

## รีเอ็กทีฟฟอร์มของ Angular

ฟอร์มของ Angular มีฟอร์มสองประเภทคือ รีเอ็กทีฟฟอร์ม (Reactive form) หรือ ฟอร์มที่มาจากแบบจำลองข้อมูล (Model Driven) ซึ่งอยู่ในไฟล์ TS กับ เทมเพลตฟอร์ม (Template form) หรือที่มีมาจากสร้างเทมเพลตในหน้า HTML

ในบทนี้ จะเน้นการทำงานกับ รีเอ็กทีฟฟอร์ม ตั้งแต่การสร้างฟอร์ม การผูกข้อมูล การตรวจสอบความถูกต้องของการกรอกข้อมูลในฟอร์ม ดังจะได้ศึกษาในหัวข้อต่อไป

- การสร้างอินพุต (input) ในแบบของรีเอ็กทีฟฟอร์ม และการสร้างอีเว้นท์ให้ฟอร์ม
- การสร้างฟอร์มกลุ่ม และการสร้างฟอร์มกลุ่มในฟอร์มกลุ่ม
- การสร้างฟอร์มแบบไดนามิก
- การตรวจสอบความผิดพลาดของอินพุต

---

### ความสำคัญของรีเอ็กทีฟฟอร์ม

รีเอ็กทีฟฟอร์มเป็นคุณสมบัติใหม่ที่ต่างจาก AngularJS ที่ใช้รูปแบบแสดงฟอร์มในรูปแบบเทมเพลตฟอร์ม คุณสมบัติใหม่นี้ เน้นความสำคัญของการสร้างโมเดล (Model) หรือแบบจำลองมาจากข้อมูล เช่น เมื่อโมเดลทำงานกับข้อมูลผู้ใช้ แบบจำลองจะเรียกว่า แบบจำลองผู้ใช้ หรือมีโมเดลเป็นผู้ใช้ ลักษณะการใช้งานแบบนี้จึงเหมาะสมการสร้างโปรแกรมใช้งานที่เน้นโมเดล หรือในทางพัฒนาโปรแกรมอาจเรียกในชื่อ ดาต้าโมเดล (Data Model) หรือโดเมนโลจิก (Domain Logic) ที่ได้จากการวิเคราะห์ระบบในชั้นงานธุรกรรม (Business logic layer)

รีเอ็กทีฟฟอร์ม จึงควบคุมการทำงานของฟอร์มกับโมเดลโดยตรง หรือข้อมูลในฟอร์มเชื่อมกับโมเดลโดยตรงและทันที ซึ่งหมายความว่า ข้อมูลที่เปลี่ยนแปลงในโมเดลจะมีผลโดยทันทีกับฟอร์ม กลไกการควบคุมโดยตรงนี้ ทำงานที่คลาสซึ่งเขียนในไฟล์ประเภท TS เป็นหลัก โดยคลาส FormControl ภายในโมดูล ReactiveFormsModule จุดนี้เป็นจุดแตกต่างกับ เทมเพลตฟอร์ม ที่โมเดลถูกควบคุมผ่าน NgModel เป็นหลัก จากไฟล์ HTML

### เริ่มสร้างรีเอ็กทีฟฟอร์ม

เราจะมาสมมติเรื่องกัน คือว่าเราต้องการสร้างหน้าเว็บที่มีฟอร์ม ให้มี <input> ของชื่อและนามสกุล ดังมีลำดับการทำงานดังนี้

1. การใช้ฟอร์มแบบรีเอ็กทีฟจำเป็นต้องใช้โมดูลชื่อ ReactiveFormsModule ดังนั้นจึงต้องนำเข้ามาโมดูลนี้ ดังเพื่อในส่วน import และเพิ่มส่วนอาร์เรย์ ในไฟล์ src/app/app.module.ts ดังตัวอย่างนี้

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    AppComponent
  ],
```

```

        imports: [
            BrowserModule,
            ReactiveFormsModule
        ],
        providers: [],
        bootstrap: [AppComponent]
    })
    export class AppModule { }

```

2. ต่อมาให้สร้างคอมโพเนนต์ที่ขึ้นมาใหม่ ผ่าน CLI โดยให้ CLI อยู่ ณ ไดเรกทอรีแอปพลิเคชันที่เราสร้าง ให้ชื่อคอมโพเนนต์ หรือฟอร์ม ว่า `NewUser`

```
ng generate component NewUser
```

3. เมื่อสร้างคอมโพเนนต์นั้นแล้ว ตรวจสอบอีกครั้งว่า ในการนำเข้าคอมโพเนนต์นี้ภายในโมดูล หรือที่ไฟล์โมดูล `src/app/app.module.ts` ซึ่งปกติระบบจะทำให้อัตโนมัติ หลักหรือไม่ ตรวจสอบที่ `import` และ `declarations`

```

import { NewUserComponent } from './new-user/new-user.component';

@NgModule({
  declarations: [
    AppComponent,
    NewUserComponent,
  ],

```

4. ผลการสร้างคอมโพเนนต์ ซึ่งยังเป็นคอมโพเนนต์ทั่วไป ให้นำเข้า `FormControl` และสร้างค่า `ออบเจกต์ fname` และ `lname` ผ่านคอนสตรัคเตอร์ `FormControl('')` ซึ่งทั้งสองออบเจกต์ กำหนดค่าเริ่มต้นเป็นอักขรว่างภายในวงเล็บของคอนสตรัคเตอร์ ดังตัวอย่างไฟล์ `src/app/new-user/new-user.component.ts` ต่อไปนี้

```

import { Component, OnInit } from '@angular/core';
import { FormControl } from '@angular/forms';

@Component({
  selector: 'app-new-user',
  templateUrl: './new-user.component.html',
  styleUrls: ['./new-user.component.css']
})
export class NewUserComponent implements OnInit {
  fname = new FormControl('');
  lname = new FormControl('');
  constructor() { }

  ngOnInit() { }
}

```

5. ใส่ไอลีเมนต์ `<input>` ลงในไฟล์ `src/app/new-user/new-user.component.html` โดยกำหนดคำสั่งฝัง `formControl` ในไอลีเมนต์นี้ ตามชื่อออบเจกต์ ที่ได้สร้างมาก่อนหน้านี้ ให้สังเกตว่า ไม่ต้องเขียนไอลีเมนต์ `<form>` (ให้เขียนข้อมูลต่อไปนี่ แทนที่ข้อมูลเดิมทั้งหมด)

```

<label>
    First Name: {{ fname.value }}
    <input type="text" [formControl]="fname">
</label>
<label>
    Last Name: {{ lname.value }}
    <input type="text" [formControl]="lname">
</label>

```

ในตัวอย่างนี้ มีการผูกข้อมูล ด้วยปีกกาคู่ {{ }} เมื่อผู้ใช้กรอกข้อมูลในฟอร์ม ข้อมูลจะแสดงในส่วนการผูกข้อมูลนี้ด้วย

6. และสุดท้ายนำฟอร์มนี้ ไปแสดงที่ คอมโพเนนต์หลัก ที่ไฟล์ src/app/app.component.html จากไฟล์นี้ปรับปรุง (ให้เขียนข้อมูลต่อไปนี้ แทนที่ข้อมูลเดิมทั้งหมด) ข้อมูลตามตัวอย่างต่อไปนี้ และได้ผลดังรูปข้างล่าง

```

<div style="text-align:center">
    <h1>
        Welcome to {{ title }}!
    </h1>
</div>
<app-new-user></app-new-user>

```



รูป 1 ผลการสร้างรีเิกที่ฟฟอร์ม

จากที่ได้สร้างมา สิ่งสำคัญที่ควรทำความเข้าใจคือ การใช้ FormControl ในไฟล์ TS และการใช้ formControl ในไฟล์ HTML สองตัวนี้ผูกข้อมูลกันอยู่ การอ่านค่าใช้คุณสมบัติ value

### สร้างฟอร์มกลุ่ม

ที่ผ่านมา ใช้ฟอร์ม ในลักษณะแต่ละออบเจ็กต์ อีระจากกัน แต่สามารถรวมกันเป็นกลุ่มเดียว หรือออบเจ็กต์เดียว แต่มี สองออบเจ็กต์ย่อยภายในได้ ลักษณะการทำงานอย่างนี้ต้องใช้ คลาส FormGroup ดังนั้นต้องนำเข้า คลาสนี้ก่อน ดังมี ลำดับดังนี้

1. ให้คอมโพเนนต์ นำเข้า FormGroup ในไฟล์ src/app/new-user/new-user.component.ts โดยดัดแปลงจาก ไฟล์เดิม เพิ่ม FormGroup

```
import { FormGroup, FormControl } from '@angular/forms';
```

2. ปรับปรุงส่วน คลาส NewUser โดยปรับปรุงจากคลาสเดิม ให้สังเกตว่า คอนสตรัคเตอร์ของ FormGroup ใส่ข้อมูล ในลักษณะออบเจ็กต์ JSON

```
export class NewUserComponent implements OnInit {
    newUserForm = new FormGroup({

```

```

        fname : new FormControl(''),
        lname : new FormControl('')
    });
    constructor() { }
    ngOnInit() { }
}

```

สิ่งที่ผิดได้บ่อย คือ ข้อมูลใน FormGroup เป็นออบเจกต์ (JSON) ใช้เครื่องหมาย : กันแทนเครื่องหมาย = และระหว่างออบเจกต์ใช้เครื่องหมาย ลูกน้ำ ,

3. แก้ไข ไฟล์ new-user.component.html ให้สังเกตว่า ตอนนี้ต้องใส่ อีลีเมนต์ <form> แล้ว

```

<form [formGroup]="newUserForm">
  <label>First Name:
    <input type="text" formControlName="fname">
  </label>
  <label>Last Name:
    <input type="text" formControlName="lname">
  </label>
</form>

```

ผลการทำงานยังเหมือนเดิม แต่การอ้างอิงใช้ formGroup และ formControlName แทน formName สำหรับการผูกข้อมูลทางเดียว หากอยากทดสอบ ให้ใช้ {{newUserForm.value.fname}} และ {{newUserForm.value.lname}}

### สร้างอีเวนต์ให้กับฟอร์ม

FormGroup รองรับอีเวนต์ (event) หรือเหตุการณ์ การส่งฟอร์ม จากการกดปุ่มที่มีชนิดเป็น submit ได้ด้วย ngSubmit ซึ่งใช้ผูกกับฟังก์ชันที่สร้างขึ้นเองว่าให้ทำอะไรเมื่อเกิดการส่งฟอร์ม เช่น สร้างฟังก์ชัน onSubmit( ) เพื่อให้แสดงข้อมูลในส่วน console.log( ) ดังมีขั้นตอนคือ

1. แก้ไข ไฟล์ new-user.component.html ใหม่ดังนี้

```

<form [formGroup]="newUserForm" (ngSubmit)="onSubmit()">
  <label>First Name:
    <input type="text" formControlName="fname">
  </label>
  <label>Last Name:
    <input type="text" formControlName="lname">
  </label>
  <button type="submit">Submit</button>
</form>

```

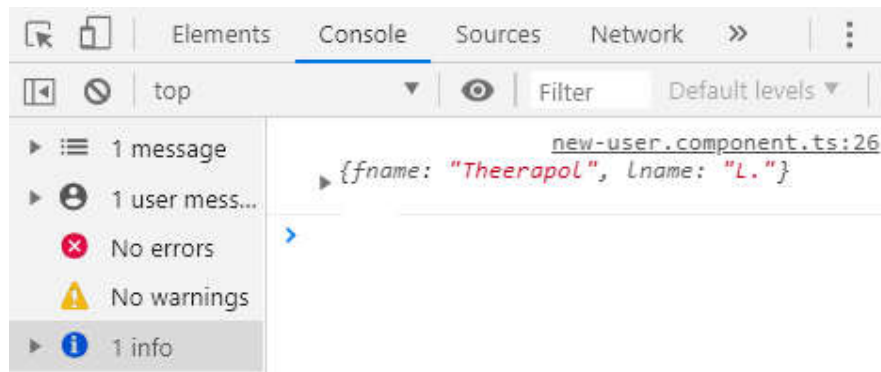
2. สร้างฟังก์ชัน onSubmit( ) เพื่ออ่านข้อมูลของฟอร์ม แล้วให้แสดงผลใน console โดยการเพิ่มฟังก์ชันนี้ในคลาส NewUserComponent (ไฟล์ new-user.component.ts)

```

onSubmit(){
  console.log(this.newUserForm.value);
}

```

3. ต่อมาทดสอบ ให้กรอกข้อมูล ทั้งในส่วน ชื่อ และนามสกุล ลงในฟอร์ม ผลการทำงานจะได้ดังรูป Console ของ Google Chrome ในส่วน Warning เมื่อคลิกส่งข้อมูล “Theerapol” และ “L.” ต่อไปนี้



รูป 2 หน้าต่าง Console (warning)

### ใช้ CSS Bootstrap

จากรูปแบบหน้าเว็บที่ผ่านมา ยังไม่ได้กำหนด CSS ซึ่งมี รูปแบบที่นิยมใช้กันแพร่หลาย คือ Bootstrap (ขณะนี้ใช้รุ่น

4) เราจะเอามาใช้ได้หลายวิธี หนึ่งในวิธีที่ง่ายที่สุด คือ การเชื่อม URL มาใส่ในหน้าหลัก index.html

#### Code 1. src/index.html

```
<head>
<meta charset="utf-8">
<title>MyAngular</title>
<base href="/">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" type="image/x-icon" href="favicon.ico">

<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css">
</head>
```

หมายเหตุ ข้อมูลนำมาจาก [https://www.w3schools.com/bootstrap4/bootstrap\\_get\\_started.asp](https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp)  
(10/1/2562)

รูป 3 ผลการใช้ Bootstrap

ต่อมาเลือก รูปแบบฟอร์ม ของ Bootstrap แล้วนำมาใช้กับฟอร์มเก่า ที่เคยสร้างไว้ ให้คงส่วนที่ต้องใช้กับ Angular ไว้ ดังตัวอย่างต่อไปนี้

**Code 2.** src/app/new-user/new-user.component.html

```
<form [formGroup]="newUserForm" (ngSubmit)="onSubmit()" style="margin:20px">
  <div class="form-group">
    <label>First Name</label>
    <input type="text" formControlName='fname'
      class="form-control" placeholder="First Name">
  </div>
  <div class="form-group">
    <label>Last Name</label>
    <input type="text" class="form-control"
      formControlName='lname' placeholder="Last Name">
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

หมายเหตุ ข้อมูลนำมาจาก <https://getbootstrap.com/docs/4.2/components/forms/> (เข้าดูเมื่อ 10/1/2562)

**ฟอร์มกลุ่ม ในฟอร์มกลุ่ม**

ฟอร์มในฟอร์ม ซึ่งคือการสร้างฟอร์มย่อย ในฟอร์มอื่น เช่น ต้องการสร้าง กลุ่มของฟอร์ม เกี่ยวกับการเรียน ที่เก็บข้อมูลเกี่ยวกับการเรียนไว้อยู่ในกลุ่มนี้ อย่างกรณี ที่มีฟอร์มหลักอยู่แล้ว ที่สร้างไว้คือ newUserForm ซึ่งเป็นฟอร์มกลุ่มตัวหนึ่ง (formGroup) แล้วเราต้องการสร้างฟอร์มกลุ่มชื่อ study อีกชุดหนึ่งภายใน newUserForm

ตัวอย่างต่อไปนี้ ใช้ ชื่อฟอร์มกลุ่มว่า study ที่ประกอบด้วยข้อมูล <input> ชื่อ id, faculty, และ major ให้สังเกตว่า อีลีเมนต์นอกสุดเป็น <div> ไม่ใช่ <form> ดังนั้นแล้ว ฟอร์มกลุ่มนี้เก็บได้ใน ฟอร์มกลุ่มของ newUserForm ตัวอย่างโปรแกรมต่อไปนี้เขียนไว้ก่อน อีลีเมนต์ <button> ของไฟล์ new-user.component.html

**Code 3.** src/app/new-user/new-user.component.html

```
<div class="form-group">
<h2 class='form-group'>Study</h2>
  <div formGroupName="study">
    <div class="form-group">
      <label>ID</label>
      <input type="number" formControlName='id' class="form-control">
    </div>
    <div class="form-group">
      <label>Faculty</label>
      <input type="text" formControlName='faculty' class="form-control">
    </div>
    <div class="form-group">
      <label>Major</label>
      <input type="text" formControlName='major' class="form-control">
    </div>
  </div>
</div>
```

จากตัวอย่างนี้ ฟอรมกลุ่มย่อยจะใช้ชื่อว่า formGroupName ซึ่งเป็นคำสั่งไตรีกที่พของแองกูลาร์ ส่วนที่เขียน class เป็น รูปแบบสไลต์ชีท ( style sheet) ในการจัดหน้าเว็บไซต์ ไม่เกี่ยวอะไรกับแองกูลาร์ และสมาชิกในกลุ่มย่อยก็ยังคงใช้ชื่อ FormControlName

เมื่อสร้างฟอรมกลุ่มขึ้นมาใหม่แล้วในไฟล์ new-user.component.ts ให้สร้างฟอรมกลุ่มในไฟล์ new-user.component.ts ด้วย โดยกำหนด โครงสร้างในรูปแบบ JSON อยู่ใน JSON โดยมีการกำหนดค่า เริ่มต้นให้กับข้อมูล faculty เป็น 'Science and Technology' และ major เป็น 'Information Technology' ค่าเหล่านี้ จะแสดงในฟอรม ถือเป็น ค่าปริยาย ให้ผู้ใช้งานเว็บ ดังตัวอย่างต่อไปนี้

**Code 4.** src/app/new-user/new-user.component.ts

```
newUserForm = new FormGroup({
  fname : new FormControl(''),
  lname : new FormControl(''),

  study: new FormGroup({
    id:new FormControl(''),
    faculty:new FormControl('Science and Technology'),
    major:new FormControl('Information Technology')
  })
});
```

จากตัวอย่างนี้จะเห็นว่า สมาชิก study เป็นสมาชิกตัวหนึ่งของ newUserForm และสมาชิก study มีสมาชิกย่อย ๆ อีกหลายตัวคือ id, faculty, และ major ซึ่งจัดเป็น FromGroup ย่อยตัวหนึ่ง การอ้างอิงข้อมูลในฟอรมจะอยู่ในรูปแบบลำดับชั้น เช่น ถ้าต้องการอ้างอิงถึงข้อมูล major จะใช้ newUserForm.value.study.major

The image shows a web form with the following structure:

- First Name**: A text input field with the placeholder text "First Name".
- Last Name**: A text input field with the placeholder text "Last Name".
- Study**: A section header.
- ID**: A text input field.
- Faculty**: A text input field with the value "Science and Technology".
- Major**: A text input field with the value "Information Technology".
- Submit**: A blue button with the text "Submit".

รูป 4 ผลการแสดงผลฟอรมกลุ่มในฟอรมกลุ่ม

## การดำเนินการกับข้อมูลในฟอร์มกลุ่ม

การอ่านข้อมูลในฟอร์ม ตามตัวอย่างก่อนนี้ใช้ อ้างอิงตามชื่อออบเจกต์ FormControl แต่ฟอร์มกลุ่ม มี FormGroup คลอบ FormControl การอ่าน มีลักษณะต่างจากเดิมเล็กน้อย คือใช้ value นำชื่อที่ต้องการจะอ่าน เช่น ถ้าต้องการอ่าน fname และ major จะใช้ลักษณะการอ่านคือ:

```
this.newUserForm.value.fname  
this.newUserForm.value.study.major
```

การอ่านค่าจาก FormControl อีกวิธีหนึ่ง ใช้ ฟังก์ชัน `get()` โดยใส่ตัวแปร เช่น

```
this.newUserForm.get('study.faculty').value
```

กรณีที่ต้องการ ส่งไปเป็นค่าออบเจกต์ ไปยังอ้างอิงในชื่อตัวแปรอื่น จะต้องใส่ชนิดข้อมูลด้วย (type) เป็นชื่อของ FormControl เนื่องจากภาษา TypeScript บังคับเรื่องชนิดข้อมูล เช่น ต้องการ ส่งไปยังตัวแปร majorFc และเรียกอ่านค่า ภายหลัง จะเขียนได้ว่า :

```
var majorFc : FormControl = this.newUserForm.get('study.major');  
console.log(majorFc.value);
```

ส่วนการกำหนดค่าของ FormControl ทำได้ด้วยฟังก์ชัน `setValue()` เมื่อต้องการดำเนินการกับทุกค่า (ย้่าว่าทุกค่าที่มี อยู่ในฟอร์มกลุ่ม) เช่น

```
this.newUserForm.setValue({  
    fname: 'A', lname: 'B',  
    study: {id: 1, faculty: 'C', major: 'D'}  
});
```

การใช้ `setValue()` เหมาะสมกับฟอร์มธรรมดามากกว่า ที่มีค่าเดียว แต่ไม่เหมาะกับฟอร์มกลุ่ม เพราะจะต้องเขียนลง ทุกค่า จึงมีฟังก์ชัน `patchValue()` ที่สามารถปรับปรุงหรือกำหนดค่าได้บางค่าที่มีอยู่แล้วใน ฟอร์มกลุ่ม (ไม่ได้เป็นการเพิ่มค่า ใหม่) เช่น

```
this.newUserForm.patchValue({study: {major: 'IT'}});
```

แต่ถ้าต้องการเพิ่มค่าใหม่ในฟอร์มกลุ่ม ใช้ฟังก์ชัน `addControl()` เช่นต้องการเพิ่มค่าใหม่ในชื่อ gender ให้เขียนว่า:

```
this.newUserForm.addControl('gender' , new FormControl(''));
```



ข้อควรระวัง การพิมพ์ผิดนิดเดียวก็ทำให้โปรแกรมทำงานไม่ได้ เช่น ถ้าชื่อ `newUserForm` แต่เขียนเป็น `newuserForm` ต่างกันแค่ตัว u เล็ก กับ U ใหญ่ ควรใช้ IDE ที่ช่วยตรวจหาได้ง่าย เช่น Visual Studio Code

สมมุติว่าต้องการปรับปรุงข้อมูลเมื่อผู้ใช้ คลิกส่งฟอร์ม ซึ่งเดิมฟอร์มนี้รองรับฟังก์ชัน `onSubmit()` อยู่แล้ว ให้นำมา ปรับปรุงทดลองการปรับปรุงข้อมูล ตามตัวอย่างต่อไปนี้



#### Code 5. src/app/new-user/new-user.component.ts

```
onSubmit(){
  console.log(this.newUserForm.value);
  this.newUserForm.patchValue({study:{major:'IT'}});
  console.log(this.newUserForm.value.study.major);
}
```

#### ฟอร์มบิลเดอร์ (FormBuilder)

ตัวสร้างฟอร์มอีกตัวหนึ่ง ที่อำนวยความสะดวกในการสร้างฟอร์มจำนวนมาก ๆ ซึ่งฟอร์มเป็นหัวใจสำคัญในการใช้งานระบบงานเว็บ ด้วยการใช้ฟอร์มบิลเดอร์ที่จะช่วยทำการสร้างฟอร์มทำได้เร็วขึ้น

การสร้างฟอร์มบิลเดอร์ ให้แก้ไขไฟล์ new-user.component.ts ดังมีขั้นตอนดังนี้

1. นำเข้าคลาส FormBuilder ซึ่งใช้แทน FormGroup และ FormControl ได้ เพราะใช้สำหรับสร้างฟอร์มเหล่านี้

```
import { FormBuilder } from '@angular/forms';
```

2. สร้างสมาชิกผ่าน คอนสตรัคเตอร์ วิธีการนี้เรียกอีกอย่างว่า การฉีดบริการ (Injection service) ซึ่งจำเป็นต้องมีคีย์เวิร์ดว่า public หรือ private

```
constructor(private fb: FormBuilder) { }
```

3. สร้างฟอร์มคอนโทรล ในรูปแบบแฟกตอรี (factory) เปรียบเหมือนโรงงานสร้างวัตถุ (วางในคอนสตรัคเตอร์)

```
newUserForm = this.fb.group({
  fname : [''],
  lname : [''],
  study: this.fb.group({
    id:[''],
    faculty:['Science and Technology'],
    major:['Information Technology']
  })
});
```

จากขั้นตอนการสร้างฟอร์มคอนโทรล ใช้ ฟังก์ชัน group( ) แทน FormGroup ข้อมูลภายในยังคงรูปแบบ JSON ค่าแต่ละตัวอยู่ในรูปอาร์เรย์ ซึ่งเมื่อแก้ไขแล้ว การทำงานยังคงได้ผลเหมือนเดิม รวมทั้ง ฟังก์ชัน onSubmit( ) ก็ยังทำงานได้กับการอ่านค่า และการปรับค่า (this.newUserForm.value, this.newUserForm.patchValue( )) ก็ยังทำงานได้เหมือนเดิม

จากที่สร้างฟอร์มบิลเดอร์ เราใส่ค่าของแต่ละอินพุต (input) ได้เลย แต่ถ้าต้องการกำหนดคุณสมบัติเพิ่มเติม เช่น การกำหนดไม่ผู้กรอกข้อมูลในอินพุตได้ การกำหนดเงื่อนไขการตรวจสอบ การเขียนต้องอยู่ในรูปแบบ JSON ตัวอย่างเช่น:

```
fname : [{value: 'Pol', disabled: true}, Validators.required],
```

แต่วิธีการนี้อาจทำให้การส่งค่าไปไม่ครบได้ หรือใช้การการอ่านค่า ด้วยการใช้คุณสมบัติ value ไม่ได้ต้องใช้งานผ่าน ฟังก์ชัน get( ) แทน ดังนั้นเพื่อหลีกเลี่ยงปัญหานี้ควรใช้การกำหนดให้อ่านค่าอย่างเดียวในฟอร์ม HTML

```
<input [readonly]=true formControlName="fname">
```

## ฟอร์มอาร์เรย์ (FormArray)

กรณีที่ต้องการให้ฟอร์มมีขนาดยืดหยุ่นได้ หรือไม่ทราบจำนวนฟอร์มคอนโทรลที่แน่นอน การใช้ฟอร์มอาร์เรย์ก็เป็นตัวช่วยที่ดีตัวหนึ่ง

การใช้ฟอร์มอาร์เรย์ต้องนำเข้า FormArray และการสร้างที่ง่ายที่สุดก็ควรสร้างจากฟอร์มบิลเดอร์ การอ้างอิงคำสั่งไคเรคทีฟ ก็ใช้ชื่อ formArrayName ซึ่งต่างกับฟอร์มกลุ่มที่ใช้ชื่อ formGroup และกลุ่มย่อยใช้ชื่อ formGroupName จากบทความการใช้ชื่อของฟอร์มกลุ่ม มีลำดับที่ทำให้สับสนได้บ้าง การทำความเข้าใจฟอร์มกลุ่ม และฟอร์มกลุ่มในฟอร์มกลุ่มให้ดี จะช่วยลดความสับสนที่เกิดขึ้นใหม่กับฟอร์มอาร์เรย์ได้

การสร้างฟอร์มให้มีความยืดหยุ่นได้นั้น โดยทั่วไปก็ไม่สร้างฟอร์มอาร์เรย์ขึ้นมาเดี่ยว ๆ แต่มักอยู่ในฟอร์มกลุ่มอีกที และบางทีในฟอร์มอาร์เรย์ก็มีฟอร์มกลุ่มได้ กลับไปกลับกันมาได้ ซึ่งเป็นเรื่องธรรมดาที่เก็บขึ้นเสมอสำหรับการสร้างฟอร์มแบบไดนามิก

เพื่อให้เข้าใจได้ง่าย ๆ FormGroup จะใช้เครื่องหมาย {} แต่ FormArray จะใช้เครื่องหมาย [] ดังนั้นถ้า FormArray อยู่ใน FormGroup การเขียนจะอยู่ในรูปแบบ:

```
fromGroup({
  a: new FormControl(''),
  b: new FormArray([
    new FormControl(''),
    new FormControl(''),
  ])
});
```

## สร้างฟอร์มอาร์เรย์ในฟอร์มกลุ่ม

การเพิ่มฟอร์มคอนโทรลแบบไดนามิกด้วย ฟอร์มอาร์เรย์นี้ ให้สมมุติงานว่า ต้องการเพิ่มหมายเลขโทรศัพท์ ได้เรื่อย ๆ ขึ้นตอนมีดังนี้

1. ให้เพิ่มการนำเข้า FormArray ของไฟล์ new-user.component.ts

```
import { FormArray } from '@angular/forms';
```

2. สร้างฟอร์มอาร์เรย์ ใช้ชื่อ tels จาก ฟอร์มบิลเดอร์

```
newUserForm = this.fb.group({
  fname : [''],
  lname : [''],
  study: this.fb.group({
    id:[''],
    faculty:['Science and Technology'],
    major:['Information Technology']
  }),
  tels:this.fb.array([
    this.fb.control('')
  ]),
});
```

3. สร้างฟังก์ชันประเภท getter ในชื่อ tels( ) ฟังก์ชันนี้จะคืนค่าอาร์เรย์ FormArray

```
get tels() {  
    return this.newUserForm.get('tels') as FormArray;  
}
```

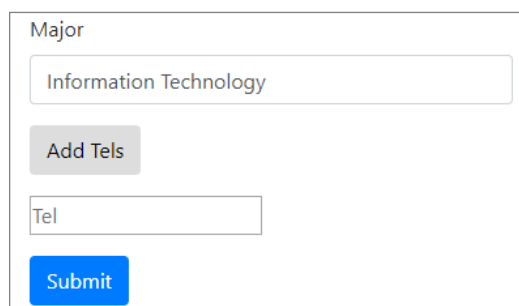
และเพิ่มฟังก์ชัน addTels( ) เพื่อเพิ่มคอนโทรลเข้า FormArray จากฟังก์ชันนี้จะเห็นว่า ใช้ this.tels ซึ่งเป็นคุณสมบัติของฟังก์ชันประเภท getter ไม่จำเป็นต้องใส่เครื่องหมายวงเล็บเหมือนฟังก์ชันทั่วไป

```
addTels() {  
    this.tels.push(this.fb.control(''));  
}
```

4. ในไฟล์ new-user.component.html ให้เพิ่ม <button> และ <input> โดย <input> อยู่ในการส่วนการวนซ้ำของฟอร์มอาร์เรย์ ด้วยคำสั่ง \*ngFor โดยให้ชื่อ ฟอร์มคอนโทรล เป็นเลขลำดับ (index) โดยทั้งหมดนี้ให้เพิ่มก่อน ปุ่ม Submit

```
<div formArrayName="tels">  
    <div class="form-group">  
        <button (click)="addTels()" class="btn">Add Tels</button>  
    </div>  
    <div *ngFor="let tel of tels.controls; let i=index"  
        class="form-group">  
        <input type="text" [formControlName]="i" placeholder="Tel">  
    </div>  
</div>
```

ซึ่งมีผล การทำงานดังรูปต่อไปนี้



รูป 5 การทำงานของฟอร์มอาร์เรย์

สิ่งที่น่าสนใจเพิ่มเติมจากการใช้ FormArray คือการอ่านอาร์เรย์จาก FormArray ใช้การอ่านจากฟังก์ชัน get tels( ) ให้สังเกตว่า get กับ tels เขียนไม่ติดกัน ซึ่งเป็นฟังก์ชัน getter/setter และกำหนดสมาชิกในรูปแบบพร็อพเพอร์ตี้ (property) อย่างหนึ่งที่ใช้เหมือนกับภาษา C# ทำให้สามารถกำหนดให้ผ่าน this.tels ได้

ส่วนการวนอ่านในส่วน HTML ใช้การอ่าน let ... of ... ; let i=index การอ่านแบบนี้เป็นการอ่านทั้งค่า และเลขดัชนีพร้อมกัน นอกจากนี้ จะนำ i ไปตั้งชื่อได้ด้วย แต่ใส่เครื่องหมายคำพูดคลุมไว้

สำหรับการลบฟอร์มอาร์เรย์ทีละตัว จากที่เราทราบ ค่า i ซึ่งเป็น index ของแต่ละตัว เราก็สร้างฟังก์ชันลบ removeTels(i) โดยผูกไว้กับปุ่มลบ โดยลบตามค่าเลขดัชนี i

**Code 6.** src/app/new-user/new-user.component.html

```
<div *ngFor="let tel of tels.controls; let i=index" class="form-group">
  <input type="text" [formControlName]="i" placeholder="Tel">
  <button (click)="removeTels(i)">Remove</button>
</div>
```

**Code 7.** src/app/new-user/new-user.component.html

```
removeTels(i: number): void {
  this.tels.removeAt(i);
}
```

The image shows a user interface for adding phone numbers. At the top is a button labeled 'Add Tels'. Below it are two identical rows. Each row consists of a text input field with the placeholder text 'Tel' and a button labeled 'Remove' to its right.

รูป 6 เพิ่มปุ่มลบ

สำหรับการอ้างอิงข้อมูลในฟอร์มอาร์เรย์ ในกรณีตัวอย่างนี้ ใช้การอ่านจากค่าพร็อบเพอร์ตี้ `tels` ซึ่งเราได้ นิยามพร็อบเพอร์ตี้ในรูปแบบ `getter` ไว้แล้ว ข้อมูลผ่านได้อยู่ในรูปอาร์เรย์ เช่น การอ่านที่ค่าที่ 0

```
this.tels.value[0]
```

แต่ถ้าอ่านทั้งหมดจะเรียกอ่านโดยไม่ต้องเติมค่าเลขลำดับในอาร์เรย์

```
this.tels.value
```

### สร้างฟอร์มกลุ่มในฟอร์มอาร์เรย์

เราได้สร้างฟอร์มอาร์เรย์มาแล้ว ซึ่งสามารถสร้างความยืดหยุ่นการเพิ่มฟอร์มได้ แต่ที่ผ่านมาเพิ่มทีละหนึ่งอินพุท (input) และถ้ารวมปุ่มกดเพื่อลบด้วย ก็สองอินพุทที่ไม่มีชื่อเรียก เหมือนอย่างในฟอร์มกลุ่ม

การสร้างชุดของอินพุทที่สัมพันธ์กัน เช่น สมมติให้สร้างกลุ่มข้อมูลวิชาที่ศึกษามาแล้ว ซึ่งมีชื่อวิชา รหัสวิชา ปี การศึกษา และเกรดที่ได้ จัดเป็นฟอร์มกลุ่ม ที่อยู่ในฟอร์มอาร์เรย์ ทำให้สามารถเพิ่มได้หลายวิชา

ในตัวอย่างต่อไปนี้จะเขียน ฟอร์มอาร์เรย์ `courses` ซึ่งบรรจุชุดฟอร์มกลุ่ม ให้เขียนไว้ต่อท้าย ฟอร์มอาร์เรย์ `tels` ที่เคยทำก่อนหน้านี้

**Code 8.** src/app/new-user/new-user.component.ts

```
tels: this.fb.array([ this.fb.control('') ]),
courses: this.fb.array([
  this.fb.group({
    id: [''],
    name: [''],
    grade: [''],
    term: [''],
```

```
    })  
  ],
```

จากการเพิ่มฟอร์มกลุ่มในฟอร์มอาร์เรย์ มีสิ่งน่าสังเกตคือ ฟอร์มกลุ่ม ไม่ต้องกำหนดชื่อ เพราะถือเป็นอาร์เรย์ตัวหนึ่ง ดังนั้นการอ้างอิงฟอร์มกลุ่มก็อ้างอิงในลำดับของอาร์เรย์ (0, 1, 2, ..) แต่ภายในฟอร์มกลุ่มยังคงอ้างอิงในชื่อได้อยู่เหมือนการอ้างอิงฟอร์มกลุ่มเหมือนเดิม

สำหรับการเขียนไฟล์ HTML รูปแบบอย่างย่อ ที่ยังไม่ได้กำหนด CSS คือ

```
<div fromArrayName='courses'  
  *ngFor='let course of courses.controls; index as i'  
  <div [formGroupName]='i'  
    <input type='number' formControlName='id'  
    <input type='text' formControlName='name'  
    <input type='text' formControlName='grade'  
    <input type='text' formControlName='term'  
  </div>  
</div>
```

จากรูปแบบอย่างย่อนี้ สิ่งน่าสังเกตคือ courses เป็นชื่อฟอร์มอาร์เรย์ที่อยู่ในฟอร์มกลุ่ม จำเป็นต้องอ่านผ่านค่าฟังก์ชัน getter เหมือนที่เคยได้ทำกับ ฟอร์มอาร์เรย์ tels เพราะจะต้องแปลงเป็นฟอร์มอาร์เรย์ในฟังก์ชัน getter จะไม่สามารถอ้างอิงในชื่อ newUserForm.courses ได้ เพราะแองกูลาร์จะเข้าใจว่าเป็นไทป์ FormGroup

---

**Code 9. src/app/new-user/new-user.component.ts**

---

```
get courses() {  
  return this.newUserForm.get('courses') as FormArray;  
}
```

ข้อสังเกตอีกประการคือ การอ้างอิงฟอร์มกลุ่มที่อยู่ในฟอร์มอาร์เรย์ จะไม่มีชื่อ แต่จะอ้างอิงผ่านลำดับในอาร์เรย์ ในตัวอย่างเลือกใช้ค่า i และ อ้างอิงในชื่อ [formGroupName]

รูปแบบที่ใช้ร่วมกับ CSS (ใช้กับ Bootstrap) เป็นตัวอย่างดังนี้ (เขียนต่อจากตัวอย่างที่ผ่านมา วางก่อนปุ่ม submit)

---

**Code 10. src/app/new-user/new-user.component.html**

---

```
<div formArrayName='courses' *ngFor='let course of courses.controls; index as i'  
  <div [formGroupName]='i'  
    <div class="form-group">  
      <label>ID</label>  
      <input type='number' formControlName='id' class="form-control">  
    </div>  
    <div class="form-group">  
      <label>Name</label>  
      <input type='text' formControlName='name' class="form-control">  
    </div>  
    <div class="form-group">  
      <label>Grade</label>  
      <input type='text' formControlName='grade' class="form-control">
```

```

    </div>
    <div class="form-group">
      <label>Term</label>
      <input type='text' fromControlName='term' class="form-control">
    </div>
  </div>
  <div class="form-group">
    <button (click)="addCourse()" class="btn">Add Course</button>
  </div>
</div>

```

ID

Name

Grade

Term

Add Course

รูป 7 ผลการเพิ่มฟอร์มกลุ่มในฟอร์มอาร์เรย์

สำหรับฟังก์ชัน `addCourse()` ใช้สำหรับเพิ่มฟอร์มกลุ่มในฟอร์มอาร์เรย์ เป็นการเพิ่มทั้งฟอร์มกลุ่ม การเพิ่มใช้ฟังก์ชัน `push()` ซึ่งฟังก์ชันสำหรับการเพิ่มของอาร์เรย์ทั่วไป อ้างอิงการผ่านผ่าน `this.courses` (ฟังก์ชัน `getter`)

**Code 11.** `src/app/new-user/new-user.component.ts`

```

addCourse(): void {
  this.courses.push(
    this.fb.group({
      id: [''],
      name: [''],
      grade: [''],
      term: [''],
    })
  );
}

```

### สร้างฟอร์มแบบ Select/Option

กรณีการให้พิมพ์ลงใน `<input type="text">` บางครั้งก็ไม่อำนวยความสะดวกแก่ผู้กรอกข้อความ ถ้าจำนวนข้อมูลที่เลือกมีจำกัดอยู่แล้ว และอีกอย่างเพื่อป้องกันการป้อนข้อมูลที่ผิดพลาดได้ด้วย

ใน HTML มีให้ใช้ <select> ที่เป็นรายการให้เลือก ในกรณีของคณะ (faculty) ในตัวอย่างที่ผ่านมา เราน่าจะเปลี่ยนเป็นให้ผู้เลือกมากกว่า ดังนั้นแล้วในส่วน <input> เดิม ให้แก้เฉพาะส่วนนี้ ดังนี้

**Code 12.** src/app/new-user/new-user.component.html

```
<div class="form-group">
  <label>Faculty</label>
  <select formControlName='faculty' class="form-control">
    <option *ngFor="let opt of faculties"
      [ngValue]="opt">{{opt}}</option>
  </select>
</div>
```

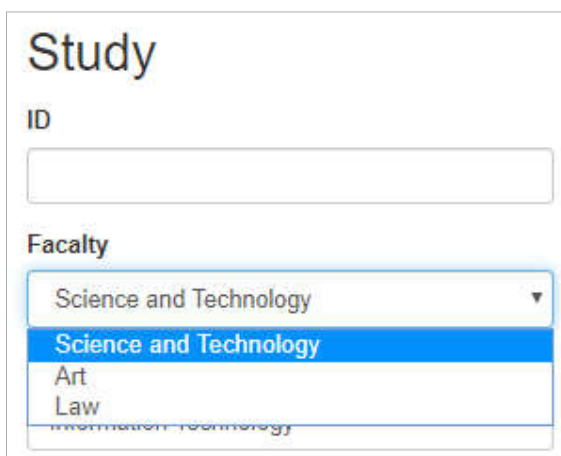
ในตัวอย่างนี้ สมมติ ให้มีเลือก 3 คณะ เพิ่มเติม Art และ Law โดยกำหนดเป็นสมาชิกหนึ่งของคลาส ในชื่อ faculties ดังนั้นต้องเพิ่มสมาชิกนี้ในคลาส NewUserComponent

**Code 13.** src/app/new-user/new-user.component.ts

```
faculties: string[] = ['Science and Technology', 'Art', 'Law'];
```

ผลจากเลือกคณะใหม่ เช่น เลือก Art เราต้องการจะทราบ ว่า เมื่อ กดปุ่ม Submit จะได้ข้อมูลนี้จริงไหม เราอาจทดสอบผ่าน ฟังก์ชัน onSubmit() ซึ่งมีอยู่ โดยเพิ่มเข้าไปอีกหนึ่งบรรทัด:

```
console.log(this.newUserForm.value.study.faculty);
```



รูป 8 ฟอรัมแบบมีตัวเลือก

เมื่อมีการเลือกคณะใดคณะหนึ่ง อันนี้ถือเป็นเหตุการณ์หรืออีเวนต์ (event) หนึ่งที่เปลี่ยนแปลง การกำหนดอีเวนต์ทำได้โดยการเติม คำสั่งเหตุการณ์ change โดยผูกกับฟังก์ชันใดฟังก์ชันหนึ่ง เช่น ผูกกับฟังก์ชัน selectFaculty()

```
<select formControlName='faculty'
  (change)="selectFaculty()" class="form-control">
```

ฟังก์ชัน selectFaculty() ยังไม่ได้สร้างมาก่อนในคลาส NewUserComponent จึงต้องสร้างฟังก์ชันนี้เพิ่มเติมลงในคลาสนี้ เช่นเติม ทดสอบ การอ่านค่าที่เลือกกว่าได้ผลหรือไม่ โดยการให้แสดงหน้าต่างเตือน (alert)

**Code 14. src/app/new-user/new-user.component.ts**

```
selectFaculty(){  
    alert(this.newUserForm.value.study.faculty);  
}
```

เนื่องจาก study เป็นกลุ่มย่อยใน newUserForm จึงอ่านผ่าน value อีกทีหนึ่ง อันนี้พึงระวังว่าไม่ได้อ่านผ่าน newUserForm โดยตรง

การรับอีเวนต์การเปลี่ยนแปลงได้ผ่าน คำสั่ง change ทำให้คิดต่อไปได้ว่า เมื่อเลือกคณะแล้ว ข้อมูลที่ต้องเปลี่ยนแปลง สาขาวิชาที่อยู่ถัดไป ควรจะขึ้นอยู่คณะที่เลือก ดังนั้นจึงควรเขียนเพิ่มเติมเพื่อรองรับความเปลี่ยนแปลงการเลือกของแต่ละคณะ ดังจะได้ทำต่อไปในหัวข้อถัดไป

**เปลี่ยนสาขาวิชา เป็น Radio**

เพื่อเป็นการเพิ่มทางเลือกอีก นอกจาก การเลือกแบบ <select> ยังมีการเลือกแบบ เรดิโอ (type=radio) ที่เป็นการบังคับเลือกเพียงอย่างอีกแบบหนึ่ง

จากตัวอย่างที่ผ่านมา มีการเลือก รายชื่อคณะ จากอีลิเมนต์ <select> เมื่อเลือกแล้ว สาขาวิชา ก็ควรจะเปลี่ยนแปลงตามด้วย ดังนั้น สิ่งที่เราสร้างเพิ่ม อย่างแรกคือ รายชื่อสาขาวิชาตามคณะ ให้เพิ่มสมาชิกใหม่ facMajors เก็บข้อมูลในรูปของเจ็ท สมาชิก majors เก็บรายการสาขาวิชา และ @Input message เป็นข้อความทั่วไป ที่อาจนำไปใช้ในส่วนต่างๆ ที่หน้าเว็บ

**Code 15. src/app/new-user/new-user.component.ts**

```
facMajors: object= {  
    'Science and Technology':['Information Technology', 'Data Science'],  
    'Art':['English Education', 'Linguistics'],  
    'Law':['Bus.Law', 'General Law']  
};  
majors: string[] = [];  
@Input() message: string;
```

ในการเปลี่ยนแปลงการเลือกคณะที่สร้างในฟังก์ชัน selectFaculty( ) ให้เตรียมรับการเลือกคณะใด ก็ให้ majors เก็บสาขาวิชาตามคณะ ใช้แบบตัวเลือก if ธรรมดา สำหรับ message เป็นข้อความทั่วไป ที่อาจนำไปแสดงที่หน้าเว็บ

**Code 16. src/app/new-user/new-user.component.ts**

```
selectFaculty(){  
    let fac: string = this.newUserForm.value.study.faculty;  
    this.message = "Major";  
    if(fac=='Science and Technology'){  
        this.majors = this.facMajors['Science and Technology'];  
    }  
    else if(fac=='Art'){  
        this.majors = this.facMajors['Art'];  
    }  
    else if(fac=='Law'){  
        this.majors = this.facMajors['Law'];  
    }  
}
```



เมื่อได้เขียนในส่วนข้อมูลที่ต้องการแล้ว ต่อไปก็จะเป็นการนำส่วนข้อมูลไปแสดงในส่วน html จากเดิมที่แสดงเพียงกล่องข้อความอย่างเดียว ให้เปลี่ยนมาใช้แบบ radio แทน ในตัวอย่างต่อไปนี้จะใช้การวนซ้ำ \*ngFor ทัวไปกับข้อมูลอาร์เรย์ majors

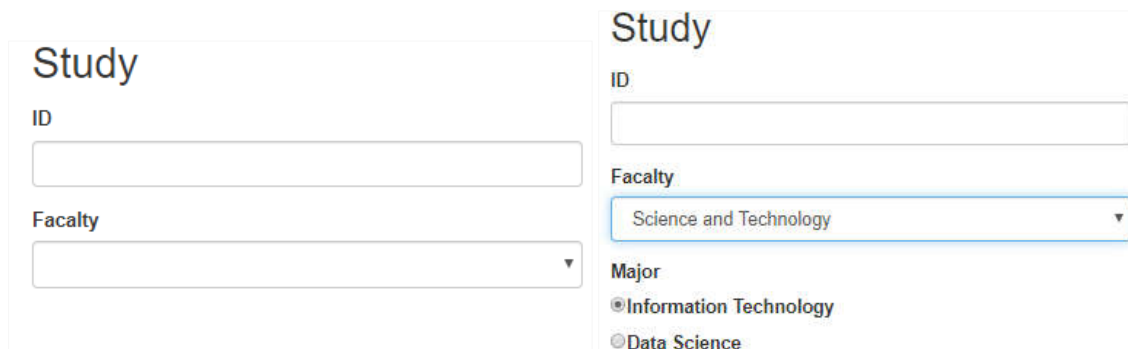
**Code 17.** src/app/new-user/new-user.component.html

```
<div class="form-group">
  <label>{{message}}</label>
  <div *ngFor="let major of majors">
    <div class="form-check">
      <input class="form-check-input" FormControlName='major'
        type="radio" value={{major}} required>
      <label class="form-check-label">
        {{major}}
      </label>
    </div>
  </div>
</div>
```

ในตัวอย่างโปรแกรมนี้ จะสังเกตเห็นว่า <label>{{message}}</label> มีค่าเป็น “Major” ตามที่กำหนดตามฟังก์ชัน selectFaculty( ) ซึ่งมีค่าภายหลังเลือกฟังก์ชันนี้แล้ว ทำให้ในตอนแรก message ไม่มีค่าอะไร ผลคือแสดงอะไรไป ด้วย

เมื่อเลือกคณะและสาขาแล้วทดสอบผลการเลือกอีกครั้ง ในฟังก์ชัน onSelect( ) โดยส่งสองบรรทัดต่อไปนี้ ให้แสดงผลที่ console.log( )

```
console.log(this.newUserForm.value.study.faculty);
console.log(this.newUserForm.value.study.major);
```



รูป 9 ขณะเริ่มแรก (ซ้าย) และหลังเลือกคณะ (ขวา)

### การตรวจความผิดพลาดของฟอร์ม

ฟอร์มควรมีการตรวจการใส่ข้อมูลของผู้ใช้ว่า ได้ใส่ข้อมูลถูกต้องบ้างหรือไม่ การตรวจทำได้หลายแบบ เช่น การใส่ครบหรือไม่ การใส่ข้อมูลตรงรูปแบบหรือไม่ เหล่านี้ตรวจสอบได้ โดยการใช้ ฟังก์ชันของ Validators ซึ่งจะตรวจความถูกต้องผ่านคอนโทรล และจะคืนค่า ความผิดพลาด หรือ ค่า null เพื่อการตรวจสอบได้

ขั้นตอนการตรวจสอบ มีดังนี้

1. นำเข้าฟังก์ชัน Validators จากตัวอย่างที่ผ่านมา ให้การนำเข้าที่ไฟล์ new-user.component.ts

```
import { Validators } from '@angular/forms';
```

2. เลือกคอนโทรลที่ต้องการจะตรวจสอบ เช่น เลือก ให้อินพุต fname ไม่ต้องการให้ผู้ใช้งานเว้นว่างไว้ โดยใช้ข้อมูล Validators.required

```
fname : ['', Validators.required]
```

3. การตรวจสอบค่าว่าง นี้ทำในเอกสาร HTML ร่วมด้วย ในไฟล์ new-user.component.html

```
<input type="text" formControlName='fname' required class="form-control" placeholder="First Name">
```

4. กำหนดปุ่มส่งฟอร์ม ให้ไม่ทำงาน เมื่อ ความผู้ยังใส่ข้อมูลไม่ครบ หรือยังไม่ถูกต้อง

```
<button type="submit" [disabled]="!newUserForm.valid" class="btn btn-primary">Submit</button>
```

5. เพิ่มเติมการ แสดงข้อความ ว่า VALID หรือ INVALID ไว้ท้าย <button>

```
<p> Form Status:{{ newUserForm.status }}</p>
```

6. ทดสอบ การกรอกข้อมูล และใส่ข้อมูลของคอนโทรล fname



รูป 10 ผลการทำงาน กรณียังไม่ถูกต้อง (ซ้าย) และถูกต้องแล้ว (ขวา)

### ตัวตรวจสอบแบบ ซิงโครนัส (Sync Validators)

จากตัวอย่างที่ผ่านมา มีการตรวจสอบสถานะความถูกต้อง (Valid, invalid) ที่ใช้กับ การกำกับ required ใน HTML ยังมีตรวจสอบความผิดพลาดของที่มีมาให้ (Built-in Validators) ที่ตรงกับการกำกับใน HTML เช่น minLength, maxLength ซึ่งดูเพิ่มเติมได้เอกสาร API ของ Angular<sup>1</sup>

ตัวตรวจสอบเหล่านี้มีชนิดเป็น ซิงโครนัส ซึ่งใส่เป็นตัวแปรที่สอง ซึ่งมีการตรวจสอบมากกว่า 1 อย่างจะต้องทำให้อยู่ในรูปอาร์เรย์ และเมื่ออยู่ในรูปอาร์เรย์ก็จะกลายเป็นตัวแปรที่สองได้ ซึ่งเป็นไปตามเงื่อนไขที่ต้องกำหนดเป็นตัวแปรที่สอง

ตัวอย่างต่อไป นี้ ดัดแปลงจากตัวอย่างก่อนหน้านี้ โดยการเพิ่ม การตรวจสอบความผิดพลาดเพิ่มเติม ให้ขนาดของชื่อมีอย่างน้อย minLegth(10) การกำหนดค่านี้จะต้องกำหนดในไฟล์ HTML ด้วย

ดังนั้นแล้ว การกำหนดค่าของ newUserForm ในไฟล์ new-user.component.ts จะต้องแก้ไขใหม่ดังนี้

---

<sup>1</sup> Angular. Validators. <https://angular.io/api/forms/Validators> (เข้าดูเมื่อ 8 เมษายน 2562)

**Code 18.**

```

newUserForm = this.fb.group({
  fname : ['',
    [Validators.required,
    Validators.minLength(6)]
  ],
  lname : [''],
  study: this.fb.group({
    id:[''],
    faculty:['Science and Technology'],
    major:['Information Technology']
  }),
  tels:this.fb.array([
    this.fb.control('')
  ])
});

```

**สรุป**

รีเิกทีฟฟอร์ม เน้นสร้างฟอร์มคอนโทรล ด้วย TS มีข้อดีที่สามารถสร้างฟอร์มแบบไดนามิกได้ดี ดังใช้จาก FormControl, FormGroup, FormBuilder, FormArray ซึ่งทั้งหมดมาจากโมดูล ReactiveFormsModule การใช้ FormControl ใช้ในรูปแบบอิสระ แต่โดยทั่วไป เรามักใช้แบบ FormGroup มากกว่า และยังพบอีกว่า ใน FormGroup ก็มี FormGroup ซ่อนอีกที เมื่อฟอร์มมีการสร้างที่ซับซ้อนขึ้น การนำ FormBuilder จะช่วยในการสร้างฟอร์มได้ง่ายและกระชับขึ้น การใช้ FormArray จะช่วยสร้างความยืดหยุ่นให้ฟอร์มที่ไม่ทราบจำนวน input ที่ไม่แน่นอน นอกจากนี้ ยังมีการสร้างอีเวนต์ให้กับฟอร์ม เพื่อรองรับข้อมูลไดนามิก และการตรวจสอบความถูกต้องในการใช้ฟอร์ม ก่อนส่งข้อมูลไปยังเซิร์ฟเวอร์

**คำถามทบทวน**

1. การใช้รีเิกทีฟฟอร์ม จะต้องนำเข้าโมดูลใด
2. การสร้างอินพุตจากรีเิกทีฟฟอร์ม เป็นออบเจกต์ สร้างจากจากคลาสใด
3. การสร้างอินพุตจากรีเิกทีฟฟอร์ม เป็นออบเจกต์ในแบบฟอร์มกลุ่ม สร้างจากจากคลาสใด
4. การเขียน ngSubmit กำกับที่ส่วนใดของฟอร์ม
5. อะไรคือความหมายของการใช้งานฟังก์ชัน get( ), patchValue( ) และ setValue( ) ของการอ้างอิงผ่านออบเจกต์รีเิกทีฟฟอร์ม
6. อะไรคือข้อดีของการใช้ ฟอร์มบิลเดอร์ แทนการใช้ฟอร์มกลุ่มแบบธรรมดา
7. มีเหตุผลใดที่ต้องสร้างฟอร์มอาร์เรย์
8. การตรวจสอบความผิดพลาดต้องนำเข้าอะไรจากโมดูล forms
9. การกำหนดไม่ให้อินพุตในฟอร์มกรอกข้อมูลได้ควรทำอย่างไร
10. เมื่อฟอร์มยังมีสถานะ INVALID ควรทำให้ปุ่มส่งค่าของฟอร์มไม่ทำงาน ควรกำหนดอ่านอะไรในฟอร์ม HTML

**แบบฝึกหัด**

1. จากการสร้างฟอร์มอาร์เรย์เก็บรายวิชา (courses) เป็นฟอร์มอาร์เรย์ในตัวอย่างในบทนี้ ให้สร้างปุ่ม ลบ ฟอร์มกลุ่มในฟอร์มอาร์เรย์นี้

2. ให้สร้างฟอร์มอาร์เรย์ ที่บรรจุฟอร์มกลุ่ม และในฟอร์มกลุ่มนี้ ให้เพิ่มพืทอย่างอื่น นอกเหนือจาก อินพุทที่เป็น text เช่น มี radio, select โดยสร้างเป็นจากข้อสมมุติของตนเอง
3. ให้สร้าง เช็กบ็อก (checkbox) เพื่อเลือกกิจกรรมที่ทำระหว่างเรียน เช่น ฟุตบอล ดนตรี ค่ายอาสา ผู้แทนนักศึกษา โดยให้สามารถเลือกได้หลายอย่าง หรือไม่ก็เลือกเลยก็ได้

#### อ่านเพิ่มเติม

[1] Angular. Dynamic Form. (15 Apr. 2020). <https://angular.io/guide/forms-overview>