



4

# Angular Routing And Modules

# Angular Routing and Modules

الكتاب الرابع من

سلسلة تعلم Angular بالعربي

تم انتاج هذا العمل في عام

٢٠٢٠

المؤلف

فيصل الفهد

وادي التقنية © النسخة الأولى 2020

هذا العمل مرخص بموجب رخصة المشاع الإبداعي: نسب المصنف –  
غير تجاري – الترخيص بالمثل 4.0 الدولي



### إهداء

إلى أطهر قلبين في حياتي... والديّ العزيزين.

إلى من شاركني السراء والضراء، ولم أرها عابسة يوماً..... زوجتي المخلصة.

إلى من أتشوّق لأن أرى مستقبلهما المشرق بإذن الله..... ابنائي وبناتي.

إلى جميع متلهف للعلم ويتوق للمعرفة

أهديكم هذا الكتاب المتواضع، وأدعو الله أن يحوز إعجابكم.

المؤلف



## مقدمة السلسلة

اللهم علمنا ما ينفعنا وانفعنا بما علمتنا إنيك انت العليم الحكيم..

أضع بين إيديكم أحبتي وأخوتي مطوري الويب وبالتحديد مطوري Front End أول سلسلة عربية تتكلم عن إطار عمل Angular، حيث تقوم فكرة هذه السلسلة على حصر جميع الميزات التي يُقدمها Angular ومن ثم في كل كتاب يتم أخذ ميزة او ميزتين والإبحار فيها بشكل مستقل محاولاً بذلك تغطية أغلب وأهم جوانبها مع الشرح المفصل والأمثلة المتعددة.

ومن هذا المنطلق يجب التنويه أن هذه السلسلة ليست للمبتدئين في البرمجة، وانما لابد ان تمتلك عزيزي المتعلم على الأقل أساسيات أركان تطوير الويب الثلاثة HTML-CSS-JavaScript ومن ثم لابد ان تمتلك اساسيات Typescript لأن إطار عمل Angular يعتمد على هذه التقنية، ولا يخفى على كل مطور واجهات أمامية أهمية Typescript والتي من وجهة نظري أرى انه يجب على كل مطور ويب تعلمها واتقانها، ونكتفي ان نقول ان Deno.js وهي اللغة الجديدة من Node.js أصبحت تدعم هذه التقنية بشكل كامل، لذلك انصح كل مطور عربي بتعلم هذه التقنية ومحاولة الإلمام بجوانبها وكيفية الاستفادة منها في إطار عمل Angular بشكل خاص ولتطوير الويب بشكل عام.

ولا يخفى عليك عزيزي المتعلم أن الكمال لله ولا يخلوا عمل من الأخطاء فالخطأ وارد والتقصير موجود، لذلك في حال وجدت أي تقصير او أي خطأ في هذه السلسلة او أردت تقديم أي اقتراح لتطوير هذه السلسلة فلا تتردد بمراسلتي على البريد الإلكتروني الذي سوف اضعه بأسفل هذه الصفحة.

وأخيراً أسأل الله العلي العظيم أن ينفع به وان يتقبله عملاً خالصاً لوجهه سبحانه،

ولا تنسوني أخوتي من صالح دعائكم.

المؤلف

فيصل الفهد

Faisal.alfahd.ksa@gmail.com

# جدول المحتويات

١	مقدمة الكتاب
٢	الفصل الأول مدخل إلى Angular Routing
٣	1.1. مقدمة:
٤	2.1. تهيئة Routing:
٧	3.1. توضيح وفهم Angular Base href:
٩	4.1. Wildcard Routes:
١٨	5.1. الربط في Angular Routing:
١٨	1.5.1. الربط عن طريق ملف Template:
٢٣	2.5.1. الربط البرمجي:
٢٧	6.1. تصميم صفحات التطبيق وإضافة البيانات:
٢٧	1.6.1. إعادة تصميم الصفحات:
٣٠	2.6.1. إضافة البيانات:
٣٧	7.1. Children Routes:
٥٤	الفصل الثاني Parameters in Angular Routing
٥٥	1.2. مقدمة:
٥٥	2.2. Required Parameters:
٥٦	1.2.2. الربط وأرسال البارامترات عن طريق URL:
٦٠	2.2.2. استقبال البارامترات المرسلة عن طريق URL:
٧٣	3.2. Optional Parameters:
٧٩	4.2. Query Parameters:
٨١	1.4.2. إرسال Query Parameters عن طريق URL:
٩٢	5.2. Fragments Parameters:
١٠٠	6.2. NavigationExtras Interface:
١٠٠	7.2. الفرق بين أنواع البارامترات:
١٠١	8.2. Relative and Absolute Navigation:
١٠٤	9.2. خاصية skipLocationChange:
١٠٥	10.2. Router Events:
١١١	الفصل الثالث Route Guards
١١٢	1.3. المقدمة:
١٣٢	2.3. CanActivate Guard:
١٤١	1.2.3. UrlTree Type:

١٤٧	2.2.3. استكمال شرح CanActivate Guard
١٥٥	3.2.3. إرسال البيانات عن طريق الخاصية data
١٦٠	4.2.3. إعادة توجيه المستخدم إلى الصفحة التي كان فيها بعد تسجيل الدخول
١٦٤	3.3. CanActivateChild Guard
١٦٩	4.3. CanDeactivate guard
١٧١	1.4.3. CanDeactivate عامة لجميع Components في التطبيق
١٧٩	1.4.3. CanDeactivate مخصصة لـ Component محدد
١٨٢	5.3. CanLoad guard
١٨٢	6.3. Angular Resolve Guard
١٩٩	7.3. إخفاء وإظهار التبويبات بناء على صلاحيات المستخدم
٢٠٦	8.3. Route Guards Priority
٢٢٥	الفصل الرابع Angular Modules
٢٢٦	1.4. مقدمة
٢٣٢	2.4. Root Module
٢٣٣	3.4. Core Module
٢٤٤	4.4. Feature Module & Routing Module
٢٦٤	5.4. Shared Module
٢٦٧	الفصل الخامس Lazy Loding
٢٦٨	1.5. مقدمة
٢٦٩	2.5. خطوات تطبيق Lazy Loding على المشروع
٢٨٤	3.5. CanLoad Guard
٢٨٩	4.5. Preloading
٢٩٧	المراجع References
٢٩٧	

## مقدمة الكتاب

بسم الله وصلى اللهم وسلم على نبينا محمد وعلى آله وصحبه أجمعين،

هذا الكتاب الرابع من سلسلة تعلم إطار عمل angular بعد الكتب الثلاثة الأولى، والتي انصح بقراءتها قبل قراءة وتعلم هذا الكتاب.

حيث أختص هذا الكتاب بشرح مفاهيم Angular Routing وAngular Modules، وقد حاولت أن أعطي أغلب المفاهيم بدأ بالاساسيات وانتهاءً بمراحل متقدمة.

وتقوم فكرة هذا الكتاب على التعلم من خلال بناء مشروع واحد بحيث نبدأ بمشروع فارغ وكلما نشرح جزء معين نضيفه عملياً إلى هذا المشروع.

وايضاً تستطيع عزيزي المتعلم تحميل المشروع بشكله النهائي مع الشفرات المصدرية source code، على هذا الرابط.

<https://github.com/Faisal-Alfahd/angular-routing>

ولا تنسى ان تكتب الامر npm install بعد تحميل المشروع لتحميل الملفات الضرورية لتشغيل هذا المشروع وإلا فإنه لن يعمل.

وأخيراً أسأل الله ان ينفع به ولا تنسوني احبتي من صالح دعائكم.

# الفصل الأول

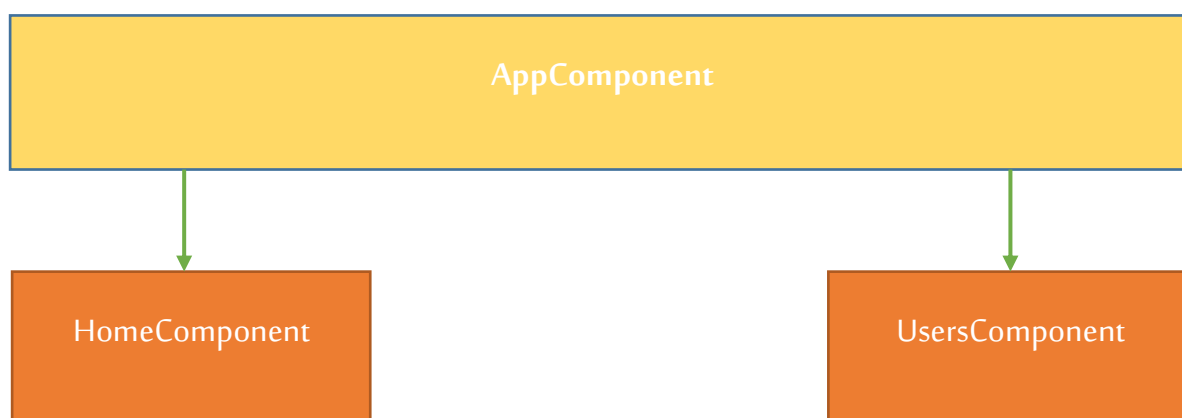
مدخل إلى

Angular Routing

## 1.1. مقدمة:

قبل البدء في هذا الفصل الرجاء انشاء مشروع جديد وليكن اسمه angular-routing مع إضافة routing تلقائياً عن طريق Angular CLI (لمعرفة كيفية انشاء مشروع جديد والتعامل مع أوامر Angular CLI الرجاء مراجعة الكتاب الأول من هذه السلسلة Angular Environment Setup)، بحيث كلما نتعلم ميزة معينة نضيفها إلى هذا المثال، وسوف نبدأ في تعلمنا بفهم وتعمق تدريجي بإذن الله.

ولنرجع للمشروع، أريد منك عزيزي المتعلم ان تنشأ اثنين Components واحد باسم home-component والثاني باسم users-component، وايضاً إذا اردت كيفية انشاء components في مشاريع Angular الرجاء مراجعة نفس الكتاب السابق، وفي حال أردت معرفة كيفية التعامل مع components الرجاء مراجعة الكتاب الثاني من هذه السلسلة angular components and services، ويمكن تمثيلها بالشكل التالي:



حيث AppComponent يمثل component الرئيسي لأي مشروع angular او ما يسمى Root Component وباقي ال components هي التي تم أردت منك انشاءها.

وأريد أن انوه عزيزي المتعلم ان هذا التطبيق الذي سوف نبنيه جزء جزء إلى ان نصل إلى نهاية الكتاب قد لا ينفع تطبيقه واقعياً لأنه يحتاج إلى بعض الأمور الأخرى لضبطه ولكن يكفيننا لشرح واستيعاب أغلب مفاهيم Angular Routing، وهذا هو الهدف الأساسي من هذا الكتاب بشكل عام ومن المشروع الذي سوف نبنيه بشكل خاص.

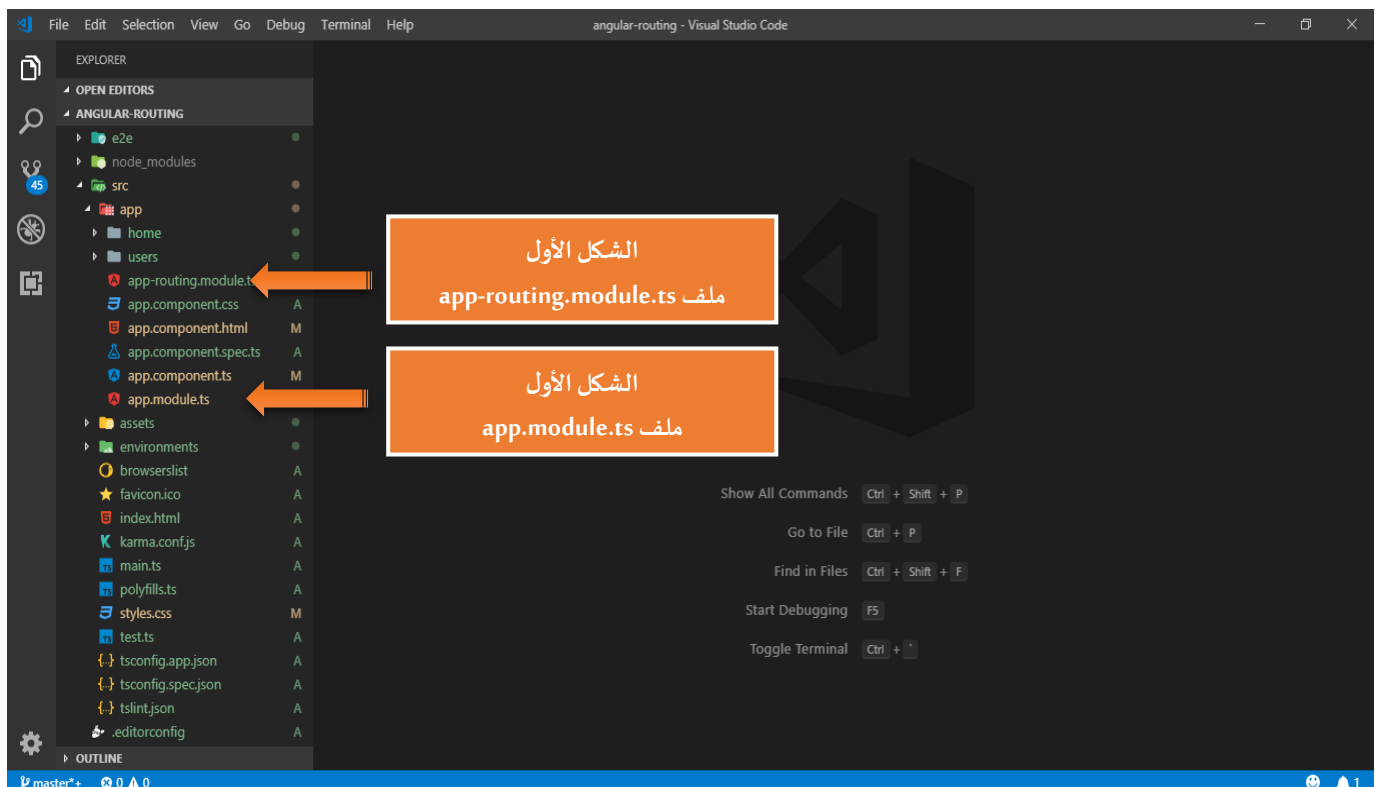
وقبل الانتقال إلى طريقة تهيئة Routing عملياً نريد ان نعرف ما هو بشكل نظري واستطيع ان أقول انه هو الطريقة او الآلية التي تتيحها لنا Angular للانتقال بين أجزاء التطبيق الخاص بك، فكما هو معروف ان Angular هو عبارة SPA مثله مثل أغلب مكتبات وأطر عمل جافاسكربت الأخرى كـ React او Vue... الخ، لذلك لو كان لدينا قائمة جانبية وهذه القائمة تحتوي علة مجموعة من الخيارات او الأزرار كأن يكون لدينا زر للانتقال إلى الصفحة الرئيسية وزر آخر للانتقال إلى صفحة المستخدمين وهكذا بقية الأزرار، ففي هذه الحال لو كُنّا نستخدم الأسلوب القديم فأنا سنستخدم `<a>` وعن طريق الخاصية href التي تحتويها هذا العنصر سوف نضع مسار هذه الصفحات وهذا يؤدي إلى إعادة تحميل كافة الصفحات في حال الانتقال إلى صفحة محددة، اما مع أطر العمل الحديثة مثل Angular فإننا نريد ان يتم تحميل فقط الصفحة التي

طلبها المستخدم، فلو ضغط على زر الصفحة الرئيسية فلن يقوم بتحميل كافة الصفحات وإنما سيتم تحميل فقط الصفحة الرئيسية وهكذا بقية الأزرار، والنظام المخصص الذي يقوم بإدارة وتهيئة هذه الآلية في Angular هو ما يسمى Angular Routing، بحيث يتيح لنا مجموعة من الطرق والأساليب التي سوف نتطرق لها جميعاً بإذن الله لكي نستطيع ربط جميع الصفحات في التطبيق ووضع آليته لانتقال المستخدم بين هذه الصفحات وإيضاً إرسال البيانات في حال انتقال المستخدم من صفحة إلى صفحة أو بصيغة أدخل من component إلى component آخر عن طريق الرابط، وهذه الآلية لتبادل البيانات بين Components لم اتطرق لها في الكتاب الثاني من هذه السلسلة Angular Components And Services لأنها متعلقة بمفاهيم Angular Routing لذلك أرتأيت أن أجعل حديثي عنها في هذا الكتاب، حيث سوف نتعلم بإذن الله كيفية التواصل والاتصال بين Components المختلفة باستخدام Routing.

والآن بعد هذه المقدمة المطولة لننتقل إلى الجزء التالي وهو طريقة تهيئة Angular Routing.

## 2.1. تهيئة Routing:

لو نظرنا إلى ملفات المشروع لوجدنا في المجلد app ملف باسم app-routing.module.ts وهذا الملف انشأه لنا angular CLI في حال كتبنا الأمر --routing عندما قمنا بإنشاء مشروع angular جديد وإيضاً قام بتسجيله في Module الرئيسي أو ما يسمى ملف app.module.ts، كما في الشكلين التاليين:



### ملف app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3.
4. import { AppRoutingModule } from './app-routing.module';
5. import { AppComponent } from './app.component';
6. import { HomeComponent } from './home/home.component';
```

```

7. import { UsersComponent } from './users/users.component';
8.
9. @NgModule({
10.   declarations: [
11.     AppComponent,
12.     HomeComponent,
13.     UsersComponent
14.   ],
15.   imports: [
16.     BrowserModule,
17.     AppRoutingModule
18.   ],
19.   providers: [],
20.   bootstrap: [AppComponent]
21. })
22. export class AppModule { }
23.

```

#### الشكل الثاني

محتويات ملف app.module.ts وهو يمثل module الرئيسي لأي مشروع angular حيث أنه يعتبر root والذي يتم تسجيل فيه أي component أو module أو service أو ... الخ في أي مشروع angular، ونلاحظ أن class الموجود في ملف app-routing.module.ts تم تسجيله هنا بشكل تلقائي

بعد تعريفنا لهذا الملف بشكل عام لنقوم الآن بتعريف مهمة هذا الملف، حيث أن هذا الملف يحتوي على routing التي نريد إنشائها للمشروع، بمعنى أي component نريد أن نُنشئ له routing نضيفه بهذا الملف، الآن لنستعرض محتويات هذا الملف ثم نتطرق لكيفية تهيئة routing، كالتالي:

#### ملف app-routing.module.ts

```

1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3.
4. const routes: Routes = [];
5.
6. @NgModule({
7.   imports: [RouterModule.forRoot(routes)],
8.   exports: [RouterModule]
9. })
10.
11. export class AppRoutingModule { }
12.

```

ما يهمنا في محتويات هذا الملف هو السطر 4 حيث أنه في هذا السطر تم تعريف مصفوفة باسم routes وهي من النوع Routes وهو أحد الأنواع التي اضافتها angular لتعريف routing والتعامل معه، ونكتب بداخل هذه المصفوفة ذات الاسم routes (بما انه متغير يمكن تعديله بالاسم الذي نريده) routes (تشابه أسماء وليس المقصود به المتغير السابق) التي نريد إنشائها لأي component على شكل كائن object حيث انه كل كائن يمثل route واحد ويحتوي على خاصيتين أساسيتين، الخاصية الأولى هي عبارة المسار path الذي يمثل مسار component في الرابط url او بصيغة أخرى نستطيع ان نقول اسم هذا component في url الخاص بتطبيق الخاص بك، والخاصية الثانية هي component ويمثل ال component نفسه الذي نريد من angular أن ينقل مستخدم تطبيق إليه عند الضغط على رابط معين، والخاصية path



لك حرية كتابة الاسم الذي تُريده اما الخاصية الثانية فهي اسم component ومُلزم أن تكتب اسم component كما هو بدون أي تغيير، والمقصود هنا باسم component هو اسم class الخاص بهذا component والموجود في ملف ts.

وبما انه لدينا اثنين components سوف نقوم بإنشاء كائنين الكائن الأول يمثل route لـ component ذو الاسم HomeComponent، والكائن الثاني يمثل route لـ component ذو الاسم UsersComponent، أما لطريقة معرفتك لاسم component الذي تُريد عمل route له، فتستطيع معرفة ذلك من خلال الذهاب إلى component المستهدف ومن ثم الذهاب إلى ملف ts الخاص به وعند فتح هذا الملف فسوف تجد اسم component هو نفسه اسم class لهذا الملف، ولتوضيح أكثر لنأخذ component الخاص بـ Users كمثال، فلنذهب إلى المجلد user ومن ثم إلى الملف user.component.ts وعند فتحنا لمحتويات هذا الملف سوف نجد اسم component هو نفسه اسم class، كالتالي:

#### ملف users.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. @Component({
3.   selector: 'app-users',
4.   templateUrl: './users.component.html',
5.   styleUrls: ['./users.component.css']
6. })
7. export class UsersComponent implements OnInit {
8.   constructor() { }
9.
10.  ngOnInit() {
11.  }
12. }
```

اسم component

الآن بعد معرفتنا لطريقة الحصول على اسم component لنقوم الآن بإنشاء routes الخاص لكل component على شكل كائن في ملف app-routing.module.ts، كالتالي:

#### ملف app-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { HomeComponent } from '../home/home.component';
4. import { UsersComponent } from '../users/users.component';
5.
6. const routes: Routes = [
7.   { path: 'home', component: HomeComponent },
8.   { path: 'users', component: UsersComponent }
9. ];
10.
11. @NgModule({
12.   imports: [RouterModule.forRoot(routes)],
13.   exports: [RouterModule]
14. })
15.
16. export class AppRoutingModule { }
17.
```

نلاحظ في السطر 3 والسطر 4 تم استدعاء components التي نريد أن نعمل لها routing وفي السطر 7 قمنا بإنشاء كائن يُمثل route الأول وقمنا بتهيئته ليكون خاص للـ component ذو الاسم HomeComponent والخاصية path له اسميتها home (كما قلنا سابقاً لك حرية اختيار الاسم الذي تريده ولكن يُفضل أن يكون معبر عما يُشير إليه وايضاً يجب أن تلتزم به عندما نقوم بعملية الربط والتي سوف نتكلم عنها في الأجزاء القادمة من هذا الكتاب بإذن الله)، وفي السطر 8 قمنا بنفس الأمر ولكن للـ component ذو الاسم UsersComponent.

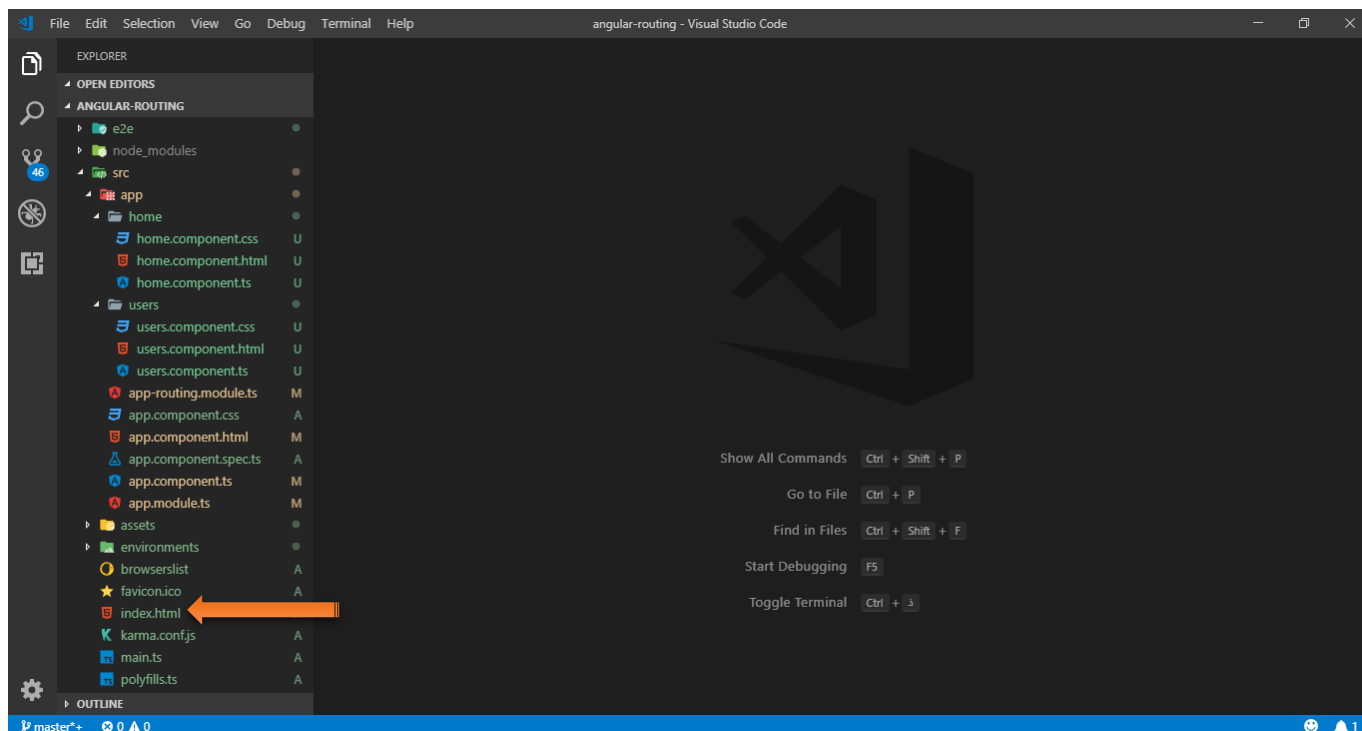
هناك نقطة مهمة لم اتطرق لها وهو ما هو موجود في السطر 12 حيث انه في هذا السطر قام angular بشكل تلقائي بتمرير المصفوفة routes لدالة forRoot وهذا السطر مهم جداً وبدونه لن يعمل routing بشكل صحيح ومهمتها تحميل جميع Modules وما تحتويه من routes والـ components المرتبطة بها عند بداية تشغيل التطبيق، وهذا بعكس الدالة الأخرى ذات الاسم forChild التي لا تقوم بتحميل كل شيء في بداية تشغيل الموقع، وسوف نتطرق إليها في الأجزاء القادمة من هذا الكتاب بإذن الله عند وصولنا لأجزاء الخاصة Modules و Lazy Loading.

ملاحظة: تستطيع تهيئة routing في app.module.ts وهو Module الرئيسي للمشروع ولكن يُفضل أن يتم تقسيم routing في Module خاص به ومن ثم استدعاء هذا Module في Module الرئيسي للمشروع.

وبذلك نكون أنهينا هذا الجزء حيث أن تهيئة routing هي ما تم كتابته في السطر 7 و 8 فقط اما باقي الشروح فهي عبارة شرح وتوضيح للمفاهيم لكي يتكون لدى المتعلم فهم أعمق واستيعاب شامل لكل جزء نتطرق إليه في هذا الكتاب.

### 3.1. توضيح وفهم Angular Base href:

لنذهب إلى ملف index.html ونستعرض محتوياته، كالتالي:



```

1. <!doctype html>
2. <html lang="en">
3. <head>
4.   <meta charset="utf-8">
5.   <title>AngularRouting</title>
6.   <base href="/">
7.
8.   <meta name="viewport" content="width=device-width, initial-scale=1">
9.   <link rel="icon" type="image/x-icon" href="favicon.ico">
10.</head>
11.<body>
12.  <app-root></app-root>
13.</body>
14.</html>

```

هذا الملف هو الملف الرئيسي لواجهة الموقع ولو لاحظنا في السطر 12 موجود فيه التاغ او العنصر الخاص بالcomponent الرئيسي للموقع AppComponent، بمعنى أن محتويات هذا component والمقصود ما يحتويه هذا component من اكواد HTML سوف يتم عرضها هنا لأن هذا الملف هو root أو الأصل الذي سوف يقرأه السيرفر لاستعراض محتويات الموقع الخاص بك او تطبيقك (لمعرفة أكثر الرجاء مراجعة الكتاب الأول والثاني من هذه السلسلة).

وحقيقة ما يهمنا هنا هو ما هو موجود في السطر 6 حيث أن هذا السطر موجود فيه عنصر باسم <base> وهذا العنصر يحتوي على خاصية باسم href وقيمة هذه الخاصية "/", وهذا معناه أن جميع صفحات الموقع سوف يتم قراءتها بعد اسم الدومين الخاص بك بعد رفع موقعك إلى السيرفر، فمثلاً لو رفعنا الموقع (التطبيق) إلى سيرفر معين على الانترنت واختارنا له دومين باسم faisal بحيث يكون بهذا الشكل www.faisal.com فإن صفحات الموقع سوف يتم قراءتها بالطريقة التالية: www.faisal.com/home - www.faisal.com/users حيث أن صفحة home تشير إلى path الخاص بالcomponent المسمى HomeComponent الذي ينتقل إليه الموقع لعرضه محتوياته، وصفحة users تشير إلى path الخاص بالcomponent المسمى UsersComponent الذي ينتقل إليه الموقع لعرض محتوياته، وهو عبارة عن routing الذي قمنا بتهيئته في الجزء السابق.

وقيمة الخاصية href في هذه الحالة هو الوضع الافتراضي ولكن ماذا لو استدعى الامر لأي سبب من الأسباب ان تكون صفحات الموقع تحت مجلد فرعي تحت اسم الدومين، كأن تكون مثلاً www.faisal.com/employee والصفحات تكون تحت المجلد employee كأن تكون www.faisal.com/employee/home أو www.faisal.com/employee/users، في هذه الحالة الموق لن يعمل وسوف تحدث فيه مشاكل ولحل هذا الأمر لديك طريقتين، كالتالي:

✓ الأولى أن تقوم بتغيير قيمة الخاصية href لتصبح "/employee" مع ملاحظة يجب كتابة هذا المجلد في عملية الربط التي سوف نأخذها إن شاء الله في الأجزاء القادمة لتصبح بالشكل مثلاً التالي "/employee/home".

✓ أما في حال أن الموقع كان تم الانتهاء منه ولكن عند عملية الرفع إلى السيرفر احتجت إلى أن يكون الموقع تحت مجلد، فلا تقلق فبدلاً من التعديل على القيمة الافتراضية للخاصية href ومن ثم التعديل في كل عملية ربط او توجيه تمت في الموقع تستطيع عند عملية بناء الموقع كتابة الأمر التالي في terminal:

```
ng build -prod --base-href /employee
```

عند كتابة هذا الأمر تلقائياً angular سوف يقوم بتعديل الموقع وإضافة الاسم employee لكل كود برمجي في الموقع الخاص بك ثم يقوم ببناؤه، كما تستطيع تجريب الموقع على جهازك أيضاً بإضافته إلى الأمر ng serve، ليصبح الأمر كالتالي:

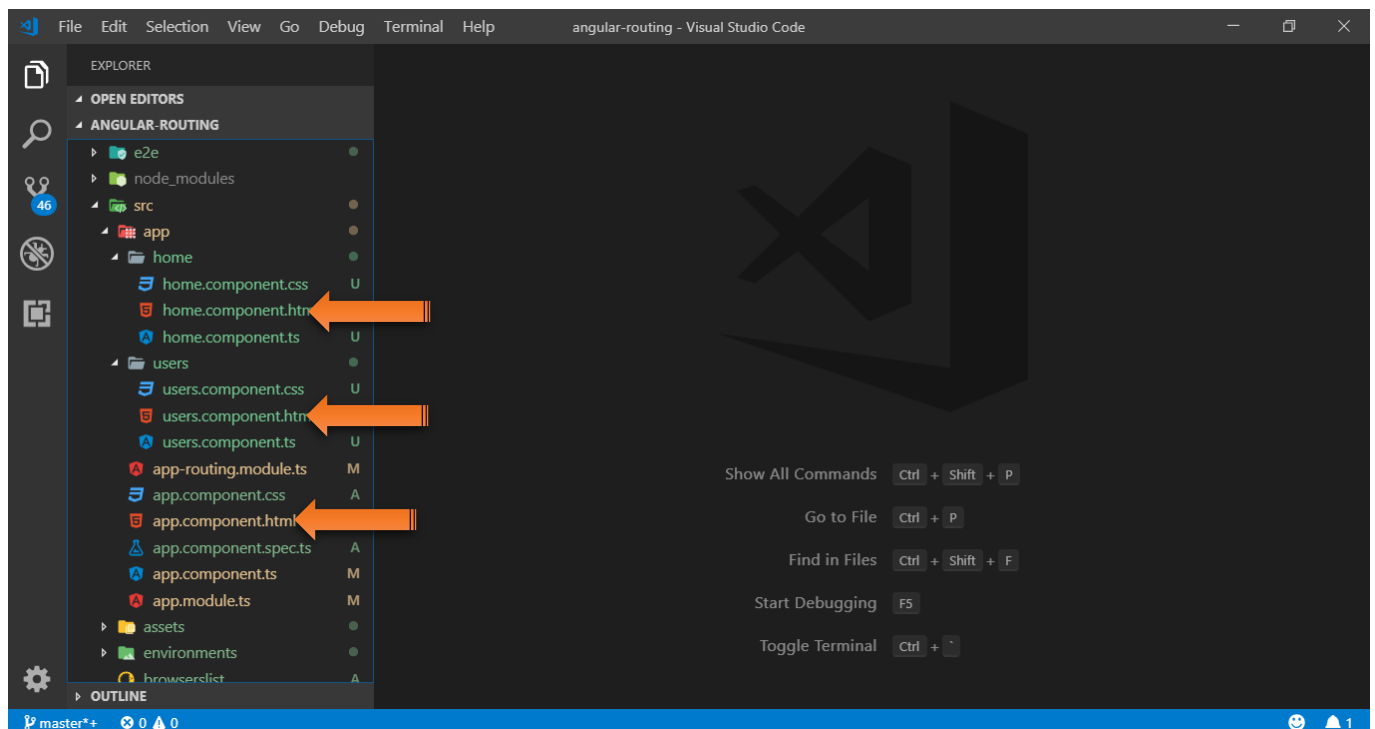
```
ng serve -o --base-href /employee
```

وبهذه الطريقة نستطيع أن نتعامل مع base href سواء بالطريقة الأولى أو الطريقة الثانية بحسب احتياجنا. (ولمعرفة أكثر عن طريقة التعامل مع Angular CLI و terminal الرجاء مراجعة الكتاب الأول من هذه السلسلة)

وبذلك نكون أنهينا هذا الجزء وننتقل في الجزء التالي لكيفية عمل Wildcard Routes والمفاهيم التي يتضمنها، قبل انتقالنا للجزء الي يليه لنتكلم عن كيفية عمل ربط بين Routes و Template لل component .

## 4.1 Wildcard Routes:

مع أننا لم نقوم بعملية الربط إلى الآن إلى أننا نستطيع أن نشاهد نتيجة تهيئة Routing على المتصفح من خلال كتابة الرابط (path) الخاص بكل component على المتصفح مباشرة، ولكن قبل القيام بذلك لنقوم بكتابة بعض أكواد HTML المؤقتة في ملفات HTML لكل component، كالتالي:



✓ في ملف app.component.html وهو component الرئيسي في مشاريع angular ويحتوي بداخله باقي components الأخرى.

لنقوم بحذف أكواد html الافتراضية ونبقي فقط على العنصر (التاغ) <router-outlet></router-outlet> وسوف نتكلم عن هذا العنصر في الأجزاء القادمة ولكن الآن لنبقه كما هو ونستبدل اكواد html الافتراضية ببعض اكواد html المؤقتة لنقوم بالتجربة فقط، كالتالي:

```
1. <h1 style="text-align: center">App Component</h1>
2. <hr />
3. <br />
4.
5. <router-outlet></router-outlet>
```

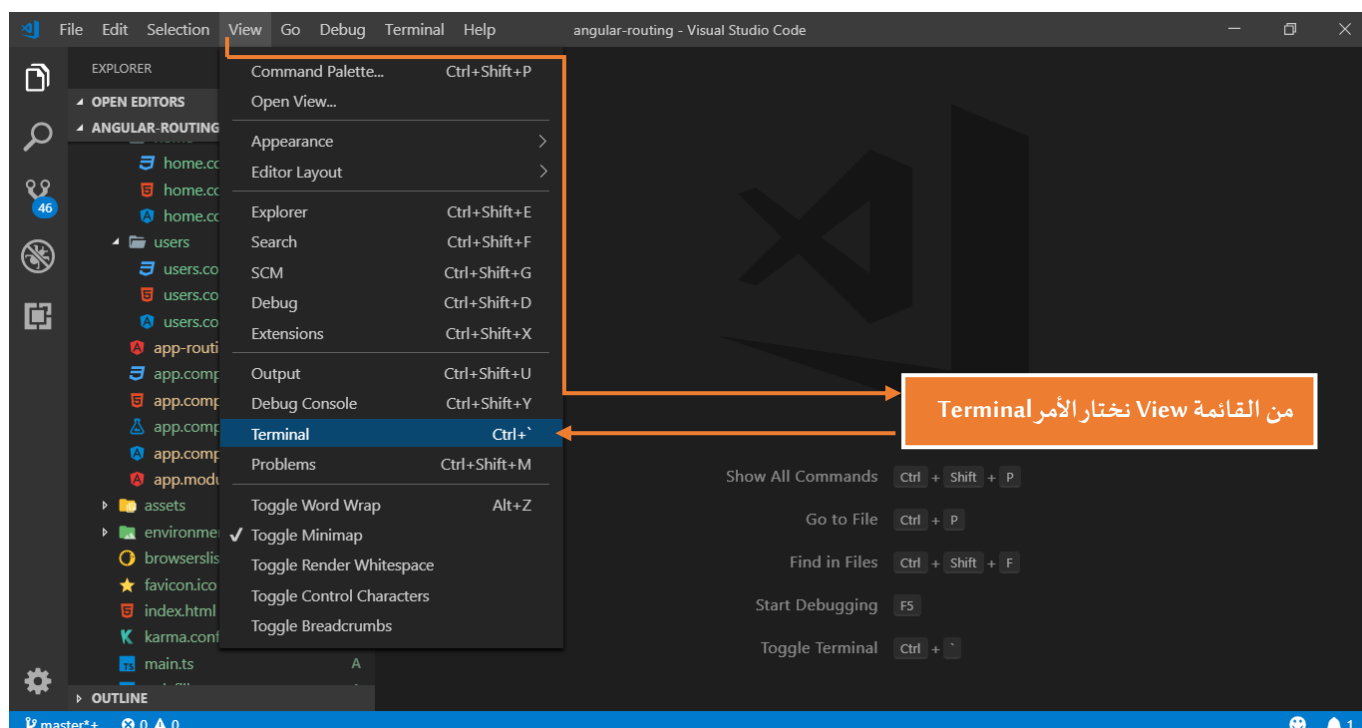
✓ في ملف home.component.html لنقوم باستبدال اكواد html الافتراضية بالاكود التالية:

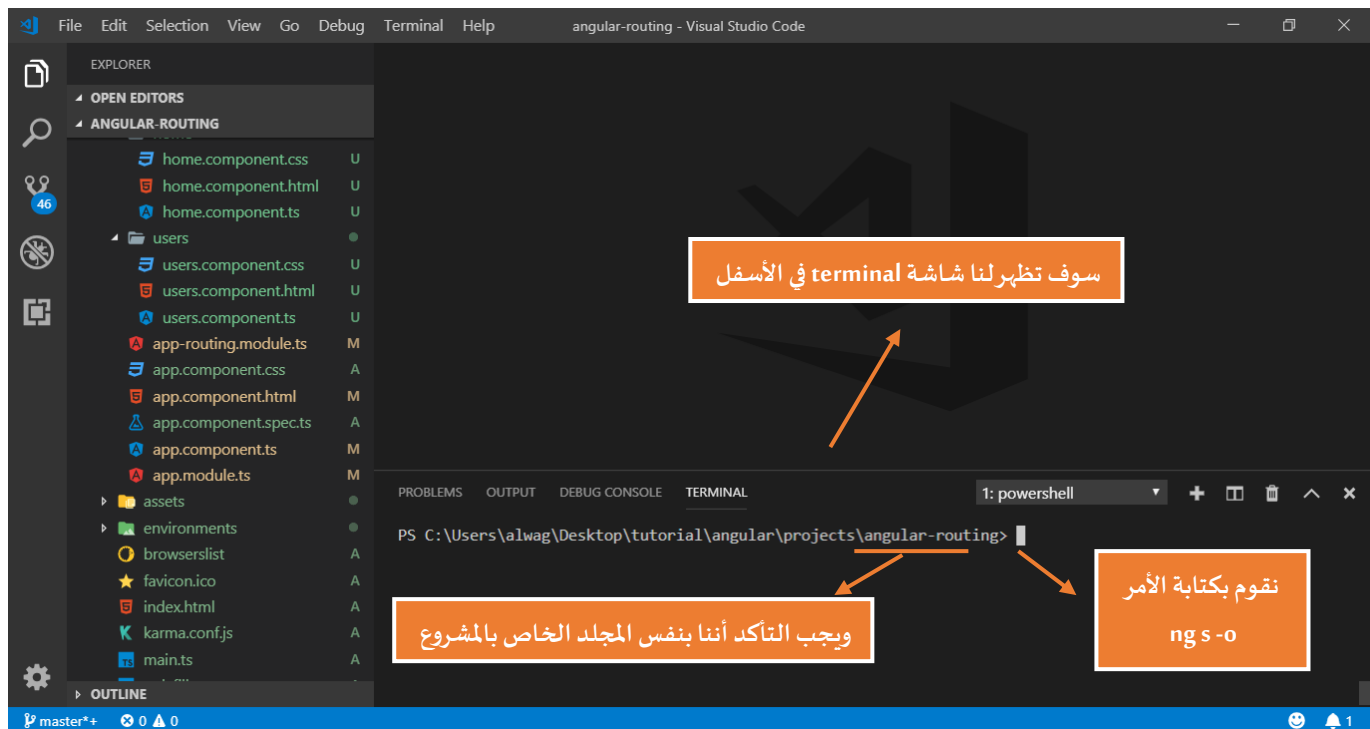
```
1. <br />
2. <h2 style="text-align: center">Home Component</h2>
```

✓ في ملف home.component.html لنقوم باستبدال اكواد html الافتراضية بالاكود التالية:

```
1. <br />
2. <h2 style="text-align: center">Users Component</h2>
```

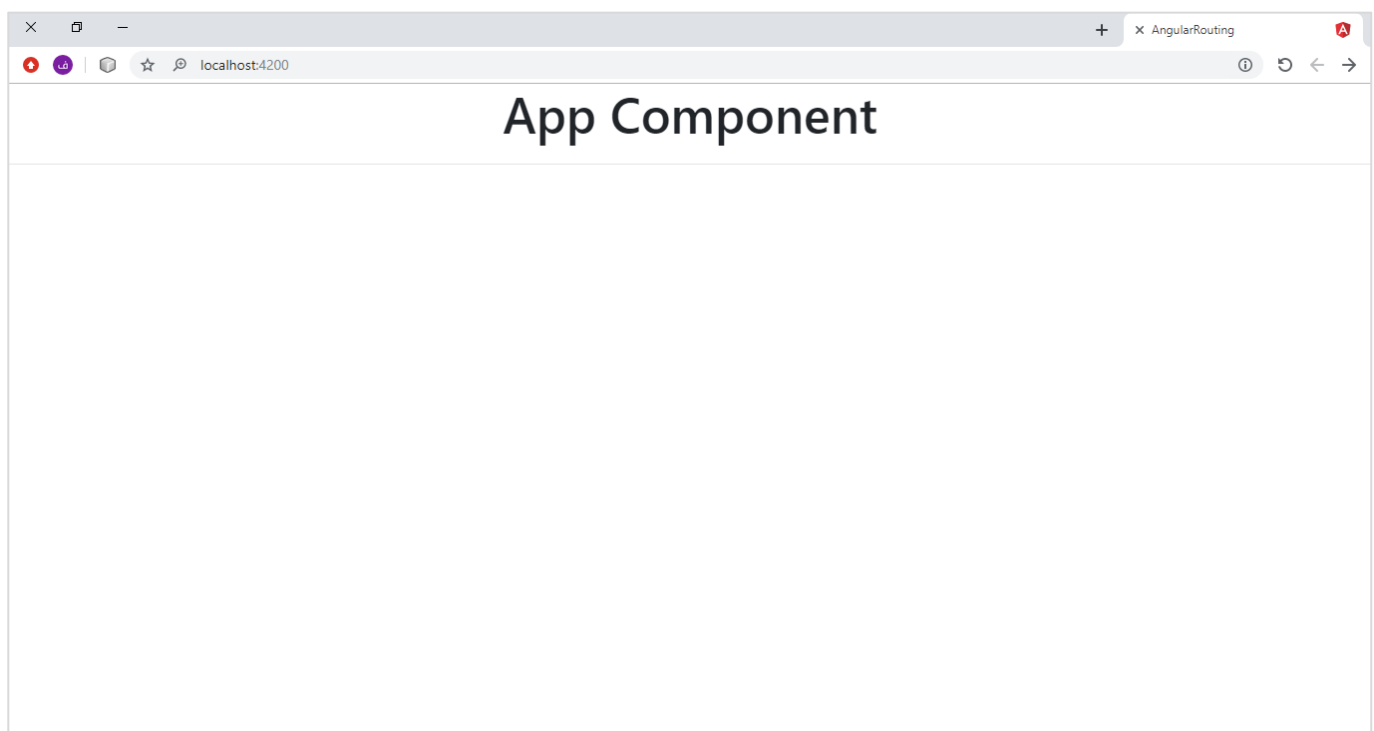
بعد كتابتنا للأكواد السابقة لنقوم بتشغيل المشروع على المتصفح لنرى النتيجة، وذلك من خلال فتح terminal الموجود في محرر الأكواد VSCode، كالتالي:





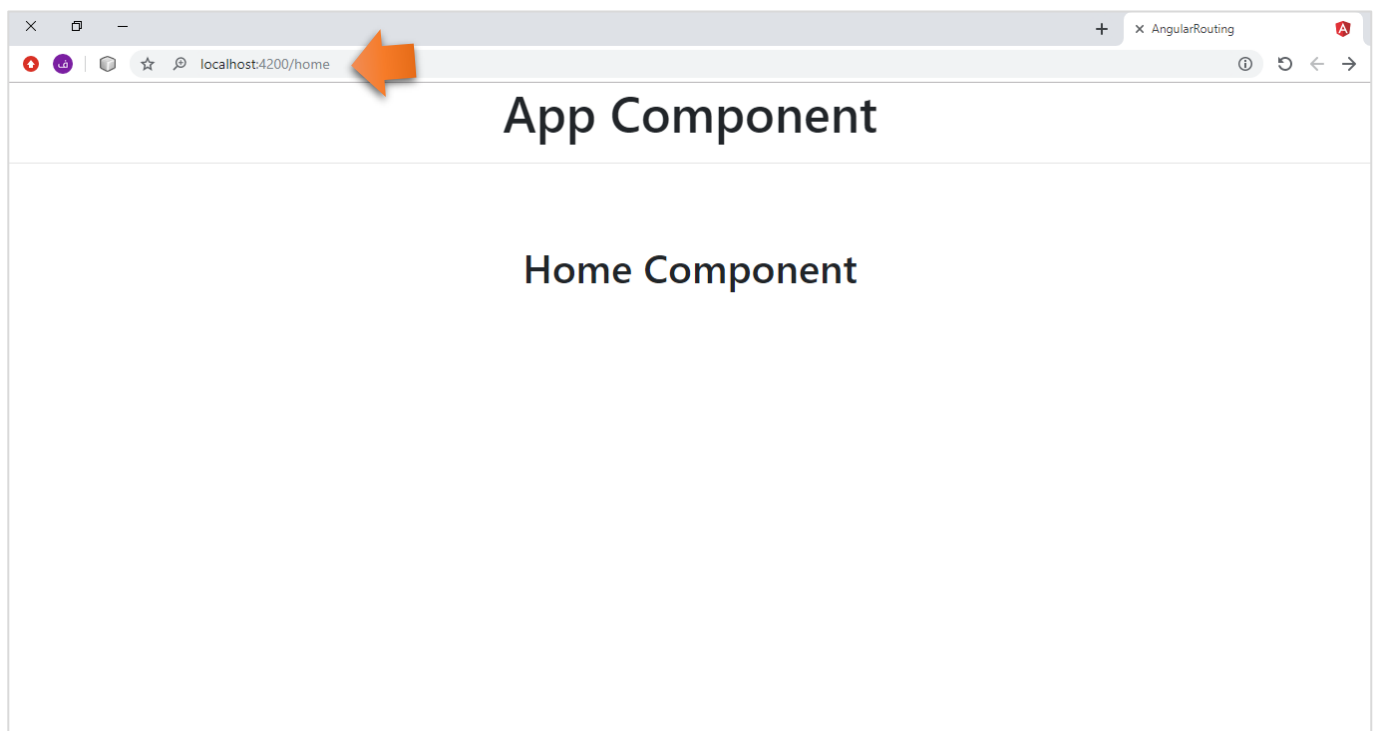
بعد كتابة الأمر السابق نقوم بالضغط على زر Enter في لوحة المفاتيح، وننتظر إلى أن يتم فتح المشروع (قد يستغرق ذلك بعض الوقت) على المتصفح الافتراضي لديك (المتصفح الافتراضي لدي هو google chrome).

بعد الانتهاء سوف يُفتح متصفح google chrome، ويتم تشغيل المشروع داخله، كما في الشكل التالي:

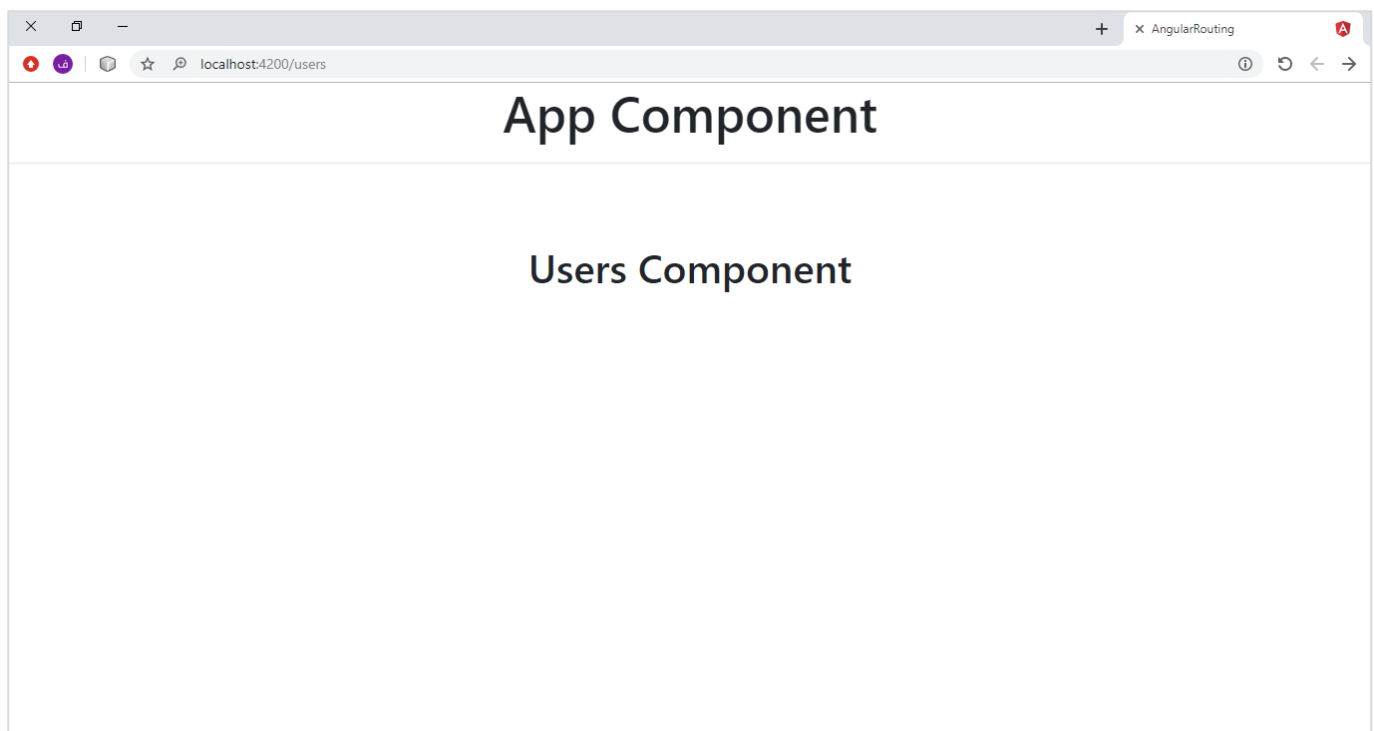


نلاحظ أن الذي ظهر لنا هو محتويات AppComponent وهو component الرئيسي كما ذكرنا سابقاً، والعنوان الموجود في رابط المتصفح هو localhost:4200 وهو العنوان الافتراضي في حال تشغيلك للمشروع على جهازك المحلي للتجربة وإثناء التصميم وسوف يتغير إلى اسم الدومين الخاص بك عند رفع المشروع على أحد السيرفرات كأن يكون مثلاً [www.faisal.com](http://www.faisal.com).

الآن لنقوم بكتابة path الخاص بالcomponent المسمى HomeComponent والذي كان اسمه عندما قمنا بتهيئته home بحيث يصبح الرابط كالتالي: localhost:4200/home، ثم نضغط Enter ولنرى ماذا يحدث:

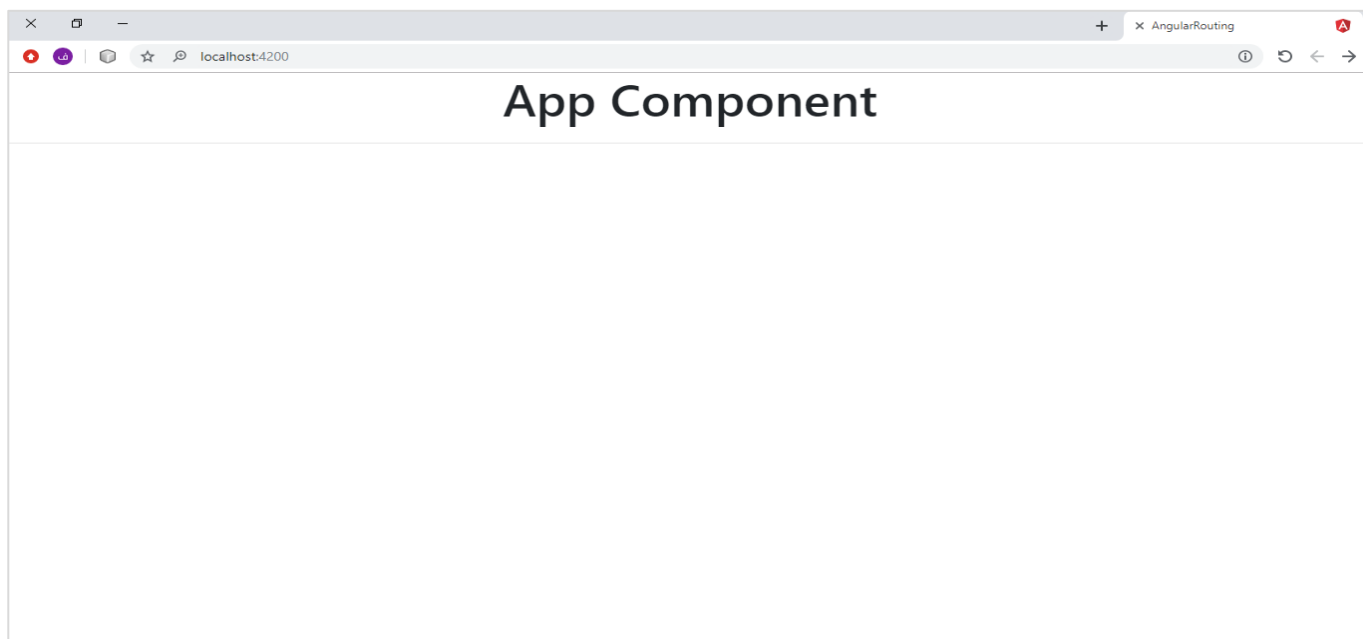


نلاحظ انه عرض لنا محتويات HomeComponent مع ثبات component الرئيسي AppComponent لأننا قلنا انه يعتبر بمثابة الحاوية التي تحوي بداخلها بقية components الأخرى، وب نفس الطريقة لنقوم بكتابة path المسمى users ولنرى ماذا يحدث:

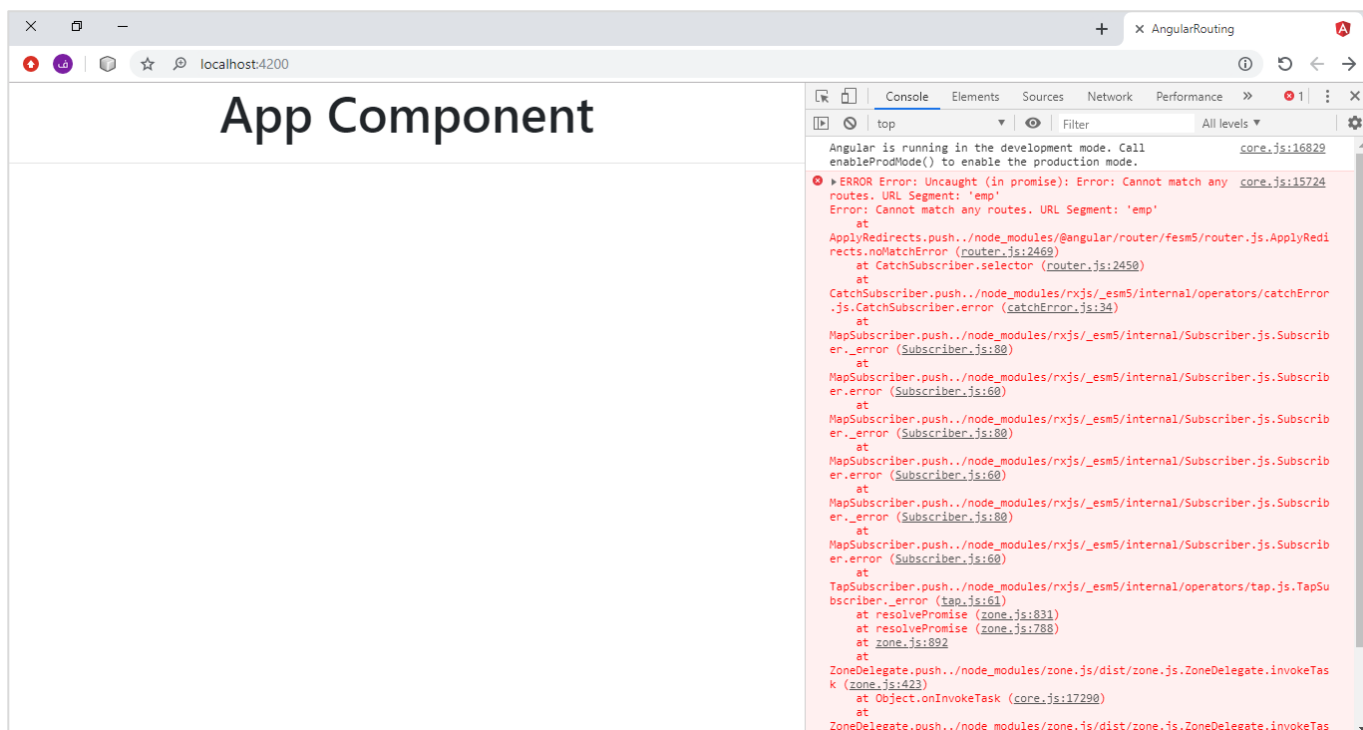


نلاحظ أنه عرض لنا محتويات UsersComponent، وبذلك نكون عملنا أول Routing لنا وهو التنقل بين components المختلفة، وايضاً شاهدنا النتيجة على المتصفح، ولكن هنالك بعض المشاكل أولها أنه في بداية تشغيل المشروع ظهر لنا

فقط AppComponent ونحن نريده أن يظهر محتويات HomeComponent بشكل افتراضي عند بداية تشغيل المشروع، والمشكلة الثانية ماذا لو قام المستخدم بكتابة أي path غير الذي قمنا بتهيئته في Routing كأن يكتب مثلاً localhost:4200/emp، لنقوم بكتابته في عنوان الرابط ونرى ماذا سوف يحدث:



نلاحظ انه لم يُظهر شيء ولكن لنفتح console الخاص بالمتصفح عن طريق الضغط بزر الفأرة الأيمن في أي مكان فارغ داخل المتصفح ثم اختيار فحص او inspect ومن ثم اختيار التبويب console، كالتالي:

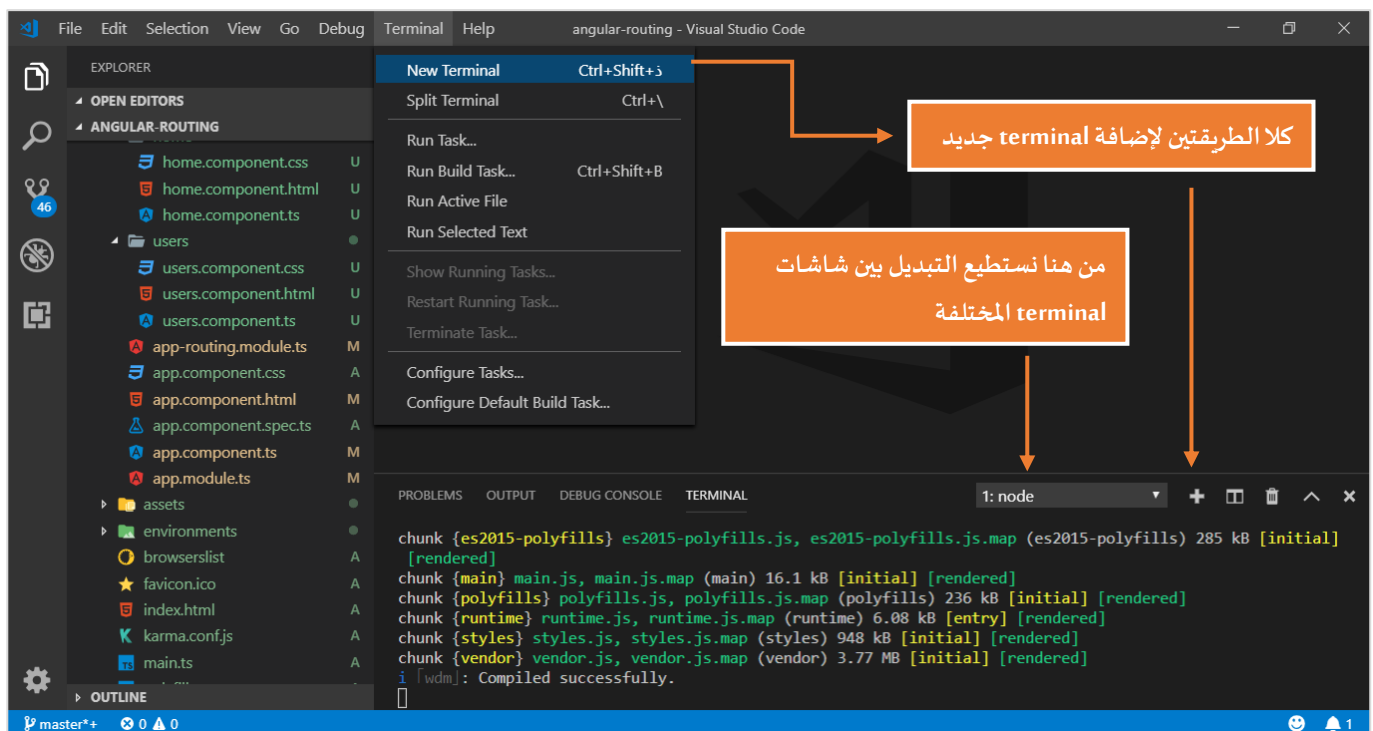


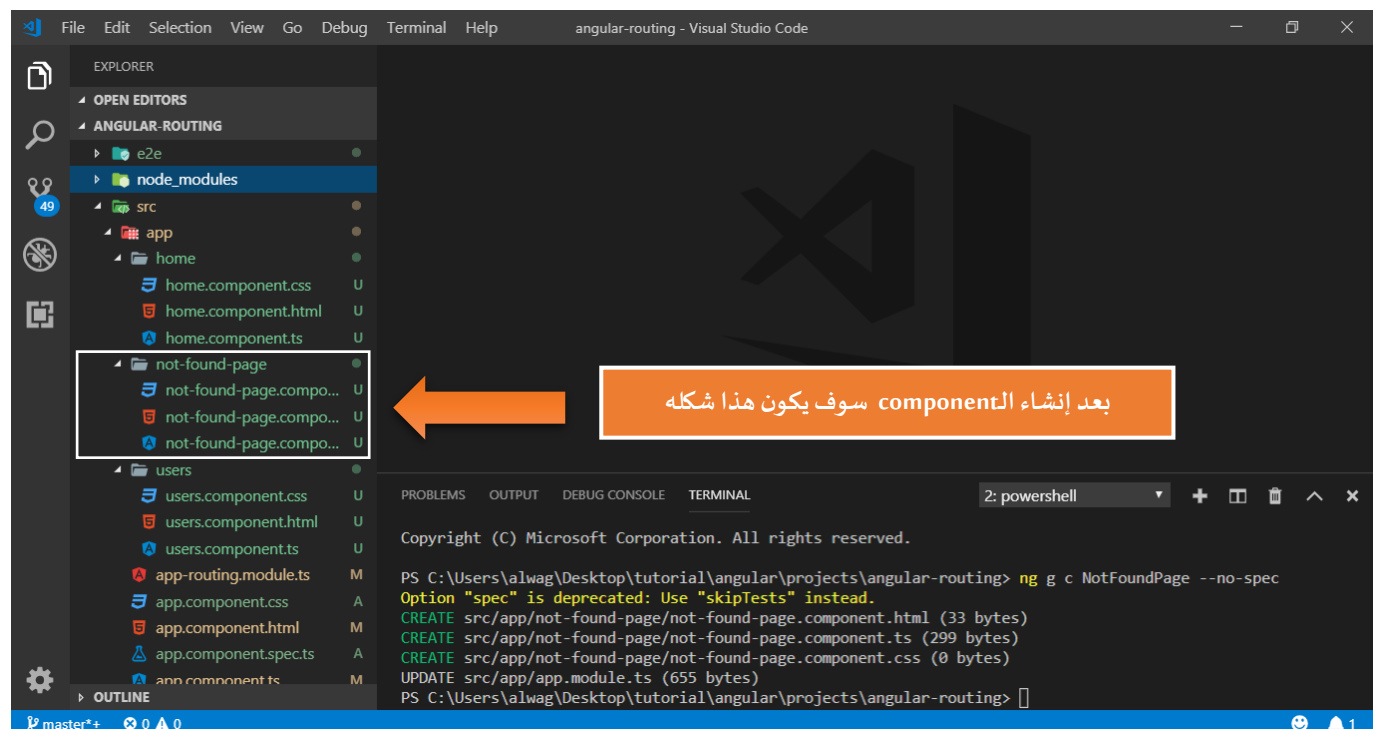
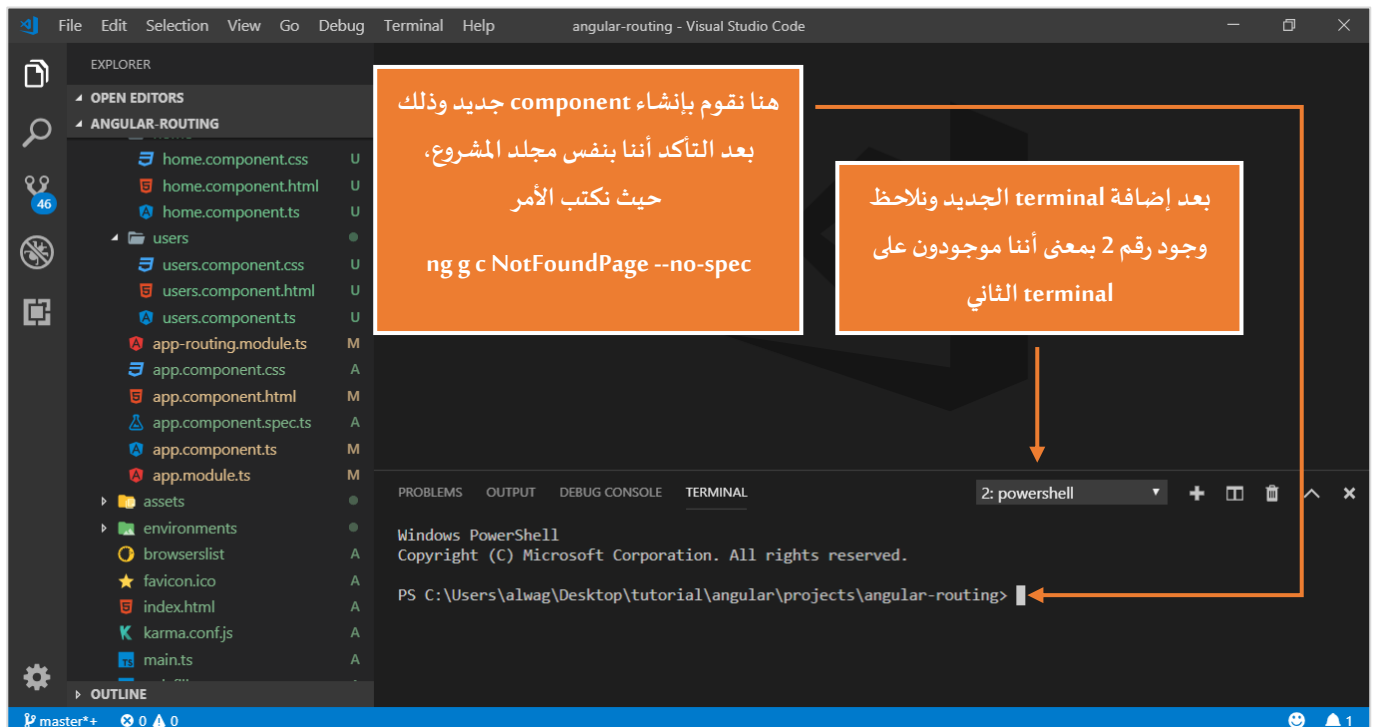
نلاحظ تم إظهار رسالة خطأ في console مختصرها أنه لا يوجد component له نفس path المدخل في عنوان الرابط، وحل هذه المشكلة والمشكلة السابقة الذكر يكمن في ما يسمى Wildcard Routes وهي عبارة عن آلية أو طريقة تسمح لنا بالتحكم بإدخالات المستخدم بحيث نقول له إذا المستخدم قام بإدخال أو كتابة أي path غير path الذي تمت تهيئته سابقاً قم



بتحويله إلى component معين أو أجعل component الافتراضي في بداية تشغيل الموقع أو المشروع هو component ذو الاسم كذا، والذي نريده في مشروعنا هذا أن يكون HomeComponent هو الذي يتم عرضه في بداية تشغيل المشروع أو في حال عدم وجود أي path فقط localhost:4200 في عنوان الرابط، والأمر الآخر الذي نريده هو في حال قيام المستخدم بكتابة أي path في عنوان الرابط نريد أن ننقله إلى صفحة جديدة موجودة فيها مثلاً الصفحة غير موجودة، ويمكن القيام بذلك عن طريق الخطوات التالية:

أولاً/ نقوم أولاً بإنشاء component جديد وليكن اسمه NotFoundPage، ويتم ذلك عن طريق الذهاب إلى terminal وإنشاء terminal جديد ونبقي على terminal السابق كما هو لأن terminal السابق مرتبط بتشغيل المشروع على المتصفح ومتصل معه، ولا نريد فصل الاتصال عنه لأننا في حال فصل الاتصال سوف نضطر إلى كتابة الأمر ng serve -o مرة أخرى والانتظار إلى أن يفتح لنا المتصفح وتشغيل المشروع ولكن في حال إبقائنا على الاتصال كما هو فكل تعديل نقوم به على المشروع عندما نقوم بحفظه فبشكل تلقائي سوف يتم تحديث المتصفح وإضافة التعديلات الجديدة، لذلك سوف ننشأ terminal جديد وذلك بالذهاب إلى قائمة terminal ومن ثم الضغط على New Terminal أو بنفس شاشة terminal السابق نضغط على علامة الزائد وسوف يُضاف terminal جديد، كما في الشكلين التاليين:





بعد إنشائنا للـ component السابق نذهب إلى صفحة HTML الخاصة بهذا component ونغير الكود الافتراضي بهذا الكود، كالتالي:

1. `<br />`
2. `<h1 style="text-align: center">Page Not Found</h1>`

وبذلك نكون أننا من إنشاء NotFoundPageComponent وفي النقطة الثانية سوف ننشأ Wildcard Routes.

ثانياً/ لإنشاء Wildcard Routes نذهب إلى ملف app-routing.module.ts ونقوم بالتعديل على تهيئة routing الذي انشأناه سابقاً، حيث سوف نضيف اثنين routes جديدة الأولى لكي يقوم بإظهار HomeComponent بشكل افتراضي عند

بداية تشغيل المشروع والثاني في حال وجود path خاطئ نريد أن نظهر `NotFoundPageComponent`، لذلك سوف أضيف هذه الأكواد ثم أقوم بشرحها، كالتالي:

ملف `app-routing.module.ts`

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { HomeComponent } from '../home/home.component';
4. import { UsersComponent } from '../users/users.component';
5. import { NotFoundPageComponent } from '../not-found-page/not-found-page.component';
6.
7. const routes: Routes = [
8.   { path: 'home', component: HomeComponent },
9.   { path: 'users', component: UsersComponent },
10.  { path: '', redirectTo: 'home', pathMatch: 'full' },
11.  { path: '**', component: NotFoundPageComponent }
12.];
13.
14. @NgModule({
15.   imports: [RouterModule.forRoot(routes)],
16.   exports: [RouterModule]
17. })
18.
19. export class AppRoutingModule { }
```

نلاحظ في السطر 10 والسطر 11 تمت إضافة إثنين `routes` جديدة حيث أن `route` الأول الموجود في السطر 10 يختص بالتأكد من إظهار صفحة `HomeComponent` في بداية تشغيل الموقع وذلك عن طريق توجيهه إلى صفحة `home` وهي قيمة `path` لـ `HomeComponent`، ولنفصل بالخصائص التي يحتويها هذا الكائن كالتالي:

**Path:** قيمته فارغة لأن بداية تشغيل الموقع يكون فقط اسم الدومين وفي حالتنا هنا يكون بالشكل التالي `localhost:4200` نلاحظ بعد اسم الدومين لا يوجد أي اسم لأي صفحة، لذلك نجعل قيمة `path` فارغة.

**redirectTo:** وهنا نقوم بتوجيه الموقع إلى صفحة معينة، بمعنى إذا كانت قيمة `path` فارغة قم بتوجيه الموقع إلى `path` ذو القيمة `home` وهذا `path` هو الخاص بعرض محتويات `HomeComponent`.

**pathMatch:** وهذه الخاصية مرتبطة في الغالب مع الخاصية السابقة `redirectTo` ويجب كتابتها معه أما في حال عدم وجود هذه الخاصية فلا نحتاج إلى كتابة `pathMatch` لأن `angular` سوف يكتبها بالنيابة عنا ويعطيها قيمة معينة، وتأخذ قيمتين هي `prefix` والقيمة الثانية `full`، أما الفرق بين هذين القيمتين هي كالتالي:

**prefix:** وهي القيمة الافتراضية لهذه الخاصية ويضيفها `angular` لها في حال عدم كتابتها، ومهمتها تقسيم الرابط تبعاً `"/"` بمعنى لو كان لدينا مثلاً صفحة فرعية باسم `profile` داخل الصفحة `users` بحيث تكون قيمة `path` تساوي `"users/profile"` فسوف يكون الرابط في المتصفح بالشكل التالي `localhost:4200/users/profile` وبالتالي سوف تقوم هذه الخاصية إذا كانت قيمتها `prefix` بتقسيم الرابط السابق إلى مجموعة أقسام فتبدأ باسم الدومين ومن ثم تنتقل إلى

الصفحة users وتبحث عنه في تهيئة routes أن وجدته تنتقل بعدها إلى الصفحة الفرعية profile وعندما تجدها ولا يوجد صفحة أخرى بعدها تقوم بعرض هذا component والذي قيمة path الخاصة به تساوي profile، وحقيقة هذا هو السلوك الطبيعي لأي routing حيث يقوم بقراءة الرابط وتقسيمه بناءً على "/", ولكن هنا لا تعمل معنا وسوف تُسبب لنا بعض المشاكل لأنه وبكل بساطة من أسباب تهيئتنا لهذا route هو أن الموقع في بداية تشغيله تكون قيمته فارغة localhost:4200 وقيمة path تساوي "" لذلك سوف يقوم في البداية بتقسيم الرابط وقيمته في البداية فارغة لذلك سوف ينقلنا لصفحة HomeComponent إلى هنا الوضع جيد ولكن عندما نقوم بعملية الربط (وهو ما سوف نشرحه في الجزء القادم بإذن الله) فلن يعمل لأنه في كل مرة يضغط المستخدم على أحد الروابط للانتقال إلى أحد صفحات موقع سوف يعامل الرابط في بدايته على أنه فارغ وينقلنا إلى صفحة HomeComponent وليس هذا الذي نريده لذلك لابد من اعتماد القيمة full لهذه الخاصية في هذه الحالة.

**full:** هذه القيمة تجعل الخاصية pathMatch تعامل الخاصية كقطعة نصية كاملة بدون أي تقسيم، فلو طبقناها على المثال السابق فلن تقسم الرابط وانما سوف تبحث عن component تكون قيمة path الخاصة به تساوي "users/profile" وبالتالي سوف يسبب خطأ لأنه لا يوجد صفحة لها نفس هذا الاسم، لأنه في الحقيقة path السابق يُشير إلى صفحتين متداخلتين الصفحة الأولى users والثانية profile وهذا سلوك غير طبيعي ولا نريده في العادة ولكنه هنا يُناسبنا لأننا نقول له لا تنتقل إلى HomeComponent إلا في حالة أن كامل الرابط بعد اسم الدومين فارغ، وبشكل عام نستخدم هذه القيمة لهذه الخاصية دائماً في حالة أن path كانت قيمته فارغة، وفي الحقيقة لم أرى أي استخدام آخر لها إلا في هذه الحالة وحتى عندما نتعامل مع children في أحد الأجزاء القادمة فإننا سوف نضع قيمة path فارغة ولم نستخدم هذه الخاصية، لذلك نستطيع أن نضع قاعدة عامة وهو في حال أردنا أن نعرض محتويات component معين في بداية تشغيل الموقع نستخدم الخاصية pathMatch ونجعل قيمتها full.

أما السطر 11 فمهمة هذا route هو التأكد من أن جميع العناوين في الرابط تمت تهيئتها في routing وفي حال وجود أي عنوان لم يتم تهيئته ينقلنا أو يعرض صفحة معينة، ويمكن القيام بذلك بكل سهولة عن طريق جعل قيمة path الموجودة في هذا route تساوي "\*" والangular سوف يقوم بباقي العمل عنا، وهنا طريقتين الأولى ننشأ component خاص في هذه الحالة فقط ونعرض component بشكل مباشر كما عملنا هنا في NotFoundPageComponent أو نستخدم الخاصية redirectTo إذا مثلاً في حال كتابة رابط خطأ ينقلنا إلى HomeComponent.

**ملاحظة: يجب كتابة هذا route في آخر الroutes الموجودة أو سوف يسبب بعض الأخطاء ولن يعمل بشكل صحيح**

وبذلك نكون أنهينا Wildcard Routes وشرحناها بشكل مفصل، وبذلك نكون مهئين للانتقال إلى الجزء الآخر وهو كيفية عمل الربط بين routing وملفات template أو التي تُسمى ملفات HTML للcomponents.

## 5.1. الربط في Angular Routing:

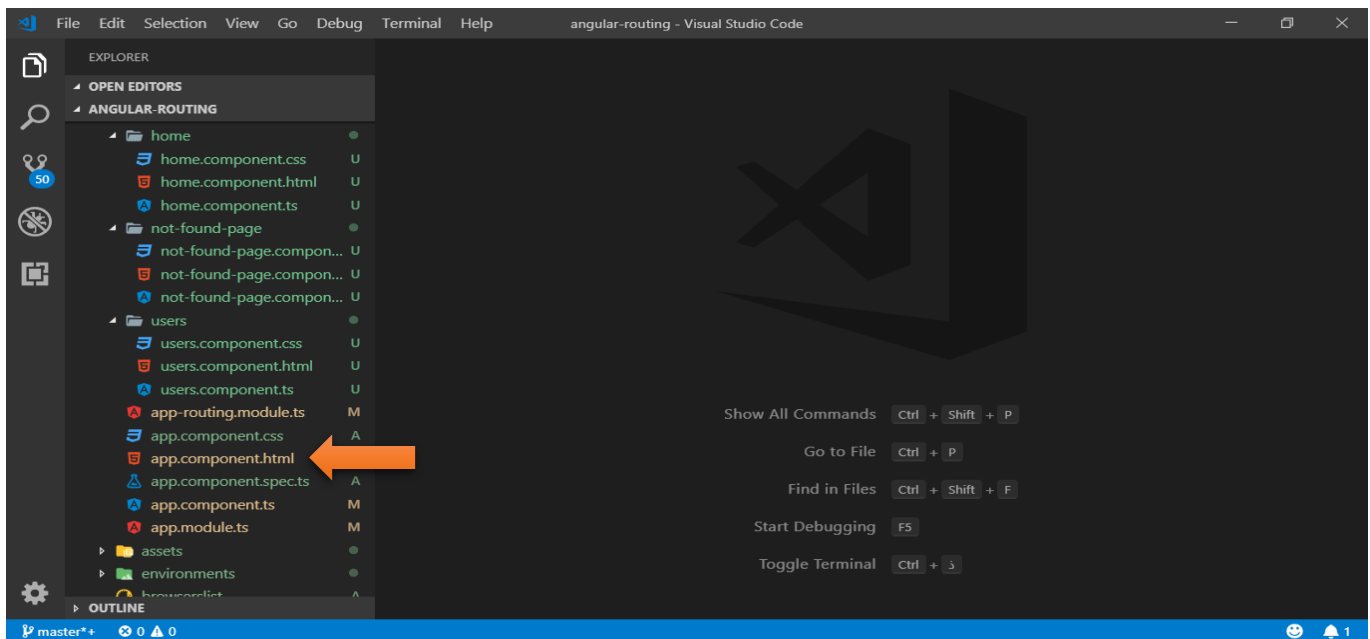
من غير الطبيعي أن نجبر المستخدم على التنقل بين components المختلفة أو بين صفحات الموقع عن طريق كتابة path الخاص بكل component في الرابط، وإنما الطبيعي أن يكون هنالك روابط في Template أو صفحات HTML وعن طريقها يتم نقل المستخدم إلى صفحة معينة عندما يتم الضغط على أحد هذه الروابط، وهذا ما سوف نتكلم عنه في هذا الجزء.

والمقصود بالربط هو الطريقة أو الآلية التي تتيح لنا ربط التهيئة السابقة بملف template أو ملف class للComponent معين، بمعنى هي الآلية التي تتيح لنا ربط زر معين في component معين بأحد أنواع التهيئة السابقة بحيث إذا ضغط المستخدم على هذا الزر يتم نقله إلى Route الذي تم ربطه فيه.

وهناك نوعين من الربط، الأول وهو الربط البرمجي والمقصود فيه الربط عن طريق ملف class للComponent والربط عن طريق Template والمقصود فيه إجراء الربط عن طريق ملف template أو ما يسمى ملف HTML للComponent.

### 1.5.1. الربط عن طريق ملف Template:

لو راجعنا بنية التطبيق لوجدنا أن AppComponent هو الثابت وباقي components الأخرى هي التي تتغير لذلك سوف نضع الروابط في ملف HTML أو ما يسمى template لهذه component، كالتالي:



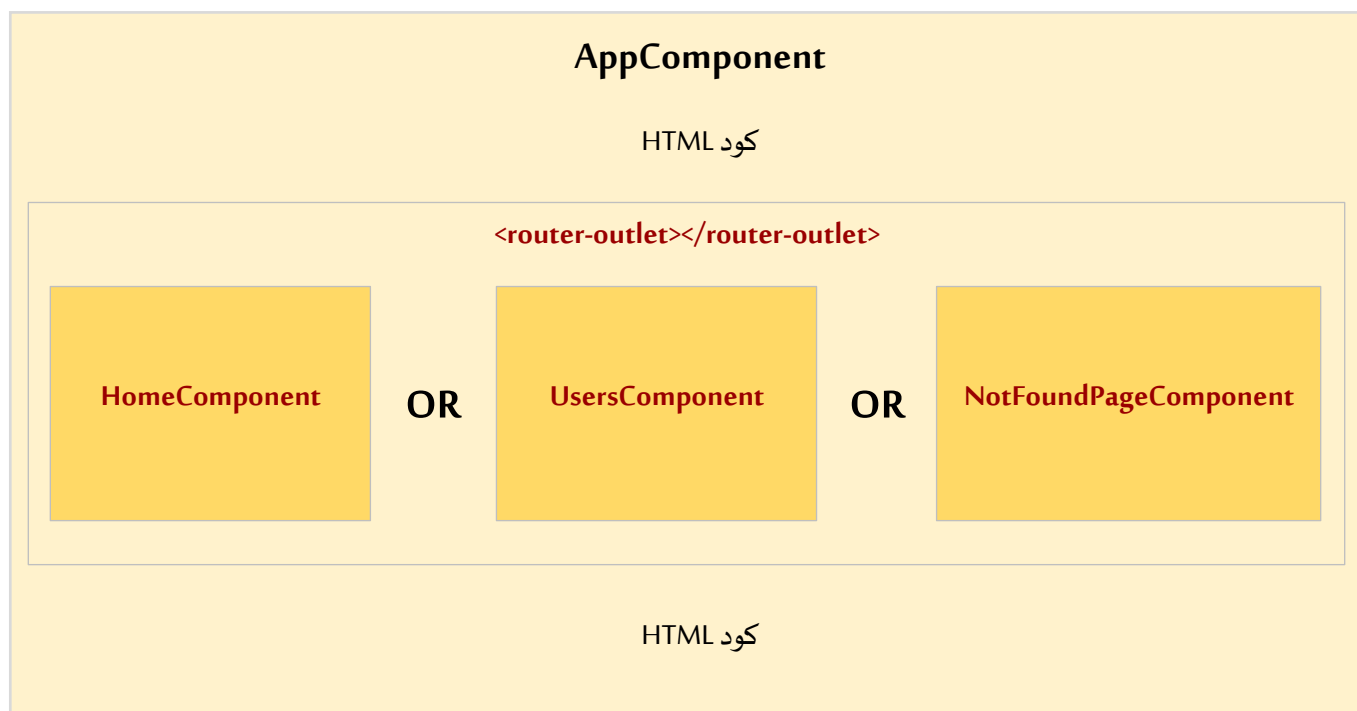
لنفتح هذا الملف ونستعرض محتوياته:

ملف app.component.html

```
1. <h1 style="text-align: center">App Component</h1>
2. <hr />
3. <br />
4.
5. <router-outlet></router-outlet>
6.
```

من السطر 1 إلى السطر 3 هو عبارة عن الكود الذي قمنا بكتابته سابقاً، أما السطر 5 فيوجد عنصر وضعه angular بشكل تلقائي عندما قمنا بإنشاء مشروع جديد واضفنا له الامر --routing، وهو من العناصر الخاصة angular وليس له علاقة بعناصر او تاغات HTML، ولكن هنا يتبادر إلى أذهاننا سؤال، ما هي مهمة هذا العنصر؟ والإجابة عن هذا السؤال بشكل عام هو أن angular يقوم بعرض محتويات أي component تم عمل له routing تحت هذا العنصر فلو كتبنا في الرابط users او ضغطنا على رابط ينقلنا إلى path ذو القيمة users فسوف يقوم angular بجلب component الذي يعرض UsersComponent وعرضها تحت هذا العنصر، وعندما نعمل نفس العملية السابقة لكن للـ path ذو القيمة home فسوف يقوم angular بتدمير UsersComponent وعرض HomeComponent وهكذا مع بقية components الأخرى، مع العلم أنه يمكن كتابة أي كود html تحت هذا العنصر ولكن سوف يُعرض في المتصفح أسفل من component الذي يتم عرضه بواسطة routing، وتستطيع تجريب ذلك بنفسك من خلال كتابة أي اكواد html وسوف ترى انها تظهر اسفل components التي تم عملها routing.

ومن ذلك نستطيع أن نستنتج أن أي كود html يتم كتابته في هذا الملف يكون ثابت ويتم عرضه بشكل دائم في الموقع، أما العنصر `<router-outlet></router-outlet>` فيعرض components التي تم عملها routing بشكل ديناميكي اثناء تشغيل التطبيق، ويمكن تمثيل ذلك بالشكل التالي:



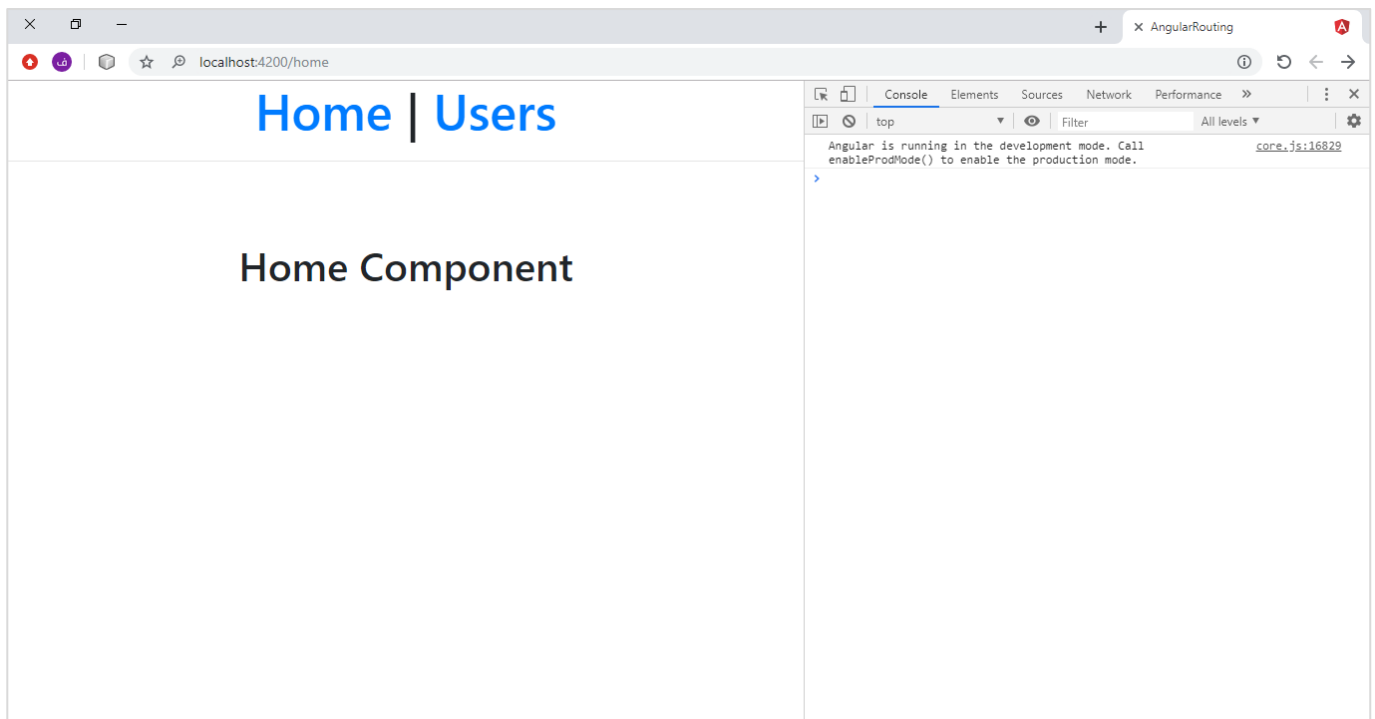
من الشكل السابق نستطيع معرفة أي من أجزاء الموقع نريد أن تكون ظاهرة دائماً وأي أجزاء لا نريدها أن تظهر ألا في حال طلبها من قبل المستخدم، ومن الأجزاء التي نريد ان تكون ظاهرة بشكل دائم هي الروابط، فنريد مثلاً مجموعة من الروابط أن تكون ظاهرة في أعلى الموقع (وبما أنها في أعلى الموقع سوف نكتبها قبل العنصر `<router-outlet></router-outlet>`) وسوف نحتاج إلى رابطين فقط الأول لعرض محتويات HomeComponent والثاني لعرض محتويات UsersComponent اما NotFoundPageComponent فمن البديهي إلا نحتاج له رابط لأنه يتم عرضه في حال كان هنالك خطأ في عنوان الرابط، والآن لنستبدل الكود السابق في ملف HTML الخاص بالـ AppComponent بالكود التالي:

```

1. <h1 style="text-align: center">
2.   <a href="#">Home</a>
3.   |
4.   <a href="#">Users</a>
5. </h1>
6. <hr />
7. <br />
8.
9. <router-outlet></router-outlet>

```

الآن لنقوم بحفظ التعديلات والذهاب إلى المتصفح لمشاهدة نتائج ما قمنا به:



هذا شكل التطبيق ولكن لو تم الضغط على الروابط لم تعمل، ولجعلها تعمل بالشكل المطلوب لابد من عمل الربط بين هذه الروابط والrouting الذي تم تهيئته سابقاً، ونستطيع القيام بذلك عن طريق إضافة بعض الدايركتيف Directives الجاهزة التي قدمتها لنا angular، وسوف نضيف اثنين دايركتيف، هما:

✓ routerLink: وهذا الدايركتيف نستبدله مكان الخاصية href الخاصة بالعنصر أو التاغ <a> لأن هذه الخاصية عند الضغط عليها تقوم بإعادة تحميل الصفحة وهذا مما ينافي أساس تقنيات SPA لذلك نستبدلها بهذا الدايركتيف أما القيمة التي نسند لها هي path للcomponent الذي نريد إظهاره.

✓ routerLinkActive: ويقوم هذا الدايركتيف بإضافة كلاس css لربط الذي تم الضغط عليه أو الرابط المفعّل بحيث يعطي انطباع للمستخدم أن هذا الرابط هو الذي تم الضغط عليه كأن يغير اللون أو يغير الحجم أو أي تأثير آخر، ولقد أنشأت كلاس واسميتها is-active وأضفته لملف style الخاص بالAppComponent أو بمسمى آخر ملف app.component.css، كالتالي:

```

1. .is-active{
2.     color: rgb(161, 156, 207);
3.     font-weight: bolder;
4. }

```

الآن لنضيف هذه الدائريكتيف Directives إلى العنصر <a> في ملف html الخاص بAppComponent ، كالتالي:

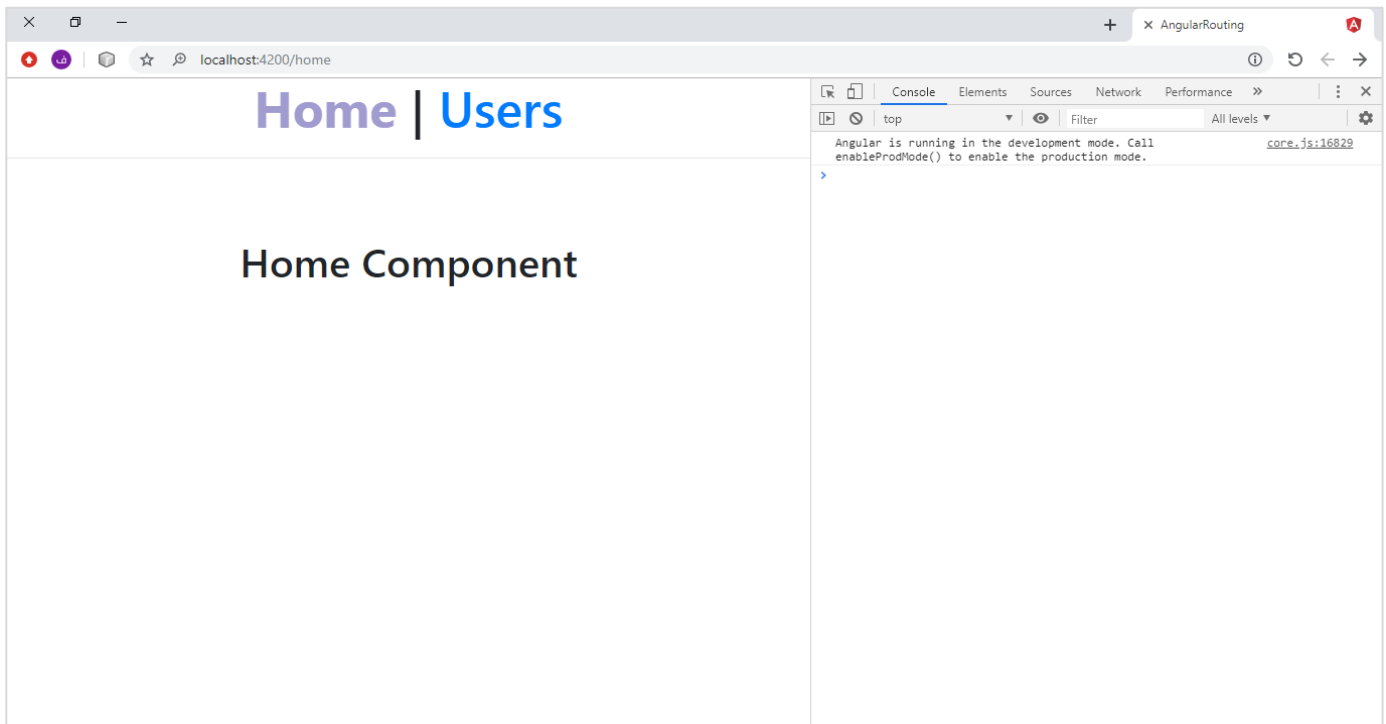
```

1. <h1 style="text-align: center">
2.     <a routerLink="/home" routerLinkActive="is-active">Home</a>
3.     |
4.     <a routerLink="/users" routerLinkActive="is-active">Users</a>
5. </h1>
6. <hr />
7. <br />
8.
9. <router-outlet></router-outlet>

```

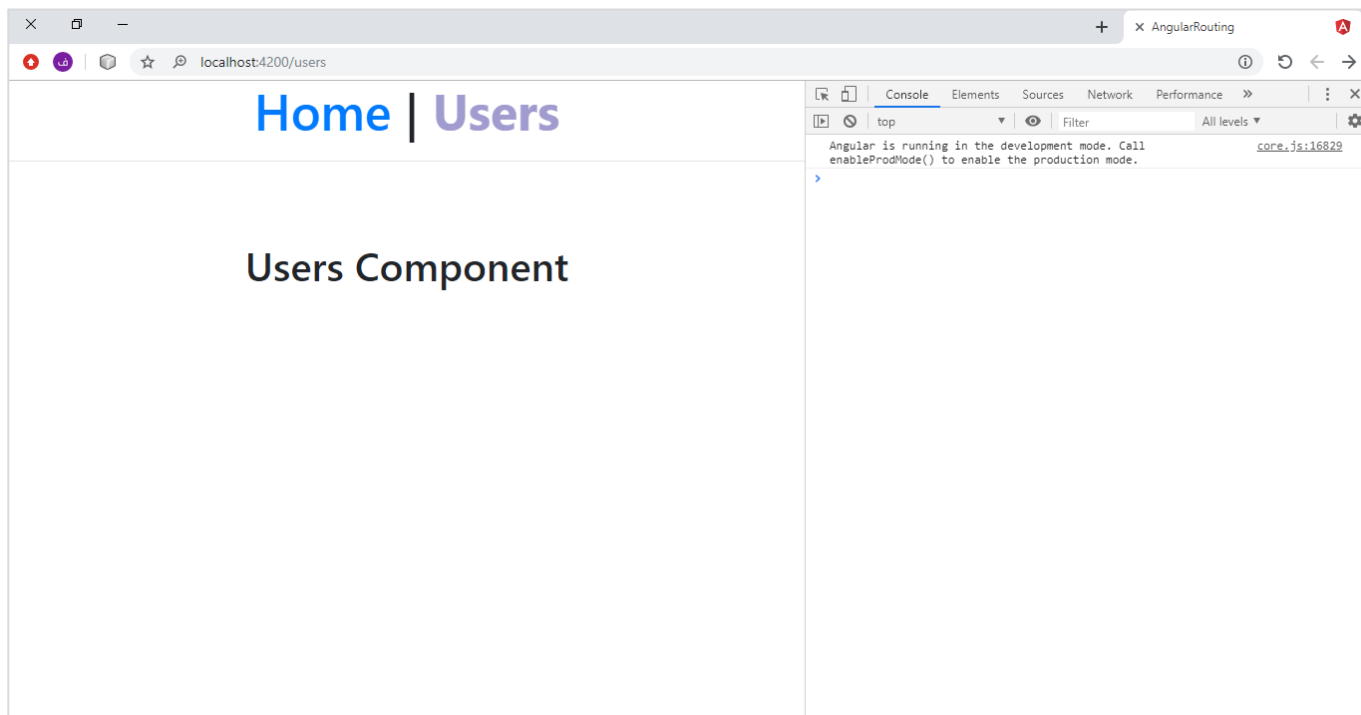
نلاحظ في السطر 2 والسطر 4 تمت إضافة الدائريكتيف السابقة وإسناد القيم المطلوبة لها.

والآن لنقوم بحفظ التعديلات والذهاب للمتصفح لنرى نتيجة ما قمنا به، كالتالي:



نلاحظ أنه في بداية تشغيل التطبيق أن الرابط Home هو المفعل لذلك تمت إضافة الكلاس is-active له وب نفس الوقت صفحة HomeComponent هي الظاهرة، الآن لنضغط على الرابط Users ونرى النتيجة:





ونفس الطريقة في حال الضغط على الرابط Users حيث تمت إضافة الكلاس is-active له وبنفس الوقت ظهر لنا UsersComponent، وكل ذلك تم بدون إعادة تحميل الصفحة وإنما تم تحميل فقط component الذي طلبه المستخدم والسبب في ذلك لأننا استخدمنا routerLink عوضاً عن href.

ملاحظة: بعض الأحيان قد نحتاج إلى إضافة دايركتيف آخر إلى العنصر <a> المربوط في HomeComponent فقط في حال أردنا أي يضيف الكلاس فقط إذا كان path في الرابط يساوي path في الرابط على المتصفح بالتمام وهذا ينتج عنه ان يكون الرابط الخاص home مفعّل دائماً مع ان المستخدم قد يكون ضغط على رابط آخر، ولحل هذه المشكلة نستخدم هذا Directive، مع العلم أن هذا الدايركتيف يُضاف إلى العنصر مع وجود routerLinkActive ولا يتم استبداله به، كالتالي:

```
[routerLinkActiveOptions] = "{exact: true}"
```

ملاحظة: في حال الرغبة بكتابة أكثر من كلاس في الدايركتيف routerLinkActive لأي سبب كان فهناك طريقتين للقيام بذلك، هما:

```
<a routerLink="/users" routerLinkActive = "is-active1 is-active2">Users</a>
```

OR

```
<a routerLink="/users" [routerLinkActive] = ["is-active1", 'is-active2']">Users</a>
```

ملاحظة: في حال أردنا تغيير النص الموجود على الرابط إذا كان مفعّل أي تم الضغط عليه وفي حال لم يكن مفعّل نُرجعه إلى وضعه الطبيعي ولا نريد استخدام كلاسات CSS، ففي هذه الحالة نستخدم الكود التالي:

```
<a routerLink="/users" routerLinkActive #rla="routerLinkActive">
```

```
  {{ rla.isActive ? 'Users Open' : 'Users' }}

```

```
</a>
```

ملاحظة: في حال كانت الروابط تأتي من قاعدة بيانات نقوم أولاً بتجهيز وهيئة وتسمية path لكل route كما هو في قاعدة البيانات، ومن ثم نقوم بعرضها بشكل ديناميكي على template الخاص بcomponent، وبما أنه ليس لدينا قاعدة بيانات حالياً، لنقوم بإنشاء مصفوفة كائنات في ملف ts الخاص بAppComponent بحيث أن كل كائن داخل هذه المصفوفة يمثل route الذي نريد عرضه، كالتالي:

```
routes = [  
  
  { linkName: 'Home', URL: '/home' },  
  
  { linkName: 'Users', URL: '/users' }  
  
];
```

ومن ثم في ملف template نقوم باستخدام حلقة Loop لإنشاء الروابط بناءً على المصفوفة السابقة، كالتالي:

```
<a *ngFor="let route of routes" [routerLink] = "route.URL" routerLinkActive="is-active">  
  
  {{ route.linkName }}  
  
</a>
```

نلاحظ أن الـ routerLink الـ daيركتيف تم إضافة الأقواس المربعة له وهذا ما يعني أن القيمة التي يستقبلها هي قيمة ديناميكية مخزنة في متغير معين وليست قيمة نصية كما مررناها سابقاً، أو بمعنى آخر استخدمنا ميزة Property Binding، ولمعرفة أكثر عن هذه الميزة الرجاء مراجعة الكتاب الثاني من هذه السلسلة Angular Components and Services.

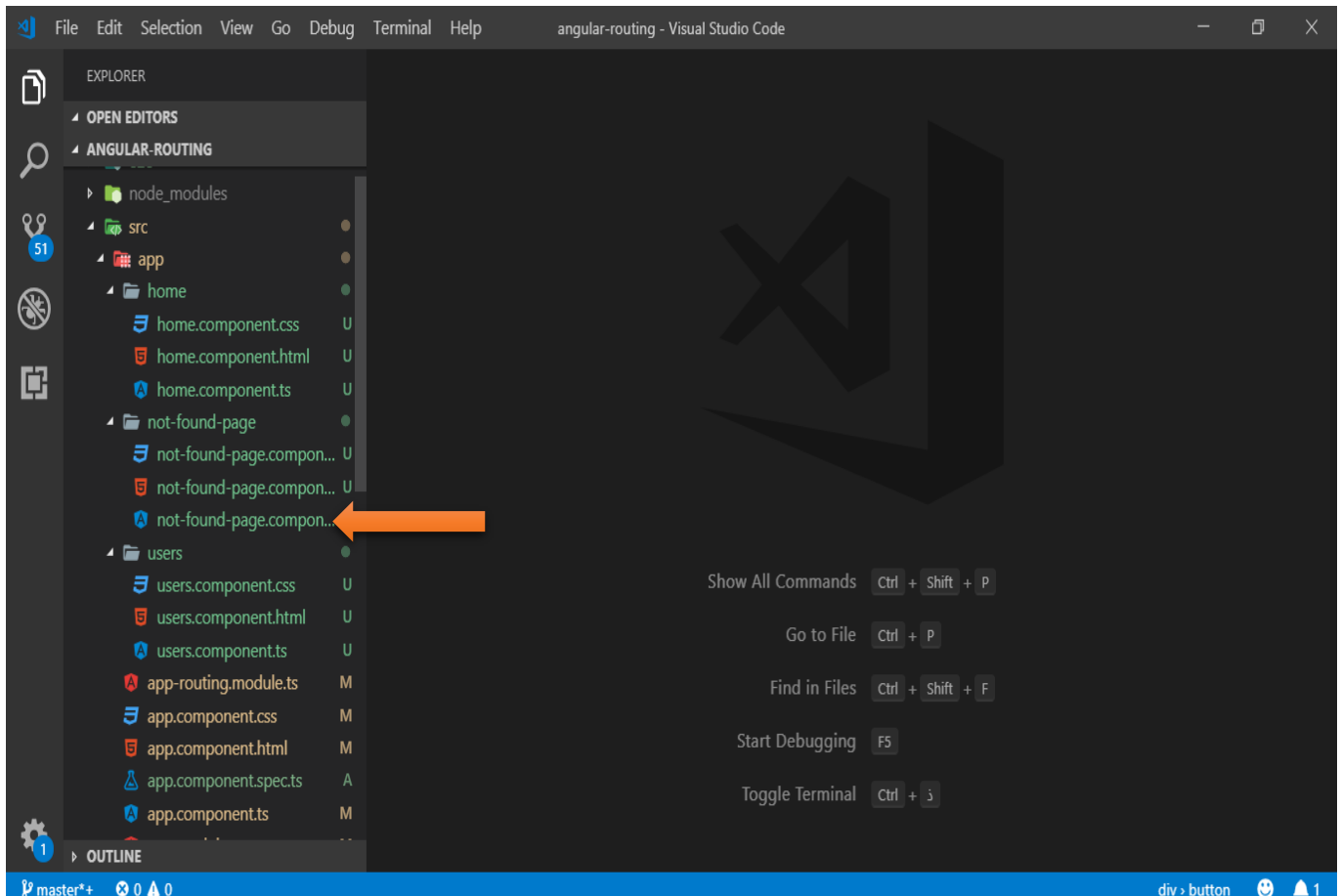
وبذلك نكون أنهيينا طريقة الربط في template الخاص بcomponent، ولاحظنا كيف أن angular سهل علينا عملية الربط بشكل بسيط وسلس، فكل ما نفعله هو إضافة بعض الـ daيركتيف Directives وإسناد القيم المطلوبة وهو الذي سوف يقوم بباقي الأمور عننا نحن كمطورين، أما في الجزء التالي سوف نتكلم عن طريقة الربط البرمجي أو الربط عن طريق ملف class أو ملف ts للـ component (تختلف المسميات والمعنى واحد).

## 2.5.1. الربط البرمجي:

يتيح لنا angular بالإضافة إلى الربط عن طريق template للـ component أن نقوم بالربط برمجياً عن طريق ملف class للـ component، وهذه الطريقة لها استخدامات متعددة منها على سبيل المثال في حال قام المستخدم بتسجيل الدخول عن طريق كتابة اسم المستخدم وكلمة السر، إذا كانت صحيحة نقوم بتوجيهه برمجياً للدخول إلى النظام أو الموقع وفي حال غير ذلك نظهر له رسالة خطأ أو نقوم أيضاً بتوجيهه إلى صفحة أخرى في حال نسيانه لكلمة المرور أو صفحة التسجيل إذا كان غير مسجل، وجميع هذه الأمور تتم برمجياً، وهذا فقط على سبيل المثال وإلا هو في الحقيقة له استخدامات أخرى متعددة سوف نتعامل مع بعضها بإذن الله في الأجزاء القادمة، أما هنا فسوف نعرف المفاهيم الأساسية وكيف يتم الربط برمجياً.

لنفرض أننا نريد إضافة زر في صفحة `NotFoundPageComponent` ومهمة هذا الزر هو إرجاع المستخدم إلى صفحة `HomeComponent`، ونستطيع القيام بذلك بكل سهولة برمجياً من خلال الاستفادة من service ذات الاسم `Router` حيث أن هذه service تقدم لنا مجموعة كبيرة من الدوال والخصائص التي تسهل لنا التعامل مع `routing`، وما يهمنا من هذه الدوال حالياً هي الدالة `navigate` حيث نمرر لهذه الدالة `path` الخاص بـ `component` الذي نريد تحويل المستخدم إليه، وتتم هذه الأمور من خلال اتباع الخطوات التالية:

(١) نذهب إلى ملف `app.component.ts` أو ملف `class` لـ `NotFoundPageComponent` ، كالتالي:



(٢) نفتح هذا الملف ومن ثم نستدعي الـ `service` ذو الاسم `Router` ومن ثم نعمل له `inject` في `constructor` وليكن اسم المتغير `router`، كالتالي:

```
ملف not-found-page.component.ts
1. import { Component, OnInit } from '@angular/core';
2. import { Router } from '@angular/router';
3.
4. @Component({
5.   selector: 'app-not-found-page',
6.   templateUrl: './not-found-page.component.html',
7.   styleUrls: ['./not-found-page.component.css']
8. })
9. export class NotFoundPageComponent implements OnInit {
10.
11.   constructor(private router: Router) { }
```

```

12.
13. ngOnInit() {
14. }
15.
16.}

```

(٣) بعدها ننشئ دالة وليكن اسمها `goToHomePage()` بحيث تُنفذ هذه الدالة في الحدث `click` في الزر الذي سوف ننشئه بعد قليل في ملف `template` او ما يسمى ملف HTML لنفس هذا `component`، أما مهمة هذه الدالة فتقوم بإعادة توجيه المستخدم إلى صفحة `HomeComponent` وذلك عن طريق الدالة `navigate` الموجودة من ضمن `service` ذو الاسم `Router` ونستطيع الوصول لها عن طريق المتغير الذي عرفناه سابقاً باسم `router` ونمرر لهذه الدالة `path` ذو القيمة `home` على شكل مصفوفة نصية، كالتالي:

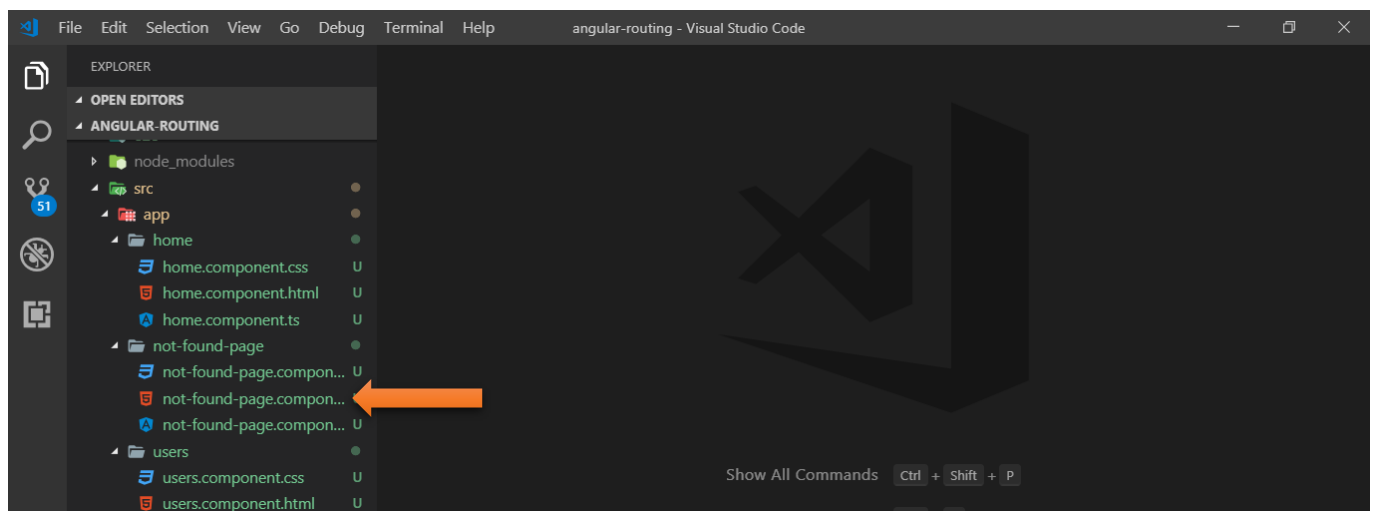
ملف `not-found-page.component.ts`

```

1. import { Component, OnInit } from '@angular/core';
2. import { Router } from '@angular/router';
3.
4. @Component({
5.   selector: 'app-not-found-page',
6.   templateUrl: './not-found-page.component.html',
7.   styleUrls: ['./not-found-page.component.css']
8. })
9. export class NotFoundPageComponent implements OnInit {
10.
11.   constructor(private router: Router) { }
12.
13.   ngOnInit() {
14.   }
15.
16.   goToHomePage() {
17.     this.router.navigate(['/home']);
18.   }
19.
20.}

```

(٤) بعد تجهيزنا لهذه الدالة نذهب إلى ملف `template` لنفس `component`، كالتالي:



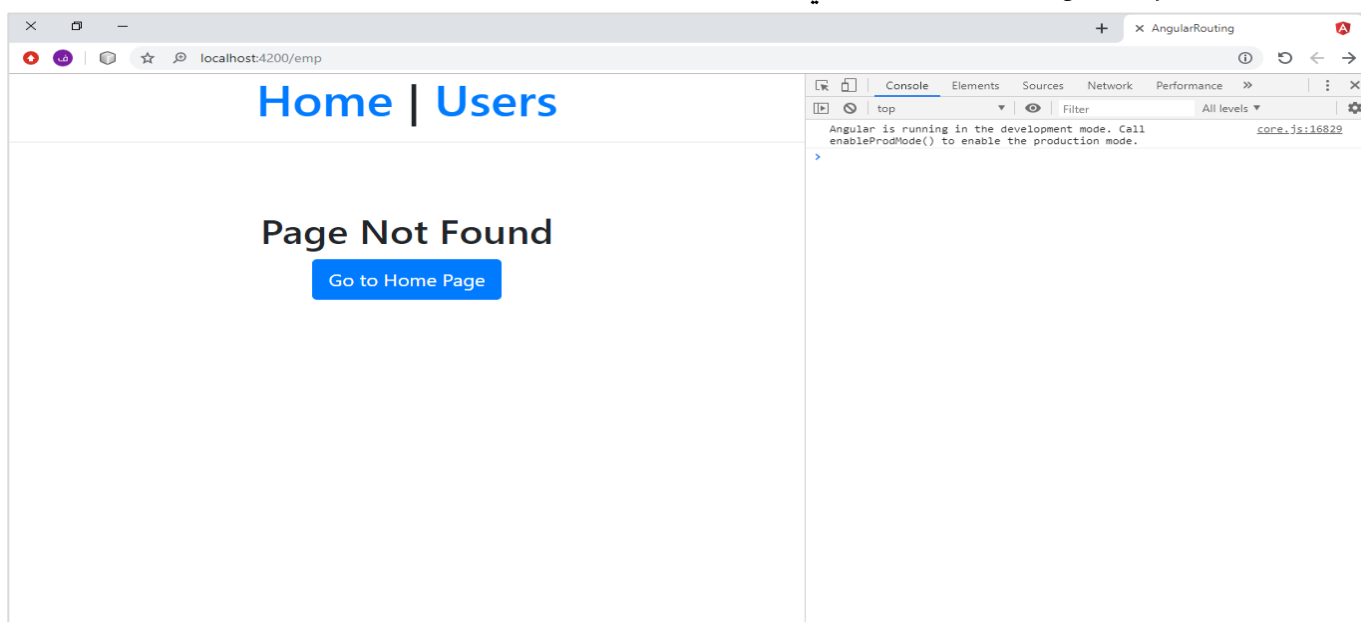
٥) وبعد فتحنا لهذه الملف نضيف له زر وفي حدث click لهذه الزر نضيف الدالة التي تم إنشاءها سابقاً، كالتالي:

ملف not-found-component.html

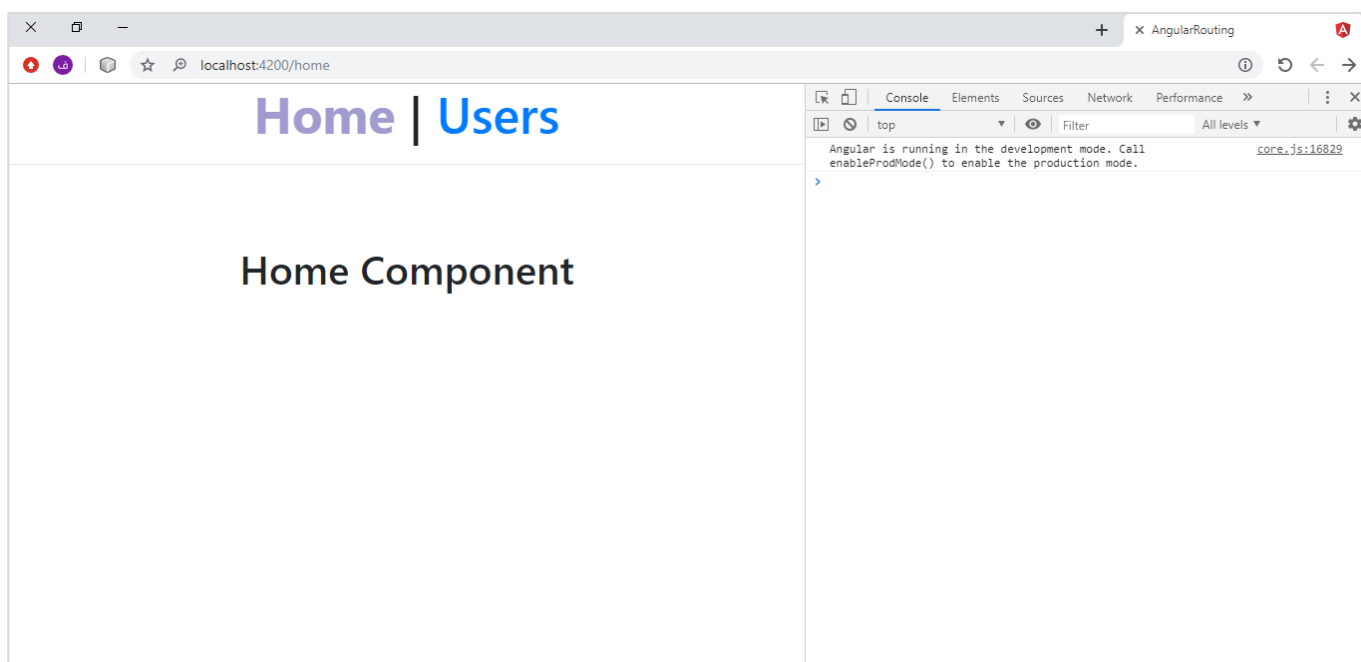
```
1. <br />
2. <h2 style="text-align: center">Page Not Found</h2>
3. <div class="d-flex justify-content-center">
4.   <button class="btn btn-primary" (click)="goToHomePage()">
5.     Go to Home Page
6.   </button>
7. </div>
```



بعد الانتهاء من الخطوات السابقة لنقم الآن بحفظ التعديلات والذهاب للمتصفح ومن ثم كتابة أي رابط خاطئ لكي تظهر لنا صفحة NotFoundPageComponent، كالتالي:



نلاحظ ظهور الزر في هذه الصفحة مع عدم تفعيل أي رابط من الرابطين الموجودين فيها، الآن لنقم بضغط على هذا الزر، ونرى النتيجة:



نلاحظ انه بعد الضغط على الزر السابق تم انتقالاً برمجياً إلى هذه الصفحة وبنفس الوقت قام بتفعيل الرابط Home وإضافة كلاس is-active له.

وبذلك نكون أنهينا هذا الجزء مع العلم أننا سوف نتطرق لهذا الربط البرمجي في أكثر من موضع في هذا الكتاب وخصوصاً عندما نتكلم عن الجزء الخاص بـ parameters ، وسوف نتكلم في الجزء التالي عن كيفية تصميم صفحات التطبيق وإضافة بعض البيانات لكي يكون ذو شكل مقبول وبنفس الوقت يحتوي على بيانات لكي نُحاكي المواقع وتطبيقات الويب الواقعية اثناء تعلمنا لتقنيات Angular Routing.

## 6.1. تصميم صفحات التطبيق وإضافة البيانات:

في هذا الجزء سوف نعيد تصميم صفحات التطبيق عن طريق إطار العمل Bootstrap (لمعرفة كيفية إضافة مكتبة Bootstrap إلى مشاريع Angular الرجاء مراجعة الكتاب الأول من هذه السلسلة Angular Environment Setup بالتحديد الفصل الأخير من الكتاب)، وإضافة بعض البيانات (dummy data)، وسوف نقسم هذا الجزء إلى جزئين الجزء الأول نتكلم فيه عن تصميم الصفحات بالمجمل بدون شرح الجزئيات لأنه ليس المقام لشرحها، والجزء الثاني كيفية إضافة البيانات وسوف نشرحها بشكل سريع.

### 1.6.1. إعادة تصميم الصفحات:

كما قلنا سابقاً في مقدمة السلسلة أنه يجب عليك عزيزي المتعلم ان تكون ملم على الأقل بأساسيات أركان تطوير الويب الثلاثة HTML و CSS و Javascript، لذلك سوف استعرض فقط الكود الذي سوف اضيفه بالاستعانة بمكتبة Bootstrap وبعض أكواد CSS و HTML، كالتالي:

ملف app.component.html

```
1. <nav class="navbar navbar-expand-sm navbar-dark bg-dark Shadow">
2.   <div class="navbar-brand-two" href="#">
3.     
4.   </div>
5.   <span class="navbar-brand">Routing</span>
6.   <div class="justify-content-left">
7.     <ul class="navbar-nav">
8.       <li class="nav-item">
9.         <a routerLink="/home" routerLinkActive="is-active"><span>Home</span></a>
10.      </li>
11.      <li class="nav-item">
12.        <a routerLink="/users" routerLinkActive="is-active"><span>Users</span></a>
13.      </li>
14.    </ul>
15.  </div>
16.</nav>
17.
18.<router-outlet></router-outlet>
```

#### ملف app.component.css

```

1. .Shadow {
2.     box-shadow: 0 0 0 1.2px grey;
3. }
4. .is-active {
5.     background-color:gray;
6.     font-weight: bold;
7.     border: 1px solid rgb(27, 26, 26);
8.
9. }
10. .is-active span {
11.     border-bottom-style: solid;
12.     border-bottom-width: 3px;
13.     border-bottom-color: #84ff6e;
14.     padding-bottom: 4px;
15. }
16. li a{
17.     color: white;
18.     padding: 10px;
19.     margin: 4px;
20. }
21. li a:hover {
22.     text-decoration: none;
23. }

```

#### ملف not-found-page.component.html

```

1. <div class="row justify-content-center">
2.     
9. </div>
10.
11. <div class="row justify-content-center">
12.     <h2 class="mt-3">Page Not Found</h2>
13. </div>
14.
15. <div class="row justify-content-center">
16.     <button class="btn btn-dark mt-3 radius" (click)="goToHomePage()">
17.         Go to HOMEPAGE
18.     </button>
19. </div>

```

#### ملف not-found-page.css

```

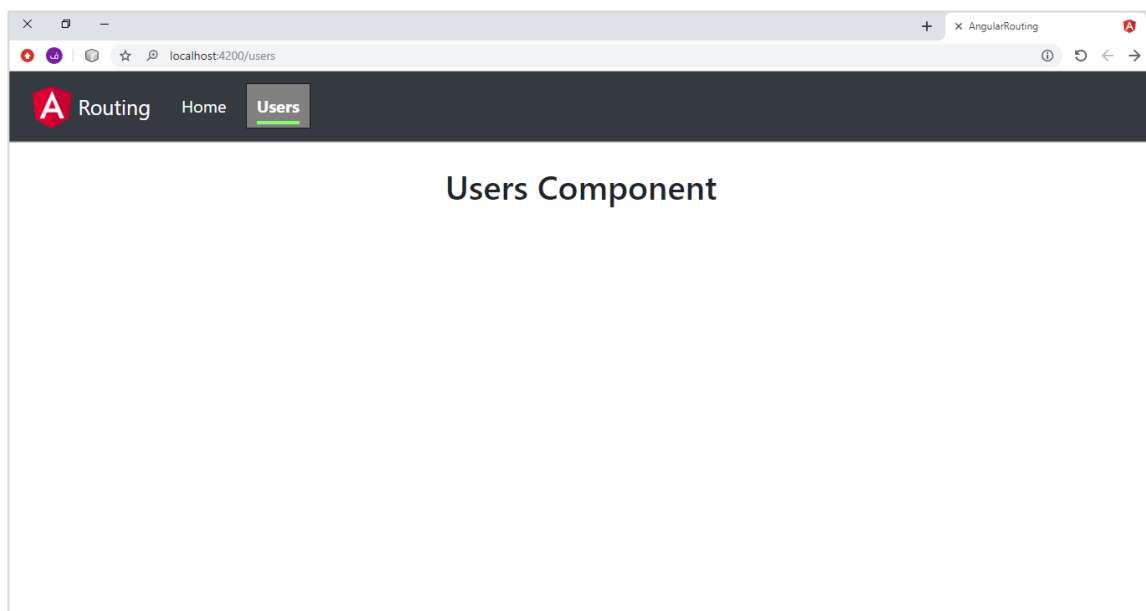
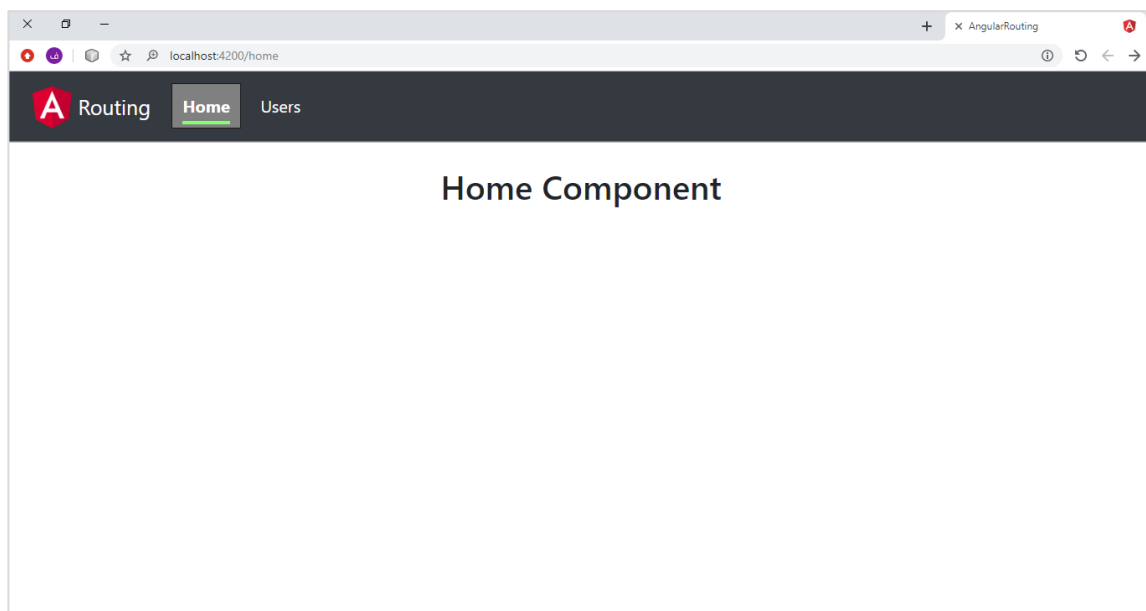
1. .radius{
2.     border-radius: 15px;
3. }

```

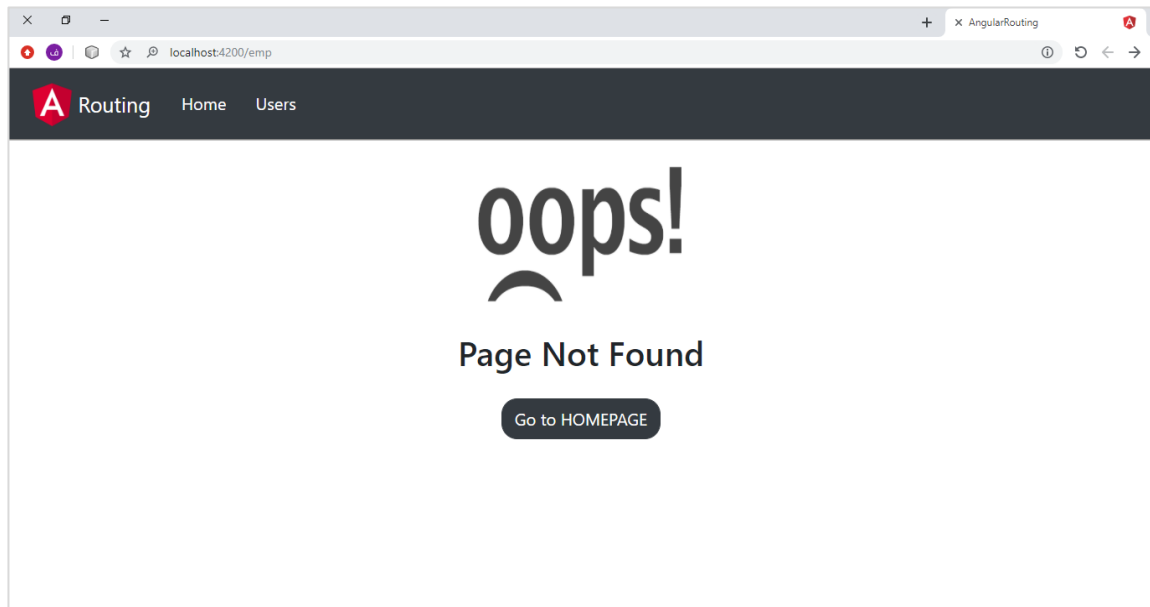
ملاحظة: في ملف `app.component.html` وتحديداً في السطر 3 تم وضع صورة عشوائية في مجلد `assets` وفي هذا السطر تم كتابة مسار هذه الصورة.

ملاحظة: في ملف `not-found-page.component.html` وتحديداً في السطر 3 تم وضع صورة عشوائية في مجلد `assets` وفي هذا السطر تم كتابة مسار هذه الصورة.

أما الآن لنحفظ التعديلات ونذهب للمتصفح ولنشاهد ما قمنا به من تعديلات:

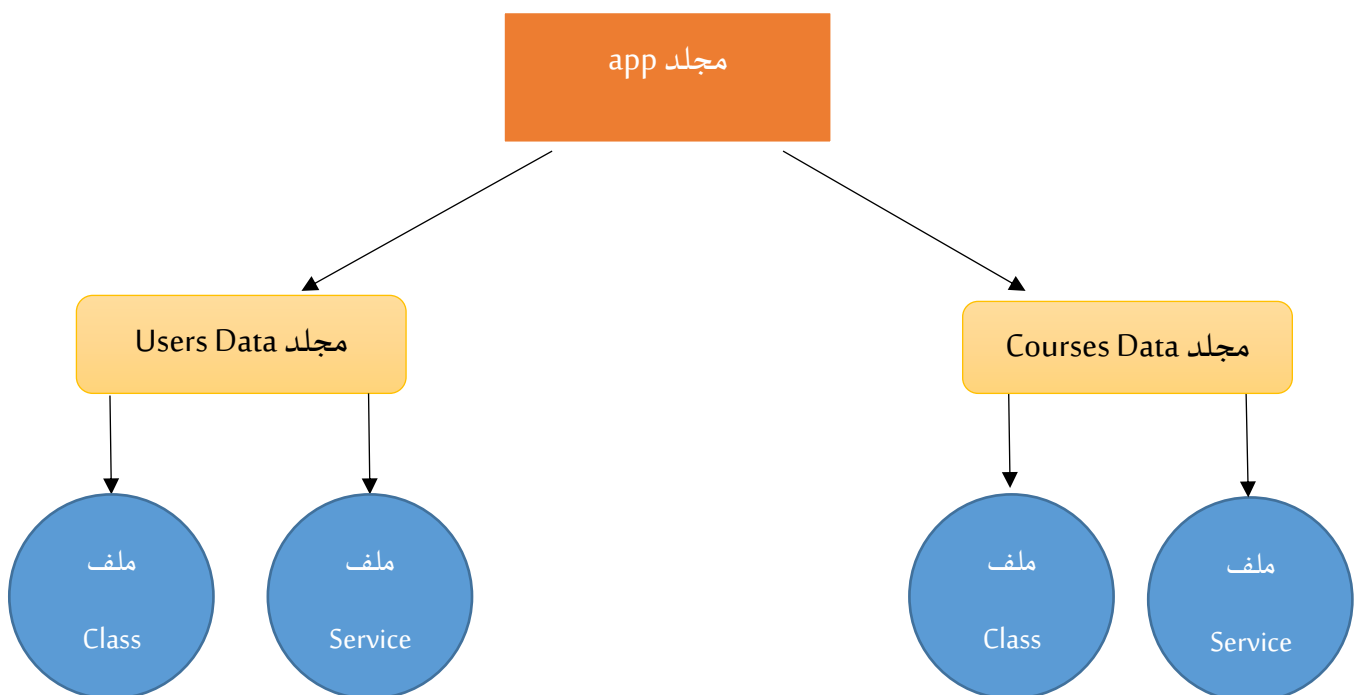






## 2.6.1. إضافة البيانات:

سوف نُقوم في هذا الجزء بإضافة البيانات إلى الصفحات وبما انه ليس لدينا قاعدة بيانات لكي نجلب منها البيانات سوف نُحاكي هذا الأمر عن طريق إنشاء service لكل component وفي كل service نعرف مصفوفة من الكائنات ونقوم بتعبئتها ببعض البيانات العشوائية وفي كل service نعرف دالة لقراءة هذه البيانات بعد تحويلها إلى observable، ولكي تكون المحاكاة أكثر واقعية نقوم بإنشاء كلاس كل كلاس نعرف فيه مجموعة من المتغيرات تقابل عدد ونوع البيانات الموجودة في كل مصفوفة من مصفوفات الكائنات التي عرفناها في كل service او بصيغة أخرى لوصف هذه البيانات، ومن أجل التنظيم والترتيب سوف نقوم بإنشاء مجلدين فرعيين تحت المجلد app بحيث يكون المجلد الأول باسم Courses Data يحتوي على service و class الخاص بعرض بيانات الكورسات في HomeComponent والمجلد الآخر يحتوي على نفس الملفات في المجلد الأول ولكن يختص بعرض بيانات المستخدمين في UsersComponent، ويمكن تمثيلهما بالشكل التالي:



مع العلم أنني سوف أقوم بتسمية الملفات التابعة للمجلد Courses Data باسم courses، والملفات التابعة للمجلد Users Data باسم users، الآن لنقم بفتح شاشة terminal جديدة كما تعلمنا سابقاً ونتأكد أننا بنفس مسار المشروع ومن ثم نقوم بكتابة الأوامر التالية لإنشاء المجلدات والملفات السابقة، كالتالي:

```
ng g service courses-data/courses
```

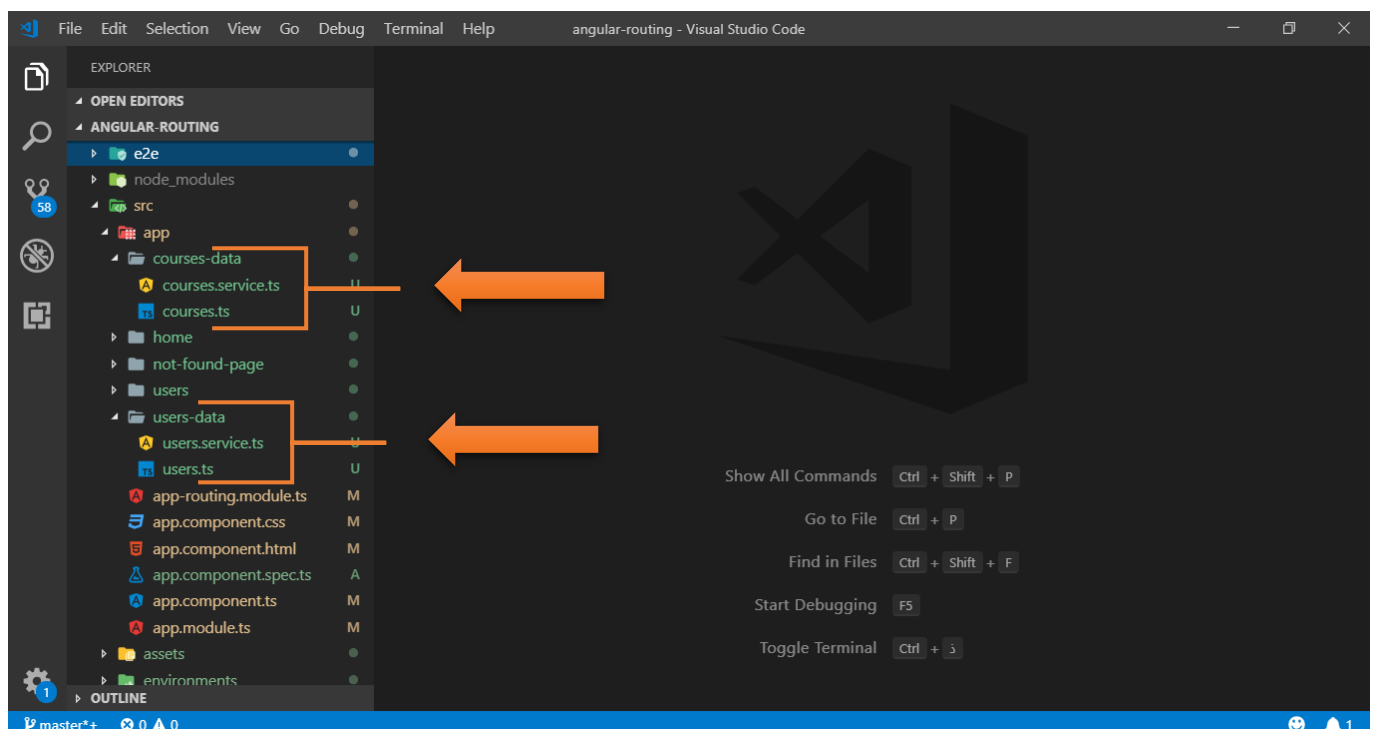
```
ng g class courses-data/courses
```

```
ng g service users-data/users
```

```
ng g class users -data/ users
```

ولمعرفة أكثر عن التعامل مع Angular CLI او Services الرجاء مراجعة الكتاب الأول والثاني من هذه السلسلة.

بعد الانتهاء من إنشاء الملفات السابقة سوف تكون بالشكل التالي:



الآن لنقم بإضافة الأكواد لوصف بيانات الكورسات، كالتالي:

ملف courses.ts

```
1. export class Courses {
2.   constructor(
3.     public courseName: string,
4.     public courseDescription: string,
5.     public path: string
6.   ) { }
7. }
```

وهنا فقط قمنا بتعريف ثلاث متغيرات (خصائص) في constructor الخاص بالكلاس تقابل نوع وعدد البيانات، وهي متغير `courseName` لتخزين اسم الكورس و `courseDescription` لتخزين معلومات عن الكورس و `path` لتخزين مسار صورة لربط اونلاين من الانترنت لعرض صورة معينة لكل كورس، ونستطيع أن نقول أن هذه المتغيرات تشابه الحقول في جداول قاعدة البيانات او بصيغة أخرى وصف لهذه الحقول في التطبيق.

أما الآن لننتقل إلى ملف `courses.service.ts`، ونقوم بكتابة الأكواد الخاصة به، كالتالي:

#### ملف `courses.service.ts`

```
1. import { Injectable } from '@angular/core';
2. import { Courses } from './courses';
3. import { of, Observable } from 'rxjs';
4.
5. const COURSES = [
6.   new Courses(
7.     'Angular',
8.     'Angular Course for Beginners',
9.     'https://via.placeholder.com/200/808080/FFFFFF?text=ANGULAR'
10.  ),
11.   new Courses(
12.     'HTML',
13.     'Advanced Course For Web Developers',
14.     'https://via.placeholder.com/200/808080/FFFFFF?text=HTML'
15.  ),
16.   new Courses(
17.     'CSS',
18.     'CSS From Zero to Hero',
19.     'https://via.placeholder.com/200/808080/FFFFFF?text=CSS'
20.  ),
21.   new Courses(
22.     'Javascript',
23.     'Learn Modern Javascript Advanced Topics',
24.     'https://via.placeholder.com/200/808080/FFFFFF?text=JAVASCRIPT'
25.  ),
26.   new Courses(
27.     'Typescript',
28.     'Learn Typescript From Scratch',
29.     'https://via.placeholder.com/200/808080/FFFFFF?text=TYPESCRIPT'
30.  )
31. ];
32.
33. const coursesList$ = of(COURSES);
34.
35. @Injectable({
36.   providedIn: 'root'
37. })
38.
39. export class CoursesService {
40.
41.   constructor() { }
```

```

42.
43. getAllCourses(): Observable<Courses[]> {
44.     return coursesList$;
45. }
46.
47. }

```

في السطر 2 استدعينا الكلاس الذي أنشأناه سابقاً لتخزين البيانات في المتغيرات الموجودة فيه، وفي السطر 3 استدعينا الدوال اللازمة للتعامل مع observable، وفي الأسطر من 5 إلى 31 قمنا بتعريف مصفوفة كائنات ومررنا لها قيم معينة على شكل كائن بحيث كل قيمه هي نسخة من الكلاس السابق، وفي السطر 33 عرفنا متغير من النوع observable عن طريق إضافة \$ له واسندنا له مصفوفة الكائنات بعد تحويلها إلى observable عن طريق الدالة of، وفي الأسطر من 43 إلى 45 عرفنا دالة تُرجع observable ونوعه هو نفس نوع الكلاس Courses على شكل مصفوفة بحيث تعيد بيانات الكورسات على شكل مصفوفة observable. ولا تقلق عزيزي المتعلم من جميع ماتم ذكره سابقاً أو سوف يتم ذكره لاحقاً في جزء إضافة البيانات لأن فهمك لهذه التقنيات من عدمها لن تؤثر على الهدف الأساسي من هذا الكتاب وهو فهم Angular Routing، ولعلنا ان أراد الله سبحانه نفرد كتاب خاص بتقنيات HTTP Client و Rxjs.

أما الآن لنقوم بفتح الملفين users.ts و users.service.ts وإضافة الأكواد الخاصة بهما مع العلم أن الأكواد مشابهة لما ذكرناه سابقاً مع اختلاف المسميات ونوع البيانات.

#### ملف users.ts

```

1. export class Users {
2.     constructor(
3.         public userId: number,
4.         public name: string,
5.         public userCity: string,
6.         public userName: string,
7.         public password: string,
8.         public userType: string
9.     ) { }
10. }

```

#### ملف users.service.ts

```

1. import { Injectable } from '@angular/core';
2. import { Users } from './users';
3. import { of, Observable } from 'rxjs';
4.
5. const USERS = [
6.     new Users(1, 'Faisal', 'Riyadh', 'DevFaisal', '1234', 'admin'),
7.     new Users(2, 'Saad', 'Riyadh', 'DevSaad', '1111', 'no-admin'),
8.     new Users(3, 'Bader', 'Dammam', 'DevBader', '2222', 'no-admin'),
9.     new Users(4, 'Fahd', 'Jeddah', 'DevFahd', '3333', 'no-admin'),
10. ];
11.
12. const usersList$ = of(USERS);
13.
14. @Injectable({
15.     providedIn: 'root'
16. })

```

```

17.
18.export class UsersService {
19.
20.  constructor() { }
21.
22.  getAllUsers(): Observable<Users[]> {
23.    return userList$;
24.  }
25.
26.}

```

لنقوم بعرض البيانات في components ولنبدأ في HomeComponent وسوف نضيف فيه الكورسات، الآن لنفتح ملف HomeComponent.ts class ونضيف فيه الأكواد اللازمة ومن ثم نشرح المهم منها، كالتالي:

ملف home.component.ts

```

1. import { Component, OnInit } from '@angular/core';
2. import { Observable } from 'rxjs';
3. import { CoursesService } from '../courses-data/courses.service';
4. import { Courses } from '../courses-data/courses';
5.
6. @Component({
7.   selector: 'app-home',
8.   templateUrl: './home.component.html',
9.   styleUrls: ['./home.component.css']
10.})
11.
12.export class HomeComponent implements OnInit {
13.
14.  courseList$: Observable<Courses[]>;
15.
16.  constructor(private courses: CoursesService) { }
17.
18.  ngOnInit() {
19.    this.courseList$ = this.courses.getAllCourses();
20.  }
21.
22.}

```

في السطر 14 عرفنا متغير على انه مصفوفة من observable من نفس نوع class الذي انشأنه سابقاً لوصف هذه البيانات، وفي السطر 16 قمنا بعمل حقن لinject Service التي تحتوي على الدالة التي تُعيد جميع المصفوفات، ومن ثم في دالة Lifecycle Hook قمنا بإضافة بإسناد بيانات الكورسات إلى هذا المتغير، والخطوة التالية هي نعمل Bind لهذا المتغير في ملف HTML لهذا Component لكي نقوم بعمل قراءة لجميع البيانات وباستخدام حلقة ngFor نقوم بإنشاء مجموعة من العناصر بناءً على عدد البيانات، كالتالي: (ولمعرفة أكثر عن \*ngFor الرجاء مراجعة الكتاب الثالث من هذه السلسلة (Angular Pipes And Directives

ملف home.component.html

```

1. <div style="margin: 1rem">
2.   <div class="card-columns">

```

```

3.     <div class="card bg-light mb-3" *ngFor="let course of courseList$ | async">
4.         <div class="card-header">{{ course.courseName }}</div>
5.         <img class="card-img-top" width="80" height="100" [src]="course.path" />
6.         <div class="card-body pt-1" style="height: 5rem">
7.             {{ course.courseDescription }}
8.         </div>
9.         <div class="card-footer">
10.            <button class="btn btn-secondary">Enroll</button>
11.        </div>
12.    </div>
13. </div>
14.</div>

```

حلقة التكرار تبدأ في السطر 3 حيث قمنا بعمل Loop على جميع البيانات في المتغير courseList\$ ومع كل دورة تخزن البيانات في المتغير course بمعنى نحن لدينا بيانات لخمس كورسات مخزنة في المتغير courseList\$ وفي الدورة الأولى تخزن بيانات الكورس الأول في المتغير course ومن ثم يقوم بعرض اسم هذا الكورس في السطر 4 ومعلومات عنه في السطر 7 وصورة هذا الكورس في السطر 5 ومن ثم في الدورة الثانية يخزن بيانات الكورس الثاني في نفس المتغير course ويقوم بعرض البيانات وهكذا إلا أن ينتهي من جميع البيانات في المتغير courseList\$، أما الفائدة من وضع pipe ذو الاسم async فهو لتسهيل التعامل مع البيانات غير التزامنية والاتصال وقطع الاتصال عند بداية فتح component أو اغلاقه.

والآن لنقوم بنفس ما قمنا به سابقاً ولكن مع UsersComponent وملفات service و class الخاصة ببيانات المستخدمين، مع العلم انها مشابهه لما قمنا به هنا، كالتالي:

#### ملف users.component.ts

```

1. import { Component, OnInit } from '@angular/core';
2. import { Observable } from 'rxjs';
3. import { Users } from '../users-data/users';
4. import { UsersService } from '../users-data/users.service';
5.
6. @Component({
7.   selector: 'app-users',
8.   templateUrl: './users.component.html',
9.   styleUrls: ['./users.component.css']
10.})
11. export class UsersComponent implements OnInit {
12.   userList$: Observable<Users[]>;
13.   constructor(private users: UsersService) { }
14.
15.   ngOnInit() {
16.     this.userList$ = this.users.getAllUsers();
17.   }
18.
19. }

```

#### ملف users.component.html

```

1. <div class="container" style="padding-top: 1rem">
2.   <ul class="list-group" *ngFor="let user of userList$ | async">

```

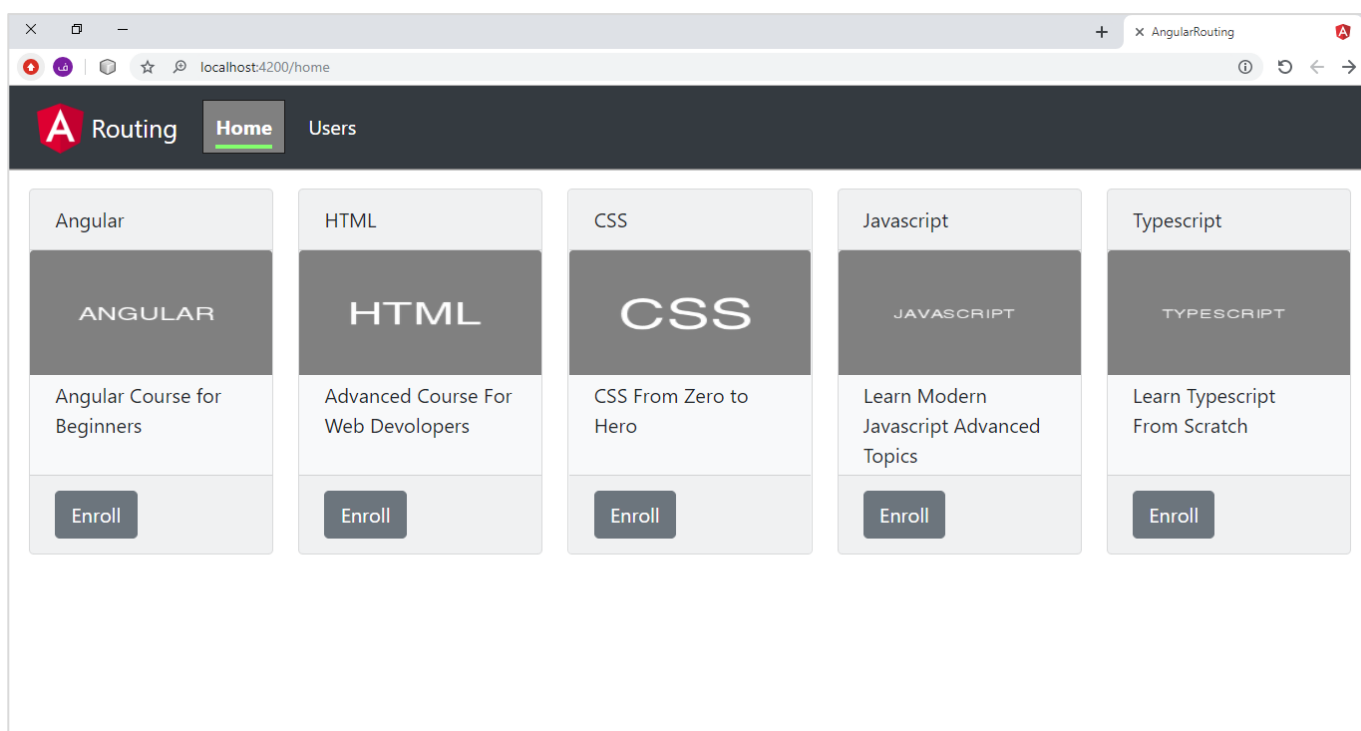
```

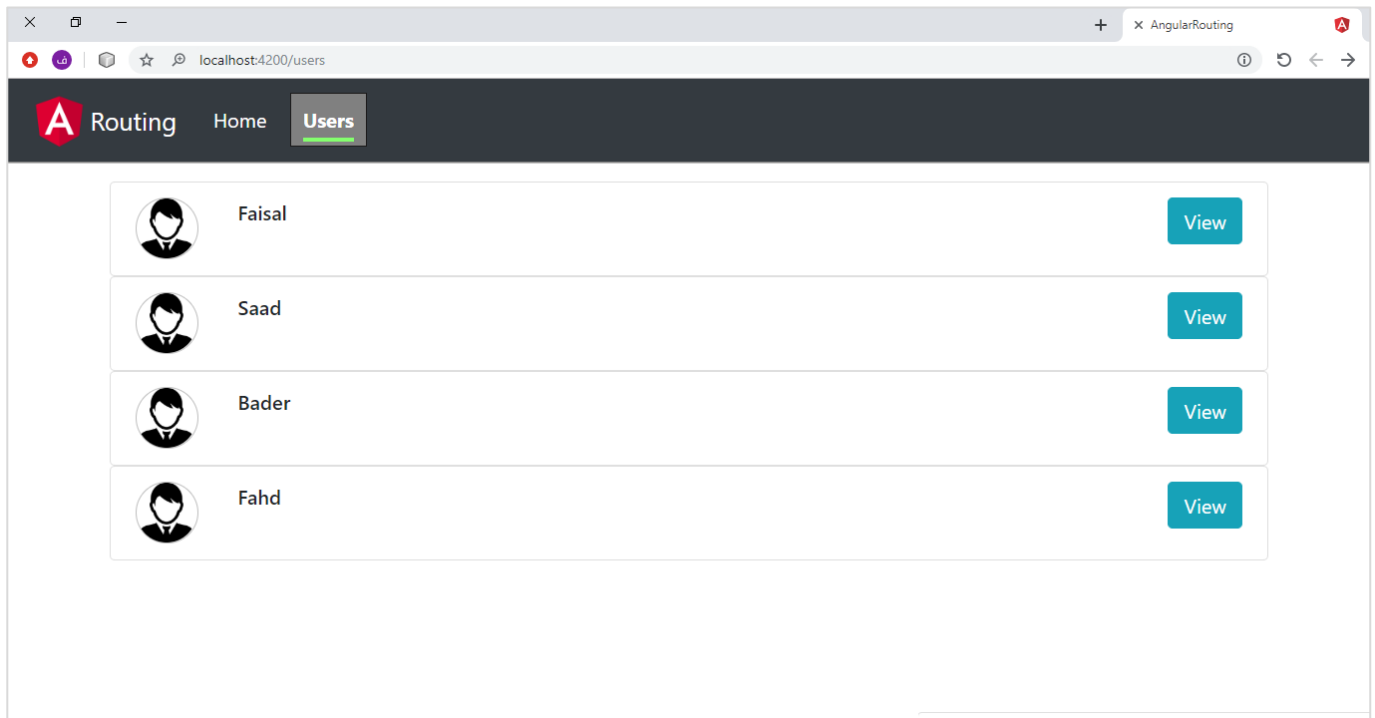
3.     <li class="list-group-item">
4.         <span class="pull-left ">
5.             
10.        </span>
11.        <h6 style="padding-left: 2rem; display: inline">{{ user.name }}</h6>
12.        <span class="pull-right">
13.            <button class="btn btn-info" style="display: inline">View</button>
14.        </span>
15.    </li>
16. </ul>
17.</div>

```

في السطر 6 اضفت صورة عشوائية في مجلد assets باسم 222.png وهذا السطر يحتوي على مسار هذه الصورة.

الآن لنقوم بحفظ التعديلات والذهاب للمتصفح لمشاهدة نتيجة ما قمنا به، وفي حال عدم تحديث المشروع الرجاء الذهاب إلى terminal الخاص بالاتصال بالمشروع وقطع الاتصال عن طريق الضغط على زر ctrl من لوحة المفاتيح وحرف c ومن ثم كتابة الامر `ng serve -o` من جديد.



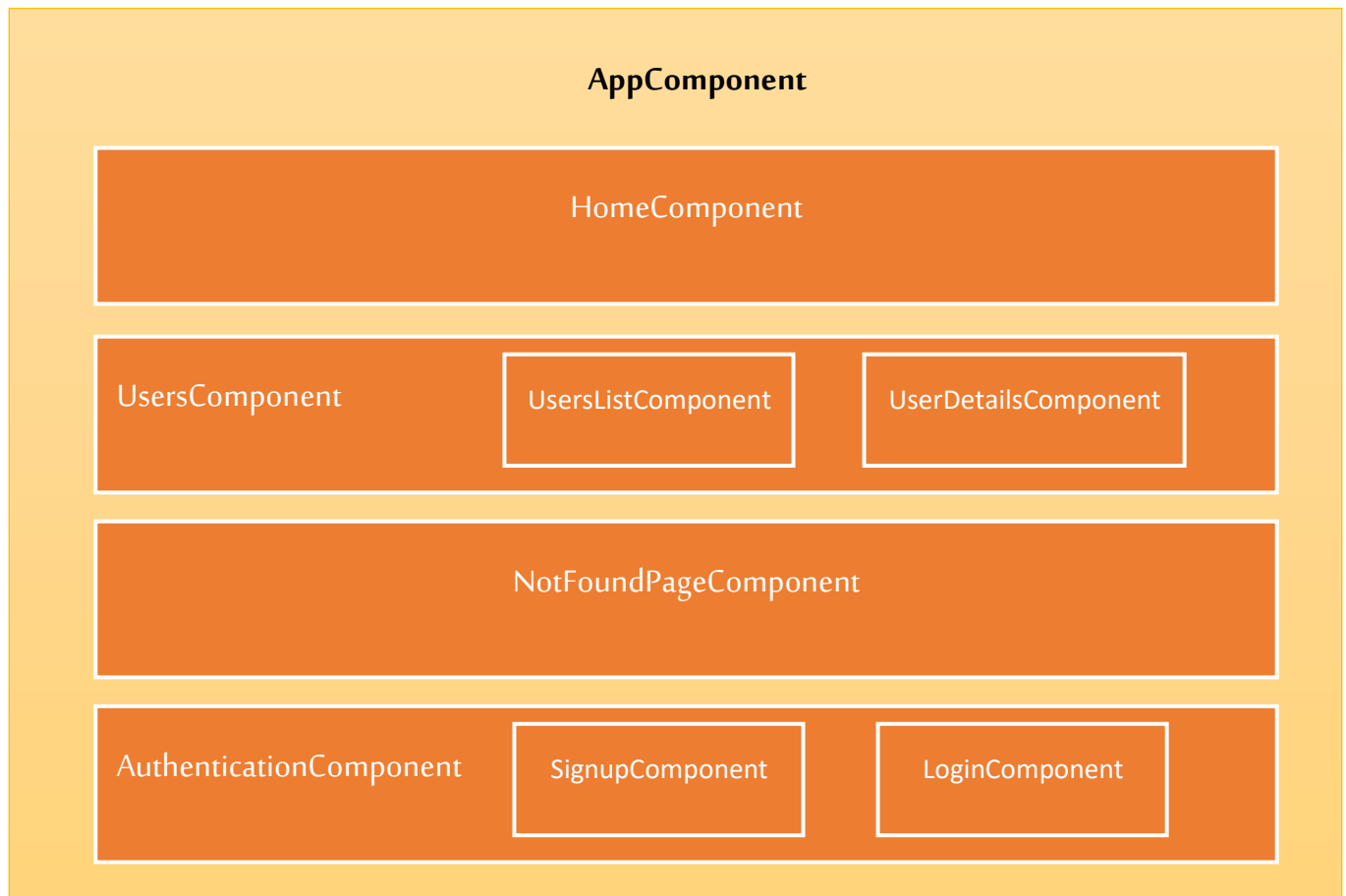


وبذلك نكون أنهينا هذا الجزء الذي انا اعتبره مراجعة سريعة لبعض أسس إطار عمل angular وليس له دخل في routing وانما أحببت أن اذكر بعض الأجزاء المهمة وفي حدوث أي لبس أو تشويش لك عزيزي القارئ المتعلم فلا تقلق أكمل تعلمك من خلال هذا الكتاب لأنه مخصص لتعلم angular routing والآن فلنستعد للجزء التالي وهو كيفية إنشاء children routes وما يحتويه من مفاهيم.

## 7.1.Children Routes:

في تطبيقات الويب الواقعية وخصوصاً المتوسطة منها او الكبيرة لا يكون بناء components كما فعلنا سابقاً، بمعنى لا تكون منفصلة وإنما متداخلة مع بعضها البعض، فمثلاً لدينا UsersComponent وهذا component نستطيع تقسيمه إلى عدة components فرعية تابعة له فنستطيع مثلاً إنشاء component لعرض قائمة المستخدمين فقط و component آخر لعرض تفاصيل كل مستخدم، وبعض هذه components لا تسمح للمستخدم بالوصول لها إلى عن طريق component آخر بمعنى component الفرعي الخاص بعرض تفاصيل بيانات مستخدم معين لا يستطيع الزائر لتطبيقك الوصول إليه إلا إذا دخل في البداية على component الذي يحتوي على قائمة المستخدمين جميعهم ومن ثم يضغط على احد المستخدمين ومنها ينتقل إلى component الذي قلنا عنه سابقاً وهو component الذي يعرض تفاصيل هذا المستخدم، وهنالك components أخرى قد نقوم بتقسيمها تحت component واحد لأنها ذات مجال واحد بمعنى أن هذه components تكون لها روابط ظاهرة في الصفحة الرئيسية للموقع ويستطيع المستخدم لتطبيقك الانتقال بينها بشكل منفصل بحيث لكل component رابط خاص به ولكن بسبب أنها ذات مجال واحد فنقوم بتجميع هذه components تحت مظلة component آخر (وهذا ينفعنا في Lazy Loading التي سوف نتكلم عنها بإذن الله في الأجزاء القادمة)، فمثلاً نريد أن ننشأ component خاص بتسجيل الدخول للمستخدم و component آخر خاص بالتسجيل في الموقع وبما أن هذه components ذات مجال واحد فسوف نقوم بجعلهم فرعيان ل component آخر ننشأه ويكون بمثابة الأب أو الحاوية لهذه components، ويمكن تمثيل هذا components في مثالنا بالشكل التالي:





ومما سبق يتضح أن صفحة التطبيق الواحدة قد تحتوي على أكثر من component فقد يقوم بعض المطورين بتقسيم حتى القائمة التي تعرض المستخدمين إلى أكثر من component فقد يقوم بجعل أسماء المستخدمين في component والأزرار الخاصة بهم في component آخر وهكذا، ولكن نحن هنا لن نغوص في هذه الجزئية وذلك لكيلا نحيد عن أساس هذا الكتاب وهو angular routing ولفهم أعمق في Components الرجاء مراجعة الكتاب الثاني من هذه السلسلة.

فما قدمناه من شرح موجز ومقدمة مقتضبة عن طريقة بناء Components في Angular مهم ويتعلق في Angular Routing حيث أن أي تعديل في components سوف يؤثر بطبيعة الحال على routing وطريقة بنائنا له، والـ components المتداخلة كما في مثالنا السابق يقابلها في routing ما يسمى Children Routes وهي طريقة تتيح لنا بناء وتهيئة routes فرعية لـ routed رئيسي، ويمكن بناء هذه الأمر بكل بساطة عن طريق استخدام الخاصية children.

أما الآن لنقوم بإنشاء components السابقة بنفس الأسماء الموجودة في الشكل السابق، وذلك كما نعمل دائماً عن طريق terminal، ومن ثم كتابة الأوامر التالية:

```
ng g c users/UsersList
```

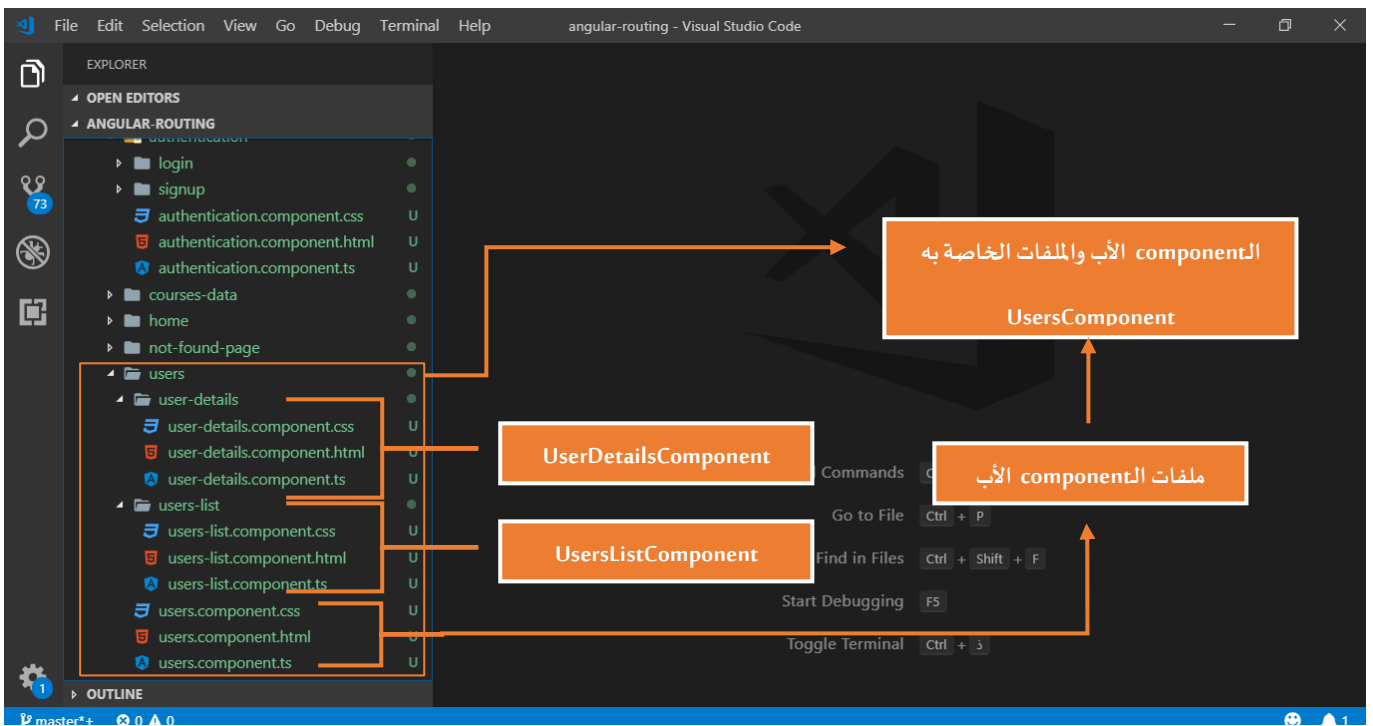
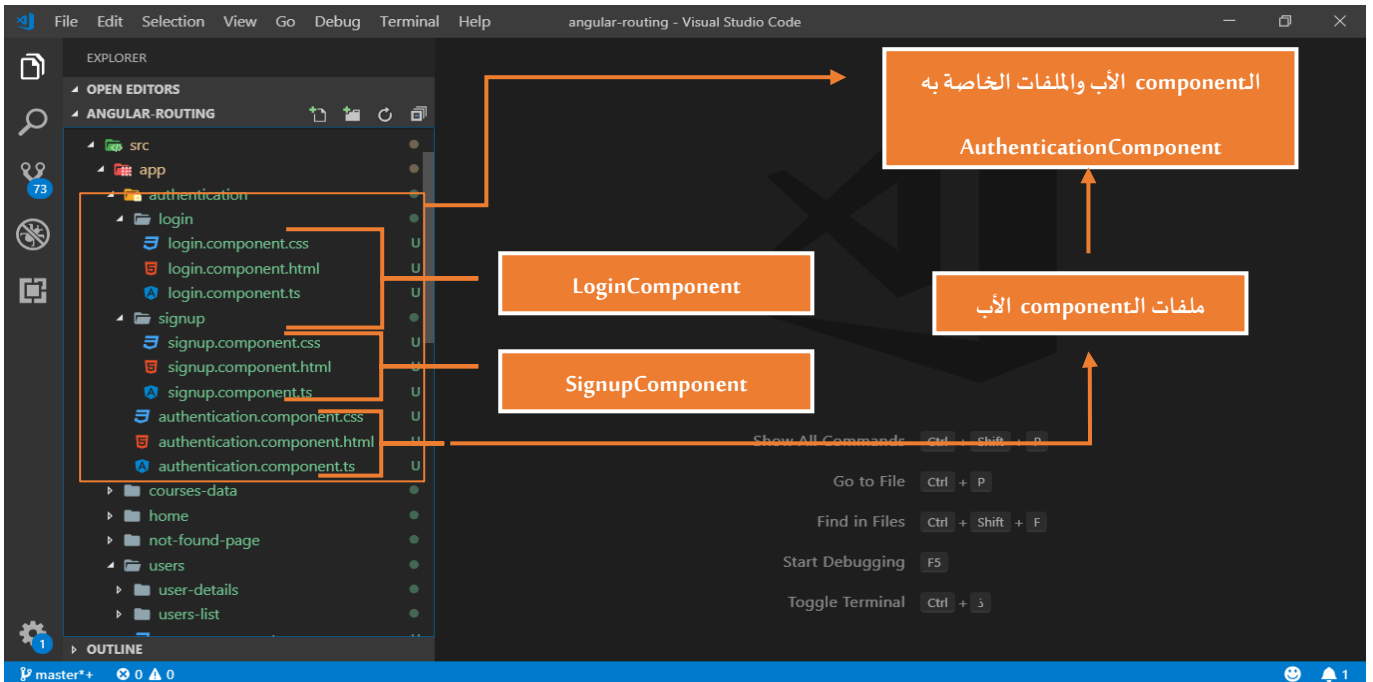
```
ng g c users/UserDetails
```

```
ng g c authentication
```

```
ng g c authentication/signup
```

```
ng g c authentication/login
```

وبعد كتابة الأوامر السابقة وإنشاء الملفات فسوف تكون بالشكل التالي:



بعد الانتهاء من انشاء الملفات السابقة، نقوم بتنفيذ الخطوات التالية:

الخطوة الأولى: نذهب إلى ملف `app-routing.module.ts` ونعيد تهيئة Routing وفق المعطيات الجديدة، كالتالي:

## app-routing.module.ts ملف

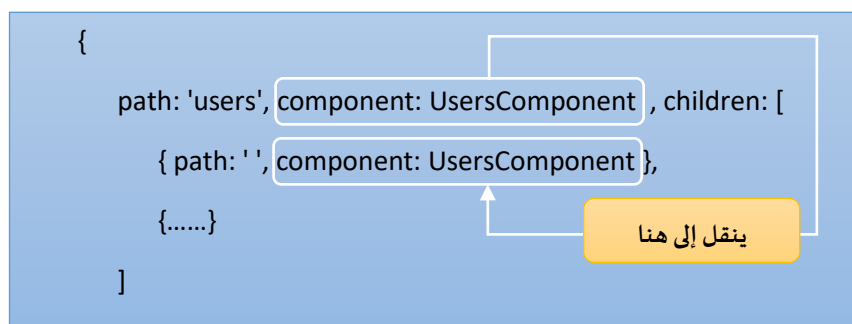
```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { HomeComponent } from './home/home.component';
4. import { UsersComponent } from './users/users.component';
5. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
6. import { AuthenticationComponent } from './authentication/authentication.component';
```

```

7. import { UsersListComponent } from './users/users-list/users-list.component';
8. import { UserDetailsComponent } from './users/user-details/user-details.component';
9. import { LoginComponent } from './authentication/login/login.component';
10. import { SignupComponent } from './authentication/signup/signup.component';
11.
12. const routes: Routes = [
13.   { path: 'home', component: HomeComponent },
14.   {
15.     path: 'users', children: [
16.       { path: '', component: UsersComponent },
17.       { path: 'user-list', component: UsersListComponent },
18.       { path: 'user-details', component: UserDetailsComponent }
19.     ]
20.   },
21.   {
22.     path: 'auth', children: [
23.       { path: '', component: AuthenticationComponent },
24.       { path: 'login', component: LoginComponent },
25.       { path: 'signup', component: SignupComponent }
26.     ]
27.   },
28.   { path: '', redirectTo: 'home', pathMatch: 'full' },
29.   { path: '**', component: NotFoundPageComponent }
30. ];
31.
32. @NgModule({
33.   imports: [RouterModule.forRoot(routes)],
34.   exports: [RouterModule]
35. })
36.
37. export class AppRoutingModule { }
38.

```

نلاحظ أننا قمنا بإعادة تهيئة routing بما يتناسب مع التعديلات الجديدة في components ويتضح ذلك في route ذو path الذي يحمل القيمة users في السطر 15 حيث أنشأنا له مجموعة من الأبناء أو ما يسمى Children Routes وذلك عن طريق إضافة الخاصية children له وهذه الخاصية هي أيضاً عبارة عن مصفوفة من الكائنات حيث يمثل كل كائن route أبن لل route الأب users، حيث route الأول يمثل root للأب users ويفضل وضعه في route فرعي، كما في الشكل التوضيحي التالي:



وفي حال تركه كما هو فسوف يعمل routing ولن تحدث مشاكل ولكن الأفضل نعمله كما في الشكل السابق.

اما routes الفرعية الأخرى (الأبناء) في الأسطر 17 و18 فهما UsersListComponent و UserDetailsComponent لكي نعمل لها انتقال أو ما يسمى Navigation.

أما route الثاني في الأسطر من 22 إلى 27 فقد أنشأنا route جديد باسم auth وأنشأنا له Children Routes وهو مشابه لما قمنا به في users.

الخطوة الثانية: نقوم بعملية الربط وسوف نبدأ auth لأنه مشابه لما قمنا به سابقاً، لذلك فلنذهب إلى ملف html AppComponent ونضيف رابطين الأول لاستعراض محتويات LoginComponent والثاني لاستعراض محتويات SignupComponent، كالتالي:

ملف app.component.html

```
1. <nav class="navbar navbar-expand-sm navbar-dark bg-dark Shadow">
2.   <div class="navbar-brand-two" href="#">
3.     
4.   </div>
5.   <span class="navbar-brand">Routing</span>
6.   <div class="justify-content-left">
7.     <ul class="navbar-nav">
8.       <li class="nav-item">
9.         <a routerLink="/home" routerLinkActive="is-active"><span>Home</span></a>
10.      </li>
11.      <li class="nav-item">
12.        <a routerLink="/users" routerLinkActive="is-active"><span>Users</span></a>
13.      </li>
14.      <li class="nav-item">
15.        <a routerLink="/auth/login" routerLinkActive="is-active"><span>Login</span></a>
16.      </li>
17.      <li class="nav-item">
18.        <a routerLink="/auth/signup" routerLinkActive="is-active"><span>Signup</span></a>
19.      </li>
20.    </ul>
21.  </div>
22.</nav>
23.
24.<router-outlet></router-outlet>
```

راجع السطر 12 والسطر 15.

ملاحظة: إذا اردت الربط البرمجي فنكتب:

```
this.router.navigate(['/auth/signup])
```

OR

```
this.router.navigate(['/auth/login])
```

ومن ثم نقوم بتصميم الصفحات لكلاً من LoginComponent و SignupComponent، كالتالي:

#### ملف login.component.html

```
1. <div class="card text-white bg-secondary mt">
2.   <div class="card-header text-center">Login</div>
3.   <div class="card-body">
4.     <div class="form-group">
5.       <input class="form-control" placeholder="User Name Here" />
6.     </div>
7.     <div class="form-group">
8.       <input class="form-control" placeholder="Password Here" />
9.     </div>
10.  </div>
11.  <div class="card-footer">
12.    <button class="btn btn-light form-control">Submit</button>
13.  </div>
14.</div>
```

#### ملف login.component.css

```
1. .mt{
2.   margin-top: 50px;
3.   width: 22rem;
4.   left: 30%;
5. }
```

#### ملف signup.component.html

```
1. <div class="card text-white bg-secondary mt">
2.   <div class="card-header text-center">Signup</div>
3.   <div class="card-body">
4.     <div class="form-group">
5.       <input class="form-control" placeholder="Enter Name Here" />
6.     </div>
7.     <div class="form-group">
8.       <input class="form-control" placeholder="Enter User Name Here" />
9.     </div>
10.    <div class="form-group">
11.      <input class="form-control" placeholder="Enter Password Here" />
12.    </div>
13.    <div class="form-group">
14.      <input class="form-control" placeholder="Enter City Here" />
15.    </div>
16.  </div>
17.  <div class="card-footer">
18.    <button class="btn btn-light form-control">Submit</button>
19.  </div>
20.</div>
```

#### ملف signup.component.css

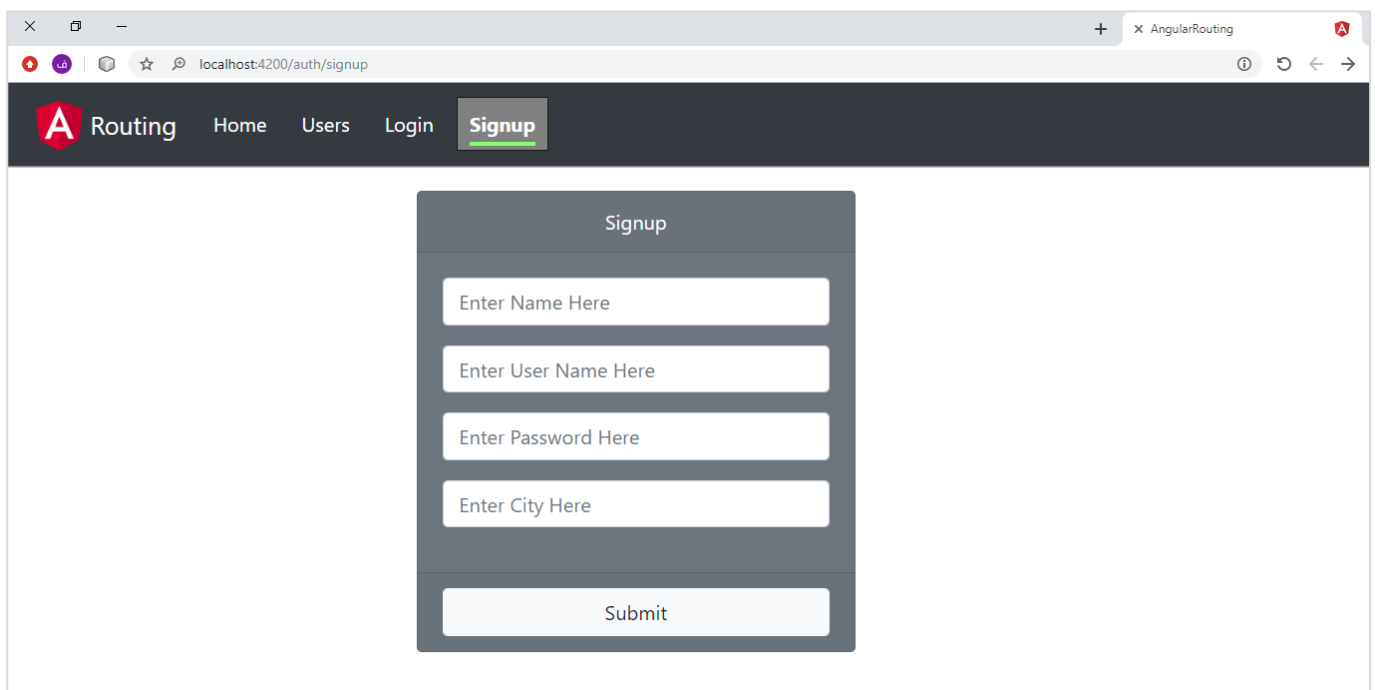
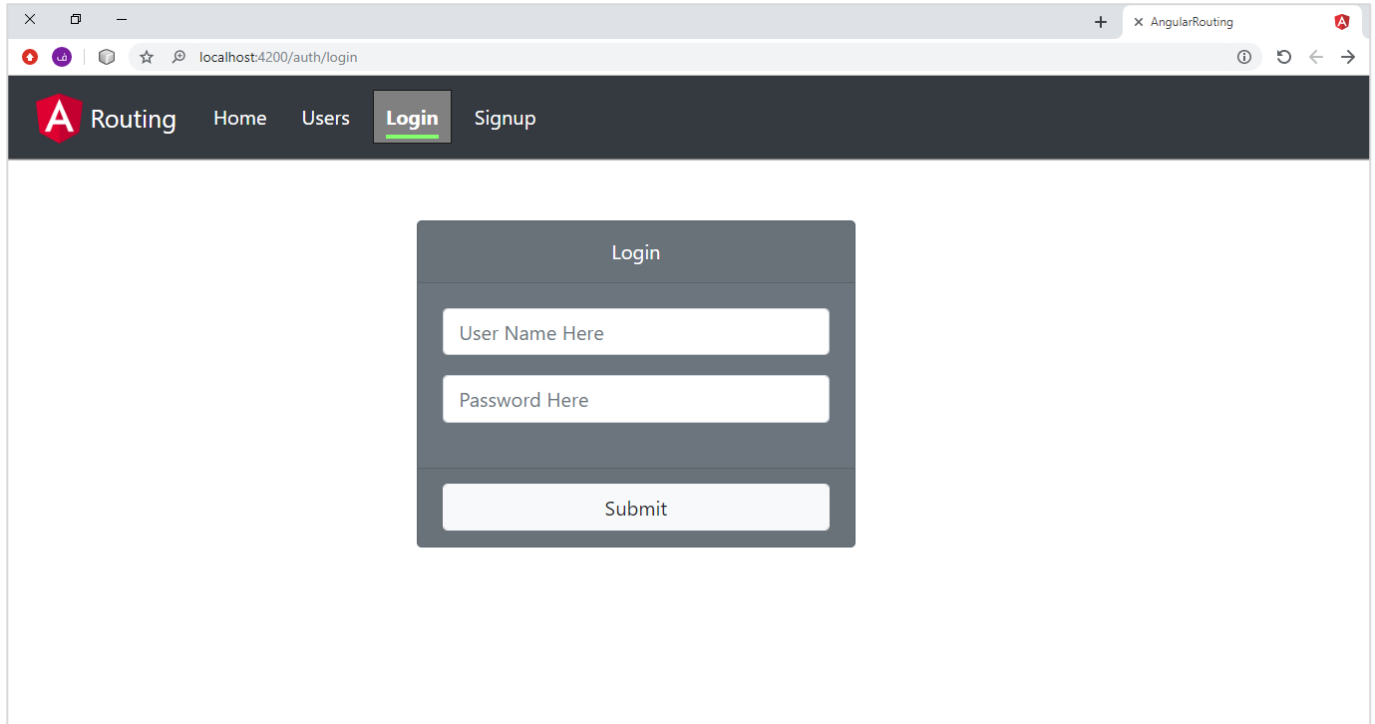
```
1. .mt{
2.   margin-top: 20px;
3.   width: 22rem;
4.   left: 30%;
5. }
```

وأخيراً نذهب إلى ملف html او ما يسمى ملف Template لـ AuthenticationComponent ونضيف له <router-outlet> لأنه أب ويحتوي بداخله مجموعة فرعية من components (أبناء) ونريد أن تظهر وتختفي هذه components الأبناء بداخله بشكل ديناميكي، كالتالي:

ملف Authentication.component.html

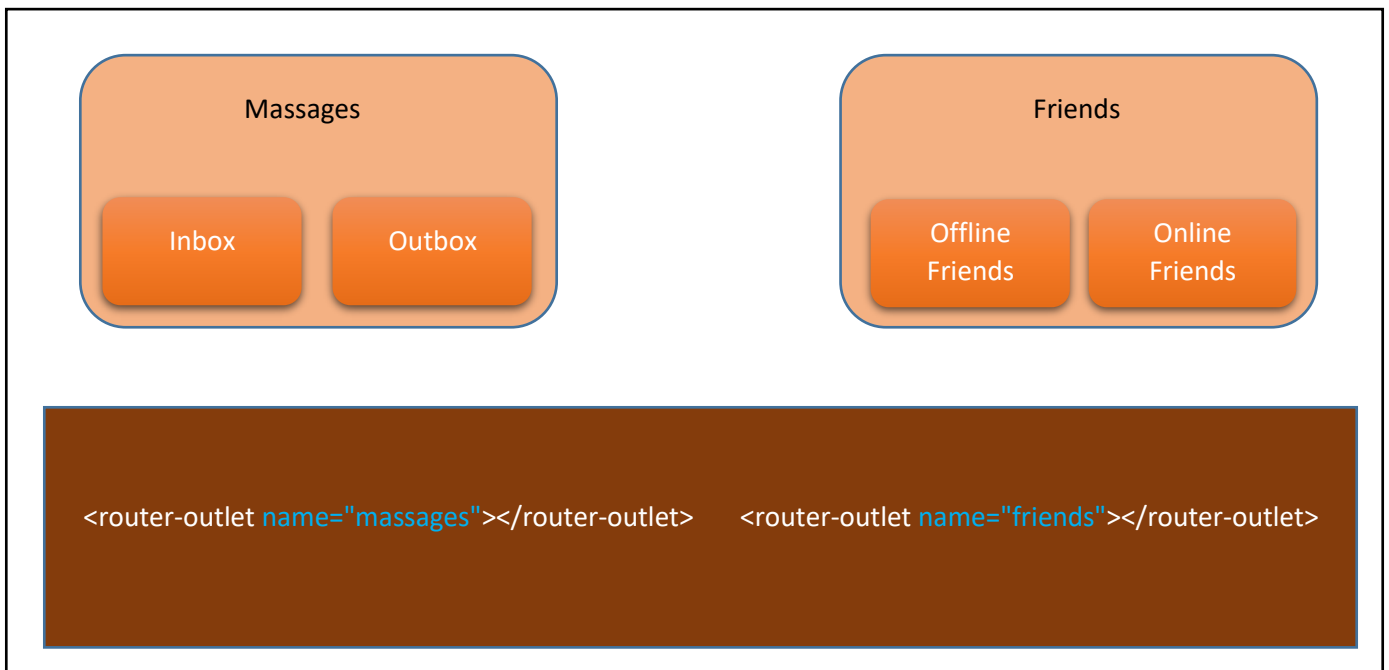
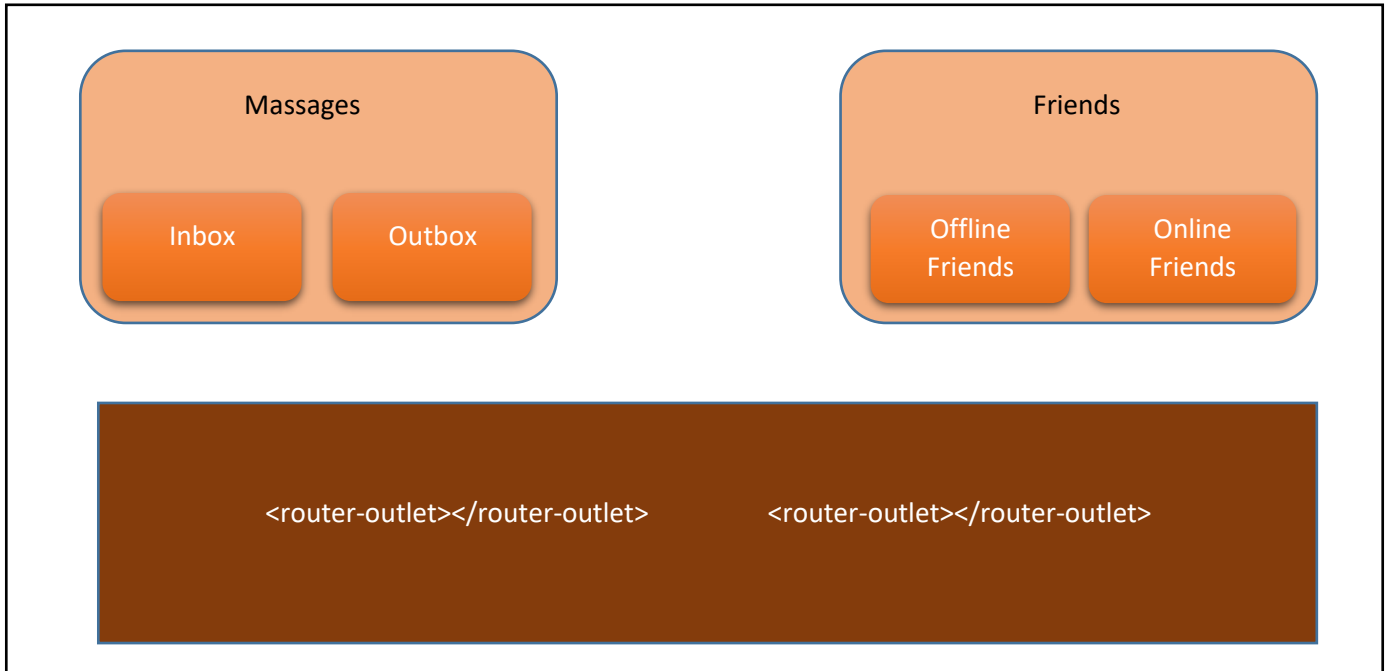
```
1. <router-outlet></router-outlet>  
2.
```

الآن لنحفظ التعديلات ونذهب إلى المتصفح ونرى نتيجة ما قمنا به، كالتالي:



نلاحظ أن routing يعمل بشكل سليم، ويستعرض components الجديدة بدون أي مشاكل.

ولكن قبل أن ننتقل إلى الخطوة الثالثة، الـ angular دكي ويعرف أن هذا <router-outlet> خاص بالأبناء ولكن ماذا لو كان لدينا صفحة أكثر تعقيداً وفي هذه الصفحة احتجنا أن يكون فيها أكثر من <router-outlet> كأن يكون لدينا صفحة تعرض فيها الرسائل الخاصة وفي جزء آخر منها يتم عرض المحادثات وفي جزء آخر أيضاً يتم عرض الأصدقاء، ...الخ، مشابهه تقريبا إلى موقع الفيسبوك، بحيث يمكن أن نمثله بالشكل التالي:

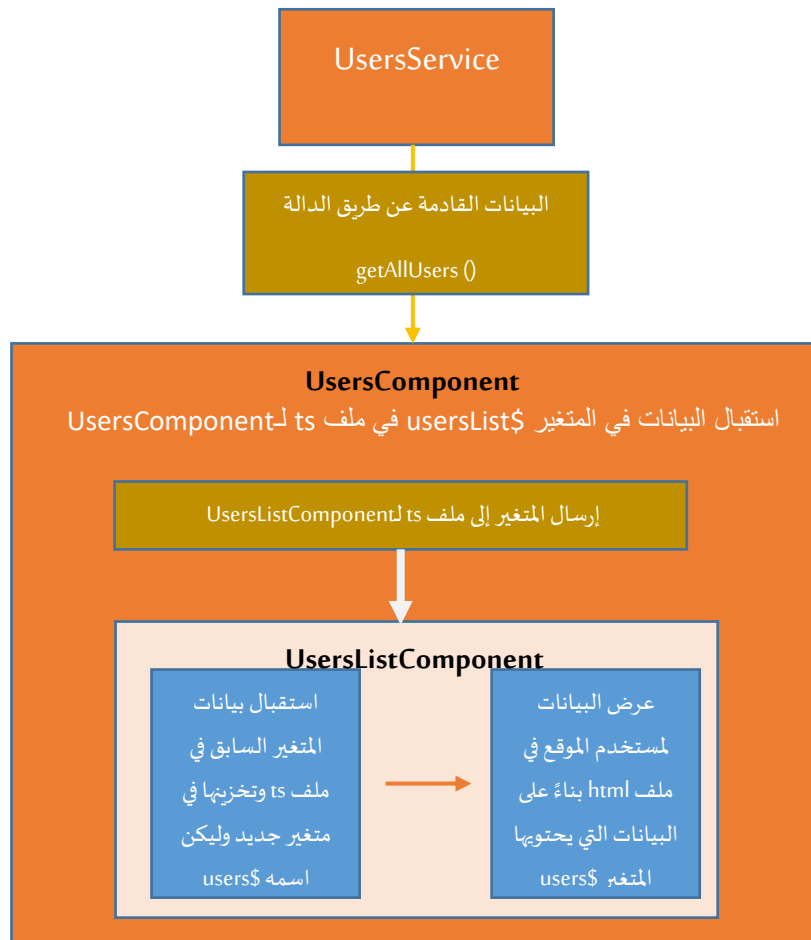


كما هو موضح في الشكل السابق لدينا صفحة رئيسية تعرض بداخلها صفحتين فرعيتين (أبناء) وكل صفحة فرعية محتواها يتغير بشكل ديناميكي عند الضغط على كل زر، في هذه الحالة لا بد أن نضيف اسم لكل <router-outlet> (لك حرية اختيار الاسم الذي تريده) وعند التهيئة نربط هذا path مع اسم <router-outlet>، كالتالي:

```
{path: 'social', children: [
  { path: '', component: SocialComponent },
  { path: 'messages-path', outlet: 'messages', component: MessagesComponent },
  { path: 'friends-path', outlet: 'friends', FriendsComponent }
]}
```

الخطوة الثالثة: نستكمل الربط ولكن المرة هذه لي users، لكن قبل شرح عملية الربط لابد أن نراجع كيف UsersComponent يستقبل بيانات المستخدمين في ملف ts الخاص به ويعرضها في ملف html الخاص به أيضاً، وتتم هذه الطريقة عن طريق استقبال البيانات من service ذات الاسم users.service.ts متمثلة بالدالة getAllUsers() ويخزنها في المتغير userList\$ ومن ثم في ملف html الخاص بهذا component عملنا حلقة Loop باستخدام \*ngFor على جميع البيانات التي يحتويها هذا المتغير وعرضنا هذه البيانات للمستخدم.

ولكن الآن أصبح هذه component يحتوي على Children Components وكما قلنا سابقاً أن component الابن ذو الاسم UsersListComponent هو الذي نريد منه أن يقوم بعرض هذه البيانات في ملف html الخاص به، وللقيام بذلك هنالك عدة طرق منها أن نترك UsersComponent على حاله يستقبل البيانات من service ويخزنها في المتغير userList\$ ولكن سوف نمرر هذه المتغير إلى الابن UsersListComponent ونجعله هو الذي يقرأ ويعرض محتوى هذه البيانات في ملف html الخاص به، ويمكن تمثيل هذه العملية بالشكل التالي:





وهذه الآلية هي ما تسمى بعالم Angular بي Component Communication ولفهم أعمق الرجاء مراجعة الكتاب الثاني من هذه السلسلة Angular Component and Services وبالتحديد الفصل الثاني.

بعدما قمنا باستعراض آلية العمل لنقوم الآن بتطبيقها عملياً:

ملف `users-list.component.ts`

```
1. import { Component, OnInit, Input } from '@angular/core';
2. import { Observable } from 'rxjs';
3. import { Users } from 'src/app/users-data/users';
4.
5. @Component({
6.   selector: 'app-users-list',
7.   templateUrl: './users-list.component.html',
8.   styleUrls: ['./users-list.component.css']
9. })
10.
11. export class UsersListComponent implements OnInit {
12.   @Input() users$: Observable<Users[]>;
13.
14.   constructor() { }
15.
16.   ngOnInit() {
17.
18.   }
19. }
```

في السطر 12 عرفنا متغير لكي يستقبل البيانات الجديدة من المتغير `usersList$` الموجود في الـ `component` الأب وأضافناه إلى `Decorator` ذو الاسم `@Input` لكي نخبر angular أن هذه المتغير يمثل قيمة آتية له من الأب `UsersComponent`.

ملف `users-list.component.html`

```
1. <div class="container" style="padding-top: 1rem">
2.   <ul class="list-group" *ngFor="let user of users$ | async">
3.     <li class="list-group-item">
4.       <span class="pull-left ">
5.         
11.       </span>
12.       <h6 style="padding-left: 2rem; display: inline">{{ user.name }}</h6>
13.       <span class="pull-right">
14.         <button class="btn btn-info" style="display: inline">View</button>
15.       </span>
16.     </li>
17.   </ul>
18. </div>
```

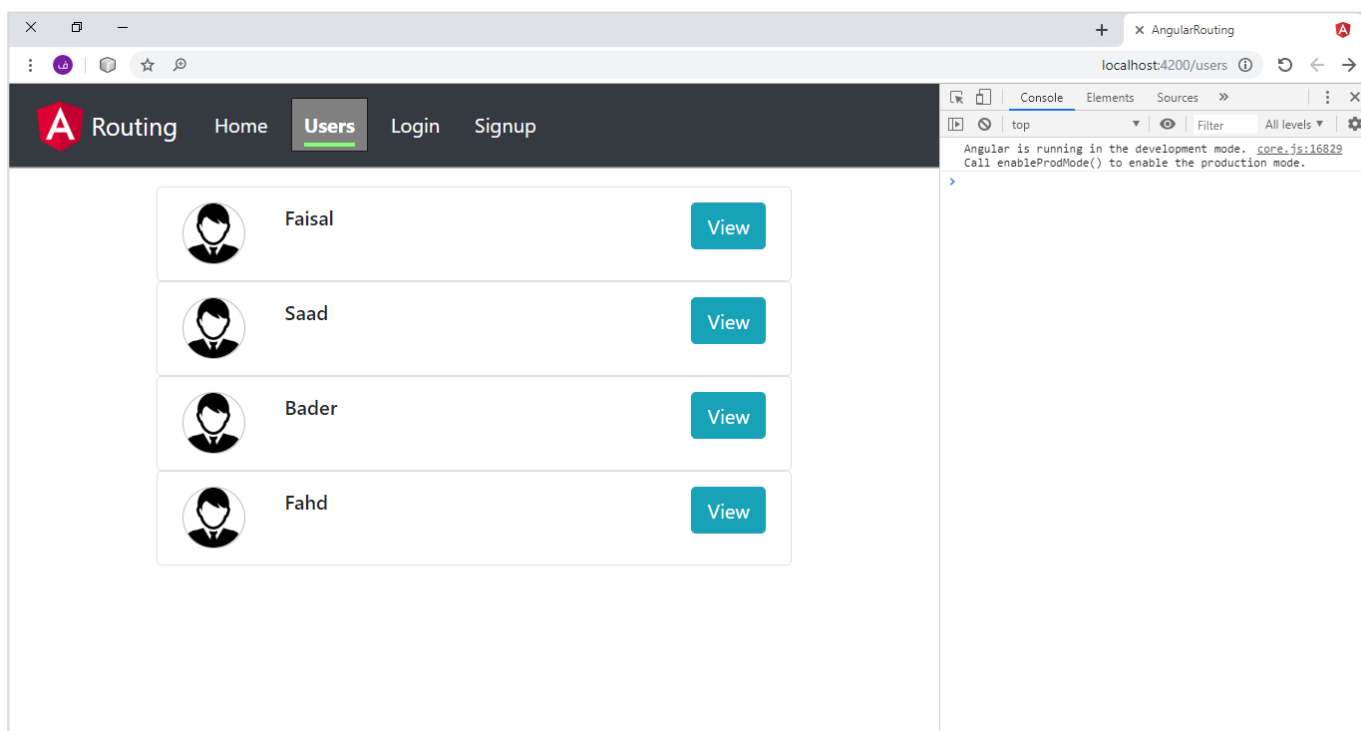
هنا قمنا بنقل جميع محتويات ملف Template لـ UsersComponent ووضعناها في ملف Template لـ UsersListComponent والذي قمنا بتغييره فقط هو المتغير الذي نقرأ منه البيانات والذي في حالتنا هنا هو users\$ بدلاً من userList\$ كما في السطر 2.

بقي أن نقوم بإرسال قيمة المتغير userList\$ من الأب UsersComponent إلى الأبن UsersListComponent وتتم هذه الطريقة بوضع العنصر الخاص بالcomponent الأب في ملف html لـ component الأب، ومن ثم نربط المتغيرين مع بعضهما البعض، وفي حال عدم معرفتك بمكان العنصر لو ذهبت إلى ملف users-list.component.ts وفي السطر 6 سوف تجد هذا العنصر لـ UsersListComponent والذي هو app-users-list.

ملف users.component.html

```
1. <app-users-list [users$]="usersList$" > </app-users-list>
2.
```

هنا قمنا بإضافة العنصر وربط المتغيرين، الآن لنقوم بحفظ التعديلات والذهاب للمتصفح لمشاهدة النتيجة:



نلاحظ أنه تم عرض البيانات بدون أي مشاكل.

وكنوع من التدريب سوف نُزيد الجرعة قليلاً (ما سوف يقال الآن ليس له علاقة بي Angular Routing وانما سوف نستخدمه لشرح بعض المفاهيم المتعلقة بي Angular Routing لاحقاً في هذا الكتاب)، حيث نريد أن نضيف مربع للبحث بحيث إذا قام المستخدم بكتابة أسماء المستخدمين نريد أن يعمل filtering لهذه القائمة بحسب القيمة التي يُدخلها المستخدم في مربع البحث، مع العمل أن مربع البحث نريده أن يكون موجود في UsersComponent الأب ويتواصل مع UsersListComponent كما فعلنا سابقاً من خلال تمرير القيم بينهما، حاول أن تجد حل لهذا التدريب وإذا لم تستطع فلا تقلق أنتقل معنا إلى الصفحة القادمة وسوف تجد الحل بإذن الله.

وحل هذا التدريب (ليس شرطاً أن يكون هو أفضل الحلول فقد تكون حللته عزيبي المتعلم بطريقة أفضل مما سوف أقدمه لك لأن هذا الحل هو ببساطة logic او تفكير منطقي وليس له قواعد ثابتة) هو مشابه لما قمنا به سابقاً ولكن هنا بشكل أوسع، حيث سوف نقوم بملف ts لـ UsersComponent بتعريف متغيرين بالإضافة إلى المتغير السابق \$usersList ومن ثم إرسالهما إلى UsersListComponent الأب، وليكن أسماء هذه المتغيرات، كالتالي:

inputSearch: ومهمة هذا المتغير هو قراءة البيانات من مربع الإدخال الخاص بالبحث عن طريق ربط هذا المتغير في مربع الإدخال باستخدام تقنية Tow-Way-Databinding او ما يسمى [(ngModel)] ولكن يجب استدعاء FormsModule وإضافتها في مصفوفة imports في ملف app.module.ts،

fullUsersList\$: وهذا المتغير يقوم بنفس مهمة المتغير الآخر \$usersList وتم إضافة هذا المتغير لأنه في حال قام المستخدم بالبحث عن مستخدم معين في مربع البحث ثم قام ومسح الاسم الذي كتبه فإن قائمة جميع المستخدمين لن تظهر لذلك عرفنا هذا المتغير بحيث يحتوي على جميع البيانات وفي حال كانت القائمة فارغة يقوم بإعادة تعبئتها مرة أخرى.

أما الآن لنقوم بتطبيق عملية الربط هذه عملياً، وأول ما سوف نقوم به هو استدعاء FormsModule (في حال عدم معرفتك بتقنية تعامل angular مع النماذج forms الرجاء مراجعة الكتاب الخامس من هذه السلسلة Angular Forms).

المهم لنقوم الآن بالذهاب إلى ملف app.module.ts، وإضافة التعديلات التالية:

#### ملف app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { FormsModule } from '@angular/forms';
4.
5. import { AppRoutingModule } from './app-routing.module';
6. import { AppComponent } from './app.component';
7. import { HomeComponent } from './home/home.component';
8. import { UsersComponent } from './users/users.component';
9. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
10. import { UsersListComponent } from './users/users-list/users-list.component';
11. import { UserDetailsComponent } from './users/user-details/user-details.component';
12. import { AuthenticationComponent } from './authentication/authentication.component';
13. import { SignupComponent } from './authentication/signup/signup.component';
14. import { LoginComponent } from './authentication/login/login.component';
15.
16. @NgModule({
17.   declarations: [
18.     AppComponent,
19.     HomeComponent,
20.     UsersComponent,
21.     NotFoundPageComponent,
22.     UsersListComponent,
23.     UserDetailsComponent,
24.     AuthenticationComponent,
25.     SignupComponent,
26.     LoginComponent
```

```

27. ],
28. imports: [
29.   BrowserModule,
30.   AppRoutingModule,
31.   FormsModule
32. ],
33. providers: [],
34. bootstrap: [AppComponent]
35.})
36.export class AppModule { }

```

بعد الانتهاء من إضافة FormsModule في ملف app.module.ts، لنذهب إلى ملف ts لـ UsersComponent ونعرف المتغيرات، ونقرأ البيانات من service ذات الاسم UsersService، كالتالي:

#### ملف users.component.ts

```

1. import { Component, OnInit } from '@angular/core';
2. import { Observable } from 'rxjs';
3. import { Users } from '../users-data/users';
4. import { UsersService } from '../users-data/users.service';
5.
6. @Component({
7.   selector: 'app-users',
8.   templateUrl: './users.component.html',
9.   styleUrls: ['./users.component.css']
10.})
11.
12.export class UsersComponent implements OnInit {
13.   userList$: Observable<Users[]>;
14.   fullUsersList$: Observable<Users[]>;
15.   inputSearch: string;
16.
17.   constructor(private users: UsersService) { }
18.
19.   ngOnInit() {
20.     this.userList$ = this.users.getAllUsers();
21.     this.fullUsersList$ = this.users.getAllUsers();
22.   }
23.
24.}

```

بعدها نذهب إلى ملف html لنفس component ونربط المتغير inputSearch في مربع الإدخال عن طريق [(ngModel)] ونقوم أيضاً بتمرير هذان المتغيران من UsersComponent الأب إلى UsersListComponent الابن كما فعلنا سابقاً مع المتغير userList\$ والفرق الوحيد هنا أنني اخترت نفس الأسماء لكلا المتغيرات سواء في الأب أو الابن، كالتالي:

#### ملف users.component.html

```

1. <div class="container">
2.   <input
3.     class="form-control"
4.     [(ngModel)]="inputSearch"

```

```

5.     placeholder="Search Users Here"
6.   />
7. </div>
8.
9. <app-users-list
10.   [users$]="usersList$"
11.   [inputSearch]="inputSearch"
12.   [fullUsersList$]="fullUsersList$"
13.</app-users-list>

```

نذهب الآن إلى ملف ts او class لـ UsersListComponent الأبن ونعرف متغيرات بنفس اسم المتغيرات القادمة له كما هو موضح في الملف السابق حيث كتبنا المتغيرات في الأقواس المربعة Property Binding ولكن لم ننشئها إلى الآن لذلك لنقوم بإنشائها لتستقبل قيم المتغيرات الجديدة الداخلة لهذا component، كالتالي:

ملف users-list.component.ts

```

1. import { Component, OnInit, Input } from '@angular/core';
2. import { Observable } from 'rxjs';
3. import { Users } from 'src/app/users-data/users';
4.
5. @Component({
6.   selector: 'app-users-list',
7.   templateUrl: './users-list.component.html',
8.   styleUrls: ['./users-list.component.css']
9. })
10.
11. export class UsersListComponent implements OnInit {
12.   @Input() users$: Observable<Users[]>;
13.   @Input() fullUsersList$: Observable<Users[]>;
14.   @Input() inputSearch: string;
15.
16.   constructor() { }
17.
18.   ngOnInit() {
19.
20.   }
21.
22. }

```

بعد استقبالنا لهذه البيانات أصبح الوضع مُبني لكتابة logic الخاص بالبحث في القائمة وعمل فلترة لهذه القائمة بناءً على ما يكتبه المستخدم في مربع الإدخال وللقيام بذلك سوف نستخدم الـ interface ذات الاسم OnChanges وهي من دوال LifeCycle Hooks ومهمتها مراقبة أي تغيير يحدث على قيمة المدخلات الداخلة إلى هذا Component الأبن عن طريق @Input، ولفهم أعمق تستطيع مراجعة الكتاب الثاني من هذه السلسلة:

ملف users-list.component.ts

```

1. import { Component, OnInit, Input, OnChanges } from '@angular/core';
2. import { Observable, of } from 'rxjs';
3. import { Users } from 'src/app/users-data/users';
4.

```

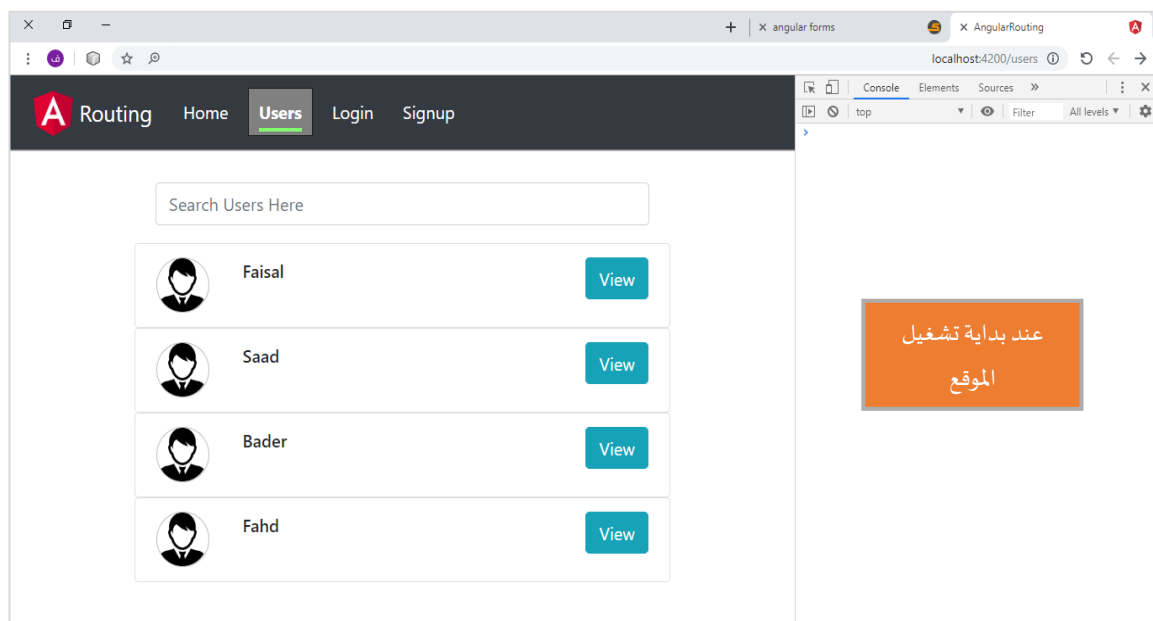
```

5. @Component({
6.   selector: 'app-users-list',
7.   templateUrl: './users-list.component.html',
8.   styleUrls: ['./users-list.component.css']
9. })
10.
11. export class UsersListComponent implements OnInit, OnChanges {
12.   @Input() users$: Observable<Users[]>;
13.   @Input() fullUsersList$: Observable<Users[]>;
14.   @Input() inputSearch: string;
15.
16.   constructor() { }
17.
18.   ngOnInit() {
19.
20.   }
21.
22.   ngOnChanges() {
23.     if (this.inputSearch) {
24.       this.users$.subscribe(data => {
25.         const result = data.filter(user =>
26.           user.name.toLocaleLowerCase().indexOf(this.inputSearch.toLocaleLowerCase()) !== -1);
27.         this.users$ = of(result);
28.       });
29.     } else {
30.       this.users$ = this.fullUsersList$;
31.     }
32.   }
33.
34. }

```

نلاحظ في السطر 1 استدعينا الـ interface ذات الاسم OnChanges وعملنا له implements في السطر 11 وكتبنا محتوى الدالة التي يحتويها هذا interface في الاسطر من 22 إلى 32، أما فكرة كود البحث فهو بسيط جداً فكل الذي عملناه أننا قمنا بعمل شرط لتأكد أن المتغير inputSearch يحتوي على قيمة في السطر 23 فإذا كان يحتوي على قيمة انتقل إلى السطر 24 وفي هذا السطر قمنا بعمل subscribe للمتغير users\$ والذي يحمل البيانات القادمة له من component الأب ومربوط أيضاً بملف html لـ UsersListComponent بحيث أي تغير في بيانات هذا المتغير سوف يؤثر على ملف html بمعنى هذا المتغير حالياً يحمل بيانات خمس مستخدمين ويعرض هذه البيانات في ملف html على شكل خمس قوائم فلو مثلاً قمنا بحذف احد المستخدمين من هذا المتغير فتلقائياً سوف يعيد تحديث القوائم في ملف html ويظهر اربع فقط للمستخدم، المهم بعدما نقوم بعمل subscribe لهذا المتغير سوف ينتج لنا مجموعة بيانات وقد خزنتها في متغير اسميته data وهذه البيانات هي نفسها البيانات الموجودة في المتغير users ولكن على شكل مصفوفة كائنات عادية من النوع Users[] وليست مصفوفة observable من النوع Users[] كما في المتغير users\$، والسبب من ذلك أننا نريد الاستفادة من الدوال الموجودة في المصفوفات العادية ومنها الدالة filter وهذا ما قمنا به في السطر 25 حيث قمنا بعمل فلتر للمتغير data باستخدام الدالة filter وهذه الدالة تقوم بعمل حلقة Loop على جميع البيانات في مصفوفة الكائنات (المتغير) data وفي كل loop تُخزن بيانات كل كائن في المصفوفة data في متغير واسميت هذا المتغير user بمعنى في البداية تأخذ بيانات

المستخدم الأول في مصفوفة الكائنات data وتُخزن أول كائن في المتغير user ومن ثم في loop الثانية تخزن بيانات المستخدم الثاني وتخزنه في المتغير user مع حذف بيانات المستخدم الأول (الكائن الأول) وهكذا، أما الذي تعمله في كل Loop فهو ما هو موجود في السطر 26 حيث عن طريق الكائن (المتغير) user وصلنا إلى الخاصية name التي تحتوي اسم المستخدم وبعد تحويلها إلى حروف صغيرة استخدمنا الدالة indexOf() التي مررنا لها قيمة المتغير inputSearch وبذلك نتأكد هل القيمة الموجودة في هذا المتغير inputSearch موجودة أيضاً في الخاصية name للكائن user فإذا كانت موجودة تُرجع هذا الكائن وتخزنه في ثابت اسميته result وإذا لم تحتويه لا يُرجع شيء، مع العلم أن الثابت result هو أيضاً مصفوفة كائنات، وبالنسبة للنتيجة النهائية أصبح هذا الثابت يحتوي على البيانات التي تحتوي على قيمة مشابهة لما هو موجود في المتغير inputSearch مع العلم أن السطر 25 والسطر 26 هما سطر واحد ولكن بسبب طوله قمت بوضعه بسطر جديد، وأخيراً نسند المتغير result إلى المتغير users\$ لكي يؤثر على الأكواد في ملف html ويظهر النتيجة كما قلنا سابقاً لكن يجب تحويل المتغير result إلى observable لأننا لا نستطيع اسناد مصفوفة إلى observable ونقوم بذلك عن طريق الدالة of() التي استدعيناها في السطر 2، وأخيراً أعدنا تعبئة المتغير users\$ في حال عدم وجود قيمة في inputSearch، كما في السطر 30، الآن بعد هذا الشرح المطول للكواد السابق لنحفظ التعديلات ونذهب إلى المتصفح لنرى النتيجة:



Routing Home **Users** Login Signup

d

	Saad	<a href="#">View</a>
	Bader	<a href="#">View</a>
	Fahd	<a href="#">View</a>

في حال وجود تطابق ونلاحظ أنه يبحث بالحرف في أي موقع بالكلمة وذلك بسبب استخدامنا للدالة `indexOf()`

Routing Home **Users** Login Signup

في حال عدم وجود تطابق وذلك يعني أن قيمة المتغير `result` فارغة وبالتالي قيمة `users$` فارغة وبشكل طبيعي سوف يؤثر هذا على صفحة `html` ولا تظهر القوائم لأنه لا يوجد بيانات

Routing Home **Users** Login Signup

Search Users Here

	Faisal	<a href="#">View</a>
	Saad	<a href="#">View</a>
	Bader	<a href="#">View</a>
	Fahd	<a href="#">View</a>

في حال قام المستخدم بحذف المدخلات تُعيد القائمة مرة أخرى وهو ما قمنا به في السطر 29 حيث أعدنا تعبئة البيانات في `users$` من البيانات الجاهزة في المتغير `fullUsersList$`



# الفصل الثاني

## Parameters in Angular Routing

## 1.2. مقدمة:

تعتبر البارامترات parameters من الأمور المهمة التي قدمها لنا angular routing وسهلت على المطورين بشكل كبير جداً، ومفهومها بشكل مبسط هي طريقة لتبادل البيانات بين components المختلفة (لذلك قد تجد بعض المراجع يضع هذه التقنيات من ضمن Angular Component Communication) سواء كانت هذه Children Components أو غير ذلك، وذلك عن طريق وضع هذه البيانات في الرابط اثناء التوجيه Navigation بين components، ومن أمثلة هذه البيانات على سبيل المثال عندما يقوم المستخدم بتسجيل الدخول عن طريقة صفحة تسجيل الدخول واثناء توجيهه إلى الصفحة الرئيسية نستطيع تحميل الرابط باسم المستخدم له ويتم قراءتها في الصفحة الرئيسية وإظهار البيانات الخاصة به فقط، والأمثلة على ذلك كثيرة جداً، كما أن لها أربعة أنواع هي:

- Required Parameters
- Optional Parameters
- Query Parameters
- Fragment Parameters

وسوف نتكلم عن هذه الأنواع الأربعة بشيء من التفصيل وتقنياتها وقواعدها وأمور أخرى.

## 2.2. Required Parameters:

ومن أسمها هي بارامترات مطلوبة بمعنى أن الصفحة التي وجهه لها المستخدم تعتمد في محتواها على هذه البارامترات لكي تعرض البيانات التي تحتويها، ومن امثلتها المشروع الذي نعمل عليه في هذا الكتاب، حيث لدينا صفحة لعرض قائمة بالمستخدمين وفي كل قائمة هنالك زر إذا ضغط عليه المستخدم يتم الانتقال إلى صفحة أخرى لعرض تفاصيل بيانات هذه المستخدم وهذه التفاصيل تعتمد لعرضها على اسم المستخدم أو id الخاص بالمستخدم لكي تعرض البيانات الخاصة به وهذا id يُرسل لها عن طريق الرابط فتقوم بقراءته ومن ثم عرض البيانات الخاصة به فقط، لذلك سُميت بيانات او بارامترات مطلوبة Required Parameters، وبما أنها مطلوبة فيجب تهيئتها في Routing الأساسي، وللقيام بذلك نذهب إلى ملف app-routing.module.ts وإضافة هذا البارامتر، كالتالي:

ملف app-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { HomeComponent } from './home/home.component';
4. import { UsersComponent } from './users/users.component';
5. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
6. import { AuthenticationComponent } from './authentication/authentication.component';
7. import { UsersListComponent } from './users/users-list/users-list.component';
8. import { UserDetailsComponent } from './users/user-details/user-details.component';
9. import { LoginComponent } from './authentication/login/login.component';
10. import { SignupComponent } from './authentication/signup/signup.component';
11.
12. const routes: Routes = [
```

```

13. { path: 'home', component: HomeComponent },
14. {
15.   path: 'users', children: [
16.     { path: '', component: UsersComponent },
17.     { path: 'user-list', component: UsersListComponent },
18.     { path: 'user-details/:id', component: UserDetailsComponent }
19.   ]
20. },
21. {
22.   path: 'auth', children: [
23.     { path: '', component: AuthenticationComponent },
24.     { path: 'login', component: LoginComponent },
25.     { path: 'signup', component: SignupComponent }
26.   ]
27. },
28. { path: '', redirectTo: 'home', pathMatch: 'full' },
29. { path: '**', component: NotFoundPageComponent }
30.];
31.
32.@NgModule({
33.  imports: [RouterModule.forRoot(routes)],
34.  exports: [RouterModule]
35.})
36.export class AppRoutingModule { }
37.

```

ما يهمنا ما هو موجود في السطر 18 حيث قمنا بعد "/" بكتابة نقطتين ثم اسم البارامتر ولك حرية اختيار أي اسم تريده ولكن الاسم الذي تختاره هنا لابد من التقيد فيه مستقبلاً، ومعنى هذا السطر أن UserDetailsComponent سوف يستقبل بارامتر ولا بد من إضافة هذه البارامتر الى الرابط URL عند توجيهنا للمستخدم إلى هذا component وإلا فلن يعمل، كما انه يمكن إضافة أكثر من بارامتر ونفصل بينهم بنقطتين فمثلاً يمكن كتابة 'user-details/:id/:name' وهكذا قم بإضافة البارامترات التي تريد بحسب احتياجك.

والآن بعدما قمنا بشرح كيفية تهيئة هذه البارامترات سوف ننتقل في الجزء التالي لشرح كيف يتم ربط وارسال هذه البارامترات عن طريق URL.

## 1.2.2. الربط وأرسال البارامترات عن طريق URL:

بعد تهيئة البارامتر في مصفوفة routing نذهب إلى UsersListComponent وبالتحديد إلى ملف html الخاص به لنقوم بعملية الربط وسوف أقوم هنا بالربط البرمجي (مع العلم أنني سوف أشير إلى كيفية الربط عن طريق template لكي يكون القارئ الكريم على اطلاع بجميع الطرق المختلفة) عن طريق انشاء دالة في الزر button وليكن اسم الدالة viewDetails() ونمرر لهذه الدالة id الخاص بالمستخدم والذي اسميناها userId راجع ملف Users.ts في حال اختلاف الأمر عليك، ومهمة هذه الدالة هي إضافة id الخاص بالمستخدم الى الرابط ومن ثم توجيه المستخدم إلى صفحة UserDetailsComponent، كالتالي:

## ملف user-details.component.html

```

1. <div class="container" style="padding-top: 1rem">
2.   <ul class="list-group" *ngFor="let user of users$ | async">
3.     <li class="list-group-item">
4.       <span class="pull-left ">
5.         
11.      </span>
12.      <h6 style="padding-left: 2rem; display: inline">{{ user.name }}</h6>
13.      <span class="pull-right">
14.        <button
15.          class="btn btn-info"
16.          style="display: inline"
17.          (click)="viewDetails(user.userId)">
18.            View
19.        </button>
20.      </span>
21.    </li>
22.  </ul>
23.</div>

```

نلاحظ في السطر 17 اضعفنا دالة ومررنا لها userId عن طريق الكائن user الذي تم تعريفه في حلقة Loop في السطر 2.

الآن لنذهب إلى ملف ts او بصيغة أخرى ملف class لهذا component ونكتب محتوى هذه الدالة، كالتالي:

## ملف user-details.component.ts

```

1. import { Component, OnInit, Input, OnChanges } from '@angular/core';
2. import { Observable, of } from 'rxjs';
3. import { Users } from 'src/app/users-data/users';
4. import { Router } from '@angular/router';
5.
6. @Component({
7.   selector: 'app-users-list',
8.   templateUrl: './users-list.component.html',
9.   styleUrls: ['./users-list.component.css']
10. })
11. export class UsersListComponent implements OnInit, OnChanges {
12.   @Input() users$: Observable<Users[]>;
13.   @Input() fullUsersList$: Observable<Users[]>;
14.   @Input() inputSearch: string;
15.
16.   constructor(private router: Router) { }
17.
18.   ngOnInit() {
19.
20.   }
21.

```

```

22. ngOnChanges() {
23.   if (this.inputSearch) {
24.     this.users$.subscribe(data => {
25.       const result = data.filter(user =>
26.         user.name.toLocaleLowerCase().indexOf(this.inputSearch.toLocaleLowerCase()) !== -1);
27.       this.users$ = of(result);
28.     });
29.   } else {
30.     this.users$ = this.fullUsersList$;
31.   }
32. }
33.
34. viewDetails(id: number) {
35.   this.router.navigate([' /users/user-details', id]);
36. }
37.

```

نلاحظ في السطر 35 كتبنا محتوى الدالة وهو مشابه لطريقة الربط البرمجي التي تعلمناها سابقاً والفرق الوحيد هنا أننا أضفنا المتغير id لها، مع العلم أنه يمكن كتابة دالة التوجيه بهذه الطريقة أو كتابتها بالطريقة التالية:

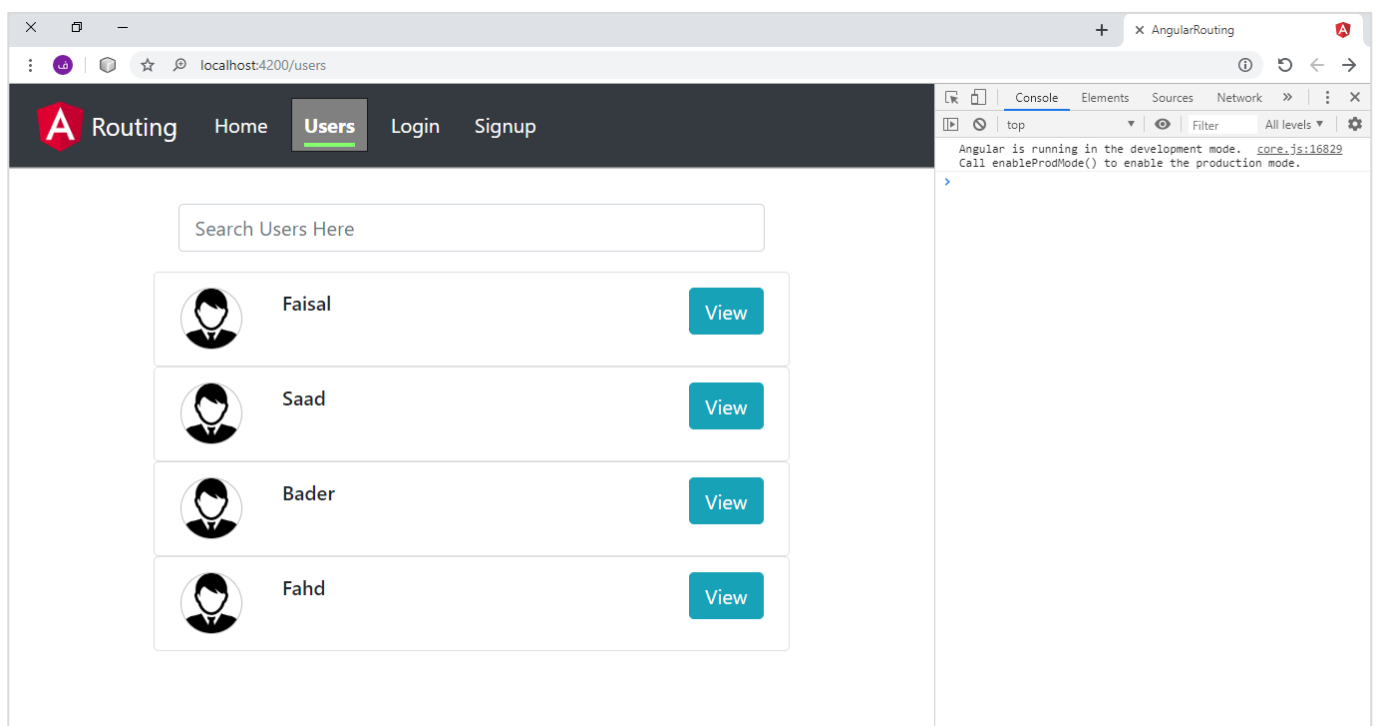
```
this.router.navigate(['users', 'user-details', id]);
```

OR

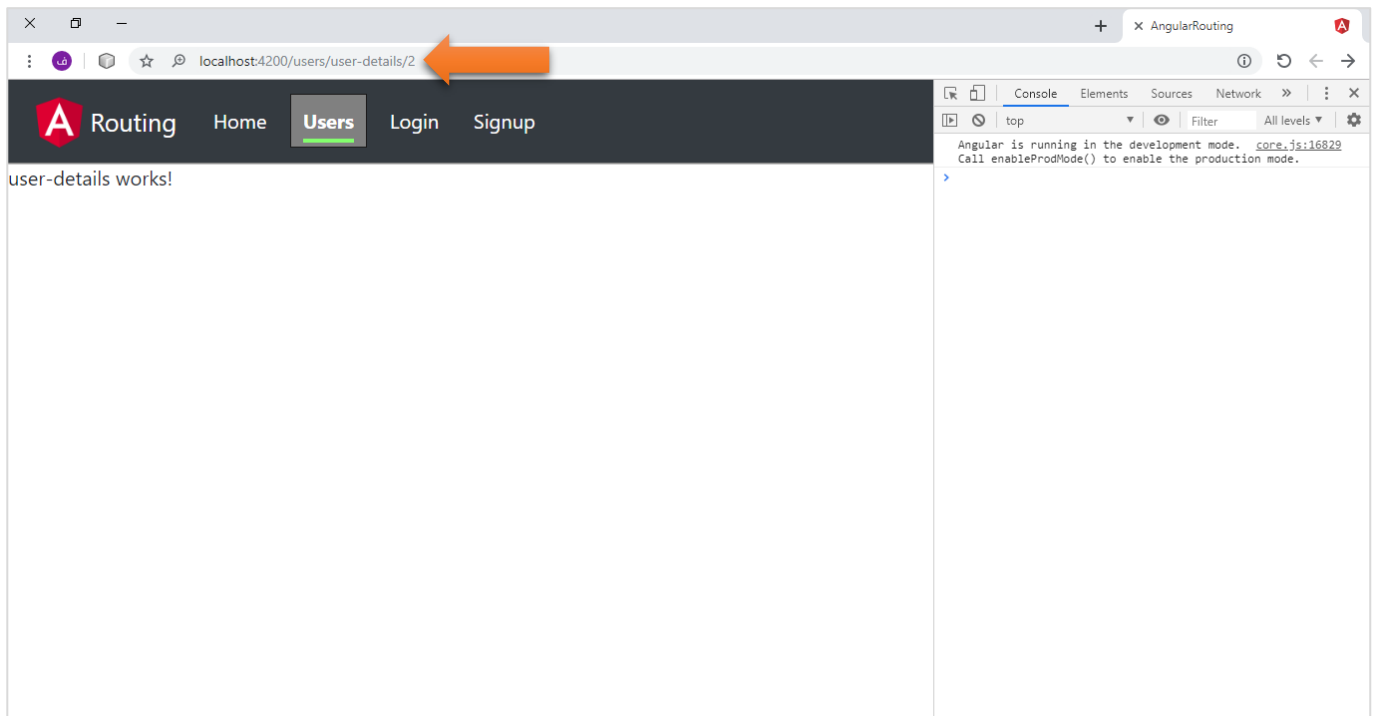
```
this.router.navigate(['../users/user-details', id]);
```

كما نلاحظ أننا حالياً عند الـ route الأب وهو users-list لذلك لا بد من أن نبدأ مسار التوجيه من rout الأب وهو users ومن ثم انتقلنا إلى الابن user-details ومعرفة مسارات routes مهمة لتنقل بين components المختلفة.

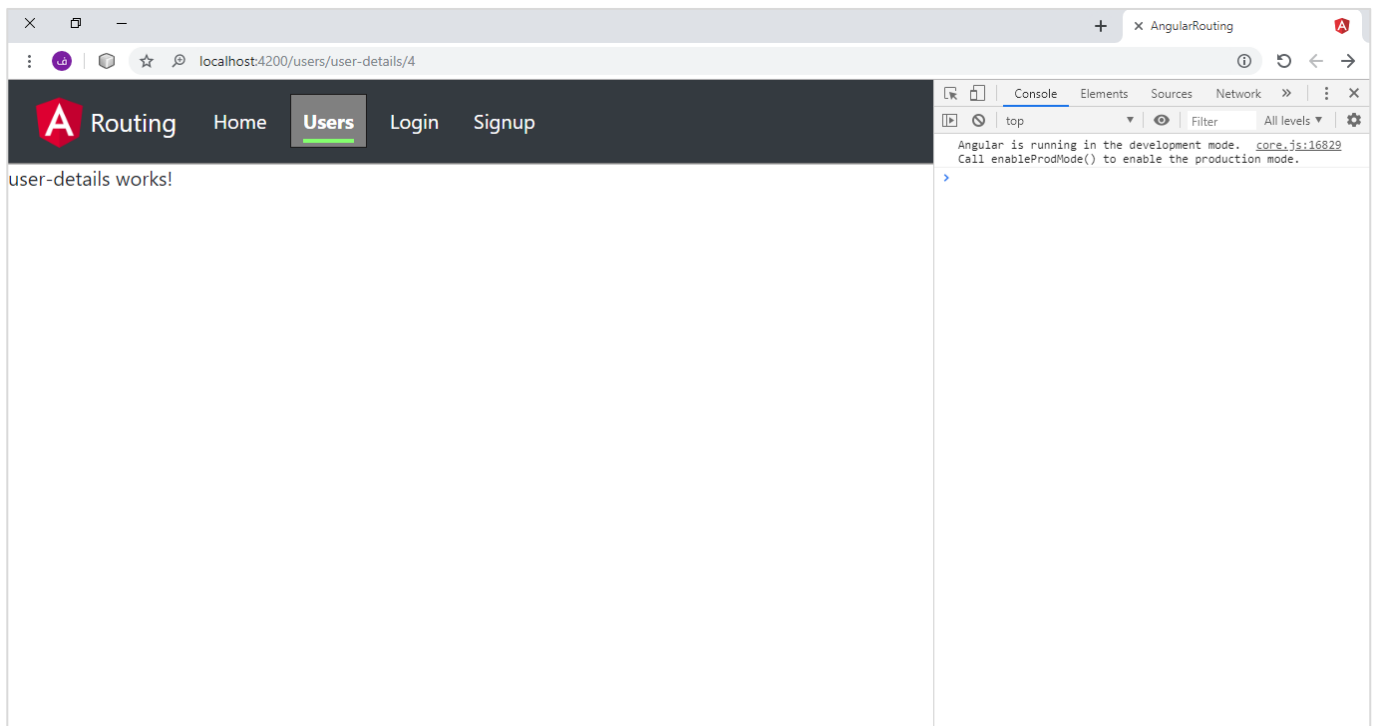
الآن لنحفظ التعديلات ونذهب إلى المتصفح لنرى نتيجة ما قمنا به:



نلاحظ أننا على المسار (الرابط) localhost:4200/users الآن لنقوم بالضغط على أحد الأزرار، ونرى النتيجة:



نلاحظ أنه في الرابط تم توجيهنا إلى المسار localhost:4200/users/user-details/2 حيث أن رقم 2 هو id للمستخدم الثاني وتمت إضافته إلى الرابط، لنقوم بتجربة مستخدم آخر وذلك بالرجوع إلى قائمة المستخدمين والضغط على مستخدم آخر غير الذي اخترناه سابقاً، كالتالي:

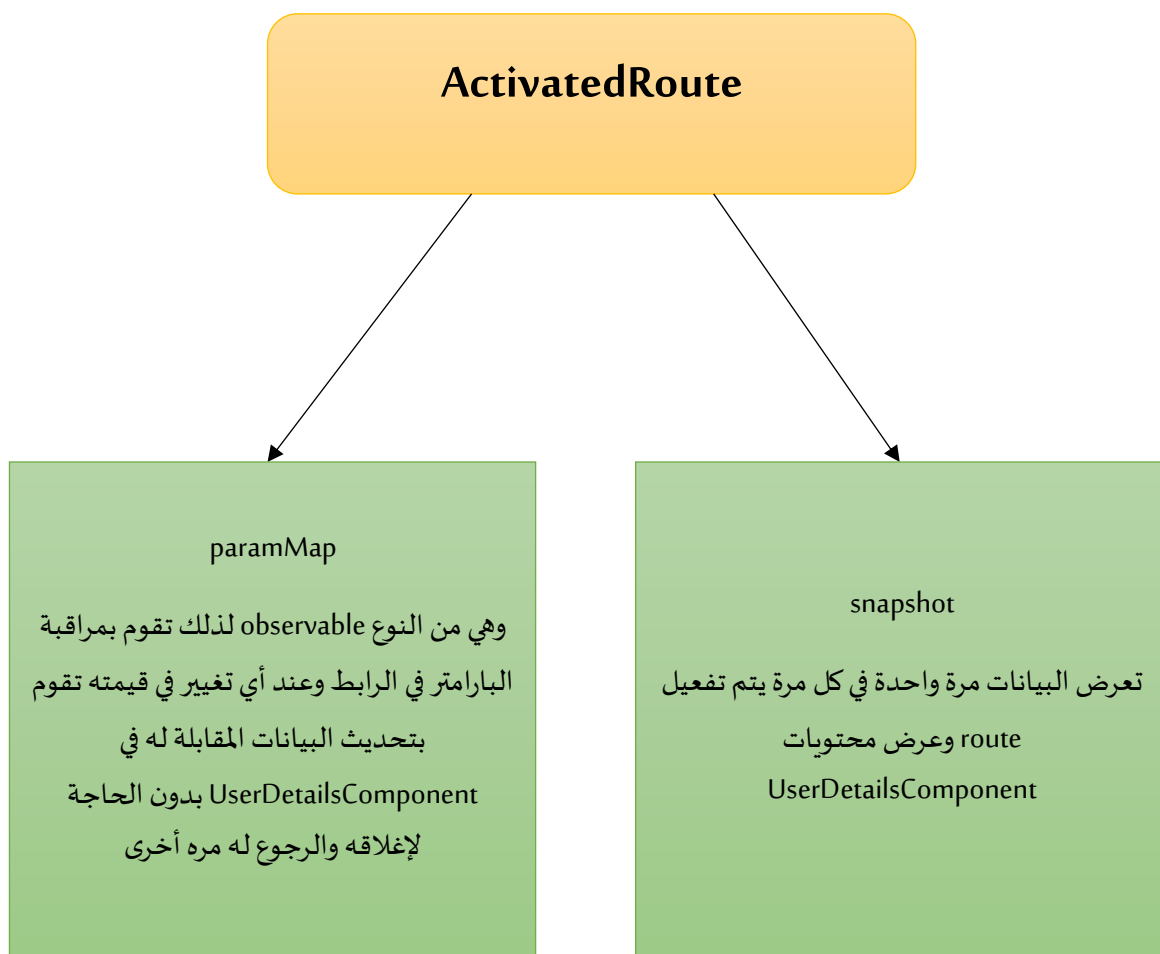


ملاحظة: لأرسال البيانات عن طريق template نستخدم directive ذو الاسم [RouterLink] ونمرر له القيم، كالتالي:

```
[routerLink]="['/users/user-details', user.userId]"
```

## 2.2.2. استقبال البارامترات المُرسلة عن طريق URL:

ولكي نستقبل id في UserDetailsComponent وذلك عن طريق قراءته عن طريق الرابط وعرض البيانات المُقابلة له، فيتم ذلك عن طريق استخدام service ذات الاسم ActivatedRoute وهي مقدمة لنا جاهزة من angular تحتوي على كم كبير من الدوال والخصائص التي تساعد في التعامل مع Route المفعّل حالياً على شاشة المستخدم، وما يهمنا حالياً من ActivatedRoute هي الدالتين هما snapshot والثانية هي paramMap حيث تقوم هذين الدالتين بالوصول إلى البارامتر الموجود في الرابط وتخزينه في متغير، أما الفرق بينهما فالدالة snapshot تقوم بقراءة الرابط مرة واحدة في كل مرة يتم إغلاق هذا component والرجوع له مرة أخرى أما الثانية paramMap فهي من النوع observable لذلك لا بد أن نعمل لها subscrip وهذا يؤدي إلى مراقبة الرابط وعند أي تغيير على قيمة البارامتر في الرابط نستطيع القيام بمهمة معينة بحسب الاحتياج، وسوف نتعامل مع كلا الدالتين من خلال التطبيق العملي، ويمكن تمثيلهما بالشكل التالي:



ولكن قبل البدء في تطبيق التعامل مع استقبال البارامتر وعرض البيانات لا بد من إنشاء دالة في service ذات الاسم UserService ومهمة هذه الدالة هي استقبال هذا البارامتر ومن ثم البحث في مصفوفة observable وتعيد لنا البيانات المطابقة لهذا البارامتر، ويتم عمل ذلك كالتالي:

ملف `users.service.ts`

```
1. import { Injectable } from '@angular/core';
2. import { Users } from './users';
3. import { of, Observable } from 'rxjs';
4. import { map } from 'rxjs/operators';
```

```

5.
6. const USERS = [
7.   new Users(1, 'Faisal', 'Riyadh', 'DevFaisal', '1234', 'admin'),
8.   new Users(2, 'Saad', 'Riyadh', 'DevSaad', '1111', 'no-admin'),
9.   new Users(3, 'Bader', 'Dammam', 'DevBader', '2222', 'no-admin'),
10.  new Users(4, 'Fahd', 'Jeddah', 'DevFahd', '3333', 'no-admin'),
11.];
12.
13. const userList$ = of(USERS);
14.
15. @Injectable({
16.   providedIn: 'root'
17. })
18.
19. export class UsersService {
20.
21.   constructor() { }
22.
23.   getAllUsers(): Observable<Users[]> {
24.     return userList$;
25.   }
26.
27.   getUserById(id: number): Observable<Users> {
28.     return this.getAllUsers().pipe(
29.       map(users => users.find(user => {
30.         return user.userId === id;
31.       })))
32.   );
33. }
34.
35. }

```

الدالة في الاسطر من 27 إلى 33 ومهمتها استقبال البارامتر واسمته id (لك حرية اختيار أي اسم تريده) وهو من النوع number وبنفس الوقت الدالة نفسها تعيد observable من النوع Users وليس مصفوفة لأننا هنا نريد الحصول على بيانات مستخدم واحد فقط وليس أكثر من واحد، أما محتواها فهو عن طريق الدالة (getAllUsers) التي تعاملنا معها سابقاً والتي تُرجع جميع المستخدمين نقوم بإخذ نسخة من هذه المصفوفة عن طريق الدالة map والتي استدعيناها في السطر 4 لكي تصبح هذه المصفوفة مصفوفة عادية من النوع Users[] وعن طريق الدالة find قمنا بعمل Loop على جميع عناصر المصفوفة وفي كل مرة نعمل مقارنة منطقية بين id المرسل وبين الخاصية userId في المصفوفة وفي حال حدوث توافق وأصبحت القيمة true يُرجع لنا بيانات الكائن الذي خاصية userId مساوية لـ id.

بعد الانتهاء من إنشاء الدالة نقوم الآن بكتابة كود استقبال البارامتر وقراءته، ويتم ذلك بالذهاب إلى ملف ts لـ UserDetailsComponent، ونقوم أولاً باستدعاء service السابقة لكي نستطيع الوصول إلى الدالة (getUserById) وبنفس الوقت نستدعي service ذات الاسم ActivatedRoute ولا ننسى نعمل inject لكلا services في constructor الخاص بالcomponent، بالإضافة إلى استدعاء الكلاس Users، كما كنا نعمل سابقاً كالتالي:



```

1. import { Component, OnInit } from '@angular/core';
2. import { ActivatedRoute } from '@angular/router';
3. import { UsersService } from 'src/app/users-data/users.service';
4. import { Users } from 'src/app/users-data/users';
5.
6. @Component({
7.   selector: 'app-user-details',
8.   templateUrl: './user-details.component.html',
9.   styleUrls: ['./user-details.component.css']
10.})
11. export class UserDetailsComponent implements OnInit {
12.   constructor(
13.     private activeRouter: ActivatedRoute,
14.     private usersService: UsersService) { }
15.
16.   ngOnInit() {
17.
18.   }
19.
20. }

```

في الاسطر من 2 إلى 3 قمنا بالاستدعاءات اللازمة وفي الاسطر 13 و 14 قمنا بعمل inject في متغيرات (لك حرية اختيار الأسماء التي تريدها).

### 1.2.2.2. استقبال البارامترات عن طريق الدالة snapshot:

في البداية سوف نطبق المثال على الدالة snapshot لذلك نحتاج إلى تعريف متغير لتخزين البارامتر القادم من الرابط ولنسميه id وايضاً نحتاج إلى متغير لتخزين البيانات المقابلة لهذا id والقادمة من الدالة (getUserById) وأخيراً بما أن هذا الدالة تُعيد observable لذلك لابد نعمل لها subscribe للوصول إلى البيانات التي تُعيدها الدالة find والموجودة داخل الدالة السابقة وبما أننا سوف نعمل subscribe لابد ايضاً أن نعمل unsubscribe لذلك نستدعي الـ interface ذو الاسم OnDestroy حيث تحتوي على دالة باسم ngOnDestroy وهذا الـ interface من نفس عائلة OnInit و OnChanges و...الخ، جميعها تُسمى Lifecycle Hooks بمعنى دورة حياة الـ component حيث تتيح لنا مجموعة من الدوال كل دالة تنفذ في وقت معين كأن تُنفذ في بداية تشغيل الـ component (ngOnInit) او عند إغلاق الـ component (ngOnDestroy) بالإضافة إلى المراحل الأخرى، لذلك سوف نعمل unsubscribe في هذه الدالة لكي نقطع الاتصال بين هذا component والبيانات القادمة إليه عند إغلاق هذا component (ولمعرفة أعمق عن دوال Lifecycle Hooks الرجاء مراجعة الكتاب الثاني من هذه السلسلة Angular Components and Services)، كالتالي:

```

1. import { Component, OnInit, OnDestroy } from '@angular/core';
2. import { ActivatedRoute, Router } from '@angular/router';
3. import { UsersService } from 'src/app/users-data/users.service';
4. import { Subscription } from 'rxjs';
5. import { Users } from 'src/app/users-data/users';
6.

```

```

7. @Component({
8.   selector: 'app-user-details',
9.   templateUrl: './user-details.component.html',
10.  styleUrls: ['./user-details.component.css']
11.})
12.export class UserDetailsComponent implements OnInit, OnDestroy {
13.  user$: Users;
14.  private subscription: Subscription;
15.  private id: number;
16.  constructor(
17.    private activeRouter: ActivatedRoute,
18.    private usersService: UsersService ) { }
19.
20.  ngOnInit() {
21.
22.    this.id = +this.activeRouter.snapshot.params.id;
23.    this.subscription = this.usersService.getUserById(this.id).subscribe(user => {
24.      this.user$ = user
25.    });
26.
27.  }
28.
29.  ngOnDestroy() {
30.    this.subscription.unsubscribe();
31.  }
32.
33.}

```

في السطر 13 و 14 و 15 عرفنا المتغيرات مع استدعائنا لنوع Subscription في السطر 4 وفي السطر 22 استخدمنا الدالة snapshot للوصول إلى البارامتر id (وهو نفس الاسم الذي وضعناه له عند تهيئتنا له في routes) وتخزين قيمته في المتغير الذي عرفناه قبل قليل باسم id في السطر 15 اما علامة الزائد فمعناها تحويل قيمة البارامتر إلى قيمه رقميه ومن ثم تخزينها في المتغير this.id، اما في الاسطر من 23 إلى 25 استدعينا الدالة getUserById ومررنا لها البارامتر الذي اخذنا قيمته في السطر السابق، ومن ثم عملنا subscribe لاستقبال البيانات وقمنا بتخزينها في متغير اسميناها user ومن ثم اسندنا هذه المتغير إلى المتغير user\$ الذي عرفناها في السطر 13 لأنه هو الذي نقوم بإرساله إلى ملف html لعرض ما يحتويه من بيانات، وأخيراً في الاسطر من 29 إلى 30 عملنا unsubscribe.

بذلك نكون استقبلنا البيانات في ملف ts وبقي لنا أن نقوم بعرضها للمستخدم في ملف html وذلك بالاستفادة من البيانات الموجودة في المتغير user\$، الآن لنذهب إلى ملف html وملف css ونقوم بإضافة الأكواد اللازمة لعرض البيانات بالإضافة إلى أنني سوف اضيف زر للرجوع إلى صفحة UsersComponent، عن طريق استخدام RouterLink كما فعلنا في أكثر من موضع، كالتالي:

ملف user-details.component.html

```

1. <div class="card">
2.   <div class="card-body">
3.     <ul>
4.       <li>

```

```

5.     
6.     <h3>{{ user$.name }}</h3>
7.     <p>
8.         User ID: <span>{{ user$.userId }}</span>
9.     </p>
10.    <p>
11.        UserName: <span>{{ user$.userName }}</span>
12.    </p>
13.    <p>
14.        User Password: <span>{{ user$.password }}</span>
15.    </p>
16.    <p>
17.        User City: <span>{{ user$.userCity }}</span>
18.    </p>
19.    <p>
20.        User Type: <span>{{ user$.userType }}</span>
21.    </p>
22. </li>
23. </ul>
24. </div>
25. <div class="card-footer">
26.     <a class="btn btn-dark" routerLink="/users">Back</a>
27. </div>
28. </div>

```

بما أن الأكواد السابق هي تكرار لما كنا نفعله سابقاً لذلك لن اعيد شرحها ولكن سوف اشير هنا إلى ما هو موجود في السطر 26 حيث لو لاحظنا بعض الأحيان نضيف الأقواس المربعة إلى routerLink وبعضها لا نضيف وحقيقة هذا الامر يعتمد على القيمة المسندة إليه فإذا كانت قيمة ديناميكية على شكل متغيرات كما فعلنا في الملاحظة التي اشرنا فيها إلى كيفية ارسال البارامترات عن طريق template ففي هذه الحالة نضيف الاقواس المربعة او ما تسمى Property Binding، وهناك حالة أخرى إذا كان الرابط مقسم إلى أكثر من جزء كأن يكون مصفوفة من path، أما إذا كان كتلة نصيه واحدة وثابته نفس الحالة السابقة فلا نضيف الأقواس المربعة.

#### ملف user-details.component.css

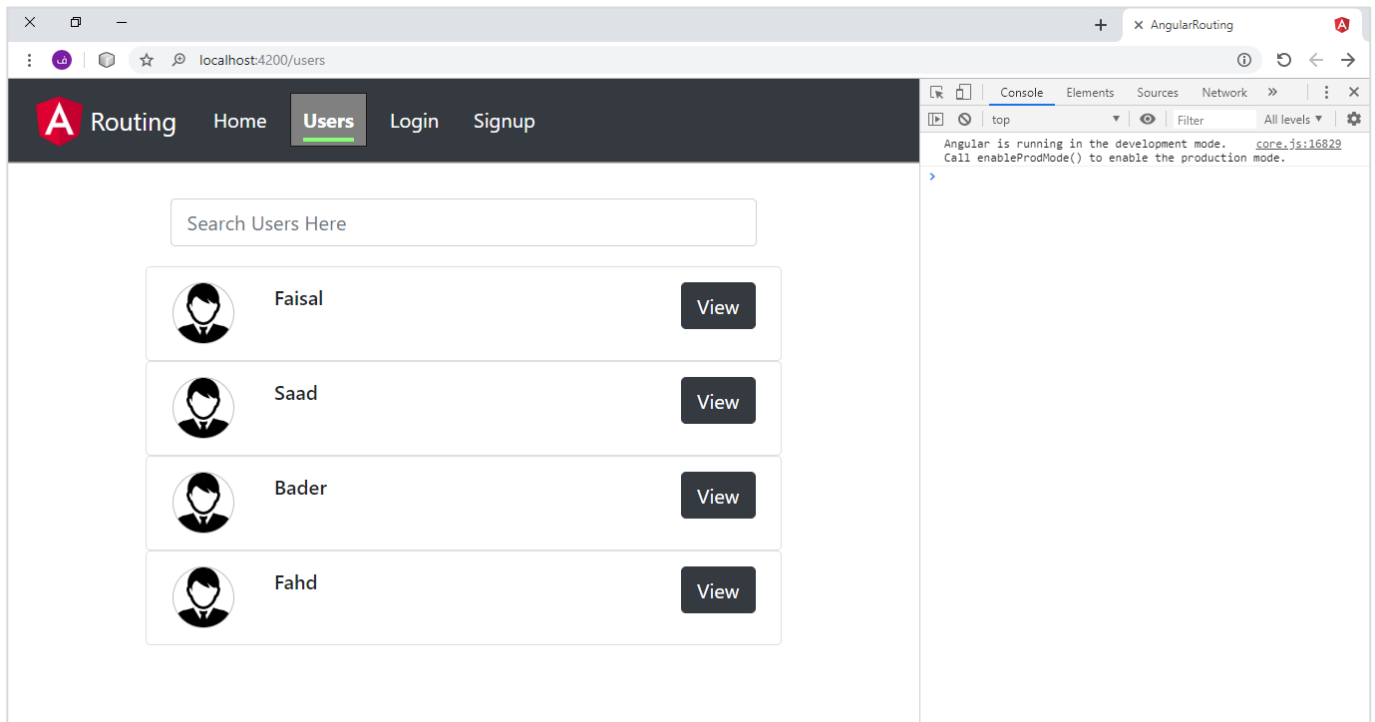
```

1. div {
2.     margin: 20px;
3.
4. }
5.
6. .card{
7.     border: 0;
8. }
9.
10. .card-body{
11.     background: rgba(253, 253, 253, 0.925);
12.     border: 1px solid silver;
13.     margin-bottom: 0px;
14. }
15.
16. .card-footer{

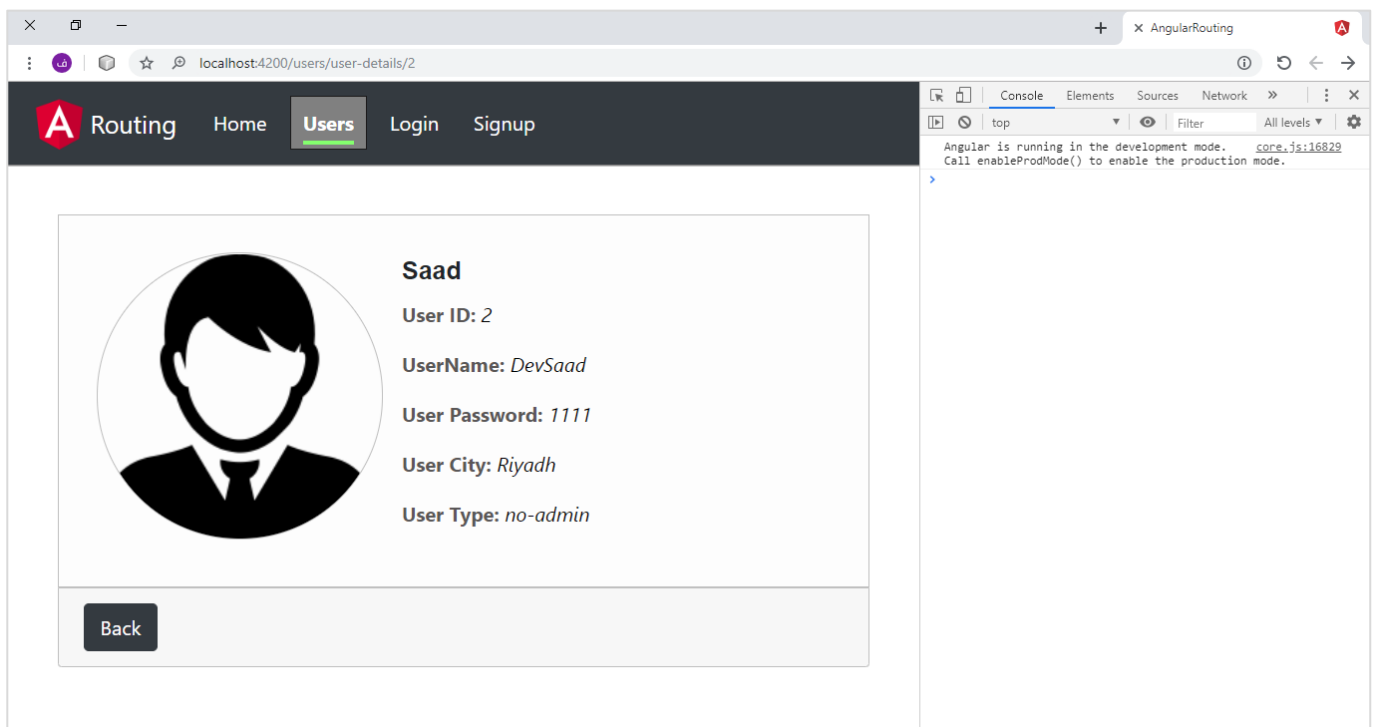
```

```
17. border: 1px solid silver;
18. background: rgba(246, 246, 246, 0.925);
19. margin-top: 0px;
20.}
21.
22.ul {
23. list-style-type: none;
24. margin: 0;
25. padding: 0;
26. width: 500px;
27.}
28.
29.li {
30. padding: 10px;
31. overflow: auto;
32.}
33.
34.li img {
35. float: left;
36. margin: 0 15px 0 0;
37. border: 1px solid silver;
38. height: 230px;
39. border-radius: 50%
40.}
41.
42.li p {
43. font: 200 12px/1.5;
44. color: rgb(90, 87, 87);
45. font-weight: bold;
46.}
47.
48.li p span {
49. font: 200 12px/1.5;
50. color: rgb(0, 0, 0);
51. font-weight: normal;
52. font-style: italic;
53.}
54.h3 {
55. font: bold 20px/1.5 Helvetica, Verdana, sans-serif;
56.}
```

الآن لنحفظ التعديلات ونذهب إلى المتصفح لنرى النتيجة:



الآن لنختار أحد المستخدمين:



نلاحظ ظهرت لنا بيانات المستخدم الذي تم اختياره، ولو ضغطنا على زر Back سوف يُرجعنا إلى الصفحة السابقة.


## 2.2.2.2. استقبال البارامترات عن طريق الدالة paramMap:

بعد أن قمنا بتطبيق الدالة الأولى snapshot لنقوم الآن بتطبيق الدالة الثانية paramMap ولكن قبل تطبيقها لنفهم بشكل عملي الفرق بين الدالتين، وكما قلنا سابقاً أن الدالة snapshot لتحديث ما بها من بيانات لا بد من إغلاق component والرجوع له مره أخرى بعكس الدالة الثانية لذلك لنقوم بإضافة زر وهذا الزر لنسميه Next User حيث

ينتقل بين المستخدمين بدون حاجة لإغلاق component والرجوع لقائمة المستخدمين واختيار مستخدم آخر، لذلك سوف نضيف الزر في صفحة html لهذا component، ونضيف لها الزر دالة وليكن اسمها viewNextUser()، كالتالي:

#### ملف user-details.component.html

```
1. <div class="card">
2.   <div class="card-body">
3.     <ul>
4.       <li>
5.         
6.         <h3>{{ user$.name }}</h3>
7.         <p>
8.           User ID: <span>{{ user$.userId }}</span>
9.         </p>
10.        <p>
11.          UserName: <span>{{ user$.userName }}</span>
12.        </p>
13.        <p>
14.          User Password: <span>{{ user$.password }}</span>
15.        </p>
16.        <p>
17.          User City: <span>{{ user$.userCity }}</span>
18.        </p>
19.        <p>
20.          User Type: <span>{{ user$.userType }}</span>
21.        </p>
22.      </li>
23.    </ul>
24.  </div>
25.  <div class="card-footer">
26.    <a class="btn btn-dark" routerLink="/users">Back</a>
27.    <button class="btn btn-dark pull-right" (click)="viewNextUser()">Next User</button>
28.  </div>
29.</div>
```



ونذهب إلى ملف ts ونكتب محتوى هذه الدالة، كالتالي:

#### ملف user-details.component.ts

```
1. import { Component, OnInit, OnDestroy } from '@angular/core';
2. import { ActivatedRoute, Router } from '@angular/router';
3. import { UsersService } from 'src/app/users-data/users.service';
4. import { Subscription } from 'rxjs';
5. import { Users } from 'src/app/users-data/users';
6.
7. @Component({
8.   selector: 'app-user-details',
9.   templateUrl: './user-details.component.html',
10.  styleUrls: ['./user-details.component.css']
11.})
12.
13. export class UserDetailsComponent implements OnInit, OnDestroy {
14.  user$: Users;
```

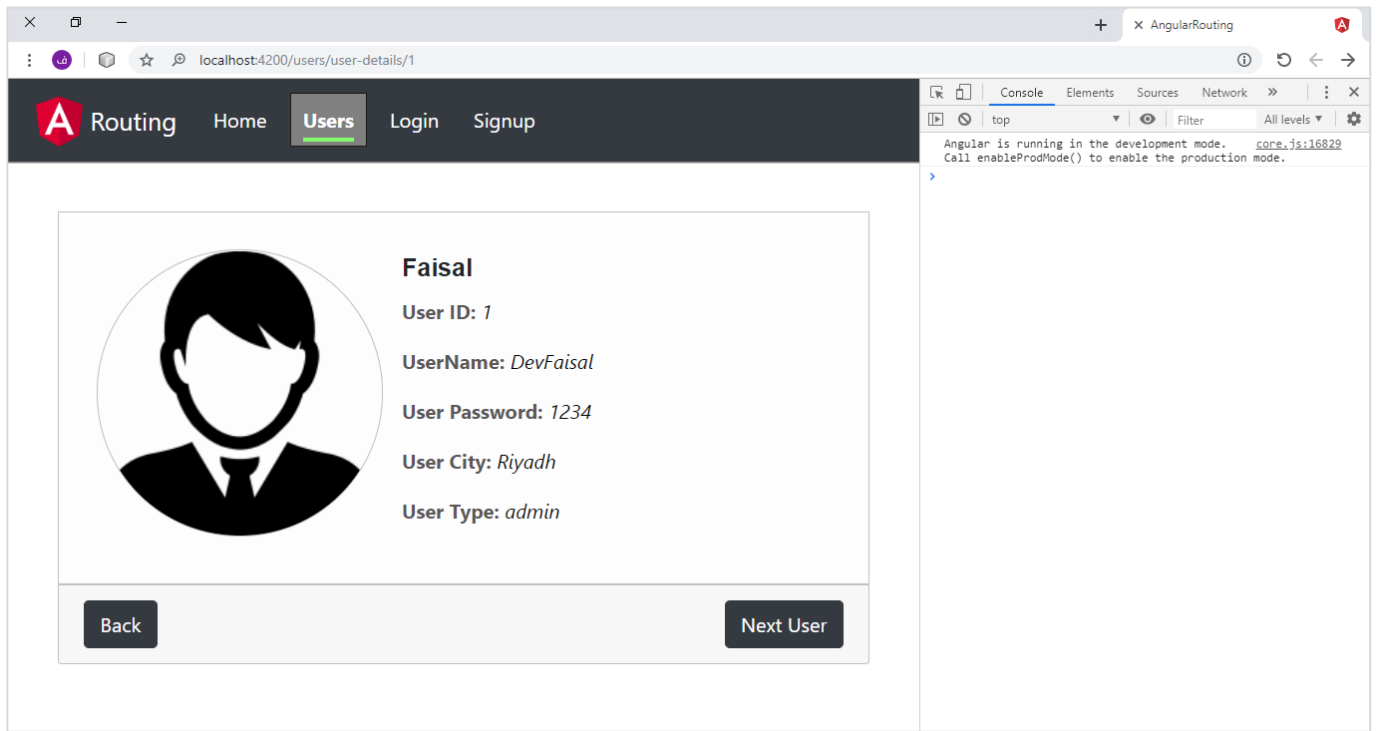
```

15. private subscription: Subscription;
16. private id: number;
17. constructor(
18.     private activeRouter: ActivatedRoute,
19.     private usersService: UsersService,
20.     private router: Router) { }
21.
22. ngOnInit() {
23.
24.     this.id = +this.activeRouter.snapshot.params.id;
25.     this.subscription = this.usersService.getUserById(this.id).subscribe(user => {
26.         this.user$ = user;
27.     });
28.
29. }
30.
31. viewNextUser() {
32.     this.subscription = this.usersService.getAllUsers().subscribe(data => {
33.         this.id === data.length ? this.id = 1 : this.id = this.id + 1;
34.         this.router.navigate(['users/user-details', this.id]);
35.     });
36. }
37.
38. ngOnDestroy() {
39.     this.subscription.unsubscribe();
40. }
41.
42. }

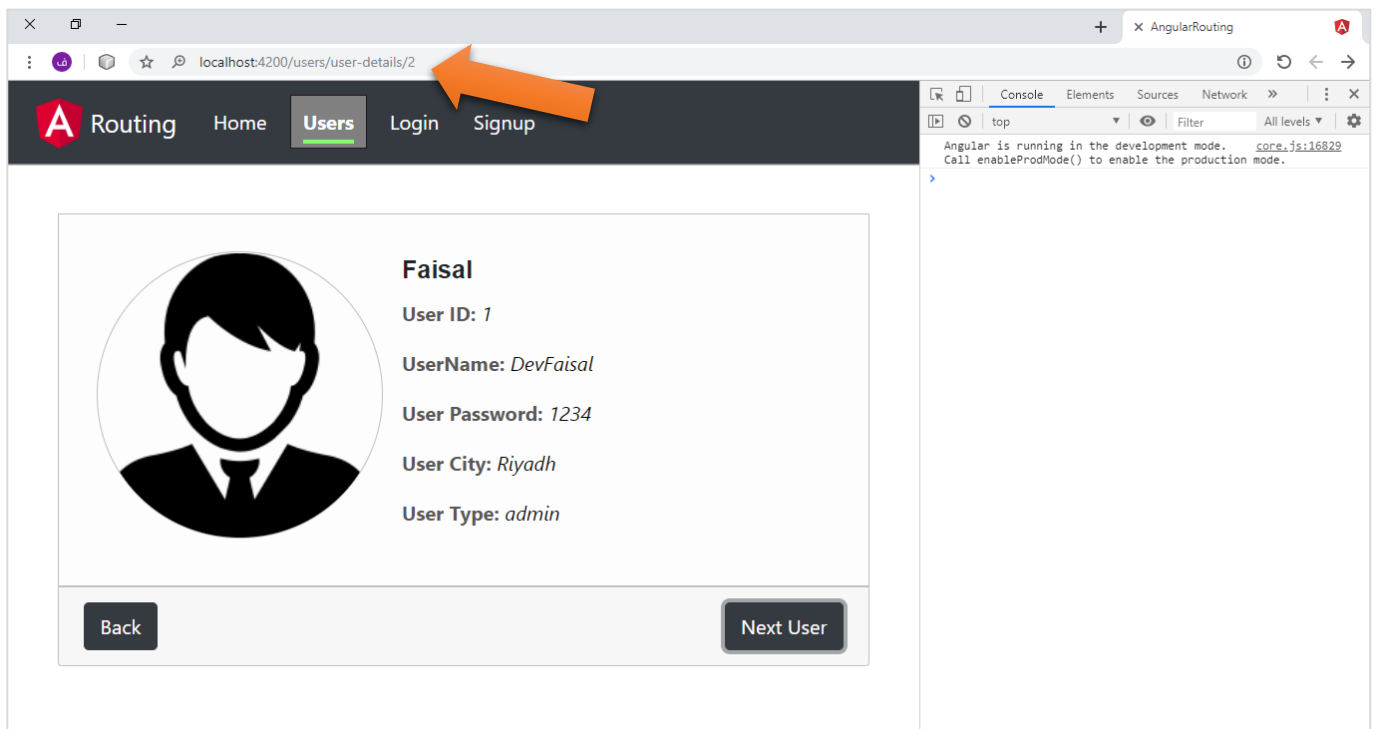
```

الدالة ومحتواها من السطر 31 إلى السطر 36 ومهمة هذه الدالة هي الانتقال بين المستخدمين وفي حال الوصول إلى آخر مستخدم تبدأ من أول مستخدم وهكذا، لذلك نحتاج أن نعرف عدد المستخدمين في البداية ولمعرفة ذلك نعمل subscribe لدالة getAllUsers() للحصول على جميع الكائنات في المصفوفة ونخزن هذه البيانات في متغير اسميته data ومن ثم نقارن هل قيمة id مساوية لعدد الكائنات في المصفوفة إذا كانت مساوية فهذا يعني أننا وصلنا إلى آخر مستخدم موجود فنقوم بتغيير قيمة id ونجعلها تساوي واحد، وإذا لم تكن أضف واحد لزيادة قيمة id، ومن ثم نمرر هذه القيمة لي navigate لكي نغير الرابط كما تعلمنا سابقاً ولا ننسى استدعاء Router كما عملنا في السطر 2 وعملنا له inject في السطر 20.

الآن لنحفظ هذه التعديلات ونذهب إلى المتصفح:



نحن الآن عن المستخدم الي حمل الid ذو القيمة واحد، الآن لنضغط على الزر Next User، والرجاء التركيز على الرابط:



نلاحظ قيمة id في الرابط تغيرت وعمل انتقال او توجيه navigate ولكن البيانات في component لم تتحدث، ومهما ضغطنا على الزر فإن القيمة في الرابط تتغير ولكن البيانات لن تتحدث، وهذا هو ما كنا نتحدث عنه وقلنا انه الفرق الجوهرى بين snapshot و paramMap فإن snapshot تقرأ قيمة البارامتر مرة واحدة في بداية تشغيل component وإذا اردنا ان نقرأ قيمة ثانية لابد أن نغلق هذا component ونرجع إلى صفحة المستخدمين ونختار مستخدم آخر، إذن ما هو الحل إذا احتجنا أن نعرض البيانات بدون أغلاق component الحل عزيزي المتعلم هو في الدالة الثانية paramMap التي تقوم بمراقبة الرابط وفي حال حدوث أي تعديل على قيمة البارامتر الذي يحتويه هذا الرابط تنفذ الأكواد التي بداخلها



وفي حالتنا هذه هي عرض بيانات المستخدم الذي يحتوي على id الموافق للid الموجود في الرابط، وبما أن هذه الدالة تُعيد observable فلا بد أن نعمل لها subscribe، كالتالي:

ملف user-details.component.ts

```
1. import { Component, OnInit, OnDestroy } from '@angular/core';
2. import { ActivatedRoute, Router } from '@angular/router';
3. import { UsersService } from 'src/app/users-data/users.service';
4. import { Subscription } from 'rxjs';
5. import { Users } from 'src/app/users-data/users';
6.
7. @Component({
8.   selector: 'app-user-details',
9.   templateUrl: './user-details.component.html',
10.  styleUrls: ['./user-details.component.css']
11.})
12. export class UserDetailsComponent implements OnInit, OnDestroy {
13.   user$: Users;
14.   private subscription: Subscription;
15.   private id: number;
16.   constructor(
17.     private activeRouter: ActivatedRoute,
18.     private usersService: UsersService,
19.     private router: Router) { }
20.
21.   ngOnInit() {
22.
23.     // this.id = +this.activeRouter.snapshot.params.id;
24.     // this.subscription = this.usersService.getUserById(this.id).subscribe(user => {
25.     //   this.user$ = user;
26.     // });
27.
28.     this.activeRouter.paramMap.subscribe(param => {
29.       this.id = +param.get('id');
30.       this.subscription = this.usersService.getUserById(this.id).subscribe(data => {
31.         this.user$ = data;
32.       });
33.     });
34.
35.   }
36.
37.   ngOnDestroy() {
38.     this.subscription.unsubscribe();
39.   }
40.
41.   viewNextUser() {
42.     this.subscription = this.usersService.getAllUsers().subscribe(data => {
43.       this.id === data.length ? this.id = 1 : this.id = this.id + 1;
44.     });
45.     this.router.navigate(['users/user-details', this.id]);
46.   }
47. }
```

نلاحظ أننا عملنا تعطيل للكود السابق الخاص بالدالة الأولى snapshot في الاسطر من 23 إلى 26 واضفنا الكود الخاص بالدالة الثانية في الاسطر من 28 إلى 33 عملنا subscribe للدالة paramMap وهي تُعيد لنا كلاس يحتوي على مجموعة دوال:

get: للحصول على قيمة بارامتر واحد على شكل قيمة نصية، وفي حال استخدمنا هذه الدالة وكانت البارامترات أكثر من بارامتر فأنها تُعيد أول بارامتر فقط.

getAll: للحصول على قيمة أكثر من بارامتر على شكل مصفوفة نصية، ولا تظهر إلا قيم البارامترات التي تم تمريرها في الرابط وفي حال وجود بارامتر ليس له قيمه فمن الطبيعي لا تُظهره.

has: لتأكد هل البارامتر موجود في الرابط أو لا وهي تُعيد قيمة منطقية true او false.

keys: في Optional Parameters أو Query Parameters ترسل البارامترات على شكل كائن، ومعروف أن الكائن يحتوي على واحد أو أكثر من keys وكل key يحتوي على value (قيمة)، وتُظهر جميع الـ Keys سواء هذا key يحمل قيمة أو لا، لذلك في هذه الحالات نستخدم هذه الدالة التي تُعيد لنا مصفوفة نصية بالـ keys الخاصة به.

ملاحظة: هذه الدوال توجد أيضاً في الدالة الأولى، ويمكن الوصول لها بالطريقة التالية:

```
this.activatedRout.snapshot.paramMap.get("");
```

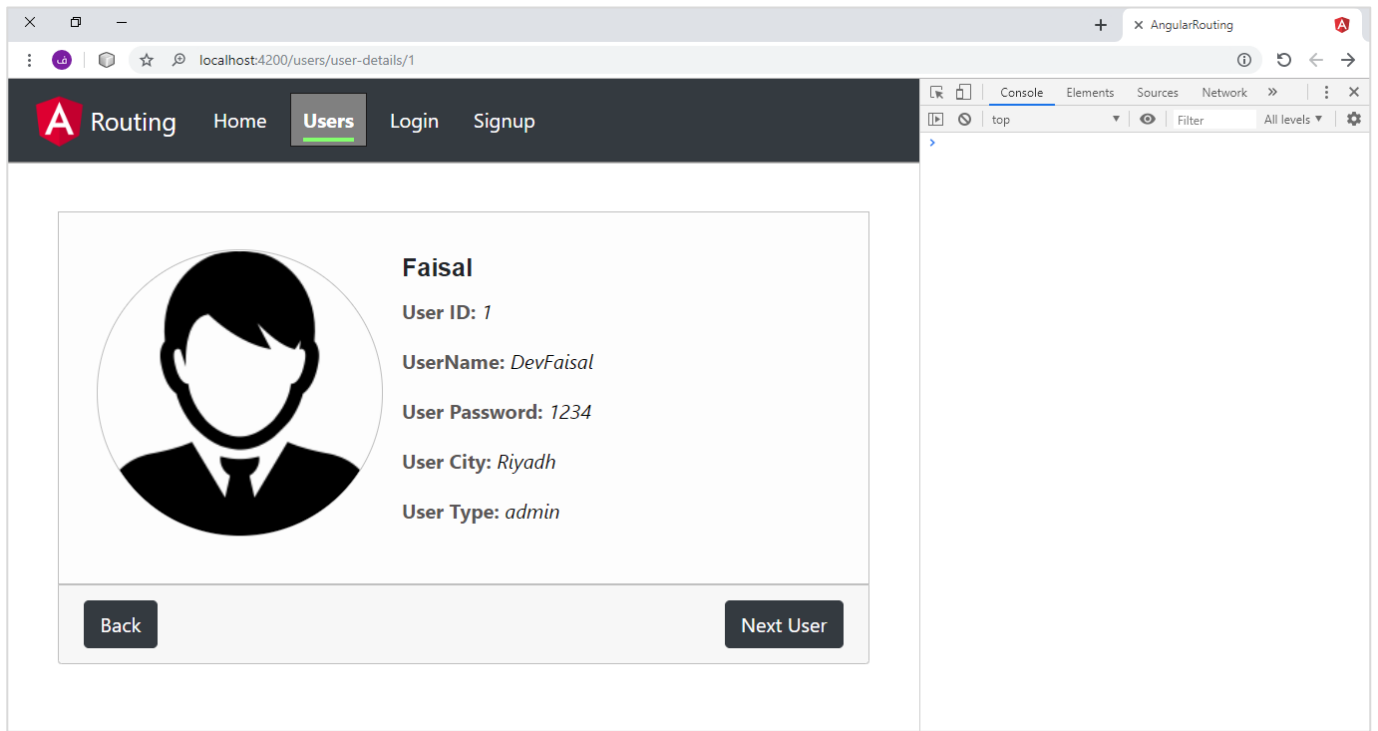
```
this.activatedRout.snapshot.paramMap.getAll("");
```

```
this.activatedRout.snapshot.paramMap.has("");
```

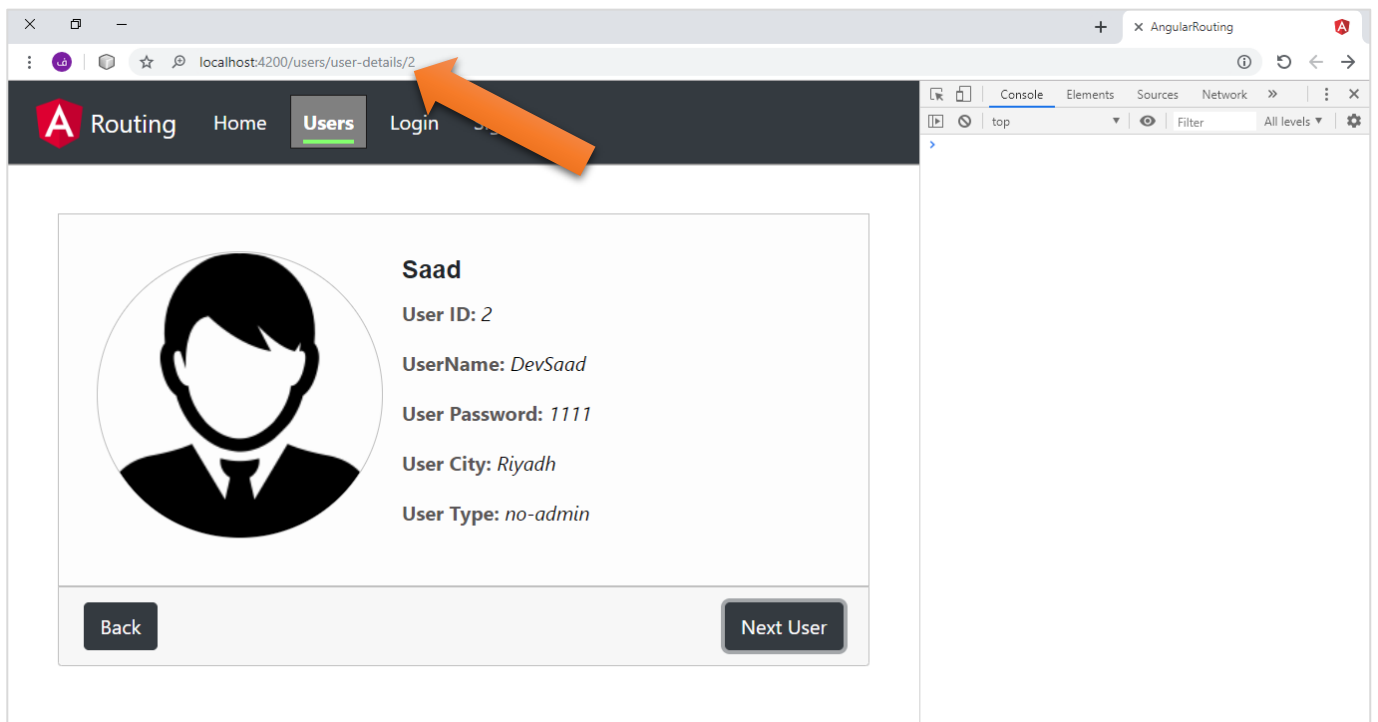
```
this.activatedRout.snapshot.paramMap.keys;
```

اما ما يُناسبنا هنا هو الدالة الأولى get لذلك قمنا باستخدامها للحصول على قيمة البارامتر بعد تحويله إلى قيمة رقمية عن طريق علامة الزائد (+) وخرنا هذه القيمة في المتغير id، أما الاسطر الباقية فهي مشابه لما قمنا به في الدالة الأولى.

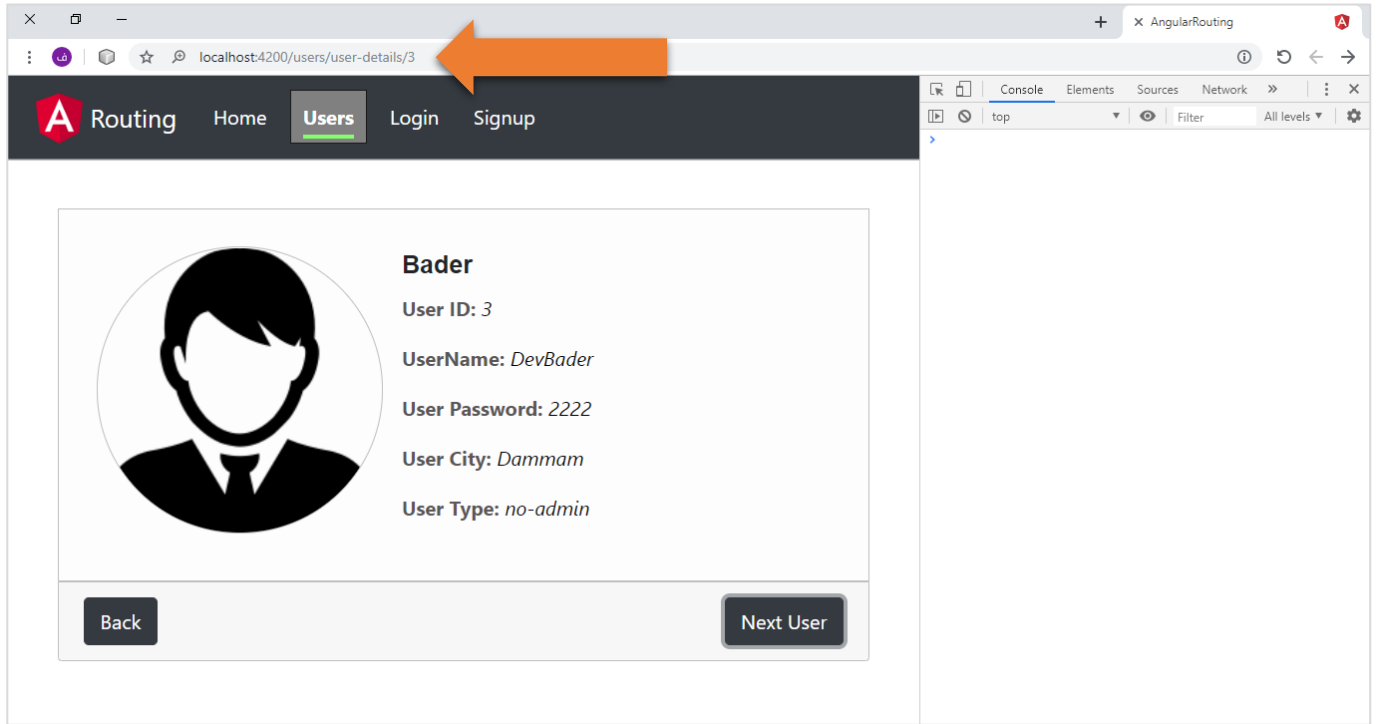
الآن لنقوم بحفظ هذه التعديلات والانتقال للمتصفح لمشاهدة النتيجة، كالتالي:



عند أول مستخدم في القائمة وقيمة البارامتر له في الرابط هو واحد، لنضغط على زر Next User:



قيمة البارامتر تغيرت وايضاً البيانات الموجودة في component تحدثت، لنقوم بالضغط على الزر مرة أخرى:



وأخيراً يجب أن نشير متى استخدم دالة snapshot ومتى استخدم دالة paramMap، الإجابة عن هذا السؤال ترجع إلى احتياجك أنت عزيز المتعلم فإذا كنت متأكد أن البيانات يتم قراءتها مرة واحدة ولا تحتاج إلى تحديث هذه البيانات فستستخدم الدالة الأولى وإذا كانت البيانات احتمال ان يتم تحديثها بدون اغلاق component فستستخدم الدالة الثانية.

### 3.2. Optional Parameters

من أسمها هي بارامترات اختيارية بمعنى أن component المستقبل لها لا يعتمد بصورة رئيسية عليها لعرض البيانات الخاصة به وبنفس الوقت لا تحتاج إلى تهيئة في routes بعكس Required Parameters، وانما هي مجموعة من البيانات التي تُرسل إلى component لأجراء تعديلات معينة، اما بالنسبة لإرسال وقراءة هذا النوع فهو مشابه تماماً لنوع السابق ولعل الفرق الوحيد أن هذه البارامترات تُرسل على شكل كائن object ويحتوي بداخله على مجموعة من keys وقيم هذه keys.

ولعل من أبسط الأمثلة عليه والمتوافق مع المشروع الذي نطبقه هنا هو في حال اختيارنا لأحد المستخدمين من قائمة المستخدمين لعرض التفاصيل الخاصة به ومن ثم عندما يضغط على زر الرجوع إلى قائمة المستخدمين نريد أن نحدد المستخدم الذي اختاره بلون مختلف عن باقي المستخدمين في القائمة.

ولعمل ذلك نذهب إلى ملف html لـ UserDetailsComponent وبالتحديد إلى زر الرجوع وبأكثر تحديداً إلى RouterLink ونقوم بتغيير قيمته وذلك بإضافة كائن له يحتوي على البارامتر او مجموعة البارامترات التي نريد إرسالها، كالتالي:

ملف user-details.component.html

```
1. <div class="card">
2.   <div class="card-body">
3.     <ul>
4.       <li>
5.         
```

```

6.      <h3>{{ user$.name }}</h3>
7.      <p>
8.          User ID: <span>{{ user$.userId }}</span>
9.      </p>
10.     <p>
11.         UserName: <span>{{ user$.userName }}</span>
12.     </p>
13.     <p>
14.         User Password: <span>{{ user$.password }}</span>
15.     </p>
16.     <p>
17.         User City: <span>{{ user$.userCity }}</span>
18.     </p>
19.     <p>
20.         User Type: <span>{{ user$.userType }}</span>
21.     </p>
22. </li>
23. </ul>
24. </div>
25. <div class="card-footer">
26.     <a class="btn btn-dark" [routerLink]="['/users', { name: user$.name }]">Back</a>
27.     <button class="btn btn-dark pull-right" (click)="viewNextUser()">Next User</button>
28. </div>
29. </div>

```

نلاحظ في السطر 26 اضعفنا لrouterLink بالإضافة إلى path الخاص بUsersComponent أضفنا كائن وهذا الكائن يحتوي على key اسميته name (لك حرية اختيار الاسم الذي تريده) والقيمة الخاصة به هي خاصية name الموجودة في المتغير users\$ التي تعاملنا معها سابقاً وشرحناها بشكل مفصل وهذه الخاصية تحمل اسم المستخدم الذي نستعرض بياناته في هذا component، مع العلم أنه يمكن إرسال أكثر من key، كالتالي:

```
[routerLink]="['/users', { name: user$.name, id: user$.id, UserName: user$.userName ...etc. }]"
```

ونلاحظ أيضاً أننا أضفنا الأقواس المربعة أو ما تسمى اقواس المصفوفة إلى routerLink لأننا كما هو واضح مررنا قيم ديناميكية متغيرة على شكل متغيرات Property Binding.

بعد إرسالنا لهذه البارامترات لنقوم الآن بقراءتها في UsersListComponent والقراءة هي مشابهة لما قمنا به سابقاً، وسوف استخدم الدالة snapshot، كالتالي:

#### ملف users-list.component.ts

```

1. import { Component, OnInit, Input, OnChanges } from '@angular/core';
2. import { Observable, of } from 'rxjs';
3. import { Users } from 'src/app/users-data/users';
4. import { Router, ActivatedRoute } from '@angular/router';
5.
6. @Component({
7.     selector: 'app-users-list',
8.     templateUrl: './users-list.component.html',
9.     styleUrls: ['./users-list.component.css']

```

```

10.})
11.export class UsersListComponent implements OnInit, OnChanges {
12.  @Input() users$: Observable<Users[]>;
13.  @Input() fullUsersList$: Observable<Users[]>;
14.  @Input() inputSearch: string;
15.  selectedUser: string;
16.
17.  constructor(private router: Router, private activatedRoute: ActivatedRoute) { }
18.
19.  ngOnInit() {
20.    this.selectedUser = this.activatedRoute.snapshot.paramMap.get('name');
21.  }
22.
23.  ngOnChanges() {
24.    if (this.inputSearch) {
25.      this.users$.subscribe(data => {
26.        const result = data.filter(user =>
27.          user.name.toLocaleLowerCase().indexOf(this.inputSearch.toLocaleLowerCase()) !== -1);
28.        this.users$ = of(result);
29.      });
30.    } else {
31.      this.users$ = this.fullUsersList$;
32.    }
33.  }
34.
35.  viewDetails(id: number) {
36.    this.router.navigate(['users/user-details', id]);
37.  }
38.
39.}

```

في السطر 20 تحصلنا على قيمة البارامتر وقمنا بتخزينه في المتغير selectedUser الذي قمنا بتعريفه في السطر 15، وكنوع من التغيير ومعرفة الطرق المتنوعة في طرق الوصول إلى حل المشكلة، قمنا باستخدام الطريقة الثانية وهي استخدام الدالة get عن طريقة الدالة snapshot.

ملاحظة: كما يمكن أيضاً استخدام الدالة params في الدالة snapshot كما كنا نفعل سابقاً وتمير البارامتر بين اقواس مربعة، كالتالي:

```
this.selectedUser = this.activatedRoute.snapshot.params['name'];
```

ملاحظة: في حال أردنا الحصول على keys الخاصة بالبارامتر نستخدم الدالة الثانية paramMap ونستخدم الدالة keys:

```
Keys: string[];
```

```
this.activatedRoute.paramMap.subscribe(param => {
```

```
  this.Keys = param.keys;
```

```
});
```

أو الدالة الأولى، بالطريقة التالية:

```
Keys: string[ ];
```

```
this.Keys = this.activatedRoute.snapshot paramMap.keys;
```

الآن لنذهب إلى ملف css لهذا component وننشأ كلاس وليكن اسمه is-active ومهمته فقط تغيير لون الخلفية، كالتالي:

ملف users-list.component.css

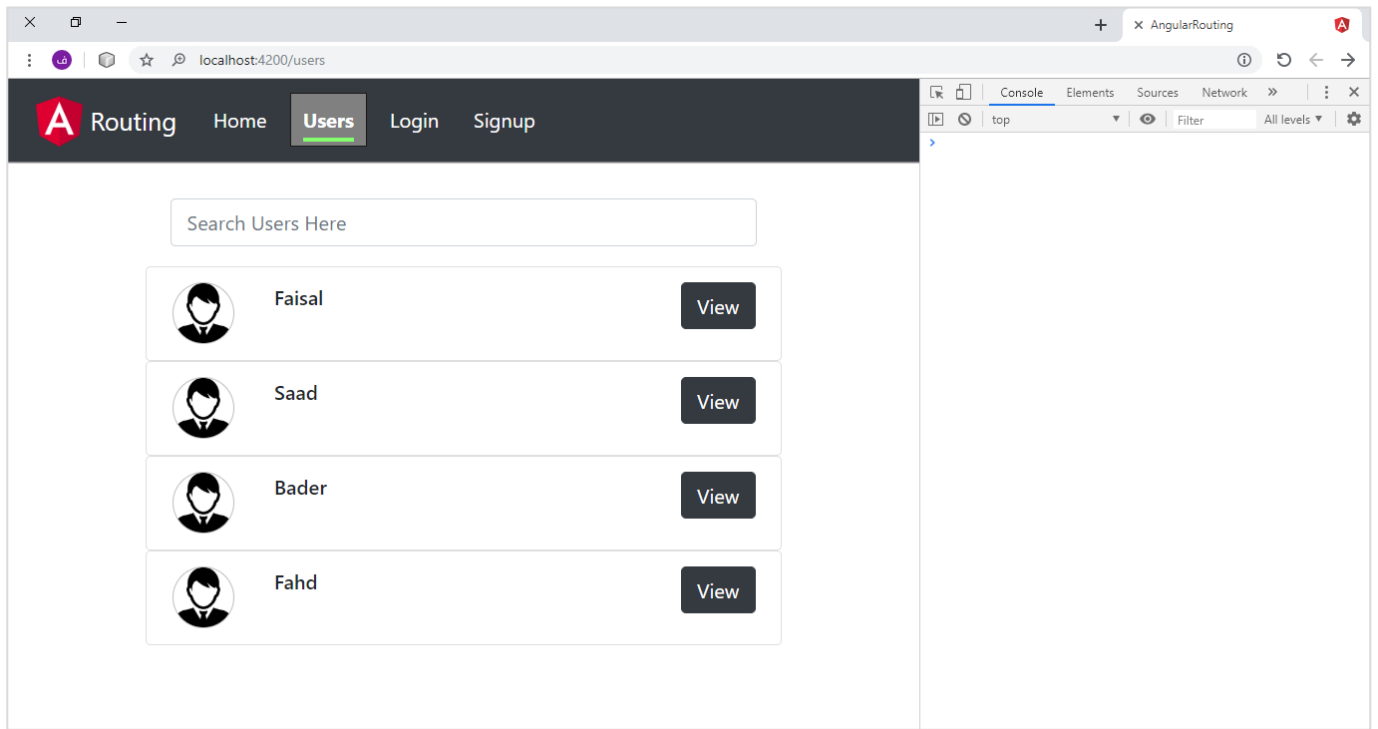
```
1. .is-active{
2.     background-color: rgb(241, 237, 237);
3. }
```

وأخيراً في ملف html نقوم بالاستفادة من ميزة class binding بحيث نضيف كلاس css السابق فقط في حال أن قيمة المتغير selectedUser مساوية لقيمة الخاصية name الموجودة في الكائن user (ولمعرفة أكثر عن Class Binding الرجاء مراجعة الكتاب الثاني من هذه السلسلة Angular Components and Services)، كالتالي:

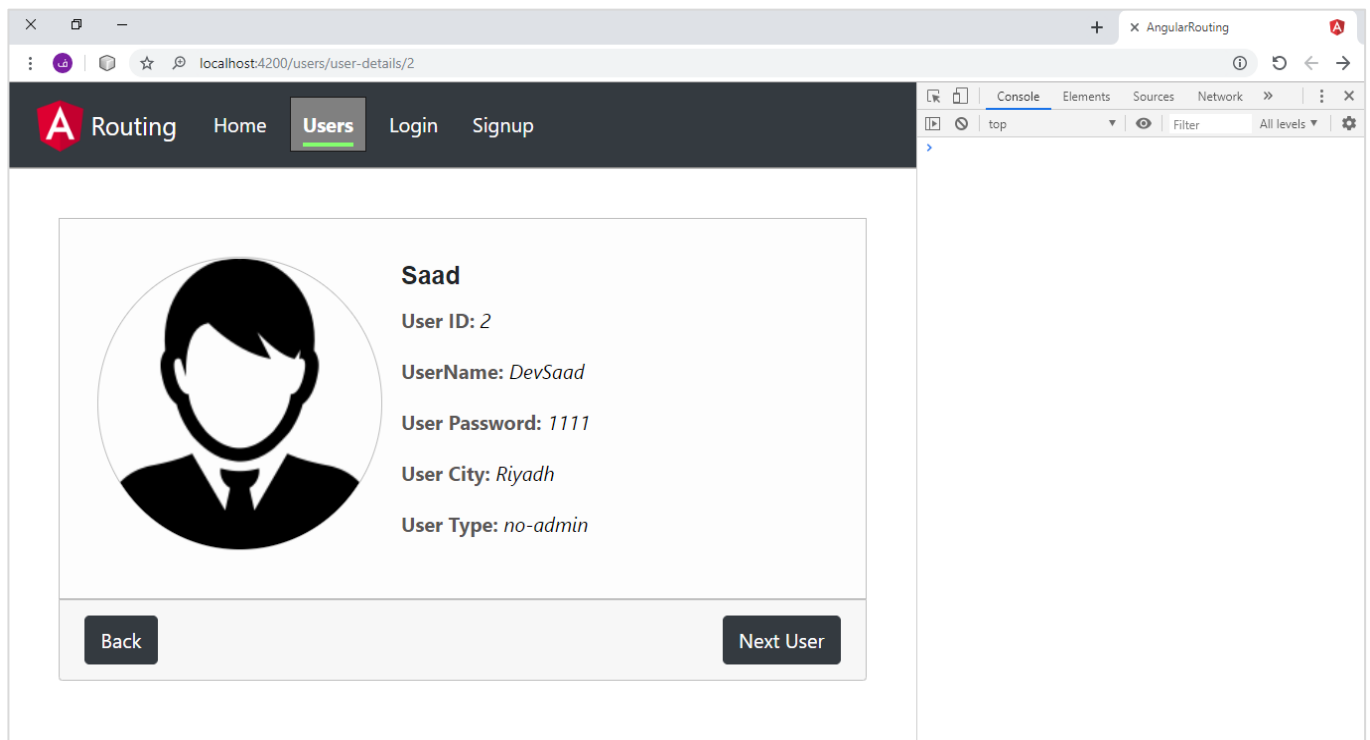
ملف users-list.component.html

```
1. <div class="container" style="padding-top: 1rem">
2.   <ul class="list-group" *ngFor="let user of users$ | async">
3.     <li class="list-group-item" [class.is-active]="selectedUser === user.name">
4.       <span class="pull-left ">
5.         
11.     </span>
12.     <h6 style="padding-left: 2rem; display: inline">{{ user.name }}</h6>
13.     <span class="pull-right">
14.       <button
15.         class="btn btn-dark"
16.         style="display: inline"
17.         (click)="viewDetails(user.userId)">View</button>
18.     </span>
19.   </li>
20. </ul>
21. </div>
```

الآن لنحفظ التعديلات ونذهب إلى المتصفح لنرى النتيجة:

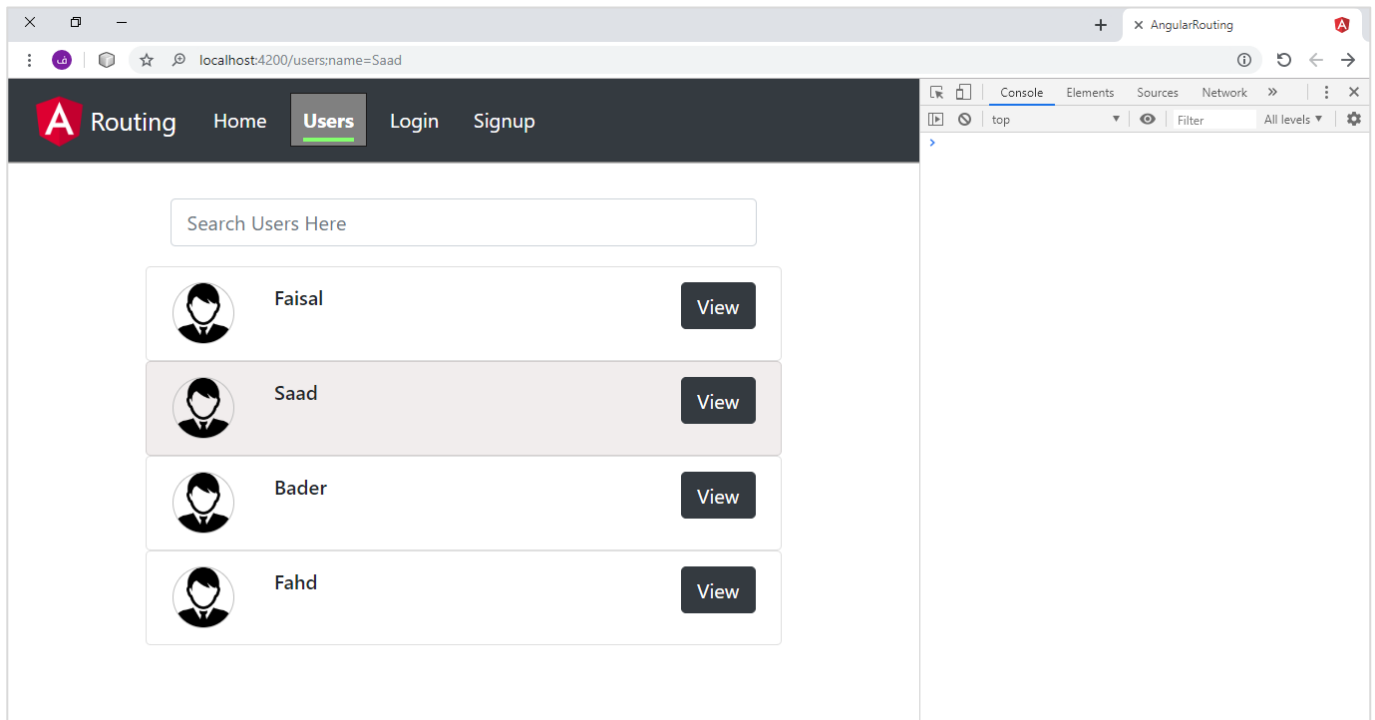


لنختار أحد المستخدمين لاستعراض بياناته:

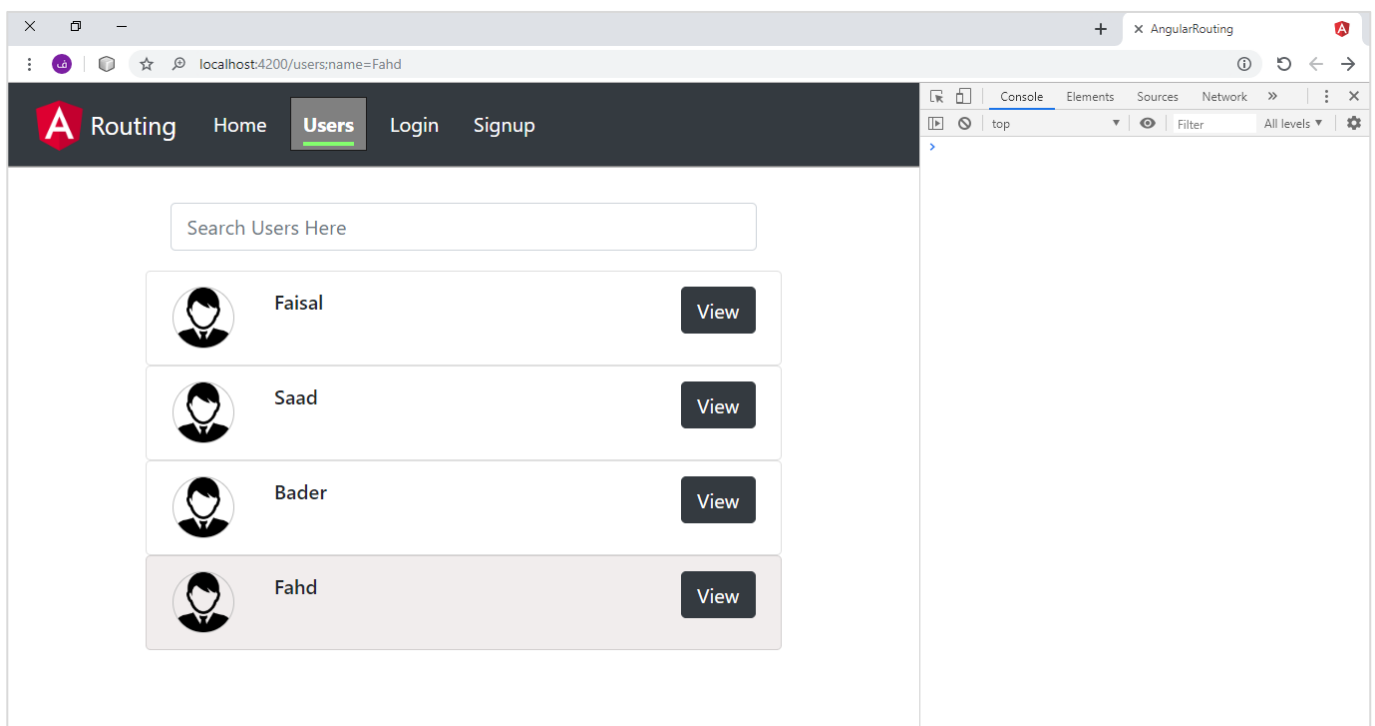


الآن لنقوم بالضغط على زر Back، ونرى النتيجة:





نلاحظ أنه تم تمييز المستخدم الذي تم استعراض بياناته عن بقية المستخدمين الآخرين، لنقوم باختيار مستخدم آخر لاستعراض بياناته ومن ثم نضغط على زر Back:

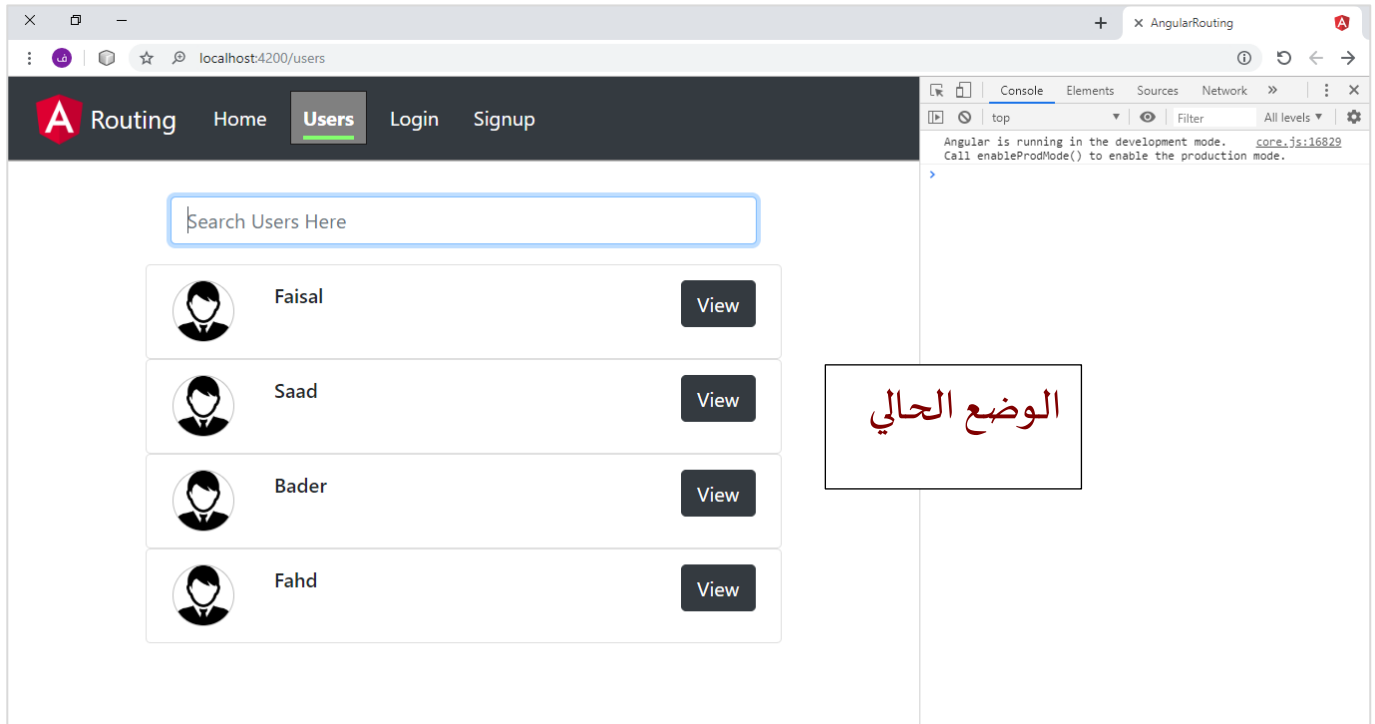


ولو نلاحظ الرابط لوجدنا أن هذا النوع من البارامتر يتم فصله عن path بفاصلة منقوطة ( ;) وبعد الفاصلة key وقيمة هذا الkey.

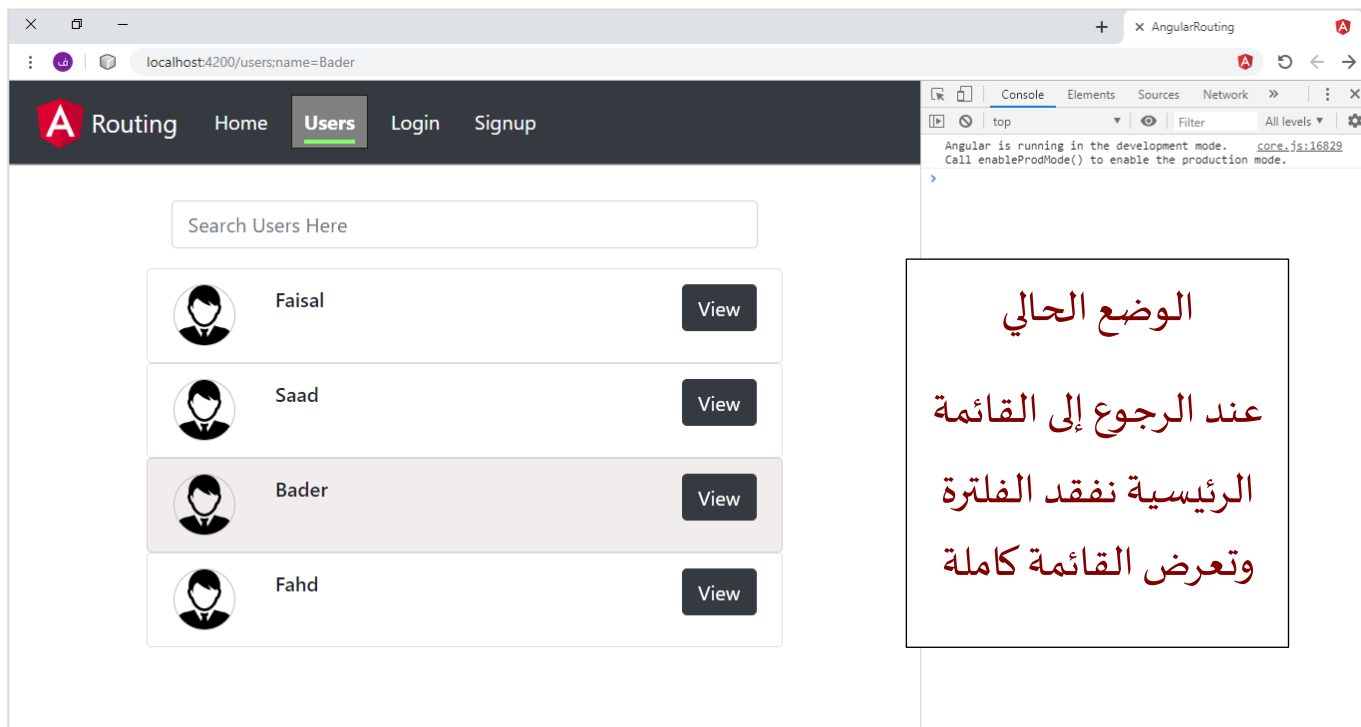
وبذلك نكون أنهينا هذا الجزء وسوف ننتقل الآن إلى آخر نوع وهو Query Parameters.

## 4.2. Query Parameters

وهذا النوع مشابه لـ Optional Parameters حتى أن بعض المراجع لا تفرق بينهما، ولكن أرى أنه هنالك بعض الفروقات مع التشابه بينهما، فمن أوجه التشابه انهما كلاهما اختياري أي بمعنى لا يحتاجان إلا تهيئة في routes، وأيضاً كلاهما تُرسلان البارامترات على هيئة كائن object، ولكن الفرق الجوهرى أن Query Parameters تسمح بإرسال البارامترات إلى component معين واسترجاعها إلى نفس component ومشاركة هذه البارامترات في أكثر من route بعكس Optional التي تسير في اتجاه واحد فقط، ومن اختلافاتها الأخرى أن Query لها دوالها الخاصة بها بعكس Optional التي تشارك مع الدوال مع Required Parameters، وايضاً يسبقها علامة الاستفهام (?) بعكس Optional التي يسبقها الفاصلة المنقوطة (:). كما أن لها مجموعة من التقنيات التي سوف نشرحها عملياً لكي تتوضح أكثر من السرد النظري، ومن ناحية تطبيقاتها العملية كثيرة ولعل أبسطها ومتوافق مع المشروع الذي نعمل عليه هو أنه في حالة البحث عن مستخدم معين واختيار أحد المستخدمين لمشاهدة تفاصيل بياناته وعند الرجوع فأن نتيجة البحث تختفي وترجع القائمة وتعرض جميع المستخدمين، لذلك نريد تعديل هذا الأمر بحيث عند الرجوع تبقى نتيجة البحث على حالها والقائمة مفلترة بحسب ما هو موجود في صندوق البحث، ويمكن حل هذا الأمر بعدة طرق منها استخدام service (كما فعلنا في الكتاب الثاني من هذه السلسلة عندما تكلمنا عن طريقة تبادل البيانات والتواصل والاتصال بين components ونفسه)، ومن الطرق أيضاً استخدام Query Parameters، وهذا الذي يخدمنا في هذا الكتاب لذلك سوف نشرحه في الأجزاء التالية.







## الوضع الحالي

عند الرجوع إلى القائمة  
الرئيسية نفقد الفلتر  
وتعرض القائمة كاملة

وهذا الوضع هو الذي اقصد به والذي نريد تغييره إلى بقاء الفلتر بعد الرجوع إلى القائمة الرئيسية وكما قلنا هنالك عدة طرق لحل هذه المشكلة منها استخدام Query Parameters.

### 1.4.2. إرسال Query Parameters عن طريق URL:

وللقيام بهذا الأمر اول خطوة نقوم بها هي إرسال قيمة البحث الموجودة في صندوق البحث وحفظها في الرابط، ويتم إرسالها في حالتنا هذه عندما يتم الضغط على زر view نريد إرسال قيمة البحث ومن ثم استرجاعها مرة أخرى، وللقيام بهذا الأمر نذهب إلى ملف ts لـ UsersListComponent وفي الدالة التي تُنفذ عندما يتم الضغط على زر View واسمها viewDetails() نكتب الكود التالي:

ملف users-list.component.ts

```
1. import { Component, OnInit, Input, OnChanges } from '@angular/core';
2. import { Observable, of } from 'rxjs';
3. import { Users } from 'src/app/users-data/users';
4. import { Router, ActivatedRoute } from '@angular/router';
5.
6. @Component({
7.   selector: 'app-users-list',
8.   templateUrl: './users-list.component.html',
9.   styleUrls: ['./users-list.component.css']
10. })
11. export class UsersListComponent implements OnInit, OnChanges {
12.   @Input() users$: Observable<Users[]>;
13.   @Input() fullUsersList$: Observable<Users[]>;
14.   @Input() inputSearch: string;
15.   selectedUser: string;
16.   constructor(private router: Router, private activatedRoute: ActivatedRoute) { }
17.
18.   ngOnInit() {
```

```

19.   this.selectedUser = this.activatedRoute.snapshot.paramMap.get('name');
20. }
21.
22. ngOnChanges () {
23.   if (this.inputSearch) {
24.     this.users$.subscribe(data => {
25.       const result = data.filter(user =>
26.         user.name.toLocaleLowerCase().indexOf(this.inputSearch.toLocaleLowerCase()) !== -1);
27.       this.users$ = of(result);
28.     });
29.   } else {
30.     this.users$ = this.fullUsersList$;
31.   }
32. }
33.
34. viewDetails(id: number) {
35.   this.router.navigate(['users/user-details', id], {
36.     queryParams: { searchText: this.inputSearch }
37.   });
38. }
39.

```

الدالة viewDetails في الاسطر من 34 إلى 37 قمنا بتعديل الأكواد وإضافة كائن وهذا الكائن يحمل key ذو الاسم queryParams (الاسم ثابت ويجب كتابته كما هو) وهذا key أيضاً يحمل كائن وفي هذا الكائن نقوم بكتابة جميع بارامترات Query التي نريد إرسالها وفي حالتنا هنا لا نريد إرسال إلا بارامتر واحد وهو القيمة الموجودة في صندوق البحث لذلك قمنا بإعطائها key ذو الاسم searchText (لك حرية اختيار الاسم الذي تريده) والقيمة هي ما يحمله المتغير inputSearch من القيمة القادمة لنا من الـ UsersComponent الأب المربوطة في صندوق البحث عن طريق تقنية -Tow Way-Databinding، ونستطيع إرسال العدد الذي نريده من البارامترات فكل المطلوب هو وضع فاصلة ومن ثم كتابة key والقيمة له: { searchText: this.inputSearch, key1: value1, key2: value2, ...etc. }

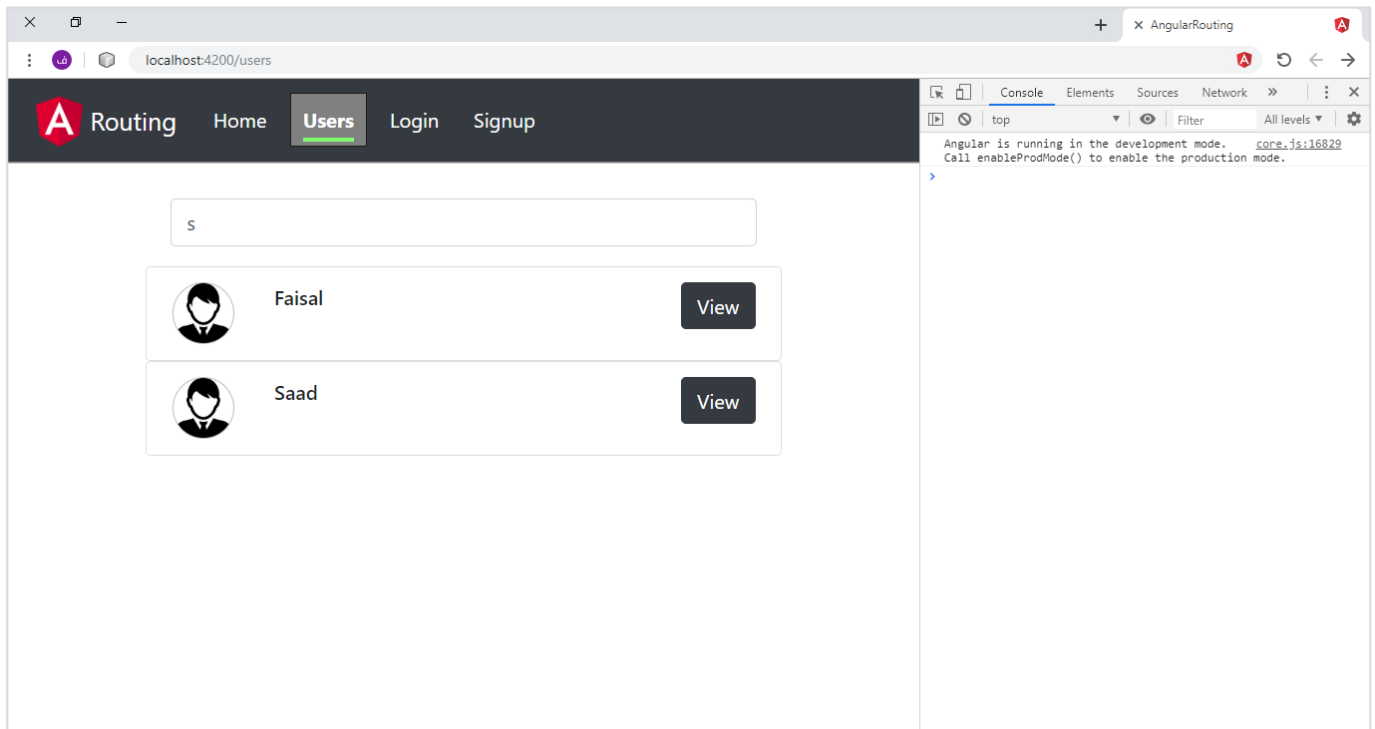
ملاحظة: قمنا هنا بالتعامل معها برمجياً أما في حال أردنا التعامل معها وإرسال هذه البارامترات عن طريق template فلا بد من كتابة routerLink في العنصر كما كنا نفعل سابقاً ومعه نكتب [queryParams] أما كتابة queryParams فقط لوحده فلن يعمل ويُظهر لك رسالة خطأ، ولنفرض أن العنصر الذي سوف نستخدمه لإرسالها button، فسوف يكون الكود كالتالي:

```

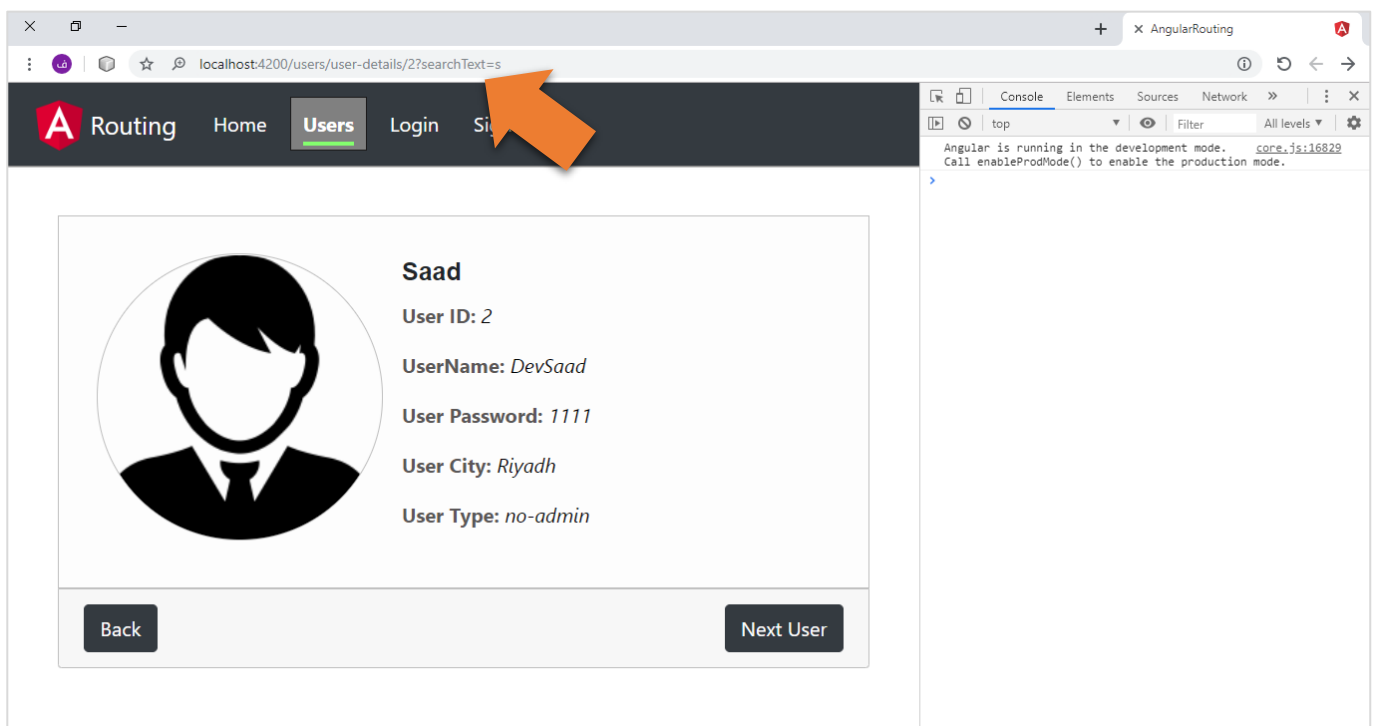
<button [routerLink] = ["../users/user-details", user.userId]" [queryParams] = "{searchText: inputSearch}">
  View
</button>

```

الآن لنقوم بحفظ التعديلات ونذهب إلى المتصفح لمشاهدة النتيجة:

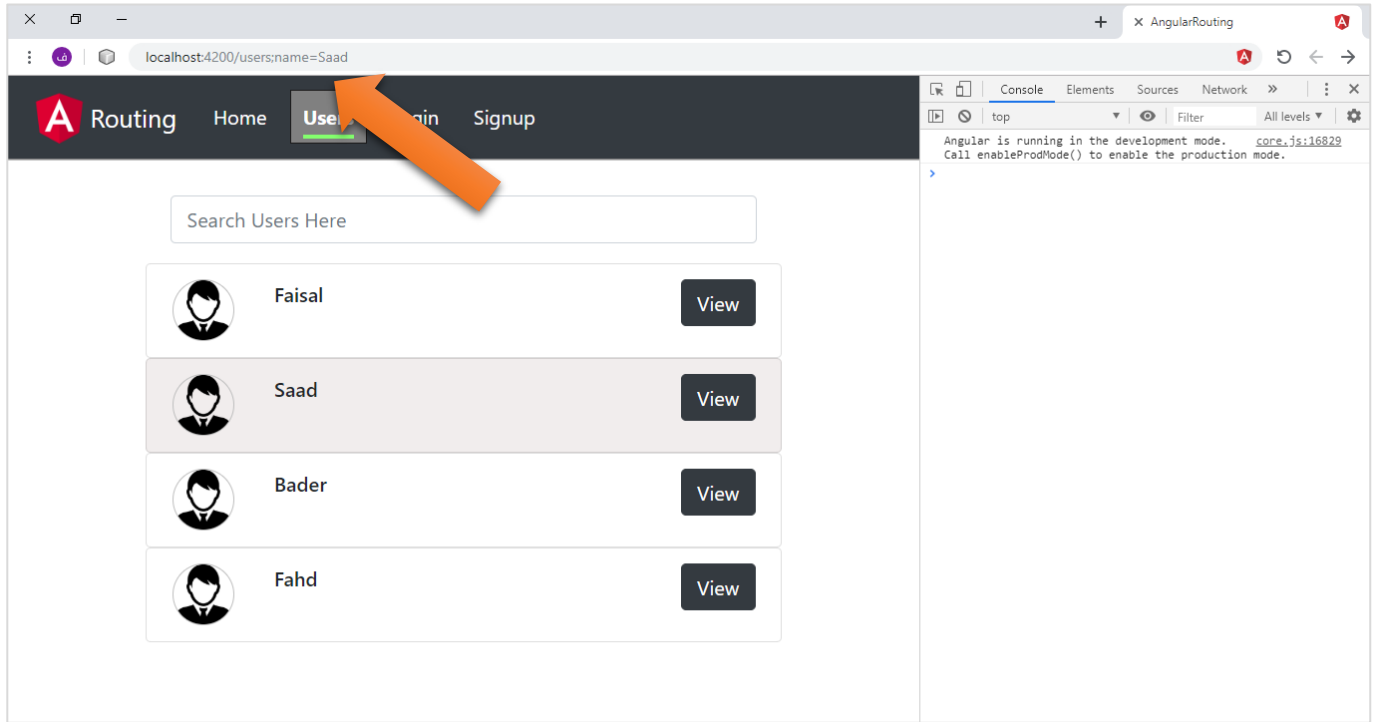


في حالة البحث نلاحظ تم فلتر القائمة، الآن لنختار أحد المستخدمين ونرى النتيجة، كالتالي:



نلاحظ تم تمرير البارامتر في الرابط على هيئة Query لأنه يبدأ بعلامة (?) بعكس Optional الذي يبدأ بفاصلة منقوطة، ويحمل هذه البارامتر key ذو الاسم searchText والقيمة الموجودة في صندوق البحث s وهي بالقيمة التي كتبناها في مربع البحث.

الآن لنضغط على زر رجوع ونرى ماذا سوف يحدث:



نلاحظ تم فقدان البارامترات الذي التي قمنا بتمريرها قبل قليل ولا يوجد إلا Optional Parameters والسبب في ذلك يعود إلى الخاصية **queryParamsHandling** حيث أن هذه الخاصية تحمل ثلاث قيم، هي:

**null**: وهي القيمة الافتراضية لها، ولا تقوم بعمل أي شيء.

**preserve**: وهي تجعل الخاصية تحفظ البارامترات في الرابط ولا تفقدها في حال الانتقال بين routes المختلفة سواء كان هذا route ينقلنا إلى component آخر أو بنفس component ولكن يغير البيانات الموجودة به.

**merge**: وهي تقوم بنفس عمل **preserve** ولكن هنا نستطيع دمج Query Parameter جديد مع الموجود حالياً.

الآن لنقوم بتطبيق ذلك عملياً، ونريد إضافة هذه الخاصية في الزر الخاص بالرجوع إلى قائمة المستخدمين في UserDetailsComponent، لذلك لنذهب إلى ملف html لهذا component ونضيف الكود التالي:

ملف `user-details.component.html`

```

1. <div class="card">
2.   <div class="card-body">
3.     <ul>
4.       <li>
5.         
6.         <h3>{{ user$.name }}</h3>
7.         <p>
8.           User ID: <span>{{ user$.userId }}</span>
9.         </p>
10.        <p>
11.          UserName: <span>{{ user$.userName }}</span>
12.        </p>
13.        <p>
14.          User Password: <span>{{ user$.password }}</span>
15.        </p>

```

```

16.     <p>
17.         User City: <span>{{ user$.userCity }}</span>
18.     </p>
19.     <p>
20.         User Type: <span> {{ user$.userType }}</span>
21.     </p>
22. </li>
23. </ul>
24. </div>
25. <div class="card-footer">
26.     <a
27.         class="btn btn-dark"
28.         [routerLink]="['/users', { name: user$.name }]"
29.         queryParamsHandling="preserve">
30.         Back
31.     </a>
32.     <button
33.         class="btn btn-dark pull-right"
34.         (click)="viewNextUser()">
35.         Next User
36.     </button>
37. </div>
38. </div>

```

نلاحظ في الاسطر من 26 إلى 31 الخاصة بالعنصر <a> والذي يقوم بإرجاعنا إلى قائمة المستخدمين عن طريق [routerLink]، قمنا بإضافة الخاصية queryParamsHandling في السطر 29 وأسندنا لها القيمة preserve، وفي حال كان هنالك Query Parameter نريد إرساله عن طريق هذا route فأولاً نضيف هذه البارامترات كما تعلمنا سابقاً في template لهذه component ومن ثم نقوم بتغيير القيمة إلى marge بدلاً من preserve، كالتالي:

```

<a
  class="btn btn-dark"
  [routerLink]="['/users', { name: user$.name }]"
  [queryParams]="[{ key: value }]"
  queryParamsHandling="marge">
  Back
</a>

```

مع العلم أننا سوف نستخدم القيمة preserve اما marge فقط كتبناها لتوضيح فقط.

وبذلك سوف نحافظ على البارامترات في الرابط عند أي تغيير يطرأ على route في حال الضغط على هذا الزر، ولكن هنالك زر ايضاً يقوم بتغيير route لدينا وسوف نفقد هذه البارامترات وهو Next User، صحيح أننا بنفس component ولكن route في الرابط يتغير، حيث مع كل ضغطة على هذه الزر يقوم بتغيير route على حسب id ويظهر البيانات المقابلة لهذا id. وكما هو معلوم أن أي تغيير على يطرأ على route سوف تُفقد هذه البارامترات لذلك لا بد من كتابة هذه الخاصية ايضاً



في هذا الزر للمحافظة على البارامترات، وبما أننا قبل قليل كتبنا هذه الخاصية في template سوف نكتبها هذه المرة برمجياً لكي يكون لديك عزيزي القارئ المتعلم المام كامل بجميع طرق التعامل مع هذه التقنيات.

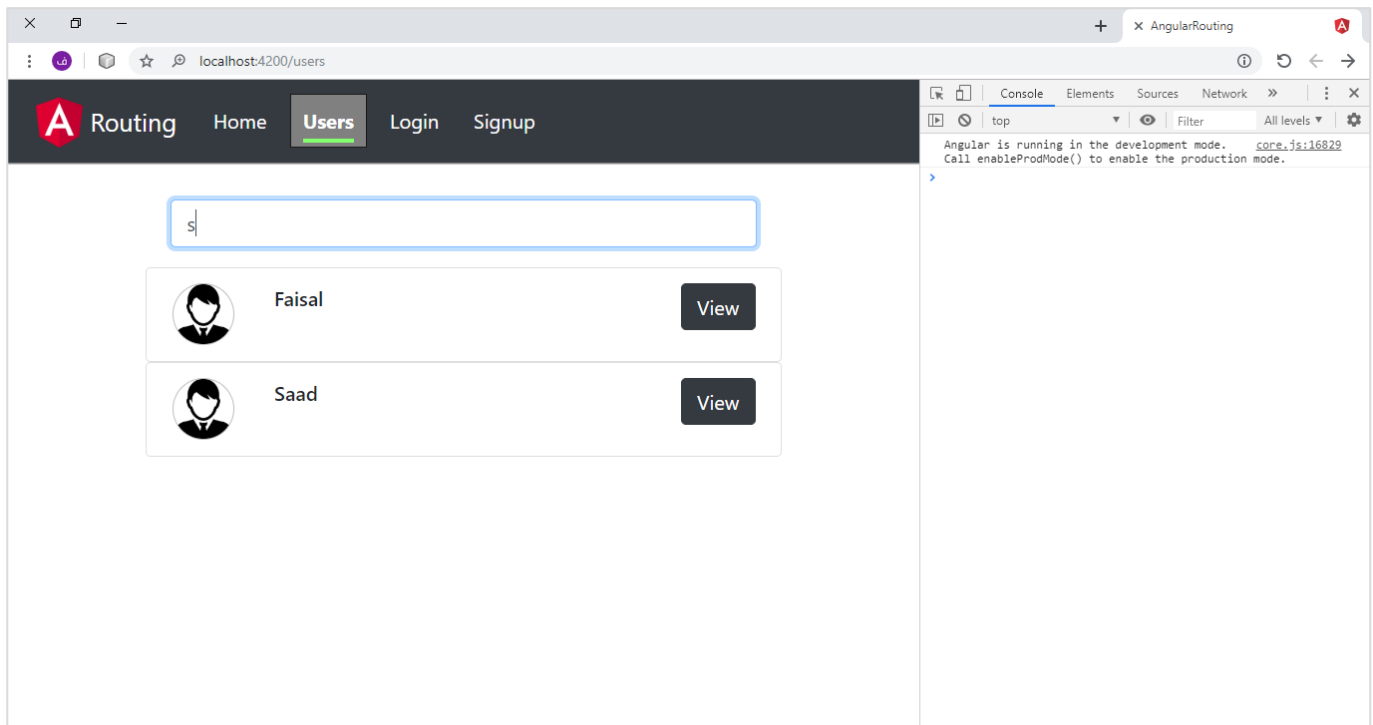
لذلك لنذهب إلى الدالة التي تُنفذ عند الضغط على هذا الزر واسمها viewNextUser() في ملف ts لهذا component ونضيف هذه الخاصية، كالتالي:

#### ملف user-details.component.ts

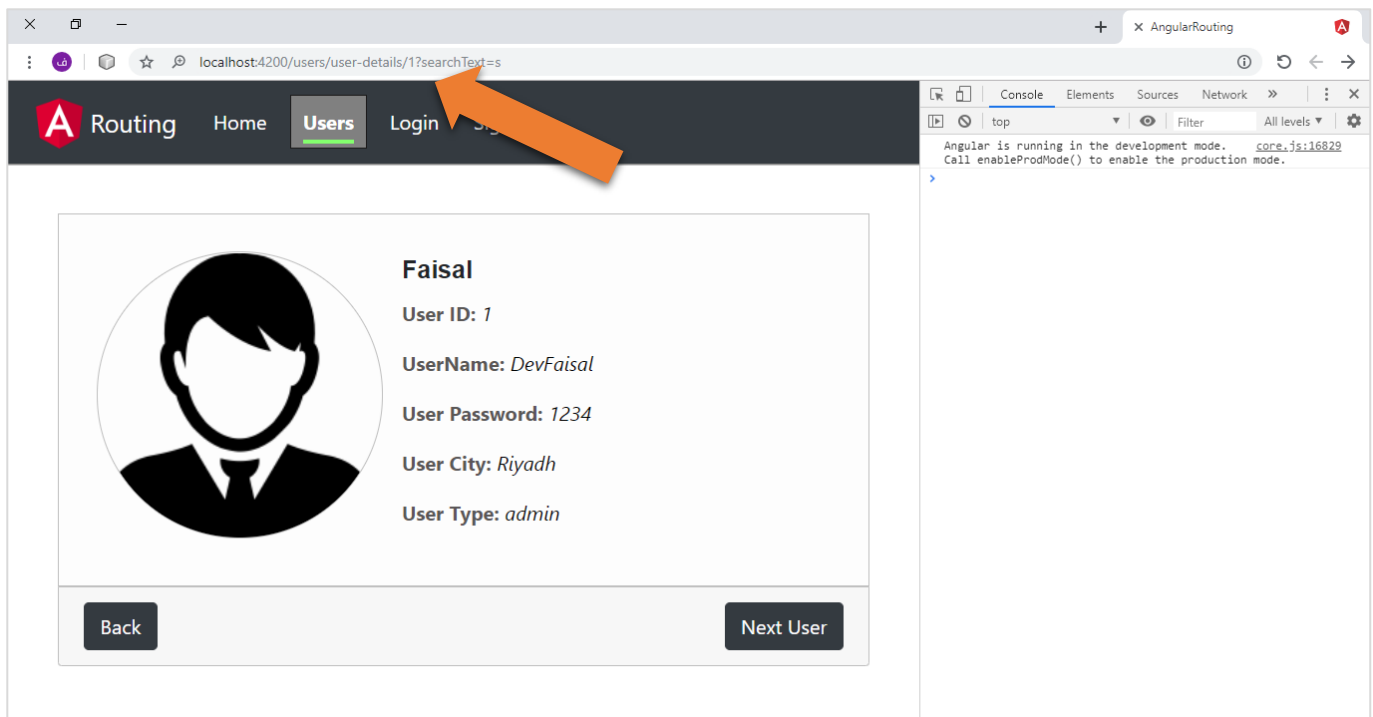
```
1. import { Component, OnInit, OnDestroy } from '@angular/core';
2. import { ActivatedRoute, Router } from '@angular/router';
3. import { UsersService } from 'src/app/users-data/users.service';
4. import { Subscription } from 'rxjs';
5. import { Users } from 'src/app/users-data/users';
6.
7. @Component({
8.   selector: 'app-user-details',
9.   templateUrl: './user-details.component.html',
10.  styleUrls: ['./user-details.component.css']
11.})
12. export class UserDetailsComponent implements OnInit, OnDestroy {
13.   user$: Users;
14.   private subscription: Subscription;
15.   private id: number;
16.   constructor(
17.     private activeRouter: ActivatedRoute,
18.     private usersService: UsersService,
19.     private router: Router) { }
20.   ngOnInit() {
21.     // this.id = +this.activeRouter.snapshot.params.id;
22.     // this.subscription = this.usersService.getUserById(this.id).subscribe(user => {
23.     //   this.user$ = user;
24.     // });
25.     this.activeRouter.paramMap.subscribe(param => {
26.       this.id = +param.get('id');
27.       this.subscription = this.usersService.getUserById(this.id).subscribe(data => {
28.         this.user$ = data;
29.       });
30.     });
31.   }
32.   ngOnDestroy() {
33.     this.subscription.unsubscribe();
34.   }
35.   viewNextUser() {
36.     this.subscription = this.usersService.getAllUsers().subscribe(data => {
37.       this.id === data.length ? this.id = 1 : this.id = this.id + 1;
38.     });
39.     this.router.navigate(['users/user-details', this.id],
40.       {queryParamsHandling: 'preserve'})
41.   }
42. }
43. }
```

ما يهمنا هنا هو ما هو موجود في السطر 40 حيث قمنا بإضافة كائن وهذا الكائن يحمل key وهي الخاصية queryParamsHandling والقيمة preserve.

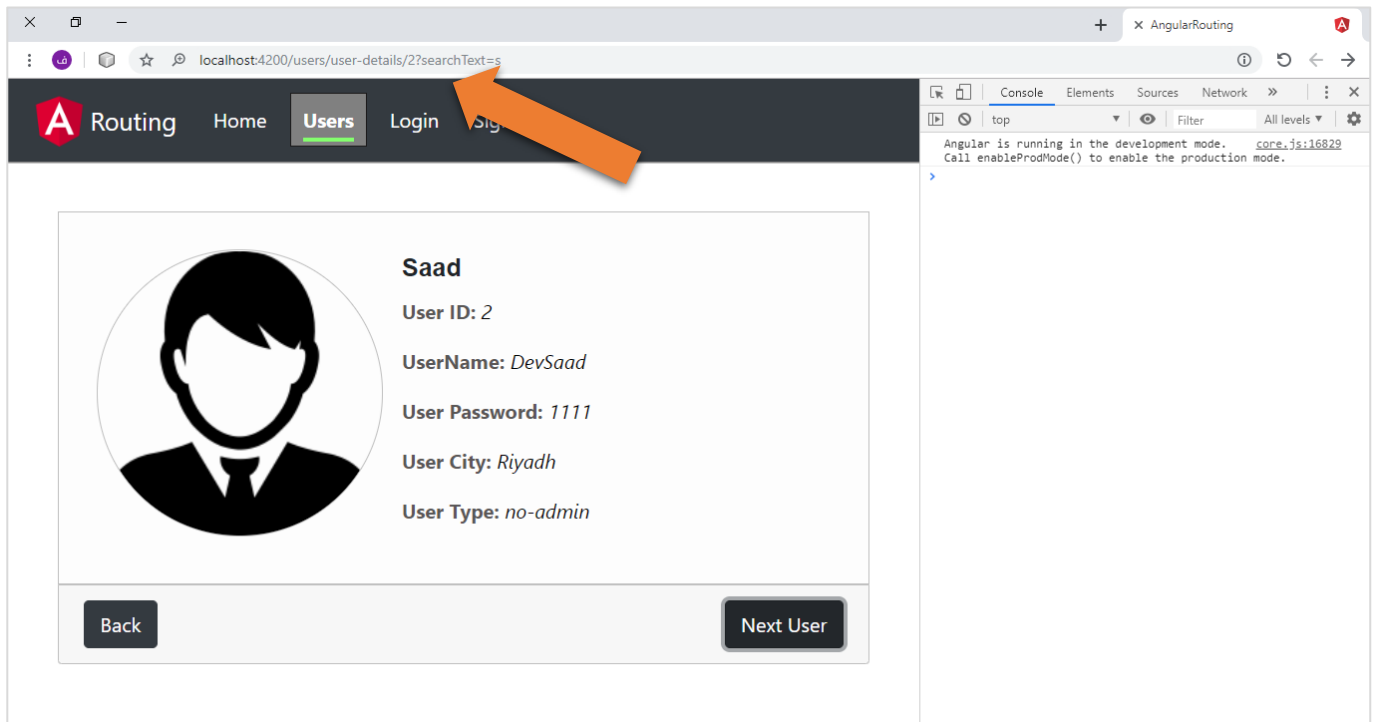
الآن لنحفظ هذه التعديلات ونذهب إلى المتصفح لنرى النتيجة:



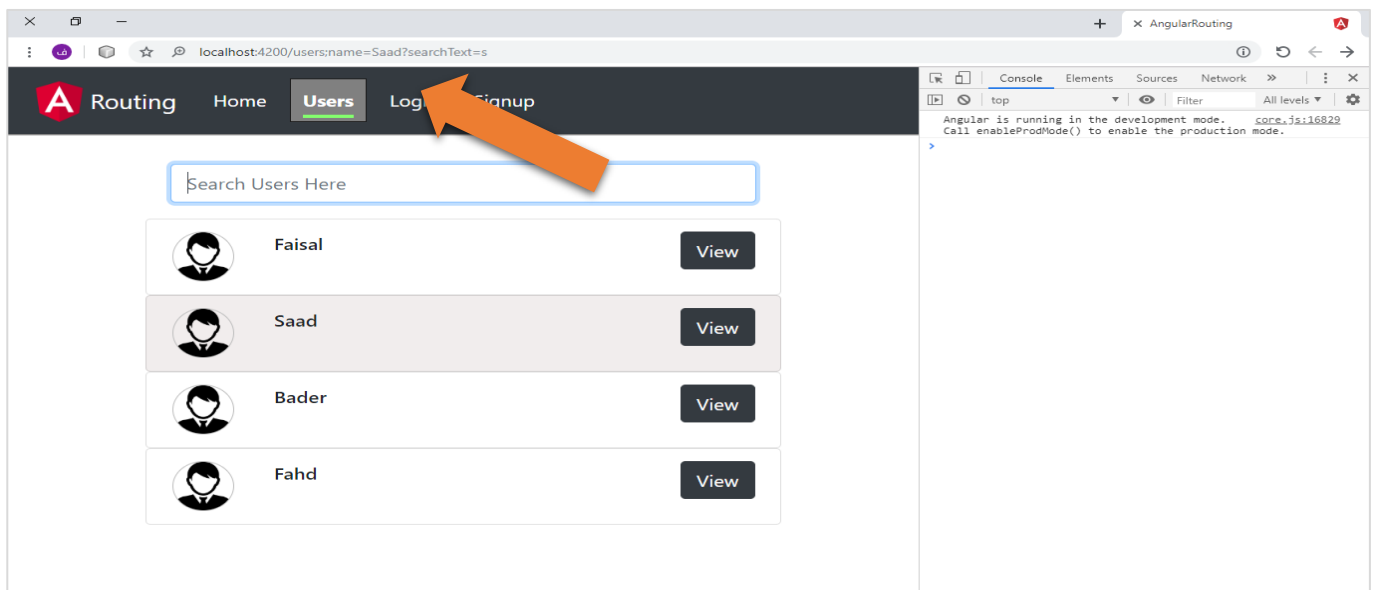
في حالة البحث، لنقوم الآن بضغط على أحد المستخدمين:



في حال اختيار أحد المستخدمين لعرض بياناته، والرابط يحمل البارامترات، لنقوم الآن بالضغط على زر Next User:



نلاحظ مع تغير route id مختلف ولكن إلى الآن الرابط محافظ على البارامترات Query Parameters، الآن لنقوم بالضغط على زر الرجوع ونرى النتيجة:



نلاحظ أن الرابط لا يزال محافظ على البارامترات، إلى الآن الوضع جيد وبقي علينا فقط قراءة هذه البارامترات لكي نُعيد فلتر القائمة بحسب القيمة الموجودة في هذه البارامترات، وهو ما سوف نوضحه بإذن الله في النقطة القادمة.

ملاحظة: هذا هو الفرق الجوهرى الذي تكلمت عنه بين Query Parameters و Optional Parameters حيث هنا كما هو ملاحظ نستطيع مشاركة البارامترات وإرسالها واسترجاعها في عدد من routes سواء كان هذا route ينقلنا إلى component معين أو يكون بنفس component مع تغيير البيانات التي يحتويها، فكل الذي علينا أن نقوم به هو وضع الخاصية queryParamsHandling لأي زر أو عنصر أو دالة برمجية تقوم بتغيير route وأُسناد القيم سواء سواء preserve أو marge بحسب الاحتياج.

## 2.4.2. قراءة Query Parameters من URL:

وآخر جزء سوف نتكلم عنه في Query Parameters هو كيفية قراءة هذه البارامترات، وقد تكلمنا سابقاً أن لقراءة البارامترات سواء كانت Required او Optional نستخدم service ذات الاسم ActivatedRoute والدالتين التي تحملها سواء snapshot او paramMap ولكن في الحقيقة هنا الوضع يختلف قليلاً، فالدالة الأولى سوف نستمر باستخدامها ولكن سوف نستخدم الخاصية paramMap والتي تحتوي بداخلها على الدوال التي تكلمنا عنها سابقاً get و getAll و..الخ، أما الدالة الثانية paramMap التي نعمل لها subscribe وتُعيد observable فنستبدلها بالخاصية paramMap التي تعمل نفس عمل هذه الدالة، كما في الشكل التالي:



وسوف أقوم بقراءة هذه البارامترات باستخدام الدالة snapshot في UsersComponent الأب والسبب في ذلك يرجع لعدة أمور أولاً أن كلا components UsersListComponent والأبن UsersComponent الأب يحملون نفس route ذو ال path الذي قيمته users، وثانياً وهو المهم أن UsersComponent موجود بداخله صندوق البحث وقد مررنا قيمته عند طريق المتغير inputSearch إلى UsersListComponent الأب، ونحن هنا نريد قراءة البارامتر ووضعه في صندوق البحث وبما أن هذا الصندوق مربوط بالمتغير فتلقائياً سوف تتغير قيمة هذا المتغير والتي بشكل طبيعي سوف تؤثر على القائمة، وبنفس الوقت أي تغير بقيمة المتغير سوف يؤثر على المحتوى في صندوق البحث.

لذلك لنذهب إلى ملف ts الخاص بالUsersComponent وكتابة الكود التالي بحيث نسند البارامتر القادم إلى المتغير inputSearch، كالتالي:

ملف users.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { Observable } from 'rxjs';
3. import { Users } from '../users-data/users';
4. import { UsersService } from '../users-data/users.service';
5. import { ActivatedRoute } from '@angular/router';
6.
7. @Component({
8.   selector: 'app-users',
9.   templateUrl: './users.component.html',
10.  styleUrls: ['./users.component.css']
11. })
12. export class UsersComponent implements OnInit {
13.   userList$: Observable<Users[]>;
14.   fullUsersList$: Observable<Users[]>;
15.   inputSearch: string;
```

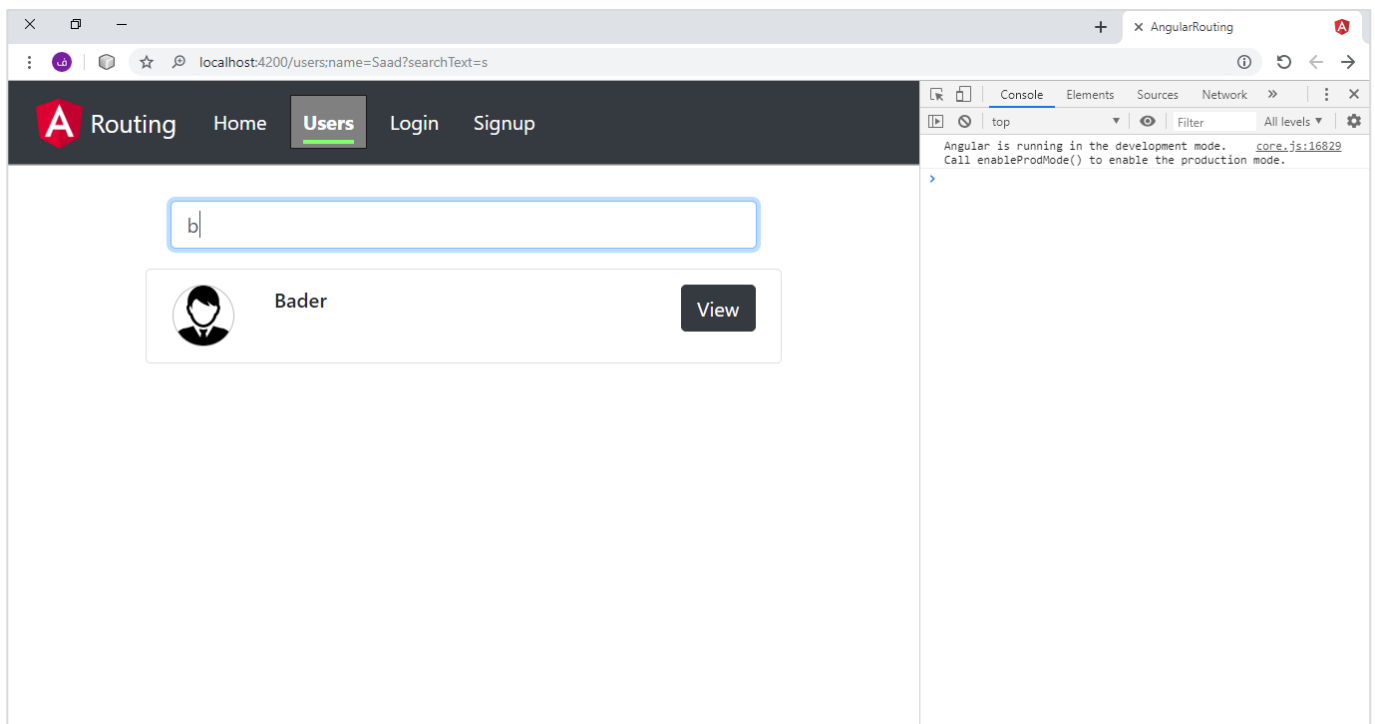
```

16.
17. constructor(private users: UsersService, private activatedRoute: ActivatedRoute) { }
18.
19. ngOnInit() {
20.   this.usersList$ = this.users.getAllUsers();
21.   this.fullUsersList$ = this.users.getAllUsers();
22.   this.inputSearch = this.activatedRoute.snapshot.queryParamMap.get('searchText');
23. }
24.
25. }

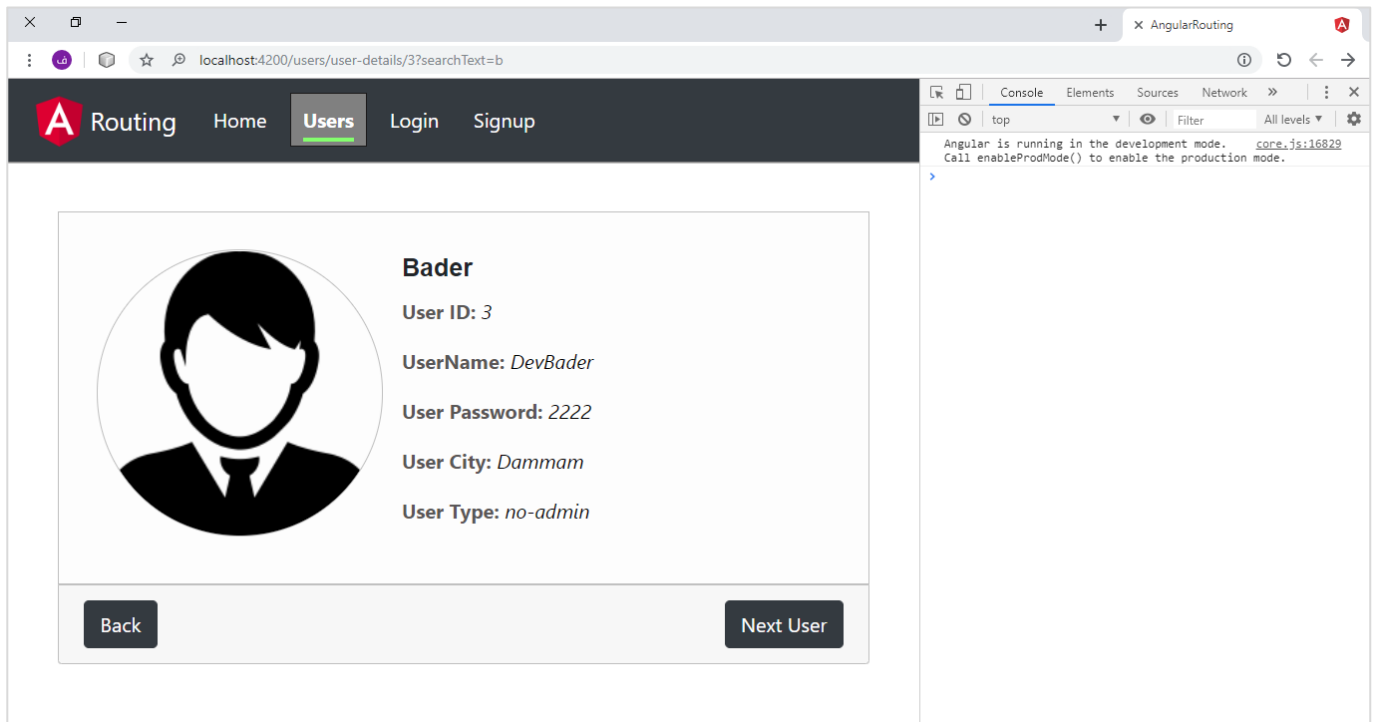
```

ما يهمنا هو ما هو موجود في السطر 22 حيث قمنا باستخدام service ذات الاسم ActivatedRoute بعدما استدعيناها في السطر 5 وعملنا لها inject في السطر 17 في المتغير activatedRoute (بداية الاسم حرف صغير بعكس اسم service ولك حرية اختيار الاسم الذي تُريده) وعن طريق الدالة snapshot وصلنا إلى الدالة queryParamMap وعن طريقها وصلنا إلى الدالة get التي تقرأ Query Parameters فقط، واسندنا قيمة هذه البارامتر إلى المتغير inputSearch وأي تغيير يطرأ على هذا البارامتر سوف يؤثر على محتوى صندوق البحث وبنفس الوقت سوف يؤثر على المتغير الموجود في UsersListComponent الأبني والذي يحمل أيضاً نفس الاسم inputSearch.

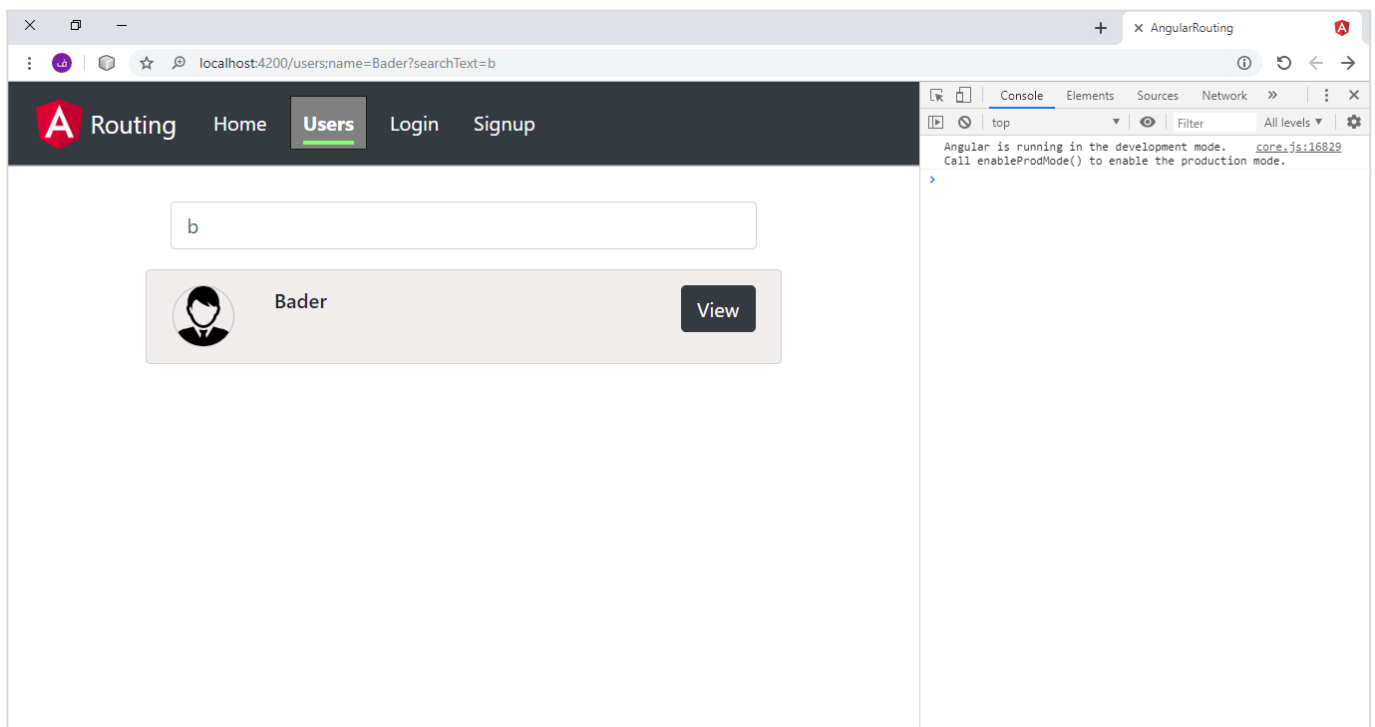
الآن لنقوم بحفظ التعديلات ونرى النتيجة في المتصفح:



لنختار المستخدم بعد فلترة القائمة بحسب القيمة الموجودة في صندوق البحث وهي الحرف b:



لنضغط على زر رجوع ونلاحظ ان البارامتر يحمل القيمة b في الرابط:



نلاحظ أن القائمة لاتزال على وضعها وذلك بسبب أننا قمنا بقراءة البارامتر في الرابط واسناد قيمته إلى المتغير المربوط في صندوق البحث.

وبذلك نكون أنهينا هذا النوع المهم من البارامترات وسوف ننتقل الآن إلى نوع آخر قليل الاستخدام ولكن لابد من الإشارة إليه ولو في عجالة.

## 5.2. Fragments Parameters

وهي من أنواع البارامترات قليلة الاستخدام ولكن لها استخداماتها ولعل من أشهر استخداماتها ان تكون كبيانات اضافية تُرسل مع routed كإرسال الـ Access Token الخاص بمستخدم معين عن طريق route، وايضاً من استخداماتها الأخرى هو الانتقال إلى نقطة معينة في الصفحة لعرض عنصر html معين عن طريق الـ id الخاص به، بمعنى لو كان لدينا صفحة فيها كم كبير من البيانات وهنالك قائمة جانبية ونريد إذا ضغط المستخدم على هذه القائمة الانتقال إلى جزء محدد وإذا ضغط على عنصر آخر من هذه القائمة ينتقل إلى جزء محدد آخر من هذه الصفحة، وجميع هذه الأجزاء المحدد هي عبارة عن عناصر html لها id وعن طريق هذا id نصل لها بكتابة اسم البارامتر بنفس اسم الـ id لهذا العنصر، وهذا هو المثال الذي سوف نضيفه إلى مشروعنا لشرح مفهوم هذا النوع من البارامترات.

اما من ناحية مفهومها فهي ايضاً اختيارية ولا تحتاج إلى تهيئة في route وتُرسل على شكل كائن ولكن تأخذ قيمة واحدة فقط بعكس البارامترات الأخرى التي ممكن إرسال أكثر من قيمة على شكل كائن، كما أنها تبدأ بعلامة # في الرابط، ولا يمكن مشاركتها بأكثر من routes.

اما الآن لنقوم بتطبيق المثال لكي يتضح المفهوم، وكما قلنا الذي نريده هو الانتقال إلى جزء محدد من الصفحة، لذلك سوف نضيف عنصر html في ملف Template الخاص بـ UserDetailsComponent بين هل المستخدم مفعّل أم غير مفعّل ونعطي هذا العنصر id وليكن active-user، كالتالي:

ملف user-details.component.html

```
1. <div class="card">
2.   <div class="card-body">
3.     <ul>
4.       <li>
5.         
6.         <h3>{{ user$.name }}</h3>
7.         <p>
8.           User ID: <span>{{ user$.userId }}</span>
9.         </p>
10.        <p>
11.          UserName: <span>{{ user$.userName }}</span>
12.        </p>
13.        <p>
14.          User Password: <span>{{ user$.password }}</span>
15.        </p>
16.        <p>
17.          User City: <span>{{ user$.userCity }}</span>
18.        </p>
19.        <p>
20.          User Type: <span>{{ user$.userType }}</span>
21.        </p>
22.      </li>
23.    </ul>
24.  </div>
25. <div class="card-footer">
```

```

26. <a
27.   class="btn btn-dark"
28.   [routerLink]="['/users', { name: user$.name }]"
29.   queryParamsHandling="preserve"
30. >
31.   Back
32. </a>
33. <button class="btn btn-dark pull-right" (click)="viewNextUser()">
34.   Next User
35. </button>
36. </div>
37.</div>
38.
39.<div style="height: 200px"></div>
40.<div class="div-style" id="active-user">
41.  <span class="pull-right" style="margin: 8px 7px 0 0">Active</span>
42.  <span class="span-style pull-right"></span>
43.</div>

```

ما يهمنا ما هو موجود في الأسطر من 39 إلى 34 وخصوصاً في السطر 40 حيث أضفنا id للعنصر (لك حرية اختيار الاسم الذي تُريده)، ومهم تذكر هذا id لأن Fragment Parameter سوف يكون بنفس id لهذا العنصر.

وبنفس الوقت قمنا بإضافة بعض اكواد css لإعطاء شكل جمالي للعنصر، كالتالي:

#### ملف user-details.component.css

```

1. div {
2.   margin: 20px;
3. }
4.
5. .card{
6.   border: 0;
7. }
8.
9. .card-body{
10.  background: rgba(253, 253, 253, 0.925);
11.  border: 1px solid silver;
12.  margin-bottom: 0px;
13.}
14.
15. .card-footer{
16.  border: 1px solid silver;
17.  background: rgba(246, 246, 246, 0.925);
18.  margin-top: 0px;
19.}
20.
21. ul {
22.  list-style-type: none;
23.  margin: 0;
24.  padding: 0;
25.  width: 500px;
26.}

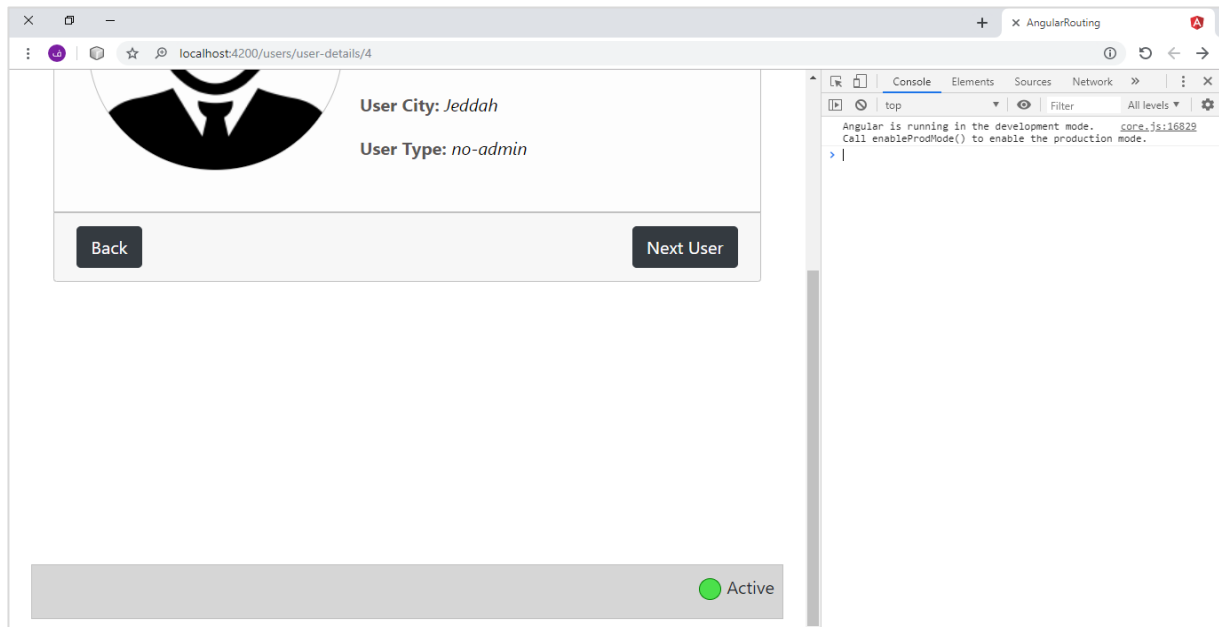
```



```
27.
28.li {
29.  padding: 10px;
30.  overflow: auto;
31.}
32.
33.li img {
34.  float: left;
35.  margin: 0 15px 0 0;
36.  border: 1px solid silver;
37.  height: 230px;
38.  border-radius: 50%
39.}
40.
41.li p {
42.  font: 200 12px/1.5;
43.  color: rgb(90, 87, 87);
44.  font-weight: bold;
45.}
46.
47.li p span {
48.  font: 200 12px/1.5;
49.  color: rgb(0, 0, 0);
50.  font-weight: normal;
51.  font-style: italic;
52.}
53.
54.h3 {
55.  font: bold 20px/1.5 Helvetica, Verdana, sans-serif;
56.}
57.
58..div-style{
59.  height: 50px;
60.  background-color: rgb(214, 214, 214);
61.  border: 1px solid silver;
62.}
63.
64..span-style{
65.  border-radius: 50%;
66.  border: 1px solid green;
67.  background-color: rgb(75, 224, 75);
68.  height: 20px;
69.  width: 20px;
70.  margin-top: 12px;
71.  margin-right: 5px;
72.}
73.
```

الرجاء مراجعة الاكواد في الاسطر من 58 إلى 72.

الآن لنحفظ التعديلات ونرى النتيجة في المتصفح:



نلاحظ الشريط السفلي مع كلمة Active وشكل دائري باللون الأخضر.

الآن لنذهب إلى ملف html الخاص بـ UsersListComponent ، ونقوم بإضافة زر لكل قائمة من قوائم المستخدمين، كالتالي:

#### ملف users-list.component.html

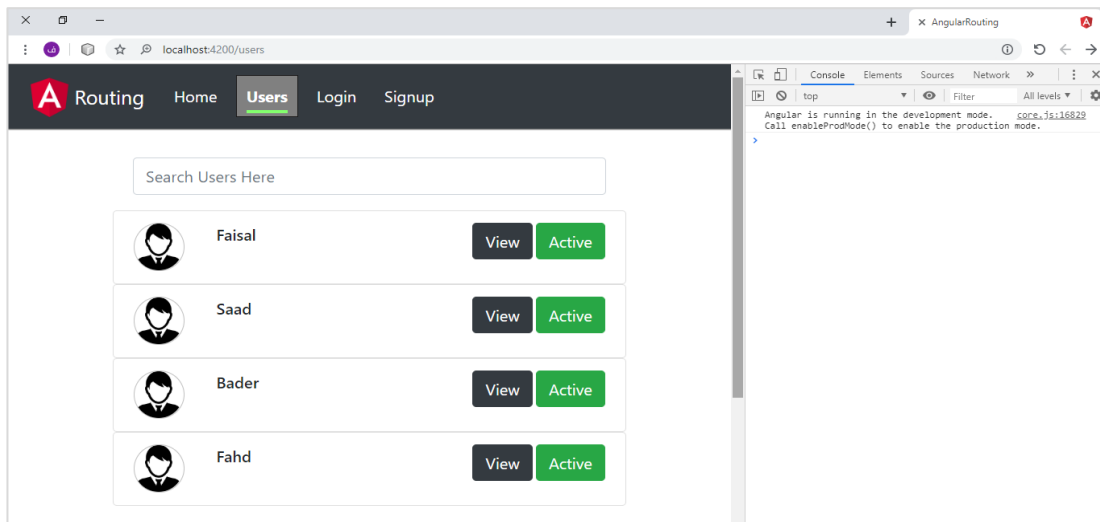
```

1. <div class="container" style="padding-top: 1rem">
2.   <ul class="list-group" *ngFor="let user of users$ | async">
3.     <li class="list-group-item" [class.is-active]="selectedUser === user.name">
4.       <span class="pull-left ">
5.         
11.     </span>
12.     <h6 style="padding-left: 2rem; display: inline">{{ user.name }}</h6>
13.     <span class="pull-right">
14.       <button
15.         class="btn btn-dark"
16.         style="display: inline"
17.         (click)="viewDetails(user.userId)">
18.         View
19.       </button>
20.       <button class="btn btn-success pull-right" style="margin-left: 3px">
21.         Active
22.       </button>
23.     </span>
24.   </li>
25. </ul>
26. </div>

```

الزر في الاسطر من 20 إلى 21.

## الآن لنحفظ التعديلات ونشاهد النتيجة:



نلاحظ ظهور الأزرار في كل عنصر من عناصر هذه القائمة.

الآن المطلوب هو كلما تم الضغط على زر Active نُريد الانتقال إلى صفحة عرض بيانات المستخدم والقفز إلى نقطة محددة أو جزء محدد من هذه الصفحة وهو العنصر الذي انشأناه سابقاً، ونستطيع القيام بها بعدة طرق منها استخدام Fragments Parameters، حيث سوف نُرسل بارامتر من هذا النوع بنفس اسم id الخاص بالعنصر وفي component الآخر يتم قراءته ونقلنا إلى العنصر الذي يحمل نفس id، ويجب التوضيح أن هذا المثال غير واقعي والبيانات هنا ثابتة بمعنى لا تتغير لجميع المستخدمين لأننا هنا لا نشرح البحث في قواعد البيانات وجلب بيانات هذا المستخدم وقراءتها ديناميكياً لأننا بذلك سوف نشنت ذهن القارئ المتعلم عن المقصد الأساسي وهو Fragments Parameters وكيفية إرسالها وقراءتها والاستفادة منها في البرامج الواقعية، أما من ناحية كيفية إرسال هذا النوع من البارامترات فهو مشابه للأنواع الأخرى من حيث إرسالها برمجياً أو عن طريق template وسوف اشير لكلا الطريقتين بإذن الله، لذلك سوف نذهب إلى نفس ملف html لهذا component وفي الزر نقوم بإضافة routerLink لنقلنا إلى صفحة بيانات المستخدمين وب نفس الوقت نضيف Directive المسعى fragment ونسند له القيمة active-user وهو نفس id للعنصر html، كالتالي:

### ملف users-list.component.html

```
1. <div class="container" style="padding-top: 1rem">
2.   <ul class="list-group" *ngFor="let user of users$ | async">
3.     <li class="list-group-item" [class.is-active]="selectedUser === user.name">
4.       <span class="pull-left ">
5.         
11.     </span>
12.     <h6 style="padding-left: 2rem; display: inline">{{ user.name }}</h6>
13.     <span class="pull-right">
14.       <button class="btn btn-dark" style="display: inline" (click)="viewDetails(user.userId)">
15.         View
```

```

16.     </button>
17.     <button
18.         class="btn btn-success pull-right"
19.         style="margin-left: 3px"
20.         [routerLink]="['../users/user-details', user.userId]"
21.         fragment="active-user"
22.     >
23.         Active
24.     </button>
25. </span>
26. </li>
27. </ul>
28. </div>

```

ما يهمنا هو ما هو موجود في السطر 20 حيث أضفنا routerLink لتوجيهه إلى صفحة تفاصيل بيانات المستخدم مع id الخاص به، وستر 21 حيث أضفنا للعنصر Directive ذو الاسم fragment مع القيمة الخاصة به.

أما في حال أردت إرسالها برمجياً فكل الذي عليك هو إنشاء دالة لهذا العنصر تستقبل بارامتر واحد وهو id الخاص بالمستخدم كما فعلنا مع الزر View، ومن ثم في هذه الدالة الجديدة تكتب محتواها في ملف ts الخاص بهذا component ويكون محتواها السطر البرمجي التالي:

```
this.router.navigate(['../users/user-details', id], {fragment: 'active-user'});
```

هذا من ناحية إرسالها أما من ناحية قراءتها فهي مشابهة أيضاً لقراءة البارامترات السابقة، لذلك لنذهب إلى ملف ts لـ UserDetailsComponent، ونضيف الكود التالي:

#### ملف user-details.component.html

```

1. import { Component, OnInit, OnDestroy, AfterViewChecked } from '@angular/core';
2. import { ActivatedRoute, Router } from '@angular/router';
3. import { UsersService } from 'src/app/users-data/users.service';
4. import { Subscription } from 'rxjs';
5. import { Users } from 'src/app/users-data/users';
6.
7. @Component({
8.   selector: 'app-user-details',
9.   templateUrl: './user-details.component.html',
10.  styleUrls: ['./user-details.component.css']
11.})
12. export class UserDetailsComponent implements OnInit, OnDestroy, AfterViewChecked {
13.   constructor(
14.     private activeRouter: ActivatedRoute,
15.     private usersService: UsersService,
16.     private router: Router) { }
17.
18.   user$: Users;
19.   private subscription: Subscription;
20.   private id: number;
21.
22.   ngOnInit() {

```

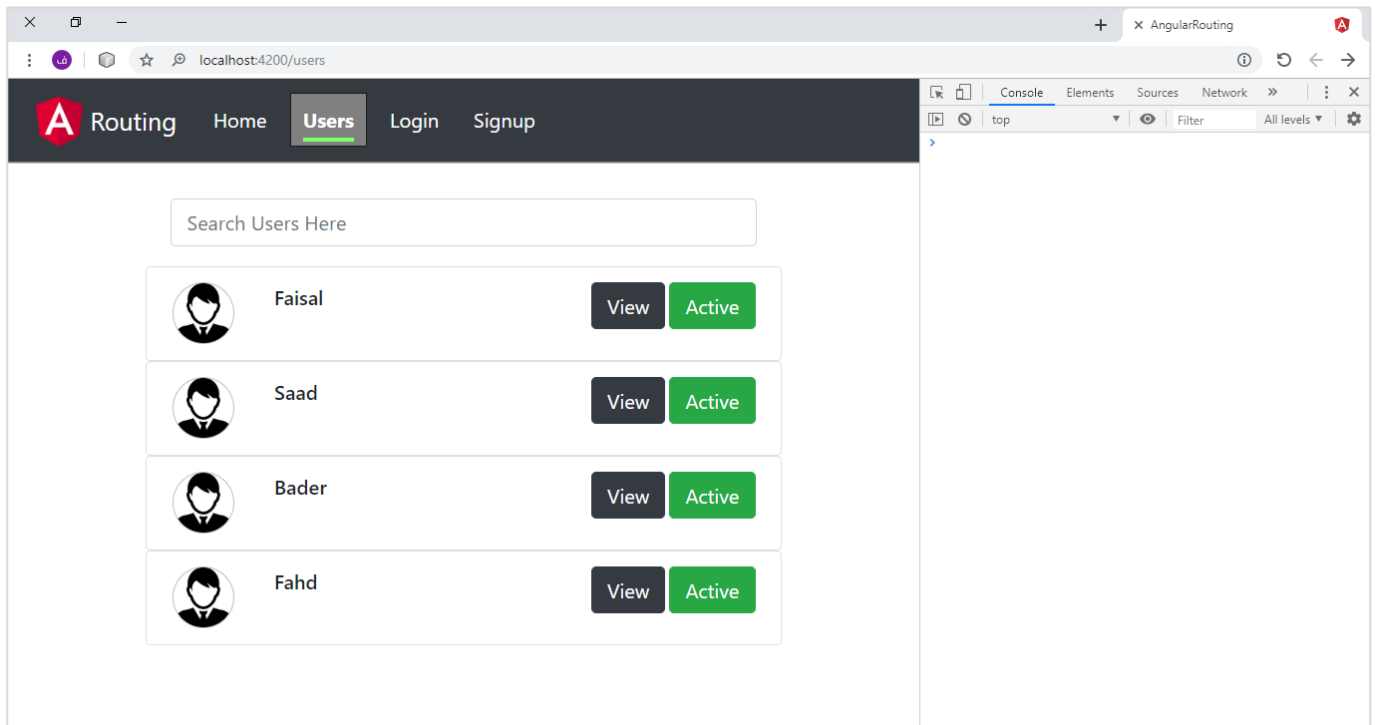
```

23. // this.id = +this.activeRouter.snapshot.params.id;
24. // this.subscription = this.usersSrevicе.getUserById(this.id).subscribe(user => {
25. //   this.user$ = user;
26. // });
27.
28. this.activeRouter.paramMap.subscribe(param => {
29.   this.id = +param.get('id');
30.   this.subscription = this.usersSrevicе.getUserById(this.id).subscribe(data => {
31.     this.user$ = data;
32.   });
33. });
34.
35. }
36.
37. ngAfterViewChecked() {
38.   const fragmentParam = this.activeRouter.snapshot.fragment;
39.   if (FragmentParam) {
40.     const elementId = document.getElementById(FragmentParam);
41.     elementId.scrollIntoView();
42.   }
43. }
44.
45. ngOnDestroy() {
46.   this.subscription.unsubscribe();
47. }
48.
49. viewNextUser() {
50.   this.subscription = this.usersSrevicе.getAllUsers().subscribe(data => {
51.     this.id === data.length ? this.id = 1 : this.id = this.id + 1;
52.   });
53.   this.router.navigate(['users/user-details', this.id], {
54.     queryParamsHandling: 'preserve'
55.   });
56. }
57.
58. }

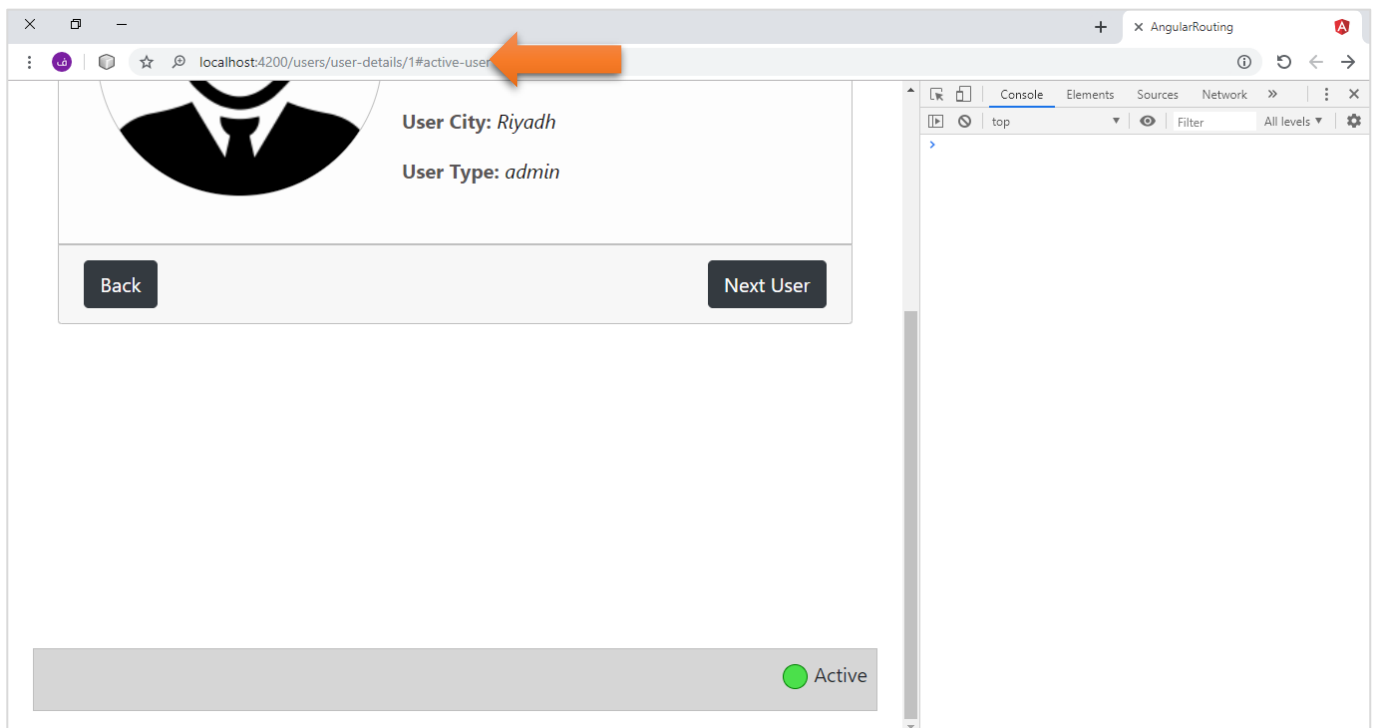
```

ما يهمننا هو ما هو موجود في الدالة `ngAfterViewChecked()` في الاسطر من 37 إلى 43 وهذه الدالة هي من دوال lifecycle hooks التي أشرنا لها في اكثر من موضع مثل `ngOnDestroy()` وغيرها ونعاملها كما عاملنا هذه الدوال من حيث الاستدعاء ونعمل لها implements، أما محتوى هذه الدالة فقد قرئنا الـ fragment القادم عن طريق دالة snapshot ونلاحظ انه لا يحمل دوال `get` و `getAll`.. الخ لأن هذا النوع لا يوجد به إلا قيمة واحدة فقط، ومن ثم قمنا بتخزين هذه القيمة في ثابت اسميته `fragmentParam` وفي السطر الذي يليه وضعت شرط لتأكد هل هنالك قيمة ام لا فإذا كانت هنالك قيمة اذهب إلى السطر 40 وجلب لنا العنصر الذي يحمل id الذي له نفس القيمة في هذا البارامتر ومن ثم في السطر 41 قمنا بعمل قفز للعنصر او الجزء المحدد عن طريق الدالة `scrollIntoView()`.

الآن لنحفظ التعديلات ونرى النتيجة:



لنختار أحد المستخدمين ونضغط على زر Active:



نلاحظ الرابط تم إضافة #active-user وهو بارامتر من النوع Fragment وبنفس الوقت تم القفز بنا إلى الجزء الذي نريد عرضه في الصفحة.

وبذلك نكون أنهيينا هذا الجزء وهو النوع الرابع والأخير من البارامترات في angular routing، وفي الجزء القادم سوف نتكلم عن NavigationExtras Interface.

## 6.2. NavigationExtras Interface

تخيل انك احتجت لأي سبب من الأسباب أن تُرسل جميع أنواع البارامترات السابقة عن طريق route واحد برمجياً، ففي هذه الحالة سوف يكون الكود كالتالي:

```
onClick() {
  this.router.navigate(
    ['any-path', { key1: 'value', key2: 'value2' }],
    { fragment: 'value', queryParamsHandling: 'preserve', queryParams: { key1: 'value1', key2: 'value2' } }
  );
}
```

وهذا الكود يعمل ولن تحدث مشاكل ولكن هو أصعب في القراءة وخصوصاً إذا كثرت البارامترات والkeys الخاصة بها، لذلك قدمت لنا angular طريقة أسهل وهو عن طريق interface المسمى NavigationExtras حيث نقوم بتعريف متغير من هذا النوع ونكتب فيه جميع البارامترات التي نريدها ثم نقوم بتمريرها لـ Router، كالتالي:

```
private navigationExtras: NavigationExtras = {
  fragment: 'value',
  queryParamsHandling: 'preserve',
  queryParams: { key1: 'value1', key2: 'value2' }
};
onClick() {
  this.router.navigate(
    ['users', { key1: 'value', key2: 'value2' }], this.navigationExtras
  );
}
```

نلاحظ في الجزء الأعلى عرفنا متغير اسميته navigationExtras من نفس نوع interface ذو الاسم NavigationExtras وقمنا بتعريف جميع البارامترات التي نريدها بداخله ماعدا Optional Parameters و Required Parameters ومن ثم قمنا بتمرير هذا المتغير إلى router.

هذا في حالة الارسال البرمجي اما إذا كان عن طريق template فكما هو معروف وتم شرحه سابقاً يتم ذلك عن طريق Directives الخاصة بكل نوع من هذا الأنواع.

## 7.2. الفرق بين أنواع البارامترات:

في هذا الجزء وهو الجزء الأخير الذي نختم به كلامنا عن البارامترات في angular routing، سوف ابين الفروقات المهمة بينها، كما هو موضح في الجدول التالي:

Required Parameters	Optional Parameters	Query Parameters	Fragment Parameters
تحتاج إلى تهيئة في routes	اختيارية ولا تحتاج إلى تهيئة في routes	اختيارية ولا تحتاج إلى تهيئة في routes	اختيارية ولا تحتاج إلى تهيئة في routes

تُرسل قيمة واحدة routed واحد ولا يمكن مشاركة هذه القيمة في route أكثر من	تُرسل أكثر من قيمة لـrouted واحد ولا يمكن مشاركة هذه القيم في أكثر من route	تُرسل أكثر من قيمة لـrouted واحد ويمكن مشاركة هذه القيم في أكثر من route	تُرسل قيمة واحدة routed واحد ولا يمكن مشاركة هذه القيمة في route أكثر من
لا تُرسل على شكل كائن object وانما قيمة نصية	تُرسل على شكل كائن لها keys value و	تُرسل على شكل كائن لها keys value و	تُرسل على شكل كائن ولكن تحتوي على key واحد وهو Fragment value و
تُرسل هي و Optional Parameters في مصفوفة واحدة	تُرسل هي و Required Parameters في مصفوفة واحدة	تُرسل على شكل كائن متداخل هي و Parameters Fragments أو Navigation Extras باستخدام	تُرسل على شكل كائن متداخل هي و Fragments Parameters أو Navigation Extras باستخدام
يتم قراءتها عن طريق الدالتين snapshot => params OR snapshot => paramMap (get-getAll-has-keys) او الدالة الثانية paramMap=>subscribe (get-getAll-has-keys)	يتم قراءتها عن طريق الدالتين snapshot => params OR snapshot => paramMap (get-getAll-has-keys) او الدالة الثانية paramMap=>subscribe (get-getAll-has-keys)	يتم قراءتها عن طريق الدالتين snapshot => queryParams (get-getAll-has-keys) او الدالة الثانية queryParams=>subscribe (get-getAll-has-keys)	يتم قراءتها عن طريق الدالتين snapshot => fragment او الدالة الثانية Fragment => subscribe (get-getAll-has-keys)

هذه بعض الفروقات التي حاولت أن أحصها، وبذلك نكون أنهينا كلامنا عن البارامترات في angular routing وقد غطينا أنواعها وأغلب مفاهيمها مع ذكر امثلة على كل مفهوم لكي تتضح الرؤية بشكل أوضح.

## 8.2. Relative and Absolute Navigation:

لقد تعلمنا سابقاً كيفية الربط بعد تهيئة routing سواء كان هذا الربط برمجياً او عن طريق template لكي نستطيع عمل توجيه Navigation لعرض محتويات صفحة معينة، ولكن هنالك بعض القواعد التي لم أشر إليها سابقاً. وقبل الكلام عنها لنستعرض بعض أنواع الربط البرمجي التي قمنا بها سابقاً، وسوف اعطي المثال الربط البرمجي في ملف users-list.component.ts حيث هذا الملف يحتوي على دالة باسم (viewDetails) وتقوم بعمل توجيه إلى صفحة UserDetailsComponent، كالتالي:

```
viewDetails(id: number) {
  this.router.navigate(['users/user-details', id], {
    queryParams: { searchText: this.inputSearch }
  });
}
```

نلاحظ أن المسار الذي وضعناه هو "users/user-details"، وهو عبارة عن قطعة نصية كاملة حيث كتبنا المسار كما قمنا بتهيئته في routing داخل علامتي التنصيص، بحيث يصبح بشكله النهائي http://localhost:4200/users/user-details



ومن ثم يتأكد من صحة هذه المسار مقارنة مع التهيئة التي قمنا بها سابقاً فإذا كانت صحيحة يُظهر هذه الصفحة، ولكن لو خطأً مثل أن نكون اخطأنا بحرف واحد في هذا المسار سوف يحدث خطأ في التوجيه، وهذا النوع يسمى Absolute Route أو Route الثابت، وهذه الطريقة جيدة وسوف تعمل ولا يشوبها أي مشكلة ولكن يعيها أنها من أسمها ثابتة بمعنى لو أخطأت في كتابة هذا المسار مثلاً بدلاً من كتابة users كتبنا user سيصبح المسار 'user/user-details' وفي هذه الحالة يقوم Angular بمقارنة هذا المسار بما هو موجود في التهيئة التي قمنا بها سابقاً في ملف app-routing.module.ts وعندما لم يجده سوف يقوم بتوجيهنا إلى صفحة NotFoundPageComponent، لذلك قدمت لنا Angular Router طريقة أكثر مرونة تحت أسم Relative Routes، بحيث يمكن إعادة كتابة التوجيه السابق في الدالة viewDetails()، كالتالي:

```
viewDetails(id: number) {
  this.router.navigate(['./user-details', id],
    {
      relativeTo: this.activatedRoute, queryParams: { searchText: this.inputSearch }
    });
}
```

نلاحظ أننا استبدلنا الاسم users بالرمز ./ وأضفنا الخاصية relativeTo واسندنا لها القيمة this.activatedRoute وهو المتغير الذي علمنا له Inject في class الخاص بالcomponent، بمعنى أنه بدأ من هذا Route النشط حالياً (في حالتنا هنا هو users) قم بالانتقال إلى Route الأب (وفي حالتنا هذه هو user-details) وأضف له البارامتر id.

وفي الجدول التالي مجموعة من حالات Relative التي يمكن استخدامها:

الحالة الأولى	
الأمر البرمجي	<code>this.router.navigate(["../../active4"], {relativeTo: this.activeRoute});</code>
الشرح	<p>يُقصد به بدءاً من Route النشط حالياً قم بالانتقال خطوتين إلى الأعلى ثم الانتقال إلى route ذو الـ Path الذي يحمل الاسم active4، كما في الشكل التوضيحي التالي:</p>

الحالة الثانية	
الأمر البرمجي	<code>this.router.navigate(["../active3"], {relativeTo: this.activeRoute });</code>
الشرح	<p>يُقصد به بدءاً من Route النشط حالياً قم بالانتقال خطوة إلى الأعلى إلى Route الأب ثم الانتقال إلى route ذو الPath الذي يحمل الاسم active3 أو ممكن أن نطلق عليه Route الأخ لRoute النشط حالياً في المتصفح، كما في الشكل التوضيحي التالي:</p> <pre> graph TD     A["Route الأب لRoute الأبن النشط حالياً http://localhost:4200/active1 2"] -- "أولاً ينتقل خطوة إلى الأعلى لRoute الأب (../)" --&gt; B["Route النشط حالياً ولنفرض أنه: http://localhost:4200/active1/active2 1"]     B -- "أخيراً ينتقل لRoute الأبن أو الأخ لRoute النشط" --&gt; C["Route الأخ لRoute النشط حالياً ولنفرض أنه: http://localhost:4200/active1/active3 3"] </pre>
الحالة الثالثة	
الأمر البرمجي	<code>this.router.navigate(["./active2"], { relativeTo: this.activeRoute })</code> OR <code>this.router.navigate(["active2"], { relativeTo: this.activeRoute })</code>
الشرح	<p>يُقصد به بدءاً من Route النشط حالياً قم بالانتقال خطوة للأسفل إلى Route الأبن، كما في الشكل التالي</p> <pre> graph TD     A["Route النشط حالياً ولنفرض أنه: http://localhost:4200/active1 1"] -- "ينتقل إلى Route الأبن لRoute النشط حالياً" --&gt; B["Route الأبن لRoute النشط حالياً: http://localhost:4200/active1/active2 2"] </pre>

بعدما استعرضنا الحالات السابقة، بقي أن نبين طرق إضافة الخاصية relativeTo في حال كان هنالك باراميترات سواء كانت Option Parameters أو Query Parameters .... الخ.

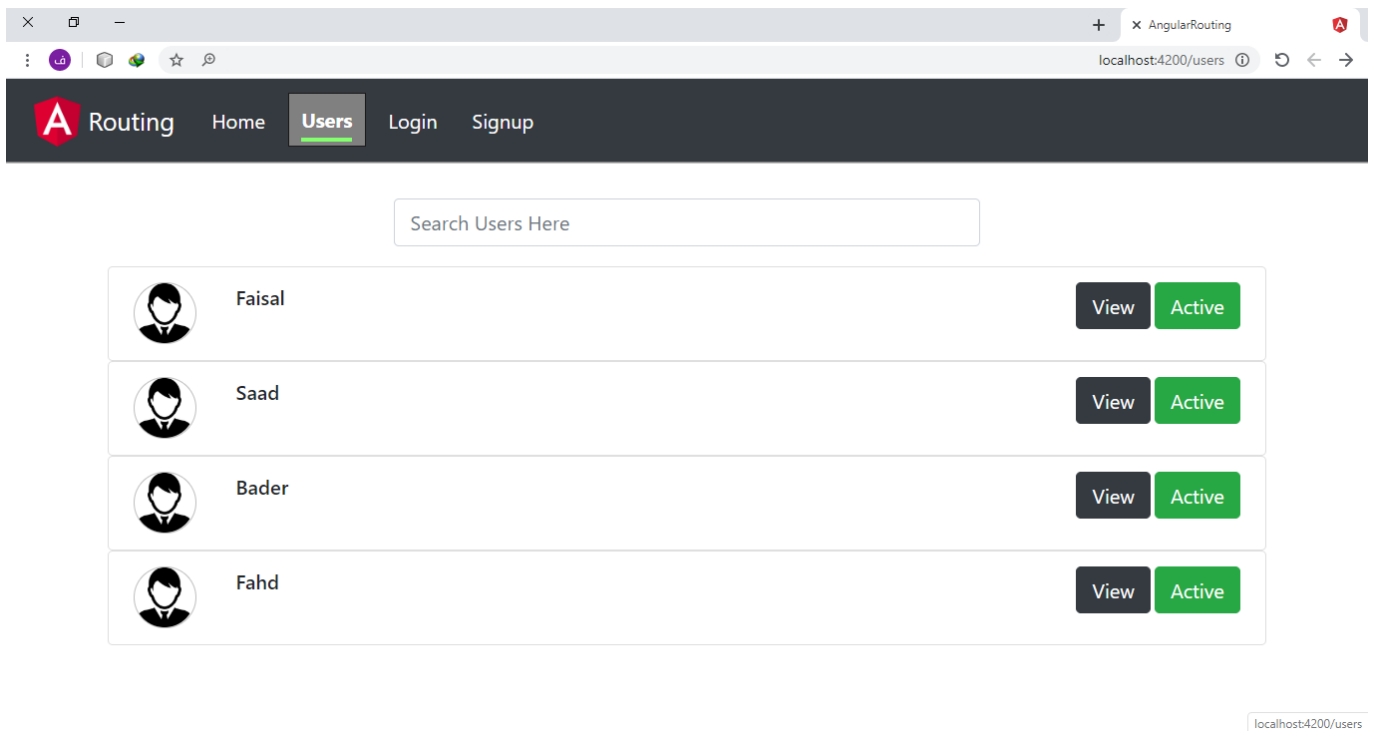
<code>this.router.navigate(["Router Path", {Option Parameters}], {relativeTo: the Active Route});</code>
<code>this.router.navigate(["Router Path"], {relativeTo: the Active Route, queryParams: {p1: 'value1', p2: 'value2'}, fragment: 'value'});</code>

## 9.2. خاصية skipLocationChange:

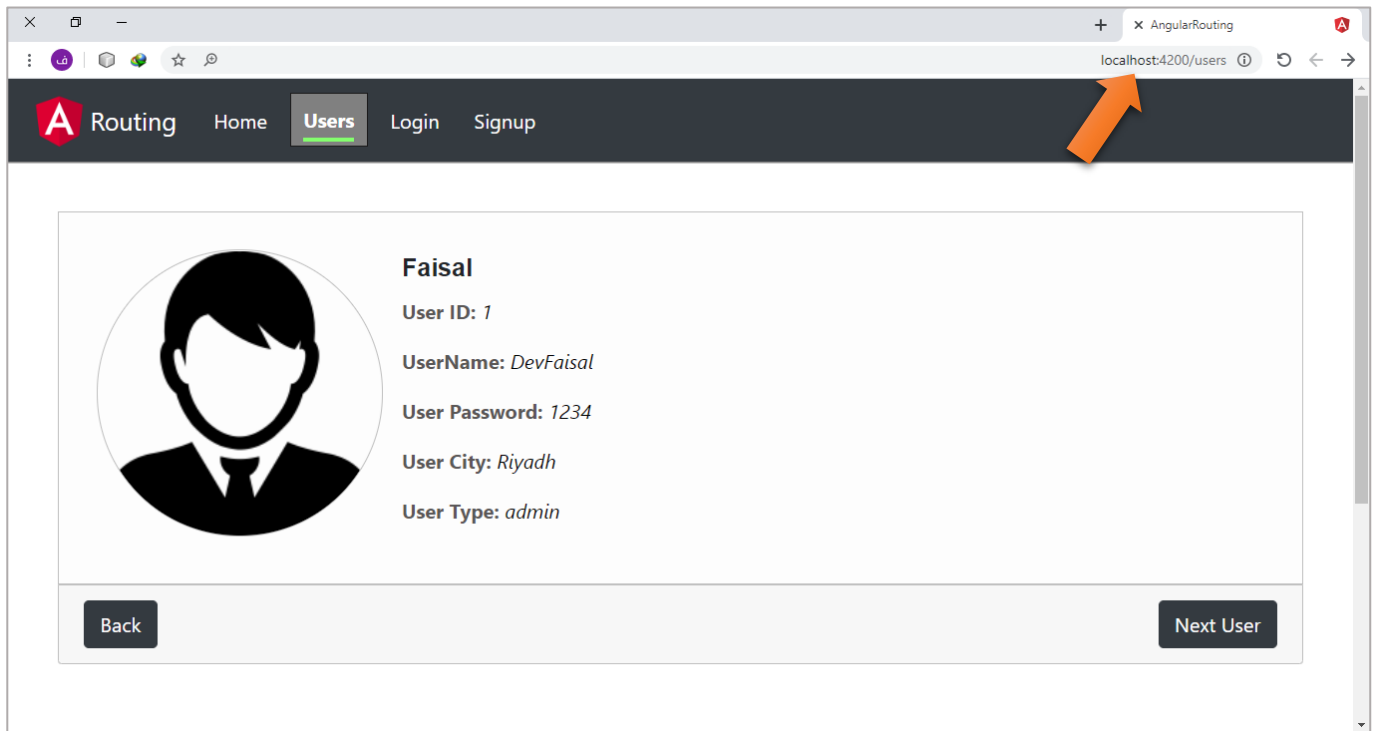
هناك خاصية أخرى تُضاف إلى Navigate Router وهي skipLocationChange وتأخذ قيمة منطقية true أو false، بحيث أن القيمة الافتراضية لهذه الخاصية هي false، أما إذا جعلنا قيمتها true فسوف تجعل الرابط في المتصفح ثابت أثناء التنقل ولا تقوم بتغييره، ولتوضيح وفهم هذه الخاصية سوف نقوم بتطبيقها على المثال الخاص بمشروعنا، لذلك سوف نذهب إلى الدالة viewDetails() ونقوم بإضافة هذه الخاصية لها ونجعل قيمتها true، كالتالي:

```
viewDetails(id: number) {  
  this.router.navigate(['./user-details', id],  
    {  
      relativeTo: this.activatedRout,  
      skipLocationChange: true  
      queryParams: { searchText: this.inputSearch }  
    });  
}
```

الآن لنحفظ التعديلات ونذهب إلى المتصفح لمشاهدة النتيجة، مع التأكد أن server يعمل بشكل صحيح.



نذهب إلى القائمة Users ومن ثم نضغط على أحد المستخدمين، ولنركز على الرابط في المتصفح <http://localhost:4200/users>



نلاحظ أن الرابط لم يتغير مع أن الصفحة تغيرت، ونستفيد من هذه الخاصية مثلاً إذا أردنا إظهار صفحة بيانات المستخدم بدون إظهار رقم id الخاص به في الرابط أو باقي البارامترات، كما نستطيع إخفاء Query فقط بكتابة هذا الأمر `preserveQueryParams` اما لإخفاء Fragment فنكتب هذا الأمر `preserveFragment`.

## 10.2. Router Events

عندما يتم الانتقال من Route إلى آخر فإن Angular Router يقوم بتنفيذ مجموعة من الأحداث Events، ومن هذه الأحداث `NavigationStart`, `NavigationEnd`, `NavigationCancel`, `NavigationError`, `ResolveStart`, ...etc، فعلى سبيل المثال يُنفذ الحدث `NavigationStart` عند بداية الانتقال و `NavigationEnd` عند الانتهاء من الانتقال و `NavigationError` في حال حدوث خطأ غير متوقع عند الانتقال ولم يتم الانتقال بالشكل المناسب.

ولاستعراض هذه الأحداث جميعها نذهب إلى نفس الملف السابق وهو ملف `user-list.component.ts` وفي ملف `class` لهذا `component` (ملف `ts`)، وما يهمنا في هذه الملف هو `Router class` الذي عملنا له `inject` سابقاً في `constructor` الخاص بهذا الملف، حيث أن هذا `Router` يحتوي على خاصية اسمها `event` وهي من النوع `observable` لذلك سوف نعمل لها `subscribe` ونمرر لها بارامتر من النوع `RouterEvent` ونعمل `console.log` لهذا البارامتر ونرى النتيجة، مع العلم أننا سوف نضيف هذه الكود في الدالة `ngOnChanges()` وهي من دوال `lifecycle Hook` التي تكلمنا عنها سابقاً.

ملف `user-list.component.ts`

```
1. import { Component, OnInit, Input, OnChanges } from '@angular/core';
2. import { Observable, of } from 'rxjs';
3. import { Users } from 'src/app/users-data/users';
4. import { Router, ActivatedRoute, RouterEvent } from '@angular/router';

5. @Component({
6.   selector: 'app-users-list',
```

```

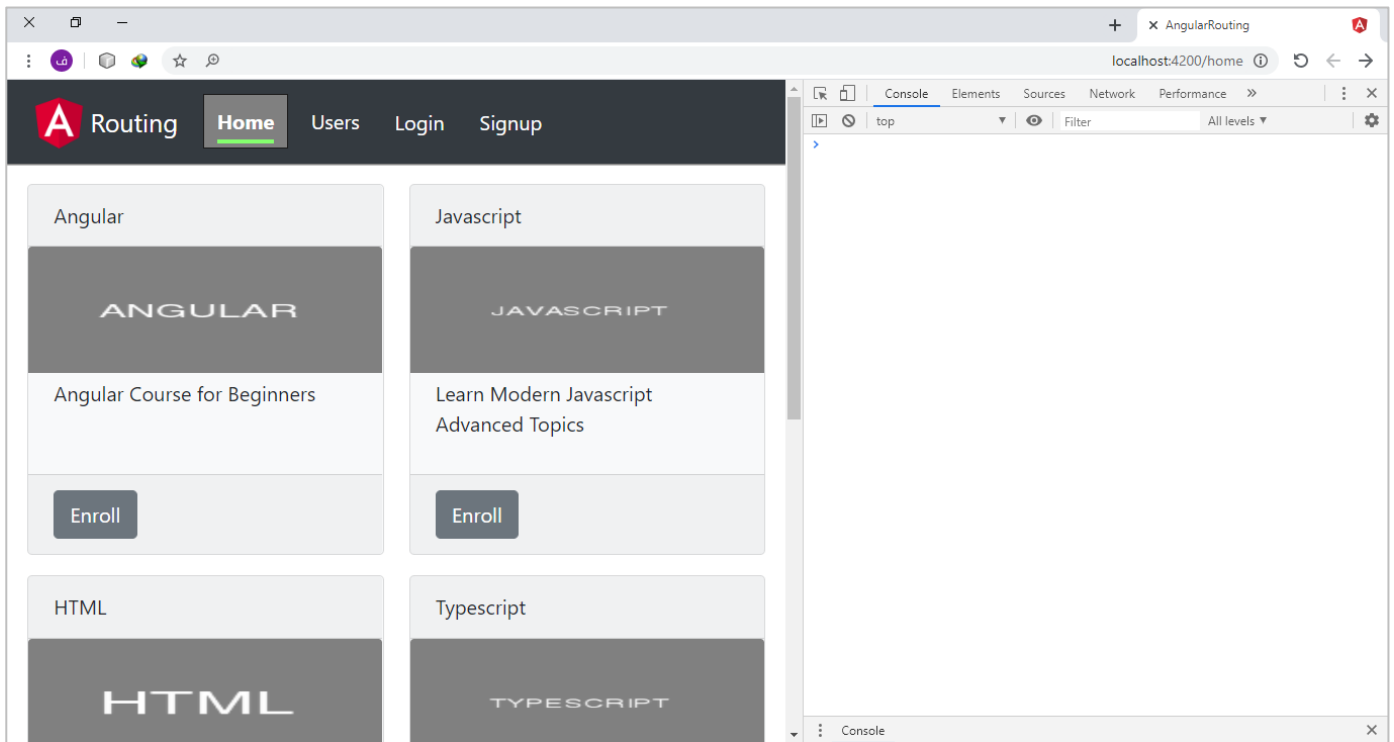
7.   templateUrl: './users-list.component.html',
8.   styleUrls: ['./users-list.component.css']
9. })
10. export class UsersListComponent implements OnInit, OnChanges {
11.   @Input() users$: Observable<Users[]>;
12.   @Input() fullUsersList$: Observable<Users[]>;
13.   @Input() inputSearch: string;
14.   selectedUser: string;

15.   constructor(private router: Router, private activatedRoute: ActivatedRoute) { }
16.   ngOnInit() {
17.     this.selectedUser = this.activatedRoute.snapshot.paramMap.get('name');
18.   }
19.   ngOnChanges() {
20.     if (this.inputSearch) {
21.       this.users$.subscribe(data => {
22.         const result = data.filter(user =>
23.           user.name.toLocaleLowerCase().indexOf(
24.             this.inputSearch.toLocaleLowerCase()) !== -1);
25.         this.users$ = of(result);
26.       });
27.     } else {
28.       this.users$ = this.fullUsersList$;
29.     }
30.     this.router.events.subscribe((event: RouterEvent) => {
31.       console.log(event);
32.     });
33.   }

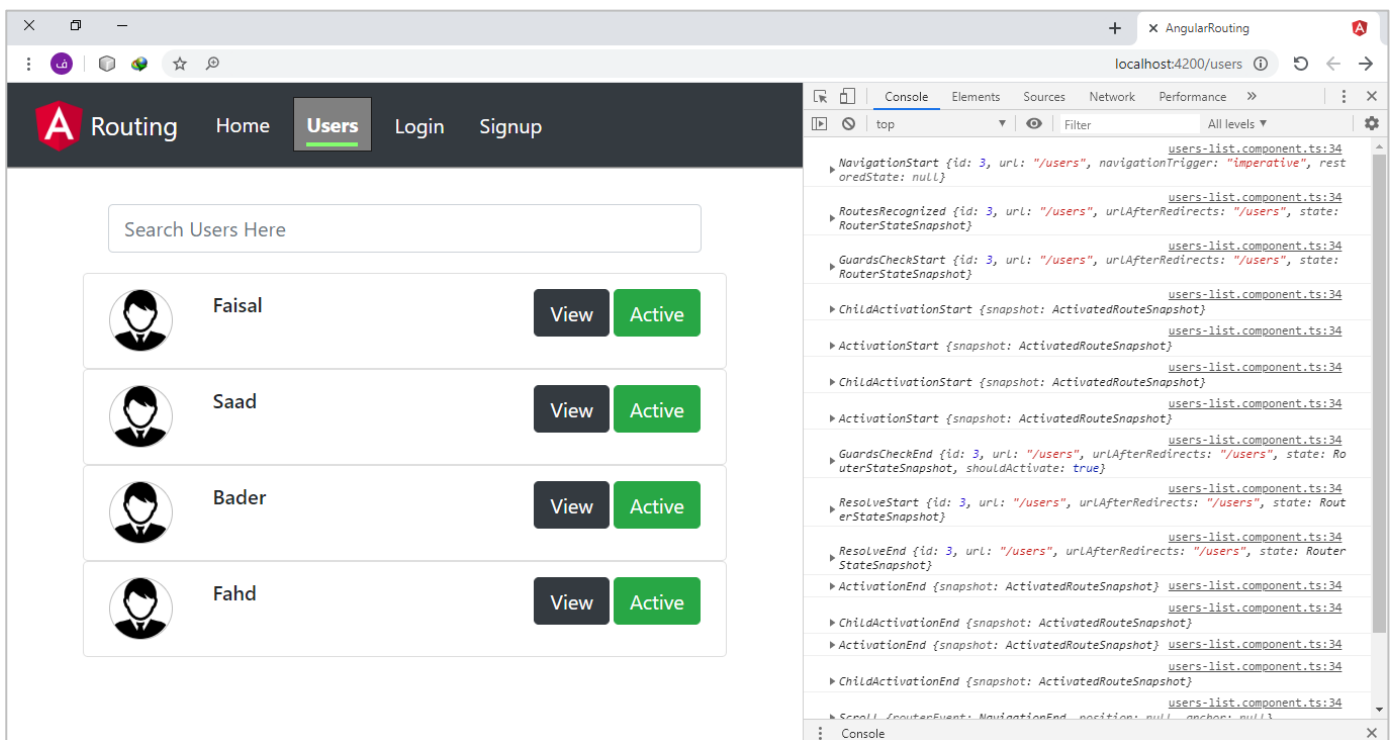
34.   viewDetails(id: number) {
35.     this.router.navigate(['./user-details', id],
36.       {
37.         relativeTo: this.activatedRoute,
38.         skipLocationChange: true,
39.         queryParams: { searchText: this.inputSearch }
40.       });
41.   }
42. }

```

الآن لنحفظ التعديلات ونذهب إلى المتصفح ونفتح صفحة console كما تعلمنا سابقاً



نلاحظ أن صفحة console فارغة، الآن لنقوم بالضغط على التبويب Users ونرى النتيجة:



نلاحظ أنه ظهرت لنا جميع الاحداث اثناء التوجيه او الانتقال من صفحة Home إلى صفحة Users بدءاً من NavigationStart وانتهاءً بـ NavigationEnd مروراً بالأحداث التي بينهما.

وفي حالة الرغبة في التعامل مع نوع محدد من هذه الأحداث، فمثلاً إذا أردنا التعامل مع الحدث NavigationEnd بالتحديد، نقوم بكتابة الكود التالي:

ملف `users-list.component.ts`

```

1. import { Component, OnInit, Input, OnChanges } from '@angular/core';
2. import { Observable, of } from 'rxjs';
3. import { Users } from 'src/app/users-data/users';
4. import { Router, ActivatedRoute, RouterEvent, NavigationEnd } from '@angular/router';

5. @Component({
6.   selector: 'app-users-list',
7.   templateUrl: './users-list.component.html',
8.   styleUrls: ['./users-list.component.css']
9. })
10. export class UsersListComponent implements OnInit, OnChanges {
11.   @Input() users$: Observable<Users[]>;
12.   @Input() fullUsersList$: Observable<Users[]>;
13.   @Input() inputSearch: string;
14.   selectedUser: string;

15.   constructor(private router: Router, private activatedRoute: ActivatedRoute) { }

16.   ngOnInit() {
17.     this.selectedUser = this.activatedRoute.snapshot.paramMap.get('name');
18.   }

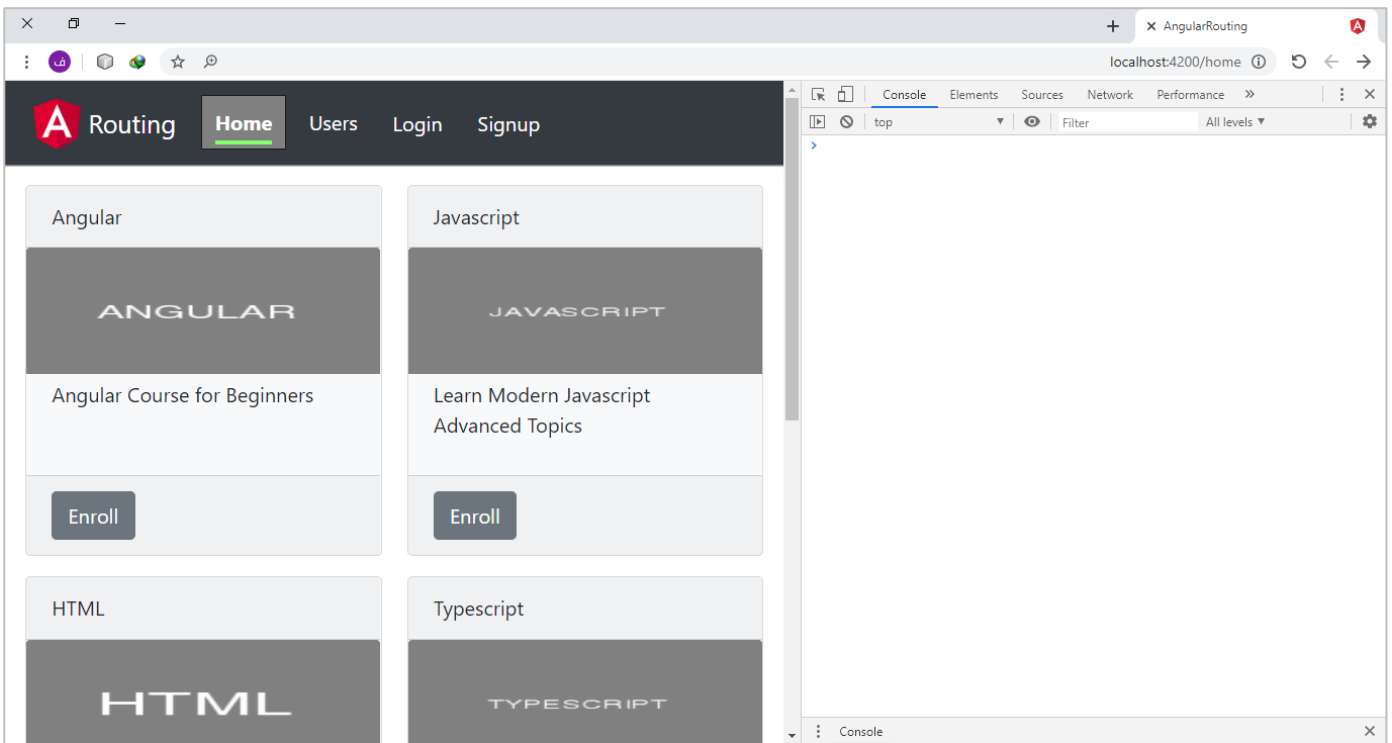
19.   ngOnChanges() {
20.     if (this.inputSearch) {
21.       this.users$.subscribe(data => {
22.         const result = data.filter(user =>
23.           user.name.toLocaleLowerCase().
24.             indexOf(this.inputSearch.toLocaleLowerCase()) !== -1);
25.         this.users$ = of(result);
26.       });
27.     } else {
28.       this.users$ = this.fullUsersList$;
29.     }
30.     this.router.events.subscribe((event: RouterEvent) => {
31.       if (event instanceof NavigationEnd) {
32.         // نقوم بكتابة ما نريده من أكواد هنا
33.         console.log(event);
34.       }
35.     });
36.   }

37.   viewDetails(id: number) {
38.     this.router.navigate(['./user-details', id],
39.       {
40.         relativeTo: this.activatedRoute,
41.         skipLocationChange: true,
42.         queryParams: { searchText: this.inputSearch }
43.       });
44.   }
45. }

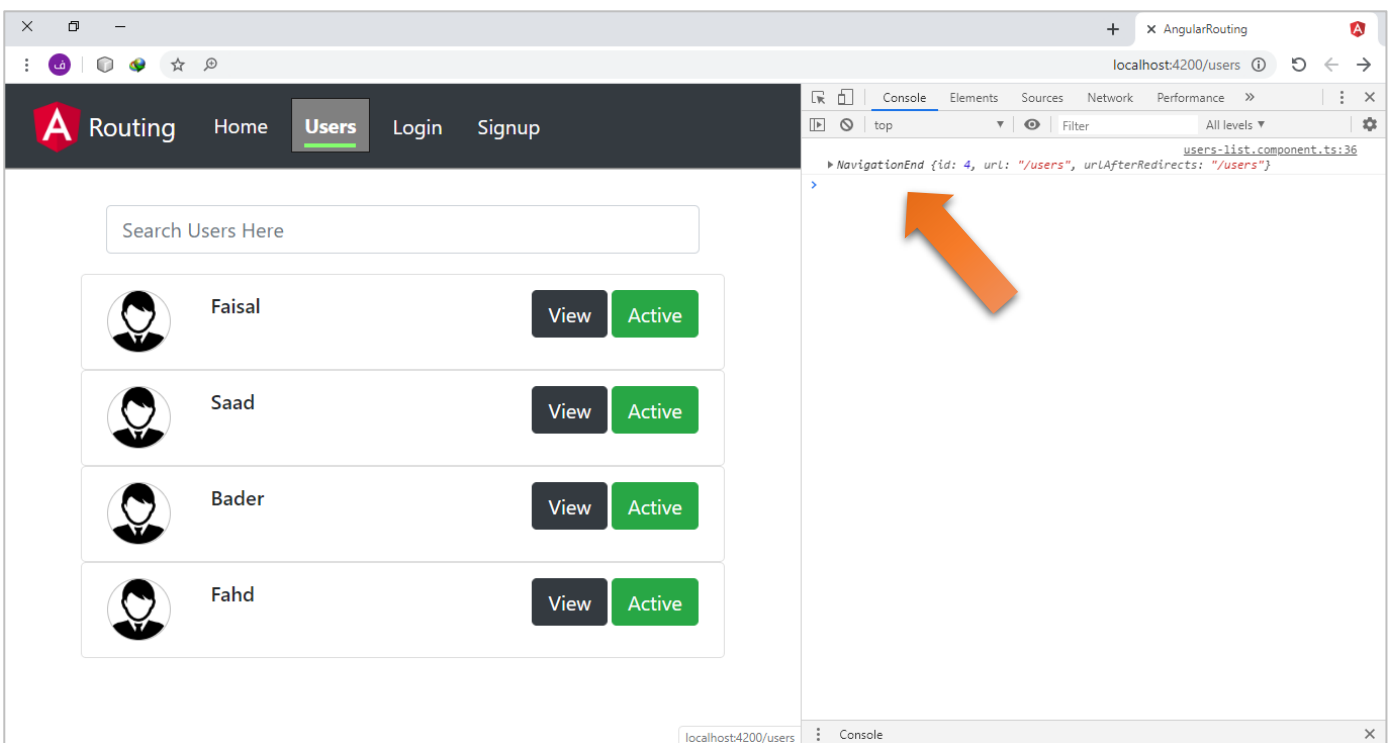
```



الآن لنقوم بحفظ التعديلات والذهاب إلى المتصفح، لكي نرى النتيجة:



الآن لنقوم باختيار التبويب Users لذهاب إلى هذه الصفحة ولنرى النتيجة في console:



نلاحظ أنه ظهر لنا الحدث `NavigationEnd` فقط ونستطيع أن نعمل ما نريده عند تنفيذ هذا الحدث. كما نستطيع أن نظهر جميع هذه الأحداث من خلال كتابة `{ enableTracing: true }` كبرامير ثاني في دالة `forRoot`.



وبالنظر إلى Documentation الخاص بموقع Angular على شبكة الانترنت فسوف نجد العديد من الإحداث سوف أذكر أهمها ووظيفتها في الجدول التالي:

م	الحدث Events	الوظيفة
1	NavigationStart	يبدأ تنفيذ هذا الحدث في بداية الانتقال من Route إلى آخر
2	NavigationCancel	يبدأ تنفيذ هذا الحدث في حالة قام المستخدم بالإلغاء الانتقال من Route إلى آخر
3	NavigationError	يبدأ تنفيذ هذا الحدث في حال حدوث خطأ غير متوقع عند الانتقال من Route إلى آخر
4	NavigationEnd	يبدأ تنفيذ هذا الحدث في حال انتهاء الانتقال من Route إلى آخر
5	GuardsCheckStart	يبدأ تنفيذ هذا الحدث عند بداية تفعيل guards عند الانتقال من Route إلى آخر
6	GuardsCheckEnd	يبدأ تنفيذ هذا الحدث عند انتهاء تفعيل guards عند الانتقال من Route إلى آخر
7	ResolveStart	يبدأ تنفيذ هذا الحدث عند بداية استقبال البيانات عند الانتقال من Route إلى آخر
8	ResolveEnd	يبدأ تنفيذ هذا الحدث عند الانتهاء من استقبال البيانات عند الانتقال من Route إلى آخر
9	RouteConfigLoadStart	يبدأ تنفيذ هذا الحدث عند بداية تفعيل Lazy Loading
10	RouteConfigLoadEnd	يبدأ تنفيذ هذا الحدث عند الانتهاء من تفعيل Lazy Loading

وبذلك نكون أنهينا هذا الجزء وسوف ننتقل في الجزء التالي إلى قسم مهم وهو Routes Guards.

# الفصل الثالث

## Route Guards

### 1.3. المقدمة:

لعل من الميزات المهمة جداً في Angular Router هي Route Guards ، حيث عن طريقها نحدد لمستخدمي الموقع (تطبيق الويب) الصلاحيات في دخول بعض الأقسام من عدمه، فمثلاً نريد من مستخدمي التطبيق أن لا يقوموا بمشاهدة الملف الشخصي إلا بعد تسجيل الدخول أو مثلاً ولو قاموا بتسجيل الدخول لكن لا يستطيعون الدخول إلى قسم التعديل على المنشورات أو على قسم بيانات المستخدمين لأن هذه الأقسام خاصة بمدير التطبيق admin....الخ، لذلك نستطيع أن نقول بصيغة أخرى: إننا بإضافة هذه الميزات قمنا بحماية التطبيق الخاص بنا.

وقدم لنا إطار عمل Angular مجموعة من الأنواع للGuards لكل نوع من هذه الأنواع مهمته التي يقوم بها، وهي كالتالي:

١) CanActivate Guard

٢) CanDeactivate Guard

٣) CanActivateChild Guard

٤) CanLoad Guard

٥) Resolve Guard

وجميع هذه الأنواع السابقة هي عبارة عن interface نقوم بعمل implements لها عن طريق كلاس class معين.

وسوف نتكلم عن جميع هذه الأنواع بالتفصيل بإذن الله، ولكن قبل التطرق إلى هذه الأنواع سوف نقوم بعمل مجموعة من التعديلات على المشروع الخاص بنا، كالتالي:

أولاً/ نذهب إلى ملف users.ts، ونجري التعديلات التالية:

ملف users.ts

```
1. export class Users {
2.   userId: number;
3.   name: string;
4.   userCity: string;
5.   userName: string;
6.   password: string;
7.   userType: string;
8. }
```

ثانياً/ نذهب إلى ملف users.service.ts، ونجري التعديلات التالية:

ملف users.service.ts

```
1. import { Injectable } from '@angular/core';
2. import { Users } from './users';
3. import { of, Observable, BehaviorSubject } from 'rxjs';
4. import { map } from 'rxjs/operators';
5. import { Router } from '@angular/router';
6. import { StorageMap } from '@ngx-pwa/local-storage';
7. @Injectable({
```

```

8.     providedIn: 'root'
9. })

10. export class UsersService {

11.     public isAdmin$ = new BehaviorSubject(false);
12.     public isLogin$ = new BehaviorSubject(false);

13.     constructor(private router: Router, private storageMap: StorageMap) { }

14.     USERS: Users[] = [
15.         {
16.             userId: 1,
17.             userType: 'admin',
18.             name: 'Faisal',
19.             userCity: 'Riyadh',
20.             userName: 'DevFaisal',
21.             password: '1234',
22.         },
23.         {
24.             userId: 2,
25.             userType: 'no-admin',
26.             name: 'Saad',
27.             userCity: 'Riyadh',
28.             userName: 'DevSaad',
29.             password: '1111',
30.         },
31.         {
32.             userId: 3,
33.             userType: 'no-admin',
34.             name: 'Bader',
35.             userCity: 'Dammam',
36.             userName: 'DevBader',
37.             password: '2222',
38.         },
39.         {
40.             userId: 4,
41.             userType: 'no-admin',
42.             name: 'Fahd',
43.             userCity: 'Jeddah',
44.             userName: 'DevFahd',
45.             password: '3333',
46.         },
47.     ];

48.     usersList$ = of(this.USERS);

49.     getAllUsers(): Observable<Users[]> {
50.         return this.usersList$;

```

```

51.     }

52.     getUserById(id: number): Observable<Users> {
53.         return this.getAllUsers().pipe(
54.             map(users => users.find(user => {
55.                 return user.userId === id;
56.             })))
57.     );
58. }

59.     addNewUser(user: Users) {
60.         user.userId = this.USERS.length + 1;
61.         user.userType = 'no-admin';
62.         this.USERS.push(user);
63.         this.isAdmin$.next(false);
64.         this.isLogin$.next(true);
65.         this.storageMap.set('id', (user.userId).toString()).subscribe(() => { });
66.         localStorage.setItem('type', user.userType);
67.         this.router.navigateByUrl('/');
68.     }

69.     login(userName: string) {
70.         this.getAllUsers()
71.             .pipe(map(users => users.find(user => user.userName === userName)))
72.             .subscribe((val) => {
73.                 this.userType = val.userType;
74.                 this.isLogin$.next(true);
75.                 val.userType === 'admin' ?
76.                     this.isAdmin$.next(true) : this.isAdmin$.next(false);
77.                 this.storageMap.set('id', (val.userId).toString()).subscribe(() => { });
78.                 localStorage.setItem('type', val.userType);
79.             });
80.     }

81.     logoutUser(): void {
82.         this.isAdmin$.next(false);
83.         this.isLogin$.next(false);
84.         this.storageMap.delete('id').subscribe(() => { });
85.         localStorage.clear();
86.         this.router.navigateByUrl('/');
87.     }

88. }

```

أهم التعديلات:

- أضفنا مكتبة خارجية من مهامها التخزين في Local Storage مع عمل subscribe لهذه القيم المخزنة ومراقبتها وترجع القيمة الجديدة في حال وجود أي تعديل، حيث قمنا باستدعاء المكتبة في السطر (6) وعملنا له inject في

constructor في متغير اسميه storageMap في السطر (14)، مع العلم أنه يمكن إضافة هذه المكتبة عن طريق كتابة هذا الأمر في terminal :

```
ng add @ngx-pwa/local-storage
```

مع العلم أن هذه المكتبة تعمل بدون مشاكل في الإصدار الثامن وما فوق لإطار عمل angular، ولمعرفة كيفية إضافة المكتبات الخارجية الرجاء الرجوع إلى الكتاب الأول من هذه السلسلة.

- وفي الأسطر (11) و (12) قمنا بتعريف متغيرين الأول اسميه isAdmin\$ والثاني isLogin\$ وهما Observable وجعلناهما من النوع BehaviorSubject (هذا النوع يسمح لنا بعمل مشاركة لهذه المتغيرات بكامل المشروع، بحيث نقوم بمراقبة هذه المتغيرات وفي حال تغير قيمها نقوم بتنفيذ مهمة معينة)، وكما هو واضح من أسماء هذه المتغيرات هي لتأكد هل المستخدم قام بتسجيل الدخول أو لا، وايضاً هل هو مدير للنظام أو لا، مع العلم أن هذه المتغيرات تُرجع قيمة منطقية true أو false، وقد قمنا بتمرير قيمة false لكلا المتغيرين بمعنى أن المستخدم لم يقوم بتسجيل الدخول وليس مدير لنظام.

- وفي الأسطر من (59) إلى (68) قمنا بإنشاء دالة لإضافة مستخدم جديد (هنا نقوم بمحاكاة إضافة مستخدم جديد لأنه ليس لدينا قاعدة بيانات) حيث أن هذه الدالة تستقبل كائن object من النوع Users وهو عبارة عن الكلاس class الموجود في ملف users.ts الذي قمنا بالتعديل عليه في الخطوة أولاً، ومن اسمها تقوم هذه الدالة بإضافة مستخدم جديد عن طريق إضافة الكائن القادم لهذه الدالة عن طريق الباراميتر user إلى المصفوفة USERS. ومن ثم نمرر القيمة false للمتغير isAdmin\$ لأنه بكل تأكيد المستخدم الذي قام بالتسجيل ليس مسئول الموقع (تطبيق الويب)، أما المتغير الآخر فقد مررنا له القيمة false بحيث بقيامه بتسجيل سوف يتم تسجيل دخوله بشكل تلقائي وتخزين id الخاص به في Local Storage عن طريق المكتبة الخارجية، وتخزين نوعه (هل هو admin أو no-admin) في Local Storage عن طريق الجافا سكريبت بالطريقة الاعتيادية لأننا هنا لا نحتاج أن نقوم بمراقبتها بشكل تلقائي ومعرفة إذا تغيرت قيمها أو لا. وأخيراً نقوم بتوجيه المستخدم إلى الصفحة الرئيسية للنظام.

- وفي الأسطر من (69) إلى (80) دالة أخرى تقوم بتسجيل الدخول حيث تستقبل باراميتر واحد من النوع string وتستدعي الدالة getAllUsers() وتبحث فيها بالاستناد إلى الباراميتر القادم إليها وهو يحمل قيمة اسم المستخدم، ومن ثم نعمل له subscribe بحيث نُخزن قيمة المستخدم في متغير اسميه val ومن ثم غيرنا قيمة المتغير isLogin\$ إلى true بمعنى أن المستخدم قام بتسجيل الدخول ومن ثم نقوم بالتأكد من نوع هذا المستخدم إذا كان admin ونقوم بتمرير القيمة true للمتغير isAdmin\$ أو false إذا كان غير ذلك مع إجراء نفس الأوامر التي عملناها في الدالة addUser().

- وفي الأسطر من (81) إلى (87) ايضاً هنالك دالة أخرى ومهمتها عمل تسجيل خروج حيث تمرر القيم false للمتغيرين isAdmin\$ و isLogin\$ وايضاً نحذف القيم من Local Storage ونُعيد توجيه المستخدم إلى الصفحة الرئيسية.

ملاحظة مهمة: جميع الأكواد السابقة في البرامج الحقيقية لا معنى لها لأنه في التطبيقات الواقعية نتعامل مع قاعدة بيانات api والكثير من التفاصيل التي ليس المقام لشرحها هنا، ولو قمنا بشرحها فسوف نحيد عن المقصد الأساسي لهذا الكتاب وهو Angular Routing، لذلك قمت بمحاولة محاكاة البرامج الواقعية لكي نركز على المقصد الأساسي من هذا الكتاب.

ثالثاً / نذهب إلى ملف `signup.component.html` ونقوم بإجراء التعديلات التالية:

ملف `signup.component.ts`

```
<div class="card text-white bg-secondary mt">
  <form #form="ngForm" (ngSubmit)="onSubmit(form)">
    <div class="card-header text-center">Signup</div>
    <div class="card-body">
      <div class="form-group">
        <input
          class="form-control"
          type="text"
          name="name"
          ngModel
          placeholder="Enter Name Here"
        />
      </div>
      <div class="form-group">
        <input
          class="form-control"
          type="text"
          name="userName"
          ngModel
          placeholder="Enter User Name Here"
        />
      </div>
      <div class="form-group">
        <input
          class="form-control"
          type="password"
          name="password"
          ngModel
          autocomplete
          placeholder="Enter Password Here"
        />
      </div>
      <div class="form-group">
        <input
          class="form-control"
          type="text"
          name="userCity"
          ngModel
          placeholder="Enter City Here"
        />
      </div>
    </div>
  </div>
  <div class="card-footer">
```

```

    <button class="btn btn-light form-control" type="submit">
      Submit
    </button>
  </div>
</form>
</div>

```

وفي ملف ts لهذه component نجري التعديلات التالية:

#### ملف singup.component.ts

```

1. import { Component, OnInit } from '@angular/core';
2. import { NgForm } from '@angular/forms';
3. import { UsersService } from 'src/app/users-data/users.service';

4. @Component({
5.   selector: 'app-signup',
6.   templateUrl: './signup.component.html',
7.   styleUrls: ['./signup.component.scss']
8. })
9. export class SignupComponent implements OnInit {

10. constructor(private userService: UsersService) { }

11. ngOnInit() {
12. }

13. onSubmit(form: NgForm) {
14.   this.userService.addNewUser(form.value);
15. }

16. }

```

ما يهمنا هنا هو ما هو موجود في السطر 14 حيث استدعينا الدالة `addNewUser()` التي قمنا بإنشائها في الخطوة ثانياً في ملف `users.service.ts` ومررنا لها القيم التي يُدخلها المستخدم في النموذج لتقوم هذه الدالة بإضافة هذه البيانات إلى المصفوفة `USERS` كما قلنا سابقاً.

مع العلم أنني لم أشرح تعامل `angular` مع النماذج لكي لا أشتت ذهن القارئ ولن أراود الاستزادة في هذا الجانب فليراجع كتابي الذي عنوانه `angular forms` على هذا الرابط <http://kutub.info/library/book/22025> #kutub.info

رابعاً / نذهب إلى ملف `login.component.html` ونقوم بإجراء التعديلات التالية:

#### ملف login.component.html

```

1. <form #form="ngForm" (ngSubmit)="onSubmit(form)">
2.   <div class="card text-white bg-secondary mt">
3.     <div class="card-header text-center">Login</div>
4.     <div class="card-body">
5.       <div class="form-group">
6.         <input
7.           class="form-control"

```



```

8.         name="userName"
9.         ngModel
10.        placeholder="User Name Here"
11.    />
12. </div>
13. <div class="form-group">
14.     <input
15.         class="form-control"
16.         name="password"
17.         ngModel
18.         placeholder="Password Here"
19.     />
20. </div>
21. </div>
22. <div class="card-footer">
23.     <button class="btn btn-light form-control">Submit</button>
24. </div>
25. </div>
26. </form>

```

وفي ملف ts لهذا component نقوم بإجراء التعديلات التالية:

#### ملف login.component.ts

```

1. import { Component, OnInit } from '@angular/core';
2. import { UsersService } from 'src/app/users-data/users.service';
3. import { NgForm } from '@angular/forms';
4. import { Router } from '@angular/router';

5. @Component({
6.   selector: 'app-login',
7.   templateUrl: './login.component.html',
8.   styleUrls: ['./login.component.scss']
9. })

10. export class LoginComponent implements OnInit {

11.   constructor(private userService: UsersService) { }

12.   ngOnInit() {
13.   }

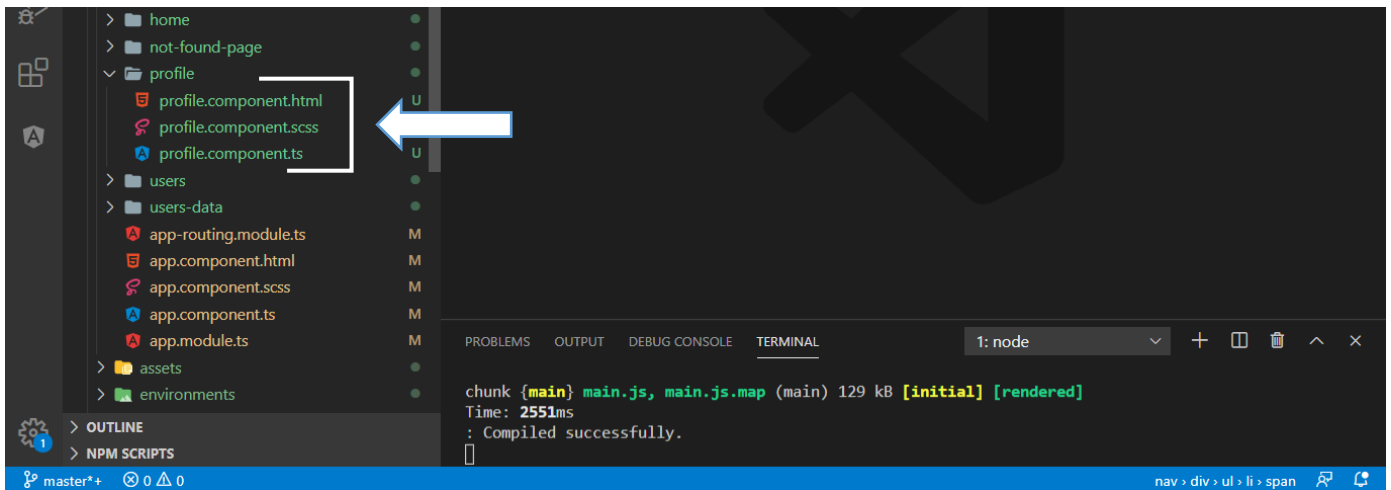
14.   onSubmit(form: NgForm) {
15.     const value = form.control.get('userName').value;
16.     this.userService.login(value);
17.   }

18. }

```

ما يهمنا ما هو موجود في الأسطر من 14 إلى 17 حيث قمنا بأخذ قيمة اسم المستخدم وتخزينها في متغير ومن ثم قمنا باستدعاء الدالة login () ونمرر لها قيمة اسم المستخدم.

خامساً / نضيف component جديد باسم profile وسوف نعرض فيه بيانات المستخدم عندما يقوم بتسجيل الدخول



نذهب إلى ملف profile.component.ts، ونقوم بإضافة التعديلات التالية:

```
ملف profile.component.ts
1. import { Component, OnInit, OnDestroy } from '@angular/core';
2. import { UsersService } from '../users-data/users.service';
3. import { ActivatedRoute, Router } from '@angular/router';
4. import { Subscription } from 'rxjs';
5. import { Users } from '../users-data/users';

6. @Component({
7.   selector: 'app-profile',
8.   templateUrl: './profile.component.html',
9.   styleUrls: ['./profile.component.scss']
10.})

11. export class ProfileComponent implements OnInit, OnDestroy {
12.   user$: Users;
13.   private subscription: Subscription;
14.   private id: number;

15.   constructor(
16.     private activeRouter: ActivatedRoute,
17.     private usersService: UsersService,
18.     private router: Router
19.   ) { }

20.   ngOnInit(): void {
21.     this.id = +this.activeRouter.snapshot.params.id;
22.     this.subscription = this.usersService.getUserById(this.id).subscribe(user => {
23.       this.user$ = user;
24.     });
25.   }

26.   ngOnDestroy() {
27.     this.subscription.unsubscribe();
28.   }

29. }
```

ونذهب إلى ملف profile.component.html، ونجري التغييرات التالية:

ملف profile.component.html

```
<div class="card">
  <div class="card-body">
    <ul>
      <li>
        
        <h3>{{ user$.name }}</h3>
        <p>
          User ID: <span>{{ user$.userId }}</span>
        </p>
        <p>
          UserName: <span>{{ user$.userName }}</span>
        </p>
        <p>
          User Password: <span>{{ user$.password }}</span>
        </p>
        <p>
          User City: <span>{{ user$.userCity }}</span>
        </p>
        <p>
          User Type: <span> {{ user$.userType }}</span>
        </p>
      </li>
    </ul>
  </div>
</div>
```

أما ملف profile.component.scss، فنجري فيه التعديلات التالية:

ملف profile.component.scss

```
div {
  margin: 20px;
}

.card {
  border: 0;
}

.card-body {
  background: rgba(253, 253, 253, 0.925);
  border: 1px solid silver;
  margin-bottom: 0px;
}
```

```

}

.card-footer {
  border: 1px solid silver;
  background: rgba(246, 246, 246, 0.925);
  margin-top: 0px;
}

ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 500px;
}

li {
  padding: 10px;
  overflow: auto;
}

li img {
  float: left;
  margin: 0 15px 0 0;
  border: 1px solid silver;
  height: 230px;
  border-radius: 50%;
}

li p {
  font: 200 12px/1.5;
  color: rgb(90, 87, 87);
  font-weight: bold;
}

li p span {
  font: 200 12px/1.5;
  color: rgb(0, 0, 0);
  font-weight: normal;
  font-style: italic;
}

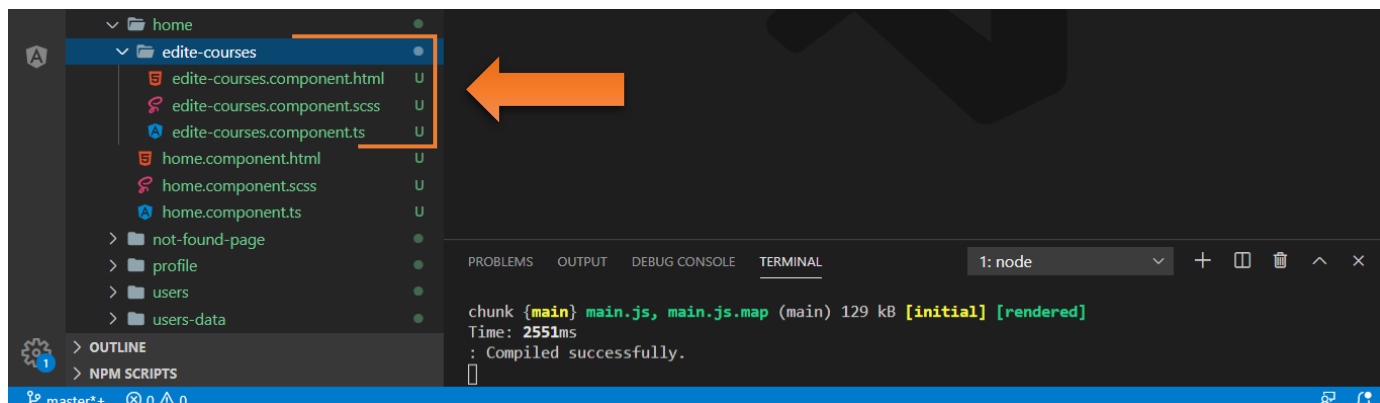
h3 {
  font: bold 20px/1.5 Helvetica, Verdana, sans-serif;
}

.div-style {
  height: 50px;
  background-color: rgb(214, 214, 214);
  border: 1px solid silver;
}

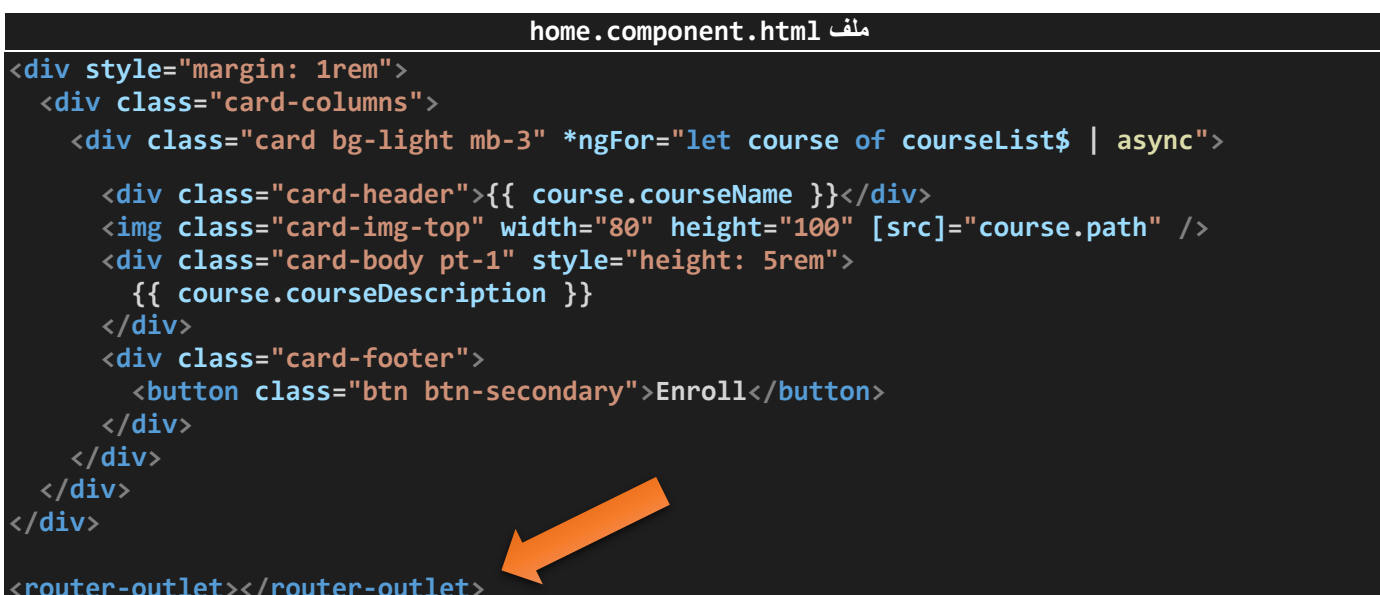
.span-style {
  border-radius: 50%;
  border: 1px solid green;
  background-color: rgb(75, 224, 75);
  height: 20px;
  width: 20px;
  margin-top: 12px;
  margin-right: 5px;
}

```

سادساً / نضيف component جديد باسم Edit Courses بحيث يكون أبن لـ component الاب ذو الاسم Home Component، ومهمة هذا component التعديل على الكورسات وهذا من صلاحيات مدير الموقع بمعنى أن هذا component لا يدخل إليه إلا المستخدم الذي قام بتسجيل الدخول وبفس الوقت مصنف على أنه مدير للنظام:

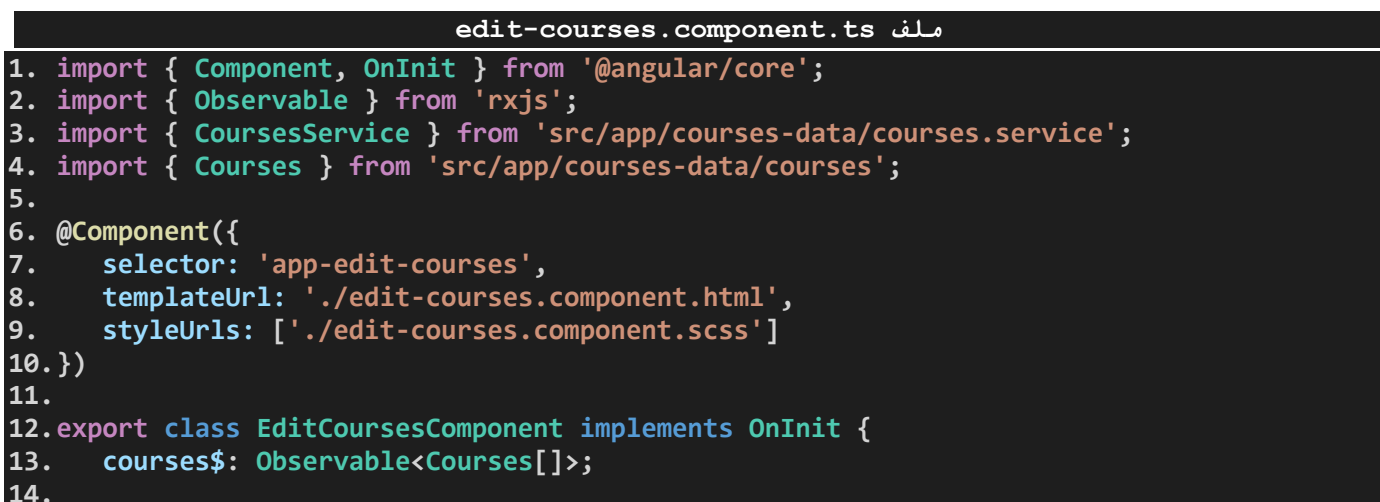


الآن نذهب إلى ملف home.component.html (الاب) ونضيف outlet له، كالتالي:



مع العلم أنه لا يلزم لكن من الأفضل كتابته كنوع من التنظيم،

الآن نذهب إلى ملف edit-courses.component.ts (الابن)، ونضيف فيه الأكواد التالية:



```

15.   constructor(private courses: CoursesService) { }
16.
17.   ngOnInit(): void {
18.       this.courses$ = this.courses.getAllCourses();
19.   }
20.
21. }

```

أما في ملف edit-courses.component.html، فنقوم بإجراء التعديلات التالية:

ملف edit-courses.component.html

```

<div class="container" style="padding-top: 1rem">
  <ul class="list-group" *ngFor="let course of courses$ | async">
    <li class="list-group-item">
      <span class="pull-left ">
        <img
          [src]="course.path"
          style="border: 1px solid silver; border-radius: 50%"
          width="50"
          height="50"
        />
      </span>
      <h6 style="padding-left: 2rem; display: inline">
        {{ course.courseName }}
      </h6>
      <span class="pull-right">
        <button class="btn btn-success" style="display: inline">
          Edit
        </button>
        <button class="btn btn-danger pull-right" style="margin-left: 3px">
          Delete
        </button>
      </span>
    </li>
  </ul>
</div>

```

وفي ملف edit-courses.component.scss فنضيف الاكواد التالية:

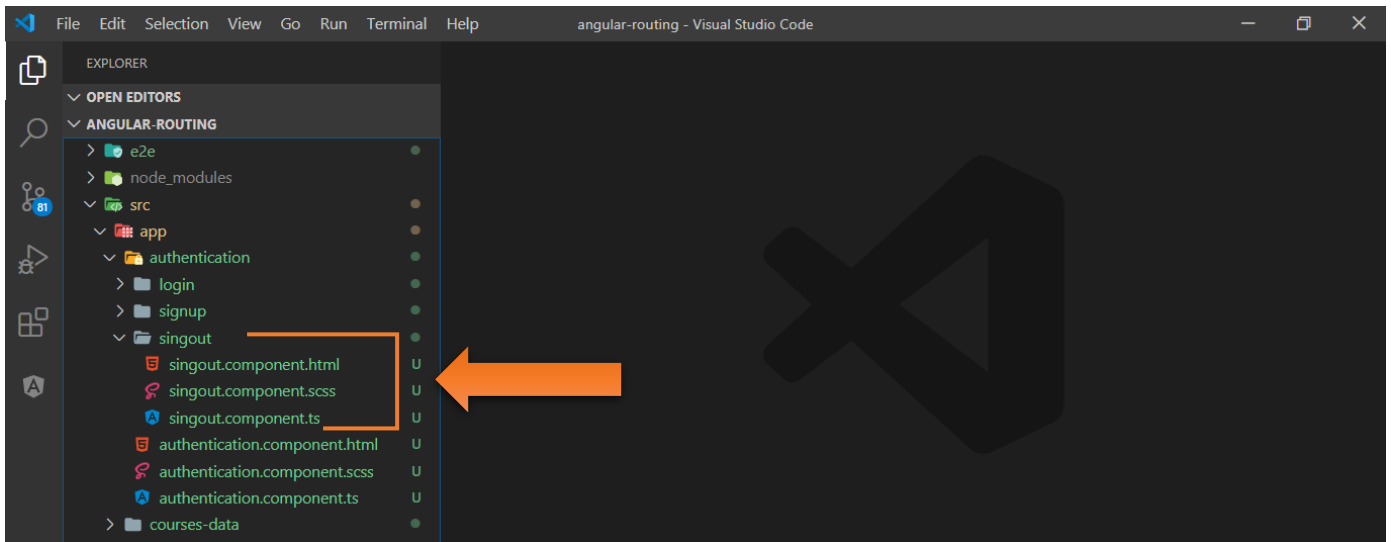
```

li {
  margin-bottom: 5px;
}

span.pull-right {
  padding-top: 6px;
}

```

سابعاً / نضيف component جديد باسم logout بحيث يكون ابن لـ component الاب ذو الاسم authentication، ومهمة هذا component بكل بساطة هو استدعاء وتنفيذ الدالة ( ) logoutUser الموجودة في service التي قمنا بإنشائها في الخطوة ثانياً، كالتالي:



الآن لنذهب إلى ملف singout.component.ts، ونضيف الأكواد التالية:

```
ملف singout.component.ts
1. import { Component, OnInit } from '@angular/core';
2. import { UsersService } from 'src/app/users-data/users.service';
3. @Component({
4.   selector: 'app-singout',
5.   templateUrl: './singout.component.html',
6.   styleUrls: ['./singout.component.scss']
7. })
8. export class SingoutComponent implements OnInit {
9.   constructor(private userService: UsersService) { }
10.  ngOnInit() {
11.    setTimeout(() => {
12.      this.userService.logoutUser();
13.    }, 1000);
14.  }
15. }
```

وكما هو واضح قمنا باستدعاء الدالة ( ) logoutUser وقمنا بتنفيذها بعد ١٠٠٠ (ثانية واحدة) لكي نحكي الاتصال بالسيرفر وانتظار الرد منه والذي بطبيعة الحال قد يأخذ وقت.

وفي ملف html لهذا component، أجرينا التعديلات التالية:

```
ملف singout.component.html
<h3 class="centered" style="top: 40%">الرجاء الإنتظار</h3>
<div class="lds-ripple centered">
  <div></div>
  <div></div>
</div>
```

ملف `logout.component.scss`

```
.lds-ripple {
  display: inline-block;
  position: relative;
  width: 80px;
  height: 80px;
  left: 50%;
}

.lds-ripple div {
  position: absolute;
  border: 4px solid #000000;
  opacity: 1;
  border-radius: 50%;
  animation: lds-ripple 1s cubic-bezier(0, 0.2, 0.8, 1) infinite;
}

.lds-ripple div:nth-child(2) {
  animation-delay: -0.5s;
}

@keyframes lds-ripple {
  0% {
    top: 36px;
    left: 36px;
    width: 0;
    height: 0;
    opacity: 1;
  }
  100% {
    top: 0px;
    left: 0px;
    width: 72px;
    height: 72px;
    opacity: 0;
  }
}

.centered {
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```



ثامناً / نذهب إلى ملف app-routing.module.ts، لإعادة تهيئة Routes بعدما قمنا بإجراء التعديلات السابقة، كالتالي:

#### ملف app-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { HomeComponent } from '../home/home.component';
4. import { UsersComponent } from '../users/users.component';
5. import { UsersListComponent } from '../users/users-list/users-list.component';
6. import { UserDetailsComponent } from '../users/user-details/user-details.component';
7. import { AuthenticationComponent } from '../authentication/authentication.component';
8. import { LoginComponent } from '../authentication/login/login.component';
9. import { SignupComponent } from '../authentication/signup/signup.component';
10. import { NotFoundPageComponent } from '../not-found-page/not-found-page.component';
11. import { SingoutComponent } from '../authentication/singout/singout.component';
12. import { ProfileComponent } from '../profile/profile.component';
13. import { EditCoursesComponent } from '../home/edit-courses/edit-courses.component';

14. const routes: Routes = [
15.   {
16.     path: 'home', children: [
17.       { path: '', component: HomeComponent },
18.       { path: 'edit-courses', component: EditCoursesComponent }
19.     ],
20.   },
21.   { path: 'profile/:id', component: ProfileComponent },
22.   {
23.     path: 'users',
24.     children: [
25.       { path: '', component: UsersComponent },
26.       { path: 'user-list', component: UsersListComponent },
27.       { path: 'user-details/:id', component: UserDetailsComponent },
28.     ],
29.   },
30.   {
31.     path: 'auth', children: [
32.       { path: '', component: AuthenticationComponent },
33.       { path: 'login', component: LoginComponent },
34.       { path: 'signup', component: SignupComponent },
35.       { path: 'signout', component: SingoutComponent }
36.     ],
37.   },
38.   { path: '', redirectTo: 'home', pathMatch: 'full' },
39.   { path: '**', component: NotFoundPageComponent }
40. ];

41. @NgModule({
42.   imports: [RouterModule.forRoot(routes)],
43.   exports: [RouterModule]
44. })
45. export class AppRoutingModule { }
```

تاسعاً / نذهب إلى ملف app.component.html، ونجري التعديلات التالية:

ملف app.component.html

```
<nav class="navbar navbar-expand-sm navbar-dark bg-dark Shadow">
  <div class="navbar-brand-two" href="#">
    
  </div>
  <span class="navbar-brand">Routing</span>
  <div class="justify-content-left">
    <ul class="navbar-nav">
      <li class="nav-item">
        <a
          routerLink="/home"
          routerLinkActive="is-active"
          [routerLinkActiveOptions]="{ exact: true }"
        ><span>Home</span></a>
      </li>
      <li class="nav-item">
        <a routerLink="/home/edit-courses" routerLinkActive="is-active"
        ><span>Edit Courses</span></a>
      </li>
      <li class="nav-item">
        <a routerLink="/users" routerLinkActive="is-active"
        ><span>Users</span></a>
      </li>
      <li class="nav-item">
        <a [routerLink]="['profile', id]" routerLinkActive="is-active"
        ><span>Profile</span></a>
      </li>
      <li class="nav-item">
        <a routerLink="/auth/login" routerLinkActive="is-active"
        ><span>Login</span></a>
      </li>
      <li class="nav-item">
        <a routerLink="/auth/signup" routerLinkActive="is-active"
        ><span>Signup</span></a>
      </li>
      <li class="nav-item">
        <a routerLink="/auth/signout" routerLinkActive="is-active"
        ><span>Signout</span></a>
      </li>
    </ul>
  </div>
</nav>
<router-outlet></router-outlet>
```

وفي ملف ts لهذا component، نضيف الأكواد التالية:

#### ملف app.component.ts

```
1. import { Component, OnInit } from '@angular/core';
2. import { StorageMap } from '@ngx-pwa/local-storage';

3. @Component({
4.   selector: 'app-root',
5.   templateUrl: './app.component.html',
6.   styleUrls: ['./app.component.scss']
7. })
8. export class AppComponent implements OnInit {
9.   id: string;
10.  constructor(private storageMap: StorageMap) { }

11.  ngOnInit(): void {
12.    this.storageMap.watch('id').subscribe((id: string) => {
13.      this.id = id;
14.    });
15.  }
16.}
```

ما يهمنا هو ما هو موجود في الاسطر من 12 إلى 14 حيث قمنا بالاستفادة من الدالة watch الموجودة في المكتبة الخارجية التي تطرقنا إليها في الخطوة ثانياً، بعد أن قمنا بعمل inject لها في السطر ١٠ واستدعيناها في السطر 2، حيث تقوم بمراقبة id وهو key الذي نخزن فيه قيمة id للمستخدم وعند حدوث أي تغير لقيمة هذا key يقوم بتخزين هذه القيمة في المتغير الذي اسميته ايضاً id والذي قمنا بتعريفه في السطر 9.

عاشراً / نذهب إلى ملف app.module.ts، ونتأكد أن FormsModule تم استدعائه بالشكل الصحيح.

#### ملف app.module.ts

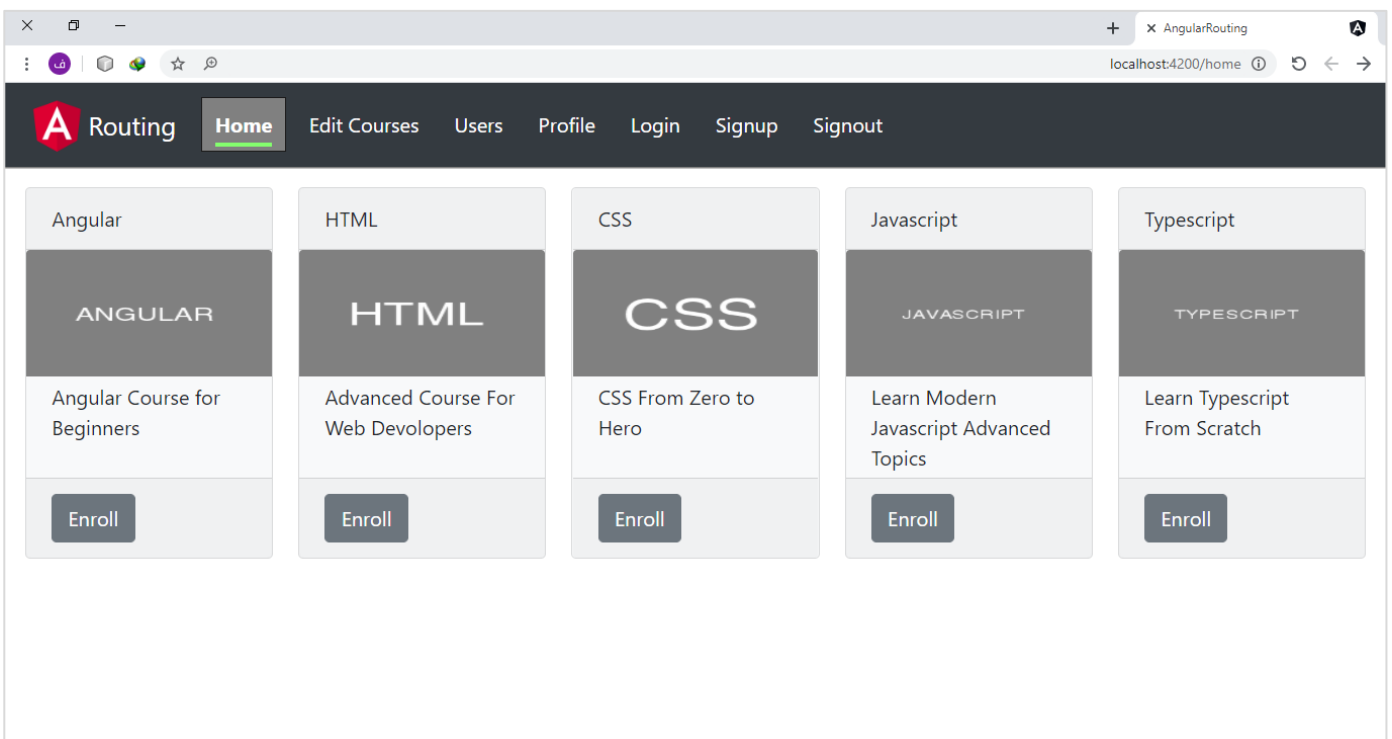
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AuthenticationComponent } from './authentication/authentication.component';
import { LoginComponent } from './authentication/login/login.component';
import { SingoutComponent } from './authentication/singout/singout.component';
import { SignupComponent } from './authentication/signup/signup.component';
import { HomeComponent } from './home/home.component';
import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
import { UsersComponent } from './users/users.component';
import { UserDetailsComponent } from './users/user-details/user-details.component';
import { UsersListComponent } from './users/users-list/users-list.component';
import { ProfileComponent } from './profile/profile.component';
import { EditCoursesComponent } from './home/edit-courses/edit-courses.component';
```

```






@NgModule({
  declarations: [
    AppComponent,
    AuthenticationComponent,
    LoginComponent,
    SingoutComponent,
    SignupComponent,
    HomeComponent,
    NotFoundPageComponent,
    UsersComponent,
    UserDetailsComponent,
    UsersListComponent,
    ProfileComponent,
    EditCoursesComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

الآن لنشاهد نتيجة هذه التعديلات على المتصفح:







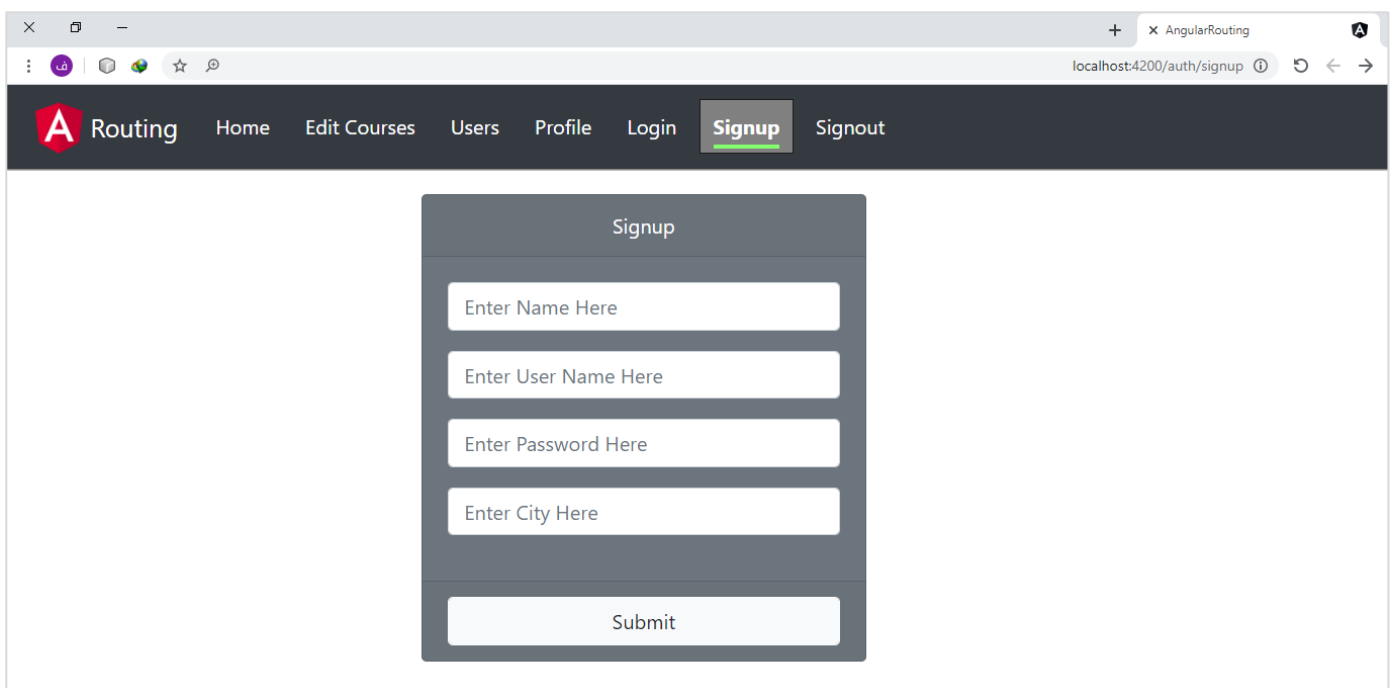
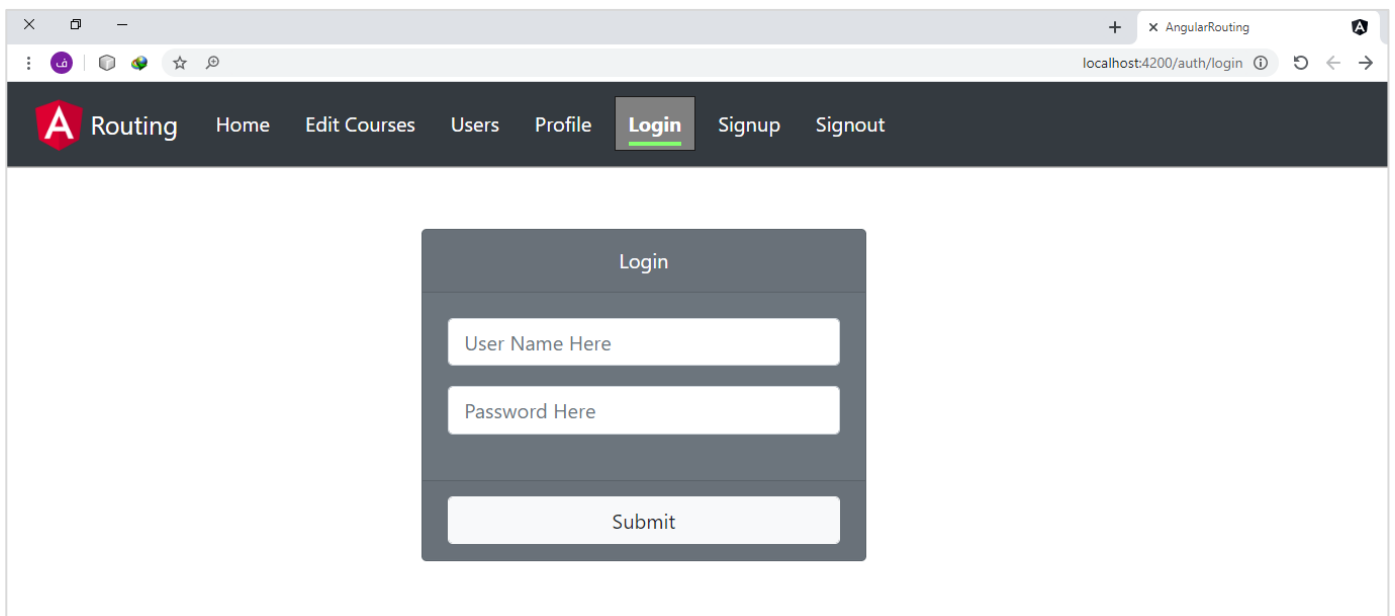
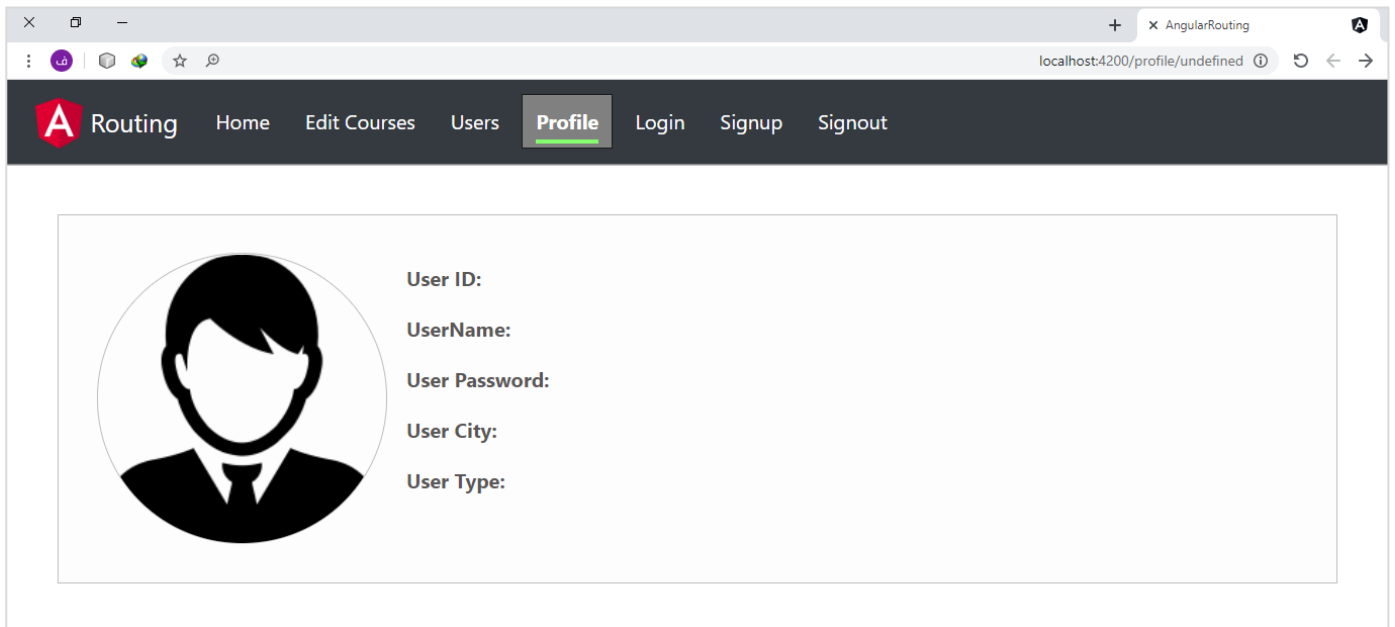
Routing Home **Edit Courses** Users Profile Login Signup Signout

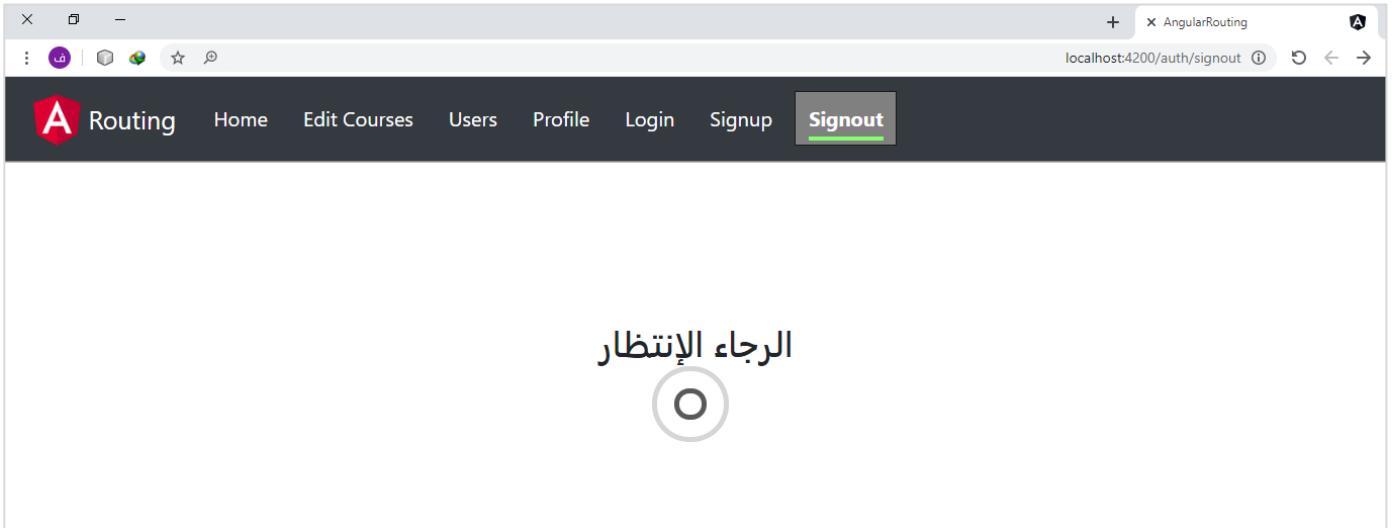
	Angular	<a href="#">Edit</a>	<a href="#">Delete</a>
	HTML	<a href="#">Edit</a>	<a href="#">Delete</a>
	CSS	<a href="#">Edit</a>	<a href="#">Delete</a>
	Javascript	<a href="#">Edit</a>	<a href="#">Delete</a>
	Typescript	<a href="#">Edit</a>	<a href="#">Delete</a>

Routing Home Edit Courses **Users** Profile Login Signup Signout

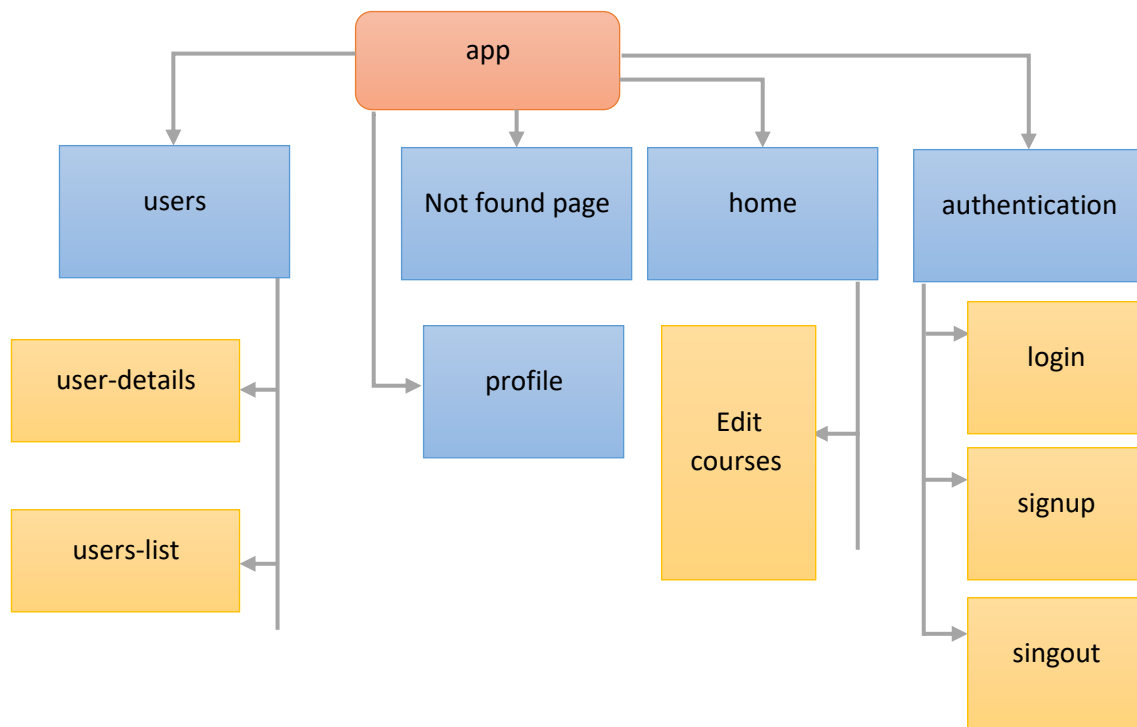
Search Users Here

	Faisal	<a href="#">View</a>	<a href="#">Active</a>
	Saad	<a href="#">View</a>	<a href="#">Active</a>
	Bader	<a href="#">View</a>	<a href="#">Active</a>
	Fahd	<a href="#">View</a>	<a href="#">Active</a>





وايضاً يمكن تمثيل هيكل البرنامج بعد التعديلات بالشكل التالي:



وبذلك نكون أنهينا من التعديلات المطلوبة في المشروع وأصبح جاهزاً لكي نشرح فيه جميع مفاهيم Angular Route Guards بجميع مفاهيمها وتقنياتها وحتى تفاصيلها الدقيقة، وسوف نبدأ بأول نوع من أنواع Angular Route Guard وهو CanActivate Guard.

### 2.3. CanActivate Guard

وهي من أشهر الأنواع وأكثرها استخداماً ويمكن استخدامها لحماية route واحد أو route وله عدة أبناء أو أحد الأبناء فقط، كما يمكن استخدام أكثر من CanActivate لroute الواحد على شكل مصفوفة، ويمكن توضيحها بالإشكال التالية:

## شكل (١)

### الحماية على مستوى route واحد

```
{path: 'any path', canActivate: [name of CanActivate class], component: any
```

خاصية لتعريف route انه سوف يُضاف له مجموعة من CanActivate، تكتب نفس ما هي

هنا يتم إضافة كلاسات CanActivate على شكل مصفوفة [ ] ويمكن إضافة أكثر من كلاس [class1, class2,..etc]

هنا يتم الحماية على مستوى Route واحد فقط

## شكل (٢)

### الحماية على مستوى route أب وله مجموعة من الأبناء

```
{
  path: 'any path', canActivate: [name of CanActivate class], component: any Component,
  children: [
    { path: 'any path', component: Son Component 1 },
    { path: 'any path', component: Son Component 2 },
    { path: 'any path', component: Son Component 3 },
  ]
}
```

خاصية لتعريف route انه سوف يُضاف له مجموعة من CanActivate، تكتب نفس ما هي

هنا يتم إضافة كلاسات CanActivate على شكل مصفوفة [ ] ويمكن إضافة أكثر من كلاس [class1, class2,..etc]

هنا تتم الحماية على مستوى Route الأب حيث تتم حماية هذا Route وجميع الأبناء الخاصة به



### شكل (٣)

#### الحماية على مستوى route أبن

```
{
  path: 'any path', component: any Component, children: [
    { path: 'any path', component: Son Component 1 },
    { path: 'any path', component: Son Component 2 },
    { path: 'any path', component: Son Component 3, canActivate: [name of CanActivate class]}
  ]
}
```

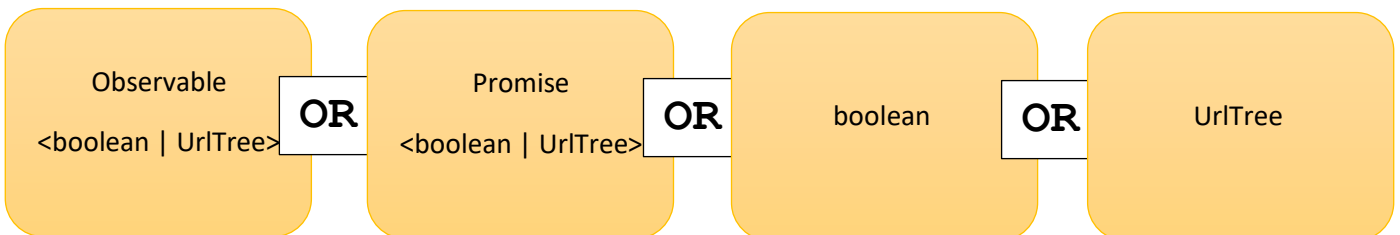
خاصية لتعريف route انه سوف يُضاف له مجموعة من CanActivate، تكتب نفس ما هي

هنا يتم إضافة كلاسات CanActivate على شكل مصفوفة [ ] ويمكن إضافة أكثر من كلاس [class1, class2,..etc]

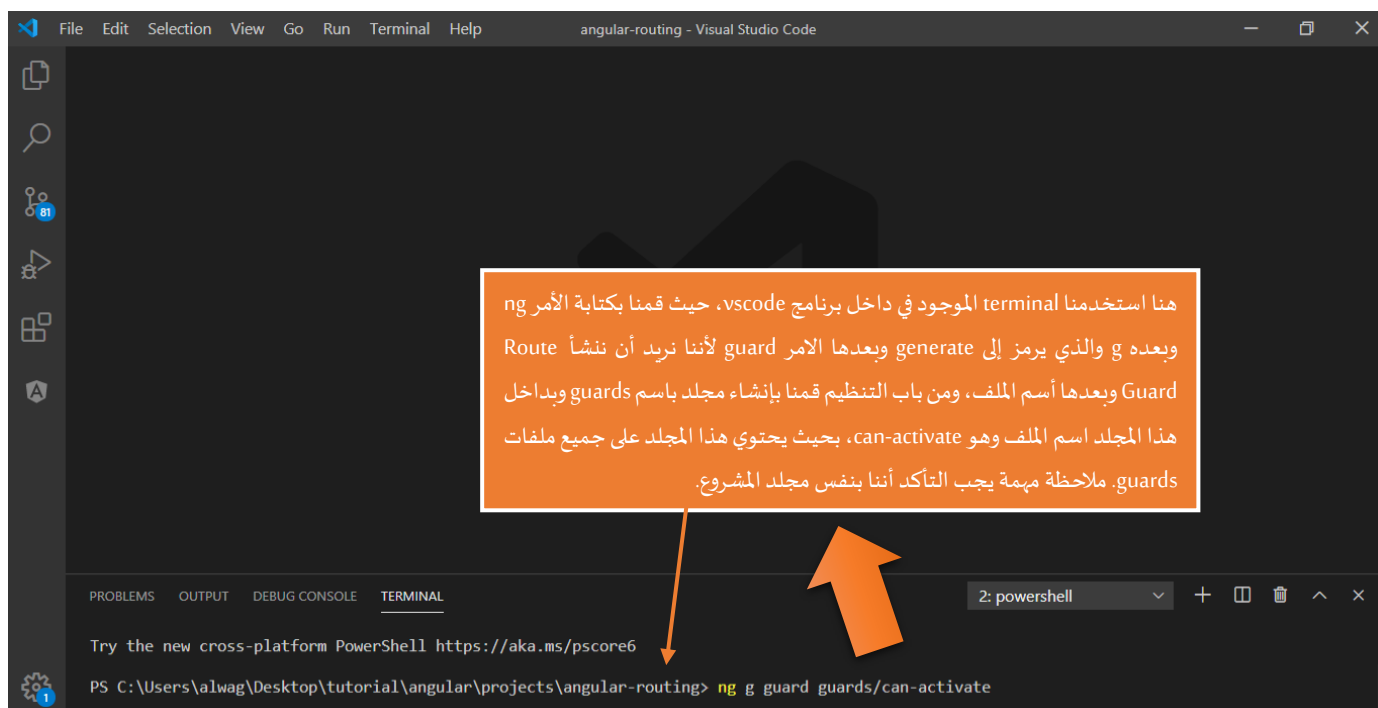
هنا تتم الحماية على مستوى Route الأب فقط

مع العلم أنه يمكن إضافة CanActivate على مستوى الأب والأبناء بنفس الوقت، على سبيل المثال نريد أن نحمي Route الأب بحيث لن يُسمح إلا للمستخدمين الذين قاموا بتسجيل الدخول لتطبيق، ولكن أحد Route الأبناء لا نريد فقط للمستخدمين الذين قاموا بتسجيل الدخول أن يتصفحوه بل لابد أن يكونوا مدراء للنظام، ففي هذه الحالة نعمل اثنين CanActivate واحد على مستوى الأب لتأكد من أن المستخدم قام بتسجيل الدخول والآخر على مستوى أحد الأبناء أو مجموعة منهم لتأكد من أن المستخدم لديه الصلاحية بأن يكون مدير للنظام admin.

ولو أتينا إلى التطبيق العملي لهذا CanActivate لوجدنا انه عبارة عن ملف (Typescript) ts وهذا الملف هو بالأساس عبارة عن service عادية معمول لها Injectable لكي نستطيع أن نعمل لها inject في ملفات المشروع الأخرى، وتحتوي على class أو أكثر من class معمول لها export لكي يمكن استخدامها خارج هذا الملف وكل class يقوم بعمل implements لinterface ذات الاسم CanActivate وهذا interface تحتوي على دالة لها نفس الاسم وتستقبل باراميتين، وهذه الدالة تُرجع أما:



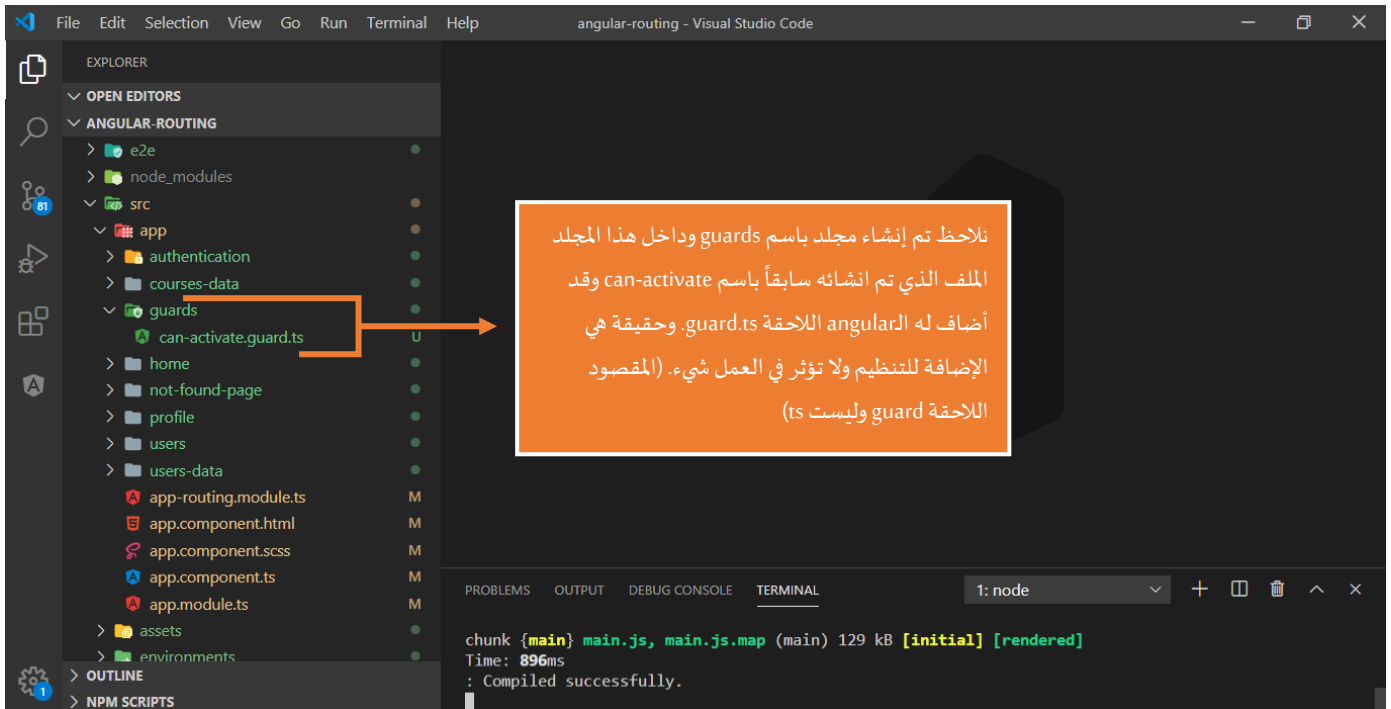
لا تقلق عزيزي المتعلم فجميع ما تم ذكره سابقاً عندما نقوم بتطبيقه عملياً سوف تجد أنه سهل ويمكن فهمه واستيعابه. أما من ناحية طريقة إنشاء CanActivate، فهناك عدة طرق ولكن أسهلها هو استخدام Angular CLI، لذلك سوف نذهب إلى terminal ونكتب الأمر التالي:



ثم نقوم بالضغط على زر Enter من لوحة المفاتيح، وننتظر قليلاً وسوف تظهر لنا الأوامر التالية:



بعد الاختيار نضغط على زر Enter وسوف يقوم بإنشاء الملفات المطلوبة، كما في الشكل التالي:



الآن لنستعرض محتويات هذا الملف:

```
ملف can-activate.guard.ts
1. import { Injectable } from '@angular/core';
2. import {
3.     CanActivate,
4.     ActivatedRouteSnapshot,
5.     RouterStateSnapshot,
6.     UrlTree
7. } from '@angular/router';
8. import { Observable } from 'rxjs';
9. @Injectable({
10.   providedIn: 'root'
11. })
12. export class CanActivateGuard implements CanActivate {
13.   canActivate( next: ActivatedRouteSnapshot, state: RouterStateSnapshot):
14.     Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
15.     return true;
16.   }
17. }
```

كما قلنا سابقاً هذا الملف هو في الحقيقة عبارة عن ملف service معمول له injectable كما هو موجود في الاسطر من 9 الى 11، وهذا الأمر يتيح لنا عمل inject لهذا الملف في ملفات أخرى من المشروع كما سوف نرى لاحقاً بإذن الله، لفهم أعمق عن Services الرجاء مراجعة الكتاب الثاني Angular Components and Services.

وايضاً كما قلنا سابقاً هذا الملف يعمل export لكلاس كما في السطر 12 حيث تم إعطائه اسم مقارب لاسم الملف، فاسم الملف can-activate.guard.ts وتم حذف (-) و (.) و (ts) ودمج الكلمات مع بعضها البعض بشكل تلقائي، وهذا الكلاس يعمل implements interface تحمل الاسم CanActivate بعد استدعائها في السطر 3، وهذا الكلاس هو اهم جزء في هذا الملف حيث فيه نقوم بكتابة logic البرمجي الذي يحدد ما إذا يُسمح او لا يسمح بزيارة route معين عند اختيار المستخدم له، كما أن اسم هذا الكلاس هو الذي نضيفه إلى مصفوفة أسماء CanActivate التي ذكرناها سابقاً، مع ملاحظة انه يمكن إضافة أكثر من كلاس بنفس الملف، فمثلاً كلاس يتحقق هل المستخدم قام بتسجيل الدخول او لا والكلاس الثاني لتأكد هل المستخدم هو مدير للنظام أم لا...الخ.

ونكرر ما تم ذكره سابقاً أن هذا interface يحمل دالة method لها نفس الاسم ( ) canActivate كما في الاسطر من 13 الى 16 (مع العلم أن الاسطر 13 و 14 هي في الحقيقة سطر واحد ولكن وضعناها على سطرين ليسهل قراءتها)، وهذه الدالة تستقبل باراميتين واحد يحمل الاسم next وهو من النوع ActivatedRouteSnapshot والثاني يحمل الاسم state وهو من النوع RouterStateSnapshot كم في السطر 13، اما في السطر 14 فيحتوي على الأنواع التي يمكن ان تُعيدها (تُرجعها) هذه الدالة، وقبل البدء بشرح هذه الأنواع بشيء من التفصيل، اريد أن ابين محتوى هذه الدالة وبشكل افتراضي محتواها يُعيد القيمة true وهي من النوع boolean أي بمعنى آخر وبمفهومها البسيط إذا كانت تُعيد true فسوف تسمح لتوجيه مستخدم التطبيق إلى Route المراد حمايته وإذا تُعيد false فسوف تلغي التوجيه. لذلك لنقوم بتطبيق هذا الامر عملياً.

لنذهب إلى ملف app-routing.module.ts ونقوم بحماية Route الذي يحمل الاسم users، وذلك عن طريق كتابة اسم هذا الكلاس CanActivateGuard، كالتالي:

ملف app-routing.module.ts

```

1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { HomeComponent } from './home/home.component';
4. import { UsersComponent } from './users/users.component';
5. import { UsersListComponent } from './users/users-list/users-list.component';
6. import { UserDetailsComponent } from './users/user-details/user-details.component';
7. import { AuthenticationComponent } from './authentication/authentication.component';
8. import { LoginComponent } from './authentication/login/login.component';
9. import { SignupComponent } from './authentication/signup/signup.component';
10. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
11. import { SingoutComponent } from './authentication/singout/singout.component';
12. import { ProfileComponent } from './profile/profile.component';
13. import { EditeCoursesComponent } from './home/edite-courses/edite-courses.component';
14. import { CanActivateGuard } from './guards/can-activate.guard';

15. const routes: Routes = [
16.   {
17.     path: 'home', children: [
18.       { path: '', component: HomeComponent },
19.       { path: 'edite-courses', component: EditeCoursesComponent }
20.     ]
  
```

```

21. },
22. { path: 'profile/:id', component: ProfileComponent },
23. {
24.   path: 'users', canActivate: [CanActivateGuard], children: [
25.     { path: '', component: UsersComponent },
26.     { path: 'user-list', component: UsersListComponent },
27.     { path: 'user-details/:id', component: UserDetailsComponent },
28.   ]
29. },
30. {
31.   path: 'auth', children: [
32.     { path: '', component: AuthenticationComponent },
33.     { path: 'login', component: LoginComponent },
34.     { path: 'signup', component: SignupComponent },
35.     { path: 'signout', component: SingoutComponent }
36.   ]
37. },
38. { path: '', redirectTo: 'home', pathMatch: 'full' },
39. { path: '**', component: NotFoundPageComponent }
40.];

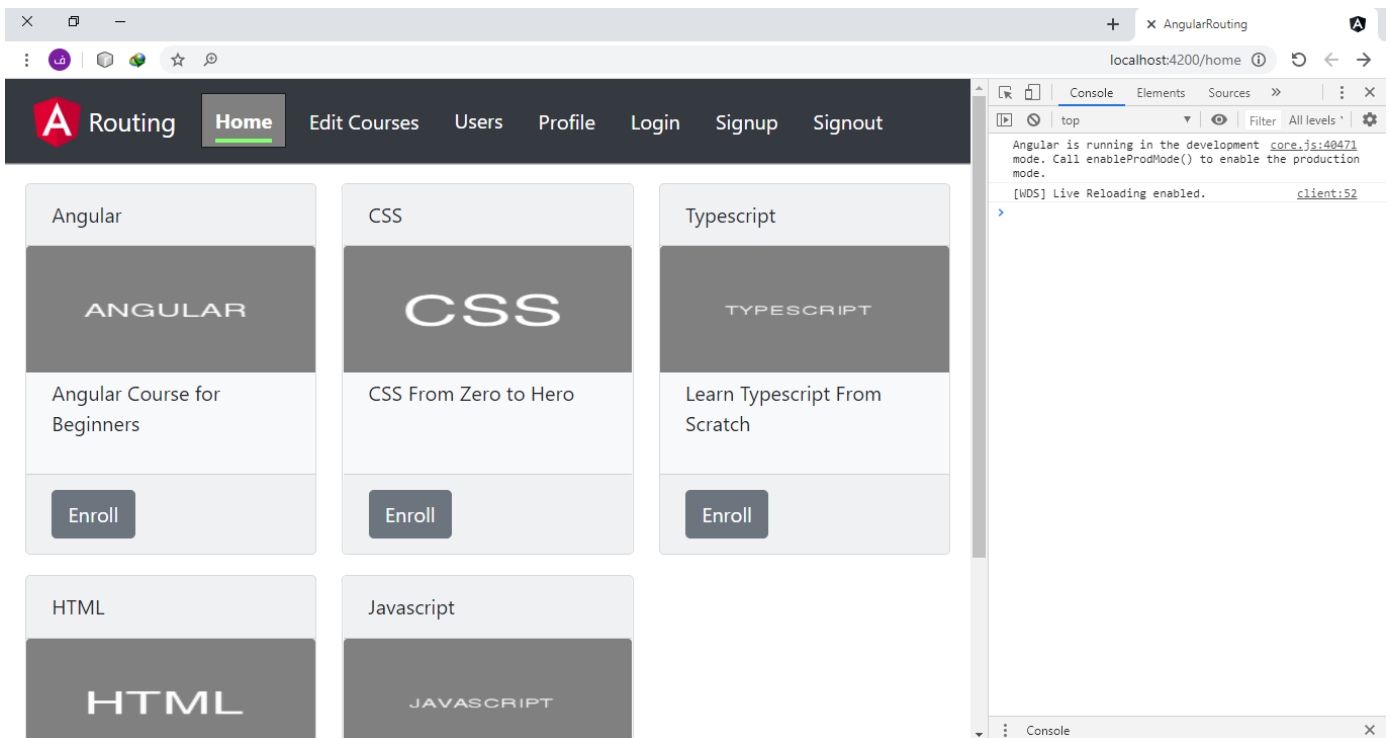
41.@NgModule({
42.  imports: [RouterModule.forRoot(routes)],
43.  exports: [RouterModule]
44.})

45.export class AppRoutingModule { }

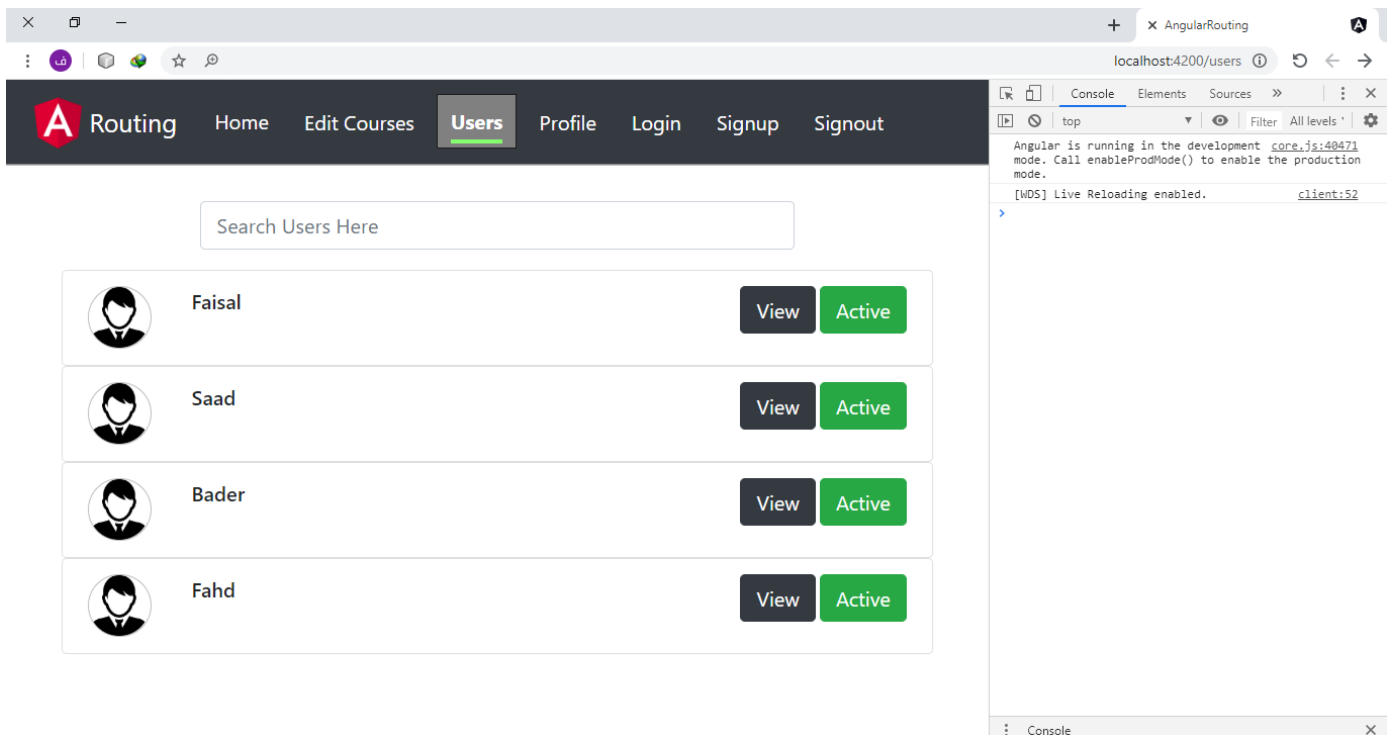
```

نلاحظ هنا أننا أضفنا الخاصية canActivate وأسندها لها القيمة اسم هذا الكلاس canActivateGuard، إلى Route الذي يحمل path ذو القيمة users

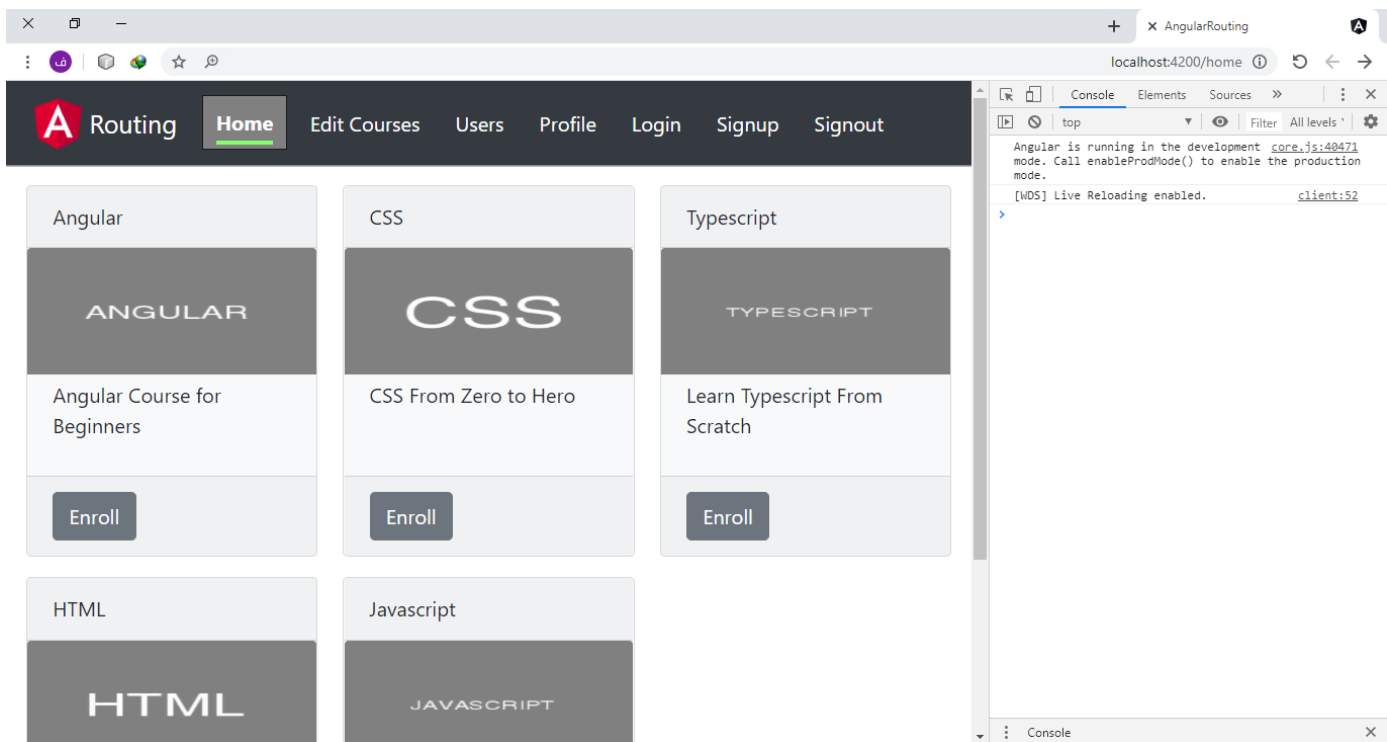
الآن وكنوع من التجربة لنحفظ التعديلات التي قمنا بها ولنذهب إلى المتصفح ونرى النتيجة:



الآن لنقوم بالضغط على التبويب Users ونرى ماذا سوف يحدث.



نلاحظ أنه لم يحدث شيء وذلك بسبب أن الدالة `canActivate` الموجود في الكلاس تُعيد القيمة `true`، الآن لنذهب إلى ملف `can-activate.guard.ts` ونقوم بتغيير القيمة إلى `false`، ومن ثم نحفظ التعديلات ونرى النتيجة في المتصفح:



نلاحظ انه مهما حاولنا الضغط على التبويب `Users` فإنه لن يعمل توجيه لهذا `Route` بسبب أن الدالة أصبحت الآن تُعيد `false` أي بمعنى آخر قم بحماية هذا `Route` ولا تسمح لأحد بالوصول إليه.

الآن بعدما وضعنا `Route Guard` بمفهومه البسيط، سوف نقوم الآن بتوضيح الأنواع الأخرى غير النوع `Boolean`.

الآن لنرجع إلى ملف `can-activate.guard.ts`، ونعيد استعراض محتوياته مرة أخرى:

```

1. import { Injectable } from '@angular/core';
2. import {
3.     CanActivate,
4.     ActivatedRouteSnapshot,
5.     RouterStateSnapshot,
6.     UrlTree
7. } from '@angular/router';
8. import { Observable } from 'rxjs';

9. @Injectable({
10.     providedIn: 'root'
11. })

12. export class CanActivateGuard implements CanActivate {
13.     canActivate( next: ActivatedRouteSnapshot, state: RouterStateSnapshot):
14.         Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
15.         return true;
16.     }
17. }

```

نلاحظ أن هذه الدالة تُعيد أنواع أخرى غير النوع boolean، وأول نوعين هما Observable أو Promise وهاذان النوعان من وجهة نظري وجهان لعملة واحدة فالنوع الأول Observable ومكتبتها rxjs هي المفضلة لدى angular أما Promise فهو من أساسيات الجافا سكريبت وكلا هذين النوعين يتعاملان مع async data او البيانات غير التزامنية ومن أمثلتها لو افترضنا أننا نريد تسجيل الدخول والتأكد من ذلك ففي هذه الحالة نحتاج أن نتصل في قاعدة البيانات وهذه القاعدة تكون في الغالب على server، ففي هذه الحالة نحتاج إلى هذه التقنيات (Observable – Promise) لكي نتعامل مع البيانات من حيث الوقت المستغرق ونوع وشكل هذه البيانات والتعامل مع الأخطاء أن وجدت..الخ.

وبما أنهما Generic كما هو واضح من < > فهما أيضاً يُعيدان إما boolean أو UrlTree بمعنى آخر إذا كنت تتعامل مع بيانات على server فأنا لي الحرية في استخدام Observable أو Promise بحيث لو اخترت Promise فإننا أُرْجِع true أو false من النوع Promise أو UrlTree (سوف نشرحها بعد قليل) من النوع Promise، ولتبسيط أكثر نستطيع أن نقول بشكل عام إذا كنت أتأكد من السيرفر بشكل مباشر من دالة () canActivate فاستخدم Promise او Obsevable، أو إذا كانت دالة الاتصال بقاعدة البيانات والتأكد من أن المستخدم قام بتسجيل الدخول من عدمه منفصله، كأن تكون في ملف service مثلاً، ففي هذه الحالة ننظر ماذا تُعيد إذا كانت تُعيد Observable فيجب علينا هنا في الدالة () canActivate أن نتعامل مع Observable ونفس الوضع مع Promise، أما إذا كانت دالة الاتصال مع قاعدة البيانات بعد التأكد من أن المستخدم قام بتسجيل الدخول او لا، وتخزين هذه الحالة في متغير عام من النوع boolean بحيث نجعل قيمة هذا المتغير true إذا المستخدم قام بتسجيل الدخول والقيمة false إذا كان غير ذلك، ففي هذه الحالة نكتفي أن نُرجع في الدالة () canActivate القيمة true او false بحسب قيمة المتغير بدون observable او promise.

وحقيقة الكلام في Promise و Observable ومكتبة rxjs يطول وليس هنا المقام لشرحها، ولعلنا نفرد كتاب كامل نشرح فيه هذه التقنيات والفائدة منها.

ولكن السؤال الذي يطرح نفسه ما هو النوع UrlTree؟ وهذا ما سوف نتطرق إليه في النقطة التالية.

### 1.2.3. UrlTree Type:

ولفهم هذا النوع لنعطي مثال، في حالتنا السابقة إذا كان المستخدم في صفحة home مثلاً وأراد الدخول إلى صفحة users ولم يتم بتسجيل الدخول فإن الدالة canActivate() تُعيد false، وبذلك سوف يلغي التوجيه ويبقى المستخدم في صفحة home ولكن لو أردنا أن يكون التطبيق أكثر احترافية، نريد أن نقوم بإعادة توجيه المستخدم إلى صفحة login لكي يقوم بتسجيل الدخول، وللقيام بهذا الأمر نستخدم الأمر navigate كما كنا نستخدم دائماً في التوجيه البرمجي ومن ثم نُعيد الأمر false، كالتالي:

```
can-activate.guard.ts ملف
import { Injectable } from '@angular/core';
import {
  CanActivate,
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  UrlTree,
  Router
} from '@angular/router';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})

export class CanActivateGuard implements CanActivate {
  constructor(private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot):
    Observable<boolean |
      UrlTree> |
    Promise<boolean |
      UrlTree> |
    boolean |
    UrlTree {
    this.router.navigate(['auth/login']);
    return false;
  }
}
```

كما هو واضح في الكود المشار إليه في الأعلى قمنا بتوجيه المستخدم إلى رابط محدد ومن ثم أرجعنا القيمة false، ولكن في الإصدار السابع وما فوق أصبحت بإمكاننا أن نُعيد الرابط نفسه في حال فشل التحقق بدلاً من الطريقة السابقة، وفي حالة إرجاع الرابط نفسه يُسمى هذا النوع UrlTree، ونستخدم معه بالعادة الأمرين createUrlTree او parseUrl والفرق الأساسي بينهما أن parseUrl يقبل باراميتراً واحداً وهو الرابط بشكله النصي string وليس مصفوفة نصية، فهو يقوم بتحليل الرابط ومن ثم يعمل توجيهه إلى هذا الرابط أما createUrlTree فهو يقبل بارامترين الأول الرابط والثاني



navigationExtras (تكلّمنا عنها بشكل مفصل سابقاً)، وايضاً يقوم بتحليل وتوجيهه للرباط، لذلك لنقوم بتعديل الكود السابق، كالتالي:

ملف can-activate.guard.ts

```
import { Injectable } from '@angular/core';
import {
  CanActivate,
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  UrlTree,
  Router
} from '@angular/router';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})

export class CanActivateGuard implements CanActivate {
  constructor(private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot):
    Observable<boolean |
      UrlTree> |
    Promise<boolean |
      UrlTree> |
    boolean |
    UrlTree {
    return this.router.parseUrl('auth/login');
  }
}
```



مع ملاحظة أنه لا يجب إن نُعيد false فبمجرد أن نعيد الرابط فإن angular سوف يفهم أن التحقق فشل وسوف يقوم بإعادة توجيهه المستخدم إلى الرابط الجديد، لكن السؤال الذي يطرح نفسه النوع UrlTree ما الذي يقدمه لنا من مميزات؟

للإجابة على هذا السؤال لنقوم بتخزين القيمة في متغير من النوع UrlTree ومن ثم نقوم بعمل console.log لهذه المتغير ونرى النتيجة، كالتالي:

ملف can-activate.guard.ts

```
import { Injectable } from '@angular/core';
import {
  CanActivate,
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  UrlTree,
  Router
} from '@angular/router';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
```

```

}))

export class CanActivateGuard implements CanActivate {
  constructor(private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot):
    Observable<boolean |
      UrlTree> |
    Promise<boolean |
      UrlTree> |
    boolean |
    UrlTree {
    const tree: UrlTree = this.router.parseUrl('auth/login');
    console.log(tree);
    return tree;
  }
}

```



الآن لنحفظ التعديلات ونذهب إلى المتصفح ونرى النتيجة، ولكي نرى النتيجة بشكل جيد لنضغط في البداية على علامة التبويب Home ومن ثم نضغط على علامة التبويب Profile، مع فتح نافذة console في المتصفح ولنرى النتيجة:

نلاحظ أن console طبع لنا مجموعة كبيرة من الخصائص والكائنات التي تصف لنا هذه الرابط ونستطيع الاستفادة من هذه الخصائص بعدة أمور مثلاً هل يحتوي على fragment أم لا وهل يحتوي queryParam والroot وهو يحتوي على جميع أجزاء الرابط على شكل مصفوفة... الخ.

ولو ذهبنا إلى تعريف UrlTree في angular Documentation الخاص بموقع angular على شبكة الانترنت لوجدنا انهم عرفوه على انه interface يحتوي على الخصائص التالية:

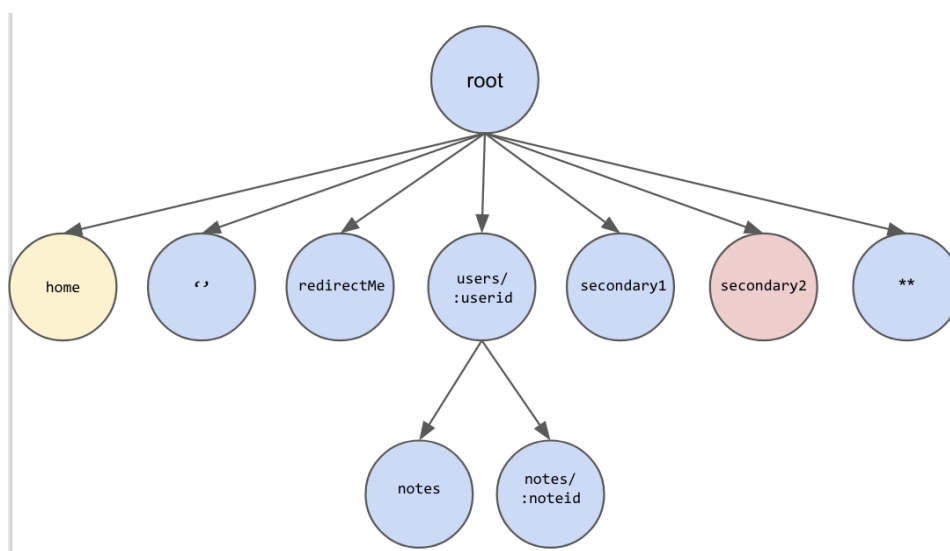
```
interface UrlTree {
    root: UrlSegmentGroup
    queryParams: Params
    fragment: string | null
    paramMap: ParamMap
    toString(): string
}
```

وقد تطرقنا سابقاً على بعض هذه الخصائص مثل queryParams – fragment – paramMap، أما الـ root فهو من النوع UrlSegmentGroup وهذا النوع يعيد جميع أجزاء الرابط على شكل مصفوفة بحيث أن كل جزء هو الذي يكون بين / /، ولفهم اعمق لهذا الـ UrlTree، لنضرب المثال التالي:

لنفرض أنه لدينا التهيئة التالية:

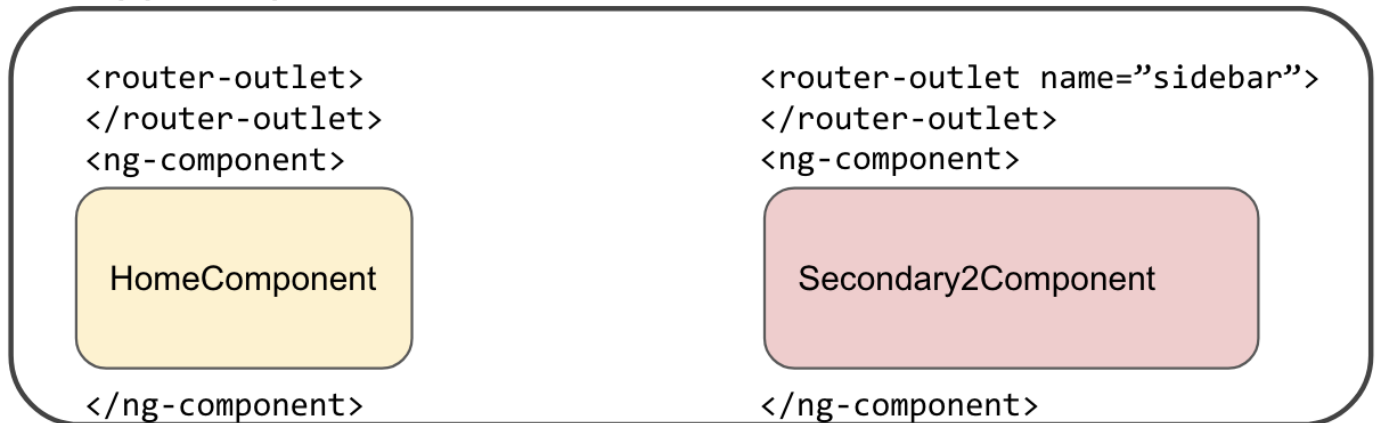
```
const ROUTES: Route[] = [
  { path: 'home', component: HomeComponent },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'redirectMe', redirectTo: 'home', pathMatch: 'full' },
  { path: 'users/:userid', component: UserComponent,
    children: [
      { path: 'notes', component: NotesComponent },
      { path: 'notes/:noteid', component: NoteComponent }
    ]
  },
  { path: 'secondary1', outlet: 'sidebar', component: Secondary1Component },
  { path: 'secondary2', outlet: 'sidebar', component:
    Secondary2Component },
  { path: '**', component: PageNotFoundComponent },
];
```

ويمكن تمثيل هذه التهيئة بالشكل المبسط التالي:



بحيث يكون لدينا اثنان outlet واحد رئيسي primary والثاني فرعي secondary باسم sidebar في AppComponent، كما في الشكل المبسط التالي:

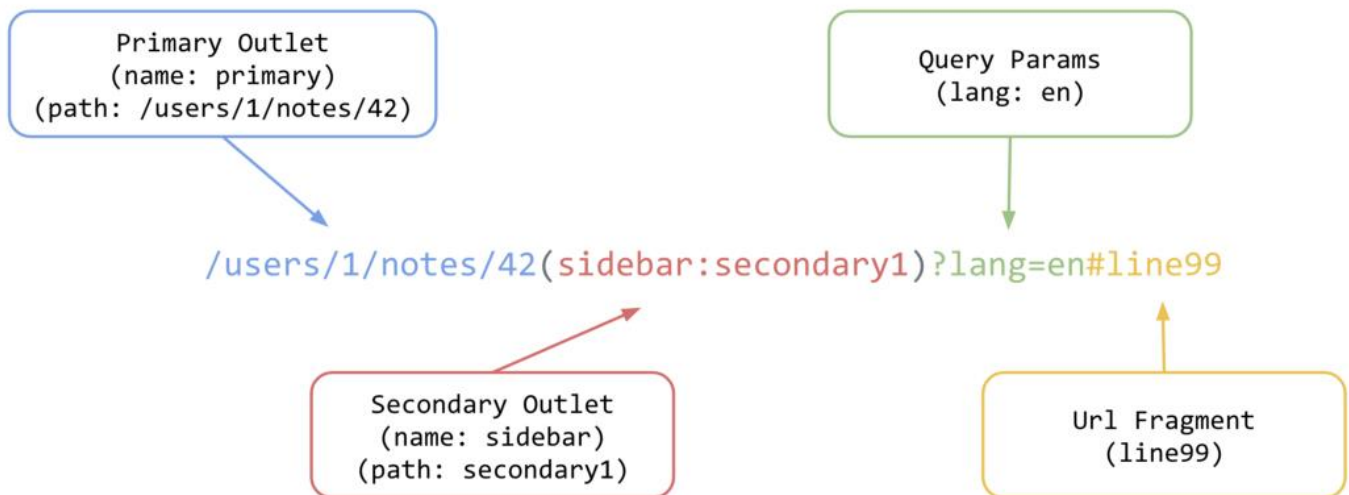
## AppComponent



الآن لنفرض أننا نريد توجيه المستخدم إلى الرابط التالي:

```
'/users/1/notes/42 (sidebar:secondary1)?lang=en#line99'
```

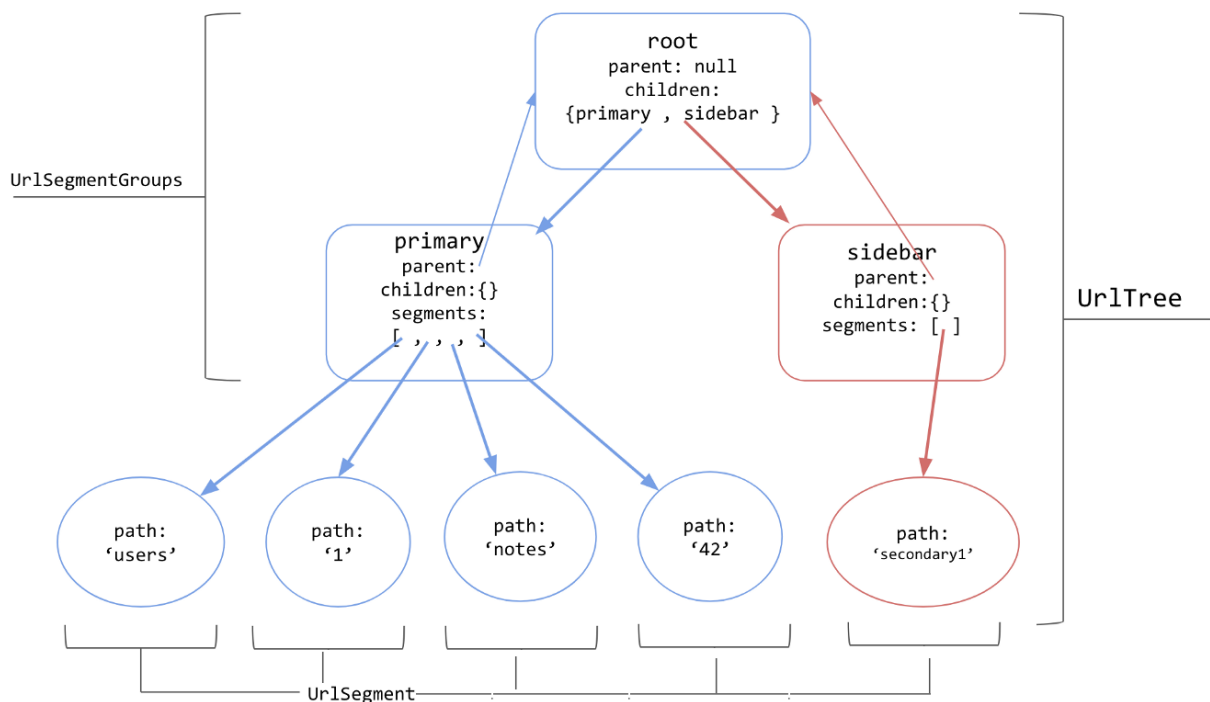
وهذا الرابط url هو في الحقيقة عبارة عن مجموعة group من الاجزاء segments، ويمكن تحليل هذا الرابط parse url بالشكل المبسط التالي:



لذلك عندما نكتب الأمر التالي:

```
const url = '/users/1/notes/42 (sidebar:secondary1)?lang=en#line99';  
const tree: UrlTree = this.router.parseUrl(url);
```

فإن angular سوف يقوم بتحويل الرابط من قطعة نصية عادية إلى سلسلة من الأجزاء او نستطيع ان نطلق عليها بصيغة أخرى tree structure، ويمكن تمثيلها بالشكل التالي:



حيث الخطوط الزرقاء ترمز إلى primary outlet والحمراء ترمز إلى secondary outlet.

وبعد أن قمنا بتحويل الرابط من النوع النصي العادي إلى النوع UrlTree أصبح بإمكاننا القيام بأمر كثيرة على هذا الرابط منها على سبيل المثال:

```

// line99
const fragment = tree.fragment;

// lang=en
const queryParams = tree.queryParams;

// gets the UrlSegmentGroup for the primary router outlet
const primary: UrlSegmentGroup = tree.root.children[PRIMARY_OUTLET];

// gets the UrlSegmentGroup for the secondary router outlet (sidebar)
const sidebar: UrlSegmentGroup = tree.root.children['sidebar'];

// returns all UrlSegments for the primary outlet.
['users', '1', 'notes', '42']
const primarySegments: UrlSegment[] = primary.segments;

// returns all UrlSegments for the secondary outlet. ['secondary1']
const sidebarSegments: UrlSegment[] = sidebar.segments;
  
```

```
class MyComponent {
  constructor(router: Router) {
    const url = '/team/33/(user/victor//support:help)?debug=true#fragment';
    const tree: UrlTree = router.parseUrl(url);
    const f = tree.fragment; // return 'fragment'
    const q = tree.queryParams; // returns {debug: 'true'}
    const g: UrlSegmentGroup = tree.root.children[PRIMARY_OUTLET];
    const s: UrlSegment[] = g.segments; // returns 2 segments 'team' and '33'
    g.children[PRIMARY_OUTLET].segments; // returns 2 segments 'user' and 'victor'
    g.children['support'].segments; // return 1 segment 'help'
  }
}
```

خلاصة الحديث عن هذا النوع انه نوع يتعامل مع الرابط بإمكانيات أكبر بدلاً من التعامل مع سابقاً على انه قطعة نصية متكاملة، ونستطيع التعامل معه سواء عن طريق الامر `createUrlTree` او `parseUrl`.

ولأن بعد ما استفضنا في شرح هذا النوع `UrlTree` ومحاولة لفهم هذا النوع، نرجع لاستكمال محور حديثنا عن `Route` `Guards` والنوع الأول `CanActivate`، وهو ما سوف نذكره في النقطة التالية:

### 2.2.3. استكمال شرح `CanActivate Guard`:

الآن لنذهب إلى ملف `can-activate-guard.ts` ونقوم بحذف الأكواد التي وضعناها لفهم النوع `UrlTree`، بحيث يكون كالشكل التالي:

ملف `can-activate.guard.ts`

```
import { Injectable } from '@angular/core';
import {
  CanActivate,
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  UrlTree,
  Router
} from '@angular/router';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class CanActivateGuard implements CanActivate {
  constructor(private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot):
    Observable<boolean |
      UrlTree> |
    Promise<boolean |
      UrlTree> |
    boolean |
    UrlTree { }
```

في هذا Route Guard نريد منه أن يتحقق هل المستخدم قام بتسجيل الدخول ام لا، ونستطيع التحقق من هذا الامر بكل بساطة عن طريق المتغير الموجود في الملف users.service.ts حيث قمنا بتعريف متغير باسم isLogin\$ وهو Observable وهو من النوع BehaviorSubject وهذا النوع يسمح لنا بعمل مشاركة لقيمة المتغير في جميع ملفات المشروع، لذلك وبناءً على ماتم ذكره سوف نستفيد من هذا المتغير بالتحقق من قيمته إذا كانت true نجعل الدالة canActivate تُعيد true وإذا كان false نجعل الدالة تُعيد الرابط عبي شكل UrlTree او نكتفي فقط بإن تُعيد false، ويمكن حلها بأكثر من طريقة، من هذه الطرق هو إن نأخذ القيمة من المتغير isLogin\$ على شكل boolean عادي ونخزنها في ثابت وعن طريق هذا الثابت نقوم بعمل شرط إذا كانت القيمة true تُعيد true وإذا كانت القيمة false تُعيد false ونعمل navigate لصفحة تسجيل الدخول، وهناك طريقة أخرى وهو بما أن المتغير هو observable فلنستفيد من الإمكانيات التي يقدمها ونعيد Observable(true) او في حال الفشل نُعيد Observable(UrlTree)، كالتالي:

ملف can-activate.guard.ts

```
import {Injectable} from '@angular/core';
import {
  ActivatedRoute,
  ActivatedRouteSnapshot,
  CanActivate,
  Router,
  RouterStateSnapshot,
  UrlTree
} from '@angular/router';
import {Observable} from 'rxjs';
import {UserService} from '../users-data/users.service';
import {map} from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class CanActivateGuard implements CanActivate {
  constructor(
    private router: Router,
    private userService: UserService,
  ) { }
  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot):
    Observable<boolean |
      UrlTree> |
    Promise<boolean |
      UrlTree> |
    boolean |
    UrlTree {
    // const isLogin = this.userService.isLogin$.value;
    // if (isLogin) { return true; }
    // this.router.navigate(['auth/login']);
    // return false;
    return this.userService.isLogin$.pipe(
      map((isLogin: boolean) => {
        if (isLogin) { return true; }
        return this.router.parseUrl('auth/login');
      })
    );
  }
}
```

الطريقة الأولى

الطريقة الثانية

الآن نذهب إلى ملف app-routing.module.ts ونتأكد أننا وضعنا هذا CanActivate في route ذو الاسم profile والroute ذو الاسم users، كالتالي:

ملف app-routing.module.ts

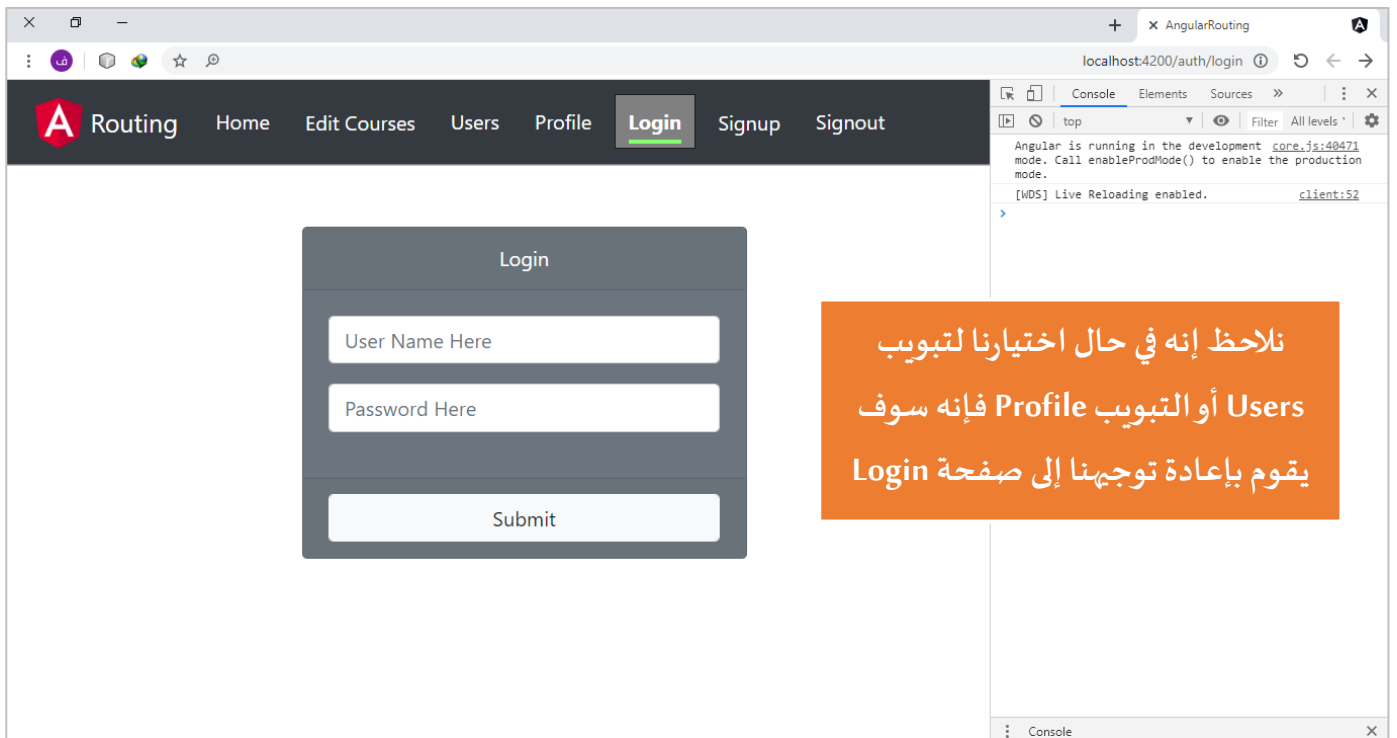
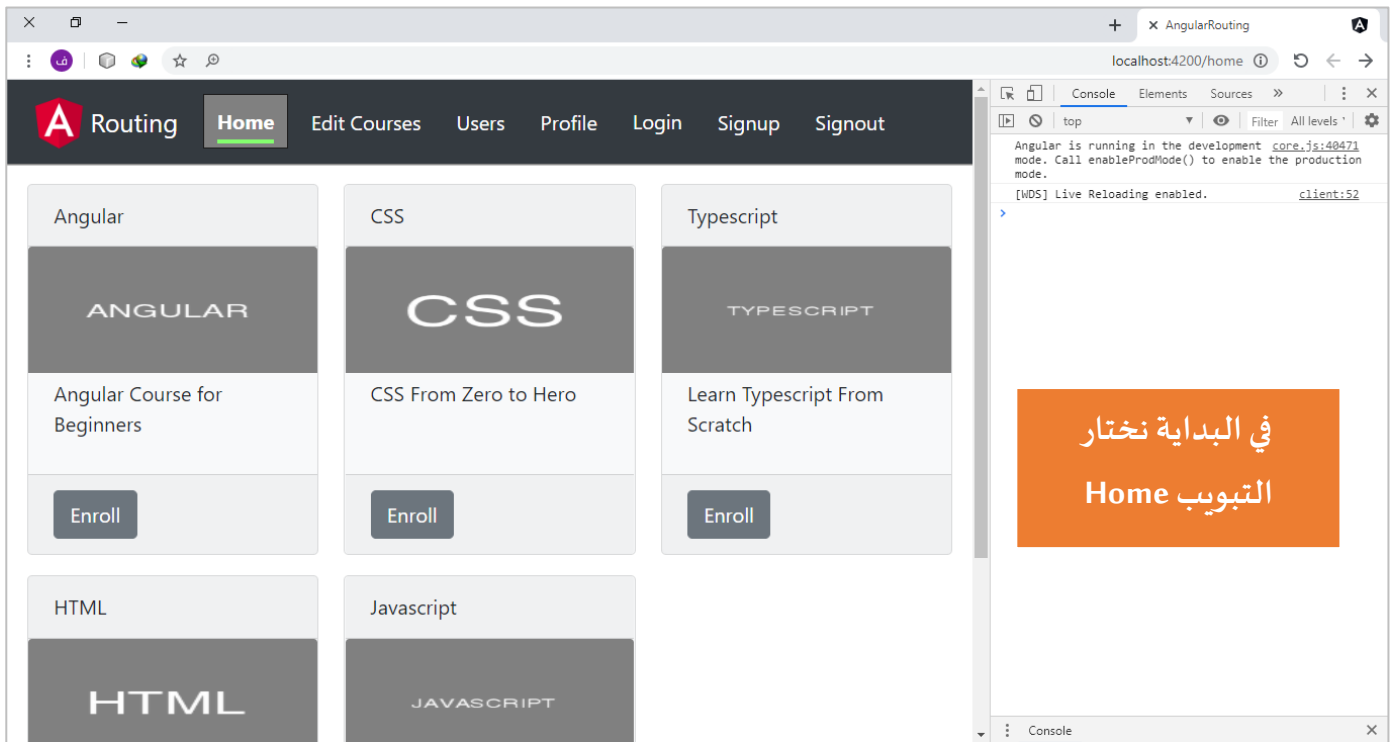
```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from '../home/home.component';
import { UsersComponent } from '../users/users.component';
import { UsersListComponent } from '../users/users-list/users-list.component';
import { UserDetailsComponent } from '../users/user-details/user-details.component';
import { AuthenticationComponent } from '../authentication/authentication.component';
import { LoginComponent } from '../authentication/login/login.component';
import { SignupComponent } from '../authentication/signup/signup.component';
import { NotFoundPageComponent } from '../not-found-page/not-found-page.component';
import { SingoutComponent } from '../authentication/singout/singout.component';
import { ProfileComponent } from '../profile/profile.component';
import { EditeCoursesComponent } from '../home/edite-courses/edite-courses.component';
import { CanActivateGuard } from '../guards/can-activate.guard';

const routes: Routes = [
  {
    path: 'home', children: [
      { path: '', component: HomeComponent },
      { path: 'edite-courses', component: EditeCoursesComponent }
    ]
  },
  { path: 'profile/:id', canActivate: [CanActivateGuard], component: ProfileComponent },
  {
    path: 'users',
    canActivate: [CanActivateGuard],
    children: [
      { path: '', component: UsersComponent },
      { path: 'user-list', component: UsersListComponent },
      { path: 'user-details/:id', component: UserDetailsComponent },
    ]
  },
  {
    path: 'auth', children: [
      { path: '', component: AuthenticationComponent },
      { path: 'login', component: LoginComponent },
      { path: 'signup', component: SignupComponent },
      { path: 'signout', component: SingoutComponent }
    ]
  },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: '**', component: NotFoundPageComponent }
];

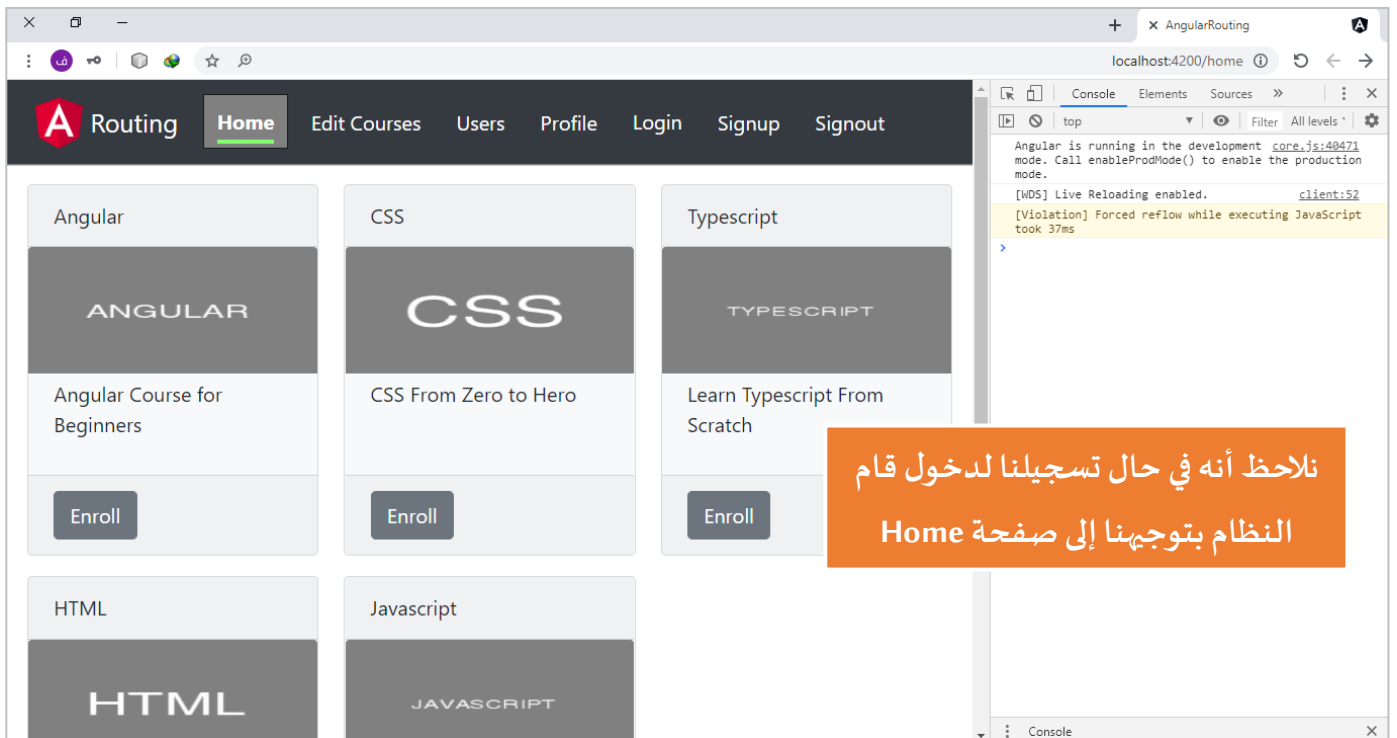
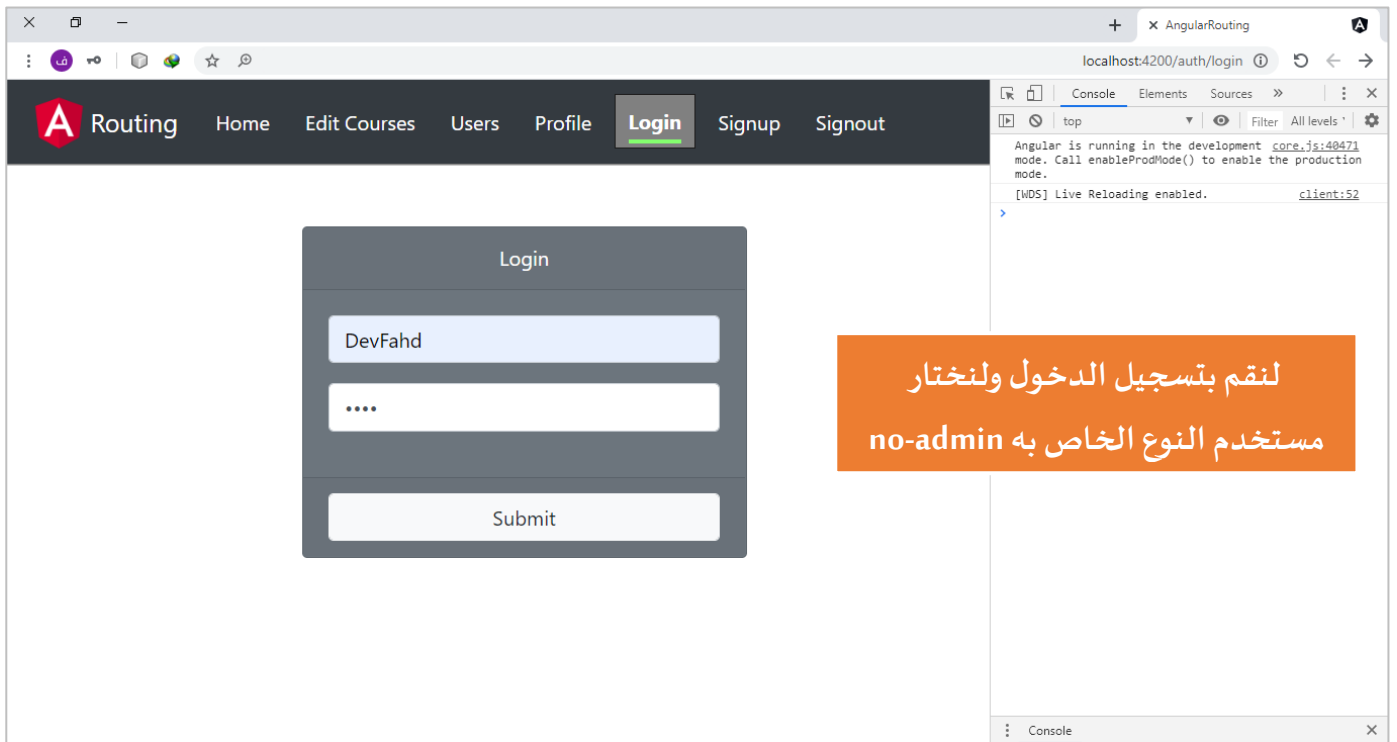
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

والآن لنقوم بتجربة ما قمنا به من تعديلات، لذلك فلنحفظ التعديلات ونذهب إلى المتصفح ونرى النتيجة، كالتالي:





الان لنقوم بتسجيل الدخول ونرى النتيجة، كالتالي:



الآن لنذهب إلى التبويب Users وايضاً التبويب Profile، كالتالي:

Routing Home Edit Courses Users **Profile** Login Signup Signout

**Fahd**  
 User ID: 4  
 UserName: DevFahd  
 User Password: 3333  
 User City: Jeddah  
 User Type: no-admin

نلاحظ أنه سمح بالتوجيه إلى صفحة profile لأن الدالة إعادة القيمة true بناءً على قيمة المتغير isLogin\$

Angular is running in the development mode. Call enableProdMode() to enable the production mode.  
 [WDS] Live Reloading enabled.  
 [Violation] Forced reflow while executing JavaScript took 37ms

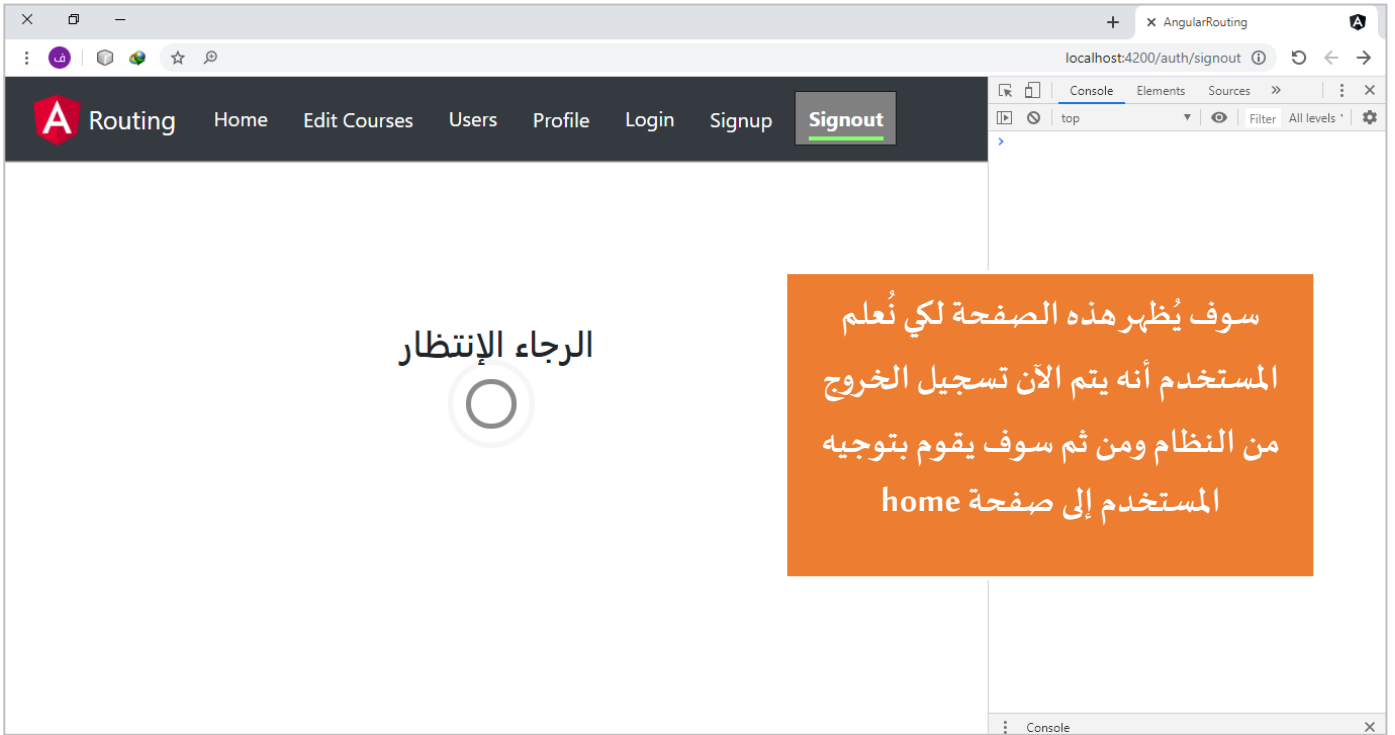
Routing Home Edit Courses **Users** Profile Login Signup Signout

Search Users Here

	Faisal	View	Active
	Saad	View	Active
	Bader	View	Active
	Fahd	View	Active

وبنفس الطريقة في صفحة Users

الآن لنضغط على تبويب Signout، ونرى النتيجة:



لو لا حظنا أن هنالك خطأ، وهو أن أي مستخدم يقوم بتسجيل الدخول سوف يملك الصلاحية للدخول إلى صفحة Users، مع العلم أن هذه الصفحة مخصصة فقط للمستخدم الذي نوعه يكون admin، ويمكن حل هذه المشكلة بطرق متعددة، منها وهي التي سوف اعتمدها هنا، عن طريق كلاس آخر من النوع CanActivate ولك الحرية أن يكون في ملف منفصل أو بنفس الملف can-activate.guard.ts، وانا هنا سوف اضعه بنفس الملف، وسوف أُسمي هذا الكلاس باسم CanActivateAdminGuard، كالتالي:

```
can-activate.guard.ts ملف
import {Injectable} from '@angular/core';
import {
```

```

    ActivatedRoute,
    ActivatedRouteSnapshot,
    CanActivate,
    Router,
    RouterStateSnapshot,
    UrlTree
  } from '@angular/router';
  import {Observable} from 'rxjs';
  import {UserService} from '../users-data/users.service';
  import {map} from 'rxjs/operators';

  @Injectable({
    providedIn: 'root'
  })

  export class CanActivateGuard implements CanActivate {
    constructor(
      private router: Router,
      private userService: UserService,
    ) { }

    canActivate(
      next: ActivatedRouteSnapshot,
      state: RouterStateSnapshot):
      Observable<boolean |
        UrlTree> |
      Promise<boolean |
        UrlTree> |
      boolean |
      UrlTree {
      // const isLogin = this.userService.isLogin$.value;
      // if (isLogin) { return true; }
      // this.router.navigate(['auth/login']);
      // return false;
      return this.userService.isLogin$.pipe(
        map((isLogin: boolean) => {
          if (isLogin) { return true; }
          return this.router.parseUrl('auth/login');
        })
      );
    }
  }

```

```

@Injectable({
  providedIn: 'root'
})

```

مهم جداً أن نعمل injectable لهذا  
الكلاس ايضاً

```

export class CanActivateAdminGuard implements CanActivate {
  constructor(private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot):
    Observable<boolean |
      UrlTree> |
    Promise<boolean |
      UrlTree> |
    boolean |
    UrlTree {
  }
}

```

ومهمة هذا الكلاس هو التأكد من أن المستخدم هو admin او لا، وايضاً تتم بعدة طرق منها أننا نستفيد من ميزة ارسال البيانات عن طريق route وباستخدام الخاصية data، والتي سوف اتطرق إليها في النقطة التالية.

### 3.2.3. إرسال البيانات عن طريق الخاصية data:

وهذا الجزء يندرج تحت طرق تبادل البيانات بين components المختلفة باستخدام Angular Routing او بصيغة أخرى طرق تواصل او الاتصال بين Components، ولكنها هنا متعلقة بمفاهيم Route Guard لذلك وضعتها في هذا الفصل.

ونستطيع إرسال البيانات وذلك بالذهاب إلى الملف app-routing.module.ts، ونقوم بإجراء تعديلين الأول نضيف الكلاس CanActivateAdminGuard، إلى route ذو الاسم usres، والتعديل الثاني نضيف الخاصية data ونمرر لها كائن وهذا الكائن يحتوي على مفتاح key سوف اسميه roles (لك حرية اختيار الاسم الذي تُريده)، وسوف اسند له قيمة وهذه القيمة نصية ولتكن 'admin' (لك حرية اختيار القيمة التي تُريدها ونوع البيانات الذي تُريده سواء نصي أو رقم او boolean) وانا هنا اخترت ان يكون نصي، مع العلم انه يمكن اسناد اكثر من قيمة على شكل مصفوفة نصية او مصفوفة رقمية او مصفوفة منطقية boolean...الخ، بحيث يكون التعديل بالشكل التالي:

ملف app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { UsersListComponent } from './users/users-list/users-list.component';
import { UserDetailsComponent } from './users/user-details/user-details.component';
import { AuthenticationComponent } from './authentication/authentication.component';
import { LoginComponent } from './authentication/login/login.component';
import { SignupComponent } from './authentication/signup/signup.component';
import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
import { SingoutComponent } from './authentication/singout/singout.component';
import { ProfileComponent } from './profile/profile.component';
import { EditeCoursesComponent } from './home/edite-courses/edite-courses.component';
import { CanActivateGuard, CanActivateAdminGuard } from './guards/can-activate.guard';

const routes: Routes = [
  {
    path: 'home', children: [
      { path: '', component: HomeComponent },
      { path: 'edite-courses', component: EditeCoursesComponent }
    ]
  },
  { path: 'profile/:id', canActivate: [CanActivateGuard], component: ProfileComponent },
  {
    path: 'users',
    canActivate: [CanActivateGuard, CanActivateAdminGuard],
    data: {
      roles: 'admin'
    },
    children: [
      { path: '', component: UsersComponent },
      { path: 'user-list', component: UsersListComponent },
      { path: 'user-details/:id', component: UserDetailsComponent },
    ]
  },
]
```

```

    path: 'auth', children: [
      { path: '', component: AuthenticationComponent },
      { path: 'login', component: LoginComponent },
      { path: 'signup', component: SignupComponent },
      { path: 'signout', component: SingoutComponent }
    ]
  },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: '**', component: NotFoundPageComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

ملاحظة الخاصية data تعمل مع route guards بحيث نستطيع ارسال بيانات واستقبالها في كلاس route guards، كما سوف نعمل بعد قليل.

الآن لنرجع إلى ملف can-activate.guard.ts، ونقوم باستقبال البيانات القادمة عن طريق الخاصية data، ومن ثم نقوم بمقارنة هذه البيانات بنوع المستخدم الذي قمنا بتخزينه في Local Storage (لو رجعنا إلى ملف users.service.ts وذهبنا بالتحديد إلى الدالة login()) لوجدنا اننا قمنا بتسجيل نوع المستخدم سواء كان admin او no-admin في local storage في كل مرة يقوم بها المستخدم بعملية تسجيل الدخول)، لذلك إذا كانت متشابهتان أي بمعنى جميعهما admin فسوف نُرجع القيمة true اما إذا كانتا مختلفتان أي بمعنى أن القيمة الموجودة في الخاصية data هي admin (وبالطبع هي admin لأن قيمتها ثابتة) والقيمة الموجودة في Local Storage هي no-admin فسوف نُرجع الرابط من النوع UrlTree وسوف نقوم بتوجيه المستخدم إلى صفحة home لأنه من غير المنطقي نقوم بإعادة توجيهه إلى صفحة login وهو قام بتسجيل الدخول من قبل، كالتالي:

ملف can-activate.guard.ts

```

import {Injectable} from '@angular/core';
import {
  ActivatedRoute,
  ActivatedRouteSnapshot,
  CanActivate,
  Router,
  RouterStateSnapshot,
  UrlTree
} from '@angular/router';
import {Observable} from 'rxjs';
import {UserService} from '../users-data/users.service';
import {map} from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})

export class CanActivateGuard implements CanActivate {
  constructor(
    private router: Router,
    private activatedRoute: ActivatedRoute,
    private userService: UserService,

```

```

) { }

canActivate(
  next: ActivatedRouteSnapshot,
  state: RouterStateSnapshot):
  Observable<boolean |
    UrlTree> |
  Promise<boolean |
    UrlTree> |
  boolean |
  UrlTree {
    return this.userService.isLogin$.pipe(
      map((isLogin: boolean) => {
        if (isLogin) { return true; }
        return this.router.parseUrl('auth/login');
      })
    );
  }
}

@Injectable({
  providedIn: 'root'
})

export class CanActivateAdminGuard implements CanActivate {
  constructor(private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot):
    Observable<boolean |
      UrlTree> |
    Promise<boolean |
      UrlTree> |
    boolean |
    UrlTree {
    const roles: string = next.data.roles;
    const admin = localStorage.getItem('type');
    if (roles === admin) {
      return true;
    }
    return this.router.parseUrl('home');
  }
}

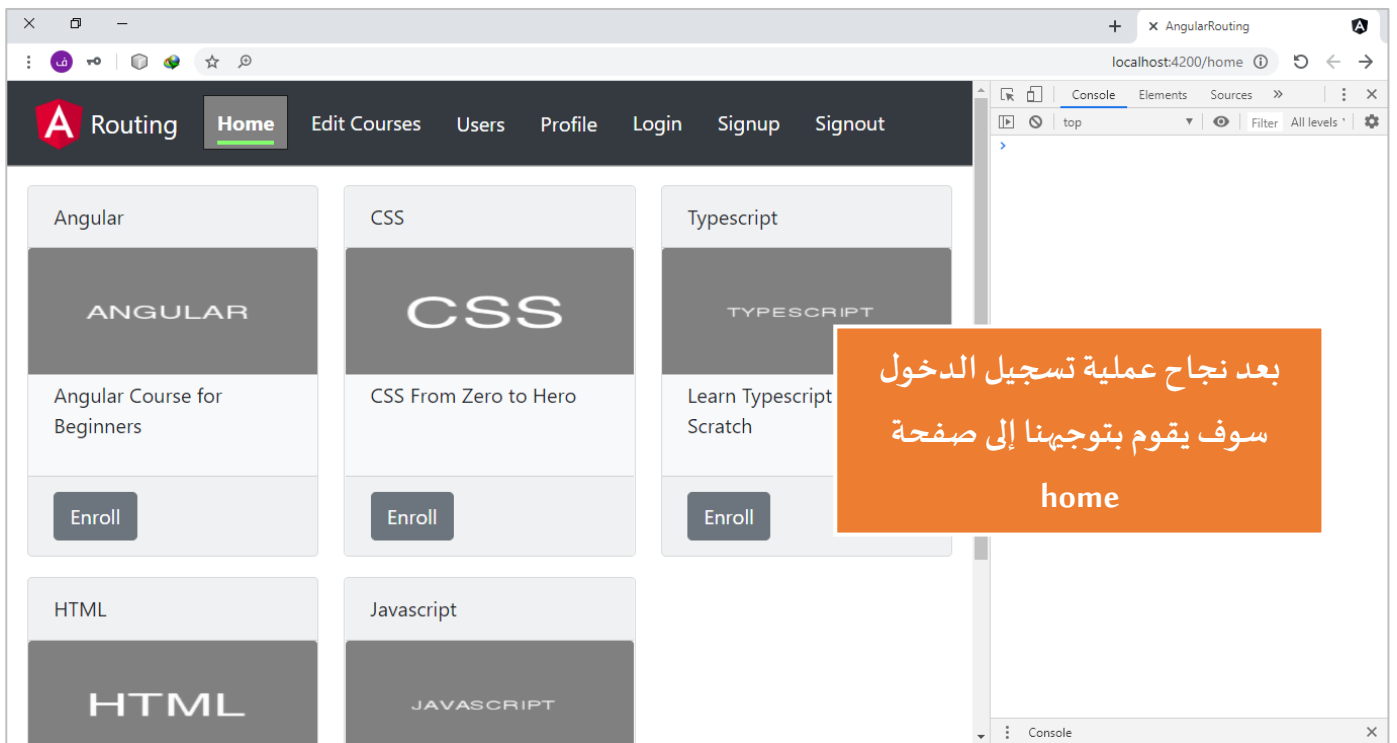
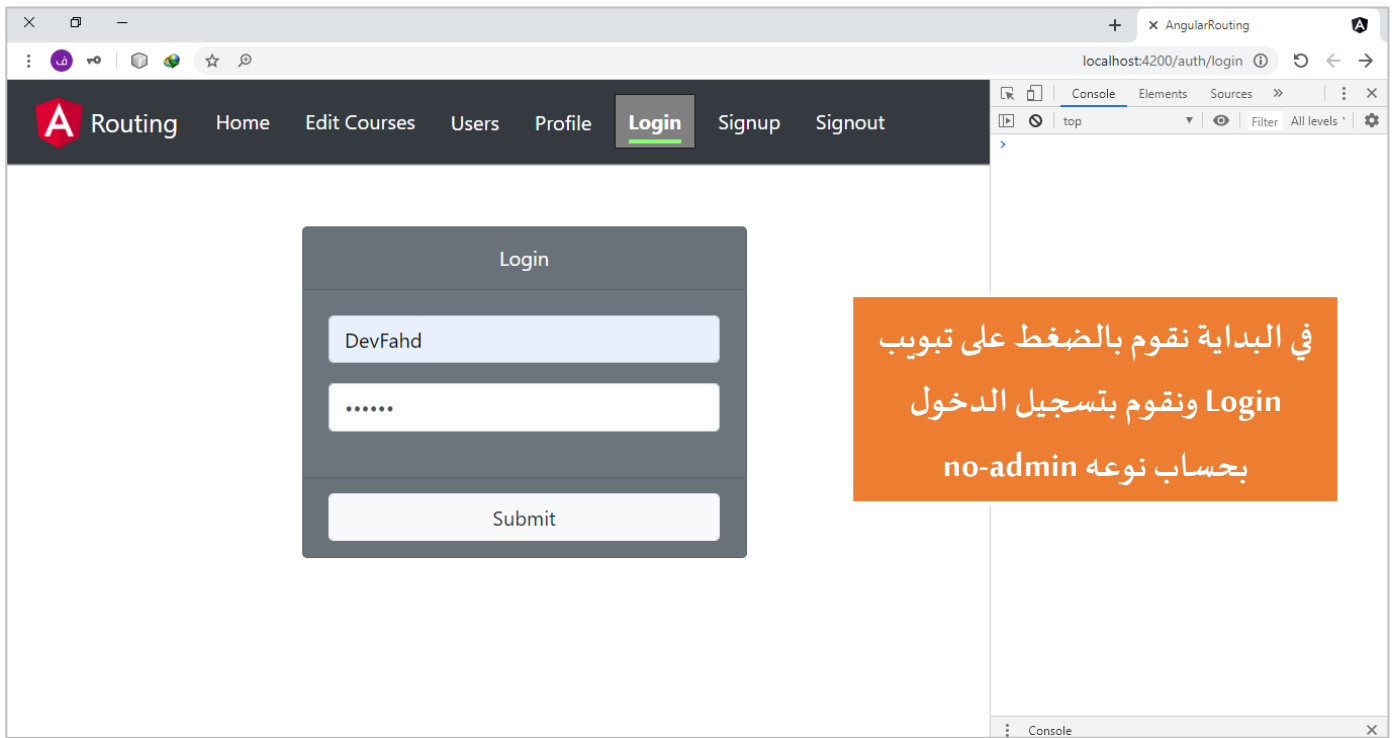
```

يتم استقبال البيانات 'admin' عند طريق الباراميتر next وعن طريق الخاصية data نستطيع الوصول إلى المفتاح roles التي قمنا بإنشائها سابقاً ومن ثم تخزين قيمتها في ثابت بنفس الاسم.

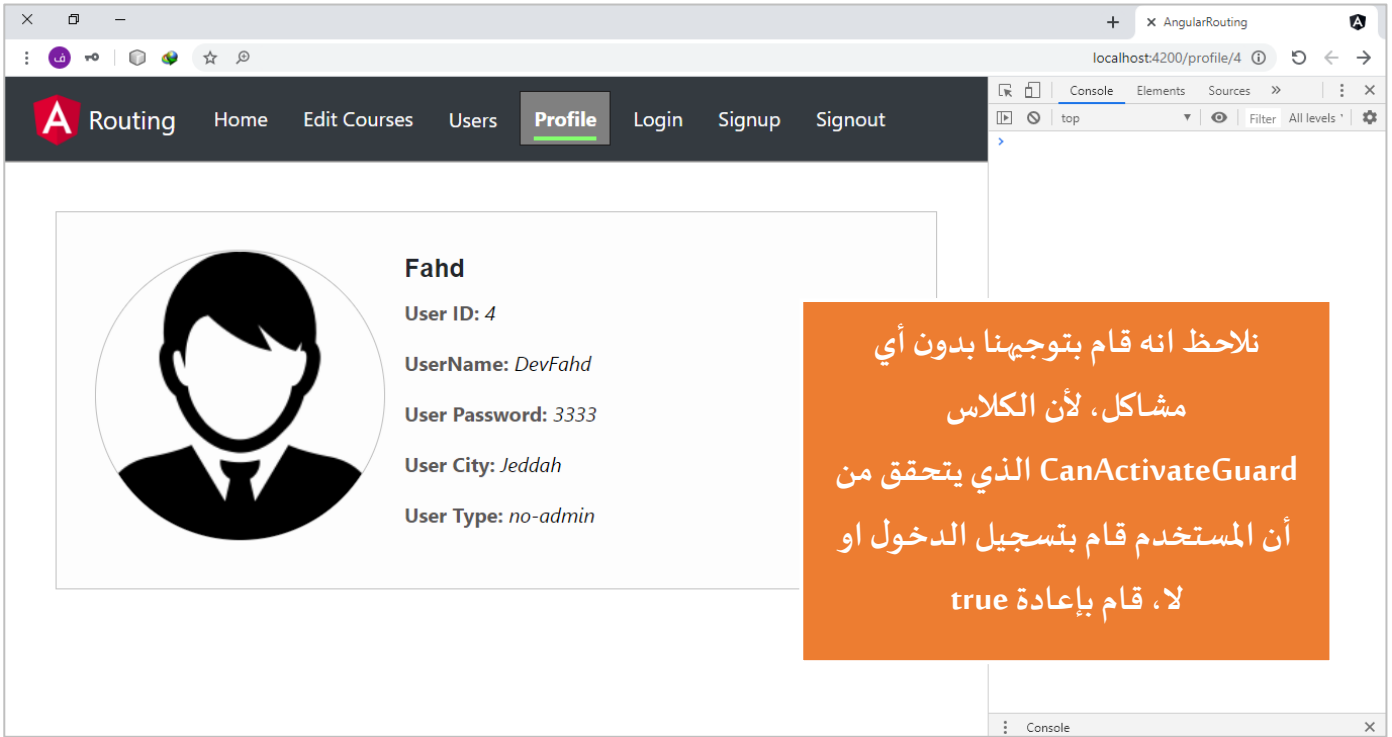
ونفس الطريقة مع القيمة الموجودة في Local Storage وبعدها قمنا بعمل مقارنة إذا كانت كلا القيمتين متساويتين فمعنى هذا ان المستخدم هو admin وفي هذه الحالة نُعيد true اما إذا كانت مختلفتين فمعنى ان التحقق فشل ونقوم بتوجيه المستخدم إلى صفحة home.

الآن لنقوم بحفظ التعديلات، ومن ثم الذهاب إلى المتصفح ونرى النتيجة، كالتالي:





الآن لنقوم بالضغط على صفحة profile، فسوف نلاحظ انه سوف يقوم بتوجيهنا بدون أي مشاكل ويعرض لنا هذه الصفحة، لأن المستخدم قام بتسجيل الدخول، كالتالي:



الآن لنقوم بالضغط على تبويب Users، ونرى النتيجة:



وبذلك نكون أننا في النوع الأول من Route Guards، وبقي علينا أن نوضح آخر جزء قبل أن ننتقل إلى النوع الثاني بإذن الله، وهذه الجزئية تتكلم كيف يمكن أن نُعيد المستخدم إلى الصفحة التي كان موجود فيها قبل توجيهه إلى صفحة تسجيل الدخول، وهذا ما سوف أتحدث عنه في النقطة التالية.

### 4.2.3. إعادة توجيه المستخدم إلى الصفحة التي كان فيها بعد تسجيل الدخول:

بمعنى إذا قام المستخدم بالضغط على صفحة Users وهو لم يقم بعملية تسجيل الدخول فإن النظام سوف يقوم بإعادة توجيهه إلى صفحة Login، والمطلوب منا بعد ما يقوم بعملية تسجيل الدخول أن نقوم بإرجاعه إلى الصفحة التي كان متواجد فيها وهي Users، وب نفس الطريقة مع Profile، ويمكن عمل هذا الامر بعدة طرق منها أن نقوم بإرسال url الخاص بالصفحة التي كان بها المستخدم على شكل Query Parameters ونقوم بقراءة هذا query في صفحة Login وبناءً عليه نقوم بإعادة توجيه المستخدم إلى الصفحة التي كان فيها من قبل، لذلك لنذهب إلى ملف can-activate.guard.ts وبالتحديد نذهب إلى الكلاس CanActivateGuard وبالتحديد إلى السطر الذي نقوم بإعادة توجيه المستخدم، ونستبدل الدالة parseUrl بالدالة createUrlTree والسبب ان الأولى كما ذكرنا سابقاً تقبل فقط باراميتر واحد وهو الرابط اما الدالة الثانية فتقبل الرابط على شكل مصفوفة بالإضافة إلى خيارات أخرى من ضمنها Query Parameters، كالتالي:

ملف can-activate.guard.ts

```
import {Injectable} from '@angular/core';
import {
  ActivatedRoute,
  ActivatedRouteSnapshot,
  CanActivate,
  Router,
  RouterStateSnapshot,
  UrlTree
} from '@angular/router';
import {Observable} from 'rxjs';
import {UserService} from '../users-data/users.service';
import {map} from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class CanActivateGuard implements CanActivate {
  constructor(
    private router: Router,
    private activatedRoute: ActivatedRoute,
    private userService: UserService,
  ) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot):
    Observable<boolean |
      UrlTree> |
    Promise<boolean |
      UrlTree> |
    boolean |
    UrlTree {
    return this.userService.isLogin$.pipe(
      map((isLogin: boolean) => {
        if (isLogin) { return true; }

        return this.router.createUrlTree(['auth/login'],
          {
            queryParams: {
              redirectUrl: state.url
            }
          }
        );
      })
    );
  }
}
```



```

    );
  })
);
}
}

@Injectable({
  providedIn: 'root'
})

export class CanActivateAdminGuard implements CanActivate {
  constructor(private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot):
    Observable<boolean |
      UrlTree> |
    Promise<boolean |
      UrlTree> |
    boolean |
    UrlTree {
    const roles: string = next.data.roles;
    const admin = localStorage.getItem('type');
    if (roles === admin) {
      return true;
    }
    return this.router.parseUrl('home');
  }
}

```

نلاحظ اننا مررنا الـ Query Parameter على شكل كائن واضفنا له مفتاح اسميناه redirectUrl (لك حرية اختيار الاسم الذي تُريده) واسندنا له القيمة وهي عبارة عن url الموجود فيه المستخدم قبل انتقال إلى صفحة login، والذي حصلنا عليه عن طريق الباراميتر state.

الآن لنذهب إلى ملف login.component.ts لكي نستقبل هذا الباراميتر ومن ثم نعيد المستخدم إذا ضغط على زر submit، كالتالي:

ملف login.component.ts

```

import { Component, OnInit } from '@angular/core';
import { UsersService } from 'src/app/users-data/users.service';
import { NgForm } from '@angular/forms';
import { ActivatedRoute, Router } from '@angular/router';
import { StorageMap } from '@ngx-pwa/local-storage';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {
  private path: string;

  constructor(
    private userService: UsersService,
    private activatedRoute: ActivatedRoute,
    private router: Router,
    private storageMap: StorageMap
  ) { }

```

قمنا بتعريف متغير لكي نُخزن الباراميتر القادم.

```

ngOnInit() {
  if (this.activatedRoute.snapshot.queryParams.redirectUrl) {
    this.path = this.activatedRoute.snapshot.queryParams.redirectUrl;
  } else {
    this.path = '';
  }
}

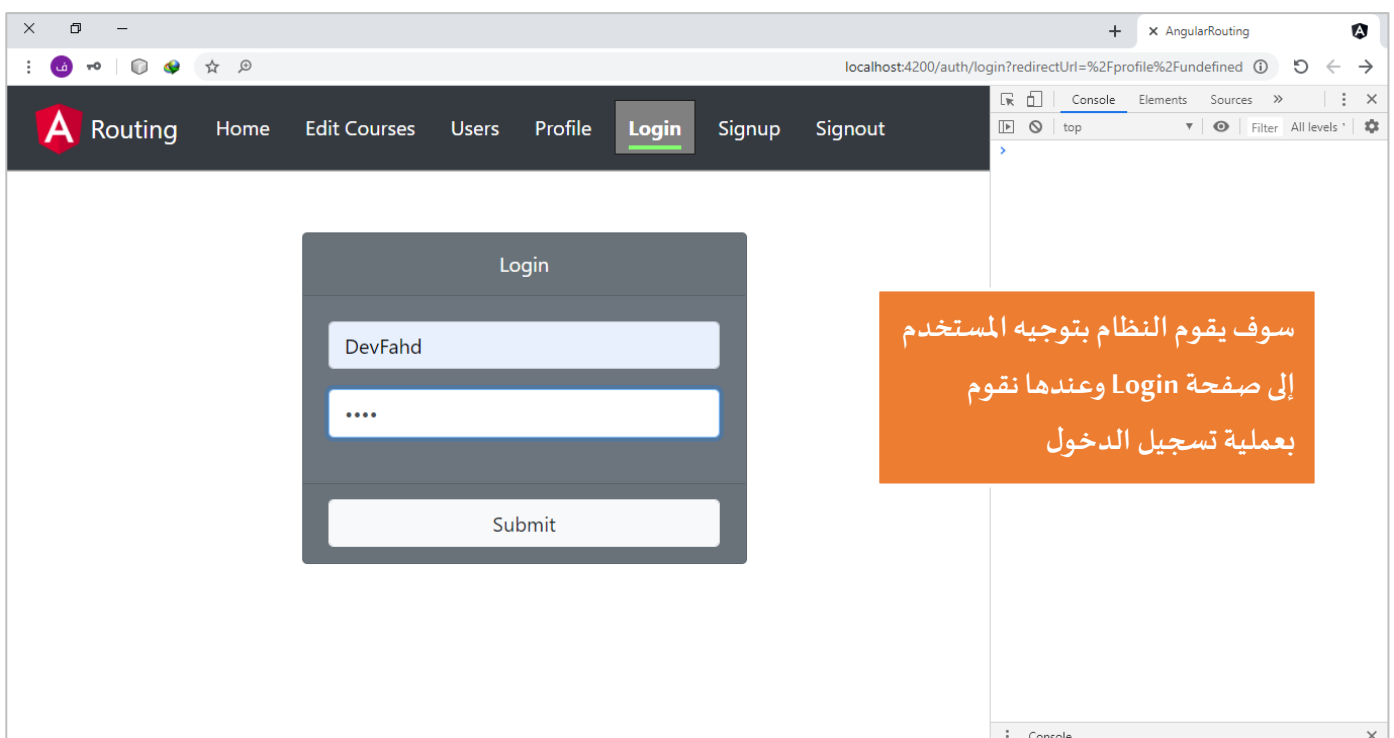
onSubmit(form: NgForm) {
  const value = form.control.get('userName').value;
  this.userService.logIn(value);
  this.storageMap.get('id').subscribe((id) => {
    if (this.path === '') {
      this.router.navigate(['/home']);
    } else if (this.path === '/users') {
      this.router.navigate(['`/${this.path}`']);
    } else {
      const path = this.path.split('/');
      this.router.navigate(['`/${path[1]}/${id}`']);
    }
  });
}
}

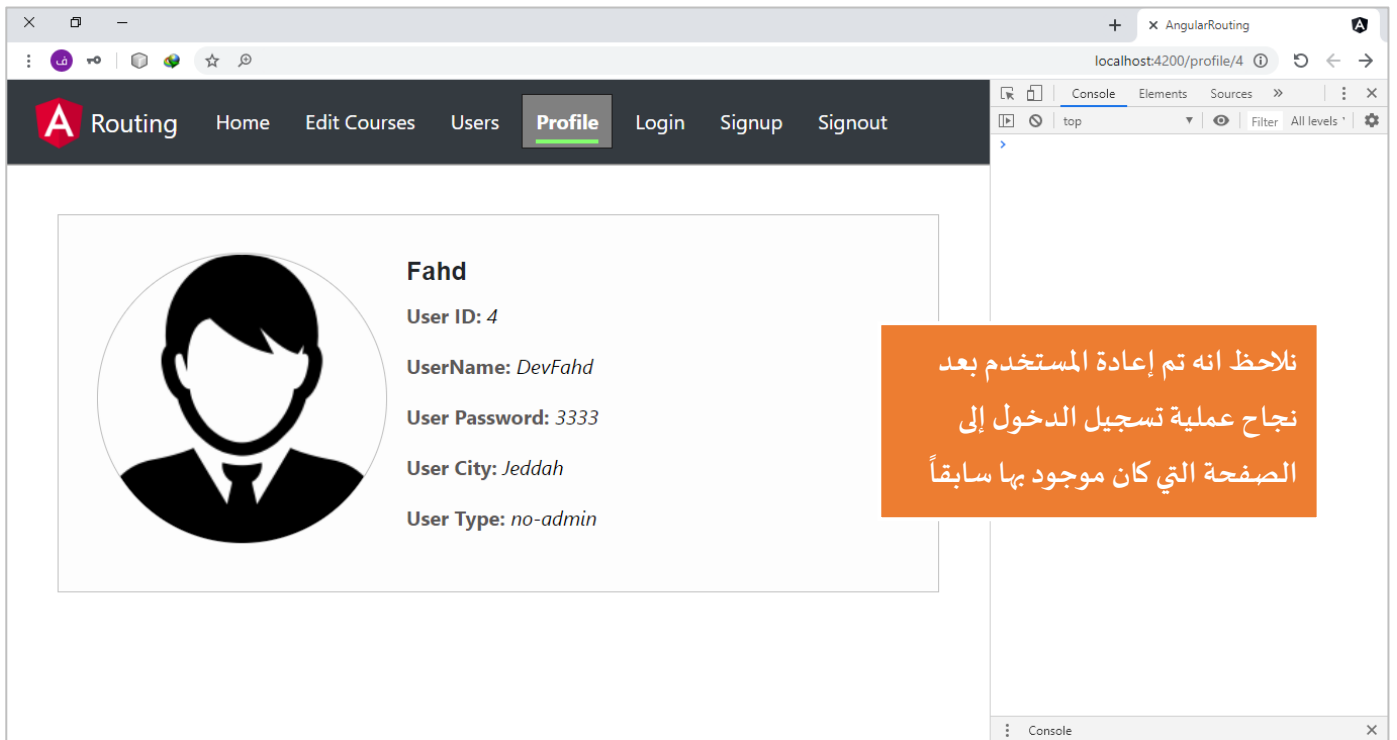
```

هنا قمنا بالتأكد هل هنالك باراميتر قادم ام لا (لأن هنالك حالة ان المستخدم ضغط بشكل مباشر على زر login ولم يتم توجيهه)، فإذا كان هنالك باراميتر فنقوم بتخزين قيمته في المتغير path وفي حال لم يكن هنالك قيمة فنجعل قيمة المتغير فارغة.

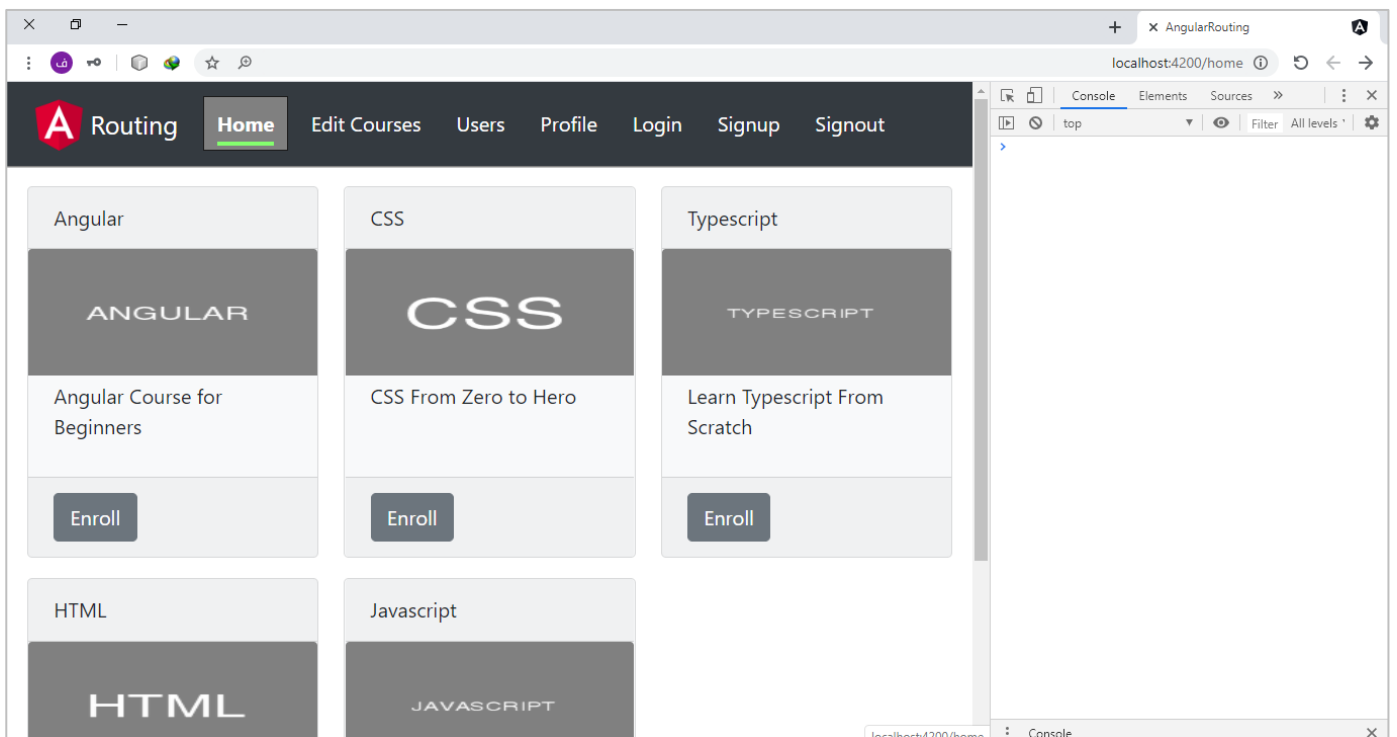
الآن لنقوم بحفظ التعديلات ونذهب إلى المتصفح ونرى النتيجة، كالتالي:

في البداية لنختار التبويب Profile:





ولو قمنا بعمل تسجيل الخروج ومن ثم قمنا بعملية تسجيل الدخول بشكل مباشر فأنا النظام سوف يقوم بتوجيهه إلى الصفحة الرئيسية، كالتالي:

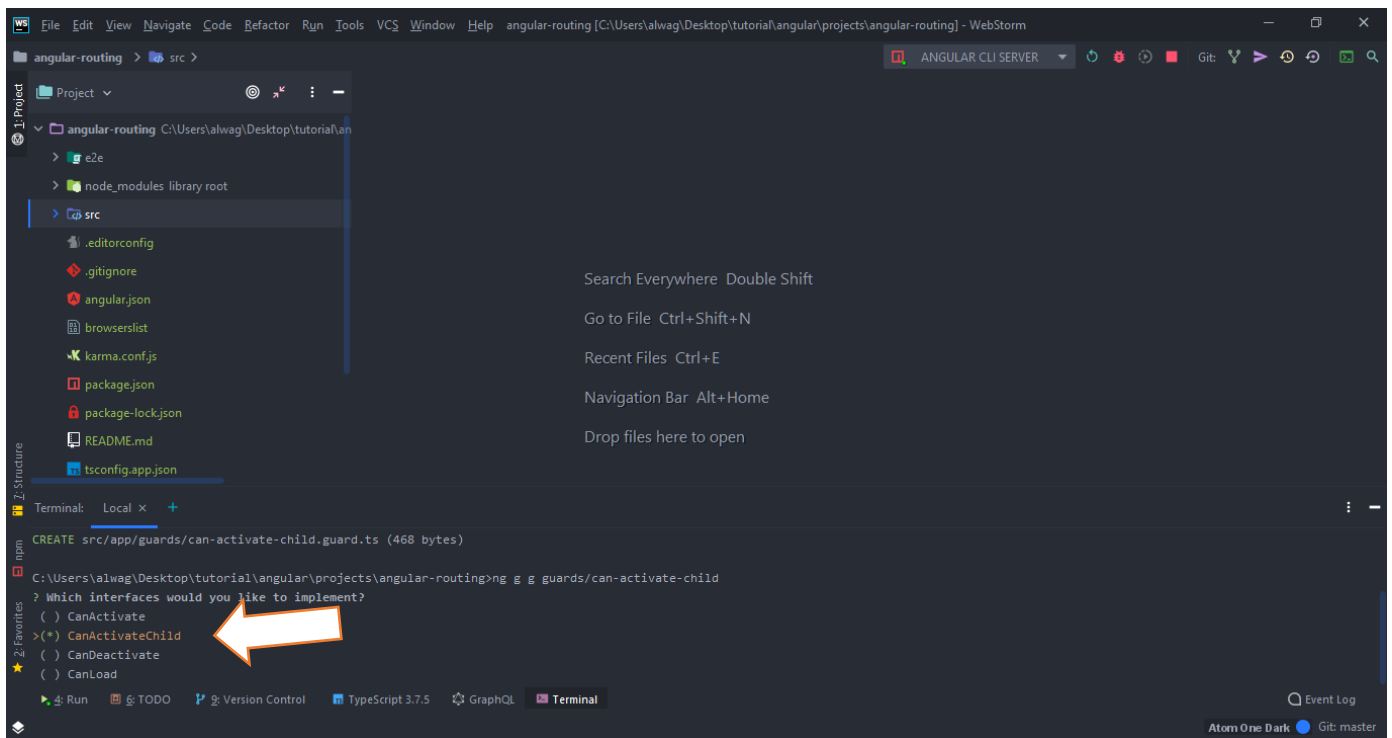


وبذلك نكون أنهينا النوع الأول من أنواع Route Guards، وسوف ننتقل في النقطة التالية إلى النوع الثاني وهو .CanActivateChild

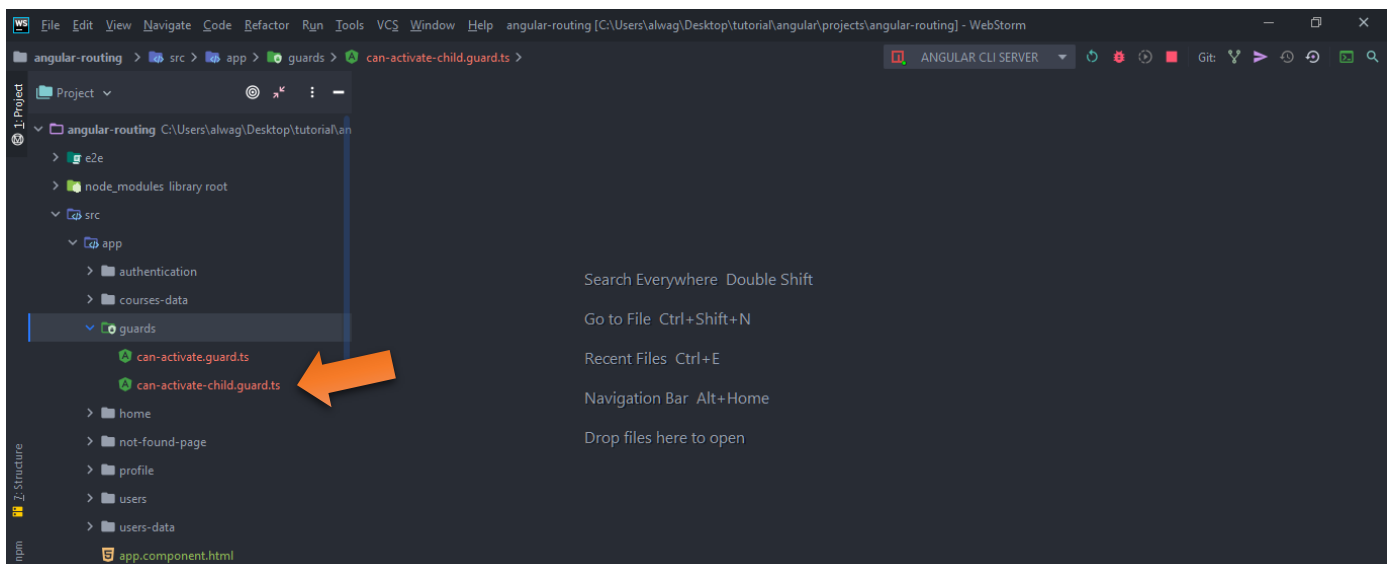
### :CanActivateChild Guard 3.3

وهو مشابه للنوع الأول في جميع ميزاته والفرق الوحيد انه يطبق الحماية على مستوى الأبناء فقط، بدون الأب، وفي مشروعنا لدينا route ذو الاسم home ولديه ابن وهو route ذو الاسم edite-courses، ونستطيع أن نستفيد من هذا النوع من Route Guards بأن نحمي الابن فقط بحيث لا يستطيع الوصول إليه إلا المستخدمين الذي عملوا تسجيل الدخول وبنفس الوقت نوعهم admin، اما route الاب فلا نريد تطبيق عليه أي حماية ويستطيع أي زائر لتطبيق تصفح الكورسات ولكن لا يستطيع التعديل عليها.

وتتم إضافة الملف بنفس طريقة النوع الأول ولكن هنا نختار الخيار الثاني CanActivateChild، وليكن اسم الملف can-activate-child،



ثم نضغط زر Enter ليقوم angular بإنشاء الملف، كالتالي:



اما محتويات الملف فهي مشابهة لنوع الأول CanActivate والفرق الوحيد اننا هنا نعمل implements لinterface ذو الاسم CanActivateChild، كالتالي:

ملف can-activate-child.ts

```
import { Injectable } from '@angular/core';
import {
  CanActivateChild,
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  UrlTree } from '@angular/router';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class CanActivateChild implements CanActivateChild {

  canActivateChild( next: ActivatedRouteSnapshot, state: RouterStateSnapshot):
    Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    return true;
  }
}
```

وسوف نقوم بنفس الخطوات التي عملناها سابقاً مع بعض الاختلافات البسيطة وهو انشاء كلاسين الأول لتأكد ان المستخدم قام بتسجيل الدخول والثاني لتأكد انه admin، كالتالي:

ملف can-activate-child.ts

```
import { Injectable } from '@angular/core';
import {
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  UrlTree,
  Router,
  CanActivateChild
} from '@angular/router';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';
import { UsersService } from '../users-data/users.service';

@Injectable({
  providedIn: 'root'
})
export class CanActivateChildGuard implements CanActivateChild {

  constructor(
    private router: Router,
    private userService: UsersService,
  ) { }

  canActivateChild( childRoute: ActivatedRouteSnapshot, state: RouterStateSnapshot):
    Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    return this.userService.isLogin$.pipe(
      map((isLogin: boolean) => {
        console.log('isLogin: ' + isLogin);
        if (isLogin) { return isLogin; }
        return this.router.createUrlTree(['auth/login'], { queryParams: { returnUrl:
state.url } });
      })
    );
  }
}
```



```

    })
  );
}

}

@Injectable({
  providedIn: 'root'
})

export class CanActivateAdminChildGuard implements CanActivateChild {

  constructor(
    private router: Router,
    private userService: UsersService,
  ) { }

  canActivateChild(childRoute: ActivatedRouteSnapshot, state: RouterStateSnapshot):
  Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    return this.userService.isAdmin$.pipe(
      map((isAdmin: boolean) => {
        console.log('isAdmin: ' + isAdmin);
        if (isAdmin) { return isAdmin; }
        return this.router.parseUrl('home');
      })
    );
  }
}

```

كما نلاحظ أن الكلاس الأول ذو الاسم CanActivateChildGuard يقوم بتأكد بأن المستخدم قام بتسجيل الدخول ام لا، والكلاس الثاني يتأكد من أن المستخدم هو admin.

الآن لنذهب إلى ملف app-routing.module.ts، ونضيف Route Guard إلى Route المستهدف، كالتالي:

ملف app-routing.module.ts

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { UsersListComponent } from './users/users-list/users-list.component';
import { UserDetailsComponent } from './users/user-details/user-details.component';
import { AuthenticationComponent } from './authentication/authentication.component';
import { LoginComponent } from './authentication/login/login.component';
import { SignupComponent } from './authentication/signup/signup.component';
import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
import { SingoutComponent } from './authentication/singout/singout.component';
import { ProfileComponent } from './profile/profile.component';
import { EditeCoursesComponent } from './home/edite-courses/edite-courses.component';
import { CanActivateGuard, CanActivateAdminGuard } from './guards/can-activate.guard';
import { CanActivateAdminChildGuard, CanActivateChildGuard } from './guards/can-activate-child.guard';

const routes: Routes = [
  {
    path: 'home', component: HomeComponent, canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
    children: [
      { path: 'edite-courses', component: EditeCoursesComponent }
    ]
  }
]

```

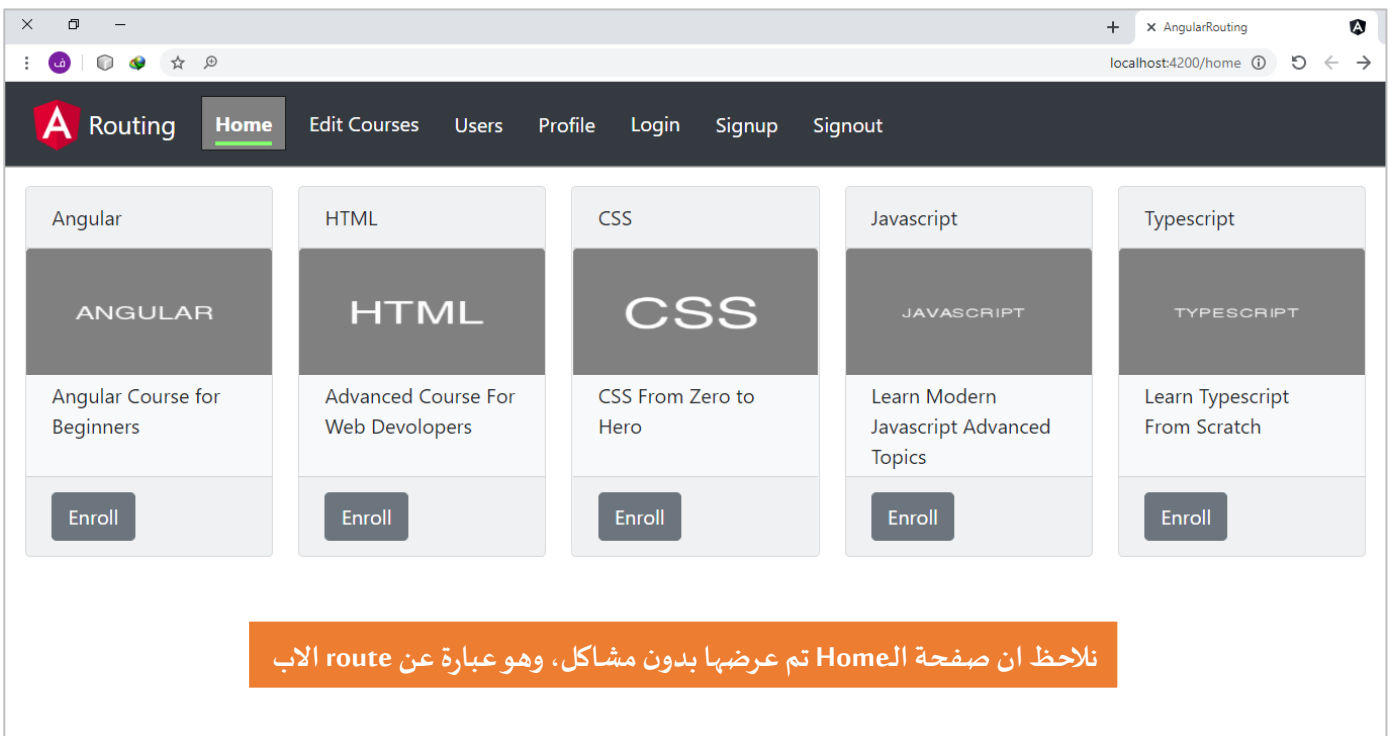
```

    ],
  },
  {
    path: 'profile/:id', canActivate: [CanActivateGuard], component: ProfileComponent },
  {
    path: 'users',
    canActivate: [CanActivateGuard, CanActivateAdminGuard],
    data: {
      roles: 'admin'
    },
    children: [
      { path: '', component: UsersComponent },
      { path: 'user-list', component: UsersListComponent },
      { path: 'user-details/:id', component: UserDetailsComponent },
    ]
  },
  {
    path: 'auth', children: [
      { path: '', component: AuthenticationComponent },
      { path: 'login', component: LoginComponent },
      { path: 'signup', component: SignupComponent },
      { path: 'signout', component: SingoutComponent }
    ]
  },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: '**', component: NotFoundPageComponent }
];

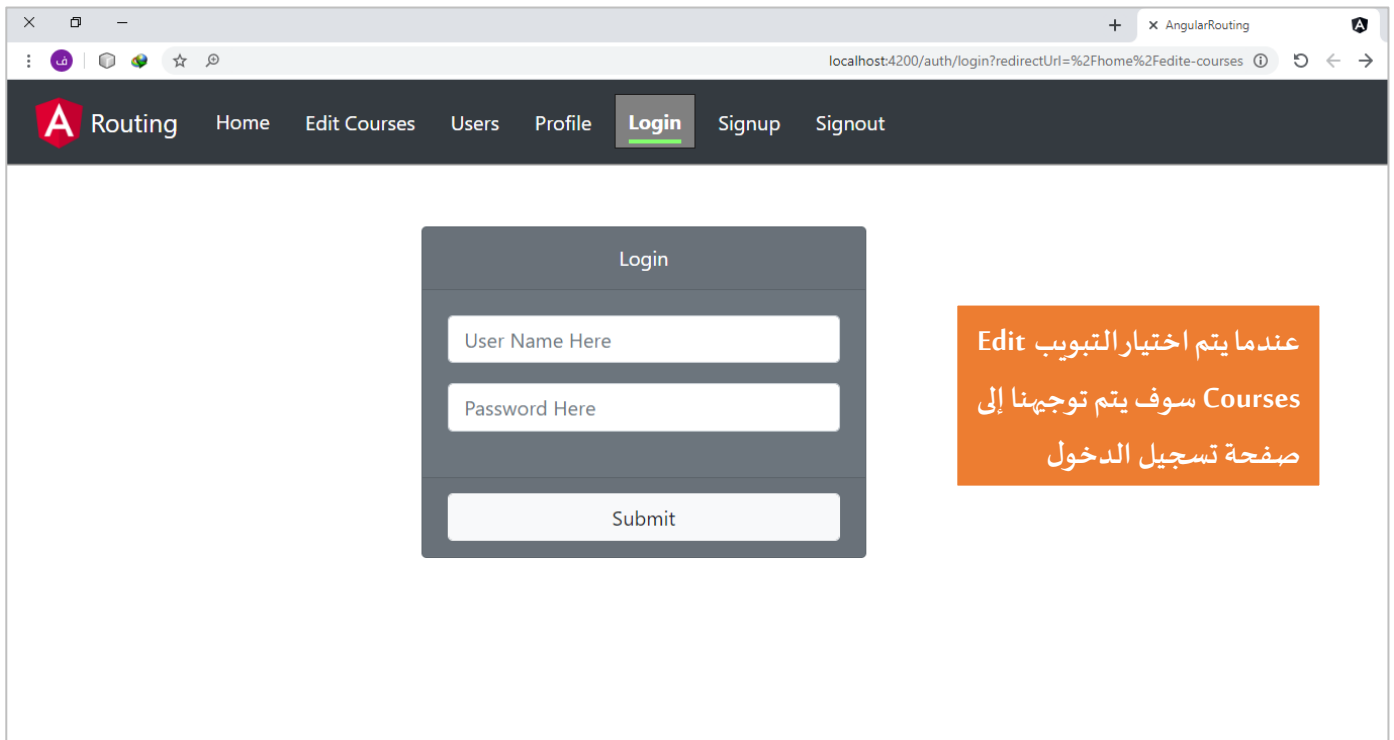
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

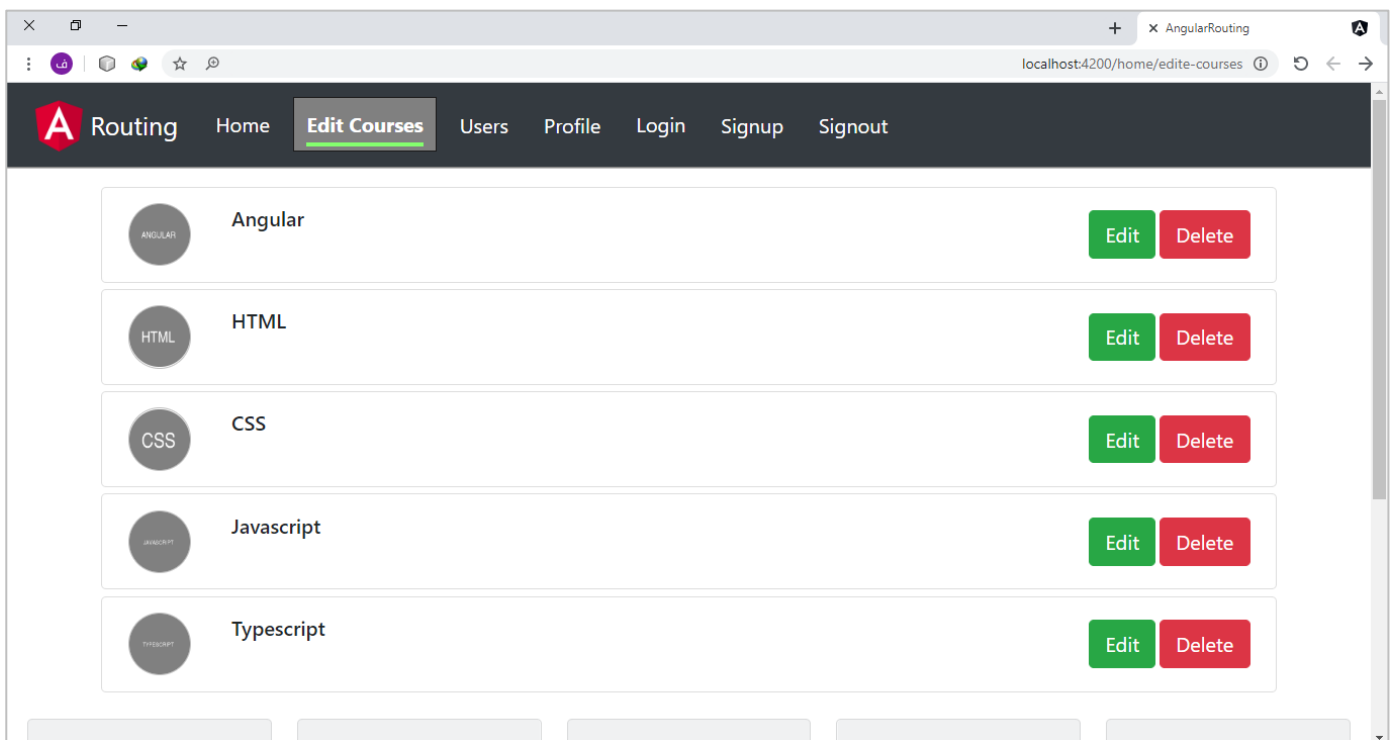
الآن لنحفظ التعديلات ونرى النتيجة في المتصفح:



الآن لنختار التبويب Edit Users، ولنرى النتيجة:



الآن لنقوم بتسجيل الدخول بحساب admin، ونرى النتيجة:



نلاحظ انه أعاد توجيهنا إلى صفحة Edit Users، وأصبح مسموح لنا بالدخول إلى هذه الصفحة.

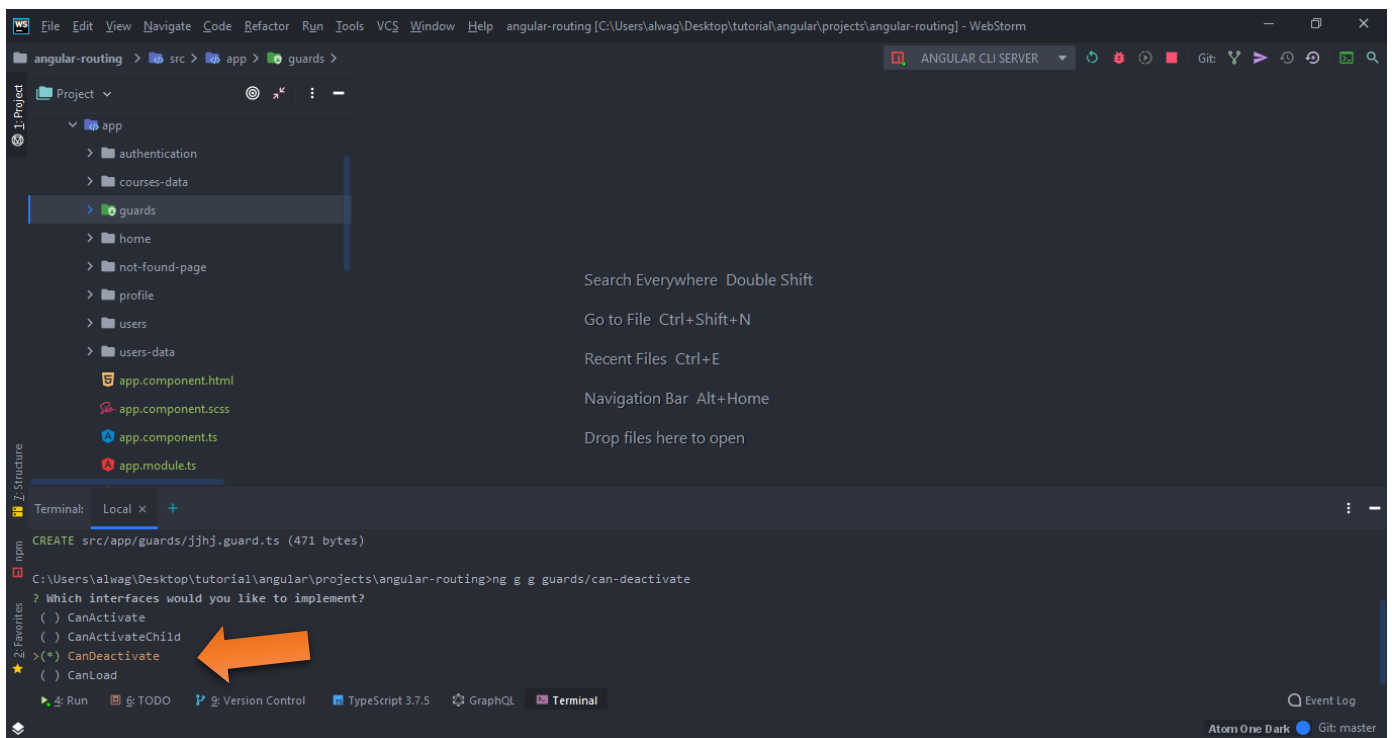
وبذلك نكون أنهينا هذا النوع، وسوف ننتقل الآن إلى النوع الثالث.

### 4.3. CanDeactivate guard

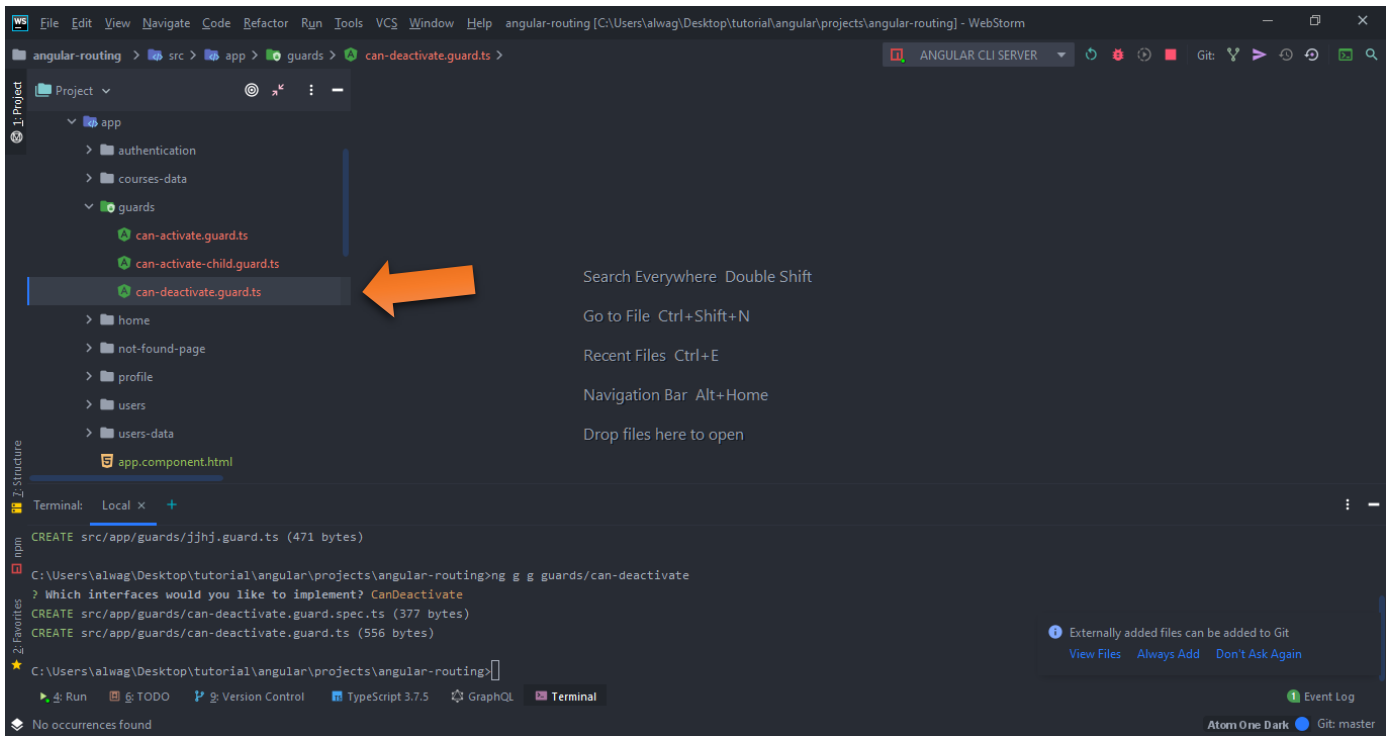
هذا النوع بعكس الأنواع الأخرى يُركز على routed الحالي وليس routed الذي تم التوجيه إليه، كما في الأنواع السابقة، فمثلاً في الأنواع السابقة لو كان المستخدم موجود في صفحة home واراد التوجه إلى صفحة users في هذه الحالة يبدأ عمل هذه الأنواع حيث تقوم بحماية routed الذي توجه إليه المستخدم والذي هو في حالتنا هذه صفحة users، اما هذا النوع فهو يعمل على نفس الصفحة الموجود فيها المستخدم قبل انتقاله وفي مثالنا هذا هو صفحة home.

ومن أشهر استخدامات هذا النوع هو التأكد من البيانات قبل مغادرة الصفحة، فمثلاً لو ان المستخدم في صفحة تسجيل مستخدم جديد singup وقام بتعبئة البيانات ولكنه قام بالتوجه إلى route آخر قبل ان يكمل البيانات او ان يقوم بحفظ هذه البيانات، ففي هذه الحال نظهر له رسالة مثلاً لنخبره انه لم يكمل البيانات او لم يتم بحفظ هذه البيانات، لذلك نستطيع أن نقول أن هذا النوع يتم تفعيله قبل مغادرة الصفحة.

اما من ناحية طريقة انشائه عملياً فهو مشابه للأنواع الأخرى، ولكن هنا نختار الخيار CanDeactivate، كالتالي:



بعده نضغط Enter من لوحة المفاتيح، ويتم إنشاء الملف، كالتالي:



والآن لنستعرض محتويات هذه الملف، كالتالي:

```

can-deactivate.guard.ts ملف
import { Injectable } from '@angular/core';
import {
  CanDeactivate,
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  UrlTree } from '@angular/router';
import { Observable } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class CanDeactivateGuard implements CanDeactivate<unknown> {
  canDeactivate(
    component: unknown,
    currentRoute: ActivatedRouteSnapshot,
    currentState: RouterStateSnapshot,
    nextState?: RouterStateSnapshot):
    Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    return true;
  }
}

```

نلاحظ أنه مشابهه بعض الشيء للأنواع السابقة، ومن ناحية الفروق أولاً نلاحظ انه يعمل implements لinterface ذو الاسم CanDeactivate، ومنها أيضاً ان هذا الinterface من النوع generic أي لا بد ان نضع له نوع وهذا النوع وافترضياً تم وضع unknown (وهو في العادة component الذي نريد ان نعمل له حماية بواسطة هذا النوع)، وهناك عدة طرق للتعامل مع هذا النوع، وسوف اذكر منها طريقتين الأولى سوف أنشئ CanDeactivate عامة لأي component يُريد استخدامها مع ايضاً رسالة عامة بحيث يستخدمها أي component، والطريقة الثانية تكون مخصصة لcomponent محدد والرسالة التي تظهر للمستخدم ايضاً خاصة لهذا component فقط.

### 1.4.3 CanDeactivate عامة لجميع Components في التطبيق:

في البداية نضيف interface بحيث يكون النوع generic الخاص بالinterface المسمى CanDeactivate هو interface الذي قمنا بإنشائه وليكن اسمه CanComponentDeactivate، ولا ننسى نضيفه أيضاً للباراميتري ذو الاسم component، كالتالي:

ملف can-deactivate.guard.ts

```
import { Injectable } from '@angular/core';
import {
  CanDeactivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree } from '@angular/router';
import { Observable } from 'rxjs';
import { LoginComponent } from '../authentication/login/login.component';

export interface CanComponentDeactivate { ←
  canDeactivate: () => boolean;
}

@Injectable({
  providedIn: 'root'
})

export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> { ←

  canDeactivate(
    component: CanComponentDeactivate, ←
    currentRoute: ActivatedRouteSnapshot,
    currentState: RouterStateSnapshot,
    nextState?: RouterStateSnapshot):
    Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

  }
}
```

والinterface الذي قمنا بإنشائه يحتوي على دالة اسميتها ( ) canDeactivate (لك حرية اختيار الاسم الذي تريده) وهذه الدالة تُعيد boolean (يمكن إضافة نوع الأعادة الذي تُريده بحسب احتياجك، فعلى سبيل المثال يمكن ان نجعل هذه الدالة تُعيد boolean | Promise<boolean> | Observable<boolean>)، بحيث أن أي component نُريد ان نستخدم معه هذا الكلاس لابد أن يحتوي على نفس اسم هذه الدالة، اما الكلاس CanDeactivateGuard فيقوم بإعادة هذه الدالة فقط بغض النظر عن ما هو محتوى هذه الدالة حيث ان محتواها يتم كتابته في نفس component، كالتالي:

ملف can-deactivate.guard.ts

```
import { Injectable } from '@angular/core';
import {
  CanDeactivate,
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  UrlTree } from '@angular/router';
import { Observable } from 'rxjs';
import { LoginComponent } from '../authentication/login/login.component';

export interface CanComponentDeactivate {
  canDeactivate: () => boolean;
}
```

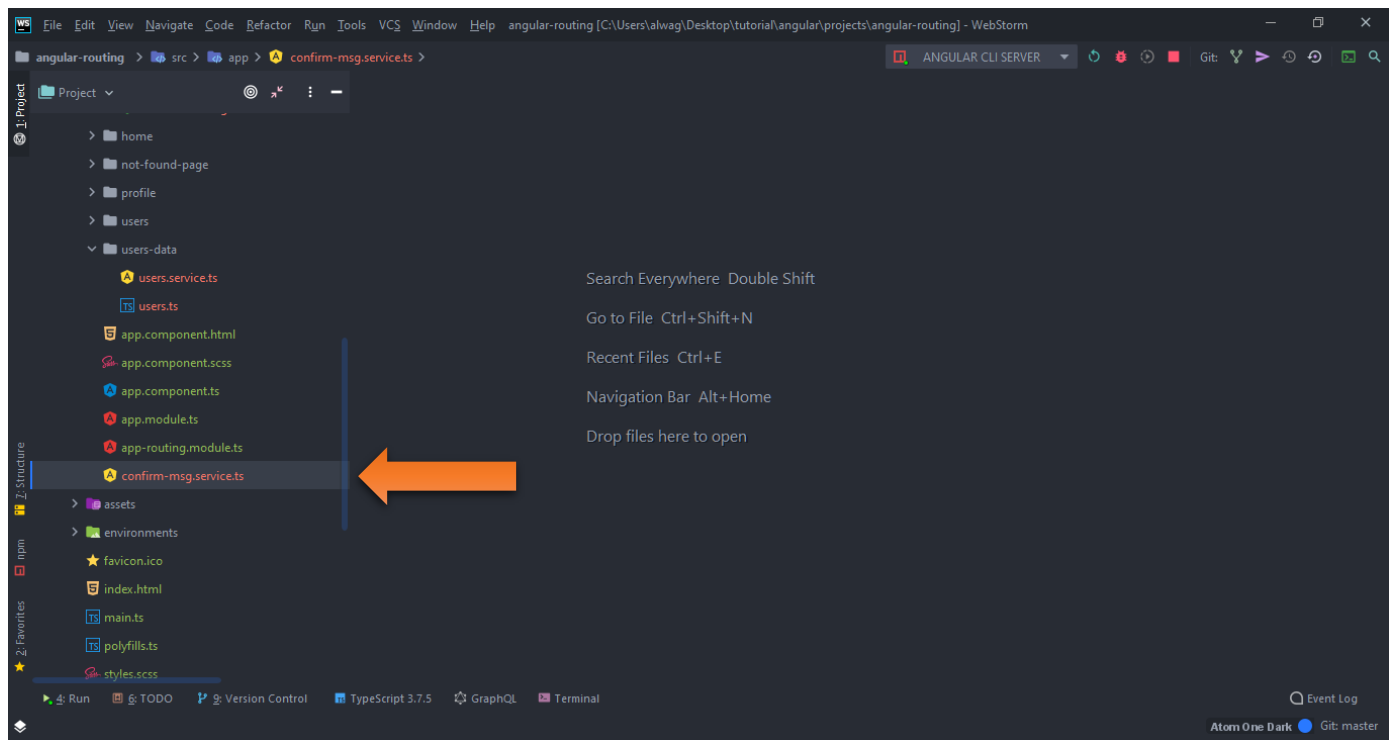
```

@Injectable({
  providedIn: 'root'
})

export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> {
  canDeactivate(
    component: CanComponentDeactivate,
    currentRoute: ActivatedRouteSnapshot,
    currentState: RouterStateSnapshot,
    nextState?: RouterStateSnapshot):
    Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    if (component.canDeactivate) {
      return component.canDeactivate();
    }
    return true;
  }
}

```

كما هو واضح في البداية يتم وضع شرط لتأكد هل الدالة موجودة وقيمتها true، فإذا كانت كذلك فنعيد محتوى هذه الدالة (الذي ارجع واعيد محتواها يتم كتابته في نفس component) اما إذا لم يكن فنعيد true بمعنى اسمح بتغيير route وتوجيه المستخدم إلى route الجديد، اما الباراميتير component فهو يشير إلى component المُحتوي على هذه الدالة. الآن لنقوم بإنشاء service ومهمتها مشاركة الرسالة التي تظهر للمستخدم، بحيث أن أي component يحتوي على الدالة السابقة، يستدعي هذا service ليظهر رسالة معينة للمستخدم، كالتالي:



الآن لنفتح هذا الملف ونكتب محتوياته، كالتالي:

```

confirm-msg.service.ts ملف
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class ConfirmMsgService {

```

```
confirm(message: string): boolean {
  return window.confirm(message);
}
}
```

وكما هو واضح تحتوي هذه الخدمة على دالة واحدة اسميتها confirm وهي تُعيد boolean لأن الدالة window.confirm التي تقدمها لنا javascript هي أيضاً تُعيد boolean بحيث إذا ضغط المستخدم موافق تُعيد true وإذا ضغط الغاء تُعيد false. (مع العلم ان هذه الخطوة غير مهمة نستطيع الاكتفاء بإظهار الرسالة داخل component مباشر باستخدام window.confirm بدون الحاجة إلى هذه service ولكن من باب التدريب تم التطرق لهذه الخطوة).

الآن نذهب إلى ملف app-routing.module.ts ونضيف هذا الكلاس CanDeactivateGuard إلى route ذو الاسم signup، كالتالي:

ملف app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { UsersListComponent } from './users/users-list/users-list.component';
import { UserDetailsComponent } from './users/user-details/user-details.component';
import { AuthenticationComponent } from './authentication/authentication.component';
import { LoginComponent } from './authentication/login/login.component';
import { SignupComponent } from './authentication/signup/signup.component';
import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
import { SingoutComponent } from './authentication/singout/singout.component';
import { ProfileComponent } from './profile/profile.component';
import { EditeCoursesComponent } from './home/edite-courses/edite-courses.component';
import { CanActivateGuard, CanActivateAdminGuard } from './guards/can-activate.guard';
import { CanActivateAdminChildGuard, CanActivateChildGuard } from './guards/can-activate-child.guard';
import { CanDeactivateGuard } from './guards/can-deactivate.guard';

const routes: Routes = [
  {
    path: 'home', component: HomeComponent, canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
    children: [
      { path: 'edite-courses', component: EditeCoursesComponent }
    ]
  },
  { path: 'profile/:id', canActivate: [CanActivateGuard], component: ProfileComponent },
  {
    path: 'users',
    canActivate: [CanActivateGuard, CanActivateAdminGuard],
    data: {
      roles: 'admin'
    },
    children: [
      { path: '', component: UsersComponent },
      { path: 'user-list', component: UsersListComponent },
      { path: 'user-details/:id', component: UserDetailsComponent },
    ]
  },
  {
    path: 'auth', children: [
      { path: '', component: AuthenticationComponent },
      { path: 'login', component: LoginComponent },
    ]
  }
]
```



```

    { path: 'signup', canActivate: [CanDeactivateGuard], component: SignupComponent },
    { path: 'signout', component: SingoutComponent }
  ],
},
{ path: '', redirectTo: 'home', pathMatch: 'full' },
{ path: '**', component: NotFoundPageComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

الآن لنذهب إلى ملف signup.component.ts ونكتب محتويات الدالة canDeactivate كما اشرنا سابقاً، كالتالي:

ملف signup.component.ts

```

import {Component, OnInit, ViewChild} from '@angular/core';
import {NgForm} from '@angular/forms';
import { UsersService } from 'src/app/users-data/users.service';
import { ConfirmMsgService } from '.../confirm-msg.service';
import { Observable } from 'rxjs';
@Component({
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrls: ['./signup.component.scss']
})

export class SignupComponent implements OnInit {

  @ViewChild('form', { static: false}) form: NgForm;
  private isSaved = false;

  constructor(
    private usersService: UsersService,
    private confirmMsgService: ConfirmMsgService) { }

  ngOnInit() {
  }

  onSubmit() {
    this.isSaved = true;
    this.usersService.addNewUser(this.form.value);
  }

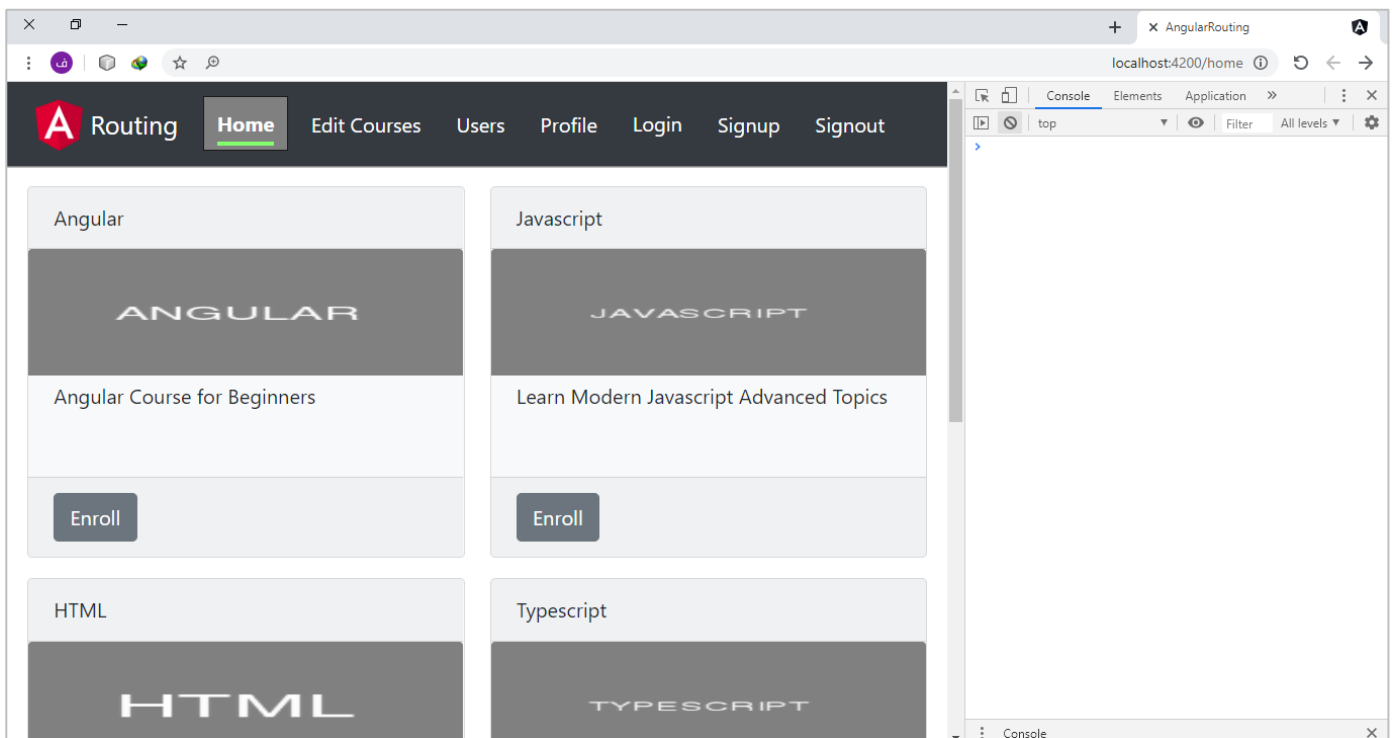
  canDeactivate(): boolean {
    if (this.form.dirty === true && this.isSaved === false) {
      return this.confirmMsgService.confirm('لم يتم حفظ البيانات، هل تريد التأكيد على الانتقال؟');
    }
    return true;
  }
}

```

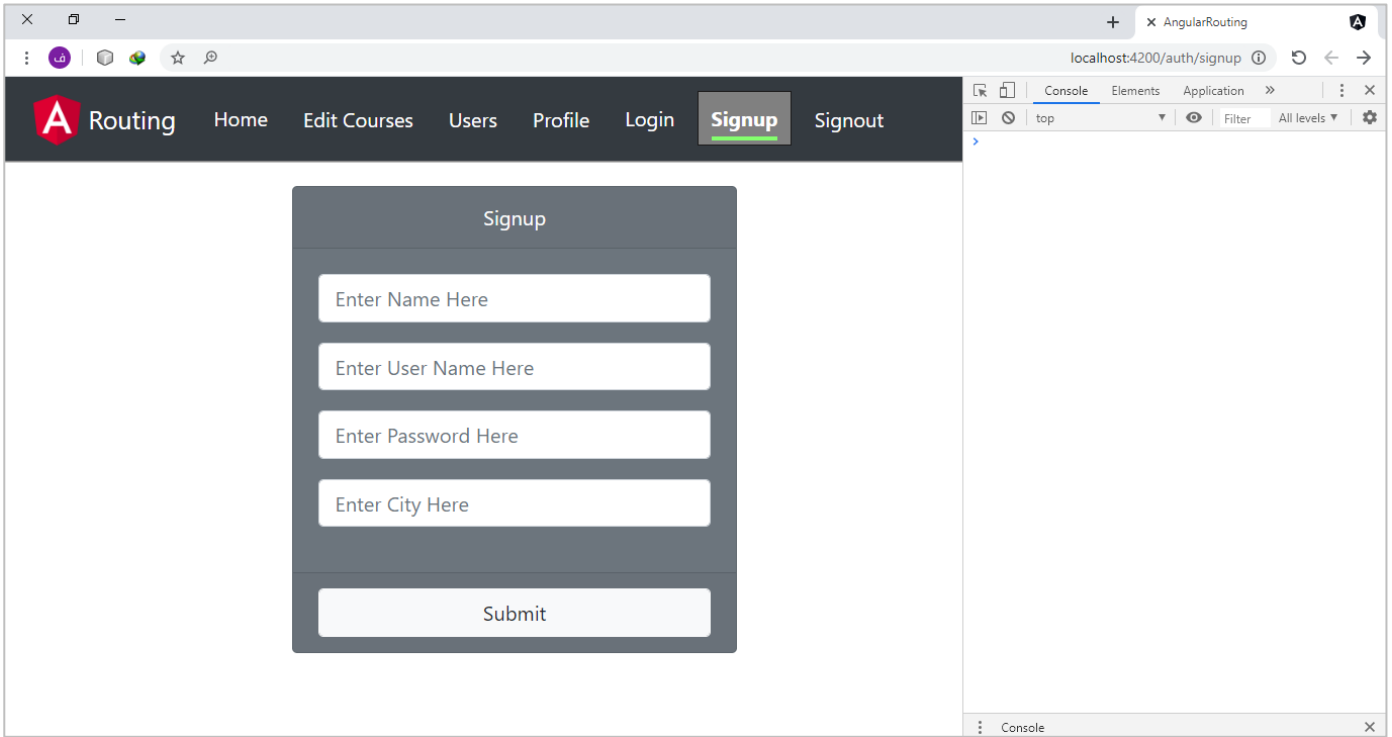
في البداية عرفنا متغير واسميناه form وهو يشير إلى المتغير الذي اسميناه بنفس الاسم في ملف singup.component.html، وهذا المتغير يرمز إلى النموذج الخاص بتسجيل مستخدم جديد، اما المتغير الثاني فهو من النوع boolean واسميناه isSaved واعطيته قيمة مبدئية false ومن اسمه مهمته التأكد من ان المستخدم ضغط على زر حفظ ام لا، بحيث إذا ضغط على زر حفظ لا نريده ان يظهر الرسالة ويسمح بالانتقال إلى route الجديد، وايضاً قمنا

يعمل inject لخدمة service التي اشئناها قبل قليل والتي مهمتها إظهار الرسالة في متغير اسميته confirmMsgService، وفي الدالة ( ) onSubmit إذا قام المستخدم بحفظ البيانات، نقوم بوضع قيمة المتغير isSaved بالقيمة true، بحيث لا يظهر الرسالة، وأخيراً نقوم بإنشاء الدالة canActivate التي تكلمنا عنها سابقاً، ومحتواها بكل بساطة في البداية نضع شرط لتأكد ان المستخدم قام بالتعديل في النموذج من خلال الخاصية dirty بحيث تصبح قيمتها true إذا قام المستخدم بإضافة أي بيانات في النموذج، والجزء الثاني من الشرط التأكد من أن قيمة المتغير تساوي false بمعنى ان المستخدم لم يقوم بضغط على زر الحفظ، فإذا تحقق هذين الشرطين نقوم بإظهار الرسالة ونمرر لها محتوى الرسالة الذي يناسبنا، وهذه الرسالة اما ان تُعيد true (ضغط المستخدم على زر موافق) وفي هذه الحالة يتم السماح بعمل route والتوجيه لـ routed الجديد، اما إذا اعادت false (المستخدم ضغط على زر الغاء) فعندها يتم عمل الغاء لـ routed والبقاء بنفس الصفحة بدون عمل أي توجيه، اما في حالة عدم تحقق أي شرط يُرجع true. (ولفهم أعمق لطريقة تعامل Angular مع النماذج الرجاء مراجعة الكتاب الخامس من هذه السلسلة Angular Forms)

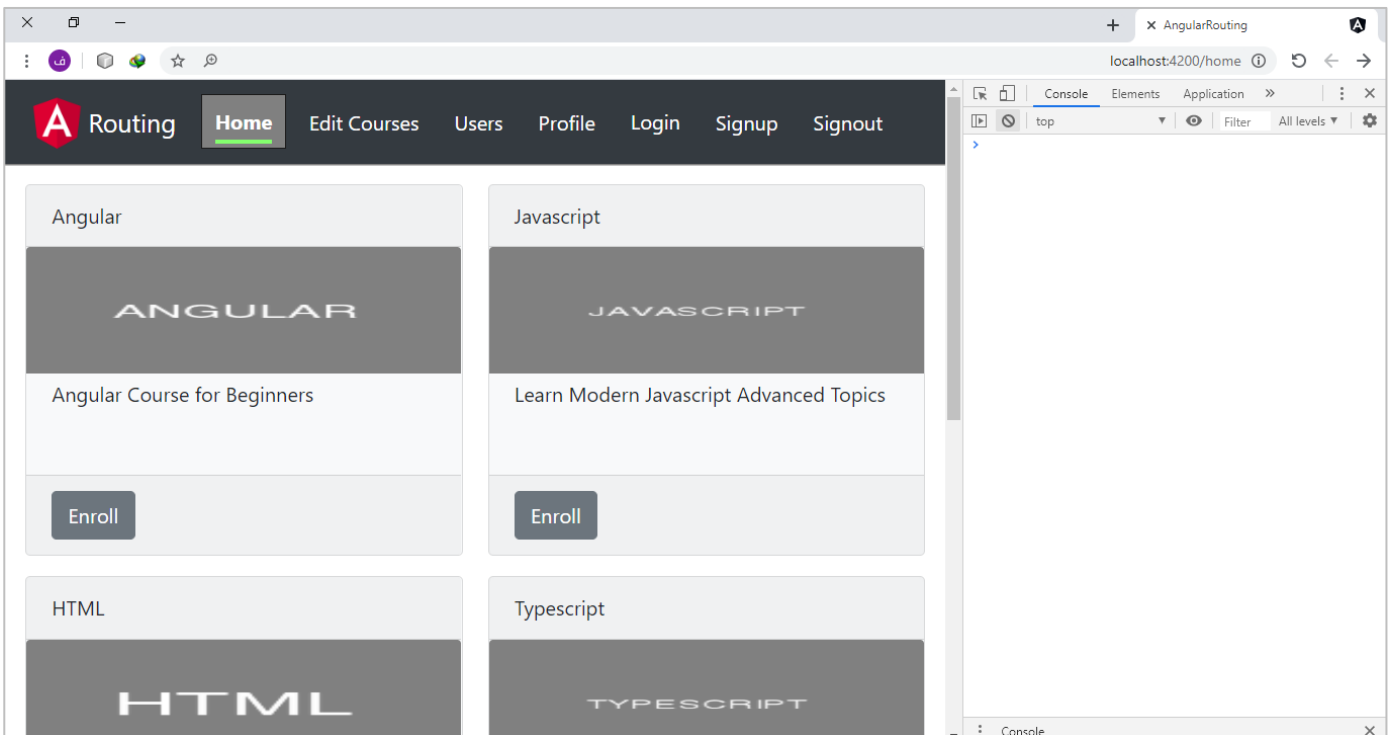
الآن لنحفظ التعديلات ونرى النتيجة في المتصفح، كالتالي:



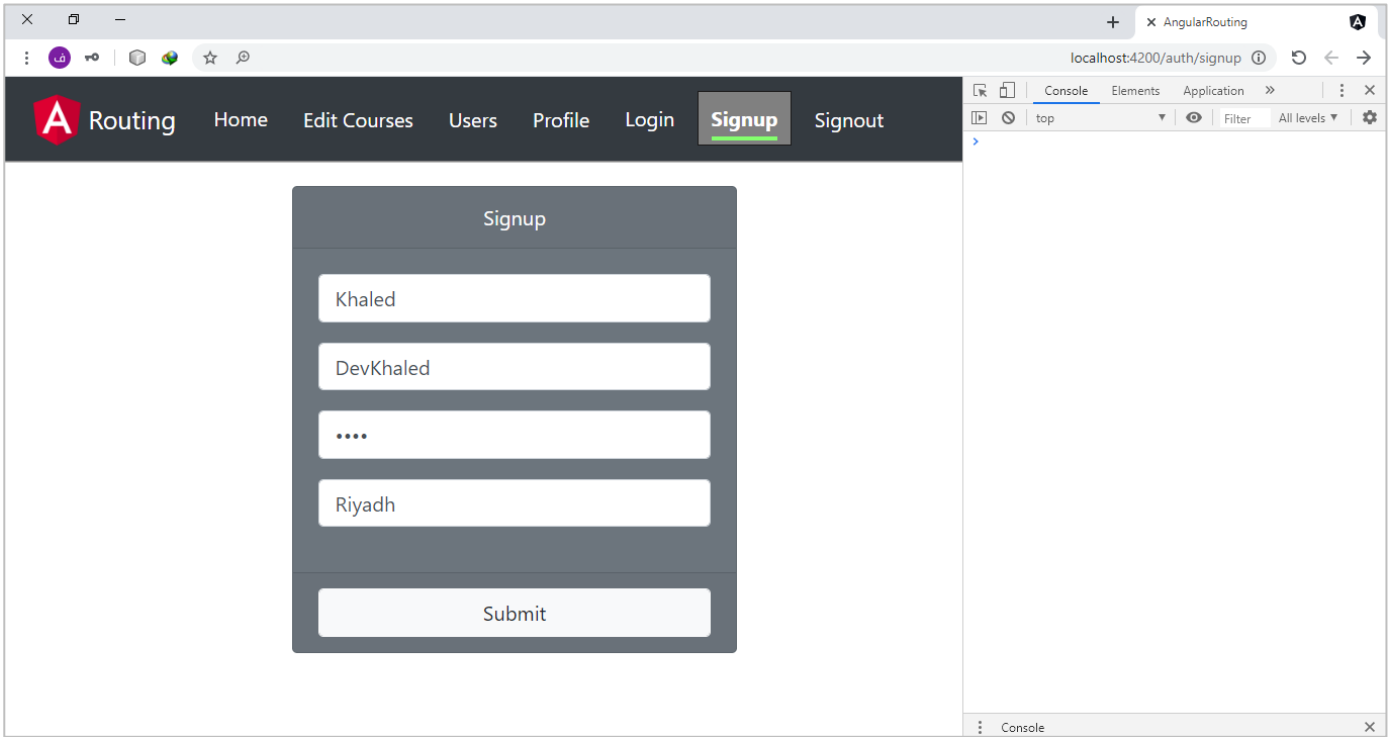
الآن لنختار التبويب Signup، ونرى النتيجة:



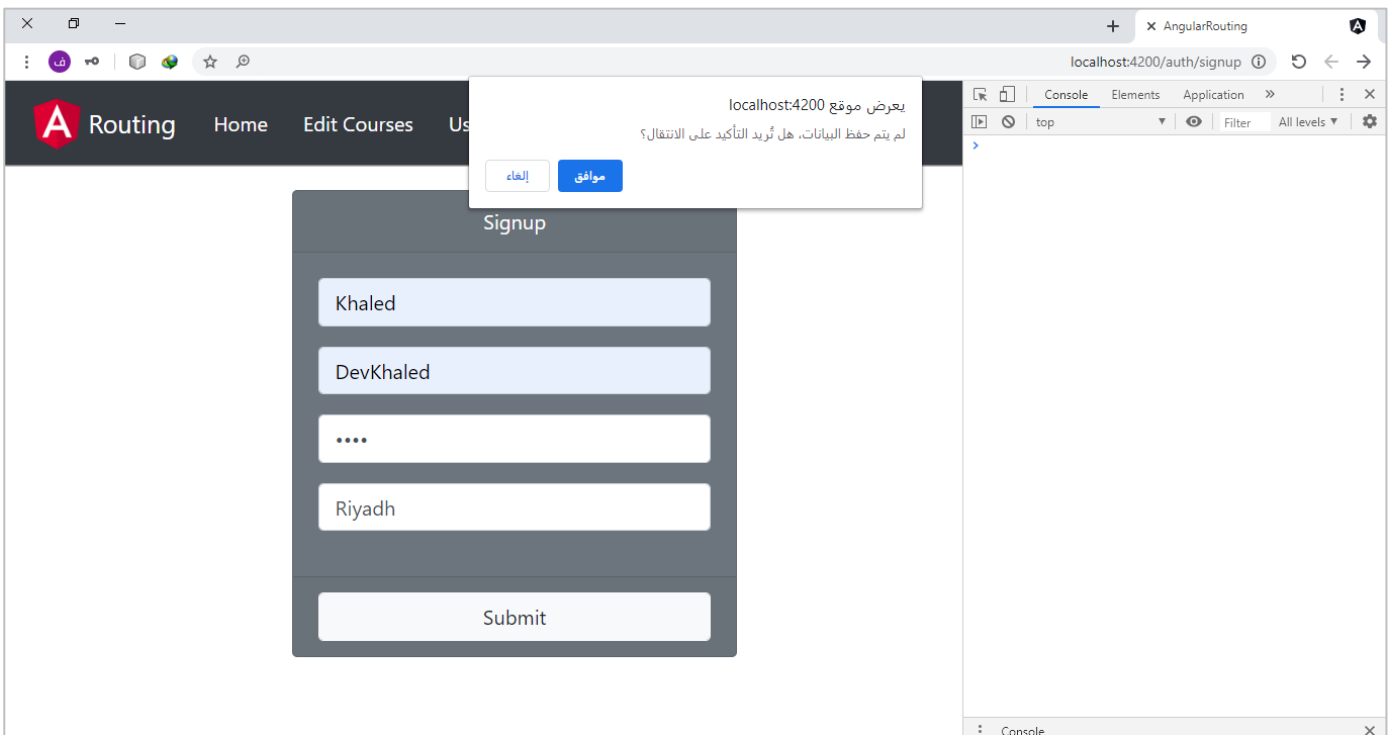
نلاحظ انه لم يحدث شيء الآن لنقم بالرجوع إلى صفحة Home مع التأكيد بعدم وضع مؤشر الفأرة على أي حقل من حقول النموذج او كتابة أي شيء، كالتالي:



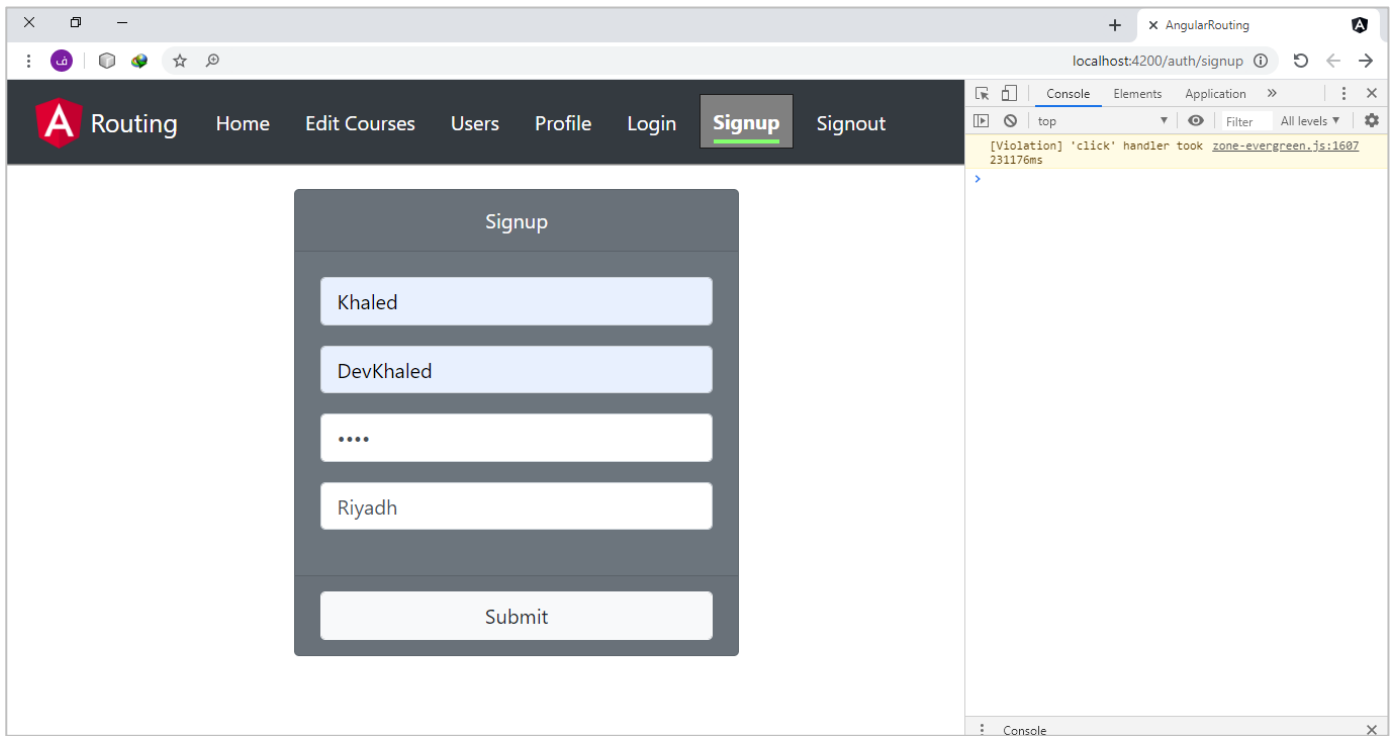
نلاحظ انه تم التوجيه إلى route الجديد Home بدون لا يظهر أي رسالة، وهذا هو المتوقع لأننا لم نقم بلمس أي حقل من حقول النموذج او إضافة أي نص وبالتالي قيمة dirty هي false ومن هذا المنطلق الشرط لم يتحقق الشرط وسوف تُعيد الدالة أي اسمح بالتوجيه، الآن لنرجع مرة أخرى إلى Signup، ولنضيف بعض البيانات ولكن لا نضغط على زر حفظ او submit، كالتالي:



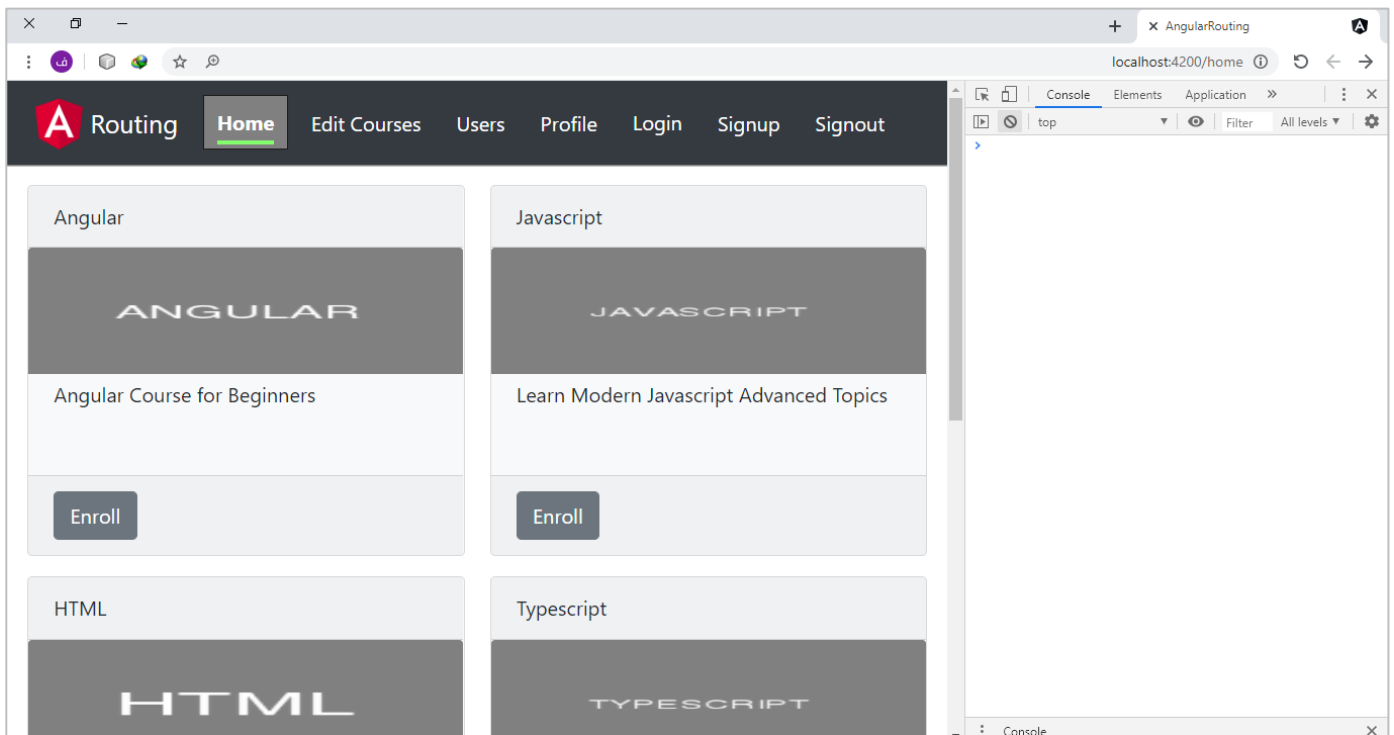
بعدما قمنا بإضافة البيانات، والتأكيد مرة أخرى بعدم الضغط على زر Submit، نضغط على تبويب Home، كالتالي:



نلاحظ انه ظهرت لنا الرسالة وهذا هو المتوقع، الآن في حال اخترنا الزر موافق فإننا سوف نعيد true ونسمح بالتوجيه إلى routed الجديد home، اما في حال اخيار الزر الغاء فإننا سوف نُعيد false وفي هذه الحالة سوف يقوم بعمل الغاء لrouted الجديد والبقاء بنفس route.



في حال الضغط على زر الغاء، اما في حال الضغط على زر موافق، فتكون النتيجة كالتالي:



وهذه في حالة اختيارنا لزر موافق فسوف يسمح بالتوجيه لـ routed الجديد home.

وبذلك نكون قمنا بتغطية الطريقة الأولى وهي التي تصلح لأي component فكل الذي نعمله هو وضع اسم الكلاس CanDeactivateGuard في routed المستهدف داخل ملف app-routing.module.ts، وفي component المستهدف نكتب محتويات الدالة () canDeactivate، والآن لنقوم بشرح الطريقة الثانية.

### 1.4.3 CanDeactivate مخصصة لـ Component محدد:

وهذه الطريقة مبسطة وتكون موجهة إلى component محدد بحيث نمرر النوع generic هو component المستهدف، وفي مثالنا هنا هو LoginComponent، حيث سوف نضيف كلاس جديد في ملف can-deactivate.guard.ts وليكن اسمه CanDeactivateLoginGuard، كالتالي:

ملف can-deactivate.guard.ts

```
import { Injectable } from '@angular/core';
import {
  CanDeactivate,
  ActivatedRouteSnapshot,
  RouterStateSnapshot,
  UrlTree } from '@angular/router';
import { Observable } from 'rxjs';
import { LoginComponent } from '../authentication/login/login.component';

export interface CanComponentDeactivate {
  canDeactivate: () => boolean;
}

@Injectable({
  providedIn: 'root'
})

export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate> {
  canDeactivate(
    component: CanComponentDeactivate,
    currentRoute: ActivatedRouteSnapshot,
    currentState: RouterStateSnapshot,
    nextState?: RouterStateSnapshot):
    Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    if (component.canDeactivate) {
      return component.canDeactivate();
    }
    return true;
  }
}

@Injectable({
  providedIn: 'root'
})

export class CanDeactivateLoginGuard implements CanDeactivate <LoginComponent> {
  canDeactivate(
    component: LoginComponent,
    currentRoute: ActivatedRouteSnapshot,
    currentState: RouterStateSnapshot,
    nextState?: RouterStateSnapshot):
    Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {

    if (component.form.dirty === true && component.isSaved === false) {
      return confirm('تسجيل قبل الانتقال في الرغبة من متأكد ');
    }
    return true;
  }
}
```

هنا جعلنا generic type لـ interface ذو الاسم CanDeactivate هو LoginComponent.

ايضاً نجعل النوع لهذا الباراميتر هو نفسه component المستهدف

if (component.form.dirty === true && component.isSaved === false) {  
 return confirm('تسجيل قبل الانتقال في الرغبة من متأكد ');  
}

وبما ان الباراميتير component اصبح يشير إلى LoginComponent فإنه يمكننا أن نصل إلى جميع الدوال والخصائص التي يحتويها ومنها form.dirty و isSaved، وعن طريقها وضعنا الشروط كما كنا نفعل سابقاً، وفي حال تحقق هذه الشروط يظهر الرسالة عن طريق الدالة confirm الموجودة اصلاً في JavaScript.

ولا ننسى أن نُعرف هذه المتغيرات (form.dirty و isSaved) في ملف login.component.ts، كالتالي:

ملف login.component.ts

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { UsersService } from 'src/app/users-data/users.service';
import { NgForm } from '@angular/forms';
import { ActivatedRoute, Router } from '@angular/router';
import { StorageMap } from '@ngx-pwa/local-storage';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {
  @ViewChild('form') form: NgForm; ←
  public isSaved = false; ←
  private path: string;

  constructor(
    private userService: UsersService,
    private activatedRoute: ActivatedRoute,
    private router: Router,
    private storageMap: StorageMap,
  ) { }

  ngOnInit() {
    if (this.activatedRoute.snapshot.queryParams.redirectUrl) {
      this.path = this.activatedRoute.snapshot.queryParams.redirectUrl;
    } else {
      this.path = '';
    }
  }

  onSubmit() {
    const value = this.form.controls.userName.value;
    this.userService.logIn(value);
    this.storageMap.get('id').subscribe((id) => {
      if (this.path === '') {
        this.router.navigate(['/home']);
      } else if (this.path === '/users' || this.path === '/home/edite-courses') {
        this.router.navigate(['`/${this.path}`']);
      } else {
        const path = this.path.split('/');
        this.router.navigate(['`/${path[1]}/${id}`']);
      }
    });
    this.isSaved = true;
  }
}
```

ولا ننسى ان نضيف الكلاس إلى route ذو الاسم login في ملف app-routing.module.ts، كالتالي:

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from '../home/home.component';
import { UsersComponent } from '../users/users.component';
import { UsersListComponent } from '../users/users-list/users-list.component';
import { UserDetailsComponent } from '../users/user-details/user-details.component';
import { AuthenticationComponent } from '../authentication/authentication.component';
import { LoginComponent } from '../authentication/login/login.component';
import { SignupComponent } from '../authentication/signup/signup.component';
import { NotFoundPageComponent } from '../not-found-page/not-found-page.component';
import { SingoutComponent } from '../authentication/singout/singout.component';
import { ProfileComponent } from '../profile/profile.component';
import { EditeCoursesComponent } from '../home/edite-courses/edite-courses.component';
import { CanActivateGuard, CanActivateAdminGuard } from '../guards/can-activate.guard';
import { CanActivateAdminChildGuard, CanActivateChildGuard } from '../guards/can-activate-child.guard';
import { CanDeactivateGuard, CanDeactivateLoginGuard } from '../guards/can-deactivate.guard';

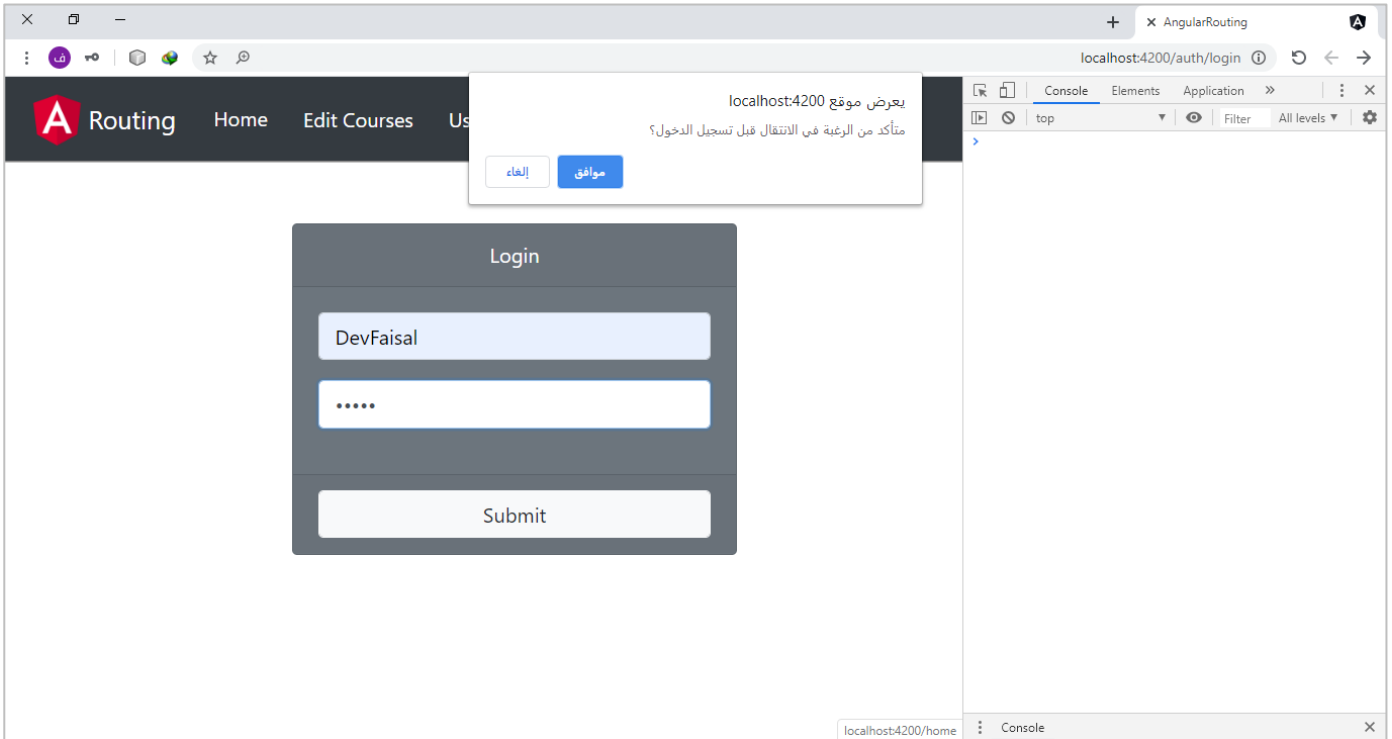
const routes: Routes = [
  {
    path: 'home', component: HomeComponent, canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
    children: [
      { path: 'edite-courses', component: EditeCoursesComponent }
    ]
  },
  { path: 'profile/:id', canActivate: [CanActivateGuard], component: ProfileComponent },
  {
    path: 'users',
    canActivate: [CanActivateGuard, CanActivateAdminGuard],
    data: {
      roles: 'admin'
    },
    children: [
      { path: '', component: UsersComponent },
      { path: 'user-list', component: UsersListComponent },
      { path: 'user-details/:id', component: UserDetailsComponent },
    ]
  },
  {
    path: 'auth', children: [
      { path: '', component: AuthenticationComponent },
      { path: 'login', canDeactivate: [CanDeactivateLoginGuard], component: LoginComponent },
      { path: 'signup', canDeactivate: [CanDeactivateGuard], component: SignupComponent },
      { path: 'singout', component: SingoutComponent }
    ]
  },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: '**', component: NotFoundPageComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

ولو قمنا بالتجربة، فسوف نشاهد ما هو متوقع من إظهار الرسالة في حالة التعديل وتغيير route، كالتالي:





وبذلك نكون أنهينا هذه النوع وسوف ننتقل إلى النوع الرابع وهو CanLoad.

### 5.3. CanLoad guard:

سوف نؤجل الكلام في هذا النوع إلى أن نتطرق إلى ميزة Lazy Loading.

### 6.3. Angular Resolve Guard:

هذا النوع بعض المراجع لا تُدرجه ضمن أنواع Route Guards، ومراجع أخرى تفعل العكس، وحقيقة من وجهة نظري أرى أنه يمكن إدراجه تحت هذه الأنواع لأنه يقوم بالعمل في حال تغير Route مثله مثل الأنواع الأخرى.

أما الفائدة من هذا النوع فتكمن في جلب او عمل fetch للبيانات قبل عرضها على الصفحة، فمثلاً لدينا صفحة وهذه الصفحة تعرض بيانات وتقوم بجلب هذه البيانات من سيرفر معين، وكما هو معروف ان الاتصال بالسيرفر وجلب البيانات بعض الأحيان قد يستغرق بعض الوقت وهذا قد يؤدي إلى عرض صفحة فارغة للمستخدم قبل عرض البيانات له مما تعتبر تجربة غير مستحبة للمستخدمين، وهذا الامر له عدة حلول منها وضع علامة Loading إلى أن يتم جلب البيانات بشكل كامل ومن ثم إظهار هذه الصفحة، ومن الحلول أيضاً هو Resolve Guard، حيث تقوم بمنع الانتقال إلى route إلى أن يتم جلب البيانات بشكل كامل من السيرفر وعندما يتم جلب هذه البيانات يسمح بالانتقال او التوجيه إلى route الجديد، و Resolve Guard أيضاً له فوائد أخرى منها عمل مشاركة لهذه البيانات التي تم عمل لها Resolver إلى Route الأبناء، ومنها أيضاً في حال فشل جلب البيانات نقوم بإلغاء التوجيه إلى هذا route من الأساس او مثلاً نعمل له توجيه إلى صفحة أخرى تبين له أن هنالك خطأ معين مع بعض طرق حل هذه الأخطاء.

لذلك بدلاً من أن يقوم component نفسه بطلب البيانات من السيرفر والانتظار إلى أن يأتي الرد ومن ثم يقوم بعرض هذه البيانات، سيقوم Resolver بطلب هذه البيانات ولن يتم عرض هذا component إلى أن يتم جلب البيانات الضرورية ومن ثم تُرسل هذه البيانات من Resolver إلى component لعرضها.

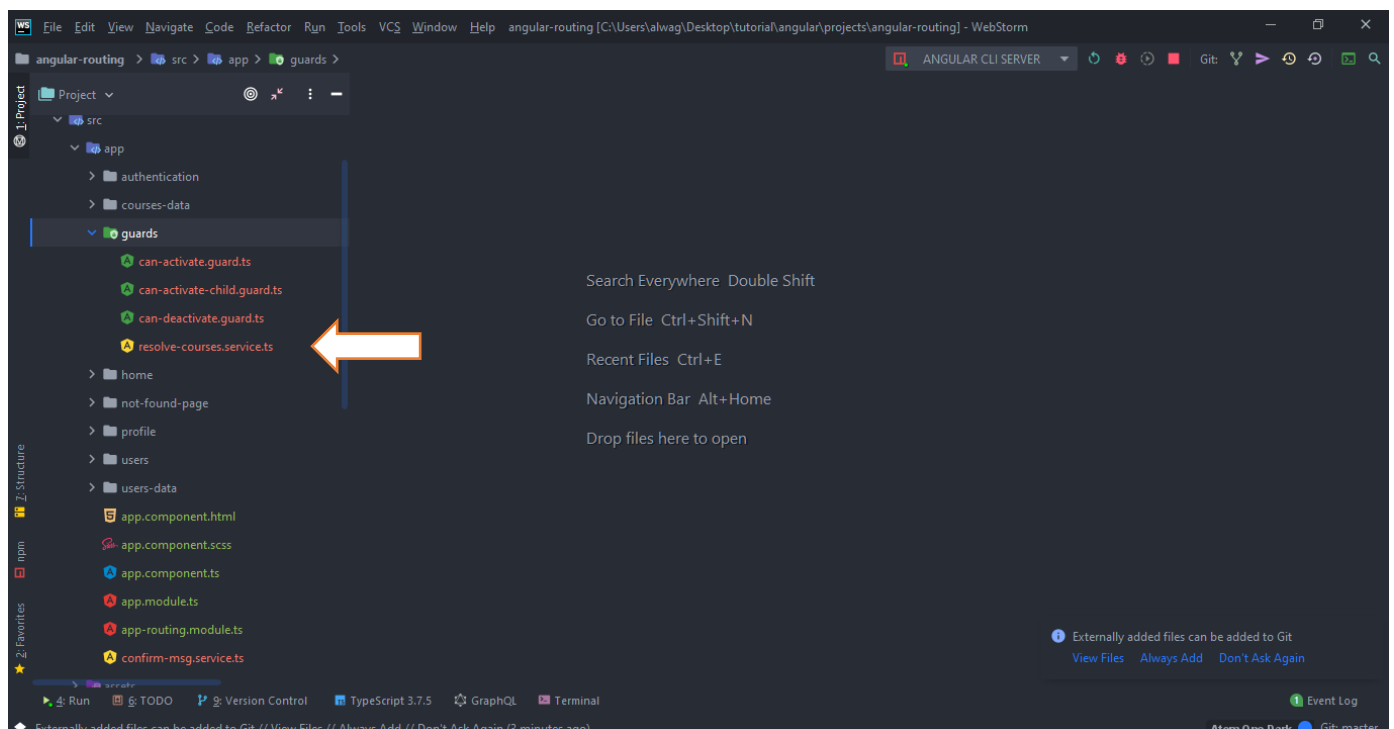
وتتم هذه العملية من خلال ثلاث خطوات رئيسية، هي:

أولاً/ بناء Resolve Guard وهو عبارة عن service.

ثانياً/ وضع resolve في تهيئة route، الذي هو في مثالنا هنا هو الموجود في ملف app-routing.module.ts.

ثالثاً/ قراءة البيانات من Resolver بعد جلبها لعرضها في component وتتم كما كنا نقرأ الباراميترات وخاصية data التي شرحناها سابقاً بواسطة ActivatedRoute.

لذلك لنقوم بأول خطوة وهي بناء Resolve Guard، وسوف نقوم بإنشاء Resolver لجلب بيانات الكورسات، ملاحظة إلى لحظة تأليف هذا الكتاب لا يوجد امر في angular cli يقوم بإنشاء Resolver بشكل تلقائي لذلك لا بد ان نقوم بإنشائه يدوياً، لذلك وكما قلنا سابقاً عبارة عن service لذلك لنقوم بإنشاء service عن طريق كتابة الامر ng g service guards/resolve-courses.



اما محتويات هذا الملف فهو مشابه لمحتويات أي service أخرى، كالتالي:

ملف resolve-courses.service.ts

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class ResolveCoursesService {
```

```

    constructor() { }
}

```

لذلك لجعل هذا الملف Resolver لابد في البداية ان نجعل الكلاس يعمل implements لـ interface التي تحمل الاسم Resolve وهي ايضاً generic أي لابد ان نممر لها نوع البيانات التي سوف تعمل لها جلب وفي حالتنا هنا هي الكلاس Courses على شكل مصفوفة، كالتالي:

ملف resolve-courses.service.ts

```

import { Injectable } from '@angular/core';
import { Courses } from '../courses-data/courses';
import { Resolve } from '@angular/router';

@Injectable({
  providedIn: 'root'
})
export class ResolveCoursesService implements Resolve<Courses[]> {

  constructor() { }
}

```

ايضاً هذا الـ interface يحتوي على دالة تحمل نفس الاسم.

ملف resolve-courses.service.ts

```

import { Injectable } from '@angular/core';
import { Courses } from '../courses-data/courses';
import { Resolve } from '@angular/router';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ResolveCoursesService implements Resolve<Courses[]> {

  constructor() { }

  resolve(): Observable<Courses[]> {
  }
}

```

وبما ان الدالة ( getAllCourses ) الموجودة في ملف service تُعيد Observable من النوع [ Courses ] فأنتنا نُعيد ايضاً نفس النوع هنا للدالة ( resolve )، الآن لنستدعي الدالة getAllCourses، كالتالي:

ملف resolve-courses.service.ts

```

import { Injectable } from '@angular/core';
import { Courses } from '../courses-data/courses';
import { Resolve } from '@angular/router';
import { Observable } from 'rxjs';
import { CoursesService } from '../courses-data/courses.service';
import { delay } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class ResolveCoursesService implements Resolve<Courses[]> {

  constructor(private coursesService: CoursesService) { }
}

```

```

resolve(): Observable<Courses[]> {
  return this.coursesService.getAllCourses().pipe(delay(2000));
}
}

```

هنا اضعنا الدالة delay ومررنا لها القيمة 2000 بمعنى ثانيتين، والمقصود هو فقط محاكاة لطلب البيانات من server ونتوقع ان تتأخر هذه البيانات بمقدار ثانيتين قبل ان تعرض للمستخدم. وبذلك نكون انشئنا Resolver الخاص بالبنا، وباقى ان نضعه في التهيئة الخاصة بـ routes، وسوف نضيف هذا الـ Resolver لـ home لأن فيه يتم عرض الكورسات، كالتالي:

ملف app-routing.module.ts

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from '../home/home.component';
import { UsersComponent } from '../users/users.component';
import { UsersListComponent } from '../users/users-list/users-list.component';
import { UserDetailsComponent } from '../users/user-details/user-details.component';
import { AuthenticationComponent } from '../authentication/authentication.component';
import { LoginComponent } from '../authentication/login/login.component';
import { SignupComponent } from '../authentication/signup/signup.component';
import { NotFoundPageComponent } from '../not-found-page/not-found-page.component';
import { SingoutComponent } from '../authentication/singout/singout.component';
import { ProfileComponent } from '../profile/profile.component';
import { EditeCoursesComponent } from '../home/edite-courses/edite-courses.component';
import { CanActivateGuard, CanActivateAdminGuard } from '../guards/can-activate.guard';
import { CanActivateAdminChildGuard, CanActivateChildGuard } from '../guards/can-activate-child.guard';
import { CanDeactivateGuard, CanDeactivateLoginGuard } from '../guards/can-deactivate.guard';
import { ResolveCoursesService } from '../guards/resolve-courses.service';

const routes: Routes = [
  {
    path: 'home',
    component: HomeComponent,
    resolve: { courses: ResolveCoursesService },
    canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
    children: [
      { path: 'edite-courses', component: EditeCoursesComponent }
    ]
  },
  { path: 'profile/:id', canActivate: [CanActivateGuard], component: ProfileComponent },
  {
    path: 'users',
    canActivate: [CanActivateGuard, CanActivateAdminGuard],
    data: {
      roles: 'admin'
    },
    children: [
      { path: '', component: UsersComponent },
      { path: 'user-list', component: UsersListComponent },
      { path: 'user-details/:id', component: UserDetailsComponent },
    ]
  },
  {
    path: 'auth', children: [
      { path: '', component: AuthenticationComponent },
      { path: 'login', canActivate: [CanDeactivateLoginGuard], component: LoginComponent },
      { path: 'signup', canActivate: [CanDeactivateGuard], component: SignupComponent },
      { path: 'signout', component: SingoutComponent }
    ]
  }
]

```

```

    },
    { path: '', redirectTo: 'home', pathMatch: 'full' },
    { path: '**', component: NotFoundPageComponent }
  ];

  @NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
  })
  export class AppRoutingModule { }

```

نلاحظ أننا أضفنا Resolver إلى Route ذو الاسم home، حيث أضفناه على شكل كائن object المفتاح key لهذا الكائن هو courses (لك حرية اختيار الاسم الذي تُريده) اما القيمة في الكلاس الموجود في service التي قمنا بإنشائها سابقاً. بقي آخر خطوة وهي قراءة هذا Resolver في component home لأن هذا component فيه يتم عرض الكورسات، لذلك لننتقل إلى ملف home.component.ts ونقوم بقراءة هذا Resolver، مع العلم ان طريقة قراءته مشابهة لحد كبير لطريقة قراءة الباراميترات التي تكلمنا عنها سابقاً، كالتالي:

ملف home.component.ts

```

import {Component, OnInit} from '@angular/core';
import {Observable} from 'rxjs';
import {ActivatedRoute} from '@angular/router';
import {Courses} from '../courses-data/courses';


@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})

export class HomeComponent implements OnInit {
  courseList$: Observable<Courses>;

  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    this.courseList$ = this.route.snapshot.data.courses;
  }
}

```




نلاحظ ان courses تُشير إلى key الذي قمنا بإنشائه سابقاً، والآن لنحذف Async pipe في ملف home.component.html، كالتالي:

ملف home.component.html

```

<div style="margin: 1rem">
  <div class="card-columns">
    <div class="card bg-light mb-3" *ngFor="let course of courseList$">
      <div class="card-header">{{ course.courseName }}</div>
      <img class="card-img-top" width="80" height="100" [src]="course.path" />
      <div class="card-body pt-1" style="height: 5rem">
        {{ course.courseDescription }}
      </div>
      <div class="card-footer">
        <button class="btn btn-secondary">Enroll</button>
      </div>
    </div>
  </div>

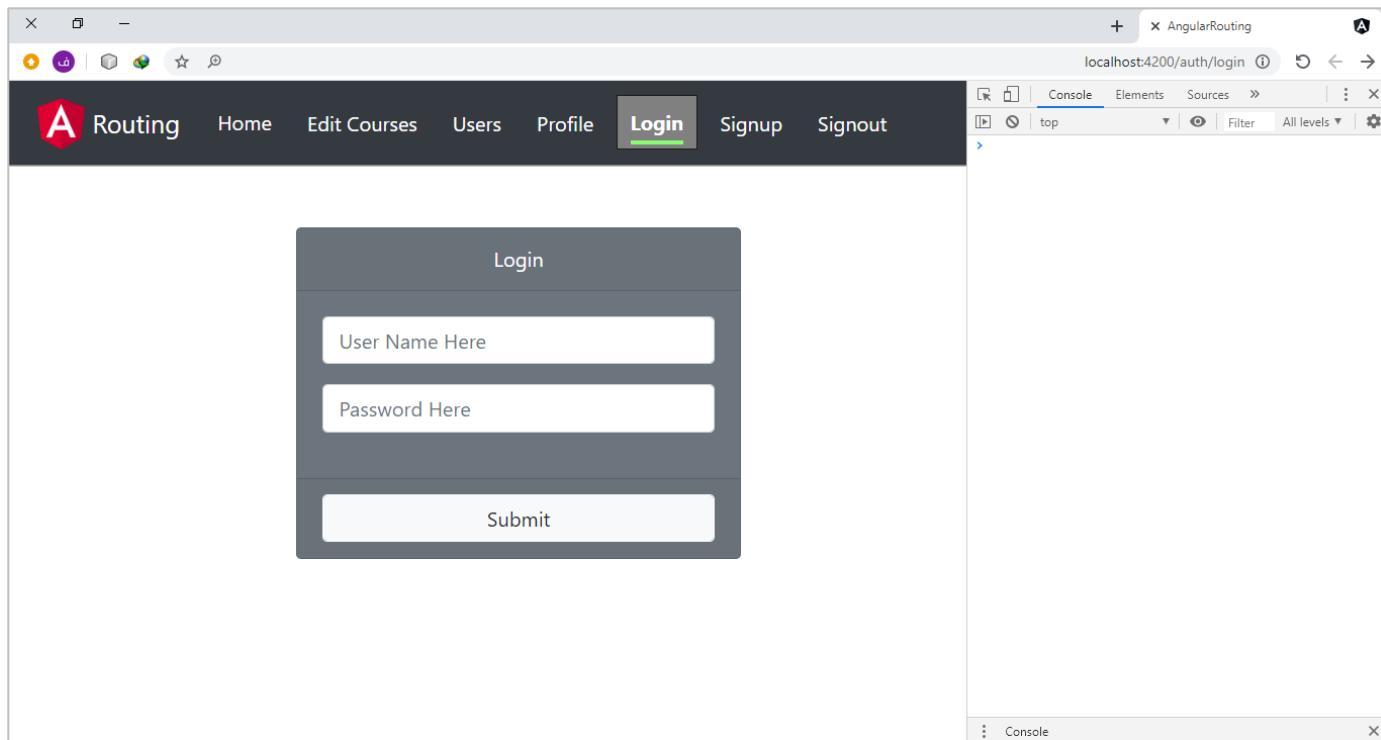
```



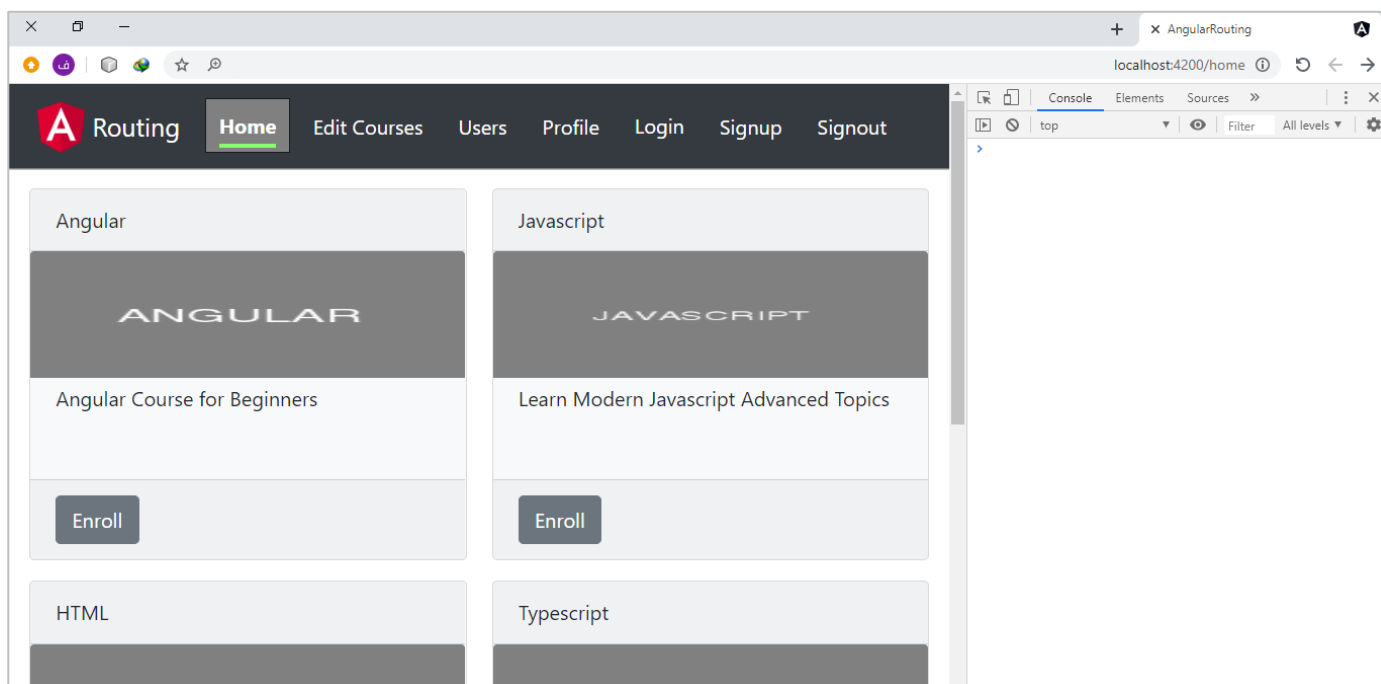
```
</div>  
</div>
```

قمنا بحذف async في السطر البرمجي المؤشر عليها بالسهم في الأعلى، مع العلم أن قراءة Resolver هي نفس قراءة البارامتر ونفسها ايضاً من ناحية استخدام snapshot او عن طريق Observable وعمل subscribe لها، ولمعلومات أكثر تستطيع الرجوع إلى قسم البارامترات وتعرف الفرق بين استخدام snapshot او Observable.

الآن لنقم بتجربة ما قمنا به من تعديلات، لنذهب إلى المتصفح ونرى النتيجة:



في البداية لنختار Login، ومن ثم نقوم باختيار صفحة Home التي تقوم بعرض الكورسات، سوف نرى انه سوف ينتظر ثانيتين قبل ان ينتقل إلى صفحة Home ويعرض البيانات، وهذا هو المتوقع، كالتالي:



وخلال فترة الانتظار التي هي في حالتنا هنا ثانيتين نستطيع ان نظهر spinner للمستخدم لكي نخبره ان النظام يجري عملية معينة وهي الانتقال إلى صفحة home، وافضل مكان لوضع spinner هو في AppComponent، اما الطريقة فهناك عدة طرق وانا هنا سوف استخدم Events التي تكلمنا عنها سابقاً بحيث نُظهر spinner في Navigation Start ونخفيها في حالة Navigation End وNavigation Cancel، كالتالي:

ملف app.component.ts

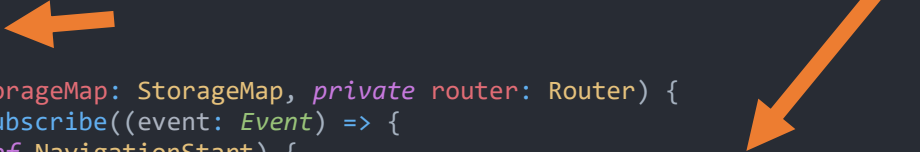
```
import { Component, OnInit } from '@angular/core';
import { StorageMap } from '@ngx-pwa/local-storage';
import { Router, Event, NavigationStart, NavigationEnd, NavigationCancel } from '@angular/router';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {

  id: string;
  showSpinner = false;

  constructor(private storageMap: StorageMap, private router: Router) {
    this.router.events.subscribe((event: Event) => {
      if (event instanceof NavigationStart) {
        this.showSpinner = true;
      }
      if (event instanceof NavigationEnd || event instanceof NavigationCancel) {
        this.showSpinner = false;
      }
    });
  }

  ngOnInit(): void {
    this.storageMap.watch('id').subscribe((id: string) => {
      this.id = id;
    });
  }
}
```



وفي ملف app.component.html، نقوم بالتعديل التالي:

ملف app.component.html

```
<nav class="navbar navbar-expand-sm navbar-dark bg-dark Shadow">
  <div class="navbar-brand-two" href="#">
    
  </div>
  <span class="navbar-brand">Routing</span>
  <div class="justify-content-left">
    <ul class="navbar-nav">
      <li class="nav-item">
        <a
          routerLink="/home"
          routerLinkActive="is-active"
          [routerLinkActiveOptions]="{ exact: true }"
          ><span>Home</span></a>
      </li>
      <li class="nav-item">
        <a routerLink = "home/edite-courses" routerLinkActive="is-active"
```

```

    ><span>Edit Courses</span></a>
  >
</li>
<li class="nav-item">
  <a routerLink="/users" routerLinkActive="is-active"
    ><span>Users</span></a>
  >
</li>
<li class="nav-item">
  <a [routerLink]="['profile', id]" routerLinkActive="is-active"
    ><span>Profile</span></a>
  >
</li>
<li class="nav-item">
  <a routerLink="/auth/login" routerLinkActive="is-active"
    ><span>Login</span></a>
  >
</li>
<li class="nav-item">
  <a routerLink="/auth/signup" routerLinkActive="is-active"
    ><span>Signup</span></a>
  >
</li>
<li class="nav-item">
  <a routerLink="/auth/signout" routerLinkActive="is-active"
    ><span>Signout</span></a>
  >
</li>
</ul>
</div>
</nav>

<div class="spinner" *ngIf="showSpinner"></div>

<router-outlet></router-outlet>

```

أما في ملف app.component.scss، فنضيف الكود التالي:

```

app.component.scss ملف
.Shadow {
  box-shadow: 0 0 0 1.2px grey;
}

.is-active {
  background-color: gray;
  font-weight: bold;
  border: 1px solid rgb(27, 26, 26);
}

.is-active span {
  border-bottom-style: solid;
  border-bottom-width: 3px;
  border-bottom-color: #84ff6e;
  padding-bottom: 4px;
}

li a {
  color: white;
  padding: 10px;
  margin: 4px;
}

li a:hover {

```



```

text-decoration: none;
}

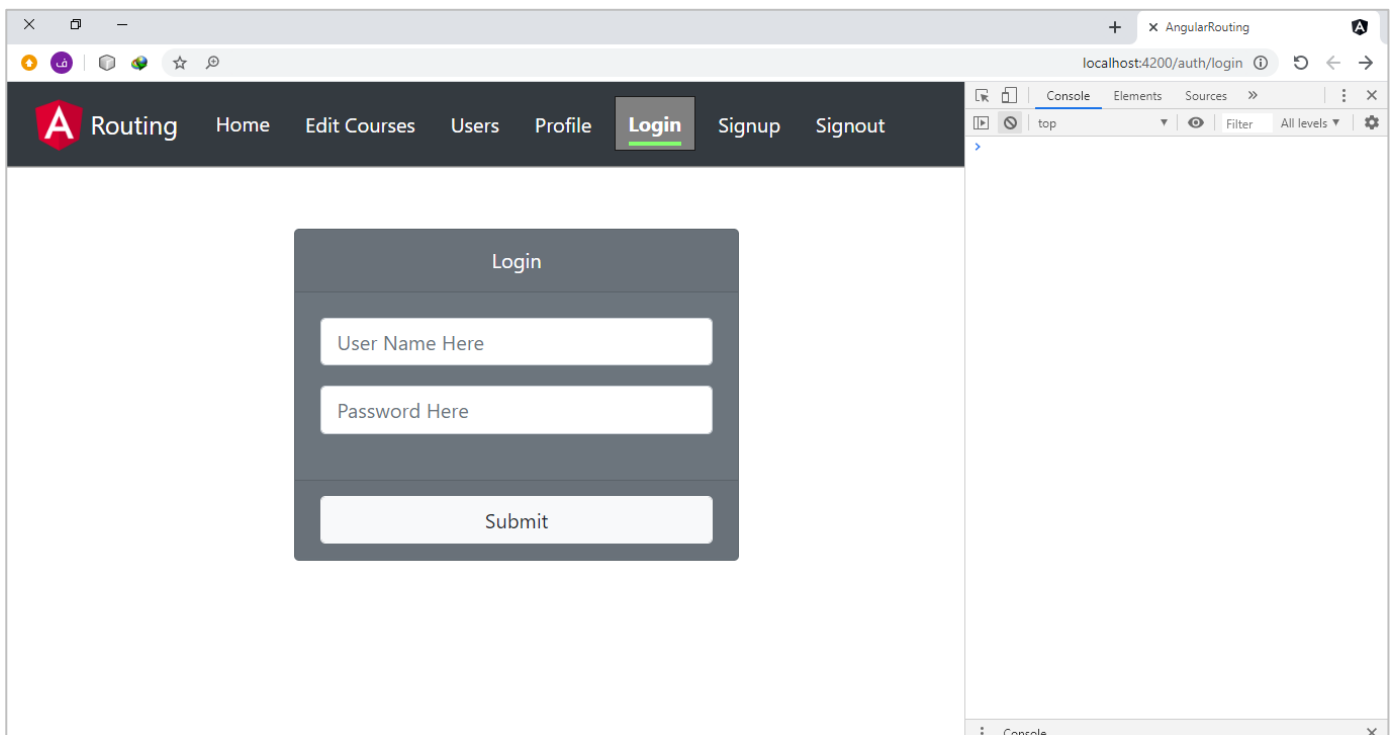
.spinner {
  position: fixed;
  width: 100%;
  left: 0;
  right: 0;
  top: 0;
  bottom: 0;
  background-color: rgba(240, 240, 240, 0.50);
  z-index: 1;
}

.spinner::after {
  content: '';
  display: block;
  position: absolute;
  border: 16px solid silver;
  border-top: 16px solid #337AB7;
  border-radius: 50%;
  width: 80px;
  height: 80px;
  animation: spin 700ms linear infinite;
  top: 50%;
  left: 50%;
}

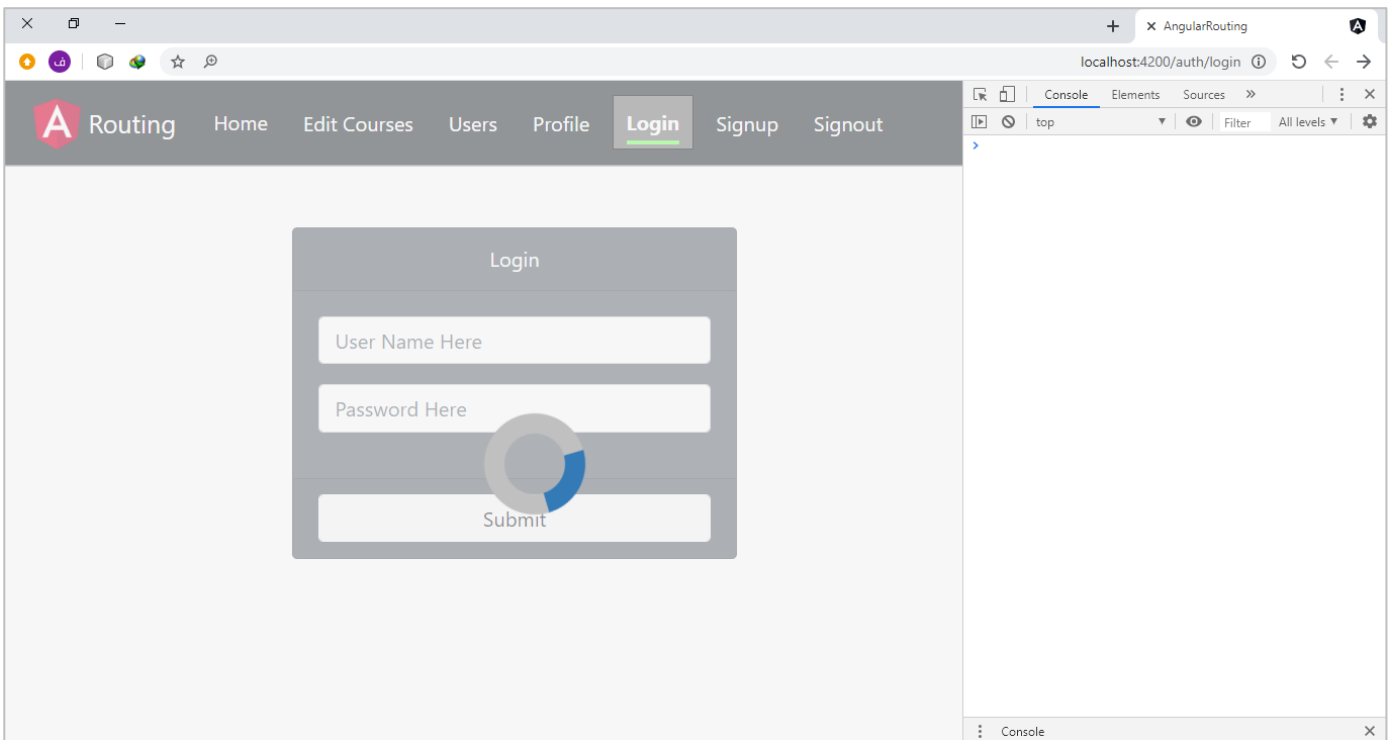
@keyframes spin {
  0% {transform: rotate(0deg)}
  100% {transform: rotate(360deg)}
}

```

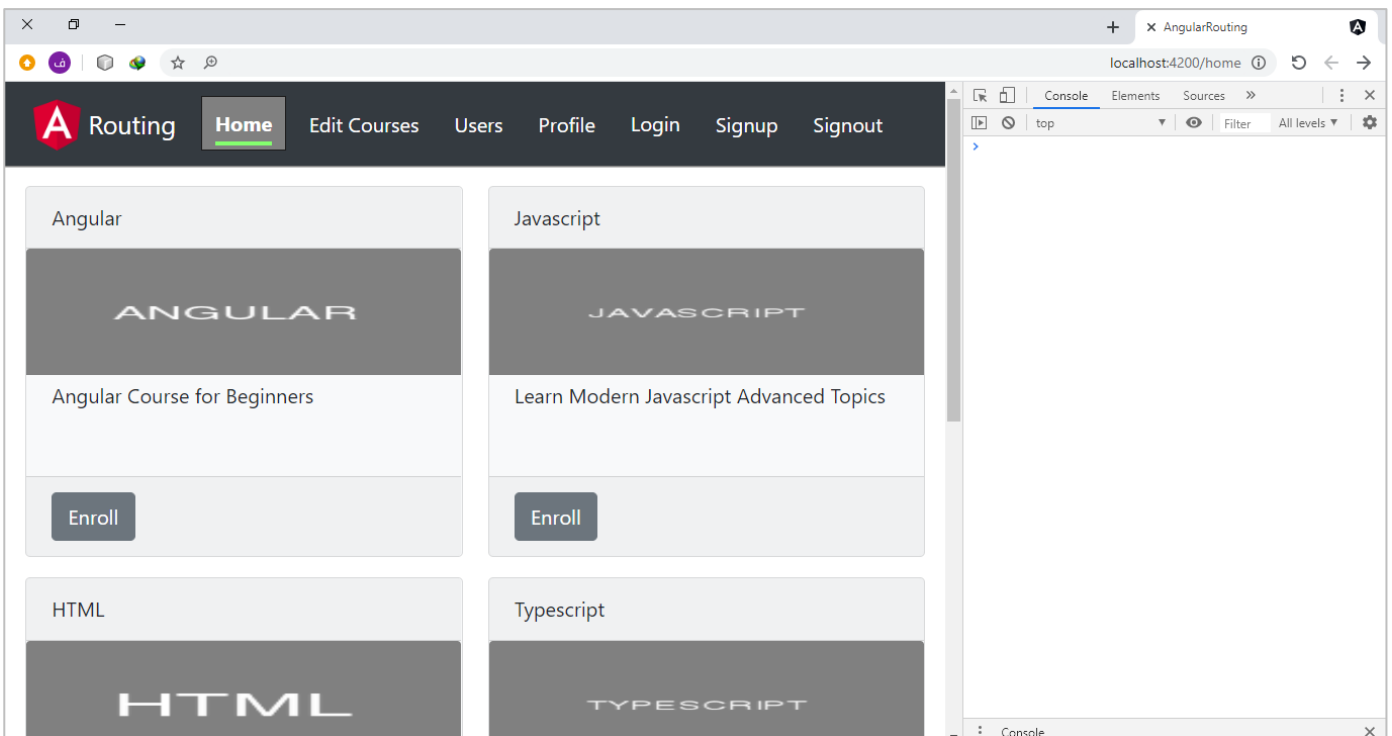
الآن لنحفظ التعديلات، ونذهب إلى المتصفح لنرى النتيجة:



في البداية لنضغط على تبويب Login، ومن ثم نضغط على تبويب Home:



نلاحظ أنه ظهر لنا spinner وعند الانتهاء وجلب كافة البيانات سوف يختفي وتظهر لنا صفحة home:



وبما ان هذا spinner هو عام لجميع الانتقالات حيث يظهر في كل مرة يحدث Navigation Start لأي route في النظام ويختفي في حال Navigation End، لذلك سوف نلغي spinner الذ قمنا بإنشائه سابقاً ونكتفي بهذا فقط، كالتالي:

ملف singout.component.html

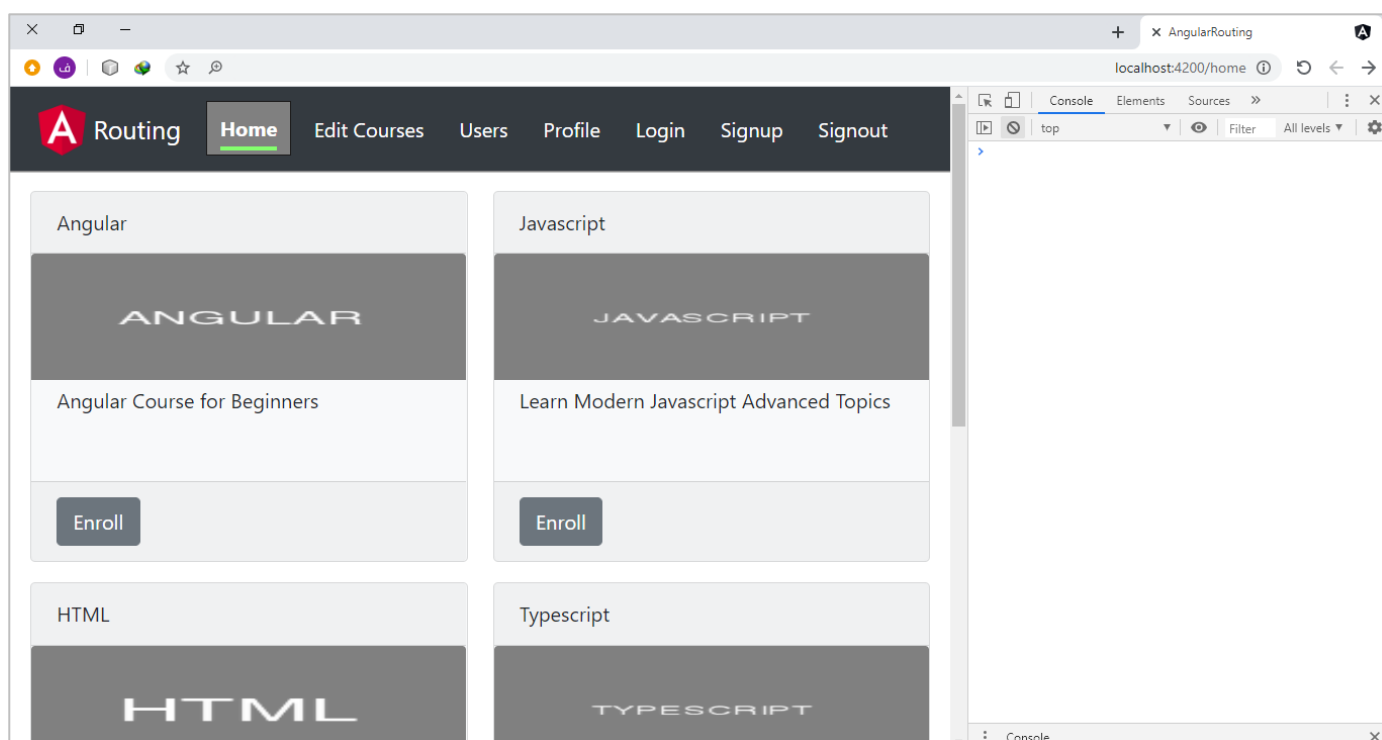
```
<h3 class="centered">please wait ...</h3>
```

وفي ملف `singout.component.scss`، نقوم بالتعديل التالي:

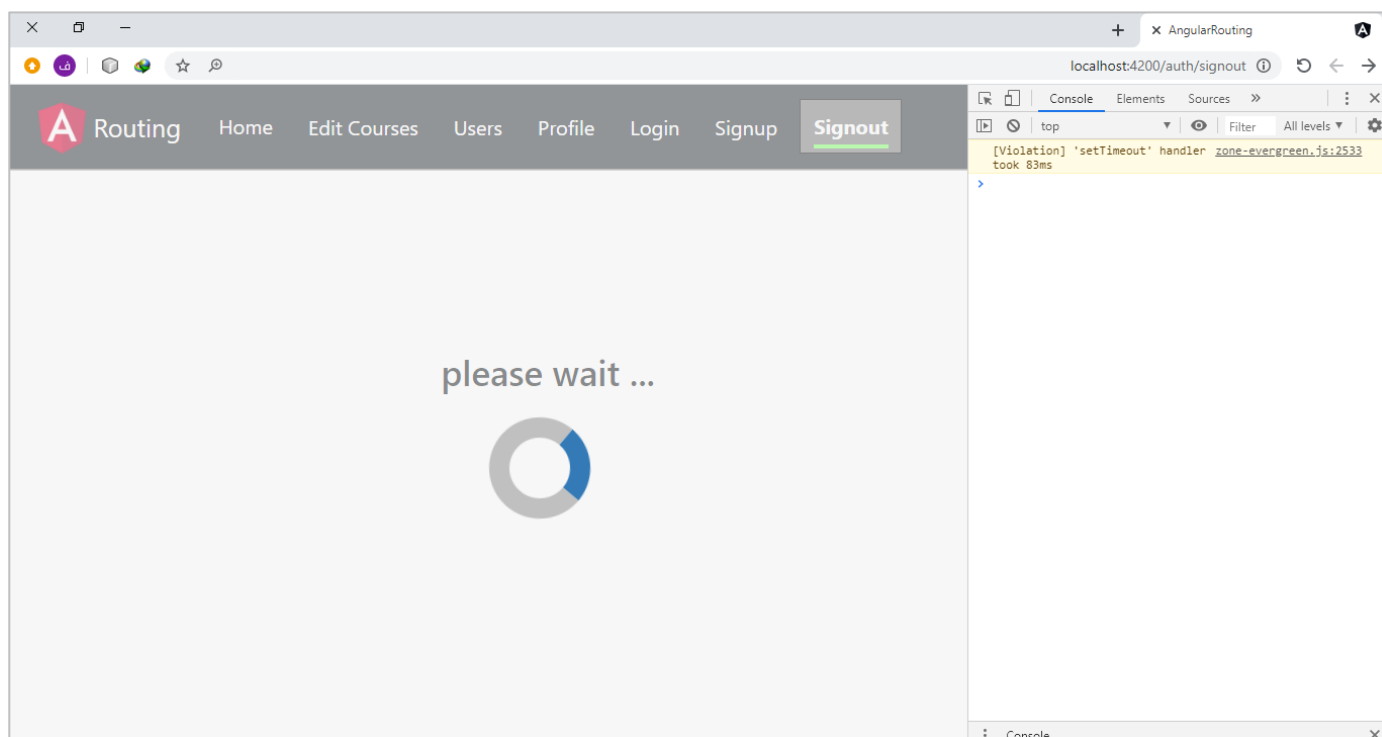
ملف `singout.component.scss`

```
.centered {position: fixed;top: 40%;left: 45%;}
```

الآن لنحفظ التعديلات، ونذهب إلى المتصفح لنرى النتيجة:

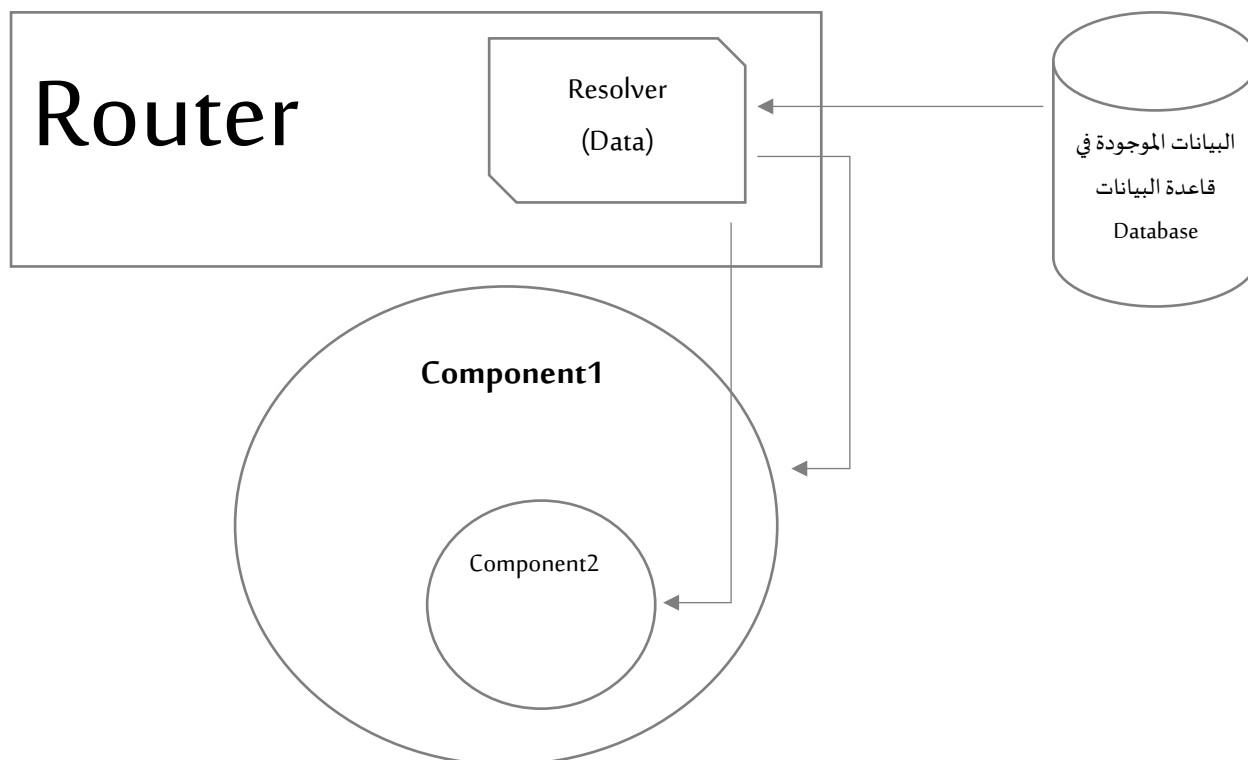


الآن لنختار التبويب Singout، كالتالي:



كما قلنا سابقاً ان من فوائد Resolver هي مشاركة البيانات مع component الأبناء، وبما اننا عملنا Resolver لcomponent الاب ذو الاسم home لذلك سوف نعمل مشاركة لهذه البيانات مع component الابن Edit Courses،

لأنهم يقرأون نفس البيانات ويعرضونها ولكن بطرق مختلفة لذلك سوف نقوم بمشاركة هذا Resolver مع component الابن، ومن فوائد هذه الطريقة أن كلا components الاب والابن يأخذون البيانات من Resolver واحد وأي تغيير في البيانات الموجودة على هذا Resolver سوف يؤثر على البيانات في كلا components، كما في الشكل التالي:



الآن لنذهب إلى ملف edit-courses.component.ts ونجري التعديلات التالية:

ملف edit-courses.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { Courses } from 'src/app/courses-data/courses';
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-edit-courses',
  templateUrl: './edit-courses.component.html',
  styleUrls: ['./edit-courses.component.scss']
})

export class EditCoursesComponent implements OnInit {
  courses$: Observable<Courses[]>;

  constructor(private route: ActivatedRoute) { }

  ngOnInit(): void {
    this.courses$ = this.route.parent.snapshot.data.courses;
  }
}
```

نلاحظ أننا بدلاً من أن نقرأ البيانات من service مباشرة، أصبحنا نقرأها من resolver الخاص بالآب.

ولو قمنا بالتجربة فسوف تظهر لنا نفس البيانات بدون أي مشاكل.

ولنعطي مثال آخر على Resolver، وليكن على صفحة user details لعرض بيانات محددة، عن طريق البارامتر id، ونقوم بهذا الامر كالتالي:

نشأ service جديدة، ونعمل بها كما عملنا بـ service الأولى وليكن اسمها resolve-user-details، كالتالي:


ملف resolve-user-details.service.ts

```
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, Resolve, RouterStateSnapshot } from '@angular/router';
import { Observable } from 'rxjs';
import { Users } from '../users-data/users';

@Injectable({
  providedIn: 'root'
})
export class ResolveUserDetailsService implements Resolve<Users> {

  constructor() { }

  resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<Users> {
  }
}
```



الفرق الوحيد هنا اننا قمنا بإضافة بارامترات كما كان موجود في الأنواع السابقة، الآن لنضيف الكود لقراءة البارامتر id من الرابط كما كنا نفعل سابقاً ونستطيع ان نوصل له عن طريق البارامتر route، كالتالي:

ملف resolve-user-details.service.ts

```
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, Resolve, RouterStateSnapshot } from '@angular/router';
import { Observable } from 'rxjs';
import { delay } from 'rxjs/operators';
import { Users } from '../users-data/users';
import { UsersService } from '../users-data/users.service';

@Injectable({
  providedIn: 'root'
})
export class ResolveUserDetailsService implements Resolve<Users> {

  constructor(private usersService: UsersService) { }

  resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<Users> {
    const id = +route.paramMap.get('id');
    return this.usersService.getUserById(id).pipe(delay(2000));
  }
}
```

الآن لنضيفه إلى تهيئة Routing وبالتحديد لـ routed ذو الاسم user-details، كالتالي:

ملف app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { UsersListComponent } from './users/users-list/users-list.component';
import { UserDetailsComponent } from './users/user-details/user-details.component';
import { AuthenticationComponent } from './authentication/authentication.component';
import { LoginComponent } from './authentication/login/login.component';
```

```

import { SignupComponent } from './authentication/signup/signup.component';
import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
import { SingoutComponent } from './authentication/singout/singout.component';
import { ProfileComponent } from './profile/profile.component';
import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
import { CanActivateGuard, CanActivateAdminGuard } from './guards/can-activate.guard';
import { CanActivateAdminChildGuard, CanActivateChildGuard } from './guards/can-activate-child.guard';
import { CanDeactivateGuard, CanDeactivateLoginGuard } from './guards/can-deactivate.guard';
import { ResolveCoursesService } from './guards/resolve-courses.service';
import { ResolveUserDetailsService } from './guards/resolve-user-details.service';

const routes: Routes = [
  {
    path: 'home',
    component: HomeComponent,
    resolve: { courses: ResolveCoursesService },
    canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
    children: [
      { path: 'edit-courses', component: EditCoursesComponent }
    ]
  },
  { path: 'profile/:id', canActivate: [CanActivateGuard], component: ProfileComponent },
  {
    path: 'users',
    canActivate: [CanActivateGuard, CanActivateAdminGuard],
    data: {
      roles: 'admin'
    },
    children: [
      { path: '', component: UsersComponent },
      { path: 'user-list', component: UsersListComponent },
      {
        path: 'user-details/:id',
        resolve: {
          userDetails: ResolveUserDetailsService
        },
        component: UserDetailsComponent
      }
    ]
  },
  {
    path: 'auth', children: [
      { path: '', component: AuthenticationComponent },
      { path: 'login', canDeactivate: [CanDeactivateLoginGuard], component: LoginComponent },
      { path: 'signup', canDeactivate: [CanDeactivateGuard], component: SignupComponent },
      { path: 'signout', component: SingoutComponent }
    ]
  },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: '**', component: NotFoundPageComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```



وبعد وضع Routed Resolver المستهدف، الآن لنذهب إلى ملف user-details.component.ts، ونقرأ البيانات من هذا Resolver لعرضها في component، كالتالي:

ملف user-details.component.ts

```
import { Component, OnInit, OnDestroy, AfterViewChecked } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { UsersService } from 'src/app/users-data/users.service';
import { Subscription } from 'rxjs';
import { UsersResolved } from 'src/app/users-data/users';

@Component({
  selector: 'app-user-details',
  templateUrl: './user-details.component.html',
  styleUrls: ['./user-details.component.scss']
})
export class UserDetailsComponent implements OnInit, OnDestroy, AfterViewChecked {
  user: UsersResolved;
  private subscription: Subscription;
  private id: number;
  private usersSum: number;

  constructor(
    private activeRouter: ActivatedRoute,
    private usersService: UsersService,
    private router: Router
  ) {
    this.usersService.getAllUsers().subscribe( data => {
      this.usersSum = data.length;
    });
  }

  ngOnInit() {
    this.subscription = this.activeRouter.data.subscribe(data => {
      this.user = data.userDetails;
      this.id = +this.user.users.userId;
    });
  }

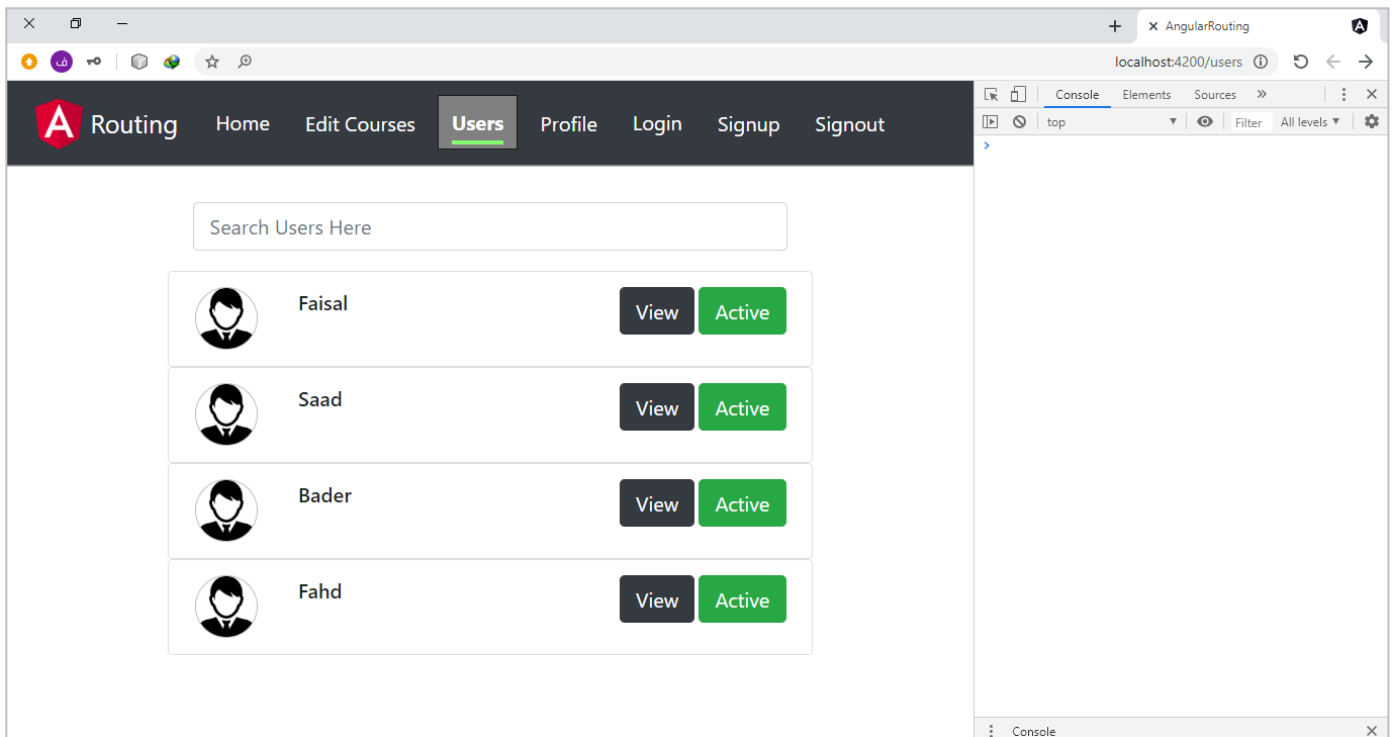
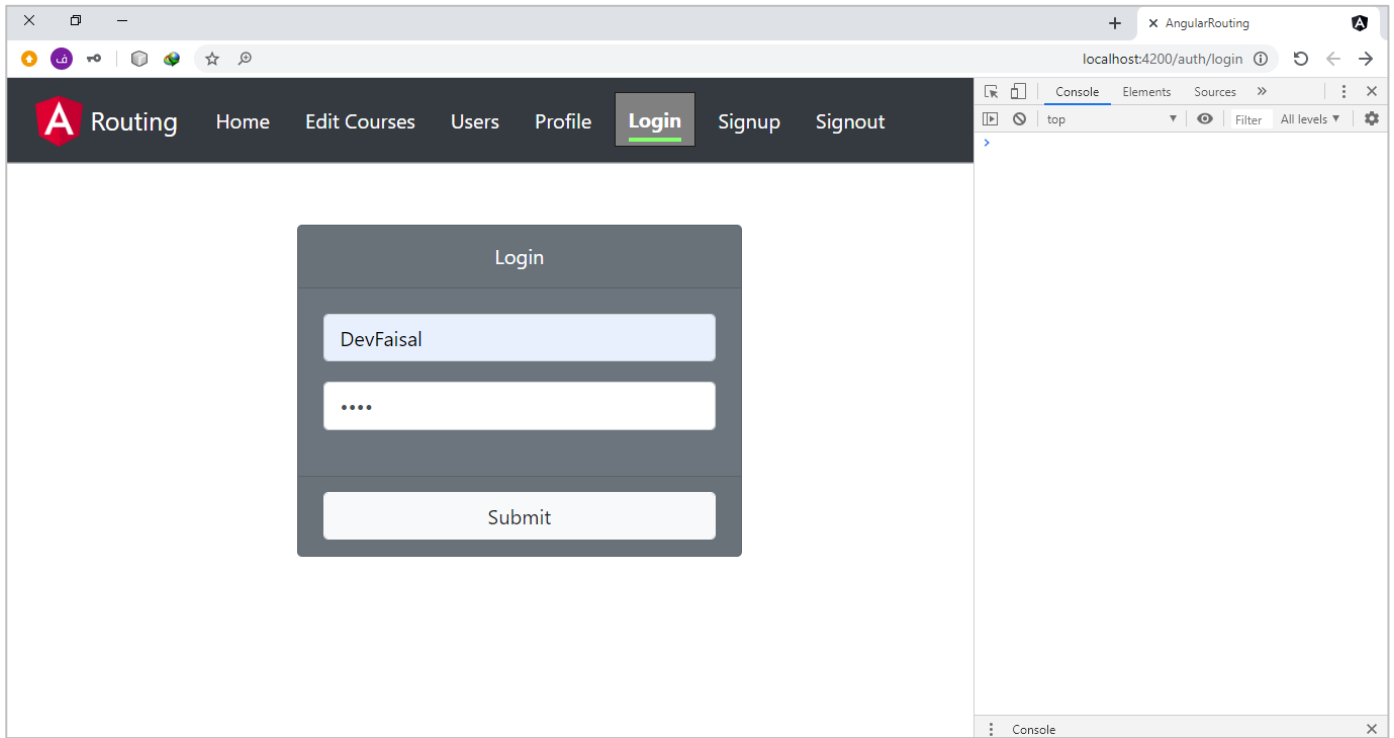
  ngAfterViewChecked() {
    const fragmentParam = this.activeRouter.snapshot.fragment;
    if (fragmentParam) {
      const elementId = document.getElementById(fragmentParam);
      elementId.scrollIntoView();
    }
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }

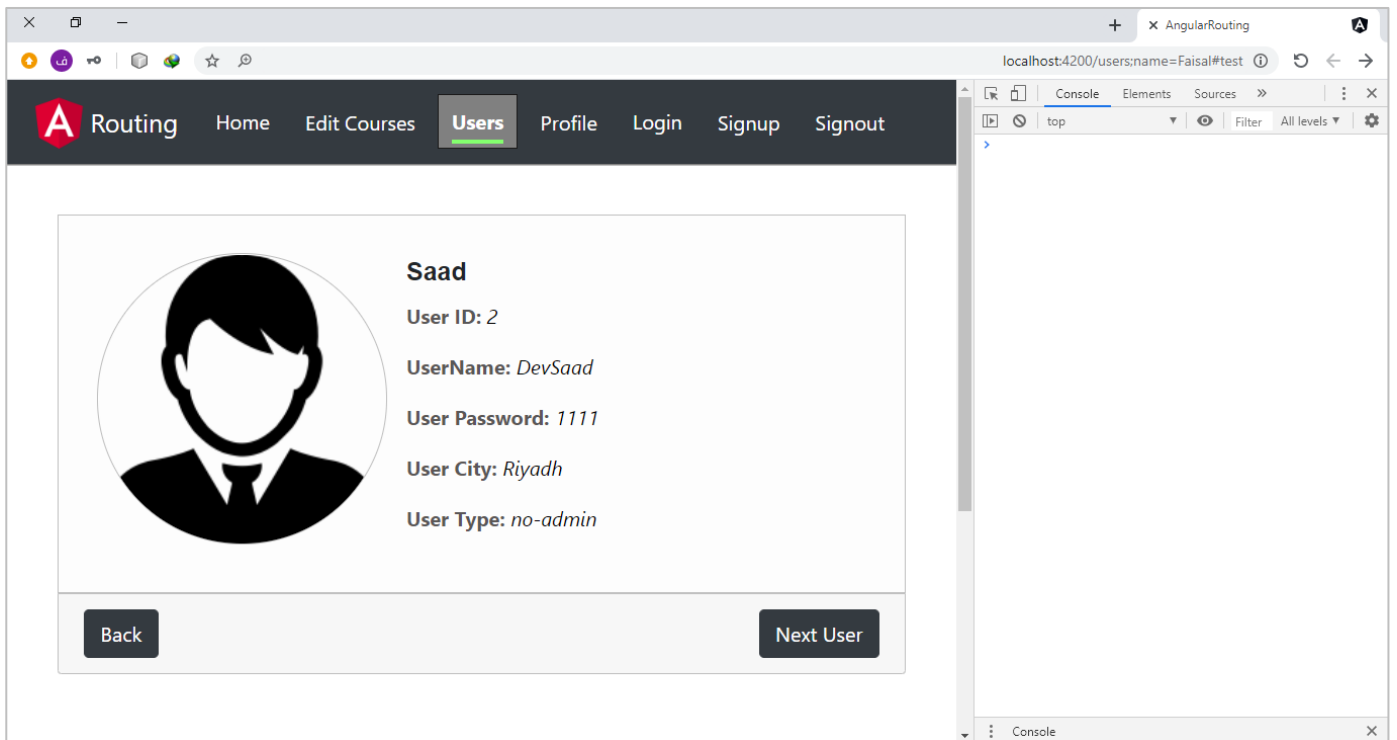
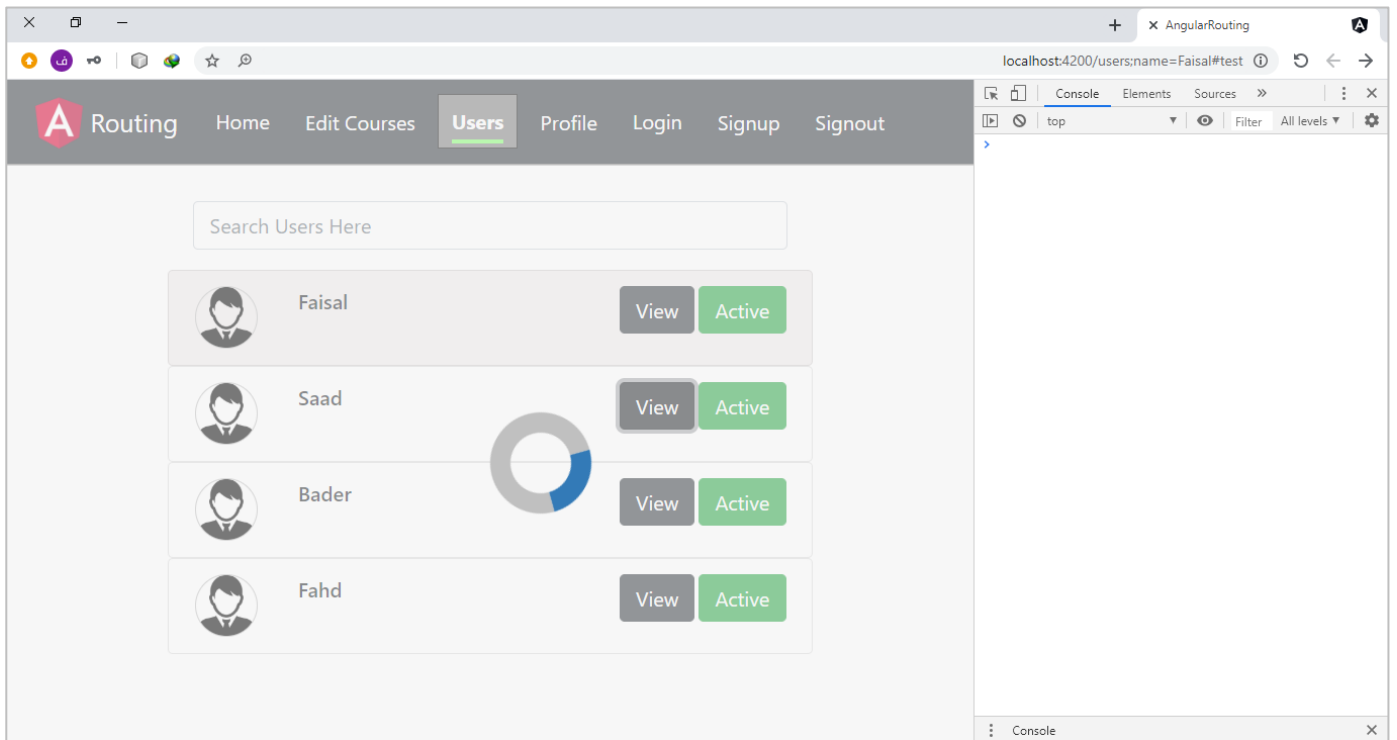
  viewNextUser() {
    this.id === this.usersSum ? this.id = 1 : this.id += 1;
    this.router.navigate(['users/user-details', this.id], {
      queryParamsHandling: 'preserve'
    });
  }
}
```

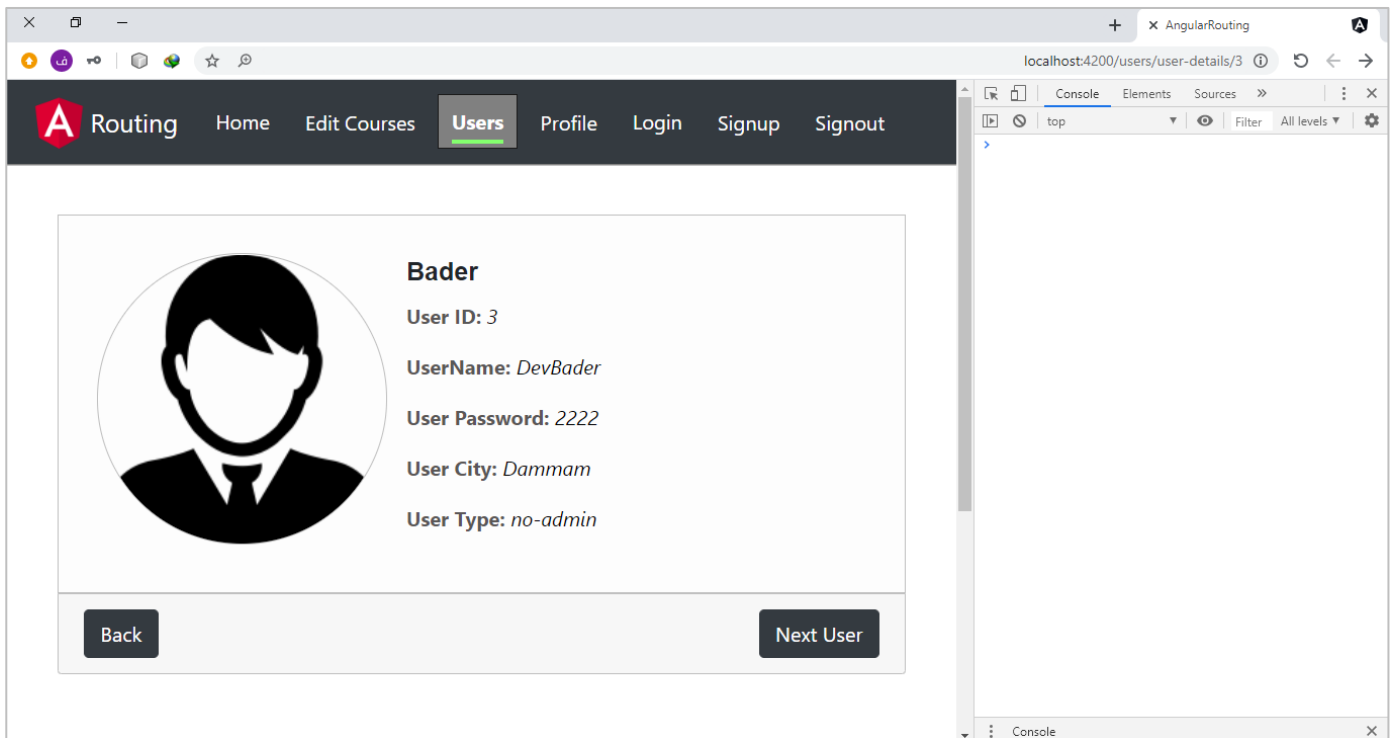
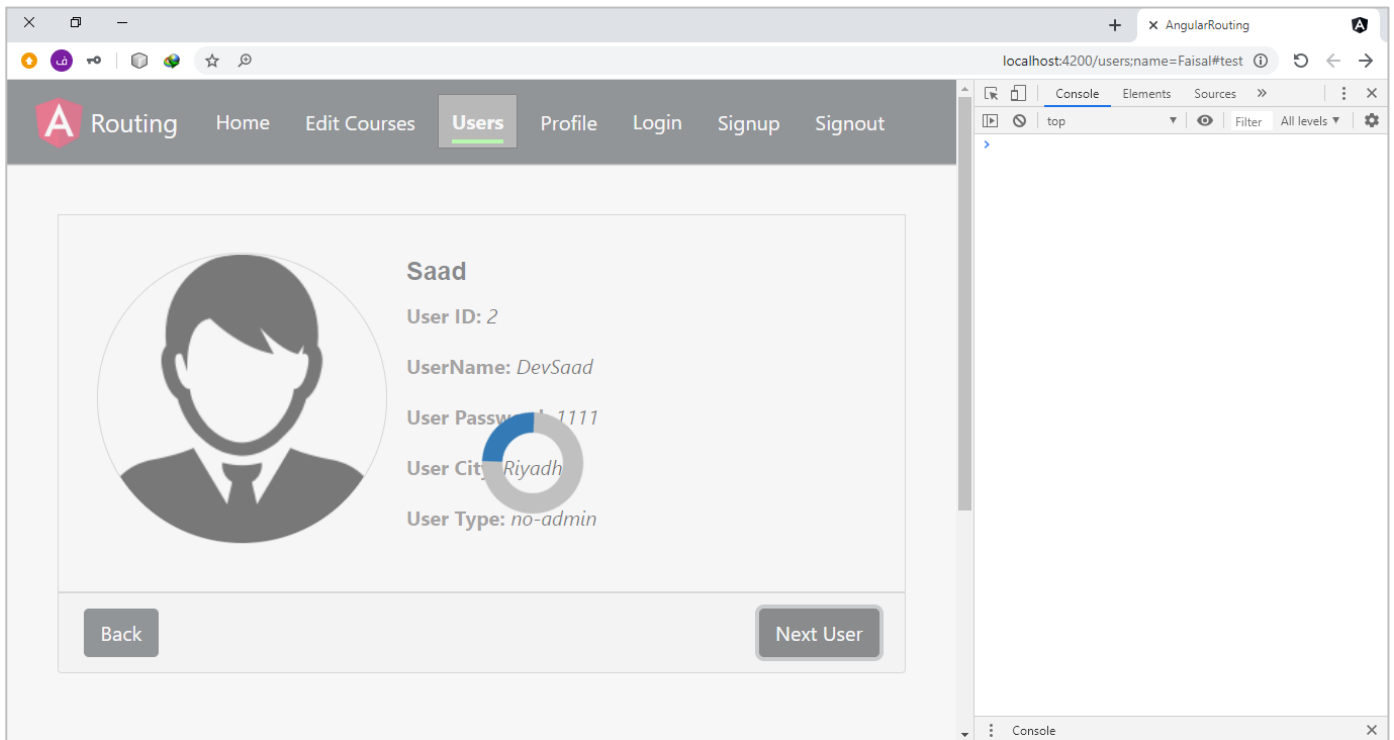
في هذا الملف أجرينا مجموعة من التعديلات، أولها اتصلنا في البيانات عن طريق service لكي نحصل على عدد المستخدمين ونخزنها في متغير، والنقطة رقم 2 هي المهمة حيث غيرنا الاتصال بدلاً من service لجلب البيانات، جلبناه من Resolver بعدما عملنا له subscribe بعكس المثال الأول الذي استخدمنا معه snapshot، والنقطة الثالثة والأخيرة قمنا بتعديل الانتقال الى user بحيث إذا وصل إلى نهاية users ينتقل إلى أول user.

الآن لنحفظ التعديلات ونذهب إلى المتصفح لنرى النتيجة:









وبذلك نكون أننا هذا الجزء من route guards، وهو resolve guard، وبقي علينا قبل الانتقال إلى نقطة الأولى في التنفيذ بين هذه الأنواع، ان نتكلم عن إخفاء وإظهار التبويبات بناء على الصلاحيات المخصصة للمستخدم، وهذا ما سوف اتطرق اليه في النقطة التالية.

### 7.3. إخفاء وإظهار التبويبات بناء على صلاحيات المستخدم:

فمثلاً إذا قام المستخدم بتسجيل الدخول فمن المنطقي ان نخفي تبويب زر الدخول ونظهر له تبويب singout اما اذا لم يقوم بتسجيل الدخول فنخفي هذا التبويب ونظهر تبويب login و signup، وهكذا في باقي التبويبات، ونستطيع عمل هذا

الامر بكل بساطة عن طريق المتغيرين isLogin\$ والمتغير isAdmin\$، حيث نستدعيهم في ملف app.component.ts وبما انهما Observable نعمل لهم subscribe، ونخزن القيم التي يُعيدونها في متغيرين، ونعمل bind لهذه المتغيرين لملف html لهذا component وبناءً على قيم هذه المتغيرين نُخفي ونظهر التبويبات، كالتالي:

ملف app.component.ts

```
import {ChangeDetectorRef, Component, OnInit} from '@angular/core';
import { UsersService } from '../users-data/users.service';
import { StorageMap } from '@ngx-pwa/local-storage';
import {Router,Event,NavigationStart,NavigationEnd,NavigationCancel} from '@angular/router';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {
  IsSignedIn = false;
  IsAdmin = false;
  id: string;
  showSpinner = false;
  constructor(
    private userService: UsersService,
    private storageMap: StorageMap,
    private router: Router,
  ) {
    this.router.events.subscribe((event: Event) => {
      if (event instanceof NavigationStart) {
        this.showSpinner = true;
      }
      if (event instanceof NavigationEnd || event instanceof NavigationCancel) {
        this.showSpinner = false;
      }
    });
  }

  ngOnInit(): void {
    this.userService.isAdmin$.subscribe(val => {
      this.IsAdmin = val;
    });
    this.userService.isLogin$.subscribe(val => {
      this.IsSignedIn = val;
    });
    this.storageMap.watch('id').subscribe((id: string) => {
      this.id = id;
    });
  }
}
```

وفي ملف app.component.html، نجري التعديلات التالية:

ملف app.component.html

```
<nav class="navbar navbar-expand-sm navbar-dark bg-dark Shadow">
  <div class="navbar-brand-two" href="#">
    
  </div>
  <span class="navbar-brand">Routing</span>
  <div class="justify-content-left">
    <ul class="navbar-nav">
      <li class="nav-item">
```

```

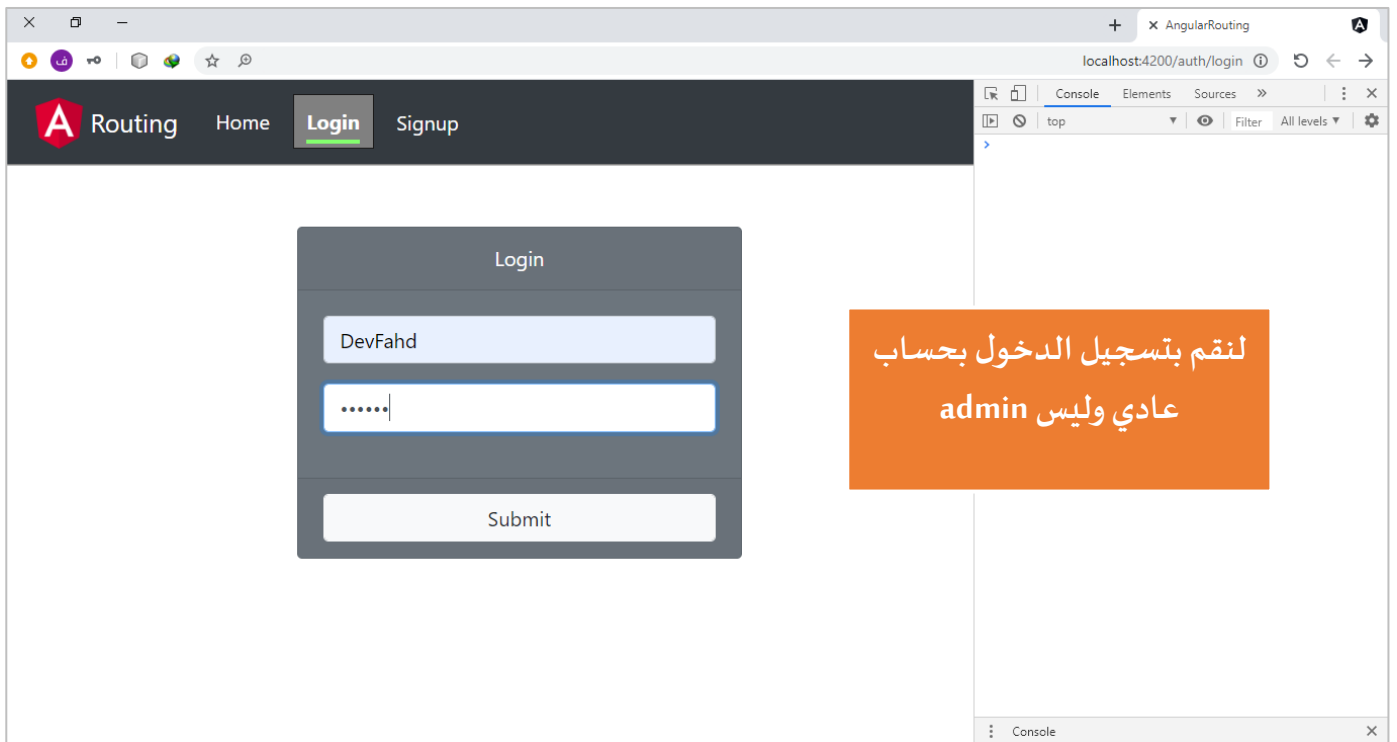
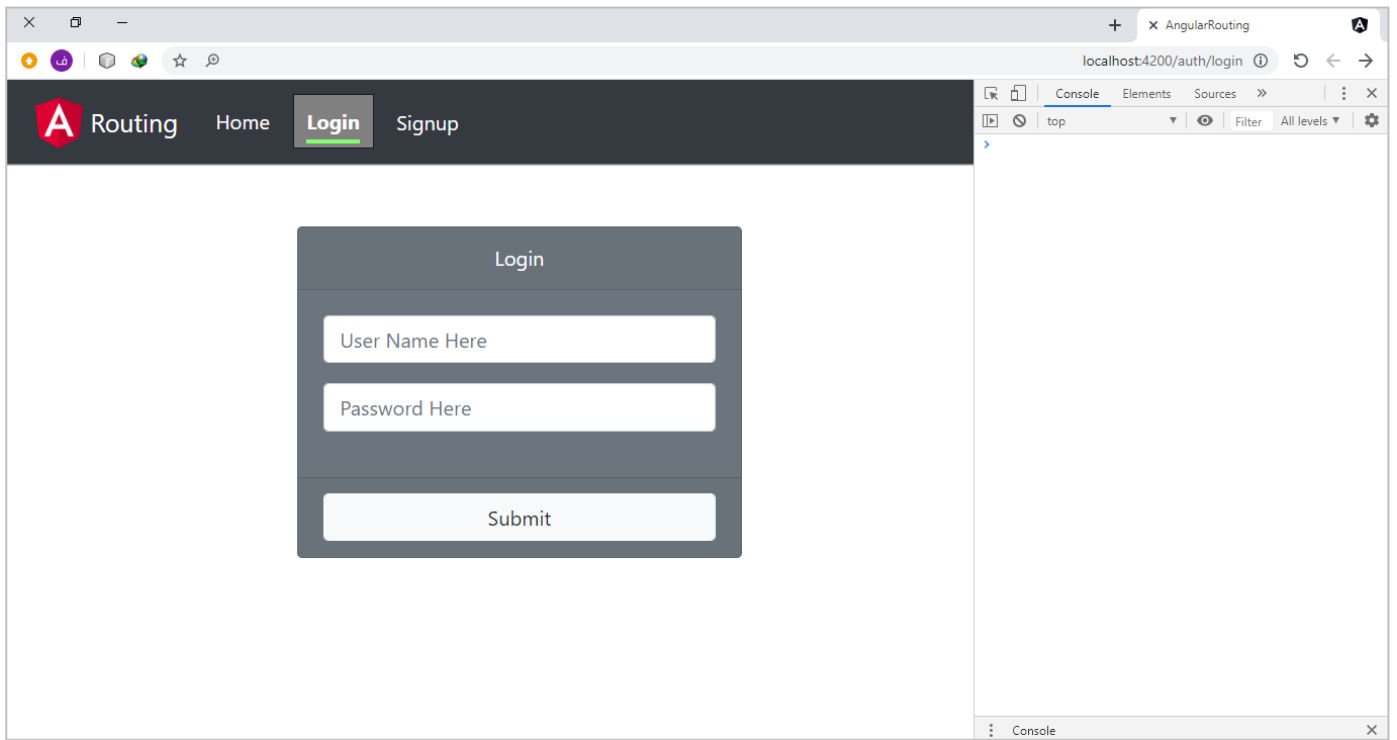
    <a
      routerLink="/home"
      routerLinkActive="is-active"
      [routerLinkActiveOptions]="{ exact: true }"
    ><span>Home</span></a>
  >
</li>
<li class="nav-item" *ngIf="IsAdmin && IsSignedIn">
  <a routerLink = "home/edit-courses" routerLinkActive="is-active"
    ><span>Edit Courses</span></a>
  >
</li>
<li class="nav-item" *ngIf="IsAdmin && IsSignedIn">
  <a routerLink="/users" routerLinkActive="is-active"
    ><span>Users</span></a>
  >
</li>
<li class="nav-item" *ngIf="IsSignedIn">
  <a [routerLink]="['profile', id]" routerLinkActive="is-active"
    ><span>Profile</span></a>
  >
</li>
<li class="nav-item" *ngIf="!IsSignedIn">
  <a routerLink="/auth/login" routerLinkActive="is-active"
    ><span>Login</span></a>
  >
</li>
<li class="nav-item" *ngIf="!IsSignedIn">
  <a routerLink="/auth/signup" routerLinkActive="is-active"
    ><span>Signup</span></a>
  >
</li>
<li class="nav-item" *ngIf="IsSignedIn">
  <a routerLink="/auth/signout" routerLinkActive="is-active"
    ><span>Signout</span></a>
  >
</li>
</ul>
</div>
</nav>

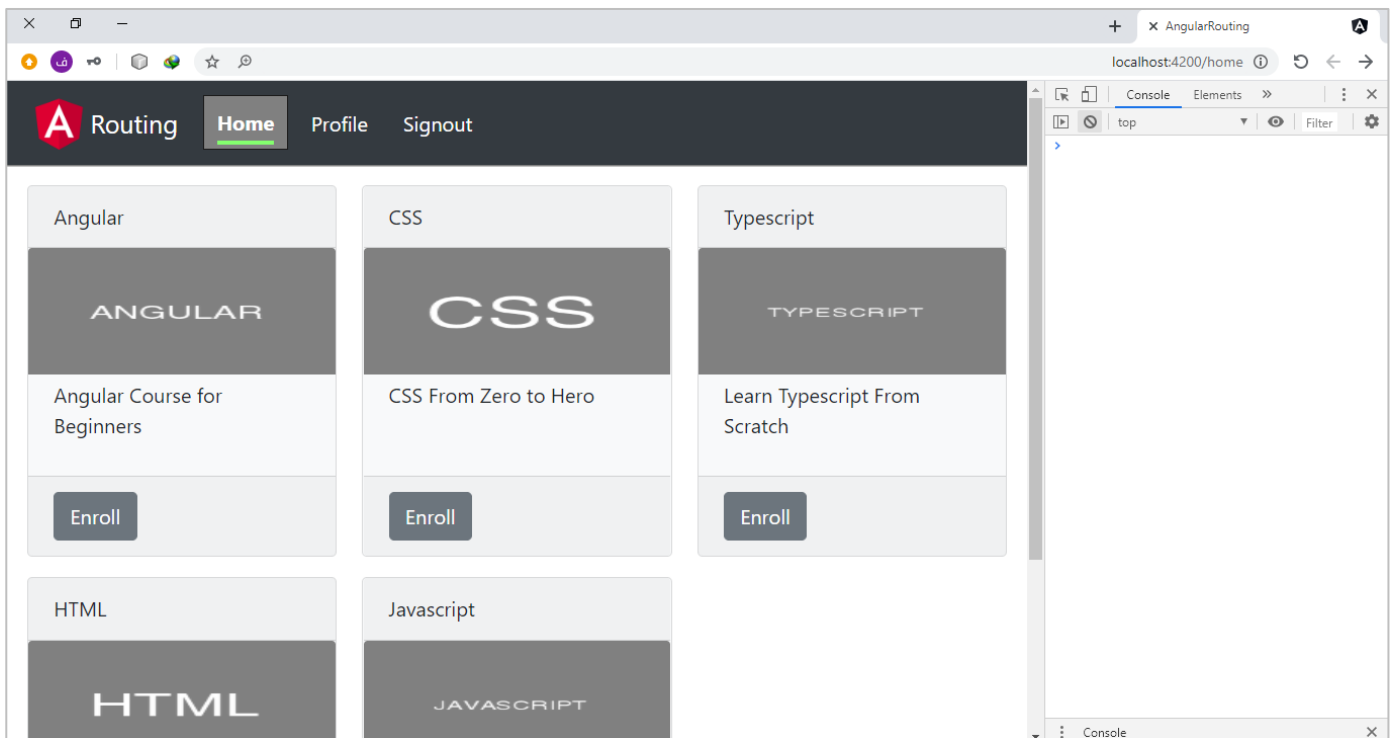
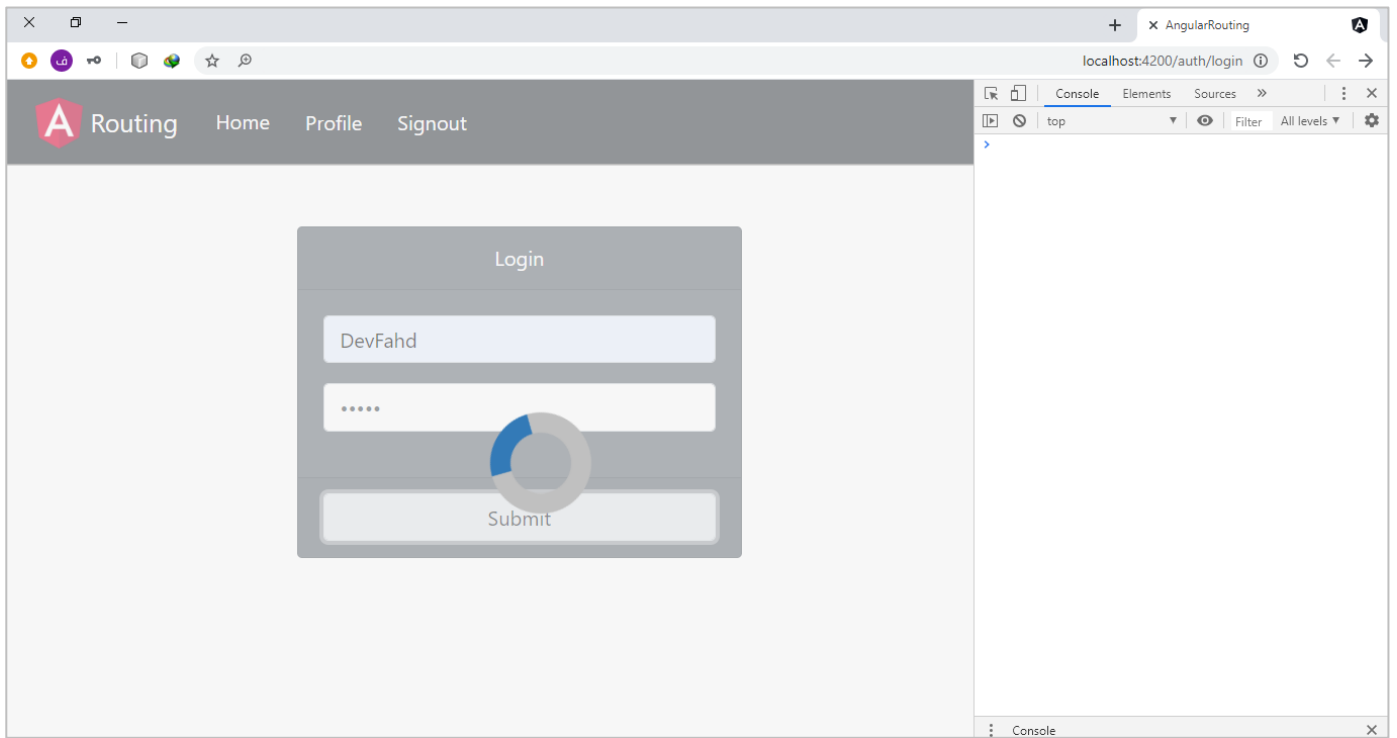
<div class="spinner" *ngIf="showSpinner"></div>

<router-outlet></router-outlet>

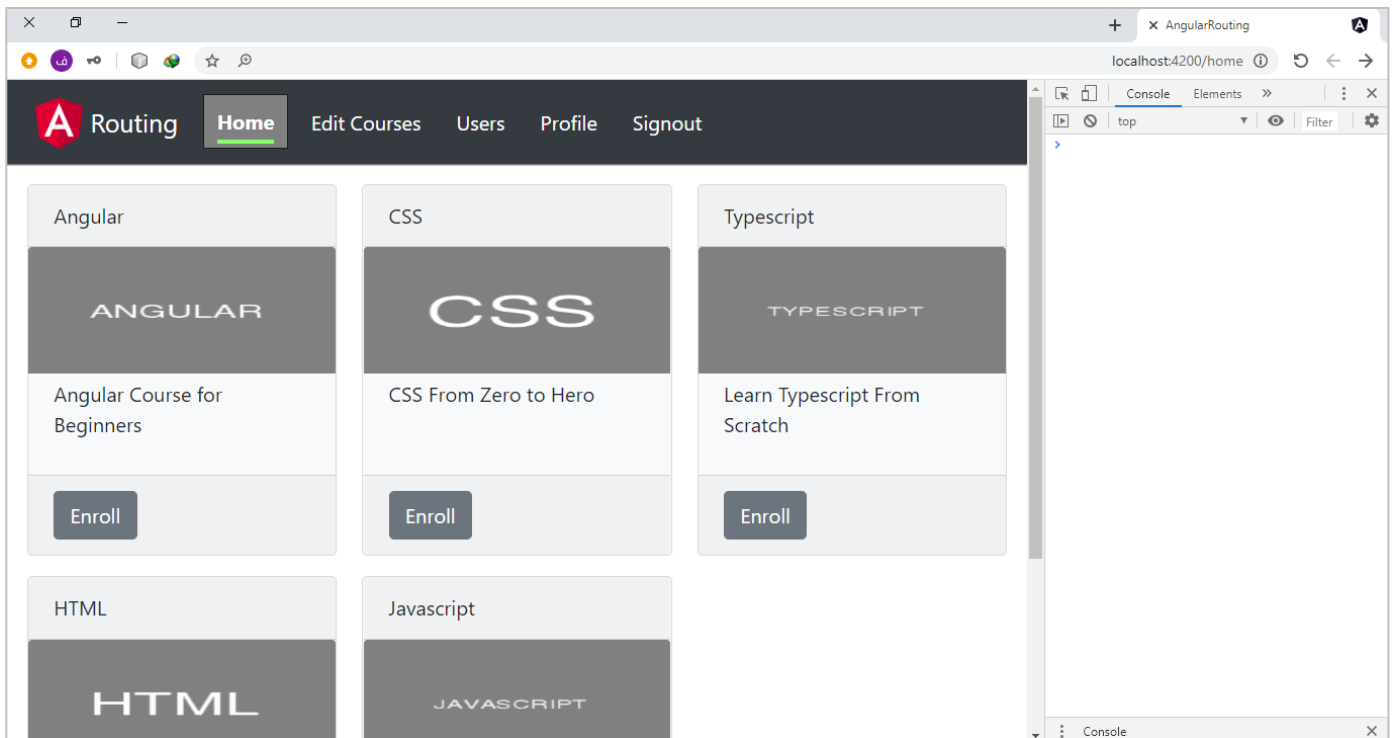
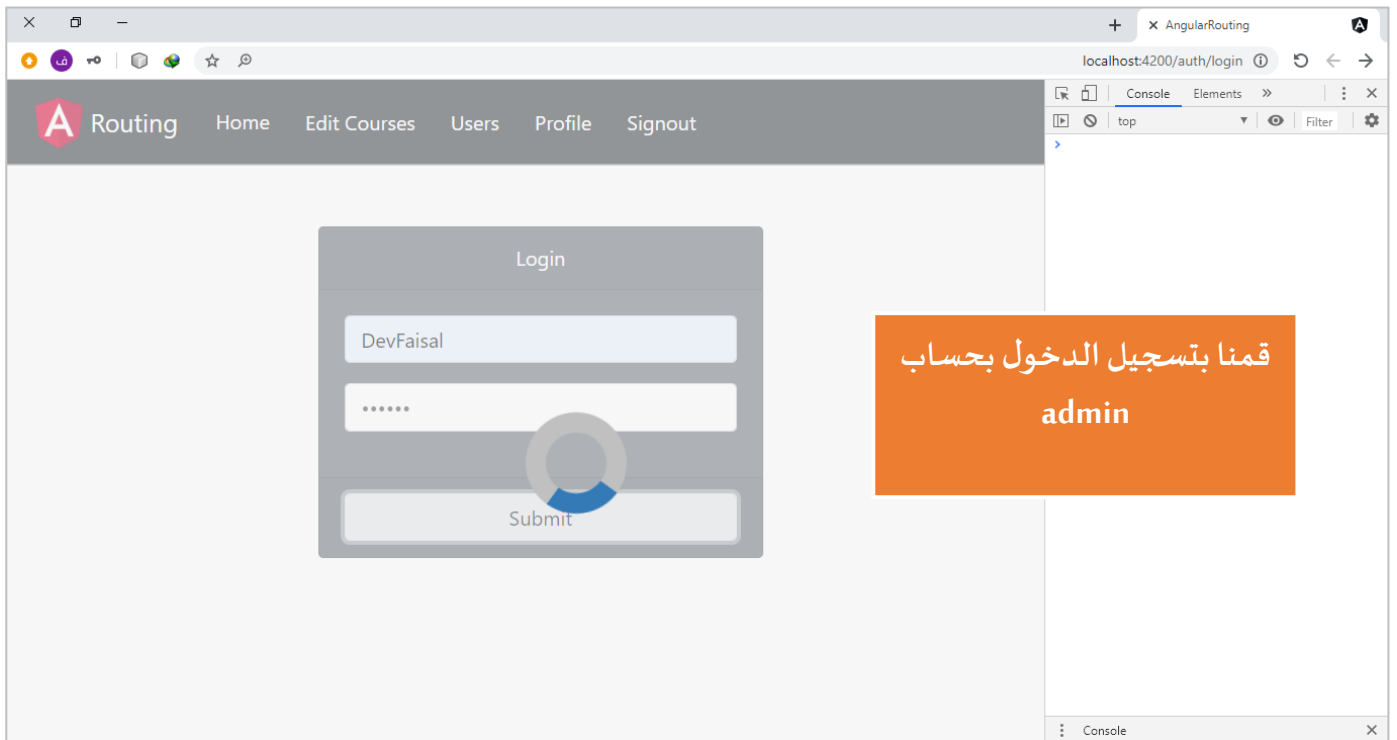
```

الآن لنحفظ التعديلات، ونذهب إلى المتصفح لنرى النتيجة:



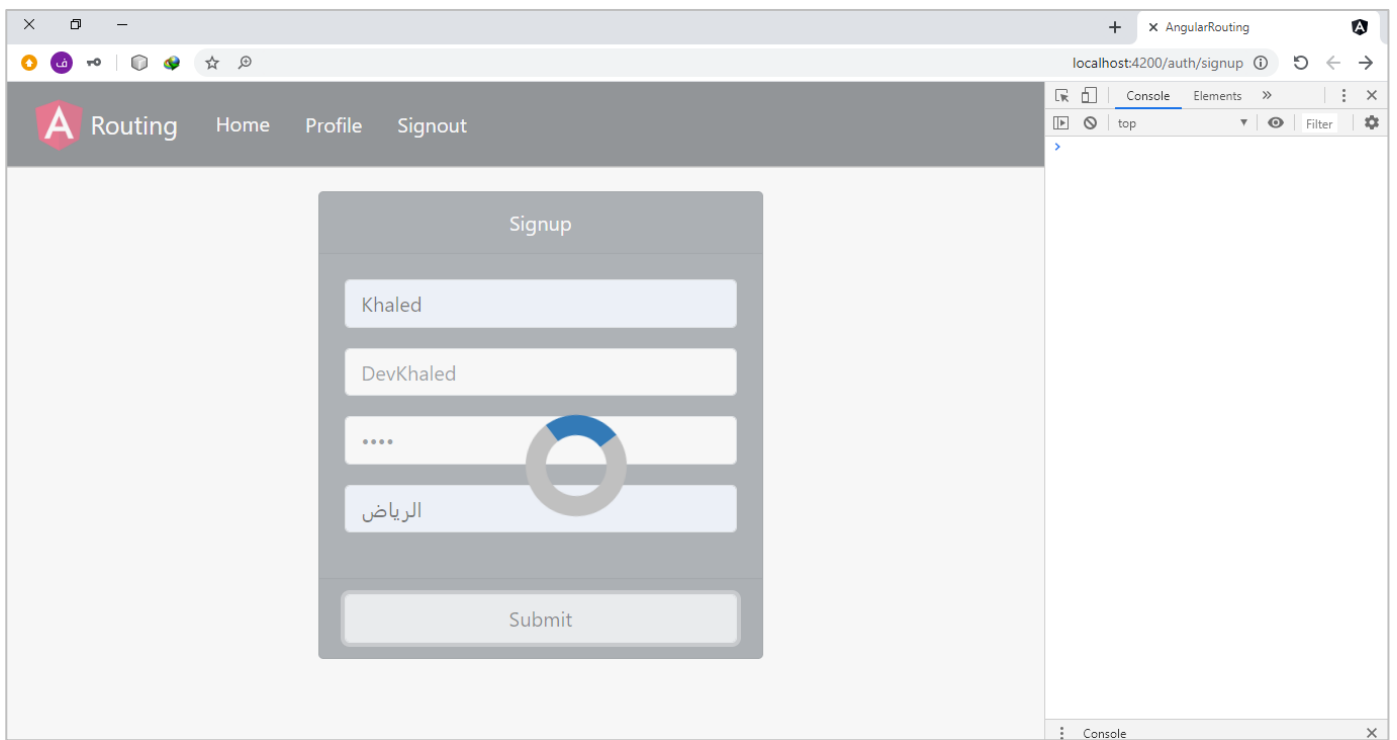
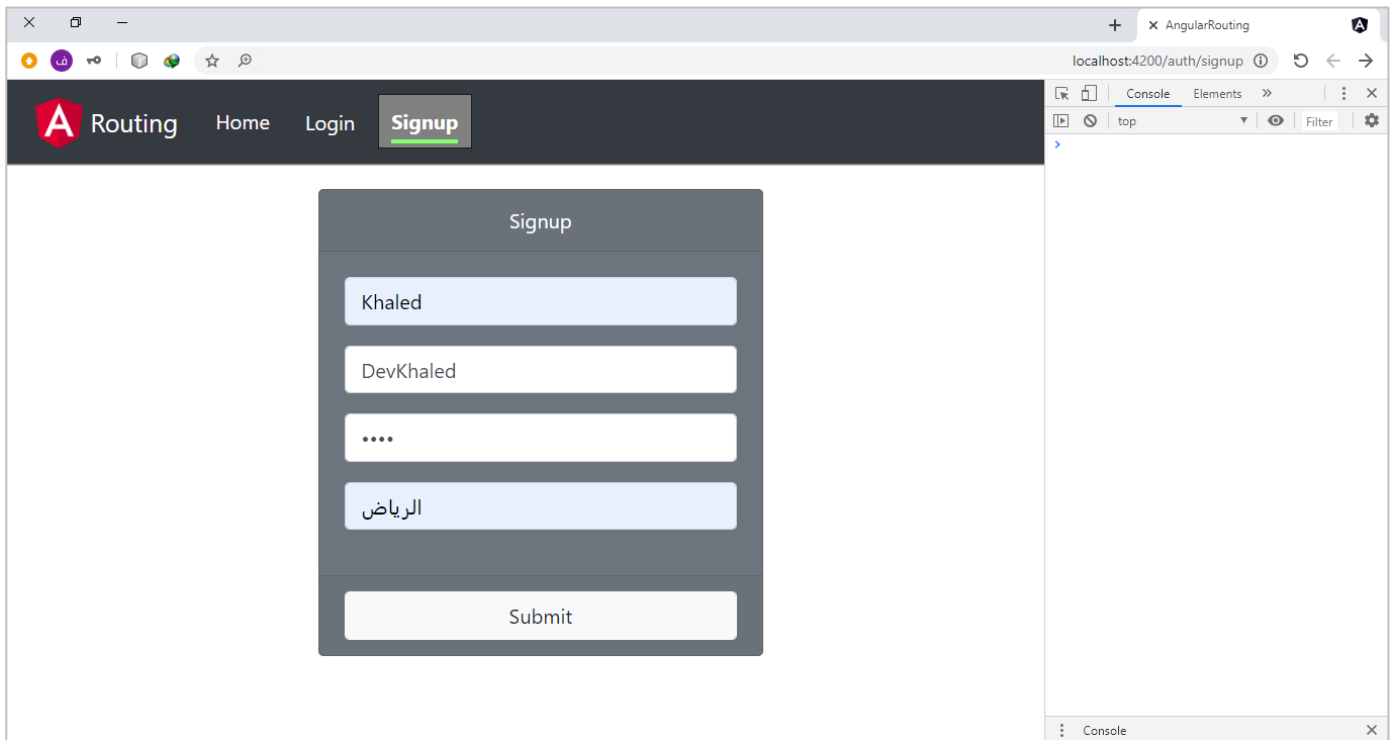


نلاحظ اخفى تبويب login و singup و اظهر signout بالإضافة إلى تبويب profile لأن هذا المستخدم ليس admin لذلك اخفى باقي التبويبات، الآن لنقم بتسجيل الخروج ومن ثم نقوم بتسجيل الدخول مرة أخرى بحساب admin، كالتالي:

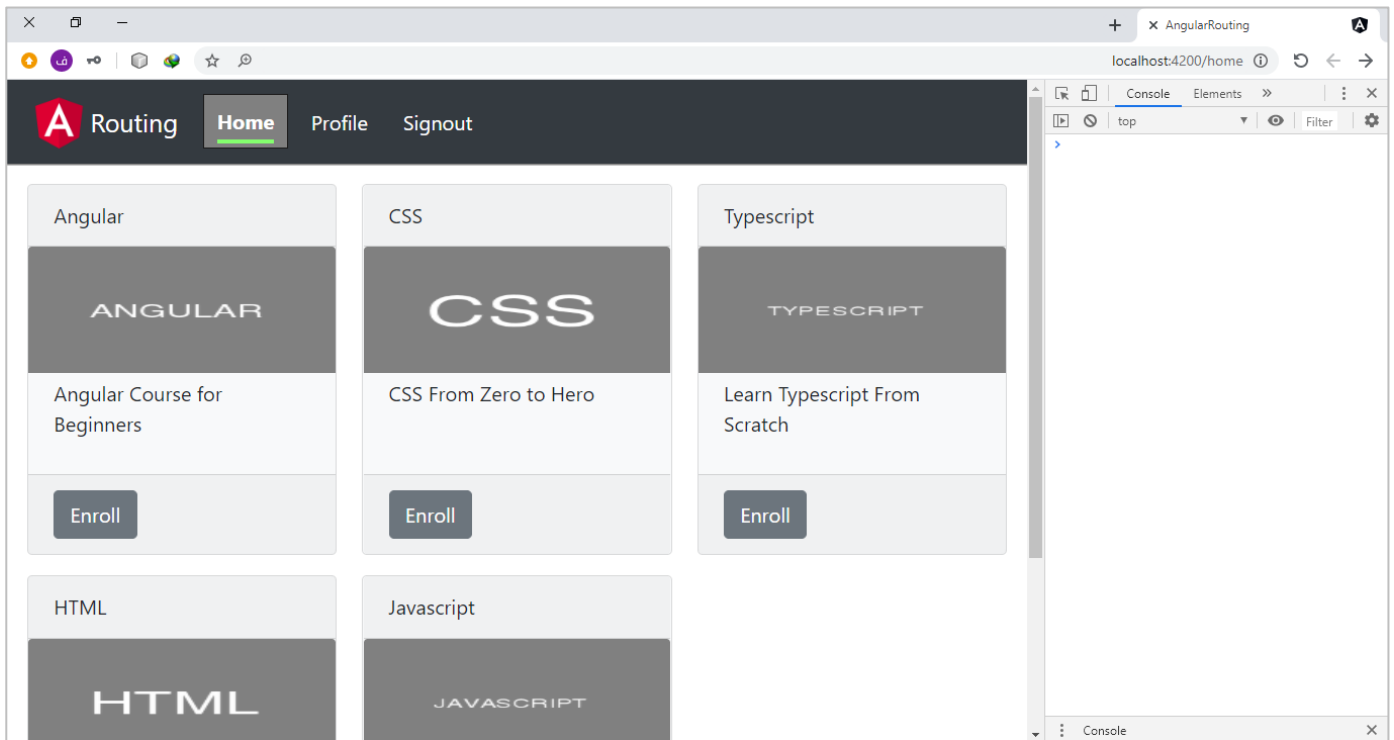


بعد التسجيل بحساب admin اظهر له الصفحات بناء على الصلاحيات المعطى له وهي الدخول على جميع الصفحات.

آخر خطوة لنقم بتسجيل الخروج ونعمل تسجيل مستخدم جديد، كالتالي:







نلاحظ أنه قام بتسجيل مستخدم جديد وبنفس الوقت عمل تسجيل دخول وجعل المستخدم نوعه عادي وليس admin وتم اظهار الصفحات بناء على الصلاحيات المخصصة له.

وبذلك نكون انهيينا هذا الجز وسوف نتكلم عن آخر نقطة في Route Guards وهي الأولوية في التنفيذ في حال كان لدينا أكثر من Guard لنفس Route او نفس نوع guard لنفس Route.

### 8.3. Route Guards Priority:

لابد عند تعاملك مع Route Guards بأنواعه المختلفة ان تعرف أولوية التنفيذ بين هذه الأنواع بمعنى من يتم تنفيذه قبل الآخر، ففي مثالنا هذا نلاحظ اننا استخدمنا أكثر من نوع لنفس Route وايضاً نفس النوع ولكن بكلاسات مختلفة لنفس Route، ومن هذا المنطلق معرفتك لأولوية التنفيذ تعطيك فهم اعمق متى وأين اضع Guard المناسب في Route المناسب.

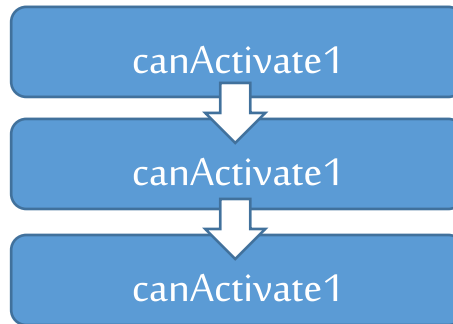
ونستطيع توضيح أولوية التنفيذ في Route Guards المختلفة من خلال الشكل التالي:



كما نلاحظ اول guard يتم تنفيذه هو canActivate والأخير هو resolve وبينهما تأتي الأنواع الأخرى.

أما من ناحية إذا كان لدينا نفس guard لنفس Route ولكن بكلاسات مختلفة ولنفرض ان هذا النوع هو canActivate، فنستطيع توضيحها بالشكل التالي:

canActivate: [canActivate1, canActivate2, canActivate3]



نلاحظ أنه ينفذ الكلاس صاحب index صفر في مصفوفة الكلاس ثم الذي يليه ثم الذي يليه وهكذا.

ولكن السؤال الذي يطرح نفسه، في حال أعاد الكلاس الأول urlTree او أعاد false او أعاد true، فماذا يحصل في الأنواع الأخرى، وسوف اجيب عن هذا السؤال على شكل نقاط كالتالي:

- في حال ارجع الكلاس صاحب index صفر (canActivate1) urlTree، فانه يفوز ويتم الغاء باقي الأنواع التي تليه.
- وفي حال ارجع false ايضاً يتم الغاء الأنواع التي تليه.
- اما في حال ارجع true فان angular سوف ينقل للنوع التالي، وفي النوع الثاني يطبق عليه نفس هذه الخطوات وهكذا.

### 9.3 Secondary Route:

لقد تكلمت عن Secondary Route في قسم Children Routes لكن مررت عليه بشكل سريع ولم اوفيه حقه، لذلك أحببت ان افرد له قسم خاص به لما له من أهمية واحتياج وخصوصاً في البرامج الكبيرة.

إذا السؤال ما هو Secondary Route، وفيما يستخدم، وما هي الفائدة منه؟

وهذه الأسئلة وغيرها، أستطيع الإجابة عليها بكل بساطة بشكل مجمل وبإجابة واحدة، وهي انه في حال لدينا أكثر من outlet بنفس الصفحة فأنا نميز كل outlet باسم خاص به والـ outlet الذي لم نضع له أي اسم يعتبر Primary وباقي outlets تعتبر Secondary Routes.

ولنعطي مثال لكي يتضح المفهوم، لنفرض اننا نريد ان نعرض رسالة ترحيبية للمستخدم في حال تسجيله للدخول على الصفحة الرئيسية للموقع بحيث تبقى ثابتة في الأعلى ولو قام المستخدم بتغيير الصفحات، ويمكن عمل ذلك بوضع outlet في صفحة app.component.html بجانب outlet الأولى كالتالي:

ملف app.component.html

```
<nav class="navbar navbar-expand-sm navbar-dark bg-dark Shadow">
  <div class="navbar-brand-two" href="#">
    
  </div>
  <span class="navbar-brand">Routing</span>
  <div class="justify-content-left">
    <ul class="navbar-nav">
      <li class="nav-item">
        <a
          routerLink="/home"
          routerLinkActive="is-active"
          [routerLinkActiveOptions]="{ exact: true }"
        ><span>Home</span></a>
      </li>
      <li class="nav-item" *ngIf="IsAdmin && IsSignedIn">
        <a routerLink = "home/edit-courses" routerLinkActive="is-active"
        ><span>Edit Courses</span></a>
      </li>
      <li class="nav-item" *ngIf="IsAdmin && IsSignedIn">
        <a routerLink="/users" routerLinkActive="is-active"
        ><span>Users</span></a>
      </li>
      <li class="nav-item" *ngIf="IsSignedIn">
        <a [routerLink]="['profile', id]" routerLinkActive="is-active"
        ><span>Profile</span></a>
      </li>
      <li class="nav-item" *ngIf="!IsSignedIn">
        <a routerLink="/auth/login" routerLinkActive="is-active"
        ><span>Login</span></a>
      </li>
      <li class="nav-item" *ngIf="!IsSignedIn">
        <a routerLink="/auth/signup" routerLinkActive="is-active"
        ><span>Signup</span></a>
      </li>
      <li class="nav-item" *ngIf="IsSignedIn">
        <a routerLink="/auth/signout" routerLinkActive="is-active"
        ><span>Signout</span></a>
      </li>
    </ul>
  </div>
</nav>

<div class="spinner" *ngIf="showSpinner"></div>

<router-outlet></router-outlet>
<router-outlet></router-outlet>
```



نلاحظ اننا اضفنا outlet جديد بجانب الأول بحيث ان هذا outlet يعرض component المحتوي على رسالة الترحيب، اما الأول فكما شرحناه سابقاً يقوم بعرض الصفحات الرئيسية للتطبيق.

ولكن هنا يوجد مشكلة وهي ان angular لا يعرف أي outlet خاص بالصفحات الرئيسية والخاص بـ components الرسالة الترحيبية، لذلك لا بد ان نميز احد outlets باسم خاص به، بحيث أي outlet له اسم خاص به يسمى Secondary Outlet اما الذي لا يحتوي على اسم يسمى Primary Outlet، لذلك لنسمي Secondary Outlet مثلاً باسم msg، كالتالي:

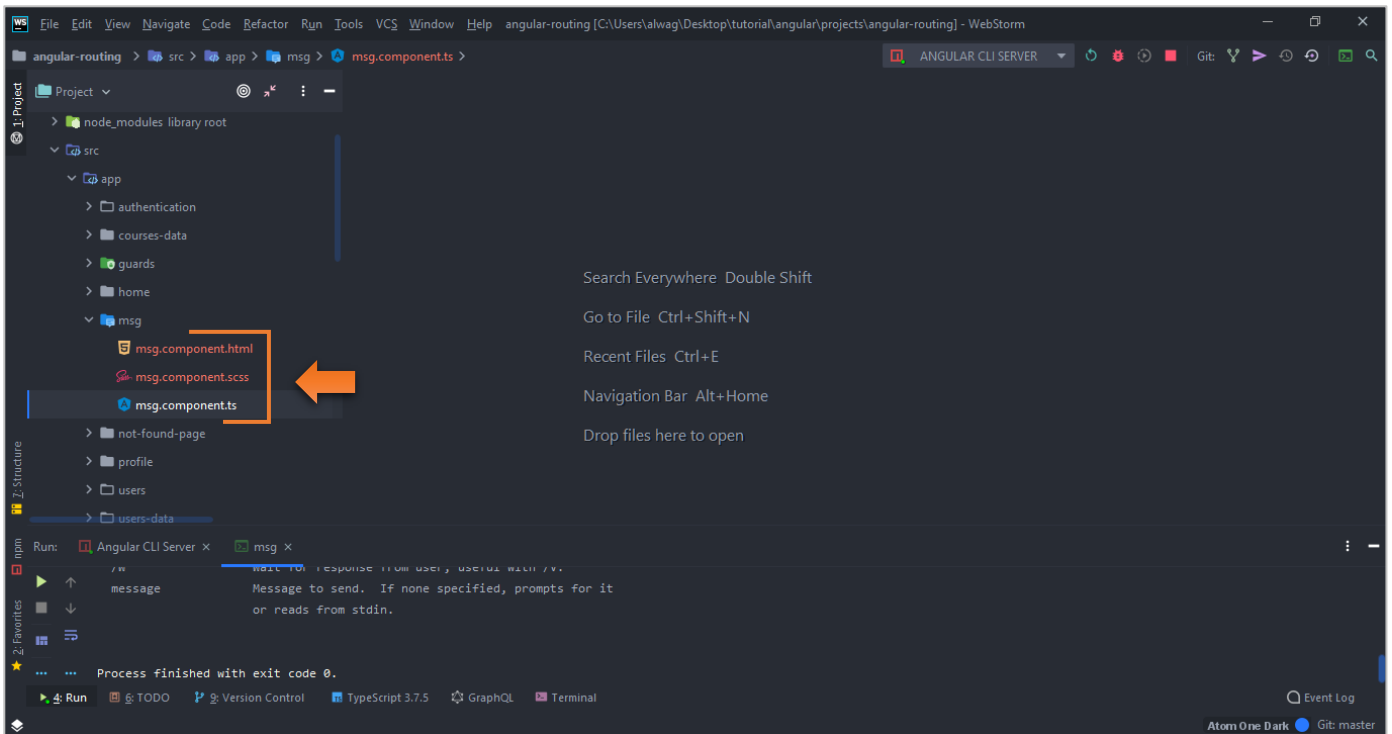
```
ملف app.component.html
<nav class="navbar navbar-expand-sm navbar-dark bg-dark Shadow">
  <div class="navbar-brand-two" href="#">
    
  </div>
  <span class="navbar-brand">Routing</span>
  <div class="justify-content-left">
    <ul class="navbar-nav">
      <li class="nav-item">
        <a
          routerLink="/home"
          routerLinkActive="is-active"
          [routerLinkActiveOptions]="{ exact: true }"
        ><span>Home</span></a>
      </li>
      <li class="nav-item" *ngIf="IsAdmin && IsSignedIn">
        <a routerLink = "home/edit-courses" routerLinkActive="is-active"
        ><span>Edit Courses</span></a>
      </li>
      <li class="nav-item" *ngIf="IsAdmin && IsSignedIn">
        <a routerLink="/users" routerLinkActive="is-active"
        ><span>Users</span></a>
      </li>
      <li class="nav-item" *ngIf="IsSignedIn">
        <a [routerLink]="['profile', id]" routerLinkActive="is-active"
        ><span>Profile</span></a>
      </li>
      <li class="nav-item" *ngIf="!IsSignedIn">
        <a routerLink="/auth/login" routerLinkActive="is-active"
        ><span>Login</span></a>
      </li>
      <li class="nav-item" *ngIf="!IsSignedIn">
        <a routerLink="/auth/signup" routerLinkActive="is-active"
        ><span>Signup</span></a>
      </li>
      <li class="nav-item" *ngIf="IsSignedIn">
        <a routerLink="/auth/signout" routerLinkActive="is-active"
        ><span>Signout</span></a>
      </li>
    </ul>
  </div>
</nav>

<div class="spinner" *ngIf="showSpinner"></div>
```

```
<router-outlet name="msg"></router-outlet>
<router-outlet></router-outlet>
```

بذلك أصبح outlet المُحتوي على اسم هو Secondary Outlet اما الثاني فهو Primary Outlet، مع العلم اننا نستطيع إضافة أي عدد من Secondary Outlets بنفس الصفحة بشرط ان نميز كل واحدة باسم خاص فيها، اما Primary فهو واحد فقط وهو outlet الذي لا يحتوي على اسم.

الآن لنقوم بإنشاء component الذي يحتوي على الرسالة الترحيبية ولنسميه مثلاً msg، كالتالي:



قبل البدء في شرح بعض المفاهيم الأخرى لـ Secondary Routed لابد ان نجد طريقة لمعرفة المستخدم هل قام بتسجيل الدخول ام لا وفي حال قام بتسجيل الدخول نريد ان نعرف بيانات هذا المستخدم، ونستطيع عمل ذلك بطرق مختلفة بحسب طريقة بناء التطبيق الخاص بنا، وفي مثالنا هذا لدينا متغير جاهز نستطيع عن طريقة نعرف هل المستخدم قام بتسجيل الدخول من عدمه وهذا المتغير هو الموجود في ملف users.service.ts باسم isLoggedIn حيث نعمل subscribe له في ملف msg.component.ts وبناءً على هذا المتغير نقوم بإظهار واخفاء هذه component، اما للحصول على بيانات هذا المستخدم، فلدينا دالة جاهزة ايضاً في ملف users.service.ts تقوم بجلب بيانات المستخدم في حال قام بتسجيل الدخول وهي login() لذلك سوف ننشأ متغير او property جديدة ونجعل نوعها public بحيث كلما قام احد المستخدمين بتسجيل الدخول تنفذ الدالة السابقة ومنها نأخذ بيانات هذا المستخدم ونخزنها في المتغير او property التي قمنا بإنشائه سابقاً، كالتالي:

ملف users.service.ts

```
import { Injectable } from '@angular/core';
import { Users } from './users';
```

```

import { of, Observable, BehaviorSubject } from 'rxjs';
import { map } from 'rxjs/operators';
import { Router } from '@angular/router';
import { StorageMap } from '@ngx-pwa/local-storage';

@Injectable({
  providedIn: 'root'
})
export class UsersService {

  public isAdmin$ = new BehaviorSubject(false);
  public isLogin$ = new BehaviorSubject(false);
  public user: Users;

  constructor(private router: Router, private storageMap: StorageMap) { }

  USERS: Users[] = [
    {
      userId: 1,
      userType: 'admin',
      name: 'Faisal',
      userCity: 'Riyadh',
      userName: 'DevFaisal',
      password: '1234',
    },
    {
      userId: 2,
      userType: 'no-admin',
      name: 'Saad',
      userCity: 'Riyadh',
      userName: 'DevSaad',
      password: '1111',
    },
    {
      userId: 3,
      userType: 'no-admin',
      name: 'Bader',
      userCity: 'Dammam',
      userName: 'DevBader',
      password: '2222',
    },
    {
      userId: 4,
      userType: 'no-admin',
      name: 'Fahd',
      userCity: 'Jeddah',
      userName: 'DevFahd',
      password: '3333',
    },
  ];
  usersList$ = of(this.USERS);

  getAllUsers(): Observable<Users[]> {
    return this.usersList$;
  }

  getUserById(id: number): Observable<Users> {
    return this.getAllUsers().pipe(
      map(users => users.find(user => {
        return user.userId === id;
      })))
  };
}

addNewUser(user: Users) {

```

```

    user.userId = this.USERS.length + 1;
    user.userType = 'no-admin';
    this.USERS.push(user);
    this.logIn(user.userName);
    this.router.navigateByUrl('/home');
  }
  logIn(userName: string) {
    this.getAllUsers()
      .pipe(map(users => users.find(user => user.userName === userName)))
      .subscribe((user) => {
        this.user = user;
        this.isLogIn$.next(true);
        user.userType === 'admin' ? this.isAdmin$.next(true) : this.isAdmin$.next(false);
        this.storageMap.set('id', (user.userId).toString()).subscribe(() => { });
        localStorage.setItem('type', user.userType);
      });
  }
  logoutUser(): void {
    this.isAdmin$.next(false);
    this.isLogIn$.next(false);
    this.storageMap.delete('id').subscribe(() => { });
    localStorage.clear();
    this.router.navigateByUrl('/home');
  }
}

```

هنا قمنا بتخزين بيانات  
المستخدم في المتغير السابق  
بعدما قام المستخدم بتسجيل  
الدخول

بعدما استطعنا الحصول على بيانات المستخدم بقي علينا ان نعمل subscribe للمتغير isLogIn\$ في ملف msg.component.ts، كالتالي:

ملف msg.component.ts

```

import { Component, OnInit } from '@angular/core';
import { UsersService } from '../users-data/users.service';

@Component({
  selector: 'app-msg',
  templateUrl: './msg.component.html',
  styleUrls: ['./msg.component.scss']
})

export class MsgComponent implements OnInit {
  message: string;

  constructor(private usersService: UsersService) {}

  ngOnInit(): void {
    this.getMessage();
  }

  getMessage(): void {
    this.usersService.isLogIn$.subscribe(LogIn => {
      if (LogIn) {
        const user = this.usersService.user;
        const currentDate = new Date();
        const date = currentDate.toDateString();
        const time = currentDate.toLocaleTimeString();
        this.message = `Hi ( ${user.name} ) Logged in at ${date} - ${time}`;
      }
    });
  }
}

```

قمنا بإنشاء دالة باسم ( getMessage ) ومحتوى هذه الدالة هو استدعاء المتغير isLogin\$ عن طريق الكلاس UserService الموجود في ملف users.service.ts، ومن ثم عملنا له subscribe والقيمة التي ترجع لنا خزناها في متغير اسمينا Login ومن ثم وضعنا شرط لتأكد إذا قيمة هذا المتغير هي true أي المستخدم قام بتسجيل الدخول، وبناء على هذا الشرط قمنا بتخزين بيانات المستخدم الموجودة في المتغير user الذي انشأناه سابقاً في ملف users.service.ts في ثابت واسميته أيضاً بنفس الاسم user (لك حرية اختيار الاسم الذي تريده)، وفي الأسطر الباقية قمنا بجلب التاريخ والوقت الحاليين من جهاز المستخدم ومن ثم في الخطوة الأخيرة قمنا بتخزين جميع هذه البيانات في متغير اسميته message، وهذا المتغير هو الذي نعمل له bind في ملف template او ما يسمى ملف html لهذا component، كالتالي:

ملف msg.component.html

```
<div class="card" *ngIf="message">
  <div class="card-body message bg-dark">
    {{message}}
  </div>
</div>
```

نلاحظ اننا قمنا بعمل binding عن طريق ميزة interpolation data binding وهي ابسط طرق Data Binding في Angular، كما وضعنا شرط عن طريق الـ Directive المسمى \*ngIf بحيث لا يعرض هذا Markup إلا إذا كان هذا المتغير يحمل قيمة بداخله، وبالطبع المتغير لا يحمل قيمة إلا إذا قام المستخدم بتسجيل الدخول.

بقي أن نقوم بوضع بعض اللمسات الجمالية لهذه الرسالة عن طريق css، كالتالي:

ملف msg.component.scss

```
.message {
  position: absolute;
  z-index: 1;
  padding: 0 0 0 10px;
  color: aliceblue;
  box-shadow: 0 0 0 1.2px grey;
  width: 100%;
  text-align: center;
}
```

طبعاً لو قمنا بتشغيل الموقع فلن يظهر شيء لأننا لم نقوم بعمل تهيئة لأي Route لهذا component، لذلك لنذهب إلى ملف app-routing.module.ts، كالتالي:

ملف app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { UsersComponent } from './users/users.component';
import { UsersListComponent } from './users/users-list/users-list.component';
import { UserDetailsComponent } from './users/user-details/user-details.component';
import { AuthenticationComponent } from './authentication/authentication.component';
import { LoginComponent } from './authentication/login/login.component';
import { SignupComponent } from './authentication/signup/signup.component';
import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
import { SingoutComponent } from './authentication/singout/singout.component';
import { ProfileComponent } from './profile/profile.component';
import { EditCoursesComponent } from './home/edit-courses/edit-courses.component';
import { CanActivateGuard, CanActivateAdminGuard } from './guards/can-activate.guard';
import { CanActivateAdminChildGuard, CanActivateChildGuard } from './guards/can-activate-
```



```

child.guard';
import {CanDeactivateGuard, CanDeactivateLoginGuard} from './guards/can-deactivate.guard';
import {ResolveCoursesService} from './guards/resolve-courses.service';
import {ResolveUserDetailsService} from './guards/resolve-user-details.service';
import {MsgComponent} from './msg/msg.component';

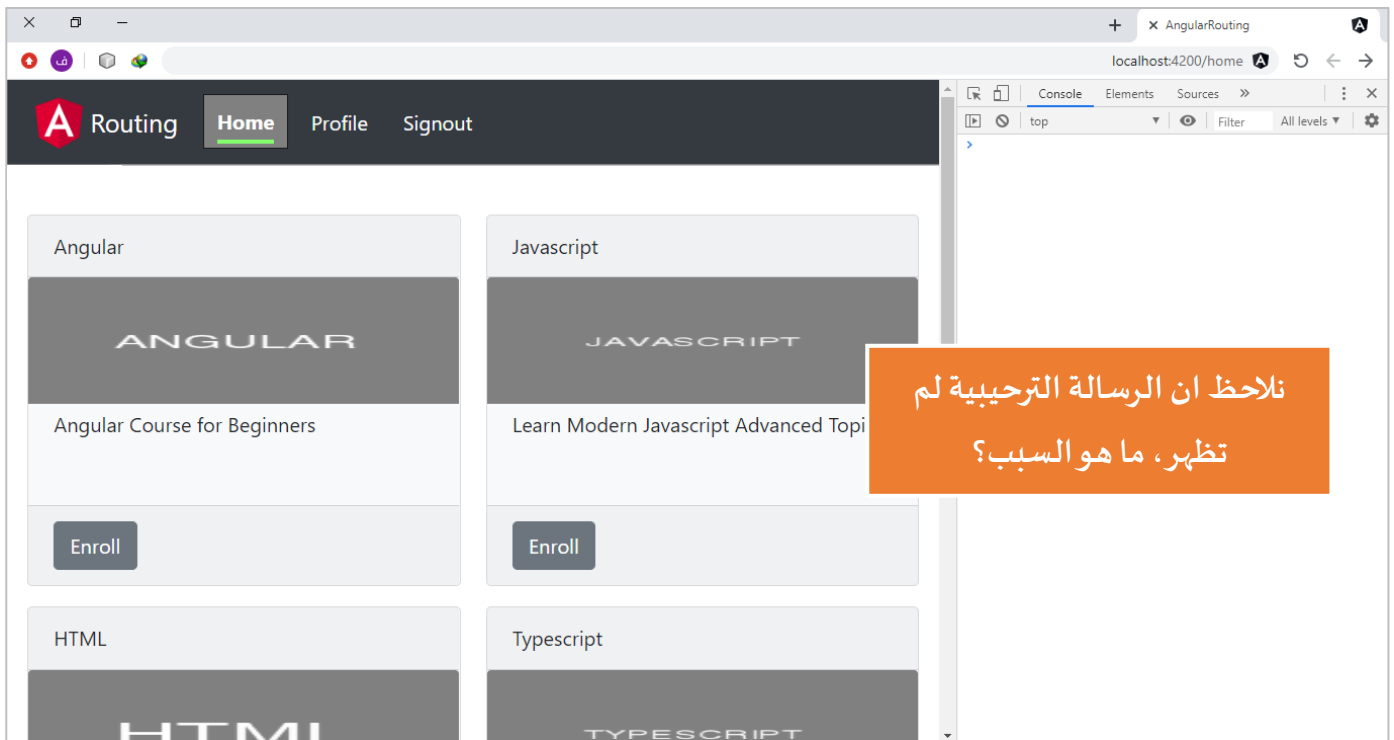
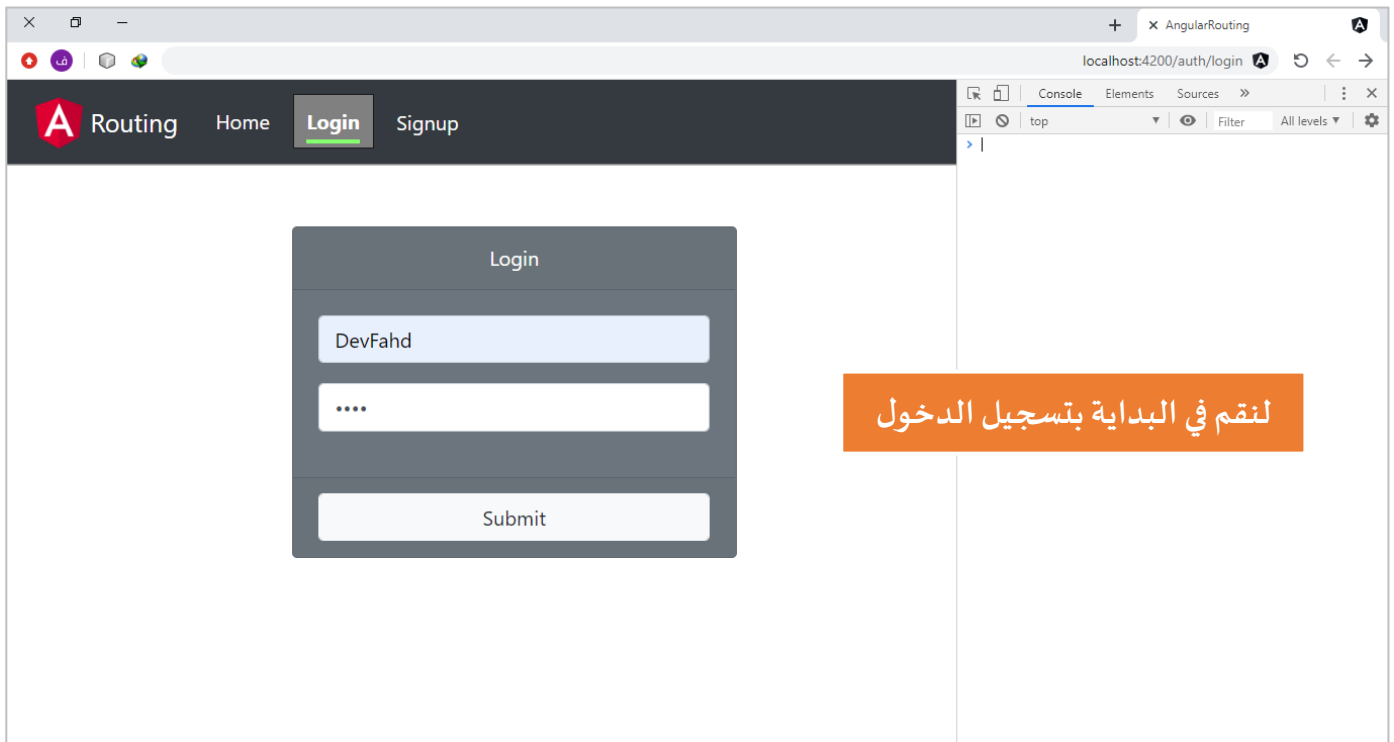
const routes: Routes = [
  {
    path: 'home',
    component: HomeComponent,
    resolve: {courses: ResolveCoursesService},
    canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
    children: [
      { path: 'edit-courses', component: EditCoursesComponent }
    ]
  },
  {path: 'messages', outlet: 'msg', component: MsgComponent},
  { path: 'profile/:id', canActivate: [CanActivateGuard], component: ProfileComponent },
  {
    path: 'users',
    canActivate: [CanActivateGuard, CanActivateAdminGuard],
    data: {
      roles: 'admin'
    },
    children: [
      { path: '', component: UsersComponent },
      { path: 'user-list', component: UsersListComponent },
      {
        path: 'user-details/:id',
        resolve: {
          userDetails: ResolveUserDetailsService
        },
        component: UserDetailsComponent
      },
    ]
  },
  {
    path: 'auth', children: [
      { path: '', component: AuthenticationComponent },
      { path: 'login', canDeactivate: [CanDeactivateLoginGuard], component: LoginComponent },
      { path: 'signup', canDeactivate: [CanDeactivateGuard], component: SignupComponent },
      { path: 'signout', component: SingoutComponent }
    ]
  },
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: '**', component: NotFoundPageComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

كما هو ملاحظ ان تهيئة Secondary Route هو مشابه لـ Primary الفرق الوحيد هو إضافة الخاصية outlet حيث اسندنا لها القيمة msg وهي القيمة الموجودة في Secondary Route الذي اضفناه سابقاً في ملف app.component.html.

الآن لنحفظ التغييرات، ونذهب إلى المتصفح ونرى النتيجة:



السبب في ذلك يعود إلى أننا لم نعمل Activating Secondary Routes صحيح أننا قمنا بعمل التهيئة ولكن لابد أن نعمل Activating، ونستطيع عمل هذا الأمر بطريقتين سواء عن طريق template أو برمجياً، وفي البداية سوف اتطرق إلى الحالات التي يمكن عمل لها Activating عن طريق template، ومن ثم أوضح الحالات عن طريق code أو برمجياً، كالتالي:

الحالة الأولى: في حال أردنا توجيهه الـ route إلى Secondary Route فقط ، مع عدم تغيير Primary Route:

### Activating Secondary Routes: Router Link

**Template**

```
<a [routerLink] = "[{ outlets: { msg: ['messages'] } }]">Go To Msg</a>
```

هنا يتم كتابة path الذي قمنا بتهيئته في ملف app-routing.module.ts

حيث هنا يتم كتابة اسم Secondary Route وفي مثالنا هنا هو msg

الحالة الثانية: في حال أردنا توجيهه الـ route إلى Secondary Route مع Primary Route بنفس الوقت:

### Activating Secondary Routes: Router Link

**Template**

```
<a [routerLink] = "[ { outlets: { primary: ['home'], msg: ['messages'] } }]">Go To Msg</a>
```

اسم Secondary Route مع القيمة الخاصة به

القيمة التي تُسند لـ primary والتي تُشير إلى الـ Route الرئيسي

خاصية تكتب نفس ماهي والقيمة التي تُسند لها هي الـ path الخاص بـ Route

الحالة الثالثة: في حال أردنا الغاء Secondary Route بدون التأثير على Primary Route:

### Activating Secondary Routes: Router Link

**Template**

```
<a [routerLink] = "[ { outlets: { msg: null } }]">Go To Msg</a>
```

فقط نقوم بإسناد القيمة null وسوف يتم الغاء Secondary Route

الحالة الرابعة: في حال أردنا الغاء Secondary Route مع الانتقال إلى Primary Route جديد:

### Activating Secondary Routes: Router Link

**Template**

```
<a [routerLink] = "[ { outlets: { primary: ['home'], msg: null } }]">Go To Msg</a>
```

الـ path الذي نريد ان ينتقل إليه Primary Route

Secondary Route مع اسناد القيمة null

خاصية تكتب كما هي ، بغرض اعلام angular اننا نريد تغيير Primary Route

اما الطريقة الثانية وهي عن طريق code او ما يسمى برمجياً، فهي مشابهة تقريباً للحالات السابقة، كالتالي:

## Activating Secondary Routes: In Code

### Component Class

```
this.router.navigate([ { outlets: { msg: [ 'messages' ] } } ] ); ①
```

```
this.router.navigate([ { outlets: { primary: [ 'home' ], msg: [ 'messages' ] } } ] ); ②
```

```
this.router.navigate([ { outlets: { msg: null } } ] ); ③
```

```
this.router.navigate([ { outlets: { primary: [ 'home' ], msg: null } } ] ); ④
```

```
this.router.navigateByUrl('/home'); ⑤
```

لعل الحالات الأربع الأولى هي مشابهة لما سبقها، بالإضافة الوحيدة هي في الحالة الخامسة حيث استخدمنا الدالة `navigateByUrl` وهذه الدالة تأخذ الرابط على شكل قطعة نصية كاملة، لذلك عندما نمرر لها أي رابط فإنها تقوم بالتحويل له كما هو مكتوب بين علامات التنصيص، وكما هو واضح الموجود بين علامات التنصيص هو `path` ذو الاسم `home` لذلك سوف يقوم بالتوجيه إلى هذا `path` وبذلك سوف يُلغى أي `Secondary Route` او بارامترات او غيره، فقط هذا `path`، وبذلك نستطيع الاستفادة من هذه الدالة في الغاء `Secondary Route`

وبعد استعراض جميع الطرق والحالات التي يمكن عن طريقها تفعيل او الغاء تفعيل `Secondary Route`، نريد الآن ان نقوم بتطبيق بعض هذه الحالات في المثال الخاص بنا، حيث نريد في البداية ان قام المستخدم بتسجيل الدخول ان يتم تفعيل `Secondary Route`، لذلك في ملف `login.component.ts` نقوم بكتابة code التالي:

ملف `login.component.ts`

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { UsersService } from 'src/app/users-data/users.service';
import { NgForm } from '@angular/forms';
import { ActivatedRoute, Router } from '@angular/router';
import { StorageMap } from '@ngx-pwa/local-storage';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})

export class LoginComponent implements OnInit {
  @ViewChild('form') form: NgForm;
  public isSaved = false;
  private path: string;

  constructor(
    private userService: UsersService,
    private activatedRoute: ActivatedRoute,
    private router: Router,
    private storageMap: StorageMap,
  ) { }

  ngOnInit() {
    if (this.activatedRoute.snapshot.queryParams.redirectUrl) {
      this.path = this.activatedRoute.snapshot.queryParams.redirectUrl;
    }
  }
}
```

```

    } else {
      this.path = '';
    }
  }

  onSubmit() {
    const value = this.form.controls.userName.value;
    this.userService.login(value);
    this.storageMap.get('id').subscribe((id) => {
      if (this.path === '') {
        this.router.navigate(['/home']);
      } else if (this.path === '/users' || this.path === '/home/edit-courses') {
        this.router.navigate(['`/${this.path}`']);
      } else {
        const path = this.path.split('/');
        this.router.navigate(['`/${path[1]}/${id}`']);
      }
    });
    this.router.navigate([{outlets: {msg: ['messages']}}]);
    this.isSaved = true;
  }
}

```

اما في ملف users.service.ts فنريد عندما يقوم المستخدم بالتسجيل كمستخدم جديد ان يقوم بتسجيل الدخول وب نفس الوقت يعمل تفعيل لـ Secondary Route لإظهار الرسالة وايضاً Primary Route لكي يقوم بتوجيه المستخدم إلى صفحة Home، لذلك سنضيف كود التفعيل في الدالة addNewUser()، وب نفس الوقت نريد ان يُلغي Secondary Route ويقوم بالتوجيه إلى صفحة Home في حال قام المستخدم بإلغاء تسجيل الخروج، لذلك سوف نكتب كود الإلغاء في دالة logoutUser()، كالتالي:

ملف users.service.ts

```

import { Injectable } from '@angular/core';
import { Users } from './users';
import { of, Observable, BehaviorSubject } from 'rxjs';
import { map } from 'rxjs/operators';
import { Router } from '@angular/router';
import { StorageMap } from '@ngx-pwa/local-storage';

@Injectable({
  providedIn: 'root'
})
export class UsersService {

  public isAdmin$ = new BehaviorSubject(false);
  public isLogin$ = new BehaviorSubject(false);
  public user: Users;

  constructor(private router: Router, private storageMap: StorageMap) { }

  USERS: Users[] = [
    {
      userId: 1,
      userType: 'admin',
      name: 'Faisal',
      userCity: 'Riyadh',
      userName: 'DevFaisal',
      password: '1234',
    },
  ],

```

```

{
  userId: 2,
  userType: 'no-admin',
  name: 'Saad',
  userCity: 'Riyadh',
  userName: 'DevSaad',
  password: '1111',
},
{
  userId: 3,
  userType: 'no-admin',
  name: 'Bader',
  userCity: 'Dammam',
  userName: 'DevBader',
  password: '2222',
},
{
  userId: 4,
  userType: 'no-admin',
  name: 'Fahd',
  userCity: 'Jeddah',
  userName: 'DevFahd',
  password: '3333',
},
];
usersList$ = of(this.USERS);

getAllUsers(): Observable<Users[]> {
  return this.usersList$;
}

getUserById(id: number): Observable<Users> {
  return this.getAllUsers().pipe(
    map(users => users.find(user => {
      return user.userId === id;
    })))
  );
}

addNewUser(user: Users) {
  user.userId = this.USERS.length + 1;
  user.userType = 'no-admin';
  this.USERS.push(user);
  this.logIn(user.userName);
  this.router.navigate([{outlets: {primary: ['home'], msg: ['messages']}}]);
  //this.router.navigateByUrl('/home');
}

logIn(userName: string) {
  this.getAllUsers()
    .pipe(map(users => users.find(user => user.userName === userName)))
    .subscribe((user) => {
      this.user = user;
      this.isLogIn$.next(true);
      user.userType === 'admin' ? this.isAdmin$.next(true) : this.isAdmin$.next(false);
      this.storageMap.set('id', (user.userId).toString()).subscribe(() => { });
      localStorage.setItem('type', user.userType);
    });
}

logoutUser(): void {
  this.isAdmin$.next(false);
}

```

نعطل هذا السطر ونستبدله بالسطر السابق

```

    this.isLogIn$.next(false);
    this.storageMap.delete('id').subscribe(() => { });
    localStorage.clear();
    this.router.navigateByUrl('/home');
  }
}

```

لنذهب إلى ملف msg.component.ts وملف msg.component.htm ونقوم بإجراء التفعيل او الغاء التفعيل لSecondary Route، مع إضافة بعض اللمسات الإضافية كالتالي:

ملف msg.component.html

```

<div class="card" *ngIf="message && !closed">
  <div class="card-body message bg-dark">
    {{message}}
    <a class="float-right closed" (click)="close()">
      <i class="fa fa-times-circle"></i>
    </a>
  </div>
</div>

<a class="button bg-dark" (click)="open()" *ngIf="closed">
  Show
  <i class="fa fa-angle-double-right"></i>
  <i class="fa fa-angle-double-right"></i>
</a>

```

ملف msg.component.ts

```

import { Component, OnInit } from '@angular/core';
import { UsersService } from '../users-data/users.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-msg',
  templateUrl: './msg.component.html',
  styleUrls: ['./msg.component.scss']
})

export class MsgComponent implements OnInit {
  message: string;
  closed = false;
  constructor(private userService: UsersService, private router: Router) {}

  ngOnInit(): void {
    this.getMessage();
  }

  close(): void {
    this.closed = true;
    this.message = '';
  }

  open(): void {
    this.getMessage();
    this.closed = false;
    this.router.navigate([{outlets: {msg: ['messages']}}]);
  }

  getMessage(): void {
    this.userService.isLogIn$.subscribe(LogIn => {
      if (LogIn) {

```

نكتفي بهذا السطر الموجود اصلاً في هذه الدالة لإلغاء Secondary Route

او يمكن تغيير السطر السابق بهذا السطر: this.router.navigate([{outlets: {primary: ['home'], msg: null}}]);

```

    const user = this.userService.user;
    const currentDate = new Date();
    const date = currentDate.toString();
    const time = currentDate.toLocaleTimeString();
    this.message = `Hi ( ${user.name} ) Logged in at ${date} - ${time}`;
  }
});
}
}

```

ملف msg.component.scss

```

.message {
  position: absolute;
  z-index: 1;
  padding: 0 0 0 10px;
  color: aliceblue;
  box-shadow: 0 0 0 1.2px grey;
  width: 100%;
  text-align: center;
}

.closed {
  padding-right: 5px;
}

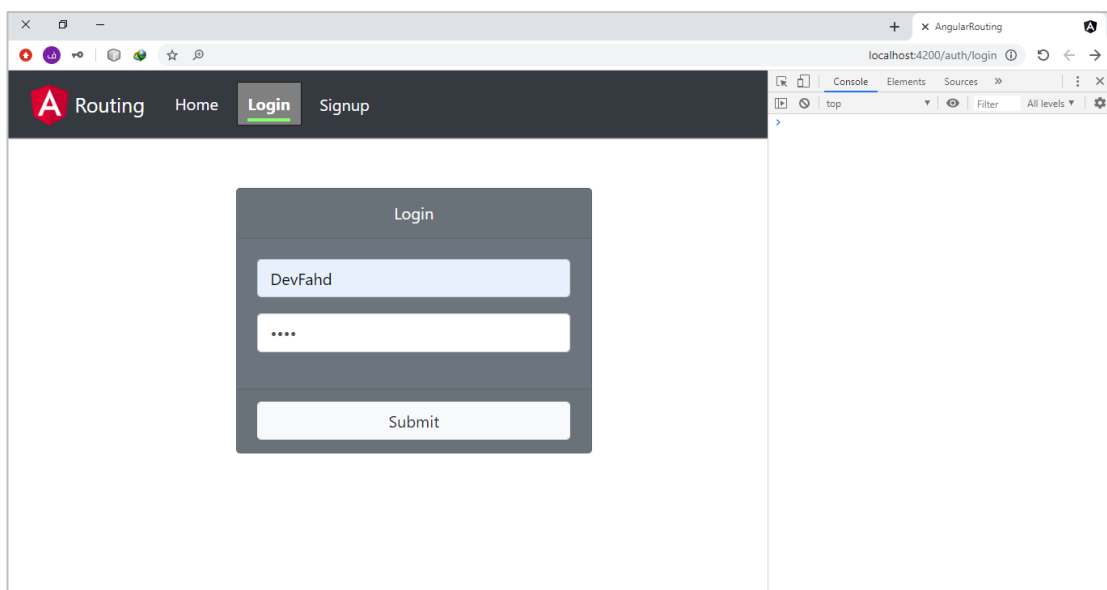
.closed:hover {
  cursor: pointer;
}

.button:hover {
  cursor: pointer;
  color: white;
}

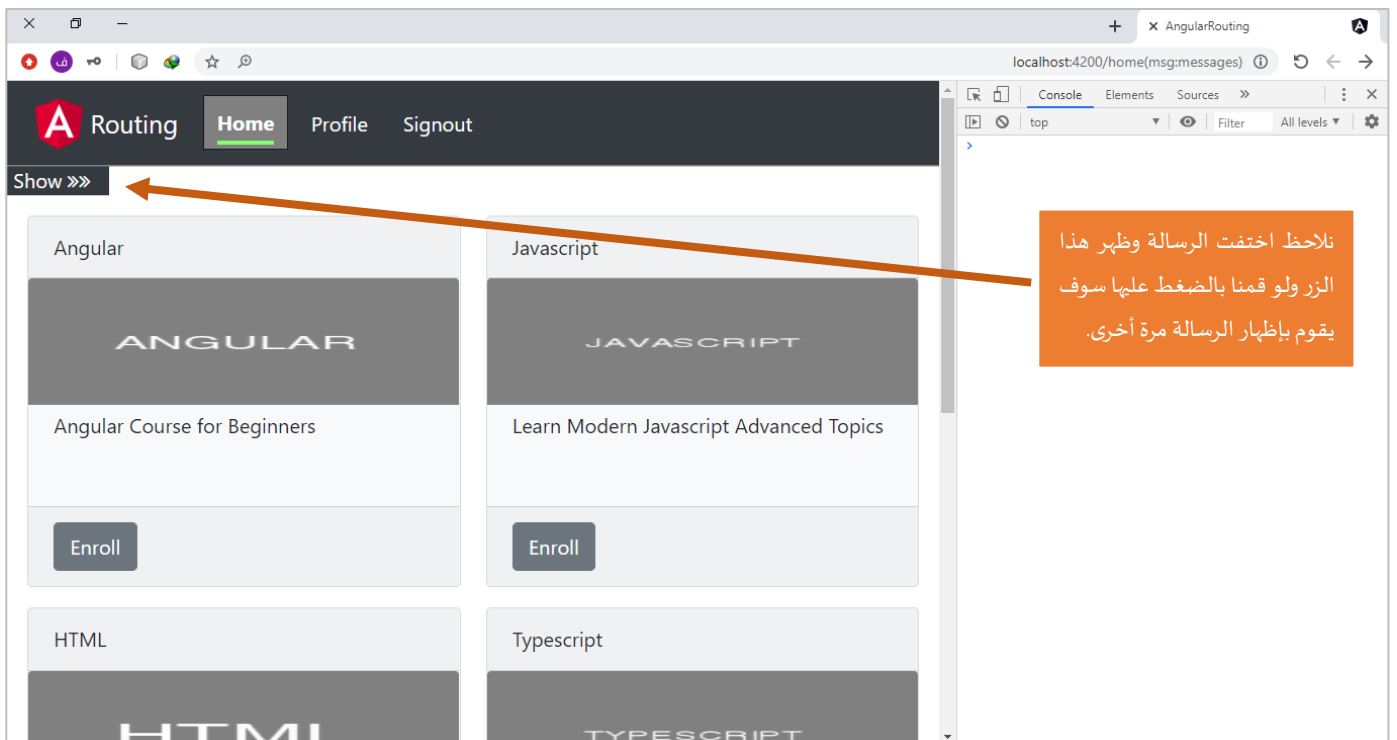
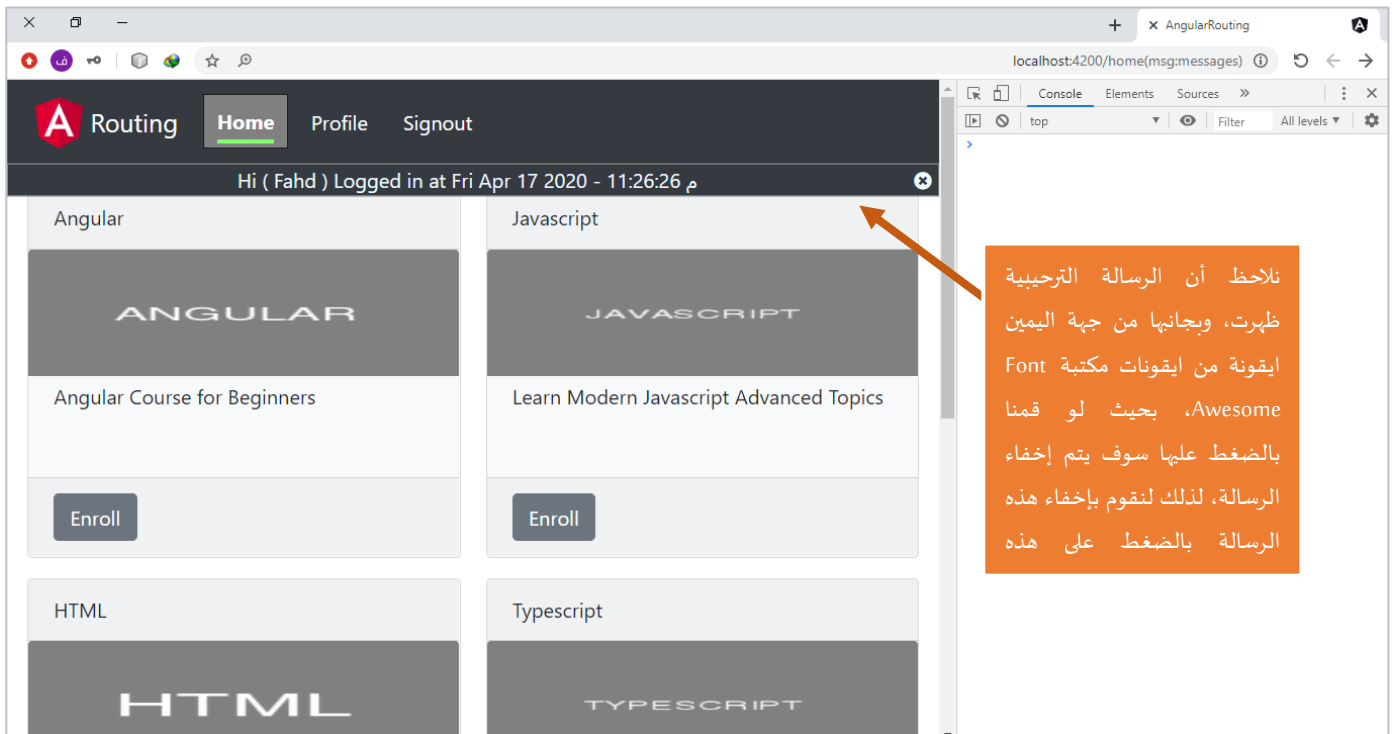
.button {
  display: inline-block;
  text-decoration: none;
  color: white;
  padding: 0 15px 0 5px;
}

```

ملاحظة: استخدمت للأيقونات مكتبة Font Awesome، الآن لنحفظ التعديلات ونذهب إلى المتصفح ونرى النتيجة:







الآن لنقوم بتكبير الرابط ونرى ما جرى فيه من تغيير:

localhost:4200/home(msg:messages)

AngularRouting

Console Elements Sources

top Filter All levels

اما هذا فيمثل Primary Route، وعند تغييره فهو لا يؤثر على Secondary Route، فيمكن مثلاً الانتقال إلى صفحة Profile، والذي سوف يتغير هو path الموجود في Primary Rote

هذا هو Secondary Route بين القوسين، حيث msg يمثل اسم outlet اما messages فهو يمثل path الذي يقوم بالتوجيه إلى msg component.

localhost:4200/profile/4(msg:messages)

AngularRouting

Console Elements Sources

top Filter All levels

نلاحظ هنا اننا عندما اخترنا صفحة profile فإنه تم تغيير path الموجود Primary Route.

اما Secondary Route فنستطيع الغاءه بأحد الطرق والحالات التي تم ذكرها سابقا

الآن لنقم بتسجيل الخروج، ونرى التغيير في الرابط، كالتالي:

AngularRouting

localhost:4200/home

Console Elements Sources

top Filter All levels

Angular

ANGULAR

Angular Course for Beginners

Enroll

Javascript

JAVASCRIPT

Learn Modern Javascript Advanced Topics

Enroll

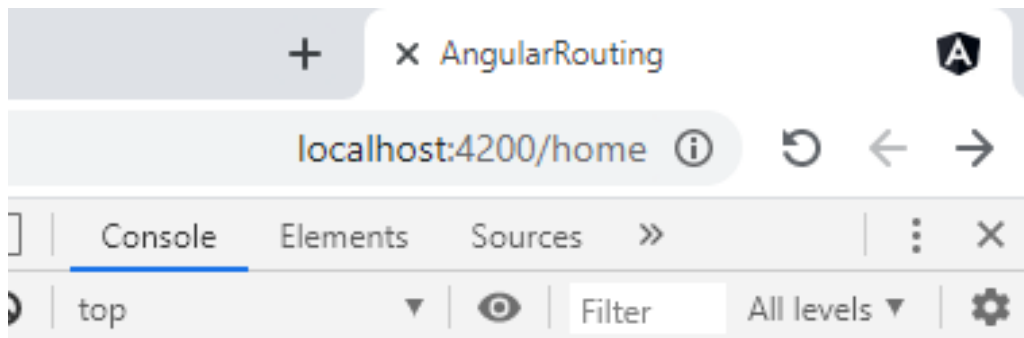
HTML

HTML

Typescript

TYPESCRIPT

لنقم بتكبير الرابط، لنرى ما جرى فيه من تغيير:



نلاحظ ان Secondary Route تم حذفه من الرابط.

وبذلك نكون أنهينا اهم واغلب مفاهيم Angular Routing، وتكون بذلك عزيزي المتعلم قد امتلكت القدرة على بناء نظام Routing متكامل، ولم يبق لنا إلى ميزات Lazy Loading و preloading وهذان الميزتان تفيد في تسريع Routing او بمعنى آخر تفيد في تسريع الموقع الخاص بك (تطبيق الويب)، ولكن لا نستطيع فهمها إلا بفهم Angular Module، وهذا ما سوف نتكلم عنه في القسم التالي بإذن الله.

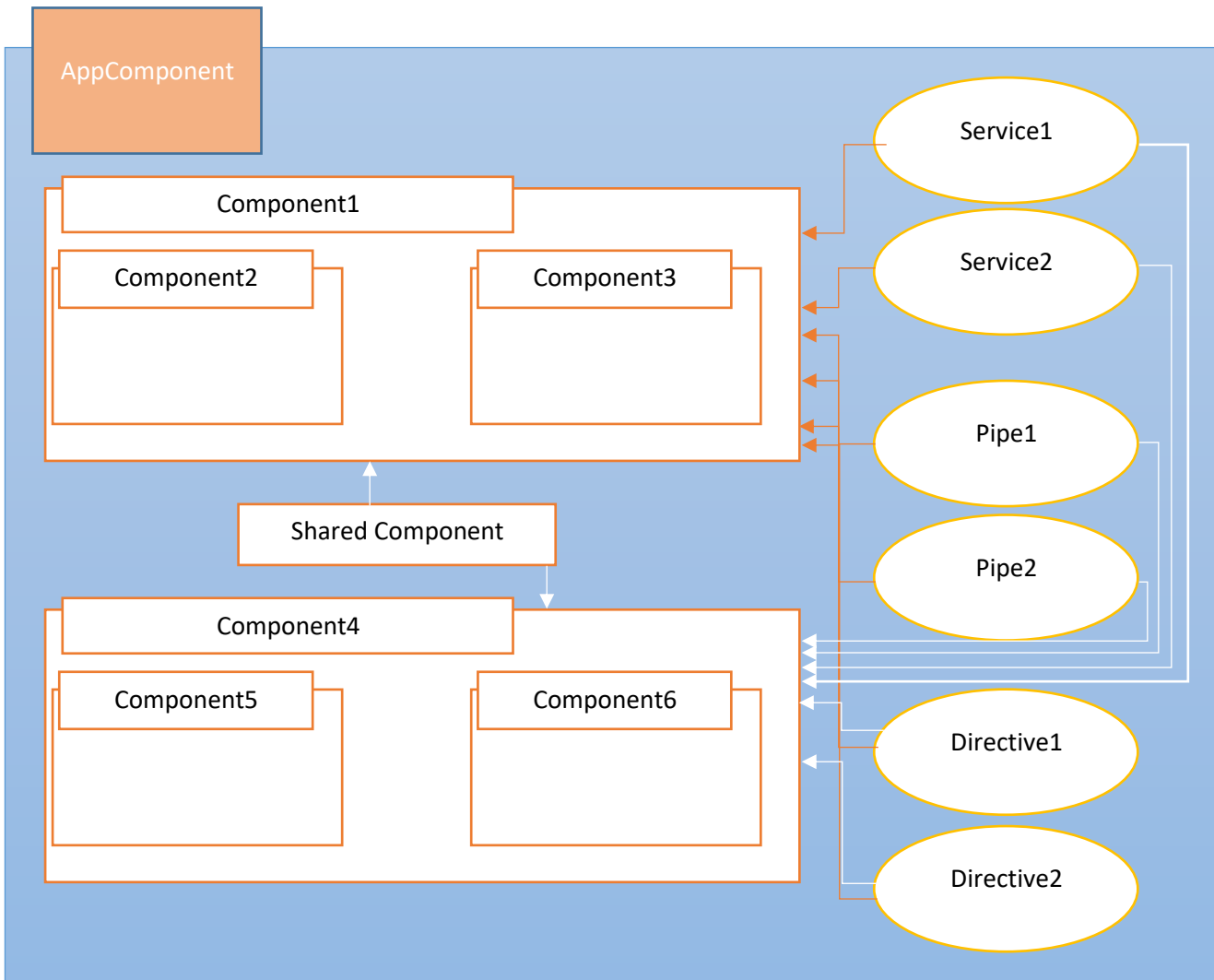
# الفصل الرابع

## Angular Modules

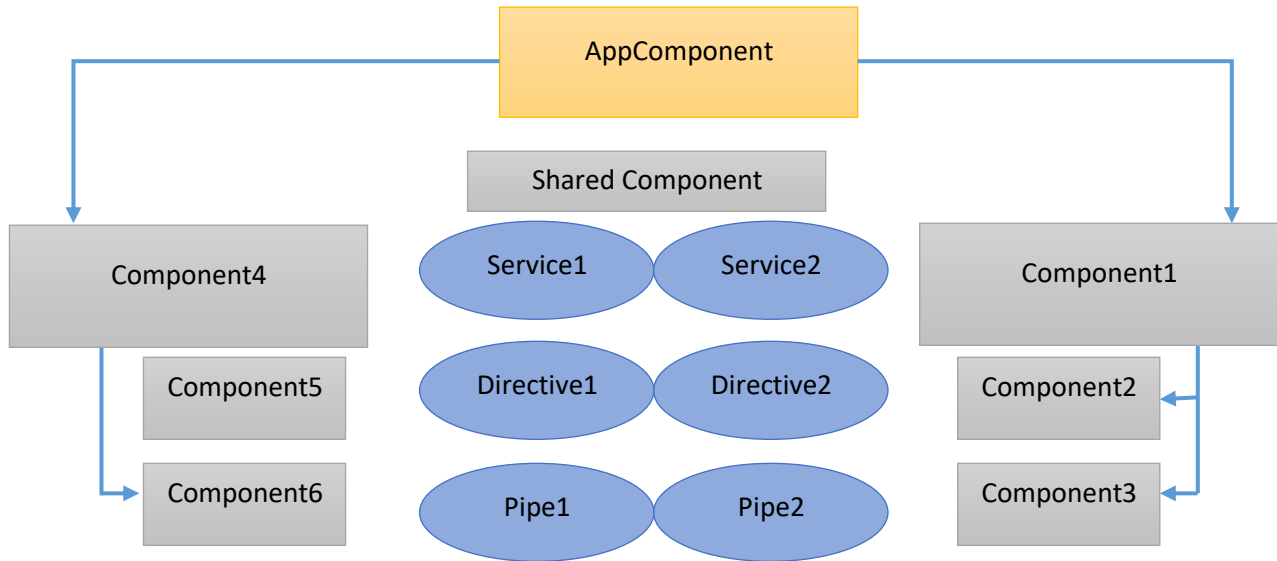
## 1.4. مقدمة:

تعتبر Angular Modules طريقة أو آلية لتنظيم اكواد المشروع ورفع كفاءته، لذلك هي اختيارية، فلو أنك عزيزي المتعلم اردت بناء مشروعك الخاص بدون تنظيمه باستخدام هذه الآلية، فلن تواجه أي مشكلة وسوف يعمل المشروع ولكن قد يكون أدائه ضعيف نسبياً، وخصوصاً إذا كان المشروع متوسط أو كبير.

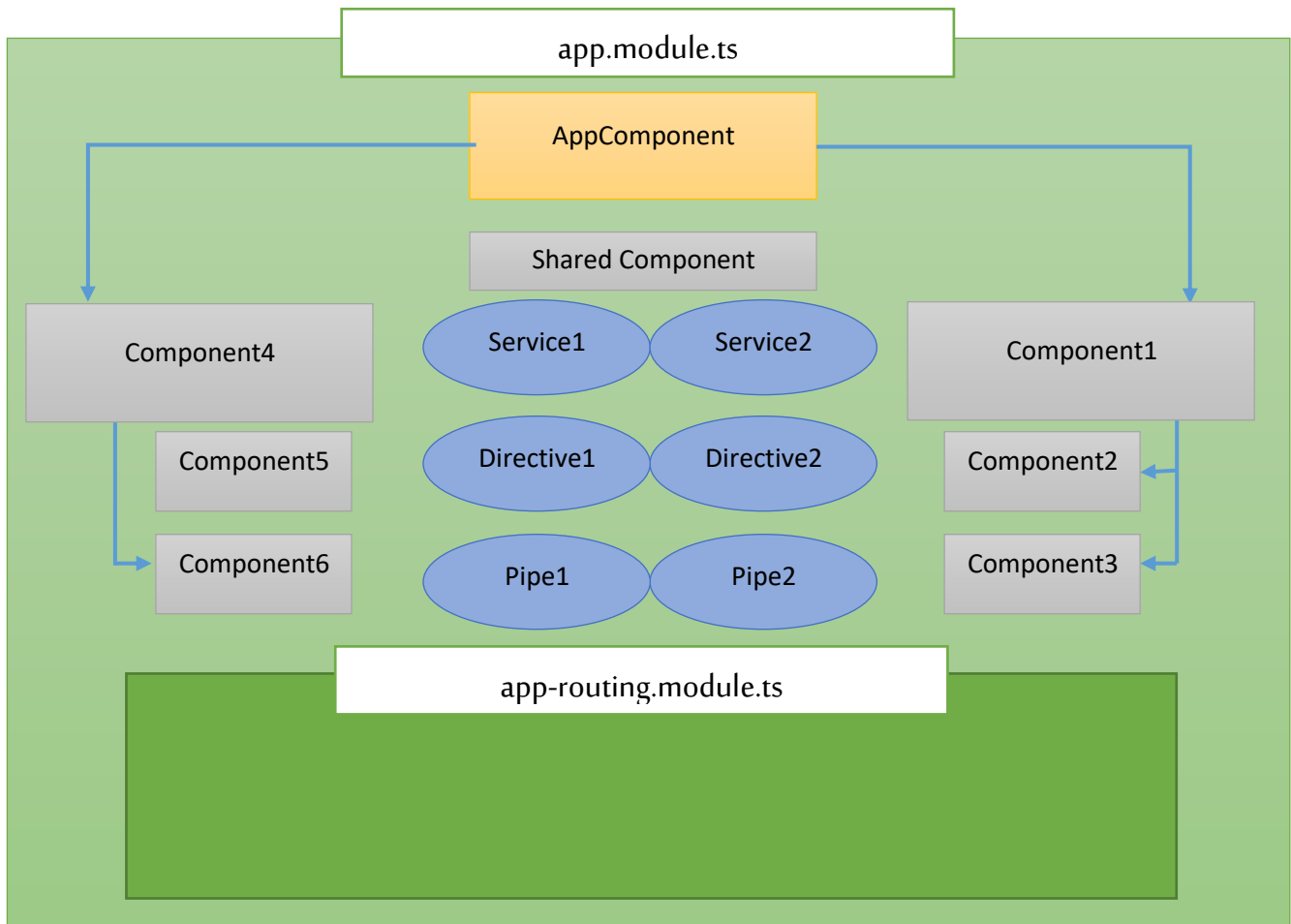
لكن السؤال الذي يطرح نفسه، ما هو Angular Modules؟ وللإجابة على هذا السؤال لنرجع لفكرة Angular في بناء المواقع، حيث تكمن الفكرة في وجود مجموعة من components متداخلة ومتجاورة مع بعضها البعض، ويمكن التنقل بين هذه components عن طريق Routing، كما انه يوجد ملفات تُدعى services مهمتها استقبال البيانات من server ومن ثم مشاركة هذه البيانات في اكثر من component او على الأقل لعرضها في component واحد، او يمكن استخدام هذه الـ services لمشاركة مجموعة من المتغيرات في اكثر من component كما كنا نفعل في المتغيرات isAdmin\$ و isLogin\$، وهناك ايضا Directives-Pipes-...etc وجميع هذه التقنيات قد نحتاج إلى أن نعمل لها Shared في جميع components المختلفة، بل بعض الأحيان قد يكون هنالك component نعمل له مشاركة shared في اكثر من component، كأن يكون لدينا component يقوم بعرض spinner عند أي عملية عرض للبيانات في أي component. وهذه الملفات تحتاج إلى تنظيم لكي يسهل التعامل معها وتطويرها وإيجاد الأخطاء...الخ، ويمكن تمثيل هذه components، المتداخلة بالشكل التالي:



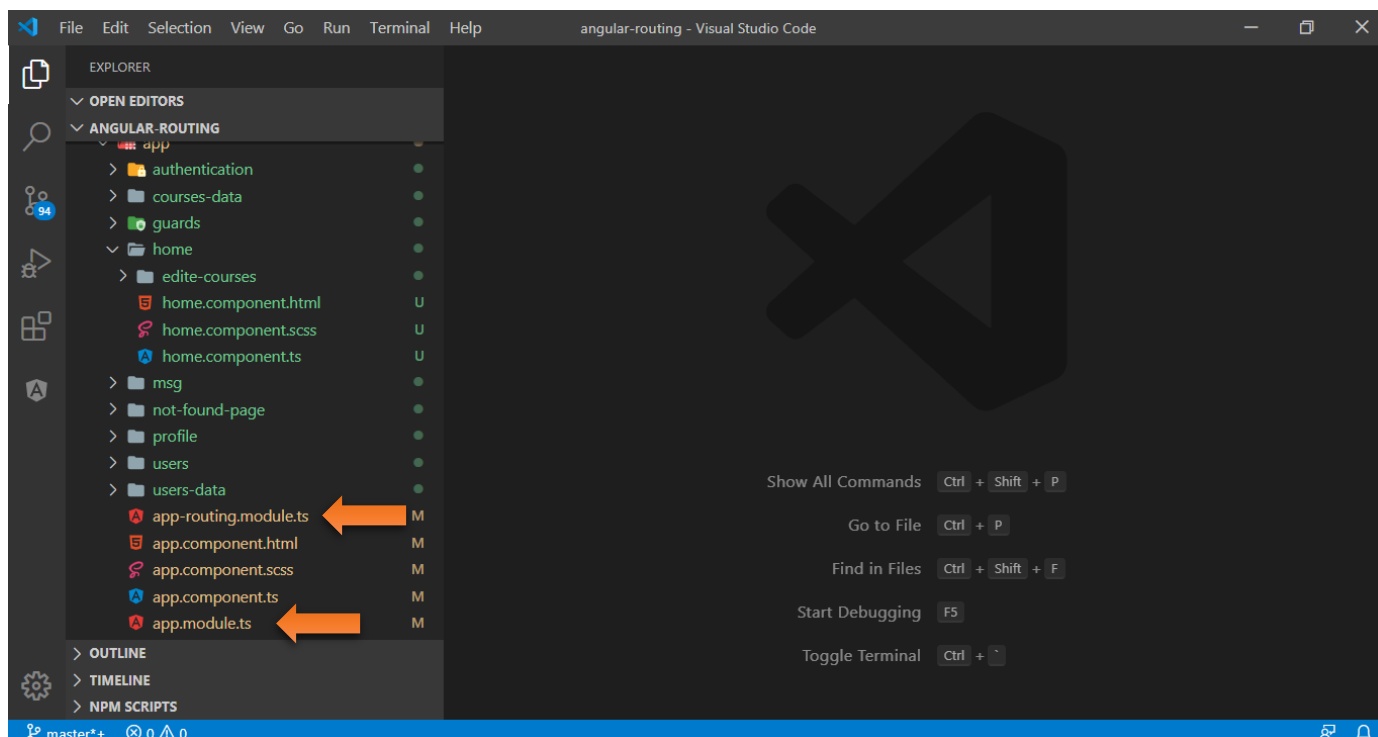
ويمكن إعادة رسم هذا الشكل بشكل شجري، كالتالي:



طبعاً هذا شكل تقريبي لمحاولة الفهم وتبسيط الأمور، اما بالنسبة لمثالنا السابق فجميع هذه الملفات مجمعة او معمول لها Grouping في Module واحد يسمى `app.module.ts`، يشمل ايضاً جميع Routes حيث تم تجميعها هي الأخرى في ملف واحد اسمه `app-routing.module.ts` كالتالي:



طبعاً الشكل السابق يُحاكي أي شكل افتراضي لأي مشروع Angular، ولو ذهبنا إلى محرر الأكواد لأستعراض مشروعنا السابق الذي قمنا بإنشائه في القسم الأول Angular Routing، لوجدنا هذين Modules، موجودة بشكل افتراضي كالتالي:



ولو استعرضنا محتويات ملف app-routing.module.ts لوجدنا جميع Routes وهيبتها موجودة في هذا الملف، كالتالي:

#### ملف app-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { HomeComponent } from './home/home.component';
4. import { UsersComponent } from './users/users.component';
5. import { UsersListComponent } from './users/users-list/users-list.component';
6. import { UserDetailsComponent } from './users/user-details/user-details.component';
7. import { AuthenticationComponent } from './authentication/authentication.component';
8. import { LoginComponent } from './authentication/login/login.component';
9. import { SignupComponent } from './authentication/signup/signup.component';
10. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
11. import { SingoutComponent } from './authentication/singout/singout.component';
12. import { ProfileComponent } from './profile/profile.component';
13. import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
14. import { CanActivateGuard, CanActivateAdminGuard } from './guards/can-activate.guard';
15. import { CanActivateAdminChildGuard, CanActivateChildGuard } from './guards/can-activate-child.guard';
16. import { CanDeactivateGuard, CanDeactivateLoginGuard } from './guards/can-deactivate.guard';
17. import { ResolveCoursesService } from './guards/resolve-courses.service';
18. import { ResolveUserDetailsService } from './guards/resolve-user-details.service';
19. import { MsgComponent } from './msg/msg.component';
20.
21. const routes: Routes = [
22.   {
23.     path: 'home',
```

```

24.     component: HomeComponent,
25.     resolve: {courses: ResolveCoursesService},
26.     canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
27.     children: [
28.       { path: 'edit-courses', component: EditCoursesComponent }
29.     ]
30.   },
31. {path: 'messages', outlet: 'msg', component: MsgComponent},
32. { path: 'profile/:id', canActivate: [CanActivateGuard], component: ProfileComponent },
33. {
34.   path: 'users',
35.   canActivate: [CanActivateGuard, CanActivateAdminGuard],
36.   data: {
37.     roles: 'admin'
38.   },
39.   children: [
40.     { path: '', component: UsersComponent },
41.     { path: 'user-list', component: UsersListComponent },
42.     {
43.       path: 'user-details/:id',
44.       resolve: {
45.         userDetails: ResolveUserDetailsService
46.       },
47.       component: UserDetailsComponent },
48.   ]
49. },
50. {
51.   path: 'auth', children: [
52.     { path: '', component: AuthenticationComponent },
53.     { path: 'login', canActivate: [CanDeactivateLoginGuard], component: LoginComponent },
54.     { path: 'signup', canActivate: [CanDeactivateGuard], component: SignupComponent },
55.     { path: 'signout', component: SingoutComponent }
56.   ]
57. },
58. { path: '', redirectTo: 'home', pathMatch: 'full' },
59. { path: '**', component: NotFoundPageComponent }
60. ];
61.
62. @NgModule({
63.   imports: [RouterModule.forRoot(routes)],
64.   exports: [RouterModule]
65. })
66. export class AppRoutingModule { }

```

نلاحظ ضخامة هذا الملف مع العلم ان مشروعنا ليس بهذا الحجم من التعقيد، حيث نرى الاستدعاءات الكثيرة لـ `imports` والتهيئة لجميع الـ `Routes`، ولو القينا نظرة على الملف الثاني `app.module.ts`، لوجدنا التالي:



```

1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import {FormsModule, ReactiveFormsModule} from '@angular/forms';
4.
5. import { AppRoutingModuleModule } from './app-routing.module';
6. import { AppComponent } from './app.component';
7. import { AuthenticationComponent } from './authentication/authentication.component';
8. import { LoginComponent } from './authentication/login/login.component';
9. import { SingoutComponent } from './authentication/singout/singout.component';
10. import { SignupComponent } from './authentication/signup/signup.component';
11. import { HomeComponent } from './home/home.component';
12. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
13. import { UsersComponent } from './users/users.component';
14. import { UserDetailsComponent } from './users/user-details/user-details.component';
15. import { UsersListComponent } from './users/users-list/users-list.component';
16. import { ProfileComponent } from './profile/profile.component';
17. import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
18. import { MsgComponent } from './msg/msg.component';
19.
20. @NgModule({
21.   declarations: [
22.     AppComponent,
23.     AuthenticationComponent,
24.     LoginComponent,
25.     SingoutComponent,
26.     SignupComponent,
27.     HomeComponent,
28.     NotFoundPageComponent,
29.     UsersComponent,
30.     UserDetailsComponent,
31.     UsersListComponent,
32.     ProfileComponent,
33.     EditCoursesComponent,
34.     MsgComponent
35.   ],
36.   imports: [
37.     BrowserModule,
38.     AppRoutingModuleModule,
39.     FormsModule,
40.     ReactiveFormsModule
41.   ],
42.   providers: [],
43.   bootstrap: [AppComponent]
44. })
45.
46. export class AppModule { }

```

نلاحظ ان الكلاس ذو الاسم AppRoutingModule الموجود في الملف app-routing.module.ts تم استدعائه وتنفيذه هنا، بالإضافة الى باقي components الأخرى وباقي Modules الأخرى التي نحتاجها لكي نقوم بتنفيذ بعض الأوامر كالتعامل مع النماذج او الدايكريتييف المبنية ضمناً (ngIf,ngFor,...etc) مثل FormsModule-NgModule-...etc، ولا يخفى عليك عزيزي المتعلم تكرار استدعاء components في كلا الملفين.

وهذا app.module.ts يسمى **Root Module** حيث أن Angular اول ما يبدأ في تنفيذ المشروع سوف يقرأ جميع الملفات الموجودة فيه، وكما قلنا سابقاً ان إبقاء الملفات بهذا الشكل لن يمنع تطبيقك من ان يعمل ولكن إذا رأيت ان مستخدمي تطبيقك بدءوا بالعزوف عن تصفح تطبيق الويب الخاص بك بسبب البطء الشديد فلا ترمي الخطأ على مثلاً الاستضافة الموجودة بها تطبيقك، وتترك المشكلة الأساسية وهي طريقة بنائك للمشروع الخاص بك، ومن طرق الحل مثلاً لماذا لانقوم بتقسيم هذا Root Module إلى مجموعة من Modules الصغيرة، بحيث كل Module يختص بمجموعة من components، وبالتوازي يتم أيضاً تقسيم app-routing.module.ts إلى أجزاء اصغر، ولو نظرنا إلى مشروعنا لوجدنا انه مقسم إلى مجموعة اقسام، القسم الذي يشمل HomeComponent والأبن الخاص به EditCoursesComponent بالإضافة إلى اقسام AuthenticationComponent وما يحتويه من أبناء بالإضافة إلى ProfileComponent وغيره من components المختلفة، لذلك لنقوم بوضع كل قسم من هذه الأقسام Module خاص به بالإضافة إلى Module نضع فيه Routes الخاص بهذه الأقسام فقط، وهذا النوع من Module يسمى **Feature Modules** اما Module الذي نضع فيه Routes يسمى **Routing Modules**، مع العلم ان هذه Feature Modules يتم استدعائها في Root Module وبنفس الطريقة مع Routing Modules حيث يتم استدعائه في AppRoutingModule، كما انه يوجد لدينا بعض Module الجاهزة التي يمكن استخدامها في أكثر من Component او قد يكون لدينا Directives او قد يكون لدينا Pipes أو حتى component معين، فهذه الأنواع من الملفات وغيرها التي يمكن ان يتم إعادة استدعائها في اكثر من Component يمكن ان نضع Module خاص بها او اكثر من Module اذا استدعت الحاجة فمثلاً قد يكون لدينا Module خاص Directive وModule آخر خاص Pipe وهكذا، بحيث يتم استدعاء هذا Module في كل Feature Module يحتاجه وهذا Feature Module هو الذي يجعله متاح لهذه components الخاص به، وهذا النوع من Modules يسمى **Shared Module**، أما إذا كان الملف فيتم استدعائه مره واحده فقط في Root Module ولا يمكن استدعائه في أي Module آخر، كملفات Service على سبيل المثال او لدينا component يتم عرضه مره واحدة فقط، كأن يكون لدينا component يعرض لنا navbar حيث اول ما يتم تشغيل التطبيق سوف يتم عرض هذا component الذي يحتوي على هذا navbar ويضلل معروض في المتصفح في حال كان التطبيق يعمل، ففي هذه الحالة وغيرها من الحالات نستطيع عمل Module خاص بهذا النوع من الحالات ويسمى هذا النوع **Core Module**، وبعد هذه المقدمة الطويلة نوعاً ما نستطيع أن نقول انه يوجد لدينا خمس أنواع من Modules يقدمها لنا إطار عمل Angular، هي:

Root Module - Feature Module - Routing Module - Shared Module - Core Module

ولكن قبل ان نتكلم عن هذه الأنواع بشيء من التفصيل لابد أن نستنتج الفوائد التي يجنيها المبرمج من استخدام هذه الآلية في بناء مشاريع Angular الخاصة به، ونستطيع حصر الفوائد في أربع فوائد هي:

- (١) التنظيم الجيد للكود، بحيث يكون المشروع مقسم إلى أجزاء صغيرة وكل جزء يحتوي على components الخاصة به وملفاته الأخرى ويتم عمل لها Grouping عن طريق Module خاص بهم.
- (٢) البعد عن تكرار الكود، بحيث يمكن استخدام الكود الواحد في أكثر من مكان عن طريق استدعائه.
- (٣) سهولة التطوير والصيانة للكود، فلو مثلاً أردنا تطوير جزء من المشروع فنستطيع عمل ذلك بكل بساطة حيث يتم التركيز على الجزئية المراد تطويرها بدون التأثير على باقي أجزاء المشروع.
- (٤) رفع أداء وكفاءة المشروع، ويتضح هذا جلياً في تقنيات Lazy Loading وتقنية PreLoading، وسوف نتكلم عن هذين النوعين بإذن الله بشيء من التفصيل لاحقاً في هذا الكتاب.

أما الآن فلنبدأ بأول نوع وهو Root Module.

## 2.4. Root Module:

لو نظرنا إلى ملف app.module.ts واستعرضنا محتوياته لوجدنا انه ملف كلاس عادي تم تعريفه عن طريق Decorator ذو الاسم @NgModule، وتم تمرير كائن له وهذا الكائن يحتوي على اربع خصائص او بمسمى آخر مفاتيح Keys، وكل خاصية من هذه الخصائص تم اسناد له مصفوفة، ويمكن توضيح هذه الخصائص كالتالي:

- (١) Declarations: ويتم فيها تعريف جميع components الأخرى في المشروع، بالإضافة إلى Directives و Pipes.
- (٢) Imports: ويتم فيها تعريف جميع Modules الأخرى سواء كانت Modules-Shared Modules-...ets او Modules الجاهزة سواء الموجودة في ضمنياً في اطار عمل Angular مثل FormsModule-BrowserModule-...etc او تلك الموجودة في مكتبات خارجية قد نحتاجها في المشروع الخاص بنا.
- (٣) Providers: وهي الطريقة القديمة لتعريف Services، حيث كان سابقاً أي services يتم إدراجها في المشروع ونريدها ان تكون عامة يستفيد منها كافة Components في التطبيق إذاً لابد من تعريفها هنا ولكن في الإصدارات الجديدة من Angular تم تغيير طريقة التعريف إلى طريقة أخرى سوف اتطرق إليها بإذن الله عندما أتكلم عن Core Module.

(٤) Bootstrap: وهذه الخاصية يتم فيها تعريف component الذي يبدأ فيه المشروع وبشكل افتراضي تم تعريف الكلاس AppComponent الموجود في ملف app.component.ts.

- (٥) Entry Component: وهذا غير موجود بشكل افتراضي ولكن نضيفه يدوياً وفي حال أردنا ان نضيفه بشكل افتراضي فلا بد ان نضيف خيار معين option في Angular CLI عند بداية انشاء هذا المشروع وهو --entry-component، اما الفائدة منه فيتم استخدامه في العادة مع Components الديناميكية، ولمعرفة أكثر عن Angular CLI و Components الديناميكية الرجاء مراجعة الكتاب الأول والثاني من هذه السلسلة.

مع العلم انه يوجد فقط Root Module واحد في مشروع Angular ولا يسمح بوجود أكثر من Root Module.

### 3.4. Core Module:

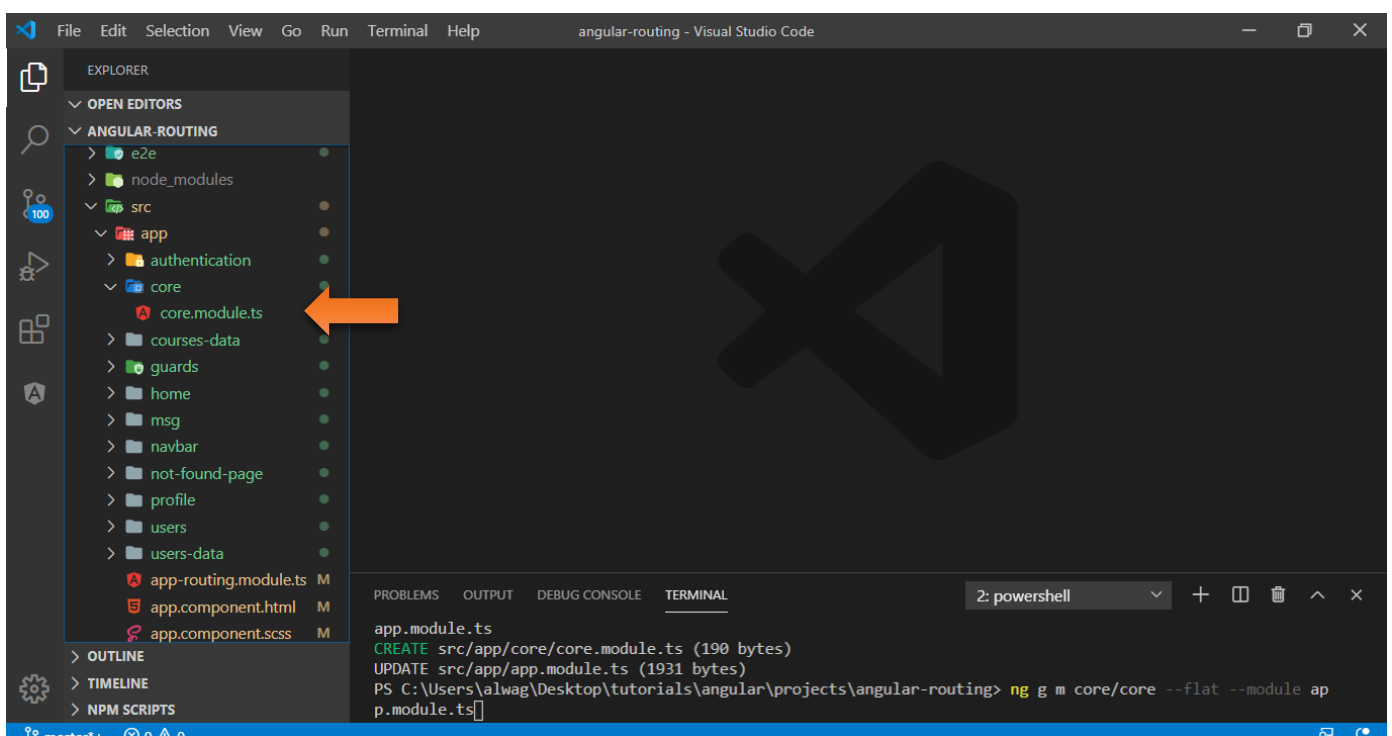
هنالك بعض الملفات التي يتم استدعاء كلاساتها فقط في Root Module ولا يمكن استدعائها في أي Module آخر، على سبيل المثال قد يكون لدينا component يتم عرضه في بداية تشغيل المشروع وتبقى محتوياته ثابتة مادام المشروع معروض على الشاشة كـ navbar مثلاً، وايضاً هنالك Root لجميع الـ Routes والموجودة في ملف `app-routing.module.ts` فهذا الملف يتم استدعائه فقط في Root Module، وايضاً لدينا `services` هي الأخرى يتم استدعائها في Root Module إذا كنا نريد ان تكون مرئية ويستفيد من جميع `Functionality` الموجودة فيها بجميع أجزاء التطبيق، فجميع هذه الميزات نستطيع وضعها في Module خاص بها ونعرف بداخله أي `services` او `component` او أي ميزة أخرى يتم عمل تشغيل لها في بداية تشغيل التطبيق ومن ثم تبقى في الذاكرة إلا أن يتم اغلاق هذا التطبيق.

ومن ثم بعد ان نقوم بتعريف هذه الميزات نستدعي هذا Core Module في Root Module، وكما قلنا سابقاً فصل هذا النوع من الميزات في هذا Core Module هو فقط لتنظيم وتسهيل التطوير مستقبلاً.

والآن لنقوم بإنشاء Module جديد (Core Module) هو عبارة عن Module عادي ولكن الفرق الوحيد هو في كيفية استعماله)، من خلال كتابة هذا الأمر في terminal، كالتالي:

**`ng g m core/core --flat --module app.module.ts`**

ومن خلال الامر السابق طلبنا من angular ان يقوم بإنشاء module جديد من خلال الامر `m` وهو اختصار `module` اما موقعه فهو في مجلد اسميانه `core` واسم هذا module ايضاً `core` (لك حرية اختيار الاسم الذي تريده ولكن يُفضل ان تسميه `core`) اما `--flat` فمعناه اننا لا نريد منه ان يُنشأ مجلد داخل `core` وانما يكتفي بالمجلد المنشأ من خلال الامر السابق اما `--module` فمعناه قم بإضافة هذا التعريف إلى Root Module، وبعدما ان نقوم بكتابة الامر السابق نضغط على زر `Enter` في لوحة المفاتيح ونرى النتيجة:



نلاحظ انه أضاف لنا Module جديد والآن لنستعرض محتويات app.module.ts، كالتالي:

#### ملف app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { FormsModule, ReactiveFormsModule } from '@angular/forms';
4. import { UsersModule } from '../users/users.module';
5. import { AppRoutingModule } from '../app-routing.module';
6. import { AppComponent } from '../app.component';
7. import { AuthenticationComponent } from '../authentication/authentication.component';
8. import { LoginComponent } from '../authentication/login/login.component';
9. import { SingoutComponent } from '../authentication/singout/singout.component';
10. import { SignupComponent } from '../authentication/signup/signup.component';
11. import { HomeComponent } from '../home/home.component';
12. import { NotFoundPageComponent } from '../not-found-page/not-found-page.component';
13. import { UsersComponent } from '../users/users.component';
14. import { UserDetailsComponent } from '../users/user-details/user-details.component';
15. import { UsersListComponent } from '../users/users-list/users-list.component';
16. import { ProfileComponent } from '../profile/profile.component';
17. import { EditCoursesComponent } from '../home/edite-courses/edit-courses.component';
18. import { MsgComponent } from '../msg/msg.component';
19. import { NavbarComponent } from '../navbar/navbar.component';
20. import { CoreModule } from '../core/core.module';
21. @NgModule({
22.   declarations: [
23.     AppComponent,
24.     AuthenticationComponent,
25.     LoginComponent,
26.     SingoutComponent,
27.     SignupComponent,
28.     HomeComponent,
29.     NotFoundPageComponent,
30.     UsersComponent,
31.     UserDetailsComponent,
32.     UsersListComponent,
33.     ProfileComponent,
34.     EditCoursesComponent,
35.     MsgComponent,
36.     NavbarComponent
37.   ],
38.   imports: [
39.     BrowserModule,
40.     AppRoutingModule,
41.     FormsModule,
42.     ReactiveFormsModule,
43.     UsersModule,
44.     CoreModule
45.   ],
46.   providers: [],
47.   bootstrap: [AppComponent]
48. })
```

```
49. export class AppModule { }
```

نلاحظ انه قام بتحديث Root Module بشكل تلقائي وأضاف CoreModule في قائمة imports كما هو موجود في السطر 44 وايضاً قام باستدعائه كما هو موجود في السطر 20.

وايضاً لنقوم باستعراض محتويات ملف core.module.ts، كالتالي:

ملف core.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';

3. @NgModule({
4.   declarations: [],
5.   imports: [
6.     CommonModule
7.   ]
8. })
9. export class CoreModule { }
10.
```

نلاحظ ان هذا الملف هو عبارة عن Module عادي يحتوي على imports و declarations بالإضافة إلى خاصية أخرى اسمها exports سوف نتكلم عنها في وقت لاحق بإذن الله.

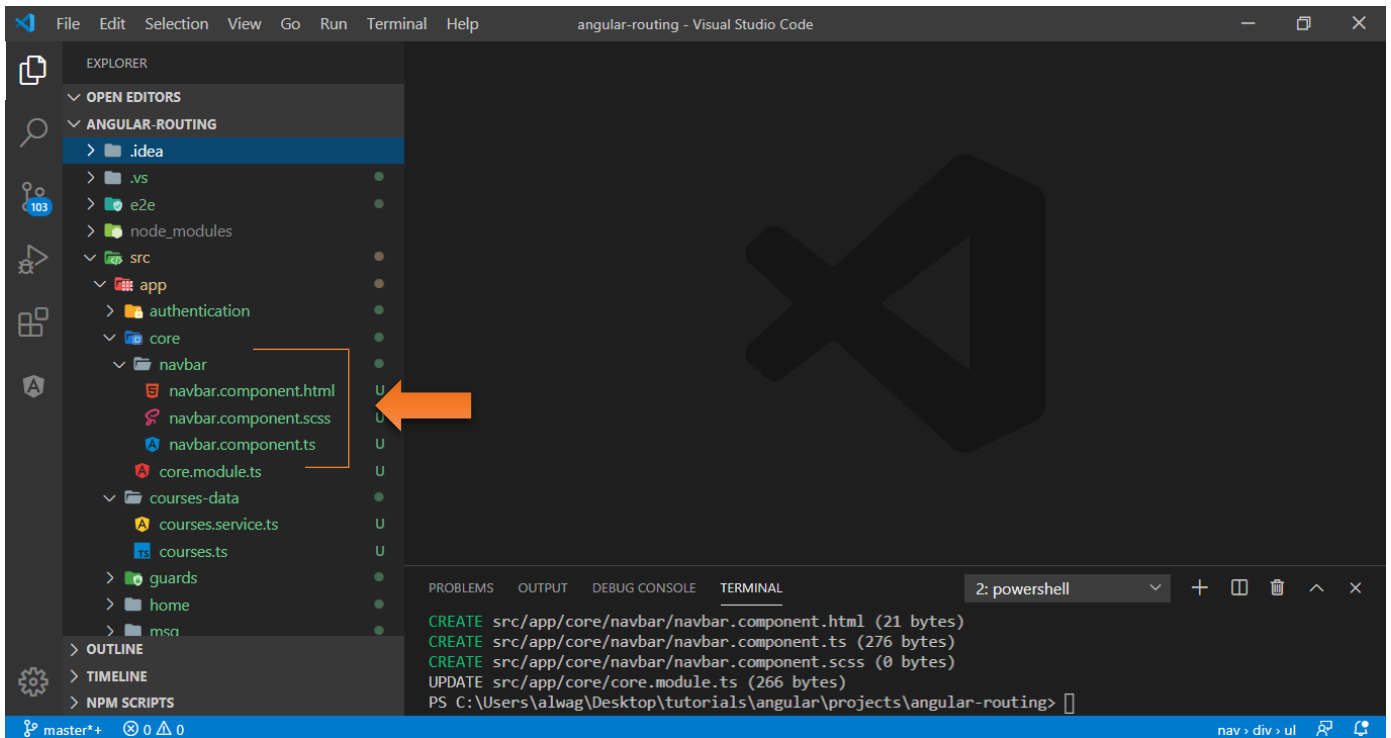
اما الآن بعد ان قمنا بإنشاء Core Module لنقوم بإنشاء component جديد ولنسميه navbar ونقوم بنقل جميع Markup الخاص بعرض navbar من AppComponent إلى هذا Component الجديد، بحيث ان Core Module يحتوي هذا component، ويتم ذلك من خلال كتابة الامر التالي:

```
ng g c core/navbar --m core/core.module.ts --skip-tests
```

ملاحظة: في حال كان هنالك Module بنفس المسار path الذي تم انشاء به component فلا يحتاج إلى ان نُحدد لـ angular أي Module ينتهي إليه هذا component، لأن angular ذكي وسوف يعرف Module ويضيف إليه هذا component، لذلك سوف نعيد كتابة الامر السابق في terminal ليصبح بهذا الشكل:

```
ng g c core/navbar --skip-tests
```

نلاحظ اننا لم نضيف الامر flat— لأننا نريد ان يضيف مجلد فرعي تحت المجلد core اما الامر skip-tests— فالغرض منه ان لا يُنشأ لنا ملف Test.



لنستعرض محتويات ملف `core.module.ts`، لنرى ماذا حدث به بعدما قمنا بإضافة هذا component، كالتالي:

```
ملف core.module.ts
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { NavbarComponent } from './navbar/navbar.component';
4.
5. @NgModule({
6.   declarations: [ NavbarComponent ],
7.   imports: [ CommonModule ]
8. })
9. export class CoreModule { }
```

نلاحظ انه أضاف component الجديد إلى مصفوفة التعريفات بشكل تلقائي.

الآن لنقوم بنقل جميع ال Marckup الخاص بعرض navbar من `app.component.html` وما يلحقه من اكواد في ملف `app.component.ts` و `app.component.scss` في ملف `app.component.html` إلى ملفات `navbar.component.html` و `navbar.component.ts` و `navbar.component.scss`، كالتالي:

```
ملف navbar.component.html
1. <nav class="navbar navbar-expand-sm navbar-dark bg-dark Shadow">
2.   <div class="navbar-brand-two" href="#">
3.     
4.   </div>
5.   <span class="navbar-brand">Routing</span>
6.   <div class="justify-content-left">
7.     <ul class="navbar-nav">
8.       <li class="nav-item">
9.         <a
10.           routerLink="/home"
11.           routerLinkActive="is-active">
```



```

12.         [routerLinkActiveOptions]="{ exact: true }">
13.             <span>Home</span>
14.         </a>
15.     </li>
16.     <li class="nav-item" *ngIf="IsAdmin && IsSignedIn">
17.         <a routerLink="home/edit-courses" = "is-active">
18.             <span>Edit Courses</span>
19.         </a>
20.     </li>
21.     <li class="nav-item" *ngIf="IsAdmin && IsSignedIn">
22.         <a routerLink="/users" routerLinkActive="is-active">
23.             <span>Users</span>
24.         </a>
25.     </li>
26.     <li class="nav-item" *ngIf="IsSignedIn">
27.         <a [routerLink]="['profile', id]" routerLinkActive="is-active">
28.             <span>Profile</span>
29.         </a>
30.     </li>
31.     <li class="nav-item" *ngIf="!IsSignedIn">
32.         <a routerLink="/auth/login" routerLinkActive="is-active">
33.             <span>Login</span>
34.         </a>
35.     </li>
36.     <li class="nav-item" *ngIf="!IsSignedIn">
37.         <a routerLink="/auth/signup" routerLinkActive="is-active">
38.             <span>Signup</span>
39.         </a>
40.     </li>
41.     <li class="nav-item" *ngIf="IsSignedIn">
42.         <a routerLink="/auth/signout" routerLinkActive="is-active">
43.             <span>Signout</span>
44.         </a>
45.     </li>
46. </ul>
47. </div>
48. </nav>

```

ملف navbar.component.ts

```

1. import { Component, OnInit } from '@angular/core';
2. import { UsersService } from '../users-data/users.service';
3. import { StorageMap } from '@ngx-pwa/local-storage';
4.
5. @Component({
6.     selector: 'app-navbar',
7.     templateUrl: './navbar.component.html',
8.     styleUrls: ['./navbar.component.scss']
9. })
10. export class NavbarComponent implements OnInit {
11.
12.     IsSignedIn = false;
13.     IsAdmin = false;

```



```

14. id: string;
15.
16. constructor(
17.     private userService: UsersService,
18.     private storageMap: StorageMap,
19. ) { }
20.
21. ngOnInit(): void {
22.     this.userService.isAdmin$.subscribe(val => {
23.         this.IsAdmin = val;
24.     });
25.     this.userService.isLogIn$.subscribe(val => {
26.         this.IsSignedIn = val;
27.     });
28.     this.storageMap.watch('id').subscribe((id: string) => {
29.         this.id = id;
30.     });
31. }
32.
33.}

```

ملف navbar.component.scss

```

1. .Shadow {
2.     box-shadow: 0 0 0 1.2px grey;
3. }
4.
5. .is-active {
6.     background-color: gray;
7.     font-weight: bold;
8.     border: 1px solid rgb(27, 26, 26);
9. }
10.
11. .is-active span {
12.     border-bottom-style: solid;
13.     border-bottom-width: 3px;
14.     border-bottom-color: #84ff6e;
15.     padding-bottom: 4px;
16. }
17.
18. li a {
19.     color: white;
20.     padding: 10px;
21.     margin: 4px;
22. }
23.
24. li a:hover {
25.     text-decoration: none;
26. }

```

ملف app.component.html

```

1. <app-navbar></app-navbar>
2.

```

```

3. <div class="spinner" *ngIf="showSpinner"></div>
4.
5. <router-outlet name="msg"></router-outlet>
6.
7. <router-outlet></router-outlet>

```

التغيير الوحيد هو ما هو موجود في السطر 1 حيث قمنا بنقل markup الذي كان هنا إلى ملف

navbar.component.html ووضعنا مكانه selector الخاص بهذا component وهو <app-navbar></app-navbar>

ملف app.component.ts

```

1. import { Component, OnInit } from '@angular/core';
2. import {
3.   Router,
4.   Event,
5.   NavigationStart,
6.   NavigationEnd,
7.   NavigationCancel
8. } from '@angular/router';
9.
10. @Component({
11.   selector: 'app-root',
12.   templateUrl: './app.component.html',
13.   styleUrls: ['./app.component.scss']
14. })
15. export class AppComponent implements OnInit {
16.
17.   showSpinner = false;
18.
19.   constructor(private router: Router) {
20.     this.router.events.subscribe((event: Event) => {
21.       if (event instanceof NavigationStart) {
22.         this.showSpinner = true;
23.       }
24.       if (event instanceof NavigationEnd || event instanceof NavigationCancel) {
25.         this.showSpinner = false;
26.       }
27.     });
28.   }
29.
30.   ngOnInit(): void { }
31.
32. }

```

ملف app.component.scss

```

1. .spinner {
2.   position: fixed;
3.   width: 100%;
4.   left: 0;
5.   right: 0;
6.   top: 0;
7.   bottom: 0;

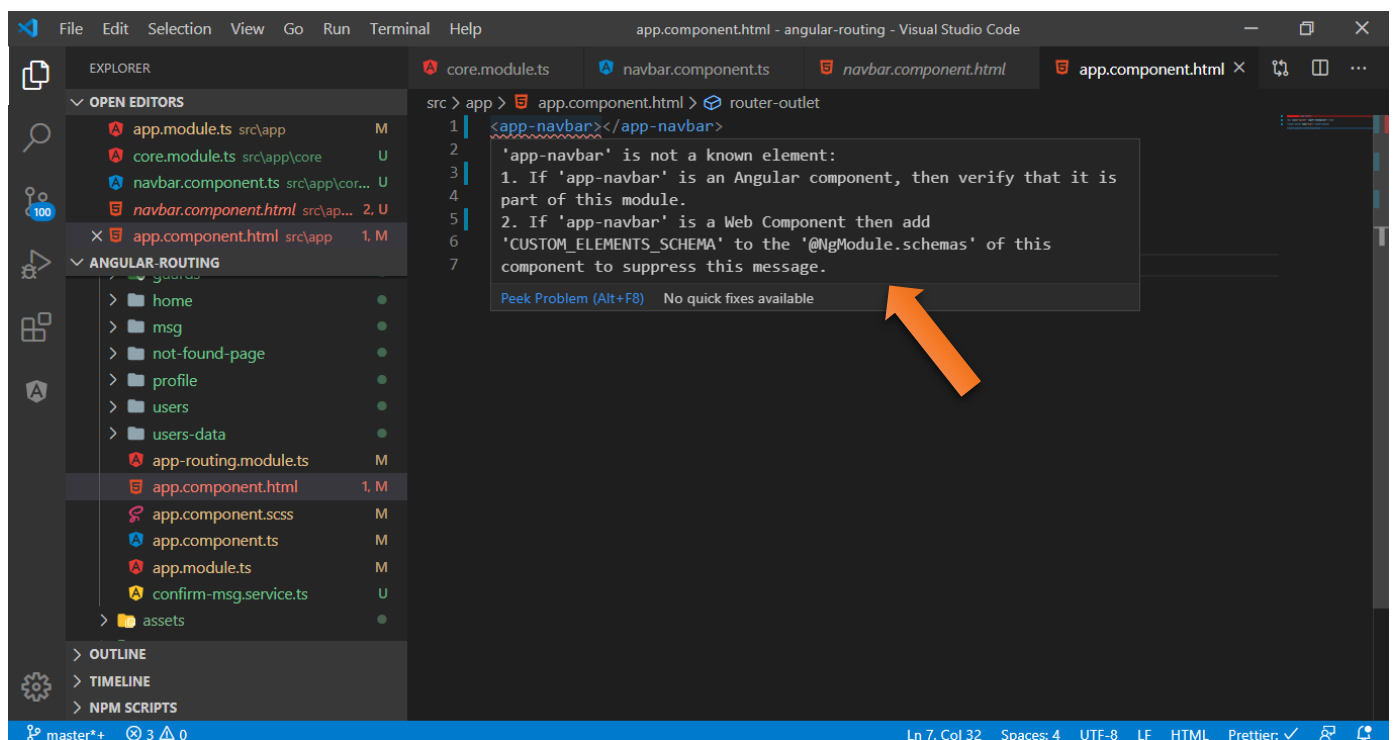
```

```

8.   background-color: rgba(240, 240, 240, 0.5);
9.   z-index: 1;
10.}
11.
12..spinner::after {
13.  content: "";
14.  display: block;
15.  position: absolute;
16.  border: 16px solid silver;
17.  border-top: 16px solid #337ab7;
18.  border-radius: 50%;
19.  width: 80px;
20.  height: 80px;
21.  animation: spin 700ms linear infinite;
22.  top: 50%;
23.  left: 50%;
24.}
25.
26.@keyframes spin {
27.  0% {
28.    transform: rotate(0deg);
29.  }
30.  100% {
31.    transform: rotate(360deg);
32.  }
33.}

```

الآن لنقم بحفظ التعديلات ولكن قبل ان نذهب إلى المتصفح لنرجع إلى ملف app.component.html، لنرى ان selector تحته خط احمر وعند تمرير مؤشر الفأرة فوقه لمعرفة نوع هذا الخطأ سوف تظهر لنا هذه الرسالة:



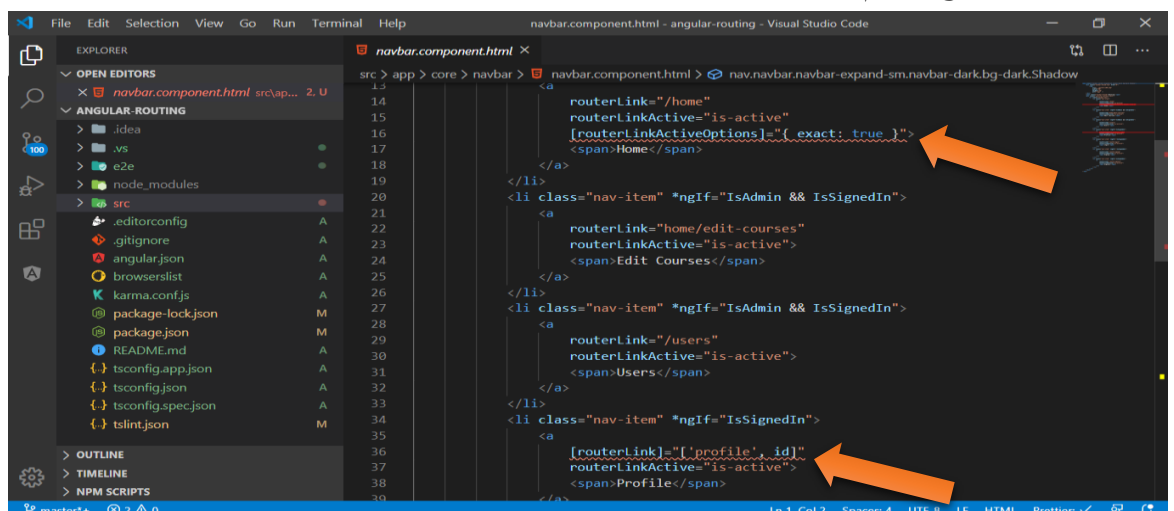
ومعنى هذه الرسالة أن angular لا يستطيع معرفة هذا selector فهو غير معروف لديه، مع العلم ان هذا selector يشير إلى NavbarComponent وتم تعريفه في CoreModule وهذا Module تم تعريفه في Root Module ولكن رغم ما قمنا بعمله لكن إلا الآن Angular لم يستطع التعرف عليه، وتكمن المشكلة في ملف core.module.ts، حيث اننا قمنا بتعريف هذا component بداخله ولكن لم نعمل له exports، لكي يُرى على مستوى كامل المشروع، فالبديل لا يستطيع ان يرى إلا الذي نعمل له exports في Modules الأخرى، لذلك لنذهب إلى ملف core.module.ts ونضيف الخاصية exports ونسند إليها مصفوفة تحتوي على جميع ما نريد ان نعمل له exports وفي مثالنا هنا NavbarComponent فقط، كالتالي:

```
Core.module.ts

1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { NavbarComponent } from '../navbar/navbar.component';

4. @NgModule({
5.   declarations: [
6.     NavbarComponent
7.   ],
8.   imports: [
9.     CommonModule,
10.  ],
11.  exports: [
12.    NavbarComponent
13.  ]
14.})
15.
16. export class CoreModule { }
```

نلاحظ اننا قمنا بعمل export لNavbarComponent في الاسطر من 11 إلى 13. الآن لنحفظ التعديلات ولكن قبل الذهاب إلى المتصفح لنقوم بالذهاب إلى ملف navbar.component.html، سوف نرى انه يوجد ايضاً أخطاء، كالتالي:



ولو حاولت معرفة نوع الخطأ هنا لوجدت انه يشير إلى ان هذه الخصائص غير معرفة لديه ولا تنتهي إلى العنصر <a>، وسبب هذا الخطأ ان جميع هذه الخصائص موجودة من ضمن Module اسمه RouterModule وهذا Module موجود في Root Module ولكن بما اننا عزلنا هذا component في CoreModule فإنه أصبح لا يستطيع ان يرى إلا الـ Modules التي نعمل لها import في CoreModule فقط، لذلك لا بد ان نعمل له استدعاء، كالتالي:

ملف core.module.ts

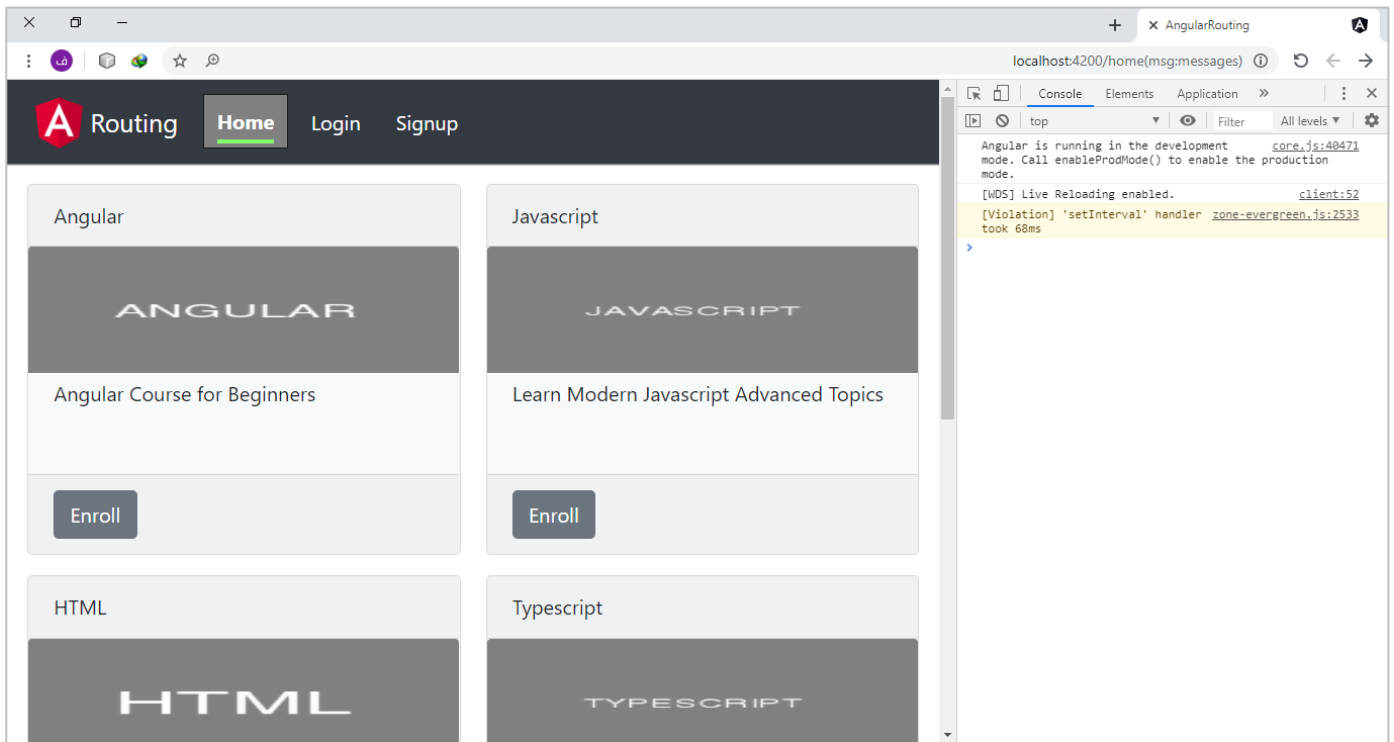
```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { NavbarComponent } from '../navbar/navbar.component';
4. import { RouterModule } from '@angular/router';

5. @NgModule({
6.   declarations: [NavbarComponent],
7.   imports: [
8.     CommonModule,
9.     RouterModule
10.  ],
11.  exports: [
12.    NavbarComponent
13.  ]
14.})
15. export class CoreModule { }
```

نلاحظ اننا قمنا في السطر 9 بعمل import لـ RouterModule واستدعيناها في السطر 4، والشيء بالشئ يذكر نلاحظ انه يوجد Module آخر اسمه CommonModule وهو موجود بشكل تلقائي عندما قمنا بإنشاء هذا Module وهو يحتوي على اغلب الدايركتيف الجاهزة مثل \*ngIf و NgClass و...الخ، ولو قمنا بالإلغاء من هنا لظهر لنا خطأ في ملف navbar.component.html مفاده ان \*ngIf غير معروف.

وبذلك نستطيع ان نضع قاعدة عامة وهي ان أي component او pipe او directive عملنا له تعريف declarations فإنه لا يرى إلى داخل هذا Module إلا ان نقوم بعمل له export، وبنفس الوقت أي كود استخدمناه في هذه components او Pipes او Directives وكان يعتمد على Module ما (مثل RouterModule) لا بد ان نعمل له import، وسوف نلاحظ ان هذه Modules مثل (RouterModule-FormsModule-CommonModule-..etc) سوف تتكرر لدينا في اغلب الأنواع الأخرى من Modules التي نضيفها، وسوف نقوم بحل مشكلة التكرار بإذن الله عندما نتكلم عن Shared Module.

الآن لنقم بحفظ التعديلات، ولكن هذه المرة لنذهب إلى المتصفح لنرى النتيجة:



نلاحظ ان المشروع يعمل بدون أي مشاكل او أخطاء.

وبذلك نكون انشأنا اول Core Module، وكما قلنا سابقاً ان هذا الـ Module نضيف فيه الـ components والـ Pipes والـ Directives وحتى Modules التي يتم استدعائها مرة واحدة فقط في Root Module، لذلك سوف نقوم بزيادة الجرعة ووضع AppRoutingModuleModule في Core Module (بما انه يتم استدعائه مره واحدة فقط في Root Module)، كالتالي:

ملف core.module.ts

```
1. import { NavbarComponent } from './navbar/navbar.component';
2. import { RouterModule } from '@angular/router';
3. import { AppRoutingModuleModule } from '../app-routing.module';
4.
5. @NgModule({
6.   declarations: [NavbarComponent],
7.   imports: [
8.     CommonModule,
9.     RouterModule,
10.    AppRoutingModuleModule
11.  ],
12.  exports: [
13.    NavbarComponent,
14.    AppRoutingModuleModule
15.  ]
16.})
17. export class CoreModule { }
```

ولا ننسى ان نعمل له export لكي يُرى على كامل المشروع لأن الكلاسات الموجودة فيه مستخدم في أكثر من ملف من ملفات المشروع، ولا ننسى ان نحذف AppRoutingModuleModule في Root Module والاستدعاء الخاص به، وكنوع من التدريب قم عزيزي المتعلم بنقل HomeComponent من Root Module إلى Core Module.

#### 4.4. Feature Module & Routing Module

وهذا النوع من اهم الأنواع وأكثرها تأثير على أداء التطبيق وايضاً اجادة هذا النوع يرفع من سرعة التطبيق وفي سهولة التطوير والصيانة للتطبيق مستقبلاً.

إذن ما هو هذا النوع؟ هذا النوع يرجع بالأساس إلى بنية مشروع Angular الخاص بك، فكما هو معروف أن أي مشروع Angular يتكون من مجموعة من components وغيره من الملفات، وجميع هذه الأجزاء تربط بينها علاقة ما، فمثلاً في مثالنا نجد أن LoginComponent و SingupComponent و LogoutComponent جميع هذه الأجزاء تربط بينها علاقة التعامل مع المستخدمين من ناحية تسجيل الدخول والخروج وتسجيل مستخدم جديد، اما لو استعرضنا UsersComponent والComponents التي تدرج تحت هذا component لوجدنا ان العلاقة التي تربط بينهم هي التعامل مع المستخدمين من ناحية حذف وازافة وتعديل واستعراض بيانات المستخدمين من قبل مسئول النظام، وهكذا بقية أجزاء المشروع، لذلك بما ان هذه الأجزاء ترتبط بعلاقة فما المانع بإذن نضع لها Module خاص بها ونربط هذا Module في Root Module كما كنا نفعل مع Core Module، وفي هذه الحالة يسمى هذا النوع Feature Module.

وايضاً هذه components عند بنائها بكل تأكيد استعنا بمجموعة من الكلاسات والدايكرتيف والبايب سواء كانت مبنية ضمناً او نحن قمنا بإنشائها لأداء مهام معينة، لذلك وجب عليك ان نعرف هذه الأنواع وهل هي متكررة في اكثر من جزء من أجزاء المشروع، وفي حال كانت متكررة فقط في الأجزاء التي بينها علاقة ما فتكتفي باستدعائها بـ Feature Module الخاص بهذه components اما إذا كان تم استخدامها في أجزاء أخرى فنضعها في Shared Module والتي سوف اتطرق اليها لاحقاً بإذن الله.

كما يتم انشاء بالتوازي مع كل Feature Module الـ Routing Module الخاص به.

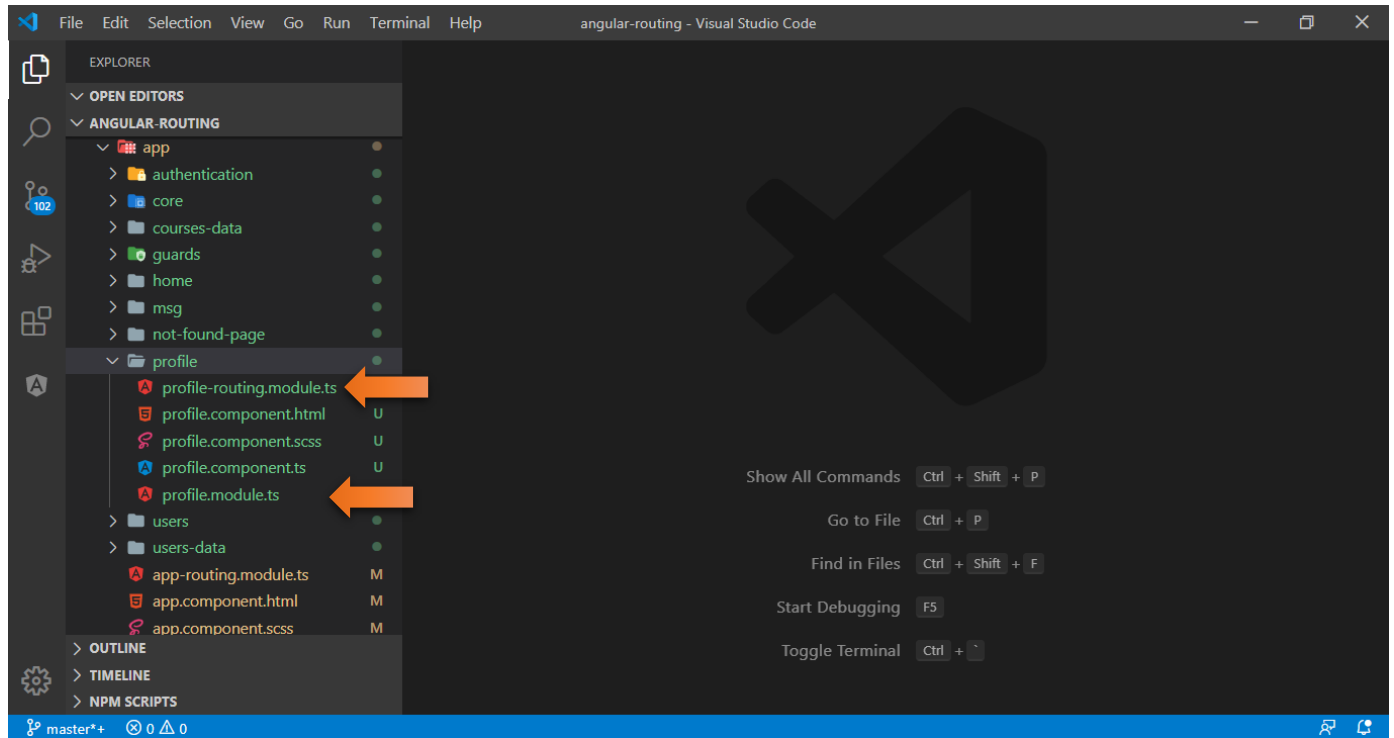
اذن نستطيع ان نقول ان هذا النوع يتم التخطيط له في بداية بناء المشروع ولا بد من المعرفة الجيدة بالأنواع المختلفة التي تربطها علاقة ما بحيث نضع لها module خاص بها وبذلك نقوم بتقسيم المشروع إلى أجزاء صغيرة يسهل تطويرها وصيانتها ولا ننسى بالتأكيد رفع كفاءة وأداء التطبيق. ولكن نحن هنا في هذا الكتاب قمنا في البداية ببناء المشروع ومن ثم اضعنا اليه Modules، وهذا خطأ ولكن كما اشرت سابقاً في حال اردت بناء أي مشروع مستقبلاً لا بد ان تخطط لـ Module كما تخطط لبقية أجزاء المشروع وفي حال كان التخطيط جيد فهذا سينعكس بالتأكيد على المشروع الخاص بك.

والآن بعد هذه المقدمة سوف ننتقل إلى الجزء العملي ونقوم بإنشاء Feature Module، ولنبدأ في البداية بـ ProfileComponent بما انه وحيد ولا يرتبط بأي أبناء، فسوف نقوم ببناء Feature Module له، ونقوم في البداية بتوجه إلى terminal وكتابة الأمر التالي:

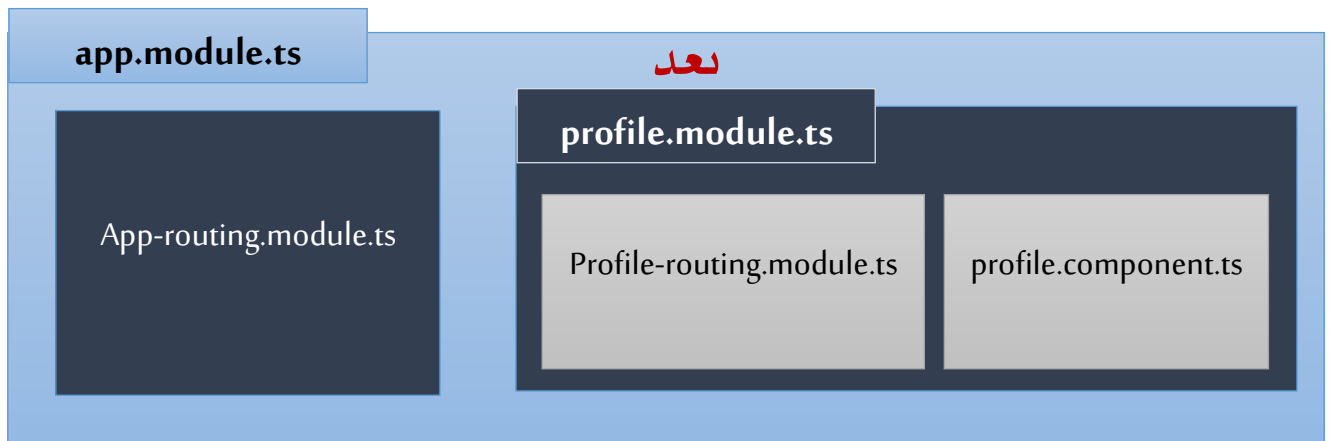
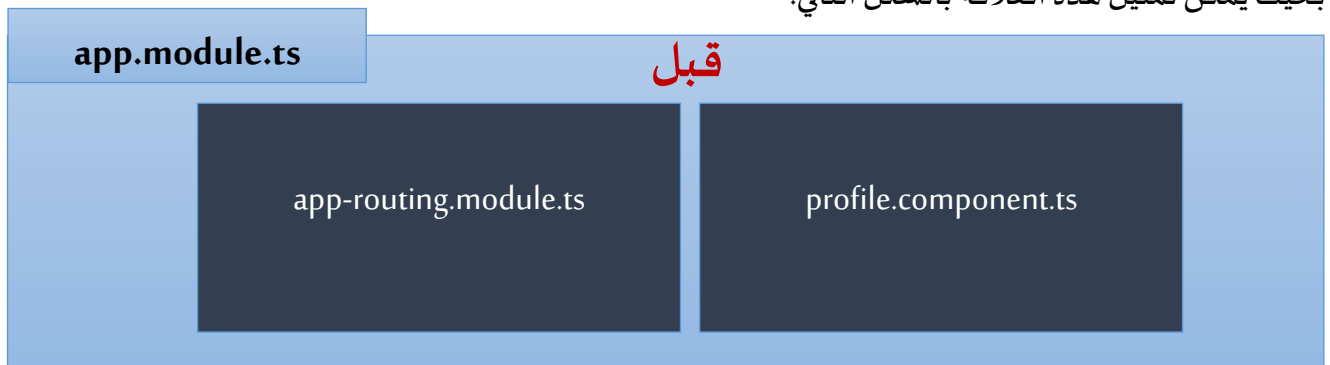
```
ng g m profile/profile --flat --routing --module app.module.ts
```

ومعنى هذا الامر قم بإنشاء Module باسم profile تحت مجلد profile وقم بإنشائه بدون لا تنشأ مجلد فقط ملف Module وبنفس الوقت قم بإنشاء ملف routing يحمل نفس الاسم واخيراً قم بتحديث ملف app.module.ts أي قم بإضافة تعريف لهذا Module الجديد في Root Module.

بعدها نضغط Enter من لوحة المفاتيح وسوف يقوم بإنشاء هذا Module بالإضافة إلى Routing Module، كالتالي:



بحيث يمكن تمثيل هذه العلاقة بالشكل التالي:





نلاحظ اننا قمنا بإعادة بناء هذا component من الشكل الأول إلى الشكل الثاني، الآن لنستعرض محتويات هذين الملفين، كالتالي:

ملف profile-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3.
4. const routes: Routes = [];
5.
6. @NgModule({
7.   imports: [RouterModule.forChild(routes)],
8.   exports: [RouterModule]
9. })
10. export class ProfileRoutingModule { }
```

نلاحظ انه مشابه لـ app-routing.module.ts ولكن الفرق الوحيد هنا اننا استخدمنا الدالة `forChild` ومررنا لها مصفوفة التهيئة الخاصة بـ `Routes` كما هو موجود في السطر 7، بعكس الملف الأول الذي استخدمنا فيه الدالة `forRoot` وبالمختصر نستطيع ان نقول ان الدالة `forRoot` يتم استخدامها مرة واحدة فقط في ملف تهيئة `Routes` الرئيسي وبشكل افتراضي هو ملف `app-routing.module.ts` اما الدالة `forChild` فيتم استخدامها في أي ملف فرعي لتهيئة `Routes`، كما هو موجود في ملف `profile-routing.module.ts`.

ملف profile.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3.
4. import { ProfileRoutingModule } from './profile-routing.module';
5.
6. @NgModule({
7.   declarations: [],
8.   imports: [
9.     CommonModule,
10.    ProfileRoutingModule
11.  ]
12. })
13.
14. export class ProfileModule { }
15.
```

اما هذا الملف فيعتبر `Feature Module` حيث يعتبر `Module` مصغر لـ `Root Module` ويتعامل مع `ProfileComponent` وايضاً مع `ProfileRoutingModule`، وبما ان component الذي يتعامل معه هذا `Module` لم نستخدم فيه أي `Directives` جاهزة مثل `*ngIf` او `NgClass` او... الخ، فبذلك لا نحتاج إلى `CommonModule` الذي تم استدعائه بشكل افتراضي عندما تم انشاء هذا الملف ونستطيع الاستغناء عنه، كالتالي:

ملف profile.module.ts

```
16. import { NgModule } from '@angular/core';
```

```

17.
18.import { ProfileRoutingModule } from './profile-routing.module';
19.
20.@NgModule({
21.  declarations: [],
22.  imports: [
23.    ProfileRoutingModule
24.  ]
25.})
26.
27.export class ProfileModule { }

```

وايضاً لو استعرضنا الـ Root Module لوجدنا انه تم تعريف هذا Feature Module (ProfileModule) بشكل تلقائي،  
كالتالي:

ملف app.module.ts

```

1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { FormsModule, ReactiveFormsModule } from '@angular/forms';
4.
5. import { AppComponent } from './app.component';
6. import { AuthenticationComponent } from './authentication/authentication.component';
7. import { LoginComponent } from './authentication/login/login.component';
8. import { SingoutComponent } from './authentication/singout/singout.component';
9. import { SignupComponent } from './authentication/signup/signup.component';
10.import { HomeComponent } from './home/home.component';
11.import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
12.import { UsersComponent } from './users/users.component';
13.import { UserDetailsComponent } from './users/user-details/user-details.component';
14.import { UsersListComponent } from './users/users-list/users-list.component';
15.import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
16.import { MsgComponent } from './msg/msg.component';
17.import { CoreModule } from './core/core.module';
18.import { ProfileModule } from './profile/profile.module';
19.
20.@NgModule({
21.  declarations: [
22.    AppComponent,
23.    AuthenticationComponent,
24.    LoginComponent,
25.    SingoutComponent,
26.    SignupComponent,
27.    HomeComponent,
28.    NotFoundPageComponent,
29.    UsersComponent,
30.    UserDetailsComponent,
31.    UsersListComponent,
32.    EditCoursesComponent,
33.    MsgComponent
34.  ],
35.  imports: [

```

```

36.   BrowserModule,
37.   FormsModule,
38.   ReactiveFormsModule,
39.   CoreModule,
40.   ProfileModule,
41. ],
42. providers: [],
43. bootstrap: [AppComponent]
44.})
45.export class AppModule { }
46.

```

راجع الاسطر 18 و40.

الآن بقي لدينا بعض الخطوات البسيطة وهي ان نقوم بنقل تهيئة Route الخاص بـ component ذو الاسم ProfileComponent من AppRoutingModuleModule إلى ProfileRoutingModule، كالتالي:

ملف profile-routing.module.ts

```

1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { CanActivateGuard } from '../guards/can-activate.guard';
4. import { ProfileComponent } from './profile.component';
5.
6. const routes: Routes = [
7.   {
8.     path: 'profile/:id',
9.     canActivate: [CanActivateGuard],
10.    component: ProfileComponent
11.  }
12. ];
13.
14. @NgModule({
15.   imports: [RouterModule.forChild(routes)],
16.   exports: [RouterModule]
17. })
18. export class ProfileRoutingModule { }
19.

```

وفي ملف app-routing.module.ts لابد ان يكون استدعاء ProfileModule قبل CoreModule، كالتالي:

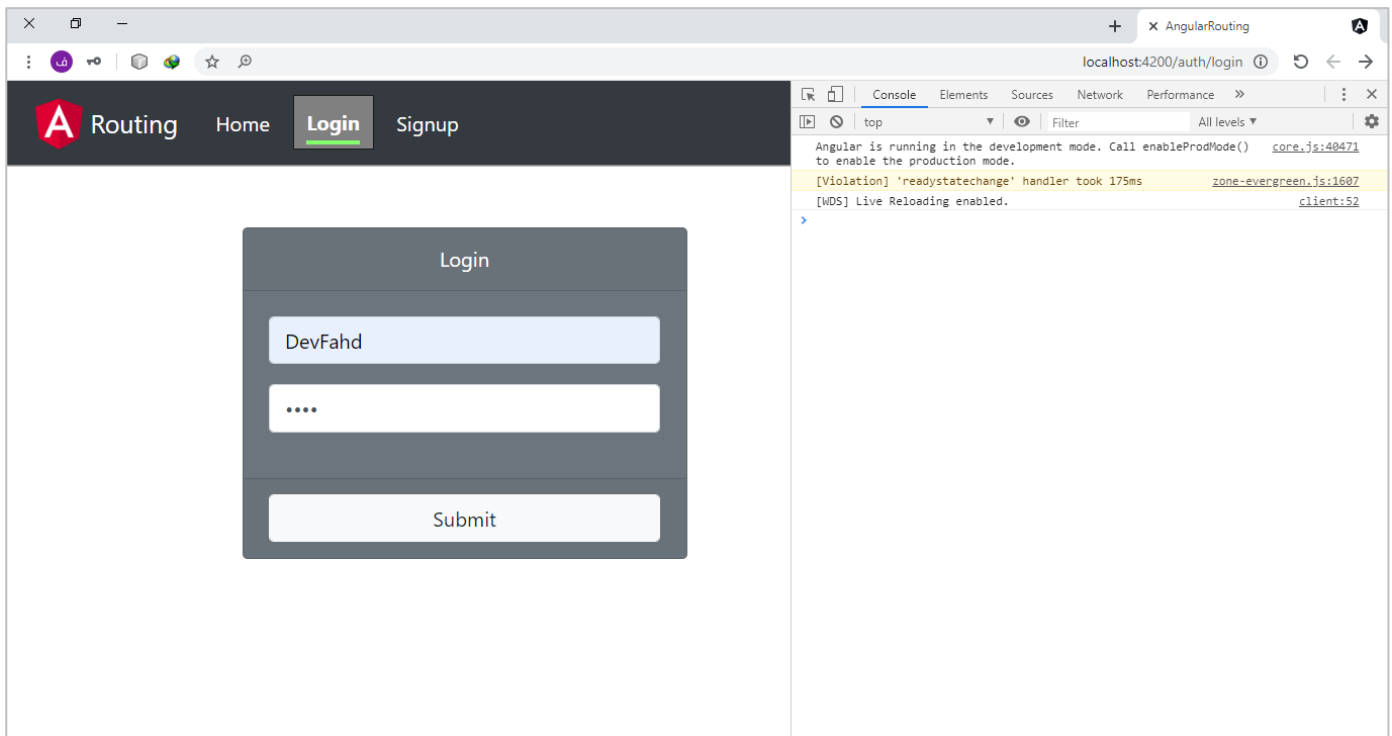


نقوم بوضع استدعاء ProfileModule قبل CoreModule وإلا فلن يعمل التطبيق

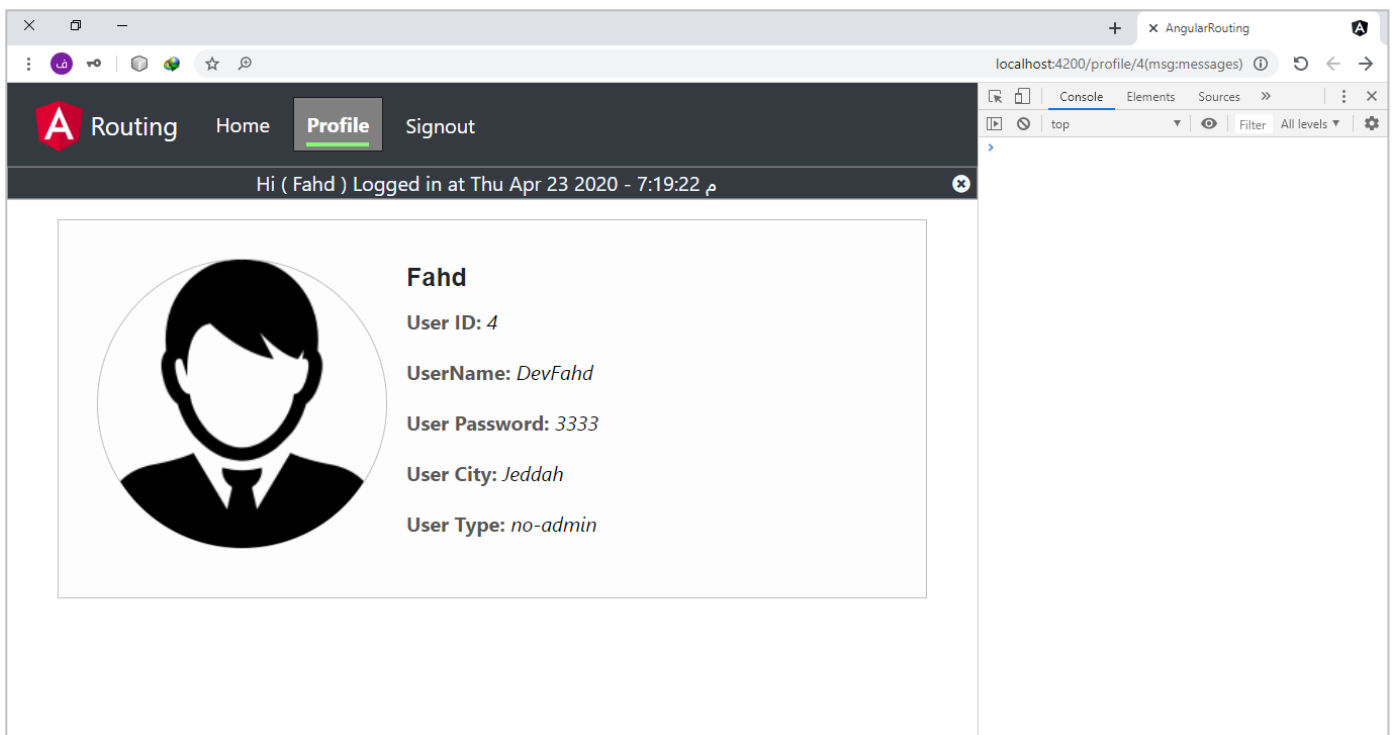
والسبب في ذلك أن CoreModule يحتوي بداخله على AppRoutingModuleModule والذي بدوره يحتوي على Wild Card Route ("\*\*")، أما ProfileModule يحتوي بداخله على ProfileRoutingModule والذي بدوره يحتوي على Route ذو Path الذي يحمل القيمة "profile/:id"، لذلك اذا اسنمر الوضع على هذا الترتيب في الاستدعاءات فانه سوف يُنفذ Wild Card Route في الأول، والذي ينافي ما قلناه سابقاً أنه يجب ان يكون آخر جزء في تهيئة Routes، ومن هذا المنطلق لابد ان نقوم

بإعداد ترتيب الاستدعاءات بوضع ProfileModule قبل CoreModule وهذا الكلام ينطبق على بقية Feature Module التي سوف ننشئها بعد قليل.

الآن لنقم بحفظ التعديلات ونذهب إلى المتصفح ونرى النتيجة، كالتالي:



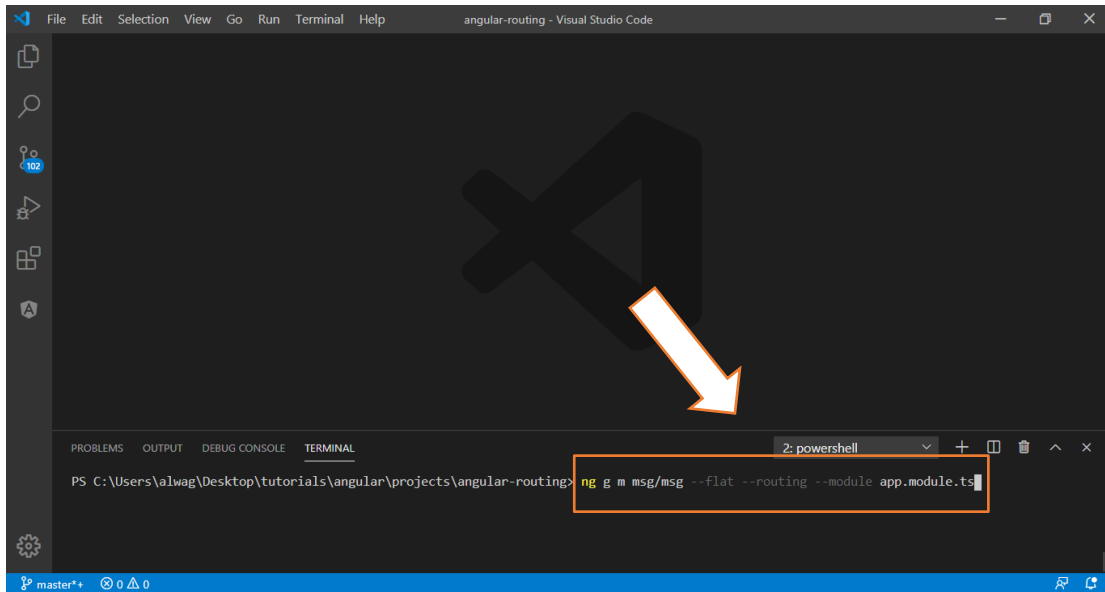
في البداية نقوم بتسجيل الدخول ومن ثم نذهب إلى التبويب Profile، كالتالي:



نلاحظ انه يعمل بدون أي مشاكل.

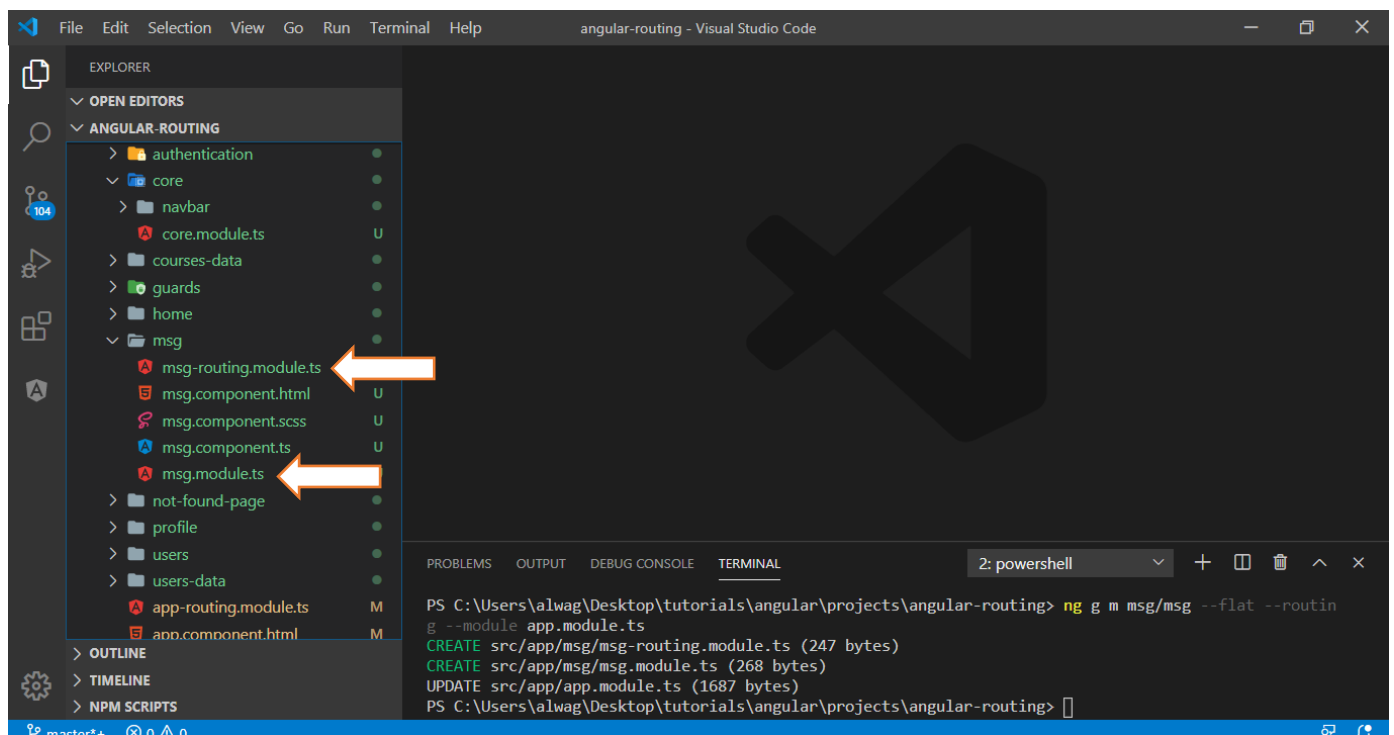
ونستطيع ان نقول الآن مبروك عزيزي المتعلم لقد أنشأت اول Feature Module، ولكن قد يتبادر إلى ذهنك سؤال وهو لماذا لم نقوم بعمل export لـ ProfileModule كما فعلنا لـ NavbarComponent في CoreModule، او بالأصح لماذا لم نقوم بالأساس باستدعاء ProfileComponent في ProfileModule، والاجابة عن هذا السؤال بسيطة جداً ولو دقت عزيزي المتعلم في طريقة بناء هذا Feature Module لعرفت الإجابة، وهي ان ProfileComponent تم استدعائه في ProfileRoutingModule وتم عمل export لهذا الModule بجميع محتوياته والذي تم استدعائه هو الآخر في ProfileModule، لذلك ليس من المنطقي ان نقوم بتكرار الاستدعاء مرة أخرى.

الآن بنفس الخطوات لنقوم بعمل Feature Module لـ MsgComponent، حيث نقوم في البداية بكتابة الأمر التالي:



```
PS C:\Users\alwag\Desktop\tutorials\angular\projects\angular-routing> ng g m msg/msg --flat --routing --module app.module.ts
```

نضغط Enter وسوف يقوم بإنشاء الملفات المطلوبة، كالتالي:



```

EXPLORER
├── OPEN EDITORS
├── ANGULAR-ROUTING
│   ├── authentication
│   ├── core
│   ├── navbar
│   ├── core.module.ts
│   ├── courses-data
│   ├── guards
│   ├── home
│   ├── msg
│   │   ├── msg-routing.module.ts
│   │   ├── msg.component.html
│   │   ├── msg.component.scss
│   │   ├── msg.component.ts
│   │   └── msg.module.ts
│   ├── not-found-page
│   ├── profile
│   ├── users
│   ├── users-data
│   ├── app-routing.module.ts
│   └── app.component.html
├── OUTLINE
├── TIMELINE
└── NPM SCRIPTS

```

```

PS C:\Users\alwag\Desktop\tutorials\angular\projects\angular-routing> ng g m msg/msg --flat --routing --module app.module.ts
CREATE src/app/msg/msg-routing.module.ts (247 bytes)
CREATE src/app/msg/msg.module.ts (268 bytes)
UPDATE src/app/app.module.ts (1687 bytes)
PS C:\Users\alwag\Desktop\tutorials\angular\projects\angular-routing>

```

ومن ثم لنقوم بنقل التهيئة لـ Routed ذو path الذي يحمل الاسم msg والخاص بعرض MsgComponent من ملف app-routing.module.ts إلى ملف msg-routing.module.ts، كالتالي:

ملف msg-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { MsgComponent } from './msg.component';
4.
5. const routes: Routes = [
6.   { path: 'messages', outlet: 'msg', component: MsgComponent },
7. ];
8.
9. @NgModule({
10.  imports: [RouterModule.forChild(routes)],
11.  exports: [RouterModule]
12. })
13. export class MsgRoutingModule { }
14.
```

وأخيراً لا ننسى نضع استدعاء MsgModule قبل استدعاء CoreModule في ملف app.module.ts وبنفس الوقت نحذف استدعاء MsgComponent من هذا الملف، كالتالي:

ملف app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { FormsModule, ReactiveFormsModule } from '@angular/forms';
4.
5. import { AppComponent } from './app.component';
6. import { AuthenticationComponent } from './authentication/authentication.component';
7. import { LoginComponent } from './authentication/login/login.component';
8. import { SingoutComponent } from './authentication/singout/singout.component';
9. import { SignupComponent } from './authentication/signup/signup.component';
10. import { HomeComponent } from './home/home.component';
11. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
12. import { UsersComponent } from './users/users.component';
13. import { UserDetailsComponent } from './users/user-details/user-details.component';
14. import { UsersListComponent } from './users/users-list/users-list.component';
15. import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
16. import { MsgComponent } from './msg/msg.component';
17. import { CoreModule } from './core/core.module';
18. import { ProfileModule } from './profile/profile.module';
19. import { MsgModule } from './msg/msg.module';
20.
21. @NgModule({
22.  declarations: [
23.    AppComponent,
24.    AuthenticationComponent,
25.    LoginComponent,
26.    SingoutComponent,
27.    SignupComponent,
```

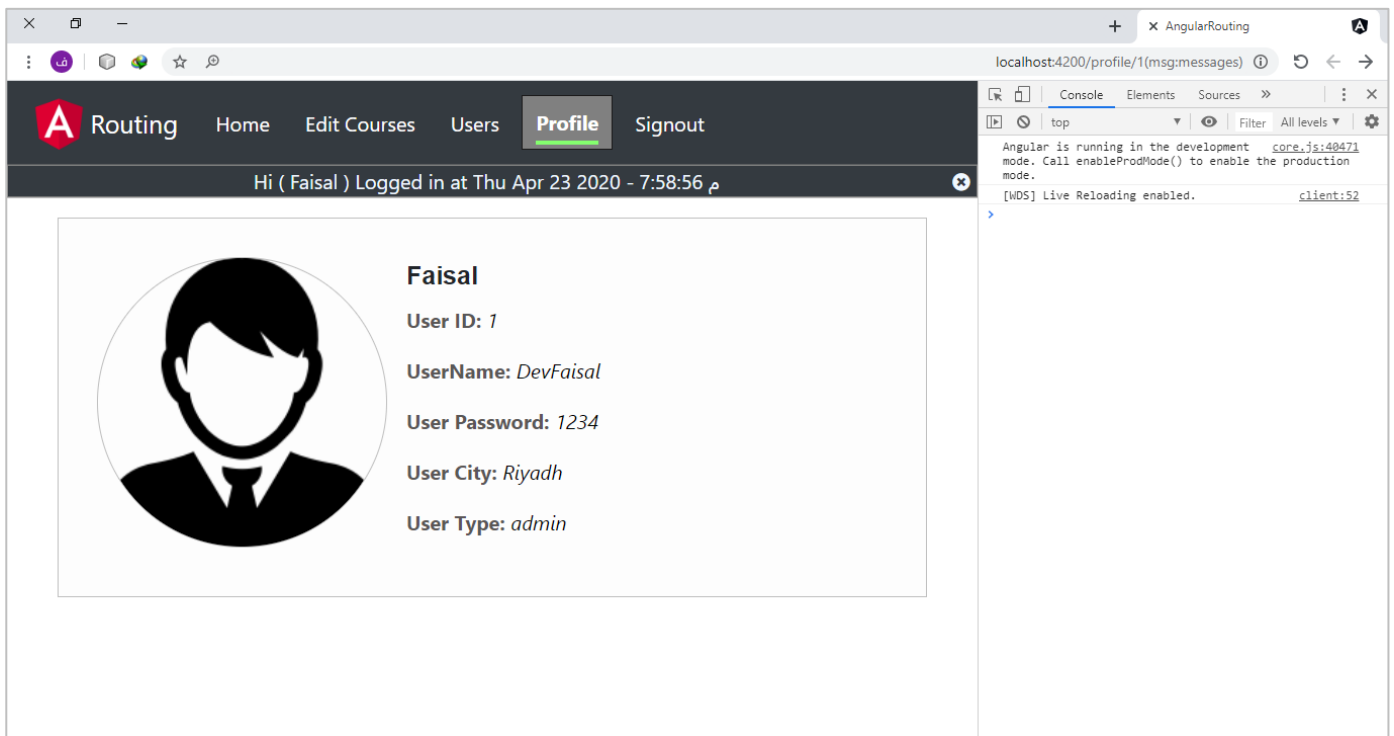
```

28. HomeComponent,
29. NotFoundPageComponent,
30. UsersComponent,
31. UserDetailsComponent,
32. UsersListComponent,
33. EditCoursesComponent,
34. MsgComponent
35. ],
36. imports: [
37.   BrowserModule,
38.   FormsModule,
39.   ReactiveFormsModule,
40.   ProfileModule,
41.   MsgModule,
42.   CoreModule,
43. ],
44. providers: [],
45. bootstrap: [AppComponent]
46.})
47.export class AppModule { }
48.

```

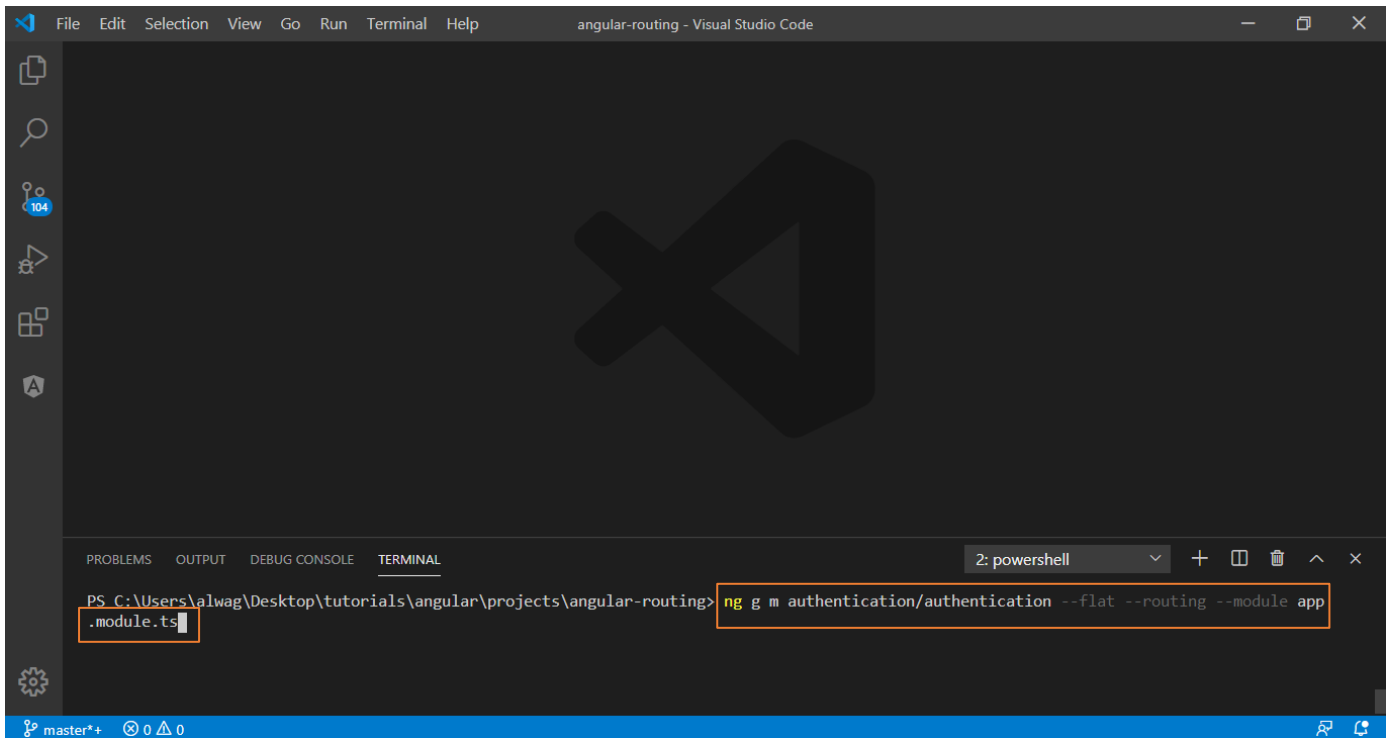
نحذف الاستدعاءات الموجودة في الاسطر 16 و 34 ونستدعي MsgModule قبل CoreModule كما هو موجود في الاسطر 41 و 42.

الآن لنقم بحفظ التعديلات ونذهب إلى المتصفح ولنقم بتسجيل الدخول ومن ثم نرى النتيجة، كالتالي:

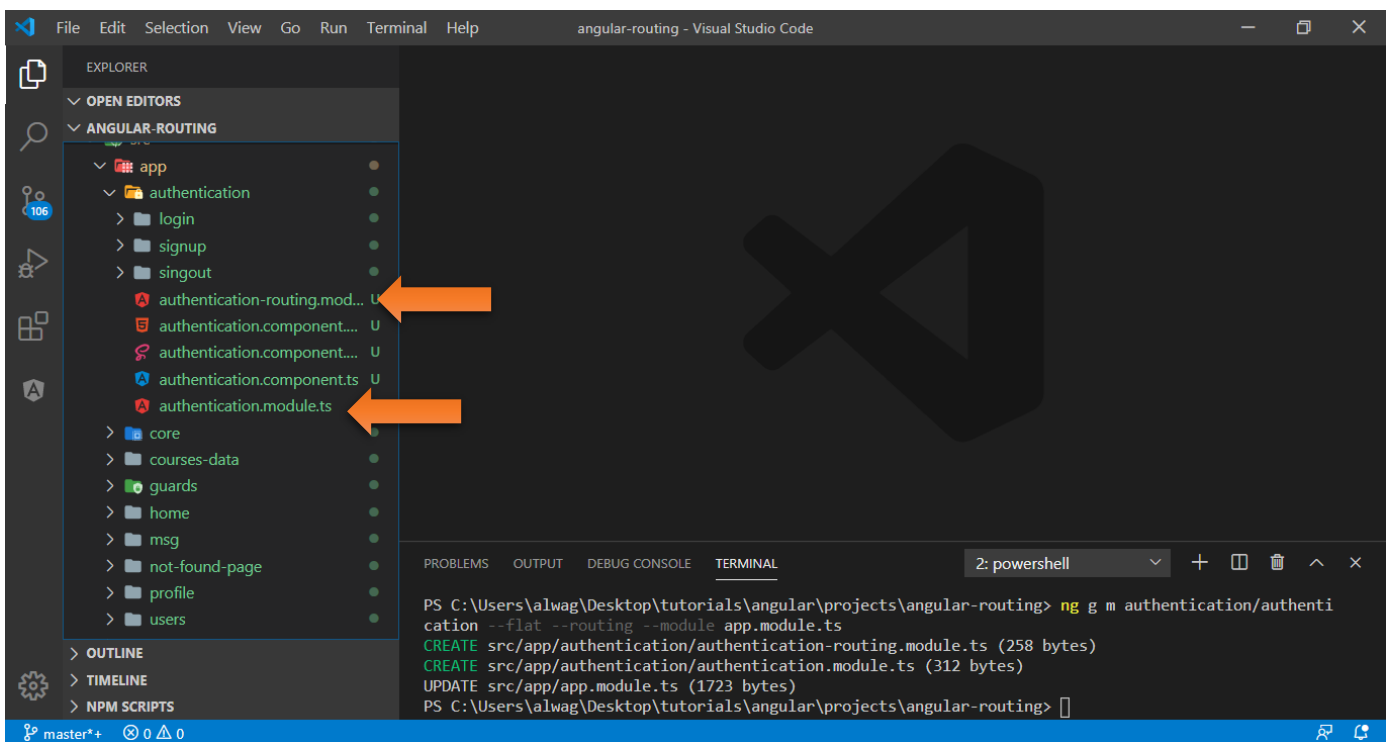


ومبروك عزيزي المتعلم أنشأت ثاني Feature Module، الآن لنتابع إلى باقي Feature Modules،

اما الآن لنتعمق أكثر وننشأ Feature Module لـ Authentication، ونبدأ كالعادة في كتابة الأمر في terminal:



ومن ثم سوف يتم انشاء الملفات التالية:



الآن لنقوم بنقل كافة التهيئة لauth إلى ملف authentication-routing.module.ts، كالتالي:

ملف authentication-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { SingoutComponent } from './singout/singout.component';
4. import { SignupComponent } from './signup/signup.component';
5. import {
    CanDeactivateGuard,
```



```

    CanDeactivateLoginGuard
  } from '../guards/can-deactivate.guard';
6. import { LoginComponent } from './login/login.component';
7. import { AuthenticationComponent } from './authentication.component';
8.
9. const routes: Routes = [
10.  {
11.    path: 'auth', children: [
12.      {
13.        path: '',
14.        component: AuthenticationComponent
15.      },
16.      {
17.        path: 'login',
18.        canDeactivate: [CanDeactivateLoginGuard],
19.        component: LoginComponent
20.      },
21.      {
22.        path: 'signup',
23.        canDeactivate: [CanDeactivateGuard],
24.        component: SignupComponent
25.      },
26.      {
27.        path: 'signout',
28.        component: SingoutComponent
29.      }
30.    ]
31.  },
32. ];
33.
34. @NgModule({
35.   imports: [RouterModule.forChild(routes)],
36.   exports: [RouterModule]
37. })
38. export class AuthenticationRoutingModule {
39.   static components = [
40.     SingoutComponent,
41.     SignupComponent,
42.     LoginComponent,
43.     AuthenticationComponent
44.   ];
45. }

```

نلاحظ اننا قمنا بنقل كافة التهيئة كما هو موجود في الاسطر من 9 إلى 32، وبما اننا هنا نريد ان ننشئ Feature Module لمجموعة من components وليس لـ component واحد، لذلك لابد ان نعمل لها declarations و exports بنفس الوقت في AuthenticationModule الذي هو اسم Feature Module لهذه components، لذلك اختصاراً للكود بما ان هذه components موجودة بالأساس في الملف authentication-routing.module.ts وبما ان الكلاس الموجود في هذا الملف

معمول له export كما هو موجود في السطر 38، لذلك قمت بإنشاء متغير من النوع static على شكل مصفوفة واضفت فيه جميع components، كما هو موجود في الاسطر من 39 إلى 44، وسوف تتوضح أكثر الفكرة عندما نستعرض الملف authentication.module.ts، كالتالي:

ملف authentication.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { FormsModule } from '@angular/forms';
4.
5. import { AuthenticationRoutingModule } from './authentication-routing.module';
6.
7. @NgModule({
8.   declarations: [
9.     AuthenticationRoutingModule.components
10.  ],
11.  imports: [
12.    CommonModule,
13.    FormsModule,
14.    AuthenticationRoutingModule
15.  ],
16.  exports: [
17.    AuthenticationRoutingModule.components
18.  ]
19.})
20. export class AuthenticationModule { }
```

نلاحظ اننا بدلاً أن نقوم باستدعاء كافة components مرة أخرى في Declarations وفي Exports واستعضنا بدلاً عنهم بالمتغير components الذي عرفناه سابقاً، كما موجود في الاسطر 9 و 17.

اما في السطر 13 فقط عملنا import لـ FormsModule لأننا استخدمنا بعض Directives الجاهزة (مثل ngModel) في هذا الـ Module في components السابقة.

وكالعادة لانسى ان نقوم بحذف جميع الاستدعاءات وترتيب استدعاء AuthenticationModule، كالتالي:

ملف app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { FormsModule } from '@angular/forms';
4.
5. import { AppComponent } from './app.component';
6. import { HomeComponent } from './home/home.component';
7. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
8. import { UsersComponent } from './users/users.component';
9. import { UserDetailsComponent } from './users/user-details/user-details.component';
10. import { UsersListComponent } from './users/users-list/users-list.component';
11. import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
```

```

12.
13.import { CoreModule } from './core/core.module';
14.import { ProfileModule } from './profile/profile.module';
15.import { MsgModule } from './msg/msg.module';
16.import { AuthenticationModule } from './authentication/authentication.module';
17.
18.@NgModule({
19.  declarations: [
20.    AppComponent,
21.    HomeComponent,
22.    NotFoundPageComponent,
23.    UsersComponent,
24.    UserDetailsComponent,
25.    UsersListComponent,
26.    EditCoursesComponent,
27.  ],
28.  imports: [
29.    BrowserModule,
30.    FormsModule,
31.    ProfileModule,
32.    MsgModule,
33.    AuthenticationModule,
34.    CoreModule,
35.  ],
36.  providers: [],
37.  bootstrap: [AppComponent]
38.})
39.export class AppModule { }

```

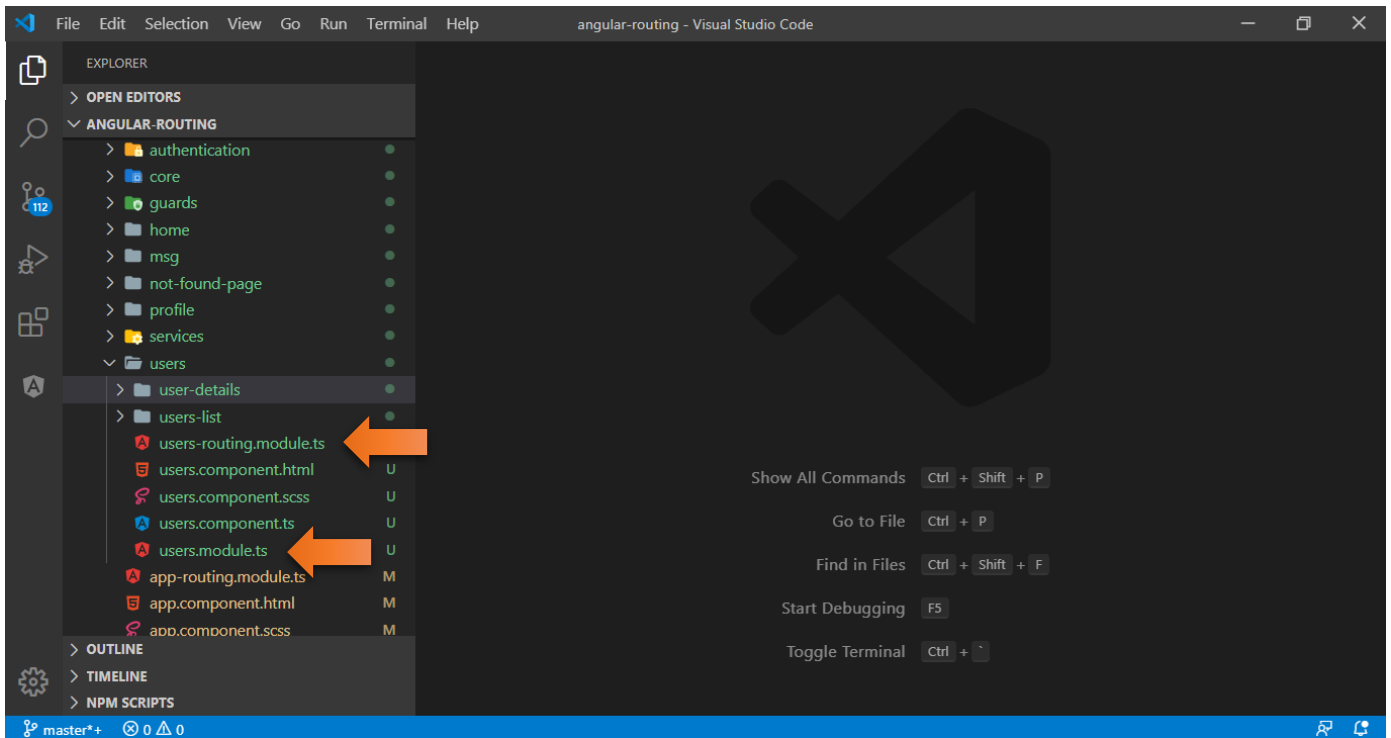
وايضاً نبارك لك عزيزي المتعلم فقد أنشأت Feature Module الثالث لك، وسوف ننتقل إلى آخر جزء والذي فيه سوف نتعلم كيف ننشأ Feature Modules متداخلة.

ولو استعرضنا الـ components الخاصة بالتعامل مع الـ users لوجدنا انه في بداية الانتقال إلى route ذو الاسم users سوف يتم عرض UsersComponent و UsersListComponent اما UserDetailsComponent فيتم الانتقال له إذا تم توجيه الرابط URL إلى Route ذو الاسم user-details/:id ومن هذا المنطلق نستطيع انشاء Feature Module لتعامل مع UsersComponent و UsersListComponent ويتفرع منه ايضاً Feature Module فرعي آخر لتعامل مع UserDetailsComponent، وهذه الطريقة تسمى Nested Feature Module.

ونستطيع عمل ذلك في البداية نقوم بإنشاء Feature Module عادي كما فعلنا سابقاً، من خلال كتابة الأمر التالي:

```
ng g m users/users --flat --routing --module app.module.ts
```

وهذا سوف يُنتج لنا الملفين التاليين، كالتالي:



وكما جرت العادة لنقوم بنقل جميع التهيئة الخاصة بـ users من ملف app-routing.module.ts إلى ملف users-routing.module.ts، كالتالي:

ملف users-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { UsersListComponent } from './users-list/users-list.component';
4. import { UsersComponent } from './users.component';
5. import { CanActivateGuard, CanActivateAdminGuard } from '../guards/can-activate.guard';
6. import { ResolveUserDetailsService } from '../guards/resolve-user-details.service';
7. import { UserDetailsComponent } from './user-details/user-details.component';
8.
9. const routes: Routes = [
10.  {
11.    path: 'users',
12.    canActivate: [CanActivateGuard, CanActivateAdminGuard],
13.    data: { roles: 'admin' },
14.    children: [
15.      {
16.        path: '',
17.        component: UsersComponent
18.      },
19.      {
20.        path: 'user-list',
21.        component: UsersListComponent
22.      },
23.      {
24.        path: 'user-details/:id',
25.        resolve: {
26.          userDetails: ResolveUserDetailsService
```

```

27.     },
28.     component: UserDetailsComponent
29.   },
30. ]
31. },
32. ];
33.
34. @NgModule({
35.   imports: [RouterModule.forChild(routes)],
36.   exports: [RouterModule]
37. })
38.
39. export class UsersRoutingModule {
40.   static components = [
41.     UsersComponent,
42.     UsersListComponent,
43.     UserDetailsComponent
44.   ];
45. }

```

وايضاً كما كنا نفعل لنذهب إلى ملف `users.module.ts` ونضيف الكود التالية:

#### ملف `users.module.ts`

```

1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { FormsModule } from '@angular/forms';
4.
5. import { UsersRoutingModule } from './users-routing.module';
6.
7. @NgModule({
8.   declarations: [UsersRoutingModule.components],
9.   imports: [
10.    CommonModule,
11.    FormsModule,
12.    UsersRoutingModule,
13.   ]
14. })
15.
16. export class UsersModule { }

```

لا ننسى ان نضيف `FormsModule` كما في سطر 11 لأننا تعاملنا مع بعض الدايكريتييف الموجودة في هذا Module في هذه الـ `components`.

وأخيراً نذهب إلى ملف `app.module.ts` ونحذف جميع الاستدعاءات الخاصة بهذه الـ `components` ونجعل الاستدعاء لـ `UsersModule` قبل `CoreModule`، كالتالي:

#### ملف `app.module.ts`

```

1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3.

```

```

4. import { AppComponent } from './app.component';
5. import { HomeComponent } from './home/home.component';
6. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
7. import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
8.
9. import { CoreModule } from './core/core.module';
10. import { ProfileModule } from './profile/profile.module';
11. import { MsgModule } from './msg/msg.module';
12. import { AuthenticationModule } from './authentication/authentication.module';
13. import { UsersModule } from './users/users.module';
14.
15. @NgModule({
16.   declarations: [
17.     AppComponent,
18.     HomeComponent,
19.     NotFoundPageComponent,
20.     EditCoursesComponent,
21.   ],
22.
23.   imports: [
24.     BrowserModule,
25.     ProfileModule,
26.     MsgModule,
27.     AuthenticationModule,
28.     UsersModule,
29.     CoreModule,
30.   ],
31.
32.   providers: [],
33.   bootstrap: [AppComponent]
34. })
35.
36. export class AppModule { }
37.

```

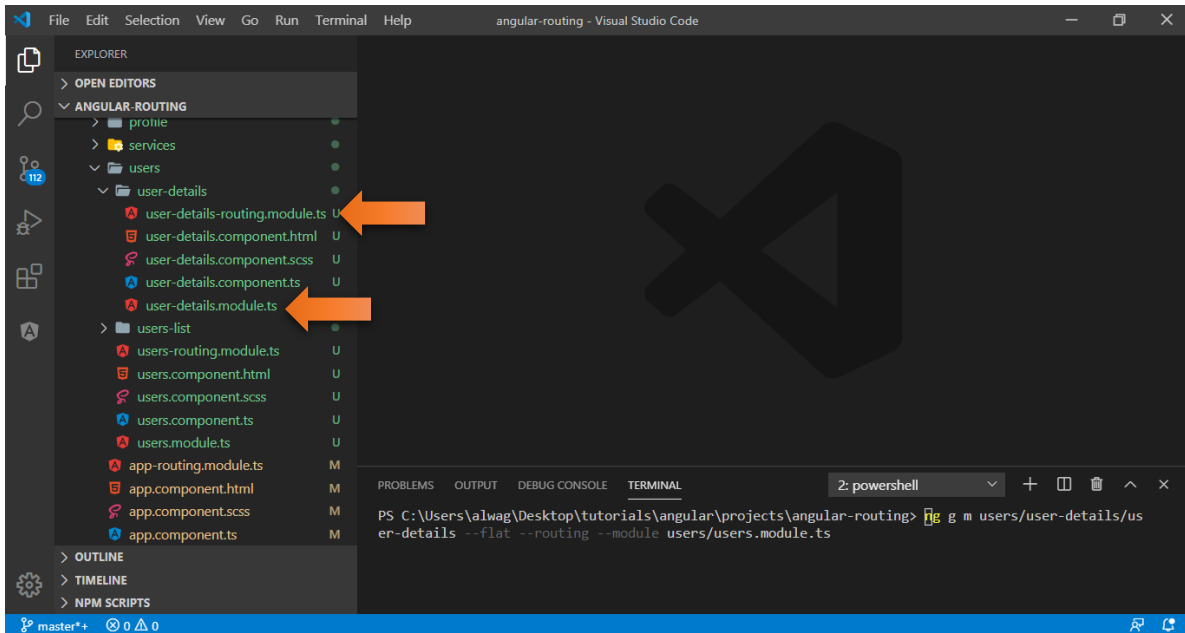
رجع الاسطر 28 و29.

جميع الذي قمنا بعمله سابقاً هو في الحقيقة تكرار لما سبق ولكن الإضافة هنا هي ان نقوم بإنشاء Feature Module فرعي لـ UsersModule الذي قمنا بإنشائه قبل قليل، لكي يتعامل مع DetailsComponent، وحقيقة طريقة انشائه هي مشابهه لما سبق ولكن هنا نجعل الـ app له هو UsersModule ليس AppModule كما كنا نفعل في السابق، ونبدأ أولاً بكتابة الامر التالي في terminal، كالتالي:

**ng g m users/user-details/user-details --flat --routing --module users/users.module.ts**

كما هو واضح من الامر السابق انشأنا Module داخل المجلد user-details وايضاً اسم الملف الذي يحتوي على هذا Module له نفس الاسم user-details، واخيراً طلبنا من ان يقوم باستدعاء هذا Module في ملف users.module.ts لكي يصبح كانه الـ app له ويتفرع منه.

وبعد تنفيذ الامر السابق سوف يُنتج لنا الملفين التاليين:



اما خطوات التهيئة فهي مشابهه لما سبق ولكن هذه المرة نقوم بنقل التهيئة الخاصة بـ route ذو الاسم "user-details/:id" من ملف users-routing.module.ts إلى ملف user-details-routing.module.ts، كالتالي:

ملف user-details-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { ResolveUserDetailsService } from 'src/app/guards/resolve-user-
  details.service';
4. import { UserDetailsComponent } from './user-details.component';
5.
6. const routes: Routes = [
7.   {
8.     path: 'users/user-details/:id',
9.     resolve: {
10.      userDetails: ResolveUserDetailsService
11.    },
12.    component: UserDetailsComponent
13.  },
14. ];
15.
16. @NgModule({
17.   imports: [RouterModule.forChild(routes)],
18.   exports: [RouterModule]
19. })
20.
21. export class UserDetailsRoutingModule {
22.   static components = [UserDetailsComponent];
23. }
24.
```

نلاحظ هنا انناى قمنا بنقل التهيئة لهذا Route والاضافة الوحيدة هنا اننا قمنا بكتابة path الخاص بالroute الاب (users) قبل path الخاص بهذا route، وان لم نعمل هذا الامر فلن يعمل كما هو موجود في السطر 8.

اما في user-details.module.ts، فنضيف تعريف components في مصفوفة declarations، كالتالي:

ملف user-details.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { UserDetailsRoutingModule } from './user-details-routing.module';
4.
5. @NgModule({
6.   declarations: [UserDetailsRoutingModule.components],
7.   imports: [
8.     CommonModule,
9.     UserDetailsRoutingModule
10.  ]
11.})
12. export class UserDetailsModule { }
```



راجع السطر 6.

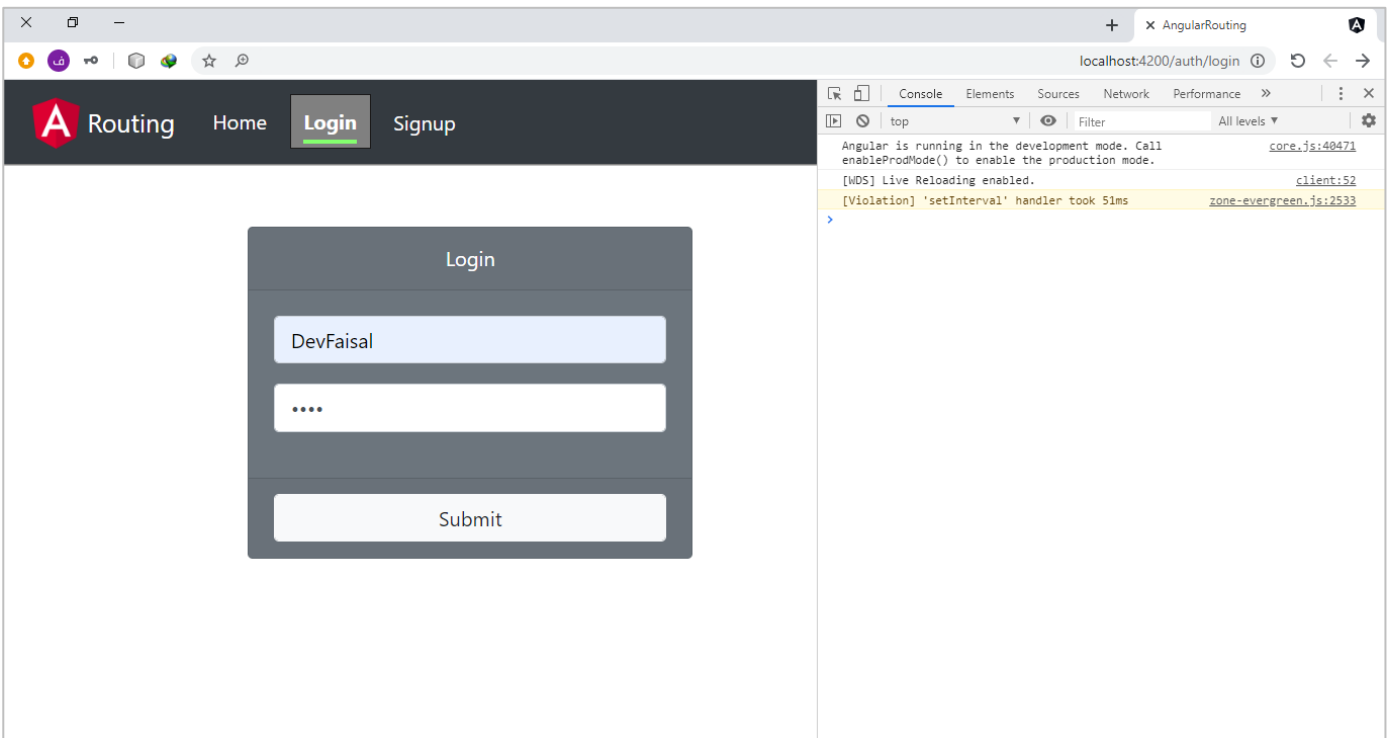
وأخيراً نتأكد ان هذا Module تم استدعائه في UsersModule، الاب، كالتالي:

ملف users.module.ts

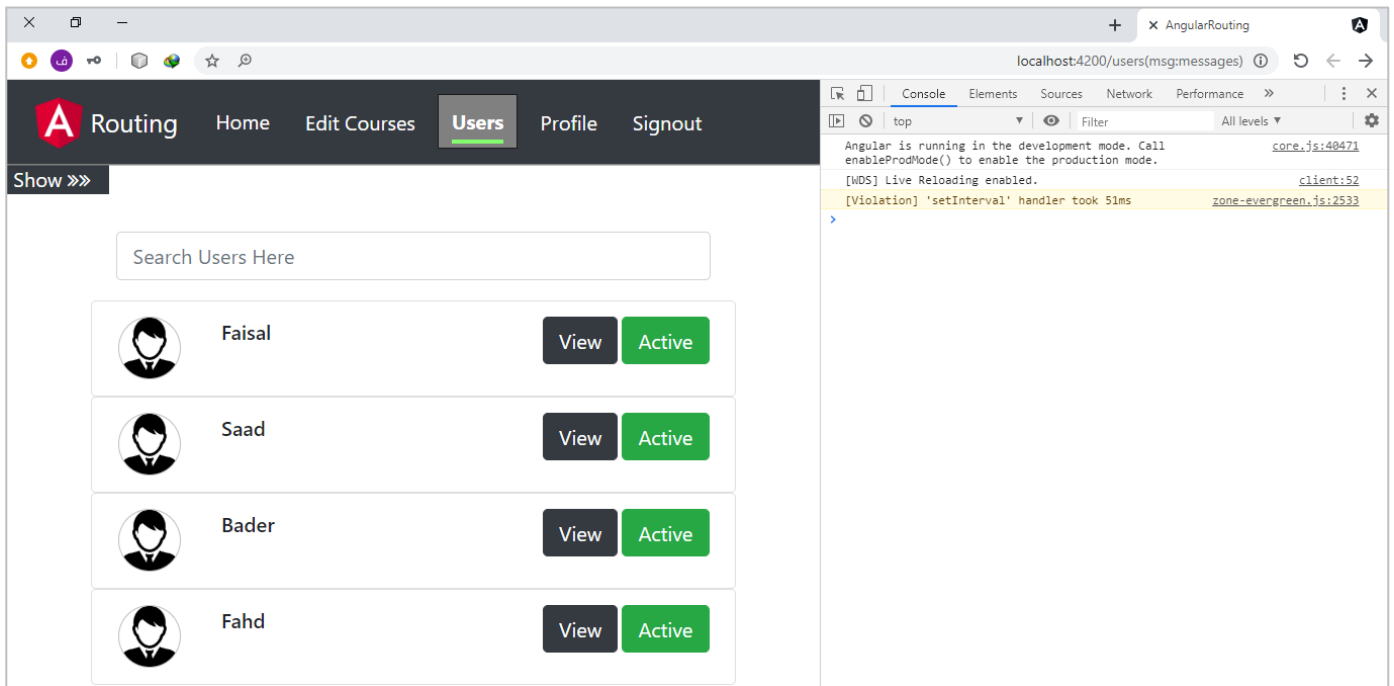
```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { FormsModule } from '@angular/forms';
4.
5. import { UsersRoutingModule } from './users-routing.module';
6. import { UserDetailsModule } from './user-details/user-details.module';
7.
8. @NgModule({
9.   declarations: [UsersRoutingModule.components],
10.  imports: [
11.    CommonModule,
12.    FormsModule,
13.    UsersRoutingModule,
14.    UserDetailsModule,
15.  ]
16.})
17.
18. export class UsersModule { }
```

راجع الاسطر 6 و14.

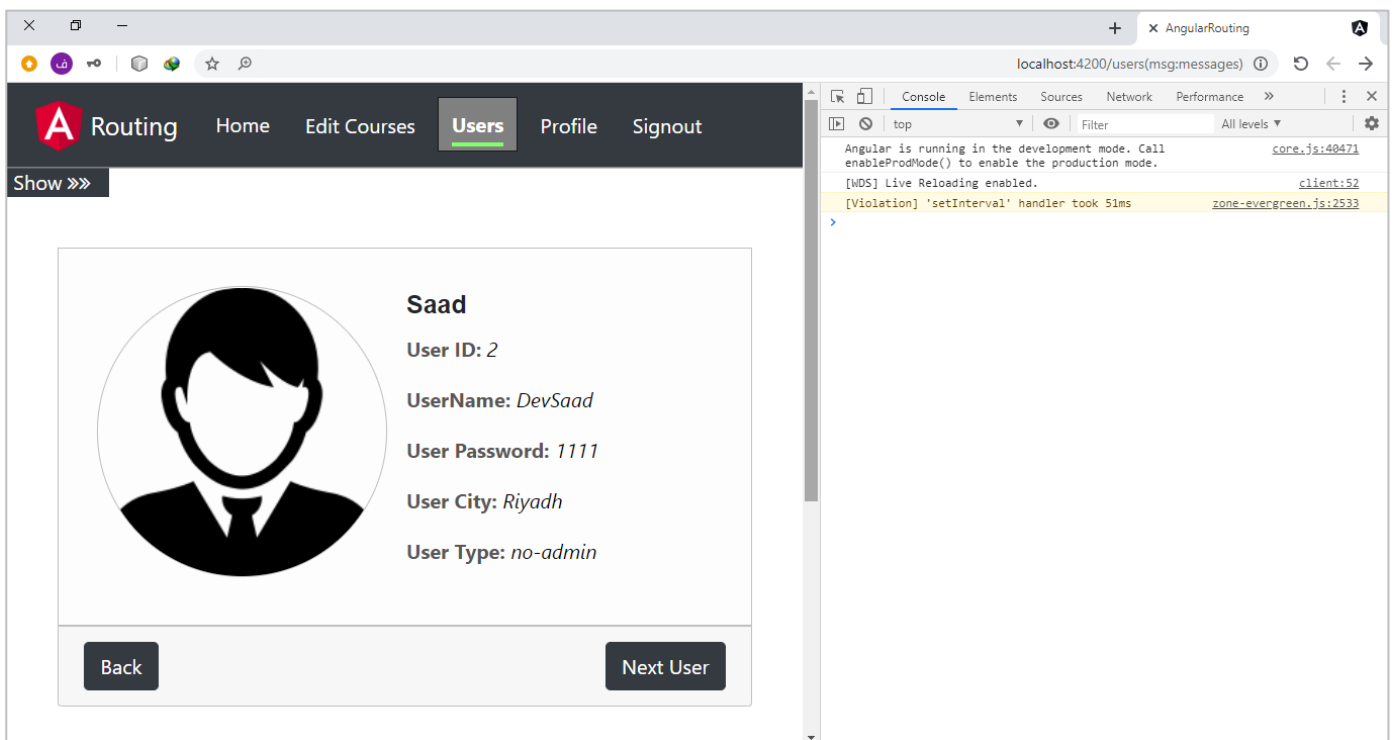
الآن لنقم بحفظ التعديلات ونذهب إلى المتصفح لنرى النتيجة، كالتالي:



لنقم بتسجيل الدخول بحساب الأدمن admin، ومن ثم نختار صفحة Users، كالتالي:



نلاحظ انه يعمل بدون أي مشاكل، الآن لنختار أحد المستخدمين، كالتالي:



نلاحظ انه أيضاً يعمل بدون أي مشاكل.

وبذلك نكون انتهينا من Featur Modules وأصبحت لديك المعرفة عزيزي المتعلم بكيفية تقسيم Routing و Root Module إلى مجموعة من الأجزاء الصغيرة المترابطة مع بعضها ومنفصلة بنفس الوقت، وهناك تحدي أريده منك وهو القيام بتقسيم Authentication إلى Nested Feature Module مع العلم اننا قمنا بإنشاء Feature Module له ولكن ما أريده منك هو ان تقوم بإنشاء Feature Modules فرعية لكلاً من login و signup.

اما الآن فسوف ننتقل إلى آخر نوع هو Shared Module.

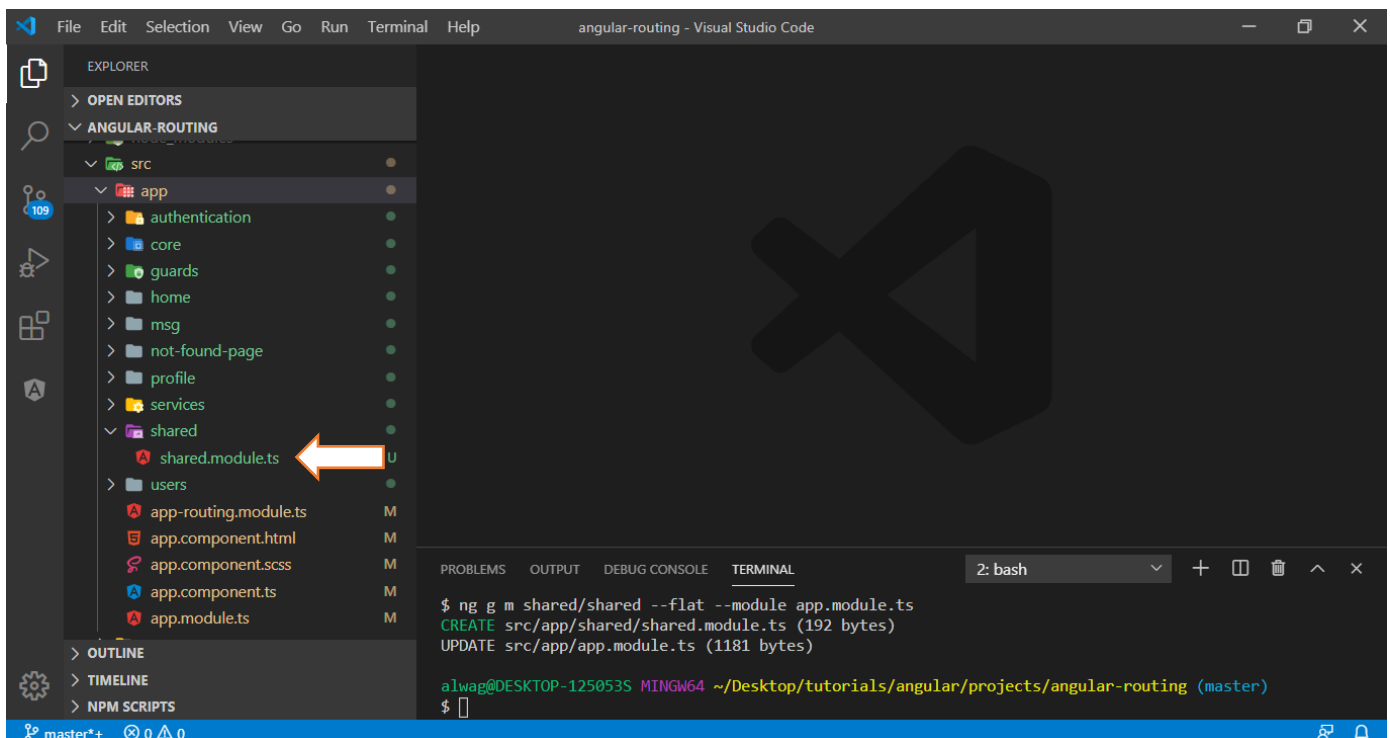
## 5.4 .Shared Module

وتكمن أهمية هذا النوع في التقليل من تكرار الاستدعاءات في كل من Feature Modules و Routing Modules بحيث نضع components و directives و pipe وحتى Modules (المقصود بها المبنية ضمناً في نفس angular مثل FormsModule و RoutingModule وغيرها الكثير) فجميع هذه الأجزاء ان تم استخدامها في اكثر من component او بمعنى آخر أن تم استدعاءها في اكثر من Feature Module (السبب لأن هذه الـ Modules تعتبر حاوية لـ component واحد او اكثر من component)، في هذه الحالة يتم تجميع هذه الأنواع ووضعها في Module واحد ويتم استدعاء هذا الـ Module فقط.

اما في مشروعنا هنا فلا يوجد لدينا أي Pipes او Directives او حتى Components تم عمل لها مشاركة Share في اكثر من component، ولكن لدينا بعض الـ Modules تم استخدامها في اكثر من Feature Module او Routing Module، ومن امثلة هذه الأنواع FormsModule و CommonModule، وسوف نقوم بإنشاء Shared Module لهذه الأنواع، وللقيام بهذا الامر نقوم في البداية بكتابة الامر التالي:

```
ng g m shared/shared --flat --module app.module.ts
```

وبعدها سوف يقوم بإنشاء الملف التالي:



والآن لنستعرض محتويات هذا الملف، كالتالي:

ملف shared.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3.
4. @NgModule({
5.   declarations: [],
6.   imports: [CommonModule]
```

```
7. })
8. export class SharedModule { }
```

وكما هو واضح نلاحظ انه ملف Module عادي (اعيد مرة أخرى ان جميع أنواع Modules هي في الحقيقة مشابهه لبعض والفرق الوحيد في كيفية استخدامها)، وبما ان CommonModule تم استدعائه افتراضياً عند انشاء هذا الملف، سنقوم الآن باستدعاء FormsModule، ووضع كلا هذين Modules في مصفوفة exports، كالتالي:

ملف shared.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { CommonModule } from '@angular/common';
3. import { FormsModule } from '@angular/forms';
4.
5. @NgModule({
6.   declarations: [],
7.   imports: [],
8.   exports: [
9.     CommonModule,
10.    FormsModule,
11.
12.  ]
13.})
14. export class SharedModule { }
```

نلاحظ اننا قمنا بنقل CommonModule من مصفوفة imports ووضعناها في مصفوفة exports، والسبب بكل بساطة انه لا يوجد لدينا هنا أي component او directive او pipe يعتمدون على هذه Modules وانما كل الذي نريده هو مشاركة هذه Modules مع Feature Modules الأخرى، ولكن في حال كان لدينا على سبيل المثال component وهذا component نريد ان نعمل له مشاركة فنضعه في مصفوفة declarations وفي مصفوفة exports، وفي حال كان هذا component استخدم بعض الكلاسات من FormsModule فلا بد ان نضع هذا Module في مصفوفة imports وبنفس الوقت في حال كان هذا Module لم يتم استخدامه في أي component في أي Feature Module أخرى فلا نضعها في مصفوفة exports، لأن الـ Feature Modules وأي Modules أخرى لا ترى في هذا Module إلا الذي نعمل له export فقط.

الآن لنبحث عن أي استدعاء لـ FormsModule وCommonModule ونستبدلها بـ SharedModule، كالتالي:

ملف authentication.module.ts

```
1. import { NgModule } from '@angular/core';
2.
3. import { AuthenticationRoutingModule } from './authentication-routing.module';
4. import { SharedModule } from '../shared/shared.module';
5.
6.
7. @NgModule({
8.   declarations: [
9.     AuthenticationRoutingModule.components
10.  ],
11.   imports: [
```

```

12.     SharedModule,
13.     AuthenticationRoutingModule,
14. ],
15. exports: [
16.     AuthenticationRoutingModule.components
17. ]
18.})
19.export class AuthenticationModule { }

```

ملف msg.module.ts

```

1. import { NgModule } from '@angular/core';
2.
3. import { MsgRoutingModule } from './msg-routing.module';
4. import { SharedModule } from '../shared/shared.module';
5.
6. @NgModule({
7.   declarations: [MsgRoutingModule.components],
8.   imports: [
9.     SharedModule,
10.    MsgRoutingModule
11.  ],
12.  exports: [MsgRoutingModule.components]
13.})
14.export class MsgModule { }

```

ملف users.module.ts

```

1. import { NgModule } from '@angular/core';
2.
3. import { UsersRoutingModule } from './users-routing.module';
4. import { UserDetailsModule } from './user-details/user-details.module';
5. import { SharedModule } from '../shared/shared.module';
6.
7. @NgModule({
8.   declarations: [UsersRoutingModule.components],
9.   imports: [
10.     SharedModule,
11.     UsersRoutingModule,
12.     UserDetailsModule,
13.   ]
14.})
15.
16.export class UsersModule { }

```

ولا ننسى ان نحذف أي استدعاء لهذهين Modules في ملف user-details.module.ts.

# الفصل الخامس

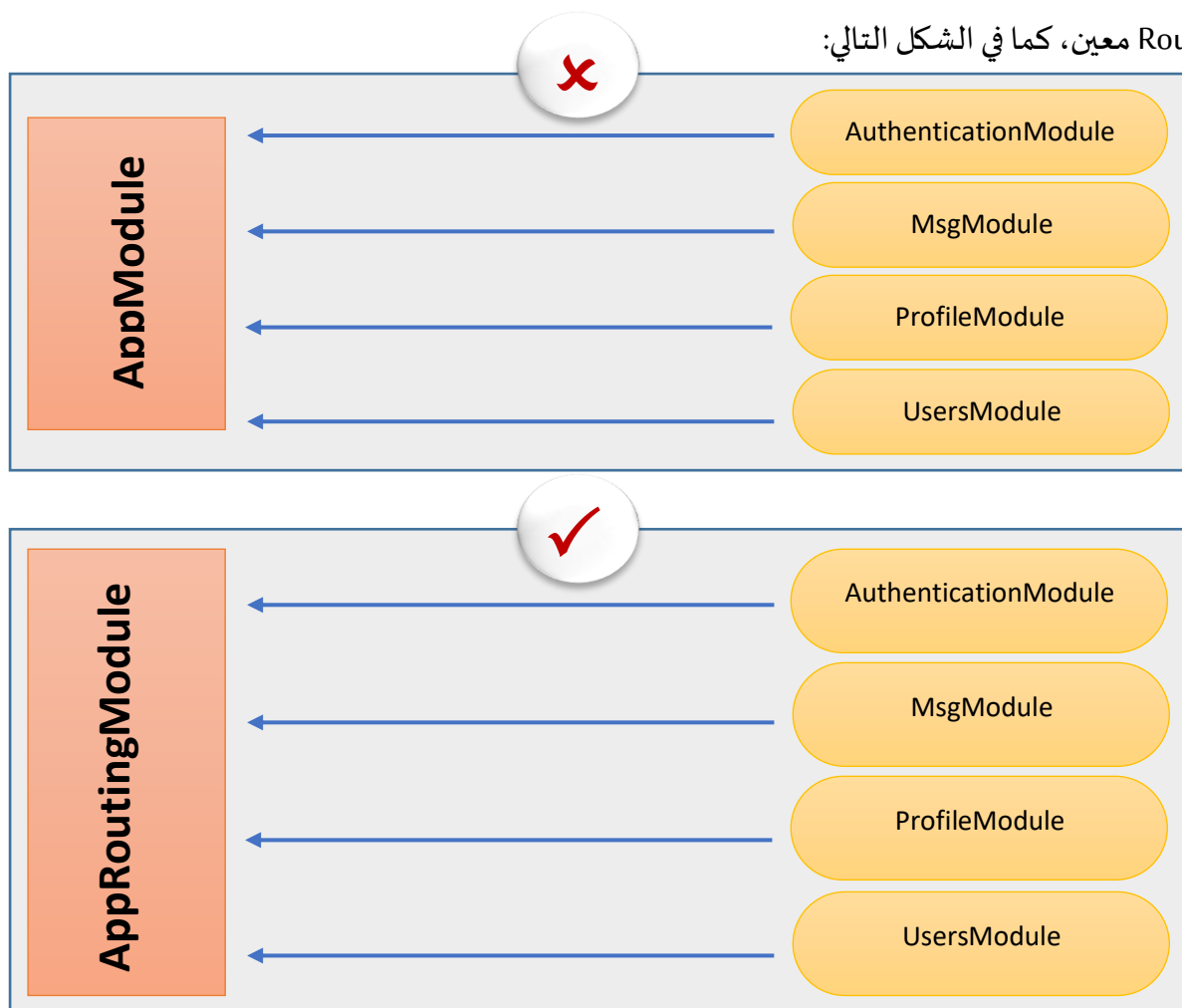
## Lazy Loading

## 1.5. مقدمة:

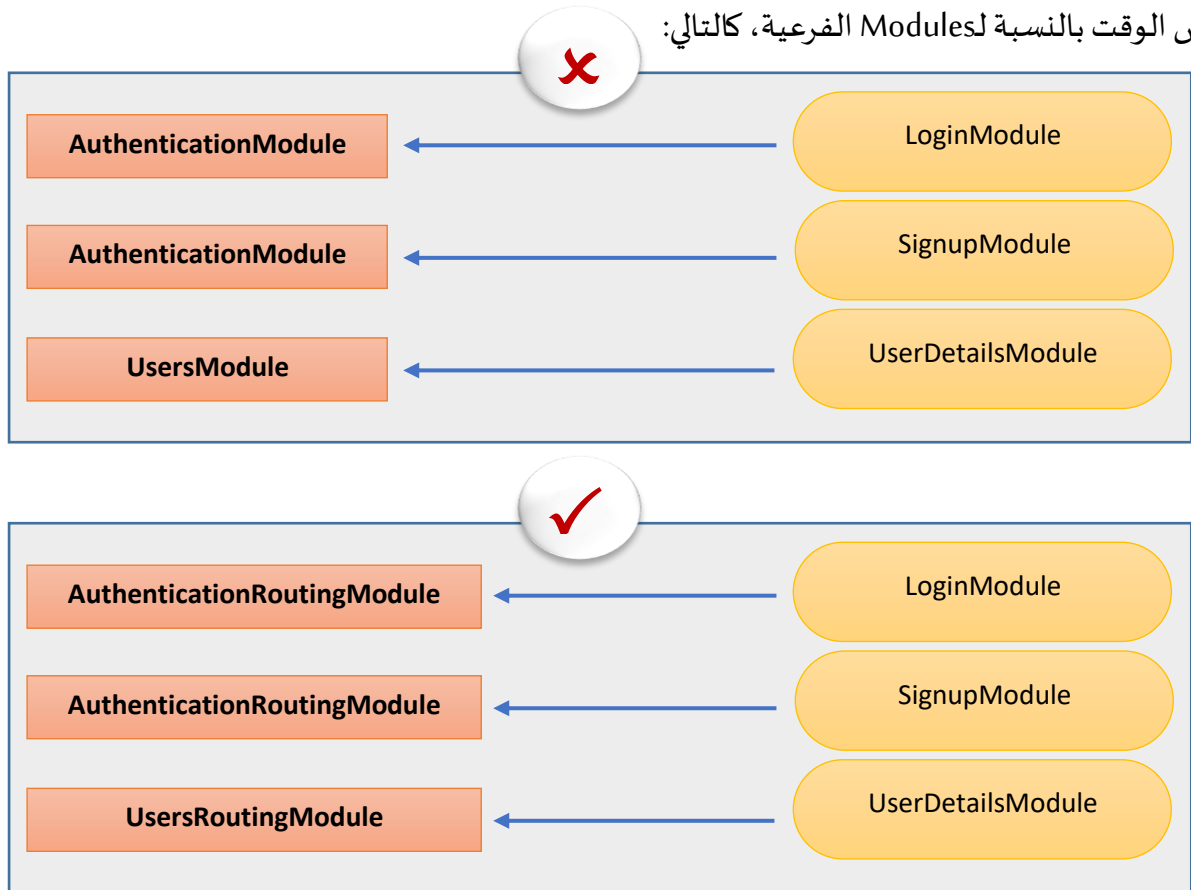
angular عند بداية تشغيل أي مشروع يقوم في البداية بقراءة ملف `app.module.ts` ومن ثم يقوم بتنفيذ جميع ما يحتويه وعند الانتهاء يقوم بعرض التطبيق للمستخدم وهذه الطريقة تسمى `Eager Loading` وهي الطريقة الافتراضية لأي مشروع angular وهي طريقة جيدة للمشاريع الصغيرة ولكن في المشاريع المتوسطة إلى الكبيرة قد تكون تجربة غير جيدة للمستخدم حيث قد ينتظر فترة من الوقت لكي يعمل له التطبيق، لذلك قدمت لنا angular طريقة أخرى تسمى `Lazy Loading` وتقوم فكرتها أننا بدلاً من أن نقوم بتحميل جميع المشروع ككتلة واحدة لنقوم بتحميل وعرض الأجزاء الرئيسية من التطبيق للمستخدم، وكلما قام المستخدم بالتوجيه إلى route معين نقوم بتحميل وعرض ما يحتويه هذا route، وبذلك قللنا من زمن انتظار المستخدم عند بداية تشغيل التطبيق ورفعنا من أداء التطبيق بنفس الوقت.

ويعمدا استعرضنا مفاهيم هذه الميزة ننتقل الآن إلى الجزء العملي، وهو في الحقيقة يعتمد اعتماد كامل على `Feature Modules`، بحيث إذا فهمت عزيزي المتعلم مفاهيم `Feature Modules` وكيفية بناءها فسوف يسهل عليك بناء `Lazy Loading`.

بحيث تكمن الفكرة الأساسية بدلاً من استدعاء `Feature Modules` في ملف `app.module.ts`، نقل الاستدعاء إلى ملف `app-routing.module.ts` ونجعل هذا الاستدعاء ديناميكي بحيث لا يتم تحميل هذا `Feature Module` إلا إذا تم توجيهه إلى Route معين، كما في الشكل التالي:



وبنفس الوقت بالنسبة لـ Modules الفرعية، كالتالي:



## 2.5. خطوات تطبيق Lazy Loding على المشروع:

ولذلك لنقوم بتطبيق هذه الخطوات عملياً، في البداية لنقم بإلغاء جميع الاستدعاءات من ملف `app.module.ts`، كالتالي:

ملف `app.module.ts`

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3.
4. import { AppComponent } from './app.component';
5. import { HomeComponent } from './home/home.component';
6. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
7. import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
8.
9. import { CoreModule } from './core/core.module';
10. import { ProfileModule } from './profile/profile.module';
11. import { MsgModule } from './msg/msg.module';
12. import { AuthenticationModule } from './authentication/authentication.module';
13. import { UsersModule } from './users/users.module';
14. import { SharedModule } from './shared/shared.module';
15.
16. @NgModule({
17.   declarations: [
18.     AppComponent,
19.     HomeComponent,
20.     NotFoundPageComponent,
```



```

21.   EditCoursesComponent,
22. ],
23. imports: [
24.   BrowserModule,
25.   ProfileModule,
26.   MsgModule,
27.   AuthenticationModule,
28.   UsersModule,
29.   CoreModule,
30.   SharedModule,
31. ],
32. providers: [],
33. bootstrap: [AppComponent]
34.})
35.export class AppModule { }

```

راجع الاسطر من 10 إلى 13 والاسطر من 25 إلى 28.

ليصبح بالشكل التالي:

ملف app.module.ts

```

1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3.
4. import { AppComponent } from './app.component';
5. import { HomeComponent } from './home/home.component';
6. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
7. import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
8.
9. import { CoreModule } from './core/core.module';
10. import { SharedModule } from './shared/shared.module';
11.
12. @NgModule({
13.   declarations: [
14.     AppComponent,
15.     HomeComponent,
16.     NotFoundPageComponent,
17.     EditCoursesComponent,
18.   ],
19.   imports: [
20.     BrowserModule,
21.     CoreModule,
22.     SharedModule,
23.   ],
24.   providers: [],
25.   bootstrap: [AppComponent]
26. })
27. export class AppModule { }

```

ليس هنا فقط وانما في أي جزء بالمشروع تم استدعاء هذا الModule لابد ان نحذف هذا الاستدعاء.

الآن كما قلنا سابقاً لنقوم بنقل كافة هذه الاستدعاءات إلى ملف التهيئة الرئيسي، app-routing.module.ts، وسوف نبدأ في البداية بـ MsgModule، كالتالي:

#### ملف app-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { HomeComponent } from '../home/home.component';
4. import { NotFoundPageComponent } from '../not-found-page/not-found-page.component';
5. import { EditCoursesComponent } from '../home/edite-courses/edit-courses.component';
6. import { CanActivateAdminChildGuard, CanActivateChildGuard } from '../guards/can-activate-child.guard';
7. import { ResolveCoursesService } from '../guards/resolve-courses.service';
8.
9. const routes: Routes = [
10.   {
11.     path: 'home',
12.     component: HomeComponent,
13.     resolve: { courses: ResolveCoursesService },
14.     canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
15.     children: [
16.       { path: 'edit-courses', component: EditCoursesComponent }
17.     ]
18.   },
19.   {
20.     path: 'messages', outlet: 'msg',
21.     loadChildren: () => import('../msg/msg.module').then(module => module.MsgModule)
22.   },
23.   { path: '', redirectTo: 'home', pathMatch: 'full' },
24.   { path: '**', component: NotFoundPageComponent }
25. ];
26.
27. @NgModule({
28.   imports: [RouterModule.forRoot(routes)],
29.   exports: [RouterModule]
30. })
31. export class AppRoutingModule { }
```

راجع الاسطر من 20 إلى 21.

اولاً قمنا بأعطائه path، وكأنا نقول له إذا تم توجيه الرابط URL إلى هذا path قم بتحميل Module ذو الاسم MsgModule، ونستطيع الوصول إلى هذا Module عن طريق الخاصية loadChildren حيث اسندنا لها قيمة وهذه القيمة هي عبارة عن دالة من النوع Arrow Function اما محتوى هذه الدالة فستخدنا الدالة import ومررنا لها المسار الموجود عليه الـ Module وهي تُعيد Promise لذلك استخدمنا then حيث قمت بتمرير بارامتر اسميته module ولك حرية اختيار الاسم الذي تُريده وهذا البارامتر يحتوي على قيمة وهذه القيمة هي عبارة عن Feature Module الذي تُريد عرضه والذي هو في حالتنا هذه هو MsgModule.

باقي آخر خطوة وهي الذهاب إلى ملف التهيئة الفرعي لهذا path وهو ملف msg-routing.module.ts، ونقوم بإجراء التعديلات التالية:

ملف msg-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { MsgComponent } from './msg.component';
4.
5. const routes: Routes = [
6.   { path: '', component: MsgComponent },
7. ];
8.
9. @NgModule({
10.  imports: [RouterModule.forChild(routes)],
11.  exports: [RouterModule]
12. })
13. export class MsgRoutingModule {
14.  static components = [MsgComponent];
15. }
```

راجع السطر 6.

حيث قمنا بحذف قيمة path لهذا route والسبب في ذلك منعاً لتكرار في الرابط لأننا لو لم نحذف هذه القيمة وتم التوجيه لهذا الرابط فسوف يكون كالتالي: [http://localhost:4200/home\(msg:messages/messages\)](http://localhost:4200/home(msg:messages/messages)) وبذلك يكون الرابط خطأ، لذلك لتجنب هذا الخطأ نقوم بتفريغه هنا. الآن بنفس هذه الكيفية لنقوم بإعادة ضبط جميع الأنواع الأخرى.

ملف app-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { HomeComponent } from './home/home.component';
4. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
5. import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
6. import { CanActivateAdminChildGuard, CanActivateChildGuard } from './guards/can-activate-child.guard';
7. import { ResolveCoursesService } from './guards/resolve-courses.service';
8.
9. const routes: Routes = [
10.  {
11.    path: 'home',
12.    component: HomeComponent,
13.    resolve: { courses: ResolveCoursesService },
14.    canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
15.    children: [
16.      { path: 'edit-courses', component: EditCoursesComponent }
17.    ]
18.  },
19.  {
```

```

20.   path: 'messages',
21.   loadChildren: () => import('./msg/msg.module').then(module => module.MsgModule)
22. },
23. {
24.   path: 'profile/:id',
25.   loadChildren: () =>
26.     import('./profile/profile.module').then(module => module.ProfileModule)
27. },
28. {
29.   path: 'auth',
30.   loadChildren: () =>
31.     import('./authentication/authentication.module')
32.     .then(module => module.AuthenticationModule)
33. },
34.   path: 'users',
35.   loadChildren: () =>
36.     import('./users/users.module').then(module => module.UsersModule)
37. },
38. { path: '', redirectTo: 'home', pathMatch: 'full' },
39. { path: '**', component: NotFoundPageComponent }
40.];
41.
42.@NgModule({
43.  imports: [RouterModule.forRoot(routes)],
44.  exports: [RouterModule]
45.})
46.export class AppRoutingModule { }
47.

```

راجع الاسطر من 20 إلى 37.

الان لنذهب إلى كل Feature Module ونقوم بعمل التعديلات عليها كما فعلنا في msg-routing.module.ts، كالتالي:

ملف profile-routing.module.ts

```

1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { CanActivateGuard } from '../guards/can-activate.guard';
4. import { ProfileComponent } from './profile.component';
5. const routes: Routes = [
6.   {
7.     path: '',
8.     canActivate: [CanActivateGuard],
9.     component: ProfileComponent
10.  }
11. ];
12.@NgModule({
13.  imports: [RouterModule.forChild(routes)],
14.  exports: [RouterModule]
15.})
16.export class ProfileRoutingModule { }

```

```

1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { UsersListComponent } from './users-list/users-list.component';
4. import { UsersComponent } from './users.component';
5. import { CanActivateGuard, CanActivateAdminGuard } from '../guards/can-
  activate.guard';
6.
7. const routes: Routes = [
8.   {
9.     path: '',
10.    canActivate: [CanActivateGuard, CanActivateAdminGuard],
11.    data: { roles: 'admin' },
12.    children: [
13.      {
14.        path: '',
15.        component: UsersComponent
16.      },
17.      {
18.        path: 'user-list',
19.        component: UsersListComponent
20.      },
21.    ]
22.  },
23.];
24.
25. @NgModule({
26.   imports: [RouterModule.forChild(routes)],
27.   exports: [RouterModule]
28. })
29. export class UsersRoutingModule {
30.   static components = [
31.     UsersComponent,
32.     UsersListComponent,
33.   ];
34. }
35.

```

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { SingoutComponent } from './singout/singout.component';
import { SignupComponent } from './signup/signup.component';
import { LoginComponent } from './login/login.component';
import { AuthenticationComponent } from './authentication.component';

const routes: Routes = [
  {
    path: '',
    children: [

```

```

    {
      path: '',
      component: AuthenticationComponent
    },
  ],
},
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class AuthenticationRoutingModule {
  static components = [
    SingoutComponent,
    SignupComponent,
    LoginComponent,
    AuthenticationComponent
  ];
}

```

الآن لنقم بالـ Feature Modules الفرعية كما فعلنا في Feature Modules الرئيسية، كالتالي:

ملف users.module.ts

```

1. import { NgModule } from '@angular/core';
2.
3. import { UsersRoutingModule } from './users-routing.module';
4. import { UserDetailsModule } from './user-details/user-details.module';
5. import { SharedModule } from '../shared/shared.module';
6.
7. @NgModule({
8.   declarations: [UsersRoutingModule.components],
9.   imports: [
10.    SharedModule,
11.    UsersRoutingModule,
12.    UserDetailsModule,
13.  ]
14.})
15.
16. export class UsersModule { }

```

نحذف استدعاء UserDetailsModule  
من هذا الملف

ونعيد استدعاء UserDetailsModule من ملف users-routing.module.ts، كالتالي:

ملف users-routing.module.ts

```

1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { UsersListComponent } from './users-list/users-list.component';
4. import { UsersComponent } from './users.component';

```

```

5. import { CanActivateGuard, CanActivateAdminGuard } from '../guards/can-
   activate.guard';
6.
7. const routes: Routes = [
8.   {
9.     path: '',
10.    canActivate: [CanActivateGuard, CanActivateAdminGuard],
11.    data: { roles: 'admin' },
12.    children: [
13.      {
14.        path: '',
15.        component: UsersComponent
16.      },
17.      {
18.        path: 'user-list',
19.        component: UsersListComponent
20.      },
21.      {
22.        path: 'user-details/:id',
23.        loadChildren: () =>
24.          import('../user-details/user-details.module').then(m => m.UserDetailsModule)
25.      }
26.    ]
27.  },
28.];
29.
30. @NgModule({
31.   imports: [RouterModule.forChild(routes)],
32.   exports: [RouterModule]
33. })
34. export class UsersRoutingModule {
35.   static components = [
36.     UsersComponent,
37.     UsersListComponent
38.   ];
39. }

```

راجع الاسطر من 22 إلى 24.

وأخيراً نستعرض ملف `user-details-routing.module.ts` ونحذف منه القيمة الموجودة في `path` الخاصة بتهيئة هذا `route`، كالتالي:

ملف `user-details-routing.module.ts`

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { ResolveUserDetailsService } from 'src/app/guards/resolve-user-details.service';
import { UserDetailsComponent } from '../user-details.component';

const routes: Routes = [
  {
    path: '',

```

```

    resolve: {
      userDetails: ResolveUserDetailsService
    },
    component: UserDetailsComponent
  },
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})

export class UserDetailsRoutingModule {
  static components = [UserDetailsComponent];
}

```

ولنفعل نفس ما فعلناه ولكن هذه المرة لـ LoginModule و SignupModule الفرعيتين لـ AuthenticationModule، كالتالي:

ملف authentication.module.ts

```

1. import { NgModule } from '@angular/core';
2.
3. import { AuthenticationRoutingModule } from './authentication-routing.module';
4. import { SharedModule } from '../shared/shared.module';
5.
6. @NgModule({
7.   declarations: [
8.     AuthenticationRoutingModule.components
9.   ],
10.  imports: [
11.    SharedModule,
12.    AuthenticationRoutingModule,
13.  ]
14.})
15. export class AuthenticationModule { }

```

نحذف الاستدعاءات لكلاً من  
SignupModule و LoginModule  
من هذا الملف

وبنفس الوقت نعيد استدعاء هذه Modules بطريقة ديناميكية في ملف authentication-routing.module.ts، كالتالي:

ملف authentication-routing.module.ts

```

1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { SingoutComponent } from './singout/singout.component';
4. import { SignupComponent } from './signup/signup.component';
5. import { LoginComponent } from './login/login.component';
6. import { AuthenticationComponent } from './authentication.component';
7.
8. const routes: Routes = [
9.   {
10.    path: '',
11.    children: [
12.      {
13.        path: '',
14.        component: AuthenticationComponent

```



```

15.     },
16.     {
17.         path: 'login',
18.         loadChildren: () => import('./login/login.module').then(m => m.LoginModule)
19.     },
20.     {
21.         path: 'signup',
22.         loadChildren: () => import('./signup/signup.module').then(m => m.SignupModule)
23.     },
24.     {
25.         path: 'signout',
26.         component: SingoutComponent
27.     }
28. ]
29. },
30. ];
31.
32. @NgModule({
33.     imports: [RouterModule.forChild(routes)],
34.     exports: [RouterModule]
35. })
36. export class AuthenticationRoutingModule {
37.     static components = [
38.         SingoutComponent,
39.         LoginComponent,
40.         SignupComponent,
41.         AuthenticationComponent
42.     ];
43. }

```

راجع الاسطر من 17 إلى 26.

وأخيراً نحذف قيمة الخاصية path في كلاً من الملفين login-routing.module.ts والملف signup-routing.module.ts.

ملف login-routing.module.ts

```

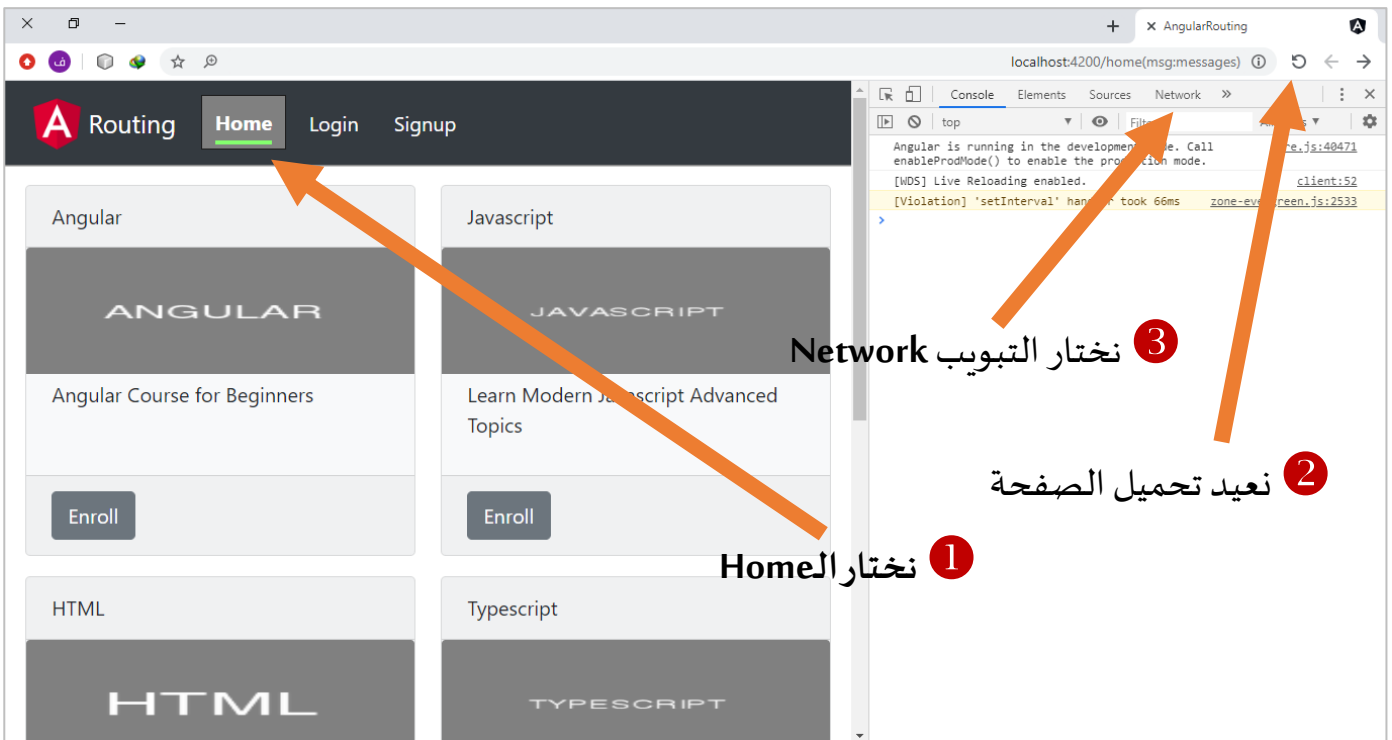
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { CanDeactivateLoginGuard } from 'src/app/guards/can-deactivate.guard';
4. import { LoginComponent } from './login.component';
5.
6. const routes: Routes = [
7.     {
8.         path: '',
9.         canDeactivate: [CanDeactivateLoginGuard],
10.        component: LoginComponent
11.    }
12. ];
13. @NgModule({
14.     imports: [RouterModule.forChild(routes)],
15.     exports: [RouterModule]
16. })
17. export class LoginRoutingModule {}

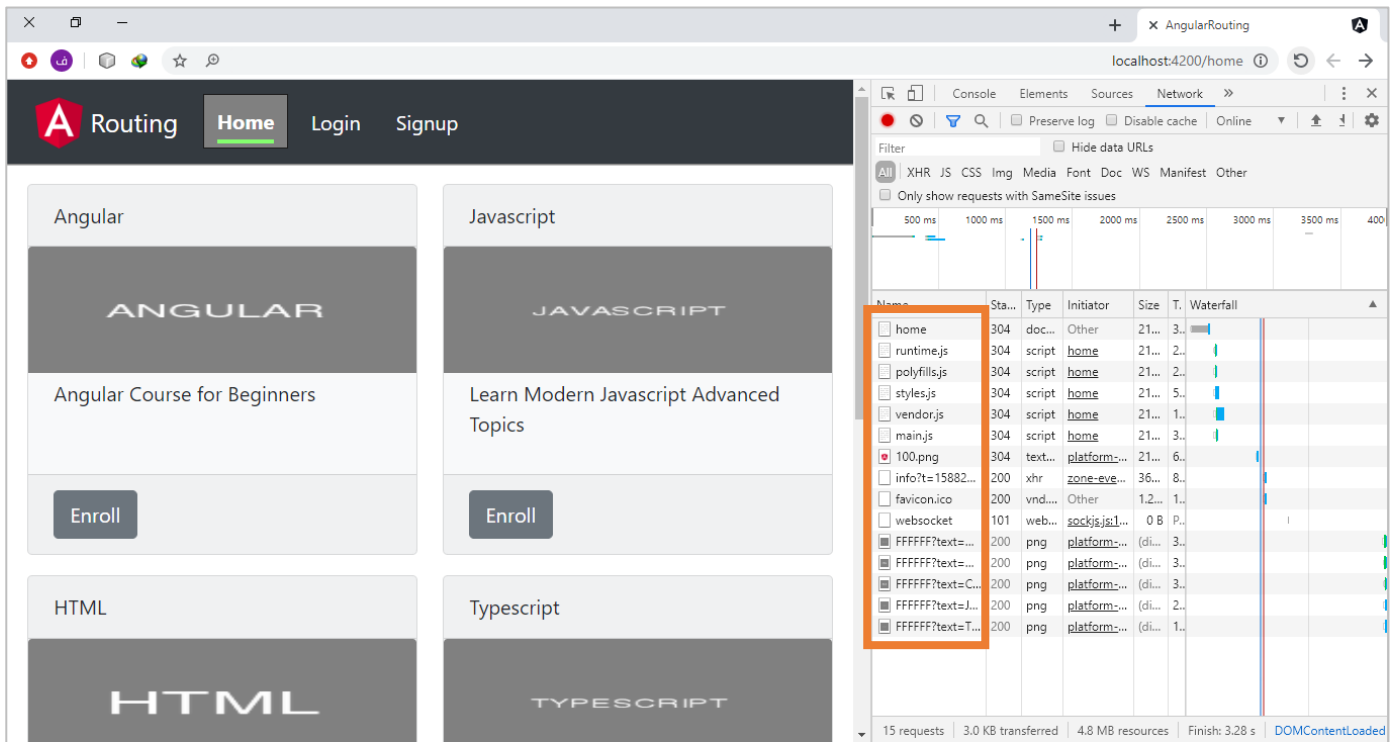
```

ملف signup-routing.module.ts

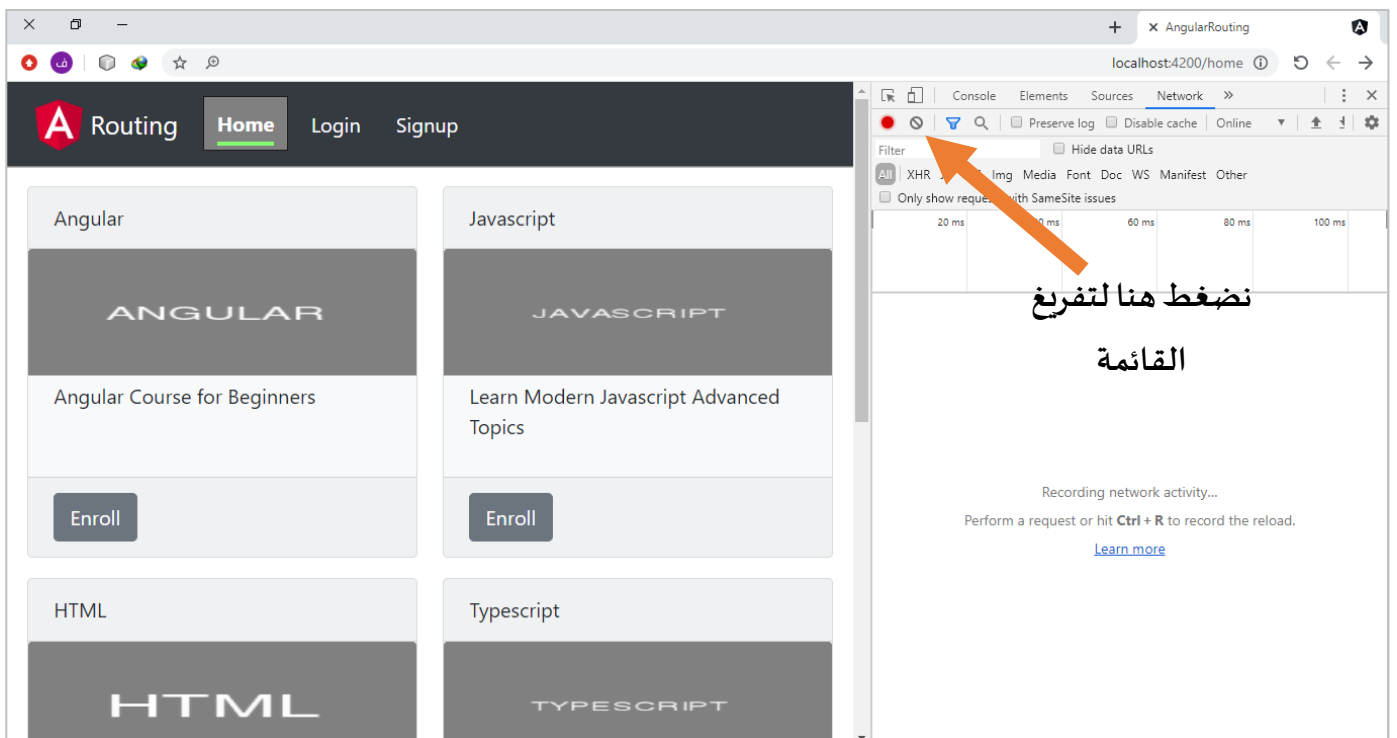
```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { CanDeactivateGuard } from 'src/app/guards/can-deactivate.guard';
4. import { SignupComponent } from './signup.component';
5.
6. const routes: Routes = [
7.   {
8.     path: '',
9.     canDeactivate: [CanDeactivateGuard],
10.    component: SignupComponent
11.  },
12. ];
13.
14. @NgModule({
15.   imports: [RouterModule.forChild(routes)],
16.   exports: [RouterModule]
17. })
18. export class SignupRoutingModule { }
```

الآن لنقم بحفظ جميع التعديلات ونذهب إلى المتصفح ونرى النتيجة، كالتالي:





نلاحظ في هذه القائمة تم تحميل home لأننا لم نعمل له Lazy Loading وتم تحميله في بداية تشغيل هذا المشروع، الآن لنضغط على زر تفريق هذه القائمة، كالتالي:



الآن لنختار التبويب Login ونرى النتيجة:

لاحظ انه في حال الانتقال إلى التبويب Login سوف يتم توجيه الرابط URL إلى path ذو القيمة login وعندها سوف يتم تحميل Feature Modules (authenticationModule - LoginModule). وهذا هو المتوقع (Lazy Loading)

الآن لنختار التبويب Signup ونرى النتيجة:

هنا قام بتحميل SignupModule فقط، وهذا هو المتوقع

الآن لنقوم بتأكد من بقية Feature Modules وهل هنالك أخطاء في Lazy Loading أم لا، ولنقم بتسجيل الدخول عن طريق الذهاب إلى التبويب Login واجراء عملية تسجيل الدخول، كالتالي:

AngularRouting

Home Edit Courses Users Profile Signout

Angular

ANGULAR

Angular Course for Beginners

Enroll

Javascript

JAVASCRIPT

Learn Modern Javascript Advanced Topics

Enroll

HTML

HTML

Typescript

TYPESCRIPT

Network

Name	Status	Type	Initiator	Size	T.	Waterfall
signup	304	document	:4200/vendor.js9...	210 B	3...	
runtime.js	304	script	signup	211 B	9...	
polyfills.js	304	script	signup	212 B	1...	
styles.js	304	script	signup	213 B	1...	
vendor.js	304	script	signup	213 B	3...	
main.js	304	script	signup	212 B	2...	
100.png	304	text/plain	platform-browse...	211 B	4...	
authentication-authentica...	304	script	bootstrap:149	211 B	7...	
info/?t=1588275719745	200	xhr	zone-evergreenj...	368 B	5...	
common.js	304	script	bootstrap:149	211 B	4...	
signup-signup-module.js	304	script	bootstrap:149	211 B	6...	
websocket	101	websocket	sockjs.js:1684	0 B	P...	
login-login-module.js	304	script	bootstrap:149	211 B	3...	
msg-msg-module.js	304	script	bootstrap:149	211 B	3...	
FFFFFF?text=ANGULAR	200	png	platform-browse...	(disk c...	2...	
FFFFFF?text=HTML	200	png	platform-browse...	(disk c...	2...	
FFFFFF?text=CSS	200	png	platform-browse...	(disk c...	1...	
FFFFFF?text=JAVASCRIPT	200	png	platform-browse...	(disk c...	1...	
FFFFFF?text=TYPESCRIPT	200	png	platform-browse...	(disk c...	1...	

19 requests | 2.8 KB transferred | 4.9 MB resources | Finish: 2.9 min | DOMContentLoaded: 4.60 s | Lc

AngularRouting

Home Edit Courses Users Profile Signout

Hi ( Faisal ) Logged in at Thu Apr 30 2020 - 10:48:56 μ

Search Users Here

Faisal View Active

Saad View Active

Bader View Active

Fahd View Active

Network

Name	Status	Type	Initiator	Size	T.	Waterfall
users-users-module.js	200	script	bootstrap:149	37.7 KB	3...	
222.png	304	png	platform-browse...	211 B	1...	

2 requests | 37.9 KB transferred | 42.6 KB resources



Routing Home Edit Courses **Users** Profile Signout

Hi ( Faisal ) Logged in at Thu Apr 30 2020 - 10:48:56 م

**Faisal**  
 User ID: 1  
 UserName: DevFaisal  
 User Password: 1234  
 User City: Riyadh  
 User Type: admin

Back Next User

localhost:4200/users(msg:messages)

Name	Status	Type	Initiator	Size	T...	Waterfall
users-users-module.js	200	script	bootstrap:149	37.7 KB	3...	
222.png	304	png	platform-browse...	211 B	1...	
user-details-user-details-...	200	script	bootstrap:149	8.3 KB	3...	

3 requests | 46.2 KB transferred | 50.6 KB resources

Routing Home Edit Courses Users **Profile** Signout

Hi ( Faisal ) Logged in at Thu Apr 30 2020 - 10:48:56 م

**Faisal**  
 User ID: 1  
 UserName: DevFaisal  
 User Password: 1234  
 User City: Riyadh  
 User Type: admin

localhost:4200/profile/1(msg:messages)

Name	Status	Type	Initiator	Size	T...	Waterfall
users-users-module.js	200	script	bootstrap:149	37.7 KB	3...	
222.png	304	png	platform-browse...	211 B	1...	
user-details-user-details-...	200	script	bootstrap:149	8.3 KB	3...	
profile-profile-module.js	200	script	bootstrap:149	17.4 KB	3...	

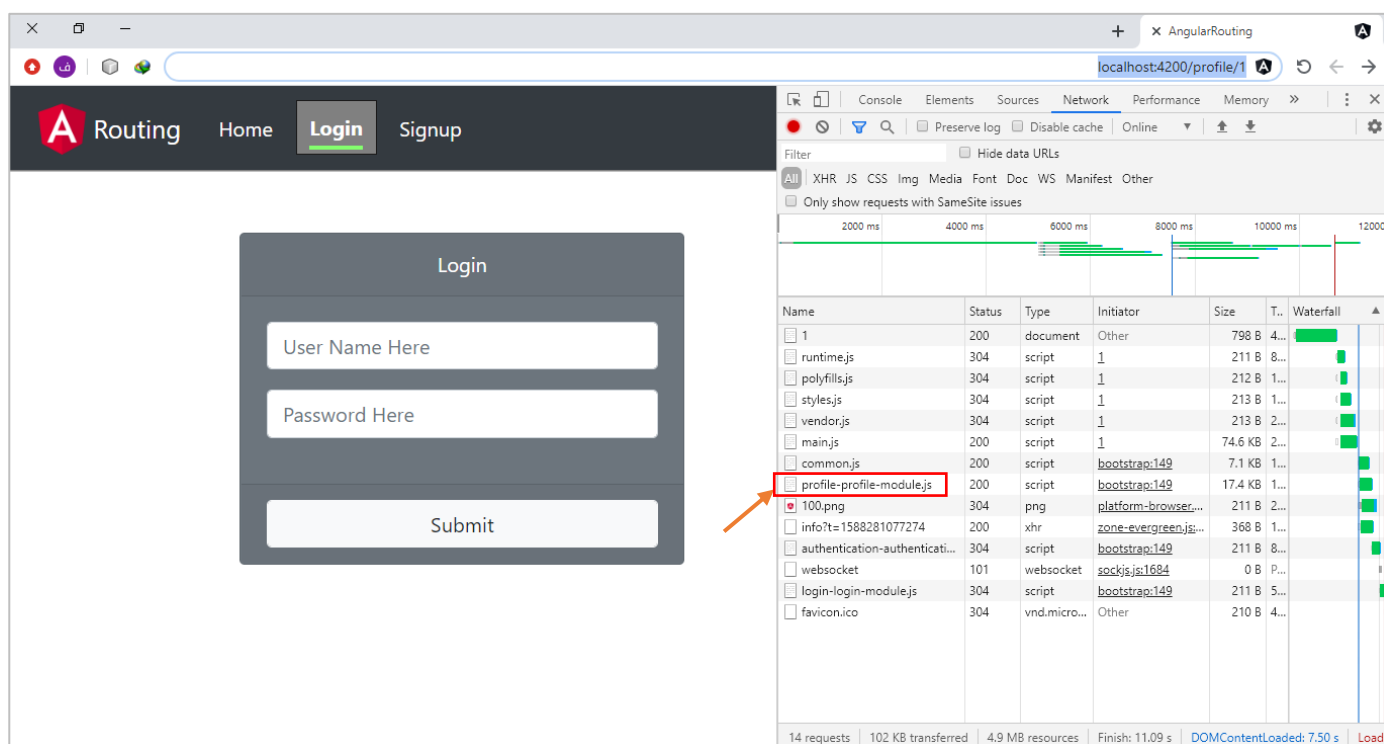
4 requests | 63.6 KB transferred | 63.7 KB resources

نلاحظ ان جميع Lazy Loading يعمل بدون أي مشاكل او أخطاء، وهذه الطريقة رفعنا من كفاءة وسرعة التطبيق الخاص بنا.

كما ان angular يقدم لنا استراتيجيات أخرى غير هذه الاستراتيجية لرفع كفاءة وسرعة التطبيق اسمها Preloading، ولكن قبل أن نستعرض مفاهيم هذه الاستراتيجية لنقم بشرح نوع من أنواع Route Guards وهو CanLoad.

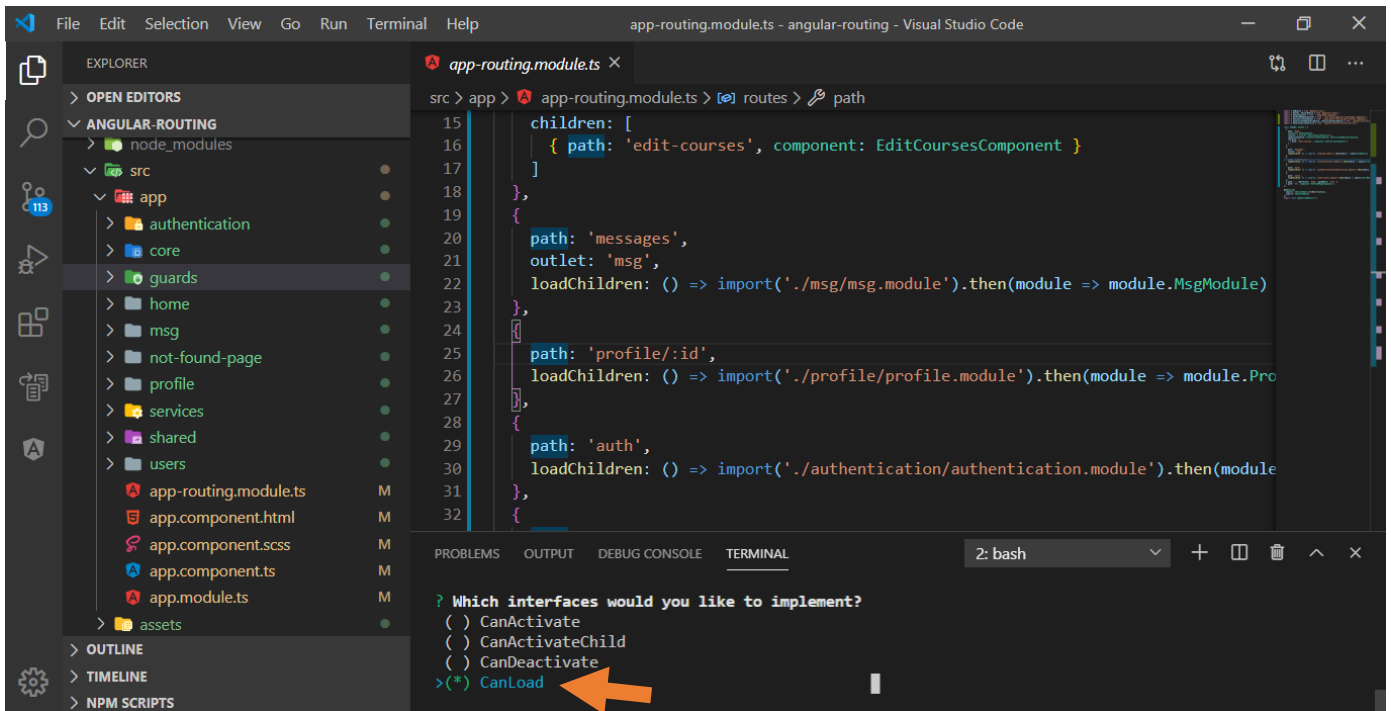
### 3.5. CanLoad Guard:

هذا النوع يتعامل مع Lazy Loading بحيث يسمح أو يمنع تحميل Feature Module معين وفق شروط معينة، اما لتوضيح الحاجة التي دعت angular لتقديم هذا النوع من guards هو عن طريق إعطاء مثال لكي يتضح المفهوم، وهو في حال أردنا الانتقال إلى تبويب Profile والمستخدم لم يقوم بتسجيل الدخول (بالطبع في مشروعنا هنا تبويب Profile تم اخفائه في حال ان المستخدم لم يقوم بتسجيل الدخول) ولكن نستطيع كتابة الرابط في المتصفح <http://localhost:4200/profile/1> لمحاولة استعراض هذه الصفحة بدون أن نقوم بتسجيل الدخول، ولنرى النتيجة:

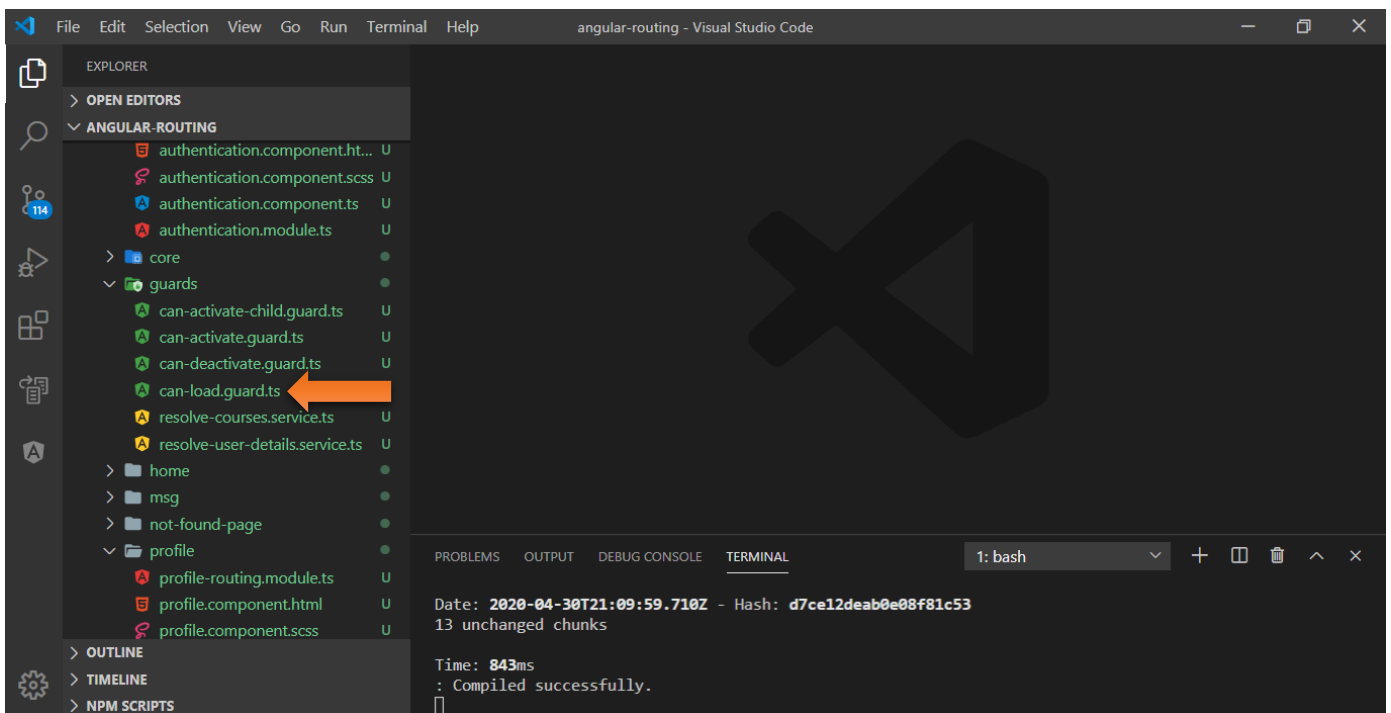


نلاحظ انه تم تحميل Module رقم انه منعنا من الدخول إلى هذه الصفحة، لذلك دعت الحاجة إلى تقديم هذا النوع من Guards وهو ليس فقط من المستخدم من دخول هذه الصفحة ولكن أيضاً من Module من التحميل.

اما انشاء هذا النوع عملياً فهو مشابهة للأنواع الأخرى، وذلك عن طريق كتابة الأمر التالي في terminal، ومن ثم اختيار النوع وهو CanLoad، كالتالي: `ng g guard guards/can-load`



ثم نضغط Enter من لوحة المفاتيح وسوف يتم انشاء الملف التالي:



ولو استعرضنا محتويات هذا الملف لوجدناها بالشكل التالي:

```

ملف can-load.guard.ts
1. import { Injectable } from '@angular/core';
2. import { CanLoad, Route, UrlSegment, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree } from '@angular/router';
3. import { Observable } from 'rxjs';
4. import { UsersService } from '../services/users-data/users.service';
5. import { map } from 'rxjs/operators';
6.
7. @Injectable({

```



```

8.   providedIn: 'root'
9. })
10. export class CanLoadGuard implements CanLoad {
11.   constructor(private userService: UsersService) { }
12.   canLoad(
13.     route: Route,
14.     segments: UrlSegment[]): Observable<boolean> | Promise<boolean> | boolean {
15.     return true;
16.   }
17. }
18.

```

نلاحظ انه مشابه لما سبقه من الأنواع الأخرى، والآن لنقم بكتابة Logic التالي:

ملف can-load.guard.ts

```

1. import { Injectable } from '@angular/core';
2. import {
3.   CanLoad,
4.   Route,
5.   UrlSegment,
6.   ActivatedRouteSnapshot,
7.   RouterStateSnapshot,
8.   UrlTree,
9.   Router
10. } from '@angular/router';
11. import { Observable } from 'rxjs';
12. import { UsersService } from '../services/users-data/users.service';
13. import { map } from 'rxjs/operators';
14.
15. @Injectable({
16.   providedIn: 'root'
17. })
18. export class CanLoadGuard implements CanLoad {
19.   constructor(
20.     private userService: UsersService,
21.     private router: Router
22.   ) { }
23.
24.   canLoad(
25.     route: Route,
26.     segments: UrlSegment[]): Observable<boolean> | Promise<boolean> | boolean {
27.     return this.userService.isLogin$.pipe(
28.       map((isLogin: boolean) => {
29.         if (isLogin) { return true; }
30.         this.router.navigate(['auth/login']);
31.         return false;
32.       })
33.     );
34.   }
35. }

```

راجع الاسطر من 27 إلى 32.

الآن لنذهب إلى ملف تهيئة Routes الرئيسي app-routing.module.ts، ونضع هذا الـ routed guard المستهدف، كالتالي:

#### ملف app-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { HomeComponent } from '../home/home.component';
4. import { NotFoundPageComponent } from '../not-found-page/not-found-page.component';
5. import { EditCoursesComponent } from '../home/edite-courses/edit-courses.component';
6. import {
7.   CanActivateAdminChildGuard,
8.   CanActivateChildGuard
9. } from '../guards/can-activate-child.guard';
10. import { ResolveCoursesService } from '../guards/resolve-courses.service';
11. import { CanLoadGuard } from '../guards/can-load.guard';
12.
13. const routes: Routes = [
14.   {
15.     path: 'home',
16.     component: HomeComponent,
17.     resolve: { courses: ResolveCoursesService },
18.     canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
19.     children: [
20.       { path: 'edit-courses', component: EditCoursesComponent }
21.     ]
22.   },
23.   {
24.     path: 'messages',
25.     outlet: 'msg',
26.     loadChildren: () =>
27.       import('../msg/msg.module').then(module => module.MsgModule)
28.   },
29.   {
30.     path: 'profile/:id',
31.     canLoad: [CanLoadGuard],
32.     loadChildren: () =>
33.       import('../profile/profile.module').then(module => module.ProfileModule)
34.   },
35.   {
36.     path: 'auth',
37.     loadChildren: () =>
38.       import('../authentication/authentication.module')
39.       .then(module => module.AuthenticationModule)
40.   },
41.   {
42.     path: 'users',
43.     loadChildren: () =>
44.       import('../users/users.module').then(module => module.UsersModule)
45.   },
46.   { path: '', redirectTo: 'home', pathMatch: 'full' },
47.   { path: '**', component: NotFoundPageComponent }
48. ];
49.
```

```

50. @NgModule({
51.   imports: [RouterModule.forRoot(routes)],
52.   exports: [RouterModule]
53. })
54. export class AppRoutingModule { }
55.

```

راجع السطر 31.

وبما اننا وضعنا هذا Guard لنحذف الأول وهو من النوع canActivate، لأن كلا النوعين يقومان بالتأكد هل المستخدم قام بتسجيل الدخول ام لا، فإذا قام بتسجيل الدخول يُسمح له بعرض الصفحة وإذا لم يتم يمنع من الوصول إلى هذه الصفحة، والفرق الوحيد ان CanLoad تمنع ايضاً من تحميل Feature Module اذ لم يتم تسجيل الدخول.

لذلك لنذهب إلى ملف profile-routing.module.ts ولنقم بحذف canActivate، كالتالي:

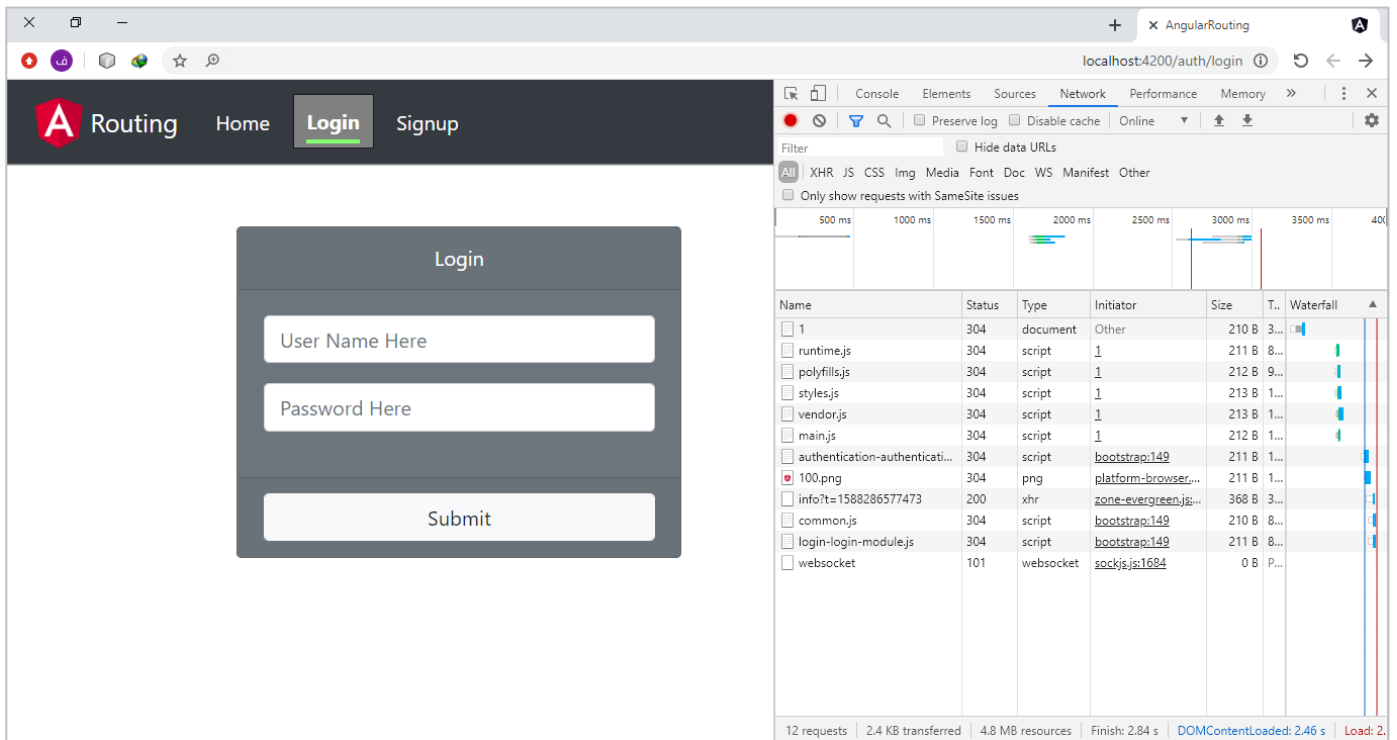
ملف profile-routing.

```

1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3. import { ProfileComponent } from './profile.component';
4.
5. const routes: Routes = [
6.   {
7.     path: '',
8.     component: ProfileComponent
9.   }
10. ];
11.
12. @NgModule({
13.   imports: [RouterModule.forChild(routes)],
14.   exports: [RouterModule]
15. })
16. export class ProfileRoutingModule { }

```

الآن لنحفظ التعديلات ونذهب إلى المتصفح، ونكتب الرابط السابق ونضغط Enter ونرى النتيجة، كالتالي:



نلاحظ انه قام بإلغاء Route وقام بتحويلنا إلى صفحة Login وبنفس الوقت لم يقوم بتحميل ProfileModule.

الآن لننتقل إلى الاستراتيجية الثانية PreLoading في الجزء التالي.

## 4.5. Preloading

تكمّن فكرة هذه الاستراتيجية في اننا نقوم بتحميل Feature Modules في الخلفية اثناء تشغيل المشروع، بمعنى في بداية تشغيل المشروع يتم تحميل الأجزاء الرئيسية من المشروع وعرضها للمستخدم، ومن ثم يقوم angular بتحميل جميع Feature Modules التي عملنا لها Lazy Loading في الخلفية، وبالتالي قللنا من زمن انتظار المستخدم، فبدلاً من ان يقوم بالانتظار في البداية إلى أن يقوم بتحميل الأجزاء الرئيسية ومن ثم ينتظر في كل مرة ينتقل إلى جزء من أجزاء المشروع معمول له Lazy Loading اصبح المستخدم ينتظر فقط في بداية تحميل المشروع فقط لعرض الأجزاء الرئيسية.

وهناك نقطة أخيرة أي Route قمنا بحمايته باستخدام CanLoad فلن يعمل مع هذه الاستراتيجية، لذلك لابد أن نغير نوع الحماية إلى CanActivate على سبيل المثال.

اما طريقة اعداد هذا النوع فهو ابسط مما تتصور عزيزي المتعلم، فإذا اتقنت بناء Feature Modules وايضاً ال Lazy Loading فكل الذي سوف نعمله هو نضيف امر معين على شكل كائن object في الدالة forRoot، لذلك لنذهب إلى ملف app-routing.module.ts، ونضيف هذا الامر، كالتالي:

ملف app-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule, PreloadAllModules } from '@angular/router';
3. import { HomeComponent } from './home/home.component';
4. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
5. import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
6. import {
```

```

7.   CanActivateAdminChildGuard,
8.   CanActivateChildGuard
9. } from './guards/can-activate-child.guard';
10. import { ResolveCoursesService } from './guards/resolve-courses.service';
11. import { CanLoadGuard } from './guards/can-load.guard';
12.
13. const routes: Routes = [
14.   {
15.     path: 'home',
16.     component: HomeComponent,
17.     resolve: { courses: ResolveCoursesService },
18.     canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
19.     children: [
20.       { path: 'edit-courses', component: EditCoursesComponent }
21.     ]
22.   },
23.   {
24.     path: 'messages',
25.     outlet: 'msg',
26.     loadChildren: () =>
27.       import('./msg/msg.module').then(module => module.MsgModule)
28.   },
29.   {
30.     path: 'profile/:id',
31.     canLoad: [CanLoadGuard],
32.     loadChildren: () =>
33.       import('./profile/profile.module').then(module => module.ProfileModule)
34.   },
35.   {
36.     path: 'auth',
37.     loadChildren: () =>
38.       import('./authentication/authentication.module')
39.       .then(module => module.AuthenticationModule)
40.   },
41.   {
42.     path: 'users',
43.     loadChildren: () =>
44.       import('./users/users.module').then(module => module.UsersModule)
45.   },
46.   { path: '', redirectTo: 'home', pathMatch: 'full' },
47.   { path: '**', component: NotFoundPageComponent }
48. ];
49.
50. @NgModule({
51.   imports: [RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules })],
52.   exports: [RouterModule]
53. })
54. export class AppRoutingModule { }

```

راجع السطر 51.

مبروك عزيزي المتعلم فقد استخدمت هذه الاستراتيجية لرفع كفاءة التطبيق الخاص بك، فسوف يقوم angular بجميع العمل بالنيابة عنك فسوف يبحث عن كل Feature Module معمول له Lazy Loading ويقوم بتحويله إلى PreLoading.

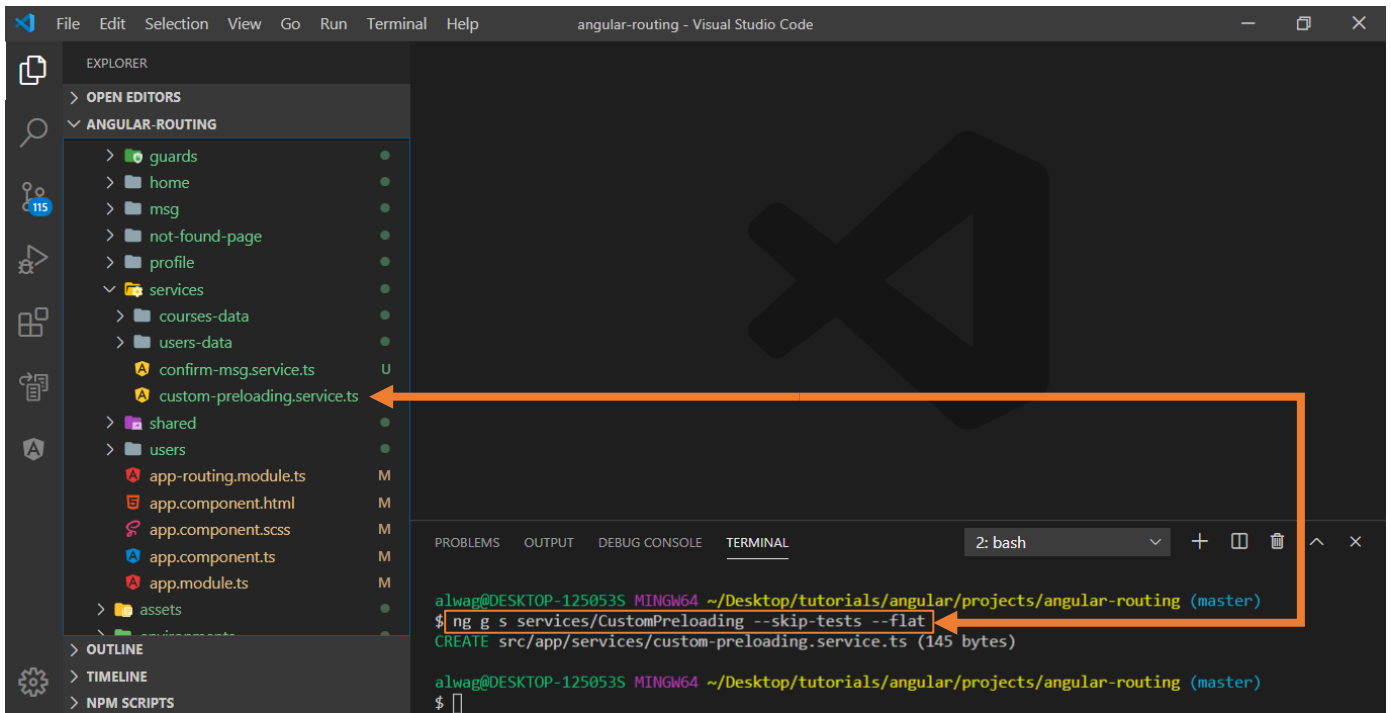
ولكن هنالك سؤال يطرح نفسه ماذا لو اردنا ان نحدد أنواع معينة يطبق فيها PreLoading وأنواع أخرى نبقىها على حالها Lazy Loadin، ومن أشهر التطبيقات هو في حالة الأجزاء من التطبيق لا يملك حق الوصول إليها إلا مدير النظام ففي هذه الحالة نجعل هذه الأجزاء Lazy Loading اما باقي الأجزاء التي يطلع عليها الزوار او أعضاء موقعك المسجلين فنجعلها PreLoading.

اما طريقة بناء هذه الطريقة المختلطة فتحتاج إلى كتابة بعض الاكواد، وسوف ارتبها على شكل مجموعة من الخطوات، كالتالي:

- (١) أولاً نحتاج إلى service وهذه service تقوم بعمل لـ PreLoadingStrategy implements.
- (٢) ومن ثم نضيف الدالة (preload)، وتُعيد قيمة بحسب الاحتياج وهنا نريدها ان تُعيد <any> Observable.
- (٣) وهذه الدالة تستقبل باراميتين احدهما من النوع Route والآخر من النوع Function.
- (٤) وبعدها نكتب محتوى هذه الدالة من خلال الاعتماد على الخاصية data التي شرحناها سابقاً حيث نُرسل قيمة منطقية true او false بحيث إذا كانت true نُعيد البارامتر الذي من النوع Function أي أن هذا Route اجعله Preloading، اما إذا كان false او لم يكن هنالك أي خاصية data في route فنعيد null.
- (٥) نضع الخاصية data لـ Routed المستهدف ونمرر له كائن يحتوي هذا الكائن على خاصية قيمتها منطقية.
- (٦) وأخيراً نذهب إلى الامر الذي كتبناه سابقاً وهو الكائن {preloadingStrategy: PreloadAllModules}، ونغير القيمة PreloadAllModules إلى اسم الكلاس في الـ Service التي قمنا بإنشاءها.

والآن لنقم بتطبيق هذه الخطوات عملياً، كالتالي:

أولاً لنقوم بإنشاء service جديدة، كالتالي:



بعد إنشاء هذه service لنقوم الآن باستعراض محتويات هذا الملف، كالتالي:

ملف custom-preloading.service.ts

```
1. import { Injectable } from '@angular/core';
2.
3. @Injectable({
4.   providedIn: 'root'
5. })
6. export class CustomPreloadingService {
7.
8.   constructor() { }
9. }
```

نلاحظ انه ملف service عادي الآن لنقم بعمل implements لinterface ذو الاسم PreloadingStrategy وبنفس الوقت نضيف الدالة preload() ونسعي البارامترات والانواع، كالتالي:

ملف custom-preloading.service.ts

```
1. import { Injectable } from '@angular/core';
2. import { Route, PreloadingStrategy } from '@angular/router';
3. import { Observable } from 'rxjs';
4.
5. @Injectable({
6.   providedIn: 'root'
7. })
8. export class CustomPreloadingService implements PreloadingStrategy {
9.
10.   constructor() { }
11.
12.   preload(route: Route, load: Function): Observable<any> {}
13.
14. }
```

الآن لنكتب محتوى هذه الدالة، من خلال وضع شرط إذا كان هذا Route يحتوي على الخاصية data وبنفس الوقت هذه الخاصية تحمل قيمة هذه القيمة تساوي true فقم بإعادة هذه الدالة اما غير ذلك فقم بإعادة null، كالتالي:

ملف custom-preloading.service.ts

```
1. import { Injectable } from '@angular/core';
2. import { Route, PreloadingStrategy } from '@angular/router';
3. import { Observable, of } from 'rxjs';
4.
5. @Injectable({
6.   providedIn: 'root'
7. })
8.
9. export class CustomPreloadingService implements PreloadingStrategy {
10.
11.   constructor() { }
12.
13.   preload(route: Route, load: Function): Observable<any> {
14.     if (route.data && route.data.preload) {
15.       return load();
16.     }
17.     return of(null);
18.   }
19.
20. }
```

راجع الاسطر من 14 إلى 17.

ملاحظة: الخاصية preload الموجودة في السطر 14 لم نقوم بإنشائها بعد، ولك حرية اختيار الاسم الذي تريده.

الآن لنذهب إلى Route المستهدف وليكن مثلاً كلاً من users ونضيف له الخاصية data والقيمة هي كائن object تحتوي على الخاصية preload التي اشرت إليها قبل قليل، ونسند لها القيمة true بمعنى اجعل هذا Route يطبق استراتيجية Preloading، كالتالي:

ملف app-routing.module.ts

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule, PreloadAllModules } from '@angular/router';
3. import { HomeComponent } from './home/home.component';
4. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
5. import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
6. import {
7.   CanActivateAdminChildGuard,
8.   CanActivateChildGuard
9. } from './guards/can-activate-child.guard';
10. import { ResolveCoursesService } from './guards/resolve-courses.service';
11. import { CanLoadGuard } from './guards/can-load.guard';
12.
13. const routes: Routes = [
```



```

14. {
15.   path: 'home',
16.   component: HomeComponent,
17.   resolve: { courses: ResolveCoursesService },
18.   canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
19.   children: [
20.     { path: 'edit-courses', component: EditCoursesComponent }
21.   ]
22. },
23. {
24.   path: 'messages',
25.   outlet: 'msg',
26.   loadChildren: () =>
27.     import('./msg/msg.module').then(module => module.MsgModule)
28. },
29. {
30.   path: 'profile/:id',
31.   canLoad: [CanLoadGuard],
32.   loadChildren: () =>
33.     import('./profile/profile.module').then(module => module.ProfileModule)
34. },
35. {
36.   path: 'auth',
37.   loadChildren: () =>
38.     import('./authentication/authentication.module')
39.     .then(module => module.AuthenticationModule)
40. },
41. {
42.   path: 'users',
43.   data: { preload: true },
44.   loadChildren: () =>
45.     import('./users/users.module').then(module => module.UsersModule)
46. },
47. { path: '', redirectTo: 'home', pathMatch: 'full' },
48. { path: '**', component: NotFoundPageComponent }
49. ];
50.
51. @NgModule({
52.   imports: [RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules })],
53.   exports: [RouterModule]
54. })
55. export class AppRoutingModule { }
56.

```

راجع السطر 43.

وأخيراً نقوم بتغيير القيمة المسندة للخاصية `preloadingStrategy` (في السطر 52) إلى اسم الكلاس لـ `service` التي أضفناها قبل قليل، كالتالي:

ملف `app-routing.module.ts`

```

1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule, PreloadAllModules } from '@angular/router';

```

```

3. import { HomeComponent } from './home/home.component';
4. import { NotFoundPageComponent } from './not-found-page/not-found-page.component';
5. import { EditCoursesComponent } from './home/edite-courses/edit-courses.component';
6. import {
7.   CanActivateAdminChildGuard,
8.   CanActivateChildGuard
9. } from './guards/can-activate-child.guard';
10. import { ResolveCoursesService } from './guards/resolve-courses.service';
11. import { CanLoadGuard } from './guards/can-load.guard';
12. import { CustomPreloadingService } from './services/custom-preloading.service';
13.
14. const routes: Routes = [
15.   {
16.     path: 'home',
17.     component: HomeComponent,
18.     resolve: { courses: ResolveCoursesService },
19.     canActivateChild: [CanActivateChildGuard, CanActivateAdminChildGuard],
20.     children: [
21.       { path: 'edit-courses', component: EditCoursesComponent }
22.     ]
23.   },
24.   {
25.     path: 'messages',
26.     outlet: 'msg',
27.     loadChildren: () =>
28.       import('./msg/msg.module').then(module => module.MsgModule)
29.   },
30.   {
31.     path: 'profile/:id',
32.     canLoad: [CanLoadGuard],
33.     loadChildren: () =>
34.       import('./profile/profile.module').then(module => module.ProfileModule)
35.   },
36.   {
37.     path: 'auth',
38.     loadChildren: () =>
39.       import('./authentication/authentication.module')
40.       .then(module => module.AuthenticationModule)
41.   },
42.   {
43.     path: 'users',
44.     data: { preload: true },
45.     loadChildren: () =>
46.       import('./users/users.module').then(module => module.UsersModule)
47.   },
48.   { path: '', redirectTo: 'home', pathMatch: 'full' },
49.   { path: '**', component: NotFoundPageComponent }
50. ];
51.
52. @NgModule({
53.   imports: [RouterModule.forRoot(routes, { preloadingStrategy: CustomPreloadingService
    })],

```

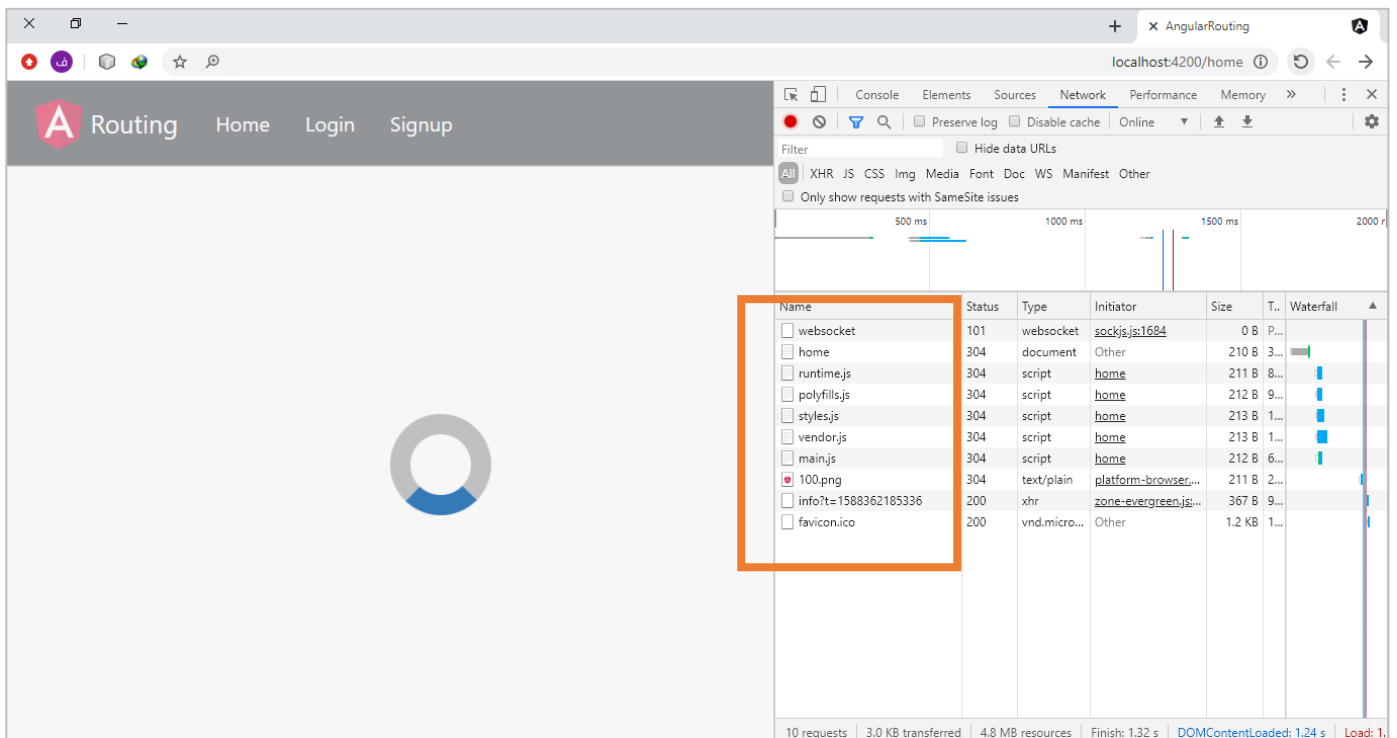
```

54. exports: [RouterModule]
55.})
56.export class AppRoutingModule { }

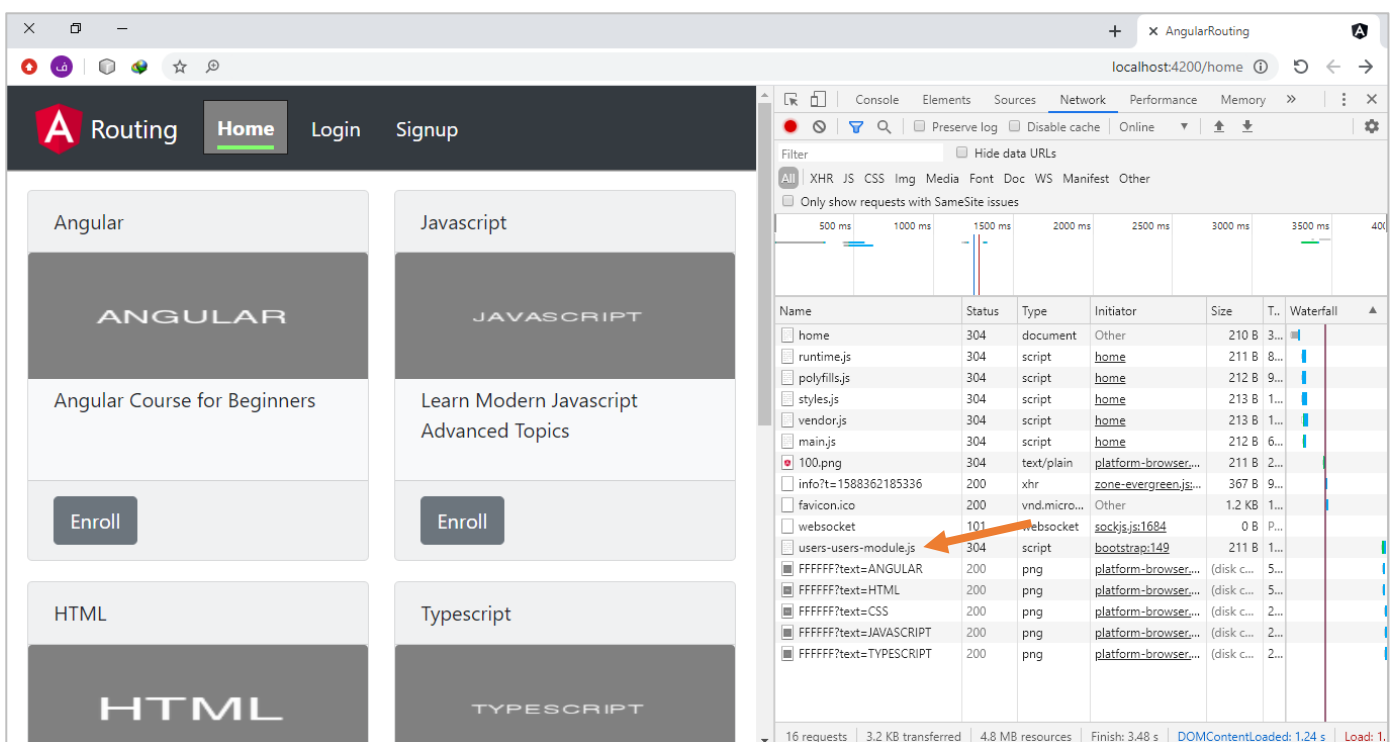
```

راجع السطر 53.

الآن لنحفظ التعديلات ونذهب إلى المتصفح ونرى النتيجة:



نلاحظ أنه قام بتحميل الملفات التي لم نعمل لها Lazy Loading بالإضافة إلى الملفات الضرورية لتشغيل المشروع، وبعد الانتظار قليلاً سوف يظهر لنا Feature Module الخاص بـ users، كالتالي:



نلاحظ انه قام بالانتهاء من التحميل.

# المراجع

## References

## References:

1. Freeman, Adam, **Pro Angular 9**, Apress, 2020.
2. Agarwal, Uttam, **Hands-On Full Stack Development with Angular 5 and Firebase**, Packt, 2018.
3. Delaney, Jeff, **The Angular Firebase Survival Guide**, leanpub, 2018.
4. Saha, Depasis, **Angular 7.0 for Beginners**, 2018.
5. Vijay, Santhosh, **Material for Angular Developer Course**, Leanpub, 2019.
6. Hussain, Asim, **Angular From Theory to Practice**, 2017.
7. D. Booth, Joseph, **Angular Succinctly**, Syncfusion, 2019.
8. Squad, Ninja, **Become a ninja with Angular**, 2019.
9. Steyer, Manfred, **Enterprise Angular**, Leanpub, 2020.
10. Clow, Mark, **Angular 5 Projects**, Apress 2018.
11. Nate Murray, Felipe Coury, Ari Lerner, and Carlos Taborda, **ng-book The Complete Guide to Angular**, Fullstack.io, 2018.
12. Bouchefra, Ahmed, **Practical Angular: Build your first web apps with Angular 8**, Leanpub, 2019.
13. Hajian, Majid, **Progressive Web Apps with Angular**, Apress, 2019.
14. Savkin, Victor, **Angular Router**, Leanpub, 2018.