Lab 9: Routing

Indien we meerdere views willen gebruiken op verschillende pagina's, moeten we gebruik maken van routing. We gaan dus ook onze applicatie opsplitsen in verschillende views en we gaan bepaalde componenten hergebruiken. Om dit te doen, maken we gebruik van de routing module van angular.

Om onze routes te definiëren, maken we gebruik van een aparte file. Maak in de app folder (ter hoogte van app.module.ts) een nieuw bestand met de naam "**app.routes.ts**".

```
import {Routes} from '@angular/router';
import { ContactListComponent } from
'./contact-list/contact-list.component';
import { AddContactComponent } from
'./add-contact/add-contact.component';
import { ContactDetailComponent } from
'./contact-detail/contact-detail.component';



export const appRoutes: Routes = [
    { path: 'contact/:id', component: ContactDetailComponent },
    { path: 'add', component: AddContactComponent },
    { path: 'list', component: ContactListComponent },
    { path: '', redirectTo: '/list', pathMatch: 'full' }
];
```

Vervolgens gaan we naar **app.module.ts**. Hier importeer je de routes en voegen we de router module toe als volgt:

```
import { appRoutes } from './app.routes';
import { RouterModule } from '@angular/router';

imports: [
    ...
    RouterModule.forRoot(appRoutes)
```

```
  ],
```

Onze applicatie kan nu gebruik maken van de routes die we gedefinieerd hebben, maar we hebben nog nergens in onze applicatie een link gelegd naar waar de components van bepaalde routes moeten verschijnen. We voorzien een soort van container in de **app-component.html file** waarin de views uit de routes komen te staan. Plaats deze als eerste regel:

```
<router-outlet></router-outlet>
```

Contact-list component

De routing module is nu volledig actief, maar in onze routes importeren we 3 components die momenteel nog niet bestaan.

We starten met het aanmaken van de contact-list component. We maken deze component aan via de angular-cli:

```
ng generate component contact-list
```

Deze component zal instaat voor het oplijsten van de contacts en het voorzien van de knop voor het filteren op isFavorite. Voorzie volgende html in **contact-list.component.html**:

```
<button type="button" [class.onlyFavorites]="onlyFavorites"
(click)="toggleView(onlyFavorites)">Show {{ onlyFavorites ? 'All' :
'Favorites' }}</button>
<app-contact *ngFor="let contact of contactList; let i = index"
     [contact]="contact" (onUpdate)="handleUpdate($event)">
</app-contact>
```

Hierna mag je deze onderdelen verwijderen uit app-component.html.

Vervolgens voorzien we de nodige methodes & properties in de **contact-list.component.ts** file. Merk op dat we de addContact methode niet voorzien, deze gebruiken we later:

```
import { ContactService } from '../services/contact.service';
import { Contact } from '../models/contact.model';

export class ContactListComponent implements OnInit {
  contactList: Contact[];
  onlyFavorites: boolean = false;
```

```
  constructor(private service: ContactService) { }
  ngOnInit(): void {
    this.fetchContactList(this.onlyFavorites);
  }
  fetchContactList(onlyFav: boolean): void {
    this.service.getContactList(onlyFav).subscribe(data => {
      this.contactList = data;
    });
  }
  toggleView(onlyFav: boolean): void {
    this.onlyFavorites = !onlyFav;
    this.fetchContactList(this.onlyFavorites);
  }
  handleUpdate(): void {
    this.fetchContactList(this.onlyFavorites);
  }
}
```

In de app.component.ts file verwijder je de fetchContactList, toggleView en handleUpdate methodes. Ook de 2 properties onlyFavorites & contactList mogen verwijderd worden. Laat createContact staan. ngOnInit moet ook blijven, maar de inhoud hiervan mag ook verwijderd worden.

Add contact component

De volgende component de we aanmaken is de add-contact component. Genereer de component via de angular CLI:

```
ng generate component add-contact
```

In de **add-contact.component.html** file voeg je volgende code toe vanuit app.component.html:

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

Daarnaast kopieer je ook de createContact functie & service uit app.component.ts naar de **add-contact.component.ts** file:

```
import { ContactService } from '../services/contact.service';
import { Contact } from '../models/contact.model';

export class AddContactComponent {
  constructor(private service: ContactService) { }
  createContact(event: Contact) {
    this.service.addContact(event).subscribe();
  }
}
```

Als het maken van een contactpersoon gelukt is, willen we ook kort een boodschap tonen. Update de createContact methode als volgt en maak created aan als property van de component klasse:

```
created: boolean = false;

createContact(event: Contact) {
    this.service.addContact(event).subscribe(() => {
      this.created = true;
      setTimeout(() => this.created = false, 5000);
    });
  }
```

Voorzie in de **add-contact-component.html** volgende melding oven de contact-form-component declaratie:

```
<p *ngIf="created">Contact created successfully!</p>
```

Tenslotte importeer je de CSS van Blackboard **i**n **Resources/add-contact.component.css** naar je **add-contact.component.css**.

Contact detail component

Tenslotte maken we de contact-detail component aan voor het weergeven van de detail-weergave van een contact. Maak deze component ook aan via de Angular CLI:

```
ng generate component contact-detail
```

Voorzie volgende code in de contact-detail.component.html file:

```
<app-contact *ngIf="contact" [contact]="contact"></app-contact>
```

We moeten data doorgeven aan de appcontact view van een enkel contact object. Om één Contact object op te halen, moeten we onze service uitbreiden met een getContact methode. Voorzie deze in de **contact.service.ts**:

```
getContact(id: string) {
        let url = `${CONTACTAPIURL}${id}.json`;
        return this.http.get(url).pipe(
            map(data => new Contact(data['name'], data['email'], 
data['phone'], data['isFavorite'], data['avatar'], id))
        );
    }
```

Nu gaan we terug naar de contact-detail component. In de routes voorzagen we een input id parameter als volgt:

```
{ path: 'contact/:id', component: ContactDetailComponent },
```

Deze id kunnen we nu gaan gebruiken in onze component. Om de id op te halen, moeten we de ActivatedRoute class importeren. Via deze klasse, kunnen we de meegegeven id opvragen als volgt in de **contact-detail.component.ts**:

```
import { ActivatedRoute } from '@angular/router';
import { Contact } from '../models/contact.model';
import { ContactService } from '../services/contact.service';

export class ContactDetailComponent implements OnInit {
  id: string;
  contact: Contact;


  constructor(private route: ActivatedRoute, private service: 
ContactService) { }
```

```
  ngOnInit() {
    this.id = this.route.snapshot.params['id'];
  }
}
```

Vervolgens kunnen we de id gebruiken om via de service een specifiek contact object op te halen:

```
  ngOnInit() {
    this.id = this.route.snapshot.params['id'];
    this.getContact(this.id);
  }

  getContact(id: string): void {
    this.service.getContact(id).subscribe(data => this.contact =
data);
  }
```

Contact detail: delete

Tijd voor uitbreiding van de app. We starten met delete functionaliteiten. Hiervoor moeten we 3 zaken voorzien:

- Nieuwe methode in de service
- Methode die de service call doet
- Een knop voor het triggeren van de methode

Voeg volgende methode toe aan de service:

```
deleteContact(id: string) {
        let url = `${CONTACTAPIURL}${id}.json`;
        return this.http.delete(url);
    }
```

Vervolgens voorzien we de implementatie in de contact-detail.component.ts file. De implementatie voorziet een bevestiging en redirect terug naar de lijst:

```
import { Router, ActivatedRoute } from '@angular/router';

constructor(private router: Router, private route: ActivatedRoute,
```

```
private service: ContactService) { }

deleteContact(id: string): void {

this.service.deleteContact(id).subscribe(() => {
      this.deleted = true;
      setTimeout(() => this.router.navigateByUrl(''), 3000);
    });
}
```

Vervolgens voorzien we een knop in de contact-detail.component.html file:

Contact detail: updating

We hebben al een formulier waarin alle velden zitten, voeg dit toe aan contact-detail.component.html:

```
<app-contact *ngIf="contact"  [contact]="contact"></app-contact>
<app-contact-form></app-contact-form>
<button type="button" (click)="deleteContact(id)"
[hidden]="deleted">Delete contact</button>
<p [hidden]="!deleted">Contact deleted.</p>
```

Het formulier moet wel nog aangepast worden zodat het ook @Input data kan verwerken. Pas het formulier aan als volgt:

```
import { Component, OnInit, Output, EventEmitter, Input } from
'@angular/core';

export class ContactFormComponent implements OnInit {
   @Input() contact: Contact;

  ngOnInit() {
    this.form = new FormGroup({
      'name': new FormControl(this.contact ? this.contact.name :
null,
        [Validators.required, Validators.minLength(3)]),
      'email': new FormControl(this.contact ? this.contact.email :
null,
```

```
        [Validators.required,
Validators.pattern(/^[a-z0-9_\.]+@[a-z0-9_\.]+/i)]),
      'phone': new FormControl(this.contact ? this.contact.phone :
null,
        [Validators.required, Validators.minLength(9)]),
      'isFavorite': new FormControl(this.contact ?
this.contact.isFavorite : false),
      'avatar': new FormControl(this.contact ? this.contact.avatar :
null)
    });
  }
}
```

Het formulier kan nu omgaan met input data. Pas de view van contact-**detail.component.html** aan:

```
<app-contact *ngIf="contact"  [contact]="contact"></app-contact>
<app-contact-form *ngIf="contact"
[contact]="contact"></app-contact-form>
```

Het form met input velden werkt. Vervolgens verbergen we het formulier als het niet nodig is en voorzien we een mogelijkheid om te bevestigen dat het updaten gelukt is. Maak 2 nieuwe booleans aan in de **contact-detail.component.ts** file:

```
  editing: boolean = false;
  updated: boolean = false;

toggleEditing(editing: boolean): void {
    this.editing = !editing;
}
```

Vervolgens linken we het [hidden] attribuut aan de contact-component. Daarnaast passen we de *ngIf van onze contact-form ook aan zodat het enkel getoond wordt als de boolean editing true is. Voorzie een extra edit knop en voeg een class toe aan beide knoppen in onze view **contact-detail.component.ts**:

```
<app-contact *ngIf="contact"  [contact]="contact"></app-contact>
<app-contact-form *ngIf="contact && editing"
[contact]="contact"></app-contact-form>
```

```
<button type="button" class="edit"
(click)="toggleEditing(editing)">{{ editing ? 'Cancel' : 'Edit
contact' }}</button>

<button type="button" (click)="deleteContact(id)" [hidden]="deleted"
class="delete">Delete contact</button>

<p [hidden]="!updated" class="update-message">Contact update
successful.</p>

<p [hidden]="!deleted" class="delete-message">Contact deleted.</p>
```

Tenslotte moeten we de update functionaliteit nog schrijven. Weet nog dat het form de data emit naar de parent. We voorzien dus een koppeling in de contact-detail component html & ts files:

```
updateContact(contact: Contact): void {
    this.service.updateContact(this.id, contact).subscribe(() => {
      this.getContact(this.id);
      this.editing = false;
      this.updated = true;
      setTimeout(() => this.updated = false, 3000);
    });
  }
```

State management

Tot nu toe gebruikte we booleans in de add contact en contact detail component op de status van een view bij te houden (bv. Created, updated, editing, deleted). Voor een duidelijke structuur is het interessanter om een enum aan te maken waarin de state bijgehouden wordt. Deze enums kunnen gegenereerd worden via de angular CLI:

```
ng generate enum models/editor-state
```

Open het bestand **editor-state.enum.ts** en geef het volgende inhoud:

```
export enum EditorState {
    null, created, editing, updated, deleted
}
```

Ga naar **add-contact.component.ts** en importeer de enum. Vervang de created boolean door de state variable van het type EditorState:

```
import { EditorState } from '../models/editor-state.enum';

export class AddContactComponent {
  state: EditorState = EditorState.null;
  editorState: any = EditorState;

  createContact(event: Contact) {
    this.service.addContact(event).subscribe(() => {
      this.state = EditorState.created;
      setTimeout(() => this.state = EditorState.null, 3000);
    });
  }
}
```

In de view voorzie je volgende aanpassing:

```
<p *ngIf="state === editorState.created">Contact created
successfully!</p>
```

Herhaal deze stappen in de contact-detail component. We starten met contact-detail.component.ts:

```
import { EditorState } from '../models/editor-state.enum';
editorState: any = EditorState;
state: EditorState;

deleteContact(id: string): void {
    this.service.deleteContact(id).subscribe(() => {
      this.state = EditorState.deleted;
      setTimeout(() => this.router.navigateByUrl(''), 3000);
    });
}

toggleEditing(editing: boolean): void {
    if (this.state === EditorState.null) {
      this.state = EditorState.editing;
    } else {
```

```
      this.state = EditorState.null;
    }
}

updateContact(contact: Contact): void {
    this.service.updateContact(this.id, contact).subscribe(() => {
      this.getContact(this.id);
      this.state = EditorState.updated;
      setTimeout(() => this.state = EditorState.null, 3000);
    });
  }
```

Vervolgens voorzie je volgende aanpassingen in de view:

```
<app-contact *ngIf="contact && state !== editorState.editing"
[contact]="contact"></app-contact>

<app-contact-form *ngIf="contact && state === editorState.editing"
[contact]="contact" (onSubmit)="updateContact($event,
id)"></app-contact-form>

<button type="button" class="edit" (click)="toggleEditing(editing)"
[hidden]="state === editorState.deleted">{{ state ===
editorState.editing ? 'Cancel' : 'Edit contact' }}</button>

<button type="button" (click)="deleteContact(id)" [hidden]="state
=== editorState.deleted" class="delete">Delete contact</button>

<p [hidden]="state !== editorState.updated"
class="update-message">Contact update successful.</p>

<p [hidden]="state !== editorState.deleted"
class="delete-message">Contact deleted.</p>
```

Kopieer de CSS van Blackboard uit Resources/contact-detail.component.css naar je eigen contact-detail.component.css file.

Alles koppelen

Het enige wat we nog niet gedaan hebben, is het voorzien van een duidelijke navigatie. Voeg volgende code toe aan de **app.component.html** file:

```
<nav>
        <ul>
```

```
                <li><a routerLink="/list"
[routerLinkActive]="'current'" title="List">List</a></li>
                <li><a routerLink="/add"
[routerLinkActive]="'current'" title="Add contact">Add
contact</a></li>
        </ul>
</nav>
```

Kopieer de stijlen van Blackboard uit Resources/app.component.css naar je eigen app.component.css file.

De routerLink kan ook toegevoegd worden aan andere HTML tags zoals die van een component. We passen dit toe om een link te leggen op elke contact component in de contact list. Open de file contact-list.component.html en voorzie volgende aanpassingen:

```
<app-contact *ngFor="let contact of contactList; let i = index"
     [contact]="contact" (onUpdate)="handleUpdate($event)"
class="isLink" routerLink="/contact/{{contact.id}}">
</app-contact>
```

Kopieer de stijlen van Blackboard uit Resources/contact.component.css naar je eigen contact.component.css file.

We hebben enkel nog één probleem: als er op de checkbox geklikt wordt, wordt de detail view nog steeds geladen. Dit is uiteraard niet de bedoeling. Pas eerst de **contact.component.html** file aan:

```
<input type="checkbox" name="isFavorite" id="isFavorite"
[(ngModel)]="contact.isFavorite"
            (click)="toggleFavorite($event, contact.id,
contact.isFavorite)">
```

Vervolgens pass je de contact.component.ts file aan:

```
toggleFavorite(event: any, id: string, isFavorite: boolean): void {
        event.stopPropagation();
        this.service.updateContact(id, {isFavorite:
!isFavorite}).subscribe(() => this.onUpdate.emit);
}
```