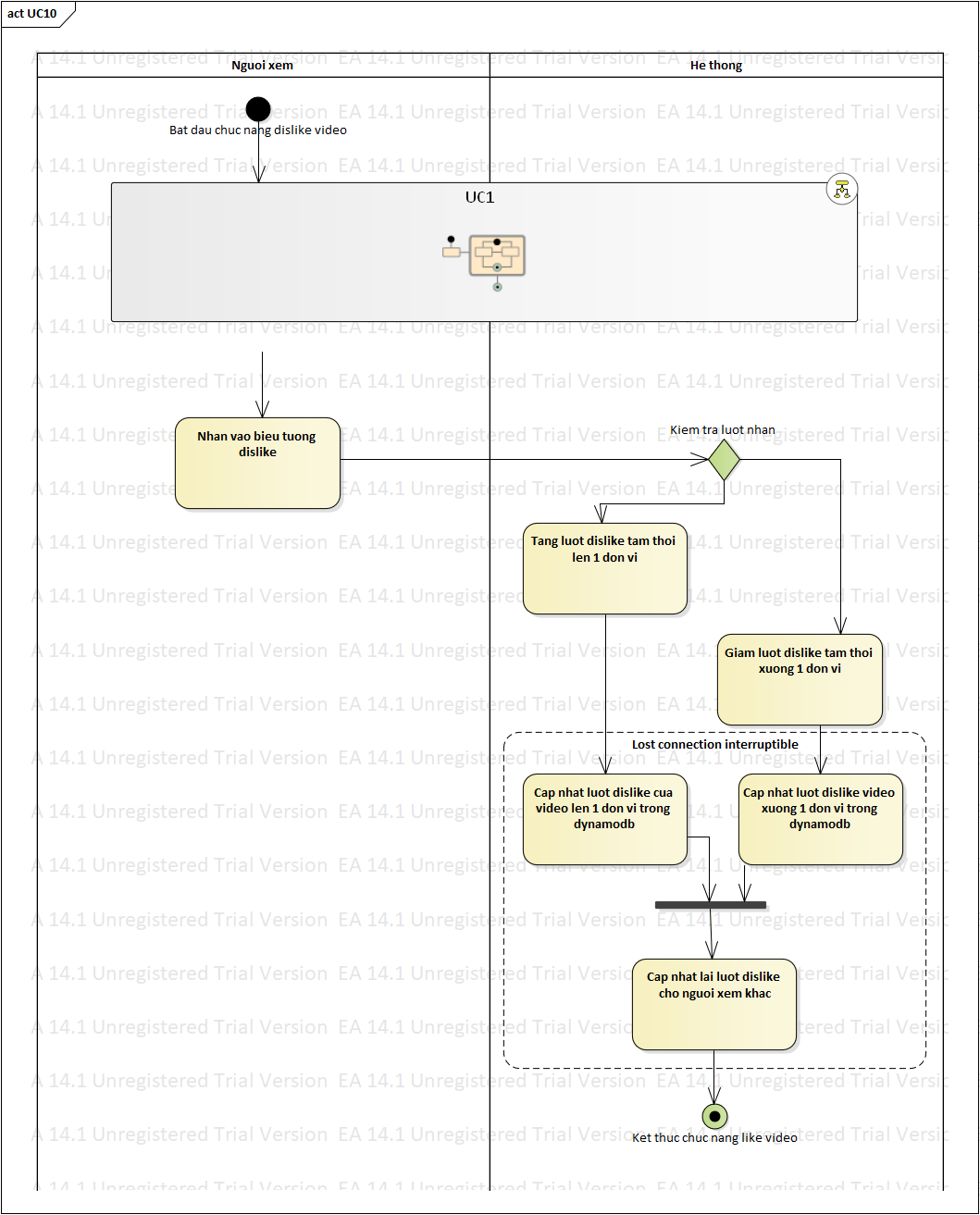
### Sơ đồ activity cho [UC6](#_UCID:_UC6)

### Sơ đồ activity cho [UC7](#_UCID:_UC7)

### Sơ đồ activity cho [UC8](#_UCID:_UC8)

### Sơ đồ activity cho [UC9](#_UCID:_UC9)

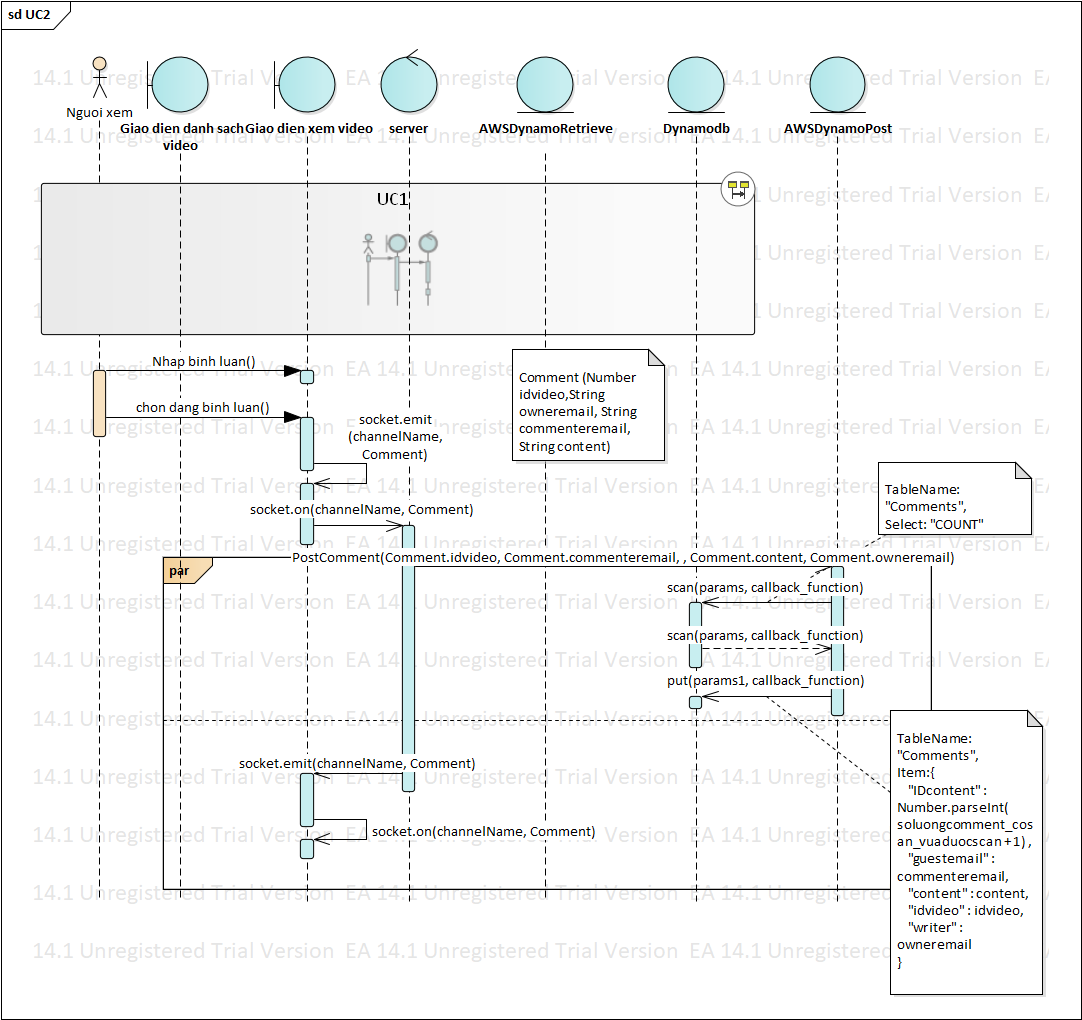
### Sơ đồ activity cho [UC10](#_UCID:_UC10)



## Sơ đồ sequence

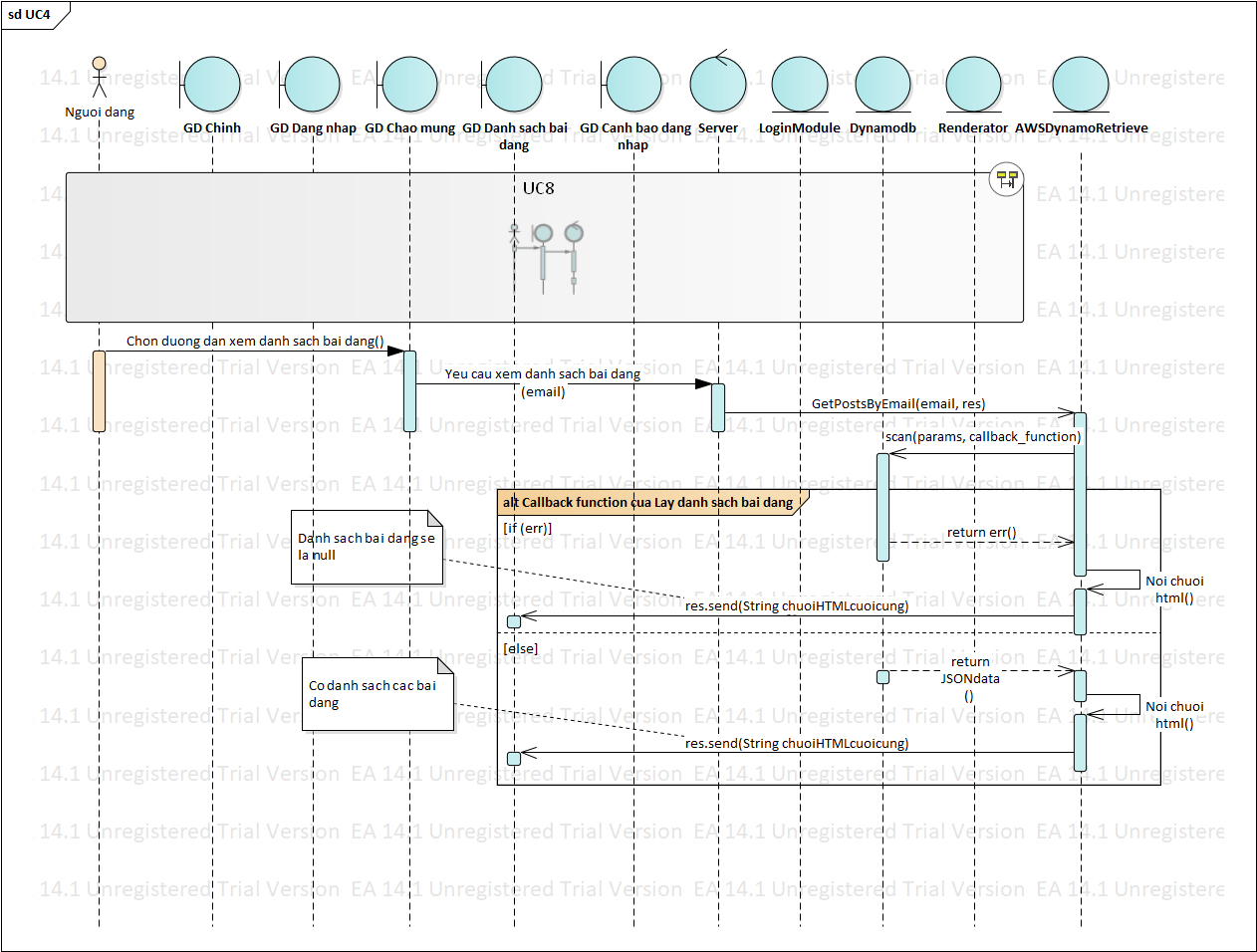
### Sơ đồ sequence cho [UC1](#_UCID:_UC1)

### Sơ đồ sequence cho [UC2](#_UCID:_UC2)

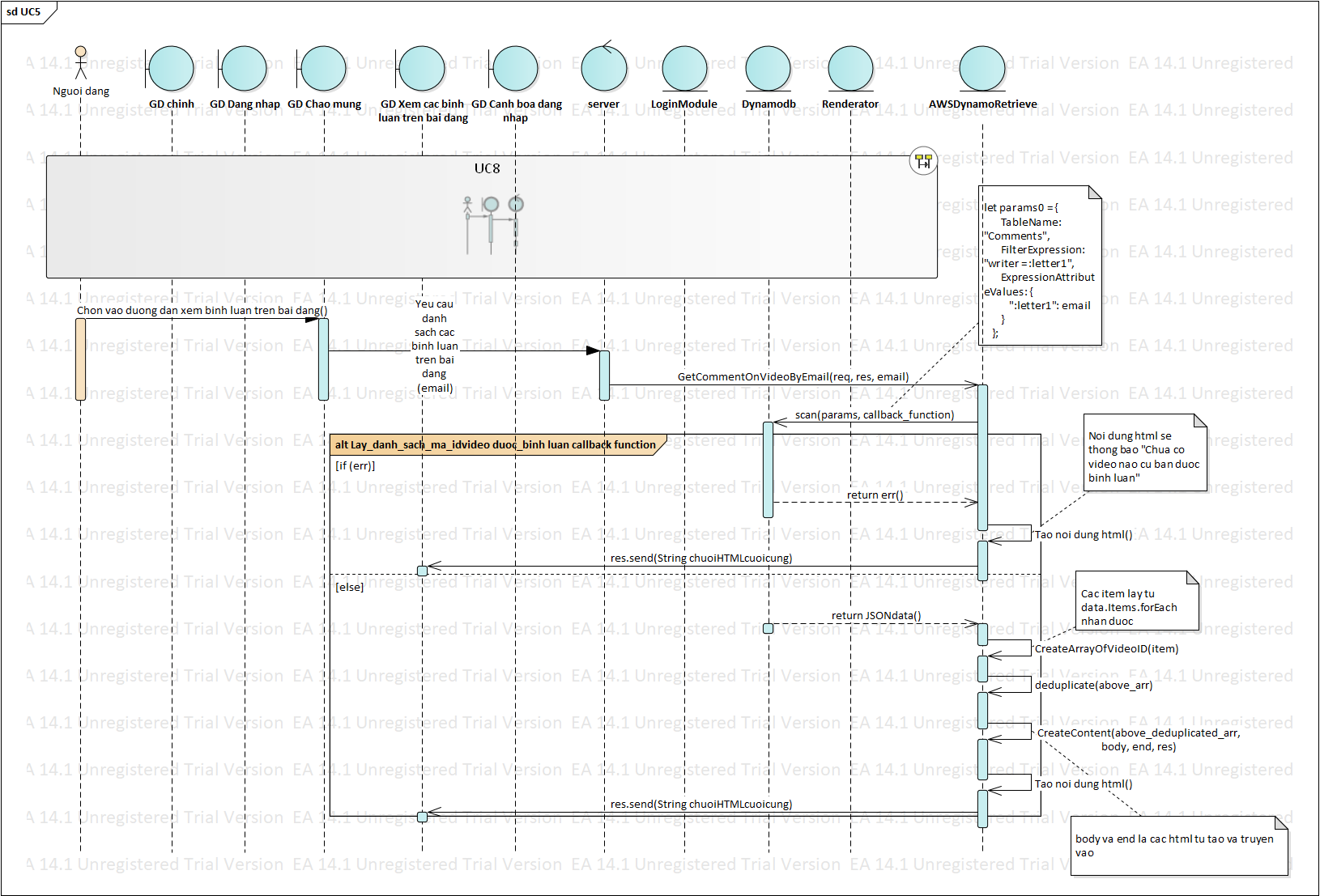


### Sơ đồ sequence cho [UC3](#_UCID:_UC3)

### Sơ đồ sequence cho [UC4](#_UCID:_UC4)

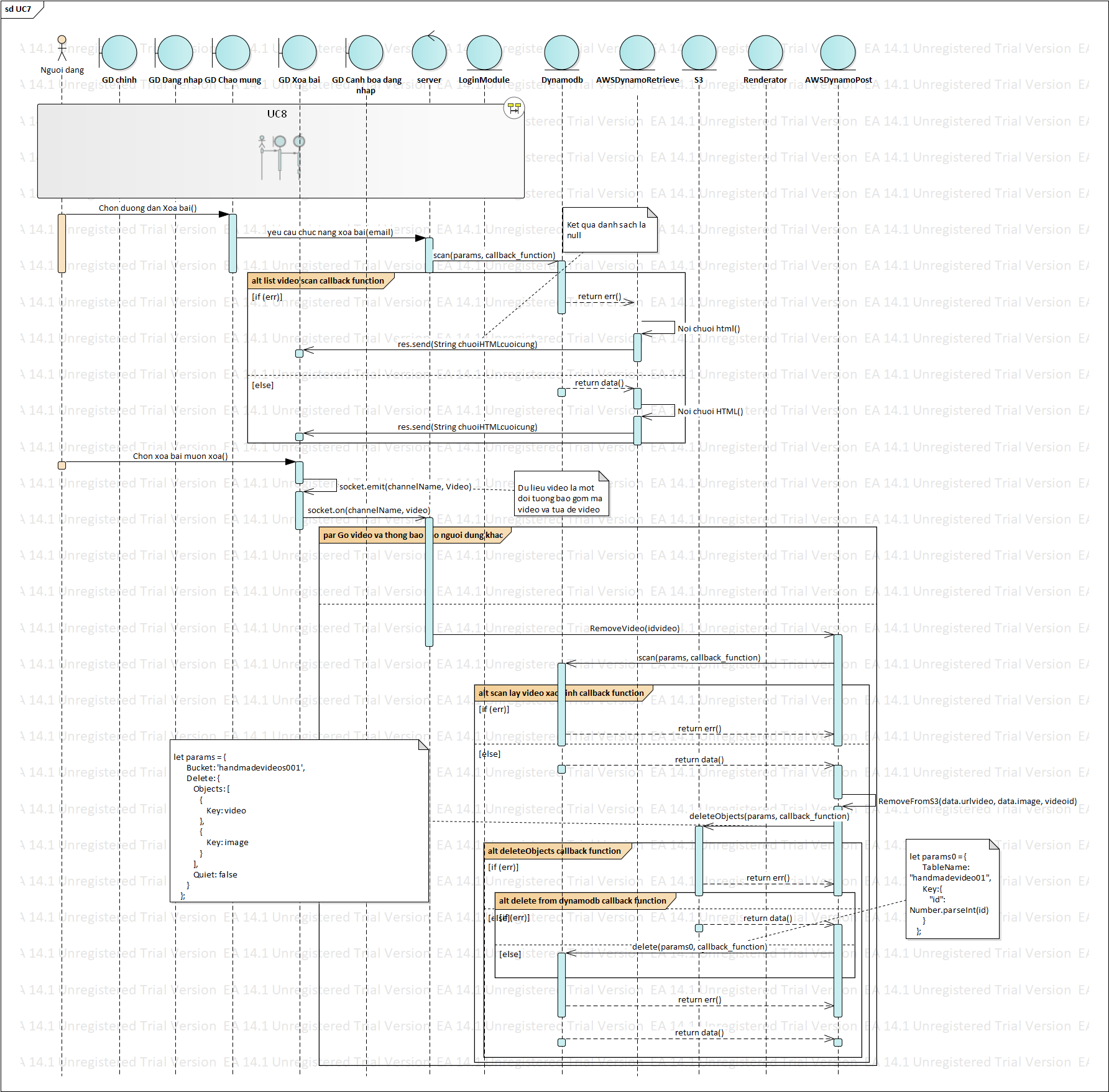


### Sơ đồ sequence cho [UC5](#_UCID:_UC5)

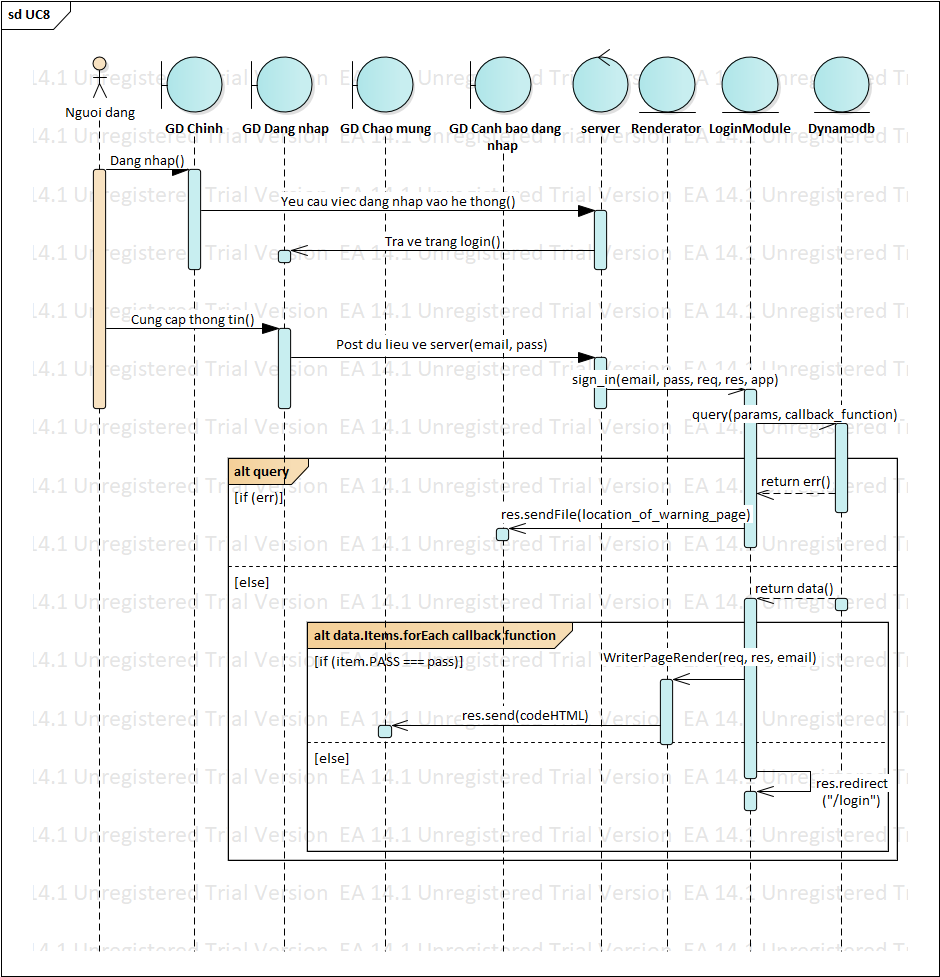


### Sơ đồ sequence cho [UC6](#_UCID:_UC6)

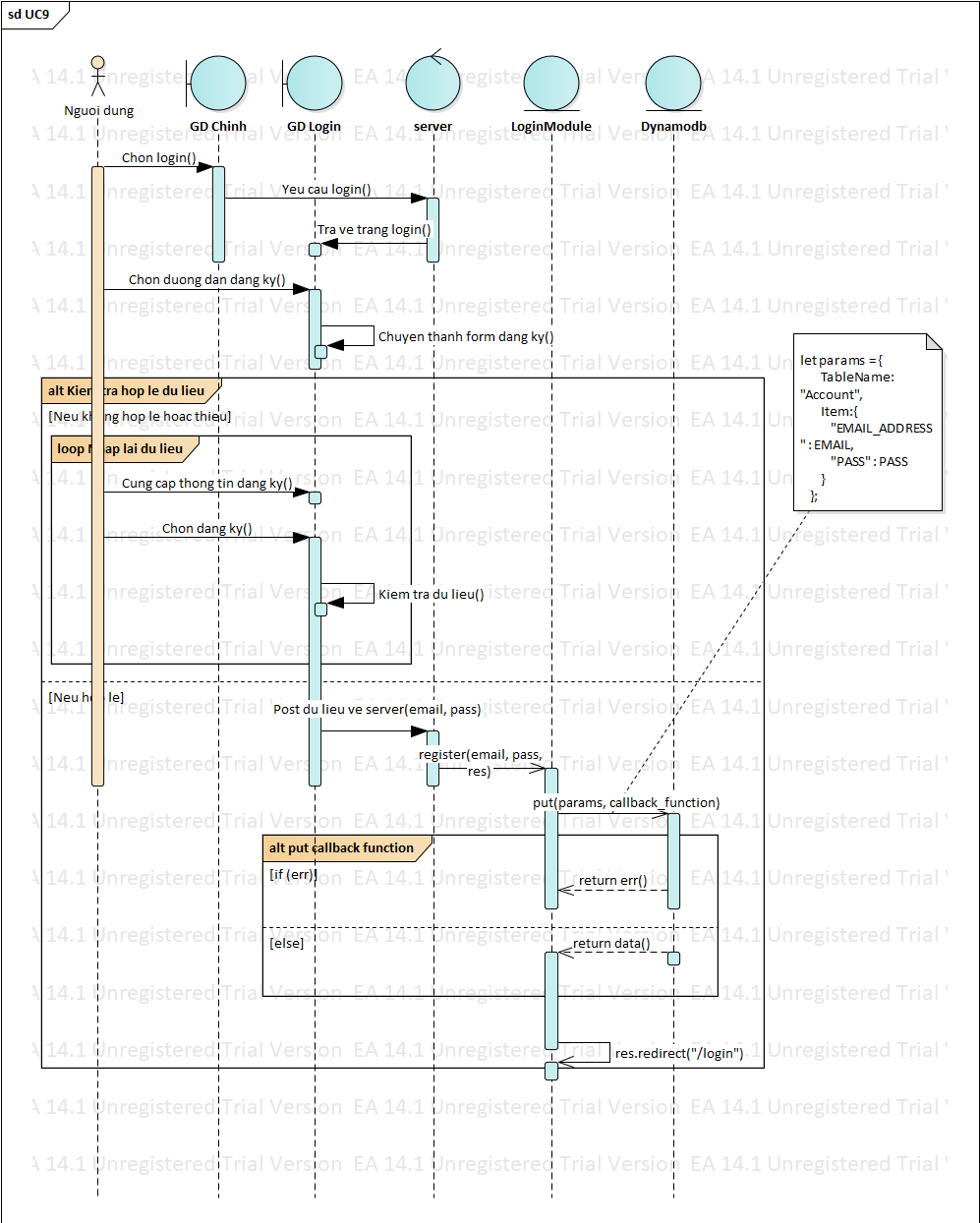
### Sơ đồ sequence cho [UC7](#_UCID:_UC7)



### Sơ đồ sequence cho [UC8](#_UCID:_UC8)



### Sơ đồ sequence cho [UC9](#_UCID:_UC9)



### Sơ đồ sequence cho [UC10](#_UCID:_UC10)

# : KẾT LUẬN

## Kết quả đạt được

Bằng sự trợ giúp về mặt lưu trữ của amazon web service như Dynamodb, amazon Simple Storage Service (s3), hiện tại handmade web server đã có thể sử dụng được các tính năng sau:

- Lấy được danh sách video theo chủ đề

- Lấy được video cụ thể thoe từ khóa tìm kiếm

- Thực hiện được chức năng đăng nhập

- Thực hiện được chức năng tạo tài khoản

- Lấy được danh sách tất cả các bài đăng của người đăng cụ thể

- Lấy được tất cả các bình luận dựa trên mỗi video của người đăng

- Thực hiện được việc đăng tải và lưu trữ video lên cơ sở dữ liệu đám mây

- Thực hiện được việc xóa bài đăng cụ thể

- Thực hiện được chức năng đăng xuất

- Có thể like video hay dislike video (có ứng dụng xử lý thời gian thực vào)

- Có thể đăng bình luận trên video cụ thể (có ứng dụng xử lý thời gian thực vào)

- Sau khi click vào xem video nào đó, sẽ load thêm danh sách video bên cạnh để tiện theo dõi

## Hạn chế của đồ án

Vì mặt hạn chế về thời gian (vì trong học kì có khá nhiều môn phải học bù những môn không đăng ký được ở học ký trước), trình độ vẫn chưa đủ, chưa có kinh nghiệm trong phát triển ứng dụng với ngôn ngữ bất đồng bộ. Nên sau đây là những mặt hạn chế của project:

Những tính năng hay chức năng chưa làm kịp:

1. Chức năng bình luận tuy có xử lý thời gian thực nhưng vẫn chưa viết kịp tính năng cho phép hiển thị ngay lập tức đến các người dùng khác đang xem cùng video.
2. Chức năng like hay dislike cũng có xử lý thời gian thực nhưng vẫn chưa viết xong tính năng cho phép hiển thị lượt like hay dislike ngay lập tức đến các người dùng khác đang xem cùng video tương tự.
3. Về trang chào mừng có viết widget lấy dữ liệu thời tiết, nhờ việc lấy api được cung cấp sẵn từ [Dark Sky Weather api](https://darksky.net/dev) tuy nhiên ngay từ lúc hiển thị lần đầu lại để nhiệt độ là độ F nhưng lại có ký tự độ C sau cùng.
4. Chức năng đăng bài mất khá nhiều thời gian để xử lý.
5. Chức năng xóa bài đăng cũng mất nhiều thời gian để xử lý.
6. Chức năng xóa video bị lỗi và không có thời gian debug.

## Hướng phát triển

Phát triển thêm Deep Learning để tự động lấy những video liên quan đến video đang xem thôi.

Cải tiến lại thuật toán để cải thiện thời gian đăng bài.

Tăng cường bảo mật, chủ yếu là sử dụng quyền xác thực bên thứ ba (có thể dùng api cung cấp sẵn hiện nay như google)

# TÀI LIỆU THAM KHẢO

Các tài liệu Tiếng Việt

-Không có

Các tài liệu Tiếng Anh

-Không có

Các tài liệu từ Internet

1. Website: <https://www.w3schools.com/>
2. Website : <https://codepen.io/>
3. Website : <https://github.com/expressjs/multer>

# PHỤ LỤC