Opinnäytetyö (AMK)

Tieto -ja viestintätekniikka

2020

Mikael Jäppilä

RAJAPINNAN SUUNNITTELU JA TOTEUTUS IOT-LAITTEELLE

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK / YAMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto -ja viestintätekniikka

Kevät 2020 | 34 sivua

Mikael Jäppilä

RAJAPINNAN SUUNNITTELU JA TOTEUTUS IOT-LAITTEELLE

IoT-laitteiden määrän kasvaessa myös niille kehitettävien rajapintojen tarve kasvaa. IoT-laitteiden rajapinnoilla pyritään hakemaan IoT-laitteen keräämää dataa tai lähettämään dataa IoT-laitteelle. Onkin tärkeää osata suunnitella ja toteuttaa IoT-laitteelle oikeanlainen rajapinta.

Tässä työssä suunnitellaan ja toteutetaan toimiva rajapinta IoT-laitteelle. Toteutetun rajapinnan kautta pystytään hakemaan mikrokontrollerin keräämää dataa sen lämpötila- ja kosteussensorilta. Työn aikana saadaan selkeä kuva sopivimmista rajapintateknologioista rajapinnan kehitystä varten teknologiavertailujen perusteella.

Ennen rajapinnan toteuttamista rajapinnalle määriteltiin vaatimukset. Rajapinnan tulisi olla nopea, rajapinnalta palautuvan datan selkeässä ja helposti luettavassa muodossa, rajapinta tulee olla autentikoitu, jolloin ei-halutut henkilöt eivät pääse käsiksi rajapintaan ja helposti päivitettävissä uusilla päätepisteillä, kun mikrokontrolleriin halutaan lisätä uusia sensoreita keräämään dataa.

Vertailtujen teknologioiden ominaisuuksien ja vaatimusten perusteella toteutettiin toimiva rajapinta IoT-laitteelle. Toteutuksen teknologioiksi valikoitui vertailujen perusteella REST-arkkitehtuuri, JSON datamuoto ja FastAPI-rajapintakirjastoa. Toteutettu rajapinta testattiin ja todettiin rajapinnan toiminnallisuuksien ja ominaisuuksien vastaavuus määriteltyihin vaatimuksiin perustuen. Toteutettu rajapinta on nopea, data on selkeässä muodossa, rajapinta on autentikoitu OAuth2-protokollalla ja valittu rajapintakirjasto mahdollistaa rajapinnan päivittämisen helposti.

Toteutettua rajapintaa pystyy hyödyntämään verkko -ja mobiilisovelluksissa. Kehittävissä sovelluksissa pystyy näyttämään rajapinnalta kerättyä lämpötila- ja kosteusdataa reaaliaikaisesti. Käyttämällä aikaa eri teknologioiden vertailuun ja tutustumiseen saadaan toteutettua toimiva, vaatimuksia vastaava rajapinta nopeasti ja helposti.

ASIASANAT:

Rajapinta, API, esineiden internet, IoT, Python

BACHELOR´S THESIS

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and communications technology

Spring 2020 | 34 pages

Mikael Jäppilä

PLANNING AND IMPLEMENTATION OF APPLICATION PROGRAMMING INTERFACE FOR IOT DEVICE

While Internet of Things devices amount keeps growing also the amount of API's for IoT devices grows. APIs designed for IoT devices fetch data or push data to the IoT device. It is important to design and implement suitable API for the IoT device.

The objectives of this thesis were to plan and implement application programming interface for IoT device. With the implemented application programming interface users can retrieve temperature and humidity data from the IoT device. This thesis gives a clearer picture about the most suitable API technologies for the implementation based on the compared technologies.

The beginning of this study focuses on familiarizing different kinds of API architectures, API data formats and three Python programming language frameworks which are created for API implementation. These three frameworks are Django REST framework, CoAPthon and FastAPI. After familiarizing and comparing different options for the technologies used for the implementation of the API, the study set requirements for the API. The implemented API must be fast. Data should be in a clear format and easy to read. API must be authenticated so unwanted people could not have access to the API. API must be well documented, and API must be updatable if new sensors are added to the IoT device.

Based on the requirements of the API and the compared architectures, data formats and python frameworks, API was developed for the IoT device. Developed API was tested and confirmed that the set requirements for the API were met. The chosen technologies for the implementation are REST architecture, JSON data format and FastAPI framework. Implemented API was tested and meeting the API's functionalities and requirements was established. The API is fast, easy to understand and use, it is authenticated with OAuth2 protocol and FastAPI framework enables updatability.

Implemented API can be used in web or mobile applications. Developed applications can display requested data from the API in real-time. Temperature and humidity data can be viewed in real-time. Using time to compare different technologies and familiarizing them, implementing suitable API for the IoT device is fast and easy.

KEYWORDS:

Application programming interface, API, Internet of Things, IoT, Python

SISÄLTÖ

KUVAT

OHJELMAT

TAULUKOT

KÄYTETYT LYHENTEET SEKÄ SANASTO

DRF	Django REST Framework. Rajapintakirjasto rajapintojen luomiseen Python-ohjelmointikielellä.
HTTP	Hypertext transfer protocol on hypertekstin siirtoprotokolla, jota käytetään tiedonsiirtoon selaimissa ja palvelimilla.
HTTPS	HTTPS on laajennus HTTP-protokollasta. Sitä käytetään turvalliseen tiedonsiirtoon selaimissa ja palvelimilla.
SSL	Salausprotokolla, jonka avulla voidaan tehdä sovellusten kommunikoinnista IP-verkkojen yli turvallista.
IoT	Internet of Things eli esineiden internet.

1 JOHDANTO

Tämän työn tarkoituksena on suunnitella ja toteuttaa rajapinta IoT-laitteelle. Rajapinnan kautta pitää pystyä hakemaan IoT-laitteen keräämää sensoridataa. Kerättyä dataa pitää pystyä jatkokäyttämään muissa sovelluksissa.

IoT-laitteiden määrä kasvaa koko ajan. Määrän kasvaessa niille suunniteltavien rajapintojen tarve kasvaa. Onkin tärkeää, että pystytään toteuttamaan IoT-laitteelle sen tarpeiden mukainen rajapinta. Vertailemalla eri teknologioita ja tekemällä oikeat teknologiset valinnat saadaan toteutettua tarpeiden mukainen rajapinta.

Työssä käsitellään rajapinnan suunnittelu ja toteutus olemassa olevalle IoT-laitteelle. Ensiksi tutustutaan REST-, SOAP- ja GraphQL rajapinta-arkkitehtuureihin, JSON, XML- ja CBOR-datamuotoihin ja Django REST Framework-, CoAPthon- ja FastAPI-rajapintakirjastoon ja rajapinnan autentikointimahdollisuuksiin. Teknologioiden ominaisuuksiin tutustumisen jälkeen rajapinnalle määritetään vaatimukset. Määriteltyjen vaatimusten perusteella verrataan teknologioita toisiinsa ja valitaan niistä parhaat rajapinnan toteutusta varten.

Työn tavoitteena on tutustua rajapinta-arkkitehtuureihin, rajapinnan datamuotoihin, rajapintakirjastoihin ja toteuttaa toimiva rajapinta, jonka avulla pystytään hakemaan sensorien dataa IoT-laitteelta. Rajapinta toteutetaan kehittämistä varten valituilla teknologioilla ja kehityksessä pyritään seuraamaan hyvän rajapinnan piirteitä. Toteutettu rajapinta testataan ja tarkistetaan, että rajapinta vastaa suunnittelua ja määriteltyjä vaatimuksia.

2 TEKNOLOGIAT

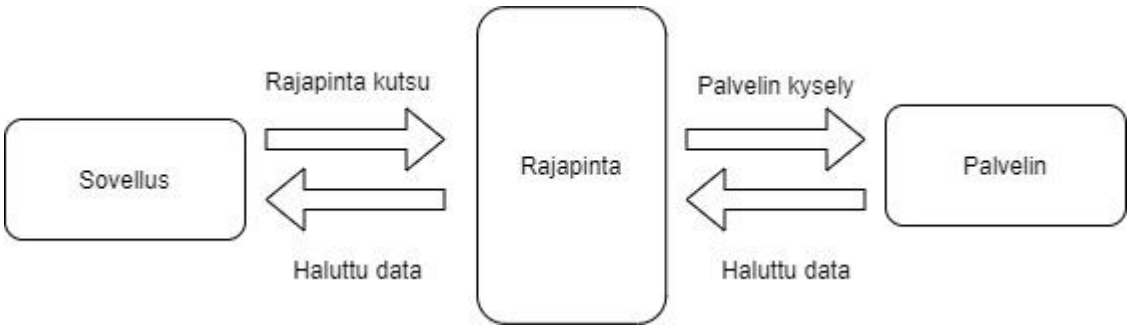
2.1 Ohjelmointirajapinta (API)

Ohjelmointirajapinta on määritelmä, miten järjestelmä välittää tietoa tai palveluita muille sovelluksille. Ohjelmointirajapinta voi olla datarajapinta tai toiminnallinen rajapinta. Datarajapinnan avulla pystytään lukemaan palvelun sisältämä data toisiin palveluihin. Toiminnallisessa rajapinnassa rajapinta tarjoaa datan muutosmahdollisuuksia, kuten datan laskenta-algoritmeja tai mahdollisuutta muuttaa järjestelmän dataa rajapinnan kautta. (avoinrajapinta.fi 2014)

Ohjelmistorajapinnat voivat olla yksityisiä tai avoimia rajapintoja. Avoimella rajapinnalla tarkoitetaan rajapintaa, jota saa vapaasti käyttää omiin tarkoituksiin ilman rajapinnan tekijän lupaa. Rajapinnan dokumentaatiot ja ominaisuudet täytyvät olla myös vapaasti saatavilla rajapinnan käyttäjälle. (avoinrajapinta.fi 2014) Hyvä esimerkki avoimesta rajapinnasta on YouTuben tarjoama rajapinta, jonka avulla käyttäjä pystyy tuomaan YouTuben ominaisuuksia omalle verkkosivulleen.

Rajapinta on suljettu eli yksityinen rajapinta, kun siitä ei ole vapaasti tietoa saatavilla ja sen käyttöä rajoitetaan. Yleisesti suljetut rajapinnat ovat organisaatioiden omia rajapintoja, joita käytetään yrityksen sisäiseen kehitykseen. (Rouse 2019)

Yleisesti rajapinnat toimivat sovelluksen ja palvelimen välillä. Käyttäjä tekee sovelluksessa rajapintakutsun, jonka perusteella rajapinta tekee kyselyn palvelimelle. Palvelin palauttaa rajapinnalle halutun informaation. Sovelluksen vastaanottaessa rajapinnan hakeman informaation palvelimelta, se näyttää haetun informaation käyttäjäluettavassa muodossa sovelluksessa (Kuva 1). (Park 2020)



Kuva 1 Ohjelmistorajapinnan toiminta.

2.2 REST-arkkitehtuuri

REST eli Representational State Transfer on arkkitehtuurinen malli sovellusten väliseen kommunikointiin. Se on yleisimmin käytetty rajapintamalli internetin välityksellä kommunikoiville laitteille. REST arkkitehtuurin hyötynä on se, että sillä on universaalit arkkitehtuurilliset vaatimukset, joka mahdollistaa usean ohjelmointikielen käyttämisen saman REST-rajapinnan kanssa. (Digia 2016)

REST nojaa tilattomaan kommunikointi protokollaan. Tilattomalla kommunikoinnilla tarkoitetaan kommunikointia, jossa lähetetään pyyntö palvelimelle ja palvelin antaa vastauksen nykyisen tilan mukaisesti. Esimerkiksi HTTP-protokolla. REST käyttää hyödykseen HTTP-protokollaa, jonka avulla sovellukset kommunikoivat internetin välityksellä keskenään. HTTP-protokollaan kuuluvilla metodeilla pystytään hakemaan, lisäämään, poistamaan ja päivittämään dataa (Taulukko 1). (IoTForAll 2018)

Taulukko 1 HTTP-protokollan metodit.

HTTP-metodi	Kuvaus	Esimerkki
GET	Hae dataa.	Hakee kaikkien käyttäjien tiedot.
POST	Lisää uutta dataa.	Lisää uuden käyttäjän.
PUT	Päivitä olemassa oleva data.	Päivittää vanhan käyttäjän tiedot.
DELETE	Poista dataa.	Poistaa halutun käyttäjän.

Rajapinnan datamuotona REST pystyy käyttämään JSON-, YAML-, XML- tai mitä tahansa muuta koneluettavissa olevaa dataa. Kuitenkin useimmin datamuotona käytetään JSON eli JavaScript Object Notation, sen helpon luettavuuden ja keveyden takia (Kuva 2). (SoapUI)

```
{
    "books": [
        {"author": "Matti Meikäläinen","title": "Testi kirja"},
        {"author": "Mari Testi","title": "Kirjan testinimi"}
    ]
}
```

Kuva 2 Esimerkki kirjojen datasta JSON-muodossa.

2.3 SOAP-arkkitehtuuri

SOAP eli Simple Objects Access Protocol on kommunikointi protokolla, joka alun perin suunniteltiin Microsoftille. Nykyään sitä käytetään verkkosovelluksien kommunikointi välineenä. SOAP, kuten REST-arkkitehtuuri, käyttää datan välittämiseen HTTP-protokollaa. Tämän lisäksi SOAP pystyy käyttämään mm. TCP, SMTP ja FTP. (altexsoft 2019)

Verrattuna REST-arkkitehtuuriin, SOAP pystyy käyttämään vain XML-datamuotoa. Tämän ansiosta viestirakenteet ja koodaus säännöt ovat vahvasti standardisoitu. Huonopuoli on se, että yleisesti viesteistä tulee todella pitkiä sulkumerkkien takia verrattuna JSON-datamuotoon (Kuva 3). (altexsoft 2019)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<books>
    <books>
        <book>
            <author>Matti Meikäläinen</author>
            <title>Testi kirja</title>
        </book>
        <book>
            <author>Mari Testi</author>
            <title>Kirjan testinimi</title>
        </book>
    </books>
</books>
```

Kuva 3 Esimerkki kirjojen datasta XML-muodossa.

2.4 GraphQL-arkkitehtuuri

GraphQL on Facebookin vuonna 2012 sisäisesti kehittämä datan kyselykieli ja määrittelymuoto. Sen kehittämistarkoituksena oli vähentää datasiirron määrää ja tehdä rajapinnoista nopeita, mukautuvia ja kehittäjäystävällisiä verrattuna sen aikaisiin REST-pohjaisiin arkkitehtuureihin. Vuonna 2015 GraphQL-arkkitehtuurista julkaistiin avoin lähdekoodi. (redhat.com)

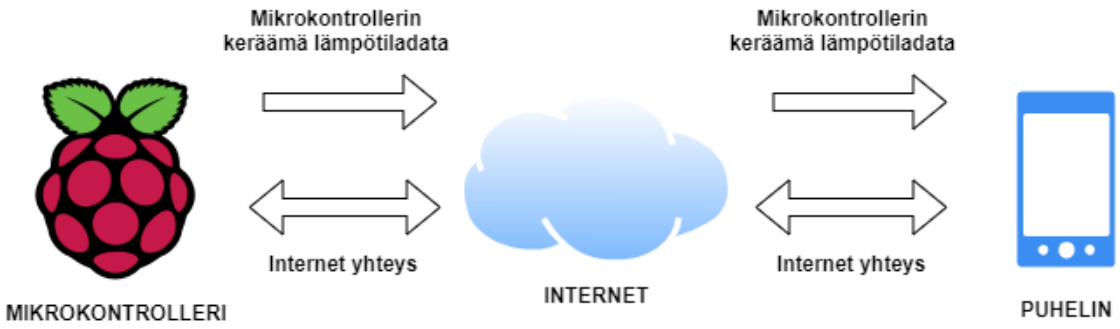
GraphQL-arkkitehtuurin avulla pystytään hakemaan vain haluttu data rajapinnasta, eikä mitään muuta. Kun kysely tehdään GraphQL validoi kyselyn luodun skeeman mukaisesti. Skeema koostuu tyyppi objekteista, jotka määrittelevät minkälaisen dataobjektin

pystyt vastaanottamaan ja mitä kenttiä dataobjekti sisältää. Onnistuneesti validoitu kysely suoritetaan ja käyttäjälle palautetaan kyselyllä haettu data. Toisin, kuin REST-arkkitehtuuri, GraphQL ei tarjoa käyttäjälle informaatiota siitä, onko rajapintakutsu onnistunut vai ei. GraphQL palauttaa aina HTTP-koodin 200, että kaikki on ok, siitä huolimatta oliko rajapintakutsu onnistunut vai ei. (redhat.com)

2.5 Esineiden internet

Esineiden internet tarkoittaa alustaa, jonka välillä internetiin yhdistetyt laitteet tai ihmiset ja laitteet kommunikoivat keskenään. Melkein mikä vain elektroniikkalaite voi olla osana esineiden internettiä. Nykyään suuri osa elektronisista laitteista pystyy yhdistämään internettiin ja kontrolloimaan sen toimintaa internetyhteyden välityksellä. Kotona valojen laittaminen päälle tai pois mobiilisovelluksen kautta on mahdollista esineiden internetin ansiosta. Esineiden internet on mahdollistanut teollisuudessakin tärkeiden komponenttien ja sensorien keräämän datan välittämisen helposti ja reaaliaikaisesti. (ibm.com 2016)

Laitteet, joissa on sisäänrakennettuja sensoreita, yhdistetään esineiden internet alustaan. Alustan kautta laitteet pystyvät kommunikoimaan keskenään. Kerätty data laitteella analysoidaan ja käsitellään, jonka jälkeen kerätty data jaetaan pilven kautta muille laitteille käytettäväksi. (ibm.com 2016) Hyvä esimerkki IoT-laitteesta on mikrokontrolleri, joka on yhdistettynä internetiin. Mikrokontrolleriin liitetty lämpötilasensori kerää lämpötiladataa, jonka mikrokontrolleri prosessoi ja jakaa reaaliaikaisesti mobiilisovellukselle internetin välityksellä. Mobiilisovellus näyttää lämpötiladatan käyttäjälleen (Kuva 4).



Kuva 4 IoT-laitteen kommunikointi.

2.6 Rajapinnan datamuodot

Rajapinnan datatyypin valinta on tärkeä osa suunnitteluvaihetta. Datamuoto määrittelee, miten rajapinta käsittelee datan generoinnin ja dataan liittyvät pyynnöt rajapinnalta. Väärän datatyypin valitseminen rajapintaa varten saattaa tehdä muuten hyvin kehitetystä ja implementoidusta rajapinnasta käyttökelvottoman. Panostettaessa oikean datatyypin valintaan varmistetaan rajapinnan tehokkuus, skaalautuvuus ja pitkä ikäisyys. (nordicapis.com)

Suoran datan formaatti (Direct Data Format) koostuu rajapinnan datatyypeistä, jotka on suunniteltu käsittelemään dataa suoraan kahden sovelluksen tai koneen välillä. Data on koneluettavissa, jolloin se on myös kompaktissa muodossa. Loistava valinta sovelluksien väliselle kommunikoinnille. (nordicapis.com)

Tässä toteutuksessa keskitytään suoran datan dataformaatteihin, koska se mahdollistaa rajapinnan tehokkaan datan kulun ja rajapinnan hyödyntämisen muissa sovelluksissa.

2.6.1 JSON-datamuoto

JSON eli JavaScript Object Notation on kevyt datamuoto, jota on helppo lukea ja kirjoittaa. Moni ohjelmisto tukee JSON-datamuotoa, sitä on helppoa generoida ja jäsentää. (json.org)

JSON koostuu kahdesta rakenteesta. Kokoelmasta nimi ja arvo pareja eli objekteista ja järjestetystä listasta arvoja. Nimi ja arvo pari koostuu kentän nimestä, joka on lainausmerkkien sisällä. Kentän nimen jälkeen tulee kaksoispiste, jota seuraa kentän arvo. Yleisesti kaikki modernit ohjelmointikielet ymmärtävät JSON-datamuotoa, koska sen datarakenne perustuu universaaliin datarakenteeseen. (json.org)

JSON-datamuoto antaa käyttäjän määritellä kenttänimen arvon datatyypin. Pystytään siis erottelemaan minkä tyyppistä dataa kukin nimikenttä voi sisältää. Datatyypit rajoittuvat JavaScript kielen datatyyppeihin eli kokonaislukuun, numeroon, merkkijonoon tai totuusarvomuuttujaan. Tämän ansiosta käyttäjä pystyy ennakoimaan minkä tyyppistä dataa nimikenttä palauttaa. Numerokentälle pystyy määrittelemään maksimi -tai minimiarvon. (Kuva 5) Ohjelmassa on määritetty oppilaan datan sisältämät kentät ja kenttien datatyypit. Pystytään myös määrittelemään mitkä kentät ovat pakollisia. (nordicapis.com)

Nimikenttien datatyypit ja nimikentän kuvaelma dokumentoidaan JSON-skeemaan. JSON-skeeman avulla pystytään validoimaan, onko JSON-tiedoston rakenne oikein. (json-schema.org)

```
{
    "title": "student",
    "type": "object",
    "properties": {
        "first_name": {
            "type":"string",
            "description": "First name of the student."
        },
        "age": {
            "type": "integer",
            "minimum": 0
        },
        "address":
        {
            "type": "string"
        }
    },
    "required": ["first_name", "address"]

}
```

Kuva 5 Esimerkki JSON-skeemasta, joka määrittelee oppilaan datan kentät.

JSON on erittäin kevyt datamuoto sen rakenteen ja syntaksin takia. Keveyden ansiosta rajapinnan kautta lähtevät kutsut ja palautuvat vastaukset tulevat erittäin nopeasti. Sitä suositaankin usein rajapintojen datamuotona. Myöskin kone -ja ihmisluettavuus on vaikuttanut sen suosioon. JSON sopii parhaiten datan välittämiseen sovelluksien välillä. (nordicapis.com)

2.6.2 XML-datamuoto

XML eli Extensible Markup Language on yksinkertainen ja skaalautuva tekstiformaatti. Alun perin se suunniteltiin ratkaisemaan laajamittaiset elektroniset julkaisut. Nykyään sitä käytetään myös laajalti internetin välityksellä tapahtuvan datan vaihdon datamuotona. Esimerkiksi rajapintojen datamuotona. (w3.org)

XML-datamuodon rakenne muistuttaa vahvasti HTML-ohjelmointi kieltä. Se koostuu yhdestä juurielementistä, joka pitää sisällään juurielementtiin liittyvät lapsielementit. Kuvassa 6 *book* on tiedoston juurielementti, joka sisältää juurielementtiin liittyvän datan lapsielementteinä *author* ja *title*.

```xml
<book>
    <author>J.R.R Tolkien</author>
    <title>Taru Sormusten Herrasta</title>
    <pages>1120</pages>
</book>
```

Kuva 6 XML-rakenne.

Kuten JSON, XML-elementeille pystyy määrittelemään, mitä datatyyppiä ne sisältävät. XML-datamuodon vahvuus onkin, että siinä on enemmän jo valmiiksi rakennettuja datatyyppejä kuin JSON-datamuodossa. Sisäänrakennettuja datatyyppejä ovat merkkijono, desimaaliluku, kokonaisluku, totuusarvomuuttuja, päivämäärä ja kellonaika. Sisään rakennettujen datatyyppien lisäksi käyttäjä pystyy määrittelemään omia datatyyppejä. Määrittelemällä elementin datatyypin pystytään validoimaan, että onko XML-tiedosto muodostettu oikein oikeilla datatyypeillä (Kuva 7). (w3schools.com)

```xml
<xs:element name="author" type="xs:string"/>
<xs:element name="title" type="xs:integer"/>
<xs:element name="pages" type="xs:integer"/>
```

Kuva 7 XML-skeema.

Yleisesti XML on kooltaan suurempaa kuin JSON, jolloin datan välittäminen XML-muodossa on aina hitaampaa kuin JSON-muodossa. Tämän takia datan prosessointi on myös hitaampaa ja sen serialisoiminen on paljon hitaampaa. XML sopiikin parhaiten dokumenttien merkintäkieleksi. Etenkin silloin, kun merkintäkieli ja metadata on tärkeässä osassa ja sitä ei voi olla jättämättä pois. (restfulapi.net)

2.6.3 CBOR-datamuoto

CBOR eli Concice Binary Object Presentation on datamuoto, jonka päätarkoitus on mahdollistaa pienet koodimäärät, pienet viestikoot ja skaalautuvuuden ilman versiointia. Sen rakenne perustuu vahvasti JSON-datamalliin eli objekteihin, jotka muodostuvat nimi ja arvo pareista. (cbor.io)

CBOR-datamuotoa on hyvä käyttää silloin, kun halutaan lähettää tai vastaanottaa dataa binäärimuodossa. Tällaisia tapauksia voivat olla salausavaimien, graafisendatan tai sensoridatan lähettäminen. Datan muuttaminen binäärimuotoon mahdollistaa datan nopean

liikuttamisen käyttäjäystävällisyyden kustannuksella, koska data ei ole enää ihmisluettavissa. (cbor.io)

Vaikka CBOR pohjautuu suurimmiltaan osin JSON-datamuotoon, sen kenttien datatyypit eivät rajoitu samoihin kenttiin, joita JSON käyttää. Näiden kenttien lisäksi CBOR tarjoaa ylimääräisiä datatyyppejä tämän lisäksi käyttäjä voi tehdä omia datatyyppejä datakentille. (cbor.io)

2.7 Python-ohjelmointikielen rajapintakirjastot

Tässä luvussa käydään läpi kolme pythonkirjastoa. Tutkitaan mitä hyviä ominaisuuksia kirjastot tarjoavat valmiina rajapintaa varten, kuten dataskeemoja, datanvalidointia, autentikointi vaihtoehtoja tai automaattista dokumentointia.

Python-kirjastot muodostuvat kokoelmasta funktioita, joiden avulla pystytään tekemään toimintoja itse kirjoittamatta koodia funktioille. Niin sanotusti käytetään hyödyksi jo jonkun kehittämää funktionaalisuutta omassa sovelluskehityksessä. Kirjastot tuovat hyödyllisiä funktioita kehitettävän sovelluksen käytettäväksi, kuten python-ohjelmointikielen Numpy-kirjasto. Kyseisen kirjaston tuomilla funktioilla pystyy toteuttamaan numeerisia operaatioita halutuille luvuille. Kuten matriisilaskentaa. (realpython.com)

2.7.1 Django REST framework (DRF)

Django REST framework on avoimeen lähdekoodiin perustuva Python-kirjasto, jonka avulla pystytään rakentamaan monimutkaisia rajapintoja. Sen joustavuus ja sisäänrakennetut ominaisuudet mahdollistavat rajapinnan toteuttamisen yksinkertaisesti ja nopeasti. Frameworkin vaatimukset ovat Python versio (2.6.5+) ja Django versio (1.30+). DRF skaalautuvuuden ja kustomointi mahdollisuuksien ansiosta rajapinnan kehitysaika vähenee huomattavasti. (django-rest-framework.org)

Verrattuna muihin Python rajapinta kirjastoihin DRF ei generoi rajapinnan osoitepolkuja automaattisesti vaan antaa käyttäjän itse määritellä osoitepolut. Tämä mahdollistaa rajapinnan osoitepolkujen joustavuuden. Sen lisäksi DRF tarjoaa sisäänrakennetun rajapinnan osoitepolun testauskäyttöliittymän suoraan selaimeen, jonka avulla pystytään tekemään kutsuja rajapinnan osoitepolkuihin ja heti näkemään minkälaista dataa rajapinta palauttaa. (quintagroup.com)

Suoraan pakasta vedettynä DRF tarjoaa loistavia valmiita ominaisuuksia hyvän rajapinnan kehittämiseen. Rajapinnassa pystyy käyttämään hyödyksi funktionaalista näkymää, joka vastaanottaa rajapintakutsun eli *request* ja palauttaa vastauksen kutsuun perustuen eli *response*. Tämän lisäksi voi käyttää kirjastolle suunniteltuja vahvempia näkymiä. Rajapinta tukee jo valmiina monia eri datatyyppejä, suosituimpina JSON ja XML. Kirjasto tarjoaa käyttäjän tarpeisiin perustuen eri vaihtoehtoja rajapinnan autentikointia varten. Rajapinnan pystyy helposti autentikoimaan HTTP-perusautentikoinnilla, rajapinta-avaimella tai vahvalla suojauksella eli OAuth1:lla tai OAuth2:lla. Rajapintaan tulevien kutsujen määrää pystyy myös helposti rajoittamaan. Tukee valmiiksi CRUD operaatioita. (django-rest-framework.org)

DRF kuuluvan serialisointimoottorin avulla Pythonin datamallit ja datakyselyt pystytään muuttamaan Pythonin natiiveiksi datatyypeiksi, jotka tämän jälkeen voidaan muuttaa joko JSON-, XML- tai johonkin toiseen datamuotoon. Serialisointi mahdollistaa deserialisonnin, jonka avulla käyttäjän lähettämä data rajapinnalle pystytään validoimaan eli varmistamaan oikea datatyyppi ja datamuoto. (django-rest-framework.org)

2.7.2 CoAPthon-rajapintakirjasto

Python-ohjelmointikielelle löytyy oma kirjasto CoAP-rajapinnan kehittämistä varten, CoAPthon. Dokumentaatio löytyy riittävästi, jonka avulla rajapinnan pystyy kehittämään. Verrattaessa Djangon REST kirjastoon se ei tarjoa läheskään niin paljon valmiita ominaisuuksia. CoaPThon käyttää hyväkseen CoAP-protokollaa. (coap.technology)

Constrained Appliaction Protocol eli lyhyesti CoAP on protokolla, joka erikoistuu internetin välityksellä tapahtuvaan datansiirtoon. Se on suunniteltu toimimaan kommunikaatio välineenä kahden laitteen välillä (M2M, machine to machine). Protokollan suunnittelussa on etenkin otettu huomioon pienet mikrokontrollerit, koska toimiakseen se tarvitsee vähäiset 10kibittiä keskusmuistia. CoAP käyttää hyväkseen REST-arkkitehtuuria. (coap.technology)

CoAP on suunniteltu käyttämään mahdollisimman vähän resursseja laitteen ja verkon puolelta katsottuna toimiakseen. Sen lähettämä data koodataan tiiviiseen muotoon, jolloin viestien koko saadaan mahdollisimman pieneksi. Tämä myös mahdollistaa sen, ettei linkki laitteiden välillä pirstaloidu. Datan välittäminen CoAPin avulla on erittäin turvallista,

sillä sen valitsema turvallisuusprotokolla on vastaava kuin 3072-bittisissä RSA-avaimissa. (coap.technology)

2.7.3 FastAPI-rajapintakirjasto

FastAPI on Python-kirjasto, joka on suunniteltu rajapintojen kehittämistä varten. Se tukee REST-rajapintamallia ja GraphQL-arkkitehtuuria. Kirjaston tarkoituksena onkin mahdollistaa modernien ja erittäin suorituskykyisten rajapintojen kehittäminen helposti ja vaivattomasti. Se on rakennettu Starlette- ja Pydantic-kirjastoja hyväksi käyttäen. Toimiakseen kirjasto vaatii vähintään Python 3.6 version. (fastapi.tiangolo.com)

FastAPI-kirjaston standardit perustuvat OpenAPI- ja JSON Schema-teknologioihin. OpenAPI-teknologian avulla kehitetään itse rajapinta ja data mallinnetaan automaattisesti JSON Schema-teknologian avulla. Kuten Djangossa FastAPI tarjoaa interaktiivisen rajapinnan dokumentoinnin ja testauskäyttöliittymän suoraan selaimeen. Kehittäjä voi itse valita haluamansa teknologian käyttöliittymää varten, kuten Swagger UI:n. Rajapintateknologia pystyy validoimaan Python-ohjelmointikielen datatyyppejä ja Pydantic-kirjaston tukemia datatyyppejä, koska se hoitaa datan validoinnin. Pydantic mahdollistaa myös omien datatyyppien määrittämisen. Rajapinnan autentikointiin voi käyttää HTTP-perusautentikointia, OAuth2 tai rajapinta-avaimia. (fastapi.tiangolo.com)

Rajapinta on erinomaisesti dokumentoitu valmiilla esimerkeillä, joiden perusteella pystyy kehittämään oman rajapinnan todella nopeasti. FastAPI on helppo opetella käyttämään hyvän dokumentoinnin ansiosta. (fastapi.tiangolo.com)

2.8 Rajapinnan autentikointi

Rajapinnan autentikoinnilla pyritään tekemään rajapinnan välittämästä datasta turvallista. Autentikoinnin avulla ei-halutut käyttäjät eivät pääse rajapinnan kautta IoT-laitteeseen käsiksi, injektoimaan vahingollista dataa. Autentikoinnilla pystyy helposti rajoittamaan, kuinka moni pystyy käyttämään rajapintaa, jolloin rajapinta ei ylikuormitu käyttäjistä. Kolme yleisintä tapaa autentikoida rajapinta ovat HTTP-perusautentikointi, rajapinta-avain ja OAuth. (nordicsapi.com)

2.8.1 HTTP-perusautentikointi

Tässä toteutuksessa HTTP-käyttäjä antaa käyttäjänimen ja salasanan, joka todistaa käyttäjän oikeaksi. Autentikointi käyttää HTTP-otsikkoa itsessään, jolloin monimutkaisia internet vastauksia ei tarvita. HTTP-perusautentikoinnin ongelma piilee siinä, jos autentikointi on tehty turvatonta internetyhteyttä pitkin, ilman SSL-protokollaa. Silloin käyttäjä voi napata kirjautumisinformaation ja autentikoida itsensä kopiolla HTTP-otsikosta, joka sisältää vahingollista dataa. Tämän pystyy välttämään SSL-protokollan avulla, mutta silloin kärsii vasteaika. HTTP-perusautentikointia on hyvä käyttää sisäisissä verkoissa ja etenkin IoT-ratkaisuissa, jolloin nopeus on tärkeässä osassa ja autentikoinnin turvallisuudesta voidaan tinkiä. (nordicsapi.com)

2.8.2 Rajapinta-avain

Uniikki rajapinta-avain generoidaan, jokaiselle uudelle rajapintakäyttäjälle. Kun käyttäjä yrittää tehdä uuden kutsun rajapintaan, järjestelmä käyttää avainta tunnistaakseen käyttäjän. Tämä on todella nopea tapa tunnistaa oikeat käyttäjät ja antaa käyttölupa rajapinnalle. Tätä tapaa onkin käytetty monia vuosia peruskäytäntönä rajapinnoissa. Rajapinta-avain kulkee kuten mikä tahansa HTTP-kutsu, jolloin se on erittäin helppoa poimia kuin mitkä tahansa muut HTTP-kutsut. Rajapinta-avain on toimiva ratkaisu silloin, kun ne ovat hyvin suojattu varmennusjärjestelmällä. (nordicsapi.com)

2.8.3 OAuth2-autentikointi

OAuth2 käyttää autentikointia ja varmennusjärjestelmää. Käyttäjän kirjautuessa järjestelmään järjestelmä palauttaa käyttäjälle poletin. Sen jälkeen käyttäjä tekee pyynnön autentikointipalvelimelle, joka hylkää tai hyväksyy autentikoinnin. Tämän jälkeen annettu poletti yhdistetään käyttäjään. Käyttäjän tehdessä pyyntöjä rajapintaan käyttäjään yhdistetty merkki tarkistetaan, jonka perusteella pyyntö tehdään tai jätetään tekemättä. Fundamentaalisesti toteutus on paljon turvallisempi ja vahvempi kuin aikaisemmin mainitut HTTP-perusautentikointi ja rajapinta-avain. (nordicsapi.com)

3 RAJAPINNAN SUUNNITTELU JA TEKNOLOGISET VALINNAT

Tässä luvussa käydään läpi, mitä rajapinnan suunnittelussa tulee ottaa huomioon. Suunnitteluvaiheessa ilmenneet rajapinnan vaatimukset vaikuttavat rajapintateknologian valintaan ja rajapinnan ominaisuuksiin. Vaiheen jälkeen on selkeä kuva siitä minkälainen rajapinta tulisi rakentaa laitteelle.

3.1 Rajapinnan vaatimukset

Toteutettavan rajapinnan täytyy olla yksinkertainen ja mahdollisimman helppo ymmärtää, ei kohtuutonta monimutkaisuutta. Rajapinnan päätepisteet tulee olla helposti ymmärrettävissä ja rajapinnalta palautuva data selkeässä muodossa. Tämä mahdollistaa rajapinnan helpon ymmärrettävyyden ja dataa pystyy helposti jatkokäyttämään. Tämä mahdollistaa rajapinnan helpon käytettävyyden kehitettävissä sovelluksissa. (dzone.com)

Rajapinnan tulee olla helposti ylläpidettävissä ja päivitettävissä. Tulevaisuudessa voidaan haluta lisätä uusia päätepisteitä perustuen uusiin haluttuihin toimintoihin. Toteutetun rajapinnan toiminnot eivät siis saa rajoittaa mahdollisia tulevaisuudessa kehitettäviä toimintoja tai tulevaisuudessa tehtävät toiminnot eivät saa rajoittaa jo olemassa olevia toimintoja. (dzone.com)

Rajapinnan käyttö pitää pystyä rajoittamaan. Käyttöä pystytään rajoittamaan autentikoimalla rajapinta, jolloin vain tietyt käyttäjät pystyvät tekemään pyyntöjä rajapintaan. Kyselyiden määrä tietyn ajanjakson aikana tulisi olla määritelty, jolloin rajapinta ei kuormittuisi liikaa, joka aiheuttaisi viiveitä datan lähettämiseen ja vastaanottamiseen. Pahimmassa tapauksessa palvelin, joka ylläpitää rajapintaa voisi kaatua. Rajoittamalla rajapinnan käyttöä pystytään välttämään myös palvelunestohyökkäykset, koska rajapintaa ei pysty kuormittamaan valtavalla määrällä kyselyitä samanaikaisesti. (Elastic.io)

Rajapinnalta tulee pystyä hakemaan dataa mahdollisimman reaaliaikaisesti. Kehitetty rajapinta pyrkii seuraamaan hyvän rajapinnan piirteitä. Hyvän rajanpinnan piirteet ovat seuraavat:

- Dataa pystyy hakemaan tietyin kriteerein.
- Käyttäjä pystyy vaikuttamaan haetun datan määrään.
- Haetun datan voi järjestää. Ainakin päivämäärän perusteella.
- Rajapinnan pitää tukea JSON-datamuotoa.
- Rajapinnan pitää olla autentikoitu.
- Rajapinta pitää olla hyvin dokumentoitu.

(Elastic.io)

3.2 Turvallinen rajapinta

Rajapinnan kehityksessä pyritään seuraamaan hyviä käytäntöjä turvallisen rajapinnan kehittämiseen. Yksityiskohdat turvallisen rajapinnan toteuttamiseen:

- Suojaa rajapinta rajapinnan tarpeiden mukaan. Ei monimutkaisia ratkaisuja.
- Käytä HTTPS protokollaa, jolloin kommunikointi on salattua.
- Salasanat on salattu, jonkin salausalgoritmin perusteella.
- Käyttäjänimet, salasanat, rajapinta-avaimet tai sessiopoletit eivät tulisi esiintyä rajapintapolussa suoraan.
- OAuth autentikoinnin käyttäminnen rajapinnassa.
- Aikaleimojen käyttö rajapintakutsuissa.
- Päätepisteparametrit tulee validoida.

(restfulapi.net)

3.3 Datamuodon valinta

IoT-laitteelle tehtävästä rajapinnasta halutaan tehdä mahdollisimman helppo käyttää eikä käytettävyyden pitäisi rajoittua rajapinnan datamuotoon. Sensoridataa lähetettäessä datan kulku on oltava nopeaa ja kevyttä. Näiden vaatimusten perusteella XML-datamuodon voi lukea pois vaihtoehdoista, koska tiedostokoot kasvavat suuriksi, jolloin datan poimiminen tiedostosta on työlästä.

JSON- ja CBOR-datamuotoja verrattaessa molemmat on suunniteltu rakenteellisen datan vaihtoa varten. Data on pienessä muodossa, sitä on nopea lähettää ja datan saa helposti poimittua tiedostosta. CBOR-datamuotoa on käyttäjänäkökulmasta kuitenkin

vaativampaa käyttää, koska data on binäärimuodossa. Rajapinnan käyttäjän tulisikin käyttää parseria, joka pystyy muuttamaan datan binäärimuotoon ja binäärimuodossa olevan data ihmisluettavaan muotoon. Ohjelmointikielissä ei ole sisäänrakennettua parseria, jonka avulla tämän datamuutoksen pystyisi suorittamaan. Käyttäjän tulisikin ladata ohjelmointikirjasto, jonka CBOR tarkoitettua parseria pystyisi käyttämään. CBOR vaatii käyttäjältään paljon enemmän kuin JSON. JSON on paljon helpompi käyttää, sillä suurin osa ohjelmointikielistä pystyy suoraan ilman mitään kirjastoja pilkkomaan JSON-tiedostoja ja poimimaan haluttavan datan.

Edellä mainitun perusteella valitaan rajapinnan datamuodoksi JSON.

3.4 Käytettävä rajapintateknologia toteutuksessa

Toteutuksessa tullaan käyttämään REST-rajapintateknologiaa. Valittu teknologia täyttää kaikki toteutuksen edellä määritellyt vaatimukset. REST tukee JSON-datamuotoa, joka mahdollistaa datan nopean kulun. Se on helppo toteuttaa. Sopii hyvin laitteiden väliselle kommunikoinnille. Tukee monia autentikointi tapoja ja rajapinnalle tehtyjen pyyntöjen määrään pystyy vaikuttamaan. Käyttäjän on helppoa ymmärtää REST-rajapinnan toimintaa ja sitä, mitä dataa eri päätepisteet palauttavat selkeiden nimeämiskäytäntöjen ansiosta.

3.5 Rajapintakirjaston valinta

IoT-laitteen, kehitettävän rajapinnan vaatimusten ja suorituskyvyn perusteella rajapintakirjastoksi valitaan FastAPI. Se tarjoaa erittäin hyvät lähtökohdat rajapinnan kehittämiseen valmiiden ominaisuuksien ja hyvän dokumentaation ansiosta. FastAPI tarjoaa myös esimerkkejä, joita soveltamalla rajapinnan saa kehitettyä ilman suurempia ongelmia.

3.6 Autentikointitavan valinta

Rajapinta halutaan autentikoida mahdollisimman hyvin, jolloin ei-halutut käyttäjät eivät pääse käsiksi IoT-laitteeseen. OAuth2-autentikointitavan avulla saadaan määriteltyä käyttäjätunnuksen ja salasanan avulla, sekä poletin avulla, ketkä pystyvät käyttämään

rajapintaa. OAuth2 avulla pystytään myös määrittelemään aikaväli, jonka aikana käyttäjä pystyy tekemään kutsuja rajapintaan.

Valittu rajapintakirjasto FastAPI, tarjoaa hyvät edellytykset OAuth2-autentikointitavan käyttämiselle rajapinnassa. FastAPI sisältää valmiiksi luokan OAuth2-autentikointitavan luomiselle ja loistavan dokumentaation, miten OAuth2-autentikointitapa kannattaa luoda toteutettuun rajapintaa.

4 RAJAPINNAN TOTEUTUS

Opinnäytetyössä kehitettävä rajapinta IoT-laitteelle toteutetaan Python-ohjelmointikielellä. Apuna kehityksessä käytetään rajapintojen kehittämistä varten suunniteltua FastAPI-kirjastoa ja sen tuomia ominaisuuksia. Rajapinnan avulla käyttäjä saa haettua dataa IoT-laitteen lämpötila- ja kosteussensorilta. Työssä keskitytään vain rajapinnan kehittämiseen, IoT-laitteen toiminnallisuudet sensoridatan keräämiseen on jo valmiina. Kehitettyä rajapintaa pystyy hyödyntämään kommunikaatiovälineenä esimerkiksi mobiili- tai verkkosovelluksen ja IoT-laitteen välillä.

4.1 Kehitysympäristö

Rajapinta kehitettiin käyttäen Python-ohjelmointikielestä versiota 3.8.2. Rajapinnan lokaalia kehitystä varten pystytettiin kehityspalvelin omalle tietokoneelle Uvicornin avulla. Uvicorn on Python-verkkopalvelimia varten kehitetty kirjasto. FastAPI ja Uvicorn asennettiin kehitysympäristöön käyttäen Pythonin kirjastonasennusohjelmaa.

4.2 Rajapinnan luominen

Asennettu FastAPI on Python-luokka, joka sisältää kaikki tarvitsemat toiminnallisuudet rajapintaa varten. Jotta rajapinnan kehityksessä saadaan hyödynnettyä FastAPI kirjastoa se pitää ensin tuoda Python-tiedostoon. Kun kirjasto on tuotu, määritetään muuttuja ja sen arvoksi FastAPI-luokka, jolloin muuttuja toimii FastAPI-luokan ilmentymänä. Määritetyn muuttujan avulla pystytään käyttämään FastAPI:n toiminnallisuuksia. Ohjelmassa 1 ensimmäisellä rivillä on tuotu FastAPI-luokka importilla Python-tiedostoon, jotta sen toiminnallisuuksia pystytään käyttämään. *app* muuttuja toimii FastAPI-luokan ilmentymänä, jonka avulla luokan tarjoamia funktioita pystytään kutsumaan.

Ohjelma 1 FastAPI-luokan tuominen main.py-tiedostoon.

```python
from fastapi import FastAPI
from sensors import sensor_data

app = FastAPI()


@app.get("/sensors", tags=["Sensors"])
async def data_from_sensors():
    return sensor_data()
```

4.3 Päätepisteet

Rajapinnalle pyritään määrittämään mahdollisimman ymmärrettävät päätepisteet, jolloin jo pelkästään lukemalla päätepisteen käyttäjä saa tietoa siitä minkälainen rajapinnantoiminnallisuus on kyseessä (Taulukko 2).

Taulukko 2 Rajapinnan päätepisteet.

Päätepiste	**HTTP-metodi**	**Kuvaus**
/sensors	GET	Hakee kaikkien sensorien datan ja palauttaa JSON-muodossa käyttäjälle.
/sensors/{sensor_name}	GET	Hakee määritetyn päätepistemuuttujan (sensor_name) perusteella tietyn sensorin datan ja palauttaa JSON-muodossa käyttäjälle.

FastAPI pystyy määrittämään päätepisteelle myös muuttujia, samalla tavalla kuin Python-ohjelmointikielen merkkijonoon pystyy sisällyttämään muuttujia. Päätepisteen muuttujat pystytään määrittelemään ennalta Pythonin Enum-luokalla, jolloin käyttäjä ei voi laittaa väärää muuttujaa päätepisteeseen (Ohjelma 2). Määritetty luokka Sensor periytyy str- ja Enum-luokasta, jolloin päätepistemuuttuja on tyyppiä merkkijono. Määrittelemällä luokkaan arvoja voidaan määritellä vain sallitut päätepistemuuttujat kyseiseen päätepisteeseen. Käyttäjän määrittäessä päätepistemuuttujan päätepisteeseen FastAPI validoi kyseisen muuttujan, jos muuttuja on väärä tai väärää tyyppiä rajapinta palauttaa virheviestin käyttäjälle sisältäen tarkan viestin virheestä ja listan oikeista arvoista päätepistemuuttujalle. Käyttäjä pystyy näkemään jo rajapinnandokumentaatiosta käytettävissä olevat rajapintamuuttujat kyseiselle päätepisteelle. Myöskin sensoreiden lisääminen

helpottuu, sillä määritettäessä lisätyt sensorit *Sensor*-luokkaan näkee myös käyttäjä heti uudet päätepisteen muuttujien arvot dokumentaatiosta (Ohjelma 2).

FastAPI-kirjastossa määritetään käytettävä HTTP-metodi ja päätepiste Pythonin koristesyntaksilla. Koristesyntaksin alapuolelle määritetään funktio. Kutsun tapahtuessa rajapinnan päätepisteeseen vaaditulla HTTP-metodilla koristeen alapuolelle määritetty funktio ajetaan. Ohjelmassa 2 määritetty *@app.get("/sensors/{sensor_name}"*) on koriste, joka määrittää päätepisteen (path operation decorator) ja kertoo FastAPI-kirjastolle, että sen alapuolelle määritelty funktio (path operation function) kuuluu sen yläpuolella määritettyyn polkuun.

Ohjelma 2 Rajapinnan päätepisteiden määrittely. Muuttujalla ja ilman muuttujaa.

```
12  class Sensor(str, Enum):
13      temperature = "temperature"
14      humidity = "humidity"
15
16
17  @router.get("/sensors", tags=["Sensors"])
18  async def data_from_sensors():
19      return sensor_data()
20
21
22  @router.get("/sensors/{sensor_name}", tags=["Sensors"])
23  async def data_from_specific_sensor(sensor_name: Sensor):
24      return sensor_name
25
```

4.4 Autentikointi ja rajapintakyselyiden rajoittaminen

IoT-laitteelle suunniteltava rajapinta halutaan autentikoida ja rajoittaa, jotta rajapinnasta saadaan turvallinen käyttää. Autentikoimalla pystytään rajoittamaan, ketä pystyy tekemään rajapintaan rajapintakutsuja. Käyttääkseen rajapintaa käyttäjä tarvitsee käyttäjänimen ja salasanan. Rajoittamalla kyselyiden määrää tietyn aikamääreen aikana, pystytään ehkäisemään tässä tapauksessa IoT-laitteen ylikuormittuminen, koska rajapinta tulee pyörimään IoT-laitteella. Samalla ehkäistään myös palvelunestohyökkäykset, koska sen tarkoituksena on ylikuormittaa palvelin monilla kyselyillä erittäin lyhyessä ajassa, jolloin palvelin ei pysty käsittelemään kaikkia kyselyitä ja menee tukkoon.

Rajapinnan autentikointiin päätettiin käyttää OAuth2-autentikointitapaa, koska se tarjoaa erittäin hyvät edellytykset rajapinnan turvallisuudelle. FastAPI-kirjasto tarjoaa valmiina OAuth2-turvallisuusskeeman, joten se on erittäin helppo implementoida rajapinnan

autentikointitavaksi. Kehitetty OAuth2-autentikointitapa käyttää hyväkseen käyttäjän syöttämää salasanaa ja käyttäjätunnusta, jolloin käyttäjän varmentaminen voidaan tehdä suoraan rajapinnassa.

Autentikointi toimii siten, että käyttäjä lähettää käyttäjänimensä ja salasanansa tiettyyn päätepisteeseen rajapinnassa. Käyttäjän salasana on salattu salausalgoritmin avulla. Rajapinta tarkistaa, että syötetty salasana ja käyttäjänimi ovat oikein ja palauttaa sen jälkeen poletin, joka koostuu pitkästä satunnaisesta merkkijonosta. Poletin avulla myöhemmin voidaan todentaa käyttäjä. Tämän jälkeen käyttäjä pystyy tekemään kyselyitä rajapinnan päätepisteisiin, jotka vaativat käyttäjän todentamisen. Todentaminen tapahtuu siten, että kyselyn auktorisointiotsikkoon laitetaan poletin arvon ja rajapinta tarkistaa täsmääkö poletin arvo annettuun polettiin.

Rajapinta rajoitetaan siten, että siihen pystyy tekemään kymmenen kutsua sekunnin aikana, jolloin rajapinta ei ylikuormitu. IoT-laitteelta rajapinta palauttaa sensoridataa, joten datan palauttaminen käyttäjälle kymmenen kertaa sekunnin aikana on riittävän reaaliaikaista.

4.5 Dokumentointi

Rajapinnan tuottamisen jälkeen keskitytään sen dokumentoimiseen, jotta tulevien käyttäjien on mahdollisimman helppo ottaa rajapinta käyttöön heidän tarpeisiinsa. Toteutettu dokumentti sisältää kuvaukset rajapinnan päätepisteistä. Kuvauksen avulla näkee, mihin tarkoitukseen päätepistettä käytetään, päätepisteen vaatimukset ja, mitä päätepiste palauttaa. Päätepiste kuvausten lisäksi esitetään muutama esimerkkikutsu rajapinnan päätepisteisiin.

Osana rajapinnan dokumentointia on tärkeää tuottaa JSON-skeema rajapinnan ominaisuuksista ja toiminnallisuuksista. Skeeman kielenä käytetään JSON-datamuotoa, koska se on ihmis- ja koneluettavissa. Hyvin tuotettu skeema rajapinnan toiminnasta auttaa kehittäjää tai käyttäjää ymmärtämään, mitä ominaisuuksia rajapinta sisältää ja mitä vaatimuksia kutsujen tekemiseen tarvitsee täyttää.

FastAPI-kirjastoa käytettäessä rajapinnan kehitykseen se generoi kirjoitetusta koodista rajapinnan dokumentaation automaattisesti, jolloin pystyy täysin keskittymään rajapinnan kehittämiseen. FastAPI generoi OpenAPI-standardeja käyttäen JSON-skeeman rajapinnasta. Generoitu dokumentti on kuvaelma koko rajapinnasta. Siitä näkee rajapinnan

päätepisteet, datamallit, palautettavan datan datatyyppi, päätepisteiden palauttamat virhekoodit kuvauksineen, autentikointimuodon ja päätepisteparametrit (Ohjelma 3).

Ohjelma 3 Esimerkki FastAPI-kirjaston generoiman ”/sensors”-päätepisteen dataskeema.

```
"SensorData":{
   "title":"SensorData",
   "type":"object",
   "properties":{
      "temperature":{
         "title":"Temperature",
         "type":"number",
         "description":"Degrees in Celsius",
         "example":22.7
      },
      "humidity":{
         "title":"Humidity",
         "type":"number",
         "description":"Percentage in decimal format",
         "example":0.475
      }
   }
},
```

4.6 Rajapinnan turvallisuus

Rajapinnan toteutuksessa mukailtiin turvallisen rajapinnan piirteitä.

Palvelin, jossa rajapintaa ylläpidetään, on konfiguroitu käyttämään HTTPS-protokollaa, jolloin kommunikointi palvelimen ja käyttäjän välillä on suojattu. Palvelimelle on asennettu SSL-sertifikaatti.

Salasanoihin käytetään salausalgoritmia, jolloin salasana merkkijonosta kryptataan satunnainen merkkijono salausalgoritmin mukaisesti. Vaikka salasana, joutuisi vääriin käsiin ei salasanan saaja pystyisi hyödyntämään salasanaa ilman salausalgoritmin tietämistä, sillä ilman sitä salasanaa ei voi dekryptata.

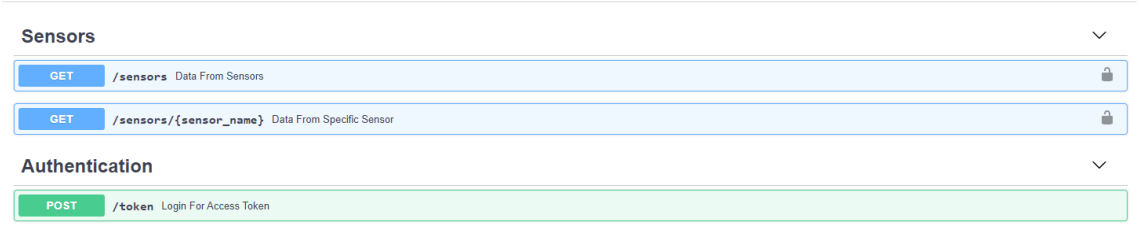
Lisäämällä aikaleiman jokaisen rajapintakutsun otsakkeeseen vältytään väkivoima hyökkäyksiltä. Rajapinnan palvelin vertaa aikaleimaa aikaan ja hyväksyy vain 1-2 minuutin sisällä olevan rajapintakutsun.

5 TESTAUS

Tässä kappaleessa testataan, että rajapinta toimii niin kuin haluttiin, sekä käydään läpi, että rajapinnalle asetetut vaatimukset käyvät toteen.

5.1 Rajapinnan kyselyiden testaaminen Swagger UI käyttöliittymän avulla

FastAPI tarjoaa käyttäjälleen Swagger UI käyttöliittymän avulla interaktiivisen rajapintadokumentaation, josta pystyy testaamaan generoitujen päätepisteiden toimintaa lähettämällä kutsuja dokumentin avulla rajapinnan päätepisteisiin. Rajapinnan ei tarvitse olla kokonaan valmis, vaan FastAPI generoi interaktiivista dokumenttia sitä mukaa, kun päätepisteitä ja datamalleja luodaan (Kuva 8).

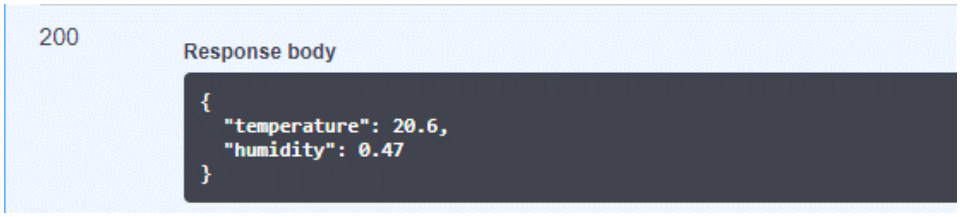


Kuva 8 Rajapinnan päätepisteet interaktiivisessa rajapintadokumentissa.

Generoidun päätepisteen pystyy klikkaamaan auki, josta löytyy tietoa päätepisteen datamallista, päätepisteen mahdolliset parametrit, testikutsun vastauksen, sekä päätepisteen polun. *Execute* -nappia painamalla dokumentti lähettää kutsun rajapintaan ja palauttaa vastauksen käyttäjälle näkyviin. Tämän avulla pystytään helposti ja todella nopeasti näkemään palauttaako rajapinta datan halutussa muodossa. Pystyy testaamaan myös mitä rajapintakutsu palauttaa, kun rajapinta kutsussa tapahtuu virhe.

Rajapinnan testauksessa käytettiin hyväksi FastAPI-kirjaston generoimaa interaktiivista rajapintadokumentaatiota. Sen kautta tehtiin kutsuja rajapintaan ja tarkistettiin, että rajapinta palauttaa oikean datan, kun kutsu on onnistunut. Pystytään toteamaan, että rajapinta palauttaa dataa selkeässä ja helposti luettavassa muodossa (Kuva 9). Testattiin myös HTTP-virhekoodien toimivuus, kun käyttäjän tekemässä rajapintakutsussa on virhe, niin palautettiin käyttäjälle virhekoodi sisältäen informaatiota siitä, miksi rajapintakutsu epäonnistui. Samalla tarkistettiin, että onnistuneessa rajapintakutsussa käyttäjälle

palautettava data on oikeassa muodossa. Rajapintaan tehtiin myös kutsuja, kun käyttäjä ei ollut autentikoinut itseään, jolloin varmistetaan, että rajapinta palauttaa virhekoodin käyttäjälle eikä palauta sensoridataa. Virhekoodi informoi käyttäjää, ettei hän ole autentikoinut itseään.



Kuva 9 Rajapinnan päätepisteeltä palautuva data. Data on selkeässä ja helposti luettavassa muodossa.

Ylitettäessä rajattujen kutsujen määrän sekunnissa huomataan, että rajapinta palauttaa käyttäjälleen HTTP-virhekoodin 429. Virhekoodi 429 kertoo, että kyselyiden määrä rajapintaan on ylitetty yhden sekunnin aikana.

5.2 Rajapinnan nopeus

Toteutetusta rajapinnasta haluttiin mahdollisimman nopea, jolloin dataa pystytään vastaan ottamaan mahdollisimman reaaliaikaisesti. Rajapinnan nopeutta mitataan rajapintakutsun lähettämisestä rajapintavastauksen vastaanottamiseen kuluvaa aikaa. Tämä voidaan mitata tekemällä komentorivillä bash-komento. Bash-komennon avulla tehdään rajapintakutsu rajapintaan. Komento palauttaa rajapintakutsuun ja vastauksensaamiseen käytetyn ajan. Työn aikana kehitetyn rajapinnan vastaus nopeus eli siihen kuluva aika oli 0,218–0,219 ms eli rajapinnasta saa vastaanotettua dataa nopeasti.

5.3 Rajapinnan versiointi ja jatkokehitysmahdollisuus

Rajapinnan vaatimuksia määriteltäessä haluttiin, että toteutettua rajapintaa on mahdollisimman helppo jatkokehittää eli kehittää uusia versioita. Rajapinnasta haluttaisiin toteuttaa uusia versioita siinä tapauksessa, kun halutaan lisätä uusia sensoreita tai kontrolloitavia laitteita, kuten valoja mikrokontrolleriin. Käytetyn rajapintakirjaston ansiosta rajapinnan päivittäminen onnistuu todella sujuvasti. Päivitettäessä rajapintaan uusia päätepisteitä aikaisemmat päätepisteet eivät mene solmuun, vaan käyttäjä pystyy käyttämään

niitä edelleen uusista päätepisteistä huolimatta. Käyttämällä FastAPI-kirjaston APIRouter-instanssia pystyy luomaan päätepistepolkuja, jotka ovat riippumattomia toisistaan, mutta silti kuuluvat samaan applikaatioon.

6 YHTEENVETO

Tämän työn tarkoituksena oli suunnitella ja toteuttaa rajapinta IoT-laitetta varten. Rajapinnan kautta pystytään hakemaan IoT-laitteen lämpötila- ja kosteussensorin keräämää dataa. Kerättyä dataa pystyisi käyttämään mobiili- tai verkkosovelluksissa.

Rajapinnan suunnittelun aikana vertailtiin ja pohdittiin parasta mahdollista rajapinta-arkkitehtuuria, rajapinnan datamuotoa ja rajapintakirjastoa toteutusta varten. Rajapinnan vaatimusten ja vertailun tulosten perusteella valittiin teknologiat, joita käyttäen rajapinta toteutettiin.

Käyttämällä aikaa hyvään pohjatyöhön, suunnitteluun ja vertailemalla eri teknologioiden tarjoamia ominaisuuksia pystytään toteuttamaan toimiva rajapinta todella helposti ja nopeasti IoT-laitetta varten. Rajapinnan kehittämistä varten valittu FastAPI-rajapintakirjasto tarjosi todella hyvät edellytykset rajapinnan kehitystä varten sen tarjoamien valmiiden ominaisuuksien ansiosta. Rajapinnan sai kehitettyä helposti ja ongelmitta. Kirjaston ansiosta saatiin kehitettyä suunnittelun ja vaatimusten mukainen rajapinta IoT-laitteelle. Rajapinta on nopea, data on helposti luettavassa muodossa, rajapintaa pystyy jatkokehittämään helposti, rajapinta on autentikoitu ja se on kevyt.

LÄHTEET

altexsoft.com. Luettu 27.03.2020. What is SOAP: Formats, Protocols, Message Structure, and How SOAP is Different from REST. https://www.altexsoft.com/blog/engineering/what-is-soap-formats-protocols-message-structure-and-how-soap-is-different-from-rest/

Andrew Park. Luettu 25.03.2020. How do APIs work? https://tray.io/blog/how-do-apis-work

Austin Wright, Hendry Andrews, Ben Hutton. 2019. JSON schema Validation: A Vocabulary for Structural Validation of JSON. https://json-schema.org/draft/2019-09/json-schema-validation.html

avoinrajapinta.fi. 2014. Avoimen rajapinnan määritelmä. http://avoinrajapinta.fi/

Carsten Bormann. Luettu 04.04.2020. RFC 7049 Concise Binary Object Representation. http://cbor.io/

Carsten Bormann. Luettu 07.04.2020. CoAP. https://coap.technology/

Carsten Bormann. Luettu 07.04.2020. Specification. https://coap.technology/spec.html

Douglas Crockford. Luettu 02.04.2020. Introduction to JSON. https://www.json.org/json-en.html

Encode OSS Ltd. Luettu 05.04.2020. Authentication. https://www.django-rest-framework.org/api-guide/authentication/

Encode OSS Ltd. Luettu 05.04.2020. Django Rest Framework. https://www.django-rest-framework.org/

Encode OSS Ltd. Luettu 05.04.2020. Serializers. https://www.django-rest-framework.org/api-guide/serializers/

FastAPI. Luettu 09.04.2020. FastAPI. https://fastapi.tiangolo.com/

FastAPI. Luettu 09.04.2020. FastAPI features. https://fastapi.tiangolo.com/features/

foundation.graphql.org. Luettu 27.03.2020. GraphQL foundation. https://foundation.graphql.org/

guest writer. 2018. IoT is eating the World: APIs and REST. https://www.iotforall.com/iot-rest-api/

Jen Clark. 2016. What is the Internet of Things? https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/

John Sturtz. Luettu 05.04.2020. Python modules and packages – an introduction. https://realpython.com/python-modules-packages/

json-schema-org. 2019. JSON Schema Specification. https://json-schema.org/specification.html

Kristopher Sandoval. 2019. 3 Common methods of API authentication explained. https://nordicapis.com/3-common-methods-api-authentication-explained/

Kristopher Sandoval. 2016. What data formats should my API support? https://nordicapis.com/what-data-formats-should-my-api-support/

Margaret Rouse. 2019. application programming interface (API). https://searchapparchitecture.techtarget.com/definition/application-program-interface-API

Ogla Annenko. 2016. 6 Characteristics that make APIs fit for application integration. https://www.elastic.io/6-characteristics-of-great-api/

Quintagroup. Luettu 05.04.2020. Django REST framework. https://quintagroup.com/cms/python/django-rest-framework

Red Hat Inc. Luettu 30.03.2020. What is GraphQL. https://www.redhat.com/en/topics/api/what-is-graphql

restfulapi.net. Luettu 03.04.2020. JSON vs XML. https://restfulapi.net/json-vs-xml/

restfulapi.net. Luettu 25.04.2020. Rest API security essentials. https://restfulapi.net/security-essentials/

soapui.org. Luettu 25.03.2020. SOAP vs REST 101: Understand The Differences. https://www.soapui.org/learn/api/soap-vs-rest-api/

Tasia Potasinski. 2019. Secrets to Great API Design. https://dzone.com/articles/secrets-to-great-api-design-1

Tero Kivisaari. 2016. API:t ovat modernin integraatiostrategian ydin. https://blog.digia.com/rest-api

W3C. 2016. Extensible Markup Language (XML). https://www.w3.org/XML/

w3schools. Luettu 03.04.2020. XSD Simple Elements. https://www.w3schools.com/xml/schema_simple.asp