**Chapter 7: Forms**

**Template-Driven & Reactive**

Forms are the primary way users interact with your app — entering data, submitting info, etc. Angular offers **two approaches** to handle forms:

* **Template-Driven Forms:** Easy for simple forms, mainly controlled in the template with minimal TypeScript.
* **Reactive Forms:** More powerful, scalable, and suited for complex scenarios with explicit form control and validation in TypeScript.

**1. Template-Driven Forms**

Forms where most of the form logic lives in the HTML template using directives like ngModel. Angular infers the form structure automatically.

**When to use?**

* Small/simple forms (e.g., login, contact form)
* When you want quick setup without much TypeScript

Make sure to import FormsModule in your standalone component:

**Basic Example: Simple Login Form**

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

import { **FormsModule**, NgForm } from '@angular/forms';

@Component({

standalone: true,

imports: [**FormsModule**],

selector: 'app-template-form',

templateUrl: './template-form.component.html',

})

export class TemplateFormComponent {

loginData = { username: '', password: '' };

submitted = false;

onSubmit(form: NgForm) {

if (form.valid) {

this.loginData = form.value;

this.submitted = true;

}

}

}

<form #loginForm="ngForm" (ngSubmit)="onSubmit(loginForm)">

<label>

Username:

<input name="username" ngModel required />

</label>

<label>

Password:

<input type="password" name="password" ngModel required minlength="6" />

</label>

<button type="submit" [disabled]="!loginForm.valid">Login</button>

</form>

<p \*ngIf="submitted">

Submitted username: {{ loginData.username }}

</p>

**2. Reactive Forms (Preferred Approach)**

Form controls, groups, and validations are explicitly defined and managed in TypeScript code, offering full control and predictability.

**When to use?**

* Large, complex, or dynamic forms
* Advanced validation needs (sync & async)
* Reactive UI updates based on form state

Import ReactiveFormsModule in your standalone component:

**Core Classes:**

| **Class** | **Purpose** |
| --- | --- |
| FormControl | Tracks value & validation of a single form input |
| FormGroup | Groups multiple FormControls |
| FormArray | Manages dynamic arrays of controls |

**Basic Example: Reactive Login Form**

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

import { FormGroup, FormControl, Validators, **ReactiveFormsModule** } from '@angular/forms';

@Component({

standalone: true,

imports: [**ReactiveFormsModule**],

selector: 'app-reactive-login',

template: `

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

<label>

Username:

<input formControlName="username" />

</label>

<div \*ngIf="username.invalid && username.touched" class="error">

Username is required.

</div>

<label>

Password:

<input type="password" formControlName="password" />

</label>

<div \*ngIf="password.invalid && password.touched" class="error">

Password is required and must be at least 6 characters.

</div>

<button type="submit" [disabled]="loginForm.invalid">Login</button>

</form>

<p \*ngIf="submitted">Logged in as: {{ loginForm.value.username }}</p>

`,

styles: [`.error { color: red; }`]

})

export class ReactiveLoginComponent {

loginForm = new FormGroup({

username: new FormControl('', Validators.required),

password: new FormControl('', [Validators.required, Validators.minLength(6)])

});

get username() { return this.loginForm.get('username')!; }

get password() { return this.loginForm.get('password')!; }

submitted = false;

onSubmit() {

if (this.loginForm.valid) {

this.submitted = true;

console.log('Form Value', this.loginForm.value);

}

}

}

**3. Advanced Reactive Forms Features**

**Dynamic Forms with FormArray**

Useful when you need variable-length lists of inputs (e.g., adding/removing phone numbers).

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

import { FormGroup, FormControl, FormArray, Validators, ReactiveFormsModule } from '@angular/forms';

@Component({

standalone: true,

imports: [ReactiveFormsModule],

selector: 'app-phone-list',

template: `

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

<div formArrayName="phones">

<div \*ngFor="let phone of phones.controls; let i=index" [formGroupName]="i">

<input formControlName="number" placeholder="Phone number" />

<button type="button" (click)="removePhone(i)">Remove</button>

</div>

</div>

<button type="button" (click)="addPhone()">Add Phone</button>

<button type="submit" [disabled]="phoneForm.invalid">Submit</button>

</form>

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

`

})

export class PhoneListComponent {

phoneForm = new FormGroup({

phones: new FormArray([

new FormGroup({

number: new FormControl('', Validators.required)

})

])

});

get phones() {

return this.phoneForm.get('phones') as FormArray;

}

addPhone() {

this.phones.push(new FormGroup({ number: new FormControl('', Validators.required) }));

}

removePhone(index: number) {

this.phones.removeAt(index);

}

onSubmit() {

if (this.phoneForm.valid) {

console.log('Phones:', this.phoneForm.value);

}

}

}