MỤC LỤC

[MỤC LỤC 1](#_Toc117628326)

[CHƯƠNG 1 – TÌM HIỀU NUXT 2](#_Toc117628327)

[1.1. Cấu trúc thư mục 2](#_Toc117628328)

[1.2. Routing 2](#_Toc117628329)

[1.3. Data Fetching 3](#_Toc117628330)

[CHƯƠNG 2 TÌM HIỂU VỀ NESTJS 5](#_Toc117628331)

[2.1. Cấu trúc thư mục 5](#_Toc117628332)

[2.2. Tìm hiểu Middleware 5](#_Toc117628333)

[2.3. Tìm hiểu Guard 6](#_Toc117628334)

[2.4. Tìm hiểu JSON Web Token 7](#_Toc117628335)

[2.4.1. Khái niệm 7](#_Toc117628336)

[2.4.2. Cách thức hoạt động 7](#_Toc117628337)

[2.4.3. Tạo một jwt 8](#_Toc117628338)

[2.5. Access Token và Refresh Token 8](#_Toc117628339)

[2.5.1. Access Token 8](#_Toc117628340)

[2.5.2. Refresh token 8](#_Toc117628341)

[2.6. Tìm hiểu JSON 9](#_Toc117628342)

# CHƯƠNG 1 – TÌM HIỀU NUXT

* 1. Cấu trúc thư mục

Thư mục assets/ dùng chứa code chưa được compile như css, js, ảnh. Những asset này sau khi build lên sẽ được chạy qua webpack để compile lại.

Thư mục components/ dùng để chứa các component.

Thư mục layouts/ là nơi chứa file layout cho trang web như layout user, admin.

Middleware/ sẽ chứa các phần code logic dùng cho việc quyết định xem ai có thể vào trang nào trong ứng dụng hoặc sẽ làm gì khi user không có quyền cố tình truy cập vào url.

Pages/ đây sẽ là nơi định nghĩa các page trên trang web. Ở đây chỉ cần tạo ra các file .vue thì nuxtjs sẽ lo phần router.

Thư mục static/ chứa các file thuộc đạng static không bị ảnh hưởng bởi quá trình compile khi build.

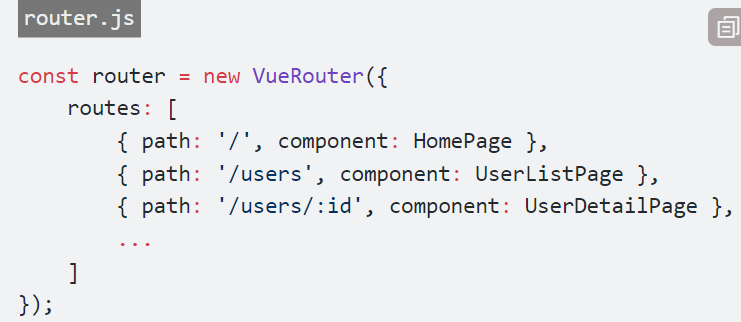
Store/ là nơi chứa code phục vụ cho sử dụng VueX.

Nuxt.config.js là file config riêng của nuxtjs.

* 1. Routing

Khi sử dụng Vuejs thì VueRouter là thư viện hỗ trợ trong việc định nghĩa việc truy cập url sẽ hiện ra page nào. Khi sử dụng Vue-Router thì ngoài việc tạo các component đại diện cho một page thì ta cũng đồng thời cần tạo cả một file cho việc routing.

Với Vuejs thì ta sẽ cấu hình file router như thế này.



Sau đó sử dụng nó trong file app.

Với Nuxtjs thì việc duy nhất ta cần làm là tạo ra component đại diện cho page mà bạn mong muốn, còn lại việc config giống như Vuejs sẽ tự động được nó xử lý. Cụ thể các url(path) trong project sẽ được tự động tạo ra dựa trên cấu trúc thư mục trong folder pages/

Với việc tạo cấu trúc folder file như trên thì tự Nuxt sẽ sinh ra file config router.js cho ta khi build ra.

Trong project không phải lúc nào url cũng dạng static mà còn cần các tham số nữa. Như trên trang vnExpress khi truy cập vào một bài viết thì url sẽ có dạng news/[new\_id]. Mỗi bài viết sẽ có new\_id khác nhau.

Bằng cách đặt tên file có bắt đầu bằng dấu \_ ta sẽ tạo ra một dynamic router. ví dụ:

Pages/

-------| new/

-------------|\_id.vue

Để điều hướng đến các trang trong ứng dụng ta nên sử dụng NuxtLink. Tương tự với tẻ a trong html ngoại trừ dùng href=”about” thì ta dùng to=”/about”.

* 1. Data Fetching

Để thực hiện việc fetching dữ liệu ta gọi trong hàm mouted().

mounted(){

      axios.get(‘’)

}

Ngoài ra còn có các hàm mặc định khác mà ta hay gặp: data(): khởi tạo dữ liệu, methods(): khởi tạo các phương thức, computed() thực hiện đồng thời với server.

* *Quản lý api trong nuxt*

Ta sẽ tìm hiểu VueX là gì?

“VueX là thư viện giúp quản lý trạng thái các component trong Vue.js, là nơi lưu trữ tập trung cho tất cả các component trong một ứng dụng, với nguyên tắc trạng thái chỉ có thể được thay đổi theo kiểu có thể dự đoán.”

Vuex có 5 core concepts:

* State
* Getters
* Mutations
* Actions
* Modules

Sử dụng store trong nuxt để quản lý api thông qua vuex bằng cách tạo file index.js. Rồi export các core concepts ở trên.

Getters nó sẽ lấy ra giá trị của state trong store. Ở trong component ta có thể truy cập thông qua $store.getters.property.

Cách duy nhất để thay đổi state là bằng cách sử dụng “commit”. Ta có thể sử dụng $store.commit(“mutationName”, payload).

Actions thường là nơi chứa các hành động như lấy dữ liệu từ API hay một hành động làm tay đổi dữ liệu trong CSDL. Actions được sử dụng trong các component. $store.dispatch(‘actionName’,payload).

# TÌM HIỂU VỀ NESTJS

* 1. Cấu trúc thư mục

Khi tạo nesjs ta cần để ý đến thư mục src với các file được tạo sẵn:

Main.ts: File để khởi tạo các đối tượng chạy ứng dụng, chẳng hạn có thể dùng lệnh NestFactory.create() để tạo instance cho Nest.

App.module.ts: Là module gốc của ứng dụng, có trách nhiệm đóng gói mọi thức có trong project.

App.controller.ts: Chứa các router để xử lý các request và trả về response cho client.

App.services,ts: Chứa các hàm xử lý logic chi service, chẳng hạn như ứng dụng có service kết nối đến CSDL hoặc xử lý file…

Nhìn chung có 3 thành phần chính trong NestJS: controller, provider và module.

Controller: Khi có request HTTP đến, cơ chế routing sẽ chuyển request này đến controller tương ứng để sử lý và trả về response thích hợp. Để xác định method cụ thể cho một request ta định nghĩa @GET() trên hàm. Việc khai báo này sẽ giúp Nest có thể ánh xạ request đến Get:/ đến hàm. Ngoài Get, nest cũng cung cấp đầy đủ các phương thức khác như Post, Delete, Put, Patch.

Provider là thành pần cơ bản trong Nest, một provider có thể được đưa vào làm một dependency để tạo ra mối liên hệ giữa nhiều đối tượng khác nhau. Các lớp như service, repository hay helper có thể được xem như provider bằng cách thêm decorator @Injectable().

Module được thiết kế để đóng gói các thành phần một cách độc lập. Module trong Nest là class được định nghĩa với decorator @Module() của Nest, dùng để mô tả các thuộc tính như controller, provider hay dependency của module.

* 1. Tìm hiểu Middleware

Middleware có thể thực hiện các nhiệm vụ sau:

* Thực hiện các thay đổi với request và response object.
* Gọi hàm middleware tiếp theo.
* Nếu hàm middleware hiện tại không kết thúc chu kỳ request-response, nó phải gọi next() để chuyển quyền điều khiển cho hàm middleware tiếp theo.

Để triển khai một Nest middleware, ta viết một function hoặc class với decorator @Injectable() và cần implement interface NestMiddleware.

Ở một số framework như express thì middleware thường được sử dụng để xác thực user, permission, validate data,… nhưng ở NestJS việc xác thực hay validate đã có các thành phần riêng biệt đảm nhận nhiệm vụ này như **Pipes** và **Guards** nên không cần phải dùng đến Middleware nữa. Tuy nhiên không chỉ có xác thực mà middleware còn làm rất nhiều nhiệm vụ khác tuỳ theo mục đích xử lý của api.

Về **Pipes**: Trước tiên, một pipe được định nghĩa là một class sử dụng decorator @Injectable(), và implement từ **pipeTransform**. Pipe thường được sử dụng trong hai trường hợp cơ bản sau:

* Transformation: Chuyển đổi dữ liệu đầu vào thành dạng dữ liệu mong muốn, ví dụ chuyển từ dạng string sang integer.
* Validation: Kiểm tra dữ liệu đầu vào và báo lỗi nếu như dữ liệu đó không toả điều kiện.

Về **Guards**: Một Guard là lớp sử dụng decorator @Injectable(). Các guards phải implement **CanActivate**. Tại đây các request đến sẽ pải đi qua bước kiểm tra về permission, roles, ACLs.

* 1. Tìm hiểu Guard

NestJs cung cấp khái niệm Guards nhằm cho phép chúng ta xây dựng logic nhằm quyết định xem khi nào thì một request được thực thi.

Mọi guard cần phải implement CanActivate(). Hàm này trả về một boolean, cho biết request hiện tại có được phép hay không. Nó có thể trả về response đồng bộ hoặc không đồng bộ (thông qua Promise hoặc Observable). Nest sử dụng giá trị trả về để kiểm soát hành động tiếp theo:

* Nếu trả về true, request sẽ được thực hiện.
* Nếu trả về false, Nest sẽ từ chối request.

Ta có thể tạo nhiều Guard khác nhau tuỳ mục đích bảo vệ route.

* 1. Tìm hiểu JSON Web Token
     1. Khái niệm

Json web token là chuỗi mã hoá giúp ta trao đổi thông tin giữa các hệ thống khác nhau một cách an toàn. Các chuỗi thông tin tin sẽ được mã hoá một cách ngẫu nhiên.

JWT gồm 3 thành phần: Header, Payload, Signature. Những thành phần này được ngăn cách bởi ký tự “.”.

Header chứa kiểu dữ liệu và thuật toán được sử dụng nhanh chóng để mã hoá ra chuỗi JWT. Header gồm 2 thành phần chính:

* Typ (Type): Loại token và được mặc định là JWT.
* ALG(Algorithm): Được xem là thuật toán mới, sử dụng để mã hoá nhanh chóng (các thuật toán phổ biến HMAC, SA256, RSA).

Phần tiếp theo của token chính là payload. Nó đóng vai trò rất quan trọng. Đây là nơi chứa nội dung của thông tin. Payload chứa các claims. Claims là các biểu thức về thực thể (chẳng hạn như user) và một số metadata phụ trợ. Có 3 loại claims thường gặp trong payload: reserved, public và private claims.

Signature là phần chữ ký bí mật. Bản thân signature đã bao gồm cả header và payload nên signature có thể dùng để kiểm tra tính toán vẹn của dữ liệu khi truyền tải.

* + 1. Cách thức hoạt động

Trong việc xác thực, khi người dùng đăng nhập thành công, hệ thông sẽ thực hiện tạo mã JWT lưu trên browser của user.

Khi người dùng muốn truy cập vào route được bảo vệ (chỉ có người dùng đăng nhập mới được phép), phía server sẽ thực hiện kiểm tra JWT có hợp lệ không. Người dùng sẽ cần có jwt để có thể xác thực các request tiếp theo.

* + 1. Tạo một jwt

Trong Nest, ta có thư viện để khai báo một jwt.

Khi người dùng đăng nhập thành công và gửi thông tin lên server thì server sẽ lấy thông tin đó + với mã secret key để tạo ra token và gửi lại cho client.

Trong nest sẽ tạo như sau:

const jwt = jwtService.sign([dữ liệu người dùng], {secret: secret\_key },{expiresIn:’1h’}).

* 1. Access Token và Refresh Token

AccessToken và Request Token về bản chất đều là JSON Web Token.

* + 1. Access Token

Access token sẽ hết hạn sau 1 khoảng tời gian ngắn. Điều này hạn chế rủi ro khi hacker có được access token, họ cũng không phá hoại được nhiều.

Các loại mã access token:

* User Access Token: Đây là một loại mã truy cập người dùng được sử dụng phổ biến nhất nó có tính năng gọi API để đọc, sửa và ghi dữ liệu của người dùng.
* Page Access Token: Đây là một loại mã truy cập trang cho phép ghi, sửa dữu liệu thuộc về một trang.
* App Access Token: Đây là mã truy cập ứng dụng để sử đổi và đọc cài đặt ứng dụng.
  + 1. Refresh token

Ví dụ để hiểu rõ tại sao cần refresh token:

Người dùng đăng nhập vào hệ thống và sau khi xác thực thành công, người dùng nhận được một mã token và mã này có thời hạn sử dụng (10 phút).

Trong mỗi lần gọi API tiếp theo, người dùng phải gửi theo mã token trong request để truy cập đến các tài nguyên khác.

Khi hết thời gian, người dùng lại phải đăng nhập lại để nhận mã token mới. Ví dụ nếu là facebook mà cứ 10 phút lại bắt người dùng đăng nhập lại thì sẽ rất phiền phức với người dùng.

Vì thế refresh token ra đời có nhiệm vụ tạo 1 access token mới từ server. Tất nhiên, nếu để lộ refresh token thì cũng bị hack, nhưng rõ ràng access token sử dụng với tuần suất lớn còn refresh token thì khi nào access token hết hạn thì mới dùng đến.

* 1. Tìm hiểu JSON

Json là định dạng giúp lưu trữ các thông tin có cấu trúc và nó chủ yếu được dùng để truyền tải dữ liệu giữa server và client.

Json sử dụng các cặp key-value để sử dụng dữ liệu. Khi đó nó hỗ trợ cho những cấu trúc dữ liệu dạng đối tượng và mảng.

Hiện nay chuỗi JSON thường được mã hoá với 2 dạng: Json\_encode và Json\_decode.

Chuỗi Json được trình bày trong dấu ngoặc {}, các cặp key và value của Json phải đáp ứng điều kiện nằm trong dấy nháy kép “”. Nếu như người dùng đặt nó trong dấu ngoặc hoặc ngoặc đơn thì không còn là chuỗi Json chuẩn xác nữa.

* Cấu trúc chuỗi dữ liệu Json

Object trong json được biểu hiện bằng dấu ngoặc nhọn {}. Định nghĩa Object trong Json cũng khá tương đồng với Object trong Javascript. Tuy nhiên, Object trong json vẫn phải tuân theo:

* Key: phải luôn nằm trong dấu ngoặc kép, không nên là biến số.
* Value: Chỉ cho phép các loại dữ liệu cơ bản: numbers, string, boolean, array, object, null. Không cho phép function, date, undefined.

Các kiểu Json

* Kiểu Object

var student =

{

“name”: “A”,

“age”: “20”

}

* Kiểu Object in array

var students =

[

{

“name”:”A”,

“age”:”20

},

{

“name”:”B”,

“age”:”20

}

]

* Kiểu nest object

var students =

{

“A”:{“name”:”A ”, “age”:”20”}

“B”:{“name”:”B”, “age”:”20”}

}

Một group data json là một mảng gồm mà mỗi phần tử đều có các key giống nhau.

