Zadatak 1 Termin 5

Ovaj zadatak se nastavlja na kod primer_1 iz ovog termina.

1 Dodavanje nove pice

U *PizzaService* je dodata nova metoda *add* koja kao parametar prima picu koju želimo da dodamo u kolekciju. Iz API-ja servera vidimo da za dodavanje pice treba da pošaljemo POST zahtev na http://localhost:3000/api/pizzas i pošaljemo novu picu sa tim zahtevom. Kako nam nije posebno bitna povratna vrednost zahteva jer ne treba ništa da prikažemo na osnovu nje, deklarisaćemo je kao *Observable<any>*.

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

import { Pizza } from '../model/pizza.model';
import { PizzaSearchResult } from '../model/pizzaSearchResult.model';

const baseUrl = "http://localhost:3000/api/pizzas";

@Injectable({
  providedIn: 'root'
})
export class PizzaService {

    constructor(private http: HttpClient) { }

    getAll(): Observable<PizzaSearchResult>{
        return this.http.get(baseUrl).pipe(map(
            data => { return new PizzaSearchResult(data) }
        ));
    }

    add(newPizza: Pizza): Observable<any> {
        return this.http.post(baseUrl, newPizza);
    }

}
```

1. U *template*-u *EditPizzaComponent* je na <form> element dodato da se na *submit* događaj poziva metoda *onSubmit()*. Metoda *onSubmit*() je dodata u *EdditPizzaComponent* klasu. Za početak, samo ispisujemo vrednost submitovane pice na konzoli da bismo proverili da li je sve u redu sa formom.

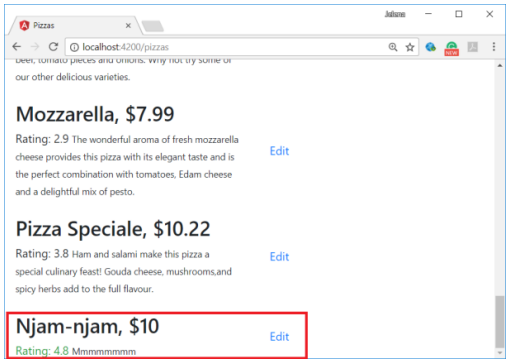
```
<form [formGroup]="pizzaForm" (submit)="onSubmit()">
```

```
onSubmit(): void {
    this.pizza = new Pizza(this.pizzaForm.value);
    console.log(this.pizza);
}
```

2. Da bismo u *onSubmit* metodi komponente *EditPizzaComponent* koristili *PizzaService*, prvo treba da ga injektujemo u komponentu. Sada ovom servisu možemo pristupati sa *this.pizzaService*.
3. U *onSubmit* metodi komponente *EditPizzaComponent* ćemo pozvati metodu *add* od *PizzaService*-a i proslediti joj picu koja treba da se doda. Budući da metoda *add* vraća *Observable* objekat, mi se *subscribe*-ujemo da pratimo promene ovog objekta (do promene će doći kada server obavi dodavanje, a tada će okinuti *next listener*).

```
export class EditPizzaComponent implements OnInit {
    pizza: Pizza = new Pizza();
    pizzaForm: FormGroup;

    constructor(private fb :FormBuilder, private pizzaService: PizzaService) {
        this.pizzaForm = this.fb.group({
            'name': '',
            'description': '',
            'grade': '',
            'price': ''
        });
    }

  ngOnInit() {
  }

  onSubmit(): void {
      this.pizza = new Pizza(this.pizzaForm.value);
      console.log(this.pizza);

      this.pizzaService.add(this.pizza).subscribe({
        next: (data: any) => {

        }
      });
  }

}
```

Možete testirati aplikaciju. Nakon klika na *submit* forme, pređite ručno na početnu stranu i pogledajte da li se pica sada nalazi u listi svih pica:



4. Nakon dodavanja pice, bilo bi dobro da se automatski redirektujemo na meni, tako da korisnik može odmah da vidi promene koje je napravio. Ovo je rešeno primenom *Router* servisa (*navigate* smo pozvali unutar *next listener*-a kako bismo se redirektovali **samo** ako uspešno dodamo picu).

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormBuilder } from '@angular/forms';
import { Router } from '@angular/router';
import { Pizza } from '../model/pizza.model';
import { PizzaService } from '../services/pizza.service';

@Component({
  selector: 'app-edit-pizza',
  templateUrl: './edit-pizza.component.html',
  styleUrls: ['./edit-pizza.component.css']
})
export class EditPizzaComponent implements OnInit {
    pizza: Pizza = new Pizza();
    pizzaForm: FormGroup;

    constructor(private fb :FormBuilder, private pizzaService: PizzaService,
        private router: Router) {
        this.pizzaForm = this.fb.group({
            'name': '',
            'description': '',
            'grade': '',
            'price': ''
        });
    }

  ngOnInit() {
  }

  onSubmit(): void {
      this.pizza = new Pizza(this.pizzaForm.value);
      console.log(this.pizza);

      this.pizzaService.add(this.pizza).subscribe({
        next: (data: any) => {
            this.router.navigate(["/pizzas"]);
        }
      });
  }

}
```

2 Ažuriranje pice

5. Pre svega ćemo rešiti problem da se, po prebacivanju na formu, moraju prikazati trenutne vrednosti pice:

   - Preuzećemo id pice koju treba da prikažemo iz URL-a naše aplikacije
   - Poslaćemo GET zahtev serveru na http://localhost:3000/api/pizzas/:id da tu picu dobijemo
   - Kada nam server vrati rezultat, prikazaćemo ga na formi.

6. U *PizzaService* je dodata nova metoda *get* koja kao parametar prima *id* pice koju želimo da dobavimo. Iz API-ja servera vidimo da picu možemo dobaviti slanjem GET zahteva na http://localhost:3000/api/pizzas/:id. Obratite pažnju da smo na *baseUrl* (http://localhost:3000/api/pizzas) dodali *id* pice koju želimo da dobavimo. Server vraća objekat koji opisuje traženu picu pa povratnu vrednost deklarišemo kao *Observable<Pizza>*:

```
get(id: number): Observable<Pizza> {
    return this.http.get(baseUrl + "/" + id).pipe(map(
        data => { return new Pizza(data) }
    ));
}
```

7. Preuzimanje *id*-a pice iz putanje možemo rešiti upotrebom *ActivatedRoute* servisa:

8. U *onInit* metodi smo proverili da li je *id != undefinded* (ako smo kliknuli na link za dodavanje pica, onda u putanji ne postoji *id* parametar pa bi bio *undefined*). U slučaju da nije *undefined*, pozivamo upravo napravljenu *get* metodu *PizzaService*-a. Pošto ova metoda vraća *Obsrvable* objekat, *subscribe*-ujemo se da pratimo promene tog objekta i time šaljemo zahtev serveru. Kada se promeni (podaci su stigli sa servera), prikazujemo vrednosti dobavljene pice na formi pomoću *patchValue* i čuvamo ih u modelu podataka:

```
export class EditPizzaComponent implements OnInit {
    pizza: Pizza = new Pizza();
    pizzaForm: FormGroup;

    constructor(private fb :FormBuilder, private pizzaService: PizzaService,
        private router: Router, private route: ActivatedRoute) {
        this.pizzaForm = this.fb.group({
            'name': '',
            'description': '',
            'grade': '',
            'price': ''
        });
    }

  ngOnInit() {
      let id: number = Number(this.route.snapshot.params["id"]);
      if (id) {
        this.pizzaService.get(id).subscribe({
            next: (data: Pizza) => {
                this.pizza = data;
                this.pizzaForm.patchValue(this.pizza);
            }
        });
      }
  }
```

9. U *PizzaService* je dodata nova metoda *update* koja kao parametar prima picu koju želimo da ažuriramo (prosleđeni objekat sadrži nove vrednosti pice). Iz API-ja servera vidimo da za ažuriranje pice treba da pošaljemo PUT zahtev na http://localhost:3000/api/pizzas/:id i pošaljemo novu vrednost pice sa tim zahtevom. Kako nam nije posebno bitna povratna vrednost zahteva jer ne treba ništa da prikažemo na osnovu nje, deklarisaćemo je kao *Observable<any>*.

   Obratite pažnju da smo na *baseUrl* (http://localhost:3000/api/pizzas) dodali id pice koju želimo da ažuriramo.

```
update(pizza: Pizza): Observable<any> {
    return this.http.put(baseUrl + "/" + pizza._id, pizza);
}
```

10. U *EditPizzaComponent* treba da u *onSubmit* metodi podržimo ažuriranje pice. Ako se sećate prošlog termina, zaključili smo da možemo razlikovati da li se radi o dodavanju nove pice ili ažuriranju postojeće na osnovu toga da li je *id* parametar definisan. Kako se u formi ne zadaje *_id* pice, on se mora dopisati objektu na osnovu pročitanog parametra *id* iz URL-a. Neophodno je da objekat poseduje u sebi *_id* jer se on dopisuje na URL zahteva u servisu.

```
onSubmit(): void {
    this.pizza = new Pizza(this.pizzaForm.value);
    console.log(this.pizza);

    let id: number = Number(this.route.snapshot.params["id"]);
    if (id) {
      this.pizza._id = id;
      this.pizzaService.update(this.pizza).subscribe({
          next: (data: any) => {
              this.router.navigate(["/pizzas"]);
          }
      });
    } else {
      this.pizzaService.add(this.pizza).subscribe({
          next: (data: any) => {
              this.router.navigate(["/pizzas"]);
          }
      });
    }
}
```

3 Brisanje pice

11. U servis *PizzaService* je dodata *remove* metoda koja kao parametar prima *id* pice koju želimo da obrišemo. Iz API-ja servera vidimo da za brisanje pice treba da pošaljemo DELETE zahtev na http://localhost:3000/api/pizzas/:id. Kako nam nije posebno bitna povratna vrednost zahteva jer ne treba ništa da prikažemo na osnovu nje, deklarisaćemo je kao *Observable<any>*.

```
remove(id: number): Observable<any> {
    return this.http.delete(baseUrl + "/" + id);
}
```

12. U templejtu komponente *PizzaDetailsComponent* je dodato *Delete* dugme i na njega je dodato da se osluškuje *click* događaj i na njega poziva metoda *onDelete()* kojojj se prosleđuje *_id* odabrane pice:

```
<div class="row">

    <div class="col-sm-6">
        <br>
        <h3>{{pizza.name}}, ${{pizza.price}}</h3>
        <div>
            <span [ngClass]="{'text-success':pizza.grade>4}">Rating: {{pizza.grade}}</span>     <small>{{pizza.description}}</small>
        </div>
    </div>

    <div class="col-sm-6 my-auto">
        <br>
        <button class="btn btn-link" [routerLink]="['/pizzas/', pizza._id]">Edit</button>
        <button class="btn btn-link" (click)="onDelete(pizza._id)">Delete</button>
    </div>
</div>
```

13. U klasi komponente *PizzaDetailsComponent* potrebno je deklarisati *output pizzaDeleted* tipa *EventEmitter<number>* (da bi podžao prosleđivanje *id*-a) kojim ćemo PizzaListComponent-u propagirati događaj brisanja pice.

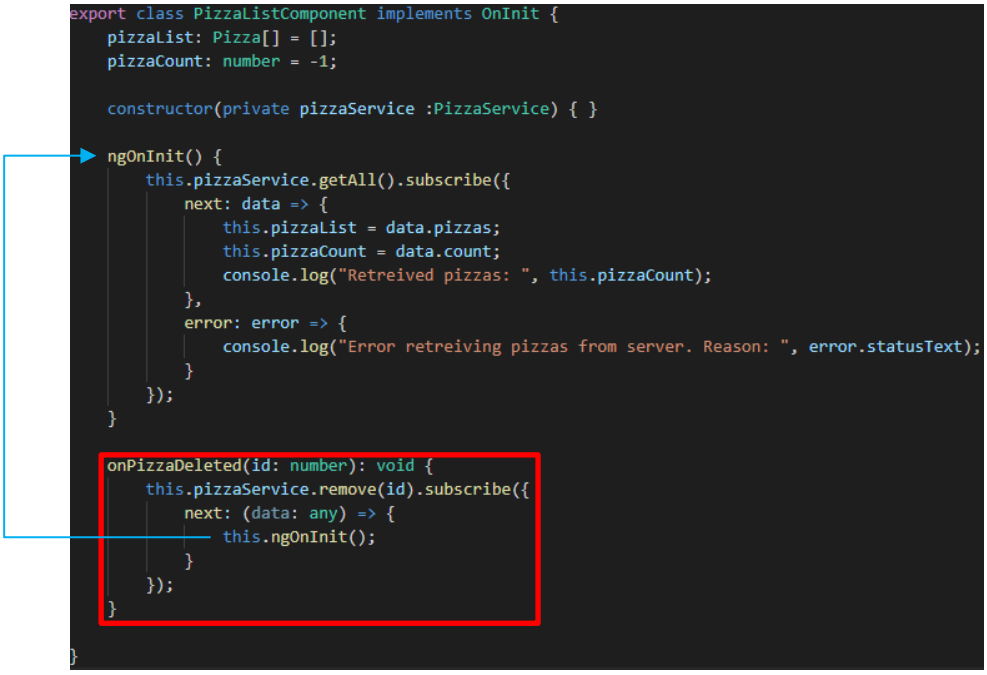
14. Treba da u *onDelete()* metodi pozovemo *emit()* metodu *pizzaDeleted output*-a i PizzaListComponent-u prosledimo id pice koju je potrebno obrisati:

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

@Component({
    selector: 'app-pizza-details',
    templateUrl: './pizza-details.component.html',
    styleUrls: ['./pizza-details.component.css']
})
export class PizzaDetailsComponent implements OnInit {
    @Input() pizza: Pizza = new Pizza();
    @Output() pizzaDeleted: EventEmitter<number> = new EventEmitter();

    constructor() {}

    ngOnInit() {}

    onSubmit(){}

    onDelete(id: number) {
        this.pizzaDeleted.emit(id);
    }

}
```

15. U *PizzaListComponent* definisaćemo metodu *onPizzaDeleted(id: number)* i u njoj ćemo implementirati poziv *remove* metode servisa. U *next listener*-u (nakon uspešnog brisanja pice) ćemo od servera zatražiti ažurnu listu pica jer se učitana lista neće automatski osvežiti jer se brisanje ne odvija nad njom. Ovo će rezultovati sekvencom poziva ka serveru:

    I. Zahtev za birsanjem

    II. Zahtev za čitanjem



```
export class PizzaListComponent implements OnInit {
    pizzaList: Pizza[] = [];
    pizzaCount: number = -1;

    constructor(private pizzaService :PizzaService) { }

    ngOnInit() {
        this.pizzaService.getAll().subscribe({
            next: data => {
                this.pizzaList = data.pizzas;
                this.pizzaCount = data.count;
                console.log("Retreived pizzas: ", this.pizzaCount);
            },
            error: error => {
                console.log("Error retreiving pizzas from server. Reason: ", error.statusText);
            }
        });
    }

    onPizzaDeleted(id: number): void {
        this.pizzaService.remove(id).subscribe({
            next: (data: any) => {
                this.ngOnInit();
            }
        });
    }
}
```

16. U *PizzaListComponent* *template*-u treba da povežemo *pizzaDeleted* *optupt* *PizzaDetails* komponente sa *onPizzaDeleted* metodom *PizzaListComponent*:

```
<app-pizza-details *ngFor="let pizza of pizzaList" [pizza]="pizza" (pizzaDeleted)="onPizzaDeleted($event)"></app-pizza-details>
```

Tesitrajte vašu aplikaciju. Sada bi trebalo da se po brisanju pice osvežava i prikazana lista pica.