เทมเพลทฟอร์ม

ฟอร์มของ แองกูล่าร์ มีฟอร์มสองประเภทคือ รีเอ็กทีฟฟอร์ม (Reactive form) หรือ ฟอร์มที่มจากโมเดลเป็นหลัก (Model Driven) กับ เทมเพลทฟอร์ม (Template form) หรือที่มีมาจากสร้างเทมเพลทในหน้า HTML เป็นหลัก

ในบทนี้ จะเน้นการทำงานกับ เทมเพลทฟอร์มเป็นหลัก ซึ่งต่อไปจะเรียกสั้นๆ ว่า ฟอร์ม ซึ่งฟอร์มในลักษณะนี้มีใช้ตั้งแต่ยังเป็น AngularJS แล้ว มีลักษณะการใช้งานที่เน้นไปที่หน้า html และผูกข้อมูลในนี้

เทมเพลทใน HTML

ใน HTML สามารถสร้างเทมเพลท ได้ด้วย เครื่องหมายปีกกาคู่ ({{ ... }}) ภายในปีกกาสามารถใส่ ตัวแปรที่มีใน แองกูล่าร์ได้ เช่นมีตัวแปรชื่อ title ก็ใส่ใน เทมเพลท ร่วมกับ HTML เพื่อผูกข้อมูล ดังนี้

```
<h2>Welcome to {{ title }} ! </h2>
```

หรือจะให้ใส่ค่าที่มีการดำเนินการทางคณิตศาสตร์ การบวกเลข

```
<p>1+ 1 = {{ 1+ 1}} </p>
```

หรือจะใส่ผูกกับเหตุการณ์ในฟอร์ม แต่การผูกกับเหตุการณ์ ใส่ได้โดยตรงเหมือนกับภาษา JavaScript

```
<button (click)="submit()"> Submit </button>
```

การผูกข้อมูลที่แนะนำผ่านมา ถือเป็นการผูกข้อมูลทางเดียว ยังมีการผูกข้อมูลสองทาง คือการใช้ฟอร์มโดยเฉพาะ เพราะมีการเปลี่ยนแปลงสองทาง กล่าวคือ เมื่อเปลี่ยนแปลงข้อมูลในฟอร์ม จะมีผลเปลี่ยนแปลงข้อมูลนั้นโดยทันทีกับตัวแปรนั้น การผูกข้อมูลสองทางใช้เครื่องหมาย [(...)] ตัวอย่างการผูกกับฟอร์ม <input> กับตัวแปร name คือ

```
<input [(name)] =  "name">
```

ในการเรียนรู้ต่อไปนี้ ใช้ฟอร์มเป็นหลักในการใช้งานการผูกข้อมูล ทั้งทางเดียว และสองทาง รวมทั้งการผูกข้อมูลกับเหตุการณ์ต่างๆ ประกอบความเข้าใจ

เริ่มสร้างฟอร์ม

การสร้างฟอร์มก็คือการสร้างคอมโพเน้นท์อย่างหนึ่ง ซึ่งได้ชุดของหน้าเว็บ ที่ประกอบไปด้วย ไฟล์ .ts, .html, และ .css ดังที่เคยสร้างมาในก่อนหน้านี้

ในที่นี้ให้สร้างเป็นฟอร์มเพื่อล็อกอินเข้าระบบ ในชื่อคอมโพเน้นท์ว่า LoginForm และสร้างคลาส User เพื่อเป็นข้อมูลที่จับคู่กับฟอร์มนี้ ซึ่งเป็นโมเดล (Model) ของฟอร์มนี้ ให้พิมพ์คำสั่งต่อไปนี้ใน CLI:

```
ng generate component LoginForm
ng generate class User
```

ให้ปรับปรุง ไฟล์ app.component.html เพื่อให้ใส่คอมโพเน้นท์ที่เพิ่งสร้าง (กรณีไม่สร้างเว็บแอปพลิเคชันขึ้นมาใหม่ หรือสร้างสร้างขึ้นมาจากของเก่าในบทที่ผ่านมา) เป็นฟอร์มแรกที่เปิดเว็บนี้

Code 1. src/app/app.component.html

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

สำหรับคลาส User ให้มีสมาชิก id, email, password, และ rememberme ให้สังเกตว่า สมาชิกสุดท้ายใช้เครื่องหมายคำถามต่อท้าย เพื่อเป็นทางเลือกให้สร้างก็ได้หรือไม่สร้างก็ได้ ตามตัวอย่างต่อไปนี้

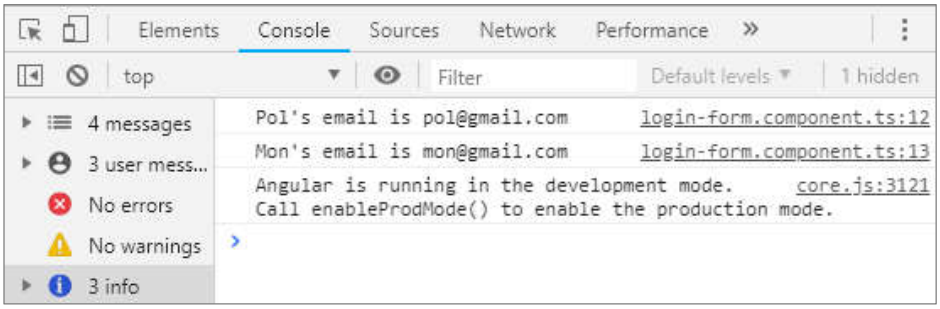
Code 2. src/app/user.ts

```
export class User {
	constructor(
		public email:string,
		public password:string,
		public rememberme?:boolean
	){}
}
```

การสร้างออบเจ็กต์ ของคลาส User สร้างได้สองแบบ คือ สร้างให้ครบทุกสมาชิก หรือเว้นสมาชิกตัวสุดท้าย ให้ทดสอบ ดังตัวอย่างการสร้างทั้งสองออบเจ็กต์ต่อไปนี้ ให้สังเกตว่า ต้องนำเข้า คลาส User มาด้วย ในส่วน import

Code 3. src/app/login-form.component.ts

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

@Component({
  selector: 'app-login-form',
  templateUrl: './login-form.component.html',
  styleUrls: ['./login-form.component.css']
})
export class LoginFormComponent implements OnInit {
  constructor() {
	let pol = new User('pol@gmail.com', '123');
	let mon = new User('mon@gmail.com', '234', true);
	console.log("Pol's email is "+ pol.email);
	console.log("Mon's email is "+ mon.email);
  }
  ngOnInit() {  }
}
```



รูป 1 ผลการสร้างออบเจ็กต์ และดูผลผ่าน info/Console

นำเข้า FormsModule

เพื่อจะใช้โมดูลของฟอร์มนี้ จะต้องปรับปรุงไฟล์โมดูลหลักก่อน (app.module.ts) โดยทำสองขั้นตอน คือ 1) ให้นำเข้า FormsModule และ 2) ประกาศในส่วน อาร์เรย์ของ import ดังตัวอย่างต่อไปนี้

Code 4. src/app/app.module.ts

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

import { AppComponent } from './app.component';
import { LoginFormComponent } from './login-form/login-form.component';

@NgModule({
  declarations: [
    AppComponent, NewUserComponent, LoginFormComponent
  ],
  imports: [
      BrowserModule,
      FormsModule,
      ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

จากตัวอย่างไฟล์นี้ หากเริ่มต้นสร้างแอปฯ ขึ้นมาใหม่ ได้นำเข้า FormsModule ซึ่งต้องปรับปรุงไฟล์นี้สองที่ กรณีที่ใช้สร้างต่อจากบทที่ผ่านมา จะมีส่วนที่เป็น รีเอ็กทีฟฟอร์ม คือ ReactiveFormsModule ซึ่งใช้กับฟอร์ม NewUserComponent ซึ่งจะเก็บไว้ก็ได้ แต่ในบทนี้ไม่ใช้งานอะไรสำหรับคอมโพเน้นท์เก่าที่เคยสร้างมาก่อน

สร้างฟอร์ม ใน HTML

ในฟอร์มในภาษา HTML เป็นฟอร์มทั่วไป ซึ่ง ในที่เลือกใช้ฟอร์ม ที่มาจาก Bootstrap[1] ซึ่งมีการจัดรูปแบบ CSS ดังที่ได้เคยใช้มาแล้ว ใน รีเอ็กทีฟอร์ม (new-user.component.html) ซึ่งหากจำกันได้ว่า ได้กำหนดการใช้ CSS และ

[1] https://getbootstrap.com/docs/4.2/components/forms/ (เข้าถึงเมื่อ 25 มกราคม 62)

JavaScript ของ Bootstrap ที่ไฟล์ index.html ไว้แล้ว หรือจะใช้การนำเข้า ในไฟล์ styles.css ซึ่งรูปแบบการเขียนจะต้องนำเข้า @import ก่อน ดังตัวอย่างต่อไปนี้ (ได้อธิบายไว้ในบทที่ว่าด้วยเรื่อง คอมโพเน้นท์ หัวข้อการใช้งานร่วมกับ Bootstrap)

Code 5. src/styles.css

```
@import
url('https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css');
```

Code 6. src/app/login-form/login-form.component.html

```
<div style="width:80%;margin:10px auto">
<form>
  <div class="form-group">
    <label>Email address</label>
    <input type="email" class="form-control" id="email"
                                        placeholder="Enter email" required>
  </div>
  <div class="form-group">
    <label>Password</label>
    <input type="password" class="form-control" id="password"
                                        placeholder="Password" required>
  </div>
  <div class="form-group form-check">
    <input type="checkbox" class="form-check-input" id="check">
    <label class="form-check-label">Remember me</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
</div>
```

จากฟอร์ม html นี้ยังไม่ได้ กำหนด การผูกข้อมูล แต่มีใส่เงื่อนไข required กับ <input> ของ email และ password ซึ่งหมายถึงจำเป็นต้องกรอกข้อมูล

ผูกข้อมูล

ความน่าใช้งาน แองกูล่าร์ อย่างหนึ่งคือ ความสามารถในการผูกข้อมูลได้สองทาง คือ การผู้ข้อมูลในฟอร์ม กับโมเดล เมื่อเปลี่ยนแปลงข้อมูลในฟอร์ม ข้อมูลในโมเดลก็จะเปลี่ยนแปลงตาม หรือกลับกัน

การผูกข้อมูลอีกแบบ คือ การผูกข้อมูลทางเดียว เพราะไม่จำเป็นที่ผู้ใช้ดำเนินการอะไรได้กับข้อมูลโดยตรง ซึ่งใช้ได้กับ ส่วนที่ไม่เป็นฟอร์ม (ไม่ใช่ <input>)

เพื่อสาธิตการทำงานแบบสองทางนี้ ให้กำหนดโมเดล จากคลาส User ที่ได้เคยทดสอบในก่อนหน้านี้ ให้ทำการสร้างโมเดลนี้ ให้ฟอร์มสามารถมองเห็นได้ และสมาชิก diagnostic( ) โดยการกำหนดเป็นสมาชิกหนึ่งของคอมโพเน้นท์ฟอร์มนี้ ให้มีค่าตั้งต้น มาให้เลย และกำหนดส่วนอื่นๆ ดังตัวอย่างต่อไปนี้ (ปรับปรุงจากไฟล์เดิม)

Code 7. src/app/login-form.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User }    from '../user';
@Component({
  selector: 'app-login-form',
  templateUrl: './login-form.component.html',
  styleUrls: ['./login-form.component.css']
```

```
})
export class LoginFormComponent implements OnInit {
  user = new User('pol@gmail.com', '123',true);
  submitted = false;
  constructor() {
  }

  onSubmit(){this.submitted = true; }

  get diagnostic() { return JSON.stringify(this.user); }

  ngOnInit() {
  }
}
```

หลังจากสร้างโมเดล user แล้ว ให้ทำการผูกข้อมูลไว้กับฟอร์ม HTML ด้วยคำสั่งฝั่ง (directive) [(ngModel)] เช่น [(ngModel)]="user.email"

จากตัวอย่างต่อไปนี้ ให้ผูกข้อมูลสองทางไว้กับทุก <input> กับโมเดล user คือ user.email, user.password, และ user.rememberme

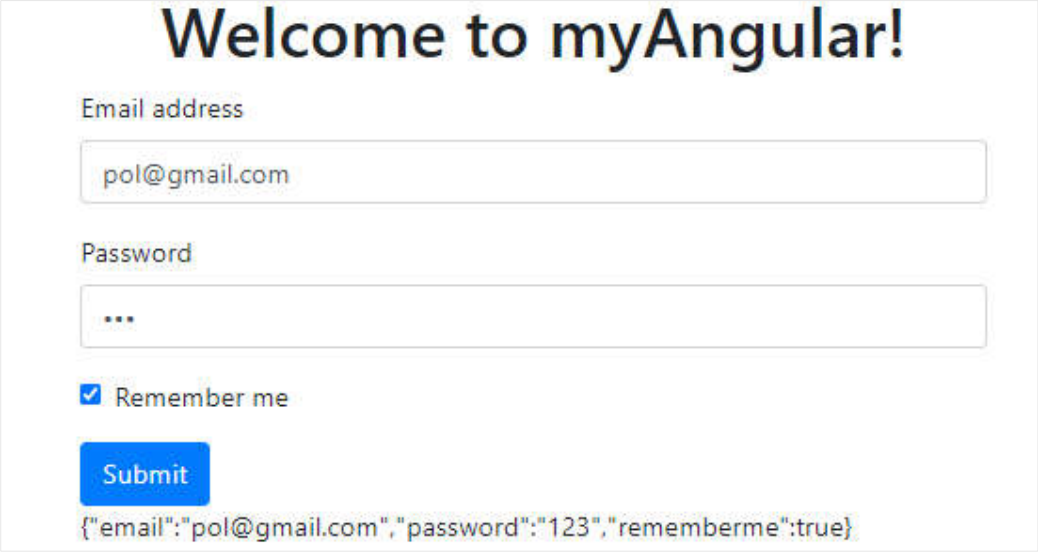
สำหรับการผูกข้อมูลทางเดียว กับสมาชิก diagnostic( ) ใช้สำหรับการผูกข้อมูลทางเดียว ในการแสดงผลอย่างเดียว คือ อ่านข้อมูลโมเดลทั้งหมด (user) ในรูป JSON การผูกข้อมูลใช้เพียงใส่เครื่องหมายปีกกาคู่คล่อมสมาชิกที่ต้องการผูกข้อมูลใน HTML เช่น {{ diagnostic }} สำหรับคำสั่ง JSON.stringify เป็นอ่านข้อมูล JSON ให้ออกมาเป็นรูปแบบอักษรที่แสดงผลได้บนหน้า HTML ไม่ใช่เป็นออบเจ็กต์ JSON แต่อย่างใด

มีข้อสังเกตอย่างหนึ่งว่า การผูกข้อมูลของสมาชิกนี้ ถึงแม้จะเป็นสมาชิกประเภท ฟังก์ชัน แต่ไม่จำเป็นต้องใส่วงเล็บต่อท้าย

Code 8. src/app/login-form/login-form.component.html

```
<div style="width:80%;margin:10px auto">
<form>
  <div class="form-group">
    <label>Email address</label>
    <input   type="email"
             class="form-control" id="email" placeholder="Enter email"
                    required
                    [(ngModel)]="user.email" name='email'>
  </div>
  <div class="form-group">
    <label>Password</label>
    <input   type="password"
             class="form-control" id="password" placeholder="Password"
                    required
                    [(ngModel)]="user.password" name='password'>
  </div>
  <div class="form-group form-check">
    <input   type="checkbox" class="form-check-input" id="check"
                    [(ngModel)]="user.rememberme" name='rememberme'>
    <label class="form-check-label">Remember me</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
```

```
</form>
{{diagnostic}}
</div>
```



รูป 2 ผลการผูกข้อมูลสองทาง และทางเดียว

ผลการผูกข้อมูลตามรูปข้างต้น จะเห็นว่า ข้อมูลในโมเดล ได้ถูกใส่ไว้ในฟอร์มเรียบร้อยแล้ว ตั้งแต่เปิดฟอร์มนี้ขึ้นมา แน่นอนว่า เมื่อเปลี่ยนแปลงข้อมูลใดๆ ในฟอร์ม จะทำให้โมเดล เปลี่ยนแปลงตามไปด้วย เราอาจทดลองเปลี่ยนแปลงค่าในฟอร์ม แล้วจะพบการเปลี่ยนแปลง ในส่วน การผูกข้อมูลทางเดียว (diagnostic)

ยังมีคำสั่งฝั่งกับอิลีเม้นท์ <form> อักตัวหนึ่ง ของ แองกูล่าร์ คือ ใช้ ngForm เพื่ออ้างอิงชื่อฟอร์ม แต่ถึงแม้เราไม่ใส่คำสั่งฝั่งนี้ แองกูล่าร์ ก็ใส่ให้อัตโนมัติ เพียงแต่ไม่มีชื่อฟอร์มในการอ้างอิงเท่านั้น เช่น ถ้าต้องการให้ฟอร์มมีชื่อว่า loginForm ก็จะเขียนดังนี้

```
<form #loginForm="ngForm">
```

ตรวจสอบความเปลี่ยนแปลงของโมเดล

นอกจากความสามารถ ในการผูกข้อมูลสองทางได้แล้ว แองกูล่าร์ ยังสามารถตรวจสอบลำดับการเปลี่ยนแปลงข้อมูลได้ด้วย ด้วยการใส่คำสั่งฝั่งลงใน อิลีเม้นท์ที่ต้องการตรวจสอบการเปลี่ยนแปลงต่าง ๆ ซึ่งต่อไป สามารถใช้งานร่วมกับ CSS เพื่อสร้างรูปแบบแสดงผลรองรับความเปลี่ยนแปลงในลักษณะต่าง ๆ ได้ ดังตารางต่อไปนี้

ตาราง 1 สถานะของคอนโทรล และคำสั่งตรวจสอบการเปลี่ยนแปลง

สถานะ	**ชื่อคลาส ถ้าเป็นจริง**	**ชื่อคลาส ถ้าเป็นเท็จ**
คอนโทรล ได้เคยใช้งานแล้ว	ng-touched	ng-untouched
คอนโทรล มีค่าเปลี่ยนแปลง	ng-dirty	ng-pristine
คอนโทรล มีค่าถูกต้อง	ng-valid	ng-invalid

การอ้างอิงความเปลี่ยนแปลงกับฟอร์ม ให้ใส่คำสั่งอ้างอิงในชื่อต้องการ เช่น ใช้ ชื่อ state เพื่อเป็นชื่อในการอ้างอิงความเปลี่ยนแปลง ของอิลีเม้นท์ <input>

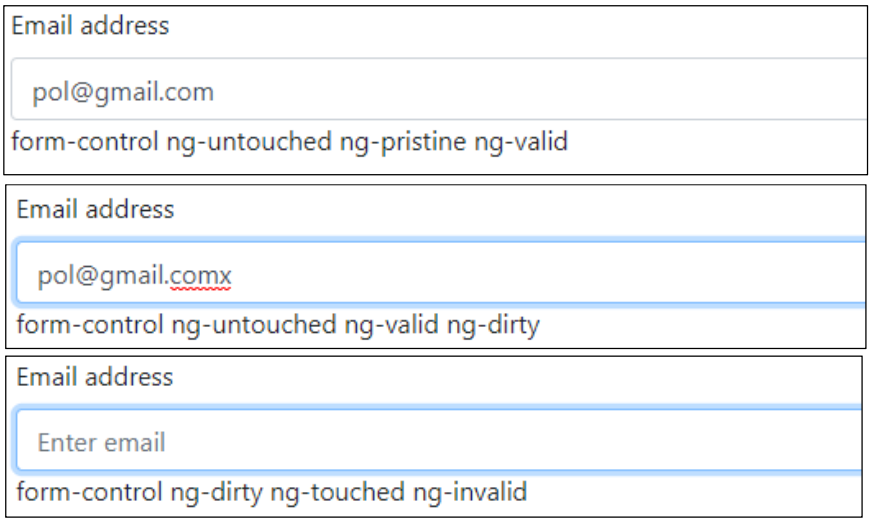
Code 9. `src/app/login-form/login-form.component.html` (บางส่วน)

```
<div class="form-group">
    <label>Email address</label>
    <input   type="email"
              class="form-control" id="emai1" placeholder="Enter email"
                   required
                   [(ngModel)]="user.email" name='email' #state>
                   {{state.className}}
</div>
```

จากตัวอย่างการแก้ไขไฟล์ login-form.component.html นี้ จะเพิ่ม #state และเพิ่มการผูกข้อมูลทางเดียว {{state.className}} เพื่ออ่านชื่อคลาสแห่งการเปลี่ยนแปลง

เมื่อทดสอบโปรแกรม การใช้งาน <input> ที่ได้แก้ไขตามตัวอย่างโปรแกรมข้างต้น จะได้สถานการณ์เปลี่ยนแปลงในส่วนที่ผูกข้อมูล state.className ใน 3 กลุ่มลักษณะ คือ

1. สถานะที่ยังไม่ได้เข้าใช้งาน จะมีชื่อคลาส ng-untouched และเมื่อเข้าไปใช้งาน จะอยู่ในสถานะ ng-touched
2. สถานะที่ยังไม่ได้แก้ไข จะมีชื่อคลาส ng-pristine แต่ถ้ามีการแก้ไขข้อมูล จะได้ชื่อคลาส ng-dirty
3. สถานะข้อมูลที่ถูกต้อง จะได้ชื่อคลาส ng-valid และถ้าข้อมูลไม่ถูกต้อง จะได้ชื่อ ng-invalid



รูป 3 สถานะ ต่าง ๆ ของ <input> รูปบน แสดงสถานะแรกเมื่อยังไม่ใช้งาน
รูปกลาง เป็นสถานะเริ่มมีการแก้ไข และรูปล่าง เมื่อไม่ได้กรองข้อมูลอะไร

การใช้ CSS กับสถานะของฟอร์ม

เมื่อเราทราบสถานะของฟอร์มได้ การใช้ CSS มาช่วยแสดงผลให้เห็นความชัดเจนของความเปลี่ยนแปลงนี้ ในการณ์นี้ จะต้องมีการเขียน CSS กำกับการแสดงผล

การควบคุมสถานะด้วย คลาส CSS มีกันหลายตัวคือ .ng-valid, .ng-invalid, .ng-pending, .ng-pristine, .ng-dirty, .ng-untouched, และ ng-touched ทั้งหมดนี้ก็เป็นไปตามสถานะที่มี ซึ่งต่อไปก็ใช้ CSS ช่วยควบคุมสถานะให้มีรูปแบบ CSS ที่กำหนดเองได้

จาก CSS ต่อไปนี้ ใช้ชื่อคลาส ng-valid และ ng-invalid ให้มีขอบซ้ายหนา 5 หน่วยเป็นสีเขียว และสีแดงตามลำดับ

Code 10. src/assets/forms.css

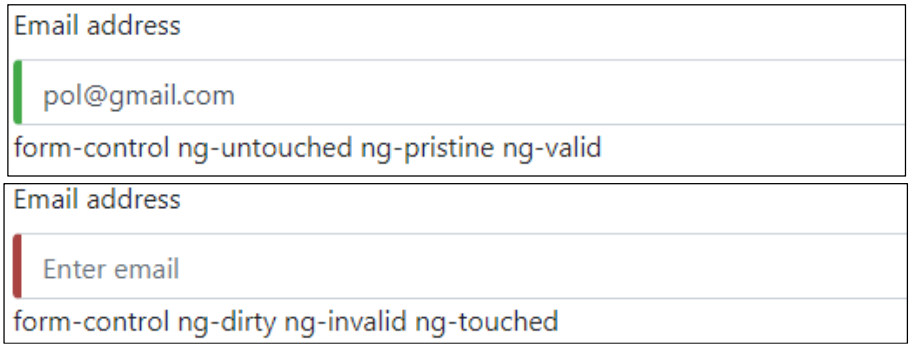
```
.ng-valid[required], .ng-valid.required  {
  border-left: 5px solid #42A948; /* green */
}

.ng-invalid:not(form)  {
  border-left: 5px solid #a94442; /* red */
}
```

จากตัวอย่างนี้ การกำหนด CSS ของ ng-valid กำหนดทั้งในลักษณะ ทางเลือก [required] และคลาสซ้อนคลาส เพื่อความแน่ใจ ให้ทำงานได้ทั้งสองรูปแบบการใช้งาน CSS

เมื่อเขียน ไฟล์ CSS แล้ว จะต้องเขียนการอ้างอิงไฟล์ CSS นี้ด้วย ที่ไฟล์ index.html

Code 11. src/index.html (บางส่วน)

```
<link rel="stylesheet" href="assets/forms.css">
```



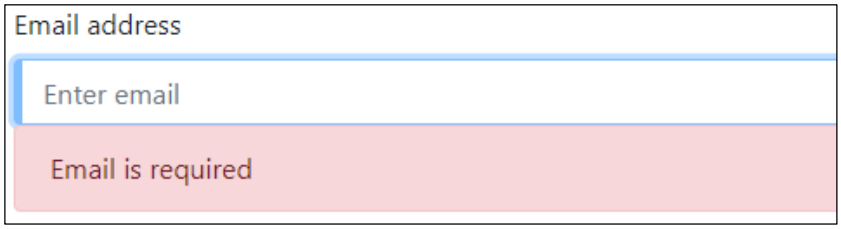
รูป 4 สถานะ ต่าง ๆ ตามที่กำหนดใน CSS ของ <input> รูปบน แสดงสถานะแรกเมื่อยังไม่ใช้งาน และรูปล่าง เมื่อไม่ได้กรองข้อมูลอะไร

ซ่อนหรือแสดงผล ข้อความสถานะของฟอร์ม

ที่ผ่านมาใช้ CSS แสดงรูปแบบของ <input> ในลักษณะต่าง ๆ เช่น การใช้กรอบซ้ายของ <input> เป็นสีแดง เพื่อแสดงความผิดพลาด

การแสดงรูปแบบของฟอร์มเพียงทำเท่าที่ผ่านมา อาจไม่ชัดเจนพอ ว่ามีอะไรผิดพลาด การเพิ่มข้อความเป็นอักษรให้อ่านเข้าใจโดยง่ายจะช่วยทำให้ผู้ใช้งานทราบผลการผิดพลาดได้ดีขึ้น

ตัวอย่างของรูปต่อไปนี้ ใช้ข้อความ “Email is required” ซึ่งแสดงว่า จำเป็นต้องใส่ข้อมูลอีเมลด้วย เป็นกล่องข้อความแสดงด้านล่าง หากผู้ใช้ไม่กรองข้อความอะไรลงไป



รูป 5 การแสดงข้อความผิดพลาด

เพื่อให้มีข้อความแสดงดัง**รูป 5** จะต้องสร้างอิลีเม้นท์ เพิ่มไปด้านล่างของ <input> ที่ต้องการให้ตรวจสอบความผิดพลาด ดังตัวอย่างต่อไปนี้

Code 12. src/app/login-form/login-form.component.html (บางส่วน)

```
<div class="form-group">
      <label>Email address</label>
      <input type="email"
             class="form-control" id="email" placeholder="Enter email"
                  required
                  [(ngModel)]="user.email" name='email' #email="ngModel">
      <div [hidden]="email.valid || email.pristine"
           class="alert alert-danger">
             Email is required
      </div>
</div>
```

ในตัวอย่างนี้ ใช้การชื่ออ้างอิง #email ซึ่งเป็นตัวแปรสำหรับ <input> ที่เป็นเทมเพลตของ แองกูล่าร์ (template reference) ในที่นี้ใช้ชื่อว่า email และให้มีค่าเป็น ngModel

ngModel ซึ่งเป็นคำสั่งฝั่งใน HTML ที่สามารถกำหนดค่าตัวแปรเพิ่มขึ้นได้ เช่น ด้วยการ ระบุ #name_var = “ngModel” โดยให้ name_var เป็นตัวแปรที่มีเพิ่มขึ้นอีกตัวเพื่อใช้เป็นชื่ออ้างอิง **ซึ่งจำเป็นสำหรับใช้ตรวจสอบสถานะ**

นอกจากการกำหนดตัวแปร #email ให้เป็นชื่อที่ใช้อ้างอิงสำหรับ ngModel แล้ว การเพิ่มกล่องข้อความ ที่ต้องการให้ซ่อนข้อมูล หากอยู่ในเงื่อนไข ถูกต้อง (valid) หรือ ยังไม่มีการเปลี่ยนแปลง (pristine) จากตัวอย่างนี้ หากเปิดหน้าเว็บเป็นครั้งแรก ซึ่งยังไม่มีการเปลี่ยนแปลงอะไร หรือ ยังไม่มีข้อความอะไรภายในในรูปแบบอีเมล จะถือว่ามีความผิดพลาด กล่องข้อความจะแสดงให้เห็น

การกำหนดการแสดงผลผลพลาดอาจกำหนดเงื่อนไขต่างกันได้ ตามความต้องการของผู้พัฒนาเลือก บางครั้งผู้พัฒนาต้องการให้แสดงหลังจากที่มีการกรองข้อมูลไปแล้ว จึงจะแสดงผล ลักษณะนี้อยู่ในสถานะ dirty

นอกจากใช้ คุณสมบัติ [hidden] ยังสามารถใช้โลจิก *ngIf แทน การแสดงหรือไม่แสดงก็ได้ เช่น รหัสหผ่านต้องการใช้มีมากกว่า 8 ตัว

Code 13. src/app/login-form/login-form.component.html (บางส่วน)

```
<input type="password"
       class="form-control" id="password" placeholder="Password"
       required minlength="8"
       [(ngModel)]="user.password" name='password' #password='ngModel' >
```

```
<div *ngIf="password.invalid"
      class="alert alert-danger">
      Minlenght is 8.
</div>
```

คำสั่ง รีกูลาร์ ตรวจสอบฟอร์ม

แม้หลายตัวจะมีตัวช่วยตรวจสอบฟอร์มอยู่แล้ว เช่นในฟอร์ม ตัวเลข อีเมล หรือจำนวนอักขระ แต่หากต้องการการตรวจสอบในรูปแบบที่ไม่ให้มา หรือต้องการสร้างตัวตรวจสอบของตนเอง เช่น การตรวจสอบวันที่ ตรวจสอบ เลขบัตรประชาชน รูปแบบรหัสผ่าน หมายเลขโทรศัพท์ รหัสไปรษณีย์ ชื่อไฟล์ และอื่นๆ นอกจากนี้ในบางครั้งตัวตรวจสอบที่มากับฟอร์มยังอาจทำงานได้ไม่ถูกต้องมากนั้น เช่น email แม้จะกรอกผิด ก็ยังถือว่า กรอกผ่านแล้ว

ตัวช่วยตัวหนึ่งที่ทำได้สั้นและใช้กันทั่วไปคือ รีกูลาร์ เอ็กเพรสชัน (Regular expression) หรือเรียกสั้นๆ ว่า เล็คกูล่าร์ ด้วยวิธีการนี้ จะใช้หารูปแบบอักขระให้เป็นไปตามต้องการ การใช้ รีกูล่าร์ มีใช้กันทั่วไปในงานอื่นๆ ที่ไม่จำกัดว่าใช้กับภาษาเขียนโปรแกรมคอมพิวเตอร์ภาษาหนึ่ง

ประวัติโดย่อยของ คำสั่ง รีกูล่าร์

ในปี ค.ศ. 1940 นักคณิตศาสตร์ นาม คลีเน่ (Stephen Kleene) ได้พัฒนารูปภาษามาตรฐาน สำหรับคัดกรองอักษร ที่เขาเรียกว่า เซต รีกูล่าร์ ต่อมา ในช่วงปี ค.ศ. 1960 เขียนนักโปรแกรม นาม ทอมป์สัน (Ken Thompson) ได้นำแนวคิด รีกูล่าร์ มาประยุกต์ใช้ในการค้นหาไฟล์ จนกระทั่งได้รับความนิยมใช้กันเรื่อยมาจนปัจจุบัน

การใช้รีกูล่าร์ จะใช้โดยใน ภาษา TypeScript จะมีคลาส Regex เป็นตัวช่วยให้ทำงานกับคำสั่งรีกูล่าร์ได้ง่ายขึ้น ซึ่งเราอาจจะใช้ตัวช่วยนี้หรือไม่ใช้ก็ได้ เพราะคำสั่งรีกูล่าร์ เป็นคำสั่งที่มีรูปแบบสากล

ดังนั้นการใช้ คำสั่งรีกูล่าร์ สามารถใช้ได้สองแบบคือ

```
let regExp = /^w{3}.[a-z]+.com$/
```

หรือ

```
let regExp = new RegExp("w{3}.[a-z]+.com")
```

ทั้งสองแบบนี้หมายถึงการใช้บังคับให้เป็นรูปแบบ **www.a-z.com** โดย a-z หมายถึงอักษร a - z กี่ตัวรวมกันก็ได้ เช่น www.a.com, www.ab.com, www.aza.com

ในที่นี้จะเน้นเฉพาะคำสั่งรีกูล่าร์ ที่นำไปใช้กับฟอร์ม โดยเลือก รูปแบบที่ใช้งานเพื่อการตรวจสอบอีเมล แต่ก็อธิบายแนวทางการสร้างคำสั่ง เพื่อการประยุกต์ในรูปแบบอื่นๆ ต่อไป

ก่อนจะเข้าสู่รายละเอียดอะไรไปมากของรีกูล่าร์ มาทดสอบแรกกันก่อน ว่าใช้งานกันอย่างไร โดยทดสอบความรูปแบบอีเมลที่ผู้ใช้กรอกถูกหรือไม่

```
ng generate directive emailValidator
```

เมื่อสร้างไฟล์ประเภทไดเร็กทีฟนี้แล้ว ให้ปรับแต่งไฟล์ดังต่อไปนี้

Code 14. src/app/email-validaton.directive.ts

```
import { Directive } from '@angular/core';
import { Validator, NG_VALIDATORS, AbstractControl  } from '@angular/forms';
```

```
@Directive({
  selector: '[emailValidate][ngModel]',
  providers: [{provide: NG_VALIDATORS,
               useExisting: EmailValidatorDirective,
               multi: true}]
})
export class EmailValidatorDirective implements Validator{
  validate(control: AbstractControl): {[key: string]: any}|null {
       let regExp: RegExp =
             /^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$/;

       const valid = regExp.test(control.value);
       console.log(valid);
       return valid?null:{'emailValidate':true};
  }
}
```

สิ่งที่ต้องนำเข้าเพื่อใช้ในการตรวจสอบ มี 3 ตัว คือ Vaidator, NG_VIDATORS, และ AbstractControl ทั้งสามตัวนี้จะทำสฐ้างเป็นฟังก์ชัน validate( )

Vaidator เป็นอินเทอร์เฟส การใช้งานต้องสร้างฟังก์ชัน validate( ) ในที่นี้ใช้เพื่อตรวจสอบกับรูปแบบคำสั่ง รีกูลาร์ ฟังก์ชันนี้จะคืนค่า ค่า null หรือไม่ก็ออบเจ็กต์ผลความผิดพลาด (JSON)

คำสั่งรีกูล่าร์ เริ่มต้น (/^) ด้วยอักษร a-z หรือ A-Z หรือ หรือ _ หรือ - หรือ . โดยมีตั้งแต่หนึ่งตัวขึ้นไป (+) ตามด้วยเครื่องหมาย @ ตามด้วย อักษร a-z หรือ A-Z หรือ _ หรือ - หรือ . ตั้งแต่หนึ่งตัวขึ้นไป (+) ตามด้วย . ต่อด้วย อักษร a-z หรือ A-Z ตั้งแต่ 2 ถึง 5 ตัว และจบด้วย $/ การตรวจสอบใช้ฟังก์ชัน test( ) ผลการตรวจสอบได้เขียนทดสอบด้วย console.log( )

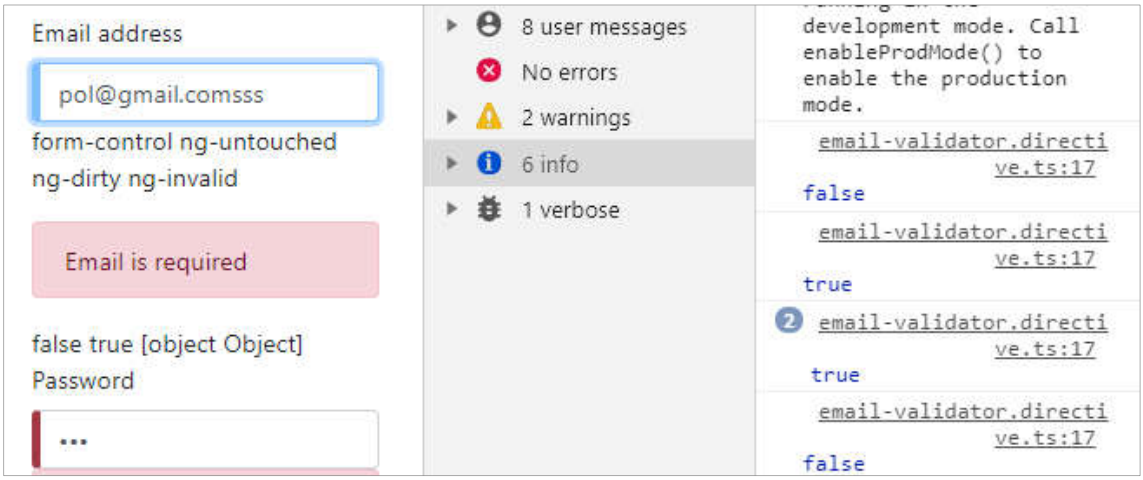
ไดเร็กทีฟนี้ ใช้ชื่อ emailValidate ซึ่งใส่เป็น คุณสมบัติ (attribute) ของ <input> ที่ต้องการจะตรวจสอบ โดยใช้คู่กับ ngModel

เมื่อสร้างฟังก์ชันตรวจสอบเสร็จแล้ว สามารถนำคำสั่งไดเร็กทีฟ emailValidate ไปใช้กับ <input> ได้โดยใส่ต่อจาก required (จำเป็นให้ผู้ใช้ต้องกรองข้อมูล) และใส่ #emailValidate='ngModel' เพื่อใช้เป็นชื่ออ้างอิง

การอ้างอิง emailValidate ซึ่งใช้ต่อไป ในการแสดงหรือไม่แสดง ตามเงื่อนไข *ngIf ที่ผูกกับความไม่ถูกต้อง (invalid) การมีข้อมูลแก้ (dirty) และข้อมูลไม่การเริ่มใช้ (touched)

Code 15. src/app/login-form/login-form.component.html

```
<div class="form-group">
    <label>Email address</label>
    <input type="email"
           class="form-control" id="emai1" placeholder="Enter email"
                    required emailValidate
                    [(ngModel)]="user.email" name='email' #state
                    #emailValidate='ngModel'>
                    {{state.className}}
</div>
<div *ngIf=
       "emailValidate.invalid && (emailValidate.dirty || emailValidate.touched)"
       class="alert alert-danger">
       Email is required
</div>
{{emailValidate.valid}}
```



รูป 6 การแสดงข้อความผิดพลาดของอีเมล

จากรูปนี้ แสดงความผิดพลาด เมื่อ ใส่ comsss ซึ่งไม่เป็นไปตามรูปแบบ เพราะมีเกิด 5 อักษร นอกจากนี้ยังแสดงผลที่ console.log( ) มีค่าสุดท้ายเป็น false

ล้างค่าในฟอร์ม และทำให้ปุ่มส่งฟอร์มไม่ทำงาน

จากเดิม เมื่อไม่มีการกรอกข้อมูล จะมีข้อความเตือน ทั้งที่ เพิ่งเริ่มเข้าหน้าฟอร์มเป็นครั้งแรก ซึ่งควรจะให้ผู้ใช้เริ่มกรอกข้อมูลก่อน หากผิดพลาดอะไรค่อยเตือนที่หลังน่าจะดีกว่า

เมื่อมีการกรอกข้อมูลแล้ว ฟอร์มจะอยู่ในสถานะ dirty และเมื่อคลิกส่งข้อมูลไป ควรจะล้างข้อมูลใหม่ใหม่ (ไม่มีค่าใดใดในฟอร์ม) เบอร์เซอร์ จะถือว่า ยังเป็น สถานะ dirty อยู่ กล่องข้อความเตือนก็จะไม่แสดง

นอกจากนี้หาก ฟอร์มยังอยู่ในสถานะที่ไม่ถูกต้อง ปุ่มส่งของฟอร์มก็ไม่ควรจะให้คลิกได้ (disable)

วิธีการแก้ไขใหม่ สมมุติให้ ให้ทำงานเรียกฟังก์ชัน login( ) ต่อด้วย การล้างค่าใหม่ และ ตรวจสอบความถูกต้องของฟอร์ม หากยังไม่ถูกต้องให้ปุ่มส่งข้อมูลของฟอร์มยังทำงานไม่ได้ ดังตัวอย่างต่อไปนี้

Code 16. src/app/login-form/login-form.component.html (บางส่วน)

```
<button type="submit" class="btn btn-primary"
        (click)="login(); loginForm.reset()"
        [disabled]="!loginForm.form.valid">Submit</button>
```

ส่วนสำคัญอีกอย่างหนึ่ง การส่งฟอร์มจะต้องส่งไปยังปลายทางได้ จะต้องมี คำสั่ง ngSubmit และผูกกับฟังก์ชันใดตัวหนึ่ง เช่น onSubmit( ) แต่ฟังก์ชันนี้ยัง ไม่มีการส่งไปยังไปปลายทางใด นอกจาก การหนดค่าบางอย่าง ในตัวอย่างต่อไปนี้

Code 17. src/app/login-form/login-form.component.html (บางส่วน)

```
<form #loginForm="ngForm" (ngSubmit)="onSubmit()">
```

ที่เหลือก็เพียงแต่เขียนฟังก์ชัน ต่าง ๆ ที่กำหนดในฟอร์มต้องการ คือฟังก์ชัน onSubmit( ) และ login( ) ดังตัวอย่างต่อไปนี้

Code 18. src/app/login-form.component.ts (บางส่วน)

```
onSubmit(){this.submitted = true; }
login(){ }
```

สรุป

ด้วยการนำเข้า FormsModule เพียงตัวเดียว ที่ใช้ในการสร้างฟอร์มในลักษณะเทมเพลต บทนี้มีหลายอย่างในการดำเนินการสร้างฟอร์ม ที่สนับสนุนใช้การผูกข้อมูลในสองทาง การตรวจสอบความถูกต้อง และอื่น ๆ คือ

- การสร้างฟอร์มเป็นเทมเพลตในไฟล์ HTML
- การสร้างคอมโพเน้นท์ฟอร์มในไฟล์ TS
- การผูกข้อมูลสองทางด้วย [(ngModel)]
- การใช้ชื่อภายในอิลีเม้นท์ <input> ด้วย #
- การอ้างอิงสถานะของฟอร์ม ในสถานะ (ng-touched, ng-untouched), (ng-valid, ng-invalid), (ng-dirty, ng-prestine)
- การสร้างรูปแบบ CSS ให้กับสถานะต่าง ๆ ของฟอร์ม
- การสร้างคำสั่งไดเร็กทีฟเพื่อตรวจสอบความผิดพลาด ทั้งแบบที่มีมาให้ และสร้างขึ้นเอง
- การซ่อนปุ่มส่งข้อมูลฟอร์มด้วย [disabled] และการสร้างเหตุการณ์ให้กับปุ่ม ด้วย (click)
- การล้างค่าในฟอร์ม ด้วย form_name.reset( )

อ้างอิง/อ่านเพิ่มเติม

- AngularDart. Template Syntax 5.3.(June 9, 2020). https://angulardart.dev/guide/template-syntax
- Radic, Z. Custom Form Validators in Angular 6 Using Regular Expressions. Medium. (June 8, 2020). https://medium.com/@zeljkoradic/custom-form-validators-in-angular-6-using-regular-expressions-d421a5655adf
- TypeScript.NET. Class Regex. (June 8, 2020). http://electricessence.github.io/TypeScript.NET/documentation/classes/_source_system_text_regularexpressions_.regex.html