# 127. Setting up and Loading Routers

Vì chia Routes là một phần quan trọng trong Angular App nên ta có thể đăng kí trong AppMudules.

Các bước đăng kí chia routes:

**B1:** khai báo biến với kiểu **Routes** (import { **Routes** } from ‘@angular/router’): const appRoutes: Routes

**B2:** Biến appRoutes được khai báo ở bước 1 sẽ chứa một array.

**B3:** Mỗi phần tử trong array thuộc appRoutes là một obj chứa thông tin config của các routes:

Const appRoutes = [

{

path: ‘tên đường dẫn URL’,

component: <Tên component sẽ hiển thị tương ứng với Route>

}

]

Tới bước này căn bản đã setup xong cho biến appRoutes nhưng nó vẫn chưa có tác dụng gì (vì angular sẽ bỏ qua nó). Vì vậy chúng ta sẽ đăng ký nó với Angular App (bước 4).

**B4:** Đăng ký các routes đã khai báo bằng cách thêm **RouterModule** vào mục imports trong config của @NgModule:

Import {**RouterModule**} from ‘@angular/router’

@NgModule({

Import: [

**RouterModule**

]

})

Tới đây đã import thành công RouterModule nhưng các routes **vẫn chưa được đăng ký thành công.**

**B5:** Để đăng ký thành công chúng ta phải gọi phương thức **forRoot()** của **RouterModule** và truyền vào biến **appRoutes** cho phương thức này**:**

****

Import {**RouterModule**} from ‘@angular/routes’

@NgModule({

Import: [

RouterModule**.forRoot(appRoutes)**

]

})

Tới đây angular đã hiểu được các routes mà ta đã khai báo. Nhưng còn một vấn đề nữa là: với mỗi route thì việc hiển thị các component sẽ như thế nào (tại vị trí nào) trong html template.

Để đánh dấu vị trí hiển thị của component tương ứng với routes trong html chúng ta dùng directive: **<router-outlet></router-outlet> trong html file.**

# 128. Navigating with Router Links

Hiện tại sau khi làm hết bước 5 của phần 127 chúng ta có thể chuyển trang được (nhưng chỉ bằng cách nhập URL trên address bar của trình duyệt).

Điều này không thật sự thuận tiện. Chúng ta có thể cải thiện điều này với việc chuyển trang bằng cách nhấn nút “chuyển trang” trên giao diện.

Để làm được nút chuyển trang chúng ta có thể dùng thẻ html **<a href=”<tên đường dẫn>”></a>.** Với cách làm này chúng ta đã có thể chuyển tranh thành công. Nhưng có một vấn đề phát sinh – trang web **load mới** sau khi chuyển trang – chúng ta refresh lại toàn bộ trang web với mỗi link mà chúng ta click. Các thành phần khác không cần reload vẫn bị reload làm cho các thuộc tính nó lưu bị mất. Ngoài ra điều này cũng trái với mục đích của Angular – Single Page Application.

Để giải quyết vấn đề load lại trang. Trong thẻ html <a> thay vì sử dụng thuộc tính href. Chúng ta sẽ dùng directive routerLink.

<a **routerLink**=”tên đường dẫn URL”></a>

Một cách khác để sử dụng routerLink là sử dụng như properties binding:

<a [**routerLink]**= ” **‘tên đường dẫn URL’** ”></a>

Hoặc: <a **routerLink**=”[ **‘tên đường dẫn URL’**, **‘giá trị gì đó’** ]”></a> ([https://localhost:4200/<tên](https://localhost:4200/%3ctên) đường dẫn>/<giá trị nào đó>

Với việc sử dụng routerLink chúng ta đã chuyển trang thành công mà không load lại trang.

# 129. Understanding Navigation Paths

Xét các trường hợp truyền giá trị cho routerLink:

**Trường hợp 1:** routerLink=”</giá trị đường dẫn tuyệt đối>” - đây là đường dẫn tuyệt đối. Với loại đường dẫn này kết quả đường dẫn mới trả về sẽ là: [**https://localhost:4200/”giá**](https://localhost:4200/) **trị đường dẫn tuyệt đối”**

**Trường hợp 2:** routerLink=”<giá trị đường dẫn tương đối>” – đây là đường dẫn tương đối. Với loại đường dẫn này kết quả đường dẫn mới trả về sẽ là: **https://localhost:4200/“giá trị đường dẫn hiện tại” + “giá trị đường dẫn tương đối gán cho routerLink”**

Trong đường dẫn tương đối để lùi lại một level trong đường dẫn URL chúng ta có thể dùng: **../**tên đường dẫn mới.

# 130. Styling Active Router Links

Trong thẻ html <a> chúng ta có thể sử dụng:

* **routerLinkActive=”active”**
* **[routerLinkOption]=”{exact: true}”**

**Trong đó active là class được CSS cho hiệu ứng active**

# 131. Navigating Programmatically

Ở các phần trước chúng ta đã có thể chuyển trang bằng directive trong html file (được kích hoạt với hành vi bấm nút của người dùng). Nhưng để **chuyển trang bằng code trong file .ts** thì làm như thế nào? (Sẽ áp dụng trong trường hợp chương trình xử lý một logic nào đó và khi thực hiện xong thì tự chuyển trang luôn mà không cần sự tác động của người dùng).

Cách làm:

* B1: trong file .ts của component chúng ta sẽ inject instance của Router

**Import { Router } from ‘@angular/router’;**

@Component({...})

Export class <tên class> {

Constructor(**private router: Router**){}

}

* B2: Ta có thể khai báo một phương thước để chuyển trang như sau:

onNavigate() {

**this.router.navigate([‘/đường dẫn tuyệt đối’]);**

}

# 132. Using Relative Paths in Programmatic Navigation

Khác với directive **routerLink** được sử dụng trong **html template.** Phương thức **navigate()** được sử dụng trong **.ts file** không nhận biết được đường dẫn hiện tại của trang web nên dù có truyền đường dẫn tương đối hay tuyệt đối vào làm đối số của phương thức navigate() thì cũng như nhau. Để phương thức navigate() có thể nhận biết được đường dẫn hiện tại của trang web, cách làm như sau:

* B1: import {Router, **AcitvatedRoute**} from ‘@angular/router’
* B2: Inject ActivateRoute vào class component:

Constructor(private router: Router, **private route: ActivatedRoute**) {}

* B3: Trong phương thức navigate ngoài đường dẫn muốn chuyển đến chúng ta sẽ truyền thêm một đối số thứ 2.

onNavigate() {

this.router.navigate([‘<tên đường dẫn>’], **{relativeTo: this.router}**)

}

Giờ thì phương thức navigate đã có thể biết được URL hiện tại của trang web và vì thế chúng ta có thể sử dụng được đường dẫn tương đối trong trường hợp muốn chuyển trang bằng code typeScript

# 133. Passing Parameters to Routers

Nhiều trường hợp đường dẫn URL cần có các params là các giá trị được dùng để query ra các data như mong muốn cho trang web. Trường hợp này đang nói tới PATH VARIABLE.

Để định nghĩa được dynamic các path variable trong URL, cách làm như sau:

Trong phần định nghĩa routes (phần 127)

Const appRoutes = [

{

Path: ‘’user/**:id**/**:name”, (:id ở đây là chính là PATH VARIABLE - giá trị của id User mà ta muốn query, nhớ có dấu “:” ở phía trước – tương tự với “:name”)**

}

]

# 134. Fetching Route Parameters

Để truy cập giá trị của các param trong route URL, trong file .ts chúng ta làm như sau:

* B1: import {**ActivatedRoute**} from ‘@angular/router’;
* B2: inject activatedRoute vào class component:

Constructor(**route: ActivatedRoute**) {}

* B3: Trong ngOnInit() chúng ta sẽ truy cập vào các param trong URL, bằng cách:

**this.route.snapshot.params[‘tên param’]**

ví dụ: **this.route.snapshot.params[‘id’]**

# 135. Fetching Route Parameters Reactively

Ở phần trước, với cách dùng phương thức **Snapshot** của route chúng ta đã có thể lấy được param trong URL. Nhưng vấn đề là nếu chúng ta muốn query dữ liệu khác (thay đổi giá trị param trong URL – format của URL không đổi (đang ở chính trang đó)– trong trường hợp 2 lần query liên tiếp) thì data trên trang vẫn giữ nguyên mà không đổi theo ý muốn. Lý do là vì angular nó nhận biết được chúng ta mặt dù thay đổi đường dẫn nhưng không hề thay đổi component để hiện thị. Nên nó sẽ không destroy component hiện tại và khởi tạo lại để tối ưu perform.

Cách sử lý như sau, dùng observeble để bắt sự kiện thay đổi URL và cập nhật lại param:

Import {ActivedRoute, **Params**} from ‘@angular/router’

ngOnInit(private **route**: AcitvedRoute){

// đoạn này để truy cập giá trị các param trong lần đầu khởi tạo trang

this.user = {

id: this.route.snapshot.params[‘id’],

name: this.route.snapshot.param[‘name’]

}

//đoạn này để bắt sự kiện thay đổi URL

this.route.params.subscribe(

(params: **Params**) => {

This.user.id = params[‘id’]

This.user.name = params.[‘name’]

}

)

}

# 136. An Important Note about Route Observables

Vì chúng ta dùng Observeble để bắt sự kiện thay đổi URL trong OnInit() nên đừng quên phải unSubcribe nó trong onDestroy() của component.

**paramsSubscription**: Subscription = null

ngOnInit(){

// đoạn này để truy cập giá trị các param trong lần đầu khởi tạo trang

this.user = {

id: this.route.snapshot.params[‘id’],

name: this.route.snapshot.param[‘name’]

}

//đoạn này để bắt sự kiện thay đổi URL

**This.paramsSubscription** = This.route.param.subcribe(

(params: **Params**) => {

This.user.id = params.[‘id’]

This.user.name = params.[‘name’]

}

)

}

ngOnDestroy() {

**this.paramsSubscription.unsubcribe()**

}

# 137. Passing Query Parameters and Fragments

Ngoài dạng param được truyền vào URL đã trình bày ở những phần trên (path variable) – ví dụ: (localhost:4200/users/:id). Ta còn một dạng nữa dưới dạng câu query (localhost:4200/user@id=2&name=Jane). Phần này sẽ trình bày về vấn đề làm việc với dạng query param (dạng 2).

**Làm thể nào để tạo ra URL chứa query params và sử dụng để chuyển trang trong html template:**

Trong html template ngoài dùng directive **routerLink** chúng ta sẽ dùng thêm **queryParams** và truyền vào một obj có các cặp key và value tương ứng với query param muốn tạo. Muốn thêm fragment thì dùng directive **fragment**

<a

**[routerLink]=” [‘/server’, 5, ‘edit’] ”,** //=>localhost:4200/server/5/edit

**[queryParams]=” {allowEdit: 1} ”** //=>localhost:4200/server/5/edit?allowEdit=1

**fragment** = “loading” //=>localhost:4200/server/5/edit?allowEdit=1#loading

></a>

**Làm thể nào để tạo ra URL chứa query params và sử dụng để chuyển trang trong .ts file:**

Trong phương thức chuyển trang chúng ta sẽ làm như sau:

Constructor(private **router**: Router){}

onNavigate(id: number) {

**this.router.navigate([‘/servers’, 5, ‘edit’], {queryParams: {allowEdit: ‘1’}, fragment: ‘loading’});**

}

# 138. Retrieving Query Parameters and Fragments

Cách để truy cập queryParams và fragment từ URL:

B1: Import {ActivatedRoute} from ‘@angular/routes’

B2: Constructor(private routes: ActivatedRoute){}

B3: truy cập trong ngOnInit()

ngOnInit() {

console.log(**this.route.snapshot.queryParams**)

console.log(**this.route.snapshot.fragment**)

**this.routes.queryParams.subcribe((queryParams: Params) => {**

**console.log(queryParams[‘<tên param>’])**

**})**

**this.routes.fragment.subcribe(fragment => {})**

}

# 139. Practicing and some Common Gotchas

Luyện tập cách pass params vào URL và truy cập các param này trong .ts file

Lưu ý khi acess params từ URL nhớ parse ra đúng dạng khi sử dụng để tránh lỗi, vì khi access params từ URL giá trị mặt định kiểu string (có thể parse nhanh ra kiểu number bằng cách thêm toán tử “+” trước giá trị string).

# 140. Setting up Child (Nested) Routes

Khi tạo ra các routes để đăng kí, đôi khi ta sẽ gặp các routes có cùng tiền tố, như code bên dưới (cùng tiền tố “users”):

Const appRoutes: Routes = [

{path: ‘users’, component: UsersComponent},

{path: ‘users/:id’, component: UserComponent},

{path: ‘users/:name’, component: UserComponent},

]

Để tránh việc trùng lặp chúng ta sẽ sử dụng thêm properties “children” trong các obj config routes:

Const appRoutes: Routes = [

{ path: ‘users’, component: UsersComponent, **children:** [

{path: **‘:id’**, component: UserComponent},

{path: **‘:name’**, component: UserComponent},

]},

]

***Lưu ý:*** Các component của các routes con phải được load lồng trong các component của routes cha. Nên trong html template của routes cha phải có <router-outlet></router-outlet> để load component của routes con.

# 141. Using Query Parameters – Practice

Không có gì chỉ là luyện tập.

# 142. Configuring the Handling of Query Parameters

Trong quá trình Navigate từ trang này sang trang khác, đôi khi chúng ta muốn bảo toàn các query params thuộc URL của route hiện tại, cách làm như sau:

Trong phương thức chuyển trang:

onNavigate() {

this.router.navigate([‘<đường dẫn>’], { relativeTo: this.route, **queryParamsHandling: ‘preserve’**})

}

# 143. Redirecting and Wildcard Routes

Đôi khi người dùng sẽ cố truy cập vào một đường dẫn mà chúng ta chưa định nghĩa, lúc đó ứng dụng sẽ báo lỗi. Vì vậy chúng ta cần phải rào lại trường hợp này.

Khi người dùng cố gắng truy cập vào các đường dẫn chưa định nghĩa chúng ra sẽ phát hiện và chuyển hướng (redirect) họ tới trang 404 not found do chúng ta định nghĩa.

Cách làm:

B1: tạo ra một route dẫn tới trang not found:

{ path: **‘not-found’**, component: NotFoundComponent}

B2: Bắt tất cả các đường dẫn không hợp lệ và redirect qua trang not-found;

{ path: ‘**\*\***’, redirectTo: **‘/not-found’**}

**(\*) Đảm bảo rằng route này nằm cuối cùng trong mảng các routes vì nếu không tất cả các routes đứng sau nó đều sẽ không truy cập được – vì bị redirect tới trang not-found. Path: ‘\*\*’ nghĩa là tất cả các path có thể tồn tại, để nó ở cuối cùng của các route nghĩa là “tất cả các path có thể tồn tại trừ các path thuộc các route ở trên” – vì angular sẽ quét các route tuần tự từ trên xuống dưới.**

# 144. Important: Redirection Path Matching

In our example, we didn't encounter any issues when we tried to redirect the user. But that's not always the case when adding redirections.

By default, Angular matches paths by prefix. That means, that the following route will match both “**/recipes**”  and just “**/**”

**{ path: '', redirectTo: '/somewhere-else' }**

Actually, Angular will give you an error here, because that's a common gotcha: This route will now **ALWAYS** redirect you! Why?

Since the default matching strategy is "prefix" , Angular checks if the path you entered in the URL does **start with the path** specified in the route. Of course every path starts with ''  (Important: That's no whitespace, it's simply "nothing").

To fix this behavior, you need to change the matching strategy to "full" :

**{ path: '', redirectTo: '/somewhere-else', pathMatch: 'full' }**

Now, you only get redirected, if the full path is ''  (so only if you got NO other content in your path in this example).

# 145. Outsourcing the Route Configuration

Tới thời điểm này thì biến appRoutes của chúng ta đã lưu rất nhiều routes và nó sẽ chiếm một không gian lớn trong file app.module.ts

Vì vậy ta nên duy chuyển các khai báo routes này sang một file mới (đặt tên là app-routing.module.ts) đặt trong cùng component

Trong file app-routing.module.ts:

Import { NgModule } from ‘@angular/core’

Import { Routes, RouterModule } from ‘@angular/router’

Import { HomeComponent } from ‘<đường dẫn tới home component>’

...

//Cắt phần khai báo routes từ app.module.ts qua đây

Const appRoutes:Routes = [

{path: ‘’, component: HomeComponent},

{path: ‘users’, component: UsersComponent},

{path: ‘servers’, component: ServerComponent}

]

@NgModule({

Imports: [

RouterModule.forRoot(appRoutes)

],

Export: [RouterModule]

})

Export class AppRoutingModule {

}

Trong file app.module.ts:

Xóa: RouterModule.forRoot(appRoutes) khỏi mục imports trong @NgModule

Thêm: AppRoutingModule vào phần imports trong @NgModule

Tới đây chúng ta đã di chuyển được phần khai báo và đăng ký routes sang một file mới, từ đó dễ đọc và quản lý code hơn.

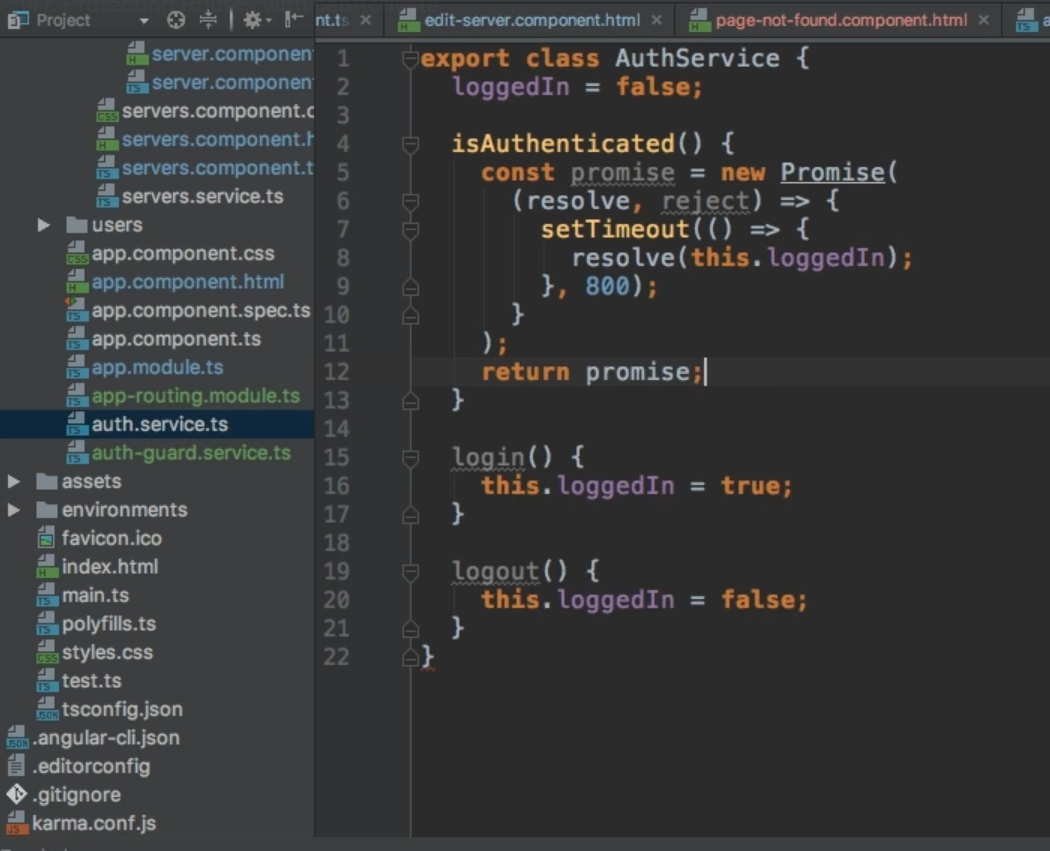
# 146. An Introduction to Guards

Chỉ là phần giới thiệu. Bảo vệ các route – chỉ những ai có quyền mới được truy cập.

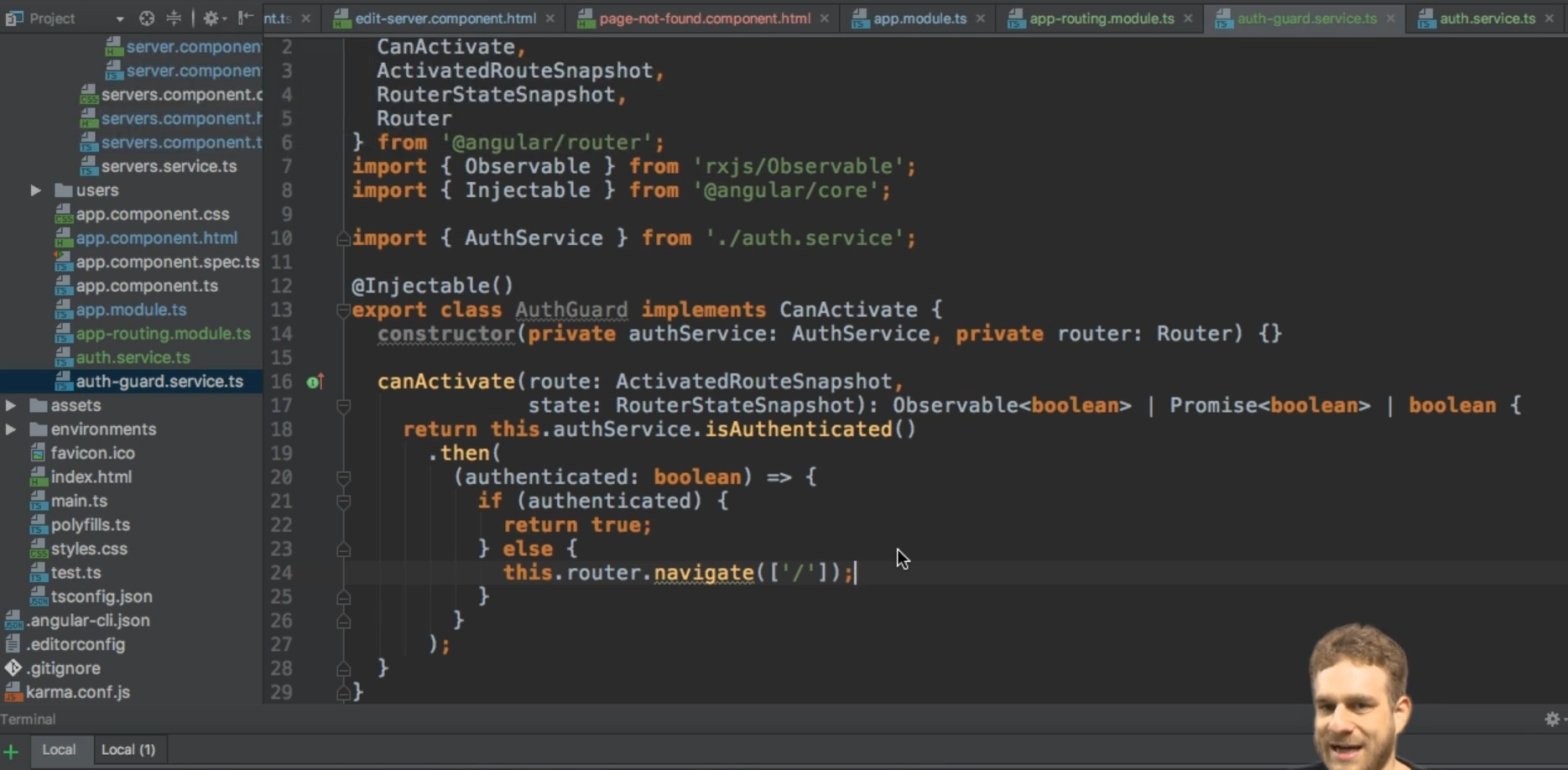
# 147. Protecting Routes with canActivate

Demo cách thực hiện thông qua code:

B1: Tạo 1 service để fake việc log in – log out và kiểm tra log in – log out như sau:



B1: Tạo Class AuthGuard được implement từ CanActivate có sử dụng Service đã tạo ở bước 1 như sau:



B3: Tại route mà ta muốn bảo vệ (phải có quyền truy cập mới được vào), chúng ta làm như sau:



Sau bước 3: Route mẹ được thêm thuộc tính canActivate và tất cả router con thuộc route mẹ đó sẽ cần quyền để được truy cập.

User chỉ có thể truy cập được các route được bảo vệ nếu phương thức canActivate thuộc class AuthGuard ở bước 2 trả về giá trị TRUE.

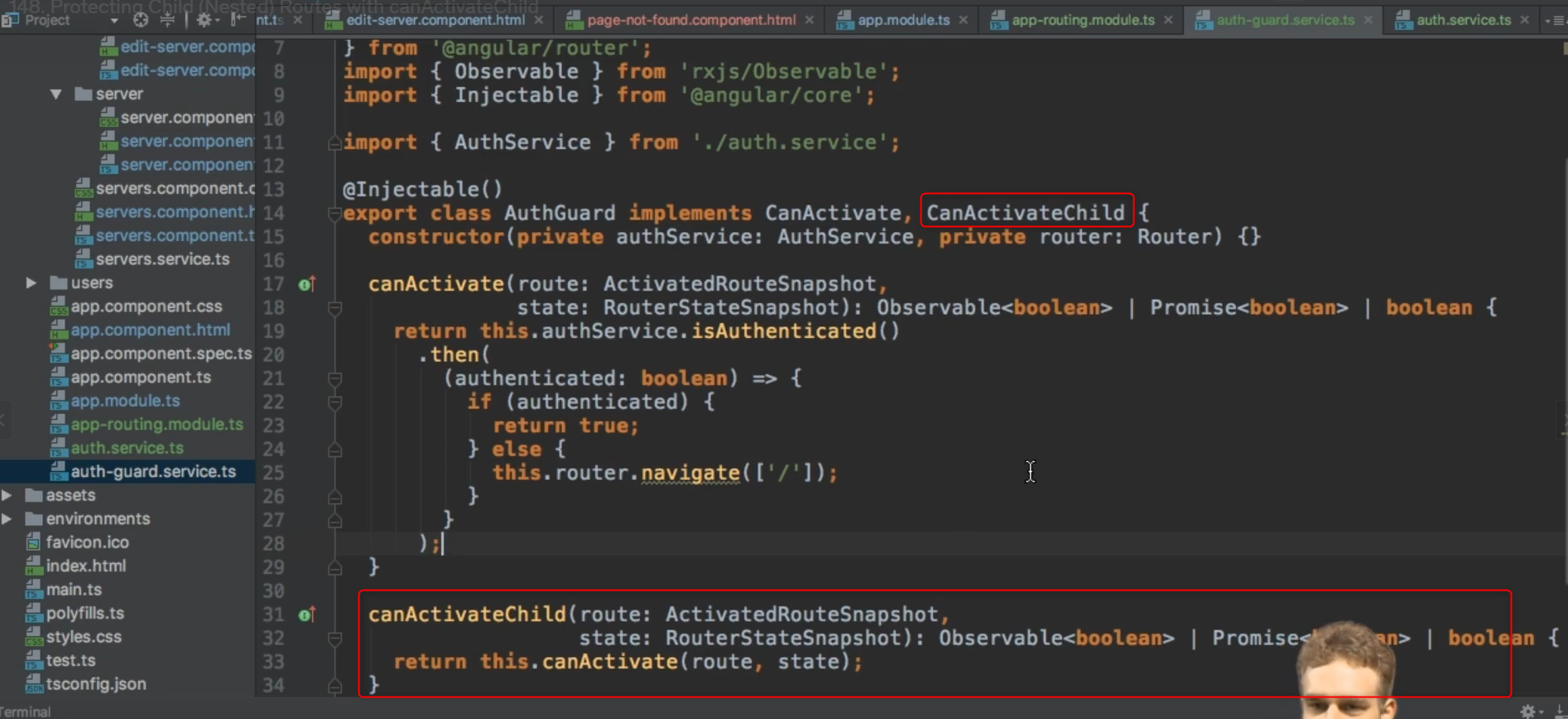
# 148. Protecting Child (Nested) Routes with canActivateChild

Đặt trường hợp 1 route cha có nhiều route con và chúng ta chỉ muốn bảo vệ các route con trước sự truy cập trái phái (còn route cha thì không).

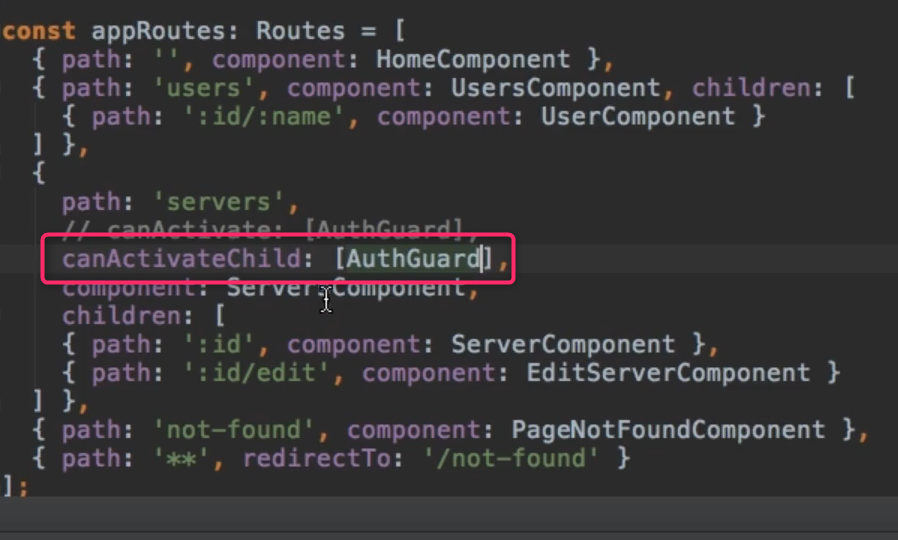
Chúng ta có thể đặt thuộc tính canActivate vào từng route con để giải quyết vấn đề này, tuy nhiên một route cha có thể có rất nhiều route con và theo thời gian số route con sẽ tăng lên nên nếu làm theo cách này chúng ta sẽ phải thêm rất nhiều thuộc tính canActivate vào từng route con (stupid way).

Angular có hỗ trợ cách giải quyết cho vấn đề này một cách hợp lý hơn – canActivateChild.

Để sử dụng được canActivateChild ta cho class AuthGuard implement từ canActivateChild như sau:



Tiếp theo ta config các route như sau:



Ở đây chỉ cần chúng ta thêm thuộc tính canActivateChild vào route cha thì tất cả các route con sẽ được được bảo vệ (ngoại trừ route cha).

# 149. Using a Fake Auth Service

# 150. Controlling Navigation with canDeactivate

Ở chương trước chúng ta đã tìm hiểu cách kiểm soát người dùng truy cập vào một route. Chương này chúng ta sẽ tìm hiểu về cách kiểm soát việc người dùng thoát khỏi một route.

Ví dụ về trường hợp áp dụng: Người dùng điền form nhưng quên submit đã thoát ra. Chúng ta có thể chặn hành động thoát ra của người dùng để đảm bảo form chứa thông tin quan trọng được submit thành công.