# 183. Module Introduction

Vì chúng ta đang tạo Single Page Application (SPA) nên sẽ không submit thẳng Form tới Server. Thay vào đó chúng ta cần sử lý Form thông qua Angular.

Angular giúp lấy data từ các trường input mà người dung đã nhập và validate các data này. Angular cũng cho phép chúng ta thay đổi cách hiển thị của một Form (Tạo các red borders cho các trường invalid…). Tóm lại Angular cung cấp các tool mạnh để làm việc với Form.

# 184. Why do we Need Angular’s Help?

Tiếp cận gần hơn để thấy rõ: Angular làm việc với Form như thế nào?

Như đã nhắc ở phần trước. Angular cho phép ta lấy các data mà người dung nhập vào các trường input, và check valid của các trường này…

Angular lấy các data mà người dung nhập vào Form và biến đổi chúng thành java script Obj đại diện cho Form đó. Ví dụ về Obj đại diện ở bên dưới - các thông tin mà người dùng nhập sẽ được lưu trong key Value của Object đại diện. Ngoài ra còn nhiều key khác trong obj để biểu diển trạng thái của một Form nhưng sẽ không được đề cập ở đây.

{

Value: {

Name: ‘Nhat’,

Email: ‘123@gmail.com’

},

Valid: true,

…

}

Từ Obj đại diện chúng ta có thể dễ dàng thao tác với Form và nắm bắt được trạng thái của nó

# 185. Template-Driven (TD) vs Reactive Approach

Một điều vô cùng quan trọng cần phải hiểu được rằng, Angular đề xuất 2 hướng tiếp cận khác nhau để sử lý Form:

* **Template-Driven:** Angular infers the Form Objet form the DOM – Đơn giản là chúng ta sẽ set up Form bằng HTML Code và angular sẽ tự động hiểu được cấu trúc của Form. Nó sẽ biết được trường nào có trong Form.
* **Reactive:** Form is created programmatically ad synchoronized with the DOM – Cấu trúc của Form sẽ được định nghĩa bằng TypeScript code. Mặt dù cách này khá phức tạp nhưng sau cùng nó sẽ cung cấp cho chúng ta những công cụ kiểm soát Form vô cùng mạnh – bạn có thể tùy chính từng phần nhỏ của Form tùy thích.

# 186. An Example Form \_ Template-Driven (Có code demo)

Vì Form trong Angular App không submit thẳng data tới Server nên ta có thể thấy trong thẻ HTML <form> sẽ không có attr “action” cũng như METHOD.

Thay vì Submit thẳng angular sẽ sử lý trước dữ liệu ở Form (như đã đề cập). Chính vì lý do trên nên khi ấn vào nút submit (trong code demo) sẽ không có bất kỳ hành động nào xảy ra trên trang web (do form không gữi đe request).

# 187. TD: Creating the Form and Registering the Controls

Ở phần này chúng ta sẽ cùng xem cách mà Angular hiểu và chuyển dữ liệu trong Form thành Obj đại diện cho Form như thế nào (Theo hướng tiếp cận Template Driven).

Để làm được điều này hãy đảm bảo rằng: Tại App Module, FormsModule đã được import. Module này cung cấp rất nhiều chức năng liên quan tới Form. Nó là bắt buộc để xử lý Form theo hướng tiếp cận Driven Template.

Cách thức: Khi angular phát hiện ra có cấu trúc form trong template của Component (thông qua thẻ Form đóng vai trò như selector) và nó sẽ tạo ra Object đại diện.

Tuy nhiên Angular vẫn chưa hiểu được các trường input trong Form. Để Angular hiểu đâu là các trường input chúng ta cần phải đăng ký các trường này với angular.

Cách thức đăng ký các input controller cho Form:

* Thêm “ngModel” vào các trường input.
* Thêm attr “name” và giá trị của name vào trường input.

# 188. TD: Submitting and Using the Form

Ở phần này chúng ta sẽ cùng làm cho Form có thể “Submit” được – hay nói cách khác là có thể thu thập được dữ liệu mà người dung nhập và trạng thái của Form thông qua event Submit.

Để Form có thể Submit được, chúng ta sẽ bind một event vào thẻ Form (chứ không phải nút submit) bằng directive “ngSubmit”. Sự kiện này sẽ kích hoạt mỗi khi Form được Submit. Ví dụ minh họa <from (ngSubmit)=”onSubmit()”>. Tới đây thì form đã có khả năng Submit.

Tuy nhiên làm thể nào để lấy dữ liệu từ form và chuyển nó thành Obj đại diện cho form. Chúng ta sẽ dùng #Reference\_Tempate tại thẻ <Form> và gán cho #Local\_Ref này giá trị là “ngForm”, sau đó truyền #Local\_Ref này vào hàm sử lý Submit đã bind vào sự kiện (ngSubmit). Ví dụ minh họa <from (ngSubmit)=”onSubmit(f)” #f=”ngForm”>.

Để xem nội dung của Obj đại diện cho Form, trong hàm sử lý submit chúng ta có thể làm như sau:

onSubmit(f: ngForm) {

console.log(f)

}

# 189. TD: Understanding Form State

Lượt sơ qua về các key trong Object đại diện của 1 form:

* value: Thông tin mà người dung nhập vào các trường input của Form được lưu ở đây.
* controls : Lưu tất cả các trường input đã đăng ký control trong Form.
* dirty: Sẽ mang giá trị False nếu chưa có trường nào trong form được điền. Nếu có trường nào thay đổi giá trị của nó là True.
* invalid: check xem trong form có bất cứ trường nào add validator không.
* valid: check xem trong form có bất cứ trường nào không hợp lệ hay không.
* Touched: kiểm tra 1 trường có bị click vào chưa
* …

# 190. TD: Accessing the Form with @ViewChild

Trong hầu hết các trường hợp, chúng ta có thể truy cập các trạng thái và thông tin của Form thông qua Objet đại diện theo cách thức ở phần (188).

Tuy nhiên bây giờ phần này sẽ giới thiệu thêm một số cách để truy cập vào form: Đó là sử dụng @ViewChild để truy cập vào form thông qua local\_referent. Ví dụ minh họa ở dưới.

Ta có code HTML (không truyền f vào onSubmit như phần 188):

<from (ngSubmit)=”onSubmit()” #f=”ngForm”>.

Để lấy được Obj đại diện của Form trong TS code:

@ViewChild(‘f’) signupForm: NgForm

onSubmit() {

console.log(this.signupForm)

}

Kết quả in ra của code trên là một Obj đại diện như phần 188.

Cách sử dụng @ViewChild để truy cập form này đặt biệt hữu ích nếu bạn muốn truy cập form không chỉ tại thời điểm Submit – không ấn nút submit vẫn truy cập được. Trong khi cách thức ở phần 188 cần phải nhấn nút submit.

# 191. TD: Adding Validation to check User Input

Angular hỗ trợ xác thực các trường input trong form thông qua các directive:

* required: check input có empty hay không.
* email: check valid email

Kết quả xác thực sẽ được hiển thị ở key “valid” trong Obj đại diện. Nếu có trường invalid thì key “valid” sẽ có giá trị là False. Ngược lại nếu tất cả các trường input đều hợp lệ thì “valid” sẽ có giá trị là true.

Ngoài ra để biết chính xác một trường input có valid hay không, chúng ta có thể chọn tới control của trường đó và chọn valid. Ví dụ: Obj.controls.email.valid, để xem kết quả xác thực cho trường email.

Angular luôn track trạng thái của các control trong form và tùy vào từng trạng thái nó sẽ add vào thẻ <input> của control tương ứng các class (như: ng-dirty: nếu input bị thay đổi, ng-touched: nếu input bị click, ng-valid: nếu thông tin người dung là hợp lệ, ng-invalid: nếu thông tin người dùng nhập không hợp lệ). Các class này cho ta biết trạng thái của input tại từng thời điểm. Và nhờ vào các class này chúng ta có thể CSS cho các thẻ input cho phù hợp (ví dụ: với ng-invalid – chúng ta có thể tạo ra red-border cho input vì nó không hợp lệ)

# 192. Built-in Validators & Using HTML5 Validation

Which Validators do ship with Angular?

Check out the Validators class: <https://angular.io/api/forms/Validators> - these are all built-in validators, though that are the methods which actually get executed (and which you later can add when using the reactive approach).

For the template-driven approach, you need the directives. You can find out their names, by searching for "validator" in the official docs: <https://angular.io/api?type=directive> - everything marked with "D" is a directive and can be added to your template.

Additionally, you might also want to enable HTML5 validation (by default, Angular disables it). You can do so by adding the ngNativeValidate  to a control in your template.

# 193. TD: Using the Form State & 194. TD: Outputting Validation Error Messages

Giả sử chúng ta khai báo local referent cho thẻ form như sau: <form #f=’ngForm”> và đã thêm các directive vào các trường input để valid.

Chúng ta có thể disable nút submits nếu có trường input không hợp lệ bằng cách: <button [disabled]=”!f.valid”>

Để CSS cho thẻ input không hợp lệ:

Input.ng-invalid.ng-touched {

border: 1px solid red;

}

Tạo warning message trên HTML:

<input

ngModel

name=”email

require

email

**#email=”ngModel”**

>

<p \*ngIf = **”!email.valid**”>input is invalid! && email.touched</p>

# 195. TD: Set Default Values with ngModel Property Binding

Để phục vụ cho mục đích gán giá trị mặt định cho input field. Thay vì để ngModel đứng riêng lẻ. Chúng ta có thể sử dụng properties biding lên nó: [ngModel]=”<giá trị mặt định>”

# 196. TD: Using ngModel with Two-Way-Binding

Cập nhật giá trị của input một cách tức thì với two way binding: [(ngModel)] = “Biến”.

# 197. TD: Grouping Form Controls

Với những Form có nhiều trường input. Để dễ dàng quản lý được các trường này chúng ta đôi khi cần gom nhóm chúng lại.

Cách thức thực hiện:

* Dùng 1 thẻ <div> để bọc các trường cần gom nhóm lại.
* Dùng directive “ngModelGroup” với thẻ <div> ở trên và gán cho nó 1 giá trị string – giá trị này chính là Tên của group input.
* Ví dụ minh họa, ở dưới.

<form>

<div id=”userd-data” **ngModelGroup=”userData”**>

<div class=”form-group”>

<label>User name</label>

<input type=”text” name=”user-name” ngModel>

</div>

<div class=”form-group”>

<label>Email</label>

<input type=”text” name=”email” ngModel>

</div>

</div>

<div class=”form-group”>

<label>Pass word</label>

<input type=”password” name=”password” ngModel>

</div>

</form>

Với việc gom nhóm này thì value(1 key trong obj đại diện lưu các giá trị của input) sẽ có dạng như sau:

{

Value: {

userData: {

user-name: ‘’,

email: ‘’

}

password: ‘’

}

}

Cấu trúc của key controls trong obj đại diện cũng thay đổi tương tự.

Cũng tương tự như các thẻ input (được angular thêm các class dùng để chỉ ra trạng thái của chúng – đã trình bày ở phần 191). Thẻ div group này cũng được thêm các class tương tự. Chúng ta hoàn toàn có thể validate tổng thể group bằng cách đặt local referent trong thẻ <div> và gán cho local\_ref này giá trị “ngModelGroup”. Ví dụ minh họa:

<div id=”userd-data” ngModelGroup=”userData” **#UserData=”ngModelGroup”**>

<div class=”form-group”>

<label>User name</label>

<input type=”text” name=”user-name” ngModel>

</div>

<div class=”form-group”>

<label>Email</label>

<input type=”text” name=”email” ngModel>

</div>

</div>

<p **\*ngIf=”!UserData.valid && UserData.touched”**>User data is invalid</p>

# 198. TD: Handling Radio Buttons

Code minh họa:

<div \*ngFor=”let gender of genders”>

<input

Type=”radio”

Name=”gender”

ngModel

[value]={{genger}}

>

</div>

Giá trị trong obj đại diện:

{

Value: {

Gender: <giá trị đã chọn>

}

}

# 199. TD: Setting and Patching Form Values

Ngoài dùng two way binding chúng ta cũng có cách tiếp cận khác để set giá trị cho trường input trong form.

Chúng ta có thể đặt #Local\_Ref=”ngForm” trong thẻ <form>. Trong code typeScript có 2 cách triển khai như sau:

**Cách 1: Set giá trị cho các trường input được chỉ định trong Form, chỉ có tác dụng trên form có đánh dấu (bao bọc bởi) ngForm**

@Viewchild(‘Local\_Ref’) refForm: NgForm

This.Local\_Ref.form**.pathchValue**({

Input\_name1: <giá trị>

Input\_group: {

Input\_member1: <giá trị>

}

})

**Cách 2: Set giá trị cho toàn Form (yêu cầu phải nhập chính xác và đầy đủ tất cả các trường trong form)**

@Viewchild(‘Local\_Ref’) refForm: NgForm

This.Local\_Ref**.setValue**({

Input\_name1: <giá trị>

Input\_group: {

Input\_member1: <giá trị>

}

})

# 200. TD: Using Form Data

Để lấy giá trị các trường input trong Form. Sử dụng thuộc tính “value” trong Obj đại diện.

Ví dụ: this.form.value.<tên trường input>

# 201. TD: Resetting Forms

Để reset lại form dụng phương thức reset(). Ví dụ: this.form.reset()

# 202. Introduction to the Reactive Approach

Giới thiệu cách làm việc thứ 2 với Form trong Angular – Reactive.

# 203. Reactive: Setup

Cách setup form với Type Script:

Tại file component.ts:

* Import { FormGroup } from @angular/forms
* myFrom: FromGroup – Biến này đại diện cho Form sẽ được tạo (Đến cuối thì bản chất của Form cũng là 1 Group of controls). Thông qua biến này chúng ta sẽ làm việc được với Form

Tại file app.module:

* Import { ReactiveFormsModule } from @angular/forms (Không cần import FormsModule – nó chỉ cần trong cách tiếp cận Template Driven).

# 204. Reactive: Creating a Form in Code