Matias Grönholm

Hajautetun anturiverkon sulautettu valvonta-, visualisointi- ja raportointipalvelin

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

21.4.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Matias Grönholm Hajautetun anturiverkon sulautettu valvonta-, visualisointi- ja raportointipalvelin 25 sivua 21.4.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Ilpo Kuivanen Alshainin hallituksen puheenjohtaja, Ilmari Lehmusoksa

Tämän insinöörityön tarkoituksena oli luoda asiakkaan toiveisiin perustuva laitteisto- ja sovelluspaketti, joka mittaisi teknisen laboratorion lämpötilaa, ilmankosteutta ja painetta, kirjaisi mittaustulokset ja näyttäisi käyttäjille laboratorion tilan verkkoympäristössä reaaliajassa.

Työssä ohjelmoitiin sulautetulle alustalle Pythonilla ohjelma, joka pyytäisi anturiverkon sensoreilta mittaustuloksia, purkaisi sen ihmiselle luettavaan muotoon ja lokittaisi nämä laitteen muistiin. Kyseinen käytetty anturiverkko on Alshain Oy:n valmistama tuote, joka oli jo asennettu asiakkaan tiloihin.

Lisäksi projektiin kuului pystyttää samalle sulautetulle alustalle verkkopalvelin ja verkkosivut, joista käyttäjät pystyisivät katsomaan tallennetuista lokeista piirrettyjä kuvaajia eri antureiden lukemista ja pyytämään eri ajanjaksoista raportteja.

Insinöörityö selittää ohjelman rakenteen, kehitysvaiheet ja käytetyt teknologiat ja antaa kuvan lukijalle sulautetun ohjelmistoprojektin kehityksestä.

Avainsanat	Anturiverkko, sulautettu, Python, verkko-ohjelmointi, verkko

Author(s) Title Number of Pages Date	Matias Grönholm Monitoring-, visualization- and reporting server for a distributed sensor network 25 pages 21 April 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Ilpo Kuivanen, Senior Lecturer Ilmari Lehmusoksa, Chairman of the Board, Alshain

The goal of this thesis was to create, according to the client's specifications, hard- and software package, which would measure the temperature, relative air humidity and pressure of a technical laboratory. These measurements would be shown in real time in a web page, which runs on an embedded computer.

An embedded program was coded with Python, which would request measurement results from the sensor network, decoded the responses to human-readable-format and then log them. The sensor network is made by Alshain, which was already installed in the clients premises.

In addition, this project included a web server, which runs in the same embedded system as the sensor network interface as well as web pages for showing the measurements in real time to the clients personnel.

This thesis explains the structure of this system, the development phases and the technologies used to implement this product. This gives insight to the reader how one can approach developing similar solutions.

Keywords	Sensor network, embedded, Python, web-development, web

Sisällys

Lyhenteet ja termit

Sensori	Tässä työssä käytetään termiä sensori tarkoittamaan Alshain Oy:n valmistamaa monianturista etäpäätelaitetta.
CRC	Cyclic redudancy check. Tiivistealgoritmi tarkistussumman luontiin.
LRC	Longitudinal redundancy check. Tiivistealgoritmi tarkistussumman luontiin.
RTU	Remote terminal unit. Etäpäätelaite.
ASCII	American Standard Code for Information Interchange. Merkistöstandardi.
RS-485	Standardi balansoidulle sarjaliikenneväylälle.
I/O	Input / Output. Siirräntä.
NTP	Network Time Protocol. Ajantahdistusprotokolla.
RAM disk	Osa keskusmuistia, joka on muutettu toimimaan kuin levy- tai massamuisti.
COTS	Commerical off-the-shelf. Helposti saatava kaupallinen tuote.
UNIX	Käyttöjärjestelmä.
LINUX	UNIX:n kaltainen käyttöjärjestelmä.
JSON	JavaScript Object Notation. Tiedonjäsennysformaatti.
SD	Secure Digital. Muistikorttityyppi.
SoC	System on Chip. Järjestelmäpiiri.
BBB	BeagleBone Black. Yhden piirilevyn tietokone.
ARM	Advanced RISC Machine. Prosessoriarkkitehtuuriperhe.
URL	Uniform Resource Locator. Standardoitu osoitetietue.

FDIR	Fault detection, isolation, and recovery. Virheiden tunnistus, eristys ja korjaus.
CSV	Comma-separated value. Tiedostotyyppi missä data on tallennettu tiedostoon taulumaisessa muodossa.
CSS	Cascading Style Sheet. WWW-standardin mukainen tyylitiedosto.
PDF	Portable Document Format. Yleinen asakirjatiedostoformaatti.

1 Johdanto

Tämän insinöörityön tarkoituksena oli luoda asiakkaan toiveisiin perustuva laitteisto- ja sovelluspaketti, joka mittaisi teknisen laboratorion lämpötilaa, ilmankosteutta ja painetta, kirjaisi mittaustulokset ja näyttäisi käyttäjille laboratorion tilan verkkoympäristössä reaaliajassa.

Työ tehtiin Alshain Oy:lle syksyllä 2016, jossa työskentelin kyseisessä asiakasprojektissa ohjelmistokehittäjän roolissa. Tehtävänäni oli ohjelmoida sulautetulle alustalle Pythonilla ohjelma, joka pyytäisi anturiverkon sensoreilta mittaustuloksia, purkaisi sen ihmiselle luettavaan muotoon ja lokittaisi nämä laitteen muistiin. Kyseinen käytetty anturiverkko on Alshain Oy:n valmistama tuote, joka oli jo asennettu asiakkaan tiloihin.

Lisäksi projetkiin kuului pystyttää samalle sulautetulle alustalle verkkopalvelin ja verkkosivut, joista käyttäjät pystyisivät katsomaan tallenetuista lokeista piirrettyjä kuvaajia eri antureiden lukemista ja pyytämään eri ajanjaksoista raportteja.

Insinöörityö selittää ohjelman rakenteen, kehitysvaiheet ja käytetyt teknologiat ja antaa kuvan lukijalle sulautetun ohjelmistoprojektin kehityksestä.

2 Järjestelmän arkkitehtuuri

Alshain Oy:n anturiverkko on kehitetty tarkkailemaan teknisiä laboratorioita. Tämän takia sensorien on oltava tarkkoja, mutta ennen kaikkea varmatoimisia. Sensorit käyttävät Modbus-protokollaa, jota on käytetty teollisuudessa standardina jo kauan. Itse laitteet ovat kytketty tähän teollisuusväylään käyttäen RS-485-sarjaliikenneväylää.

2.1 Modbus

Projektissa käytetty Alshain Oy:n tuottama anturiverkko toimii Modbus-protokollaa käyttäen, joka on vuodelta 1978. Modbus tiedonvälitys toimii pyyntö-vastaus-pareina, jossa asiakas (sulautettu palvelintoteutus) aloittaa aina kysymys-vastaus-parin. Tämä johtuu siitä, että Modbus on rinnankytketty väylä, jossa jokainen väylään kytketty laite on fyysisesti samassa piirissä kiinni. Pitääkseen väylän vapaana ja minimoidakseen virheitä, palvelimet (yksittäiset anturit) lähettävät ainoastaan vastauksia, kun niille lähetetään kysely. [1; 2; 3.]

Modbus viestintäprotokolla tukee kahta erilaista kehystä viesteille: RTU-formaattia tai ASCII-formaattia. Näiden kahden eri formaatin eroavaisuus on viestien enkoodaaminen Modbus-viestin rakenteeseen. ASCII-formaatissa viestin osat ovat ASCII-muodossa ja viestin tarkistussumma luodaan LRC-tiivistealgoritmilla, kun taas RTU-formaatissa viestin osat ovat binäärinä ja tarkistussumma on luotu CRC-tiivistealgoritmilla. [1; 2; 3.]

Modbus-viesti RTU-muodossa koostuu neljästä osasta, jonka alussa ja/tai lopussa on oltava tyhjää 3,5 merkin verran. Tämä aika lasketaan systeemin tiedonsiirtonopeuden perusteella. Esimerkiksi, mikäli tiedonsiirtonopeus olisi 38400 bittiä sekunissa, tauon pituudeksi tulisi 911 µs. Yhdessä siirrettävässä tavussa sarjaväylässä on kymmenen bittiä: Kahdeksan databittiä ja aloitus- ja lopetusbitti. Täten tauon pituus lasketaan kaavalla $\frac{(3.5*(8+2))bit}{38400\ bit/s} = 911\ \mu s$. [1; 2; 3.]

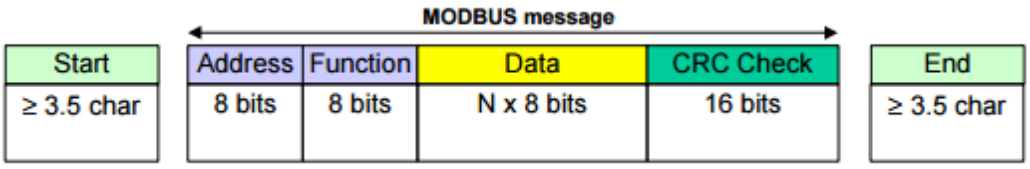
Viesti alkaa kohdelaitteen tunnistenumerolla, joka on kirjattu viestin ensimmäiseen tavuun. Mikäli tunnistenumero on nolla, kaikki väylään kytketyt laitteet lukevat viestin, mutta eivät lähetä vastausta. Yleisimpiä käyttötarkoituksia kaikille laitteille lähtevällä viestillä on laitteiden sammuttaminen ja hätäseis-signaali. Tunnistenumeroita voi täten olla vain 254 kappaletta, mutta johtuen laitteiden lähetinvastaanottimien kuormasta,

väylässä voi yleensä olla enintään 32 laitetta. Tämä johtuu siitä, että väyläajurin äärellisen sisäänottoimpedanssin takia jokainen laite väylässä vuotaa pienen määrän virtaa maahan. [1; 2; 3.]

Viestin seuraavassa lohkossa lähetetään tieto siitä, minkä funktion laitteen tulisi suorittaa. Standardissa on määritetty lista funktiokoodeista ja niiden tarkoituksista. Mikäli vastauksessa funktiokoodin ylin bitti on päällä, tarkoittaa tämä, että on tapahtunut virhe. [1; 2; 3.]

Datalohkossa voidaan kuljettaa haluttu määrä tavuja tietoja, mitä vastaanottaja käyttää funktiosta riippuen. Data voi olla kahdessa eri muodossa: Joko kuudentoista bitin etumerkittöminä kokonaislukuina tai yksittäisinä bitteinä, joita Modbus-standardi kutsuu käämeiksi. Tämä kutsumanimi johtuu siitä, että MODICON-teollisuusautomaatiojärjestelmässä, mitä varten Modbus kehitettiin, yksittäiset bitit kuvasivat todellisuudessa fyysisen laitteen sisällä olevia releitä ja niitten käämejä. [1; 2; 3.]

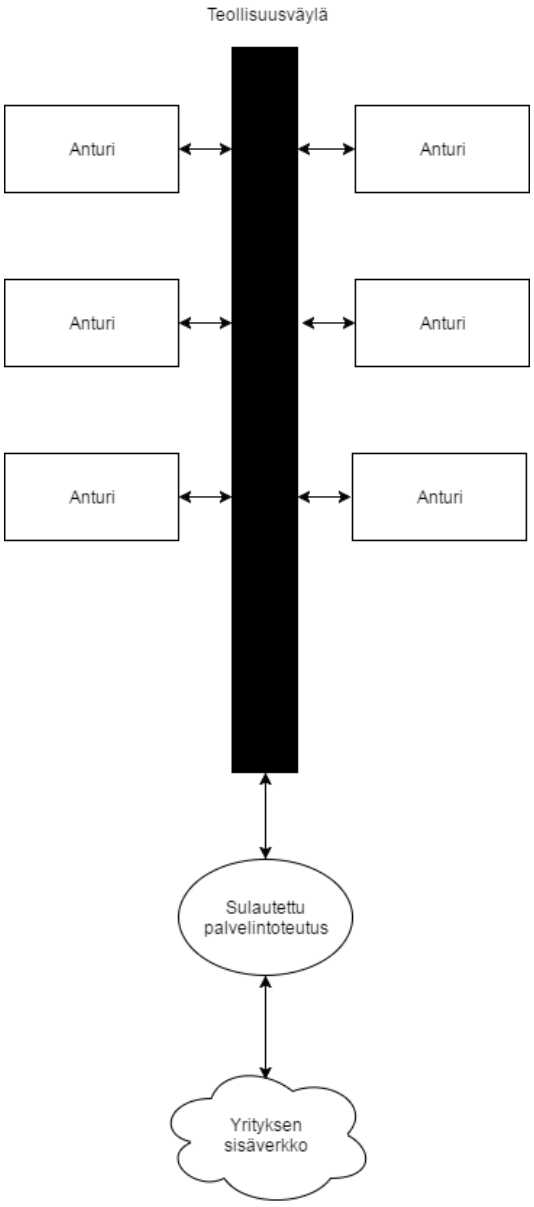
Viimeinen lohko on varattu tarkistussummaa varten, joka on laskettu CRC-algoritmilla viestin perusteella. [1; 2; 3.]



Kuva 1. Modbus RTU-viestin rakenne. [2]

2.2 RS-485

Modbus-viestit kuljetettiin RS-485-sarjaliikenneväylää käyttäen Modbus Serial -standardin mukaisesti. Tämä standardi on peräisin vuodelta 1983, ja se on suunniteltu häiriösietoiseksi. Väylä toimi erotusmuotoisesti: Kaapelin sisällä kulkee kaksi johdinta, ja digitaalisen signaalin arvo määräytyy näitten kahden johtimen jänniteiden erotuksesta. Lisäksi tämä standardi on protokollaneutraali sekä rinnankytkeytyvä.



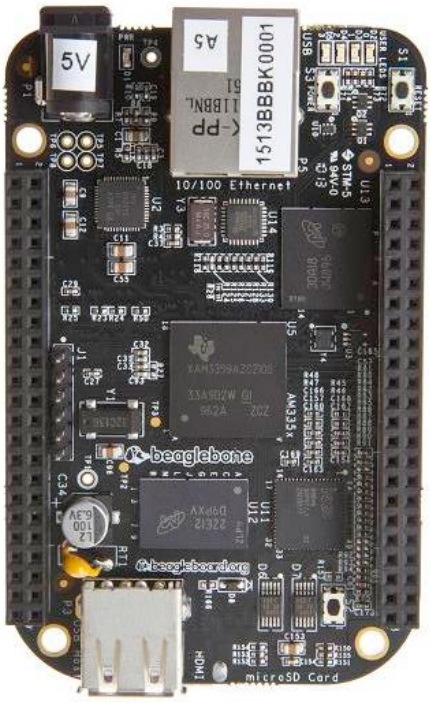
Kuva 2. Anturiverkon arkkitehtuuria kuvaava kaaviokuva.

3 Laitteisto ja sen asettamat rajoitukset

Projektin suunnitteluvaiheessa iteroitiin useaan otteeseen ohjelmiston fyysistä alustaa. Alustava suunnitelma oli toteuttaa koko toiminnallisuus erillisellä mikrokontrollerisirulla (esim. STM32F7xx Cortex M7), mutta tällöin tuotteen kehitykseen olisi kulunut moninkertainen aika verrattuna enemmän kehittäjäystävällisempiin ratkaisuihin (COTS-tuotteet, kuten Arduino ja BeagleBone).

3.1 Laitteisto

Teknisen määrittelyn mukaan sulautetun palvelimen alustan piti olla yhden piirilevyn tietokone tai SoC pohjainen tietokone, joka olisi tarpeeksi tehokas pyörittämään kevyttä UNIX-pohjaista käyttöjärjestelmää. Tähän tehtävään valittiin BeagleBone Black, yhden piirilevyn tietokone. Tämä alusta valittiin kustannustehokkuuden, fyysisen koon ja tehonkulutuksen takia. Tuotteen määritelmän ja asiakkaan toiveiden mukaisesti tavoitteena oli rakentaa ulkoinen, erillinen laite, joka olisi yhteydessä asiakkaan sisäverkkoon toimien rajapintana sisäverkon ja anturiverkon välillä.



Kuva 3. BeagleBone Black. [4]

BeagleBone Black valittiin tämän kahden yleisimmän kilpailijan, Raspberryn ja Arduinon, yli kahdesta syystä: Arduino on resurssirajoitteinen ja eikä siinä ei ole Ethernet-liitäntää, mitä tarvittiin asiakkaan sisäverkkoon kytkeytymistä varten, ja Raspberryssä virtaliittimenä toimii vain micro-USB-liitin. Micro-USB:hen sopivia virtalähteitä oli vaikea löytää, jotka riittäisivät projektin vaatimuksiin. Lisäksi micro-USB-liitin on mekaanisesti paljon epävarmempi kuin BBB:n 2,5 mm :n virtaliitin.

BBB:n käyttöjärjestelmäksi valittiin Debian-jakelupaketti Linux-käyttöjärjestelmästä. Debian on yksi vanhimpia ja varmimpia Linuxin jakelupaketteja. Lisäksi sitä pystytään ajamaan ARM-arkkitehuurisilla prosessoreilla, jollainen on BBB:ssä.

Taulukko 1. Merkittävät erot BeagleBone Blackin ja STM32F7-sarjan Cortex-M7-prosessorin välillä projektinkehityksen kannalta.

Ominaisuus	**BBB (ARM Cortex-A8)**	**STM32F7xx Cortex-M7**
Arkkitehtuuri	ARM	ARM
Kellotaajuus	600 Mhz – 1 GHz	216 MHz
Luotettavuus	Välttävä. Käyttöjärjestelmä ja moniajo voivat aiheuttaa kriittisillä hetkillä ongelmia.	Hyvä. Kehittäjä itse ohjelmoi kaiken prosessorilla suoritettavan logiikan.
Hinta	40 € - 70 €	14 € - 18 €
Rajapinnat	Linux, standardikirjastot	Itse kehitettävä – ei valmiita

3.2 Rajoitukset

Tekniset rajoitukset, jotka ohjasivat ohjelmistokehitystä, olivat BeagleBonen sisäinen flash-muisti ja laskentateho. Laitteen kerätessä kymmenen sekunin välein tietoja kuudelta mittauspisteeltä, missä jokaisessa oli neljä anturia, ja kirjoittaen nämä flash-muistiin, tämän piti myös tarjota lokitetut mittaustulokset jokaiselle käyttäjälle, joka lataisi verkkosivuston. Valitettavasti Flash-muistit eivät tue samanaikaista luku- ja kirjoitusoperaatioita eivätkä ne tekniikkansa takia kestä kovinkaan monta kirjoitusoperaatiota verrattuna tyypilliseen kiintolevyyn. Tämä on yleinen ongelma yhden piirilevyn tietokoneilla, sillä nämä käyttävät lähes poikkeuksetta aina tallennustilana joko sisäistä tai ulkoista flash-muistia (yleensä SD-kortin muodossa).

$$6_{sensoria} * 4_{anturia} * 6_{mittausta\ minuutissa} = 144\ mittausta\ minuutissa$$
$$144_{mittausta\ minuutissa} * 60s = 8640\ mittausta\ tunnissa$$
$$8640_{mittausta\ tunnissa} * 24h = 207360\ mittausta\ vuorokaudessa$$

Kuvio 1. Mittausten lukumäärä minuutissa, tunnissa ja vuorokaudessa.

BBB:n piti pystyä myös kaiken tämän lisäksi pyörittämään verkkopalvelinta, joten laskentatehon säästämiseksi piti myös keksiä keino.

Näitten ongelmien ratkaisemiseksi alustettiin BBB:lle RAM disk -osio, mihin tasaisin väliajoin siirretään mittauslokeja, jotka esitetään verkkosivustolle saapuvalle käyttäjälle. Näin säästetään flash-muistin luku- ja kirjoitusoperaatioita, jolloin laitteen käyttöikä moninkertaistuu. Tämä myös poistaa kuormitusta prosessorilta, sillä tämän ei tarvitse suorittaa levyltälukua, vaan tämä hakee kaiken muistista. Lisäksi tämä mahdollistaa suuremman I/O-nopeuden, mikä tässä työssä ei tosin ollut pullonkaulana.

Condition	Part Number			Units
	MTFC2GMDEA-0M WT MTFC4GLDEA-0M WT	MTFC4GMDEA-1M WT MTFC8GLDEA-1M WT	MTFC16GJDEC-2M WT MTFC32GJDED-3M WT MTFC64GJDDN-3M WT	
Sequential write	6.6	13.5	20	MB/s
Sequential read	30	44	44	MB/s
Random write	90	90	90	IOPs
Random read	1080	1080	1100	IOPs

Kuva 4. BBB:n Flash-muistin I/O-nopeus. Käytetyssä BBB:ssä oli MTFC8GLDEA-1M WT eMMC muistia. [6]

Viimeinen ongelma, joka tuli ratkaista, oli BBB:n sisäinen kello, sillä tässä ei ole paristovarmenteista reaaliaikakelloa. Aina käynnistyessään BBB unohtaa kellonaikansa palauttaen sen oletusaikaansa. Tätä varten BBB:n yhdyskäytäväkoneelle asennettiin NTP-palvelu, josta BBB haki nykyisen kellonajan.

Tarkkaa kellonaikaa tarvittiin luonnollisesti mittaustulosten kirjaamiseen ja arkistointiin.

4 Tiedonkeruuohjelmisto

4.1 Teknologioitten valinnat

4.1.1 Python

Anturirajapinta toteutettiin Python-ohjelmointikielellä helpottamaan kehitysprosessia, mikä nopeuttaisi tätä ja sekä Pythonin siirrännäisyyden takia. Python on korkean tason ohjelmointikieli, joka ei tarvitse kääntämistä suorittamista varten, vaan tätä ajetaan rivi riviltä komentotulkin läpi.

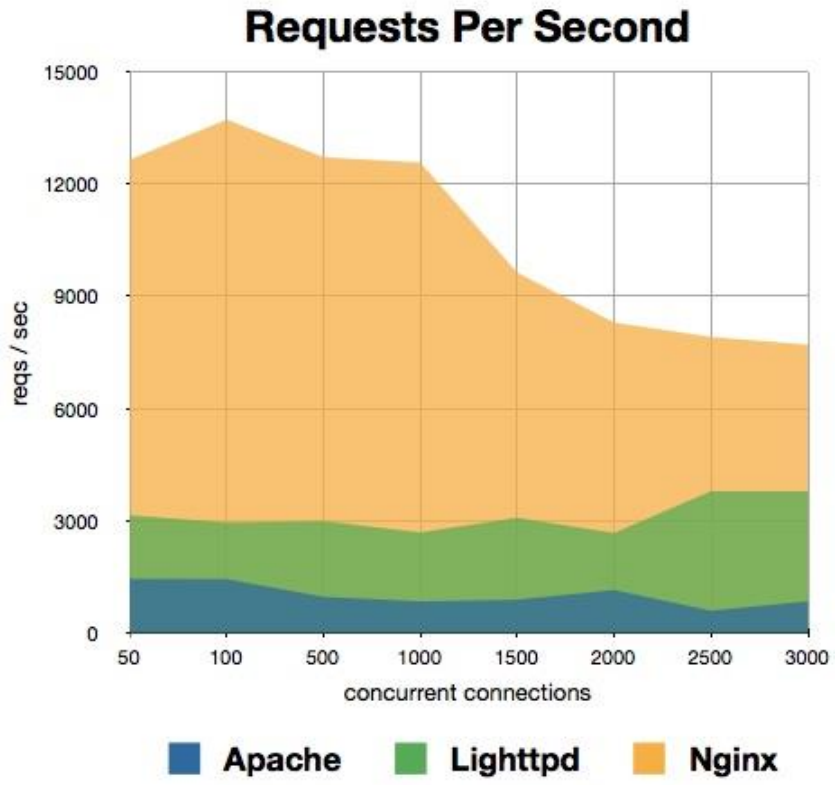
Koska Pythonia suoritetaan komentotulkin läpi, poistuvat fyysiseen laitteistoon sidotut rajoitukset, kuten esimerkiksi prosessorin arkkitehtuuri. Tämän takia anturirajapinta ja verkkopalvelin voitiin kehittää ja testata normaalilla PC-tyylisellä työpisteellä käyttöjärjestelmästä riippumatta.

4.1.2 Nginx

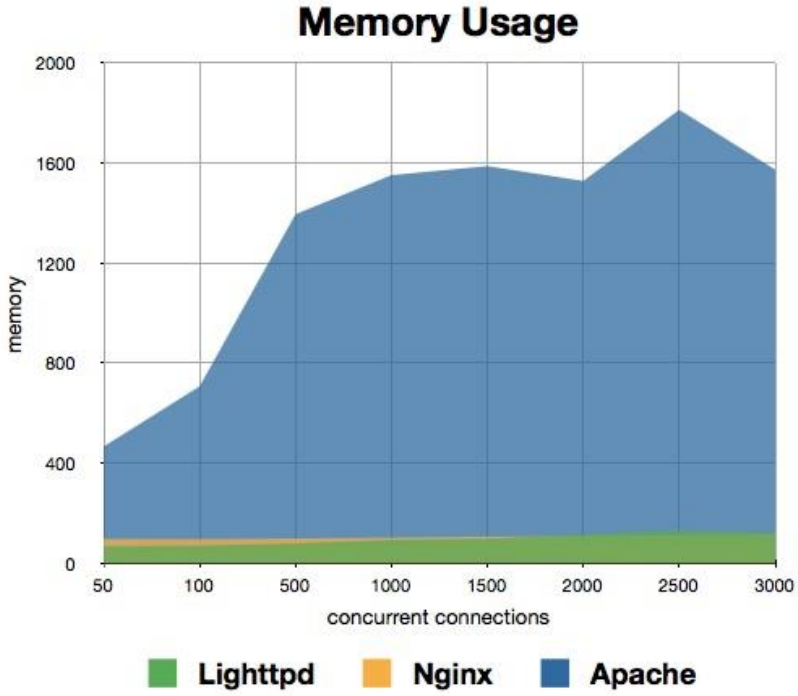
Verkkopalvelimeksi valittiin vapaaseen ja avoimeen lähdekoodiin perustuva Nginx. Verkkopalvelintoimintonsa lisäksi Nginx:ää voidaan käyttää myös käänteisenä välityspalvelimena. Tätä nimenomaista ominaisuutta käytettiin hyödyksi tarjoilemaan lokitettua mittausdataa verkkosivua ladatessa. [5; 7.]

Nginx on suunniteltu nimenomaisesti päihittämään Apachen HTTP-palvelin suorituskyvyssään, missä Nginx on onnistunut merkittävän onnistuneesti. Tomcatin muistinkäyttö jokaista samanaikaista yhteyttä kohti on yli kolmenkertainen suurilla käyttäjämäärillä. Pyyntöjen käsittelyissä Nginx on noin kahdeksan kertaa nopeampi kuin Apachen, kuten kuvasta 5 selviää. Nginx:n käyttöönotto ja konfigurointi on myös suunniteltu hyvin nopeaksi, mikä auttoi projektin testaamisessa eri alustoilla. [5; 7.]

Tämän takia Nginx soveltui ratkaisuksi. Alhaisen prosessori- ja muistinkäyttönsä takia tämä soveltui täydellisesti sulautetulle alustalle ajettavaksi.



Kuva 5. Kuvaaja Nginx:n kapasiteesta käsitellä pyyntöjä. [5]



Kuva 6. Kuvaaja Nginx:n muistinkäytöstä yhteyksiä kohtaan. [5]

4.2 Anturirajapinta ja lokitus

4.2.1 Kyselyn lähettäminen ja vastaanottaminen

Itse ohjeman rakenne ja arkkitehtuuri ei ollut monimutkainen. Ohjelma sisälsi yhden luokan, joka sisälsi kommunikoinnin yksittäisen anturin kanssa, joita instantioitiin yksi jokaista anturia kohden, ja silmukan, joka suoritti antureiden lukemista kymmenen sekunnin välein.

```
class SensorReader:
    def __init__(self,port,baudrate,to,ibto,modbus_id,unit_name,offsets):
        self.port = port
        self.com = serial.Serial( port,baudrate,timeout = to,inter_byte_timeout = ibto )
        self.unit_name = unit_name
        self.modbus_id = modbus_id
        self.offsets = offsets
        self.error_count = 0
        self.error_flag = False
```

Kuva 7. Anturinlukijaluokan alustus.

Jokaiselle oliolle annetaan seuraavat tiedot:

- Tunnistenumero portista, joka on kytketty koneeseen. Viestit lähetetään ja vastaanotetaan tästä portista.
- Baud kertoo, kuinka monta symbolia sekunnissa portista lähetetään ja vastaanotetaan.
- Katkaisuaika määrittää, kuinka kauan vastausta sensorilta odotetaan, ennen kuin yhteys katkaistaan.
- Tavujen välinen tauko. Tämä arvo kuvaa, kuinka pitkä väli on Modbus-viestin loppumerkin ja seuraavan MODBUS-viestin aloituksen välillä. Modbus-protokollassa tämä on vähintään 3,5 tavua.
- Sensorin tunnistenumero, jonka avulla voidaan yhdistää saadut mittaustulokset tiettyyn sensoriin.
- Sensorin nimi on valinnainen, mikäli sensorille halutaan asettaa nimi tunnistenumeron lisäksi.
- Valinnaiset kertoimet, joilla painotetaan mittaustuloksia, mikäli tämä on tarpeellista.

Ainut viesti, minkä ohjelmiston piti osata rakentaa ja lähettää, oli sensorin rekisterin lukeminen. Viestin osat enkoodattiin tietueeseen formaattiin ”>BBHH”, missä ”>” tarkoittaa big-endian-tavujärjestystä (eli merkitsevimmät bitit tallennetaan ensin), ”B” etumerkitöntä esitystä kirjaimesta ja ”H”, mikä tarkoittaa etumerkitöntä lyhyttä kokonaislukua. [8]

Viestistä lasketaan CRC-tiivistealgoritmilla tarkistussumma, millä varmistetaan, että viesti vastaanotettiin kokonaisena ja eheänä perille.

Viestin lähettämiseen sarjaportin yli käytettiin Pythonin ”serial”-kirjastoa, mikä pitää sisällään portin rekisteröimisen, sen kuuntelemisen ja siihen datan lähettämisen. Tietueita ei varsinaisesti ole eksplisiittisesti Pythonissa, joten näitä varten käytettiin Pythonin ”struct”-kirjastoa.

```
def create_read_register_message(self,addr,amount ):
    #Device id 8 bits, function code 8 bits, address 16 bits, amount of registers read 16 bits
    msg = struct.pack( ">BBHH",int(self.modbus_id),READ_HOLDING_REGISTERS,addr,amount )
    checksum = self.calc_crc( msg )
    msg += struct.pack( "<H",checksum )
    return msg
```

Kuva 8. Viestin luominen.

```
def calc_crc(self, msg ):
    crc = 0xFFFF
    for ch in msg:
        crc ^= ch
    for i in range( 8 ):
        if (crc & 0x0001) != 0:
            crc >>= 1
            crc ^= 0xA001
        else:
            crc >>= 1
    return crc
```

Kuva 9. Tarkistussumman laskeminen CRC:llä.

Koska Modbus-protokolla toimii pyyntö-vastaus-pareina, viestin lähetysfunktio rakennettiin samalla tavalla. Funktiolle syötetään valmisteltu viesti, ja tämä palauttaa vastauksen monikkona.

Sensorit tarjoavat mittaustiedot kuudentoista bitin rekistereinä neljää ensimmäistä rekisteriä lukuunottamatta, jotka on yhdistetty muodostamaan kaksi 32:n bitin rekisteriä. Kun mittaustulokset on saatu purettua, tulokset pitää jakaa sadalla, sillä sensorit tallentavat kaikki mittaukset sadasosina kokonaislukuina.

```
def send(self,msg):
    if not error_flag:
        done = False
        while not done:
            print("writing:" +repr(msg))
            self.com.write(msg)
            self.com.flush()
            response = self.com.read(256)
            done = len(response) > 0
            print( "reading:" + repr(response) )
        response_function_code = response[1]
        if response_function_code & 0x80 > 0:
            print("Sensor returned an error code")
            sys.exit(1)

        bytecount = response[2]
        payload = response[3:-2]
        registers = struct.unpack( ">%iH"%(bytecount/2),payload )
        (t0,p,rh,t1) = struct.unpack( ">IIHH",payload )

        t0 /= 100.0
        t0 += self.offsets[0]

        p /= 100.0
        p += self.offsets[1]

        rh /= 100.0
        rh += self.offsets[2]

        t1 /= 100.0
        t1 += self.offsets[3]

        return (t0,p,rh,t1)
```

Kuva 10. Viestin lähetys ja vastauksen käsittely

Kun viesti on syötetty sarjaporttiin kulkeutuvaan *output-stream*iin, portin puskuri tyhjennetään, mikä estää mahdollisuuden ylimääräisten tavujen lukemiseksi. Tämän jälkeen portista luetaan enintään 256 tavua toistuvasti, kunnes tavujen välinen tauko

löydetään. Modbus-protokollan mukaan Modbus-viesti voi olla korkeintaan 256 tavua pitkä.

Mikäli vastausken funktiokoodissa on ylin bitti ylhäällä, viestin purku lopetetaan, ja lähetetään uusi rekisterinlukuviesti. Sensori on vastaanottanut viestin ja lähettänyt vastauksensa, mutta rekisterinlukuviestissä on ollut virhe.

Mikäli sensori ei ikinä lähetä vastausta, tai vastaus on vain satunnaisia tavuja, on tapahtunut vakavampi virhe. Virhe voi syntyä muutamasta syystä: elektronisesta häiriöstä sarjaväylässä (ulkopuolinen sähkökenttä indusoi häiriövirtoja tai ryömintää käyttöjännitteessä), sensori ei saa yhteyttä yksittäisiin antureihinsa, sarjaporttiajurin häiriöstä (bugista) tai siitä, että jokin toinen sensori lähettää viestiä. Viimeisin näistä virheistä on kaikkein vakavin, sillä RS-485-sarjaväylä on vuorosuuntainen (eng. half-duplex), mikä tarkoittaa, että jokin sensoreista rikkoo systeemin arkkitehtuuria.

Mikäli virhekoodia ei ole, voidaan viestissä ollut tietue purkaa samalla enkoodauksella kuin viestiä luodessa. Purettu tietue sijoitetaan monikkoon ja tämän arvoihin lasketaan poikkeamat, mikäli näitä oli asetettu, minkä jälkeen arvot palautetaan monikkona.

Ohjelman käynnistyessä sensorit ladataan JSON-tiedostosta käyttäen Pythonin ”json”-kirjastoa. JSON-tiedossa on listattu jokainen sensori tämän omine asetuksineen. Jokaisesta sensorista listataan tämän Modbus-tunnistenumero, sensorin nimi, sensorin tyyppi ja tiedonsiirtonopeus (mikäli nämä halutaan erikseen kirjata) sekä poikkeamat. Mikäli ohjelmistorakenteessa ei ole omaa kansiota sensorille, ohjelma luo kansion sensorin tunnistenumeron perusteella, minne mittaustulokset kirjataan.

```
readers = []

with open("sensorconf.json") as fh:
    conf = json.load(fh)
    for sensor in conf['devices']:
        readers.append( SensorReader("/dev/ttyUSB0",38400,4,0.25,str(sensor['modbus_id']),sensor['name'],sensor['offsets']) )
        path = os.path.join("results",str(sensor['modbus_id']))
        if not os.path.exists(path):
            os.makedirs(path)
```

Kuva 11. Sensorilukijoitten instantointi JSON-tiedoston perusteella.

```
{
    "devices": [{
        "modbus_id": 250,
        "name":"testi",
        "unit_type":"",
        "bitrate":0,
        "offsets":[0.0,0.0,0.0,0.0]
    }]
}
```

Kuva 12. Esimerkki JSON-tiedostossa, missä konfiguroidaan sensori.

Ohjelman pääsilmukka toistettiin kymmenen sekunnin välein. Jokainen sensori käydään läpi, luodaan rekisterinlukuviesti, tarkastetaan päivämäärä oikeaksi, jottei mittaustulosten lokituksessa ilmene virheitä, lähetetään pyyntö ja vastaanotetaan vastaus, joka kirjataan CSV-tiedostoon. CSV-tiedostot arkistoidaan tiedostorakenteeseen, jonka kansiot on nimetty päivämäärien mukaan. Hakemistorakenne on muotoa tunnistenumero / vuosi / kuukausi / päivä.

```
while not done:
    for reader in readers:
        msg = reader.create_read_register_message(1000,6)
        time.sleep(0.1)
        curtime = datetime.datetime.today()
        if curtime > MIN_TIMESTAMP:
            dirpath = os.path.join("results",reader.modbus_id,str(curtime.year),str(curtime.month))
            filename = str(curtime.day)+".csv"
            if not os.path.exists(dirpath):
                os.makedirs(dirpath)
            filepath = os.path.join(dirpath,filename)
            with open(filepath, "a" ) as handle:
                handle.write(curtime.strftime("%Y-%m-%d %H:%M:%S")+","+",".join(["%.2f" % v for v in reader.send(msg)]) + "\n" )

    time.sleep(10.0)
```

Kuva 13. Anturilukijan pääsilmukka

Tämä arkistointitapa valittiin tämän kustannustehokkuuden takia. BBB:n laskentateho on rajattu, joten erillisten tietokantaohjelmistojen katsottiin olevan turhia näitten resurssinkulutuksen takia. Lisäksi nämä pitäisivät kaiken datan laitteen massamuistissa, jolloin ongelmaksi jälleen kerran tulisi BBB:n massamuistin kuluminen.

4.2.2 Mittaustulosten siirtäminen RAM diskille

Taustalla pyörii erillinen ohjelma, joka 30 sekunnin välein siirtää tuhat viimeisintä mittaustulosta RAM diskille, josta verkkopalvelin tarjoaa nämä käyttäjälle.

Hakemistorakenteesta selvitetään kahden apufunktion avulla oikeat kansiot, mistä mittaustulokset haetaan. Tässä käytetään apuna Pythonin ”os”-kirjastoa. Nämä mittaustulokset ylikirjataan CSV-tiedostoon, jota palvelin käyttää muodostamaan Dygraphs-kirjastolla kuvaajat verkkosivuilla.

CSV valittiin tallenusformaatiksi tämän yksinkertaisuutensa vuoksi. Teknisen määrittelyn mukaan mittaustulokset tuli tallentaa ASCII-muodossa, jotta mikä tahansa UNIX:in komentorivityökalu osaisi suoraan lukea näitä. CSV-tiedoston rakenne on taulumainen: Tiedoston ensimmäiselle riville merkitään eri kenttien otsikot, jotka kuten tiedostoon tallennettava data, erotetaan pilkuilla. Tiedostoon kirjataan data yksi taulun rivi kerrallaan ja nämä rivit erotetaan toisistaan rivinvaihdoilla.

CSV-tiedostojen lukemiseen ja kirjoittamiseen löytyy useita eri kolmannen osapuolen kirjastoja, mutta tehokkaammaksi katsottiin tehdä yksinkertainen kirjoitusoperaatio itse (kts. kuva 14).

```
while True:


    for modbus_id in sorted(os.listdir("results/")):
        lines=[]
        formatted_lines=[]
        filepaths = list_all_files( os.path.join("results",modbus_id) )
        sortedFiles = sorted( filepaths, key = humanist_date_mangler )

        if len(sortedFiles) > 1:
            with open( sortedFiles[-2],"r") as fh:
                for line in fh:
                    line = line.strip()
                    lines.append(line)

                for line in lines:
                    formatted_lines.append(line.strip().split(','))

            with open( sortedFiles[-1],"r") as fh:
                for line in fh:
                    line = line.strip()
                    lines.append(line)

                for line in lines:
                    formatted_lines.append(line.strip().split(','))
        else:
            with open( sortedFiles[0],"r") as fh:
                for line in fh:
                    line = line.strip()
                    lines.append(line)

        formatted_lines = formatted_lines[-1000:]


        path = os.path.join(os.path.join("/mnt","ramdisk",modbus_id))
        if not os.path.exists(path):
            os.makedirs(path)

        with open(os.path.join("/mnt","ramdisk",modbus_id,"t0.csv"), "w") as t0:
            t0.write("Time,T0"+'\n')
            for line in formatted_lines:
                t0.write(line[0]+','+line[1]+'\n')

        with open(os.path.join("/mnt","ramdisk",modbus_id,"t1.csv"), "w") as t1:
            t1.write("Time,T1"+'\n')
            for line in formatted_lines:
                t1.write(line[0]+','+line[4]+'\n')

        with open(os.path.join("/mnt","ramdisk",modbus_id,"p.csv"), "w") as p:
            p.write("Time,P"+'\n')
            for line in formatted_lines:
                p.write(line[0]+','+line[2]+'\n')

        with open(os.path.join("/mnt","ramdisk",modbus_id,"rh.csv"), "w") as rh:
            rh.write("Time,RH"+'\n')
            for line in formatted_lines:
                rh.write(line[0]+','+line[3]+'\n')

    time.sleep(30.0)
```

Kuva 14. Silmukka, missä mittaustulokset siirretään Debianiin alustetulle RAM diskille.

```
def list_all_files( root ):
    out = []
    files = os.listdir( root )
    for fn in files:
        path = os.path.join( root, fn )
        if os.path.isdir( path ):
            out.extend( list_all_files( path ) )
        else:
            out.append( path )
    return out
```

Kuva 15. Apufunktio minkä avulla selvitetään päivämäärän perusteella oikea kansio

```
def humanist_date_mangler( path ):
    parts = path.split('.')[0].split("/")
    return "-".join(["%02i" % int(p) for p in parts[-3:]])
```

Kuva 16. Apufunktio millä saatiin muutettua päivämäärä eri muotoon.

4.3 Verkkopalvelin ja verkkosivut

4.3.1 Verkkopalvelin

Kuten luvussa 4.1.2 selvennettiin, palvelinteknologiaksi valittiin Nginx.

Itse Nginx:n käyttöönotto ja asetusten asettaminen on helppoa, minkä takia tämä soveltui parhaimmaksi työkaluksi projektiin. UNIX-pohjaisessa käyttöjärjestelmässä Nginx löytyy paketinhallintajärjestelmästä, ja tämän asentamisen jälkeen tarvitaan vain asetustiedoston mukauttaminen projektin määritelmien mukaisesti.

Vaatimusmäärittelyn mukaisesti palvelimelle ei tarvinnut implementoida käyttöoikeuksien hallintaa tai erillisiä käyttäjiä, sillä palvelin ei ole kytköksissä ulkoiseen verkkoon, joten ainut toteutettu asetus oli käänteisen välityspalvelimen asettaminen ja kohdistamaan sen haluttuun hakemistoon, missä mittausdata sijaitsi.

Nginx-asetusten rakenteessa *location* merkitsee URL:n, mikä kautta laukaistaan haluttu toiminto. Ohjaus verkkosivulle tapahtuu yksinkertaisesti antamalla HTML-tiedoston nimi *index*-kenttään. *Alias*-kenttä määrittää polun palvelimen hakemistorakenteesta, mihin annettu *location*-polku ohjaa. Tähän hakemistoon siirrettiin näytettävät mittaustiedot. Ngnix antaa mahdollisuuden tallentaa dataa välimuistiin, joten *Cache-control*-asetus

asetettiin henkilökohtaiseksi, mikä tallentaa polusta haetun datan jokaisen käyttäjän selaimen välimuistiin.

```
location / {
    root   html;
    index  index.html index.htm;
}

        location /graphs/ {
                alias  /mnt/ramdisk/;
                expires 0;
                add_header      Cache-Control private;
        }

        location /reporting/ {
                alias  /usr/share/nginx/html/results/;
                expires 0;
                add_header      Cache-Control private;
        }
```

Kuva 17. Mittaustietojen polun asettaminen Nginx-palvelimelle, jotta käänteinen välityspalvelin toimisi.

4.3.2 Sivujen muodostaminen

Mikäli sensoreita lisätään tai poistetaan anturiverkostaverkosta, tai sensoriasetuksia muutetaan, palvelun verkkosivut on luotava uudelleen näyttämään muutokset. Tämän takia rakennettiin ohjelma, joka muodostaa tarvittavat sivut valmiiksi rakennetuilta pohjilta. Pohjissa on HTML-rakenteessa avainsanoja, mitkä korvataan haluttavilla HTML-elementeillä tai JavaScript-koodilla.

```
#Store templates to memory
with open ("indexcharttemplate.html","r") as fh:
    index = fh.read()

#For index
with open("graphconf.json","r") as fh:
    conf = json.load(fh)
    for group in conf['groups']:
        row ='<h3 class="title-row"><b>'+group['name']+'</b></h3><br><br>' + "\r\n"
        for graph in group["graphs"]:
            group['name'] = group['name'].replace( " ", "_" )
            graphId = group['name'] + str(graph['id']) + graph['type']
            tmp = index
            tmp = tmp.replace("title",graph['title'])
            tmp = tmp.replace("graphId",graphId)
            tmp = tmp.replace("prefix",graphId)
            tmp = tmp.replace("source",graph["source"])
            row += tmp
            listing_rows.append(row)
            row = ""
            graphs.add(group['name']+"-"+str(graph['id']))
```

Kuva 18. HTML-pohjan täyttö.

Sivulla näytettävät kuvaajat kirjataan samanlaiseen JSON-asetustiedostoon kuin sensorit. Tiedostoon kirjataan piirrettävät kuvaajat ryhmittäin ja jokaisesta kuvaajasta kerrotaan tämän tyyppi (lämpötila, paine, ilmankosteus), otsikko, kuvaajan akselien kuvaus, mistä tiedostosta mittaustulokset haetaan ja sensorin tunnistenumero.

```
{
    "groups": [{
        "name":"TestGroup1",
        "graphs":[
        {
            "id":250,
            "type":"t0",
            "title":"Temperature 0",
            "source":"/graphs/250/t0.csv",
            "legend":"Temperature [C]"
        },
        {
            "id":250,
            "type":"p",
            "title":"Pressure",
            "source":"/graphs/250/p.csv",
            "legend":"Pressure [mbar]"
        },
        {
            "id":250,
            "type":"rh",
            "title":"Relative Humidity",
            "source":"/graphs/250/rh.csv",
            "legend":"Humidity [%]"
        }]
    },{
        "name":"TestGroup2",
        "graphs":[
        {
            "id":12,
            "type":"t0",
            "title":"Temperature 0",
            "source":"/graphs/12/t0.csv",
            "legend":"Temperature [C]"
        },
        {
            "id":12,
            "type":"p",
            "title":"Pressure",
            "source":"/graphs/12/p.csv",
            "legend":"Pressure [mbar]"
        }]
    }
    ]
}
```

Kuva 19. Esimerkki verkkosivujen kuvaajien asetustiedostosta. Kuvasta käy ilmi JSON:in rakenne: Arvot ovat avain-arvo-pareina rakenteissa.

```
<div class="listing-row">
    <h3>title</h3><br>
    <div class="chart" id="graphId">
        <script type="text/javascript">
            var graphId =   new Dygraph(document.getElementById("graphId"),"source",
                            {ylabel:"legend",width:"600",yAxisLabelWidth:70,xAxisHeight:30});
            function prefixgraph()
            {
                setTimeout(function()
                {
                    graphId = new Dygraph(document.getElementById("graphId"),"source",
                            {ylabel:"legend",width:"600",yAxisLabelWidth:70,xAxisHeight:30});

                },47000);
            }
            prefixgraph();
        </script>
    </div>
</div>
```

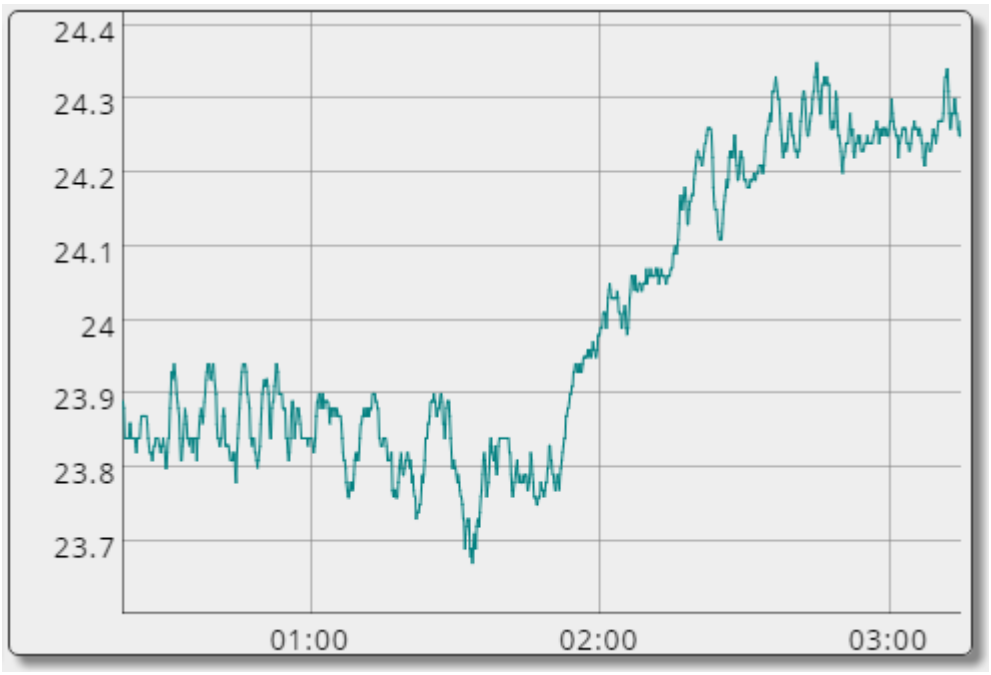
Kuva 20. Käytetty HTML-pohja etusivun kuvaajista. Korvattavat avainsanat tässä olivat ”title”, ”graphId”, ”source”, ”legend” ja ”prefix”.

4.3.3 Verkkosivut

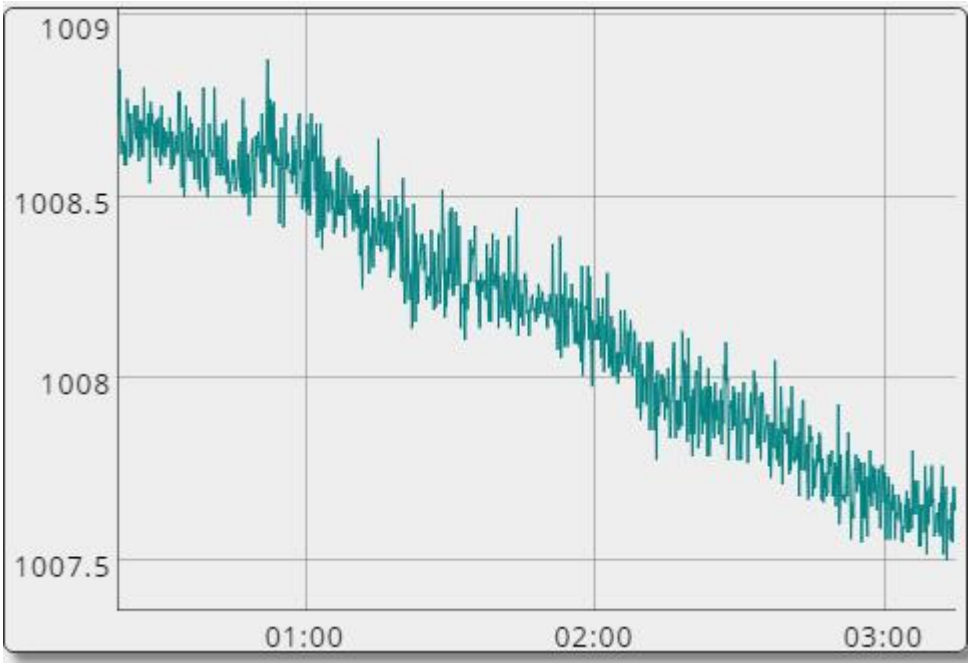
Verkkosivujen teknillinen toteutus haluttiin pitää mahdollisimman yksinkertaisena. Ainut ulkopuolinen kirjasto, mikä liitettiin sivustoon oli Dygraphs-kuvaajakirjasto. Näitten kuvaajien muodostamiseen, ja esitettävien tietojen hakemiseen palvelimelta ohjelmoiltiin lyhyehkö JavaScript-skripti.

Dygraphs on avoimen lähdekkoodin JavaScript-kirjasto, jolla pystytään esittämään erilaisia aikasarjakuvaajia käyttäjän mieltymysten mukaisesti. Dygraphs-kuvaajalle voidaan syöttää piirrettävän kuvaajan data monenlaisessa formaatissa, joista yksi on CSV-formaatti, jota tässä työssä käytettiin. Sivustolla kuvaajat esitetään ryhmissä, jotka on määritelty kuvan 19 mukaisesti sivujen luontiasetustiedostoon. [9]

Sivuston ulkoasu toteutettiin myös ilman ulkopuolisia resursseja. Teknisen määrittelyn mukaisesti sivuston tuli olla yksinkertainen, kevyt ja selkeä. Valmiita kehyksiä tai tyylisivuja ei käytetty, vaan sivuston elementtien rakenne ja tyylisivut rakennettin itse.



Kuva 21. Esimerkki Dygraphs-kuvaajasta. Tässä esimerkissä mitattiin lämpötilavaihtelua yöllä.



Kuva 22. Esimerkki Dygraph-kuvaajasta. Tässä esimerkissä mitattiin ilmanpaineen vaihtelua (millibareina) yöllä.

4.3.4 Raporttien luominen

Raporttinäkymä toteutettiin käyttämällä hyväksi oletusnäkymän rakennetta.

Raportointisivulla käyttäjä valitsee aloitus- ja lopetuspäivämäärän ja kaikkien antureitten mittaustulokset piirretään suurempina kuvaajina, jossa aika-akseli on kahden valitun päivämäärän pituinen. Käyttäjä voi halutessaan tallentaa tämän PDF-muodossa.

5 Jatkokehitys

Tuotepakettia lähdettiin suunnittelemaan ja toteuttamaan tulevaa jatkokehitystä silmälläpitäen.

Tulevaisuuden tavoitteena olisi, että anturirajapinta tarjoaisi mahdollisuuden lukea monien muiden valmistajien sensoreita kuin vain Alshain Oy:n tuottamia. Asetustiedoston rakennetta muutettaisiin siten, että tiedonsiirtotapa tallennetaan tiedostoon ja antureille luodessa olioita, instansoidaan oikeanlainen. Tällöin itse ohjelmistoa voitaisiin myydä erillisenä itse tuotepaketista.

Toinen haluttu ominaisuus olisi karttakuva anturien sijoittelusta. Asiakas tarjoaisi pohjapiirustuksen ja sijainnit antureille, minkä perusteella sivustolle räätälöitäisiin käyttöliittymä tämän pohjakuvan avulla, missä yksittäisiä antureita voisi seurata klikkaamalla tämän ikonia kartalla.

Raporttien luomista varten jatkokehityksessä suunniteltiin erilaisia rajaustapoja: Käyttäjä voisi valita tietyt anturit näitten ryhmien, tyypin tai käyttöttarkoituksen mukaan, tai yhdistelmänä näistä kaikista.

6 Tulokset ja yhteenveto

Insinöörityön päätteeksi todettiin, että tuotettu ohjelmisto- ja laitteistopaketti oli toimiva ja toteutti teknillisen määritelmän täyttämät tavoitteet. Asiakkaan asettamat tavoitteet täyttyivät suunnitellusti ja hyväksytysti.

Suurin ongelma, mihin projektin edetessä törmättiin, oli Flash-muistin hitaus ja kuluminen, minkä johdosta siirryttiin RAM disk-pohjaiseen ratkaisuun (kts. luku 3.2). Lisäksi kohtalaiseksi ongelmaksi ilmeni BBB:n virtaliittimen mekaaninen väljyys.

Ideaalitilanteessa, missä aikaa olisi ollut rajattomasti, käyttöliittymän ulkoasua ja funktionalisuutta olisi voinut parantaa selkeästi. Eri valkointi- ja/tai suodatustoimintoja mittaustulosten rajaamista varten oltaisiin voitu kehittää, mikäli kehitysaikaa olisi ollut jäljellä. Nämä ominaisuudet siirtyivät jatkokehityksen puolelle.

Projektin jälkikatselmuksessa päästiin johtopäätökseen, että kyseinen järjestelmä olisi ollut kannattavampi kehittää, toteuttaa ja ylläpitää, mikäli sulautettu alusta olisi korvattu tavallisella pöytätietokoneella. Tällöin järjestelmä olisi saatu toteutettua luotettavammaksi ja nopeammaksi pienemmällä kehitystyöllä. Sulautetun järjestelmän etuna oli kuitenkin tämän pieni koko: Asiakkaan tiloihin ei ollut suotavaa asentaa täysikokoista pöytätietokonetta laboratorion tilarajoitusten takia.

Mikäli projekti toteutettaisiin uudestaan tämän projektin tietojen pohjalta, järjestelmä suunniteltaisiin käyttämään sulautetun tietokoneen sijaan tavallista pöytäkonetta.

Kaiken kaikkiaan projekti oli onnistunut ja tästä saatiin hyödyllistä kokemusta vastaavien projektien kehittämistä varten.

Lähteet

1 Modbus application protocol specification v1.1b3. Verkkojulkaisu. http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf Luettu 30.1.2017

2 Modbus over serial line specification and implementation guide v1.02. Verkkojulkaisu. http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf .Luettu 2.2.2017.

3 Multi-Drop Bus / Internal Communication Protocol. Verkkojulkaisu. https://namanow.org/images/pdfs/technology/mdb_version_4-2.pdf Luettu 2.2.2017.

4 BeagleBone Black. Verkkojulkaisu. https://beagleboard.org/black. Luettu 6.2.2017.

5 Web server performance comparison. Verkkojulkaisu. https://help.dreamhost.com/hc/en-us/articles/215945987-Web-server-performance-comparison Luettu 9.2.2017.

6 Micron Technology Inc EMMC Series Data Sheet. Verkkojulkaisu. http://media.digikey.com/pdf/Data%20Sheets/Micron%20Technology%20Inc%20PDFs/EMMC_Series.pdf Luettu 9.2.2017.

7 NGINX vs. Apache: Our View of a Dedace-Old Question. Verkkojulkaisu. https://www.nginx.com/blog/nginx-vs-apache-our-view/ Luettu 10.2.2017.

8 Struct – Interpret strings as packed binary data. Verkkojulkaisu. https://docs.python.org/2/library/struct.html Luettu 19.2.2017.

9 Dygraphs. Verkkojulkaisu. http://dygraphs.com/ Luettu 19.2.2017.