



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Forms

ANGULARarchitects.io

Contents

- Overview
 - Approaches
 - Template-driven Forms
- Reactive Forms
- Validation

Approaches

Template-driven

- ngModel
- Angular creates object graph
- FormsModule

Reactive

- App creates object graph
- More control
- ReactiveFormsModule

Data-driven

- Angular generates form for data model
- Self-made and community solutions

Template-driven Forms

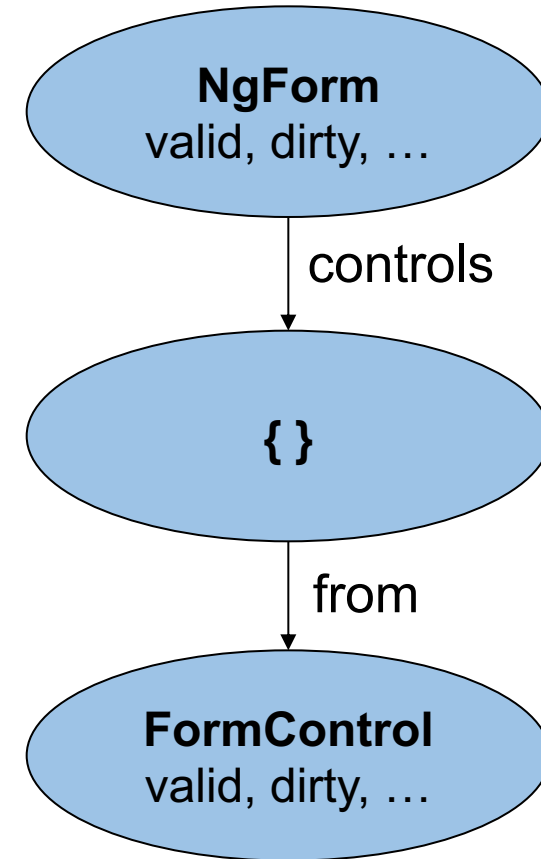


Template-driven Forms

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

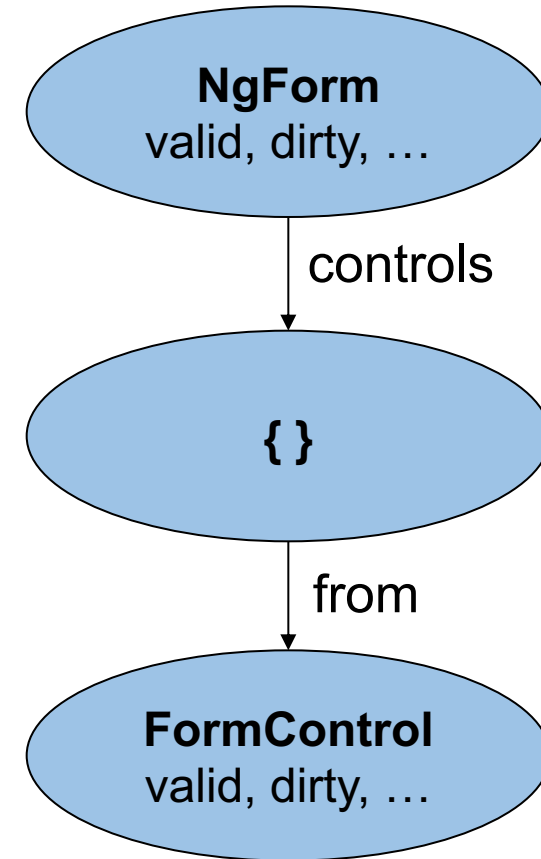
View

```
<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>
```



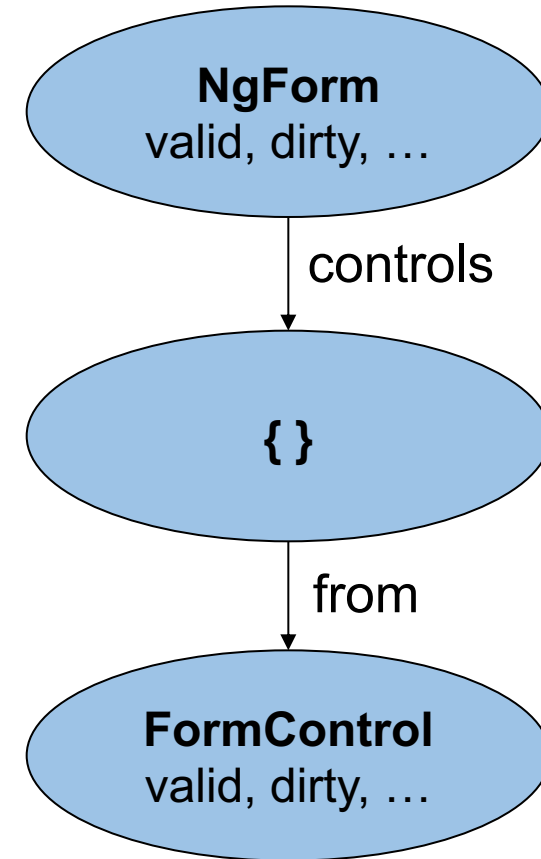
View

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

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

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

</form>
```



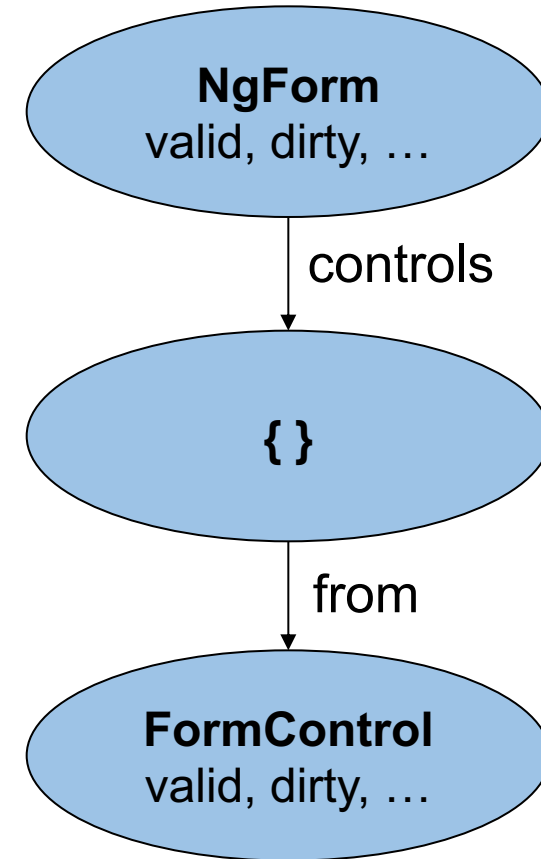
View

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

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

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

</form>
```



View

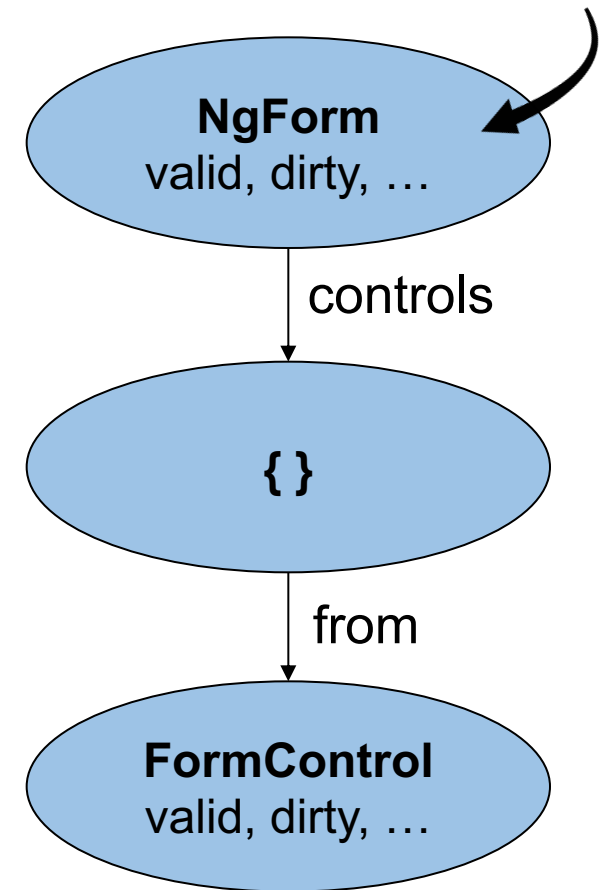
```
<form #f="ngForm">

  <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>
```

Wraps FormGroup ~1:1



DEMO

Pro

Contra

Angular creates
object graph

Simple

Dynamic Forms?

Control?

Testability?

Lots of code in view

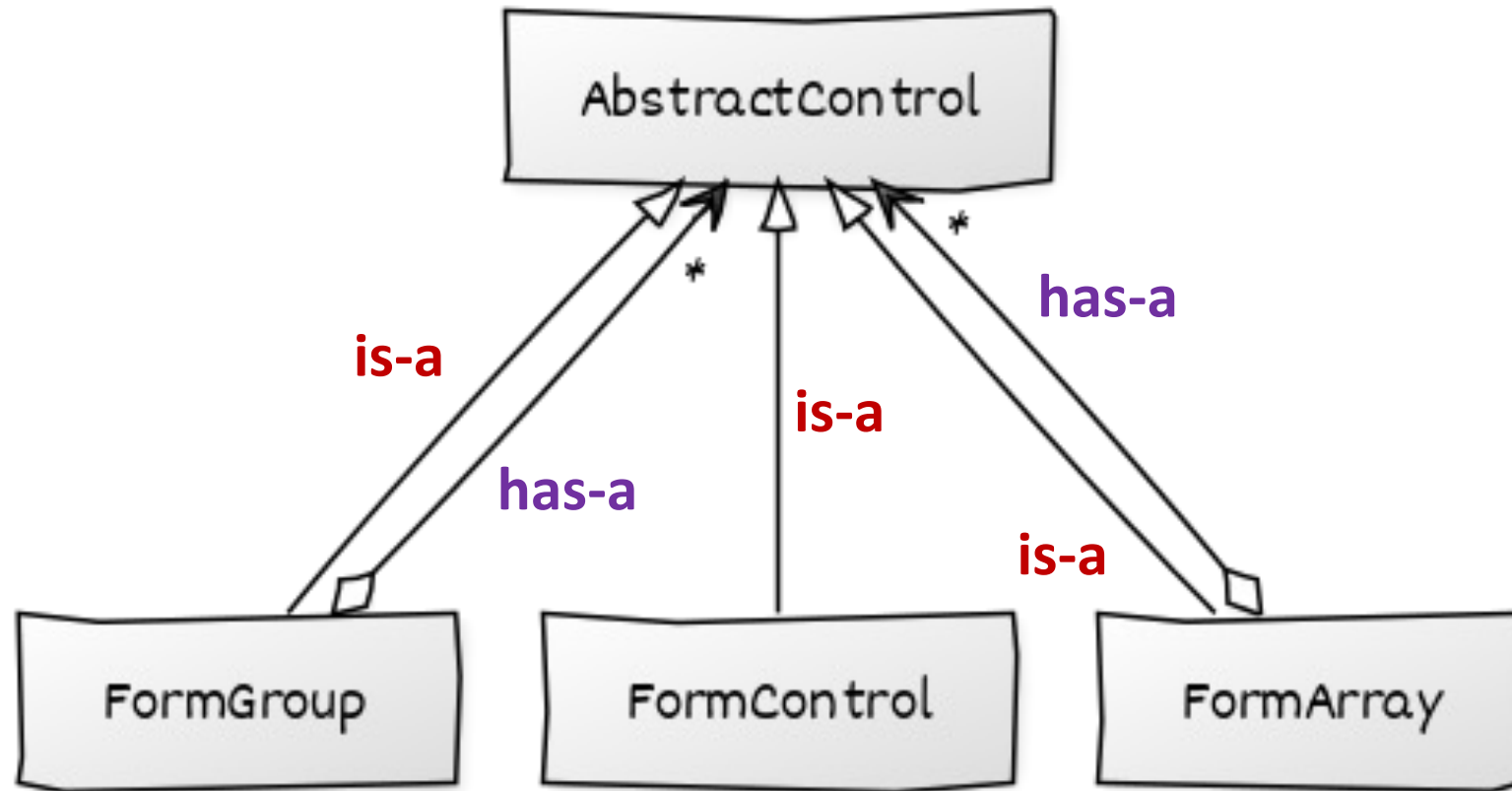


@ManfredSteyer

Reactive Forms



Classes for Object Graph



ReactiveFormsModule

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

Reactive Forms

```
export class FlightSearchComponent {  
  form: FormGroup;  
  [...]  
}
```


Reactive Forms

```
export class FlightSearchComponent {  
  
  form: FormGroup;  
  
  constructor(...) {  
    let fromControl = new FormControl('Graz');  
    let toControl = new FormControl('Hamburg');  
    this.form = new FormGroup({ from: fromControl, to: toControl});  
  
    [...]  
  
  }  
}
```

Reactive Forms

```
export class FlightSearchComponent {  
  
  form: FormGroup;  
  
  constructor(...) {  
    let fromControl = new FormControl('Graz');  
    let toControl = new FormControl('Hamburg');  
    this.form = new FormGroup({ from: fromControl, to: toControl});  
  
    fromControl.validator = Validators.required;  
    [...]  
  }  
}
```

Reactive Forms

```
export class FlightSearchComponent {  
  
  form: FormGroup;  
  
  constructor(...) {  
    let fromControl = new FormControl('Graz');  
    let 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(...) {  
    let fromControl = new FormControl('Graz');  
    let toControl = new FormControl('Hamburg');  
    this.form = new FormGroup({ from: fromControl, to: toControl});  
  
    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('formular hat sich geändert', change);  
});
```

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

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

```
let 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'].valid">...Error...</div>
```

```
  [...]
```

```
</form>
```



DEMO

Reactive Validation



Reactive Validators == Functions



@ManfredSteyer

A First Simple Validator

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



Using Validators

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

Validators with Parameters

```
function validateWithParams(allowedCities: string[]) {  
    [...]  
}
```


Validators with Parameters

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

Validators with Parameters

```
function validateWithParams(allowedCities: string[]): ValidatorFn {  
    return (c: AbstractControl): ValidationErrors => {  
        [...]  
    };  
}
```



Validators with Parameters

```
function validateWithParams(allowedCities: string[]): ValidatorFn {  
    return (c: AbstractControl): ValidationErrors => {  
        if (allowedCities.indexOf(c.value) > -1) {  
            return { }  
        }  
        return { city: true };  
    };  
}
```



Using Validators

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



DEMO

Async Validators

```
export function cityValidatorAsync(flightService: FlightService) {  
    return (control: AbstractControl): Observable<ValidationErrors> => {  
        [...]  
        return observable;  
    }  
}
```



Using Async Validators

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

Multifield Validators

```
export function validateMultiField(...): ValidationFn {  
    return (control: AbstractControl): ValidationErrors {  
        const formGroup = control as FormGroup;  
        [...]  
    }  
};
```



Using Validators

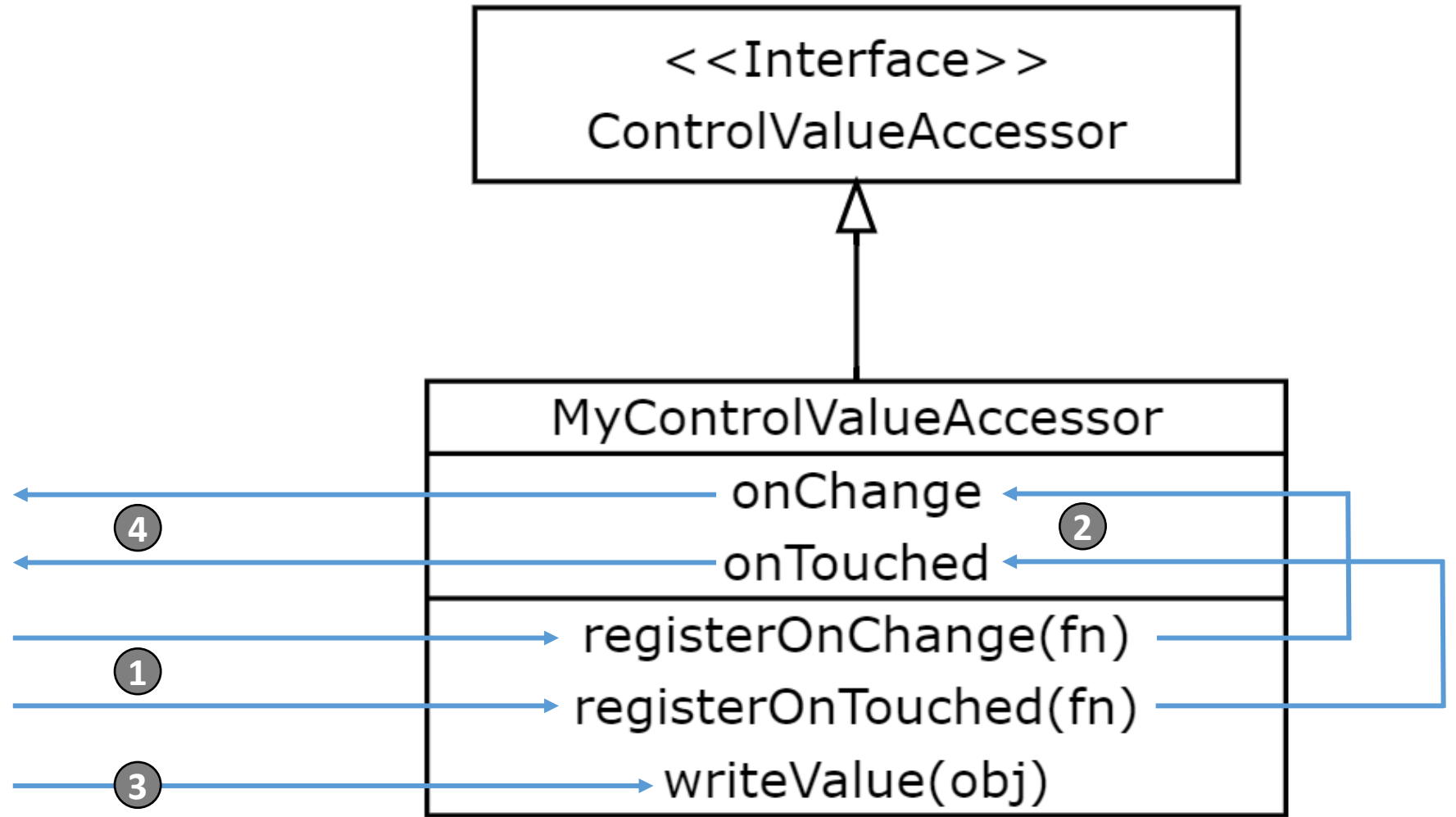
```
this.form = fb.group({ ... });  
this.form.validator = validators.compose([validateMultiField([...])])
```

DEMO

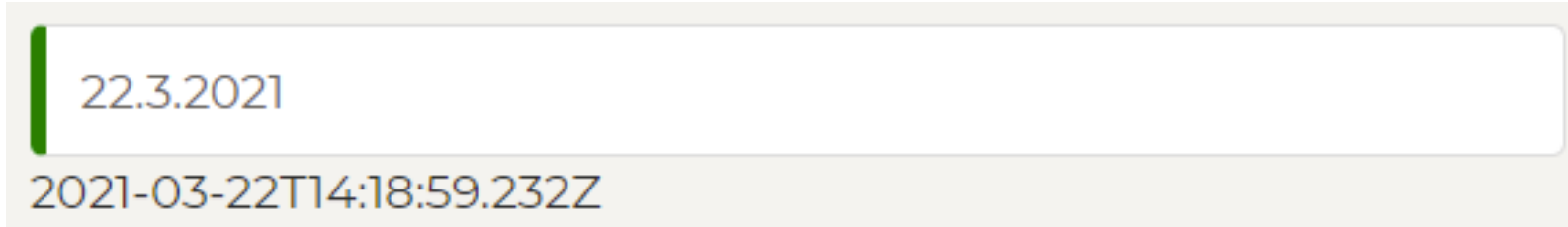
LAB

Custom Form Controls with Control Value Accessors





Case Study #3: Formatting/Parsing Dates



22.3.2021

2021-03-22T14:18:59.232Z

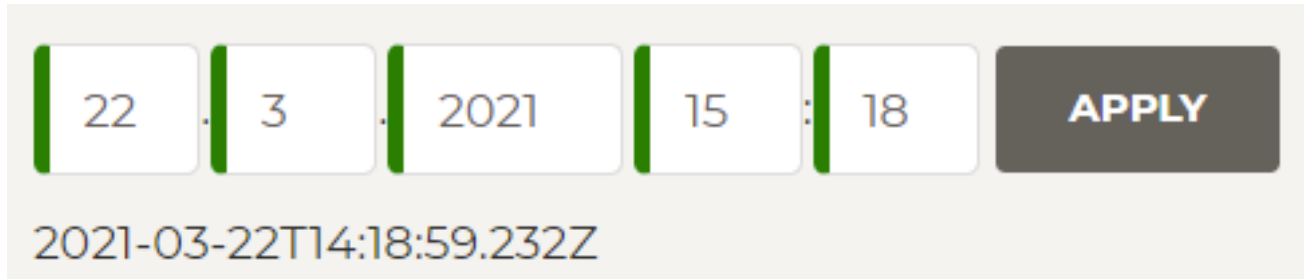
```
<input [(ngModel)]="date" appDate name="date">
```

DEMO



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Case Study: DateControl

A screenshot of a date control UI component. It features five input fields for date and time: day (22), month (3), year (2021), hour (15), and minute (18). Each field has a green vertical bar on its left side. To the right of these fields is a dark grey button with the text 'APPLY' in white. Below the input fields, the ISO 8601 date string '2021-03-22T14:18:59.232Z' is displayed in a smaller, grey font.

22 . 3 . 2021 15 : 18 **APPLY**

2021-03-22T14:18:59.232Z

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


LAB



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE