Vežba 2

U vežbi 2 pravimo node.js express web server (*app servis*) sa kog ćemo servirati html stranice napravljene u vežbi 1. Definisaćemo API rute i dodati AJAX u front-end interfejs. Podatke ćemo čuvati u tekstualnim fajlovima, u JSON formatu, za sad. U vežbi 3 povezaćemo se na MySQL bazu podataka. Vežba 2 nastavlja se na vežbu 1.

Sadržaj vežbe:

1. Postavljanje projekta
2. Dodavanje static middleware za isporučivanje statičkog sadržaja
3. Dohvatanje sadržaja forme - get i post request parametari i upotreba bodyparser middleware
4. Validacija forme upotrebom Joi na backend strani - za novo-jelo.html
5. Upisivanje sadržaja u json formatu u tekst fajl u post ruti - unos novog jela i upis u fajl
6. Učitavanje sadržaja iz fajla i ispisivanje u json formatu na get ruti - ispis svih jela iz fajla
7. Dohvatanje sadržaja sa front-end strane kroz AJAX - popunjavanje tabele jelima iz fajla

0 Instalacija potrebnog softvera

Sa https://nodejs.org/en/download preuzmite node.js instalaciju za vas OS i instalirajte. NPM dobijate u sklopu instalacije.

1 Postavljanje projekta

1. Napravite novi projektni folder **app_servis** i u njemu skript **app.js** i podfolder **static**.
2. U podfolder static prebacite fajlove iz vežbe 1.
3. Otvorite konzolu u projektnom folderu (node1) i inicijalizujte projekat sa **npm init**
4. Instalirajte express - **npm install express**

U app.js napravite osnovni kostur express aplikacije

5. Uključite pakete express i path (path sadrži funkcije za rad sa putanjama)

```
const express = require('express');
const path = require('path');
```

6. Dohvatite instancu expressa

```
const app = express();
```

7. Registrujte home rutu - želimo da isporučimo index.html iz static podfoldera, pa moramo da kreiramo putanju do fajla. __dirname je globalna konstanta koja ima predefinisanu vrednost trenutnog foldera, gde se nalazi app.js

```
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'static', 'index.html'));
});
```

8. Otvorite konekciju na portu 8000

```
app.listen(8000);
```

9. Pokrenite aplikaciju u konzoli
   node app
10. Testirajte u browseru:
    http://localhost:8000
11. Trebalo bi da dobijete sadržaj stranice ali bez slika, vašeg css-a i js
    Index.html koristi dodatne css, js fajlove i slike - naša node aplikacija nije podešena da odgovara na te zahteve i isporučuje te fajlove. Svi ti zahtevi trenutno idu "u prazno"

2 Dodavanje static middleware

Dodajte middleware za isporučivanje statičkog sadržaja-fajlova iz javno dostupnog foldera - za to smo napravili *static* folder.

12. Nakon dohvatanja instance expressa, a pre definisanja home rute, dodajte kod kojim će express priključiti *static* middleware. Middleware prilikom inicijalizacije traži putanju do foldera čiji će sadržaj biti javno dostupan - u našem slučaju podfolder static.
    ```
    app.use( express.static( path.join(__dirname, 'static') ) );
    ```

13. Testirajte u browseru. Trebalo bi da dobijete lepu stranicu.
    a. Proverite da li linkovi rade, pogledajte kako izgledaju url-ovi stranica
    b. Ako ipak ne radi - možda ste zaboravili da restartujete aplikaciju u konzoli
       i. Ili vam linkovi nisu dobri

3 Dohvatanje sadržaja forme

Na primeru stranice novo-jelo.html napravićemo funkcionalnost slanja sadržaja popunjene forme na back-end i obrade poslatih podataka na back-end strani-u node.js express ruti.

14. Otvorite static/novo-jelo.html
15. Pronađite <form> i promenite action u link ka ruti vašeg servisa: action="/novo-jelo"
16. method treba da ostane post
17. Sva input polja već imaju atribut name. Imamo naziv, opis, kategorija, cena.
    name se koristi kao naziv parametra koji šalje na backend. Ako input nema *name* njegova vrednost se uopšte ne šalje

    Parametri se šalju u request-u, u obliku koji zavisi od method.

    Za method=get parametri se stavljaju u url, u obliku
    sajt.com/skript?parametar=vrednost&parametar2=vrednost…

    Postoji i sefurl varijanta, gde se parametri pišu kao deo url-a, npr.
    sajt.com/blog/2022/10 je lepši zapis za sajt.com/blog?godina=2022&mesec=10
    Ako parametre šaljemo ovako biće smešteni u req.params, ali ruta izgleda nešto drugačije - u nastavku ćemo probati i to.

Za method=post parametri se stavljaju u request header.

Express te parametre dohvata i smešta u parametar *req* koji se prosleđuje handleru rute.

get parametri su u req.query

post parametri su u req.body, ali treba da dodamo middleware *bodyparser* koji će sadržaj poslat kroz post request da pročita i pretvori u objekat.

18. U app.js napravite novu post rutu /novo-jelo koja će da ispiše vrednosti poslatih parametara

```
app.post("/novo-jelo", (req, res) => {
    res.send(req.body);
})
```

19. Dodajte bodyparser, negde na početku skripta

```
const BP = require('body-parser');
```

20. Primenite bodyparser middleware na rutu /novo-jelo

```
app.use('/novo-jelo', BP.urlencoded({extended: false}));
```

Ovo treba da napišete pre definicije rute.

Middleware može da se primeni i globalno, tada ne biste pisali prvi parametar, /novo-jelo, pa bi se bodyparser izvršavao za svaku rutu

21. Testirajte rutu - otvorite kontakt stranicu, popunite formu i pošaljite. Trebalo bi da vidite JSON objekat sa vrednostima koje ste uneli u formu.
    a. Ako koristite chrome dobićete string. Ako koristite firefox dobićete formatirani JSON objekat, sa opcijom da vam prikaže raw data-string

    b. Ne zaboravite da restartujete aplikaciju u konzoli

22. Probajte da pošaljete praznu formu - dobićete prazan objekat, ali bi aplikacija (html forma) trebalo to da spreči jer imate validaciju na strani klijenta.

4 Validacija forme upotrebom Joi na backend strani

Uopšte gledano, front-end validacija je neophodna sa strane UI, kao element bitan korisniku. Front-end validacija nije bezbedna, korisnik može da na klijentskoj strani izmeni stranicu i omogući slanje parametara ili da generiše http request po svojoj volji.

Validacija na back-end strani je neophodna sa strane bezbednosti aplikacije. Backend je jedino mesto u koje imate poverenja, jer je potpuno u vašoj kontroli (na vašoj mašini). Uradićemo validaciju pomoću Joi paketa.

Joi radi tako što napravite objekat šeme validacije, gde se nalaze atributi čije su vrednosti validatori. Validator pravite tako što nanižete sve one funkcije koje proveravaju željene osobine vrednosti tog atributa - dužina, format itd. Zatim Joi šemom validirate objekat, u našem slučaju request.body, odakle dobijete grešku ili uspešnu validaciju.

23. Otvorite app.js
24. Dodajte paket Joi

```
const Joi = require('joi');
```

25. Instalirajte Joi

```
npm install joi
```

26. Upoznajte se sa paketom na https://www.npmjs.com/package/joi

27. Unutar handlera rute /novo-jelo ubacite kreiranje Joi šeme. Imamo polja naziv, opis, kategorija, cena.
    a. naziv treba da bude između 5 i 25 karaktera, obavezan.
    b. opis treba da bude neprazan string
    c. kategorija treba da bude neprazan string
    d. cena treba da bude broj veći od 0

```
const shema = Joi.object().keys({
    naziv: Joi.string().trim().min(5).max(25).required(),
    opis: Joi.string().trim().min(1).required(),
    kategorija: Joi.string().trim().min(5).required(),
    cena: Joi.number().greater(0).required()
});
```

U slučaju da vam treba složenija provera formata vrednosti možete da koristite regex - regularni izraz. Na primer, za broj telefona bi bio format: tri cifre, opciono /, pa još 6 ili 7 cifara, obavezan

```
    phone:
Joi.string().trim().pattern(/^[0-9]{3}\/?[0-9]{6,7}$/).required(),
```

28. Zatim dodajte validaciju, koja vraća objekat u kome su greška i uspeh, zavisno šta se desilo (ovo važi od verzije 14 ili tako nešto, u starijim verzijama se koristi callback - gledajte dokumentaciju)

```
const {error, succ} = shema.validate(req.body);
```

29. Na kraju, obradite rezultat - ispisaćemo dobijene statuse kako bismo videli šta Joi daje, ali ćete zavisno od situacije raditi odgovarajući prikaz statusa greški odnosno poruke o uspešnoj radnji.

```
if(error){
    res.send(error);
} else {
    res.send(succ);
}
```

30. Testirajte - vrednosti za neka polja će front end pustiti ali backend neće, npr. za naziv frontend validacija prihvata string duži od 3 slova a backend zahteva da bude duži od 5.

5 Upisivanje sadržaja u json formatu u tekst fajl

31. U ruti /novo-jelo ćemo malo ispremeštati kod koji obrađuje rezultat validacije

```
if(error){
    res.send("Greska: " + error.details[0].message);
    return;
}
//nije error
res.send("Poruka je poslana, očekujte odgovor");
```

32. Pošto želimo da pišemo u fajl uključićemo paket fs

```
const fs=require("fs");
```

33. Upisaćem jedno jelo u jedan zaseban red u fajlu. Pošto opis može da bude multiline string uradićemo zamenu newline karaktera sa
 tagom, čime zapravo i zadržavamo prelom reda u prikazu kod klijenta. Upotrebićemo regularni izraz za pronalaženje \r\n u svim mogućim kombinacijama (različiti OS koriste različite karaktere)

```
req.body.opis.replace(/\r?\n|\r/g, '<br>');
```

34. Pozvaćemo fs.appendFile, koji prima ime fajla, sadržaj za upis, i callback funkciju koja će se izvršiti nakon upisa - ovo je I/O operacija i funkcija je *async*

```
fs.appendFile("jela.txt",
    JSON.stringify(req.body) + "\n",
    function(err, succ){
        res.send("Poruka je poslana, očekujte odgovor uskoro");
    }
);
```

35. Restartujte aplikaciju, popunite kontakt formu i proverite da li se u projektnom folderu napravio novi fajl **jela.txt** u kome je json objekat sa sadržajem forme. Svako jelo je zapisano u zasebnom redu.

6 Učitavanje sadržaja iz fajla i ispisivanje u json formatu na get ruti

Dodaćemo novu rutu koja čita **jela.txt**, formira niz objekata na osnovu učitanog sadržaja, i taj niz vraća kao http response u JSON formatu klijentu. U fajlu je svaki red jedno jelo, u JSON formatu. Potrebno je da učitamo fajl, podelimo ga u niz redova, i svaki red zasebno iščitamo kao JSON string i dodamo u niz jela.

36. Otvorite app.js
37. Ispod rute /novo-jelo dodajte get rutu /jela

```
app.get("/jela", (req, res) => {
      res.send("sva jela");
})
```

38. Unutar handlera definišemo prazan niz

```
const jela = [];
```

const je u lokalnom scope-u i onemogućava reassignement promenljive - mogu da menjam niz ali ne mogu da u promenljivu upišem nešto drugo kao vrednost i pregazim postojeći niz.

39. Zatim otvaramo fajl jela.txt i čitamo ga kao niz redova

```
fs.readFile('jela.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading file:', err);
    res.status(500).send({ error: "Greška" });
    return;
  }
  //else…
  const redovi = data.split('\n');
  //i ovde dalje nastavljamo parsovanje redova i punjenje niza
});
```

readFile je async operacija. Ova funkcija ne vraća vrednost na mestu poziva već joj se kao poslednji parametar zadaje anonimna funkcija koja će se pozvati da obradi rezultat (sadržaj fajla) onda kada bude spreman.
Mogli bismo da pišemo i nešto oblika

```
sadrzajFajla = fs.readFileSync("jela.txt", "utf8");
```

..itd na način na koji smo navikli, ali ovo je blokirajuća operacija i čini da cela aplikacija-web servis stoji i čeka da OS učita fajla, što nije poželjno ponašanje.

40. Niz redova sadrži jela u JSON formatu. Parsujemo svaki element niza **redovi** i dobijeni objekat dodajemo u **jela**.

```
for(let i=0; i<redovi.length; i++){
      let obj = JSON.parse( redovi[i] );
      jela.push(obj);
}
```

41. Dobijeni niz objekata prepakujemo u JSON format i pošaljemo kao odgovor klijentu

```
res.json(jela);
```

42. Restartujte aplikaciju i probajte da li ovo radi. Iz browsera gađajte rutu http://localhost:8000/jela
43. Ako ste pitate zašto nismo prosto isporučili sadržaj fajla u kome su objekti već u json formatu - uporedite sadržaj fajla i sadržaj odgovora u browseru, postoji jedna razlika.

7 Dohvatanje sadržaja sa front-end strane kroz AJAX

Za kraj ćemo da u stranici sa spiskom jela omogućimo učitavanje jela sa get rute i da napunimo tabelu podacima prilikom učitavanja stranice, tj. na *load* događaj.

44. Otvorite static/jela.html
45. Dodajte <script> tag pred kraj stranice, pre </body>. Možete da izdvojite ovaj kod u zaseban js fajl ako ste tako postavili kostur svoje aplikacije, ali i ne morate, svakako pišemo unobtrusive kode.
46. Definišite funkciju koja će da se pozove na load

```
window.addEventListener("load", function(){
    //…
});
```

47. Unutar funkcije želimo da pošaljemo request na rutu /jela i da za početak ispišemo to u konzoli. Upotrebićemo fetch() funkciju, koja je wrapper oko XMLHttpRequest. fetch() je *async* funkcija i vraća promise, pa ćemo rezultat da obradimo u nadovezanom .then()

```
fetch('/jela')
  .then(response => {
     let data = response.json();
     console.log(data);
     //...ovde ide dalja obrada odgovora
  })
  .catch(error => {
     console.error('Error:', error);
  });
```

48. Proverite da li ovo radi. Otvorite stranicu jela.html i na F12 otvorite webmaster tools, i konzolu.
    a. Primetite da nije potrebno restartovati aplikaciju ako menjate html fajl, jer aplikacija čita html fajl u hodu, u trenutku obrade requesta.

Pošto ste utvrdili da klijent uspešno dohvata podatke možemo da te podatke dinamički umetnemo u tabelu.

49. Dodajte id="spisak" u <tbody> tabele sa spiskom jela.
50. U fetch…then… dodajte kod koji će da prođe kroz niz *data* i za svaki element da kreira novi red u tabeli i doda u tabelu.
    a. Red izgleda ovako nekako (zavisno od toga kako se radili vežbu 1):

```
<tr>
  <td>naziv</td>
  <td>kategorija</td>
  <td>cena</td>
  <td>
        <button class="btn btn-primary" onclick="">Promena cene</button>
        <a href="jelo.html?id=1" class="btn btn-primary">Izmeni</a>
  </td>
</tr>
```

U petlji za svaki element kreirajte ovu strukturu i prikačite <tr> čvor u tabelu

```
for(let i=0; i<data.length; i++){
```

```
            let tr = document.createElement("tr");
            let td1 = document.createElement("td");
            td1.innerHTML = data[i].naziv;
            tr.appendChild(td1);
            //… ostatak
            document.getElementById("spisak").appendChild(tr);
            //data[i].kategorija
            //data[i].cena

            //ovde ide i pridruzivanje click eventa na <button>  Promena cene
            let btn = document.createElement("button");
            btn.addEventListener("click", function(){
                  //..telo funkcije…
            });
      }
```

51. Sačuvajte, probajte. Trebalo bi da vidite tabelu napunjenu jelima koja se nalaze u fajlu **jela.txt**
52. Dodajte još neko jelo i proverite spisak.
53. Razmislite kako biste dodali mogućnost refreshovanja spiska.

U nastavku bi trebalo da napravimo mogućnost izmene jela i promene cene, ali za to nam je potreban ID i u ovakvoj postavci, sa čuvanjem podataka u fajlu, to postaje nezgodno. U sledećoj vežbi kreiramo bazu podataka i implementiramo kompletan CRUD set funkcionanosti (create, read, update, delete).

Svaki entitet vašeg projekta trebalo bi da ima stranice

- entiteti.html
- entitet-nov.html
- entitet-izmena.html

rute

- GET /entiteti vraća niz svih entiteta iz baze
- GET /entitet/id vraća jedan traženi entitet, sa svim svojim podacima, po ID-u
- POST /entitet unosi novi entitet na osnovu poslatih parametara iz forme
- PUT /entitet/id menja postojeći entitet po ID-u na osnovu poslatih parametara iz forme
- DELET /entitet/id briše postojeći entitet po ID-u

tabelu

- entitet

i to ćemo napraviti u sledećoj vežbi