Angular 16 : Online Class

Forms in Angular

By: Sahosoft Solutions

Presented by : Chandan Kumar

Angular Forms

Forms are the main building blocks of any application. When we use forms for login, registration, submission, help request, etc.,

it is necessary that whatever form we are developing, they must be user-friendly. And, it should have the indication of what went wrong etc.

Forms are really important to collect the data from the users. Often, each website contains forms to collect the data.

Forms are the mainstay of business applications. You use forms to log in, submit a help request, place an order, book a flight, schedule a meeting, and perform countless other data-entry tasks.

Developing forms requires design skills as well as framework support for two-way data binding, change tracking, validation, and error handling.

Angular Forms

Angular provides 2 different ways to collect and validate the data from a user.

1. Template-driven forms
2. **Model-driven forms (Reactive forms)**

Template Driven Forms

Template driven forms are simple forms which can be used to develop forms. These are called template-driven as everything that we are going to use in an application is defined into the template that we are defining along with the component.

Angular Forms

Template Driven Forms

Prerequisite

We need to import **FormsModule** in an Application module file (**i.e.app. module.ts**).

Template Driven Forms features

- Easy to use.
- Suitable for simple scenarios and fails for complex scenarios.
- Similar to Angular 1.0.
- Two way data binding (using [(NgModel)] syntax).
- Minimal component code.
- Automatic track of the form and its data.
- Unit testing is another challenge.

Angular Forms

Model-driven forms (Reactive forms)

In a model-driven approach, the model which is created in the .ts file is responsible for handling all the user interactions/validations.

For this, first, we need to create the Model using Angular's inbuilt classes like **formGroup** and **formControl** and then we need to bind that model to the HTML form.

This approach uses the Reactive forms for developing the forms which favor the explicit management of data between the UI and the Model.

With this approach, we create the tree of Angular form controls and bind them in the native form controls. As we can create the form controls directly in the component, it makes it a bit easier to push the data between the data models and the UI elements.

Angular Forms

Model-driven forms (Reactive forms)

Prerequisite

we need to import **ReactiveFormsModule** in our **app.module.ts** file.

Reactive Forms Features

- More flexible, but needs a lot of practice
- Handles any complex scenarios
- More component code and less HTML markup
- Easier unit testing

Angular Forms

Which one is better - Template Driven or Reactive?

Neither Reactive nor Template Driven are better over each other. They both are different approaches, so you can use whichever suits your needs the most. You can even use both in the same application.

Angular 16 : Online Class

Template Driven Forms in Angular

By: Sahosoft Solutions

Presented by : Chandan Kumar

Template Driven Forms

Template driven forms are simple forms which can be used to develop forms. These are called template-driven as everything that we are going to use in a application is defined into the template that we are defining along with the component.

Let's see step by step how we can develop and use these forms in the application.

Prerequisite

We need to import **FormsModule** in an Application module file (**i.e.app. module.ts**).

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import {FormsModule} from '@angular/forms'
import { AppComponent } from './app.component';
@NgModule({
  declarations: [
    AppComponent,

  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Template Driven Forms

app.component.html

```
<div class="container">
   <div class="row">
       <div class="form_bg">
            <form #regForm='ngForm' (ngSubmit)="Register(regForm)" >
                 <h2 class="text-center">Registration page</h2>
                <br/>
                <div class="form-group">
                    <input type="text" class="form-
control" placeholder="First Name" name="firstname" ngModel >
                </div>
                <div class="form-group">
                    <input type="text" class="form-
control"  placeholder="Last Name" name="lastname" ngModel >
                </div>
                <div class="form-group">
                    <input type="email" class="form-
control" id="email" placeholder="Email" name="email" ngModel>

                </div>
                 <div class="form-group">
                </div>
                <br/>
                <div class="align-center">
                    <button type="submit" class="btn btn-
default" id="register"  >Register</button>
                </div>
            </form>
       </div>
   </div>
</div>
```

Template Driven Forms

NgForm

It is the directive which helps to create the control groups inside form directive. It is attached to the <form> element in HTML and supplements form tag with some additional features. Some interesting things we can say about View are that whenever we use the directive in the application view, we need to assign some selector with it and the form is the said selector in our case. The next thing that we need to consider is the ngModel attribute that we have assigned to each HTML control.

NgModel

When we add NgModel Directive to the control, all the inputs are registered in the NgForm.
It creates the instance of the FormControl and assigns it to the form control element. This control keeps track of the user information and the state and validation status of the form control.
Next important thing to consider is that when we use **NgModel** with the form tag or most importantly, with the **NgForm**,

We make use of the **name property** of HTML control. When we look at the above snippet, every control is assigned a name property and we have added the ngModel attribute to the control.

Template Driven Forms

Two main functionalities offered by **NgForm** and **NgModel** are the permission to retrieving all the values of the control associated with the form and then retrieving the overall state of controls in the form.

To expose ngForm in the application, we have used the following code snippet.

```
<form #regForm='ngForm' (ngSubmit)="Register(regForm)" >
```

In this, we are exporting the ngForm value in the local variable “regform”. Now, the question arises, whether or not we need to use this local variable. Well, the answer is no.

We are exporting ngForm in the local variable just to use some of the properties of the form and these properties are -

1. regForm.Value: It gives the object containing all the values of the field in the form.
2. regForm.Valid: This gives us the value indicating if the form is valid or not if it is valid value is true else value is false.
3. regForm.touched: It returns true or false when one of the field in the form is entered and touched.

Template Driven Forms

In the above case, we have noticed that the Form Tag has no action method or attribute specified there, so how so we post the data in the component?

Let's have a look at the (ngSubmit)="Register(regForm)". Here, we are using the Event Binding concept and we are binding which will call the Register method in the component. Instead of the submit event of the form, we are using ngSubmit which will send the actual HTTP request instead of just submitting the form.

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
@Component({
  selector: 'app-formdemobasics',
  templateUrl: './formdemobasics.component.html',
  styleUrls: ['./formdemobasics.component.css']
})
export class FormdemobasicsComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

  Register(regForm:NgForm){
    console.log(regForm);

  }
}
```

Angular 16 : Online Class

Template Driven Forms with Validation

By: Sahosoft Solutions

Presented by : Chandan Kumar

Template Driven Forms with Validation

Validation is an important aspect of programming. We cannot trust the user that's why we always want to validate the data. So, to prevent the user from entering wrong data and show some proper value, we check and ask the users to add the proper data.

Angular set of common validators like **minlength**, **maxlength**, **required**, **email**. We just need to add the validator directive to the control for assigning the controls the validators.

The following are the classes which will be attached whenever the state is changed.

- ✓ ng-touched: Controls have been visited
- ✓ ng-untouched: controls have not been visited

- ✓ ng-dirty : control value has been changed
- ✓ ng-pristine: Controls value have not been changed

- ✓ ng-valid : control values are valid
- ✓ ng-invalid : control values are invalid

So, in order to draw a border around the first name whenever a user visits the control and does not add any value, we need to use these classes.

Template Driven Forms with Validation

Here is how -

```
input.ng-invalid.ng-touched
{
   border-color: red;
}
```

Angular 16 : Online Class

Reactive forms in Angular

By: Sahosoft Solutions

Presented by : Chandan Kumar

Reactive forms

In a model-driven approach, the model which is created in the .ts file is responsible for handling all the user interactions/validations. For this, first, we need to create the Model using Angular’s inbuilt classes like **formGroup** and **formControl** and then, we need to bind that model to the HTML form.

This approach uses the Reactive forms for developing the forms which favor the explicit management of data between the UI and the Model. With this approach, we create the tree of Angular form controls and bind them in the native form controls. As we can create the form controls directly in the component, it makes it a bit easier to push the data between the data models and the UI elements.

Model driven forms are more powerful and more flexible.

There are some things that we can't do with template driven form but with model driven form we can perform those things. Like if we want to detect if any changes occur in any variable we can do it with model driven form , we can easily do two-way data binding with model drivven forms and that is very important in large applications.

Reactive forms

Reactive form in Angular is a technique to manage your form in a reactive manner, it means that you can manage your form and validation from our component itself .

These forms are the best option when we have a complex form requirement .

Compared to Template deriven forms, reactive forms are more suitable because we can define validations and model from component , that gives us more control on form .

Primarily it is also called "Model Driven Forms" , because model acts as a mediator between component and template .

It's really easy to implement, like using model in controller with form control object and binding it to component formcontrol template .

Reactive forms

Prerequisite

we need to import **ReactiveFormsModule** in our app.module.ts file.

Reactive Forms Features

- More flexible, but needs a lot of practice
- Handles any complex scenarios
- No data binding is done (immutable data model preferred by most developers)
- More component code and less HTML markup
- Easier unit testing

Reactive forms

You can see that in app.module.ts file, we have FormsModule in it, replace it with ReactiveFormsModule as below,

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
@NgModule({
    declarations: [
    AppComponent
    ],
    imports: [
        BrowserModule,
        ReactiveFormsModule
    ],
    bootstrap: [AppComponent]
    })
export class AppModule{}
```

Reactive forms

Now lets come back to app.component.ts file,we need to import 3 more libraries
import {FormGroup,FormControl,FormBuilder} from '@angular/forms'

Now in our app.component, we first need to add a form class and after that we need to define a function in order to detect if any of the elements in html changed or not.

```
ngOnInit() {
    this.form = new FormGroup({
        firstname: new FormControl(""),
        lastname: new FormControl(""),
        languages: new FormControl(""),
    })
}
```

Reactive forms

FormControl

It tracks the value of the controls and validates the individual control in the form.
In Reactive forms we initialize FormControl object to use form functionality into our component , which engaged with our html form .
And when we update our form control's value than it will directly be reflected to our FormControl object that we created previously .

FormGroup

Tracks the validity and state of the group of FormControl Instance or moreover, we can say the formgroup to be a collection of FormControls.
like :- Validations , name of input control etc ...

FormBuilder

This helps us to develop the forms along with their initial value and there validations.
Angular has a new helper Class called FormBuilder. FormBuilder allows us to explicitly declare forms in our components. This allows us to also explicitly list each form control’s validators.