Angular Template Forms and Validation - Complete Training Guide

Table of Contents

---

1. Introduction to Angular Forms {#introduction}

What are Angular Forms?

Angular provides two approaches to handle user input through forms:

- **Template-Driven Forms**: Use directives in the template to create and manipulate forms
- **Reactive Forms**: Provide a model-driven approach to handling form inputs

This training focuses on **Template-Driven Forms**.

Why Template-Driven Forms?

✅ Easy to use for simple forms ✅ Less code required ✅ Familiar to developers coming from AngularJS ✅ Automatic tracking of form and input states ✅ Built-in validation

---

2. Template-Driven Forms {#template-driven-forms}

2.1 Prerequisites

Module Import

First, import `FormsModule` in your Angular module:

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';  // Import FormsModule

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule  // Add FormsModule here
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

For Standalone Components (Angular 14+):

```
// app.component.ts
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [FormsModule],  // Import FormsModule in standalone component
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  // Component logic
}
```

2.2 Basic Template-Driven Form

Component TypeScript File

```
// user-form.component.ts
import { Component } from '@angular/core';

interface User {
  firstName: string;
  lastName: string;
  email: string;
  age: number;
  address: string;
}

@Component({
  selector: 'app-user-form',
  templateUrl: './user-form.component.html',
  styleUrls: ['./user-form.component.css']
})
export class UserFormComponent {
  // Model to bind form data
  user: User = {
    firstName: '',
    lastName: '',
    email: '',
    age: 0,
    address: ''
  };

  // Flag to track form submission
  submitted = false;

  // Method to handle form submission
  onSubmit(form: any): void {
    if (form.valid) {
      this.submitted = true;
      console.log('Form Submitted!', this.user);
      // Here you would typically send data to a service/API
    }
  }

  // Method to reset form
  resetForm(form: any): void {
    form.reset();
    this.submitted = false;
    this.user = {
      firstName: '',
      lastName: '',
      email: '',
      age: 0,
      address: ''
    };
  }
}
```

Component Template File

```
<!-- user-form.component.html -->
<div class="form-container">
  <h2>User Registration Form</h2>

  <!-- #userForm creates a template reference variable -->
  <!-- ngForm directive is automatically applied to form elements -->
  <form #userForm="ngForm" (ngSubmit)="onSubmit(userForm)">

    <!-- First Name Field -->
    <div class="form-group">
      <label for="firstName">First Name:</label>
      <input
        type="text"
        id="firstName"
        name="firstName"
        [(ngModel)]="user.firstName"
        #firstName="ngModel"
        required
        minlength="2"
        class="form-control"
      >
```

```html
</div>

<!-- Last Name Field -->
<div class="form-group">
  <label for="lastName">Last Name:</label>
  <input
    type="text"
    id="lastName"
    name="lastName"
    [(ngModel)]="user.lastName"
    #lastName="ngModel"
    required
    class="form-control"
  >
</div>

<!-- Email Field -->
<div class="form-group">
  <label for="email">Email:</label>
  <input
    type="email"
    id="email"
    name="email"
    [(ngModel)]="user.email"
    #email="ngModel"
    required
    email
    class="form-control"
  >
</div>

<!-- Age Field -->
<div class="form-group">
  <label for="age">Age:</label>
  <input
    type="number"
    id="age"
    name="age"
    [(ngModel)]="user.age"
    #age="ngModel"
    required
    min="18"
    max="100"
    class="form-control"
  >
</div>

<!-- Address Field -->
<div class="form-group">
  <label for="address">Address:</label>
  <textarea
    id="address"
```

```
        name="address"
        [(ngModel)]="user.address"
        #address="ngModel"
        required
        minlength="10"
        rows="4"
        class="form-control"
      ></textarea>
    </div>

    <!-- Form Buttons -->
    <div class="form-actions">
      <button
        type="submit"
        [disabled]="!userForm.valid"
        class="btn btn-primary"
      >
        Submit
      </button>
      <button
        type="button"
        (click)="resetForm(userForm)"
        class="btn btn-secondary"
      >
        Reset
      </button>
    </div>
  </form>

  <!-- Display submitted data -->
  <div *ngIf="submitted" class="success-message">
    <h3>Form Submitted Successfully!</h3>
    <p><strong>Name:</strong> {{ user.firstName }} {{ user.lastName }}</p>
    <p><strong>Email:</strong> {{ user.email }}</p>
    <p><strong>Age:</strong> {{ user.age }}</p>
    <p><strong>Address:</strong> {{ user.address }}</p>
  </div>
</div>
```

2.3 Key Directives and Concepts

ngModel Directive

`ngModel` creates a two-way data binding between the input element and the component property.

```
<!-- Two-way binding -->
<input [(ngModel)]="user.firstName" name="firstName">

<!-- Is equivalent to: -->
<input
  [ngModel]="user.firstName"
  (ngModelChange)="user.firstName = $event"
  name="firstName"
>
```

Important: The `name` attribute is **mandatory** when using `ngModel` in a form.

Template Reference Variables

```
<!-- #userForm creates a reference to the form -->
<form #userForm="ngForm">

<!-- #firstName creates a reference to the input control -->
<input #firstName="ngModel" name="firstName" [(ngModel)]="user.firstName">
```

ngForm Directive

Angular automatically attaches the `ngForm` directive to `<form>` tags. It:

- Tracks form value and validity
- Provides form-level validation
- Exposes properties like `valid`, `invalid`, `pristine`, `dirty`, `touched`, `untouched`

2.4 Form State Properties

Property	Type	Description
`valid`	boolean	True if all controls pass validation
`invalid`	boolean	True if any control fails validation
`pristine`	boolean	True if user hasn't changed any value
`dirty`	boolean	True if user has changed a value
`touched`	boolean	True if user has focused and blurred the control
`untouched`	boolean	True if user hasn't focused the control
`value`	object	Object containing all form values
`errors`	object	Object containing validation errors

Example: Using Form State

```
<form #registrationForm="ngForm">
  <!-- Form fields here -->
</form>

<!-- Display form state -->
<div class="debug-info">
  <h4>Form State Debug Information:</h4>
  <p>Form Valid: {{ registrationForm.valid }}</p>
  <p>Form Invalid: {{ registrationForm.invalid }}</p>
  <p>Form Pristine: {{ registrationForm.pristine }}</p>
  <p>Form Dirty: {{ registrationForm.dirty }}</p>
  <p>Form Touched: {{ registrationForm.touched }}</p>
  <p>Form Value: {{ registrationForm.value | json }}</p>
</div>
```

---

3. Form Validation {#form-validation}

3.1 Built-in Validators

Angular provides several built-in validators:

Validator	Description	Example
`required`	Field must have a value	`<input required>`
`minlength`	Minimum string length	`<input minlength="5">`
`maxlength`	Maximum string length	`<input maxlength="20">`
`min`	Minimum numeric value	`<input type="number" min="0">`
`max`	Maximum numeric value	`<input type="number" max="100">`
`pattern`	Must match regex pattern	`<input pattern="[0-9]{3}">`
`email`	Must be valid email format	`<input type="email" email>`

3.2 Displaying Validation Errors

Basic Error Display

```
<!-- registration-form.component.html -->
<div class="form-group">
  <label for="email">Email Address:</label>
  <input
    type="email"
    id="email"
    name="email"
    [(ngModel)]="userEmail"
    #email="ngModel"
    required
    email
    class="form-control"
    [class.is-invalid]="email.invalid && email.touched"
  >

  <!-- Display error messages -->
  <div *ngIf="email.invalid && email.touched" class="error-message">
    <small *ngIf="email.errors?.['required']">Email is required.</small>
    <small *ngIf="email.errors?.['email']">Please enter a valid email address.</small>
  </div>
</div>
```

3.3 Comprehensive Validation Example

Component TypeScript

```
// registration.component.ts
import { Component } from '@angular/core';

interface RegistrationData {
  username: string;
  email: string;
  password: string;
  confirmPassword: string;
  phone: string;
  age: number;
  website: string;
  bio: string;
  termsAccepted: boolean;
  gender: string;
  country: string;
}

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

  registration: RegistrationData = {
    username: '',
    email: '',
    password: '',
    confirmPassword: '',
    phone: '',
    age: 0,
    website: '',
    bio: '',
    termsAccepted: false,
    gender: '',
    country: ''
  };

  countries: string[] = ['USA', 'UK', 'Canada', 'India', 'Australia'];
  submitted = false;

  onSubmit(form: any): void {
    if (form.valid) {
      this.submitted = true;
      console.log('Registration Data:', this.registration);
      // API call would go here
    } else {
      // Mark all fields as touched to show validation errors
      Object.keys(form.controls).forEach(key => {
        form.controls[key].markAsTouched();
      });
    }
  }

  resetForm(form: any): void {
    form.reset();
    this.submitted = false;
  }
}
```

Component Template

```
<!-- registration.component.html -->
<div class="registration-container">
  <h2>Complete Registration Form with Validation</h2>

  <form #regForm="ngForm" (ngSubmit)="onSubmit(regForm)" novalidate>

    <!-- Username Field -->
    <div class="form-group">
      <label for="username">Username: <span class="required">*</span></label>
      <input
        type="text"
        id="username"
        name="username"
        [(ngModel)]="registration.username"
        #username="ngModel"
        required
        minlength="3"
        maxlength="20"
        pattern="^[a-zA-Z0-9_]+$"
        class="form-control"
        [class.is-invalid]="username.invalid && username.touched"
        [class.is-valid]="username.valid && username.touched"
      >
      <div *ngIf="username.invalid && username.touched" class="validation-errors">
        <small *ngIf="username.errors?.['required']">Username is required</small>
        <small *ngIf="username.errors?.['minlength']">
          Username must be at least 3 characters (current: {{ username.value?.length || 0
}})
        </small>
        <small *ngIf="username.errors?.['maxlength']">
          Username cannot exceed 20 characters
        </small>
        <small *ngIf="username.errors?.['pattern']">
          Username can only contain letters, numbers, and underscores
        </small>
      </div>
    </div>

    <!-- Email Field -->
    <div class="form-group">
      <label for="email">Email Address: <span class="required">*</span></label>
      <input
        type="email"
        id="email"
        name="email"
        [(ngModel)]="registration.email"
        #email="ngModel"
        required
        email
        class="form-control"
        [class.is-invalid]="email.invalid && email.touched"
        [class.is-valid]="email.valid && email.touched"
      >
      <div *ngIf="email.invalid && email.touched" class="validation-errors">
        <small *ngIf="email.errors?.['required']">Email is required</small>
        <small *ngIf="email.errors?.['email']">Please enter a valid email address</small>
      </div>
    </div>

    <!-- Password Field -->
    <div class="form-group">
      <label for="password">Password: <span class="required">*</span></label>
      <input
        type="password"
        id="password"
        name="password"
        [(ngModel)]="registration.password"
        #password="ngModel"
        required
        minlength="8"
        pattern="^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$"
        class="form-control"
        [class.is-invalid]="password.invalid && password.touched"
        [class.is-valid]="password.valid && password.touched"
      >
      <div *ngIf="password.invalid && password.touched" class="validation-errors">
        <small *ngIf="password.errors?.['required']">Password is required</small>
        <small *ngIf="password.errors?.['minlength']">
          Password must be at least 8 characters
        </small>
        <small *ngIf="password.errors?.['pattern']">
          Password must contain at least one uppercase letter, one lowercase letter,
          one number, and one special character
        </small>
      </div>
```

```
      <small class="form-hint">
        Password requirements: minimum 8 characters, 1 uppercase, 1 lowercase, 1 number, 1 
special character
      </small>
    </div>

    <!-- Confirm Password Field -->
    <div class="form-group">
      <label for="confirmPassword">Confirm Password: <span class="required">*</span></label>
      <input
        type="password"
        id="confirmPassword"
        name="confirmPassword"
        [(ngModel)]="registration.confirmPassword"
        #confirmPassword="ngModel"
        required
        class="form-control"
        [class.is-invalid]="confirmPassword.invalid && confirmPassword.touched"
        [class.is-valid]="confirmPassword.valid && confirmPassword.touched"
      >
      <div *ngIf="confirmPassword.touched && registration.password !== 
registration.confirmPassword"
           class="validation-errors">
        <small>Passwords do not match</small>
      </div>
    </div>

    <!-- Phone Number Field -->
    <div class="form-group">
      <label for="phone">Phone Number: <span class="required">*</span></label>
      <input
        type="tel"
        id="phone"
        name="phone"
        [(ngModel)]="registration.phone"
        #phone="ngModel"
        required
        pattern="^[\+]?[(]?[0-9]{3}[)]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}$"
        class="form-control"
        [class.is-invalid]="phone.invalid && phone.touched"
        [class.is-valid]="phone.valid && phone.touched"
        placeholder="123-456-7890"
      >
      <div *ngIf="phone.invalid && phone.touched" class="validation-errors">
        <small *ngIf="phone.errors?.['required']">Phone number is required</small>
        <small *ngIf="phone.errors?.['pattern']">
          Please enter a valid phone number (e.g., 123-456-7890)
        </small>
      </div>
    </div>

    <!-- Age Field -->
```

```html
<div class="form-group">
  <label for="age">Age: <span class="required">*</span></label>
  <input
    type="number"
    id="age"
    name="age"
    [(ngModel)]="registration.age"
    #age="ngModel"
    required
    min="18"
    max="120"
    class="form-control"
    [class.is-invalid]="age.invalid && age.touched"
    [class.is-valid]="age.valid && age.touched"
  >
  <div *ngIf="age.invalid && age.touched" class="validation-errors">
    <small *ngIf="age.errors?.['required']">Age is required</small>
    <small *ngIf="age.errors?.['min']">You must be at least 18 years old</small>
    <small *ngIf="age.errors?.['max']">Please enter a valid age</small>
  </div>
</div>

<!-- Website Field -->
<div class="form-group">
  <label for="website">Website:</label>
  <input
    type="url"
    id="website"
    name="website"
    [(ngModel)]="registration.website"
    #website="ngModel"
    pattern="https?://.+"
    class="form-control"
    [class.is-invalid]="website.invalid && website.touched"
    [class.is-valid]="website.valid && website.touched"
    placeholder="https://example.com"
  >
  <div *ngIf="website.invalid && website.touched" class="validation-errors">
    <small *ngIf="website.errors?.['pattern']">
      Please enter a valid URL (must start with http:// or https://)
    </small>
  </div>
</div>

<!-- Gender Radio Buttons -->
<div class="form-group">
  <label>Gender: <span class="required">*</span></label>
  <div class="radio-group">
    <label class="radio-label">
      <input
        type="radio"
        name="gender"
        [(ngModel)]="registration.gender"
        value="male"
        required
        #gender="ngModel"
      >
      Male
    </label>
    <label class="radio-label">
      <input
        type="radio"
        name="gender"
        [(ngModel)]="registration.gender"
        value="female"
        required
      >
      Female
    </label>
    <label class="radio-label">
      <input
        type="radio"
        name="gender"
        [(ngModel)]="registration.gender"
        value="other"
        required
      >
      Other
    </label>
  </div>
  <div *ngIf="gender.invalid && gender.touched" class="validation-errors">
    <small>Please select a gender</small>
  </div>
</div>

<!-- Country Dropdown -->
<div class="form-group">
  <label for="country">Country: <span class="required">*</span></label>
  <select
    id="country"
    name="country"
    [(ngModel)]="registration.country"
    #country="ngModel"
    required
    class="form-control"
    [class.is-invalid]="country.invalid && country.touched"
    [class.is-valid]="country.valid && country.touched"
  >
    <option value="">Select a country</option>
    <option *ngFor="let c of countries" [value]="c">{{ c }}</option>
  </select>
  <div *ngIf="country.invalid && country.touched" class="validation-errors">
    <small>Please select a country</small>
  </div>
```

```html
</div>

<!-- Bio Textarea -->
<div class="form-group">
  <label for="bio">Bio: <span class="required">*</span></label>
  <textarea
    id="bio"
    name="bio"
    [(ngModel)]="registration.bio"
    #bio="ngModel"
    required
    minlength="50"
    maxlength="500"
    rows="5"
    class="form-control"
    [class.is-invalid]="bio.invalid && bio.touched"
    [class.is-valid]="bio.valid && bio.touched"
  ></textarea>
  <small class="char-count">
    {{ bio.value?.length || 0 }} / 500 characters
    (minimum 50 required)
  </small>
  <div *ngIf="bio.invalid && bio.touched" class="validation-errors">
    <small *ngIf="bio.errors?.['required']">Bio is required</small>
    <small *ngIf="bio.errors?.['minlength']">
      Bio must be at least 50 characters
    </small>
  </div>
</div>

<!-- Terms and Conditions Checkbox -->
<div class="form-group">
  <label class="checkbox-label">
    <input
      type="checkbox"
      name="termsAccepted"
      [(ngModel)]="registration.termsAccepted"
      #terms="ngModel"
      required
    >
    I accept the Terms and Conditions <span class="required">*</span>
  </label>
  <div *ngIf="terms.invalid && terms.touched" class="validation-errors">
    <small>You must accept the terms and conditions</small>
  </div>
</div>

<!-- Form Buttons -->
<div class="form-actions">
  <button
    type="submit"
    class="btn btn-primary"
```

```
          [disabled]="regForm.invalid"
        >
          Register
        </button>
        <button
          type="button"
          (click)="resetForm(regForm)"
          class="btn btn-secondary"
        >
          Reset Form
        </button>
      </div>

      <!-- Form Status Display -->
      <div class="form-status">
        <p>Form Valid: <strong>{{ regForm.valid ? 'Yes' : 'No' }}</strong></p>
        <p>Form Touched: <strong>{{ regForm.touched ? 'Yes' : 'No' }}</strong></p>
        <p>Form Dirty: <strong>{{ regForm.dirty ? 'Yes' : 'No' }}</strong></p>
      </div>
    </form>

    <!-- Success Message -->
    <div *ngIf="submitted" class="success-card">
      <h3>✓ Registration Successful!</h3>
      <div class="submitted-data">
        <p><strong>Username:</strong> {{ registration.username }}</p>
        <p><strong>Email:</strong> {{ registration.email }}</p>
        <p><strong>Phone:</strong> {{ registration.phone }}</p>
        <p><strong>Age:</strong> {{ registration.age }}</p>
        <p><strong>Gender:</strong> {{ registration.gender }}</p>
        <p><strong>Country:</strong> {{ registration.country }}</p>
        <p><strong>Website:</strong> {{ registration.website || 'Not provided' }}</p>
        <p><strong>Bio:</strong> {{ registration.bio }}</p>
      </div>
    </div>
  </div>
```

3.4 Validation CSS Styling

```
/* registration.component.css */

.registration-container {
  max-width: 600px;
  margin: 40px auto;
  padding: 30px;
  background-color: #f8f9fa;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}
```

```css
h2 {
  color: #333;
  margin-bottom: 30px;
  text-align: center;
}

.form-group {
  margin-bottom: 20px;
}

label {
  display: block;
  margin-bottom: 5px;
  color: #555;
  font-weight: 500;
}

.required {
  color: #dc3545;
}

.form-control {
  width: 100%;
  padding: 10px;
  border: 2px solid #ddd;
  border-radius: 4px;
  font-size: 14px;
  transition: border-color 0.3s;
}

.form-control:focus {
  outline: none;
  border-color: #007bff;
}

.form-control.is-invalid {
  border-color: #dc3545;
  background-color: #fff5f5;
}

.form-control.is-valid {
  border-color: #28a745;
  background-color: #f0fff4;
}

.validation-errors {
  margin-top: 5px;
}

.validation-errors small {
  display: block;
  color: #dc3545;
  font-size: 12px;
  margin-top: 3px;
}

.form-hint {
  display: block;
  color: #6c757d;
  font-size: 12px;
  margin-top: 5px;
  font-style: italic;
}

.char-count {
  display: block;
  text-align: right;
  color: #6c757d;
  font-size: 12px;
  margin-top: 5px;
}

.radio-group {
  display: flex;
  gap: 20px;
  margin-top: 8px;
}

.radio-label {
  display: flex;
  align-items: center;
  gap: 5px;
  font-weight: normal;
  cursor: pointer;
}

.radio-label input[type="radio"] {
  cursor: pointer;
}

.checkbox-label {
  display: flex;
  align-items: center;
  gap: 8px;
  font-weight: normal;
  cursor: pointer;
}

.checkbox-label input[type="checkbox"] {
  cursor: pointer;
  width: 18px;
  height: 18px;
}
```

```css
.form-actions {
  display: flex;
  gap: 15px;
  margin-top: 30px;
}

.btn {
  padding: 12px 30px;
  border: none;
  border-radius: 4px;
  font-size: 16px;
  cursor: pointer;
  transition: all 0.3s;
}

.btn-primary {
  background-color: #007bff;
  color: white;
}

.btn-primary:hover:not(:disabled) {
  background-color: #0056b3;
}

.btn-primary:disabled {
  background-color: #6c757d;
  cursor: not-allowed;
  opacity: 0.6;
}

.btn-secondary {
  background-color: #6c757d;
  color: white;
}

.btn-secondary:hover {
  background-color: #545b62;
}

.form-status {
  margin-top: 20px;
  padding: 15px;
  background-color: #e9ecef;
  border-radius: 4px;
  font-size: 14px;
}

.success-card {
  margin-top: 30px;
  padding: 25px;
  background-color: #d4edda;
  border: 2px solid #28a745;
  border-radius: 8px;
  animation: slideIn 0.5s ease-out;
}

.success-card h3 {
  color: #155724;
  margin-bottom: 20px;
}

.submitted-data p {
  margin: 8px 0;
  color: #155724;
}

@keyframes slideIn {
  from {
    opacity: 0;
    transform: translateY(-20px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}
```

---

4. Custom Validation {#custom-validation}

4.1 Creating Custom Validators

Custom validators are created as **Directives** in Angular template-driven forms.

4.1.1 Simple Custom Validator - No Whitespace

```
// no-whitespace.validator.ts
import { Directive } from '@angular/core';
import { NG_VALIDATORS, Validator, AbstractControl, ValidationErrors } from
'@angular/forms';

@Directive({
  selector: '[appNoWhitespace]',
  providers: [
    {
      provide: NG_VALIDATORS,
      useExisting: NoWhitespaceValidatorDirective,
      multi: true
    }
  ]
})
export class NoWhitespaceValidatorDirective implements Validator {

  validate(control: AbstractControl): ValidationErrors | null {
    // Check if control has a value
    if (!control.value) {
      return null; // Don't validate empty values
    }

    // Check if value contains only whitespace
    const isWhitespace = (control.value || '').trim().length === 0;

    return isWhitespace ? { whitespace: true } : null;
  }
}
```

Usage in Template:

```
<input
  type="text"
  name="username"
  [(ngModel)]="username"
  #usernameField="ngModel"
  appNoWhitespace
  required
>
<div *ngIf="usernameField.errors?.['whitespace']" class="error">
  Field cannot contain only whitespace
</div>
```

4.1.2 Custom Validator with Parameter - Min Age

```
// min-age.validator.ts
import { Directive, Input } from '@angular/core';
import { NG_VALIDATORS, Validator, AbstractControl, ValidationErrors } from
'@angular/forms';

@Directive({
  selector: '[appMinAge]',
  providers: [
    {
      provide: NG_VALIDATORS,
      useExisting: MinAgeValidatorDirective,
      multi: true
    }
  ]
})
export class MinAgeValidatorDirective implements Validator {

  @Input('appMinAge') minAge: number = 18;

  validate(control: AbstractControl): ValidationErrors | null {
    if (!control.value) {
      return null;
    }

    const birthDate = new Date(control.value);
    const today = new Date();

    // Calculate age
    let age = today.getFullYear() - birthDate.getFullYear();
    const monthDiff = today.getMonth() - birthDate.getMonth();

    if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < birthDate.getDate())) {
      age--;
    }

    return age < this.minAge
      ? { minAge: { requiredAge: this.minAge, actualAge: age } }
      : null;
  }
}
```

Usage in Template:

```
<input
  type="date"
  name="birthDate"
  [(ngModel)]="birthDate"
  #birthDateField="ngModel"
  [appMinAge]="21"
>
<div *ngIf="birthDateField.errors?.['minAge']" class="error">
  You must be at least {{ birthDateField.errors?.['minAge'].requiredAge }} years old.
  Current age: {{ birthDateField.errors?.['minAge'].actualAge }}
</div>
```

4.1.3 Password Match Validator (Cross-Field Validation)

```
// password-match.validator.ts
import { Directive, Input } from '@angular/core';
import { NG_VALIDATORS, Validator, AbstractControl, ValidationErrors } from
'@angular/forms';

@Directive({
  selector: '[appPasswordMatch]',
  providers: [
    {
      provide: NG_VALIDATORS,
      useExisting: PasswordMatchValidatorDirective,
      multi: true
    }
  ]
})
```

```
export class PasswordMatchValidatorDirective implements Validator {

  @Input('appPasswordMatch') passwordToMatch: string = '';

  validate(control: AbstractControl): ValidationErrors | null {
    if (!control.value) {
      return null;
    }

    // Check if passwords match
    const passwordsMatch = control.value === this.passwordToMatch;

    return !passwordsMatch ? { passwordMismatch: true } : null;
  }
}
```

Usage in Template:

```
<!-- Password field -->
<input
  type="password"
  name="password"
  [(ngModel)]="password"
  #passwordField="ngModel"
  required
>

<!-- Confirm password field -->
<input
  type="password"
  name="confirmPassword"
  [(ngModel)]="confirmPassword"
  #confirmPasswordField="ngModel"
  [appPasswordMatch]="password"
  required
>
<div *ngIf="confirmPasswordField.errors?.['passwordMismatch'] &&
confirmPasswordField.touched" class="error">
  Passwords do not match
</div>
```

4.1.4 Email Domain Validator

```
// email-domain.validator.ts
import { Directive, Input } from '@angular/core';
import { NG_VALIDATORS, Validator, AbstractControl, ValidationErrors } from
'@angular/forms';

@Directive({
  selector: '[appEmailDomain]',
  providers: [
    {
      provide: NG_VALIDATORS,
      useExisting: EmailDomainValidatorDirective,
      multi: true
    }
  ]
})
export class EmailDomainValidatorDirective implements Validator {

  @Input('appEmailDomain') allowedDomains: string[] = [];

  validate(control: AbstractControl): ValidationErrors | null {
    if (!control.value || this.allowedDomains.length === 0) {
      return null;
    }

    const email = control.value as string;
    const domain = email.split('@')[1];

    if (!domain) {
      return null; // Let email validator handle format validation
    }

    const isDomainAllowed = this.allowedDomains.some(
      allowedDomain => domain.toLowerCase() === allowedDomain.toLowerCase()
    );

    return !isDomainAllowed
      ? {
          emailDomain: {
            actualDomain: domain,
            allowedDomains: this.allowedDomains
          }
        }
      : null;
  }
}
```

Usage in Template:

```
<input
  type="email"
  name="email"
  [(ngModel)]="email"
  #emailField="ngModel"
  [appEmailDomain]="['company.com', 'example.com']"
  email
  required
>
<div *ngIf="emailField.errors?.['emailDomain']" class="error">
```

```
  Email must be from one of these domains:
  {{ emailField.errors?.['emailDomain'].allowedDomains.join(', ') }}
</div>
```

4.1.5 Username Availability Validator (Async Validator)

```
// username-availability.validator.ts
import { Directive } from '@angular/core';
import { NG_ASYNC_VALIDATORS, AsyncValidator, AbstractControl, ValidationErrors } from
'@angular/forms';
import { Observable, of } from 'rxjs';
import { map, catchError, debounceTime, switchMap } from 'rxjs/operators';
import { Injectable } from '@angular/core';

// Mock service - replace with actual HTTP service
@Injectable({
  providedIn: 'root'
})
export class UsernameService {
  checkUsernameAvailability(username: string): Observable<boolean> {
    // Simulate API call
    return of(['admin', 'user', 'test'].includes(username.toLowerCase())).pipe(
      map(exists => !exists)
    );
  }
}

@Directive({
  selector: '[appUsernameAvailable]',
  providers: [
    {
      provide: NG_ASYNC_VALIDATORS,
      useExisting: UsernameAvailabilityValidatorDirective,
      multi: true
    }
  ]
})
export class UsernameAvailabilityValidatorDirective implements AsyncValidator {

  constructor(private usernameService: UsernameService) {}

  validate(control: AbstractControl): Observable<ValidationErrors | null> {
    if (!control.value) {
      return of(null);
    }

    return of(control.value).pipe(
      debounceTime(500), // Wait 500ms after user stops typing
      switchMap(username =>
        this.usernameService.checkUsernameAvailability(username).pipe(
          map(isAvailable => isAvailable ? null : { usernameTaken: true }),
          catchError(() => of(null))
        )
      )
    );
  }
}
```

Usage in Template:

```
<input
  type="text"
  name="username"
  [(ngModel)]="username"
  #usernameField="ngModel"
  appUsernameAvailable
  required
>
<div *ngIf="usernameField.pending" class="info">
  Checking username availability...
</div>
<div *ngIf="usernameField.errors?.['usernameTaken']" class="error">
  This username is already taken
</div>
```

4.1.6 Credit Card Validator

```
// credit-card.validator.ts
import { Directive } from '@angular/core';
import { NG_VALIDATORS, Validator, AbstractControl, ValidationErrors } from
'@angular/forms';

@Directive({
  selector: '[appCreditCard]',
  providers: [
    {
      provide: NG_VALIDATORS,
      useExisting: CreditCardValidatorDirective,
      multi: true
    }
  ]
})
export class CreditCardValidatorDirective implements Validator {

  validate(control: AbstractControl): ValidationErrors | null {
    if (!control.value) {
      return null;
    }

    const cardNumber = control.value.replace(/\s/g, ''); // Remove spaces
    // Check if it contains only digits
    if (!/^\d+$/.test(cardNumber)) {
      return { creditCard: { message: 'Card number must contain only digits' } };
    }

    // Luhn Algorithm for credit card validation
    let sum = 0;
    let isEven = false;

    for (let i = cardNumber.length - 1; i >= 0; i--) {
      let digit = parseInt(cardNumber[i], 10);

      if (isEven) {
        digit *= 2;
        if (digit > 9) {
          digit -= 9;
        }
      }

      sum += digit;
      isEven = !isEven;
    }

    const isValid = sum % 10 === 0;

    return isValid ? null : { creditCard: { message: 'Invalid credit card number' } };
  }
}
```

Usage in Template:

```
<input
  type="text"
  name="cardNumber"
  [(ngModel)]="cardNumber"
  #cardField="ngModel"
  appCreditCard
  required
  maxlength="19"
  placeholder="1234 5678 9012 3456"
>
<div *ngIf="cardField.errors?.['creditCard']" class="error">
  {{ cardField.errors?.['creditCard'].message }}
</div>
```

4.1.7 URL Validator with Protocol Check

```
// url-protocol.validator.ts
import { Directive, Input } from '@angular/core';
```

```typescript
import { NG_VALIDATORS, Validator, AbstractControl, ValidationErrors } from
'@angular/forms';

@Directive({
  selector: '[appUrlProtocol]',
  providers: [
    {
      provide: NG_VALIDATORS,
      useExisting: UrlProtocolValidatorDirective,
      multi: true
    }
  ]
})
export class UrlProtocolValidatorDirective implements Validator {

  @Input('appUrlProtocol') requiredProtocol: 'http' | 'https' | 'both' = 'both';

  validate(control: AbstractControl): ValidationErrors | null {
    if (!control.value) {
      return null;
    }

    const url = control.value as string;
    const hasHttp = url.startsWith('http://');
    const hasHttps = url.startsWith('https://');

    let isValid = false;

    switch (this.requiredProtocol) {
      case 'http':
        isValid = hasHttp;
        break;
      case 'https':
        isValid = hasHttps;
        break;
      case 'both':
        isValid = hasHttp || hasHttps;
        break;
    }

    return !isValid
      ? {
          urlProtocol: {
            required: this.requiredProtocol,
            actual: hasHttp ? 'http' : (hasHttps ? 'https' : 'none')
          }
        }
      : null;
  }
}
```

Usage in Template:

```html
<input
  type="url"
  name="website"
  [(ngModel)]="website"
  #websiteField="ngModel"
  appUrlProtocol="https"
>
<div *ngIf="websiteField.errors?.['urlProtocol']" class="error">
  URL must use HTTPS protocol
</div>
```

4.2 Complete Custom Validation Example

Module Registration

```typescript
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { CustomFormComponent } from './custom-form/custom-form.component';

// Import all custom validators
import { NoWhitespaceValidatorDirective } from './validators/no-whitespace.validator';
import { MinAgeValidatorDirective } from './validators/min-age.validator';
import { PasswordMatchValidatorDirective } from './validators/password-match.validator';
import { EmailDomainValidatorDirective } from './validators/email-domain.validator';
import { UsernameAvailabilityValidatorDirective } from './validators/username-
availability.validator';
import { CreditCardValidatorDirective } from './validators/credit-card.validator';
import { UrlProtocolValidatorDirective } from './validators/url-protocol.validator';

@NgModule({
  declarations: [
    AppComponent,
    CustomFormComponent,
    // Register all custom validators
    NoWhitespaceValidatorDirective,
    MinAgeValidatorDirective,
    PasswordMatchValidatorDirective,
    EmailDomainValidatorDirective,
    UsernameAvailabilityValidatorDirective,
    CreditCardValidatorDirective,
    UrlProtocolValidatorDirective
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Complete Form with Custom Validators

```
// custom-form.component.ts
import { Component } from '@angular/core';

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

  formData = {
    username: '',
    email: '',
    password: '',
    confirmPassword: '',
    birthDate: '',
    website: '',
    cardNumber: ''
  };

  allowedEmailDomains = ['company.com', 'example.org'];
  submitted = false;

  onSubmit(form: any): void {
    if (form.valid) {
      this.submitted = true;
      console.log('Form Data:', this.formData);
    } else {
      this.markFormGroupTouched(form);
    }
  }

  markFormGroupTouched(form: any): void {
    Object.keys(form.controls).forEach(key => {
      form.controls[key].markAsTouched();
    });
  }

  resetForm(form: any): void {
    form.reset();
    this.submitted = false;
  }
}
```

```html
<!-- custom-form.component.html -->
<div class="custom-form-container">
  <h2>Advanced Form with Custom Validators</h2>

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

    <!-- Username with No Whitespace and Async Validation -->
    <div class="form-group">
      <label for="username">Username:</label>
      <input
        type="text"
        id="username"
        name="username"
        [(ngModel)]="formData.username"
        #username="ngModel"
        appNoWhitespace
        appUsernameAvailable
        required
        minlength="3"
        class="form-control"
        [class.is-invalid]="username.invalid && username.touched"
      >
      <div class="validation-feedback">
        <div *ngIf="username.pending" class="checking">
          ⏳ Checking username availability...
        </div>
        <div *ngIf="username.invalid && username.touched" class="errors">
          <small *ngIf="username.errors?.['required']">Username is required</small>
          <small *ngIf="username.errors?.['minlength']">Minimum 3 characters
required</small>
          <small *ngIf="username.errors?.['whitespace']">Cannot contain only
whitespace</small>
          <small *ngIf="username.errors?.['usernameTaken']">Username is already
taken</small>
        </div>
        <div *ngIf="username.valid && username.touched" class="success">
          ✓ Username is available
        </div>
      </div>
    </div>

    <!-- Email with Domain Restriction -->
    <div class="form-group">
      <label for="email">Corporate Email:</label>
      <input
        type="email"
        id="email"
        name="email"
        [(ngModel)]="formData.email"
        #email="ngModel"
        [appEmailDomain]="allowedEmailDomains"
        email
        required
        class="form-control"
        [class.is-invalid]="email.invalid && email.touched"
      >
      <small class="hint">
        Allowed domains: {{ allowedEmailDomains.join(', ') }}
      </small>
      <div *ngIf="email.invalid && email.touched" class="errors">
        <small *ngIf="email.errors?.['required']">Email is required</small>
        <small *ngIf="email.errors?.['email']">Invalid email format</small>
        <small *ngIf="email.errors?.['emailDomain']">
          Email must be from allowed domains: {{ allowedEmailDomains.join(', ') }}
        </small>
      </div>
    </div>

    <!-- Password and Confirm Password -->
    <div class="form-group">
      <label for="password">Password:</label>
      <input
        type="password"
        id="password"
        name="password"
        [(ngModel)]="formData.password"
        #password="ngModel"
        required
        minlength="8"
        class="form-control"
        [class.is-invalid]="password.invalid && password.touched"
      >
      <div *ngIf="password.invalid && password.touched" class="errors">
        <small *ngIf="password.errors?.['required']">Password is required</small>
        <small *ngIf="password.errors?.['minlength']">Minimum 8 characters required</small>
      </div>
    </div>

    <div class="form-group">
      <label for="confirmPassword">Confirm Password:</label>
      <input
        type="password"
        id="confirmPassword"
        name="confirmPassword"
        [(ngModel)]="formData.confirmPassword"
        #confirmPassword="ngModel"
        [appPasswordMatch]="formData.password"
        required
        class="form-control"
        [class.is-invalid]="confirmPassword.invalid && confirmPassword.touched"
      >
      <div *ngIf="confirmPassword.invalid && confirmPassword.touched" class="errors">
        <small *ngIf="confirmPassword.errors?.['required']">Please confirm password</small>
```

```
        <small *ngIf="confirmPassword.errors?.['passwordMismatch']">Passwords do not
match</small>
      </div>
    </div>

    <!-- Birth Date with Age Validation -->
    <div class="form-group">
      <label for="birthDate">Birth Date (Must be 21+):</label>
      <input
        type="date"
        id="birthDate"
        name="birthDate"
        [(ngModel)]="formData.birthDate"
        #birthDate="ngModel"
        [appMinAge]="21"
        required
        class="form-control"
        [class.is-invalid]="birthDate.invalid && birthDate.touched"
      >
      <div *ngIf="birthDate.invalid && birthDate.touched" class="errors">
        <small *ngIf="birthDate.errors?.['required']">Birth date is required</small>
        <small *ngIf="birthDate.errors?.['minAge']">
          You must be at least {{ birthDate.errors?.['minAge'].requiredAge }} years old
          (Current age: {{ birthDate.errors?.['minAge'].actualAge }})
        </small>
      </div>
    </div>

    <!-- Website with HTTPS Protocol -->
    <div class="form-group">
      <label for="website">Website (HTTPS only):</label>
      <input
        type="url"
        id="website"
        name="website"
        [(ngModel)]="formData.website"
        #website="ngModel"
        appUrlProtocol="https"
        class="form-control"
        [class.is-invalid]="website.invalid && website.touched"
        placeholder="https://example.com"
      >
      <div *ngIf="website.invalid && website.touched" class="errors">
        <small *ngIf="website.errors?.['urlProtocol']">
          Website must use HTTPS protocol
        </small>
      </div>
    </div>

    <!-- Credit Card with Luhn Algorithm -->
    <div class="form-group">
      <label for="cardNumber">Credit Card Number:</label>
```

```html
      <input
        type="text"
        id="cardNumber"
        name="cardNumber"
        [(ngModel)]="formData.cardNumber"
        #cardNumber="ngModel"
        appCreditCard
        required
        class="form-control"
        [class.is-invalid]="cardNumber.invalid && cardNumber.touched"
        placeholder="1234 5678 9012 3456"
      >
      <div *ngIf="cardNumber.invalid && cardNumber.touched" class="errors">
        <small *ngIf="cardNumber.errors?.['required']">Card number is required</small>
        <small *ngIf="cardNumber.errors?.['creditCard']">
          {{ cardNumber.errors?.['creditCard'].message }}
        </small>
      </div>
    </div>

    <!-- Submit Button -->
    <div class="form-actions">
      <button
        type="submit"
        [disabled]="customForm.invalid || customForm.pending"
        class="btn btn-primary"
      >
        <span *ngIf="!customForm.pending">Submit</span>
        <span *ngIf="customForm.pending">Validating...</span>
      </button>
      <button
        type="button"
        (click)="resetForm(customForm)"
        class="btn btn-secondary"
      >
        Reset
      </button>
    </div>

    <!-- Debug Info -->
    <div class="debug-panel">
      <h4>Form Status</h4>
      <p>Valid: {{ customForm.valid }}</p>
      <p>Invalid: {{ customForm.invalid }}</p>
      <p>Pending: {{ customForm.pending }}</p>
      <p>Touched: {{ customForm.touched }}</p>
    </div>
  </form>

  <!-- Success Message -->
  <div *ngIf="submitted" class="success-message">
    <h3>✓ Form Submitted Successfully!</h3>
    <pre>{{ formData | json }}</pre>
  </div>
</div>
```

---

5. Advanced Concepts {#advanced-concepts}

5.1 Dynamic Form Fields

```
// dynamic-form.component.ts
import { Component } from '@angular/core';

interface DynamicField {
  id: number;
  value: string;
}

@Component({
  selector: 'app-dynamic-form',
  template: `
    <form #dynamicForm="ngForm">
      <div *ngFor="let field of fields; let i = index" class="dynamic-field">
        <input
          type="text"
          [name]="'field_' + field.id"
          [(ngModel)]="field.value"
          placeholder="Field {{ i + 1 }}"
          required
        >
        <button type="button" (click)="removeField(i)">Remove</button>
      </div>

      <button type="button" (click)="addField()">Add Field</button>

      <button
        type="submit"
        [disabled]="dynamicForm.invalid"
        (click)="onSubmit()"
      >
        Submit
      </button>
    </form>
  `
})
export class DynamicFormComponent {
  fields: DynamicField[] = [{ id: 1, value: '' }];
  nextId = 2;

  addField(): void {
    this.fields.push({ id: this.nextId++, value: '' });
  }
  removeField(index: number): void {
    this.fields.splice(index, 1);
  }

  onSubmit(): void {
    console.log('Form Values:', this.fields);
  }
}
```

5.2 Conditional Validation

```
<form #conditionalForm="ngForm">
  <!-- Shipping Address Checkbox -->
  <label>
    <input
      type="checkbox"
      name="needsShipping"
      [(ngModel)]="needsShipping"
    >
    Requires Shipping
  </label>

  <!-- Shipping fields only required if needsShipping is true -->
  <div *ngIf="needsShipping">
    <input
      type="text"
      name="shippingAddress"
      [(ngModel)]="shippingAddress"
      #shipping="ngModel"
      [required]="needsShipping"
    >
    <div *ngIf="shipping.invalid && shipping.touched">
      <small>Shipping address is required</small>
    </div>
  </div>
</form>
```

5.3 Form Groups with ngModelGroup

```
<form #userForm="ngForm">
  <div ngModelGroup="personalInfo" #personalInfo="ngModelGroup">
    <h3>Personal Information</h3>
    <input
      type="text"
      name="firstName"
      [(ngModel)]="user.personalInfo.firstName"
      required
    >
    <input
      type="text"
      name="lastName"
      [(ngModel)]="user.personalInfo.lastName"
      required
    >

    <div *ngIf="personalInfo.invalid && personalInfo.touched">
      Personal information is incomplete
    </div>
  </div>

  <div ngModelGroup="address" #address="ngModelGroup">
    <h3>Address</h3>
    <input
      type="text"
      name="street"
      [(ngModel)]="user.address.street"
      required
    >
    <input
      type="text"
      name="city"
      [(ngModel)]="user.address.city"
      required
    >

    <div *ngIf="address.invalid && address.touched">
      Address information is incomplete
    </div>
  </div>

  <button type="submit" [disabled]="userForm.invalid">Submit</button>
</form>
```

5.4 Accessing Form Data

```
// Component
onSubmit(form: NgForm): void {
  console.log('Form Valid:', form.valid);
  console.log('Form Value:', form.value);
  console.log('Form Errors:', form.errors);

  // Access specific control
  const emailControl = form.controls['email'];
  console.log('Email Valid:', emailControl.valid);
  console.log('Email Value:', emailControl.value);
  console.log('Email Errors:', emailControl.errors);
}
```

5.5 Debouncing Input Validation

```
// debounce-validator.directive.ts
import { Directive, Input, OnDestroy } from '@angular/core';
import { NG_VALIDATORS, Validator, AbstractControl, ValidationErrors } from
'@angular/forms';
import { Subject } from 'rxjs';
import { debounceTime } from 'rxjs/operators';

@Directive({
  selector: '[appDebounceValidation]',
  providers: [
    {
      provide: NG_VALIDATORS,
      useExisting: DebounceValidationDirective,
      multi: true
    }
  ]
})
export class DebounceValidationDirective implements Validator, OnDestroy {
  @Input() debounceTime: number = 500;

  private destroy$ = new Subject<void>();

  validate(control: AbstractControl): ValidationErrors | null {
    // Validation logic here
    return null;
  }

  ngOnDestroy(): void {
    this.destroy$.next();
    this.destroy$.complete();
  }
}
```

---

6. Best Practices {#best-practices}

6.1 Form Design Best Practices

✅ **Always provide clear labels**

```
<label for="email">Email Address:</label>
<input type="email" id="email" name="email">
```

✅ **Use appropriate input types**

```
<input type="email">    <!-- For emails -->
<input type="tel">      <!-- For phone numbers -->
<input type="number">   <!-- For numeric values -->
```

```
<input type="date">     <!-- For dates -->
<input type="url">      <!-- For URLs -->
```

✅ **Provide helpful placeholders**

```
<input
  type="tel"
  placeholder="123-456-7890"
  pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}"
>
```

✅ **Show validation errors clearly**

```
<div *ngIf="field.invalid && field.touched" class="error">
  <small *ngIf="field.errors?.['required']">This field is required</small>
</div>
```

6.2 Validation Best Practices

✅ **Use built-in validators when possible**

```
<input type="email" required email minlength="5">
```

✅ **Validate on blur, not on every keystroke**

```
<!-- Check errors only when field is touched -->
<div *ngIf="field.invalid && field.touched">
  <!-- Error messages -->
</div>
```

✅ **Provide real-time feedback for async validation**

```
<div *ngIf="username.pending">
  Checking availability...
</div>
```

✅ **Disable submit button when form is invalid**

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

6.3 Code Organization Best Practices

✅ **Keep validators in separate files**

```
src/
  app/
```

```
    validators/
      email-domain.validator.ts
      password-match.validator.ts
      credit-card.validator.ts
```

✅ **Create reusable validator directives**

```
// Use the same validator across multiple forms
@Directive({ selector: '[appNoWhitespace]' })
export class NoWhitespaceValidatorDirective { }
```

✅ **Use interfaces for form models**

```
interface UserForm {
  firstName: string;
  lastName: string;
  email: string;
}
```

6.4 Performance Best Practices

✅ **Use trackBy for dynamic form arrays**

```
<div *ngFor="let item of items; trackBy: trackByFn">
  <!-- Form fields -->
</div>
```

✅ **Debounce expensive validations**

```
// Use debounceTime for async validators
validate(control: AbstractControl): Observable<ValidationErrors | null> {
  return of(control.value).pipe(
    debounceTime(500),
    // Validation logic
  );
}
```

✅ **Lazy load large forms**

```
// Load form component only when needed
const routes: Routes = [
  {
    path: 'complex-form',
    loadChildren: () => import('./forms/forms.module').then(m => m.FormsModule)
  }
];
```

6.5 Accessibility Best Practices

✅ **Associate labels with inputs**

```
<label for="firstName">First Name:</label>
<input type="text" id="firstName" name="firstName">
```

✅ **Use ARIA attributes**

```
<input
  type="email"
  aria-required="true"
  aria-invalid="true"
  aria-describedby="emailError"
>
<div id="emailError" role="alert">
  Invalid email address
</div>
```

✅ **Provide keyboard navigation**

```
<form>
  <input type="text" tabindex="1">
  <input type="email" tabindex="2">
  <button type="submit" tabindex="3">Submit</button>
</form>
```

6.6 Security Best Practices

✅ **Never trust client-side validation alone**

- Always validate on the server side as well

✅ **Sanitize user input**

```
import { DomSanitizer } from '@angular/platform-browser';

constructor(private sanitizer: DomSanitizer) {}

sanitizeInput(value: string): string {
  return this.sanitizer.sanitize(SecurityContext.HTML, value) || '';
}
```

✅ **Use HTTPS for form submissions**

```
// In your service
submitForm(data: any): Observable<any> {
  return this.http.post('https://api.example.com/submit', data);
}
```

---

7. Common Patterns and Examples

7.1 Multi-Step Form

```
// multi-step-form.component.ts
export class MultiStepFormComponent {
  currentStep = 1;
  totalSteps = 3;

  formData = {
    step1: { name: '', email: '' },
    step2: { address: '', city: '' },
    step3: { cardNumber: '', cvv: '' }
  };

  nextStep(): void {
    if (this.currentStep < this.totalSteps) {
      this.currentStep++;
    }
  }

  previousStep(): void {
    if (this.currentStep > 1) {
      this.currentStep--;
    }
  }

  onSubmit(): void {
    console.log('Complete Form Data:', this.formData);
  }
}
```

```
<!-- multi-step-form.component.html -->
<div class="multi-step-form">
  <div class="progress-bar">
    Step {{ currentStep }} of {{ totalSteps }}
  </div>

  <form #multiForm="ngForm">
    <!-- Step 1 -->
    <div *ngIf="currentStep === 1" ngModelGroup="step1">
      <h3>Step 1: Personal Information</h3>
      <input type="text" name="name" [(ngModel)]="formData.step1.name" required>
      <input type="email" name="email" [(ngModel)]="formData.step1.email" required>
    </div>

    <!-- Step 2 -->
    <div *ngIf="currentStep === 2" ngModelGroup="step2">
      <h3>Step 2: Address</h3>
      <input type="text" name="address" [(ngModel)]="formData.step2.address" required>
      <input type="text" name="city" [(ngModel)]="formData.step2.city" required>
    </div>

    <!-- Step 3 -->
    <div *ngIf="currentStep === 3" ngModelGroup="step3">
      <h3>Step 3: Payment</h3>
      <input type="text" name="cardNumber" [(ngModel)]="formData.step3.cardNumber" required>
      <input type="text" name="cvv" [(ngModel)]="formData.step3.cvv" required>
    </div>

    <div class="navigation-buttons">
      <button type="button" (click)="previousStep()" [disabled]="currentStep === 1">
        Previous
      </button>
      <button type="button" (click)="nextStep()" *ngIf="currentStep < totalSteps">
        Next
      </button>
      <button type="submit" *ngIf="currentStep === totalSteps" (click)="onSubmit()">
        Submit
      </button>
    </div>
  </form>
</div>
```

7.2 Search Form with Filters

```
// search-form.component.ts
export class SearchFormComponent {
  searchCriteria = {
    keyword: '',
    category: '',
    minPrice: 0,
    maxPrice: 1000,
    inStock: false,
    sortBy: 'relevance'
  };

  categories = ['Electronics', 'Clothing', 'Books', 'Home & Garden'];
  sortOptions = ['relevance', 'price-low', 'price-high', 'newest'];

  onSearch(form: NgForm): void {
    if (form.valid) {
      console.log('Search Criteria:', this.searchCriteria);
      // Perform search
    }
  }

  clearFilters(form: NgForm): void {
    form.reset();
    this.searchCriteria = {
      keyword: '',
      category: '',
      minPrice: 0,
      maxPrice: 1000,
      inStock: false,
      sortBy: 'relevance'
    };
  }
}
```

---

8. Troubleshooting Common Issues

Issue 1: ngModel not working

Problem: Two-way binding not working

Solution: Ensure FormsModule is imported and name attribute is present

```
// app.module.ts
import { FormsModule } from '@angular/forms';

@NgModule({
  imports: [FormsModule]
})
```

```
<!-- Must have name attribute -->
<input [(ngModel)]="value" name="myField">
```

Issue 2: Validation not triggering

Problem: Validation errors not showing

Solution: Check that validators are registered and form is touched

```
// Register validator
@Directive({
  selector: '[appMyValidator]',
  providers: [{
    provide: NG_VALIDATORS,
    useExisting: MyValidatorDirective,
    multi: true  // Don't forget multi: true!
  }]
})
```

Issue 3: Form state not updating

Problem: Form pristine/dirty state not changing

Solution: Ensure ngForm directive is applied

```html
<!-- Correct -->
<form #myForm="ngForm">

<!-- Incorrect (missing template reference) -->
<form>
```

---

9. Summary and Key Takeaways

Template-Driven Forms

- ✅ Easy to use for simple forms
- ✅ Minimal component code
- ✅ Automatic form state tracking
- ✅ Two-way data binding with ngModel

Validation

- ✅ Built-in validators (required, email, minlength, etc.)
- ✅ Custom validators using directives
- ✅ Async validators for server-side checks
- ✅ Visual feedback with CSS classes

Custom Validators

- ✅ Created as directives
- ✅ Implement Validator interface
- ✅ Register with NG_VALIDATORS token
- ✅ Can accept parameters via @Input

Best Practices

- ✅ Validate both client and server side
- ✅ Provide clear error messages
- ✅ Use appropriate input types
- ✅ Make forms accessible
- ✅ Organize code properly

---

10. Additional Resources

- **Official Angular Forms Documentation:** https://angular.io/guide/forms-overview
- **Angular Forms API Reference:** https://angular.io/api/forms
- **Form Validation Guide:** https://angular.io/guide/form-validation
- **Accessibility Guidelines:** https://www.w3.org/WAI/tutorials/forms/

End of Training Document

This comprehensive guide covers all aspects of Template-Driven Forms and Validation in Angular. Practice these concepts by building real-world forms in your applications.