Overview of Forms in Angular

Introduction

Forms in Angular enable users to work with user inputs like login, register, save, update, and many other data entry tasks.

There are two ways through which we can handle user inputs in Angular: reactive forms and template-driven forms. Both capture user input events from the view, validate the user input, create a form model and data model to update, and provide a way to track changes.

Reactive Forms

Reactive forms are more scalable, testable, reusable, and robust. If you build applications which involve more reactiveness, then use reactive forms of Angular to build applications.

Let's consider an example where we are setting up the Reactive form using the form model FormControl instance.

Filename: example.component.ts

1

2

3

4

5

6

7

8

9

10

11

12

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

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

@Component({

selector: 'app-reactive-example',

template: `

Example input: <input type="text" [formControl]="exampleControl">

`

})

export class ExampleComponent {

exampleControl = new FormControl('');

}

typescript

Template-Driven Forms

Template-driven forms are not as scalable as Reactive forms. If you want a very basic form requirement and logic that can be managed solely in the template, use template-driven forms.

Let's consider an example for a template-driven form where we are binding the text value exampleText to the input element.

Filename: example.component.ts

1

2

3

4

5

6

7

8

9

10

11

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

@Component({

selector: 'app-template-example',

template: `

Example Text: <input type="text" [(ngModel)]="exampleText">

`

})

export class ExampleComponent {

exampleText = '';

}

typescript

From this guide, you will be able to identify which form type to choose based on your requirement.

Key Differences Between Reactive Forms and Template-Driven Forms:

1. The data model in reactive forms is more structured than template-driven forms.
2. Reactive forms are synchronous in nature, whereas template-driven forms are asynchronous.
3. Form validations in reactive forms are handled through functions, whereas in template-driven forms they are handled through directives.
4. Reactive forms are immutable in nature, whereas template-driven forms are mutable.
5. Reactive forms are more explicit and created in the component class and template-driven forms are less explicit and created by directives.

Building Blocks of Angular Forms

1. FormControl control instance tracks the value and validation status of an individual form control.
2. FormGroup control instance tracks the same values and status for a collection of form controls.
3. FormArray control instance tracks the same values and status for an array of form controls.
4. ControlValueAccessor control instance creates a bridge between Angular FormControl instances and native DOM elements.

Forms in Angular use a form model to keep track of value changes between Angular forms and form input elements.

Data Flow in Angular Forms

Data flow in Angular is handled differently in both the reactive and template-driven forms.

The data flow examples below begin with the example text input field example from above, and then show how changes to example text are handled in reactive forms compared to template-driven forms.

Data Flow in Reactive Forms

Updates or changes from the view (or DOM) to the model and from the model to the view are synchronous and aren't dependent on the UI rendering. In reactive forms, each form element in the view is directly linked to a form model (FormControl instance).

Let's consider an example shown below and go through the steps of data flow in reactive forms.

Filename: example.component.ts

1

2

3

4

5

6

7

8

9

10

11

12

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

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

@Component({

selector: 'app-reactive-example',

template: `

Example input: <input type="text" [formControl]="exampleControl">

`

})

export class ExampleComponent {

exampleControl = new FormControl('');

}

typescript

The steps below outline the data flow from view to model.

1. The user types a value into the input element, in this case, the text value says "Hello World".
2. The form input element emits an "input" event with the latest value.
3. The control value accessor listening for events on the form input element immediately relays the new value to the FormControl instance.
4. The FormControl instance emits the new value through the valueChanges observable.
5. Any subscribers to the valueChanges observable receive the new value.

The steps below outline the data flow from model to view.

1. The user calls the exampleControl.setValue() method, which updates the FormControl value.
2. The FormControl instance emits the new value through the valueChanges observable.
3. Any subscribers to the valueChanges observable receive the new value.
4. The control value accessor on the form input element updates the element with the new value.

Data Flow in Template-Driven Forms

In template-driven forms, each form element is linked to a directive that manages the form model internally.

Let's consider an example shown below and go through the steps of data flow in template-driven forms.

Filename: example.component.ts

1

2

3

4

5

6

7

8

9

10

11

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

@Component({

selector: 'app-template-example',

template: `

Example Text: <input type="text" [(ngModel)]="exampleText">

`

})

export class ExampleComponent {

exampleText = '';

}

typescript

The steps below outline the data flow from view to model.

1. The user types "Hello World" into the input element.
2. The input element emits an "input" event with the value "Hello World".
3. The control value accessor attached to the input triggers the setValue() method on the FormControl instance.
4. The FormControl instance emits the new value through the valueChanges observable.
5. Any subscribers to the valueChanges observable receive the new value.
6. The control value accessor also calls the NgModel.viewToModelUpdate() method which emits an ngModelChange event.
7. Because the component template uses two-way data binding for the exampleText property, the exampleText property in the component is updated to the value emitted by the ngModelChange event.

The steps below outline the data flow from model to view.

1. The exampleText value is updated in the component.
2. Change detection begins.
3. During change detection, the ngOnChanges lifecycle hook is called on the NgModel directive instance because the value of one of its inputs has changed.
4. The ngOnChanges() method queues an async task to set the value for the internal FormControl instance.
5. Change detection completes.
6. On the next tick, the task to set the FormControl instance value is executed.
7. The FormControl instance emits the latest value through the valueChanges observable.
8. Any subscribers to the valueChanges observable receive the new value.
9. The control value accessor updates the form input element in the view with the latest exampleText value.

Form Validation in Angular

Validation is one of the important parts of forms for security and other reasons. Angular provides a set of built-in validators as well as the ability to create custom validators.

* In reactive forms, validations are handled through custom validators as functions that receive control to validate.
* In template-driven forms validations are handled through directives and must provide custom validator directives that wrap validation functions.

Conclusion

In this guide, we have explored the forms in angular and we saw different types of forms like reactive and template-driven forms.

You can learn more about Angular in my guide [Angular Data Binding Overview](https://app.pluralsight.com/guides/angular-data-binding-overview).