



UNIVERZITET SINGIDUNUM

**-MASTER STUDIJSKI PROGRAM-
SAVREMENE INFORMACIONE TEHNOLOGIJE**

**KOMUNIKACIONI PROBLEMI KLIJENT-
SERVER PROTOKOLA KOD MODERNIH WEB
APLIKACIJA**



KOMUNIKACIONI PROBLEMI KLIJENT- SERVER PROTOKOLA KOD MODERNIH WEB APLIKACIJA

Mentor:

Prof Dr Mladen Veinović

Student:

Simo Simeunović

Br. indeksa: 410506/2011



Beograd, 2012.

Komunikacioni problemi klijent server protokola kod modernih web aplikacija

Sažetak: U ovom master radu razmatraju se komunikacioni problemi na relaciji klijent-server kod modernih web aplikacija. U radu su teorijski analizirani komunikacioni problemi koji se javljaju u tom procesu, a zatim su kroz praktičnu implementaciju predloženi načini njihovog rješavanja. U radu su izložene tehnologije i protokoli koji omogućavaju ovaj vid komunikacije i njihove karakteristike. Rad naglašava značaj rešenja ove problematike kad se govori o razvoji modernih web aplikacija.

Ključne reči: WebSocekt protokol, HTTP protokol, HTML5, web aplikacije, klijent-server komunikacija, Java, Falash, JavaScript, Node.js, Socket.io, Twitter, Google Maps

Communication problems of client-server protocol for modern web applications

Abstract: This master thesis considers the communication problems between client and server for modern web applications. This paper presents a theoretical analysis of communication problems that arise in the process. In addition to that, there are presented ways of solving them, through the practical implementation. The paper presents the technologies and protocols that enable this type of communication and their characteristics. The paper presents the technologies and protocols that allow this form of communication features of the considered technologies and protocols. This work emphasizes the importance of solving this problem when it comes to the development of modern web applications.

Keywords: WebSocekt protocol, HTTP protocol, HTML5, web applications, client-server communication, Java, Falash, JavaScript, Node.js, Socket.io, Twitter, Google Maps

Sadržaj:

1 Metodologija istraživačkog rada	8
1.1 Uvodne napomene i obrazloženje rada	8
1.2 Predmet istraživanja	8
1.3 Hipotetički okvir	8
1.4 Ciljevi istraživanja.....	9
1.5 Metode istraživanja i tok istraživačkog procesa.....	9
2 Pojava Interneta kao globalne mreže	10
2.1 Prvi sajtovi i pojava dinamičkih aplikacija	10
3 Web aplikacije danas:	11
3.1 Prednosti u odnosu na tradicionalne desktop aplikacije	11
3.2 Tehnologije koje se koriste pri izradi	11
4 HTTP protokol	15
4.1 HTTP sesija	16
4.2 Metodi za zahtjev	16
4.3 Sigurni metodi.....	17
4.4 Statusni kod	18
4.5 Otvorena konekcija	18
4.6 HTTP stanje sesije	18
5 AJAX.....	20
5.1 Uvod	20
5.2 Osnove AJAX tehnologije.....	20
5.3 AJAX tehnologija i sigurnost	22
5.3.1 Uvećana mogućnost napada	22
5.3.2 Problemi s programskim kodom koji se izvršava na klijentskoj strani ..	22
5.3.3 AJAX tehnologija i ispitivanje sigurnosti	23
5.4 Problem gubitka pojma "stanja"	24
5.5 Vremenski okidači i pokretanje događaja	25
5.6 Dinamički DOM	25

5.7	AJAX long polling (Commet).....	26
5.8	Rešenja treće strane.....	27
5.8.1	Prednosti	27
5.8.2	Mane	27
5.8.3	Razlika u odnosu na WebSocket protokol.	27
6	HTML5 standard.....	29
6.1	Uvod	29
6.2	Prednosti u odnosu na prethodnika.....	29
6.3	Noviteti koje donosi.....	30
7	WebSocket	32
7.1	Opis	32
7.2	Primjer zahtijva.....	32
7.3	Primjer dobijenog odgovora	33
7.4	Sigurnost.....	33
7.5	Prednosti.....	34
7.6	Nedostatci	36
8	Aplikacije:.....	37
8.1	TWstream	37
8.1.1	Uvod.....	37
8.1.2	Početna strana	38
8.1.3	Info panel.....	39
8.1.4	Proces konektovanja	40
8.1.5	Promjena ključne riječi	42
8.1.6	Pozicioniranje na karti	44
8.1.7	Izvorni kod aplikacije	45
8.1.8	Serverski dio aplikacije.....	51
8.2	Chat aplikacija.	58
8.2.1	Opis problematike	58
8.2.2	Opis rešenja.....	59
8.2.3	Upotreba.....	60

8.2.4 Izvršni kod aplikacije.....	63
8.2.5 Serverski dio aplikacije.....	65
8.3 PinPoint.....	69
8.3.1 Opis aplikacije i princip rada	69
8.3.2 Izvorni kod.....	74
8.3.3 Serverska strana.....	78
9 Zaključak	80
10 Literatura	81
10.1.1 Pisana izdanja	81
10.1.2 Internet izvori	81

1 Metodologija istraživačkog rada

1.1 Uvodne napomene i obrazloženje rada

Ovaj master rad razmatra komunikacione probleme kod standardnih klijent-server arhitektura u Internet aplikacijama. U radu su precizno definisani problemi koji se odnose na razvoj modernih web aplikacija, a zatim su predložena moguća rešenja. Za potpuno razumijevanje izložene materije potrebna su odgovarajuća predznanja kako iz oblasti mrežnih tehnologija tako i iz oblasti web programiranja.

1.2 Predmet istraživanja

Predmet istraživanja ovog rada predstavlja problem klasične komunikacije između klijenta, tj. korisnika web aplikacije i servera na kojem se aplikacija nalazi. Razvoj modernih računarskih mreža, te pojava novih i konstantno unapređivanje starih tehnologija koje svoju primjenu nalaze na web-u, dovode do toga da navedeni problem komunikacije dobija sve veću težinu. Novi računarski protokoli i standardi nastoje da eliminišu probleme koji nastaju u klasičnoj klijent-server komunikaciji.

1.3 Hipotetički okvir

Opšta hipoteza ovog rada predstavlja činjenica da se problem komunikacije između klijenta i servera kod modernih web aplikacija može najlakše riješiti na adekvatan način uvođenjem novog protokola koji će eliminisati probleme koje sa sobom nosi komunikacija preko HTTP protokola.

Opšta ili radna hipoteza ogleda se u konstataciji da je do svega prije par godina bilo gotovo nemoguće osmisliti, dizajnirati i implementirati modernu web aplikaciju koja potražuje visok stepen dinamičnosti i interakcije sa korisnikom bez korišćenja određenog rešenja treće strane koje kao takvo unosi dodatne probleme.

Pojedinačna hipoteza. Internet je postao neizostavan u svakodnevnom životu. Sa njegovim razvojem napreduje i razvoj svih grana koje se oslanjaju na njega. Trend razvoja web aplikacija pokazuje da će one ubrzo preuzeti primat od desktop aplikacija. Imajući ovo u vidu, od izuzetne važnosti je da fundamentalni segmenti istih budu implementirani na najbolji mogući način.

1.4 Ciljevi istraživanja

Cilj istraživanja jeste da analizira i u potpunosti razjasni proces komunikacije koja se odvija na relaciji klijent - server kod modernih web aplikacija. Konkretni ciljevi sprovedenog istraživanja ogledaju se u sledećim tačkama:

- kako funkcioniše komunikacija između klijenata i servera danas,
- koji su problemi takvog pristupa te kako se oni prevazilaze,
- šta je mana sadašnjih rešenja te šta predstavlja optimalno rešenje.

1.5 Metode istraživanja i tok istraživačkog procesa

Metodologija istraživanja zasnivala se prvenstveno na proučavanju literature, koja uglavnom potiče sa Interneta ali takođe i iz knjiga uglednih autora. Analizirani su problemi iz literature, a zatim su razvijane praktične aplikacije kojima su demonstrirani aktuelni problemi kod standardnih klijent-server arhitektura. Aplikacije koje su pisane samo za potrebe ovog istraživanja tj. rada osmišljene su tako da što više istaknu propuste u dosadašnjem dizajnu tako da bi dizajn rešenja bio što lakše prepoznatljiv i razumljiv.

2 Pojava Interneta kao globalne mreže

Može se slobodno reći da poslednjih deceniju pa i više godina Internet igra dominantu ulogu kako na polju Informatike tako i u ostalim domenima. Ne samo što se razvio u svjetskog najvećeg snabdjevača informacija nego je postao i mjesto za razmenu email pošte, omogućava obavljanje elektronskog poslovanja, obezbeđuje funkcionisanje različitih informacionih sistema, koristi se za razmenu zabavnih sadržaja, sve više se koristi za rezervisanje tursitičkih destinacija, avionskog prevoza i sl. Još davne 1969 godine tvorci tadašnjeg ARPANET-a, preteče današnjeg Interneta, nisu mogli ni zamisliti u šta će se njihov, tada tajni projekat američke vojske, pretvoriti i kakav će uticaj imati na živote ljudi pedeset godina kasnije.

2.1 Prvi sajtovi i pojava dinamičkih aplikacija

Početkom devedesetih godina prošloga vijeka, američki naučnik Tim Bernards-Lee predstavlja projekat World Wide Web i objelodanjuje HTTP protokol, koji predstavlja predmet analize u nastavku rada. Nedugo zatim počinju da se pojavljuju i prvi web sajtovi. Broj sajtova raste iz dana u dan i nastaju prvi komercijalni sajtovi, a počinju da se formiraju tzv. Internet kompanije. Web servisi postaju osnova za najveći broj aplikacija na Internetu. Svi tadašnji web sajtovi su se oslanjali gotovo isključivo na HTML markap jezik i HTTP protokol. Tek kasnije se uvode dodatne tehnologije kao što su JavaScript, CSS i jezici koji se izvršavaju na strani servera kao što su CGI skripte, PHP itd.. Kako popularnost web sajtova eksponencijalno raste kompanije nastoje da što više svojih usluga prema klijentima pruže *on-line* tj. preko Interneta. Jednostavno informisanje klijenta više nije dovoljno. Bilo je potrebno kreirati dinamički sadržaj, dati korisniku više kontrole i omogućiti mu interakciju sa sadržajem. Očigledan primjer potrebe za interakcijom na relaciji korisnik – web sajt jesu web sajtovi kompanije koje su se bavile e-commerce tj. prodajom preko Interneta. Ovakvi sajtovi su morali omogućiti korisniku funkcionalnosti izbora i naručivanja određenih artikala. Ove funkcionalnosti nije bilo moguće postići korišćenjem samo HTML jezika za označavanje. Potrebno je bilo osmisliti tehnologije koje bi omogućile takve funkcionalnosti. Pojavom tih tehnologija web sajtovi prestaju da slove za jednostavne *on-line* informatore već sve više poprimaju ulogu web aplikacija koje nastoje da pruže sve funkcionalnosti tradicionalnih desktop aplikacija.

3 Web aplikacije danas:

3.1 Prednosti u odnosu na tradicionalne desktop aplikacije

Poslednjih deceniju pa i više web aplikacije sve više preuzimaju primat nad tradicionalnim desktop aplikacijama. Uzrok tomu su prvenstveno brojne pogodnosti, kako za korisnike tako i za ljude koji se bave razvojem istih, u odnosu na desktop aplikacije. Jedna od najbitnijih prednosti je svakako centralizovani kod. Izvršni kod web aplikacije se obično nalazi na jednom serveru kojem pristupaju korisnici te aplikacije. To znači da svakim ažuriranjem koda svi korisnici automatski imaju najnoviju verziju bez potrebe bilo kakvog ažuriranja. Kod desktop aplikacija kada se pojavi nova verzija iste svi korisnici, da bi imali najnoviju verziju aplikacije, moraju da ažuriraju kopije tj. instalacije koje se nalaze na njihovim računarima. Ovo je jako loš ali neizbežan pristup jer ukoliko se radi o aplikaciji koju koristi više hiljada ili čak stotina hiljada korisnika proces ažuriranja može da stvori ogromna opterećenja na serverima sa kojih aplikacije svlače ažuriranja, zagušenja u mreži i sl.. Ukoliko je slučaj o široko korišćenoj aplikaciji gotovo je nemoguće postići da svi klijenti imaju ažurnu verziju jer može da se desi da korisnik nema pristup ažuriranoj verziji (nema izlaz na Internet), postoji greška u njegovoj kopiji aplikacije koja sprečava ažuriranje ili jednostavno korisnik ne želi da ažurira svoju kopiju iz nekih trećih razloga. Prisjetimo se samo problema kompanije Microsoft i njenog operativnog sistema Microsoft Windows XP. Većina ljudi nije željela da ažurira svoju kopiju OS-a na novu verziju tj. Windows Vista što je stvorili velike sigurnosne probleme pogotovu kad imamo u vidu da je ova kompanije i zvanično ugasila podršku za ovaj OS. Kod web aplikacija ovaj problem ne postoji jer ukoliko možete da pristupite aplikaciji možete biti poprilično sigurni da imate najnoviju verziju iste. Pored problema ažuriranja desktop aplikacije se susreću is sa uslovno rečeno problemom instalacije.

3.2 Tehnologije koje se koriste pri izradi

Da bismo koristili određenu aplikaciju na našem računaru potrebno ju je naravno prethodno instalirati na isti. Ovaj proces je u većini slučajeva trivijalan i ne zahtijeva pretjerano poznavanje rada na računaru korisnika koji vrši instalaciju, dakako ovo i nije tačno u 100 odsto slučajeva. Katkad imamo aplikacije koje zahtijevaju određene uslove da bi radile. Ponekad je potrebno instalirati određene radne okvire ili određene tehnologije da bi aplikacija radila. Uzmimo za primjer da instaliramo Java aplikaciju. Da bi Java aplikacija

radila na našem računaru potrebno je da imamo instaliran JRE¹ tj. Java Runtime environment što dodatno komplikuje proces same instalacije. Iako je, kao što sam već napomenuo, proces instalacije trivijalan ponekad, a često kod nestručnih korisnika, može da dovede do odustajanja i potrage za drugim rešenjem. Potrebno je naglasiti da su desktop aplikacije u većini slučajeva platformski zavise što znači da aplikacija pisana za Mac OS neće raditi na nekoj od mnogobrojnih Linux distribucija. Postoje načini da se ovaj problem prevaziđe ali to je moguće izvesti samo uz korišćenje dodatnih tehnologija kao što je gore pomenuta JRE. Ideja web aplikacija jeste da one ne zahtijevaju instalaciju niti su platformski zavisne. Za njihovo korišćenje je potrebno imati instaliran samo web pretraživač i ništa više. Ovakve uslove je jako teško postići baš zbog ograničenja koja se ogledaju u standardima na koje se oslanjaju današnje web tehnologije a koje će biti razmatrane u nastavku rada. Zbog ovih ograničenja ljudi koji se bave razvojem web aplikacija često pribjegavaju upotrebi tehnologija treće strane da bi prenijela sve funkcionalnosti desktop aplikacije na web. Web aplikacije koristimo svakodnevno, bilo da čitamo mail ili koristimo usluge neke od mnogih socijalnih mreža bilo da istražujemo daleke gradove pomoću nekog od servisa kao što je Google Maps ili provodimo slobodno vrijeme igrajući neku od bezbroj igrica on line u suštini samo koristimo usluge određene web aplikacije. Sve one implamentirane su manje više korišćenjem istih tehnologija. Možemo da prepoznamo tri glavna segmenta u dizajniranju web aplikacija a to su. Klijentska strana, serverska strana te komunikacija između te dvije strane. Klijentska strana aplikacije se obično realizuje u web pretraživaču korišćenjem HTML jezika za označavanje te tehnologije koje svoju primjenu nalaze na klijentskoj strani. Najpoznatija od njih jeste JavaScript. JavaScript jeste programski jezik koji se izvršava na strani klijenta i omogućava nam manipulaciju nad HTML dokumentom čime postićemo kako razne animacije tako i dinamiku sadržaja. Za vizuelnu reprezentaciju sadržaja koristi se CSS tehnologija tj. css jezik pomoću kojeg diktiramo izgled html elemenata. Ponekad se koriste i tehnologije treće strane ali samo da bi se prevazišla ograničenja gore navedenih tehnologija. Ograničenja se najčešće ogledaju u nemogućnosti obrade video i audio datoteka. Ovi problemi se postepeno rešavaju tako da se upotrebe rešenja treće strane sve više smanjuje. Primjera radi, ranijih godina sva animacija na web sajtovima bila je moguća jedino ukoliko bi se koristili Adobe Flash ili neka slična rešenja. Danas je to moguće zaobići korišćenjem JavaScript-a ili čak CSS3 tako da tehnologije kao što je Flash

¹ Java virtuelna mašina - Internet programiranje pomoću programskog jezika JAVA, Nikolić B.

sve više gube na značaju. Takođe bitno je istaći da do skora nije bilo moguće omogućiti streaming video ili audio fajlova na web sajtu bez korišćenja rešenja treće strane. HTML5 standard, o kojem će biti riječi u nastavku rada, prevazilazi ovaj problem uvođenjem video i audio oznaka. Naravno težnja jeste na eliminisanju rešenja treće strane, jer ona podrazumijevaju dodatnu instalaciju softvera na računar korisnika što se kosi sa osnovnom idejom web aplikacija, i pojednostavljenju izrade aplikacija. Što se tiče serverske strane tu dominiraju jezici kao što su PHP, Java, .NET i ostali. Serverska strana je motor svih web aplikacija, ona obavlja sav teški posao. Na serverskoj strani se vrši većina obrade podataka, njihov skladištenje i razne operacije nad istim. Klijentska strana služi samo da predstavi rezultate obrade serverske strane. Pored nabrojanih jezika na serverskoj strani često nailazimo i na server baze podataka, web server koji služi web strane te još neke opcione komponente. Poslednji segment jeste komunikacija između serverske i klijentse strane. Upravo u ovom segmentu nailazimo na najveću prepreku u razvoju modernih web aplikacija. Web aplikacije kao takve omogućavaju korisniku interakciju sa sadržajem. Realizacija ove interakcije postiže se tako što korisnik inicira određeni zahtjev, tj. zahtjev se proslijedi serveru koji ga obradi i vrati rezultate klijentu. Uzmimo za primjer mail servis kompanije google. Korisnik počinje rad tako što zatraži od servera formular za prijavljivanje na sistem, jer je sistem tako dizajniran da svaki korisnik ima svoj nalog i pod tim nalogom svoje mailove. Dakle, korisnik upućuje zahtjev serveru a formular za prijavu. Server mu šalje taj zahtjev, web pretraživač potom učitava odgovor servera tj. konkretni formular i korisniku se pred očima pojavljuju polja za korisničko ime i lozinku za prijavu na sistem. Kada korisnik primi odgovor servera sva komunikacija između njih prestaje i klijent i server više ne znaju jedan za drugog sve dok se ne uputi novi zahtjev. Ovakav problem nastaje zbog činjenice da se komunikacija na relaciji klijent server realizuje korišćenjem HTTP protokola koji je stateless tj. ne održava stalnu komunikaciju između klijenta i servera. HTTP protokol nije ni dizajniran da omogući full duplex komunikaciju i nesmetano slanje paketa naprijed-nazad preko jedne otvorene konekcije. Apstrakcija logike HTTP protokola je sledeća. Klijent inicira konekciju ka serveru, kada se konekcija uspostavi klijent šalje svoj zahtjev, server prihvata zahtjev, obrađuje ga te rezultate šalje klijentu čime se konekcija zatvara. Svaki novi zahtjev predstavlja otvaranje posebne konekcije ka serveru. U html dokumentu, svaka referenca ka spoljnjem CSS fajlu, svaka referenca ka spoljnjem JavaScript fajlu, svaka fotografija svi oni predstavljaju zaseban HTTP zahtjev prema serveru tj. zasebnu konekciju. Kada se Internet pojavio ovo nije bio veliki problem jer lepeza sadržaja nije bila toliko široka kao

danas te samim tim nije bilo mnogo saobraćaja na relaciji klijent server. Danas je to veliki problem pogotovu kada se realizuju moderne web aplikacije. U nastavku rada biće izložen problem komunikacije na relaciji klijent server te načini njegovoh prevazilaženja.

4 HTTP protokol

HTTP (***Hyper Text Transfer Protocol***) je protokol aplikacionog sloja OSI modela koji predstavlja temelj komunikacije mreže svih mreža tj. Interneta. HTTP kao standard pojavljuje se u junu 1999 godine. Na razvoju ovog standarda radili su IETF (Internet Engineering Task Force) i W3C (World Wide Web Consortium). Publikacija ovog standarda je izdana kao niz RFC(*Requests for Comments*) dokumenata od kojih je svakako najznačajniji RFC 2616. HTTP funkcioniše kao zahtjev-odgovor (eng. *request-response*) protokol u klijent-servr paradigmi. Najčešće web pretraživa, koji predstavlja klijenta, uputi odgovarajući zahtjev za određenu web stranicu računaru na kome se vrši hosting određenog web sajta tj. aplikacije koji u ovom slučaju predstavlja server. Server prihvata zahtjev, obrađuje ga, ponekad je potrebno vršiti filtraciju materijala koji će biti poslat klijentu ili neke druge proračune prije nego što se odgovor pošalje klijentu, te valjan odgovor vraća klijentu koji nakon toga renderuje dobijeni odgovor i rezultate prikazuje korisniku. Odgovor se obično sastoji iz dva dijela i to zaglavlja i tijela. U zaglavlju se nalaze meta informacije dok tijelo odgovora sadrži realan sadržaj. HTTP protokol je dizajniran tako da može da se odvija i preko posrednika u komunikaciji kao što su recimo proxy serveri. Web sajtovi i aplikacije koje imaju veoma veliki mrežni promet često koriste pomoć keš servera koji na određeni vremenski period vrše keširanja sadržaja tako da klijent tj. korisnik dobije brži odziv sajta ili aplikacije, ovo naravno uz sve probleme koji nastaju prilikom upotrebe keš mehanizama kao na primjer zastarjeli sadržaj. kao što je već napomenuto HTTP je protokol aplikacionog sloja OSI modela dizajniran u okviru Internet Protocol Suite radnog okvira. Po svojoj definiciji podrazumijeva da protokol implementiran na transportnom sloju bude pouzdani transportni protokol. HTTP se najčešće oslanja na TCP protokol kao pouzdani protokol transportnog sloja mada HTTP protokol može i da se osloni na nepozdani transportni protokol kao što je recimo UDP. HTTP resursi koji se nalaze na mreži su identifikovani i locirani pomoću URI-a (eng. *Uniform Resource Identifiers*) ili preciznije URL-a (eng. *Uniform Resource Locators*) koristeći http ili za šifrovani tj. zaštićeni režim rada https. HTTP 1.1. verzija predstavlja nadogradnju na postojeću HTTP 1.0. verziju i donosi mogućnost ponovnog korišćenja iste konekcije ka serveru za preizimanje novih sadržaja (fotografija, skripti, stilova) čime se smanjuje latentnost jer se eliminiše ponovna uspostava TCP konekcije. Iako napredak u odnosu na prethodnu verziju ovo i dalje nije konačno rešenje problema ove komunikacije.

4.1 HTTP sesija

HTTP sesija predstavlja niz zahtjev odgovr sekvenci. Sesija započinje sa prvim zahtjevom klijenta upućenim serveru na odgovarajuću adresu i port, tipično se koristi port broj 80 za komunikaciju bez šifrovanja te port 443 za šifrovanu komunikaciju. Server osluškuje za zahtjeve na top odgovarajućem portu te nakon što primi zahtjev klijenta šalje odgovarajući status. Ukoliko je sve u redu klijent će da primi statusni kod 200 što znači da nije došlo ni do kakve greške. Opšte poznati statusni kod jeste i 404 što znači da traženi resurs nije dostupan ili 500 koji server šalje ukoliko se dogodio neku unutrašnji problem. Nakon statusnog koda tj. statusne linije server počinje da isporučuje sadržaj. Po isporuci poslednjeg dijela sadržaja sesija se prekida.

4.2 Metodi za zahtjev

Postoji devet metoda za zahtjev. Ovim metodima se serveru definiše kakvu akciju da izvrši nad željenim resursom. Zavisno od reprezentacije resursa na serveru, da li je riječ o već postojećim podacima u obliku datoteke ili recimo iste trepa proračunati na osnovu nekih trećih podataka. Najčešći slučaj jeste da su to podatci zapisani u neku datoteku koju valja isporučiti klijentu. Razlikujemo sledeće zahtjeve:

- **HEAD** - zahtijeva odgovor od servera identičan odgovoru GET zahtjeva ali samo u vidu meta podataka tako da se realan sadržaj ne isporučue
- **GET** - zahtijeva od servera isporuku reprezentacije specifičnog resursa. GET zahtjevi treba samo da preuzmu odgovarajuće podatke te ne bi trebalo da imaju nikakvih nusefekata
- **POST** - pomoću ovog metoda klijent može da pošalje odgovarajuće podatke ka serveru. Primjenu ovog metoda možem susresti kad god koristimo HTML formulare te imamo potrebu da njihov sadržaj pošaljemo web serveru na obradu
- **PUT** - transferuje reprezentaciju specifičnog resursa
- **DELETE** - briše odgovarajući resurs
- **OPTIONS** - vraća listu HTTP metoda koje su podržane od strane servera za odgovarajući URL.
- **TRACE** - ispisuje primljeni zahtjev tako da klijent može uočiti eventualne promjene

prouzrokovane posrednicima u komunikaciji

- **CONNECT** - konvertuje zahtijevanu konekciju u transparentni TCP/IP tunel. Obično da bi se omogućila šifrovana komunikacija preko HTTP proxy-ja koji nemaju sigurnosnu konekciju omogućenu
- **PATCH** - koristi se da bi se primjenile odgovarajuće modifikacije željenom resursu

//Primjer jednog zahtjeva za dokument od strane klijenta:

```
GET /index.html HTTP/1.1
Host: www.singiduum.ac.rs
```

//Primjer odgovora na HTTP zahtjev:

```
HTTP/1.1 200 OK
Date: Mon, 07 Jun 2012 13:42:34 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
Last-Modified: Mon, 05 Jun 2012 10:30:55 GMT
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
...
```

Tri tačke označavaju sadržaj traženog dokumenta.

4.3 Sigurni metodi

Metodi kao HEAD, GET, OPTIONS, TRACE označavaju se kao sigurni metodi jer njihovo izvršavanje ne utiče na promjenu stanja servera. Njihova primjena trebalo bi da se svodi samo na preuzimanje dogovarajućih informacija sa servera te da ne inicira bilo kakve procese na serveru koji bi mogli dovesti server u stanje koje nije validno iako ovo nije sasvim tačno, pogotovu kod moderih web aplikacija koje koriste GET metod za pozivanje svojih funkcionalnosti.

4.4 Statusni kod

Prva linija HTTP odgovora HTTP 1.0 i ranijih verzija jeste statusn linija. Statusna linija sastoji se od tekstualne fraze i statusnog koda. Na primjer, ukoliko klijent uputi zahtjev server te isti obradi zahtjev i sve prođe bez problema klijent će dobiti statusni kod 200 te tekstualnu frazu "OK". Ukoliko recimo klijent uputi zahtjev serveru te server ne uspije da pronađe potraživani resurs klijent će dobiti statusni kod 404 te poruku "Not Found". Moguće je uvesti i porizvoljne statusne kodove ali oni će biti mogući jedino ukoliko ih i klijent koji potražuje zahtjev zna tumačiti. Takođe moguće je i na primjer lokalizovati statusne poruke tako da "Not Found" može biti prepravljeno u "Resurs nije pronađen".

4.5 Otvorena konekcija

Kod HTTP protokola ranijih verzija konekcija je bila zatvarana odmah nakon što bi server isporučio odgovor klijentu. Svaki novi zahtjev klijenta podrazumijevao bi ponovnu uspostavu konekcije sa server te se raskidao odmah pri prijemu odgovora istog. Ovo je još tada prepoznato kao problem te se pojavom HTTP protokola verzije 1.1 nastojao ublažiti. Poboljšanje koje je implementirano novom verzijom ogledalo se u tome što je sada bilo moguće zahtijevati i isporučivati više zahtjeve preko jedne konekcije. Ovo je bilo moguće jer je uveden keep-alive mehanizam koji je omogućio ponovnu upotrebu otvorene konekcije što je doprinijelo smanjenju latentnosti i smanjenju mrežnog protoka te opterećenja servera. Naravno ovaj pomak je bio dobrodošao te je povećao performanse web sajtova i web aplikacija ali on još uvijek nije pravo rešenje problema komunikacije jer i dalje nije bio moguće uspostaviti otvorenu konekciju u full duplex režimu rada između klijenta i servera.

4.6 HTTP stanje sesije

Kao što je već rečeno HTTP je stateless protokol. Ovakvi protokoli ne zahtijevaju od servera da pamti informacije o konekciji ili o stanju iste za svakog korisnika na duže od trajanja jedne transakcije. Samim tim što je stateless protokol server nema mogućnost da identifikuje svakog korisnika po na osob. Ukoliko na primjer korisnik pristupi aplikaciji a pregled elektronske pošte on, da bi koristio istu, mora da se prijavi na sistem sa svojim korisničkim imenom i lozinkom. Pošto se svaka transakcija prema serveru satra nezavisnom, kada korisnik pošalje svoje korisničko ime i lozinku server može da izvrši autentifikaciju te obavijesti klijenta o rezultatu ali sledeći zahtjev klijenta, recimo ukoliko klijent klikne ne opciju primljene poruke, smatra se kao potpuno novi zahtjev jer korisnik ne može da prepozna da se ovaj klijent već uspješno prijavio ta mu ponovo alje formular za prijavu na sistem. Da bi se ovaj problem riješio uveo se mehanizam kolačića (eng.

cookies). Pomoću kolačića web server, po potrebi, kreira jedinstveni niz karaktera za svakog korisnika nazvan Session ID te ga upisuje u svoju memoriju, takođe on šalje taj niz karaktera korisniku koji ga takođe pamti na određeni vremenski period. Sledeći put kada korisnik pristupi serveru prvo se provjerava kombinacija kljnetskog zapisa sesije iz kolačića i serverskog zapisa te ako se podudaraju server zna da je taj korisnik već prethodno prijavio na sistem.

5 AJAX

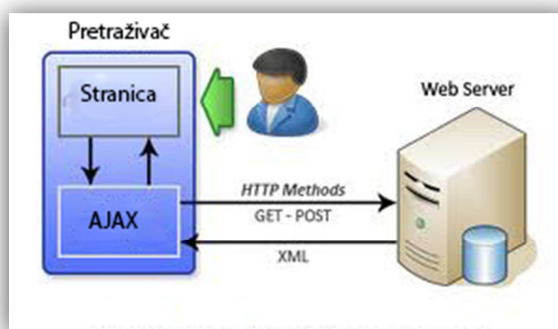
5.1 Uvod

Posljednjih vremena sve je češća upotreba AJAX (eng. *Asynchronous JavaScript and XML*) tehnologije u web aplikacijama. Ovom tehnologijom podstiče se dinamičnost web aplikacija a postala je popularna i zbog činjenice da je koriste „Google Sugest“ i „Google Maps“ aplikacije. AJAX predstavlja novi način upotrebe postojećih tehnologija. Akcenat se stavlja na dinamičnost i asihronost procesa prenosa podatka na relaciji klijent server. Ova nova, unaprijeđena, primjena novih tehnologija donosi sa sobom i nove probleme u pogledu sigurnosti i zaštite krajnjeg korisnika od potencijalnih napada.

5.2 Osnove AJAX tehnologije

AJAX (eng. *Asynchronous JavaScript and XML*) predstavlja skup tehnologija koje služe za razvoj dinamičkih i interaktivnih web aplikacija. U prošlosti su web aplikacije bile isključivo statične što znači da su sadržavale i ograničenja kao na primjer, smanjenu interaktivnost zbog nemogućnosti preuzimanja samo određene porcije podataka s web servera. 1995. godine počela su da se razvijaju programska okruženja koja se izvršavaju na strani klijenta. Neki od novonastalih programskih jezika su: JavaScript, VBScript, kao i Java Applet tehnologija i ActiveX kontrole uz još nekoliko drugih tehnologija. Razvoj navedenih tehnologija kretao se u smjeru prenosa aktivnosti sa servera na klijenta. Iako je razvoj započeo već 1995. godine, javili su se problemi u radu s klijentskim tehnologijama te se velik dio aktivnosti bio vraćen nazad na server. Termin „AJAX“ definisao je 2005. godine Jesse James Garrett. U posljednje vrijeme sve je češća upotreba AJAX tehnologija te većina web aplikacija koristi jedan od oblika klijentskih tehnologija u jednakom obimu kao i serverske tehnologije, odnosno obrada podataka i funkcionalnosti web aplikacija nisu isključivo koncentrirani na serverskoj strani. Tehnologija koja je najčešće korišćena na strani klijenta jeste JavaScript programski jezik. AJAX tehnologija omogućuje razmjenu samo određenih porcija podataka između klijenta i servera, za razliku od statičnih web aplikacija kod kojih se treba ponovno učitati čitava web stranica da bi se promjena na stranici mogla uočiti. Web aplikacije koje se ne zasnivaju na AJAX tehnologiji funkcioniraju prema sinhronom modelu gdje klijent za svaki zahtjev prema serveru dobija odgovor koji je popraćen određenom akcijom u prezentacionom sloju ISO/OSI modela. Uzmimo za primjer da korisnik, nakon što je na određenoj web stranici popunio zadati formular,

pritisne dugme pošalji(submit). Klijent ovom akcijom šalje zahtjev web serveru sa određenim parametrima te podacima koje je prethodno unio u formular. Ovo je uobičajeno ponašanje „klikni i čekaj“ koje u velikoj mjeri ograničava interaktivnost aplikacije. U ovom primjeru korisnik ne dobija povratnu informaciju odmah od servera, on mora pričekati da se ponovno učita čitava web stranica. Navedeni problem moguće je riješiti integracijom AJAX tehnologija u izradu stranica. AJAX omogućava asihronu komunikaciju sa serverom bez potrebe za ponovnim učitavanjem čitave web stranice. Ovu pogodnost omogućavaju tri sastvne komponente AJAX tehnologije a to su: skript jezik koji se izvršava na klijentskoj strani, XMLHttpRequest (XHR) objekt i XML format za enkapsulaciju podataka. Ukratko, skript jezik na klijentskoj strani koristi se kod iniciranja zahtjeva serveru te za programski pristup i ažuriranje DOM objekta u klijentskom web pretraživaču. Kao što je već naglašeno najpopularniji skript jezik jeste JavaScript. Ovaj jezik ima visok stepen podržanosti među modernim web pretraživačima. JavaScript koristi XHR objekt za slanje zahtjeva serveru putem HTTP (eng. *Hyper Text Transfer Protocol*) protokola. XML jezik za označavanje se koristi kao format zapisa podataka čija se razmjena vrši na relaciji klijent server.. Međutim, za postizanje asihronosti sistema nije nužno korišćenje XML kao format zapisa podataka. Mnoge web stranice upotrebljavaju JSON (eng. *JavaScript Object Notation*) format umjesto XML formata jer ga je lakše obraditi. Takođe, mnogo stranica ne koristi niti XML niti JSON kao format zapisa podataka za prijenos. Umjesto spomenutih formata koristi se HTML format (eng. *HyperText Markup Language*), čiji se dijelovi dinamički ugrade u kod stranice. Dakle, AJAX ne uvodi nikakve nove tehnologije, već jednostavno kombinuje, stare, već postojeće tehnologije na novi, bolji način te time omogućava pojačanu interakciju i dinamičnost web aplikacija.



Slika 1 - AJAX šema.

5.3 AJAX tehnologija i sigurnost

Primjena AJAX tehnologija u kreiranju web aplikacija ne donosi nove sigurnosne propuste. Sigurnost web aplikacija koje se oslanjaju na AJAX tehnologiju u velikoj mjeri zavisi od programera koji razvija istu. Treba imati u vidu da ukoliko se ne koristi zaštićena tj. šifrovana komunikacija podaci koji se prenose su javno vidljivi svakome ko ima pristup da vrši monitoring te komunikacije. Važno je istai da AJAX podržava takozvanu "Same origin ploicy" tako da suda je XSS napade teže ali ne i nemoguće izvesti. AJAX aplikacije su, baš kao i sve ostale, ranjive i na "JavaScript injection" napade ali ne zbog načina na koji rade nego zbog tehnologija na kojima se zasnivaju. U nastavku rada biće izloženi neki od sigurnosnih problem koji se javljaju kod AJAX aplikacija.

5.3.1 Uvećana mogućnost napada

Primjena AJAX tehnologija znači i povećanje kompleksnosti sistema klijent-server. Čes je slučaj da se na serveru kreiraju posebne stranice koje implementiraju određenu funkcionalnost, kao što je recimo automatsko dopunjavanje riječi. Takve stranice mogu biti meta napadača i predstavljati još jedan dodatni pristup ranjivosti sistema. Programer ili razvojni tim koji se bavi razvojem date apikacije treba posebno obratiti pažnju na ovakve stranice, ograničiti tj. eliminisati mogućnost zloupotrebe istih, jer one mogu biti plodno tle za napade na aplikaciju. Dakle, aplikacije koje oristi AJAX tehnologiju posjeduju više tačaka pristupa od klasničnih aplikaija. Što više načina da se pristupi aplikaciji, tj. što više mogućnosti da joj se uputi odgovarajući zahtjev to veća vjerovatnoća da će potencijalni napad naći propust u implementacij aplikacije.

5.3.2 Problemi s programskim kodom koji se izvršava na klijentskoj strani

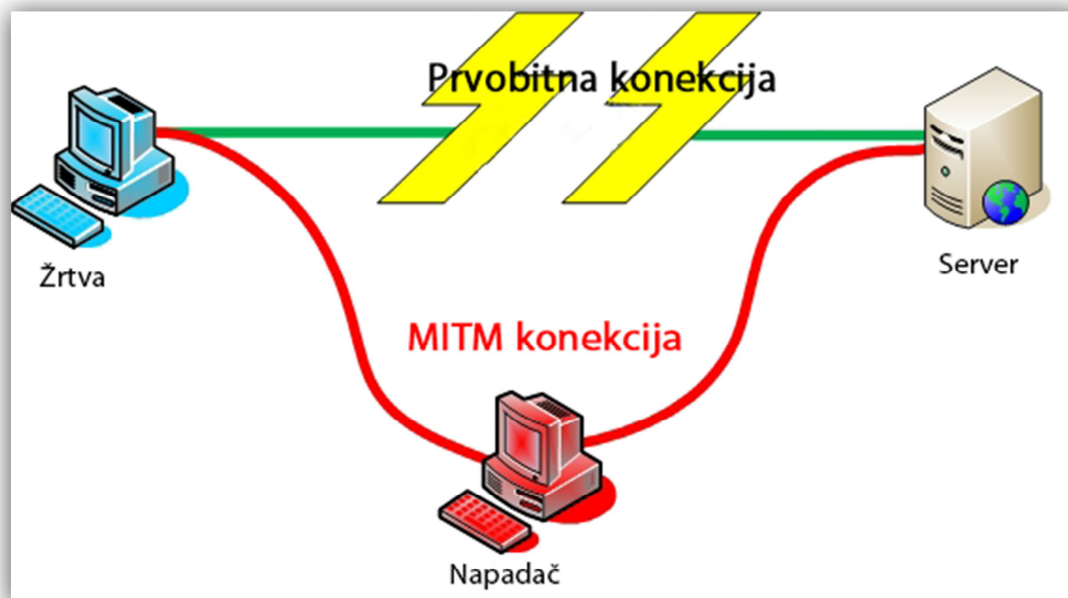
Samim tim što se dio programskog koda izvršava na klijentsom dijelu zathijeva primjenu prikladnih sigurnosnih mjera. Razvojni programeri trebaju posvetiti posebnu pažnju kod implementacije klijentskih kontrola i njihove sigurnosti. Na primjer, ukoliko određana aplikacija od korisnika zahtijeva da se prijavi sa svojim korisničkim imenom i lozinkom da bi je koristio, ta provjera, niti bilo koji njen krucijalni dio, se ne smije odigravati na klijentskoj strani. Razlog leži u tome da korisnik posjeduje svu kontrolu nad kodom koji se izvršava na njegovoj strani te samim tim ima mogućnost da ga izmijeni i prilagodi kako god to njemu odgovara. Najčešće se propusti javljaju kod aplikacija kojima su AJAX

funkcionalnosti dodane naknadno. Programeri kod takvih stranica često implementiraju prikupljanje podataka AJAX pozivima bez ostvarenja sigurnosnih provjera na serveru. Ako ne postoji autorizacijski dio koda na serverskom računaru, napadač može jednostavno dodati nove korisnike te izmijeniti bitne podatke postojećih korisnika, kao što su lozinke. Dakle, za svaki zahtjev za podacima treba postojati prikladna sigurnosna provjera na serversom dijelu. Web aplikacije nikad ne bi trebale vjerovati zahtjevima klijenata, a programeri bi uvijek trebali računati na mogućnost pristizanja zlonamjernih zahtjeva servera. Prema tome, autentifikacija i primjerena provjera ulaznih podataka je obavezna na serveru a na klijentu je samo opcionalna.

5.3.3 AJAX tehnologija i ispitivanje sigurnosti

Kod ispitivanja sigurnosti običnih web aplikacija, ispitivanje počinje prikupljanjem tragova koje aplikacija ostavlja za sobom. Prikupljanje tragova je faza ispitivanja u kojoj se prihvataju zahtjevi klijenta i odgovori servera u svrhu razumijevanja načina komunikacije aplikacije s serverom. Kod ispitivanja sigurnosti web aplikacija (ili izvršavanja napada) moguće je iskoristiti komunikaciju između klijenta i servera. Ispitivač može presretati podatke koji se šalju u komunikaciji. Za takvo ispitivanje obično se koristi posrednik, najčešće računar, na kojem se nalaze posebni programi za testiranje ranjivosti. Javno su dostupni besplatni programi kao što su Webscrab i Paros koji predstavljaju interaktivne proksije za testiranje web aplikacija i izvođenje napada na istim. Moglo bi se reći da ispitivač glumi napadača jer obavlja sve radnje koje bi i napadač obavio. Te radnje su koncentrisane na prikupljanje informacija o internom mehanizmu rada aplikacija. Napad koji uključuje opisani scenario je oblik napada čovjek-u-sredini (eng. *man-in-the-middle*). Prilikom ovakvog napada postavlja računar već instaliranim odgovarajućim softverom postavlja se kao posrednik u komunikaciji klijent-server. To se postiže time što računar koji igra ulogu posrednika "ubijedi" server da je on u stvari klijent koji mu se obraća a klijenta da je on u stvari server kojem ovaj treba da šalje svoje zahtjeve. Nakon ovog koraka sve što je preostalo jeste da posrednik sve pakete pristigle od klijenta proslijedi serveru i obrnuto. Na ovaj način moguće je imati uvid u svu komunikaciju koja se odvija na relaciji klijent server. Ovakav vid napada je jako teško uočiti jer klijent ima utisak da se obraća serveru te niti ne zna da sva komunikacija u stvari ide preko posrednika. Od ovog napada moguće je se odbratiti jedino ukoliko se koristi šifrovana tj. zaštićena komunikacija jer u

tom slučaju iako ima uvid u svu komunikaciju napadač tj. posrednik neće biti u stanju da izvuče bilo kakav podatak jer su isti šifrovani.



Slika 2 - Man-in-the-middle napad

5.4 Problem gubitka pojma "stanja"

Kod običnih aplikacija stanje aplikacije je dobro definisano. Sve što se nalazi u DOM komponenti smatra se trenutnim stanjem stranice. Ako se stanje treba promijeniti, šalje se zahtjev serveru i njegov odgovor određuje kako se stanje mijenja. Za razliku od uobičajnih aplikacija, kod AJAX aplikacija sve se odvija dosta suptilnije. Aplikacija može generisati različite tipove zahtjeva u zavisnosti od trenutnog stanju stranice. Zahtjev koji se stvara kada korisnik klikne na polje za unos podataka može biti različit od zahtjeva koji se stvara kada korisnik prije spomenutog klika označi nešto na stranici. Uz to, odgovor servera može obnoviti samo dio stranice te na primjer korisniku se mogu prikazati nove hiperveze (eng. *link*) ili nove kontrole na stranici. Tokom obavljanja testiranja sigurnosti treba obratiti pažnju na opisano ponašanje jer je mnogo teže odrediti jesu li predviđeni svi tipovi zahtjeva koje web aplikacija može da generiše.

5.5 Vremenski okidači i pokretanje događaja

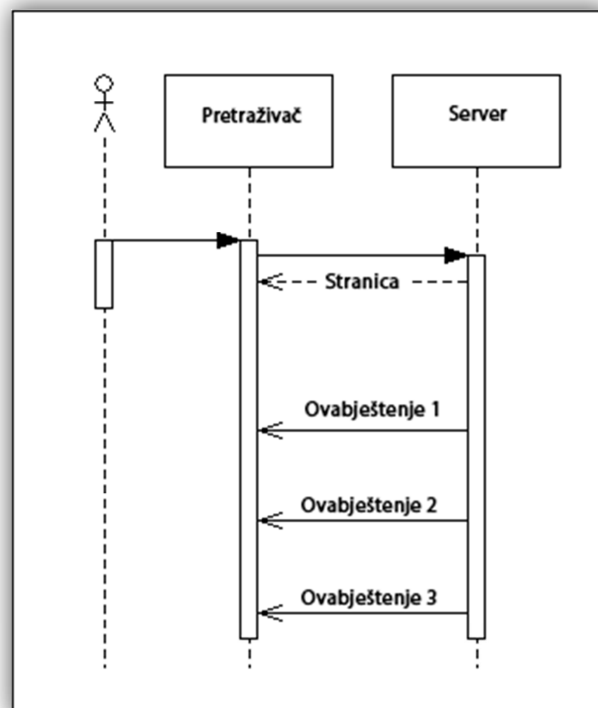
Događaji pokrenuti vremenskim okidačima vezani su uz korisnički interfejs kod kojeg nema interakcije s korisnikom. Aplikacije mogu u određenim vremenskim intervalima slati zahtjeve serverima u smislu obnavljanja informacija prikazanih na web stranici. Na primjer, aplikacija za rukovanje finansijama može koristiti XHR objekt za osvježavanje dijelova stranice koji prikazuju trenutno stanje deviznog kursa. Kod ispitivanja sigurnosti web aplikacije lako je previdjeti da se taj proces odigrava u pozadini. Kako ne postoje korisničke kontrole, zahtjev za osvježavanjem podataka treba uhvatiti u tačno određeno vrijeme.

5.6 Dinamički DOM

AJAX odgovori mogu sadržavati dijelove JavaScript koda kojei web aplikacija treba da obradi i prikaže na korisničkom interfejsu. Dijelovi JavaScript koda mogu sadržavati nove veze, pristup novim dokumentima na serveru itd. Jedan od načina postizanja evaluacije je upotreba funkcije *eval()* koja prihvata jedan parametar, niz znakova, i izvodi ga kao da je dio programa, odnosno obavi određene operacije nad tim nizom znakova i pretvori ih u format kojim se prikazuje sadržaj web stranice. Dobar primjer ovog scenarija je aplikacija „Google Suggest“ gdje web stranica prima dijelove JavaScript koda koji se izlistavaju kao moguće sugestije za dopunjavanje upita unešenog u polje za pretragu. Ponašanje aplikacije u ovom slučaju može stvarati probleme kod ispitivanja sigurnosti, a napadač može funkciju *eval()* iskoristiti za presretanje odgovora servera. U svakom slučaju potrebno je shvatiti kontekst upotrebe JavaScript koda u web aplikaciji. Kod ispitivanja treba posvetiti posebnu pažnju kada se šalje odgovor na zahtijevani parametar čija se evaluacija vrši na klijentskoj strani. Situacija predstavlja tipičnu XSS ranjivost.

5.7 AJAX long polling (Commet)

AJAX long polling tehnologija, poznatija još pod imenima Comet, AJAX push, HTTP Streaming, predstavlja posebnu implementaciju AJAX tehnologije. Naime, ovaj pristup zadržava sve funkcionalnosti "obične" AJAX tehnologije s tim da posjeduje jednu razliku. Ta razlika se ogleda u činjenici da se ovom tehnologijom nastoji što više moguće iskoristiti trenutno aktivna HTTP konekcija da bise preko nje prenijelo što više paketa. Princip rada ove tehnologije je sledeći. Korisnik pomoću AJAX-a inicira HTTP zahtjev prema željenom serveru. Server, nakon što je primio i obradio zahtjev, šalje korisniku niz odgovora sve dok HTTP konekcija ne dostigne svoj maksimum trajanja i ne prekine se. Ovakav pristup predstavlja do maksimuma iskorištava HTTP konekciju jer se ona nastoji održati aktivnom što duži vremenski period. Iako nije široko korišćena ova tehnologija svoju primjenu nalazi kod aplikacija koje zahtijevaju konstantan priliv podataka. Jedan primjer bi bila aplikacija na kojoj bi se nalazile cijene akcija raznih preduzeća neke od svjetski vodećih berzi. Za polje sigurnosti važe sve već navedene opasnosti kao za običan AJAX.



Slika 3 - AJAX long polling

5.8 Rešenja treće strane

Pod rešenjima treće strane podrazumijeva se sav softver koji je potrebno, pored web pretraživača, instalirati da bi se omogućio nemetani rad određene web aplikacije ili web sajta. Zbog ograničenja koje postoje u tradicionalnim web tehnologijama programeri su često pomoć tražili u tehnologijama koje nisu meternje za web. Sa ovim tehnologijama se susrećemo svakodnevno i one su svoju široku primjenu stekle baš zahvaljujući ograničenjima koja HTML standard 5 nsatoji ukloniti. Neke od najšire korišćenih i najpoznatijih su svakako Flash kompanije Adobe, Sun-ova Java te Silverlight kompanije Microsoft. Pomoću ovih tehnologije moguće je implementirati rešenja koja su van dohvata tradicionalnim tehnologijama. Uzmimo video nakratko za primjer. Sve do pojava HTML5 video oznake jedini način da se implementira vido na web sajtu ili web aplikaciji bio je korišćenje nekog od rešenja treće strane najčešće flash-a. Najveća svjetska video biblioteka YouTube sve do par mjeseci je kao jedinu mogućnost pružala streaming videa pomoću Flash tehnologije. Takođe važno je naglasiti da većina chat aplikacija koje nalazimo danas na Internetu svoju funkcionalnost zahvaljuje Java programskom jeziku i njenim Appletima.

5.8.1 Prednosti

Prednosti koje donose rešenja treće strane jesu svakako uvećanje performansi web aplikacija, implementacija dodatnih funkcionalnosti koje nije moguće implementirati korišćenjem samo osnovnih web tehnologija.

5.8.2 Mane

Mane rešenja treće strane ogledaju se u činjenici da korisnik mora da instalira dodatni softver na svom računaru da bi određeno rešenje uopšte moglo da radi. Iako većina ovih rešenja jesu platformski nezavisna neka poput Microsoft Silverlighta ne rade najbolje na svim operativnim sistemima što ne mož da se kaže za tradicionalne web tehnologije za koje se može reći da podjednako dobro rade u svim okuženjima.

5.8.3 Razlika u odnosu na WebSocket protokol.

Iako primjenom ovih tehnologija možemo da riješimo problem komunikacije u realnom vremenu između klijenta i servera to dolazi uz odgovarajuću cijenu. Kao što sam

već napomenuo za izvršavanje koda ovih aplikacija korisnik mora da posjeduje snimljen odgovarajući softver, zatim ukoliko i posjeduje odgovarajući softver i njegov pretraživač uspije da uspostavi konekciju u full duplex režimu rada sa željenim serverom te omogućiti nesmetani protok podataka ponovo može da dođe do problema. Naime uzmimo za primjer sledeći scenario. Korisnik pokušava sa svog radnog mesta da otvori našu aplikaciju koja prati promjene na berzi i u realnom vremenu korisniku dostavlja ažurne informacije o cijenama akcija zadatih preduzeća. Kompanija u kojoj je naš klijent zaposlen posjeduje svoju intranet mrežu koje je isti član. U nastojanju da zaštite svoju intranet mrežu administratori su odlučili da implementiraju firewall te da zabrane sve portove osim nekih standardnih kao što su podrazumijevani portovi za elektronsku mrežu te port 80 koji služi za HTTP komunikaciju. Samim tim što je port 80 rezervisan za HTTP protokol naša aplikacija u koliko je realizovana pomoću recimo Java appleta neće biti u mogućnosti da koristi taj port za internu komunikaciju već će biti primorana da uzme neki ne standardni port kao na primjer 8080. Pošto firewall blokira sve portove sem samo par privilegovanih naša aplikacija neće biti u stanju da nesmetano funkcioniše i samim tim će klijent pokušati da potraži neko drugo rešenje. Ukoliko bismo koristili WebSocket protokol sva komunikacija bi se odvijala preko porta 80 tako da ne bismo imali problem sa posrednicima kao što je firewall niti bi bilo potrebe da klijent instalira dodatni softver.

6 HTML5 standard.

6.1 Uvod

Nemogućnost dosadašnjih HTML standarda da se izbore sa sve većim zahtjevima kojim ih opterećuju moderne web aplikacije dovela je do potrebe za kreiranjem nove verzije ovog opštepoznatog standarda. Novi standard nosi broj 5 jer je on predstavlja petu generaciju svojih prethodnika. Važno je odmah napomenuti da ova verzija HTML standarda još uvijek nije zvanična te da je podložna mijenjanju. Iako nije zvanični standard brojne funkcionalnosti koje on donosi već sada koriste brojni programeri koji se bave programiranjem web aplikacija. Takođe velike kompanije svoje proizvode prepravljaju tako da njihove funkcionalnosti budu implementirane pomoću HTML5 standarda.

6.2 Prednosti u odnosu na prethodnika

Prethodnik HTML-a verzije 5 jeste HTML standard verzije 4. Za ovaj standard nikako se ne može reći da je loš već više da je nepotpun. U vrijeme kada je nastajao ljudi koji su ga dizajnirali nisu imali u vidu kakav bum će doživjeti Internet i kako će se rapidno širiti broj i funkcionalnosti aplikacija na njemu. Verzija 4 pomenutog standarda razvijana je od strane W3 konzorcijuma. W3 konzorcijum je organizacija koja se brine o razvoju Interneta te samim tim i HTML standarda, ona nizom preporuka i specifikacija diktira i usmjerava razvoj modernog weba. Mnoge kompanije iz industrije čiji se proizvodi zasnivaju na ovim standardima nisu bile zadovoljne njime. Smarali su da je on nepotpun, neprecizan i ograničen te su stoga riješile da razviju svoju viziju HTML5 standarda. W3 grupa koja je već razvijala svoju specifikaciju za HTML5 odlučila je da je upotpuni preporukama ljudi iz industrije tako da je nastao jedan jedinstvan HTML5 standard. Specifikacija ovog novog standarda je obimna i precizno dokumentuje kako nove elemente tako i njihovo ponašanje. Ovo sve da bi se izbjegla dosadašnja pojava da proizvođači pretraživača ne moraju proizvoljno da diktiraju ponašanje elemenata koja nisu definisana standardom. Problem je to što se jedan te isti kod ne ponaša istu u dva različita pretraživača jer recimo podrazumijevana širina elementa nije propisana standardom tako da su programeri iz Mozille odlučili da to bude 0 dok su programeri iz Google-a odlučili da to bude puna širica strane. Kada korisnik otvori stranicu iz Firefox pretraživača element na stranici se neće vidjeti jer ima širinu 0 dok recimo kada istu stranicu otvori iz Chrome pretraživača element će biti jedina stvar koju će korisnik vidjeti jer ima podrazumijevanu

širinu od 100%. Pored obimne dokumentacije kojom se rešava problem univerzalnosti koda nova verzija standarda rešava dugogodišnji problem audio i video sadržaja implementacijom novih oznaka koje korisniku omogućavaju da na jako jednostavan način implementira audio ili video na web sajt ili aplikaciju. Novine se ogledaju i u canvas oznaci koja omogućava iscrtavanje dinamičnog sadržaja na sajtu ili aplikaciji što omogućava implementaciju online igara. Ovo je pogotovu popularno jer pomoću web gl grafike programer može implementirati i mnogo zahtjevnije grafičke aplikacije.

6.3 Noviteti koje donosi

Lista novina koje HTML5 donosi je dugačka i skoro sve od njih su bile neophodne za normalan razvoj web aplikacija. To i jeste jedan od razloga što je ovaj standard toliko iščekivan i zbog čega je tema gotovo svih modernih web seminara i skupova. Prije nego što se akcenat prebaci na jednu od bitnijih, možda čak i najbitniju, novina HTML5 WebSockets biće prikazana lista sa nekim od najčešće pominjanih HTML5 novina, web pretraživača te statuska podrške datog pretraživača i novine.

	MAC					WIN								
														
	SAFARI	FIREFOX	OPERA	CHROME	SAFARI	IE	FIREFOX	OPERA	CHROME					
	5.1	11	11.62	18	5.1	6	7	8	9	11	11.61	18		
Local Storage	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	94%	
Session Storage	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	95%	
Post Message	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	96%	
Offline Applications	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	84%	
Workers	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	81%	
Query Selector	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	93%	
WebSQL Database	✓	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓	46%	
IndexedDB Database	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✓		
Drag and Drop	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	94%	
Hash Change (Event)	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	92%	
History Management	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	61%	
WebSockets	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓	42%	
GeoLocation	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	87%	
Touch	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	5%	
File API	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓		
meter element	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓		
progress element	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓		

Slika 4 - Lista novina koje se tiču HTML5 web aplikacija.

Izvor: <http://www.findmebyip.com/litmus/>

Kao što može da se vidi iz gore navedene liste WebSocket protokol je podržan kod svega 42% pretraživača što predstavlja poprilično veliki ograničavajući faktor. Ovaj broj je prije samo godinu dana je bio čak i još manji jer je u to vrijeme Google Chrome bio jedini pretraživač koji je imao podršku za isti. Vidimo kako vrijeme odmiče da sve više i više pretraživača ima podršku za ovaj protokol što po automatizmu znači da će sve više i više korisnika moći u potpunosti da iskorisni sve njegove potencijale.

7 WebSocket

7.1 Opis

WebSocket protokol, kao što je već napomenuto, dolazi kao dio HTML5 standarda. Nastao je kao rešenje problem dvosmjerne komunikacije između klijenta i servera. WebSocket je poseban protokol dizajniran i osmišljen tako da omogući dvosmjernu komunikaciju između klijenta i servera. WebSocket je iskorištava već postojeći HTTP protokol tako da nije potrebno otvaranje novih portova ka serveru kao što je to slučaj sa rešenjima treće strane kao što je Java ili Falash. Pošto WebSocket koristi portove na kojima funkcioniše HTTP protokol nije potrebno praviti nikakva dodatna firewall pravila kojima bi se omogućio protok paketa. Portovi koje koristi HTTP protokol su dozvoljeni već an podrazumijevanim podešavanjima većine današnjih firewall-ova. Takođe pošto se radi o unapređenoj verziji proxz serveri takođe neće praviti nikakav probelm prilikom transporta paketa. Ovaj protokol se inicira tako što klijent šalje serveru HTTP zahtjev sa stavkom Connection: Upgrade koje serveru govori da treba da se prebaci na ws protokol.

7.2 Primjer zahtijva

```
//Primjer iniciranja HTTP zahtjeva sa unapređenjem na WS protokol  
GET /mychat HTTP/1.1  
Host: server.example.com  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==  
Sec-WebSocket-Protocol: chat  
Sec-WebSocket-Version: 13  
Origin: http://example.com
```

7.3 Primjer dobijenog odgovora

Nakon što server primi zahtjev, obradi ga, on šalje odgovor klijentu. Ukoliko i server podržava WS protokol odgovor će izgledati ovako:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

Pošto su uspostavili konekciju klijent i server mogu da komuniciraju u full duplex režimu rada tj. posjeduju mogućnost da međusobno razmjenjuju podatke bez iniciranja novih zahtjeva.

7.4 Sigurnost

WS protokol podržava takozvani "Same origin policy"² sigurnosni pristup već viđen kod AJAX implementacije. Tako da nije moguće premostiti komunikaciju sa jednog između servera i nekog trećeg domena preko postojećeg domena čime se eliminiše mogućnost XSS napada. Takođe, kako se web socekt protokol oslanja na HTTP protokol on je naslijedio i njegova dva osnovna oblika. Kod običnog ws protokola podaci se prenose u plain text formatu tj. bilo ko, ukoliko osmatra mrežni saobraćaj, može da vidi sve informacije koje se šalju jer one nisu zaštićene ni na koji način. Pored ovog pristupa postoji i šifrovani režim rada koji koristi PKI infrastrukturu da bi obezbijedio sigurnu komunikaciju između klijenta i servera. Ovakav pristup onemogućuje zlonamjernog napadača da vidi podatke čak i kada prisluškuje svu komunikaciju. Portovi koje koristi WS protokol jesu standardni portovi HTTP protokola. Za komunikaciju bez šifrovanja koristi se dobropoznati port 80(ws) dok se sva šifrovana komunikacija odvija preko porta 443(wss). Naravno moguće je implementirati WS server koji će osluškivati na nekom trećem portu. Ovakav vid implementacije nosi sa sobom rizik od firewalla jer moguće je da firewall iza kojeg se nalazi klijent ima u svojim podešavanjima zabranjene sve portove se nekih opšte poznatih kao što je na primjer 80. Samim tim komunikacija klijenta i servera preko WebSocket protokola neće moći biti uspostavljena jer je protok paketa nemoguć.

² Wikipedia - http://en.wikipedia.org/wiki/Same_origin_policy

7.5 Prednosti

Velika prednost ove implementacije komunikacije u realnom vremenu između klijenta i servera jeste što se od krajnjeg korisnika ne zahtijeva da ima instalirano bilo šta drugo sem operativnog sistema i web pretraživača koji ima podršku za WebSocket protokol. Napominjem da je ovakav vid komunikacije ranije bio moguć jedino korišćenjem neke od dodatnih tehnologija kao što je Java ili Flash. Ove tehnologije od korisnika podrazumijevaju da ima snimljen dodatni softver koji bi omogućio njihovu realizaciju. Takođe kao što je već rečeno ne otvaraju se dodatni portovi na klijentu i serveru tako da nema potrebe za dodavanje dodatnih pravila za firewall uređaje koji bi bili eventualni posrednici u komunikaciji. Programer upravlja komunikacijom koja se odvija preko ovog protokola pomoću javascripta. Programer ima kontrolu nad slanjem i primanjem paketa te pomoću JavaScript-a tj. odgovarajućij WebSocket API-a ³implementira pozadinsku logiku, obrađuje primljene podatke te ih eventualno prezentuje korisniku aplikacije. Što se tiče podržanosti pretraživača ona je prezentovana na grafiku iznad.

//WebSocket API

```
var myWebSocket = new WebSocket("ws://www.websockets.org");
myWebSocket.onopen = function(evt) { alert("Konekcija otvorena ..."); };
myWebSocket.onmessage = function(evt) { alert( "Poruka primljena: " + evt.data);};
myWebSocket.onclose = function(evt) { alert("Konekcija zatvorena."); };
myWebSocket.send("Pozdrav websocket!");
myWebSocket.close();
```

Što se tiče unapređenja performansi WebSocket donosi jako velika poboljšanja na ovom polju. Uzevši za primjer da aplikacija koju razvijamo ima potrebu za stalnom komunikacijom te da posjeduje potencijal da postane korišćena od strane izuzetno velikog broja korisnika. Ukoliko bismo se oslonili samo na AJAX tehnologiju i HTTP protokol imali bismo sledeći scenario. Imajući u vidu da je veličina HTTP header-a približno 870 bajta a veličina WS header-a samo 2 bajta dobijamo sledeće podatke:

Tradicionalni HTTP zahtjev

- **Slučaj A:** 1,000 klijenata šalje zahtjev svakih 1s: Mrežni protok iznosi (871 x 1,000) = 871,000 bajta = 6,968,000 bita po sekundi (6.6 Mbps)

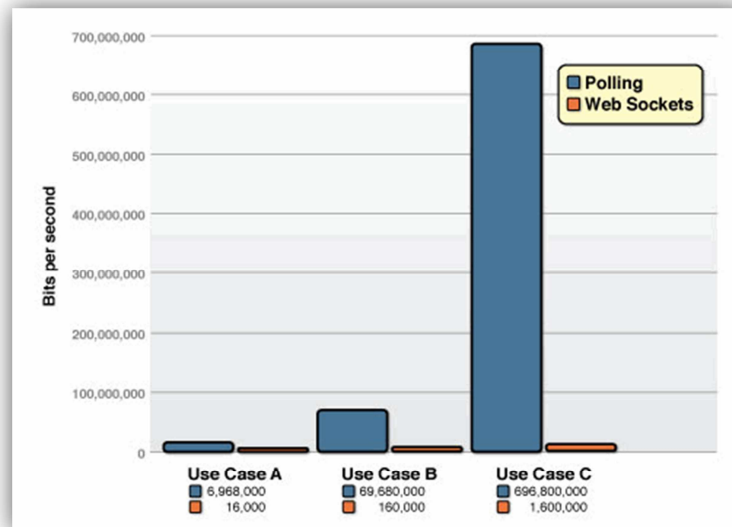
³ Lubbers, P., Albers B., Salim F. (2011) Pro HTML5 Programming (Professional Apress)

- **Slučaj B:** 10,000 klijenata šalje zahtjev svakih 1s: Mrežni protok iznosi $(871 \times 10,000) = 8,710,000$ bajta = 69,680,000 bita po sekundi (66 Mbps)
- **Slučaj C:** 100,000 klijenata šalje zahtjev svakih 1s: Mrežni protok iznosi $(871 \times 100,000) = 87,100,000$ bajta = 696,800,000 bita po sekundi (665 Mbps)

HTML5 WebSocket

- **Slučaj A:** 1,000 klijenata prima 1 poruku za 1 sekundu: Mrežni protok iznosi $(2 \times 1,000) = 2,000$ bajta = 16,000 bita po sekundi (0.015 Mbps)
- **Slučaj B:** 10,000 klijenata prima 1 poruku za 1 sekundu: Mrežni protok iznosi $(2 \times 10,000) = 20,000$ bajta = 160,000 bita po sekundi (0.153 Kbps)
- **Slučaj C:** 100,000 klijenata prima 1 poruku za 1 sekundu: Mrežni protok iznosi $(2 \times 100,000) = 200,000$ bajta = 1,600,000 bita po sekundi (1.526 Kbps)

Vidimo da korišćenje WebSocket-a u ovom scenariju drastično smanjuje mrežni protok te samim time eliminiše zagušenja u mreži. Opterećenje servera koji vrši distribuciju poruka je takođe drastično manje prilikom korošćenja WebSocket protokola nego kod tradicionalnog HTTP metoda. Na slici 5 možemo vidjeti i grafičku prezentaciju gore navedenog oglada.



Slika 5 - Poređenje HTTP i WebSocket protokola.

Izvor: <http://tinyurl.com/d3vwxnz>

7.6 Nedostatci

Serverska strana implementacije WS protokola još uvijek nije potpuno realizovana. Vodeći web serveri kao što su Apache, Nginx ili IIS još uvijek nisu omogućili implementaciju ovog protokola u svoje servere. Postoje određene besplatne tj. open source biblioteke programskih jezika koje omogućavaju implementaciju ws protokola na strani servera ali to su većinom beta verzije koje su podložne čestim prekidima rada i bagovima. Jedan od najpoznatijih jeste Node.js koja predstavlja skup JavaScript klasa koje se izvršavaju na serveru pomoću JavaScript engine-a kompanije google V8⁴. Aplikacije koje su opisane u ovom radu svoje serverske implementacije duguju upravo skupu ovih JavaScript klasa. Takođe je jako bitno razumjeti kontekst u kome treba koristiti WS. Ukoliko se recimo dizajnira aplikacija koja će od servera dobijati određene podatke svakih sat vremena onda to nije prava prilika za upotrebu WS protokola jer će većinu vremena konekcija biti otvorena i neiskorišćena. U ovakvim slučajevima je bolje i pogodnije koristiti samo klasični HTTP.

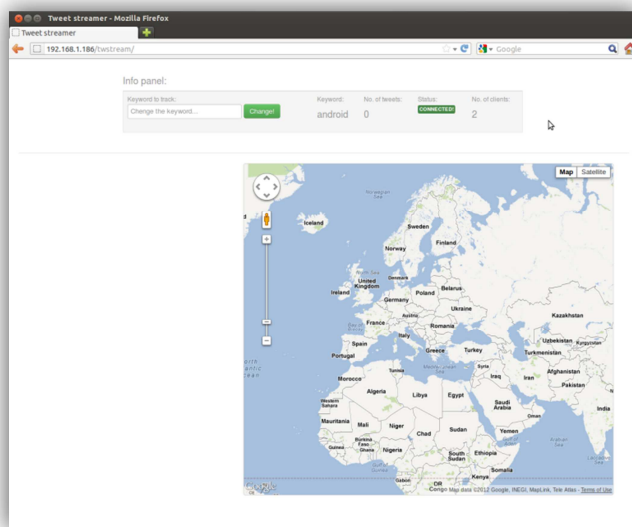
⁴ Google JavaScript Engine - 6. <https://developers.google.com/>

8 Aplikacije:

8.1 TWstream

8.1.1 Uvod

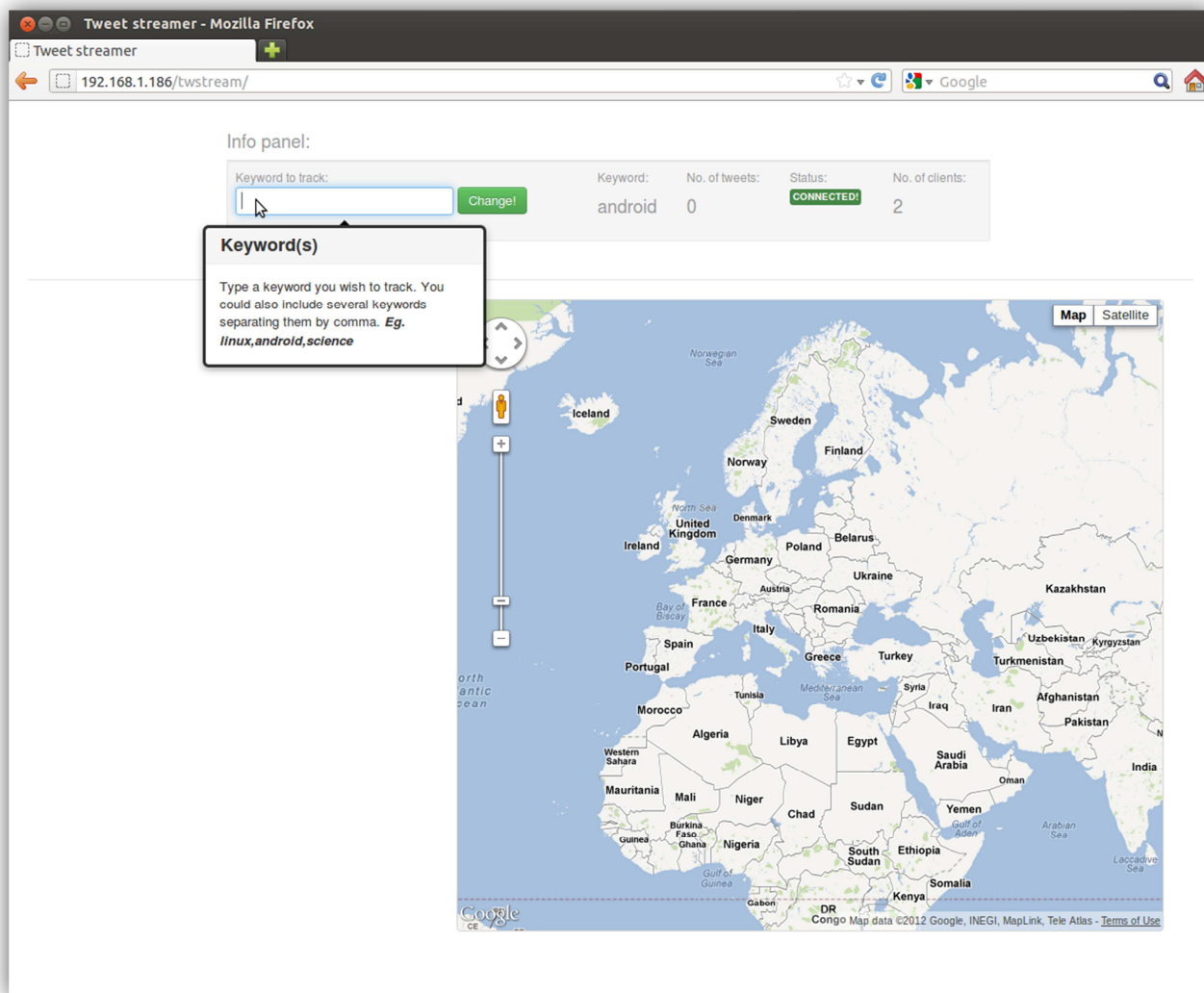
TWStream (*Twitter Stream*) je web aplikacija koja je osmišljena i dizajnirana tako da demonstrira pun potencijal WebSocket protokola te prednost njegovog korišćenja u odnosu na dosadašnja rešenja poput AJAX tehnologije ili rešenja treće strane. TWStream aplikacija omogućuje korisniku da u realnom vremenu prati kratke tekstualne poruke (tvitove) socijalne mreže twitter tako što u odgovarajuće polje unese željenu riječ ili frazu. Svaki tvit koji u sebi sadrži zadatu riječ biće prikazan korisniku. Takođe, ukoliko tvit sadrži geo logaciju korisnik će na mapi mapi moći vidjeti lokaciju osobe koja je poslala taj tvit. Za svoj rad aplikacija se oslanja na javni API koji je omogućila socijalna mreža twitter. Podaci se preuzimaju na web server odakle se distribuiraju svim klijentima putem WebSocket protokola. Razmjena i distribucija odvija se korišćenjem JSON-a. Aplikacija je dizajnirana tako da je za njeno korišćenje potreban jedino web pretraživač koji podržava WebSocket protokol. Nakon pokretanja pretraživača potrebno je samo u polje predviđeno za to ukucati adresu na kojoj se aplikacija nalazi i ista je spremna za rad. Izgled početne strane aplikacije dat je na slici 6. TWStream je predviđea kao javna aplikacija koju svi mogu nesmetano da koriste tako da se nije pridavao preveliki značaj kategorizaciji niti autorizaciji njenih korisnika.



Slika 6 - Početna strana TWStream aplikacije

8.1.2 Početna strana

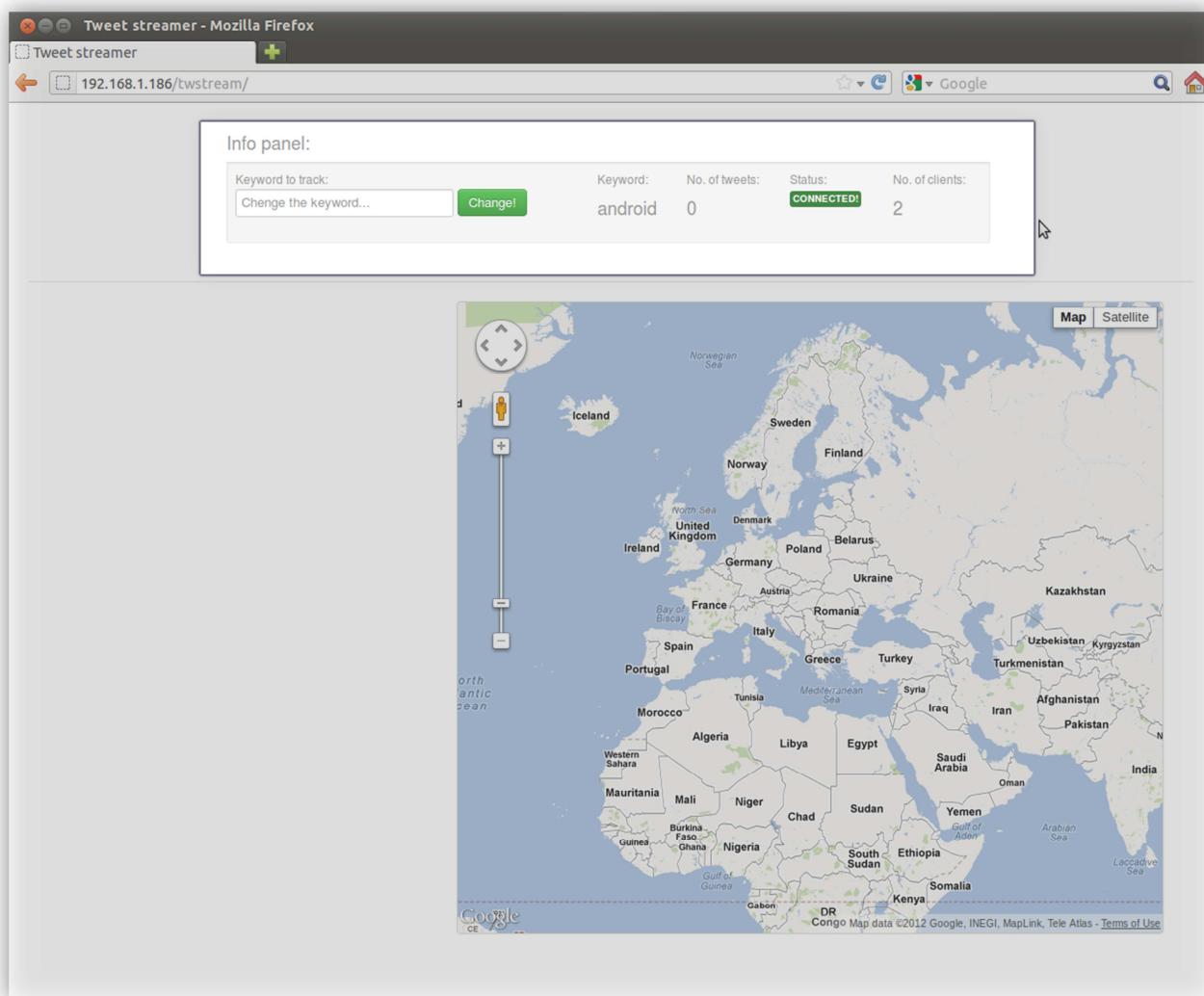
Na početnoj strani aplikacije možemo razlikovati dvije odvojene cjeline (slika 7). Prva cjelina sadrži info panel dok je odmah ispod cjelina koja sadrži poruke korisnika koje će se ispisivati te globalnu, interaktivnu kartu na kojoj će biti iscrtavane lokacije korisnika koji su omogućili funkcionalnosti geolokacije.



Slika 7 Polje za unos ključne riječi.

8.1.3 Info panel

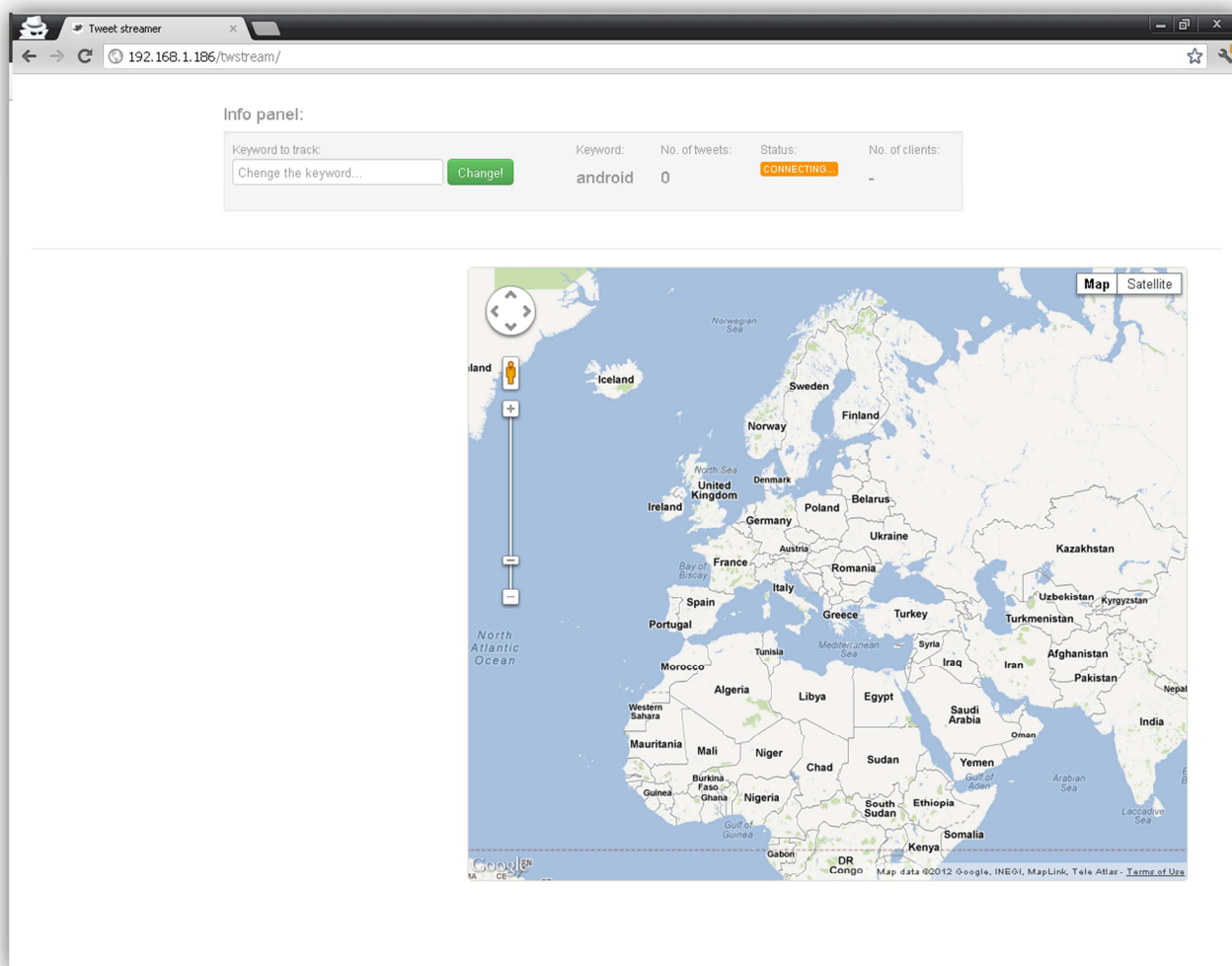
Info panel pruža korisniku uvid u stanje konekcije tj. informaciju da li je korisnik konektovan ili ne, broj izistanih poruka od trenutka konekcije, trenutno aktivnu ključnu riječ i broj konektovanih clijenata. Aplikacija podržava neograničen broj clijenata koji mogu da se konektuju slika 8.



Slika 8 Izgled info panela.

8.1.4 Proces konektovanja

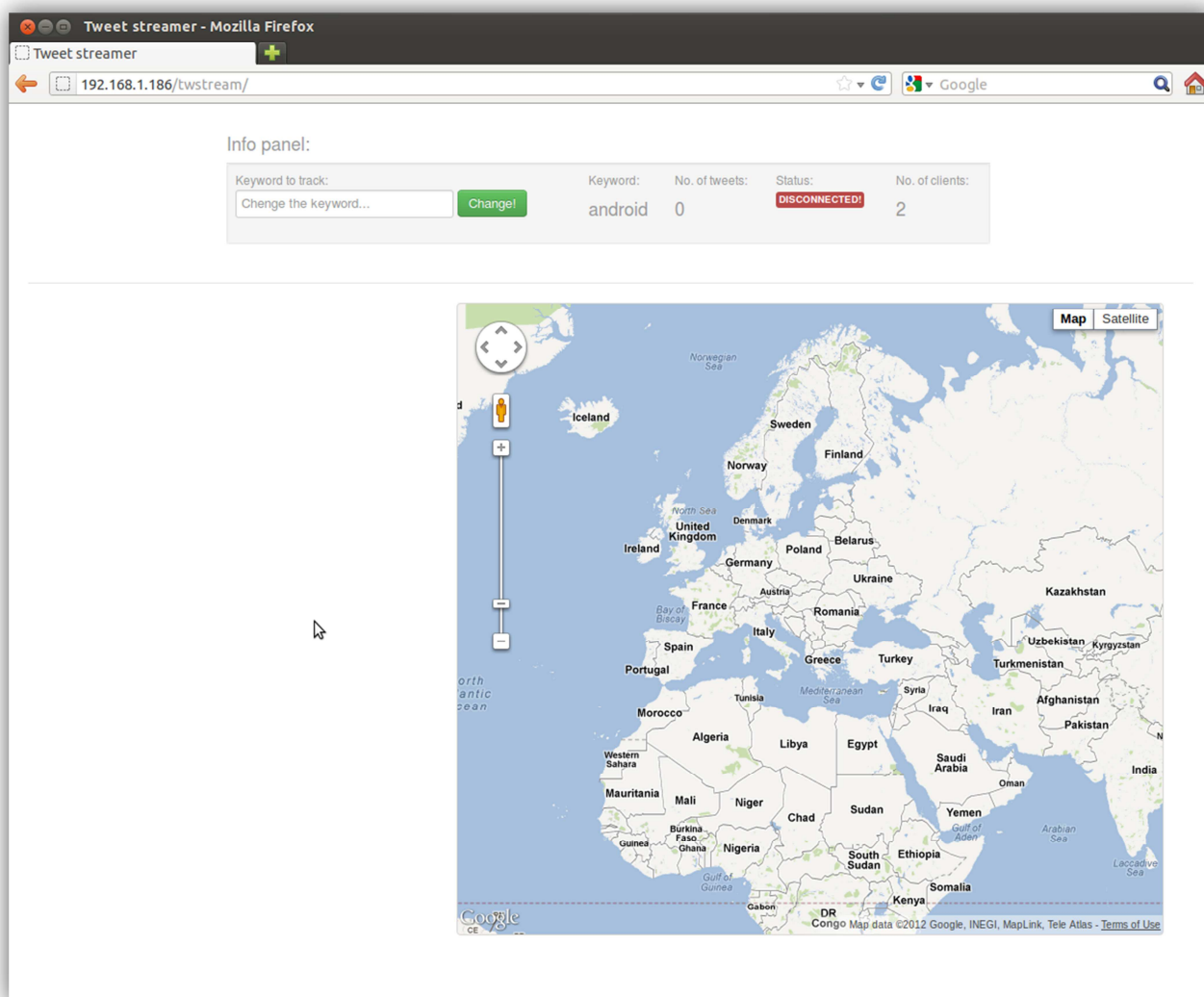
Kada korisnik pokrene aplikaciju započinje se proces konektovanja na server (slika 9). U polju u kojem je naznačen status konekcije biće ispisano "*Connecting*" i ono će biti obojeno žutom bojom. Ovo polje predstavlja indikator konekcije, ukoliko korisnikov pretraživač ne podržava WebSocket protokol konekciju neće biti moguće uspostaviti. Korisnik u svakom trenutku ima realno stanje konekcije.



Slika 9 Započeti proces konekcije.

Ukoliko dođe do bilo kakvog problem u komunikaciji sa serverom. Bilo da se dogodi kvar na klijentovom računaru ili koji slučajem server prestane sa radom polje polje za indicaciju konekcije će reagovati i ispisaće se adekvatna poruka (slika 10). Bitno je naglasiti da mrežni posrednici takođe mogu da prekinu već uspostavljenu konekciju. Kod HTTP

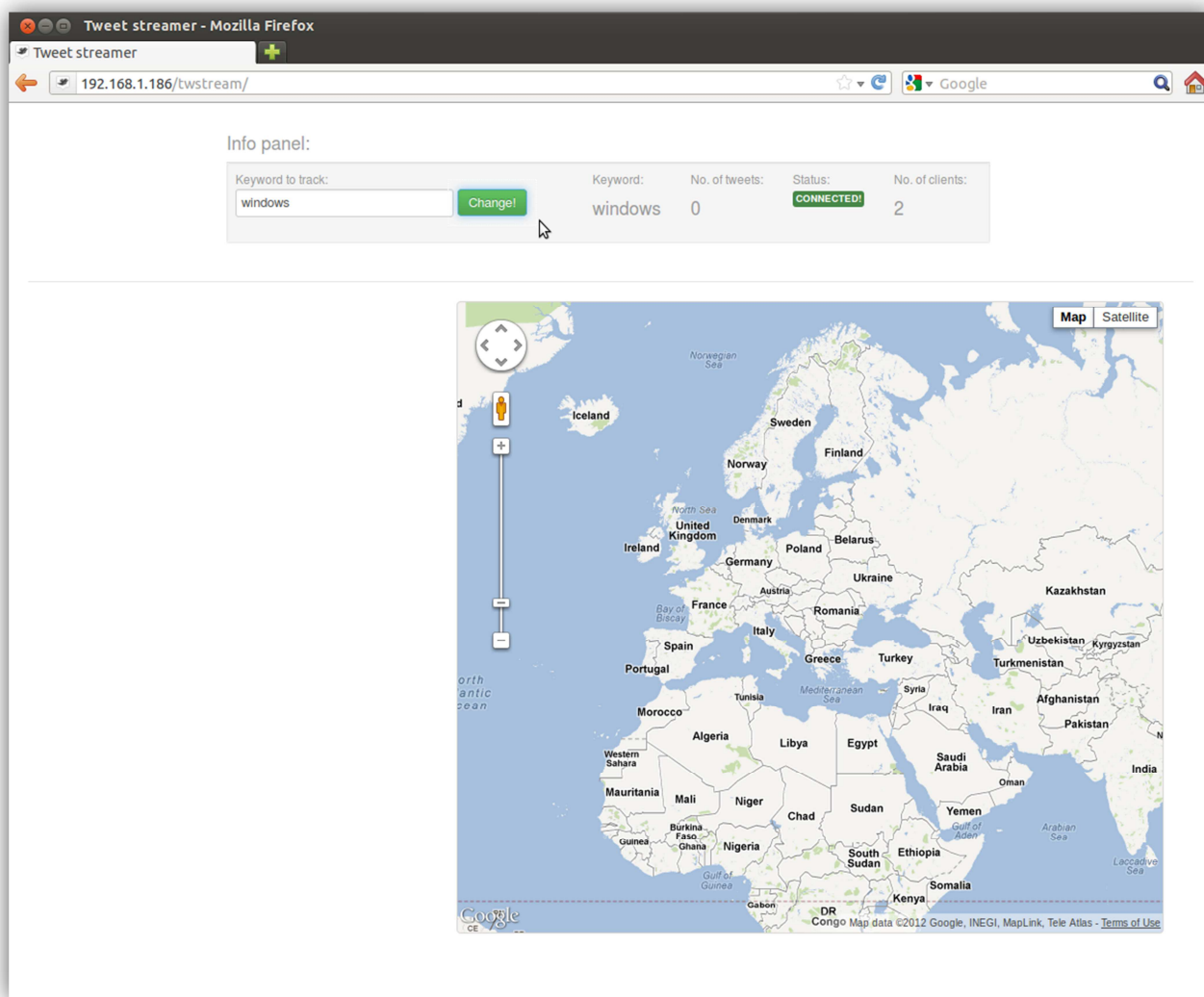
protokola smo imali slučaj da ukoliko bi konekcija bila prekinuta u trenutku kada korisnik ne intereaguje sa aplikacijom te uspostavljena prije nego što se interakcija započne korisnik ne bi ni primijetio da je server nije bio dostupan. Ovo nastaje zbog svojstva HTTP protokola da ne podržava otvorenu konekciju. U ovom slučaju ukoliko se dogodi prekid klijent odmah ima saznanje o tome.



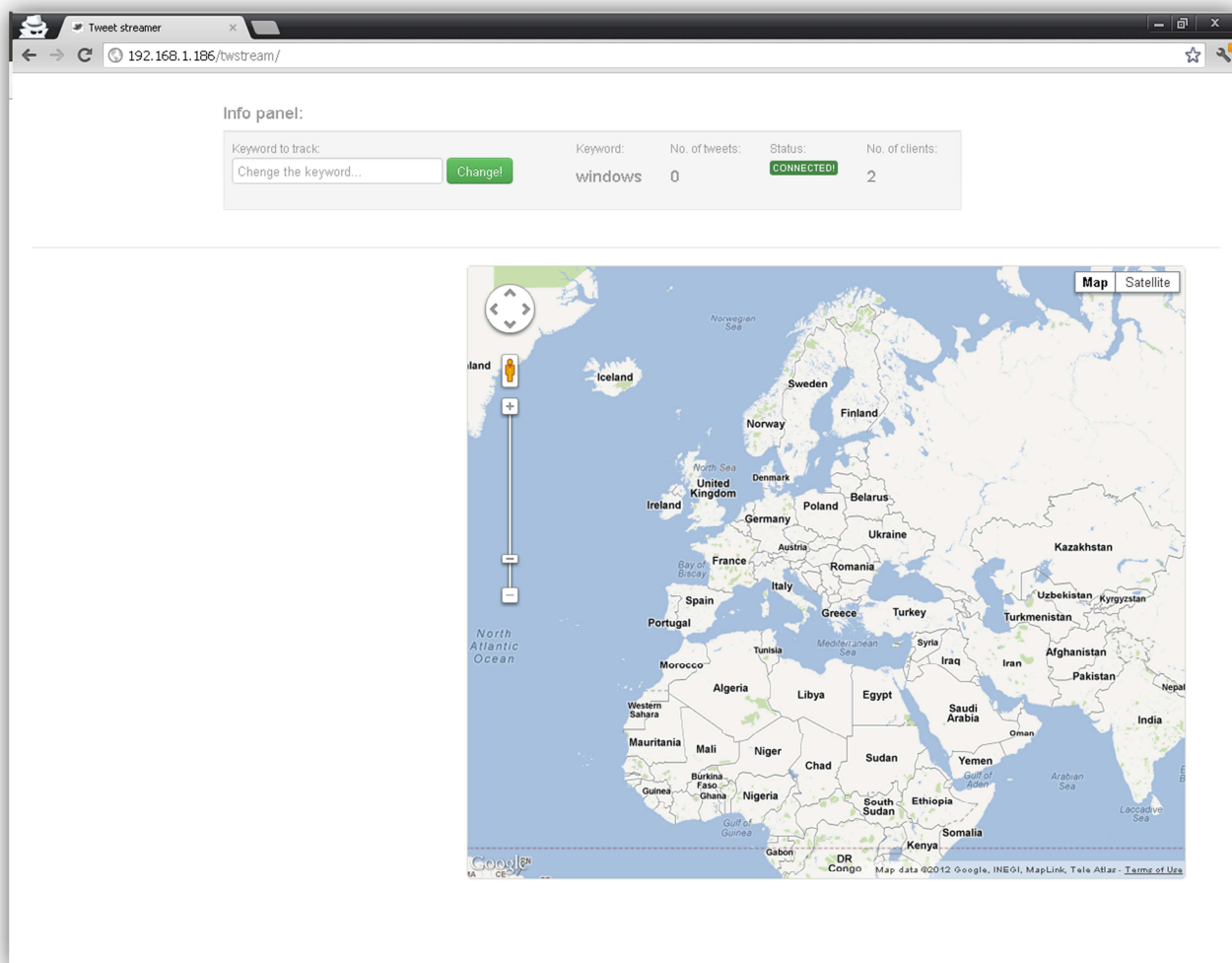
Slika 10 Prekinuta konekcija.

8.1.5 Promjena ključne riječi

Svi konektovani klijenti komuniciraju sa serverom putem WebSocket prookola. Kada samo jedan korisnik promijeni ključnu riječ ona se dalje propagira na sve konektovane klijente (slika 11, 12). Promjena ključne riječi ili fraze je omogućena svim korisnicima jer je težnja aplikacije da pokaže pun potencijal komunikacije putem WebSocket protokola te da je koristi što više korisnika. Autorizacija korisnika radi promjene ključne riječi mogle bi da budu jedan vid dodatnih funkcionalnosti..



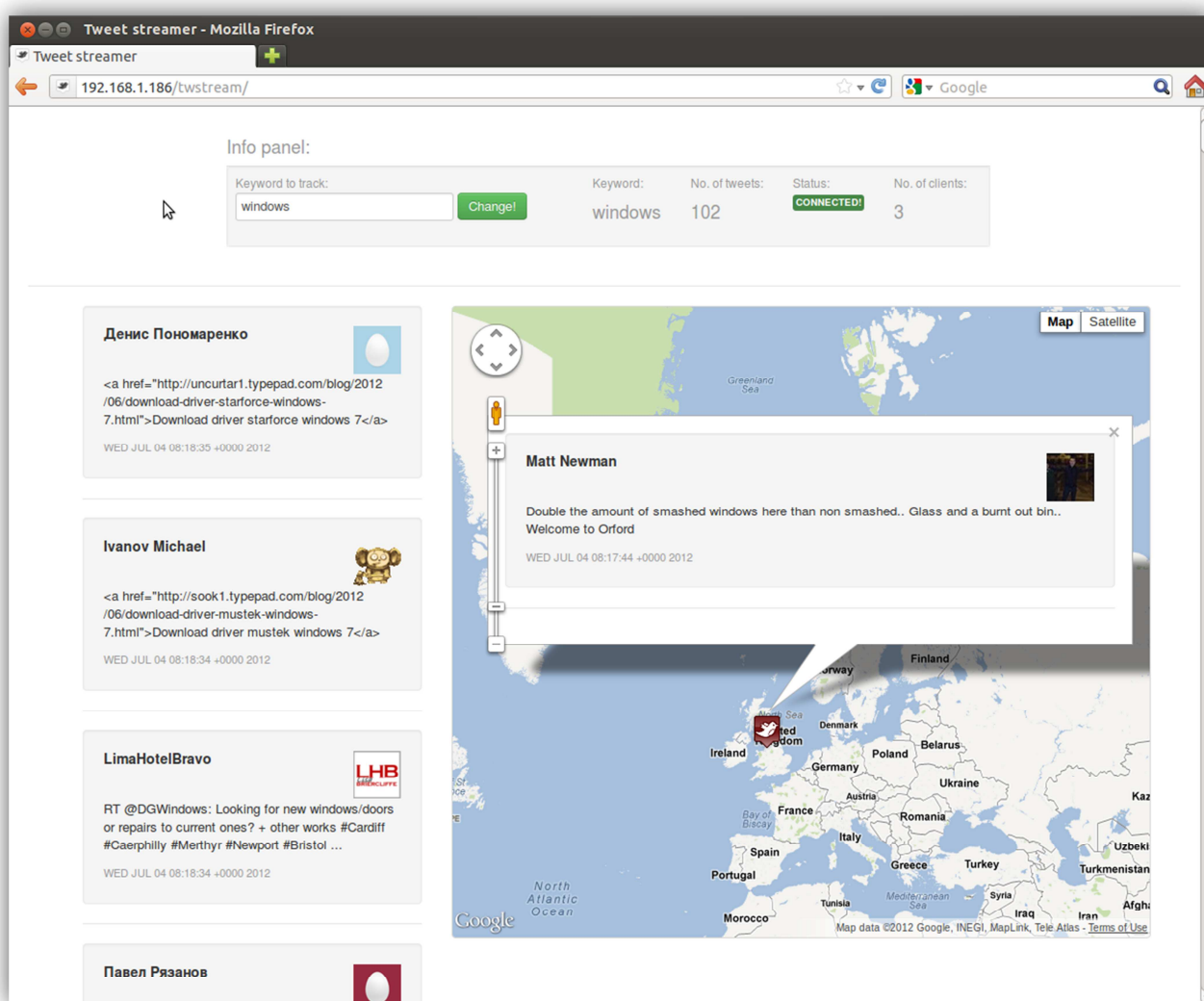
Slika 11 Promena ključne riječi na pretraživaču A.



Slika 12 Promena ključne riječi na pretraživaču B.

8.1.6 Pozicioniranje na karti

Kao što je već naznačeno, ispod panela nalazi se sadržaj podijeljen u 2 kolone. U lijevoj koloni izlistavaju se sve poruke koje u sebi sadrže zadatu ključnu riječ ili frazu dok se u desnoj koloni nalazi interaktivna globalna mapa na kojoj se iscrtavaju markeri za tvitove koji posjeduju geo lokaciju (slika 13). Korisnici koji šalju poruke (tvitove) se na mapi iscrtavaju u vidu crvenih markera. Korisnik ima punu interakciju koje nudi Maps servis kompanije Google. Moguća su uveličavanja, smanjenja mape, promjena tipa prikaza itd. Ukoliko korisnik aplikacije klikne na neki od markera na mapi pojaviti će mu se prozor u kom se nalazi poruka koju je poslao korisnik sa te lokacije te fotografija istog (slika 13).



Slika 13 Izgled sadržaja i tvita sa geo oznakom.

Aplikacija se sastoji iz dva dijela, klijentskog i serverskog. Klijentski dio realizovan je pomoću HTML jezika za označavanje, CSS jezika za stilizovanje i JavaScript programskog jezika koji se izvršava na strani klijenta dok je serverski dio aplikacije reaizovan pomoću prilagođenog JavaScript-a koji se izvršava na strani servera. Korišćena je opensource biblioteka Javascipta nazvana node.js koja se oslanja na JavaScript interpreter kompanije goolge poznat pod nazivom JavaScript. Node.js je niz JavaScript klasa koje su napisane da bi omogućile serverima da implementiraju WebSocket protokol.

8.1.7 Izvorni kod aplikaicije

Izvorni kod klijentskog dijela aplikacije.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Tweet streamer</title>
    <meta content="width=device-width, initial-scale=1.0" name="viewport">
    <meta content="TWStream" name="description">
    <meta content="Simo Simeunović" name="author">
    <link href="images/icons/favicon.ico" rel="shortcut icon" type="image/x-icon">

    <link href="adit_libs/bootstrap/css/bootstrap.css" rel="stylesheet">
    <style type="text/css">
      body {
        padding-top: 20px;
        padding-bottom: 40px;
      }
      .sidebar-nav {
        padding: 9px 0;
      }
      #map img {
        max-width: none;
      }
    </style>
    <link href="adit_libs/bootstrap/css/bootstrap-responsive.css" rel="stylesheet">
    <!--[if lt IE 9]>
      <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
    <![endif]-->
    <link href="../assets/ico/favicon.ico" rel="shortcut icon">
    <link href="../assets/ico/apple-touch-icon-114-precomposed.png" rel="apple-touch-icon-precomposed"
sizes="114x114">
    <link href="../assets/ico/apple-touch-icon-72-precomposed.png" rel="apple-touch-icon-precomposed"
sizes="72x72">
    <link href="../assets/ico/apple-touch-icon-57-precomposed.png" rel="apple-touch-icon-precomposed">
    <script src="http://code.jquery.com/jquery-1.7.2.min.js" type="text/javascript"></script>
    <script src="node_modules/socket.io/node_modules/socket.io-client/dist/socket.io.js"
type="text/javascript"></script>
    <script src="http://maps.googleapis.com/maps/api/js?key=google_maps_api_key"
type="text/javascript"></script>
  </head>
  <body>
    <div class="container-fluid">
      <div class="row span8 offset2">
```

```

<h1><small>Info panel:</small></h1>
<table class="table well">
  <tbody>
    <tr>
      <td>
        <h4><small>Keyword to track:</small></h4>
        <form class="form form-inline" onsubmit= "return false;">
          <input autocomplete="off" class="input-large" data-content= "type a keyword you
wish to track. you could also include several keywords separating them by comma. &lt;em>&lt;b>eg.
linux,android,science&lt;/b>&lt;/em>" data-original-title="Keyword(s)" id="keyword"
placeholder="Chenge the keyword..." type= "text"><input class="btn btn-success" id= "track" type="button"
value="Change!">
        </form>
      </td>
      <td>
        <div>
          <h4><small>Keyword:</small></h4>
          <h2><small><span id= "keyword-show">android</span></small></h2>
        </div>
      </td>
      <td>
        <h4><small>No. of tweets:</small></h4>
        <h2 id="number-of-tweets"><small>0</small></h2>
      </td>
      <td>
        <h4><small>Status:</small></h4>
        <p id="connection-status">
          <span class= "label label-warning">Connecting...</span>
        </p>
      </td>
      <td>
        <h4><small>No. of clients:</small></h4>
        <h2 id="number-of-clients"><small>-</small></h2>
      </td>
    </tr>
  </tbody>
</table>
</div>
<div class="row-fluid">
  <hr>
  <div class="span4" id="tweets">
  </div>
  <div class="span7 well" id="map" style="height: 600px;">
  </div>
</div>
</div>
<script type="text/javascript">
  // MAP INITIALISATION
  var myMap;
  function initialize() {
    var myOptions = {
      center: new google.maps.LatLng(44.8206, 20.4622),
      zoom: 3,
      mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    var map = new google.maps.Map(document.getElementById("map"),
    myOptions);
    myMap = map;
  }
  // GOOGLE MAP INITIALISATION

```

```

$(function() {
    initialize();
    $("[rel=popover]").popover({
        animation:true,
        placement:'bottom'
    });
    // NODE JS
    var noTw = 0;
    var ntwhHolder = document.getElementById('number-of-tweets');
    var socket = io.connect("http://192.168.1.186:80");
    socket.on('disconnect', function() {
        $("#connection-status").html("<span class='label label-important'>Disconnected!</span>");
    });
    socket.on('connect', function() {
        $("#connection-status").html("<span class='label label-success'>Connected!</span>");
    });
    socket.on('newKeyword', function(obj) {
        $("#keyword-show").html(obj.word);
    });
    socket.on('noOfClients', function(obj) {
        $("#number-of-clients small").html(obj.number);
    });
    $("#track").click(function(){
        var word = $("#keyword").val();
        if(word.length > 0){
            socket.emit('launch', {"word":word});
        }
    });
    socket.on('tweets', function(data){
        var holder = document.getElementById('tweets');
        var imageURL = imageURL =
'http://s.twimg.com/a/1281473923/images/default_profile_4_bigger.png';
        $.each(data, function(index, value) {
            if(value.user.profile_image_url != undefined){
                imageURL = value.user.profile_image_url;
            }
            noTw++;
            var contentString = "<div class='well'><table><tr><td><img class='pull-right' "
src='\"'+imageURL+\"'
/><h4>\"+value.user.name+\"</h4><td><tr><td><p>\"+value.text+\"</p><td><tr><td><h6><small
>\"+value.created_at+\"</small><h6><td><tr><td></table></div><hr>";
            $("#number-of-tweets small").text(noTw);
            $("#tweets ul").prepend(contentString);
            if(value.geo){
                var myLatLng = new
google.maps.LatLng(value.geo.coordinates[0],value.geo.coordinates[1]);
                var twitter = 'images/tweet.png';
                var marker = new google.maps.Marker({
                    position: myLatLng,
                    map: myMap,
                    animation: google.maps.Animation.DROP,
                    icon: twitter,
                    title: value.user.name
                });
                addInfoWindow(marker, contentString);
            }
        });
    });
    // NODE JS
});

```

```

// HELPER FUNCTIONS
function addInfoWindow(marker, message) {
    var info = message;
    var infoWindow = new google.maps.InfoWindow({
        content: message
    });
    google.maps.event.addListener(marker, 'click', function () {
        infoWindow.open(myMap, marker);
    });
}
</script>
<footer>
    <script src="adit_libs/bootstrap/js/bootstrap-transition.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-alert.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-modal.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-dropdown.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-scrollspy.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-tab.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-tooltip.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-popover.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-button.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-collapse.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-carousel.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-typeahead.js"></script>
</footer>
</body>
</html>

```

Na početku se nalazi HTML5 doctype koji pretraživaču govori da HTML kod koji slijedi interpretirka kao HTML5 kod te meta podatci o autoru, CSS stilovi, reference ka spoljnim JavaScript i CSS fajlovima. Body sekcija ovog HTML dokumenta sadrži mapu na kojoj će se iscrtavati lokacije tvitova i kolonu u kojoj će biti ispisan tekst tvitova. Takođe u body sekciji se nalazi i JavaScript kod koji je zadužen za ostvarivanje komunikacije ka serveru i obradu svih pristiglih poruka.

```

var socket = io.connect("http://192.168.1.186:80");
socket.on('disconnect', function() {
    $("#connection-status").html("<span class='label label-important'>Disconnected!</span>");
});
socket.on('connect', function() {
    $("#connection-status").html("<span class='label label-success'>Connected!</span>");
});
socket.on('newKeyWord', function(obj) {
    $("#keyword-show").html(obj.word);
});
socket.on('noOfClients', function(obj) {
    $("#number-of-clients small").html(obj.number);
});
$("#track").click(function(){
    var word = $("#keyword").val();
    if(word.length > 0){

```

```

        socket.emit('launch', {"word":word});
    }
});
socket.on('tweets', function(data){
    var holder = document.getElementById('tweets');
    var imageURL = 'http://s.twimg.com/a/1281473923/images/default_profile_4_bigger.png';
    $.each(data, function(index, value) {
        if(value.user.profile_image_url != undefined){
            imageURL = value.user.profile_image_url;
        }
        noTw++;
        var contentString = "<div class='well'><table><tr><td><img class='pull-right' src='"+imageURL+"\"
/><h4>"+value.user.name+"Vh4>Vtd>Vtr><tr><td><p>"+value.text+"Vp>Vtd>Vtr><tr><td><h6><small>"+val
ue.created_at+"Vsmall>Vh6>Vtd>Vtr>Vtable>Vdiv><hr/>";
        $('#number-of-tweets small').text(noTw);
        $('#tweets ul').prepend(contentString);
        if(value.geo){
            var myLatLng = new google.maps.LatLng(value.geo.coordinates[0],value.geo.coordinates[1]);
            var twitter = 'images/tweet.png';
            var marker = new google.maps.Marker({
                position: myLatLng,
                map: myMap,
                animation: google.maps.Animation.DROP,
                icon: twitter,
                title: value.user.name
            });
            addInfoWindow(marker, contentString);
        }
    });
});

```

Konekcija se ostvaruje pomoću funkcije `io.connect`. Ovoj funkciji prosleđuje se adresa servera i port na kojem server osluškuje za zahtjeve klijenata. Nakon uspostavljenе konekcije kreira se objekat `socket` preko kojeg se odvija komunikacija. Socket objekat posjeduje on metod kome se prosleđuje odgovarajući parametar u zavisnosti od željenog rezultata. Na primjer, `socket.on("connect")` izvršiće se kada server emituje poruku sa zaglavljem `connect`.

Takođe moguća je i komunikacija sa serverom tako što se koristi metod `emit` objekta `socket`. Ovaj metod primijenjen je prilikom konekcije novog klijenta na server. Naime, kada se novi korisnik konektuje na aplikaciju on emituje poruku ka serveru i govori mu da je novi klijent konektovan. Server obrađuje ovu poruku, inkrementira broj klijenata i ažurni broj prosleđuje svim već konektovanim korisnicima. Isto tako kada jedan korisnik promijeni ključnu riječ on šalje serveru poruku. Server obrađuje poruku te propagira poruku svim već konektovanim korisnicima.

```
//Iniciranje promjene ključne riječi od strane korisnika.
```

```
$("#track").click(function(){  
  var word = $("#keyword").val();  
  if(word.length > 0){  
    socket.emit('launch', {"word":word});  
  }  
});
```

```
//Prihvatanje nove ključne riječi
```

```
socket.on('newKeyWord', function(obj) {  
  $("#keyword-show").html(obj.word);  
});
```

Izvorni kod serverskog dijela aplikacije

```
var app = require('http').createServer(handler)  
  , io = require('socket.io').listen(app)  
  , fs = require('fs')  
  , util = require('util')  
  , http = require('https')  
  , interval  
  , keyword = "android"  
  , twUser = "korisnik"  
  , twPass = "lozinka";  
app.listen(80, '192.168.1.186');  
  
function get_tweets() {  
  var options = {host: "stream.twitter.com", port:443, path:"/1/statuses/filter.json?track="+keyword,  
method:"GET", auth:twUser+" ":"+twPass }  
  console.log("In get tweets: "+keyword);  
  var request = http.request(options, function(res){  
    var body = "";  
    res.on("data", function(data) {  
      if(data.length > 5)  
        body += data + ",";  
    });  
  
    res.on("end", function() {  
      console.log("on ent trigger activated!!! Data length: "+body.length);  
      if(body.length > 68){  
        body = "["+body+"}";  
        body = body.replace(/,/g, ",");  
        var tweets = JSON.parse(body);  
        io.sockets.emit("tweets", tweets);  
      }  
    });  
  });  
  request.end();  
}  
  
function handler (req, res) {
```

```

fs.readFile(__dirname + '/index.html',
function (err, data) {
  if (err) {
    res.writeHead(500);
    return res.end('Error loading index.html');
  }

  res.writeHead(200);
  res.end(data);
});
}

io.sockets.on('connection', function (socket) {
  console.log('Client connected!');
  io.sockets.emit("noOfClients", {"number": io.sockets.clients().length});
  socket.emit('newKeyWord', {"word": keyword});
  socket.on('launch', function (obj){
    keyword = encodeURIComponent(obj.word);
    io.sockets.emit("newKeyWord", {"word": obj.word});
    clearInterval(interval);
  });
  interval = setInterval(get_tweets, 90000);
});

io.sockets.on('disconnect', function(){
  io.sockets.emit('Client disconnected!');
  io.sockets.emit("noOfClients", {"number": io.sockets.clients().length});
});

```

8.1.8 Serverski dio aplikacije

Serverski dio aplikacije realizovan je pomoću node.js JavaScript biblioteke koja omogućava realizaciju WebSocket servera. Na početku fajda definisane su globalne promjenljive te uključene sve potrebne klase. Nakon toga kreiran je server i zadato mu je da osluškuje dati port na datoj adresi. Server prihvata konekcije korisnika na datom portu i distribuira odgoarajuće poruke.

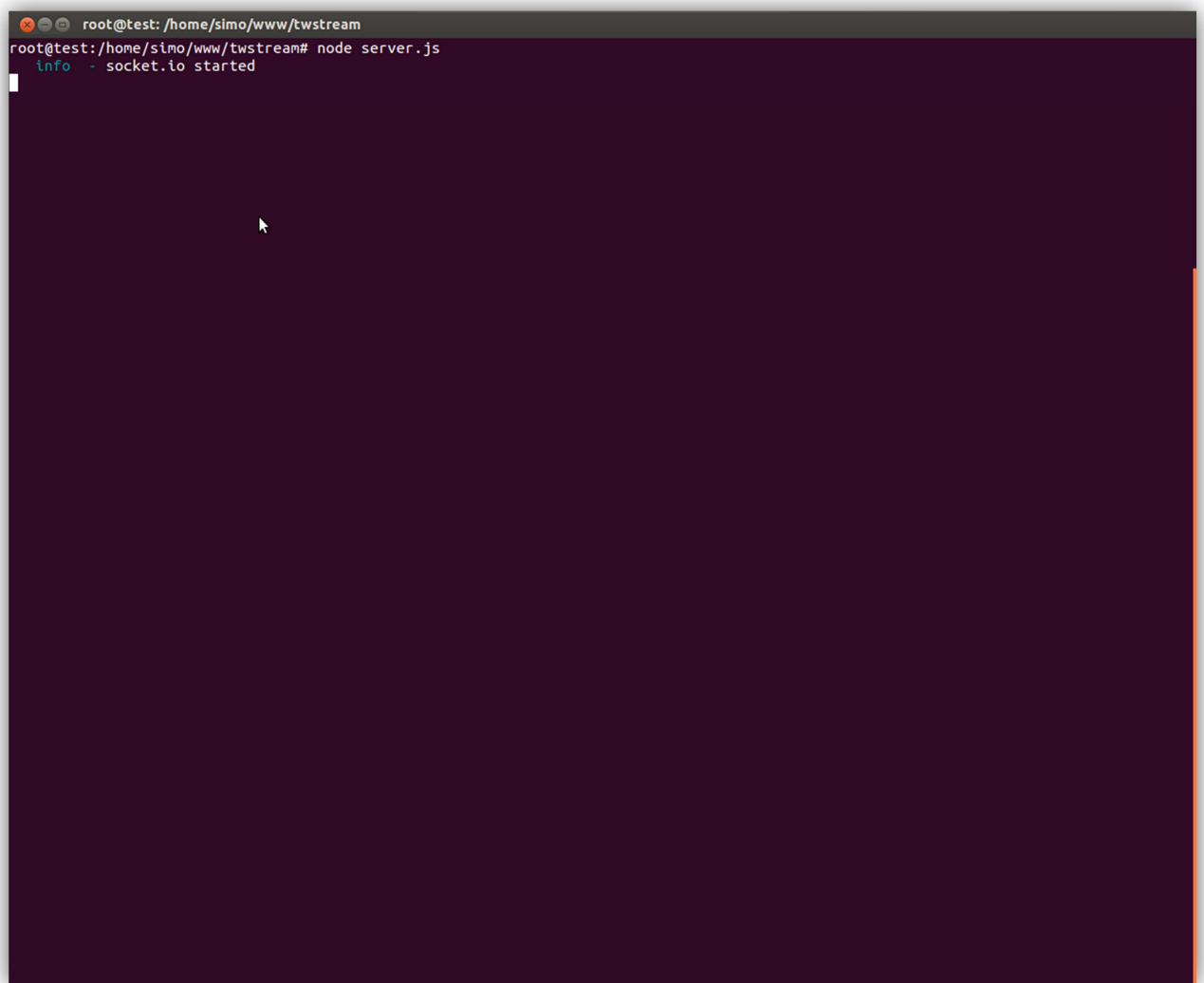
Socijalna mreža twitter pruža javni API pomoću kojeg bilo ko ko posjeduje nalog na toj mreži može da se konektuje te da pošalje zahtjev za željenim porukama tj. tvitovima. Serveri ove mreže obrađuju klijentov zahtjev te prosleđuju odgovor u vidu JSON-a klijentu. Problem je to što se ova komunikacija odvija preko HTTP protokola te nije moguća komunikacija u realnom vremenu. Zbog ove činjenice serverska aplikacija se svaki određeni interval vremena konektuje, prosleđuje zahtjev te obrađuje dobijenu listu tvitova. Ovo je neizbježno jer ne postoji drugačiji način jer naprosto to ova mreža ne omogućava.

//Funkcija koja se konektuje na twitterov server, preuzima listu tvitova te distribuira iste svim konektovanim korisnicima

```
function get_tweets() {
  var options = {host: "stream.twitter.com", port:443, path:"/1/statuses/filter.json?track="+keyword,
method:"GET", auth:twUser+" ":"+twPass }
  console.log("In get tweets: "+keyword);
  var request = http.request(options, function(res){
    var body = "";
    res.on("data", function(data) {
      if(data.length > 5)
        body += data + ",";
    });

    res.on("end", function() {
      console.log("on ent trigger activated!!! Data length: "+body.length);
      if(body.length > 68){
        body = "["+body+"{}]";
        body = body.replace(/,/g, ",");
        var tweets = JSON.parse(body);
        io.sockets.emit("tweets", tweets);
      }
    });
  });
  request.end();
}
```

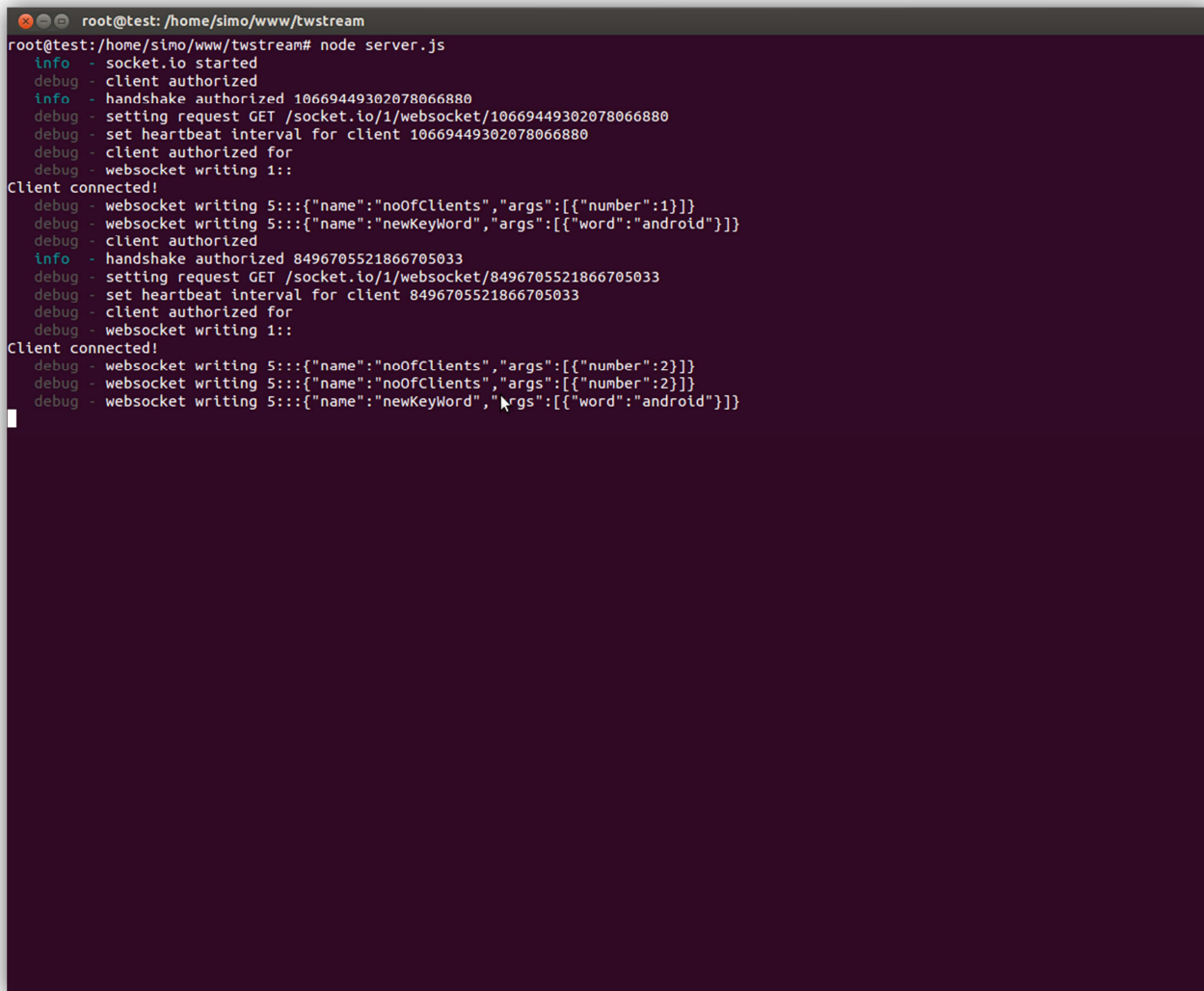
Server razmatran u ovom radu je implementiran na linux platformi iako ovo nije obavezno jer node.js biblioteka je platformski nezavisna te može biti implementirana i na Windows ili MacOS platformu. Server se pokreće pomoću node programa koji prethodno valja instalirati. Komanda za pokretanje servera je node <naziv_fajla_servera>. U našen slučaju to je node server.js. Valja napomenuti da ukoliko server osluškuje na portu koji je ispod broja 1000 onda mora da se pokreće sa privilegijama super korisnika, najčešće pomoću naredbe sudo tako da bi smo server pozivali sa sudo nano server.js ukoliko bismo recimo koristili port 300. Prikaz pokrenutog servera dat je na slici 14.

A terminal window with a dark purple background. The title bar at the top shows three window control icons (red, yellow, green) followed by the text "root@test: /home/simo/www/twstream". The terminal content shows the command "root@test:/home/simo/www/twstream# node server.js" and its output "info - socket.io started". A white cursor is positioned on the line following the output. The terminal window has a thin orange border on the right side.

```
root@test: /home/simo/www/twstream
root@test:/home/simo/www/twstream# node server.js
info - socket.io started
```

Slika 14 - Pokrenuti WebSocket server

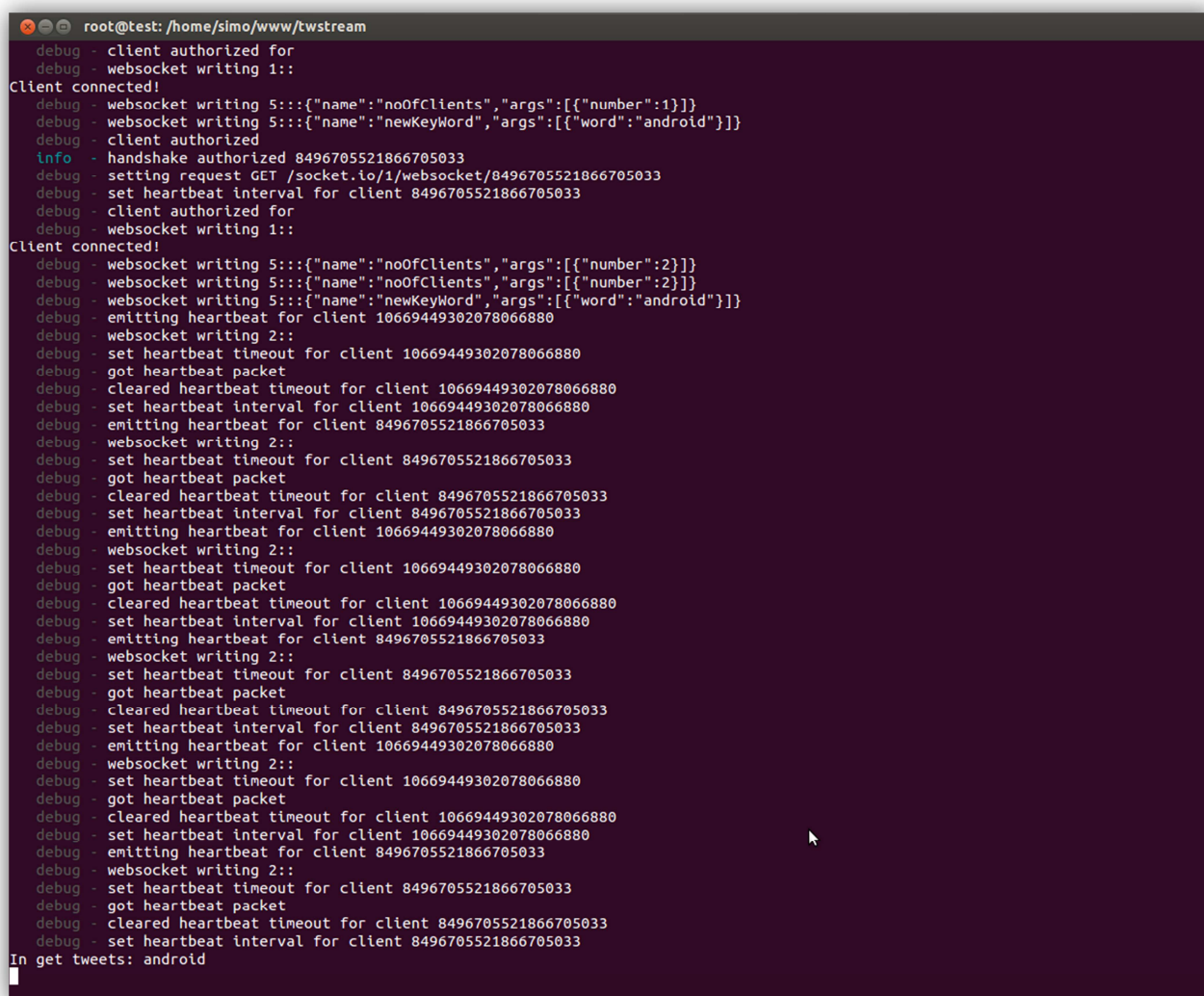
Prilikom konekcije novih korisnika na server isti će ispisati odgovarajuće podatke na konzoli tako da administrator može da prati rad servera samo kroz konzolu slika 15.



```
root@test: /home/simo/www/twstream
root@test: /home/simo/www/twstream# node server.js
info - socket.io started
debug - client authorized
info - handshake authorized 10669449302078066880
debug - setting request GET /socket.io/1/websocket/10669449302078066880
debug - set heartbeat interval for client 10669449302078066880
debug - client authorized for
debug - websocket writing 1::
Client connected!
debug - websocket writing 5::{"name":"noOfClients","args":[{"number":1}]}
debug - websocket writing 5::{"name":"newKeyword","args":[{"word":"android"}]}
debug - client authorized
info - handshake authorized 8496705521866705033
debug - setting request GET /socket.io/1/websocket/8496705521866705033
debug - set heartbeat interval for client 8496705521866705033
debug - client authorized for
debug - websocket writing 1::
Client connected!
debug - websocket writing 5::{"name":"noOfClients","args":[{"number":2}]}
debug - websocket writing 5::{"name":"noOfClients","args":[{"number":2}]}
debug - websocket writing 5::{"name":"newKeyword","args":[{"word":"android"}]}
```

Slika 15 - Ispis poruka prilikom konekcije novog klijenta.

Na konzoli će biti ispisana odgovarajuća poruka svaki put kada aplikacija pokuša da uspostavi komunikaciju sa serverima twitter socijalne mreže te preuzme listu poruka od njih(Slika 16, 17). Ukoliko se dogodi greška prilikom povezivanja na konzoli će se ispisati odgovarajući podatak o toj grešci. Problem može da nastane ukoliko se loše formira zahtjev koji se šalje twitter serveru ili aplikacija iskusi određeni interni problem. Tipična greška koja može da nastane jeste prekid rada aplikacije ukoliko se preuzme loše formiran JSON zapis. Interni JSON parser neće uspeti da parsira zahtjev i na konzoli će biti prikazan tekst te greške.



```
root@test: /home/simo/www/twstream
debug - client authorized for
debug - websocket writing 1::
Client connected!
debug - websocket writing 5::{"name":"noOfClients","args":[{"number":1}]}
debug - websocket writing 5::{"name":"newKeyWord","args":[{"word":"android"}]}
debug - client authorized
info - handshake authorized 8496705521866705033
debug - setting request GET /socket.io/1/websocket/8496705521866705033
debug - set heartbeat interval for client 8496705521866705033
debug - client authorized for
debug - websocket writing 1::
Client connected!
debug - websocket writing 5::{"name":"noOfClients","args":[{"number":2}]}
debug - websocket writing 5::{"name":"noOfClients","args":[{"number":2}]}
debug - websocket writing 5::{"name":"newKeyWord","args":[{"word":"android"}]}
debug - emitting heartbeat for client 10669449302078066880
debug - websocket writing 2::
debug - set heartbeat timeout for client 10669449302078066880
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 10669449302078066880
debug - set heartbeat interval for client 10669449302078066880
debug - emitting heartbeat for client 8496705521866705033
debug - websocket writing 2::
debug - set heartbeat timeout for client 8496705521866705033
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 8496705521866705033
debug - set heartbeat interval for client 8496705521866705033
debug - emitting heartbeat for client 10669449302078066880
debug - websocket writing 2::
debug - set heartbeat timeout for client 10669449302078066880
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 10669449302078066880
debug - set heartbeat interval for client 10669449302078066880
debug - emitting heartbeat for client 8496705521866705033
debug - websocket writing 2::
debug - set heartbeat timeout for client 8496705521866705033
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 8496705521866705033
debug - set heartbeat interval for client 8496705521866705033
debug - emitting heartbeat for client 10669449302078066880
debug - websocket writing 2::
debug - set heartbeat timeout for client 10669449302078066880
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 10669449302078066880
debug - set heartbeat interval for client 10669449302078066880
debug - emitting heartbeat for client 8496705521866705033
debug - websocket writing 2::
debug - set heartbeat timeout for client 8496705521866705033
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 8496705521866705033
debug - set heartbeat interval for client 8496705521866705033
In get tweets: android
```

Slika 16 - Poziv funkcije za preuzimanje tvitova.

```
root@test:/home/simo/www/twstream
isted count":0,"profile_sidebar_border_color":"CODEEED","statuses_count":126,"time_zone":null,"created_at":"Tue Sep 13 05:48:43 +0000 2011","followers_count":0,"profile_image_url":"http://a0.twimg.com/sticky/default_profile_images/default_profile_0_normal.png","default_profile":true,"verified":false,"profile_use_background_image":true,"protected":false,"url":null,"id_str":"372666623","is_translator":false,"profile_text_color":"333333","profile_image_url_https":"https://s10.twimg.com/sticky/default_profile_images/default_profile_0_normal.png","name":"Emil Ahmadvov","profile_background_image_url":"http://a0.twimg.com/images/themes/theme1/bg.png","id":372666623,"notifications":null,"utc_offset":null,"profile_link_color":"0084B4","location":null},"retweeted":false,"id":219790880054325250,"in_reply_to_status_id":null,"entities":{"hashtags":[],"user_mentions":[{"screen_name":"tuxinfo","indices":[3,11],name":"tuxinfo","id_str":"16298926","id":16298926}],urls":[{"indices":[64,85],"display_url":"play.google.com/store/apps/det..."},"url":"https://t.co/eTPFP9fm"],"expanded_url":"https://play.google.com/store/apps/details?id=org.videolan.vlc.beta&v=neon"}},"text":"RT @tuxinfo: Ultimo momento VLC beta , disponible para Android! https://t.co/eTPFP9fm","place":null,"truncated":false,"possibly_sensitive_editable":true,"retweet_count":0,"entites":{"hashtags":["user_mentions"]:"urls":[{"indices":["51,72"],display_url":"play.google.com/store/apps/det..."},"url":"https://t.co/eTPFP9fm"],"expanded_url":"https://play.google.com/store/apps/details?id=org.videolan.vlc.beta&v=neon}}},"text":"Ultimo momento ! VLC beta, disponible para Android! https://t.co/eTPFP9fm","place":null,"truncated":false,"possibly_sensitive_editable":true,"favorite_count":false,"possibly_sensitive":false,"in_reply_to_screen_name":null,"source":{"a href="http://www.tweetdeck.com/" rel="nofollow">TweetDeck</a>","coordinates":null,"geo":null,"in_reply_to_status_id_str":null,"created_at":"Mon Jul 02 10:54:10 +0000 2012","retweet_count":0,"in_reply_to_user_id_str":null,"id_str":"219745775759011840","in_reply_to_user_id":null,"contributors":null,"user":{"default_profile_image":false,"lang":"es","screen_name":"tuxinfo","contributors_enabled":false,"geo_enabled":false,"favourites_count":0,"friends_count":2,"profile_background_color":"","CODEEED","follow_request_sent":null,"profile_background_tile":false,"following":null,"profile_picture_image_urls":"https://s10.twimg.com/images/themes/theme1/bg.png","profile_sidebar_fill_color":"DDEEF6","description":"Twitter oficial de la primer revista de software libre y Linux gratuita en PDF de Argentina.",,"show_all_inline_media":false,"listed_count":72,"profile_sidebar_border_color":"CODEEED","statuses_count":1339,"time zone":"Buenos Aires","created at":"Mon Sep 15 17:55:53 +0000 2008","followers count":834,"profile_image url":"http://a0.twimg.com/profile_images/60080573/tuxinfo_banner_normal.jpgg","default profile true","verified":false,"profile use background image":true,"protected":false,"url":"http://www.tuxinfo.com.ar","id str":"16298926","is translator":false,"profile text color":"333333","profile image url https":"https://s10.twimg.com/profile_images/60080573/tuxinfo_banner_normal.jpgg","name":"tuxinfo","profile background image url":"http://a0.twimg.com/images/themes/theme1/bg.png","id":16298926,"notifications":null,"utc offset":-10800,"profile link color":"0084B4","location":"","retweeted":false,"id":219745775759011840,"in_reply_to status id":null,"favorited":false,"possibly sensitive":false,"in_reply_to screen name":null,"source":{"a href="http://itunes.apple.com/us/app/twitter/id409789998?mt=12" rel="nofollow">Twitter for Mac</a>","coordinates":null,"geo":null,"in_reply_to status id str":null,"created at":"Mon Jul 02 13:53:23 +0000 2012","retweet count":0,"in_reply_to user id str":null,"id str":"219790879827832834","in reply to user id":null,"contributors":null,"user":{"default profile image":false,"lang":"es","screen name":"gomezhub","contributors enabled":false,"geo enabled":true,"favourites count":1,"friends count":67,"profile background color":"CODEED","follow request sent":null,"profile picture image url":true,"following":null,"profile background image url https":"https://s10.twimg.com/profile background image s/196122598/20090803Internet map collage.jpgg","profile sidebar fill color":"DDEEF6","description":"Chief Technology Officer de Gomez Hub Startup","show all inline media":true,"listed count":0,"profile sidebar border color":"CODEED","statuses count":1307,"time zone":"Mountain Time (US & Canada)","created at":"Sat Jan 22 22:56:34 +0000 2011","followers count":59,"profile image url":"http://a0.twimg.com/profile_images/2248879542/radiationAV_normal.png","default profile image":false,"verified":false,"profile use background image":true,"protected":false,"url":"http://www.gomezhub.mx","id str":"241700387","is translator":false,"profile text color":"333333","profile image url https":"https://s10.twimg.com/profile_images/2248879542/radiationAV_normal.png","name":"Gomez Tech Talk Hub","profile background image url":"http://a0.twimg.com/profile background images/196122598/20090803Internet map collage.jpgg","id":241700387,"notifications":null,"utc offset":-25200,"profile link color":"0084B4","location":"public class Geek {}"},"retweeted":false,"id":219790879827832830,"in_reply_to status id":null,"entities":{"hashtags":[{"text":"Narenji","indices":["67,75]]","user mentions [{"screen_name":"narenji","indices":["46,66]"}]},"display url":"narenji.r/download-vlc-a-.."},"url":"http://t.co/J90UwhSc","expanded url":"http://narenji.r/download-vlc-android"}},{"text":"[عنوان لا ی درج کیلئے اپنا نام لکھیں]" ,"id":390090HSC "Narenji","place":null,"truncated":false,"possibly_sensitive_editable":true,"favorited":false,"possibly_sensitive":false,"in_reply_to_screen_name":null,"source":{"a href="http://ifttt.com/" rel="nofollow">IFTTT</a>","coordinates":null,"geo":null,"in_reply_to_status_id_str":null,"created at":"Mon Jul 02 13:53:25 +0000 2012","retweet count":0,"in_reply_to_user_id_str":null,"id_str":"219790887193022464","in_reply_to_user_id":null,"contributors":null,"user":{"default profile image":false,"lang":"en","screen name":"avahead","contributors_enabled":false,"geo_enabled":true,"favourites count":4,"friends count":79,"profile background color":"B2DFDA","follow_request_sent":null,"profile_background_tile":false,"following":null,"profile background image url https":"https://s10.twimg.com/images/themes/theme13/bg.gif","profile_sidebar_fill_color":"ffffff","description":"Birth in 18th June 1980 in razye firooz hospital of kerman.\nWeb Developer,\n\nTechwriter,\n\nPerfect film lover!\n\nuniversity: SBU Kerman","show_all_inline_media":false,"listed count":5,"profile_sidebar_border color":"eeeeee","statuses count":4679,"time zone":"Tehran","created at":"Thu Feb 28 14:31:54 +0000 2008","followers count":323,"profile image url":"http://a0.twimg.com/profile_images/57619102/my_logo_ptco_normal.jpgg","default profile image":false,"verified":false,"profile
```

Slika 17 - Ispisivanje poruka prilikom preuzimanja tvitova.

Omogućeno je da aplikacija ispisuje poruku kada se započne procedura promjene ključne riječi ili fraze (Slika 18). Ova poruka sadrži aktuelnu riječ koje tj. frazu koje postaje aktivna te JSON zapis koji se šalje klijentima. Na ovaj način administratoru je omogućen uvid u sveku promjenu te može uočiti ukoliko neki korisnik pokušava na ovaj način da izvrši neki napad putem malicioznog koda.

```

root@test: /home/simo/www/twstream
ed":false,"possibly_sensitive":false,"in_reply_to_screen_name":null,"source":{"a href="http://www.tweetdeck.com" rel="nofollow">TwitterDeck</a>,"coordinates":null,"geo":null,"in_reply_to_status_id_str":null,"created_at":"Mon Jul 02 10:54:10 +0000 2012","retweet_count":0,"in_reply_to_user_id_str":null,"id_str":"21974577575901840","in_reply_to_user_id":null,"contributors":null,"user":{"default_profile_image":false,"lang":"es","screen_name":"tuxinfo","contributors_enabled":false,"geo_enabled":false,"favourites_count":0,"friends_count":2,"profile_background_color":"C0DEED","follow_request_sent":null,"profile_background_image_url_https":"https://si0.twimg.com/images/themes/theme1/bg.png","profile_sidebar_fill_color":"DDEEF6","description":"Twitter oficial de la primer revista de software libre y Linux gratuita en PDF de Argentina. ","show_all_inline_media":false,"listed_count":72,"profile_sidebar_border_color":"C0DEED","statuses_count":1339,"time_zone":"Buenos Aires","created_at":"Mon Sep 15 17:55:53 +0000 2008","followers_count":834,"profile_image_url":"http://a0.twimg.com/profile_images/60080573/tuxinfo_banner_normal.jpg","default_profile_image":true,"verified":false,"profile_use_background_image":true,"protected":false,"url":"http://www.tuxinfo.com.ar","id_str":"16298926","is_translator":false,"profile_text_color":"333333","profile_image_url_https":"https://si0.twimg.com/profile_images/60080573/tuxinfo_banner_normal.jpg","name":"tuxinfo","profile_background_image_url":"http://a0.twimg.com/images/themes/theme1/bg.png","id":"16298926","notifications":null,"utc_offset":-10800,"profile_link_color":"0084B4","location":"","retweeted":false,"id":21974577575901840,"in_reply_to_status_id":null,"favorited":false,"possibly_sensitive":false,"in_reply_to_screen_name":null,"source":{"a href="http://itunes.apple.com/us/app/twitter/1409789998?mt=12" rel="nofollow">Twitter for Mac</a>,"coordinates":null,"geo":null,"in_reply_to_status_id_str":null,"created_at":"Mon Jul 02 13:53:23 +0000 2012","retweet_count":0,"in_reply_to_user_id_str":null,"id_str":"219790879827832834","in_reply_to_user_id":null,"contributors":null,"user":{"default_profile_image":false,"lang":"es","screen_name":"gomezhub","contributors_enabled":false,"geo_enabled":true,"favourites_count":1,"friends_count":67,"profile_background_color":"C0DEED","follow_request_sent":null,"profile_background_image_url_https":"https://si0.twimg.com/profile_images/196122598/20090803Internet_map_collage.jpg","profile_sidebar_fill_color":"DDEEF6","description":"Chief Technology Officer de Gomez Hub Startup","show_all_inline_media":true,"listed_count":0,"profile_sidebar_border_color":"C0DEED","statuses_count":1307,"time_zone":"Mountain Time (US & Canada)","created_at":"Sat Jan 22 22:56:34 +0000 2011","followers_count":59,"profile_image_url":"http://a0.twimg.com/profile_images/2248879542/radiationAV_normal.png","default_profile":false,"verified":false,"profile_use_background_image":true,"protected":false,"url":"http://www.gomezhub.mx","id_str":"241700387","is_translator":false,"profile_text_color":"333333","profile_image_url_https":"https://si0.twimg.com/profile_images/2248879542/radiationAV_normal.png","name":"Gomez Tech Talk Hub","profile_background_image_url_https":"http://a0.twimg.com/profile_images/196122598/20090803Internet_map_collage.jpg","id":"241700387","notifications":null,"utc_offset":-25200,"profile_link_color":"0084B4","location":"public class Geek { }","retweeted":false,"id":219790879827832830,"in_reply_to_status_id":null,"entities":{"hashtags":[{"text":"Narenji","indices":[67,75]}],"user_mentions":[],"urls":[{"indices":[46,66]}]},"display_url":"narenji.ir/download-vlc-a","url":"http://ft.co/3900Whsc","expanded_url":"http://narenji.ir/download-vlc-android"},"text":"ایورس ای اب هیو لای و شریکوب ایزی ازی ا مین http://ft.co/3900Whsc #Narenji","place":null,"truncated":false,"possibly_sensitive":false,"favorited":false,"possibly_sensitive":false,"in_reply_to_screen_name":null,"source":{"a href="http://ifttt.com" rel="nofollow">IFTTT</a>,"coordinates":null,"geo":null,"in_reply_to_status_id_str":null,"created_at":"Mon Jul 02 13:53:25 +0000 2012","retweet_count":0,"in_reply_to_user_id_str":null,"id_str":"219790879193022460","in_reply_to_user_id":null,"contributors":null,"user":{"default_profile_image":false,"lang":"en","screen_name":"avahedi","contributors_enabled":false,"geo_enabled":true,"favourites_count":4,"friends_count":79,"profile_background_color":"B2DFDA","follow_request_sent":null,"profile_background_image_url_https":"https://si0.twimg.com/images/themes/theme13/bg.gif","profile_sidebar_fill_color":"ffffff","description":"Birth in 18th June 1980 in razye firooz hospital of kerman.\nWeb Developer ,\nTchewriter ,\nPerfect film lover!\n\nnuniversity : SBU Kerman","show_all_inline_media":false,"listed_count":5,"profile_sidebar_border_color":"eeeeec","statuses_count":4679,"time_zone":"Tehran","created_at":"Thu Feb 28 14:31:54 +0000 2008","followers_count":323,"profile_image_url_https":"http://a0.twimg.com/profile_images/5761912/2/my_logo_pic6_normal.jpg","default_profile":false,"verified":false,"profile_use_background_image":true,"protected":false,"url":"http://www.narenji.ir","id_str":"14054823","is_translator":true,"profile_text_color":"333333","profile_image_url_https":"https://si0.twimg.com/profile_images/57619102/my_logo_pic6_normal.jpg","name":"abbas vahedi","profile_background_image_url_https":"http://a0.twimg.com/images/themes/theme13/bg.gif","id":"14054823","notifications":null,"utc_offset":12600,"profile_link_color":"93A644","location":"Kerman,Iran"},"retweeted":false,"id":219790879193022460,"in_reply_to_status_id":null,{}]]}

debug - emitting heartbeat for client 8496705521866705033
debug - websocket writing 2:
debug - set heartbeat timeout for client 8496705521866705033
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 8496705521866705033
debug - set heartbeat interval for client 8496705521866705033
debug - websocket writing 5::{"name":"newKeyword","args":[{"word":"windows"]}}
debug - websocket writing 5::{"name":"newKeyword","args":[{"word":"windows"]}}

```

Slika 18 - Ispis poruka prilikom promjene ključne riječi

8.2 Chat aplikacija.

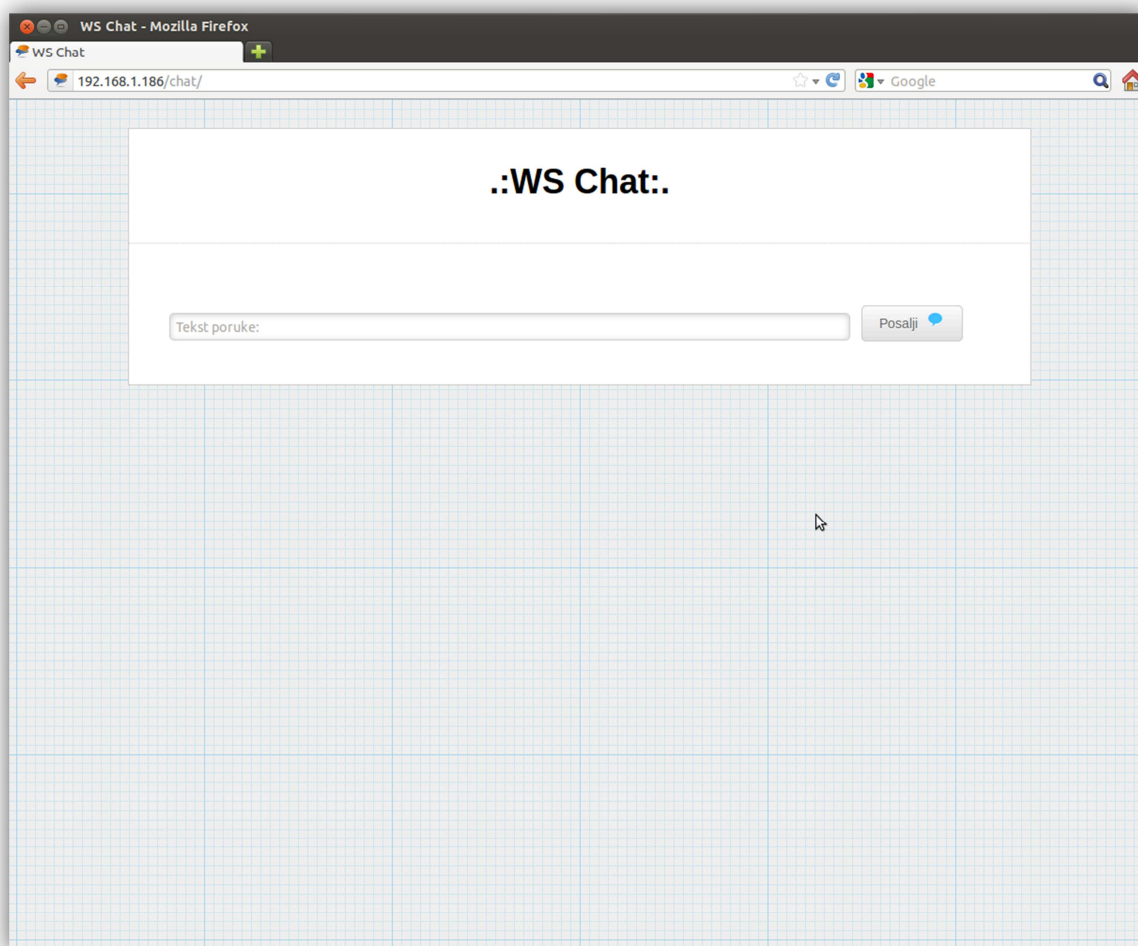
8.2.1 Opis problematike

IT inženjeri koji se bave razvijanjem web aplikacija nailaze na ogromne probleme kad god se javi potreba za implementiranjem neke od funkcionalnosti koje nosi on line časkanje. Naime, problem koji ovde nastaje jeste taj što aplikacija ima veoma visok stepen dinamičnosti, korisnici non stop pišu poruke jedni drugima te aplikacija mora biti u stanju da korisnicima pruži osjećaj kao da uživo razgovaraju što znači da nikakva kašnjenja i zakašnjenja nisu dozvoljena. Ovo je izrazito teško, gotovo nemoguće postići oslanjajući se samo na HTTP protokol i tradicionalne tehnologije koje se koriste u web programiranju. Problem nastaje u činjenici da svaki konektovani klijent mora da po isteku određenog vremenskog intervala pošalje zahtjev serveru i upita ga da li ima nekih novih poruka za njega te ukoliko ima da dobije listing istih te da ih ispiše klijentu. Ovakav pristup zahtijeva skladištenje poruka na serveru te generiše ogromnu količinu mrežnog saobraćaja. Podsjetimo se da aplikacija mora korisnicima da pruži osjećaj da razgovaraju u realnom vremenu te da ne postoji način da server, bez zahtjeva klijenta, istom pošalje listing njegovih poruka korišćenjem samo HTTP protokola. Proces chat aplikacije bez korišćenja dodatnih rešenja je sledeći. Korisnik pokreće aplikaciju te šalje zahtjev serveru za listu njegovih poruka, server odgovara na upućeni zahtjev te dostavlja klijentu njegov poruke. Klijentov web pretraživač učitava odgovor od servera te renderuje stranicu na osnovu doijenog html-a i tu se komunikacija završava. Korisnik sada vidi svoje poruke. Druga strana sa kojom korisnik komunicira šalje dodatne poruke. Pošto korisnici nisu povezani međusobne nego se sva komunikacija odvija preko servera poslate poruke se čuvaju na serveru i čekaju ponovni upit korisnika za njegovim porukama. Na ponovni upit korisnika server dostavlja ažuriranu listu njegovih poruka. Problem leži u činjenici da se ovi zahtjevi ponavljaju na desetine puta u toku samo jednog minuta jer je težnja da se ostvari doživljaj razgovora u realnom vremenu te je listu poruka potrebno što češće ažurirati. Korišćenje ovakvog metoda iziskuje osvježavanja stranice klijenta svaki put kada server odgovori ažurnom verzijom liste poruka. Da bi se ovo izbjeglo jedno rešenje je koristiti AJAX tehnologije tj. asihroni Javascript XML zahtjev. Pomoću AJAX-a klijent može da učita naovi sadržaj na stranicu bez potrebe da osvježava čitavu stranu. Međutim ni ovo rešenje nije dovoljno učinkovito jer i dalje postoji problem generisanja ogromne količine nepotrebnog mrežnog saobraćaja. Dodatno rešenje predstavlja upotreba takozvanoe commet tehnike opisane ranije u radu, međutim iako dosta učinkovitija od čistog AJAX-a ni ova tehnologije

ne predstavlja konačno rešenje. Konačno rešenje koje je do sada bilo aktuelno jeste korišćenje neke od tehnologije treće strane ali kao što je već rečeno ove tehnologije podrazumijevaju dodatne plug inove koje korisnik možda ne posjeduje.

8.2.2 Opis rešenja

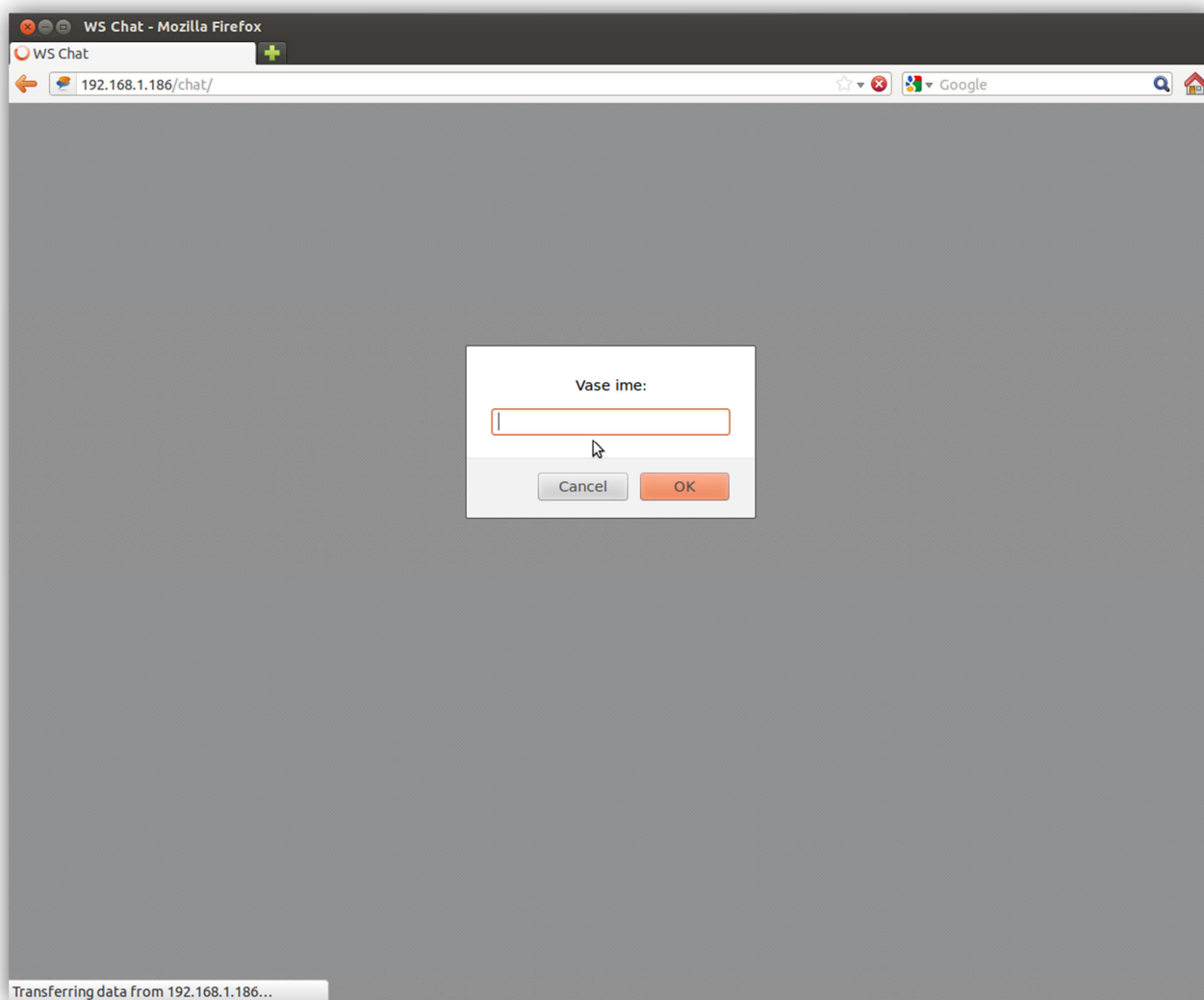
WebSocket nudi rešenje ovog problema. Korišćenjem WebSocket tehnologije aplikacija opisana u nastavku rada pruža mogućnost da klijent i server komuniciraju u full duplex modu tj. da server posjeduje mogućnost da korisniku isporuči ažuriranja bez potrebe klijenta da šalje zahtjev serveru. Ovo je omogućeno otvorenom konekcijom između klijenta i servera. Izgled početne strane aplikacije dat je na slici 19.



Slika 19 - Početna strana aplikacije

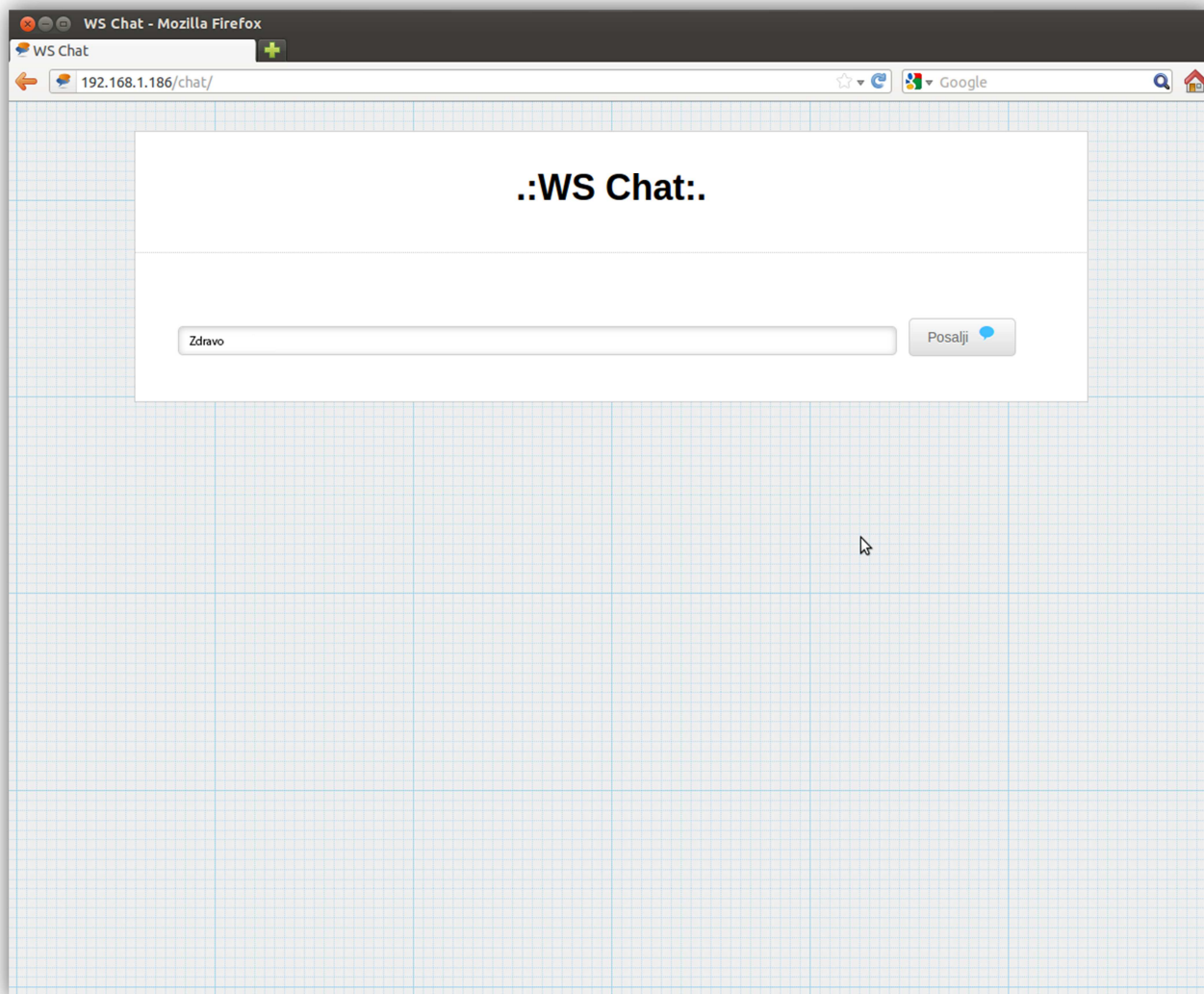
8.2.3 Upotreba

Kao i kod svih ostalih web aplikacija ideja je da se aplikaciji pristupa iz web pretraživača. Korisnik ne treba da ima potrebu snimanja dodatnog softvera da bi omogućio sve funkcionalnosti web aplikacije. Vođena tom idejom ova aplikacija za nesmetani rad od korisnika zahtijeva da jedino da posjeduje web pretraživač koji ima podršku za WebSocket protokol. Nakon što pokrene aplikaciju ukucavajući njenu adresu u URL polje svog pretraživača aplikacija će od korisnika tražiti da se predstavi tako što će unijeti svoje ime (slika 20).



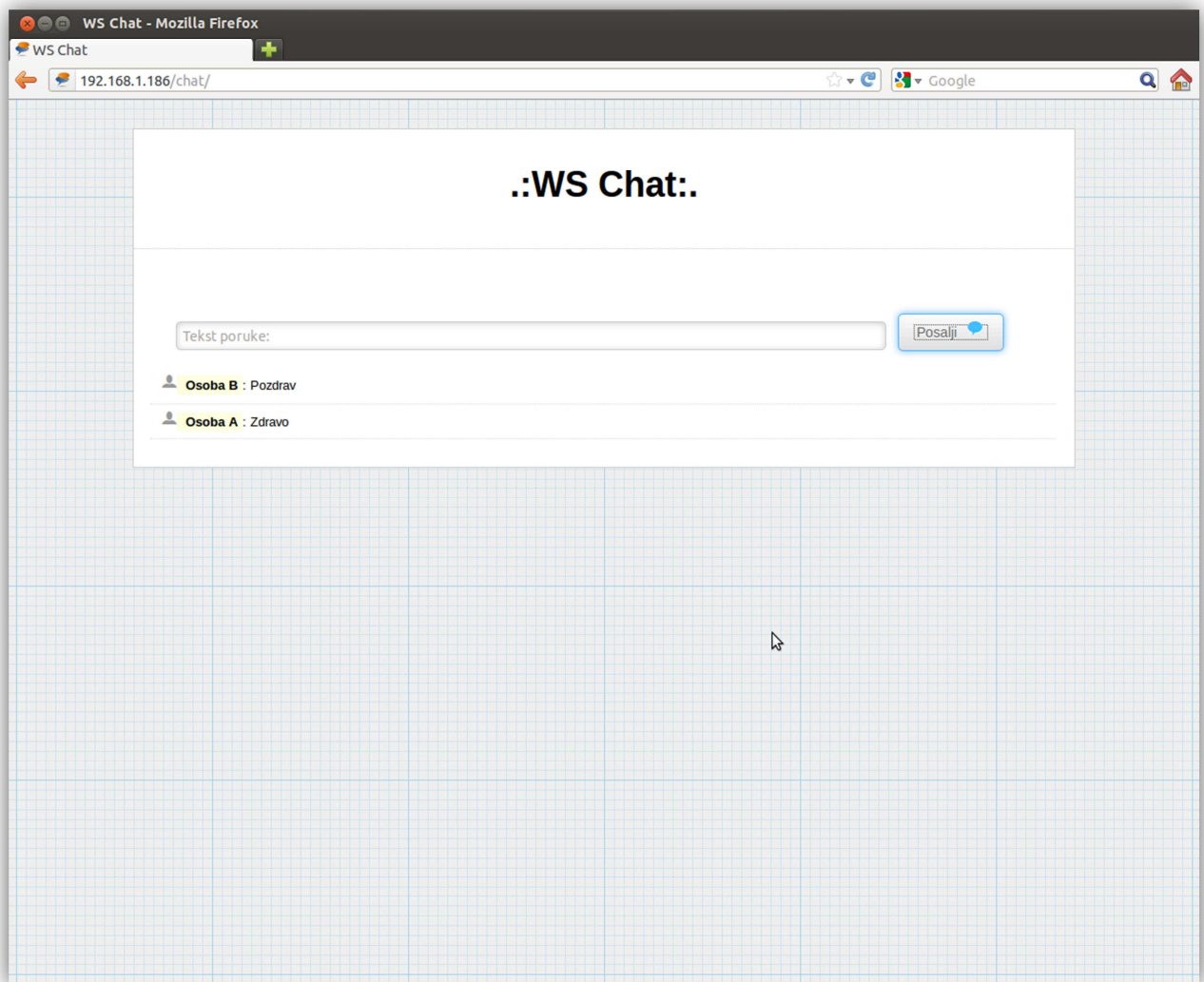
Slika 20 - Polje za upis imena tj. nadimka korisnika.

Nakon što se prdstavi korisniku će se prezentovati jednosavan interejs(Slika 21). Pored polja za unos poruke tu je dugme pomoću kojeg se poruka šalje te prostor na kome će biti ispisane sve poruke korisničke konverzacije.



Slika 21 - Unos teksta poruke.

Poruke koje korisnici šalju jedni drugima transportuje se do servera koji ih server dalje distribuira svim konektovanim klijentima. Sva ova komunikacija odvija se putem WebSocket protkola tako da je broj paketa sveden na minimum čime se smanjuje opterećenje same mreže. Prednosti koje WebSocket protokol donosi u odnosu na HTTP protokol su jako značajne imajući u vidu da ovakve aplikacije obično koristi veliki broj klijenata što u slučaju HTTP protokola podrazumijeva i ogromno opterećenje i mreže i web servera koji treba da opsluži ogroman broj zahtjeva.



Slika 22 - Komunikacija između dva korisnika.

8.2.4 Izvršni kod aplikacije

Izvršni kod klijentskog dijela aplikacije.

```
<!DOCTYPE html>
<html><head>
  <title>WS Chat</title>
  <meta charset="UTF-8">
  <meta name="description" content="Chat aplikacija" />
  <link rel="shortcut icon" href="images/icons/favicon.ico">
  <script type="text/javascript" src="http://code.jquery.com/jquery-1.7.2.min.js"></script>
  <script type="text/javascript" src="node_modules/socket.io/node_modules/socket.io-
client/dist/socket.io.js"></script>
  <script type="text/javascript">
    var ime = prompt("Vase ime:");
    var socket = io.connect("http://192.168.1.186:80");
    socket.on('private message', function(data){
      var holder = document.getElementById('holder');
      console.log(data);
      $('#holder').prepend(data.msg);
    });

    function emit(){
      var value = "<tr><td><span class='icon small gray\'" data-icon='\"u\"' title='\"User\"' style='\"display: inline-block;
\"];<span aria-hidden='\"true\">u</span></span></td><td><b>"+ime+"</b>: "
+htmlEscape(document.getElementById('abc').value)+"</td></tr>";
      $('#holder').prepend(value);
      socket.emit('private message', value);
      document.getElementById('abc').value = "";
      return false;
    }

    function checkKey()
    {
      if (window.event.keyCode == 13)
      {
        return emit();
      }
    }
  </script>
  <!--[if lt IE 9]><script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script><![endif]-->
  <script type="text/javascript" src="html_kickstart/js/prettify.js"></script>
  <script type="text/javascript" src="html_kickstart/js/kickstart.js"></script>
  <link rel="stylesheet" type="text/css" href="html_kickstart/css/kickstart.css" media="all" />
  <link rel="stylesheet" type="text/css" href="html_kickstart/style.css" media="all" />
</head>
<body>
  <div id="wrap" class="clearfix">
    <br/>
    <h2 class="center">.:WS Chat.:</h2>
    <hr/>
    <div class="col_12">
      <table class="striped" cellspacing="0" cellpadding="0">
        <thead>
          <tr></tr>
        </thead>
        <tbody id="holder"><tr></tr></tbody>

        <div class="col_12">
          <input placeholder="Tekst poruke:" onkeypress='return checkKey();' type="text" id="abc" name="abc"
class="col_10 medium">
          <button onclick="return emit();" class="medium">Posalji &nbsp; <span class="icon small blue" data-
icon="" style="display: inline-block; "></span></button>
        </div>
      </div><!-- END WRAP -->
  </div>
```

```
</body>
</html>
```

Zaglavlje aplikacije sadrži HTML5 doctype te sve potrebne meta podatke kao što su podatci o autoru te reference ka spoljnim JavaScript i css fajlovima. U body sekciji koda nalazi se kod koji služi za konekciju na server. Ovaj dio je realizovan pomoću JavaScript-a. Nakon ostvarene konekcije kreira se objekat socket pomoću kojeg se vrši sva WebSocket komunikacija između servera i klijenta. Ukoliko klijentov pretraživač ne podržava rad sa WebSocket protokolom klijent neće biti u mogućnosti da šalje ili prima chat poruke.

Slanje poruka serveru vrši se pomoću emit funkcije. Ova funkcija uzima tekst poruke, prosleđuje ga serveru a isti ga onda distribuira svim konektovanim korisnicima.

```
//Emit funkcija koja služi za slanje chat poruka
```

```
function emit(){
    var value = "<tr><td><span class='icon small gray' data-icon='u' title='User' style='display: inline-block; ' "><span aria-hidden='true'>u</span></span><b> "+ime+"</b>: "
    +htmlEscape(document.getElementById('abc').value)+"</td></tr>";
    $('#holder').prepend(value);
    socket.emit('private message', value);
    document.getElementById('abc').value = "";
    return false;
}
```

Klijent prima poruke od severa pomoću on metoda objekta socket. Ovaj metod neprestano osluškuje dolazne poruke te ukoliko se ista pojavi na nju primjeni odgovarajuću lgiku tj. u našem slučaju ispiše je na ekran tako da korisnik može da je pročita.

```
//Prijem poruka od servera
```

```
socket.on('private message', function(data){
    var holder = document.getElementById('holder');
    console.log(data);
    $('#holder').prepend(data.msg);
});
```

Serverksa strana ove aplikacije, baš kao što je slučaj i sa prethodnom, realizovana je pomoću node.js biblioteke i postavljena na linux platformu. Izvršni kod serverskog dijela je sledeći:

```
var app = require('http').createServer(handler)
, io = require('socket.io').listen(app)
, fs = require('fs');

app.listen(80, '192.168.1.186');
```

```

function handler (req, res) {
  fs.readFile(__dirname + '/index.html',
    function (err, data) {
      if (err) {
        res.writeHead(500);
        return res.end('Error loading index.html');
      }

      res.writeHead(200);
      res.end(data);
    });
}

io.sockets.on('connection', function (socket) {
  console.log('Client connected!');

  socket.on('private message', function (val) {
    socket.broadcast.emit('private message', { msg: val });
    console.log('Private message received: ', val);
  });
});

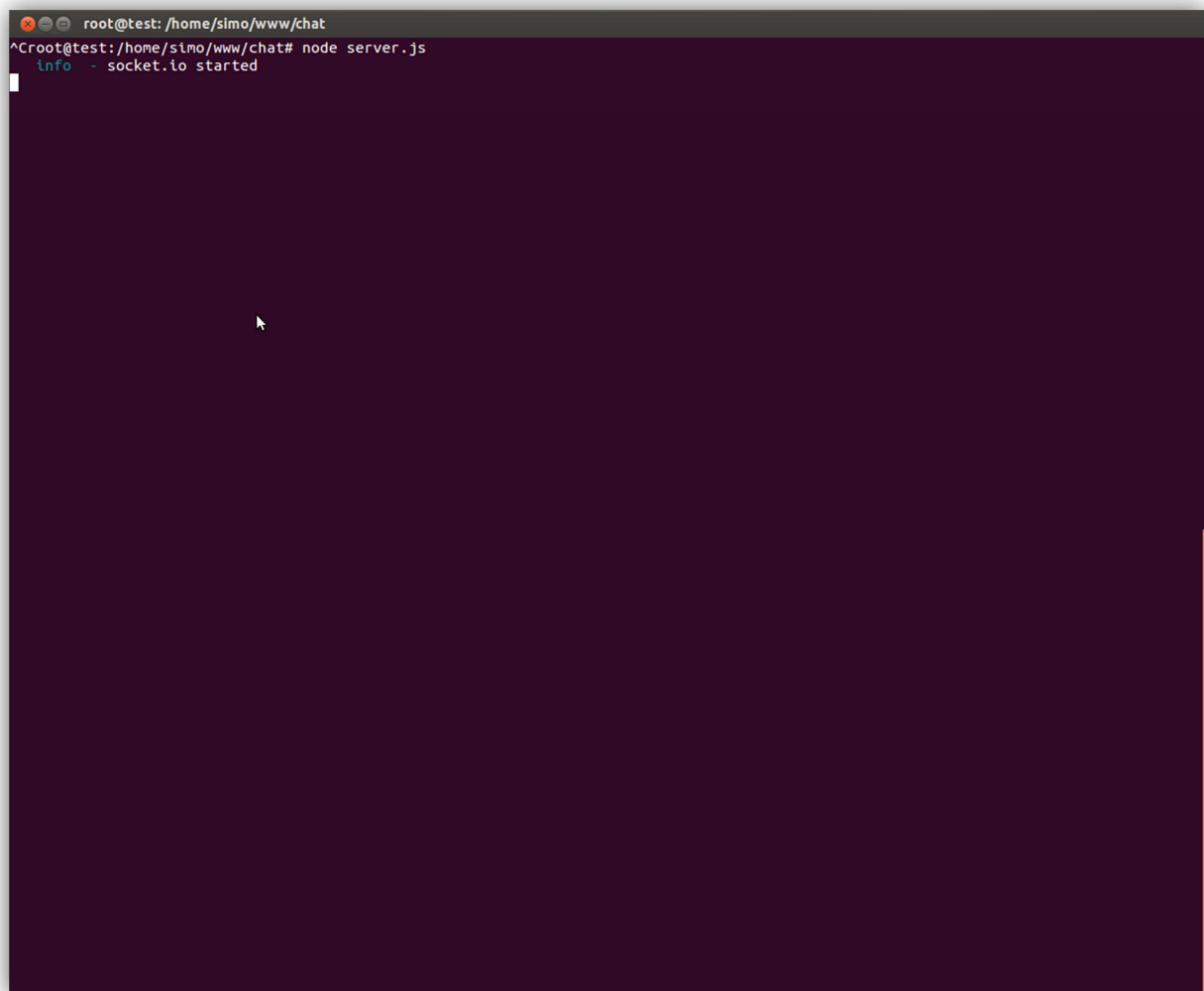
io.sockets.on('disconnect', function(){
  io.sockets.emit('Client disconnected!');
});

```

Na početku fajla se uključuju sve potrebene klase i kreira se server koji osluškuje na zahtjeve na datom portu datog servera. Server reaguje na konekciju novih klijenata, prekid veze sa klijentom te prijem novih poruka.

8.2.5 Serverski dio aplikacije

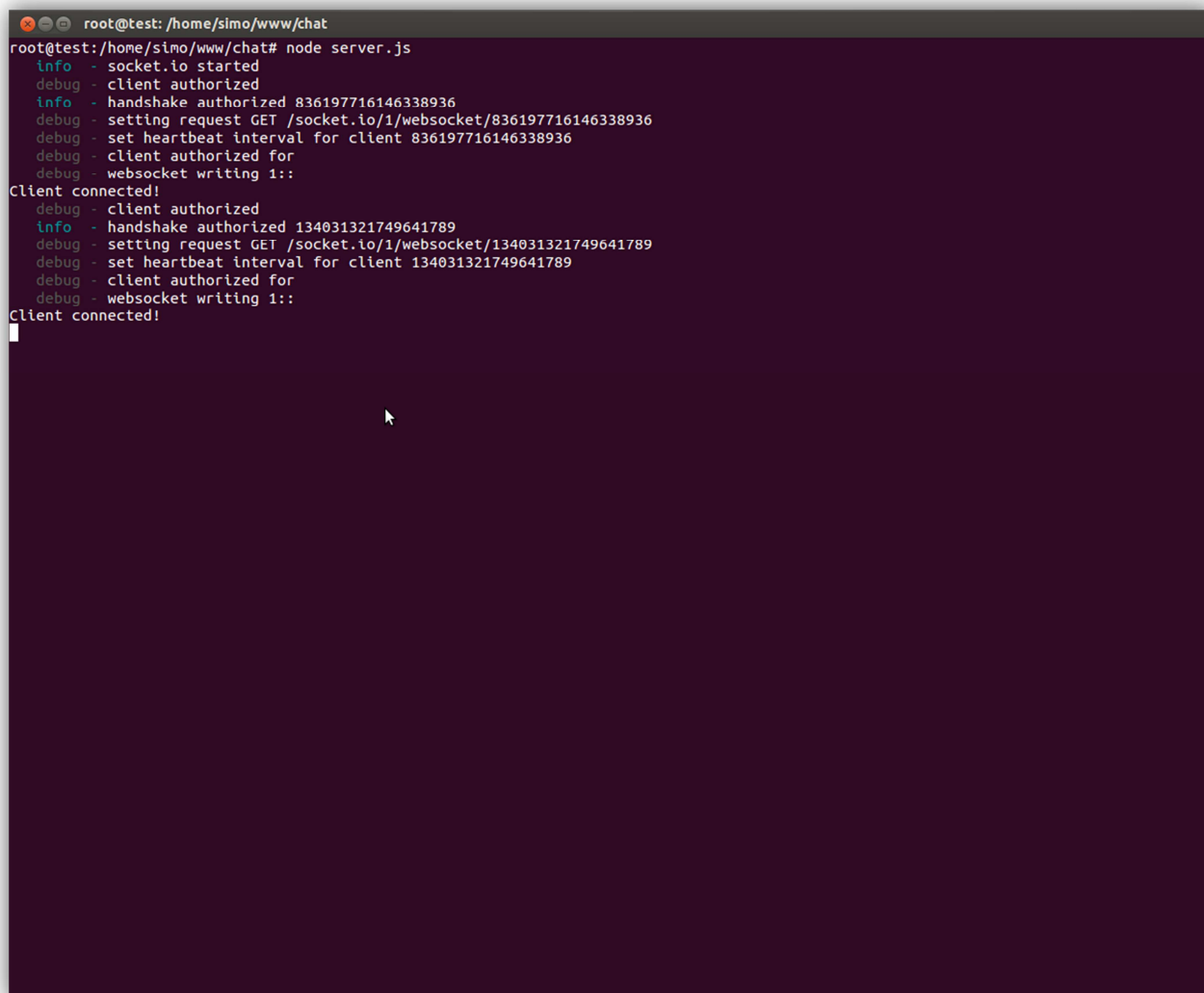
Server se pokreće iz konzole linux operativnog sistema pomoću programa node i to narednom node <ime_fajla_servera> u našem slučaju node server.js. Važno je napomenuti da ukoliko se koristi privilegovani port server mora biti pokrenut sa svim pravima super korisnika. Ovo se najčešće postiže naredbom sudo. Ukoliko bismo koristili privilegovani port naredba za pokretanje servera bi bila sudo nano server.js. Aplikacija razmatrana u ovom radu realizovana je na linux platformi ali to ne mora da bude pravilo. Node.js je platformski nezavisna tako da naš server može da se implementira kako na Windows-u tako i na recimo Mac-OS operativnom sistemu. Prilikom pokretanja servera dobićemo poruku prikazanu na slici II-5. Ukoliko se dogodi greška i server se neuspješno startuje biće prikazana odgovarajuća poruka o uzroku neuspjeha.

A terminal window with a dark purple background. The title bar at the top shows window control icons and the text 'root@test: /home/simo/www/chat'. The terminal content shows the command '^Croot@test:/home/simo/www/chat# node server.js' and its output 'info - socket.io started'. A white cursor is positioned on the line following the output. The terminal has a thin orange vertical bar on the right side.

```
root@test: /home/simo/www/chat
^Croot@test:/home/simo/www/chat# node server.js
info - socket.io started
```

Slika 23 - Pokretanje chat servera.

Prilikom konekcije novih korisnika na server u konzoli će biti ispisane odgovarajuće poruke koje administratoru servera mogu pružiti uvid u trenutno stanje konekcija. Ovakve poruku su izuzetno korisne jer pružaju uvid u procese koji se odvijaju u aplikaciji.



```
root@test: /home/simo/www/chat
root@test:/home/simo/www/chat# node server.js
info - socket.io started
debug - client authorized
info - handshake authorized 836197716146338936
debug - setting request GET /socket.io/1/websocket/836197716146338936
debug - set heartbeat interval for client 836197716146338936
debug - client authorized for
debug - websocket writing 1::
Client connected!
debug - client authorized
info - handshake authorized 134031321749641789
debug - setting request GET /socket.io/1/websocket/134031321749641789
debug - set heartbeat interval for client 134031321749641789
debug - client authorized for
debug - websocket writing 1::
Client connected!
```

Slika 24 - Ispis poruka prilikom konekcije novih korisnika.

Svaka poslata poruka korisnika biće evidentirana i ispisana na serverskoj konzoli. U poruci se prenosi HTML kod, ime tj. nadimak osobe koja je poslala poruku te tekst same poruke. Sva komunikacija odvija se preko JSON-a(Slika 25).

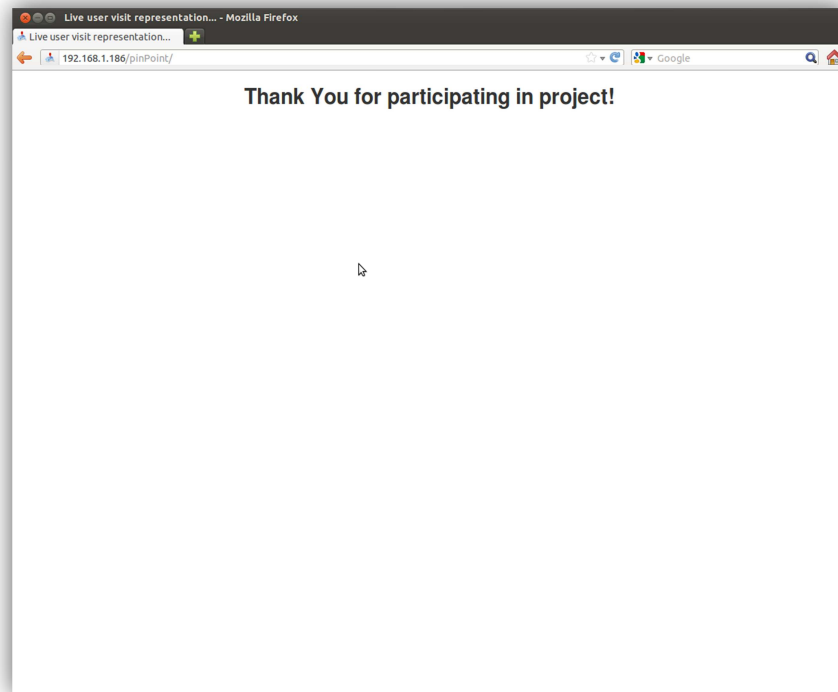
```
root@test: /home/simo/www/chat
root@test: /home/simo/www/chat# node server.js
info - socket.io started
debug - client authorized
info - handshake authorized 836197716146338936
debug - setting request GET /socket.io/1/websocket/836197716146338936
debug - set heartbeat interval for client 836197716146338936
debug - client authorized for
debug - websocket writing 1::
Client connected!
debug - client authorized
info - handshake authorized 134031321749641789
debug - setting request GET /socket.io/1/websocket/134031321749641789
debug - set heartbeat interval for client 134031321749641789
debug - client authorized for
debug - websocket writing 1::
Client connected!
debug - emitting heartbeat for client 836197716146338936
debug - websocket writing 2::
debug - set heartbeat timeout for client 836197716146338936
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 836197716146338936
debug - set heartbeat interval for client 836197716146338936
debug - emitting heartbeat for client 134031321749641789
debug - websocket writing 2::
debug - set heartbeat timeout for client 134031321749641789
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 134031321749641789
debug - set heartbeat interval for client 134031321749641789
debug - broadcasting packet
debug - websocket writing 5::{"name":"private message","args":[{"msg":"<tr><td><span class=\"icon small gray\" data-icon=\"u\" title=\"User\" style=\"display: inline-block; \"><span aria-hidden=\"true\">u</span></span><b> Osoba A</b>: Zdravo!</td></tr>"}]}
Private message received: <tr><td><span class="icon small gray" data-icon="u" title="User" style="display: inline-block; "><span aria-hidden="true">u</span></span><b> Osoba A</b>: Zdravo!</td></tr>
debug - broadcasting packet
debug - websocket writing 5::{"name":"private message","args":[{"msg":"<tr><td><span class=\"icon small gray\" data-icon=\"u\" title=\"User\" style=\"display: inline-block; \"><span aria-hidden=\"true\">u</span></span><b> Osoba B</b>: Pozdrav!</td></tr>"}]}
Private message received: <tr><td><span class="icon small gray" data-icon="u" title="User" style="display: inline-block; "><span aria-hidden="true">u</span></span><b> Osoba B</b>: Pozdrav!</td></tr>
debug - emitting heartbeat for client 836197716146338936
debug - websocket writing 2::
debug - set heartbeat timeout for client 836197716146338936
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 836197716146338936
debug - set heartbeat interval for client 836197716146338936
debug - emitting heartbeat for client 134031321749641789
debug - websocket writing 2::
debug - set heartbeat timeout for client 134031321749641789
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 134031321749641789
debug - set heartbeat interval for client 134031321749641789
```

Slika 25 - prikaz konzole servera i protoka poruka.

8.3 PinPoint.

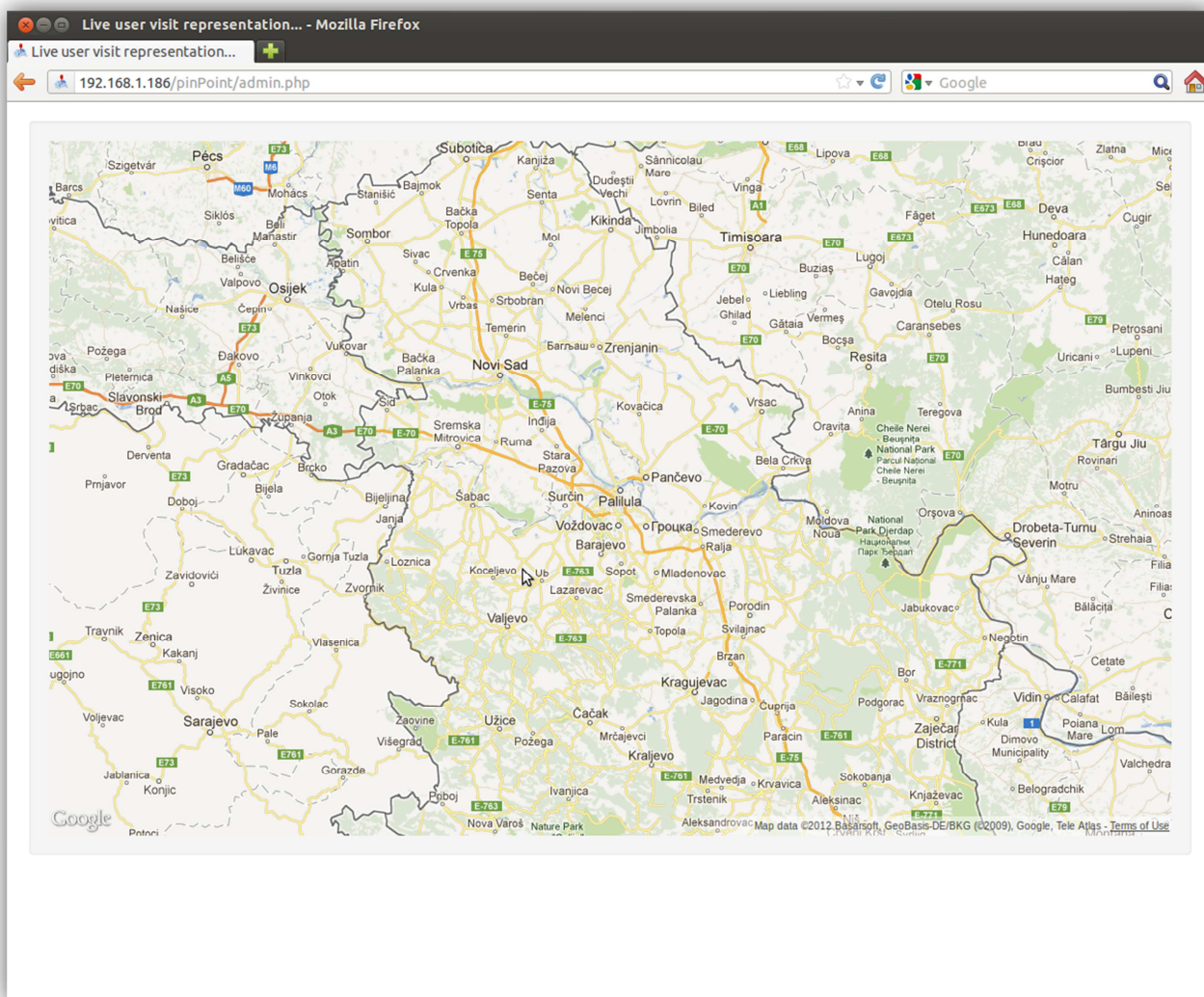
8.3.1 Opis aplikacije i princip rada

Poslednja aplikacija razmotreba u ovom radu demonstriraće kakve nove mogućnosti WebSocket može da nam omogućiti. Aplikacija pinpoint osmišljena je tako da webmasterima omoguće uvid u realnom vremenu broj posjeta njihovom sajtu te lokacije klijenata koji pristupaju istom. Aplikacija se sastoji od 3 dijela i to korisničkog, administratorskog i serverskog. Klijentski dio koda predstavlja samo jedno parče koda koje je potrebno implementirati na naslovnu stranu određenog web sajta. Ovaj dio koda kada se aktivira serveru dostavlja podatak o lokaciji korisnika te informaciju da se dogodio pristup sajtu slika 26.



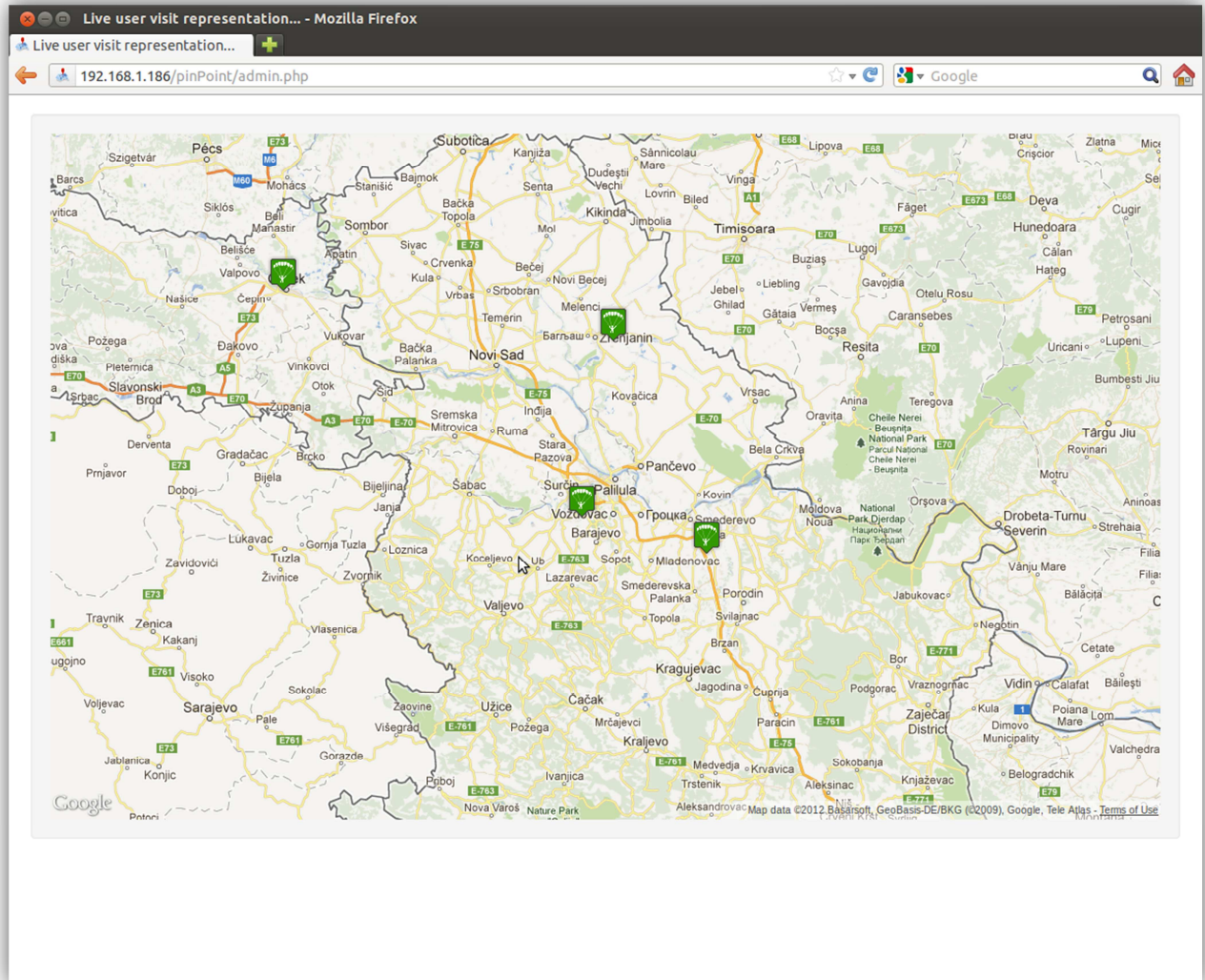
Slika 26 - Klijentska strana aplikacije

Administratorska strana aplikacije korisniku tj. webmasteru prezentuje na karti lokaciju sa koje je web sajt posjećen. Serverska strana zadužena je za očičku obradu podataka. Ova aplikacija realizovana je korišćenjem WebSocket protokola tako da se sve deava u realnom vremenu te webmaster ima trenutne podatke o posjećenosti određenog web sajta (Slika 27).



Slika 27 - Administratorska strana aplikacije.

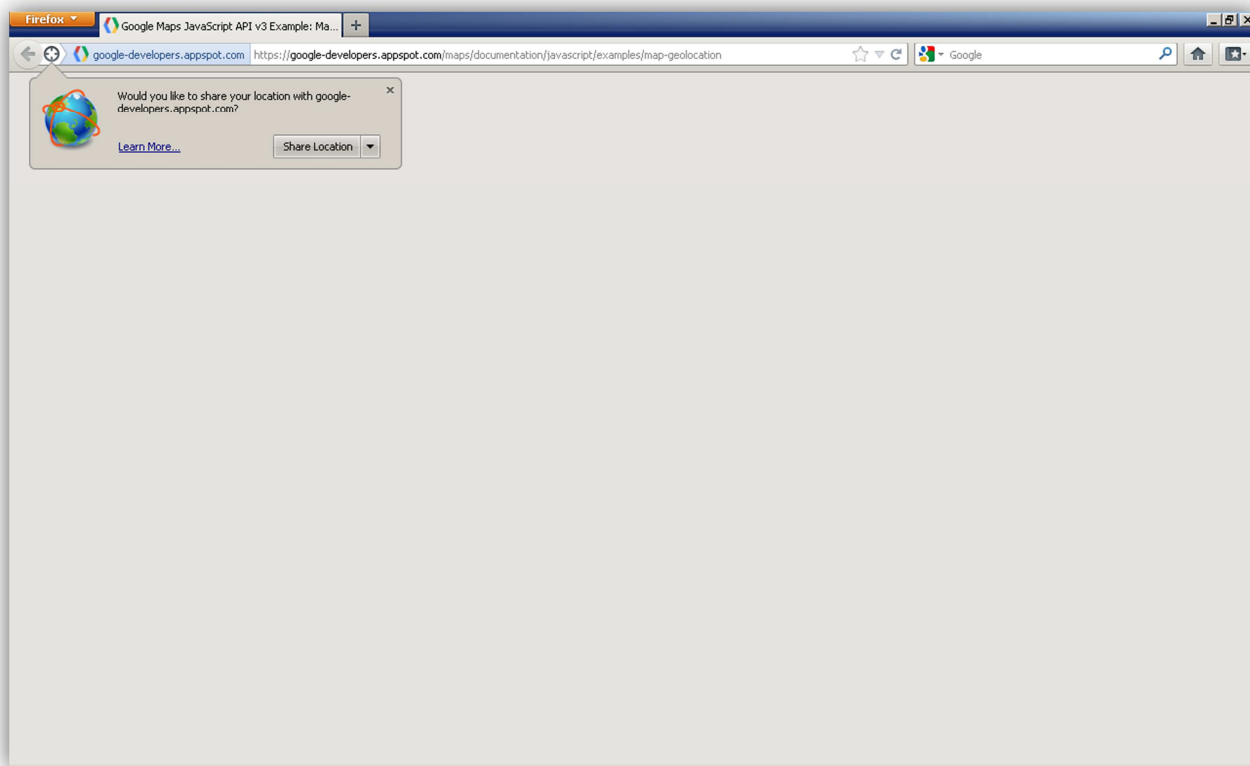
Svaki put kada stranica na kojoj se nalazi klijentski dio koda učita na administratorskoj strani lokacija klijenta koji je pristupio sajtu biće iscrtana na mapi kao što je to prikazano na slici 28.



Slika 28 - Ucrtavanje lokacija klijenta koji je pristupio sajtu.

Postoji nekoliko načina za dobijanje geografskih koordinata klijenta koji pristupa sjatu ili aplikaciji. Rezultati mogu da variraju zavisno od implementacije do implementacije. Neke pružaju preciznije koordinate ali zauzvrat zahtijevaju eksplicitan pristanak korisnika druge pak dobijaju nešto nepreciznije koordinate ali zato ze izvršavaju bez eksplicitne dozvole korisnika.

HTML5 standard nudi programerima takozvani "Geolocation API" preko kojeg je moguće dobiti izuzetno precizne geografske koordinate klijenta. Ukoliko klijent koristi neki od mobilnih uređaja i ima pristup GSM mreži geografske koordinate mogu biti izračunate pomoću baznih stanica što za rezultat ima jako tačnu lokaciju korisnika. Međutim i ova medalja ima svoju drugu stranu. Da bismo dobili geografske koordinate klijenta on mora da potvrdi da je saglasan sa tim (Slika 29, 30).



Slika 29 - Potvrda saglasnosti korisnika u web pretraživaču.



Slika 30 - Potvrda saglasnosti korisnika na mobilnom uređaju.

Naravno, koliko god bezazleno ovo bilo većina korisnika će po automatizmu odlučiti da ne prihvati uslove tj. da se ne složi s tim da njegova lokacija postane dostupna. Samim tim što korisnik ima pravo da odbije pristanak podaci koje dobijamo neće biti sasvim tačni tj. u našem slučaju nećemo imati uvid u pravo stanje stvari. Kao posjetioci sajta tj. aplikacije biće evidentirani samo klijenti koji su se složili s tim da sa nama podijele svoju lokaciju.

Drugo rešenje jeste da se lokacija klijenta utvrdi na osnovu njegove IP adrese sa kojom pristupa aplikaciji. Ovakav pristup za razliku od prethodnog daje geografske koordinate koje imaju visok stepen netačnosti. Preciznost je često na naivou grada što znači da će svi klijenti koji pristupaju aplikaciji a dolaze iz istog grada imati jednu zajedničku lokaciju. U našem slučaju to ne predstavlja problem jer webmasterima nije bitno naselje ili ulica i broj.

Klijentski segment ove aplikacije sastoji se iz svega par linij koda. Njegov zadatak jeste samo da otvori konekciju ka WebSocket serveru i preko te konekcije pošalje koordinate novopridošlog klijenta. Dakako u ovom slučaju nije naročito bitno da li je korišćen WebSocket kao protokol ili običan HTTP jer se ionako radi samo o jednom zahtjevu.

8.3.2 Izvorni kod

Izvorni kod klijentske strane aplikacije.

//Kod koji se nalazi na klijentskoj strani aplikacije

```
<script type="text/javascript">
$(function() {
    var socket = io.connect("http://192.168.1.186:80");
    console.log(navigator);
    console.log(google);
    if(google.loader.ClientLocation){
        position = new Object();
        position.lat = google.loader.ClientLocation.latitude;
        position.lng = google.loader.ClientLocation.longitude;
        position.info = "<div><table
class='\"table\"'><tr><td>Platform:</td><td>"+navigator.platform+"</td></tr><tr><td>Location:</td><td>"+go
ogle.loader.ClientLocation.address.city+", "+google.loader.ClientLocation.address.country+",
"+google.loader.ClientLocation.address.region+"</td></tr></table></div>";
        socket.emit('newVisitor',position);
    }
});
</script>
```

Klijentski kod ove aplikacije oslanja se na Google Maps API preko kojeg se izvlače geografska širina i dužina na kojoj se korisnik nalazi.

Administratorski dio ove aplikacije zadužen je za iscrtavanje lokacija novopridošlih korisnika na mapi. On dobija podatke od WebSocket servera, vrši njihovu obradu te tačke iscrta na mapi.

//Izvorni kod administratorskog dijela aplikacije

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Live user visit representation...</title>
    <meta charset="utf-8">
    <meta name="Autor" content="simo simeunovic">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="Lokacije korisnika">
    <meta name="author" content="Simo Simeunović">
    <link rel="shortcut icon" type="image/x-icon" href="images/icons/favicon.ico">

    <link href="adit_libs/bootstrap/css/bootstrap.css" rel="stylesheet">
    <style type="text/css">
        body {
            padding-top: 20px;
            padding-bottom: 40px;
        }
        .sidebar-nav {
            padding: 9px 0;
        }
    </style>
    <link href="adit_libs/bootstrap/css/bootstrap-responsive.css" rel="stylesheet">
```

```

<!--[if lt ie 9]>
  <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
<![endif-->

<link rel="shortcut icon" href="../assets/ico/favicon.ico">
<link rel="apple-touch-icon-precomposed" sizes="114x114" href="../assets/ico/apple-touch-icon-114-
precomposed.png">
<link rel="apple-touch-icon-precomposed" sizes="72x72" href="../assets/ico/apple-touch-icon-72-
precomposed.png">
<link rel="apple-touch-icon-precomposed" href="../assets/ico/apple-touch-icon-57-precomposed.png">
<script type="text/javascript" src="http://code.jquery.com/jquery-1.7.2.min.js"></script>
<script type="text/javascript" src="node_modules/socket.io/node_modules/socket.io-
client/dist/socket.io.js"></script>
<script type="text/javascript"
  src="http://maps.googleapis.com/maps/api/js?key=google_api_key&sensor=true">
</script>
</head>
<body>
  <div class="container">
    <div id="map" class="well">
      <div id="map_canvas" style="height: 700px;"></div>
    </div>
  </div>
  <script type="text/javascript">
    var map;
    var socket = io.connect("http://192.168.1.186:80");

    function initializeMap() {
      var myOptions = {
        center: new google.maps.LatLng(44.8206, 20.4622),
        zoom: 8,
        panControl: false,
        zoomControl: false,
        mapTypeControl: false,
        scaleControl: false,
        streetViewControl: false,
        overviewMapControl: false,
        mapTypeId: google.maps.MapTypeId.ROADMAP
      };
      map = new google.maps.Map(document.getElementById("map_canvas"), myOptions);
    }

    function addMarker(obj){
      var myLatLng = new google.maps.LatLng(obj.lat,obj.lng);
      var markerIco = 'images/paragliding.png';
      var marker = new google.maps.Marker({
        position: myLatLng,
        map: map,
        animation: google.maps.Animation.DROP,
        icon: markerIco
      });
      addInfoWindow(marker, obj.info);
    }

    socket.on('locationNew', function(obj) {
      marker = addMarker(obj);
    }
  </script>

```

```

});

function addInfoWindow(marker, message) {
    var info = message;

    var infoWindow = new google.maps.InfoWindow({
        content: message
    });

    google.maps.event.addListener(marker, 'click', function () {
        infoWindow.open(map, marker);
    });
}

$(function() {
    initializeMap();
});
</script>
<footer>
    <script src="adit_libs/bootstrap/js/bootstrap-transition.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-alert.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-modal.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-dropdown.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-scrollspy.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-tab.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-tooltip.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-popover.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-button.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-collapse.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-carousel.js"></script>
    <script src="adit_libs/bootstrap/js/bootstrap-typeahead.js"></script>
</footer>
</body>
</html>

```

Na početku html dokumenta se ostvari konekcija sa WebSocket serverom i konstruiše se objekat preko kojeg se vrši komunikacija sa WS serverom. Takođe renderuje se mapa na kojoj će se iscrtavati lokacije novih klijenata. Nakon kreiranog objekta istom se određuje da osluškuje na sve poruke od servera koje nose naslov "locationNew". Kad god server pošalje ovakvu poruku web pretraživač će prihvatiti to, obraditi podatke te pozvati funkciju addMarket koja iscrtava marker na mapi.

```

//Osluškivanje za nove poruke od servera

socket.on('locationNew', function(obj) {
    marker = addMarker(obj);
});

```

Serverska strana ove aplikacije, kao što je to slučaj i kod ranijih primjera, pisana je korišćenjem node.js biblioteke JavaScript klasa koje omogućavaju implementaciju web socket servera. Sav zadatak koji server ima jeste da prihvati poruku o konekciji novog

korisnika te da istu distribuira na sve aktivne konekcije koje dolaze sa administrativnog dijela aplikacije.

```
//Izvorni kod serverske strane aplikacije

var app = require('http').createServer(handler)
, io = require('socket.io').listen(app)
, fs = require('fs')
, util = require('util')
, http = require('https');
app.listen(80, '192.168.1.186');

function handler (req, res) {
  fs.readFile(__dirname + '/index.php',
    function (err, data) {
      if (err) {
        res.writeHead(500);
        return res.end('Error loading index.html');
      }

      res.writeHead(200);
      res.end(data);
    });
}

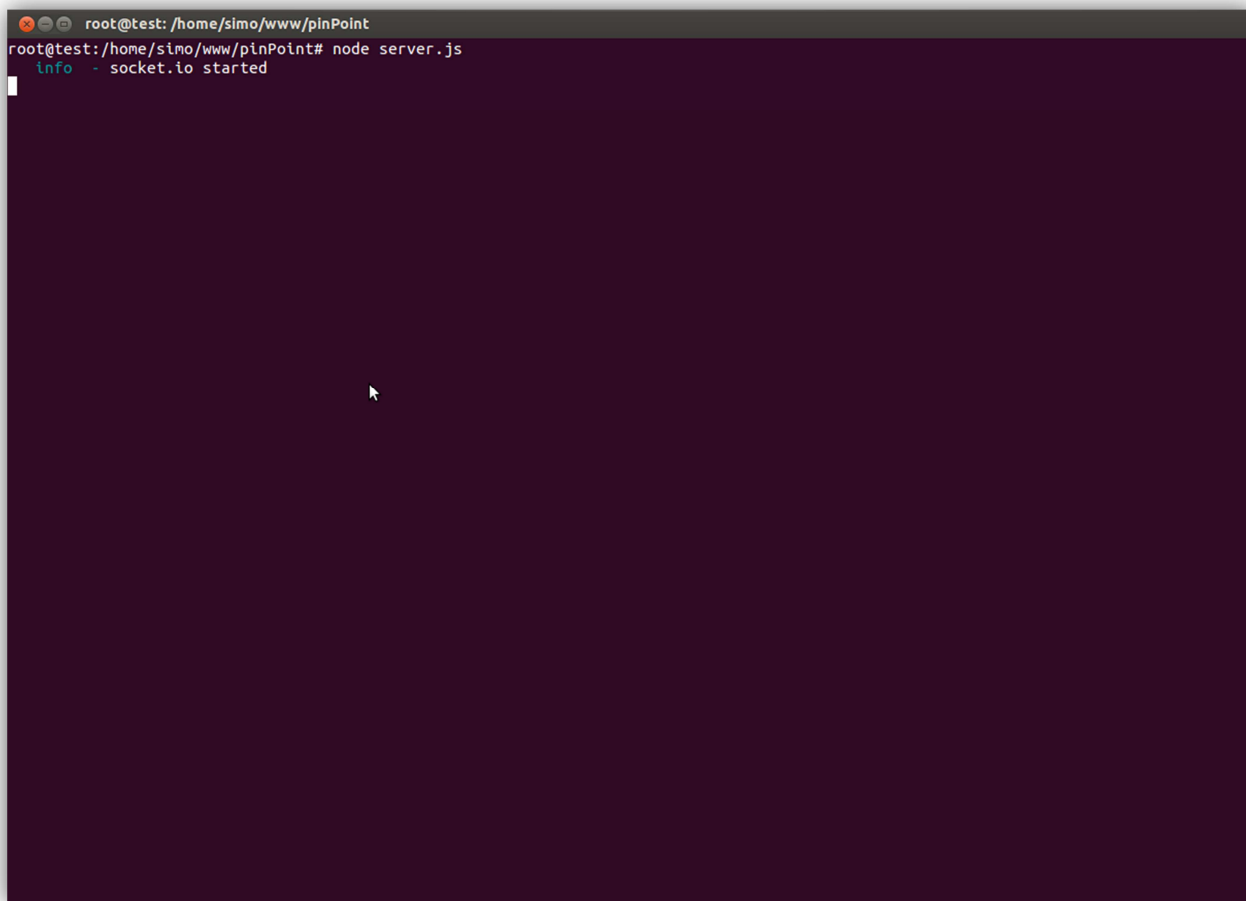
io.sockets.on('connection', function (socket) {
  console.log('Client connected!');
  socket.on('newVisitor', function(obj){
    console.log(obj);
    io.sockets.emit('locationNew',obj );
  });
});

io.sockets.on('newVisitor', function(obj){
  console.log(obj);
  io.sockets.emit('locationNew',obj );
});

io.sockets.on('disconnect', function(){
  console.log("Disconected");
});
```

8.3.3 Serverska strana

Server se pokreće istovjetno kao i serveri prethodnih aplikacija. Pomoću programa node tj. uz privilegije super korisnika ukoliko se koriste privilegovani portovi. Nakon komande za pokretanje server počinje da osluškuje na zadatom portu slika 31.



```
root@test: /home/simo/www/pinPoint
root@test:/home/simo/www/pinPoint# node server.js
info - socket.io started
```

Slika 31 - Pokretanje servera pinPoint aplikacije.

Nakon što je pokrenut server on će osluškivati za sve nove konekcije. Kada se dogodi posjeta novog klijenta poruka će biti distribuirana na sve aktivne konekcije ali samo one koje dolaze sa administratorskog dijela aplikacije.

```
root@test: /home/simo/www/pinPoint
debug - client authorized for
debug - websocket writing 1::
Client connected!
info - transport end (undefined)
debug - set close timeout for client 11847020031381128810
debug - cleared close timeout for client 11847020031381128810
debug - cleared heartbeat interval for client 11847020031381128810
debug - discarding transport
debug - client authorized
info - handshake authorized 5264578661791616662
debug - setting request GET /socket.io/1/websocket/5264578661791616662
debug - set heartbeat interval for client 5264578661791616662
debug - client authorized for
debug - websocket writing 1::
Client connected!
info - transport end (undefined)
debug - set close timeout for client 5264578661791616662
debug - cleared close timeout for client 5264578661791616662
debug - cleared heartbeat interval for client 5264578661791616662
debug - discarding transport
debug - client authorized
info - handshake authorized 11127732291112804173
debug - setting request GET /socket.io/1/websocket/11127732291112804173
debug - set heartbeat interval for client 11127732291112804173
debug - client authorized for
debug - websocket writing 1::
Client connected!
info - transport end (undefined)
debug - set close timeout for client 11127732291112804173
debug - cleared close timeout for client 11127732291112804173
debug - cleared heartbeat interval for client 11127732291112804173
debug - discarding transport
debug - client authorized
info - handshake authorized 13017282252043500071
debug - setting request GET /socket.io/1/websocket/13017282252043500071
debug - set heartbeat interval for client 13017282252043500071
debug - client authorized for
debug - websocket writing 1::
Client connected!
debug - emitting heartbeat for client 15681219361136360368
debug - websocket writing 2::
debug - set heartbeat timeout for client 15681219361136360368
debug - got heartbeat packet
debug - cleared heartbeat timeout for client 15681219361136360368
debug - set heartbeat interval for client 15681219361136360368
```

Slika 32 - Konekcije novih klijenata.

9 Zaključak

Možemo sa sigurnošću da konstatujemo da će web aplikacije nastaviti da preuzimaju primat nad tradicionalnim, robusnim desktop aplikacijama. Naravno ova slika nikada neće biti crna ili bijela tj. nikada neće postojati samo jedna ili druga vrsta jer one ne mogu da funkcionišu jedna bez druge, ali će jedna uvijek imati primat u odnosu na drugu. Svakim danom sve se više tradicionalnih desktop servisa prenosi na Internet tj. kako se to danas popularno naziva postaje dio "Računarstva u oblaku". Sa ovakvim naglim razvojem web aplikacija unapređenja će morati biti primjenjena i na tradicionalni način komuniciranja ovih aplikacija zato što HTTP protokol jednostavno nije u mogućnosti da podrži takozvani "event-based" način rada modernih web aplikacija. Imajući u vidu prepreke koje su se javili odgovorni ljudi koji brinu za o sudbini Interneta razvili su, tj. bolje rečeno još uvijek razvijaju HTML5 standard koji među velikim brojem novina donosi i rešenje ovog rastućeg problem komunikacije.

WebSocket će svakako poprimati na značaju kako vrijeme bude odmicalo jer baš kao što je jedan zvaničnik kompanije Google izjavio *"Reducing kilobytes of data to 2 bytes...and reducing latency from 150ms to 50ms is far more than marginal. In fact, these two factors alone are enough to make WebSockets seriously interesting to Google."* prevod ovog citata na Srpski jezik glasi: Smanjenje kolobajti podataka na samo 2 bajta i smanjenje latencije sa 150ms na 50ms je mnogo više nego samo marginalan napredak. U stvari, samo ova dva faktora daju dovoljno povoda da kompanija Google ozbiljno shvati WebSocket.

10 Literatura

10.1.1 Pisana izdanja

- [1] Veinović, M. i Jevremović, A. (2011), Računarske mreže, Univerzitet Singidunum
- [2] Nikolić, B. (2008), Programiranje grafičkih aplikacija, 2. dopunjeno izdanje, Univerzitet Singidunum
- [3] Nikolić, B. (2008), Internet programiranje pomoću programskog jezika JAVA, Univerzitet Singidunum
- [4] Lubbers, P., Albers B., Salim F. (2011) Pro HTML5 Programming (Professional Apress) 2. edition, Apress
- [5] Matthew, D. (2010) HTML Designing rich internet applications, USA

10.1.2 Internet izvori

- 1. <http://kaazing.com/products/html5-edition>
- 2. <http://www.codeproject.com/Articles/209041/HTML5-Web-Socket-in-Essence>
- 3. <http://nodejs.org/>
- 4. <http://socket.io/>
- 5. <https://dev.twitter.com/>
- 6. <https://developers.google.com/>
- 7. <http://en.wikipedia.org/wiki/WebSocket>
- 8. <https://developer.mozilla.org/en/WebSockets/>
- 9. <http://tools.ietf.org/html/rfc6455>
- 10. <http://www.whatwg.org/specs/web-socket-protocol/>
- 11. <http://twitter.github.com/bootstrap/>