

TheDiningRoom

Uvod.....	2
Kreiranje aplikacije.....	2
Prepoznavanje statičkih i dinamičkih komponenti	3
Rutiranje.....	5
Prikaz restorana	6
Dobavljanje restorana sa back-end servera.....	6
Prikaz restorana	9
Query parametri	11
Filtriranje (po ceni).....	12
Paramteri rute (filtriranje po kuhinji)	13
Input i Output (paginacija).....	14
Rating	16
Eksterne biblioteke (NgBootstrap, Modal)	17

Uvod

U ovom dokumentu opisan je detaljan postupak izrade zadatka TheDiningRoom. Ovaj dokument može biti upotrebljen kao uputstvo za izradu zadatka ili kao podsednik za pojedine delove rešenja.

Kreiranje aplikacije

Na samom početku izrade, potrebno je napraviti novi Angular projekat komandom:

ng new TheDiningRoom

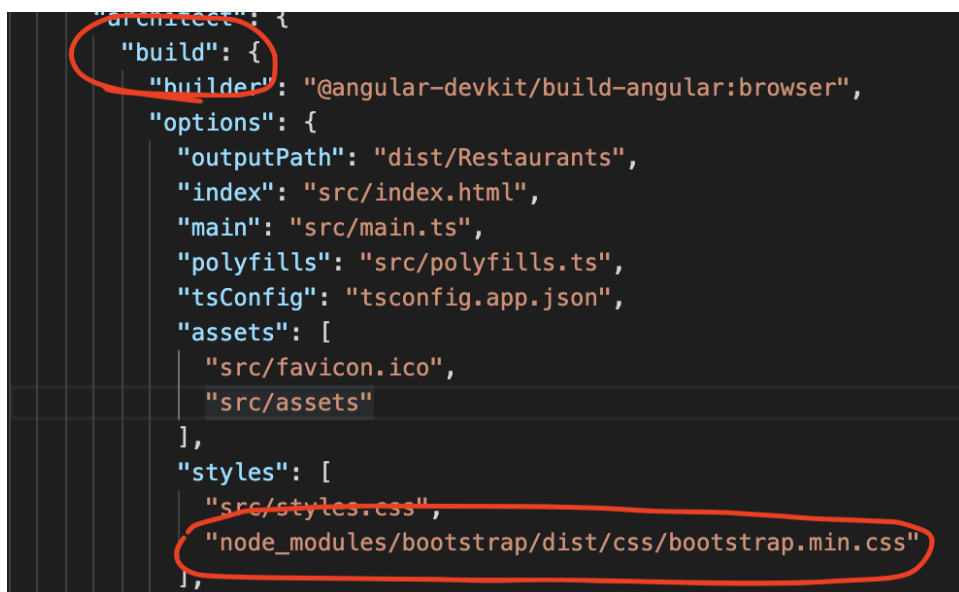
U konzoli dobijamo pitanje da li želimo da uključimo rutiranje u aplikaciju. Za izradu ovog zadatka odabrano je "da" zbog jednostavnosti, a preporučuje se samostalno kreiranje modula za rutiranje.

Takođe dobijamo izbor tehnologije za stilizovanje, gde je za ovaj zadatak izabran **css**.

U kreiranu aplikaciju potrebno je dodati **bootstrap**. Postoji više načina da se to uradi, od kojih je jedan komandom:

npm install bootstrap

Zatim dodavanjem putanje do bootstrap.css datoteke u odgovarajuće polje u **angular.json** datoteci.



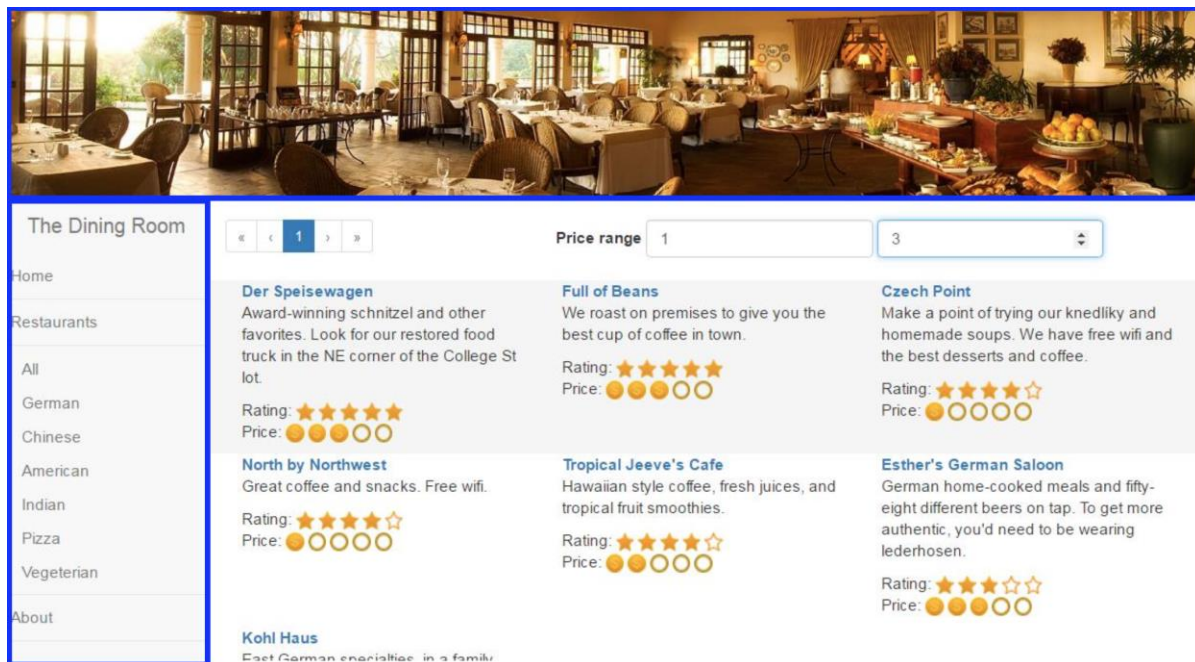
```
  "build": {
    "builder": "@angular-devkit/build-angular:browser",
    "options": {
      "outputPath": "dist/Restaurants",
      "index": "src/index.html",
      "main": "src/main.ts",
      "polyfills": "src/polyfills.ts",
      "tsConfig": "tsconfig.app.json",
      "assets": [
        "src/favicon.ico",
        "src/assets"
      ],
      "styles": [
        "src/styles.css",
        "node_modules/bootstrap/dist/css/bootstrap.min.css"
      ]
    }
  },
```

Ovo je dobar momenat da se ubace i ostale resursi potrebni za izradu aplikacije, kao što su slike. Direktorijum **images** iz zadatka u baciti u **assets** direktorijum projekta.

Preporučeno je pokrenuti aplikaciju komandom **ng serve** kako bismo proverili da je sve dobro podešeno. Aplikacija će se nalaziti na **localhost:4200**.

Prepoznavanje statičkih i dinamičkih komponenti

Prilikom izrade nove aplikacije, dobro je početi od dela koji se uvek nalazi na stranici, odnosno od *statičkog* dela aplikacije. U slučaju ovog zadatka, to je slika na vrhu i komponenta za navigaciju na levom delu stranice.



Pravimo komponentu za navigaciju, koju ćemo nazvati *SideBar*, komandom:

ng generate component core/side-bar

ili skraćeno: **ng g c** code/side-bar

Komponentu smeštamo u direktoriju core, gde bi trebalo da se nalaze statički i osnovni delovi aplikacije.

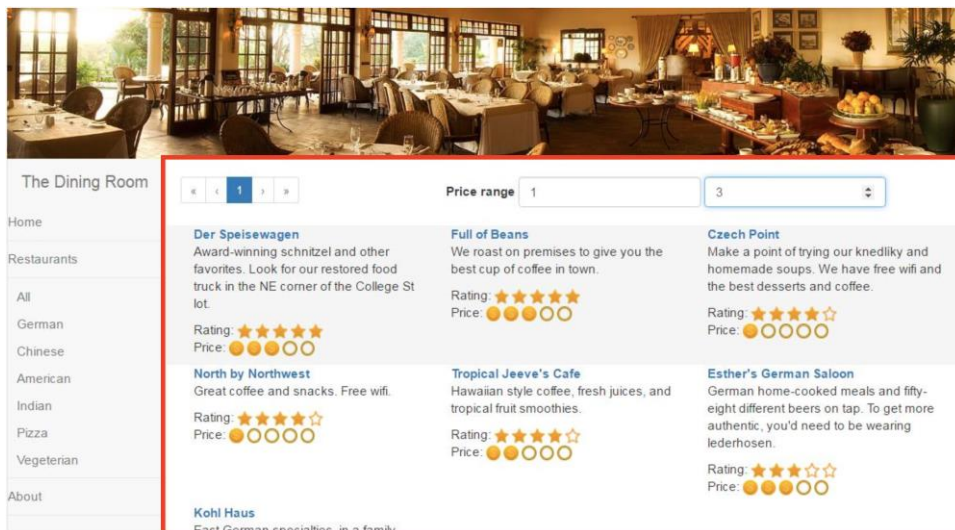
U *app-component.html* pravimo osnovni izgled aplikacije.

```

1  <div class="container-fluid">
2    <div class="row">
3      
4    </div>
5
6    <div class="row">
7      <div class="col-2">
8        <app-side-bar></app-side-bar>
9      </div>
10     <div class="col-10">
11       <!-- SADRŽAJ -->
12     </div>
13   </div>
14 </div>

```

Pored statičkih komponenti, na stranici imamo i dinamičke komponente. Ovo su komponente koje se dinamički prikazuju na stranici u zavisnosti od upotrebe korisnika. U ovom zadatku, imamo tri glavne dinamičke komponente: *Home*, *About* i *Restaurants*.



Home i *About* možemo nazvati osnovnim delovima stranice, i staviti u *core* direktorijum. *Restaurants* komponenta predstavlja glavnu funkcionalnost stranice i time zasluži da se nalazi direktno u *app* direktorijumu. Kreiramo komponente komandama:

```
ng g c core/home
```

```
ng g c core/about
```

```
ng g c restaurants
```

Rutiranje

Rutiranje služi kako bismo dinamički prikazali sadržaj stranice u zavisnosti od rute na kojoj se nalazimo. Ruta je deo URL-a koji se nalazi nakon domena (u ovom slučaju nakon *localhost:4200*)

Potrebno je napraviti po jednu rutu za svaku dinamičku komponentu: *Home*, *About* i *Restaurants*. U *app-routing.module.ts* popunjavamo listu ruta objektima koji povezuju rutu sa komponentom koja treba na njoj da se prikaže.

```
const routes: Routes = [  
  { path: "home", component: HomeComponent },  
  { path: "about", component: AboutComponent },  
  { path: "restaurants", component: RestaurantsComponent },  
  { path: "", redirectTo: "/home", pathMatch: "full" }  
];
```

Poslednja ruta služi kako bi prilikom odlaska na *localhost:4200* korisnik bio automatski prebačen na *localhost:4200/home*.

U *app-component.html* postavljamo tag *router-outlet* na čijem mestu će biti postavljena komponenta na osnovu trenutne rute.

```
<div class="container">  
  <div class="row">  
      
  </div>  
  <div class="row">  
    <div class="col-2">  
      <app-side-bar></app-side-bar>  
    </div>  
    <div class="col-10">  
      <router-outlet></router-outlet>  
    </div>  
  </div>  
</div>
```

U ovom momentu možemo proveriti funkcionalnost rutiranja odlaskom na odgovarajuću rutu, očekujući da se na stranici prikaže komponenta sa te rute.

Implementiramo *SideBar* komponentu tako da klikom na odgovarajuće dugme navigacije menjamo rutu. Za ovo koristimo *routerLink* atribut. Za sada preskačemo dugmadi za filtriranje po kuhinjama, koje ćemo dodati kada implementiramo tu funkcionalnost.

```

<h2 class="ml-5">The Dining Room</h2>
<ul class="nav flex-column">
  <li class="nav-item">
    <a class="nav-link" [routerLink]="['/home']">Home</a>
  </li>
  <hr>
  <li class="nav-item">
    <a class="nav-link" [routerLink]="['/restaurants']">Restaurants</a>
  </li>
  <hr>
  <li class="nav-item">
    <a class="nav-link" [routerLink]="['/about']">About</a>
  </li>
</ul>

```

Ovo se može postići i osnovim *a* tagovima. Na slici je prikazana implementacija sa nekoliko *bootstrap* klasa za lepši prikaz.

Prikaz restorana

Sledeći korak u izradi aplikacije je napraviti osnovni prikaz restorana. Pre nego što ih prikažemo, potrebno je dobiti restorane.

Dobavljanje restorana sa *back-end* servera

Potrebno je pokrenuti *back-end* server. Da bi smo to uradili, u terminalu unutar direktorijuma servera unosimo komande:

npm install

npm start

U tekstu zadatke nalazi se *endpoint* za pristup restoranima sa *back-end* servera:

<http://localhost:3000/api/restaurants>. Preporučeno je ispitati *endpoint* tako što pošaljemo zahtev putem internet pretraživača. *Odgovor* koji dobijamo je u *JSON* formatu. Za lakše tumačenje podataka, možemo upotrebiti <https://jsonformatter.curiousconcept.com/> ili sličan alat.

Kako bismo poslali zahtev na *back-end* server i prihvatili odgovor unutar Angular aplikacije, potrebno je:

1. Napraviti model podataka
2. Napraviti metodu u servisu za slanje zahteva
3. Pozvati metodu iz servisa unutar komponente u kojoj su nam potrebni traženi podaci

U kontekstu našeg zadatka, model je skup klasa koje oponašaju strukturu podataka koje dobijamo kao odgovor sa *back-end* server-a. Klase kreiramo u direktorijumu *model*. Za dobavljanje restorana, pravimo dve klase:

restaurant.model.ts

```
export class Restaurant {
  name: string;
  cuisine: string;
  _id: number;
  description: string;
  location: string;
  price: number;
  rating: number;

  constructor(obj?:any) {
    this.name = obj && obj.name || '';
    this.cuisine = obj && obj.cuisine || '';
    this._id = obj && obj._id || 0;
    this.description = obj && obj.description || '';
    this.location = obj && obj.location || '';
    this.price = obj && obj.price || 0;
    this.rating = obj && obj.rating || 0;
  }
}
```

restaurant-list.model.ts

```
import { Restaurant } from "../restaurant.model";

export class RestaurantList {
  count: number;
  results: Restaurant[];

  constructor(obj?:any) {
    this.count = obj && obj.count || 0;
    this.results = obj && obj.results && obj.results.map((elem: any) => new Restaurant(elem)) || [];
  }
}
```

Cilj modela je da od objekata tipa *any* dobijemo tipizirane objekte, čime se štitimo od eventualnih promena na *back-end* strani aplikacije, kao i od nevalidnih vrednosti. Takođe, rukovanje tipiziranim podacima je lakše i sigurnije tokom izrade aplikacije. Ovo postizemo konstruktorom koji prima objekat tipa *any* i vraća instancu klase sa popunjenim atributima. Treba napomenuti da ukoliko neki atribut klase iz našeg modela sadrži drugu klasu iz modela, potrebno je pozvati i taj konstruktor. Za slučaj liste podataka neke klase, svaki element liste prosleđujemo kroz konstruktor, upotrebom funkcije *map*.

Sledeći korak je pravljenje metode u servisu za slanje zahteva na željeni *endpoint*. Pošto je ovo prvi zahtev koji šaljemo u zadatku, potrebno je prvo napraviti novi servis:

ng g service services/restaurant (skraćeno **ng g s**)

Servise odvajamo u poseban direktorijum, i nazivamo po entitetima kojima rukuju, u ovom slučaju *RestaurantService*.

Da bi smo poslali *http* zahtev, potrebno je uvesti *HttpClientModule* u aplikaciju. To radimo kroz *app-module.ts*.

```
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [
    AppComponent,
    SideBarComponent,
    HomeComponent,
    AboutComponent,
    RestaurantsComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
  ],
})
```

Unutar servisa, **injektujemo** *HttpClient*.

restaurant.service.ts

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

Zatim pravimo funkciju koja će da šalje zahtev. Poželjno je osnovni deo URL-a sačuvati u zasebnu promeljivu, jer se uglavnom ponavlja više puta unutar servisa.

```
const baseUrl = "http://localhost:3000/api/"

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

  constructor(private http: HttpClient) { }

  getAll(): Observable<RestaurantList> {
    return this.http.get(`${baseUrl}restaurants`).pipe(map((data: any) => {
      return new RestaurantList(data);
    }));
  }
}
```

Zahtev šaljemo *HttpClient.get* metodom, čiji je rezultat *Observable* nad podacima koji dolaze sa *back-end server-a*. Ovo je mehanizam da se sačeka odgovor na zahtev, koji traje nedefinisani vremenski period. Dobijeni podaci su u tipa *any*. Upotrebom *pipe* i *map* funkcija pretvaramo te podatke u podatke tipova iz našeg modela, tako što ih proseđujemo odgovarajućem konstruktoru. Kao rezultat naša funkcija vraća objekat tipa *Observable<RestaurantList>*.

U *restaurants-component.ts* potrebno je injektovati servis, i zatim unutar *ngOnInit* metode pozvati metodu *getAll* iz servisa.

```
constructor(private service: RestaurantService) { }

ngOnInit(): void {
  this.service.getAll().subscribe((restaurants: RestaurantList) => {
    console.log(restaurants);
  })
}
```

Kako je rezultat *getAll* metode *Observable*, da bismo prihvatili rezultate potrebno je iskoristiti *subscribe* metodu i njoj proslediti funkciju koja će da obradi dobijenu vrednost. Poželjno je da za početak da obrada bude ispis u konzolu, kako bismo mogli da verifikujemo funkcionalnost dobavljanja restorana u konzoli unutar pretraživača. (obavezno navigirati na */restaurants*)

Prikaz restorana

Pravimo novu komponentu, *RestaurantItem*, za prikaz jednog restorana, koju ćemo prikazati po jednom za svaki restoran.

ng g c restaurants/restaurant-item

Smeštamo je unutar *Restaurants* komponente, jer je njen sastavni deo.

Za svaki restoran instanciramo po jednu komponentu *RestaurantItem*. Pre toga smeštamo restorane koje nam je vratila metoda *getAll* u promenljivu *restaurants*.

```
restaurants: RestaurantList = new RestaurantList();

constructor(private service: RestaurantService) { }

ngOnInit(): void {
  this.service.getAll().subscribe((restaurants: RestaurantList) => {
    this.restaurants = restaurants;
  })
}
```

Pomoću **ngFor* petlje, prolaskom kroz listu restorana koja se nalazi u atributu *restaurants.results*, u tagu unutar kojeg se nalazi tag *app-restaurant-item*.

restaurants.component.ts

```
<div class="row">
  <div class="col-4" *ngFor="let item of restaurants.results">
    <app-restaurant-item></app-restaurant-item>
  </div>
</div>
```

Svakoj instanci *RestaurantItem* komponente potrebno je proslediti po restoran. Da bismo to postigli, u *restaurant-item.component.ts* navodimo atribut klase *restaurant* sa anotacijom *@Input()*.

```
@Input()  
restaurant: Restaurant = new Restaurant();
```

Zatim prilikom instanciranja komponente u *restaurants.component.html* prosеđujemo restoran:

```
<div class="row">  
  <div class="col-4" *ngFor="let item of restaurants.results">  
    <app-restaurant-item [restaurant]="item"></app-restaurant-item>  
  </div>  
</div>
```

Poslednji korak je da u *restaurant-item.component.html* interpolacijom prikazemo informacije o restoranu:

```
<h4>{{restaurant.name}}</h4>  
<div class="row">  
  <div class="col">  
    {{restaurant.description}}  
  </div>  
</div>  
<div class="row">  
  <div class="col">  
    Rating: {{restaurant.rating}}  
  </div>  
</div>  
<div class="row">  
  <div class="col">  
    Price: {{restaurant.price}}  
  </div>  
</div>
```

Query parametri

U sklopu GET zahteva (u ređim slučajevima i ostalih HTTP metoda) možemo da prosledimo parametre putem URL-a, koji se koriste za parametrizovane upite nad kolekcijom koju dobavljamo. Potrebno je da *getAll* metodu našeg *RestaurantService*-a prilagodimo tako da prihvata i šalje parametre navedene u zadatku.

```
getAll(params?:any): Observable<RestaurantList> {
  let queryParams = {};

  if(params) {
    queryParams = {
      params: new HttpParams()
        .set('page', params.page || "")
        .set('pageSize', params.pageSize || "")
        .set('filter', params.filter && JSON.stringify(params.filter) || "")
    }
  }

  return this.http.get(baseUrl, queryParams).pipe(map((data: any) => {
    return new RestaurantList(data);
  })))
}
```

Parametre čuvamo i prosleđujemo iz komponente *RestaurantsComponent*, jer ona koristi ovu metodu servisa, i čuva listu restorana.

```
params = {
  page: 1,
  pageSize: 10,
  sort: '',
  sortDirection: '',
  filter: {
    cuisine: '',
    priceFrom: 1,
    priceTo: 5
  }
}
```

Poziv `getAll` metode izdvajamo u odvojenu funkciju, zato što će biti korišćena više puta.

```
getRestaurants() {  
  this.service.getAll(this.params).subscribe((restaurants: RestaurantList) => {  
    this.restaurants = restaurants;  
  })  
}
```

Parametri `sort` i `sortDirection` se ne koriste u zadatku, tako da mogu biti izostavljeni.

Ovako postavljeni parametri lako se mogu menjati iz različitih delova aplikacije, i dovoljno je pozvati opet metodu `getRestaurants()` iz `RestaurantsComponent` kako bismo osvežili prikaz restorana uz upotrebu filtriranja i paginacije.

Filtriranje (po ceni)

Kao što je već pomenuto, filtriranje radimo izmenom polja `params` u `RestaurantsComponent` i ponovnim pozivom metode `getRestaurants()`.

Počecemo sa implementacijom filtriranja po ceni. Potrebna su nam dva `input` polja koja menjaju vrednosti `params.filter.priceFrom` i `params.filter.priceTo`. Kako bismo povezali polja sa vrednostima, upotrebićemo `ReactiveForms`.

restaurants.component.html

```
<div class="col-8">  
  <form>  
    <div class="row">  
      <div class="col-2"></div>  
      Price range:  
      <div class="col-4">  
        <input type="number" class="form-control"  
          [formControl]="priceFromControl" (change)="onPriceChanged()">  
      </div>  
      <div class="col-4">  
        <input type="number" class="form-control"  
          [formControl]="priceToControl" (change)="onPriceChanged()">  
      </div>  
    </div>  
  </form>  
</div>
```

U primeru je upotrebljena `bootstrap` klasa `form-control` za lepši prikaz. U klasi komponente potrebno je da dodamo dva atributa tipa `FormControl` sa kojima će polja forme biti povezana:

restaurants.component.ts

```
priceFromControl = new FormControl(1)  
priceToControl = new FormControl(5)
```

Kako bismo koristili *ReactiveForms* potrebno je da odradimo **import *ReactiveFormsModule***-a unutar *app.module.ts*.

```
imports: [  
  BrowserModule,  
  AppRoutingModule,  
  HttpClientModule,  
  FormsModule  
],
```

Definisaćemo metodu u komponenti kojom reagujemo na promene cena u poljima forme, tako što podesimo vrednosti odgovarajućih atributa *params.filter* objekta na vrednosti iz forme. Nakon toga opet pozivamo metodu *getRestaurants* da bismo osvežili prikaz restorana sa novim filterom.

restaurants.component.ts

```
onPriceChanged(): void {  
  this.params.filter.priceFrom = this.priceFromControl.value || 1;  
  this.params.filter.priceTo = this.priceToControl.value || 1;  
  this.getRestaurants();  
}
```

Paramteri rute (filtriranje po kuhinji)

Zadatak zahteva da na klik naziva kuhinje u navigaciji prikazemo restorane te kuhinje. Da bismo to postigli, potrebno je da prosledimo informaciju o traženoj kuhinji iz *SideBarComponent* u *RestaurantsComponent*. Najelegantniji način da to postignemo je putem parametara rute. Parametri rute pišu se u definiciji rute, pomoću znaka ":". Izmenićemo postojeću rutu za *RestaurantsComponent* tako da prihvata parametar *cuisine*.

```
{ path: 'restaurants/:cuisine', component: RestaurantsComponent },
```

Potrebno je u *SideBar* komponenti prikazati nazive kuhinja. Ovo ćemo postići tako što definišemo listu kuhinja u *side-bar.component.ts* i pomoću **ngFor* potelje prikazati nazive. Na *a* tagove stavljamo *routerLink* atribut tako da šalje naziv kuhinje na mestu parametra rute.

side-bar.component.ts

```
export class SideBarComponent implements OnInit {  
  cuisines = ['All', 'German', 'Chinese', 'American', 'Indian', 'Pizza', 'Vegetarian']
```

side-bar-component.html

```
<div class="row" *ngFor="let cuisine of cuisines">
  <div class="col">
    <a [routerLink]="['/restaurants/', cuisine]">{{cuisine}}</a>
  </div>
</div>
```

Prosleđenu informaciju o kuhinji potrebno je preuzeti unutar *Restaurants* komponente, a zatim izmeniti parametar *params.filter.cuisine* na novu vrednosti i pozvati ponovo *getRestaurants*. Specijalni slučaj predstavlja vrednost parametra "all", kada želimo da dobijemo sve restorana, odnosno *params.filter.cuisine* treba postaviti na vrednost "".

Pošto je moguć slučaj da se parametar rute menja, a da stranica sve vreme prikazuje *Restaurants* komponentu, nije dovoljno jednom pokupiti vrednost parametra rute (pomoću *route.snapshot*). Da bismo ispratili svaku promenu parametra rute, koristimo *Observable route.params* na koju moramo da se *subscribe*-ujemo. Prvobitno injektujemo *ActivatedRoute* u konstruktoru komponente (pod nazivom *route*).

```
ngOnInit(): void {
  this.route.params.subscribe((params: any) => {
    let cuisine = params['cuisine'];
    if (cuisine == 'All') cuisine = '';
    this.params.filter.cuisine = cuisine;
    this.getRestaurants();
  })
}
```

Input i Output (paginacija)

Paginaciju u ovom zadatku implementiramo u zasebnoj komponenti.

ng g c restaurants/pagination

Paginaciju postićemo *query* parametrima *page* i *pageSize*, koji su implementirani u skleciji za *query* parametre. Potrebno je da kroz komponentu menjamo vrednost *page* parametra i pozovemo *getRestaurants* da bismo upotpunili funkcionalnost.

Osnovni zadatak komponente za paginaciju je da omogući unos promene stranice prikazanih restorana. Ovo možemo postići sa dva dugmeta: *Next* i *Previous*, na koje ćemo nadograditi direktan izbor stranice.

Potrebna nam je informacija o trenutnoj stranici:

```
@Input()
page: number = 1;
```

I potreban nam je događaj promene strane, koji je ujedno i *Output* ove komponente:

```
@Output()
pageChange: EventEmitter<number> = new EventEmitter();
```

Na klik dugmeta šaljemo događaj sa vrednosti broja nove strane koju treba prikazati. Za dugme *Previous* to će biti *page-1*, a za dugme *Next* - *page+1*. Napravićemo funkciju u *pagination.component.ts* koja šalje događaj i povezati je sa klikom dugmadi.

```
onButtonClick(newPage: number) {
  this.pageChange.emit(newPage)
}
```

```
<button class="btn btn-primary" (click)="onButtonClick(page-1)">Previous</button>
<button class="btn btn-primary" (click)="onButtonClick(page+1)">Next</button>
```

U roditeljskoj komponenti, odnosno *Restaurants* komponenti, prosledićemo vrednosti trenutne stranice i povezati se sa događajem (*Output*) promene strane. Na događaj se povezujemo funkcijom, koja u ovom slučaju ima zadatak da postavi vrednost parametra *params.page* na novu vrednost stranice i pozove *getRestaurants* da bismo osvežili prikaz.

restaurants.component.html

```
<div class="col-4">
  <app-pagination [page]="params.page" (pageChange)="onPageChanged($event)"></app-pagination>
</div>
```

restaurants.component.ts

```
onPageChanged(newPage: number) {
  this.params.page = newPage;
  this.getRestaurants();
}
```

Dosadašnjim postupkom smo implementirali minimalnu funkcionalnost paginacije. Bilo bi dobro ograničiti moguće stranice na koje korisnik može da dospe, tako što onemogućimo klik na dugmadi *Previous* i *Next* kada se korisnik nađe na prvoj, odnosno poslednjoj stranici. Da bismo znali koja je

poslednja stranica, potrebna nam je informacija o ukupnom broju restorana (*restaurants.count*) i o veličini stranice (*params.pageSize*).

pagination.component.ts

```
@Input()
pageSize: number = 10;

@Input()
collectionSize: number = 0;
```

restaurants.component.ts

```
<div class="col-4">
  <app-pagination
    [page]="params.page"
    [pageSize]="params.pageSize"
    [collectionSize]="restaurants.count"
    (pageChange)="onPageChanged($event)"></app-pagination>
</div>
```

Zatim postavljamo vrednost atributa *disabled* na rezultat uslova o prvoj tj. poslednoj strani:

```
<button
  [disabled]="page <= 1"
  class="btn btn-primary"
  (click)="onButtonClick(page-1)">Previous</button>
<button
  [disabled]="page >= collectionSize/pageSize"
  class="btn btn-primary"
  (click)="onButtonClick(page+1)">Next</button>
```

Finalna implementacija paginacije data je u rešenju. Prikazana su dugmad sa brojevima strane, i primenjene su *bootstrap* klase za lepši prikaz. Kako bismo prikazali dugmad, napravljen je niz mogućih vrednosti strana na osnovu ukupnog broja restorana i veličine stranice. Pošto je ovaj niz potrebno ažurirati na svaku promenu ovih vrednosti, upotrebljena je *ngOnChanges Angular* metoda, koja se pokreće na svaku promenu na *Input* delu komponente. Ovu metodu koristimo sa rezervom, jer može da izazove neželjene efekte ukoliko se *Input* neke komponente često menja. Nad kreiranim nizom strana primenjena je **ngFor* petlja koja pravi po dugme za svaku stranu.

Rating

Implementacija komponente za prikaz ocene i cene restorana data je u rešenju. Predstavlja vežbu za razmišljanje i snalaženje. Dato rešenje nije jedini način da se napravi i preporučeno je samostalan pokušaj kreiranja ove komponente.

Eksterne biblioteke (*NgBootstrap*, *Modal*)

Za gotovo svaki programski jezik i tehnologiju postoji mnoštvo biblioteka. Biblioteke su delovi koda (paketi, moduli...) koje je neko napravio i podelio (ili naplaćuje), koja sadrži neka gotova rešenja konkretnih problema. Dobre biblioteke se prave tako da budu lako primenljive, bez potrebe za razumevanjem koda koji se nalazi unutar njih. Za to nam posebno služi dokumentacija biblioteke, gde se uobičajeno nalazi sve što je potrebno za njenu upotrebu.

U zadatku se traži upotreba *NgBootstrap* biblioteke, odnosno komponente *Modal* unutar nje. *Angular* biblioteke dodajemo u projekat komandom:

ng add

odnosno za *NgBootstrap*:

ng add @ng-bootstrap/ng-bootstrap

Instrukcije za instalaciju biblioteke obično se nalaze na samom početku njene dokumentacije.

Rezultat prethodne komande je instaliran *npm* paket *NgBootstrap* i import *NgbModule* unutar *app.module.ts*. Kao sporedni efekat, ova biblioteka dodaje i *bootstrap* u projekat.

Ovaj deo zadatka namenjen je samostalnom istraživanju. Predstavlja najnapredniji zadatak i preporučeno je posvetiti mu se tek nakon kompletnog razumevanja implementacije ostatka zadatka.

Ukoliko niste u stanju da implementirati modalni dijalog, napravite novu stranicu (rutu) za prikaz menija restorana.

Prikaz menija

Nakon implementacije modalnog dijaloga, sadržaj dijaloga popunjavamo na osnovu podataka sa *back-end server-a* koje još nismo dobavili. U ovom dokumentu biće navedeni samo koraci za implementaciju ovog dela zadatka:

1. Proširiti klasu *Restaurant* da uključuje i polje *days*
2. Napraviti model za dobavljanje menije za odgovarajućeg *endpoint-a*
3. Napraviti metodu u servisu za slanje zahteva na taj *endpoint*
4. Pozvati metodu u *Menu* komponenti
5. U *html* delu *Menu* komponente napraviti prikaz radnog vremena i adrese
6. Ukoliko ne postoji vrednost za neki dan, ne prikazati ništa za taj dan
7. Napraviti tabelu za prikaz stavki menija