



ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Reactive Forms and Validation

Alex Thalhammer

Pro

Contra

Auto generated
object tree

Simple to use

Dynamic Forms?

Control?

Testing?

Lot of code in
HTML-template



Reactive Forms



ReactiveFormsModule

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



Reactive Forms

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



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});  
  
    [...]  
  
  }  
}
```



Reactive Forms

```
export class FlightSearchComponent {  
  
  form: FormGroup;  
  
  constructor(...) {  
    const fromControl = new UntypedFormControl('Graz');  
    const toControl = new UntypedFormControl('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 value has changed', change);  
});
```

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

```
let fromValue = this.form.controls.from.value;  
const toValue = this.form.controls.to.value;
```

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



Reactive Forms

```
<form [formGroup]="form">  
  <input formControlName="from">  
  [...]  
</form>
```



Reactive Forms

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

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

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

```
  [...]
```

```
</form>
```



DEMO



LAB



Validators for Reactive Forms



Reactive Validators === functions

A simple validator

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



Apply validators

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



Parametrizable validators

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



Parametrizable validators

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



Parametrizable 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 (ctrl: AbstractControl): Observable<ValidationErrors | null> => {  
        [...]  
        return observable;  
    }  
}
```



Use 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 | null {  
        const formGroup = control as FormGroup;  
  
        [...]  
    }  
};
```



Use multifielid validators

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

DEMO



LAB

