**1. Template-Driven Forms**

* **Simpler Setup:** For basic forms, this approach can be more concise.
* **Direct Binding:** Uses ngModel directive to bind form controls directly to component properties.
* **Built-in Validation:** Leverages built-in validation directives like required, minlength, maxlength, etc.

HTML

<form (ngSubmit)="onSubmit()">

<div>

<label for="name">Name:</label>

<input type="text" id="name" name="name" [(ngModel)]="name" required>

<div \*ngIf="name.invalid && (name.dirty || name.touched)">

<span \*ngIf="name.errors?.required">Name is required.</span>

<span \*ngIf="name.errors?.minlength">Name must be at least 3 characters long.</span>

</div>

</div>

<button type="submit" [disabled]="!myForm.valid">Submit</button>

</form>

TypeScript

import { Component } from '@angular/core';

@Component({

selector: 'app-template-driven-form',

template: /\* ... template as above ... \*/

})

export class TemplateDrivenFormComponent {

name: string = '';

onSubmit() {

// Handle form submission

console.log(this.name);

}

}

**2. Reactive Forms**

* **More Control:** Offers greater flexibility and control over form logic.
* **Data-Driven:** Forms are built using a FormGroup and FormControl objects, making them easier to manage and test.
* **Powerful Validation:** Supports complex validation scenarios using custom validators and asynchronous validation.

TypeScript

import { Component } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

@Component({

selector: 'app-reactive-form',

template: `

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

<div>

<label for="name">Name:</label>

<input type="text" id="name" formControlName="name">

<div \*ngIf="myForm.get('name')?.hasError('required')">Name is required.</div>

<div \*ngIf="myForm.get('name')?.hasError('minlength')">Name must be at least 3 characters long.</div>

</div>

<button type="submit" [disabled]="!myForm.valid">Submit</button>

</form>

`

})

export class ReactiveFormComponent {

myForm: FormGroup;

constructor(private fb: FormBuilder) {

this.myForm = this.fb.group({

name: ['', [Validators.required, Validators.minLength(3)]]

});

}

onSubmit() {

if (this.myForm.valid) {

// Handle form submission

console.log(this.myForm.value);

}

}

}

**Key Considerations:**

* **Choose the right approach:** Template-driven forms are suitable for simpler forms, while reactive forms are better for complex scenarios with dynamic controls, advanced validation, or when you need more control over form behavior.
* **Validation:** Implement robust validation logic using built-in validators or custom validators to ensure data integrity.
* **Asynchronous Operations:** Handle asynchronous operations (e.g., API calls) within your form handling logic.
* **Testing:** Write unit tests for your forms to ensure they function correctly and handle user input effectively.

By effectively utilizing these form handling techniques, you can create user-friendly and robust forms in your Angular applications.