ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Template-driven Forms and Validation

Alex Thalhammer

Outline

- Approaches
- Template-driven forms
  - How to use
  - Validation
- Reactive forms
  - How to use
  - Validation

Forms in Angular

Template-driven

- Add ngModel within the HTML-template
- Angular creates object tree for form
- FormsModule

Reactive

- We create the object tree in our component (TS-file)
- More control, more power
- ReactiveFormsModule

Data-driven

- Angular generates a form for a data model
- Handed over to the community ("formly")

ANGULAR ARCHITECTS INSIDE KNOWLEDGE SOFTWARE ARCHITECT

Template-driven Forms

Template-driven Forms

```
export class FlightSearchComponent {

    from: string;
    to: string;

    constructor(private flightService: FlightService) {
            this.from = 'Graz';
            this.to = 'Hamburg';
    }
}
```

Template-driven Forms

```
export class FlightSearchComponent {

    from = 'Graz';
    to = 'Hamburg';

    constructor(private flightService: FlightService) {}
}
```

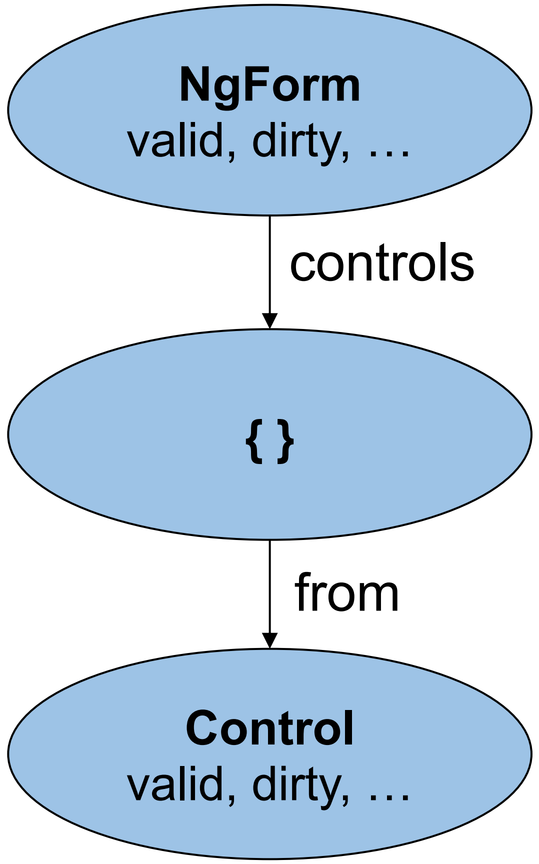
View

```
<form>

    <input type="text" name="from"
        [(ngModel)]="from" required minlength="3">

    […]

</form>
```


$$\begin{tikzcd}
\begin{array}{c}\textbf{NgForm}\\ \text{valid, dirty, …}\end{array} \arrow[d, "\text{controls}"] \\
\{\,\} \arrow[d, "\text{from}"] \\
\begin{array}{c}\textbf{Control}\\ \text{valid, dirty, …}\end{array}
\end{tikzcd}$$

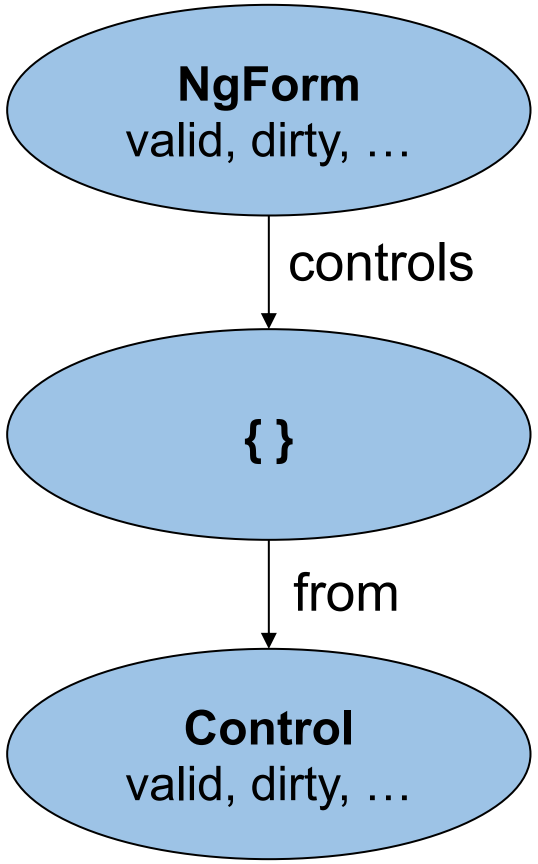
View

```
<form #f="ngForm">

    <input type="text" name="from"
        [(ngModel)]="from" required minlength="3">

    […]

</form>
```



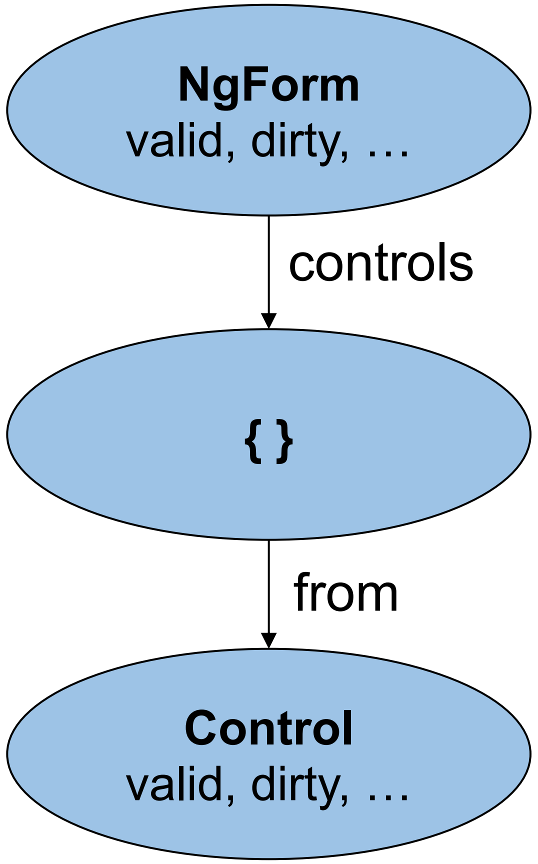
View

```
<form #f="ngForm">

   <input type="text" name="from"
      [(ngModel)]="from" required minlength="3">

   <div *ngIf="!f.controls.from.valid">
      …Error…
   </div>

</form>
```



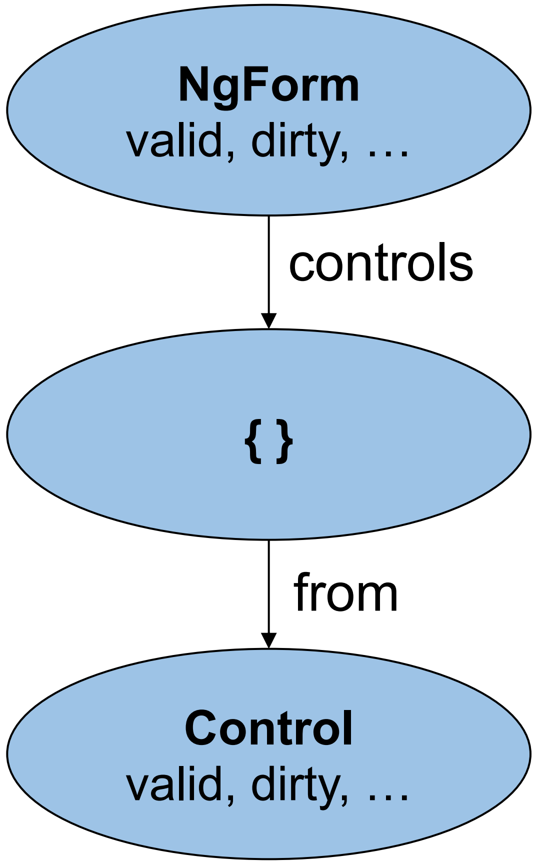
View

```
<form #f="ngForm">

    <input type="text" name="from"
       [(ngModel)]="from" required minlength="3">

    <div *ngIf="!f.controls?.from?.valid">
       …Error…
    </div>

</form>
```



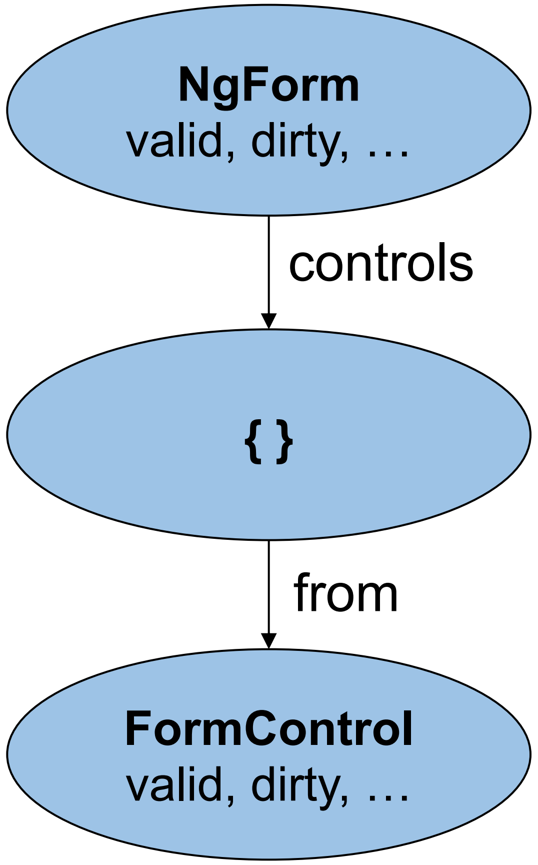
View

```
<form #f="ngForm">

    <input type="text" name="from"
        [(ngModel)]="from" required minlength="3">

    <div *ngIf="!f?.controls?.from?.valid">
        …Error…
    </div>

    <div
        *ngIf="f?.controls?.from?.hasError('required')">
        …Error…
    </div>
</form>
```



DEMO

LAB

Own Valididators

Page ▪ 14

Directives

- Add behaviour to a component or any other HTML tag

- Built in examples
  - Attribute directives: ngModel, ngClass, ngStyle
  - Structural directives: *ngIf, *ngFor, *ngSwitch

- Custom attribute directives
  - E.g. validation directive

- No template (in contrast to components)

Validation directive

```
<input [(ngModel)]="from" name="from" city>
```

Validation directive

```
@Directive({
    selector: 'input[city]'
})
export class CityValidatorDirective implements Validator {

    validate(c: AbstractControl): ValidationErrors | null {
        const value = c.value;

        […]

        if (…) return { city: true }; // error

        return null; // no error
    }
}
```

Validation directive

```
@Directive({
    selector: 'input[city]',
    providers: [{ provide: NG_VALIDATORS,
                  useExisting: CityValidatorDirective, multi: true}]
})
export class CityValidatorDirective implements Validator {

    validate(c: AbstractControl): ValidationErrors | null {
        const value = c.value;

        […]

        if (…) return { city: true };

        return null; // no error
    }
}
```

.hasError('city')

Using attributes

```
<input [(ngModel)]="from" name="from"
            [city]="['Graz', 'Hamburg', 'Zürich']">
```

Using attributes

```
@Directive({
    selector: 'input[city]',
    providers: [{ provide: NG_VALIDATORS,
                  useExisting: CityValidatorDirective,
                  multi: true }]
})
export class CityValidatorDirective implements Validator {

    @Input() city: string[];

    validate(c: AbstractControl): ValidationErrors | null {
        […]
    }
}
```

Using attributes

```
@Directive({
    selector: 'input[city]',
    providers: [{ provide: NG_VALIDATORS,
                  useExisting: CityValidatorDirective,
                  multi: true }]
})
export class CityValidatorDirective implements Validator {

    @Input() city: string[];
    @Input() strategy: string;

    validate(c: AbstractControl): ValidationErrors | null {
        […]
    }
}
```

Using attributes

```
<input [(ngModel)]="from" name="from"
          [city]="['Graz', 'Hamburg', 'Zürich']" [strategy]="'strict'">
```

Using attributes

```
<input [(ngModel)]="from" name="from"
       city="Graz, Hamburg, Zürich" strategy="strict">
```

DEMO

Asynchronous validation directives

```
@Directive({
    selector: 'input[asyncCity]',
    providers: [ … ]
})
export class AsyncCityValidatorDirective implements AsyncValidator {

    validate(control: AbstractControl): Observable<ValidationErrors | null> {
        […]
    }

}
```

Asynchronous validation directives

Token: NG_ASYNC_VALIDATORS

Multifield Validators

```
@Directive({
    selector: 'form[roundTrip]',
    providers: [ … ]
})
export class RoundTripValidatorDirective implements Validator {

    validate(control: AbstractControl): ValidationErrors | null {
        […]
    }
}
```

Multifield Validators

```
export class RoundTripValidatorDirective implements Validator {

    validate(control: AbstractControl): ValidationErrors | null {
        let group = control as FormGroup;

        let from = group.controls['from'];
        let to = group.controls['to'];

        if (!from || !to) return { };

        […]
}
```

Multifield Validators

```
export class RoundTripValidatorDirective implements Validator {

    validate(control: AbstractControl): ValidationErrors | null {
        let group = control as FormGroup;

        let from = group.controls['from'];
        let to = group.controls['to'];

        if (!from || !to) return { };

        if (from.value === to.value) return { roundTrip: true };

        return { };
    }
}
```

DEMO

LAB