IMB IoT Cloud ja HTTP REST API -kommunikaatio

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeen ammattikorkeakoulu, tieto- ja viestintätekniikka

Syksy, 2019

Arto Haaranen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikan insinööri
Riihimäki

Tekijä Arto Haaranen **Vuosi** 2019

Työn nimi IMB IoT Cloud ja HTTP REST API -kommunikaatio

Työn ohjaaja/t Timo Karppinen

TIIVISTELMÄ

Työn tarkoituksena oli selvittää kuinka IMB Watson IoT Platformissa saadaan sensori kommunikoimaan REST API -protokollalla. Sensorin antama data käsiteltäisiin IBM Cloud NodeRED -työkalun kautta ja visualisoitaisiin web-sivulle.

Järjestelmän testaukseen käytettiin Arduino MKR1000 -mikrokontrolleria ja siihen kytkettyä Pmod ALS ympäristön valaistusta mittaavaan sensoriin.

Ensimmäiseksi testattiin testilaitteiden toiminnallisuutta MQTT-kommunikaatioprotokollalla IBM Watson IoT Platformin ja NodeRED -työkalun kanssa. Valoisuussensorin SPI -sarjaporttikommunikaatio toimi ja Arduinon Wlan -moduuli yhdistyi lähiverkkoon. MQTT -kommunikaatio ja datan visualisointi web-sivulle onnistui.

Työn toisessa vaiheessa testattiin, miten REST API -kommunikaatiolla saatu data onnistuttaisiin visualisoimaan. IMB Watson IoT Platformin kanssa oli ongelmia, koska alusta on suunniteltu pääasiallisesti MQTT -kommunikaatiolla. Tästä syystä päädyttiin käyttämään IBM Db2 on Cloud -palvelua järjestelmän toteuttamiseen.

Työtä tehdessä etsittiin IBM Cloud -palvelun työkaluvaihtoehtoja, joilla voitaisiin työ toteuttaa. Ongelmaksi kuitenkin muodostui monesti se, että ilmaisten kokeiluversioiden toiminnallisuus ei vastannut ohjelehtien ja alkuopasteiden tietoja. Datan säilöntä onnistui RESP API -kommunikaatiolla toimivan datan säilöntä ja visualisointi järjestelmä IBM Cloud -palvelussa.

Avainsanat Db2, IBM Cloud, NodeRED, REST API, SQL

Sivut 30 sivua, joista liitteitä 12 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology
Riihimäki

Author Arto Haaranen **Year** 2019

Subject IMB IoT Cloud ja HTTP REST API communication

Supervisors Timo Karppinen

ABSTRACT

The purpose of this project was to find out how we can make a sensor communicate with the REST API protocol in the IBM Watson IoT Platfrom. The data sent by the sensor would be parsed with the IBM Cloud NodeRED tool and visualized in the web user interface.

The system was tested with an Arduino MKR1000 microcontroller and a Pmod ALS sensor connected to it.

First, the functionality of the test devices was tested with the MQTT communication protocol in the IBM Watson IoT Platform and the NodeRED tool. The light sensor's SPI serial port communication worked, the Arduino Wlan module was connected to the LAN successfully and MQTT communication and visualization of the data to the web page was succesful.

The second phase of the work involved testing how the data generated by the REST API communication could be successfully visualized. There were problems with the IMB Watson IoT Platform because the platform is primarily designed for MQTT communication. For this reason, it was decided to use the IBM Db2 on Cloud service to implement the system.

As the work progressed, we looked for IBM Cloud tool options that could work. However, during the project a problem often arose because the functionality of the free trial versions did not match the information in the manuals and tutorials. However, it was possible to create a data repository and visualization system with the IBM Cloud service using RESP API communication.

Keywords Db2, IBM Cloud, NodeRED, REST API, SQL

Pages 30 pages including appendices 12 pages

SISÄLLYS

1 JOHDANTO 1

2 KEHITTÄMISTYÖN TIETOPERUSTA 2

2.1 HTTP REST API 2
2.2 Tietokannat 3
2.3 Mikrokontrolleri ja sen kommunikaatio 3
2.4 Internet of Things (IoT) 5

3 KEHITTÄMISTYÖN TAVOITE JA TARKOITUS 6

4 SYSTEEMIN SUUNNITTELU JA TOTEUTUS 7

4.1 Olemassa olevan ratkaisun testaus 7
4.2 REST API toteutus 9

5 JOHTOPÄÄTÖKSET JA POHDINTA 15

5.1 Projektin eteneminen 15
5.2 Kehitysmahdollisuudet ja käyttöönotto 16

LÄHTEET 17

Liitteet
Liite 1 MQTT Arduino -koodi
Liite 2 REST API Arduino -koodi

1 JOHDANTO

Projektin tarkoituksena oli tutkia Hämeen ammattikorkeakoululle, kuinka http REST API -kommunikaatio toimisi IMB Watson IoT Platformissa. Koulussa oli tehty samassa ympäristössä harjoituksia käyttäen MQTT-kommunikaatioprotokollaa. Ympäristön testaamiseen projektiin lainattiin Arduino MKR1000 -mikrokontrolleria ja Pmod ALS -sensoria, joka mittaa ympäristön taustavaloa. Tarve systeemin muokkaamiselle syntyi uusien http REST API -kommunikaatiota käyttävien Advantech datalogger -sensorien myötä, joita Hämeen ammattikorkeakoulu oli tilannut.

Oma kiinnostus IBM Cloud -ympäristöihin heräsi toisella lukuvuodella sulautettujenjärjestelmien opintokursseilla. Kiinnostusta herätti myös tieto siitä, että monessa tietotekniikan yrityksessä hyödynnetään vastaavia ja samoja palveluita. Näiden ympäristöjen harjoittelu ja opiskelu auttaisi varmasti tulevaisuudessa työnhaussa. Työssä pääsisi myös perehtymään IoT-laitteiden kommunikaatioon verkossa.

Haluttu toiminnallisuus olisi se, että sensorilta data saataisiin haettua NodeRED-käyttöliittymästä käsin. Data voitaisiin visualisoida NodeRED-käyttöliittymässä ja tallentaa johonkin palveluun talteen. Tutkittavana oli, millä menetelmillä tämä kaikki olisi mahdollista toteuttaa.

2 KEHITTÄMISTYÖN TIETOPERUSTA

Hämeen ammattikorkeakoulussa on testattu MQ Telemetry Transport eli MQTT-kommunikaatioprotokollalla toimivia sensorilaitteiden yhdistämistä IMB Watson IoT Platformiin. Sensorilaitealustana on käytössä Arduino MKR1000 -mikrokontrolleri. Sensorina on käytetty Pmod-taustavalosensoria. Arduinolle on koodattu sensorin luenta ja lähiverkkoon liittymiseen tarvittavat funktiot C-kielellä.

2.1 HTTP REST API

Representational State Transfer joka lyhennetään REST on tekstimuotoinen tapa tietojärjestelmän kommunikoida verkkopalvelimen kanssa. Application programming interface eli API on määritelmä rajapinnalle, missä ohjelmat voivat kommunikoida keskenään. REST-protokolla hyväksikäyttää olemassa olevaa Hypertext Transfer Protocol eli http-kommunikaatio tapaa viestin välittämiseen internetissä. REST-komentoihin kuuluu joukko ennalta määrättyjä komentoja, joista yleisimmin käytettävissä olevat ovat GET, PUT, POST ja DELETE. Kaikessa yksinkertaisuudessaan REST-komennot muokkaavat yksittäistä objektia, kokonaista tiedostoa tai data luku jonoa. GET kutsuu resurssin, PUT päivittää tietoa tai vaihtaa resurssin tilaa, POST luo ja lähettää jotain uutta ja DELETE poistaa tiedon. (Rouse, 2019)

REST-kommunikaation salaustapoina on käytössä Secure Sockets Layer eli SSL ja Transport Layer Security eli TLS, jotka ovat myös käytössä http-protokollassa. SSL ja TLS ovat salaustapoja, joissa data kryptataan ja avaimet annetaan vain kahden systeemin välille. Näin dataan ei pääse ulkopuolinen käsiksi, kun dataa liikutetaan. Ero SSL- ja TLS-salausmenetelmissä on se, että TLS-protokolla on paranneltu versio SSL-protokollasta. TLS-protokolla on myös raskaampi ja hitaampi salausmenetelmä, jonka takia SSL-protokolla on vielä käytössä ympäristöissä, joissa voidaan uhrata turvallisuutta tehokkuuden varjolla. (Digicert, n.d.)

Komentoja REST-viestille voi kirjoittaa lähes millä tahansa koodikielellä, koska REST-kommunikaatio tarvitsee vain koodikielen toimittamaan sille verkkomuotoisen http-sanoman (Rouse, 2019). Tämän takia REST-komentoja voi lähettää kaikenlaisista ympäristöistä, kuten palvelimilta tai jopa patterikäyttöisiltä mikrokontrollereilta.

2.2 Tietokannat

Kun sensorilukemia, yhteystietoja tai mitä tahansa dataa tarvitsee säilyttää, tarvitaan tietokantoja. Tietokannat ovat tiedon järjestelmällistä tallentamista, jotta yksittäisiä tapahtumia voidaan myöhemmin selata helposti.

Yleisimmin käytetty tietokantamalli on relaatiotietokanta, joka tallentaa tiedon datatauluun. Taulujen kolumneihin tallennetaan datan arvot ja datan tyyppi on yleensä ennalta määritellyn tyylistä. Esimerkiksi sensoridatalla on selvä datan lähetyskaava. Tiedettiin, että tallennettavat datat ovat tälle sensorille aikaleima, lämpötila ja valoisuus. Ennalta määrätty tyyppi mahdollistaa datan oikeaan muotoon parsimisen ennen tallennusta. Parsimisen tarve voi tulla vastaan, jos samaan tauluun halutaan tallentaa kahdelta eri sensorilta dataa, jotka lähettävät datan eri aikavyöhykkeiltä. Tyyppimäärittely myös mahdollistaa datan vertaamisen taulun sisällä oikein. Esimerkiksi tekstimuotoista aikaleimaa ei voi luotettavasti järjestää kronologiseen järjestykseen. Relaatiotietokannan eri taulujen datojen välille voidaan muodostaa yhteys. Näin voidaan luoda yhdisteltyä datan tarkastelu näkymään toiseen tauluun. (Guru99, n.d.)

Relaatiotietokantojen hallinnointiin on tehty oma koodikieli Structured Query language eli SQL. SQL-komennoilla hoidetaan kaikki tarvittavat toimenpiteet. Komennoilla luodaan datatauluja ja -kolumneja, muutetaan käyttäjien oikeuksia sekä tallennetaan ja haetaan tietoa.

Nonrelational Structured Query language, joka lyhennetään NoSQL, on moderni tietokanta. Sen sijaan, että tietokantaan määriteltäisiin lukittuja relaatioita datan ja tietokannan taulujen sekä kolumnien välille, käytetään datan muotoiluun älykästä kaaviota. NoSQL-tietokanta on joustava ja tehokas tietokanta, joka on tärkeää valtavien datavirtojen tallentamiseen moderneista IoT-ympäristöistä. NoSQL-tietokantoja on viittä eri tyyppiä, jotka ovat avainarvo, dokumentti, graafi, muisti ja etsintä. (Amazon, 2019)

IBM Cloud -ympäristön NoSQL-tietokanta on IBM Cloudant ja se on dokumentti tyyppinen tietokanta. Datan tallennus tietokantaan tehdään JavaScript Object Notation -muodossa. Tietokanta on monistettava, joka parantaa datan synkronointia eri systeemien välillä. (IBM, 2019)

2.3 Mikrokontrolleri ja sen kommunikaatio

Mikrokontrollerit ovat tietokoneita pienoiskoossa. Näitä minitietokoneita on sulautettu lähes kaikkeen elektroniikkaan. Mikrokontrollerit ovat vähävirtaisia, pienikokoisia ja mukautuvia systeemejä. Yleensä mikrokontrolleri suorittaa erittäin yksinkertaista toistuvaa toimintaa, joka vaatii jonkinlaista laskentaa ja prosessointia. Jokainen laite on suunniteltu ja koodattu toimittamaan ennalta määrättyä viestinvälittämistä tai muun laitteen ohjausta.

Viestintätapoina mikrokontrollereissa käytetään digitaalista-, analogista- ja myös sarjaliikennekommunikaatiota. Digitaalinen viestintä toimii TOSI- ja EPÄTOSI-tilojen välittämisellä yksittäisten lähtöjen ja tulojen välillä. Näin saadaan aikaiseksi eri aikoja jatkuvia yksi- tai nollatiloja. Analoginen viestintä pyrkii käyttämään pinniin johdettavan jännitteen kokoskaalaa. Arvoväli on yleensä porrastettu mikrokontrollereissa kahdeksan bitin asteikolle eli saadaan 256 mahdollista arvoa.

Sarjakommunikaatiossa viestit ovat kokonaisia bittijonoja. Viesti muodostetaan kahdella eri tavalla. Viesti voi olla tahdistettu, joka tarkoittaa, että erillisessä johtokytkennässä annetaan tahti. Tahtia kutsutaan kellopulssiksi ja se on tasainen virta ykkösiä ja nollia vuoron perään. Tahdistamaton viesti ei tarvitse kytkentää antamaan tahtia, mutta viestit pitää muodostaa rajoitetummin. Bittijonojen täytyy sisältää määrätty aloitus- ja lopetusbitti sekä laitteiden täytyy käyttää samaa siirtonopeutta. (Sparkfun, n.d.)

Yksi tahdistetuista sarjaliikenneprotokollista on Serial Peripheral Interface, joka lyhennetään SPI. Kommunikaatio muodostetaan neljän yhdistetyn pinnin avulla. Ensimmäinen pinni on viestikanava ohjaavalta laitteelta ohjattavalle. Tätä kutsutaan Master Output Slave Input pinniksi, joka lyhennetään MOSI. Toinen pinni on taas ohjattavalta laitteelta takaisin ohjaavalle tuleva viesti ja pinni on nimetty Master Input Slave Output eli MISO. Kolmas pinni antaa tahtia kellopulssilla. Viimeinen neljäs pinni on tarkoitettu ilmoittamaan ohjattavalle laitteelle, että on tämän vuoro vastata. Ohjaavalla laitteella voi siis olla useampi ohjattava ja kuunneltava laite. Näin voidaan kuunnella tyypillisesti samalle piirikortille kytkettyä useampaa sensoria neljällä pinnillä. (Dhaker, 2018)

Pidempien etäisyyksien ja eri piirikorteilla olevien laitteiden väliseen sarjaliikennekommunikaatioon käytetään yleisimmin sähköisiä RS422- ja RS485-sarjaliikenneratkaisuja tai Controller Area Network eli CAN-väyläprotokollaa. CAN-väyläprotokolla tarkoittaa järjestelmän laitteiden kommunikaatioverkkoa ja sen fyysistä johdotusta, joka on pyritty saavuttamaan tehokas mutta yksinkertaisella verkolla, jossa vain yksi johto kulkee laitteiden välillä. CAN-väylässä viesti lähetetään kaikille verkon laitteille ja viestin sisältö määrittää, mille laitteelle viesti oli tarkoitettu. CAN-viesti muodostuu 11:n bitin jonosta ja päivitetyssä Extended CAN -ympäristössä 29:n bitin jonosta. CAN-viestit lähetetään väistyvällä ja dominoivalla prioriteeteilla. Kun samassa verkossa viestitetään yhtäaikaisesti suuremmalla prioriteetilla oleva sanoma saa etuoikeuden ja väistyvä sanoma unohdetaan toistaiseksi. Määrätyn ajan kuluttua väistyvän sanoman lähettänyt laite pyrkii lähettämään sanomansa uudestaan. CAN-väyläprotokollassa käytetään signaalisointitapaa, joka on resistiivinen häiriöille. Tämä tekee protokollasta erittäin toimivan ja käytetyn ratkaisun teollisuudessa. (Stephen, 2019)

2.4 **Internet of Things (IoT)**

Esineiden internet eli IoT tarkoittaa tekniikoita, joilla laitteita kytketään internet-verkkoon. IoT-ympäristö voidaan jakaa kolmeen osa-alueeseen, jotka ovat IoT-laitteet, IoT-yhdyskäytävät ja IoT-alustat. IoT-laitteita ovat sensorit, tietokoneet, tehdasvälineet tai monet muut laitteet, mitkä lähettävät viestejä internet-verkkoon. Tavanomaisesti laitteet ovat vähävirtaisia ja monesti akkukäyttöisiä, mutta määritelmä voi kattaa laajemmankin skaalan dataa lähettävää kalustoa. Laite ei kuitenkaan kommunikoi kuin yhden yhteyspisteen kanssa eli laite toimii yhtenä ympäristön päätepisteenä. (Arm, n.d.)

Toisessa päässä IoT-ympäristöä on IoT-alustat. Alustat ovat monitasoisia-järjestelmiä, joilla ympäristössä olevat laitteet saadaan yhdistymään toisiinsa. Alustoilla voidaan tehdä älykkäitä järjestelmiä, joissa laitteet liitetään internet-verkossa toimiviin palveluihin turvallisesti. Nämä palvelut voivat olla suuria datananalysointijärjestelmiä, loppukäyttäjälle luotuja käyttöliittymiä tai monia muita systeemejä. Alustat ovat helposti skaalautuvia ja muokattavia sekä nykypäivän suojausmenetelmät tuovat turvaa arkaluontoisille tiedoille. (Kaaproject, n.d.)

Näiden kahden loppupään välille tarvitaan käytävä missä tieto kulkee. Tätä yhteyspistettä kutsutaan IoT-yhdyskäytäväksi. Käytävä voi olla hyvin yksinkertainen laite, joka vain yhdistää sensorin verkkoon ja sitä kautta välittää tiedon käytettyyn alustaan. IoT-yhdyskäytävä voi olla myös älykäslaite, joka kerää useammalta laitteelta tietoja, suodattaa ja jäsentelee datan ennen sen lähettämistä loppualustalle. (OAS, n.d.)

3 KEHITTÄMISTYÖN TAVOITE JA TARKOITUS

Tavoitteena työssä on selvittää, miten IBM Cloud -ympäristöä voidaan käyttää RESP API -protokollalla kommunikoivien sensorien kanssa. Olemassa oleva Arduino-mikrokontrolleri lähettää dataa MQTT-kommunikaatioprotokollalla IBM Watson Platformiin. Kokoonpano pitää muuntaa toimimaan suoraan REST-kommunikaatiolla ilman MQTT-autentikointimenetelmiä. Tiedostettiin, että autentikointitavan muuttaminen voisi aiheuttaa ongelmia IBM Watson Platform -ympäristössä. Tarve vain REST-kommunikaatiolla toimivan systeemin rakentamiseen tuli Hämeen ammattikorkeakoulun hankkimien Advantech datalogger -sensorien takia. Laitteet oli suunniteltu toimimana suoraan REST-protokollakutsuilla.

Ensisijainen vaatimus olisi saada reaaliaikaista sensoridataa näkymään selaimelta. Sensorin lähettämä data tulisi visualisoida NodeRED-ympäristöön. Sensoridatan tallentaminen jonkinlaiseen tietokantaan tai palveluun lisäisi myös ympäristön mahdollisuuksia. Tallennetusta datasta voisi muodostaa erilaisia histogrammeja, jotka voisivat antaa arvokkaampaa tietoa seurattavasta kohteesta.

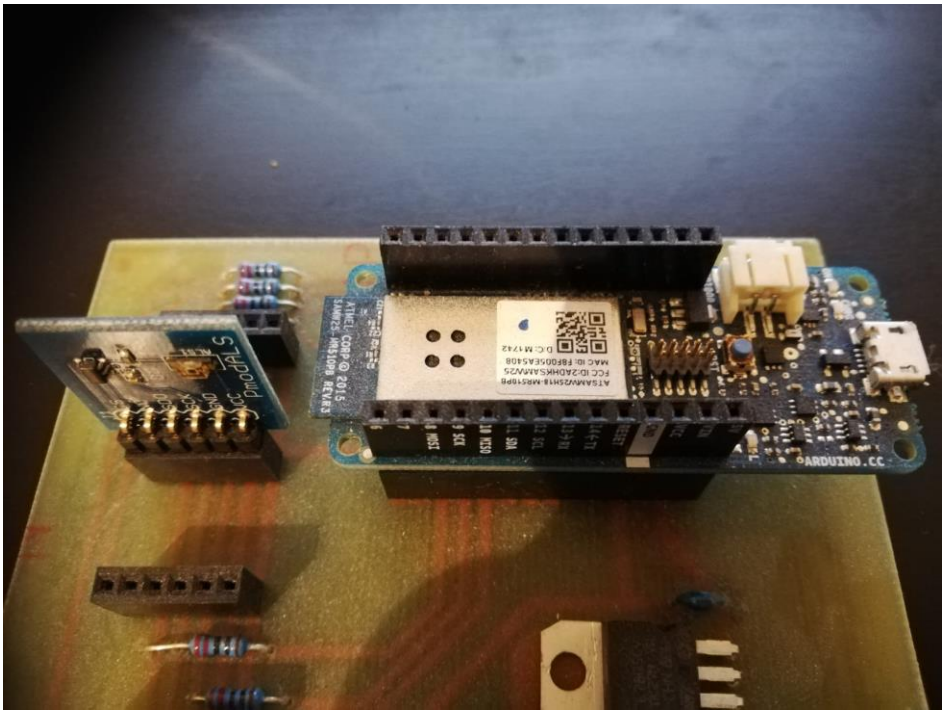
Täytyisi myös pohtia, miten datan saisi suojattua ja mitä riskejä systeemissä on. Tulisi arvioida onko data kuinka salattavaa, että minkä tasoisilla salausprosesseilla saavutettaisiin tarpeellinen suoja. Ympäristöä ei saisi myöskään ylikuormittaa liian raskaalla suojauksella. Oletus on, että käytettävät laitteen tulevat aina olemaan pienitehoisia ja vähävirtaisia. Tämä tuo rajoituksia ympäristöön, joka täytyy ottaa huomioon toteutuksessa.

4 SYSTEEMIN SUUNNITTELU JA TOTEUTUS

Toteutus jaettiin seuraaviin osiin. Ensimmäiseksi tulisi todentaa olemassa olevan ympäristön toiminnallisuus ja mahdolliset puutteet. IBM Watson Platformin moduulien itse pystyttäminen saatujen ohjeiden mukaan, auttaisi ymmärtämään systeemin toimintaperiaatetta. Samalla tulisi tutustua Arduino-mikrokontrolleriin ajettuun koodiin ja selvittää, miten Arduinoon liitetyn sensorin dataa luetaan, kuinka laite yhdistyy verkkoon ja miten IBM Cloud -ympäristöön voidaan yhdistyä. Näin toiminnallisuutta voitaisiin jatkossa muokata. Seuraavaksi suunnitelmana olisi tutkia kuinka Arduinon sensoridata saataisiin lähetettyä samaan IBM Watson Platform -palveluun. Miten saataisiin palvelu kutsumaan suoraan laitetta REST-komennoilla ja mitä lisämoduuleja systeemi tarvitsee toimiakseen. Kun ratkaisu olisi selvillä systeemi pystytettäisiin ja testattaisiin fyysisesti. Viimeiseksi tarkoituksena olisi pohtia, miten systeemiä voisi tulevaisuudessa päivittää. Datan tallentaminen tietokantaan ja tietoturvatoteutuksen parantaminen olisivat hyviä parannuskohteita.

4.1 Olemassa olevan ratkaisun testaus

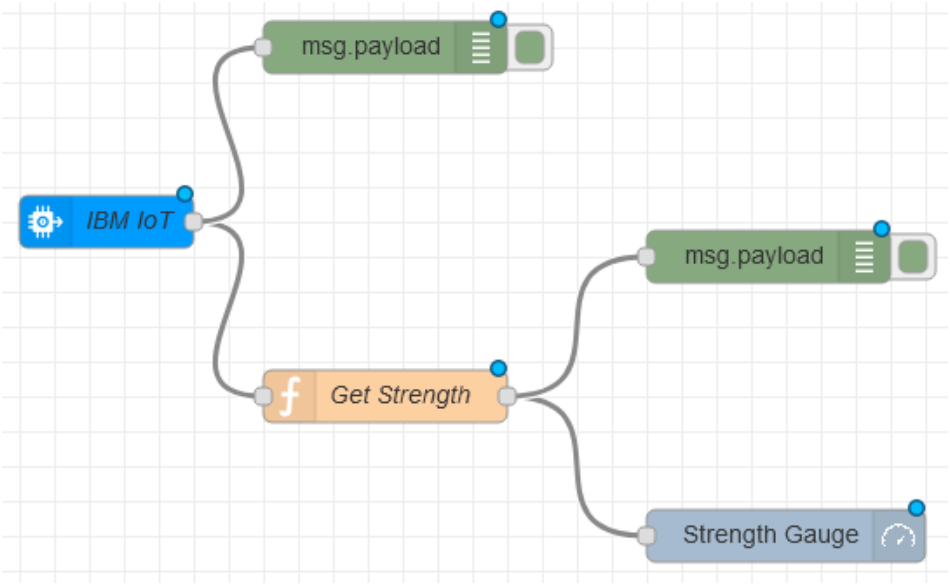
Sensorin mallintamiseen ja testaamiseen käytettiin Arduino MKR1000 -mikrokontrolleria, johon oli kytketty Pmod-taustavalosensori. Laite tuli valmiiksi SPI-sarjakommunikaatiomenetelmällä kytkettynä taustavalosensoriin (Kuva 1).



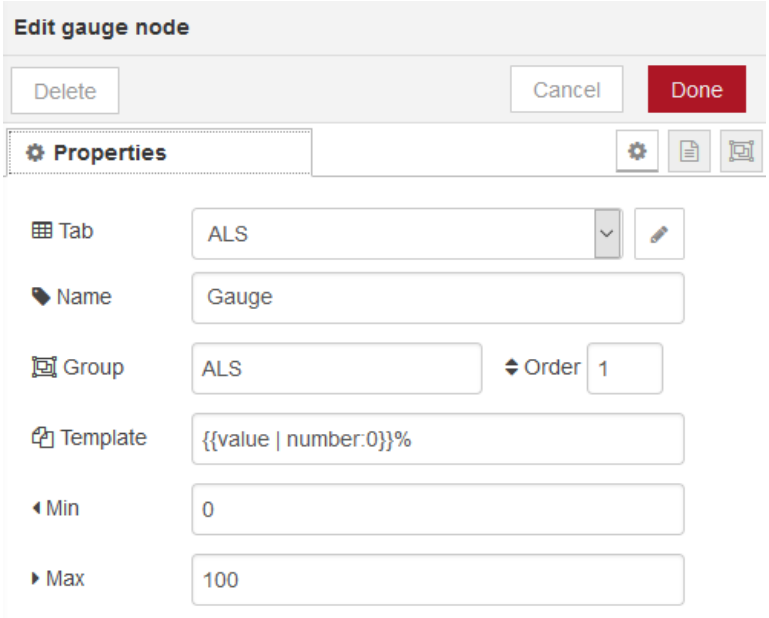
Kuva 1. Taustavalosensori ja Arduino MKR1000

Käytössä oleva koodi, hallitsi lähiverkkoon kytkeytymisen, sensoridatan lukemisen ja MQTT-kommunikaation IBM Watson Platformin kanssa (Liite 1). Oma IBM Watson Platform -ympäristö ja laitteen tunnistautumistiedot luotiin. Arduinossa olevaa koodia muokattiin Arduino Integrated Development Environmentillä eli IDE:llä, niin että, laite yhdisti juuri luotuun IBM Watson Platform -ympäristöön. Yhteys onnistuttiin luomaan ja pienien koodimuutosten jälkeen data saatiin lähetettyä verkkoon oikeanlaisessa muodossa.

Viimeiseksi piti todentaa miten visualisointi onnistuisi IBM NodeRED -ympäristön kautta. NodeRED sisältää valmiita liitännäisiä, joilla voidaan kuunnella dataa IBM Watson Platformin viestin välittäjältä (Kuva 2). Viesti vastaanotetaan JavaScript Object Notation eli JSON muotoisena ja jotta viestin sisältämä arvo saataisiin esitettävään muotoon, se piti ensin jäsentää (Kuva 3).

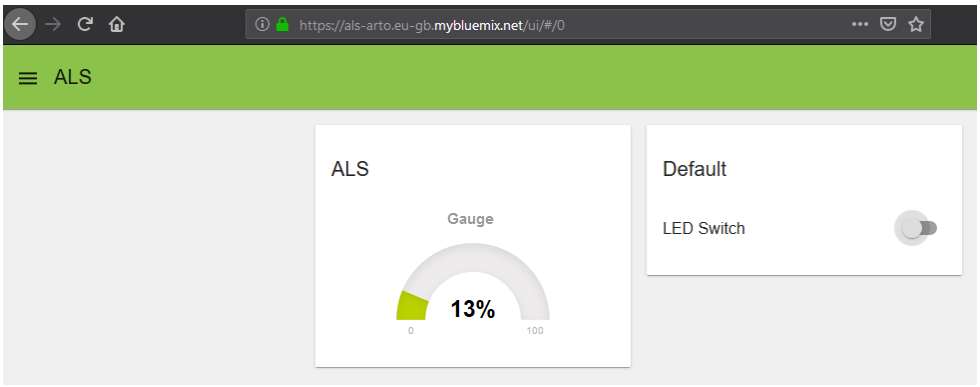


Kuva 2. NodeRED-funktiokaavio



Kuva 3. JSON-arvomuotoilu käyttöliittymään

NodeRED-ympäristössä ei automaattisesti ole aktiivisena käyttöliittymän liitännäisiä. Nämä täytyy käydä aktivoimassa asetuksista palettien alta. Datan visualisointiin tarvittavat paketit ovat node-red-contrib-ui ja node-red-contrib-aggregator. Testikäyttöliittymään visualisoitiin taustavalosensorin antama data prosentteina, koska sensorin asteikko ei ole muodostettu vastaamaan mitään valoa kuvaavaa vakiota (Kuva 4). Asteikko on muodostettu 8 bitin tarkkuudella. Nolla tarkoittaa pientä valoisuutta ja 256 tarkoittaa suurta valoisuutta (Digilent, 2016).



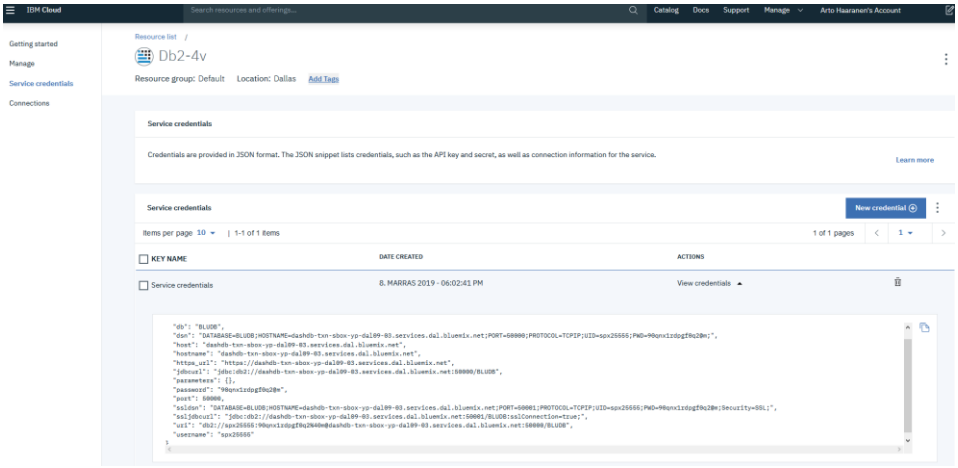
Kuva 4. Testiympäristön käyttöliittymä

Testattiin myös, miten REST API -kommunikaatio toimii suoraan NodeRED-ympäristöstä. Laite saatiin vaihtamaan ledin tilaa REST-komennolla. Tässä onnistuttiin vain silloin, kun laite oli jo autentikoinut itsensä MQTT-turvallisuusprotokollien kautta.

4.2 REST API toteutus

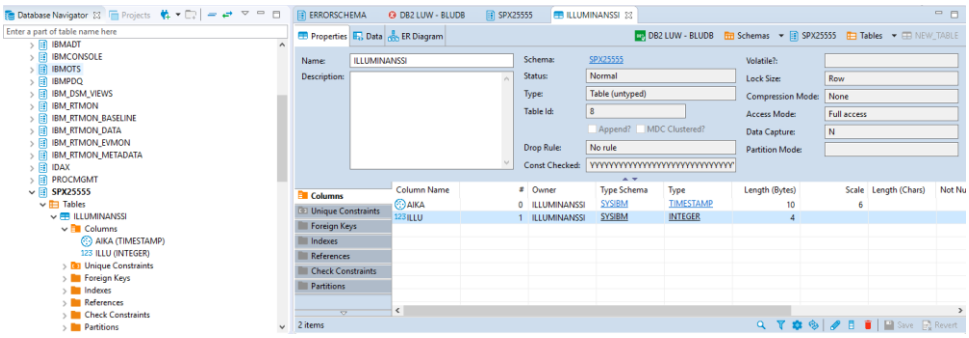
Uuden toteutuksen tutkiminen aloitettiin IBM Watson Platformin sisältä. Monessa dokumentaatiossa mainittiin mahdollisuus käyttää RESP API -komentoja tässä ympäristössä. Ei kuitenkaan löytynyt tapaa tunnistautua järjestelmään suoraan pelkkien REST-komentojen kautta. Datan suora lähettäminen ympäristöön taas aina tarvitsi jonkinlaisen tunnistautumisen. Hankaluuksien takia päätettiin lähteä etsimään ratkaisua muista IBM:n tarjoamista pilvipalveluista.

IBM Db2 Cloud osoittautui tarkoitukseen sopivaksi palveluksi. Db2 on IBM:n relaatiotietokanta pilvessä. Jotta tietokantaan voisi puskea sensoridataa, piti ensin luoda halutun muotoiset taulut ja kolumnit. IBM:n ohjeiden mukaan tämä pitäisi onnistua palvelun ilmaisversiolla, joka on varattu Dallasin palvelimilta. Kirjautumaan hallintatyökaluun pääsi palveluun avatuilla valtuutustiedoilla (Kuva 5).



Kuva 5. Valtuutuksien luominen

Kaikki toiminnallisuus tulisi toimia web-käyttöliittymän kautta. Näin ei kuitenkaan ollut. Konsoliin tarkoitettu linkki ei ollut näkyvissä siellä, missä se ohjeissa oli, eikä sitä löytynyt muualtakaan. Ongelman pääsi ohittamaan kokeilemalla muutamia web-osoitteita, joita näkyi ohjevideoissa. Uusien taulujen luominen osoittautui myös mahdottomaksi, koska jostain syystä ilmaisversion käyttäjällä ei ollut oikeuksia luoda uusia tauluja web-käyttöliittymän kautta. Tähän ongelmaan löytyi ratkaisu kolmannen osapuolen ohjelman kautta. Tietokantaan yritettiin kirjautua DBeaver-tietokantahallintaohjelmalla. Ohjelma vaati IBM Db2 -tulkkauspaketin, mikä piti löytyä heidän ylläpitämiltään pakettienlatausverkkosivulta. Tämäkin osoittautui hankalaksi ja täytyikin turvautua ulkopuolisen sivuston tarjoamaan tulkkauspakettiin. Vaikka samaa ilmaiskäyttäjää käytettiin, tietokantahallintaohjelmalla päästiin kirjautumaan sisään IBM Db2 -tietokantaan, sekä luomaan uusia tauluja ja kolumneja (Kuva 6).



Kuva 6. Tietokantahallintatyökalu

Seuraavaksi muodostettiin tarvittavat REST API -komennot datan tietokantaan tallentamista varten. IBM Db2 Cloud REST API -käyttöoppaista löydettiin erittäin kattavat ohjeet millaisessa muodossa komennot tulisi muodostaa. Aluksi luultiin, että tarvittava komento olisi suora datan latauskomento. Tämä komento oli kuitenkin tarkoitettu käytettäväksi kokonaisten tiedostojen lataamiseen, parsimiseen ja tallentamiseen tietokantaan. Tämä voisi olla hyödyllinen ominaisuus ympäristössä, missä haluttaisiin kerätä dataa ensin paikalliselle palvelimelle ja vasta myöhemmin liittää yhteiseen datakollaasiin. Pienellä muistilla varustettu Arduino ei voi varastoida sensoridataa kovin suuria määriä ja tarkoituksena oli saada reaaliaikaista dataa nähtäväksi käyttöliittymässä. Olikin parempi löytää ratkaisu, jossa data lähetetään välittömästi tietokantaan. Tähän sopiva REST-komento oli sisällyttää itse SQL-tallennusviesti komennon sisälle. Ensimmäiseksi tarvittiin ylläpitäjävaltuutusavainkyselykomento, jolla saatiin turva-avain. Tällä avaimella voitiin lähettää REST POST -komentoja, jotka sisälsivät haluttu data SQL-tallennusviestin sisällä (Kuva 7).

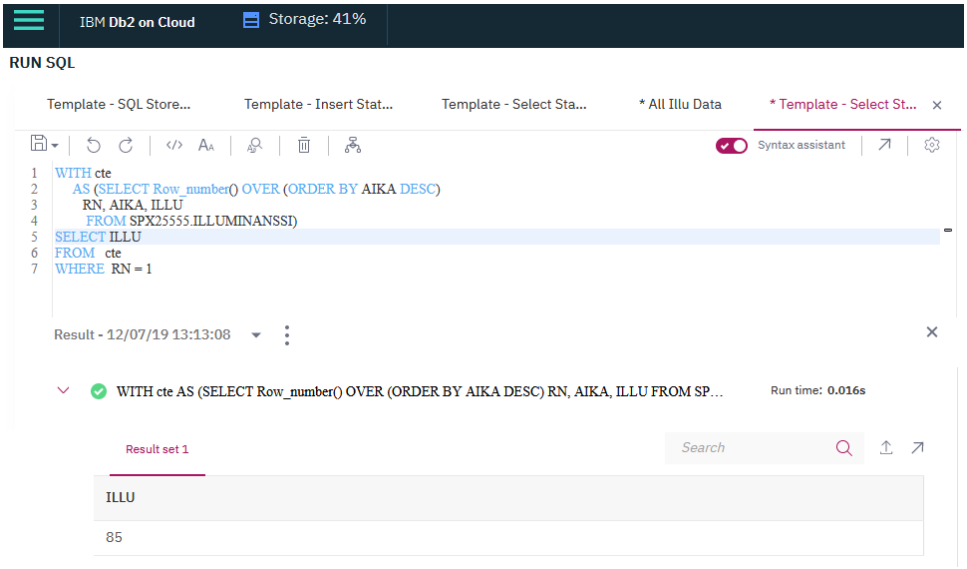
```
C:\Users\Awazir>curl -X POST https://dashdb-txn-sbox-yp-dal09-03.service
s.dal.bluemix.net/dbapi/v4/auth/tokens -H "content-type: application/jso
n" -d "{  \"userid\": \"spx25555\",  \"password\": \"90qnx1rdpgf0q2@m\"}
"
{"userid":"spx25555","token":"eyJraWQiOiJXVEkxNTc1MDI5NDk1NTAwUWxvK2ZHeX
hRV05UVVQ4KldlWFBTVWNObGxaQ1NRTVFpd25vVS1MbC02ZG5CMGxuSVBWTno2UXAxSFN1QW
l1SG44aXJ4SmtBSEoxUk9rcjlWN0FlLUVQZnlmZDJDRXpTM0llNWhQVEs4RnB4OWhKa1E2Zl
J4Nk5WV1lHVVhFMzVaVUdYQkVGa0VjUkFZTFMwLVc2cDV0TC1IRFZSekV6cmhEd01rNXItLW
FTd3lhVGwwT0M4TU5kZjVxWkZOcFRvdnhYbDhzQnVZZkwrQTBwMkhQM1dCU0lIK3FvTFM5LT
NmZnF0RjFQd2FxUmlUNmxGIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.eyJpc3MiOiJ
odHRwczovL2xvY2FsaG9zdDo0NDMiLCJnZW5lcmF0ZVRpbWVzdGFtcCI6IjE1NzU5MjkwMTU
wNzYiLCJleHAiOjE1NzU5NTc4MTUsImV4cGlyZWRUaW1lIjoiMTU3NTk1NzgxNTA3NiIsInV
zZXJuYW1lIjoic3B4MjU1NTUifQ.Njc8sDeUO0xyDZB_iwLjmA8ieFnI4G3IrkOUYoKCPFxZ
juYfiN8kH8ufDk1nTK6KK6W_6vGFv_D9GfVdiutEfZz3RaqbvOj_0G8IX_RvqmlPFP4AKqQI
s2fLJmVU373GoFsp-ph1lGreeKEim7OGuusJny6f5-eSvJ1GLJJignM"}

C:\Users\Awazir>curl -X POST https://dashdb-txn-sbox-yp-dal09-03.service
s.dal.bluemix.net/dbapi/v4/sql_jobs -H "authorization: Bearer eyJraWQiOi
JXVEkxNTc1MDI5NDk1NTAwUWxvK2ZHeXhRV05UVVQ4KldlWFBTVWNObGxaQ1NRTVFpd25vVS
1MbC02ZG5CMGxuSVBWTno2UXAxSFN1QWl1SG44aXJ4SmtBSEoxUk9rcjlWN0FlLUVQZnlmZD
JDRXpTM0llNWhQVEs4RnB4OWhKa1E2ZlJ4Nk5WV1lHVVhFMzVaVUdYQkVGa0VjUkFZTFMwLV
c2cDV0TC1IRFZSekV6cmhEd01rNXItLWFTd3lhVGwwT0M4TU5kZjVxWkZOcFRvdnhYbDhzQn
VZZkwrQTBwMkhQM1dCU0lIK3FvTFM5LTNmZnF0RjFQd2FxUmlUNmxGIiwidHlwIjoiSldUIi
wiYWxnIjoiUlMyNTYifQ.eyJpc3MiOiJodHRwczovL2xvY2FsaG9zdDo0NDMiLCJnZW5lcmF
0ZVRpbWVzdGFtcCI6IjE1NzU5MjkwMTUwNzYiLCJleHAiOjE1NzU5NTc4MTUsImV4cGlyZWR
UaW1lIjoiMTU3NTk1NzgxNTA3NiIsInVzZXJuYW1lIjoic3B4MjU1NTUifQ.Njc8sDeUO0xy
DZB_iwLjmA8ieFnI4G3IrkOUYoKCPFxZjuYfiN8kH8ufDk1nTK6KK6W_6vGFv_D9GfVdiutE
fZz3RaqbvOj_0G8IX_RvqmlPFP4AKqQIs2fLJmVU373GoFsp-ph1lGreeKEim7OGuusJny6f
5-eSvJ1GLJJignM" -H "content-type: application/json" -d "{\"commands\":\
"INSERT INTO SPX25555.ILLUMINANSSI (AIKA, ILLU) VALUES ('2019-12-10 00:0
4:00', 25);\",\"separator\":\";\",\"stop_on_error\":\"yes\"}"

{"commands_count":1,"database":"BLUDB","instance":"db2inst1","limit":100
0,"host":"10.120.72.10","start":1575929080884,"id":"1575929076908_211401
3841","type":"JDBC","run_options":{"current_path":"","current_schema":""
,"success_action":"COMMIT","failure_action":"STOP","type":"JDBC","separa
tor":";","current_sql_id":"","reuse_connection":"no"},"user":"spx25555",
"commands":["INSERT INTO SPX25555.ILLUMINANSSI (AIKA, ILLU) VALUES ('201
9-12-10 00:04:00', 25)"]}
```

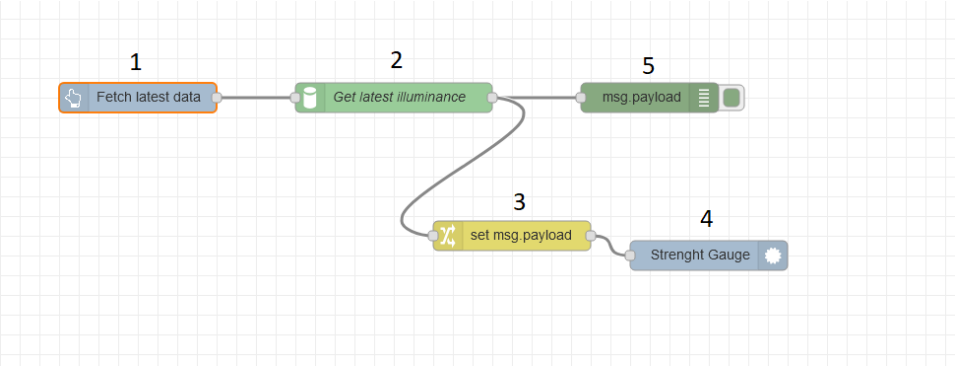
Kuva 7. REST API -todennusavainkysely, todennusavain vastaus, datan tallennusviesti ja tallennus työn numero vastauksena.

Tallennusviestikomennon paluuviestin sisältä saadaan aloitetun työn tunnistenumero, jota voidaan uudella REST-komennolla kutsua. Näin voidaan tarkistaa missä vaiheessa tallennusprosessi on ja onko se onnistunut. Tämä on tarpeellinen komento, kun rakennetaan sensoriympäristöön nopeasti ajavaa koodia. Onnistuneen datatallennuksen jälkeen voitiin suorittaa SQL-kysely tallennetulle datalle, jotta voitiin varmistaa, että data tallentui kolumneihin halutulla tavalla. Tämä onnistuttiin testaamaan IBM Db2 -käyttöliittymän kautta (Kuva 8).

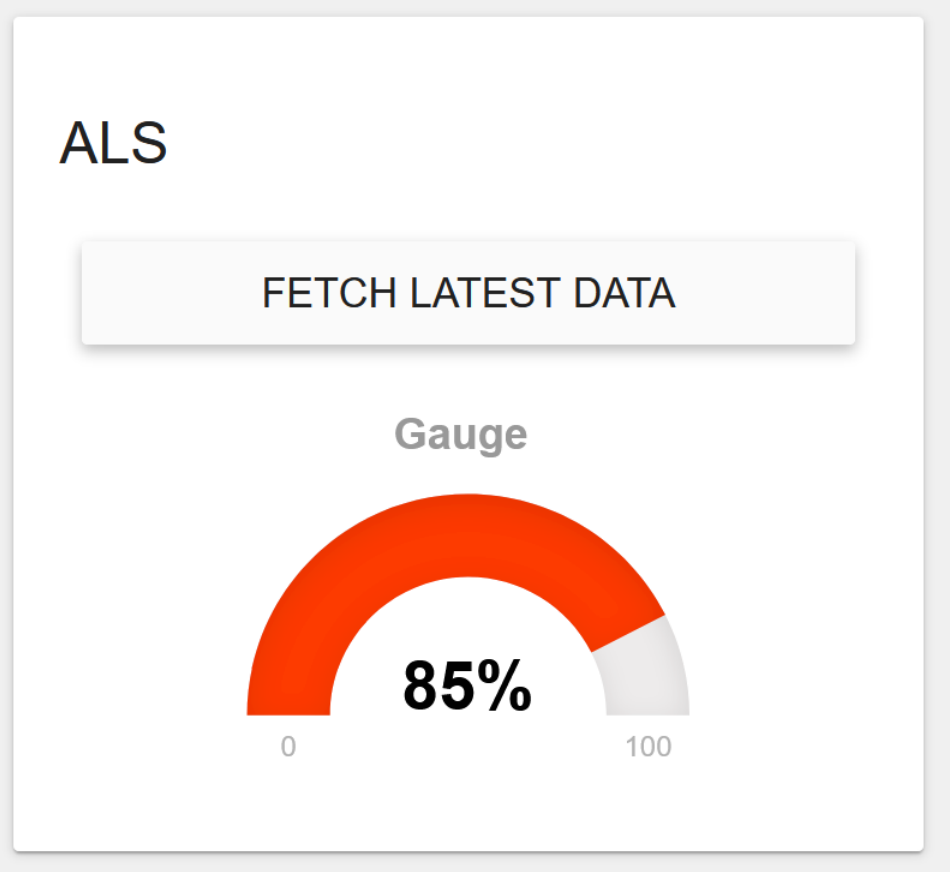


Kuva 8. Datan haku SQL-komennolla

Tallennetun datan esittäminen NodeRED-käyttöliittymässä onnistui kätevästi, koska ympäristö sisältää valmiit työkalut datan hakemiseen Db2-tietokannasta. Käyttöliittymään rakennettiin nappi, joka kutsuu tietokannasta viimeisimmän sensoriviestin ja poimii siitä valoisuuden arvon (Kuva 9). Data esitettiin käyttöliittymässä prosentteja esittävässä mittarissa (Kuva 10).



Kuva 9. NodeRED-funktiokaavio datan hakemiselle Db2-tietokannasta. 1. käyttöliittymän nappi, 2. SQL-datahakukomento, 3. JSON-viestin jäsentely, 4. käyttöliittymän mittari ja 5. debuggaus.



Kuva 10. NodeRED-käyttöliittymä Db2-tietokantadatalle

Tiedonkuljetusympäristön ollessa valmis, voitiin muuttaa Arduino lähettämään automaattisesti sensoridata tietokantaan (Liite 2). Arduinon käyttämästä koodista pystyttiin siivoamaan tarpeeton MQTT-kommunikaatiota koskettava koodi. Arduino-kirjastoista ladattiin ArduinoHttpClient.h-paketti, jolla voitaisiin lähettää REST-viestejä. Koodiin lisättiin tarvittavat REST API -alustukset ja jatkuvaan ajosilmukkaan lisättiin kellontarkastus, ylläpitäjävaltuutusavainkyselykomento ja datan tietokantaan tallennuskomento (Kuva 11 ja 12).

```
void loop() {

  //Get auth tokens if needed
  if (statusCode == 401){
  Serial.println("making POST request with HTTP basic authentication");
  client.beginRequest();
  client.post("/dbapi/v4/auth/tokens");
  client.sendHeader("content-type", "application/json");
  client.sendBasicAuth("spx25555", "90qnxlrdpgf0q2@m");
  client.endRequest();

  // read the status code and body of the response
  int statusCode = client.responseStatusCode();
  String response = client.responseBody();
  authToken = response;

  Serial.print("Status code: ");
  Serial.println(statusCode);
  Serial.print("Response: ");
  Serial.println(response);
  }
```

Kuva 11. Jos edellinen viesti sai vastaukseksi koodin 401, joka tarkoittaa ei oikeuksia, Arduinon REST API -autentikointiavainkysely suoritetaan.

```
// publish a message every  180 second.
 if(millis() - lastIBMMillis > 180000)
 {
  Serial.println("Publishing to IBM Db2...");
  lastIBMMillis = millis();
  get_ntp_time();
  String contentType = "application/json";
  String postData = "\"commands\":\"INSERT INTO SPX25555.ILLUMINANSSI (AIKA, ILLU) VALUES ('";
  postData += String(ntpTime) +"', " + String(int(alsScaledL));
  postData += ");\",\"separator\":\";\",\"stop_on_error\":\"yes\"";

  client.beginRequest();
  client.post("/dbapi/v4/sql_jobs", contentType, postData);
  client.sendHeader("authorization", "Bearer "+authToken); // authentication with Bearer token
  client.endRequest();

  // read the status code and body of the response
  int statusCode = client.responseStatusCode();
  String response = client.responseBody();
```

Kuva 12. Arduinon REST API -datatallennusviestin muodostus ja lähetys

5 JOHTOPÄÄTÖKSET JA POHDINTA

IoT-järjestelmissä käytetään MQTT-protokollaa, jonka mukainen sanoma on erittäin yksinkertainen. MQTT-sanomassa hyötydatan määrä suhteessa protokollan mukaisen yhteyden luonnin, yhteyden ylläpidon, sanomien reitityksen ja sanomien virheettömyyden tarkistuksen tarvitsemaan ns. overhead datan määrään on suuri. Näin MQTT-protokollan sanomien muodostus, lähetys ja vastaanotto käyttävät erittäin vähän energiaa. Vähäinen energiankulutus on tärkeää juuri IoT-anturilaitteiden toteutuksessa. (Wang, 2018)

REST API -sanomat http-protokollan yli sopivat IoT-järjestelmissä Gateway-laitteiden ja IoT-alustan väliseen kommunikaatioon. IoT Gateway vastaanottaa MQTT-sanomia anturilaitteilta, käsittelee ja luokittelee mittausarvoja paikallisesti sekä lähettää käsitellyn tiedon pilvipalveluna toteutetulle IoT-alustalle. Gateway toimii myös järjestelmän ylläpidossa. Se voi ylläpitää tietoa anturilaitteiden toimintakunnosta. Tässäkin anturilaiteen ja Gatewayn välinen yhteys voi olla toteutettu MQTT-protokollalla ja Gatewayn ja IoT-alustan välinen yhteys voi olla toteutettu REST API -sanomina HTPP -protokollalla.

5.1 Projektin eteneminen

Työ ei edennyt alkuperäisen aikataulusuunnitelman mukaisesti. Hankaluuksia aiheutti IBM Watson Platform -ympäristön suoraan käyttäminen REST API -kommunikaatiolla. Projektin aikana ei onnistuttu löytämään tapaa, jossa laitetta ei tarvitsisi autentikoida MQTT-kommunikaation tunnistautumisprotokollilla. Monesta palvelusta oli yllättävän hankalaa ja aikaa vievää löytää ajan tasalla olevaa dokumentaatiota IBM Cloudin info- ja käyttöopassivulta. Useasti törmättiin myös tilanteeseen, jossa ilmaisen testiversion toiminnallisuudet eivät enää vastanneet mitään ohjetta. Hidastusta järjestelmien testaamiseen aiheutti myös IBM-palveluiden kaatuilu. Yhden kerran järjestelmä oli pois käytöstä reilun viikon.

Ongelmista huolimatta järjestelmälle löydettiin vaihtoehtoinen toimintaperiaate, joka myös toteutti yhden jatkokehityssuunnitelmista. Datan tallentaminen suoraan tietokantaan ja tietokannan käyttämistä NodeRED-käyttöliittymän viittauskohteena eivät olleet työn alkuperäinen ajatus, mutta todettiin, että tällä ratkaisulla saavutettaisiin haluttu toiminnallisuus. Projektissa onnistuttiin vaikeuksista huolimatta hyvin ja saatiin todennettua yksi tapa, miten IBM Cloud -ympäristössä voisi sensoridataa hallita.

5.2 **Kehitysmahdollisuudet ja käyttöönotto**

Järjestelmä saatiin toimimaan hyvin Arduinolla, joka kykenee itse tekemään suuren osan päätöksistä koodillisesti, milloin data kuuluu lähettää kirjattavaksi tietokantaan. Hämeen ammattikorkeakouluun käyttöön tulevat sensorilaitteen vaativat, että jokin IoT-alusta tai -yhdyskäytävä käy kyselemässä datan niiltä. Arduino voisi toimittaa myös IoT-yhdyskäytävän virkaa lisäämällä koodiin uuden REST GET -komennon, joka hakisi ulkopuoliselta sensorilta datan ja toimittaisi sen edelleen eteenpäin tietokantaan. Arduino ei kuitenkin sovellu tähän tehtävään kovinkaan hyvin, varsinkaan sensorimäärän kasvaessa. IoT-yhdyskäytävän voisikin esimerkiksi rakentaa Raspberry Pi -tietokoneella.

Työssä käytettiin yhtä sensoria ja sen uusinta dataa esitettiin NodeRED-käyttöliittymässä. Sensorimäärän lisääntyessä ja datan muuttuessa monipuolisemmaksi Db2-tietokantaan voisi rakentaa erilaisia näkymiä. Näillä voitaisiin esimerkiksi muodostaa vuorokausien keskiarvotaulukko kaikilta sensoreilta tai poimia kunkin sensorin ääriarvot.

NodeRED-käyttöliittymään täytyisi lisätä jonkinlainen suojaus, koska asennettaessa se on suojaamaton. Kuka tahansa, joka sattuu löytämään web-osoitteen, pääsee painelemaan sinne rakennettuja nappeja. Salauksen tekeminen hoituisi NodeRED-lähdekoodia muuttamalla. Kaikki työssä käytetyt salasanat jäivät koodiin avoimeksi tekstiksi ja olisikin hyvä saada jonkinlainen kryptaus kaikille salasanoille mitä käytetään.

LÄHTEET

Amazon. (2019). What is NoSQL? Julkaistu 6.12.2019. Haettu 15.12.2019 osoitteesta https://aws.amazon.com/nosql/

Arm. (n.d.). What Are IoT Devices? Haettu 16.12.2019 osoitteesta https://www.arm.com/glossary/iot-devices

Dhaker, P. (2018). Introduction to SPI Interface. Julkaistu 13.8.2018. Haettu 10.12.2019 osoitteesta https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html#

Digicert. (n.d.). What is an SSL Certificate? Haettu 10.12.2019 osoitteesta https://www.websecurity.symantec.com/security-topics/what-is-ssl-tls-https

Digilent. (2016). PmodALS™ Reference Manual. Julkaistu 15.4.2016. Haettu 9.12.2019 osoitteesta https://reference.digilentinc.com/_media/reference/pmod/pmodals/pmodals_rm.pdf

Guru99. (n.d.). What is Database? What is SQL? Haettu 9.12.2019 osoitteesta https://www.guru99.com/introduction-to-database-sql.html

IBM. (2019). IBM Cloudant. Julkaistu 15.11.2019. Haettu 15.12.2019 osoitteesta https://cloud.ibm.com/docs/services/Cloudant?topic=cloudant-overview

Kaaproject. (n.d.). What is an IoT platform? Haettu 16.12.2019 osoitteesta https://www.kaaproject.org/what-is-iot-platform

OAS. (n.d.). What is an IoT Gateway? Haettu 16.12.2019 osoitteesta https://openautomationsoftware.com/blog/what-is-an-iot-gateway/

Rouse, M. (2019). RESTful API (REST API). Julkaisu päivitetty 25.6.2019. Haettu 10.12.2019 osoitteesta https://searchmicroservices.techtarget.com/definition/RESTful-API

Sparkfun. (n.d.). Serial Communication. Haettu 7.12.2019 osoitteesta https://learn.sparkfun.com/tutorials/serial-communication/all

Stephen, M. (2019). Introduction to CAN (Controller Area Network). Julkaistu 19.1.2019. Haettu 15.12.2019 osoitteesta https://www.allaboutcircuits.com/technical-articles/introduction-to-can-controller-area-network/

Wang, C. (2018). HTTP vs. MQTT: A tale of two IoT protocols. Julkaistu 26.11.2018. Haettu 15.12.2019 osoitteesta https://cloud.google.com/blog/products/iot-devices/http-vs-mqtt-a-tale-of-two-iot-protocols

MQTT Arduino -koodi

```
/*
MKR1000 connecting to IBM Watson IoT Platform

Based on documentation and "recipes" on IBM Bluemix
https://www.ibm.com/cloud-computing/bluemix/watson
Timo Karppinen 19.2.2017

Modified for testing SPI Ambient light sensor board Digilent PmodALS
Please connect
MKR1000 - PmodALS
GND - 5 GND
Vcc - 6 Vcc
9 SCK - 4 SCK
10 MISO - 3 MISO
2 - 1 SS

ALS data on the SPI
D15 ... D13 - three zeros
D12 ... D04 - 8 bits of ambient light data
D03 ... D00 - four zeros

Details on the connection of ADC chip on ALS board are given
http://www.ti.com/lit/ds/symlink/adc081s021.pdf
Timo Karppinen 25.1.2018
 */

#include <SPI.h>
#include <WiFi101.h>
#include <WiFiSSLClient.h>
#include <MQTTClient.h>

// WLAN
//char ssid[] = "HAMKvisitor"; //  your network SSID (name)
//char pass[] = "xxxxxxxxxx";    // your network password (use for WPA)

int status = WL_IDLE_STATUS;

// IBM Watson
// Your organization and device needs to be registered in IBM Watson IoT Platform.
// Instruction for registering on page
// https://internetofthings.ibmcloud.com/#

//char *client_id = "d:<your Organization ID>:<your Device Type>:<your Device ID>";
char *client_id = "d:pps5im:A_DeskTop_Editor:1234";
char *user_id = "use-token-auth";   // telling that authentication will be done with token
```

Liite 1/2

```
char *authToken = "test1234"; // Your IBM Watson Authentication Token

//char *ibm_hostname = “your-org-id.messaging.internetofthings.ibmcloud.com”;
char *ibm_hostname = "pps5im.messaging.internetofthings.ibmcloud.com";

// sensors and LEDS
const int LEDPin = LED_BUILTIN;       // must be a pin that supports PWM. 0...8 on MKR1000
// PModTC1
const int alsCS = 2;       // chip select for sensor SPI communication
int alsByte0 = 0;          // 8 bit data from sensor board
int alsByte1 = 0;          // 8 bit data from sensor board
int als8bit = 0;
int alsRaw = 0;
float alsScaledF = 0;


int blinkState = 0;

/*use this class if you connect using SSL
 * WiFiSSLClient net;
*/
WiFiClient net;
MQTTClient MQTTc;

unsigned long lastSampleMillis = 0;
unsigned long previousWiFiBeginMillis = 0;
unsigned long lastWatsonMillis = 0;
unsigned long lastPrintMillis = 0;


void setup()
{
 Serial.begin(9600);
 delay(2000); // Wait for wifi unit to power up

 while (!Serial) {
  ; // wait for serial port to connect. Needed for native USB port only
 }

 // check for the presence of the shield:
 if (WiFi.status() == WL_NO_SHIELD) {
  Serial.println("WiFi shield not present");
  // don't continue:
  while (true);
 }
```

Liite 1/3

```
  // attempt to connect to WiFi network:
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }
  // you're connected now, so print out the status:
  printWiFiStatus();

  /*
    client.begin("<Address Watson IOT>", 1883, net);
    Address Watson IOT: <WatsonIOTOrganizationID>.messaging.inter-
netofthings.ibmcloud.com
    Example:
    client.begin("iqwckl.messaging.internetofthings.ibmcloud.com", 1883, net);
  */
  //MQTTc.begin(ibm_hostname, 1883, net);  // Cut for testing without Watson
  MQTTc.begin("193.167.167.59", 1883, net);  // Cut for testing without Watson
  connect();


  SPI.begin();
  // Set up the I/O pins
  pinMode(alsCS, OUTPUT);
  digitalWrite(alsCS, HIGH);   // for not communicating with MIC3 at the moment

  pinMode(LEDPin, OUTPUT);


}

void loop() {
   MQTTc.loop();  // Cut for testing without Watson


  // opening and closing SPI communication for reading sensor
  if(millis() - lastSampleMillis > 500)
  {
    lastSampleMillis = millis();
    SPI.beginTransaction(SPISettings(14000000, MSBFIRST, SPI_MODE0));
    digitalWrite(alsCS, LOW);
```

```
  alsByte0 = SPI.transfer(0x00);
  alsByte1 = SPI.transfer(0x00);

  digitalWrite(alsCS, HIGH);
  SPI.endTransaction();


  alsByte0 = alsByte0 << 4;
  alsByte1 = alsByte1 >> 4;


  als8bit =( alsByte0 | alsByte1 );

  alsRaw = als8bit; //
  alsScaledF = float(alsRaw)/4;

 }

 // Print on serial monitor once in 1000 millisecond
 if(millis() - lastPrintMillis > 1000)
 {
  Serial.print("als8bit   ");
  Serial.println(als8bit, BIN);
  Serial.print("  alsRaw  ");
  Serial.println(alsRaw);
  Serial.print("  alsScaledF  ");
  Serial.println(alsScaledF);

  lastPrintMillis = millis();
 }

 // publish a message every  180 second.
 if(millis() - lastWatsonMillis > 180000)
 {
 Serial.println("Publishing to Watson...");
 if(!MQTTc.connected()) {    // Cut for testing without Watson
  connect();              // Cut for testing without Watson
 }                       // Cut for testing without Watson
 lastWatsonMillis = millis();
 //Cut for testing without Watson
 // MQTTc.publish("iot-2/evt/TemperatureTC1/fmt/json", "{\"d\":{\"TemperatureSen-
sor\":\"TC1     \",\"TempScaledF3AC\":\""     +     String(tempScaledF)+"\",     \"Temp-
StreightDF48\": \"" + String(temp14bit)+"\"}}");
```

```
 //ok    MQTTc.publish("iot-2/evt/TemperatureTC1/fmt/json", "{\"d\":{\"Temperature-
Sensor\":\"TC1    \",\"TempScaledF3AC\":12345,    \"TempStreightDF48\":    \""    +
String(temp14bit)+"\"}}");
 //not   MQTTc.publish("iot-2/evt/TemperatureTC1/fmt/json", "{\"d\":{\"Temperature-
Sensor\":\"TC1 \",\"TempScaledF3AC\":float(tempScaledF), \"TempStreightDF48\": \""
+ String(temp14bit)+"\"}}");
 //ok    MQTTc.publish("iot-2/evt/TemperatureTC1/fmt/json", "{\"d\":{\"Temperature-
Sensor\":\"TC1    \",\"TempScaledF3AC\":34.567,    \"TempStreightDF48\":    \""    +
String(temp14bit)+"\"}}");
 //     char payload = "{\"d\":{\"TemperatureSensor\":\"TC1 \",\"TempScaledF3AC\":\""
+ String(tempScaledF)+"\", \"TempStreightDF48\": \"" + String(temp14bit)+"\"}}";
 //ok          String wpayload = "{\"d\":{\"TemperatureSensor\":\"TC1  \",\"Temp-
ScaledF3AC\":1234, \"TempStreightDF48\":5678}}";
 //ok          String wpayload = "{\"d\":{\"TemperatureSensor\":\"TC1  \",\"Temp-
ScaledF3AC\":123.4, \"TempStreightDF48\":567.8}}";
 //                      String wpayload = "{\"d\":{\"AmbientLightSensor\":\"ALS1
\",\"ALSScalednnnn\":"    +    String(alsScaledF)+    ",    \"ALSStreightnnnn\":"    +
String(alsRaw)+"}}";
 String wpayload = String(alsScaledF);

 //     MQTTc.publish("iot-2/evt/AmbientLightALS1/fmt/json", wpayload);
 MQTTc.publish("Illu_MKR", wpayload);
   }

  delay(1);

// end of loop
}

void connect()
{
 Serial.print("checking WLAN...");
 while (status != WL_CONNECTED)
 {
  Serial.print(".");      // printing a dot every half second
  if ( millis() - previousWiFiBeginMillis > 5000) // reconnecting
  {
   previousWiFiBeginMillis = millis();
   WiFi.begin(ssid, pass);
   delay(5000); // Wait for WiFi to connect
   Serial.println("Connected to WLAN");
   printWiFiStatus();
  }
  delay(500);

 }
```

```
 /*
   Example:
   MQTTc.connect("d:iqwckl:arduino:oxigenarbpm","use-token-
auth","90wT2?a*1WAMVJStb1")

   Documentation:
   https://console.ng.bluemix.net/docs/services/IoT/iotplatform_task.html#iotplat-
form_task
 */

 Serial.print("\nconnecting Watson with MQTT....");
 // Cut for testing without Watson
while (!MQTTc.connect(client_id,user_id,authToken))
 {
  Serial.print(".");
  delay(30000); // try again in thirty seconds
 }
 Serial.println("\nconnected!");
}

// messageReceived subroutine needs to be here. MQTT client is calling it.
void messageReceived(String topic, String payload, char * bytes, unsigned int length) {
 Serial.print("incoming: ");
 Serial.print(topic);
 Serial.print(" - ");
 Serial.print(payload);
 Serial.println();
}

void printWiFiStatus() {
 // print the SSID of the network you're attached to:
 Serial.print("SSID: ");
 Serial.println(WiFi.SSID());

 // print your WiFi shield's IP address:
 IPAddress ip = WiFi.localIP();
 Serial.print("IP Address: ");
 Serial.println(ip);

 // print the received signal strength:
 long rssi = WiFi.RSSI();
 Serial.print("signal strength (RSSI):");
 Serial.print(rssi);
 Serial.println(" dBm");
}
```

Liite 2/1

REST API Arduino -koodi

```
// Import required libraries
#include <SPI.h>
#include <WiFi101.h>
#include <ArduinoHttpClient.h>
#include <aREST.h>
#include <WiFiUdp.h>
#include <time.h>

// Status
int status = WL_IDLE_STATUS;
int statusCode;
String authToken;
unsigned long ntpTime;

// Callback function for the cloud connection
void callback(char* topic, byte* payload, unsigned int length);

void get_ntp_time();

//IBM Db2 service
char *ibm_hostname = "https://dashdb-txn-sbox-yp-dal09-03.services.dal.blue-
mix.net";
int port = 50000;

const long utcOffsetInSeconds = 0;
// PModTC1
const int alsCS = 2;       // chip select for sensor SPI communication
int alsByte0 = 0;          // 8 bit data from sensor board
int alsByte1 = 0;          // 8 bit data from sensor board
int als8bit = 0;
int alsRaw = 0;
float alsScaledL = 0;
int blinkState = 0;

unsigned long lastSampleMillis = 0;
unsigned long previousWiFiBeginMillis = 0;
unsigned long lastIBMMillis = 0;
unsigned long lastPrintMillis = 0;

// local port to listen for UDP packets
unsigned int localPort = 2390;
IPAddress timeServer(216, 239, 35, 0); // Google NTP server
const int NTP_PACKET_SIZE = 48; // NTP time stamp is in the first 48 bytes of the mes-
sage
byte packetBuffer[ NTP_PACKET_SIZE]; //buffer to hold incoming and outgoing packets
```

```
// A UDP instance to let us send and receive packets over UDP
WiFiUDP Udp;
// The port to listen for incoming TCP connections
#define LISTEN_PORT          80

// Create an instance of the server
WiFiServer server(LISTEN_PORT);

// Clients
WiFiClient wifiClient;
HttpClient client = HttpClient(wifiClient, ibm_hostname, port);

// WiFi parameters
char ssid[] = "*****";
char password[] = "******";

// Variables to be exposed to the API
int temperature;
int humidity;

void setup(void)
{
  // Start Serial
  Serial.begin(115200);

  // Connect to WiFi
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    status = WiFi.begin(ssid, password);

    // Wait 10 seconds for connection:
    delay(10000);
  }
  Serial.println("WiFi connected");

  //Start UDP
  Udp.begin(localPort);

  SPI.begin();
  // Set up the I/O pins
  pinMode(alsCS, OUTPUT);
  digitalWrite(alsCS, HIGH);   // for not communicating with MIC3 at the moment
}

void loop() {
```

```
//Get auth tokens if needed
if (statusCode == 401) {
  Serial.println("making POST request with HTTP basic authentication");
  client.beginRequest();
  client.post("/dbapi/v4/auth/tokens");
  client.sendHeader("content-type", "application/json");
  client.sendBasicAuth("spx25555", "90qnx1rdpgf0q2@m");
  client.endRequest();

  // read the status code and body of the response
  int statusCode = client.responseStatusCode();
  String response = client.responseBody();
  authToken = response;

  Serial.print("Status code: ");
  Serial.println(statusCode);
  Serial.print("Response: ");
  Serial.println(response);
}
//Read illuminance
if (millis() - lastSampleMillis > 500)
{
  lastSampleMillis = millis();
  SPI.beginTransaction(SPISettings(14000000, MSBFIRST, SPI_MODE0));
  digitalWrite(alsCS, LOW);

  alsByte0 = SPI.transfer(0x00);
  alsByte1 = SPI.transfer(0x00);

  digitalWrite(alsCS, HIGH);
  SPI.endTransaction();

  alsByte0 = alsByte0 << 4;
  alsByte1 = alsByte1 >> 4;

  als8bit = ( alsByte0 | alsByte1 );
  alsRaw = als8bit; //
  alsScaledL = float(alsRaw) / 2.55;
}
// publish a message every  180 second.
if (millis() - lastIBMMillis > 180000)
{
  Serial.println("Publishing to IBM Db2...");
  lastIBMMillis = millis();
  get_ntp_time();
  String contentType = "application/json";
```

```
  String postData = "\"commands\":\"INSERT INTO SPX25555.ILLUMINANSSI (AIKA, ILLU) VALUES ('";
  postData += String(ntpTime) + "', " + String(int(alsScaledL));
  postData += ");\",\"separator\":\";\",\"stop_on_error\":\"yes\"";
  Serial.println("making POST request with HTTP basic authentication");
  client.beginRequest();
  client.post("/dbapi/v4/sql_jobs", contentType, postData);
  client.sendHeader("authorization", "Bearer " + authToken); // authentication with Bearer token
  client.endRequest();

  // read the status code and body of the response
  int statusCode = client.responseStatusCode();
  String response = client.responseBody();

  Serial.print("Status code: ");
  Serial.println(statusCode);
  Serial.print("Response: ");
  Serial.println(response);
 }

 delay(1);
 // end of loop
}

void get_ntp_time() {
 sendNTPpacket(timeServer); // send an NTP packet to a time server
 // wait to see if a reply is available
 delay(1000);
 if ( Udp.parsePacket() ) {
  Serial.println("packet received");
  // We've received a packet, read the data from it
  Udp.read(packetBuffer, NTP_PACKET_SIZE); // read the packet into the buffer

  //the timestamp starts at byte 40 of the received packet and is four bytes,
  // or two words, long. First, esxtract the two words:

  unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
  unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);
  // combine the four bytes (two words) into a long integer
  // this is NTP time (seconds since Jan 1 1900):
  unsigned long secsSince1900 = highWord << 16 | lowWord;
  Serial.print("Seconds since Jan 1 1900 = " );
  Serial.println(secsSince1900);
  // now convert NTP time into everyday time:
  Serial.print("Unix time = ");
```

```
    // Unix time starts on Jan 1 1970. In seconds, that's 2208988800:
    const unsigned long seventyYears = 2208988800UL;
    // subtract seventy years:
    unsigned long epoch = secsSince1900 - seventyYears;
    ntpTime = epoch;
    // print Unix time:
    Serial.println(epoch);

    // print the hour, minute and second:
    Serial.print("The UTC time is ");       // UTC is the time at Greenwich Meridian (GMT)
    Serial.print((epoch  % 86400L) / 3600); // print the hour (86400 equals secs per day)
    Serial.print(':');
    if ( ((epoch % 3600) / 60) < 10 ) {
      // In the first 10 minutes of each hour, we'll want a leading '0'
      Serial.print('0');
    }
    Serial.print((epoch  % 3600) / 60); // print the minute (3600 equals secs per minute)
    Serial.print(':');
    if ( (epoch % 60) < 10 ) {
      // In the first 10 seconds of each minute, we'll want a leading '0'
      Serial.print('0');
    }
    Serial.println(epoch % 60); // print the second
  }
  // wait ten seconds before asking for the time again
  delay(10000);
}

// send an NTP request to the time server at the given address
unsigned long sendNTPpacket(IPAddress& address)
{
  //Serial.println("1");
  // set all bytes in the buffer to 0
  memset(packetBuffer, 0, NTP_PACKET_SIZE);
  // Initialize values needed to form NTP request
  // (see URL above for details on the packets)
  //Serial.println("2");
  packetBuffer[0] = 0b11100011;   // LI, Version, Mode
  packetBuffer[1] = 0;     // Stratum, or type of clock
  packetBuffer[2] = 6;     // Polling Interval
  packetBuffer[3] = 0xEC;  // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  packetBuffer[12]  = 49;
  packetBuffer[13]  = 0x4E;
  packetBuffer[14]  = 49;
  packetBuffer[15]  = 52;
```

```
  //Serial.println("3");

  // all NTP fields have been given values, now
  // you can send a packet requesting a timestamp:
  Udp.beginPacket(address, 123); //NTP requests are to port 123
  //Serial.println("4");
  Udp.write(packetBuffer, NTP_PACKET_SIZE);
  //Serial.println("5");
  Udp.endPacket();
  //Serial.println("6");
}
```