

# Formulários

# Forms

- Os formulários Angular são usados para coletar os dados do usuário;
- Os formulários de entrada de dados podem ser muito simples ou muito complexos;
- Ele pode conter um grande número de campos de entrada, abrangendo vários tabs;
- Os formulários também podem conter lógica de validação complexa, interdependente de vários campos.

# Duas abordagens

- Template-driven (Orientados por template)
  - A lógica do formulário é colocada no template (Código HTML)
- Model-driven (Reactive)
  - A lógica do formulário é definida no componente como um objeto;
  - Esta abordagem tem mais benefícios, pois facilita o teste do componente.

# Template-driven Forms

- Se baseia nas diretivas do modelo para criar e manipular o modelo de objeto subjacente;
- São úteis para adicionar um formulário simples a um aplicativo, como um formulário de inscrição em uma lista de e-mail;
- São fáceis de adicionar a um aplicativo, mas não são escalonáveis tão bem quanto os formulários reativos;
- Se a aplicação tiver requisitos e lógica de formulário muito básicos que podem ser gerenciados somente no template, os formulários baseados em template podem ser uma boa opção.

# Model-driven (Reativos)

- Fornece acesso direto e explícito ao objeto do formulário subjacente;
- São mais robustos: são mais escalonáveis, reutilizáveis e testáveis;
- Se os formulários forem uma parte fundamental no desenvolvimento do aplicativo é recomendado o uso dos formulários reativos.

# Formulários reativos vs Template-driven

Recurso	Reativos	Orientados por Template
Configuração (Importação)	ReactiveFormsModule	FormsModule
Modelo de dados	Estruturado e Imutável	Implícito, criado por diretivas
Data-flow	Síncrono	Assíncrono
Validação de formulário	Funções	Diretivas
Escalabilidade	Alta (melhor para cenários complexos)	Baixa (melhor para cenários simples)
Testes unitários	Mais fácil	Mais difícil
Legibilidade	Menos legível (devido à lógica estar na classe do componente)	Mais legível (devido à lógica estar no template)

# Classes básicas de formulário comuns

Tanto os formulários reativos e template-driven são criados nas seguintes classes:

<b><u>Classe</u></b>	<b>Descrição</b>
FormControl	Utilizada para rastrear o valor e a validação de um único campo de formulário.
FormGroup	Usada para rastrear o valor e o estado de validação de um grupo de FormControl.
FormArray	Rastreia o valor e o estado de validação de uma matriz de FormControl, FormGroup ou FormArray instâncias.
ControlValueAccessor	Cria uma ponte entre instâncias Angular FormControl e elementos DOM integrados.

# Template-Driven

```
<div class="form-container">
  <form id="formComum" (ngSubmit)="onSubmit(formulario)" #formulario="ngForm">
    <label>
      ID:
      <input name="id" ngModel>
    </label>
    <label>
      Modelo:
      <input name="modelo" ngModel>
    </label>
    <label>
      Capacidade:
      <input name="capacidade" ngModel>
    </label>
    <label>
      Companhia Aérea:
      <input name="empresa" ngModel>
    </label>
    <button type="submit">Enviar</button>
  </form>
</div>
```

1\_forms\src\app\formulario-comum\formulario-comum.component.html



# Template-Driven

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

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

  onSubmit(formulario: NgForm){
    let { id, modelo, capacidade, empresa } = formulario.value;
    console.log(id, modelo, capacidade, empresa);
  }
}
```

1\_forms\src\app\formulario-comum\formulario-comum.component.ts

# Template-Driven

```
import { Component, ViewChild } from '@angular/core';
import { NgForm } from '@angular/forms';

@Component({
  selector: 'app-formulario-comum',
  templateUrl: './formulario-comum.component.html',
  styleUrls: ['./formulario-comum.component.css']
})
export class FormularioComumComponent {
  @ViewChild('formulario') aviaoForm: NgForm | undefined;

  onSubmit(){
    if(this.aviaoForm){
      let { id, modelo, capacidade, empresa } = this.aviaoForm.value;
      console.log(id, modelo, capacidade, empresa);
    }
  }
}
```

1\_forms\src\app\formulario-comum\formulario-comum.component.ts

# ngForm

- Assim que o FormsModule é importado, esta diretiva se torna ativa por padrão em todas as tags <form> do template;
- É possível exportar a diretiva para uma variável de modelo (ngModel) local usando ngForm como chave. Ex: `#formulario="ngForm"`
- Muitas propriedades da instância FormGroup subjacente são duplicadas na própria diretiva, essa referência fornece dentre outras funcionalidades, o status de validade do formulário e as propriedades de interação com o usuário;

# Acessando valores e status de Validade do Formulário

```
<form id="formComum" (ngSubmit)="onSubmit()" #formulario="ngForm">
```

ID:

Modelo:

Capacidade:

Companhia Aérea:

E-Mail

Email:

```
<label>
  Modelo:
  <input
    name="modelo"
    ngModel
    required
    #modelo="ngModel">
</label>
```

Enviar

Modelo: Mustang P51

```
<p>Modelo: {{modelo.value}}</p>
```

Valido?: true

```
<p>Valido?: {{modelo.valid}}</p>
```

Formulario: { "id": "1", "modelo": "Mustang P51", "capacidade": "1", "empresa": "FAB", "email": "p51@fab.com" }




















```
<p>Formulario: {{formulario.value | json}}</p>
```

# Validação

- Validadores são usados para garantir que os valores em um formulário atendam a determinados requisitos;
- Eles estão disponíveis para formulários template-driven ou formulários reativos em aplicativos Angular;
- Existem vários validadores integrados, como `required`, `email` e `minLength`;
- Também é possível desenvolver **validadores personalizados** para abordar funcionalidades que não são tratadas por um validador embutido no Angular.

# Diretivas

## forms

 AbstractFormGroupDirective	 CheckboxControlValueAccessor...	 CheckboxRequiredValidator
 DefaultValueAccessor	 EmailValidator	 FormArrayName
 FormControlDirective	 FormControlName	 FormGroupDirective
 FormGroupName	 MaxLengthValidator	 MaxValidator
 MinLengthValidator	 MinValidator	 NgControlStatus
 NgControlStatusGroup	 NgForm	 NgModel
 NgModelGroup	 NgSelectOption	 NumberValueAccessor
 PatternValidator	 RadioControlValueAccessor	 RangeValueAccessor
 RequiredValidator	 SelectControlValueAccessor	 SelectMultipleControlValueAc...

<https://angular.io/api?type=directive>

# EmailValidator

- Uma diretiva que adiciona o validador de email aos controles marcados com o atributo email;
- A validação de e-mail é baseada na especificação HTML WHATWG com algumas melhorias para incorporar mais regras RFC

```
<input type="email" name="email" ngModel email>  
<input type="email" name="email" ngModel email="true">  
<input type="email" name="email" ngModel [email]="true">
```

# Validação (email, minlength)

Companhia Aérea:  Escreva no mínimo de 3 caracteres

E-Mail

teste Entre com um email válido

```
<label for="email">E-Mail</label>
<input
  type="email"
  id="email"
  name="email"
  required
  ngModel
  email
  #email="ngModel">
<span *ngIf="!email.valid && email.touched">
  Entre com um email válido
</span>
```

<https://angular.io/api/forms/EmailValidator>

```
<input
  name="empresa"
  ngModel
  required
  minlength="3"
  empresa
  #empresa="ngModel">
<span *ngIf="!empresa.valid && empresa.touched">
  Escreva no mínimo de 3 caracteres
</span>
```

<https://angular.io/api/forms/MinLengthValidator>

<https://angular.io/api/forms/NgModel>



# Model- driven (Reactive) Forms

```
export class FormReativoComponent implements OnInit{  
  exemploForm: FormGroup;  
  
  constructor() {  
    this.exemploForm = new FormGroup({  
      'id': new FormControl(null),  
      'modelo': new FormControl(null),  
      'capacidade': new FormControl(null),  
      'empresa': new FormControl('FAB')  
    });  
  }  
  
  onSubmit(){  
    console.log(this.exemploForm);  
    console.log(this.exemploForm.value);  
  }  
}
```

# Sincronizando Form - HTML

```
<!-- directive formGroup -->
<!-- directive formControlName -->
<form [formGroup]="exemploForm" (ngSubmit)="onSubmit()">
  <label for="id">Id:</label>
  <input
    |   formControlName="id"
    |   name="id"> <br>
  <input
    |   formControlName="modelo"><br>
  <input
    |   formControlName="capacidade"><br>
  <input
    |   formControlName="empresa"><br>
  <button type="submit">Enviar</button>
</form>
```

## Validação


```
constructor() {  
  this.exemploForm = new FormGroup({  
    'id': new FormControl(null, Validators.required),  
    'modelo': new FormControl(null, [Validators.required,  
                                     Validators.minLength(3),  
                                     Validators.maxLength(10)]),  
    'capacidade': new FormControl(null),  
    'empresa': new FormControl('FAB')  
  });  
}
```

<https://angular.io/api/forms/Validators>

# Validação

```
<form [formGroup]="exemploForm" (ngSubmit)="onSubmit()">
```

```
<input
  FormControlName="modelo">
  <span *ngIf="!exemploForm.get('modelo')?.valid &&
    exemploForm.get('modelo')?.touched">Min:3 Max:10 caracteres</span>
```

```
input.ng-invalid.ng-touched {
  border: 1px solid  red;
}
```

<https://angular.io/api/forms/FormGroup>

# FormArray

- FormArray é um array de FormControl;
- No FormGroup cada FormControl é uma propriedade com o nome do controle como chave;
- Em FormArray há um array de FormControl.

# FormArrayName

- Sincroniza um FormArray aninhado com um elemento DOM.

# FormArray

```
constructor() {  
  this.exemploForm = new FormGroup({  
    'id': new FormControl(null, Validators.required),  
    'modelo': new FormControl(null, [Validators.required,  
                                     Validators.minLength(3),  
                                     Validators.maxLength(10)]),  
    'capacidade': new FormControl(null),  
    'empresa': new FormControl('FAB'),  
    'logVoos': new FormArray([])  
  });  
}
```

src\app\form-reactivo\form-reactivo.component.ts

```
<div formArrayName="logVoos">  
  <div *ngFor="let log of getControls(); let i=index">  
    <label for="input{{i}}">{{labelLogs[y]}}</label>  
    {{ incY() }}  
    <input id="input{{i}}" type="text" [formControlName]="i">  
    <hr>  
  </div>  
</div>
```

src\app\form-reactivo\form-reactivo.component.html

```
addLogVoo(){  
  const data = new FormControl(null);  
  const origem = new FormControl(null);  
  const destino = new FormControl(null);  
  (<FormArray>this.exemploForm.get('logVoos')).push(data);  
  (<FormArray>this.exemploForm.get('logVoos')).push(origem);  
  (<FormArray>this.exemploForm.get('logVoos')).push(destino);  
}
```

src\app\form-reactivo\form-reactivo.component.ts

# Criando Validadores Customizados

- É possível criar validadores customizados quando os validadores embutidos do angular não atender a demanda do problema atual;
- Exemplo de validador para o campo input empresa em que o campo será válido apenas se for digitado 'FAB' ou 'AZUL' .



# Validador Personalizado empresaValidador

```
constructor() {  
  this.exemploForm = new FormGroup({  
    'id': new FormControl(null, Validators.required),  
    'modelo': new FormControl(null, [Validators.required,  
                                     Validators.minLength(3),  
                                     Validators.maxLength(10)]),  
    'capacidade': new FormControl(null),  
    'empresa': new FormControl('FAB', [this.empresaValidator.bind(this)]),  
    'logVoos': new FormArray([])  
  });  
}
```

```
empresaValidator(control: AbstractControl):  
{ [key: string]: boolean } | null {  
  
  const value = control.value;  
  if (value !== 'FAB' && value !== 'AZUL') {  
    return { 'invalidEmpresa': true };  
  }  
  return null;  
}
```

```
<input formControlName="empresa" name="empresa">  
<div *ngIf="exemploForm.get('empresa')?.errors?.['invalidEmpresa']">  
  <p>Empresa invalida</p>  
</div>
```



# Exercício 01

- Crie um formulário com os seguintes controles (FormControls) e validações (Validators);
  - Nome da Estação climática (Não pode permanecer vazio);
  - Email de notificação (Não pode permanecer vazio e deve ser um email válido);
  - Localização (Não pode permanecer vazio)
  - Status de instalação. Um dropdown que deve conter 3 valores: 'Não instalada', 'Instalada', 'Operante';
  - Botão de Submit

# Referências

- <https://angular.io/guide/forms-overview>
- <https://blog.angular-university.io/introduction-to-angular-2-forms-template-driven-vs-model-driven/>
- <https://angular.io/api/forms/Validators>
- <https://angular.io/api?type=directive>
- <https://angular.io/api/forms/NgForm#description>
- <https://angular.io/api/forms/FormGroup>
- <https://angular.io/guide/form-validation>