Angular Template Driven Forms example

10 Comments / 10 minutes of reading / April 17, 2024

← Angular Form Tutorial | Angular Tutorial | Set Value in template Driven Forms →

Template driven Forms in Angular is one of the two ways of building forms in Angular. In this tutorial, we will learn how to build a simple Template-driven Form. First, we build a simple HTML form using a few form elements. Then use the `ngForm` directive to convert them to Template-driven Form, which creates the top-level `FormGroup` control. Next, we use the `ngModel` directive to create the `FormControl` instance for each of the HTML form elements. Later, we will learn how to submit the form data to the component class. We will also learn how to initialize or reset the form data and use the data binding to access the data in the component class.

If you have not gone through our Angular Forms Tutorial, we strongly recommend you to do so. In that article. we have covered fundamental concepts of the Angular Forms Module.

Table of Contents

What is Template-driven form?

In Template Driven Forms we specify behaviors/validations using directives and attributes in our template and let it work behind the scenes. All things happen in Templates hence very little code is required in the component class. This is different from the reactive forms, where we define the logic and controls in the component class.

The Template-driven forms

1. The form is set up using `ngForm` directive
2. controls are set up using the `ngModel` directive
3. `ngModel` also provides the two-way data binding
4. The Validations are configured in the template via directives

Template-driven forms are

1. Contains little code in the component class
2. Easier to set up

While they are

1. Difficult to add controls dynamically
2. Unit testing is a challenge

Create the Example Application

Use `ng new` to create a new application.

```
ng new tdf
```

Run `ng serve` and verify if everything is installed correctly.

Import FormsModule

To work with Template-driven forms, we must import the `FormsModule`. The `FormsModule` contains all the form directives and constructs for working with forms.

How we import `FormsModule` depends on whether our Component is **Standalone Component or Module Based Component**. If you created the application in Angular 17 or above then it defaults to Standalone component else it will be using the Module based Component.

Standalone Components

If you are using Standalone component, then open the `app.component.ts` and import the `FormsModule`. In a Standalone Component you will see the `standalone: true,` flag in the component decorator. If not then it is a module based component.

```

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

```

Add the `FormsModule` to imports metadata of component decorator as shown below

```

@Component({
  imports: [CommonModule, FormsModule],
  selector: 'app-root',
  standalone: true,
   template: ``
})

```

Module Based Components

If you are using the **module based component**, then we usually import the `FormsModule` in root module or in a shared module. (You can also import it in the ngModule to which component belongs).

Open the `app.module.ts` and add the following import.

```
1
2 import { FormsModule } from '@angular/forms';
3
```

Next, add the `FormsModule` to the *imports metadata property array*

```
1
2 import { BrowserModule } from '@angular/platform-browser';
3 import { NgModule } from '@angular/core';
4 import { FormsModule } from '@angular/forms';       //import FormsModule
5
6 import { AppRoutingModule } from './app-routing.module';
7 import { AppComponent } from './app.component';
8
9 @NgModule({
10   declarations: [
11     AppComponent
12   ],
13   imports: [
14     BrowserModule,
15     AppRoutingModule,
16     FormsModule                  //Add in Imports Array
17   ],
18   providers: [],
19   bootstrap: [AppComponent]
20 })
21 export class AppModule { }
22
```

HTML Form

The first task is to build the template. The following is a regular HTML `form`. We enclose it in a `<form>` tag. We have included two text input (`firstName` & `lastName`), a email (`email`), a radio button (`gender`), a checkbox (`isMarried`), and a select options list (`country`). These are form elements.

```
1
2 <form>
3
4   <p>
5     <label for="firstname">First Name</label>
```

```
    <input type="text" id="firstname" name="firstname">
  </p>

  <p>
    <label for="lastname">Last Name</label>
    <input type="text" id="lastname" name="lastname">
  </p>

  <p>
    <label for="email">Email </label>
    <input type="text" id="email" name="email">
  </p>

  <p>
    <label for="gender">Geneder</label>
    <input type="radio" value="male" id="gender" name="gender"> Male
    <input type="radio" value="female" id="gender" name="gender"> Female
  </p>

  <p>
    <label for="isMarried">Married</label>
    <input type="checkbox" id="isMarried" name="isMarried">
  </p>

  <p>
  <label for="country">country </label>
  <select name="country" id="country">
    <option selected="" value=""></option>
    <option [ngValue]="c.id" *ngFor="let c of countryList">
      {{c.name}}
    </option>
  </select>
  </p>

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

</form>

```

Component Class (Standalone component)

We have exported a class `Country`. `countryList` is an array of `Country`, which we used in the `country` dropdown.

This is a standalone component. For Module based component remove the statements `imports: [CommonModule, FormsModule]` & `standalone: true` . Rest of the code is same for both.

```
1
2  import { CommonModule } from '@angular/common';
3  import { Component, OnInit } from '@angular/core';
4  import { FormsModule } from '@angular/forms';
5
6  @Component({
7    imports: [CommonModule, FormsModule],     //For Standalone Components
8    selector: 'app-root',
9    standalone: true,          //For Standalone Components
10   templateUrl: './app.component.html',
11 })
12 export class AppComponent implements OnInit {
13
14   countryList: country[] = [
15     new country('1', 'India'),
16     new country('2', 'USA'),
17     new country('3', 'England'),
18   ];
19
20   constructor() {}
21
22   ngOnInit() {}
23
24 }
25
26 export class country {
27   id: string;
```

```
28   name: string;
29
30   constructor(id: string, name: string) {
31     this.id = id;
32     this.name = name;
33   }
34 }
35
```

ngForm

Once, we have a form with few form elements, the angular automatically converts it into a Template-driven form. This is done by the `ngForm` directive.

The `ngForm` directive is what makes the Angular template-driven forms work. But we do not need to do anything explicitly.

When we include `FormsModule`, the Angular is going to look out for any `<form>` tag in our HTML template. The `ngForm` directive automatically detects the `<form>` tag and automatically binds to it. You do not have to do anything on your part to invoke and bind the `ngForm` directive.

The `ngForm` does the following

1. Binds itself to the `<Form>` directive
2. Creates a top-level `FormGroup` instance
3. Creates `FormControl` instance for each of child control, which has `ngModel` directive.
4. Creates `FormGroup` instance for each of the `NgModelGroup` directive.

We can export the `ngForm` instance into a local template variable using `ngForm` as the key (ex: `#contactForm="ngForm"`). This allows us to access the many properties and methods of `ngForm` using the template variable `contactForm`

Hence, update the `form` element as shown below.

```
<form #contactForm="ngForm">
```

FormControl

The `FormControl` is the basic building block of the Angular Forms. It represents a single input field in an Angular form. The Angular Forms Module binds the input element to a `FormControl`. We use the `FormControl` instance to track the value, user interaction and validation status of an individual form element. Each individual `Form` element is a `FormControl`.

We have six form elements in our HTML template. They are `firstName`, `lastname`, `email`, `gender`, `isMarried` & `country`. We need to bind them to `FormControl` instance. We do this by using the `ngModel` directive. Add the `ngModel` directive to **every control** as shown below.

```
<input type="text" name="firstname" ngModel>
```

`ngModel` **will use the** `name` **attribute to create the** `FormControl` **instance** for each of the `Form` field it is attached.

Submitting the Form

Now have the template ready, except for the final piece i.e. submitting data to the component.

We use the ngSubmit event, to submit the form data to the component class. We use the event binding (parentheses) to bind ngSubmit to OnSubmit method in the component class. When the user clicks on the submit button, the ngSubmit event will fire.

```
1
2 <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">
3
```

We are passing the local template variable contactForm in onSubmit method. contactForm holds the reference to the ngForm directive. We can use this in our component class to extract the data from the form fields.

Final Template

Our final template is as shown below.

```
1
2 <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">
3
4   <p>
5     <label for="firstname">First Name</label>
6     <input type="text" name="firstname" ngModel #fname="ngModel">
7   </p>
8
9   <p>
```

```
    <label for="lastname">Last Name</label>
    <input type="text" name="lastname" ngModel>
  </p>

  <p>
    <label for="email">Email </label>
    <input type="text" id="email" name="email" ngModel>
  </p>

  <p>
    <label for="gender">Geneder</label>
    <input type="radio" value="male" name="gender" ngModel> Male
    <input type="radio" value="female" name="gender" ngModel> Female
  </p>

  <p>
    <label for="isMarried">Married</label>
    <input type="checkbox" name="isMarried" ngModel>
  </p>

  <p>
    <label for="country">Country</label>
      <select name="country" ngModel>
        <option [ngValue]="c.id" *ngFor="let c of countryList">
          {{c.name}}
        </option>
      </select>
  </p>

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

</form>

```

Receiving the form data in Component

We need to receive the data in component class from our form. To do this we need to create the `onSubmit` method in our component class. The `submit` method receives the reference to the `ngForm` directive, which we named is as `contactForm`. The `contactForm` exposes the `value` method which returns the form fields as a Json object.

```

```

```
  onSubmit(contactForm) {
    console.log(contactForm.value);
  }

```

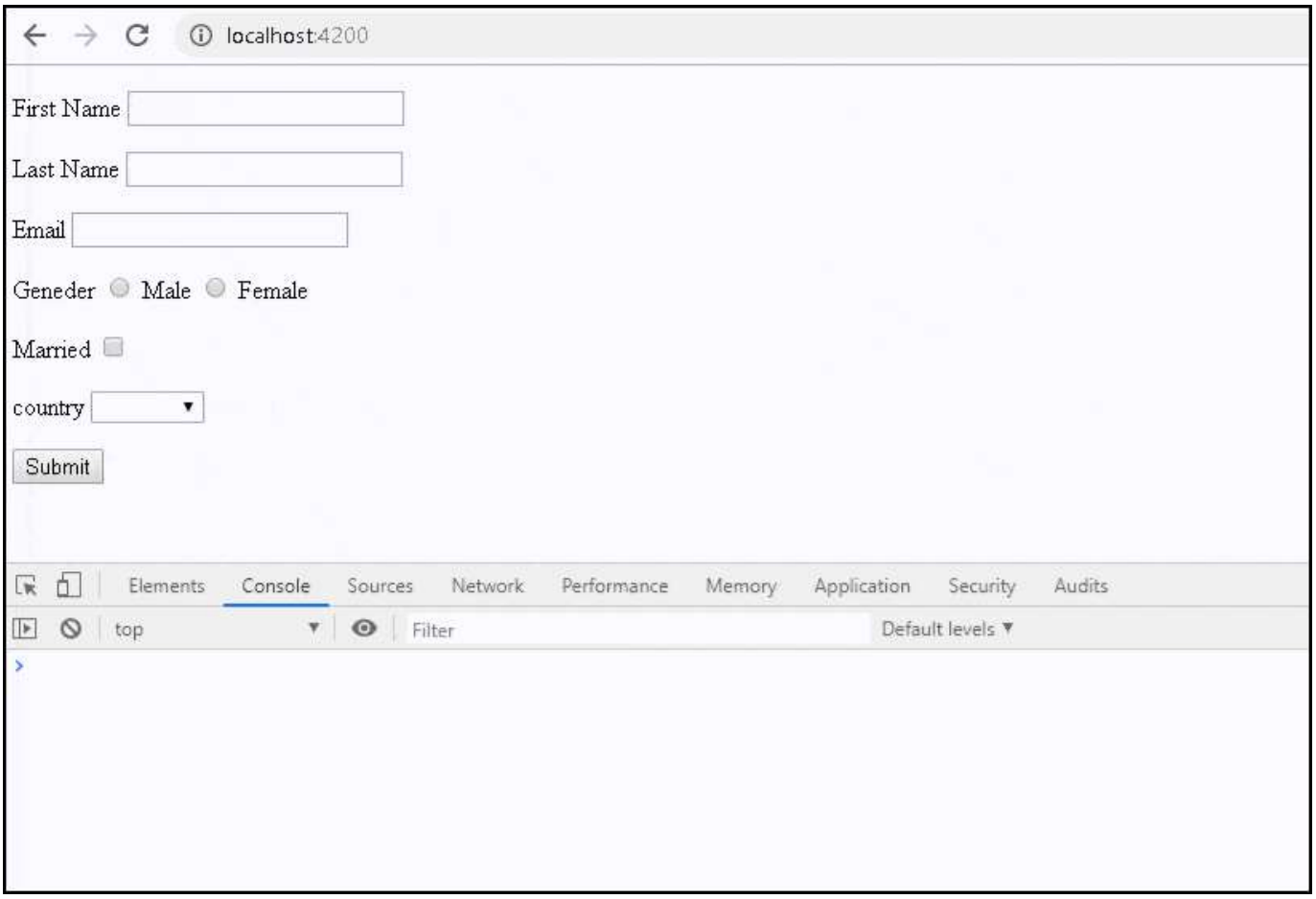
You can print the value to the console using the console.log(contactForm.value)

Run the code now and enter some data into the form. Open the Developer Console in your browser and check the output, when you submit the data.

```

country: "1"
firstname: "Sachin"
email:"sachin@gmail.com"
gender: "male"
isMarried: true
lastname: "Tendulkar"

```



Angular template-driven forms in Action

Local Variable

We can assign the ngForm , FormControl or FormGroup instance to a template local variable. This allows us to check the status of the form like whether the form is valid , submitted , and value of the form elements, etc

ngForm

We have access to the ngForm instance via the local template variable #contactForm .

```
1
2 <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">
3
```

Now, we can make use of some of the properties & methods to know the status of form. For Example

```
1
2 <p>
3   <button type="submit">Submit</button>
4 </p>
5
6 <pre>Value : {{contactForm.value | json }} </pre>
7 <pre>Valid : {{contactForm.valid}} </pre>
8 <pre>Touched : {{contactForm.touched  }} </pre>
9 <pre>Submitted : {{contactForm.submitted  }} </pre>
10
```

value : The value property returns the object containing the value of every FormControl

valid : Returns true if the form is Valid else returns false.

touched : True if the user has entered a value in at least in one field.

submitted : Returns true if the form is submitted. else false.

FormControl

Similarly, we can also get access to the `FormControl` instance by assigning the `ngModel` to a local variable as shown below

```
<input type="text" name="firstname" #fname="ngModel" ngModel>
```

Now, the variable `#fname` holds the reference to the `firstname` FormControl. We can then access the properties of FormControl like `value`, `valid`, `isvalid`, `tocuhed` etc

```
<p>
  <label for="firstname">First Name </label>
  <input type="text" name="firstname" #fname="ngModel" ngModel>
</p>

<pre>Value    : {{fname.value}} </pre>
<pre>valid    : {{fname.valid}} </pre>
<pre>invalid  : {{fname.invalid}} </pre>
<pre>touched  : {{fname.touched}} </pre
```

`value` : Returns the current value of the control

`valid` : Returns true if the value is Valid else false

invalid : True if the value is invalid else false

touched : Returns true if the value is entered in the element

Nested FormGroup

The FormGroup is a collection of FormControl . It can also contain other FormGroup's .

The ngForm directive creates the top Level FormGroup behind the scene, when we use the <Form> directive.

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

We can add new FormGroup using the ngModelGroup directive. Let us add street , city & Pincode form controls and group them under the address FormGroup

All you need to do is to enclose the fields inside a div element with ngModelGroup directive applied on it as shown below

```
<div ngModelGroup="address">

   <p>
     <label for="city">City</label>
     <input type="text" name="city" ngModel>
   </p>

   <p>
     <label for="street">Street</label>
     <input type="text" name="street" ngModel>
   </p>
   <p>
     <label for="pincode">Pin Code</label>
     <input type="text" name="pincode" ngModel>
   </p>
```

```
</div>
```

Run the App and submit. The resultant object is as shown below.

```
Value : {
  "firstname": "Sachin",
  "lastname": "Tendulkar",
  "email":"sachin@gmail.com"
  "gender": "male",
  "isMarried": true,
  "country": "1",
  "address": {
    "city": "Mumbai",
    "street": "Fashin Street",
    "pincode": "400600"
  }
}
```

Setting the Initial Value

The form is usually pre-filled with some default data. In the case of editing, we have to show the user the current data. You can refer to the next tutorial on How to set value in the template-driven form.

Validating the Form

Validating the form is another important task. We have covered it in Validation in template-driven form tutorial.

Summary

Angular Template-driven Forms is simpler compared to the reactive forms. The FormsModule is imported first either in the component (in case of standalone