Tampereen yliopisto

Rasmus Tuokko

LÄMPÖTILAN VALVONTAJÄRJESTELMÄ KÄYTTÄEN IOT-LAITTEITA

Kandidaatintyö
Tekniikan ja luonnontieteiden tiedekunta
Tarkastaja: Mikko Salmenperä
Toukokuu 2024

TIIVISTELMÄ

Rasmus Tuokko: Lämpötilan valvontajärjestelmä käyttäen IoT-laitteita
Kandidaatintyö
Tampereen yliopisto
Teknisten tieteiden kandidaatin tutkinto-ohjelma
Toukokuu 2024

Automaatiojärjestelmien yksi keskeinen vaatimus on epäherkkyys parametrimuutoksille. Pidemmällä aikavälillä järjestelmään kohdistuu käytöstä ja olosuhteista aiheutuvia rasitteita, kuten kulumista, ikääntymistä, likaantumista ja lämpenemistä. Nämä rasitteet eivät saisi aiheuttaa järjestelmään merkittävää haittaa, joten niiden vaikutuksia täytyy ennaltaehkäistä. Tässä kandidaatintyössä keskitytään lämpötilan mittaamiseen, jotta tarkasteltavaan järjestelmään ei aiheutuisi vahinkoa mahdollisen ylikuumenemisen seurauksena. Ylikuumeneminen aiheuttaa yksittäiseen laitteeseen useita haitallisia vaikutuksia, kuten laitteen eliniän lyhenemistä, suorituskyvyn heikkenemistä ja luotettavuuden menettämistä. Työn tarkoituksena on toteuttaa lämpötilan valvontajärjestelmä IoT-laitteiden (engl. Internet of Things) avulla Tampereen yliopiston Hervannan kampuksen Sähkötalon palvelinhuoneeseen, jotta tilan palvelinkaapin laitteet eivät kärsisi ylikuumenemisen seurauksista.

Työn taustatutkimusluvussa perehdytään projektin kannalta tärkeimpiin käsitteisiin, jotta projektin käyttämää tekniikkaa ymmärrettäisiin sekä osattaisiin soveltaa tehokkaasti. Luvussa esitetään teoriaa IoT:stä, IIoT:stä (engl. Industrial Internet of Things), erilaisista tiedonsiirtoprotokollista ja ohjelmointirajapinnoista. IoT sekä IIoT ovat tärkeitä käsitteitä, koska projektissa käytetään IoT-laitteita. Kokonaisuudessaan projekti kuitenkin kuuluu enemmän IIoT:n sovellusdomainiin sen tietoturvavaatimusten vuoksi. On myös tärkeää esitellä ja vertailla tiedonsiirtoprotokollia, jotta projektiin osataan valita tehokkaimmat vaihtoehdot. API (engl. Application Programming Interface) eli ohjelmointirajapinta on projektin kannalta keskeinen käsite, koska ohjelmointirajapintojen käyttö tekee projektin toteuttamisesta helpompaa ja tehokkaampaa.

Projekti toteutettiin asentamalla mittausanturit sekä reititin palvelinhuoneeseen ja ohjelmoimalla dataa jalostava sovellus. Laitteisto ja sovellus konfiguroitiin toimimaan optimaalisesti yhdessä. Reitittimen tarkoituksena on vastaanottaa mittausantureiden lähettämää dataa ja välittää se eteenpäin sovellukselle. Ensiksi sovellus tallentaa lämpötiladatan CSV-tiedostomuodossa (engl. Comma-Separated Values). Toiseksi sovellus lähettää mittausdatan paikallisesti toimivalle verkkopalvelimelle, joka esittää datan kuvaajana. Viimeiseksi sovellus lähettää tarvittaessa hälytysviestin Mattermost-viestipalveluun, jos lämpötilan mittaus ylittää raja-arvon.

Avainsanat: IoT, IIoT, tiedonsiirtoprotokolla, API

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin Originality Check -ohjelmalla.

SISÄLLYSLUETTELO

1. JOHDANTO 1
2. TAUSTATUTKIMUS 2
2.1 IoT:n määritelmä ja arkkitehtuuri 2
2.2 IIoT:n määritelmä ja IIoT-laitteita kuvaavat kuusi kategoriaa 4
2.3 IoT:n ja IIoT:n keskeisimmät erot 6
2.4 Tiedonsiirtoprotokollat ja protokollapinomallit 7
2.5 Ohjelmointirajapinnat 11
3. PROJEKTIN TOTEUTUS 12
3.1 Projektin laitteisto 13
3.2 Sovelluksen toteutus, MQTT-ohjelmistojen valinta ja laitteiston konfigurointi 14
4. TULOKSET 17
4.1 Mittausdatan kerääminen ja tallentaminen tietorakenteisiin 17
4.2 Lämpötiladatan varastointi 18
4.3 Lämpötiladatan graafinen esitys 19
4.4 Hälytysviestit 20
4.5 Ohjelman pääfunktio 20
5. YHTEENVETO 22
LÄHTEET 24
LIITE A: MITTAUSDATAN KERÄÄMISEN JA TIETORAKENTEISIIN TALLENTAMISEN OHJELMAKOODI 27
LIITE B: MITTAUSDATAN TIEDOSTOON KIRJOITTAMISEN JA TILATIEDOSTON TOIMINNAN OHJELMAKOODI 29
LIITE C: VERKKOPALVELIMEN KUVAAJAN PÄIVITTÄMISEN OHJELMAKOODI 32
LIITE D: HÄLYTYSVIESTIN LÄHETTÄMISEN OHJELMAKOODI 33
LIITE E: PÄÄFUNKTION OHJELMAKOODI 34

LYHENTEET JA MERKINNÄT

IoT	engl. Internet of Things, esineiden internet
IIoT	engl. Industrial Internet of Things, teollinen esineiden internet
KILL-ilmiö	Kuluminen, Ikääntyminen, Likaantuminen, Lämpeneminen
MQTT	engl. Message Queuing Telemetry Transport, julkaisija/tilaajaperustainen viestintäprotokolla
HTTP-protokolla	engl. HyperText Transfer Protocol, tilaton tiedonsiirtoprotokolla
API	engl. Application Programming Interface, ohjelmointirajapinta
MAC-osoite	engl. Media Access Control, verkkosovittimen yksilöimä numerosarja
IP-protokolla	engl. Internet Protocol, pakettivälitteisen tiedonsiirron yhteyskäytäntö, joka määrittelee säännöt tietoliikennepakettien reitityksellä ja liikkumiselle tietoverkossa ja tietoverkkojen välillä
TCP-protokolla	engl. Transmission Control Protocol, kuljetusyhteyskäytäntö, joka huolehtii tiedonsiirrosta varmistaen tiedon perillemenon lähettämällä tiedon tarvittaessa uudelleen
OSI-malli	engl. Open Systems Interconnection, malli, joka kuvaa tiedonsiirtoprotokollien yhdistelmän seitsemässä kerroksessa
UDP-protokolla	engl. User Datagram Protocol, kuljetuskäytäntö, joka huolehtii tiedonsiirrosta varmistamatta tiedon perillemenoa mahdollistaen näin suuremman nopeuden
HTTPS-protokolla	engl. HyperText Transfer Protocol Secure, HTTP-protokollan ja TLS/SSL-protokollan yhdistelmä, jota käytetään tiedon suojattuun siirtoon tietoverkossa
REST	engl. Representational State Transfer, arkkitehtuurinen tyyli, joka määrittelee joukon rajoitteita, joita käytetään verkkopalvelujen luomiseen
TLS-protokolla	engl. Transport Layer Security, salausprotokolla
HTML	engl. HyperText Markup Language, hypertekstin ja rakenteen ulkoasun kuvaukseen käytettävä merkintäkieli
XML	engl. Extensible Markup Language, helposti laajennettavissa oleva rajattu merkintäkieli
JSON	engl. JavaScript Object Notation, yksinkertainen ja kevyt tiedostomuoto tiedonvälitykseen ja tallennukseen
CSV	engl. Comma-Separated Values, tiedostomuoto, jota käytetään taulukkomuotoisen datan tallentamiseen tekstimuodossa

1. JOHDANTO

Teknisten laitteiden määrä maailmassa kasvaa eksponentiaalisesti ja yhä useampi laite on yhteydessä internetiin. Laitteiden määrän maailmassa arvioidaan olevan yli 207 miljardia vuoden 2024 loppuun mennessä [1]. Internetiin yhdistettyjen laitteiden kokonaisuutta kutsutaan esineiden internetiksi (engl. IoT, Internet of Things). Laite voi olla esimerkiksi mittausanturi, kahvinkeitin tai auto. Yksityiskäytössä kahvinkeittimen päälle napsauttaminen puhelimen näytöltä kuulostaa yksinkertaiselta ja harmittomalta, kun taas teollisessa kontekstissa antureiden mittausdata saattaa olla kriittinen prosessin toiminnalle. Teollisuudella on oma versionsa esineiden internetistä: teollinen esineiden internet (engl. IIoT, Industrial Internet of Things). Tämä työ sisältyy IIoT-järjestelmien sovellusdomainiin sen tietoturva- sekä luotettavuusvaatimusten vuoksi.

Automaatiojärjestelmät altistuvat elinkaariensa aikana erilaisille rasitteille, kuten kulumiselle tai lämpenemiselle. Systeemiä suunniteltaessa tulisi ottaa huomioon nämä rasitteet, jotta säästyttäisiin mahdollisilta haittavaikutuksilta, kuten ylikuumenemiselta. Ylikuumeneminen aiheuttaa järjestelmän komponentteihin vaurioita ja se voi aiheutua järjestelmän sisäisistä tai ulkoisista vaikutteista [2]. Näin ollen järjestelmässä tulee ottaa huomioon itse laadukkaiden komponenttien lisäksi niiden ympäristö. Helppo tapa ennaltaehkäistä ympäristöstä aiheutuvaa järjestelmän ylikuumenemista on mitata ympäristön lämpötilaa.

Tässä työssä keskitytään lämpötilan mittaamiseen ja mittausdatan hyödyntämiseen, jotta lämpötilavaihteluiden negatiivisilta vaikutuksilta järjestelmään säästyttäisiin. Lämpötila halutaan pitää järjestelmälle optimaalisella välillä ja mittausten lämpötila-arvojen poiketessa tältä väliltä lähetetään asianmukaisille henkilöille hälytysviesti poikkeavasta lämpötila-arvosta. Henkilöt voivat hälytysviestin saatuaan tehdä toimenpiteitä, kuten avata ikkunan huoneesta, jotta lämpötila saadaan takaisin tavoitevälille.

Tämän kandidaatintyön luvussa 2 tehdään taustatutkimusta projektin kannalta tärkeimmistä ilmiöistä ja käsitteistä. Luvussa 3 kerrotaan projektin tavoitteet, esitetään käytettävä laitteisto ja selitetään sovelluksen toteuttamisesta sekä laitteiston konfiguroinnista. Luku 4 sisältää projektin tulosten esittelyn. Lopuksi luku 5 sisältää tämän kandidaatintyön yhteenvedon.

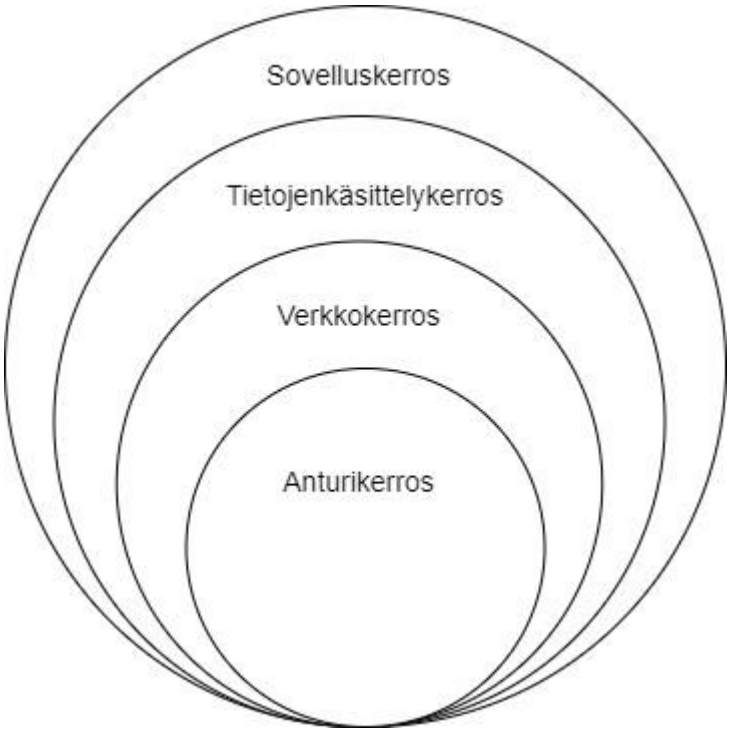
2. TAUSTATUTKIMUS

Tämä luku sisältää teoriaa projektin kannalta tärkeimmistä käsitteistä, kuten IoT:stä, IIoT:stä, tiedonsiirrosta MQTT- (engl. Message Queuing Telemetry Transport) sekä HTTP-protokollalla (engl. HyperText Transfer Protocol) ja lopuksi ohjelmointirajapinnoista (engl. API, Application Programming Interface). Projektin toteuttamisen kannalta on tärkeää ymmärtää siinä käytettävää tekniikkaa, koska tekniikan ymmärtäminen mahdollistaa tehokkaan järjestelmän toteutuksen. Esimerkiksi ohjelmointirajapintojen ymmärtäminen ja hyödyntäminen mahdollistaa sen, että jokaista järjestelmän osaa ei tarvitse itse toteuttaa. Ajoittain on mahdollista käyttää ohjelmointirajapintoja, joilla saa helposti käyttöön jo valmiiksi toteutettuja tehokkaita toimintoja.

2.1 IoT:n määritelmä ja arkkitehtuuri

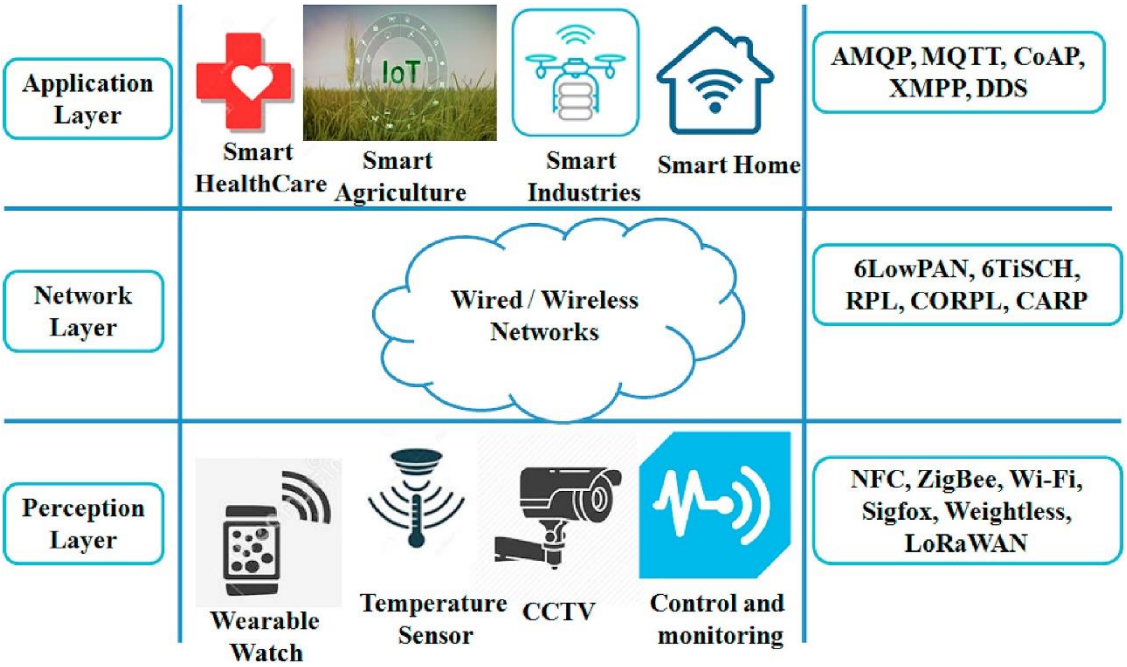
Esineiden internet tarkoittaa nimensä mukaisesti laitteita, jotka ovat yhteydessä internetiin. IoT on laitteiden muodostama ekosysteemi, joka koostuu miljoonista verkoista, jotka voivat olla yksityisiä, julkisia, akateemisia, yritysten tai hallitusten omistamia. Nämä laitteiden muodostamat verkostot skaalautuvat pienistä lokaaleista verkostoista suuriin maailmanlaajuisiin verkostoihin. Laitteet ja niiden muodostamat verkostot ovat yhteydessä muhin laitteisiin ja verkostoihin laajalla valikoimalla langattomia, langallisia, elektronisia ja optisia verkkotekniikoita. [3] Esineiden internet on siis hankala selittää täysin yksikäsitteisesti, koska se on suuri kokonaisuus.

IoT:n rakenne voidaan jakaa neljään osaan, jotka ovat anturikerros, verkkokerros, tietojenkäsittelykerros ja sovelluskerros. Kerrokset on havainnollistettu kuvassa 1. Anturikerros koostuu antureiden verkostosta, jossa jokaisella anturilla on oma yksilöivä tunnus. Verkkokerros on vastuussa antureiden keräämän mittausdatan kuljettamisessa seuraavalle kerrokselle, ja verkkokerroksella tulisi olla korkea suorituskyky, jotta mittausdataa voidaan kuljettaa tehokkaasti kerrokselta toiselle. Tietojenkäsittelykerros toimii verkkokerroksen ja sovelluskerroksen välillä molempiin suuntiin. Suuren datamäärän vastaanottaminen sekä tietojen turvallisuuden ja yksityisyyden varmistaminen ovat tietojenkäsittelykerroksen tehtäviä. Ylimpänä kerroksena IoT:ssä on sovelluskerros, joka tarjoaa sovelluksen käyttäjälle käyttöliittymän. [4] Yleisesti esineiden internetin rakenne mielletään kerroksittain, jotta kompleksista ilmiötä olisi helpompi ymmärtää ja hahmottaa.



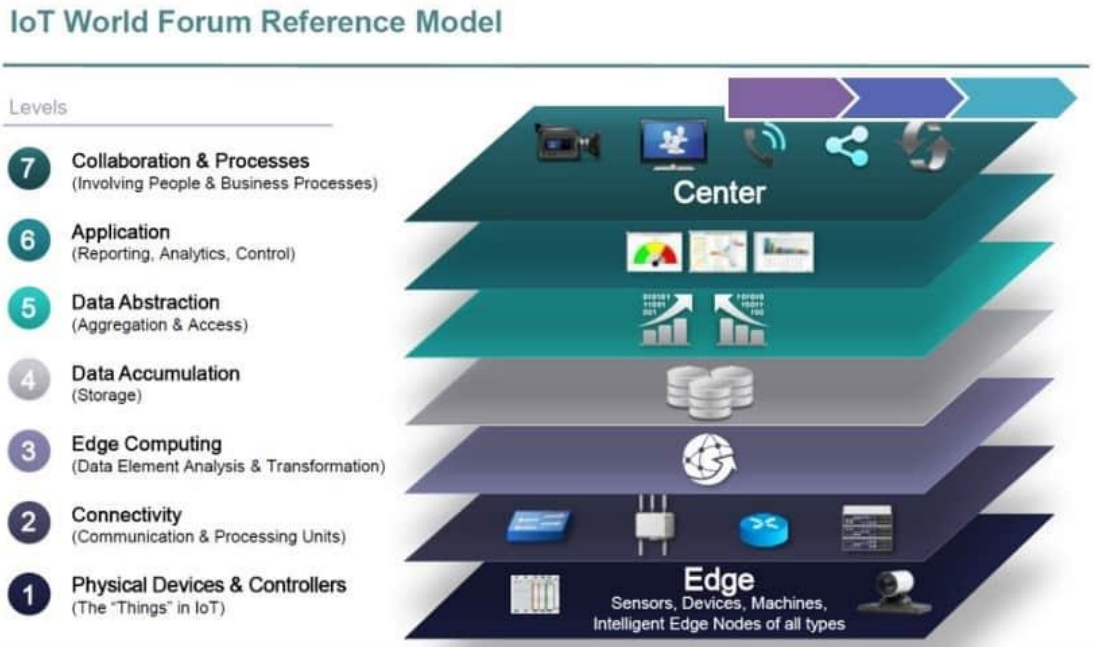
Kuva 1. *IoT:n nelikerroksinen rakenne, muokattu lähteestä [4].*

Kuvan 1 nelikerroksinen IoT:n rakenne on kuitenkin vain yksi tulkinta esineiden internetin arkkitehtuurista. Rakenteessa verkkokerros ja tietojenkäsittelykerros voidaan mieltää yhdeksi kerrokseksi tai ne voidaan jakaa useampaan osaan. Kolmikerroksisessa määrittelyssä IoT:n rakenne koostuu havaitsevasta kerroksesta, verkkokerroksesta ja sovelluskerroksesta [5].



Kuva 2. *IoT:n kolmikerroksinen rakenne [5].*

Kuvan 2 IoT:n arkkitehtuuri koostuu samoista elementeistä kuin nelikerroksisessa määrittelyssä, mutta neljästä kerroksesta kaksi keskimmäistä on yhdistynyt yhdeksi eli verkkokerrokseksi. Rakenne voidaan myös määrittää laajemmalla tulkinnalla, kuten seitsenkerroksisella rakenteella, joka koostuu fyysisistä laitteista, yhteyksistä, reunalaskennasta, tietojen keräämisestä, datan abstraktoinnista, sovelluksesta sekä yhteistyöstä ja prosesseista [6].



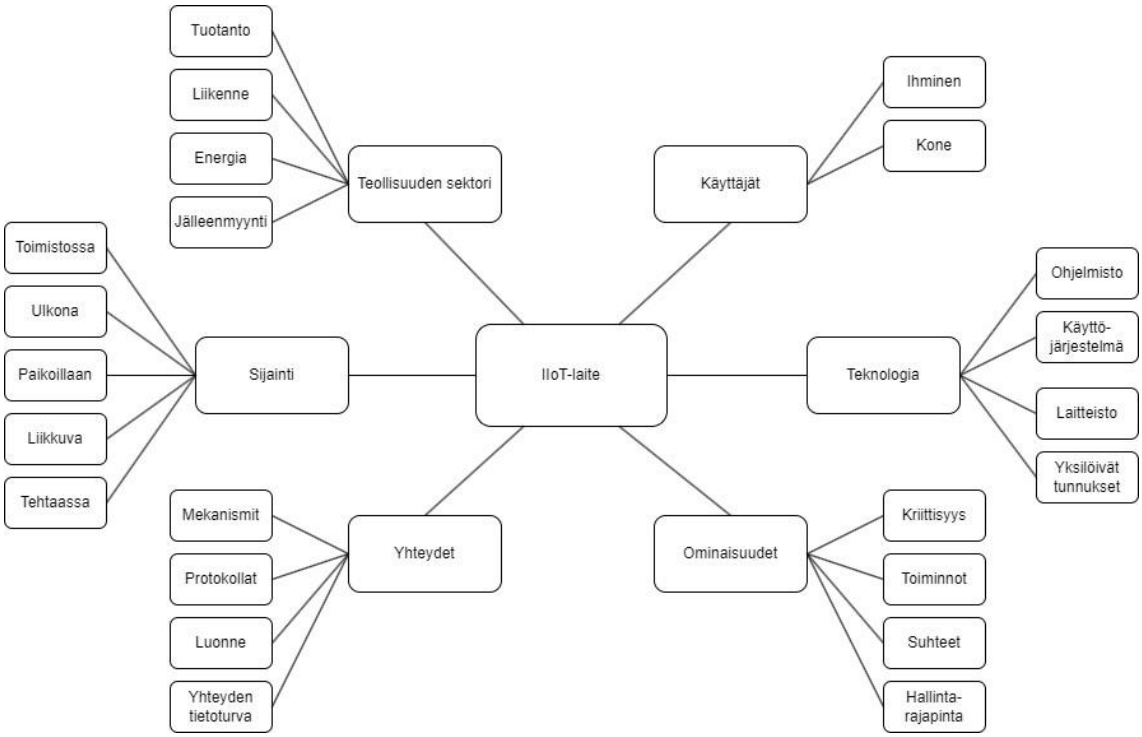
Kuva 3. *IoT:n rakenteen seitsenkerroksinen malli [6].*

Kuvassa 3 esitetään huomattavasti yksityiskohtaisempi malli IoT:n rakenteesta. IoT:n arkkitehtuurista ei siis ole yksiselitteistä rakennetta vaan monimutkaista ilmiötä ymmärretään ja havainnollistetaan erilaisilla malleilla.

2.2 IIoT:n määritelmä ja IIoT-laitteita kuvaavat kuusi kategoriaa

Teollinen esineiden internet liittyy hyvin vahvasti esineiden internetin käsitteeseen, mutta erilaisessa kontekstissa ja erilaisilla vaatimuksilla. IIoT tarkoittaa järjestelmäkokonaisuutta, joka koostuu älykkäistä laitteista ja tiedon automaattisesta siirrosta sekä käsittelystä. Tiedon käsittelyyn ja siirtämiseen käytetyt tekniikat voivat olla mahdollisia pilvi- tai reunalaskenta-alustoja, jotka mahdollistavat reaaliaikaisen ja älykkään kommunikaation sekä tiedon keräämisen, analysoinnin ja hyödyntämisen teollisessa ympäristössä. [7] IIoT:n määritelmä on siis lähellä IoT:n määritelmää, mutta IIoT:n laitteilla on erilaisia vaatimuksia ja erilainen ympäristö, jossa käsitettä sovelletaan.

IIoT:n sovellusalueeseen liittyviä laitteita voidaan kuvailla kuudella kategorialla, jotka ovat teollisuuden sektori, laitteen sijainti, yhteydet, laitteen ominaisuudet, laitteen tekniikka ja käyttäjätyyppi [7]. Laitteet ovat siis itsessään kokonaisuuksia, jotka liittyvät suurempaan verkostoon internetin välityksellä. IIoT-laitteiden attribuutteja voidaan kategorisoida, jotta monimutkaisia laitteita olisi helpompi ymmärtää. Kategoriat on esitettynä graafisesti kuvassa 4.



Kuva 4. *IIoT-laitetta kuvaavat kuusi kategoriaa, muokattu lähteestä [7].*

Teollisuuden sektorit kuten teollinen tuotanto, liikenne, energiantuotanto tai jälleenmyynti vaativat jokainen erilaisia toimintoja ja vaatimuksia IIoT-laitteilta. Jokainen teollisuuden ala tarvitsee automaatio- ja ohjausjärjestelmiä yhä enemmän. [7] Esimerkiksi jälleenmyynnin varastoissa automaatiojärjestelmien kokonaisuudet ovat hyvin erilaisia kuin ydinvoimalaitoksessa.

Sijainti vaikuttaa IIoT-laitteen altistumista riskeille sekä kyber- että fyysisen turvallisuuden kannalta [7]. Jos laite sijaitsee ulkona, siihen vaikuttavat luonnonvoimat kuten sade tai tuuli sekä siihen saattaa kohdistua ulkoisen fyysisen hyökkäyksen riski. Toisaalta rakennuksessa lukittujen ovien takana sijaitseva laitteeseen ei kohdistu näitä uhkia. Kyberhyökkäyksen riski on myös huomattavasti suurempi laitteella, joka on kaikille saavutettavassa tilassa. Toisaalta jos laite olisi pelkästään asianmukaisen henkilökunnan saavutettavissa, hyökkäyksen riski pienenisi. IIoT-laitteen sijainti vaikuttaa siis hyvin paljon siihen, millaisille uhille laite altistuu.

Identifioimalla IIoT-laitteen yhteyksiä pyritään tunnistamaan sen ja suuremman IIoT-järjestelmän välisen verkko- ja tietoliikenneyhteyden tärkeimmät ominaisuudet. Tärkeimpiin ominaisuuksiin kuuluvat fyysiset viestinnän mekanismit, yhteyksien tietoturva, viestinnän protokollat ja laitteen luonne eli onko se esimerkiksi reaaliaikainen [7]. IIoT-laitteen yhteyksien ominaispiirteiden tutkimisella voidaan siis tunnistaa merkittäviä tietoturvariskejä suuremmassa IIoT-järjestelmän kokonaisuudessa.

IIoT-laitteen ominaisuuksia ovat sen toiminnot, suhteet, hallintarajapinnat ja kriittisyys järjestelmälle. Järjestelmäkriittisyys tarkoittaa sitä, kuinka tärkeä laite on kokonaisprosessille ja kuinka helppoa laitteen korjaaminen tai vaihtaminen mahdollisesti olisi. Mitä kriittisempi laite on järjestelmälle, sitä suuremmat turvallisuusriskit syntyvät laitteeseen kohdistuvista mahdollisista uhista. [7] IIoT-järjestelmästä voidaan identifioida laitteen ominaisuuksia, jotka edesauttavat koko järjestelmän turvallisuuden analysointia.

Tekniikan tunnistamisella voidaan vaikuttaa IIoT-laitteen suunnitteluun ja toimintaan turvallisuuden näkökulmasta. Laitteen tekniikkaa ovat esimerkiksi sen käyttämä ohjelmisto, käyttöjärjestelmä, laitteisto sekä yksilöivät tunnukset, kuten sarjanumero tai MAC-osoite (engl. Media Access Control) [7]. Varsinkin ohjelmiston turvallisuutta ja luotettavuutta voidaan analysoida erilaisilla keinoilla. Ulkopuoliset uhat voivat murtautua järjestelmiin hyödyntämällä ohjelmistovirheitä, kuten puskurin ylivuotovirhettä tai virhetilanteiden epäjohdonmukaista käsittelyä [8]. IIoT-laitteen tekniikan osaavalla suunnittelulla ja toteutuksella on mahdollista ehkäistä tietoturvariskejä huomattavasti.

Viimeinen IIoT-laitetta kuvaava kategoria on käyttäjä eli ihminen tai kone. Laitteella saattaa olla käyttöliittymä, jolloin ihminen pystyy ohjaamaan sitä halutulla tavalla. Toisaalta laitetta voi olla mahdollista käyttää vain toisen IIoT-laitteen kautta. Joissakin tapauksissa laitteen tilasta tai toiminnasta ei saada tietoa. [7] Käyttäjän ja IIoT-laitteen vuorovaikutuksesta voidaan siis tunnistaa laitteelle käyttäjäominaisuudet.

2.3 IoT:n ja IIoT:n keskeisimmät erot

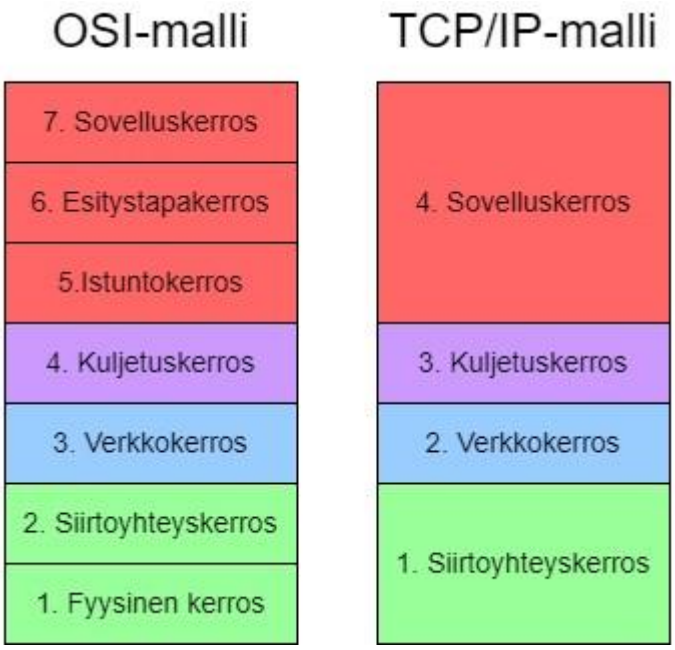
IoT ja IIoT molemmat toimivat internetiin yhdistettyjen laitteiden avulla, mutta IIoT:n tapauksessa luotettavuus ja turvallisuus korostuvat. Teollisuudessa väärästä toiminnosta tai väärentyneestä mittausdatasta saattaa tapahtua mittavia vahinkoja, joten IIoT-ratkaisut vaativat kehittyneempiä turvatoimia aina turvallisesta järjestelmäarkkitehtuurista ja salauksesta uhkien havaitsemiseen [9]. Yksityishenkilöiden elämää helpottavat IoT-ratkaisut eivät vaadi saman tason luotettavuutta ja tietoturvaa, koska esimerkiksi olohuoneen lämpötilan mittauksen katkeaminen ei johda fyysiseen vahinkoon tai taloudelliseen menetykseen.

IIoT-järjestelmät ovat kokonaisuudessaan yleensä suurempia kuin IoT-järjestelmät. Teollisen esineiden internetin nimi viittaa laajamittaisiin laitteiden verkostoihin, jotka sisältävät jopa yli kymmeniä tuhansia laitteita [9]. Varsinkin kotikäyttöiset IoT-järjestelmät ovat yleensä pieniä sovelluksia, kun taas IIoT-järjestelmä voi olla suuren tehtaan teknisten laitteiden muodostama kokonaisuus.

2.4 Tiedonsiirtoprotokollat ja protokollapinomallit

Internetissä tiedonsiirto tapahtuu tietyillä säännöillä ja toimintatavoilla eli protokollilla. Onnistuneen tiedonsiirron on toimittava tiettyjen protokollien mukaan. Protokollia on monenlaisia, kuten IP-protokolla (engl. Internet Protocol), jonka vastuulla on tietokoneen reititys tai TCP-protokolla (engl. Transmission Control Protocol), joka vastaa tiedonsiirron luotettavuudesta. [10] Tietoliikenne internetissä perustuukin pääosin TCP/IP-protokollaperheeseen [11].

Internetin toimintaa määrittäessä hyödynnetään usein protokollapinoa, joka tarkoittaa tiedonsiirron kerroksellista rakennetta, jossa kukin kerros on vastuussa tietyistä toiminnoista tiedonsiirron kompleksisesta kokonaisuudesta [12]. Protokollapinolla on erilaisia rakenteita, joista tässä kappaleessa käsitellään OSI-mallia (engl. Open Systems Interconnection) sekä TCP/IP-mallia. Mallit ovat esitettynä kuvassa 5.



Kuva 5. *OSI-malli sekä TCP/IP-malli, muokattu lähteestä [13].*

OSI-malli koostuu seitsemästä kerroksesta, kun taas TCP/IP-mallissa on vain neljä kerrosta. Jälkimmäiseksi mainitun mallin ylin kerros sisältää OSI-mallin kolme ylintä kerrosta. TCP/IP-mallin alin kerros sisältää OSI-mallin kaksi alinta kerrosta. Kuljetus- ja verkkokerros ovat molemmissa malleissa yhtäläiset.

Siirtoyhteyskerros tarjoaa kehystystoiminnon, joka muodostaa kerroksen kehysrakenteen, johon sijoitetaan ylemmältä kerrokselta saatu data kehysrakenteen dataosuuteen. Toinen siirtoyhteyskerroksen tarjoama palvelu on halutun laatutason luotettava tiedonsiirto. [s. 138, 12] OSI-malli sisältää myös alimmaisena fyysisen kerroksen, jonka vastuulla on bittivirran lähettäminen verkon kautta. Fyysinen kerros tarjoaa nimensä mukaisesti fyysisiä laitteita, kuten kaapeleita [14].

Verkkokerroksen päätehtävänä on pakettien reititys lähettäjältä kohteelle. IP-protokolla reitittää tietoliikennepaketteja kohteen IP-osoitteen perusteella oikeaan osoitteeseen. [s. 16, 12] IP-protokolla tarjoaa yhteydettömän sekä niin sanotun ”best-effort” tietoliikennepaketin lähettämisen, eli paketin onnistuneesta saapumisesta ei voida olla varmoja [Luku 5.1, 11]. Kuvan 5 protokollapinomalleissa verkkokerrokset ovat molemmissa samat.

Kuljetuskerroksen tarkoitus on nimensä mukaan kuljettaa tietoa lähettäjän ja vastaanottajan välillä. Kuljetuskerros tarjoaa luotettavan tiedonsiirron TCP-protokollalla. Tiedonsiirto voi olla myös yksinkertaisempaa ja epäluotettavaa UDP-protokollalla (engl. User Datagram Protocol). TCP-protokolla on yhteydellinen ja luotettava, kun taas UDP-protokolla on yhteydetön ja epäluotettava. [s. 59–75, 12] Kumpaakin protokollaa käytetään eri käyttötarkoituksiin. Luotettavaa tiedonsiirtoa vaativaan tiedonsiirtoon käytetään TCP-protokollaa ja lyhytkestoisille sovelluksille, joille tiedonsiirron luotettavuus ei ole elintärkeää, käytetään UDP-protokollaa. OSI- sekä TCP/IP-mallin kuljetuskerrokset ovat samoja kuten verkkokerroksetkin.

Sovelluskerros on vastuussa vuorovaikutuksista käyttöjärjestelmän tai sovelluksen kanssa [14]. Sovelluskerros toteuttaa tietoliikenteen sovelluskohtaiset palvelut omalla protokollallaan. Tyypillisesti sovellus rakentuu asiakkaan (engl. client) sekä palvelimen (engl. server) prosesseista, jotka kommunikoivat keskenään tiedonsiirtoverkon välityksellä. Ensiksi mainittu asiakasprosessi on aktiivinen avaaja eli se aloittaa kommunikoinnin ja jälkimmäisenä mainittu palvelinprosessi odottaa verkosta tulevia palvelupyyntöjä. [s. 87, 12] OSI-malli sisältää sovelluskerroksen lisäksi vielä istunto- ja esitystapakerroksen. Istuntokerros vastaa sovellusten välisen yhteyden muodostamisesta, hallinnasta ja katkaisemisesta. Esitystapakerros taas tarjoaa tietojen muotoilun ja toimittamisen sovelluskerroksen ymmärtämään muotoon. [14] TCP/IP-mallissa sovellus-, esitystapa- ja istuntokerrosten toiminnot ovat sisällytetty yhteen kerrokseen eli sovelluskerrokseen.

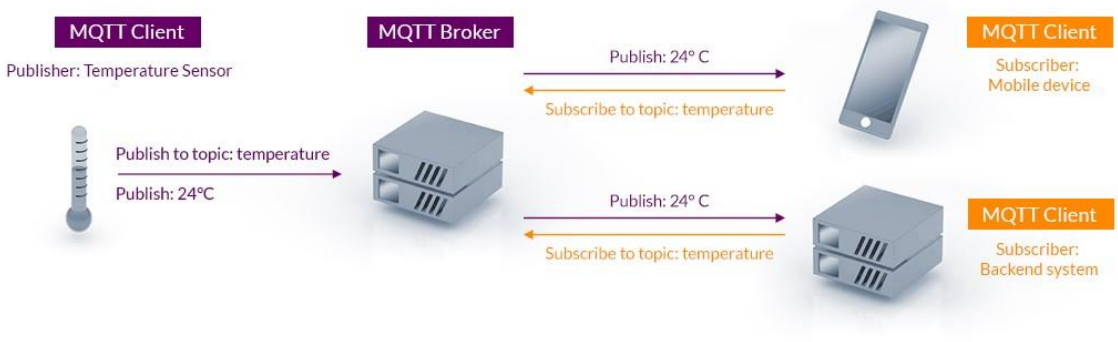
2.4.1 MQTT-protokolla

MQTT on sovelluskerroksen viestintäprotokolla. Koska MQTT on kevyt, avoin ja yksinkertainen toteuttaa, on se ideaalinen valinta rajoitetuille ympäristöille, kuten IoT-järjestelmille. MQTT toimii yleensä kuljetuskerroksen yhteydellisen ja häviöttömän TCP/IP-protokollan päällä. [15]

MQTT toimii asiakas- ja palvelinprosessien väliseen julkaise-tilaa-malliin [15]. Asiakas-palvelin-mallissa asiakas pyytää palveluita palvelimelta. Palvelin taas tarjoaa asiakkaalle pyydettyjä palveluita. [s. 1, 16] Julkaise-tilaa-mallissa järjestelmällä on julkaisija, joka lähettää tietoa sekä tilaaja, joka vastaanottaa kyseistä tietoa. Tilaaja ei halua välttämättä kaikkea tietoa, jota julkaisija lähettää, joten tilaaja voi nimensä mukaisesti tilata vain spesifiä tietoa julkaisijalta. [s. 1–4, 17] Esimerkiksi anturi voi mitata useita suureita, kuten lämpötilaa, ilmanpainetta ja ilmankosteutta. Jos järjestelmässä ollaan kiinnostuneita vain lämpötilan arvoista, voidaan tilaaja konfiguroida tilaamaan vain viestejä, joiden aiheena on lämpötila. Julkaisija ja tilaaja voivat kommunikoida asynkronisesti [18]. Tämä tarkoittaa sitä, että julkaisija ja tilaaja eivät ole ajallisesti toisistaan riippuvaisia.

Tiedon tilaaminen ja julkaiseminen tehdään erillisen välittäjän (engl. broker) kautta, joka toteuttaa saapuvan tiedon suodatuksen sekä toimittamisen siitä kiinnostuneille tilaajille. Julkaise-tilaa-mallisessa tiedonsiirrossa on erilaisia suodatusvaihtoehtoja, joiden perusteella dataa suodatetaan. [18] Erilasia datan suodatusperusteita ovat aihe, tyyppi tai sisältö. Jokaisessa viestissä on aihe. Jos tilaaja on kiinnostunut ja tilannut viestin aiheen, saa tilaaja kyseisen viestin. Tyypin perusteella suodatus on hyvin samankaltainen aiheen perusteella suodatuksen kanssa, jokaisella viestillä on tyyppi ja tilaaja voi tilata tietyn tyyppisiä viestejä. Viestejä voidaan myös suodattaa niiden sisällön perusteella, esimerkiksi jos viestin mittausdata poikkeaa viitearvoista. [s. 148–149, 17] Edellä mainituista suodatustavoista MQTT-välittäjä suodattaa viestejä aiheen perusteella [19].

MQTT-asiakas voi samanaikaisesti tilata ja julkaista viestejä tietyillä aiheilla välittäjälle. Asiakas voi olla esimerkiksi resurssirajoitettu anturi tai graafinen asiakasjärjestelmä, joka voi kommunikoida käyttäen MQTT-protokollaa. Kaikki MQTT-viestit täytyy mennä välittäjän kautta, joka vastaanottaa julkaisijoiden viestit ja välittää ne tilaajille. [18]



Kuva 6. *MQTT-protokollan tiedonsiirto [20].*

Kuvassa 6 havainnollistetaan MQTT-protokollan julkaise-tilaa-mallinen tiedonsiirto. Mittausanturi voi julkaista mitattua lämpötiladataa lämpötila-aiheella. Jos tilaaja eli MQTT-asiakas on yhdistänyt MQTT-välittäjään ja tilannut kyseisen aiheen, lähettää välittäjä datan tilaajalle.

2.4.2 HTTP-protokolla

MQTT-protokollan tavoin HTTP-protokolla on sovelluskerroksen TCP-yhteydellä toimiva viestintäprotokolla. Pääosin HTTP-protokollaa käytetään verkkosivujen tiedostojen siirtämiseen, mutta nykyisin sitä voidaan pitää yleiskäyttöisenä protokollana. HTTP-protokolla on MQTT:stä poiketen yhteydetön protokolla, mikä tarkoittaa sitä, että palvelin ei säilytä tietoa aikaisemmista pyynnöistä, joten jokainen pyyntö palvelimelle on täysin itsenäinen. Tämän takia HTTP-palvelimen toiminta on tehokasta ja yksinkertaista. [s.96–97, 12]

HTTP-protokolla toimii asiakas-palvelin-periaatteella. Palvelin kuuntelee tiettyä porttinumeroa, johon asiakas voi lähettää pyynnön, johon taas palvelin vastaa. HTTP-asiakkaan palvelupyyntö voi olla muun muassa GET-, PUT-, DELETE- ja POST-metodi. Esimerkiksi selain voi lähettää palvelimelle GET-pyynnön, jolloin palvelin lähettää selaimelle asiakkaan pyytämän verkkosivun. HTTP-protokollaa käytettäessä IoT:n kontekstissa POST-pyyntö on usein hyödynnetyin metodi. Metodilla laite voi lähettää palvelimelle tietoa, kuten mittausdataa tai laitteen tilan. [21]

HTTPS (engl. HyperText Transfer Protocol Secure) on nimensä mukaisesti HTTP-protokollan suojattu versio. HTTPS käyttää tiedon suojattuun siirtoon suojausprotokollaa, kuten TLS-protokollaa (engl. Transport Layer Security). HTTPS-protokolla on siis HTTP-protokollan sekä suojausprotokollan yhdistelmä. Nykypäivänä suurin osa verkkosivuista käyttää HTTPS-protokollaa HTTP-protokollan sijaan. [22]

2.5 Ohjelmointirajapinnat

Ohjelmointirajapinta eli API mahdollistaa projektin käyttämien funktioiden ja toimintojen toteuttamisen ilman, että niitä tarvitsee ohjelmoida uudelleen. API on joukko funktioita ja toimintoja, jotka mahdollistavat sen, että muut ohjelmistot voivat hyödyntää tiettyjä toiminnallisuuksia. [23]

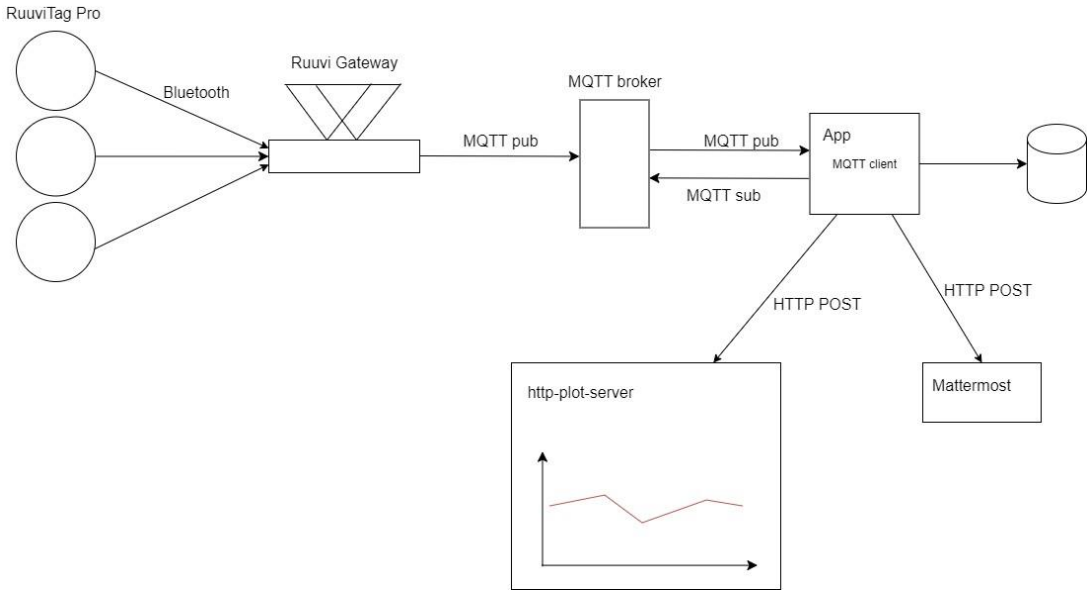
Oman ohjelmiston kehittämisessä on hyödyllistä käyttää ohjelmointirajapintoja. Esimerkiksi suunnitellessa verkkosovellusta, jonka tarvitsee tehdä HTTP-pyyntöjä ei ole välttämättä järkevää kehittää koko koodia pyynnön tekemiseen. Sen sijaan järkevä valinta olisi hyödyntää API:a, joka huolehtii pyynnön tekemisessä. [23]

REST API (engl. Representational State Transfer Application Programming Interface) on palvelu, joka tarjoaa ohjelmoijalle toimintoja, joilla on mahdollista hyödyntää turvallisesti verkkopalvelua, joka ei ole omassa sovelluksessa. HTTP-standardi tukee täysin REST API -kirjastoa. [23] REST API:n toimintaa voidaan esittää yksinkertaisella esimerkillä, jossa asiakas lähettää palvelimelle HTTP GET-, POST-, PUT- tai DELETE-pyynnön, johon palvelin vastaa resurssin muodossa. Resurssi voi olla mikä tahansa, kuten HTML-tiedosto (engl. HyperText Markup Language), XML-tiedosto (engl. Extensible Markup Language) tai JSON-objekti (engl. JavaScript Object Notation). Tällä hetkellä JSON on suosituin verkkopalveluissa käytetty resurssimuoto. [24] JSON on kevyt tiedonsiirtomuoto, jota useat järjestelmät ovat sopineet käyttävänsä tiedon välittämiseen [25].

3. PROJEKTIN TOTEUTUS

Projektin tavoitteena on luoda palvelinhuoneeseen lämpötilan valvontajärjestelmä internetiin yhdistetyillä IoT-laitteilla. Kyseinen tila sijaitsee Tampereen yliopiston Hervannan kampuksen Sähkötalossa. Palvelinhuoneessa lämpötilaa tarvitsee mitata vähintään kahdesta kohtaa, koska siellä on palvelinkaappi, jonka laitteet työntävät lämmintä poistoilmaa taakseen. Siispä yksi mittaus olisi huoneen yleisen ilmastoinnin aiheuttamasta lämpötilasta ja toinen mittaus olisi palvelinkaapin poistoilman aiheuttamasta lämpimämmästä kohdasta. Yksinkertaisuuden vuoksi lämpötilan mittaukset voidaan luokitella olevan palvelinkaapin edessä tai takana. Työ toteutetaan mittausantureilla, reitittimellä ja ohjelmoimalla järjestelmän toimintalogiikan toteuttava sovellus.

Ohjelmoitavalle sovellukselle on määritetty vaatimukset, jotka sen tulee täyttää. Ensinnäkin sovelluksen tulee ottaa vastaan mittausdataa MQTT-välittäjältä ja varastoida sitä johonkin talteen. Toiseksi mittausdata halutaan verkkosivulle graafisesti esitettynä. Viimeiseksi sovelluksen tulee osata lähettää tarvittaessa hälytysviesti Mattermost-viestipalveluun. Näin asianmukaisten henkilöiden on mahdollista tehdä nopeita toimenpiteitä esimerkiksi mahdollisen ylikuumenemisen seurauksena.



Kuva 7. *Järjestelmä kokonaisuudessaan.*

Kuvasta 7 ilmenee koko järjestelmän toiminta sekä tiedonsiirtoon käytettävät protokollat. Kevyt MQTT-protokolla on ideaalinen tapa välittää mittausdataa reitittimeltä MQTT-välittäjälle ja siitä eteenpäin vielä ohjelmoitavalle sovellukselle. HTTP POST -metodi on taas yksinkertainen tapa lähettää tietoa palvelimen verkko-osoitteeseen.

3.1 Projektin laitteisto

RuuviTag Pro -mittausanturit ovat langattomia avoimen lähdekoodin antureita, jotka mittaavat lämpötilaa, ilmankosteutta, ilmanpainetta ja liikettä. Anturit lähettävät mittausdataa Bluetooth-signaalina. Anturin mittaustarkkuus on ±0,1 °C ja mittausresoluutio on 0,01 °C. Mittausanturi toimii tarkasti laajalla lämpöalueella -20…+50 °C. Anturin kotelo on kestävä ja siinä on kaksi kiinnitysreikää, jotka tekevät anturin kiinnityksestä helppoa ja luotettavaa. [26] Siispä mittausanturi on ominaisuuksiltaan erinomainen ja se soveltuu projektiin erittäin hyvin. Työssä käytettävä mittausanturi on esitettynä kuvassa 8.

Kuva 8. *Projektin käyttämä RuuviTag Pro -mittausanturi.*

Ruuvi Gateway -reititin mahdollistaa Ruuvin mittausantureiden seurannan Bluetooth-kantaman ulkopuolella. Mittausdata reititetään oletusarvoisesti Ruuvi Cloud -pilvipalveluun, mutta reitittimen voi konfiguroida lähettämään dataa omalle palvelimelle. [27] Juuri tämän reitittimen kustomointiominaisuuden takia se valikoitui projektiin.

Kuva 9. *Projektin käyttämä Ruuvi Gateway -reititin.*

Työssä käytettävä reititin on esitettynä kuvassa 9. Projektissa hyödynnetään kevyttä MQTT-tiedonsiirtoprotokollaa, jota Ruuvi Gateway -reititin tukee.

3.2 Sovelluksen toteutus, MQTT-ohjelmistojen valinta ja laitteiston konfigurointi

Sovelluksen toteutukseen valikoitui Python-ohjelmointikieli, josta käytetään versiota 3.11.0. Kyseinen ohjelmointikieli on helposti luettava sekä opeteltava, ja sillä on kehittyneet ja helposti käytettävät ohjelmointikirjastot eri sovellusalueilla. Tällä hetkellä Pythonin moduuleja on saatavilla yli 530 000 projektia [28]. Tämä sovellus hyödyntää useita valmiita ohjelmointikirjastoja, jotta toteutus olisi mahdollisimman helppoa ja tehokasta. Sovelluksen toteutukseen pohdittiin myös muita vaihtoehtoja, kuten virtauspohjaista ohjelmointiympäristö Node-RED:iä. Pythonin helppo integrointi eri ympäristöihin ja muut edellä mainitut syyt johtivat Pythonin valintaan sovelluksen toteutukseen.

Sovellusta lähdettiin toteuttamaan sille asetettujen vaatimusten pohjalta. Sovelluksen täytyy tilata tietty aihe, jotta sovellus saisi MQTT-välittäjän julkaisemaa dataa. MQTT-välittäjäksi projektiin valikoitui Eclipse Mosquitto, joka on kevyt ja avoimen lähdekoodin MQTT-välittäjä [29]. MQTT-välittäjiä on useita vaihtoehtoja, mutta Mosquitto on käyttäjien keskuudessa suosittu ja arvostettu valinta, joten sitä todennäköisesti ylläpidetään ja kehitetään laadukkaasti. Mosquitto on helppo käynnistää ja sulkea komentokehotteessa yksinkertaisilla komennoilla. Sovelluksessa MQTT-protokollaa käytetään Eclipse Paho

Python asiakasprosessilla. Paho tarjoaa asiakasluokan, jolla on yksinkertaista tehdä MQTT julkaisuja tai tilauksia. Luokka sisältää myös useita hyödyllisiä funktioita MQTT-tiedonsiirtoon. [30] Pahon avulla on yksinkertaista tilata tietty aihe, jotta sovellus saisi MQTT-välittäjältä mittausdataa. Kuvasta 7 on helppo nähdä visuaalisesti järjestelmän tiedonsiirron eri vaiheet. Vastaanotettu data puretaan ja dekoodataan siten, että päästään käsiksi anturin MAC-osoitteeseen sekä lämpötilan mittaukseen. Muita antureiden mittaamia suureita ei tässä projektissa hyödynnetä. Datan dekoodaamiseen käytetään ruuvi_decoders kirjastoa, joka on Ruuvin mittausanturien lähettämän raa'an datan dekooderi [31]. Reititin kuuntelee kaikkia lähistöllä olevia Ruuvin mittausantureita, minkä takia antureiden MAC-osoitteet täytyy ottaa talteen. Järjestelmän käyttämien antureiden MAC-osoitteet ovat tallennettuina tekstitiedostoihin, jotta saapuneen datan MAC-osoitteita voidaan verrata järjestelmän käyttämien antureiden MAC-osoitteisiin. Tällä tavoin voidaan tallentaa vain haluttujen mittausanturien lähettämä mittausdata.

Sovellus tallentaa lämpötiladataa CSV-tiedostomuodossa (engl. Comma-Separated Values). Tiedostomuotoa hyödyntääkseen voi tuoda csv-moduulin, jolla on yksinkertaista kirjoittaa ja lukea dataa CSV-tiedostosta [32]. Dataa tallennetaan tiedostoon rivi kerrallaan, kunnes tiedosto saavuttaa halutun rivimäärän. Kun haluttu rivimäärä on saavutettu, luodaan uusi tiedosto, johon dataa kirjoitetaan. Kun tiedostojen lukumäärä saavuttaa halutun maksimimäärän, poistetaan vanhin tallennetuista tiedostoista ja luodaan uusi. Tällä tavoin datan määrä pysyy maltillisena, eikä sitä varastoida yli järjestelmän kantokyvyn. Rivimäärää ja tiedostojen lukumäärää on tarvittaessa helppo kustomoida, koska ne ovat määritelty ohjelmakoodin alussa vakioina. Ohjelma osaa myös pitää kirjaa jo tallennetuista lämpötiladatatiedostoista. Aina kun ohjelma käynnistetään, luetaan tilatiedosto. Tilatiedosto sisältää tiedoston järjestysnumeron, rivinumeron ja listan tallennettujen datatiedostojen nimistä. Jokainen kerta, kun ohjelma suljetaan, tallennetaan tilatiedosto. Jos tilatiedostoa ei ole vielä, luodaan sellainen. Jos tilatiedosto on jo, kirjoitetaan siihen päälle päivitetty tilatieto. Näin toimimalla ohjelma muistaa, mihin tiedostoon on jääty ja miltä rivinumerolta pitää jatkaa datan kirjoittamista. Tallennettujen tiedostojen nimien muistaminen on tärkeää, koska niiden avulla on helppoa poistaa aina vanhin tiedosto tarvittaessa. Jos tilatiedostoa ei ole, se tarkoittaa sitä, että datatiedostoja ei ole vielä tallennettu. Tällöin mittausdatan tallentaminen aloitetaan alusta, eli ensimmäisen tiedoston ensimmäiseltä riviltä.

Kuvaajan piirtämiseen verkkosivulle käytetään http-plot-server-moduulia, joka on kevyt Pythonilla kirjoitettu http-kaaviopalvelin datan visualisointiin paikallisesti [33]. Palvelimesta käytetään versiota 0.1. Palvelin voidaan aloittaa komentokehotteesta ja oletusva-

linnoilla palvelin toimii paikallisesti yleisesti käytetyllä porttinumerolla 8080. Datan lähettäminen palvelimelle toimii HTTP:n POST-metodilla, jossa data lähetetään JSON-muodossa. POST-metodia voidaan käyttää, kun ohjelmakoodissa on tuotu requests-kirjasto, joka on helppokäyttöinen HTTP-kirjasto [34]. Palvelimelle voidaan piirtää useita kaavioita, jotka yksilöidään nimen perusteella. Kaavioon voidaan syöttää POST-metodilla nimi, aikaleima, data ja yksikkö eli tässä tapauksessa Celsius-asteikon tunnus. Oletuksena kaavio tallentaa 100 datapistettä, mutta sitä on mahdollista kustomoida palvelinta käynnistäessä komentokehotteesta.

Hälytysviesti Mattermost-viestipalveluun voidaan toteuttaa myös yksinkertaisena HTTP POST-metodina. POST-metodin parametreihin syötetään Mattermost-kanavan webhook sekä lämpötilan mittauksen arvo. Webhookit ovat käyttäjän määrittämiä HTTP-vastakutsuja. Kun käyttötapauksen tapahtuma tapahtuu, lähdesivusto tekee HTTP-pyynnön webhookille määritettyyn osoitteeseen. [35] Hälytysviesti lähetetään, jos mitattu lämpötila alittaa tai ylittää ennalta määritetyn lämpötilavälin. Palvelinkaapin etupuolella olevilla mittausantureilla on eri viitearvot kuin kaapin takana olevilla antureilla, koska palvelinkaapin laitteet työntävät lämmintä poistoilmaa kaapin taakse. Viitearvoja on tarvittaessa helppo muokata, koska ne ovat määritelty vakioina ohjelmakoodin alussa.

Ruuvi Gateway -reitittimen konfiguroinnissa yhdistetään reititin yliopiston tietoturvalliseen verkkoon langattomasti. Tämän jälkeen reititin voidaan määrittää lähettämään dataa Ruuvin omaan pilvipalveluun tai syöttää joko HTTP-palvelimen tai MQTT-palvelimen osoite. Projektissa reititin määritetään toimimaan omalla tietoturvallisella MQTT-palvelimella käyttäen tiedonsiirrossa luotettavaa TCP-protokollaa. Palvelimen osoitteen lisäksi määritetään vaihtoehtoiset käyttäjätunnukset sekä porttinumero, joka on oletuksena MQTT-protokollalla 1883. MQTT-viestin aiheen voi valita itse, tässä järjestelmässä ”ruuvi/<anturin_MAC_osoite>” on hyvä valinta aiheelle, jotta on mahdollista yksilöidä, miltä anturilta dataa vastaanotetaan. Reititin testaa seuraavaksi yhteyttä palvelimelle. Kun yhteys on onnistuneesti luotu, muodostetaan reitittimelle oma IP-osoite DHCP-protokollalla (engl. Dynamic Host Configuration Protocol). Laite, jolla ei ole IP-osoitetta, voi hankkia DHCP-palvelimelta itselleen IP-osoitteen automaattisesti [s. 31, 12]. Reititin ilmoittaa lopuksi, kun se on toiminnassa.

4. TULOKSET

Projektin tuloksena saatiin toimiva lämpötilan valvontajärjestelmä. RuuviTag Pro -mittausanturit lähettävät dataa Ruuvi Gateway -reitittimelle, joka konfiguroitiin lähettämään dataa kustomoidulle MQTT-palvelimelle. Sovelluksessa voidaan tilata sopiva aihe, jolloin saadaan välittäjän lähettämää mittausdataa käytettäväksi sovellukselle. Mittausdata kerätään talteen tietorakenteisiin, jonka jälkeen sille voidaan tehdä halutut toimenpiteet. Mittausdata varastoidaan talteen tiedostoon, esitetään graafisesti paikallisesti toimivalla verkkopalvelimella ja lähetetään tarvittaessa hälytysviesti Mattermost-viestipalveluun.

Tiedonsiirto Reitittimeltä sovellukselle toimii MQTT-protokollalla, joka on hyvä valinta IoT ja IIoT-sovelluksiin. Sovelluksesta verkkopalvelimelle tiedonsiirto toimii HTTP POST -metodilla. Samoin hälytysviestin lähettäminen Mattermost-viestipalveluun toimii HTTP POST -metodin avulla. Tiedonsiirto koko järjestelmässä on esitettynä graafisesti kuvassa 7.

4.1 Mittausdatan kerääminen ja tallentaminen tietorakenteisiin

Mittausdatan keräämien toteutettiin tilaamalla aihe käyttäen Paho-kirjastoa, jotta MQTT-välittäjän lähettämä data saataisiin talteen. Paho-kirjastosta käytetään versiota 1.6.1 ja Mosquitto MQTT-välittäjästä käytetään versiota 2.0.18. Kerätty data purettiin ja dekoodattiin ruuvi_decoders-kirjaston dekooderilla siten, että datasta saatiin anturin MAC-osoite ja lämpötilan mittaus tallennettua tietorakenteeseen.

Mittausdatan kerääminen tapahtuu `on_connect` sekä `on_message`-funktiossa, jotka hoitavat sovelluksen toimintalogiikan yhdistäessä MQTT-välittäjään sekä vastaanottaessa viestin MQTT-välittäjältä. Molemmat funktiot toimivat Paho-kirjastoa hyödyntäen. Funktiot ovat esitettynä liitteessä A. Kun ohjelma yhdistää MQTT-välittäjään onnistuneesti, tulostaa `on_connect`-funktio viestin onnistuneesta yhdistämisestä ja tilaa aiheen ”ruuvi/#”, jolla voidaan tilata kaikki MQTT-viestit, joiden aiheet alkavat ”ruuvi/” etuliitteellä. Tällä tavoin voidaan tilata kaikki Ruuvin mittausanturit. Ohjelman vastaanottaessa viestin `on_message`-funktio toteuttaa siihen tarvittavat toimenpiteet. Funktion täytyy tarkistaa, sisältääkö viesti datakenttää, koska reititin voi lähettää sovellukselle statusviestin, joka sisältää vain tiedon siitä, onko reititin valmiustilassa. Jos viesti sisältää datakentän, puretaan data siten, että siitä saadaan anturin MAC-osoite sekä lämpötilan mittaus. Data puretaan ruuvi_decoders-kirjaston Df5Decoder-dekooderilla. MAC-osoite

formatoidaan haluttuun muotoon ja tallennetaan tietorakenteeseen lämpötilan mittauksen kanssa.

4.2 Lämpötiladatan varastointi

Lämpötiladatan varastointi toteutettiin kirjoittamalla CSV-tiedostoon. Tiedostoja päätettiin säilyttää kolme kappaletta ja jokaisessa tiedostossa rivimäärä päätettiin olevan 10000 riviä. Näillä valinnoilla datan määrä ei pääse kasvamaan yli järjestelmän kantokyvyn.

Mittausdatan kirjoittaminen tiedostoon toteutettiin `write_to_file`-funktiossa, joka on esitettynä liitteessä B. Heti funktion alussa tarkistetaan, jos tiedostoja on tallennettu jo haluttu lukumäärä. Jos tiedostoja on jo haluttu lukumäärä, poistetaan vanhin datatiedosto ja aloitetaan uuden tiedoston alusta datan kirjoittaminen. Tiedoston poistaminen tapahtuu `delete_file`-funktiossa, joka poistaa vanhimman tallennetun tiedostonimen listasta sekä itse laitteelta. Funktio on esitettynä myös liitteessä B. Funktio tulostaa viestin siitä, onnistuiko tiedoston poistaminen ja palauttaa listan tallennetuista tiedostonimistä. Mahdollisen tiedoston poiston jälkeen `write_to_file`-funktio kirjoittaa uuteen tiedostoon aluksi otsikot. Otsikoiksi tiedostoon kirjoitetaan MAC, lämpötila, onko anturi palvelinkaapin edessä vai takana ja aikaleima mittauksesta. Tämän jälkeen tiedostoon kirjoitetaan lämpötilamittauksesta kyseiset tiedot rivi kerrallaan. Kun lämpötilamittausten tietorakenteista on kirjoitettu kaikki data tiedostoon, tulostetaan viesti tiedostoon kirjoittamisen onnistumisesta ja palautetaan funktiossa päivitetty tiedostojen lukumäärä, rivinumero sekä lista jo tallennetuista datatiedostoista.

Ohjelman sulkeutuessa tarvitsee muistaa tieto siitä, mihin tiedostoon ja mille riville on jääty kirjoittamaan dataa. Tämän lisäksi kaikkien tallennettujen datatiedostojen nimet on hyvä pitää myös muistissa, koska tällä tavoin on helppo poistaa tarvittaessa vanhin datatiedosto. Tämän toimintalogiikan hoitaa liitteessä B esiintyvä `save_state`-funktio, joka tallentaa tilatiedoston. Funktiota kutsutaan pääfunktion `finally`-lohkossa, kun ohjelman toiminta päättyy. Tilatiedoston ensimmäiselle riville kirjoitetaan tiedoston numero, toiselle riville rivinumero ja seuraaville riveille tallennettujen datatiedostojen nimet. Lisäksi funktio tulostaa viestin siitä, onnistuiko tilatiedostoon kirjoittaminen.

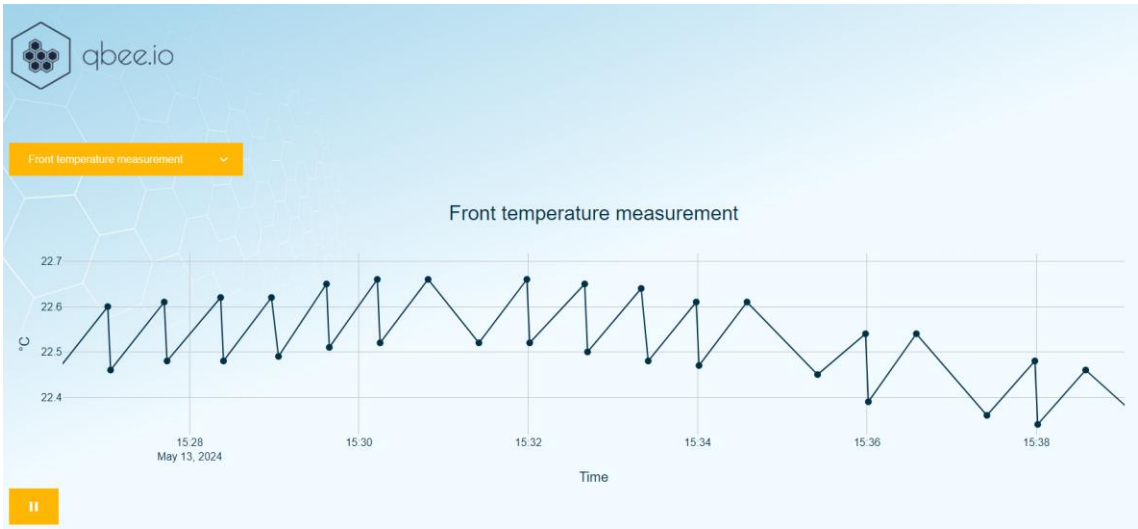
Ohjelman käynnistyessä tarvitsee lukea tilatiedosto, jotta ohjelma osaa jatkaa datan kirjoittamista oikeaan tiedostoon oikealle riville. Tilatiedoston lukeminen tapahtuu `read_state`-funktiossa, joka on esitettynä liitteessä B. Funktio avaa tilatiedoston ja lukee siitä tiedostonumeron, rivinumeron ja datatiedostojen nimet. Lopuksi tulostetaan

viesti onnistuneesta tilatiedoston lukemisesta ja tilatiedot palautetaan funktion paluuarvona. Jos tilatiedostoa ei löydetä järjestelmästä, tulostetaan tästä ilmoitusviesti ja palautetaan tiedostonumeroksi 1, rivinumeroksi 0 ja datatiedostojen nimilistana tyhjä lista.

4.3 Lämpötiladatan graafinen esitys

Lämpötiladatan esittäminen graafisesti toteutettiin http-plot-server 0.1 -moduulilla. Paikallisesti toimivalle verkkopalvelimelle luodaan kaksi kuvaajaa: ”Front temperature measurement” sekä ”Behind temperature measurement”. Palvelinkaapin etupuolella olevat mittausanturit lähettävät datansa ensiksi mainittuun kuvaajaan ja kaapin takana olevat anturit puolestaan lähettävät mittausdatansa jälkimmäiseksi mainittuun kuvaajaan. Verkkopalvelin aloitettiin komentokehotteesta oletusasetuksilla. Oletusasetuksilla palvelin toimii paikallisesti osoitteessa 0.0.0.0 porttinumerolla 8080 ja palvelimen suurin sallittu datapistemäärä on 100. Tämä on hyvä raja datapisteille, koska tällöin kuvaajalle ei tule liian suurta määrää datapisteitä, jolloin kuvaajaa olisi hankalampi lukea.

Verkkopalvelimelle mittausdatan piirtäminen tapahtuu funktiossa, joka on kuvattuna liitteessä C. Funktiossa muodostetaan tallennetuista lämpötilanmittauksista JSON-muotoista dataa, jonka elementtejä ovat kuvaajan nimi, mittausdata, yksikkö ja aikaleima. Yksikkö on luonnollisesti Celsius-asteikon tunnus. Datan lähettäminen verkkopalvelimelle toimii HTTP POST-metodilla, jossa parametreina ovat palvelimen osoite sekä JSON-muotoinen data. Molemmille ohjelmassa oleville lämpötiladatan tietorakenteille toistetaan JSON-muotoisen datan muodostaminen sekä lähettäminen verkkopalvelimelle. Lopuksi funktio tulostaa tiedon siitä, onnistuiko palvelimen kuvaajan päivittäminen.



Kuva 10. *Verkkopalvelimen lämpötilan mittausdatapisteistä piirtämä kuvaaja.*

Kuvassa 10 esitetään kuvankaappaus verkkopalvelimelta, johon lämpötilan mittausdatasta on piirretty kuvaaja. Kyseisen kuvaajan datapisteet ovat palvelinkaapin etupuolella olevien antureiden lämpötilan mittauksia. Kuvaaja muodostetaan kahden mittausanturin samaan aikaan mittaamasta datapisteestä, mistä syystä kuvaaja saa sahalaitamaisen kuvion. Keltaisesta pudotusvalikosta on mahdollista vaihtaa palvelinkaapin takana olevien antureiden mittauksista piirrettyyn kuvaajaan.

4.4 Hälytysviestit

Hälytysviestin lähettäminen Mattermost-viestipalveluun toteutettiin HTTP POST -metodilla, joka saa parametrikseen Mattermost kanavan webhookin. Metodia hyödynnetään requests-kirjaston avulla. Hälytysviesti lähetetään, jos lämpötilamittauksen arvo poikkeaa ennalta määritetyistä lämpötilarajoista. Ihanteellinen lämpötila palvelinhuoneelle on 19–21 °C. Alle 10 °C ja yli 35 °C lämpötilat ovat jo haitallisia laitteille. [36] Siispä palvelinkaapin etupuolella oleville mittausantureille lämpötilan ylärajaksi voidaan asettaa 32 °C ja alarajaksi 13 °C. Palvelinkaapin takapuolella olevat mittausanturit ovat sijoitettuna lämpimämpään kohtaan huoneessa, koska palvelinkaapin laitteet työntävät lämmintä poistoilmaa taakseen. Näille mittausantureille voidaan määrittää lämpötilan ylärajaksi 35 °C ja alarajaksi 17 °C. Näillä valinnoilla voidaan tehdä ajoissa varotoimenpiteet, jotta haitallisiin lämpötiloihin ei päädyttäisi.

Liitteessä D esitetään hälytysviestin lähettämisen ohjelmakoodi. Funktiossa käydään läpi lämpötilan mittaukset ja verrataan niitä ylä- ja alarajoihin. Funktio käsittelee palvelinkaapin etu- ja takapuolella olevat mittausanturit erikseen, koska niillä on eri lämpötilarajat. Jos arvo on liian suuri tai pieni, lähetetään Mattermost-viestipalveluun hälytysviesti, joka sisältää lyhyen kuvaustekstin ja lämpötilan mittauksen arvon. Onnistuneen hälytysviestin lähettämisen jälkeen ohjelma tulostaa tästä vielä varmistusviestin. Jos funktiossa tapahtuu jokin virhe ja hälytysviestin lähettämien ei onnistu, ilmoittaa ohjelma tästä tulosteella. Näin virheen tapahtuessa koko ohjelma ei kaadu, koska se osaa käsitellä virhetilanteet.

4.5 Ohjelman pääfunktio

Ohjelman pääfunktio toimii asynkronisesti hyödyntämällä asyncio-moduulia, joka tarjoaa työkalut asynkronisen ohjelman kirjoittamiseen [37]. Ohjelman on hyvä kommunikoida MQTT-välittäjän kanssa asynkronisesti, koska tällöin ohjelma ja välittäjä eivät ole ajallisesti toisistaan riippuvia. Ohjelma voi siis vain tilata aiheen ja odottaa, että MQTT-välittäjä lähettää ohjelmalle dataa.

Pääfunktio on esitettynä liitteessä E. Aluksi pääfunktiossa luetaan tilatiedosto, jotta dataa osataan kirjoittaa oikeaan tiedostoon oikealle riville. Tämän jälkeen alustetaan MQTT-asiakas ja määritetään MQTT-yhteyden avaamisesta sekä viestien vastaanottamisesta vastuussa olevat funktiot. Alustusten jälkeen ohjelma yhdistää MQTT-välittäjään. Jos yhdistämien ei onnistu, tulostetaan tästä viesti ja lopetetaan ohjelma. Yhteyden onnistuessa aloitetaan MQTT-asiakkaan silmukka ja odotetaan muutama sekunti, jotta yhteys MQTT-välittäjään ja MQTT-tilaus ovat varmasti toteutuneet. Nyt ohjelma on alustanut MQTT-asiakkaan toimimaan halutulla tavalla. Seuraavaksi ohjelmassa aloitetaan pääsilmukka, jossa mittausdatalle toteutetaan vaaditut toimenpiteet. Ensiksi mittausdata tallennetaan halutuilta mittausantureilta tietorakenteisiin `read_temperatures`-funktiolla. Sitten mittausdata kirjoitetaan tiedostoon `write_to_file`-funktiolla, päivitetään verkkopalvelimen kuvaaja `update_server_plot`-funktiolla ja lähetetään tarvittaessa hälytysviesti `send_mattermost_message`-funktiolla. Silmukan lopussa lämpötilamittausten tietorakenteet tyhjennetään ja odotetaan 30 minuuttia ennen kuin silmukka toistetaan. Mittausanturit lähettävät mittausdataa jatkuvasti, ja jos kaikki tämä data otettaisiin talteen, sitä olisi liian suuri määrä. Ei ole myöskään järkevää tallentaa mittauksia muutaman sekunnin välein, koska lämpötila ei voi muuttua noin pienessä aikaikkunassa huomattavasti. Kun ohjelma suljetaan, pääfunktiossa `finally`-lohko ajetaan läpi. Lohkossa tallennetaan tilatiedosto ja lopetetaan MQTT-asiakkaan silmukka.

5. YHTEENVETO

Tämän työn tarkoituksena oli luoda lämpötilan valvontajärjestelmä käyttämällä IoT-laitteita. Ennen itse sovelluksen toteutusta, työssä perehdyttiin IoT:n ja IIoT:n käsitteisiin, erilaisiin tiedonsiirtoprotokolliin ja ohjelmointirajapintoihin. Kun teoriaan oli perehdytty tarpeeksi, ohjelmoitiin sovellus järjestelmän toimintalogiikan toteuttamiseksi. Sovelluksen vaatimuksina oli vastaanottaa mittausdataa, varastoida lämpötilan mittaukset, piirtää verkkopalvelimelle kuvaaja lämpötilan mittauksista ja lähettää tarvittaessa hälytysviesti lämpötilan mittausten poiketessa viitearvoista.

Taustatutkimuksessa keskityttiin aluksi IoT:n arkkitehtuuriin sekä IIoT:n käsitteeseen. IoT:n arkkitehtuuriin on tärkeä perehtyä, jotta IoT-järjestelmien kokonaisuutta voisi ymmärtää. Työn toteutettava järjestelmä kuuluu IIoT:n sovellusdomainiin, joten IIoT:n käsite on oleellista sisäistää. Seuraavaksi teoriassa käsiteltiin tiedonsiirtoprotokollia, joista tarkemmin MQTT- ja HTTP-protokollia. MQTT on IoT-järjestelmissä erinomainen protokolla esimerkiksi mittausdatan välittämiseen, koska se on kevyt ja yksinkertainen toteuttaa. HTTP-protokollalla puolestaan on yksinkertaista lähettää dataa verkkopalvelimelle. Työn järjestelmässä hyödynnetään MQTT-protokollaa tiedonsiirtoon reitittimeltä ohjelmoitavalle sovellukselle, ja HTTP-protokollaa hyödynnetään verkkopalvelimelle kuvaajan piirtämiseen sekä hälytysviestien lähettämiseen. Taustatutkimuksen lopussa käsitellään ohjelmointirajapintoja, joita hyödynnetään sovelluksen ohjelmoinnissa.

Työ toteutettiin onnistuneesti konfiguroimalla Ruuvi Gateway -reititin lähettämään dataa MQTT-palvelimelle ja ohjelmoimalla sovellus järjestelmän toimintalogiikan toteuttamiseksi. Ohjelmoitava sovellus toteutettiin Python-ohjelmointikielellä sen laajojen ohjelmointikirjastojen ja helppokäyttöisyyden takia. Sovellus toimii MQTT-asiakkaana, joka yhdistää MQTT-välittäjään ja tilaa sopivan aiheen. Tällä tavoin sovellus vastaanottaa mittausdataa välittäjältä. Sovellus tallentaa mittausdatan sopiviin tietorakenteisiin ja varastoi sen CSV-tiedostoihin, piirtää lämpötilan mittauksista kuvaajan verkkopalvelimelle ja lähettää tarvittaessa hälytysviestin Mattermost-viestipalveluun. Datatiedostoon kirjoitetaan haluttu määrä rivejä, jonka jälkeen aloitetaan datan kirjoittaminen uuteen tiedostoon. Tiedostoja tallennetaan myös haluttu määrä, jonka ylittämisen jälkeen vanhin tiedosto tuhotaan. Kun ohjelma lopettaa toimintansa, tallennetaan tilatiedosto, joka pitää kirjaa jo tallennetuista datatiedostoista. Ohjelman käynnistyessä taas luetaan kyseinen tilatiedosto. Lämpötilan mittauksista piirretään paikallisesti toimivalle verkkopalvelimelle kuvaaja, jotta datasta nähdään graafinen esitys. Hälytysviesti lähetetään Mattermost-viestipalveluun, jos mittaukset poikkeavat lämpötilarajoista. Sovellus on toteutettu siten,

että sitä on helppo konfiguroida. Esimerkiksi tiedostomäärä, rivimäärä, lämpötilojen ala- ja ylärajat ovat kaikki määritelty vakioina ohjelman alussa, josta niitä on helppo muokata tarvittaessa.

Järjestelmää on mahdollista jatkokehittää useilla eri tavoilla, kuten lisäämällä mittausantureita, hyödyntämällä muita anturien mittaamia suureita, lähettämällä ilmoitusviesti anturien paristonvaihtotarpeesta tai hyödyntämällä tietokantaa datan varastoimiseen. Mittausantureiden lisääminen tekisi lämpötilan mittauksesta luotettavampaa, koska tällöin yksittäisen mittausanturin vikatilanne ei olisi niin merkittävä. Työssä käytettävät mittausanturit mittaavat lämpötilan lisäksi ilmanpainetta, liikettä ja ilmankosteutta. Palvelinhuoneille on määritelty ideaaliset olosuhteet, joissa lämpötilalle sekä ilmankosteudelle on määritelty viitearvot. Ilmankosteuden mittausta hyödyntämällä voisi palvelinhuoneen olosuhteista tehdä mahdollisimman ideaaliset. Jos mittausantureilta saataisiin ilmoitus paristonvaihtotarpeesta, olisi silloin kätevä käydä vaihtamassa anturin paristo tarvittaessa. Datan varastoinnin voisi vaihtoehtoisesti toteuttaa hyödyntämällä tietokantaa, joka on erinomainen tapa tallentaa dataa järjestelmällisesti.

LÄHTEET

[1] B. Marr. 2024 IoT And Smart Device Trends: What You Need To Know For The Future. Forbes. 2023. Saatavissa (Viitattu 11.3.2024): https://www.forbes.com/sites/bernardmarr/2023/10/19/2024-iot-and-smart-device-trends-what-you-need-to-know-for-the-future/?sh=3556a9617f34

[2] A. Almubarak. The Effects of Heat on Electronic Components. international Journal of Engineering Research and Applications, vol. 7, no. 5, May 2017, pp. 52–57. Saatavissa (Viitattu 17.08.2024): https://www.ijera.com/papers/Vol7_issue5/Part-5/H0705055257.pdf

[3] S. Madakam, R. Ramaswamy, S. Tripathi. Internet of Things (IoT): A Literature Review. Journal of Computer and Communications, vol. 3, no. 5, May 2015, pp. 164–173. Saatavissa (Viitattu 13.3.2024): 10.4236/jcc.2015.35021

[4] N. Soumyalatha. Study of IoT: Understanding IoT Architecture, Applications, Issues and Challenges. International Journal of Advanced Networking & Applications (IJANA). 5.2016. Saatavissa (Viitattu 13.3.2024): https://www.rsearchgate.net/profile/Soumyalatha-Naveen/publication/330501274_Study_of_IoT_Understanding_IoT_Architecture_Applications_Issues_and_Challenges/links/5c434fea458515a4c731d4bb/Study-of-IoT-Understanding-IoT-Architecture-Applications-Issues-and-Challenges.pdf

[5] S. Chithra. & R. Yugha. A survey on technologies and security protocols: Reference for future generation IoT. Journal of Network and Computer Applications, vol. 169, November 2020. Saatavissa (Viitattu 14.3.2024): https://doi.org/10.1016/j.jnca.2020.102763

[6] A. El Hakim. Internet of Things (IoT) System Architecture and Technologies, White Paper. ResearchGate. 2018. Saatavissa (Viitattu 14.3.2024): https://doi.org/10.13140/RG.2.2.17046.19521

[7] H. Boyes. J. Cunningham. B. Hallaq. T. Watson. The industrial internet of things (IIoT): An analysis framework. Computers in Industry, vol. 101, 2018, pp. 1–12. Saatavissa (Viitattu 15.3.2024): https://doi.org/10.1016/j.compind.2018.04.015

[8] G. McGraw. Software security. IEEE Security & Privacy, vol. 2, no. 2, March 2004, pp. 80–83. Saatavissa (Viitattu 15.3.2024): https://doi.org/10.1109/MSECP.2004.1281254

[9] IIoT World. IoT vs. IIoT. Verkkosivu. 2022. Saatavissa (Viitattu 5.3.2024): https://www.iiot-world.com/industrial-iot/connected-industry/iot-vs-iiot/

[10] T. Maria. Protokolla. Techopedia. 2024. Saatavissa (Viitattu 19.3.2024): https://www.techopedia.com/fi/sanasto/protokolla

[11] K. Fall. & W. Stevens. TCP/IP Illustrated, Volume 1: The Protocols, 2nd Edition. Addison-Wesley Professional. 2011.

[12] J. Vihervaara. Tietoverkkotekniikat I. Tampereen yliopisto. Saatavissa (Viitattu 19.3.2024): https://moodle.tuni.fi/pluginfile.php/1260689/mod_resource/content/9/Tietoverkkotekniikat-kirja_2023.pdf

[13] Osi Model. Imperva. Verkkosivu. Saatavissa (Viitattu 19.3.2024): https://www.imperva.com/learn/application-security/osi-model/

[14] OSI model and TCP/IP stack Bit, Frame, Packet, Segment, Datagram Information Encapsulation. IT Bundle. 2015. Saatavissa (Viitattu 25.3.2024): https://it-bundle.net/archives/289

[15] MQTT Version 5.0. OASIS Open. 2019. Saatavissa (Viitattu 26.3.2034): https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.pdf

[16] S. Singh. & S. Yadav. Introduction to Client Server Computing. New Age International Ltd. 2009.

[17] S. Tarkoma. Publish / Subscribe Systems: Design and Principles. Wiley. 2012. Saatavissa (Viitattu 26.3.2024): https://doi.org/10.1002/9781118354261

[18] S. Manandhar. MQTT based Communication in IoT. Tampereen yliopisto. 2017.

[19] MQTT Topics, Wildcards, & Best Practices – MQTT Essentials: Part 5. HiveMQ. 2019. Saatavissa (Viitattu 26.3.2024): https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/

[20] MQTT: The Standard for IoT Messaging. MQTT.org. Verkkosivu. 2022. Saatavissa (Viitattu 12.4.2024): https://mqtt.org/

[21] I. Craggs. MQTT Vs. HTTP for IoT. HiveMQ. 2022. Saatavissa (Viitattu 12.4.2024): https://www.hivemq.com/blog/mqtt-vs-http-protocols-in-iot-iiot/

[22] Definition of HTTPS. PCMAG. Verkkosivu. Saatavissa (Viitattu 19.04.2024): https://www.pcmag.com/encyclopedia/term/https

[23] S. Saha. What is an API? Tech N Techie - Technology Simplified for You. 2019. Saatavissa rajoitetusti (Viitattu 18.4.2024): https://www.proquest.com/docview/2248107721?accountid=14242&parentSessionId=%2BKvnQu9wDraak5W18B5Hi5hl%2BmP0fULjF%2B3rAiQwA6Q%3D&pq-origsite=primo&sourcetype=Blogs,%20Podcasts,%20&%20Websites

[24] REST API Introduction. GeeksforGeeks. Verkkosivu. 2023. Saatavissa (Viitattu 18.04.2024): https://www.geeksforgeeks.org/rest-api-introduction/

[25] L. Bassett. Introduction to JavaScript Object Notation. O'Reilly Media Inc. 2015. Saatavissa rajoitetusti (Viitattu 9.5.2024): https://learning.oreilly.com/library/view/introduction-to-javascript/9781491929476/

[26] RuuviTag Pro -tuotteen kotisivut. Ruuvi Innovations Oy. https://ruuvi.com/fi/ruuvitag-pro/

[27] Ruuvi Gateway -tuotteen kotisivut. Ruuvi Innovations Oy. https://ruuvi.com/fi/gateway/

[28] The Python Package Index. Python Software Foundation. Verkkosivu. Saatavissa (Viitattu 6.5.2024): https://pypi.org/

[29] Eclipse Mosquitto. The Eclipse Foundation. Verkkosivu. Saatavissa (Viitattu 13.5.2024): https://mosquitto.org/

[30] Eclipse Paho MQTT Python Client. The Eclipse Foundation. Saatavissa (Viitattu 13.5.2024): https://eclipse.dev/paho/index.php?page=clients/python/index.php

[31] S. Isiodoro. ruuvi-decoders. Github repositorio. 2021 Saatavissa (Viitattu 15.5.2024): https://github.com/ruuvi-friends/ruuvi-decoders

[32] csv—CSV File Reading and Writing. Python Documentation. Saatavissa (Viitattu 6.5.2024): https://docs.python.org/3/library/csv.html

[33] qbee.io. http-plot-server 0.1. The Python Package Index. 2022. Saatavissa (Viitattu 6.5.2024): https://pypi.org/project/http-plot-server/

[34] K. Reitz. requests 2.31.0. The Python Package Index. 2023. Saatavissa (Viitattu 6.5.2024): https://pypi.org/project/requests/

[35] Webhooks. Atlassian Developer. Saatavissa (Viitattu 6.5.2024): https://developer.atlassian.com/server/jira/platform/webhooks/

[36] Cool-Tec. What Is The Correct Temperature For A Server Room? Cool-Tec Services Ltd. 2023. Saatavissa (Viitattu 15.5.2024): https://cool-tec.co.uk/blog/correct-temperature-for-server-rooms/

[37] asyncio—Asynchronous I/O. Python Documentation. Saatavissa (Viitattu 15.5.2024): https://docs.python.org/3/library/asyncio.html

LIITE A: MITTAUSDATAN KERÄÄMISEN JA TIETORAKENTEISIIN TALLENTAMISEN OHJELMAKOODI

```
def on_message(client, userdata, msg):
    """
    Kun tämä ohjelma saa MQTT-viestin tilatulla aiheella, tämä funktio
ajetaan. Funktio purkaa lähetetyn viestin haluttuun muotoon ja lopulta
tallentaa Ruuvi-mittausanturin MAC-osoitteen sekä lämpötilan mittaus-
arvon dictiin.
    :param client: MQTT-asiakas.
    :param userdata: Tietorakenne. Tässä ohjelmassa tätä ei hyödyn-
netä.
    :param msg: MQTT-välittäjän lähettämä viesti kokonaisuudessaan,
eli se sisältää aiheen sekä itse viestin.
    :return: Ei palauta mitään.
    """
    message = msg.payload.decode()
    raw_data = json.loads(message)

    if "data" in raw_data:
        # Puretaan viesti
        decoder = Df5Decoder()
        data = raw_data["data"]
        data = data.split("FF9904")[1]
        sensor = decoder.decode_data(data)

        temp = sensor["temperature"]
        mac = sensor["mac"]
        # Muokataan mac-osoite haluttuun formaattiin
        formatted_mac = ":".join([mac[i:i + 2] for i in range(0,
len(mac), 2)])
        formatted_mac = formatted_mac.upper()

        # Tallennetaan mac-osoite ja lämpötilan mittaus dictiin
        temp_measurements[formatted_mac] = temp


def on_connect(client, userdata, flags, rc):
    """
    Kun MQTT-välittäjään yhdistetään, niin tämä funktio ajetaan. Funk-
tio tilaa aiheen "ruuvi/#", jolloin kaikki MQTT-data Ruuvi-antureilta
tilataan.
    :param client: MQTT-asiakas.
    :param userdata: Tietorakenne. Tässä ohjelmassa tätä ei hyödyn-
netä.
    :param flags: tietorakenne, joka sisältää MQTT-välittäjän vastaus-
flagit.
    :param rc: Kokonaisluku, joka kertoo yhteyden onnistumisen tulok-
sen.
    :return: Ei palauta mitään.
    """
    print("Yhdistetty MQTT-välittäjään onnistuneesti.")
    client.subscribe("ruuvi/#")
```

```
def read_temperatures(temp_measurement):
    """
    Lukee lämpötiladatan halutuilta antureilta tietorakenteisiin.
    :param temp_measurement: Gatewaylta saadut lämpötilanmittaukset.
    :return: Palauttaa palvelinkaapin etupuolella olevien anturien
lämpötilamittaukset sekä palvelinkaapin takana olevien anturien lämpö-
tilamittaukset.
    """
    front_temp_measurements = {}
    behind_temp_measurements = {}

    try:
        file_front_sensors = open('front_sensor_MAC_addresses.txt',
'r')
        front_sensor_mac_addresses = file_front_sensors.readlines()
        front_sensor_mac_addresses = [mac.strip() for mac in
                                      front_sensor_mac_addresses]
        file_front_sensors.close()

        file_behind_sensors = open('behind_sensor_MAC_addresses.txt',
'r')
        behind_sensor_mac_addresses = file_behind_sensors.readlines()
        behind_sensor_mac_addresses = [mac.strip() for mac in
                                       behind_sensor_mac_addresses]
        file_behind_sensors.close()

    except OSError:
        print("Anturien MAC-osoitteiden lukemisessa tapahtui virhe")
        sys.exit()

    print("Nämä arvot otetaan talteen:")
    for mac in temp_measurement.keys():

        if mac in front_sensor_mac_addresses:
            temp = temp_measurement[mac]
            front_temp_measurements[mac] = temp
            print(f'Front: {mac}: {temp}')

        if mac in behind_sensor_mac_addresses:
            temp = temp_measurement[mac]
            behind_temp_measurements[mac] = temp
            print(f'Behind: {mac}: {temp}')

    return front_temp_measurements, behind_temp_measurements
```

LIITE B: MITTAUSDATAN TIEDOSTOON KIRJOITTAMISEN JA TILATIEDOSTON TOIMINNAN OHJELMAKOODI

```
def write_to_file(file_count, row_count, file_names_list,
                  front_temp_measurements, behind_temp_measurements):
    """
    Kirjoittaa mittausdatan talteen csv-tiedostoon. Huolehtii myös
siitä, mihin tiedostoon dataa pitää kirjoittaa.
    :param file_count: Monesko tiedosto on menossa, johon kirjoitetaan
dataa.
    :param row_count: Monesko rivi on menossa tiedostossa.
    :param file_names_list: Lista tiedostojen nimistä, johon dataa on
jo tallennettu. Tämän avulla viimeisen tiedoston poistaminen on help-
poa.
    :param front_temp_measurements: palvelinkaapin etupuolella olevien
        antureiden lämpötilamittaukset.
    :param behind_temp_measurements: palvelinkaapin takapuolella ole-
vien antureiden lämpötilamittaukset.
    :return: Palauttaa päivitettyinä tiedostonumeron, rivinumeron ja
listan tiedostojen nimistä.
    """
    if len(file_names_list) >= MAX_FILES:
        file_names_list = delete_file(file_names_list)

    if row_count > ROW_LIMIT:
        file_count += 1
        row_count = 0

    try:
        current_data_file_as_string = f"{DATA_FILE_NAME}{file_count}"\
                                      f"{DATA_FILE_EXTENSION}"
        current_data_file = open(f"{DATA_FILE_NAME}{file_count}"
                                 f"{DATA_FILE_EXTENSION}", "a", new-
line='')

        csvwriter = csv.writer(current_data_file, delimiter=",")

        # Jos uusi tiedosto, rivi = 0
        if row_count == 0:
            # Kirjoitetaan datatiedostoon otsikot
            csvwriter.writerow(FIELDS)
            # Lisätään file_name_list-listaan, jotta voidaan poistaa
helposti vanhin tiedosto tarvittaessa
            file_names_list.append(current_data_file_as_string)

        # Kirjoitetaan itse mittausdata tiedostoon
        for mac in front_temp_measurements:
            row = [mac, front_temp_measurements[mac],
                   'Front', datetime.now()]
            csvwriter.writerow(row)
            row_count += 1

        for mac in behind_temp_measurements:
            row = [mac, behind_temp_measurements[mac],
                   'Behind', datetime.now()]
```

```
            csvwriter.writerow(row)
            row_count += 1

        print("Kirjoitettu tiedostoon onnistuneesti.")
        current_data_file.close()

    except Exception as e:
        print("Tapahtui virhe tiedostoon kirjoittaessa: " + str(e))

    return file_count, row_count, file_names_list


def delete_file(file_names_list):
    """
    Poistaa vanhimman datatiedoston.
    :param file_names_list: Lista tiedostonimistä, jotka ovat jo tal-
lennettu.
    :return: Palauttaa päivitetyn listan tiedostonimistä, joista vii-
meinen poistettu listasta.
    """
    try:
        os.remove(file_names_list[0])
        file_names_list.pop(0)
        print("Tiedosto poistettu onnistuneesti.")

    except Exception as e:
        print("Tapahtui virhe tiedostoa poistaessa: " + str(e))

    return file_names_list


def read_state():
    """
    Lukee state.txt tiedostosta tilan, mihin ollaan jääty datan
    kirjoittamisessa.
    :return: Palauttaa tiedostonumeron ja rivinumeron mihin ollaan
jääty sekä listan jo tallennetuista datatiedostoista
    """
    try:
        statefile = open("state.txt", "r")
        data = statefile.read().splitlines()
        last_file_number = int(data[0])
        last_row_number = int(data[1])
        file_names_list = data[2:]
        print("state.txt-tiedosto luettu onnistuneesti.")
        return last_file_number, last_row_number, file_names_list

    except FileNotFoundError:
        # Jos tilatiedostoa ei ole olemassa, aloitetaan alusta
        print("state.txt-tiedostoa ei löytynyt, aloitetaan alusta")
        return 1, 0, []


def save_state(file_number, row_number, file_name_list):
    """
    Kirjoittaa state.txt tiedostoon tilan, mihin tallennetaan tieto
siitä, mihin tiedostonumeroon ja rivinumeroon ollaan jääty sekä listan
jo tallennettujen datatiedostojen nimistä.
    :param file_number: Monesko tiedosto on menossa, johon kirjoite-
taan dataa.
    :param row_number: Monesko rivi on menossa tiedostossa.
```

```
    :param file_name_list: Lista jo tallennetuista datatiedostoista.
    :return: Ei palauta mitään.
    """
    try:
        statefile = open(STATE_FILE, "w")
        statefile.write(f"{file_number}\n{row_number}\n")
        for file_name in file_name_list:
            statefile.write(f"{file_name}\n")
        statefile.close()
        print("state-tiedosto kirjoitettu onnistuneesti")

    except Exception as e:
        print("state-tiedoston kirjoittaminen ei onnistunut." +
str(e))
```

LIITE C: VERKKOPALVELIMEN KUVAAJAN PÄIVITTÄMISEN OHJELMAKOODI

```
def update_server_plot(front_temp_measurements, behind_temp_measure-
ments):
    """
    Päivittää verkkopalvelimen kuvaajaa.
    :param front_temp_measurements: palvelinkaapin etupuolella olevien
antureiden lämpötilamittaukset.
    :param behind_temp_measurements: palvelinkaapin takana olevien an-
tureiden lämpötilamittaukset.
    :return: Ei palauta mitään.
    """
    tag = "Front temperature measurement"
    unit = "°C"

    try:
        for temperatue_value in front_temp_measurements.values():
            data = {
                "tag": tag,
                "value": temperatue_value,
                "unit": unit,
                "ts": time.time()
            }
            requests.post(WEBSITE_URL, json=data)

        tag = "Behind temperature measurement"
        for temperatue_value in behind_temp_measurements.values():
            data = {
                "tag": tag,
                "value": temperatue_value,
                "unit": unit,
                "ts": time.time()
            }
            requests.post(WEBSITE_URL, json=data)

    except Exception as e:
        print("Palvelimen kuvaajan päivittäminen epäonnistui: " +
str(e))

    print("Palvelimen kuvaaja päivitetty onnistuneesti.")
```

LIITE D: HÄLYTYSVIESTIN LÄHETTÄMISEN OHJELMAKOODI

```
def send_mattermost_message(front_temp_measurements, behind_temp_mea-
surements):
    """
    Lähettää tarvittaessa Mattermost-hälytysviestin, jos lämpötilamit-
taukset poikkeavat lämpötilarajoista.
    :param front_temp_measurements: palvelinkaapin etupuolella olevien
        antureiden lämpötilamittaukset.
    :param behind_temp_measurements: palvelinkaapin takana olevien an-
tureiden lämpötilamittaukset.
    :return: Ei palauta mitään.
    """
    try:
        for temperature_value in front_temp_measurements.values():
            if temperature_value > FRONT_TEMP_MAX:
                values = '{"text": "Temperature too high:  '\
                         + str(temperature_value).replace(".", ",") +\
                         ' degrees Celcius"}'
                requests.post(MATTERMOST_WEBHOOK, headers=HEADERS,
data=values)
                print("Lähetetty Mattermost viesti")

            if temperature_value < FRONT_TEMP_MIN:
                values = '{"text": "Temperature too low:  '\
                         + str(temperature_value).replace(".", ",") +\
                         ' degrees Celcius"}'
                requests.post(MATTERMOST_WEBHOOK, headers=HEADERS,
data=values)
                print("Lähetetty Mattermost viesti")

        for temperature_value in behind_temp_measurements.values():
            if temperature_value > BEHIND_TEMP_MAX:
                values = '{"text": "Temperature too high:  '\
                         + str(temperature_value).replace(".", ",") +\
                         ' degrees Celcius"}'
                requests.post(MATTERMOST_WEBHOOK, headers=HEADERS,
data=values)
                print("Lähetetty Mattermost viesti")

            if temperature_value < BEHIND_TEMP_MIN:
                values = '{"text": "Temperature too low:  '\
                         + str(temperature_value).replace(".", ",") +\
                         ' degrees Celcius"}'
                requests.post(MATTERMOST_WEBHOOK, headers=HEADERS,
data=values)
                print("Lähetetty Mattermost viesti")

    except Exception as e:
        print("Tapahtui virhe Mattermost viestin lähettämisessä: " +
str(e))
```

LIITE E: PÄÄFUNKTION OHJELMAKOODI

```
async def main():
    last_file_number, last_row_number, file_names_list = read_state()
    client = paho.Client(CLIENT_ID)
    client.on_message = on_message
    client.on_connect = on_connect

    if client.connect(BROKER, PORT, 60) != 0:
        print("Yhdistäminen MQTT-välittäjään ei onnistunut.")
        sys.exit(-1)
    client.loop_start()
    # Odotetaan muutama sekunti, jotta MQTT-tilaus on varmasti to-
teutunut
    time.sleep(MQTT_WAIT_TIME)

    try:
        while True:
            # Tallenna mittausdata dicteihin
            front_temp_measurements, behind_temp_measurements = \
                read_temperatures(temp_measurements)

            # Kirjoita mittausdata .csv tiedostoon
            last_file_number, last_row_number, file_names_list = \
                write_to_file(last_file_number, last_row_number,
                              file_names_list, front_temp_measure-
ments, behind_temp_measurements)

            # Päivitä serverin graafi
            update_server_plot(front_temp_measurements,
                               behind_temp_measurements)

            # Lähetä tarvittaessa mattermostiin hälytysviesti
            send_mattermost_message(front_temp_measurements,
                                    behind_temp_measurements)

            # Lopuksi nollataan dictit ja odotetaan haluttu aika,
jonka jälkeen while-looppi jatkuu
            front_temp_measurements.clear()
            behind_temp_measurements.clear()
            time.sleep(TIME_SLEEP)

    finally:
        # Tallennetaan ohjelman sulkeutuessa state.txt-tilatiedosto.
        # Lopetetaan myös MQTT-asiakkaan looppi.
        save_state(last_file_number, last_row_number, file_names_list)
        client.loop_stop()

loop = asyncio.get_event_loop()
loop.run_until_complete(main())
```