Jani Skarp

Node.js ja reaaliaikainen viestintä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

1.12.2015

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Jani Skarp Node.js ja reaaliaikainen viestintä 29 sivua + 2 liitettä 1.12.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Simo Silander Yliopettaja Erja Nikunen

Insinöörityön tavoitteena oli luoda node.js-palvelinkehystä hyödyntävä reaaliaikainen keskusteluohjelma. Tavoitteena oli hyödyntää node.js:n nopeutta, mikä mahdollistaa reaaliaikaisten nettisovellusten luomisen. Tarkoituksena oli myös selvittää reaaliaikaisen viestinnän historiaa ja tarpeellisuutta ja miksi node.js soveltuu juuri reaaliaikaisten sovellusten tekemiseen.

Luotu chat-ohjelma käyttää paljon hyödyksi node.js:n socket.io-moduulia. Socket.io antaa helpon tavan kommunikoida palvelimen ja selaimen välillä ja ohjelmakoodi on yksinkertaista. Työn tuloksena nähdään, miten yksinkertaista reaaliaikainen viestintä voi olla, kun käytetään node.js:ää ja sen eri moduuleja.

Avainsanat	Node.js, reaaliaikainen

Author(s) Title Number of Pages Date	Jani Skarp Node.js and Real-Time Communication 29 pages + 2 appendices 1 December 2015
Degree	Bachelor of Engineering
Degree Program	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Senior Lecturer Erja Nikunen, Principal Lecturer

The goal of this thesis was to create a real-time communication web application that uses node.js. Another goal was to find out about the history of real-time communication and why using node.js is a good choice at a real-time web application

The real-time chat application created in this thesis uses heavily node.js module socket.io. Socket.io allows for an easy way to communicate between server and browser. It also allows creation of a simple code. Thesis shows how easy and simple making a real-time chat application can be while using node.js

Keywords	Node.js, real-time

Sisällys

1 Johdanto 1

2 Muita web-palvelinteknologioita 2

2.1 Staattinen nettisivu 2

2.2 Dynaamiset nettisivut 3

2.2.1 Active Server Pages 4

2.2.2 ColdFusion 5

2.3 Ajax 6

3 Perustietoa 7

3.1 Node.JS:n historia 7

3.2 Node.js lyhyesti 8

3.3 Node Package Manager 10

3.4 Soketit ja Web-soketit 10

3.5 Express.js 13

4 Node.js 13

4.1 Node.js perusrakenne 13

4.2 Chrome V8 -moottori 15

4.3 Event Loop 17

5 Node.js-pohjainen Chat-ohjelma 17

5.1 Chat-ohjelmien historia ja käyttö nykypäivänä 17

5.2 Toimintaympäristö 19

5.3 Sisäinen arkkitehtuuri 19

5.4 Moduulien sisäinen toiminta 19

5.5 Käyttöohje 25

6 Yhteenveto 27

Lähteet 29

Liitteet

Liite 1. Keskusteluohjelman koodi (selain)

Liite 2. Keskusteluohjelman koodi (palvelin)

1 Johdanto

Suuri osa web-applikaatioissa sisältää sekä selain- ja palvelinpuolen. Palvelinpuolen tekeminen on ollut vaikeaa ja työlästä, koska se vaati paljon tietoa rinnakkaisista säikeistä, skaalautumisesta ja palvelin käyttöönotosta. Toisena ongelma oli, että selainpuoli koodattiin usein HTML:llä ja JavaScriptillä, kun taas palvelimessa käytettiin staattisempaa koodikieltä. Tämä pakotti ohjelmoinnista vastaavan käyttämään monia ohjelmointikieliä ja päättämään aikaisessa vaiheessa, missä jotkin koodilogiikat sijaitsivat.

Tämä alkoi muuttua, kun V8 JavaScript-moottori tuli Google Chromen kanssa ja antoi ratkaisut kahteen ongelmaa, jotka estivät JavaScriptin käytön palvelinpuolella. Nämä ongelmat olivat huono ajonaikainen suorituskyky ja lelumainen muistinkäsittely. Jäljelle jäi vai kolmas este eli käyttöjärjestelmän integraation puute. Ryan Dahl näki tässä mahdollisuuden tuoda JavaScriptin palvelinpuolelle yhdistämällä V8:sin OS integraatio tasoon, joka sisälsi asynkronisia rajapintoja perustana olevaan käyttöjärjestelmään. Tämä johti Node.JS:n syntymiseen.

Node.js tuli heti suosituksi ja sille syntyi energinen avoin lähdekoodiyhteisö, avustavia yrityksiä ja oma konferenssi. Suosio syntyi, koska Node.js mahdollisti saman ohjelmointikielen käyttämisen sekä palvelin- että selainpuolella. JavaScriptin käyttäminen teki helpoksi kehityksen ja kokeilun palvelin koodissa, joka vapautti koodaajat hitaasta työkaluohjelmointimallista. (1, s. 13.)

Tässä työssä tehty chat-ohjelma on tehty käyttäen node.js kehystä, joka mahdollistaa palvelin- ja selainpuolen koodauksen Javascriptillä. Työn tarkoituksena on tutkia, kuinka yksinkertaisesti voi chat-ohjelman toteuttaa käyttämällä node.js:ää ja sen moduuleja. Socket.io moduulilla on työssä iso rooli. Se mahdollistaa yksinkertaisen ja hyvin yhteensopivan socket-teknologian käytön.

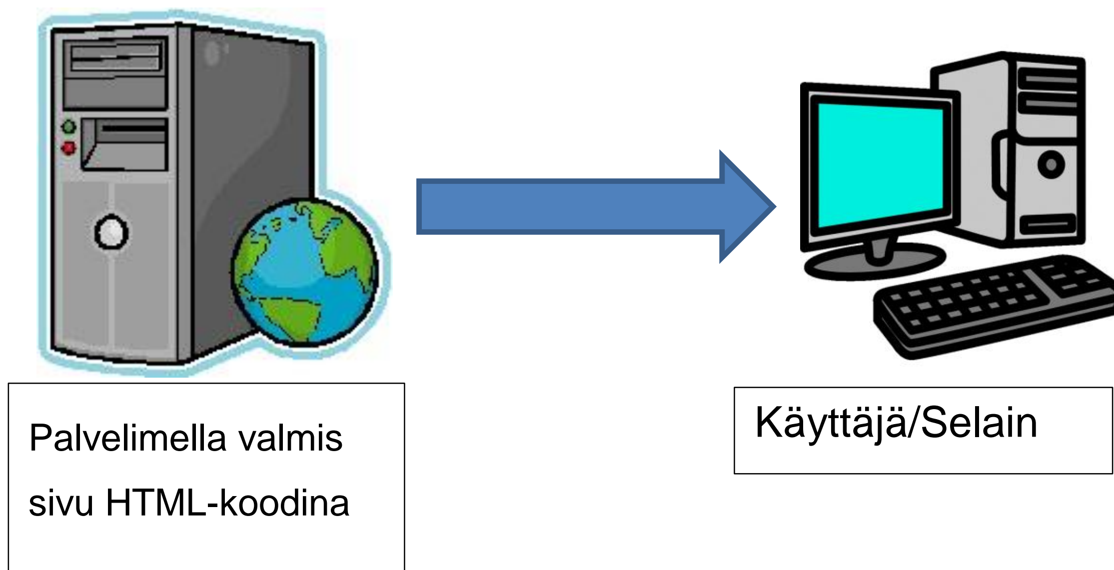
2 Web-palvelinteknologioita

Jo ennen modernin internetin syntyä oli jo joitakin palvelinselainohjelmia. Silloin ohjelmissa oli palvelinkoodi ja jokaiseen käyttäjä pisteeseen asennettu käyttäjäkoodi. Ongelmana oli, että käyttäjäkoodi piti asentaa ja päivittää paikanpäällä, jonka takia tukitoiminnat olivat kalliita. Ohjelmat toimivat myös usein vain tietyissä tietokonearkkitehtuureissa ja käyttöjärjestelmissä, minkä takia vain suurien ohjelmien tekeminen oli kannattavaa.

Vastauksena syntyivät web-ohjelmat, jotka pystyttiin suoraan lataamaan käyttäjien koneisiin silloin, kun he käyvät ohjelmaa vastaavalla nettisivulla. Web-ohjelmat perustuivat web-dokumentteihin, jotka kirjoitettiin standardisoidulla formaatilla, kuten HTML:llä tai JavaScriptillä.

2.1 Staattinen nettisivu

Staattiset nettisivut ovat yksinkertaisimpia ja helpoimmin tehtäviä nettisivuja. Ne on täysin koodattu HTML:llä ja niissä näkyy sama tieto kaikille käyttäjille. Yhtään nettikoodausta tai tietokantasuunnittelua ei tarvita, sillä sivujen tekemiseen tarvitaan vain HTML-sivuja, jotka on julkaistu netti palvelimelle. Toiminta periaate näkyy kuvassa 1.



Kuva 1. Staattisen nettisivun toimintaperiaate

Koska staattiset nettisivut on tehty käyttäen kiinteää koodia, ne eivät muutu ellei niistä vastaava muuta niiden koodia. Tämä toimii hyvin pienissä sivustoissa hyvin. Ongelmia syntyy kuitenkin suurien sivustojen ylläpidossa, sillä niissä voi olla tuhansia sivuja. Tämän takia suurissa sivustoissa käytetään usein valmiita pohjia, minkä takia voidaan päivittää useita sivuja samanaikaisesti ja niillä on yhtenäinen ulkonäkö. (2.)

2.2 Dynaamiset nettisivut

Dynaamiset nettisivut ovat HTML:llä koodattuja nettisivuja, mutta niihin on lisätty netti-scripting-koodia, jolla niistä on tullut dynaamisia. Nämä nettisivut luodaan reaaliajassa eli aina kun nettisivu avataan selaimessa. Käyttämällä ohjelmakoodia, jonka pohjalta sivun osia luodaan. Näitä koodausvälineitä ovat esimerkiksi ASP tai ColdFusion. Kun koodit on käyty läpi, saa käyttäjä reaaliaikaisen näkymän nettisivusta.

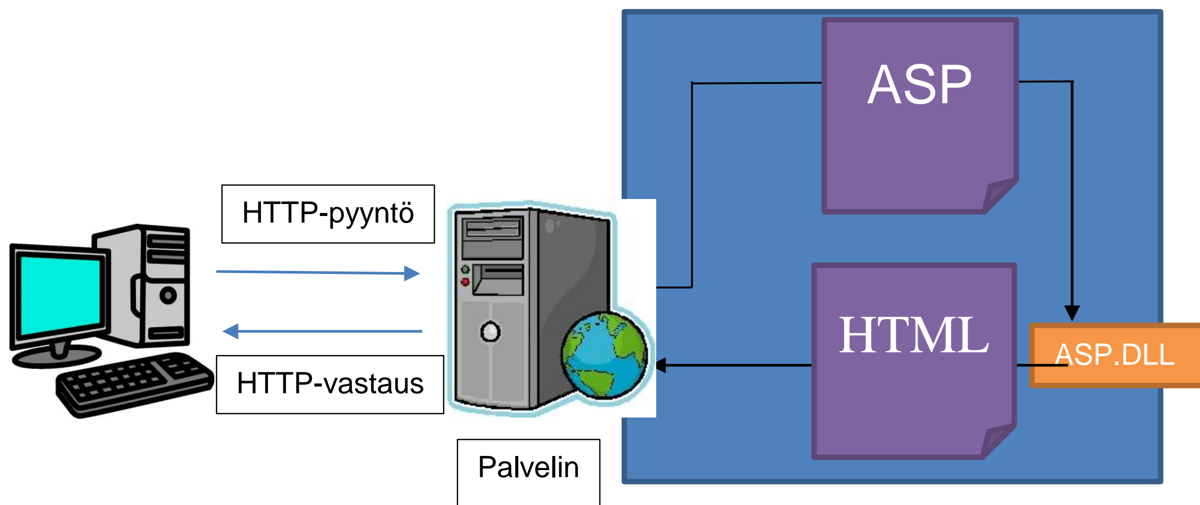
Useimmat suuret nettisivut ovat dynaamisia, kosta niitä on siten helpompi ylläpitää. Toisin kuin staattisissa sivuissa, joissa tieto on kovakoodattu jokaiseen sivuun, dynaamisissa sivuissa päivitettävä tieto on usein tietokannoissa. Tämän takia sivustojen päivitys onnistuu helposti vain tietokantaa päivittämällä. Tämä mahdollistaa myös sen, että useat henkilöt voivat päivittää sivua ilman tarvetta muuttaa nettisivun ulkonäköä. (3.)

2.2.1 Active Server Pages

ASP eli Active Server Pages on Microsoftin vuonna 1996 julkaisema teknologia palvelinpuolen koodaukseen. ASP käyttää Jsciptiä (Microsoftin versiota JavaSciptistä) ja VBScriptiä, joilla tehdään palvelinpuolella ajattavia koodeja. Tämä mahdollistaa HTML-sivujen luonnin ilman kovakoodausta. ASP käyttää sivujen luonnissa käyttäjän syöttöä ja Microsoft Data-Object-tietoja, jotka otetaan tietokannasta. ASP käyttää myös hyödyksi ActiveX-komponentteja. Tämä mahdollistaa uusien toiminnallisuuksien tekemisen, mikäli niitä ei ole jo valmiiksi ladatuissa komponenteissa.

Esimerkkinä ASP-toiminnasta toimii radion ohjelman näyttäminen nettisivulla. Radion nettisivu kertoo asp-tiedoston avulla päivän ohjelman ottamalla tiedon tietokannasta. Tietokanta pitää sisällään tiedon viikonpäivästä, ajasta, nimestä ja muista tiedoista. Nettisivun avatessa ASP hankkii tiedot kyseisen päivän perusteella ja muuttaa sen tekstiksi ja näyttää sen sivulla. Näin jokaisena päivänä sivu näyttää erilaiselta.

Palvelinpuolella tämä toimii niin, että kun käyttäjältä tulee pyyntö saada sivu ASP-palvelin luo HTML-sivun ASP-koodin pohjalta ja antaa sen sitten käyttäjälle. Tämä tarkoittaa sitä, että käyttäjä ei koskaan näe ASP-koodia vaan ASP:n luoman HTML koodin, mikä näkyy alla olevasta kuvassa 2. (4.)

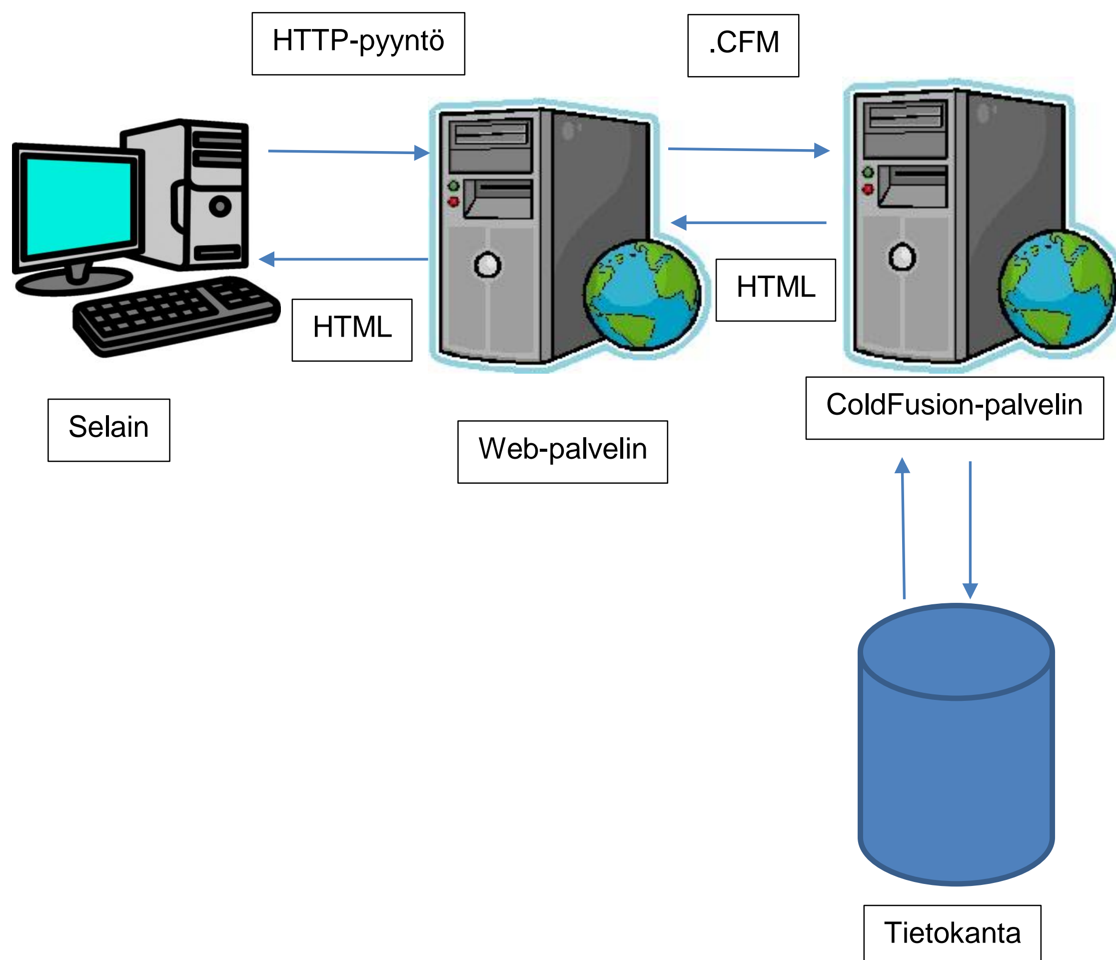


Kuva 2. ASP-toiminta

2.2.2 ColdFusion

ColdFusion on Macromedian tekemä palvelinsovellus, joka mahdollistaa interaktiivisten ja paljon dataa sisältävien sivujen tekemisen helposti ilman suurempaa koodaustaitoa. Nettisivut luodaan lisäämällä korkean tason alustusfunktioita, ehto-operaattoreja ja tietokantaoperaattoreja HTML-koodiin mukaan. Uusien komentojen avulla ColdFusion luo tarvittavat rakennuspalikat ja tekee nettisivukokonaisuuksia.(5.)

Käytännössä tämä toimii niin, että ColdFusion lukee HTML-koodissa olevat CF-tagit tai CF-muuttujat ja tekee asiat joita niissä halutaan. Kaikki HTML-koodi, joissa ei ole merkintöjä, jätetään huomioimatta ja annetaan palvelimelle muuttumattomana. Kun ColdFusion tapahtuman on käyty läpi, annetaan ne palvelimelle, joka yhdistää ne HTML-osan kanssa ja lähettää käyttäjälle, mikä näkyy alla olevassa kuvassa 3. (6.)



Kuva 3. ColdFusion-toimintakaavio

2.3 Ajax

Jokainen, joka on käyttänyt Gmail-, Google suggest- tai Google Maps-ohjelmia on käyttänyt Ajaxia. Ajax on lyhenne sanoista Asynchronous JavaScript Technology and XML, ja tämä teknologioiden yhdistelmä syntyi dynaamisten verkkosivujen luomisen avuksi.

JavaScript-teknologiaa käyttäen HTML-sivu kutsuu asynkronisesti palvelinta ja hakee tietoa, joka voidaan muuttaa XML-dokumentiksi, HTML-sisällöksi, pelkäksi tekstiksi tai JavaScript Object Notationiksi (JSON). Tämän jälkeen JavaScript voi päivittää tai muuttaa HTML-sivun Document Object Modelia (DOM).

Perinteisissä web-ohjelmissa jokainen tapahtuma sivulla antoi käyttäjälle uuden päivitetyn HTML-sivun. Ajaxin avulla pystyttiin luomaan sivuja, joissa ei tarvinnut päivittää kokosivua uudelleen, vaan pystyttiin päivittämään vain pieni osa sivua. Tämä oli mahdollista, koska sivu pystyi hakemaan sivun tiedot HTML-pohjasta, jota päivitettiin palvelimelta saatavan XML-datan avulla. (7.)

3 Perustietoa

3.1 Node.JS:n historia

2009 vuoden lopulla Ryan Dahl julkisti teknologian nimeltä Node.js JavaScript messuilla Berliinissä. Kaikkien yllätyksesi kyseistä teknologiaa ei ole suunniteltu käymään selaimessa, jonka JavaScript oli vallannut ja johon sen luutiin aina pysyvän.

Node.JS:n ytimenä oli JavaScriptin käyttö palvelimessa. Kyseinen tieto herätti yleisön mielikuvituksen hetkessä, ja yksi lause oli kaikkien mielessä. ”if done right, we could write web application in just one language”. Aikaisemin modernin web-applikaation tekeminen vaati hyvää kykyä käyttää JavaScriptiä selaimeen tehtävän sivun takia, mutta palvelin vaati täysin erillisiä asioita.

Ryanin näyttämä demonstraatio yksinkertaisella mutta ytimekkäällä ”hello world” Node.js-ohjelmalla muutti tämän kaiken

```
var http = require(‘http’);
var server = http.createServer(function (req, res) {
 res.writeHead(200);
 res.end(’Hello world’);
});
server.listen(80);
```

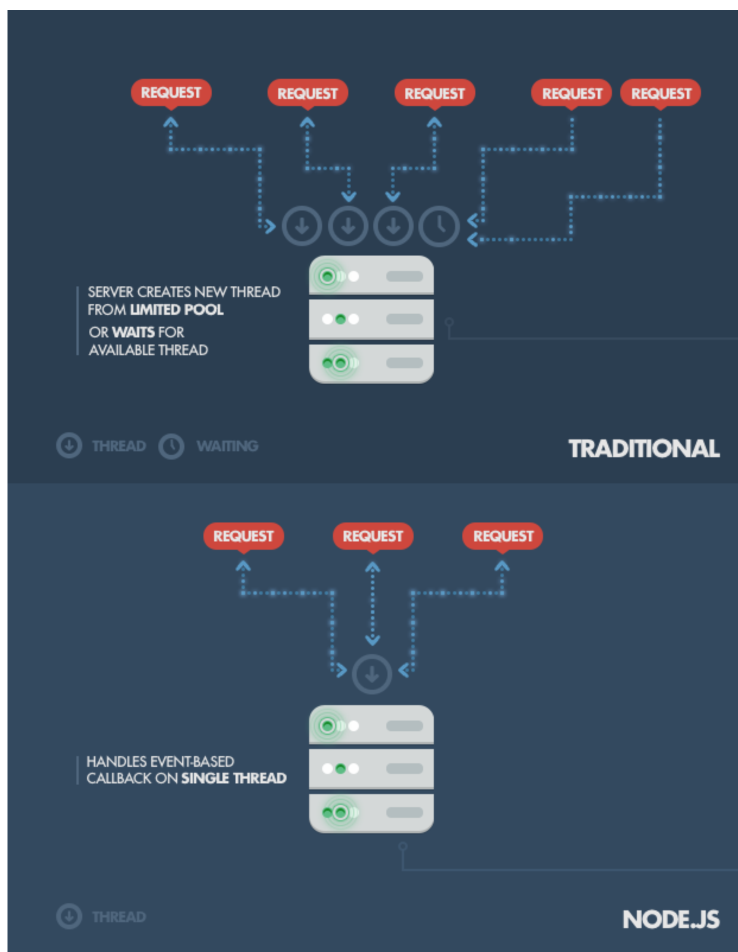
Koodi 1. Node.js ”hello world” ohjelma

Tämä näytti ensimmäisen kerran, että JavaScript-koodia voidaan käyttää palvelimella ja sen avulla voidaan rakentaa palvelinsovellus. Node.js tarjoaa käytännössä vaihtoehdon palvelimen toteutukselle ja se pärjää yhtä hyvin kuin aikaisemmin tulleet Apache ja Nginx ellei jopa paremmin useissa eri tilanteissa. Julkistaessa Node.js:n sanottiin olevan työkalu, jolla verkkoapplikaatiot tehdään juuri oikealla tavalla.

Node.JS:n suuren nopeuden ja suorituskyvyn mahdollisti sen käyttämä ”event loop” -systeemi ja sen käy V8 JavaScript-moottorilla, jota Google Ghrome käyttää sen suuren nopeutensa saavuttamiseksi. Tämä kaikki mahdollisti sen, että web-kehityksen aikana ei tarvitse kirjoittaa erillisiä ohjelmakoodeja web-palvelimen käyttöön, jotka pitää asentaa erikseen ja tarvitsee PHP:tä tai Apachea. Näin Node.js mahdollisti monien uusien reaaliaikaisten web-applikaatioiden tekemisen ja aikaisempien web-applikaatioiden luultujen rajojen rikkomisen. (1, s. 15.)

3.2 Node.js lyhyesti

Node.js:n pääidea on käyttää esteetöntä, eli mikään käsky ei estä toisen käskyn ajoa, ja tapahtumapohjaista I/O-järjestelmää. Tämän takia se pysyy kevytrakenteisena ja tehokkaana dataintensiivisissä ajantasaisissa ohjelmissa monissa eri laitteissa. Käytännössä tämä tarkoittaa sitä, että verrattuna perinteiseen verkkopalvelutekniikkaan, missä jokainen yhteys saa uuden säikeen käyttäen paljon RAM-muistia ja lopulta vie kaiken käytössä olevan. Node.js käyttää vain yhtä säiettä tukemaan jopa kymmeniä tuhansia samanaikaisia yhteyksiä, mikä näkyy kuvassa 4. Tämän takia Node.js loistaa rakennettaessa nopeita ja hyvin skaalautuvia verkko-ohjelmia.



Kuva perinteisten ja node.js-yhteyksien toiminnasta (8)

Jos ajatellaan, että jokainen säie käyttää kaksi MB-muistia, niin kahdeksan GB RAM antaa teoriassa maksimissaan 4000 samanaikaista yhteyttä säikeiden sisältämän tietojen vaihdon käytön kulutuksen lisäksi. Tämä tilanne tulee vastaan perinteisessä verkkopalvelutekniikassa, mutta Node.js välttää tämän ja pystyy siksi skaalautumaan yli miljoonan samanaikaisen käyttäjän tasolle.

Node.js ei kuitenkaan ole uusi alusta, joka on täysin ylitse muiden, vaan se antaa vastauksen joihinkin ongelmiin. Esimerkiksi paljon prosessoria käyttävissä operaatioissa Node.js:sän hyödyt katoavat melkein kokonaan. Myös vain yhden säikeen käyttäminen voi luoda ongelmia, mikäli se saa paljon laskentaan tarvitsemia operaatioita, jotka voivat hidastaa ohjelman käyttöä muilta käyttäjiltä. Myöskään virheilmoituksien jääminen Node.js:n korkeimpaan tapahtumasarjaan pitää välttää tai Node.js sammuttaa itsensä.

Tämä korjataan lähettämällä virheilmoitus takaisin käyttäjälle aina kutsun yhteydessä callback-muuttujana, eikä käyttämällä throw-komentoja kuten muissa ympäristöissä. (8.)

3.3 Node Package Manager

NPM eli Node Package Manager on Node.js:sän käyttöä auttava ohjelma. Se toimii online-säilytyspaikkana julkistetuille open-source Node.js -projekteille ja komentorivityökaluna, joka auttaa kyseisen säilytyspaikan pakettien asentamisessa, version hallinnassa ja riippuvuussuhteiden hallinnassa. Paljon Node.js-kirjastoja ja ohjelmia on julkistettu npm:ään ja uusia tulee päivittäin. Näitä ohjelmia voi helposti etsiä sivulta http://search.npmjs.org/, ja kun haluttu paketti on löytynyt, sen voi helposti asentaa yhdellä komentorivi-komennolla.

Esimerkiksi jos haluaa asentaa uuden kirjaston kuten Caolan McMahonin async omaan Node.js-ohjelmaan, niin se onnistuu komennolla npm install async, minkä jälkeen kyseinen kirjasto ladataan senhetkisen kansioon kohtaan ./node_modules/. Tämän jälkeen koodissa tarvitsee käyttää vain require(), ja kaikki kirjaston osat ovat käytettävissä. Riippuvuuksien asentamiseen taas jotain.json-tiedostot tekevät helpoksi. Jos juuri kansiossa on json-tiedosto, niin npm-komennolla npm install kaikki riippuvuudet, joita kyseinen Node.js ohjelma tarvitsee. Tämä tekee Node.js-ohjelmien haun ja asentamisen gitistä todella helpoksi. (9.)

3.4 Soketit ja Web-soketit

Soketit ovat kauan käytössä ollut kommunikointirajapinta sovellusten välillä. Käytännössä tähän kuuluu kaksi osapuolta: ServerSocket-yhteyden muodostamista varten palvelinpäässä ja Socket-oliot kaksisuuntaisen kommunikoinnin toteuttamiseen. Tällaisen ratkaisumallin tarjoaa esimerkiksi Javan API: java.net.Socket- ja java.net.ServerSocket-luokat.

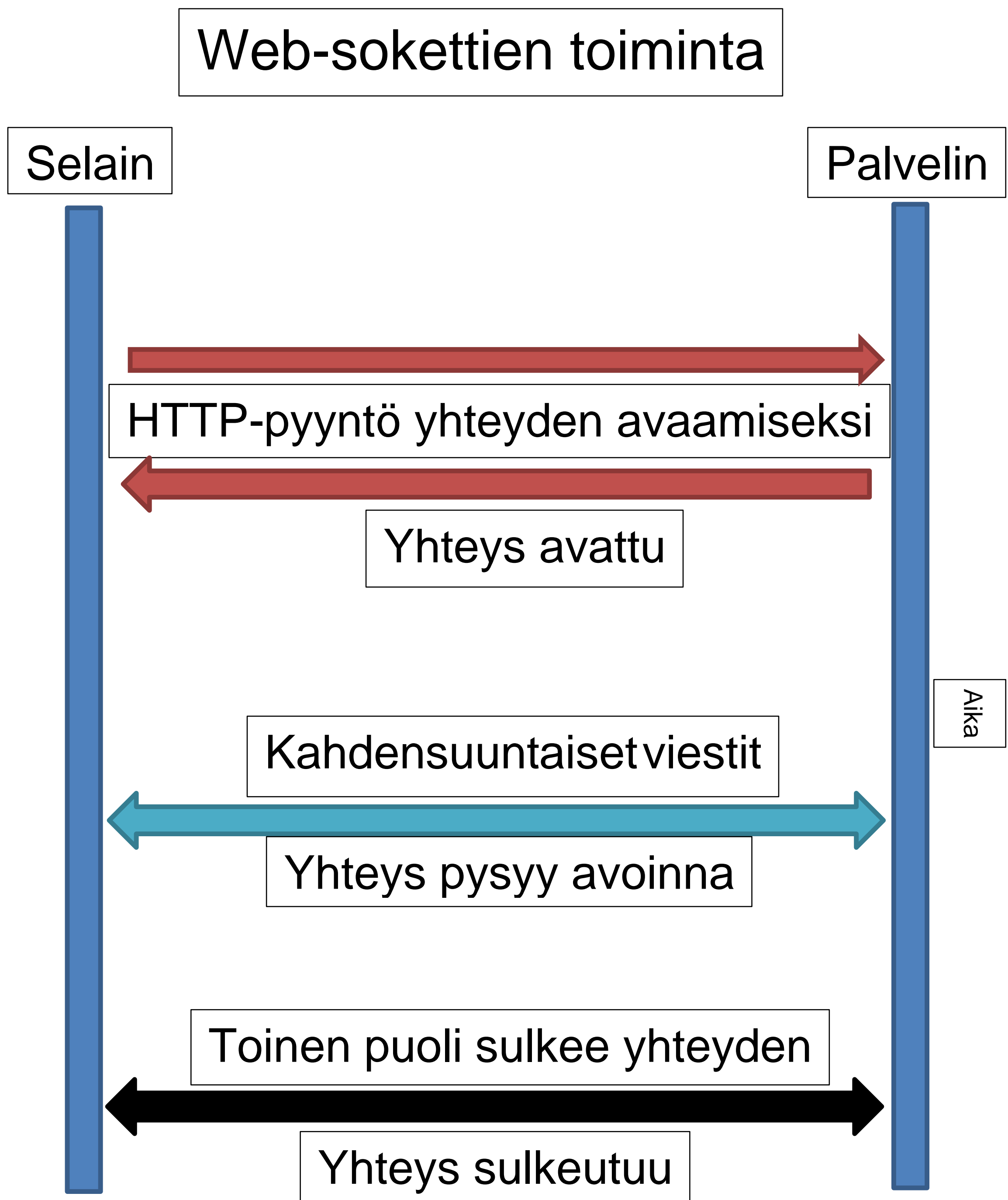
Kaikki nettiyhteydet sisältävät selainpuolen ja palvelinpuolen. Näitä puolia yhdistävät soketit. Palvelinpuolella ohjelma odottaa, että selain ottaa yhteyttä siihen. Kun yhteydenotto tulee palvelinsoketti kiinnittää selaimen koneen portin, minkä jälkeen palvelin

kuuntelee tulevia yhteydenottoja. Palvelimen huomatessa yhteydenoton se yrittää hyväksyä yhteyden. Tämä luo soketin, jota palvelin ja selain käyttävät kommunikoimiseen.

Palvelimen yhteen porttiin voi olla monta eri selainta yhteydessä samanaikaisesti. Tuleva data erotellaan sen mukaan, mihin porttiin se on yhteydessä, ja portin, mistä se tulee. Palvelin pystyy myös kertomaan, mitä palvelua (http/ftp) tarvitaan portin perusteella. Se kertoo myös, mihin avoimeen sokettiin palvelun data pitää lähettää katsomalla selaimen osoitetta ja porttia, joka on tallennettu datassa.

Vain yksi palvelinsoketti voi kuunnella yhtä porttia samanaikaisesti, ja koska palvelin voi joutua kuuntelemaan monia yhteyksiä samanaikaisesti, palvelinohjelmat ovat usein monisäikeisiä. Usein palvelin kuunteleekin vain käyttäjien yhteydenottoyrityksiä ja antaa yhteyksien prosessoinnin toisille säikeille. (10.)

Web-soketit (WebSocket) ovat uusi palvelimen ja selaimen välisen yhteyden välittäjiä. Web-soketit mahdollistavat samanaikaisen viestinnän palvelimen ja selaimen välillä, mikä on muuton AJAX:iin, jossa selaimen pitää lähettää pyyntö ensin palvelimelle, ennen kuin palvelin voi lähettää tietoa selaimeen. Web-soketeilla pystytään myös olla yhteydessä eri verkkotunnuksien välillä, mikä ei onnistunut AJAX:illa.



Kuva 4. Web-sokettien toiminta

Web-sokettien toiminta alkaa http-pyynnön lähettämisellä selaimelta palvelimelle, minkä jälkeen luodaan soketti yhteys, joka sammuu joko selaimen tai palvelimen toimesta, ja tätä selvennetään ylläolevassa kuvassa 5. Web-soketit eivät kuitenkaan toimi kaikilla selaimilla. Esimerkiksi Internet Explorer ei tue Web-sokettien käyttöä. Tässä tilanteessa on

tyydyttävä käyttämään joko Flashiä, joka on yksinkertainen vaihtoehto, mutta ei tue kaikkia käyttäjiä, tai Ajax Long-Polling teknologiaa, joka simuloi web-soketteja, mutta ei ole optimoitu viestien lähettämiseen. Siksi on hyvä asia, että on olemassa Socket.io, joka yhdistää web-sokettien ominaisuudet, palvelinratkaisuja ja vaihtoehtoisia viestien lähetystapoja, kuten AJAX Long-Polling, yhteen API:iin.

Socket.io on tapahtumapohjainen kaksisuuntainen kommunikaatiokerros reaaliaikaisille web-ohjelmille. Se antaa mahdollisuuden luoda Web-ohjelmia kaikille selaimille ilman huolta yhteensopivuusongelmista. Tämä onnistuu, kun socket.io havaitsee, mitä toimintoja pitää käyttää, minkä jälkeen se päättä käytetäänkö web-soketteja, AJAX Long-Polling;ia vai jotain muuta mahdollista teknologiaa. (11.)

Socket.io saavutti version 1.0 toukokuussa 2014. Ennen tätä Socket.io sisälsi siirron hallintaosan ja korkean tason API:n. Siirron hallintaosa siirtyi toiseen projektiin nimeltä Engine.io. Tämä salli muiden kehittäjien rakentaa uusia rajapintoja ja projekteja reaaliaikaiselle netille ilman, että pitäisi tehdä kaikkea uudestaan.(12.)

3.5 Express.js

Express.js on nodella ohjelmiin ladattava lisäkehys. Expressiä käytetään paljon, sillä se nopeuttaa joidenkin asioiden koodausta, kuten reittien tekemisen nettisivulla. Tämä onnistuu monien middleware-funktioiden avulla, jotka lisäävät jotain hyödyllistä pyyntöön, kun se liikkuu niiden läpi suorituksen aikana.

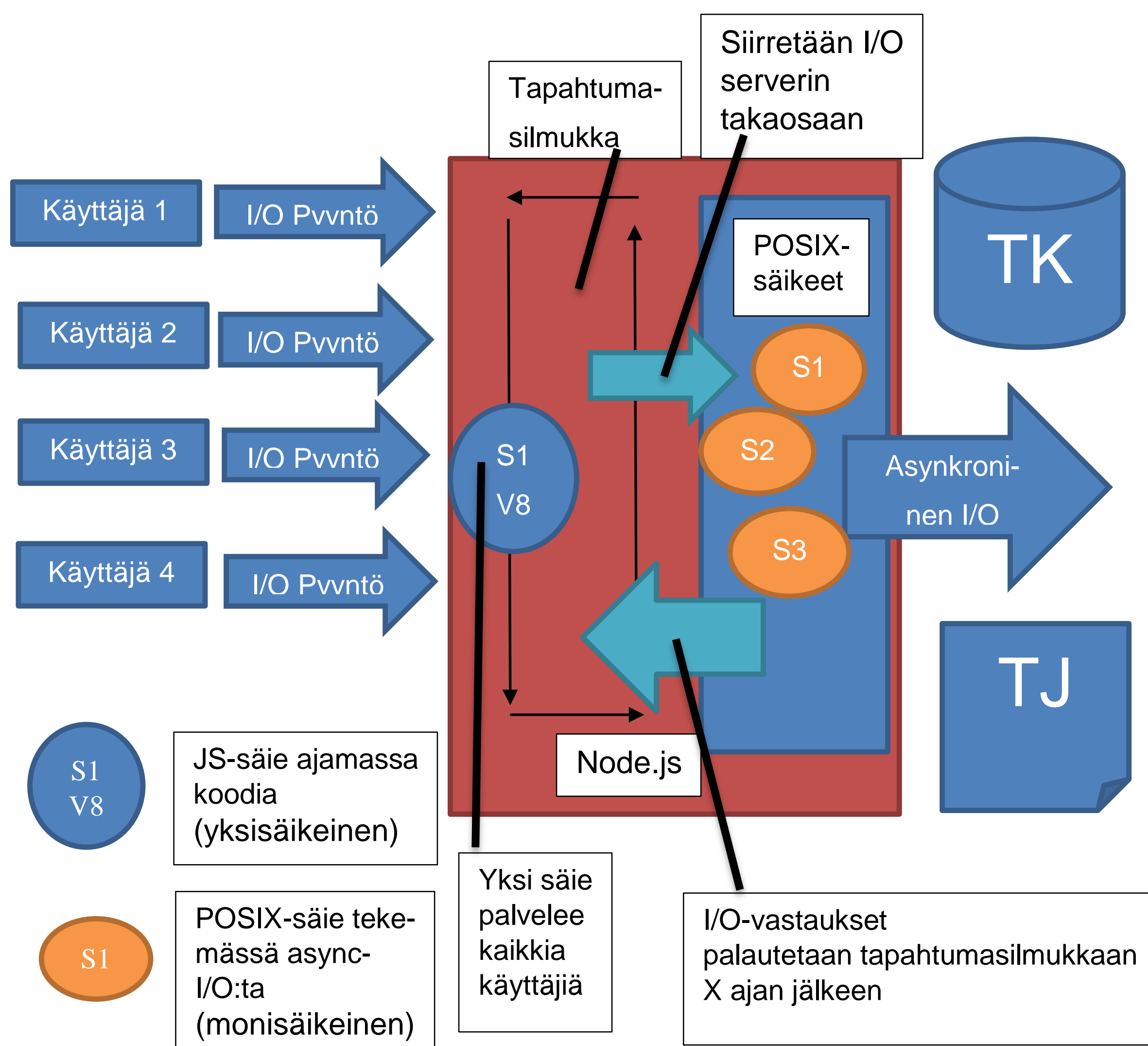
4 Node.js

4.1 Node.js perusrakenne

Node.js mahdollistaa JavaScript-koodin käytön palvelinpuolella. Noin 80 % siitä on tehty käyttäen C/C++-koodia ja noin 20 % on JavaScriptillä. Node.js:n C/C++-osio sisältää kirjastoja, jotka vastaavat JavaScriptin käytöstä Chrome V8 JavaScript -moottorin avulla ja tukevat tärkeiden palvelin osien käyttöä kuten http:n, DNS:n ja TCP:n. JavaScript-osion tehtävä on helpottaa palvelinpuolen kehitystä sen eri kirjastoilla.

Node.js:n palvelin pitää sisältää kaksi osaa. Palvelimen etuosa (front-end) sisältää V8-moottorin, tapahtumasilmukan ja muita C/C++-kirjastoja, jotka ajavat JavaScript-koodin ja kuuntelevat HTTP/TCP-pyyntöjä. Takaosa (back-end) sisältää libuv- ja muita C/C++ -kirjastoja, jotka mahdollistavat asynkronisen I/O:n.

Käytännössä tämä toimii niin, että kun pyyntö saapuu palvelimelle, pääsäe V8-moottorin sisällä tarkistaa, onko pyyntö I/O. Mikäli pyyntö on I/O, se delegoidaan heti palvelimen takaosalle, missä yksi POSIX-säe tekee siitä asynkronisen I/O:n. Tämän takia pääsäie vapautuu ja on valmis vastaanottamaan uusia pyyntöjä tai tapahtumia. Heti kun vastaus I/O:n tulee, takaosa luo tapahtuman, jonka takia V8 vapauduttuaan sen hetkisestä työstään antaa vastauksen käyttäjälle, kuten alla olevassa kuvassa 6 näkyy. (13.)



Kuva 5. Node.js:n toimintakaavio

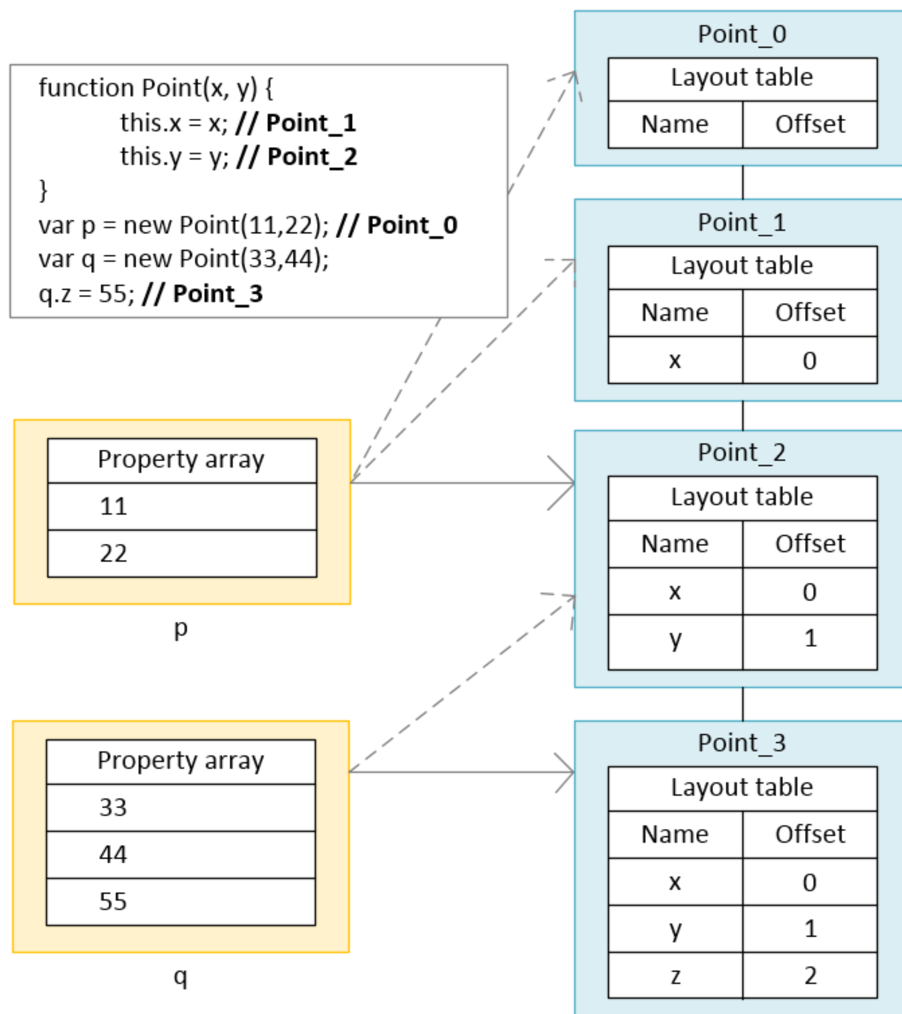
4.2 Chrome V8 -moottori

Chromen V8 moottori on C++:aan perustuva JavaScript-moottori, joka ajaa JavaScript-koodia ja on tehty avoimella lähdekoodilla. V8 kehitettiin lisäämään JavaScriptin suorituskykyä selaimen sisällä. Suuremman nopeuden V8 saavuttaa kääntämällä JavaScript-koodin konekoodiksi tulkkaamisen sijaan. Tämä tapahtuu käyttämällä JIT (Just-In-Time)-kääntäjää, jota monet modernit JavaScript-moottorit, kuten Rhino (Mozilla), käyttävät.

JavaScript on prototyyppipohjainen kieli, mikä tarkoittaa sitä, että se ei sisällä luokkia, vaan kaikki objektit tehdään kloonaamalla. JavaScript on myös dynaaminen, mikä tulee

siitä, että tyypit ja tyyppien tiedot eivät ole täsmällisiä ja tietoa voidaan poistaa ja lisätä objekteihin lennosta. Aikaisemmat JavaScript-moottorit ovat käyttäneet avain-arvo-tyyppisiä (dictionary) tietorakenteita objektien tietojen tallentamiseen ja hakuihin. V8 taas luo piilotettuja luokkia, joihin perustuen se esittää objektit, mikä mahdollistaa nopean pääsyn objektin ominaisuuksiin.

Käytännössä tämä tarkoittaa, että aina, kun JavaScriptin konstruktorifunktio määrittelee ominaisuuden, V8 luo uuden piilotetun luokan, joita se käytännössä ylläpitää useita. Tämä tarkoittaa, että jos luodaan kaksi oliota samalla konstruktorifunktiolla ja toiseen lisätään jotain jälkeenpäin, niin V8 käyttää ensimmäisen olion piilotettua luokkaa ja tekee sen perusteella uuden piilotetun luokan toiselle oliolle, mikä näkyy kuvassa 7. (14.)



Kuva 6. Piilotettujen luokkien käyttö.(14)

4.3 Event Loop

Event loop tai tapahtumasilmukka on yleisesti ottaen ohjelman sisällä oleva rakennelma, joka vastaanottaa ja käsittelee tapahtumia. Vastaanotettuaan tapahtumapyynnön tapahtumasilmukka lähettää tapahtuman suoritettavaksi ja jää odottamaan seuraava pyyntöä. Silmukka ei synny siitä, että tapahtuma olisi aktiivien jatkuvasti, vaan koska silmukka valmistautuu tapahtumaan, tutkiin tapahtuman, lähettää tapahtuman suoritettavaksi ja aloittaa koko prosessin uudestaan alusta. (15.)

Node.js:n tapahtumasilmukka on yhdellä säikeellä toimiva rakennelma. Jokainen I/O:n sisältävälle kutsulle rekisteröidään takaisinkutsu (callback) ja kutsu palautetaan heti. Tämä mahdollistaa käyttäjän tehdä monta eri I/O:ta ilman, että käytetään itse ohjelman säikeitä. I/O:n valmistuttua takaisinkutsu siirretään tapahtumasilmukkaan, missä se suoritetaan heti, kun sitä ennen tulleen takaisinkutsut on suoritettu. Tämän takia ei ole hetkeä, jolloin olisi kaksi eri suorituspolkua. (16.)

5 Node.js-pohjainen Chat-ohjelma

5.1 Chat-ohjelmien historia ja käyttö nykypäivänä

Chat-ohjelmat antavat mahdollisuuden kommunikoida reaaliaikaisesti toisen ihmisen kanssa internetin välityksellä. Tämä on mahdollistanut helpon tavan ihmisten olla yhteydessä toisiinsa ympäri maailmaa ja näin ollen tuoda ihmisiä yhteen. Chat-ohjelmien kehitys kuitenkin alkoi jo ennen 1990-lukua, jolloin ne alkoivat levitä kaikkialle.

Ensimmäisiä Chat-ohjelmia alkoi syntyä jo 1960-luvulla. Compatible Time-Sharing System (CTSS) syntyi 1962 Massachusetts Institute of Technologyn (MIT) toimesta. CTSS mahdollisti 30 käyttäjän kirjautua sisään samanaikaisesti ja lähettää toisillensa viestejä. Tämä kuitenkin vastasi enemmän nykyistä sähköpostia, mutta silloin se yhdisti eri yliopistoja Uudessa Englannissa 1965 mennessä.

1980 luvulla projekti Athenan seurauksena MIT loi Zephyr Notification Service, joka käytti Unixia käyttäjien paikallistamiseen ja viestien siirtelyyn heidän välillä. Vuonna 1982

Commodore International julkisti Commodore 64:n. Commodore 64 sisälsi internetpalvelun nimelta Quantun Link. Tämä palvelu mahdollisti viestien lähettämisen modeemien välillä ja mahdollisti käyttäjien vastaan ottaa viestejä tai vain olla välittämästä niistä.

Chat-ohjelmien suosia alkoi nousta nopeasti vuonna 1997, kun AIM (AOL Instant Messenger) tuli markkinoille. AIM piti sisällään mahdollisuuden viestien välitykseen, oman profiilin tallentamiseen, Away viestit ja erilaisten iconien lisäämisen. Nämä asiat vetivät puoleensa paljon käyttäjiä. Vuoteen 2005 mennesssä AIM oli markkinoiden suosituin Chat-palvelu 53 miljoonalla käyttäjällä.

Vuonna 2003 Skype toi mahdollisuuden käyttäjille olla yhteydessä chatin, äänen ja videon välityksellä. Moni käyttää Skypen teksti chat-ohjelmaa, vaikka se ei ole Skypen suosituin piirre. Vuonna 2011 Skype mahdollisti Facebook-chatin yhdistäminen Skypen chatin kanssa. Nykyään myös verkkokaupat ja monet muut nettisivut pitävät sisällään jonkinlaisen chat-ohjelman. (17.)

Tämän päivän reaaliaikainen kommunikointi on tärkeää niin yksityiselämässä kuin yrityksissä. Yksityiset henkilöt haluat kommunikoida ystävien ja tuttujensa kanssa, ja chat-ohjelmat antavat tähän helpon ja nopean mahdollisuuden, vaikka ystävä olisi maailman toisella puolella. Chat-ohjelmien kehitys myös mahdollistaa videon ja äänen helpon siirron, mitkä tekevät siitä nopeamman ja helpomman kuin vain sähköpostin, kirjeiden tai puhelimen avulla kommunikoinnin.

Yritysten maailmassa sosiaalisen media antaa hyvää näkyvyyttä, sillä niissä on tuhansia käyttäjiä, jotka näkevät yritysten mainoksia. Näin sosiaalisten sivujen ylläpitäjät saavat rahaa yrityksiltä, yritykset saavat näkyvyyttä ja käyttäjät saavat hyvän reaaliaikaisen kommunikointiohjelman. Erilaiset reaaliaikaiset kommunikointiohjelmat ovat käytössä myös erilaisissa palvelu ja myynti yritysten nettisivuilla. Tämä on tärkeää, sillä se mahdollistaa helpon ja nopean tavan asiakkaisen antaa palautetta ja ottaa yhteyttä yritykseen ongelmien takia. Myös yrityksen maine voi saada kolauksia, jos sillä ei ole nopeaa asiakaspalvelua.

Chat-ohjelmat mahdollistavat myös videokokousten pitämisen, mikä nykypäivänä mahdollistaa kokousten pitämisen ilman, että kaikkien osaan ottajien pitää olla fyysisesti paikalla. Jotkut yritykset pitävät myös reaaliaikaisia chat-ohjelmia parempina vaihtoehtoina kuin puhelinen käyttö yritysten sisällä tai ulkopuolella. (18.)

5.2 Toimintaympäristö

Ohjelman toimintaympäristönä toimivat internetselaimet. Tämä tarkoittaa sitä, että jos palvelin on päällä ja käyttäjä haluaa käyttää chat-ohjelmaan, hänen täytyy vain tietää nettisivun osoite päästäkseen käyttämään sitä. Tämän takia tarvittavat laitteet, jotka mahdollistavat ohjelman käyttöön, ovat palvelinkone, jossa palvelinta pidetään päällä, ja käyttäjällä tietokone, jolla on mahdollista päästä internetiin. Ohjelma on siis täysin riippuvainen internetyhteydestä ja palvelinkoneen kunnosta. Jos toinen näistä ei toimi, niin chat-ohjelma ei toimi.

Node.js:n Socket.io on tehty niin, että se tukee kaikkia mahdollisia selaimia ja jopa matkapuhelimia. Chat-ohjelma siis kaikilla alustoilla, jotka tukevat Javascriptiä.

5.3 Sisäinen arkkitehtuuri

Chat-ohjelmassa on käytössä kaksi node.js-moduulia, jotka mahdollistavat helpon ja yksinkertaisen koodauksen ohjelmassa. Ensimmäisenä on espress.js-kehys, jonka tarkoitus on tehdä ohjelman palvelimeen siirtämisen koodin yksinkertaiseksi. Suuremmassa osassa on Socket.io. Socket.io mahdollistaa nopean yhteyden selaimen ja palvelimen välillä käyttäen web-soketteja. Jokainen tapahtuma, joka ohjelmassa huomioidaan, siirretään sokettien avulla selaimelta palvelimelle ja palvelimelta muille selaimille

5.4 Moduulien sisäinen toiminta

Tässä käydään läpi Chattiohjelman koodin toiminta osissa. Koodi kokonaisuudessaan on esitetty liitteissä 1 ja 2.

Chattiohjelma kysyy avautuessaan käyttäjältä käyttäjänimen (koodi 2). Käyttäjänimen antamisen jälkeen nimi tallennetaan muuttujaan, joka lähetetään sitten palvelimelle Socket.io:n socket.emit-komennolla.

```
var socket = io();
var kayttajat = [];
var kayttajalista = '';
var typing = false;
var timeout = undefined;

var name = window.prompt("Gimme your nick")

socket.emit('kayttajat', name);
```

Koodi 2. Käyttäjän nimen kysyminen selaimella ja lähetys palvelimelle

Palvelin odottaa pyyntöä (koodi 3) ja sen vastaanotettuaan tallentaa nimen taulukkoon, joka lähetetään kaikille käyttäjille io.emit-komennolla ja näin uuden käyttäjän nimi ilmestyy myös senhetkisten käyttäjien listaan.

```
socket.on('kayttajat', function(name){
  kayttajat.push(name);
  socket.kayttajan_nimi = name
  io.emit('chat message', socket.kayttajan_nimi+' has joined');
  io.emit('kayttajat', kayttajat);
});
```

Koodi 3. Nimen vastaanottaminen palvelimella ja sen prosessointi

Tässä vaiheessa kaikki käyttäjien selaimet vastaanottavat ilmoituksen siitä, että uusi käyttäjä on tullut chattiin mukaan (kuva 8).

Kuva 7. Uuden käyttäjän ilmoitus ja käyttäjälista selaimessa

Käyttäjälistan saatuaan (koodi 4) vanha lista poistetaan selaimesta ja uusi kirjoitetaan tilalle.

```
socket.on('kayttajat', function(kayttajat){
  $('#kayttajat li').remove()
  for (var i = 0; i < kayttajat.length; i++){
      $('#kayttajat').append('<li>' + kayttajat[i] + '</li>');
  }
});
```

Koodi 4. Käyttäjälistan kirjoittaminen selaimeen

Käyttäjän aloittaessa kirjoittaminen (koodi 5) tarkistetaan, painaako käyttäjä enter-näppäintä. Jos painaa, niin palvelimelle lähetettään tieto, että joku kirjoittaa ja kirjoitusaika laitetaan viiteen sekuntiin. Jos painetaan enter-näppäintä, nollataan kirjoitusaika.

```
$("#m").keypress(function(e){
  if (e.which !== 13){
      typing = true;
      socket.emit('is typing', typing);
      timeout = setTimeout(timeoutFunction, 5000);
  }
  else {
      clearTimeout(timeout);
      timeout = setTimeout(timeoutFunction, 5000);
  }
});
```

Koodi 5. Tarkistetaan kirjoitetaanko mitään

Kirjoitusajan nollaantuessa (koodi 6) palvelimelle ilmoitetaan kirjoittamisen loppuneen.

```
function timeoutFunction(){
  typing = false;
  socket.emit('is typing', typing);
}
```

Koodi 6. Kirjoitusaika nollataan ja ilmoitetaan, että kukaan ei kirjoita.

Palvelin odottaa pyyntöä (koodi 7) siitä, että kirjoittaako joku, ja sen vastaanottaessaan tutkii, onko pyynnön mukana tullut muuttuja true vai false. Jos muuttuja on true, tutkitaan, onko kirjoitus alkanut juuri, jolloin kirjoittajan nimi siirretään kirjoittajien listaan ja lähetetään lista selaimiin, vai onko kirjoittaminen alkanut jo aikaisemmin, jolloin lista lähetään selaimiin. Jos pyynnössä saatu muuttuja on false, poistetaan kirjoittajan nimi kirjoittajat-listasta ja lähetään uusi lista selaimiin

```
socket.on('is typing', function(typing){
    if (typing === true){
       rtopush = true;
       for (var i = 0; i < kayttajat.length; i++){
                    if (typers[i] == socket.kayttajan_nimi){
                                rtopush = false;
                                }
                    }

                    if (rtopush === true) {
                                typers.push(socket.kayttajan_nimi);
                    }
                    console.log(typers);
                    io.emit('typing', typers);
       }
       else {
           for (var i = 0; i < kayttajat.length; i++){
                    if (typers[i] == socket.kayttajan_nimi){
                                typers.splice(i, 1);
                                }
                    }
                    io.emit('typing', typers);
       }
  });
```

Koodi 7. Tiedon kirjoittaako vastaanottaminen joku palvelimella ja sen prosessointi

Kirjoittajalistan saapuessa selaimeen (koodi 8) tarkistetaan, onko lista tyhjä. Jos lista ei ole tyhjä, poistetaan vanha kirjoittaa teksti ja uusi kirjoitetaan tilalle (Kuva 9). Jos lista oli tyhjä, poistetaan vain vanha kirjoittaa teksti.

Julius is typing
Olli is typing

Kuva 8. Kirjoittaa ilmoitus

```
socket.on('typing', function(user){
  if (user !== '') {
      $('#istyping p').remove();
      for (var i = 0; i < user.length; i++){
      $('#istyping').append('<p>' + user[i]+ " is typing" + '</p>');
       }
  }
  else {
    $('#istyping p').remove();
  }
});
```

Koodi 8. Kirjoitetaan kirjoittajalista selaimen ikkunaan

Käyttäjän kirjoitettua tekstinsä selaimeen (koodi 9) ja painettuaan enter-näppäintä tai Send-painiketta selaimessa lähetetään teksti palvelimella

```
$('form').submit(function(){

  socket.emit('chat message', name+': '+$('#m').val());
  $('#m').val('');
  return false;
});
```

Koodi 9. Siirretään teksti palvelimeen

Palvelimen vastaanottaessa pyyntö viestistä (koodi 10), se lähetään eteenpäin kaikille selaimille.

```
socket.on('chat message', function(msg){
    io.emit('chat message', msg);
  });
```

Koodi 10. Viestin lähetys selaimiin palvelimessa

Palvelimelta saatu viesti kirjoitetaan (koodi 11) selaimen ikkunaan ja nollataan kirjoitusaika, jolla ilmoitetaan, että kirjoitus on loppunut.

```
socket.on('chat message', function(msg){
  $('#messages').append($('<li>').text(msg));
  clearTimeout(timeout);
  timeout = setTimeout(timeoutFunction, 0);
});
```

Koodi 11. Kirjoitetaan teksti selaimeen ja nollataan kirjoitusaika

Lisätään palvelinpuoleen express.js- ja socket.io-moduuli (koodi 12). Määrätään myös, että palvelin saa palvelintiedot express-kehyksen kautta.

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

Koodi 12. Express.js ja socket.io alustetaan

Express-moduuli saa tiedot palvelimen koodista koneelta (koodi 13).

```
var kayttajat = [];
var kayttajalista = '';
var typers =[];
var rtopush = true;

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

Koodi 13. Annetaan tiedoston tiedot express.js-moduulille

Avataan sokettiyhteys ja kirjataan konsoliin (koodi 14), jos joku avaa selaimen.

```
io.on('connection', function(socket){
  console.log('a user connected');
```

Koodi 14. Sokettiyhteyden avaaminen

Kun yhteys palvelimen ja selaimen välillä katkeaa (koodi 15), palvelin lähettää viestin siitä selaimiin (kuva 10). Palvelin poistaa myös käyttäjän nimen käyttäjälistasta ja päivitetty lista lähetetään selaimille, jossa se kirjoitetaan selaimen ikkunaan (koodi 4).

Mikael has left

Julius has left

Kuva 9. Käyttäjän lähtemisen ilmoitus

```
  socket.on('disconnect', function(){
    console.log(socket.kayttajan_nimi + ' user disconnected');
    io.emit('chat message', socket.kayttajan_nimi+' has left');
           for (var i = 0; i < kayttajat.length; i++){
                   if (kayttajat[i] == socket.kayttajan_nimi){
                       kayttajat.splice(i, 1);
                       io.emit('kayttajat', kayttajat);
                   }
           }
  });
});
```

Koodi 15. Yhteyden katkeaminen ja sen toiminnot

Määritellään (koodi 16), mitä porttia ja IP-osoitetta palvelin kuuntelee ja kirjataan se konsoliin.

```
http.listen(3000, "0.0.0.0",function(){
  console.log('listening on *:3000');
});
```

Koodi 16. Määrätään, mitä porttia palvelin kuuntelee

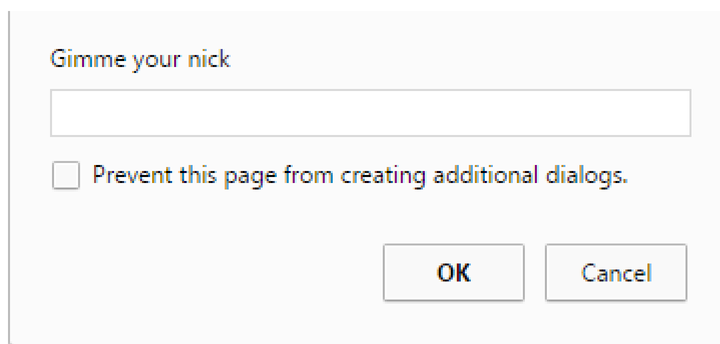
5.5 Käyttöohje

Palvelin laitetaan päälle komentorivityökalulla komennolla ”node index.js”, mikä onnistuu vain samassa kansiossa kuin missä ohjelma sijaitsee. Tämän jälkeen node.js käynnistää kyseisen palvelimen ja alkaa kuunnella ohjemassa määrättyä porttia.

```
D:¥Ohjelmat¥chat thing>node index.js
listening on *:3000
```

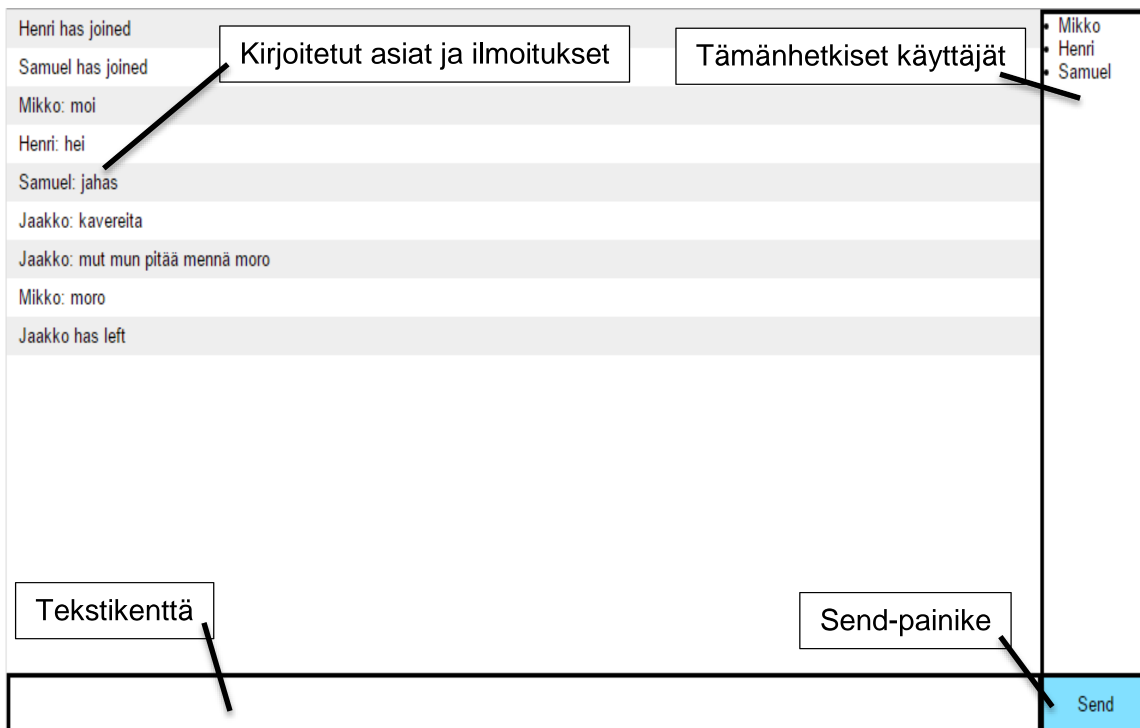
Kuva 10. Käynnistetään palvelin

Tämän jälkeen sovellukseen pääsee antamalla selaimeen palvelimen IP-osoitteen ja perään ohjelman portin numeron esimerkiksi ”xxx.xxx.xxx.xxx:3000”. Tämän jälkeen ohjelma kysyy nimeä ja sen annettua käyttäjä voi keskustella ohjelman avulla kaikkien niiden kanssa, ketkä käyttävät ohjelmaa.



Kuva 11. Kysytään käyttäjältä nimeä

Keskustelu tapahtuu kirjoittamalla tekstikenttään jotain tekstiä, ja sen jälkeen painamalla enter-näppäintä näppäimistöllä tai Send-painiketta selaimessa. Oikeassa reunassa näkyy ohjelmaa käyttävien henkilöiden nimet ja vasemmalla kirjoitetut asiat ja ilmoitukset.



Kuva 12. Chat-ohjelma

6 Yhteenveto

Työssä tutkittiin, miten helposti nopeita web-sovelluksia pystytään toteuttamaan node.js:n avulla. Node.js:n mahdollistama palvelimen ja selaimen javascript-koodaus mahdollistaa sen, että jopa kokemattoman palvelinohjelmoijan on helppo luoda web-sovelluksia. Myös node.js:n nopeuseroavaisuus aikaisemmista web-sovelluksista tuli todettua ja miten sen yksisäikeinen tapahtumasilmukka mahdollistaa sovellusten nopeuden.

Mahdollisia toteutuksia, joissa node.js loistaa, ovat monen pelaajan nettipelit, reaaliaikaiset keskusteluohjelmat, kauko-ohjattavien lelujen hallintaohjelmat, tietokoneen kaukohallinta ohjelmia ja vaikka mitä muuta. Node.js ei kuitenkaan sovi kaikkeen, sillä se tehtiin ratkaisemaan I/O-komentojen aiheuttama kuormitus, mutta ei ohjelmien suorituksesta aiheutuvaa kuormitusta.

Node.js ei ole vanha teknologia, ja se tulee varmasti kehittää sen eri päivitysten, moduulien ja aktiivisen kehitys ympäristön avulla. Tämän takia vain tulevaisuus tietää, mitä kaikkea hyödyllistä tai hauskaa tullaan node.js:n avulla luomaan.

Lähteet

1 Guillermo Rauch, 2012. Smashing Node.js JavaScript Everywhere. Verkkodokumentti. <http://aod.zyklon-b.tk/library/web-development/%7BEN%7D_%5BRauch%5D_Smashing_Node.js.pdf>. Luettu 30.04.2015.

2 2009. Static Website. Verkkodokumentti. <http://techterms.com/definition/staticwebsite>. Luettu 27.10.2015.

3 2009. Dynamic Website. Verkkodokumentti. <http://techterms.com/definition/dynamicwebsite>. Luettu 5.11.2015.

4 Akshay Luther. 2000. Getting Started with ASP. Verkkodokumentti. <http://www.4guysfromrolla.com/webtech/090800-1.shtml>. Luettu 27.10.2015.

5 Ben Forta. 2003.Introducing ColdFusion. Verkkodokumentti. <http://www.adobepress.com/articles/article.asp?p=31062&seqNum=4>. Luettu 28.10.2015.

6 Ben Forta. 2003.Introducing ColdFusion. Verkkodokumentti. <http://www.adobepress.com/articles/article.asp?p=31062&seqNum=5>. Luettu 28.10.2015.

7 Gred Murray. 2005. Asynchronous JavaScript Technology and XML (Ajax) With the Java Platform. Verkkodokumentti. <http://www.oracle.com/technetwork/articles/java/ajax-135201.html>. Luettu 30.04.2015.

8 Tomislav Capan. Why The Hell Would I Use Node.js? A Case-by-Case Tutorial. Verkkodokumentti. <http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js>. Luettu 30.4.2015.

9 MR. Nico Reed. 2011. What is npm? Verkkodokumentti. <https://docs.nodejitsu.com/articles/getting-started/npm/what-is-npm>. Luettu 30.04.2015.

10 Eliotte Rusty Harold. 1997. Server Sockets. Verkkodokumentti. <http://www.cafeaulait.org/course/week12/24.html>. Luettu 12.11.2015.

11 David Walsh. 2010. WebSocket and Socket.IO. Verkkodokumentti. <https://davidwalsh.name/websocket>. Luettu 23.11.2015.

12 Fionn Kelleher. 2014. Understanding Socket.IO. Verkkodokumentti. <https://nodesource.com/blog/understanding-socketio>. Luettu 30.04.2015.

13 Raja Rao DV. 2012. Future-proofing Your Apps: Cloud Foundry and Node.js. Verkkodokumentti. <http://blog.pivotal.io/cloud-foundry-pivotal/products/future-proofing-your-apps-cloud-foundry-and-node-js>. Luettu 04.05.2015.

14 Thibault Laurens. 2013. How the V8 engine works? Verkkodokumentti. <http://thibaultlaurens.github.io/javascript/2013/04/29/how-the-v8-engine-works/>. Luettu 04.05. 2015.

15 Alex Newth. 2015. What is Event Loop. Verkkodokumentti. <http://www.wisegeek.com/what-is-an-event-loop.htm#>. Luettu 12.11.2015.

16 Willem D'Haeseleer. 2014. Event loop. Verkkodokumentti. <http://stackoverflow.com/questions/25568613/node-js-event-loop>. Luettu 12.11.2015.

17 Matt Petrozio. 2012. A Brief History of Instant Messaging. Verkkodukomentti. < http://mashable.com/2012/10/25/instant-messaging-history/#rhHN80O.IEqa>. Luettu 12.11.2015.

18 Tagove. 2015. Why active online businesses needed live software. Verkkodokumentti. < http://www.tagove.com/active-online-businesses-needed-live-chat-software/>. Luettu 12.11.2015.

Keskusteluohjelman koodi (selain)

```
var socket = io();
var kayttajat = [];
var kayttajalista = '';
var typing = false;
var timeout = undefined;

var name = window.prompt("Gimme your nick")

socket.emit('kayttajat', name);

    $('form').submit(function(){
      socket.emit('chat message', name+': '+$('#m').val());
      $('#m').val('');
      return false;
    });

function timeoutFunction(){
            typing = false;
            socket.emit('is typing', typing);
            }

$("#m").keypress(function(e){
            if (e.which !== 13){
                        typing = true;
                        socket.emit('is typing', typing);
                        timeout = setTimeout(timeoutFunction, 5000);
            }
            else {
                        clearTimeout(timeout);
                        timeout = setTimeout(timeoutFunction, 5000);

            }
});

socket.on('typing', function(user){
            if (user !== '') {
                        $('#istyping p').remove();
                        for (var i = 0; i < user.length; i++){
                        $('#istyping').append('<p>' + user[i]+ " is typing" + '</p>');
                                    }
                        }
            else {
                        $('#istyping p').remove();
                        }
             });

socket.on('kayttajat', function(kayttajat){
            $('#kayttajat li').remove()
            for (var i = 0; i < kayttajat.length; i++){
                        $('#kayttajat').append('<li>' + kayttajat[i] + '</li>');
```

```
        }
});

socket.on('chat message', function(msg){
  $('#messages').append($('<li>').text(msg));
  clearTimeout(timeout);
  timeout = setTimeout(timeoutFunction, 0);
});
```

Keskusteluohjelman koodi (palvelin)

```
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);
var kayttajat = [];
var kayttajalista = '';
var typers =[];
var rtopush = true;

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

io.on('connection', function(socket){
  console.log('a user connected');

  socket.on('chat message', function(msg){
    io.emit('chat message', msg);
  });

  socket.on('kayttajat', function(name){
		kayttajat.push(name);
    socket.kayttajan_nimi = name
    io.emit('kayttajat', kayttajat);
  });

  socket.on('is typing', function(typing){
		if (typing === true){
			rtopush = true;
			for (var i = 0; i < kayttajat.length; i++){
				if (typers[i] == socket.kayttajan_nimi){
					rtopush = false;
					}

			}

			if (rtopush === true) {
				typers.push(socket.kayttajan_nimi);
			}
			console.log(typers);
			io.emit('typing', typers);
		}
		else {
			for (var i = 0; i < kayttajat.length; i++){
				if (typers[i] == socket.kayttajan_nimi){
					typers.splice(i, 1);
					}
			}
			io.emit('typing', typers);
		}
  });
```

```
  socket.on('disconnect', function(){
    console.log(socket.kayttajan_nimi + ' user disconnected');
		io.emit('chat message', socket.kayttajan_nimi+' has left');
			for (var i = 0; i < kayttajat.length; i++){
				if (kayttajat[i] == socket.kayttajan_nimi){
					kayttajat.splice(i, 1);
					io.emit('kayttajat', kayttajat);
				}
			}
  });
});

http.listen(3000, "0.0.0.0",function(){
  console.log('listening on *:3000');
});
```