Creating Reactive Forms in Angular

Introduction

Reactive forms in Angular provide a way to achieve a model-driven approach to handling form inputs whose values change over time. In this guide, we will learn about how to create and update a simple form- control and how to progress using multiple controls in a group, validate form values, and implement more advanced forms.

Reactive Forms are useful in managing the state of the form at any given point in time since reactive forms are explicit and immutable. Each change to the form state returns a new state, which maintains the integrity of the model between changes. Reactive forms are built around observable streams, where form inputs and values are provided as streams of input values, which can be accessed synchronously.

## Building the Reactive Form Application

In this section, we will see how to build a form application using the Reactive Forms in Angular. To use reactive forms, we need to import ReactiveFormsModule from the @angular/forms package and add it to your NgModule's imports array. To register a single form control, import the FormControl class into your component and create a new instance of the form control to save as a class property.

Filename: employee-editor.component.ts

1

2

3

4

5

6

7

8

9

10

11

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

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

@Component({

selector: 'app-employee-editor',

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

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

})

export class EmployeeEditorComponent {

employee = new FormControl('');

}

typescript

Here, we are using the constructor of FormControl to set its initial value. After creating the control in the component class, then we need to update the template with the form control using the formControl binding provided by FormControlDirective included in ReactiveFormModule.

Filename: employee-editor.component.html

1

2

3

4

<label>

Employee:

<input type="text" [formControl]="employee">

</label>

html

The form control assigned to the name is displayed when the component is added to a template.

Filename: app.component.html

1

<app-employee-editor></app-employee-editor>

typescript

## Managing Reactive Form Control Values

We can manipulate the current state and value through the component class or the component template. The following examples display the value of the form control instance and how we can change it. Using the valueChanges observable, you can listen for changes in the form's value in the template using AsyncPipe or in the component class using the subscribe () method. With the value property, which gives you a snapshot of the current value, we can display the value.

Filename: employee-editor.component.html

1

2

3

<p>

Value: {{ employeeName.value }}

</p>

html

The displayed value will change as you update the form control element.

We can change the control's value in Reactive Forms programmatically, which gives you the flexibility to update the value without user interaction. A form control instance provides a setValue() method that updates the value of the form control and validates the structure of the value provided against the control's structure.

Filename: employee-editor.component.ts

1

2

3

updateEmployeeName() {

this.employeeName.setValue('Yallaling');

}

typescript

A form control instance provides a setValue() method that updates the value of the form control and validates the structure of the value provided against the control's structure.

Filename: employee-editor.component.html

1

2

3

<p>

<button (click)="updateEmployeeName()">Update Employee Name</button>

</p>

typescript

## Grouping Form Controls

A form group instance tracks the form state of a group of form control instances, instead of a single instance.

The following example shows how to manage multiple form control instances in a single group.

Filename: employeeProfile-editor.component.ts

1

2

3

4

5

6

7

8

9

10

11

12

13

14

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

import { FormGroup, FormControl } from '@angular/forms';

@Component({

selector: 'app-employeeProfile-editor',

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

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

})

export class EmployeeProfileEditorComponent {

employeeProfile = new FormGroup({

employeeFirstName: new FormControl(''),

employeeLastName: new FormControl(''),

});

}

typescript

Each of the individual form controls is now collected within a group. A FormGroup instance provides its model value as an object reduced from the values of each control in the group.

Next, let's associate the form group model and the view. A form group tracks the status and changes for each of its controls, so that if one of the controls changes, the parent control also emits a new status or value change.

Filename: employeeProfile-editor.component.html

1

2

3

4

5

6

7

8

9

10

11

12

13

<form [formGroup]="employeeProfileForm">

<label>

Employee First Name:

<input type="text" formControlName="employeeFirstName">

</label>

<label>

Employee Last Name:

<input type="text" formControlName="emploeeLastName">

</label>

</form>

html

## Saving Form Data

The FormGroup directive listens for the submit event emitted by the form element and emits a ngSubmit event that you can bind to a call back function.

Filename: employeeProfile-editor.component.html

1

<form [formGroup]="profileForm" (ngSubmit)="onSubmit()">

html

The onSubmit() method in the EmployeeProfileEditor component captures the current value of employeeProfileForm. Use the EventEmitter to keep the form encapsulated and to provide the form value outside the component.

1

2

3

onSubmit() {

console.warn(this.profileForm.value);

}

typescript

The submit event is emitted by the form tag using the native DOM event. You trigger the event by clicking a button with submit type. This allows the user to press the Enter key to submit the completed form.

## Displaying the Reactive Form Component

To display the EmployeeProfileEditor component that contains the form, add it to a component template.

Filename: app.component.html

1

<app-employeeProfile-editor></app-employeeProfile-editor>

typescript

EmployeeProfileEditor allows you to manage the form control instances for the firstName and lastName controls within the form group instance.

## Reactive Form Validation in Angular

Form validation will be useful for validating the user input and to ensure its completeness and correctness.

Reactive forms include validator functions that will receive control to validate against and return an error object or a null value based on the validation check.

### **1. Importing a Validator Function**

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

Filename: employeeProfile-editor.component.ts

1

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

typescript

### **2. Making a Field Required in Form**

In this section, we will see how to add a required validation to the employeeFirstName control.

Now, let's add the EmployeeProfileEditor component. Add the Validators.required static method as the second item in the array for the employeeFirstName control.

Filename: employeeProfile-editor.component.ts

1

2

3

4

5

6

7

8

9

10

profileForm = this.fb.group({

employeeFirstName: ['', Validators.required],

employeeLastName: [''],

employeeAddress: 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 employeeFirstName input element.

Filename: employeeProfile-editor.component.html

1

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

html

### **Displaying Form Controls Status**

The initial status, when you add a form control, is invalid. This invalid status propagates to the parent form group element, making it's status invalid. The parent form group element accesses the current status of the form group instance through its status property. It will then display the current status of profileForm using interpolation.

Filename: employeeProfile-editor.component.html

1

2

3

<p>

Form Status: {{ profileForm.status }}

</p>

html

## Conclusion

In this guide, we have explored the Reactive Forms in Angular. We have also seen how we can build or create a reactive form from scratch and use it in our application.

You can learn more about Angular binding in my guide [Creating Template Driven Forms in Angular](https://app.pluralsight.com/guides/angular-creating-template-driven-forms).