



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)
- More control, more power
- ReactiveFormsModule



# Template-driven Forms



# Template-driven Forms

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



# Template-driven Forms

```
export class FlightSearchComponent {  
  from = 'Graz';  
  to = 'Hamburg';  
  
  constructor(private flightService: FlightService) {}  
}
```



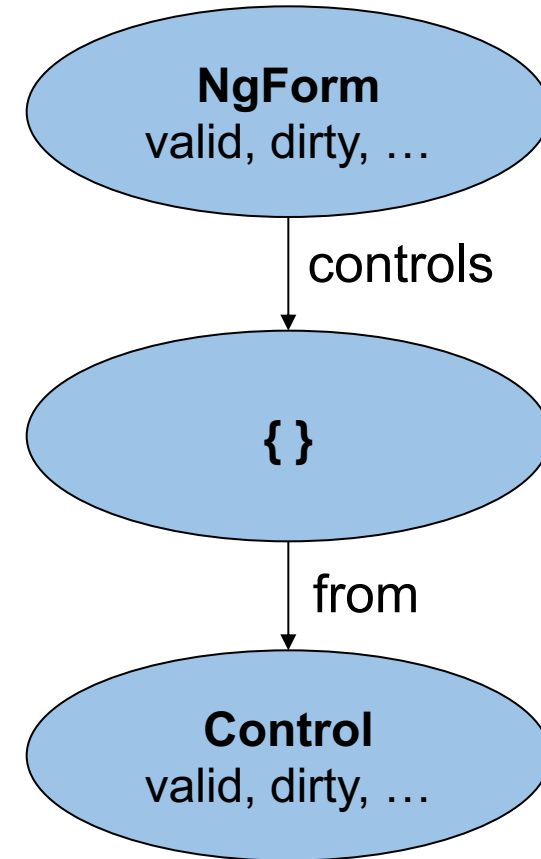
# View

```
<form>

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

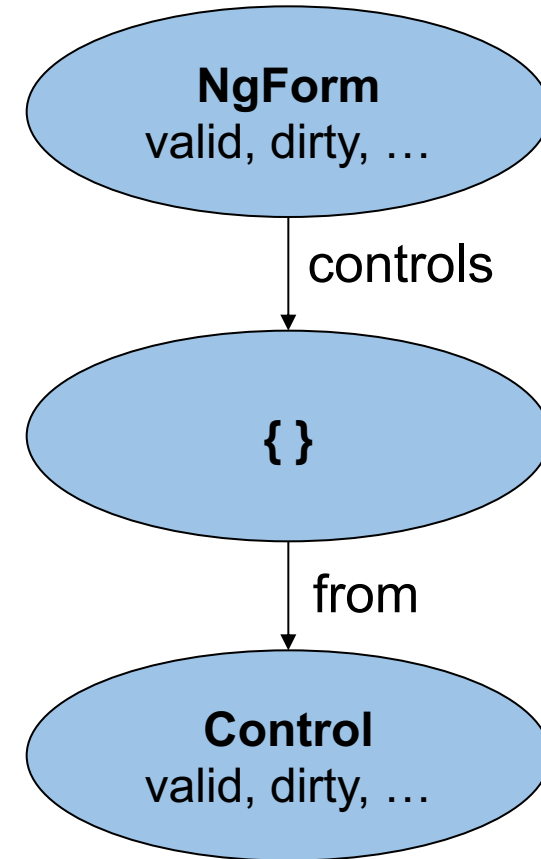
  [...]

</form>
```



# View

```
<form #flightSearchForm="ngForm">  
  <input type="text" name="from"  
    [(ngModel)]="from" required minlength="3">  
  [...]  
</form>
```





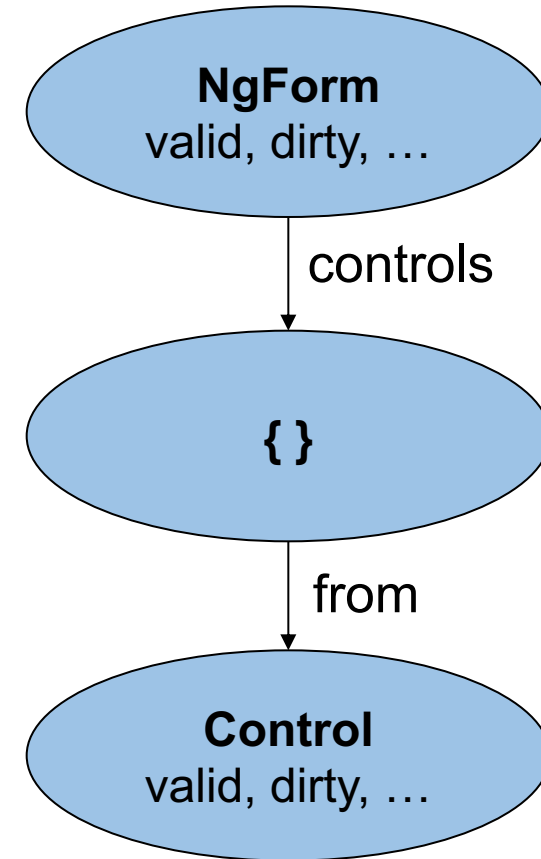
# View

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

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

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

</form>
```



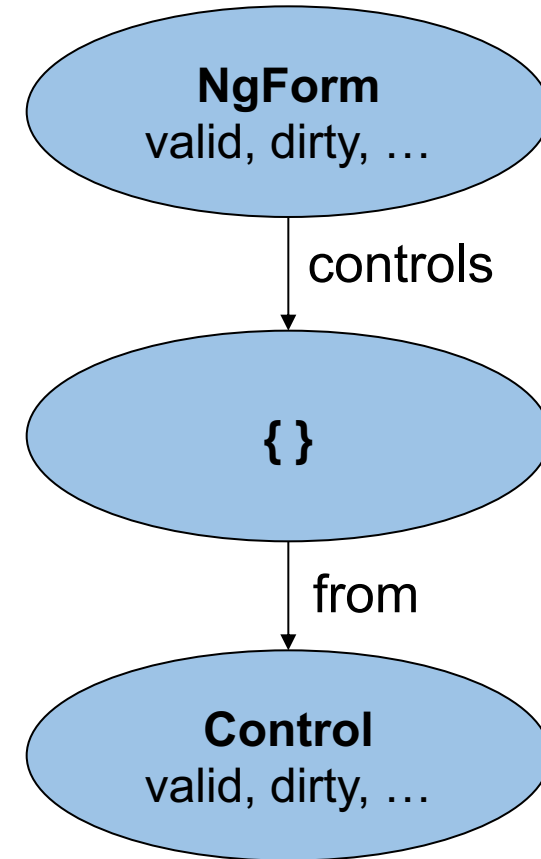
# View

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

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

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

</form>
```



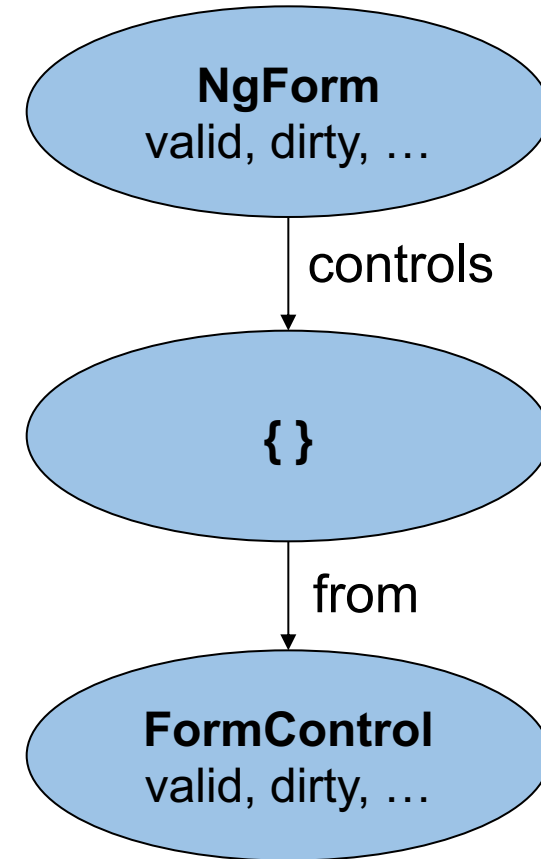
# View

```
<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')">
    ...Field is required...
  </div>
</form>
```



# DEMO



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# LAB



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# Own Validators



# 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}; --> .hasError('city')
    return null; // no error
  }
}
```

# 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 = '';

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



# Using attributes

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

# DEMO



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# 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 {  
    const group = control as FormGroup;  
  
    const from = group.controls.from;  
    const to = group.controls.to;  
  
    if (!from || !to) return null;  
  
    [...]  
  }  
}
```



# Multifield Validators

```
export class RoundTripValidatorDirective implements Validator {  
  validate(control: AbstractControl): ValidationErrors | null {  
    const group = control as FormGroup;  
  
    const from = group.controls.from;  
    const to = group.controls.to;  
  
    if (!from || !to) return null;  
  
    if (from.value && from.value === to.value) return { roundTrip: true };  
  
    return null;  
  }  
}
```



# DEMO



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# LAB



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**