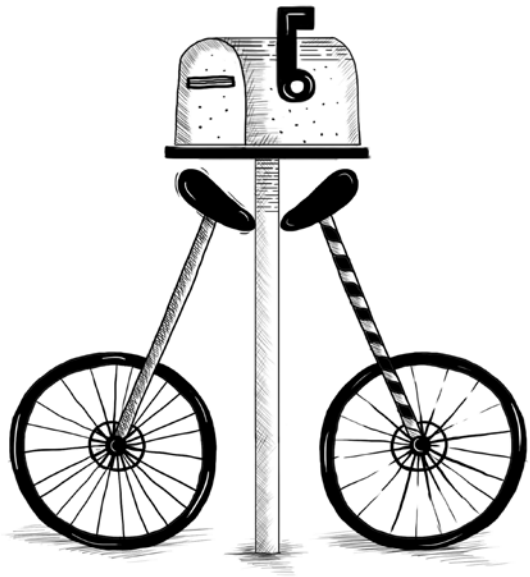
„Ako imate znanje, podelite ga sa drugima.“
– Margaret Fuller



21

PROJEKAT: VEB LOKACIJA ZA RAZMENU VEŠTINA

Razmena veština (engl. *skill-sharing meeting*) događaj je na kojem se okupljaju ljudi sa zajedničkim interesovanjima i daju male, neformalne prezentacije o stvarima koje znaju. Na baštovanskoj razmeni veština, neko bi mogao objasniti kako se gaji celer. A vi biste mogli svratiti u programersku grupu za razmenu veština i ispričati ljudima o Node.js-u.

Takva okupljanja, koja se često nazivaju i *korisničke grupe* (engl. *users group*) kada se bave računarima – odličan su način da se prošire horizonti, upozna sa novim dostignućima, ili da se prosto upoznaju ljudi sa sličnim interesovanjima. Mnogi veći gradovi imaju JavaScript okupljanja. Ona su obično besplatna, a ja sam, posetivši neka, otkrio da su ljudi na njima druželjubivi i otvoreni.

U poslednjem projektnom poglavlju, naš cilj je da napravimo veb lokaciju za upravljanje predavanjima održanim tokom sastanka za razmenu veština. Zamislite malu grupu ljudi koji se redovno sastaju u kancelariji nekog od članova da bi razgovarali o vožnji monocikla. Prethodni organizator sastanaka se preselio u drugi grad i niko se nije javio da preuzme njegovo mesto. Želimo sistem koji će omogućiti učesnicima da daju predloge za predavanja i međusobno raspravljaju o predavanjima, bez glavnog organizatora.

Ceo kôd za projekat možete preuzeti sa veb lokacije za ovu knjigu, *https://eloquentjavascript.net/code/skillsharing.zip*.

Dizajn

Ovaj projekat ima *serverski* (engl. *server*) deo, napisan za Node.js, i *klijentski* (engl. *client*) deo, napisan za čitač veba. Server čuva sistemske podatke i obezbeđuje ih za klijenta. On isporučuje i datoteke koje implementiraju sistem na klijentskoj strani.

Server čuva spisak predavanja predloženih za naredni sastanak, a klijent prikazuje tu listu. Svako predavanje ima ime predavača, naslov, sažetak i niz komentara povezanih s njim. Klijent omogućava korisnicima da predlažu nova predavanja (dodaje ih listi), da brišu predavanja i komentarišu postojeća. Kad god korisnik napravi takvu izmenu, klijent pravi HTTP zahtev kojim obaveštava server o tome.

Aplikacija će biti podešena tako da daje *živi* prikaz tekućih predloženih predavanja i njihove komentare. Kad god neko, negde, pošalje novo predavanje ili doda komentar, svi ljudi kojima je strana otvorena u čitaču trebalo bi odmah da vide izmenu. To predstavlja mali izazov – ne postoji način da veb server otvori konekciju sa klijentom, niti postoji dobar način da se zna koji klijenti trenutno posmatraju datu veb lokaciju.

Uobičajeno rešenje za taj problem naziva se *dugačko anketiranje* (engl. *long polling*), i ono je jedan od razloga za Nodeov dizajn.

Dugačko anketiranje

Da bismo mogli odmah da obavestimo klijenta da se nešto promenilo, potrebna nam je veza s tim klijentom. Pošto čitači veba tradicionalno ne prihvataju konekcije, a klijenti su često iza rutera koji bi svejedno blokirali takve konekcije, nije praktično podesiti da server inicira tu konekciju.

Možemo urediti da klijent otvori konekciju i održava je tako da je server može koristiti za slanje informacija kada je to potrebno.

Međutim, HTTP zahtev dozvoljava samo jednostavan tok informacija: klijent šalje zahtev, server vraća jedan odgovor i to je to. Postoji tehnologija nazvana *WebSockets*, podržana u savremenim čitačima, koja omogućava da se otvore konekcije za proizvoljnu razmenu podataka. Međutim, nije jednostavno ispravno je koristiti.

U ovom poglavlju ćemo koristiti jednostavniju tehniku – dugačko anketiranje – u kojoj klijenti neprekidno traže od servera nove informacije koristeći obične HTTP servere, a server odugovlači sa svojim odgovorom kada nema ništa da prijavi. Sve dok se klijent brine da neprekidno ima otvoren zahtev anketiranja, on će primati informacije od servera ubrzo nakon što postanu dostupne. Na primer, ako Fatma u svom čitaču ima otvorenu aplikaciju za razmenu veština, taj čitač će postaviti zahtev za ažurne informacije i čekaće odgovor na taj zahtev. Kada Iman pošalje predavanje o ekstremnom spustu monociklom, server će primetiti da Fatma čeka ažurne informacije i poslaće odgovor koji sadrži novo predavanje za njen zahtev na čekanju. Fatmin čitač će primiti podatke i ažurirati ekran da prikaže predavanje.

Da bi se sprečilo isticanje konekcije (njeno prekidanje zbog neaktivnosti), tehnika dugačkog anketiranja obično zadaje maksimalno vreme za svaki zahtev, nakon kog će server svejedno odgovoriti, čak i ako nema ničega o čemu bi izvestio, a nakon toga će klijent započeti nov zahtev. Periodično restartovanje zahteva čini ovu tehniku i izdržljivijom, omogućavajući klijentima da se oporave od privremenih prekida konekcije ili problema sa serverom.

Zauzet server koji koristi dugačko anketiranje mogao bi imati hiljade zahteva na čekanju i otvorenih TCP konekcija. Node, koji olakšava upravljanje velikim brojem konekcija, bez pravljenja posebne niti izvršavanja za svaku, dobar je izbor za takav sistem.

HTTP interfejs

Pre nego što počnemo da dizajniramo server ili klijenta, razmislimo o tački gde se oni dodiruju: HTTP interfejsu preko kog će komunicirati. Koristićemo JSON kao format tela zahteva i odgovora. Kao i za server datoteka iz poglavlja 20, trudićemo se da pametno koristimo HTTP metode i zaglavlja. Interfejs se vrti oko putanje `/talks`. Putanje koje ne počinju sa `/talks` koristiće se za isporučivanje statičnih datoteka – HTML i JavaScript koda za sistem na klijentskoj strani.

Zahtev `GET` za `/talks` vraća JSON dokument poput ovoga:

```
[{"title": "Unituning",
  "presenter": "Jamal",
  "summary": "Modifying your cycle for extra style",
  "comments": []}]}
```

Novo predavanje se pravi postavljanjem zahteva `PUT` URL-u kao što je `/talks/Unituning`, gde je deo nakon druge kose crte naslov predavanja. Telo zahteva `PUT` treba da sadrži JSON objekat koji ima svojstva `presenter` (predavač) i `summary` (rezime).

Pošto naslovi predavanja mogu da sadrže razmake i druge znakove koji se obično ne javljaju u URL-u, znakovni nizovi naslova moraju biti šifrovani funkcijom `encodeURIComponent` pri izgradnji takvog URL-a.

```
console.log("/talks/" + encodeURIComponent("How to Idle"));
// → /talks/How%20to%20Idle
```

Zahtev da se napravi predavanje o stajanju u mestu na uniciklu (engl. *idling*) mogao bi izgledati ovako:

```
PUT /talks/How%20to%20Idle HTTP/1.1
Content-Type: application/json
Content-Length: 92

{"presenter": "Maureen",
"summary": "Standing still on a unicycle"}
```

Takvi URL-ovi podržavaju i zahteve `GET` za preuzimanje JSON-ove predstave predavanja i zahteve `DELETE` za brisanje predavanja.

Komentari se predavanju dodaju pomoću zahteva `POST` za URL kao što je `/talks/Unituning/comments`, sa JSON telom koje ima svojstva `author` (autor) i `message` (poruka).

```
POST /talks/Unituning/comments HTTP/1.1
Content-Type: application/json
Content-Length: 72

{"author": "Iman",
"message": "Will you talk about raising a cycle?"}
```

Da bi podržali dugačko anketiranje, zahtevi `GET` za `/talks` mogu uključivati dodatna zaglavlja koja javljaju serveru da odloži odgovor ako nema dostupnih novih informacija. Mi ćemo koristiti par zaglavlja koja su obično namenjena upravljanju keširanjem: `ETag` i `If-None-Match`.

Serveri mogu u odgovor da uključe zaglavlje `ETag` (skraćeno od „*entity tag*“ – oznaka entiteta). Njegova vrednost je znakovni niz koji označava tekuću verziju resursa. Klijenti, kada kasnije ponovo budu zahtevali taj resurs, mogu napraviti *uslovljeni zahtev* (engl. *conditional request*) uključujući zaglavlje `If-None-Match` čija vrednost sadrži taj isti znakovni niz. Ako se resurs nije promenio, server će odgovoriti statusnim kodom 304, što znači „nije izmenjen“, govoreći klijentu da je verzija koju ima u kešu i dalje aktuelna. Kada se oznaka ne poklapa, server će odgovoriti kao i obično.

Nešto takvo nam je potrebno da bi klijent mogao serveru da kaže koju verziju liste predavanja ima, i da server odgovara samo kada je ta lista izmenjena. Ali umesto trenutnog vraćanja odgovora 304, server bi mogao odugovlačiti sa odgovorom i vratiti ga samo kada je nešto novo dostupno ili kada je

zadata količina vremena istekla. Da bismo napravili razliku između zahteva dugačkog anketiranja i običnih uslovljenih zahteva, daćemo im još jedno zaglavlje, `Prefer: wait=90`, koje serveru govori da je klijent voljan da čeka do 90 sekundi na odgovor.

Server će čuvati broj verzije koji će ažurirati svaki put kada se predavanja promene i koristiće je kao vrednost `ETag`. Klijenti prave zahteve poput ovoga da bi bili obavešteni kada se predavanja promene:

```
GET /talks HTTP/1.1
If-None-Match: "4"
Prefer: wait=90

(prolazi vreme)

HTTP/1.1 200 OK
Content-Type: application/json
ETag: "5"
Content-Length: 295

[....]
```

Protokol koji je ovde opisan ne obavlja nikakvu kontrolu pristupa. Svi mogu da komentarišu, menjaju predavanja, pa čak i da ih brišu. (Pošto je internet pun huligana, postavljanje takvog sistema na mrežu, bez dodatne zaštite, verovatno se ne bi dobro završilo.)

Server

Počnimo sa izgradnjom serverske strane programa. Kôd u ovom odeljku pokreće Node.js.

Usmeravanje

Naš server će koristiti `createServer` za pokretanje HTTP servera. U funkciji koja obrađuje nov zahtev, moramo da napravimo razliku između raznih vrsta zahteva (kao što je određeno metodom i putanjom) koje podržavamo. To se može uraditi dugim lancem naredbi `if`, ali postoji i elegantniji način.

Ruter ili usmerivač (engl. *router*) komponenta je koja pomaže u otpremanju zahteva do funkcije koja će ga obraditi. Na primer, ruteru možete reći da se datom funkcijom mogu obraditi zahtevi `PUT` sa putanjom koja se poklapa sa regularnim izrazom `/^\/talks\/([^\/]+)$/` (`/talks/` nakon čega sledi naslov predavanja). Pored toga, to će pomoći u izdvajanju smislenih delova putanje (u ovom slučaju, naslova predavanja), obavijenih zagradama u regularnom izrazu, i njihovom prosleđivanju do funkcije za obradu.

Postoji više dobrih paketa rutera na NPM-u, ali mi ćemo ga ovde napisati sami da bismo prikazali princip.

Ovo je `router.js`, koji ćemo kasnije zahtevati (koristeći `require`) od našeg serverskog modula:

```
const {parse} = require("url");

module.exports = class Router {
  constructor() {
    this.routes = [];
  }
  add(method, url, handler) {
    this.routes.push({method, url, handler});
  }
  resolve(context, request) {
    let path = parse(request.url).pathname;

    for (let {method, url, handler} of this.routes) {
      let match = url.exec(path);
      if (!match || request.method != method) continue;
      let urlParts = match.slice(1).map(decodeURIComponent);
      return handler(context, ...urlParts, request);
    }
    return null;
  }
};
```

Modul izvozi klasu `Router`. Objekat rutera dozvoljava registrovanje novih obrađivača metodom `add` i može da razreši zahteve svojom metodom `resolve`.

Ova druga će vratiti odgovor kada obrađivač bude pronađen, a inače će vratiti `null`. Ona isprobava rute jednu po jednu (redom kojim su definisane) sve dok ne pronađe odgovarajuću.

Funkcije obrađivača pozivaju se sa vrednošću `context` (koja će u našem slučaju biti instanca servera), sa znakovnim nizovima za pronalaženje grupa koje su definisane u njihovom regularnom izrazu i sa objektom zahteva. Znakovni nizovi moraju biti dekodirani za URL jer sirovi URL može da sadrži kodove u stilu `%20`.

Isporučivanje datoteka

Kada se zahtev ne poklapa ni sa jednim tipom zahteva definisanim u ruteru, server mora da ga tumači kao zahtev za datoteku iz direktorijuma `public`. Mogli bismo koristiti server datoteka definisan u poglavlju 20 za isporučivanje takvih datoteka, ali nije nam ni potrebna ni poželjna podrška za zahteve `PUT` i `DELETE` za datoteke, a voleli bismo naprednije mogućnosti kao što je podrška za keširanje. Zbog toga ćemo koristiti pouzdani, provereni server statičnih datoteka sa NPM-a.

Ja sam se odlučio za `ecstatic`. To nije jedini takav server na NPM-u, ali dobro radi i odgovara našim potrebama. Paket `ecstatic` izvozi funkciju koja se može pozvati sa konfiguracionim objektom da bi nastala funkcija za obradu zahteva. Koristićemo opciju `root` da bismo serveru rekli gde da traži datoteke. Funkcija obrađivača prihvata parametre `request` i `response` i može se proslediti direktno do funkcije `createServer` da bi nastao server koji isporučuje *samo* datoteke. Međutim, prvo želimo da proverimo ima li zahteva koje treba da obradimo posebno, pa ga obavijamo drugom funkcijom.

```
const {createServer} = require("http");
const Router = require("./router");
const ecstatic = require("ecstatic");

const router = new Router();
const defaultHeaders = {"Content-Type": "text/plain"};

class SkillShareServer {
  constructor(talks) {
    this.talks = talks;
    this.version = 0;
    this.waiting = [];

    let fileServer = ecstatic({root: "./public"});
    this.server = createServer((request, response) => {
      let resolved = router.resolve(this, request);
      if (resolved) {
        resolved.catch(error => {
          if (error.status != null) return error;
          return {body: String(error), status: 500};
        }).then(({body,
                  status = 200,
                  headers = defaultHeaders}) => {
          response.writeHead(status, headers);
          response.end(body);
        });
      } else {
        fileServer(request, response);
      }
    });
  }
  start(port) {
    this.server.listen(port);
  }
  stop() {
    this.server.close();
  }
}
```

Ovde se za odgovore koristi slična konvencija kao za server datoteka iz prethodnog poglavlja – obrađivači vraćaju obećanja koja se razrešuju kao objekti koji opisuju odgovor. Ona obavija server objektom koji čuva i njegovo stanje.

Predavanja kao resursi

Predložena predavanja sačuvana su u svojstvu `talks` servera, objektu čija su imena svojstava naslovi predavanja. Ona će biti izložena kao HTTP resursi u okviru `/talks/[title]`, pa ruteru treba da dodamo obrađivače koji implementiraju razne metode koje klijenti mogu koristiti da bi radili sa njima.

Obrađivač za zahteve koji dobavljaju (`GET`) jedno predavanje mora da potraži predavanje i da odgovori ili JSON podacima predavanja ili odgovorom koji prijavljuje grešku 404.