Lab 7: Data ophalen met HTTP

De service in de applicatie maakt momenteel gebruik van een array met mock data. In deze lab wordt deze mock data vervangen door het aanspreken van een rest API.

De contactService wordt aangepast zodat dit het centrale aanspreekpunt wordt voor het ophalen en wegschrijven van de data. Om http calls uit te kunnen voeren, moet de HttpClientModule geïmporteerd worden in de app.module.ts file:

```
imports: [
    BrowserModule,
    ...
    HttpClientModule
  ],
```

We hebben 4 zaken nodig om van start te gaan:

- De http klasse van Angular
- RxJS Observables om de data op te vangen die terug komt
- De RxJS map operator
- Een instantie van http wat doorgegeven wordt aan de constructor van onze service

Voorzie volgende aanpassingen aan de code van je service:

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs'
import { map } from 'rxjs/operators'

...

constructor(private http: HttpClient) {}
```

Verwijder de initialisatie van de contactList variabele in je klasse.

Contact component & contact model

De achterliggende database die we gebruiken is Firebase. Firebase ondersteunt geen arrays van nature uit. In plaats daarvan gebruikt firebase lijsten van objecten met een unieke key voor elk object. We moeten deze key toevoegen aan onze model om deze objecten aan te kunnen spreken.

Voeg een nieuwe property **id** toe aan contact.model.ts:

```
id: string;

constructor(name: string, email: string, phone: string, isFavorite =
false, avatar = ‘assets/avatar.png’, id?: string){
   this.name = name;
   this.email = email;
   this.phone = phone;
   this.isFavorite = isFavorite;
   this.avatar = avatar;
   this.id = id;

}
```

De contact component kreeg een index mee als input property. Verwijder deze input binding van de component en verwijder de bindings uit de app component html:

```
<app-contact *ngFor="let contact of contactList; let i = index"
             [contact]="contact" (onUpdate)="handleUpdate($event)">
</app-contact>
```

getContactList methode upgrade

De getContactList methode wordt volledig aangepast om te werken met een api call. Voorzie een nieuwe variabele **boven** de service klasse met de URL naar onze API:

```
const BASEAPIURL: string =
'https://webexpert1718.firebaseio.com/contacts.json';
```

Vervolgens pas je de getContactList method aan. Deze maakt nu een http call en zet de JSON data om naar contact objecten. Hierna wordt er een Observable teruggegeven met de data of een foutboodschap. Op Observables kunnen verschillende operators toegepast worden. Werken met Observables heeft gelijkenissen met werken met arrays.

```
getContactList(): Observable<Contact[]> {
        return this.http.get(BASEAPIURL).pipe(
            map(this.parseContactData)
        );
}
```

De http.get methode haalt de data op vanuit de API. Deze data wordt omgezet naar JSON. Vervolgens wordt het JSON object doorgegeven aan de methode parseContactData via de map operator. De methode parseContactData bestaat nog niet. Voeg ze toe aan de service:

```
parseContactData(rawContacts: any[]): Contact[] {
        return Object.keys(rawContacts).map(key => {
            let contact = rawContacts[key];
            return new Contact(
                contact.name,
                contact.email,
                contact.phone,
                contact.isFavorite,
                contact.avatar,
                key
            );
        });
    }
```

Zoals reeds aangehaald, werkt firebase met een key gekoppeld aan objecten. Via Object.keys krijgen we welke key uit onze contact data. Via de map operator geven we elke key door naar een functie. In deze functie wordt er een nieuw Contact object gemaakt en worden de waardes voor die key gekoppeld aan dit object. Wat we terugkrijgen is een array van Contacts die vervolgens gebruikt kan worden in onze applicatie.

getContactList gebruiken in een component

Om observables te gebruiken, moet je “subscriben” op zijn updates. Pas de app.component.ts aan als volgt:

```
ngOnInit(): void {
    this.fetchContactList();
  }

  fetchContactList(): void {
    this.service.getContactList().subscribe(data => {
      this.contactList = data;
    });
```

```
  }

  handleUpdate(): void {
    this.fetchContactList();
  }
```

Zet de code in de addContact() methode voorlopig in commentaar. Bij het starten van de app, zou de lijst nu vanuit de API ingeladen moeten worden.

addContact aanpassen

De aanpassingen binnen addContact liggen in dezelfde lijn als die van getContactList. We werken hier ook met observables.

Bij het toevoegen van data hoeft er niet gedaan worden met zaken die terugkomen. We moeten enkel weten of onze POST geslaagd is of niet. We voorzien dus enken een post methode en een catch methode. Pas je service aan als volgt:

```
addContact(contact: Contact): Observable<any> {
        return this.http
                .post(BASEAPIURL, contact)
}
```

Pas daarna de methode createContact() uit de app.component.ts aan als volgt:

```
createContact(event: Contact) {
   this.service.addContact(event).subscribe(() =>
this.fetchContactList())
  }
```

toggleFavorite aanpassen.

We hebben ook nog de functionaliteit op de favorite optie aan/uit te zetten. Hiervoor moeten we ook de nodige functionaliteit voor voorzien. Voorzie eerst een nieuwe constante in je contact.service.ts:

```
const BASEAPIURL: string =
'https://webexpert1718.firebaseio.com/contacts.json';
const CONTACTAPIURL: string =
'https://webexpert1718.firebaseio.com/contacts/';
```

de toggleFavorite methode mag je uit de service verwijderen. In plaats daarvan maken we een methode updateContact aan. Deze methode krijgt een id en data binnen. Deze data versturen we naar onze API via http.patch. De methode kan gebruikt worden voor elke property van eender welk object:

```
    updateContact(id: string, data: any): Observable<any> {
        let url: string = `${CONTACTAPIURL}${id}.json`;
        return this.http.patch(url, data);
    }
```

Vervolgens passen we de contact.component.html file aan:

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

Hierna pas je de contact.component.ts file aan:

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

De methode in de parent hoeft niet aangepast te worden, omdat deze enkel de fetchContactList() methode gaat uitvoeren.

Filtering voorzien

Momenteel hebben de contacten een isFavorite veld, maar we doen hier nog niets mee. In onze app component voorzien we een knop waarmee we kunnen filteren op dit veld.

Ten eerste wordt de getContactList methode in de service aangepast zodat het een argument meekrijgt. We voegen ook een extra map operator toe die de contacten zal filteren:

```
getContactList(onlyFavorites: boolean): Observable<Contact[]> {
        return this.http.get(BASEAPIURL).pipe(
            map(this.parseContactData),
            map((contacts: Contact[]) => {
```

```
                return onlyFavorites ? this.filterContacts(contacts)
: contacts;
```

```
            })
        );
    }
```

In de nieuwe map operator, callen we de functie filterContacts. Maak deze functie aan:

```
filterContacts(contacts: Contact[]): Contact[] {
        return contacts.filter(contact => contact.isFavorite);
}
```

Nu dat de service geüpdatet is, moet de app component nog aangepast worden. Start met het toevoegen van een button in de app.component.html die in staat voor het oproepen van de filtering:

```
<button type="button" [class.onlyFavorites]="onlyFavorites"
        (click)="toggleView(onlyFavorites)">Show {{ onlyFavorites ?
'All' : 'Favorites' }}</button>
```

Pas vervolgens de app.component.ts file aan. **Hierin maak je een nieuwe property onlyFavorites die standaard de waarde false krijgt**. Voorzie daarna volgende aanpassingen:

```
ngOnInit(): void {
    this.fetchContactList(this.onlyFavorites);
}

fetchContactList(onlyFav: boolean): void {
    this.service.getContactList(onlyFav).subscribe(data => {
      this.contactList = data;
    });
}

handleUpdate(): void {
```

```
    this.fetchContactList(this.onlyFavorites);
}


createContact(event: Contact) {
    this.service.addContact(event).subscribe(() =>
this.fetchContactList(this.onlyFavorites))
}


toggleView(onlyFav: boolean): void {
    this.onlyFavorites = !onlyFav;
    this.fetchContactList(this.onlyFavorites);
}
```

De knop zorgt nu voor een toggle tussen alle contacten en enkel de favorieten. Tenslotte voeg je de css stijlen toe die terug te vinden zijn op blackboard onder Resources/app.component.css.