**Angular-Form( part 2)**

**2. Reactive Forms Trong Angular**

Thuật ngữ **Reactive Forms** hay còn được gọi là **Model-Driven Forms**, là một phương pháp để tạo form trong Angular, phương pháp này tránh việc sử dụng các directive ví dụ như ngModel, required, etc, thay vào đó tạo các Object Model ở trong các Component, rồi tạo ra form từ chúng.  
Một điều lưu ý đó là Template-Driven là async còn Reactive là sync.

Trong Reactive forms, chúng ta tạo toàn bộ form control tree ở trong code (khởi tạo ngay, khởi tạo trong constructor, hoặc khởi tạo trong ngOnInit), nên có thể dễ dàng truy cập các phần tử của form ngay tức thì.

Trong Template-driven forms, chúng ta ủy thác việc tạo form control cho directives, để tránh bị lỗi changed after checked, directives cần một cycle nữa để build toàn bộ form control tree. Vậy nên bạn cần đợi một tick nữa để có thể truy cập vào các phẩn tử của form. Chính điều này khiến việc test template-driven form trở nên phức tạp hơn.

Bạn có thể tham khảo thêm tại document sau: <https://angular.io/docs/ts/latest/guide/reactive-forms.html#!#async-vs-sync>

**2.1 Các thành phần cơ bản của form**

* AbstractControl là một abstract class cho 3 lớp con form control: FormControl, FormGroup, và FormArray. Nó cung cấp các hành vi, thuộc tính chung cho các lớp con.
* FormControl là đơn vị nhỏ nhất, nó lưu giữ giá trị và trạng thái hợp lệ của một form control. Tương ứng với một HTML form control như input, select.
* FormGroup nó lưu giữ giá trị và trạng thái hợp lệ của một nhóm các đối tượng thuộc AbstractControl – có thể là FormControl, FormGroup, hay FormArray – đây là một dạng composite. Ở level cao nhất của một form trong component của bạn là một FormGroup.
* FormArray nó lưu giữ giá trị và trạng thái hợp lệ của một mảng các đối tượng thuộc AbstractControl giống như FormGroup. Nó cũng là một dạng composite. Nhưng nó không phải là thành phần ở level cao nhất.

**2.2 Template**

Cũng giống như trong [Template-driven forms](http://www.tiepphan.com/thu-nghiem-voi-angular-template-driven-forms-trong-angular/), chúng ta sẽ sử dụng một template giống thế.

<form novalidate (ngSubmit)="onSubmit()"

class="row justify-content-md-center">

<div class="col-md-8">

<div class="form-group row">

<label for="example-text-input" class="col-2 col-form-label">

Name:

</label>

<div class="col-10">

<input class="form-control" type="text"

id="example-text-input">

</div>

</div>

<div class="form-group row">

<label for="example-email-input" class="col-2 col-form-label">

Email:

</label>

<div class="col-10">

<input class="form-control" type="email"

id="example-email-input">

</div>

</div>

<div class="form-group row">

<label for="example-url-fb" class="col-2 col-form-label">

Facebook:

</label>

<div class="col-10">

<input class="form-control" type="url" id="example-url-fb">

</div>

</div>

<div class="form-group row">

<label for="example-url-twt" class="col-2 col-form-label">

Twitter:

</label>

<div class="col-10">

<input class="form-control" type="url" id="example-url-twt">

</div>

</div>

<div class="form-group row">

<label for="example-url-web" class="col-2 col-form-label">

Website:

</label>

<div class="col-10">

<input class="form-control" type="url" id="example-url-web">

</div>

</div>

<div class="form-group row">

<label for="example-tel-input" class="col-2 col-form-label">

Tel:

</label>

<div class="col-10">

<input class="form-control" type="tel"

id="example-tel-input">

</div>

</div>

<div class="form-group row">

<div class="col-10 offset-2">

<button class="btn btn-primary"

type="submit">Submit</button>

</div>

</div>

</div>

</form>

Trên đây chỉ là một form HTML thông thường, khi browser render chúng ta sẽ có form trông giống như sau:

Nhưng trong template trên chúng ta mặc định sử dụng ngSubmit, các bạn có thể trở lại bài trước để hiểu về ngSubmit vs submit event.

**2.3 Import APIs cho Reactive forms**

Để có thể sử dụng các APIs mà Angular cung cấp cho việc thao tác với Reactive forms, chúng ta cần import NgModule là ReactiveFormsModule từ package @angular/forms như sau:

Lưu ý rằng bạn có thể import cả Reactive form và Template-driven form modules.

import { ReactiveFormsModule } from '@angular/forms';

@NgModule({

declarations: [...],

imports: [

...,

ReactiveFormsModule

],

providers: [...],

bootstrap: [...]

})

export class AppModule { }

**2.4 Khởi tạo form trong Component**

Như đã nói ở phần trước, chúng ta có thể khởi tạo form trong Component ở một trong 3 giai đoạn: khởi tạo ngay lúc khai báo property, trong constructor, trong ngOnInit. Để thống nhất về code, chúng ta sẽ khởi tạo trong ngOnInit.

Giả sử chúng ta tạo một component như sau:

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

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

@Component({

selector: 'tpc-contact-reactive-form',

templateUrl: './contact-reactive-form.component.html',

styleUrls: ['./contact-reactive-form.component.scss']

})

export class ContactReactiveFormComponent implements OnInit {

rfContact: FormGroup;

constructor() { }

ngOnInit() {

this.rfContact = new FormGroup({

contactName: new FormControl(),

email: new FormControl(),

social: new FormGroup({

facebook: new FormControl(),

twitter: new FormControl(),

website: new FormControl()

}),

tel: new FormControl()

});

}

onSubmit() {

// Do something awesome

console.log(this.rfContact);

}

}

**2.5 Binding controls**

Ở top-level (form), chúng ta sẽ bind một formGroup như sau:

<form novalidate (ngSubmit)="onSubmit()" [formGroup]="rfContact"

class="row justify-content-md-center">

</form>

Lần lượt với các FormControl instances, chúng ta sẽ bind với property formControlName như sau:

<input class="form-control" type="text" id="example-text-input"

formControlName="contactName">

Đối với formGroup lồng trong formGroup như cấu trúc ở trên, chúng ta có thêm một property cần bind là formGroupName:

<fieldset formGroupName="social">

<div class="form-group row">

<label for="example-url-fb" class="col-md-2 col-form-label">Facebook:</label>

<div class="col-md-10">

<input class="form-control" type="url" id="example-url-fb" formControlName="facebook">

</div>

</div>

<div class="form-group row">

<label for="example-url-twt" class="col-md-2 col-form-label">Twitter:</label>

<div class="col-md-10">

<input class="form-control" type="url" id="example-url-twt" formControlName="twitter">

</div>

</div>

<div class="form-group row">

<label for="example-url-web" class="col-md-2 col-form-label">Website:</label>

<div class="col-md-10">

<input class="form-control" type="url" id="example-url-web" formControlName="website">

</div>

</div>

</fieldset>

Việc binding property vào template cần map cấu trúc cây DOM giống với cấu trúc của model ở component, bạn không thể binding một property không thuộc về một control, chẳng hạn ở model trên, control facebook thuộc về group social, nên bạn không thể bind control này trực tiếp vào group rfContact.

**2.6 Reactive Forms Validator**

Việc thêm Validator vào cho một control rất đơn giản, việc của bạn là thêm giá trị tham số tiếp theo như sau:

Lưu ý: có 2 loại validators, một là validator validator đồng bộ (sync), ví dụ required, minlenghth, etc. Hai là async validator (validator bất đông bộ) chẳng hạn như validator để xem user tồn tại chưa, lúc này bạn gọi AJAX để thực hiện tìm kiếm và đó là một quá trình xử lý bất đồng bộ. Các class FormControl và FormGroup cho phép thêm validator theo thứ tự sync – async vào danh sách tham số của constructor.

export declare class FormGroup extends AbstractControl {

constructor(controls: {

[key: string]: AbstractControl;

}, validator?: ValidatorFn | null,

asyncValidator?: AsyncValidatorFn | null);

}

Tương tự cho FormArray.

Và FormControl:

export declare class FormControl extends AbstractControl {

constructor(formState?: any,

validator?: ValidatorFn | ValidatorFn[] | null,

asyncValidator?: AsyncValidatorFn | AsyncValidatorFn[] | null);

}

Giả sử chúng ta thêm required cho form control contactName ở trên, chúng ta chỉ cần thêm vào constructor như sau:

contactName: new FormControl('', Validators.required)

Với Validators.required là một validator function.

Hoặc có thể thêm nhiều validator với Validators.compose, hoặc truyền vào là một mảng các validator functions:

contactName: new FormControl('',

[Validators.required, Validators.minLength(3)])

Ở đoạn code trên, chúng ta gộp required và minLength(3) cho control contactName, hoặc bạn có thể thêm nhiều các validator function nữa theo ý muốn.

Để hiển thị thông báo cho người dùng biết về lỗi, chúng ta có thể sử dụng phương pháp như đã đề cập trong phần Template-driven form.

<div class="col alert alert-danger" role="alert"

\*ngIf="rfContact.controls.contactName?.errors?.required

&& rfContact.controls.contactName?.touched">

Name is required!

</div>

Hoặc sử dụng [hasError](https://github.com/angular/angular/blob/4.1.x/packages/forms/src/model.ts#L488):

<div class="col alert alert-danger" role="alert"

\*ngIf="rfContact.controls.contactName?.hasError('required')

&& rfContact.controls.contactName?.touched">

Name is required!

</div>

Không những thế, Reactive form còn cho phép chúng ta lấy ra một control theo path đến nó một cách dễ dàng, ví dụ để lấy về contactName:

rfContact.get('contactName')

Hoặc với control facebook:

rfContact.get('social.facebook')

// or

rfContact.get(['social', 'facebook'])

Với phương pháp này chúng ta không cần tạo nhiều template variable nữa.

**2.7 Form Builder**

Một trong những tính năng tuyệt vời của Angular form là Form Builder, nó giúp bạn tạo form model một cách nhanh chóng và tiện lợi. Việc bạn phải làm là inject class FormBuilder vào class bạn muốn tạo form, và gọi các API như group, array, control để tạo form.

constructor(private fb: FormBuilder) { }

ngOnInit() {

this.rfContact = this.fb.group({

contactName: this.fb.control('', [Validators.required, Validators.minLength(3)]),

email: this.fb.control(''),

social: this.fb.group({

facebook: this.fb.control('', [Validators.required, Validators.minLength(3)]),

twitter: this.fb.control(''),

website: this.fb.control('')

}),

tel: this.fb.control('')

});

}

Hoặc một cách rút gọn hơn nữa:

this.rfContact = this.fb.group({

contactName: ['', [Validators.required, Validators.minLength(3)]],

email: '',

social: this.fb.group({

facebook: ['', [Validators.required, Validators.minLength(3)]],

twitter: '',

website: ''

}),

tel: ''

});

Wow, quá tuyệt vời phải không nào.

**2.8 FormArray**

Giả sử bây giờ bạn muốn người dùng có thể nhập vào một hoặc nhiều số điện thoại. Vậy có cách nào tạo form với số lượng thay đổi như thế hay không. Rất may cho chúng ta, Reactive form có một loại control là FormArray, nó sẽ giúp chúng ta làm được việc đó, bây giờ hãy thay đổi code một chút.

ngOnInit() {

this.rfContact = this.fb.group({

// ...

tels: this.fb.array([

this.fb.control('')

])

});

}

get tels(): FormArray {

return this.rfContact.get('tels') as FormArray;

}

addTel() {

this.tels.push(this.fb.control(''));

}

removeTel(index: number) {

this.tels.removeAt(index);

}

Và template của chúng ta sẽ có:

<div formArrayName="tels" \*ngIf="tels.controls.length">

<div class="form-group row" \*ngFor="let c of tels.controls; index as i">

<label for="example-tel-input" class="col-md-2 col-form-label">Tel:</label>

<div class="col-md-10">

<input class="form-control" type="tel" id="example-tel-input" [formControlName]="i">

<div class="text-right">

<button class="btn btn-info" (click)="addTel()">+</button>

<button class="btn btn-danger" (click)="removeTel(i)" \*ngIf="tels.controls.length > 1">-</button>

</div>

</div>

</div>

</div>

Khi render chúng ta sẽ có form có dạng:

Ở đoạn code trên chúng ta đã tạo ra một FormArray instance và khi binding vào template chúng ta thông báo nó với directive formArrayName.  
Khi thực hiện việc lặp, chúng ta tạo ra biến index có tên là "i", với mỗi biến index như thế, Angular sẽ lưu trữ tương ứng với một phần tử trong FormArray là một AbstractControl instance, trong trường hợp này của chúng ta là một FormControl instance, vậy nên chúng ta có đoạn binding property như sau: [formControlName]="i".

FormArray cung cấp một số phương thức cho phép chúng ta thêm, xóa phần tử trong array như insert, push, removeAt. Hay phương thức at để lấy ra phần tử ở vị trí cụ thể. Chúng sẽ có ích khi bạn sử dụng trong ứng dụng của mình – trong trường hợp ở trên chúng ta có dùng để thêm hoặc xóa.

**2.9 Forms with a single control**

Giả sử bạn có một form search chẳng hạn, bạn chỉ có một form control, lúc này bạn không muốn phải bao cả form và form group, có cách nào làm được thế không.

Đây là lúc formControl directive phát huy tác dụng. Chúng ta hãy xem xét search form sau đây.

<input type="search" class="form-control" [formControl]="searchControl">

Còn đây là component code:

export class SearchComponent implements OnInit {

searchControl = new FormControl();

ngOnInit() {

this.searchControl.valueChanges.subscribe(value => {

// do search with value here

console.log(value);

});

}

}

Giờ đây, bạn có thể tạo ra một form seach cực kỳ nhanh và đơn giản phải không nào.

**2.10 Cập nhật giá trị cho form, control**

Có 2 phương thức để cập nhật giá trị cho form control được mô tả bởi class AbstractControl là setValue và patchValue. Chúng là các abstract method, vậy nên các class dẫn xuất sẽ phải implement riêng cho chúng.

Đối với class FormControl, không có gì khác biệt giữa 2 phương thức – thực chất patchValue gọi lại setValue.

Đối với các class FormGroup và FormArray, patchValue sẽ cập nhật các giá trị được khai báo tương ứng trong object value truyền vào. Nhưng setValue sẽ báo lỗi nếu một control nào bị thiếu.

Vậy nên nếu bạn muốn cập nhật một phần của form thì hãy dùng patchValue, nếu bạn muốn set lại tất cả và đảm bảo không cái nào bị thiếu thì dùng setValue để tận dụng việc báo lỗi của nó.

Ngoài ra, còn có phương thức reset để bạn có thể reset lại trạng thái lúc khởi tạo của form hoặc control.