Angular 7

Forms

> scalac

.mo.

Let’s create a new app

- ng new forms-demo --style=scss
- ng add @angular/material
  - update material theme
- add person-editor-td and person-editor-md components

Person model

```
export interface IPerson {
  firstName: string;
  lastName: string;
  phone?: string;
  dateOfBirth: Date;
}
```

Template-driven forms

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

```
<input [(ngModel)]="person.firstName" name="firstName">
```

[] + () = two-way binding

Reactive forms (Model-driven)

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


<form [formGroup]="personForm">
  <input formControlName="firstName">
</form>


this.personForm = new FormGroup({
  firstName: new FormControl(person.firstName)
});
```

using FormBuilder is more compact

Reactive forms custom two-way binding

```
@Input() person: IPerson;
@Output() personChange = new EventEmitter();
this.personForm.valueChanges.subscribe(value => {
  this.personChange.emit(value);
});
```

```
<app-person-editor-md [(person)]="person"></app-person-editor-md>
```

Submit

```
<form (ngSubmit)="onSubmit()">
  <button type=”submit”>save</button>
</form>
```

Reset

```
<form [formGroup]="personForm">      <!-- reactive forms -->
<form #personForm="ngForm">          <!-- template-driven -->
  <button type="reset" (click)="personForm.reset()">Reset</button>
</form>
```

Forms CSS classes

.ng-valid

.ng-invalid

.ng-pending

.ng-pristine

.ng-dirty

.ng-untouched

.ng-touched

Built-in validators

required

maxlength

minlength

pattern

requiredTrue

Validation messages

validation errors hashtable for each control:

- null - everything OK!
- {

  minlength: {requiredLength: 3, actualLength: 2}

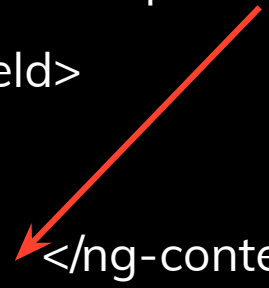
  pattern: {requiredPattern: "^[a-z]$", actualValue: "11"}

  }

Validation messages - better approach

let’s write component shared/app-form-field which will display validation messages from its child control

```
<app-form-field [errors]="getErrors('lastName')">
  <input type="text" placeholder="Last name" formControlName="lastName">
</app-form-field>
```



<ng-content> </ng-content> - slot for child control in component’s template

Custom validators

phone number validation

npm i libphonenumber-js --save

```
const validatePhone = (control: AbstractControl): ValidationErrors | null => { ... }
this.personForm = this.fb.group({
  phone: [phone, [validatePhone]]
})
```

Asynchronous validators

Angular docs: Async Validation

```
this.personForm = this.fb.group({
  login: [login, [Validators.required], isLoginUnique]
})
```

template-driven

import { FormsModule }

works directly on model

2-way binding by design

logic in HTML template

less code

reactive form (model-driven)

import { ReactiveFormsModule }

creates own new model

1-way binding, (2-way can be added)

logic in JS code

can handle really complex cases

easier to unit tests

Pipes - formatting values

Built-in pipes

ng g c person/person-card

date of birth: {{person.dateOfBirth | date:short}}

date of birth: May 10, 2019

Custom pipes

ng g pipe shared/full-name

```
@Pipe({
  name: 'fullName'
})
export class FullNamePipe implements PipeTransform {
  transform(value: any, args?: any): string { ... }
}
```

Homework

- It wasn't everything about forms! read: Angular forms docs
- add IPerson.email field, use builn-in EmailValidator, update forms and card
- add IPerson.iban field, and write validator and pipe using fast-iban library, sample IBANs you can find here, update forms and card
- add signup component with fields: email (as login), password, password confirmation, mandantory checkbox, validate email (required, valid email), password (required, min length: 6, [a-zA-Z0-9], check confirmation)
- write unit tests for signup component

Thank You