**Introduction**

This document compares two implementations of a custom Angular directive (MatchPasswordDirective) used for validating password confirmation fields in forms.

* **Version 1** is the **correct** implementation.
* **Version 2** contains **structural issues** that prevent it from working as intended.

**Version 1: Correct Implementation**

**Code**

typescript

import { Directive, Input } from '@angular/core';

import { AbstractControl, NG\_VALIDATORS, ValidationErrors, Validator } from '@angular/forms';

@Directive({

selector: '[appMatchPassword]',

providers: [

{

provide: NG\_VALIDATORS,

useExisting: MatchPasswordDirective,

multi: true

}

]

})

export class MatchPasswordDirective implements Validator {

@Input('appMatchPassword') matchTo!: string;

validate(control: AbstractControl): ValidationErrors | null {

if (!control || !control.parent) return null;

const passwordControl = control.parent.get('password');

const confirmPasswordControl = control;

if (!passwordControl || !confirmPasswordControl) return null;

if (passwordControl.value !== confirmPasswordControl.value) {

return { passwordMismatch: true };

}

return null;

}

}

**Key Features**

✅ **Proper Control Access**

* Uses control.parent.get() to access sibling controls.
* Correctly treats the current control (confirmPassword) as the validation target.

✅ **Clean Validation Logic**

* Compares password vs. confirmPassword values.
* Returns { passwordMismatch: true } if they don’t match.

✅ **No Unnecessary Dependencies**

* Does not use forwardRef (not needed here).

✅ **Clear Null Checks**

* Safely checks for control.parent existence.

**Usage**

**Template-Driven Form Example:**

html

<input type="password" name="password" ngModel>

<input type="password" name="confirmPassword" ngModel appMatchPassword>

**Reactive Form Example:**

typescript

this.form = this.fb.group({

password: ['', Validators.required],

confirmPassword: ['', [Validators.required, this.matchPassword.validate]]

});

**Version 2: Incorrect Implementation**

**Code**

typescript

import { Directive, forwardRef, Input } from '@angular/core';

import { AbstractControl, NG\_VALIDATORS, ValidationErrors, Validator } from '@angular/forms';

@Directive({

selector: '[appMatchPassword]',

providers: [

{

provide: NG\_VALIDATORS,

useExisting: forwardRef(() => MatchPasswordDirective),

multi: true

}

]

})

export class MatchPasswordDirective implements Validator {

@Input('appMatchPassword') matchTo!: string;

validate(control: AbstractControl): ValidationErrors | null {

if(!control || !control.parent) return null;

const password = control.get('password');

const confirmPassword = control.get('confirmPassword');

if(!password || !confirmPassword) return null;

if (password && confirmPassword && password.value !== confirmPassword.value) {

return { passwordsNotMatch: true };

}

return null;

}

}

**Key Issues**

❌ **Wrong Control Access**

* Uses control.get() instead of control.parent.get().
* Assumes both fields (password and confirmPassword) are direct children of the same FormGroup.

❌ **Unnecessary**forwardRef

* Not required for this use case.

❌ **Redundant Null Checks**

* Extra checks (if (password && confirmPassword && ...)) make logic harder to read.

❌ **Incorrect Usage**

* Would only work if applied to the **entire**FormGroup, not individual fields.

**Comparison Table**

| **Feature** | **Version 1 (Correct)** | **Version 2 (Incorrect)** |
| --- | --- | --- |
| **Control Access** | control.parent.get() | control.get() (wrong) |
| **Validation Target** | Current control vs password | Tries to get both fields |
| forwardRef | Not needed | Unnecessarily used |
| **Null Checks** | Clean and efficient | Redundant and confusing |
| **Usage** | Works on confirmPassword field | Only works on FormGroup level |

**Why Version 1 is Correct**

✔ **Follows Angular’s Form Validation Patterns**  
✔ **Works in Both Template-Driven & Reactive Forms**  
✔ **Clear and Maintainable Code**  
✔ **No Unnecessary Complexity**

**Best Practices**

1. **Use**control.parent.get() to access sibling controls.
2. **Avoid**forwardRef unless circular dependencies exist.
3. **Keep null checks simple**—don’t overcomplicate them.
4. **Test with both template-driven and reactive forms**.

**Conclusion**

* **Version 1** is the **recommended approach** for password matching validation.
* **Version 2** has **structural flaws** and should be avoided.