UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Nejc Štebe

Izdelava spletne aplikacije za video klepet

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2015

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Nejc Štebe

Izdelava spletne aplikacije za video klepet

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš. pred. dr. Aljaž Zrnec

Ljubljana, 2015

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Izdelava spletne aplikacije za video klepet

Tematika naloge:

Večina dozdajšnjih spletnih video klepetov je za delovanje potrebovala vtičnike za brskalnike, najbolj pogosto Adobe Flash. Leta 2011 je podjetje Google predstavilo WebRTC, vmesnik za brskalnik Google Chrome, ki omogoča povezavo med dvema brskalnikoma brez uporabne dodatnih vtičnikov. Vmesnik WebRTC sedaj podpirajo le najnovejši brskalniki. V okviru diplomskega dela predstavite delovanje, zmožnosti in uporabo vmesnika. Izdelajte aplikacijo, ki bo dvema odjemalcema omogočala video komunikacijo. Raziščite še področja varnosti, zmogljivosti in prenosa drugih oblik podatkov.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Nejc Štebe sem avtor diplomskega dela z naslovom:

Izdelava spletne aplikacije za video klepet v brskalniku

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Aljaža Zrneca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 16. 12. 2015 Podpis avtorja:

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
WebRTC	Web real-time communication	Spletna realno časovna komunikacija.
HTML	Hypertext markup language	Jezik za označevanje nadbesedila.
CSS	Cascading style sheets	Kaskadne stilske podloge.
TCP	Transmission control protocol	Protokol za nadzor prenosa.
NAT	Network address translation	Preslikava omrežnih naslovov.
ICE	Interactive connectivity establishment	Tehnika, ki omogoča neposredno komunikacijo med soležnimi napravami.
STUN	Session Traversal Utilities for NAT	Nabor orodij, ki napravam omogoča, da izvejo za svoj javni naslov IP.
TURN	Traversal Using Relays around NAT	Tehnika, ki omogoča komunikacijo med soležnimi napravami s posredovanjem sporočil.
IP	Internet protocol	Internetni protokol.
HTTP	HyperText Transfer Protocol	Je protokol za prenos dokumentov HTML.
HTTPS	HyperText Transfer Protocol Secure	Varna različica protokola http.
RTC	Real-time communication	Realno časovna komunikacija.
TLS	Transport layer security	Protokol za šifriranje omrežnih povezav

Povzetek

Cilj diplomskega dela je razviti spletno aplikacijo za video klepet in prenos datotek. Uporabniki se združujejo v sobah, v katerih lahko vzpostavijo prenos video posnetka, si izmenjujejo besedilna sporočila ali pošiljajo datoteke. V okviru diplomskega dela je bila razvita spletna aplikacija, ki je sestavljena iz dela, ki se izvaja v brskalniku odjemalca in dela, ki se izvaja na strežniku. Za izdelavo so bile uporabljene najnovejše tehnologije spletnega razvoja in vmesniki odprtokodnega projekta WebRTC.

Ključne besede: video klepet, spletna aplikacija, WebRTC, pošiljanje datotek, HTML5, JavaScript

Abstract

The goal of this thesis is to develop a video chat application with the additional functionality of transferring files. Users can join rooms in which they communicate using text messages and video streaming. For this thesis a web application has been developed in two parts. One is executing in the browser and the other is executing on the server. The latest web development technologies and the interfaces of the WebRTC open source project have been used to develop the application.

Keywords: video chat, web application, WebRTC, sending files, HTML5, JavaScript

Poglavje 1 Uvod

Komunikacija je ena glavnih funkcionalnost svetovnega spleta. Pa vendar je video klepet omejen na posebno programsko opremo, ki ni odprtokodna in jo je potrebno za uporabo predhodno namestiti za uporabo. Rešitev za ta problem omogoča projekt WebRTC. To je odprtokodni projekt, ki med brskalniki omogoča neposreden prenos slike, zvoka ali podatkov v drugih oblikah brez uporabe vtičnikov.

V okviru diplomske naloge smo si zato zadali nalogo izdelati spletno aplikacijo, ki omogoča video klepet, izmenjavo besedilnih sporočil in prenašanje datotek. Aplikacija je razvita na način, ki ne bo potreboval zahtevnega procesiranja ali shranjevanja večjih količin podatkov na strežniku in bo preprosta za uporabo. Uporabili smo najnovejše tehnologije, ki jih ponuja peta različica označevalnega jezika HTML5.

V okviru poglavja 2 smo opisali programske jezike, knjižnice in ostale tehnologije, ki smo jih uporabili za razvoj aplikacije ter s primeri predstavili njihovo uporabo. V tretjem poglavju smo predstavili projekt WebRTC, katere vmesnike ponuja in zakaj se jih uporablja. Predstavljena je tudi podpora brskalnikov. Četrto poglavje opisuje, kako smo razvili aplikacijo. Opis razvoja smo razdelili na dva dela: prvi opisuje razvoj aplikacije na strani odjemalca, drugi pa na strani strežnika. V petem poglavju smo predstavili projekt WebRTC z varnostnega vidika. V šestem poglavju pa smo podali sklepne ugotovitve.

Poglavje 2 Tehnologije

V tem poglavju bom predstavil tehnologije, ki smo jih uporabili za razvoj spletne aplikacije. Pri nekaterih bom prikazal tudi sintakso in podal primer uporabe za boljše razumevanje namembnosti teh tehnologij.

2.1 JavaScript

JavaScript je skriptni objektni jezik namenjen izdelavi programov, ki se izvajajo v spletnih straneh [1]. Ker ga podpirajo vsi spletni brskalniki in se izvaja na skoraj vsaki spletni strani, je eden najbolj razširjenih programskih jezikov. JavaScript skupaj z označevalnim jezikom HTML in stilnimi predlogami CSS tvori trojček jezikov, ki skrbijo za postavitev, izgled in delovanje spletnih strani. JavaScript omogoča izvajanje t.i. odzivnih funkcij, ki se začnejo izvajati po določenem dogodku, na primer: pritisk na gumb na spletni strani. V sledečem odseku kode je prikazana sintaksa jezika JavaScript in izvajanje odzivne funkcije.

```
var odzivnaFunkcija = function() {
     alert('primer');
};
// Zaženi podano funkcijo po treh sekundah
setTimeout(odzivnaFunkcija, 3000);

// Ko uporabnik klikne na element z razredom gumb, zaženi funkcijo
var element = document.getElementById('.gumb')[0];
element.onclick = odzivnaFunkcija;
```

2.2 Node.js

Programski jezik JavaScript se ne izvaja le v brskalniku, temveč se lahko izvaja tudi na strežniku. Node.js je izvajalno okolje za jezik JavaScript in se ga pogosto uporablja za izvajanje spletnih storitev na strežniku [2]. Izbrali smo ga, ker nam je omogočil, da izdelamo strežniško aplikacijo in spletne aplikacijo na strani odjemalca v istem jeziku.

2.3 Express

Express je knjižnica za izvajalno okolje Node.js napisana v jeziku JavaScript. Namenjena je za preprostejše pisanje spletnih storitev in upravljanje z zahtevki protokola HTTP.

2.3.1 Sintaksa Express in Node.js

Primer zagona strežnika Node.js, ki odgovarja na zahtevke tako, da kot odgovor pošlje dokument HTML.

```
var app = require('express')();
var server = require('http').Server(app);

app.get('/', function(req, res) {
     res.sendfile('index.html');
});

http.listen(80, function(){
     console.log('listening on port 80');
});
```

2.4 WebSocket

WebSocket je protokol, ki omogoča dvosmerni komunikacijski kanal med strežnikom in spletnim brskalnikom z le eno odprto TCP povezavo [3]. Oba udeleženca v povezavi sta enakovredna zaradi česar je način uporabe enak na obeh straneh komunikacijskega kanala. Pogosto se ga uporablja za realno časovno komunikacijo in prenašanje tokov podatkov. Protokol podpirajo vsi novejši brskalniki in spletni strežniki.

2.5 Socket.IO

Socket.IO je knjižnica za programe napisane v jeziku JavaScript, ki poenostavi uporabo protokola WebSocket in omogoča uvrščanje povezav v t.i. sobe. Sobe olajšajo posredovanje sporočil od enega odjemalca do drugih odjemalcev v isti sobi. Knjižnica Socket.IO je dvodelna, saj se mora za uspešno povezavo izvajati tako na strežniku kot pri odjemalcu [4].

2.5.1 Sintaksa Socket.IO

Primer vzpostavitve povezave na strani strežnika:

```
var io = require('socket.io')(server);

io.on('connection', function(socket) {
	socket.join('test-room');

	socket.emit('message',
		{text: 'You have entered a room.'}
	);
	socket.broadcast.to('test-room').emit('message',
		{text: 'Somebody entered the room.'}
	);
});
```

Po prenosu spletne strani in zagonu knjižnice Socket.IO, se vzpostavi povezava s strežnikom. Odzivna funkcija, ki se izvede po uspešno vzpostavljeni povezavi prejme objekt *socket*, ki se navezuje na trenutno povezavo. S klicem funkcije join nad prejetim objektom lahko odjemalca uvrstimo v sobo *test-room* in v naslednjih korakih pošljemo sporočilo nazaj odjemalcu ter vsem ostalim, ki so uvrščeni v isto sobo.

Drugi del knjižnice Socket.IO pa se prenese skupaj s spletno stranjo v brskalnik odjemalca. Ker je tudi tamkajšnji program napisan v jeziku JavaScript, je sintaksa podobna.

Primer vzpostavitve povezave na strani odjemalca:

```
<script src="/scripts/socket.io.js"></script>
<script>
var socket = io();
socket.on('message', function(message) {
	alert(message.text);
});
socket.emit('message to server', {text: 'Vsebina sporočila'});
</script>
```

Ko uporabnik obišče spletno stran, se mu prikaže opozorilo z vsebino »You have entered a room.«, vsem ostalim uporabnikom, ki so že na spletni strani pa sporočilo »Somebody entered the room.«.

2.6 jQuery

Knjižnica jQuery je napisana v jeziku JavaScript. Narejena je bila zato, da bi programerjem olajšala delo s spletnimi stranmi. Ko je koda vdelana v dokument HTML, se knjižnica shrani v objekt, ki vsebuje vse metode, ki jih ponuja jQuery. Knjižnica omogoča preprosto iskanje elementov v dokumentu in upravljanje z njimi, na primer: spreminjanje vsebine tekstovnih značk, premikanje gradnikov spletne strani in tudi odpiranje pojavnih oken.

2.6.1 Sintaksa jQuery

Knjižnica jQuery je shranjena v objekt »$«, katerega lastnosti so metode, ki jih knjižnica ponuja. Iskanje elementov poteka tako, da izvedemo objekt »$« z vhodnim parametrom, nizom, ki vsebuje ključ za iskanje elementov. Primer sintakse prikazuje skrivanje okvirja za video po tem, ko uporabnik pritisne na gumb.

```
<script>
$('button').click(function() {
     $('div#video-container').hide();
});

$('.razred').addClass('novi-razred');
</script>
```

2.7 HTML5

HTML5 je najnovejša različica označevalnega jezika HTML, ki jo je oktobra 2014 objavila organizacija World wide web consortium. Najnovejša različica je omogočila uporabo različnih multimedijskih značk kot `<video>`, `<audio>` in `<canvas>.` Za izdelavo spletne klepetalnice je uporabna značka <video>, saj omogoča prikazovanje video posnetkov. Vir posnetkov ni pomemben, saj jih lahko brskalnik naloži s strežnika, z naprave uporabnika ali pa se, kot v našem primeru, video posnetek prenaša neposredno od kamere nekega drugega uporabnika.

Novi standard omogoča preprosto upravljanje z elementom `<video>` tako, da v kodi spletne strani zaženemo njegove kontrolne funkcije.

2.7.1 Primer uporabe elementa <video>

V sledečem primeru je prikazano iskanje značke <video> in izvajanje njenih metod za dodajanje video posnetka, začetek predvajanja ter prekinitev.

```
<video></video>
<script>
	var videoElement = $('video')[0];
	videoElement.addStream(streamObject);
	videoElement.play();
	setTimeout(function() {
		videoElement.stop();
	}, 5000);
</script>
```

2.8 Bootstrap

Bootstrap je zbirka gradnikov uporabniških vmesnikov in stilskih pravil, ki razvijalcem olajša razvoj spletnih strani, saj je za umestitev gradnika potrebno le izbrati pravilen razred in ga določiti ustrezni znački HTML [5]. Zbirka Bootstrap ponuja veliko različnih gradnikov, na primer: pojavna okna, vnosna polja, gumbe različnih stanj in opozorila. Uporabili smo jo za hitrejši razvoj uporabniškega vmesnika in z njo omogočili, da se postavitev spletne strani prilagodi različnim velikostim zaslonov uporabnikov.

Poglavje 3 Opis projekta WebRTC

V tem poglavju je predstavljen projekt WebRTC, kaj so cilji projekta in kaj ponuja. V poglavjih 3.1, 3.2, 3.3, in 3.4 so predstavljene tehnologije in vmesniki, ki jih ponujajo brskalniki, ki so našteti v poglavju 3.5.

WebRTC je nabor programskih vmesnikov, ki omogočajo komunikacijo med dvema spletnima brskalnikoma brez uporabe vtičnikov. Razvijati ga je začelo podjetje Google in ga javnosti predstavilo kot odprtokodni projekt leta 2011. Standard vmesnika je v pripravi v organizaciji World wide web consortium [6]. Cilj projekta je omogočiti brskalnikom kakovostno realnočasovno povezavo z drugimi brskalniki. Nekaj naprednih implementacij je že v uporabi v novejših različicah brskalnikov Google Chrome in Mozilla Firefox.

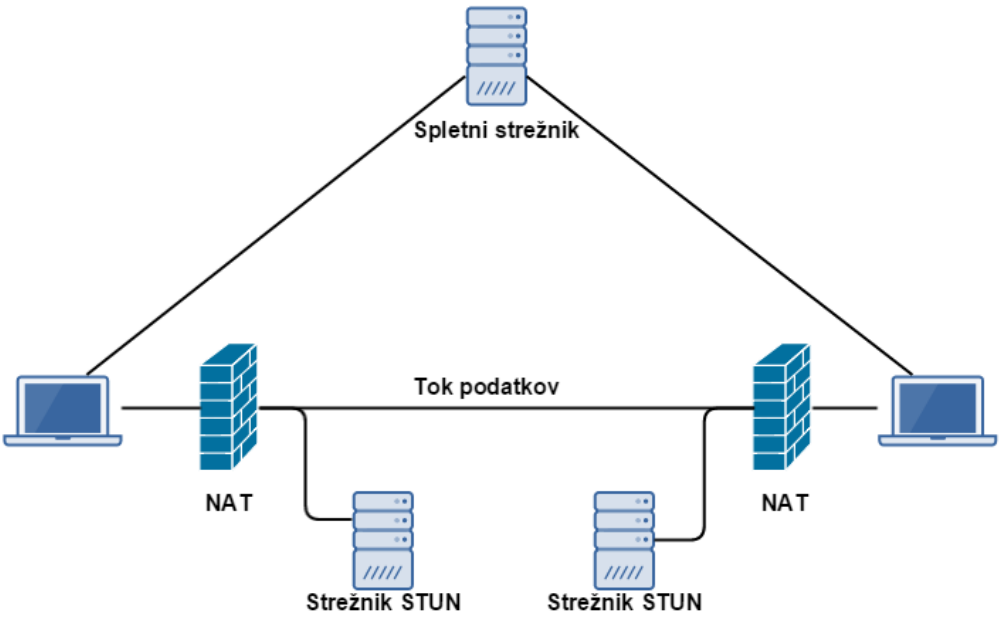
WebRTC omogoča različne vrste realnočasovnih komunikacij, na primer: prenos zvoka, video posnetka, besedila ali toka binarnih podatkov. Za vzpostavitev povezave si morata odjemalca izmenjati omrežne podatke, kot na primer: naslov IP in številko vrat ter vrsto medija in njegove lastnosti, na primer: ločljivost slike ali kodeke. Za prenos teh podatkov mora poskrbeti razvijalec sam, saj vmesnik WebRTC tega ne podpira.

Čeprav vmesnik WebRTC omogoča t.i. »peer to peer« povezavovanje, potrebuje za delovanje spletni strežnik [7]. Ta pogosto igra dve vlogi in sicer: postreže s spletno stranjo in posreduje pri izmenjavi omrežnih podatkov. Po vzpostavljeni povezavi poteka vsa komunikacija neposredno med brskalnikoma, brez vmesnega strežnika. Vsa komunikacija je zaščitena s kriptografskim protokolom TLS vendar le v primeru, da se je spletna stran prenesla s protokolom HTTPS.

3.1 Strežniki STUN in TURN

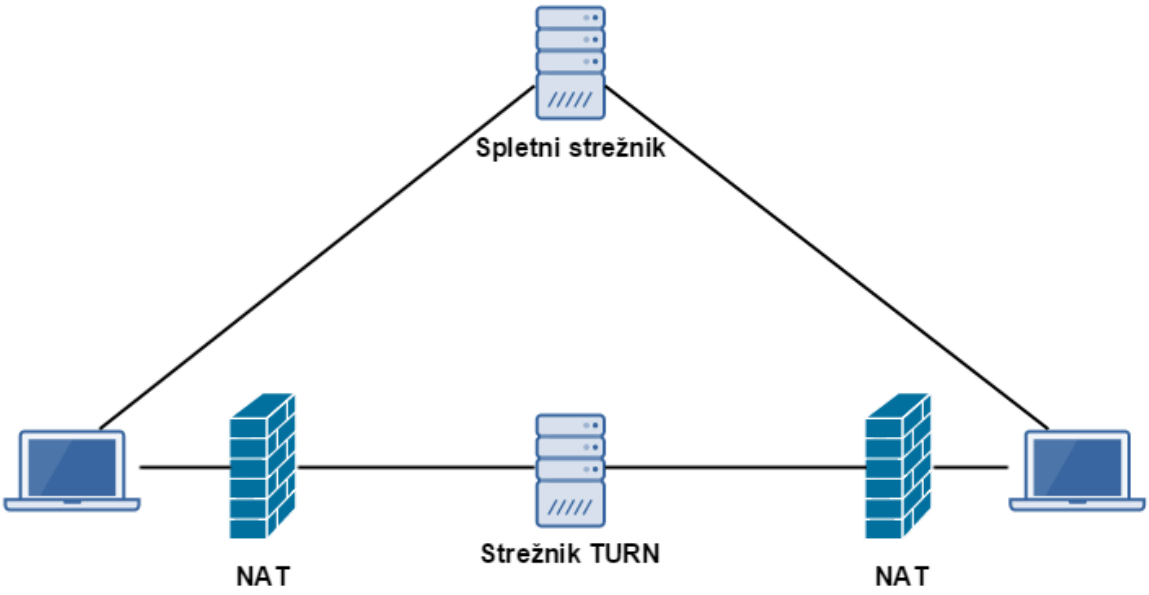
Uporabnik za neposredno povezavo z drugim uporabnikom potrebuje njegov naslov. Če se le eden nahaja za plastjo NAT, se ne bosta uspela povezati. Za rešitev te težave se uporablja ogrodje ICE, ki poskuša najti najboljši možni način za povezovanje dveh uporabnikov in strežnike STUN. Strežniki STUN opravljajo preprosto nalogo: na prispele zahtevke odgovorijo s paketom, ki vsebuje naslov IP in številko vrat, katera sta zapisana v glavi prispelega zahtevka. Strežniki STUN torej omogočajo napravam, da odkrijejo svoj javni naslov, če se nahajajo za plastjo NAT.

Ogrodje ICE najprej poizkusi vzpostaviti povezavo z uporabo naslovov obeh gostiteljev, kar pa ne uspe, če se nahajata za plastjo NAT. Druga možnost je, da ogrodje ICE pošlje poizvedbo na strežnik STUN, ki odgovori z naslovom pošiljatelja. S tem aplikacija pridobi svoj javni naslov IP, ki ga lahko uporabi za vzpostavitev neposredne povezave. To naj bi zadostovalo za uspešno vzpostavitev v 92% odstotkih primerov [8]. Slika 3.1 prikazuje delovanje spletne aplikacije s pomočjo strežnikov STUN.



Slika 3.1: Delovanje spletne aplikacije s pomočjo strežnikov STUN

V primeru, da neposredna povezava ne more biti vzpostavljena, poskrbi ogrodje ICE za posredovanje povezave prek strežnika TURN, kot je prikazano na sliki 3.2. Strežnik TURN je nadgrajena različica strežnika STUN z dodatno funkcionalnostjo pošiljanja toka podatkov do drugega uporabnika [9]. Strežniki TURN zahtevajo več sistemskih virov še posebej pasovne širine kot strežniki STUN, saj prejemajo in odpošiljajo tokove podatkov. Zaradi potrebe po višji zmogljivosti strežnika, ogrodje ICE sprva poizkusi vzpostaviti povezavo brez strežnika TURN. Dodatna pomanjkljivost je tudi to, da imajo takšne povezave večjo zakasnitev zaradi vmesnega strežnika.



Slika 3.2: Delovanje spletne aplikacije, ko neposredna povezava ni mogoča.

WebRTC je v brskalnikih predstavljen s tremi vmesniki: MediaStream, RTCPeerConnection in RTCDataChannel.

3.2 MediaStream

Vmesnik MediaStream predstavlja sinhronizirane tokove podatkov, na primer: tok slike zajete z videokamero in tok zvoka zajetega z mikrofonom. Sinhroniziran tok se pridobi s klicem metode navigator.getUserMedia(). Vmesnik MediaStream omogoča tudi zajem zaslonske slike uporabnika vendar je to možno le, če je vzpostavljena šifrirana povezava s protokolom HTTPS.

3.3 RTCPeerConnection

Je vmesnik za vzpostavljanje povezave med dvema brskalnikoma [10]. Objekt, ki je rezultat inicializacije nove povezave, potrebuje reference na funkciji, ki se izvedeta med poizkusom vzpostavitve. Ti funkciji definira programer in opravljata dve vlogi. Prva je pošiljanje ponudbe za vzpostavitev povezave, druga pa odgovarjanje na prejeto prošnjo z odgovorom. Prek vmesnika RTCPeerConnection lahko pošljemo tudi sinhronizirane tokove, ki jih dobimo s klicem MediaStream vmesnika.

3.4 RTCDataChannel

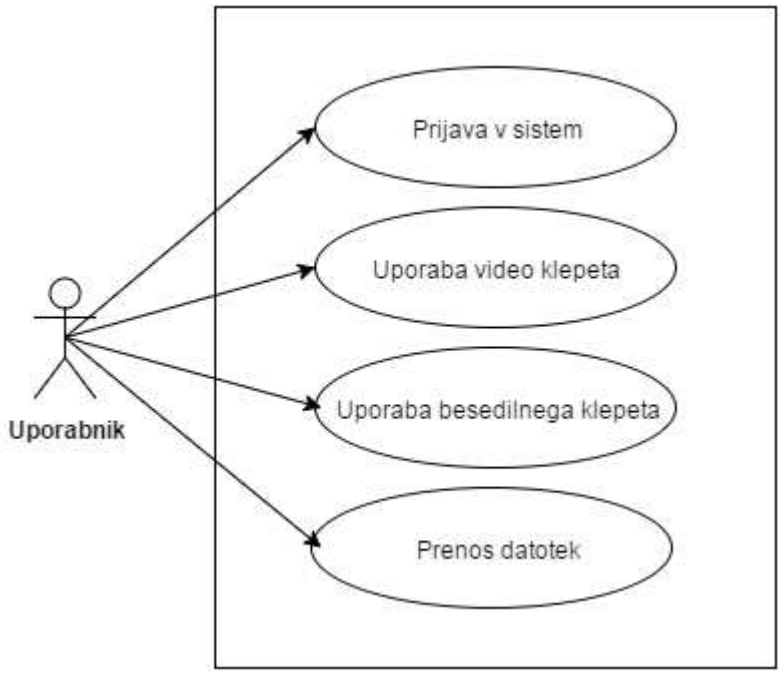
Poleg videa in zvoka, lahko prenašamo tudi besedilo in binarne podatke z uporabo podatkovnega kanala RTCDataChannel [11]. Po vzpostavljeni povezavi RTCPeerConnection lahko odpremo kanal, ki je predstavljen kot objekt z metodo send() in odzivno funkcijo onmessage, ki se izvede po prispelem sporočilu.

3.5 Podpora brskalnikov

Ker je WebRTC nova tehnologija, je podpora omejena na najnovejše brskalnike Chrome, Opera, Firefox in brskalnik mobilnega operacijskega sistema Android. Čeprav je v teh brskalnikih WebRTC že podprt, se vmesniki imenujejo in delujejo različno. V brskalniku Chrome je metoda getUserMedia imenovana webkitGetUserMedia in v brskalniku firefox, mozGetUserMedia. Podobno se razlikujejo imena skoraj vseh funkcij v različnih brskalnikih. Tudi tipi podatkov, ki jih lahko pošilja RTCDataChannel so različni. Zaradi teh razlik smo se osredotočili razviti aplikacijo, ki podpira le brskalnik Chrome.

Poglavje 4 Razvoj aplikacije za video klepet in prenos datotek

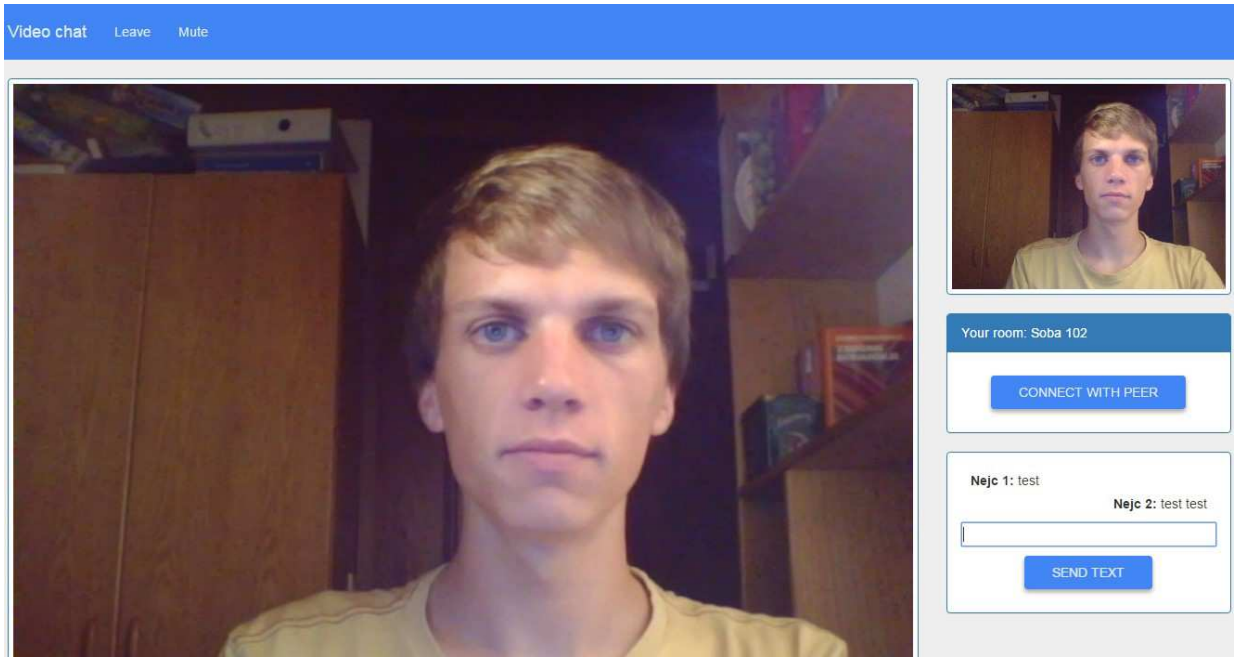
Aplikacija vzpostavi video, zvočno in besedilno povezavo med dvema uporabnikoma. Omogoča tudi prenos datoteke od enega uporabnika do drugega. Primeri uporabe aplikacije so prikazani v diagramu primerov uporabe na sliki 4.1.



Slika 4.1: Diagram primerov uporabe spletne aplikacije.

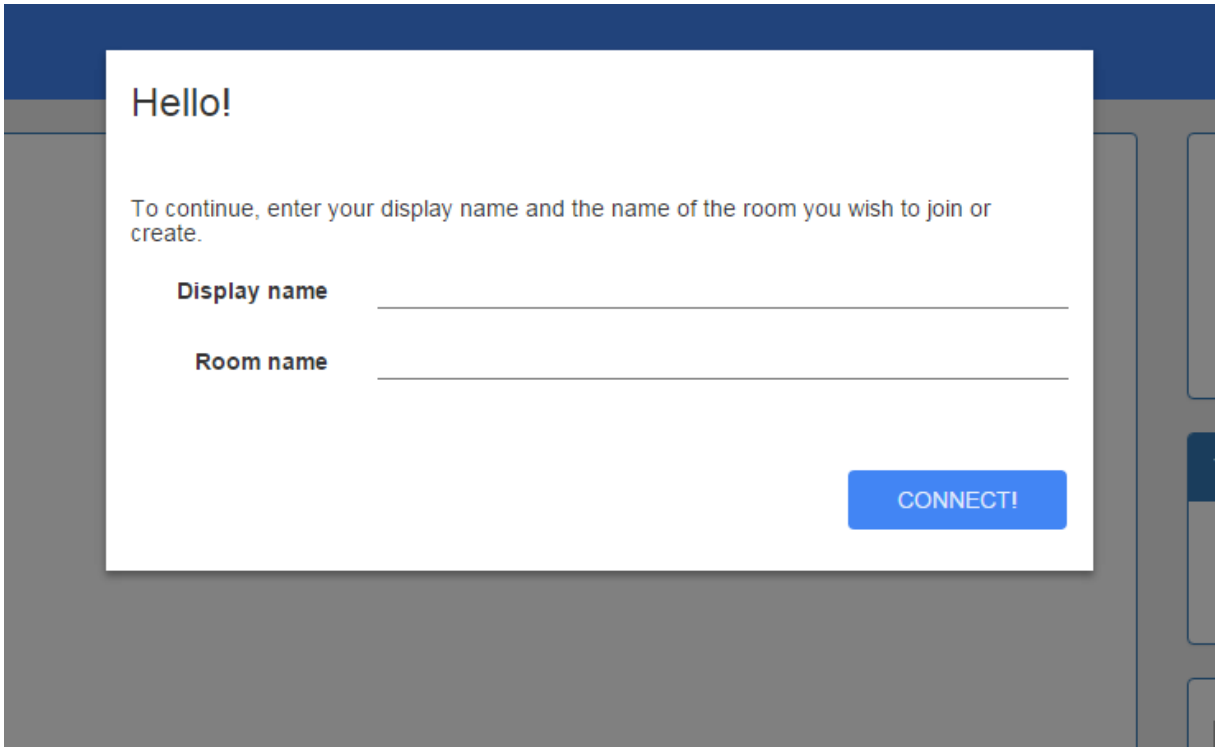
Uporabniški vmesnik je sestavljen iz treh osnovnih gradnikov, kot je razvidno na sliki 4.2:

- Velikega sredinskega okvirja, ki vsebuje prenos slike s spletne kamere uporabnika na drugi strani aplikacije.
- Okvir na desni, ki prikazuje ime klepetalnice in nazadnje izmenjana besedilna sporočila. Lastna sporočila so poravnana z levi robom, tista, ki jih je poslal drugi uporabnik pa so poravnana z desnim robom.
- Majhen okvir v zgornjem desnem kotu, ki vsebuje prenos slike s spletne kamere uporabnika z namenom, da lahko le-ta vidi, če njegova kamera deluje pravilno in če je kakovost slike ustrezna.



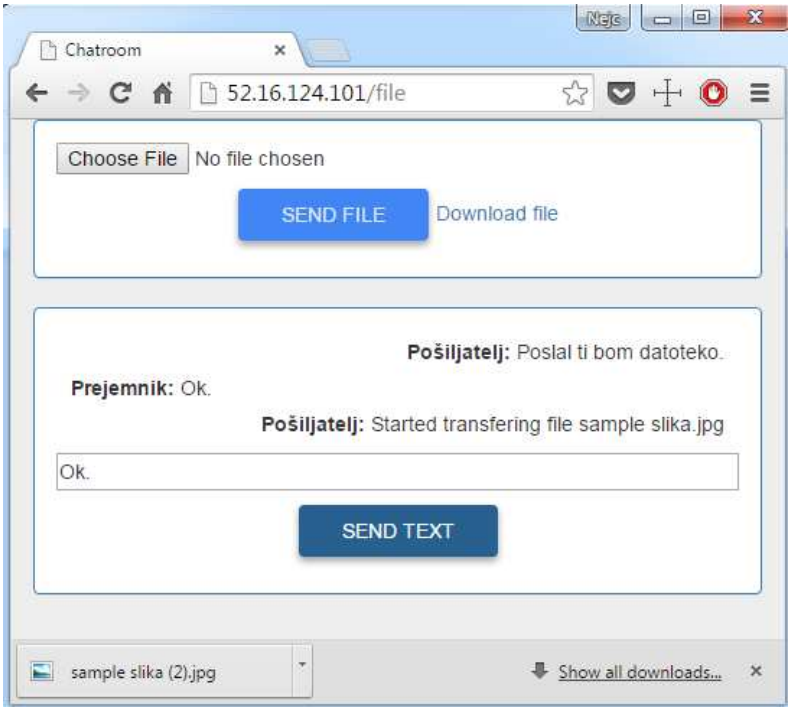
Slika 4.2: Glavni gradniki spletne aplikacije.

Takoj po obisku spletne strani se uporabniku prikaže okence z vnosnima poljema za uporabniško ime in ime sobe v katero se želi povezati. Okence z vnosnima poljema je prikazano na sliki 4.3. Po tem, ko v sobo vstopi še drugi uporabnik se lahko medsebojno povežeta in začneta komunicirati prek video kanala.



Slika 4.3: Pojavno okno, za vpis potrebnih podatkov.

Aplikacija za pošiljanje datotek prek neposrednega kanala WebRTC je ločena od video klepeta in je zaradi tega tudi bolj preprosta. Njen uporabniški vmesnik je prikazan na sliki 4.4. Po vpisu podatkov potrebnih za vstop v sobo, uporabnik počaka na uspešno povezavo drugega uporabnika. Ko se drugi poveže, lahko prvi izbere datoteko, ki jo želi poslati ter pritisne na gumb za začetek. Po izvedenem prenosu, se datoteka drugemu uporabniku samodejno shrani na disk. Ves čas pa si lahko uporabnika izmenjujeta besedilna sporočila.



Slika 4.4: Pogovor dveh uporabnikov in prenos slikovne datoteke.

4.1 Razvoj aplikacije

Opis razvoja spletne aplikacije smo razdelili na dva dela, opis razvoja na strani odjemalca in opis razvoja na strani strežnika.

4.2 Razvoj aplikacije na strani odjemalca

Glavna opravila aplikacije na strani odjemalca so: komunikacija s strežnikom, vzpostavitev povezave z drugimi uporabniki, pošiljanje podatkov in upravljanje uporabniškega vmesnika. Aplikacija je napisana v jezikih HTML in JavaScript ter se izvaja v brskalniku. Ogrodje uporabniškega vmesnika je sestavljeno iz elementov HTML, ki imajo določene razrede zato, da jih ustrezno oblikuje knjižnica Bootstrap.

Na vrh smo postavili orodno vrstico, ki ima gumba za izhod iz sobe in utišanje prenesenega video posnetka ter naslov aplikacije. V sledečem odseku kode je predstavljeno ogrodje HTML za prikaz orodne vrstice. Oblikovanje ni bilo potrebno, saj je za to poskrbela knjižnica Bootstrap.

```
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle collapsed">
        <span class="sr-only">Toggle navigation</span>
      </button>
      <a class="navbar-brand">Video chat</a>
    </div>

    <div class="collapse navbar-collapse">
      <ul class="nav navbar-nav">
        <li><a id="leave-room" href="#">Leave</a></li>
        <li><a id="mute-audio" href="#">Mute</a></li>
      </ul>
    </div>
  </div>
</nav>
```

Pod orodno vrstico se nahaja glavni okvir, ki zavzame preostali prostor na zaslonu. Z uporabo mrežne razporeditve, ki jo ponuja knjižnica Bootstrap, je glavni okvir razdeljen na dva stolpca. Takšna razporeditev razdeli širino okvirja na dvanajst delov, katere razvijalec dodeli notranjim elementom. V našem primeru levi stolpec vsebuje veliki video posnetek drugega uporabnika in zato zavzema devet od dvanajstih delov.

```
<div class="col-md-9">
    <div class="panel panel-primary away-video-container">
        <video id="away-video"></video>
    </div>
</div>
```

Desni stolpec pa zavzema preostale tri dele in vsebuje okvir za besedilni klepet, gumb za povezovanje in manjši videoposnetek uporabnika. Za obrobo in barvanje okvirjev teh elementov sem uporabil gradnik panel iz knjižnice Bootstrap. V sledečem odseku kode je predstavljena koda za postavitev in oblikovanje desnega stolpca uporabniškega vmesnika.

```
<div class="col-md-3">
    <div class="panel panel-primary home-video-container">
        <video id="home-video"></video>
    </div>
    <div class="panel panel-primary">
        <div class="panel-heading room-name">
            Your room:
        </div>
        <div class="panel-body room-members">
            <button id="makeCall" class="btn btn-primary btn-raised"
disabled="disabled">Connect with peer</button>
        </div>
    </div>
    <div class="panel panel-primary text-chat-container">
        <div class="panel-body">
```

```
            <div class="panel-text-containertext-containertext-
containerbody text-chat">
            </div>
            <input type="text" id="text-container">
            <button id="send-text" class="btn btn-primary btn-
raised">Send text</button>
        </div>
    </div>
</div>
```

Spletna stran ima vdelano kodo s knjižnico Socket.IO, ki samodejno poskrbi za povezavo s strežnikom. Za njo se po vrstnem redu izvajanja nahaja naša koda. Ta po eni sekundi sproži izris pojavnega okna s klicem funkcije »modal« in doda odzivno funkcijo joinRoom za vstop v klepetalnico. Funkcija se sproži po pritisku na gumb enter ali po kliku na gumb. Sledeča koda prikazuje izris pojavnega okna in dodajanje odzivnih funkcij za vstop v sobo.

```
setTimeout(function() {
      $container.find("#myModal").modal();
      $container.find("#myModal button").click(joinRoom);

      $(document).bind("keypress.key13", function(e) {
          if(e.which == 13) {
              joinRoom();
          }
      });
}, 1000);
```

Funkcija joinRoom si shrani vneseno uporabniško ime in ime klepetalnice ter prikaže opozorila v primeru, da kateri od podatkov manjka. Shranjene uporabniške podatke pošlje strežniku s klicem funkcije socket.emit('roomName', userName, roomName) in se naroči na sporočila z naslovom »peer joined room« in »joinedRoom«. Po prejetem slednjem sporočilu, sproži zahtevo za vklop uporabnikove kamere in mikrofona s klicem funkcije getUserMedia, zaradi česar se v uporabnikovem brskalniku pojavi okno za odobritev zahteve. Ta korak ni

mogoče izvesti samodejno, saj standard HTML5 zahteva, da se za to odloči uporabnik. To zlonamernim razvijalcem preprečuje snemanje brez uporabnikovega dovoljenja.

```
var joinRoom = function() {
	userName = $container.find("#myModal #userNameInput").val();
	roomName = $container.find("#myModal #roomNameInput").val();

	if ((!userName || userName === '') && (!roomName || roomName
=== '')) {
		$container.find("#myModal .alert.no-data").show();
		return;
	}
	socket.on("joinedRoom", function(doc) {
		peerName = doc.peerName;
		isInitiator = doc.isInitiator;
		$container.find("div.room-name").append(doc.roomName);
		isChannelReady = !doc.isInitiator;

		if (doc.isInitiator) {
			console.log("You created the room.");
		} else {
			console.log("You have joined an existing room.");
		}

		getUserMedia.call(navigator, {video: true, audio: true},
handleUserMedia, handleError);
	});
	socket.on("peer joined room", function(doc) {
		isChannelReady = true;
		peerName = doc.peerName;
	});
```

```
    socket.emit('roomName', userName, roomName);
    $modal.modal('toggle');
}
```

Po izvedbi tega izseka kode, sta oba uporabnika povezana s strežnikom s povezavo WebSockets in včlanjena v skupno klepetalnico. Oba lahko strežniku pošiljata sporočila in jih tudi prejmeta. Za izmenjavo omrežnih podatkov in vzpostavitev povezave z uporabo vmesnika WebRTC pa potrebujeta medsebojen komunikacijski kanal. Tak kanal lahko postavita s pomočjo strežnika, ki deluje kot posrednik sporočil. V našem primeru spletni strežnik razpošilja vsa prejeta sporočila enega uporabnika do ostalih, ki so včlanjeni v isto klepetalnico. Ta sporočila aplikacija na strani odjemalca pošilja drugim odjemalcem s klicem funkcije:

```
socket.emit('message to peers', data);
```

Po izvedbi zgornjih izsekov kode imamo izpolnjene vse pogoje za vzpostavitev neposredne povezave WebRTC. Po pritisku na gumb za začetek povezovanja se pokliče funkcija start. V sledečem odseku kode se inicializirata objekta RTCPeerConnection in DataChannel.

```
var start = function() {
    if (!isStarted && typeof localStream != "undefined" &&
isChannelReady) {
        pc = new webkitRTCPeerConnection(servers, {optional:
[{RtpDataChannels: true}]});
        pc.onicecandidate = handleIceCandidate;
        pc.onaddstream = handleRemoteStreamAdded;
        pc.addStream(localStream);
        isStarted = true;

        textChannel = pc.createDataChannel("textChat", null);
        textChannel.onerror = handleError;
        textChannel.onmessage = receiveText;
```

```
            if (isInitiator) {
                  pc.createOffer(setLocalAndSendMessage, handleError);
            }
      }
};
```

V prejšnjem odseku kode se pokliče vmesnik RTCPeerConnection, ki vrne objekt, ki vsebuje funkcije za pošiljanje video toka, vzpostavitev binarne povezave DataChannel in druge kontrolne funkcije. Izvede se tudi klic:

```
pc.createOffer(setLocalAndSendMessage, handleError);
```

Ta sporoči drugemu uporabniku, da se je zagnal proces vzpostavitve povezave in mu pošlje objekt sessionDescription, ki vsebuje podatke kot na primer: vrsta uporabljenega kodeka, najvišjo dovoljeno pasovno širino in časovne informacije videa in zvoka.

Ko drugi uporabnik prejme opis seje in povabilo za začetek povezave, se shranijo prejeti podatki, izvede funkcija maybeStart, ki se je pri prvemu uporabniku že izvedla in pošlje odgovor s potrjenim opisom seje.

```
if (message.type === 'offer') {
      if (!isInitiator && !isStarted) {
            start();
      }
      pc.setRemoteDescription(new RTCSessionDescription(message));
      pc.createAnswer(setLocalAndSendMessage, null, sdpConfig);
}
```

Po izmenjavi opisnih sporočil, se začne pošiljati tudi tok podatkov s spletne kamere in mikrofona, na kar pa se aplikaciji odzoveta s klicem funkcije handleRemoteStreamAdded. Ta ustvari referenco na prejet tok podatkov v obliki objectURL in elementu za prikaz video posnetkov nastavi vir. Po izvedbi sledečega izseka kode se uporabnikoma na spletni strani prikaže slika in zvočniki začnejo predvajati zvok.

```
var handleRemoteStreamAdded = function(event) {
      $awayVideo.src = window.URL.createObjectURL(event.stream);
```

```
        remoteStream = event.stream;
        $awayVideo.play();
    };
```

4.2.1 Pošiljanje datotek in besedilnih sporočil

Za pošiljanje drugih tipov podatkov se uporablja vmesnik RTCDataChannel. Vzpostavitev povezave RTCPeerConnection je v tem primeru enaka, le da nam ni potrebno zahtevati uporabe kamere ter pošiljati njenega toka podatkov. Namesto tega pa izvedemo funkcijo createDataChannel, ki vrne objekt, ki predstavlja podatkovni kanal za pošiljanje binarnih ali tekstovnih podatkov. Funkcija prejme vhodni parameter, ki določi ali naj povezava poteka s uporabo protokola TCP ali UDP. Prvi je počasnejši vendar poskrbi za pravilni vrstni red podatkov in zagotavlja, da je vsebina nespremenjena, drugi pa je hitrejši, vendar ne zagotavlja vrstnega reda ali točnosti vsebine. Za prenos datotek smo uporabili TCP, za sporočila pa UDP.

```
// Ustvari podatkovni kanal za prenos besedila
textChannel = pc.createDataChannel("textChat", null);
textChannel.onerror = handleError;
textChannel.onmessage = receiveText;

// Ustvari podatkovni kanal za prenos datotek
fileChannel = pc.createDataChannel("fileTransfer", {reliable:
true});
fileChannel.onerror = handleError;
fileChannel.onmessage = receiveFile;
```

Po izvedbi zgornje kode se vzpostavi podatkovni kanal RTCDataChannel in nastavi odzivne funkcije, ki se prožijo po prejetih podatkih. Besedilna sporočila se pošilja s klicem:

```
textChannel.send('Vsebina sporočila');
```

Pošiljanje datotek pa je bolj zapleteno, saj so datoteke lahko prevelike, da bi jih vseboval le en TCP okvir. Zaradi tega brskalniki omejujejo velikost sporočila. Vmesnik WebRTC ne omogoča samodejnega razdeljevanja prevelikih sporočil, zato mora za to poskrbeti razvijalec.

Za izbiro datoteke, ki jo uporabnik želi poslati, smo na spletno stran dodali element HTML:

```
<input type='text'>
```

Po tem, ko uporabnik izbere datoteko, se referenca nanjo shrani v objekt tega elementa. Z uporabo vmesnika za branje datotek si shranimo velikost datoteke in njeno ime ter oba podatka po besedilnem kanalu pošljemo drugemu uporabniku. Napisali smo funkcijo, ki datoteko prebere in razdeli na enako velike dele, katere prenese v časovnih razmikih, dolgih 750ms. V naslednjem odseku kode je predstavljeno branje datoteke in pošiljanje njenih delov.

```
var file = document.getElementById('file-selector').files[0];
textChannel.send(JSON.stringify({
      size: file.size,
      name: file.name
}));
var reader = new window.FileReader();
var chunkSize = 750;

function onReadAsDataURL(event, text) {
    var data = {};
    if (event) {
      // save the text on first call
      text = event.target.result;
    }
    if (text.length > chunkSize) {
        data.message = text.slice(0, chunkSize);
    } else {
        data.message = text;
        data.last = true;
    }
    fileChannel.send(JSON.stringify(data));
    var remainingDataURL = text.slice(data.message.length);

    if (remainingDataURL.length) {
      setTimeout(function () {
            // continue transmitting
               onReadAsDataURL(null, remainingDataURL);
```

```
            }, 500);
        }
}
reader.onload = onReadAsDataURL;
reader.readAsDataURL(file);
```

Prejemnik v razmiku prejema dele datoteke in jih shranjuje na sklad. Ko prejme zadnji del, združi sklad in sproži shranjevanje datoteke na disk, na kar brskalnik obvesti uporabnika.

4.3 Razvoj aplikacije na strani strežnika

Vloga strežnika je, da odgovori na zahtevek za prenos spletne strani, povezovanje uporabnikov v sobe in posredovanje sporočil za izmenjavo omrežnih podatkov med odjemalci. Zaradi lažjega razvoja smo izbrali strežnik in izvajalno okolje Node.js, za katerega so programi napisani v jeziku JavaScript. Ker se isti jezik uporablja tudi na strani odjemalca, je prejemanje in pošiljanje sporočil bolj preprosto, saj so podatki v isti obliki in jih ni treba pretvarjati. Za odgovarjanje na zahtevke HTTP in streženje datotek HTML, CSS in JavaScript smo si pomagali z ogrodjem Express. Koda s katero zaženemo spletni strežnik:

```
var app = require('express')();
var express = require('express');
app.use(express.static('public'));

app.get('/', function(req, res){
     res.sendFile(__dirname + '/calling.html');
});

app.get('/file', function(req, res) {
     res.sendFile(__dirname + '/fileSharing.html')
});
```

Ko odjemalec prenese spletno stran in se poveže z uporabo protokola WebSockets, začne strežnik čakati na sporočilo s podatkom o imenu uporabnika ter imenom sobe v katero želi vstopiti. Sobe so predstavljene kot razred Room, ki nosi podatke o včlanjenih uporabnikih. V naslednjem odseku kode je predstavljen del razreda Room in njegovi glavni metodi »join« in »leave«.

```
module.exports = function (roomName, clientId, limit) {
    var data = {
        members: [],
        roomName: false,
        closed: false,
        limit: 2
    };
```

```
      data.roomName = roomName;
      data.members.push(clientId);

      if (limit) { data.limit = limit; }

      var join = function(clientId) {
            if (!data.closed) {
                  data.members.push(clientId);
            } else {
                return false;
            }
            if (data.members.length >= data.limit) {
                  this.lock();
            }
            return true;
      };
      var leave = function(clientId) {
            var index = data.members.indexOf(clientId);
             if (index) {
                  data.members.splice(index, 1);
             }
    };
      return {leave: leave, join: join};
}
```

Posredovanje prejetih sporočil vsem članom sobe je z uporabo knjižnice Socket.IO preprosto:

```
socket.on('message', function(message) {
      socket.broadcast.to(room).emit('message to peers', message);
});
```

Poglavje 5 Varnostni vidik projekta WebRTC

V sledečem poglavju bom predstavil s čim vmesniki WebRTC omogočajo varno komunikacijo in na kaj mora biti pozoren uporabnik komunikacijskih aplikacij, ki uporabljajo vmesnike WebRTC.

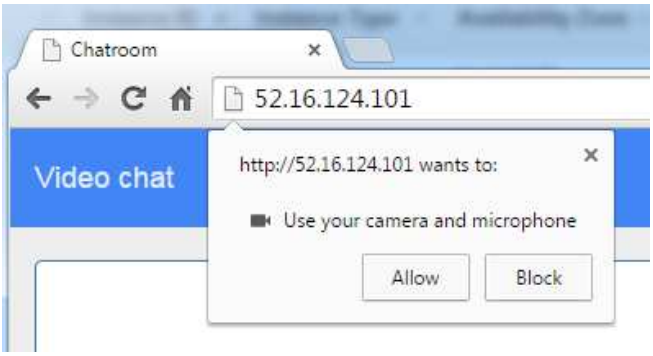
5.1 Spletni brskalniki

Projekt WebRTC je že v osnovi zasnovan tako, da omogoča vendar ne zagotavlja varne komunikacije. S perspektive uporabnika je najbolj pomembno, da uporablja spletni brskalnik, katerega izdelovalec je vreden zaupanja in da je bil brskalnik nameščen na način, ki zagotavlja, da je nameščeni brskalnik takšen kot ga ponuja izdelovalec. To pomeni, da mora uporabnik prenesti brskalnik s spletne strani izdelovalca, ki mu zaupa. Če bi uporabnik prenesel brskalnik s strani, ki ne zagotavlja njegove neoporečnosti, bi brskalnik lahko bil tako spremenjen, da bi popolnoma onemogočal vse varnostne tehnike vmesnikov WebRTC. Vendar pa to velja tudi za splošno uporabo brskalnikov, ne le za komuniciranje kot ga omogoča naša aplikacija.

Prednost vmesnikov WebRTC je tudi to, da so vgrajeni v brskalnike, ki se pogosto posodabljajo. Izdelovalca spletnih brskalnikov Mozilla in Google sta znana po tem, da imata vzpostavljen stalen in samodejen proces posodobitev. V primeru, da se odkrije varnostno pomanjkljivost v protokolih, ki jih uporabljajo vmesniki WebRTC, lahko izdelovalci hitro ukrepajo in popravijo pomanjkljivosti ter samodejno posodobijo že nameščene brskalnike.

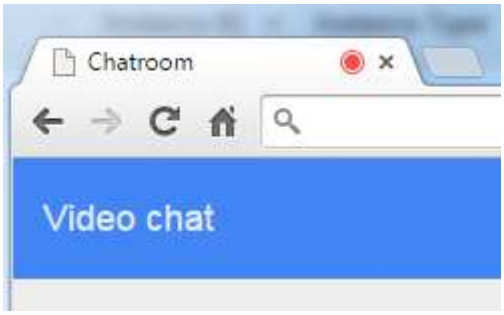
5.2 Dostop do spletne kamere in mikrofona

Zlonamernež bi lahko izdelal spletno aplikacijo, ki začne zajemati sliko s pomočjo uporabnikove spletne kamere takoj po obisku spletne strani. Zaradi te nevarnosti morajo brskalniki vsakič znova pokazati uporabnikom opozorilo, da obiskana spletna stran želi uporabljati mikrofon ali spletno kamero. Opozorilo brskalnika je prikazano na sliki 5.1. Uporabnik lahko odobri uporabo teh virov ali pa zavrne.



Slika 5.1: Pojavno okno, ki uporabniku omogoči odobritev ali zavrnitev zahteve po virih.

Poleg tega morajo brskalniki prikazati tudi kdaj spletna stran uporablja kamero ali mikrofon. Na sliki 5.2 je primer rdeče pike, ki sporoča, da spletna stran snema zvok ali sliko.



Slika 5.2: Z rdečo piko brskalnik prikazuje, da spletna stran zajema sliko s kamere.

5.3 Šifriranje prenosa podatkov

Morebitna nevarnost aplikacij za realno časovno komunikacijo je prestrezanje poslanih podatkov po nezavarovanem komunikacijskem kanalu. Za preprečitev te nevarnosti morajo šifriranje uporabljati vsi vmesniki WebRTC.

Ker nobeden od vmesnikov WebRTC ne omogoča izmenjave podatkov za vzpostavitev povezave, mora za varno delovanje v tem koraku poskrbeti razvijalec. Šifriranje v tem koraku je kritično za nadaljnje varno delovanje aplikacije, ker si uporabniki v času vzpostavljanja povezave izmenjajo ključne informacije za šifriranje sledečega prenosa podatkov.

Za prenos slike in zvoka se uporablja varnostni protokol SRTP, za prenos datotek in drugih podatkov pa protokol DTLS.

Za šifriranje podatkov, ki se prenašajo z uporabo vmesnika RTCDataChannel, WebRTC uporablja varnostni protokol DTLS. DTLS je uveljavljen protokol, ki se lahko uporablja tudi

za prenos spletne pošte in je zasnovan na protokolu TLS. DTLS deluje tudi v primeru, da prenos podatkov poteka po vmesnem strežniku TURN, saj strežniki TURN ne pregledujejo vsebine prispelih datagramov, vendar te le posredujejo.

Za šifriranje prenosa slike in zvoka RTCPeerConnection uporablja protokol SRTP, ker je ta hitrejši in ima manjše okvirje. SRTP je varna različica protokola RTP, ki se uporablja za prenos realno časovnih podatkov. SRTP ima pomanjkljivost, saj šifrira le vsebino datagramov in ne njegove glave. V glavi pa se nahajajo nekateri podatki, ki bi jih zlonamerni poslušalec zlorabil, na primer: podatek o jakosti prenesenega zvoka.

Poglavje 6 Sklepne ugotovitve

V diplomski nalogi smo razvili spletno aplikacijo, ki omogoča uporabnikom vzpostavljanje video klicev, izmenjavo besedilnih sporočil in pošiljanje datotek.

Cilj naloge je olajšati komunikacijo med več uporabniki. Aplikacijo smo razvili tako, da je preprosta za uporabo, saj mora uporabnik le nekajkrat klikniti in vnesti uporabniške podatke. Osnovana je na najnovejših tehnologijah, ki jih večina brskalnikov že omogoča, v prihodnosti pa bo to omogočal skoraj vsak brskalnik. Ker se večina programske logike za povezovanje nahaja na strani odjemalca, smo dosegli, da je delo, ki ga opravlja strežnik, minimalno in zato ne potrebuje večjih zmogljivosti.

Med raziskovanjem tematike in razvojem aplikacije smo ugotovili, da so uporabljene tehnologije primerne za izdelavo spletnih aplikacij, ki omogočajo sodelovanje uporabnikov pri različnih opravilih, kot na primer: skupinsko urejanje besedilnih datotek, risanje, programiranje in igranje iger. Z uporabljenimi tehnologijami bi lahko aplikacijo dodatno nadgradili tako, da bi omogočala skupinske video pogovore, oddajanje spletne oddaje v načinu eden proti mnogo (one-to-many) ali celo porazdeljeno razpošiljanje velikih datotek med veliko uporabniki.

V nadaljevanju razvoja bi si želeli odpraviti nekatere pomanjkljivosti, ki pa jih zaradi pomanjkanja informacij o delovanju brskalnikov zdaj še nismo uspeli. Omogočili bi hitrejše pošiljanje datotek in hkratno pošiljanje binarnih podatkov z uporabo vmesnika RTCDataChannel ter prenos toka s spletne kamere. Da bi dosegli še večjo množico potencialnih uporabnikov, bi omogočili delovanje spletne aplikacije v brskalnikih Mozilla Firefox in Opera katerih vmesniki WebRTC se nekoliko razlikujejo.

Med razvojem smo spoznali, da se nabor programskih vmesnikov WebRTC lahko uporabi pri zelo različnih načinih prenašanja podatkov in da bo skupaj z modernimi standardi, kot sta HTML5 in WebSockets ter razvojem brskalnikov omogočal izdelavo funkcionalnih realnočasovnih spletnih aplikacij, ki bodo lahko sčasoma nadomestile tradicionalne aplikacije za video klepet, besedilno sporočanje in prenos datotek.

Literatura

[1] D. Flanagan. JavaScript: The Definitive Guide, Fifth Edition, O'Reilly Media, Inc., 2006.

[2] R. Manson. Getting Started with WebRTC. Packt Publishing Ltd. 2013

[3] B. McLaughlin. What Is Node?, O'Reilly Media, Inc. 2011

Spletni viri:

[4] Web API Interfaces: WebSockets, Dostopno 14. 9. 2015 na:

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[5] Socket.IO docs, Dostopno 14. 9. 2015 na:

http://socket.io/docs/

[6] Bootstrap Overview, Dostopno 14. 9. 2015 na:

http://getbootstrap.com/css/#grid

[7] Getting Started with WebRTC, Dostopno 15. 9. 2015 na:

http://www.html5rocks.com/en/tutorials/webrtc/basics/

[8] WebRTC Statistics and Metrics, Dostopno 21. 9. 2015 na:

http://www.webrtcstats.com

[9] WebRTC in the real world: STUN, TURN and signaling, Dostopno 16. 9. 2015 na:

http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/

[10] Web API Interfaces: RTCPeerConnection, Dostopno 16. 9. 2015 na:

https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection

[11] WebRTC data channels, Dostopno 16. 9. 2015 na:

http://www.html5rocks.com/en/tutorials/webrtc/datachannels