OULUN AMMATTIKORKEAKOULU

Jarno Rankinen

LÄMMITYS- JA IV-KONEEN YHDISTÄMINEN AVOIMEN LÄHDEKOODIN KOTI-AUTOMAATIOON

LÄMMITYS- JA IV-KONEEN YHDISTÄMINEN AVOIMEN LÄHDEKOODIN KOTI-AUTOMAATIOON

Jarno Rankinen
Opinnäytetyö
Kevät 2023
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Jarno Rankinen
Opinnäytetyön nimi: Lämmitys- ja IV-koneen yhdistäminen avoimen lähdekoodin kotiautomaatioon
Työn ohjaaja: Teemu Korpela
Työn valmistumislukukausi ja -vuosi: Kevät 2023
Sivumäärä: 42 + 2 liitettä

Opinnäytetyön tavoitteena oli yhdistää omakotitalon Enervent Pingvin Kotilämpö -lämmitys- ja ilmanvaihtojärjestelmä Home Assistantiin, avoimen lähdekoodin kotiautomaatiojärjestelmään. Työn ohessa tavoiteltiin myös asumismukavuuden parantamista, tutkimalla aktiivista puuttumista lämmitysjärjestelmän toimintaan. Näin saavutettavalla huonelämpötilavaihteluiden vähentämisellä haluttiin myös pyrkiä energiansäästöön.

Raspberry Pi -alustalle kirjoitettiin sovellus, joka luo kotiverkkoon HTTP-pohjaisen REST-rajapinnan. Tälle rajapinnalle Home Assistant -sovellukselta saapuvat pyynnöt välitetään Modbus-protokollalla RS-485-väylää pitkin Pingvin-laitteelle. Sovellus kirjoitettiin Go-ohjelmointikielellä. Home Assistantille luotiin kojelauta laitteen mittausten tarkastelemiseksi ja toimintojen käyttämiseksi.

Opinnäytetyön päätavoite, lämmityksen integrointi Home Assistantiin onnistui hyvin. Home Assistant-kojelaudasta saatiin Pingvin-laitteen omaan hallintapaneeliin verrattuna miellyttävä käyttää. Energiansäästötavoitteet todettiin vaikeaksi saavuttaa. Pingvin itsessään on jo energiatehokas pyöriväkennoisine lämmöntalteenottoineen. EC-ohjattu kiertoilmapuhallin osoittautui myös energiatehokkaammaksi kuin opinnäytetyötä aloittaessa arvioitiin, joten saavutettava energiansäästö jäi pieneksi. Opinnäytetyön kehitysjaksolla ei koettu aikaisempina talvina havainnoituja sisälämpötilan muutoksia. Pingvin-laitteen toimintaan puuttumisen automatisoimiseen ei siten koettu tarvetta. Mahdollinen kehitystyö tällä saralla jouduttiin jättämään myöhempään ajankohtaan.

Asiasanat: Go, Golang, kotiautomaatio, älykoti, IoT, Modbus

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

---

Author: Jarno Rankinen
Title of thesis: Integrating a Residential Heating and Ventilation System into Open-Source Home Automation
Supervisor: Teemu Korpela
Term and year when the thesis was submitted: Spring 2023
Number of pages: 42 + 2 appendix

The goal of this thesis was to integrate a residential heating / ventilation system into Home Assistant, an open-source smart home and home automation platform. This was implemented by connecting a Raspberry Pi with an RS-485 adapter into the heating / ventilation system. Software was written for listening on a HTTP REST API, translating between HTTP API calls and Modbus commands. The software was written in Golang, and templates for Home Assistant dashboards and other components were created. Secondary object was to improve quality of living by examining if the heating system's built-in automation can be improved by actively changing settings in the system. This proved to be difficult due to restrictions on the functions allowed over the RS-485 bus.

During the thesis process, achieving possible energy conservation was examined. This was deemed challenging, as the heating system was quite energy-efficient by default. Resulting Home Assistant integration improved the interface over the aging touch screen control panel of the ventilation unit, providing with possibilities to integrate heating or ventilation functions into other home automation tasks in the future.

Keywords: Go, Golang, home automation, smart home, IoT, Modbus

SISÄLLYS

SANASTO

ASCII	American Standard Code for Information Interchange, standardoitu merkistö tietotekniseen käyttöön
CSS	Cascading Style Sheets, Markup-kielisen tiedoston muotoiluun käytetty kieli
DOM	Document Object Model, dokumenttioliomalli. HTML-sivujen muokkaamiseen käytetty rajapinta
GPIO	General Purpose Input / Output, yleiskäyttöinen liitin mikrokontrollereissa
HAT	Hardware Attached on Top, Raspberry Pihin liitettävä laajennuspiiri
HTTP	HyperText Transfer Protocol, verkkoliikenneprotokolla
IoT	Internet of Things, esineiden Internet
JavaScript	Web-sovelluksissa suosittu kevyt tulkattava ohjelmointikieli
RS-485	Standardoitu sarjaliikenneväylä
Modbus	Sarjaliikenneprotokolla
MQTT	MQ Telemetry Transport, rajoitetuissa verkkoympäristöissä toimivaksi tarkoitettu viestintäprotokolla
REST	Representational State Transfer, joukko määritteitä rajapinnalle
RESTful	REST-periaatetta noudattava rajapinta
(Modbus) RTU	Remote Terminal Unit, binäärimuotoinen sarjatiedonsiirto Modbus-protokollalla
SBC	Yhden piirilevyn tietokone
YAML	YAML Ain't Markup Language, rakenteellisen tiedon esittämiseen käytetty merkintäkieli

1 JOHDANTO

1979 rakennetun asuintaloni lämmityksestä ja ilmanvaihdosta vastaa Enervent Pingvin Kotilämpö eAir -ilmalämmitysjärjestelmä (myöhempänä Pingvin). Laite on suunniteltu korvaamaan 1970- ja 80-luvuilla suosittu Valmet Kotilämpö -ilmalämmitysjärjestelmä (1). Talon lämmitysmuotona on kaukolämpö, ja lämmönjako tapahtuu alajakoisella ilmalämmityksellä. Asuintaloni lämmitysjärjestelmä uusittiin vuonna 2018 vaihtamalla Valmet Kotilämpö Pingviniin.

Pingvin eroaa alkuperäisestä Valmet Kotilämpö -järjestelmästä laitteeseen rakennetulla automaatiolla, koneellisella tuloilmalla sekä jatkuvalla koneellisella poistoilmalla. Mahdollisesti näiden eroavaisuuksien takia laitteen toiminta on jättänyt alkuperäiseen verrattuna toivomisen varaa asumismukavuuden osalta. Talon alkuperäinen ilmanvaihto toimi pääasiassa painovoimaisesti. Koneellinen poistoilma oli kytkettävä käsin päälle esimerkiksi suihkun ajaksi. Automatiikan suunnittelussa on myös selkeästi pyritty energiansäästöön, joka näkyy asukkaille usein hitaana reagointina lämpötilojen muutoksiin. Hidas reagointi puolestaan johtaa Suomen vaihtelevissa olosuhteissa usein suuriin sisäilman lämpötilan vaihteluihin ja edelleen hitaana normalisoitumisena haluttuun lämpötilaan. Pingvinin kosketusnäytöllinen ohjauspaneeli on myös nykymittapuulla kankea käytettävä. Esimerkiksi mittaustietojen tarkastelu ongelmatilanteissa on osoittautunut hankalaksi.

Ominaisuuksien tarkemman tutkiskelun jälkeen paljastui, että Pingvinistä löytyy mahdollisuus taloautomaatioon liittämiseen RS-485-väylää ja Modbus-protokollaa käyttäen (2). Energia-asioiden ollessa erityisen ajankohtaisia olin hiljattain alkanut ottaa käyttöön Home Assistant -kotiautomaatioalustaa, tavoitteena säästää sähköä muun muassa etäohjattavia pistorasioita hyödyntäen. Home Assistant mahdollistaa lukemattomien erilaisten kodin laitteiden integroinnin yhteen hallintajärjestelmään esimerkiksi suoraan Modbus-protokollaa käyttäen (3). Tässä opinnäytetyössä kyseistä integraatiota ei kuitenkaan käytetä.

Huomattavaa tämän opinnäytetyön toteutuksessa on, että laitteen toimintahistoria ja siihen tavoitteena olleet muutokset perustuvat täysin omaan ja perheeni aistinvaraiseen havainnointiin. Osa laitteen toiminnoista ei saa aikaan valmistajan tarkoittamaa vaikutusta, vaan usein jopa päinvastaisen. Tämä opinnäytetyö ei käsittele asiaa talotekniseltä kannalta, vaan toteutuksessa pyrittiin parantamaan asumismukavuutta mahdollisimman pienillä muutoksilla lämmityksen tekniseen toimintaan.

Tämän opinnäytetyön tavoitteena oli yhdistää omakotitalon Enervent Pingvin Kotilämpö eAir -ilmalämmitysjärjestelmä avoimen lähdekoodin Home Assistant -kotiautomaatiosovellukseen. Tarkoituksena oli saavuttaa laitteen mittaustietojen lukeminen sekä asetusarvojen muuttaminen Home Assistantista käsin.

Opinnäytetyön toinen tavoite oli tutkia, pystytäänkö aktiivisella puuttumisella laitteen toimintaan mahdolliseen energiansäästöön. Vaikka kyseisen omakotitalon lämmitysmuotona on kaukolämpö, muodostuu ylivoimaisesti suurin osa talouden energiakustannuksista lämmitysenergiasta, etenkin talvisaikaan. Lisäksi vuoden 2022 sähköenergian hinnannousun takia laitteen kiertoilmapuhaltimen pyörimisnopeutta pienentämällä sääolosuhteiden salliessa arvioitiin ennen työn aloittamista päästävän mahdollisesti merkittäviin kustannussäästöihin.

Kolmas tavoite oli vähentää huonelämpötilan vaihteluita luomalla automatiikkaa Pingvinin asetusten muuttamiselle. Tämän arvioitiin jo alkuvaiheessa olevan mahdollisesti haastavaa, koska joudutaan ohittamaan tai yliajamaan Pingvinin omaa automatiikkaa. Tämän automatiikan toimintaan liittyy myös Pingvinin mittaaman datan tallentaminen johonkin aikasarjatietokantaan myöhemmän tarkastelun helpottamiseksi.

Pingvinille kommunikointiin käytettiin Raspberry Pi -laitetta (myöhempänä RPi) ja GPIO-väylään kytkettävää RS-485 HAT -adapteria. Opinnäytetyön pääasiallinen työn kohde oli sovellus, joka toimii siltana Pingvinin Modbus-väylän ja lähiverkon välillä. Sovellus luo HTTP-pohjaisen RESTful-rajapinnan, jota käyttäen Home Assistant voi lukea Pingvinin tietoja ja muuttaa asetuksia. Rajapintaa ei ole tarkoitus käyttää paikallisverkon ulkopuolelta käsin suoraan.

Käytetyt ohjelmointi-, merkintä- ja muut kielet ovat Go, Python, JavaScript, HTML, CSS sekä YAML. Alustava kokeiluversio Pingvin-laitteelle kommunikoinnista toteutettiin Python-ohjelmointikielellä. Kun kartoitus mahdollisuuksista Pythonilla oli tehty, lopullinen versio toteutettiin Go-ohjelmointikielellä. Kielivalintaa perustellaan luvussa 2. Lisätavoitteena oli perehtyä Go-ohjelmointikieleen paremmin projektin edetessä. Modbus-väylän raakadatan tarkkailuun toteutettiin yksinkertainen selainpohjainen käyttöliittymä, jonka luettavuutta parannettiin pienillä tyylimuutoksilla CSS:ää käyttäen. Varsinaisena käyttäjärajapintana toimivat Home Assistant -sovellukseen luotavat koje-

laudat ja kortit. RESTful-rajapinnan sekä graafisen käyttöliittymän määrittely Home Assistantiin tapahtui YAML-merkintäkielellä. JavaScriptiä käytettiin päivittämään Modbus-raakadatan tarkastelusivun sisältö reaaliaikaisesti.

Opinnäytetyön teknisen toteutuksen suunnittelussa ja järjestelyssä hyödynnettiin GitHub Projects -alustaa. Lähdekoodin versionhallintaan käytettiin Git-sovellusta.

2 MENETELMÄT JA LAITTEET

Ennen projektin aloitusta selvitettiin tarkoitukseen parhaiten sopivat laitteet, protokollat sekä ohjelmointikielet. Pingvin-laitteelle ulkoiseen kommunikointiin ainoa mahdollisuus on Modbus-protokolla. Vaikka Pingvin tarjoaa myös selainpohjaisen verkkokäyttöliittymän, ei tätä projektia haluttu sitoa Internet-yhteyden toimivuuteen. Home Assistant oli kohteessa jo käytössä, joten käyttöliittymän luominen sen yhteyteen oli luonnollinen valinta.

Projektin muodostama kokonaisuus pitäisi toimia hyvin myös muulla kuin projektin toteutuksessa käytetyllä laitteistolla. Sovellukselle alustana toimii mikä tahansa prosessoriarkkitehtuuri, jolle Go-kieltä voidaan kääntää. Modbus-kommunikointiin on useita erilaisia USB-pohjaisia adapterivaihtoehtoja. Linux-konttien suorittaminen on nykyisin mahdollista myös Windows-käyttöjärjestelmällä (4). Home Assistant ei täten myöskään vaadi täyttä Linux-käyttöjärjestelmän asennusta.

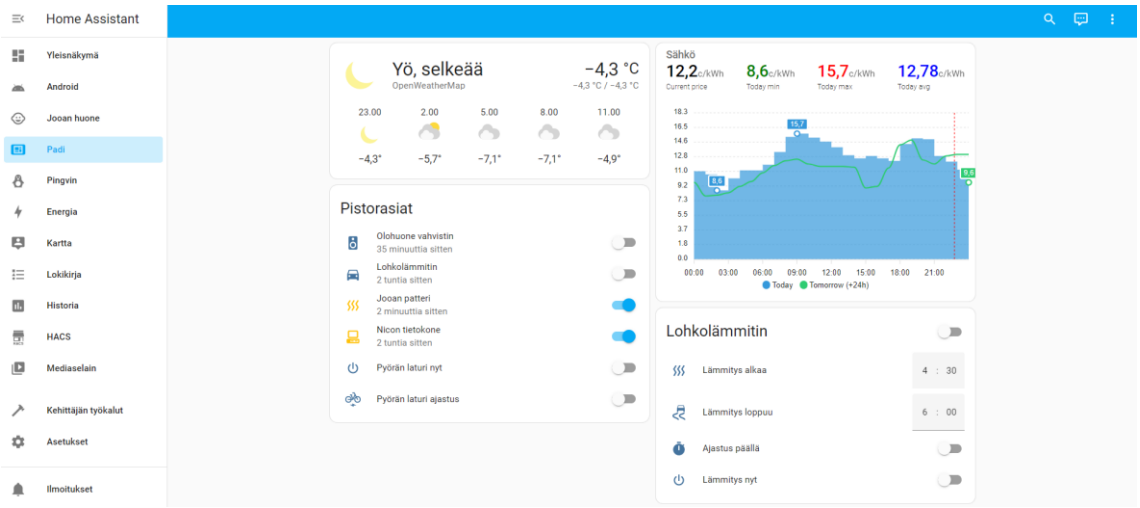
2.1 Home Assistant -alusta

Home Assistant on avoimen lähdekoodin kotiautomaatioon keskittynyt alustasovellus. Sen ensimmäinen julkisesti saatavilla oleva lähdekoodirevisio on kirjattu projektin GitHub-arkistoon vuonna 2013. Silloisessa readme-tiedostossa sanotaan, että projektin kehitys on kuitenkin aloitettu jo aiemmin. Home Assistant on kirjoitettu Python-ohjelmointikielellä. Se tarjoaa käyttäjälleen monipuolisen alustan älykodin toimintojen luomiseksi. (5.)

Home Assistant on suunniteltu asennettavaksi esimerkiksi yhden piirilevyn tietokoneelle (SBC) tai tavallisella Linux-pohjaisella kotitietokoneelle (6). Tämän opinnäytetyön toteutuksessa sovellusta ajettiin Linux-kontissa kotipalvelimena toimivalla Dell Optiplex Ultra Small Form Factor -tietokoneella Rocky Linux 8.7 -käyttöjärjestelmän päällä.

Home Assistantia käytetään selainpohjaisen käyttöliittymän kautta tai Progressive Web App (PWA) -mobiilisovelluksilla. Käyttöliittymä muodostuu erilaisista kojelaudoista. Yksittäinen kojelauta rakentuu erilaisista korteista, jotka tarjoavat joko toimintoja, mittaustietoja, kuvia tai muita tietoja integraatioista (kuva 1). Integraatiot voivat olla fyysisiä Internet of Things (IoT) -laitteita, kuten älypistorasioita ja lämpömittareita, tai ohjelmistoja, jotka tarjoavat ohjelmistorajapinnan hallinnalleen tai

datan keräämiselle. Python-ohjelmointikieli lukuisine kirjastoineen on mahdollistanut käytettävissä olevien kommunikaatioprotokollien laajan kirjon. Yleisimpiin protokolliin kuuluvat Bluetooth, Zigbee, Z-Wave, HTTP ja MQTT (7).



KUVA 1. Home Assistant-kojelauta

Integraatioiden, automaatioiden ja muiden kohteiden lisääminen ja määrittely Home Assistantiin tapahtuu joko graafiselta käyttöliittymältä tai YAML-tiedostoja luomalla sekä muokkaamalla. YAML-asetustiedostoissa voidaan hyödyntää myös Jinja-mallinnuskieltä. Jinja-mallinnuskieltä käyttäen voidaan luoda esimerkiksi lämpötila-asetusta ja huonelämpötilaa mittaavista sensoreista kolmas binäärisensori, joka osoittaa totuustilaa sille, sallitaanko aktiivinen lämmitys (kuva 2).

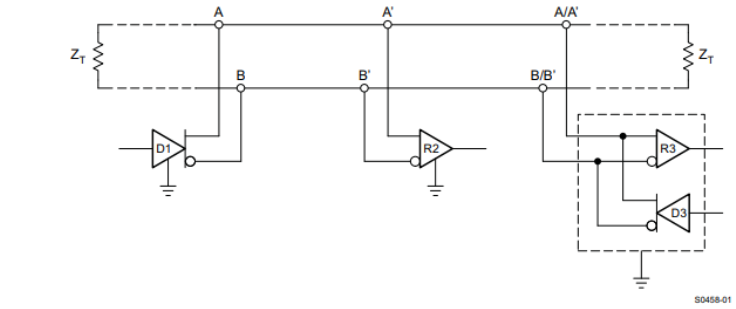
```
template:
  - binary_sensor:
    - name: "Penguin max heating allowed"
      state: >
        {{ states('input_number.penguin_temperature_setting_helper') >
        states('sensor.penguin_room_temperature_1') }}"
```

KUVA 2. Ote Home Assistantin YAML-asetustiedostosta Jinja-mallinnuskieltä hyödyntäen

2.2 RS-485 ja Modbus

RS-485 on balansoitu väyläjärjestelmä tiedon siirtämiseen pitkiä matkoja häiriöisessä ympäristössä. Väylää voi käyttää vain yksi laite kerrallaan (half-duplex), ja päätelaitteiden välillä tieto siirtyy

fyysisesti kahta johdinta pitkin (kuva 3). RS-485:n käyttöjännitteet ovat luokkaa −7–12 V, joten kytkentöjen suorittaminen katsottiin turvalliseksi. (8, s. 9.)



A D1 = driver
B D3/R3 = transceiver
C R2 = receiver
D Z_T = termination impedance
E Up to 32 U.L.s [receiver, driver (off state), transceiver]

Figure 9. RS-485 Balanced-Voltage Digital-Interface Circuit

KUVA 3. RS-485 havainnekuva (8, s. 9)

Modbus on yleisesti teollisuudessa ja elektronisten laitteiden välisessä kommunikaatiossa käytetty sarjaliikenneprotokolla. Protokolla perustuu asiakas–palvelin-toimintamalliin. Yleisimmin asiakasosapuoli (master) lähettää komennon, jonka perusteella palvelin (slave) suorittaa komennon ja vastaa takaisin. Protokolla on julkaistu alun perin vuonna 1979. (9.)

Samalla väylällä voi olla useita palvelinlaitteita, kuten RS-485-järjestelmä sallii (8, s. 10). Modbus-protokollan mukainen kehys sisältää tarkoitetun vastaanottajan osoitteen, jolla kehykset voidaan osoittaa tietylle palvelimelle (9). Tässä opinnäytetyössä väylään oli kytkettynä vain kaksi laitetta, asiakas ja palvelin. Palvelimena toimi Pingvin ja asiakkaana RPi Zero.

Modbusin tietorakenne jakautuu neljään tietotyyppiin. Discrete Input, hakutieto, on yhden bitin vain luettavissa oleva arvo. Coil, ohjelmallinen piste, on yhden bitin luettava sekä kirjoitettava arvo. Input Register, reaaliaikainen rekisteri, on 16 bitin vain luku -arvo. Holding Register, pitorekisteri, on luettavissa sekä kirjoitettavissa oleva 16 bitin arvo. Jokainen tietotyyppi sisältää useita osoitteellisia yksittäisiä arvoja. Osoite on 16-bittinen, eli jokaista tietotyyppiä voi olla 65 536 arvoa (10). Pingvin-laitteen Modbus-sovelluksessa käytetään ainoastaan ohjelmallisia pisteitä sekä pitorekistereitä. Tiedot ja mittaukset tallennetaan 72 ohjelmalliseen pisteeseen sekä 800 pitorekisteriin (11). Näistä suurin osa on kuitenkin joko käyttämättömiä, tai valmistajan omaan lisenssinalaiseen käyttöön varattuja.

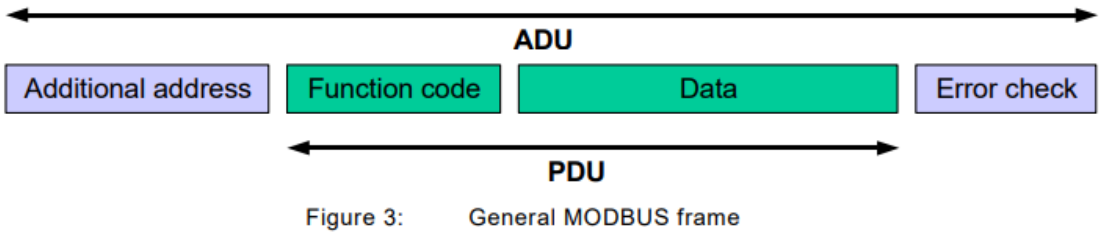


Figure 3: General MODBUS frame

KUVA 4. Modbus-tietokehys (9.)

Modbus-kehykset voidaan välittää joko sarjaväyliä RS-232 tai RS-485 pitkin tai Ethernet-väylää ja TCP/IP-protokollaa käyttäen (12). Modbus-kehykset voivat käyttää kahta eri lähetystilaa, ASCII- sekä RTU-tilaa. ASCII-tilassa jokaista lähetettyä 8-bittistä tavua vastaa kaksi 7-bittistä ASCII-merkkiä. RTU-tilassa jokaista 8 bitin tavua vastaa kaksi 4 bitin heksadesimaalimerkkiä (13).

Tässä opinnäytetyössä käytettiin valmista GoBurrow/Modbus -pakettia hoitamaan väylän yksityiskohtien käsittely. Moduuli hoitaa itse väylälle kirjoitettavien kehysten muodostamisen ja purkamisen, mutta kehyksen sisältämän datan tulkintaan se ei osallistu (14). Pingvin käyttää Modbus RTU -tilaa (2).

2.3 Raspberry Pi Zero

Tämän opinnäytetyön laitteistoalustaksi valittiin ensimmäisen sukupolven yhden piirilevyn tietokone Raspberry Pi Zero W. Laitevalintaa ohjasi laitteiston saatavuus sekä energiankulutus. Laitteen on tarkoitus olla päällä jatkuvasti, joten pieni energiajalanjälki oli oleellista opinnäytetyön tavoitteiden saavuttamiseksi. Sovelluksen ei haluttu ei vaativan laitteistolta suurta laskentatehoa, jotta tarvittava laitteisto saataisiin pidettyä mahdollisimman yksinkertaisena.

RPi Zero W tarvitsee virtalähteeltään tyypillisesti 0,18 W:n tehon (15). Tästä syystä se soveltui projektin laitteistoalustaksi hyvin. Raspberry Pi -laitteiden saatavuus oli opinnäytetyön aloitushetkellä syksyllä 2022 erittäin huono (16). Zero W -mallia oli saatavilla ja se oli edullinen.

RS-485-väylä vaatii erityisiä kytkentöjä, kuten terminointivastukset, joten Raspberry Pi -laite ei ole kytkettävissä väylään sellaisenaan (8). Tähän tarkoitukseen hankittiin Zihatec GmbH:n valmistama, RPi:n GPIO-väylään kytkettävä RS-485 HAT (Hardware Attached On Top) -adapteri. Valtaosa RPi

Zero W:n käyttöjärjestelmävaihtoehdoista ovat Linux-pohjaisia (17). RPi Zero W:ssä on yksiytiminen 32-bittinen prosessori, joka toimii 1 GHz:n kellotaajuudella. Lähiverkkoon RPi Zero W kytkeytyy WLAN-yhteydellä, Ethernet-liitintä laitteessa ei ole. RAM-muistia RPi Zero W:ssä on 500 MiB (18).

Koska projektin kokonaisuus on suunniteltu kotiympäristöön, HTTP-rajapinnalle tulevien pyyntöjen määrä ennakoitiin vähäiseksi. RPi Zero W:n suorituskyvyn arvioitiin olevan riittävää.

2.4 Go-ohjelmointikieli

Go on vuonna 2012 julkaistu ohjelmointikieli, jonka perusperiaatteisiin kuuluu staattinen tyypitys, ajonaikainen tehokkuus, luettavuus, helppokäyttöisyys sekä korkea verkko- ja moniajosuorituskyky. Go-kielisen ohjelman suorittaminen vaati kääntämisen. Go-ohjelmointikielessä ei myöskään ole luokkajärjestelmää. (19, s. 15–16.)

Käännettynä ohjelmointikielenä Go-sovellukset ovat usein yksittäisiä, staattisia, ajettavia binääritiedostoja (20, luku 3.6.2). Yksittäiset tiedostot ovat yksinkertaisia toimittaa loppukäyttäjille. Tämä oli yksi ohjelmointikielen valintakriteereistä. Staattinen binääritiedosto myös poistaa ongelmat esimerkiksi Python-tulkin versioiden osalta, jos sovellusta ajetaan vaikkapa vanhemmalla Raspberry Pi OS:n tai Raspbianin versiolla. Koska sovellus pääasiallisesti vain siltaa pyyntöjä kahden eri verkkoprotokollan, Modbusin ja HTTP:n, välillä, arvioitiin Go-ohjelmointikielen lähtökohtainen verkkosuorituskyky edulliseksi tavoitteiden saavuttamiseksi. Sisäänrakennettu moniajokyky mahdollistaa esimerkiksi HTTP-pyyntöjen käsittelyn toisistaan riippumattomina tapahtumina.

3 TOTEUTUS

Tarvittava laitteisto hankittiin verkkokaupoista. Kun laitteisto oli vastaanotettu, suoritettiin kytkennät. RS-485 HAT on suunniteltu kiinnittymään Raspberry Pin GPIO-liitimeen, eikä muita kiinnityksiä näiden välillä tarvita (21, s. 1). Alkukartoitukseen käytettiin RPi Zero W:tä huomattavasti tehokkaampaa 4B-mallia. RS-485-HAT:n asetuskytkimet asetettiin Modbus-liikenteelle sopiviin asetuksiin. Käyttöjärjestelmäksi 4B:lle valittu Raspberry Pi OS 64-bit kopioitiin SD-kortille. GPIO-sarjaportti asetettiin päälle Raspberry Pi OS:n *raspi-config*-ohjelmalla.

Myöhemmässä vaiheessa projektia vaihdettiin Raspberry Pi Zero W:hen, ja kokonaisuus koteloitiin. Koteloksi muokattiin 4B-mallille tarkoitettu muovinen kotelo (kuva 5).

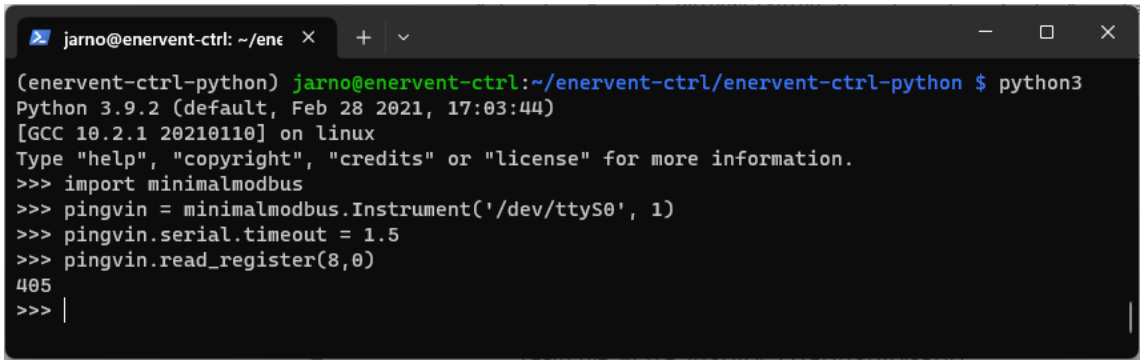
KUVA 5. Raspberry Pi Zero W ja RS-485 HAT niille muokatussa kotelossa

3.1 Modbus-protokollan toimintojen kartoitus

Enervent Pingvin -laitteessa RS-485-liitin sijaitsee emolevyllä (22, s. 17). Emolevy ja muu elektroniikka sijaitsevat kotelossa, eristettynä ilmanvaihdosta. Laite sammutettiin ja yhteys verkkovirtaan katkaistiin. Modbus-liittimenä laitteessa oli käytetty samanlaista liitintä kuin Zihatec GmbH:n RS-

485 HAT:ssä. Liitinmallissa on irrotettava johtoon kiinnitettävä osa. Johtimiin kiinnitettävä osa irrotettiin emolevystä, ja johtimet kiinnitettiin ruuvikiinnityksillä liittimeen. RS-485-väylän johtimia ei kytketä ristiin, vaan päätelaitteiden A-liittimet kytketään toisiinsa ja B-liittimet toisiinsa (8, s. 9.) (kuva 3).

Seuraavaksi kartoitettiin, voidaanko haluttuja toimintoja suorittaa käyttämällä RS-485-väylää. Ensimmäiset kokeilut suoritettiin Python-ohjelmointikielellä. Sekä ohjelmallisten pisteiden että pitorekisteritietueiden lukeminen onnistui suoraan Python-ohjelmointikielen interaktiivista komentoriviä ja PyModbus-kirjastoa käyttäen (kuva 6). Tässä vaiheessa laitteistoalustana toimi 4B-malli. Python-ohjelmointikielellä kirjoitettiin myös alustavaa versiota RESTful-rajapinnasta. Projektin suunnittelussa oli päätetty, että lopullinen toteutus tapahtuu Go-ohjelmointikielellä, joten tämä esiversio jätettiin keskeneräiseksi.



KUVA 6. Tuloilman lämpötilaa kuvaavan pitorekisterin lukeminen Python-komentoriviltä

Python-ohjelmointikielellä tehdyssä esikokeilussa havaittiin, että jotkut halutuista toiminnoista eivät ole mahdollisia RS-485-väylän kautta. Vaikka kaikki tietueet ovat Modbus-protokollan määritelmän mukaan kirjoitettavissa, suurin osa pitorekistereistä ei ottanut niille syötettyä arvoa vastaan. Modbus-protokolla määrittää, että palvelin vastaa vastaanottamaansa kyselyyn. Poikkeustilanteissa vastausta ei joko lähetetä ollenkaan tai se sisältää virhekoodin (9, s. 47). Kaikkiin arvojen muuttamisiin saatiin vastaus onnistuneesta käsittelystä. Pitorekisterien arvot kuitenkin pysyivät joko samana tai palautuivat nopeasti edelliseen lukemaansa. Tästä voitiin päätellä, että Pingvinin valmistaja ei ole mahdollistanut arvojen muuttamista ulkoisesta taloautomaatiosta. Rajoitettuihin toimintoihin kuuluivat mm. kiertoilmapuhaltimen nopeuden suora muuttaminen sekä tuloilman lämpötilan raja-arvojen vaihtaminen, jotka olisivat mahdollistaneet lämmityksen täyden ulkoisen hallinnan.

3.2 Tietorakenteet

Python-esitoteutuksessa käytettiin luokkia edustamaan Pingvinin tietorakenteita (kuva 7). Luokat sisälsivät varsinaisen tiedon lisäksi jäsenfunktioita tiedon käsittelyyn. Go-ohjelmointikielessä ei ole luokkien käsitettä (19). Niiden sijasta tietorakenne koostui tietueista (struct), jotka ovat yksi Gon tyyppijärjestelmän keskeisistä käsitteistä (23) (kuva 8).

```python
class PingvinCoil():
    """Single coil data structure"""
    def __init__(self, symbol="-", description="-"):
        self.symbol = symbol
        self.value = False
        self.description = description
        self.reserved = symbol == "-" and description == "-"

    def serialize(self):
        return {
                    "value": self.value,
                    "symbol": self.symbol,
                    "description": self.description,
                    "reserved": self.reserved
                }

    def get(self):
        return jsonify(self.serialize())

    def flip(self):
        self.value = not self.value
```

KUVA 7. Ohjelmallista pistettä edustava Python-luokka

```go
// single coil data
type pingvinCoil struct {
    Address     int              `json:"address"`
    Symbol      string           `json:"symbol"`
    Value       bool             `json:"value"`
    Description string           `json:"description"`
    Reserved    bool             `json:"reserved"`
    PromDesc    *prometheus.Desc `json:"-"`
}
```

KUVA 8. Ohjelmallista pistettä edustava Go-tietue

Go-ohjelmointikielen tietueen yksi huomattavimmista eroista luokkaan on, että tietueiden oletusarvoja ei voi määritellä. Go-kääntäjä alustaa muuttujat niiden nolla-, eli oletusarvoilla. Kaikilla Go-ohjelmointikielen tietotyypeillä on nolla-arvo (24). Tietueen esiintymän luominen tapahtuu vakiintu-

neen käytännön mukaan tietueeseen luotavalla New-funktiolla, jonka lisääminen luokkaan tapahtuu erillään itse tietueen määrittelystä, samassa tiedostossa. Funktio voi ottaa parametreja, jotka syötetään arvoiksi luotavalle tietue-esiintymälle. Tietue-esiintymä voidaan myös luoda käyttämällä *composite literal* -menetelmää, jossa tietuetyypin nimen jälkeen annetaan mahdolliset arvot aaltosulkujen sisässä (kuva 9) (24). Kuvassa 9 rivillä 618 luodaan Pingvin-tietueen esiintymä oletusarvoilla *composite literal* -menetelmällä. Riveillä 626 ja 635 käytetään newCoil() ja newRegister() -funktioita luomaan pingvinCoil- ja pingvinRegister-tietue-esiintymät parametreineen.

```
616  // create a Pingvin struct, read coils and registers from CSVs
617  func New(serial string, debug bool) *Pingvin {
618      pingvin := Pingvin{}
619      pingvin.Debug.dbg = debug
620      pingvin.buslock = &sync.Mutex{}
621      pingvin.createModbusClient(serial)
622      log.Println("Parsing coil data...")
623      coilData := readCsvLines("coils.csv")
624      for i := 0; i < len(coilData); i++ {
625          pingvin.Coils = append(pingvin.Coils,
626              newCoil(coilData[i][0],
627                  coilData[i][1],
628                  coilData[i][2]))
629      }
630      log.Println("Parsed", len(pingvin.Coils), "coils")
631      log.Println("Parsing register data...")
632      registerData := readCsvLines("registers.csv")
633      for i := 0; i < len(registerData); i++ {
634          pingvin.Registers = append(pingvin.Registers,
635              newRegister(registerData[i][0],
636                  registerData[i][1],
637                  registerData[i][2],
638                  registerData[i][3],
639                  registerData[i][6]))
640      }
641      log.Println("Parsed", len(pingvin.Registers), "registers")
642      return &pingvin
643  }
644
```

KUVA 9. New-funktion määrittely sekä composite literal -menetelmä.

Tietueiden yksittäiset jäsenmuuttujat voivat olla myös muita tietueita (24). Liitteessä 1 on esitetty Pingvin-laitetta mallintava tietorakenne. Koska ohjelmallisia pisteitä on 72 ja pitorekistereitä 800, todettiin jokaisen määrittely käsin työlääksi toteuttaa. Pingvinin valmistaja Enervent on asettanut laitteen pitorekisterilistauksen julkisesti saataville (11). Valmistajan Excel-taulukosta muokattiin ohjelmallisille pisteille ja pitorekistereille CSV-tiedostot, jotka luetaan ohjelman käynnistyessä. Näiden tiedostojen pohjalta muodostetaan Pingvin-tietueen dynaamiset Coils- ja Registers-taulukot (slice),

jotka puolestaan koostuvat pingvinCoil- ja pingvinRegister-tietueista. Yksittäisiä ohjelmallisia pisteitä ja pitorekistereitä edustavat tietueet alustetaan myös CSV-tiedostojen perusteella.

3.3 Tietojen lukeminen

RS-485-väylällä kommunikointia varten Pingvin-tietueeseen luotiin jäsen säilyttämään GoBurrow/modbus-kirjaston Handler- ja Client-tietue-esiintymät ohjelman ajon ajaksi. Projektin alkuvaiheessa jokaista kommunikaatiokertaa varten luotiin uudet Handler- ja Client-tietueet, mutta tämä aiheutti paljon koodin toistoa ja vaikeutti luettavuutta. Myöhemmissä kehitysversioissa siirryttiin käyttämään staattisia uudelleenkäytettäviä esiintymiä. Handler-tietue määrittää Modbus-väylän parametrit, kuten baudinopeuden, pariteetin ja palvelimen Modbus-osoitteen. Client-tietue puolestaan sisältää tietojen lukemiseen ja kirjoittamiseen käytetyt rajapintafunktiot. Tietueet luovaa funktiota kutsutaan New-funktiosta, jonka jälkeen esiintymiä säilytetään muistissa ohjelma ajon ajan. CreateModbusClient-funktio on myös Pingvin-tietueen jäsen (kuva 10). Go-ohjelmointikielessä funktioita voi liittää tietuetyypin jäseniksi käyttämällä merkintää func (t *Type) functionname {}. Tietuetyypin nimen edessä olevalla tähdellä ilmaistaan, että t on viittaus tiettyyn Type-tyyppisen esiintymän muistiosoitteeseen. Ilman tähtimerkintää funktio operoi esiintymän kopioon.

```
184  // Create modbus.Handler, store it in p.handler,
185  // connect the handler and create p.modbusclient (modbus.Client)
186  func (p *Pingvin) createModbusClient(serial string) {
187      // TODO: read configuration from file, mostly hardcoded for now
188      log.Println("Connecting to serial console on", serial)
189      p.handler = modbus.NewRTUClientHandler(serial)
190      p.handler.BaudRate = 19200
191      p.handler.DataBits = 8
192      p.handler.Parity = "N"
193      p.handler.StopBits = 1
194      p.handler.SlaveId = 1
195      p.handler.Timeout = 1500 * time.Millisecond
196      err := p.handler.Connect()
197      if err != nil {
198          log.Fatal("createModbusClient: p.handler.Connect: ", err)
199      }
200      p.Debug.Println("Handler connected")
201      p.modbusclient = modbus.NewClient(p.handler)
202  }
```

KUVA 10. Modbus-väylän konfigurointifunktio

Tietojen lukeminen väylältä ohjelman sisäisesti tapahtuu modbus.Client().ReadCoils()-funktiolla. Modbus-protokollan mukaan ohjelmallisia pisteitä voi lukea kerralla 1–2 000 kappaletta (9, s. 11).

Koska Pingvin-laitteessa on vain 72 ohjelmallista pistettä käytössä, näiden lukeminen suoritetaan yhdellä pyynnöllä (kuva 11). ReadCoils-funktio palauttaa datan taulukkona tavuja (byte) (14). Jokainen tavutaulukon bitti edustaa yhtä ohjelmallista pistettä. Taulukko muunnetaan sisäiseen käsittelyyn totuusarvoiksi (bool) vertaamalla silmukassa jokaista bittiä arvoon 1. Tarvittavan muistin määrän kasvaminen totuusarvojen käyttämisen takia katsottiin merkityksettömäksi.

```
211  // Update all coil values
212  func (p *Pingvin) updateCoils() {
213      p.buslock.Lock()
214      results, err := p.modbusclient.ReadCoils(0, uint16(len(p.Coils)))
215      p.buslock.Unlock()
216      if err != nil {
217          log.Fatal("updateCoils: client.ReadCoils: ", err)
218      }
219      // modbus.ReadCoils returns a byte array, with the first
220      // byte's bits representing coil values 0-7,
221      // second byte coils 8-15 etc.
222      // Within each byte, LSB represents the lowest n coil,
223      // while MSB is the highest
224      // e.g. reading the first 8 coils might return a byte array of length 1,
225      // with the following:
226      // [4], which is 00000100, meaning all other coils
227      // are 0 except coil #2 (3rd coil)
228      //
229      k := 0                              // pingvinCoil index
230      for i := 0; i < len(results); i++ { // loop through the byte array
231          for j := 0; j < 8; j++ {
232              // Here we loop through each bit in the byte, shifting right
233              // and checking if the LSB after the shift is 1 with a bitwise AND
234              // A coil value of 1 means on/true/yes, so == 1 returns the bool value
235              // for each coil
236              p.Coils[k].Value = (results[i] >> j & 0x1) == 1
237              k++
238          }
239      }
240  }
```

KUVA 11. Kaikkien ohjelmallisten pisteiden lukeminen ja käsittely

Pitorekisterien lukeminen vaati hieman monimutkaisemman toteutuksen. Jokainen pitorekisteri sisältää 16-bittisen arvon. Modbus RTU-protokollan perusyksikön ollessa 1 tavu, jokaista yksittäistä pitorekisteriä edustaa vastauksena saatavasta tavutaulukosta kaksi peräkkäistä tavua (9, s. 5). Näistä tavuista ensimmäinen edustaa 16-bittisen arvon 8:aa eniten merkitsevää bittiä, jälkimmäisen edustaessa 8:aa vähiten merkitsevää bittiä.

Pingvinin Modbus-sovelluksessa arvo voi myös olla etumerkitön (uint) tai etumerkillinen (int) lukuarvo, bittikenttä-tyyppinen tai esimerkiksi tilaa edustava luettelointiarvo (enumeration) (11). Osalla arvoista on myös kerroin. Lähdekoodin tulkittavuuden parantamiseksi päätettiin arvot tallentaa muistiin suoraan lähes lopullisessa käyttäjän tulkittavassa muodossa. Kertoimien huomioiminen

jätettiin asiakasohjelmille, jotta arvoja voitiin käsitellä kokonaislukuina ohjelman sisäisesti. Yksittäisen arvon tyyppi, kerroin ja kuvaus luetaan CSV-tiedostosta pingvinRegister-tietueen jäseniin ohjelman käynnistyessä.

Modbus-protokolla rajoittaa yhdellä käskyllä luettavissa olevien pitorekisterien määrän 125:een (9, s. 16). Koska Pingvin-laitteessa pitorekistereitä on 800 kappaletta, täytyi pitorekistereiden lukeminen suorittaa 6 erillisellä komennolla. Vaihtoehtona tutkittiin myös vain varaamattomien pitorekistereiden lukemista. Varaamattomat rekisterit ovat kuitenkin pilkkoutuneina useaan erilliseen yhtäjaksoiseen osioon. Näitä osioita todettiin olevan enemmän kuin kaikkien rekisterien lukemiseen vaadittiin. Vain varattujen rekisterien lukemiseen tarvittiin siten useampi komento Modbus-väylälle kuin kaikki rekisterit luettaessa. Varattujen rekisterien lukematta jättäminen todettiin kokeiluissa hitaammaksi.

Reaaliaikaista raakatietojen tarkkailua varten luotiin REST-polut palauttamaan Pingvin-tietueen Coils- ja Registers-taulukot JSON-muotoisena. Modbus-väylältä saatujen tietojen helppo tarkkailu reaaliajassa koettiin tärkeäksi, jotta sovelluksen toiminta voidaan varmistaa kehitysprosessin aikana. Varsinaisessa lopputoteutuksessa Home Assistant hakee tiedot siihen määritellyltä REST-palvelulta säädettävin väliajoin. Väliaikaa ei toteutushetkellä saanut säädettyä alle 15 sekuntiin, joka koettiin liian hitaaksi päivitysnopeudeksi kehitys- ja ongelmanratkaisuprosessiin. Tietueiden kääntäminen JSON-muotoiseksi merkkijonoksi toteutettiin Go-ohjelmointikielen standardikirjaston encoding/json -paketilla, käyttäen tietueissa `json:"key"`-merkintää. Merkinnällä voi muuttaa tietueen jäsenien nimiä, kun ne käännetään JSON-sanakirjaksi (25). JSON on web-pohjaisissa sovelluksissa rakenteellisen tiedon siirtoon laajalti käytetty tekstipohjainen muotoilu. Sen etuihin kuuluvat mahdollisuus monimutkaisiin tietorakenteisiin sekä hyvä käsiteltävyys ohjelmallisesti (26). JSON-muotoista tietoa on myös mahdollista lukea sellaisenaan.

HTTP-palvelin on myös toteutettu Go-ohjelmointikielen standardikirjastolla, käyttäen net/http-pakettia. Ohjelman suunniteltu käyttöympäristö on kotiverkko. Perusajatuksena on, että REST-rajapintaa käyttää ainoastaan Home Assistant, joten pyyntöjen määrän oletetaan säilyvän vähäisenä. Ohjelma oli alusta alkaen tarkoitus asettaa myös muiden käyttäjien saataville, joten riippuvuuksia ulkoisiin kirjastoihin haluttiin mahdollisimman vähän. Go-ohjelmointikielen net/http-kirjasto tarjoaa kattavat mahdollisuudet pyyntöjen käsittelyyn.

Pitorekisterien ja ohjelmallisten pisteiden arvot haetaan REST-rajapinnalle muistista. Tietojen päivittäminen laitteelta tapahtuu erillisessä aliohjelmassaan. Tähän ratkaisuun päädyttiin, koska tietojen lukeminen Modbus-väylältä hidastaisi HTTP-pyynnön käsittelyä. Useat samanaikaiset REST-pyynnöt voisivat lisäksi aiheuttaa ohjelman jumiutumista, koska RS-485 väylällä voidaan käsitellä vain yksi pyyntö kerrallaan. Hidas pyyntöjen käsittely mahdollistaisi myös helpot palvelunestohyökkäykset, tosin kotiverkoissa tämän ei katsottu olevan kovin realistinen uhkakuva.

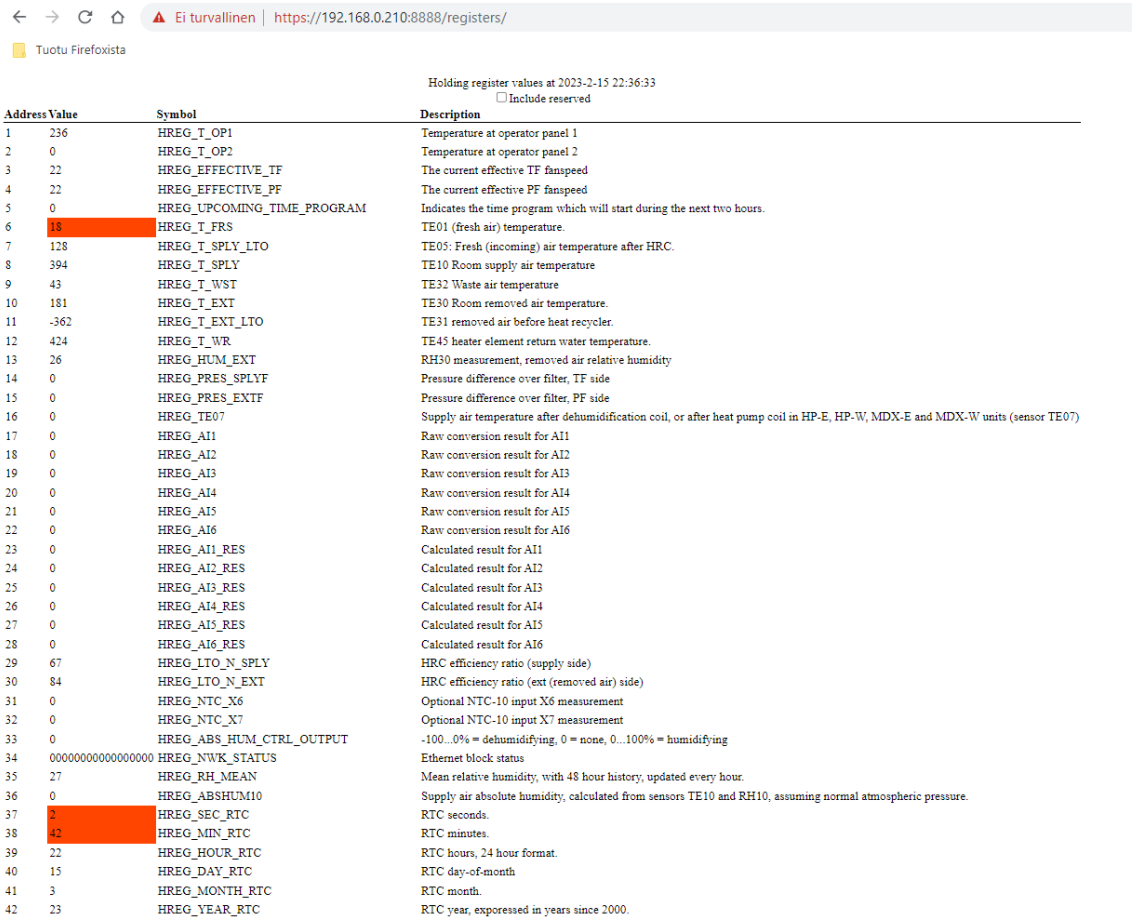
Go-ohjelmointikieli mahdollistaa moniajon aliohjelmia, eli Goroutineita käyttämällä. Taustapäivitys tapahtuu erillisessä funktiossa, jota kutsutaan while-silmukasta tasaisin väliajoin. While-silmukka käynnistetään omalle säikeelleen kutsumalla funktiota go func() -kirjoitusasulla pelkän func()-kutsun sijaan (kuva 12). Huomattavaa on, että Go-ohjelmointikielessä ei ole varsinaista while-silmukaa. Sama toiminta saavutetaan käyttämällä for-silmukkaa ilman parametreja.

```
568 func (p *Pingvin) Monitor(interval int) {
569     for {
570         time.Sleep(time.Duration(interval) * time.Second)
571         p.Debug.Println("Updating values")
572         p.Update()
573     }
574 }

358 func main() {
359     log.Println("enervent-ctrl version", version)
360     configure()
361     device = *pingvin.New(config.SerialAddress, config.Debug)
362     device.Update()
363     go device.Monitor(config.Interval)
364     serve(&config.SslCertificate, &config.SslPrivatekey)
365     device.Quit()
366 }
367
```

KUVA 12. Monitor-aliohjelma ja sen käynnistäminen

Varsinainen tarkkailunäkymä toteutettiin käyttäen HTML:ää ja JavaScriptiä (kuva 13). Näkymää tarkastellaan web-selaimella. JavaScriptillä haetaan REST-rajapinnalta tiedot 5 sekunnin välein ja näkymän taulukon sisältö päivitetään. Ensimmäinen toteutus muodosti näytettävän taulukon for-silmukassa HTML-merkkijonoksi. Tällä toteutuksella 800 rekisterin käsittelemiseen kului jopa 10 sekuntia. Tämä rajoitti myös reaaliaikaisen näkymän päivitysvälin 10 sekuntiin. Kun taulukon muodostus toteutettiin DOM-manipulaatiolla, aika väheni alle yhteen sekuntiin. Näkymään toteutettiin myös muuttuvien arvojen korostaminen tarkkailun helpottamiseksi. Tarvittavat HTML-, JavaScript- ja CSS-tiedostot sisällytetään ohjelman binääritiedostoon kääntövaiheessa.

Näkymästä voidaan kytkeä päälle myös valmistajan varatuiksi määrittämien arvojen näyttäminen. Varatut arvot päätettiin pitää mukana ohjelman käsittelemissä tiedoissa, vaikka ne eivät vaikuttaneetkaan tarjoavan varsinaista hyötyä ohjelman toiminnan kannalta. Tämä päätös kuitenkin osoittautui myöhemmin edulliseksi. Vaihtamalla toimintatiloja virallisesta ohjauspaneelista ja reaaliaikaisia arvoja tarkkailemalla huomattiin, että varatuiksi merkityistä ohjelmallisista pisteistä löytyi arvot Pingvin-laitteen ylipaine-, liesituuletin- ja keskuspölynimuri-toimintatiloille. Tämä mahdollisti toimintojen lisäämisen sovellukseen ja ohjauksen Home Assistant -kojelaudalta.



Address	Value	Symbol	Description
1	236	HREG_T_OP1	Temperature at operator panel 1
2	0	HREG_T_OP2	Temperature at operator panel 2
3	22	HREG_EFFECTIVE_TF	The current effective TF fanspeed
4	22	HREG_EFFECTIVE_PF	The current effective PF fanspeed
5	0	HREG_UPCOMING_TIME_PROGRAM	Indicates the time program which will start during the next two hours.
6	18	HREG_T_FRS	TE01 (fresh air) temperature.
7	128	HREG_T_SPLY_LTO	TE05: Fresh (incoming) air temperature after HRC.
8	394	HREG_T_SPLY	TE10 Room supply air temperature
9	43	HREG_T_WST	TE32 Waste air temperature
10	181	HREG_T_EXT	TE30 Room removed air temperature.
11	-362	HREG_T_EXT_LTO	TE31 removed air before heat recycler.
12	424	HREG_T_WR	TE45 heater element return water temperature.
13	26	HREG_HUM_EXT	RH30 measurement, removed air relative humidity
14	0	HREG_PRES_SPLYF	Pressure difference over filter, TF side
15	0	HREG_PRES_EXTF	Pressure difference over filter, PF side
16	0	HREG_TE07	Supply air temperature after dehumidification coil, or after heat pump coil in HP-E, HP-W, MDX-E and MDX-W units (sensor TE07)
17	0	HREG_AI1	Raw conversion result for AI1
18	0	HREG_AI2	Raw conversion result for AI2
19	0	HREG_AI3	Raw conversion result for AI3
20	0	HREG_AI4	Raw conversion result for AI4
21	0	HREG_AI5	Raw conversion result for AI5
22	0	HREG_AI6	Raw conversion result for AI6
23	0	HREG_AI1_RES	Calculated result for AI1
24	0	HREG_AI2_RES	Calculated result for AI2
25	0	HREG_AI3_RES	Calculated result for AI3
26	0	HREG_AI4_RES	Calculated result for AI4
27	0	HREG_AI5_RES	Calculated result for AI5
28	0	HREG_AI6_RES	Calculated result for AI6
29	67	HREG_LTO_N_SPLY	HRC efficiency ratio (supply side)
30	84	HREG_LTO_N_EXT	HRC efficiency ratio (ext (removed air) side)
31	0	HREG_NTC_X6	Optional NTC-10 input X6 measurement
32	0	HREG_NTC_X7	Optional NTC-10 input X7 measurement
33	0	HREG_ABS_HUM_CTRL_OUTPUT	-100...0% = dehumidifying, 0 = none, 0...100% = humidifying
34	0000000000000000	HREG_NWK_STATUS	Ethernet block status
35	27	HREG_RH_MEAN	Mean relative humidity, with 48 hour history, updated every hour.
36	0	HREG_ABSHUM10	Supply air absolute humidity, calculated from sensors TE10 and RH10, assuming normal atmospheric pressure.
37	2	HREG_SEC_RTC	RTC seconds.
38	42	HREG_MIN_RTC	RTC minutes.
39	22	HREG_HOUR_RTC	RTC hours, 24 hour format.
40	15	HREG_DAY_RTC	RTC day-of-month
41	3	HREG_MONTH_RTC	RTC month.
42	23	HREG_YEAR_RTC	RTC year, exporessed in years since 2000.

KUVA 13. Pitorekisterien reaaliaikainen tarkkailunäkymä

3.4 Home Assistant -kojelaudan luominen

Home Assistantille kulutettavaksi luotiin erillinen REST-polku. Tätä varten muodostettiin oma pingvinStatus-tietue alitietueineen sisältämään tietoja niiden tosielämää vastaavassa muodossa. Erillinen tietue voidaan helposti muuntaa suoraan JSON-muotoiseksi merkkijonoksi encoding/json-pa-

ketin avulla. Esimerkiksi lämpötilat ovat tallennettuna pitorekistereihin kokonaislukuina, joista varsinainen lämpötila saadaan jakamalla kertoimella. Jotkut arvot tulee tulkita bittikenttinä. Muun muassa Pingvin-laitteen käytössä oleva toimintatila tallennetaan 16-bittisenä numerona, jonka kukin bitti edustaa tiettyä toimintatilaa. Esimerkiksi arvo 1, binäärimuotoisena 0000 0000 0000 0001, tarkoittaa että laite on asetettu jäähdyttämään maksimiteholla. Arvo 1024, binäärimuotoisena 0000 0100 0000 0000, puolestaan osoittaa ylipainetilaa. Arvojen käsittelyn helpottamiseksi Home Assistantissa katsottiin parhaaksi muokata ne luettavaan muotoon palvelinpuolella. JSON-tiedonsiirtoformaatti on hyvin tuettu Home Assistantissa.

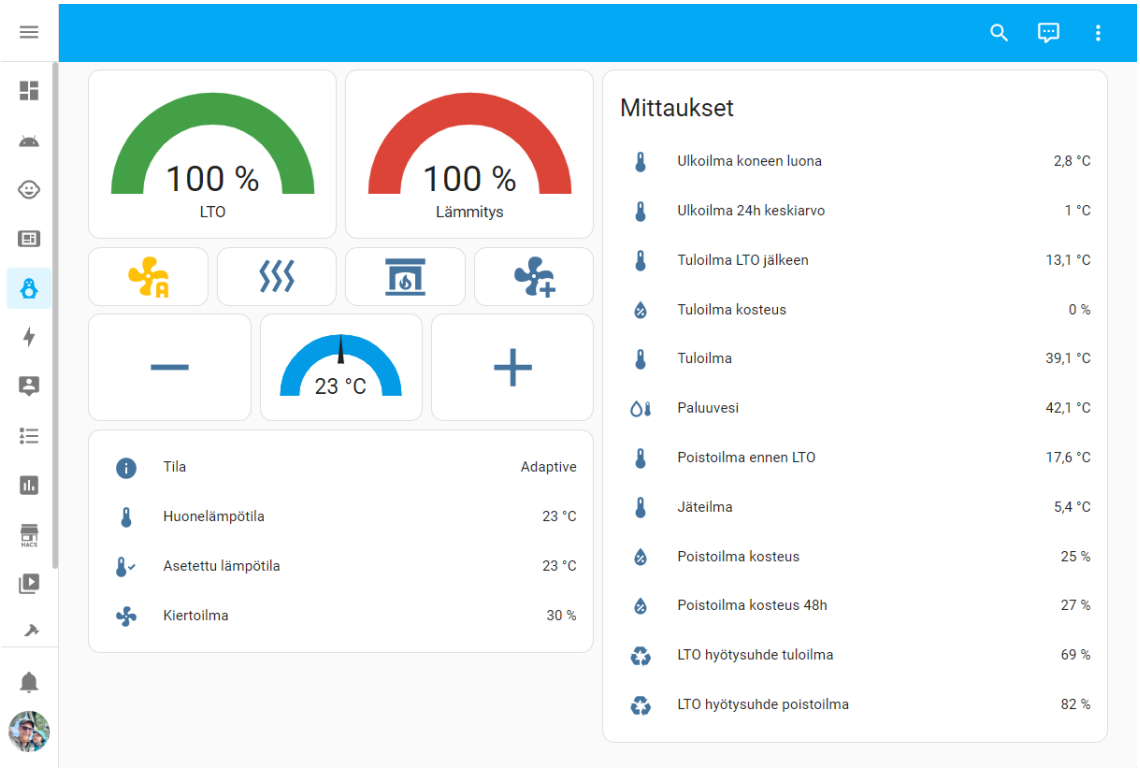
REST-rajapinnan määrittely Home Assistantiin tapahtuu YAML-merkintäkielisillä asetustiedostoilla. Yhdestä rajapintakyselystä voidaan muodostaa useita eri antureita (kuva 14). Antureiden arvojen näyttäminen kojelaudassa voidaan määritellä joko graafisesti Home Assistantin käyttöliittymästä, tai YAML-merkintäkielellä asetustiedostossa. Ohjelman käytön helpottamiseksi mahdollisille ulkopuolisille luotiin valmiit YAML-asetuspohjat rajapinnalle, avustinarvoille, automaatioille sekä suomen- ja englanninkielisille kojelaudoille (kuva 15). Kojelaudan ja anturien määrittely uudelle käyttäjälle tapahtuu kopioimalla annetut esimerkkimäärittelyt Home Assistantin asetustiedostoihin ja vaihtamalla rajapinnan URI sekä käyttäjätunnus ja salasana paikallisiin arvoihin. Home Assistantin uudelleenkäynnistyksen jälkeen lopputuloksena on toimintavalmis kojelauta (kuva 16). Kojelaudan toteutuksessa päätettiin käyttää pelkästään Home Assistantin oletuksena käyttämän Lovelace-käyttöliittymän elementtejä. Vaikka Home Assistant mahdollistaa kaikkien käyttöliittymäelementtien luomisen tai muokkaamisen, katsottiin tuetuissa elementeissä pysymisen tuoma toimintavarmuus edulliseksi. Kojelautaan valittiin päivittäin eniten käytössä olevat toiminnot ja mittaukset.

```
rest:
  - resource: https://IP_ADDRESS:8888/api/v1/status
    scan_interval: 5
    verify_ssl: false
    username: pingvin
    password: enervent
    sensor:
      - name: "Penguin operating mode"
        value_template: "{{ value_json['op_mode'] }}"
        icon: mdi:information
      - name: "Penguin room temperature 1"
        value_template: "{{ value_json['measurements']['room_temp1'] }}"
        unit_of_measurement: "°C"
        icon: mdi:thermometer
```

KUVA 14. Ote REST-rajapintasensorien määrittelystä Home Assistantiin

```yaml
views:
  - title: Penguin
    cards:
      - type: vertical-stack
        cards:
          - type: horizontal-stack
            cards:
              - type: gauge
                entity: sensor.penguin_heat_recovery_pct
                name: LTO
                severity:
                  green: 100
                  yellow: 0
                  red: 0
              - type: gauge
                entity: sensor.penguin_after_heater_pct
                name: Lämmitys
                severity:
                  green: 0
                  yellow: 0
                  red: 100
```

KUVA 15. Ote kojelaudan YAML-kielisestä määrittelystä Home Assistantiin



KUVA 16. Valmis Home Assistant -kojelauta

Ohjaustoimintojen toteuttaminen vaati sisäisten tarkistusten, eli automaatioiden toteuttamista Home Assistantiin. Esimerkiksi maksimilämmitys-toiminto Pingvin-laitteessa pysähtyy, kun asetettu tavoitelämpötila saavutetaan. Jos vallitseva huonelämpötila on yli asetetun tavoitelämpötilan, ei toiminto käynnisty. Jotta Home Assistant saadaan huomioimaan tämä, ei toimintoa vastaavan pai-

nikkeen painaminen lähetä suoraan pyyntöä REST-rajapinnalle. Painallus laukaisee Home Assistant -automaation, jossa tarkistetaan reunaehtojen täyttyminen, tehdään pyyntö REST-rajapinnalle, tarkistetaan toimintatilan muutos, jonka jälkeen painikekytkimen tila vaihdetaan näyttämään lopullista oikeaa tilaa. Maksimilämmityksen automatiikka on kuvattu vuokaaviona liitteessä 2. Muut toiminnot käyttävät samankaltaista logiikkaa. Automaatioiden määrittely voidaan tehdä graafisesti tai YAML-kielellä. Projektin käyttämät automaatiot määriteltiin ensin graafisesti, jonka jälkeen YAML-asetustiedostot laitettiin saataville.

3.5 Mittaushistoria

Pingvin-laite kerää itse mittaamiansa tietoja muistiin. Mittaushistoria on ladattavissa USB-tallennusmedialle (27). Toiminnolle ei kuitenkaan löydy selkeitä käyttöohjeita. Myös Home Assistant kerää siihen määriteltyjen sensorien mittaushistoriaa. Oletuksena Home Assistant säilyttää kuitenkin vain 15 päivän historian (6). Projektin yhteydessä päätettiin toteuttaa mittaustietojen keruu aikasarjatietona ulkoiseen tietokantaan. Tämä helpottaisi mittaushistorian tarkastelua tulevaisuudessa, sekä parantaisi mahdollisen automaation kehittämistä opinnäytetyön sovellukseen.

Projektin tarkoituksiin päätettiin, että tietojen tallennus tapahtuu erilliselle palvelimelle. RPi käyttää oletuksena tallennusmediana microSD-korttia. Vaikka RPi on myös mahdollista määritellä käyttämään myös ulkoista USB-kovalevyä päätallennusmediana, päätettiin sen olevan projektin tarkoituksiin kömpelö ratkaisu (15). Esteettisen fyysisen kokonaisuuden luominen vaatisi joko hintavia valmiita koteloratkaisuja tai käytettävissä olleita kehittyneempiä työstömenetelmiä. MicroSD-kortin kestoikä on riippuvainen kirjoitustapahtumien määrästä (29). Teollisen luokan muistikorttien korkean hinnan takia projektissa käytettiin kuluttajatason High Endurance -muistikorttia. Kortin eliniän maksimoimiseksi päätettiin, että mikä tahansa tietokanta lyhentäisi kortin elinikää mahdollisesti kriittisesti.

Varteenotettavimmiksi tietokantavaihtoehdoiksi valikoituivat InfluxDB ja Prometheus. Vaihtoehdot ovat toimintaperiaatteeltaan erilaisia, mutta suunnattu hyvin samantyyppisiin käyttötarkoituksiin. Projektin käyttöä ajatellen merkittävin ero tietokantojen välillä oli, että InfluxDB toimii työntöperiaatteella (push), kun taas Prometheus keräysperiaatteella (pull). (28.)

Tietokantavaihtoehtoja vertaillessa päädyttiin Prometheuksen käyttöön, koska sen käyttäminen ei vaadi projektin sovellukseen erillistä asetusten määrittelyä. Prometheuksen kohdesovellukset tarjoavat HTTP-polun, Prometheuksen käyttämällä termistöllä *exporter*, josta mittaukset ovat luettavissa. Mittaustietojen keräämisen määrittely tapahtuu täysin Prometheus-tietokannan asetuksissa. Ulkoinen Prometheus-palvelin kyselee määritellyin väliajoin arvot tästä HTTP-polusta. Prometheuksen kehittäjädokumentaatiossa suositellaan, että avaimet arvoineen luodaan dynaamisesti kyselyn saapuessa sen sijaan, että tiedot kerättäisiin ja pidettäisiin muistissa kohdesovelluksessa taustalla jatkuvasti (30). Tätä ohjenuoraa seurattiin myös projektin sovelluksen toteutuksessa. Tämän takia Prometheus-tietokantatuki ei aiheuta projektin sovelluksessa resurssien käyttöä muulloin, kuin mittauksia kyseltäessä. Prometheus-HTTP-polun tiedot ovat selkokielisiä ja luettavissa verkkoselaimella (kuva 16).

```
# HELP pingvin_hreg_007_t_sply_lto TE05: Fresh (incoming) air temperature after HRC.
# TYPE pingvin_hreg_007_t_sply_lto gauge
pingvin_hreg_007_t_sply_lto 12.9
# HELP pingvin_hreg_008_t_sply TE10 Room supply air temperature
# TYPE pingvin_hreg_008_t_sply gauge
pingvin_hreg_008_t_sply 39.2
# HELP pingvin_hreg_009_t_wst TE32 Waste air temperature
# TYPE pingvin_hreg_009_t_wst gauge
pingvin_hreg_009_t_wst 4.3
# HELP pingvin_hreg_010_t_ext TE30 Room removed air temperature.
# TYPE pingvin_hreg_010_t_ext gauge
pingvin_hreg_010_t_ext 18.1
# HELP pingvin_hreg_011_t_ext_lto TE31 removed air before heat recycler.
# TYPE pingvin_hreg_011_t_ext_lto gauge
pingvin_hreg_011_t_ext_lto -36.2
# HELP pingvin_hreg_012_t_wr TE45 heater element return water temperature.
# TYPE pingvin_hreg_012_t_wr gauge
pingvin_hreg_012_t_wr 42.1
```

KUVA 16. Prometheus-exporter-tietoja verkkoselaimella

Ohjelmointi toteutettiin Prometheuksen omia Go-ohjelmointikielen asiakaskirjastoja käyttäen. Kirjastot tarjoavat net/http-kirjaston kanssa yhteensopivan promhttp.Handler-funktion, joka määriteltiin käsittelemään polkuun /metrics saapuvia pyyntöjä (kuva 17). Prometheus-mittaustietojen muodostamista varten Pingvin-tietueeseen luotiin Describe- ja Collect-jäsenfunktiot. Pingvin-tietueesiintymä rekisteröidään Prometheus-metriikkalähteeksi prometheus.MustRegister-funktiolla (kuva 18). Describe ja Collect ovat Prometheus-kirjastojen vaatimat funktiot, jotka hyödyntävät Go-ohjelmointikielen kanavia (kuva 19). Kanavien avulla voidaan viestiä sovelluksen eri osien välillä. HTTP-pyynnön saapuessa promhttp.Handler-funktiolle, se kutsuu Describe- ja Collect-funktioita. Describe-funktiossa kirjoitetaan kanavalle jokaisen senhetkisen mittaustiedon nimi, kuvaus sekä tyyppi.

Tämän jälkeen Collect-funktio asettaa arvot jokaiselle Describe-funktion muodostamalle mittaustiedolle ja kirjoittaa ne kanavalle. Promhttp.Handler lukee mittaukset kanavalta ja kirjoittaa tekstimuotoisen vastauksen HTTP-pyyntöön.

```
// Start the HTTP server
func serve(cert, key *string) {
    log.Println("Starting service")
    http.HandleFunc("/api/v1/coils/", authHandlerFunc(coils))
    http.HandleFunc("/api/v1/status", authHandlerFunc(status))
    http.HandleFunc("/api/v1/registers/", authHandlerFunc(registers))
    http.HandleFunc("/api/v1/temperature/", authHandlerFunc(temperature))
    if config.EnableMetrics {
        http.Handle("/metrics", promhttp.Handler())
    }
```

KUVA 17. HTTP-polkujen ja niitä käsittelevien funktioiden määrittelyä

```
    if config.EnableMetrics {
        log.Println("Prometheus exporter enabled (/metrics)")
        prometheus.MustRegister(&device)
    }
```

KUVA 18. Pingvin-tietueen määrittely Prometheus-metriikkalähteeksi

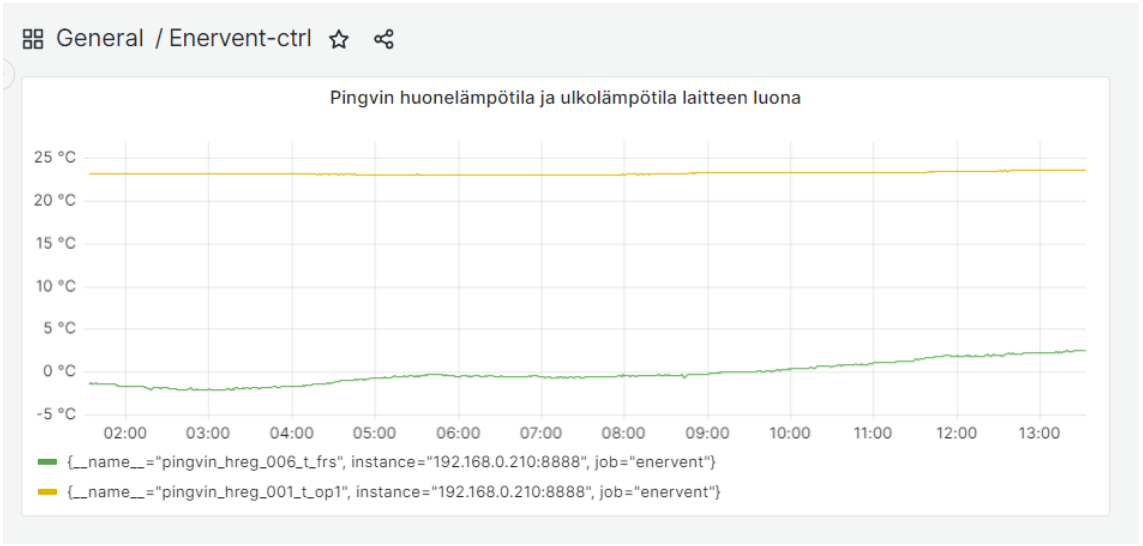
```go
// Implements prometheus.Describe()
func (p *Pingvin) Describe(ch chan<- *prometheus.Desc) {
    for _, hreg := range p.Registers {
        if !hreg.Reserved {
            ch <- hreg.PromDesc
        }
    }
    for _, coil := range p.Coils {
        if !coil.Reserved {
            ch <- coil.PromDesc
        }
    }
}

// Implements prometheus.Collect()
func (p *Pingvin) Collect(ch chan<- prometheus.Metric) {
    for _, hreg := range p.Registers {
        if !hreg.Reserved {
            ch <- prometheus.MustNewConstMetric(
                hreg.PromDesc,
                prometheus.GaugeValue,
                float64(hreg.Value)/float64(hreg.Multiplier),
            )
        }
    }
    for _, coil := range p.Coils {
        val := 0
        if coil.Value {
            val = 1
        }
        if !coil.Reserved {
            ch <- prometheus.MustNewConstMetric(
                coil.PromDesc,
                prometheus.GaugeValue,
                float64(val),
            )
        }
    }
}
```

KUVA 19. Pingvin-tietueen Describe- ja Collect-funktiot

Mittaustiedot muodostetaan jokaiselle HTTP-pyynnölle uudestaan, eikä niitä säilytetä pinomuistissa pyyntöjen välillä. Koska mittaustietoja olisi tarkoitus säilyttää ainakin kahden vuoden ajalta, jätettiin varatut arvot Prometheus-metriikoista pois tietokannan koon rajoittamiseksi. Sovellukseen toteutettiin asetusvaihtoehto Prometheus-metriikan poiskytkemiseksi.

Prometheus tarjoaa mittausten tarkasteluun oman selainpohjaisen käyttöliittymänsä (kuva 20). Mittausten syvempään tarkasteluun se ei kuitenkaan sovellu. Jos halutaan esittää useampia mittauksia samassa kaaviossa, voidaan käyttää esimerkiksi Grafanaa (kuva 21).

KUVA 20. Prometheuksen tarkastelunäkymä



KUVA 21. Grafana-kaavio huone- ja ulkoilman lämpötilasta 12 tunnin aikana

3.6 Asetusarvojen muuttaminen

Pingvin-laitteen RS-485-väylän yli muutettavien asetusten määrä paljastui esikokeiluissa rajalliseksi. Home Assistantin kautta saataville laitettavat asetusarvot rajoittuivat siten pääasiassa samoihin toimintoihin, jotka löytyvät Pingvinin omasta hallintapaneelista ilman syvempää asetusvalikoiden selaamista.

Pingvin-laitteen toimintatilan muuttamiseksi toteutettiin käärefunktio yksittäisten ohjelmallisten pisteiden tilan pakottamiseksi (kuva 22). Käärefunktioita päätettiin käyttää lähdekoodin luettavuuden

parantamiseksi ja koodintoiston välttämiseksi. Taustatietoa etsiessä löytyi GitHub-alustalta toiselle Enervent-laitteelle Node.js-ajoympäristöllä toteutettu sovellus, Jalle19/modbus-eda-bridge (31). Modbus-eda-bridgen GitHub-arkistoa tutkimalla havaittiin, että tilaa muuttaessa vain yksi toiminta-tiloja edustavista ohjelmallisista pisteistä voi saada arvon 1 samanaikaisesti. Tätä varten toteutettiin apufunktio, jossa tarkistetaan, kuuluuko pakotettava ohjelmallinen piste näiden toisensa poissulkeviin arvoihin. Jos pakotettava ohjelmallinen piste kuuluu näihin arvoihin ja piste halutaan pakottaa arvoon 1, pakotetaan kaikki toisensa poissulkevat ensin arvoon 0 (kuva 23). Funktion lopuksi käytetään ReadCoil-funktiota päivittämään pisteen arvo sovelluksen tietueessa. Huomattavaa kuvassa 23 on, että funktio käyttää tarkoitukseen sovellukseen kiinteästi ohjelmoitua listaa mutexcoils.

```
// Force a single coil
func (p *Pingvin) WriteCoil(n uint16, val bool) bool {
    if val {
        p.checkMutexCoils(n, p.handler)
    }
    var value uint16 = 0
    if val {
        value = 0xff00
    }
    p.buslock.Lock()
    results, err := p.modbusclient.WriteSingleCoil(n, value)
    p.buslock.Unlock()
    if err != nil {
        log.Println("ERROR: WriteCoil: ", err)
    }
    if (val && results[0] == 255) || (!val && results[0] == 0) {
        log.Println("WriteCoil: wrote coil", n, "to value", val)
    } else {
        log.Println("ERROR: WriteCoil: failed to write coil")
        return false

    }
    p.ReadCoil(n)
    return true
}
```

KUVA 22. Yksittäisen ohjelmallisen pisteen pakottaminen

```
// Some of the coils are mutually exclusive, and can only be 1 one at a time.
// Check if coil is one of them and force all of them to 0 if so
func (p *Pingvin) checkMutexCoils(addr uint16, handler *modbus.RTUClientHandler) error {
    for _, mutexcoil := range mutexcoils {
        if mutexcoil == addr {
            for _, n := range mutexcoils {
                if p.Coils[n].Value {
                    p.buslock.Lock()
                    _, err := p.modbusclient.WriteSingleCoil(n, 0)
                    p.buslock.Unlock()
                    if err != nil {
                        log.Println("ERROR: checkMutexCoils:", err)
                        return err
                    }
                }
            }
            return nil
        }
    }
    return nil
}
```

KUVA 23. Toisensa poissulkevien ohjelmallisten pisteiden tarkistaminen ja nollaus

Modbus-protokollaan kuuluu komento useiden ohjelmallisten pisteiden tai pitorekisterien arvojen muuttamiseksi yhdellä kertaa. Lopullista toisensa poissulkevien pisteiden tarkistusfunktion toteutustapaa valitessa päädyttiin kirjoittamaan jokainen yksittäinen piste erikseen. Toimintatiloja edustavat ohjelmalliset pisteet eivät ole sijoitettu peräkkäin, jolloin kaikkien kirjoittamiseksi yhdellä komennolla oli tarpeellista lukea ensin myös toimintatilapisteiden välissä sijaitsevat muut pisteet. Vastauksena saadusta tavutaulukosta tuli sen jälkeen muuttaa halutut bitit arvoon 0, jonka jälkeen kaikki joukkoon kuuluvat arvot pakotetaan muodostuneen tavutaulukon arvoihin. Kokeilemalla eri menetelmien välillä ei havaittu merkittävää nopeuseroa. Usean ohjelmallisen pisteen yhdellä komennolla kirjoittamiseen tarvittu koodi oli kuitenkin huomattavasti lopullista toteutustapaa monimutkaisempi.

Pitorekisteriarvoista ainoaksi hyödylliseksi muutettavissa olevaksi arvoksi todettiin toivottua lämpötilaa edustava pitorekisteri. Mahdollista myöhempää käyttöä varten yksittäisten pitorekisterien kirjoittamiselle toteutettiin myös käärefunktio. Kohdelämpötilan muuttamiseksi kirjoitettiin erillinen jäsenfunktio Pingvin-tietueeseen, jolla toivottua lämpötilaa voidaan muuttaa yhden asteen askeleilla ylös-, tai alaspäin. Funktio voi ottaa parametrina myös uuden kohdelämpötilan suoraan numeroarvona. Asetettavat arvot päätettiin rajoittaa 20:n ja 30 °C:n välille. Näiden raja-arvojen ulkopuolisten lämpötilojen arvioitiin olevan käytännössä tarpeettomia.

Olemassa oleviin /api/v1/coils ja /api/v1/registers REST-rajapintapolkuihin lisättiin toiminnallisuus muuttaa arvoja HTTP PUT-pyynnöllä. Esimerkiksi aktiivisen lämmityksen estäminen tapahtuu pakottamalla ohjelmallinen piste 54 arvoon 0. REST-rajapinnalla tämä komento suoritetaan lähettämällä PUT-pyyntö polkuun /api/v1/coils/54/0. REST-komennot päätettiin pitää yksinkertaisina, jotta niiden YAML-merkintäkielinen määrittely Home Assistantiin säilyisi mahdollisimman hyvin luettavana. Muuttujien siirtämisellä HTTP-pyynnön polusta pyynnön runkoon ei arvioitu saavutettavan mainittavaa hyötyä kotiverkkoympäristössä.

3.7 Lämmitystehon hallinta

Lämmityksen hallintaan vaikutti jo ennen projektin alkamista kaksi automatiikkajärjestelmää. Rakennuksen teknisessä tilassa kaukolämpökeskuksen ohjainyksikkö, Ouman EH-203, säätää lämmitykseen käytettävän veden lämpötilaa ulkolämpötilan mukaan. Ouman EH-203 mittaa ulkolämpötilaa omalla rakennuksen ulkopuolelle sijoitetulla lämpötila-anturillaan (32). Enervent Pingvin Kotilämpö säätelee tuloilmaa lämmittävän kennon läpi virtaavan lämpimän veden määrää kennon yhteydessä olevalla kolmitieventtiilillä. Lisäksi Pingvin ohjaa myös kiertoilmapuhaltimen nopeutta mukautuvan puhallintilan ollessa valittuna. Pingvin mittaa ulkolämpötilaa laitteen luona yksikköön integroidulla anturilla (2).

Ulkolämpötilan laskiessa Ouman nostaa lämmitysveden lämpötilaa (32). Aikaisempien talvien aikana on havainnoitu, että lämmitysveden lämpötilan noustessa Pingvin vähentää lämmityskennon läpi virtaavan lämpimän veden määrää pitääkseen kiertoilman lämpötilan vakiona. Lisäksi Pingvin on vaikuttanut pyrkivän lämmittämään kiertoilmaa mahdollisimman vähän. Laite nostaa kiertoilman lämpötilaa hitaasti, suosien kiertoilmapuhaltimen nopeuden nostoa. Jopa käyttäjän valitessa erillisen maksimilämmitystilan, on havainnoitu täyden kennon lämmitysvesivirtauksen saavuttamiseen kuluvan usein tunteja. Kiertoilman pieni lämpötilaero huoneilmaan yhdistettynä suurempaan virtausnopeuteen johtavat vedon tunteeseen asunnossa. Näistä syistä kiertoilman minimilämpötila oli säädetty Pingvin-laitteen asetuksista korkeaksi, 45 °C:seen, jo ennen projektin aloitusta. Tällä asetuksella Pingvinin lämmityskennon venttiili on n. -20 °C:n ulkolämpötiloihin saakka täysin auki. Lämpötilaa säätelee tällöin käytännössä ainoastaan Ouman-yksikkö.

Esikartoitusvaiheessa havaittiin, että lämmityskennon kolmitieventtiiliin ei pysty suoraan vaikuttamaan RS-485-väylän kautta muutettavissa olevilla asetusarvoilla. Samoin kiertoilmapuhaltimen nopeuden suoran muuttamisen havaittiin olevan mahdollista vain laitteen omasta hallintapaneelista. Lämmitystehon vaihtelu REST-rajapinnalta mahdollistettiin Pingvinin asetusten luovalla käytöllä. Pingvinin käyttötilat ovat valittavissa RS-485-väylän kautta. Näihin kuuluvat kiertoilmapuhaltimen nopeuden mukautuva- ja vakionopeustila, sekä maksimilämmitys ja -jäähdytys. Kiertoilmapuhaltimen vakionopeustila esiasetettiin pienimpään mahdolliseen, eli 20 %:n nopeuteen. Kiertoilmapuhaltimen mukautuvalle tilalle annettiin nopeusalueeksi 20–38 %.

Normaalitilassa kiertoilman mukautuva puhallintila on valittuna. Pingvinin havainnoitiin käytännössä pitävän nopeuden 30 %:ssa jatkuvasti. Kun lämmitystehoa halutaan nostaa, valitaan laitteeseen maksimilämmitystila. Tällöin Pingvin nostaa kiertoilmapuhaltimen nopeuden suurimpaan sallittuun arvoon, 38 %, kunnes mitattu huonelämpötila saavuttaa asetetun kohdearvon. Toivotun huonelämpötilan saavutettuaan laite pudottaa kiertoilmapuhaltimen nopeuden takaisin 30 %:iin. Jos lämmitystehoa halutaan pienentää normaalitilasta, valitaan kiertoilmapuhaltimen vakionopeustila, jolloin kiertoilmapuhallin toimii 20 %:n teholla.

Projektin sovelluksen kannalta lämpötilan ohjaus tapahtuu kahden ohjelmallisen pisteen arvoja vaihtamalla. Kiertoilmapuhaltimen mukautuva nopeustila asetetaan päälle pakottamalla ohjelmallinen piste 11 arvoon 1. Maksimilämmitystila asetetaan pakottamalla ohjelmallinen piste 6 arvoon 1. Huomioitavaa toiminnassa on, että kiertoilmapuhaltimen ollessa vakionopeustilassa maksimilämmitystila ei vaikuta kiertoilmapuhaltimen nopeuteen. Lämmitystehon nostamiseksi pisteet 11 ja 6 tulee pakottaa arvoon 1. Vähennetty lämmitysteho, eli kiertoilmapuhaltimen vakionopeustila, kytketään päälle pakottamalla piste 11 arvoon 0. Tarvittaessa kiertoilman aktiivinen lämmitys voidaan myös estää kokonaan käyttämällä maksimijäähdytystilaa, pistettä 7, tai lämmityksen estämistä edustavaa erillistä ohjelmallista pistettä 32.

Home Assistant -kojelaudan maksimilämmitystä ilmaisevan painikkeen oikean tilan näyttämiseksi täytyi toteuttaa Home Assistantin sisäinen automaatio, joka tarkkailee ohjelmallisen pisteen 6 arvoa. Arvon muuttuessa painikkeen tila asetetaan pisteen oikean arvon mukaan. Samankaltainen automaatio tarvittiin myös muille toimintatiloille. Tilat kytkeytyvät pois päältä tietyn ajanjakson jälkeen Pingvin-laitteen toimesta (2). Ajanjakso on tilakohtaisesti käyttäjän asetettavissa, mutta ajastimia ei saa kytkettyä kokonaan pois päältä.

3.8 Sovelluksen asetukset

Sovelluksen asetusten muuttamiseen toteutettiin mahdollisuus määrittää asetuksia joko parametreina ohjelmaa komentoriviltä ajettaessa, tai YAML-merkintäkielisen asetustiedoston kautta. Komentoriviparametrit yliajavat asetustiedoston määritteet.

Käynnistyessään ohjelma tarkistaa asetustiedoston olemassaolon, ja kirjoittaa uuden oletusarvoilla tarvittaessa. Määriteltävät asetusarvot ovat käyttöjärjestelmän sarjaportin osoite, REST-rajapinnan käyttämä TCP-portti, polut SSL-sertifikaatille ja yksityiselle avaimelle, HTTP Basic Auth -käyttäjätunnus ja salasana, lukemien taustapäivityksen väliaika, Prometheus-metriikan päälle kytkeminen, lokitiedoston polku, saapuvien HTTP-pyyntöjen lokiin kirjaamisen päälle kytkeminen sekä vianetsintälokikirjausten päälle kytkeminen.

3.9 Tietoturva

Sovelluksen ensisijainen käyttöympäristö on koti- tai paikallisverkko. Useimmiten tämä tarkoittaa, että verkkoliikenne julkisesta Internetistä verkon laitteille on estetty palomuurilla. Tästä huolimatta sovelluksen verkkoliikenne päätettiin salata SSL-salauksella. Käynnistyessään ohjelma luo tarvittaessa uuden SSL-sertifikaatin ja -avaimen, joita käytetään HTTPS-liikenteen salaamiseen. Sovellus voidaan määrittää käyttämään itse luotua tai luotetun tahon allekirjoittamia sertifikaatteja.

REST-rajapinta suojattiin HTTP Basic Auth -menetelmällä. Valittaessa kirjautumismenetelmää todettiin, että käyttäjätunnus–salasanapari riittää suojaamaan pääsy sovelluksen pääasiallisessa toimintaympäristössä. Käyttäjän näkökulmasta toiminto on selkeä, ja Home Assistant tukee käyttäjätunnusta sekä salasanaa. Käyttäjätunnus ja salasana ovat käyttäjän muutettavissa. Ajonaikaiseen muistiin tallennetaan käyttäjätunnuksen ja salasanan SHA256-hash, jotka eivät ole takaisinpäin pääteltävissä kohtuullisella laskentateholla (33). Tällä ratkaisulla estetään myös käyttäjätunnus-salasanaparin päätteleminen pyyntöjen käsittelyajasta (34). Menetelmää ei tosin koettu sovelluksen käyttöympäristöön nähden realistiseksi uhkakuvaksi, lisäksi RPi Zero W:n resurssien rajallisuuden tuomat vaihtelut pyyntöjen käsittelyajassa tekevät menetelmän muutenkin kyseenalaiseksi.

Sovellus ei käsittele arkaluontoista tietoa. Sovellus itse ei myöskään tallenna mittaustietoja pysyvään tallennustilaan. Poikkeus tähän on valinnainen lokitiedosto, joka voi sisältää REST-rajapintaan tulleet pyynnöt IP-osoitteineen. Pyyntöjen kirjaaminen lokiin ei ole oletuksena päällä, ja sen päälle kytkeminen on käyttäjän valittavissa. Päätökset sovelluksen pitämisestä mahdollisimman tilattomana ja pysyvän tallennustilan käytön minimoinnista perustuivat tosin lähinnä tallennusmedian käyttöiän pidentämiseen.

Tietoturvaa olisi ollut mahdollista parantaa siirtämällä REST-rajapinnan käyttämät parametrit HTTP-pyynnön polusta pyynnön runkoon. Menetelmän käyttäminen olisi vaatinut Home Assistant -integraation monimutkaisempia YAML-merkintäkielisiä asetuksia. Todennäköisyys sille, että kotiverkosta löytyy verkkoliikennettä tarkkaileva haittaohjelma, joka hyötyy pyyntöjen yksityiskohtien päättelemisestä pyynnön polusta, arvioitiin pieneksi. Käyttäjätunnus, salasana ja vastauksen tiedot ovat SSL-salattuja siirron aikana.

Pääsy asetustiedostoon vaatii käyttöjärjestelmätason tunnistautumisen, tai fyysisen pääsyn käsiksi laitteen SD-korttiin. Käyttöjärjestelmän verkkopääsyn turvaaminen koettiin käyttäjän vastuuksi, eikä aihe kuulu tähän opinnäytetyöhön. Asetustiedosto on selkokielinen. Käyttäjätunnuksen ja salasanan salaamisella ei koettu saavutettavan merkittävää tietoturvaa.

4 PROJEKTIN TULOKSET

Enervent Pingvin Kotilämpö eAir -laitteen ohjaus ja mittaustietojen tarkastelu saatiin toteutettua Home Assistantiin. Kojelaudan käyttö on koettu parannukseksi laitteen omaan ohjauspaneeliin verrattuna. Päivittäisessä käytössä tarvittavat asetusarvojen muuttamiset, lähinnä lämmitystehon lisääminen tai vähentäminen, on koettu noin kuukauden käyttökokeilun aikana hyvin toimivaksi. Hetkellisten mittaustietojen lukemat ovat saatavilla Home Assistant -kojelaudalla.

Pingvin-laitteen toimintaan puuttumalla saavutettavissa oleva energiansäästö havaittiin jo projektin alkuvaiheessa ennakkoon arvioitua pienemmäksi. Laitteen sähköenergian kulutusta mitattiin verkkovirtapistokkeeseen kytkettävällä mittarilla. Pingvin-laitteen kiertoilmapuhallin toimi kahden vuorokauden ajan suurimmalla asetuksissa sallitulla pyörimisnopeudella. Laitteen ottosähkötehoksi mitattiin 40 W. Ennakkoon sähköenergian kulutus oli arvioitu eri suuruusluokkaan.

Tasaisen huonelämpötilan ylläpitäminen vaatii talviaikaan kiertoilmapuhaltimen pyörimisen suurimmalla sallitulla nopeudella suurimman osan ajasta. Kesäaikaan kiertoilmapuhallin on jatkuvasti hitaimmalla asetuksella. Mahdolliset säästöt olisivat saavutettavissa ulkolämpötilan vaihteluiden yhteydessä. Sähköenergian kulutuksen ollessa jo lähtökohtaisesti pientä, myös saavutettava säästö jää pieneksi.

Automaatio jouduttiin jättämään toistaiseksi toteuttamatta. Kevät 2023 oli sääolosuhteiltaan tasainen. Aiempina talvina koettuja sisälämpötilan vaihteluita ei kuukauden kestäneellä tarkkailujaksolla juuri koettu. Automaation kehittäminen todettiin liian arvaamattomaksi testitilanteiden puuttuessa. Projektin kehitys ei kuitenkaan lopu tämän opinnäytetyön viimeistelyyn. Sovellus ja Home Assistant jäävät päivittäiseen käyttöön.

5 POHDINTA

Opinnäytetyön lopputulokseen oltiin pääosin tyytyväisiä. Päätavoite kodin laitteiden hallinnan ja tarkastelun keskittämiseksi Home Assistantiin saavutettiin ja lopputulos paransi Pingvin-laitteen käytettävyyttä. Energiansäästötavoite jäi todennäköisesti saavuttamatta. Tässä onnistumista arvioidessa täytyy ottaa huomioon ulkoisten tekijöiden, lähinnä säätilan, vaikutus. Epäsuorana onnistumisena voidaan tosin pitää lämmitystehon muuttamisen helpottumista. Koska kynnys tehon pienentämiselle on projektin myötä matalampi, johtaa tämä luultavasti myös energiansäästöön. Toisaalta myös lämmitystehon lisääminen helpottui, joka puolestaan voi johtaa lisäkulutukseen.

Automaatio tullaan todennäköisesti lisäämään sovellukseen myöhemmin. Sovelluksen oltua käytössä kuukauden todettiin kolmena päivänä tarvetta lämmityksen tehostamiselle. Näiden perusteella on kuitenkin vaikea arvioida, minkä tyyppistä automaatiota kaivataan. Testitilanteen luominen esimerkiksi lämpötila-anturia viilentämällä ei anna todellista kuvaa automaation toimivuudesta, ja aktiivisessa käytössä olevan asuintalon ovien availeminen talon viilentämiseksi ei vaikuta houkuttelevalta vaihtoehdolta. Automaatio saatetaan toteuttaa Home Assistantia käyttäen sen sijaan, että se toteutettaisiin itse projektin sovellukseen. Home Assistant mahdollistaa muun muassa sääennusteen hyödyntämisen automaatioissa helposti, käyttämällä esimerkiksi tuettua OpenWeatherMap-integraatiota.

Go-ohjelmointikieli osoittautui suunnitteluperiaatteitansa vastaavaksi. Ohjelmointikielen luettavuudesta voidaan tosin olla monta mieltä. Esimerkiksi Python-ohjelmointikieli koettiin helpommin luettavaksi. Go-ohjelmointikielen staattinen tyypitys sekä luokkajärjestelmän puuttuminen aiheuttivat alkuun päänvaivaa. Kun muuttujien tyypin ennalta määräämiseen ja tietueiden käyttöön luokkien sijasta totuttiin, tuntui niiden käyttö luontevalta. Myös omalaatuinen virheenkäsittelyjärjestelmä koettiin totuttelun jälkeen tehokkaaksi.

REST-rajapinnan toteuttaminen Go-ohjelmointikielellä koettiin vaivattomaksi. Suurena hyötynä mainittakoon useimpien toimintojen onnistumisen pelkästään standardikirjastoa käyttäen. Go-ohjelmointikielen tee se itse -asenne koettiin miellyttäväksi. Toimintojen toteuttaminen itse valmiiden kirjastojen käyttämisen sijaan koettiin parantavan ymmärrystä sovelluksen toimintaan, ja antavan enemmän mahdollisuuksia yksityiskohtien hallintaan.

Opinnäytetyön toteutuksen aikana Go-ohjelmointikielen ominaisuuksiin saatiin hyvää näkemystä, ja useat alkuvaiheessa toteutetut funktiot tullaankin todennäköisesti muokkaamaan paremmin kielen ominaisuuksia hyödyntäväksi. Projektin lähdekoodi on julkiseksi saatavilla GitHub-alustalla muiden Pingvin-laitteen omistajien ja Home Assistant -käyttäjien kokeiltavaksi (35). Hyvällä onnella muitakin käyttäjiä ja jopa kehitystyöhön halukkaita löytyy. Toisaalta kuukauden kestäneessä käytössä ongelmia ei ole juurikaan paljastunut. Riskinä on jatkokehityksen tekemättä jääminen, jos ulkopuolista painetta jatkokehitykselle ei löydy.

Opinnäytetyön aihe koettiin uuteen ohjelmointikieleen tutustumisen näkökulmasta hyväksi. Sovellus on periaatteessa aika yksinkertainen, mutta vaatii toimintavarmuutta ja myös matalatasoista operointia bittitasolla. Tämän opinnäytetyön tarkoitukseen koettiin Go-ohjelmointikielen olevan tarkoituksenmukainen valinta.

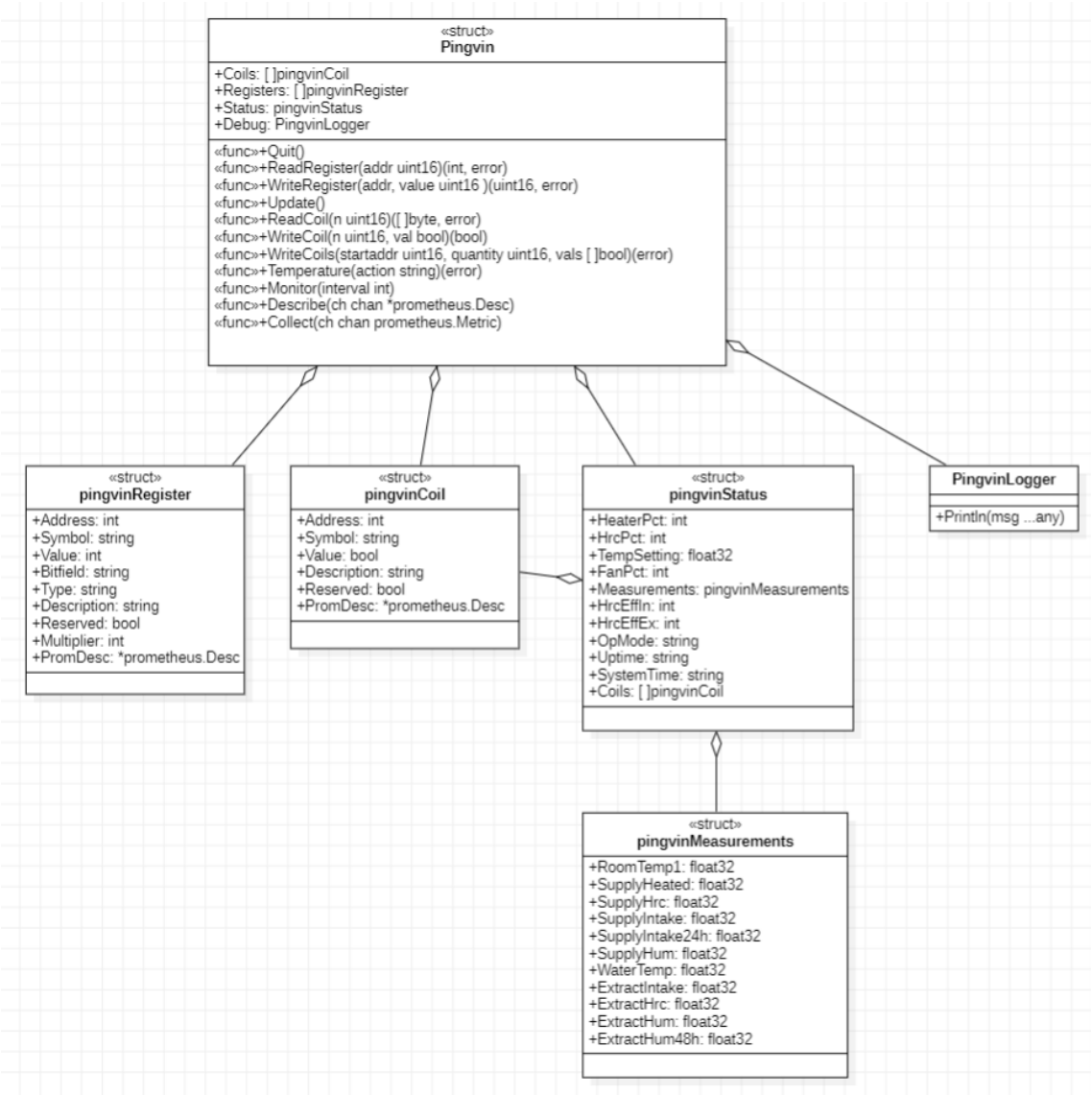
LÄHTEET

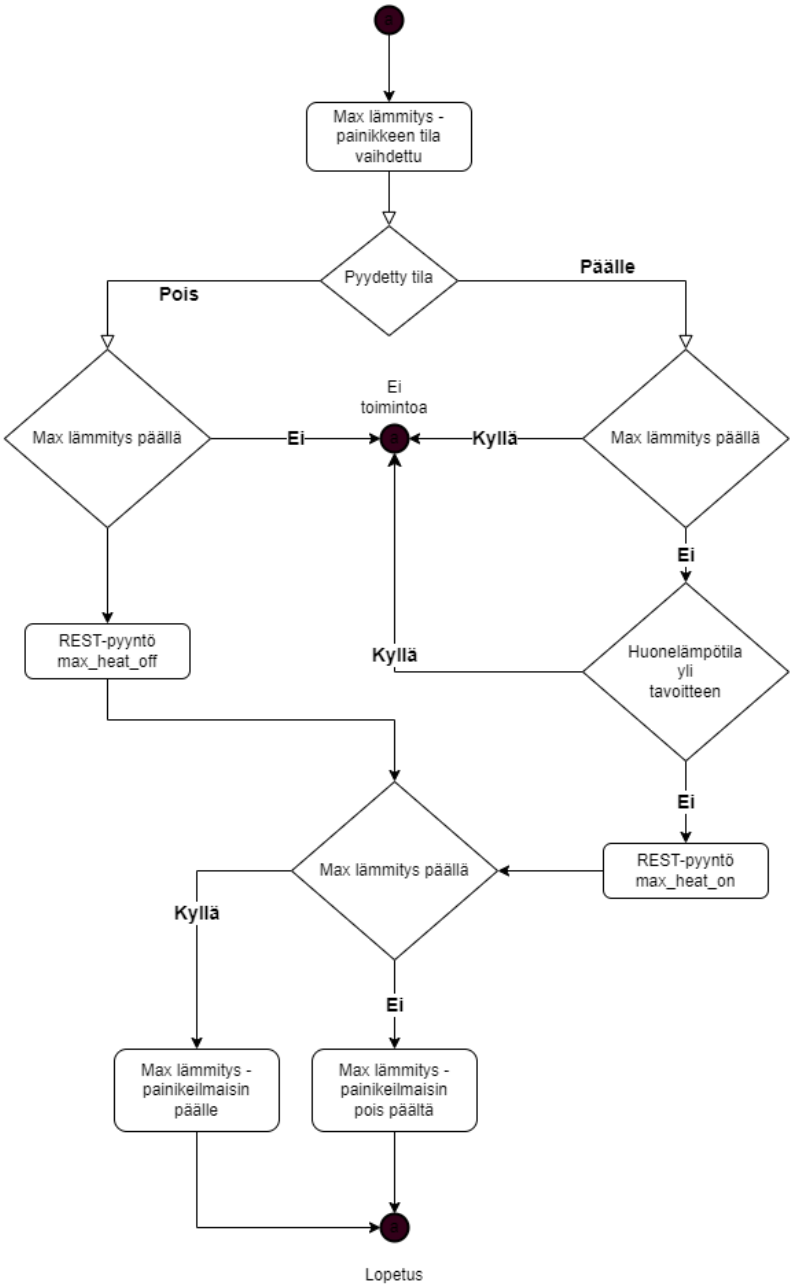
1. Enervent Zehnder Oy 2023. Salla Kotilämpö eAir. Hakupäivä 11.2.2023. https://www.enervent.fi/pingvin-kotilampo-eair/
2. Enervent Zehnder Oy 2018. Enervent eAir. Asennusohje. Hakupäivä 11.2.2023. https://doc.enervent.com/op/op.ViewOnline.php?documentid=940&version=1
3. Home Assistant 2023. Modbus. Hakupäivä 11.2.2023. https://www.home-assistant.io/integrations/modbus/
4. Butler, Brandon 2016. Docker containers are coming to Windows. Network World 2016. Hakupäivä 11.3.2023. https://www.networkworld.com/article/3123697/docker-containers-are-coming-to-windows.html
5. Home Assistant 2023. Home Assistant Core Initial Commit. Hakupäivä 10.3.2023. GitHub. https://github.com/home-assistant/core/commit/d55e4d53cccc9123d03f45c53441e7cbfc58e515
6. Home Assistant 2023. Installation. Hakupäivä 10.3.2023. https://www.home-assistant.io/installation/
7. Wikipedia 2023. Home Assistant. Hakupäivä 10.3.2023. https://en.wikipedia.org/wiki/Home_Assistant
8. Texas Instruments 2010. RS-422 and RS-485 Standards Overview and System Configurations. Hakupäivä 10.3.2023. https://www.ti.com/lit/an/slla070d/slla070d.pdf
9. Modbus 2012. Modbus Application Protocol Specification. Hakupäivä 10.3.2023. https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf
10. Schneider Electric 2020. Modbus Master-Slave Principle. Hakupäivä 11.3.2023. https://product-help.schneider-electric.com/ED/ES_Power/NT-NW_Modbus_IEC_Guide/EDMS/DOCA0054EN/DOCA0054xx/Master_NS_Modbus_Protocol/Master_NS_Modbus_Protocol-2.htm
11. Enervent Zehnder Oy 2018. eAir MD Modbus Register List. Hakupäivä 11.12.2022. https://doc.enervent.com/out/out.ViewDocument.php?documentid=59&showtree=1
12. DPS Telecom 2023. Understanding Modbus Protocol - RTU vs TCP vs ASCII. Hakupäivä 11.3.2023. https://www.dpstele.com/modbus/index.php
13. Cyburt, Bruce 2012. Introduction to Modbus. Automation.com. Hakupäivä 11.3.2023. https://www.automation.com/en-us/articles/2012-1/introduction-to-modbus
14. GoBurrow 2023. Modbus. Hakupäivä 10.3.2023. GitHub. https://github.com/goburrow/modbus

15. Raspberry Pi 2023. Typical Power Requirements. Hakupäivä 11.13.2023. https://www.raspberrypi.com/documentation/computers/raspberry-pi.html
16. Upton, Eben 2022. Production and supply-chain update. Raspberry Pi News 2022. Hakupäivä 11.3.2023. https://www.raspberrypi.com/news/production-and-supply-chain-update/
17. Okoi, Divine 2021. 20 Best Operating Systems You Can Run on Raspberry Pi in 2021. Fossmint. Hakupäivä 11.3.2023. https://www.fossmint.com/operating-systems-for-raspberry-pi/
18. Raspberry Pi 2023. Raspberry Pi Zero W. Hakupäivä 11.3.2023. https://www.raspberrypi.com/products/raspberry-pi-zero-w/
19. Strecansky, Bob 2020. Hands-On High Performance with Go. Birmingham: Packt Publishing Ltd. Hakupäivä 11.3.2023. Google Books. Vaatii käyttöoikeuden.
20. Anagnostopoulos, Achilleas 2020. Hands-On Software Engineering with Golang. Packt Publishing Ltd. Hakupäivä 11.3.2023. O'Reilly. Vaatii käyttöoikeuden.
21. Zihatec GmbH 2022. Datasheet RS422/RS485 Shield Rev D. Hakupäivä 16.3.2023. https://www.hwhardsoft.de/app/download/12227208397/Datasheet+RS485+HAT+Rev+D.pdf?t=1662369539
22. Ensto Enervent Oy 2015. Enervent Pingvin Kotilämpö eAir asennusohje. Hakupäivä 16.3.2023. https://doc.enervent.com/op/op.ViewOnline.php?documentid=2980&version=1
23. Salmi, Miikka 2023. Geneerisyyttä Golangiin. Buutti Software 2023. https://buutticonsulting.com/blogi/2020/06/11/geneerisyytta-golangiin-osa-1/
24. Google Ltd. 2023. Effective Go. Hakupäivä 16.3.2023. https://go.dev/doc/effective_go
25. Google 2023. Go Packages. Encoding/json. Hakupäivä 20.3.2023. https://pkg.go.dev/encoding/json
26. Target, Sinclair 2017. The Rise and Rise of JSON. Hakupäivä 24.3.2023. https://twobithistory.org/2017/09/21/the-rise-and-rise-of-json.html
27. Enervent Zehnder Oy 2023. Kotilämpö Ohjaus. Hakupäivä 29.3.2023. https://www.enervent.fi/kotilampo-ohjaus/
28. Prometheus.io 2023. Comparison to Alternatives. Hakupäivä 29.3.2023. https://prometheus.io/docs/introduction/comparison
29. Kingston Technology 2023. Industrial microSD Memory Card. Hakupäivä 29.3.2023. https://www.kingston.com/en/memory-cards/industrial-grade-microsd-uhs-i-u3
30. Prometheus.io 2023. Writing Exporters. Hakupäivä 29.3.2023. https://prometheus.io/docs/instrumenting/writing_exporters/

31. Stenvall, Sam 2023. Eda-modbus-bridge. Hakupäivä 30.3.2023. GitHub. https://github.com/Jalle19/eda-modbus-bridge
32. Ouman Oy 2023. EH-203 Lämmönsäädin. Käsikirja. Hakupäivä 2.4.2023. http://ouman.fi/documentbank/EH-203__manual__fi.pdf
33. Gilbert, Henri, Handschuh, Helena 2004. Security Analysis of SHA-256 and Sisters. Lecture Notes in Computer Science, vol 3006. Berlin: Springer. Hakupäivä 30.3.2023. https://doi.org/10.1007/978-3-540-24654-1_13
34. Edwards, Alex 2021. How to correctly use Basic Authentication in Go. Hakupäivä 30.3.2023. https://www.alexedwards.net/blog/basic-authentication-in-go
35. Rankinen, Jarno 2023. Enervent-ctrl. Hakupäivä 30.3.2023. GitHub. https://github.com/0ranki/enervent-ctrl

PINGVIN-TIETORAKENNE LIITE 1




Max lämmitys - painikkeen tila vaihdettu
Pyydetty tila
Pois
Päälle
Max lämmitys päällä
Ei
Ei toimintoa
Kyllä
Max lämmitys päällä
Ei
REST-pyyntö max_heat_off
Kyllä
Huonelämpötila yli tavoitteen
Ei
Max lämmitys päällä
REST-pyyntö max_heat_on
Kyllä
Ei
Max lämmitys - painikeilmaisin päälle
Max lämmitys - painikeilmaisin pois päältä
Lopetus