Creating Template-driven Forms in Angular

Forms are an important feature of any business applications. Forms are useful in many aspects of the data-entry applications like ticket booking, scheduling a meeting, doctor appointments etc.

## **Introduction**

Developing template-driven forms requires design skills which are out of scope now. Also, forms require framework support for two-way data binding, change tracking, validation, and error handling.

In this guide, you will learn how to build a simple form from scratch. Below are the things you will learn:

1. How to build an Angular form with a component and template.
2. How to use ngModel to create two-way data bindings for reading and writing input-control values.
3. How to track state changes and the validity of form controls.
4. How to provide visual feedback using special CSS classes that track the state of the controls.
5. How to display validation errors to users and enable/disable form controls.
6. How to share information across HTML elements using template reference variables.

## **Building the Template-Driven Application**

Let's consider an example where we are building the Employee Form.

Filename: person.ts

1

2

3

4

5

6

7

8

9

10

export class Person {

constructor(

public id: number,

public firstName: string,

public award: string,

public address?: string

) { }

}

typescript

The address is optional, so the constructor lets you omit it; note the question mark (?) in “address?”.

You can create a new person like this:

1

2

3

4

let myPerson = new Person(42, 'Raj',

'ABC',

'Banglore');

console.log('My person name is ' + myPerson.name);

typescript

In the below example code, we will see how to create a form component.

Filename: person-form.component.ts

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

import { Component } from '@angular/core';

import { Hero } from '../person';

@Component({

selector: 'app-person-form',

templateUrl: './person-form.component.html',

styleUrls: ['./person-form.component.css']

})

export class PersonFormComponent {

awards = ['ABC', 'XYZ',

'Marvel'];

model = new Person(18, 'Raj', this.awards[0], 'Banglore');

submitted = false;

onSubmit() { this.submitted = true; }

// TODO: Remove this when we're done

get diagnostic() { return JSON.stringify(this.model); }

}

typescript

The code imports the Angular core library and the Person model you just created. The @Component selector value of "app-person-form" means you can drop this form in a parent template with a tag. The templateUrl property points to a separate file for the template HTML. You defined dummy data for model and awards, as befits a demo.

We need to add the FormsModule to the array of imports for the application module before you can use forms. In it, you identify the external modules you'll use in the application and declare the components that belong to this module, such as the PersonFormComponent. Replace the contents of app.component.html with .

## **Create HTML Form Template**

Filename: person-form.component.html

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

<div class="container">

<h1>Person Form</h1>

<form>

<div class="form-group">

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

<input type="text" class="form-control" id="firstName" required>

</div>

<div class="form-group">

<label for="address">Address</label>

<input type="text" class="form-control" id="address">

</div>

<button type="submit" class="btn btn-success">Submit</button>

</form>

</div>

typescript

Here, we are presenting two of the Person fields: firstName and the address. We added a Submit button at the bottom with some classes on it for styling. Let's add awards using \*ngFor iterator.

1

2

3

4

5

6

<div class="form-group">

<label for="award">Employee Award</label>

<select class="form-control" id="award" required>

<option \*ngFor="let award of awards" [value]="award">{{award}}</option>

</select>

</div>

typrscript

This code repeats thetag for each award in the list of awards. The award template input variable is a different award in each iteration; you display its name using the interpolation syntax.

Similarly, modify the HTML file to include the two-way data binding feature.

1

2

3

4

<input type="text" class="form-control" id="firstName"

required

[(ngModel)]="model.firstName" name="firstName">

FirstName is: {{model.firstName}}

typescript

We need one more addition to display the data. Declare a template variable for the form. Update the tag with #personForm="ngForm" as follows:

1

<form #personForm="ngForm">

typescript

The variable personForm is now a reference to the NgForm directive that governs the form as a whole.

## **Control State and Validity of Forms with ngModel**

ngModel in the forms not only will help us to achieve two-way data binding, but it also tells us if the user touched the control, if the value changed, or if the value became invalid.

Along with tracking the state, the ngModel directive will also update the control with special Angular CSS classes that reflect the state. Below are the different states.

### **Control States**

1. Control has been visited: ng-touched class is added if true and ng-untouched class is added if false.
2. Control's value has changed: ng-dirty class is added if true and ng-pristine class is added if false.
3. Control's value is valid: ng-valid class is added if a true and ng-invalid class is added if false.

We can improve the form using the control states. The below example shows how we can achieve this in the Angular Form.

1

2

3

4

5

6

7

8

9

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

<input type="text" class="form-control" id="firstName"

required

[(ngModel)]="model.firstName" firstName="firstName"

#firstName="ngModel">

<div [hidden]="firstName.valid || firstName.pristine"

class="alert alert-danger">

First Name is required

</div>

typescript

## **Using ngSubmit to Submit the Form**

Once the user fills the form, the user should be able to submit the form. The submit button in the below example will trigger a form submit because of its type "submit". To make the form useful, bind the form's ngSubmit event property to the person form component's onSubmit() method.

1

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

typescript

We will bind the form's overall validity via the personForm variable to the button's disabled property using an event binding. Let's see the code below.

1

<button type="submit" class="btn btn-success" [disabled]="!heroForm.form.valid">Submit</button>

typescript

## **Conclusion**

In this guide, we have explored the Template-driven forms in Angular. We have also seen how we can build or create a template-driven form from scratch and use it in our application.

You can learn more about Angular binding in my guide [Attribute, Class, and Style Bindings in Angular](https://app.pluralsight.com/guides/attribute-class-style-bindings-angular).