Reactive Forms

Assim como o Template Driven Forms, os Reactives Forms também manipulam e controlam um formulário, mas de maneira reativa. Ele utiliza uma estrutura de objetos que controlam os elementos do seu formulário de diversas formas. Vamos iniciar nossos trabalhos com os Reactives Forms. Para trabalharmos com ele, devemos importar o módulo **ReactiveFormsModule** no módulo principal (*AppModule*).

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

import { AppComponent } from './app.component';
import { ReactiveFormsModule } from '@angular/forms';

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

Agora, nossa aplicação está apta para trabalhar com os Reactive Forms. Nós precisamos saber que o Reactive Forms é dividido em quatro partes essenciais: **Form Control, Form Group, Form Array e Form Builder**. Veremos cada uma dessas partes importantes.

Form Control

Os Form Controls são elementos do Reactive Forms que controlam um único elemento do formulário. Seja um input, um select, ou qualquer outro elemento. Para criarmos um Form Control, devemos criar uma propriedade na nossa classe que seja um Form Control. Vamos criar uma propriedade chamada **username** para testarmos.

```
import { Component } from '@angular/core';
import { FormControl } from '@angular/forms';

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  username: FormControl = new FormControl('')
}
```

Na instanciação do FormControl, nós informamos um valor inicial. Neste caso, uma string vazia. Agora, assim como fizemos com o Template Driven Forms, temos que informar que essa propriedade se relaciona com um elemento do nosso formulário. Para isso, em um dos elementos do nosso formulário, vamos colocar o atributo **[formControl]**, informando o nome do nosso FormControl, nesse caso, **username**.

```
Username: <input type="text" id="username" [formControl]="username">
```

Dessa maneira, a propriedade do nosso componente vai estar linkada ao input e toda vez que o valor do input ser alterado o valor da propriedade também será. Sendo assim, outra maneira de se criar um **Two Way Data Binding**. Podemos acessar o valor do Form Control a partir da propriedade **value**, que retorna o que foi digitado dentro do campo.

Form Group

O Form Group é utilizado para agrupar dois ou mais FormControls e, até mesmo, outros Form Groups, caso você necessite. Vamos considerar o seguinte cenário: você deve criar um formulário para registro de um usuário. Você pode criar um Form Group que possua todos os campos do seu formulário. Assim, quando você precisar salvar as informações desse formulário, todas elas estarão disponíveis no mesmo lugar. Vamos criar um FormGroup para esse formulário de usuários com os campos **nome, username, email e senha**.

```ts
import { Component } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  usuarioForm: FormGroup = new FormGroup({
    nome: new FormControl(''),
    username: new FormControl(''),
    email: new FormControl(''),
    senha: new FormControl('')
  })
}
```

Agora nosso Form Group com os campos desejados está pronto. Precisamos informar agora em qual formulário do nosso HTML esse Form Group irá funcionar. Para isso, em algum formulário do seu HTML, adicione o atributo **[formGroup]=””** e informe o nome o nome do seu Form Group

```html
<form [formGroup]="usuarioForm" (ngSubmit)="doSomething()">
  <label>Nome </label>
  <input type="text" name="nome" id="nome"> <br><br>

  <label>Username </label>
  <input type="text" name="username" id="username"> <br><br>

  <label>E-mail </label>
  <input type="email" name="email" id="email"> <br><br>

  <label>Senha </label>
  <input type="password" name="senha" id="senha"> <br><br>

  <button type="submit">Registrar!</button>
</form>
```

Nosso formulário, assim, está sendo referenciado ao nosso Form Group. No entanto, ainda precisamos referenciar os inputs aos respectivos Form Controls. Dessa vez, como os Form Controls estão dentro de um Form Group, não usaremos **[formControl]=””**, e sim, **formControlName=””**, onde o valor será o nome do Form Control dentro do Form Group.

```
<form [formGroup]="usuarioForm" (ngSubmit)="doSomething()">
  <label>Nome </label>
  <input
    type="text"
    name="nome"
    id="nome"
    formControlName="nome"> <br><br>

  <label>Username </label>
  <input
    type="text"
    name="username"
    id="username"
    formControlName="username"> <br><br>

  <label>E-mail </label>
  <input
    type="email"
    name="email"
    id="email"
    formControlName="email"> <br><br>

  <label>Senha </label>
  <input
    type="password"
    name="senha"
    id="senha"
    formControlName="senha"> <br><br>

  <button type="submit">Registrar!</button>
</form>
```

Agora nosso formulário está totalmente completo. Nós podemos acessar os valores acessados em cada Form Control do Form Group a partir da propriedade **value**, que retorna um objeto com o nome dos Form Controls e seus respectivos valores.

Form Array

Vamos considerar o seguinte cenário: em seu formulário, o usuário deve adicionar um número de telefone ou mais. Mas, por padrão, o seu formulário irá apresentar somente um único campo. Caso o usuário queira adicionar mais números de telefone, ele clicará em um botão que adiciona um novo campo para adicionar um novo número de telefone. O Form Array nos permitiria fazer isso de uma maneira mais ágil.

O Form Array é um elemento do Reactive Forms que nos permite armazenar, em um array, vários Form Controls que possuem a mesma funcionalidade, nesse exemplo, a funcionalidade de receber números de telefone. Vamos definir nosso Form Array

```ts
import { Component } from '@angular/core';
import { FormArray, FormControl } from '@angular/forms';

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  telefones: FormArray = new FormArray([
    new FormControl('')
  ])
}
```

Agora, com nosso Form Array definido, nós devemos mostrá-lo na tela. No entanto, devido ele ser um array, nós não podemos simplesmente colocá-lo na tela. Devemos percorrer esse array para mostrarmos nossos Form Controls que estão dentro dele. Então, vamos percorrer esse array utilizando a diretiva **ngFor** no nosso HTML.

```html
<form>
  <input
    type="tel"
    *ngFor="let controls of telefones.controls; let i = index"
    [formControlName]="i"
  >
</form>
```

Para acessarmos os controls para fazer o loop, utilizamos a propriedade **controls** dos Form Arrays. Nós também precisamos utilizar o **FormControlName**. Mas, nesse caso, o valor dele deve ser o índice do elemento no Array, já que estão em um Form Array. E agora, visualmente, os inputs dentro do Form Array serão renderizados.

Como vimos, nós podemos adicionar novos Form Controls ao nosso Form Array. Vamos fazer uma lógica para fazermos isso. De início, vamos criar um botão para adicionar um novo Form Control ao array. Quando clicar, ele vai executar um método chamado **add()**.

```html
<form>
  <input
    type="tel"
    *ngFor="let controls of telefones.controls; let i = index"
    [formControlName]="i"
  > <br> <br>

  <button (click)="add()">+</button>
</form>
```

Vamos criar esse método no nosso componente. Esse método vai adicionar um novo elemento ao Array e irá renderizar sempre que for adicionado um novo.

```typescript
import { Component } from '@angular/core';
import { FormArray, FormControl } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  telefones: FormArray = new FormArray([
    new FormControl('')
  ])

  add() {
    this.telefones.push(new FormControl(''))
  }
}
```

Como nós temos um Array, basta utilizarmos o método **push()** adicionando um novo Form Control que, automaticamente, ele será adicionado no array e o novo campo aparecerá na tela, pronto para ser usado.

Form Builder

Como vimos, para criarmos um Form Control, Form Group ou um Form Array, nós temos que instanciar objetos, e isso pode se tornar muito verboso, dependendo da quantidade de campos que seu formulário terá. Por isso, para diminuir a verbosidade, nós podemos usar o **Form Builder**. Um Form Builder nos permite criar esses elementos com uma sintaxe totalmente menor, facilitando o trabalho com os Reactive Forms.

Para nós trabalharmos com o Form Builder, nós temos que criar um construtor para nosso componente e adicionar ele ao nosso componente como uma propriedade.

```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  constructor(
    private fb: FormBuilder
  ) {}
}
```

Agora nosso Form Builder está pronto. Vamos criar um Form Group simples com 3 campos: **username, senha** e **telefone**, onde o telefone será um Form Array, como fizemos antes.

```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  cadastroForm: FormGroup = this.fb.group({
    username: this.fb.control(''),
    senha: this.fb.control(''),
    telefones: this.fb.array([
      this.fb.control('')
    ])
  })

  constructor(
    private fb: FormBuilder
  ) {}
}
```

Assim nós criamos um Form Group com Form Controls e um Form Array utilizando a sintaxe do Form Builder. O Form Builder nos permite criar uma sintaxe menor ainda. Nós

podemos representar os Form Controls, no Form Builder, como **[‘’]** , deixando a sintaxe ainda menor.

```
import { Component } from '@angular/core';
import { FormArray, FormBuilder, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  cadastroForm: FormGroup = this.fb.group({
    username: [''],
    senha: [''],
    telefones: this.fb.array([
      ['']
    ])
  })

  constructor(
    private fb: FormBuilder
  ) {}
}
```

Dessa maneira, teremos uma sintaxe ainda menor utilizando o Form Builder. Agora, basta colocarmos na tela esses campos. No entanto, como nosso Form Array está dentro de um Form Group, devemos fazer um “macete” para mostrarmos ele na tela. Vamos colocar o Form Array em uma propriedade fora a parte.

```ts
import { Component } from '@angular/core';
import { FormArray, FormBuilder, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  cadastroForm: FormGroup = this.fb.group({
    username: [''],
    senha: [''],
    telefones: this.fb.array([
      ['']
    ])
  })

  telefones: FormArray = this.cadastroForm.get('telefones') as FormArray

  constructor(
    private fb: FormBuilder
  ) {}
}
```

Agora sim, vamos poder montar nosso formulário no HTML e renderizar essa aplicação.

```html
<form [formGroup]="cadastroForm">
  <label for="username">Username</label>
  <input type="text" id="username" formControlName="username"> <br><br>

  <label for="senha">Senha</label>
  <input type="text" id="senha" formControlName="senha"> <br><br>

  <h4>Adicione seus telefones</h4>

  <input
    type="tel"
    *ngFor="let control of telefones.controls; let i = index"
    [formControlName]="i"
  >
</form>
```

E agora, basta trabalharmos com esse formulário como quisermos.

Validação de Formulários com o Reactive Forms

Para validarmos um formulário que montamos com o Reactive Forms, utilizamos os **Validators**. Os Validators é um objeto que possui todas as propriedades HTML de validação de formulários (maxlength, required, etc…) para utilizarmos nos formulários. Vamos criar um Form Control para vermos como utilizar.

```
import { Component } from '@angular/core';
import { FormControl } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  meuControl: FormControl = new FormControl('')
}
```

Agora, vamos adicionar nossos Validators nesse Form Control, para isso, vamos passar outro parâmetro do Form Control que será um Array com os validators. Para isso, vamos informar que esse campo é obrigatório.

```
import { Component } from '@angular/core';
import { FormControl, Validators } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  meuControl: FormControl = new FormControl('', [Validators.required])
}
```

Agora nosso Form Control é obrigatório.

```html
<label for="teste">Teste </label>
<input type="text" [formControl]="meuControl">

<button type="button" [disabled]="meuControl.invalid">Enviar</button>
```

Agora, quando o input estiver inválido, será impossível enviar o formulário. Também temos todos os outros elementos de validação. Vamos adicionar mais alguns para vermos como funciona.

```ts
import { Component } from '@angular/core';
import { FormControl, Validators } from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  meuControl: FormControl = new FormControl('', [
    Validators.required,
    Validators.minLength(5),
    Validators.maxLength(20)
  ])
}
```