Sustav za komunikaciju u stvarnom vremenu

Jakab, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

SUSTAV KOMUNIKACIJE U STVARNOM VREMENU

Završni rad

Ivan Jakab

Osijek, 2019. godina

SADRŽAJ

1 UVOD

Komunikacija u stvarnom vremenu ima brojne aplikacije u računarstvu. Omogućuje korisnicima pregled podataka gotovo u isto vrijeme kada se oni promijene. U sustavima u kojima se promjene podataka događaju vrlo često, stvarnovremenska komunikacija smanjuje potrebu za ručnim osvježavanjem podataka.

Najčešća je potreba ostvariti komunikaciju između poslužiteljskog računala i više klijentskih računala. U takvom sustavu potrebno je obavijestiti poslužitelja o određenim događajima na nekom od klijentskih računala. Poslužitelj bi obradio dobivene podatke te bi u kontroliranim uvjetima obavijestio ostala klijentska računala.

U točki „Osnovni principi“ objašnjeni su neki protokoli komunikacijskih mreža koji omogućavaju komunikaciju u stvarnom vremenu i na kojima je izrađena demonstrativna aplikacija. Točka „Korištene tehnologije“ opisuje suvremene tehnologije koje se koriste u izradi aplikacije. Zatim, u točki „Korisničko sučelje“ detaljno će biti objašnjena implementacija svakog dijela aplikacije. Na samom kraju, zaključak daje sumu cijeloga rada.

1.1 Zadatak završnog rada

U radu će biti objašnjen sustav komunikacije u stvarnom vremenu na primjeru web aplikacije. Koristeći suvremene tehnologije i okvire, aplikacija će se povezati sposlužiteljem treće strane s kojim će komunicirati u stvarnom vremenu. Bit će izrađena chat aplikacija, koja omogućava kreiranje soba, slanje i primanje poruka, uređivanje i brisanje spomenutih i druge mogućnosti.

2 OSNOVNI PRINCIPI

Postoje različiti načini pomoću kojih različiti sustavi mogu implementirati komunikaciju u stvarnom vremenu. Ovaj se rad koncentrira na komunikaciju unutar TCP (Transmission Control Protocol) protokola. Potrebno je javiti poslužitelju kada se na klijentu dogodio nekakav događaj. Komunikacija u ovom smjeru nije izazov, riješena je već godinama, a najčešće se koristi HTTP protokol. Problem nastaje kada poslužitelj treba dati informaciju o nekom događaju klijentu. Zbog ovoga koristimo takozvane WebSockete.

2.1 TCP Konekcija

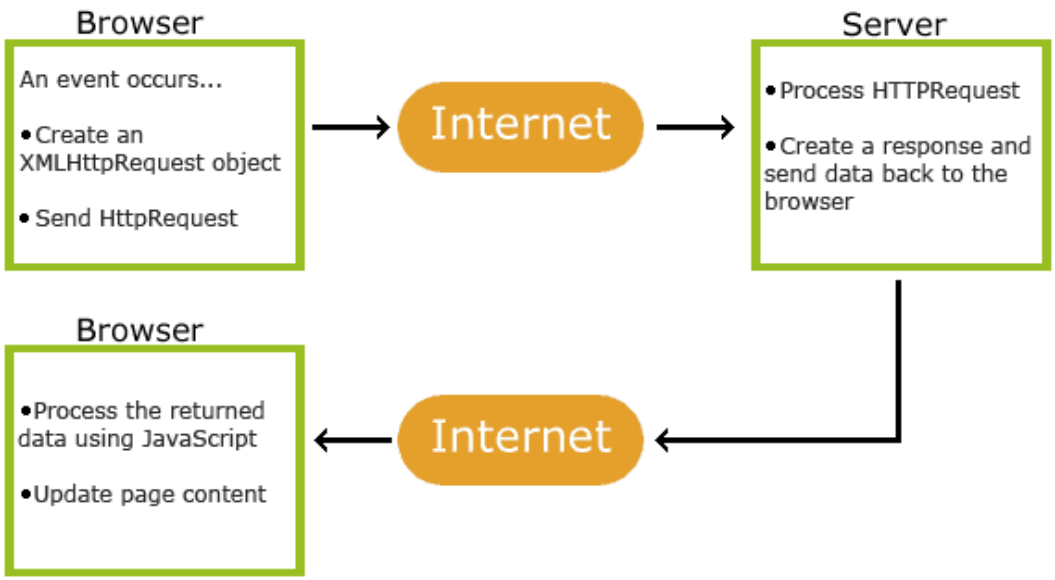
Transmission Control Protocol protokol je transportnog sloja komunikacijskih mreža. Definira način na koji računala šalju podatke, brine se da su paketi došli ispravno i u ispravnom redoslijedu, ima ugrađenu mehaniku potvrde, ponovnog slanja i mnoge druge mehanike.

[1]Klasični HTTP pristup jednako kao i WebSocket pristup koristi TCP u transportnom sloju. Razlika je u tome što WebSocketi održavaju TCP konekciju, a time omogućuju dvosmjernu komunikaciju. [2]No, to sa sobom donosi i mane – ponajviše mogućnost rasta (eng. scalability) jer je potrebno imati dobru arhitekturu za održavanje većeg broja WebSocket konekcija.

2.2 HTTP protokol

[3]Hyper Text Transfer Protocol najraširenija je protokol komunikacije na webu u aplikacijskom sloju. Koristi strukturu upit/odgovor (eng. request/response). Klijent daje upite poslužitelju – pri svakom upitu šalju se svi potrebni podaci koje poslužitelj pri svakom upitu provjerava. Sastoji se od url-a, ključne riječi, tzv. query stringa, zaglavlja i tijela. Ako su podaci u redu, poslužitelj odgovara zatraženim podacima – statusni kod, zaglavlje i tijelo.

[1] (The WebSocket Protocol, 2019)
[2] (10M Concurrent Websockets, 2019)
[3] (What is HTTP?, 2019)



Slika 2.1 HTTP protokol (What is HTTP?, 2019)

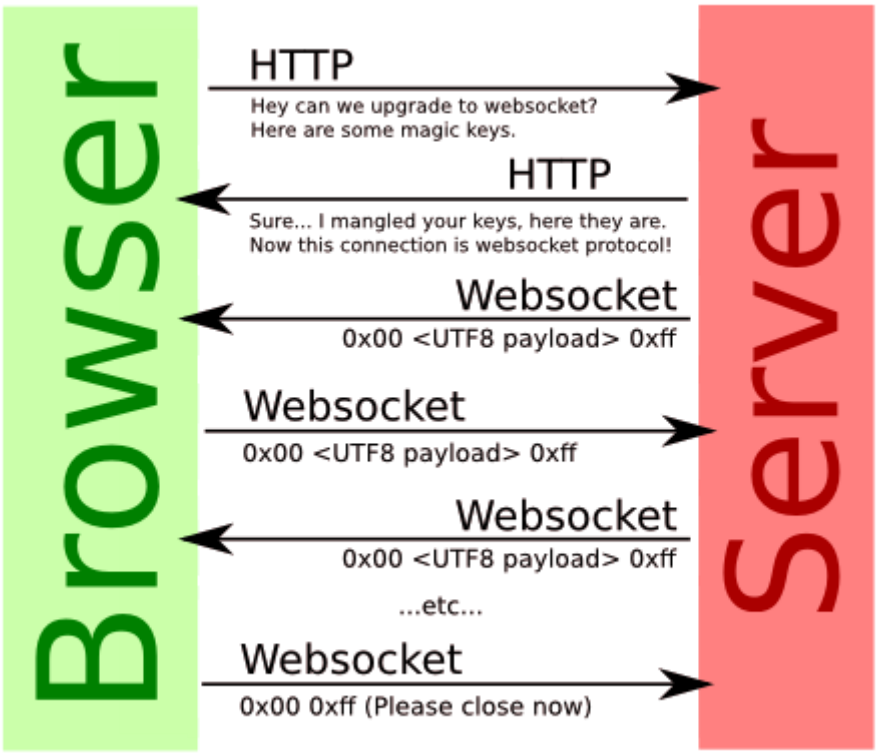
2.3 WebSocket

[4]WebSocketi primarno su uvedeni radi rješavanja problema koji se dogodi kada poslužitelj mora obavijestiti klijenta o nekakvom događaju. Jedan pristup tome bio bi da klijent konstantno šalje upite poslužitelju (eng. *pinga*) za provjeru ima li novih događanja. Takav je pristup nespretan, koristi puno računalnih resursa i može trošiti puno podataka ukoliko se korisnik spaja preko mobilne mreže.

WebSocketi održavaju konstantnu konekciju između poslužitelja i klijenta, a obje strane je mogu koristiti bilo kada. Proces započinje „rukovanjem“ (eng. *handshake*), što je zapravo HTTP upit prilikom kojega jedna od strana može i odbiti konekciju. Url na koji se upit šalje započinje s ws:// umjesto s http://

Nakon rukovanja, komunikacija može početi. Komunikacija se sastoji od UTF-8 podataka poslanih u oba smjera. Okviri i biblioteke za rad s WebSocketima obično daju nivo apstrakcije – razvojni programeri rade sa „sobama“ u koje se korisnik „pretplati“, a zatim u određenim sobama puštaju „događaje“ koje slušaju svi pretplaćeni na tu sobu. Uz događaje šalju se i podatkovni paketi ukoliko su oni potrebni.

[4] (An Introduction to WebSockets, 2019)



Slika 2.2 WebSocket životni vijek (PHP WebSocket server, 2019)

3 KORIŠTENE TEHNOLOGIJE

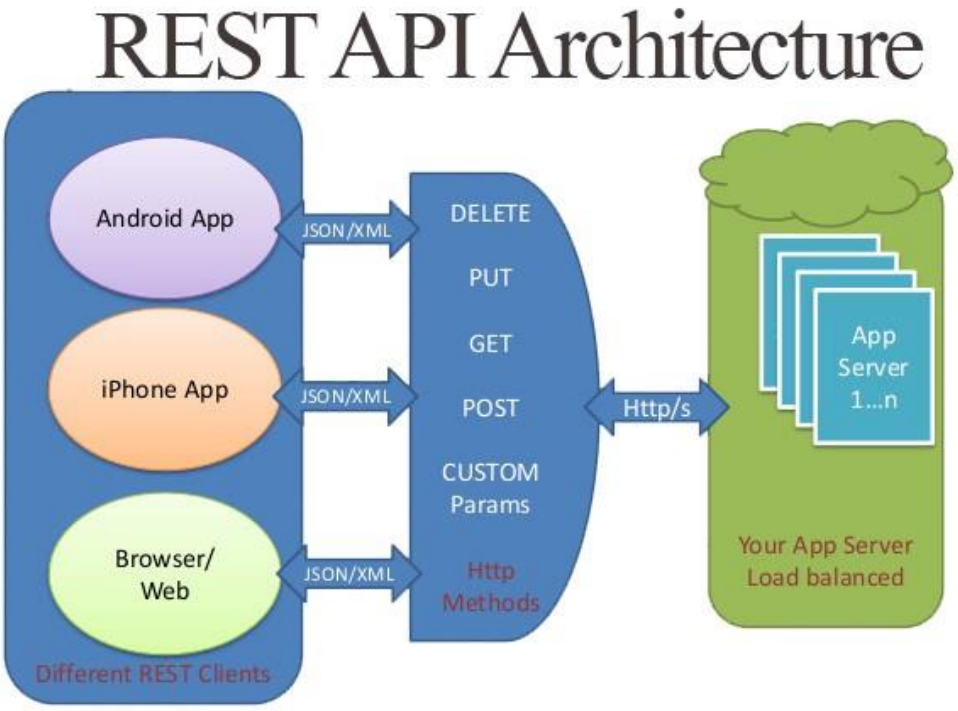
Za demonstraciju svih gore navedenih principa, u ovom se radu izrađuje aplikacija koja prikazuje izmjenjivanje poruka u stvarnom vremenu. Izrađeno je korisničko web sučelje koje se spaja na poslužitelj treće strane, te komunicira s njim preko HTTP upita i preko WebSocketa.

Web sučelje koristi HTML (Hyper text markup language) za prikaz svojih elemenata, CSS (Cascading Style Sheets) za stiliziranje i Javascript koji daje dinamiku. Korišten je okvir Vue.js zbog brojnih implementiranih pogodnosti s kojima dolazi. Također, korištena je i WebSocket biblioteka koja nam pruža apstraktno sučelje za spajanje na poslužitelj.

3.1 REST API

Tradicionalan pristup posluživanja sadržaja na webu bio je da internet preglednici šalju HTTP upite neposredno na glavni poslužitelj, koji im vraća HTML, CSS i Javascript, na osnovu kojih se korisniku prikazuje sučelje. Međutim, ponajviše zbog rasta mobilnih aplikacija, javlja se potreba za tzv. „agnosticizmom platforme“ – ideja da se jednom napisana logika može koristiti neovisno o platformi koja ju koristi, te da sve platforme koriste iste podatke. Dobar su primjer većina današnjih poznatih web aplikacija, koje imaju i mobilnu verziju – ona se logički ponaša kao i verzija na webu, prikazuje iste podatke, samo na drugoj platformi.

REST (Representational state transfer) API (Application programming interface) podijeljen je na logičke cjeline, a korisnička sučelja pristupaju mu preko HTTP upita i s njima najčešće komunicira u JSON (Javascript object notation) i/ili XML (EXtensible Markup Language) formatu. Tako omogućava da se „glavni“ poslužitelj, onaj koji ima pristup bazi podataka i koji provodi svu logiku, odvoji potpuno od aplikacija korisničkih sučelja. Tako i ova aplikacija neće biti poslužena od strane „glavnog“ poslužitelja, nego će od njega pokupiti podatke te ih prikazivati neovisno o njemu.



Slika 3.1 (Creating a simple REST API in PHP, 2019)

3.1.1 Gaussbox

Gaussbox je poslovno orijentirana SaaS (Software as a service) aplikacija koja pruža REST API za spajanje na nju. Za potrebe ovoga projekta, kreirana je demonstrativna licenca s demonstrativnim korisnicima te joj je dopušten pristup Chat modulu. Aplikacija izrađena u projektu spajat će se na ovaj poslužiteljski API, dok će se poslužitelj brinuti o čuvanju podataka, obavještavanju drugih klijenata i slično.

Korisnici su kreirani od strane vlasnika licence ili administratora, a prilikom prijave dobiva se pristupni podatak – digitalno potpisani JWT (JSON web token) koji služi za potvrdu identiteta. Pomoću ovoga tokena i REST API-a, aplikacija će komunicirati s GaussBoxom preko HTTP upita i WebSocketa.

3.2 HTML, CSS i Javascript

Hyper text markup language jezik je kojeg internet preglednici razumiju i znaju prikazati. Služi za prikazivanje svih elemenata web sučelja, poput teksta, formi, slika, veza i slično. Koristi tzv. tagove, koji prikazuju neki element stranice ili služe kao kontejner za druge. Održavaju dijete-

roditelj veze, pa tako jedan tag može imati više tagova u sebi. Svi su tagovi opisani atributima, koji im daju određene vrijednosti i tako ih pobliže opisuju.

Cascading Style Sheets jezik je koji se veže na HTML te opisuje stil elemenata. Sastoji se od dva dijela – selektora, koji služi za definiranje elementa na koji se stil odnosi te od tijela, kojim se definira sam stil. Ovime se opisuje raspored elemenata na stranici, boje, obrubi, fontovi i slično.

Javascript je interpreterski skriptni jezik koji je nastao radi dodavanja dinamičnosti u web sučelja, poput slušanja na određene događaje miša ili dinamičko dodavanja i micanje sadržaja. Kroz godine se razvio, te je danas u pozadini većine web sučelja, pogoni brojne poslužitelje, biblioteke i okvire. Radi instaliranja nekih od paketa, u ovom se projektu koristi npm (node package menager)

3.2.1 Bootsrap

Radi brže izrade nekih čestih komponenti, kao što su modali, značke ili ikonice, u projektu se koristi bootstrap. [5]Bootstrap je kolekcija izrađenog CSS-a i Javascripta, koja nam ubrzava izradu nekih čestih komponenti i stilova.

3.3 Vue.js

[6]Vue.js progresivan je okvir za izradu dinamičkih korisničkih sučelja. Iako je tehnički biblioteka, češće se koristi kao okvir. Izrađen je u Javascriptu te pružna brojne mogućnosti, do te mjere da se može reći da u potpunosti mijenja način na koji se pišu web aplikacije, u odnosu na tradicionalni pristup.

Samo neki od izrazito korisnih dodataka su:

- Podjela u komponente – logički dio aplikacije, koji uključuje skup HTML-a, CSS-a i Javascripta, se može povezati u komponentu koja se lako može ponovo iskoristiti bilo gdje u aplikaciji. Dolazi sa svojim životnim vijekom te je moguće izvesti kôd na neki događaj iz životnog ciklusa – poput kreiranja ili uništavanja komponente
- Reaktivnost – logiku koju pišemo za komponente možemo pisati u kontroleru ili direktno u HTML-u. Takav modificiran HTML koji sadrži i logiku u sebi naziva se

[5] (Bootstrap, 2019)
[6] (Vue dokumentacija, 2019)

predložak, a ovisnost predloška o nekom logičkom izrazu naziva se *binding*. Ako se neki parametar o kojem je logika ovisila promijeni, rezultat će se ponovno izračunati u pozadini bez potrebe dodavanja ikakvog kôda.

- Kondicionalno renderiranje – Moguće je dodati uvjet ispisa nekog elementa unutar predloška. Povezano je s reaktivnošću, pa ako se taj uvjet promijeni u budućnosti, HTML će se automatski ažurirati.
- Renderiranje liste – Prikazivanje nekog elementa za svaki element koji postoji u nekom polju. Reaktivnost je opet prisutna, pa ako bismo dodali ili maknuli element iz polja, HTML će se automatski ažurirati.

3.3.1 Nuxt i SSR

Vue.js nije prvi niti jedini web okvir takvog tipa – najpoznatiji slični su Angular i React. Popularnost ovih tehnologija zadnjih je godina znatno porasla, no svi imaju jednu veliku prepreku. Kako je opisano u točki „REST API“, ova je aplikacija odvojena od „glavnog“ poslužitelja s podacima, te ne može posluživati pravi HTML neposredno. Kada se pokrene, pošalje upit na „glavni“ poslužitelj, pokupi podatke i tek onda Javascriptom napravi HTML.

Prepreku ovdje čini SEO (search engine optimization), jer roboti koji pretražuju web često ne čekaju na da se HTML generira ili ne izvrše Javascript, nego očekuju posluženi HTML. Zbog ovoga je uveden SSR (server side rendering), ideja da postoji „sporedni“ poslužitelj, koji na upit klijenta šalje vlastiti upit na „glavni“ poslužitelj, pokupi podatke te onda posluži HTML i Javascript na standardan način. Nakon ovog inicijalnog upita na „sporedni“ poslužitelj, kontrolu nad aplikacijom dalje preuzima web okvir, odnosno Javascript.

[7]Nuxt je implementacija SSR ideje za Vue.js, no osim SSR-a dolazi s brojnim drugim pogodnostima i unaprjeđenjima. Pruža sučelje za jednostavno korištenje te se sam brine kada se neki kod treba izvršiti na poslužitelju, a kada na klijentu, bez potrebe da se dva puta piše.

[7] (Nuxt dokumentacija, 2019)

3.3.2 Vuex

Prilikom pisanja Vue.js aplikacije javlja se problem komuniciranja između komponenata, kada događaji u jednoj komponenti trebaju utjecati na drugu komponentu. Ovo ne predstavlja problem ukoliko su komponente u dijete-roditelj odnosu, no to često nije slučaj.

Ovome možemo pristupiti na nekoliko načina. Jedan je tzv. *dependency injection*, koji instancira klase u pozadini i predaje ih kroz konstruktor drugim klasama. Takav sustav može osigurati da se preda uvijek ista instanca klase, pa bi dvije komponente koje zatraže neku klasu dobile istu instancu, a time i iste podatke. Izmjenom podataka iz jedne komponente automatski bi se izmijenili u drugoj komponenti (jer je ista instanca). Kada se to spoji s gore opisanom reaktivnošću, možemo jednostavno utjecati iz jedne komponente na drugu.

Drugi je pristup vrlo sličan, ali malo pogodniji za SSR – tzv. *store*. Ideja je da imamo centralne objekte koji drže na sebi podatke. Princip je sličan prethodnom – jedna referenca spojena s reaktivnosti.

[8]Vuex je implementacija store-ova u Vue.js okviru. Osim gore navedenoga, kada radi s nuxtom, osigurava da podaci promijenjeni na klijentu automatski budu promijenjeni na poslužitelju i obrnuto.

3.4 Axios

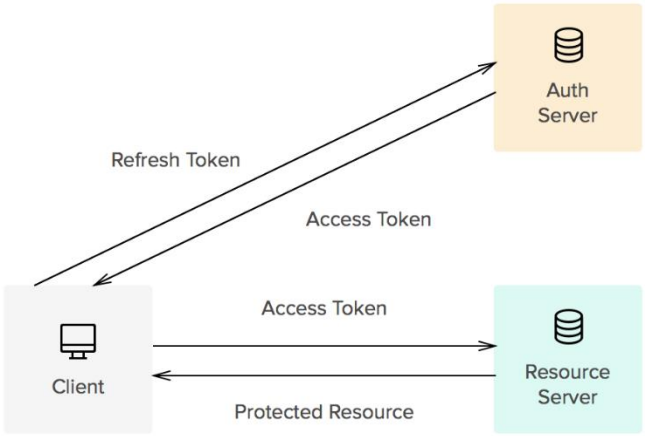
[9]Axios je biblioteka koja često dolazi u kombinaciji s Vue.js okvirom – daje jednostavno sučelje za slanje HTTP upita, kao i mogućnost kreiranja instanci standardne konfiguracije. Ove standardizirane instance jako su korisne kada se određene akcije trebaju obaviti na gotovo svakom upitu – najbolji je primjer dodavanje JWT autorizacijskog tokena u zaglavlje upita, ukoliko postoji. Drugi je slučaj povezan s prvim, a to je osvježavanje tog tokena.

Način na koji Gaussbox funkcionira, koji prati OAuth standard, je da pristupni token istekne nakon određenog vremena, ali uz njega dolazi i token za osvježavanje koji puno duže traje. Ukoliko pristupni token istekne, pošalje se osvježavajući token na poslužitelj, te on izda novi pristupni token. Axios nudi mogućnost dodavanja logike kroz standardnu konfiguraciju kojom je lako implementirati ovu logiku te je iskoristiti po cijeloj aplikaciji.

[8] (Vuex dokumentacija, 2019)
[9] (Axios, 2019)

3.5 Nuxt Auth modul

[10]Nuxtov auth modul brine se o svemu povezanom s autorizacijom. Sigurnosno sprema tokene, brine se o osvježavanju tokena, a prema programeru daje apstraktno sučelje – prijavi se, odjavi se, uzmi token i slično. Koristi axios u pozadini te se brine o konfiguriranju axiosa tako da su mu autorizacijska zaglavlja već postavljena. Daje i mogućnost izrade autorizacijskih strategija, koje će biti iskorištene za automatsko osvježavanje tokena.



Slika 3.2 Access i refresh token (Peyrott, 2019)

3.6 Lodash

[11]Lodash je kolekcija korisnih pomagala napisana u Javascriptu za neke česte probleme. Samo neke od funkcionalnosti su sortiranje, mapiranje, izrada objekata od liste, poravnanje liste i brojne druge. U ovom projektu se osobito koristi metoda *debounce*, koja osigurava da se neki dio kôda pozove samo jednom u određenom vremenskom razmaku.

3.7 Adonis websocket client

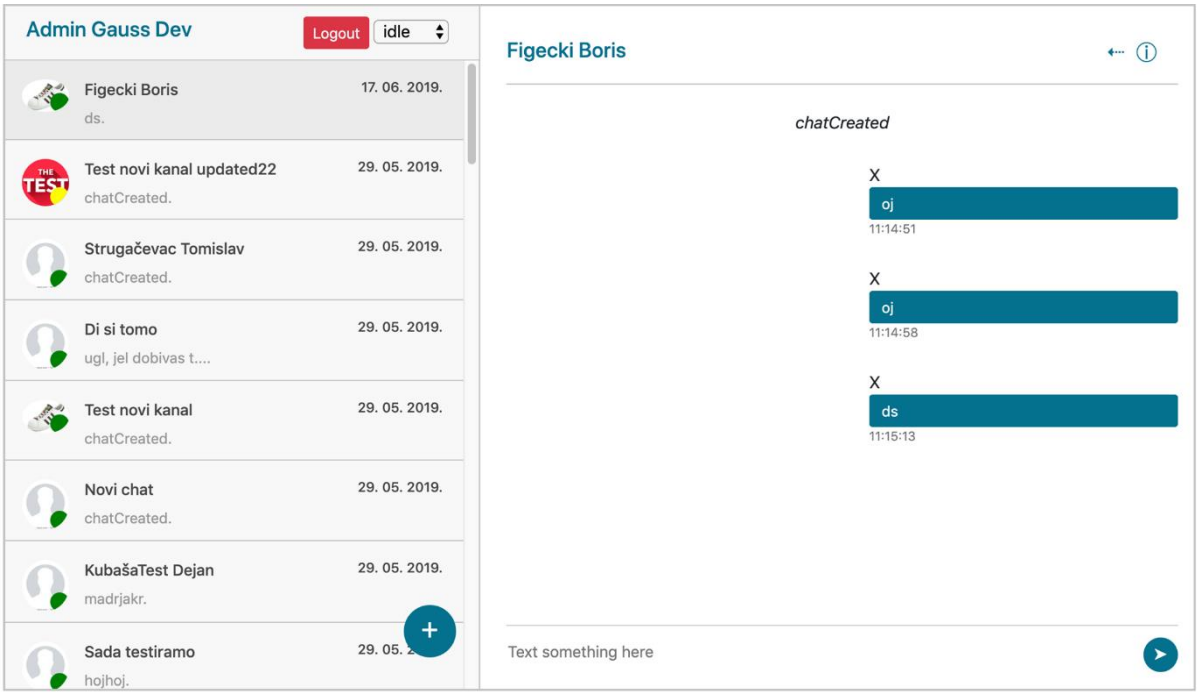
[12]Radi lakšeg povezivanja preko WebSocketa, koristi se biblioteka adonis websocket klijent. Ona pruža čistu razinu apstrakcije u radu – uvodi koncepte kanala, pretplata i događaja. Radi tako da se inicijalno poveže s poslužiteljem (*handshake*), kada šalje JWT za autorizaciju.

[10] (Nuxt auth dokumentacija, 2019)
[11] (Lodash dokumentacija, 2019)
[12] (Adonis websocket client, 2019)

Nakon toga, pokušava se pretplatiti na kanal, što poslužitelj može dopustiti ili odbiti. Kada je pretplaćena na kanal, kroz njega šalje i prima događaje s podatkovnim paketom. Odrađuje cijelu pozadinsku logiku, formatira događaje, odrađuje ping-pong logiku, rješava ponovno povezivanje u slučaju pucanja veze i slično.

4 KORISNIČKO SUČELJE

Korisničko sučelje za korištenje WebSocket tehnologije izrađeno je uz pomoć gore opisanih tehnologija. Konačni je produkt aplikacije za dopisivanje. Podijeljena je na nekoliko dijelova: prijava, sobe, kreiranje i uređivanje soba, i poruke.



Slika 4.1 Prikaz sučelja

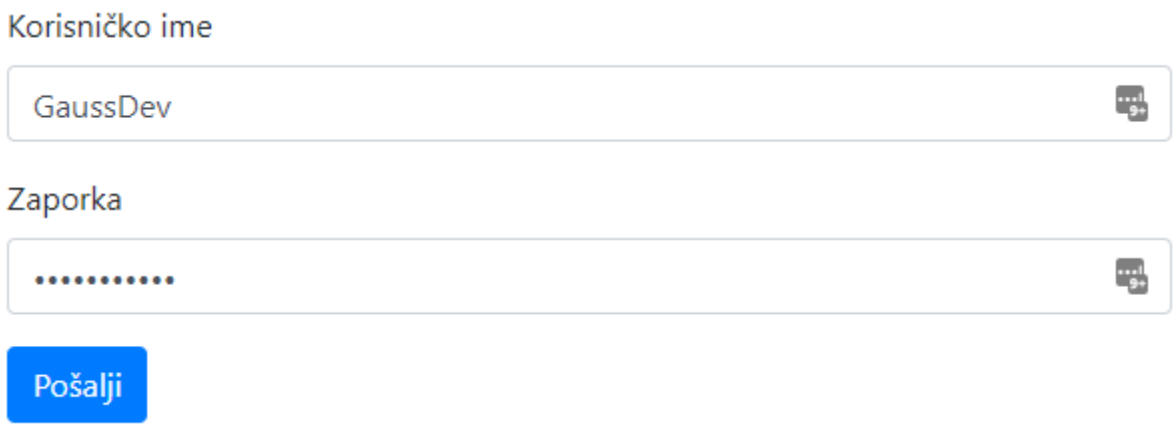
4.1 Prijava

Za prijavu izrađeno je sučelje u kojem se korisniku omogućuje unos korisničkog imena i zaporke. Iskorištena je funkcionalnost Vue.js kako bi se povezali korisnički unosi s lokalnim varijablama, koje se izmjenjuje svaki puta kada se unos izmijeni. Prilikom podnošenja forme, standardna akcija se prekida te se uzima trenutna vrijednost korisničkog imena i zaporke koja je upisana. Ta se vrijednost šalje kroz HTTP upit na poslužitelj koji vraća odgovor – dobar (statusni kôd 200) ako su uneseni ispravni podaci, ili greška (statusni kôd 400).

Kada je prijava uspješna, Auth modul zapisuje dobivene tokene u kolačiće, te će osigurati da se na budućim upitima taj token šalje kroz zaglavlje.

```vue
<template>
  <form v-on:submit.prevent="login()">
      <input type="text" v-model="username" class="form-control">
      <input type="password" v-model="password" class="form-control">
  </form>
</template>

<script>
      async login() {
        try {
          await this.$auth.loginWith('refresh', {
            data: {
              username: this.username,
              password: this.password,
              subdomain: env.env.subdomain,
            }
          });
        } catch (e) {
          if (e.response && e.response.data) {
            Toast.fire({
              type: 'error',
              title: e.response.data.message
            })
          }
        }
      }

</script>
```

Slika 4.2 Kod za prijavu. Brojni dijelovi su maknuti radi čitljivosti



Slika 4.3 Forma za prijavu

4.1.1 Auth strategija

Nakon prijave, u kolačićima su spremljeni pristupni i osvježavajući token. Auth strategija poslužit će osvježavanju tokena. U njoj možemo urediti standardnu konfiguraciju instanciranja axiosa. Pri uređivanju te konfiguracije, axios omogućuje da se određeni kôd pozove prije svakog upita. Ovdje se čita sadržaj JWT-a, vidi se do kada vrijedi, te ako je istekao osvježava se i zapisuje novo dobiveni token.

```
$axios.onRequest(async () => {
        let token = this.$auth.getToken('accessToken')
        let refreshToken = this.$auth.getRefreshToken('resfreshToken')
        const token_expires_at = jwtDecode(token).exp * 1000
        const now = Date.now()

        // token has expired
        if (now > token_expires_at) {
          this.$auth.ctx.app.$axios.setHeader('refreshToken', refreshToken)
          let {data} = await $axios.post(url)
          token = data.tokens.accessToken
          refreshToken = data.tokens.refreshToken

          this.$auth.setToken('accessToken', token)
          this.$auth.setRefreshToken('resfreshToken', refreshToken)
        }
    })
```

Slika 4.4 Kod za osvježavanje tokena. Dijelovi izmijenjeni radi čitljivosti.

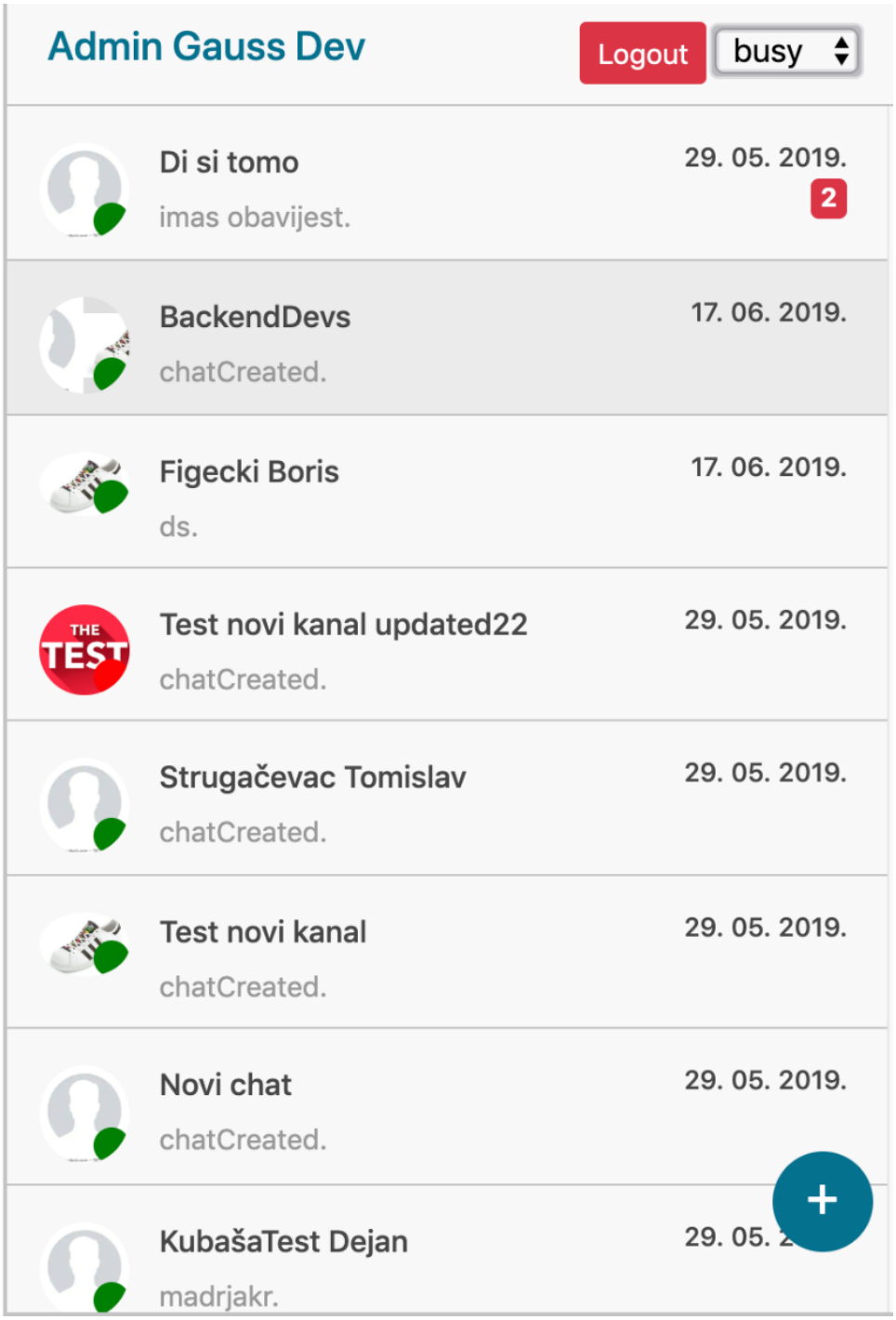
4.2 Otvaranje WebSocket veze

Kada je osoba prijavljena, postoji pristupni token kojim se može otvoriti veza s poslužiteljem. Adonis klijentska biblioteka otvara WebSocket vezu s poslužiteljem – odrađuje se handshake i provjerava se valjanost pristupnog tokena. Nakon što je veza uspostavljena, može krenuti pretplaćivanje na kanal. Kanali na ovom konkretnom projektu su tipa *backOfficeUser:ID* i *chat:ID*, gdje je ID identifikacijski broj prijavljenog korisnika, odnosno sobe u kojoj je trenutno. Nakon pretplate u kanal potrebno je registrirati osluškivanje događaja koje poslužitelj može poslati klijentu kroz taj kanal. Događaji u obrnutom smjeru šalju se prilikom korisnikove akcije.

```
  let adonisWs = Ws(socketUrl, {
    path: 'gbox-ws',
    reconnection: false
  })
  Vue.prototype.$adonisWs = adonisWs

  adonisWs.subscribeToUser = function(id, store) {
    return new Promise(resolve => {
      const userChannel = adonisWs.subscribe(`backOfficeUser:${id}`)
      userChannel.on('ready', () => {
        adonisWs.userChannel =
adonisWs.getSubscription(`backOfficeUser:${id}`)
        //listen to server events
        adonisWs.userChannel.on('chatCreated', data => {
          store.commit('chats/addChat', data)
        })
        adonisWs.userChannel.on('messageCreated', data => {
          store.commit('chats/putToTop', data.chat_id)
          store.commit('chats/addMessageFromSocket', data)
        })
        adonisWs.userChannel.on('userSeen', data => {
          store.commit('chats/userSeenChat', data)
        })
        adonisWs.userChannel.on('userStatusChange', data => {
          store.commit('chats/setUserStatus', data)
        })
        adonisWs.userChannel.on('userTyping', data => {
          store.commit('chats/setTyping', data)
        })
        adonisWs.userChannel.on('messageUpdated', data => {
          store.commit('chats/updateMessage', data)
        })
        adonisWs.userChannel.on('chatUpdated', data => {
          store.commit('chats/updateChat', data)
        })
        resolve()
      })
    })
  }

//kasnije, kada pozivamo
this.$adonisWs.withJwtToken(jwt).connect()
this.$adonisWs.on('open', () => {
    this.$adonisWs.subscribeToUser(this.$auth.user.id, this.$store)
})
```

Slika 4.5 Spajanje preko WebSocketa. Dijelovi izmjenjeni radi čitljivosti

4.3 Ispis soba



Slika 4.6 Lista soba

Podaci o trenutno aktivnim sobama dobivaju se od poslužitelja. Oni se zatim spremaju u store-ove, Store-ovi se inicijalno pune podacima koji dolaze preko HTTP upita, ali ako se dogodi kreiranje nove sobe tokom korištenja aplikacije, okine se *chatCreated* događaj (slika 4.5) te se store dopuni. Također, u slučaju primanja nove poruke, soba u koju je poruka poslana otići će na početak liste.

U predlošku za ispis soba, HTML se generira po listi iz store-a, koja je reaktivna. Zbog ovoga, bilo koja promjena u ovoj listi automatski će rezultirati promjenom u samom HTML-u. Događaji koji mogu promijeniti listu jesu *chatCreated*, *messageCreated*, *chatUpdated*, *messageUpdated* (slika 4.5)

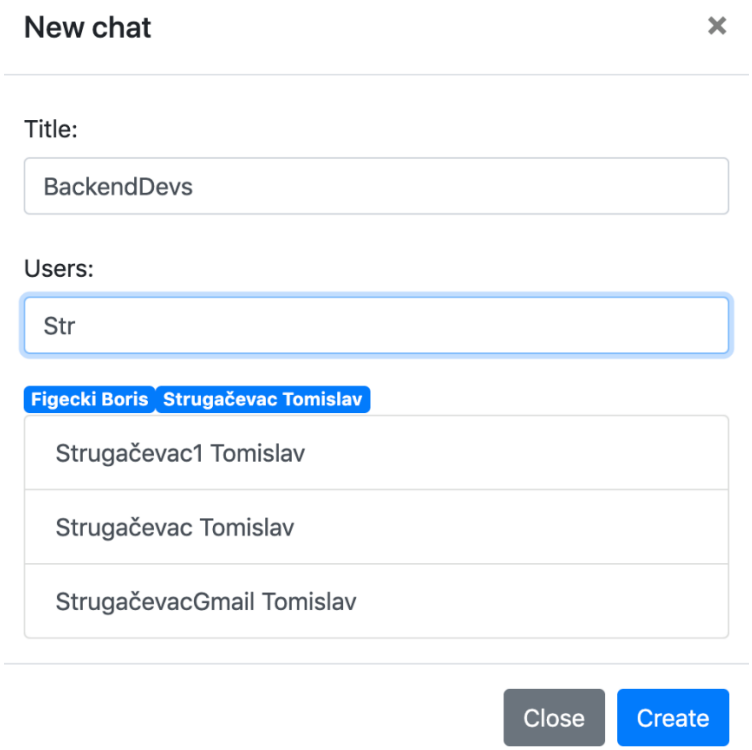
```
<div>
      <div class="inbox_chat">
        <div class="chat_list" v-for="chat in chats" :key="chat.id" v-
bind:class="{active_chat: activeChatId === chat.id}" >
          <nuxt-link v-bind:to="'/chats/'+chat.id">
            <div class="chat_people">
              <div class="chat_img">
                <img :src="profilePic" :alt="chat.displayName" v-
for="profilePic in chat.profilePics">
                <span class="badge badge-pill chat_status"
:class="statusColor(chat)+'-status'">j</span>
              </div>
              <div class="chat_ib">
                <h5>{{chat.displayName}}
                  <span class="chat_date">{{lastMessageAt(chat)}}</span>
                  <br>
                  <span class="badge badge-danger" v-
if="chat.me.unread">{{chat.me.unread}}</span>
                </h5>
                <p>{{lastMessageDisplay(chat.lastMessage)}}.</p>
              </div>
            </div>
          </nuxt-link>
        </div>
      </div>
    </div>

<script>
    computed: {
      chats() {
        return this.$store.state.chats.chats
      },
      lastMessageDisplay() {
        return messageObject => {
          if(!messageObject || !messageObject.body) return ''
          if(messageObject.body.length < 20) return messageObject.body
          else return messageObject.body.substr(0, 18) + '...'
        }
      },
    },
</script>
```

Slika 4.7 Ispis soba. Dijelovi izmijenjeni radi čitljivosti.

4.3.1 Dodavanje sobe

Kada korisnik zatraži dodavanje nove sobe, otvara se novi modal, koji je zapravo Vue.js komponenta sa svojom logikom. Prvi je dio izbor imena sobe, što je samo tekstualni okvir povezan s lokalnom varijablom.

Nakon toga slijedi biranje korisnika. U sobi može biti neograničen broj korisnika. Kada se upisuje ime za pretragu, poziva se Gaussbox modul za korisnike s upisanom vrijednosti preko HTTP upita. On zatim vraća pronađene korisnike, koji se prikazuju. Pri ovome se koristi tzv. *debouncing* metoda, koja osigurava da se ovaj HTTP upit dogodi samo jednom u nekom vremenskom intervalu, a ne svaki puta kada korisnik nešto upiše.



Slika 4.8 Dodavanje sobe

Lokalno se drže dva niza: pronađenih korisnika i izabranih korisnika. Pronađeni korisnici ažuriraju se na osnovu gore opisanog upita. Kada se klikne na pronađenog korisnika on se prebacuje u listu izabranih korisnika. Kada se klikne na izabranog korisnika, on izlazi iz te liste.

Nakon što korisnik potvrdi kreiranje sobe, preko WebSocketa poslužitelju se šalje *createChat* događaj, zajedno s izabranim naslovom i korisnicima. Nakon što poslužitelj kreira sobu, na klijentu će se okinuti *chatCreated* događaj, i novi će automatski biti dodan u store, što će automatski promijeniti HTML (slika 4.5).

```
<template>
      <input type="text" class="form-control" v-model="title">
      <input type="text" class="form-control" id="message-text" v-
on:keyup="searchUsers" v-model="searchText" />
      <a v-for="user in selectedUsers" v-on:click="unselectUser(user)">
{{user.details.displayName}}</a>
      <a v-for="user in foundUsers" v-on:click="selectUser(user)">
        {{user.details.displayName}}
      </a>
</template>
<script>
  export default {
    data: () => {
      return {
        searchText: '',
        foundUsers: [],
        selectedUsers: [],
        title: ''
      }
    },
    methods: {
      searchUsers: debounce(async function (){
        let res = await this.$axios.post('/api/v1/user/filter', {
          keywords: this.searchText
        })
        this.foundUsers = res.data.data.records
      }, 300),
      selectUser(user) {
        this.selectedUsers.push(user)
      },
      unselectUser(user) {
        let index = this.selectedUsers.indexOf(user)
        if(index !== -1) {
          this.selectedUsers.splice(index, 1)
        }
      },
      createChat() {
        let userIds = this.selectedUsers.map(user => user.id)
        this.$adonisWs.userChannel.emit('createChat', {
          userIds,
          title: this.title
        })
      }
    },
  }
</script>
```

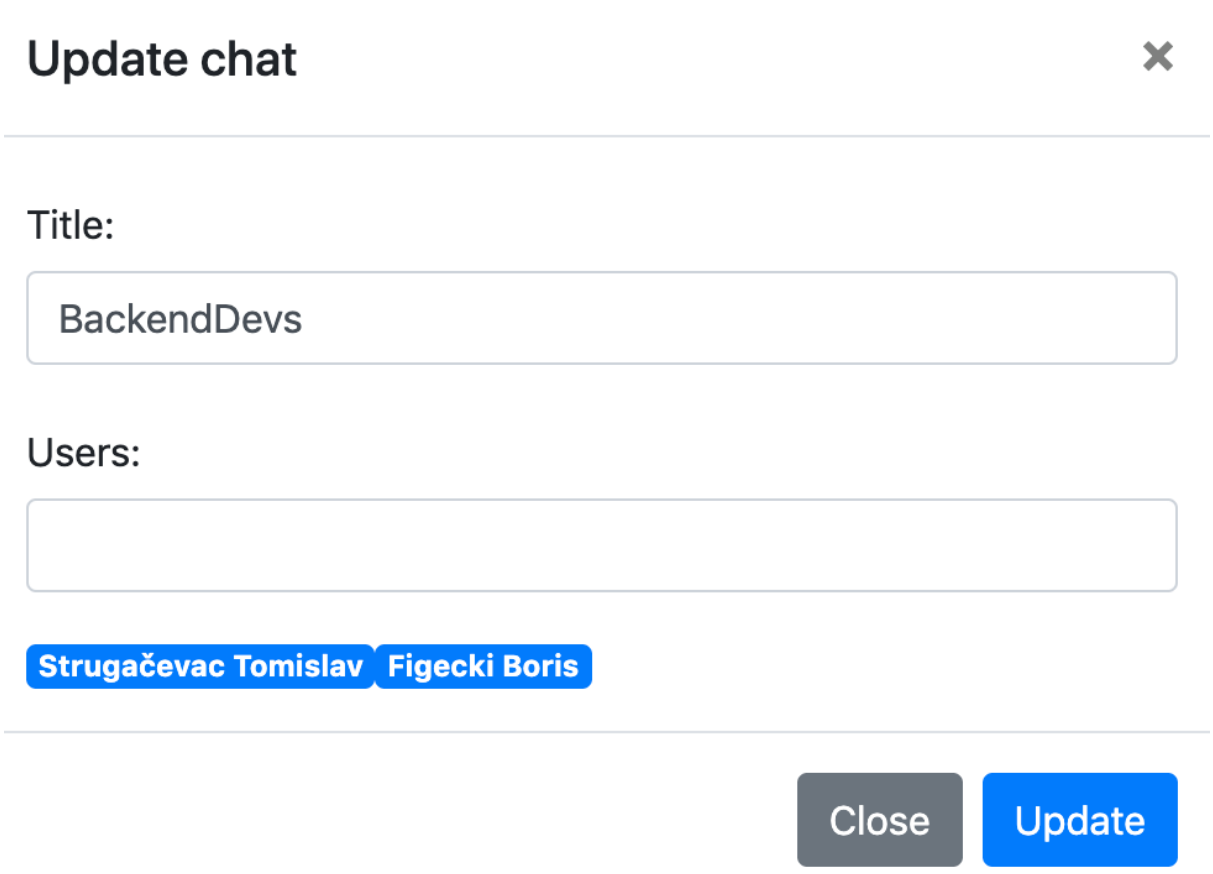
Slika 4.9 Dodavanje sobe. Dijelovi izmijenjeni radi čitljivosti.

4.3.2 Uređivanje postojeće sobe

Uređivanje sobe funkcionira na sličan način kao i dodavanje. Logika za izbor korisnika i naslova je ista, ali se šalje događaj *updateChat*. Osnovna je razlika u tome što su na samom početku podaci već popunjeni sa starim korisnicima i naslovom. Nakon što poslužitelj ažurira sobu, klijentu će poslati *chatUpdated* događaj (slika 4.5), što će ažurirati store i zbog reaktivnosti će sve biti ažurirano.

```
data: function() {
  return {
    searchText: '',
    foundUsers: [],
    selectedUsers: Object.values(this.chat.users),
    title: this.chat.title
  }
}
```

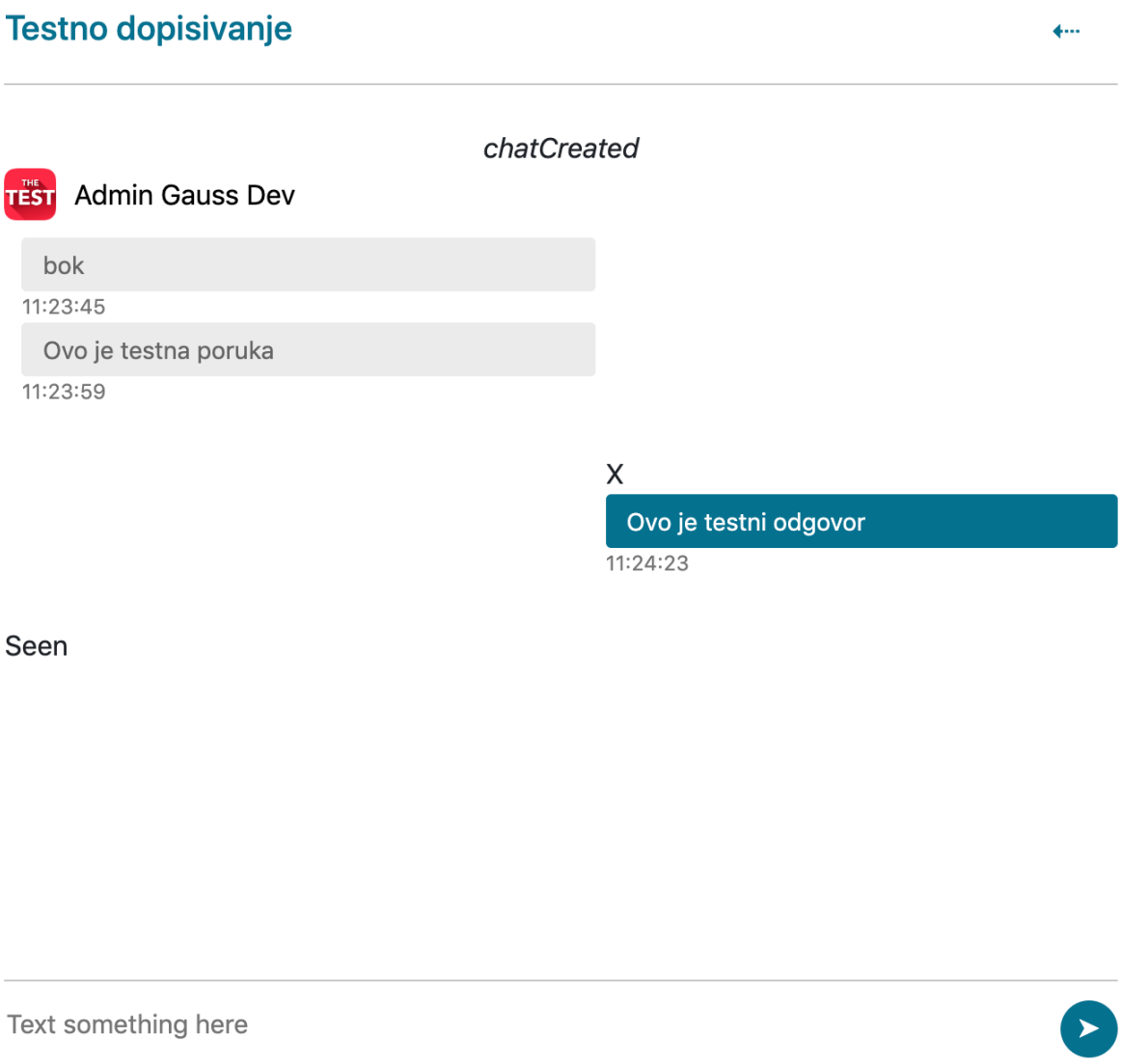
Slika 4.10 Popuna starim podacima. Dijelovi izmijenjeni radi čitljivosti.



Slika 4.11 Uređivanje sobe

4.4 Ispis poruka

Inicijalno popunjavanje store-a porukama određene sobe, slično kao popisa soba, vrši se preko HTTP upisa na poslužitelj, a kasnije poruke u stvarnom vremenu dolaze preko WebSocket događaja – *messageCreate* i *messageUpdated* (slika 4.5). Podaci za HTML dobivaju se iz store-a, pa se ažuriranjem zbog reaktivnosti HTML automatski ažurira.



Slika 4.12 Ispis poruka.

Postoji nekoliko vrsta poruke: sistemske (npr. chatCreated), vlastite, tuđe, i indikatori. Svi su primjeri vidljivi na slici 4.6

Također, izrađena je logika da se kod tuđih poruka pri promjeni pošiljaoca ispiše i njegovo korisničko ime i profilna slika. Ove podatke daje Gaussbox modul za korisnike, te se tamo mogu mijenjati.

```
<div v-for="(message, index) in chat.messages" :class="{incoming_msg:
!isMine(message), outgoing_msg: isMine(message)}" >

      <div v-if="!isMine(message) && message.type === 'standard'">
        <div v-if="!chat.messages[index-1] || !chat.messages[index-1].user_id
|| chat.messages[index-1].user_id !== message.user_id"
class="incoming_msg_img">
          <img :src="(chat.users[message.user_id] || chat.me).thumbnail"
class="user-item"/>
          <a href="#" class="displayName user-
item">{{(chat.users[message.user_id] || chat.me).displayName}}</a>
        </div>
      </div>

      <div :class="{received_msg: !isMine(message), sent_msg:
isMine(message)}">
        <div v-if="message.type === 'standard'">
          <div :class="{received_withd_msg: !isMine(message)} ">

            <div class="msg_actions" v-if="isMine(message)">
              <span v-on:click="deleteMessage(message)">X</span>
            </div>

            <p>{{message.body}}</p>
            <span class="time_date">{{messageTime(message)}}</span> </div>
        </div>
      </div>

      <div v-if="message.type === 'system'" class="text-center">
        <i>{{message.body}}</i>
      </div>

    </div>
```

Slika 4.13 Ispis poruka. Dijelovi izmijenjeni radi čitljivosti.

Osim ispisa poruka, na ovom se zaslonu događa nekoliko stvari u pozadini: praćenje jesu li ostali korisnici vidjeli poruku (eng. seen), praćenje kada korisnik tipka, beskonačno učitavanje te slanje i brisanje poruka.

4.4.1 Beskonačno učitavanje

Broj poruka u nekoj sobi može s vremenom znatno narasti, osobito u grupnim sobama. U ovakvom se sustavu to mora predvidjeti, kako se bi se izbjeglo zagušenje klijentskog računala.

Zbog toga se inicijalno uzima manji broj poruka, a kada korisnik dođe do vrha učitaju se nove poruke – beskonačno učitavanje.

Radi uspješnog implementiranja ove funkcionalnosti poslužitelj daje mogućnost paginacije podataka. Klasična paginacija radi na osnovu dva parametra – trenutna stranica i broj jedinica po stranici. No, za ovaj slučaj taj pristup nije dobar. Problem nastaje ako zatražimo slijedeću stranicu, a u međuvremenu je došao novi podatak (u ovom slučaju nova poruka). Tada bi nam poslužitelj na slijedećoj stranici vratio posljednji podatak s prethodne stranice, jer je u međuvremenu prešao na sljedeću stranicu.

Kako bismo ovo riješili, logika paginacije prilagođava se problemu. Umjesto tražene stranice, šaljemo identifikacijsku oznaku zadnjeg podatka kojeg imamo. Tada će poslužitelj znati koje podatke vratiti.

```
async loadMore({commit}, chat) {
    let oldestMessageId = chat.messages[0].id
    let messages = await this.$socketRequestService.post(`messa-
ges/${chat.id}/filter?lastId=${oldestMessageId}`)
    chat.messages.unshift(...messages)
  }
```

Slika 4.14 Učitavanje dodatnih poruka. Dijelovi izmijenjeni radi čitljivosti

4.4.2 Slanje poruka

Kada korisnik unese tekst, stisne enter ili tipku za slanje, preko WebSocketa šalje se *sendMessage* događaj, s identifikacijskom oznakom sobe i upisanim tekstom. Poslužitelj, nakon što upiše poruku, šalje opet preko WebSocketa događaj *messageCreated*, te klijent puni store i HTML se ažurira automatski.

```
<template>
   <form v-on:submit.prevent="sendMessage">
     <input type="text" v-model="currentMessage" class="write_msg" />
     <button @click="sendMessage">➤</button>
   </form>
</template>
<script>
  export default {
    data: () => {
      return {
        currentMessage: '',
      }
    },
    methods: {
      sendMessage() {
        this.channel.emit('message', {
          chatId: this.chatId,
          body: this.currentMessage
        })
        this.currentMessage = ''
      }
  }
</script>
```

Slika 4.15 Slanje poruke. Dijelovi izmijenjeni radi čitljivosti

4.4.3 Brisanje poruka

Kada korisnik pritisne tipku X iznad svoje poruke, kroz WebSocket se šalje događaj *deleteMessage*. Brisanje poruke zapravo je ažuriranje – ona postaje sistemska, s tijelom *deletedMessage*. Nakon što poslužitelj to napravi, šalje kroz WebSocket događaj *messageUpdated*.

```
updateMessage(state, data) {
    let chat = state.chats.find(chat => chat.id === data.chatId)
    let message = chat.messages.find(message => message.id ===
data.message.id)
    Object.assign(message, data.message)
  },
```

Slika 4.16 Ažuriranje poruke. Dijelovi izmijenjeni radi čitljivosti

4.4.4 Korisnik je vidio poruku

Pri inicijalnom dohvaćanju poruka, poslužitelj daje listu korisnika koji su vidjeli sve poruke. Taj se podatak stavlja u store, te se iz njega izračunava kako će izgledati *seen by* poruka.

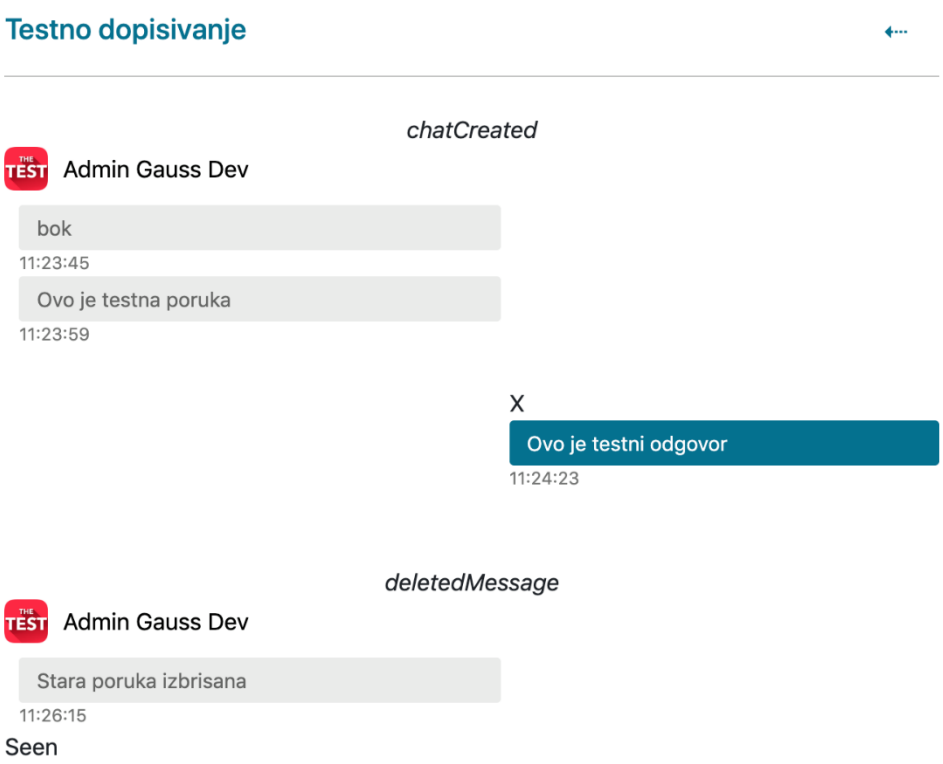
```
      seenBy() {
        let chat = this.chat
        let seenArr = this.chat.seenBy
        if(!seenArr.length) return ''
        if(seenArr.length === this.userArray.length) return 'Seen'
        return 'Seen by ' + seenArr.map(id =>
chat.users[id].firstName).join(', ')
      }
```

Slika 4.17 Izračunavanje seen by pruke. Dijelovi izmijenjeni radi čitljivosti

Svaki put kada poslužitelj pošalje *messageCreated* događaj, pošalje novu listu korisnika koji su tu poruku vidjeli. S ovim podacima se ažurira store, pa se HTML ažurira automatski. Kako bi poslužitelj znao tko je vidio poruku u stvarnom vremenu, kada korisnik dođe na zaslon od sobe, aplikacija se pretplati na chat:ID kanal, gdje je ID identifikacijska oznaka sobe. Kada korisnik izađe iz sobe, ova se pretplata prekida.

```
<script>
    mounted() {
        this.subscription = this.$adonisWs.subscribe(`chat:${this.chatId}`)
        this.channel.emit('seen', {
          chatId: this.chatId
        })
    },
    destroyed() {
      this.subscription.close()
    },
</script>
```

Slika 4.18 Obavještavanje poslužitelja o prisutnosti. Dijelovi izmijenjeni radi čitljivosti



Slika 4.19 Primjer ispisa poruke seen i obrisane poruke. Dijelovi izmijenjeni radi čitljivosti

4.4.5 Korisnik upisuje

Kako bi poslužitelj znao kada korisnik aktivno upisuje, kroz WebSocket kanal se šalje *typing* događaj. Na početak pisanja šalje se poslužitelju događaj da je korisnik krenuo pisati, a na kraj da je stao. Kako bi se osigurao vremenski razmak prije kraja pisanja, koristi se metoda *debounce* opisana u 4.3.1

Za svaku sobu zapisano je tko upisuje u store. Kada se ovo promijeni, poslužitelj kroz WebSocket šalje događaj *userTyping*, i šalje sve korisnike koji upisuju. Store se ažurira, a time i HTML.

```
<template>
   <input type="text" @keydown="startTyping" @keyup="stopTyping" />
</template>
<script>
    methods: {
      startTyping() {
        if (this.typing) return
        this.typing = true
        this.channel.emit('typing', {
          chatId: this.chatId,
          isTyping: true
        })
      },
      stopTyping: debounce(function () {
        this.typing = false
        this.channel.emit('typing', {
          chatId: this.chatId,
          isTyping: false
        })
      }, 2000),
    },
</script>
```

Slika 4.20 Obavještavanje poslužitelja da korisnik trenutno tipka. Dijelovi izmijenjeni radi čitljivosti.

4.5 Korisnički status

Korisnici mogu birati između četiri statusa, što će se odraziti na boju pored njihove sobe: online (zelena boja), busy (crvena boja), idle (žuta boja) i offline (siva boja). Promjena statusa šalje se poslužitelju kroz WebSocket događaj *setChatStatus* zajedno sa zatraženim statusom. Nakon što poslužitelj izvrši akciju, ostalim korisnicima šalje događaj *userStatusChange*.

Boja neke sobe kalkulira se na osnovu statusa korisnika koji toj sobi pripadaju.

```
statusColor() {
  let statuses = this.statuses
  return chat => {
    let users = Object.values(chat.users)
    if(users.length === 0) {
      return statuses[chat.me.chatStatus].color
    } else if(users.length === 1) {
      return statuses[users[0].chatStatus].color
    } else {
      let status = users[0].chatStatus
      for(let user of users) {
        if(user.chatStatus !== status) {
          console.log(user)
          return 'gray'
        }
      }
      return statuses[status].color
    }
  }
}
```

Slika 4.21 Izračunavanje statusa sobe. Dijelovi izmijenjeni radi čitljivosti.

4.6 Mogućnost dodatnog unaprjeđenja

Aplikacija u ovoj verziji nudi standardne mogućnosti aplikacije za dopisivanje. Postoji nekoliko načina da se aplikacija dodatno unaprijedi. Najveći su:

- Mobilna ili desktop aplikacija – kako je poslužitelj odvojen od ovoga sučelja, bilo koje drugo sučelje može se spojiti na njega, pod pretpostavkom da podržava WebSockete.
- Push notifikacije – kada korisnik primi poruku, dolazi mu *push* obavijest na smarthphone ili desktop.
- Slanje multimedijskog sadržaja – Novi tip poruke ili multimedija koji sa sobom u privitku donosi neku datoteku.
- Audio i video pozivi – Mogućnost komuniciranja preko videa i audia u stvarnom vremenu
- Spominjanje – Mogućnost upisivanja @Korisnik, kako bi tom korisniku došla posebna obavijest

5. ZAKLJUČAK

Klasični pristupi rješavanju problema u velikom broju slučajeva zadovoljavaju potrebe nekog problema. No, razvojem tehnologije povećavaju se i očekivanja krajnjih korisnika, te kada standardnim pristupom nije moguće doseći suvremeni standard, inženjeri moraju naći nove pristupe rješavanja problema.

Dvosmjerna komunikacija u stvarnom vremenu između klijenta i poslužitelja dobar je primjer koji se ne može riješiti ustaljenim načinom – HTTP upitima. Kada poslužitelj treba obavijestiti klijenta o nekom događaju, potrebno je pronaći novu tehnologiju koja to dopušta. Standardna tehnologija koja može riješiti ovu stvar jest WebSocket.

WebSocketi miču se od poznatih koncepata upita i odgovora, zaglavlja, tijela i sličnoga, te u svojoj apstrakciji uvodi koncepte pretplata, kanala i događaja. Događaji se mogu slati u oba smjera, što omogućuje korisnicima pregled promjena u stvarnom vremenu.

Klasični primjer sustava kojem je potrebna dvosmjerna komunikacija u stvarnome vremenu je aplikacija za slanje poruka. Poruka može biti poslana u bilo kojem trenutku, soba može biti kreirana ili se može dogoditi bilo koja druga akcija na računalu nekog korisnika, a da se rezultat u stvarnom vremenu mora vidjeti na računalu drugog korisnika. WebSocketi omogućuju ovu funkcionalnost. Spajanjem WebSocketa s modernim tehnologijama za izradu web sučelja, izrađena je takva aplikacija.

LITERATURA

[1] *10M Concurrent Websockets*. (16. Lipanj 2019). Dohvaćeno iz Goroutines: http://goroutines.com/10m

[2] *Adonis websocket client*. (16. Lipanj 2019). Dohvaćeno iz Adonis: https://adonisjs.com/docs/4.1/websocket-client

[3] *Axios*. (11. Lipanj 2019). Dohvaćeno iz Github: https://github.com/axios/axios

[4] *Bootstrap*. (14. Lipanj 2019). Dohvaćeno iz Bootstrap: https://getbootstrap.com/

[5] Fette, M. (16. Lipanj 2019). *The WebSocket Protocol*. Dohvaćeno iz Internet Engineering Task Force: https://tools.ietf.org/html/rfc6455

[6] James, G. (17. Lipanj 2019). *Creating a simple REST API in PHP*. Dohvaćeno iz Shareurcodes: https://shareurcodes.com/blog/creating%20a%20simple%20rest%20api%20in%20php

[7] *Lodash dokumentacija*. (12. Lipanj 2019). Dohvaćeno iz Lodash: https://lodash.com/docs/

[8] *Nuxt auth dokumentacija*. (15. Lipanj 2019). Dohvaćeno iz Nuxt auth dokumentacija: https://auth.nuxtjs.org/

[9] *Nuxt dokumentacija*. (15. Lipanj 2019). Dohvaćeno iz nuxtjs.org: https://nuxtjs.org/guide

[10] Peyrott, S. (17. Lipanj 2019). *Refresh Tokens: When to Use Them and How They Interact with JWTs*. Dohvaćeno iz Auth0: https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/

[11] *PHP WebSocket server*. (16. Lipanj 2019). Dohvaćeno iz Ollyxar: https://ollyxar.com/websockets

[12] *Vue dokumentacija*. (15. Lipanj 2019). Dohvaćeno iz vuejs.org: https://vuejs.org/v2/guide/

[13] *Vue.js documentation*. (13. 6 2019). Dohvaćeno iz Vue.js: https://vuejs.org/v2/guide/

[14] *Vuex dokumentacija*. (15. Lipanj 2019). Dohvaćeno iz vuex.vuejs.org: https://vuex.vuejs.org/guide/

[15] West, M. (17. Lipanj 2019). *An Introduction to WebSockets*. Dohvaćeno iz Treehouse: https://blog.teamtreehouse.com/an-introduction-to-websockets

[16] *What is HTTP?* (16. Lipanj 2019). Dohvaćeno iz w3schools: https://www.w3schools.com/whatis/whatis_http.asp

SAŽETAK

Sustav za komunikaciju u stvarnom vremenu mora omogućiti dvosmjernu komunikaciju između poslužitelja i klijenta kako bi akcije jednog korisnika imale utjecaj na sučelje drugog korisnika u stvarnome vremenu. Jedna od najpogodnijih tehnologija za ostvarivanje ovakve komunikacije je WebSocket.

U radu su opisani osnovni principi komunikacije u stvarnom vremenu na transportnom i aplikacijskom sloju, razlika između HTTP-a i WebSocketa, kao i moderne tehnologije izrade web sučelja. Uz pomoć opisanih principa i tehnologija izrađeno je web sučelje za izmjenu poruka između korisnika u stvarnom vremenu.

Ključne riječi: komunikacija u stvarnome vremenu, web sučelje, komunikacijski protokoli, kanali, pretplate, Vue.js, razmjena poruka

ABSTRACT

REAL TIME COMMUNICATION SYSTEM

Real time communication system has to enable two-way communication between server and client, so that one user's actions could affect another user's interface in real time. One of the most convenient technologies that empowers this kind of communication is WebSocket.

Firstly, basic principles of real time communication on transport and application layer are described. Then, HTTP and WebSockets are compared. Furthermore, this seminar goes through modern technologies for creating web interfaces. Using abovementioned principles and technologies, web interface for real time message exchange was created.

Keywords: real time communication, web interface, communication protocols, channels, subscriptions, Vue.js, message exchange

ŽIVOTOPIS

Ivan Jakab, rođen 26. lipnja 1997. u Vinkovcima. Osnovnoškolsko obrazovanje završio u „OŠ Ivana Brlić-Mažuranić Rokovci-Andrijaševci“. Pohađao matematičku gimnaziju Matije Antuna Reljkovića u Vinkovcima i maturirao 2016. godine. Iste godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija na sveučilištu Josipa Jurja Strossmayera u Osijeku. Upoznat s procesom izrade web aplikacija, te trenutno zaposlen u Gauss developmentu.