Jani Peltonen, Joonas Soininen, Matias Vainio, Teemu Viljanen

Tietoturvallinen IoT-järjestelmä ja sen hallinnointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Innovaatioprojekti

16.12.2022

Tiivistelmä

Tekijät:	Jani Peltonen, Joonas Soininen, Matias Vainio, Teemu Viljanen
Otsikko:	Tietoturvallinen IoT-järjestelmä ja sen hallinnointi
Sivumäärä:	17 sivua
Aika:	16.12.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Amir Dirin

Projektissa on toteutettu IoT-ohjelmistokehys, joka mahdollistaa erilaisten sensorijärjestelmien nopean kehittämisen. Uusien laitteiden lisäämisen yhteydessä, laitteen identiteetti pyritään todentamaan. Ohjelmistokehyksen lisäksi projektissa toteutettiin kaksi erilaista käyttöliittymää, joiden avulla käyttäjä voi tarkastella laitteiden tilaa, sekä sensoreiden lähettämää dataa. Ohjelmistokehyksen toimintaa testattiin käyttötapauksella, jossa webkamera ottaa kuvan, mikäli sensoreiden mittausdata täyttää tietyt arvot. Projektin tuotoksista toteutettiin myös tuotantoversio, joka yhdistettiin Nokian LuxTurrim-tuotantojärjestelmäympäristöön.

Avainsanat: IoT, tietoturva, TPM, MQTT, Python

Abstract

Authors:	Jani Peltonen, Joonas Soininen, Matias Vainio, Teemu Viljanen
Title:	A Secure IoT-System and its Management
Number of Pages:	17 pages
Date:	16 December 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Amir Dirin, Senior Lecturer

In this project, we created an IoT software framework that enables the rapid development of various sensor systems. When adding new devices, the identity of the device is verified. In addition to the software framework, the project implemented two different user interfaces that allow the user to view the status of the devices and the data sent by the sensors. The functioning of the framework was tested with a use case, in which a web-camera takes a picture if the data gathered from the sensors meets certain criteria. We also implemented a production version to the Nokia LuxTurrim5G pilot environment.

Keywords: IoT, security, TPM, MQTT, Python

1 Johdanto 1

2 IoT-järjestelmä 1

2.1 Kokonaiskuva 2
2.2 MQTT-kommunikointi 3
2.3 Rasperry Pi-kokoonpano 4
2.3.1 Arkkitehtuuri 5
2.3.2 Käynnistysautomaatio 6
2.4 Manager-luokan kokoonpano 7
2.5 TPM ja laitteen todentaminen 7
2.6 Datan aggregointi 8

3 Hallinnointikäyttöliittymät 8

3.1 Management log -käyttöliittymä 9
3.2 NodeRed-käyttöliittymä 9

4 Järjestelmän käyttöönotto 11

4.1 Device ja Sensor osioiden asentaminen ja ajaminen 11
4.2 ManagementAttestorin asennus ja käynnistäminen 12
4.3 ManagementUI:n asennus ja käynnistäminen 13

5 Käyttötapaus – kuvan ottaminen 14

5.1 Toimintamekanismi 14
5.2 Tietojen tallentaminen 15

6 Järjestelmän tuotantoversio LuxTurrim-ympäristössä 15

7 Yhteenveto 16

Lähteet 17

Lyhenteet

TPM: *Trusted Platform module.* Fyysinen turvapiiri, jota käytetään laitteiden turvallisuuden parantamiseen.

IoT: *Internet of Things.* Fyysisten internetiin yhdistettyjen laitteiden luoma verkosto.

MQTT: *Message Queue Telemetry Transport.* Verkkoprotokolla laitteesta laitteeseen tapahtuvan julkaisu-/tilausviestinnän jonotuspalvelu.

1 Johdanto

Tämä projekti toteutettiin yhteistyössä Nokia Bell Labs:in kanssa, ja työskentely tapahtui pääasiassa Nokian garage-työskentelytilassa.

Projektin tarkoituksena oli rakentaa malli tietoturvallisesta sensoreita käyttävästä IoT-järjestelmästä ja tehdä sen hallinnointiin käyttöympäristö, sekä toteuttaa käyttötapausesimerkki.

Asiakkaan tavoite muuttui projektin edetessä useaan otteeseen. Alkuperäinen projektin suunnitelma sisälsi pelkästään yksittäisen sensorijärjestelmän rakentamisen, ja järjestelmän keräämän datan muuttamisen lohkoketjussa säilytettävään muotoon. Näitä projekti-iteraatioita oli useita, mutta lopulta päädyttiin tekemään sovelluskehys, joka mahdollistaa yleisluontaisten sensorijärjestelmien kehittämisen.

Lopullinen tuotantoversio rakennettiin hyödyntäen tekemäämme sovelluskehystä, ja se asennettiin Nokian LuxTurrim-ekosysteemiin, joka koostuu 5G teknologialla toisiinsa yhdistetyistä IoT-laitteista ja sensoreista.

Projektin IoT-järjestelmän puolella on käytetty Raspberry Pi –pientietokoneita, Python-ohjelmointikieltä, MQTT-viestinvälitysteknologiaa, sekä Flask-kirjastoa. Järjestelmän hallinnointikäyttöliittymä on rakennettu erikseen käyttäen Javascript-ohjelmointikieltä ja palvelin on tehty Node.js-teknologiaa hyödyntäen.

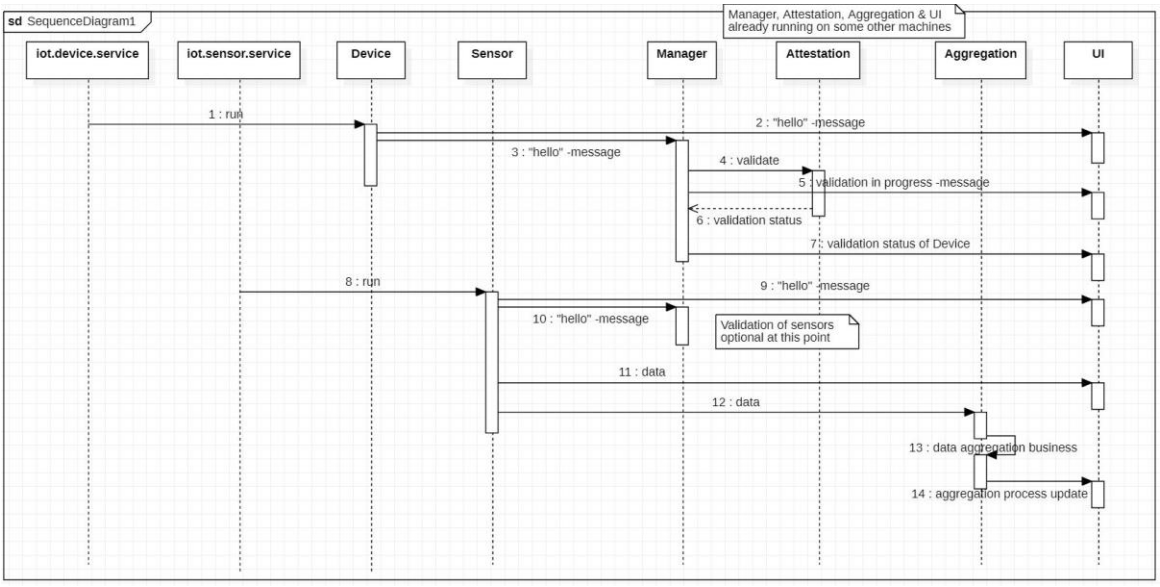
2 IoT-järjestelmä

Lyhenteellä IoT tarkoitetaan järjestelmiä, jotka koostuvat erilaisilla sensoreilla varustetuista ja prosessointikykyisistä olioista [1]. Näiden olioiden välinen viestintä tapahtuu verkon välityksellä. Tässä luvussa kerrotaan, minkälaisen IoT-järjestelmän saimme projektin aikana kehitettyä.

2.1 Kokonaiskuva

Projektin tavoitteena oli tehdä sensorien käyttöön ohjelmistokehys, joka mahdollistaa erilaisten sensorijärjestelmien nopean ja yksinkertaisen kehittämisen. Päämääränä oli abstrahoida rakenne niin, että kehittäjän tarvitsee vain luoda oman systeeminsä käyttämät tietyt koodirivit kehyksen hoitaessa kaiken muun.

Järjestelmä koostuu kolmesta eri osa-alueesta: 1) Raspberry Pi –koneelle asennetusta sensori- ja laiteohjelmistosta, 2) välitason manageriohjelmasta ja todentajarajapinnasta, sekä 3) käyttöliittymistä. Järjestelmän toiminnan kannalta on suositeltavaa, että välitason ohjelmat sekä käyttöliittymät pyörivät eri laitteilla kuin sensorit.



Kuva 1. Järjestelmän sekvenssikaavio.

Kuvassa 1 on esitetty järjestelmän toiminta pääpiirteissään. Yllä mainittujen kolmen osa-alueen jako on kuvassa seuraava: 1) service-tiedostot, Device ja Sensor, 2) Manager, Attestation ja Aggregation, ja 3) UI, eli *user interface*.

2.2 MQTT-kommunikointi

Järjestelmän eri osa-alueet kommunikoivat keskenään käyttäen MQTT-viestejä. Kaikkien osa-alueiden ohjelmistot käyttävät samaa MQTT-palvelinta, ja lähettävät viestejä erikseen määritettyjen konventioiden mukaan nimetyillä kanavilla.

Hallinnointikanava on nimeltään “management”, ja sillä kulkevat viestit ovat järjestelmän hallinnallisia viestejä, kuten esimerkiksi tieto laitteen tai sensorin käynnistymisestä.

Kukin sensori lähettää keräämänsä datan omalla kanavallaan, jotka on nimetty konventiolla “prefix/datatype”, jossa *prefix* on sensorin nimi tai tunnistustieto ja *datatype* on kerätyn mittausdatan tyypin kuvaus. Esimerkiksi lämpötilasensori voi lähettää mittaamansa datan kanavalla “TemperatureSensor043/temperature”.

Lisäksi käytössä on kanava nimeltä “alert”, jolla lähetetään viestejä, jotka liittyvät enemmänkin järjestelmän päälle rakennetun ohjelmiston toimintaan. Esimerkiksi dataa keräävä aggregointiohjelmisto voi alert-kanavalla ilmoittaa kriittisistä muutoksista sensorien datan mittausarvoissa.

Management-kanavalla MQTT-viestin mukana kulkee Python sanakirja -olio, joka sisältää suuren määrän tietoja niin laitteesta, kuin sensorista sekä eri tapahtumat ja viestit.

```
# This is just a a structure for the payload sent from different
# parts of the program.
self.message = {
    "event": "",
    "message": "",
    "messagetimestamp": "",
    "device": {
        "itemid": self.device_config["itemid"],
        "hostname": self.device_config["hostname"],
        "address": self.device_config["address"],
        "starttimestamp": "",
        "valid": False,
        "validtimestamp": ""
    },
    "sensor": {
        "name": "",
        "starttimestamp": "",
        "valid": False,
        "validtimestamp": ""
    }
}
```

Kuva 2. Python sanakirjan rakenne.

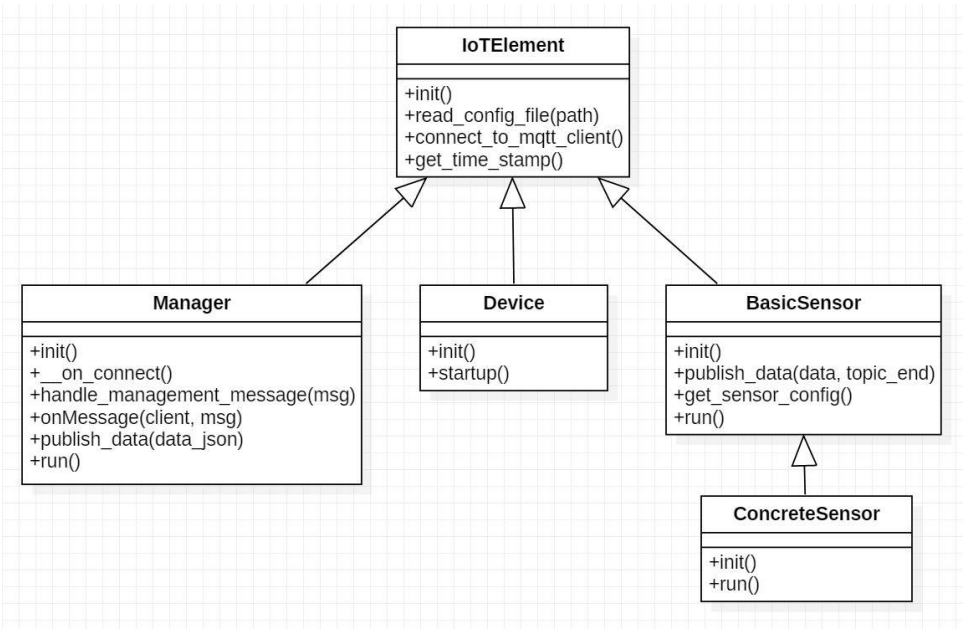
Olion sisällön näkee kuvasta 2. Merkityksellisiä tietoja ovat event-osio, jonka pohjalta eri luokkien toiminnallisuus määräytyy, ja message-osio, joka määrittää esitettäviä viestejä järjestelmän käyttöliittymissä.

2.3 Rasperry Pi-kokoonpano

Jokaisessa Raspberry Pi:ssä olisi suositeltavaa olla /opt/ kansiossa ohjelmistokehyksemme repositorio ja konkreettisten sensorien tiedostot. Ohjelmistokehystämme käyttävän laitteen konfigurointitiedostot, joissa määritetään laitteen nimi, osoite, todentamisessa käytetty itemid-tunniste, ja MQTT-palvelimen tiedot, luodaan /etc/iotDevice/ -kansioon. Laitteen ja sensorien systemd:n käyttämät service-käynnistystiedostot tiedostot kopioidaan asennusvaiheessa /etc/systemd/system/ kansioon. Käyttäjä voi kuitenkin itse luoda service-tiedostoja ja siirtää niitä aikaisemmin mainittuun kansioon.

2.3.1 Arkkitehtuuri

Järjestelmä on ohjelmoitu käyttäen olio-ohjelmoinnin periaatteita. Arkkitehtuurin suunnittelussa pyrittiin huomioimaan se, että kehittäjän ei tarvitse ohjelmoida muuta, kuin hänen järjestelmälleen ominaiset piirteet. Kaikki järjestelmälle yhteinen ja järjestelmästä riippumaton tila, sekä toiminnallisuus, on nostettu hierarkiassa mahdollisimman korkealle.



Kuva 3. Luokkakaavio, josta ilmenee perintäpuu.

Kuvassa 3 esitetyn arkkitehtuurin perustana on IoTElement-luokka, jonka harteille jää konfiguraatiotiedostojen lukeminen ja näiden perusteella oikealle MQTT-palvelimelle yhdistäminen. Luokkaan on toteutettu metodit, joita käytetään suoraan, kun tämä otetaan käyttöön jossakin kolmesta aliluokasta.

BasicSensor-luokka sisältää geneerisen toiminnallisuuden, mitä konkreettinen sensori voi tarvita. Tärkeimpänä metodina toimii publish_data, jonka avulla saadaan lähetettyä tietoa yliluokassa määritetylle MQTT-palvelimelle. Ohjelmistokehyksen käyttäjä antaa datan ja MQTT-topic:in loppuosan parametreina, jotta viesti menee oikeaan osoitteeseen. Jokaiselle sensorille on myös toteutettu

konfiguraatiotiedosto, jonka lukeminen tapahtuu myös tässä BasicSensor-luokassa.

Manager-luokkaa käytetään sensoreiden hallintaan. Tämänkin luokan tulee olla yhteydessä samaan MQTT-palvelimeen, joten kokoonpanoasetukset tulevat yhtä lailla IoTElementiltä.

Device-luokka kuvastaa jotakin laitetta. Jos käyttäjällä on tarve kehittää virtuaalisia laitteita, voi hän hyödyntää tätä luokkaa. Kuten muutkin luokat, Device-luokka hyödyntää IoTElementistä saamaansa tilaa erilaisiin toimintoihin.

Raspberry Pi –laitteille tulevasta ohjelmistosta tehtiin myös Pythonin paketinhallintaohjelma pip:illä asennettava kirjasto, jotta uusien laitteiden ja sensoreiden käyttöönotto olisi mahdollisimman helppoa.

2.3.2 Käynnistysautomaatio

Systemd on Linux-käyttöjärjestelmän ja siinä ajettavien palveluiden hallinnoimisesta vastaava järjestelmä [2]. Systemd:n palvelut määritetään niin sanotuissa service tiedostoissa, jotka otetaan käyttöön systemctl nimisellä työkalulla.

Ohjelmistokehyksemme käyttää systemd:n service tiedostoja käynnistymisen aikana tapahtuvien toimintojen suorittamiseen. Alustavasti käytössä on kaksi service-tiedostoa iot.devices.service ja laitteella olevan sensorin tiedosto esimerkiksi iot.sensors.ir_sensor.service. Sovelluskehyksemme asennusvaiheessa käyttäjä aktivoi iot.devices.service-tiedoston, jotta se suoritetaan käynnistys vaiheessa. Itse service lähettää laitteen käynnistymisestä MQTT-viestin management-kanavalle, kertoen oman nimensä ja tapahtumansa.

2.4 Manager-luokan kokoonpano

Manager-luokka on erillinen ohjelma, joka pyörii erillisellä koneella, projektin osalta demoversiossa palvelinta pyörittävällä koneella ja tuotantoversiossa Lux-Turrim-ympäristöön liitetyllä tietokoneella [3]. Luvussa 6. on kerrottu tuotantoversiosta tarkemmin.

Luokan pääasiallinen tehtävä on hallinnoida järjestelmää yhdistämällä MQTT-palvelimeen ja tilaamalla management-kanavan. Projektin aikana luokan ainoaksi toteutuneeksi tehtäväksi tuli hallinnoida laitteiden ja sensorien validointia. Manager-luokka kuulee kaikki management-kanavan viestit ja aloittaa uuden säikeen jokaisesta viestistä. Luokka tekee jotain vain kuullessaan tietyn eventin Python-objektista, joka kulkee management-viestien mukana. Laitteen validointi käynnistyy eventin ollessa ”device startup”, jonka kuullessaan manager-luokka tekee REST-kutsun todentajarajapintaan ja saa paluuarvona tiedon validoinnin onnistumisesta. Tämän jälkeen manager-luokka lähettää uuden viestin management-kanavalla, joka kertoo todennuksen onnistumisesta tai epäonnistumisesta.

Manager-luokalle suunnittelimme useita lisäominaisuuksia ja toiminnallisuuksia, joiden avulla laitteita sekä sensoreita olisi mahdollista ohjata. Ominaisuuksiin kuului muun muassa sensorin uudelleenkäynnistys, sensorin lähettämän datan intervallin muuttaminen sekä koko laitteen uudelleenkäynnistys. Nämä ominaisuudet olisivat toimineet yhtä lailla kuuntelemalla management-viestien Python-objektien sisällön event-osuutta.

2.5 TPM ja laitteen todentaminen

TPM on nykyisissä suorittimissa valmiina oleva turvapiiri. Raspberry Pi vaatii erillisen lisäosan. TPM:llä pystyy esimerkiksi sattumanvaraisesti generoimaan numeroita, varmentamaan turvallisen salausavainten luomisen ja etänä todentamaan, että laitteen kokoonpanoa ei ole muutettu. [4] Ohjelmistokehyksemme

mahdollistaa erillisen TPM-turvallisuuspiirin käyttämisen Raspberry Pi:n kanssa, jotta sille voidaan suorittaa todentaminen.

Todentaminen pohjautuu siihen, että erilliselle Nokialla kehitettyyn AttestationEngine-palvelimelle on lisätty kyseisen laitteen tunnistetieto [5]. Mikäli laitteen tunnistetieto puuttuu, tai sitä on jollain tavalla muokattu, todentaminen epäonnistuu.

Konkreettisesti todentaminen tapahtuu siten, että palvelimen REST-rajapintaan tehdään erilaisia kyselyjä. Aluksi avataan uusi istunto, jotta seuraavat pyynnöt menevät kyseisen istunnon alle. Seuraavaksi suoritetaan todentaminen, jonka tekemiseen tarvitaan laitteen tunniste, todentamiskäytänteen tunniste sekä edellä mainitun istunnon tunniste. Todentamisen onnistuttua tuloksena saadaan vaatimustunniste, jota käytetään lopullisessa varmistuksessa. Lopulta suljetaan istunto.

2.6 Datan aggregointi

Datan aggregointi on erillinen järjestelmän päälle toteutettava toiminto, joka tämän projektin puitteissa on toteutettu aggregation.py-ohjelmatiedostolla. Se kerää sensorien lähettämää dataa ja tallentaa ne tiedostoiksi lokaalisti koneen muistiin ja influxDB-tietokantaan.

Aggregaatio-ohjelman yhteyteen voi myös lisätä erilaisia toimintoja liittyen kerätyn datan luonteeseen. Käyttötapausesimerkkimme käyttääkin aggregaatio-ohjelmaa juuri näin. Siitä enemmän tämän raportin luvussa 5.

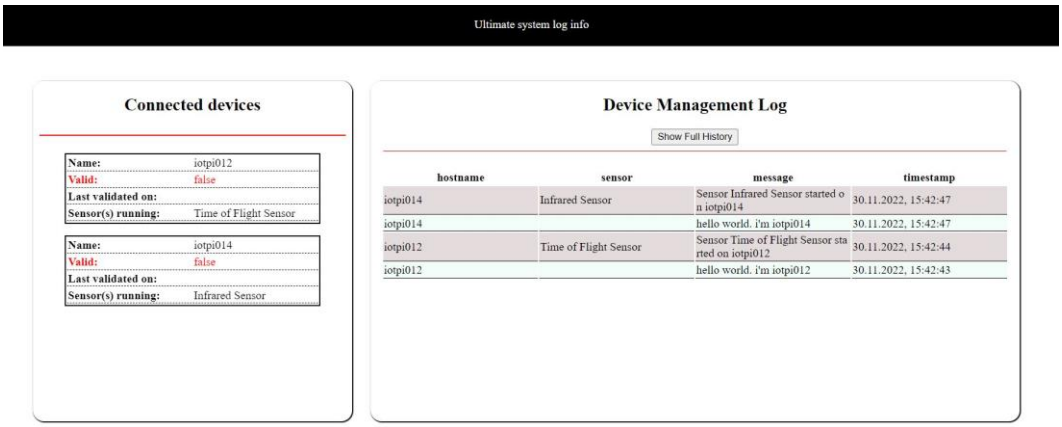
3 Hallinnointikäyttöliittymät

Toteutimme järjestelmälle kaksi erillistä käyttöliittymänäkymää. Toinen on hallinnointitietoja keräävä näkymä, joka kuuntelee MQTT viestejä systeemin management-kanavalla ja näyttää tiedot siitä mitä kanavalla on tapahtunut. Toinen on

dataa keräävää näkymä, joka kuuntelee systeemin datakanavia, ja esittää systeemissä liikkuvan datan visuaalisessa muodossa.

3.1 Management log -käyttöliittymä

Hallinnointitietoja keräävä näkymä on toteutettu käyttäen HTML, CSS, Javascript, Pug, Websocket ja Node.js teknologioita. Näkymä-ohjelmisto koostuu Node.js-pohjaisesta palvelimesta, joka kuuntelee MQTT-viestejä ja lähettää tietoja websocketin ja Pugin avulla Javascript- ja HTML-pohjaiseen visuaaliseen näkymään.

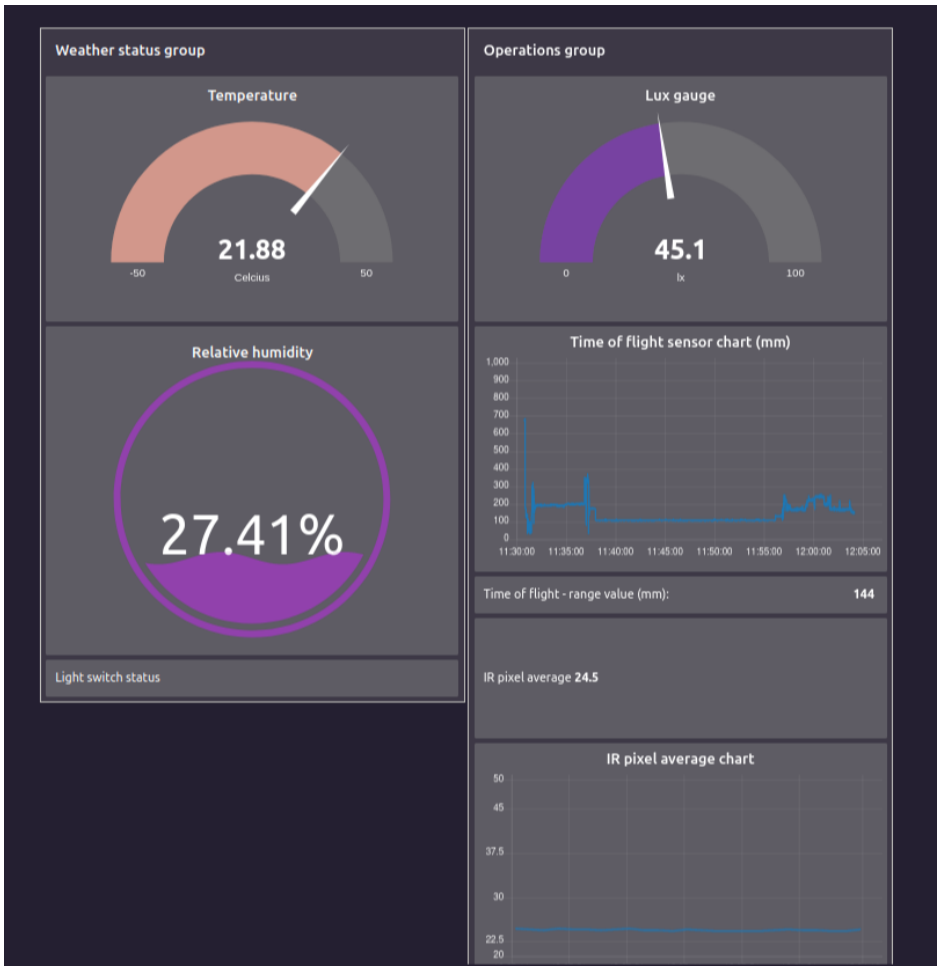


Kuva 4. Hallinnointitietoja keräävä käyttöliittymä.

Kuten kuvasta 4 näkyy, näkymän vasemmassa laidassa on kerätty tiedot kaikista järjestelmään kytketyistä laitteista. Laitteista kerrotaan nimi, onko laite validoitu, milloin laite on viimeksi validoitu, ja mitä sensoreita laitteessa on kiinni. Näkymässä oikealla on myös lista jokaisesta MQTT-viesteistä, joka kulkee management-kanavalla.

3.2 NodeRed-käyttöliittymä

NodeRed on käyttöliittymien tekemiseen tarkoitettu visuaalisen ohjelmoinnin kehitystyökalu. Toteuttamassamme NodeRed-käyttöliittymässä näkyy sensorien lähettämä mittausdata.



Kuva 5. NodeRed-käyttöliittymä

Kuvassa 5 näkyy käyttöliittymän ulkoasu. Näkymän vasemmassa laidassa näkyvät arvot, ilman lämpötila ja suhteellinen kosteus, ovat samasta sensorista. Näkymän oikealla puolella on kolmen eri laitteen sensorien lähettämät arvot. *Lux gauge* kertoo valoisuusasteen, *Time of flight* -sensori mittaa etäisyyttä, ja *IR pixel* arvo kertoo infrapunakameran pikselien näennäisten lämpötilojen keskiarvon.

4 Järjestelmän käyttöönotto

Järjestelmä otetaan käyttöön lataamalla GitHubista valmis paketti ja noudattamalla Intron alla annettuja ohjeita [6]. Asennuksessa on kolme erillistä osiota: 1) Device ja Sensor, 2) ManagementAttestor sekä 3) Management UI. Tässä dokumentissa olevat ohjeet on yksinkertaistettu ja käännetty suomeksi, parhaiten asennus onnistuu noudattamalla GitHubin ohjeita.

4.1 Device ja Sensor osioiden asentaminen ja ajaminen

1. Tarpeellisten tiedostojen asentaminen

    a) Mene terminaalissa tiedostopolkuun `/opt/`

    b) Kloonaa säilytyspaikka: `sudo git clone https://github.com/teemvil/iot.git`

    c) Mene tallennuskansioon: `iot/secure_sensor_management_system/install` ja käynnista asennusskripti: `sudo python3 install.py`

    d) Aktivoi laitteen devices.serice-tiedosto käyttämällä systemd:tä: `sudo systemctl enable iot.devices.service`

    e) Käynnistä laitteen devices.serice-tiedosto käyttämällä systemd:tä: `sudo systemctl start iot.devices.service`

2. Sensorin luominen ja toimeenpano

    a) Mene tiedostopolkuun: `/opt/iot/secure_sensor_management_system/` ja suorita komento: `sudo pip3 install .`

    b) Luo uusi sensorikansio tiedostopolkuun: `/opt/iot/secure_sensor_management_system/apps/sensors/`

c) Luo uusi sensoriskripti. Tärkeintä on periä BasicSensor SensorManagementLibrarysta:
`from SensorManagementLibrary.BasicSensor import BasicSensor`
Helpoin tapa on käyttää sensoripohjaa tiedostopolusta:
`opt/iot/secure_sensor_management_system/apps/sensors/ExampleSensor/`

d) Luo konfiguraatiotiedosto sensorille ja nimeä se: `sensor_config.json`. Tämä tiedosto tulee olla sensoriskriptin kanssa samassa kansiossa.

3. Aja uusi sensoritiedosto: `python3 YourNewSensor.py`
Saadaksesi sensorin käynnistymään laitteen käynnistyessä korvaa sensorin tiedostopolku käyttämään oman sensorisi polkua tiedostossa `iot.sensors.service` joka on kansiossa `/etc/systemd/system/`. Muuta tiedostossa olevaa riviä: `ExecStart=python3 /opt/iot/secure_sensor_management_system/apps/sensors/ExampleSensor/RNGSensor.py`
Aktivoi sensors.service-tiedosto käyttämällä systemd:tä:
`sudo systemctl enable iot.sensors.service`
Käynnistä sensors.service-tiedosto käyttämällä systemd:tä:
`sudo systemctl start iot.sensors.service`

4.2 ManagementAttestorin asennus ja käynnistäminen

a) Mene terminaalissa tiedostopolkuun `/opt/`

b) Kloonaa säilytyspaikka: `sudo git clone https://github.com/teemvil/iot.git`

c) Mene tallennuskansioon:
`iot/secure_sensor_management_system/install` ja käynnistä asennusskripti:
`sudo python3 install.py`

d) Mene kansioon tiedostopolussa `/opt/iot/secure_sensor_management_system/apps/ManagementAttestor/` ja lisää oikeat ip- ja porttiosoitteet tiedostoon `manager_config.json`.

e) Managerin voi käynnistää komennolla `sudo python3 Manager.py`

4.3 ManagementUI:n asennus ja käynnistäminen

a) Mene terminaalissa tiedostopolkuun `/opt/`

b) Kloonaa säilytyspaikka: `sudo git clone https://github.com/teemvil/iot.git`

c) Tarkista MQTT-asetukset tiedostopolun `/opt/iot/website/server` tiedostosta `server_config.json`. Varmista, että tässä tiedostossa on oikeat MQTT-tiedot.

d) Aja komento `npm install`.

e) Mene tiedostopolkuun `/opt/iot/website/` ja käynnistä palvelin komennolla `npm start`. Jos tämä ei toimi varmista, että NPM on asennettuna laitteelle. Sen voi asentaa komennolla `sudo apt install npm nodejs`.

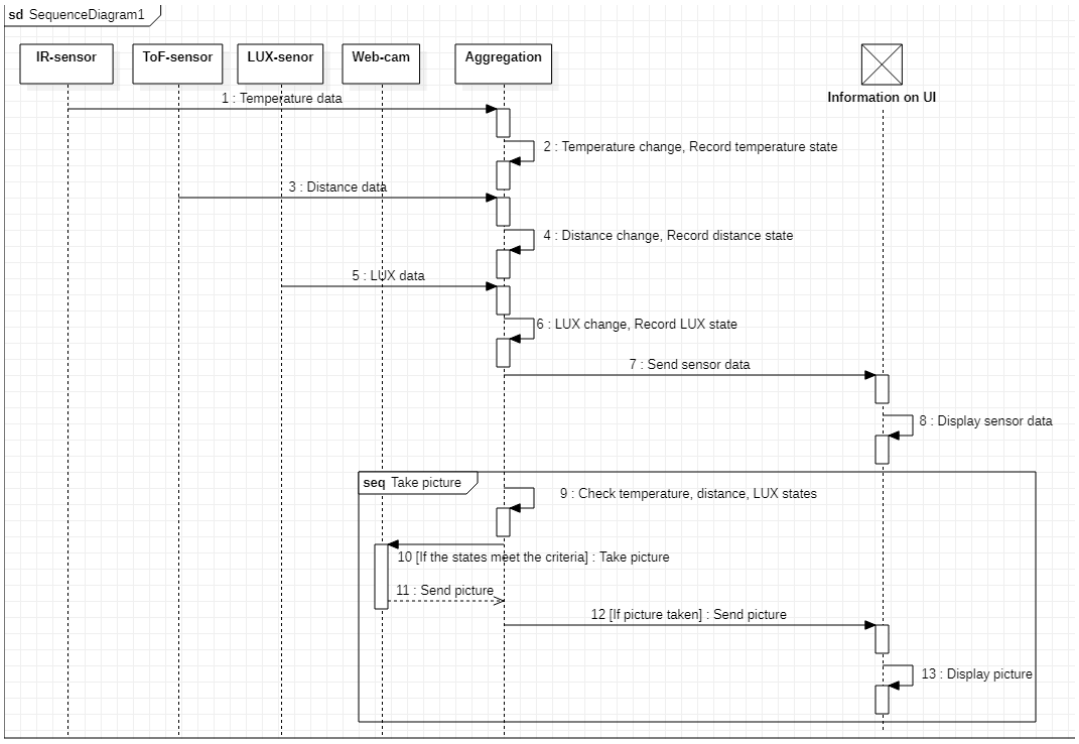
f) Jos käytät palvelinta laitteella paikallisesti, pääset ManagementUI:n osoitteesta: `http://localhost:3000`.
Jos palvelin on toisella laitteella: `http://<ip_address>:3000`.

5 Käyttötapaus – kuvan ottaminen

Projektissa tehty järjestelmä taipuu moneksi, ja mahdollisia käyttötapauksia on lukematon määrä. Käyttötapausesimerkki, jonka toteutimme järjestelmällä, on kuvan ottaminen webkameralla tiettyjen olosuhteiden vallitessa.

5.1 Toimintamekanismi

Kuvan ottamiselle on kolme kriteeriä: 1) huoneessa täytyy olla riittävän valoisaa, 2) infrapunasensorin täytyy tunnistaa riittävästi lämpöä tilassa, mikä viittaisi ihmisen tai eläimen läsnäoloon, ja 3) etäisyyssensorin on tunnistettava jonkin kiinteän massan olevan lähellä itseään.



Kuva 6. Käyttötapaus-sekvenssidiagrammi.

Kuvasta 6 voi nähdä miten mekanismi on rakennettu data-arvoja keräävän aggregaatio-ohjelman yhteyteen. Aggregaatio-tasolla kunkin sensorin arvoille on asetettu raja-arvo, jonka perusteella kriteerin tiedetään täyttyneen. Kun kaikki

kolme kriteeriä ovat täyttyneet, ohjelma lähettää kutsun webkameran REST-rajapinnalle kuvan ottamiseksi ja lopulta kuva ilmestyy näkyviin käyttöliittymässä.

5.2 Tietojen tallentaminen

Aggregaatio-ohjelma tallentaa sensorien lähettämää dataa csv-tiedostoihin siten, että se luo uuden tiedoston ensinnäkin aina kun ohjelma käynnistyy, ja aina kuvan ottamisen jälkeen. Täten periaatteessa jokaisen otetun kuvan yhteyteen on liitettävissä tiedostollinen kerättyä sensoridataa. Varsinainen kuvan ja sensoridatan yhteen pakettiin liittämisen toteuttaminen jäi kuitenkin jatkokehitysideaksi.

Sensorien lähettämää mittausdataa kerätään myös erilliseen InfluxDB-tietokantaan. Tietokanta sijaitsee samalla koneella MQTT-palvelimen kanssa.

Valitettavasti myös otetun kuvan tallentaminen jäi jatkokehitysvaiheeseen. Toinen jatkokehitysidea on metadatan lisääminen kuvaan ja sen tallentaminen lohkoketjuun, mikä mahdollisesti parantaisi autentikoinnin varmuutta entisestään. Itse kuva tallennettaisiin erilliseen tietokantaan.

6 Järjestelmän tuotantoversio LuxTurrim-ympäristössä

LuxTurrim on Nokian kampusalueella toimiva tuotantojärjestelmäympäristö, joka on kehitetty erilaisten älykaupunki-sovellusten testaamista varten [3]. Liitimme tuotantoversion ohjelmistokehyksestämme toimimaan tähän ympäristöön.

Käytimme tuotantoversiossamme palvelimena kannettavaa tietokonetta, johon oli asennettu AttestationEngine-palvelin ja sen käyttämä MongoDB-tietokanta. Kannettava liitettiin Nokian verkkoon. Käytössämme oli yksi Raspberry Pi –laite, johon oli kiinnitetty sensori. Koteloimme laitteen säänpitäväksi ja sijoitimme sen LuxTurrim-tolppaan Nokian kampusalueella. Käyttämämme sensori mittasi il-

man suhteellista kosteutta, lämpötilaa, ilmanpainetta, ilmanlaatua sekä korkeutta merenpinnasta. Erilliselle kannettavalle tietokoneelle asennettiin manager-ohjelma, attestation-ohjelma sekä käyttöliittymät.

Kaikki ohjelmistokehyksemme osat asetettiin kommunikoimaan LuxTurrim-ympäristön käyttämän MQTT-palvelimen kautta, minkä jälkeen järjestelmä testattiin käytössä. Testi oli onnistunut. Kaikki osat toimivat niin kuin niiden pitikin ja kommunikoivat keskenään onnistuneesti. Hallinnointikäyttöliittymä ilmoitti käynnistyvästä laitteesta ja sensorista, ja dataa keräävän käyttöliittymän näkymä ilmoitti sensorista tulevat mittausarvot.

7 Yhteenveto

Vaatimukset muuttuivat asiakkaan toimesta useaan otteeseen projektin aikana, mutta lopulta saimme valmiiksi suunnitellun ohjelmistokehyksen. Käyttötapausesimerkki jäi vielä pientä jatkokehitystä vaille. Tuotteen suunnittelun ja rakentamisen lisäksi saimme hyvää kokemusta siitä, miten asiakkaan kanssa tulee toimia, ja miten mahdolliset sudenkuopat voinee välttää jatkossa. Suunnittelutyö ei sujunut projektin aikana täysin ilman ongelmia, mutta virheitä täytyy tehdä, jotta jatkossa pystyy niistä jotakin oppimaan. Esimerkiksi keskusteluyhteyden ylläpitäminen koko projektin ajan on äärimmäisen tärkeää. Pienillä tarkentavilla kysymyksillä pystyy välttämään päivien turhan kehitystyön. Ketterät ohjelmistokehitysmenetelmät tulivat entistä tutummiksi, ja näitäkin taitoja pystyy jokainen meistä hyödyntämään seuraavissa projekteissa.

Lähteet

1 https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/

2 https://www.linux.fi/wiki/Systemd

3 https://www.luxturrim5g.com/pilot-environment

4 https://en.wikipedia.org/wiki/Trusted_Platform_Module

5 https://github.com/nokia/AttestationEngine

6 https://github.com/teemvil/iot