# Primer 1: komunikacija sa serverom

1. Napravljena je nova aplikacija pizzas: ng new pizzas
2. U pizza/model/pizza.model.ts je (na osnovu podataka koje vraća server za pojedinačnu picu) napravljen model pice (klasa *Pizza*)
3. U pizza/model/pizzaSearchResult.ts je napravljena klasa *PizzaSearchResult* koja predstavlja model podataka za objekat koji server vraća na http://localhost:3000/api/pizzas.
    a. Ovaj objekat ima dva polja *count* (ukupan broj pica na serveru) i *results* (pice vraćene kao rezultat upita)
    b. (Baš kao u *Pizza* klasi) napravićemo konstruktor koji može da primi *JavaScript* objekat sa poljima *count* i *results* i na osnovu njega inicijalizuje polja *PizzaSearchResult*
    c. Polje *pizzas* koje je niz ***Pizza*** objekata inicijalizujemo na osnovu *obj.results* koji je niz ***JavaScript*** objekata. Za ovo ćemo iskoristiti funkciju *map* (funkcija definisana nad *JavaScript* nizom). Ova funkcija ima mogućnost da jedan niz konvertuje u drugi: u ovom slučaju svaki element niza *obj.results* (*JavaScript* objekat) se konvertuje u *Pizza* obekat. Dakle, *map* u ovom slučaju niz *JavaScript* objekata konvertuje u niz *Pizza* objekata.

```
constructor(obj? :any){
    this.pizzas = obj && obj.results.map((elem: any) => { return new Pizza(elem); }) || [];
    /*
    // Linija iznad ima isti efekat kao kod u ovom komentaru:
    this.pizzas = [];
    for (let elem of obj.results) {
        this.pizzas.push(new Pizza(elem));
    }
    */
    this.count = obj && obj.count || null;
}
```

4. Dodat je *Bootstrap* u aplikaciju.
5. Da bismo koristili *HttpClient* servis, moramo u modul naše aplikacije ubaciti *HttpClientModule*:

```
...
import { HttpClientModule } from '@angular/common/http';
...
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

6. U direktorijumu pizza/services/ napravićemo servis *PizzaService* koji će služiti za komunikaciju sa serverom: ng g service pizza/services/pizza

7. U *PizzaService*u ćemo napraviti *getAll*() metodu čiji zadatak je da se obrati serveru u cilju dobavljanja svih pica sa servera:

    a. Napravili smo konstantu *baseUrl* koja sadrži URL servera koji pruža pristup podacima o picama

    b. Da bismo slali *http* zahtev serveru potreban nam je *HttpClient* servis. Importovali smo *HttpClient* iz '@angular/common/http' i injektovali ga u *PizzaService* (tako što smo ga naveli kao parametar konstruktora)

    c. *getAll*() metoda ne može kao svoju povratnu vrednost da vrati direktno rezultat koji stiže sa servera – ne znamo koliko će trajati dobavljanje podataka sa servera, a ne želimo da *getAll*() metoda bude blokirajuća (da ceo kod naše aplikacije mora da čeka dok se ona ne izvrši). Zbog toga *getAll*() kao povratnu vrednost vraća *Observable* objekat koji predstavlja rezultate sa servera (otud tip *PizzaSearchResult*). Svi koji su zainteresovani mogu da se *subscribe*uju da osluškuju ovaj *observable* objekat – kada stignu podaci sa servera, njegova vrednost će se promeniti i on će o toj promeni obavestiti sve *subsribe*re i poslati im svoju novu vrednost.

    d. Moramo importovati *Observable* iz 'rxjs'

    e. O *Observable* objektu možemo donekle razmišljati kao o nizu – predstavlja niz (sekvencu) vrednosti koje uzima neki objekat. Recimo, *http.get*() vraća *Observable<Response>* (niz vrednosti objekta tipa *Response*). Međutim, naša namera je da iz metode vraćamo *Observable<PizzaSearchResult>*. Dakle, želimo da *Observable<Response>* objekat koji vraća *http.get*() konvertujemo u *Observable<PizzaService>* objekat. Slično kao što postoji *map* funkcija definisana nad *JavaScript* nizovima, na *observable* objektu možemo primeniti *pipe(map(...))* da vrednosti *observable* objekta konvertujemo u (mapriamo na) druge vrednosti.

        i. *map* će za svaku vrednost (koju dobijemo kada se *observable* promeni) pozvati funkciju koja joj je prosleđena: u toj funkciji mi dobijenu *Response* vrednost konvertujemo u *PizzaSearchResult* vrednost

    f. *map* operator moramo importovati iz 'rxjs/operators'

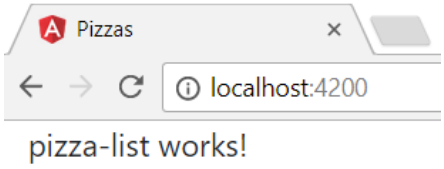
```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';  b
import { Observable } from 'rxjs';  d
import { map } from 'rxjs/operators';  f

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

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

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

  constructor(private http :HttpClient) { }  b

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

8. Napravljena je komponenta pizza-list u direktorijumu pizza: ng g component pizza/pizza-list

9. Na templejt *AppComponent* se je dodat selektor *PizzaListComponent*:

```
<div class="container-fluid">
  <div class="row">
    <div class="col">
      <app-pizza-list></app-pizza-list>
    </div>
  </div>
</div>
```

Za sada, aplikacija izgleda ovako:




10. U klasi komponente *PizzaListComponent* je napravljeno polje *pizzaList* tipa *Pizza[]* koje će da sadrži niz pica koje treba da se prikažu i polje *pizzaCount* koje će da čuva ukupan broj pica na serveru. Na prethodnom terminu smo rekli da ćemo *onInit*() metodu koristiti za „teže“ inicijalizacije. Zbog toga ćemo u okviru ove metode zatražiti podatke o picama sa servera i (kada nam server vrati odgovor smestiti ih u polje *pizzaList*):
    a. Napravili smo polje *pizzaList* tipa *Pizza[]* i *pizzaCount* tipa *number*
    b. Injektovali smo servis *PizzaService* u komponentu. Sada mu možemo pristupati sa *this.pizzaService*
    c. Importovali smo klase *PizzaService* i *Pizza* iz odgovarajućih fajlova
    d. U *onInit*() metodi smo pozvali *getAll*() metodu servisa *PizzaService*. Ova metoda kao povratnu vrednost vraća *Observable*:
        i. Mi se *subscribujemo* da pratimo vrednost ovog observable objekta jer želimo da znamo kada se njegova vrednost menja (promeniće se kada stigne odgovor sa servera)
        ii. Kada stigne odgovor sa servera (observable se promenio), nama će taj observable da javi da se promenio i pošalje svoju novu vrednost.
        iii. Ako pogledate metodu *getAll* od *PizzaService*-a, observable objekat koji posmatramo je tipa *PizzaSearchResult*. Dakle, u *subscribe* metodi promenjena vrednost observable objekata koga smo primili će biti tipa *PizzaSearchResult* (parametar koji smo nazvali *data* je tipa *PizzaSearchResult*)
        iv. Mi ćemo objekat *data* iskoristiti da ažuriramo polja *pizzaList* i *pizzaCount* i, radi testiranja ćemo na konzolu ispisati broj pica koje smo primili sa servera.

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

@Component({
  selector: 'app-pizza-list',
  templateUrl: './pizza-list.component.html',
  styleUrls: ['./pizza-list.component.css']
})
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);
            }
        });
    }

}
```

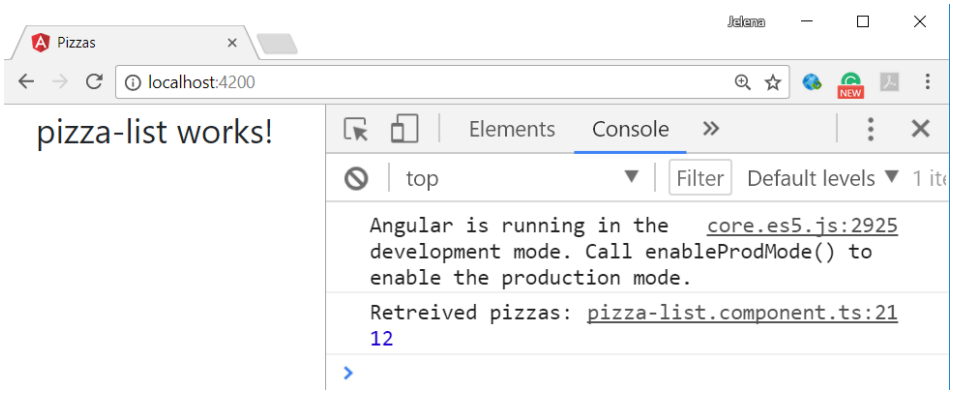
b

a

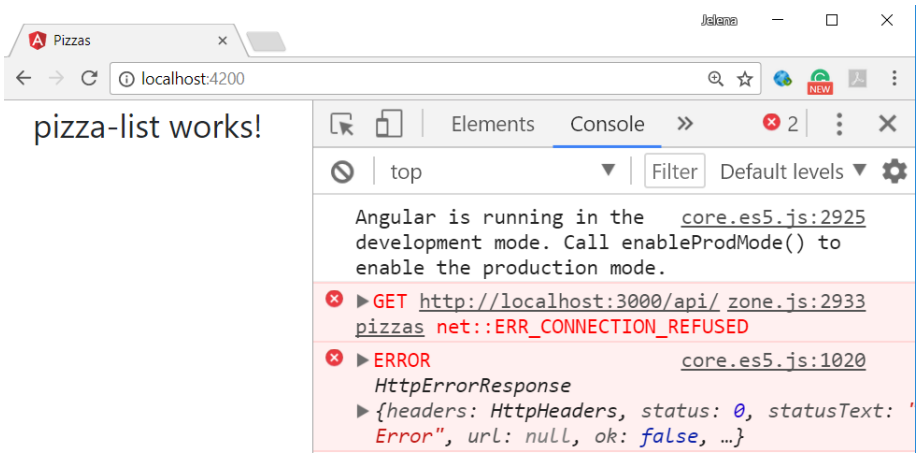
c

d

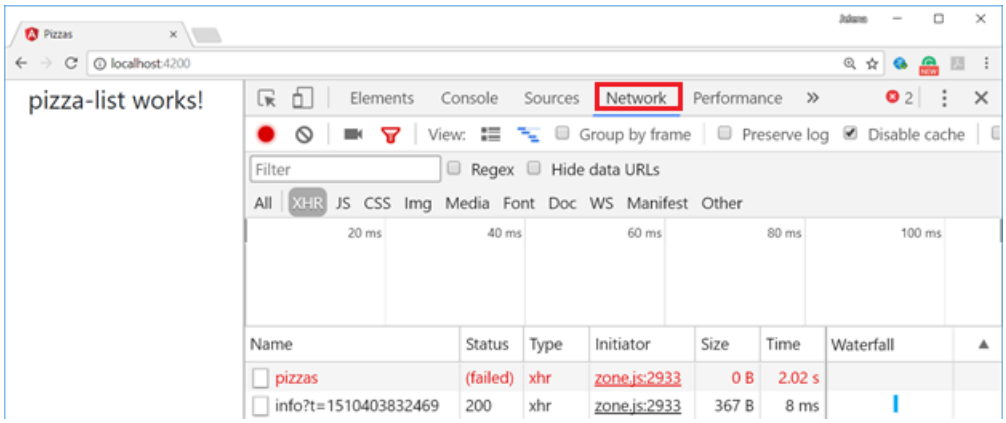
Sada testirajte aplikaciju: odmah po pokretanju (jer se tada prikaže komponenta *PizzaListComponent* pa će se izvršiti i njena *onInit* metoda) bi trebalo da se na konzoli ispiše broj pica skinutih sa servera:



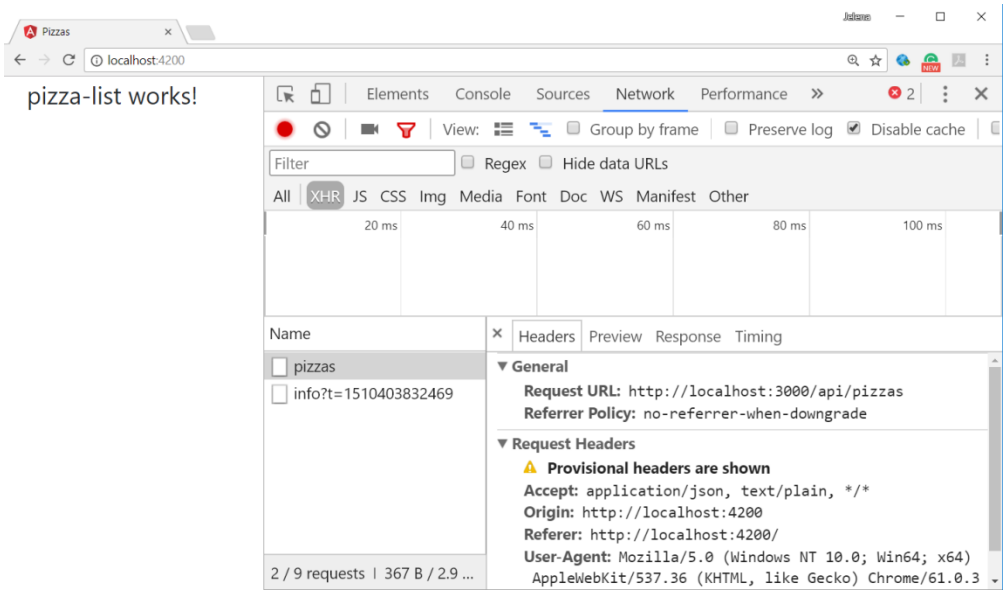

**Napomena: da bi ovo radilo, back-end server mora biti pokrenut** (vidite slajd 4). Ako server nije pokrenut, dobićete ovakvu grešku u konzoli:



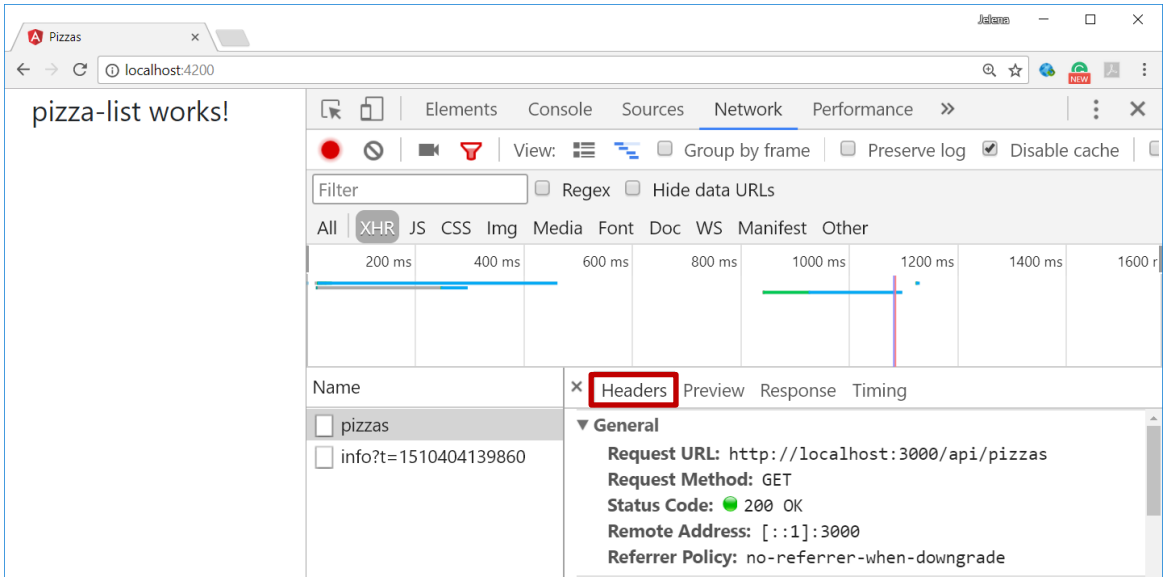

**Dobar način za debagovanje jeste da pratite šta se dešava u *Network:***



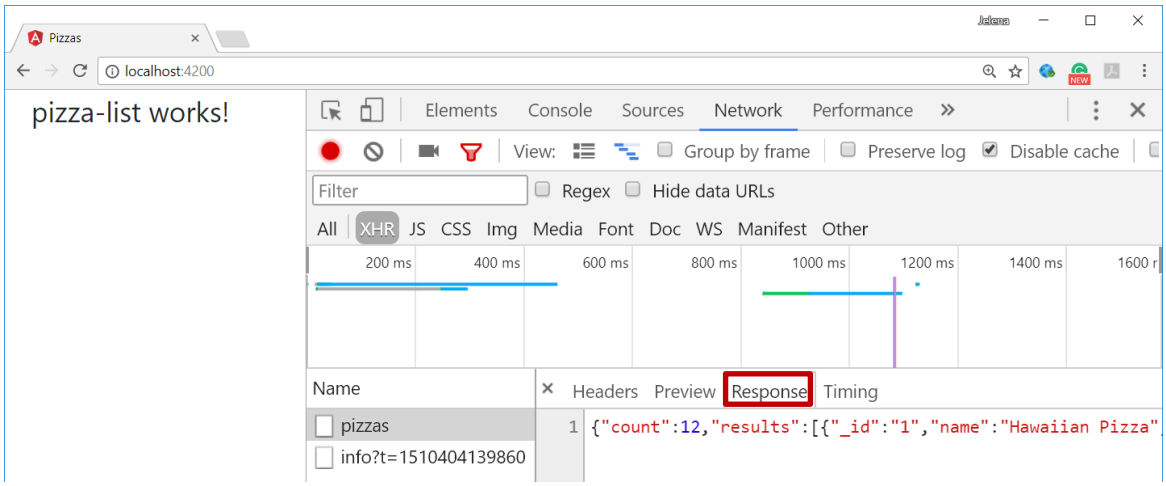

Ako kliknemo da pizzas, možemo videti koji zahtev smo slali i kako se izvršio:




Sledeće dve slike prikazuju uspešno poslat zahtev:




U **Response** možete videti odgovor dobijen sa servera:




11. Sada ćemo prikazati rezultate na templejtu komponente *PizzaListComponent*. U ovu svrhu:
    a. Napravljena je komponenta pizza-details u direktorijumu pizza: ng g component pizza/pizza-details
    b. U klasi komponente *PizzaDetailsComponent* je dodato polje *pizza* označeno *Input* dekoratorom:

```
export class PizzaDetailsComponent implements OnInit {
    @Input() pizza: Pizza = new Pizza();

    constructor() {}
```

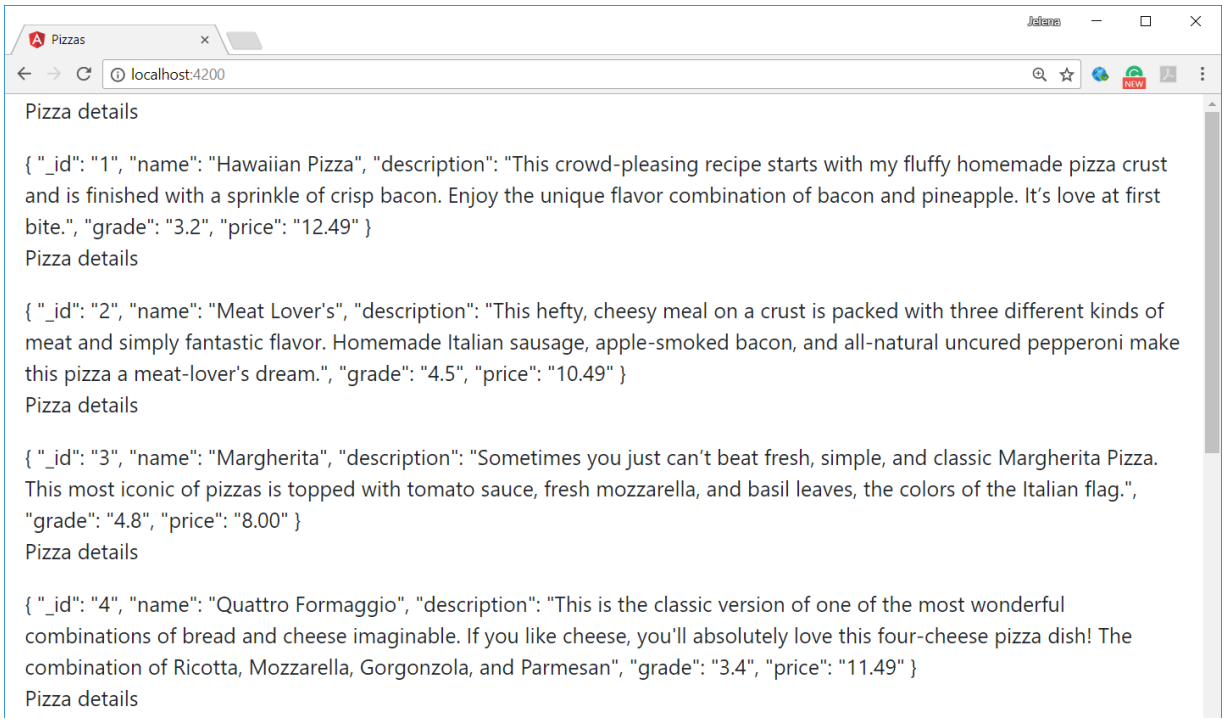
    c. Na templejtu komponente ćemo za sada samo prikazati *pizza* polje pomoću interpolacije. Deo | *json* je *pipe* koji omogućava da se objekat *pizza* prikaže kao JSON string.

```
<p>Pizza details</p>
{{pizza | json}}
```

d. U templejtu *PizzaListComponent* ćemo pomoću *NgFor* direktive za svaki element polja *pizzaList* prikazati po jedan *PizzaDetailsComponent*. Pri tome, na input *PizzaDetailsComponent* (koji se zove *pizza*) šaljemo odgovarajući element liste *pizzaList* (sadržan u iteratoro *pizza* iz **ngFor*):

```
<app-pizza-details *ngFor="let pizza of pizzaList" [pizza]="pizza"></app-pizza-details>
```

U ovom trenutku aplikacija izgleda ovako:




.

## Ostatak promena u primeru služe samo kao priprema za naredni zadatak i ne odnose se na upotrebu *HttpClient* servisa

12. Sređen je templejt *PizzaDetailsComponent*. Iskorišćena je *NgClass* direktiva da se ocena (*grade*) veća od 4 prikazuje zelenom bojom
13. Napravljena je komponenta *EditPizzaComponent* koja će sadržati formu za ažuriranje pice:
    a. Napravljeno je polje *pizza* (model podataka pice koja treba da se prikaže na formi)
    b. Napravljeno je polje *pizzaForm* (model forme)
    c. Injektovan je *FormBuilder* (dodat je kao parametar konstruktora)
    d. U konstruktoru je napravljena *pizzaForm* pomoću *FormBuider*a
    e. Da bismo mogli koristiti *reactive* forme, u modul aplikacije su ubačeni moduli *FormsModule* i *ReactiveFormsModule*
14. Napravljena je komponenta *NavbarComponent*
15. U aplikaciju je dodato rutiranje. U *AppModule*:
    a. Importovani su modul *RouterModule* i klasa *Routes* iz '@angular/router'
    b. Napravljena je *routes* konstanta koja sadrži pravila rutiranja
    c. *Router* modul je dodat u listu *imports* i njegovoj *forRoot* metodi je prosleđena konstanta *routes*

16. U templejtu *AppComponent* je iskorišćena <router-outlet> direktiva i linkovima u templejtu *NavbarComponent* su podešeni *routerLink* atributi. Sada smo u mogućnosti da putem navbara smenjujemo prikaze *PizzaListComponent* i *EditPizzaComponent*
17. U *PizzaDetailsComponent* je dodato dugme *Edit*. Klikom na njega se redirektujemo na *EditWineComponent*, pri čemu kao parametar šaljemo *id* pice na koju je kliknuto (okida se 3. pravilo riutiranja).