Kari Tirkkonen

Reaaliaikainen hierarkkinen sovelluksen etäkäyttö

Insinööri (AMK)

Tietotekniikka

Kevät 2017

KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tekijä(t): Tirkkonen Kari

Työn nimi: Reaaliaikainen hierarkkinen sovelluksen etäkäyttö

Tutkintonimike: Insinööri (AMK), tietotekniikka

Asiasanat: reaaliaikainen, etäkäyttö, websocket, nodejs

Opinnäytetyössä esitellään reaaliaikaisen etätuen toteutus erääseen työn tilaajan jo toteutettuun verkkosovellukseen. Tavoitteena oli luoda sovellukselle reaaliaikainen etätuki, jonka avulla tilaaja sekä sovelluksen asiakkaat pystyvät kouluttamaan ihmisiä sovelluksen käyttöön hierarkkisesti. Toinen syy etätuelle oli helpottaa sovelluksen markkinointia, koska etätuen avulla sovelluksen ominaisuuksia pystytään esittelemään asiakkaille helpommin etänä.

Normaalista etätuesta poiketen, tukea ei toteutettu videon suoratoiston avulla, vaan sovelluksen toiminnot synkronoidaan reaaliajassa WebSocket-yhteyden kautta. Tällä mahdollistetaan myös palautteen saaminen koulutustilanteesta reaaliajassa takaisin kouluttajalle. Käyttäjän ei myöskään tarvitse asentaa mitään kolmannen osapuolen ohjelmia etätukea varten. Työn tilaajana toimii kajaanilainen yritys, joka tuottaa SaaS-palveluita, pääasiassa verkko- ja mobiilisovelluksia.

Lopputuloksena saatiin integroitua toimiva etätuen palvelin työn tilaajan sovellukseen. Palvelin mahdollistaa myös tulevien ominaisuuksien synkronoimisen kaikille käyttäjille helposti, ilman että palvelinta tarvitsee muokata, jolloin pelkkä asiakaspään muokkaaminen riittää.

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Author(s): Tirkkonen Kari

Title of the Publication: Real-time Hierarchical Remote Use of Web Application

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: real-time, remote use, websocket, nodejs

This thesis is about the implementation and development of real-time remote support system to already deployed web application. Purpose of this thesis was to create a real-time remote support for application, which would allow the company as well as the clients of the web application to educate and instruct their users, in use of the application, hierarchically. Remote support can also be used by the company for marketing the application, because it's easier to demonstrate the application remotely.

Unlike how remote support is usually done, by video streaming, this support system is implemented by synchronizing the events of the application through WebSockets in real-time. This also makes it possible to receive real-time feedback from trainees, right back to the instructor. Because of this approach, user doesn't need to install any third-party software or plugins for remote support. Project was done for a company located in Kajaani. The company specializes in design and development of SaaS web- and mobile applications.

The result of the project is an integrated remote support server for the application. The server also provides easy synchronization of any unreleased features for clients, without the need for making any modifications to the server. This means that modifying only the client-side is required, to synchronize any new features.

SISÄLLYS

1 JOHDANTO 1

2 TYÖHÖN KÄYTETYT TEKNOLOGIAT 2
2.1 WebSocket 2
2.2 WAMP 3
2.2.1 Publish & Subscribe 3
2.2.2 RPC 4
2.3 Crossbar 4
2.4 Autobahn 5
2.5 Node.js 6
2.6 Redis 7

3 TOTEUTUS 8
3.1 Palvelintoteutus 8
3.2 Yhteyden luonti 9
3.3 Käyttöoikeudet 10
3.4 Datan lähettäminen 11
3.5 Huoneet ja käyttäjien hallinta 14

4 TULOKSET 16

5 YHTEENVETO 18

LÄHTEET 19

SYMBOLILUETTELO

- AJAX — Asynchronous JavaScript and XML
- Autobahn — Avoimen lähdekoodin toteutus WAMP protokollasta
- Crossbar — WAMP protokollan mukainen verkkoalusta
- Node.js — Nettipalvelin, kehitysalusta
- PubSub — Viestintämalli
- RPC — Remote Procedure Calls, etäfunktio
- WAMP — Web Application Messaging Protocol
- WSS — Suojattu WebSocket-yhteys

1 JOHDANTO

Työn tilaajana toimii kajaanilainen yritys joka tuottaa SaaS-palveluita, pääasiassa verkko- ja mobiilisovelluksia. Yritys työllistää noin 3 henkilöä. Itse verkkosovellus, johon etätuki lisätään, ei ole vielä julkaistu, joten opinnäytetyössä ei kerrota itse sovelluksesta yksityiskohtia.

Tilaaja on tekemässä uutta versiota verkkosovelluksestaan, johon haluttiin lisätä tuki etäkäytölle, sekä tehdä sovelluksen käytöstä entistä sujuvampaa reaaliaikaisuudella. Etäkäyttö on yleisesti ottaen kehitetty kokonaan videon suoratoiston kautta, jolloin etäkäyttäjä saa haltuunsa koko asiakkaan käyttöliittymän. Muissa etätuen järjestelmissä käyttäjiäkin voi yleensä olla vain yksi samanaikaisesti. Opinnäytetyössä on otettava huomioon erilaiset välityspalvelin- ja palomuuriasetukset, jotka voivat joissain tilanteissa hylätä WebSocket-yhteyden kokonaan, jolloin etäyhteyden luomista varten on hyvä olla varateknologia, kuten esimerkiksi long-polling.

Tavoitteena on luoda olemassa olevaan verkkosovellukseen etäkäyttötuki, jota voidaan hyödyntää sovelluksen markkinoinnissa ja esittelyssä sekä sovelluksen käyttökoulutuksessa tilaajan ja asiakkaiden toimesta. Etätuesta on tarkoitus tehdä myös hierarkkinen, mikä tarkoittaa sitä, että kouluttaja saa koulutettavilta palautetta sovelluksen käytöstä reaaliajassa koulutustilanteessa, jolloin ongelmatilanteisiin voidaan vaikuttaa heti. Tarkoituksena on, että etäkäyttö auttaa tehokkaassa vuorovaikutuksessa yrityksen ja asiakkaan kanssa. Lähdin toteuttamaan etätukea WAMP-protokollan mukaisesti.

2 TYÖHÖN KÄYTETYT TEKNOLOGIAT

Tässä luvussa esitellään teknologioita, joita työssä käytettiin. WebSocket valittiin tiedon lähettämisen protokollaksi, koska se mahdollisti reaaliaikaisen molemman suuntaisen kommunikoinnin käyttäjien välillä mahdollisimman pienellä viiveellä, mikä on tarpeellista reaaliaikaisen etätuen antamisessa.

Crossbar valittiin reitittimeksi työhön, koska se tarjoaa hyvät käyttäjien todennusmenetelmät sekä dynaamiset käyttöoikeudet, joita voidaan jakaa käyttäjien roolien mukaisesti. Crossbarista löytyy myös tuki WSS-protokollalle, jolloin lähetetty tieto saadaan salattua. Tämä auttaa WebSocket-yhteyden luomisessa sekä lisää tietoturvaa.

2.1 WebSocket

WebSocket on standardi-verkkoprotokolla, joka tarjoaa kestävän ja kaksisuuntaisen yhteyden. Tämä tarkoittaa sitä, että palvelin voi lähettää dataa asiakkaalle milloin tahansa, eikä yhteyttä tarvitse luoda uudelleen joka kerta. Sama toimii myös toisinpäin, eli asiakas voi myös lähettää dataa palvelimelle, mikä mahdollistaa nettisivuilla reaaliaikaisen ja sulavan käytettävyyden, koska koko sivua ei tarvitse ladata uudelleen. Koska yhteyttä ei tarvitse luoda uudelleen joka kerta, kun dataa halutaan lähettää, WebSocketit vievät paljon vähemmän resurssejakin. WebSocketit mahdollistavat reaaliaikaisen sovelluksen luomisen paljon paremmin myös tästä syystä, koska yhteyden luomiseen ei kulu turhaa aikaa, toisin kuin esim. long-polling-menetelmässä, jossa asiakkaan pitää jatkuvasti luoda uusi yhteys palvelimeen nähdäkseen, onko palvelimella uutta dataa. [1]

Yhteys muodostetaan päivittämällä perinteinen HTTP GET- pyyntö. Koska WebSocket-yhteys muodostetaan HTTP-yhteyden kautta, voi palvelin käyttää samaa avointa porttia HTTP- ja WebSocket-yhteyksille. Sen jälkeen, kun palvelin

palauttaa vastauksen, WebSocket-yhteys voi alkaa. Tietyt välipalvelimet voivat estää suojaamattomien WebSocket-yhteyksien avaamisen, joten on suositeltavaa käyttää suojattua WebSocket-yhteyttä yhteyden luomiseen (WSS). [1]

2.2 WAMP

Web Application Messaging Protocol on WebSocketin aliprotokolla, joka tarjoaa kaksi sovellusten välistä viestintämallia yhdistettynä yhden protokollan alle. WAMP on monikielinen, sillä eri kirjastoja ja toteutuksia löytyy monille eri kielille, mm. Java, C++, JavaScript ja PHP, ja kaikki toteutukset ovat yhteensopivia keskenään. Tämä tarkoittaa sitä, että voit tehdä toteutuksen ja lähettää viestin, vaikka PHP:llä ja vastaanottaa sen C++:ssa. Asiakasrajapinnan lisäksi tarvitaan reititin, joka pitää kirjaa kirjautuneista käyttäjistä ja rekisteröidyistä funktioista ja ohjaa viestit edelleen käyttäjille. WAMP tarjoaa myös kehittyneempiä ominaisuuksia, kuten kuormituksen tasausta komponenttien välillä, vastaanottajien listauksen, ja erilaisia käyttäjien varmennus-menetelmiä. [2]

2.2.1 Publish & Subscribe

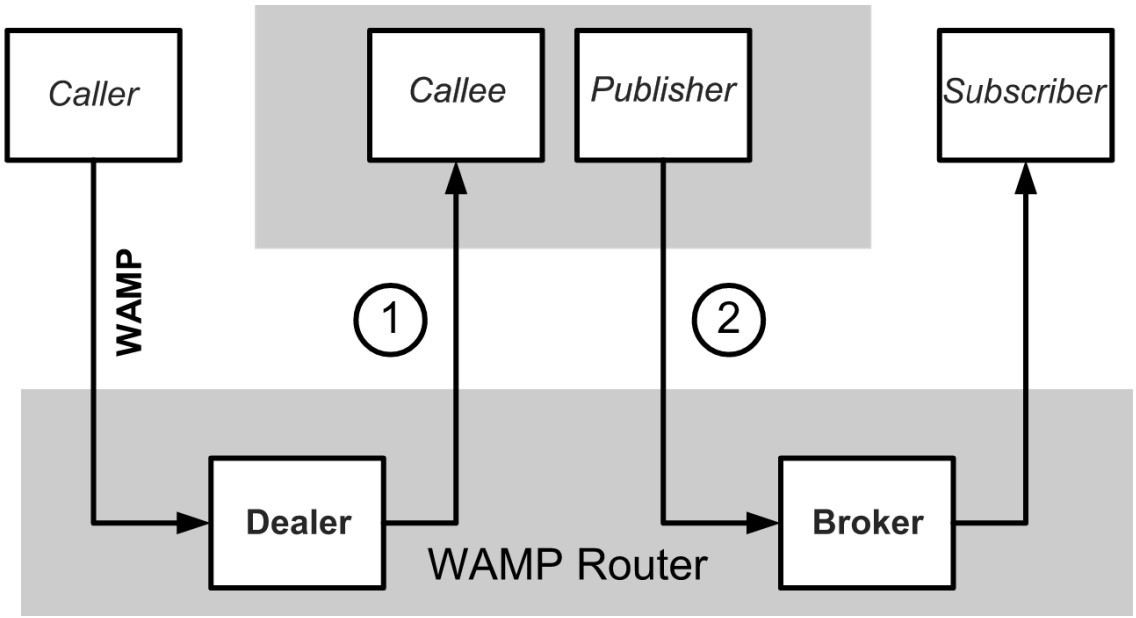
Asiakas kirjautuu kuuntelemaan aihetta, jonka jälkeen toinen asiakas voi julkaista viestin tähän aiheeseen. Julkaisun jälkeen kaikki asiakkaat, jotka ovat kuuntelemassa samaa aihetta, vastaanottavat viestin. Tässä mallissa ei ole mahdollista lähettää palautusarvoa, mutta viesti voidaan lähettää niin monelle asiakkaalle kuin aiheella on kuuntelijoita. Asiakkaat eivät ole tietoisia toisista käyttäjistä, vaan tarvitaan broker, joka pitää listaa aihetta kuuntelevista asiakkaista ja lähettää edelleen viestin kuunteleville asiakkaille. [3]

2.2.2 RPC

Remote Procedure Call sallii funktioiden kutsumisen WebSocketin kautta tai lokaalisti, ja se toimii useiden eri kielien välillä. Sallii palautuksen lähettämisen, mutta voidaan lähettää vain yhdelle asiakkaalle kerrallaan. Caller kutsuu etäfunktiota, jonka jälkeen Callee suorittaa funktion ja lähettää palautusarvon Callerille. Kutsuttava funktio on rekisteröitävä ensin URI:lle, jota kutsutaan. Asiakkaat eivät tiedä tässäkään mallissa toisistaan, vaan tarvitaan dealer, joka pitää kirjaa rekisteröidyistä funktioista ja ohjaa ne edelleen. [3]

2.3 Crossbar

Molemmissa WAMP-protokollan viestintämalleissa tarvitaan broker ja dealer, jotka välittävät viestit eteenpäin. Kun nämä molemmat yhdistetään, saadaan reititin. Reititin mahdollistaa viestien välittämisen molemmissa viestintämalleissa ilman, että lähettäjä ja vastaanottaja tietävät toisistaan. Seuraavaksi kuvassa 1 esitellään miten molemmat viestintämallit toimivat reitittimen kautta. [3]



Kuva 1. Viestintämallikaavio

Crossbar on WAMP-protokollan mukainen avoimen lähdekoodin reititin. Crossbariin voi liittää komponentteja, jotka sitten pystyvät viestimään keskenään. Esimerkiksi nettisivu voi toimia yhtenä komponenttina, ja palvelintoteutus omana komponenttinaan. Crossbar voi toimia tarvittaessa myös nettipalvelimena nettisivujen tarjoamiseen. WAMP:n lisäksi Crossbar tarjoaa lisää ominaisuuksia käyttäjien todennukseen, jotka ovat määriteltävissä joko staattisesti asetuksista, tai käyttäjät voidaan varmentaa dynaamisesti yhteydenluonnin yhteydessä. Staattisessa todennuksessa käyttäjän salaisuus ja rooli määritellään Crossbarin kokoonpano-asetuksissa. Käyttäjän rooli määrittää, mihin aiheisiin tai etäfunktioihin käyttäjä voi julkaista tai kirjautua kuuntelemaan. Dynaamisessa todennuksessa voidaan käyttäjän tietoja verrata ensiksi esimerkiksi tietokannasta, ja antaa sitten sopiva rooli. [4]

Yhtenä tarjottuna esimerkkinä todennuksesta on WAMP-CRA, Challenge-Response Authentication. Se käyttää jaettua salaisuutta asiakkaan ja palvelimen väliseen todennukseen. Salaisuutta ei koskaan lähetetä, jolloin ei periaatteessa haittaa, vaikka yhteys ei ole salattu, mutta salaisuuden jakaminen ei kuulu todennusmenetelmälle. [5]

2.4 Autobahn

Autobahn tarjoaa avoimen lähdekoodin asiakasrajapinnan WAMP-protokollasta ja WebSocketista useille eri ohjelmointikielille ja alustoille, mm. JavaScript, Android ja C++ ovat tuettuja. Kuvassa 2 esitellään miten Autobahn.js luo yhteyden Crossbariin määrittelemällä osoite ja realm mihin yhdistetään. Tämän jälkeen määritetään, mitä tapahtuu, kun yhteys on auki, tässä tapauksessa kirjaudutaan kuuntelemaan aihetta ja julkaistaan sen jälkeen samaan aiheeseen viesti. [6]

```
var connection = new autobahn.Connection({
         url: 'ws://127.0.0.1:9000/',
         realm: 'realm1'
});

connection.onopen = function (session) {

   function onevent(args) {
      console.log(args[0]);
   }

   session.subscribe('com.myapp.hello', onevent);
   session.publish('com.myapp.hello', ['Hello, world!']);
};

connection.open();
```

Kuva 2. Koodiesimerkki yhteyden muodostamisesta.

Koska Autobahnilta löytyy toteutus usealle eri alustalle, sama koodi toimii näin myös palvelinpuolen toteutuksessa. [6]

2.5 Node.js

Node.js on avoimen lähdekoodin monialustainen JavaScript-ajoympäristö palvelinpuolen kehitykseen. Node.js käyttää tapahtumapohjaista systeemiä, joka on asynkroninen. Tämä mahdollistaa hyvän suoritustehon ja skaalautuvuuden reaaliaikaisiin sovelluksiin. Node.js käyttää paketinhallintajärjestelmä npm:ää, joka helpottaa kehittäjien työtä julkaista ja jakaa kirjastoja sekä asentaa ja päivittää kirjastoja. [7]

2.6 Redis

Redis on avoimen lähdekoodin datastore, jota voidaan käyttää tietokantana, välimuistina ja viestien välittäjänä. Redisin tukemia tietorakenteita ovat mm. merkkijonot, hashit ja listat. Redis on yksi suosituimmista avain-arvotietokannoista, ja tuki löytyy kaikista suosituimmista alustoista ja ohjelmointikielistä. Redis ei tallenna tietoa lokaalisti, vaan pitää tiedon muistissa. Tämän takia Redis on paljon nopeampi kuin perinteiset tietokannat, jotka joutuvat kirjoittamaan tiedon talteen. Koska Redis ei tallenna tietoa pysyvästi ja on nopeakäyttöinen, soveltuvat käyttökohteet keskittyvät esimerkiksi sessioiden tallennuspaikaksi tai viestijonoksi. [8]

3 TOTEUTUS

Seuraavaksi käydään läpi itse etätuen toteuttaminen. Etätuki tarvitsee palvelimen, jonka kautta viestit kulkevat, koska sovelluksessa voi olla eri käyttäjäryhmiä. Palvelin käsittelee liikenteen niin, että vain samassa käyttäjäryhmässä oleville käyttäjille lähetetään viestit, ja mahdollistaa myös viestien kuuntelun vain, jos käyttäjä kuuluu samaan ryhmään. Tässä mallissa viivettä viestin kulkemiseen tulee lisää, koska palvelin joutuu lähettämään viestin edelleen oikealle käyttäjäryhmälle. Viivettä pystyttäisiin minimoimaan, lähettämällä viestit suoraan käyttäjältä käyttäjälle, mutta etätuen pitää toimia useamman kuin kahden käyttäjän välillä, ja käyttäjät eivät saa tietoa muista käyttäjistä, jolloin palvelimen pitää hoitaa viestien uudelleen lähetys.

3.1 Palvelintoteutus

Palvelinpuolella on kaksi erillistä komponenttia. Yksi komponenteista hoitaa käyttäjien varmennuksen, ja toinen komponenteista lähettää käyttäjän datan eteenpäin huoneessa tai sessiossa oleville käyttäjille. Kuvassa 3 on esimerkki, miten erillinen komponentti voidaan lisätä Crossbarin kokoonpanoasetuksiin.

```
{
    "type": "guest",
    "executable": "node",
    "arguments": [
        "authenticator.js",
        "ws://127.0.0.1:8080",
        "realm1"
    ]
}
```

Kuva 3. Komponentin määritys

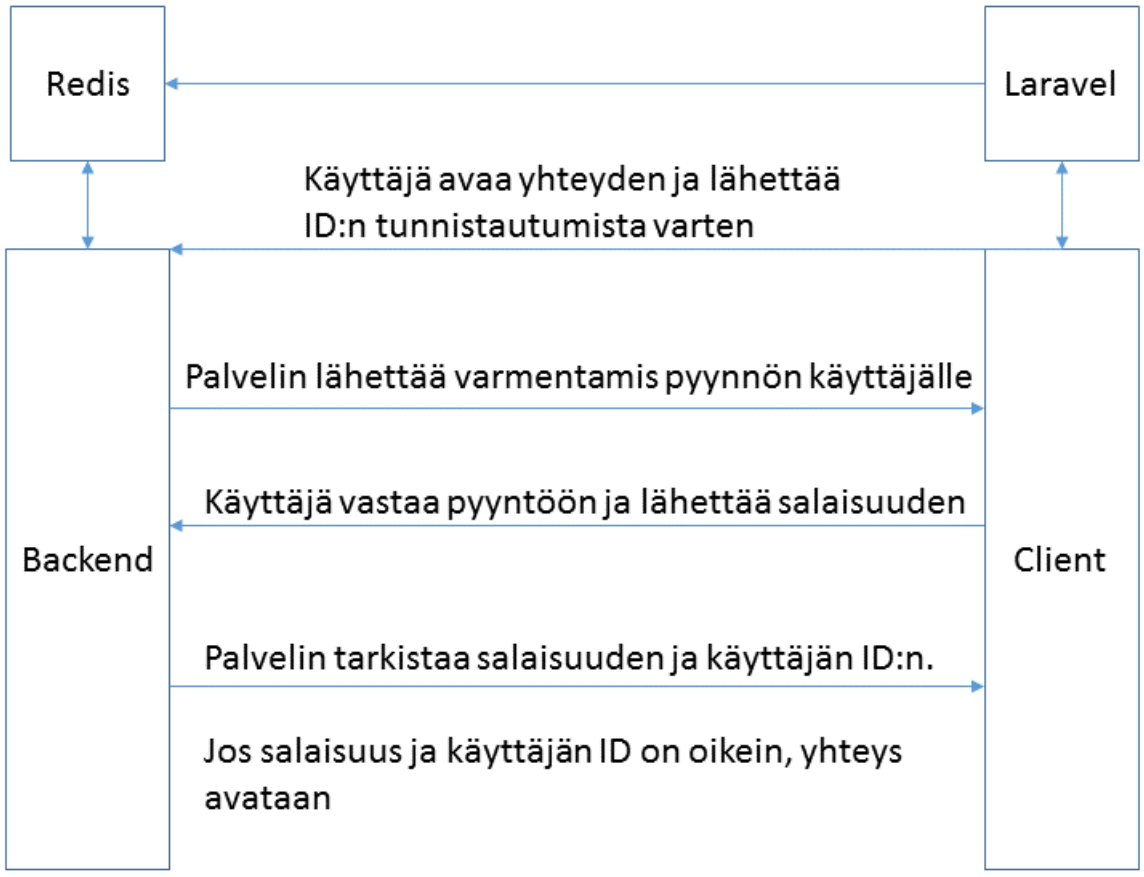
Erilliset komponentit helpottavat modulaarista kehitystä ja selkeyttävät koodia. Tässä tapauksessa komponentin lisääminen on pakollista käyttäjien dynaamiseen

varmennukseen. Kaikki komponentit ottavat erikseen yhteyden reitittimeen, minkä vuoksi komponentit myös varmentavat itsensä itsenäisesti. Autobahn.js on käytössä yhteydenluontiin myös palvelinpuolella.

Käyttäjät viestivät palvelimen kanssa julkaisemalla viestin tai kutsumalla etäfunktiota. Palvelin tarkistaa käyttäjien käyttöoikeudet yhteydenluonnin yhteydessä, ja tämän perusteella käyttäjät oikeutetaan vain tiettyihin aiheisiin tai etäfunktioihin.

3.2 Yhteyden luonti

Yhteyden luonti reitittimeen tapahtuu asiakaspäässä ja palvelimella käyttäen Autobahn.js kirjastoa. Kuvassa 4 esitellään, miten käyttäjän varmennus tapahtuu yhteyden luonnin yhteydessä etäpalvelimen ja sovelluksen välillä.



Kuva 4. Varmentaminen

Asiakaspäässä haetaan ensin käyttäjän ID sovelluksen palvelimelta, jonka jälkeen käyttäjän ID ja huone tallennetaan Redisiin ja lähetetään takaisin käyttäjälle yhteyden avaamista varten. Kun palvelin saa yhteydenavauspyynnön, se lähettää käyttäjälle tunnustautumispyynnön, jossa käyttäjän pitää lähettää salaisuus palvelimelle. Tämän jälkeen, jos ID ja salaisuus ovat oikein, yhteys avataan. Kuvassa 5 on esimerkki, miten dynaaminen todennusmenetelmä määritetään WebSocket-yhteyksille.

```
"ws": {
    "type": "websocket",
    "auth": {
        "wampcra": {
            "type": "dynamic",
            "authenticator": "com.application.authenticate"
        }
    }
}
```

Kuva 5. Dynaaminen todennus

Kaikelle WebSocket liikenteelle voidaan määritellä dynaaminen todennusmenetelmä Crossbarin kokoonpanoasetuksista. ”com.example.authenticate” on käytännössä todennuskomponentin rekisteröimä etäfunktio, jota kutsutaan aina ennen yhteyden avaamista. Kaikki määritellyt roolit kutsuvat täten tätä todennusta ensin ennen kuin yhteys avataan reitittimeen, mutta ainoastaan asiakas ja avustaja varmennetaan Redisin kautta.

3.3 Käyttöoikeudet

Rooleina on kokoonpanoon määritelty palvelin, todentaja, asiakas ja avustaja. Jokaiselle roolille on kokoonpanoasetuksissa määritelty erikseen mihin aiheisiin roolilla on oikeus julkaista ja kuunnella, tai kutsua ja rekisteröidä etäfunktio, ja missä tilanteessa käyttäjä voidaan tunnistaa. Kuvassa 6 on määritelty asiakaspään rooli Crossbarin kokoonpanoasetuksiin.

```
"name": "client",
"permissions": [
{
    "uri": "com.application.frontend",
    "match": "prefix",
    "allow": {
        "call": false,
        "register": true,
        "publish": false,
        "subscribe": true
    },
    "disclose": {
        "caller": false,
        "publisher": false
    }
}
```

Kuva 6. Oikeuksien määrittäminen

Ensimmäisenä määritellään aihe tai etäfunktion nimi, jonka jälkeen määritellään, että oikeudet koskevat kaikkia ”com.example.frontend.*” alkuisia aiheita. Asiakkaalta on tässä tapauksessa kielletty etäfunktion kutsuminen tai rekisteröiminen, tai aiheeseen julkaisu ja tiedon saaminen julkaisijasta. Ainut sallittu toiminto asiakkaspäälle tässä aiheessa on kirjautuminen kuuntelemaan aiheesta saapuvia viestejä.

Laravel tallentaa Redisiin käyttäjän ID:n lisäksi myös tämän ennalta määritellyt oikeudet. Jos yhteyden avaajalla on oikeus antaa etätukea, käyttäjä varmennetaan avustajan rooliin. Jos käyttäjällä ei ole tätä oikeutta, käyttäjälle annetaan pelkkä asiakkaan rooli.

3.4 Datan lähettäminen

Elementtien synkronoiminen sovelluksessa tapahtuu lähettämällä tarvittava data, esimerkiksi valitun tai päivitetyn elementin ID palvelimelle, jonka jälkeen palvelin

lähettää datan eteenpäin oikeille vastaanottajille. Kuvassa 7 on esimerkki, miten dataa voidaan lähettää.

```
application.connect.synchronize('selectElement', elementID);
```

Kuva 7. Datan lähettäminen

Jotta kyseinen elementin valinta voidaan synkronoida toisen käyttäjän koneella, pitää samalle tapahtumalle luoda myös kuuntelija, joka tekee elementin päivityksen myös toisella koneella. Esimerkki tästä kuvassa 8.

```
application.$on('selectElement', (elementID) => {
      this.selectedElement = elementID;
});
```

Kuva 8. Datan vastaanottaminen

Esimerkki kuvassa 7 julkaisee aiheeseen “com.application.sync.selectElement” päivityksen, ja lähettää elementin ID:n. Ainoastaan käyttäjillä, jotka ovat varmentaneet oman WebSocket-yhteyden roolille avustaja, on oikeus julkaista viestejä aiheeseen. Tämän jälkeen palvelinpuolella on kirjauduttu kuuntelemaan aihetta, ja kun aiheeseen julkaistaan päivitys, ohjaa palvelin viestin käyttäjille aiheeseen, johon pelkästään palvelimen rekisteröityneillä komponenteilla on julkaista. Tämä on tietoturvan takia tärkeää, ettei kuka tahansa pysty julkaisemaan järjestelmään viestejä. Aihe johon viesti julkaistaan, ja aihe josta viesti vastaanotetaan eroavat sen takia, että myöskään avustajaan ei luoteta liikaa, vaan palvelinpuolen komponentit ohjaavat viestit oikean huoneen käyttäjille. Tietoa päivitetään myös palvelinpuolelta, esimerkiksi huoneessa olevista ja etätuen hyväksyneistä käyttäjistä lähetetään tieto ainoastaan avustajalle.

Koska etätuki päivittää ainoastaan elementtejä, ei etätukeen vaikuta esimerkiksi erilaiset resoluutiot käyttäjien välillä. Avustaja voi näyttää, mistä joku asia sovelluksessa löytyy, ja sama asia voidaan korostaa myös toisella käyttäjällä, vaikka sivun asettelu ja näytön resoluutio olisikin erilainen. Ainoastaan tilanteissa, joissa

näkymä voi olla piilotettu todella pienen resoluution takia, esimerkiksi mobiilinäkymässä, voi ongelmia syntyä, jos toiselta käyttäjältä on piilotettu tietty näkymä kokonaan.

Jotta sovelluksesta saadaan reaaliaikainen, on tiettyjä ominaisuuksia, jotka voidaan synkronoida myös ilman että käyttäjä on etätuen piirissä. Esimerkiksi jos käyttäjä luo, muokkaa tai poistaa uusia objekteja, voidaan objekti myös päivittää muille käyttäjille reaaliajassa. Tämän toteuttamiseen on käytetty Redisiä, joka voi toimia avain-arvotietokannan lisäksi myös aiheen julkaisu tai kirjautumiskanavana. Dataa ei lähetetä suoraan asiakaspäästä, vaan Redis hoitaa sen Laravelin palvelinpuolelta objektin luonnin, poiston tai muokkauksen yhteydessä, koska siellä voidaan luotettavasti varmentaa ja tallentaa ensin tietokantaan muutokset. Kuvassa 9 lähetetään objektin päivitysviesti siihen huoneeseen missä käyttäjä on.

```
$room = $request->session()->get('room')->hash;
Redis::publish('com.application.redis.' . $room, $object);
```

Kuva 9. Objektien päivittäminen

Julkaisun aiheeseen lisätään huoneen ID, jolloin viesti lähetetään ainoastaan sisään kirjautuneen käyttäjän huoneeseen. Tämä on tärkeää tietoturvan kannalta, etteivät tiedot mene toiseen huoneeseen väärille käyttäjille. Lähetetystä objektista voidaan palvelinpuolella katsoa, mitä tyyppiä se on, ja mikä toiminto on kyseessä, esimerkiksi luominen tai päivittäminen.

Kun julkaisu saapuu palvelimelle, viesti julkaistaan uudelleen, mutta tällä kertaa WebSocket-yhteyden kautta, jotta viesti voidaan ohjata oikean huoneen käyttäjille suoraan. Kun käyttäjä kuuntelee ja vastaanottaa aiheen, objekti voidaan luoda, muokata tai poistaa sovelluksessa. Näin käyttäjän ei tarvitse avata sovellusta uudelleen nähdäkseen muiden käyttäjien tekemiä muutoksia.

Objektien luonnin ja päivittämisen lisäksi kaikilla käyttäjillä on mahdollisuus myös lähettää ja vastaanottaa viestejä huoneeseen. Chat on palvelinpuolella rekisteröity etäfunktio, jota kaikilla käyttäjillä on oikeus kutsua. Kun käyttäjä lähettää viestin

etäfunktiolle, haetaan käyttäjän nimi ID:n perusteella. Tämän jälkeen käyttäjän viesti ja nimi julkaistaan kaikille huoneessa oleville käyttäjille. Käyttäjä ei voi itse lähettää omaa nimeään asiakaspäässä, vaan nimi tarkistetaan ja lisätään viestiin palvelinpuolella, jotta toisen käyttäjän nimissä ei voida lähettää viestejä huoneeseen.

3.5 Huoneet ja käyttäjien hallinta

Huonepohjaisen järjestelmän käyttäminen auttaa lähettämään viestit oikealle käyttäjäryhmälle. Huoneita käytetään lähettäessä viesti etätuen avustajalta käyttäjille. Kun viesti halutaan julkaista, voidaan viesti julkaista ainoastaan tietylle käyttäjäryhmälle, joka voi muodostua useamman käyttäjän ID:stä. Tästä syystä huoneisiin tallennetaan käyttäjien ID:t, jolloin voidaan tarkistaa viestin lähettäjän huone, ja sen jälkeen julkaista viesti ainoastaan tämän huoneen käyttäjille. Käyttäjän todennuksen yhteydessä tarkistettiin käyttäjän huoneen ID. Tämän ID:n perusteella käyttäjät voidaan lisätä omiin huoneisiinsa.

Käyttäjät lisätään kahteen eri huoneeseen Tämä siitä syystä, että ensimmäisenä tarvitaan rajata viestit oikean huoneen käyttäjille. Ensimmäiseen huoneeseen käyttäjät lisätään heti todennuksen jälkeen. Toiseen huoneeseen käyttäjät lisää avustaja. Tämä huone on niitä viestejä varten, jotka kulkevat vain etätuen ollessa päällä. Ainoastaan avustaja voi kutsua käyttäjiä ensimmäisestä huoneesta etätukihuoneeseen, jonka jälkeen kaikki etätukiominaisuudet ovat toiminnassa ainoastaan niille käyttäjille, jotka ovat huoneessa. Käyttäjien on hyväksyttävä ensin avustajan kutsu, ennen kuin he ovat etätukihuoneessa. Tämä siitä syystä, että avustaja voi tehdä muutoksia sovellukseen. Avustajalla on myös oikeus potkia käyttäjiä pois etätuesta, tai lopettaa koko etätukisessio. Avustaja on myös ainut, jolla on tieto joko huoneessa olevista käyttäjistä, tai etätukipyynnön hyväksyneistä käyttäjistä.

Toinen vaihtoehto, jota olisi voitu käyttää sen sijaan että käyttäjiä lisätään itse huoneisiin ja sen jälkeen julkaista viestit vain tiettyyn huoneeseen, on dynaamiset real-

mit. Crossbar julkaisi tuen dynaamisien realmien luonnille 17.3.1 versiossa. Realmit ovat Crossbarin vastine huoneille, mutta koska realmeja ei ollut mahdollista luoda muuta kuin staattisesti määrittelemällä ne kokoonpanoasetuksissa, ei tätä voitu käyttää tätä hyväksi työssä, koska tarvittiin mahdollisuus luoda huoneita dynaamisesti. Kun käyttäjä luo yhteyden palvelimelle, Crossbar yhdistää käyttäjän ennaltamääriteltyyn realmiin. Tämän jälkeen käyttäjä voi vastaanottaa tai lähettää viestejä vain tässä tietyssä realmissa. Koska huoneita voidaan luoda sovellukseen käyttäjien tai työn tilaajan toimesta, tarvittiin mahdollisuus dynaamisiin huoneisiin.

4 TULOKSET

Sovellukseen saatiin osittainen reaaliaikainen etätuki, jotkin sovelluksen osa-alueet jäivät synkronoimatta, koska sovellus oli koko opinnäytetyön ajan jatkuvassa kehityksessä, jolloin tiettyjä ominaisuuksia ei vielä edes löytynyt sovelluksesta työn aloitusvaiheesta. Tämä olisi voitu välttää tekemällä etätuesta vieläkin joustavampi, jotta jokaista sovelluksen ominaisuutta ei tarvitsisi erikseen synkronoida toisille käyttäjille. Toteutettu etätuki soveltuu koulutuskäytön lisäksi myös esimerkiksi suunnittelun avuksi, koska sen avulla voidaan sovelluksesta näyttää osa-alueita tai osoittaa haluttuja elementtejä.

Sovellusta testattiin ulkomaalaisella palvelimella niin, että itse sovellus sekä etätukipalvelin sijaitsivat molemmat samalla palvelimella. Toisaalta palvelimien hajauttaminen voisi myös lisätä tietoturvaa, mutta koska etätukipalvelimen ja sovelluksen välinen tieto ei kulje palvelimen ulkopuolelle, MITM-hyökkäyksien todennäköisyys vähenee. Koska sovellus ja etäpalvelin sijaitsevat samalla palvelimella, myös viive vähenee käyttäjien todennuksessa.

Hierarkkisuuden toteuttaminen palvelinpuolella onnistui hyvin. Alun perin viestit kulkivat etätuessa ainoastaan avustajalta koulutettaville, mutta hierarkkisuus lisäsi myös viestien kulkemisen koulutettavilta takaisin avustajalle päin. Ongelmaksi nousi itse datan visualisointi avustajan päässä. Esimerkiksi, miten visualisoidaan, että usea eri käyttäjä on sovelluksessa painanut tiettyä elementtiä, tai mennyt vaikka kokonaan eri huoneeseen. Jos huoneessa on paljon käyttäjiä, niin avustajan sovellus saattaa täyttyä listasta käyttäjiä eri kohtaa sivulla, tai avustaja näkee vain yhden käyttäjän kerrallaan.

Salatun WebSocket-yhteyden eli WSS:n lisääminen lisäsi varmuutta WebSocket-yhteyksien muodostamisessa, ja lisää myös tietoturvaa, kun yhteys on salattu. Testeissä ei havaittu, että WebSocket-yhteyksiä olisi hylätty välityspalvelimien tai palomuurien takia, jota voi sattua, jos yhteyttä ei ole salattu.

Tällä hetkellä etätuki toimii kahden huoneen periaatteella, mutta koska sovellusta kehitettiin jatkuvasti opinnäytetyön tekemisen ajan, tuli tarve myös tehdä huoneita huoneiden sisälle. Tätä varten etätukeen tarvittaisiin jatkokehitystä varten lisätä mahdollisuus lisätä huoneita joiden sisälle voi luoda huoneita joiden sisälle voi luoda huoneita. Toinen vaihtoehto olisi käyttää Crossbarin uusia ominaisuuksia jotka mahdollistavat realmien dynaamisen luonnin ja käyttää niitä huoneiden sijasta osittain.

Toinen jatkokehityksen kohde olisi mahdollinen ääni-chat etätuen viestinnän avuksi, joka olisi mahdollisesti parasta toteuttaa WebRTC-protokollalla. WebRTC soveltuu paremmin äänen suoratoistoon kuin pelkkä WebSocket-yhteys.

Etätuen palvelin tukee etätuen hierarkkisuutta, mutta asiakaspään toteutus jäi joidenkin ominaisuuksien kohdalla vajaaksi. Mutta koska palvelinpuoli on valmis, on asiakaspään muutokset helppo toteuttaa mahdollisessa jatkokehityksessä.

5 YHTEENVETO

Opinnäytetyön tavoitteena oli suunnitella ja kehittää etätuki jo osittain käytössä olevalle verkkosovellukselle. Etätuen tarkoitus on auttaa sovelluksen markkinoinnissa sekä asiakkaiden ja käyttäjien kouluttamisessa. Tavoitteena oli myös suunnitella etätuesta hierarkkinen, eli käyttäjien toimet etätuen alaisuudessa heijastuisivat etätuen antajalle.

Etätuen palvelinpuolen integrointi sovellukseen onnistui hyvin, käyttäjien ylläpito ja lisääminen sekä poistaminen huoneista mahdollistaa tulevienkin ominaisuuksien helpon synkronoinnin käyttäjien välillä, ilman että palvelinpuolta tarvitsee muokata, riittää vain, että tiedon lähettämisen ja vastaanottamisen hoitaa sovelluksessa asiakaspäässä. Tämä mahdollistaa helpon jatkokehitysmahdollisuudet etätuelle.

Sovelluksessa muodostetaan nyt salattu WebSocket-yhteys, jonka avulla etätukea voidaan antaa kaikille samassa huoneessa oleville kutsun hyväksymisen jälkeen. Viestinnän helpotukseksi sovellukseen lisättiin myös chat-ominaisuus, jonka avulla avustaja voi neuvoa sovelluksen käytössä.

LÄHTEET

(1) WebSocket. 2017; Available at: https://en.wikipedia.org/w/index.php?title=WebSocket&oldid=773412146. Accessed Apr 4, 2017.

(2) WAMP - Web Application Messaging Protocol. Available at: http://wamp-proto.org/. Accessed Apr 4, 2017.

(3) Unified Application Routing - Why WAMP? Available at: http://wamp-proto.org/why/. Accessed Apr 4, 2017.

(4) Crossbar.io. Available at: http://crossbar.io. Accessed Apr 4, 2017.

(5) Challenge-Response-Authentication. Available at: http://crossbar.io/docs/Challenge-Response-Authentication/. Accessed Apr 4, 2017.

(6) Autobahn|JS. Available at: http://autobahn.ws/js. Accessed Apr 4, 2017.

(7) About Node.js. Available at: https://nodejs.org/en/about/. Accessed Apr 4, 2017.

(8) Redis. Available at: https://en.wikipedia.org/wiki/Redis. Accessed Apr 4, 2017.