academeez

@angular/forms

3. Building forms using Template Driven approach

Template Driven

academeez

- In previous lesson we learned that one of the toolset angular gives us for dealing with forms is the template driven approach
- With template driven approach we describe the structure of the form using **NgForm, NgModel** directives
- Instead of FormGroup we have a directive that hides the FormGroup called **NgForm**
- Instead of **FormControl** we have a directive that hides the **FormControl** called **NgModel**

NgForm

- The selector of this directive is the <form> tag
- The directive will be added to every form tag when you add the **FormsModule** to the imports array

```
export declare class NgForm extends ... {
  readonly submitted: boolean;
  form: FormGroup;
  ngSubmit: EventEmitter<any>;
  get controls(): {
    [key: string]: AbstractControl;
  };

  // from inheritance we get these methods as well
  get valid(): boolean | null;
  get invalid(): boolean | null;
  reset(value?: any): void;
}
```

NgForm

- The ngForm creates a FormGroup for the form
- It will attach controls to the FormGroup
- It will track the status of the form
- Has a submit Event
- The directive also disables HTML5 validation on the form

ngSubmit

- @Output on the **NgForm** directive, use this event to subscribe to form submission
- $event will be the original form submission event
- Safer to use than the original submit method

NgModel

academeez

- NgModel can be used for 2 way binding
- NgModel can be used to add control to NgForm

```
@Component({
  selector: 'ngmodel-demo',
  template: `
    <form>
      <input type="email" name="email" ngModel ↗
      <input type="password" name="password" [(ngModel)]="password" ↗
    ↙form>
  `
})
```

NgModel

- NgModel creates a form control and binds it with the input
- It will manage the form control status, validation and values for us

```
export declare class NgModel extends ... {
  readonly control: FormControl;
  name: string;

  // from inheritance we get these methods as well
  get valid(): boolean | null;
  get invalid(): boolean | null;
  reset(value?: any): void;
}
```

NgModel - 2 way binding

academeez

- NgModel can be used for 2 way binding by wrapping it with: [(ngModel)]=“var”

```
@Component({
  selector: 'two-way-binding',
  template: `
    <input type="password" name="password" [(ngModel)]="title" ↗
  `
})
export class TwoWayBindingComponent {
  title: string = 'initial value';
}
```

NgModel - adds control to NgForm

- NgModel wrapped in form will add a control to NgForm
- The control is registered under the name attribute - name is required

```
@Component({
  selector: 'ngmodel-add-ngform',
  template: `
    <form>
      <input type="password" name="password" ngModel ↗

      <!-- Error no name -→
      <input type="email" ngModel ↗
    ↙form>
  `
})
```

Grab NgForm in Template reference variable

academeez

- You can grab ngForm with Template Reference Variable
- Can be useful to display error messages
- Can be useful to grab form values with no need for 2 way binding

```
@Component({
  selector: 'grab-ngform',
  template: `
    <form #myForm="ngForm" (ngSubmit)="login(myForm)">
      <input type="password" name="password" ngModel ↗
    ↙form>
  `
})
```

Summary

- The template driven approach provides us with the ngModel and NgForm for building our form
- The NgForm is added automatically for the form tag
- We can use it to connect to (ngSubmit)
- NgModel will register a control with the NgForm using the name attribute
- NgModel can also be for 2 way binding

academeez

Thank You

Next Lesson: 4. Reactive forms