Tampereen ammattikorkeakoulu

Yleiskäyttöisen WebRTC komponentin suunnittelu ja toteutus

Opinnäytetyö
30.05.2020
Tietojenkäsittely
Web-palvelut

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Web-palvelut

MUSTANIEMI, JONI
Yleiskäyttöisen WebRTC komponentin suunnittelu ja toteutus

Opinnäytetyö 39 sivua
Toukokuu 2020

Työn tarkoituksena oli suunnitella ja toteuttaa TypeScriptillä yleiskäyttöinen ja helposti laajennettavissa oleva WebRTC komponentti, jolla pystytään kommunikoimaan verkon yli ja jakamaan ääni-, video- tai muuta dataa usean asiakkaan kanssa samanaikaisesti. Tavoitteena oli helpottaa WebRTC:stä kiinnostuneiden ohjelmoijien kehitysprosessia tarjoamalla valmis komponentti, jonka voi upottaa olemassa olevaan järjestelmään tai käyttää pohjana omalle toteutukselle.

Työssä tehtiin avoimen lähdekoodin WebRTC komponentin joka täyttää yleiskäytettävyyden määritelmän. Komponentin suunnitelussa on otettu huomioon jatkokehitys sekä dokumentointilogiikan pysyminen yhtenäisenä myös laajentamisen jälkeen hyödyntämällä luokkapohjaisen ohjelmistokehityksen ja UML-mallinnuksen mukaisia menetelmiä komponentin dokumentoimiseen.

Toteutettu komponentti on yleiskäyttöisen luonteensa takia monikäyttöinen ja sitä voidaan hyödyntää reaaliaikaisten kommunikointityökalujen rakentamisessa, joka voi olla mitä tahansa digitaalisesta videokonferenssisovelluksesta yksinkertaisempaan tekstipohjaiseen viestittelyyn perustuva ohjelma. Koska komponentin käyttäminen on täysin vapaata, voidaan siitä ottaa myös erillisiä osioita, jolloin WebRTC:n kanssa kamppailevat ohjelmoijat voivat eriyttää esimerkiksi signalointi- tai yhteyslogiikan omaan toteutukseensa.

Komponenttia voidaan jatkokehittää yhteyslogiikan osalta, mikäli Chrome tulevaisuudessa tukee rollback-toiminnallisuutta, jolloin toisesta yhteysolioista voitaisiin luopua. Tämän komponentin toteutushetkellä se ei kuitenkaan vielä ole mahdollista.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Web Services

MUSTANIEMI, JONI
The Design and Implementation of a General Purpose WebRTC Component

Bachelor's thesis 39 pages
May 2020

The purpose of this work was to design and implement a general-purpose, easily extensible WebRTC component with TypeScript. The component needed to be able to communicate over a network and share voice, video or other data with several clients simultaneously. The goal was to facilitate the development process for programmers interested in WebRTC by providing a ready-made component, which can be embedded in an existing system or used as a basis for their own implementation.

In this work, an open source WebRTC component that meets the definition of general usability and is designed to take into account further development was created. The component keeps the documentation logic consistent even after expansion by utilizing class-based software development and UML modeling methods to document the component.

Due to its general-purpose nature, the implemented component is versatile and can be utilized in building real-time communication tools, which range from a digital video conferencing application to a simpler text-based communication. Since the use of the component is free, invidual sections can also be taken from it, allowing programmers struggling with WebRTC to differentiate, for example, signaling or connection logic into their own implementation.

The component can be further developed in terms of connection logic if Chrome supports rollback functionality in the future, in which case another connection object could be dropped. However, at the time of implementation of this component it was not possible.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

SISÄLLYS

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

LYHENTEET

WebRTC	Web Real-Time Communication
MDN	Mozilla Developer Network
SDP	Session Description Protocol
NAT	Network Address Translation
TURN	Traversal Using Relays around NAT

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

1 JOHDANTO

Organisaatioissa on tarve kommunikoida reaaliaikaisesti verkon yli. Tätä varten on aiemmin ollut erillisiä sovelluksia ja selainlaajennuksia sekä tapauskohtaisia palvelinpään ratkaisuja standardiratkaisun sijaan. Tämä on ollut ongelma erityisesti www-pohjaisten sovellusten tekemisessä, sillä verkkoselaimet eivät ole pystyneet kommunikoimaan reaaliaikaisesti toistensa kanssa suoraan ilman ulkopuolista ratkaisua, etenkään vertaisverkossa.

Työn tarkoituksena on suunnitella ja toteuttaa TypeScriptillä yleiskäyttöinen ja helposti laajennettavissa oleva WebRTC komponentti, jolla pystytään kommunikoimaan verkon yli ja jakamaan ääni-, video- tai muuta dataa usean asiakkaan kanssa samanaikaisesti. Työn tavoite on helpottaa WebRTC:stä kiinnostuneiden ohjelmoijien kehitysprosessia tarjoamalla valmis komponentti, jonka voi upottaa olemassa olevaan järjestelmään tai käyttää pohjana omalle toteutukselle.

Lähdemateriaalina aion käyttää MDN Web Docsin WebRTC osiota ja muita avoimen lähdekoodin kehittäjäyhteisön materiaaleja sekä verkkolähteitä. Opinnäytetyössä kuvaan reaaliaikaisuuden olemassa olevia toteutustapoja, WebRTC:n toiminnallisuutta, komponentille asetettuja vaatimuksia sekä toteutuksen osa-alueita.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

2 REAALIAIKAISUUS JA WEB-SOVELLUKSET

Nykymaailmassa on enemmän internettiä käyttäviä laitteita, kuin koskaan aikaisemmin. Yhä useampi toiminnallisuus, joka ennen tehtiin paperilla ja kynällä tai paikan päällä siirtyy internettiin erilaisten applikaatioiden taakse. Tarve reaaliaikaiselle datalle ja sen käsittelylle on tullut jäädäkseen.

Termiä reaaliaikainen voidaan käyttää, kun tapahtuma on käyttäjän näkökulmasta lähes huomaamaton (Realtime computing n.d). Reaaliaikaiselle datalle ominaista on, ettei sitä tallenneta vaan se välitetään käyttäjälle mahdollisimman nopeasti (Realtime Data 13.6.2018). Reaaliaikainen datan käsittely voidaan jakaa kolmeen osa-alueeseen (Kuvio 1): reaaliaikaiseen datan keräämiseen, reaaliaikaiseen analysointiin ja prosessointiin sekä reaaliaikaiseen tietokantahakuun (Big Data Technologies and Cloud Computing 2015).

Kuvio 1. Wenhong Tian, Yong Zhao, in Optimized Cloud Resource Management and Scheduling, 2015. [Viitattu 12.1.2020]. Saatavissa: https://www.sciencedirect.com/topics/computer-science/real-time-computing

Seuraavassa osiossa käydään läpi olemassa olevia reaaliaikaisen kommunikoinnin tekniikoita kuten Http Polling, Long Polling, WebSockets ja Server Sent Events sekä sitä miten nämä ovat vaikuttaneet maailmassa, jossa tarve reaaliaikaiselle datalle on alati kasvava.

2.1 HTTP Polling ja Long Polling

Internetin alkuaikoina HTTP oli järkevä valinta yksinkertaiseksi pyyntö ja vastaus protokollaksi koska se oli suunniteltu jakamaan jäsenneltyjä dokumentteja niistä kiinnostuneille lukijoille. Tekniikka oli siihen aikaan paljon rajallisempaa eikä yhteyden pitäminen auki ollut järkevä ratkaisu (Polling By Ably n.d). HTTP

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

kyselytekniikassa käyttäjä lähettää kyselyitä palvelimelle ja palvelin reagoi lähettämällä vastauksia takaisin käyttäjälle. Viive näissä pyynnöissä voi olla kuitenkin jopa 10 sekuntia (Http Polling 30.11.2019), joten nykypäivän maailmassa, jossa dataa liikkuu huomattavasti enemmän ja useammin kuin internetin alkuaikoina, ei tällainen viive ole hyväksyttävä.

HTTP kyselytekniikassa on myös toinen ratkaisu, jota kutsutaan pitkäksi kyselyksi. Pitkä kysely on käytännössä lyhyen kyselyn tehokkaampi muoto, jossa yhteys pidetään auki, kunnes uutta informaatiota ilmenee, jolloin se lähetetään käyttäjälle ja jäädään odottamaan taas uutta informaatiota (Long Polling 30.11.2019). Vaikka tämä aikanaan onkin ollut hyvä ratkaisu ovat nykyajan vaatimukset datan lähettämiselle ja vastaanottamiselle kasvaneet. Koska jokainen pyyntö ja vastaus sisältää kaikki HTTP-otsikot voi suuri osa kokonaisdatasta muodostua näistä (Long Polling Issues 4.1.2011). Myös tietyissä tilanteissa voi esiintyä ongelmia pakettien toimittamisessa sillä yhden asiakkaan vastaus viestin vastaanottamisesta voi aiheuttaa sen, että toinen ei koskaan saa haluttua viestiä (WebSocket vs Long Polling 4.6.2019). Maailmassa, jossa luotamme elektroniikan ja tekniikan haltuun lähes kaiken, ei tällainen epäluotettavuus ole vaihtoehto.

2.2 WebSocket

WebSocket protokolla nosti tullessaan internetin viestinnän uudelle tasolle ja mahdollisti synnyn aidosti reaaliaikaiselle internetille. Reaaliaikainen internetti oli olemassa jo ennen WebSockettia, mutta se oli hidas ja oli mahdollista modifioimalla olemassa olevia teknologioita, joita ei ollut tarkoitettu reaaliaikaiseen viestintään. Tähän saatiin ratkaisu vuoden 2008 puolivälissä, kun Michael Carter ja Ian Hickson tuskastuivat asioiden vaikeuteen todella robustien toteutuksien kanssa toimiessaan. erinäisten tahojen yhteistyöllä he suunnittelivat uuden standardin modernille, reaaliaikaiselle ja kaksisuuntaiselle kommunikaatiolle internetissä. WebSocket oli syntynyt. (WebSockets By Ably 8/2018)

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

WebSocket on vastaus aikaisempien HTTP kyselytekniikoiden yleiskustannuksiin. WebSocketin avulla pystytään luomaan pieniviiveinen pysyvä yhteys, joka tukee sekä käyttäjän, että palvelimen aloittamia tapahtumia ja dataa voidaan liikuttaa milloin tahansa (Introduction to WebSockets 18.10.2013). Täten WebSocket sopii ominaisuuksiltaan hyvin modernin maailman kasvavaan reaaliaikaisuuden tarpeeseen.

2.3 Server Sent Events

Olemme kovaa vauhtia etenemässä kohti tapahtumavetoista maailmaa ja siinä maailmassa palvelimien lähettämät tapahtumat täyttävät erittäin tietyn markkinaraon: avoin, kevyt ja tilaajakohtainen protokolla tapahtumavetoisille datavirroille. Idea palvelimen lähettämissä tapahtumissa on yksinkertainen: selain voi tilata palvelimen luoman tapahtumavirran ja saada uutta informaatiota aina tapahtuman yhteydessä. (Server-Sent Events By Ably).

WebSocketin ja palvelimen lähettämien tapahtumien välillä yksi merkitsevä ero. Toisin kuin WebSocket, palvelimen lähettämät tapahtumat ovat yksisuuntaisia, palvelimelta selaimelle. Tämä voi kuitenkin olla WebSockettia parempi ratkaisu, jos applikaatiossa on tarve nopeasti päivittyvälle datalle, kuten esimerkiksi osakekurssien hintojen päivitys. (Server-Sent Events 25.1.2018)

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

3 WEBRTC

Teknologian edistyminen kautta historian on aiheuttanut muutoksia tapoihin tehdä asioita ja ihmisten kokoontuminen ei ole poikkeus tässä asiassa. Etätyöskentelyn kasvaessa jatkuvasti kasvaa myös tarve kokouksien järjestämiseen etäyhteyksien kautta.

Tarve virtuaaliselle yhteydelle ja videoneuvotteluille on ollut olemassa jo jonkin aikaa ja ennen tämä saavutettiin Flashin avulla tai vaihtoehtoisesti asentamalla selaimeen erinäisiä lisäosia, mutta nämä olivat ongelmallisia sekä käyttäjän, että kehittäjän näkökulmasta. WebRTC teknologiaa kehitti ensin Global IP Solutions, mutta omistajuuden vaihduttua Googlelle alkoi W3C kehittää WebRTC standardia ja tämän jälkeen WebRTC:tä on alkanut tukemaan muut verkkoselainten jätit, kuten Mozilla ja Opera (History of WebRTC).

WebRTC mahdollistaa verkkosovellusten ja -sivujen tallentaa ja suoratoistaa ääni- ja videodataa sekä muuta satunnaista dataa ilman tarvetta välittäjälle, käyttäjän asentamille lisäosille tai muille kolmannen osapuolen sovelluksille. (WebRTC API 12.3.2020). WebRTC on vastaus teknologian valtaaman maailman huutoon reaaliaikaisuuden tarpeesta.

WebRTC:n toiminta perustuu kolmeen ohjelmointirajapintaan: MediaStream, RTCPeerConnection ja RTCDataChannel. MediaStreamin avulla hallitaan käyttäjän kameraa ja mikrofonia, RTCPeerConnectionilla ääni- ja videopuheluiden sekä yhteydenmuodostamista ja RTCDataChannelillä kaikkea muuta geneeristä dataa. (WebRTC By WebPlatform n.d). Näiden avulla WebRTC:llä kyetään toteuttamaan todella laaja skaala erilaisia sovellutuksia jokaiseen reaaliaikaisen kommunikoinnin tarpeeseen.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

3.1 Turn

Internetissä toimiessa kaksi käyttäjää ovat harvoin samassa verkossa, kun tarve reaaliaikaiselle kommunikaatiolle ilmenee. Tämän takia useimmat WebRTC sovellutukset tarvitsevat palvelimen välittämään tarvittava data käyttäjältä toiselle ja tähän käytetään TURN palvelinta, joka on tähän tarkoitukseen luotu protokolla verkkoliikenteen välittämiseen ja sen käyttöön on nykyään useita vaihtoehtoja, aina itse hostaamisesta pilvipalveluratkaisuihin (TURN Server 28.5.2019).

TURN palvelimen valintaa ja käyttöä ylipäätänsä tulee kuitenkin miettiä jokaisen sovellutuksen kohdalla erikseen, sillä mikäli yhteys voidaan muodostaa käyttäen paikallisverkkoyhteyttä, tulisi sitä suosia ennen TURN palvelimen käyttämistä. Muutoin voidaan joutua tilanteeseen, jossa kaksi käyttäjää, jotka ovat lähellä toisiaan lähettävät mediaa palvelimen läpi, joka on kaukana heistä. Tämä johtaa viiveen kasvamiseen, joka syö reaaliaikaisuuden perusperiaatetta, eli mahdollisimman huomaamatonta viivettä informaation välityksessä käyttäjien välillä.

3.2 Signalointi

Signalointi on WebRTC:n toiminnan kannalta välttämätöntä. Tähän tarvitaan signalointi palvelin, joka välittää käyttäjien välillä SDP-tietoja. WebRTC standardi ei itsesään tarjoa tähän mitään tiettyä menetelmää, vaan on kehittäjän vastuulla valita sovellukseen sopiva signalointitapa.

Signaloinnilla tarkoitetaan viestintäistunnon perustamis-, hallinta- ja lopetusprosesia. Jotta käyttäjät voisivat muodostaa yhteyden, tarvitaan kolmen tyyppisiä tietoja: istunnonhallintotiedot, verkkotiedot ja mediatiedot. Istunnonhallintatiedot määrittelevät viestinnän alustuksen, lopetuksen ja muokkauksen. Verkkotiedot kertovat käyttäjien sijainnin IP-osoitteen ja portin kautta, kun taas mediatietoja vaihtamalla osapuolet määrittävät yhteiset koodekit ja mediatyypit.
Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

WebRTC:ssä näiden mediatietojen vaihtaminen tapahtuu asiakkaiden vaihtaessa paikallisia yhteydenkuvaustietojaan SDP-muodossa. (WebRTC Signaling n.d)

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

4 VAATIMUKSET YLEISKÄYTTÖISELLE KOMPONENTILLE

Komponentin tärkeimmät vaatimukset ovat yleiskäytettävyys, vaivaton laajennettavuus sekä dokumentoinnin pysyminen mahdollisimman selkeänä. Yleiskäytettävyys on yksi kriittisimmistä vaatimuksista ja jotta yleiskäytettävyyden määritelmä toteutuisi tulee komponentti pystyä upottamaan olemassa olevaan järjestelmään ilman suuria muutoksia tai kokonaisuuden vaarantamista. Komponentin laajentaminen tulee kyetä toteuttamaan ilman, että komponentin yleinen logiikka tai rakenne hajoaa. Komponentin rakenteen mukainen laajentaminen ei saa vaikuttaa dokumentointilogiikkaan, ellei niin erikseen haluta. Dokumentointi tulee pystyä toteuttamaan ja esittämään selkeällä tavalla, joka on johdonmukainen ja mahdollistaa myös laajennuksien esittämisen selkeästi olemassa olevaa rakennetta hyödyntäen.

Valmiissa komponentissa yleiskäytettävyys tullaan saavuttavaan erottamalla käyttöliittymä ja komponenttilogiikka toisistaan, jolloin komponentin upottaminen olemassa olevaan järjestelmään onnistuu minimaalisin muutoksin ja käyttöliittymän luonti on mahdollista kehittäjän toiveiden mukaisesti. Komponentin ominaisuuksien laajentaminen on suunniteltu toteutettavaksi käyttäen luokkapohjaisen rakenteen mahdollistamaa periyttämislogiikkaa, jolloin dokumentointi on mahdollista toteuttaa selkeästi ja olemassa olevia dokumentteja laajentamalla.

Komponentin on tarkoitus olla avuksi kaikille, jotka ovat kiinnostuneita WebRTC:stä. Komponenttia tulee voida käyttää valmiina pohjana tai karsittuna kokonaisuutena osana suurempaa toteutusta. Täten komponentti julkaistaan mahdollisimman vapaalla lisenssillä joka mahdollistaa sen hyödyntämisen kaikissa tilanteissa ilman rajoituksia.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

5 AVOIN LÄHDEKOODI JA KOMPONENTIN LISENSSI

Avoimella lähdekoodilla tarkoitetaan jotain, jota voidaan muokata ja jakaa vapaasti koska se on julkisesti saatavissa. Termin syntyaikoina sillä tarkoitettiin tiettyä lähestymistapaa ohjelmistokehitykseen, mutta nykyään sen tarkoitus on laajentunut tarkoittamaan projekteja, joissa keskiössä on yhteistyökeskeinen ja avoin kehitys. (Open Source 2019)

Avoimen lähdekoodin lisenssejä on useita erilaisia, mutta ne voidaan jakaa viiteen yleiseen päätyyppiin:

- Julkinen
    - Kaikkien vapain lisenssityyppi. Kuka tahansa voi muokata ja käyttää ohjelmistoa ilman rajoituksia.
- Salliva
    - Tunnetaan myös nimellä Apache-tyyli tai BSD tyyli. Sisältää vähimmäisvaatimukset siitä, kuinka ohjelmistoa voidaan muokata ja uudelleen jakaa.
- Vähempi yleinen lisenssi
    - Sallii linkittämisen avoimen lähdekoodin kirjastoihin. Linkittäessä tällä lisenssillä olevan kirjaston omalla koodilla voidaan sovellus julkaista millä tahansa lisenssillä, mutta jos kirjastoa muutetaan tai sen osia kopioidaan koodiin, tulee sovellus julkaista tällä lisenssillä.
- Copyleft
    - Kutsutaan myös vastavuoroiseksi tai rajoittavaksi lisenssiksi. Antaa muokata lisensoitua koodia ja julkaista sen pohjalta tehtyjä uusia teoksia, kunhan ne julkaistaan saman lisenssin nojalla.
- Patentoitu
    - Kaikista rajoittavin lisenssi. Lisenssin ajatuksena on, että kaikki oikeudet pidätetään. Käytetään ohjelmistoon tai sovellukseen, jota ei saa muokata tai jakaa uudelleen.
      (Types of Software Licenses 7.4.2020)

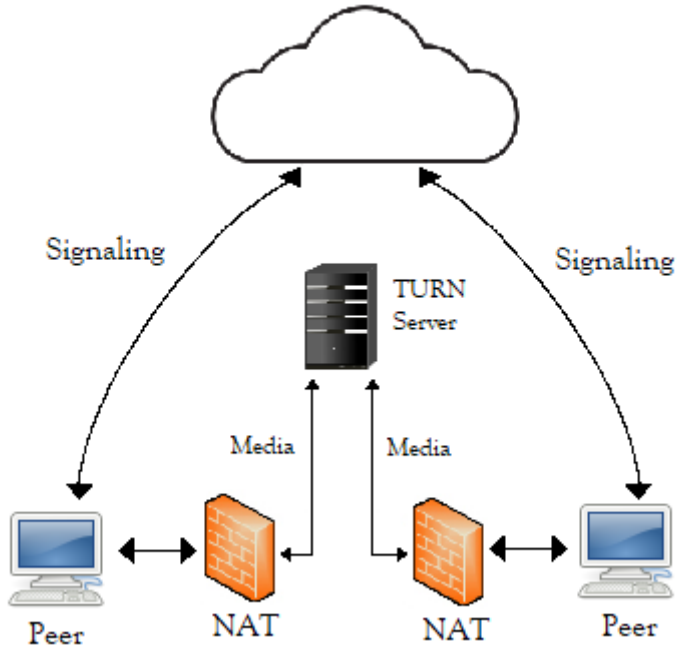
Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

Koska komponentti on suunniteltu uudelleen käytettäväksi ja avuksi muille ohjelmoijille, valittiin tähän komponenttiin mahdollisimman salliva lisenssi: Unlicence. Unlicense:ssa ei ole minkäänlaisia ehtoja ja sen alla julkaistuja teoksia voidaan muokata ja jakaa vapaasti (The Unlicense n.d). Tämä mahdollistaa komponentin vapaan käyttämisen mihin tahansa tarkoitukseen ja vastaa komponentin suunniteltua tarkoitusta.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

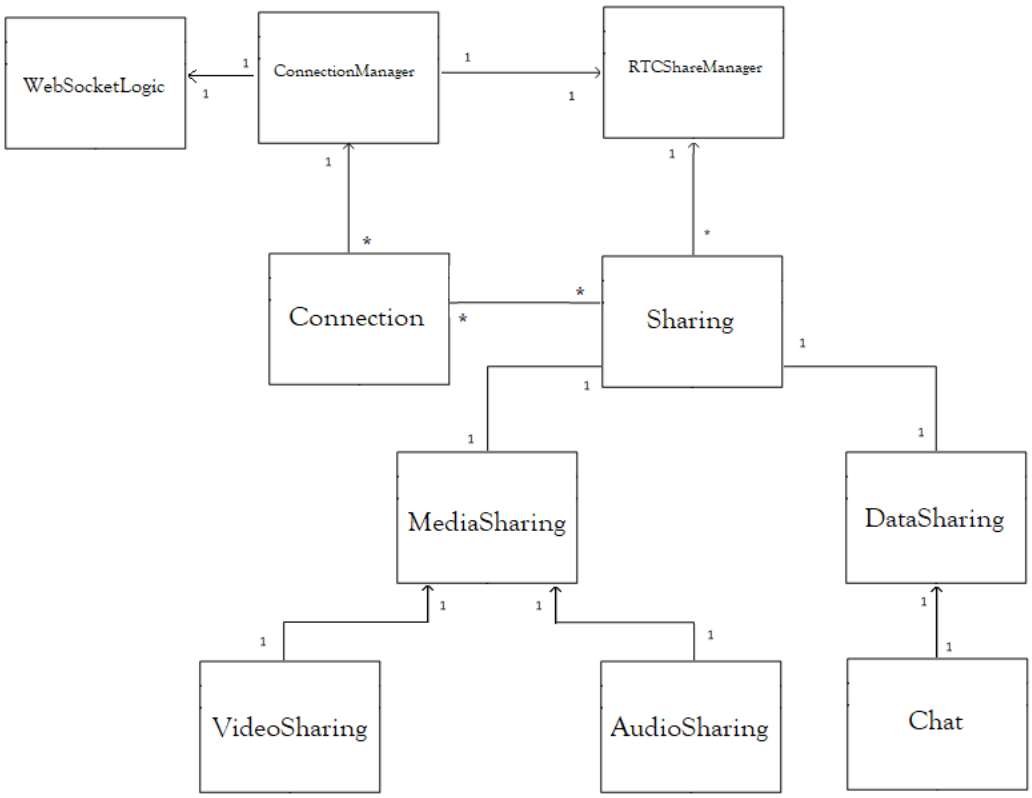
6 KOMPONENTIN RAKENNE

Komponentti on suunniteltu toimivaksi yhden TURN palvelimen kautta, joka välittää mediaa asiakkaiden välillä ja kuviossa 2 kuvataan tätä komponentin arkkitehtuuria. Komponenttilogiikka on rakennettu hyödyntäen TypeScriptin luokkapohjaista rakennetta, jota kuvataan kuviossa 3. Jokaisella luokalla on vastuualue, joka määrittää luokan tehtävän komponentin toiminnassa. Luokkapohjainen rakenne mahdollistaa komponentin jatkokehityksen joko laajentamalla jo olemassa olevaa logiikkaa tai periyttämällä luokista uusia luokkia, jolloin rakenne pysyy selkeänä ja helposti dokumentoitavana. Komponentti on rakennettu kymmenestä luokasta: RTCShareManager, ConnectionManager, Connection, Sharing, DataSharing, MediaSharing, AudioSharing, VideoSharing, Chat ja WebSocketLogic. Seuraavissa osuuksissa käydään läpi näiden luokkien vastuita komponentin toiminnassa.



Kuvio 2. Komponentin arkkitehtuuri

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä



Kuvio 3. Komponentin luokkakaavio.

6.1 RTCShareManager

RTCShareManager-luokka vastaa komponentin hallinnallisista tehtävistä, kuten käyttöliittymäelementtien ja komponentin toiminnallisuuksien ohjaamisesta. Näitä toiminnallisuuksia ovat äänen, videon ja tekstin lähettäminen sekä vastaanottaminen. Vastuualueeseen kuuluu myös ConnectionManagerin ohjaaminen äänen, videon ja chatin vastaanottamiseen ja lähettämiseen liittyvissä tehtävissä, kuten vastuu mediaa lähettävän osapuolen mikrofonin tai webkameran käyttämisen aloittamisesta ja lopettamisesta. RTCShareManager ohjaa käyttöliittymäelementtejä vastaanottamalla parametrinä viittaukset käyttöliittymäelementteihin ja on vastuussa niiden toiminnallisuuden ohjaamisesta. Käyttöliit-

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

tymä on erotettu komponenttilogiikasta Index.html tiedostoon, jossa RTCShare-Manager alustetaan ja sille syötetään argumenttina viittaukset kaikkiin HTML-elementteihin. Kuvio 4 kuvaa RTCShareManagerin alustusta.

```
var rtc = new RTCShareManager(null, {
behaviour: {
  live_mode: {
    start_connection: true,
    chat: false,
    audio_share: true
  }
},
buttons: {
  live_mode: "liveMode",
  audio_share_button: "audioShare",
  video_share_button: "videoShareButton",
  chat_exit_button: "chatCloseButton",
  chat_open_button: "chatOpenButton"
},
keys: {

},
ice_servers: [{
    urls: "https://example_url.fi:XXXXX",
    username: "example_username",
    credential: "example_password"
}],
event_handlers: {
  on_receive_audio: "",
  on_audio_share: "",
  on_receive_message: "",
  on_video_share: "",
  on_receive_video: "",
```

Kuvio 4. RTCShareManagerin alustus.

6.2 ConnectionManager

ConnectionManager-luokka hallinnoi yhteyden luontiin liittyviä asioita kuten liikennettä asiakkaan ja WebSocket-palvelimen välillä sekä yhteysolioita. Vastuualueen WebSocket osuuteen kuuluu myös WebRTC yhteydelle välttämättömän signaloinnin hoitaminen. ConnectionManager vastaa yhteysolioiden osalta myös tapahtumakäsittelijöiden ja datakanavien luomisesta. WebsocketLogic-

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

luokka alustetaan ConnectionManagerin kautta ja signalointi hoituu ConnectionManagerin ohjauksen kautta hyödyntämällä WebSocketLogic-luokan sendSignal-funktiota.

6.3 Connection

Connection-luokka vastaa asiakasolion luomisesta, joka sisältää kaksi yhteysoliota sekä tiedot siitä mitä asiakas on jakamassa ja vastaanottamassa milläkin hetkellä. Yhteysolioille syötetään argumenttina configuraatio-olio, jossa määritetään TURN palvelimen hyväksytyt kandidaattiparit. Tässä komponentissa käytetään TURN palvelinta, sillä komponentti on suunniteltu käytettäväksi NAT:n yli. Komponentti ei lähtökohtaisesti hyväksy muita kuin TURN palvelimen kautta tulevia yhteyksiä, mutta tämä on muutettavissa yhteysolioiden konfiguraatiosta, mikäli se koetaan toiminnan kannalta tarpeelliseksi. Tämä tapahtuu vaihtamalla konfikuraatio-olion iceTransportPolicyn arvoksi ’all’.

Asiakkaan jakamisen tilojen seuraaminen tapahtuu pitämällä kirjaa jakamisen ja vastaanottamisen alkamisesta ja loppumisesta. Tätä varten jokaisella asiakasoliolla on jokaiselle jako-ominaisuudelle vastaanotto- ja lähetystotuusarvomuuttuja, joka ilmentää senhetkistä jakamisen tai vastaanottamisen tilaa.

6.4 Sharing

Sharing on abstrakti luokka, joka helpottaa komponentin ylläpidettävyyttä ja vähentää toistuvaa koodia alaluokissa. Komponentti ei tässä aseta rajoituksia vaan Sharing-luokka on olemassa, jotta ominaisuuksia voitaisiin laajentaa mahdollisimman helposti ja selkeästi.

6.5 DataSharing

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

DataSharing on abstrakti luokka ja helpottaa komponentin datan jakamiseen liittyvien ominaisuuksien ylläpidettävyyttä ja vähentää toistuvaa koodia alaluokissa. Komponentti asettaa yhden esimerkkifunktion ’getMedia’, joka toimii kaikissa DataSharing-luokan alaluokissa jakamisen aloittamisfunktiona ja määrittää, että kaikki alaluokat vastaanottavat parametrinä datakanavat sekä tarvittaessa toisen vapaavalintaisen parametrin, kuten ID:n.

6.6 MediaSharing

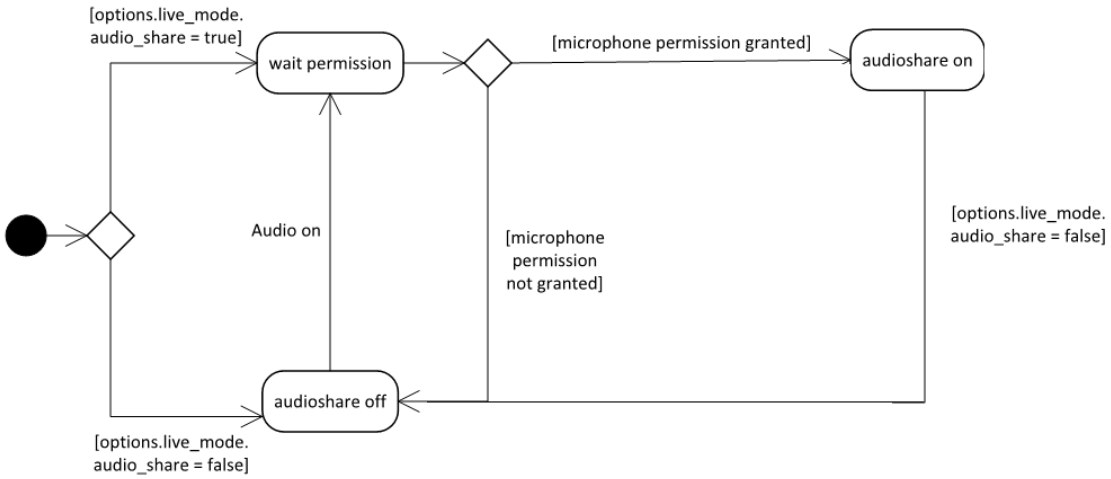
MediaSharing on abstrakti luokka ja helpottaa komponentin median jakamiseen liittyvien ominaisuuksien ylläpidettävyyttä ja vähentää toistuvaa koodia alaluokissa. Komponentti asettaa edellä mainitunlaisen ’getMedia’ esimerkkifunktion, joka määrittää, että jokainen MediaSharing-alaluokka vastaanottaa yhteysoliot ja tarvittaessa datakanavat argumenttina. Tämän lisäksi komponentti määrittelee median, mediaelementtien, ja rtpSenderien poisto- tai lopetusfunktiot sekä median vastaanottofunktion. Näiden toteutukseen MediaSharing luokka asettaa joitakin rajoituksia: elementtien ja rtpSenderien poistofunktioille tulee syöttää id argumenttina ja median vastaanottofunktiolle track-tapahtuma, id sekä yhteysoliot argumentteina.

6.7 AudioSharing

AudioSharing-luokka vastaa komponentin äänen jakamisen logiikasta, kuten asiakkaan mikrofonin käyttöönotosta, ääniraitojen liittämisestä yhteysolioihin, äänen vastaanottamisesta toiselta asiakkaalta ja äänen jakamisen lopettamisesta. Koska AudioSharing luokka on MediaSharingin alaluokka, sillä on määrityksien mukainen getMedia-funktio. Tässä funktiossa käyttäjältä kysytään mikrofonin käyttöoikeus ja mikäli oikeus saadaan, haetaan mikrofonista ääniraidat, jotka liitetään yhteysolioihin ja päivitetään asiakkaan jakamistila ilmentämään

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

aktiivista äänenjakotilaa. Mikrofonia kysyttäessä voidaan määrittää myös äänenjakoa koskevat asetukset, kuten esimerkiksi kaiunpoisto. Kuvio 5 kuvaa AudioSharingin tiloja.

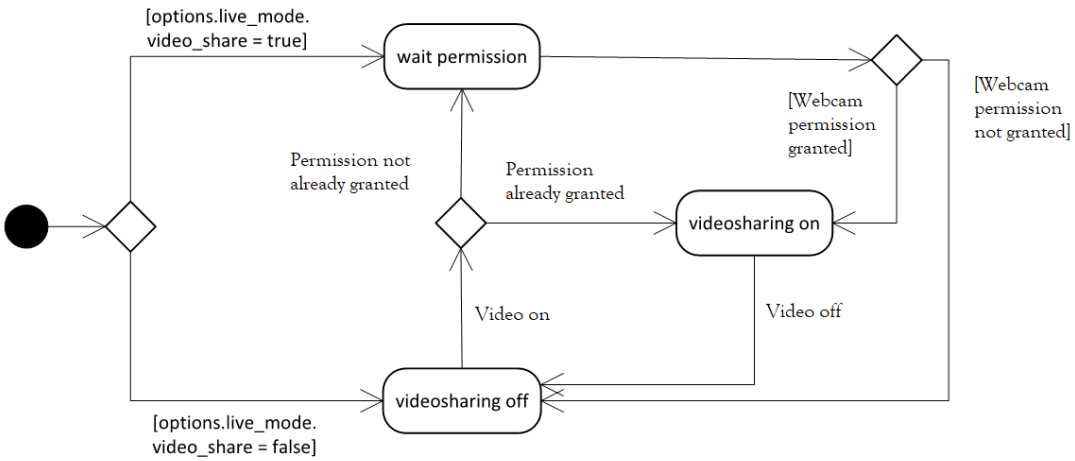


Kuvio 5. AudioSharing luokan tilakaavio.

6.8 VideoSharing

VideoSharing-luokka vastaa komponentin videon jakamisen logiikasta, kuten asiakkaan verkkokameran käyttöönotosta, videoraitojen liittämisestä yhteysolioihin, videon vastaanottamisesta toiselta asiakkaalta ja videon jakamisen lopettamisesta. Koska VideoSharing luokka on MediaSharing luokan alaluokka, on sillä MediaSharing luokan määrityksien mukainen getMedia-funktio. Tässä funktiossa käyttäjältä kysytään oikeus käyttää verkkokameraa ja mikäli oikeus saadaan haetaan verkkokamerasta videoraidat, jotka liitetään yhteysolioihin ja päivitetään asiakkaan jakamistila ilmentämään aktiivista videonjako tilaa. Videoraitoja haettaessa voidaan hakea myös pelkät videoraidat, mikäli videoon ei haluta ääntä mukaan. Komponentti hakee oletuksena sekä ääni-, että videoraidat. Verkkokameraa kysyttäessä voidaan määrittää myös videonjakoa koskevat asetukset kuten esimerkiksi videon resoluutio. Kuvio 6 kuvaa VideoSharingin tiloja.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

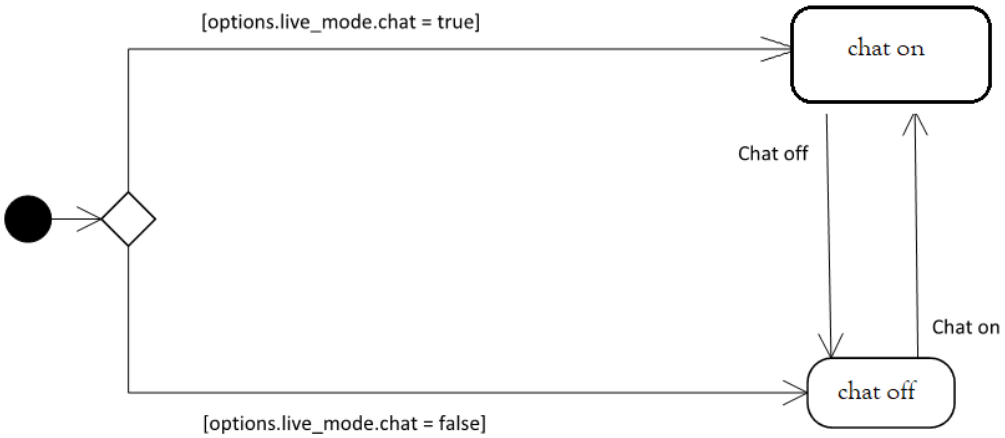


Kuvio 6. VideoSharing luokan tilakaavio.

6.9 Chat

Chat-luokka vastaa viestien lähettämisestä, vastaanottamisesta, chatin sulkemisesta ja viestien tarkistamisesta. Koska Chat-luokka on DataSharing alaluokka, sillä on määrityksien mukainen getMedia-funktio, jolla viestien lähettämiseen ja vastaanottamiseen tarvittavat toimet toteutetaan. Tässä funktiossa hyödynnetään RTCShareManagerin tarjoamia asetuksia käyttöliittymäelementtien luomiseen ja tarvittavien nappien alustamiseen. Chat luokka toimii yhteistyössä ConnectionManager- ja RTCShareManager luokan kanssa päivittäessään datakanavia. Kun ConnectionManager luo uuden datakanavan välittää RTCShareManager sen Chat-luokalle, joka päivittää datakanavat. Kuvio 7 kuvaa Chatin tiloja.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä



Kuvio 7. Chat luokan tilakaavio

6.10 WebSocketLogic

WebSocketLogic-luokka vastaa signalointilogiikasta, kuten käyttäjän ID:n asettamisesta, WebSocket yhteyden luomisesta, huoneen rekisteröinnistä sekä signaloinnista asiakkaiden välillä. Yhteyttä muodostaessa WebSocket palvelimen osoite syötetään StartWebSocketConnection-funktion socket-muuttujalle, jossa määritetään myös muut yhteysasetukset, kuten yhteyden automaattinen uudelleenmuodostus ja maksimaalinen viive.

Koska WebRTC ei itsessään tarjoa valmista tapaa hoitaa signalointia, voidaan signaloinnissa hyödyntää mitä tahansa signalointimenetelmää. Tähän komponenttiin on kuitenkin rakennettu WebSocket signalointilogiikka, sillä WebSocketin huoneet tarjoavat jatkokehitykseen hyvän pohjan. Kuvio 8 kuvaa komponentin WebSocketLogic luokkaa, joka on vastuussa komponentin signalointilogiikasta.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

```
class WebSocketLogic implements IWebSocketClient {
  private socket: any;
  private myID: string;
  private room: IRoomInfo;
  private userID: string;

  constructor(room: IRoomInfo, userID: string) {
    this.room = room;
    this.userID = userID;
  }

  public getUserId(): string {
    return this.myID;
  }

  public startWebSocketConnection() {
    this.socket = io("https://example_url.fi:XXXXX", {
      reconnectionAttempts: 100,
      reconnectionDelay: 5000,
      reconnectionDelayMax: 20000,
      autoConnect: false
    });

    this.socket.connect();
    this.socket.on("connect", () => {
      console.log("Connected to the server");
      this.myID = this.socket.id;
      console.log("My ID is: " + this.socket.id);
      console.log("--------------------------");
      this.socket.emit("register", {
        room: this.room,
        userId: this.userID
      });
    });
  }

  public sendSignal(signal: ISocketSignal): void {
    this.socket.emit("signal", signal);
  }

  public registerSignalHandler(handler: IServerSignalHandlerCallbackDelegate): void {
    this.socket.on("signal", handler);
  }
}
```

Kuvio 8. WebSocketLogic luokka.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

7 ASIAKASOLIO

Connection luokan kaksi yhteysoliota toimivat siten, että kun yhteys on luotu onnistuneesti niin toinen yhteysolioista vastaa kaikesta lähetyksestä ja toinen vastaanottamisesta. Vastuiden kannalta näitä olioita voikin käsitellä nimillä lähetysolio ja vastaanotto-olio. Kuvio 9 kuvaa asiakasoliota.

```
this.client = {
  id: id,
  connectionObjects: {
    object1: new RTCPeerConnection(this.configuration),
    object2: new RTCPeerConnection(this.configuration)
  },
  shareStatusAudio: this.isSharingAudio,
  shareStatusVideo: this.isSharingVideo,
  receiveStatusVideo: this.isReceivingVideo,
  receiveStatusAudio: this.isReceivingAudio,

};
```

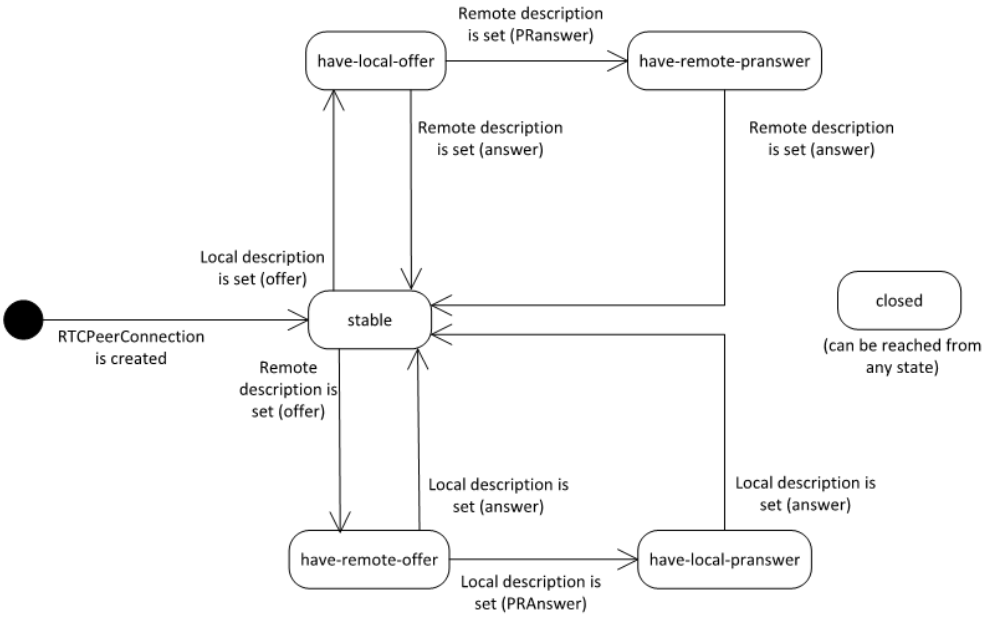
Kuvio 9. Asiakasolio.

WebRTC:ssä kahden asiakkaan välille yhteyttä muodostaessa tarvitaan vähintään yksi yhteysolio per asiakas, mutta tässä komponentissa käytän kahta yhteysoliota per asiakas. Syy tähän on yhteysolioiden signalointitilassa ja siinä, miten tila vaikuttaa mahdollisiin toimiin yhteysneuvottelujen aikana. Seuraavassa osiossa käydään läpi näitä tiloja.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

8 YHTEYSOLION SIGNALOINTITILAT

Yhteysolion singnalointiloihin tulee siirtyä tietyssä järjestyksessä, jotta yhteys voidaan muodostaa onnistuneesti. Alla on kuvattuna yhteysolioiden signalointitilat (Kuvio 10) ja niiden selitykset:



Kuvio 10. Yhteysolion signalointitilat.

STABLE
Yhteysolio (lähetysolio tai vastaanotto-olio) on uusi, eli olion oma yhteydenkuvaus ja toisen asiakkaan yhteydenkuvaus ovat tyhjiä tai neuvottelu on käyty loppuun asti ja yhteys on muodostettu onnistuneesti.

HAVE-LOCAL-OFFER
Lähetysolio on asettanut oman yhteytensä kuvauksen ja luonut siitä tarjouksen, joka toimitetaan toiselle asiakkaalle signalointikanavaa pitkin.

HAVE-REMOTE-OFFER
Vastaanotto-olio on asettanut toisen asiakkaan luoman tarjouksen toisen asiakkaan yhteydenkuvaukseksi, jonka se on saanut signalointikanavaa pitkin toiselta asiakkaalta.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

HAVE-LOCAL-PRANSWER
Vastaanotto-olio on saanut tarjouksen toiselta asiakkaalta ja asettanut toisen asiakkaan yhteydenkuvauksen ja lähettää vastauksena oman yhteydenkuvauksen signalointikanavaa pitkin.

HAVE-REMOTE-PRANSWER
Lähetysolio on asettanut toisen asiakkaan vastauksena luoman yhteydenkuvauksen toisen asiakkaan yhteydenkuvaukseksi, jonka se on saanut signalointikanavaa pitkin vastaanotto-oliolta.

(Signaling States 24.1.2020)

Jotta yhteyden luonti onnistuisi tulee seuraavat tilat toteutua kunkin osapuolen kohdalla seuraavasti:

Lähetysolio:
STABLE – HAVE-LOCAL-OFFER – HAVE-REMOTE-PRANSWER – STABLE

Vastaanotto-olio:
STABLE – HAVE-REMOTE-OFFER – HAVE-LOCAL-PRANSWER – STABLE

Mikäli tätä järjestystä ei noudateta, ovat vaihtoehdot joko tappaa yhteysolio ja luoda uusi tai käyttää palautus mekanismia (rollback). palautusmekanismia tukee tällä hetkellä vain Firefox ja koska komponentti on tehty tukemaan sekä Firefoxia että Chromea, ei rollbackin käyttö ollut mahdollista.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

9 YHTEYDEN MUODOSTAMINEN

WebRTC:n yhteydenmuodostus tapahtuu yksinkertaistettuna kuuden askeleen kautta:

1. SDP tarjouksen luominen ja lähettäminen.
2. SDP tarjouksen vastaanottaminen ja asettaminen etäpariksi.
3. SDP vastauksen luominen ja lähettäminen.
4. SDP vastauksen vastaanottaminen ja asettaminen etäpariksi.
5. ICE-kandidaattien lähettäminen etäparille.
6. ICE-kandidaatin vastaanottaminen etäparilta ja sen asettaminen

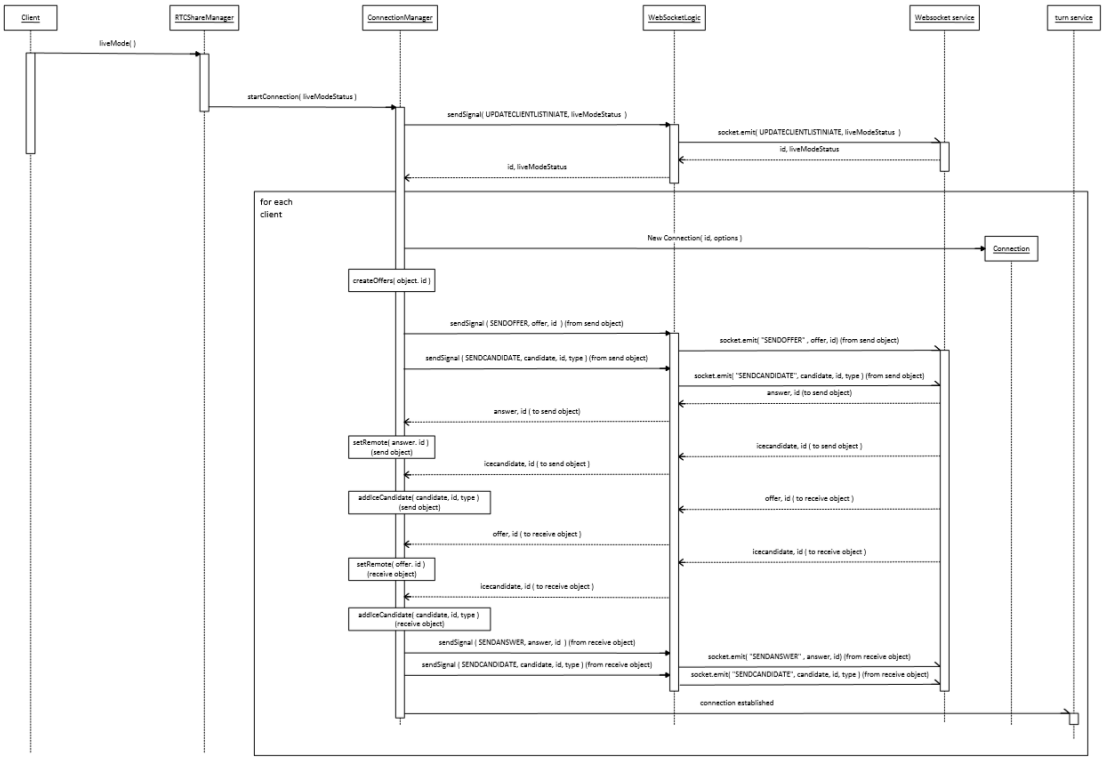
avoimen lähdekoodin yhteisö kuvaa prosessia seuraavanlaisesti 13 askeleen esimerkissä:

1. Kutsuva osapuoli ottaa yhteyden paikalliseen mediaan getUserMedia()-funktiolla
2. Kutsuja luo yhteysolion ja lisää haetut raidat kutsumalla addTrack() metodia.
3. Kutsuja kutsuu createOffer() metodia luodakseen SDP tarjouksen
4. Kutsuja kutsuu setLocalDescription() asettaakseen SDP tarjouksen yhteyden paikalliseksi kuvaukseksi
5. Kutsuja kysyy STUN palvelinta luomaan ICE-kandidaatit
6. Kutsuja käyttää signalointi palvelinta välittääkseen tarjouksen oikealle vastaanottajalle.
7. Vastaanottaja vastaanottaa SDP tarjouksen ja kutsuu setRemoteDescription() metodia asettaakseen tarjouksen yhteyden etäkuvaukseksi.
8. Vastaanottaja tekee tarvittavat medioiden haltuunotot ja asettaa mediaraidat kutsumalla addTrack() metodia.
9. Vastaanottoja luo SDP vastauksen kutsumalla createAnswer() metodia.
10. Vastaanottaja kutsuu setLocalDescription() metodia ja asettaa luodun vastauksen yhteyden paikalliseksi kuvaukseksi. Vastaanottajalla on nyt yhteyden paikallinen- ja etäkuvaus.
11. Vastaanottaja käyttää signalointi palvelinta välittääkseen SDP vastauksen Kutsujalle.
12. Kutsuja vastaanottaa SDP vastauksen.
13. Kutsuja kutsuu setRemoteDescription() metodia ja asettaa SDP vastauksen yhteyden etäkuvaukseksi. Kutsujalla on nyt yhteyden paikallinen ja -etäkuvaus. Media alkaa virtaamaan osapuolten välillä konfikuraation mukaisesti.

(WebRTC Connectivity 12.12.2019)

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

Komponentin vastaava logiikka, jossa yhteys luodaan TURN palvelimeen, on kuvattu alla olevassa kuviossa 11 joka käsittelee komponentin yhteyden muodostusta ja sen eri vaiheita.



Kuvio 11. Yhteyden muodostaminen.

Prosessi alkaa, kun vähintään kaksi asiakasta on live mode-tilassa. Tämä tila kertoo komponentille, että yhteys halutaan muodostaa ja mahdollistaa asiakkaiden irtautumisen tarpeen vaatiessa. Tälle mahdollisia käyttötarkoituksia voi olla esimerkiksi yhteyden manuaalinen uudelleen käynnistäminen, huoneen vaihtaminen, irrottautuminen muista asiakkaista ja määrityksien vaihtaminen ennen uudelleen yhdistämistä muihin asiakkaisiin.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

Seuraava vaihe, startConnection(), on automaattinen mutta on mahdollista muuttaa tapahtumaan asiakkaan käskystä. Tässä vaiheessa asiakkaat vaihtavat identifikaatio tunnisteita, eli ID:eitä, sekä varmistuksen siitä, että kaikki asiakkaat ovat live mode-tilassa WebSocketin avulla.

Seuraavana on yhteysolioiden luonti. Asiakkaat käyttävät hankittua identifikaatio tunnistetta ja luovat tähän ID:hen sidotun asiakasolion. Asiakasolio sisältää kaksi yhteysoliota, joista toinen on lähettämisetä varten ja toinen vastaanottoa varten. Yhteysolion luomisen jälkeen kukin asiakas kutsuu createOffers() funktiota (Kuvio 12) jossa se asettaa paikallisen yhteyden kuvauksen ja lähettää tarjouksen toiselle asiakkaalle WebSocketin avulla. Paikallisen kuvauksen asettamisen jälkeen asiakas voi lähettää myös ICE-kandidaatit vastaanottavalle osapuolelle (Kuvio 13). Vastaanottava asiakas vastaanottaa tarjouksen (Kuvio 14), asettaa tarjouksen yhteyden etäkuvaukseksi, luo vastauksen ja lähettää sen oikealle asiakkaalle (Kuvio 15). Seuravaksi Kutsuva asiakas vastaanottaa vastauksen ja asettaa sen paikallisen yhteyden etäkuvaukseksi (Kuvio 16)

```
//sets local description, creates offer and sends it to correct client
private async createOffers(object: RTCPeerConnection, client: string) {
  try {
    let offer = await object.createOffer();
    await object.setLocalDescription(offer);
    if (object.signalingState == "have-local-offer") {
      this.socket.sendSignal({
        room: this.room,
        data: {
          Sender: this.myID,
          Offer: offer,
          Receiver: client,
          event: this.SENDOFFER
        }
      });
    }
  } catch (error) {
    console.log("Local(offer)[" + client + "]: " + error);
  }
}
```

Kuvio 12. Paikallisen yhteyskuvauksen luonti.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

```
object1.onicecandidate = (event: RTCPeerConnectionIceEvent) => {
  if (event.candidate) {
    let candidate = event.candidate;
    let type = "1";
    this.socket.sendSignal({
      room: this.room,
      data: {
        Sender: this.myID,
        Candidate: candidate,
        Type: type,
        Receiver: id,
        event: this.SENDCANDIDATE
      }
    });
  }
}
```

Kuvio 13. ICE-kandidaattien lähettäminen.

```
if (signalData.data.event == this.SENDOFFER && signalData.data.Receiver == this.myID) {
  for (let i = 0; i < this.connections.length; i++) {
    if (this.connections[i].client.id == signalData.data.Sender) {
      let object = this.connections[i].client.connectionObjects.object2;
      this.setRemote(object, "offer", signalData.data.Offer, this.connections[i].client.id);
    }
  }
}
```

Kuvio 14. Tarjouksen vastaanottaminen.

```
//sets remote description based on type (offer or answer)
//if type is 'answer' completes the handshake and checks if RTCPeerConnection object is connected succesfully
private async setRemote(object: RTCPeerConnection, type: string, offer: RTCSessionDescription, id ? : string) {
  if (type == "offer") {
    try {
      await object.setRemoteDescription(offer);
      let answer = await object.createAnswer();
      await object.setLocalDescription(answer);

      if (object.signalingState == "stable") {
        this.socket.sendSignal({
          room: this.room,
          data: {
            Sender: this.myID,
            Answer: answer,
            Receiver: id,
            event: this.SENDANSWER
          }
        });
      }
    } catch (error) {
      console.log("Remote(offer)[" + id + "]: " + error);
    }
  } else if (type == "answer") {
    try {
      await object.setRemoteDescription(offer);
    } catch (error) {
      console.log("Remote(answer)[" + id + "]: " + error);

    }
  }
}
```

Kuvio 15. Vastauksen luominen ja lähettäminen.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

```
if (signalData.data.event == this.SENDANSWER && signalData.data.Receiver == this.myID) {
  for (let i = 0; i < this.connections.length; i++) {
    if (this.connections[i].client.id == signalData.data.Sender) {
      let object = this.connections[i].client.connectionObjects.object1;
      this.setRemote(object, "answer", signalData.data.Answer, this.connections[i].client.id);
    }
  }
}
```

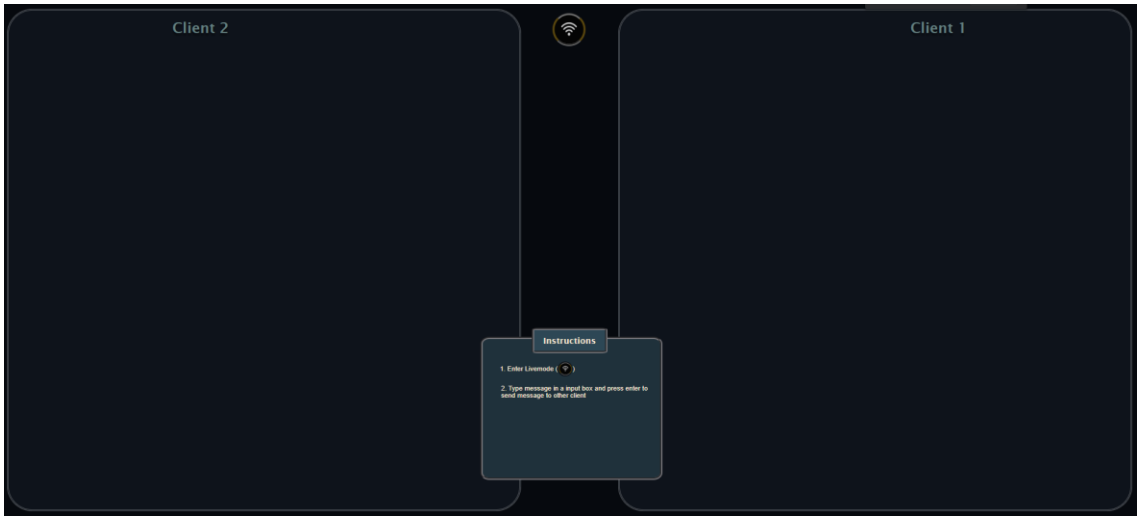
Kuvio 16. Vastauksen vastaanottaminen.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

10 ESIMERKKISOVELLUS – CHAT

Komponenttia voidaan käyttää lukuisiin tarkoituksiin, joista yksi on Chat-sovellus. Tässä esimerkkisovelluksessa on käytetty pohjana tätä komponenttia. Sovellus on karsittu kokonaisuus tässä työssä esitellystä komponentista ja sen tarkoitus on esitellä komponentin toiminnallisuutta ja yleiskäytettävyyttä. Sovellus ei käytä TURN palvelinta, joten se on rajattu paikallisessa verkossa toimivaksi esimerkiksi komponentin eri osa-alueiden käytöstä.
Esimerkkisovellusta voi tarkastella täällä:
https://github.com/JoniMustaniemi/WebRTC-DataChannel-Demo

Sovelluksen lähtötilassa (kuvio 17) käyttöliittymässä on kaksi asiakasta, asiakas 1 ja asiakas 2 sekä keskellä live mode -nappi. alareunasta löytyy myös sovelluksen käyttöohjeet englanniksi.



Kuvio 17. Sovelluksen lähtötila

live mode-nappia painettaessa komponentille kerrotaan, että asiakkaat haluavat muodostaa yhteyden. Yhteysneuvottelut aloitetaan luomalla yhteysoliot ja tehdään tarvittavat toimet yhteyden luomiseksi ja jos yhteys onnistuu, ilmestyy käyttöliittymään viesti-ikkunat sekä tekstinsyöttökentät molemmille asiakkaille. Tässä tilassa asiakkaat voivat lähetellä viestejä toisilleen. Kuviossa 18 kuvataan

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

käyttöliittymää live mode-tilan ollessa aktiivinen. Viestintä voidaan lopettaa painamalla uudelleen live mode-nappia, jolloin palataan sovelluksen lähtötilanteeseen.

Kuvio 18. live mode-tila aktiivisena

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

11 POHDINTA

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa yleiskäyttöinen ja helposti laajennettavissa oleva WebRTC komponentti, jolla voidaan kommunikoida verkon yli ja jakaa ääni-, video- tai muuta dataa usean asiakkaan kanssa samanaikaisesti. Työn tavoite oli helpottaa WebRTC:stä kiinnostuneiden ohjelmoijien kehitysprosessia tarjoamalla valmis komponentti, joka voidaan upottaa olemassa olevaan järjestelmään tai käyttää pohjana omalle toteutukselle.

Työn tuloksena syntyi yleiskäyttöinen avoimen lähdekoodin WebRTC komponentti, jolla voidaan kommunikoida TURN palvelimen yli reaaliaikaisesti ilman ulkopuolisia ratkaisuja. Kuka tahansa kehittäjä tai WebRTC:stä kiinnostunut yksilö voi hyödyntää komponenttia kokonaisuutena tai osina ilman rajoituksia. Komponentin dokumentaatio, josta ilmenee komponentin rakenne sekä tärkeimpien osa-alueiden toiminnallisuus toteutettiin hyödyntämällä UML-mallinnuksen menetelmiä.

Komponenttia suunnitellessa lähtökohtana oli tukea Chromea ja Firefoxia. Tämä osoittautui myöhemmin ongelmalliseksi johtuen selainten välisistä eroista WebRTC:n ominaisuuksien tukemisessa. Suurimmaksi haasteeksi nousi rollback toiminnallisuuden puutteellinen tuki Chromessa. Tämä johti siihen, että komponentin toimintaan jouduttiin tekemään muutoksia, jotka vaikuttivat sen peruslogiikkaan merkittävästi. Suunniteltu yleiskäytettävyys ja dokumentointilogiikka sen sijaan toteutuivat toivotulla tavalla ja komponentin saattaminen mahdollisimman monen käyttöön onnistui ennakoitua kivuttomammin.

Toteutettu komponentti on yleiskäyttöisen luonteensa takia monikäyttöinen ja sitä voidaan hyödyntää reaaliaikaisten kommunikointityökalujen rakentamisessa, joka voi olla mitä tahansa digitaalisesta video konferenssi sovelluksesta yksinkertaisempaan tekstipohjaiseen viestittelyyn perustuva ohjelma. Koska komponentin käyttäminen on täysin vapaata, voidaan siitä ottaa myös erillisiä osioita, jolloin WebRTC:n kanssa kamppailevat ohjelmoijat voivat eriyttää esimerkiksi signalointi- tai yhteyslogiikan omaan toteutukseensa. Komponentti

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

suunniteltiin alusta asti jatkokehitystä ajatellen ja täten sen ominaisuuksien laajentaminen on mahdollista hyödyntäen olemassa olevaa rakennetta, jolloin dokumentointilogiikka pysyy samanlaisena myös laajennuksien jälkeen.

Komponenttiin jäi vielä parannettavaa. Mikäli Chrome tulevaisuudessa tukee rollback toiminnallisuutta olisi komponenttiin hyvä lisätä tuo toiminnallisuus ja luopua toisesta yhteysoliosta. Tämä kuitenkin tarkoittaisi suuria rakenteellisia muutoksia komponentin perustoiminnallisuuksiin ja olisi täten työläs toteuttaa. Komponenttiin olisi voinut myös erikseen rakentaa tuen molemmille selaimille, jolloin Firefoxia käytettäessä rollbackiä olisi voitu hyödyntää, mutta tässä kysymykseksi nousisi selainten välinen ristituki.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

12 LÄHTEET

Realtime Computing. n.d. luettu 12.1.2020. https://www.techopedia.com/definition/16991/real-time-computing-rtc.

Realtime Data. 13.6.2018. Luettu 12.1.2020. https://www.techopedia.com/definition/31256/real-time-data.

Big Data Technologies and Cloud Computing. 2015. Luettu 3.3.2020. https://www.sciencedirect.com/topics/computer-science/real-time-computing.

Polling By Ably. n.d. Luettu 23.5.2020. https://www.ably.io/concepts/long-polling

Http Polling. 30.11.2019. Luettu 12.1.2020. https://javascript.info/long-polling#regular-polling.

Long Polling. 30.11.2019. Luettu 12.1.2020. https://javascript.info/long-polling#long-polling.

Long Polling Issues. 1.4.2011. Luettu 12.1.2020. https://tools.ietf.org/id/draft-loreto-http-bidirectional-07.html#polling-issues.

WebSocket vs Long Polling. 4.6.2019. Luettu 12.1.2020. https://www.ably.io/blog/websockets-vs-long-polling.

Introduction to WebSockets. 18.10.2013. Luettu 13.1.2020. https://blog.teamtreehouse.com/an-introduction-to-websockets.

WebSockets By Ably. 8/2018. Luettu 23.5.2020. https://www.ably.io/concepts/websockets

Server-Sent Events. 25.1.2018. Luettu 13.1.2020. https://medium.com/conectric-networks/a-look-at-server-sent-events-54a77f8d6ff7.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

Server-Sent Events By Ably. n.d. Luettu 23.5.2020. https://www.ably.io/concepts/server-sent-events

History of WebRTC. 6.8.2017. Luettu 23.5.2020. https://princiya777.wordpress.com/2017/08/06/webrtc-a-detailed-history/

WebRTC API. 12.3.2020. Luettu 15.1.2020. https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API.

WebRTC By WebPlatform. n.d. Luettu 16.1.2020. https://webplatform.github.io/docs/concepts/Internet_and_Web/webrtc/.

TURN Server. 28.5.2019. Luettu 16.1.2020. https://webrtc.org/getting-started/turn-server.

WebRTC Signaling. n.d. Luettu 16.1.2020. https://www.onsip.com/voip-resources/voip-fundamentals/webrtc-signaling.

Signaling States. 24.1.2020. Luettu 18.1.2020. https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/signalingState.

WebRTC Connectivity. 12.12.2019. Luettu 19.1.2020. https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Connectivity.

Open Source. 2019. Luettu 22.3.2020. https://opensource.com/resources/what-open-source.

Types of Software Licenses. 7.4.2020. Luettu 25.3.2020. https://www.synopsys.com/blogs/software-security/5-types-of-software-licenses-you-need-to-understand/.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä

The Unlicense. n.d. Luettu 30.3.2020. https://choosealicense.com/licenses/unlicense/.

Asiasanat: webrtc, avoin lähdekoodi, yleiskäytettävä