Vježba 7 - Povezivanje frontend-a i backend-a

U prethodnoj vježbi smo vidjeli kako je moguće napraviti server koji će osluškivati zahtjeve od klijenta. Kada klijent uputi zahtjev na neki endpoint (url) potrebno je da se desi neka akcija te da server pošalje klijentu odgovarajući odgovor. Proces određivanja kako će skripta odgovoriti u zavisnosti na koji endpoint je zahtjev došao zove se rutiranje. Rutiranje u običnoj Node skripti zna biti dosta naporno (prošla vježba) zbog toga ćemo koristiti express framework u svrhe rutiranja zahtjeva.
Express uključujemo kao i dosadašnje pakete. Uključeni modul pozovemo kako bi dobili našu instancu aplikacije. Rutiranje se radi na način app.METODA(PUTANJA,FUNKCIJA), gdje je app instanca express aplikacije, METODA je HTTP metoda, PUTANJA je path dio url-a na koji se zahtjev šalje i FUNKCIJA je funkcija koja se detektuje ovakav zahtjev. Parametri FUNKCIJA su request i response objekti.

Primjer 1. Napišimo Primjer 2 sa prethodne vježbe koristeći express

```
const express = require('express');
const bodyParser = require('body-parser');
const fs = require('fs');
const app = express();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.post('/',function(req,res){
  let tijelo = req.body;
  let novaLinija = "\n"+tijelo['ime']+","+tijelo['prezime']+
     ","+tijelo['adresa']+","+tijelo['broj_telefona'];
  fs.appendFile('imenik.txt',novaLinija,function(err){
     if(err) throw err;
     res.json({message:"Uspješno dodan red",data:novaLinija});
  });
});
app.listen(8085);
```

Da bi pokrenuli ovaj primjer trebate instalirati express. Za instalaciju express-a koristit ćemo **npm**. Npm je package manager koji nam omogućava da uključimo pakete koji su drugi pisali.

Paket možete instalirati globalno - za sve projekte ili lokalno - samo za projekat na kojem trenutno radite. Da bi pakete instalirali lokalno trebamo napraviti package.json datoteku. U ovoj datoteci se nalaze osnovne informacije o projektu, kao i lista svih paketa koji su mu potrebni za izvršavanje (dependencies). Ovu datoteku kreiramo izvršavanje komante **npm init** u folderu projekta. Nakon toga u istom folderu izvršite komandu **npm install PAKET**, gdje je PAKET naziv paketa kojeg instalirate. Kako je za ovaj primjer potreban express izvršite sljedeću komandu **npm install express**, nakon uspješno izvršene komande u folderu node_modules se nalaze instalirali moduli.

Paketi koje koristimo su express i body-parser. Body-parser nam omogućava da se tijelo zahtjeva automatski parsira, sa app.use navodimo kakve sve formate očekujemo **bodyParser.json()** je za JSON, a **bodyParser.urlencoded({ extended: true }))** za x-www-form-urlencoded podatke.

Sa **app.post** definišemo šta će se desiti kada POST zahtjev stigne na adresu localhost:8085/. Tijelo zahtjeva u expressu možemo dobiti sa **req.body**, a ukoliko smo koristili pravilan bodyParser kao rezultat ćemo imati asociativni niz sa svim prosljeđenim parametrima.

Odgovor klijentu šaljemo sa **res.json**, slanje status koda se može uraditi korištenjem metode **res.status**, slanje odgovora možemo uraditi i sa **res.send**. Više o ovome na linku.

Zadatak 1*.
U prethodni primjer dodajte pravilo da ukoliko se pošalje get zahtjev na “localhost:8085/unos” da se ispiše forma za unos koja ima polja: ime, prezime, adresa i broj_telefona. Kada se submita ova forma treba se poslati POST zahtjev na localhost:8085. Odgovor nakon posta neka bude HTML tabela sa listom svih zapisa iz imenika.
Formu možete vratiti korisniku korištenjem metode response.sendFile. Napravite dokument forma.html i u njemu neka se nalazi html kod forme, kada korisnik uputi zahtjev na /unos pozovite metodu sendFile. Statički fajlovi link.
Zadatak 2.
Dodajte novo pravilo ako se pošalje GET zahtjev na “localhost:8085/” da se prikaže tabela sa svim zapisima iz imenika i pored svakog postavite dugmad “delete” i “edit”. Kada se pritisne na delete dugme šalje se GET/POST zahtjev na “localhost:8085/:ime” i skripta briše sve redove gdje ime odgovara imenu iz reda. Nakon brisanja prikažite ažuriranu tabelu. Klik na edit dugme treba prikazati formu kao iz zadatka 1 stim da su polja ispunjena sa vrijednostima iz reda u kojem je dugme bilo.
Hint: http://expressjs.com/en/guide/routing.html

Cross-site scripting (XSS)
XSS je jedan od najčešćih oblika sigurnosnih propusta na webu, pa i uopšte (uz SQL injection). Dešava se na stranicama gdje je moguće putem forme poslati HTML kod koji zatim bude u neizmijenjenom obliku prikazan na stranici. Problem je posebno izražen ako tako generisan HTML bude pohranjen u bazi podataka i zatim se prikazuje drugim korisnicima.

U mnogim slučajevima propust se svodi na to da napadač može prikladnim kodom sakriti drugi legitiman sadržaj stranice ili otežati njeno korištenje. Međutim ponekad XSS otvara mogućnost pristupa osjetljivim podacima (kao što su cookies) koristeći JavaScript i njihovog slanja napadaču.

Da biste se zaštitili od XSS **morate** očistiti sve ulaze (bilo da dolaze sa forme ili iz cookija) od svih znakova koji bi mogli biti validan HTML: https://www.npmjs.com/package/htmlencode
Ako želite da omogućite korisnicima da unose HTML, morate napraviti prilično kompliciran zaštitni sistem koristeći kombinaciju funkcije strip_tags i pažljivog izbora tagova koji nemaju atribute sa opasnim efektima (tipa izvršenja JavaScript koda) ili dodatno očistiti takve atribute pogodnim uzorcima.

Probajte u neku od formi iz prethodih zadataka unijeti sljedeće (svaka linija je zaseban primjer):

```
      "><img
src="http://www.danbailey.net/wp-content/uploads/2011/03/pwned.jpg">

      <!--

      "><script>alert('hello world');</script><
      " onclick=alert('a')
```

Svaki put pogledajte i source stranice da biste shvatili šta se dešava. Posljednja dva primjera neće raditi na nekim browserima zbog ugrađene zaštite od XSSa.

AJAX

- https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

Asynchronous JavaScript And Xml (AJAX) je skup web tehnologija koje omogućuju JavaScript programima da komuniciraju sa web serverom asinhrono (u pozadini) pri čemu nema ponovnog učitavanja web stranice.

AJAX je realiziran preko XMLHttpRequest objekta. Ključna metoda ovog objekta je **open(method,url,async)**. Prvi parametar je HTTP metoda (POST ili GET), drugi je URL a treći je true ako je zahtjev asinhron ili false ako je sinhron (što znači da je izvršenje JavaScript-a blokirano dok se ne dobije odgovor od servera). Naravno, za pravi AJAX ovdje je potrebno staviti **true**.

Poziv upućujemo metodom **send()**. Pošto je zahtjev asinhron, moramo definisati funkciju koja će se izvršiti kada se dobije odgovor od servera. Ova funkcija se pridružuje atributu onreadystatechange i njome se mogu ispitati razne stvari npr. da li je došlo do greške na serveru i

slično. Atribut readyState označava stanje zahtjeva (4 je uspješno završen zahtjev, a <4 su različite faze komunikacije sa serverom). Atribut status je kôd odgovora servera (200 je uspješan zahtjev, a npr. 404 je nepostojeći dokument). Atribut responseText je tekst koji je dobijen od servera. U slučaju da je vraćen XML kôd, možete koristiti atribut responseXML.

Primjer:

```
var ajax = new XMLHttpRequest();
ajax.onreadystatechange = function() {// Anonimna funkcija
      if (ajax.readyState == 4 && ajax.status == 200)
            document.getElementById("polje").innerHTML = ajax.responseText;
      if (ajax.readyState == 4 && ajax.status == 404)
            document.getElementById("polje").innerHTML = "Greska: nepoznat
URL";
}
ajax.open("GET", "sadrzaj.html", true);
ajax.send();
```

Ako vaš željeni URL očekuje da se proslijede i neki parametri (putem forme), moramo ih kodirati. U slučaju GET zahtjeva parametri se navode iza URLa u sljedećem formatu:

proba?parametar1=vrijednost1&parametar2=vrijednost2&...

Ovo će raditi ako se parametar1,2,... i vrijednost1,2,... sastoje isključivo od velikih i malih slova i brojeva. Većina ASCII karaktera se kao takva ne može uključiti u URL nego je potrebno kodirati ih što postižemo funkcijom **encodeURIComponent**. Primjer:

```
  var ime=document.getElementById('ime').value; // Tekstualno polje forme
  ime = encodeURIComponent(ime);
  var prezime = ... // slicno
  ajax.open("GET", "provjera?ime=" + ime + "&prezime=" + prezime, true);
  ...
```

U slučaju metode POST parametri se ne mogu na ovaj način poslati nego ih je potrebno definisati parametrima metode **send** na sljedeći način:

```
ajax.open("POST", "http://localhost:8085/forma", true);
ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
ajax.send("ime=" + ime + "&prezime=" + prezime);
```

jQuery

Biblioteka jQuery nudi znatno kraći i efikasniji način pozivanja AJAX funkcija. $.get() i $.post() definišu i odmah izvršavaju pozive sa GET odnosno POST metodom. $.ajax() daje puno više kontrole te se s njom rješava i problem slanja odnosno prijema parametara iz forme, te obrade rezultata prema tipu (npr. JSON). $.load() samo preuzima sadržaj sa servera i učitava ga u innerHTML elementa sa datim IDom.

Zadatak 3*.

Napravite web stranicu koja je organizirana u kartice (tabove) kao na slici:

Stranica 1 Stranica 2 Stranica 3
Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1 Sadržaj stranice 1

Klikom na link Stranica 1 potrebno je učitati sadržaj fajla stranica1.html i prikazati ga u okvir ispod menija, slično za Stranica 2 i Stranica 3. Samu okvirnu stranicu snimite kao index.html. Pri tome, klikanje na linkove ne treba izazvati reload (ponovno učitavanje stranice sa servera)!

Savjet: Ovaj zadatak nećete moći riješiti ako .html fajlove otvarate dvoklikom! Morate im pristupati kroz web server, odnosno preko linka http://localhost/index.html. U Node-u putem express-a možete to uraditi na sljedeći način http://expressjs.com/en/starter/static-files.html. Ili putem wampa ako stranice postavite u wamp folder.

Zadatak 4. Napravite dokument ajaxForma.html koji će sadržavati formu sa poljima kao u zadatku 1. Napravite skriptu ajaxForma.js koja će sadržavati kod koji će se pozivati kada se klikne na dugme pošalji (sada neka tip elementa bude button, a ne submit). Ovaj kod treba poslati POST zahtjev putem AJAX-a na localhost:8085 i ispisati tabelu svih unosa ispod forme (bez reloada stranice). Podatke šaljite u JSON formatu.

Primjer 2. Napravite dokument koji će sadržavati tabelu naziva i vlasnika svih repozitorija koji su podijeljeni sa vama na Bitbucket-u. Ajax skripta za učitavanje podataka neka se pozove pri učitavanju stranice.

Da bi mogli dobiti listu repozitorija od Bitbucket-a moramo dokazati naš identitet. Bitbucket-ov api kojeg ćemo kroristiti podržava OAuth2 protokol. OAuth2 je protokol za autentifikaciju klijenata. OAuth se često koristi kako bi se u ime korisnika putem druge aplikacije pristupilo sistemu (obično sa ograničenim pravima) na kojem korisnik ima korisnički račun i uradile određene akcije u ime korisnika. Korisnik mora moći dozvoliti drugoj aplikaciji pristup obično kroz neki vid redirekcije na stranicu sistema gdje korisnik unosi svoje podatke nakon čega se šalje token (string sa random znakovima) aplikaciji koja želi da pristupa sistemu. Aplikacija treba uz svaki svoj zahtjev poslati dobiveni token. Primjera korištenja OAuth protokola ima mnogo, npr. omogućavanje djeljenja sadržaja na Facebook putem vlastite aplikacij/web stranice.

***Korak 1.** Potrebno ja loginovati se na Bitbucket sa vašim korisničkim podacima. Idete na account i bitbucket settings. U OAuth dijelu trebate dodati consumer-a: Name: 'probaWt',*

*Callback URL: 'http://localhost', Permissions: Repositories: 'read'. Kao rezultat izvršavanja ovih koraka dobit ćete **Key** i **Secret**. Ove dvije vrijednosti su vam potrebne kako bi mogli dobiti token.*

***Korak 2.** Ajax skripta za dobijanje tokena:*

```
function ispisi(error,token){
   if(!error) console.log(token);
}
function getAccessToken(proslijedi){
   var ajax = new XMLHttpRequest();

   ajax.onreadystatechange = function() {// Anonimna funkcija
      if (ajax.readyState == 4 && ajax.status == 200)
         proslijedi(null,JSON.parse(ajax.responseText).access_token);
      else if (ajax.readyState == 4)
         proslijedi(ajax.status,null);
   }
   ajax.open("POST", "https://bitbucket.org/site/oauth2/access_token", true);
   ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
   ajax.setRequestHeader("Authorization", 'Basic ' + btoa(KEY:SECRET'));
   ajax.send("grant_type="+encodeURIComponent("client_credentials"));
}

getAccessToken(ispisi);
```

***KEY:SECRET** zamjenite sa onim koje ste dobili iz koraka 1*

***Korak 3.** Pozivanje Bitbucket api-a kako bi dobili listu repozitorija. U prethodnu skriptu dodajte funkciju listRepositories i pozovite getAccessToke(listRepositories).*
Implementacija listRepositories:

```
function listRepositories(error,token){
   if(error) throw error;
   var ajax = new XMLHttpRequest();
   ajax.onreadystatechange = function(){
      if (ajax.readyState == 4 && ajax.status == 200)
         {
            console.log(ajax.responseText);
```

```
            console.log(JSON.parse(ajax.responseText).values[0].name);
            console.log(JSON.parse(ajax.responseText).values[0].owner.username);
            //Dodajte dio koji će u HTML ispisati tabelu sa repozitorijima
            //Podatke parsirajte sa var podaci = JSON.parse(ajax.responseText);
            //Imena repozitorija možete dobiti sa podaci.values[i].name
            //Imena vlasnika repozitorija sa podaci.values[i].owner.username
        }
    else if (ajax.readyState == 4)
        console.log(ajax.status);
  }
  ajax.open("GET","https://api.bitbucket.org/2.0/repositories?role=member");
  ajax.setRequestHeader("Authorization", 'Bearer ' + token);
  ajax.send();
}
```

U ovom kodu rezultat se ispisuje u konzoli, ispravite ga tako da radi kako je navedeno u komentarima.

Korak 4. *Kreirajte primjer2.html sa sljedećim kodom div sa id-em tabela će da služi kao container za tabelu koju dodajete u prethodnom koraku, primjer2.js je skripta iz prethodnih koraka:*

```
<!DOCTYPE>
<html>
<head>
  <title>Lista Repozitorija</title>
  <script type="text/javascript" src="primjer2.js"></script>
</head>
<body>
  <div id="tabela"></div>
</body>
</html>
```

Korak 5. *Snimite ove dvije datoteke u neki folder public, a direktno izvan njega dodaje node skriptu:*

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

```
app.use(express.static('public'));
app.listen(8085);
```

Ova skripta kada je pokrenemo će da služi kao web server koji će na adresi localhost:8085/primjer2.html nuditi html iz ovog primjera.