OULUN AMMATTIKORKEAKOULU

Tatu Alapoikela

# TCP-YHTEYDEN KÄYTTÄMINEN WEB-SELAIMEN KAUTTA

# TCP-YHTEYDEN KÄYTTÄMINEN WEB-SELAIMEN KAUTTA

Tatu Alapoikela
Opinnäytetyö
Kevät 2017
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

# TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

---

Tekijä(t): Tatu Alapoikela
Opinnäytetyön nimi: TCP-yhteyden käyttäminen web-selaimen kautta
Työn ohjaaja(t): Pertti Heikkilä
Työn valmistumislukukausi ja -vuosi: Kevät 2017 Sivumäärä: 28

---

Työn tarkoituksena oli suunnitella ja toteuttaa Meiko Oy -nimiselle yritykselle sovellus, jonka avulla voidaan käyttää selaimessa toimivaa kassajärjestelmää lähiverkossa olevien maksupäätteiden kanssa. Sovellus ohjaa määritellystä WebSocket-portista tulevan datan TCP:lle ja TCP:ltä takaisin WebSocketille.

Sovelluksen avulla kassajärjestelmää voidaan käyttää verkkoselaimen kautta, eikä sitä tarvitse toteuttaa erillisenä ohjelmana.

Sovellus toteutettiin JavaScriptillä ja HTML-kielellä. Node.JS:n ja Electronin avulla. Opinnäytetyön aikana sain toteutettua sovelluksen, joka ohjaa tarvittavat tiedot maksupäätteen ja kassan välillä.

---

Asiasanat: tiedonsiirto, web-ohjelmointi, JavaScript, Node.js, Electron, TCP, WebSocket, palvelin

# ALKULAUSE

Haluan kiittää mielenkiintoisesta opinnäytetyöaiheesta Meiko Oy:tä ja Meikon toimitusjohtajaa Mikko Kutilaista. Kiitokset myös ohjaavalle opettajalle Pertti Heikkilälle opinnäytetyön ohjauksesta ja Tuula Hopeavuorelle kieliasun tarkastamisesta.

24.3.2017

Tatu Alapoikela

# SISÄLLYS

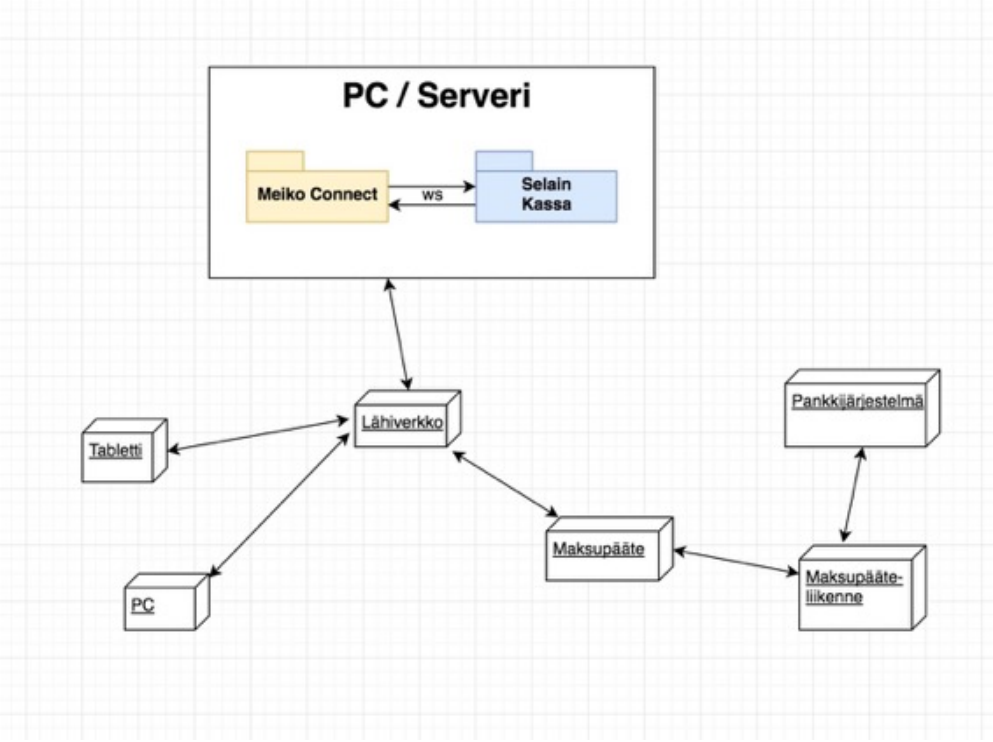
## SANASTO

| | |
|---|---|
| Electron | Node.js-sovelluskehys työpöytäsovelluksien toteuttamiseen |
| HTTP | Hypertext Transfer Protocol. Hypertekstin siirtoprotokolla |
| JavaScript | Web-ympäristössä käytettävä ohjelmointikieli |
| Node.js | JavaScriptiin perustuva kehitysalusta |
| REST | Representational State Transfer, arkkitehtuurimalli rajapintojen toteuttamiseen |
| TCP | Transmission Control Protocol, tietoliikenneprotokolla |
| WebSocket | Kaksisuuntainen kommunikaatioprotokolla |
| WLAN | Wireless Local Area Network, Langaton lähiverkkotekniikka |

# 1 JOHDANTO

Tämän opinnäytetyön aiheena oli suunnitella ja toteuttaa Meiko Oy -nimiselle yritykselle sovellus, jonka avulla voidaan käyttää nykyistä selainpohjaista kassajärjestelmää lähiverkossa toimivan maksupäätteen kanssa. Sovellus ohjaa verkkoselaimelta WebSocket-viestit maksupäätteelle TCP-tietoliikenne protokollan kautta. Tämän sovelluksen avulla myös iPadilla pystyy käyttämään kassajärjestelmää ja maksupäätettä. Verkkoselaimesta ei voi lähettää ulospäin viestejä käyttäen TCP:tä. WebSocketin avulla voidaan kommunikoida HTTP-yhteydellä palvelimen kanssa. Maksupääte toimii WLAN-yhteydellä ja se ottaa vastaan TCP-viestejä. Sovellus ottaa vastaan verkkoselaimen WebSocket-viestit ja ohjaa ne maksupäätteelle TCP-yhteydellä. (Kuva 1.)

Työn toteuttamisessa lähdin liikkeelle tutkimalla mahdollisia työkaluja ja tapoja, joilla sovellus voitaisiin toteuttaa. Sovelluksen toteutin JavaScriptillä käyttäen NodeJS:ää ja Electronia. Työn aloittaessa minulla ei ollut kovin paljoa kokemusta näistä tekniikoista.




*KUVA 1. Kuvaus järjestelmän toiminnasta*

# 2 TEKNOLOGIAT JA KIELET

## 2.1 Node.js

Node.js on avoimen lähdekoodin ajoympäristö. Se käyttää Googlen kehittämää V8-JavaScript-moottoria. Ryan Dahlin julkaisi ensimmäisen version Node.js:stä toukokuussa 2009. (1.)

Node.js:n avulla voidaan tehdä kokonaan palvelinpuolen koodi JavaScriptillä, mukaan lukien web-palvelin, palvelinpuolen skriptit ja web-sovelluksen toiminallisuus (2, s. 2–3).

## 2.2 JavaScript-ohjelmointikieli

JavaScript on kevyt alustariippumaton olio-ohjelmointikieli. JavaScript sisältää useita kirjastoja, kuten Array, Date, Math, ja lisäksi erilaisia kielielementtejä. JavaScriptin avulla voidaan lisätä verkkosivuille dynaamista toiminnallisuutta. JavaScriptiä voidaan laajentaa lisäosien avulla. (3.)

Asiakaspuolella JavaScriptiä voi hallita esimerkiksi HTML-lomakkeita ja ottaa vastaan käyttäjän tapahtumia, kuten hiiren klikkauksia, lomakkeen lähettämistä ja sivulla navigointia. JavaScriptiä voidaan ajaa myös palvelinpuolella, joka antaa mahdollisuuden web-ohjelmalle kommunikoida esimerkiksi tietokannan kanssa ja muokata tiedostoja palvelimella. (3.)

JavaScript julkaistiin toukokuussa 1995 ja sen kehitti alun perin Netscapen Brendan Eich. Alun perin JavaScript julkaistiin nimellä Mocha, myöhemmin se nimettiin LiveScriptiksi, ja lopulta nimi muuttui JavaScriptiksi, kun Netscape yhdistyi Sun Microsystemsin kanssa. Sun Microsystems on kehittänyt Java-ohjelmointikielen. JavaScript ja Java ovat kuitenkin eroavia tekniikoita, eikä niillä ole tekemistä toistensa kanssa. JavaScriptin suosion ansiosta Microsoft kehitti yhteensopivan version vuonna 1996 nimeltä JScript. (2, s. 5.)

Netscape esitteli joulukuussa 1995 JavaScriptin julkaisun jälkeen mahdollisuuden käyttää JavaScriptiä myös palvelinpuolen skripteissä. Myöhemmin julkais-

tiin Node.js, jolla voidaan käyttää JavaScriptiä palvelinpuolella tehokkaammin. (4.)

### 2.3 NPM

NPM eli Node Package Manager on Node.js:lle kehitetty pakettimanageri, joka hallitsee projektin riippuvuudet, asennukset ja päivittämisen. NPM on säilö avoimen lähdekoodin Node.js-projekteille ja lisäksi komentorivityökalu, jonka avulla voidaan asentaa paketteja. NPM:stä löytyy paljon Node.js-kirjastoja ja ohjelmia. Uusia paketteja ja kirjastoja julkaistaan päivittäin ja niitä voi etsiä osoitteesta http://search.npmjs.org/. Halutut paketit voidaan asentaa helposti yhdellä komentorivikomennolla. Asennettavat paketit voivat vaatia myös muita julkaistuja paketteja, NPM asentaa tarvittavat paketit automaattisesti. (5.)

Esimerkiksi Electron-sovelluskehyksen asentaminen projektikansioon tapahtuu yksinkertaisesti komennolla ”npm install electron-prebuilt”.

### 2.4 Electron

Electron on GitHubin kehittämä järjestelmäriippumaton sovelluskehys. Ensimmäinen versio Electronista julkaistiin 15. heinäkuuta 2013. Alun perin Electronia kutsuttiin Atom Shelliksi. (7.)

Electronilla voidaan kehittää Node.js:ää käyttäen alustariippumattomia työpöytäsovelluksia. Sillä on kehitetty esimerkiksi Atom-tekstieditori ja Visual Studio Code. (8.)

Electron sisältää monia ominaisuuksia, jotka helpottavat sovelluksen kehittämistä. Siihen saa automaattiset päivitykset, jotka toimivat macOS-, Windows- ja Linux-käyttöjärjestelmissä. Se tukee suoraan käyttöjärjestelmän natiiveja valikoita ja ilmoituksia. Sovellusten julkaiseminen on helppoa, koska se sisältää myös asennusohjelmat Windowsille. Electron käyttää ytimenä Chromium-verkkoselainta. Sovellusten käyttöliittymän voi toteuttaa esimerkiksi HTML-kuvauskielellä. Electronilla perussovellus muodostuu kolmesta tiedostosta.

**package.json**

Ensimmäinen sovelluksen perustiedostoista on package.json, joka on metatieto-tiedosto. Tähän tiedostoon sisällytetään kaikki tarvittavat tiedot sovelluksen kehittämiseen. Kuvassa 2 projektin package.json-tiedosto. Tämä tiedosto voi sisältää esimerkiksi:

- Ohjelman version
- Kuvaus ohjelmasta
- Avainsanat
- Sovelluksen kehittäjät
- Sovelluksen riippuvuudet ja riippuvuuksien tietolähteet (9.)


*KUVA 2. package.json-tiedosto*

**Index.html**

Toinen sovelluksen perustiedostoista on index.html, jolla toteutetaan sovellukseen graafinen käyttöliittymä. Index.html-tiedosto ladataan oletuksena HTTP-palvelimen osoitetta kutsuttaessa. Electron-sovelluksessa index.js-tiedostossa määritellään index.html-tiedosto ladattavaksi näkymään.

### index.js

Kolmas sovelluksen perustiedostoista on index.js Tähän tiedostoon voidaan tehdä sovelluksen sisältämät toiminallisuudet. Index.js-tiedosto ladataan Node.js sovelluksessa aluksi package.json-tiedoston tarkistamisen jälkeen. Node.js-sovelluksessa tiedostoa ei ole pakollista nimetä index.js:ksi, mutta se helpottaa koodin tarkkailussa ymmärtämään, mikä tiedosto ajetaan ensimmäiseksi.

## 2.5 WebSocket

WebSocket on kaksisuuntainen (full-duplex) yhteys, joka mahdollistaa kommunikoinnin verkkoselaimen ja palvelimen välillä. WebSocket yhteyden luomisen jälkeen yhteys pysyy auki, kunnes käyttäjä tai palvelin katkaisee yhteyden. Avoimella yhteydellä käyttäjä ja palvelin voivat lähettää viestejä molempiin suuntiin- Tämä mahdollistaa tapahtumapohjaisen verkkosovelluskehittämisen. (10.)

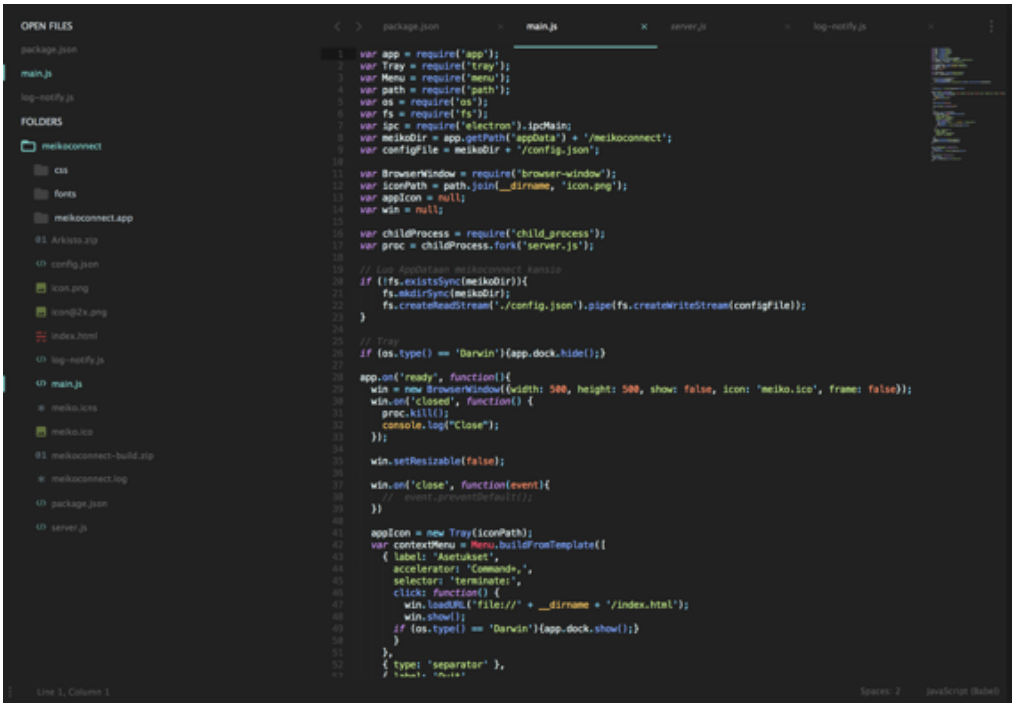
## 2.6 TCP

TCP (Transmission Control Protocol) on tietoliikenneprotokolla, jolla luodaan yhteys laitteiden välillä. TCP toimii yhdessä IP:n (Internet Protocol) kanssa. Sillä määritellään, kuinka laitteet lähettävät datapaketteja toisten laitteiden välillä. Yhteys muodostetaan eri laitteiden välillä käyttäen IP-osoitetta ja porttia. TCP-protokollan päälle on rakennettu esimerkiksi HTTP-, SMTP-, Telnet-, SSH-, FTP- ja WebSocket-protokollat. Protokollat toimivat porttinumeron perusteella.

## 2.7 Käytetyt työkalut

### 2.7.1 Sublime Text 3

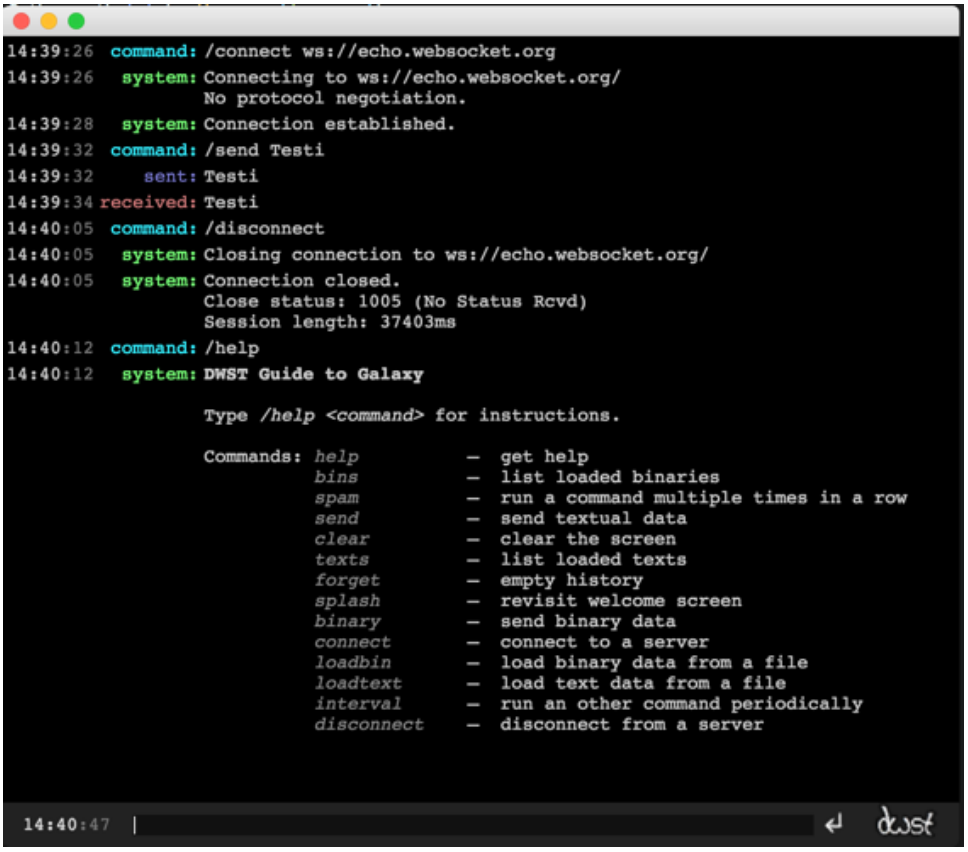
Sublime Text on alustariippumaton tekstieditori, joka tukee natiivisti useita eri ohjelmointikieliä, ja sitä pystyy laajentamaan erilaisilla laajennuksilla. Sublime toimii nopeasti isojenkin projektien kanssa ja se sisältää paljon hyödyllisiä ominaisuuksia kehittämiseen. Opinnäytetyöprojektin toteuttamiseen Sublime Text 3 soveltui hyvin, enkä tarvinnut muita editoreita lisäksi. Kuvassa 3 on Sublime Text 3 -editorin käyttöliittymä.




*KUVA 3. Sublime Text 3 -käyttöliittymä*

### 2.7.2 Dark WebSocket Terminal

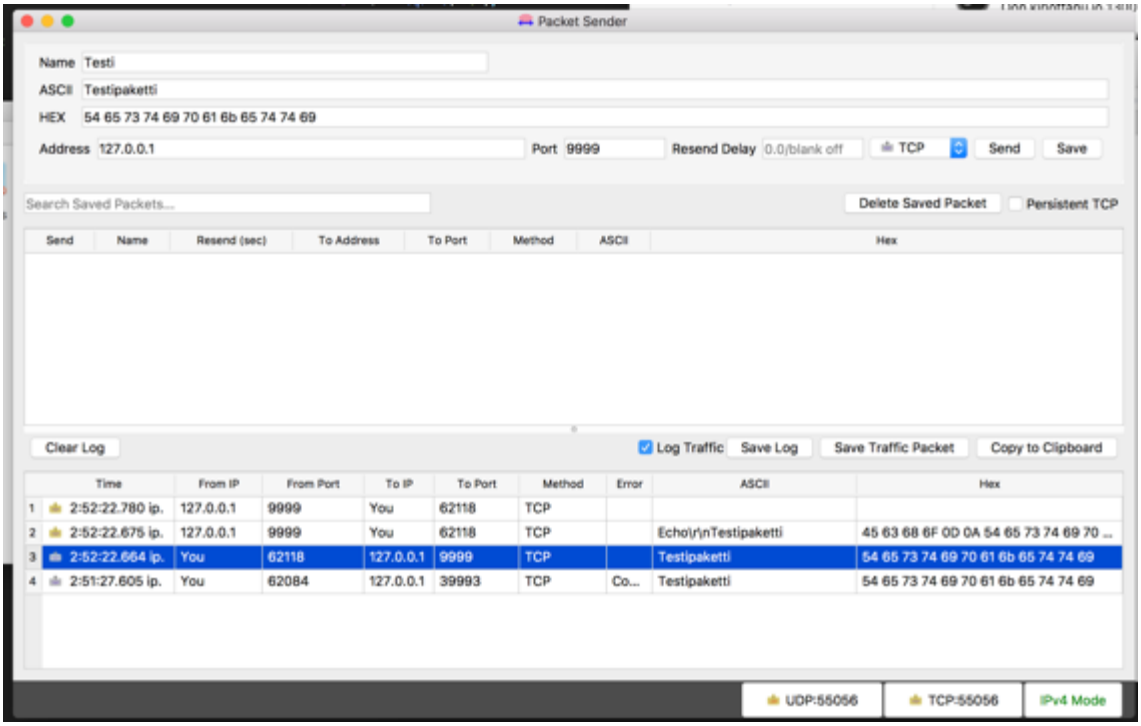
DWST on pieni komentorivisovellus, jolla pystyy testaamaan WebSocket-palvelimen toimivuutta. Sovellus on ladattavissa ilmaiseksi Chrome-selaimen kaupasta. Sovellus toimii Chrome-selaimen lisäosana. Kuvassa 4 näkyy WebSocket-viestien lähettäminen DWST-sovelluksella.




*KUVA 4. DWST-WebSocket-ohjelma*

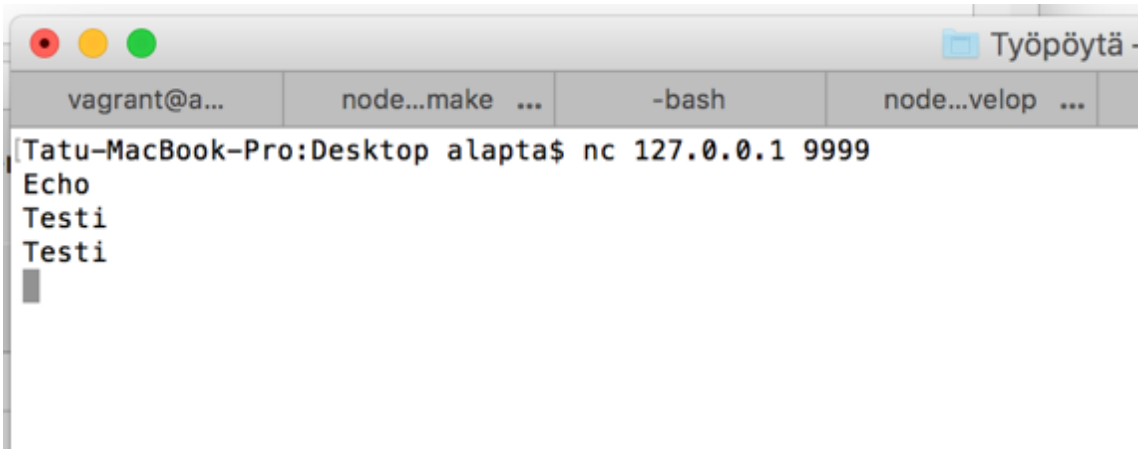
### 2.7.3 PacketSender ja Netcat

PacketSender on alustariippumaton avoimen lähdekoodin sovellus TCP/UDP-yhteyksien testaamiseen. Sovelluksella voi lähettää viestit ASCII- tai HEX-muodossa. PacketSender-sovellus sisältää myös komentorivilaajennoksen. Itse käytin kuitenkin yhteyksien testaamiseen graafista käyttöliittymää. Lisäksi käytin myös Unix-ympäristöistä tuttua Netcat (nc) -verkkotyökalua TCP-yhteyksien luomiseen. Kuvassa 5 näkyy TCP-viestin lähettäminen Packet Senderillä.




*KUVA 5. PacketSender-käyttöliittymä*

Saman viestin lähettäminen komentorivin kautta käyttäen Netcatia tapahtuu yhdistämällä ensin palvelimeen komennolla: ”nc ip-osoite portti”. Yhteyden muodostamisen jälkeen voi lähettää viestejä yhdistetylle palvelimelle. (Kuva 6.)




*KUVA 6. TCP-yhteyden muodostaminen käyttäen Netcatia*

# 3 SOVELLUKSEN TOTEUTUS

Projektin toimeksiantajana toimi Meiko Oy -niminen yritys Jyväskylästä. Sovelluksen toteuttamiseen oli vaatimuksena, että sovelluksen avulla verkkoselaimessa toimiva kassajärjestelmä pystyisi kommunikoimaan lähiverkossa toimivan maksupäätteen kanssa. Sovelluksen avulla ohjataan selaimesta tulevat WebSocket-viestit TCP:n kautta maksupäätteelle ja palautetaan TCP-viesti takaisin WebSocketin kautta kassajärjestelmälle selaimeen. Sovelluksen pitäisi toimia taustalla, eikä vaatia käyttäjältä erillisiä toimenpiteitä. Työpaikalla oli ajatuksena, että työ toteutettaisiin JavaScript-ohjelmointikielellä. Työkalut ja tavan sovelluksen toteutukseen sain kuitenkin itse valita.

Päätin toteuttaa sovelluksen JavaScriptillä ja Node.js:llä, vaikkei minulla ollutkaan sen kummempaa kokemusta työskentelystä kyseisistä tekniikoista. Sovelluskehyksiä vertaillessa jäi vaihtoehdoiksi kaksi samantyylistä kehystä: Electron ja NW.js.

Työkalujen valinnan jälkeen aloitin tutustumalla Node.js:n ja JavaScriptin toimintaan. Node.js avulla TCP- ja WebSocket-palvelimien toteuttaminen oli melko yksinkertaista. Aloitin toteuttamalla erikseen toimivat palvelimet ja Electron-sovelluksen.

## 3.1 WebSocket-palvelin

Ensimmäisenä vaiheena toteutin Node.js:llä WebSocket-palvelimen. WebSocket-palvelimen toteuttaminen oli melko yksinkertaista Node.js:n valmiiden kirjastojen avulla. Yksinkertaisuudessaan Websocket-palvelimen voi toteuttaa muutamalla rivillä koodia, käyttäen ws-kirjastoa. Koodin toimintaa voidaan testata ajamalla se komentorivissä komennolla ”node websocket.js”. Websocket-palvelin kuuntelee porttia 8080, ja palauttaa yhdistämisen jälkeen käyttäjälle ilmoituksen ”Yhdistetty”. (Kuva 7.) Viestin lähettämisen jälkeen sovellus palauttaa käyttäjälle lähetetyn viestin.

```
var WebSocketServer = require('ws').Server
  , wss = new WebSocketServer({ port: 8080 });

wss.on('connection', function connection(ws) {
  ws.on('message', function incoming(message) {
    console.log('Vastaanotettu: %s', message);
  });

  ws.send('Yhdistetty');
});
```

*KUVA 7. WebSocket-palvelin*

### 3.2 TCP-palvelin

Seuraavaksi tutustuin Node.js:n net-kirjastoon, joka sisältää tarvittavat asiat TCP-palvelimen toteuttamiseen. Yksinkertaisuudessaan sovelluksen saa ottamaan vastaan paketteja muutamalla rivillä koodilla. Kuvan 8 esimerkissä luodaan aluksi palvelin, joka kuuntelee koneen paikallisverkon portista 9999 tulevia paketteja. Ensimmäisenä kun asiakasohjelma yhdistää palvelimeen, se palauttaa käyttäjälle viestin ”Yhdistetty”. Tämän jälkeen se kuuntelee asiakasohjelman lähettämiä viestejä ja palauttaa ne takaisin.

tcp.js

```
// Server

var net = require('net');

var server = net.createServer(function(socket) {
    socket.write('Yhdistetty\r\n');
    socket.pipe(socket);
});

server.listen(9999, '127.0.0.1');
```

*KUVA 8. TCP-palvelin ja toiminta Netcatissa*

### 3.3 Electron-sovelluksen luominen

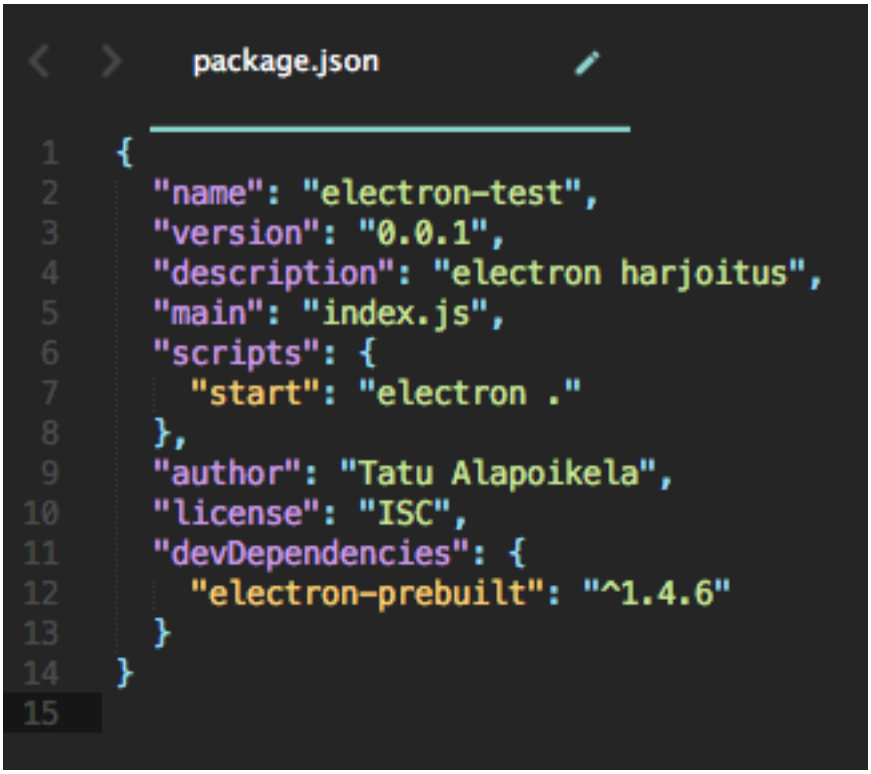
Electron-sovelluksen kehittäminen aloitetaan määrittelemällä projekti npm:n avulla, käyttäen komentoa ”npm init”, jossa lisätään sovelluksen tiedot ja määritykset package.json-tiedostoon. Tämän jälkeen asennetaan npm:llä Electron-paketti komennolla ”npm i electron-prebuilt --save-dev”. (Kuva 9.)

```
Tatu-MBP:Opinnaytetyo alapta$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults
.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (Opinnaytetyo) electron-test
version: (1.0.0) 0.0.1
description: electron harjoitus
entry point: (index.js)
test command:
git repository:
keywords:
author: Tatu Alapoikela
license: (ISC)
About to write to /Users/alapta/Documents/Opinnaytetyo/package.json:

{
  "name": "electron-test",
  "version": "0.0.1",
  "description": "electron harjoitus",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Tatu Alapoikela",
  "license": "ISC"
}


Is this ok? (yes) yes
Tatu-MBP:Opinnaytetyo alapta$ npm i electron-prebuilt --save-dev

> electron-prebuilt@1.4.6 postinstall /Users/alapta/Documents/Opinnaytetyo
/node_modules/electron-prebuilt
```

*KUVA 9. Electron-sovelluksen alkumäärittelyt*

### 3.4 Package.json

Package.json-tiedostoon määritellään sovelluksen tiedot, kuten nimi, versionumero ja ensimmäisenä ladattava tiedosto. Sovelluksen päätiedostona toimii index.js, joka määritellään main-tagilla. Seuraavaksi lisätään start-skripti, jonka perusteella sovelluksen käynnistyessä ajetaan määritelty komento. Start-skriptiin kirjoitetaan electron ja sovelluksen käynnistyskohta. (Kuva 10.) Electron ladataan node_modules-kansiosta.

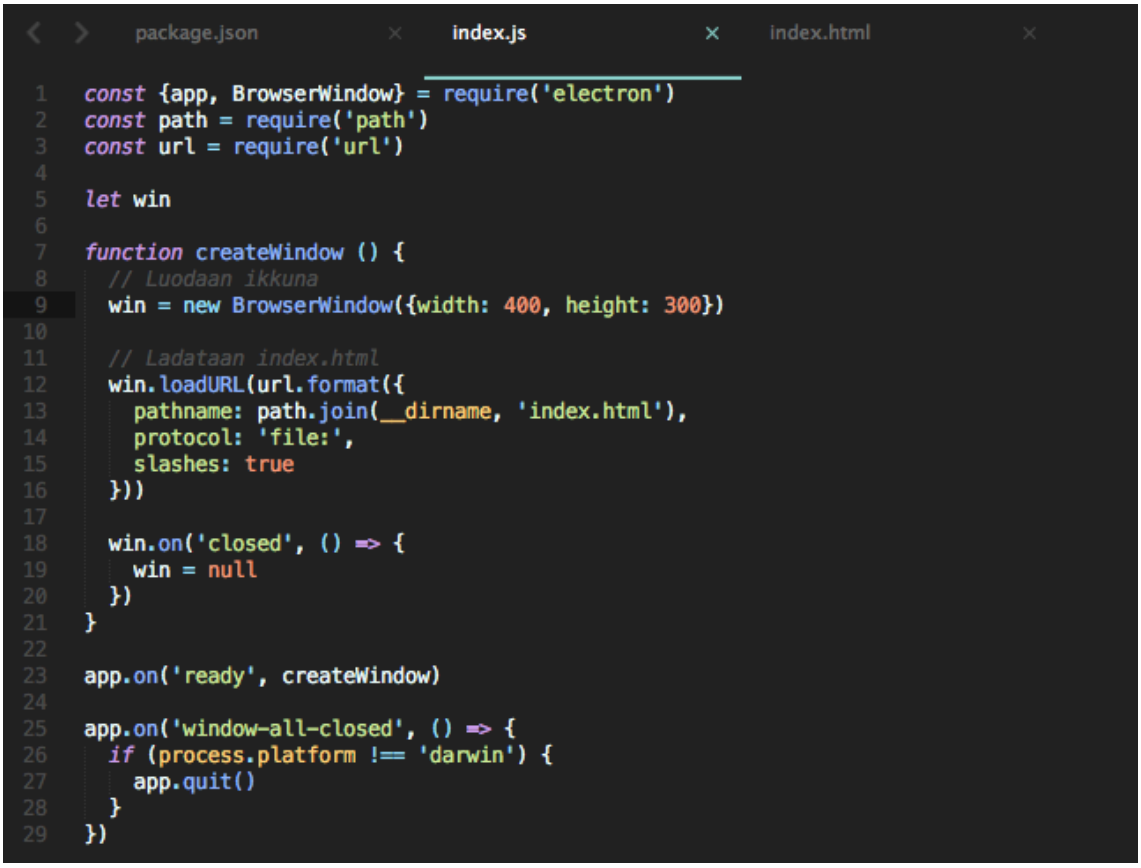



*KUVA 10. package.json-tiedosto*

### 3.5 Sovelluksen rakenne

Electron-sovelluksella ei ole pakollista kansiorakennetta, vaan tiedostot voidaan ladata halutuista paikoista. Electronin asentamisen jälkeen projektiin pitää lisätä myös tiedostot toiminnallisuutta ja ulkoasua varten. Aluksi luodaan index.js- ja index.html-tiedostot.

Index.js-tiedosto ladataan sovelluksen käynnistyessä ja siihen kirjoitetaan mitä sovellus tekee. Ensin ladataan sovelluksen riippuvuudet ja määritellään käytettävät muuttujat. Tämän jälkeen luodaan ikkuna, johon ladataan index.html-tiedosto, joka sisältää mitä ikkunassa näytetään. (Kuva 11.)

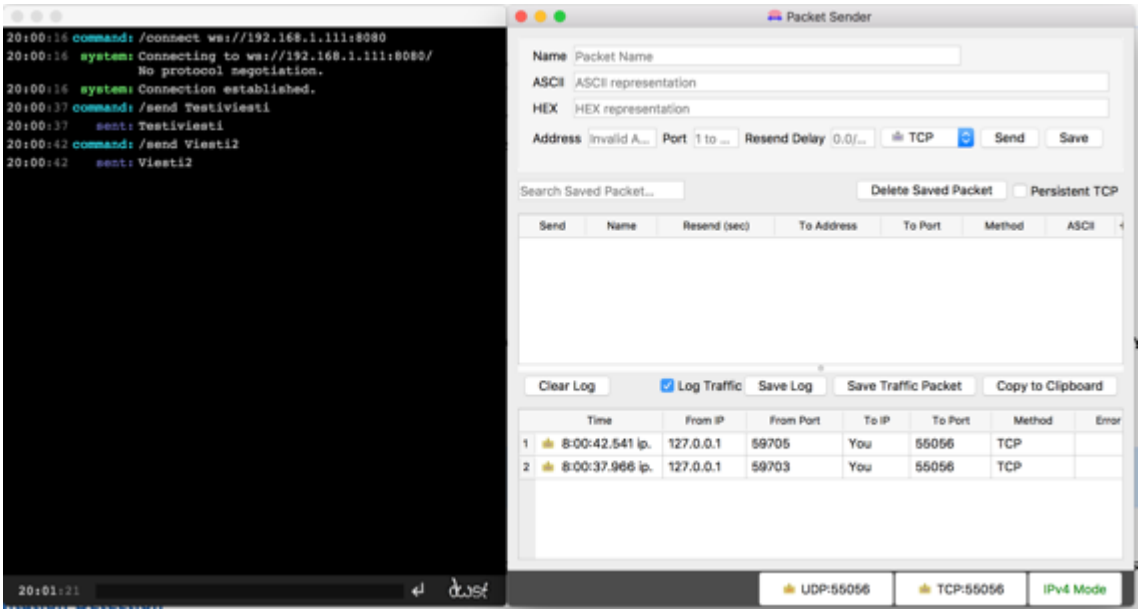



*KUVA 11. Index.js-tiedosto*

Index.html-tiedosto ladataan ikkunan käynnistyessä ja ikkunan sisältö tulee tämän tiedoston perusteella.

### 3.6 WebSocket-viestien ohjaaminen TCP:lle

Aluksi loin sovellukseen WebSocket-palvelun käyttäen nodejs-websocket-pakettia. WebSocket-palvelun sisällä pyörii TCP-palvelu, joka hyödyntää Noden.js:n net-pakettia. WebSocket-palvelin kuuntelee selaimelta tulevia viestejä ja kerää ne jonoon ja lähettää ne yhdistyessä TCP:llä asetustiedostossa määrättyyn osoitteeseen. Samanaikaisesti WebSocket-palvelun sisällä TCP-palvelu kuuntelee vastaantulevia TCP-viestejä ja ohjaa ne takaisin Websocketille. Yhteys muodostetaan DWST-työkalua käyttäen komennolla ”/connect ws://ip-osoite:portti”. Viestin lähettämisen jälkeen viesti näkyy Packet Sender- sovelluksessa, joka ottaa vastaan TCP-viestit. (Kuva 12.)




Kuva 12. *Dark WebSocket Terminal ja PacketSender*

Käynnistyessään WebSocket-palvelu odottaa, kunnes TCP-yhteys on muodostettu ja laittaa sovelluksen yhdistetty-tilaan ja ohjaa osoitteeseen saapuvat WebSocket-viestit jonosta TCP:lle. (Kuva 13.)

```
function send(message) {
    if(message)buffer.push(message);

    if (connected) {
        for (var i in buffer) {
            client.write(buffer[i]);
        }
        buffer = [];
        return true;
    }
    else {
        return false;
    }
}

function connect() {
  client.connect(config.tcpPort, config.tcpIP, function() {
    connected = true;
    send();
  });
}
```

KUVA 13. *Viestien lähetys ja yhteyden kuunteleminen*

Sovellus seuraa, jos yhteyttä ei jostain syystä saada muodostettua. Sovellus luo virhetilanteista lokitiedoston ja palauttaa käyttäjälle virheilmoituksen. Jos maksupäätteeseen yhdistämisessä tulee virhe, sovellus katkaisee myös Websocket-yhteden selaimessa, joten viestejä ei voi lähettää, ennen kuin yhteys on muodostettu uudelleen. Sovelluksen virheilmoitukset hoidetaan log-notify.js-tiedoston kautta. (Kuva 14.)


KUVA 14. *Virheiden seuraaminen ja virheilmoitus*

### 3.7 Virheilmoitukset

Sovellus huomaa, jos jostain syystä selaimen kautta ei pystytä ottamaan yhteyttä maksupäätteeseen ja ilmoittaa siitä käyttäjälle. Sovellus tekee lisäksi merkinnän lokitiedostoon. Virheilmoitukset käyttäjälle näytetään käyttäen käyttöjärjestelmän ilmoituskeskusta. Ilmoitukset sovelluksessa on toteutettu node-notifier-paketin avulla. Lokitiedosto kirjoitetaan käyttöjärjestelmän AppData-kansioon. Kyseinen kansio sisältää myös sovelluksen asetustiedostot. Virheilmoituksia varten sovelluksessa luodaan käyttöjärjestelmäkohtaisesti sovellukselle kansio järjestelmän AppData-kansioon. (Kuva 15.)

```
var fs = require('fs');
var os = require('os');
var util = require('util');
var path = require('path');

var log_stdout = process.stdout;
const notifier = require('node-notifier');

if (os.platform() == 'win32'){
  var log_file = fs.createWriteStream(process.env.APPDATA + '\\meikoconnect\\meikoconnect.log', {flags : 'a'});
}
else{
  var log_file = fs.createWriteStream(process.env.HOME + '/Library/Application Support/meikoconnect/meikoconnect.log', {flags : 'a'});
}

function logNotify(d, m) { //
  log_file.write(new Date().toISOString()+ ':\n')
  log_file.write(util.format(d) + '\n\n');
  log_stdout.write(util.format(d) + '\n\n');

  notifier.notify({
    title: "Meiko Connect",
    message: m,
    icon: path.join(__dirname, 'icon@2x.png'),
  });
};

exports.logNotify = logNotify;
```

KUVA 15. *Virheilmoitusten näyttäminen ja kirjaaminen lokiin.*

### 3.8 Asetukset

Sovelluksen asetukset ladataan config.json-tiedostosta, joka sijaitsee käyttöjärjestelmän AppData-kansiossa. Tiedostossa säilytetään tiedot maksupäätteen WebSocket-portista, IP-osoitteesta ja TCP-portista. Lisäksi sovelluksessa on myös oletusasetukset tiedoille. Asetustietoja voi päivittää sovelluksen asetusvalikosta. (Kuva 16.)


KUVA 16. *Asetusvalikko ja config.json-tiedosto*

Asetusvalikon ulkoasu tulee index.html tiedostosta, joka ladataan piilotettuna sovelluksen käynnistyessä. Asetusvalikko avataan käyttöjärjestelmän valikkopalkissa. Sovelluksessa ei ole itsessään muuta käyttöliittymää kuin asetusvalikko, kuvake valikkopalkissa ja virheilmoitukset. Sovelluksen käynnistyessä luodaan asetusikkuna piilotettuna. (Kuva 17.) Käyttöjärjestelmän sovellusvalikon kuvakkeesta avattaessa asetukset, ikkunan tila vaihdetaan näkyväksi.

```
// Tray
if (os.type() == 'Darwin'){app.dock.hide();}

app.on('ready', function(){
  win = new BrowserWindow({width: 500, height: 500, show: false, icon: 'meiko.ico', frame: false});
  win.on('closed', function() {
    proc.kill();
    console.log("Close");
  });

  win.setResizable(false);
```

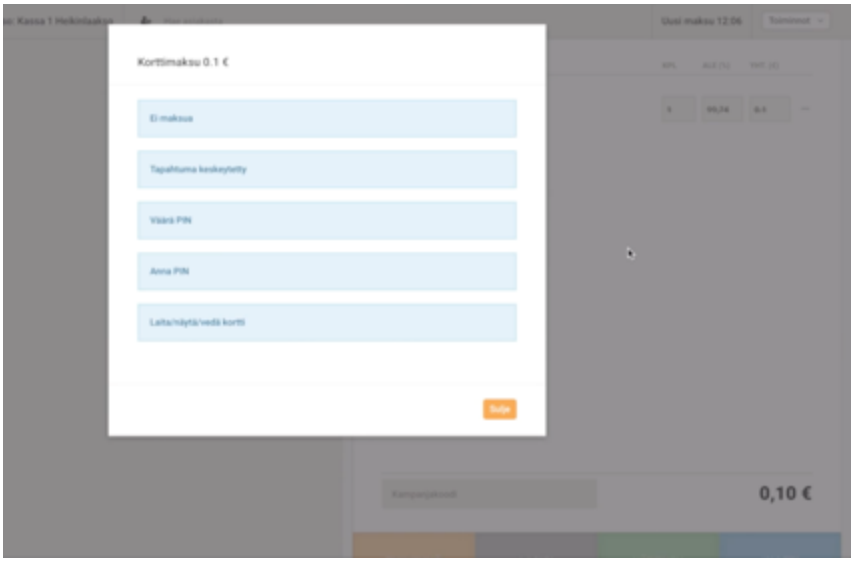
KUVA 17. *main.js-tiedosto ja ikkunan luonti*

### 3.9 Toimivuuden testaaminen

Sovellusta kehittäessä testasin sovelluksen toimivuutta käyttäen Dark WebSocket Terminalia, PacketSenderiä ja netcatia. Näillä työkaluilla pystyin testaamaan, että sovellus ohjasi WebSocketilta lähtevät viestit TCP:lle. Toiminallisuuden varmistuttua kävin Jyväskylässä Meiko Oy:n toimistolla kokeilemassa sovelluksen toimintaa maksupäätteen ja oikean kassasovelluksen kanssa. Aluksi käynnistettiin kassasovellus ja kassan asetuksissa määriteltiin maksupäätteen osoitteeksi luodun MeikoConnect-sovelluksen tiedot. Kassasovelluksesta lähtevät maksupyynnöt tulivat maksupäätteelle ja maksun pystyi hoitamaan. (Kuva 18.)


KUVA 18. *Testausta Yomani-maksupäätteellä*

Selaimessa kassasovellus saa maksupäätteeltä tiedot maksun tilasta. Maksupääte palauttaa kassajärjestelmälle tiedot PIN-koodin syötöstä ja maksun onnistumisesta. (Kuva 19.)




KUVA 19. *Maksutilanne kassajärjestelmässä*

# 4 JATKOKEHITYS

Sovellus toimii tällä hetkellä vaatimusten mukaisesti, niin että maksupäätteliikenne toimii. Asetusvalikon ulkoasua pitää päivittää myöhemmin myös. Käyttöliittymän toteuttamiseen voisi käyttää React-desktopia, joka tukee suoraan käyttöjärjestelmän natiivikomponentteja, jolloin sovelluksesta saisi yhtenäisen näköisen käyttöjärjestelmän kanssa. Tulevaisuudessa sovellukseen on tarkoituksena lisätä myös mahdollisuus tulostamisen ohjaamiselle, niin että sovellus ohjaisi automaattisesti kuittien tulostamisen määriteltyyn kuittitulostimeen.

# 5 YHTEENVETO

Opinnäytetyön tavoitteena oli luoda Meiko Oy:lle sovellus, jonka avulla voi käyttää yhtä maksupäätettä useammalla laitteella. Toimeksiantajalla oli alkutilanteessa käytössä verkkoselaimessa toimiva kassajärjestelmä ja maksupäätteen kanssa kommunikoitiin hyödyntäen Google Chromelle kehitetyn lisäosan avulla. Itselläni ei ollut aiempaa kokemusta tällaisen kokonaisuuden toteuttamisesta, mutta tutkimalla mahdollisia tekniikoita toimiva sovellus tuli valmiiksi.

Työn toteuttaminen oli mielenkiintoista ja samalla oppi paljon JavaScriptillä kehittämisestä. Projektin tein etänä kotoa ja työn etenemistä seurattiin viikoittaisilla nettipalavereilla Skypeä käyttäen. Palavereissa kävimme läpi työn tilanteen ja mitä seuraavina päivinä olisi tarkoitus tehdä.

# LÄHTEET


1. Node.js. 2016. Wikipedia. Saatavissa: https://en.wikipedia.org/wiki/Node.js. Hakupäivä 17.5.2016.

2. Negrino, Tom – Smith, Dori. 2007. Javascript - Tehokas hallinta.

3. Introduction of JavaScript. 2016. Mozilla Developer Network. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction Hakupäivä 8.6.2016.

4. Server-Side Javascript: Back With a Vengeance. 2009. Readwrite. Saatavissa: http://readwrite.com/2009/12/17/server-side_javascript_back_with_a_vengeance/. Hakupäivä 8.6.2016.

5. Reed, Nico. 2011. What is npm?. Nodejitsu. Saatavissa: https://docs.nodejitsu.com/articles/getting-started/npm/what-is-npm/. Hakupäivä 18.5.2016.

6. Dayley, Brad. 2014. Node.js, MongoDB and AngularJS Web Development. New Jersey: Addison-Wesley.

7. Sawicki, Kevin. 2015. Atom Shell is now Electron. Atom.io. Saatavissa: http://electron.atom.io/blog/2015/04/23/electron Hakupäivä 18.5.2016.

8. Electron. 2016. Atom.io. Saatavissa: http://electron.atom.io/. Hakupäivä 19.10.2016.

9. Package.json. 2016. Npmjs. Saatavissa: https://docs.npmjs.com/files/package.json. Hakuäivä 19.10.16.

10. Introduction to WebSockets. 2016. socketo.me. Saatavissa: http://socketo.me/docs. Hakupäivä 20.10.16.