

Vježba 10 - Sequelizee

Na vježbi 9 Node aplikaciju smo povezali sa bazom. Za povezivanje smo koristili mysql paket i sve SQL upite smo izvršavali koristeći query metodu iz navedenog paketa nad objektom konekcije. U ovoj vježbi ćemo vidjeti primjer jednog ORM-a koji značajno olakšava rad sa bazom.

Sequelize je ORM (Object-relational mapping) baziran na promise-ima (više o ovome poslije). ORM je način kako je moguće objekte spasiti i vratiti iz baze podataka tako da sa njima u aplikaciji možemo raditi kao i sa ostalim objektima kreiranim u aplikaciji. Sequelizee podržava razne baze podataka, a među njima i MySQL kojeg koristimo.

Da bi koristili sequelize u Node aplikaciji potrebno je instalirati pakete **sequelize** i **mysql2**. Povezivanje sa bazom se ostvaruje pri kreiranju Sequelizee objekta, a za njegovo kreiranje potrebni su parametri, naziv baze, username i password, kao i objekat u kojem definišemo host, dialect sa vrijednosti mysql i pool objekat.

Primjer definisanja konekcije:

```
const Sequelize = require('sequelize');
const sequelize = new Sequelize('dbname', 'username', 'password', {
  host: 'localhost',
  dialect: 'mysql',
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
});

// Ili preko connection stringa
const sequelize = new
Sequelize('mysql://user:pass@localhost:3306/dbname');
```

U primjeru vidimo da je potrebno uključiti sequelize biblioteku te kreirati novi sequelize objekat. Postoje dva načina kako kreiramo konekciju (oba su navedeni u primjeru iznad).

Definisanje modela

Model predstavlja jednu tabelu u bazi, veze između modela su ujedno veze između tabela, a instanca jednog modela predstavlja red iz tabele. Model u sequelize-u definišemo sa

`sequelize.define('ime modela',{atributi},{opcije})`. Ime modela naziv koji će vrijediti za svaku instancu modela, ukoliko je naziv modela user tada sequelize kreira tabelu sa nazivom users.

Primjer definisanja modela:

```
sequelize.define('nazivModela', {
  nazivKolone: {
    type: Sequelize.STRING,
    validate: {
      is: ["[a-z]", 'i'], // regex koji dozvoljava samo slova
      isIn: {
        args: [
          ['ba', 'en']
        ],
        msg: "Treba biti bosanski ili engleski"
      }
    },
    field: 'naziv_kolone'
  }
})
```

U prethodnom primjeru model je definisan sa nazivom nazivModela i kolonom nazivKolone, tip je postavljen na Sequelize.STRING, neki od često korištenih tipova:

- **Sequelize.STRING** - VARCHAR dužine 255
- **Sequelize.STRING(N)** - VARCHAR dužine N
- **Sequelize.TEXT** - TEXT
- **Sequelize.INTEGER**
- **Sequelize.FLOAT**
- **Sequelize.DATE**

Atributom validate postavljamo pravila za validaciju koja trebaju da vrijede za navedenu kolonu.

U prethodnom primjeru kolona ima dvije validacije prva je da mora biti zadovoljen regex, a druga validacija je da uneseno treba biti neka iz niza vrijednosti, sa msg možemo navesti poruku koja će se vratiti u slučaju da validacija nije zadovoljena. Field atributom navodimo ime kolone koje će se koristiti u bazi podataka. Više o ovome na:

<http://docs.sequelizejs.com/manual/tutorial/models-definition.html>

<http://docs.sequelizejs.com/class/lib/model.js~Model.html>

Kreiranje tabela

Tabele iz modela kreiramo tako što pozovemo metodu **sequelize.sync()**. Ova metoda će da kreira tabele za sve definisane modele, ako tabele već ne postoje. Ukoliko želimo da se tabela kreira i u slučaju da postoji, tako da se prvo obrišu tabele koje postoje pozivamo **sequelize.sync({force:true})**.

Metode za rad sa modelom

Slijedi lista metoda koje se često koriste:

- **MojModel.findById** - pretraga modela po primarnom ključu
- **MojModel.findOne** - vraća prvu instancu modela koja zadovoljava proslijeđene ulove, uslov se proslijeđuje kao parametar {where:{JSObjekat_uslova}}. JSObjekat_uslova je js objekat čiji atributi odgovaraju kolonama nad kojim postavljamo uslov jednakosti, a vrijednosti atributa predstavljaju vrijednosti koje kolona treba da zadovoljava.
 - `MojModel.findOne({where:{atribut:'vrijednost'}})`
 - U where uslovu mogu se nalaziti i operatori poput OR, AND, LIKE i sl. Njih navodimo na sljedeći način:
 - `MojModel.findOne({where:{atribut:[{Op.operator}:[lista operanada]]})`
 - Primjer 1. `Knjiga.findOne({where:{naslov:[{Op.or}:['jedan','dva']]})`
 - Primjer 2.
`Knjiga.findOne({where:[{Op.or}:[{naslov:'jedan'},{autor:'neki'}]})`
 - Listu operatora možete vidjeti na [link](#)
- **MojModel.findAll** - vraća instance svih objekata koji zadovoljavaju uslov, uslove pišete kao u prethodnoj metodi
- **MojModel.findOrCreate** - ukoliko model sa zadatim uslovom ne postoji kreira se novi
- **MojModel.create** - kreira se nova instanca modela i upisuje se u tabelu, objekat se proslijeđuje kao parametar metode
- Sortiranje rezultata definišemo postavljanjem [order atributa](#).
- Izmjenu modela radimo tako što izmijenimo objekat, a promjene spasimo kao **objekat.save()** ili pozivanjem **objekat.update()**, gdje kao parametar proslijedimo objekat sa atributima koje želimo ažurirati
- Objekat brišemo tako što nad njim pozovemo metodu **destroy()**
- Veze između modela se definišu na sljedeći način:
 - Jedan na jedan veza - **Model1.belongsTo(Model2)**, foreign key se kreira tako što se uzme naziv model2 i na njega doda naziv primarnog ključa (npr. model2_id). Ako želimo da se veza odnosi na neko drugo polje, a ne na primarni ključ onda kao drugi parametar metodi proslijedimo objekat {foreignKey:'naziv_ključa',targetKey:'drugo_polje'}
 - Jedan na jedan veza (ključ veze je na modelu2) - **Model2.hasOne(Model1)**
 - Jedan na više veza - **Model1.hasMany(Model2,{as:'kolekcija'})**, Model2 dobija atribut sa nazivom model1_id, a model1 dvije metode getKolekcija i setKolekcija koje predstavljaju listu model2 instanci.
 - Primjer 1. **Knjiga.hasMany(Autor,{as:'autori'})**, sve instance knjiga imaju getAutori i setAutori.
 - [Više detalja o relacijama](#).

Promise

Kako smo ranije rekli sequelize radi na principu promise-a. Promise je drugi način kako je moguće procesirati podatke asinhrono metode (pored pristupa putem callback metoda). U pristupu gdje se asinhronost rješava kroz callback metode odgovornost za nastavljjanje posla nakon izvršenja asinhrono metode je sa našeg programa (biblioteka/modul sa asinhronim funkcijama) prebačena na pozivaoca. Često nam treba način gdje program može biti siguran da će nastavak izvršavanja biti pod njegovim uslovima. Kod callback metode smo prinuđeni da se nadamo da će onaj ko programira callback metodu uraditi sve kako je planirano kako bi se osigurao nastavak izvršavanja programa. Način gdje nastavak izvršavanja ostaje na programu, a pozivalac će kasnije dobiti potrebne informacije o obavljenom poslu se naziva **Promise**.

Promise radi sa vrijednostima koje ne moraju odmah biti dostupne, primjer je ajax poziv nekog web servisa, rezultat tog poziva će biti neka vrijednost ili će se desiti neka greška. O promise-ima možemo razmišljati kao o event listenerima, event listener reaguje na događaje, a promise "registruje" događaje završetka asinhronog posla ili prekida. Ako je **X** promise on ima metodu **then** u ovoj metodi se definiše šta se dešava u slučaju uspješno završenog posla ili u slučaju greške.

Primjer 1.

```
function dajRandom() {
    return Math.floor(Math.random() * 10);
}

function asinhronoDajRandom() {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            var x = dajRandom();
            if (x <= 5) resolve(x);
            else reject("broj je veći od 5");
        }, 2000);
    });
}

var uspjeh = function (x) {
    console.log(x);
}

var neuspjeh = function (poruka) {
    console.log("Došlo je do greške: " + poruka);
}

asinhronoDajRandom().then(uspjeh, neuspjeh);
```

Primjer 2.

Koristeći promise moguće je uzastopno pozvati asinhronone metode jednu za drugom:

```
asinhronoDajRandom().then(function (x) {  
    console.log(x);  
    return asinhronoDajRandom();  
}, neuspjeh).then(uspjeh, neuspjeh);
```

Primjer 3. Ukoliko pozivamo više asinhronih metoda istovremeno možemo putem promise-a detektovati kada se svi pozivi uspješno završe:

```
function dajRandom() {  
    return Math.floor(Math.random() * Math.floor(10));  
}  
  
function asinhronoDajRandom(redBr = 1) {  
    return new Promise(function (resolve, reject) {  
        setTimeout(function () {  
            var x = dajRandom();  
            if (x <= 5) resolve(x);  
            else reject("broj je veći od 5, broj poziva " + redBr);  
        }, 2000);  
    });  
}  
  
var uspjeh = function (x) {  
    console.log(x);  
}  
  
var neuspjeh = function (poruka) {  
    console.log("Došlo je do greške: " + poruka);  
}  
  
var nizPromisea = [];  
for (var i = 0; i < 5; i++) {  
    nizPromisea.push(asinhronoDajRandom(i + 1));  
}  
Promise.all(nizPromisea).then(uspjeh, neuspjeh);
```

Ako želimo da detektujemo kada je prvo uspješno završeno izvršavanje nekog promise-a umjesto all koristimo race.

Više o promiseima [1](#) i [2](#).

Prethodno opisane metode findOne, findAll i ostale kao rezultat vraćaju promise te da bi dobili rezultat upita koristimo then metodu.

Zadatak 1*. Zadatke sa prethodne vježbe napravite da koriste sequelize.

Pomoćni kod:

baza.js

```
const Sequelize = require("sequelize");
const sequelize = new Sequelize("imenikWT", "root", "root", {
  host: "127.0.0.1",
  dialect: "mysql"
});
module.exports = sequelize;
```


imenik.js

```
const Sequelize = require("sequelize");
const sequelize = require("./baza.js");

module.exports = function (sequelize, DataTypes) {
  const Imenik = sequelize.define('Imenik', {
    ime: Sequelize.STRING,
    prezime: Sequelize.STRING,
    adresa: Sequelize.STRING,
    brojTelefona: {
      type: Sequelize.STRING,
      validate: {
        is: {
          args: [/^[0-9]{3}\/[0-9]{3}\-[0-9]{3}/],
          msg: "Nije pravilan format broja telefona"
        }
      }
    },
    datumDodavanja: Sequelize.DATE
  });
  return Imenik;
}
```

app.js

```
const Sequelize = require('sequelize');
const bodyParser = require('body-parser');
const sequelize = require('./baza.js');
const express = require("express");
const app = express();
const Imenik = sequelize.import(__dirname+"/imenik.js");
Imenik.sync();
```



Zadatak 2. U tabelu dodajte dugmadi delete i edit koja će služiti za editovanje i brisanje reda.

Zadatak 3. Napravite kada se klikne na ime nekoga iz tabele da se prikazuje tabela njegovih poznanika, na dnu tabele dodajte drop-down u kojem je lista onih koji nisu poznanici i pored drop-down-a dugme za dodavanje poznanika.