เส้นทาง (Routing)

บทนี้เป็นบทต่อเนื่องจาก Service ซึ่งจำเป็นต้องใช้ไฟล์ต่าง ๆ ที่ได้สร้างในบทที่ผ่านมาใช้งาน เพื่อกำหนดเส้นทางเพิ่มเติม โดยมีการสร้างเส้นทางในสองลักษณะคือ เส้นทางที่วิ่งไปยังหน้าคอมโพเน้นท์โดยตรง เช่น หน้าที่แสดงข้อมูลทั้งหมดของผู้ใช้งาน กับวิ่งไปยังหน้าคอมโพเน้นท์แบบมีตัวแปรเพิ่มเติม เช่น การส่ง รหัสผู้ใช้งาน (id) เพื่ออ่านเฉพาะราย ซึ่งต้องเรียกใช้โมดูลเฉพาะคือ router จากการเรียกใช้คลาส RouterModule และ Routes ดังจะได้ทำความเข้าใจในประเด็นต่อไปนี้

- การสร้างโมดูลเส้นทาง และลำดับของเส้นทาง
- การนำทางทั้งบนไฟล์ TS และ HTML
- การสร้างเส้นทางแบบมีตัวแปร
- การสร้างเส้นทางในโมดูลย่อย
- การป้องเข้าและออกเส้นทาง

สำหรับแองกูล่าร์แล้ว วิธีที่ดีในการเก็บเส้นทางทั้งหมด ควรสร้างไฟล์เก็บข้อกำหนดเส้นทางต่างหาก เพื่อแยกเป็นโมดูลหนึ่งที่ทำหน้าที่เกี่ยวกับเส้นทางโดยเฉพาะ การใช้โมดูลเส้นทาง ทำให้มีหน้าหลัก และหน้าย่อย ๆ โดยหน้าหลักถือเป็นหนึ่งหน้าแต่ภายในหน้านี้สลับเปลี่ยนเป็นหน้าเว็บอื่น ๆ ได้มากมายตามเส้นทางที่กำหนดในโมดูลเส้นทาง จึงถือได้ว่าเป็นเว็บหน้าเดียวเป็นแกน (Single Page Application - SPA)

สร้างโมดูล AppRoutingModule

กรณีที่แอปฯ เดิม ยังไม่เคยได้สร้างโมดูลเส้นทางมาก่อน เราสามารถเพิ่มโมดูเส้นทาง AppRoutingModule ดังมีการสร้างดังนี้:

```
ng generate module app-routing --flat --module=app
```

โดยมีค่า ตามเงื่อนไขคือ

app-routing เป็นชื่อไฟล์ ซึ่งต่อไปจะกลายเป็นคลาส AppRoutingModule จะมีการเติม Module ต่อท้าย

--flat เป็นการสร้างไฟล์ ในโฟลเดอร์ src/app แทนที่จะสร้างเป็นโฟลเดอร์ของตนเอง

--module=app เป็นการลงทะเบียนนำเข้า หรือ import ใน AppModule

กรณีที่เริ่มต้นสร้างแอปฯ ขึ้นมาใหม่ และต้องการมีโมดูลเกี่ยวกับเส้นทางมาด้วย ให้ตอบคำถามการเพิ่มเส้นทางเป็น “Y” ดังรูปต่อไปนี้

```
D:\SBC\Courses\Angular\Code\06Routing>ng new myAngular
? Would you like to add Angular routing? (y/N) y
```

รูป 1 สร้างแอปฯขึ้นใหม่ แบบมี AppRoutingModule ขึ้นพร้อมกับ แอปฯ

หรือจะใช้คำสั่งใส่เส้นทางมาพร้อมมาด้วย:

```
ng new myAngular --routing
```

เมื่อสร้างโมดูลสำหรับกำหนดเส้นทางของเว็บแอปฯ แล้ว จะได้ไฟล์ดังนี้ (กรณีสร้างโมดูลภายหลัง ต้องปรับปรุงข้อมูลตามตัวอย่างต่อไปนี้)

Code 1. src/app/app-routing.module.ts (v1)

```
import { NgModule }             from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [];

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

จากตัวอย่างนี้ จะเห็นว่า โมดูลเส้นทางนี้ ไม่มีการใช้งานคำสั่งไดเร็กทีฟ (Directive) เช่น NgIF, NgForOf จึงไม่มีการนำเข้าคอมโพเน้นท์ CommonModule แต่ต้องการใช้เส้นทางผ่านโมดูล RouterModule และไทป์ Routes จากไลบรารี router จึงต้องนำเข้าโมดูลและไทป์นี้ และต้องส่งออก(export) RouterModule เพื่อให้คอมโพเน้นท์สามารถนำไปใช้ได้ รวมถึงโมดูลอื่น

นอกจากนี้ จะเห็นว่ามีเส้นทางเริ่มต้นถูกสร้างมาให้ด้วย จาก RouterModule.forRoot(routes) ตัวแปร routes คือค่าคงที่ ที่กำหนดแทนเส้นทางในรูปแบบอาเรย์ เช่น มีเส้นทาง (path: 'users') สำหรับแสดงผล UsersComponent ดังนั้นแล้วทุกค่าในค่าคงที่นี้ จะเป็นเส้นทางที่เริ่มจาก เส้นทางเริ่มต้น (root) ตาม URL เริ่มต้น ในที่นี้ URL เริ่มต้นคือ localhost:4200/ หากเราใส่ URL เป็น localhost:4200/users ก็จะแสดงค่า UsersComponent แต่ขณะนี้ยังเป็นอาร์เรย์ว่าง เพราะยังไม่มีเส้นทางใดถูกกำหนดให้ไปคอมโพเน้นท์ใด

สร้างเส้นทางเริ่มต้น

ทุกๆ เว็บแอปฯ ควรกำหนดเส้นทางเริ่มต้น ที่เป็นจุดอ้างอิงแรก สำหรับหน้าเว็บอื่นๆ ว่าอยู่บนเส้นทางระดับชั้นใดของระบบไฟล์ เพราะการอ้างอิงในระบบเว็บ มีลักษณะอ้างอิงเป็นชั้นๆ เช่น เมื่อเว็บหน้าหนึ่งอยู่ในระดับชั้นไป 2 โฟลเดอร์ การออกไปยังโฟลเดอร์แรก ก็จะใช้ เครื่องหมายจุดสองจุด (../) หรือไม่ก็ใช้ url แบบเต็ม เช่น http:/www.myapp.com แต่วิธีการแรกเป็นวิธีการที่ดีกว่า เพราะอยู่ในภายเซิร์บเวอร์เดียวกัน ไม่ต้องออกไปใช้บริการ http เต็มรูปแบบ

วิธีการที่อ้างอิงตนเอง คือการเพิ่ม อีลีเม้นท์ <base> ไว้ใน <head> ของไฟล์ html เริ่มต้น สำหรับแองกูล่าร์ มีโฟลเดอร์ src/app เป็นโฟล์งานทั้งหมด แต่เปิดหน้าเว็บบน src/index.html ซึ่งเป็นไฟล์เริ่มต้นนั้นเอง

Code 2. src/index.html

```
<head>
  <base href="/">
```

ไฟล์ index.html ไม่ได้เป็นตัวแสดงผลหน้าเว็บที่แท้จริง เพียงแต่เก็บ ส่วนที่จะแสดงผลซึ่งอยู่ใน โฟลเดอร์ src/app ซึ่งระบบจัดการเส้นทาง ส่วนแสดงเริ่มต้นจริงอยู่ที่ src/app/app.component.html จะนำส่วนเริ่มต้นเฉพาะเส้นทางมาร่วมแสดงผลด้วย เช่น localhost:4200/users จะนำส่วน UsersComponent มาแสดงผลร่วม ตามตัวแปรของเส้นทาง ที่ระบุไว้ใน routes โดยจะต้องเพิ่ม RouterOutlet

ดังนั้นแล้ว จากเดิมที่กำหนดตายตัวให้แสดงผลเดิมใน app.component.html จะต้องลบออกและแทนที่ด้วย RouteOutlet ดังแก้ไขได้ใหม่คือ:

Code 3. src/app/app.component.html

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
</div>
<router-outlet></router-outlet>
```

RouterOutlet เป็นคำสั่งในอิลีเม้นท์ตัวหนึ่ง ที่มาจาก การนำเข้า AppRoutingMoudule ของ app.module.ts ซึ่งได้มาจาก app-routing.module.ts

ถึงตอนนี้ทดสอบการทำงานได้แล้ว ใช้ CLI: ng serve --open และทดลองไปยังเส้นที่ตามค่าคงที่ ที่ระบุใน routes ทั้งแบบมีเพียงเส้นทางเริ่มต้น ซึ่งคือ localhost:4200/ และไม่มีเส้นใดกำหนดใน routes จึงเพียงได้แต่แสดงหน้า app.component.html

เพิ่มคอมโพเน้นท์ dashboard และ คลาส User

เพื่อทำความเข้าใจมากขึ้น ให้ทำเส้นทางเพิ่มอีก โดยให้สร้างคอมโพเน้นท์เพิ่มอีก ซึ่งจะเป็นอีกเส้นทางหนึ่งที่จะแสดงผลแบบอื่น ในที่นี้ให้เพิ่ม dashboard ด้วย CLI ต่อไปนี้

```
ng generate component dashboard
ng generate class user
```

หลังจากได้สร้างคอมโพเน้นท์ใหม่นี้แล้ว ให้ดัดแปลงตามไฟล์ต่อไปนี้ 4 ไฟล์ จะช่วยประหยัดเวลา แล้วมาทำความเข้าใจการทำงานกันต่อไป

1. index.html ใช้ CSS ของ Bootstrap ใส่ในส่วน <head>
2. user.ts เป็นการสร้างสมาชิกภายในคลาส
3. dashboard.component.ts เป็นคอมโพเน้นท์เริ่มต้นในการแสดงผล
4. dashboard.component.html เป็นหน้าแสดงผลที่แสดงรายชื่อทุกรายการ

Code 4. src/index.html

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
integrity="sha384-
Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
crossorigin="anonymous">
```

Code 5. src/app/user.ts

```
export class User {
    constructor(
        public id:number,
        public fname:string,
        public lname:string,
        public email:string) {}
}
```

Code 6. src/app/dashboard/dashboard.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User } from '../user';

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
  public users: User[];
  constructor() { }
  ngOnInit(): void {  }
}
```

Code 7. src/app/dashboard/dashboard.component.html

```
<h3 style='text-align:center'>Users</h3>
<div class="container">
  <div class="row row-cols-3" >
    <div class="col" *ngFor='let user of users' >
      <div class="card" style="width: 18rem; background-color:#eee;">
       <div class="card-body">
       <h5 class="card-title">{{user.fname}}</h5>
       <p class="card-text">{{user.id}} {{user.fname}} {{user.lname}}</p>
       <p class="card-link">{{user.email}}</p>
       </div>
      </div>
    </div>
  </div>
</div>
```

สำหรับไฟล์ dashboard.component.html มีคำสั่งที่น่าสนใจดังนี้

- *ngFor ใช้สำหรับการวนซ้ำของกลุ่มข้อมูล users โดยมีตัววิ่ง กำหนดเป็นชื่อ user
- มีการผูกข้อมูลทางเดียวด้วย {{ }}

ตอนนี้ข้อมูลยังไม่ได้สร้างขึ้น ให้จำลองข้อมูลจากไฟล์ user-data.ts โดยบรรจุ ออบเจ็กต์ User ต่าง ๆ ไว้

Code 8. src/app/user-data.ts

```
import { User } from './user';
export const USERS: User[] = [
  new User(1,'Pol', 'L.','pol@gmail.com'),
  new User(2,'Mon', 'T.','Mon@gmail.com'),
  new User(3,'Tee', 'F.','Tee@gmail.com'),
```

```
  new User(4,'Kon', 'A.','Kon@gmail.com'),
  new User(5,'Den', 'A.','Den@gmail.com'),
  new User(6,'Ten', 'A.','Ten@gmail.com')
];
```

สร้างเส้นทางไปยังคอมโพเน้นท์

ตอนนี้มีคลาสเส้นทางแล้ว (AppRoutingModule) ต่อไปก็กำหนดเส้นทางเดินของเว็บไซต์ เมื่อคลิกลิงค์ตาม URL โดยทั่วไปการกำหนดเส้นทาง กระทำผ่านคุณสมบัติสองอย่างคือ

- กำหนดเส้นทาง path ซึ่งหมายถึง URL ที่ใช้งาน
- กำหนดคอมโพเน้นท์ที่จะแสดงผล ผ่าน component ตามชื่อคอมโพเน้นท์

แต่ก่อนที่จะกำหนดเส้นทางได้จะต้องนำเข้าคอมโพเน้นท์ที่อ้างอิงก่อน แล้วค่อยกำหนดเส้นทางตามคุณสมบัติสองอย่างที่กล่าวมา โดยเก็บอยู่ในรูปอาร์เรย์ ดังตัวอย่างต่อไปนี้ กำหนดเส้นทางเริ่มต้น ไปยังคอมโพเน้นท์ (หรือหน้าเว็บ) dashboard

Code 9. src/app/app-routing.module.ts

```
import { DashboardComponent } from './dashboard/dashboard.component';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

const routes: Routes = [
  { path:'dashboard', component: DashboardComponent},
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];
```

เส้นทาง path มีค่าว่าง ไม่มีอักษรใด หรือตรงกับ path ค่าว่างนี้ ส่วน full แสดงว่า ให้ทำเป็นเส้นทางเริ่มต้น (/) ซึ่งหมายความว่าหากผู้ใช้ ไม่ระบุเส้นทางไปที่ใด ใส่เพียง localhost:4200 จะให้ไปยัง /dashboard ตามคำสั่ง redirecTo โดยที่ redirectTo จะไปที่ผ่าน dashboard ที่กำหนดไว้ในบรรทัดต่อมา นั้นเอง

อีกเส้นทางหนึ่ง (**) คือเส้นทางใดๆ หรือเส้นทางที่หาไม่พบ เช่น ผู้ใช้กรอก URL เป็น localhost:4200/book ซึ่งไม่มีในสาระบบเส้นทางใด จึงควรสร้างคอมโพเน้นท์รองรับเส้นทางที่หาไม่พบนี้เพื่อแจ้งว่าไม่มีหน้าเว็บบน URL นี้

```
ng generate component PageNotFound
```

และให้เขียนข้อความในหน้าคอมโพเน้นท์นี้แจ้งว่า ไม่มีหน้าเว็บที่ค้นหา กรณีที่ไม่ต้องการสร้างคอมโพเน้นท์นี้เพิ่ม ก็ให้เขียนเส้นทางไปโดยใช้ คำสั่ง redirectTo ยังหน้าเว็บเริ่มต้นก็ได้

> ถึงตอนนี้ถ้ามีอะไรผิดพลาด เช่น ระบบแจ้งว่า page-not-found หาไม่พบ ให้ปิดเซิร์บเวอร์ แล้วเปิดใหม่ โดยไปที่ CLI ที่กำลังเปิดเซิร์บเวอร์ให้แองกูล่าร์ทำงาน ให้กด Ctrl + C ระบบจะถามว่าให้ปิดหรือไม่ (Terminate) ให้ตอบ Y

นอกจากนี้ให้สังเกตว่า การกำหนดเส้นทาง ในอาร์เรย์ routes ลำดับของอาร์เรย์มีนัยสำคัญ กล่าวคือ ระบบเส้นทางจะอ่านจากเส้นทางแรกก่อน ดังนั้น การกำหนดเส้นทางที่หาไม่พบจึงควรเขียนไว้ลำดับสุดท้าย

เมื่อกำหนดเส้นทางเสร็จแล้วทดสอบผ่าน URL ได้ โดยระบุ URL ในเบราเซอร์เป็น localhost:4200/ หรือ localhost:4200/dashboard ซึ่งเป็นเส้นทางที่เพิ่งกำหนดไปนั้นเอง ซึ่งคงได้หน้าว่างเปล่า เพราะข้อมูลยังไม่ได้ใส่ในคอมโพเน้นท์

สร้างงานบริการ UserService

หากว่าบทที่แล้ว เรื่องการสร้างงานบริการ ยังดูไม่เข้าใจชัดเจน นี้ก็จะเป็นการทบทวนอีกครั้ง ในการสร้างงานบริการแต่จะเลือกใช้เป็น Observable<> ใช้ CLI สร้างงานบริการ UserService

```
ng generate service user
```

ให้คลาส UserService อ่านข้อมูลจาก ไฟล์ user-data.ts ซึ่งอยู่ในรูปค่าคงที่ของอาร์เรย์ของคลาส User จึงจำเป็นต้องนำเข้า User และ USERS และเพื่อรองรับงานบริการ Observable ก็ต้องนำเข้า Observable เพื่อทำงานแบบอซิงโครนัส รวมทั้ง of เพื่ออ่านค่าที่ได้มาแบบอซิงโครนัส และสร้างฟังก์ชัน getUserObservable( ) ให้คืนค่า อาร์เรย์หรือตัวให้สังเกตการณ์(Observable) ของ USERS ดังตัวอย่างต่อไปนี้

Code 10. src/app/user.service.ts

```
import { Injectable } from '@angular/core';
import { User } from './user';
import { USERS } from './user-data';
import { Observable, of } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  constructor() {   }
  getUserObservable(): Observable<User[]> {
       return of(USERS);
  }
}
```

รับงานบริกาด้วยการสมัครรับข้อมูล

ให้คอมโพเน้นท์ dashboard รับข้อมูลรายการ User ผ่าน UserService ก็ต้องนำเข้าสองส่วนนี้ด้วย

Code 11. src/app/dashboard.component.ts

```
import { User } from '../user';
import { UserService } from '../user.service';
```

หากย้อนไปดูตัวอย่างโปรแกรมก่อนหน้านี้ ตัวแปร users ของคลาส DashboardComponent มีชนิดข้อมูลเป็น User[ ] แต่ยังไม่มีค่าอะไร จึงต้องสร้างฟังก์ชันอ่านข้อมูล ให้กับตัวแปร users ใช้การรับสมัครข้อมูล และเรียกใช้ทันทีผ่านฟังก์ชัน ngOnInit( )

Code 12. src/app/dashboard/dashboard.component.ts

```
public users:User[];
constructor(private userService:UserService) { }
```

```
ngOnInit(): void {
        this.getUser();
}
getUser():void{
       this.userService
           .getUserObservable()
           .subscribe(users => this.users = users);
}
```

นำทางด้วย routerLink

รายการนำทาง ที่ไปยังหน้าต่าง ๆ หรือ คอมโพเน้นท์ต่าง ๆ ควรอยู่ส่วนบน คล้ายเป็นดัง เมนู หรือรายการนำทาง ให้เพิ่มรายการนำทางนี้ คือ ให้ไปยัง /users และ /dashboard ได้ ผ่านคุณสมบัติ routerLink

Code 13. src/app/app.component.html

```
<nav class="nav">
  <a class="nav-link active" routerLink="/dashboard">Dashboard</a>
  <a class="nav-link" routerLink="/users">Users</a>
</nav>
<router-outlet></ router-outlet>
```

ในตัวอย่างนี้มีคอมโพเน้นท์ Users เพื่อแสดงทุกรายการ ให้เป็นอีกเส้นทางหนึ่ง และทำคอมโพเน้นท์ user เพื่อแสดงรายการเพียง 1 รายก็ถึงเวลาต้องสร้างคอมโพเน้นท์นี้ไว้ด้วย

```
ng generate component users
ng generate component user
```

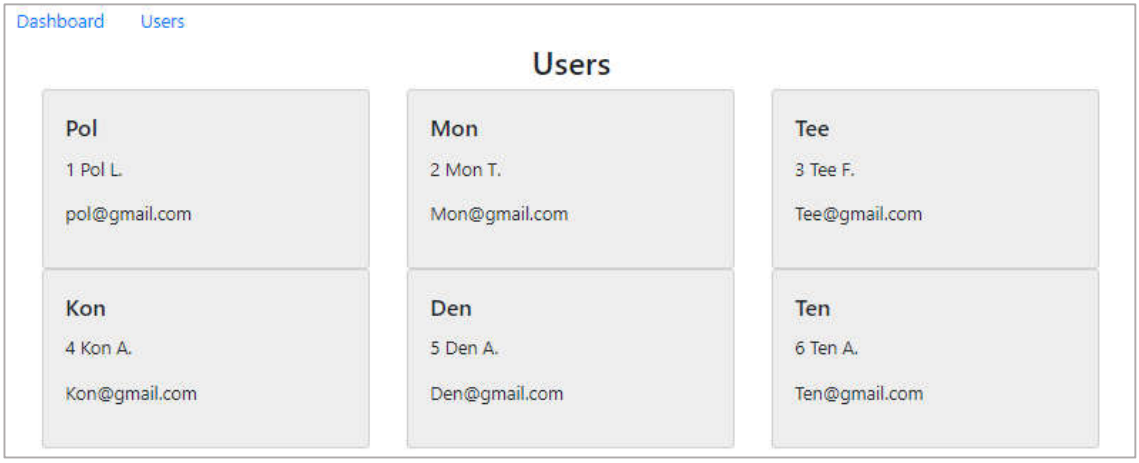
และต้องไม่ลืม กำหนดเส้นทางในโมดูล routing โดยเพิ่มสองส่วน คือส่วนนำเข้า และ อาร์เรย์เส้นทาง ตามตัวอย่างมีได้กำหนดไว้ 3 เส้นทางแล้ว

คอมโพเน้นท์ users ยังไม่ต้องกำหนดค่าอะไร แต่เตรียมเผื่อไว้สำหรับ แสดงข้อมูลที่เป็นทุกรายการของ Users หรือรายการรวม ที่แสดงคล้ายตาราง ส่วนคอมโพเน้นท์ user เตรียมไว้สำหรับแสดงรายละเอียดของแต่ละราย (user)

Code 14. src/app/app-routing.module.ts

```
import { DashboardComponent } from './dashboard/dashboard.component';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';
import { UsersComponent } from './users/users.component';

const routes: Routes = [
  { path:'dashboard', component: DashboardComponent},
  { path:'users', component: UsersComponent},
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
   { path: '**', component: PageNotFoundComponent },
];
```



รูป 2 หน้าแสดงผลคอมโพเน้นท์ dashboard

สร้างเส้นทางส่วนรายละเอียด User

คอมโพเน้นท์ UserComponent ที่เคยสร้างก่อนหน้านี้ยังว่างเปล่า ไม่มีข้อมูลอะไร แต่ตั้งใจที่ใช้แสดงรายละเอียดข้อมูลเมื่อ มีการเลือก user รายใดรายหนึ่งจากหน้าคอมโพเน้นท์ Dashboard

ในความสามารถของหน้าเว็บแล้ว สามารถที่จะทำให้ส่วนรายละเอียดแสดงใดหลายวิธี เช่น

1. คลิกรายการ user ที่ คอมโพเน้นท์ dashboard โดยส่งค่าผ่าน URL โดยตรง
2. คลิกรายการ user ที่ คอมโพเน้นท์ users โดยส่งค่าผ่าน ฟังก์ชัน ในคลาส UserComponent

ในวิธีการที่ 1 เส้นทางผ่าน URL ไปสู่ UserComponent คือการแสดงข้อมูลได้บน URL เช่น ระบุแบบเต็มใน URL เป็น localhost:4200/user/1

ในวิธีการที่ 2 เส้นทางกำหนดผ่านเหตุการณ์คลิก (click event) โดยมีฟังก์ชันรองรับเหตุการณ์นี้ และฟังก์ชันนี้เขียนส่งไปยัง URL เช่น `this.router.navigate(['/user/'+id]);`

เมื่อเส้นทาง user ให้เป็นเส้นทางที่ต้องระบุตัวแปร เข้าไปในอาร์เรย์ routers กรณีที่ไม่ใส่ตัวแปรให้ถือว่าใส่ตัวแปร เช่น มี id เป็น 1 และไม่ลืมนำเข้า คอมโพเน้นท์นี้ด้วย ตามตัวอย่างที่ให้มาดังต่อไปนี้

Code 15. src/app/app-routing.module.ts

```
{ path: 'user/:id', component: UserComponent },
```

Code 16. src/app/app-routing.module.ts

```
import { UserComponent } from './user/user.component';
```

การเพิ่มเส้นทางนี้จะพบว่า มีการใส่ตัวแปร id หลังเครื่องหมาย : เพื่อระบุรหัส User ที่ต้องการแสดงผลตามรหัส และ ณ ตอนนี้ ตัวแปรค่าคงที่ Routes มีเส้นทางต่าง ๆ ดังนี้

Code 17. src/app/app-routing.module.ts

```
const routes: Routes = [
  { path:'dashboard', component: DashboardComponent},
  { path:'users', component: UsersComponent},
  { path: 'user/:id', component: UserComponent },
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
```

```
  { path: '**', component: PageNotFoundComponent }
];
```

คอมโพเน้นท์ Dashboard ชี้ไปยัง UserComponent

Dahsboard ยังไม่สร้างลิงค์ไปยัง คอมโพเน้นท์ User จึงต้องสร้างแก้ไข ให้มีการเชื่อมไปยังคอมโพเน้นท์ User แบบระบุตามรหัส ในรูปแบบของ user/:id

Code 18. src/app/dashboard/dashboard.component.html

```
<a routerLink='/user/{{user.id}}' style="text-decoration:none">
 <div class="card-body" >
   <h5 class="card-title">{{user.fname}}</h5>
   <p class="card-text">{{user.id}} {{user.fname}} {{user.lname}}</p>
   <p class="card-link">{{user.email}}</p>
 </div>
</a>
```

จากตัวอย่างนี้จะเห็นว่า ได้เพิ่มอีลีเม้นท์ <a> คล่อม ไว้ ทำให้ส่วนที่ถูกคล่อมด้วย <a> (ส่วนอื่นยังเหมือเดิม) จะกลายเป็นจุดเชื่อมไปยัง url; user/:id โดยมีคำสั่งโดยตรงของ แองกูล่าร์ สร้างจุดเชื่อมคือ routerLink

ใส่ค่าให้กับ routerLink ทำได้สองแบบ วิธีแรกใส่เป็นค่าอักษรร่วมเป็นชุดเดียวตามตัวอย่างที่ผ่านมา อีกวิธีหนึ่งใส่เป็นค่าอาร์เรย์ วิธีนี้เขียนแทน วิธีก่อนหน้านี้ก็ได้

```
[routerLink]="['/user', user.id]"
```

ถึงตอนนี้ถ้าทดลองคลิก แต่ละ คาร์ด(card) ก็จะไปยัง คอมโพเน้นท์ user พร้อมกับเลข id (สังเกตที่ URL ของหน้าเว็บ) เพียงแต่หน้าคอมโพเน้นท์ User ยังไม่มีรายการใด ซึ่งจะได้สร้างกันต่อไป

คอมโพเน้นท์ Users ชี้ไปยัง UserComponent

คอมโพเน้นท์ Users ก็เป็นอีกคอมโพเน้นท์หนึ่งที่ยังเป็นว่าง เรามาเริ่มปรับปรุงข้อมูลคลาส UsersComponent จากตัวอย่างต่อไปนี้ (แบบเต็มคลาส) จะเห็นว่า คล้ายกับ คลาส DashboardComponent มาก

Code 19. src/app/users/users.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User } from '../user';
import { UserService } from '../user.service';

@Component({
  selector: 'app-users',
  templateUrl: './users.component.html',
  styleUrls: ['./users.component.css']
})
export class UsersComponent implements OnInit {
  public users:User[];
  constructor(private userService:UserService) { }

  ngOnInit(): void {
        this.getUser();
  }
```

```
  getUser():void{
       this.userService
         .getUserObservable()
         .subscribe(users => this.users = users);
  }
  onSelect(id:number):void{
  }
}
```

แต่ถ้าสังเกตดี ๆ จะพบว่า มีฟังก์ชัน onSelect( ) เพิ่มมา ฟังก์ชันนี้เตรียมไว้สำหรับให้เป็นเส้นทางไปยังอีกคอมโพเน้นท์อื่น ซึ่งนี้ก็เป็นอีกวิธีหนึ่งที่จะสร้างเส้นทางไปคอมโพเน้นท์ User ซึ่งสมมุติเป็นการสร้างฟังก์ชันคลิก ผ่านฟังก์ชัน onSelect( ) ดังตัวอย่างไฟล์ต่อไปนี้

Code 20. src/app/users/users.component.html

```
<table class="table">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Email</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor='let user of users'
       (click)="onSelect(user.id)" style="cursor:pointer" >
      <th scope="row">{{user.id}}</th>
      <td>{{user.fname}}</td>
      <td>{{user.lname}}</td>
      <td>{{user.email}}</td>
    </tr>
  </tbody>
</table>
```

จากตัวอย่างนี้ ใช้การวนซ้ำของ *ngFor เป็นการวนซ้ำทั้งแถว <tr> โดยกำหนด CSS ให้ตัวชี้เป็นรูปมือ (pointer) เพื่อให้ผู้ใช้งานรู้ได้ว่าแต่ละแถวสามารถคลิกได้ และทำการผูกกับฟังก์ชัน onSelect(user.id) โดยรับค่าตัวแปรเป็น user.id

ต่อมาสร้างฟังก์ชัน onSelect( ) ไว้ที่คลาส UserComponent รับตัวแปรเข้าเป็น id และให้ router สร้างเส้นทางไปยัง คอมโพเน้นท์ user

Code 21. src/app/users/users.component.ts

```
onSelect(id: number): void {
    this.router.navigate(['/user/'+id]);
}
```

การใช้ this.router เป็นตัวนำทางได้จะต้องนำเข้า Router ก่อน และกำหนดตัวแปร router แทน Router กับ constructor

Code 22. src/app/users/users.component.ts

```
import { Router } from '@angular/router';
```

Code 23. src/app/users/users.component.ts

```
constructor(
  private userService: UserService,
  private router: Router
){  }
```

#	First	Last	Email
1	Pol	L.	pol@gmail.com
2	Mon	T.	Mon@gmail.com
3	Tee	F.	Tee@gmail.com
4	Kon	A.	Kon@gmail.com
5	Den	A.	Den@gmail.com
6	Ten	A.	Ten@gmail.com

รูป 3 ผลแสดงของคอมโพเน้นท์ Users

การใช้เส้นทางผ่าน navigate( ) ถือเป็นระบุเส้นทางที่สัมพันธ์เส้นทางปัจจุบัน แต่ถ้าต้องการใช้กับเส้นทางที่ไม่ได้สัมพันธ์กับเส้นทางปัจจุบัน จะต้องใช้เส้นทางเต็มรูปแบบแต่ยังคงอยู่ในรูปแบบตามที่กำหนดเส้นทางใน Routes ซึ่งนำไปใช้ได้กับเมธอด navigateByUrl) ตัวอย่างใช้แบบระบุเต็มรูป URL เช่น การให้ไปยัง localhost:4200/user/1 ซึ่งการใช้แบบนี้แทนวิธีการก่อนหน้านี้ได้ผลเหมือนกัน

```
this.router.navigateByUrl('localhost:4200/user/1');
```

การแสดงผลของคอมโพเน้นท์ User

คอมโพเน้นท์ Dashboard และ Users สร้างลิงค์ไปยัง UserComponent ได้กันหมดแล้ว แต่การสร้างลิงค์ที่ผ่านมาเป็นสร้างแบบมีตัวแปร เช่น /user/1 ซึ่งหมายความว่าเลือกเฉพาะรายการที่มีรหัส 1 เท่านั้น

การทำให้ UserComponent แสดงผลตามรายการเฉพาะรหัสได้นั้น จะมีขั้นตอนดังนี้

- อ่านเส้นทางที่ได้มา
- แยกรหัส จากเส้นทาง
- สืบค้นรหัส ผ่านบริการ UserService

ด้วยการดำเนินการดังกล่าว จึงต้องเพิ่มการนำเข้า ActivateRoute, Location และ UserService และทำให้ส่วนนำเข้าเหล่านี้ เป็นสมาชิกแบบ private ของคอมโพเน้นท์ User

Code 24. src/app/user/user.component.ts

```
import { ActivatedRoute } from '@angular/router';
import { Location } from '@angular/common';
```

```
import { UserService }  from '../user.service';
import { User } from '../user';
```

Code 25. src/app/user/user.component.ts

```
constructor(
  private route: ActivatedRoute,
  private userService: UserService,
  private location: Location
) {}
```

สิ่งที่นำเข้า ActivatedRoute ใช้สำหรับอ่านค่า URL ซึ่งช่วยในการแยกค่าตัวแปร หรือรหัส (id) ออกจาก URL ได้ เพื่อนำ รหัส ไปสืบค้นต่อไปยัง UserService โดย UserService เปรียบเสมือนหน่วยบริการข้อมูลระยะไกล ซึ่งต่อไปจะใช้ HTTP (HTTP จะได้ศึกษาในบทต่อไป) เป็นบริการใน userService และหน่วยบริการสุดท้าย คือ location เป็นเหมือนตัวกำหนดเส้นทางผ่านเบราเซอร์ ให้เชื่อมต่อไปยังส่วนใดของคอมโพเน้นท์ แต่ตอนนี้ยังไม่ใช้งานบริการส่วนนี้

แยกรหัส ออกจาก URL

ให้ฟังก์ชัน ngOnInit( ) เรียกข้อมูล user ตามรหัส ทันทีเมื่อสร้างคอมโพเน้นท์นี้ จากฟังก์ชัน getUser( ) ซึ่งในฟังก์ชันนี้มีการเรียกใช้บริการ route.snapshot.paramMap.get('id') ซึ่งตัดแยกตัวแปร id ออกจาก URL โดย id ที่ใส่นี้ให้ถือเป็นชนิดข้อมูลอักษร ไม่ใช้ตัวเลข เพราะเป็นผลมาจากการอ่านค่า URL แต่ด้วยหน้าฟังก์ชันมีเครื่องหมาย + จึงทำหน้าที่แปลงชนิดข้อมูลให้กลายเป็นตัวเลข

ต่อมานำตัวเลขรหัสที่ตัดแยกได้ นำไปเก็บเป็นค่าคงที่ id ค่าคงที่นี้ จะไม่เปลี่ยนแปลงใด ๆ ตอนใช้งานออเบจ็กต์คอมโพเน้นท์นี้

นำค่า id ไปสืบค้นจากบริการ userService.getUser(id) และใช้การสืบค้นแบบอซินโครนัส จึงต้องลงทะเบียนฟังก์ชัน เรียกกลับ (call back function) กับ subscribe( ) ดังตัวอย่างต่อไปนี้

Code 26. src/app/user/user.component.ts

```
ngOnInit(): void {
  this.getUser();
}
user:User;
getUser(): void {
  const id = +this.route.snapshot.paramMap.get('id');
  this.userService
      .getUser(id)
      .subscribe(user => this.user = user);
}
```

ฟังก์ชัน getUserService ที่อยู่ในคอมโพเน้นท์ User ต้องใส่ตัวแปรประกอบด้วย จึงจะทำงานได้ ดังนั้นแล้ว งานบริการ UserService จะต้องสร้างฟังก์ชัน getUser(id) ต่อไป ที่รับค่าตัวแปรได้ด้วย

ฟังก์ชันของ UserService อ่านค่าแบบมีตัวแปร

จากเดิมงานบริการของ UserService มีฟังก์ชัน getUsers( ) ซึ่งจะอ่านมาได้ทุกรายการของ Users แต่เราต้องการฟังก์ชันที่อ่านเฉพาะรหัส ใดรหัสหนึ่งเท่านั้น จึงต้องสร้างฟังก์ชันเพิ่มขึ้นมาใหม่อีก หนึ่งฟังก์ชัน ชื่อว่า getUser(id) ให้สังเกตว่า ชื่อฟังก์ชันไม่เติม s ท้ายฟังก์ชัน เพราะให้ตรงกับไวยกรณ์ภาษาอังกฤษ สำหรับอ่านค่าเดี่ยว

Code 27. src/app/user.service.ts

```
getUser(id: number): Observable<User> {
  return of(USERS.find(user => user.id === id));
}
```

สำหรับฟังก์ชันนี้ of( ) ที่มาจากไลบรารี่ rxjs เพื่อการทำงานแบบอซินโครนัส ในการคืนค่าตามการค้นรหัส ให้สังเกตว่า ตัวแปรเข้าของ of( ) เป็น ฟังก์ชัน ซึ่งคืนค่าที่เป็นไปตามเงื่อนไข ของอาร์เรย์ find( ) โดยตัวแปรของ find( ) เป็นแบบ แลมบ์ดา (lambda) หรือฟังก์ชันไม่มีชื่อ ที่วนซ้ำ โดยมี user เป็นวิ่งในอาร์เรย์ USERS ทำการหาตัววิ่ง user ใด มีค่าตรงกับ id ที่เป็นตัวแปรที่ต้องการค้น ก็จะคืนค่า รายการนั้นออกไป (ฟังก์ชัน find นี้ย่อมาก น่าจำและนำไปใช้ประโยชน์ได้ดี)

ต่อไปให้ แสดงรายละเอียดข้อมูลผู้ใช้ ซึ่งแก้ไข ใน user.component.html ดังตัวอย่างนี้

Code 28. src/app/user/user.component.html

```
<div class="container">
  <div class="row row-cols-1">
       <div class="card" style="width: 18rem;"  >
         <div class="card-body">
              <h5 class="card-title">Card title</h5>
              <p class="card-text">ID:{{user.id}} :
                     Name:{{user.fname}} {{user.lname}}</p>
         </div>
       </div>
  </div>
</div>
```

ถึงตอนนี้ สามารถทดสอบเว็บไซต์ ได้แล้ว ทดลองคลิกไป ยังคอมโพเน้นท์ต่าง ๆ หรือแม้แต่ เขียนผ่าน URL เองโดยไม่ต้องคลิกผ่านคอมโพเน้นท์ เช่น localhost:4200/user/1 ใน URL ของเบราเซอร์

อ่านค่าโดยใช้ SwitchMap

ยังมีอีกวิธีหนึ่งที่ใช้อ่านค่าตัวแปรของ URL โดยที่ไม่ต้องผ่านการใช้ฟังก์ชัน subscript( ) ที่จะสะดวกในการค่าเดี่ยว หรือแสดงผลเพียงที่เดียว คือใช้ switchMap( ) ซึ่งตัวดำเนินการหนึ่งของ rxjs/operators เรามาทดลองใช้งานกัน เริ่มจากนำเข้า อินเทอร์เฟสนี้ ส่วนเกี่ยวข้องอื่นๆ

Code 29. src/app/user/user.component.ts

```
import { switchMap } from 'rxjs/operators';
import { ActivatedRoute, ParamMap } from '@angular/router';
import { Observable, of } from 'rxjs';
```

การนำเข้า Observable เพราะใช้รับค่าแทนการใช้ฟังก์ชัน subscribe( ) และ ParamMap เป็นไทป์ของตัวแปรที่อ่านค่าผ่าน URL จึงต้องสร้างตัวแปรประเภท Observable<User>

Code 30. src/app/user/user.component.ts

```
export class UserComponent implements OnInit {
  user:User;
  user$:Observable<User>;
```

ให้สังเกตว่า user: User ตัวเดิมยังเก็บไว้เพื่อทำงานเปรียบเทียบ และตัวแปรที่เพิ่มไปใหม่คือ user$ มีเครื่องหมาย $ ต่อท้ายเพื่อแยกความแตกต่างกับตัวแปรทั่วไป ต่อไปก็ใส่การผ่าน user$ ในฟังก์ชัน ngOnInit( )

Code 31. src/app/user/user.component.ts

```
ngOnInit(): void {
        this.getUser();
        this.user$ = this.route.paramMap.pipe(
            switchMap( (params:ParamMap)=>
               this.userService.getUser(+params.get('id'))
        ));
}
```

ให้สังเกตอีกว่า การใช้งานบริการ userService ในครั้งนี้ไม่มีการใช้ฟังก์ชัน subscribe( ) อีกแล้วแต่ใช้ทำงานคล้ายการฟังก์ชันนี้ในภายหลังบนหน้า HTML แทน

เราจะเพิ่มการอ่านบนหน้า HTML โดยใส่ async ต่อ | เพื่อการทำงานแบบอซิงโครไนส์ รูปแบบการใช้ควรตรวจสอบก่อนว่า มีค่า user$ อยู่หรือไม่ ถ้ามีก็แสดงผล id ดังตัวอย่างนี้ ใส่ต่อท้าย <h5> เพื่อการทดสอบเท่านั้น

Code 32. src/app/user/user.component.html

```
<h5 class="card-title">User</h5>{{ (user$ | async)?.id }}
```

วิธีการนี้ช่วงลดฟังชันการ subscribe( ) ซึ่งจะพบวิธีการใช้งานแบบนี้เพื่อความกระชับของการเขียนโปรแกรม

ทำให้ย้อนกลับได้

ตัวบริการ location ตอนนี้เราจะนำมาใช้งาน เพื่อเขียนระบุตำแหน่งที่ย้อนกลับ มีฟังก์ชันหนึ่งของบริการนี้คือ back( ) เว็บจะย้อนกลับไปยัง URL ที่ผ่านมา

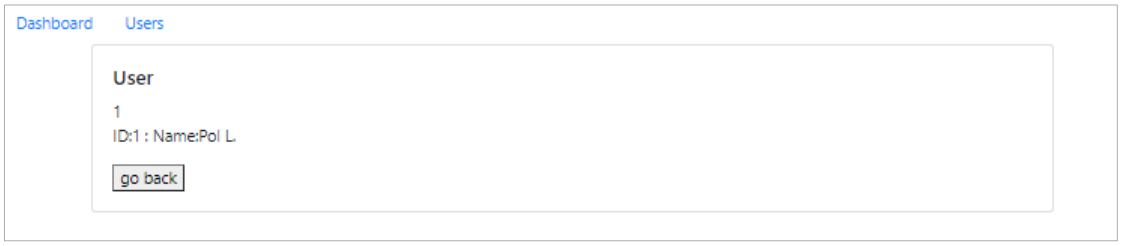
เราเริ่มต้นจากสร้างปุ่มให้คลิก เพื่อย้อนกลับได้ ภายในไฟล์ HTML ของคอมโพเน้นท์ User ให้ใช้อีเว้นท์คลิกเพื่อผูกกับฟังก์ชัน goBack( ) ของคอมโพเน้นท์ User ดังตัวอย่างต่อไปนี้

Code 33. src/app/user/user.component.html

```
<div class="card-body">
            <h5 class="card-title">Card title</h5>
            <p class="card-text">ID:{{user.id}} :
                  Name:{{user.fname}} {{user.lname}}</p>
            <button (click)="goBack()">go back</button>
</div>
```

Code 34. src/app/user/user.component.ts

```
goBack(): void {
  this.location.back();
}
```



รูป 4 ผลการแสดงของคอมโพเน้นท์ User/1

ทดสอบการทำงานอีกครั้ง ทดสอบการใช้ฟังก์ goBack( ) โดยคลิกปุ่มที่เพิ่งสร้าง เว็บนี้ก็ทำงานไปตามเส้นทางต่าง ๆ ทุกเส้นทาง ตามที่ต้องการได้ทั้งหมด

เส้นทางเชิงสัมพันธ์

เมื่ออยู่เส้นทางหนึ่งที่ เช่น localhost:4200/users ซึ่งแสดงหน้ารายการผู้ใช้ทั้งหมด และในหน้านี้มีการเชื่องโยงให้คลิกเลือกไปที่อื่นได้ เช่น ไปยังหน้ารายการใดรายการหนึ่ง เช่น ต้องการไป users/1 เมื่อคลิกจุดเชื่อมนั้น จะกลายเป็นรูปแบบ URL: localhost:4200/users/users/1 ซึ่งแสดง URL ที่สัมพันธ์กับ URL เดิม

ดังนั้นแล้วจะต้องแก้ให้ถอยออกไปหนึ่งขั้น ในการใช้ routerLink ใช้การใส่จุดสองจุด (..) เพิ่มหน้า URL จะทำให้ได้ URL: localhost:4200/users/1 ตามที่ต้องการ ดังตัวอย่างต่อไปนี้

Code 35. src/app/users/users.component.html

```
<td> <a routerLink="../user/{{user.id}}">{{user.fname}}</a></td>
```

แองกูล่าร์ สร้างต้นทาง URL เพื่อการอ้างอิงเชิงสัมพันธ์ให้ไว้แล้ว โดยการใส่ <base herf=”/”> ไว้ที่ส่วน <head> ของไฟล์ src/index.html ไว้อยู่แล้ว

เส้นทางในโมดูลย่อย

โมดูลหนึ่งๆ เป็นอิสระและมีเส้นทางของตนเอง (Routing) การนำโมดูลที่เป็นอิสระในใส่ในอีกโมดูลหนึ่ง เส้นทางของโมดูลย่อยก็ไม่มีผลกระทบต่อโมดูลหลัง

ตัวอย่างต่อไปนี้จะสร้างโมดูลหนึ่งที่มีเส้นทางของตนเอง สมมุติให้ชื่อโมดูลว่า Courses แล้วนำไปใส่ในโมดูลที่มีอยู่ทำไว้ก่อนแล้ว

```
ng generate module courses --routing
ng generate component courses/courses
```

ผลของการสร้างโมดูลนี้ จะได้ไฟล์โมดูล (courses.module.ts) และไฟล์เส้นทาง (courses-routing.moudle.ts) และได้คอมโพเน้นท์ Courses ฟ้อยู่ภายใต้โมดูล courses ในไฟล์โมดูล Courses (courses.module.ts) จะมีคอมโพเน้นท์ Component และ โมดูเส้นทางใส่ให้มาด้วยแล้ว

ต่อไปจะกำหนดเส้นทางของโมดูลนี้ให้เชื่อมโยงไปยังคอมโพเน้นท์ Courses ให้

Code 36. src/app/courses/courses-routing.module.ts

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

```
import { Routes, RouterModule } from '@angular/router';

import { CoursesComponent } from './courses/courses.component';
const routes: Routes = [
      {path: 'courses', component:CoursesComponent },
];

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

จากตัวอย่างนี้การกำหนดเส้นทางไม่ได้ต่างอะไรจากที่เคยได้ทำมาแล้ว แต่มีสิ่งน่าสังเกตอย่างหนึ่งคือ ภายใต้ @NgModule มีการใช้ forChild(routes) เพื่อบอกว่าเป็นเส้นทางย่อยภายใต้อีกโมดูลหนึ่ง

เมื่อนำไปใช้งาน จะต้องในเข้าโมดูลหลัก ทั้งในส่วนหัว คือนำเข้าจากไฟล์อะไร และนำเข้าในส่วน @NgModule

Code 37. src/app/app.module.ts

```
import { CoursesModule } from './courses/courses.module';
//ส่วนอื่นๆ
@NgModule({
  imports: [
    BrowserModule,
    CoursesModule,
    AppRoutingModule,

 ],
//ส่วนอื่นๆ
```

สิ่งพิเศษในการนำเข้าส่วน @NgModule จะต้องให้ลำดับ CoursesModule อยู่บน AppRoutingModule ซึ่งเป็นโมดูลเส้นทางหลัก เพราะการอ่านเส้นทางจะอ่านจากบนลงล่าง หากจำกันได้เส้นทางสุดท้ายที่กำหนดในโมดูลหลัก คือหน้าที่ไม่มีเส้นทาง (PageNotFound) หากให้หน้าไม่พบเส้นทางอยู่ก่อน CoursesModule จะทำให้ไม่สามารถไปยังเส้นทางสุดท้ายของ AppRoutingModule ได้

การทดสอบให้ใส่เส้นทางไปยังเมนูหน้าหลัก เพิ่มเส้นทาง “/courses” ซึ่งเป็นเส้นทางในโมดูลย่อย และเป็นเส้นทางระดับเดียวกับเส้นโมดูลหลัก

Code 38. src/app/app.component.html

```
<nav class="nav">
  <a class="nav-link active" routerLink="/dashboard">Dashboard</a>
  <a class="nav-link" routerLink="/users">Users</a>
  <a class="nav-link" routerLink="/courses">Courses</a>
</nav>
<router-outlet></ router-outlet>
```

เส้นทางย่อยในโมดูล

โมดูลย่อยที่มีเส้นทางของตนเอง แต่ต้องการทำให้เป็นเส้นทางย่อยในโมดูลย่อย เช่น มีเส้นทาง courses/manage เราจะเริ่มจากสร้างคอมโพเน้นท์เพิ่มอีกตัวหนึ่งในโมดูล Courses

```
ng generate component courses/manage
```

หลังจากนั้นสร้างเส้นทางย่อย ในเส้นทางหลักของโมดูล Courses ด้วยการเพิ่ม การนำเข้าคอมโพเน้นท์ ManageComponent และเส้นทางไปยังคอมโพเน้นท์นี้

Code 39. src/app/courses/courses-routing.module.ts

```
import { ManageComponent } from './manage/manage.component';
const routes: Routes = [{
       path: 'courses',
       component:CoursesComponent,
      children:[{
             path:'',
             children:[
                    {path:'manage', component:ManageComponent},
       ]}]
}];
```

เส้นทางย่อยของเส้นทางย่อยจะใช้ children โดยไม่มี path หลัง แต่ใช้ path ใน children อีกชั้นหนึ่ง ซึ่งใช้ชื่อเส้นทางว่า manage

นอกจากนี้ต้องเพิ่มการไปยังเส้นทางย่อยในไฟล์หลักของโมดูลย่อย ในที่นี้ให้คอมโพเน้นท์ Courses เป็นกลุ่มไฟล์หลัก

Code 40. src/app/courses/courses.component.html

```
<p>courses works!</p>
<a class="nav-link" routerLink="./manage" >Manage Courses</a>
<router-outlet></router-outlet>
```

เมื่อทดสอบคลิกไปที่ ./mange สังเกตว่ามีจุด (.) นำหน้าเพื่อแสดงเป็นเส้นทางย่อยต่อจากเส้นทางปัจจุบัน ซึ่งเป็นเส้นทาง URL:http://localhost:4200/courses/manage ก็จะได้ผลเป็นหน้าเว็บของคอมโพเน้นท์ ManageComponent ซึ่งแสดงในส่วน <router-outlet>

ป้องกันเข้าเส้นทาง

ในบางเส้นทางต้องการให้ยืนยันตัวตนก่อนที่จะโหลดเส้นทางนั้นได้ ในแองกูล่าร์ใช้วิธีการป้องกันนี้ว่า การป้องกันเส้นทาง (Route guards) การสร้างทางให้มีความยืดหยุ่นนี้มีประโยชน์มาก ในการใช้ป้องกันการเปิดในบางเส้นทาง หรือป้องกันการปิดในบางเส้นทาง หรือโหลดข้อมูลก่อนเปิดบางเส้นทาง

เรามาเริ่มจากสร้างโมดูล auth ภายในโมดูลนี้เก็บเรื่องการรักษาความปลอดภัย ซึ่งประกอบด้วย guard, component, และ service ดังสร้างผ่าน CLI ดังนี้

```
ng generate module auth --routing
```

```
ng generate guard auth/auth --implements=CanActivate
ng generate component auth/login
ng generate service auth/auth
```

ไฟล์ auth.module มีการนำเข้าส่วนต่างๆ ที่จำเป็นต้องใช้ในโมดูลดังแสดงดังต่อไปนี้

Code 41. src/app/auth/auth.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { LoginComponent } from './login/login.component';
import { AuthRoutingModule } from './auth-routing.module';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [LoginComponent],
  imports: [
    CommonModule,
       AuthRoutingModule,
       FormsModule
  ]
})
export class AuthModule { }
```

ไฟล์ auth.guards.ts เป็นป้องกันโดยเขียนขยายมาจาก CanActivate เพื่อตรวจสอบว่า เส้นทางที่จะไปสามารถไปได้หรือไม่

Code 42. src/app/auth/auth.guards.ts

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

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(
       private authService: AuthService,
       private router: Router)
  {}
  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): true|UrlTree {
       console.log("AuthGuard@CanActivate running");
       let url:string = state.url
       return this.checkLogin(url);
  }
  checkLogin(url:string):true | UrlTree{
         if(this.authService.isLoggedIn){return true;}
         this.authService.redirectUrl = url;
         return this.router.parseUrl('/login');
```

```
  }
}
```

ด้วยการเขียนฟังก์ชัน canActivate เสียใหม่นี้ หน้าที่หลักของฟังก์ชันนี้จะตรวจสอบสิทธิ์การเข้าถึงเส้นทางที่กำหนดให้มีการป้องกัน จาก state.url เป็นเส้นทางปัจจุบัน (url ที่ต้องการตรวจสอบ (guard)) การตรวจสอบตรวจสอบกับฟังก์ชัน checkLogin( ) ซึ่งทำงานร่วมกับ authService หากตรวจพบว่ายังไม่ได้ ล็อกอินหรือยังไม่ได้สิทธิ์ ก็จะให้ไปยังฟอร์มล็อกอินใน url: /login แต่ถ้าได้รับสิทธิ์เข้าระบบแล้ว ก็ไปยังหน้าที่ต้องการได้ หรือหน้าต่อไป (next)

สำหรับเส้นทางในโมดูล auth มีเส้นทางเป็นของตนเอง กำหนดไว้ที่ไฟล์ auth-routhing.module.ts ดังเขียนกำหนดให้มีเส้นทาง login ที่แสดงผลคอมโพเน้นท์ LoginComponent ให้สังเกตว่า เส้นทาง /login การกำหนดเส้นทางไม่ทำเป็นเส้นทางย่อย ในการแสดงที่ URL เป็น http://localhost:4200/login ซึ่งต่างกับเส้นทางภายในโมดูล courses ที่กำหนดเป็นเส้นย่อยภายในโมดูลของตนเอง

Code 43. src/app/auth/auth-routing.moudle.ts

```
import { NgModule }             from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent }       from './login/login.component';

const authRoutes: Routes = [
  { path: 'login', component: LoginComponent }
];

@NgModule({
  imports: [
    RouterModule.forChild(authRoutes)
  ],
  exports: [
    RouterModule
  ]
})
export class AuthRoutingModule {}
```

ไฟล์ auth.service.ts รับหน้าที่ให้บริการตรวจสอบสิทธิ์ ซึ่งในที่ต้องการตรวจสอบสิทธิ์เฉพาะผู้ที่มีอีเมล a@a.com เท่านั้นที่เข้าระบบเส้นทางจำกัดสิทธิ์ได้ ฟังก์ชันสำคัญคือ login(email:string) โดยฟังก์ชันนี้รับตัวแปรเข้าเป็นอีเมลที่ต้องการตรวจสอบสิทธิ์ ฟังก์ชันทำงานแบบอซิงโครนัส (Obserable<Boolean>) คืนค่าจริงหรือเท็จ จากผลการตรวจอีเมลว่าตรงกับที่ตั้งไว้หรือไม่ นอกจากนี้ยังเพิ่มการหน่วงเวลา (delaly) และกำหนดค่าผลการเข้าระบบ (isLoggedIn)

Code 44. src/app/auth/auth.service.ts

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';
import { tap, delay } from 'rxjs/operators';

@Injectable({
  providedIn: 'root',
})
export class AuthService {
  email:string='a@a.com'; //ตั้งรหัสอีเมล์ที่ใช้ตรวจสอบ
  isLoggedIn = false;
```

```
  redirectUrl: string;

  login(email:string): Observable<boolean> {
    return of((email==this.email)?true:false)
       .pipe(
        delay(1000),
       tap(val => this.isLoggedIn = val)
    );
  }

  logout(): void {
    this.isLoggedIn = false;
  }
}
```

ไฟล์ login.component.ts มีฟังก์ชัน login(email:string) เป็นฟังก์ชันตรวจสอบเข้าระบบที่ส่งต่อไปยังงานบริการ authService ด้วยสมัครใช้บริการ ตัวงานบริการจะส่งผลการเข้าระบบได้ (true) หรือไม่ได้ (false) ผ่านตัวแปร isLoggedIn ถ้าได้ผลเป็น true ก็สามารถไปยังเส้นทาง /courses ได้ (CanActivate) แต่ถ้าได้ผลเป็น false จะคงอยู่หน้าเดิม และมีข้อความแจ้งความผิดพลาดในการเข้าระบบ

Code 45. src/app/auth/login/login.component.ts

```
import { AuthService } from '../auth.service';
import { NgForm } from '@angular/forms';
import { Router } from '@angular/router';
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  message: string;
  email:string;
  constructor(public authService: AuthService, public router: Router) {
  }

  login(email:string) {
    this.message = 'Trying to log in ...';
    this.authService.login(email).subscribe(() => {
      if (this.authService.isLoggedIn) {
             const redirectUrl = '/courses';
             this.router.navigate([redirectUrl]);
      }
       else{
             this.message="อีเมล์ไม่ถูกต้อง";
       }
    });
  }
  logout() {
    this.authService.logout();
  }
}
```

ไฟล์ login.component.html แสดงในรูปฟอร์มที่อยู่ภายในโมดูล auth ทำหน้าที่ส่งข้อมูลอีเมล เพื่อตรวจสอบความถูกต้อง

Code 46. src/app/auth/login/login.component.html

```
<div style="width:70%; margin:2px auto">
<h3 style="text-align:center">LOGIN</h3>
<form (ngSubmit)="login(email)">
  <div class="form-group">
    <label >Email address</label>
    <input type="email" class="form-control"
       [(ngModel)]='email' name='email'>
  </div>
  <button type="submit" class="btn btn-primary">Login</button>
</form>
<p>{{message}}</p>
</div>
```

ไฟล์ courses-routing.modules.ts ได้กำหนดการป้องกัน ด้วย canActive:[AuthGuard] การตั้งการป้องกันทำที่ระดับเส้นทางหลักของโมดูล courses ทำให้เส้นทางย่อย (children) ถูกป้องกันไปด้วย

Code 47. src/app/course/course-routing.modules.ts

```
import { AuthGuard } from '../auth/auth.guard';
const routes: Routes = [{
       path: 'courses',
       component:CoursesComponent,
       canActivate:[AuthGuard],
       children:[{
              path:'',
              children:[
                     {path:'manage', component:ManageComponent},
       ]}]
}];
```

ในส่วนโมดูลของ CoursesModule ก็ต้องนำเข้า AuthModule ด้วย

Code 48. src/app/courses/courses.comdule.ts

```
import { AuthModule } from '../auth/auth.module';

@NgModule({
  declarations: [
       CoursesComponent,
       ManageComponent
 ],
  imports: [
    CommonModule,
    CoursesRoutingModule,
    AuthModule
  ]
})
```

ป้องกันเข้าเส้นทางย่อย

ในบางป้องกันเส้นทางย่อย อาจไม่จำเป็นถ้าเส้นทางหลักได้ป้องกันไว้แล้ว แต่ถ้าเส้นทางหลักไม่ต้องการป้องกัน เราจะป้องกันเส้นทางย่อยได้ด้วย canActivateChild

การป้องกันเส้นทางย่อยต้องนำเข้า CanActivateChild และเขียนขยายฟังก์ชัน canActivateChild( ) โดยการเรียกใช้งานซ้ำฟังก์ชัน canActive( ) ได้เลย และใส่ตัวแปรเป็น route, state

Code 49. src/app/auth/auth.guards.ts

```
import {     CanActivate,
             ActivatedRouteSnapshot,
             RouterStateSnapshot,
             UrlTree,
             CanActivateChild,
} from '@angular/router';
```

Code 50. src/app/auth/auth.guards.ts

```
canActivateChild(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): true|UrlTree {
    return this.canActivate(route, state);
}
```

สำหรับเส้นทางย่อยที่ต้องการให้ใส่ป้องกันเส้นทาง canActivateChild บนเส้นทางย่อย เท่านี้ก็จะป้องกันเส้นทางย่อยไว้ได้ และถ้าไม่ต้องการป้องกันเส้นทางหลัก ก็ไม่ต้องใส่ canActivate ในเส้นทางหลัก

Code 51. src/app/course/course-routing.modules.ts

```
import { AuthGuard } from '../auth/auth.guard';
const routes: Routes = [{
       path: 'courses',
       component:CoursesComponent,
       //canActivate:[AuthGuard],
      children:[{
             path:'',
             canActivateChild: [AuthGuard],
             children:[
                    {path:'manage', component:ManageComponent},
       ]}]
}]
```

ป้องกันออกจากเส้นทาง

ในบางครั้งผู้ใช้อาจไม่ตั้งใจออกจากเส้นทางปัจจุบัน หรือออกไปเส้นทางอื่นโดยยังไม่ได้บันทึกข้อมูลบางอย่าง การป้องนี้ป้องกันได้ด้วย canDeactivate

```
ng g guard can-deactivate --implements=CanDeactivate
```

เมื่อสร้าง can-deactive ได้แล้ว ต่อไปให้นำเข้าคอมโพเน้นท์ที่ต้องการป้องกันการออกเส้นทาง ในที่นี้เลือกคอมโพเน้นท์ ManageComponent สิ่งต้องนี้เขียนขยายเพิ่มเติมคือฟังก์ชัน canDeactivate( ) ฟังก์ชันนี้จะคืนจริงหรือเท็จในรูปแบบอซิงโครนัส หรือใช้ ตัวให้สังเกตการณ์ได้ (Obserable<Boolean>) ในการคืนค่าใช้หน้าต่างของตอบ-กลับ เพื่อยืนยันการ

ยอมรับการออกจากหน้า ซึ่งจะมีให้ตอบ Yes หรือ No กรณีตอบ Yes เท่ากับยอมรับการไปยังเส้นทางอื่น แต่ถ้าตอบ No ก็จะคงอยู่ที่ URL เดิม อย่างไรก็ตามการป้องกันการออกจากเส้นทางนี้จะไม่สามารถป้องกันได้เมื่อผู้ใช้ ใช้การพิมพ์ URL ด้วยตนเอง

Code 52. src/app/can-deactivate.guard.ts

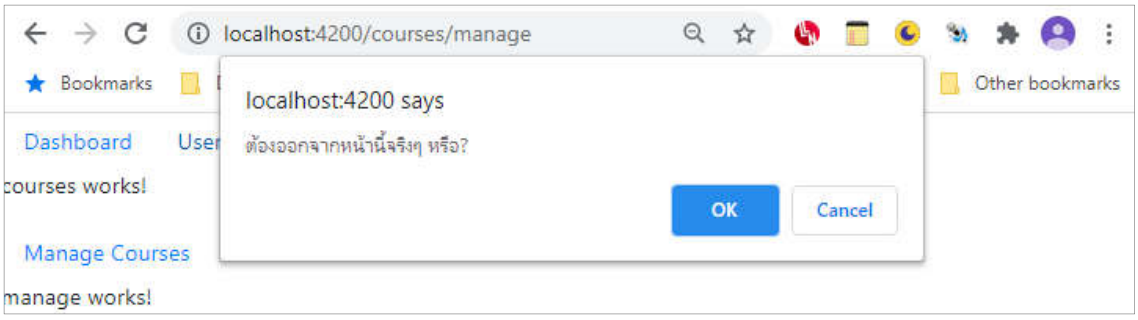
```
import { Injectable } from '@angular/core';
import { CanDeactivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree }
from '@angular/router';
import { Observable } from 'rxjs';
import { ManageComponent} from './courses/manage/manage.component';

@Injectable({
  providedIn: 'root'
})
export class CanDeactivateGuard implements CanDeactivate<ManageComponent> {
  canDeactivate(
       component: ManageComponent,
       route: ActivatedRouteSnapshot,
       state: RouterStateSnapshot
  ):Observable<boolean>|boolean {
    console.log(state.url);
       return window.confirm("ต้องออกจากหน้านี้จริงๆ หรือ?");
  }
}
```

ต่อไปต้องไปกำหนดเส้นทางของ ManageComponent ที่จะป้องกัน โดยการใส่เพิ่ม canDeactivate

Code 53. src/app/course/course-routing.modules.ts

```
import { CanDeactivateGuard } from '../can-deactivate.guard';
const routes: Routes = [{
       path: 'courses',
       component:CoursesComponent,
       //canActivate:[AuthGuard],
      children:[{
             path:'',
             canActivateChild: [AuthGuard],
             canDeactivate:[CanDeactivateGuard],
             children:[
                   {path:'manage', component:ManageComponent},
       ]}]
}];
```



รูป 5 ผลการป้องกันออกจากเส้นทาง ManageComponent

โหลดข้อมูลก่อนเปิดเส้นทาง

เมื่อคลิกเลือกเส้นทางใดเส้นทางหนึ่ง ข้อมูลจะถูกโหลดตามหลังจากคลิกเลือก แต่ข้อมูลที่เลือกคลิกไปนั้นอาจต้องการตรวจสอบก่อนว่ามีอยู่หรือไม่ จะเป็นการดีถ้าได้มีการตรวจสอบข้อมูลก่อนที่จะแสดงผล หรือบางทีข้อมูลมาช้ากว่าการแสดงผลที่พร้อมแสดงแล้ว ข้อมูลจึงยังไม่พร้อมที่จะแสดง แต่อาจแสดงความก้าวหน้ากำลังรอข้อมูลให้ผู้ใช้รู้ว่ากำลังโหลดข้อมูล ปัญหานี้แก้ได้ด้วยการใช้ Resolve

สมมุติว่าผู้ใช้คลิกเลือกรายการ user/2 ซึ่งควรจัดการอะไรบางอย่างได้ก่อนที่จะโหลดข้อมูล ก่อนที่หน้าคอมโนเน้นท์ User จะเปิดแองกูล่าร์ใช้ Resolve เพื่อการนี้ ก่อนอื่นใช้ CLI สร้างงานบริการ Resolver

```
ng generate service user-resolver
```

งานบริการนี้ตั้งใจให้โหลดข้อมูล ผู้ใช้ตามรหัสมาล่วงหน้า ใช้อินเฟส Resolve<T> ซึ่งเป็นอินเทอร์เฟส มีฟังกชัน resolve( )

```
interface Resolve<T> {
 resolve(route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot): Observable<T> | Promise<T> | T
}
```

ดังมีการเขียนขยายฟังก์ชัน resolve เพื่อคืนค่าให้สังเกตการณ์ (Obserale<T>) ในที่ค้นค่าของ User

Code 54. src/app/user-resolver.service.ts

```
import { Injectable } from '@angular/core';
import { UserService }  from './user.service';
import { User } from './user';
import { Observable, of } from 'rxjs';
import { Resolve, ActivatedRouteSnapshot, RouterStateSnapshot }
from '@angular/router';

@Injectable({
  providedIn: 'root'
})
export class UserResolverService implements Resolve<User>{
  constructor(private userService:UserService) { }

  resolve(
       route: ActivatedRouteSnapshot,
       state:RouterStateSnapshot): User | Observable<User> | Promise<User>{
       console.log("Resolve User is loading..");
       const userId = +route.paramMap.get('id');
       return this.userService.getUser(userId);
  }
}
```

ดังนั้นเส้นทาง user/:id จึงเป็นเส้นทางที่ให้โหลดข้อมูลก่อนเปิดแสดงผลของคอมโพเน้นท ดังให้ปรับปรุงเส้นทางนี้ในเส้นทางในโมดูลหลัก

Code 55. src/app/app-routing.module.ts

```
import {UserResolverService} from './user-resolver.service';

const routes: Routes = [
  { path:'dashboard', component: DashboardComponent},
  { path:'users', component: UsersComponent  },
  { path: 'user/:id', component: UserComponent,
          resolve:{user:UserResolverService}},
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];
```

สุดท้ายปรับปรุงคอมโพเน้นท์ User ให้อ่านข้อมูลขณะเริ่มทำงานในคลาสนี้

Code 56. src/app/user/user.component.ts

```
ngOnInit(): void {
      this.route.data.subscribe(
          (data:{user:User})=>{this.user=data.user;}
      );
}
```

สรุป

ในบทนี้ ได้ทดลองสร้างเส้นทางการเดินทางของลิงค์ (link) ต่างๆ คือ การเดินทางไปโดยตรงที่คอมโพเน้นท์ใดคอมโพเน้นท์ และการเดินทางโดยส่งค่าตัวแปรผ่านลิงค์ ขั้นตอนแรก คือต้องสร้าง กับ RouterModue โดยกำหนดเป็น ค่าคงที่เป็นอาร์เรย์ของ Routes ที่มีเส้นทาง (path) และคอนโพเน้นท์ เช่น

```
const routes: Routes = [
  { path:'users', component: UsersComponent },
  { path:'', redirectTo: '/dashboard', pathMatch: 'full' },
  { path:'dashboard', component:DashboardComponent }
];
```

สำหรับการกำหนดเส้นทางโดยมีตัวแปรเพิ่มเติมไปในเส้นทางนั้น จะต้องนำเข้าคลาส ActivateRoute และให้สร้างคอมโพเน้นท์ที่รับตัวแปรที่เพิ่มเติมนี้ ด้วยฟังก์ชัน snapshot.paramMap.get( ) เช่น

```
const id = +this.route.snapshot.paramMap.get('id');
```

นอกจากนี้ นำเข้าคลาส Router ใช้คลาสนี้ กำหนดเส้นทางผ่าน navigate( )

```
onSelect(id: number): void {
      this.router.navigate(['/user/'+id]);
}
```

เพื่อการรองรับเส้นทางย้อนกลับได้ จะต้องนำเข้า คลาส Location และเรียกใช้ฟังก์ชัน back( ) เพื่อย้อนกลับไปหน้าก่อนหน้านี้ได้ ผ่านฟังก์ชันที่สร้างเพื่อรองรับอีเว้นท์การคลิก เช่น

```
goBack(): void {
      this.location.back();
```

```
}
```

และการสร้างโมดูลย่อยที่กำหนดให้มีเส้นทางของตนเอง ก่อนอื่นต้องนำเข้าโมดูลย่อยในโมดูลหลักก่อน การนำเข้าต้องทำเข้าในลำดับก่อน โมดูลเส้นทางของโมดูลหลัก การสร้างโมดูลย่อยแบบมีเส้นทาง โปรแกรมจะกำหนดมาให้ว่า ใช้การนำเข้า RouterModule.forChild( ) ส่วนเส้นทางย่อยจะต้องกำหนดขึ้นเอง การกำหนดเส้นทางจะทำผ่าน อาร์เรย์ของ children ของคอมโพเน้นท์ของโมดูลย่อย โดยมี path เป็นค่าว่าง

```
const routes: Routes = [{
      path: 'courses',
      component:CoursesComponent,
      children:[{
            path:'',
            children:[
                  {path:'manage', component:ManageComponent},
      ]}]
}];
```

การป้องกันเส้นทางด้วยการสร้าง gard ซึ่งเป็นคลาสที่ต้องสร้างขยายมาจากอินเทอร์เฟส CanActivate และทำการเขียนฟังก์ชัน canActivate เพื่อตรวจสอบสิทธิ์การเข้าถูกเส้นทาง จากตัวอย่าง ใช้การตรวจสอบผ่านหน้า Login กลไกสำคัญคือต้องให้คืนค่าจริงเพื่อให้ผ่านเส้นทางได้ หรือคืนค่า UrlTree เพื่อไปยังเส้นทางตรวจสอบสิทธิ์ก่อน

การใช้งานโมดูลในโมดูล จากตัวอย่างในบนทนี้ มีโมดูลย่อยที่มีเส้นทางของตนเอง โมดูลใดต้องต้องใช้โมดูลย่อยก็ต้องนำเข้าโมดูลย่อย ในตัวอย่างนี้มีโมดูล app ไม่ได้ใช้งานโมดูล auth จึงไม่มีการนำเข้าภายในโมดูล app แต่โมดูล courses มีการใช้โมดูล auth จึงมีการนำเข้าโมดูล auth จะเห็นว่าโมดูลแต่ละโมดูลมีความอิสระ สามารถทำสำเนาทั้งโมดูลไปใช้งานกับโครงการเว็บอื่น ๆ ได้โดยอิสระ

คำถามทบทวน

1. เราจะสร้างโมดูลเส้นทางด้วยคำสั่งใด ใน CLI
2. ในอาร์เรย์ของเส้นทาง Routes[ ] หน้าที่ค้นไม่พบควรอยู่ลำดับใดในอาร์เรย์
3. คำสั่ง routerLink ใช้นำทางที่ไฟล์ประเภทใด ระหว่างไฟล์ TS กับ HTML
4. อิลีเม้นท์ (<>) ที่ใช้กำกับในไฟล์ HTML เพื่อแสดงคอมโพเน้นท์ตามเส้นทางที่เลือก มีชื่ออิลีเม้นท์ว่าอะไร
5. ถ้าต้องการไปยังเส้นทาง user และมีตัวแปรเป็น 1 ซึ่งเป็นเลือกเฉพาะรายที่มี id เท่ากับ 1 จะเขียนกำกับใน routerLink ได้อย่างไร
6. จากคำสั่ง

   ```
   const id = +this.route.snapshot.paramMap.get('id');
   ```

   เครื่องหมาย + หมายถึงอะไร และคำสั่งนี้ใช้ทำอะไร
7. การสร้างให้ย้อนกลับได้ไปหน้าเว็บก่อนหน้านี้ใช้คำสั่งอะไร
8. การสร้างเส้นทางย่อย ใช้คีย์เวิร์ดว่าอะไร ในโมดูลเส้นทางย่อย
9. เส้นทางที่ต้องการตรวจสอบการเข้าถึง canActive ในโมดูลเส้นทางย่อยมีค่าเป็นอะไรจึงป้องการเข้าเส้นทางได้
10. การป้องกันออกจากเส้นทาง ต้องเขียนกำกับด้วยฟังก์ชันใด

แบบฝึกหัด

1. จากหน้าเข้าระบบที่มีในตัวอย่าง เมื่อมีการเข้าระบบแล้วปุ่ม Logout จะแสดงที่เมนูหลัก
2. ให้แก้ไข ให้มีการตรวจสอบสิทธิ์การใช้งานก่อน ตั้งแต่แรกเปิดหน้าเว็บ โดยนำโมดูล **auth** ไปใช้งาน