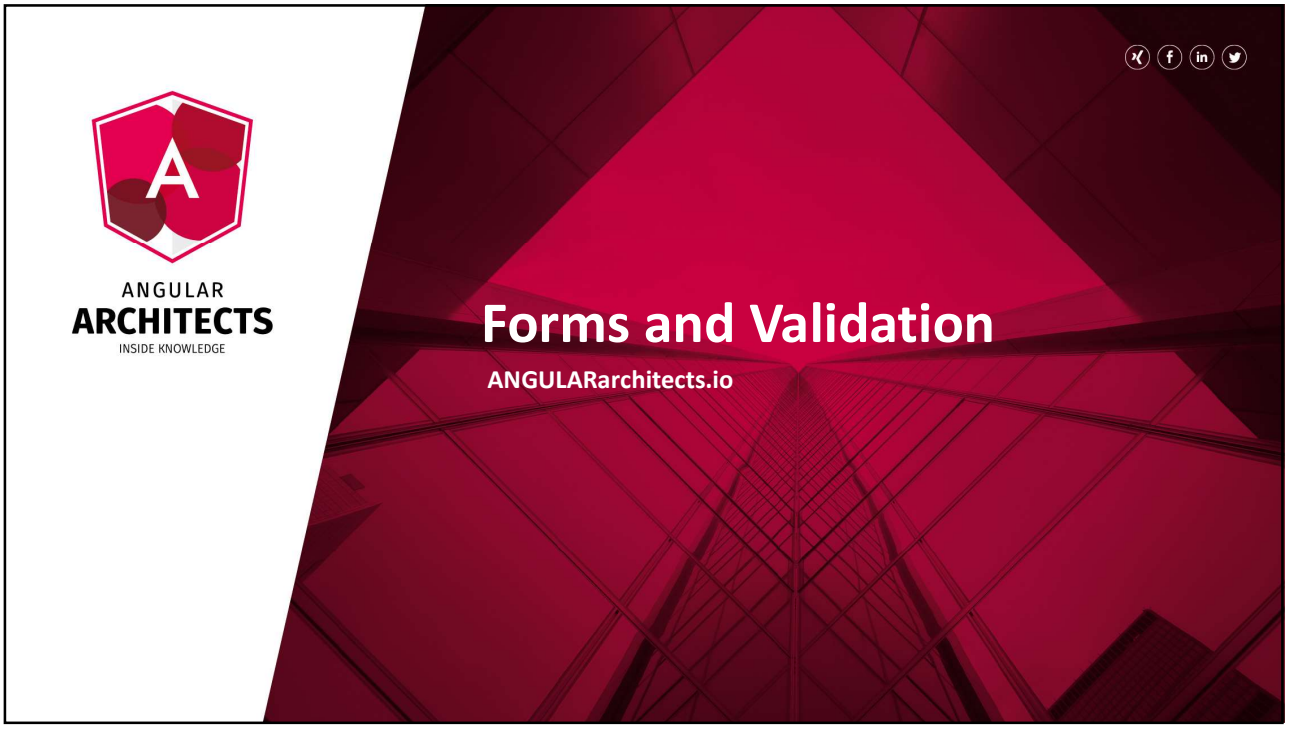


Contents

- Angular Forms Support
- Template-driven Forms
- Reactive Forms
- Validation

Angular Forms Support

Template-driven	• ngModel within the Template • Angular creates the object graph for forms • FormsModule
Reactive	• Application creates the object graph for forms • More control • ReactiveFormsModule
Data-driven	• Angular generates form for a data model • Community projects

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

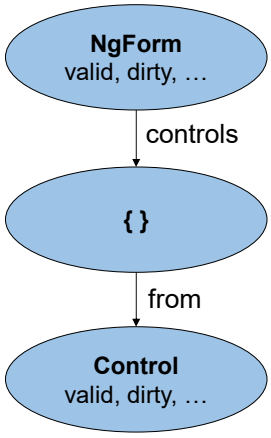
Template-driven forms

Template-driven Forms

```
export class FlightSearchComponent {

   from: string;
   to: string;

   constructor(private flightService: FlightService) {

         from = 'Graz';
         to = 'Hamburg';
   }
}
```

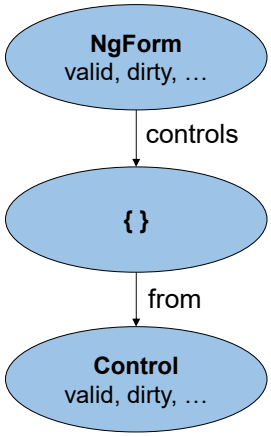
View

```
<form>

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

   […]

</form>
```



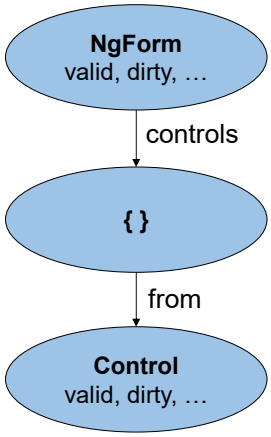
View

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

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

   […]

</form>
```



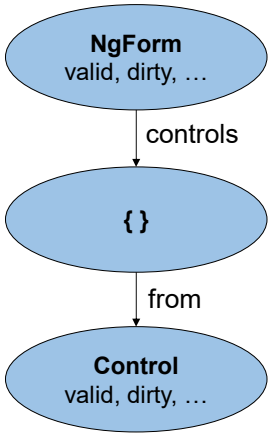
View

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

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

   <div *ngIf="f.controls['from'].invalid">
      …Error…
   </div>

</form>
```



View

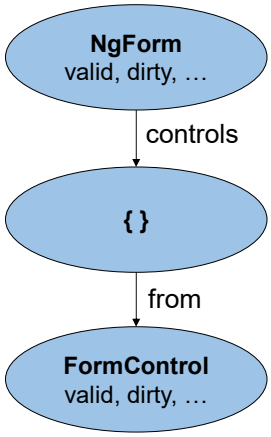
```
<form #f="ngForm">

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

   <div *ngIf="f?.controls['from']?.invalid">
      …Error…
   </div>

</form>
```



ANGULAR ARCHITECTS

View

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

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

   <div *ngIf="f?.controls['from']?.invalid">
      …Error…
   </div>

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



ANGULAR ARCHITECTS

DEMO

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Custom Validators

Directives

- Directives add behavior to your application
- Examples:
  - ngModel
  - ngClass
  - ngIf
  - ngFor
- In contrast to Components they have no Template

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 };
        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')

Use Attributes

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

Use 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 {
        […]
    }
}
```

Use 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 {
        […]
    }
}
```

Use Attributes

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

DEMO

Multi-Field-Validators

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

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

Multi-Field-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;

        […]
}
```

Multi-Field-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 === to.value) return { roundTrip: true };

        return null;
    }
}
```

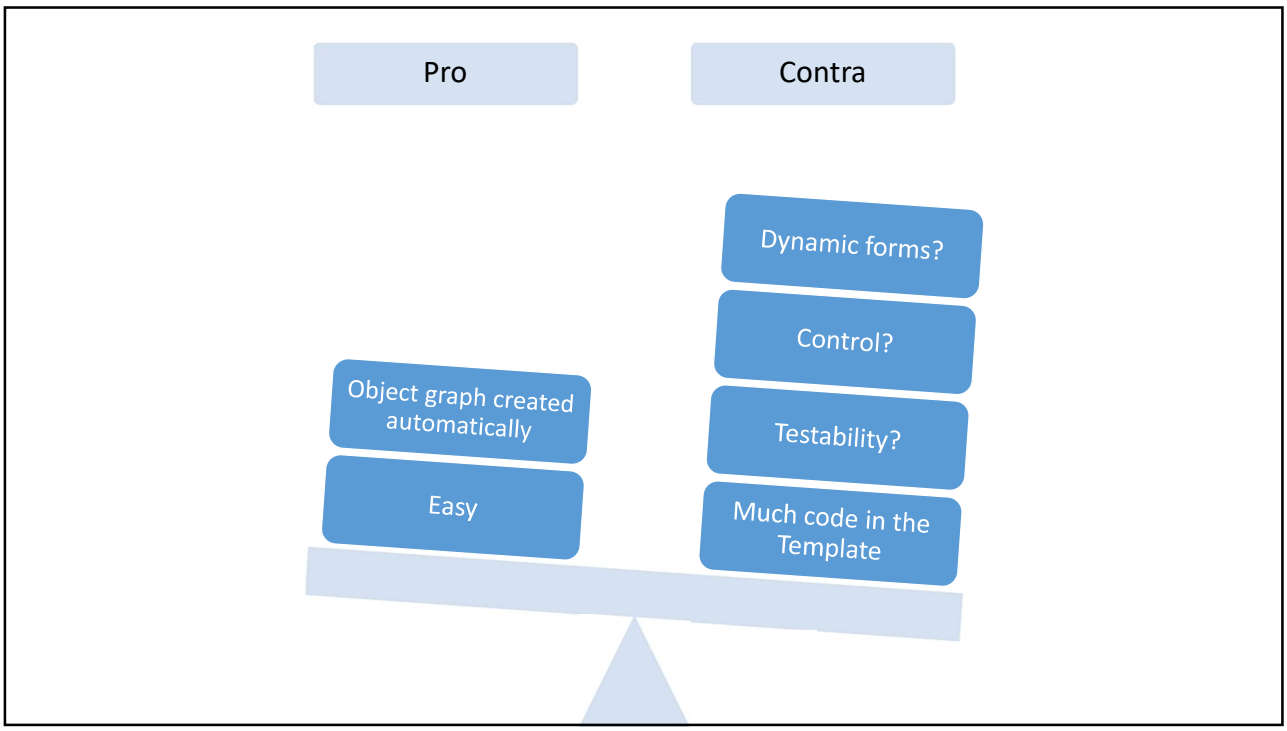
Asynchronous Validation Directives

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

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

}
```

Asynchronous Validation Directives

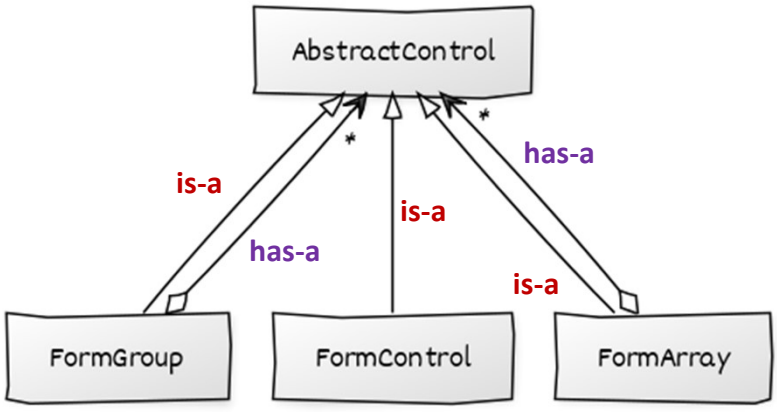
Token: NG_ASYNC_VALIDATORS

DEMO


Pro
Contra
Dynamic forms?
Control?
Object graph created automatically
Testability?
Easy
Much code in the Template

Reactive Forms

Classes for Object Graph



ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

ReactiveFormsModule

```
@NgModule({
  imports: [
    ReactiveFormsModule,
    CommonModule,
    SharedModule,
    [...]
  ],
  [...]
})
export class FlightBookingModule { }
```

Reactive Forms

```
export class FlightSearchComponent {

  form: FormGroup;

  […]
}
```

Reactive Forms

```
export class FlugSuchenComponent {

   form: FormGroup;

   constructor(…) {
       const fromControl = new FormControl('Graz');
       const toControl = new FormControl('Hamburg');
       this.form = new FormGroup({ from: fromControl, to: toControl});

       […]

   }
}
```

Reactive Forms

```
export class FlightSearchComponent {

   form: FormGroup;

   constructor(…) {
       const fromControl = new FormControl('Graz');
       const toControl = new FormControl('Hamburg');
       this.form = new FormGroup({ from: fromControl, to: toControl});

       fromControl.validator = Validators.required;
       […]
   }
}
```

Reactive Forms

```
export class FlightSearchComponent {

  form: FormGroup;

  constructor(…) {
    const fromControl = new FormControl('Graz');
    const toControl = new FormControl('Hamburg');
    this.form = new FormGroup({ from: fromControl, to: toControl});

    fromControl.validator =
            Validators.compose([Validators.required, Validators.minLength(3)]);
  }
}
```

Reactive Forms

```
export class FlightSearchComponent {

  form: FormGroup;

  constructor(…) {

    fromControl.validator =
            Validators.compose([Validators.required, Validators.minLength(3)]);

    fromControl.asyncValidator =
            Validators.composeAsync([…]);
  }
}
```

FormBuilder

```
export class FlightSearchComponent {

   form: FormGroup;

   constructor(fb: FormBuilder, …) {
      this.form = fb.group({
         from: ['Graz', Validators.required],
         to: ['Hamburg', Validators.required]
      });
      […]
   }

}
```

FormBuilder

```
export class FlightSearchComponent {

   form: FormGroup;

   constructor(fb: FormBuilder, …) {
      this.form = fb.group({
         from: ['Graz', [Validators.required, Validators.minLength(3)] ],
         to: ['Hamburg', Validators.required]
      });
      […]
   }

}
```

FormBuilder

```
export class FlightSearchComponent {

   form: FormGroup;

   constructor(fb: FormBuilder, …) {
      this.form = fb.group({
         from: ['Graz', [Validators.required, Validators.minLength(3)], [ /* asyncValidator */ ] ],
         to: ['Hamburg', Validators.required]
      });
      […]
   }

}
```

API

```
this.form.valueChanges.subscribe(change => {
    console.debug('form changed', change);
});
```

```
this.form.controls['from'].valueChanges.subscribe(change => {
    console.debug('from control changed', change);
});
```

```
const fromValue = this.form.controls['from'].value;
const toValue = this.form.controls['to'].value;
```

```
const formValue = this.form.value;
```

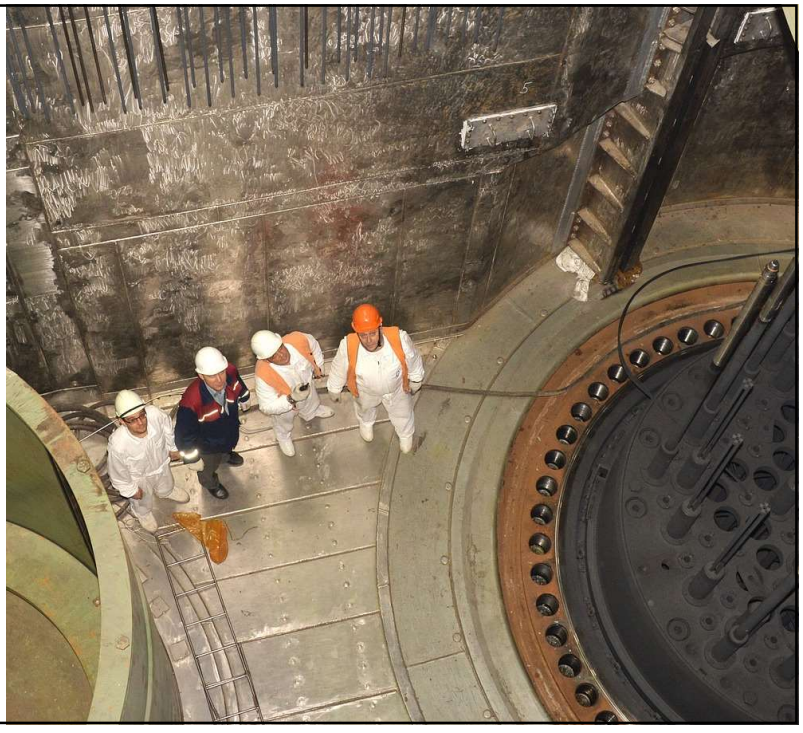
Reactive Forms

```
<form [formGroup]="form">

   <input id="from" formControlName="from" type="text">

   […]

</form>
```

Reactive Forms

```
<form [formGroup]="form">

   <input id="from" formControlName="from" type="text">
   <div *ngIf="form.controls['from'].invalid">…Error…</div>

   […]

</form>
```

DEMO



Validators for Reactive Forms

Reactive Validators == Functions

Simple Validator

```
function validate (c: AbstractControl): ValidationErrors | null {
    if (c.value == 'Graz' || c.value == 'Hamburg') {
        return null;
    }
    return { city: true };
}
```

Use Validators

```
this.form = fb.group({
    from: [
        'Graz',
        [
            validate
        ],
        [
            /* asyncValidator */
        ]
    ],
    to: ['Hamburg', Validators.required]
});
```

Parameterized Validators

```
function validateWithParams(allowedCities: string[]): ValidatorFn {

        […]

}
```

Parameterized Validators

```
function validateWithParams(allowedCities: string[]): ValidatorFn {

    return (c: AbstractControl): ValidationErrors | null => {
            […]
    };

}
```

Parameterized Validators

```
function validateWithParams(allowedCities: string[]): ValidatorFn {

    return (c: AbstractControl): ValidationErrors | null => {

        if (allowedCities.indexOf(c.value) > -1) {
            return null
        }

        return { city: true };

    };
}
```

Use Validators

```
this.form = fb.group({
    from: [
        'Graz',
        [
            validateWithParams(['Graz', 'Hamburg'])
        ],
        [
            /* asyncValidator */
        ]
    ],
    to: ['Hamburg', Validators.required]
});
```

DEMO

Asynchronous Validators

```
export function cityValidatorAsync(flightService): AsyncValidatorFn {

    return (control: AbstractControl) => {
       […]
       return observable;
    }
}
```

Use Validators

```
this.form = fb.group({
    from: [
        'Graz',
        [
            validateWithParams(['Graz', 'Hamburg'])
        ],
        [
            cityValidatorAsync(this.flightService)
        ]
    ],
    to: ['Hamburg', Validators.required]
});
```

Multi-Field-Validators

```
export function validateMultiField([…]): ValidatorFn {

    return (control: AbstractControl) => {

        const formGroup = control as FormGroup;

        […]
    }

};
```

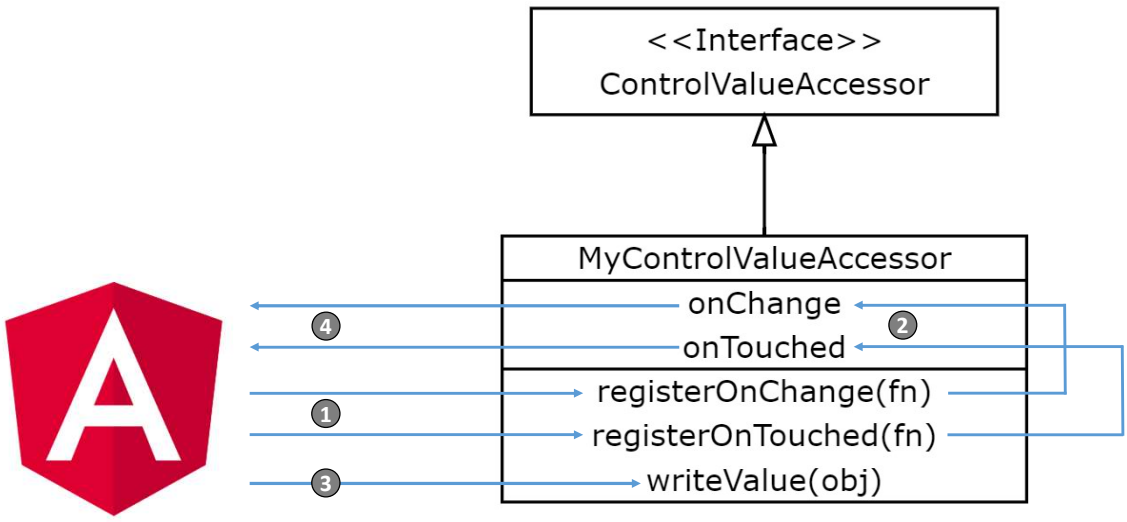
Use Validators

```
this.form = fb.group({ … });
this.form.validator = validators.compose([validateMultiField([…])])
```

DEMO

Custom Form Controls with

Control Value Accessors



ANGULAR ARCHITECTS

Case Study #3: Formatting/Parsing Dates

22.3.2021

2021-03-22T14:18:59.232Z

```
<input [(ngModel)]="date" appDate name="date">
```

ANGULAR ARCHITECTS

DEMO

Case Study: DateControl


```
<app-date [(ngModel)]="date"></app-date>
```

LAB

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE