Pristup izradi web aplikacija pomoću NodeJS-a

Bogdan, Ivan

Sveučilište
Sjever

Završni rad br. 658/MM/2019

Pristup izradi web aplikacija pomoću NodeJS-a

Bogdan Ivan, 0791/336

Varaždin, rujan 2019. godine

Sveučilište
Sjever

Odjel za Multimediju, oblikovanje i primjenu

Završni rad br. 658/MM/2019

Pristup izradi web aplikacija pomoću NodeJS-a

Student

Ivan Bogdan, 0791/336

Mentor

Vladimir Stanisavljević, mr. sc.

Varaždin, rujan 2019. godine

Predgovor

Ovu temu sam pripremio najviše kako bi si približio dubine koncepata tehnologija koje svakodnevno koristim na radnom mjestu. Surfajući internetom davno sam nabasao na Feynman-ovu tehniku kojom on proces prenošenja znanja smatra najplemenitijom i najboljom vještinom učvršćivanja istog. Moj generalni pristup i ton ovog rada stoga će biti namijenjen široj publici hrvatskog govornog područja sa namjerom da dobiju pristojno razumijevanje djelovanja i tempa djelatnosti modernog back-end i front-end web developmenta koristeći NodeJS i ostale web development alate, pogotovo onom svojstvene JavaScript-u.

Sažetak

Putanja web development trendova je dugoročno nepouzdana zbog dinamične prirode samog Weba i tehnologija kojim ga razvijamo. Temeljni zadatak developera danas je da ostane u doticaju sa razvojem novih tehnologija koje, ako su pravilno korištene, služe kao moćan set alata za izgradnju modernog Weba kakav danas svi aktivno koristimo i poznajemo. Univerzalan pristup razvoju development vještina unutar Weba jest učenje koncepta i arhitekture na kojoj Web djeluje, odnosno globalne mreže računala koju zovemo Internet, koji su već toliko utemeljeni da u samoj srži uvijek ostaju isti.

Web dijelimo na dva temeljna segmenta kada pričamo o developmentu, koja imaju jasno definiran i standardiziran model komunikacije, a to su klijentski i poslužiteljski Web development. Ovaj rad unutar praktičnog dijela objedinjuje kompletno softversko rješenje koje se proteže od klijenta do poslužitelja koji komunicira sa bazom podataka, i pritom koristi aktualno neke od najmodernijih dostupnih web development alata. Svaka od korištenih tehnologija je zasebno detaljno opisana kroz teoretski dio rada, a glavnu ulogu u cijeloj komunikaciji između klijenta, na kojem koristimo VueJS kao razvojni okvir, i MongoDB baze podataka obavlja NodeJS server, koji je ujedno i središnja tema ovog rada.

Abstract

The trajectory of web development is unreliable in long term because of the dynamic nature of Web itself and technologies with which we develop it. Core task of a web developer today would be to stay in touch with evolution of new technologies which, if properly used, serve as a mighty toolset for building modern Web as we all actively use and know of. Universal approach to web development skills in Web is learning concepts and architecture on which Web works upon, in other words, the global network of computers which we call the Internet, which have been so grounded that they always stay the same inside their core.

We divide the Web on two fundamental segments when we talk about development, with both of those two having clearly defined and standardized model of communication, and those are client-side and server-side Web development. This paper from inside of its practical part provides a complete software solution which extends from client to the server side which communicates with the database, and it does that by using some of the most modern of web development tools available. Each of those used technologies is separately described through the theoretical part of this paper, and the main role in whole of communication between client, on which we use the Vue JavaScript framework and the MongoDB database, is done by NodeJS server, which is also the central topic of this paper.

Popis korištenih kratica

HTML HyperText Markup Language
Sintaksa za obilježavanje hipertektualnih dokumenata.
CSS Cascading Style Sheets
Sintaksa za obilježavanje hipertektualnih dokumenata.
JS JavaScript
Sintaksa za obilježavanje hipertektualnih dokumenata.
ECMA European Computed Manufacturer's Association
Firma zaslužna za standardiziranje JavaScripta
W3C World Wide Web Consortium
Konzorcij za standardizaciju i razvijanje tehnologija na Webu
URL Uniform Resource Locator
DNS Domain Name Service
SQL Structured Query Language
LAN Local Area Network
ARPANET Advanced Research Programs Agency Network

Sadržaj

1. Uvod

Internet ili web su dva često nezgrapna pojma kada tek želimo dobiti tehnički adekvatan dojam kako oni zapravo funkcioniraju. Samo razlikovanje ta dva pojma često prouzrokuje zbunjenost iako ih sami ne rijetko koristimo naizmjenice. U digitalnoj eri u kojoj živimo moglo bi se reći da je bazično znanje o internoj arhitekturi interneta temeljno opće znanje, baš zato što je postao neizostavan alat naše svakodnevnice.

Internet je danas medij za komunikaciju između nekoliko milijardi uređaja, i njime već napajamo kompleksne sustave koji su temelji naše globalne civilizacije, bilo to cijela svjetska ekonomija, ili prometni sustavi najvećih svjetskih metropola. Preko Interneta nesmetano održavamo videokonferencije sa poslovnim klijentima bilo gdje se oni nalazili, ili se natječemo na ogromnim natjecanjima zasnovanim na videoigrama koje će uskoro postati dijelom Olimpijskih igara zbog popularnosti. Web, kao samo jedna porcija Interneta, ili jedan od tih navedenih sustava nam omogućuje vizualno optimalan i intuitivan, ali i kompleksan set višenamjenskih alata na nekim od svojih lokacija, ili stranica, koji nam može poslužiti kao virtualni dnevni boravak unutar kojeg se družimo sa svojim prijateljima iz cijelog svijeta, sve zbog inteligentnog modela na kojem je zasnovan i koji nam to omogućava.

Lako se izgubiti u apstrakciji samog koncepta Interneta ili Weba sa laičke perspektive, međutim ne trebamo tražiti dalje od vlastitog doma, unutar kojeg često imamo više uređaja spojen na Internet, kako bi si predočili temeljne koncepte na kojima je Internet zasnovan. Taj dio ćemo itekako pokriti ovim radom, i to već na samom početku jer nije rijetka pojava da čak i iskusni developeri ne razumiju gdje se njihova stranica nalazi nakon što je učitaju na Web ili kako s njom komuniciramo.

Kao developerima, gotovo je nemoguće da svojim znanjem obuhvatimo sve što Internet može ponuditi jer bi takvo što zahtijevalo vrsno poznavanje raznih segmenata proizvodnje od kojih je svaki nekome prije nas bio životno djelo, a i nikako ne bismo mogli doći do takvog znanja bez da već na putu graničimo sa inženjerstvom tih disciplina i djelatnosti, što je priča sama za sebe.

Prilikom ulaska u svijet web developmenta, te ako je entuzijazam prisutan, često mislimo da znamo više nego što znamo i da smo već sposobni činiti čudesa već sa osnovnim alatima poput HTML-a, CSS-a i JS-a. Naravno, čudesa nam i jesu dostupna koristeći samo te tri tehnologije, pogotovo u formatu u kojem su dostupne danas, te su upravo ta tri alata već dugi niz godina, pa od samog osnivanja Weba temelji skoro svih web stranica i aplikacija danas. Svaki od navedenih alata su kroz prethodnih deset godina doživjeli svoj preokret kojim niti jednom od njih ne uzimamo za zlo što su kroz proces razvijanja imali svoje slabe točke koje smo mi kao developeri morali znati

zakrpati, već ih cijenimo što tek kako ih otkrivamo shvaćamo za što su sposobni, te što sve objedinjuju.

Naime, pristupanjem u razvoj Weba, često smo zahvalni svojim prethodnicima na čijim ramenima stojimo unutar ove discipline, čiji je životni rad učinio naš proces developmenta bar malo lakšim. Makar se mnogo razvijenih tehnologija više ne koristi, sigurno je svaka od njih poslužila razvoju koncepta koji danas napajaju najnaprednije Web aplikacije, od kojih svaka objedinjuje desetke ako ne i stotine tih tehnologija.

Najbizarnije u cijeloj povijesti razvoja Weba sigurno mora biti skok koji je napravio JavaScript u trenutku kada se, pojavom NodeJS-a, odvojio od klijentske strane Weba, i postao dostupan i na serveru, a primjenu pronašao i u mnogim drugim okruženjima. Upravo NodeJS, kao pokretačko okruženje (eng. *runtime environment*) na serveru, središnji je fokus ovog završnog rada, jer njegova popularnost u Web developmentu rapidno raste i koristi se već u development tvrtkama svjetskih razmjera poput Netflix-a ili PayPal-a i sličnih kao temeljni alat za programiranje na poslužiteljskoj strani. [1]

Sukladno rapidnom razvoju JavaScripta i na klijentskoj i na poslužiteljskoj strani, razvile su se i NoSQL baze, od kojih MongoDB pristupa skladištenju podataka u formatu i strukturi koja je najprikladnija *mindset*-u JavaScript developera, a to je BSON format, koji je u principu samo binarna reprezentacija JSON-a koji je prirodan svim JS developerima. Ova kombinacija tehnologija znatno olakšava JS pristup web developmentu od klijentske do poslužiteljske strane, uključujući i samu komunikaciju sa bazom podataka.

Prilikom rada na završnom radu većina izvora informacija će mi biti oni izvori koje koristim svakodnevno radom unutar te branše, kao što su MDN (Mozilla Developer Network) [2], službene specifikacije navedenih tehnologija, te praktični savjeti iskusnih developera putem članaka ili odgovora na Stack Overflow [3] stranici, a i samo iskustvo u struci će sigurno poslužiti kao temeljni izvor informacija ili bar vodilja. Unutar rada će biti prikazan i mali praktični rad zasnovan na većini spomenutih tehnologija unutar rada. S obzirom da je i klijentska i serverska strana toliko kompleksna da bi se mogla tek temeljito obraditi u nekoliko ovakvih radova, rad se najviše fokusira na serversku stranu razvoja Weba, iako će unutar praktičnog segmenta rada biti korištena i klijentska aplikacija. Svrha ovog rada jest da istražimo koncepte iza alata koje developer svakodnevno koristi prilikom razvoja Web aplikacija na poslužiteljskoj strani, te da dobijemo cjelokupan dojam kako Web danas funkcionira.

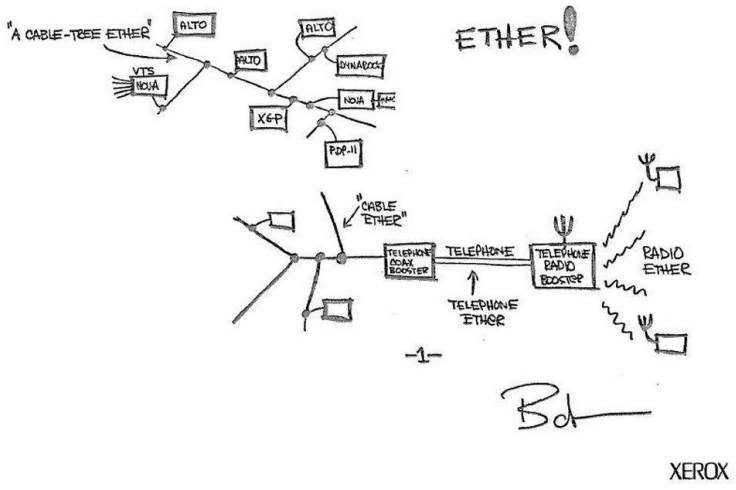
2. Komunikacija između klijenta i poslužitelja

Kako bi uopće razumjeli središnju temu ovog rada, moramo bar bazično objasniti neke od temeljnih koncepta na kojima su Internet i Web zasnovani. Također, sama priroda razvoja web aplikacija u principu se temelji najviše oko komunikacije između pretraživača (eng. *browser*), koji je u osnovi klijentska (eng. *front-end*) strana web developmenta, i servera, čiji development nazivamo poslužiteljskim (eng. *back-end*).

2.1. Internet

Kada bi definiciju Interneta htjeli svesti na prostu ili jednostavnu rečenicu, ona bi nam obznanila da je Internet globalno rasprostranjena mreža povezanih fizičkih računala koji međusobno komuniciraju. Ta definicija u principu na banalan i vješt način zaista opisuje rad cijelog Interneta, međutim, tu banalnu definiciju moramo razgranati na nekoliko temeljnih koncepta kojim već uspostavljamo adekvatnu sliku rada cijelog interneta, ponajviše blisku web developmentu.

Ako bi razložili tu definiciju na proste jednostavne oblike, globalnu mrežu bi mogli predstaviti kao nekoliko računala međusobno povezanih **ethernet**[4] kablom unutar neke prostorije, odnosno jednog građevinskog objekta. Specifično, taj tip mreže nazivamo **Local Area Network (LAN)**[5].



Slika 1: Bob Metcalfe-ova originalna schema etherneta

Računala unutar ovakvog zatvorenog tipa mreže možemo spojiti na više načina, bilo to koristeći **'hub'** ili **'switch'** uređaj.[6] Količina podataka koja se može prenijeti kroz računalnu mrežu u jedinici vremena nazivamo ***bandwidth***[7] i tu vrijednost izražavamo u ***Hz*** (herc). Taj oblik mreže može uspostaviti osnovnu komunikaciju kakva je danas dostupna na Internetu, ali kako

bismo je eksterno spojili na ostatak interneta moramo se povezati sa ISP-em (Internet Service Provider) koristeći **'router'** koji preusmjerava pakete koje šaljemo putem interneta ovisno o **IP adresi**[8] (internet protokol) koja je označena na svakom paketu. Router je zapravo svojevrsno računalo koje u svojoj memoriji ima pohranjen registar IP adresa na koji čita i piše sve IP adrese koje su mu dostupne. Podatke koje šaljemo putem interneta nazivamo porukama (eng. *message*), ali zbog fleksibilnosti same komunikacije i puta od pošiljatelja do primatelja, tu istu poruku prilikom slanja dijelimo na manje **pakete**[9], te ih na samom odredištu ponovno gradimo u inicijalnu poruku. Cijela komunikacija je zasnovana na **TCP**[10] protokolu koji je zbog raznih parametara zaslužan za vjerodostojnost i integritet same poruke. Svaki paket prilikom svoj putovanja do destinacije prolazi kroz nekoliko routera koji ga preusmjeravaju do cilja, i put koji paket prođe od jednog routera do drugog nazivamo **skok** (eng. *hop*)[11].

```
 1     3 ms     3 ms     5 ms  172.20.0.3
 2     7 ms    11 ms     6 ms  77.77.212.85
 3     5 ms     3 ms     *     77.77.201.72
 4     7 ms     *        9 ms  77.77.197.136
 5     5 ms     *        7 ms  77.77.201.0
 6    21 ms    18 ms    21 ms  89.216.5.68
 7    21 ms    20 ms    17 ms  bg-ne-r-1-hg4-0.sbb.rs [89.216.5.251]
 8    29 ms    29 ms    25 ms  carnet.cix.hr [185.1.87.65]
 9     *        *        *     Request timed out.
10     *        *        *     Request timed out.
11     *        *        *     Request timed out.
12    24 ms    26 ms    24 ms  legen.unin.hr [193.198.63.12]
```

Slika 2: Traceroute sa računala u Mostaru, BiH do web stranice unin.hr

2.2. Web

Web je s druge strane uspostavljen kao softverski dio povrh Internetske arhitekture kojem pristupamo pomoću web pretraživača. Atomska gradivna jedinica Weba je web stranica od kojih je on u principu sav sastavljen, te pravu mrežu ostvarujemo kada povežemo više takvih stranica putem ***hyperlinkova***. Momentalno ovu mrežu čini oko milijardu i pol stranica[12]. Svaka od tih stranica je spremljena na zasebnoj IP adresi, te je običaj registrirati domenu za svaku pojedinačnu web stranicu putem **DNS**-a (Domain Name Service)[13] kako bi toj adresi mogli pristupiti u čitljivijem formatu nego što je skup znamenkih u IP adresi (www.unin.hr – 193.198.63.70:443, IPv4)[14].

2.2.1. Klijent

Klijentska strana Weba, a sukladno time i web developmenta, ograničena je onim što se događa u pretraživaču. Pretraživač djeluje kao softver koji koristi resurse računala na kojem se nalazi kako bi obavljao zadatke i procese potrebne za vizualno renderiranje web stranica ili aplikacija koje dohvati sa servera. Svaki pretraživač unutar sebe radi na sličnim principima i glavna mu je funkcionalnost da čita HTML, CSS i JS datoteke. Pretraživači su s vremenom postali snažni softveri sposobni za mnoge različite zadatke poput prikazivanja različitih sadržaja i formata, ne samo onih svojstvenih Webu. Pretraživač koji je najskloniji web developmentu i samim time središnjoj temi ovog rada je **Google Chrome**, čiji je **V8** motor (eng. *engine*) interno korišten u NodeJS pokretačkom okruženju (eng. *runtime environment*)[15].

2.2.2. Poslužitelj

Temeljni element Weba osim same web stranice bi bio 'spremnik' na kojem je ta web stranica pohranjena, a taj spremnik nazivamo server. Server je pokrenut, odnosno server je u principu obično računalo, međutim prilagođen je potrebi Weba, te je često znatno jači od standardnih prijenosnih računala. Najrasprostranjeniji oblik serverskog okruženja je **LAMP**[16] model čiji akronim stoji za **Linux** kao kernel na kojem je server pokrenut, **Apache** HTTP server kao softver koji omogućava primanje zahtjeva i slanje odgovora, **MySQL** kao relacijska baza i **PHP** (Postprocessed Hypertext Markup Language) kao programski kod kojim dohvaćamo i spremamo podatke u bazu podataka, uz brojne mnoge mogućnosti. LAMP kao akronim može također predstavljati i drugačiji stog tehnologija, ali koji u principu obavljaju isti posao. Također za LAMP model postoji i varijanta na drugim operativnim sustavima poput Windowsa i MacOS-a. Iako je LAMP jako versatilan, pouzdan i skalabilan, unutar ovog rada ćemo se posvetiti svježijoj i praktičnijoj alternativi kao što je NodeJS, koji u isto vrijeme obavlja i ulogu servera i alata za dinamičnu obradu podataka na tom serveru, kao i komunikaciju s bazama podataka. Primarna razlika između ove dvije vrste servera jest što je NodeJS zasnovan na jednom ***threadu*** te podržava ne blokirajuću I/O izvodnju preko asinkronog modela, što mu omogućuje da primi ogromnu količinu zahtjeva te ih paralalelno prosljeđuje ***worker pool***-u koji su zaduženi za obavljane težih komputacijskih zadataka[17], a LAMP omogućava korištenje više *thread*-ova čime paralelno može izvršavati više zahtjeva odjednom. Svaki od pristupa ima svoju specifičnu primjenu, i svoje prednosti i mane. Korisno je još spomenuti da je LAMP primarno lako vertikalno skalabilan, dok je NodeJS često horizontalno skalabilan.

2.3. Komunikacija između klijenta i poslužitelja

Komunikacija između klijenta i poslužitelja prilično je jednostavna po pitanju samog koncepta jer se cijel koncept zasniva na *request – response*[18] modelu kojim je klijent uvijek taj koji šalje zahtjev poslužitelju, a poslužitelj na svaki zahtjev vraća odgovarajuću poruku. Tekst koji sadrži *hyperlinkove* nazivamo *hypertext*, te je po tome i nazvan **HTTP** (HyperText Transfer Protocol) protokol na kojem Web funkcionira. Taj protokol omogućuje korisniku da pri unosu domene web stranice zapravo šalje HTTP **GET** zahtjev serveru da pošalje nazad resurs namijenjen tom **URL**-u (Uniform Resource Locator), kojem je format, odnosno tip sadržaja **'text/html'**, što znači da server prilikom GET zahtjeva nazad šalje datoteku nazivom index.html, osim kad je drugačije konfiguriran. Moderna inačica HTTP-a je proširena sa SSL certifikatom (**HTTPS**) što osigurava asinkronu enkripciju sadržaja putem javnih i privatnih ključeva, kako nikakav posrednik ne može osluškivati komunikaciju.[19]

Za svaku ponuđenu izlaznu točku (eng. *endpoint*), server ima pripremljen sadržaj koji šalje nazad klijentu, a za svaku ne podržanu rutu protokol je vratiti famoznu poruku sa statusom **'404: Page Not Found'**. HTTP metoda svakog zahtjeva je sadržana unutar HTTP zaglavlja, pored ostalih korisnih informacija kao što bi bili **CORS**, **'Content-Type'**, **origin** i slično. [20]

2.3.1. RESTful API

Pripremanje i format dostupnosti podataka na serveru prilikom zahtjeva od klijenta je također standardiziran, i taj standard u principu nije ključan, ali se koristi na većini web servera danas koji dostavljaju ikakve kompleksnije podatke ili koji imaju izložen ikakav **API** (*Application Programming Interface*)[21]. Taj standard, odnosno arhitekturu servera u odnosu na *request – response* model zovemo **RESTful** (Representational State Transfer).

Komunikacija između klijenta i poslužitelja je uvjetovana HTTP protokolom, a unutar sebe ima dostupnih šest metoda[22]:

- **GET** zahtjev koristimo kako bi dohvatili podatke sa servera te ovu metodu smatramo sigurnom zato što ni u jednom trenutku ne mijenjamo podatke koje primamo, ili drugim riječima, GET zahtjev je idempotentan što znači da bi na isti zahtjev uvijek dobili iste rezulatate dok se podatci na serveru ne promijene jednom od sljedećih metoda. Nakon svakog uspješnog zahtjeva, server vraća status `200 OK` skupa sa odgovorom tipično u JSON ili XML formatu.

Slika 3: Primjer GET zahtjeva za učitavanje stranice unin.hr

- **POST** zahtjev koristimo kako bi napravili novi podatak na resursu koji bio po REST principima morao biti isključivo nova instanca već postojećeg formata dokumenta na serveru. Klijent poslužitelju šalje resurs tipično u JSON ili FormData formatu. Prilikom uspješnog slanja zahtjeva server vraća status kod `201 Created`, ali i dokument koji je spremljen samom tom metodom.
- **PUT** metodu koristimo kako bi uredili već postojeći podatak na serveru. Ako taj podatak već ne postoji, možemo konfigurirati server da napravi novi, ali pritom po REST standardu klijentu vraćamo `201 Created`. U slučaju da je nije kreiran novi podatak već samo uređen postojeći vraćamo `200 OK` ili `204 No Content` ako podatak ne postoji, ali ga server nije nadodao.
- **PATCH** metoda služi kako bi uredili postojeći podatak, ali samo parcijalno, te bi ispravnija praksa bila koristiti PATCH umjesto PUT zahtjeva jer PUT zapravo funkcionira tako da na mjestu podatka koji uređuje zapravo stavlja posve novi podatak,

što može biti ne sigurna operacija. PATCH treba također koristiti sa oprezom, jer još nije podržan na nekim platformama poput IE8, Tomcat-a, PHP-a i Djanga.

- **DELETE** metodu koristimo kada želimo izbrisati resurs sa servera. Ako je metoda prošla uredno, odnosno izbrisala odabrani entitet, onda server vraća poruku `200 OK`, sa `null` vrijednosti tijelu (eng. *body*) odgovora.
- **OPTIONS** metodu šaljemo kako bi provjerili koje su metode uopće dopuštene na serveru, ali je OPTIONS metoda isto i poruka poslana prije svakog zahtjeva kako bi se klijent uvjerio da je CORS (Cross-Origin Resource Sharing).

Jako koristan alat za testiranje ovih metoda ili uopće API izlaznih točaka je **Postman** kojem je temeljna funkcionalnost da služi kao grafičko sučelje za **cURL** (*client* URL) zahtjeva, koji koristimo kako bi imitirali HTTP zahtjeve direktno iz konzole, čija je praktična upotreba generalno kako bi se testirale izlazne točke na API-u, ili kako bi se rekreirali zahtjevi koji su prošli neuspješno, čime bi se lakše mogao otkriti problem samom zahtjeva ili servera. [23]

```
curl 'https://www.unin.hr/' -H
'Connection: keep-alive' -H
'Pragma: no-cache' -H
'Cache-Control: no-cache' -H
'Upgrade-Insecure-Requests: 1' -H
'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3939.0
Safari/537.36' -H
'Sec-Fetch-User: ?1' -H
'Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/w
ebp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.9' -H
'Sec-Fetch-Site: none' -H
'Sec-Fetch-Mode: navigate' -H
'Accept-Encoding: gzip, deflate, br' -H
'Accept-Language: en-GB,en-US;q=0.9,en;q=0.8' -H
'Cookie: pll_language=hr; _ga=GA1.2.1740863133.1570020753;
displayMode=Gray; _gid=GA1.2.170356472.1570804421; _gat=1' --
compressed
```

Programski kod 1: cURL reprezentacija inicijalnog GET zahtjeva za dohvaćanje stranice unin.hr

RESTful nam nalaže da zahtjeve sa klijenta strukturiramo sa svim gore navedenim podatcima tako da server ima sve potrebne informacije da programski vrati odgovor bez da mora pamtiti informacije o samom zahtjevu.

3. NodeJS

Kao dobar uvod u naredna poglavlja valja spomenuti jedan ‘zakon’ kruži Webom, iako nepoznatog autora, mnogi ga pripisuju Jeff Atwood-u, tvorcu Stack Overflow-a, stranice koja služi kao jako pouzdan izvor informacija za sve računalno vezane teme. Taj zakon kaže:

‘Svaka aplikacija koja može biti napisana u JavaScriptu,
eventualno će biti napisana u JavaScriptu’

Ta izjava, već danas vidimo da nije tako neostvariva, prvo zbog uzlazne putanje razvoja tehnologija na Internetu općenito, drugo zbog konstantnog razvoja na optimizaciji pretraživačkih enginea koji su zaduženi za izvođenje JavaScripta na pretraživačkom okruženju, te ako je u pitanju Chrome V8 engine na kojem se pokreće NodeJS, i na poslužiteljskom okruženju. Te optimizacije su zaslužne za ogromne brzine izvođenja JavaScripta što ga čini versatilnim i pogodnim za širok spektar primjena.

Prije NodeJS-a, JavaScript smo koristili skoro pa isključivo u pretraživačkom okruženju. Naime, JS kod bi pokrećali tako što bi .js datoteku ubacili pomoću `<script>` taga u .html dokument i JavaScript bi jednostavno funkcionirala. Kako bi bolje objasnili NodeJS, temeljitije ćemo proći kroz proces izvođenja JS-a na klijentu, počevši od sintaktičnog dijela samog JavaScripta.

3.1. ECMAScript

ECMAScript jest specifikacija za skriptne jezike nastala kako bi standardizirala JavaScript, iako JavaScript nije jedini programski jezik koji su je implementirali. Tu specifikaciju također koriste jezici poput JScript-a i ActionScripta. [24]

ECMAScript (ECMA-262) standardizaciju piše firma European Computer Manufacturer Association, te u načelu postavlja standarde koje bi skriptni jezici poput JavaScripta trebali ispuniti. Sam naziv ECMA je odabran zato što je ime JavaScript bilo patentirano od strane Sun firme, danas nazvane Oracle. ECMA standard je ostvario konsenzus između različitih pretraživača i engine-a da jednako interpretiraju JS ili ostale jezike definirane ECMA standardom, što je ogromna prednost jer možemo očekivati da će svaki pretraživač ili engine moći pročitati naš JS kod jednako. Jedino o čemu zapravo trebamo razmišljati kao developeri jest podržavaju li

pretraživači funkcionalnosti iz neke specifične ECMA standardizacije (ES5, ES6, ... , ES9). Mnogi pretraživači još uvijek ne podržavaju sve funkcionalnosti koje nudi ES6, a funkcionalnosti uvedene posljednjim standardizacijama biti će implementirane u pretraživače tek kroz narednih nekoliko godina. ES5 je u cijelosti podržan od strane svih popularnih pretraživača. [25]

ECMAScipt 2015, ili kako je skraćeno zovemo ES6 je sinonim za novu generaciju JavaScipt programiranja. Nakon ES6 standardizacije ECMA nije postala živući standard kao što su HTML ili CSS, već nova verzija izlazi svake godine. ES6+ je developerima omogućio ogroman broj novih funkcionalnosti[26] koje služe kao neizostavan set alata u modernim Web development trendovima. Neki od tih alata su:

- **Objektno-orijentirane klase** – iako u principu služe više kao sintaktički šećer povrh već dostupnih objektno-orijentiranih funkcionalnosti, ova nadogradnja olakšava čitljivost koda i prilagođenija je univerzalnom pristupu kreiranja klasa i instanciranja objekta unutar drugih programskih jezika.
- **Arrow funkcije** – primarna svrha ovog tipa funkcija je da uredi kod i da smanji utrošeno vrijeme na pisanje funkcija što je u jeziku poput JavaScripta koji omogućuje funkcionalno programiranje iznimno bitno.

```
// ES5
var add = function (num, num2) {
return num1 + num2;
}
// ES6
var add = (num1, num2) => num1 + num2
```

Programski kod 2: Usporedba standardne funkcije sa posebnom ES6 arrow funkciojm

Također jedna od velikih prednosti arrow funkcija je što nam ne ograničava scope riječi this unutar funkcije kao što bi to uradile standardne funkcije, što samo po sebi zna zbuniti mnogo developera.

- **Moduli** – definiramo ih kao manje jedinice nezavisnog koda dostupnog za višekratnu upotrebu. Dok ova funkcionalnost nije uvedena morali smo koristiti sisteme poput CommonJS-a ili AMD-a (Asynchronous Module Definition) kako bi unutar JavaScript datoteka uvezli kod iz drugih datoteka. Sa ES6 ta funkcionalnost je dostupna nativnom JavaScriptu koristeći `import` sintaksu, ili koristeći `export` ako bi htjeli učiniti kod dostupnim van iste datoteke. ES6 sa sobom donosi i nova sintaktička unaprjeđenja u odnosu na CommonJS iako je već postao standard u NodeJS okruženjima.

- **Template literals** - možemo ih gledati kao dinamične stringove, koji nam omogućavaju da unutar stringova nadodamo varijable koristeći posebnu sintaksu:

```
const ime = 'Ivan';
console.log('Moje ime je ' + ime);
console.log(`Moje ime je ${ime}`)
// Oba primjera u konzolu ispisuju jednak rezultat
```

Programski kod 3: Usporedba standardnih stringova i template literal stringova

Također jedna od korisnih funkcionalnosti template literals-a naspram regularnih stringova jest što omogućuje pisanje vrijednosti u više redova.

- **Varijable** – var kao jedini način deklariranja varijabla unutar JS-a imao je mnogo propusta. Skoro svi ti propusti su pokriveni ES6 specifikacijom sa `const` i `let` deklaracijama, a skoro svaki je redovno prouzrokovao glavobolju kod developera. Kada bismo inicijalizirali varijablu unutar `window` globalnog konteksta, zapravo bismo je pridružili globalnom objektu kao jedno od svojstava, dok je Node taj problem riješio na specifičan način čineći kreiranje globalnih varijabli dostupnih isključivo putem `global` globalnog objekta o čemu ćemo detaljno pisati kasnije. Varijable `let` i `const` za razliku od var su ograničene blokom uvjetovanim vitičastim zagradama tako da ih možemo koristiti unutar petlji, funkcija objekata bez straha da budu dostupne van tog bloka. Jedna od nelogičnosti `var` varijable jest što se može deklarirati više puta, dok bi ponovna deklaracija sa novim tipom varijabli izazvala error. Ne logičnost koja prouzrokuje najviše zabune je vjerojatno **hoisting**. Naime, JS prilikom kompajliranja sve varijable i deklaracije funkcija prvotno spremi u memoriju prije izvršavanja ostatka koda, tako da možemo zamisliti da su deklaracije bez ikakvih pridruženih vrijednosti uvijek dostupne unutar cijelog koda kao `undefined` vrijednosti dok im ne pridružimo neku određenu vrijednost. Također, vrijedi napomenuti da const varijabla za razliku od let i var mora imati pridruženu vrijednost pri samoj inicijalizaciji, i ta vrijednost se ne može promijeniti.

Kako bismo provjerili što je od ECMAScript specifikacije podržano za korištenje unutar NodeJS okruženja, specifične nadogradnje možemo pronaći na web stranici https://node.green/.

Iako je JavaScipt u svojoj jezgri zapravo ECMAScript, odnosno sve što još JavaScript u sintaktičkom smislu nudi gradi se iz te jezgre, kao potpuni programski jezik ga smatramo tek kad samim funkcionalnostima unutar sintakse pridodamo sve ono što mu pretraživač, ili poslužitelj, kao globalni kontekst omogućuje. To su moćni setovi alata, odnosno **aplikacijskih programskih sučelja** (API) koji su nam omogućeni kao domaćinski objekti (eng. *host objects*) . [27] dostupni

ovisno o okruženja u kojem se nalazimo. Na klijentskoj strani nam je pomoću pretraživača omogućen **`window`** object unutar kojeg su dostupne sve metode metode i svojstva svojstvena pretraživaču koji također djeluje kao globalni ***scope***, a preko poslužiteljske strane, odnosno unutar NodeJS-a koristimo **module**, unutar kojeg su dostupni razni globalni objekti, a kontekst samog modula je lokalan. JavaScipt kao skriptni jezik ovisi o svom okruženju baš zbog tih API-eva, zato što za razliku od drugih programskih jezika uopće ne sadrži koncepte poput *input/outputa*, već se oslanja na host objekte unutar svog okruženja da bi uopće komunicirao sa 'vanjskim svijetom'. JS možemo pronaći u više okruženja ,osim klijentskog i poslužiteljskog. Neka od tih okruženja su Adobe Acrobat, Adobe Photoshop, Internet-of things uređaji, NoSQL open-source baza podataka Apache CouchDB, pa čak računalno okruženje GNOME koje se koristi kao grafičko sučelje za najpopularnije Linux distribucije. [28]

3.2. V8

Kada izvodimo kod iz neke .js datoteke na pretraživaču, taj kod je ne razumljiv računalu čiju procesorsku snagu zapravo koristimo. Upravo zbog toga nam je potreban neki medij koji će prevesti naš JS kod u onaj koji je srodan računalu. Taj medij je engine koji interno koriste pretraživači kako bi izvodili JS, a kod u koji ga prevode je **strojni kod** (eng. *machine code*). Taj kod najbolje možemo zamisliti kao skup nula i jedinica, odnosno binarni brojevni sustav, iako su u njemu dostupni i heksadekadski znakovi. Najpoznatiji takav engine je Chrome V8 engine na kojeg ćemo se isključivo fokusirati u ovoj sekciji, iako svi motori interno funkcioniraju slično, te se moraju prilagoditi mnogim standardima kao što su ECMAScript ili slični. Izlazak V8 enginea je bio monumentalni trenutak u razvoju JS-a jer je cijel napisan u nižem C++ jeziku te je prvotno bio interno potreban za svrhe Google Mapsa, jer su tadašnji *engine*-i bili spori za takav zadatak. [29]

Prilikom prevođenja JS-a u strojni kod, engine izvodi veliki niz optimizacija kako bi se postigle maksimalne brzine, te su ljudi koji rade na tim optimizacijama često među boljim inženjerskim ekspertima u području računalne znanosti s obzirom na opus posla. Prvu od tih optimizacija obavlja parser koji prođe kroz cijel JS kod te iz njega izvuče pojedinačne elemente koje kasnije spremi u JSON objekt AST (Abstract Syntax Tree) modela. Enginei su prilagođeni takvom modelu podataka te ga naknadno prosljeđuju **JIT compileru** koji djeluje kao kombinacija **realtime interpretera** i standardnog **kompajlera**. Razlog ovo pristupa jest što i interpreter i kompajler imaju svojih prednosti koje objedinjuju sve te prednosti u JIT kompajleru radi postizanja najveće moguće brzine. Koncept rada funkcionira tako što parsirana, odnosno prevedena AST JSON struktura JS koda ulazi u interpreter koji ogromnim brzinama u stvarnom vremenu (eng. *realtime*)

čita liniju po liniju koda te ga prevodi u **bytecode** koji nije toliko nizak (eng. *low-level*) jezik kao što je strojni kod, ali se može izvršavati na JS interpreteru, a kompajler tijekom izvođenja osluškuje interpreter te na licu mjesta kompilira kod koji se može optimizirati i dostavlja ga u strojnom kodu, što znači da kompajler preuzima izvedbu JS od interpretera nakon što kompajlira sav neizveden ostatak koda. JIT kompajler koji korist V8 interpreter zove se **TurboFan**. Osim compile optimizacije, V8 se brine za alokaciju memorije, sakupljanje smeća od kojih svaki znatno pridonosi brzini izvođenja JS-a. [30]

3.3. NPM

Node Package Manager, neizostavan alat u bilo kakvom obliku JavaScript developmenta današnjice. Omogućuje nam da putem paketa unutar svog projekta uključimo eksterne **knjižnice** (eng. *library*) ili radne okvire (eng. *framework*), dostupne na online NPM registru. Ove pakete unutar projekta spremamo u posebnu mapu **node_modules** koju ignoriramo prilikom spremanja stanja projekta na Gitu, ali sve informacije o paketima koje za jedan projekt koristimo držimo u korijenu projekta u datotekama koje se zovu *package.json* i *package-lock.json*.[31] Package.json drži temeljne informacije o projektu, paketima koje projekt sadrži, te ima posebnu sekciju za npm skripte koje možemo iskoristiti da bi pokrenuli izgradnju svog projekta za lokalni development ili build za puštanje na produkciju. Kada na postojećem projektu želimo instalirati sve ovisnosti, odnosno module koje taj projekt koristi, sve što trebamo je otvoriti terminal u mapi gdje nam je projekt sadržan i pokrenuti komandu `npm install`, koja povuče podatke o svakom korištenom modulu i dohvati ih sa NPM registra, te ih pritom instalira. API iz svakog pojedinačnog paketa dostupan nam je kao JS modul koji možemo učitati unutar JS koda.

3.4. NodeJS

NodeJS je pokretačko okruženje otvorenog koda (eng. *open* source) na poslužitelju koje koristi Google Chrome-ov V8 engine za interpretiranje i kompajliranje JavaScripta. NodeJS nije ograničen platformom na kojoj se pokreće te ga vrlo jednostavno možemo instalirati na OS X, Windows i Linux uređajima. NodeJS aplikacije zbog svog asinkronog principa rada omogućavaju ogromnu količinu zahtjeva kojim ne blokiraju svoj I/O, te se zbog toga ove aplikacije najčešće koriste kod aplikacija koje rade u stvarnom vremenu. NodeJS nakon instalacije možemo koristiti unutar terminala čime terminal pretvaramo u REPL (Read-Eval-Print-Loop) stanje kojim možemo izvoditi JS operacije.[32]

3.5. Globalni objekti i osnovni moduli

NodeJS, kao i pretraživači, ima specifičan set globalnih objekata i metoda svojstvenih izvršavanju koda na serveru. Za razliku od pretraživačkog globalnog window konteksta, riječi this ne referencira na global objekt, već je scope unutar svakog modula lokalan i dostupan isključivo tom modulu, a global objekt koristimo kako bi deklarirali globalne varijable dostupne kroz cijelu aplikaciju. S ovim varijablama trebamo znati što radimo i koristiti ih kao iznimku, ne kao normu, zato što one zauzimaju memoriju tijekom rada cijele aplikacije, zato jer je dostupna unutar svake .js datoteke bez obzira je li se koristi. Osim global objekta, postoji mnogo drugih metoda i objekta koji su specifični za NodeJS okruženje, ali i onih koji su svojstveni JavaScriptu generalno. Treba spomenuti da su neki od NodeJS globalnih objekata dostupni isključivo unutar modula, odnosno ne mogu se koristiti u konzoli ili slično. Osim globalnih objekata, NodeJS interno ima pohranjenih nekoliko modula koje koristimo kako bi u stvari postigli temeljne funkcionalnosti NodeJS-a kao kompletno serversko rješenje. Module unutar NodeJS-a unosimo (eng. *import*) putem CommonJS API-a za module kojim svaki modul prišijemo na `exports` objekt te ga unosimo unutar drugog modula putem `require()` metode čiji parametar je ime modula u slučaju da je modul već instaliran putem npm registra, ili NodeJS modul. Ako želimo unijeti lokalni modul, kao parametar stavljamo put do tog modula. Moduli koje NodeJS ima interno pohranjene su[33]:

- **File System** `('fs')` – Node za razliku od pretraživača nije ograničen izvođenju samo u pretraživačkom okruženju već ima pristup lokalno pohranjenim podatcima na uređaju na kojem se pokreće. Kako bi pristupili tim uređajima koristimo fs modul koji nam omogućava razne metode za sinkrono i asinkrono čitanje i pisanje datoteka.
- **HTTP** `('http')` – ovaj modul nam omogućava da postignemo isto što bi u LAMP modelu postigli korištenjem Apache-a. HTTP modulom inicijaliziramo server unutar NodeJS aplikacije te njime prisluškujemo zahtjeve koje server dobije kada ih sa klijenta ili Postmana uputimo na port koji smo mu zadali. Pri dolasku zahtjeva, izvršimo *callback* funkciju koja ima dva parametra *request* i *response* od kojih *request* sadrži sve informacije o zahtjevu, kao na primjer HTTP zaglavlja, dok *response* koristimo kako bi putem tog servera vratili odgovor klijentu putem raznih dostupnih metoda.
- **URL** `('url')` – jako koristan i jednostavan modul koji koristimo kako bi parsirali, odnosno preveli URL putem kojeg nam je upućen zahtjev na serveru. Koristeći ovaj modul svaki primljeni URL razlažemo na strukurirane segmente unutar JSON objekta.

```
// Prikaz rada metode .parse url modula
const url = require('url');
console.log(url.parse('https://www.unin.hr/', true);
// Output konzole:
{
     protocol: 'https',
     slashes: true,
     auth: null,
     host: 'www.unin.hr',
     port: null,
     hostname: 'www.unin.hr',
     hash: null,
     search: null,
     query: {},
     pathname: '/',
     href: 'https://www.unin.hr/'
}
```

Programski kod 4: Konzolni ispis sadržaja objekta dobivenog parisranjem URL-a

- **Path** (`'path'`) – ovaj modul unutar sebe sadrži metode i svojstva svojstvena radom sa puteivma (eng. *path*) direktorija na računalu na kojem je Node pokrenut.
- **Querystring** (`'querystring'`) – ovaj modul koristimo kako bi temeljitije parsirali, odnosno preveli upitne karakterne nizove (eng. *query string*) unutar URL-ova.
- **Utilities** (`'util'`) – skup pomoćnih funkcija za rad unutar NodeJS-a

Unutar poglavlja vezanog za praktični rad napravit ćemo najjednostavniji oblik aplikacije koja koristi ove module, te tu aplikaciju prevesti u Express radni okvir kako bi usporedili korist apstrakcije povrh nativnog NodeJS-a i navedenih osnovnih modula. Praktični rad će također nadopuniti ovo poglavlje sa instalacijom i korištenjem NodeJS-a, a samim time i NPM-a.

3.6. Express

Prilikom prvotnih upoznavanja sa NodeJS arhitekturom ili ekosustavom, gotovo je nemoguće ne pronaći Express unutar konteksta svojstvenog NodeJS-u. Express, radni okvir NodeJS-a, momentalno je jedan od najpopularnijih paketa dostupnih na NPM registru[34] sa preko 10 milijuna skidanja tjedno, te je postao dostupan samo godinu dana nakon izlaska samog NodeJS-a. Koristeći Express možemo si znatno olakšati pisanje Node aplikacija time što imamo apstrakcije povrh često ponavljanih funkcija.

4. Rad s bazama podataka

Baze podataka koristimo kako bi strukturirali podatke odgovarajućim modelima ili standardima kako bismo podatcima iz tih baza mogli efikasno pristupiti koristeći računalne upite. Različiti modeli baza podataka su pogodni za različite operacije za pretraživanje i spremanje podataka. Baze podataka kao termin su najzastupljenije unutar računalne znanosti, a na Webu se danas najviše koriste dva različita modela baza podataka, a to su relacijske i dokumentno-orijentirane baze podataka. Treba napomenuti da zbog različitih beneficija navedenih tipova baza podataka, nije neobično da jedna kompanija koristi više različitih baza podataka za različite svrhe pohrane podataka.

4.1. Relacijske baze podataka

Relacijske baze podataka su najrasprostranjeniji tip baza podataka na Webu danas, a kao koncept su nastale 1970. godine u članku Edgar F. Codd-a. [35] Ovaj tip baza podataka je jako efikasan za spremanje velikih količina podataka jer se zbog svoje relacijske prirode podatci ne moraju duplicirati. Najpoznatiji jezik za pristupanjem ovakvog tipa baza podataka je SQL (Structured Query Language), te su danas najkorištenije baze otvorenog koda MySQL i MariaDB. Podatci unutar ovih baza su spremljeni u formatu tablica sa striktno definiranim shemama (eng. *scheme*). Shema jedne tablice je navedena kao set kolumni, dok je svaki red instanca te sheme, odnosno jedan podatak unutar te tablice. Svaki podatak ima svoj jedinstveni identifikator, odnosno ključ kojim se razlikuje od svakog drugog podatka u toj tablici. Relacije se postižu tako što svakom podatku možemo pridodati i strani ključ koji referencira na podatak ili podatke iz druge tablice. Podatke dohvaćamo koristeći SQL upite navodeći specifične komande, vrijednosti i tablicu iz koje dohvaćamo podatke.

```
CREATE TABLE IF NOT EXISTS korisnici (
     id INT AUTO_INCREMENT PRIMARY KEY,
     ime VARCHAR(255) NOT NULL,
     prezime VARCHAR(255) NOT NULL,
     datum_rodjenja DATE,
     napravljeno TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Programski kod 5: SQL upit kojim kreiramo tablicu u bazi podataka

Vrijednosti unesene kao kolumne u tablici:

- **`id`** – INT znači da je uneseni podatak cijeli broj, AUTO-INCREMENT znači da ćemo prilikom dodavanja svakog novog podatka dizati ovaj broj za vrijednost jedan u odnosu na prethodni podatak u tablici, PRIMARY KEY znači da nam id kolumna unutar tablice stoji kao primarni ključ.
- **`ime, prezime`** – VARCHAR(255) služi kako bi spriječili unos više od 255 znakova, a NOT_NULL za kolumnu znači da ne smije primati null vrijednosti, odnosno da je ovo polje traženo (eng. *required*).
- **`datum_rođenja`** – DATE koristimo kako bi spremili datum kao tip podatka
- **`napravljeno`** – TIMESTAMP je tip podatka koji označava vrijeme od UNIX epohe, DEFAULT koristimo kako bi postavili zadani unos u slučaju da podatak nije unesen, a CURRENT_TIMESTAMP je taj podatak što unosimo kao zadanu vrijednost.

```
INSERT INTO korisnici(ime, prezime, datum_rodjenja)
VALUES
     ('Ivan', 'Bogdan', '1997-10-14'),
     ('Filip', 'Vujčić', NULL),
     ('Bruno', 'Lukaček', NULL),
     ('Luka', 'Koren', NULL)
```

Programski kod 6: SQL upit kojima unosimo podatke u tu tablicu

	id	ime	prezime	datum_rodjenja	kreirano
Edit Copy Delete	5	Ivan	Bogdan	1997-10-14	2019-10-14 13:07:24
Edit Copy Delete	6	Filip	Vujčić	*NULL*	2019-10-14 13:07:24
Edit Copy Delete	7	Bruno	Lukaček	*NULL*	2019-10-14 13:07:24
Edit Copy Delete	8	Luka	Koren	*NULL*	2019-10-14 13:07:24

Slika 4: Grafički prikaz tablice unutar phpMyAdmin sučelja

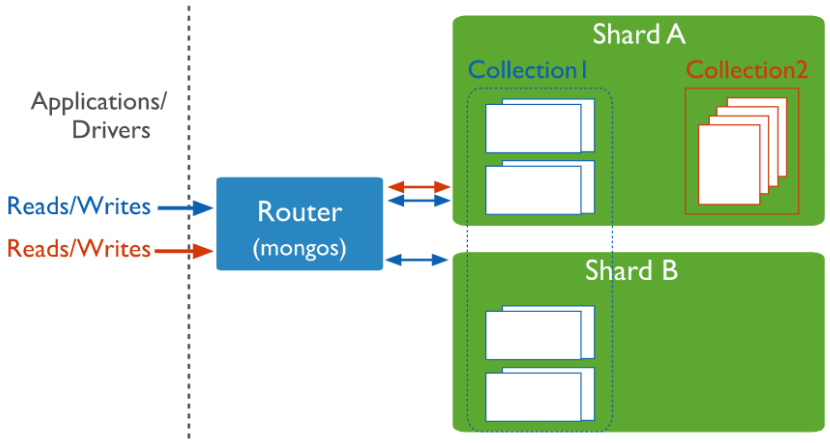
Kako bi naknadno nadodali relacije za druge podatke koristili bi strane ključeve unutar druge tablice koji bi referencirali na `id` ove tablice.

4.2. Dokumentno-orijentirane baze podataka

NoSQL (None Structured Query Language) su baze podataka koje ne koriste SQL kako bi pristupali i modificirali podatke, već spremaju svoje podatke u različite modele među kojima ćemo se najviše fokusirati na dokumentno-orijentirani model. Dokumentno-orijentirane baze podataka imaju znatno drugačiji pristup spremanju podataka od relacijskih baza, što im daje puno veću fleksibilnost za vrijednosti svakog podatka, ali zbog nedostatka relacija velika količina podataka se duplicira. Dokumenti u ovom tipu baze podataka su u principu najsličniji JSON objektima. Oni ne koriste predefiniranu shemu već je svaki podatak u kolekciji, odnosno nizu dokumenata fleksibilan da sadrži podatke koji mu pripadaju. Ova fleksibilnost dolazi s cijenom jer za svaki pojedinačni podatak u memoriji imamo spremljen i ključ (eng. *key*) i vrijednost (eng. *value*), dok bi u relacijskoj bazi podataka ključ bio jedanput naveden kao zaglavlje tablice u kojoj se podatci spremaju. [36]

4.2.1. MongoDB

MongoDB je dokumentno-orijentirana NoSQL baza podataka otvorenog koda koja podatke zapisuje u serijaliziranom BSON formatu. BSON format je binarna reprezentacija JSON-a, te je po svom obliku identična standardnom JSON objektu, ali također podržava tipove podataka. MongoDB možemo instalirati na različitim operacijskim sustavima, te mu pristupamo koristeći grafičko *shell* sučelje. Kao alternativu za tradicionalne tablice kod relacijskih baza, MongoDB koristi kolekcije (eng. *collections*) unutar kojih sprema više dokumenata koji bi se u relacijskoj bazi mogli usporediti najbliže sa redovima unutar tablice. Velika prednost MongoDB-a naspram relacijskih baza jest *denormalizing* što omogućuje da unutar jednog ključa za svaki dokument možemo spremiti više podataka, odnosno polje, ili čak posve novi dokument, a često je slučaj da koristimo više novih dokumenata unutar jednog ključa kako ćemo navesti u primjeru. Podatci spremljeni u BSON formatu imaju limit od 16 megabajta što ih ograničava da koriste prekomjernu RAM memoriju ili troše previše bandwidth-a. Slično relacijskim bazama, svaki dokument u MongoDB-u sadrži jedinstveni ključ koji osigurava da je svaki dokument unutar kolekcije jedinstven. Taj ključ je programski *hashirani* 12-bajtni niz znamenki, a koristeći ga olakšavamo horizontalno skaliranje baze. MongoDB koristi *sharding* kako bi skalirao baze horizontalno, odnosno distribuira podatke istih ili različitih kolekcija unutar *shard-ova* te raspolaže podatcima u njima koristeći *routere*. [37]



Slika 5: Prikaz rada shadrova (clustera) unutar MongoDB arhitekture

Najzastupljenija baza podataka korištena unutar NodeJS aplikacija je MongoDB najviše zato što je svojim formatom pristupačna JavaScript developerima, ali i zato što neke njene metode i upitne (eng. *query*) varijable djeluju slično i dijele nazive sa metodama nativnim samom JavaScriptu.

MongoDB na Windows sustavima instaliramo lokalno spremajući .exe datoteku za instalaciju dohvaćenu sa njihove web stranice. U narednom poglavlju gdje ćemo ovu bazu podataka koristiti u praktične svrhe, koristimo MongoDB v4.2, te slijedimo svaki korak u instalaciji, te naposljetku idemo u Program Files/MongoDB direktorij gdje smo instalirali bazu podataka te paralelno pokrećemo mongod.exe i mongo.exe, od kojih prva služi za sam server postavljen lokalno na našem računalu, a drugu koristimo kao *shell* kojim zapravo upravljamo bazom podataka.

Kako bismo napravili bazu sličnu onoj prethodnoj koristeći MySQL, unutar *shell*-a unosimo sljedeće komande:

```
use unin        // ovom komandom prelazimo u navedenu bazu ili
                // stvaramo novu ako ona ne postoji
db.korisnici.insertMany([{  // isto tako pravimo kolekciju
ime: 'Ivan'                 // korisnici ako nije već kreirana
prezime: 'Bogdan',
datum_rodjenja: 14.10.1997
},
{ ime: 'Filip', prezime: 'Vujcic' },
{ ime: 'Bruno', prezime: 'Lukacek'},
{ ime: 'Luka', prezime: 'Koren'}
])
```

Programski kod 7: NoSQL upit kojim popunjavamo tablicu istim podatcima kao u SQL primjeru

Na stranici https://www.mongodb.com/download-center/compass možemo skinuti aplikaciju Compass koja nam služi kao intuitivno grafičko sučelje za spajanje na MongoDB i klasične CRUD

funkcionalnosti unutar baze podataka. MySQL slična varijanta ovakvog sučelja bi bila phpMyAdmin putem kojeg možemo unositi upite ali i ručno pospremati ili uređivati podatke u bazi. Compass unutar sebe također omogućava širok set alata kojima možemo mjeriti performanse i vršiti mnoge analize brzine i efikasnosti naše baze, kao i mnoštvo drugih mogućnosti.

MongoDB nam je također omogućio grafički urednu web aplikaciju Atlas koju koristimo kao oblak (eng. *cloud*) servis putem kojeg možemo lako skalirati našu bazu putem interneta koristeći *clustere*. Cluster možemo gledati kao instancu naše baze podataka iako ga koristimo kako bi objedinili već spomenute *routere* i *shardove* za lagodno horizontalno skaliranje naše baze. Putem Atlasa također jako jednostavno možemo naš cluster spojiti na Compass aplikaciju, shell i na web aplikaciju putem specijalnog URL-a. Unutar naše web aplikacije koristimo i MongoDB ODM (Object Data Modelling) *library*, od kojih je najpoznatiji Mongoose koji djeluje kao apstrakcija povrh samog MongoDB-a, kao što je Express apstrakcija povrh NodeJS-a. Mongoose nam omogućuje definiranje sheme za naše podatke, relacije za naše dokumente, ima visoku kompatibilnost sa radom u NodeJS-u, dostupna je bazična validacija podataka te mnogo drugih pomoćnih metoda i svojstava. [38]

5. Praktični rad

Kako bismo predstavili spomenute tehnologije, napraviti ćemo praktični oblik svih spomenutih tehnologija uključujući neke koji su univerzalno svojstveni praktičnom radu u developmentu. Svrha aplikacije je unošenje korisnika, odnosno dokumenata u MongoDB bazu koja se nalazi na Mongo Atlas *clusteru*. Sa klijentske strane koristimo frontend JS radni okvir Vue, pomoću kojeg smo vizualno omogućili korisniku unos, pregled, i standardnu modifikaciju podataka. Tip aplikacije svojim mehanizmima zadovoljava CRUD (Create, Read, Update, Delete) standard time što smo za svaku od slova u akronimu na serveru omogućili izlazu točku koju korisnik sa klijentske strane 'gađa' HTTP zahtjevima sa potrebnim zaglavljima. Povrh NodeJS-a koristimo Express razvojni okvir koji nam je znatno omogućio smisleno strukturiranje aplikacije u MVC format koji smo naknadno dodatno prilagodili potrebama aplikacije. Povrh Vue-a za stiliziranje korisničkog sučelja koristimo Vuetify koji je razvojni okvir za korištenje komponenti zasnovani na Material dizajn principima koje je osmislio Google, te koji su standardi dizajna usvojeni na svim Google-ovim proizvodima. Kako bi se uopće unutar JS koda, odnosno Vue koda spojili na MongoDB Atlas *cluster* koristimo Mongoose ODM razvojni okvir koji znatno olakšava rad sa Mongo bazama podataka za JavaScript developere tako što pristupa dokumentima kao da su JSON objekti što je prirodan format svakom iskusnom JavaScript developeru.

Prilikom razvoja koristimo Git VCS (Version Control System) kako bi pratili razvoj aplikacije i napredak od početka do kraja putem *commit*-ova, te kako bi učinili kod iz aplikacije dostupnim na GitHub online repozitoriju, u slučaju potrebne evaluacije ili slično. Sav učitani kod će biti kod otvorenog karaktera, što znači da svaki developer ima svu slobodu da skine kod i koristi ga za vlastite svrhe, jer osobno mislim da je taj princip upravo zaslužan za aktualni razvoj Weba. Linkovi za repozitorije su dostupni na kraju ovog poglavlja.

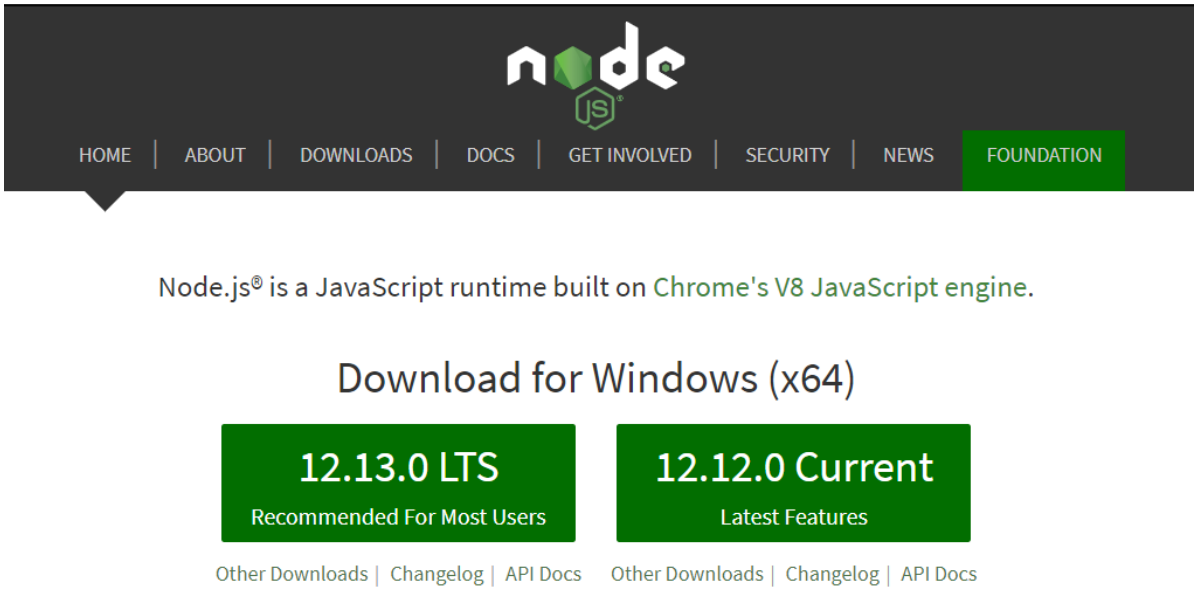
Operativni sustav koji koristim za development generalno i ovu aplikaciju specifično je Windows iako se baziram isključivo na njegovu ekstenziju sa Linux-om koji je najprirodnije okruženje za development većini developera i inženjera na svijetu. Ekstenzija koju koristim zove se WSL (Windows Subsystem for Linux) i ona mi omogućuje da koristim Linux-ov kernel nativno unutar Windows operativnog sustava, koji bi inače trebao biti ili instaliran metodom *dualboot* ili pokrenut putem virtualne mašine na instaliranom operativnom sustavu što je ne praktičan pristup jer paralelno trošimo kompjuterske resurse za dva pokrenuta operativna sustava.

Još jedan jako bitan alat koji je korišten prilikom developmenta je Visual Studio Code uređivač teksta koji mnogim ozbiljnim developerima diljem svijeta služi kao glavni alat unutar kojeg pišu i uređuju kod. Iako većina uređivača teksta, odnosno koda, omogućava ekstenzije zbog kojih je i VSCode najviše zastupljen, kodiranje u VSCode-u mi se osobno činilo na prvu najpraktičnijim.

Najkorisnije ekstenzije za ovakve uređivače su one koje omogućuju grafičko praćenje promjena kad je repozitorij inicijaliziran u projektu, automatsko dovršavanje riječi koje su svojstvene formatu datoteke u kojoj pišemo. Iznimno je bitno i vizualno uređivanje sučelje radi lakše snalažljivosti jer kod velikih aplikacija broj linija koda zna preći milijune.

5.1. Inicijaliziranje NodeJS aplikacije

Nakon što zaista uđemo u NodeJS filozifiju i koncepte, korištenje istog se čini kao da ne može biti lakše. Kako bi koristili NodeJS na našem računalu prvo ga moramo instalirati putem službene NodeJS stranice (https://nodejs.org/en/). Sve što trebamo raditi kako bi instalirali NodeJS je slijediti upute u instalaciji jer je u pitanju standardna .exe datoteka za instalaciju. Unutar te instalacije također dobijemo i NPM CLI (Command Line Interface) pomoću kojeg ćemo instalirati dodatne pakete, odnosno module u našoj aplikaciji.



Slika 6: NodeJS službena web stranica

Nakon što smo uspješno instalirali NodeJS možemo ga početi koristiti odmah na dva načina. Jedan od načina bi bio unošenjem riječi `node` unutar terminala čime ulazimo u REPL stanje, putem kojeg nam je omogućeno korištenje NodeJS-a i svih dostupnih funkcionalnosti ali putem direktnog unosa operacija u konzolu. Drugi način korištenja NodeJS-a bi bio unošenjem iste `node` komande te put do datoteke koju želimo pokrenuti i.e. `node index.js`.

Datoteka koju smo time pokrenuli NodeJS-om unutar sebe ima dostupne sve Node module i metode, odnosno sve što Node kao pokretačko okruženje nudi, isto kao što pretraživač kad ga

koristimo kao pokretačko okruženje za JavaScript nudi klijentu svojstvene globalne metode i kontekst.

5.2. Bazična NodeJS aplikacija

Iako praktičnim radom nastojimo ostvariti određene funkcionalnosti za koje ćemo koristiti slične ali drugačije, optimiziranije alate, svakako moramo prikazati standardni pristup postizanjem istog koristeći 'sirove' tehnologije, bez ikakvih razvojnih okvira. Sve što želimo napraviti je najjednostavniji oblik NodeJS servera koji radi i osluškuje zahtjeve na određenom portu dok mu rad ne prekinemo. Prilikom izrade aplikacije koristimo 'localhost' lokalni server kojem možemo pristupiti na IP adresi '127.0.0.1' te koristiti razne 'port-ove' za različite aplikacije, kao što u ovom slučaju koristimo broj 3000 za vrijednost porta.

```
// UNOS OSNOVNIH MODULA
const http = require('http');

// KREIRAMO BAZIČNI SERVER
const server = http.createServer((req, res) => {
  res.end('Zdravo svijete!');
});

// POSTAVLJAMO SERVER DA SLUŠA NA LOCALHOST PORTU 3000
server.listen(3000, '127.0.0.1', () => {
  console.log(`Slušamo na portu 3000!`);
});
```

Programski kod 8: Najjednostavniji oblik NodeJS servera

Ovaj server nam ne omogućava mnogo, odnosno pristupanjem ovom serveru preko klijentske strane putem http://localhost:3000/ URL adrese jedino što nam je ispisano na stranici je tekst 'Zdravo svijete!'. Kako bi proširili ovaj server sa dodatnim funkcionalnostima, odnosno kako bi rasporedili izlazne točke po RESTful principu najviše moramo koristiti objekte *request* i *response* dobivene unutar *callback* funkcije u `http.createServer()` metodi. Sve što nam je u principu dovoljno za otvaranje novih izlaznih točaka je navedeno unutar *request* objekta koji je sadrži sve potrebne informacije o upućenom zahtjevu, kao što su metoda koja je iskorištena, URL koji je tražen, i slično.

```
const url = require('url');
const querystring = require('querystring');

const server = http.createServer((req, res) => {

// PARSIRAMO URL POMOCU OSNOVNIH MODULA URL I
// QUERYSTRING I IZ NJEGA IZVLAČIMO POTREBNE PODATKE
  const { pathname, query } = url.parse(req.url);
  const { id } = querystring.parse(query);

// ODGOVOR ŠALJEMO OVISNO O IZLAZNOJ TOČKI
  if ( pathname === '/' ) {
    res.writeHead(200, { 'Content-type': 'text/html' });
    res.end('<h1>Naslovnica</h1>')
  } else if ( pathname === '/api') {
    res.writeHead(200, { 'Content-type': 'application/json' });
    res.end({ ime: 'Ivan', prezime: 'Bogdan' });
  } else {
    res.writeHead(200, { 'Content-type': 'text/html' });
    res.end('<h1>Naslovnica</h1>')
  }
});
```

Programski kod 9: Jednostavan oblik servera sa API izlaznim točkama

Na navedenom primjeru već nam je jasnije kako koristimo *req* objekt, odnosno objekt koji služi za primanje informacija, i *res* objekt koji služimo za slanje informacija nazad klijentu ili kome god tko nam je uopće uputio zahtjev.

5.3. ExpressJS

Iako se gore naveden prikaz aplikacije može razložiti na manje module kojim bi sačuvali čitljivost, za takvo što obično ipak koristimo dodatni razvojni okvir povrh NodeJS-a koji sve gore prikazane funkcionalnosti iz Node-ovih osnovnih modela strpa u pažljivo strukturirane apstrakcije koje značajno olakšavaju development i čitljivost koda. Također u ovom stadiju razvoja već počinjemo koristiti *environment* varijable pomoću *dotenv* NPM paketa koji nam omogućuje da sve varijable spremljene u .env datoteku čitamo unutar koda putem `process.env` objekta.

Express, kao NodeJS razvojni okvir nije toliko različit od sirovog oblika NodeJS-a korištenog u prethodnim primjerima te se temeljnim developer znanjem može zaključiti kako kako Express zapravo apstrahira kod u odnosno na NodeJS.

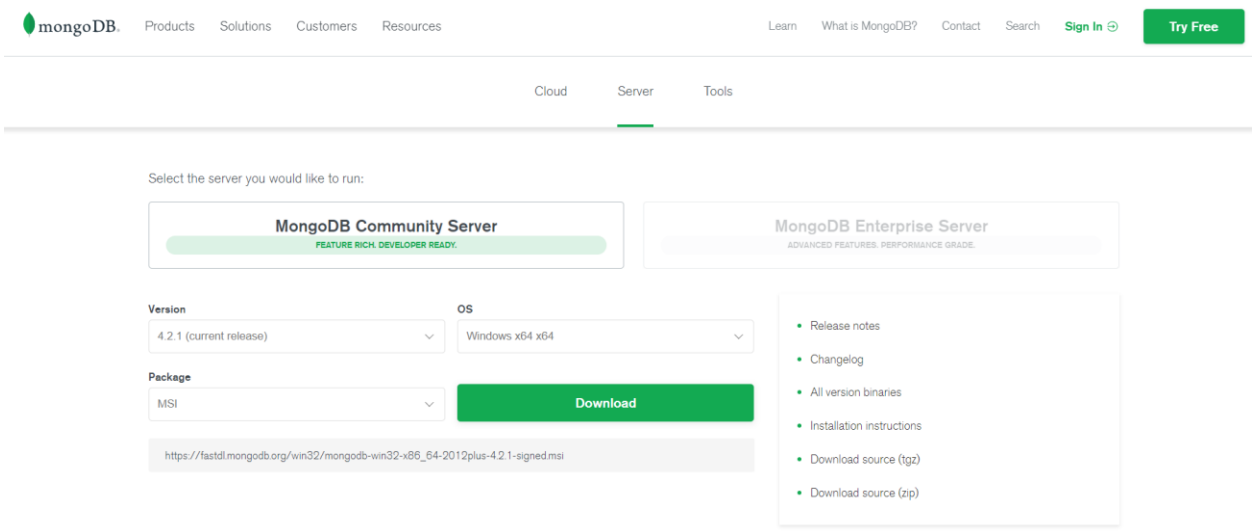
```
const express = require('express');
const app = express();

require('dotenv').config();

// ROUTERI
app.get('/', (req, res) => {
  res.status(200).send('Naslovnica');
});

app.get('/api', (req, res) => {
  res.status(200).json({
    status: 'success',
    data: { ime: 'Ivan', prezime: 'Bogdan' }
  });
});

// IZLAZNI MIDDLEWARE AKO NITI JEDNA OD PONUĐENIH RUTA NIJE POGOĐENA
app.use((req, res) => {
  res.status(404).send('Page not found!');
});

const PORT = process.env.PORT;
app.listen(PORT, '127.0.0.1', () => {
  console.log(`Listening on port ${PORT}!`);
});
```

Programski kod 10: NodeJS aplikacija sa Express razvojnim okvirom

Iako nam se na prvu možda ne čini da smo mnogo postigli prepisivanjem koda u Express apstrakcije, prednosti se očigledno vide tek kad nam se aplikacija skalira da zadovolji temeljne funkcionalnosti REST arhitekture potrebne za CRUD aplikacije. U tu svrhu ćemo koristiti Express funkcionalnosti *middleware*-ove i *router*-e kako bi ovisno o primljenim podatcima i pogođenoj izlaznoj točki aktivirali odgovarajući kontroler koji na koncu radi što smo mu zadali sa primljenim podatcima i vraća odgovor klijentu. Spominjući kontrolere već možemo razmišljati o strukturiranju naše aplikacije u MVC model kako bi si olakšali daljnje modifikacije i skaliranje koda, ali nam nedostaju još neki elementi.

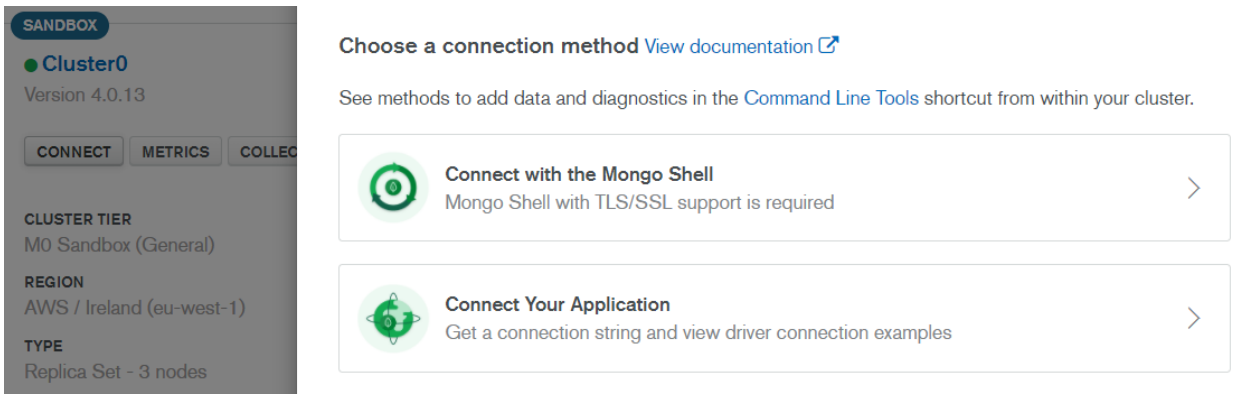
5.4. Spajanje sa bazom podataka MongoDB

Kako bismo uopće počeli koristiti Mongo bazu podataka, također je prvo moramo instalirati sa službene web stranice https://www.mongodb.com/.



Slika 7: Web stranice preko koje skidamo MongoDB Server za lokalni devleopment

Iako MongoDB unutar svog ekosustava ima mnoge dostupne development alate, za našu aplikaciju mi ćemo koristiti samo MongoDB Community Server za lokalni development i MongoDB Atlas *cloud* servis kako bi učinili našu bazu dostupnom na Webu. Kako bi napravili eksternu bazu podataka trebamo se registrirati na Atlas servis koji nas od same registracije navodi na kreiranje besplatnog *clustera*. Besplatni *cluster* koji smo kreirali na Atlasu unutar sebe ima dostupna tri *sharda* od kojih je jedan primarni, a dvije replike, a kako bismo se spojili na njega putem svoje web aplikacije moramo kliknuti na 'Connect' gumb na njemu i odabrati opciju 'Connect Your Application' gdje ćemo dobiti link na koji se možemo spojiti na *cluster*. Ova opcija se nije pokazala prigodnom za demonstraciju ovog projekta jer je na personalnom računu te unutar tog linka moramo koristiti i lozinku kako bi se povezali na *cluster*, ali za produkcijske aplikacije je ovo često standardan pristup.



Slika 8: Grafički prikaz clustera te dijaloga putem kojeg se spajamo na bazu

Za našu web aplikaciju planiram koristiti lokalno pokrenutu Mongo bazu koja bi u odnosu na spomenute *clustere* djelovala kao jedan primarni *shard*. Proces instalacije lokalne baze je već spomenut unutar poglavlja za MongoDB, te nakon instalacije jedino moramo pokrenuti .exe datoteku mongod.exe koji služi kao baza kojoj pristupamo putem 'mongodb://localhost:27017/unin', a taj URL spremamo kao *environment* varijablu DB_LOCAL putem koje ćemo zaista spojiti aplikaciju na bazu. *Driver* putem kojega ćemo programski pristupati bazi podataka je Mongoose, koji nam ujedno služi i kao ODM *library*.

```
const mongoose = require('mongoose');

require('dotenv-flow').config();

// USE TO CONNECT TO ATLAS
const DB = process.env.DB_ATLAS.replace('<PASSWORD>', process.env.
DB_PASSWORD);

mongoose.connect(process.env.DB_LOCAL, {
  useNewUrlParser: true,
  useCreateIndex: true,
  useFindAndModify: false,
  useUnifiedTopology: true
}).then(con => {
  console.log('Connected to MongoDB!');
}).catch(err => {
  console.log(err);
});

module.exports = mongoose;
```

Programski kod 11: Spajanje sa bazom podataka (datoteka db.js)

Za svrhu demonstracija navedenih tehnologija u radu odlučio sam u bazu koristiti kao korisničku bazu podataka te ćemo putem mongoose-a napraviti novu shemu sa temeljnim

podatcima za svakog korisnika. Od te iste sheme ćemo napraviti i model nad kojim zapravo pozivamo funkcije kojim spremamo i čitamo podatke iz baze. Kao što ćemo u narednom primjeru moći vidjeti, shema će također sadržavati neku temeljnu validaciju.

```
const { Schema, model } = require('mongoose');

module.exports = model('User', new Schema({
  firstName: {
    type: String,
    required: true,
  },
  lastName: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  vocation: {
    type: String,
    required: true,
  }
}));
```

Programski kod 12: Prikaz sheme koju koristimo za model u aplikaciji

Bazi ćemo pristupati putem kreiranog modela kojeg ćemo spremiti unutar users.model.js, a čije ćemo metode koristiti unutar kontrolera dostupnih u users.controller.js datoteci. Router unutar users.router.js datoteke preusmjerava svaki zahtjev odgovarajućem kontroleru koji se u biti brine da izvrši odgovarajuću metodu nad modelom, odnosno nad bazom podataka, ovisno o tipu poslanog zahtjeva, bilo to uputa za pisanje(POST), čitanje(GET), uređivanje(PUT) ili brisanje(DELETE) podataka iz baze (Create, Read, Update, Delete).

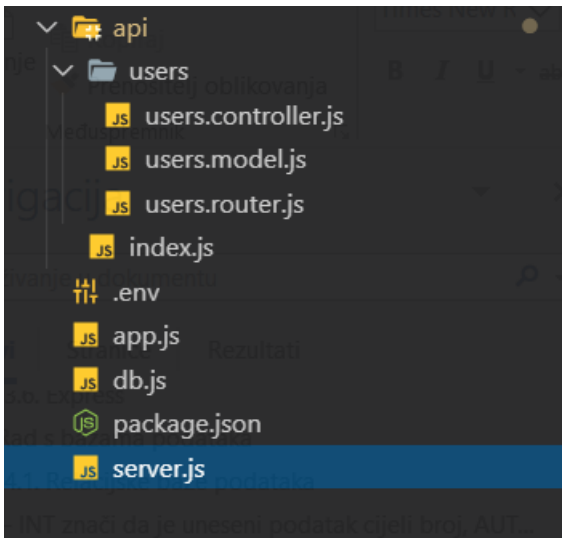
Naredni primjer prikazuje zbog čitljivosti pojednostavljen oblik kontrolera koji mi unutar aplikacije koristimo. Napominjem da za svaku od navedenih asinkronih funkcija moramo pažljivo rukovati sa potencijalnim *error*-ima pomoću `try` i `catch` blokova.

```
const User = require('./users.model');

module.exports = {
  getUsers: async (req, res) => {
    res.status(200).json(await User.find());
  },
  getUser: async (req, res) => {
    res.status(200).json(await User.findById(req.params.id));
  },
  postUser: async (req, res) => {
    await User.create(req.body);
    res.status(201).json(user);
  },
  putUser: async (req, res) => {
    res.status(204).json(await User.findByIdAndUpdate(req.params.i
d, req.body, {
      new: true,
      useValidators: true
    }));
  },
  deleteUser: async (req, res) => {
    User.findByIdAndDelete(req.params.id);
    res.status(204).json({});
  }
}
```

Programski kod 13: Kontroler segment unutar aplikacije (users.controller.js)

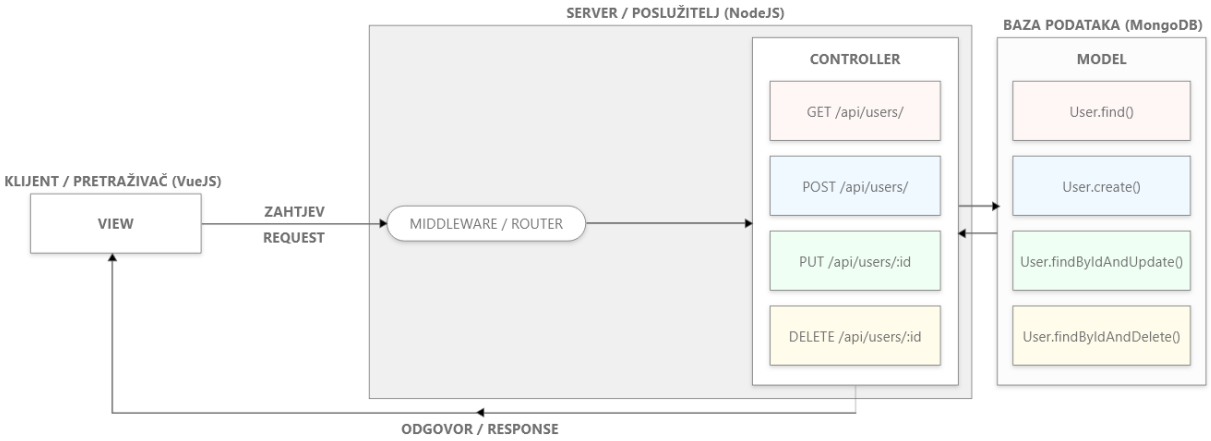
5.5. MVC (Model, View, Controller) arhitektura

Postoje silni principi strukturiranja aplikacija koje koristimo kako naknadnim developmentom i skaliranjem ne bi poremetili i zakomplicirali aplikaciju. Svaki od pojedinačnih elemenata u MVC modelu zastupa jedan segment aplikacije čijim ćemo objašnjavanjem i lakše definirati i objasniti naknadnu strukturu naše aplikacije. Postoje mnogi pristupi ovom modelu koji se može primijeniti na više tipova aplikacija, pa najčešće na isključivo klijentsku aplikaciju, gdje će nam ona unutar ovog modela služiti ipak samo kao *View* segment cijele arhitekture. *Model* u našoj aplikaciji predstavlja bazu podataka, i može biti sinonim za biznis model naše aplikacije, ali i za mongoose model kojim zapravo pristupamo bazi. *Controller* će u našoj aplikaciji biti server kojim svaki zahtjev programski preusmjerimo odgovarajućoj funkciji koja pristupa bazi podataka.



Slika 9: Pojednostavljeni prikaz kompletne strukture datoteka koje koristimo

U sljedećoj shematskom prikazu možemo vidjeti rad naše aplikacije, kojoj ćemo za kraj nadodati i klijentski dio:



Slika 10: MVC shema samog rada aplikacije

5.6. Usporedba brzine rada relacijske i ne relacijske baze

Kako bi smo dobili vjerodostojan dojam kako jedna baza radi naspram druge po brzini izvođenja, unutar obje baze ćemo unijeti iste podatke, te putem console API-a prije svakog zahtjeva bazi izračunati kojoj bazi treba koliko vremena za različite zahtjeve.

Baze koje uspoređujemo su već spomenut MongoDB kao NoSQL baza, a za SQL bazu koristimo MariaDB bazu koja je u principu ista kao što MySQL baza zato što obje potječu od iste otvorene MySQL tehnologije, samo što se MariaDB nastavila razvijati samostalno od trenutka kada je Oracle kupio MySQL bazu, zbog straha da bi je mogli početi naplaćivati.

Za obje baze unutar aplikacije koristiti ćemo objektno modeliranje podataka putem sukladnih *library*-a. Za MongoDB koristimo Mongoose, a za MariaDB koristimo Sequelize ORM. Iako nam oba *library*-a omogućuju jako slične pogodnosti, u unutarnjim mehanizmima one rade dosta drugačije s obzirom da su u pitanju NoSQL i SQL baze podataka.

```
module.exports = sequelize.define('user', {
  _id: {
    type: INTEGER,
    primaryKey: true,
    unique: true,
    autoIncrement: true,
    allowNull: false
  },
  firstName: {
    type: STRING,
    allowNull: false
  },
  lastName: {
    type: STRING,
    allowNull: false
  },
  vocation: {
    type: STRING,
    allowNull: false
  },
  email: {
    type: STRING,
    allowNull: false
  }
});
```

Programski kod 14: Primjer Sequelize modela, identičan onom koji koristimo u Mongoose-u

```
  putUser: async (req, res) => {
    try {
      console.time(`PUT /mariadb/users/${req.params.id}`);
      const user = await User.update( req.body,
      {
        where: {
          _id: req.params.id
        }
      })
      console.timeEnd(`PUT /mariadb/users/${req.params.id}`);
      res.status(204).json(user);
    } catch(err) {
      console.log(err);
      res.status(404).json({});
    }
  },
```

Programski kod 15: Primjer HTTP PUT kontrolera koji koristimo za Sequelize, i unutar kojeg mjerimo brzinu izvođenja query-a

Svo mjerenje radimo na zasebnom git branchu koji smo nazvali mariadb. Ovaj sustav mjerenja služi kao nadogradnja na postojeću aplikaciju te time što smo je razdvojili na poseban branch možemo lako 'skakati' sa jednog brancha na drugi ne mijenjajući stanje aplikacije.

Sva mjerenja brzine smo prikazali na konzoli, i jasno možemo vidjeti da je za običan tip objekta koji spremamo Mongoose znatno brži.

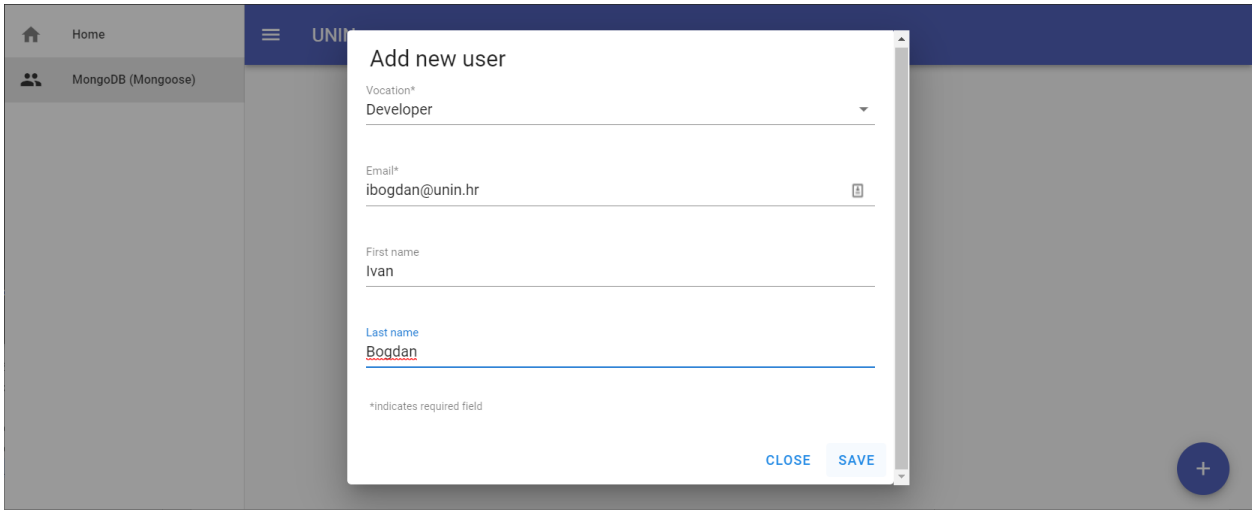
```
[nodemon] starting `node server.js`
Connected to MongoDB!
Connected to MariaDB!
Listening on port 3000...
GET /mongodb/users: 12.884ms
POST /mongodb/users: 23.859ms
GET /mongodb/users: 12.099ms
PUT /mongodb/users/5dba5cc1df0975182dc134f6: 15.423ms
GET /mongodb/users: 11.098ms
DELETE /mongodb/users/5dba5cc1df0975182dc134f6: 4.72ms
GET /mongodb/users: 2.989ms
GET /mariadb/users: 13.42ms
POST /mariadb/users: 85.083ms
GET /mariadb/users: 12.758ms
PUT /mariadb/users/10: 67.2ms
GET /mariadb/users: 13.466ms
DELETE /mariadb/users/10: 56.171ms
GET /mariadb/users: 18.032ms
```

Slika 11: Prikaz dužine trajanja svakog poslanog query-a

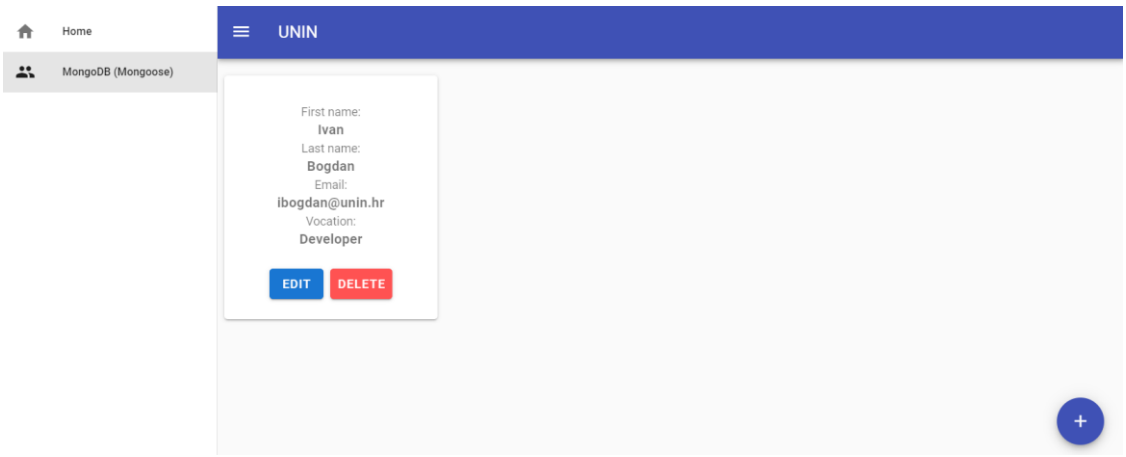
5.7. Klijentski dio aplikacije

Unutar ovog poglavlja možemo i grafički vidjeti što smo si omogućili ovom cjelokupnom serverskom i klijentskom rješenju. Za klijentsku stranu developmenta koristiti ćemo Vue JavaScript razvojni okvir jer je jako pristupačan za brz i kvalitetan razvoj. Vue koristimo tako što prvotno na naše računalo globalno putem NPM-a instaliramo Vue CLI (Command Line Interface) koristeći komandu `npm install -g @vue/cli` unutar terminala. Nakon što instalacija završi možemo koristiti CLI naredbu `vue create <app-name>` kako bi napravili početnu konfiguraciju za aplikaciju koja nam značajno smanji vrijeme developmenta automatizirajući repetativne zadatke odgovarajućim *task runnerima*, *bundlerima*, *linterima* i slično. Velika prednost Vue radnog okvira jest što povrh njega lako koristimo i Material komponente putem Vuetify razvojnog okvira te si time olakšavamo dizajn aspekt aplikacije čijoj je generalna svrha tehnička implementacija.

Pomoću Vue aplikacije smo napravili dvije dostupne stranice od kojih jedna služi kao 'Home' stranica, koju ostavljamo praznom, dok druga služi kao grafički prikaz svakog korisnika unutar naše baze podataka, gdje za svakog korisnika unutar 'kartice' imamo mogućnosti da mu uredimo vrijednosti podataka ili da ga izbrišemo. Nove korisnike dodajemo kada pritisnemo plus na donjem desnom dijelu stranice, kojim si otvaramo modal sa formularem za upis podataka.



Slika 12: Prikaz formulara za dodavanje korisnika u bazu



Slika 13: Prikaz dodanog korisnika u bazi

5.8. Kratke upute za lokalno pokretanje aplikacije u svrhu testiranja

Aplikacija je dostupna na dva GitHub repozitorija, od kojih je jedan za klijentski dio, a drugi za poslužiteljski dio, koji i ujedno putem Git *commit*-ova ima dostupnu i povijest razvoja aplikacije kroz razne korake koji su također korišteni unutar ovog rada kao esencijalni dijelovi koda u aplikaciju kroz razvoj.

- Klijentska aplikacija - https://github.com/itsadesigning/unin-vueJS
- Poslužiteljska aplikacija - https://github.com/itsadesigning/unin-nodeJS

Navedeni repozitoriji su javni i dostupni za bilo kakvo kopiranje ili korištenje u osobne svrhe, a na svoje ih računalo dohvaćamo unoseći komandu `git clone <clone-link>` u terminal koji 'razumije' Git komande, kao na primjer službeni Git bash terminal koji možemo instalirati putem službene Git stranice https://git-scm.com/downloads. Link za kloniranje koji trebamo unijeti je isti kao i URL od samog repozitorija, ali sa ekstenzijom .git na samom kraju.

Nakon što kloniramo repozitorije terminalom navigiramo u njih te pokrećemo komandu npm install za svaki od repozitorija koja će instalirati svaki eksterni modul korišten u svakoj zasebnoj aplikaciji. Popis svih eksternih modula je dostupan u package.json datoteci.

Klijentsku aplikaciju pokrećemo naredbom `npm run serve`, te joj možemo pristupiti na adresi http://localhost:8080 ako ista nije zauzeta, dok poslužiteljsku NodeJS aplikaciju pokrećemo naredbom `npm run start`, te joj možemo pristupiti na adresi http://localhost:3000.

Jedino što nam je preostalo je pokretanje baze podataka putem mongod.exe datoteke dostupne na računalu nakon instalacije MongoDB Community Servera. Baza je dostupna na adresi mongodb://localhost:27017, te joj prilikom korištenja u aplikaciji nadodajemo ekstenziju /unin kako bismo napravili i pristupili bazi podataka naziva 'unin'.

6. Zaključak

Odabir tehnologijskog stoga koji je korišten unutar nekog softverskog rješenja je često najbitniji stadij developmenta istog. Svako softversko rješenje ima prikladan pristup razvoju i prikladan stog kojem rješavamo specifične probleme s kojim se u developmentu susrećemo, te temeljni problem koji softverskim putem rješavamo.

Moderne tehnologije često nisu najprikladnije za problem koji želimo riješiti, ali često svakako pružaju neki oblik progresa u odnosu na prijašnja dostupna rješenja, pogotovu u svijetu rapidnog developmenta kakav nam JavaScript omogućuje.

Svi korišteni alati unutar ovog rada su prilagođeni JavaScript stilu developmenta kao najkorištenijem programskom jeziku danas. NodeJS je iznimno moćan alat koji ako je pravilno korišten omogućuje relativno jednostavan development za kompleksne probleme sa kojima se danas u developmentu susrećemo, pogotovo u kombinaciji sa MongoDB bazom podataka koja je svojom strukturom jako slična, pa i gotovo identična JSON formatu sa kojim je upoznat svaki JavaScript developer, pogotovo kad za pristup Mongo bazi koristimo ODM *library* poput Mongoose-a.

Najveća prednost kakvu nam je JavaScript omogućila u odnosu na prijašnja dostupna rješenja jest korištenje jednog jezika za cijeli sustav koji smo gradili, te tonu dostupnih pomoćnih alata koji nam development čine lakšim. Time si je JavaScript osigurala bistru budućnost u svijetu Web developmenta te je već dug niz godina sinonim za Web programiranje.

U Varaždinu, 15. studenoga 2019. student Ivan Bogdan

7. Literatura

[1] Kadivar A. (2019.) *How Netflix and Paypal did product transformation using Node.js*, Hackernoon, dostupno na: https://hackernoon.com/how-netflix-and-paypal-did-product-transformation-using-node-js-22074e13caad (16.09.2019.)

[2] Mozilla Developer Network - https://developer.mozilla.org/en-US/

[3] Stack Overflow - https://stackoverflow.com/

[4] Metcalfe B. (2006.) *The History of Ethernet*, NetEvents TV, YouTube [Video datoteka], dostupno na: https://www.youtube.com/watch?v=g5MezxMcRmk

[5] T. Pralas (2009.) *Računalne mreže, razvoj i značajke*, dostupno na: https://sysportal.carnet.hr/node/342

[6] PowerCert (2017.) *Hub, Switch & Router explained*, YouTube [Video datoteka], dostupno na: https://www.youtube.com/watch?v=1z0ULvg_pW8

[7] Behrouz A. Forouzan (2007.) *Data communications and networking*, McGraw Hill

[8] Korać M., Car D. (2014.) *Uvod u računalne mreže*, Algebra, Zagreb

[9] Carrie Ann (2017.) *Crash Course Computer Science #29: The Internet*, YouTube [Video datoteka], dostupno na: https://www.youtube.com/watch?v=AEaKrq3SpW8,

[10] Carrie Ann (2017.) *Crash Course Computer Science #29: The Internet*, YouTube [Video datoteka], dostupno na: https://www.youtube.com/watch?v=AEaKrq3SpW8,

[11] Carrie Ann (2017.) *Crash Course Computer Science #29: The Internet*, YouTube [Video datoteka], dostupno na: https://www.youtube.com/watch?v=AEaKrq3SpW8

[12] Internet Live Stats, dostupno na: https://www.internetlivestats.com/total-number-of-websites/ (20.09.2019)

[13] Carrie Ann (2017.) *Crash Course Computer Science #30: The World Wide Web*, YouTube [Video datoteka], dostupno na: https://www.youtube.com/watch?v=guvsH5OFizE,

[14] Korać M., Car D. (2014.) *Uvod u računalne mreže*, Algebra, Zagreb

[15] Nepoznati autor (2017.) *JavaScript V8 Engine Explained*, dostupno na: https://hackernoon.com/javascript-v8-engine-explained-3f940148d4ef

[16] *LAMP (software bundle)*, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/LAMP_(software_bundle)

[17] Jonas Schmedtmann (2019.) *Node.js, Express, MongoDB & More*, Udemy, dostupno na: https://www.udemy.com/course/nodejs-express-mongodb-bootcamp/

[18] Bergsten H. (2003.) *JavaServer Pages 3rd Edition*, Oreilly

[19] Carrie Ann (2017.) *Crash Course Computer Science #30: The World Wide Web*, YouTube [Video datoteka], dostupno na: https://www.youtube.com/watch?v=guvsH5OFizE,

[20] *HTTP Headers* (2019.) Mozilla Developer Network, dostupno na: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

[21] Eising P. (2017.) *What exactly is an API?*, Medium, dostupno na: https://medium.com/@perrysetgo/what-exactly-is-an-api-69f36968a41f

[22] *HTTP Methods* (2019.) Rest API Tutorial, dostupno na: https://restfulapi.net/http-methods/

[23] *cURL* (2019.) Wikipedia, dostupno na: https://en.wikipedia.org/wiki/CURL

[24] Madasamy M. (2019.) *JavaScript brief history and ECMAScript features (ES6, ES7, ES8, ES9)*, Medium, dostupno na: https://medium.com/@madasamy/javascript-brief-history-and-ecmascript-es6-es7-es8-features-673973394df4

[25] Can I Use? ECMAScript 5, dostupno na: https://caniuse.com/#feat=es5

[26] Madasamy M. (2019.) *JavaScript brief history and ECMAScript features (ES6, ES7, ES8, ES9)*, Medium, dostupno na: https://medium.com/@madasamy/javascript-brief-history-and-ecmascript-es6-es7-es8-features-673973394df4

[27] *Standard built-in objects*, Mozilla Developer Network, dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

[28] *A reintroduction to JavaScript*, Mozilla Developer Network, dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

[29] Alexander Zlatkov (2017.) *How JavaScript works, an overview of the engine, the runtime and the stack*, Medium, dostupno na: https://blog.sessionstack.com/how-does-javascript-actually-work-part-1-b0bacc073cf

[30] Jonas Schmedtmann (2019.) *Node.js, Express, MongoDB & More*, Udemy, dostupno na: https://www.udemy.com/course/nodejs-express-mongodb-bootcamp/

[31] *A manifestation of the manifest, npm-package-lock.json*, NPM službena dokumentacija, dostupno na: https://docs.npmjs.com/files/package-lock.json

[32] Flavio Copes (2018.) *How to use the NodeJS REPL*, dostupno na: https://flaviocopes.com/node-repl/

[33] NodeJS Module, TutorialsTeacher, dostupno na: https://www.tutorialsteacher.com/nodejs/nodejs-modules

[34] ExpressJS, NPM, dostupno na: https://www.npmjs.com/package/express (24.09.2019.)

[35] Relational database, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Relational_database

[36] *What is NoSQL?*, MongoDB službena dokumentacija, dostupno na: https://www.mongodb.com/nosql-inline

[37] MongoDB službena dokumentacija, dostupno na: https://docs.mongodb.com/manual/

[38] Mongoose službena dokumentacija, dostupno na: https://mongoosejs.com/docs/guide.html

Popis slika

Popis programskog koda

Prilozi

Uz ovaj rad prilažem CD koji će sadržavati rad u digitalnom obliku te skenirani dokument prijave zadatka i skenirani dokument potpisane izjave o autorstvu i suglasnosti za javnu obranu.

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL Odjel za multimediju

STUDIJ preddiplomski stručni studij Multimedija, oblikovanje i primjena

PRISTUPNIK Bogdan Ivan | MATIČNI BROJ 0791/336

DATUM 27.09.2019. | KOLEGIJ Programski alati 3

NASLOV RADA Pristup izradi web aplikacija pomoću NodeJS-a

NASLOV RADA NA ENGL. JEZIKU NodeJS approach to web development

MENTOR mr.sc. Vladimir Stanisavljević | ZVANJE Viši predavač

ČLANOVI POVJERENSTVA

1. doc.dr.sc. Andrija Bernik, pred. - predsjednik
2. doc.dr.sc. Dean Valdec - član
3. mr.sc. Vladimir Stanisavljević, v.pred. - mentor
4. mr.sc. Matija Mikac, v. predavač - rezervni član
5.

Zadatak završnog rada

BROJ 658/MM/2019

OPIS

Većina web aplikacija sastoji se klijentskog i poslužiteljskog dijela. U web razvoju već dugo postoji tendencija unificiranja programskog jezika na isti jezik koji se koristi za klijentski i poslužiteljski dio. Dosad je u tu svrhu prevladavala koncepcija popu LAMP-a gdje se različit NodeJS razvojna platforma približila je taj koncept stvarnosti korištenjem JavaScripta za oba segmenta razvoja.
U ovom radu potrebno je:
* opisati osnove NodeJS-a
* napraviti pregled njegovih trenutnih mogućnosti i popularnih modula te kroz kraće primjere pokazati kako se razvija aplikacija za klijentski i poslužiteljski dio
* posebnu pozornost obratiti na rad s bazama podataka i koristeći ih pohraniti jednostavne i složene objekte u neku bazu podataka
* napraviti osnovna mjerenja perfomansi učitavanja i pohranjivanja podataka
* detaljno opisati programski kod i pozive koji su korišteni u radu

ZADATAK URUČEN 24.10.2019. POTPIS MENTORA

REPUBLIKA HRVATSKA SVEUČILIŠTE SJEVER

Sveučilište Sjever

SVEUČILIŠTE SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Ivan Bogdan *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom Pristup izradi web aplikacija pomoću NodeJS-a *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)
IvanBogdan
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, Ivan Bogdan *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom Pristup izradi web aplikacija pomoću NodeJS-a *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)
IvanBogdan
(vlastoručni potpis)