TAMPEREEN
AMMATTIKORKEAKOULU

MITTAUSARVOJEN VÄLITTÄMINEN ETÄHALLINTAKÄYTTÖLIITTYMÄÄN

Miika Ahvenjärvi

Opinnäytetyö
Marraskuu 2015
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

AHVENJÄRVI MIIKA:
Mittausarvojen välittäminen etähallintakäyttöliittymään

Opinnäytetyö 32 sivua
Marraskuu 2015

Tavoitteena oli suunnitella ja toteuttaa mittausnäkymä WRM 247-etähallintalaitteen lokaaliin konfigurointikäyttöliittymään ja samalla perehtyä prosessien välisen kommunikaation eri toteutustapoihin. Mittausnäkymästä oli tarkoitus nähdä laitteen viimeisimmät mittausarvot jokaiselle siihen konfiguroidulle datanoodille.

Laiteohjelmistoon ohjelmoitiin oma lohkonsa terminaalisovellukseen ja REST-pohjaiseen rajapintaan selaimelta tulevia pyyntöjä hoitava lohko. Näiden välille tehtiin IPC-toteutus, jossa terminaalisovelluksen moduuli hakee selaimelta tulleiden pyyntöjen perusteella halutut tiedot ja lähettää ne takaisin rajapinnan kautta selaimelle. Käyttöliittymäpäähän toteutettiin lohko, jossa päivitetään mittausarvoja piirtävää kuvaajaa laitteelta saatujen tietojen perusteella, sekä hoidetaan Ajaxin avulla pyynnöt serverille. Työlle asetetut tavoitteet täytettiin, ja lopputuloksena oli toimiva uusi ominaisuus etähallintalaitteelle.

Asiasanat: etähallinta, IPC, käyttöliittymä

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Embedded Systems and Electronics

AHVENJÄRVI MIIKA:
Transmission of Measurement Data to Remote Management Interface

Bachelor's thesis 32 pages
November 2015

The objective of this bachelor’s thesis was to design and implement a measurements view for local configuration interface of WRM 247-remote control device, and at the same time study different mechanisms for the inter process communication. The purpose of the measurements view was to see the latest measurements data for each datanode configured to the device.

For the device firmware, two separate sections were programmed: a section for the terminal application and a section for the REST-based interface, which handles requests from the browser. An IPC-implementation was done between these two modules, in which the module in the terminal application retrieves the data based on request, and sends it back to the browser via the REST-interface. For the front-end, a section was made to handle the requests to server, and to update a graph that draws the latest measurement values from the device. The goals set for the thesis were met, and the end result was a working new feature for the remote control device.

Key words: remote management, IPC, interface

SISÄLLYS

LYHENTEET JA TERMIT

Ajax	Yhdistelmä eri teknologioita tiedonsiirtoon
AMQP	Advanced Message Queuing Protocol, avoimen standardin protokolla väliohjelmistoille
Big Data	Suuri määrä dataa, joka on muodoltaan vaihtelevaa (ei vakiintunutta määritelmää)
CGI	Common Gateway Interface, tekniikka, jolla välitetään dataa selaimelta palvelimelle
CSS	Cascading Style Sheets, tyyliohjeet
DOM	Document Object Model, sopimus HTML, XHTML ja XML-dokumenttien rakenteen mallinnukseen
FIFO	first in, first out –periaate
HTML	Hypertext Markup Language, merkintäkieli
IMAP	Internet Message Access Protocol, protokolla sähköpostien lukemiseen
IoT	Internet of Things, esineiden internet, kuvaa esineiden ja laitteiden liittymistä internettiin
IPC	Inter Process Communication, prosessien välinen kommunikaatio
IPv6	IPv4-kommunikaatioprotokollan seuraaja
JSON	JavaScript Object Notation, kevyt standardi tiedonsiirtoon
PGM	Pragmatic General Multicast, verkkoliikenteen monilähetysprotokolla
POP	Post Office Protocol, protokolla sähköpostin lukemiseen
REST	Representational State Transfer, arkkitehtuurityyli
SOAP	Simple Object Access Protocol, tietoliikenneprotokolla
SVG	Scalable Vector Graphics, kaksiulotteisten vektorikuvien kuvauskieli
TCP	Transmission Control Protocol, tietoliikenneprotokolla
UNIX	Laitteistoriippumaton käyttöjärjestelmä
URI	Uniform Resource Identifier, merkkijono resurssien paikantamiseen
WSDL	Web Service Description Language, XML-perustainen kieli

XML	Extensible Markup Language, rakenteellinen metakieli tiedon siirtoon ja tallentamiseen
XMLHttpRequest	Ohjelmointirajapinta HTML-pyyntöjen lähetykseen
ZMQ	ZeroMQ, kommunikaatiokirjasto

1 JOHDANTO

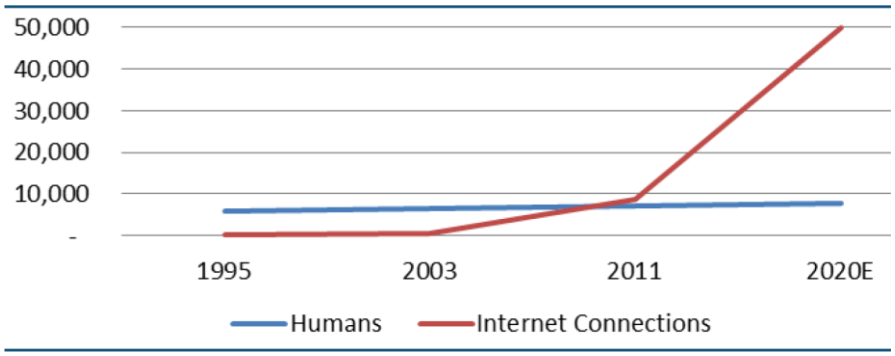
Tämän opinnäytetyön toimeksiantaja Wapice Oy on vuonna 1999 perustettu ohjelmistoalan yritys, joka keskittyy sulautettujen- ja teollisuusohjelmistojen kehitykseen, elektroniikkasuunnitteluun sekä liiketoimintaratkaisuihin. Wapice työllistää yli 270 ohjelmisto- ja elektroniikka-asiantuntijaa. Yrityksen pääkonttori sijaitsee Vaasassa ja muut yksiköt Vaasan Airport Parkissa, Tampereella, Hyvinkäällä, Seinäjoella, Oulussa ja Jyväskylässä. Työ tehtiin Wapicen kehittämälle WRM 247 -etähallintalaitteelle, joka toimii osana yrityksen IoT-Ticket-etähallintajärjestelmää.

Työn tarkoituksena on ohjelmoida WRM 247 -laitteelle uusi ominaisuus, joka antaa laitteen käyttäjille mahdollisuuden selata reaaliaikaisesti laitteelta saatavia mittaustietoja, kun verkkoyhteyttä etähallintajärjestelmän palvelimille ei ole muodostettu. Työn tavoitteena oli toteuttaa mittausnäkymä laitteen lokaaliin käyttöliittymään, sisältäen toteutuksen laitteen sisäisen REST-pohjaisen rajapinnan ja terminaalisovelluksen väliseen kommunikaatioon. Tähän tavoitteeseen sisältyi tutustuminen prosessien välisen kommunikoinnin eri toteutustapoihin sekä muihin työssä tarvittaviin teknologioihin. Työssä esitellään miten esiteltyjen tekniikoiden avulla luotiin tavoiteltu ominaisuus, millaisia ongelmia työssä ilmeni, miten ne ratkaistiin, ja pohditaan, mitä olisi voitu tehdä toisin.

2 IOT JA ETÄHALLINTAJÄRJESTELMÄT

2.1 Esineiden internet

Esineiden internet, IoT (Internet of Things) on termi, jolla kuvataan esineiden, laitteiden tai laajemmin, minkä tahansa fyysisten objektien, liittymistä verkkoon. Laitteiden sisäänrakennetut verkkovalmiudet mahdollistavat laitteiden välisen kommunikoinnin keskenään, yhteydet internet-palveluihin ja entistä laajemman vuorovaikutuksen käyttäjien kanssa. Laitteiden tila ja niiden keräämät tiedot voidaan tallentaa suoraan tietojärjestelmiin, missä kerättyä tietoa voidaan analysoida. Myös etähallinta internetin välityksellä on mahdollista. Internettiin yhteydessä olevien laitteiden määrä on kasvanut räjähdysmäisesti viime vuosina ja määrän uskotaan kasvavan vuoteen 2020 mennessä 26–50 miljardiin (Kuva 1). (Floerkemeier & Mattern 2010, 1–6; Bellini ym. 2014, 1–4)



Kuva 1. Ihmisten ja Internettiin yhteydessä olevien laitteiden määrä (miljoonia) (McCourt ym. 2014)

Esineiden internetin kasvuun on vaikuttanut useita eri tekijöitä, muun muassa keskeisen laitteistoelektroniikan (erityisesti sensorien ja prosessorien) ja ohjelmiston kehittyminen että halventuminen, internet-yhteyksien kustannusten lasku, sensoritekniikan kehitykset, entistä tehokkaammat ja kattavammat langattomat viestintätekniikat, siirtyminen IPv6:een, entistä suurempien tietomäärien tallennus, pilvilaskenta ja *Big datan* tuomat mahdollisuudet. Esineiden internetin kasvuennustukset perustuvat uskoon siitä, että elektroniikan sekä kommunikaatio- ja tietotekniikan kehitys tulee jatkumaan tasaisesti tulevaisuudessa. (Floerkemeier & Mattern 2010, 1; Bellini ym. 2014, 4).

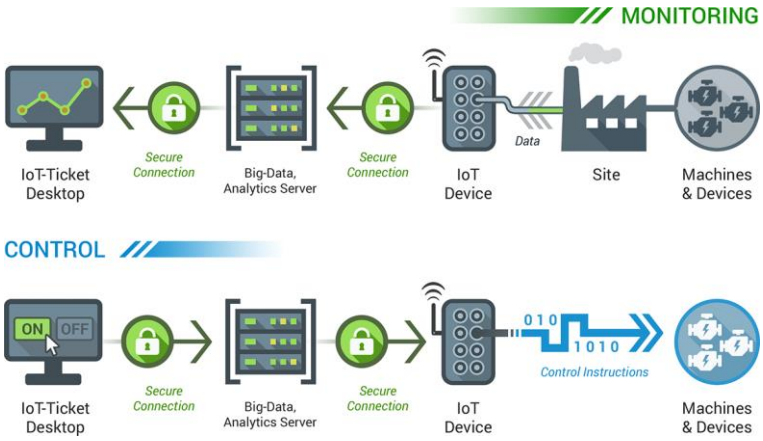
Evansin (2011, 4) mukaan IoT tulee parantamaan laiteverkkojen sisäisiä hallintamahdollisuuksia, tietoturvaa ja analytiikkaa. Sijoittamalla IoT-järjestelmiin yritykset voivat parantaa tuottavuuttaan ja tehdä säästöjä esimerkiksi käyttöomaisuusinvestoinneissa, työvoimassa ja energiakuluissa. Automaatiota teollisuudessa voidaan lisätä, kun IoT mahdollistaa esimerkiksi valvonnan jättämisen pääosin antureiden varaan. Yleisesti ottaen IoT:n mahdollisuudet motivoivat yrityksiä tuomaan verkkoyhteyksiä käytettäviä laitteita markkinoille – laitteita, jotka kykenevät tietopalveluihin tavanomaisten toimintojen ohella. Käyttökohteet vaihtelevat henkilökohtaisesta ja kotikäytöstä yritysten tarpeisiin sekä alueellisiin että valtiotason sovelluksiin (taulukko 1). (Kyriakopoulos ym. 2014, 4)

Taulukko 1. IoT:n käyttökohteita (Buyya, Gubbi, Marusic & Palaniswami 2013, 1649–1651; Wikipedia)

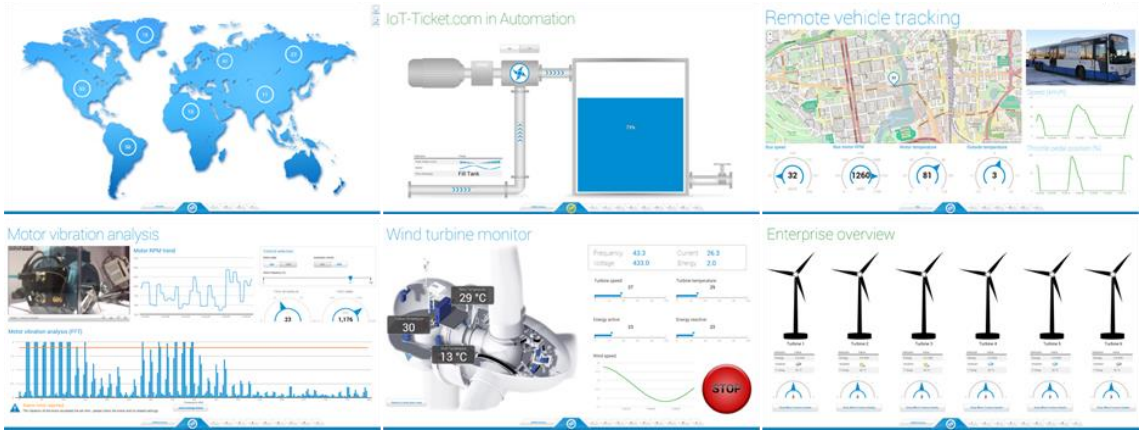
Energianhallinta	Älymittarit sähkövoimalle ja vedelle, älykäs sähköverkko, sähkön tuotannon ja kulutuksen hallinta
Liikenne	Kommunikoiva liikenne, älykkäämpi logistiikka, reitin optimointi
Maatalous	Datan keruu ja analytiikka
Tehdasautomaatio	Prosessien valvonta, optimointi, huolto
Hätätilannepalvelut	Hätätilannejärjestelmät, reagointisuunnitelmat, resurssienhallinta, väkijoukon seuranta
Terveydenhuolto	Resurssien hallinta, potilas- ja henkilöstöseuranta, taudin leviämisen mallinnus ja eristys
Tuotanto	Tehokas energianhallinta, hukka-ajan minimointi
Media	Mainosten kohdistaminen, datan keruu
Infrastruktuurin hallinta	Infrastruktuurin monitorointi ja etähallinta, korjaustöiden koordinointi, toimintakustannusten vähentäminen
Ympäristönseuranta	Luonnonkatastrofit, lämpötila, ilman- ja vedenlaatu, ilmakehän ja maaperän olosuhteet
Kodin automaatio	Valaistuksen, lämpötilan, ilmastoinnin ja ilmanvaihdon säätö, älykkäät kodinkoneet, energiansäästö, kodin turvallisuus

2.2 IoT-Ticket

Wapice Oy:n kehittämä IoT-Ticket (entiseltä nimeltään WRM, Wapice Remote Management) on kokonaisvaltainen etäseuranta ja -hallintajärjestelmä, joka mahdollistaa IoT-palvelujen käyttöönoton helposti ja edullisesti. Järjestelmään lähetetään dataa WRM 247-laitteelta, miltä tahansa IoT-Ticketiä tukevalta laitteelta tai tarjotun ohjelmistorajapinnan kautta. Järjestelmän toimintaa on havainnollistettu kuvassa 2. Kun järjestelmä on liitetty elektroniikkaan ja data on ladattu palveluun, voidaan sitä analysoida IoT-Ticketin analytiikan avulla. Järjestelmä mahdollistaa myös analytiikan Big datalle. Järjestelmä sisältää käyttöliittymäsuunnittelutyökalun ja toisen työkalun käyttöliittymän toimintojen määrittelyyn. Näiden työkalujen avulla on mahdollista luoda *Dashboardeja*, jotka voivat sisältää annettua dataa lukuisassa eri muodossa (kuva 3) ja komponentteja etähallintaan. IoT-Ticket sisältää myös raportointityökalun, jolla on mahdollista luoda muun muassa asennus-, kunto-, tulos-, vika- ja huoltoraportteja. (IoT-Ticket 2015)



Kuva 2. IoT-Ticket järjestelmän toiminta (IoT-Ticket 2015)



Kuva 3. IoT-Ticket Dashboard esimerkkejä (IoT-Ticket 2015)

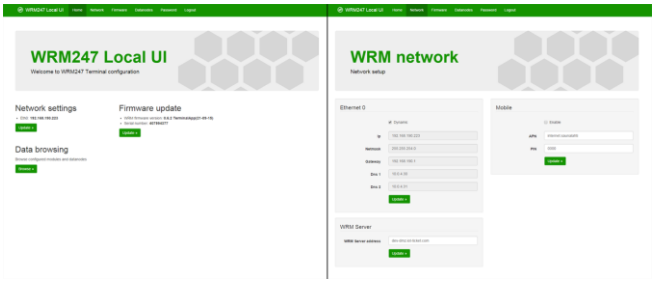
2.3 WRM 247 -laite

WRM 247 on Wapice Oy:n kehittämä etähallintalaite, joka pystyy mittaamaan ja kontrolloimaan siihen liitettyjä laitteita ja koneistoa. Mittaustiedot laite lähettää eteenpäin palvelimelle. Laite on suunniteltu kestämään teollisia olosuhteita ja sitä voidaan kustomoida asiakkaan tarpeisiin. Laitetta myydään ensisijaisesti osaksi IoT-Ticket-etähallintajärjestelmää, mutta myös sen itsenäinen käyttö on mahdollista.

Kuva 4. WRM 247+ (Wapice Oy 2015)

Laiteohjelmiston keskeisin osa on terminaalisovellus. Tämän lisäksi laite sisältää käytettävät kirjastot, skriptit ja valmiiksi käännetyt binäärit. Terminaalisovellus sisältää lukuisia eri ohjelmistomoduuleja, jotka ovat vastuussa eri toiminnoista kuten mittaustietojen luvusta, kirjoituksesta tai protokollatuesta. Jokainen moduuli voi pitää sisällään *datanoodeja*, jotka sisältävät dataa laitteen sisäisistä prosesseista kuten mittaustiedoista. Moduulit ja datanoodit konfiguroidaan suoraan palvelimelta konfiguraatiotiedostoon, joka lähetetään laitteelle, jolloin laite ottaa uuden konfiguraation käyttöönsä.

WRM 247 -laite sisältää lokaalin web-pohjaisen käyttöliittymän laitteen verkkoasetusten ja laiteohjelmiston muuttamiseen (kuva 5). Tähän käyttöliittymään pääsee käsiksi laitteen Ethernet-liitännän tai WLANin ja laitteen IP-osoitteen kautta. Käyttöliittymästä voidaan muuttaa laitteen verkkoasetuksia ja päivittää laiteohjelmisto. Osaksi käyttöliittymää oli kaavailtu mahdollisuutta seurata laitteen mittaustietoja kuvaajasta, mutta sitä ei opinnäytetyön aloittamishetkellä ollut toteutettu. Kyseisestä puuttuvasta ominaisuudesta kehittyi opinnäytetyön aihe. (Wapice Oy)



Kuva 5. WRM 247:n lokaali käyttöliittymä (Wapice Oy 2015)

3 TEKNOLOGIAT

3.1 Bootstrap

Bootstrap on monipuolinen, responsiivista suunnittelua tukeva koodikirjasto front-end-toteutuksille. Se sisältää monia käyttöliittymäsuunnitteluun soveltuvia komponentteja ja toiminnallisuuksia, nopeuttaen ja helpottaen kehitystyötä. Bootstrap on yhdistelmä HTML-rakenteita ja CSS-sunnittelumalleja sekä vapaavalintaisia JavaScript- sekä jQuery-funtioita. Bootstrapin kehittivät Mark Otto ja Jacob Thornton Twitterillä työskennellessään ja se julkaistiin vuonna 2011 GitHubissa avoimella lähdekoodilla. (Bootstrap 2015)

3.2 Javascript ja jQuery

JavaScript on järjestelmäriippumaton oliopohjainen ohjelmointikieli. Isäntä-ympäristönsä sisällä (esimerkiksi selaimen) JavaScriptilla voidaan kontrolloida tämän ympäristön objekteja ohjelmallisesti. JavaScript on mahdollistanut monia moderneja web-sovelluksia ja tehnyt mahdolliseksi vuorovaikutuksellisen internetin selauksen. Javascript julkaistiin vuonna 1995 uutena tapana lisätä ohjelmia nettisivuille Netscape Navigator -selaimessa. Sittemmin JavaScript on otettu mukaan jokaiseen merkittävään selaimeen ja se on kriittinen osa nykyaikaisia internet-sivustoja. (Flanagan 2011, 2; MDN 2015)

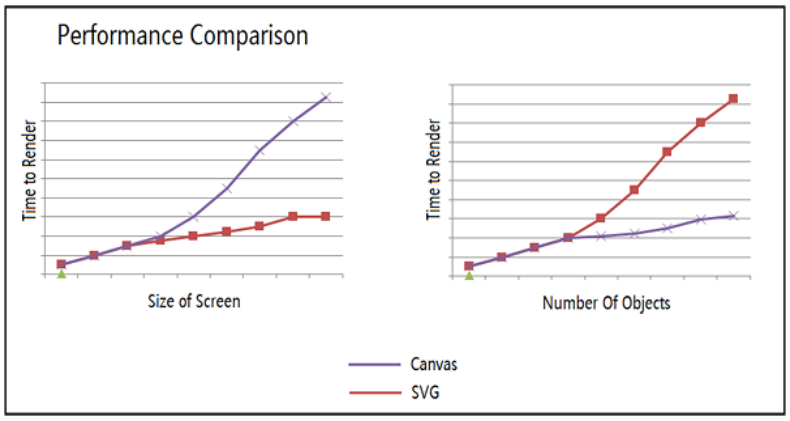
jQuery on järjestelmärippumaton avoimen lähdekoodin JavaScript-kirjasto, joka on suunniteltu yksinkertaistamaan monia JavaScriptillä ohjelmoitavia toimintoja kuten DOM-elementtien valintaa ja manipulointia, tapahtumakäsittelyä, animointia ja Ajax-sovellusten kehittämistä. jQuery poistaa selaintenväliset JavaScriptin yhteensopivuus-ongelmat ja sitä voidaan laajentaa liitännäisillä. jQuery on suosituin käytössä oleva JavaScript-kirjasto: sitä käyttää 65 % 10 miljoonasta eniten vieraillusta internet-sivustoista (W3Techs 2015). (jQuery Foundation 2015)

3.3 HTML5 Canvas ja SVG

HTML5 on vuonna 2014 julkaistu uusi versio HTML-merkintäkielestä (Hypertext Markup Language) ja se sisältää lukuisia uusia ominaisuuksia ja parannuksia, joista kuitenkin

rajoitutaan tarkastelemaan vain Canvas-elementtiä. Canvasin alkuperäinen kehittäjä oli Apple vuonna 2004. Myöhemmin sen standardisoi WHATWG (Web Hypertext Application Technology Working Group) ja se julkaistiin W3C:n patenttisopimuksen alla (Roberts, 2014). Canvas-elementti on itsessään matalan tason malli, joka päivittää sivulle sulautettua bittikarttaa – se toimii säiliönä *kontekstille* eli piirrettävälle grafiikalle. Canvasin 2D -ohjelmointirajapinta mahdollistaa eri muotojen piirtämisen, tekstin generoimisen ja kuvien näyttämisen elementin sisällä JavaScript-metodien avulla. Luotua grafiikkaa voidaan jälkeenpäin muokata monella eri tavalla. (Fulton 2011, 1–2)

SVG (Scalable Vector Graphics) on kaksiulotteisten vektorikuvien kuvauskieli. SVG on avoin ja laajasti tuettu formaatti, joka soveltuu erinomaisesti animaatioon sekä selaimen ja käyttäjän väliseen vuorovaikutukseen. Yleisesti käytetyistä selaimista vain Internet Explorer 8 ja vanhemmat Android-selaimet eivät tue SVG:tä. SVG-kuvien muokkaus on mahdollista JavaScript-kielellä. (W3 2011; CanIUse 2015)



Kuva 6. SVG:n ja HTML5 Canvas-elementin suorituskykyvertaus (Microsoft)

Suurin ero Canvasin ja SVG:n välillä on Canvasin rasteri-pohjaisuus ja SVG:n vektori-pohjaisuus. SVGkuvat määritellään XML-metakielellä (Extensible Markup Language), minkä seurauksena jokainen SVG-elementti liitetään DOM:iin (Document Object Model). Elementtejä voidaan muokata JavaScriptilla ja CSS:llä sekä niihin voidaan liittää tapahtumienkäsittelijöitä. Koska kaikkia SVG-elementtejä käsitellään DOM:in sisällä, kompleksiset grafiikat, joissa on paljon objekteja, häviävät nopeasti suorituskyvyssä Canvas-toteutukselle, joka ei vaadi selaimelta kuin pikseligrafiikan renderöimisen. SVG-elementit skaalautuvat vaivatta vaikuttamatta suorituskykyyn, toisin kuin Canvaksen kanssa, joka joutuu renderöimään enemmän pikseleitä näytölle, jolloin SVG:llä saadaan parempi suorituskyky korkearesoluutioisilla näytöillä, kunhan objektien määrä on rajoitettu (kuva

6). SVG sopii paremmin kuvaajiin ja Canvas esimerkiksi peleihin ja kompleksiseen grafiikkaan. (SVG vs canvas: how to choose)

3.4 JSON

JSON (JavaScript Object Notation) on tekstipohjainen, selväkielinen ja avoin standardi tiedonsiirtoon. JSONin rakennemalli on JavaScript-kielestä peräisin, ja sen tarkoitus on esittää tietorakenteita ja assosiatiivisia taulukkoja eli olioita. JSON on riippumaton käytettävästä ohjelmointikielestä, ja siihen on saatavilla jäsentimiä monille eri kielille. (The JSON Data Interchange Format 2013, ii)

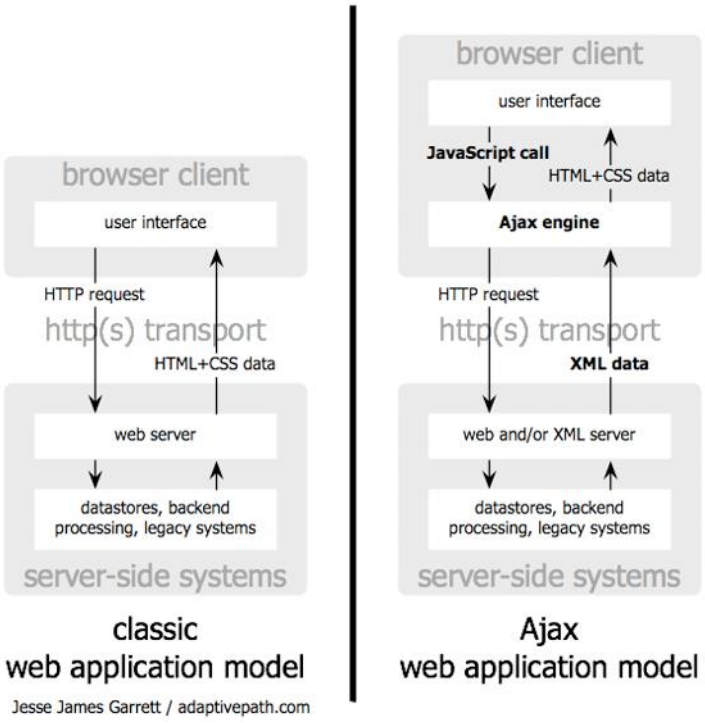
3.5 Ajax

Ajax on yhdistelmä monia eri teknologioita, jotka toimivat yhdessä mahdollistaen asynkronisen tiedon siirron asiakkaan selaimen ja palvelimen välillä ilman, että koko www-sivua tarvitsee ladata uudelleen (kuva 7). Teknologiat, joita Ajax käyttää, ovat DOM CSS, (X)HTML, XML/JSON ja Javascript. DOM esittää XML- ja HTML-dokumenttien rakenteen, ja DOM-ohjelmointirajapinnat tarjoavat JavaScriptille tavan päivittää sivun saatujen tietojen mukaisesti. CSS:llä sivu esitetään standardien mukaisesti. XML:ää tai JSON:ia käytetään datan manipuloimiseen ja konversioon. JavaScriptin XMLHttpRequest-oliota käytetään asynkroniseen datan hakuun palvelimelta HTTP-metodeilla. JavaScriptilla yhdistetään käytetyt teknologiat. (Morin 2007; jQuery Foundation 2015)

Klassisessa web-sovellusmallissa www-sivu pitää päivittää jokaisen pyynnön jälkeen. Ajax-mallissa web-sovellus voi pyytää vain tarvittavat tiedot, mitä sen tarvitsee päivittää (ei koko sivua), nopeuttaen käyttäjän ja sivun välistä vuorovaikutusta. Ajaxia käytettäessä käyttäjän ei tarvitse odottaa sivun loppuun latautumista, vaan komponentit latautuvat/päivittyvät asynkronisesti taustalla. Näin Ajax vähentää latautumisaikoja, kaistan kulutusta ja palvelinpuolen kuormitusta. Ajax perustuu avoimeen standardiin, se on selainriippumaton (tukevat selaimet toteuttavat sen identtisesti) ja suurin osa web-palvelimista tukee sitä. (Morin 2007, jQuery Foundation)

Ajaxin käytöllä on muutamia haittapuolia: muun muassa sivujen antama palaute käyttäjille voi häiriintyä, jos internet-yhteys on hidas tai pätkivä. Kaikki indeksointirobotit eivät

suorita JavaScript-koodia lainkaan, jolloin sivujen on tarjottava vaihtoehtoisia keinoja sisällön indeksointiin. Selaimet, jotka eivät tue JavaScriptia tai XMLHttpRequestia, eivät voi käyttää Ajaxilla toimivia sivuja oletetusti. HTML5:ttä edeltävät selaimet eivät toimineet Ajaxin kanssa ongelmitta: edellinen-napin painaminen ei aina vienyt käyttäjää loogiseen edeltävään kohtaan sivua ja sivuston tallentaminen kirjanmerkkeihin ei välttämättä tallentanut sivua halutussa tilassa. (Morin 2007)



Kuva 7. Web-sovellusmalli: klassinen ja Ajax (Garrett 2005)

3.6 REST-arkkitehtuurityyli

REST (Representational State Transfer) on arkkitehtuurityyli hajautetuille järjestelmille, mitä käytetään web-sovelluskehityksessä, sekä muun muassa WWW-hypertekstijärjestelmässä (World Wide Web). REST perustuu yhtenäisen rajapinnan tilattomaan, asiakas-palvelin-malliseen, välimuistilliseen ja kerroksittaiseen kommunikaatioprotokollaan – eli lähes yksinomaan HTTP-protokollaan. REST sivuuttaa komponenttien toteutuksien yksityiskohdat ja protokollan syntaksin keskittyäkseen komponenttien välisiin rooleihin ja keskinäisen vuorovaikutuksen rajoituksiin. Se käsittää olennaiset rajoitteet järjestelmän komponenteille ja datalle, mitkä määrittelevät arkkitehtuurin perusteet. Näin REST on enemminkin kokoelma periaatteita, kuin se on joukko standardeja. REST on noussut tämänhetkiseksi hallitsevaksi verkkopalveluiden suunnittelumalliksi, jos luokitellaan sitä käyttävien web-palvelujen määrän mukaan. REST onkin laajalti korvannut edeltävät

SOAP- ja WSDL-pohjaiset web-palvelut. (Fielding 2000, 76–77; Fredrich 2013, 6; Rodriguez 2015, 1)

REST-palvelut kommunikoivat HTTP-metodeilla (GET, POST, PUT ja DELETE), kun tietoa haetaan, lisätään, muutetaan tai poistetaan. REST:in päällimmäinen käsite on *resurssi*. Resurssi voi olla mitä tahansa sovellukselle tärkeätä, jota siinä halutaan jollain tapaa esittää tai käyttää. Jokaisella kutsuttavalla resurssilla on oma URI (Uniform Resource Identifier), joka on resurssin nimi tai osoite. REST-rajapinta on pohjimmiltaan kokoelma URIja, joita kutsutaan HTTP-metodeilla ja JSON/XML-esityksiä resursseista. URI-osoitteet tulee valita huolellisesti, että niistä muodostuu looginen kokonaisuus järjestelmän loppukäyttäjälle. (Richardson & Ruby 2007, 52–54; Fredrich 2013, 14–16; Rodriguez 2015, 2)

3.6.1 REST rajoitteet

Yhteinen rajapinta käsittää asiakkaiden ja palvelimien välisen rajapinnan: se yksinkertaistaa ja erottaa arkkitehtuuria siten, että jokainen osa kykenee skaalautumaan itsenäisesti. *Tilattomuus* tarkoittaa, että jokainen pyyntö asiakkaalta pitää sisällään kaiken tarvittavan tiedon pyynnön käsittelyyn, eikä asiakas-palvelin-session tilaa tallenneta palvelimelle. Tilattomuus parantaa a) skaalautuvuutta, koska palvelimen ei tarvitse ylläpitää sessio-tilaa, b) luotettavuutta, koska se helpottaa häiriöistä palautumista ja c) näkyvyyttä, koska palvelimen ei tarvitse katsoa pyyntöä pidemmälle todetakseen pyynnön vaatimuksia. *Asiakas-palvelin-malli* mahdollistaa käyttöliittymän ja tiedon tallennuksen erotuksen toisistaan. Tällainen toiminnallisuuden jako osa-alueisiin helpottaa käyttöliittymän siirtoa eri alustoille, parantaa skaalautuvuutta palvelinpäässä ja mahdollistaa osa-alueiden itsenäisen kehityksen tai korvauksen – olettaen, että rajapinta säilyy muuntumattomana. Turhat asiakas-palvelin-vuorovaikutukset voidaan eliminoida, kun (sallittu) osa palvelimen vastauksista säilötään asiakkaan selaimessa *välimuistiin.* Tallöin parannetaan yhteyden suorituskykyä ja skaalautuvuutta. Palvelin ei tiedä, onko se yhteydessä asiakkaaseen, yhdyskäytävään tai välityspalvelimeen – tätä kutsutaan *kerroksittaiseksi järjestelmäksi*, missä välityspalvelin voi parantaa järjestelmän skaalautuvuutta kuormantasauksella ja yhteisellä välimuistilla. Rajoittamalla jokaisen kerroksen näkyvyyttä voidaan systeemin kompleksisuutta rajata ja edistää riippumattomuutta alustasta. Viimeinen REST:iin kohdistuva rajoite on *ladattava koodi*, joka mahdollistaa Java-sovelmien ja asiakaspuolen skriptien lähettämisen palvelimen vastauksen mukana asiakkaalle. Ladattava koodi lisää

järjestelmän laajennettavuutta, mutta myös heikentää sen näkyvyyttä, minkä vuoksi se on ainoa vapaaehtoinen rajoite. (Fielding 2000, 78–85; Fredrich 2013, 6–9)

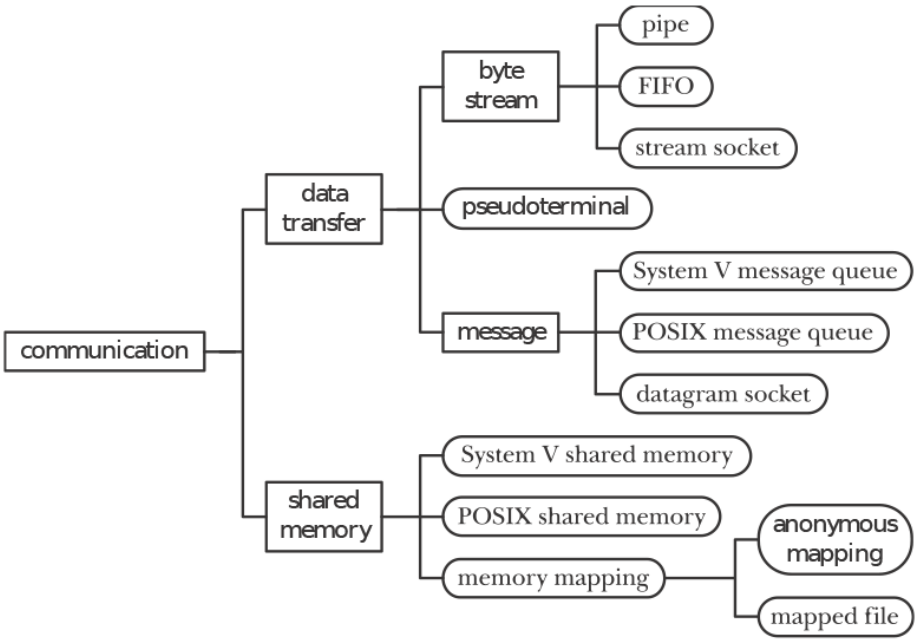
Nämä kuusi REST-arkkitehtuurintyylin sisältämää rajoitetta määriteltiin ensimmäisen kerran vuonna 2000 julkaistussa Roy Fieldingin väitöskirjassa. Jokainen rajoite on ennalta määritelty suunnittelupäätös, jolla on sekä hyvät, että huonot puolensa. Tavoitteena on, että hyvät puolet yhdessä ylittävät huonot puolet. Jos arkkitehtuuri ei noudata edellä mainittuja rajoitteita, sitä ei luokitella REST-arkkitehtuurityylin mukaiseksi. Näitä rajoitteita noudattamalla millä tahansa hajautettu hypermediajärjestelmällä on edellytykset toteuttaa toivottuja ominaisuuksia kuten hyvää suorituskykyä, luotettavuutta, muokattavuutta ja alustasta riippumattomuutta. (Fielding 2000, 86–88; REST Constraints)

3.7 Prosessien välinen kommunikaatio

Prosessien välisellä kommunikaatiolla (IPC, Inter Process Communication) tarkoitetaan prosessien tai ohjelmien välistä informaation vaihtoa eri kommunikaatioprotokollilla. Prosessit voivat olla ajossa yhdellä tai useammalla tietokoneella saman verkon sisällä. IPC:tä käyttävät sovellukset toteuttavat yleensä asiakas-palvelin-mallia. Termi viittaa yleisesti kahden eri prosessin väliseen kommunikaatioon, missä asiakas yhdistää palvelimeen ja tekee tyypillisesti *pyynnön* ja saa palvelimelta *vastauksen*. Tässä mallissa tiedon luonti tai haku on palvelimen vastuulla. Asiakkaan täytyy tietää palvelimen osoite ja varmuus sen olemassaolosta, kun taas palvelimen ei tarvitse tietää asiakkaasta teoriassa mitään. (Mathew & Wood 2008, 1)

Prosessien väliseen kommunikaatioon löytyy useita toteutuksia, joita on esitelty kuvassa 8. Viestintämahdollisuudet voidaan luokitella tiedonsiirron ja jaetun muistin toteutuksiin. Tiedonsiirto-viestinnässä käyttöjärjestelmän ydin (engl. kernel) välittää kommunikaation, joka tapahtuu luku- ja kirjoitus-systeemikutsuilla IPC-olioissa. Nämä kutsut voivat olla synkronisia tai asynkronisia, ja niitä voivat tehdä prosessit saman järjestelmän sisällä tai useat tietokoneet jaetun verkon sisällä. Jaetun muistin viestinnässä prosessien annetaan kartoittaa osan osoiteavaruuksistaan samoihin kohtiin välimuistia. Kommunikaatio tapahtuu yksinkertaisesti siten, että prosessi kirjoittaa muistiin ja toinen prosessi lukee viestin. Tämä viestintapa tarvitsee synkronointia prosessien välillä sekä muuta muistihal-

lintaa virhetilanteiden varalta, jotta vältytään muistivirheiltä. Koska työssä käytettiin tiedonsiirtoon luokiteltavia viestintätapoja, käsitellään niitä seuraavaksi tarkemmin. (Mathew & Wood 2008, 1; Zacchiroli 2011, 4–11)



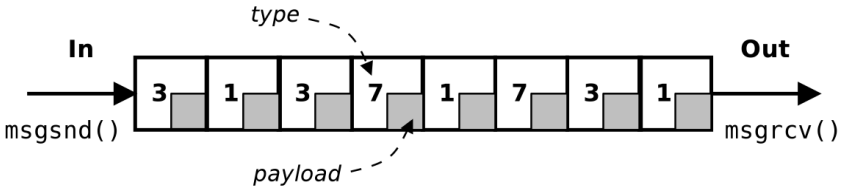
Kuva 8. Luokittelu Unixin IPC-toteutuksista (Zacchiroli 2011, muokattu)

3.7.1 Putki

Putket (pipe) ovat vanhimpia IPC-toteutuksia Unix-pohjaisissa käyttöjärjestelmissä: *pipe()* -systeemikutsu lisättiin kolmanteen Unix versioon vuonna 1973. Putket ovat prosesseja yhdistäviä bittivirtoja prosessien väliseen tiedonsiirtoon. Yksittäisen putken voidaan katsoa olevan kooltaan rajoitettu tiedosto, joka käyttää FIFO-periaatetta (first in, first out) datan hakemiseen, eli toisin sanoen putket ovat yksisuuntaisia: data kirjoitetaan putken toiseen päähän ja luetaan toisesta ulos. Putkeen kirjoittaminen ja lukeminen on synkronisoitu siten, että molempien operaatioiden suoritus estää toisen. Käyttöjärjestelmän ydin on vastuussa tästä sekä puskuroinnista ja viestien allokoinnista. Ohjelmoijan näkökulmasta toteutustapa on siis varsin yksinkertainen ja helppo realisoida. Putkia on kahden tyyppisiä: nimettyjä ja nimeämättömiä. Vain toisistaan periytyneet prosessit voivat käyttää nimeämättömiä putkia ja putket häviävät niitä käyttävien prosessien tuhoutuessa. Nimetyt putket ovat pysyviä rakenteita, joita voivat käyttää myös periytymättämät prosessit. Putket ovat hyödyllisiä, kun prosessit ovat vuorotettavissa ja saman laitteen sisällä. Putkesta lukemista ei voida estää (oli kyseessä mikä tahansa prosessi), joten putkitoteutus ei välttämättä ole tietoturvallinen vaihtoehto. (Lewandowski 1997, 3–4; Bhatt 2004, 8–12; Zacchiroli 2011, 17–25)

3.7.2 Sanomajono

Sanomajonossa (message queue) kaksi prosessia jakavat tietoa yhteisen sanomajonon kautta noudattaen FIFO-periaatetta. Lähettävä prosessi asettaa sanomajonoon tunnuksellisen viestin, josta vastaanottava prosessi voi sen noutaa olettaen, että se tietää viestin tunnuksen (kuva 9). Sanomajono voi säädellä siinä lähetettävien viestien luku- ja kirjoitusoikeuksia prosesseille – esimerkiksi siten, että vain valituilla prosesseilla on oikeus sanomajonon viesteihin. Toisin kuin putkitoteutuksen bittivirrassa, sanomajonossa viesteillä on määritellyt koot. Myös kokonaismäärälle dataa, jota sanomajonossa voi olla, on määritelty rajoituksia eri järjestelmissä: esimerkiksi viestien määrä tai viestin koko voi olla rajoitettu. Toteutuksen systeemikutsuja voidaan kutsua asynkronisesti avoimena tai synkronisesti estävänä. Estävässä viestinvälityksessä prosessi on pysäytetty, kunnes viesti on siirretty, asynkronisessa viestinvälityksessä prosessi voi jatkaa ajoaan normaalisti. Sanomajonojen suorituskykyä rajoittaa erityisesti suurien viestien sisältöjen kopiointi kahdesti. Kun viesti lähetään, se kopioidaan lähettäjän puskurista muistiin, ja kun lähettäjä vastaanottaa viestin, se tallennetaan puskuriin. Viestien kopiointia voidaan kuitenkin kiertää eri toteutuksissa. Sanomajonot eivät vaadi kommunikoivilta prosesseilta synkronointia toimiakseen. (Cortesi 1996, 127–130; Marshall 1999)



Kuva 9. Sanomajonon toiminta (Zacchiroli 2011)

3.7.3 Pistoke

Pistokkeet (socket) ovat hyvin monipuolisia peruskomponentteja, jotka mahdollistavat kaksisuuntaisen järjestelmien ja prosessien välisen kommunikaation. Pistokkeen *tunnus* on abstraktio, joka tarjoaa nimeämisrakenteen ja protokollat. Pistoketunnuksia on useita, mutta niistä käytetään yleisesti vain internetin verkkotunnuksia, missä kaksi palvelinta kommunikoivat internetin välityksellä, ja UNIX-tunnuksia, joissa kaksi prosessia kommunikoivat keskenään saman tiedostojärjestelmän sisällä. (Marshall 1999)

Pistoketyyppejä on viittä erilaista, mutta niistä käytetään enimmäkseen vain *stream*- ja *datagrammi*-pistokkeita. Jokaisella pistoketyypillä on omat suorituskyky- ja luotettavuusominaisuutensa ja vain samantyyppiset pistokkeet voivat olla yhteydessä toisiinsa. Stream-pistokkeet käsittelevät kommunikaatiokanavaa jatkuvana bittivirtana, kun taas datagram-pistokkeet voivat lukea vain kokonaisia viestejä. Stream-pistokkeissa data saapuu perille samassa järjestyksessä, missä se on lähetetty, kun molemmat prosessit ovat hyväksyneet kommunikointikutsun ennen kommunikoinnin aloittamista. Tämä ominaisuus helpottaa stream-pistokkeiden ohjelmointia ja tekee niistä hyvin luotettavia. Datagrammi-pistokkeet eivät takaa, että data saapuu perille samassa järjestyksessä, kuin se on lähetetty, koska yksittäiset datagrammit voivat kulkea eri polkujen kautta päämääräänsä. Datagrammi-pistokkeet eivät käsitä yhteyttä, vaan jokainen datagrammi lähetetään ja prosessoidaan erikseen. Datagrammi-pistokkeiden lähettämä data ei välttämättä kulkeudu odotetulle vastaanottajalle, minkä vuoksi pistoketyyppiä pidetään epäluotettavana. (Lewandowski 1997, 6–7; Marshall 1999)

3.7.4 ZeroMQ

ZeroMQ (lyh. ZMQ) ZMQ on C++-kielellä kirjoitettu asynkroninen kommunikaatiokirjasto, joka ei ole väliohjelmisto viestinvälitykseen vaan ohjelmistoihin upotettava sovelluskehys. ZMQ käyttää pistoke-toteutukselle ominaista ohjelmointirajapintaa, vaikka sen sisäinen toiminta on lähempänä pipe-toteutusta. ZMQ:n etuja ovat sen suorituskyky, ohjelmointirajapinnan yksinkertaisuus, skaalautuvuus, luotettavuus sekä laaja tuki eri viestintämalleille ja tietoliikenneprotokollille. (Hintjens 2010; Piël 2010)

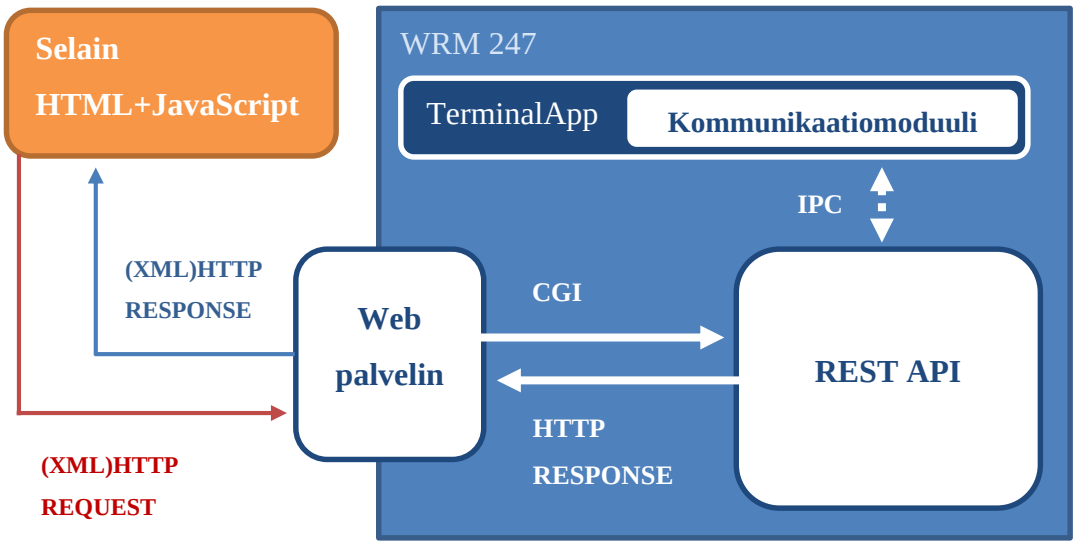
ZMQ on erittäin suorituskykyinen; se päihittää suorituskyvyssä monet muut AMQP-protokollat (Piël 2010). Osasyynä ZMQ:n suorituskykyyn on sen älykäs viestien annostelu. Ohjelmointirajapinta ZMQ:ssa on erittäin helppokäyttöinen sen yksinkertaisuuden ansiosta. Kirjasto hoitaa kaiken asynkronisesti taustaprosessina omassa säikeessään, jolloin ohjelmoijan ei tarvitse huolehtia muun muassa jatkuvasta datan syötöstä pistoke-puskurille. Protokollapuskureita käyttämällä ZMQ:n kautta voi lähettää viestejä missä tahansa formaatissa. Yksi ZMQ pistoke voi yhdistää itsensä moneen eri päätepisteeseen ja tasoittaa viestikuormaa niiden välillä. ZMQ tukee eri viestintämalleja kuten pyyntö-vastaus, pub-sub ja push-pull, TCP-, PGM- ja IPC -tietoliikenneprotokollien kautta. ZMQ on julkaistu GNU lesser general public -lisenssin alla. (Hintjens 2010; Piël 2010)

4 TOTEUTUS

4.1 Määrittely

Laitteen sisäiseen konfigurointikäyttöliittymään toteutetaan laitteen reaaliaikaisten mittausarvojen visuaalinen esittäminen kuvaajalla. Käyttöliittymästä valitaan ensin moduuli eli laitteelle konfiguroitu mittausominaisuus (esimerkiksi kiihdytysanturi) ja sen jälkeen jokin moduuleille konfiguroiduista datanoodeista, joihin mittausarvot on tallennettu. Kun nämä on valittu, käyttöliittymässä oleva kuvaaja piirtää datanoodin mittausarvoja reaaliaikaisesti eli lisäten mittausarvoja kuvaajaan aina kun uusi mittaus on laitteella tehty. Asettamalla kursorin mittauspisteen päälle käyttäjä saa tarkemmat tiedot mittausajankohdasta ja pisteen mittausarvosta.

Konfigurointikäyttöliittymä pyörii WRM 247:n Lighttpd-web-palvelimelta, joka on yhteydessä REST-rajapintaan CGI-tekniikalla. Kun selaimesta lähetetään XMLHTTP-pyyntö serverille, CGI ohjaa pyynnön serveriltä REST-pohjaiselle-rajapinnalle. Rajapinta ottaa IPC:n kautta yhteyden laitteen terminaalisovelluksessa (TerminalApp) sijaitsevaan kommunikointimoduuliin, joka hoitaa pyydettyjen tietojen etsinnän ja lähetyksen takaisin. Rajapinta lähettää moduulilta saadun vastauksen JSON-muodossa takaisin selaimelle, jossa tiedot sijoitetaan käyttöliittymään (kuva 10). Käytettävää IPC-tekniikkaa voidaan soveltaa myös muihin laitteen sisäisten prosessien väliseen kommunikaatioon.



Kuva 10. Local UI -arkkitehtuuri

4.2 Suunnittelu

Suunnittelun ensimmäinen vaihe oli tarvittavien kirjastojen etsiminen. Käyttöliittymäpään toteutusta varten tarvittiin kirjasto mittausarvojen piirtämiseen kuvaajaan. Sopivaa JavaScript-kirjastoa etsittiin tiedostokoon, suorituskyvyn, laajennettavuuden ja yksinkertaisuuden perusteilla. Kuvaajakirjaston käyttämäksi tekniikaksi valittiin SVG sen paremman interaktiivisuuden myötä. Käyttöliittymässä tarkastellaan rajoitettua määrää mittauspisteitä, jolloin suuren tietomäärän aiheuttamilta suorituskykyongelmilta vältytään. Vaihtoehtoja oli useita, mutta näistä valittiin ensiksi MetricsGraphics.js -kirjasto sen ominaisuuksien, keveyden ja hiotun käyttöliittymän perusteella. Myöhemmin työssä MetricsGraphics.js-kirjasto korvattiin Rickshaw-kirjastolla, koska ensiksi mainittu kirjasto ei tukenut reaaliaikaista kuvaajan piirtämistä yhtä moitteettomasti kuin jälkimmäinen. Rickshaw on Shutterstockin kehittämä D3.js-JavaScript-kirjaston päälle rakennettu työkalu aikasarja-kuvaajien luomiseen.

REST-rajapinnan resurssihierarkiaksi suunniteltiin olemassa olevaa rajapintaa mukaillen taulukossa 2 esitelty kokonaisuus. Järjestelmässä tapahtuneet virheet palautettaisiin relevanttina HTTP-virhekoodina takaisin selaimelle.

Taulukko 2. REST-rajapinnan resurssihierarkia

Pyynnön URI	https://<WRM_IP_ADDRESS>/api/module
Metodi	GET
Vastaus	{ "modules": ["<Modulename>", …] }
Pyynnön URI	https://<WRM_IP_ADDRESS>/api/module/<module>
Metodi	GET
Vastaus	{"datanodes": [{ "<Datanodename>": "<MID>"}, …] }
Pyynnnön URI	https://<WRM_IP_ADDRESS>/api/module/<module>/<datanoodi>
Metodi	GET
Vastaus	{ "data": { "<time>", "<value>" } }

Prosessien väliseen kommunikaatioon valittiin ensimmäiseksi toteutustavaksi *ZeroMQ*-niminen kommunikointikirjasto. Viestintämalliksi valittiin pyyntö-vastaus-malli (request-reply), joka on hyvin yleisesti käytetty tämän kaltaisissa sovelluksissa. Tämän opinnäytetyön tarkoitukseen tämä malli soveltuu yksinkertaisuudessaan täysin riittävästi.

4.3 Työkalut

Konfigurointikäyttöliittymän front-end-koodin (HTML, CSS, JavaScript, PHP) editointiin käytettiin Brackets-lähdekoodieditoria sen nopeuden, joustavuuden ja helppokäyttöisyyden vuoksi. Koska WRM247 -laitteen kääntäminen tehdään Linux-käyttöjärjestelmällä, laitteen sisäisen koodin luonti ja editointi tehtiin virtuaalikoneen sisällä. Tämän luontiin käytettiin Oracle VirtualBox -virtualisointiohjelmistoa ja olemassa olevaa (Ubuntu) imagea. Virtuaalikoneen sisällä koodin editointi tehtiin Qt-kehitysympäristöllä. Laitteeseen saatiin SSH-yhteys PuTTy-pääte-emulaattorin avulla. Laitteen tiedostoja muokattiin WinSCP-ohjelmalla, joka yhdistää laitteeseen SCP-protokollalla.

4.4 IPC-kommunikoinnin toteutus

Laitteen IPC-kommunikaatio koostuu kahdesta osiosta: REST-pohjaista rajapintaa toteuttavasta XHTML-kutsuihin vastaava lohkosta ja terminaalisovelluksessa sijaitsevasta IPC-moduulista. REST-lohko on jaettu useaan eri toiminnallisuuksia toteuttavaan osaan: laiteohjelmiston tietoja-, verkkoyhteys-tietoja- ja tämän työn tietoja käsittelevä osa. Pyynnöt käsitellään loogisesti eri osissa pyynnön URI:sta päätellen. REST-lohkon ja sisäisen web-palvelimen välinen CGI-kommunikaatio on myös hoidettu tässä kohtaa laitteen sisäistä ohjelmistoa.

REST-lohko saa selaimelta XHTML-pyynnön, jonka URI-merkkijonosta voidaan tunnistaa pyydetty resurssi. Kun resurssipyyntö on tiedossa, se lähetetään eteenpäin terminaalisovelluksen moduulille. Lähetystä varten on koodissa ensin alustettava molemmissa osioissa ZMQ-ohjelmointirajapinnan mukaisesti *konteksti*, joka toimii eräänlaisena säiliönä prosessin sisäisille pistokkeille. Kontekstin avulla luodaan käytettävät pistokkeet, minkä jälkeen pistokkeet sidotaan paikalliseen tiedostopolkuun laitteella. Seuraavaksi asiakkaana toimiva puolisko (tässä toteutuksessa REST-lohko) luo viestin, johon sijoitetaan resurssipyyntö merkkijonona ja viesti lähetetään eteenpäin IPC-moduulille.

Terminaalisovelluksen IPC-moduuli toimii omassa säikeessään, jolloin moduuli on jatkuvasti ajossa ja voi vastaanottaa viestejä. Säikeen sisällä viestiä odotetaan estämällä muun koodin suoritus, kunnes REST-lohkolta saadaan viesti. Pyyntö jäsennetään saadusta viestistä, ja sen sisällön perusteella haetaan joko lista kaikista laitteille määritel-

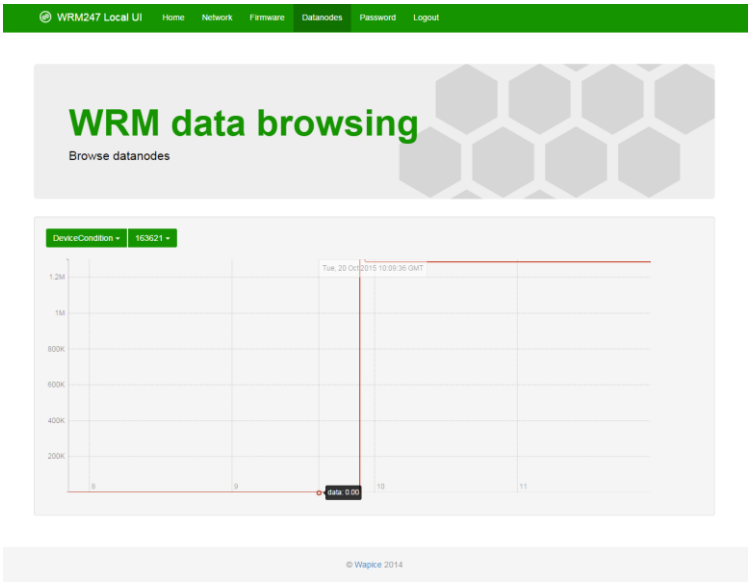
lyistä (mittaus ja toiminnallisuus) moduuleista, lista kaikista valitun moduulin datanoodeista tai datanoodin viimeisin mittausarvo. Molemmat listat haetaan laitteen konfiguraatiotiedostoa lukemalla. Viimeisimmät mittausarvot tallentuvat aina erilliseen välimuistiin, josta moduulit voivat niitä lukea. Laitteen sisäinen muisti on rajallinen, joten sen ei ole tarkoitus tallentaa kaikkea mittaamaansa tietoa, vaan mittausarvot lähetetään eteenpäin palvelimille talletukseen. Laite tallentaa mittaustietoja, välimuistin ohella, vain kun servereihin ei saada yhteyttä, jolloin tallennetut tiedot lähetetään heti kun yhteys saadaan uudestaan muodostettua. Datan haku servereiltä olisi vastoin mittausnäkymän käyttötarkoitusta, sillä ensisijaisesti dataa tarkastellaan IoT-Ticketin web-käyttöliittymästä, ja tämän vuoksi mittausnäkymän tarkoitus on rajoitettu reaaliaikaisten mittausarvojen seurantaan.

Kun haluttu resurssi on haettu, se muutetaan JSON-muotoon ja lähetetään takaisin REST-lohkolle. Jos pyynnön suorituksessa tapahtui virheitä, vastaukseksi lähetään ”Error <virheen syy>” ja laitteen ulostuloon kirjoitetaan saatu virhe testausta ja kirjanpitoa varten. REST-rajanpinnan lohkossa vastaus tarkastetaan virheiden varalta, ja jos sellainen havaitaan, lähetetään virheilmoitusta vastaava HTTP-tilakoodi web-serverille CGI-tekniikalla ja siitä eteenpäin selaimelle. Muulloin terminaalisovelluksesta saatu vastaus lähetetään sellaisenaan: vastaus on jo JSON-muodossa, sitä ei tarvitse enää muokata.

4.5 Front-end-toteutus

Työtä edeltävä versio konfigurointikäyttöliittymästä oli Bootstrapia hyödyntävä sivusto, jossa toiminnallisuus oli tehty JavaScriptilla ja PHP-kielellä. PHP:llä (serveripuolella) hoidettiin sisäänkirjautuminen, sessionhallinta, käyttäjän tunnistus (basic access authentication) HTTP-metodeihin ja salasanan muuttaminen. JavaScriptilla hallitaan sivujen vaihto, tapahtumat (esimerkiksi nappien painallukset) ja tärkeimpänä Ajaxin hallinta. Huomionarvoista on, että käyttöliittymässä sivu pysyy aina samana, ja sisällön vaihto hoidetaan JavaScriptillä siten, että muu kuin valittu sisältö piilotetaan näkyvistä. Tällä tavalla vältytään turhilta pyynnöiltä ja nopeutetaan sivujen vaihtoa. Haittapuolena tällaisessa toteutuksessa on muun muassa edellinen-napin menetetty toiminnallisuus. Ajax-metodeilla tehdään tarvittavat HTTP GET-pyynnöt REST-rajapinnalle, josta saatavat tiedot päivitetään DOMiin.

Käyttöliittymään luotiin HTTP:llä uusi sivu mittausnäkymää varten, mihin sijoitettiin kuvaaja, kaksi pudotusvalikollista nappia, virheilmoituselementit ja tekstiä. Kuvaaja toteutettiin JavaScript-kielellä käyttämällä MetricsGraphics.js-kirjaston ohjelmointirajapintaa. JavaScriptilla täydennettiin olemassa olevaa toteutusta niin, että uusi sivu sulautui vanhaan toteutukseen, sekä kirjoitettiin tapahtumankäsittelijämetodit, jotka lähettävät GET-pyynnön Ajaxilla eteenpäin. REST-rajapinnalta saatu JSON-muotoinen vastaus jäsennetään ja sijoitetaan pudotusvalikkoihin, kun kyseessä on moduuli tai datanoodi-lista. Lista moduuleista ladataan heti, kun mittausnäkymä avataan, ja datanoodi-lista haetaan valitun moduulin perusteella. Kun datanoodi on valittu listalta, lähetetään pyyntö, saatu mittausdata jäsennetään vastauksesta ja lisätään kuvaajaan sekä asetetaan intervalli, jolla selain lähettää pyyntöjä rajapinnalle seuraavan datan hakemiseen. Jos valittuun moduuliin ei ole konfiguroitu datanoodeja, rajapinnalta saadaan HTTP-virhekoodi ja käyttöliittymässä näytetään virheilmoitus. Virheilmoitus näytetään myös, jos järjestelmässä tapahtuu jokin muu virhe. Kuvaaja ja pyyntö-intervalli nollataan, kun valitaan eri datanoodi tai mittausnäkymästä poistutaan.



Kuva 11. Käyttöliittymän mittausarvojen tarkastelunäkymä

5 TESTAUS

Laitteen sisäisen koodin testaus koostui terminaalisovelluksen kääntämisestä, kun ohjelmakoodia oli muokattu, sen siirtämisestä laitteelle ja laitteen ulostulon tarkastelusta virheilmoitusten varalta PuTTy-pääte-emulaattorin avulla. Front-end-toteutuksen testaus rajoittui enimmäkseen JavaScript-tiedostojen muokkaamiseen suoraan laitteelta.

5.1 Terminaalisovelluksen kääntäminen

Uuden toiminnallisuuden lisäämiseksi ohjelmakoodi ja käytettävät kirjastot tulee kääntää laitteelle ja muuttaa laiteohjelmiston päivityspaketiksi. Tässä tapauksessa päivitys siirretään laitteelle konfigurointikäyttöliittymän kautta, jonka jälkeen laite päivittää ohjelmistonsa ja käynnistyy uudelleen. Työssä käytetyn ZeroMQ-kommunikointikirjaston kääntö laitteelle osoittautui pulmalliseksi. Kirjastosta täytyi ensiksi konfiguroida pois turhaa tilaa vievät tarpeettomat osat. Kääntäjä ei löytänyt linkitystiedostoa kirjaston käyttöön C++-kielellä, joten se jouduttiin ottamaan mukaan manuaalisesti CMake-tiedostoja muokkaamalla. Laite ei tue IPV6-protokollaa, mutta kirjastossa oli tuki sille, mikä aiheutti käännösvirheitä. ZMQ:n konfiguroinnista ei löytynyt vaihtoehtoa tuen poistolle, joten IPV6-tuki jouduttiin manuaalisesti kommentoimaan ulos kirjaston lähdekoodista. Kun ZMQ saatiin vihdoin kääntymään virheittä, kävi ilmi, että kirjaston binaarien koko ylitti tiedostokoolle asetut rajat. Koko kirjaston käytöstä jouduttiin luopumaan laitteen rajallisen muistikapasiteetin vuoksi.

Uudeksi prosessien välisen kommunikaation toteutustavaksi valittiin Unix-pistokkeet, joita laite tukee natiivisti. IPC-kommunikaatiototeutuksen uudelleenkirjoitus ei kuitenkaan vienyt paljoakaan aikaa, kiitos ZMQ:n rajapinnan, jonka koodirakenne vastasi toteutettavaa pistoke-toteutusta. Suurimmat eroavaisuudet liittyivät yhteyden alustukseen, viestien sijoitukseen puskureihin ja yhteyden sulkemiseen. ZMQ hoiti nämä kaikki itse, jolloin ohjelmointi oli korkeatasoisempaa ja mahdollisuus virheisiin pienempi. Unix-pistokkeita käytettäessä tarvitsi vain olla huolellisempi ja varautua perusteellisesti vikatilanteisiin.

Ensimmäinen käännös tällä toteutuksella tuotti ensin oletetun määrän käännösvirheitä tyyppimuunnosvirheistä puuttuviin kirjastoihin. Kun nämä käännösvirheet oli korjattu,

terminaalisovellus voitiin vihdoin käännetää. IPC ei ensin toiminut lainkaan, vaan palautti *bind-erroria* eli virhettä REST-lohkon ja moduulin välisessä yhteydenotossa. Havaittiin, että ongelma johtui pistokkeille annetusta tiedostopolusta. Ratkaisu oli Linuxin abstraktin nimiavaruuden käyttö tiedostopolun sijasta.

5.2 Kuvaajan piirto

Käytetty kuvaaja-kirjasto ei toiminutkaan reaaliaikaisen datan kanssa niin sujuvasti, kuin oli luultu. Koska toteutuksen vaihto ei tässä vaiheessa aiheuttanut suuremmin lisätyötä, vaihdettiin kuvaajakirjastoksi Rickshaw-kirjasto. Selaimen ja REST-rajapinnan välisessä kommunikoinnissa oli aluksi ongelmia, joista suurin osa osoittautui kirjoitusvirheiksi koodissa ja ne korjattiin nopeasti. jQuery ei ensiksi pystynyt parsimaan rajapinnalta saatua JSON-muotoista vastausta, koska vastausta ei ollut muotoiltu oikein IPC-moduulissa. Kyseisen ongelman ratkaisu vaati muutoksia kommunikaatiomoduulin JSON-muunnokseen.

6 POHDINTA

Tämän opinnäytetyön tavoitteena oli tutustua prosessien välisen kommunikaation toimintaan ja toteuttamistapoihin sekä suunnitella ja kehittää WRM247 -laitteelle sen konfiguraatiokäyttöliittymään näkymä laitteen reaaliaikaisista mittaustiedoista. Työn aikana eri IPC-toteutustapoihin tutustuttiin kattavasti, ja itse työssä IPC-kommunikaatioon kokeiltiin kahta eri lähestymistapaa. Työlle asetetut tavoitteet täytettiin, ja lopputuloksena oli toimiva uusi ominaisuus etähallintalaitteelle, mikä voidaan lisätä varsinaisen laiteohjelmiston päivityspakettiin.

Prosessien välisestä kommunikaatiosta löytyi verkosta ja kirjallisuudesta paljon luotettavaa materiaalia, dokumentaatiosta esimerkkeihin. Terminaalisovellukselle oli tehty toiminnallisen määrittelyn dokumentti, mutta itse koodi oli suurimmaksi osaksi kommentoimatonta, ja siihen piti tutustua laajemmin ymmärtääkseen koodin toimintaa. Työn toteutuksessa törmättiin muutamassa kohtaa järjestelmästä laajempaa tietoutta vaativiin ongelmiin, joihin kiitettävästi apua antoi työnohjaaja ja muut laitteen kehittäjät. Linuxin alaisesta ohjelmistokehityksestä ja sulautettujen järjestelmien toiminnasta oli paljon uutta opittavaa työn kuluessa. Suurimmaksi virheeksi osoittautui toteutukselle liian laajan ZMQ-kirjaston käyttö, kun olisi voitu käyttää yksinkertaisempaa tapaa alusta lähtien.

Työssä tehtyä IPC-kommunikaatiototeutusta voidaan jatkokehittää toimimaan sulavammin myös muiden komponenttien kanssa. Kuvaaja näyttää tällä hetkellä vain yhden datanoodin mittausarvot, mutta se voidaan muuttaa näyttämään myös useampaa. Mittausnäkymän toimintaa ei ole vielä testattu mobiililaitteilla, koska sitä ei ollut määritelty vaatimuksiin, mutta sen toiminta voidaan ainakin osittain varmistaa pienillä lisäyksillä.

LÄHTEET

Bellini, H., Costa, D., Covello, J., Jankowski, S. & Ritchie, J. 2014. The Internet of Things: Making sense of the next mega-trend. The Goldman Sachs Group, Inc. Sijoitusanalyysi. Luettu 9.10.2015. http://www.goldmansachs.com/our-thinking/outlook/internet-of-things/iot-report.pdf

Bhatt, P.C.P. 2004. Module 7: Inter-Process Communication. Indian Institute of Science. http://www.nptel.ac.in/courses/106108101/pdf/Lecture_Notes/Mod%207_LN.pdf

Bootstrap team. About. Luettu 10.10.2015. http://getbootstrap.com/about/

Buyya, R., Gubbi, J., Marusic, S. & Palaniswami, M. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems. Amsterdam: Elsevier Science Publishers.

David, C. 1996. Topics in IRIX™ Programming. Silicon Graphics, Inc. http://techpubs.sgi.com/library/manuals/2000/007-2478-004/pdf/007-2478-004.pdf

Deveria, A. CanIUse SVG. Luettu 12.10.2015. http://caniuse.com/#feat=svg

Evans, D. 2011. The Internet of Things - How the Next Evolution of the Internet Is Changing Everything. Cisco. Luettu 10.10.2015. http://www.iotsworldcongress.com/documents/4643185/3e968a44-2d12-4b73-9691-17ec508ff67b

Fielding, R.T. 2000. Representational State Transfer (REST). Architectural Styles and the Design of Network-based Software Architectures. University of California.

Flanagan, D. 2011. JavaScript: The Definitive Guide. 5. painos. Kalifornia: O'Reilly Media.

Floerkemeier, C. & Mattern, F. 2010. From the internet of computers to the internet of things. Institute for Pervasive Computing. Luettu 10.10.2015. http://vs.inf.ethz.ch/publ/papers/Internet-of-things.pdf

Fredrich, T. RESTful Service Best Practices. Versio 1.2. Paerson eCollege. Luettu 15.10.2015. https://github.com/tfredrich/RestApiTutorial.com/blob/master/media/RESTful%20Best%20Practices-v1_2.pdf

Fulton, J-S. 2011. HTML5 Canvas. 2. painos. Kalifornia: O'Reilly Media.

Garrett, J.J. 2005. Ajax: A New Approach to Web Applications. Luettu 13.10.2015. http://adaptivepath.org/ideas/ajax-new-approach-web-applications/

Hintjens, P. 2010. ØMQ - The Guide. Luettu 16.10.2015. http://zguide.zeromq.org/page:all

Wapice Oy. 2015. IoT-Ticket. Luettu 6.10.2015. https://www.iot-ticket.com/

Kyriakopoulos, G., Lemus, E., Leopold, S., Louthan, F.G., McCourt, T.C., Mosesmann, H., Peterson, B., Sklar, A., Smigie, J.S., Tillman, T. & Toomey, D. 2014. The Internet of Things - A Study in Hype, Reality, Disruption, and Growth. Raymond James. Luettu 9.10.2015. http://www.vidyo.com/wp-content/uploads/The-Internet-of-Things-A-Study-in-Hype-Reality-Disruption-and-Growt....pdf

Lewandowski, S.M. 1997. Interprocess Communication In UNIX and Windows NT. Brown University. http://cs.brown.edu/~scl/files/IPCWinNTUNIX.pdf

Marshall, D. 1999. IPC:Message Queues. Cardiff School of Computer Science. Luettu 18.10.2015. https://www.cs.cf.ac.uk/Dave/C/node25.html

Mathew, V. & Wood, T. 2008. CMPSCI 377: Operating Systems Lecture 5. Laboratory for Advanced Software Systems. Luettu 17.10.2015. http://lass.cs.umass.edu/~shenoy/courses/fall10/lectures/Lec05_notes.pdf

MDN. 2015. JavaScript Guide. Luettu 10.10.2015. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

Microsoft. SVG vs Canvas: how to choose. Luettu 12.10.2015. https://msdn.microsoft.com/en-us/library/gg193983(v=vs.85).aspx

Morin, P. 2007. COMP2405: Internet Application Programming. 15.03.2007. Carleton University. Luettu 15.10.2015.
http://cglab.ca/~morin/teaching/2405/notes/ajax1.pdf

Piël, N. 2010. ZeroMQ an introduction. Luettu 16.10.2015.
http://nichol.as/zeromq-an-introduction

Richardson, L. & Ruby S. 2007. RESTful Web Services. 1. painos. Kalifornia: O'Reilly Media.

Roberts, A. 2014. Introduction to the HTML5 Canvas Element. Luettu 11.10.2015.
http://www.sitepoint.com/web-foundations/introduction-html5-canvas-element/

Rodriguez, A. 2015. RESTful Web services: The basics. IBM. developerWorks. Luettu 16.10.2015.
https://www.ibm.com/developerworks/library/ws-restful/ws-restful-pdf.pdf

The jQuery Foundation. 2015. Learn jQuery: Ajax. Luettu 13.10.2015
https://learn.jquery.com/ajax/

The jQuery Foundation. What is jQuery? Luettu 10.10.2015.
https://jquery.com/

The JSON Data Interchange Format. 2013. Ecma International. Luettu 13.10.2015.
http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

W3. 2011. SVG Introduction. Luettu 12.10.2015.
http://www.w3.org/TR/SVG/intro.html

W3Techs. Usage of JavaScript libraries for websites. Luettu 11.10.2015.
http://w3techs.com/technologies/overview/javascript_library/all

WhatIsRest.com. REST Constraints. Luettu 15.10.2015.
http://whatisrest.com/rest_constraints/index

Wikipedia. Internet of Things. Luettu 10.10.2015.
https://en.wikipedia.org/wiki/Internet_of_Things

Zacchiroli, S. 2011a. Programmation Systèmes Cours 6. Interprocess Communication and Pipes. Paris Diderot University. Luettu 18.10.2015.
https://upsilon.cc/~zack/teaching/1112/progsyst/cours-06.pdf

Zacchiroli, S. 2011b. Programmation Systèmes Cours 10. System V IPC. Paris Diderot University. Luettu 18.10.2015.