Vežba 3

U vežbi 3 pravimo node.js express web server (*api servis*) koji se povezuje na mysql i definišemo REST API za entitete u bazi. Dovršićemo front-end interfejs čime zaokružujemo adminsku aplikaciju.

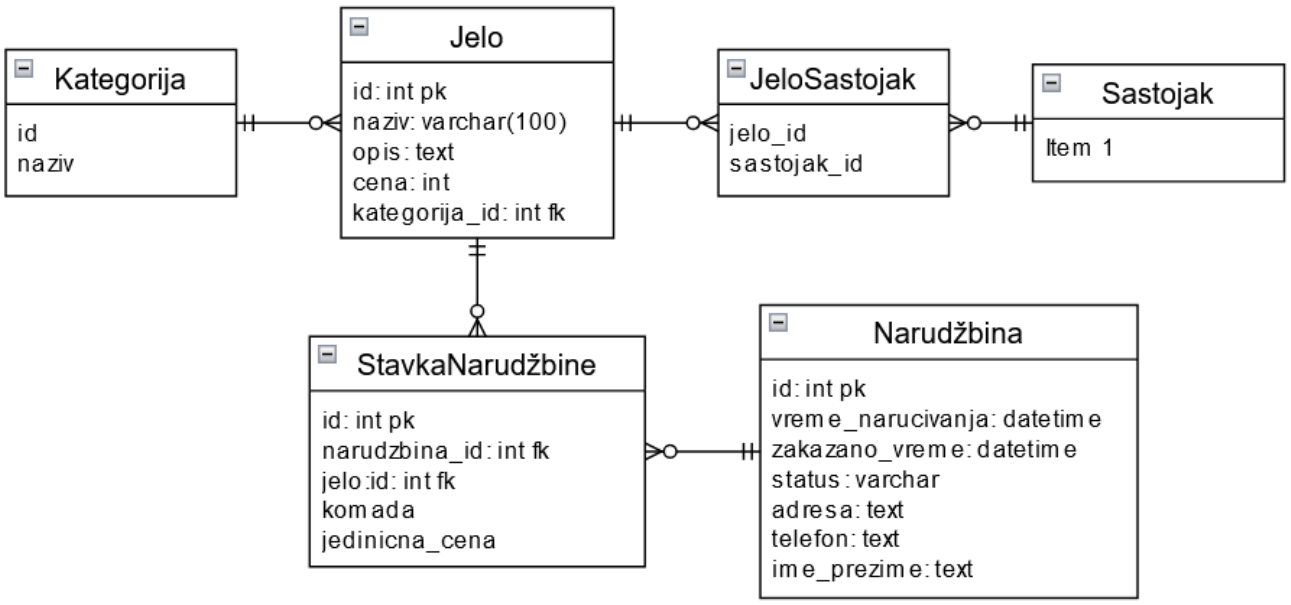
Rok za završetak ove vežbe i postavljanje na repozitorijum je 3.12.2023 u 23:59

1 Baza podataka

Za ovu i naredne vežbe trebaće vam MySQL ili MariaDB. Možete da instalirate zasebno a možete i kroz XAMPP.

1 XAMPP

1. Sa https://www.apachefriends.org/download.html možete da preuzmete XAMPP. Instalacija ima nekoliko koraka, gde nema šta puno da se dira i bira.
2. Pokrenite XAMPP Control Panel i kliknite na Start pored Apache i MySQL da biste pokrenuli ta dva servisa.
3. Apache nam treba jer uz XAMPP dolazi i phpmyadmin, softver za pristup bazi. U browseru otvorite localhost/phpmyadmin
4. U levom delu nalazi se spisak baza u pokrenutom MySQL serveru. Kliknite na New i unesite ime nove baze, izaberite utf8_mb64_general_ci (poslednja opcija u listi) za collation kako bi default enkoding za tabele koje budete pravili bio utf8.
5. U levom delu kliknite na ime nove baze, trebalo bi da imate opciju New table ili Create table, i tu možete da počnete pravljenje nove tabele. Isto postoji i ako se klikne na naziv baze - u glavnom delu prikazuje se spisak tabela (inicijalno je prazno, naravno) i imate opciju za kreiranje nove tabele.
6. Zatim upišite imena svih kolona, izaberite tipove, označite A_I (auto increment) na ID koloni i, ako ne uradi to automatski, označite i primary key.
7. Kada se tabela napravi biće prikazana i u listi levo. Klik na tabelu otvara Browse ekran gde se prikazuju podaci u tabeli. Podatke u tom prikazu-tabeli možete da menjate i brišete. Podatke možete i da unesete, na Insert tabu.
8. I tabela i cela baza mogu da se eksportuju i importuju. Tipičan format je .sql fajl koji u sebi sadrži generisane SQL upite koji rekreiraju bazu podataka u stanju u kom je eksportovana (upiti za kreiranje tabela i insertovanje podataka u tabele).
9. Napravite bazu skript_j, odnosno po vašem nazivu projekta.
10. Nemojte da pravite tabele. Tabele ćemo da generišemo iz projekta, na osnovu modela koje budemo napisali.



2 Konzolni pristup bazi

Ako nemate phpmyadmin, svakako imate konzolni pristup bazi.

1. Pronađite gde je folder u kome je instaliran mysql i gde se nalazi mysql.exe. Za XAMPP to je na C:\xampp\mysql\
2. Otvorite konzolu u tom folderu
    a. Ako je mysql podešen u system path onda je svejedno u kom ste folderu. Ovo je uglavnom slučaj za linux mašine.
3. Pokrenite mysql konzolni interfejs ovako:
    ```
    mysql.exe -u root -p
    ```
    Kada traži unos passworda unesite. Ako ne znate koji je, default user je uglavnom root a password je prazan string ili root. Pritisnite enter, najverovatnije će da prođe.
4. Sada ste u mysql konzoli, vidite drugačiji prompt. Napravite novu bazu sa
    ```
    create database skriptj;
    ```
    ; na kraju je obavezna, to je oznaka za kraj naredbe, inače prelazi u sledeći red i čeka nastavak sql naredbe
5. Proverite da li je baza napravljena sa
    ```
    show databases;
    ```
6. Uđite u bazu sa
    ```
    use skriptj;
    ```
    Svi upiti nadalje će biti izvršeni u ovoj bazi
7. Napravite novu tabelu sa
    ```
    create table jelo …
    ```
    Pogledajte mysql dokumentaciju, w3schools ima korektnu dokumentaciju
8. Napunite tabelu sa nekim inicijalnim sadržajem sa
    ```
    insert into jelo (naziv, opis, cena, kategorija) values
    ('vegeterijana', 'masna', 1200, 'pica'), ('kobasica',
    'posna', 300, 'rostilj');
    ```
    itd…
9. Prikažite sadržaj tabele sa

```
select * from jelo;
```

Ovo je osnovno za pristup i rad sa bazom.

2 Postavljanje REST servisa

Sada ćemo napraviti web servis koji ima REST API - sve potrebne rute za pristup podacima. Ovaj servis pravimo kao zasebnu node.js aplikaciju. App servis (vaš GUI aplikacije) će slati upite ovom servisu za dohvatanje i izmenu podataka. Ovaj servis sadrži sve CRUD rute.

1. Napravite novi projektni folder **api_servis** i u njemu skript **app.js**
2. Otvorite konzolu u projektnom folderu (node1) i inicijalizujte projekat sa **npm init**
3. Instalirajte express - **npm install express**

U app.js napravite osnovni kostur express aplikacije

1. Uključite paket express
   ```
   const express = require('express');
   ```
2. Dohvatite instancu
   ```
   const app = express();
   ```
3. Registrujte home rutu koja vraća jednostavnu hello poruku
   ```
   app.get('/', (req, res) => {
       res.send('Hello from REST API service'));
   });
   ```
4. Otvorite konekciju na portu 9000 - obratite pažnju na ovo, imaćete pokrenuta dva web servisa istovremeno, ne smeju da imaju isti port.
   ```
   app.listen(9000);
   ```
5. Pokrenite aplikaciju u konzoli
   node app
6. Testirajte u browseru:
   http://localhost:9000
7. Trebalo bi da dobijete hello poruku

Grupisanje ruta u zasebne fajlove

8. Napravite folder *routes* - imaćemo nekoliko entiteta i za svakog entiteta po nekoliko ruta. Ima smisla da grupišemo rute entiteta po fajlovima, kako bi kod bio uredniji.
9. U routes/ napravite js fajlove za svaki entitet (vidi šemu). Ne pravimo za međutabele.
   a. jelo.js
   b. kategorija.js
   c. narudzbina.js
   d. sastojak.js

10. Pre app.listen(9000) umetnite kod kojim se rute uključuju u aplikaciju

```
const jeloRoutes = require("./routes/jelo.js");
app.use("/jelo", jeloRoutes);
//... dodajte ostale
```

    a. Na ovaj način sve rute koje budemo definisali u jelo.js biće ubačene u aplikaciju sa prefiksom /jelo/
    b. U ./routes tačka označava tekući folder, putanja je relativna u odnosu na app.js

Definisanje ruta

11. Otvorite jelo.js
12. Prvo ćemo da učitamo i dohvatimo express instancu unutar ovog modula i njen router, i da
    a. const express = require("express");
    b. const route = express.Router();
13. Zatim dodajemo json i urlencoded middleware, koji će da prepakuju json string u objekat i počiste url od specijalnih znakova, po potrebi
    a. route.use(express.json());
    b. route.use(express.urlencoded({extended:true}));
14. Činimo route objekat vidljivim spolja, tj. u app.js koji koristi ovaj modul (zamislite ovo kao public access specifier u objektu)
    a. module.exports = route;
15. I sada definišemo sve potrebne API rute.
    a. Potrebne su nam sledeće rute
        i. GET /jelo
        ii. GET /jelo/:id
        iii. POST /jelo
        iv. PUT /jelo/:id
        v. DELETE /jelo/:id
    b. GET, POST, PUT, DELETE su http "glagoli" - vrste requestova koje mogu da se pošalju. REST koristi ovo da razdvoji requestove za čitanje, unos, izmenu, brisanje podataka.
    c. Pošto rute pišemo unutar modula, a modul smo u app.js ubacili sa prefiksom /jelo onda ćemo imati zapravo ovakve rute:
        i. GET /
        ii. GET /:id
        iii. POST /
        iv. PUT /:id
        v. DELETE /:id

Primeri ruta

16. Napravićemo prvo GET rutu koja će da vraća sve zapise iz baze

```
route.get("/", async (req, res) => {
   try{
     return res.json("sva jela");
   }catch(err){
     console.log(err);
```

```
      res.status(500).json({ error: "Greska", data: err });
   }
});
```

a. Kod za pristup bazi dodaćemo kasnije.
b. Funkciju proglašavamo za *async* jer unutar nje očekujemo slanje upita u bazu, koji je I/O operacija i kao takva je asinhrona, tj. implementirana kroz Promise. Da ne bismo pisali ceo lanac sa .then() primenićemo async-await sintaksu pomoću koje možemo da asinhroni kod pišemo u sinhronom stilu. Videćemo ovo kasnije, kada ubacimo pozive kada bazi.

17. Po istom principu pravimo i ostale rute

```
route.get("/:id", async (req, res) => {
   try{
      return res.json("jelo čiji je id=" + req.params.id);
   }catch(err){
      console.log(err);
      res.status(500).json({ error: "Greska", data: err });
   }
});


route.post("/", async (req, res) => {
   try{
      return res.json("unos novog jela čiji su podaci
                           u req.body");
   }catch(err){
      console.log(err);
      res.status(500).json({ error: "Greska", data: err });
   }
});


route.put("/:id", async (req, res) => {
   try{
      return res.json("izmena podataka jela čiji
          je id="+req.params.id+" a podaci su u req.body");
   }catch(err){
      console.log(err);
      res.status(500).json({ error: "Greska", data: err });
   }
});


route.delete("/:id", async (req, res) => {
   try{
      return res.json(req.params.id);  //id obrisanog
   }catch(err){
      console.log(err);
      res.status(500).json({ error: "Greska", data: err });
```

```
    }
});
```

18. Ovo možete da iskoristite kao osnovu i za ostale entitete, tj. module sa rutama za ostale entitete.
19. Pokrenite servis sa **node app**

3 Testiranje REST servisa

Za proveravanje svih API ruta nije nam neophodna gotova klijentska aplikacija. Kada se servis pokrene rute su dostupne kroz standardne HTTP zahteve koje možemo da generiše i iz alata kao što je postman. Za Visual Studio Code postoji jednostavna ekstenzija ThunderClient

20. Otvorite Visual Studio Code, ako već niste
21. U Visual Studio Code, u levom sidebaru, kliknite na Extensions
22. Upišite thunder
23. Pronađite u listi Thunder Client i kliknite na install (pojaviće se verovatno prvi na listi)
24. Kada se instalacija završi dobićete novu ikonicu u levom sidebaru

25. Kliknite na to, otvoriće se Thunder Client aplikacija za testiranje API-ja i web servisa, koja vam omogućava da kreirate proizvoljne http zahteve i da vidite dobijene http odgovore.
26. Kliknite na New request
27. Izaberite GET, upišite http://localhost:9000/jelo i kliknite na Send
28. U desnom delu dobićete http response sa svim detaljima
29. Primetite da ispod GET postoji Query, Body itd. gde možete da podesite parametre/vrednosti koje se šalju u requestu. Ovo će nam trebati u nastavku, za testiranje API-ja
30. Probajte i za ostale rute
    a. GET http://localhost:9000/jelo/1
    b. POST http://localhost:9000/jelo
    c. PUT http://localhost:9000/jelo/1
    d. DELETE http://localhost:9000/jelo/1
31. Trebalo bi da dobijete odgovarajuće poruke.

4 Dodavanje Sequelize ORM u express aplikaciju

Naravno da možemo da koristimo i sirov pristup SQL-u, pisanje upita i fetchovanje rezultata, ali ćemo da iskoristimo Sequelize kao gotovo rešenje koje apstrahuje bazu podataka i omogućava nam da opišemo tabele u bazi klasama-modelima, i da pristupanje bazi bude kroz upotrebu objekata i pozivanje ugrađenih metoda. Zato nismo ni pravili baze na osnovu šeme, već ćemo da napravimo modele tabela i pomoću sequelize uslužnih alata generišemo tabele.
Još jedna prednost ORM sistema je što možete da lako migrirate sa jednom SQL na drugi, ako ORM ima podršku za oba.

1. Instalirajte drajver sa
   ```
   npm install mariadb
   npm install sequelize mariadb
   ```

2. Instalirajte sequelize-cli sa
   ```
   npm install -g sequelize-cli
   ```
   a. -g je za globalno
   b. Trebalo bi da u konzoli mozete da pozovete *sequelize*
   c. Ako *sequelize* neće da se pokrene (jer instalacija nije ubacila *sequelize* kao globalno dostupni servis, što na raf računarima verovatno ne može da uradi) onda možete da ga pokrenete direktno iz node_modules foldera ovako:
      `node node_modules/sequelize-cli/lib/sequelize`
   d. I u nastavku dopišete šta treba
      i. Npr. ako je naredba **sequelize db:seed** to će onda biti
         node node_modules/sequelize-cli/lib/sequelize db:seed
   e. Eventualno ce biti potrebno da uradite instalaciju *sequelize-cli* unutar projekta sa `npm install sequelize-cli` (bez -g)

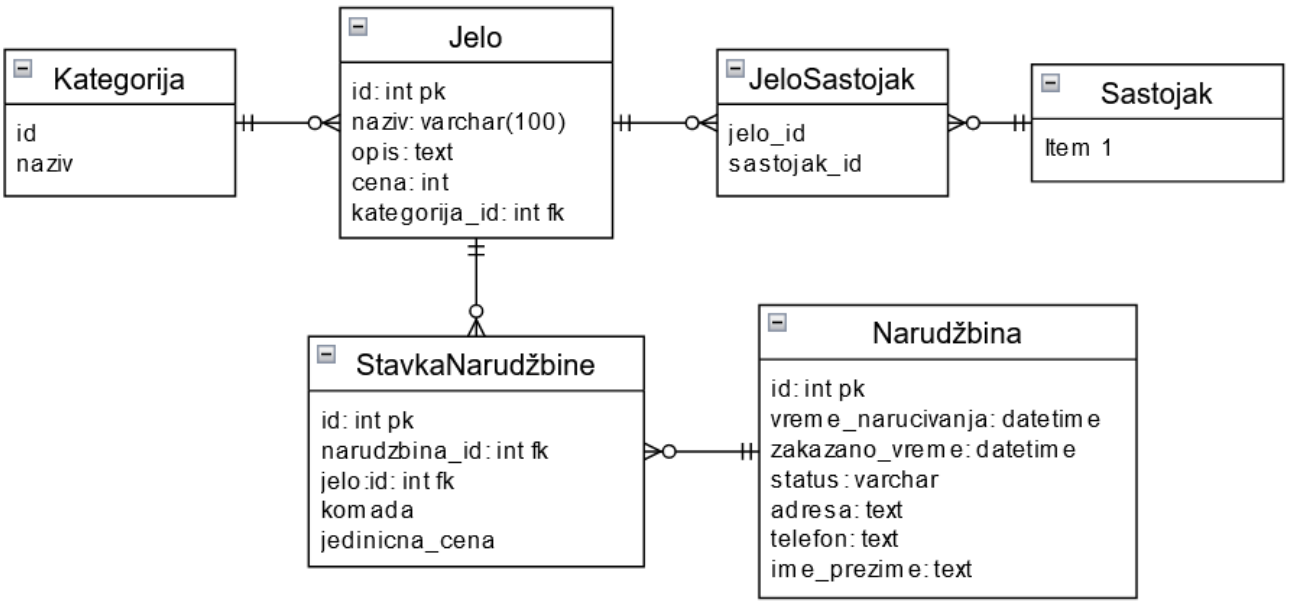
3. Inicijalizujte sequelize u projektu
   ```
   sequelize init
   ```

4. Primetite neke nove foldere u projektu. Otvorite config/config.json
5. U develpment objektu promenite parametre da odgovaraju vašoj bazi
   username: "root", password: "", database: "skriptj", dialect: "mariadb"

   Obratite pažnju na config.json i tip baze koji je naveden. MySQL i MariaDB su slični ali ne i potpuno isti, i trebalo bi da stavite mariadb ako koristite to (XAMPP dolazi sa mariadb).

6. Napravite model za jelo. U konzoli pišite:
   a. sequelize model:generate --name Jelo --attributes naziv:string,opis:string,cena:int
7. Ovo će napraviti novi fajl /models/jelo.js
8. Po istom principu napravite modele i za ostale entitete
   a. sequelize model:generate --name Kategorija --attributes naziv:string
   b. sequelize model:generate --name Sastojak --attributes naziv:string
   c. sequelize model:generate --name JeloSastojak
   d. sequelize model:generate --name Narudzbina --attributes status:string
   e. sequelize model:generate --name StavkaNarudzbine --attributes komada:int

9. Pogledajte kako izgledaju ovi fajlovi. U init() metodi dobićete generisanu strukturu entiteta, tj. atribute-kolone koje ima. Ovde nisu navedene sve kolone i treba dopuniti modele.



10. Otvorite jelo.js. Primetite da postoji definicija za naziv ali ne i za id. To je zato što se id podrazumeva. Dodajte definicije ostalih polja

```
naziv: {
      type: DataTypes.STRING(120),
      unique: true,
      allowNull: false
},
opis: {
      type: DataTypes.TEXT,
      allowNull: true
},
cena: {
      type: DataTypes.INTEGER,
      allowNull: false
},
kategorija_id: {
      type: DataTypes.INTEGER,
      allowNull: false
}
```

    Tipove možete videti na https://sequelize.org/docs/v7/models/data-types/
    Pored ovoga sequelize omogućava i dodavanje pravila za validaciju, sakrivanje kolona itd. Prilično je bogat mogućnostima i vredi istražiti.

11. Sada ćemo da dodamo relaciju između Jelo i Kategorija. Iznad init() primetite metod static associate(). Tu se definišu relacije između modela.
12. Navedite nazive svih modela sa kojima je jelo u relaciji: kategorija, sastojak, stavkanarudzbine
    a. static associate({Kategorija, Sastojak, StavkaNarudzbine})
13. Definišite relacije unutar associate()
    a. Jelo ima jednu kategoriju
        this.belongsTo(Kategorija, {foreignKey: "kategorija_id", as: "kategorija"});

b. Jelo se pojavljuje u više stavki narudžbine
this.hasMany(StavkaNarudzbine, {foreignKey: "jelo_id", as: "stavke"});
c. Jelo ima više sastojaka, kroz međutabelu jelosastojak
this.belongsToMany(Sastojak, {foreignKey: "jelo_id", as: "sastojci", through:"JeloSastojak"});

14. Primenite ovo i na ostale tabele

15. Kada završite modele vreme je za migraciju modela u bazu podataka, tj. generisanje i izvršavanje sql naredbi koje će napraviti tabele odgovarajućeg imena i strukture, u skladu sa modelima.
16. U konzoli izvršite
sequelize db:migrate
17. Pristupite bazi i proverite da li sve tabele napravljene. (Vidi deo 1)
a. Primetite da tabele imaju sufiks *s* - jer je to množina u engleskom. Naziv modela/klase se dinamički mapira na ime tabele
18. Ako želite da menjate nešto u strukturi baze, možete da obrišete sve sa
sequelize db:migrate:undo:all

5 Kreiranje inicijalnih podataka u bazi - seedovanje

Sada ćemo pripremiti incijalni paket podataka koji treba da postoji u bazi prilikom setup-a aplikacije. Ovo je zgodno imati ako često radite testiranja, pa pišete i brišete po bazi i želite da se lako vratite na početno stanje.

1. U konzoli generišite seedere za entitete sa
a. sequelize seed:generate --name create-jela
b. sequelize seed:generate --name create-kategorije
c. sequelize seed:generate --name create-sastojci
2. Ovo će kreirati fajlove u folderu seeds/
3. Otvorite seeders/####-create-jela.js i primetite dva metoda: up i down. Up se poziva kada se insertuju podaci, down kada se brišu podaci iz baze.
4. Dodajte kod u up(), u kome kreirate nekoliko jela
```
await queryInterface.bulkInsert('jelos',
[
  {id:"1",naziv:"Vegeterijana", opis:"masna", cena: 1200, kategorija_id:1},
  {id:"2",naziv:"Kobasica", opis:"posna", cena: 300, kategorija_id:2}
]);
```
5. Dodajte kod u down() kojim brišete sve iz tabele
a. await queryInterface.bulkDelete('Jela', null, {});
6. Isto uradite i za ostale seedere.
7. U konzoli izvršite
a. sequelize db:seed:all
8. Proverite da li su tabele u bazi popunjene podacima.

6 Pristup bazi podataka u rutama

Sada ćemo konačno napisati kod koji u rutama dohvata podatke iz baze. Prvo treba da u aplikaciju i module uključimo sequelize i modele.

1. Otvorite app.js
2. Ispod require("express") dodajte
   ```
   const { sequelize, Jelo, Kategorija, JeloSastojak, Sastojak, StavkaNarudzbine, Narudzbina } = require("./models");
   ```
3. Umesto app.listen(9000); stavićemo poziv koji ima callback, u kome ćemo, kada se aplikacija pokrene, pozvati ispis poruke u konzolu ("REST servis je pokrenut") i pozvati sequelize inicijalizaciju - otvaranje konekcije i ostalo na osnovu config.json
   ```
   app.listen({ port:8000 }, async () => {
       console.log("Started server on localhost:8000");
       await sequelize.sync({force:true});
       console.log("DB synced");
   });
   ```

4. Otvorite routes/jelo.js
5. Ispod require("expres") dodajte
   ```
   const { sequelize, Kategorija, JeloSastojak, Sastojak, StavkaNarudzbine} = require("../models");
   ```
   a. Dodajete one modele koji su vam potrebni u rutama u ovom modulu
6. Isto primenite i na ostale modele

Sve rute koje budete nadalje definisati proglasite za async, jer će u sebi sadržati pozive kao ORM operacijama koje su asinhrone i-o operacije ka bazi podataka, pa ćemo da koristimo await radi čitljivijeg koda.

7. U rutu GET /jelo/ u try blok umesto ispisa poruke "sva jela" dodaćete dohvatanje svih redova tabele i vraćanje klijentu u JSON formatu
   ```
   const kategorije = await Category.findAll();
   return res.json(kategorije);
   ```

8. U rutu GET /jelo/:id koja prikazuje jedno jelo po zadatom ID-u dodaćete dohvatanje tog jednog reda po id
   ```
   const jelo = await Jelo.findByPk(req.params.id);
   return res.json(jelo);
   ```

9. U rutu POST /jelo koja vrši unos sumitovane forme-podataka u bazu dodaćete unos objekta u bazu. Ovde pretpostavljamo da forma submituje objekat sa atributima koji se zovu isto kao u tabeli-modelu, inače biste morali da pravite zaseban objekat i prepakujete podatke
   ```
   const novi = await Jelo.create(req.body);
   return res.json(novi);

   a ako objekat nije iste strukture kao model onda bi išlo nešto u ovom stilu:
   const novi = {};
   ```

```
novi.naziv = req.body.mojNaziv;
novi.opis = req.body.opisKojiSeDrugacijeZove;
const insertovani = await Jelo.create(novi);
return res.json(insertovani);
```

10. U rutu PUT /jelo/:id koja vrši izmenu postojećeg jela koje ima zadati ID dodaćete update modela. Prvo dohvatamo model po id, pa postavljamo nove vrednosti atributa, i pozivamo save()

```
const jelo = await Jelo.findByPk(req.params.id);
jelo.naziv = req.body.naziv;
jelo.opis = req.body.opis;
jelo.cena = req.body.cena;
jelo.kategorija_id = req.body.kategorija_id;
jelo.save();
return res.json(jelo);
```

11. U rutu DELETE /jelo/:id koja vrši brisanje postojećeg jela po ID dodaćete dohvatanje modela po id, pa poziv destroy() koji ga briše iz baze.

```
const jelo = await Jelo.findByPk(req.params.id);
jelo.destroy();
return res.json( jelo.id );
```

Testirajte rute kroz ThunderClient, a kroz phpmyadmin ili konzolu proveravajte da li se sadržaj baze menja na očekivani način. Svakako i u rezultatima requestova vidite ispis podataka, pa i na osnovu toga možete da primetite da li sve radi kako treba.

7 Dovršavanje GUI aplikacije na app_servis

Pošto imamo rest servis završen, možemo da pređemo nazad na app_servis i da ga povežemo sa ovim servisom. U vežbi 2 napravili smo nekoliko ruta kao primer, koje rade sa tekstualnim fajlom. Sve to ćemo preusmeriti na rest servis, uz manje korekcije postojećeg koda.

1 CORS

Pošto ćemo sada da imamo dva servisa, gde browser sa jednog dobija GUI aplikacije (html i ostalo) a sa drugog traži podatke, dolazimo u problem - ovo nije dozvoljeno. Iako su oba na localhost, rade na različitim portovima i što se browsera tiče to su dva različita servisa, kao što i jesu. Browser onda sumnja zašto stranica sa jednog servisa šalje request ka drugom servisu. Ovo se zove cross-origin resource sharing, i uz malo podešavanja možemo browseru da objasnimo da je sve u redu.

1. U rest servisu instalirajte cors
    a. npm install cors
2. U rest servisu, u app.js, nakon const app=express(); dodajte

```
const corsOptions = {
  origin: ['http://localhost:8000', 'http://127.0.0.1:8000']
```

```
};
app.use(cors(corsOptions));
```

3. Pazite da i 127.0.0.1 i localhost rade isto, ali browser ih ne vidi kao isto, jer nije isti string…
4. Restartujte oba servisa i proverite da li stranice servirane sa app servisa mogu da komuniciraju sa rutama na api servisu.
5. Sa F12 otvorite dev tools i nađite tab Network. Tu možete da proveravate saobraćaj ka i od servera i da vidite opise grešaka i problema ako ih bude u toku rada. Deo grešaka se ispisuje i u Console. CORS problem bi trebalo da vidite prijavljen u konzoli browsera.

2 CRUD - spisak (AJAX način)

1. U app servisu otvorite static/jela.html
2. Pronađite kod koji dohvata podatke-spisak jela: fetch(...)
3. Promenite u fetch("http://localhost:9000/jelo/") …. <- pazi na port i koji server gađaš
4. Sačuvajte, probajte da li se spisak učitava, proverite koje podatke vuče u konzoli.
5. Kategorija je sada objekat unutar objekta jelo, pa treba da promenite ispis u nešto oblika
   td2.innerHTML = data[i].kategorija.naziv
   unutar funkcije gde kreirate redove i dodajete ih u tabelu
6. Ako ne vidite unutrašnji objekat kategorija pogledajte ovo https://sequelize.org/docs/v6/advanced-association-concepts/eager-loading/#required-eager-loading
   a. Lazy loading je koncept da se prilikom učitavanja modela ne učitavaju asocirani modeli dok to nije nužno potrebno - do trenutka poziva asociranih modela. Prilikom ispisivanja u JSON to bi trebalo da se uradi. Ako ne, onda možete da forsirate eager loading prilikom upita, i to će da za element povuče iz baze i pridružene elemente iz drugih tabela.
7. Linkove *Izmeni* prilagodite tako da u linku sadrže ID jela u tom redu. Nešto ovako:
   a. `<a href="jelo.html?id=${data[i].id}" class="btn btn-primary">Izmeni</a>`

Promena cene

8. U svakom redu potrebno je da imamo i ID tog zapisa u bazi, kako bi *Promena cene* mogla da pošalje odgovarajući upit.
   a. U <tr> dodajte data-attribute id (proizvoljni atribut sa podacima, naziv treba da bude u obliku data-nesto="vrednost") i pridružite mu vrednost id-a zapisa u bazi.
   b. tr.dataset.id = data[i].id;
9. U api servisu dodajte rutu za izmenu cene, koja dobija dva parametra: cenu i id jela. Pošto je u pitanju update operacija, trebalo bi koristiti PUT http request, koji ima id u url-u. Koristićemo i ovde async/await sintaksu da bismo izbegli callback hell, pošto su operacije ka bazi takođe asinhrone i-o operacije
   a. app.put("http://localhost:9000/promeni-cenu/:id", **async** (req,res)=>{

```
try{
    //dohvati objekat po ID
    const jelo = await Jelo.findByPk(req.params.id);  //iz url
    jelo.cena = req.body.cena;  //iz body
    jelo.save();
    return res.json(jelo);  //vrati json nove vrednosti jela i završi funkc.
```

```
        } catch(err){
            console.log(err);
            res.status(500).json({ error: "Greska", data: err });
        }
    });
```

10. U app servisu, na stranici jela.html, pridružite event listener događaja click na dugme Promeni cenu. To ćete uraditi unutar funkcije u kojoj se popunjava tabela, i gde pravite nove button elemente sa document.createElement().
    a.
```
btn.addEventListener("click", function(){
    …promena cene
});
```
11. Prikažite prompt za unos nove cene, dohvatite id tog reda i napravite http put request ka ruti za izmenu cene

```
var c = prompt("Unesi novu cenu:");
var id = this.parentNode.dataset.id;
fetch("http://localhost:9000/promeni-cenu/" + id, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ cena: c })       /* ime:vrednost */
})
.then( response=>response.json() )
.then( data=>{
    /* obradjujemo odgovor */
})
.catch( err=>{
    alert("Desila se greska");
    console.log(err);                   /*ispisujemo gresku u konzoli browsera*/
});
```

12. Na osnovu responsa osvežite vrednost cene u spisku. Odgovor sadrži objekat u kome je id i nova cena. U then() koji obrađuje odgovor imaćete nešto oblika

```
document.querySelectorAll(` #spisak > tr[data-id='${data.id}'] > td:nth-child(2) `)
  .innerHTML = data.cena;

// dohvatamo u <tbody id="spisak"> onaj <tr> koji ima atribut data-id čija je vrednost
id zapisa kome je menjana cena. Menjamo innerHTML u novu vrednost, dobijenu u
odgovoru od update rute.
```

3 Novo jelo

Ovo bi trebalo da radi samo po sebi, pošto se submituje forma po method="POST". Međutim, ruta vraća novouneti objekat u JSON formatu, pa UX neće biti baš najsjajniji ako samo ostavimo ovako. Umesto da samo submitujemo formu, uradićemo fetch() ka insert ruti, sa POST metodom, i sadržajem novog jela (vrednosti polja u formi) upakovanim kao objekat koji je u json formatu u telu zahteva (req.body). Možemo da ostavimo <form> i da na onsubmit nakon validacije uradimo fetch i suspendujemo submit event - praktično da umesto defaultnog ponašanja forme (post zahtev) pošaljemo svoj request kako nam odgovara.

1. Formi i input elementima unutar forme pridružite id atribute
2. Pridružite formi događaj submit u onload

```
document.getElementById("forma").addEventListener("submit", function(event){
	event.preventDefault();		//sprecimo default ponasanje
	var validno = validacija();	//uradimo validaciju
	if(!validno){ return; }		//ako nije validno - kraj

	var novoJelo = {};		//napravimo novi objekat jela
	novoJelo.naziv = document.getElementById("naziv").value;
	// isto i za opis, kategorija, cena

	fetch("http://localhost:9000/jelo/novo-jelo", {
		method:"POST",
		headers: { 'Content-Type': 'application/json' },
		body: JSON.stringify(novoJelo)
	})
	.then(succ=>succ.json())
	.then(data=>{
		//dobili smo objekat novounesenog jela, koje ima svoj id, super
		//mozemo nazad na spisak, a mozemo i na izmenu tog jela
		window.location.href=`/jelo/izmeni/${data.id}`;
	})
	.catch(err => console.log(err));
});
```

4 Izmena jela

ID jela se nalazi u url-u. Iz url dohvatamo id, zatim puštamo fetch da bismo dobili objekat-zapis tog jela u bazi i na osnovu toga postavili default vrednosti, prilikom onload. I na kraju pridružujemo onsubmit gde želimo da umesto očekivanog ponašanja forme (post) pošaljemo PUT request - koji nije predviđen html standardom, pa ćemo to svakako morati da uradimo pomoću fetch(). Ako dobijemo dobar odziv vratićemo se na spisak, inače ispisujemo poruku o grešci i ostajemo na stranici.

1. Dodajte id atribute svim input poljima u formi
2. Deklarišite globalnu promenljivu **id**
    a. U <script>, u globalnom scope (van funkcija) stavite
        i. var id = null;
3. U onload, dohvatimo id iz url

```
var url = new URL( window.location.href );
id = url.searachParams.get("id");  //shvatiće da se misli na globalnu varijablu id
alert(id);  //proverite sta ste dohvatili
```

4. Na osnovu id dohvatimo objekat jela sa get rute. GET je default

```
fetch("http://localhost:9000/jela/" + id).then( resp=>resp.json() )
	.then( data=>{
		console.log(data); //proverite sta ste dobili
	})
	.catch(err=>console.log(err));
```

5. Iz dobijenog postavimo default vrednosti

```
document.getElementById("naziv").value = data.value;
//itd za ostala polja
```

6. Za sastojke ćete pozvati funkciju dodajSastojak() u petlji koja prolazi kroz niz sastojaka

```
for(let i=0; i<data.sastojci.length; i++){
        dodajSastojak(data.sastojci[i]); //zavisno sta je u nizu, mozda vam treba .id
}
```

7. Probajte da li se prilikom otvaranja *Izmeni* polja automatski popunjavaju prilikom učitavanja stranice.

8. Slično kao u **3. Novo jelo**, sprečavamo default ponašanje forme, pozivamo validaciju, pripremamo objekat sa podacima i šaljemo request sa fetch(). U ovoj stranici već imate onsbumit handler koji delimično priprema podatke u json (sastojci), pa se nastavite na to.

5 Brisanje jela

Brisanje je jednostavno: pitamo korisnika da potvrdi brisanje (sa confirm()), pa ako potvrdi, šaljemo DELETE request sa id-em koji se briše iz baze, sa fetch(). Prikazemo nekakav status i predjemo na spisak.

1. Na *Obriši* dodajte događaj click
    a. btnObrisi.addEventListener(...)
2. Tražite povrdu
    a. if(confirm("Potvrdi brisanje")){

    } else {
        return;
    }
3. Ako se potvrdi, dohvatite id i sa fetch() pošaljite request
    a. ID je već deklarisan kao globalna promenljiva u onload, za potrebe postavljanja default vrednosti
    b. fetch("http://localhost:9000/jelo/"+id, { method:"DELETE" })
        .then(resp=>resp.json()).then(data=>{
            //response sadrzi samo id obrisanog
            alert("Obrisan je zapis "+data);
            window.location.href("/jelo"); //predji na spisak
        })
        .catch(err=>console.log(err));

Samostalni rad

Primenite ovo na vaš projekat. Napravite tabele za sve entitete. Napravite API rute za sve CRUD operacije nad svim entitetima. U readme stavite primer ili opis strukture objekata koji se šalju na rute, kao podsetnik/dokumentaciju. U app serveru u static/ dodajte sve potrebne fajlove tako da imate potpun set stranica za sve CRUD operacije (tabela/spisak, detaljnije/izmena, unos novog). Na glavnom meniju stavite linkove ka svim stranicama sa spiskovima. Time bi trebalo da imate kompletiranu aplikaciju za administratora.
Login funkcionalnost ćemo dodati naknadno, o tome ne treba da brinete u ovom trenutku.