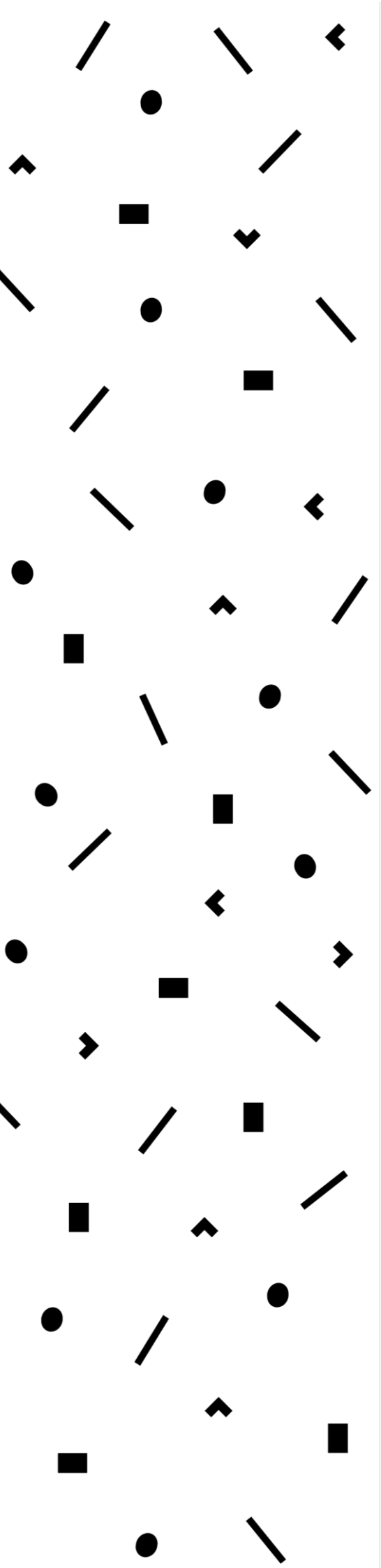


LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

# Keskustelupalvelu hybridiso-velluksille

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikka
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2016
Joni Pieviläinen

Lahden ammattikorkeakoulu
Tietotekniikka

PIEVILÄINEN, JONI: Keskustelupalvelu hybridisovelluksille

Ohjelmistotekniikan opinnäytetyö, 42 sivua

Kevät 2016

TIIVISTELMÄ

---

Opinnäytetyössä tarkastellaan keskustelupalvelua, joka on toteutettu Node.js-palvelinympäristöä käyttämällä. Ohjelmistotalo Koodiavain Oy antoi toimeksiannon opinnäytetyölle.

Yrityksellä oli ilmennyt tarvetta keskustelupalvelurajapinnalle ja palveluympäristölle, jossa käyttäjät voivat keskustella muiden käyttäjien kanssa. Käyttäjien pitäisi pystyä hallitsemaan omia keskustelujaan haluamallaan tavalla. Lisäksi palvelun täytyi pystyä toimimaan itsenäisesti sen käynnistämisen jälkeen.

Node.js valittiin keskustelupalvelun alustaksi siksi, että palvelulle voisi hyödyntää uudenaikaista ja tehokasta suoritusteknologiaa. Tiedonvarastointia varten valittiin MongoDB, jonka valintaperusteena oli sen kevyt ja nopea toiminta. Käyttäjien keskustelujen välittämiseen valittiin Socket-teknologia, jolla käyttäjät saatiin vaivattomasti yhdistettyä toisiinsa. Käyttäjien keskustelujen välittämiseen valittiin Socket-teknologia, jolla käyttäjät saatiin vaivattomasti yhdistettyä toisiinsa.

Opinnäytetyön lopputuloksena syntyi vaadittu keskustelupalvelu, jota käytetään osana toista sovellusta. Keskustelupalvelu odottaa testausta ja käyttöönottoa.

Asiasanat: keskustelupalvelu, Node.js, JavaScript, Socket, MongoDB

Lahti University of Applied Sciences

Degree Programme in Information Technology

PIEVILÄINEN, JONI: Chat Service for a hybrid application

Bachelor's Thesis in Software Engineering, 42 pages

Spring 2016

ABSTRACT

---

This thesis deals with a chat service that was made with the Node.js server environment. The thesis was commissioned by Ohjelmistotalo Koodiavain Oy.

The company detected a need for a chat service interface and service environment where the users are able to communicate with each other. Users should be able to manage their own conversation preferences. Additionally, the service had to be able to run independently without administrator interaction after the initial service startup.

Node.js was chosen as the platform for the chat service, so that the service could utilize modern and efficient processing technology. MongoDB was chosen as the information storage technology for its lightweight and fast operations. Socket technology was chosen for the user communication. It allows connecting the users together easily.

The outcome of the thesis was the required chat service, which is used as a part of another application. The chat service is awaiting testing and deployment.

Key words: chat service, chat, Node.js, JavaScript, Socket, MongoDB

LYHENNELUETTELO

| | |
|---|---|
| AJAX | Asynchronous JavaScript And XML |
| APN | Apple Push Notification-palvelu |
| GCM | Google Cloud Messaging-rajapinta |
| BSON | Binary JSON |
| JIT | Just In Time |
| JSON | JavaScript Object Notation |
| NPM | NPM-paketinhallinta |
| MySQL | Relaatiotietokantaohjelma |
| SQL | Kyselykieli relaatiotietokannoille |
| TCP | Transmission Control Protocol |
| URL | Uniform Resource Locator |

SISÄLLYS

# 1 JOHDANTO

Nykypäivänä keskustelupalveluiden määrä on kasvanut vauhdilla. Valtaosalla maailman ihmisistä löytyy jonkinlainen laite, jolla voidaan käydä keskustelua verkon kautta. Ihmiset ovat siis tottuneet tavoittamaan läheisiään helposti ja nopeasti. Keskustelupalveluiden tarjonta on miltei loputon, ja jopa yksinkertaisimmista asioista voi löytää pienenkin keskustelupalvelun.

Yleisimmät keskustelupalvelun ominaisuudet ovat käyttäjien yhdistäminen toisiinsa. Ihmiset saattavat haluta keskustella yksityisesti jonkun ystävänsä kanssa tai avoimesti muiden ihmisten kanssa. Keskusteluja on parikeskusteluista ryhmäkeskusteluihin, joiden rajana ovat vain palveluntarjoajan asettamat rajat.

Mobiililaitteiden yleistyttyä on keskustelupalveluidenkin täytynyt kehittyä valtavirran mukana. Keskustelut halutaan käydä entistä itsenäisemmin ja omakohtaisemmin. Oman laitteen käyttäminen on lähes päivänselvä asia keskusteluja käytäessä.

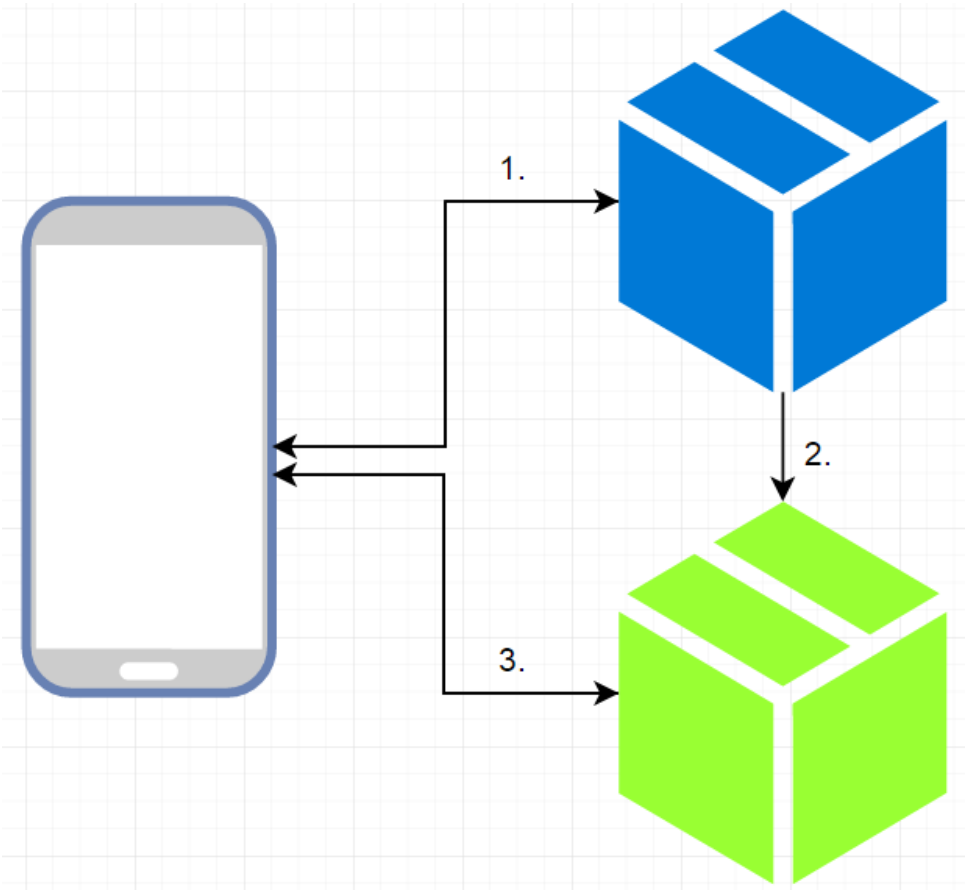
Keskustelupalvelu toteutettiin Ohjelmistotalo Koodiavain Oy:lle. Yritys aloitti toimintansa vuonna 2006 ja toimi tällöin vielä nimellä MN-Ohjelmisto Oy. Vuodesta 2008 lähtien yritys rekisteröi apunimen Ohjelmistotalo Koodiavain. Yrityksen toimipisteet sijaitsevat Nurmijärvellä ja Lahdessa. Asiallinen palveluntarjonta on ohjelmistokehittäminen sekä konesali- ja integraatiopalveluitten konsultointi.

Opinnäytetyön luvussa 2 käydään lävitse palvelun kehitys- ja toimintaympäristö, jossa palvelua suoritetaan. Luvussa 3 käydään lävitse palvelussa käytettyjä teknologioitten teoriaa ja esimerkkejä käyttötarkoituksista. Luvussa 4 esitetään ja käydään lävitse palvelun rakenne ja toiminnallisuus. Luvussa 5 käydään lävitse palvelun käyttöön vaadittavat alustukset ja palvelun suorittaminen.

# 2 TYÖN LÄHTÖKOHDAT

Keskustelupalvelun kehittäminen alkoi olemassa olevan hybridisovelluksen tarpeesta viestitellä muiden käyttäjien kanssa. Tällä sovelluksella oli valmis käyttäjäryhmä, joka täytyi huomioida keskustelupalvelun suunnittelussa.

Ennalta olevalla sovelluksella on oma palveluympäristönsä, joka keskustelee käyttäjien kanssa. Kyseisessä palveluympäristössä on tietoja, joita haluttiin tuoda keskustelupalveluun. Nämä tiedot koskivat lähinnä käyttäjiä.




KUVIO 1. Keskustelupalvelun liitettävyys

Kuviossa 1 esitetään, kuinka keskustelupalvelun tulee toimia ennalta tehdyn palvelun kanssa. Kuiviossa 1 esitetty sininen palvelu-kuvake kuvastaa ennalta olevaa palvelua hybridisovelluksien kanssa. Kuvion puhelin-kuvake esittää palvelun hybridisovelluksia ja vihreä palvelu-kuvake esittää keskustelupalvelua.

Kuviosta 1 kohta 1. esittää, kuinka palvelu on ennalta toiminut. Hybridisovellukset ovat kommunikoineet vuorovaikutteisesti olemassa olevan palvelun kanssa. Kohta 2. esittää, kuinka vanhasta palvelusta haetaan tietoja keskustelupalveluun ensimmäisellä kerralla. Kohdassa 3. esitetään, kuinka keskustelupalvelu alkaa toimia itsenäisesti hybridisovellusten kanssa vuorovaikutteisesti.

# 3 PALVELINTEKNOLOGIA

## 3.1 Node.JS

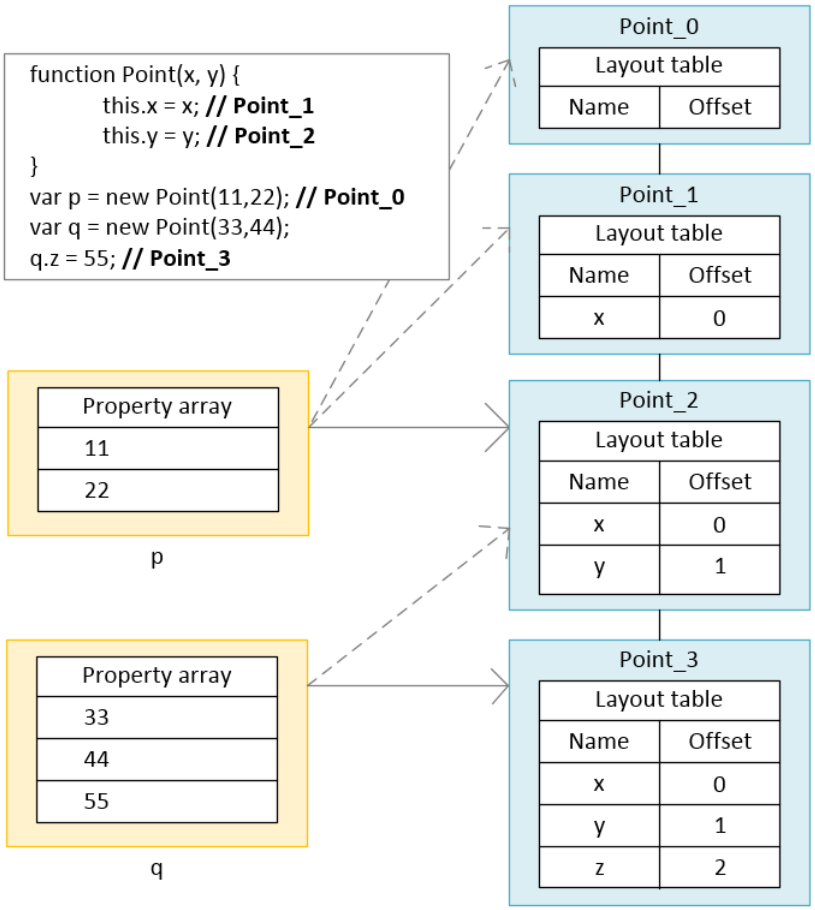
Node.js on avoimeen lähdekoodin perustuva palvelinympäristö, jolla voidaan suorittaa JavaScript-ohjelmaa palvelimella. Ensimmäisen Node.js nimeä kantava kehitysympäristö julkaistiin vuonna 2009 Git-versionhallintaan ja kehitysympäristö on edelleen samanniminen. Samana vuonna Node.js sai ensimmäisen laajennettavan pakettihallintansa, joka tunnetaan tänä päivänä NPM-paketinhallintana. (RisingStack Engineering 2016.)

### 3.1.1 Kehitysmoottori

Node.js:n moottorina toimii avoin Chromen V8-kehitysmoottori, joka tukee kaikkia yleisimpiä käyttöjärjestelmiä. V8-moottori on kehitetty C++-kieltä käyttämällä ja erityisesti JavaScriptin tukemiseksi. (Laurens 2016.)

Lähtökohtainen käyttötarkoitus Chrome V8-moottorilla on ollut parantaa suorituskykyä selaimessa. Moottorin tehokkuus tulee esille JavaScripti-koodin kääntämisellä konekoodiksi, kun taas perinteisesti JavaScriptiä tulkattaisiin suorituksen aikana. Konekoodiksi kääntäminen tapahtuu JIT-teknologian avulla. (Laurens 2016.)

Kyseessä on prototyyppipohjainen ohjelmointikieli, jolla tarkoitetaan sitä, että kieli ei sisällä luokkia, vaan se sisältää ainoastaan kloonattuja objekteja. Tällä tarkoitetaan sitä, että JavaScripti mahdollistaa dynaamisen tietojen muokkaamisen. Chrome V8 sen sijaan toteuttaa piilotettuja luokkia, joiden perusteella se esittää objektit, joilla päästään objektin ominaisuuksiin nopeasti. (Laurens 2016.)



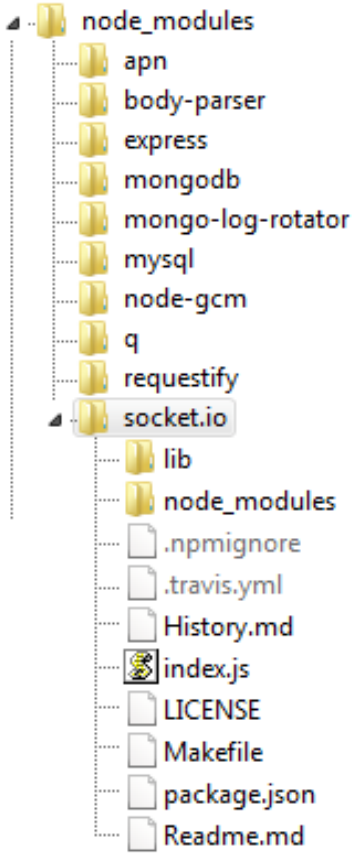

KUVIO 2. Chrome V8:n piilotettujen luokkien idea (Laurens 2016)

Kuviossa 2 esitetään, kuinka Chrome V8 luo uuden piilotetun luokan JavaScriptin konstruktoritoiminnon mukaisesti. Samassa kuviossa esitetään, miten samaa konstruktoritoimintoa käyttämällä tehdään useampia piilotettuja luokkia. Uusi piilotettu luokka käyttää edellistä piilotettua luokkaa ja tekee tämän pohjalta uuden piilotetun luokan.

### 3.1.2 Laajennukset

Node.js:n laajentaminen onnistuu NPM-paketinhallintaa käyttämällä. Laajennuksista puhutaan yleisesti moduuleina ja ne asentuvat Node.js-projektin sisälle kansioon node_modules. Kuviossa 3 nähdään perinteinen laajennus tai yleisemmältä nimeltä moduuli kansiorakenne. Rakenteesta huomataan, että moduulit jakautuvat moduulikohtaisiin kansioihinsa ni-

mensä mukaisesti. Näiden kansioiden sisältä löydetään rajapintatiedosto index.js, joka käyttää moduulin kirjastoja ja toiminnallisuuksia.




KUVIO 3. Node.js:n laajennuskansion rakenne

Moduulit pystytään jakamaan kahteen ryhmään: ydinmoduulit ja tiedostomoduulit. Lähtökohtaisesti käytetään ydinmoduuleita, joita NPM- paketinhallinta luo oletuksena. Ydinmoduulin kansiorakenteen on oltava standardin mukainen toimiakseen oikein. Mikäli kansiorakenne ei ole standardin mukainen, se tulkitaan automaattisesti tiedostomoduuliksi. Tiedostomoduuleita käytetään yleensä pienemmissä ja epävirallisissa laajennuksissa. Esimerkiksi JSON-pohjaiset listat ovat vain yhden tiedoston omaavia kirjastoja, joihin ei kannata luoda ydinmoduulin toiminnallisuutta.

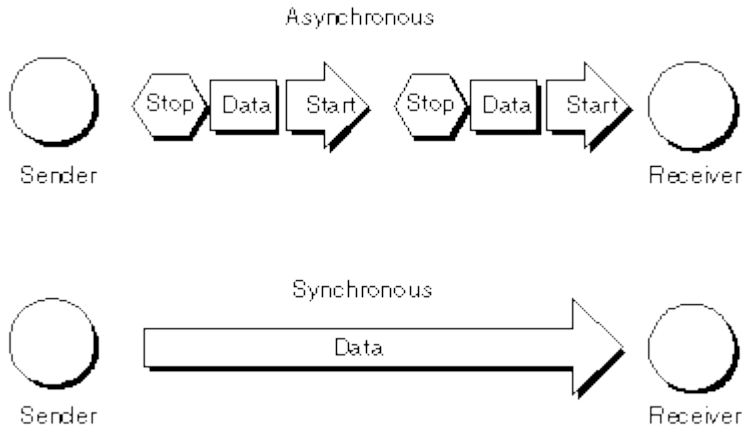
```
$ npm install forever -g
```

KUVIO 4. Julkisen laajennuksen asennus

Kaikki laajennukset eivät ole pelkästään sidonnaisia projektin kanssa. On olemassa laajennuksia, jotka eivät välttämättä vaikuta projektin toiminnallisuuteen vaan tuottavat lisätoiminnallisuutta Node.js:n käyttöön. Kuviossa 4 esitetään laajennuksen asennuslause, jolla ei ole projektin kannalta toiminnallista vaikutusta. Kyseinen laajennus mahdollistaa Node.js:n ajamisen pitkäaikaisesti taustalla. Kuviosta huomataan ”-g”-optio, jolla viitataan paketinhallinnassa julkisesti saatavissa oleviin laajennuksiin.

## 3.2 JavaScriptin asynkronisuus

JavaScript-ohjelmasuorituksessa yleisin toiminnallisuus suoritetaan asynkronisella suorittamisella. Asynkronisuudella tarkoitetaan toiminnallisuuden suorittamista riippumatta muista ohjelmasuoritteista. Asynkronisessa toimintojen sisällä on yleensä synkronista suorittamista. Esimerkiksi ajanottotoiminto olisi asynkronista toimintaa muulle ohjelmasuorittamiselle mutta itse ajanottotoiminto sisältää synkronisen ohjelmakulun ajanottamista varten.




KUVIO 5. Asynkronisuuden ja synkronisuuden ero (Webopedia 2016)

Kuvio 5 esittää asynkronisen ja synkronisen ohjelmien kulut. Kuviosta huomataan, että synkronisen ohjelman kulku on selkeä ja yksiviivainen, kun taas asynkronisen ohjelmakulku on monimutkaisempi. Kuvion 5 asynkronisuuden “Stop” ja “Start” kuvaavat toiminnallisuuden pysäyttämistä ja jatkumista “Data”-toiminnallisuutta varten. (Win8 2016.)

### 3.2.1 Takaisinkutsu

Takaisinkutsu (Callback) -toiminto on osa asynkronista ohjelmointia. Takaisinkutsuilla pyritään pitämään asynkronisen toiminnon ajojärjestys tiedossa. Asynkronisuuden toiminaan päätteeksi kutsutaan takaisinkutsua, jonka jälkeen tiedetään jatkaa muita toiminnallisuuksia.

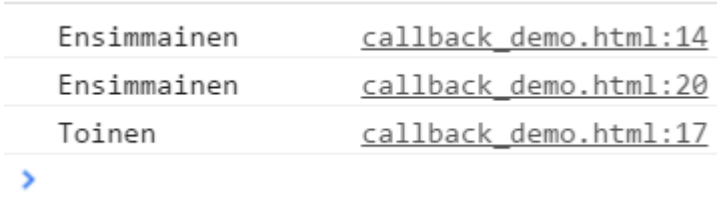
Kuviossa 6 esitetään yksinkertainen asynkroninen takaisinkutsu-toiminto. Rivillä kuusi on esitetty muuttuja, johon on asetettu teksti lähtötilanteessa. Riviltä seitsemän riville kaksitoista esitetään toiminto, jossa on toteutettu takaisinkutsu-toiminto. Toiminnon asynkroninen ominaisuus tulee ilmi rivillä kahdeksan esiintyvä ”setTimeout”-toiminnon kanssa. ”SetTimeout”-toiminnon sisällä tapahtuvat toiminnot tapahtuvat vasta sadan millisekunnin jälkeen toiminnon suorittamisesta. Rivillä kymmenen on esitetty takaisinkutsu, jolla saadaan tietää, onko ohjelman suorittaminen päättynyt asynkronisessa ohjelma-ajossa.

Kuviossa 6 rivillä kuusitoista suoritetaan ”esimerkki”-toiminto, joka käynnistää asynkronisen toiminta-ajon. Kun ”esimerkki”-toiminnolla on määritetty takaisinkutsu-toiminto, pystytään ensin suorittamaan asynkroninen toiminto ja tämän jälkeen suoritetaan rivillä seitsemäntoista kehittäjätyökaluun tulostaminen.

```
6   var muttuja = 'Ensimmainen';
7   function esimerkki(callback){
8       setTimeout(function(){
9           muttuja = 'Toinen';
10          callback();
11      }, 100);
12  };
13
14  console.log(muttuja); // >'Ensimmainen'
15
16  esimerkki(function() {
17      console.log(muttuja); // >'Toinen'
18  });
19
20  console.log(muttuja); // >'Ensimmainen'
```

KUVIO 6. JavaScriptillä toteutettu takaisinkutsu-toiminto

Kun kuvion 6 ohjelma suoritetaan, saadaan tulokseksi kuvion 7 mukainen vastaus. Kuviosta 7 huomataan, että ”Ensimmainen” teksti tulostuu kahdesti peräkkäin ja viimeisenä tulostuu ”Toinen” teksti, vaikka kuvion 6 ohjelmarakenteesta on esitetty kommentoitu tulostusjärjestys eri lailla. Jos ohjelmakulun mukainen kommentoitu tulostaminen olisi haluttu toteuttaa, olisi esimerkiksi kaikki tulosteet jouduttu siirtämään takaisinkutsun jälkeen. Tällä tarkoitetaan sitä, että kuvion 6 rivit neljätoista ja kaksikymmentä olisi siirretty rivien kuusitoista ja kahdeksantoista välille.




KUVIO 7. Takaisinkutsu toiminen suorittavuus

### 3.2.2 Promise

Promise eli “lupautuminen” on JavaScriptiin kehitteillä oleva ominaisuus, jolla hallitaan asynkronista toimintaa. Virallisesti Promisea ei ole vielä tuotu JavaScriptiin vakioksi, mutta monet kirjastot ovat ottaneet ja kehittäneet omia versioita Promisesta (Mozzilla Developer Network 2016).

Koska asynkronisen ohjelman tulkinta saattaa olla hankalaa ja takaisinkutsutoiminnallisuudet eivät ole aina standardeja, tarjoaa Promise tähän helpotusta. Promisen ideana on pitää takaisinkutsut aina samanlaisina, jotta tulkittavuus pysyisi aina samankaltaisena.

```
1  var p = new Promise(function(resolve, reject) {
2
3      // asynkroonin suorittaminen
4
5      if(/* ehtolause */) {
6          resolve('Onnituminen!');
7      }
8      else {
9          reject('Epäonnistuminen!');
10     }
11 });
12
13 p.then(function() {
14 /* vastauksen käsittely */
15 }).catch(function() {
16 /* virhe */
17 })
```

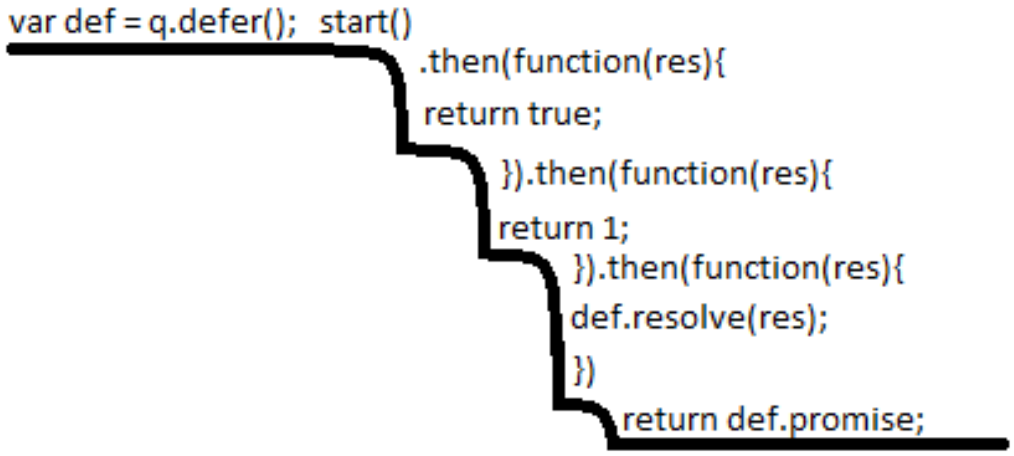
KUVIO 8. Promisen käyttöesimerkki (Walsh 2016)

Kuviossa 8 esitetään yksinkertainen Promisen käyttötilanne. Riviltä yksi riville yksitoista on esitetty asynkroninen toiminto, joka on alustettu ”p”-muuttujaan. ”P”-muuttujalle on asetettu Promise-toiminnallisuus, jolla hoidetaan asynkronista tilannetta tämän sisällä. Riveillä kuusi ja yhdeksän on Promiselle määritellyt takaisinkutsu-toiminnot, joiden perusteella pystytään tarkastelemaan asynkronisen toiminnon onnistumista.

Kuvion 8 rivillä kolmetoista on esimerkki Promisen käyttämisestä. Riviltä huomataan, että ensimmäisenä kutsutaan ”p” muuttuja, johon on ennalta asetettu asynkroninen toiminto. ”p”-muuttujan jälkeen siirrytään jälkikäsittelyyn, joka ilmaistaan ”then”-toiminnolla. Tämän toiminnon sisällä pystytään tarkastelemaan ”p”-muuttujasta tulleita vastauksia. Lopuksi kuvion 8 rivillä viisitoista esitetään varmistustilanne, mikäli Promisen toiminnallisuus pysähtyy ennenaikaisesti.

### 3.2.3 Q-kirjasto

Q-kirjasto on Promise-kirjasto, jolla hallitaan JavaScriptin asynkronisuutta ja takaisinkutsu-toimintoja. Kirjaston perusideaa kuvataan vesiputousmallilla. Q-kirjastoa pystytään käyttämään käyttäjän puollella ja Node.js-palvelimen puolella. Kirjasto on hankittavissa erillisenä kirjastona tai laajennuksena NPM-pakettienhallinnasta (Nallan 2013).




KUVIO 9. Promisen vesiputousmalli

Kuviossa 9 oleva esimerkki esittää, miten perinteinen vesiputousmalli toteutuu. Kuten luonnollisissa vesiputouksissa vesi valuu alaspäin, niin käy myös tässä esimerkissä. Vetenä tässä tapauksessa toimivat palautettavat arvot ja vesiputoustasoina toimivat ”then”-toiminnot.

## 3.3 Palvelun tiedonsiirto

Palvelun tiedonsiirrolla tarkoitetaan käyttäjän ja palvelun välistä yhteyttä. Ulospäin tiedonsiirto yleensä tapahtuu tapahtumapohjaisilla toiminnoilla, joita palvelun käyttäjät hallitsevat tietämättään. Esimerkiksi käyttäjä saattaa painaa nappia, joka hakee lisää tietoa palvelusta. Näin ollen on tapahtunut tapahtumapohjainen toiminto. Näissä tapauksissa yleensä toiminnallisuuksien aikana käyttäjän ja palvelun yhteys luodaan vain vaadittavaksi ajaksi.

Joissakin tapauksissa vaaditaan, että käyttäjät ovat jatkuvassa yhteydessä palveluun. Näitä tapauksia saattavat olla muun muassa reaaliaikaiset tiedonsiirrot. Lisäksi jatkuvassa yhteydessä pystytään seuraamaan käyttäjien paikallaoloa ja käyttäytymistä.

### 3.3.1 HTTP-kyselymetodit

Verkkosivujen kehittämisessä käytetyimmät kyselyt ovat GET- ja POST-kyselymetodit, joiden avulla voidaan välittää tietoa käyttäjän ja palvelimen välillä. Muita metodeita ovat muun muassa HEAD-, PUT- ja DELETE-metodit.

POST- ja GET-metodien käyttämisessä ja rakenteellisuudessa ei ole paljoakaan eroja. Kummassakin metodissa luodaan sisältötaulukko, joka sisältää avaimia ja niille arvoja. Taulukossa 1 esitetään kummallekin metodille sama lähtökohta ja käännösmuoto, millaisena tiedot kulkevat metodissa. Taulusta huomataan, että ero tulee viestien välityksessä.

TAULUKKO 1. POST- ja GET-kyselyt

| | Lähtökohtaiset arvot | Metodissa esitetty muoto |
|---|---|---|
| POST | id: 1<br>nimi: ”Kalle” | id=1&nimi=”Kalle” |
| GET | id: 1<br>nimi: ”Kalle” | /index.php?id=1&nimi=”Kalle” |

GET-metodissa sisältö luodaan kyselyosoitteen perään. Metodin sisältö tulkitaan tekstiksi, joka on rajoitettu yleensä 2048 merkkiin (W3Schools 2016).

POST-metodissa sisältö luodaan muuttujaan, joka lähetetään kyselyosoitteeseen. Metodin sisältö tulkitaan tiedostomaisesti ja näin ollen POST-metodille eräs rajoittava tekijä on kokoraja, joka määritellään palvelimen

puolella. HTTP-protokollan POST-metodin tiedonsiirtorajana on 20 megatavua (NixCraft 2010).

### 3.3.2 Socket.IO

Nykyisin vaaditaan entistä enemmän reaaliaikaista tiedonsiirtoa verkkosivun ja palvelun välillä. Ennen reaaliaikaisuutta toteutettiin AJAX- toiminnolla, jolla pystyttiin siirtämään vaadittavia tietoja sovelluksen ja palvelun välillä. Kuitenkaan kyseinen toiminto ei ole välttämättä tarpeeksi tehokas jatkuvaan reaaliaikaiseen tiedonsiirtoon, jossa kommunikointi palvelun kanssa on lähes jatkuvaa. Hitaammissa verkkoyhteystilanteissa hitaus saattaa koitua myös AJAX- toimintojen kohtaloksi, jolloin saatetaan tulkita, että sivusto ei välttämättä toimi.

Socket.IO on JavaScript-kirjasto, jolla voidaan toteuttaa sovelluksen ja palvelun välinen reaaliaikainen tiedonsiirto. Kirjasto on jaettu kahteen osaan siten, että käyttäjälle ja palvelulle on oma kirjastonsa. (Cheng 2013.) Palvelun kirjastolla on tyypillisesti kattavampi toimintovalikoima, jolla pystyttään hallitsemaan asiakas-socketteja.

Socket.IO:lla on kuusi erilaista tiedonsiirtoprotokollaa. Protokollat ovat WebSocket, Adobe® Flash® Socket, AJAX long polling, AJAX multipart streaming, Forever Iframe, JSONP Polling (Cheng 2013). Ensisijainen protokolla, jota Socket.IO käyttää, on ”AJAX long polling”. Kyseiselle protokollalle on tunnetumpi nimike ”xhr-polling”, jolla tarkoitetaan HTTP-kyselyitten tuottamista (AbraEnter 2013). Socket.IO:n ideana ei ole kuitenkaan käyttää vain yhtä protokollaa, vaan se osaa tilanteesta riippuen valita itselleen sopivimman protokollan.

Socket.IO:n yleinen käyttö alkaa Socket-palveluympäristön tuottamisella. Palvelu yleisesti avataan TCP-protokollan omaavaan porttiin, jonka kautta Socketteja käyttävät sovellukset ottavat yhteyttä ja kommunikoivat palvelun kanssa (Kaazing 2014).

TAULUKKO 2. Yleiset Socket.IO-toiminnot

| | Asiakasohjelma | Palveluohjelma |
|---|---|---|
| 1. Socketin yhdistäminen | var socket = io('http://localhost'); | io.on('connection', function (socket) {}); |
| 2. Keskusteluun liittyminen | socket.emit(‘liity’, ‘keskustelu1’); | socket.on('liity', function(huone) { socket.join(huone); }); |
| 3. Viestin lähettäminen | socket.send(viesti); | io.emit(' keskustelu1', 'palvelimen tervedys'); |
| 4. Keskustelusta poistuminen | socket.emit(‘poistu, ‘keskustelu1’); | socket.on(‘poistu’, function(huone) {socket.leave(huone); }); |
| 5. Yhteyden katkeaminen | socket.on('disconnected', function(){}); | socket.on('disconnect', function(){}); |

Taulukossa 2 esitetään yleisimmät toiminnot, joilla socketteja ohjataan asiakasohjelmassa ja palvelinohjelmassa. Kohdassa yksi esitetään, kuinka asiakasohjelma ottaa yhteyden socket-palveluun ja palvelu on valmiina yhteydenluontia varten. Kohdassa kaksi esitetään, kuinka asiakasohjelma välittää pyynnön, jolla haluaa liittyä keskusteluun. Palvelu kuuntelee, että keskusteluun halutaan liittyä ja liittää käyttäjäsovelluksen socketin keskusteluun. Kohdassa kolme esitetään, kuinka viesti voidaan lähettää keskusteluun asiakasohjelmasta tai palveluohjelmasta. Kohdassa neljä esitetään kuinka keskustelusta päästään poistumaan. Poistuminen on samankaltainen kuin kohdan kaksi liittyminen. Asiakasohjelma välittää pyynnön, jolla haluaa poistua keskustelusta, ja palvelinohjelma poistaa pyynnön perus-

teella socketin keskustelusta. Kohdassa viisi esitetään ennalta arvaamaton tilanne, jossa yhteys syystä tai toisesta katkeaa.

## 3.4 Tietokannat

Yleinen tietokanta-käsite tarkoittaa tietovarastoa, jossa on kaikki palveluihin haluttavat tiedot. Tietokanta itsessään ei tee muuta, kuin pitää halutun datan käyttövalmiina muita palveluita varten. Yleisesti tietokanta voidaan jakaa kahteen erilaiseen tyyppiin: dynaamisiin ja staattisiin käyttötarkoituksiin. Esimerkiksi web-kehityksessä voidaan tehdä kysymyslomakkeesta pohja, joka on aina vakio. Näin ollen, kun kyseessä on kiinteä pohja, se voidaan tallentaa staattisesti tietokantaan ja tarvittaessa kutsua sitä. Kun kysymyslomake täytetään ja sen vastaukset halutaan tallentaa tietokantaan, käytetään dynaamista ajattelumaailmaa. Lomakkeesta voi tulla milloin tahansa melkein mitä tahansa arvoja, jotka yleensä tarkastetaan ohjelmallisesti ennen tietokantaan tallentamista.

Relaatiotietokantojen kehityksessä SQL-kieli on tehnyt pysyvimmän vaikutuksen pitkän historiansa johdosta tietokantojen kehityksessä. SQL-kieli on lähes ainut kieli tietokantoja käytettäessä.

### 3.4.1 MySQL

Relaatiotietokantojen tueksi on kehitetty hallintajärjestelmä MySQL, joka perustuu avoimeen lähdekoodiin. MySQL-hallintajärjestelmä perustuu sen sisällä oleviin varastointimoottoreihin, joilla pystytään varastoimaan tietoa tilanteesta riippumatta (MySQL 2016).

MySQL on yksi suosituimmista hallinnointijärjestelmistä. Suosion suurimmat tekijät ovat ilmainen käyttö ja saatavuus, helppokäyttöisyys ja muokattavuus. Täten MySQL on verkkosivukehittäjien kesken erittäin suosittu.

Node.js:n kannalta tarvitaan erillinen laajennus MySQL:n käyttöä varten. “MySQL”- niminen laajennusta on yksi monista Node.js:lle saatavista laajennuksista, joilla saadaan MySQL-rajapinta Node.js:n käyttöön. Kuviossa

10 riviltä yksi esitettyyn ”pool”-määrittelyyn on alustettu erillinen MySQL-kirjasto, jolla pystytään hallitsemaan tietokantaa. Määrittelyn jälkeen toteutetaan perinteinen kyselylause ja kyselyn päätteeksi toteutetaan takaisinkutsu-toiminto, jolla saadaan vastaus tietokannasta.

```
1  pool.query('SELECT id, firstname, tokenID, profile_picture, type FROM
   mt_userdevices, mt_users where mt_users.id = mt_userdevices.userid',
   function(err, rows, fields) {
2      if (err) throw err;
3      var response = rows;
4  });
```

KUVIO 10. Node.JS:n MySQL-laajennuksen esimerkkikäyttö

### 3.4.2 MongoDB

Avoimeen lähdekoodiin perustuva MongoDB on yleistyvä ei-relaatiotietokanta. MongoDB ei pohjaudu perinteisiin relaatiotietokantamalleihin eikä noudata perinteistä SQL-kieltä. Näin ollen MongoDB:n käyttämiseen on kehitetty omakohtaiset kyselyt. Jos tietokannalta vaaditaan relaatiota, se tulee toteuttaa MongoDB:tä käyttävässä sovelluksessa. Tämä tekee MongoDB:stä yhden suosituimmista ei-relaatiotietokannoista.

MongoDB käyttää tietojen varastoinnissa JSON-formaattia, jossa jokainen tieto tallennetaan avaimella. Virallinen varastointitapa on kuitenkin BSON, joka toimii muuten samalla lailla kuin JSON, mutta prosessiin vielä lisätään binääreiksi kääntäminen. Täten tieto on JSON helppoudella käytettävissä, binäärimuodon ansiosta kevyesti ja nopeasti saatavissa. (MongoDB 2016c.)

TAULUKKO 3. MySQL:n ja MongoDB:n termistöjen vertailu (MongoDB 2016d)

| MySQL | MongoDB |
|---|---|
| Table | Collection |
| Row | Document |
| Column | Field |
| Joins | Embedded documents, linking |

Totuttuun MySQL-termistöön löytyy MongoDB:stä omat terminsä. Taulukossa 3 on esitettynä, mitkä MySQL:n termit vastaavat MongoDB:n termejä. Alemmassa taulukossa 4 esitetään esimerkkejä, kuinka samankaltaisia asioita toteutetaan MysSQL:lla ja MongoDB:llä. Taulukosta 4 voidaan huomata selkeä objektimainen käytettävyys MongoDB:ssä, mikä on tyypillistä kyseisen tietokannan kanssa.

TAULUKKO 4. MySQL:n ja MongoDB:n samankaltaiset toiminnot (MongoDB 2016d)

| | MySQL | MongoDB |
|---|---|---|
| Tietokantaan uuden sisällön lisääminen | INSERT INTO users (user_id, age, status) VALUES ("bcd001", 45, "A") | db.users.insert({ user_id: "bcd001", age: 45, status: "A" }) |
| Tietokannasta tiedon hakeminen | SELECT * FROM users | db.users.find() |
| Tietokantaan tietojen päivittäminen | UPDATE users SET status = "C" WHERE age > 25 | db.users.update( { age: { $gt: 25 } }, { $set: { status: "C" } }, { multi: true }) |

MongoDB:tä ei lähtökohtaisesti suojata erillisillä käyttäjillä ja salasanoilla. Tietoturvan kannalta kuitenkin tietokannalle kannattaa asettaa käyttäjä-

varmennukset. Kuviossa 11 esitetään miten käyttäjävarmennukset asetetaan päälle.

```
mongod --auth --config /etc/mongodb/mongodb.conf
```

KUVIO 11. Vaadittavien käyttöoikeuksien lisääminen MongoDB:hen (MongoDB 2016b)

Ennen salausta kuitenkin kannattaa luoda käyttäjä, jolla on oikeudet toimia MongoDB:n sisällä. Kuviossa 12 esitetään komentotulkilla toteutettavat komennot käyttäjän lisäämiseksi. Kuviossat selkeästi käy ilmi, että kyseessä on pääkäyttäjä, kun rooliarvoksi on annettu ”userAdminAnyDatabase” ja hallittavaksi tietotokannaksi ”admin”. Yleensä sovelluksille ei anneta pääkäyttäjä-oikeuksia, elleivät käyttötarkoitukset vaadi sitä erikseen. Tällaisia poikkeuksia ovat esimerkiksi lokitiedostojen hallitsemiset.

```
use admin
db.createUser(
  {
    user: "myUserAdmin",
    pwd: "abc123",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

KUVIO 12. MongoDB:n pääkäyttäjän luonti ja lisääminen (MongoDB 2016c)

Node.js:n kannalta tarvitaan vielä erillinen laajennus, jolla saadaan erillinen rajapinta MongoDB:n käyttöä varten. Laajennuksia on kattava valikoima, mutta pääpiirteittäin kaikki toimivat samalla ajatusmallilla. Kuviossa 13 esitetään ”mongodb”-laajennuksen alustaminen ja käyttö. Kuviosta huomataan, kuinka toisella rivillä määritetään tietokannan käyttöosoite ja portti. Tämän jälkeen ilmaistaan vielä kokoelman nimi, jota halutaan käyttää. Lopuksi tarkastetaan käyttäjäoikeudet ja viimeisenä toteutetaan kysely haluttuun tauluun ja suljetaan tietokantayhteys.

```
var MongoClient = require('mongodb').MongoClient;
MongoClient.connect("mongodb://localhost:27017/testDB",
function(err, db) {
    db.authenticate(mongoUser, mongoPsw, function(err, res) {
        if(!err) {
            db.collection('testCollection').find({rID:room,
            user:userID}).toArray(function(err1, items) {
                var response = items;
                db.close();
            });
        }
    });
});
```

KUVIO 13. Node.js:n MongoDB-laajennuksen esimerkkikäyttö

# 4 PALVELUN TOTEUTUS

## 4.1 Vaatimusten määrittely

Keskustelupalvelun vaatimukset määräytyivät palvelua käyttävän sovelluksen mukaan. Työn alussa listattiin, mitä toimintoja sovellus tarvitsisi toimiakseen, ja sen pohjalta suunniteltiin palvelun toiminnallisuus sekä ohjelmarakenne.

Palvelulle olennaista on reaaliaikainen toiminta. Tämän takia kehitysalustaksi valittiin Node.js, jolla saadaan suoritettua tehokkaasti JavaScript-ohjelmistoa palvelutasolla. Palvelun tietokannalta vaadittiin mahdollisimman kevyttä ja nopeatoimista suorittamista. Tietokannaksi siis valittiin MongoDB-tietokanta, jonka käyttäminen on nopeaa BSON-kääntämisen johdosta.

Keskustelupalvelu on lähtökohtaisesti suunniteltu hybridisovellukselle, jonka tuki ulottuu Android- ja IOS-laitteille. Palvelullekin määrittyi kyseistein laitteitten tukeminen. Tämä tuki täytyi huomioida tilanteissa, joissa käyttäjä ei ole palveluun yhdistettynä.

Nimensä mukaisesti keskustelupalvelulle olennaista on keskustelumahdollisuus. Keskustelun täytyi tapahtua käyttäjien välillä siten, että kaksi käyttäjää voi luoda yksityisen keskustelun, johon muut eivät pääse. Lisäksi vaadittiin, että käyttäjät voivat luoda ja liittyä ryhmäkeskusteluihin mielensä mukaisesti.

Keskusteluja piti pystyä hallitsemaan käyttäjän toimesta. Keskusteluista täytyi pystyä poistumaan ja liittymään tarvittaessa. Lisäksi keskustelut täytyi olla poistettavissa käyttäjän keskustelulistalta.

Palvelun käynnistämisen jälkeen vaadittiin palvelulta itsenäistä toimivuutta ja ylläpitoa. Tällä tarkoitetaan, että palvelu pystyy itsenäisesti hoitamaan eritilanteita, joita käyttäjät voivat toteuttaa. Esimerkiksi viestin välityksessä pitää huomioida käyttäjän paikallaolo ja sen perusteella lähettää viesti tai ilmoitus Push-ilmoitustekniikalla.

Kehittämisen aikana käytettiin erillistä versionhallintajärjestelmää, joita käytetään nykyisin monissa ohjelmointiprojekteissa. Versionhallintana toimi Linux-pohjalle toimiva Git-versionhallintajärjestelmä. Versionhallintaa käytettiin SourceTree -nimisellä versionhallintatyökalulla, joka oli yhdistetty palvelun versionhallintaan.

Palvelusta vaadittiin toteuttaa erilliset toiminnallisuus ja asennusdokumentaatiot. Dokumentteihin vaadittiin selkeät tiedot keskustelupalvelun käyttörajapinnoista, toimintojen kuluista, toiminnallisuudesta, asetuksista ja asennusvaatimuksista.

## 4.2 Käyttäjät

Palvelun käyttäjät yksilöidään uniikeilla käyttäjätunnisteilla. Palvelu ei tarjoa käyttäjälle tunnisteen generointia, vaan käyttäjällä pitää olla tunniste tiedossa ennen palvelun käyttämistä. Mikäli käyttäjät käyttävät samaa tunnistetta, ovat he yksi ja sama käyttäjä, jolla käyttäjätiedot päivittyvät.

### 4.2.1 Käyttäjän tietojen säilöminen

Jokaista käyttäjää kohden varataan tietokannasta oma kohtansa. Kaikki käyttäjät listataan yhteen ”users”-nimiseen kokoelmaan, jossa käyttäjästä säilytetään käyttäjän tunniste, push-ilmoituksen tunniste, profiilikuvan url-osoite, käyttäjän yksityiset keskustelut, käyttäjän ryhmäkeskustelut ja keskusteluiden viestien lukemisstatukset. Kuviossa 14 esitetään MongoDB:n tallennettava dokumentti objektin käyttäjästä, jossa on edellä mainitut tiedot.

```
{
    "_id": ObjectId("16cd5833bc4c03bf6e3553fd"),
    "user": "key***********",
    "name": "Joni",
    "device": "ios",
    "token":
    "9762672ad83a6225ce4a620b51dc8808c8eaaebe37a30c924c84c9379ae0629d",
    "imgUrl": "
    http://graph.facebook.com/***********/picture?width=200&height=200",
    "privateRooms": "[]",
    "publicRooms": "[]",
    "status": "{}"
}
```


KUVIO 14. Käyttäjästä tallennettavat tiedot

Mikäli käyttäjä ei ole paikalla ja hänelle lähetetään keskusteluun viestejä, tulevat käyttäjän viestit omaan kokoelmaansa, joka on nimetty ”OfflineMsg”. Viesti tallennetaan käyttäjän tunnisteen ja keskustelutunnisteen perusteella. Viestit tallennetaan yhtenä JSON-pinona, joka on käännetty tekstimuotoon tietokannassa säilyttämistä varten. Jos käyttäjällä on ennestään viestejä keskustelussa, viestit lisätään täten viesti-pinoon ja viesti-pino päivitetään tietokantaa. Kuviossa 15 esitetään dokumenttiobjekti, joka on tallennettu ”OfflineMsg”- kokoelmaan käyttäjän poissa ollessaan palvelusta.


```
{
    "_id": ObjectId("589866dcff56bd24274a0934"),
    "rID" : "wFnc6",
    "user" : "key***********",",
    "msgs" :
    "[\"{\\\"userID\\\":\\\"key***********",\\\
    ",\\\"msg\\\":\\\"Testi: Hei!\\\"}\"]"
}
```


KUVIO 15. Käyttäjän poissa ollessa tallennettava viestidokumentti

### 4.2.2 Rekisteröinti

Ennen keskustelun aloittamista tarvitsee toteuttaa käyttäjän rekisteröinti palveluun. Käyttäjä lähettää rajapinnan vaatimat tiedot palvelimelle POST-metodia käyttämällä.


KUVIO 16. Rekisteröinnin rajapintaa

Kuviossa 16 esitetään palvelua käyttävän sovelluksen rajapinta. Palvelulle lähetetään rekisteröinnin yhteydessä käyttäjän tunniste, käyttäjänimi, laitteen push-ilmoitus-tunniste ja laitetyyppi.

Lähetettävistä tiedoista tärkeimpänä toimii käyttäjän tunniste, jonka perusteella tarkastetaan, onko käyttäjä jo entuudestaan olemassa. Mikäli käyttäjällä on jo palvelussa tietoja, käyttäjän muut lähettämät tiedot päivitetään käyttäjän tietokantaan varatulle paikalle. Näin ollen käyttäjä ei ole sidonnainen yhteen laitteeseen, vaan voi vaihtaa laitetta halutessaan.

Mikäli käyttäjä ei ole entuudestaan käyttänyt palvelua, hänelle luodaan palveluun vaadittavat dokumentit. Kuviossa 17 esitetään, kuinka uusi käyttäjä lisätään kokoelmaan.

```
451  db.collection('users').find({user:userID}).toArray(function(err1, items) {
452      if(items.length == 0){
453          var arr = [];
454          var id = userID.split("key").pop();
455          var img = "http://graph.facebook.com/"+id+
             "/picture?width=200&height=200";
456          var document = {user:userID,name:name2,device:device,token:token,imgUrl:
             img,privateRooms:JSON.stringify(arr),publicRooms:JSON.stringify(arr),
             status:JSON.stringify({})};
457          db.collection('users').insert(document, function(err, records){
458              db.close();
459          });
460          def.resolve(true);
461      }
```

KUVIO 17. Uuden käyttäjän lisääminen

### 4.2.3 Keskustelukumppanit

Keskustelun käyttäjillä voi olla kaksi tilaa, jotka ovat paikalla tai poissa. Käyttäjien tiloja tarkastellaan ja päivitetään Socket-yhteyksien mukaan.

Esimerkiksi jos käyttäjä sulkee keskustelua käyttävän sovelluksen, silloin palveluun päivittyy tieto, että käyttäjä ei ole enää paikalla.

```
1  io.sockets.on('connection', function(socket) {
2      allOnlineUsers.push(socket);
3      socket.on('myStatus', function(userID, userName){
4          socket.userID=userID;
5          socket.userName=userName;
6      });
```

KUVIO 18. Käyttäjäsocketin lisääminen palveluun

Kuvassa 18 esitetään, mitä socket-yhteyden luonnin aikana tapahtuu. Kun käyttäjä yhdistää keskustelupalveluun, palvelu lisää käyttäjän sockettiin käyttäjän tunnisteen ja nimen, jotka ovat järjestelmälle olennaisia. Nämä lisäykset esitetään kuvion 18 riveillä neljä ja viisi. Kun Socket on ”valmis”, se lisätään paikalla olevien käyttäjien listaan. Lisäys esitetään rivillä kaksi.

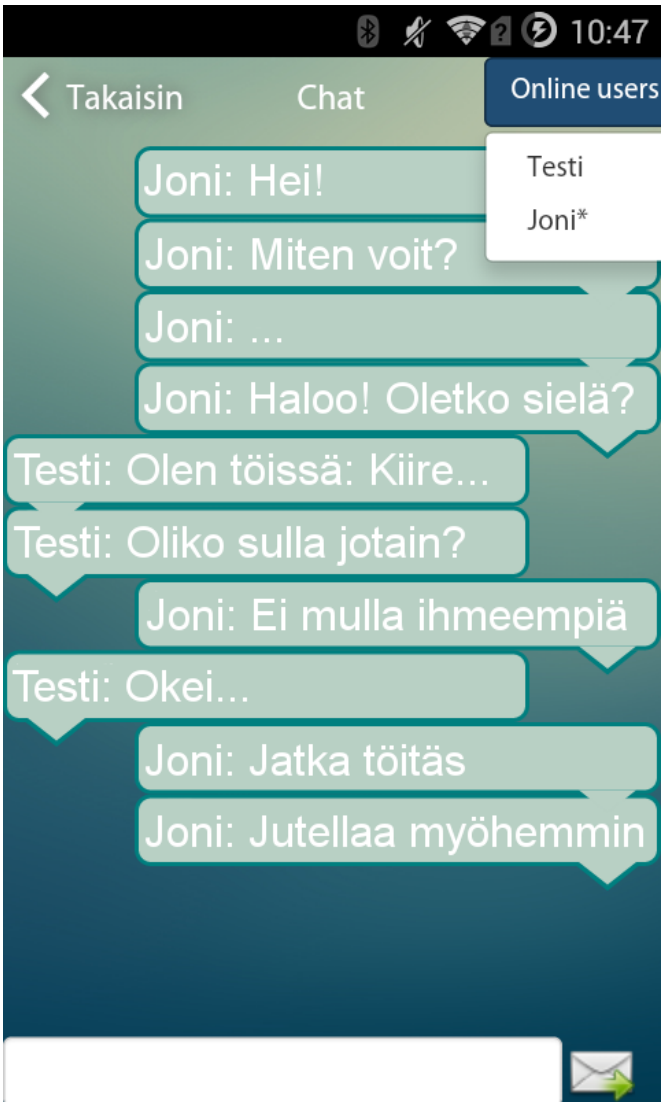
```
1  socket.on('disconnect', function() {
2      allOnlineUsers.splice(allOnlineUsers.indexOf(socket), 1);
3  });
```

KUVIO 19. Socket-yhteyden katkeaminen käyttäjän ja palvelun välillä

Kuvio 19 esittää, mitä tapahtuu, jos käyttäjän ja palvelimen välinen socket-yhteys katkeaa. Palvelu ottaa katkaisseen käyttäjän socketin tiedot ja poistaa ne paikalla olevien käyttäjien listalta. Poistaminen esitetään rivillä kaksi.

### 4.2.4 Viestit

Kaikissa keskustelupalveluun tulleissa viesteissä on sisällettyä tietoa viestin lähettäjästä, jotta viestiä voitaisiin käsitellä vastaanottavan käyttäjän puolella. Palvelun kannalta viestin sisällöllä ei ole merkitystä. Tällaisella sisälletyllä tiedolla voidaan toteuttaa esimerkiksi viestien erottelu. Kuviossa 20 esitetään sovellus, jossa viestit erotellaan niihin sisälletyn tiedon perusteella.




KUVIO 20. Viestien erottelu

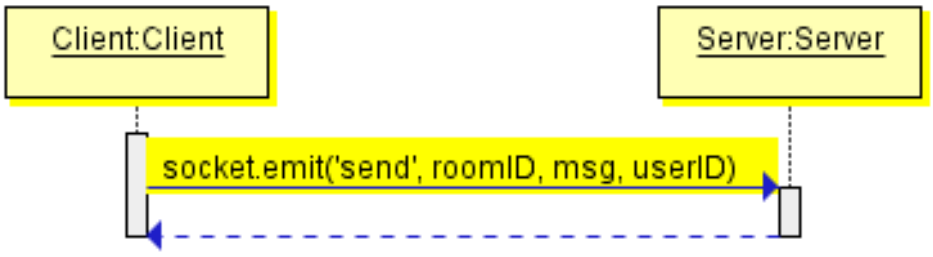
Palvelu kokoaa viestin lähettäjän tunnisteesta ja lähetetystä viestistä. Viestistä tehdään objekti, jossa kummallekin tiedolle on oma avaimensa. Lisäksi viesti muutetaan takaisin tekstimuotoon käyttäen JSON-serialisointia. Viesti täytyy olla tekstimuodossa, jotta Socketti pystyy viestin lähettämään. Mikäli viesti on muuta kuin tekstimuotoinen viesti, saattaa viestin lähetys epäonnistua ja aiheuttaa toiminnallista epävakautta. Kuviossa 21 esitetään, kuinka viestin sisältö luodaan edellä mainitulla tavalla.

```
565     msg = JSON.stringify({userID:userID,msg:msg});
```

KUVIO 21. Viestin sisällön luominen

### 4.2.5 Viestien välitys

Viestien lähettämisessä käytetään Socket.IO-kirjastoa, jolla saadaan toteutetua viestittely käyttäjien välillä. Socketissa olevalla automaattisen protokollan valinnalla saadaan toteutettua tehokas viestittely, jolla ei kuormiteta tiedonsiirtoa ylimääräisesti.




KUVIO 22. Rajapinta viestin lähetyksestä

Kuviossa 22 kuvataan viestin lähettämisen rajapinta palvelua käyttävän sovelluksen ja palvelun välillä olettaen, että sovelluksen ja palvelimen välille on muodostettu ennalta Socket-yhteys. Kuviossa esiintyvät neljä parametria, jotka luovat viestin lähetyksen. Ensimmäinen parametrina toimiva termi “send” toimii avaimena viestin lähetykseen palvelussa. Tämän parametri määrittää, mitä toiminnallisuutta palvelusta halutaan käyttää. Seuraava parametri kertoo keskustelun tunnisteen, johon viesti kuuluu. Kolmas parametri sisältää käyttäjän viestin. Viimeisenä parametrina ilmoitetaan viestin lähettäjä.

Kaikki palvelussa kulkevat viestit lähetetään saman toiminnallisuuden kautta riippumatta siitä, onko kyseessä yksityinen keskustelu vai ryhmäkeskustelu. Ennen viestien välittämistä käyttäjille tarkastellaan keskusteluihin kuuluvien käyttäjien paikalla olemista. Jos keskustelussa olevat käyttäjät ovat paikalla, ei tarvitse toteuttaa erillisiä toimintoja, kuten push-ilmoituksen lähettämistä.

```
1  service.getOfflineMsgs(room, offlinesInRoom).then(function(msgs){
2      if(msgs.length == 0){
3          msgs.push(msg);
4          logMode&&console.log("Ei ole viestejä: "+JSON.stringify(
           msgs));
5          service.createOfflineMsg(room, offlinesInRoom, msgs).
           then(function(){
6          });
7      }
8      else{
```

KUVIO 23. Poissaolevan käyttäjän viestin tallentaminen

Kuviossa 23 nähdään viestin tallentaminen kokoelmaan. Ennen tallentamista haetaan sisältö, jolla tarkastellaan, onko viestejä entuudestaan. Tämä esitetään rivillä yksi. Rivillä kaksi ja kahdeksan tehdään vertailu, jonka perusteella tarkastetaan, onko viestejä aikaisemmin talletettu. Rivillä kolme luodaan tallennettava viesti. Rivillä viisi viesti välitetään viestintallennus-toiminnalle.

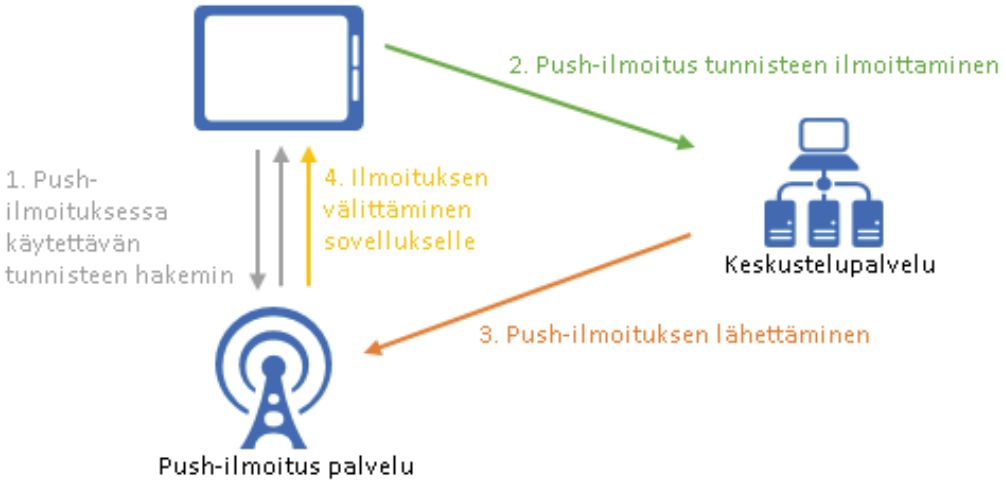
Jos käyttäjiä puuttuu keskustelusta, joudutaan viestit tallentamaan kokoelmaan ja ilmoittamaan siitä käyttäjälle push-ilmoitusta käyttämällä. Ennen käyttäjälle ilmoittamista tarkastellaan onko käyttäjällä viestejä kokoelmassa. Mikäli käyttäjällä on jo keskusteluun liittyviä viestejä, tallennetaan ne kokoelmaan, eikä hänelle lähetetä ilmoitusta uudestaan.

```
1  db.collection('users').find({user:userID}).toArray(
   function(err1, items2) {
2      if(items2.length == 0){
3          def.resolve(true);
4          db.close();
5      }
6      else{
7          var devices = [];
8          devices.push(items2[0]["token"]);
9          if(items2[0]["device"] == "Android"){
```

KUVIO 24. Käyttäjän push-ilmoitus-tunnisteen ja käyttäjän laitteen käyttöjärjestelmän hakeminen

Kun käyttäjälle lähetetään ilmoitus uudesta viestistä hänen poissa ollessa palvelusta, käyttäjän push-ilmoitus-tunniste taytyy hakea kokoelmasta. Kuvassa 24 esitetään kuinka käyttäjän push-ilmoitus tunniste haetaan käyttäjän tiedoista. Jotta viesti voitaisiin lähettää, täytyy hakea myös käyttäjän laitteen käyttöjärjestelmän tiedot. Tiedon perusteella pystytään määrittelemään, mitä push-ilmoitusmenetelmää käytetään.

Android-käyttöjärjestelmällä toimiviin laitteisiin push-ilmoitus lähettämiseen tarvitaan erillinen ”Node-gcm”-laajennus. Laajennuksen tehtävänä on välittää push-ilmoitus käyttäjän laitteelle. Kuviossa 25 esitetään, kuinka push-ilmoitus kulkeutuu käyttäjälle. Kuvion 25 vaiheessa yksi sovellus ottaa ensimmäisenä yhteyden push-ilmoitus-palveluun ja saa vastaukseksi push-ilmoitus tunnisteensa. Vaiheessa kaksi sovellus ilmoittaa keskustelupalvelulle oman push-ilmoitus-tunnisteensa. Vaiheessa kolme lähetetään ilmoitus uudesta viestistä push-ilmoitus-palveluun ja kohdassa neljä push-ilmoitus-palvelu välittää uudesta viestistä push-ilmoituksen sovellukselle.



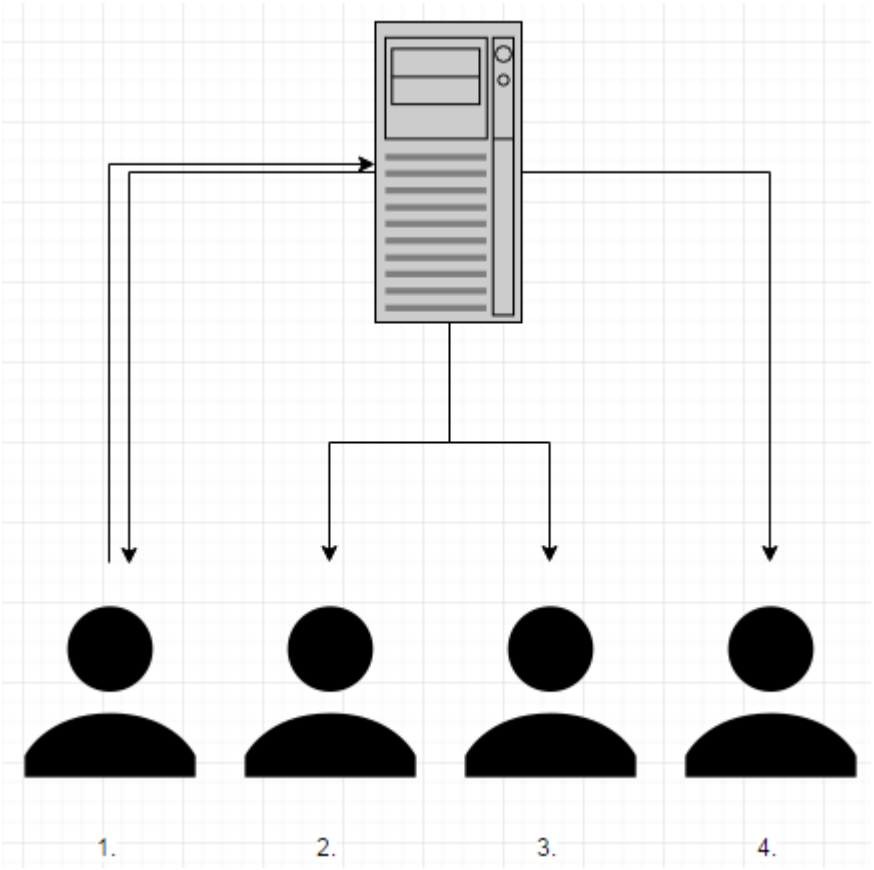

KUVIO 25. Push-lähetys-prosessi (Microsoft 2016)

Applen iOS-laitteiden push-ilmoitusten lähettämiseen tarvitaan omanlainen laajennus. Laajennuksena työssä käytetään ”APN”-nimistä laajennusta. Itsessään ilmoituksen lähettäminen ei eroa Androidin ilmoituksen lähettämisestä. Kuviossa 25 on toimiva esimerkki myös Applen iOS-laitteiden push-ilmoitusten lähettämisestä.

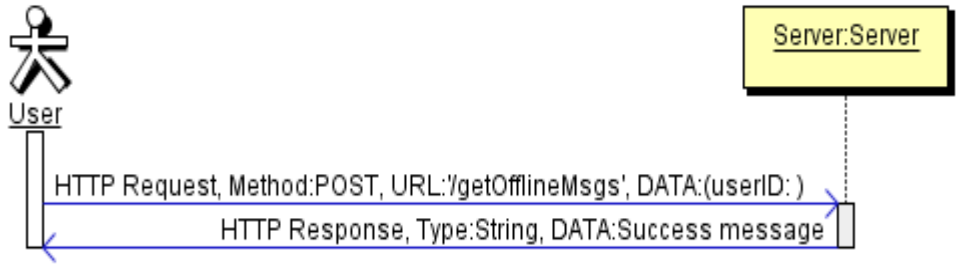
```
629     io.to(room).emit('listenRoom',room, msg);
```

KUVIO 26. Viestin lähettäminen

Viestin lähettäminen tapahtuu keskustelussa vaikka käyttäjät eivät olisi paikalla. Kuviossa 26 on yksinkertainen esimerkki viestin lähettämisestä. Viesti lähtee kaikille keskusteluun kuuluville socketeille, eli käyttäjille. Näihin käyttäjiin lukeutuu myös viestin lähettäjä. Kuviossa 27 esitetään kyseinen lähetysprosessi. Käyttäjä yksi lähettää viestin palveluun ja palvelu välittää viestin kaikille käyttäjille.




KUVIO 27. Viestin välittyminen käyttäjille




KUVIO 28. Poissaolleen käyttäjän viestien pyytäminen

Käyttäjien viestit, jotka ovat tulleet käyttäjien poissa ollessa, voidaan pyytää välitetyksi erillisellä POST- kutsulla. Kuviossa 28 esitetään poissa olleen käyttäjän kutsuntarajapinta viestien hakemiseen. POST:n palautus ei palauta viestejä, vaan POST suorittaa toiminnallisuudet, jotka hakevat viestit ja välittävät ne socketin kautta käyttäjälle. Kuviossa 29 esitetään yksityisten keskusteluiden hakeminen edellä mainitulla POST-kutsulla. Riviltä 8 esitetään viestien lähettäminen socketin kautta.

```
1  var msgLoop = setInterval(function(){
2      useMongo.getOfflineMsgs(rooms["privateRooms"][i], userID).then(
       function(msgs){
3          if(msgs.length<=i){
4              clearInterval(msgLoop);
5          }
6          else if(msgs.length> 0){
7              if(msgs.toString() != oldMsgs){
8                  io.sockets.emit(userID, rooms["privateRooms"][i],
                   "", JSON.stringify(msgs), "");
9                  oldMsgs = msgs.toString();
10             }
11
12         }
13     }).then(function(){
14         useMongo.clearOfflineMsg(rooms["privateRooms"][i], userID);
15         i++;
16     });
17
18 },400);
```

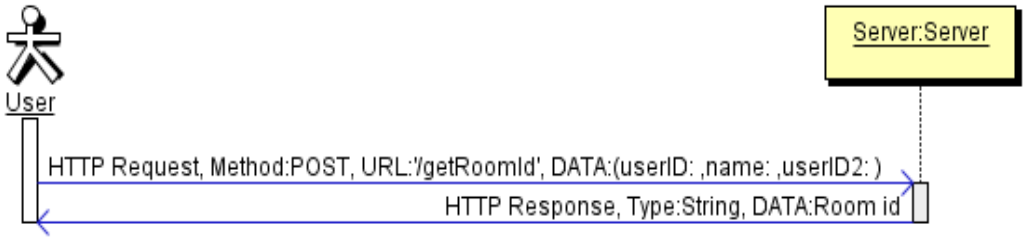
KUVIO 29. Yksityisten keskusteluitten hakeminen ”getOfflineMsgs” POST-kutsulla

## 4.3 Keskustelut

Käyttäjien välinen keskustelu tapahtuu aina palvelun kautta. Kun käyttäjä haluaa lähettää viestin toiselle käyttäjälle, viesti tarvitsee ensin välittää palvelulle, joka osaa sen välittää sitten vastaanottajilleen. Myös viestin lähettäjä on itsessään viestin vastaanottaja. Lähettäjän on hyvä olla itsekin vastaanottajana, jotta keskustelussa saadaan karsittua viestien synkronointiongelmat, esimerkiksi jos viesti ei lähdekään muille keskustelussa oleville käyttäjille mutta päivittyy viestin lähettäjälle.

### 4.3.1 Yksityiset keskustelut

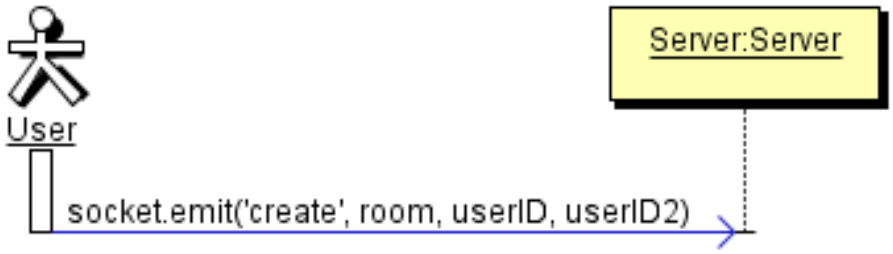
Yksityisen keskustelun aloittamisessa tarvitaan aloittavan käyttäjän ja vastapuolen käyttäjän tunniste. Aloittava osapuoli ”ilmoittaa” palvelulle haluavansa aloittaa keskustelun näiden kahden käyttäjä tunnisteen välillä. Palvelu tarkistaa yksityisistä keskusteluista, että käyttäjillä ei ole ennestään keskustelua aloitettuna.




KUVIO 30. Kahden käyttäjän keskustelun tunnisteen hakeminen

Mikäli käyttäjillä ei ole ennestään aloitettuna keskustelua, palvelu luo uniikin keskustelutunnisteen ja tallentaa palveluun kyseisten käyttäjien välisen keskustelun. Keskustelusta lisätään myös tieto keskusteluun osallistuvien tietoihin. Lisäksi keskustelun aloituksesta palautetaan aloittavalle käyttäjälle luodun keskustelun tunniste ja vastapuolen käyttäjälle ilmoitus, että uusi keskustelu hänen kanssaan on aloitettuna. Kuviossa 30 on esitetty keskustelun tunnisteen luonti ja tarkastelu -rajapinta.

Jos käyttäjillä on ennestään keskustelu aloitettuna, palvelun ei tarvitse muuta, kuin palauttaa keskustelun aloittavalle keskustelun tunniste ja tarvittaessa vastapuolen käyttäjälle ilmoittaa uudesta keskustelusta.




KUVIO 31. Yksityiseen keskusteluun liittyminen

Kun käyttäjä on saanut keskustelun luoduksi ja palvelu on palauttanut käyttäjälle keskustelun tunnisteen, käyttäjä pystyy liittymään keskusteluun tunnisteen perusteella. Kuviossa 31 nähdään, kuinka keskusteluun liitytään keskustelun tunnisteen perusteella. Kuvio 32 on kuvaus siitä, miten socketti liitetään keskusteluun kyselyn mukana tulleen tunniste eli keskustelun tunnisteen perustella.

```
1  socket.on('create', function(room, userID, userID2) {
2      socket.join(room);
3  });
```

KUVIO 32. Keskusteluun liittymistoiminto

Yksityisestä keskustelusta poistuminen ei varsinaisesti tapahdu kuin muodollisesti. Kun kyseessä on käyttäjäkohtaiset tunnisteet, jotka eivät vaihdu, on turhaa myöskään poistaa yksityisiä keskusteluja. Palvelun kannalta on nopeampaa käyttää jo ennalta luotuja keskusteluja, kuin luoda uusia. Keskustelusta poistuminen on siis käyttäjäkohtainen muutos, jolla saadaan karsittua vain halutut yksityiset keskustelut käyttäjän keskustelulistaan.

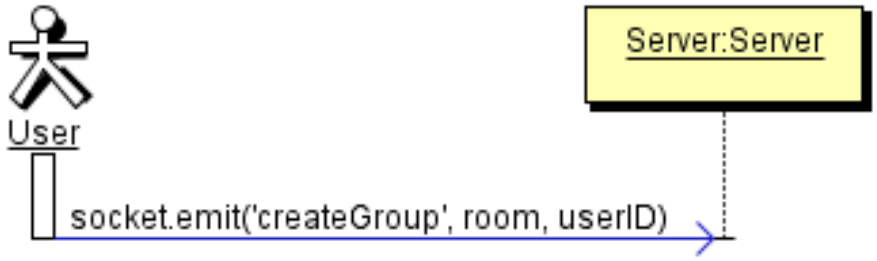
### 4.3.2 Ryhmäkeskustelut

Ryhmäkeskustelut toimivat melkein samalla periaatteella, kuin yksityisetkin keskustelut. Keskusteluissa käytetään samaan tapaan tunnistetta, jonka perusteella käyttäjät yhdistyvät toisiinsa.

```
socket.on('createGroup', function(room, userID) {
    socket.join(room);
    useMongo.insertUserRooms(userID, room, 1);
    useMongo.addPublicRoom(room, null, userID);
});
```

KUVIO 33. Ryhmäkeskusteluun liittyminen

Keskustelun tunnisteen luontivastuu on jätetty käyttäjän vastuulle. Keskustelun tunniste rakentuu käyttäjän tunnisteen ja keskustelun nimen mukaan. Ajatuksena tässä on, että käyttäjän tunniste on yksilöllinen ja keskustelun nimikin on oletetusti uniikki. Kuviossa 33 nähdään, kuinka ryhmäkeskustelu luodaan ja siihen liitytään. Toiminnossa huomataan “insetUserRooms”- ja “addPublicRoom”-toiminnot, joilla lisätään käyttäjän ja ryhmäkeskustelujen tietoihin tieto uudesta keskustelusta.




KUVIO 34. Ryhmäkeskusteluun liittyminen

Muiden käyttäjien liittyminen keskusteluun ei aiheuta suurempia toimenpiteitä. Käyttäjä ilmoittaa palvelulle, että haluaa liittyä keskusteluun keskustelutunnisteen perusteella. Lisäksi liittyessä ilmoitetaan oma tunniste, joka lisätään ryhmäkeskustelun tietoihin. Kuviossa 34 nähdään käyttäjän liittymisilmoitus ja käyttäjän tietojen lisääminen ryhmäkeskustelun tietoihin.

Ryhmäkeskustelusta poistuttaessa käyttäjältä poistetaan ryhmäkeskustelun tiedot, kuten yksityisestä keskustelusta poistuttaessa. Lisäksi ryhmäkeskustelun tiedoista poistetaan käyttäjän tiedot.

```
1  var name2 = items[0]["name"];
2  var users = JSON.parse(items[0]["users"]);
3  users.splice(users.indexOf(userID), 1);
4  if(users.length == 0){
5      db.collection('publicRooms').remove({room:room});
6      db.close();
7      }
8  else{
9      var document = {room:room,name:name2,users:JSON.stringify(users)};
10     db.collection('publicRooms').update({room:room}, document,
       function(err, records){
11         db.close();
12     });
13 }
```

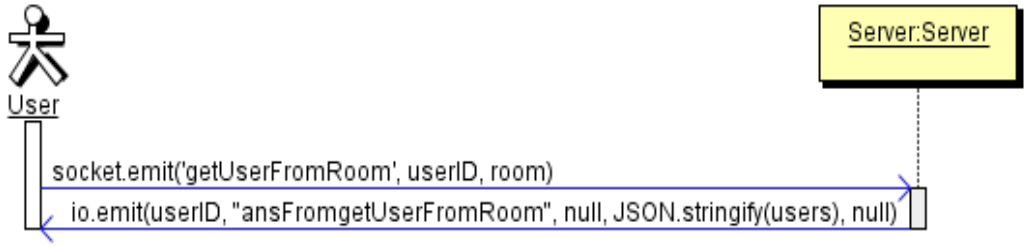
KUVIO 35. Viimeisen käyttäjän poistuminen

Jos ryhmäkeskustelusta kaikki käyttäjät poistuvat, keskustelu poistetaan kokonaan palvelun tiedoista. Kuviossa 35 käydään lävitse viimeisen keskusteluun kuuluvan käyttäjän poistuminen ja tämän jälkeen keskustelun poistaminen tiedoista. Rivillä viisi poistetaan keskustelu palvelun tiedoista, kun viimeinen käyttäjäkin on poistunut.

### 4.3.3 Keskustelukumppaneitten tilat

Kummassakin keskustelumuodossa on tarvittaessa käytettävissä keskusteluun kuuluvien käyttäjien tilojen tarkastelu. Tarkastelussa nähdään, ketkä käyttäjät kuuluvat keskusteluun ja ketkä heistä ovat paikalla.

Palvelussa käytetyssä socketissa ei pystytä käyttämään ajatusmallia lähetys ja vastaus. Sockettia käyttämällä täytyy aina luoda lähettävä osapuoli ja vastaanottava puoli. Jotta keskusteluun kuuluvien käyttäjien tiedot saadaan, tarvitaan lähettää kysely käyttäjistä keskustelun tunnisteen perusteella. Kuviossa 36 esitetään, kuinka kysely toteutuu.




KUVIO 36. Keskusteluun kuuluvien käyttäjien hakeminen

Kun edellä mainittu kysely havaitaan palvelussa, palvelu toteuttaa tarkistuksen, jolla tarkastetaan, ketkä kaikki käyttäjät kuuluvat keskusteluun. Tämän lisäksi palvelu tarkistaa keskusteluun kuuluvien henkilöitten paikallaolotilan kaikista paikalla olevista käyttäjistä. Tarkastusten päätteeksi palvelu palauttaa käyttäjät, jotka kuuluvat keskusteluun, ja merkitsee käyttäjät, jotka ovat paikalla.

# 5 PALVELUN YLLÄPITO

Ennen keskustelupalvelun käyttöönottoa täytyy huomioida muutamia asioita, jotka vaikuttavat palveluun. Jos näitä asioita ei oteta huomioon, saattaa palvelun ajaminen pysähtyä ennenaikaisesti tai sekoittua muuhun palveluun, esimerkiksi samalla portilla pyörivät palvelut saattavat vaikuttaa toisiinsa negatiivisesti.

## 5.1 Vaadittavat laajennukset

Palvelu tarvitsee toimiakseen tietyt laajennukset NPM-pakettienhallinnasta. Kaikki laajennukset yhtä lukuun ottamatta ovat projekti sidonnaisia. Taulukossa 5 on esitettynä palveluun vaadittavat laajennukset; poikkeuslaajennuksena huomataan ”Forever”-laajennus, joka täytyy asentaa julkisesti Node.js:n käyttöön.

TAULUKKO 5. Palvelun vaadittavat NPM-laajennukset ja niiden selitys

| | |
|---|---|
| express@4.10.2 | Framework |
| socket.io | Socket.IO kirjasto |
| body-parser | POST- ja GET- parametrien parsimistyökalu |
| mysql | Mysql laajennus |
| q | Promise kirjasto |
| mongodb | MongoDB laajennus |
| node-gcm | Android push-ilmoitus laajennus |
| apn | Apple push-ilmoitus laajennus |

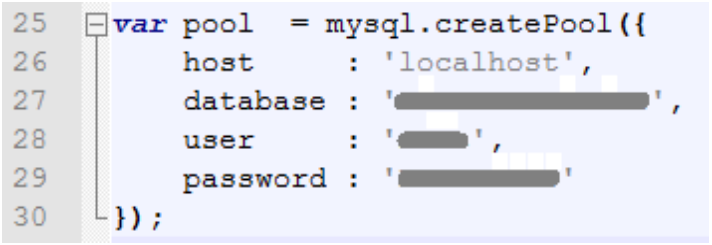
(jatkuu)

TAULUKKO 5. (jatkuu)

| | |
|---|---|
| forever –g | Node.JS itsenäinen JavaScript suoritin |
| mongo-log-rotator | MongoDB:n lokitiedoston jaottelija |

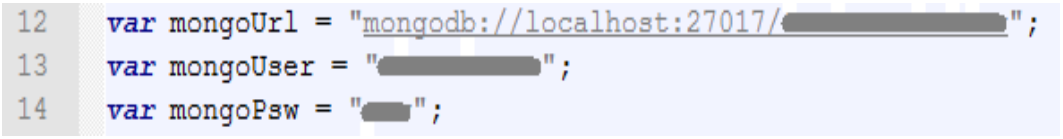
## 5.2 Palvelun asetukset

Palveluun pystytään tuomaan vanhoja käyttäjiä MySQL-tietokannasta. Tätä varten palveluun täytyy lisätä haettavan tietokannan tiedot. Kuviossa 37 esitetään tietokantojen tietojen alustaminen.




KUVIO 37. MySQL-tietokannan alustaminen

Kun palvelun oletustietokantana toimii MongoDB, tarvitsee MongoDB:n asetukset määritellä. Kuviossa 38 esitetään tietokannan tietojen alustaminen.




KUVIO 38. MongoDB-tietokannan alustaminen

MongoDB luo itsenäisesti lokia palveluympäristöön. Tämä loki saattaa joissain tapauksissa kasvaa liian suureksi ja täyttää turhaan palvelimen resursseja. Tätä varten on asennettu laajennus ”Mongo-Log-Rotator”, jolla pystytään kontrolloimaan lokitiedoston kokoa. Kuviossa 39 esitetään Mongo-Log-Rotatir:n asetusten alustaminen. Asetusta tehdessä täytyy huomi-

oida, että asetettavilla käyttäjätiedoilla on riittävät oikeudet, verrattuna kuvion 38 rivin 13 MongoDB-käyttäjään. Riittävillä oikeuksilla tarkoitetaan MongoDB-hallinnointia pääkäyttäjän oikeuksilla. Mikäli näitä oikeuksia alustettavalla käyttäjälle ei ole, ei palvelu pysty kontrolloimaan lokitiedostoa, koska lokitiedostojen luonti ja käsittely tapahtuu pääkäyttäjän toimesta.


KUVIO 39. Mongo-Log-Rotatorinasetukset

Push-ilmoituksia varten täytyy tehdä alustuksia. Palvelussa tarjotaan kahdelle laitteelle push-ilmoitus-palveluita. Ilman näitä asetuksia palvelu ei pysty lähettämään push-ilmoituksia.

Android-käyttöjärjestelmille tarkoitettuja push-ilmoituksia varten käytetään Node-GCM laajennusta. Node-gcm tarvitsee toimiakseen ”api-keyn”, joka saadaan Googlelta. Kuviossa 40 esitetään ”api-keyn” asettaminen.


KUVIO 40. Node-GCM:n api-keyn asettaminen

Apple iOS-käyttöjärjestelmille tarkoitettu push-ilmoituksia varten käytetään laajennusta ”APN”. Verrattuna Androidin push-ilmoitus-palveluun, Applen push-ilmoitus-palvelussa tarvitaan erilaisia tietoja, joilla lähettäminen tapahtuu. Applen käyttöjärjestelmille tarvitaan erilliset sertifikaatti- ja avaintiedostot. Kuviossa 41 nähdään tiedostojen asettaminen palveluun.

```
var servicePush = new apn.Connection({
    cert:"pem/cert.pem",
    key:"pem/key.pem",
    production: false,
    errorCallback: pushCallbackError,
    batchFeedback  : true,
    interval       : 300,
    maxConnections : 5
});
```

KUVIO 41. APN:n tiedostojen määrittely

Palvelun suorittamista voidaan tarkkailla erillisellä boolean tyyppisellä muuttujalla. Mikäli tarkkailu muuttujan arvo muokataan arvoon tosi, alkaa palvelu syöttämään toimintoja komentotulkille. Lähtökohtaisesti arvo on epätosi ja palvelu ei syötä komentotulkille mitään toimintoja. Kuviossa 42 esitetään boolean muuttuja, jota muuttamalla saadaan tulkkaaminen päälle.

```
var logMode = false;
```

KUVIO 42. Komentotulkkauksen määrittely

Keskustelupalvelun kannalta tärkein määrittely on palvelun portin asetus. Portin kautta tapahtuu kaikki liikenne palvelun ja käyttäjän välillä. Kuviossa 43 nähdään portin määrittely, joka on kuvassa 4001.

```
http.listen(4001, function(){
    // server ready
});
```

KUVIO 43. Portin määrittely

## 5.3 Palvelun käyttäminen

Kun palvelulle vaadittavat laajennukset ja asetukset on tehty, palvelu voidaan käynnistään. Palvelua voidaan käynnistää kahdella eri tapaa. Voidaan käyttää Node.js:n perinteistä suorituskomentoa, joka näkyy kuviossa

44. Komento käynnistää palvelun kyseisen komentorivi-istuntoon. Tätä tapaa on hyvä käyttää, mikäli palvelusta halutaan saada tulkintaa toiminnollisuuksien suorittamisesta. Pitkäaikaiseen ajoon tämä komento ei kuitenkaan sovellu. Mikäli komentoikkunan istunto päättyy, päättyy myös palvelun suoritus. Palvelu päättyy myös, mikäli palvelu päätetään komentotulkilla.

```
node index2_dev.js
```

KUVIO 44. Node.Js:n suorituskomento

Palvelun tarkoituksena on kuitenkin toimia jatkuvasti. Tätä varten palveluun on asennettu ”Forever”-laajennus, jolla pystytään pitämään palvelua tausta-ajossa. Kuviossa 45 esitetään palvelun käynnistäminen tausta-ajoon laajennuksen avulla.

```
forever start index2_dev.js
```

KUVIO 45. Palvelun käynnistäminen tausta-ajoon

Taustalla ajettavien palveluiden määrää voidaan tarkastella samalla laajennuksella. Laajennus listaa kaikki taustalla olevat Node.js-palvelut. Kuviossa 46 esitetään komento, jolla saadaan lista taustalla olevista palveluista.

```
$ forever list
info: Forever processes running
data: uid command scriptsforever pid id logfile uptime
data: [0] tfwI /usr/bin/nodejs index2_dev.js 44333
44335 /home/user/.forever/tfwI.log 0:0:3:15.687
```

KUVIO 46. Taustalla ajettavien palveluiden listaus

Palvelun sammuttaminen tapahtuu ”Forever”-laajennuksen omalla komennolla. Kuviossa 47 esitetään komento, jolla palvelu sammutetaan. Palvelu poistuu tausta-ajosta, minkä vuoksi kaikki palvelua käyttävät käyttäjät saavat yhteys virheen palvelua käyttäessään.

```
forever stop index2_dev.js
```

KUVIO 47. Palvelun sammuttaminen tausta-ajosta

# 6 YHTEENVETO

Opinnäytetyön tavoitteena oli toteuttaa hybridisovelluksille keskustelupalvelu, jossa käyttäjät voivat keskustella keskenään. Keskustelut muodostuvat yksityisistä keskusteluista, joissa kahden käyttäjän keskustelut ovat yksityisiä ja ryhmäkeskusteluista, joissa kaikki käyttäjät voivat keskustella keskenään. Käyttäjien piti pystyä hallitsemaan omia keskusteluja haluamallaan tavallaan. Palvelulta odotettiin itsenäistä toimintaa käynnistämisen jälkeen.

Työn teoriaosuudessa käytiin läpi teknologioita, joita palvelussa käytettiin. Palvelussa käytettiin uudenaikaisia teknologioita, joilla saadaan palvelulle toteutettua mahdollisimman tehokas ja viiveetön toiminnallisuus. Teoriaosuudessa selvitettiin ja havainnollistettiin tekniikoita esimerkkien avulla.

Käytännön osuudessa käytiin läpi keskustelupalvelun rakennetta, toiminnallisuutta ja käyttäjän ja palvelun rajapintoja. Lopuksi teoriaosuudessa käytiin läpi palvelun alustamista käyttöä varten ja sen käyttämistä.

Opinnäytetyö tuotti vaadittavan palvelun hybridisovelluksille. Palvelu on tällä hetkellä liitettynä palvelua vaativaan hybridisovellukseen ja on testausta vaille. Tulevaisuudessa palvelu olisi tarkoitus ottaa pysyvästi käyttöön hybridisovellukselle.

LÄHTEET

AbraEnter. 2013. HTTP Long-Poll [viitattu 21.4.2016]. Saatavissa: http://www.abrandao.com/2013/05/php-http-long-poll-server-push/

Cheng, C. 2013. What is Socket.IO [viivattu 18.3.2016]. Saatavissa: http://learn-gevent-socketio.readthedocs.org/en/latest/socketio.html

Kaazing. 2014. What is WebSocket [viitattu 21.4.2016]. Saatavissa: http://kaazing.com/websocket/

Laurens, T. 2016 How the V8 engine works [viivattu 18.3.2016]. Saatavissa: http://thibaultlaurens.github.io/javascript/2013/04/29/how-the-v8-engine-works/

Microsoft. 2016. Notification Hubs Overview [viivattu 18.3.2016]. Saatavissa: https://msdn.microsoft.com/en-us/library/azure/jj927170.aspx

MongoDB. 2016a. Enable Authentication after Creating the User Administrator [viivattu 18.3.2016]. Saatavissa: https://docs.mongodb.org/v2.6/tutorial/enable-authentication-without-bypass/

MongoDB. 2016b. Enable Client Access Cotrol [viivattu 18.3.2016]. Saatavissa: https://docs.mongodb.org/manual/tutorial/enable-authentication/

MondoDB. 2016c. JSON and BSON [viivattu 18.3.2016]. Saatavissa: https://www.mongodb.com/json-and-bson

MongoDB. 2016d. MongoDB and MySQL Compared [viivattu 22.4.2016]. Saatavissa: https://www.mongodb.com/compare/mongodb-mysql

Mozzilla Developer Network. 2016. Promise [viivattu 18.3.2016]. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

MySQL. 2016. What is MySQL [viivattu 22.4.2016]. Saatavissa: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html

Nallan, K. 2013. Promises – an alternative way to approach asynchronous JavaScript [viivattu 18.3.2016]. Saatavissa: http://12devs.co.uk/articles/promises-an-alternative-way-to-approach-asynchronous-javascript/

NixCraft. 2010. PHP Increase Upload File Size Limit [viitattu 23.4.2016]. Saatavissa: http://www.cyberciti.biz/faq/linux-unix-apache-increase-php-upload-limit/

RisingStack Engineering. 2016. History of Node.js [viitattu 18.3.2016]. Saatavissa: https://blog.risingstack.com/history-of-node-js/

W3Schools. 2016. HTTP Methods: GET vs POST [viivattu 23.4.2016]. Saatavissa: http://www.w3schools.com/tags/ref_httpmethods.asp

Walsh, D. 2016. JavaScript Promise API [viivattu 18.3.2016]. Saatavissa: https://davidwalsh.name/promises

Webopedia. 2016. Asyncronous [viivattu 18.3.2016]. Saatavissa: http://www.webopedia.com/TERM/A/asynchronous.html

Win8. 2016. Asynkroninen ohjelmointi [viivattu 18.3.2016]. Saatavissa: http://win8.fi/materiaali/8-edistyneita-aiheita