Using FormBuilder In Angular

Introduction

Forms are an important part of any application. Applications use forms to enable users to log in, to update a profile, to enter sensitive information, and to perform many other data-entry tasks.

Angular helps us manage these various data handling tasks using two different types of forms: *reactive* and *template-driven*. Both capture user input events from the view, validate the user input, create a form model and a data model to update, and provide a way to track changes. There are also differences, however. Reactive forms are more robust: they're more scalable, reusable, and testable. Template-driven forms are useful for adding a simple form to an app, such as a user list signup form.

Using FormBuilder to Generate Controls

In this guide we will learn to build forms in Angular using *FormBuilder*. Creating multiple form control instances manually can become repetitive when dealing with multiple forms. The FormBuilder service provides convenient methods for generating controls.

To use the FormBuilder service, follow the steps below:

* Import the FormBuilder class.
* Inject the FormBuilder service.
* Generate the form contents.

Let's look at the following examples of how to refactor a component called EmployeeDetailsEditor to use the FormBuilder service to create form control and form group instances.

Import the FormBuilder Class

Import the FormBuilder class from the @angular/forms package.

File name : employeeDetails-editor.component.ts

1

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

typescript

Inject the FormBuilder Service

The FormBuilder service is an injectable provider that is provided with the reactive forms module. We will inject this dependency by adding it to the component constructor.

File name : employeeDetails-editor.component.ts

1

constructor(private fb: FormBuilder) { }

typescript

Generate Form Controls

The FormBuilder service has three methods: control(), group(), and array(). These are factory methods for generating instances in your component classes, including form controls, form groups, and form arrays.

Use the group() method to create the employeeDetailsForm controls.

File name : employeeDetails-editor.component.ts

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

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

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

@Component({

selector: 'app-employeeDetails-editor',

templateUrl: './employeeDetails-editor.component.html',

styleUrls: ['./employeeDetails-editor.component.css']

})

export class EmployeeDetailsEditorComponent {

employeeDetailsForm = this.fb.group({

firstName: [''],

lastName: [''],

address: this.fb.group({

street: [''],

city: [''],

state: [''],

zip: ['']

}),

});

constructor(private fb: FormBuilder) { }

}

typescript

In the example above, you used the group() method with the same object to define the properties in the model. The value for each control name is an array containing the initial value as the first item in the array.

You can define the control with just the initial value, but if your controls need sync or async validation, add sync and async validators as the second and third items in the array.

Validating Form Input

Form validation ensures that user input is complete and correct. In this section we will cover adding a single validator to a form control and displaying the overall form status.

To use form validation, follow these steps:

* Import a validator function in your form component.
* Add the validator to the field in the form.
* Add logic to handle the validation status.

One of the most commonly used validations is making a field required. The following example shows how to add a required validation to the firstName control and display the result of validation.

Import a Validator Function

Reactive forms include a set of validator functions for common use cases. These functions receive a control to validate against and return an error object or a null value based on the validation check.

Let's import the Validators class from the @angular/forms package as shown below.

File name : employeeDetails-editor.component.ts

1

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

typescript

Make a Field Required

Add Validators.required in the EmployeeDetailsEditor component static method as the second item in the array for the firstName control.

File name : employeeDetails-editor.component.ts

1

2

3

4

5

6

7

8

9

10

emplyeeDetailsForm = this.fb.group({

firstName: ['', Validators.required],

lastName: [''],

address: this.fb.group({

street: [''],

city: [''],

state: [''],

zip: ['']

}),

});

typescript

HTML5 has a set of built-in attributes that you can use for native validation, including required, minlength, and maxlength. You can take advantage of these optional attributes on your form input elements. Add the required attribute to the firstName input element.

1

<input type="text" formControlName="firstName" required>

html

You must use these HTML5 validation attributes in combination with the built-in validators provided by Angular's reactive forms. Using these in combination prevents errors when the expression is changed after the template has been checked.

Display Form Status

You can now add a required field to the form control. Its initial status is invalid. This invalid status propagates to the parent form group element, making its status invalid. Access the current status of the form group instance through its status property.

Display the current status of employeeDetailsForm using interpolation.

File name : employeeDetails-editor.component.html

1

2

3

<p>

Form Status: {{ employeeDetailsForm.status }}

</p>

html

The Submit button will be disabled because employeeDetailsForm is invalid due to the required firstName form control. After you fill out the firstName input, the form becomes valid and the Submit button is enabled.

Conclusion

In this guide, we explored how to use FormBuilder in Angular to build reactive forms. We also learned how to validate form input using the Validators function.

You can learn more about using forms in Angular in another one of my guides: [Creating Reactive Forms in Angular](https://app.pluralsight.com/guides/angular-creating-reactive-forms).