Opinnäytetyö (AMK)

Tietotekniikan koulutusohjelma

Sulautetut ohjelmistot

2017

Akseli Aarnio

ETÄTERMINAALIYHTEYS SELAIMELLA

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Sulautetut ohjelmistot

2017 | 19

Akseli Aarnio

ETÄTERMINAALIYHTEYS SELAIMELLA

Tämän opinnäytetyön tavoitteena oli suunnitella ja toteuttaa etäterminaaliyhteys asiakaslaitteen ja selaimen välille käyttäen WebSocket-yhteyttä. Työn tarkoituksena oli tehdä toimeksiantaja Inoi Oy:lle yksinkertainen ja helppo ratkaisu laitteiden etäterminaaliyhteyden muodostamiseen, ja joka voidaan liittää heidän laitteiden hallintajärjestelmään.

Työssä toteutettiin ohjelmat asiakaslaitteelle ja selaimelle sekä välityspalvelin, joka yhdistää edellä mainitut toisiinsa. Asiakasohjelma ja välityspalvelin tehtiin Node.js ohjelmointiympäristöllä. Selainohjelman toiminnallisuus tehtiin JavaScriptillä ja sen käyttöliittymä Xterm.js:llä.

Tulokseksi saatiin suunnitellut ohjelmat, joiden avulla etäterminaaliyhteys saadaan muodostettua selaimen ja laitteen välille suunnitellusti. Valmistettu ohjelma voidaan liittää Inoin laitteiden hallintajärjestelmään tekemättä muutoksia siihen.

ASIASANAT:

JavaScript, Node.js, WebSocket.

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Embedded Software

2017 | 19

Akseli Aarnio

REMOTE TERMINAL CONNECTION FROM BROWSER

The purpose of this thesis was to design and create a remote terminal connection between a client device and a browser using WebSocket protocol. The software for the client and the browser were developed, as well as the proxy server which connects client and browser.

The client and the proxy server were developed using the Node.js JavaScript environment. The functionality for browser software was created using JavaScript and the user interface with Xterm.js.

The outcome of this project was three pieces of functional software that work together as planned.

KEYWORDS:

JavaScript, Node.js, WebSocket.

SISÄLTÖ

KUVAT

KOODIT

KÄYTETYT LYHENTEET

HTML	Hypertekstin merkkauskieli.
HTTP	Protokolla, jota selaimet ja WWW-palvelimet käyttävät tiedonsiirtoon.
TCP	Tietoliikenneprotokolla, jolla luodaan yhteyksiä tietokoneiden välille.
URL	Merkkijono joka osoittaa Internetissä olevan tiedon paikan.

1 JOHDANTO

Tässä opinnäytetyössä käydään läpi projekti, joka toteutettiin ohjelmistoyhtiö Inoi Oy:lle. Projektissa oli tarkoituksena tehdä ohjelmat, joiden avulla saadaan etäterminaaliyhteys selaimen kautta asiakaslaitteelle WebSocket-yhteydellä. Työ tullaan liittämään Inoi DMP:hen, joka on Inoin kehittämä laitteiden hallintajärjestelmä.

Suunnitelmana oli toteuttaa ohjelmat asiakaslaitteelle ja selaimelle sekä välityspalvelin, joka yhdistää edellä mainitut toisiinsa.

Asiakasohjelman ja välityspalvelimen ohjelmointikieleksi valittiin JavaScript ja niiden tekoon käytetään Node.js-ohjelmointiympäristöä. Selainohjelman toiminnallisuus toteutetaan JavaScriptillä.

Opinnäytetyön laajuus rajattiin ohjelmien tekemiseen ja siihen pisteeseen, että ne voidaan liittää Inoin laitteiden hallintajärjestelmään tekemättä muutoksia siihen.

2 KÄYTETYT TEKNOLOGIAT

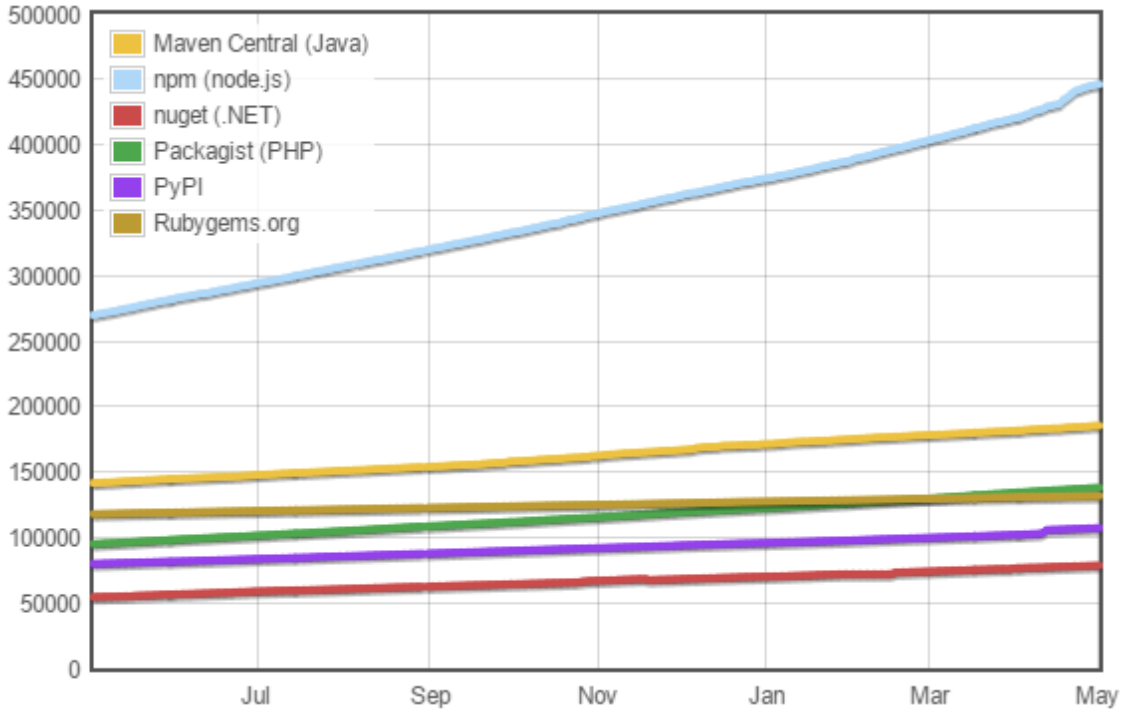
Projektissa käytetään ohjelmointikielenä JavaScriptiä, joka on alustariippumaton oliopohjainen komentosarjakieli, jota käytetään mm. www-sivujen toiminallisuuden lisäämiseen ja palvelinpuolen ohjelmointiin.

2.1 Node.js-ohjelmointiympäristö

Node.js on JavaScript-ohjelmointiympäristö palvelinpuolelle. Se mahdollistaa sekä asiakas-, että palvelinohjelmiston tekemisen samalla ohjelmointikielellä. Se käyttää Google Chromen V8 JavaScript -moottoria JavaScriptin suorittamiseen.

Node.js on suunniteltu skaalautuvien web-sovellusten tekemiseen, ja sen vahvuutena on sen tapahtumapohjainen ja asynkroninen ohjelman suorittaminen. Tämä tarkoittaa sitä, että ohjelma ei jää odottamaan pitkään kestävän suorituksen valmistumista, vaan jatkaa muun ohjelman suoritusta. Kun pitkään kestänyt suoritus, esimerkiksi lataus, on valmistunut, ohjelma suorittaa siitä syntyvän tapahtuman. (Viikon VALO 2014.) Tämä tekee Node.js:stä erittäin hyvin skaalautuvan, ja se sopii hyvin reaaliaikaisien sovellusten tekemiseen. Jos ohjelma vaatii paljon suoritustehoa prosessorilta, yhtä säiettä käyttävä Node.js ei ole välttämättä paras siihen tarkoitukseen.

Node.js:lle on saatavilla valtava määrä moduuleja ja paketteja, joilla voi laajentaa ohjelman ominaisuuksia. Kuvassa 1 vertaillaan Node.js:n moduulien määrää muihin suosittuihin ohjelmointiympäristöihin viimeisen vuoden aikana.



Kuva 1. Moduulien määrä suosituille ohjelmointiympäristöille (Modulecounts.com 2017).

Node.js moduulien asentamiseen käytetään Node Packaged Modules (npm) nimistä työkalua. Npm:n kautta JavaScriptin kehittäjät voivat ladata ja jakaa koodia, jonka tarkoituksena on korjata jokin ongelma. Tyypillinen Node.js sovellus sisältää kymmenittäin npm paketteja. Paketit ovat usein pieniä, ja ideana on, että paketti on rakennuspalikka, joka korjaa yhden ongelman ja tekee sen hyvin. (Npm 2017.)

Npm paketteja hallinnoidaan package.json tiedostolla, joka määrittelee mitä paketteja Node.js ohjelma käyttää. Koodissa 1 on esimerkki tästä tiedostosta, jossa on määritelty riippuvuudeksi Express.

```
{
  "name": "example",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.15.2"
  }
}
```

Koodi 1. Package.json

Tämä tekee muiden pakettien käyttämisestä huomattavasti helpompaa, kun tarvittavat riippuvuudet ladataan automaattisesti ohjelman asennusvaiheessa.

2.2 Express.js

Express on web-sovelluksien tekemiseen suunniteltu minimalistinen Node.js viitekehys. Sen pääominaisuuksiin kuuluu mm. middleware-funktiot, jotka vastaavat Express sovelluksen saamiin pyyntöihin.

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.listen(3000)
```

Koodi 2. Esimerkki Express.js sovelluksesta

Koodi 2 näyttää esimerkin yksinkertaisesta Express.js:llä tehdystä sovelluksesta, jossa Express kuuntelee porttia 3000 ja sovelluksen polkuun ’/’ saapuvia HTTP GET-pyyntöjä, ja vastaa niihin ’Hello World!’.

2.3 JSON

JSON eli JavaScript Object Notation, on kevyt tiedostomuoto datan välittämiseen palvelimen ja asiakkaan välillä. JSON:lla on kaksi eri muotoa: objektimuoto, joka sisältää nimi- ja arvopareja, ja lista, joka sisältää vain arvoja. Nimensä mukaisesti JSON perustuu JavaScriptiin, mutta on täysin riippumaton ohjelmointikielestä. (JSON.org 2017.)

2.4 JSON Web Token

JSON Web Token, eli JWT, on turvallinen laitteiden väliseen varmennukseen tarkoitettu standardi. Se käyttää JSONia, joka tekee siitä helpon käyttää monen suositun ohjelmointikielen kanssa. JWT allekirjoitetaan digitaalisesti tehden siitä turvallisen tavan siirtää dataa ja varmentaa yhteys.

JWT koostuu kolmesta osasta:

1. otsake (header).
2. tietosisältö (payload).
3. allekirjoitus (signature).

Otsake sisältää yleensä kaksi osaa, toinen määrittää tiivistefunktion, ja toinen tokenin tyypin. Koodi 3 on esimerkki otsakkeesta, jossa tiivistefunktiona käytetään HS256, ja jonka tyyppinä on JWT.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Koodi 3. JWT otsake

Tietosisältö voi sisältää tietoa jota halutaan jakaa asiakkaan ja palvelimen välillä. Tämä voi olla esimerkiksi tietoja käyttäjästä, esimerkkinä koodi 4. Se voi myös sisältää vanhentumisajan, jonka jälkeen tietoa ei voi enää käyttää.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Koodi 4. JWT tietosisältö

Otsake ja tietosisältö koodataan Base64:llä, jota käytetään yleisesti kun halutaan siirtää dataa ja varmistaa, että data ei ole muuttunut matkan varrella. Allekirjoitus tehdään allekirjoittamalla otsakkeessa määritellyllä algoritmilla salaisuus ja Base64:lla koodatut otsake ja tietosisältö. (JWT.io 2017.)

Valmis JWT, koodi 5, on yhdistelmä kolmesta edellä mainitusta merkkijonosta, jotka ovat erotultu pisteillä.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Koodi 5. Valmis JWT

JWT on pienikokoinen, ja se voidaan lähettää esimerkiksi osana URL-osoitetta tai HTTP-pyyntöä.

2.5 WebSocket

WebSocket on protokolla, joka tuo kaksisuuntaisen, nopean ja vähemmän dataa käyttävän yhteyden asiakkaan ja palvelimen välille. WebSocket-yhteyttä käytetään, kun

halutaan päivittää esimerkiksi verkkosivulla olevaa tietoa reaaliajassa, esimerkiksi chat-sovellukset ja pelit. (Websocket.org 2017.)

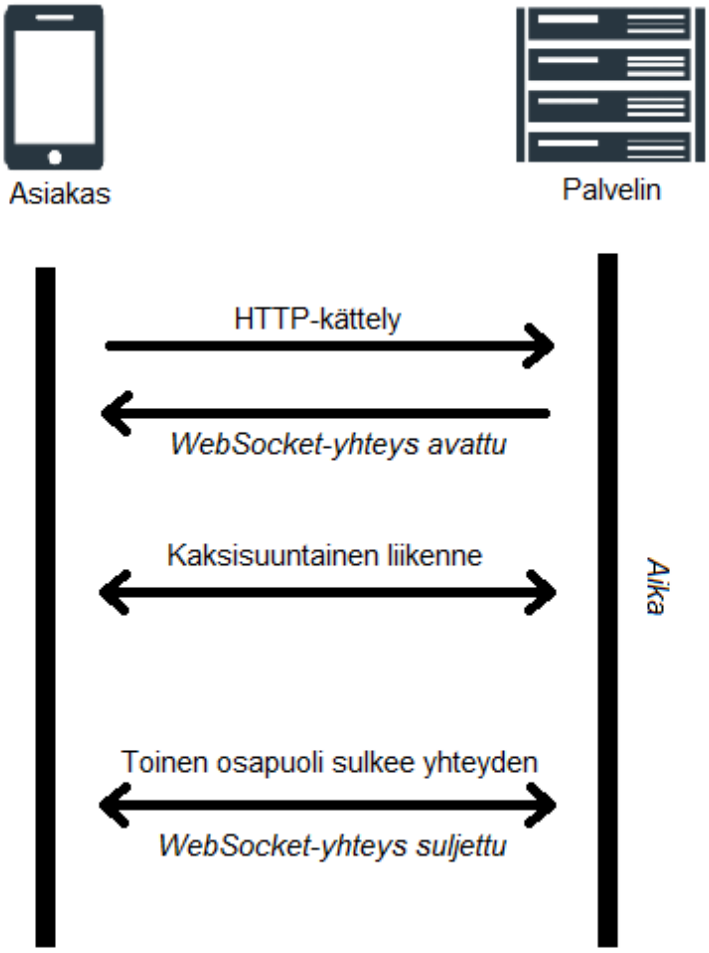
HTTP-yhteys asiakkaan ja palvelimen välillä on tyypillisesti toiminut sillä tavalla, että asiakas lähettää pyynnön palvelimelle, jota varten avataan TCP-yhteys. Palvelin vastaanottaa viestin, tekee mahdollisesti jotain ja vastaa takaisin, jonka jälkeen TCP-yhteys suljetaan.

WebSocket-yhteydessä, yhteys avataan avaamalla TCP-yhteys ja lähettämällä HTTP-pyyntö, joka sisältää pyynnön päivittää HTTP-yhteys WebSocket-yhteydeksi. Esimerkki tästä pyynnöstä näkyy kuvassa 2.

```
GET ws://echo.websocket.org/?encoding=text HTTP/1.1
Origin: http://websocket.org
Cookie: __utma=99as
Connection: Upgrade
Host: echo.websocket.org
Sec-WebSocket-Key: uRovscZjNol/umbTt5uKmw==
Upgrade: websocket
Sec-WebSocket-Version: 13
```

Kuva 2. Esimerkki asiakkaan pyynnöstä aloittaa WebSocket-yhteys.

Jos pyyntö hyväksytään, HTTP-yhteys korvataan WebSocket-yhteydellä, joka käyttää samaa TCP-yhteyttä joka avattiin yhteyden luonnissa. Kuva WebSocket-yhteyden toimintaperiaatteesta kuvassa 3.



Kuva 3. WebSocket-yhteys.

Molemmat osapuolet voivat käyttää luotua WebSocket-yhteyttä ilman kummankaan erillistä suostumusta. Samoin kumpikin osapuoli voi sulkea yhteyden koska vain.

WebSocket ei itse käsittele varmennusta mitenkään, joten varmennus täytyy tehdä esimerkiksi JSON Web Tokenilla. WebSocket sisältää Origin-otsakkeen, joka kertoo mistä URL-osoitteesta yhteys saapuu ja onko laite selain vai jokin muu laite. Tämä on kuitenkin helppo muuttaa, ja siihen ei voi luottaa. (Heroku 2015.)

2.6 Xterm.js

Xterm.js on JavaScriptillä kirjoitettu terminaaliemulaattori, joka tuo terminaaliemulaattorin täysin ominaisuuksin selaimeen. Terminal on Xterm.js:n pääluokka, joka antaa Xterm.js:lle täydet terminaaliemulaattorin ominaisuudet.

3 OHJELMIEN TOTEUTUS

Opinnäytetyö rajattiin ohjelmien tekeminen siihen vaiheeseen, että ne voidaan liittää Inoi DMP:hen, joka on Inoin kehittämä laitteiden hallintajärjestelmä. Se toimii selaimessa, johon työssä tehtävä selainohjelma liitetään.

Projektissa toteutetaan kolme ohjelmaa:

- Selain
- Asiakas
- Välityspalvelin

3.1 Selain

Selainohjelma toteutetaan JavaScriptillä ja Xterm.js:llä. Koodissa 6 on työssä käytetty HTML-koodi, johon on liitetty Xterm.js:n vaatimat tiedostot ja sen lisäosat Attach ja Fit. Attach lisäosaa käytetään liittämään WebSocket yhteys terminaaliemulaattoriin, ja Fit sovittaa sen sivun terminal-container DIV-elementtiin.

```
<html>
    <head>
        <link rel="stylesheet" href="/build/xterm.css" />
        <link rel="stylesheet" href="/build/addons/fullscreen/fullscreen.css" />
        <link rel="stylesheet" href="style.css" />
        <script src="https://cdnjs.cloudflare.com/ajax/libs/fetch/1.0.0/fetch.min.js"></script>
        <script src="/build/xterm.js" ></script>
        <script src="/build/addons/attach/attach.js" ></script>
        <script src="/build/addons/fit/fit.js" ></script>
    </head>
    <body>
        <div id="terminal-container"></div>
        <script src="main.js" defer ></script>
    </body>
</html>
```

Koodi 6. HTML-koodi

Selaimen toiminnallisuus tehdään JavaScriptillä, ja se liitetään HTML:n <script>-tagiin. Selainsovelluksen tarkoituksena on selaimen auettua avata WebSocket-yhteys

välityspalvelimen terminals-polkuun. Yhteyden mukaan liitetään JSON Web Token, joka lähetetään evästeenä. Koodissa 7 näkyy selainsovelluksen toiminta.

```
var terminalContainer = document.getElementById('terminal-container');
..
socket = new WebSocket('wss://<välityspalvelin>/terminals');

socket.onopen = function createTerminal() {
..
..
  term = new Terminal({
    cursorBlink: true
  });
..
..
  term.open(terminalContainer);
  term.fit();
```

Koodi 7. Selainsovelluksen toiminta

Kun yhteys palvelimeen on avattu, luodaan selaimeen terminaaliemulaattori, joka liitetään ja sovitetaan terminal-container DIV-elementtiin. Tämän jälkeen luotu terminaaliemulaattori liitetään WebSocket yhteyteen Xterm.js:n Attach-lisäosalla, joka tarkoittaa sitä, että jokainen painallus minkä terminaali saa, lähetetään välityspalvelimelle.

3.2 Asiakas

Asiakasohjelma tehdään Node.js:llä, ja siinä käytetään Node.js moduulia nimeltään ws. Ws on erittäin kevyt toteutus asiakas- ja palvelinohjelmille WebSocket-yhteyden muodostamiseen. Pseudo-terminaalin luomiseksi asiakas käyttää pty.js Node.js moduulia. Pseudo-terminaali tuo kaksisuuntaisen kommunikoinnin kahden virtuaalisen laitteen välille, tässä tapauksessa asiakaslaitteen ja välityspalvelimen välille.

Ohjelman käynnistyessä asiakas luo pseudo-terminaalin ja avaa WebSocket-yhteyden välityspalvelimen pty-polkuun. Tämän jälkeen asiakas odottaa saapuvia viestejä, ja niiden saapuessa kirjoittaa ne pseudo-terminaaliin, josta saapuva vastaus lähetetään palvelimelle.

3.3 Välityspalvelin

Työssä tehdään välityspalvelin, johon asiakas ja selain yhdistävät. Palvelimen tehtävänä on varmentaa ja yhdistää selain ja asiakas toisiinsa. Palvelin toteutetaan Node.js:llä, Expressillä ja Express-ws:llä. Express-ws avulla WebSocket-yhteyksiä voidaan käyttää Expressissä samalla tavalla kuin muitakin middleware-funktioita, kuten koodista 8 nähdään.

```
app.ws('/terminals', function (ws, req) {
  ..
  ..
});
```

Koodi 8. WebSocket Express.js:ssä

Yhteyden saapuessa palvelin varmentaa yhteyden JSON Web Tokenin perusteella. Jos varmennus epäonnistuu, yhteys katkaistaan. Selaimen yhteyden onnistuessa, selain luo terminaalin, jota selaimessa oleva Xterm.js käyttää toiminnallisuuden luomiseen.

Kun asiakas ja selain ovat yhdistäneet välityspalvelimeen samalla tunnistenumerolla, joka saadaan JWT:stä, voidaan aloittaa viestien lähetys. Selaimessa olevaan terminaaliemulaattoriin kirjoitetaan komento, joka lähetetään välityspalvelimen kautta asiakkaalle. Asiakas vastaanottaa komennon, kirjoittaa sen omaan pseudo-terminaaliinsa, ja lähettää vastauksen takaisin välityspalvelimen kautta selaimeen. Viestit lähetetään ohjelmien kesken JSON-muodossa.

4 YHTEENVETO

Työn tavoitteena oli toteuttaa ohjelmat, joiden avulla saadaan etäterminaaliyhteys selaimen kautta sulautettuun laitteeseen käyttäen WebSocket-protokollaa. Tavoitteena oli lisäksi tehdä työ siihen pisteeseen, että sen voi liittää Inoin laitteiden hallintajärjestelmään tekemättä muutoksia siihen.

Tulokseksi saatiin ohjelmat laitteelle ja selaimelle, sekä välityspalvelin. Etäterminaaliyhteys selaimesta laitteeseen luotiin välityspalvelimen kautta WebSocket-protokollalla. Ohjelma laitteelle ja välityspalvelin toteutettiin Node.js:llä ja sen moduuleilla. Selainohjelma tehtiin JavaScriptin ja Xterm.js:llä.

LÄHTEET

Heroku 2015. WebSocket Security. Viitattu 1.5.2017. https://devcenter.heroku.com/articles/websocket-security.

JSON.org 2017. Introducing JSON. Viitattu 30.4.2017. http://www.json.org/.

JWT.io 2017. Introduction to JSON Web Tokens. Viitattu 30.4.2017. https://jwt.io/introduction/.

Modulecounts.com 2017. Module counts. Viitattu 1.5.2017. http://www.modulecounts.com/.

Npm 2017. What is npm? Viitattu 30.4.2017. https://docs.npmjs.com/getting-started/what-is-npm

Viikon VALO 2014. Node.js. Viitattu 30.4.2017. http://viikonvalo.fi/Node.js/.

Websocket.org 2017. About HTML5 WebSocket. Viitattu 30.4.2017. https://www.websocket.org/aboutwebsocket.htm.