Template driven form validation in Angular

In this tutorial, we will learn [template-driven form](https://www.tektutorialshub.com/angular/angular-template-driven-forms/) validation in Angular. It is very common that the users will make mistakes when filling out the web form. This is where the validations come into play. The validation module must ensure that the user has provided accurate and complete information in the form fields. We must display the validation error messages to the users, disable the submit button until validation. In this tutorial, we will look at how validations are handled in [Template-driven forms](https://www.tektutorialshub.com/angular/angular-template-driven-forms/) in Angular and learn how to use the Angular built-in validators.

This tutorial is a continuation of the [Angular template-driven forms tutorial](https://www.tektutorialshub.com/angular/angular-template-driven-forms/), where we built a simple form. In the next tutorial, we learned how to set the values to the form fields. We suggest you read those tutorials if you are new to Template-driven forms in Angular

**Table of Contents**

* [Template-driven Form Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#template-driven-form-validation)
* [Template](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#template)
* [Component Class](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#component-class)
* [Disabling the Browser validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#disabling-the-browser-validation)
* [Built-in Validators](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#built-in-validators)
* [Adding in Built-in Validators](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#adding-in-built-in-validators)
  + [Required Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#required-validation)
  + [Minlength Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#minlength-validation)
  + [Maxlength Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#maxlength-validation)
  + [Pattern Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#pattern-validation)
  + [Email Validation](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#email-validation)
* [Disable Submit button](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#disable-submit-button)
* [Displaying the Validation/Error messages](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#displaying-the-validationerror-messages)
  + [Why check dirty and touched?](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#why-checknbspdirtynbspandnbsptouched)
  + [Error message](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#error-message)
* [Final Template](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#final-template)
* [Summary](https://www.tektutorialshub.com/angular/template-driven-form-validation-in-angular/#summary)

## Template-driven Form Validation

Validations in [Template-driven forms](https://www.tektutorialshub.com/angular/angular-template-driven-forms/) are provided by the Validation directives. The [Angular Forms Module](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/) comes with several built-in validators. You can also create your own custom Validator.

## Template

Consider the following template-driven form. It has firstname, lastname, email, gender & istoc form fields.

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

  <p>

    <label for="firstname">First Name </label>

    <input type="text" id="firstname" name="firstname" [(ngModel)]="contact.firstname">

  </p>

  <p>

    <label for="lastname">Last Name </label>

    <input type="text" id="lastname" name="lastname" [(ngModel)]="contact.lastname">

  </p>

  <p>

    <label for="email">email </label>

    <input type="text" id="email" name="email" [(ngModel)]="contact.email">

  </p>

  <p>

    <label for="gender">Geneder </label>

    <input type="radio" value="male" id="gender" name="gender" [(ngModel)]="contact.gender"> Male

    <input type="radio" value="female" id="gender" name="gender" [(ngModel)]="contact.gender"> Female

  </p>

  <p>

    <label for="isToc">Accept TOC</label>

    <input type="checkbox" id="isToc" name="isToc" [(ngModel)]="contact.isToc">

  </p>

  <p>

    <button type="submit">Submit</button>

  </p>

</form>

## ****Component Class****

import { Component, ViewChild, ElementRef, OnInit } from '@angular/core';

import { NgForm } from '@angular/forms';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit  {

  title = 'Template driven forms';

  @ViewChild('contactForm',null) contactForm: NgForm;

  contact:contact;

  ngOnInit() {

    this.contact = {

      firstname:"",

      lastname:"",

      gender:"male",

      isToc:true,

      email:"",

    };

  }

  onSubmit() {

    console.log(this.contactForm.value);

  }

}

export class contact {

  firstname:string;

  lastname:string;

  gender:string;

  isToc:boolean;

  email:string;

}

## Disabling the Browser validation

First, we need to disable browser validator interfering with the Angular validator. To do that we need to add novalidate attribute on <form> element as shown below

|  |  |
| --- | --- |
| 1  2  3 | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)" novalidate> |

## Built-in Validators

The Built-in validators use the HTML5 validation attributes like required, minlength, maxlength & pattern. Angular interprets these validation attributes and add the validator functions to [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) instance.

## Adding in Built-in Validators

### Required Validation

The required validator returns true only if the form control has non-empty value entered. Let us add this validator to all fields

|  |  |
| --- | --- |
| 1  2  3 | <input type="text" id="firstname" name="firstname" required [(ngModel)]="contact.firstname"> |

### Minlength Validation

This Validator requires the control value must not have less number of characters than the value specified in the validator.

For Example, minlength validator ensures that the firstname value has at least 10 characters.

|  |  |
| --- | --- |
| 1  2  3 | <input type="text" id="firstname" name="firstname" required minlength="10" [(ngModel)]="contact.firstname"> |

### Maxlength Validation

This Validator requires that the number of characters must not exceed the value of the attribute.

|  |  |
| --- | --- |
| 1  2  3 | <input type="text" id="lastname" name="lastname" required maxlength="15" [(ngModel)]="contact.lastname"> |

### Pattern Validation

This Validator requires that the control value must match the regex pattern provided in the attribute. For example, the pattern ^[a-zA-Z]+$ ensures that the only letters are allowed (even spaces are not allowed). Let us apply this pattern to the lastName

|  |  |
| --- | --- |
| 1  2  3  4 | <input type="text" id="lastname" name="lastname" required maxlength="15"      pattern="^[a-zA-Z]+$" [(ngModel)]="contact.lastname"> |

### Email Validation

This Validator requires that the control value must be a valid email address. We apply this to the email field

|  |  |
| --- | --- |
| 1  2  3 | <input type="text" id="email" name="email" required email [(ngModel)]="contact.email"> |

## Disable Submit button

Now, we have successfully added the validators. You will notice that the click submit button still submits the form.

We need to disable the submit button if our form is not valid.

Angular forms module keep track of the state of our form and each of its form elements. These states are exposed to the user through FormGroup, FormArray & FormControl objects.

We get the reference to the top-level FormGroup instance by creating a template variable and bind it to ngForm. We have already done it when we had added the #contactForm="ngForm" in our form tag.

The [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) has a valid property, which is set to true if all of its child controls are valid. We use it to set the disabled attribute of the submit button.

|  |  |
| --- | --- |
| 1  2  3 | <button type="submit" [disabled]="!contactForm.valid">Submit</button> |

So long as contactForm.valid remains false, the submit button remains disabled.

## Displaying the Validation/Error messages

We need to provide a short and meaningful error message to the user.

Angular creates a FormControl for each and every field, which has ngModel directive applied. The FormControl exposes the state of form element like valid, dirty, touched, etc.

There are two ways in which you can get the reference to the FormControl.

One way is to use the contactForm variable. We can use the contactForm.controls.firstname.valid to find out if the firstname is valid.

The other way is to create a new local variable for each FormControl For Example, the following firstname="ngModel" creates the firstname variable with the FormControl instance.

|  |  |
| --- | --- |
| 1  2  3  4 | <input type="text" id="firstname" name="firstname" required minlength="10"              #firstname="ngModel" [(ngModel)]="contact.firstname"> |

Now, we have a reference to the firstname FormControl instance, we can check its status. We use the valid property to check if the firstname has any errors.

valid: returns either invalid status or null which means a valid status

|  |  |
| --- | --- |
| 1  2  3  4  5 | <div \*ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)">     Invalid First Name  </div> |

### Why check dirty and touched?

We do not want the application to display the error when the form is displayed for the first time. We want to display errors only after the user has attempted to change the value. The dirty & touched properties help us do that.

dirty: A control is dirty if the user has changed the value in the UI.  
touched: A control is touched if the user has triggered a blur event on it.

### Error message

The error message ” “Invalid First Name” ” is not helpful. The firstname has two validators. required and minlength

Any errors generated by the failing validation is updated in the errors object. The errors object returns the error object or null if there are no errors.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <div \*ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)">    Invalid First Name    <div \*ngIf="firstname.errors.required">       First Name is required    </div>    <div \*ngIf="firstname.errors.minlength">      First Name Minimum Length is {{firstname.errors.minlength?.requiredLength}}    </div>  </div> |

Note that the minlength validators return the {{firstname.errors.minlength?.requiredLength}}, which we use the display the error message.

## Final Template

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81 | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)" novalidate>      <p>      <label for="firstname">First Name </label>      <input type="text" id="firstname" name="firstname" required minlength="10" #firstname="ngModel"        [(ngModel)]="contact.firstname">    </p>    <div \*ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)" class="error">      <div \*ngIf="firstname.errors.required">        First Name is required      </div>      <div \*ngIf="firstname.errors.minlength">        First Name Minimum Length is {{firstname.errors.minlength?.requiredLength}}      </div>    </div>      <p>      <label for="lastname">Last Name </label>      <input type="text" id="lastname" name="lastname" required maxlength="15" #lastname="ngModel"              pattern="^[a-zA-Z]+$"  [(ngModel)]="contact.lastname">    </p>    <div \*ngIf="!lastname?.valid && (lastname?.dirty || lastname?.touched)" class="error">      <div \*ngIf="lastname.errors.required">        Last Name is required      </div>      <div \*ngIf="lastname.errors.maxlength">        Last Name Minimum Length is {{lastname.errors.maxlength?.requiredLength}}      </div>      <div \*ngIf="lastname.errors.pattern">        Only characters are allowed      </div>    </div>          <p>      <label for="email">email </label>      <input type="text" id="email" name="email" required email #email="ngModel" [(ngModel)]="contact.email">    </p>    <div \*ngIf="!email?.valid && (email?.dirty || email?.touched)" class="error">      <div \*ngIf="email.errors.required">        Email is required      </div>      <div \*ngIf="email.errors.email">        Invalid Email Address      </div>    </div>      <p>      <label for="gender">Geneder </label>      <input type="radio" value="male" id="gender" name="gender" #gender="ngModel" required [(ngModel)]="contact.gender">      Male      <input type="radio" value="female" id="gender" name="gender" #gender="ngModel" required        [(ngModel)]="contact.gender"> Female    </p>    <div \*ngIf="!gender?.valid && (gender?.dirty || gender?.touched)" class="error">      <div \*ngIf="gender.errors.required">        Gender is required      </div>    </div>        <p>      <label for="isToc">Accept TOC</label>      <input type="checkbox" id="isToc" name="isToc" required #isToc="ngModel" [(ngModel)]="contact.isToc">    </p>    <div \*ngIf="!isToc?.valid && (isToc?.dirty || isToc?.touched)" class="error">      <div \*ngIf="isToc.errors.required">        Please accept the TOC      </div>    </div>      <p>      <button type="submit" [disabled]="!contactForm.valid">Submit</button>    </p>      <p>{{contactForm.valid}} </p>    </form> |

## Summary

Angular template-driven form validation uses the directives known as validators. The validators handle form validations and display validation messages. The Angular comes up with several built-in validators for this purpose. They are minlength, maxlength, email, pattern, required, etc.