Primjena Javascript kao poslužiteljskog jezika

Đakovac, Matej

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij računarstva

PRIMJENA JAVASCRIPTA KAO POSLUŽITELJSKOG JEZIKA

Završni rad

Matej Đakovac

Osijek, 2016.

SADRŽAJ

1. UVOD

Tema završnog rada je "Upotreba Javascripta kao poslužiteljskog jezika". U teorijskom dijelu dan je opis jezika koje se koriste prvenstveno na klijentskoj i poslužiteljskoj strani, detaljan opis klijentskog programskog jezika Javascript korištenog na poslužitelju – tehnologija Node.js. Dodatno, opisani su i popularni programski okviri (*framework*) za korištenje Javascripta na strani poslužitelja u sklopu Node.js.

Korištenjem Node.js tehnologije na strani poslužitelja, tehnologija HTML, CSS i Javascript na strani klijenta praktičan zadatak je izrada jednostavne chat aplikacije. Dodatno, funkcionalnost chat aplikacije ostvarena je uporabom websocketa za prijenos podataka, koji će također biti teorijski objašnjeni. Praktični dio izveden je na operacijskom sustavu Windows 7, te je potrebno znanje Windows naredbi koje se unose u komandni prozor (*cmd*). Korišteni alati uključuju obrađivač teksta - Sublime text, a instalacijom Node.js tehnologije uključen je Node Package Manager (NPM), te lokalni poslužitelj (*localhost*).

1.1 Zadatak završnog rada

U teorijskom dijelu potrebno je opisati razlike između programskih jezika koji se prvenstveno izvode na poslužiteljskoj strani i onih koji se izvode na klijentskoj strani. Opisati programske okvire (eng. Framework) koji omogućuju korištenje klijentskih jezika (prvenstveno Javascript programskog jezika) na poslužiteljima. Posebno se osvrnuti na Node.js tehnologiju. Za praktični dio potrebno je izraditi jednostavnu aplikaciju za razgovor u realnom vremenu (chat) pomoću Node.js tehnologije.

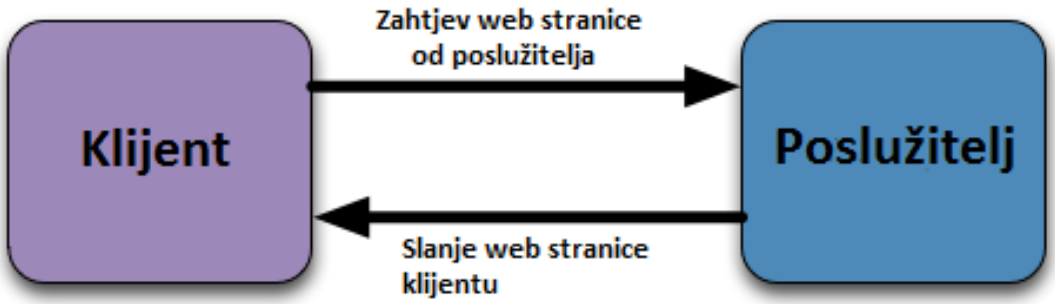
2. PROGRAMSKI JEZICI SA STRANE POSLUŽITELJA I KLIJENTA

2.1 Općenito

Razvoj mrežnih aplikacija i web stranica se u suštini svodi na međusobnu komunikaciju dvije strane koristeći HTTP protokol:

- Poslužitelj – strana odgovorna za posluživanje stranica.
- Klijent – strana koja šalje zahtjev za web stranicom i prikazuje ju krajnjem korisniku. U većini slučajeva klijent je web pretraživač.

Programiranje obje strane se odnosi na kod koji se pokreće na određenom operacijskom sustavu – poslužiteljskom ili klijentskom. [1]



Sl. 2.1. Odnos klijenta i poslužitelja [2]

2.2 Poslužitelj (*server-side*)

Skriptiranje sa strane poslužitelja je tehnika korištena u razvoju web aplikacija koja uključuje korištenje skripti na web poslužitelju koje određuju odgovor poslužitelja koji je prilagođen svakom korisničkom (klijentskom) zahtjevu. Često se koristi za pružanje prilagođenog korisničkog sučelja korisniku. Ovakve skripte mogu koristiti u prilagođavanju odgovora na temelju ovih karakteristika, zadanih uvjeta od strane korisnika, prava pristupa i slično. Skriptiranje sa strane poslužitelja dodatno služi za skrivanje izvornog koda koji generira sučelje. Općenito, kad poslužitelj pruža podatke, koristeći na primjer HTTP ili FTP protokol, korisnici imaju izbor između više klijentskih programa gdje većina modernih web pretraživača ima mogućnost zahtijevanja i primanja podataka korištenjem oba protokola. U slučaju

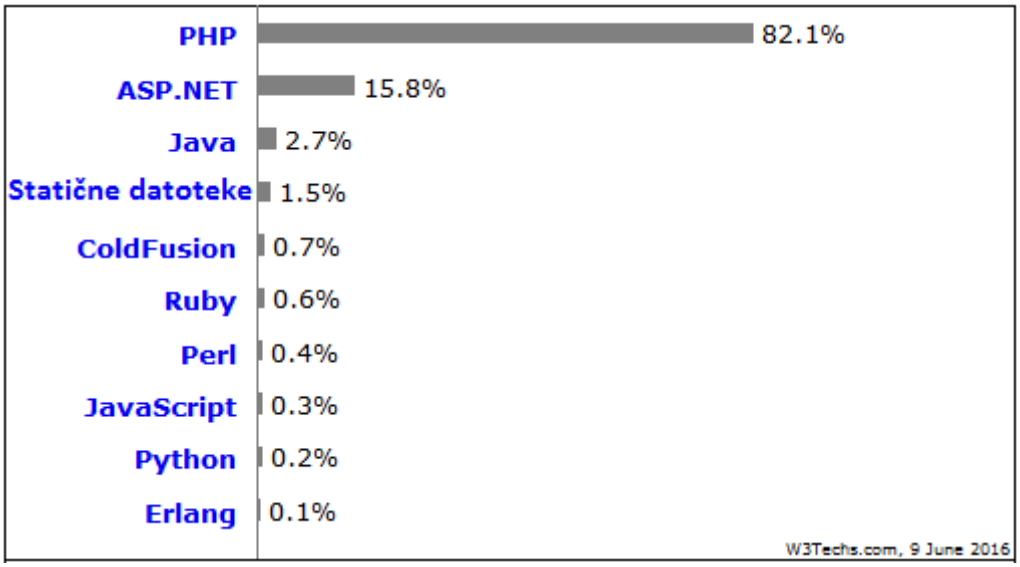
specijaliziranih aplikacija, programeri pišu vlastite poslužiteljske, klijentske i komunikacijske protokole koji se mogu koristiti samo u ovisnosti jedan o drugom. [3]

Skripte sa strane poslužitelja se i obrađuju sa strane poslužitelja. Kada klijent zahtijeva web stranicu koja sadrži poslužiteljske skripte, poslužitelj obrađuje skripte i proslijeđuje HTML stranicu klijentu.

Koriste se za:

- Obradu korisničkog unosa.
- Odgovaranje na zahtjeve klijenata.
- Strukturiranje web aplikacija.
- Interakciju sa trajnim spremnicima (baze podataka, datoteke).

Najčešće korišteni programski jezici sa strane poslužitelja su:



Sl. 2.2. Učestalost korištenja programskih jezika sa strane poslužitelja [4]

Ostali jezici, koje koristi manje od 0.1% web stranica: Miva Script, Lasso, Lua, Scala, Tcl, C++, Haskell, Smalltalk, Lisp i Ada. [4]

2.3 Klijent (*client-side*)

Web razvoj "prednjeg kraja" (*front-end*), još zvan razvoj sa strane klijenta je primjena HTML i CSS označnih jezika i JavaScripta za proizvodnju web stranice ili aplikacije. Odnosi se na onaj dio web stranice koju krajnji korisnik vidi i s kojim može upravljati.

Svojstva:

- Parsiran od strane korisničkog web pretraživača.
- Reagira na korisnički unos.
- Može se pregledavati i mijenjati od strane korisnika.
- Nije moguće spremanje podataka koji su očuvani nakon osvježenja stranice.
- Nije moguće izravno očitavanje podataka sa poslužitelja, komunicira se putem HTTP zahtjeva.

Koristi se za:

- Razvoj interaktivnih web stranica.
- Dinamičke promjene web stranica.
- Interakciju s privremenim i lokalnim spremnicima – kolačići, localStorage.
- Slanje zahtjeva poslužitelju i dobivanje podataka sa poslužitelja
- Omogućavanje udaljenih servisa za aplikacije sa strane klijenta, kao što je registracija softvera, dostava sadržaja ili multiplayer videoigre. [5]

Alati:

HyperText Markup language (HTML)

Standardni označni jezik koji se koristi za razvoj web stranica. Uz CSS i JavaScript, HTML je osnovna tehnologija za razvoj web stranica. Uz to se koristi i za razvoj korisničkih sučelja za mobilne i web aplikacije. HTML elementi su osnovni alat za razvoj. HTML omogućuje ugradnju slika i ostalih objekata i igradnju interaktivnih formi. Pruža alate za proizvodnju strukturiranih dokumenata omogućavanjem tekstualne semantike kao što su zaglavlja, odlomci, liste, poveznice, navodi i drugo. HTML elementi su opisani oznakama i zagradama < i >. Oznake kao <input /> ili <button> stavljaju sadržaj izravno na stranicu. Ostali, kao na primjer ili <p> okružuju i pružaju informacije o tekstu dokumenta i mogu uključivati druge oznake kao podređene elemente. Web pretraživači ne prikazuju HTML oznake nego ih koriste za interpretaciju sadržaja stranice. HTML može ugrađivati skripte pisane JavaScriptom korištenjem oznaka <script>. Skripte utječu na ponašanje HTML stranice.

Cascading Style Sheets (CSS)

Jezik korišten za opis prezentacije dokumenta pisan označno. Najčešće se koristi za određivanje vizualnog stila web stranica i korisničkih sučelja pisanih HTML jezikom, ali može se primijeniti

na bilo koji XML dokument, uključujući običan XML, SVG i XUL. Dizajniran je prvenstveno za odvajanje sadržaja dokumenta od prezentacije. [6]

JavaScript

Uz HTML i CSS jedna je od tri osnovne tehnologije razvoja web stranica i sučelja – koriste ih većina web stranica i podržavaju ih svi moderni web pretraživači bez ikakvih dodatnih ugradnji. Sadrži programsku sučelje za rad s tekstom, poljima, datumima i regularnim izrazima, ali ne uključuje I/O kao što je mrežno povezivanje, spremišta ili grafičke segmente.

Vrlo je kompaktan ali i vrlo fleksibilan, pa tako postoji velik broj Javascript alata koji omogućavaju mnogo dodatnih mogućnosti, kao na primjer:

- Aplikacijska programska sučelja (*APIs*) ugrađena u web pretraživače koja omogućavaju razne funkcionalnosti kao što je proizvodnja dinamičnih HTML stranica i CSS stilova, dohvaćanje i manipulacija video prijenosa sa web kamere ili rad s 3D grafičkim i audio zapisima.
- Vanjski API koji omogućava razvojnim programerima da omoguće funkcionalnost drugih stranica, kao na primjer Facebook ili Twitter.
- Vanjski programski okviri koji se mogu primijeniti na HTML s ciljem brzog razvoja stranica i aplikacija. [7]

3. NODE.JS

3.1 Općenito

Node.js je višeplatformska programska okolina otvorenog koda koja služi za razvoj web aplikacija sa strane poslužitelja. Iako Node nije JavaScript programski okvir, mnogo osnovnih modula je pisano JavaScriptom, te je moguć razvoj novih modula koristeći JavaScript. Programska okolina interpretira JavaScript koristeći Googleov V8 JavaScript pokretač. Omogućuje događajno orijentiranu, neblokirajuću infrastrukturu za izgradnju softvera.

Nodeov paketni sustav, NPM, je najveći sustav biblioteka otvorenog koda u svijetu. [8][9]

3.2 Osnovne značajke sintakse

```
1  var http = require('http');
2
3  http.createServer(function(request, response){
4      response.writeHead(200, {"Content-Type": "text/plain"});
5      response.write('Hello World');
6      response.end();
7  }).listen(8888);
```

Sl. 3.1. Hello World [10]

Prvi primjer pokazuje postavljanje jednostavnog lokalnog poslužitelja koji odgovara na zahtjev klijenta riječima *"Hello World"*.

Prvi red koda traži modul http koji dolazi s Node.js i omogućuje mu pristup preko varijable http. Nakon toga se poziva funkcija koju nudi http modul: *createServer*. Ova funkcija vraća objekt koji ima metodu zvana *listen* i prima numeričku vrijednost koja označava port na kojem će lokalni poslužitelj čekati zahtjeve.

U Javascript programskom jeziku moguće je predati funkciju kao argument. U prvom primjeru predaje se bezimena (anonimna) funkcija kao argument metodi *createServer* objekta http sa argumentima *request* i *response*. Važno je i naglasiti da ovakva funkcija ne mora biti anonimna, što znači da se može deklarirati na drugom mjestu i predati preko vlastitog imena. Argument *request* označava objekt koji predstavlja zahtjev klijenta koji se sastoji od zaglavlja i tijela, a *response* odgovor poslužitelja s istim sastavnim dijelovima. Ovakva funkcija, koja je predana

kao argument, se naziva funkcija povratnog poziva (*callback*) i upravo je to ono što omogućava događajno orijentirano programiranje korištenjem Node.js tehnologije. [10]

3.3 Događajno orijentirano (*event-driven*) programiranje

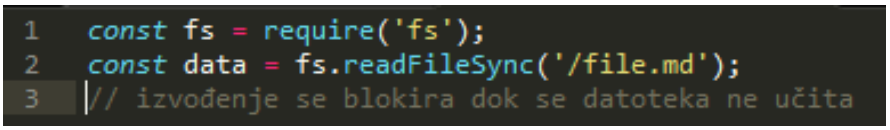
Programiranje na tradicionalan način obrađuje I/O na isti način na koji obrađuje lokalne pozive funkcija – obrada se ne može nastaviti dok se trenutna operacija ne završi. Ovakav model blokiranja pri izvođenju I/O operacija proizlazi iz ranih dana sustava podijeljenog vremena gdje je jedan proces odgovarao samo jednom korisniku. Svrha takvih sustava je bila izolirati korisnike. Korisnik je morao završiti jednu operaciju prije odlučivanja što će biti slijedeća operacija. Upravljanje većim brojem procesa je veliki teret za operacijski sustav što se tiče memorije i procesne moći te brzina izvođenja ovakvih operacija počinje opadati povećavanjem broja procesa.

Višenitnost je jedna od alternativa ovom programskom modelu. Nit je najmanja sekvenca programiranih instrukcija kojom se može upravljati neovisno. Dijeli memoriju sa svim ostalim nitima istog procesa. Niti su napravljene u svrhu proširenja prijašnjeg modela za prilagodbu izvođenju više niti istovremeno. Dok jedna nit čeka izvršenje I/O operacije, druga može preuzeti procesnu jedinicu. Kad se završi I/O operacija, odgovarajuća nit se budi, što znači da se nit koja radi trenutno može zaustaviti i nastaviti kasnije. Nadalje, neki sustavi omogućuju paralelno izvršenje niti u različitim jezgrama procesora. To znači da programeri moraju biti oprezni s istovremenim pristupom dijeljenoj memoriji. Moraju se koristiti sinkronizacijski elementi kao brave i semafori za sinkronizaciju pristupa nekim podatkovnim strukturama, prisiljavajući ih da predviđaju sve mogućnosti raspoređivanja niti za izvršenje s ciljem prevencije problema.

Događajno orijentirano programiranje je stil programiranja gdje je tok izvršenja određen događajima. Svi događaji se obrađuju preko rukovatelja događajima ili povratnim pozivima. Povratni poziv (*callback*) je funkcija koja se izvršava u slučaju nekog značajnog događaja – na primjer kad je završen upit baze podataka ili klik na neku tipku. [11]

Rad ovih funkcija pokazuje slijedeći primjer:

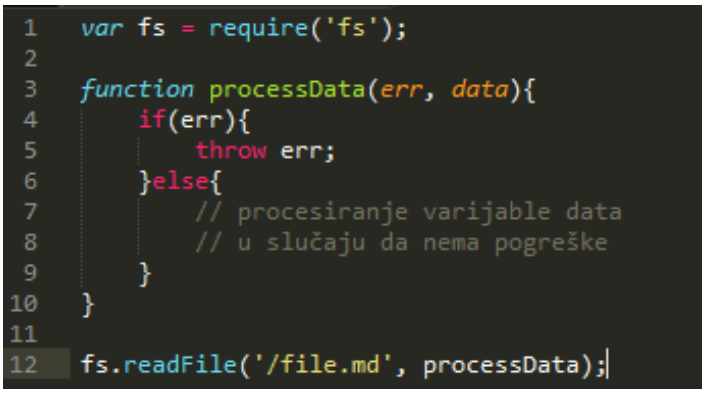
Očitavanje podataka iz neke datoteke. Dok se cijela datoteka ne učita, program ne može nastaviti s izvođenjem. Ovakve funkcije ili metode se nazivaju blokirajuće funkcije.



Sl. 3.2. Blokirajuća funkcija

Blokirajuće metode se izvode sinkrono.

Slijedi ekvivalentni asinkroni, neblokirajući primjer:



Sl. 3.3. Neblokirajuća funkcija

Definira se funkcija *processData* koja određuje način obrade učitanih podataka iz datoteke. Nakon toga se ta funkcija predaje kao argument metodi *readFile.* Kada se završi učitavanje poziva se funkcija *processData* umjesto vraćanja rezultata.

Iako prvi primjer izgleda mnogo jednostavnije od drugog, ima nedostatak u tome da drugi red blokira izvršenje cijelog programa dok se ne učita cijela datoteka. U sinkronoj verziji u slučaju pogreške ili iznimke, svaka iznimka mora biti uhvaćena ili cijeli proces prestaje s radom. U asinkronoj verziji se autoru daje izbor želi li ili ne baciti iznimku.

Ovakav stil programiranja – gdje se umjesto korištenja povratne vrijednosti definiraju funkcije koje se pozivaju pri pojavi nekog događaja se naziva asinkrono ili događajno orijentirano programiranje i jedno je od definirajućih svojstava Node.js tehnologije. Trenutni proces se ne blokira pri I/O operaciji, što znači da se više takvih operacija može izvoditi paralelno i za svaku će se pozivati odgovarajuća funkcija povratnog poziva.

Proširujući prvi primjer, dobiva se:

```
1  const fs = require('fs')
2  const data = fs.readFileSync('/file.md'); // blokiranje izvođenja
3  console.log(data);
4  // moreWork(); izvršava se nakon console.log
```

Slika 3.4. Sinkrono izvođenje naredbi

I sličan asinkroni primjer:

```
1  const fs = require('fs')
2  fs.readFile('/file.md', function(err, data){
3      if(err) { throw err; }
4      console.log(data);
5  });
6
7  // moreWork(); izvršava se prije console.log
```

Sl. 3.5. Asinkrono izvođenje naredbi

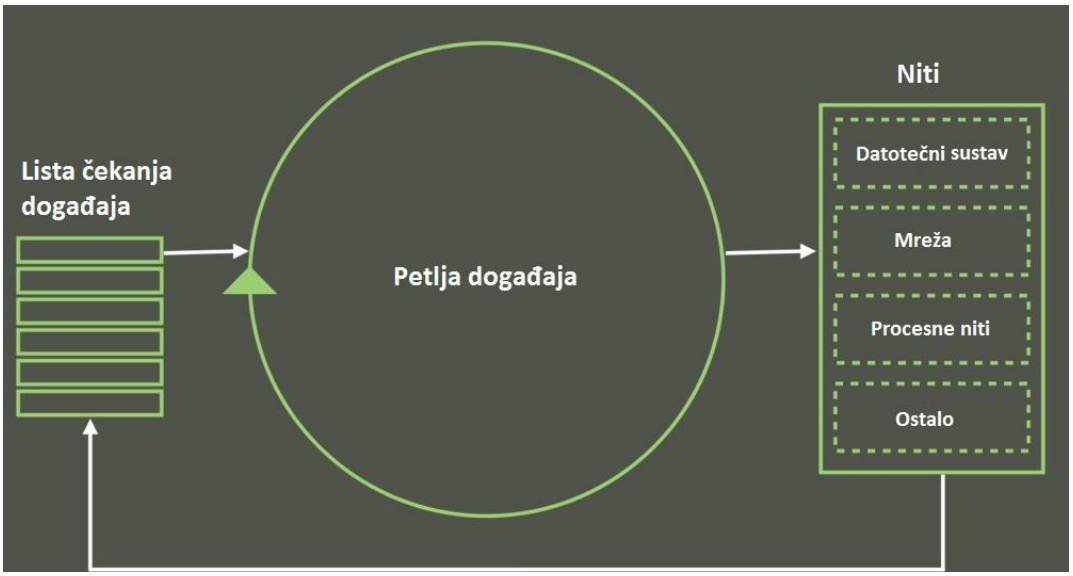
U prvom primjeru (sl.3.4.), *console.log* će biti pozvan prije funkcije *moreWork*. U drugom primjeru (sl.3.5.) *fs.readFile* je neblokirajuća funkcija pa se izvršenje naredbi može nastaviti te će funkcija *moreWork* biti pozvana prva. Mogućnost izvršenja funkcije *moreWork* bez čekanja na učitavanje datoteke je ključni dizajnerski izbor koji omogućava veću propusnost. [12]

3.4 Petlja događaja (*event loop*)

Petlja događaja je konstrukt koji primarno izvodi dvije funkcije u zatvorenoj petlji – detekcija događaja i okidanje rukovatelja događajima. U bilo kojoj iteraciji petlje detektira sve događaje te nakon toga se određuje i poziva funkcija povratnog poziva (*callback*) koja je rukovatelj događajima. Petlja događaja je samo jedna radna nit unutar jednog procesa, što znači da se kod nekog događaja rukovatelj tim događajem izvršava bez prekida. To znači slijedeće:

- U bilo kojem trenutku izvršava se samo jedan rukovatelj događajima
- Svaki rukovatelj događajima se izvršava do kraja bez prekida

To omogućava progamerima da smanje sinkronizacijske zahtjeve i da se ne moraju brinuti o usporednim nitima izvršenja koje mogu mijenjati dijeljenu memoriju.



Sl. 3.6 Petlja događaja [13]

Izvođenje Node i JavaScript aplikacija su uglavnom jednonitne petlje događaja. U svakoj iteraciji petlje procesira se slijedeći događaj na listi čekanja pozivanjem odgovarajuće funkcije povratnog poziva. Kad se događaj obradi, petlja dohvaća i procesira slijedeći događaj. Ovakvo procesiranje traje dok se ne isprazni lista čekanja. Ako se jedna od funkcija povratnog poziva izvodi duže vrijeme, petlja u međuvremenu ne procesira ostale događaje, što može dovesti do sporosti aplikacije. Korištenje funkcija koje koriste veliku količinu resursa (memorije ili procesne moći) pri rukovanju događajima može dovesti do sporosti petlje događaja, zagušenja na listi čekanja gdje događaji ostanu neobrađeni ili blokirani.

Rad petlje može se demonstrirati na slijedećem primjeru:

```
3   console.log(1);
4
5   setTimeout(function log(){
6       console.log(5);
7   }, 5000);
8
9   console.log(2);
10
11  setTimeout(function long(){
12      for(var i=0; i<10000000000; i++){ var a=i^2; }
13      console.log('Long callback responded');
14  }, 3000);
15
16  console.log(3);
17
18  request('https://www.youtube.com/watch?v=2GWPzBXNY2g',
19      function YT(error, response, body){
20          console.log('Youtube responded');
21      });
```

Sl. 3.7. Ispitivanje rada petlje događaja

Pokretanjem ove aplikacije dobiva se:

Sl. 3.8. Dobiveni rezultati ispitivanja

Objašnjenje:

- Naredba *console.log(1)* se izvršava odmah.
- Naredba *setTimeout* kojoj se kao argument daje funkcija *log*, a za drugi argument se daje 5000ms daje se na izvršenje operacijskom sustavu. Operacija čekanja 5 sekundi simulira neku dužu operaciju kao što je npr. čekanje odgovora mrežnog aplikacijskog programskog sučelja ili učitavanje velike tekstualne datoteke u memoriju. Za to vrijeme se nastavlja izvršenje Node aplikacije. Nakon 5 sekundi se stavlja funkcija *log* na listu čekanja.
- *console.log(2)* se izvršava odmah.
- Nova naredba *setTimeout* čeka 3 sekunde prije nego stavi funkciju *long* na listu čekanja. Za to vrijeme se nastavlja izvršenje Node aplikacije.
- *console.log(3)* se izvršava odmah.
- *request* modul radi zahtjev na neku web-stranicu, a predana funkcija *YT* ide na listu čekanja tek kada web-stranica odgovori. YouTube daje odgovor u roku od oko 1.5 sekundi.
- Po vremenu izvršenja na listu čekanja redom dolaze funkcije: *YT, long, log,* i tim redom se izvršavaju. Izvršavanje svake od ovih funkcija je jedna iteracija petlje događaja. Nakon što se na konzolu ispišu redom brojevi 1, 2, 3, nakon oko 1.5 sekundi se izvršava funkcija *YT* i ispisuje se tekst *"Youtube responded"*. Nakon 3 sekunde počinje izvršenje funkcije *long*. Ova funkcija sadržava for petlju čije izvršenje traje oko 20 sekundi. U 5. sekundi na listu čekanja dolazi *log* funkcija. Nakon izvršenja *long* funkcije izvršava se *log*.

Korištenjem funkcije *process.nextTick*, može se odgoditi izvršenje manje važnih operacija na slijedeću iteraciju petlje, oslobađajući petlju događaja za izvršenje ostalih događaja na čekanju. U slijedećem primjeru odgađa se dugotrajna operacija *for* petlje na slijedeću iteraciju petlje

događaja. Odgađanje funkcionira samo ako postoji funkcija na čekanju iza trenutne funkcije. [11]

```
11  setTimeout(function long(){
12      process.nextTick(function(){
13          for(var i=0; i<100000000; i++){ var a=i^2; }
14          console.log("For loop finished");
15      });
16  }, 3000);
```

Sl. 3.9. *process.nextTick*

3.5 Node Package Manager – NPM

NPM omogućava JavaScript razvojnim programerima dijeljenje koda vlastite kreacije, a koji riješava određeni problem, koji ostali mogu koristiti u vlastitim aplikacijama. Ako se koristi vanjski kod, NPM omogućava provjeru postoje li nadogradnje ili nove verzije, i preuzimanje tih nadogradnji.

Node.js i NPM imaju specifičnu definiciju paketa i modula. Dano je objašnjenje oba pojma i njihove razlike, te konvencije imenovanja određenih datoteka.

Dijelovi koda koji je moguće iskoristiti više puta se zovu paketi ili moduli. Paket je mapa s jednom ili više datoteka, te koja ima datoteku zvanu package.json s metapodacima o tom paketu. Tipična aplikacija, kao što je web stranica ovisi o desecima ili stotinama paketa, koji su najčešće vrlo mali. Općenito, namjena je stvaranje manjih blokova koji rješavaju jedan problem s ciljem izgradnje cjelokupne, prilagođene aplikacije koristeći manje, dijeljene blokove.

Modul je bilo koji dio koda koji se može učitati naredbom *require* u Node.js aplikaciji. Slijede primjeri datoteka koje mogu biti učitane kao moduli:

- Mapa s package.json datotekom koja sadrži popunjeno main polje.
- Mapa s datotekom index.js.
- JavaScript datoteka.

Neki paketi, na primjer CLI-paketi, sadrže samo sučelje komandne linije u obliku izvršne datoteke, i nemaju polje main za korištenje u Node.js programu. Ovakvi paketi nisu moduli.

Gotovo svi NPM paketi (oni koji su Node programi) sadrže veći broj modula. Razlog tome je što se svaka datoteka učitana pomoću naredbe require() smatra modulom. Prema slijedećem primjeru naredbe:

```
1   var req = require('request');
```

Sl. 3.10. korištenje naredbe *require*

Može se reći da se varijabla *req* odnosi na modul zvan *request*. [14][15]

package.json

```
{
  "name": "angular",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "angular": "^1.5.5",
    "body-parser": "^1.15.0",
    "bootstrap": "^3.3.6",
    "connect-multiparty": "^2.0.0",
    "express": "^4.13.4",
    "fs-extra": "^0.30.0",
    "mongoose": "^4.4.14",
    "ng-file-upload": "^12.0.4",
    "path": "^0.12.7"
  }
}
```

Sl. 3.10. *package.json*

Primjer *package.json* datoteke koja sadrži opće informacije o Node aplikaciji. Pod stavkom "dependencies" su nabrojani svi NPM moduli potrebni za određenu aplikaciju te korištena verzija.

4. JAVASCRIPT PROGAMSKI OKVIRI SA STRANE POSLUŽITELJA

4.1 Definicija

U kontekstu računarstva, programski okvir (*framework*) je apstrakcija u kojoj softver koji omogućava osnovnu funkcionalnost može biti selektivno izmijenjen dodavanjem novog koda, pružajući na taj način softver specifičan za određenu aplikaciju. Programski okvir je univerzalna programska okolina za višekratnu uporabu koja pruža određenu funkcionalnost kao dio veće softverske platforme s ciljem razvoja softverskih aplikacija, proizvoda i rješenja. Okviri mogu uključivati dodatne programe, kompajlere, biblioteke, alate i aplikacijska programska sučelja koja spajaju sve komponente s ciljem omogućenja razvoja projekta. [16]

4.2 MEAN.io

MEAN je programski okvir koji omogućuje jednostavno započinjanje rada koristeći MongoDB, Express, Node.js i AngularJS za razvoj aplikacije. Dizajniran je da omogući brz i organiziran način razvoja web aplikacija na bazi MEAN okvira te sadrži module kao što su Mongoose (interakcija s bazom podataka) i Passport (autentikacija) koji su već konfigurirani. Glavna svrha je optimizacija spojnih točaka između postojećih programskih okvira i rješavanje čestih integracijskih problema.

MEAN je skraćenica za Mongo, Express.js, Angular.js i Node.js. [17]

Mongo – besplatna višeplatformska dokumentno-orijentirana baza podataka otvorenog koda. Klasificira se kao NoSQL baza podataka, što znači da Mongo ne koristi tradicionalne tablične relacijske baze podataka. Umjesto toga, koriste se JSON dokumenti s dinamičnim shemama (u kontekstu Mongo baze podataka se ovakav format naziva BSON), što omogućava integraciju podataka u određenim vrstama aplikacijama na lakši i brži način. MongoDB je razvijen od strane MongoDB Inc. [18]

Express – jednostavnan programski okvir za web aplikacije koji pomaže u organizaciji aplikacije u MVC arhitekturu na strani poslužitelja. Kao jezik za pisanje predložaka (*templating*) moguće je korištenje EJS, Jade ili Dust.js. [19]

Angular.js – programski okvir sa strane klijenta na bazi JavaScripta otvorenog koda. Proširuje HTML atribute direktivama (*directives*), i veže podatke na HTML koristeći izraze (*expressions*). Angular.js direktive su HTML atributi s prefiksom ng-. Izrazi u Angular.js okviru se pišu unutar dvostrukih vitičastih zagrada – {{expression}}, a podaci izlaze točno tamo gdje ispisan izraz. Na primjer: {{7 + 5 + var}} zbraja dvije cjelobrojne vrijednosti i vrijednost varijable *var* te ispisuje ih na mjestu u dokumentu koji odgovara smještaju oznake u strukturi dokumenta. [20][21]

4.3 Meteor.js

Besplatan programski okvir na bazi JavaScripta otvorenog koda pisan koristeći Node.js. Omogućava brzo prototipiranje i proizvodi višeplatformski kod. Integriran je s MongoDB i koristi protokol distribuiranih podataka i način automatskog dijeljenja promijenjenog koda klijentima koji se bazira na objavi i pretplati. Tako je omogućeno nadograđivanje bez pisanja sinkronizacijskog koda. Na strijeni klijenta Meteor ovisi o jQuery biblioteci i može se koristiti s JavaScript UI bibliotekom. [22]

Svojstva:

- Omogućava razvoj koristeći jedan programski jezik – JavaScript u svim kontekstima – poslužitelj, klijent (web pretraživač) i mobilni uređaji.
- Koristi "data on the wire" pristup, što znači da poslužitelj šalje podatke koji nisu u HTML obliku a klijent ih procesira. [23]
- Omogućava pisanje predložaka koji se automatski nadograđuju kad se promijene modeli podataka. Najčešće se to postiže koristeći Backbone.js, Ember.js, Knockout.js, ili neki drugi alat.
- Razmjena podataka između klijenta i poslužitelja se obavlja putem websocketa koriteći socks.js ili socket.io.
- Omogućava spajanje klijenta s MongoDB na način da replicira pokretač baze sa strane poslužitelja na klijentsku stranu. Mana je što još uvijek postoje sigurnosni problemi pri ovakvom spajanju s bazom podataka.
- Kompenzacija kašnjenja se radi na način da se prvo nadogradi model sa strane klijenta i nakon toga šalje nadogradnja na stranu poslužitelja. [24]

4.4 Keystone.js

Programski okvir otvorenog koda koji služi za razvoj web aplikacija, stranica i aplikacijskih programskih sučelja pokretanih bazom podataka. Izrađen je koristeći Express i MongoDB.

Svojstva:

- Polja baze podataka – ID, stringovi, boolovi, datumi i brojevi su temeljni blokovi baze podataka. Keystone ih nadograđuje ubacivanjem korisnih polja koja se koriste u stvarnom svijetu kao što su ime, e-mail, lozinka, adresa, slike i ostala polja.
- Automatski generirano sučelje za administratore – Koristi se pri razvoju aplikacija ili u produkciji kao sustav upravljanja sadržajem baze podataka. Ubrzava i pojednostavljuje upravljanje podacima.
- Procesiranje formi – Omogućava validaciju formi, upload slika ili ažuriranje baze podataka koristeći podatkovne modele definirane od strane korisnika.
- Upravljanje sesijama – Uključene su mogućnosti upravljanja sesijama i autentikacijom, uključujući automatsku enkripciju za polja za lozinke. [25]

4.5 Ostali

Postoji mnogo drugih programskih okvira koji se koriste ovisno o namjeni, zahtjevima i lakoći rukovanja. Neki od popularnijih su: AllcountJS, Feathers, SocketStream, Meatier, Mojito, Seeds.js, SANE, COKE, Sleekjs, Danf, Catberry, Nuke.js, We.js, seneca.js i Horizon. [26]

5. PRAKTIČNI DIO – CHAT APLIKACIJA

5.1 Socket.io

Za razvoj chat aplikacije korišteni su slijedeći NPM moduli:

```
"dependencies": {
  "angular": "^1.5.6",
  "angular-sanitize": "^1.5.6",
  "angular-socket-io": "^0.7.0",
  "express": "^4.13.4",
  "mongoose": "^4.5.0",
  "socket.io": "^1.4.5"
}
```

Sl. 5.1. Korišteni NPM moduli

Angular moduli su korišteni sa strane klijenta a express, mongoose i socket.io sa strane poslužitelja.

Glavni dio aplikacije je modul socket.io, koji omogućuje dvosmjernu događajno orijentiranu komunikaciju u realnom vremenu. Za komunikaciju koristi websockete.

Websocket je protokol koji omogućava jednu stalnu TCP konekciju između poslužitelja i klijenta koja omogućava dvosmjernu komunikaciju. Dodatno, omogućuje prijenos poruka iznad TCP protokola. Sam TCP se odnosi na prijenos bajtova bez koncepta kao što je poruka. Dizajniran je za implementaciju u web pretraživačima i poslužiteljima, ali se može koristiti u bilo kojoj klijentskoj ili poslužiteljskoj aplikaciji. To je omogućeno pružanjem standardiziranog načina prijenosa sadržaja od strane poslužitelja prema klijentu bez zahtjeva upućenog od strane klijenta, što znači da omogućuje slanje poruka držeći vezu otvorenom cijelo vrijeme. Na ovaj način se ostvaruje dvosmjerna komunikacija između klijenta i poslužitelja. Komunikacija se odvija na portu 80.

Prije nego je standardiziran websocket protokol glavna metoda asinkrone komunikacije je bila *'Asychronous Javascript And XML'* (AJAX). AJAX se odnosi na slanje zahtjeva od strane klijenta prema poslužitelju i procesiranje odgovora poslužitelja bez osvježavanja web stranice - asinkrono. Za ostvarivanje ove funkcionalnosti koristi se JavaScript, koji šalje zahtjev i procesira primljeni odgovor – podatke u obliku XML ili JSON. Nakon toga se ažurira dokument kako bi se omogućio pogled na primljene podatke. Nedostatak je što nema mogućnosti komunikacije od strane poslužitelja bez da postoji zahtjev klijenta. Jedno od rješenja je bilo slanje zahtjeva

poslužitelju u određenim vremenskim intervalima koji bi odgovorio novim podacima, u slučaju da postoje. [27]

Websocket protokol je standardiziran kao RFC 6455 2011.g. od strane IETF. Websocketi su podržani u svim poznatijim web pretraživačima kao što su Google Chrome, Internet Explorer, Firefox, Safari i Opera. [28]

```
socket.io-client:url "parse http://localhost:2000" +0ms
socket.io-client "new io instance for http://localhost:2000" +2ms
socket.io-client:manager readyState closed +3ms
socket.io-client:manager "opening http://localhost:2000" +0ms
engine.io-client:socket creating transport "polling" +0ms
engine.io-client:polling polling +1ms
engine.io-client:polling-xhr xhr poll +1ms
engine.io-client:polling-xhr "xhr open GET: http://localhost:2000/socket.io/?EIO=3&transport=polling&t=LLkq1Rc" +1ms
engine.io-client:polling-xhr xhr data null +1ms
engine.io-client:socket setting transport polling +3ms
socket.io-client:manager connect attempt will timeout after 20000 +10ms
socket.io-client:manager readyState opening +2ms
engine.io-client:polling polling got data [object ArrayBuffer] +180ms
engine.io-client:socket socket receive: type "open", data "{"sid":"6PDvXu0VF7Ef7_YZAAAL","upgrades":["websocket"],"pingInterval":25000,"pingTimeout":60000}" +1ms
engine.io-client:socket socket open +1ms
socket.io-client:manager open +180ms
socket.io-client:manager cleanup +1ms
socket.io-client:socket transport is open - connecting +1ms
engine.io-client:socket starting upgrade probes +4ms
engine.io-client:socket probing transport "websocket" +0ms
engine.io-client:socket creating transport "websocket" +1ms
engine.io-client:polling polling +2ms
```

Sl. 5.2. Socket.io segmenti

Slika prikazuje isječak iz razvojne konzole web pretraživača Mozilla Firefox. Na slici se vidi inicijalizacija svih segmenata socket.io paketa sa strane klijenta koji omogućuju funkcionalnost:

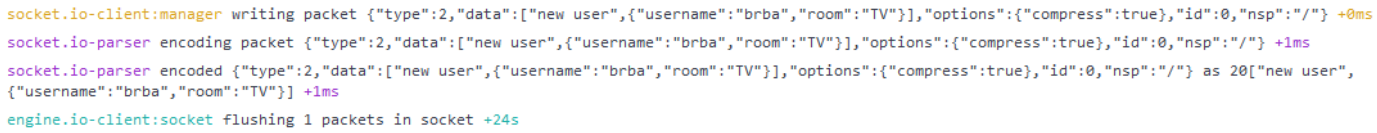
socket.io -client	url	Aktivira se pri spajanju na web stranicu
	manager	Upravlja otvaranjem i zatvaranjem veze između klijenta i poslužitelja te ispisivanjem paketa koji se šalju
	socket	Entitet koji omogućuje komunikaciju. Zadužen za emitiranje događaja, kao što su ugrađeni *connect, disconnect* i *message* te događaja definiranih od strane korisnika od poslužitelja prema klijentu.
	parser	Kodira pakete koji se šalju i dekodira primljene pakete.
engine.io -client	socket	Entitet koji omogućuje komunikaciju. Zadužen za otvaranje i konfiguraciju komunikacijskih kanala koje ispituje operacijama 'polling' i 'probing'. Također zadužen za primanje i slanje paketa. U slučaju pauze, to jest da se ne događa komunikacija, ispituje funkcionalnost poslužitelja operacijom 'ping'.

Tablica 5.1. Objašnjenje segmenata socket.io paketa

Svakom klijentu se pri prvom spajanju na web stranicu dodjeljuje socket s jedinstvenim identifikatorom, što se vidi iz isječka:

..."{"sid":"omIq5WIjZpTD8OJxAAAB","upgrades":["websocket"],"pingInterval":25000,...

Slika prikazuje slanje paketa, to jest emitiranje *message* događaja imena "new user":



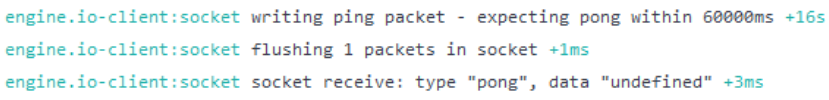
Sl. 5.3 Emitiranje događaja *new user*

Sva komunikacija poslužitelja i klijenta u ovoj aplikaciji se odvija preko *message* događaja. Socket.io omogućuje korisnicima da imenuju vlastite *message* događaje kako bi se omogućilo "slušanje" na obje strane da se utvrdi postoji li emitiranje. Nakon kodiranja, paket se šalje poslužitelju.

Sl. 5.4. Primanje paketa *old messages*

Slika prikazuje primanje paketa imena "old messages". Ovaj paket sadrži zadnjih 5 poruka (u ovom slučaju 3 jer ne postoji 5) u nekoj sobi za razgovor, neovisno o tome je li korisnik bio prisutan ili ne. Nakon dekodiranja sadržaj se ispisuje u prostor za poruke. Zadnji ispis prikazuje događaj emitiran s poslužitelja za koji je zadužen socket.io-client:socket, a prihvaća ga engine.io-client:socket u prvom ispisu.

Zadnja slika prikazuje komunikaciju poslužitelja i klijenta kad nema korisnički uvjetovane komunikacije:



Sl. 5.5. *Idle* način komunikacije

Provjerava se otvorenost kanala slanjem paketa ping i odgovorom poslužitelja.

5.2 Server.js

Server.js je datoteka koja omogućava funkcionalnost skripti sa strane poslužitelja. Sadržava sve module preuzete pomoću NPM-a te jedan vlastiti modul. Dan je pregled svih važnijih segmenata koji omogućuju funkcionalnost poslužitelja i povezivost.

Za realizaciju datoteke server.js korišten je programski okvir Express.

```
1  var express = require('express');
2  var app = express();
3  var server = require('http').createServer(app);
4  var io = require('socket.io').listen(server);
5
6  // middleware
7  // poslužuje statičke datoteke
8  app.use('/client', express.static(__dirname + '/client'));
9  app.use('/node_modules', express.static(__dirname + '/node_modules'));
10
11
12 // usmjeravanje zahtjeva klijenata
13 app.get('/', function(req, res){
14     res.sendFile(__dirname + '/index.html')
15 });
16
17 app.get('/api/rooms/get', function(req, res){
18     res.json(nicknames);
19 });
20
21
22 server.listen(2000);
23
```

Sl. 5.6. Inicijalizacija Express programskog okvira

Nakon učitavanja modula Express, aplikacija se inicijalizira i predaje metodi *createServer* gdje se nakon toga određuje port na kojem poslužitelj "sluša" - 2000.

Middleware je kod koji određuje kroz koje funkcije se predaje objekt koji sadrži zahtjev klijenta. To mogu biti funkcije za autentikaciju, obradu sadržaja zahtjeva klijenta i slično. U ovom slučaju korišten je samo middleware koji omogućuje posluživanje statičkih datoteka. Statičke datoteke su skripte i moduli korišteni sa strane klijenta. U ovoj aplikaciji služi tome da se identificira put do tih statičkih datoteka – u suprotnom skripte sa strane klijenta ne mogu pristupiti skriptama i modulima.

app.get usmjerivači određuju odgovor poslužitelja na bilo koji GET zahtijev klijenta. U slučaju da klijent traži stranicu localhost:2000/ poslužuje se datoteka *index.html*. /api/rooms/get je

zahtjev koji poslužuje sve dostupne sobe za razgovor u aplikaciji u obliku JSON dokumenta, to jest ne poslužuje datoteku već šalje podatke na obradu klijentskoj strani. Analogno tome, koriste i usmjerivači i za ostale HTTP glagole kao što su POST, DELETE, PUT i slično.

```
12  var nicknames = {
13      'Music': [],
14      'Videogames': [],
15      'Sports': [],
16      'TV': [],
17      'Politics': []
18  };
```

Sl. 5.7. Objekt *nicknames*

Objekt *nicknames* sadrži sve dostupne sobe za razgovor kao svojstva, te je svako svojstvo polje koje sadrži korisnička imena svih korisnika u toj sobi. Služi za ispis imena svih korisnika u sobi te za provjeru dostupnosti unesenog korisničkog imena.

```
41  // funkcionalnost socket.io
42  io.sockets.on('connection', function(socket){
43
44      socket.on('new user', function(data, callback){
45          util.returnAvailable(io, socket, nicknames, Message, data, callback);
46      });
47
48      // izvršava se kad je primljen događaj naziva 'send message'
49      socket.on('send message', function(data){
50          util.processMessage(io, socket, Message, data);
51      });
52
53      // izvršava se kad je primljen događaj naziva 'disconnect'
54      socket.on('disconnect', function(data){
55          util.removeUser(io, socket, nicknames);
56      });
57
58      socket.on('leave room', function(){
59          util.removeUser(io, socket, nicknames);
60      });
61
62  });
```

Sl. 5.8. Funkcionalnost socket.io modula

.on je metoda koja omogućuje funkcionalnost socket.io modula. Pruža mogućnost slušanja novih događaja – komunikacije između klijenta i poslužitelja. Kad je klijent spojen (ugrađeni događaj *connection*), dodjeljuje mu se socket s jedinstenim identifikatorom i sluša pojavu novih događaja. Događaji *new user*, *send message* i *leave room* su korisnički definirana imena događaja *message*. Svim slušateljima je predana funkcija kao rukovatelj događajima koja određuje način obrade primljenih podataka (*data*). Funkcije koje obrađuju podatke – *returnAvailable, processMessage i removeUser* dio su vanjskog korisnički definiranog modula *util.* Pri radu na većim projektima važno je razdvajati kod u module kako bi se postigla preglednost i olakšalo održavanje.

new user se aktivira pri ulasku korisnika u sobu za razgovor. Predana funkcija *returnAvailable* provjerava popis prisutnih korisnika te vraća bool vrijednost koja određuje je li moguć ulazak u željenu sobu. Ako je ulazak odobren, korisničko ime se dodaju objektu *nicknames*, a socket se pridružuje sobi. Raspoređivanje socketa na sobe je funkcionalnost modula socket.io i poziva se naredbom *socket.join(room);*. Nakon toga se iz MongoDB baze podataka dohvaća zadnjih 5 poruka u toj sobi i šalju klijentu naredbom *io.sockets.to(socket.room).emit('old messages', results);*. *Results* je objekt koji sadrži dohvaćene poruke iz baze podataka. Također se klijentu šalje popis svih korisnika trenutno spojenih na tu sobu.

Analogno tome, *leave room* se aktivira pri izlasku iz sobe. Također, *disconnect* se aktivira pri zatvaranju ili osvježavanju web stranice, što se isto tretira kao izlazak iz sobe. Predana funkcija može primati podatke, ali nije obavezno. Funkcija *removeUser* uklanja korisničko ime odspojenog korisnika iz objekta *nicknames* te šalje ažurirani popis svih spojenih korisnika.

Događaj *send message* je događaj koji se aktivira pri svakom slanju poruke koju vide svi ostali korisnici u toj sobi. Funkcija povratnog poziva prima objekt *data*, čija su svojstva *user*, koji sadrži korisničko ime, *message*, koji sadrži poruku te *room* koji sadrži sobu iz koje je poruka poslana. Dobiveni objekt se emitira svim korisnicima u sobi, uključujući onog koji ju je poslao. Nakon toga se ista poruka sprema u bazu podataka sa sva tri svojstva, i jednim dodanim – svojstvo *created* koji sadrži vrijeme zaprimanja poruke. Definiran je model, to jest objekt koji se sprema u bazu podataka, a uključen je kao vanjski modul naredbom *module.exports*:

```
var mongoose = require("mongoose");

module.exports = mongoose.model('Message', {
    created: {type: Date, default: Date.now()},
    user: String,
    message: String,
    room: String
});
```

Sl. 5.10. Mongoose model

5.3 Index.html

Index.html je dokument koji određuje strukturu dokumenta koji se poslužuje korisniku. Uz označni jezik HTML korišten je JavaScript s programskim okvirom Angular.js te CSS. Angular.js se koristi za jednostavnu autentikaciju formi i prikaz pogreški te za povezivanje elemenata HTML-a s JavaScript varijablama. Većina funkcionalnosti sa strane klijenta je u

drugoj datoteci – *app.js*. Slijedeći primjer prikazuje korištenje Angular.js direktiva u HTML dokumentu:

```
32   <body ng-app="chat">
33       <div ng-controller='main'>
```

Sl. 5.11. Osnovne Angular.js direktive

Direktiva ng-app označuje ime aplikacije. Aplikacija se po imenu koje ju identificira predaje u modul. Moduli u Angular.js deklarativno određuju funkcionalnost aplikacije. Nekoliko je prednosti ovakvog pristupa:

- Lakše razumijevanje deklarativnog procesa.
- Pakiranje koda u module koji su iskoristivi više puta.
- Moduli se mogu učitavati bilo kojim redom ili paralelno jer imaju odgodu izvršenja.
- Ubrzavanje testiranja – pri testiranju se učitavaju samo relevantni moduli. [29]

Slijedeći primjer prikazuje korištenje Angular direktiva u validaciji formi:

```
36          <div ng-show="!inRoom">
37              <h3>Choose a chatroom and your username and click the Submit button</h3>
38                  <form name="loginForm" id="loginForm" ng-submit="enter()" novalidate>
39                      <select name="room" ng-model="chatRoom">
40                          <option ng-repeat="room in availableRooms" required>{{room}}</option>
41                      </select>
42
43                      <input  name="username"
44                              ng-keypress="error=''"
45                              type="text"
46                              ng-model="username"
47                              placeholder="Username"
48                              ng-maxlength=12
49                              required>
50
51                      <span ng-show="loginForm.username.$dirty && loginForm.username.$error.required">
52                          Username is required!
53                      </span>
54                      <span ng-show="loginForm.username.$error.maxlength">Too long! 12 chars max!</span>
55                      <span ng-show="error">{{error}}</span>
56
57                      </br></br>
58
59                      <button type="submit" ng-disabled="loginForm.$invalid || inRoom">Start chatting!</button>
60                  </form>
61          </div>
```

Sl. 5.12. Angular.js validacija

ng-show direktiva određuje hoće li neki element dokumenta biti prikazan – ovisi o rezultatu logičkog izraza. *!inRoom* označava logički izraz koji je istinit ako je postavljena varijabla *inRoom.*

ng-model označava direktivu koja omogućuje dvosmjerno vezivanje varijabli sa HTML elementima. Dvosmjerno vezivanje varijabli je koncept koji povezuje ispisane varijable u

HTML-u i varijable u JavaScriptu, što znači da se promjena varijable u JavaScriptu odražava u HTML i promjena u HTML-u, na primjer unosom preko <input> oznake se odražava u upravljaču.

Validacija formi se izvršava preko Angular.js – to jest klasa dodanih u CSS kao što su $error, $valid, $invalid, $pristine (nije bilo interakcije s elementom), $dirty (suprotno od $pristine), $touched (događaj blur se pojavio), $untouched. U gornjem primjeru se pojavljuje pogreška ako je bilo interakcije s elementom a nije upisano korisničko ime, ili je ime predugačko. Dakle ako je prekršeno ijedno od tih pravila, klasa je $invalid i tada je onemogućena predaja forme.

```
63          <!-- CHATROOM -->
64          <div ng-show="inRoom">
65              <h3 id="title">{{chatRoom}} chatroom</h3>
66              <div id="{{chatRoom}}" class="contentWrap">
67                  <div id="chat" ng-bind-html=chatWindow>{{chatWindow}}</div>
68                  <div id="users" ng-bind-html=usersOnline></div>
69                  <form name ="sendMessageForm" id="sendMessageForm" ng-submit="sendMessage()" novalidate>
70                      <input size=45 type="text" ng-model="message">
71                      <button type="submit">Submit</button>
72                      <button id="leavebtn" ng-click="leave()">Leave room</button>
73                  </form>
74              </div>
75          </div>
```

Sl. 5.13 Dio dokumenta za poruke

Isječak prikazuje dio dokumenta koji se odnosi na sam chat. Vidi se ispis same varijable u dvostrukim zagradama – {{chatRoom}}. Od direktiva koje nisu spomenute postoji *ng-bind-html* – analizira izraz i dodaje ga HTML dokumentu na siguran način. Koristi se kad se unutar vrijednosti varijable nalaze HTML oznake kao
 ili i slično. *ng-click* je direktiva koja izvršava neku naredbu kada se klikne na odgovarajući HTML element.

5.4. App.js

Datoteka koja sadrži JavaScript naredbe koje omogućuju funkcionalnost aplikacije sa strane klijenta. Angular.js module se inicijalizira na slijedeći način:

```
5   (function(){
6       var app = angular.module('chat', ['btford.socket-io', 'ngSanitize']);
7
8       app.factory('socket', function (socketFactory) {
9           return socketFactory();
10      });
11
12      app.controller('main', ['$scope', '$http', 'socket', function($scope, $http, socket){
```

Sl. 5.14. Inicijalizacija Angular modula

Funkcija *module* prima dva argumenta: prvi je ime, a drugi je polje modula o kojima ovisi rad aplikacije. btford.socket-io omogućava rad socket.io paketa s Angular.js programskim okvirom. ngSanitize je paket, i ujedno i direktiva koja omogućava sigurno dodavanje *ng-bind-html* direktive u HTML dokument.

Factory je jedan od servisa koji su uključeni u Angular.js. Koristi se za definiciju objekata ili funkcija izvan glavnog modula. Korištenje ovakvih servisa doprinosi modularizaciji i čišćem kodu što olakšava razvoj i održavanje. *socketFactory* vraća objekt koji je dio btford.socket-io paketa.

Sastavni dio modula je upravljač (*controller*). U upravljaču se nalaze sve varijable i funkcije korištene u izvedbi skriptiranja sa strane klijenta. Deklarira se direktivom *ng-controller* po imenu. Upravljač je konstrukt koji sadrži funkcije i varijable. Prima dva argumenta: prvi je ime, a drugi je polje zavisnih varijabli. $scope je objekt koji označava opseg varijabli – ovim varijablama mogu pristupiti samo datoteke koje izričito navode uključenost ovog modela. $http označava objekt koji služi za zahtjeve poslužitelju. Preko $http objekta je moguće napraviti GET, POST, DELETE, PUT i slične zahtjeve. Već spomenuti socket je zavisnost koja se injektira kako bi se mogao koristiti socketFactory objekt.

```
29          $scope.enter = function(){
30              socket.emit('new user', {username: $scope.username, room: $scope.chatRoom}, checkAvailable);
31          }
32
33          $scope.leave = function(){
34              socket.emit('leave room');
35              $scope.chatWindow = '';
36              $scope.inRoom = false;
37              $scope.wroteOld = false;
38          }
39
40          $scope.sendMessage = function(){
41              if($scope.message === ''){ return; }
42
43              var date = new Date;
44
45              socket.emit('send message', $scope.message);
46              $scope.message = '';
47          }
```

Sl 5.15. Funkcije unutar upravljača vezane za HTML elemente

Ove funkcije unutar upravljača omogućuju reaktivnost HTML dokumenta, i vezane su za *ng-click* direktive. *enter* se poziva kada korisnik odabere sobu u koju želi ući – ovisno o tome je li korisničko ime dostupno mijenja se korisničko sučelje bez osvježavanja web stranice. Događaj *new user* se emitira i server koji sluša obrađuje takav događaj.

leave je funkcija koja se poziva klikom na tipku Leave Room. Emitira se odgovarajući događaj i ažuriraju se odgovarajuće varijable.

sendMessage je funkcija koja se aktivira klikom na tipku Send Message. Ako polje za poruku nije prazno emitira se odgovarajući događaj.

```
49          socket.on('new message', function(data){
50              var time = new Date;
51              var hms = getTime(time);
52              $scope.chatWindow += '['+ hms + ']  <b>' + data.nick + ': </b>' + data.msg + "<br/>";
53          });
54
55          socket.on('old messages', function(data){
56              if(!$scope.wroteOld){
57                  console.log('old messages fires')
58                  for(var i=data.length-1; i>=0; i--){
59
60                      var time = new Date(data[i].created);
61                      var hms = getTime(time);
62                      $scope.chatWindow += '['+ hms + ']  <b>' + data[i].user + ': </b>' + data[i].message + "<br/>";
63                  }
64              }
65              $scope.wroteOld = true;
66
67          });
68
69          socket.on('usernames', function(data){
70              console.log(data);
71              var html = '';
72              for(i=0; i<data.length; i++){
73                  html += '   ' + data[i] + '<br/>';
74              }
75              $scope.usersOnline = html;
76          });
77
```

Sl 5.16. Slušanje sa strane klijenta

Isječak prikazuje slušanje dolaska novih podataka sa poslužitelja koji se obrađuju na odgovarajući način. Pri *new message* događaju, dodaje se na varijablu *chatWindow* nova poruka uz odgovarajuće vrijeme prilagođeno na ovu vremensku zonu. Događaj *old messages* se pojavljuje pri ulasku u sobu i na varijablu *chatWindow* dodaje zadnjih 5 poruka za koje korisnik nije bio prisutan. Događaj *usernames* obrađuje primljeni popis korisnika koji se ažurira svaki put kad neki od korisnika uđe ili napusti sobu.

6. ZAKLJUČAK

Node.js tehnologija je vrlo prilagođena aplikacijama gdje je potrebno održavanje stalne veze između klijenta i poslužitelja. Korištenjem websocketa, moguće je napisati aplikaciju koja ažurira podatke sa strane klijenta u realnom vremenu. Ovakav oblik stalne veze bi na platformama kao što su Django ili Ruby on Rails preopteretio poslužitelja (iako postoje alati koji omogućuju ovakvu strukturu) iz razloga što se za svakog korisnika koristi jedan proces, dok kod Node.js nema potrebe za održavanjem različitih niti za svaku otvorenu vezu. Nodeov jednonitni sustav vrlo brzo obrađuje veliki broj zahtjeva u isto vrijeme. Dodatna prednost je korištenje istog programskog jezika – Javascript sa strane poslužitelja i klijenta što olakšava razvoj, prototipiranje i testiranje. Funkcije povratnog poziva su izvrstan način ostvarivanja događajno orijentiranog programskog koda, jedina negativna strana koju sam primijetio je povremena nečitkost ovakvog stila pisanja koda, što potencijalno otežava održavanje i debagiranje na dugi rok. Iz tog razloga vrlo je važno modularizirati vlastiti kod na logičan i čitljiv način.

Programski okviri kao što su Express, MEAN i Meteor osiguravaju vrlo jednostavan i brz način da se započne razvoj projekata. Korištenjem generatora moguće je jednom naredbom kreirati strukturu mapa i dokumenata koja će se koristiti u dokumentu. Middleware funkcije su univerzalan i cjelovit način obrade klijentskih zahtjeva kroz definirani stog operacija, što znači da je zahtjeve moguće prilagođavati koristeći funkcije s ciljem ispunjavanja npr. sigurnosnih odredbi.

Situacije u kojima Node.js nije najbolji izbor za skriptiranje sa strane poslužitelja su one u kojima se koriste komplicirane operacije ili algoritmi koji zauzimaju veliki dio procesne moći ili memorije. Iako je vrlo dobar za blokirajuće operacije kao što je I/O, one intenzivnije za procesor će blokirati događajnu petlju i onemogućiti rad aplikacije. Stoga je potrebno analizirati zahtjeve i ograničenja pri biranju tehnologije na početku projekta.

LITERATURA

[1] What are the differences between server-side and client-side programming?, http://programmers.stackexchange.com/questions/171203/what-are-the-differences-between-server-side-and-client-side-programming 30.4.2016.

[2] Introduction to server-side scripting, Adam McNicol, http://www.pythonschool.net/server-side-scripting/introduction-to-server-side-scripting/ 30.4.2016.

[3] Architecture of JavaScript Applications, http://docs.oracle.com/cd/E19957-01/816-6411-10/getstart.html 1.5.2016.

[4] Usage of server-side programming languages for websites, https://w3techs.com/technologies/overview/programming_language/all 30.4.2016.

[5] Web Application Architecture from 10,000 Feet, Part 1 – Client-Side vs. Server-Side, Mel Klimushyn, https://spin.atomicobject.com/2015/04/06/web-app-client-side-server-side/ 2.5.2016.

[6] Programming languages on the internet, http://www.webdevelopersnotes.com/basics/languages_on_the_internet.php3 2.5.2016.

[7] developer.mozilla.org, Javascript basics, https://developer.mozilla.org/enUS/Learn/Getting_started_with_the_web/JavaScript_basics 5.5.2016.

[8] About Node.js, https://nodejs.org/en/about/ 5.5.2016.

[9] Init.js: A Guide to the Why and How of Full-Stack JavaScript, Alejandro Hernandez, https://www.toptal.com/javascript/guide-to-full-stack-javascript-initjs 3.5.2016.

[10] M. Kiessling, The Node Beginner Book, Leanpub, 2014.

[11] P. Teixeira, Professional Node.js, John Wiley & Sons, Inc., 2013.

[12] Github, Overview of Blocking vs Non-Blocking, https://github.com/nodejs/node/blob/master/doc/topics/blocking-vs-non-blocking.md 6.5.2016.

[13] Event loop, http://softwareengineeringdaily.com/wp-content/uploads/2015/07/event-loop.jpg 6.5.2016.

[14] What is NPM?, https://docs.npmjs.com/getting-started/what-is-npm 10.5.2016.

[15] 6 things you should know about Node.js, Ben Wen, http://www.javaworld.com/article/2079190/scripting-jvm-languages/6-things-you-should-know-about-node-js.html 15.5.2016.

[16] Framework Design: A Role Modeling Approach, Dirk Riehle, http://dirkriehle.com/computer-science/research/dissertation/diss-a4.pdf 15.5.2016.

[17] Github, linnovate/mean, https://github.com/linnovate/mean 16.5.2016.

[18] Release Notes for MongoDB 3.2, https://docs.mongodb.com/manual/release-notes/3.2/ 16.5.2016.

[19] What is Express.js?, http://stackoverflow.com/questions/12616153/what-is-express-js 17.5.2016.

[20] Learn Angular, Lessons, http://www.learn-angular.org/#!/lessons/the-essentials 17.5.2016.

[21] w3schools, AngularJS Introduction http://www.w3schools.com/angular/angular_intro.asp 25.5.2016.

[22] Meteor Explained - A Journey Into Meteor's Reactivity, Arunoda Susiripala, https://gumroad.com/l/meteor-explained 18.5.2016.

[23] Introducing Meteor API Docs, http://docs.meteor.com/#/full/ 19.5.2016.

[24] How does the Meteor JavaScript framework work?, http://stackoverflow.com/questions/10214385/how-does-the-meteor-javascript-framework-work 29.5.2016.

[25] Keystone.js, http://keystonejs.com/ 5.6.2016.

[26] Node Framewoks, http://nodeframework.com/ 10.6.2016.

[27] How does AJAX work?, http://stackoverflow.com/questions/1510011/how-does-ajax-work 12.6.2016.

[28] Websocket URIs, Ian Fette, Alexey Melnikov, https://tools.ietf.org/html/rfc6455#section-3 13.6.2016.

[29] Angular.js, Module Guide, https://docs.angularjs.org/guide/module 15.6.2016.

SAŽETAK

Web stranice i aplikacije razvijene korišenjem Node.js tehnologije vrlo dobro riješavaju velik broj istovremenih zahtjeva te usporedne blokirajuće operacije kao što je I/O – čitanje i pisanje u datoteke ili baze podataka, te mrežne operacije, što se postiže funkcijama povratnog poziva i petljom događaja. Korištenjem websocket protokola (za Node.js je dostupan u obliku NPM paketa socket.io) je moguća izgradnja aplikacija gdje klijent i poslužitelj komuniciraju u realnom vremenu. NPM je sustav za dijeljenje programskih rješenja te omogućava jednostavnu instalaciju vanjskih paketa. Korištenje JavaScript programskih okvira sa strane poslužitelja kao što su Express ili MEAN omogućava brzo započinjanje projekta te nudi dodatne mogućnosti.

Ključne riječi: JavaScript, Node.js, poslužitelj, klijent, NPM, programski okvir, socket.io, websocket

ABSTRACT

Usage of Javascript as a server-side language

Web pages and applications developed using Node.js easily deal with large numbers of concurrent client requests and also blocking operations such as I/O – reading and writing to files or databases, or network operations. This is done using callback functions and the event loop. Using the websocket protocol, which is accessible to Node as an NPM package called socket.io, applications which use real-time communication between the client and the server can be built. NPM is a code sharing system which enables easy installation of external solutions. Using server-side JavaScript frameworks, such as Express or MEAN, projects can be scaffolded and most frameworks offer additional functionality.

Key words: JavaScript, Node.js, server, client, NPM, framework, socket.io, websocket

ŽIVOTOPIS

Đakovac Matej rođen je 19. lipnja 1994.g. u Našicama. Živi u Osijeku te se u istom mjestu obrazuje. Pohađao je Osnovnu školu Ante Starčevića u Viljevu od 2001. do 2009. Zatim nastavlja svoje obrazovanje u Srednjoj školi Donji Miholjac smjera Opća gimnazija od 2009. do 2013. Završetkom srednje škole, uspješno upisuje preddiplomski sveučilišni studij računarstva na Elektrotehničkom fakultetu u Osijeku 2013. godine, te sve ispite redovno polaže i pretežito sa vrlo dobrim i dobrim uspjehom.

Potpis:

_____

PRILOZI

GitHub repozitorij: https://github.com/ace196/chat-app