Lab 5A: Template driven forms

Een nieuwe component

Er zijn al enkele input velden in onze contact component voorzien, maar ze doen nog niet veel. Genereer een nieuw component via de command line:

```
ng generate component contact-form
```

Hierdoor wordt er een nieuwe folder gemaakt in /src/app met de naam contact-form. Hierin zitten volgende bestanden:

- Contact-form.component.ts (component code)
- Contact-form.component.html (component view)
- Contact-form.component.css (css styles)
- Contact-form.component.spec.ts (testing)

De nieuwe component wordt ook automatisch toegevoegd aan de app.module.ts file. Om de nieuwe component te gebruiken, wordt hij toegevoegd aan de app.component.html:

```
<app-contact *ngFor="let contact of contactList;"
             [contact]="contact">
</app-contact>
<app-contact-form></app-contact-form>
```

Een formulier bouwen

Kopieer al de input velden (inclusief labels) en de button van de contact component zijn HTML naar de nieuwe contact-form.component.html. Kopieer ook alle input, label en button gerelateerde css van contact.component.css naar contact-form.component.css.

Refactoring:

- Verwijder de formuliervelden, behalve de favorite checkbox & label, uit de contact component. De h2 en ul mogen blijven staan.
- Verwijder de functie die gekoppeld was aan de button van de contact component zijn code.
- Verwijder de EventEmitter property van de contact component.
- Verwijder de EventEmitter en Output van de imports in de contact component.

Voeg een extra input van het type tekst toe aan het formulier voor de avatar image. Sluit alle inputs & button in een <form> tag. Verander het type van de button naar “**submit**” en verwijder “**contact**” uit de ngModel bindings.

De nieuwe contact-form.component.html heeft volgende inhoud:

```
<form>
<input type="text" name="name"
      placeholder="Name" [(ngModel)]="name"><br/>
<input type="email" name="email"
      placeholder="name@domain.com" [(ngModel)]="email"><br/>
<input type="tel" name="phone"
      placeholder="Phone number" [(ngModel)]="phone"><br/>
<input type="text" name="avatar"
      placeholder="avatar url" [(ngModel)]="avatar">
<input type="checkbox" name="isFavorite"
      id="favorite" [(ngModel)]="isFavorite">
<label for="isFavorite">Favorite</label><br>

<button type="submit">Submit</button>
</form>
```

Momenteel is het een normaal HTML formulier met ngModel bindings. Voeg de template referentie variabele toe om een Angular form object te maken en koppel het aan de submit functie:

```
<form #form="ngForm" (ngSubmit)="submit(form)">
```

Vervolgens maken we de submit functie aan. Het formulier beslist niet zelf wat de functionaliteit is achter de submit functie. We gebruiken een eventEmitter om de parent de verdere afhandeling te laten doen. Op deze manier is het formulier later herbruikbaar en flexibel.

Voorzie de nodige imports:

```
import { Component, OnInit, Output, EventEmitter } from
'@angular/core';
import { Contact } from '../models/contact.model';
```

Maak een nieuwe eventEmitter aan van het type Contact:

```
 @Output() onSubmit: EventEmitter<Contact> = new EventEmitter();
```

Vervolgens maken we een submit functie waarin we de form values koppelen aan een nieuw Contact object. Dit object geven we door via de eventEmitter.

```
  submit(form): void {
    let contact: Contact = new Contact(
      form.value.name,
      form.value.email,
      form.value.phone,
      form.value.isFavorite,
      form.value.avatar
    );
    this.onSubmit.emit(contact);
  }
```

Validatie voorzien

Om validatie te voorzien, starten we met basis HTML validatie. Voeg aan alle input velden behalve avatar en de checkbox het attribuut **required** toe.

De velden zijn nu verplicht in te vullen, maar de gebruiker weet dit nog niet. We kunnen de state van een input veld opvragen via form.controls.name?.invalid:

```
<input type="text" name="name" placeholder="Name"
      [(ngModel)]="name" required>
<p *ngIf="form.controls.name?.invalid">Required</p>
```

Achter de control “name” is een vraagteken (?) toegevoegd. Als de view voor de eerste keer geladen wordt, bestaat “form.controls” nog niet. Zonder het vraagteken zou de view stuk gaan omdat je een property probeert aan te spreken die niet bestaat. **'?'** Toevoegen na properties waarvan je niet zeker bent of ze bestaan, lost dit probleem op.

Geef de naam een minimum lengte:

```
<input type="text" name="name" placeholder="Name"
      [(ngModel)]="name" minlength="3" required>
```

Gebruik een reguliere expressie voor de email:

```
<input type="email" name="email" placeholder="name@domain.com"
[(ngModel)]="mail" pattern="[a-zA-Z0-9_\.]+@[a-zA-Z0-9_\.]+"
required>
```

Vervolgens zorgen we dat er niet op de submit knop geklikt kan worden als het formulier invalid is:

```
<button type="submit" [disabled]="form.invalid">Submit</button>
```

Nu dat er meer validatie voorzien is, voorzien we ook visuele validatie berichten bij foute ingave.

```
<form #form="ngForm" (ngSubmit)="submit(form)">
<input type="text" name="name" placeholder="Name"
      [(ngModel)]="name" minlength="3" required>
<p *ngIf="form.controls.name?.errors?.minlength"
class="error">Minimum 3 characters.</p>
<p *ngIf="form.controls.name?.errors?.required"
class="error">Required.</p>
<input type="email" name="email" placeholder="name@domain.com"
      [(ngModel)]="email" pattern="[a-zA-Z0-9_\.]+@[a-zA-Z0-9_\.]+"
required>
<p *ngIf="form.controls.email?.errors?.pattern"
class="error">Invalid email.</p>
<p *ngIf="form.controls.email?.errors?.required"
class="error">Required.</p>
<input type="tel" name="phone" placeholder="Phone number"
      [(ngModel)]="phone" minlength="9" required>
<p *ngIf="form.controls.phone?.errors?.minlength"
class="error">Minimum 9 characters.</p>
<p *ngIf="form.controls.phone?.errors?.required"
class="error">Required.</p>
<input type="text" name="avatar"
      placeholder="avatar url" [(ngModel)]="avatar">
<input type="checkbox" name="isFavorite"
      id="favorite" [(ngModel)]="isFavorite">
<label for="isFavorite">Favorite</label><br>
```

```
<button type="submit" [disabled]="form.invalid">Submit</button>
</form>
```

Nu hebben we werkende HTML5 validatie, maar na het drukken op de submit knop, moet het formulier leeggemaakt worden. We callen hiervoor de reset functie in de submit handler:

```
submit(form): void {
    let contact: Contact = new Contact(
      form.value.name,
      form.value.email,
      form.value.phone,
      form.value.isFavorite,
      form.value.avatar
    );
    form.reset();
    this.onSubmit.emit(contact);
  }
```

Het enige wat er nu nog dient te gebeuren, is het opvangen van de data in de parent, app.component.ts.

Update de HTML van de app.component als volgt:

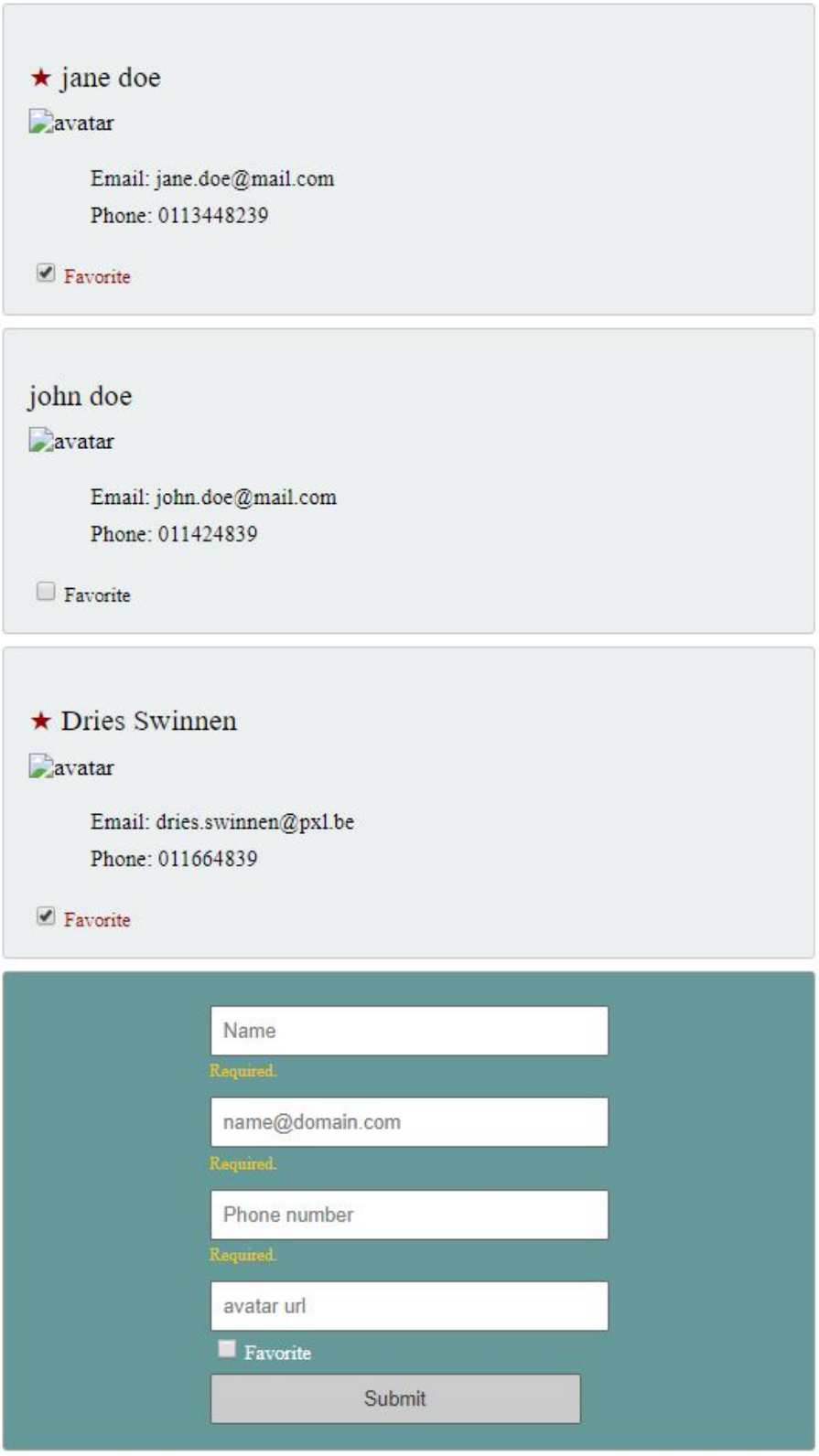
```
<app-contact-form
(onSubmit)="createContact($event)"></app-contact-form>
```

Voorzie de createContact methode in de app.component.ts file:

```
createContact(event: Contact) {
    this.contactList.push(event);
  }
```

Nu zou je via het formulier nieuwe contacten moeten kunnen toevoegen. De nieuwe contacten verschijnen automatisch in de lijst.

Tenslotte kopieer je uit de Resources folder op blackbord de inhoud van contact-form.component.css naar jouw contact-form.component.css. Het formulier zou er nu als volgt moeten uitzien:



Lab 5B: Model driven forms

Nota: Tijdens deze labo zal je de template driven implementatie overschrijven. Best maak je dus even een nieuwe commit aan in je GIT repo of zet je de add-contact component even aan de kant.

Het formulier uit de vorige lab bestond volledig uit html met bindings. Het is ook mogelijk om een formulier op te bouwen in de component. Op die manier wordt het formulier niet meer aangestuurd door de HTML en kunnen we meer functionaliteit voorzien (zoals bijvoorbeeld het toevoegen van custom validators).

De **FormsModule** import is nodig voor het gebruik van template driven forms en ngModel, maar model driven forms hebben extra functionaliteit nodig uit de **ReactiveFormsModule**. Voeg deze import toe aan je app.module.ts file:

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

imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule
  ],
```

Navigeer naar de contact-form component. Voeg volgende imports toe:

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

De structuur van het formulier blijft hetzelfde, maar wordt nu beheerd door de component. Om te starten wordt er een form group (groep van inputs) aangemaakt. Daarna koppelen we een serie van controls aan de formGroup.

Voorzie een nieuwe property in de contact-form component klasse:

```
form: FormGroup;
```

Daarnaast voorzie je onderstaande code in de ngOnInit methode:

```
ngOnInit() {
    this.form = new FormGroup({
      'name': new FormControl(null),
      'email': new FormControl(null),
      'phone': new FormControl(null),
```

```
      'isFavorite': new FormControl(false),
      'avatar': new FormControl(null)
    });
  }
```

Hierin wordt er een form control aangemaakt voor elk veld in ons formulier. Ieder veld krijgt automatisch een waarde als een object doorgegeven wordt of de waarde **null** (false bij de checkbox) als er geen value beschikbaar is.

Vervolgens moeten we het formulier aanpassen. Verwijder de **ngModel** bindings en **validatie** attributen. Voeg een formGroup attribuut toe aan de <form> tag. Dit zorgt voor de link naar de variabele uit onze klasse. Daarnaast voegen we een nieuw **formControlName** attribuut toe dat zorgt voor de link naar bovenstaande code in de ngOnInit methode.

```
<form [formGroup]="form" (ngSubmit)="submit(form)">
<input type="text" name="name" placeholder="Name"
formControlName="name">
<input type="email" name="email" placeholder="name@domain.com"
formControlName="email">
<input type="tel" name="phone" placeholder="Phone number"
formControlName="phone">
<input type="text" name="avatar" formControlName="avatar">
<input type="checkbox" name="isFavorite"
formControlName="isFavorite">
<label for="isFavorite">Favorite</label><br>

<button type="submit" [disabled]="form.invalid">Submit</button>
</form>
```

Het originele formulier had ook validators (required, minlength, pattern) die we momenteel nog niet hebben. Deze kunnen we bij voorzien bij elke formControl als volgt:

```
ngOnInit() {
    this.form = new FormGroup({
      'name': new FormControl(null, [Validators.required,
Validators.minLength(3)]),
      'email': new FormControl(null, [Validators.required,
Validators.pattern(/^[a-z0-9_\.]+@[a-z0-9_\.]+/i)]),
```

```
      'phone': new FormControl(null, [Validators.required,
Validators.minLength(9)]),
      'isFavorite': new FormControl(false),
      'avatar': new FormControl(null)
    });
  }
```

Validatie is nu actief op het formulier, maar de foutboodschappen staan nog in de oude format. Hier moeten de ngIf statements nog aangepast worden:

```
<p *ngIf="form.get('name').hasError('minlength')"
class="error">Minimum 3 characters.</p>
<p *ngIf="form.get('name').hasError('required')"
class="error">Required.</p>


<p *ngIf="form.get('email').hasError('pattern')"
class="error">Invalid email.</p>
<p *ngIf="form.get('email').hasError('required')"
class="error">Required.</p>


<p *ngIf="form.get('phone').hasError('minlength')"
class="error">Minimum 9 characters.</p>
<p *ngIf="form.get('phone').hasError('required')"
class="error">Required.</p>
```

In de volgende hoofdstukken bouwen we verder op de model driven (reactive form) versie van het formulier.