ANGULAR – 9. ČASŤ

Peter Gurský, peter.gursky@upjs.sk

Komponent na registráciu používateľa

- Čo si ukážeme
  - Reaktívne formuláre
    - Model pre formulárový element
    - Modely pre skupiny formulárových elementov
  - Validátory
    - Vstavané
    - Vlastné
      - Synchrónne
      - Asynchrónne

Validátory

- Angular má niekoľko vlastných validátorov na kontrolu formulárov, alebo môžeme dorobiť vlastné
- Neparametrické validátory sú funkcie (ValidatorFn), ktoré majú
  - na vstupe kontrolovaný formulárový element, skupinu formulárových elementov alebo celý formulár
  - na výstupe
    - null ak je validátor úspešný
    - objekt s nájdenými chybami, napr. {'length': 'small string', 'format': 'wrong character', }
- Parametrické validátory sú funkcie druhého rádu
  - na vstupe majú jeden alebo viac hodnôt na ich nakonfigurovanie (napr. Validator.minLength(5))
  - na výstupe má ValidatorFn

Vstavané validátory

- https://angular.io/api/forms/Validators
  - parametrické
    - min, max, minLength, maxLength, pattern, compose, composeAsync
  - neparametrické
    - required, requiredTrue, email, nullValidator

Template-driven formuláre

- založené na
  - [(ngModel)] na mapovanie hodnôt
  - povinný parameter name
- model formulára, ktorý vyhodnocuje validitu, je dostupný **len v šablóne** cez premennú elementu
  - ngForm v elemente form
  - ngModel vo vstupných formulárových komponentoch
- model formulára nie je dostupný z kódu komponentu
- ak chceme použiť vo formulári validátory, vieme ich dodať formulárovým elementom iba cez direktívy, t.j. atribúty elementov
  - https://angular.io/guide/form-validation#adding-to-template-driven-forms

Vloženie validátora pre TD-formuláre

- direktívy pre zabudované validátory:
  - https://angular.io/api/forms#directives
    - CheckboxRequiredValidator, PatternValidator,,...
- Napr. EmailValidator je direktíva definovaná pravidlami:
  - [email][formControlName] ← pre Reakt. formuláre
  - [email][formControl] ← pre Reakt. formuláre
  - [email][ngModel] ← pre TD-formuláre
- takže ho aktivujeme tak, že napíšeme v šablóne
  - <input type="email" name="email" **ngModel email**>
  - <input type="email" name="email" [(**ngModel**)]="user.email" **email**>

Použitie validácie

- validita sa dá zobraziť používateľovi pomocou css, ak je prítomný validátor nastaví sa pre daný formulárový element jedna z tried
  - ng-touched / ng-untouched – element bol navštívený / nebol
  - ng-dirty / **ng-pristine** – hodnota je zmenená / nie je
  - **ng-valid** / **ng-invalid** - hodnota je správna / nie je
    - ak element má atribút required, hodnota musí byť vyplnená
    - okrem required máme aj: minlength, maxlength, pattern
    - …zložitejšie kontroly viď. Validator v ngModel-i
- Pozrime si cez inšpektora

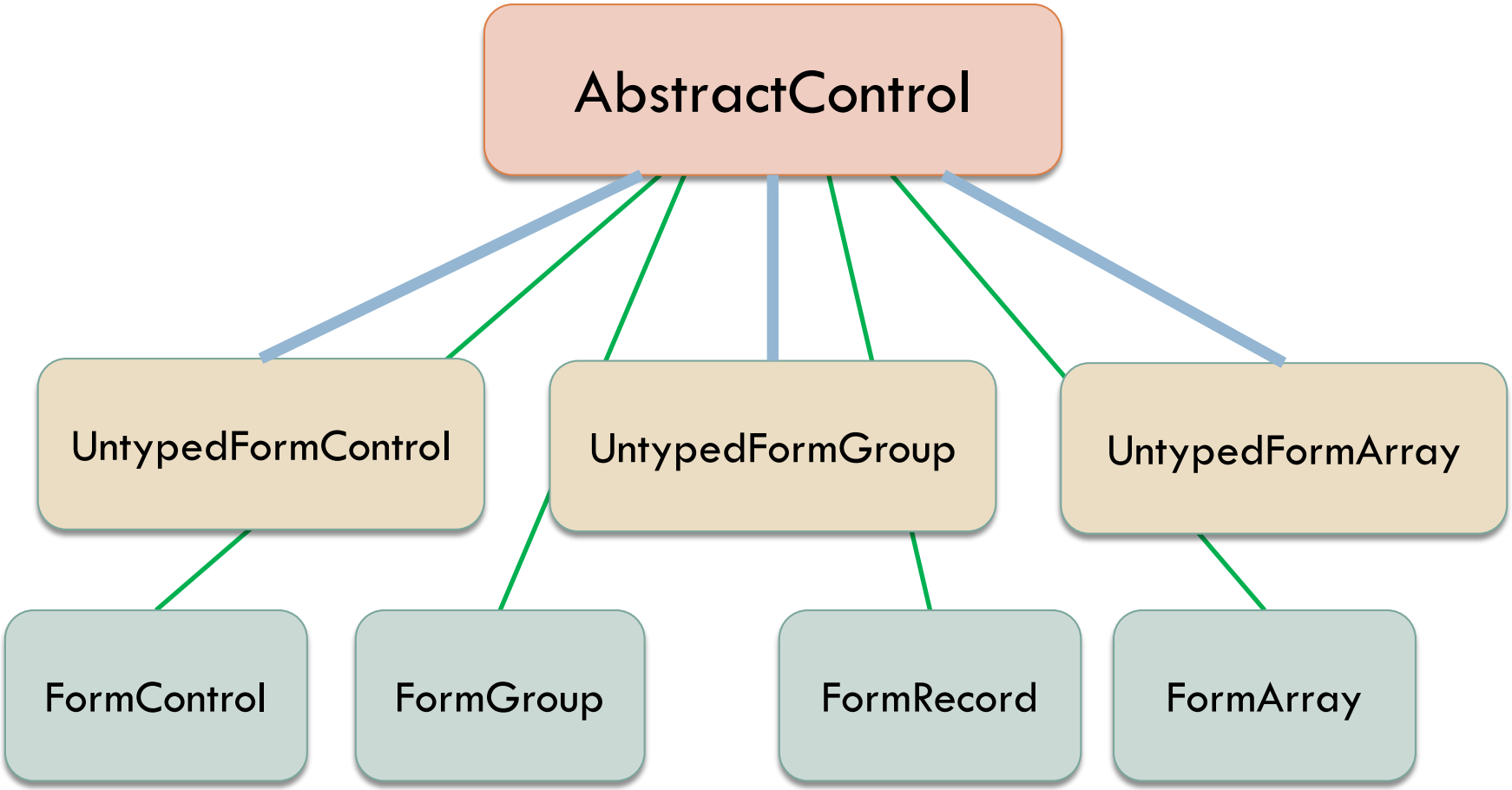
Validita cez Angular Material

- nevalidný input je označený červenou farbou
- mat-form-field môže mať v sebe okrem input elementu aj
  - <mat-hint> - zobrazenie textu pod input elementom
  - <mat-label> - pomenovanie vstupu
    - ak je input prázdny zobrazuje sa namiesto hodnoty
    - ak je input vyplnený zobrazuje sa nad hodnotou
  - <mat-error> - zobrazenie červenej chybovaj hlášky pod input elementom
    - použijeme *nglf, aby sa chyba zobrazila, iba ak je input nevalidný
    - ak sa zobrazuje mat-error, nezobrazuje sa mat-hint

Reaktívne formuláre

- ak ich chceme používať
  - importujeme ReactiveFormsModule do @NgModule
- 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

Vlastný validátor pre formulárový komponent

- vytvoríme si validátor na kvalitu hesla cez knižnizu zxcvbn
  - npm install @zxcvbn-ts/core @zxcvbn-ts/language-common @zxcvbn-ts/language-en

```
passwordValidator(): ValidatorFn {
  return (control: AbstractControl): ValidationErrors => {
    const test = zxcvbn(control.value);
    const message = 'Password strength: '
         + test.score + ' / 4 - must be 3 or 4';
    return test.score < 3 ? { weakPassword: message } : null;
  };
}
```

Vlastný validátor pre formulár

- vytvoríme si validátor na zhodu hesla a kontrolného hesla

```
passwordsMatchValidator(control: FormGroup): ValidationErrors {
    const password = control.get('password');
    const password2 = control.get('password2');
    if (password.value === password2.value) {
      password2.setErrors(null);
      return null;
    } else {
      password2.setErrors({ differentPasswords: 'Passwords do not
 match' });
      return { differentPasswords: 'Passwords do not match' };
    }
  }
```

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