**ANGULAR-FORM ( part 1)**

**Thử Nghiệm Với Angular – Forms Trong Angular**

Hầu hết các ứng dụng web hiện đại đều làm việc với forms để thu thập dữ liệu từ người dùng. Angular cung cấp cho chúng ta hai phương pháp để tạo forms, một là Template-driven forms (mà có thể bạn đã quen thuộc từ Angularjs) và hai là Reactive forms hay Model-driven forms.

Trong bài này chúng ta sẽ cùng tìm hiểu cách tạo forms trong Angular, và để bắt đầu chúng ta sẽ tìm hiểu Template-driven forms.

**1. Template-Driven Forms Trong Angular**

Template-driven forms là phương pháp mà chúng ta sẽ tạo forms dựa vào template (giống như trong Angularjs). Chúng ta thực hiện việc thêm các directives và hành vi vào template, sau đó Angular sẽ tự động tạo forms để quản lý và sử dụng.

**1.1 Template**

Giả sử chúng ta có template form như sau:

<form novalidate (submit)="onSubmit()" class="row justify-content-md-center">

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

<div class="form-group row">

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

<div class="col-md-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-md-2 col-form-label">Email:</label>

<div class="col-md-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-md-2 col-form-label">Facebook:</label>

<div class="col-md-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-md-2 col-form-label">Twitter:</label>

<div class="col-md-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-md-2 col-form-label">Website:</label>

<div class="col-md-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-md-2 col-form-label">Tel:</label>

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

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

</div>

</div>

<div class="form-group row">

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

<button class="btn btn-primary" type="submit">Submit</button>

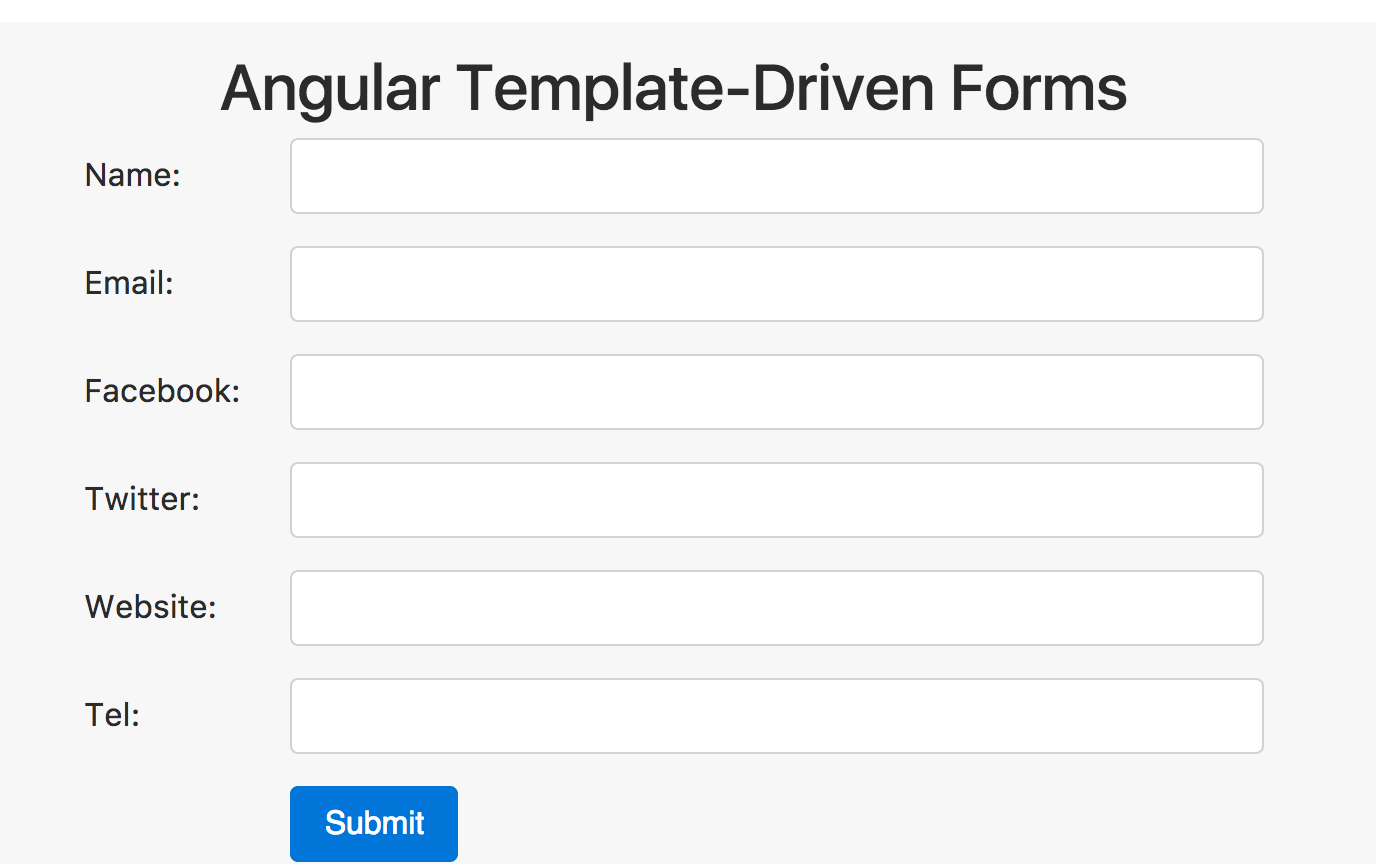
</div>

</div>

</div>

</form>

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

[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/qqzz1dbvi1_ContactAppForm.png)

**1.2 Import APIs cho Template-driven forms**

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

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

@NgModule({

declarations: [...],

imports: [

...,

FormsModule

],

providers: [...],

bootstrap: [...]

})

export class AppModule { }

**1.3 ngForm và ngModel directives**

Nhiệm vụ đầu tiên chúng ta cần làm là truy cập vào form instance và gán các control vào form. Chúng ta sẽ lần lượt sử dụng ngForm và ngModel directives như sau:

<form novalidate #form="ngForm" ...>

</form>

Angular cung cấp một giải pháp để có thể truy cập được directive/component instance ở trong template của component bằng cách sử dụng exportAs trong khai báo directive/component metadata.

Ở trong đoạn code phía trên, chúng ta đã tạo một template variable là form, nó sẽ là một instance của directive ngForm, như thế chúng ta có thể sử dụng các public API mà directive này cung cấp như lấy ra value của nó chẳng hạn như form.value.

Giờ đây chúng ta có thể sử dụng form value cho việc submit form chẳng hạn.

<form novalidate #form="ngForm"

(submit)="onSubmit(form.value)" ...>

</form>

<p>Form value:</p>

<pre>{{ form.value | json }}</pre>

Và để dễ dàng trong quá trình development, chúng ta có thể thêm một phần hiển thị ở template để biết được form value đang có gì như hai dòng code cuối ở trên.

Tại thời điểm này, cho dù chúng ta có form control ở template nhưng Angular không thể biết cái nào cần quản lý nên chúng ta chỉ nhận được một object rỗng.

Bây giờ công việc tiếp theo là chúng ta phải nói cho Angular biết các form control nào cần phải quản lý. Đây chính là lúc chúng ta dùng đến ngModel directive.

Chúng ta sẽ thêm ngModel vào các control như sau:

<input class="form-control" type="text" ngModel ...>

Nhưng nếu bạn không khai báo attribute name cho form control, bạn sẽ gặp phải một lỗi giống như sau:

Error: If ngModel is used within a form tag, either the name attribute must be set or the form control must be defined as ‘standalone’ in ngModelOptions.

Kèm với đó là bạn sẽ có các ví dụ để sửa lỗi trên.

OK, chúng ta cần thêm một số config để Angular biết cách tạo ra form control của nó để quản lý. Và chúng ta sẽ thêm attribute name cho các form control ở template trên.

<input class="form-control" type="text" ngModel name="contact-name" ...>

Bây giờ quan sát form value chúng ta sẽ có một object có key contact-name chẳng hạn:

{

"contact-name": ""

}

Nếu bạn quen với camel case, chúng ta có thể sửa đổi chút để object của chúng ta có key nhìn quen thuộc hơn chẳng hạn.

<input class="form-control" type="text" ngModel name="contactName" ...>

Kết quả nhận được:

{

"contactName": ""

}

OK cool, giờ chúng ta có thể cài đặt tương tự cho các phần tử khác của form.

Và khi bạn nhập giá trị cho các control thì Angular sẽ tự cập nhật cho các control của form tương ứng, chẳng hạn sau khi nhập xong và submit thì form sẽ có value như sau:

{

"contactName": "Tiep Phan",

"email": "abc@deg.com",

"facebook": "facebook.com",

"twitter": "twitter.com",

"website": "tiepphan.com",

"tel": "1234-5678-90"

}

Bây giờ có một tình huống phát sinh là bạn cần bind data cho các control với một dữ liệu có sẵn, lúc này chúng ta sẽ dùng đến binding cho property, và property chúng ta nhắc đến ở đây chính là ngModel.

Chúng ta có dạng binding quen thuộc như sau:

[ngModel]="obj.prop"

Giả sử object mà chúng ta có ở đây có dạng:

contact = {

"contactName": "Tiep Phan",

"email": "abc@deg.com",

"facebook": "facebook.com",

"twitter": "twitter.com",

"website": "tiepphan.com",

"tel": "1234-5678-90"

}

Template của chúng ta sẽ thay đổi như sau:

<input [ngModel]="contact.contactName" name="contactName" class="form-control" type="text" ...>

Mọi thứ đều bắt nguồn từ những điều cơ bản nhất, [ngModel] chính là one-way binding mà chúng ta vẫn thường dùng.

Lưu ý rằng, khi bạn update form control, bản thân control được form quản lý sẽ thay đổi – form.value, nhưng object contact ở trên sẽ không hề hấn gì, vì chúng ta không hề đụng chạm gì tới nó, chúng ta chỉ binding một chiều, mà không binding ngược trở lại. Điều này dẫn đến chúng ta có thêm một dạng khác của ngModel đó là cú pháp two-way binding [(ngModel)].

<input [(ngModel)]="contact.contactName" name="contactName" class="form-control" type="text" ...>

Như vậy, nếu bạn không cần binding thì chỉ cần thêm ngModel là đủ, nếu bạn muốn one-way binding thì sử dung [ngModel], cuối cùng là muốn dùng two-way binding với [(ngModel)]. Mặc dù vậy, khi bạn thay đổi form control value, thì Angular sẽ cập nhật lại giá trị của các control của form mà nó đang quản lý.

**1.4 ngModelGroup directive**

Đến lúc này, chúng ta vẫn đang chỉ quản lý form control với một object chứa tất cả các keys cần thiết, vậy làm thế nào chúng ta có thể gom nhóm một số key lại thành một group riêng, câu trả lời là ngModelGroup. Directive này tạo ra một group lồng vào group cha, giống như object nằm trong một object khác.

Giả sử như template kể trên, chúng ta sẽ nhóm các url thành một nhóm có tên là social chẳng hạn:

<fieldset ngModelGroup="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"

ngModel name="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"

ngModel name="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"

ngModel name="website">

</div>

</div>

</fieldset>

Kết quả thu được chúng ta có form value với cấu trúc:

{

"contactName": "",

"email": "",

"social": {

"facebook": "",

"twitter": "",

"website": ""

},

"tel": ""

}

**1.5 Submit form**

Lưu ý rằng phần nội dung này có thể áp dụng cho cả Reactive forms mà chúng ta sẽ tìm hiểu tiếp theo.

Ở phần trước, chúng ta đã listen event submit của form, nhưng ngoài ra, còn một event khác cũng được fired ra khi thực hiện submit form, đó là ngSubmit. Vậy có điều gì khác biệt giữa submit và ngSubmit?

Giống như submit, event ngSubmit cũng thực hiện hành động khi form thực hiện submit – người dùng nhấn vào button submit chẳng hạn. Nhưng ngSubmit sẽ thêm một số nhiệm vụ để đảm bảo form của bạn không thực hiện submit form theo cách thông thường – tải lại trang sau khi submit.

Giả sử, chúng ta thực hiện một tác vụ nào đó trong hàm listen form submit mà sinh ra exception, lúc này nếu bạn sử dụng submit, trang web của bạn sẽ reload, còn nếu bạn sử dụng ngSubmit, nó sẽ không reload – phiên bản lúc này tôi đang sử dụng.

onSubmit(formValue) {

// Do something awesome

console.log(formValue);

throw Error('something go wrong');

}

Lời khuyên dành cho bạn là nên dùng ngSubmit cho việc listen form submit.

**1.6 Template-driven error validation**

– Huh, có vẻ như user nhập sai thông tin rồi, làm sao tôi có thể hiển thị thông báo cho họ biết để sửa đây?

– Bình tĩnh, Angular đã có sẵn tính năng cơ bản cho việc validation của bạn. Bây giờ hãy bắt đầu với việc kiểm tra lỗi và cảnh báo.

Angular cung cấp một số Validators cơ bản mà bạn có thể dùng ngay trong template như: required, minlength, maxlength, pattern và từ Angular v4 trở đi có thêm email. Chúng được viết là các directives, nên bạn có thể sử dụng như các directives khác trong template của bạn.

Chúng ta sẽ bỏ qua validation của HTML5, vậy nên ngay từ đầu form chúng ta đã thêm novalidate attribute vào khai báo form, thay vào đó là sử dụng Angular validation.

Giả sử chúng ta cần cài đặt contactName là required, chúng ta sẽ đặt như sau:

<input ngModel name="contactName" required class="form-control" type="text" ...>

Và để dễ dàng quan sát, chúng ta sẽ thêm phần hiển thị lỗi như sau:

<p>Form contactName errors:</p>

<pre>{{ form.controls.contactName?.errors | json }}</pre>

Chúng ta sử dụng **safe navigation operator** để truy cập property của một object có thể bị null/undefined mà không gây ra lỗi chương trình.

Khi không nhập gì vào input contactName, chúng ta có thể thấy một thông báo lỗi như sau:

{

"required": true

}

Khi input này được nhập dữ liệu thì chúng ta sẽ thấy key required của object trên sẽ bị xóa bỏ.

Công việc của chúng ta bây giờ là sử dụng ngIf chẳng hạn để show/hide error cho người dùng được biết.

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

\*ngIf="form.controls.contactName?.errors?.required">

Name is required!

</div>

Và có thể thêm việc không cho người dùng nhấn button submit khi trạng thái của form là invalid như sau:

<button class="btn btn-primary" type="submit"

[disabled]="form.invalid">

Submit

</button>

Bây giờ, nếu bạn muốn chỉ hiển thị thông báo error khi người dùng đã focus vào input đó mà không nhập gì, lúc này chúng ta có thể thông qua trạng thái của form bằng cách truy cập các propeties như touched, dirty hay pristine, …

Trong đó:

* touched: true nếu người dùng đã focus vào input rồi không focus vào nữa.
* untouched: true nếu người dùng chưa đụng chạm gì hoặc lần đầu tiên focus và chưa bị mất focus (ngược lại với touched)
* dirty: true nếu người dùng đã tương tác với control – nhập một ký tự vào input text chẳng hạn.
* pristine: true nếu người dùng chưa tương tác gì với control, mặc dù có thể đã touched, nhưng chưa sửa đổi gì.

Chúng ta sẽ thay đổi một chút form validation:

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

\*ngIf="form.controls.contactName?.errors?.required && form.controls.contactName?.touched">

Name is required!

</div>

OK cool, giờ đây chỉ khi nào người dùng touched vào control và có error thì validation message sẽ hiển thị.

– Huh, giờ chúng ta muốn dùng template variable để truy cập control thay vì truy cập dài dài như trên có được không?

– Câu trả lời là có, chúng ta hoàn toàn có thể dùng template variable như sau:

<input ngModel name="contactName" required #contactName="ngModel" class="form-control" type="text" ...>

Và sử dụng như một template variable thông thường:

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

\*ngIf="contactName?.errors?.required && contactName?.touched">

Name is required!

</div>

Như vậy, việc sử dụng Template-driven form trong Angular khá dễ dàng, trong phần này chúng ta chưa đề cập đến custom validation cho form control. Thay vào đó chúng ta sẽ có một phần riêng để thảo luận về tính năng này.