Fakultet informatike u Puli

Web aplikacije

Nastavnik: dr. sc. Nikola Tanković

1. poglavlje - Osnove Javascripta kroz node.js

Ishodi poglavlja

1. Razumijevanje osnovnih pojmova i alata.
2. Inicijalizacija Javascript aplikacije.
3. Razumijevanje osnova rada s Javascript paketima.
4. Posluživanje statičkog front-end koda.

Video lekcije uz ovu skriptu

1. Inicijalizacija paketa: https://www.youtube.com/watch?v=M2KXZzfJLeE&list=PLIeA2EYS12RW4obW-O64LsfcS6lhKIHre&index=1
2. Osnove backenda: https://www.youtube.com/watch?v=3kIExyjNzuA&list=PLIeA2EYS12RW4obW-O64LsfcS6lhKIHre&index=2

Što je to node.js?

Najjednostavnije rečeno, `node.js` je proces koji izrvšava Javascript program izvan internet preglednika. Izrađen je korištenjem Googleovog **V8** JavaScript interpretera (koji se koristi u Google Chromeu) koji se izdvojio u zaseban samostojeći modul izvan internet preglednika. Svako računalo koje ima instalirano `node.js` može pokretati Javascript programe (za izvršavanje kroz preglednik nije bila potrebna dodatna instalacija jer je V8 ili slični interpreter dostupan kroz sam preglednik).

Među najprominentnijim korisnicima `node` procesa jesu Netflix, Uber, LinkedIn i mnogi drugi. **Node** odlikuju visoke performanse u izvršavanju web aplikacija zbog arhitekture koja se temelji na događajima (**event-driven**) i korištenja tzv. *non-blocking I/O* operacija, što znači da programski kod ne čeka na ulaze i izlaze s primjerice diska ili mreže.

Što je to *node package manager - npm* ?

Sustav `npm` služi za upravljanje `node` paketima, dakle modulima pisanim u Javascriptu. Sastoji se od istoimenog procesa `npm` koji dolazi sa instalacijom `node` a te online baze javnih i privatnih paketa koje možemo koristiti u projektima. Prilikom odabira paketa potrebno je pripaziti odnosi li se paket na frontend kod (Angular, Vue, React, …) ili pak na node.js i backend. Ponekad se paketi mogu uporabiti u oba slučaja (npr. `moment.js`).

Što je to *express.js* ?

Kako bi lakše razvijali aplikacije u `node.js` okruženju možemo koristiti različite programske okvire (*framework*). **Express** je najpoznatiji takav okvir za `node.js` . Ukratko, pomoću njega lakše i brže možemo definirati ponašanje web aplikacije. Omogućuje nam upravljanje HTTP upitima i definiranje tzv. `middleware` modula koji mogu presresti određene upite i dodati nove funkcionalnosti (npr. autentifikacija, logovi, ...).

Što je to *MongoDB* ?

MongoDB je NoSQL (ne-relacijska) dokument baza podataka. Ne temelji se na poznatoj relacijskoj algebri kao većina baza i nije ograničena unaprijed zadanom shemom. Nota bene: zavisno u namjeni, postojanje sheme može itekako biti velika prednost. Osnovna jedinica pohrane jest dokument koji se specificira koristeći JSON notaciju. Samim time MongoDB veoma je sličan dokument bazi podataka **Firebase**. Razlika u tome je što je **Firebase** komercijalna usluga u oblaku koju pruža Google, dok je MongoDB *open-source* alat.

NoSQL baze poput MongoDB-a omogućuju lakše skaliranje na više procesa, po cijenu napuštanja stroge konzistantnosti. Takve su baze često *eventual consistent* - što znači da ipak postižu konzistentnost, ali ne garantiraju da će to nužno biti održano baš u svakom trenutku.

Najveća razlika u korištenju Mongo baze u odnosu na relacijsku bazu jest nepostojanje tablica i redaka - te samim time nije potreban poseban **ORM** (*Object Relational Mapping*) sloj koji povezuje nativne objekte programskog jezika (npr. instance klase ili Javascript objekte) u retke tablica - podatke iz baze. Umjesto analogije **redak - tablica** koristimo termine **dokument - kolekcija**. Dakle kolekcija je skupina srodnih dokumenata od kojih svaki ima jedinstveni pokazatelj - ID. U Mongu on postoji *defaultno* . Ta se razlika najzornije ilustrira u primjerima gdje za jedan entitet (poput entita **Online narudžba**) u relacijskoj bazu potrebno više tablica (npr. **Zaglavlje** i **Stavke**), dok je u dokument bazi to i dalje jedan objekt - **dokument** u kolekciji.

Primjer jednog dokumenta (primjeti JSON strukturu):

```
{
  "datum": "01.01.2020",
  "kupac": "Kupac d.o.o.",
  "stavke": [
    {
      "rbr": 1,
      "naziv": "Majica XXL",
      "kolicina": 1
    },
    {
      "rbr": 2,
      "naziv": "Hlače L",
      "kolicina": 4
    }
  ]
}
```

Za spremanje dokumenata nije unaprijed potrebno definirati **shemu**, npr. činjenicu da će u dokumentu postojati polje "datum" određenog tipa uz sva ostala polja. To dodatno povlači da svaki dokument može biti i različit (**što i nije uvijek poželjno**).

Again: with great freedom comes great responsibility. Ovo svakako vrijedi i za MongoDB :)

Naposljetku, velika prednost MongoDB-a je što koristi upravo Javascript i JSON u mnogim operacijama baratanja s podacima (čitaj: nema više SQL-a).

Konzolna aplikacija

Instalacija node.js-a

Node.js možemo instalirati pomoću uputa sa službene web stranice (**https://nodejs.org/**). Kako bi se uvjerili u uspješnost instalacije možemo možemo iskoristiti naredbu `npm -v` dostupnu kroz konzolu - Command Prompt (Windows) ili Terminal (MacOS, Linux). Navedena naredba ispisuje inačicu `node` procesa, primjerice `v12.12.0` .

Prva konzolna aplikacija

Za početak, kreirat ćemo najosnovniju **node** aplikaciju koja ispisuje informacije u konzolu (tamo gdje smo ju i pokrenuli).

Primjer:

```
// spremiti u datoteku index.js

console.log("Hello world");
```

Navedenu konzolnu aplikaciju možemo pokrenuti naredbom `node` nakon čega upišemo naziv datoteke (potrebno je biti u istom direktoriju gdje se nalazi datoteka `index.js`):

```
$ node index.js
# Hello world
```

⭐ Inicijalizacija novog web projekta

Paket **express** možemo instalirati koristeći `npm` , ali to činimo za svaki projekt zasebno, samim time potrebno je prvo inicijalizirati novi projekt pa tek onda u projektu uključiti `express` paket i ostale potrebne pakete:

```
$ mkdir novi_projekt  # kreiramo folder s novim projektom
$ cd novi_projekt  # ulazimo u direktorij

$ npm init  # inicijalizacija novog node projekta, dovoljne su `default` postavke

$ npm install express --save  # instalacija express paketa
```

Za korištenje naprednijih Javascript funkcionalnosti (poput npr. paketa), potreban nam je i **Babel** prevoditelj:

```
$ npm install @babel/core @babel/node @babel/preset-env --save-dev
```

Kako bi inicijalizacija **Babela** bila potpuna potrebna je i njegova konfiguracija. Moramo kreirati datoteku `babel.config.json` unutar glavnog direktorija projekta:

```
// spremiti ovo u "babel.config.json" datoteku - pripazi da datoteka počinje s točkom!
{
  "presets": ["@babel/preset-env"]
}
```

Osim `express` a koristit ćemo i `nodemon` modul tako da se aplikacija tijekom razvoja sama **restart-a** na svaku promjenu koda (jako korisno :)):

```
$ npm install nodemon --save-dev
```

te ćemo promijeniti konfiguracijsku datoteku `package.json` sa uputim kako će `npm` pokretati naš projekt:

```
// promijeniti u package.json

{
  ...
  "scripts": {
    "serve": "nodemon --exec babel-node src/index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  ...
}
```

Time je **npm** u dana uputa da prilikom pokretanja koristi folder `src` i u njemu glavnu datoteku `index.js`, samim time moramo ih kreirati u našem projektu:

```
# napraviti novi folder `src` i u njemu ovo datoteku `index.js`

console.log("Hello world");
```

Sada možemo pokrenuti našu aplikaciju sa `npm run serve`.

Primjetimo kako se automatski aplikacija nanovo pokreće ukoliko dođe do promjene u `src/index.js`.

Prva jednostavna web aplikacija

Jednom kada je projekt inicijaliziran, možemo u `src/index.js` zamijeniti sa **Hello world** web aplikacijom:

```
import express from 'express';

const app = express()  // instanciranje aplikacije
const port = 3000  // port na kojem će web server slušati

app.get('/', (req, res) => res.send('Hello World, ovaj puta preko browsera!'))

app.listen(port, () => console.log(`Slušam na portu ${port}!`))
```

Ponavljanje: u kreiranju ove male aplikacije koristili smo se **arrow** funkcijama. **Arrow** funkcije su skraćeni način kako u Javascriptu definirati funkciju, prisjetimo se tri načina na koja u JS-u možemo definirati funkciju:

```
function f1(a, b) {
  return a + b;
}
// ili
let f2 = function(a, b) {
  return a + b;
}
// ili
let f3 = (a, b) => {
  return a + b;
}
// ili, ako je samo jedan izraz u tijelu funkcije
let f4 = (a, b) => a + b;
```

U ovo prvoj aplikaciji definirana je samo jedna HTTP ruta: `/` - početna ruta. Kao odgovor na zahtjev za ovom rutom ispisujemo odgovor pomoću `res` objekta - drugog parametra funkcije. Dakle naša aplikacija definirana je u varijabli `app` koja ima definiranu metodu `get` za posluživanje HTTP GET zahtjeva. Metoda `get` prima dva parametra:

- prvi parametar tipa `string` - naziv rute
- drugi parametar tipa `function` - *callback* funkcija koja će obraditi rezultat, ta funkcija mora imati barem 2 parametra (važna je njihova pozicija, naziv je proizvoljan):
  - `req` - Javascript objekt s informacijama o zahtjevu (**https://expressjs.com/en/4x/api.html#req**)
  - `res` - Javascript objekt za oblikovanje odgovora (**https://expressjs.com/en/4x/api.html#res**)

Naša prva aplikacija definira samo jednu rutu: `/` te će za svaku drugu rutu **express** automatski vratiti HTTP 404 grešku.

Ukoliko želimo, možemo proizvoljno dodavati nove rute u aplikaciju:

```
import express from 'express';

const app = express()  // instanciranje aplikacije
const port = 3000  // port na kojem će web server slušati

app.get('/', (req, res) => res.send('Hello World, ovaj puta preko browsera!'))
app.get('/student', (req, res) => res.send('Ruta za studente...'))
app.get('/primjer/student', (req, res) => res.send('Ugnježdena ruta'))

app.listen(port, () => console.log(`Slušam na portu ${port}!`))
```

Osnove rada s Javascript paketima

Kako bi struktura naše aplikacije bila urednija, poželjno je kod odijeliti u Javascript pakete. Za razliku od paketa instaliranih preko **npm**a, možemo definirati i vlastite Javascript pakete dostupne za `import`.

Za primjer, odijelit ćemo definiranje ruta u zaseban paket `routes.js`:

```
// src/index.js

import express from 'express';
import routes from './routes';  // . označava da tražimo modul u istom direktoriju gdje se nalazi ovaj modul

const app = express()  // instanciranje aplikacije
const port = 3000  // port na kojem će web server slušati

app.get('/', routes.home)
app.get('/student', routes.student)

app.listen(port, () => console.log(`Slušam na portu ${port}!`))
```

```
// src/routes.js

let home = (req, res) => res.send('Hello World, ovaj puta iz paketa!')
let student = (req, res) => res.send('Ruta za studente preko novog paketa.')

export default { home, student }  // na kraju navodimo koje JS objekte treba eksportati
```

Posluživanje front-end koda

Do sada smo vidjeli primjere gdje naša aplikacija ispisuje tekst u preglednik na pojedinoj ruti, no kako možemo napraviti da aplikacija isporuči cjelokupni HTML/CSS/JS kod? Rute čiji se sadržaj ne mijenja nazivamo **statičnim rutama** - one se također mogu dohvaćati kroz **express** na sljedeći način:

1. Unutar projekta definira se direktorij koji će sadržavati statične datoteke (obično `public` ili `static`)
2. Konfigurira se **express** ruta (obično `/`) koja poslužuje taj direktorij na sljedeći način:

```
// src/index.js

import express from 'express';
import routes from './routes';

const app = express()
const port = 3000
const staticMiddleware = express.static('public')  // direktorij sa statičkim datotekama

app.use('/', staticMiddleware)

app.get('/dinamicki', (req, res) => res.send(`Dinamički odgovor`))

app.listen(port, () => console.log(`Slušam na portu ${port}!`))
```

Alternativno, posluživanje front-end koda uopće ne mora ići kroz **express** već se za to može definirati zasebni poslužitelj (npr. **Zeit**, ili kroz Python sa komandom u terminalu `python -m http.server .`), ali onda moramo brinuti o CORS-u kada šaljemo upite na express backend. Vaš projekt može u potpusnosti odvojiti front-end (npr. Js/Vue) i back-end (Js/Express) kod u dva zasebna repozitorija i web servera. Front-end web server isporučuje samo statičke fajlove i može biti na jednoj domeni (npr. **https://www.mojapp.com**) dok back-end kod može biti na sasvim drugoj domeni (npr. **https://api.mojapp.com**). Više o tome u sljedećim poglavljima.

Zadaci za vježbanje

⚠️ Riješeni zadaci predaju se na **http://bit.ly/wa-zadatak**, a ukoliko je rješenje cijeli projekt, možeš ga predati kao zip, ili još bolje postaviti na svoj Github profil i predati link

1. Isprobaj template za backend projekt sa uputama na **https://github.com/fipu-nastava/wa-backend-template**. Koristiti će ti u sljedećim zadacima.
2. **(WA-101)** Implementirati jednostavni backend web aplikacije koja definira samo tri rute:
   - `/datum` - vraća trenutno datum/vrijeme formatirano u obliku `dd.mm.yyyy HH:MM`.
   - `/prognoza` - vraća nasumičnu rečenicu u obliku "Danas će biti [sunčano/kišovito/oblačno]"
   - `/` - vraća kratak opis na kojim se adresama nalaze prve dvije rute

   Možete se poslužiti **npm** paketom `moment`.

   Kao rješenje dovoljno je dostaviti samo `index.js`.
3. (**WA-102**) Implementirati backend jednostavne web aplikaciju koja definira nekoliko ruta:
   - `/dodaj` - dodaje nasumično odabrani broj između 0 i 100 u listu brojeva (može biti prazna u

početku)
    - `/dohvati` - dohvaća listu i ispisuje ju kao HTTP odgovor u obliku gdje su brojevi odvojeni razmakom, npr. `1 23 53 21 99` (ispis ide u browser, ne u konzolu)
    - Potrebno je da se `callback` funkcija za svaku rutu nalazi u zasebnom paketu.
    - Na koji način moduli mogu dijeliti zajedničku listu brojeva? Naime, `/dodaj` mora dodati u istu listu koju `/dohvati` vidi. **Hint**: možda novi paket? :)

  Kao rješenje dostaviti cijeli projekt u `.zip` datoteci.

4. (**WA-103**) Konfigurirati `public` direktorij za isporučivanje statičkih datoteka. Napraviti mali primjer **Vue** aplikacije/komponente koja se nalazi u `index.html` datoteci i koja poziva **back-end** kod iz zadatka 2. te prikazuje real-time datum i prognozu. Možeš se poslužiti templateom dostupnim na **https://github.com/vuejs/vuejs.org/blob/master/src/v2/examples/vue-20-hello-world/index.html** Za pozivanje koda na back-endu koristiti `fetch` (**https://dev.to/shoupn/javascript-fetch-api-and-using-asyncawait-47mp**)

5. **(WA-104)** Implementiraj novi back-end koji sadrži podatke o 5 proizvoljnih studenata. Svaki student ima svoj JMBAG, ime, prezime i godinu studija. Implementirati sljedeće rute:
    - `/studenti` - vraća listu svih studenata u aplikaciji
    - `/studenti/first` - vraća samo prvog studenta
    - `/studenti/last` - vraća samo zadnjeg studenta