Angular Validation Interview Questions E-Book

By

www.questpond.com

Contents

What are two ways of doing validation in Angular?

There are two ways to do validations in Angular: -

1. Template driven forms.
2. Reactive forms.

Please note:- Template means the HTML part of Angular.

Differentiate between Template driven forms VS Reactive Forms?

In template driven forms you write validation inside the template (Declarative) while in reactive forms you can write the validation programmatically (Imperative) in the component.

In what situations you will use what?

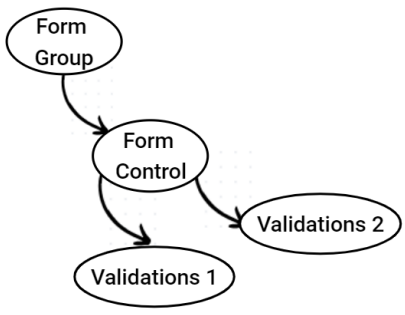
	Template Driven	Reactive forms
Simplicity	Declarative, Simple to write.	Complex to write but you have more control.
Complex scenarios	Template code becomes complex.	Can be handled nicely in Component.
Dynamic validation	Very difficult.	Can be easily handled.
Unit testing	Difficult due to Async nature.	Can be tested easily.
My personal view: - I would always prefer reactive forms because for some reason if the **complexity** of the validation increases, porting from template drive to reactive is again a big work. Also by using same way of implementation we have **uniformity**.		

Explain template reference variables ?

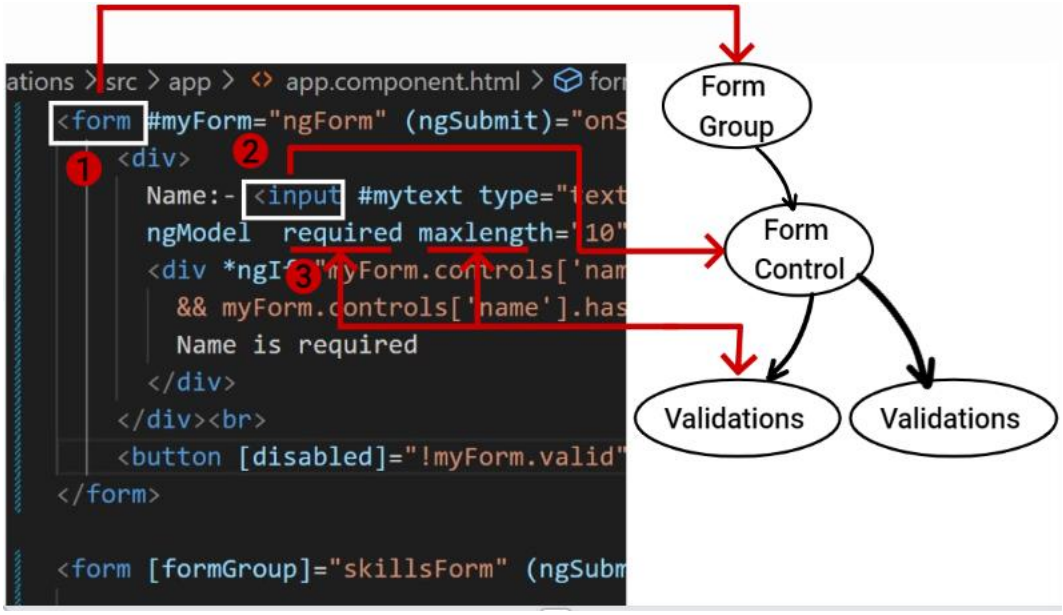
Template variables help you use access DOM elements inside the Angular template. You can see the code below we have create “mytext” variable and we are accessing it inside the angular expression.

```
<input #mytext type="text"
      (keyup)="0"><br>
  {{mytext.value}}
```

How do we implement Template driven forms?

In order to answer this question we need to keep the angular validation object model in mind. At the top we have “FormGroup” , inside “FormGroup” we have “FormControl” and “Validations” are applied to “FormControl”.	

1. Create Form tag and define template reference variable on the form tag (check the below image).
2. On the HTML controls apply validations.
3. Then use the “valid” and “hasError” to check if the validations are proper.

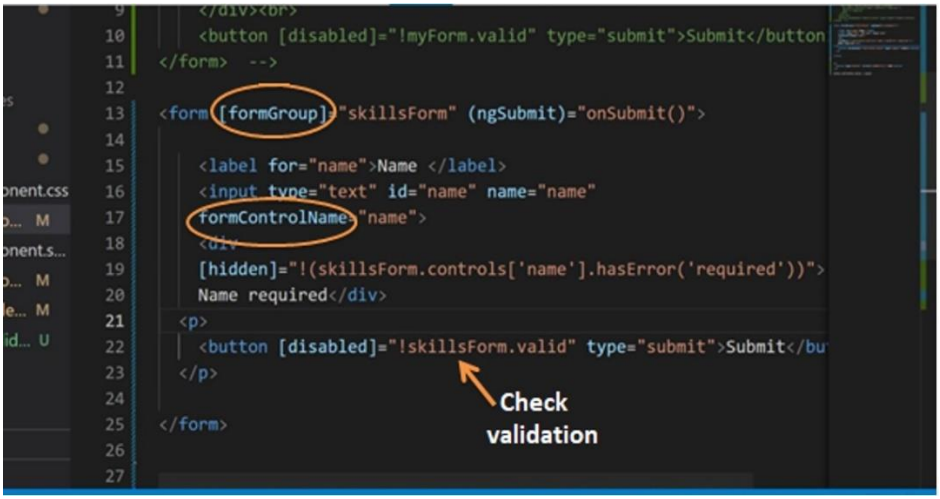


How do we implement Reactive forms ?

Reactive form validation is done in typescript code as shown below. We will need to use the “FormBuilder” to create the “FormGroup” and inside that we can define array of “Control” and

validations.

Once you have created the validations in the back end you need to apply on the front end by using “formGroup” and “formControlName”. To check if validations are ok we can use “valid method” as shown in below. For specific validation you need to use “hasError”.



How can we implement composite validations?

You can put the validation as collection as shown in the below code snippet.

```
this.skillsForm = this.fb.group({
     name: ['',[Validators.required , Validators.maxLength(20)]],
     skills: this.fb.array([]) ,
```

```
    });
```

How can we implement dynamic validations?

Step 1 :- Create an array of validations. Step 2 :- Use the FormBuilder and add the validator dynamically to the collection.	

Step 3 :- Once dynamic validation is created you can apply it dynamically using the angular “forloop”

```
<div formArrayName="skills">
    <div *ngFor="let skill of skills.controls; let i=index">
      <div [formGroupName]="i">
        <input type="text" formControlName="skillName">
         <div [hidden]="!(skill.get('skillName')?.hasError('required'))">
          Skill Name required
        </div>
      </div>
    </div>
```

How to check if overall validation and specific validations are good ?

To check if all validations of the “FormGroup” are good we can use the “valid” property.

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

To check if specific validation and specific control is valid we will need to use “hasError” method as shown in the below code snippet where “name” is the control and we are checking if “required” validator is valid or not.

```
<div *ngIf="myForm.controls['name'].touched
        && myForm.controls['name'].hasError('required')">
        Name is required
```

```
      </div>
```

Can you talk about some inbuilt validators?

Some inbuilt validators which you can remember talk during interview. You do not have to remember all pick some and speak.

- min :- min number that should be provided
- max :- max number that should be provided
- required :- control should have value
- requiredTrue :- control that should have the value true
- email :- string that matches valid email pattern
- minLength :- string that of minimum length
- maxLength :- string that of maximum length
- pattern :- string should match a pattern

How can you create your own custom validator ?

You need to implement “ValidatorFn” which is present in “@angular/forms”. You can write your custom code and return null if all validations are good or you return true.

```
export const passwordValidator = (): ValidatorFn => {
  return (control: AbstractControl):
    ValidationErrors | null => {
    const passwordRegex = /^(?=.*[A-Z])(?=.*\d).{8,}$/;

    if (passwordRegex.test(control.value)) {
      return null; // passes the test, return null to clear any errors
     }
     // fails the test, therefore the password is of invalid structure
     return { invalidPassword: true };
  };
};
```

Can Angular validations work with out FORM tag?

No, Angular validations need HTML “Form” tag.

What is [ngModelOptions]="{standalone: true}”?

If you have a control which is inside the “formGroup” and it has “ngModel” applied as shown in the below figure (second textbox “test1”) then a “formControl” is created by default and applied to that control. This leads to an error as we do not have validation applied to it.

```
<form [formGroup]="skillsForm" (ngSubmit)="onSubmit()">
<input type="text" id="name" name="name" formControlName="name">
<input [(ngModel)]="title" type="text" name="test1"/>
</form>
```

So one way to handle is by removing the textbox outside the “form” tag. Now this solution can disturb the designing or structure.

```
<form [formGroup]="skillsForm" (ngSubmit)="onSubmit()">
<input type="text" id="name" name="name" formControlName="name">
</form>
<input [(ngModel)]="title" type="text" name="test1"/>
```

The more clean way is to specify “standalone:true” which says that do not create a “formControl” on this control.

```
<form [formGroup]="skillsForm" (ngSubmit)="onSubmit()">
<input type="text" id="name" name="name" formControlName="name">
<input [ngModelOptions]="{standalone: true}"  [(ngModel)]="title" type="text"
name="test1"/>
</form>
```