Angular JS 2

Created by :

Sangeeta Joshi

Agenda

- Template driven
- Reactive
- Form Validations

Forms

- Angular offers two form-building technologies:

  reactive forms

  template-driven forms
- The two technologies belong to the @angular/forms library and share a common set of form control classes.
- they diverge markedly in philosophy, programming style, and technique.
- They even have their own modules:

  ReactiveFormsModule and FormsModule.

Template Driven Forms

- Template-driven forms, introduced take a completely different approach.

- We place HTML form controls (such as <input> and <select>) in ***component template*** and bind them to data model properties in the component, using directives like ***ngModel.***

- We don't create Angular form control objects. Angular directives create them for us, using the information in our data bindings.

Template Driven Forms

- We don't push and pull data values.
- Angular handles that for us with ngModel.
- Angular updates the mutable data model with user changes as they happen.
- For this reason, the ngModel directive is not part of the ReactiveFormsModule.
- While this means less code in the component class***, template-driven forms are asynchronous*** which may complicate development in more advanced scenarios.

Reactive Forms

- Angular reactive forms favour explicit data management

  flowing between a non-UI data model (typically retrieved from a server)

  &

  a UI-oriented form model that retains the states and values of the HTML controls on screen.
- Reactive forms offer the ease of using reactive patterns, testing, and validation.
- With reactive forms, you create ***a tree of Angular form control objects in the component class*** and bind them to native form control elements in the component template

Reactive Forms

- We create and manipulate form control objects directly in the component class.
- As the component class has immediate access to both ***the data model and the form control structure*** :
  - we can push data model values into form controls
  - pull user-changed values back out.
  - The component can observe changes in form control state and react to those changes.

Reactive Forms

- advantage of working with form control objects directly :
  - value and validity updates are always synchronous and under our control.
  - We won't encounter the timing issues that sometimes plague a template-driven form
  - reactive forms can be easier to unit test.

Reactive Forms

- In keeping with the reactive paradigm:
  - the component preserves the immutability of the data model, treating it as a pure source of original values.
  - Rather than update the data model directly, the component extracts user changes and forwards them to an external component or service, (for such as saving them) and returns a new data model to the component that reflects the updated model state.

Reactive Forms

- Reactive forms or data driven forms are new approach taken by the angular team.
- With Reactive forms we don’t rely heavily on template for structuring our forms.
- we use the underlying API’s exposed by angular which is called the Reactive API and can be used by importing a module called ReactiveModule to create the forms.
- This way we can very easily get rid of the declarative nature of the template forms
- Improved testability.
- It also makes the logic sit in one place does not clutter the form template

Reactive Forms

- In template forms we use ngModel ngSubmit and localRef to the form
- and structure it in the template before sending it to the method we want our form to be submitted to.
- with Reactive forms all the declaration and structuring of the forms happens in the code(JS/TS) that model is then used in templates to complement the form structure.

steps to create a reactive form.

- Import Reactive forms module
- Define the form(formGroup) in code. Generally in a component.
- Wire the form created in component to the form in template.

Template Driven Vs. Reactive

Template Driven Forms	Reactive Forms Features
• Easier to use • Suitable for simple scenarios • Uses Two way data binding(using [(NgModel)] syntax) • Minimal component code • Automatic track of the form and its data(handled by Angular) • Unit testing is challenge	• Easier Unit testing • More flexible,So Handles any complex scenarios. • Reactive transformations can be made possible such as Adding elements dynamically • No data binding is done(Immutable data model) • More component code and less HTML markup

Form Validation

Template-driven validation

- To add validation to a template-driven form, you add the same validation attributes as with native HTML form validation.

- Angular uses directives to match these attributes with validator functions in the framework.

- Every time the value of a form control changes, Angular runs validation and generates either a list of validation errors, which results in an INVALID status, or null, which results in a VALID status.

- You can then inspect the control's state by exporting ngModel to a local template variable.

Form Validation

- The following example exports NgModel into a variable called name:
- <input id="name" name="name" class="form-control" required minlength="4" forbiddenName="bob" [(ngModel)]="hero.name" #name="ngModel" >
- <div *ngIf="name.invalid && (name.dirty || name.touched>
- <div *ngIf="name.errors.required"> Name is required. </div>
- <div *ngIf="name.errors.minlength"> Name must be at least 4 characters long. </div>
- <div *ngIf="name.errors.forbiddenName"> Name cannot be Bob. </div>

Form Validation

- #name="ngModel" exports NgModel into a local variable called name.
- NgModel mirrors many of the properties of its underlying FormControl instance, so you can use this in the template to check for control states such as valid and dirty

Why check dirty and touched?

- You may not want your application to display errors before the user has a chance to edit the form.
- The checks for dirty and touched prevent errors from showing until the user does one of two things:
  - changes the value, turning the control dirty;
  - or blurs the form control element, setting the control to touched.

Reactive form validation

- In a reactive form, the source of truth is the component class. Instead of adding validators through attributes in the template, you add validator functions directly to the form control model in the component class. Angular then calls these functions whenever the value of the control changes.

Built-in validators

- You can choose to write your own validator functions, or you can use some of Angular's built-in validators.
- The same built-in validators that are available as attributes in template-driven forms, such as required and minlength, are all available to use as functions from the Validators class.
- To update the form to be a reactive form, you can use some of the same built-in validators—this time, in function form.

Built-in validators

- ngOnInit(): void {
-   this.heroForm = new FormGroup({
-     'name': new FormControl(this.hero.name, [
-       Validators.required,
-       Validators.minLength(4),
-       forbiddenNameValidator(/bob/i) // <-- Here's how you pass in the custom validator.
-     ]),
-     'alterEgo': new FormControl(this.hero.alterEgo),
-     'power': new FormControl(this.hero.power, Validators.required)
-   });
- }

- get name() { return this.heroForm.get('name'); }

- get power() { return this.heroForm.get('power'); }