Razvoj aplikacije za razmjenu trenutnih poruka

Savanović, Ivan

Sveučilište u Rijeci – Odjel za informatiku

Jednopredmetna informatika

Ivan Savanović

RAZVOJ APLIKACIJE ZA RAZMJENU TRENUTNIH PORUKA

Diplomski rad

Mentor: doc. dr. sc. Lucia Načinović Prskalo

Rijeka, prosinac 2021.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 3.5.2021.

Zadatak za diplomski rad

Pristupnik: Ivan Savanović

Naziv diplomskog rada: Razvoj aplikacije za razmjenu trenutnih poruka

Naziv diplomskog rada na eng. jeziku: Development of instant messaging application

Sadržaj zadatka: Glavni zadatak završnog rada je izraditi web aplikaciju za razmjenu trenutnih poruka. Pritom će se koristiti odabrani alati za razvoj frontend i backend dijela aplikacije koji će se u radu i detaljno opisati. Također će se opisati i demonstrirati svi važni elementi i funkcionalnosti izrađene web aplikacije.

Mentor:
Doc. dr. sc. Lucia Načinović Prskalo

Načinović Prskalo

Voditeljica za diplomske radove:
Izv. prof. dr. sc. Ana Meštrović

Zadatak preuzet: 3.5.2021.

Ivan Savanović

(potpis pristupnika)

Sažetak

Rad se sastoji od teorijskog i praktičnog dijela. U teorijskom dijelu dana je povijest trenutnih poruka, sigurnost te je opisana klijent server komunikacija. Predstavljeni su alati i razvojna okruženja React, Node.js i MongoDB te njihove prednosti i mane. Zatim je dan opis servisa Amazon Elastic Compute Cloud. U drugom dijelu opisan je praktični dio rada, odnosno izrađena aplikacija i njezine funkcionalnosti.

Ključne riječi: trenutne poruke, node.js, websocket protokol, mongodb, react, Amazon elastic compute cloud

SADRŽAJ

1. Uvod

Razmjena trenutnih poruka predstavlja komunikaciju u stvarnom vremenu koja se temelji na tekstu. Razvoj aplikacija za razmjenu trenutnih poruka je započeo s ciljem omogućavanja komunikacije s prijateljima, a danas su postale važna komunikacijska platforma za cijelu internetsku zajednicu i za poslovni svijet. U ovom radu dan je teorijski uvod i povijest trenutnih poruka te je opisan njihov princip rada. Kako bi se bolje objasnila prikazana tema, uz teorijske osnove, bit će prikazana i izrađena aplikacija za razmjenu trenutnih poruka. Dan je i pregled sigurnosnih rizika do kojih može doći pri korištenju aplikacija za trenutne poruke. U četvrtom poglavlju je opisana klijent server komunikacija, način na koji se ona provodi korištenjem isključivo HTTP protokola te upotrebom WebSocket-a. U petom poglavlju je opisana JavaScript biblioteka React, njezina svrha te prednosti i nedostaci. Šesto poglavlje opisuje MongoDB te njezine glavne značajke. U sedmom poglavlju opisana su svojstva Node.js-a. Zatim je opisan Amazon Elastic Compute Cloud (Amazon EC2) i njegove značajke. Web adresa aplikacije je 3.69.171.38. Deveto poglavlje sadrži opis izrađene aplikacije. U posljednjem poglavlju dan je kratki zaključak.

2. Trenutne poruke

Trenutne poruke (*engl. Instant Messaging* ponekad nazvan IM ili TIMing), tj. njihova razmjena, vrsta je mrežnog razgovora koji nudi prijenos teksta u stvarnom vremenu putem interneta [1]. Dakle, trenutne poruke predstavljaju oblik komunikacije u stvarnom vremenu između dvije ili više osoba na temelju upisanog teksta. Tekst se prenosi putem povezanih računala ili mobilnih uređaja preko mreže kao što je Internet. Pojavom tehnologija kao što je bežični aplikacijski protokol (*engl. Wireless Application Protocol ili WAP*) i porastom popularnosti ručnih uređaja poput mobilnih telefona, usluga kratkih poruka (SMS) dodala je novu dimenziju integraciji razmjena trenutnih poruka. Kada je uslijedio i masovni porast uporabe interneta, popularnost aplikacije za trenutne poruke je eksplodirala, a danas možemo reći da su one sastavni dio društvenog života.

Povijesno gedano, postojanje trenutnih poruka datira u vrijeme razvoja aplikacija s pričanjem i pisanjem koje su bile razvijene za UNIX operacijski sustav. Jedan od prethodnika formalnih trenutnih poruka bio je kompatibilan sustav dijeljenja vremena *(engl. Compatible Time-Sharing System ili CTSS)*, koji je nastao 1961. godine u Računskom centru MIT-a koji je bio smješten u velikom glavnom računalu. Korisnici su bili povezani s glavnim računalom putem telefonskih priključaka kako bi mogli međusobno slati poruke i razmjenjivati datoteke. Kompatibilan sustav dijeljenja vremena je do 1965. godine prerastao MIT te je usvajao moderne kvalitete koje nalikuju na današnje trenutne poruke dopuštajući nekoliko stotina korisnika s više fakulteta da međusobno razgovaraju [8].

Tijekom 1973. godine pojavio se prvi softver za javni chat - “Talk”, dizajniran za rad u operacijskom sustavu UNIX koji je također zahtijevao da korisnici budu prijavljeni na isto računalo kako bi mogli koristiti program. Ovo je bio preteča sustava trenutnih poruka budući da su korisnici mogli poslati poruku bilo kome u sustavu. Ovaj softver često se koristio u kombinaciji s "Finger" programom koji je korisnicima omogućavao da vide koji su korisnici online. Prvo veliko uvođenje trenutnih poruka uvela je kompanija America Online (AOL) koja je pokrenula vlastitu verziju trenutnih poruka s komponentom koja se koristi za upravljanje svim dolaznim i odlaznim porukama u obliku popisa poznanika koji

korisnicima America Online daje do znanja kada su njihovi prijatelji, rodbina ili drugi poznanici koji su također korisnici America Onlinea bili na mreži. Ova komponenta je popularno poznata kao lista prijatelja. Takvi popisi nazvani su liste prijatelja nakon uvođenja AOL Instant Messengera (AIM) 1997. godine [10].

Do 1990. godine nije došlo do značajne promjene statusa trenutnih poruka.. Uglavnom zato što se trenutne poruke do tada nisu uzimale ozbiljno. Naknadno je korporacijski svijet promijenio svoj stav prema izravnim porukama zahvaljujući internetskoj revoluciji. Naglim porastom popularnosti interneta i dolaskom nove tehnologije, poput glasovne pošte i internetskih transakcija, korporacijski svijet je počeo ozbiljno shvaćati razmjenu trenutnih poruka. Razmjena trenutnih poruka doista je eksplodirala na internetskoj sceni 1996. godine [2]. Tada je Mirabilis predstavio besplatni uslužni program za razmjenu trenutnih poruka ICQ, koji je svatko mogao koristiti. ICQ - skraćenica za "Tražim te" koristi softversku aplikaciju zvanu klijent, koja se nalazi na računalu korisnika [10]. Klijent komunicira s ICQ poslužiteljem kad god je korisnik na mreži, a klijent radi. Internet nije doveo samo do značajnog povećanja broja korisnika trenutnih poruka, nego i do spoznaje potencijala trenutnih poruka te ozbiljnim pokušajima da se uklone ograničenja trenutnih poruka kako bi se mogle iskoristi u potpunosti. ICQ model temelj je većine pomoćnih programa za razmjenu trenutnih poruka na tržištu danas [4].

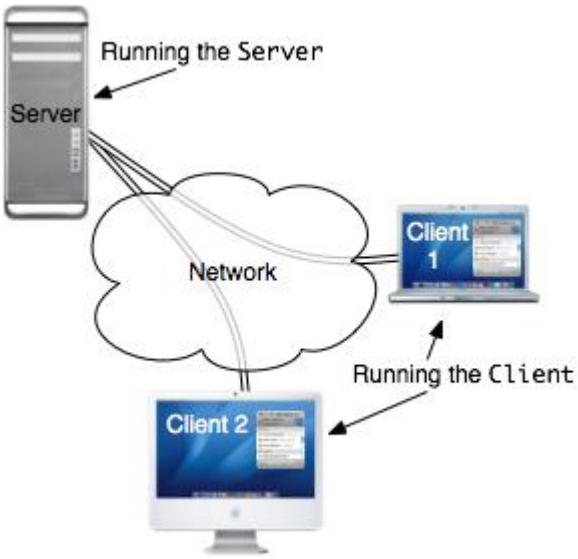
Pojavom iPhone i Android uređaja došlo je do značajnog porasta korištenja prijenosnih uređaja. Stoga su aplikacije za razmjenu trenutnih poruka postale sve popularnije na ovim uređajima. Aplikacije za trenutne poruke koje su razvijene posebno za upotrebu na mobilnim uređajima i obično služe kao alternativa SMS porukama, uključuju: Apple iMessage, WhatsApp Messenger, Discord, Viber i druge. Budući da je metoda komunikacije putem interneta postala popularna, komunikacija trenutnim porukama u određenoj je mjeri zamjenila komunikaciju e-poštom. Za razliku od e-pošte, razmjena trenutnih poruka omogućuje korisnicima da vide je li odabrani dopisnik spojen na internet [2].

Razmjena trenutnih poruka također se razlikuje od e-pošte u tome što se poruke razmjenjuju izravno gotovo trenutno, što omogućuje dvosmjernu komunikaciju u

stvarnom vremenu. Broj korisnika softvera za razmjenu trenutnih poruka brzo raste ne samo zbog poboljšanja komunikacije, već takva vrsta komunikacije pruža i najjeftiniji način komunikacije na daljinu pomoću besplatnih glasovnih poruka, video konferencija ili dijeljenja datoteka i direktorija [7]. Tijekom godina trenutne poruke su se pokazale kao prikladna tehnologija ne samo za zabavu, već i za poslovanje. Uz opće prihvaćanje trenutnih poruka, poslovni svijet danas gleda na njih kao na idealan alat za promicanje svojih interesa. Razmjena trenutnih poruka revolucionirala je poslovnu komunikaciju jer je vrlo pogodna i za radnike koji rade od doma [3].

2.1. Opis arhitekture razmjene trenutnih poruka

Većina programa za razmjenu trenutnih poruka preko interneta se povezuje s tvrtkom za razmjenu trenutnih poruka koja provjerava identitet korisnika. Taj se postupak zove autentifikacija. Kada se pokrene aplikacija za trenutne poruke, klijent pokušava kontaktirati poslužitelja trenutnih poruka određene tvrtke, poput Google-a ili Facebook-a. Nakon što se veza s poslužiteljem uspostavi, korisnik unosi svoje ime i lozinku za prijavu na poslužitelj. Zatim poslužitelj provjerava dano korisničko ime i lozinku, prijavljuje korisnika te je tada korisnik povezan [9]. Klijent zatim poslužitelju šalje i ostale podatke o povezivanju kao što su primjerice broj porta (*engl. port*), imena ljudi s popisa kontakata i IP adresa računala koje korisnik koristi. Poslužitelj stvara privremenu datoteku u koju će se smjestiti informacije o vezi i popis kontakata. Slika 1 prikazuje standardnu arhitekturu komunikacije trenutnim porukama.



Slika 1 Standardna arhitektura komunikacije trenutnim porukama [6]

Ako je bilo koji od korisnikovih kontakata prijavljen, poslužitelj šalje poruku koja sadrži podatke o vezi za tog korisnika natrag klijentu na korisnikovom računalu. Poslužitelj također šalje podatke o korisnikovoj vezi osobama na korisnikovom popisu kontakata koji su prijavljeni. Kako se korisnikovi kontakti prijavljuju i odjavljuju, tako se pojavljuju i nestaju s popisa kontakata. Ukoliko aplikacija ima te mogućnosti, statusi kontakata se automatski mijenjaju iz "Na mreži" u "Odsutni", "Zauzet" ili "Van mreže" ovisno o tome je li ta osoba prijavljena i koristi li računalo [5]. Budući da korisnikov klijentski softver ima broj porta i IP adresu računala ili drugog uređaja osobe kojoj korisnik šalje poruku, korisnikova će se poruka poslati izravno u softver klijenta na uređaju te osobe. Od tada poslužitelj više nije uključen jer se komunikacija sada odvija između dva klijentska softvera.

3. Sigurnost

Trenutne poruke su u današnje vrijeme sveprisutne i koriste ih mnogi zaposlenici kod kuće i na radnom mjestu. Stoga se u mnogim tvrtkama pored usluga trenutnih poruka za poduzetnike (*engl. Enterprise Instant Messaging ili EIM*) poput Sametime, Lync nalazi i mnoštvo drugih besplatni komunikacijskih alata uglavnom u privatnoj uporabi (primjerice Skype, AIM, Hangout i WhatsApp). Besplatni komunikacijski alati prikladni su za osobnu upotrebu, ali kada se koriste u organizaciji kojoj je sigurnost važna u svakodnevnoj komunikaciji, one predstavljaju sigurnosni rizik te često postaju meta programera zlonamjernog softvera (*engl. malware*) i virusa. Jedan od nedostataka ovih usluga je i omogućavanje pristupa telefonskim brojevima korisnika i cjelovitom imeniku. Aplikacije za razmjenu trenutnih poruka donose mnoge pogodnosti te se zbog toga sigurnosni problemi ne rješavaju detaljno [1].

Većina pružatelja usluga za razmjenu trenutnih poruka često veću važnost daje brzini rada nauštrb sigurnosti aplikacije te postoji mogućnost da pružatelj usluge bude prisiljen predati poruke nekoj trećoj strani. Najpopularniji klijenti sažimaju brojne komunikacijske funkcije u male, jednostavne za korištenje i jednostavne za konfiguriranje aplikacije s malim prostorom. Na primjer, nakon što je korisnik instalirao i prijavio se na mrežu javne usluge za razmjenu trenutnih poruka prikazuje se popis kontakata. Korisnik može komunicirati s bilo kojim od svojih kontakata koji su također na mreži. Sve se tekstualne poruke mogu preusmjeriti putem centralizirane zbirke poslužitelja, a zatim prema primatelju, ili se mogu izravno povezati putem peer-to-peer veza. Kod funkcija koje zahtijevaju veliku propusnosti poput prijenosa zvuka, videa i drugih digitalnih datoteka, takve peer-to-peer veze su posredovane od strane poslužitelja. Mnoge kompanije imaju svoje poslovne aplikacije za trenutne poruke koje su sigurne. Uobičajeno, zaposlenik ili član organizacije mora dobiti podatke za prijavu i odgovarajuća dopuštenja za korištenje sustava za razmjenu poruka. Pravljenje posebnog korisničkog računa za svakog korisnika omogućuje organizaciji da identificira, prati i evidentira svu upotrebu njihovog sustava za poruke. Kako bi se izbjegao nadzor, zaposlenici često instaliraju ili koriste aplikacije za komercijalnu upotrebu.

Kako bi umanjili poteškoće pri povezivanju, mnogi popularni klijenti trenutnih poruka usmjeruju promet kroz dobro zaštićena mrežna okruženja koristeći neovlaštene portove u korporacijskim vatrozidima. Ovakav pristup omogućuje stvaranje dodatnih ulaznih točaka u mrežu za viruse i lažne protokole zaobilazeći korporacijske sustave i kontrole autentifikacije. Ranjivosti unutar klijenata trenutnih poruka od strane svih značajnih proizvođača dovodi u opasnost integritet i povjerljivost korporativnih podataka što potencijalno dopušta da svi podaci kojima zaposlenik može pristupiti također postanu dostupni hakerima koji zloupotrebljavaju nedostatke u aplikaciji klijenta za razmjenu trenutnih poruka.

Neki od sigurnosnih rizika do kojih može doći kod aplikacija za razmjenu trenutnih poruka [27]:

- Klijentske ranjivosti - kao i mnoge druge softverske aplikacije, klijenti trenutnih poruka imaju povijest uobičajenih sigurnosnih propusta. Iskorištavanje ovih ranjivosti može se očitati u vidu uskraćivanja usluge, na primjer dostizanja maksimalne iskorištenosti propusnosti mreže i rušenja radne stanice, slanja neželjenih obavijesti, pristup neovlaštenim podacima korisnika ili potpuni gubitak podataka.
- Nesigurni mrežni promet - mrežno okruženje je zaštićeno sustavom obrane poput vatrozida, IDS/IPS, filtriranja sadržaja, protuvirusne aplikacije itd. Uspostavljene obrambene mjere bi trebale blokirati sve zlonamjerne mrežne aktivnosti započete izvan mreže. Klijenti trenutnih poruka učinkovito perforiraju vatrozid i pružaju alternativni kanal za viruse, neželjenu poštu i druge neovlaštene datoteke.
- Otvorene veze - kada se prenose datoteke, glasovni chat ili druge metode dijeljenja datoteka, klijent trenutnih poruka korisnicima otkriva pravu IP adresu. S tim podacima zlonamjerni korisnik može ući u sustav domaćina u svrhu hakiranja ili napada uskraćivanjem usluge [1].
- Krađa identiteta – klijenti trenutnih poruka obično koriste malo ili nimalo šifriranja za prijenos vjerodajnica za prijavu (*engl. login credentials*). Na internetu postoje vodiči koji pružaju najbolje savjete kako ih presresti i zabilježiti. Ukradene vjerodajnice tako se lako mogu koristiti za lažno predstavljanje nekom drugom.

- Krađa podataka - sposobnost tuneliranja kroz obranu perimetra čini učinkovitu metodu prijenosa povjerljivih materijala. Korisnici mogu koristiti klijente trenutnih poruka za prijenos podataka kao što su baze podataka korisnika i razvojni izvorni kod na vanjske kontakte bez upozorenja što može dovesti do krađe podataka ili koda. Kod nekih klijenata trenutnih poruka to se može nenamjerno postići lošom konfiguracijom usluga za dijeljenje datoteka.
- Gubitak privatnosti - ako šifriranje poruka nije uspjelo ili nije provedeno, poslane poruke su nezaštićene, što znači da promatrač može lako presresti i pročitati te poruke. U slučaju kada se uspostavljaju veze koje nisu peer-to-peer, sve poruke moraju putovati na središnji poslužitelj prije nego što budu proslijeđene primatelju gdje se mogu evidentirati i pohraniti. Slično, primatelj poruke također može evidentirati i spremiti te podatke za kasniju upotrebu. Ako je riječ o dijeljenu poruka unutar organizacije, korisnici možda ne znaju odvija li se ta razmjena preko interneta ili unutarnjeg servera.
- Nepostojanje provjere autentičnosti - budući da svaki korisnik može izabrati vlastiti identitet, nema jamstva da je primatelj poruke zaista onaj za koga se pretpostavlja. To može dovesti do toga da korisnik pošalje povjerljive ili privatne podatke osobi koja se predstavlja kao netko drugi.

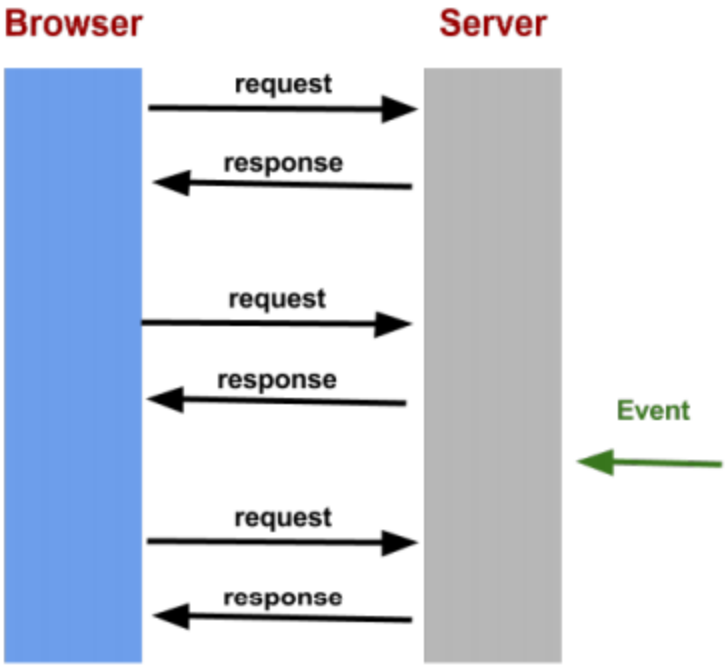
4. Klijent server komunikacija

Današnje aplikacije zahtijevaju slanje poruka u stvarnom vremenu. Bez obzira na to radi li se o web, mobilnoj ili kompozitnoj aplikaciji, korisnici očekuju da će moći komunicirati s podacima bez kašnjenja. Takvu komunikaciju se može postići korištenjem HTTP rješenja ili upotrebom WebSocket-a.

4.1. HTTP komunikacija u stvarnom vremenu

Kada se preko preglednika pristupi web stranici, šalje se HTTP zahtjev web poslužitelju na kojem je stranica postavljena. Web poslužitelj prihvaća zahtjev i šalje odgovor. U mnogim slučajevima, na primjer za cijene dionica, vijesti, prodaju karata, poruka preko interneta, e-pošta i drugo, odgovor bi mogao postati zastarjeli do trenutka kada preglednik prikaže stranicu. Kako bi se dobili najnoviji podatci u stvarnom vremenu, tu stranicu treba stalno osvježavati ručno, ali to očito nije dobro rješenje. Pokušaji pružanja usluga web aplikacija u stvarnom vremenu uglavnom se svode na anketiranje i druge poslužiteljske push tehnologije, od kojih je najistaknutija "Comet". Comet odgađa završetak HTTP odgovora kako bi isporučio poruku klijentu [20].

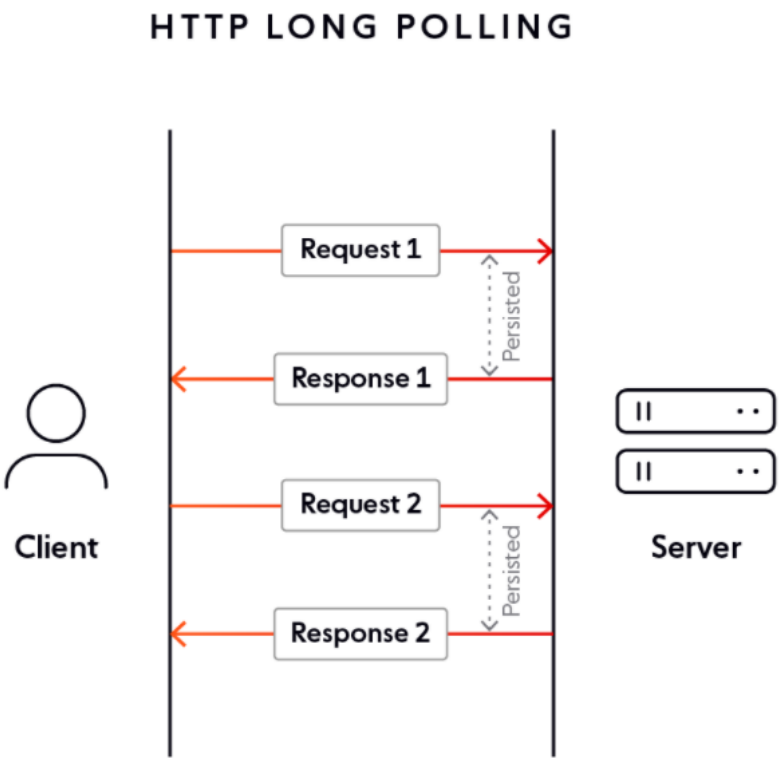
Anketiranjem (*engl. polling*) preglednik u redovitim intervalima šalje HTTP zahtjeve i odmah prima odgovor. Ova tehnika je bila prvi pokušaj da preglednik isporuči informacije u stvarnom vremenu. Na Slici 2 prikazano je HTTP anketiranje.



Slika 2 HTTP anketiranje [15]

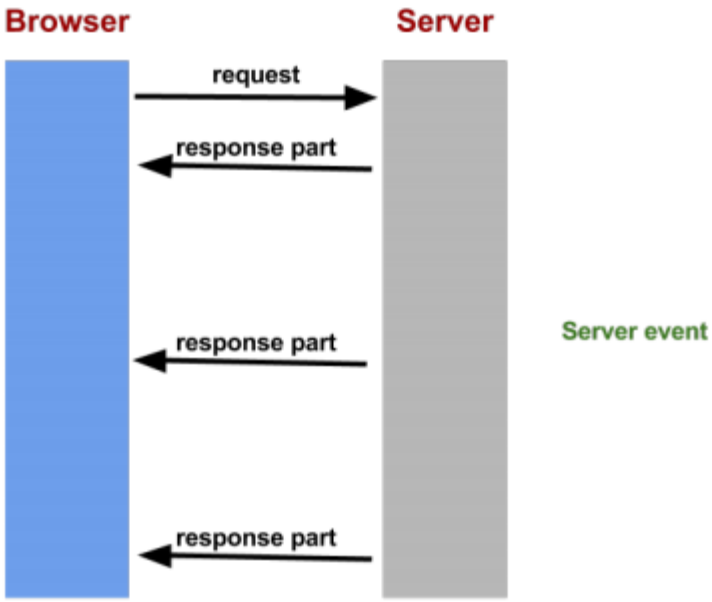
Očigledno je ovo dobro rješenje ako se zna točan interval dostava poruka, jer se mogu sinkronizirati zahtjevi klijenta tako da se dogode samo kada su informacije dostupne na poslužitelju. Međutim podaci u stvarnom vremenu često nisu toliko predvidljivi, pa su nepotrebni zahtjevi neizbježni, a kao rezultat toga mnoge se veze otvaraju i zatvaraju bespotrebno u situacijama s niskim brojem poruka.

Uz dugotrajno anketiranje (*engl. long polling*), preglednik šalje zahtjev poslužitelju, a poslužitelj zadržava zahtjev otvoren na neko određeno vrijeme. Ako se obavijest primi u tom roku, odgovor koji sadrži poruka se šalje klijentu [14]. Ako obavijest nije primljena u zadanom vremenskom razdoblju, poslužitelj šalje odgovor za prekid otvorenog zahtjeva. Kada se radi o velikom broju poruka, dugotrajno anketiranje ne pruža značajnije poboljšanje performansi u odnosu na tradicionalno anketiranje. Na Slici 3 je prikazana komunikacija klijenta i servera koristeći HTTP dugotrajno anketiranje [15].



Slika 3 HTTP dugotrajno anketiranje [11]

Strujanjem (*engl. streaming*) preglednik šalje potpuni zahtjev dokle poslužitelj šalje i održava otvoren odgovor koji se stalno ažurira i ostaje otvoren u nekom određenom vremenskom razdoblju ili neograničeno dugo. Odgovor se ažurira kad god je poruka spremna za slanje pri čemu poslužitelj nikada ne signalizira dovršen odgovor. Tako se održava veza otvorenom za isporuku budućih poruka. Strujanje je prikazano na Slici 4.



Slika 4 Strujanje [15]

Budući da je strujanje još uvijek inkapsuliran u HTTP-u, interventni vatrozidi i proxy poslužitelji mogu izabrati međuspremnik za odgovor što povećava potrebno vrijeme za isporuku poruka. Stoga se mnoga rješenja za strujanje vraćaju na dugotrajno anketiranje u slučaju da se otkrije proxy poslužitelj za spremanje u međuspremnik.

Alternativno, TLS (SSL) veze se mogu koristi kako bi se zaštitio odgovor od slanja u međuspremnik. U tom slučaju postavljanje i micanje tih veza povećava opterećenje na resurse koji su dostupni poslužitelju.

U konačnici sve ove metode za pružanje podataka u stvarnom vremenu uključuju HTTP zaglavlja zahtjeva i odgovora koji sadrže puno dodatnih nepotrebnih podataka unutar zaglavlja koji dovode do usporenije komunikacije između klijenta i poslužitelja. Povrh toga, obo smjerno (*engl. full-duplex*) povezivanje zahtijeva više od nizvodne veze od poslužitelja do klijenta. U nastojanju da se simulira obosmjerna komunikacija putem

polusmjernog (*engl. half-duplex*) HTTP-a, mnoga današnja rješenja koriste dvije veze - jednu za nizvodnu i jednu za uzvodnu komunikaciju. Održavanje i koordinacija ove dvije veze donosi značajne troškove u smislu potrošnje resursa [20].

4.2. WebSocket protokol

WebSocket protokol omogućuje dvosmjernu komunikaciju između klijenta, koji pokreće nepouzdani kod u kontroliranom okruženju, na udaljenog poslužitelja koji se uključio u komunikaciju preko tog koda. Sigurnosni model koji se za to koristi je na strani web preglednika. Protokol se sastoji od otvaranja rukovanja, a zatim slijedi osnovno uokvirivanje poruke slojevito preko TCP-a. Cilj ove tehnologija je osigurati mehanizam za web pregledničke aplikacije koje trebaju dvosmjernu komunikaciju s poslužiteljima koje se ne oslanjaju na otvaranje više HTTP veza [21].

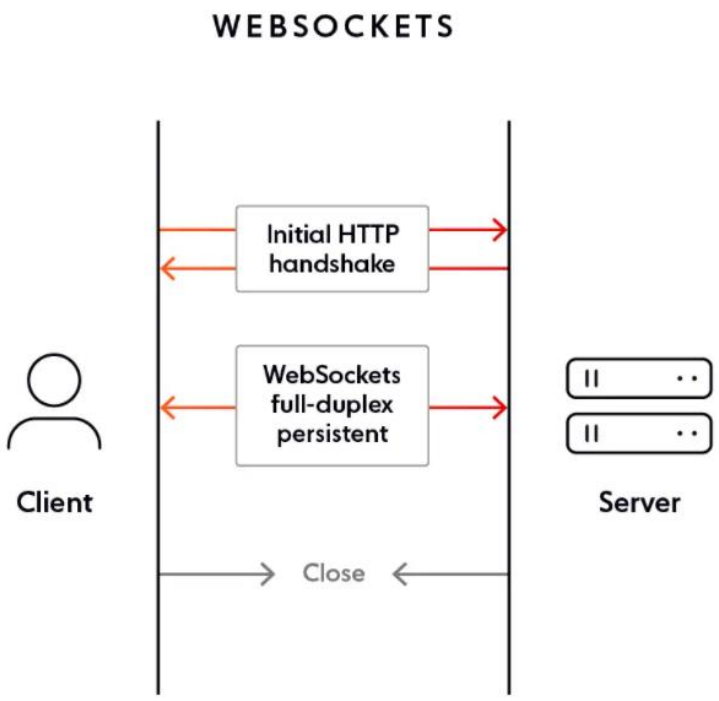
WebSocket protokol je dizajniran da zamijeni postojeće dvosmjerne komunikacijske tehnologije koje koriste HTTP kao prijenosni sloj kako bi iskoristile postojeću infrastrukturu (proxy, filtriranje, ovjera). Takve tehnologije implementirane su kao kompromisi između učinkovitosti i pouzdanosti jer HTTP u početku nije bio namijenjen za dvosmjernu komunikaciju. Prednost WebSocket protokola je to što oni ne dovode do sljedećih problema:

- Poslužitelj je prisiljen koristiti niz različitih TCP veze za svakog klijenta. Prema tome potrebno je napraviti jednu vezu za slanje informacija klijentu i za svaku dolaznu poruku novu vezu.
- Žičani protokol (*engl. wire protocol*) - svaka klijent-poslužitelj poruka ima HTTP zaglavlje.
- Skripta na strani klijenta mora održavati mapiranje iz odlaznih veza na dolazne veze kako bi pratila odgovore.

Konceptualno, WebSocket je zapravo samo sloj iznad TCP-a koji radi sljedeće [21]:

- Dodaje sigurnosni model web-podrijetla za preglednike;
- Dodaje mehanizam za adresiranje i protokole imenovanja za podršku više usluga na jednom portu i više klijenata na jednoj IP adresi;
- Dodaje sloj mehanizmu za uokvirivanje na vrhu TCP-a kako bi se vratio na IP paketni mehanizam na kojem je izgrađen TCP, ali bez ograničenja duljine;
- Uključuje dodatno rukovanje zatvaranja. Ono je dizajnirano za rad uz prisutnost proxy-a i drugih posrednika.

WebSocket omogućuje obosmjernu (*engl. full-duplex*) komunikaciju između klijenta i poslužitelja što omogućuje lak prijenos podataka i prema potrebi, za razliku od mehanizma prozivanja gdje se stalno poziva poslužitelj u intervalima kako bi se provjerilo ima li promjena [13]. Na Slici 5 je prikazan rad WebSocket-a.



Slika 5 WebSocket [11]

Kako bi se pokrenula komunikacija putem WebSocket-a prvo se treba napraviti HTTP rukovanje. WebSocket protokol je trebao biti objavljen u već postojećoj web infrastrukturi. Stoga je dizajniran da bude kompatibilan unatrag. Prije nego što WebSocket komunikacija može započeti, mora se pokrenuti HTTP veza. Preglednik šalje zaglavlje za nadogradnju poslužitelju kako bi ga obavijestio da želi pokrenuti WebSocket vezu. Prebacivanje s HTTP protokola na WebSocket protokol se naziva rukovanje. Nakon završetka rukovanja, WebSocket veza je aktivna te klijent ili poslužitelj mogu slati podatke.

Podaci su sadržani u okvirima. Svaki okvir je unaprijed fiksiran s četiri do dvanaest bajtova kako bi se osiguralo da se poruka može rekonstruirati. Nakon što se poslužitelj i preglednik dogovore o početku WebSocket komunikacije, postavlja se prvi zahtjev za početak ethernet komunikacije nakon čega slijedi zahtjev za TCP/IP komunikaciju [15].

Internet se temelji na dva protokola transportnog sloja - User Datagram Protocol (UDP) i Transmission Control Protocol (TCP). Oba koriste uslugu mrežnog sloja koju pruža internetski protokol (IP).

TCP je pouzdan protokol za prijenos podataka. Podaci se pohranjuju u segmentima bajt po bajt i prenose se prema određenim mjeračima vremena. Ova kontrola protoka osigurava konzistentnost podataka. Za TCP se kaže da je orijentiran na tok jer se podaci šalju u neovisnim segmentima.

UDP je nepouzdan ali brz protokol koji ne jamči da će podaci biti dostavljeni u svojoj cjelovitosti ili biti duplicirani. UDP radi na strategiji najboljeg napora bez kontrole protoka. Svaki segment podatka se prima nezavisno- To je protokol orijentiran na poruke.

WebSocket je izgrađen preko TCP-a zbog svoje pouzdanosti. Kako bi se postigla niska latencija, komunikacijski protokol mora paziti da ne izgubi niti jedan paket inače razmjena traje dva puta duže. WebSocket protokol se oslanja na nekoliko drugih protokola - HTTP za inicijalizaciju komunikacije, ethernet, TCP/IP i konačno TLS u slučaju da su potrebne sigurne veze [15].

5. React

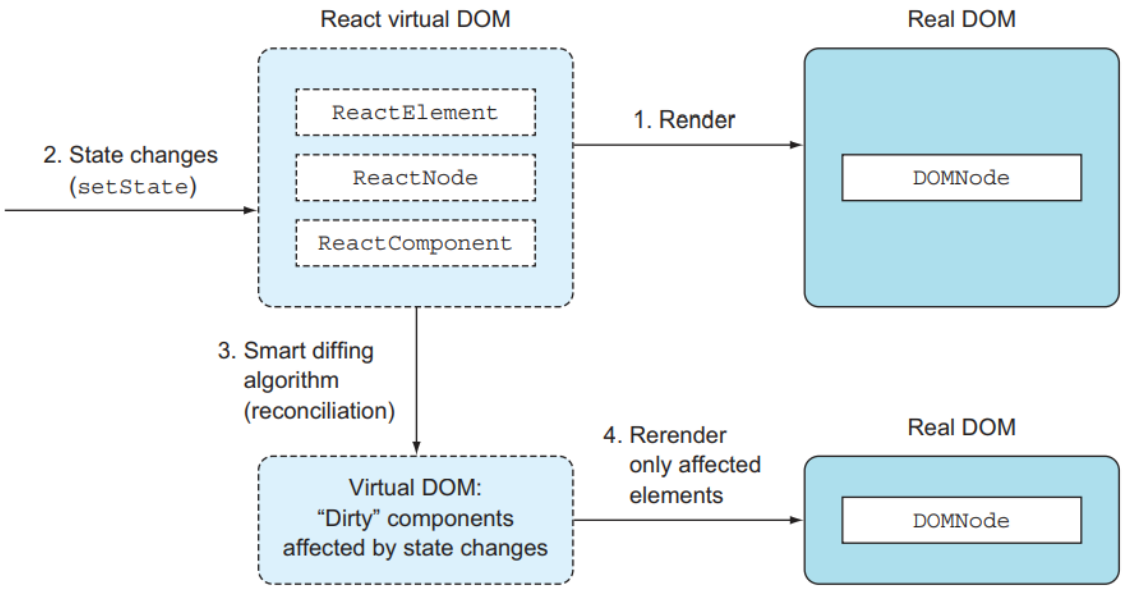
React, također poznat kao React.js ili ReactJS je besplatna JavaScript biblioteka otvorenog koda za izgradnju korisničkog sučelja temeljenim na komponentama korisničkog sučelja. React se može koristiti kao baza u razvoju jednostraničnih ili mobilnih aplikacija. Međutim, React se bavi samo upravljanjem stanja i prikazivanjem tog stanja u DOM-u, tako da stvaranje React aplikacija obično zahtijeva korištenje dodatnih biblioteka za usmjeravanje i za određene funkcionalnosti na strani klijenta [16].

Komponente korisničkog sučelja React-a su samostalni blokovi funkcionalnosti koji se odnose na specifične probleme. Na primjer, postoje komponente za birač datuma, captcha, adresa i elemente poštanskog broja. Takve komponente imaju vizualni prikaz i dinamičku logiku. Neke komponente mogu čak i samostalno razgovarati s poslužiteljem. Na primjer, komponenta za automatsko dovršavanje može dohvatiti popis za automatsko dovršavanje s poslužitelja. Arhitektura temeljena na komponentama CBA (*engl. component based architecture*), gdje su web komponente samo jedna od najnovijih implementacija CBA, postojala je prije React-a. Takve arhitekture općenito je lakše ponovno koristiti, održavati i proširivati nego monolitna korisnička sučelja. Ono što React omogućava je korištenje čistog JavaScripta (bez predložaka) i novi način gledanja na sastavljanje komponenti [17].

Sljedeći popis ističe neke od prednosti React-a u odnosu na druge biblioteke i okvire:

- Jednostavnije aplikacije - React ima CBA s čistim JavaScriptom koji je deklarativnog stila i jake DOM apstrakcije prilagođene programerima (i to ne samo DOM, već i iOS, Android i drugi).
- Brzo korisničko sučelje - React pruža izvanredne performanse zahvaljujući svom virtualnom DOM-u i algoritmu pametnog pomirenja koji omogućuje izvođenje i testiranje bez ponovnog pokretanja preglednika. Slika 6 na pojednostavljeni način prikazuje kako algoritam radi kad dođe do promjene u podacima.

- Manje koda za pisanje – React-ova zajednica i baza komponenti pružai programerima razne biblioteke i komponente.



Slika 6 pregled kako radi React-ov virtualni DOM [17]

Neki nedostaci React-a:

- Programeri trebaju React upariti s bibliotekom kao što je Redux ili React Router kako bi se postigla funkcionalnost usporediva s Angular-om ili Ember-om. Ovo također može biti prednost ako je potrebna minimalistička UI biblioteka.
- React nije potpun kao drugi okviri. React-ov osnovni API se još uvijek mijenja iako vrlo malo nakon izdanja 0.14. Najbolje prakse za React (kao i ekosustav komponenti i dodataka) se još uvijek razvijaju.
- React koristi donekle novi pristup web razvoju, te JSX i Flux (često se koristi s React-om kao biblioteka podataka) može biti zastrašujuća za početnike. Nedostaju najbolji primjeri iz prakse, knjige, tečajevi i resursi za savladavanje React-a.
- React ima samo jednosmjernu vezu. Iako je jednosmjerno uvezivanje bolje za složene aplikacije i uklanja dosta složenosti, neki su programeri navikli na dvosmjerno uvezivanje pa im takav način uvezivanja može predstavljati problem.
- React nije reaktivan (kao u reaktivnom programiranju i arhitekturi) izvan okvira. Programeri moraju koristiti druge alate kao što su Reactive Extensions (RxJS) za sastavljanje asinkronih tokova podataka s Observables.

6. MongoDB

MongoDB je baza podataka dokumenata koja je skalabilna i fleksibilna te s upitima i indeksiranjem [18]. MongoDB je program za upravljanje NoSQL bazom podataka otvorenog koda. NoSQL se koristi kao alternativa tradicionalnim relacijskim bazama podataka. NoSQL baze podataka su vrlo korisne za rad s velikim skupovima distribuiranih podataka. MongoDB je alat koji može upravljati informacijama orijentiranim na dokumente, pohranjivati ili dohvaćati informacije. MongoDB podržava različite oblike podataka. To je jedna od mnogih tehnologija nerelacijskih baza podataka koje su se pojavile sredinom 2000-ih pod NoSQL zastavom. Obično za upotrebu u aplikacijama velikih podataka (engl. *big data*) i drugih obrada koje uključuju podatke koji se ne uklapaju dobro u kruti relacijski model. Umjesto korištenja tablica i redaka kao u relacijskim bazama podataka, MongoDB arhitektura se sastoji od zbirki i dokumenata [19].

Glavne značajke MongoDB su [18]:

- MongoDB pohranjuje podatke u fleksibilne dokumente nalik JSON-u, što znači da se polja mogu razlikovati od dokumenta do dokumenta, a struktura podataka se može mijenjati tijekom vremena.
- Model dokumenta preslikava se na objekte u kodu aplikacije, što olakšava rad s podacima.
- Ad hoc upiti, indeksiranje i združivanje u stvarnom vremenu pružaju moćne načine za pristup i analizu podataka.
- MongoDB je distribuirana baza podataka, tako da su visoka dostupnost, horizontalno skaliranje i geografska distribucija ugrađeni i jednostavni za korištenje.
- MongoDB je besplatan za korištenje. Verzije objavljene prije 16. listopada 2018. objavljuju se pod licencom AGPL. Sve verzije objavljene nakon 16. listopada 2018., uključujući popravke zakrpa za prethodne verzije, objavljene su pod Server Side Public License (SSPL) v1.

MongoDB koristi zapise koji se sastoje od dokumenata koji sadrže strukturu podataka sastavljenu od parova polja i vrijednosti. Dokumenti su osnovna jedinica podataka u MongoDB-u. Dokumenti su slični JavaScript objektnoj notaciji, ali koriste varijantu koja se zove binarni JSON (BSON). Prednost korištenja BSON-a je da prihvaća više tipova podataka [19]. Polja u ovim dokumentima slični su stupcima u relacijskoj bazi podataka. Sadržane vrijednosti mogu biti različite vrste podataka, uključujući druge dokumente, nizove i nizove dokumenata [18]. Dokumenti će također sadržavati primarni ključ kao jedinstveni identifikator.

Skupovi dokumenata nazivaju se zbirke, koje funkcioniraju kao ekvivalent tablicama relacijskih baza podataka. Zbirke mogu sadržavati bilo koju vrstu podataka, ali ograničenje je da se podaci u zbirci ne mogu širiti po različitim bazama podataka.

7. Node.js

Node.js je višeplatformsko JavaScript okruženje otvorenog koda. Node.js pokreće V8 JavaScript stroj (engl. *engine*), jezgra Google Chrome, izvan preglednika. To omogućuje Node.js-u da bude vrlo učinkovit. Aplikacija Node.js radi u jednom procesu, bez stvaranja nove niti za svaki zahtjev. Node.js pruža skup asinkronih I/O primitiva u svojoj standardnoj biblioteci koja sprječava blokiranje JavaScript koda. Biblioteke su općenito u Node.js-u napisane korištenjem neblokirajućih paradigmi, što čini blokiranje iznimkom, a ne normom. Kada Node.js izvrši I/O operaciju, poput čitanja s mreže, pristupa bazi podataka ili datotečnom sustavu te umjesto da blokira nit i troši CPU cikluse na čekanje, Node.js će nastaviti s operacijama kada se odgovor vrati. To omogućuje Node.js-u rukovanje tisućama istodobnih veza s jednim poslužiteljem bez uvođenja tereta upravljanja istodobnošću niti, što bi moglo biti značajan izvor grešaka. U Node.js novi ECMAScript standardi mogu se koristiti bez problema jer se ne mora čekati ažuriranje preglednika korisnika [24].

Node.js je sličan po dizajnu i pod utjecajem sustava poput Ruby's Event Machine i Python's Twisted. Model događaja u Node-js-u predstavlja petlju događaja kao vrijeme izvođenja konstrukcije umjesto kao biblioteku. U drugim sustavima uvijek postoji blokirajući poziv za pokretanje petlje događaja. Obično se ponašanje definira povratnim pozivima na početku skripte, a na kraju se poslužitelj pokreće putem blokirajućeg poziva. Node.js jednostavno ulazi u petlju događaja nakon izvršavanja ulazne skripte, a izlazi iz petlje događaja kada više nema povratnih poziva za izvođenje. Ovo ponašanje je poput JavaScripta preglednika, petlja događaja je skrivena od korisnika [23].

Budući da Node.js koristi JavaScript na poslužitelju to nosi i druge pogodnosti [22]:

- Programeri mogu pisati web aplikacije u jednom jeziku, što pomaže pri smanjenju prebacivanja konteksta između razvoja klijenta i poslužitelja s tim da dopušta dijeljenje koda između klijenta i poslužitelja (primjerice poput ponovne upotrebe istog koda za provjeru valjanosti obrasca ili logike igre).
- JSON je danas vrlo popularan format za razmjenu podataka i izvorni je JavaScript.

- JavaScript je jezik koji se koristi u raznim NoSQL bazama podataka (kao što su CouchDB i MongoDB), tako da je sučelje s njima prirodno (na primjer, jezik ljuske i upita MongoDB-a je JavaScript).
- JavaScript je rezultat kompiliranja, a postoji niz jezika koji se već kompiliraju u njega.
- Node.js koristi jedan virtualni stroj (V8) koji drži korak sa standardom ECMAScript.

Node.js je vrlo lagan na I/O, dobar je u miješanju (engl. *shuffling*) ili proxy-anju podataka s jedne cijevi na drugu. Omogućuje poslužitelju da drži otvorene brojne veze dok obrađuje mnoge zahtjeve pri čemu zadržava mali memorijski otisak. Dizajniran je da reagira poput preglednika. Bilo da se radi o postojećim aplikacijama ili o potpuno novim vrstama aplikacija. Nove vrste web aplikacija zahtijevaju platforme koje mogu gotovo trenutno odgovoriti velikom broju istodobnih korisnika. Node.js je dobar u tome i to ne samo za web aplikacije već i za druge I/O aplikacije [22].

8. Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud (Amazon EC2) je web usluga koja pruža sigurni računalni kapacitet promjenjive veličine u oblaku. Dizajniran je kako bi programerima olakšao na web skali računalstvo u oblaku. Jednostavno sučelje usluge Amazon EC2 omogućuje dobivanje i konfiguriranje kapaciteta uz minimalni napor. Pruža potpunu kontrolu nad dodijeljenim računalnim resursima i omogućuje rad na Amazon-ovom dokazanom računalskom okruženju [25]. Amazon Elastic Compute Cloud dio je Amazon.com platforme za računalstvo u oblaku, Amazon Web Services (AWS), koja korisnicima omogućuje iznajmljivanje virtualnih računala na kojima se mogu pokretati vlastite računalne aplikacije. Amazon EC2 potiče skalabilnu implementaciju aplikacija pružanjem web usluge putem koje korisnik može pokrenuti Amazon Machine Image (AMI) kako bi konfigurirao virtualni stroj koji Amazon naziva "instanca". On sadrži bilo koji softver po želji. Korisnik može kreirati, pokretati i prekidati instance poslužitelja prema potrebi, plaćajući za sekunde za aktivne poslužitelje odakle dolazi pojam "elastičan". Amazon EC2 korisnicima pruža kontrolu nad zemljopisnom lokacijom instanci koja omogućuje optimizaciju kašnjenja i visoku razinu redundancije [26].

9. Aplikacija za razmjenu trenutnih poruka

Aplikacija za razmjenu trenutnih poruka je izrađena korištenjem React-a, Node.js, WebSockets i MongoDB. Za izradu servera korišten je Socket.IO što je Javascript biblioteka za web aplikacije koja omogućuje komunikaciju u stvarnom vremenu između klijenata i poslužitelja. Za postaviti HTML web stranicu koja daje obrazac i popis poruka koristiti se Express što je Node.js web okvir. Aplikacija je podijeljena na klijent i server dio.

9.1. Klijent

Klijent sadrži datoteke Chat.js, Login.js, Register.js, App.js, indeks.js i socket.js gdje se uključuje Socket.IO client biblioteka.

9.1.1. Indeks.js

Indeks.js je zadužen za prikaz aplikacije

```
ReactDOM.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>,
    document.getElementById("root")
);
```

9.1.2. App.js

App.js sadrži rute koje se koriste za navigaciju po aplikaciji. Definirane rute su:

- / - glavna ruta dovodi do početne stranice (*Login*)
- */Register* - vodi do ekrana za registraciju
- */Chat* - ruta vodi do ekrana za chat

```
function App() {
```

```
    return (
        <Router className="App">
            <Routes>
                <Route path="/" element={<Login />} exact />
                <Route path="/Register" element={<Register />} />
                <Route path="/Chat" element={<Chat />} />
                <Route element={Error} />
            </Routes>
        </Router>
    );
}
```

9.1.3. Login.js

Login.js je zadužen za početni ekran na kojem se nalazi registracija i prijava. Sastoji se od obrasca koji sadrži polja za unos email adrese i lozinke te linka za registraciju.

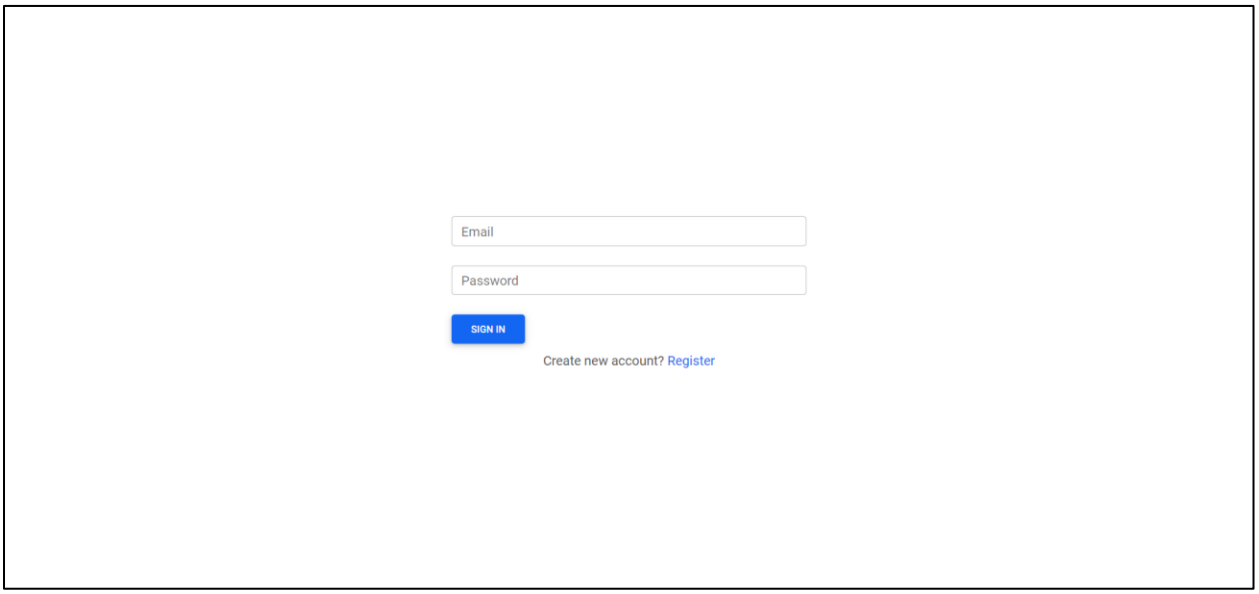
```
export default function Login() {
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const navigate = useNavigate();

    useEffect(() => {
        socket.on("login", (state) => {
            localStorage.setItem("sessionToken", state);
            localStorage.setItem("myEmail", email);
            if (state !== "") {
                navigate("/Chat");
                console.log("Email, password correct");
            } else {
                alert("Email, password incorrect");
                navigate("/");
            }
        });
    }, [navigate, email]);
```

Funkcija *useEffect* se koristi kako bi za svaku promjenu stanja koje ovisi o konstantama *email* i *navigate* dogodilo renderiranje, koje zatim pokreće efekt preusmjerena na chat ili na registraciju. Ukoliko je pogrešna zaporka ili email adresa, prikazuje se greška. Također, lokalno pohranjuje *myEmail* (email korisnika) i prima sa servera *sessionToken* kojeg pohranjuje.

```
    function handleSubmit(e) {
        e.preventDefault();
        socket.emit("login", email, password);
    }
```

Funkcija *handleSubmit* je zadužena za predaju obrasca i slanje unesenih podataka serveru kako bi se prijava mogla izvršiti. Na Slici 7 prikazan je početni ekran odnosno ekran za prijavu.



Slika 7 Početni ekran (prijava)

9.1.4. Register.js

Registracija se sastoji o obrasca koji sadrži polja za unos imena, prezimena, email adrese i lozinke te gumba za povratak na prijavu.

```
export default function Register() {
    const [username, setUsername] = useState("");
    const [surname, setSurname] = useState("");
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const navigate = useNavigate();

    useEffect(() => {
        socket.on("createNewUser", (status) => {
            if (status) {
                console.log("Account registered");
                navigate("/");
            } else {
```

```
                alert("Email is already in use");
                navigate("/Register");
            }
        });
    }, [navigate]);
```

Funkcija *useEffect* pokreće efekt preusmjeravanja na prijavu ukoliko je obrazac ispravno ispunjen ili vraća na registraciju. Ukoliko je uneseni email već korišten za korisnički račun ispisuje se obavijest da je taj email u upotrebi.

```
    function handleSubmit(e) {
        e.preventDefault();
        socket.emit("createNewUser", username, surname, email, password);
        console.log("Submit");
    }
```

Funkcija *handelSubmit* je zadužena za predaju obrasca i emitiranje unesenih podataka serveru kako bi se oni pohranili u bazu. Slika 8 prikazuje izgled sučelja za registraciju.

Slika 8 Sučelje za registraciju

9.1.5. Chat.js

Chat se sastoji od prikaza korisnika, poruka između korisnika i polja za unos poruka te gumba za odjavu.

```
export default function Chat() {
    const sessionTokenLS = localStorage.getItem("sessionToken");
    const localMyEmail = localStorage.getItem("myEmail");
    const navigate = useNavigate();
    const [myEmail, setMyEmail] = useState("");
    const [contacts, setContacts] = useState([]);
    const [activeButton, setActiveButton] = useState("");
    const [message, setMessage] = useState("");
    const [prevMessages, setPrevMessages] = useState([]);
    const intervalRef = useRef(0);

    const setRef = useCallback((node) => {
        if (node) {
            node.scrollIntoView({ smooth: true });
        }
    }, []);
```

Funkcija *setRef* služi kako bi se pri unosu poruke tekst pomakao prema dolje po stranici ukoliko se sve poruke ne mogu prikazati.

```
useEffect(() => {
        if (myEmail) {
            intervalRef.current = setInterval(() => {
                socket.emit("setActive", myEmail);
            }, 3000);
        }
        return () => {
            clearInterval(intervalRef.current);
        };
    }, [myEmail]);
```

Prvi *useEffect* šalje na server korisnikov email svake 3 sekunde.

```
    useEffect(() => {
        if ((sessionTokenLS === null) | (localMyEmail === null)) {
            alert("No session token or email");
            navigate("/");
        }
        if (localMyEmail !== null) {
            setMyEmail(localMyEmail);
        }

        socket.emit("getContacts", localMyEmail);
        socket.on("getContacts", (c) => {
            setContacts(c);
        });
```

```
        socket.on("getMessages", (c) => {
            setPrevMessages(c);
        });

        socket.on("chatMessage", (msg) => {
            if (msg.from !== activeButton) {
                setPrevMessages([
                    ...prevMessages,
                    {
                        message: msg,
                        from: activeButton,
                        to: myEmail,
                    },
                ]);
            }
        });
    }, [
        localMyEmail,
        sessionTokenLS,
        navigate,
        prevMessages,
        activeButton,
        myEmail,
    ]);
```

Ovaj *useEffect* provjerava jesu li *sessionToken* i *myEmail* lokalno pohranjeni. Ukoliko nisu, vraća na početnu stranicu. Također emitira *myEmail* serveru kako bi dobio od servera kontakte koji se zatim spremaju, dohvaća poruke sa servera te prikazuje nove poruke koje dolaze.

```
function handelSubmit(e) {
        e.preventDefault();

        socket.emit("chatMessage", message, myEmail, { to: activeButton });
        setMessage("");
        setPrevMessages([
            ...prevMessages,
            {
                message,
                from: myEmail,
                to: activeButton,
            },
        ]);
    }
```

Funkcija *handelSubmit* emitira poruku serveru i čisti područje za unos teksta u kojem se piše poruka.

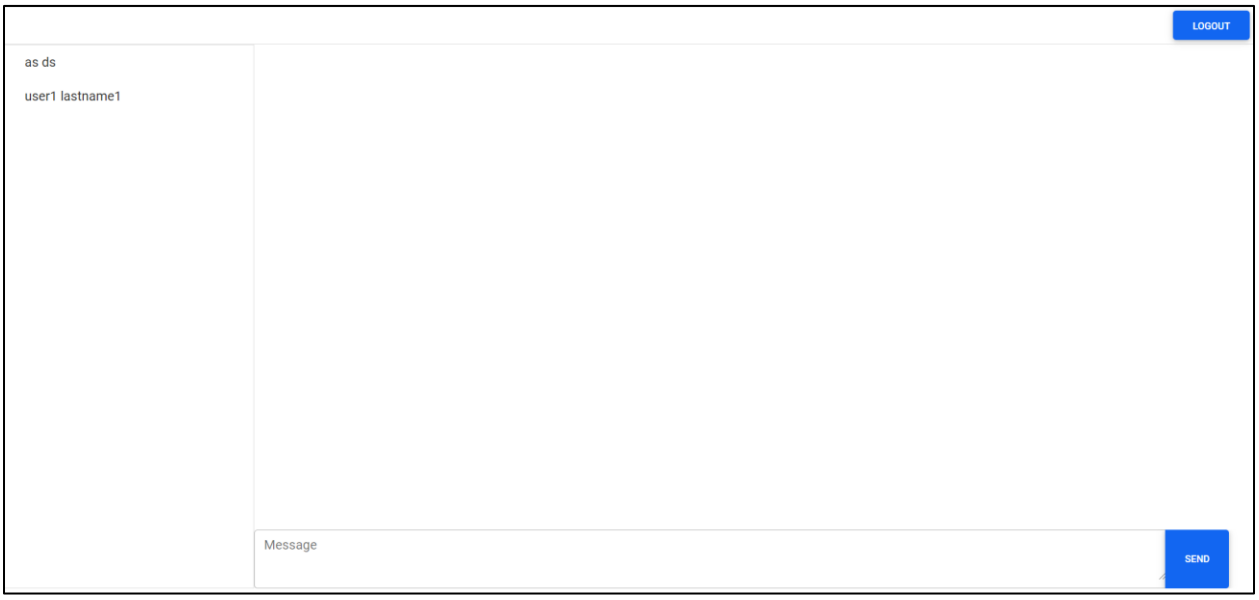
```
const renderContactsList = () => {
        let contactItems = [];
        for (let contact of contacts) {
            contactItems.push(
                <MDBListGroup flush key={contact.email}>
                    <MDBListGroupItem
                        key={contact.email}
                        action
                        active={activeButton === contact.email}
                        onClick={() => {
                            setActiveButton(contact.email);
                            socket.emit("getMessages", myEmail, contact.email);
                        }}
                    >
                        {contact.username} {contact.surname}
                    </MDBListGroupItem>
                </MDBListGroup>
            );
        }
        return contactItems;
    };
```

Funkcijom *renderContactsList* se prikazuje lista kontakata te šalje serveru korisnikov email i email kontakta odnosno selektiranog kontakta.

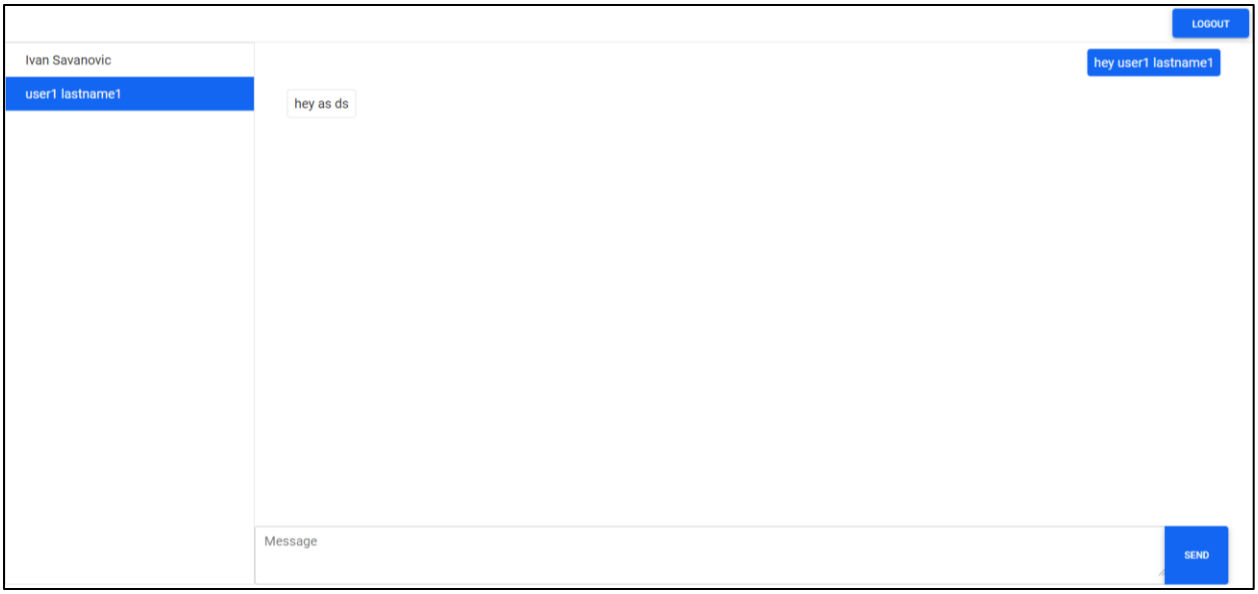
```
 Const renderConversation = () => {
        let messageList = [];
        let I = 0;
        const lastMessage = messageList.length === i;
        for (let msg of prevMessages) {
            messageList.push(
                <MDBListGroup flush key={i} style={{}}>
                    <MDBListGroupItem
                        className={`${
                            msg.from === myEmail
                                ?“"align-self-end align-items-en”"
                                :“"align-items-start align-self-star”"
                        }`}
                        id”"message”"
                        ref={lastMessage ? setRef : null}
                    >
                        <MDBListGroup
```

```
                        className={`rounded px-2 py-1 ${
                            msg.from === myEmail
                                ?“"bg-primary text-whit”"
                                :“"borde”"
                        }`}
                    >
                        {msg.message}
                    </MDBListGroup>
                </MDBListGroupItem>
            </MDBListGroup>
        );
        i++;
    }
    return messageList;
};
```

Funkcija *renderConversation* prikazuje postojeće poruke između dva korisnika i sve novo dodane poruke u razgovoru. Slika 9 prikazuje chat nakon uspješne prijave. Na Slici 10 je prikazan chat s otvorenim razgovorom.



Slika 9 Chat



Slika 10 Chat s otvorenim razgovorom

9.2. Server

Server sadrži Schemas.js i server.js datoteke.

Schemas datoteka sadrži mongo sheme *userSchema*, *sessionSchema* i *messageSchema*. Mongo shema je JSON objekt koji definira strukturu i sadržaj podataka.

9.2.1. Schemas.js

```
const userSchema = new mongoose.Schema({
    username: {
        type: String,
    },
    surname: {
        type: String,
    },
    email: {
        type: String,
        unique: true,
    },
    password: String,
});
```

Shema *userSchema* sadrži *username* i *surname* (korisničkovo ime i prezime), email koji mora biti jedinstven jer se koristi kao jedinstveni identifikator i *password* (korisnikova lozinka).

```
const sessionSchema = new mongoose.Schema({
    email: {
        type: String,
        unique: true,
    },
    sessionToken: {
        type: String,
        unique: true,
    },
});
```

Shema *sessionSchema* sadrži email koji je jedinstven i *sessionToken* koji je također jedinstven za svakog korisnika.

```
const messageSchema = new mongoose.Schema({
    message: {
        type: String,
    },
    from: {
        type: String,
    },
    to: {
        type: String,
    },
});
```

Shema *messageShema* se sastoji od *message* (poruke), *from* (pošiljateljev email) i *to* (email kome se šalje poruka).

```
exports.createNewUser = async (username, surname, email, password) => {
    const User = mongoose.model("User", userSchema);
    const c = new User({
        username: username,
        surname: surname,
        email: email,
        password: password,
    });
    console.log(c);
    await c.save();
    console.log("Saved");
};
```

Funkcijom *createNewUser* se dodaje novog korisnika u bazu.

```
Exports.getUserByEmail = async (email) => {
    const User = mongoose.model(“User”, userSchema);
    const getEmail = await User.find(
        { email: email },
        { _id: 0, username: 0, surname: 0, __v: 0 }
    );
    return getEmail;
};
```

Za provjeru da je li korisnik već registriran koristi se funkcija *getUserByEmail*.

```
Exports.saveSessionToken = async (email, sessionToken) => {
    const Session = mongoose.model(“Session”, sessionSchema);
    const checkEmail = await Session.find(
        { email: email },
        { _id: 0, sessionToken: 0, __v: 0 }
    );
    if (!checkEmail.length) {
        const s = new Session({
            email: email,
            sessionToken: sessionToken,
        });
        await s.save();
        console.log(“Email not in use creating new entry: “, s);
    } else {
        await Session.deleteOne({ email: email });
        const s = new Session({
            email: email,
            sessionToken: sessionToken,
        });
        await s.save();
        console.log(“Deleting existing email and creating new entry: “, s);
    }
};
```

Funkcijom *saveSessionToken* provjerava se postoji li *sessionToken* za dani email. Ako ne postoji, sprema email i *sessionToken*, inače briše iz baze email i *sessionToken* te pravi novi.

```
exports.getContacts = async (email) => {
    const User = mongoose.model("User", userSchema);
    try {
        const getContact = await User.find(
            { email: { $ne: email } },
```

```
            { _id: 0, password: 0, __v: 0 }
        );
        return getContact;
    } catch (error) {
        console.log(error);
    }
};
```

Pomoću funkcije *getContacts* dohvaćaju se svi kontakti.

```
exports.saveMessage = async (msg, from, to) => {
    const Message = mongoose.model("Message", messageSchema);
    try {
        const c = new Message({
            message: msg,
            from: from,
            to: to,
        });
        console.log(c);
        await c.save();
        console.log("Message saved");
    } catch (error) {
        console.log("Save message failed", error);
    }
};
```

Funkcija *saveMessage* sprema poruke.

```
Exports.getMessages = async (from, to) => {
    const Message = mongoose.model(“Message”, messageSchema);
    try {
        const getMsg = await Message.find(
            {
                $or: [
                    { $and: [{ from: from }, { to: to }] },
                    { $and: [{ from: to }, { to: from }] },
                ],
            },
            { __v: 0 }
        );
        return getMsg;
    } catch (error) {
        console.log(“Failed to get messages”, error);
    }
};
```

Poruke između dva korisnika se dohvaćaju funkcijom *getMessage.*

9.2.2. Server.js

```
const app = express();
app.use(express.static('build'))
const httpServer = createServer(app);
const io = require("socket.io")(httpServer, {
    cors: {
        origin: "*",
        methods: ["GET", "POST"],
    },
});
httpServer.listen(5000, () => {
    console.log("Listening on http://localhost:5000");
});
```

Express inicijalizira aplikaciju da bude rukovalac funkcijama koje se dostavljaju HTTP poslužitelju. HTTP poslužitelj sluša na portu 5000.

```
socket.on("setActive", email =>{
        contacts[email] = socket.id;
    })
```

Dobiveni email s klijenta postavlja kao *socket id*.

```
socket.on("login", async (email, password) => {
        try {
            let c = await mongo.getUserByEmail(email);
            console.log(c);
            if (!c.length) {
                socket.emit("login", "");
            } else {
                let isPasswordTheSame = await bcrypt.compare(
                    password,
                    c[0].password
                );
                console.log(isPasswordTheSame);
                if (isPasswordTheSame) {
                    let sessionToken = await jwt.sign(
                        { email, password },
                        "app555"
                    );
                    await mongo.saveSessionToken(email, sessionToken);
                    socket.emit("login", sessionToken);
                    contacts[email] = socket.id;
                    const c = await mongo.getContacts();
```

```
                    socket.emit("getContacts", c);
                } else {
                    socket.emit("login", "");
                }
            }
        } catch (error) {
            console.log(error);
            socket.emit("login", "");
        }
    });
```

Postavlja se upit bazi *getUserByEmail.* Ukoliko dobiveni email postoji, uneseni *password* (lozinka) se dekriptira koristeći bcrypt biblioteku te se dobivena lozinka uspoređuje s postojećim lozinkama. Pravi se *sessionToken* koristeći jwt biblioteku, sprema se u bazu, šalje te dohvaća kontakte iz baze i šalje klijentu.

```
socket.on("createNewUser", async (username, surname, email, password) => {
        try {
            const hashedPassword = await bcrypt.hash(password, 8);
            await mongo.createNewUser(username, surname, email, hashedPassword);
            socket.emit("createNewUser", true);
        } catch (error) {
            console.error(error);
            socket.emit("createNewUser", false);
        }
    });
```

Dobivena lozinka se hashira koristeći bcrypt biblioteku te se sprema s ostalim dobivenim podacima u bazu.

```
socket.on("getContacts", async (email) => {
        contacts[email] = socket.id;
        const c = await mongo.getContacts(email);
        socket.emit("getContacts", c);
    });
```

Postavlja se upit bazi koji vraća kontakte te ih šalje klijentu.

```
socket.on("getMessages", async (from, to) => {
        const c = await mongo.getMessages(from, to);
        console.log(c);
        socket.emit("getMessages", c)
    });
```

Postavlja se upit bazi koji vraća poruke te ih šalje klijentu.

```
    socket.on("chatMessage", async (msg, myEmail, to) => {
```

```
        io.to(contacts[to.to])
            .emit(
                "chatMessage",
                msg,
                myEmail,
                { from: myEmail },
            );
            console.log(to)
        await mongo.saveMessage(msg, myEmail, to.to);
    });
```

Dobivenu poruku sprema se u bazu.

10. Zaključak

U ovom radu je dan pregled komunikacije i arhitekture trenutnih poruka. Prikazan je razvoj trenutnih poruka od 1961. godine do modernih aplikacija za trenutne poruke. Objašnjeno je kako trenutne poruke rade i sigurnosni problemi koji proizlaze korištenjem trenutnih poruka. Opisana je HTTP komunikacija u stvarnom vremenu, kako se ona izvodi bez socket-a i sa socket-om te njihove prednosti i mane.

U praktičnom djelu prikazana je aplikacija za razmjenu trenutnih poruka. Opisani su važni elementi aplikacije te je prikazana regulirajuća aplikacija. Aplikacija je izrađena korištenjem React-a, Node.js, WebSockets i MongoDB.

11. Literatura

[1] Spahn, Michael, Dr. 9.10.2014. "Secure instant messaging."

[2] Vikas, Gupta, Avnish Dass, Gaurav Malhotra, and Pratul Katyal. 2002. *Instant Messaging Systems.* New York: Wiley Publishing, Inc.

[3] Goran D. Putnik, Maria Manuela Cunha. 2007. *Encyclopedia of Networked and Virtual Organizations.*

[4] 2021. *Wikipedia.* August 7. https://en.wikipedia.org/wiki/Instant_messaging.

[5] Bazara I. A. Barry, Fatma M. Tom. 2010. *Instant Messaging: Standards, Protocols, Applications, and Research Directions.* Nova Science Publishers, Inc.

[6] 2021. *Project 3: Instant Messaging.* August 7. http://web.mit.edu/6.005/www/fa08/projects/guichat/assignment.html.

[7] Simone Leggio, Tuomas Kulve, Oriana Riva, Jarno Saarto, Markku Kojo. 2004. *An Analysis of Instant Messaging and Email Access Protocol Behavior in Wireless Environment.* Helsinki: University of Helsinki, March 26.

[8] Larson, Gary W. 2021. August 7. https://www.britannica.com/topic/instant-messaging.

[9] 2021. August 7. https://www.uniassignment.com/essay-samples/information-technology/the-background-of-instant-messaging-information-technology-essay.php.

[10] Ivanova, Lora. 2021. August 7. https://www.brosix.com/blog/what-is-instant-messaging/.

[11] Fietkiewicz, Martin. 2021. August 7. https://ably.com/topic/websockets-vs-http.

[12] 2021. August 7. https://socket.io/.

[13] Chopra, Varun. 2015. *WebSocket Essentials - Building Apps with HTML5 WebSockets.* Birmingham: Packt Publishing.

[14] Lombardi, Andrew. 2015. *WebSocket.* O'Reilly Media.

[15] Muller, Gabriel L. 2014. *HTML5 WebSocket protocol and its application to distributed computing.* Cranfield: Cranfield University.

[16] 2021. *React_(JavaScript_library).* 10 21. https://en.wikipedia.org/wiki/React_(JavaScript_library)#cite_note-4.

[17] Mardan, Azat. 2017. *React_(JavaScript_library) Quickly Painless Web Apps with React, JSX, Redux and GraphQL .* Manning Publications Co.

[18] 2021. *MongoDB.* 10 21. https://www.mongodb.com/.

[19] 2021. *MongoDB.* 10 23. https://searchdatamanagement.techtarget.com/definition/MongoDB.

[20] Peter Lubbers, Brian Albers, Frank Salim. 2010. *Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development.* New York: Springer Science Business Media, LLC.

[21] I. Fette, A. Melkinov. 2011. *The websocket protocol.* Request for Comments, Internet Engineering Task Force Trust.

[22] Mike Cantelon, Marc Harter,T.J. Holowaychuk, Nathan Rajlich. 2014. *Node.js in Action.* Manning Publications Co.

[23] *nodejs.* Accessed 11 21, 2021. https://nodejs.org/.

[24] *nodejs.dev.* Accessed 11 21, 2021. https://nodejs.dev/.

[25] Amazon EC2. *Amazon.* Accessed 12 10, 2021. https://aws.amazon.com/ec2/?nc2=h_ql_prod_fs_ec2&ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc.

[26] Amazone Elastic Compute Cloud. *Wikipedia.* Accessed 12 11, 2021. https://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud.

[27] Instant Messaging Security. *Technical info.* Accessed 10 11, 2021. http://www.technicalinfo.net/papers/IMSecurity.html.