Angular 12 Template Driven Forms With Validation

Template-driven forms fit small or simple forms, while Reactive forms are more scalable and proper for complex forms.

In Angular template-driven forms, control elements are bound to data properties, with input validation to maintain data integrity and styling to improve the user experience.

Ng new templatedrivenFormDemo

npm install bootstrap@latest –save

## **Step 1: Create a new Angular project**

To create an Angular 12 project, you have to install [Angular CLI 10](https://appdividend.com/2020/06/25/how-to-update-angular-cli-version/).

Type the following command to create a new Angular project.

ng **new** angulartemplate

Go inside the folder and install the Bootstrap CSS Framework.

npm install **bootstrap** --**save**

Open an Angular.json file and add the path to the Bootstrap CSS Framework.

"styles": [

"src/styles.css",

"./node\_modules/bootstrap/dist/css/bootstrap.min.css"

],

## **Step 2: Import FormsModule**

Inside the **app.module.ts**file, we have to import the FormsModule.

*// app.module.ts*

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

imports: [

...

FormsModule

],

The application enables the Forms feature and registers the created form component.

Template-driven forms depend on directives defined in the FormsModule.

1. The **NgModel** directive regulates value changes in the attached form element with changes in the data model, letting you respond to user input with input validation and error handling.
2. The **NgForm** directive builds a top-level FormGroup instance and binds it to an <form> element to track aggregated form value and validation status. When you import FormsModule, this directive becomes active by default on all <form> tags. You don’t need to append a particular selector.
3. The **NgModelGroup** directive makes and binds a FormGroup instance to a DOM element.

## **Step 3: Build a Register Form**

The form layout and details are defined in the AppComponent class. Write the following code inside the **app.component.ts**file.

*// app.component.ts*

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

@Component(*{*

*selector: 'app-root',*

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

*styleUrls: ['./app.component.css']*

*}*)

**export** **class** AppComponent **implements** OnInit *{*

*formData: any = {}*;

**constructor**() *{*

*}*

**ngOnInit**(): void *{*

*}*

}

The component’s selector value of “app-root” means you can drop this form in the app.component.html file. The next step is to write the HTML elements of the form.

Write the following code inside the **app.component.html**file.

*<!-- app.component.html -->*

<div class="container">

<div class="row">

<div class="col-md-5">

<h1 class="page-title">User Registration</h1>

<form>

<div class="form-group">

<label for="username">Username</label>

<input type="text"

class="form-control"

required />

</div>

<div class="form-group">

<label for="email">Email</label>

<input type="email"

class="form-control"

required />

</div>

<div class="form-group">

<label for="password">Password</label>

<input type="password"

class="form-control"

required />

</div>

<div class="form-group">

<label for="passwordConfirmation">Confirm Password</label>

<input type="password"

class="form-control"

required />

</div>

<button

type="submit"

class="btn btn-warning">

Register

</button>

</form>

</div>

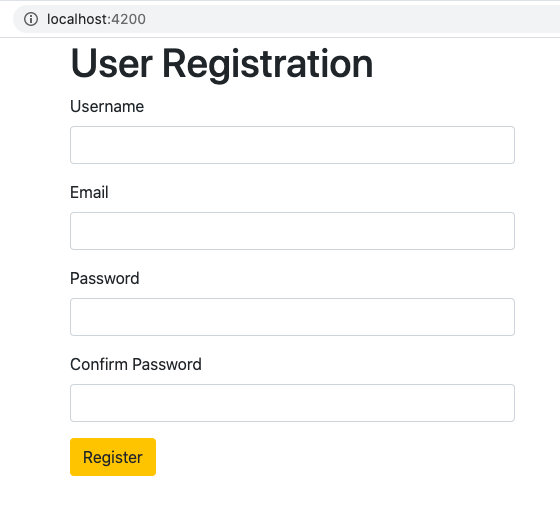
</div>

</div>

Save the file and run the Angular development server by the following command.

ng serve -o

It will give us the following output.

[](https://appdividend.com/wp-content/uploads/2020/07/Angular-10-Template-Driven-Forms.png)

We have created four form fields.

1. username
2. email
3. password
4. passwordConfirmation

## **Step 3: Bind input controls to data properties**

The next step is to bind the input controls to the corresponding form properties with two-way data binding so that they respond to user input by updating a data model and also respond to programmatic modifications in the data by updating the display.

The **ngModel** directive declared in the **FormsModule** lets you bind controls in your template-driven form to properties in your data model.

When you include the directive using the syntax for two-way data binding, **[(ngModel)]**, Angular can track the value and user interaction of the control and keep the view synced with the model.

Add the ngModel directive in each input element of the form, using two-way data binding syntax [(ngModel)]=”…”.

*<!-- app.component.html -->*

<div class="form-group">

<label for="username">Username</label>

<input type="text"

class="form-control"

[(ngModel)]="formData.name"

required />

</div>

Do this for the remaining three form elements. So, your AppComponent file looks like below.

<!-- app.component.html -->

<**div** class="container">

<**div** class="row">

<**div** class="col-md-5">

<**h1** class="page-title">User Registration</h1>

<**form** #registerForm="ngForm">

<**div** class="form-group">

<**label** **for**="username">Username</label>

<**input** type="text"

class="form-control"

[(ngModel)]="formData.name"

required />

</div>

<**div** class="form-group">

<**label** **for**="email">Email</label>

<**input** type="email"

class="form-control"

[(ngModel)]="formData.email"

required />

</div>

<**div** class="form-group">

<**label** **for**="password">Password</label>

<**input** type="password"

class="form-control"

[(ngModel)]="formData.password"

required />

</div>

<**div** class="form-group">

<**label** **for**="passwordConfirmation">Confirm Password</label>

<**input** type="password"

class="form-control"

[(ngModel)]="formData.passwordConfirmation"

required />

</div>

<**button**

type="submit"

class="btn btn-warning">

Register

</button>

</form>

</div>

</div>

</div>

## **Step 4: Access the overall form status**

When you imported the FormsModule in your component, Angular automatically created and attached a NgForm directive to the <form> tag in the Template (because NgForm has the selector form that matches <form> elements).

To get access to the NgForm and the overall form status, declare a template reference variable.

*<!-- app.component.html -->*

<form #registerForm="ngForm">

## **Step 5: Naming control elements**

When you use [(ngModel)] on an element, you must define a name attribute for that element.

Angular uses the assigned name to register the element with the NgForm directive attached to the parent <form> element.

*<!-- app.component.html -->*

<div class="form-group">

<label for="username">Username</label>

<input type="text"

class="form-control"

[(ngModel)]="formData.name"

name="username"

required />

</div>

Do this and add a name attribute to the remaining form elements.

Each <input> element also has the required **name** property Angular uses to register the control with the form.

So, our **app.component.html**file looks like this.

<**div** class="container">

<**div** class="row">

<**div** class="col-md-5">

<**h1** class="page-title">User Registration</h1>

<**form** #registerForm="ngForm">

<**div** class="form-group">

<**label** **for**="username">Username</label>

<**input** type="text"

class="form-control"

[(ngModel)]="formData.name"

name="username"

#username="ngModel"

required />

</div>

<**div** class="form-group">

<**label** **for**="email">Email</label>

<**input** type="email"

class="form-control"

[(ngModel)]="formData.email"

name="email"

#email="ngModel"

required />

</div>

<**div** class="form-group">

<**label** **for**="password">Password</label>

<**input** type="password"

class="form-control"

[(ngModel)]="formData.password"

name="password"

#password="ngModel"

required />

</div>

<**div** class="form-group">

<**label** **for**="passwordConfirmation">Confirm Password</label>

<**input** type="password"

class="form-control"

[(ngModel)]="formData.passwordConfirmation"

name="passwordConfirmation"

#passwordConfirmation="ngModel"

required />

</div>

<**button**

type="submit"

class="btn btn-warning">

Register

</button>

</form>

</div>

</div>

</div>

Also, extend the <input> tag with the template reference variable that you can use to access the form’s input box’s Angular control from within the template. In our example, the variable is **#username**=”**ngModel**“.

So, write these template reference variables for the remaining three input fields.

## **Step 6: Track control states in Angular Forms**

The NgModel directive on a control tracks the state of that control. It tells you if the user touched the control, if the value changed, or if it became invalid.

Angular sets special CSS classes on the control element to reflect the state, as shown in the following table.

|  |  |  |
| --- | --- | --- |
| **State** | **Class if true** | **Class if false** |
| The control has been visited. | ng-touched | ng-untouched |
| The control’s value has changed. | ng-dirty | ng-pristine |
| The control’s value is valid. | ng-valid | ng-invalid |

You use these CSS classes to define the styles for your control based on their status.

Based on these statuses, we will display the validation error.

## **Step 7: Show and hide validation error messages**

After every input element, you need to write the following conditional element.

*<!-- app.component.html -->*

<div \*ngIf="username.invalid && (username.dirty || username.touched)"

class="alert alert-danger">

<div \*ngIf="username.errors.required">

Username is required

</div>

</div>

In this code, based on form control status, we are displaying the validation message. If everything is good, then the validation message will be hidden; otherwise, it will show.

Write the above snippet for every form input element.

<!*-- app.component.html -->*

<**div** class="container">

<**div** class="row">

<**div** class="col-md-5">

<h1 class="page-title">User Registration</h1>

<form *#registerForm="ngForm">*

<**div** class="form-group">

<label **for**="username">Username</label>

<input type="text"

class="form-control"

[(ngModel)]="formData.name"

name="username"

*#username="ngModel"*

required />

</**div**>

<**div** \*ngIf="username.invalid && (username.dirty || username.touched)"

class="alert alert-danger">

<**div** \*ngIf="username.errors.required">

Username **is** required

</**div**>

</**div**>

<**div** class="form-group">

<label **for**="email">Email</label>

<input type="email"

class="form-control"

[(ngModel)]="formData.email"

name="email"

*#email="ngModel"*

required />

</**div**>

<**div** \*ngIf="email.invalid && (email.dirty || email.touched)"

class="alert alert-danger">

<**div** \*ngIf="email.errors.required">

Email **is** required

</**div**>

</**div**>

<**div** class="form-group">

<label **for**="password">Password</label>

<input type="password"

class="form-control"

[(ngModel)]="formData.password"

name="password"

*#password="ngModel"*

required />

</**div**>

<**div** \*ngIf="password.invalid && (password.dirty || password.touched)"

class="alert alert-danger">

<**div** \*ngIf="password.errors.required">

Password **is** required

</**div**>

</**div**>

<**div** class="form-group">

<label **for**="passwordConfirmation">Confirm Password</label>

<input type="password"

class="form-control"

[(ngModel)]="formData.passwordConfirmation"

name="passwordConfirmation"

*#passwordConfirmation="ngModel"*

required />

</**div**>

<**div** \*ngIf="passwordConfirmation.invalid && (passwordConfirmation.dirty || passwordConfirmation.touched)"

class="alert alert-danger">

<**div** \*ngIf="passwordConfirmation.errors.required">

Confirm Password **is** required

</**div**>

</**div**>

<button

type="submit"

class="btn btn-warning">

Register

</button>

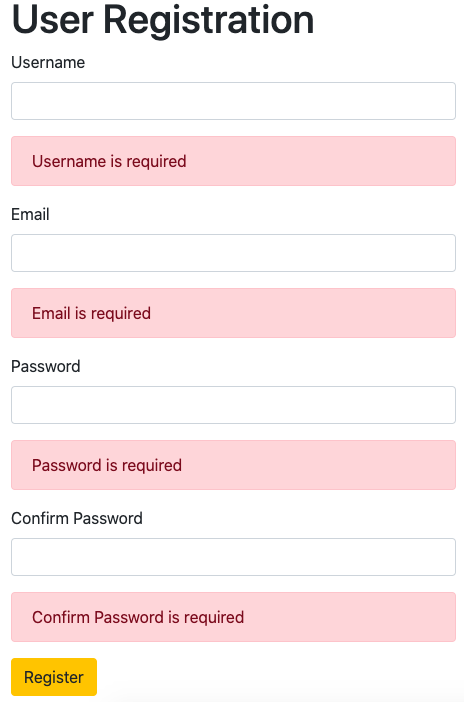
</form>

</**div**>

</**div**>

</**div**>

Now, if you touch any input form fields and don’t write anything, it will show us an error like the following.

[](https://appdividend.com/wp-content/uploads/2020/07/Angular-10-Form-Validation.png)

## **Step 8: Disable the Submit or Register button.**

In the above code, you can see that even if there are errors, we can still submit the form, which is not correct.

If a single error is there in the form fields, the submit or register button should be disabled.

Edit the button element and add the following code instead.

*<!-- app.component.html -->*

<button

type="submit"

class="btn btn-warning"

[disabled] = "!registerForm.form.valid">

Register

</button>

What this code does is that if our **registerForm** is invalid, then it will keep disabling the button.

If any input form fields have a single error, it will keep disabling. If everything is valid, then and then it will enable the **Register**button.

## **Step 9: Create a register() method and log the data**

If everything is perfect and valid, then we have to submit the form data. To do that, let’s put a button click event and call the register() function.

Again, edit the <button> element and add the click event.

<button

**type**="submit"

**class**="btn btn-warning"

[disabled] = "!registerForm.form.valid"

(click)="register()">

**Register**

</button>

Okay, now we have to define the **register()**function inside the **app.component.ts**file. Let’s do that.

*// app.component.ts*

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

**@Component**({

selector: 'app-root',

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

styleUrls: ['./app.component.css']

})

**export** **class** AppComponent **implements** OnInit {

formData: any = {};

**constructor**() {

}

ngOnInit(): void {

}

register(): void {

console.log(**this**.formData);

}

}

That is it. Now, if the form is valid, then we can get all the form values inside the register() method.

[](https://appdividend.com/wp-content/uploads/2020/07/Angular-10-Template-Form-Validation.png)

That is it for Angular 12 Template Driven Forms with Validation Example.