ANGULAR 6

Peter Gurský, peter.gursky@upjs.sk

Vlastný asynchrónny validátor

- vytvoríme si validátor na konflikt s menom alebo emailom pri registrácii
- použijeme na serveri endpoint /user-conflicts
  - vracia pole s názvami konfliktných premenných, možné hodnoty sú "email" a "name", alebo vráti prázdne pole ak konflikt nie je

Vlastné validátory – asynchrónne

- validateHttp(schemaPath.niečo, {
  request: input => {url: '…'},
  onSuccess: (result, context) => null alebo {kind, message},
  onError: (error) => null alebo {kind, message}
  }
- validateAsync(schemaPath.niečo, {
  params: input => dáta pre factory napr. na základe input.value(),
  factory: (params) => rxResource<návratový typ, typ vstupu>({
  params: () => vracia vstup pre stream na základe params()
  stream:({params}) => vracia observable<návratový typ>
  }),
  onSuccess: (result, context) => null alebo {kind, message},
  onError: (error) => null alebo {kind, message}
  }

Vymazanie používateľa

app/users/users-service.ts

```
deleteUser(id: number):Observable<boolean> {
  return this.http.delete(this.url + 'user/' + id + '/'+ this.token).pipe(
    map(() => {
      this.msgService.successMessage("User deleted successfully");
      return true;
    }),
    catchError(error => this.processHttpError(error))
  );
}
```

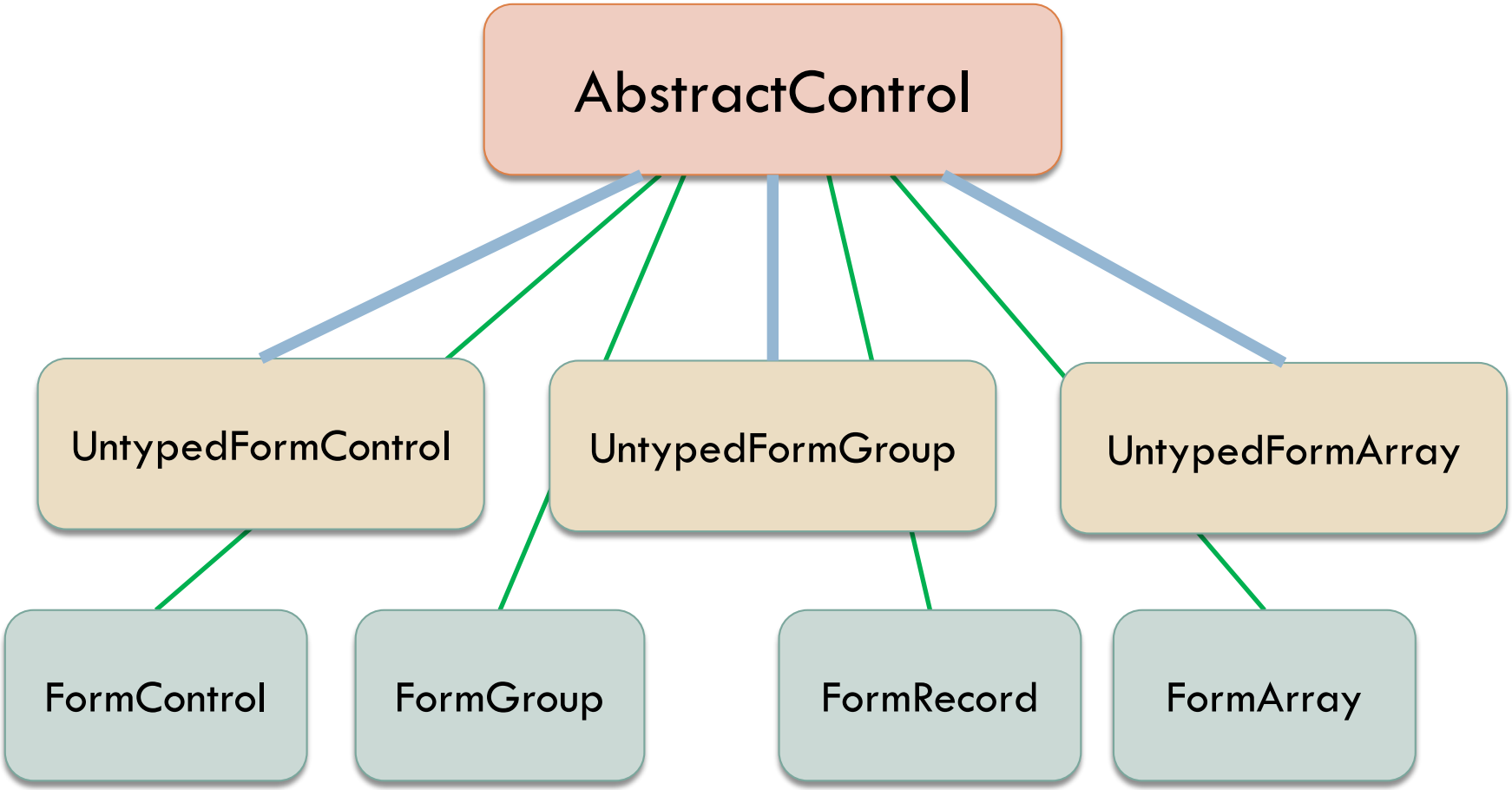
Dialóg na potvrdenie zmazania

- https://material.angular.io/components/dialog/api
- dialóg je obyčajný komponent, ktorý si nechá v konštruktore injektnúť MatDialogRefa prípadné dodatočné dáta
  - constructor(private dialogRef: MatDialogRef<MyDialogComponent>, @Inject(MAT_DIALOG_DATA) private data: any) { }
  - dialogRef má metódu close(myResult), ktorým vieme odovzdať voľbu používateľa cez Observable kódu, ktorý dialog vyvolal
- vyvolanie dialógu:
  - necháme si injektkúť dialog: MatDialog v konštruktore
  - zavoláme const dialogRef = this.dialog.open(MyDialogComponent, {data: "delete?"})
  - počkáme si na odpoveď: dialogRef.afterClosed().subscribe(result => { … })

Reaktívne formuláre

- ak ich chceme používať
  - importujeme ReactiveFormsModule
- model formulára a jeho komponentov sú vytvárané ako (inštančné) premenné v kóde komponentu
  - formulár alebo skupina komponentov je typu FormGroup
  - jeden komponent (napr. <input>) je typu FromControl
- hodnota komponentu formulára nie je prepojená s inštančnou premennou, ako v TD-formulároch
  - nepoužívame [(ngModel)]="username“ pre inštančnú premennú username='Jano',
  - hodnota komponentov sa dá z kódu meniť len cez funkcie setValue(), patchValue() – viď neskôr

Hierarchia reaktívnych komonentov



Prepojenie šablóny a reaktívneho modelu

- samostatné elementy – bez nadradeného <form> elementu:
  - v šablóne: <input type="text" [formControl]="name">
  - v kóde: name = new FormControl<string|null>('Jano');
- pre skupinu elementov:

```
<form [formGroup]="profileForm">
  <input type="text“ formControlName="firstName">
  <input type="text" formControlName="lastName">
</form>
```

```
profileForm = new FormGroup({
    firstName: new FormControl(''),
    lastName: new FormControl(''),
  });
```

```
constructor(private fb: FormBuilder) { }
profileForm = this.fb.group({
    firstName: [''], lastName: ['']
});
```

Prepojenie šablóny a reaktívneho modelu

úvodné hodnoty

- samostatné eleme
  elementu:
  - v šablóne: <input type="text" [formControl]="name">
  - v kóde: name = new FormControl<string | null>('Jano');
- pre skupinu elementov:

```
<form [formGroup]="profileForm">
  <input type="text“ formControlName="firstName">
  <input type="text" formControlName="lastName">
</form>
```

```
profileForm = new FormGroup({
    firstName: new FormControl(''),
    lastName: new FormControl(''),
  });
```

```
constructor(private fb: FormBuilder) { }
profileForm = this.fb.group({
    firstName: [''], lastName: ['']
});
```

Prepojenie šablóny a reaktívneho modelu

- samostatné elementy – bez nadradeného <form> elementu:
  - v šablóne: <input type="text" [formControl]="name">
  - v kóde: name = new FormControl<string|null>('Jano');

Typ možných hodnôt uvádzame v zobákoch

Typ písať nemusíme, ak je odvoditeľný z úvodnej hodnoty (okrem booleanu – ten píšeme vždy)

Odvodí sa typ <string | null>

tov:

```
">
Name="firstName">
Name="lastName">
```

```
profileForm = new FormGroup({
  firstName: new FormControl(''),
  lastName: new FormControl(''),
});
```

```
constructor(private fb: FormBuilder) { }
profileForm = this.fb.group({
  firstName: [''], lastName: ['']
});
```

Prepojenie šablóny a reaktívneho modelu

Typ možných hodnôt uvádzame v zobákoch

Ak nechceme null hodnoty vo FormControl<string | null>, musíme uviesť náš zámer cez druhý parameter konštruktora:

firstName: new FormControl<string>('', {nonNullable: true})

alebo použiť fb:FormBuilder: firstName = this.fb.nonNullable.control('');

alebo použiť nnfb:NonNullableFormBuilder: firstName = this.nnfb.control('');

Ak potom zavoláme firstName.reset(), nenastaví hodnotu na null, ale na iniciálnu hodnotu.

```
profileForm = new FormGroup({
  firstName: new FormControl(''),
  lastName: new FormControl(''),
});
```

```
constructor(private fb: FormBuilder) { }
profileForm = this.fb.group({
  firstName: [''], lastName: ['']
});
```

Prepojenie šablóny a reaktívneho modelu

- pre pole elementov:

```
<form [formGroup]="profileForm">
  …
  <div formArrayName="aliases">
  @for(alias of aliases.controls; track i; let i=$index) {
    <input type="text" [formControlName]="i">
  }
  </div>
</form>
```

```
profileForm = new FormGroup({
    …
    aliases: new FormArray([
      new FormControl('Johny'),
      new FormControl('Janči')
    ]),
});
```

```
constructor(private fb: FormBuilder) { }
profileForm = this.fb.group({ …,
    this.fb.array([
        this.fb.control(''),
        this.fb.control('')
    ])
});
```

Prepojenie šablóny a reaktívneho modelu

- pre pole elementov:

```
<form [formGroup]="profileForm">
  …
  <div formArrayName="aliases">
  @for(alias of aliases.
    <input type="text"
  }
  </div>
</form>
```

```
profileForm = new Forr
    …
    aliases: new FormAr
      new FormControl('J
      new FormControl('J
    ]),
});
```

Pridanie nového aliasu:

```
<button (click)="addAlias()">Add Alias</button>
```

```
get aliases() {
  return this.profileForm.get('aliases') as FormArray;
}

addAlias() {
   this.aliases.push(this.fb.control(''));
}
```

Hodnoty komponentov r. formulárov

- získanie hodnoty samostatného elementu
  - surname = new FormControl('Pekný');
  - v šablóne: {{surname.value}}
  - let currentName = this.surname.value;
  - this.surname.valueChanges.subscribe(value => x = value);
- získanie hodnoty vnoreného elementu

```
profileForm = new FormGroup({
    person = new FormGroup({
      name: new FormControl(''),
      surame: new FormControl('')
    })
  });
```

  - let currentFirstName = this.profileForm.get('person.name').value
  - v šablóne: {{name.value}}
    - ak máme getter: get name() { return this.profileForm.get('name'); }

Zmena hodnoty r. formulárov

- hodnoty sú immutable
  - nastavíme všetky hodnoty formulára cez setValue()
  - nastavíme iba niektoré hodnoty cez patchValue()
    - platí pre ľubovoľnú úroveň

```
profileForm = new FormGroup({
  firstName: new FormControl(''),
  lastName: new FormControl(''),
  address: new FormGroup({
    street: new FormControl(''),
    city: new FormControl(''),
    state: new FormControl(''),
    zip: new FormControl('')
  })
});
```

```
updateProfile() {
  this.profileForm.patchValue({
    firstName: 'Nancy',
    address: {
      street: '123 Drew Street'
    }
  });
}
```

Validátory pre r. formuláre

- validátory dodávame ako druhý parameter konštruktora, asynchrónne validátory ako tretí
  - name = new FormControl(user.name, Validators.required);
  - email = new FormControl(user.email,
    [Validators.required, Validators.email], MyAsyncValidator);
- alternatívne ako objekt cez druhý parameter konštruktora, napr.:
  - myForm = new FormGroup({ 'name': new FormControl(),
    'email': new FormControl()},
    { validators: myFormValidator,
      asyncValidators: [FirstValidator, SecondValidator]
    });

Výsledok validity vo formulári

```
<input id="name" class="form-control" [formControl]="name" required >
 @if (name.invalid && (name.dirty || name.touched)) {
        @if (name.errors['required']) {
          <div>Name is required</div>
        }
        @if (name.errors['minlength']) {
          <div>Name must be at least 4 characters long.</div>
         }
 }
```

required nie je povinné zadať, ale je to odporúčané

```
name = new
FormControl(this.user.name, [
      Validators.required,
      Validators.minLength(4)]);
```

ďalšie vlastnosti viď: https://angular.io/api/forms/AbstractControl