Angular Reactive Forms mit Bootstrap

- Controls per Code erzeugen und mit Template synchronisieren
  - Besserer Zugriff auf Verhalten und Struktur der Controls
  - Testbarkeit
- Manches ist nur per Reactive Form implementierbar
  - Asynchrone Validierung
  - Forms mit Arrays von Controls

Infrastruktur

pupil-reactive-form.component.html | app.module.ts ●

```
1  import { NewCourseFormComponent } from './new-course-form/new-course-form.component';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { NgModule } from '@angular/core';
4  import { FormsModule, ReactiveFormsModule } from '@angular/forms'
5
6
7  import { AppComponent } from './app.component';
8  import { PupilFormComponent } from './pupil-form/pupil-form.component';
9  import { FormsTestComponent } from './forms-test/forms-test.component';
10 import { ChangePasswordComponent } from './change-password/change-password.component';
11
12
13 @NgModule({
14   declarations: [
15     AppComponent,
16     ChangePasswordComponent,
17     PupilFormComponent,
18     NewCourseFormComponent,
19     FormsTestComponent
20   ],
21   imports: [
22     BrowserModule,
23     FormsModule,
24     ReactiveFormsModule
25   ],
26   providers: [],
27   bootstrap: [AppComponent]
28 })
29 export class AppModule { }
```

Wurzelelement FormGroup erstellen

- Constructor
  - Objekt controls ➔ Baum der Controls
  - Validaton optional, synchron oder asynchron

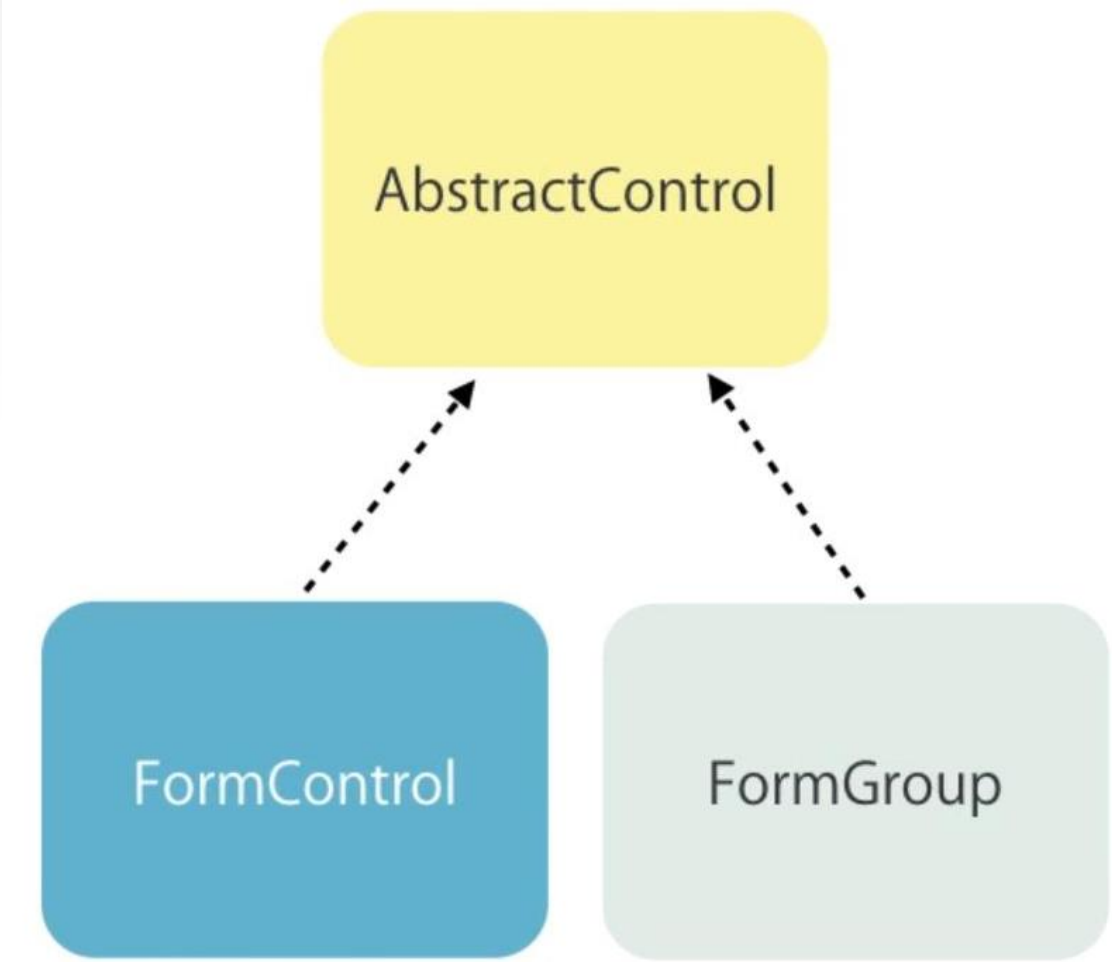
```
import { FormGroup } from '@angular/forms';

@Component({
  selector: 'app-pupil-
  templateUrl: './pupil
  styleUrls: ['./pupil-
})
export class PupilReact

  form = new FormGroup()
```

```
FormGroup(controls: { [key: string]: AbstractControl; }
,
validatorOrOpts?: ValidatorFn | ValidatorFn[] |
AbstractControlOptions
,
asyncValidator?: AsyncValidatorFn | AsyncValidatorFn[]
): FormGroup
```

FormGroup, FormControl is-a AbstractControl



Component mit UI-Controls

pupil-reactive-form.component.ts

```
1   import { Component, OnInit } from '@angular/core';
2   import { FormGroup, FormControl } from '@angular/forms';
3
4   @Component({
5     selector: 'app-pupil-reactive-form',
6     templateUrl: './pupil-reactive-form.component.html',
7     styleUrls: ['./pupil-reactive-form.component.css']
8   })
9   export class PupilReactiveFormComponent {
10    form = new FormGroup(
11      {
12        name : new FormControl(),
13        isSubscribed : new FormControl(),
14        contactMethod : new FormControl()
15      }
16    )
17    contactMethods = [ ···
20    ];
```

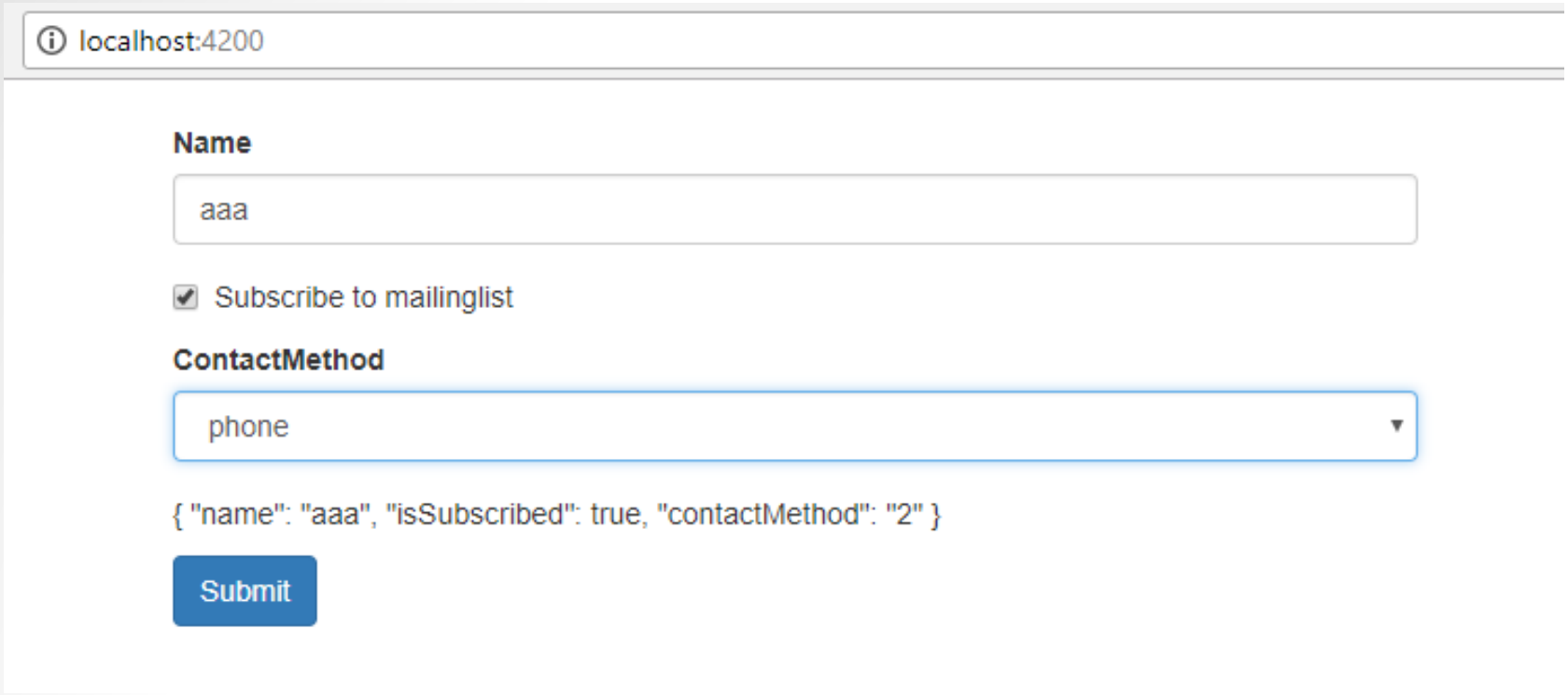
In Html-Template verbinden

pupil-reactive-form.component.html

```
1   <div class="col-sm-4 col-sm-push-1">
2     <form [formGroup]=form>
3       <div class="form-group">
4         <label for="name">Name</label>
5         <input
6           formControlName="name"
7           name="name"
8           id="name"
9           type="text"
10          class="form-control"/>
11      </div>
12      <div class="checkbox">···
20      </div>
21      <div class="form-group">···
34      </div>
35      <p>{{ form.value | json }}</p>
36      <button ···
40    </form>
41  </div>
```


HTL LEONDING

Formular funktioniert intern



HTL LEONDING

Validation hinzufügen

- Constructor von FormControl
  - Initialer Status
  - Validierungsfunktionen optional, synchron/asynchron

```
@Component({
  selector: 'app-pupil-reacti
  templateUrl: './pupil-react
  styleUrls: ['./pupil-reacti
})
export class PupilReactiveFor
  form = new FormGroup(
    {
      name : new FormControl()
```

```
FormControl(formState?: any,
validatorOrOpts?: ValidatorFn | ValidatorFn[] |
AbstractControlOptions
,
asyncValidator?: AsyncValidatorFn | AsyncValidatorFn[]
): FormControl
```

Vorsicht: fehlerhafte Imports

pupil-reactive-form.component.ts ●

```
1   import { Component, OnInit } from '@angular/core';
2   import { FormGroup, FormControl } from '@angular/forms';
3   import { Validators } from '@angular/forms/src/validators';
4
```

pupil-reactive-form.component.ts ✖

```
1   import { Component, OnInit } from '@angular/core';
2   import { FormGroup, FormControl, Validators } from '@angular/forms';
```

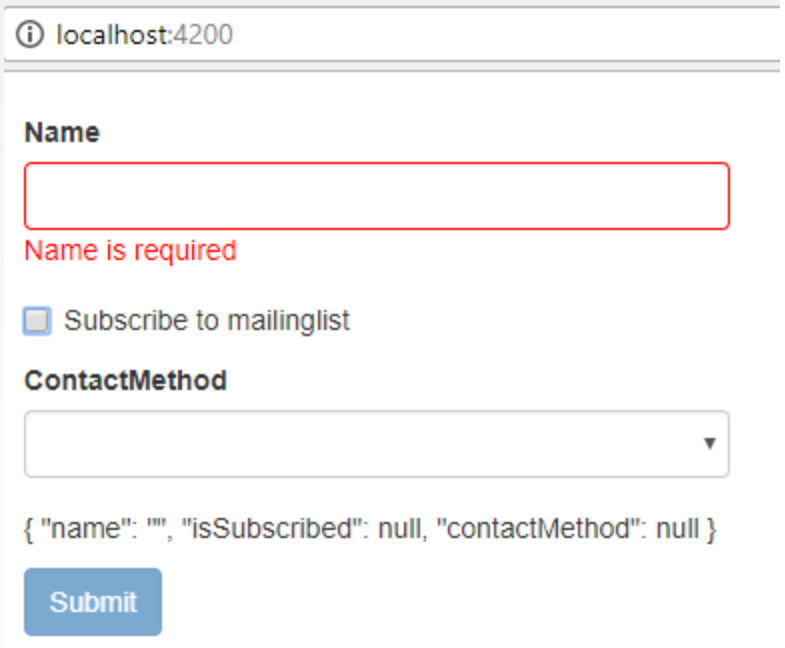
Validation im Template

- Nicht über ngModel sondern über Formular
  - Form.get() liefert das Control
  - Redundant ➔ Kapseln in Field/Property

```
<div class="form-group">
  <label for="name">Name</label>
  <input
      formControlName="name"
      id="name"
      type="text"
      class="form-control"/>
    <span
      class="invalid"
      *ngIf="form.get('name').touched && !form.get('name').valid">
        Name is required
    </span>
</div>
```

Control in Property kapseln

```
get name(){
  return this.form.get('name');
}
```




```
<div class="form-group">
  <label for="name">Name</label>
  <input
      formControlName="name"
      id="name"
      type="text"
      class="form-control"/>
    <span
      class="invalid"
      *ngIf="name.touched && name.invalid">
        Name is required
    </span>
</div>
```


HTL LEONDING

CustomValidation – Name ohne Leerzeichen

- Angular.io ➔ Interfacebeschreibung
  - Parameter: AbstractControl
  - Result: ValidationErrors oder null

ValidatorFn INTERFACE

npm Package	@angular/forms
Module	`import { ValidatorFn } from '@angular/forms';`
Source	forms/src/directives/validators.ts

Interface Overview

```
interface ValidatorFn {
  (c: AbstractControl): ValidationErrors | null
}
```

Validator Function[s]

- static
- Liefert Errorobjekt (Value wird hier nicht benötigt)

pupil-reactive-form.component.ts | name.validators.ts ● | pupil-reactive-form.component.html

```ts
import { Validators, ValidationErrors, AbstractControl } from '@angular/forms';

export class NameValidators implements Validators{

    static couldNotContainSpace(control : AbstractControl){
        if((control.value as string).indexOf(' ') >= 0){
            return { couldNotContainSpace : true};
        }
        return null;
    }
}
```

Asynchrone Validierung

- Motivation
  - UI soll bedienbar bleiben, auch wenn Validierung etwas länger dauert
- Beispiel
  - Name darf nicht Max sein
  - Prüfung dauert 2 Sekunden (Timer)
  - Liefert ein Promise zurück, das später eingelöst wird
    - LambdaExpression mit zwei Callbacks (resolve, reject) die nach Zeitablauf Ergebnis liefern

Html-Template

- shouldBeUnique feuert, wenn Name ‚Max' ist

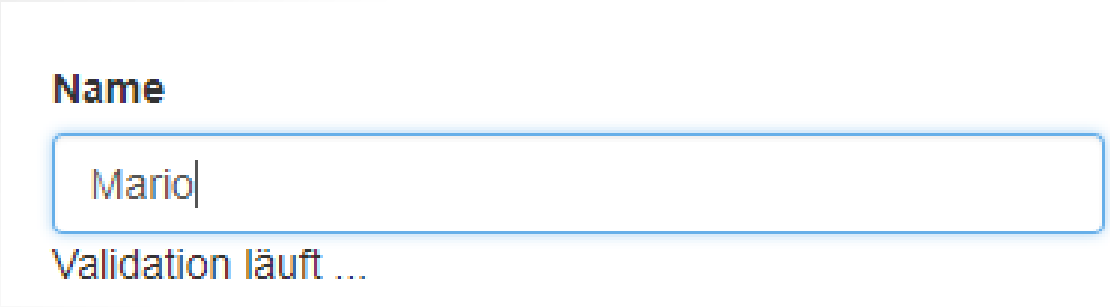
```
<div class="form-group">
  <label for="name">Name</label>
  <input
      formControlName="name"
      id="name"
      type="text"
      class="form-control"/>
    <span class="invalid" *ngIf="name.touched && name.invalid">
      <span *ngIf="name.hasError('required')">Name is required</span>
      <span *ngIf="name.hasError('couldNotContainSpace')">Name could not contain spaces!</span>
      <span *ngIf="name.hasError('shouldBeUnique')">Name should be unique!</span>
    </span>
</div>
```

Validator

- setTimeout() führt LambdaExpression in vorgegebener Zeit aus
- Promise übergibt zwei CallbackFunktionen
  - resolve() liefert Ergebnis
  - reject() verwirft Promise

```
static shouldBeUnique(control : AbstractControl): Promise<ValidationErrors | null> {
    return new Promise( (resolve, reject) => {
        setTimeout( () => {
            if(control.value === 'Max'){
                resolve({ shouldBeUnique : true});
            }
            else{
                resolve(null);
            }
        }, 2000);
    });
}
```

Laufende Validierung signalisieren




```
<div class="form-group">
  <label for="name">Name</label>
  <input
      formControlName="name"
      id="name"
      type="text"
      class="form-control"/>
    <span *ngIf="name.pending">Validation läuft ...</span>
```


HTL LEONDING

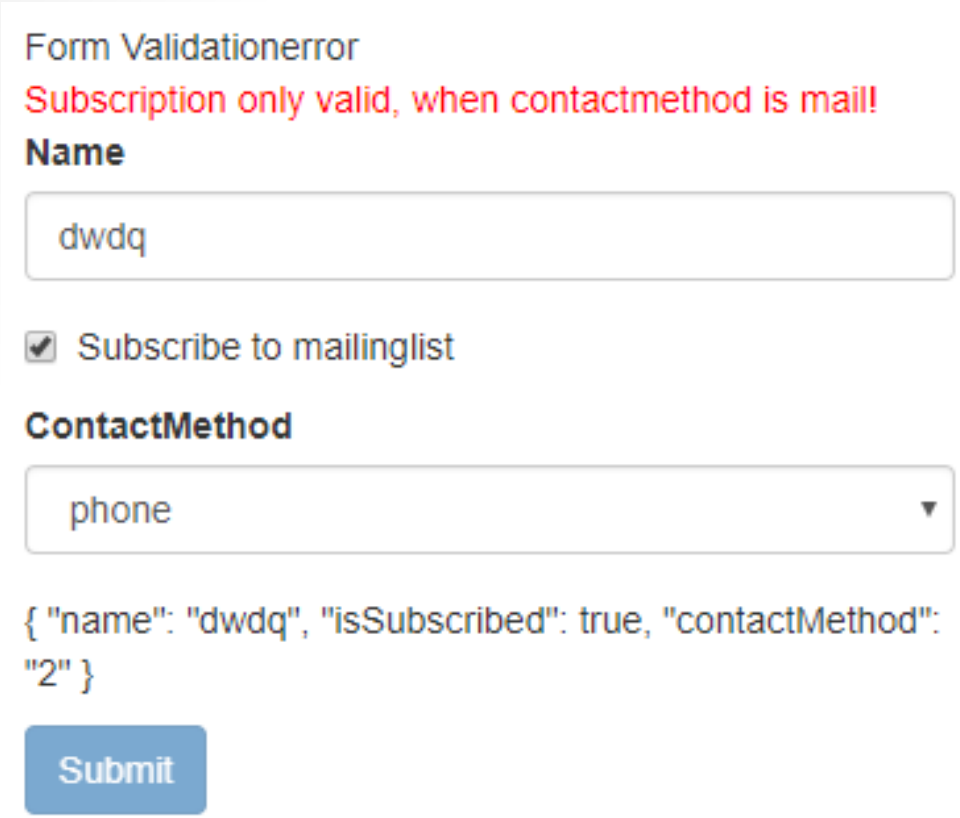
Gesamte Form validieren

- Bei submit() auslösen

```
onSubmit(){
  console.log(this.form.value);
  if(this.form.get('isSubscribed').value
     && this.form.get('contactMethod').value !== "1"){
    console.log('Formerror is set!')
    this.form.setErrors({ invalidSubscriptionAndContactMethod : true});
  }
}
```

```
<div *ngIf="form.hasError('invalidSubscriptionAndContactMethod')">
  <span>Form Validationerror</span>
  <br/>
  <span class="invalid" >
    Subscription only valid, when contactmethod is mail!
  </span>
</div>
```

Ergebnis



HTL LEONDING

Direkte Verbindung mit Model-Object

```
ngOnInit(): void {
  this.heroForm = new FormGroup({
    'name': new FormControl(this.hero.name, [
      Validators.required,
      Validators.minLength(4),
      forbiddenNameValidator(/bob/i) // <-- Here's how you pass in the custom validator.
    ]),
    'alterEgo': new FormControl(this.hero.alterEgo),
    'power': new FormControl(this.hero.power, Validators.required)
  });

}

get name() { return this.heroForm.get('name'); }

get power() { return this.heroForm.get('power'); }
```