Criação de Formulário Validado com Standalone Component no Ionic Angular

1. Introdução

Neste manual, você aprenderá a criar um formulário reativo com validação em um projeto Ionic utilizando componentes standalone. Além do passo a passo prático, o documento explica em detalhes os conceitos técnicos utilizados, incluindo sintaxes especiais como *ngIf, *ngFor, formControlName, e os módulos responsáveis pela funcionalidade de formulários reativos.

2. O que é um Standalone Component?

Um Standalone Component é um tipo de componente Angular que não precisa ser declarado em um NgModule. Ele é autossuficiente e pode importar os módulos que precisa diretamente.

- Elimina a necessidade de arquivos como home.module.ts.

- Permite carregamento direto via loadComponent nas rotas.

- Reduz dependências e melhora o tempo de carregamento da aplicação.

3. Criação do Projeto

```
ionic start formularioApp blank --type=angular
```

formularioApp: nome do projeto.

blank: modelo de estrutura limpa.

--type=angular: especifica Angular como framework.

4. Configuração de Rota com loadComponent

```
const routes: Routes = [
  {
    path: '',
    loadComponent: () => import('./home/home.page').then(m =>
m.HomePage),
  }
];
```

loadComponent substitui loadChildren para carregar standalone components.

import(...).then(...): função assíncrona que importa o componente dinamicamente.

5. Criando o Componente home.page.ts com Formulário Reativo

```
@Component({
  selector: 'app-home',
  standalone: true,
  templateUrl: './home.page.html',
  styleUrls: ['./home.page.scss'],
  imports: [IonicModule, CommonModule, ReactiveFormsModule]
})
```

- standalone: true define o componente como autônomo.

- imports especifica os módulos usados dentro do componente.

6. Criando o Formulário Reativo com Validações

```
this.form = this.fb.group({
  nome: ['', Validators.required],
  email: ['', [Validators.required, Validators.email]],
  senha: ['', [Validators.required, Validators.minLength(6)]]
});
```

- FormBuilder facilita a criação de formulários.

- Validators asseguram regras específicas: required, email, minLength.

7. HTML com Campos e Validações

```
<ion-item>
  <ion-label position="floating">Nome</ion-label>
  <ion-input formControlName="nome"></ion-input>
</ion-item>

<ion-text color="danger" *ngIf="form.get('nome')?.invalid &&
form.get('nome')?.touched">
  <p class="ion-padding-start">Nome é obrigatório.</p>
</ion-text>
```

- formControlName vincula o campo ao controle definido no FormGroup.

- *ngIf exibe o elemento apenas se a condição for verdadeira.

```
<p *ngIf="mostrarMensagem">Olá!</p>
```

- *ngFor percorre listas e renderiza elementos com base em cada item.

```
<ion-item *ngFor="let produto of produtos">{{ produto.nome }}</ion-
item>
```

8. Envio do Formulário com Toast de Sucesso

```
async enviar() {
  if (this.form.valid) {
    console.log('Dados enviados:', this.form.value);
    this.form.reset();

    const toast = await this.toastController.create({
      message: 'Formulário enviado com sucesso!',
      duration: 2000,
      color: 'success',
      position: 'top'
    });

    toast.present();
  } else {
    this.form.markAllAsTouched();
  }
}
```

- this.form.valid verifica se todos os campos são válidos.

- this.form.reset() limpa o formulário.

- toastController exibe uma notificação temporária.

- markAllAsTouched força exibição de erros.

9. CSS (Opcional)

```
ion-text {
  font-size: 0.9rem;
  margin-top: 4px;
}
```

10. Estrutura Final do HTML (Resumo)

```
<form [formGroup]="form" (ngSubmit)="enviar()">
  <ion-item>
    <ion-label position="floating">Nome</ion-label>
    <ion-input formControlName="nome"></ion-input>
  </ion-item>
  <ion-text color="danger" *ngIf="form.get('nome')?.invalid &&
form.get('nome')?.touched">
    <p class="ion-padding-start">Nome é obrigatório.</p>
  </ion-text>

  <ion-item>
    <ion-label position="floating">Email</ion-label>
    <ion-input type="email" formControlName="email"></ion-input>
  </ion-item>
  <ion-text color="danger" *ngIf="form.get('email')?.invalid &&
```

```
form.get('email')?.touched">
    <p class="ion-padding-start">Email inválido ou obrigatório.</p>
  </ion-text>

  <ion-item>
    <ion-label position="floating">Senha</ion-label>
    <ion-input type="password" formControlName="senha"></ion-input>
  </ion-item>
  <ion-text color="danger" *ngIf="form.get('senha')?.invalid &&
form.get('senha')?.touched">
    <p class="ion-padding-start">Senha precisa ter no mínimo 6
caracteres.</p>
  </ion-text>

  <ion-button expand="full" type="submit" class="ion-margin-
top">Enviar</ion-button>
</form>
```

11. Conclusão

Esse formulário utiliza boas práticas modernas com Angular e Ionic:

- Standalone Components para modularidade

- Reactive Forms com validações robustas

- Mensagens de erro dinâmicas com *ngIf

- Feedback com toastController

- Limpeza automática dos campos após envio