**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 và hai là Reactive forms hay Model-driven forms.

* **Template-driven Forms**: Cơ chế hoạt động của dạng forms này sẽ chủ yếu dựa vào các **directives** trên template như NgForm, NgModel, required, etc; để làm việc. Form dạng này sử dụng Two-way binding để update data model giữa template và component.
* **Reactive Forms**: Chúng ta sẽ xây dựng form từ các model, là các object có một số chức năng đặc biệt để quản lý được các form input. Nó cũng sử dụng một số (nhưng rất ít) các directives.

# Template-Driven Forms

## Template

|  |
| --- |
| <div class="container">      <form class="sign-in-form">        <h2>Sign in</h2>        <div class="row-control">          <mat-form-field appearance="outline">            <mat-label>Username</mat-label>            <input matInput placeholder="Username" />          </mat-form-field>        </div>        <div class="row-control">          <mat-form-field appearance="outline">            <mat-label>Password</mat-label>            <input type="password" matInput placeholder="Password" />          </mat-form-field>        </div>        <div class="row-control row-actions">          <button mat-raised-button color="primary" type="submit">Sign in</button>        </div>      </form>    </div> |

## 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 vào AppModule:

|  |
| --- |
| import { FormsModule } from '@angular/forms';  @NgModule({    declarations: [AppComponent, SignInComponent],    imports: [  *// other NgModules*      FormsModule,    ],  })  export class AppModule {} |

## ngForm và ngModel directives

Để có thể lấy về **NgForm** instance, chúng ta chỉ cần tạo một template variable cho thẻ form kèm theo chỉ định về exportAs như sau:

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

Từ đây có thể sử dụng variable signInForm này ở trong template, cũng như truyền về component class.

export class SignInComponent implements OnInit {

  constructor() {}

  ngOnInit(): void {}

  onSubmit(*form*: NgForm): void {

    console.log(*form*);

  }

}

<form

  class="sign-in-form"

  novalidate

  #signInForm="ngForm"

  (submit)="onSubmit(signInForm)"

>

  <h2>Sign in</h2>

  <div class="row-control">

    <mat-form-field appearance="outline">

      <mat-label>Username</mat-label>

      <input matInput placeholder="Username" ngModel name="username" />

    </mat-form-field>

  </div>

  <div class="row-control">

    <mat-form-field appearance="outline">

      <mat-label>Password</mat-label>

      <input

        type="password"

        matInput

        placeholder="Password"

        ngModel

        name="password"

      />

    </mat-form-field>

  </div>

  <div class="row-control row-actions">

    <button mat-raised-button color="primary" type="submit">Sign in</button>

  </div>

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

</form>

Để có thể lấy được value của form, chúng ta cần phải register các control vào form thông qua **ngModel** directive.

<input matInput placeholder="Username" ngModel name="username" />

## Event Submit vs 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

onSubmit(form: NgForm) {

*// Do something awesome*

  console.log(form);

  throw new Error('something went wrong');

}

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

## ngModel, [ngModel] and [(ngModel)]

Như chúng ta đã biết, ngModel là directive giúp chúng ta register control với form, vậy hai hình thái còn lại có tác dụng gì?

* **[ngModel]:** là property binding, tức là chúng ta có thể binding một value hoặc một property cho nó.
* **[(ngModel)]:** là two-way data binding, lúc này nó sẽ kết hợp propety binding và event binding để đồng bộ dữ liệu giữa template và model với nhau.

## Template-driven forms validation

Angular Template-driven Forms có cung cấp sẵn một số directives cơ bản để thực hiện việc validation.

* **required**: Yêu cầu form control không được phép bỏ trống.
* **minlength**: Yêu cầu form control phải có value có length ít nhất bằng một giá trị nào đó.
* **maxlength**: Yêu cầu form control phải có value có length không vượt quá một giá trị nào đó.
* **pattern**: Yêu cầu form control phải có value thỏa mãn một pattern nào đó (RegEx).
* **email**: Yêu cầu form control phải có value thỏa mãn pattern của một email

Chúng được viết là các directives, nên bạn có thể sử dụng như các directives khác trong template của bạn.

Validation status cho một control sẽ bao gồm các status sau:

* **touched**: true nếu người dùng đã focus vào control (như là input, textarea, etc) rồi sau đó blur khỏi control đó. Hoặc khi gọi markAsTouched.
* **untouched**: true nếu người dùng chưa đụng chạm gì đến control hoặc lần đầu tiên focus và chưa bị mất focus (ngược lại với touched)
* **dirty**: true nếu người dùng đã thay đổi value của control – nhập một ký tự vào input text chẳng hạn, kể cả việc nhập vào rồi xóa đi thì cũng tính là đã thay đổi.
* **pristine**: true nếu người dùng chưa thay đổi value của control, mặc dù có thể đã touched, nhưng chưa sửa đổi gì.

Ví dụ validate trên input username:

<div class="row-control">

    <mat-form-field appearance="outline">

      <mat-label>Username</mat-label>

      <input

        matInput

        placeholder="Username"

        required

        minlength="6"

        maxlength="32"

        [pattern]="usernamePattern"

        ngModel

        name="username"

      />

      <mat-error

        \*ngIf="

          signInForm.controls['username'].touched &&

          !signInForm.controls['username'].valid

        "

      >

        <span \*ngIf="signInForm.controls['username'].errors?.['required']"

          >Username is required</span

        >

        <span

          \*ngIf="signInForm.controls['username'].errors?.['minlength'] || signInForm.controls['username'].errors?.['maxlength']"

          >Length from 6 to 32 characters</span

        >

        <span

          \*ngIf="!(signInForm.controls['username'].errors?.['minlength'] || signInForm.controls['username'].errors?.['maxlength']) && signInForm.controls['username'].errors?.['pattern']"

          >Only alphabet</span

        >

      </mat-error>

    </mat-form-field>

  </div>

  <pre>{{ signInForm.controls["username"].errors | json }}</pre>

  <div class="row-control">

# 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 Forms là async còn Reactive Forms 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.

Form state ở trong Reactive Forms là immutable, mỗi sự thay đổi của form state sẽ đều tạo ra một state mới.

Reactive Forms sử dụng khá nhiều Observable streams, ví dụ như valueChanges, statusChanges chẳng hạn. Bạn hoàn toàn có thể combine, manipulate stream đó như những Observable thông thường.

Validation ở trên Reactive Forms cũng rất dễ dàng, nó chỉ là một function, và bạn hoàn toàn có thể thay đổi trong khi application đang chạy.

## Import APIs cho Reactive forms

Để sử dụng được **Reactive Forms** ở trong ứng dụng, chúng ta cần imports một NgModule là **ReactiveFormsModule** vào **NgModule** quản lý component của chúng ta - trong trường hợp của component hiện tại là **AppModule**.

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

@NgModule({

  declarations: [

*// components, pipes, directives*

  ],

  imports: [

*// other imports*

    ReactiveFormsModule,

  ],

*// ...*

})

export class AppModule {}

## Các thành phần cơ bản của form

* **AbstractControl** đây là base class của 3 thành phần kể trên - chúng là các thành phần cơ bản của form.
* **FormControl** là đơn vị nhỏ nhất của một form, dùng để track thông tin về value, validation của một form control như là thông tin của một input, một checkbox, etc.
* **FormGroup** là một tập hợp của các control/group/array (**AbstractControl**) khác. Dạng như một Object, nó có thể chứa các value đơn lẻ, hoặc các Object khác.
* **FormArray** cấu trúc dạng mảng, để quản lý các **AbstractControl** theo dạng mảng, dùng cho trường hợp cấu trúc có thể thêm bớt phần tử một cách linh hoạt.

Thông thường, mỗi một form sẽ bắt đầu bởi một **FormGroup**, nó sẽ đăng ký các **AbstractControl** khác bên trong nó.

export class SignInRfComponent implements OnInit {

  signInForm = new FormGroup({

    username: new FormControl(''), *// <== default value*

    password: new FormControl(''), *// <== default value*

  });

  constructor() {}

  ngOnInit(): void {}

}

## Binding Form

Để binding giữa form model và template lại với nhau, chúng ta sẽ dùng directive [**formGroup**].

Để binding các **FormControl** vào các form control như textbox, checkbox chúng ta sẽ dùng một directive đó là **formControlName**. Đầu vào của nó sẽ là key mà chúng ta dùng để khai báo control trong **FormGroup** gần nhất.

<form class="sign-in-form" [formGroup]="signInForm">

    <h2>Sign in Reactive Form</h2>

    <div class="row-control">

      <mat-form-field appearance="outline">

        <mat-label>Username</mat-label>

        <input matInput placeholder="Username" formControlName="username" />

      </mat-form-field>

    </div>

    <div class="row-control">

      <mat-form-field appearance="outline">

        <mat-label>Password</mat-label>

        <input

          type="password"

          matInput

          placeholder="Password"

          formControlName="password"

        />

      </mat-form-field>

    </div>

    <div class="row-control row-actions">

      <button mat-raised-button color="primary" type="submit">Sign in</button>

    </div>

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

  </form>

## FormBuilder Service

Chúng ta có thể thấy là nếu form có số lượng control lớn mà cứ phải **new** như ở trên thì khá là vất vả. Vì thế Angular Reactive Forms cung cấp luôn cho chúng ta một service là **FormBuilder** để các bạn có thể khởi tạo form nhanh hơn.

export class SignInRfComponent implements OnInit {

  signInForm = this.fb.group({

    username: '',

    password: '',

  })

  constructor(private *fb*: FormBuilder) {}

  ngOnInit(): void {}

}

## FormArray

Giả sử chúng ta muốn người dùng có thể nhập vào một hoặc nhiều ô input cùng loại (vd nhập nhiều SĐT). 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.

signInForm = this.fb.group({

    username: '',

    password: '',

    tels: this.fb.array([

      this.fb.control('')

    ])

  })

  get tels(): FormArray {

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

  }

  addTel() {

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

  }

  removeTel(*index*: number) {

    this.tels.removeAt(*index*);

  }

Template của FormArray **tels:**

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

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

            <mat-form-field appearance="outline">

              <mat-label>Số điện thoại</mat-label>

              <input

                type="tel"

                matInput

                placeholder="SĐT"

                [formControlName]="i"

              />

            </mat-form-field>

            <button mat-raised-button color="primary" (click)="addTel()">+</button>

            <button mat-raised-button color="warn" (click)="removeTel(i)" \*ngIf="tels.controls.length > 1">-</button>

          </div>

    </div>

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".

## Reactive Forms Validation

Bởi vì với Reactive Forms, chúng ta set up form ở trong component và từ đó link đến phần template HTML. Nên phần validators thay vì dùng các attribute trên template như template form, chúng sẽ được định nghĩa khi setup form thông qua FormBuilder. Phần validate này sẽ đều là các function.

Angular có cung cấp một set các validate function trong class [Validators](https://angular.io/api/forms/Validators), cụ thể:

class Validators {

  static min(*min*: number): ValidatorFn;

  static max(*max*: number): ValidatorFn;

  static required(*control*: AbstractControl): ValidationErrors | null;

  static requiredTrue(*control*: AbstractControl): ValidationErrors | null;

  static email(*control*: AbstractControl): ValidationErrors | null;

  static minLength(*minLength*: number): ValidatorFn;

  static maxLength(*maxLength*: number): ValidatorFn;

  static pattern(*pattern*: string | RegExp): ValidatorFn;

  static nullValidator(*control*: AbstractControl): ValidationErrors | null;

  static compose(*validators*: ValidatorFn[]): ValidatorFn | null;

  static composeAsync(*validators*: AsyncValidatorFn[]): AsyncValidatorFn | null;

}

Khi khởi tạo FormControl thì validators sẽ được truyền vào ở argument số 2. Argument số 1 sẽ là giá trị mặc định khi khởi tạo form.

let control = new FormControl('', Validators.required);

*//Or*

this.fb.control('', Validators.required);

signInForm = this.fb.group({

      username: [

        '',

        Validators.compose([

          Validators.required,

          Validators.minLength(6),

          Validators.pattern(/^[a-z]{6,32}$/i),

        ]),

      ],

      password: ''

    });

Chúng ta dùng **Validators.compose** và truyển vào một mảng các validators để có thể kết hợp được nhiều loại validators với nhau.

Cú pháp khi khởi tạo **FormControl** trong form group bằng **FormBuilder** sẽ hơi khác, thay vì 3 argument riêng biệt thì bạn sẽ truyền vào một mảng có 3 phần tử, đầu tiên là giá trị mặc định, thứ 2 là sync validator và cuối cùng là validator.

Phần hiển thị lỗi thì như Template-driven forms.

Trong Reactive Forms có 2 loại validator functions:

* **Sync validators (đồng bộ):** như chúng ta đã trình bày ở trên
* **Async validators (bất đồng bộ):** Đây là các validate function sẽ trả về Promise hoặc Observable mà kết quả sẽ được emit trong tương lai. Ví dụ như bạn muốn validate xem username nhập vào đã có trong hệ thống hay chưa. Thì bắt buộc bạn phải gửi một yêu cầu lên server để làm việc này, HTTP request thường sẽ trả về Promise/Observable.