IOT-DATAN MITTAUS JA KÄSITTELY PILVIPALVELUIDEN AVULLA

LAB-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikka,
Ohjelmistotekniikka
Insinööri (AMK)
Syksy 2020
Jani Vihervuori

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Vihervuori, Jani	Opinnäytetyö, AMK	Syksy 2020
	Sivumäärä	
	54	

Työn nimi

IoT-datan mittaus ja käsittely pilvipalveluiden avulla

Tutkinto

Tieto- ja viestintätekniikka, insinööri (AMK)

Tiivistelmä

Tavoitteena opinnäytetyössä oli luoda Internet of Things -kokonaisuus IoT-laitteen ja pilvipalveluiden avulla. Työssä käytettiin Arduino-yhteensopivaa IoT DevKit-laitetta sensoridatan mittaamisessa ja Azurea pilvipalveluiden tarjoajana.

Opinnäytetyössä ensin käydään teoriatasolla yleisesti läpi jokainen osa-alue ja tämän jälkeen esitellään itse projektin toteutus käytännössä; laitteisto, käytössä olevat pilvipalvelut ja koodiosuus näiden takana.

Pilvipalveluista käytettiin IoT Hubia keskeisenä pisteenä tiedonsiirrossa laitteesta tietokantaan, verkkosivulle sekä verkkosivulta laitteeseen. Yhtä Azure Funktiota käytettiin lisäämään IoT Hubiin saapuvaa dataa tietokantaan. Event Gridin avulla live dataa siirrettiin IoT Hubista verkkosivulle sekä Queue Storageen. Toista Azure Funktiota sekä Blob Storagea käytettiin hälytysten käsittelyssä ja tallettamisessa tietokantaan.

Kaikki komponentit toimivat suunnitellusti yhdessä, mutta vaativat kuitenkin lisätestausta erityisesti käytettäessä useita laitteita samanaikaisesti. Verkkosivulla datan visualisointi, live datan ja hälytysten seuranta sekä laitteiden etähallinta toimii hyvin, mutta vaatii jatkokehittelyä sekä visuaalisen ilmeen hienosäätöä.

Asiasanat

Arduino, IoT, Azure, pilvipalvelut, tietokanta

Abstract

Author(s)	Type of publication	Published
Vihervuori, Jani	Bachelor's thesis	Autumn 2020
	Number of pages	
	54	

Title of publication

IoT data collection and handling using cloud services

Name of Degree

Bachelor or Information and Communication Technology

Abstract

The aim of this thesis was to create an Internet of Things project using an IoT device and cloud services. Arduino compatible device IoT DevKit was used to collect telemetry and Azure worked as a cloud service provider.

The text starts by going though theoretic part of the project, explaining general information on each component, and then describing inner workings of the project, with device, cloud services in use and code behind each part.

IoT Hub was used as a central data transfer point between the device, a database, a web application and between the web application and the device. One Azure Function was used to store telemetry data into the database and another Azure Function with Blob Storage, were used to insert alerts into the database. To transfer live data from IoT Hub to the web application and to Queue Storage, Event Grid was implemented.

All components work together as intended but need more testing, especially when using multiple devices simultaneously. On the web application, visualizing data, monitoring live data and alerts and remote control of devices work well, but need further development and finetuning of visual design.

Keywords

Arduino, IoT, Azure, cloud, database

SISÄLLYS

1 JOHDANTO

Ensimmäinen pilvipalvelujen tarjoaja Salesforce.com vuonna 1999 tarjosi käyttäjille sovelluksia yksinkertaisen verkkosivun välityksellä. 2000-luvun alussa mukaan asettui Amazon omilla ratkaisuillaan ja tämän jälkeen Google. Koko 2000-luvun ajan pilvipalvelut ovat yleistyneet ja laajentuneet yhä isommiksi kokonaisuuksiksi. Pilvipalvelujen käyttö on lisääntynyt räjähdysmäisesti 2000-luvulla ja tällä hetkellä on nopeimmin kasvava sektori koko maailmassa. (Foote 2017.) Vuonna 2018 maailmanlaajuinen pilvipalvelujen liikevaihto (Yhdysvaltojen dollareina) oli 182,4 miljardia (Gartner 2019). Tänä vuonna (2020) sen ennustetaan olevan 257,87 miljardia ja seuraavana jo 306,95 miljardia (Gartner 2020). Markkinoita johtavat yritykset Amazon 32 %:n osuudella, Microsoft 18 %:n osuudella ja Google 8 %:n osuudella, ja ovat pitäneet suunnilleen samaa tasoa vuodesta 2017 lähtien (Statista 2020).

Nousujohteiseen suosioon on useita syitä. Pilvipalvelut mahdollistavat sovellusten käytön ilman paikallista asentamista. Tämä poistaa tarpeen hankkia ja ylläpitää fyysisiä laitteita, nopeuttaa sovelluskehitystä ja helpottaa ylläpitoa. Lisäksi skaalautuvuus, vakaus, virheensietokyky ja turvallisuus ovat isoja etuja verrattuna perinteisiin ratkaisuihin, ja pilvipalveluita on myös mahdollista käyttää mistä päin maailmaa tahansa. Pilvipalveluiden tarjoaja huolehtii, että palvelut ovat jatkuvasti saatavilla ja varmistaa palveluiden ylläpidon myös virhetilanteessa. Tämä maksimoi käyttöaikaa ja tuo käyttäjille vakaamman käyttökokemuksen. (Microsoft 2020l.)

Projekti toteutetaan Lahdessa sijaitsevalle Tekoälytiimille, joka toimii LAB-ammattikorkeakoulun teknologiayksikössä tekoälyprojektien ja Azure-pilvipalvelujen parissa. Projektin tavoitteena on toteuttaa datan mittaaminen Arduino-yhteensopivan laitteen avulla. Datan käsittelyyn ja tallentamiseen käytetään Azuren pilvipalveluita. Lisäksi tavoitteena on luoda verkkosivu datan visuaalista esitystä ja laitteen etähallintaa varten. Työssä käytetään aiemmin mainitun laitteen lisäksi pelkästään Microsoftin tarjoamia Azure pilvipalveluita. Pääkielenä toimii C# ja verkkosivussa myös JavaScript.

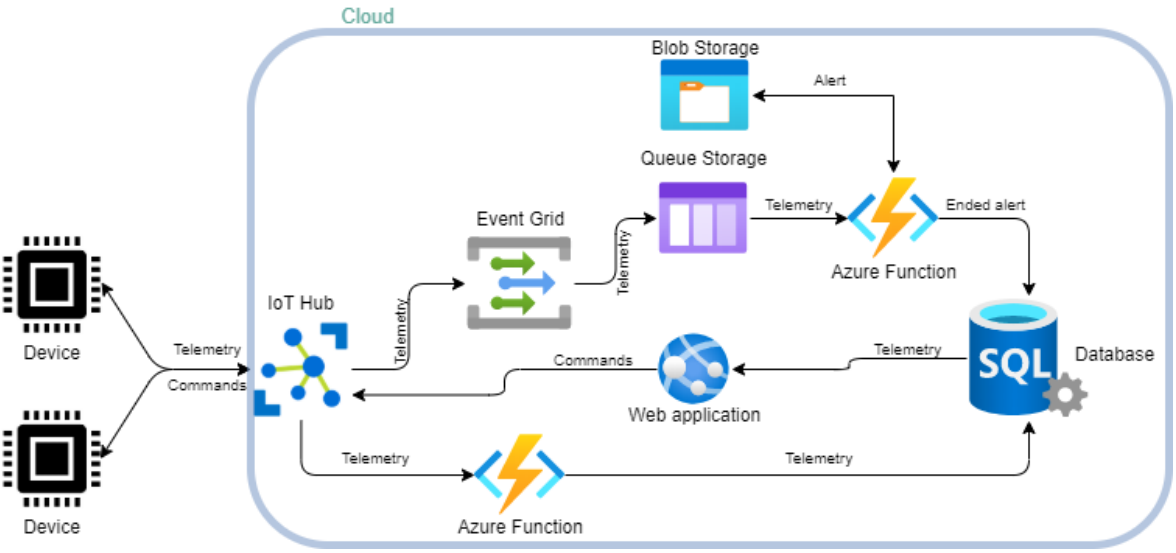
Opinnäytetyössä keskitytään pelkästään palvelinpuolen toimintaan ja jätetään verkkosivusto aihealueen ulkopuolelle. Verkkosivun toiminnasta mainitaan ainoastaan muihin komponentteihin liittyvä toiminnallisuus. Aluksi kerrotaan yleistä tietoa projektissa

käytössä olevista pilvipalveluista, esitellään tavoitteeseen tarvittavat komponentit teoriatasolla ja niiden toiminta yleisesti. Tämän jälkeen kerrotaan käytännön toimista itse projektin toteuttamisessa ja esitellään jokaisen osion taustalla toimiva koodi.

2 TOIMINTAYMPÄRISTÖ

2.1 YLEISTÄ

Projektissa tavoitteena oli toteuttaa seuraavan kuvan (Kuva 1) mukaisesti datan keräys laitteista, tallennus pilveen ja laitteiden etähallinta. Tässä osiossa kerrotaan jokaisen komponentin toiminnasta teoriatasolla.



Kuva 1. Projektin suunniteltu toteutus (Mukailtu: Microsoft 2020t)

2.2 ARDUINO

Arduino on suosittu avointa lähdekoodia käyttävä elektroniikka-alusta (Kuva 2) ja ohjelmointiympäristö. Mikroprosessorissa on liittimiä, joihin voi kytkeä ulkoisia komponentteja, kuten sensoreita ja moottoreita. Alun perin Arduino suunniteltiin Ivrea Interaction Design Instituten toimesta oppilaiden käyttöön ja on siitä lähtien kerännyt suosiota yleisessäkin käytössä. Arduinon ohjelmointiin on käytössä avoimen lähdekoodin kehitysympäristö Arduino IDE (Integrated Development Environment) ja kielenä on erikoistettu versio C++:sta. (Arduino 2020a.) Virallisia laitteita on olemassa erilaisilla kokoonpanoilla, jotka eroavat toisistaan komponenttien, muistin ja liittimien määrässä (Arduino 2020d).

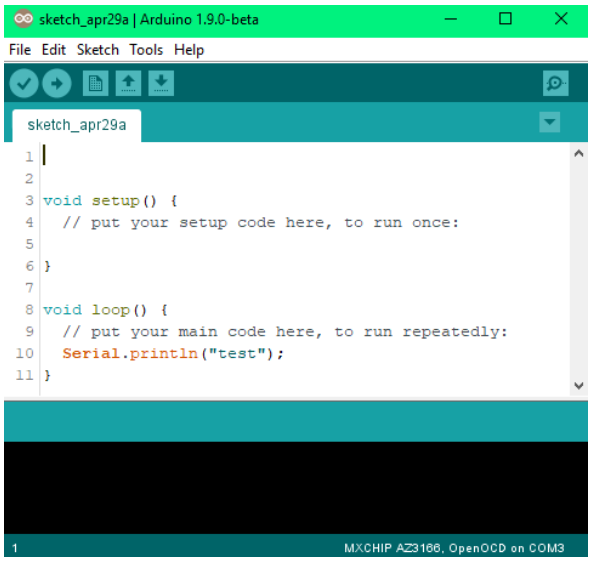
Kuva 2. Arduino Uno (Arduino 2020e)

Myynnissä virallisten Arduino -laitteiden lisäksi on Arduino-yhteensopivia epävirallisia tuotteita. Viralliset laitteet eroavat näistä siten, että ne on kehitetty erityisesti Arduino -tiimin kanssa yhteistyössä ja täten yhteensopivuus virallisten ohjelmistojen kanssa on varmistettu. Ainoastaan virallisilla laitteilla on oikeus käyttää ”Arduino” -nimitystä.

Laitteiden toiminnallisuutta voi laajentaa käyttämällä kirjastoja. Osa kirjastoista on valmiiksi asennettuina Arduino IDE:ssä, mutta niitä voi myös luoda itse tai ladata muista lähteistä. Kirjastojen yhteensopivuus vaihtelee laitteesta toiseen ja kaikki kirjastot eivät ole yhteensopivia edes virallisten Arduinojen kesken. (Arduino 2020f.)

Arduinojen ja yhteensopivien laitteiden toimintamalli on sama. Käytössä on kaksi pääfunktiota, joita laite suorittaa käynnistyksen jälkeen. Ensimmäinen setup() -funktio aktivoituu ainoastaan kerran laitteen käynnistyttyä ja sen tehtävänä on alustaa laitteisto. Tässä vaiheessa laitteen voi esimerkiksi yhdistää verkkoon tai initialisoida muuttujia, määrittää tarvittavat liitännät tai aloittaa kirjastojen käytön. (Arduino 2020b.) Setup() -funktion jälkeen laite alkaa suorittamaan loop() -funktiota, jota toistetaan virran sammuttamiseen asti. Loop() -funktiossa toteutetaan itse laitteen toiminta. (Arduino 2020c.)

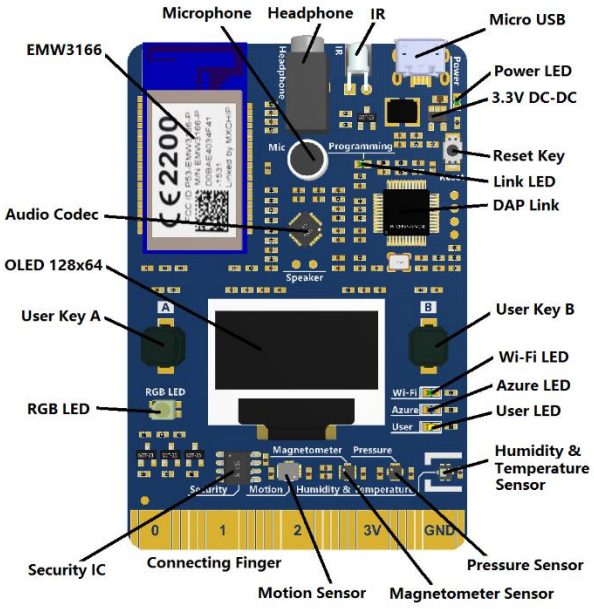
Seuraavassa kuvassa (Kuva 3) on näkymä Arduinon kehitysympäristöstä, jossa näkyvät pääfunktiot. Musta alue toimii tulostekenttänä.



Kuva 3. Arduino IDE

MXChip IoT DevKit

MXChip IoT DevKit (AZ3166) on MXChip -yrityksen kehittämä ja Microsoftin tukema Arduino-yhteensopiva laite, jossa on valmiiksi sisäänrakennettuna useita sensoreita ja muita yleisimmin käytettäviä komponentteja (Kuva 4). IoT Devkit tukee osaa yleisimmistä Arduinon virallisista kirjastoista, mutta siihen on saatavissa myös erityisesti sitä varten kehitetyt kirjastot. (Microsoft 2017a.)

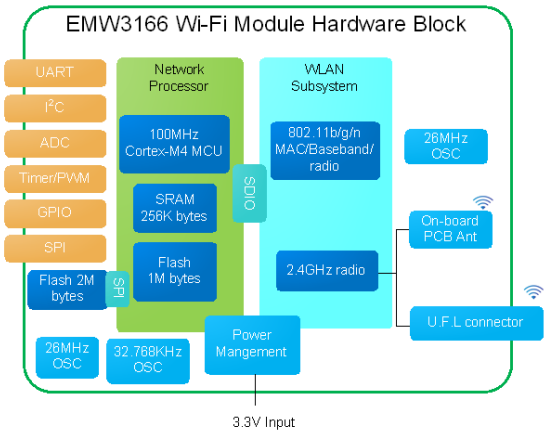


Kuva 4. IoT DevKitin sisäänrakennetut komponentit (MxChip 2019a)

Tekniset tiedot ovat

- ohjausyksikkö: EMW3166-a moduuli
- EMW3166 Wifi moduuli (256K SRAM,1M+2M Byte SPI Flash)
- virtalähteen jännite 3.3V DC-DC, maksimivirta 1,5A
- DAP Link Emulaattori
- MicroUsB
- mikrofoni ja kuulokeliitäntä
- 128x64 OLED näyttö
- 2 nappia
- 1 RGB valo
- salaussiru
- infrapunalähetin.

Alla olevassa kuvassa (Kuva 5) on pääohjausyksikkönä toimiva EMW3166 Wifi moduuli, johon on sisäänrakennettu ARM Cortex-M4 mikroprosessori. Prosessori sisältää 256 kilotavun SRAM -muistin sekä 1 megatavun flash -muistin. Lisäksi laitteeseen on liitettävissä myös toinen 2 megatavun flash muisti. Moduuli tukee 802.11 b/g/n yhteysstandardeja sekä WEP, WPA/WPA2 ja PSK -salauksia. Toimintalämpötila on -30 asteesta +85 asteeseen. (MxChip 2019b.)

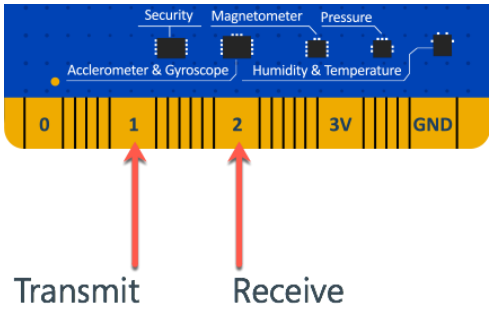


Kuva 5. EMW3166 Wifi moduuli

MXChip IoT DevKitissä on valmiina sisäänrakennettuna seuraavat sensorit:

- lämpötilasensori (HTS221)
- kosteussensori (HTS221)
- magnometri (LIS2MDL)
- painesensori (LPS22HB)
- kiihtyvyyssensori (LSM4DSL)
- kulmasensori (LSM6DSL).

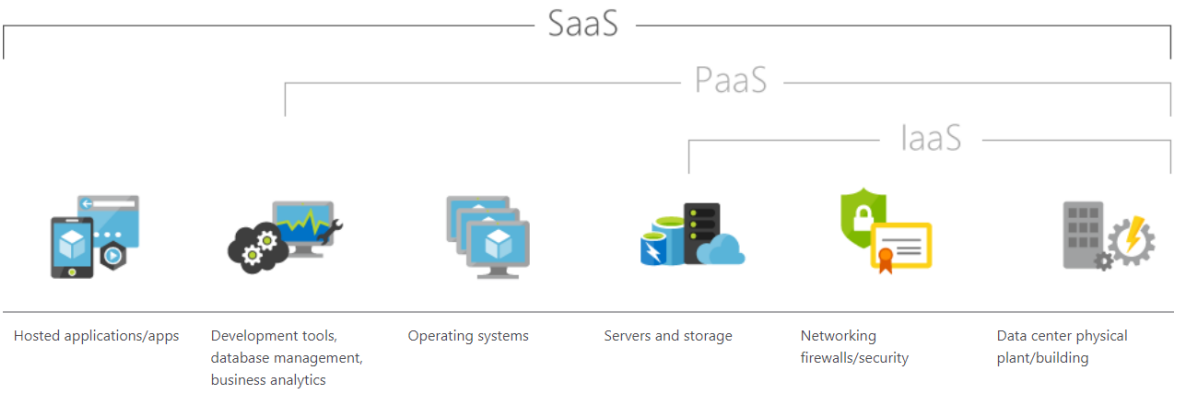
Ulkoiset komponentit yhdistetään käyttämällä alla olevassa kuvassa (Kuva 6) näkyviä liitoskohtia. Pin 1 -liitäntää käytetään datan lähettämiseksi ulkoisiin komponentteihin ja pin 2 -liitäntää datan vastaanottamiseksi. (Microsoft 2019c.)



Kuva 6. Ulkoisten komponenttien liitoskohdat (Microsoft 2019c)

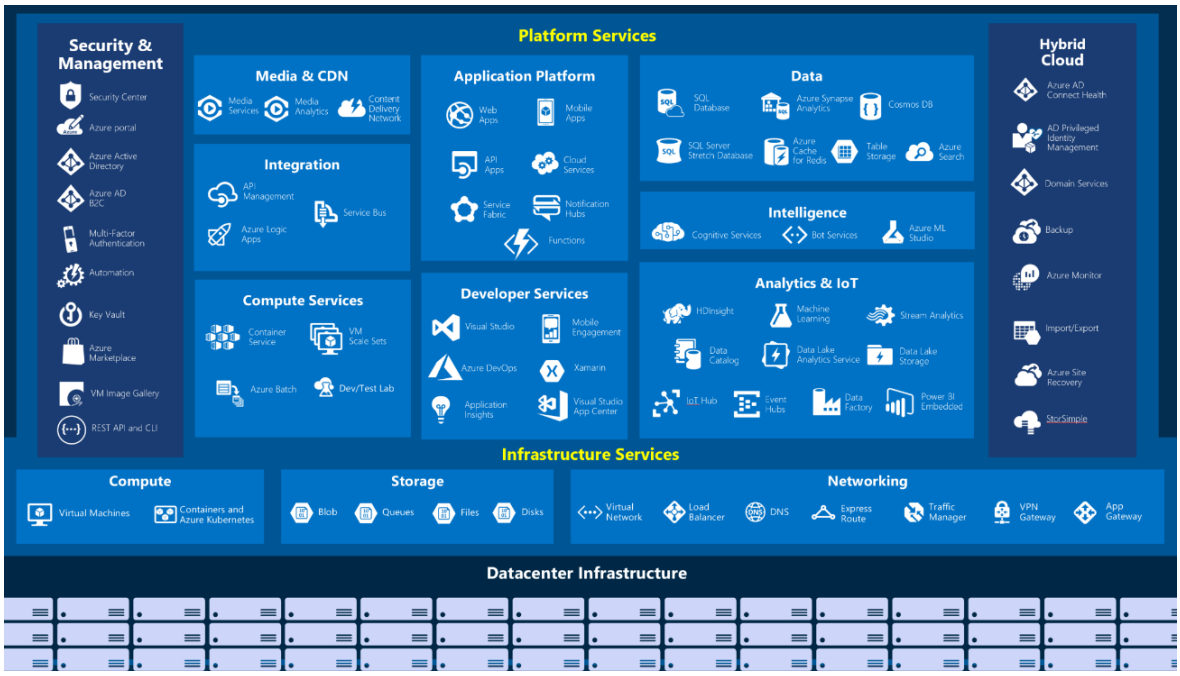
2.3 AZURE

Azure on Microsoftin tarjoama kokoelma pilvipalveluja, joka tarjoaa IaaS (Infrastructure as a Service), PaaS (Platform as a Service) ja SaaS (Software as a Service) -pohjaisia ratkaisuja, työkaluja ja komponentteja. IaaS -palveluihin kuuluu ainoastaan fyysinen infrastruktuuri, kuten palvelimet ja tallennustila sekä ylläpito. PaaS -palveluihin sisältyy IaaS -palveluiden lisäksi sovellusalusta, kuten käyttöjärjestelmä ja kehitystyökalut. SaaS -palveluihin kuuluvat edellä mainittujen IaaS -palveluiden sekä PaaS -palveluiden lisäksi ohjelmistot (Kuva 7). (Microsoft 2020o.)



Kuva 7. IaaS, PaaS, Saas (Microsoft 2020o)

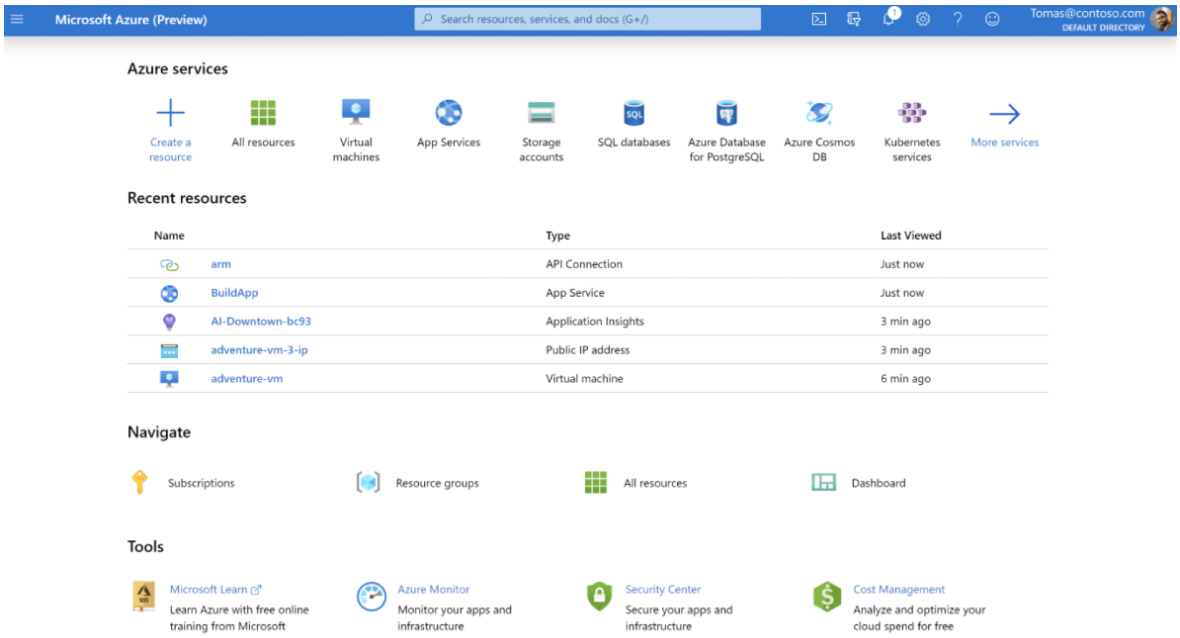
Azuressa on kymmeniä palveluja, jotka mahdollistavat projektien kehityksen, testauksen, julkaisemisen ja ylläpidon alusta loppuun (Kuva 8). Se tarjoaa myös esimerkiksi tekoäly-palveluja ja analytiikkatyökaluja. Osasta työkaluista on olemassa ilmaistason ominaisuuksia tietyillä rajoituksilla ja resursseja pystytään skaalaamaan sekä ominaisuuksia vaihtamaan tarvittaessa. (Microsoft 2020a.)



Kuva 8. Azuressa olevat palvelut (Microsoft 2020f)

Resurssien hallinnointi tapahtuu käyttämällä joko Visual Studiota tai Azuren tarjoamaa, verkossa toimivaa, visuaalista käyttöliittymää Azure Portalia (Kuva 9). Azure Portal

mahdollistaa komponenttien luomisen, muokkaamisen, käyttöoikeuksien hallinnan ja yleisen seurannan. Komponenttien kehittäminen onnistuu myös käyttämällä Visual Studiota, joka on virallinen Microsoftin tarjoama kehitysympäristö Azuren pilvipalveluille. Azuren kehitystyökalut ovat valmiiksi sisäänrakennettuna Visual Studioon mahdollistaen komponenttien luomisen, kehityksen ja julkaisemisen suoraan Azuren pilveen. (Microsoft 2020h.)



Kuva 9. Esimerkkinäkymä Azure Portalista (Microsoft 2020d)

2.3.1 Azure Funktiot

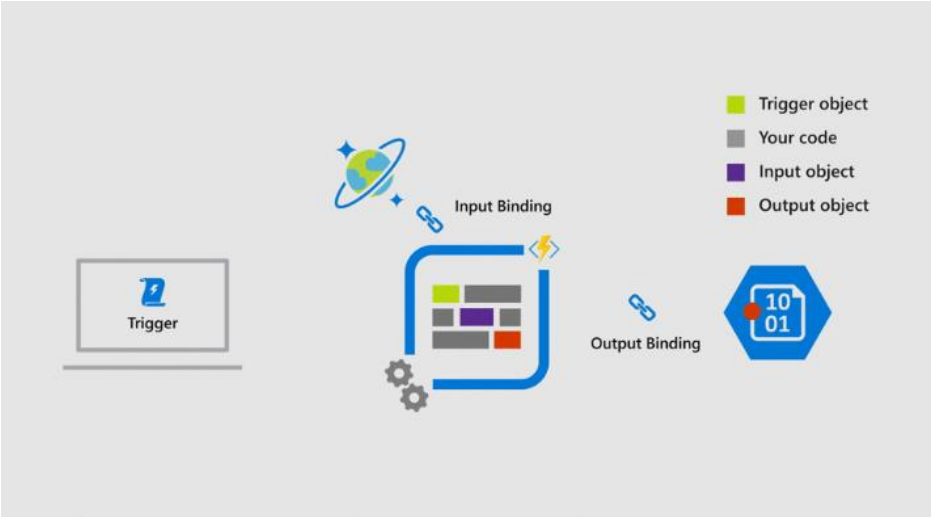
Azure Funktiot mahdollistavat tapahtumapohjaisen koodin suorittamisen ilman erillistä palvelinta. Funktiot toimivat ennalta määrätyillä laukaisimilla (trigger), jotka käynnistävät funktion, kun tietty tapahtuma on ilmennyt. Tapahtumia voivat olla esimerkiksi HTTP-pyyntö, IoT Hubiin saapuva data tai ajastimen laukeaminen. (Microsoft 2020g.) Funktioita voi kehittää paikallisesti esimerkiksi Visual Studiolla ja tämän jälkeen julkaista ne Azuren pilveen.

Azure tukee kahdenlaisia funktioita; tilattomia (stateless) ja tilallisia (stateful/durable functions). Tilattomien funktioiden tavoitteena on suorittaa lyhyttä, yhteen tarkoitukseen pyrkivää koodia (Microsoft 2020g). Kuvassa 10 on näkymä QueueTriggerillä käynnistyvästä funktiosta, joka tulostaa Queue Storageen saapuneen viestin.

```
namespace FunctionApp1
{
    public static class Function1
    {
        [FunctionName("QueueTriggerCSharp")]
        public static void Run([QueueTrigger("myqueue-items",
            Connection = "QueueStorage")]string myQueueItem, ILogger log)
        {
            log.LogInformation($"C# Queue trigger function processed: {myQueueItem}");
        }
    }
}
```

Kuva 10. Esimerkki Azure Funktiosta

Alla olevassa kuvassa (Kuva 11) on esimerkki tilattomasta Azure Funktion toiminnasta, jossa keskellä oleva funktio toimii eri palvelujen ja komponenttien välikätenä. Näin esimerkiksi IoT Hubiin saapuva data voidaan käsitellä ja tallentaa tietokantaan funktion avulla.



Kuva 11. Tilattoman Azure Funktion toimintamalli (Microsoft 2020v)

”Durable Functions” ovat jatkoa edellä mainituille tilattomille funktioille. Niiden tavoitteena on ylläpitää tiloja ja välivaiheita. Näin esimerkiksi voi ketjuttaa useita yksittäisiä funktioita, suorittaa monia välivaiheita ja prosesseja, jotka ovat toiminnassa niin kauan, kunnes jokainen on suoritettu. (Microsoft 2020i.)

Seuraavassa kuvassa (Kuva 12) on esimerkki funktioiden ketjuttamisesta. Kutsuva funktio on toiminnassa niin kauan, kunnes jokainen ”E1_SayHello” -funktio on suoritettu.

```
[FunctionName("E1_HelloSequence")]
public static async Task<List<string>> Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
{
    var outputs = new List<string>();

    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Tokyo"));
    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello", "Seattle"));
    outputs.Add(await context.CallActivityAsync<string>("E1_SayHello_DirectInput", "London"));

    // returns ["Hello Tokyo!", "Hello Seattle!", "Hello London!"]
    return outputs;
}
```

Kuva 12. Funktioiden ketjutus (Microsoft 2020p)

Alla olevassa kuvassa (Kuva 13) on ”E1_SayHello” -funktion koodi, jossa edellisessä kuvassa (Kuva 12) lähetetty kaupungin nimi palautetaan lauseen muodossa takaisin kutsuvalle funktiolle.

```
[FunctionName("E1_SayHello")]
public static string SayHello([ActivityTrigger] IDurableActivityContext context)
{
    string name = context.GetInput<string>();
    return $"Hello {name}!";
}
```

Kuva 13. E1_SayHello -funktio (Microsoft 2020p)

2.3.2 SQL tietokanta

Azure SQL relaatiotietokanta pohjautuu SQL Server Database Engine arkkitehtuuriin, joka on muokattu erityisesti pilviympäristöön sopivaksi. Se käyttää SQL-kyselykieltä (Structured Query Language) ja eroaa perinteisestä vaihtoehdosta eniten ylläpidollisesti. Tietokanta päivittyy automaattisesti ja pilvipalvelun tarjoaja huolehtii ylläpidosta, suorituskyvystä, virheenhallinnasta, turvallisuudesta sekä varmuuskopioinnista. Vaihtoehtoina tietokannan käytössä on yksittäinen palvelimeton tietokanta, jossa on useita eri tasoja tarpeen mukaan tai tietokantajoukko, joka käyttää yhteisiä resursseja. (Microsoft 2020e.)

Tietokantakomentoja voi suorittaa joko käyttämällä Query Editor -työkalua Azure Portalissa tai Azure Funktion avulla. Seuraavassa kuvassa (Kuva 14) funktio aktivoituu ajastimella 15 sekunnin välein ja päivittää taulussa olevan Status -sarakkeen.

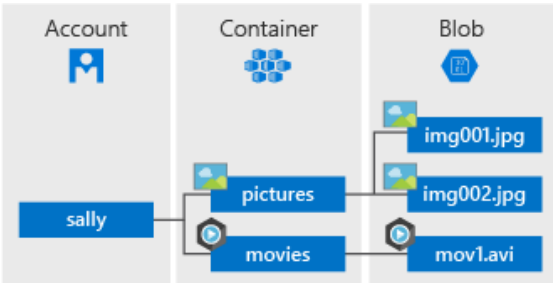
```
[FunctionName("DatabaseCleanup")]
public static async Task Run([TimerTrigger("*/15 * * * * *")]TimerInfo myTimer, ILogger log)
{
    // Get the connection string from app settings and use it to create a connection.
    var str = Environment.GetEnvironmentVariable("sqldb_connection");
    using (SqlConnection conn = new SqlConnection(str))
    {
        conn.Open();
        var text = "UPDATE SalesLT.SalesOrderHeader " +
                "SET [Status] = 5  WHERE ShipDate < GetDate();";

        using (SqlCommand cmd = new SqlCommand(text, conn))
        {
            // Execute the command and log the # rows affected.
            var rows = await cmd.ExecuteNonQueryAsync();
            log.LogInformation($"{rows} rows were updated");
        }
    }
}
```

Kuva 14. Taulun päivittävä Azure Funktio (Microsoft 2019b)

2.3.3 Storage

Blob Storage

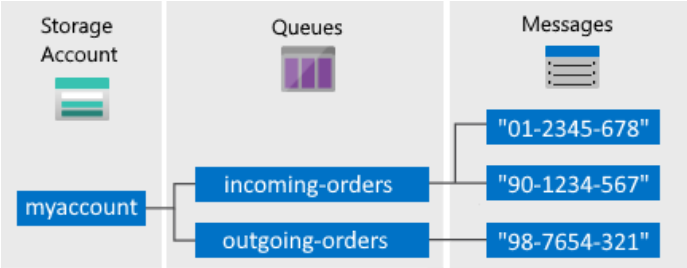
Blob (Binary Large Object) Storage on objektien säilytykseen tarkoitettu palvelu, joka on optimoitu suurien datamäärien talletukseen. Sen avulla on mahdollista esimerkiksi suoratoistaa mediaa, säilyttää tiedostoja tai tallettaa dataa varmuuskopiointitarkoituksessa. Data on mahdollista organisoida käyttämällä kontteja (container). (Microsoft 2020m.) Alla olevassa kuvassa (Kuva 15) esitellään Blob Storagessa olevia resurssityyppejä; tili (account), kontti (container) ja dataobjekti (blob). Tilillä voi olla rajattomasti kontteja ja kontit voivat sisältää rajattomasti dataobjekteja. (Microsoft 2020m.)



Kuva 15. Resurssien välinen yhteys Blob Storagessa (Microsoft 2020n)

Queue Storage

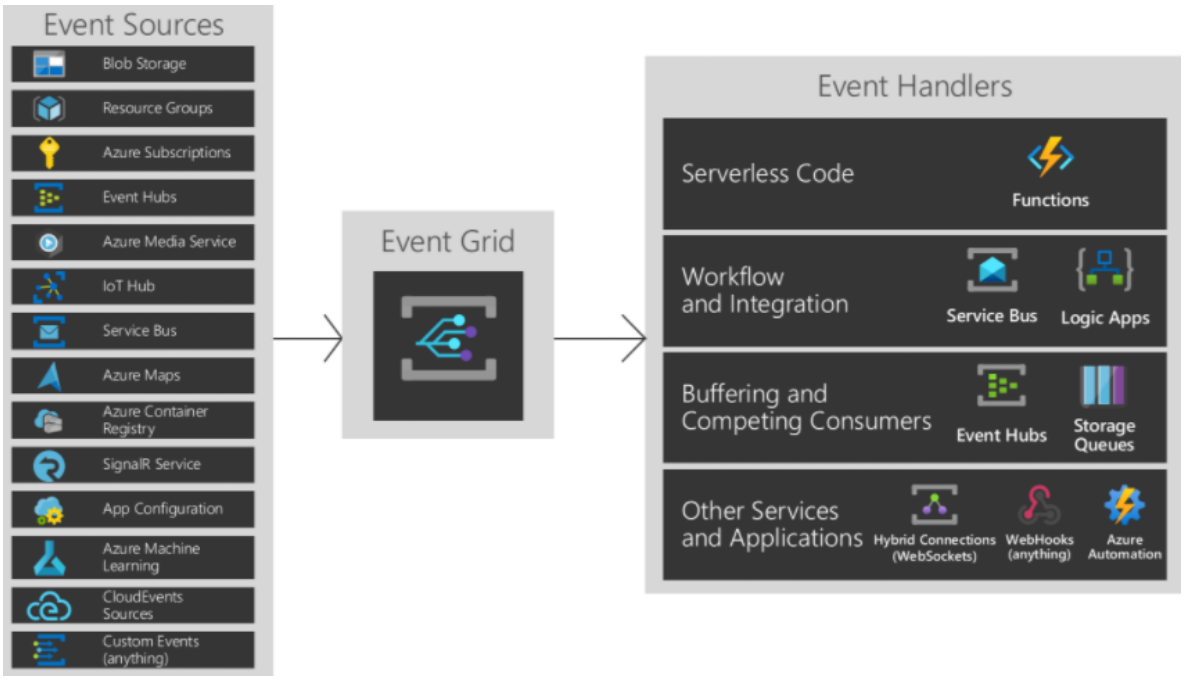
Queue Storage on isojen viestimäärien talletukseen tarkoitettu palvelu. Yhden viestin maksimikoko voi olla 64 kilotavua. Viestejä voi olla miljoonia, maksimimäärä riippuu tallennustilan koosta. Viestit säilyvät Queue Storagessa käyttäjän määräämän ajan, oletusarvoisesti seitsemän päivää. Yhteyden Queue Storageen saa joko käyttämällä autentikoituja http tai https -pyyntöjä tai esimerkiksi Azure Funktioiden avulla käyttämällä yhteysavainta (connection string). Seuraavassa kuvassa (Kuva 16) on esitetty resurssien välinen yhteys. (Microsoft 2020u.)



Kuva 16. Queue -palvelun sisältämät komponentit (Microsoft 2020w)

2.3.4 Event Grid

Event Grid toimii tapahtumien (events) reitittäjänä, jonka avulla voi välittää tapahtumia komponenttien ja palveluiden välillä (Microsoft 2020q). Kuvassa 17 on esitetty mahdolliset tapahtumalähteet (Event Sources) ja tapahtuman käsittelijät (Event Handlers).



Kuva 17. Event Gridin tukemat tapahtumalähteet ja käsittelijät (Microsoft 2020q)

Tapahtumien reititys luodaan käyttämällä Event Grid System Topicia, joka toimii tapahtumalähteistä tulevien viestien loppupisteenä. Jokaiseen Event Grid System Topiciin on mahdollista luoda tilauksia (subscriptions), joiden avulla määrätään, mitkä tapahtumat vastaanotetaan ja mihin kohteisiin ne siirretään. Tapahtumia voi erotella tyypin tai aiheen perusteella. (Microsoft 2020r.)

2.3.5 IoT Hub

IoT Hub toimii laitteiden ja pilvessä olevien työkalujen välisenä pisteenä. Tämä voi esimerkiksi olla sovelluksen automatisointiin, tekoälytoimintoihin tai analytiikkaan liittyvien työkalujen käyttämistä IoT Hubin kanssa. Saapuva data voidaan siirtää muihin kohteisiin jatkokäsittelyä varten käyttämällä muita komponentteja kuten esimerkiksi Event Gridiä. IoT Hub toimii näin eräänlaisena monisuuntaisena tiedonsiirtoväylänä IoT -laitteiden, palvelujen ja sovellusten välillä. Se tukee datan siirtoa sekä laitteista pilveen että pilvestä laitteisiin. (Microsoft 2019a.)

IoT Hub ja Azure Iot device SDK kirjastot tukevat seuraavia tiedonsiirtoprotokollia:

- HTTPS
- AMQP
- AMQP WebSocketien kautta
- MQTT
- MQTT WebSocketien kautta.

Mikäli edellä mainittuja ei ole mahdollista käyttää, vaihtoehtoina on myös mukautettuja Azuren työkaluja käyttäviä ratkaisuja. Kehityskielinä voidaan käyttää erilaisia ohjelmointikieliä. Tuettuja kieliä ovat mm. C, C#, Java, Python ja Node.js. (Microsoft 2019a.)

Tukeakseen kaikkia edellä mainittuja tiedonsiirtoprotokollia, IoT Hub käyttää yhteistä viestiformaattia. Tätä formaattia käytetään sekä laitteesta-pilveen, että pilvestä-laitteeseen tapahtuvissa tiedonsiirroissa. Viesti koostuu ennalta määrätyistä ominaisuuksista ja ehdoista (Kuva 18). (Microsoft 2020b.)

Kuva 18. IoT Hubin käyttämä JSON viestiformaatti laitteesta pilveen kulkevissa viesteissä (Microsoft 2020b)

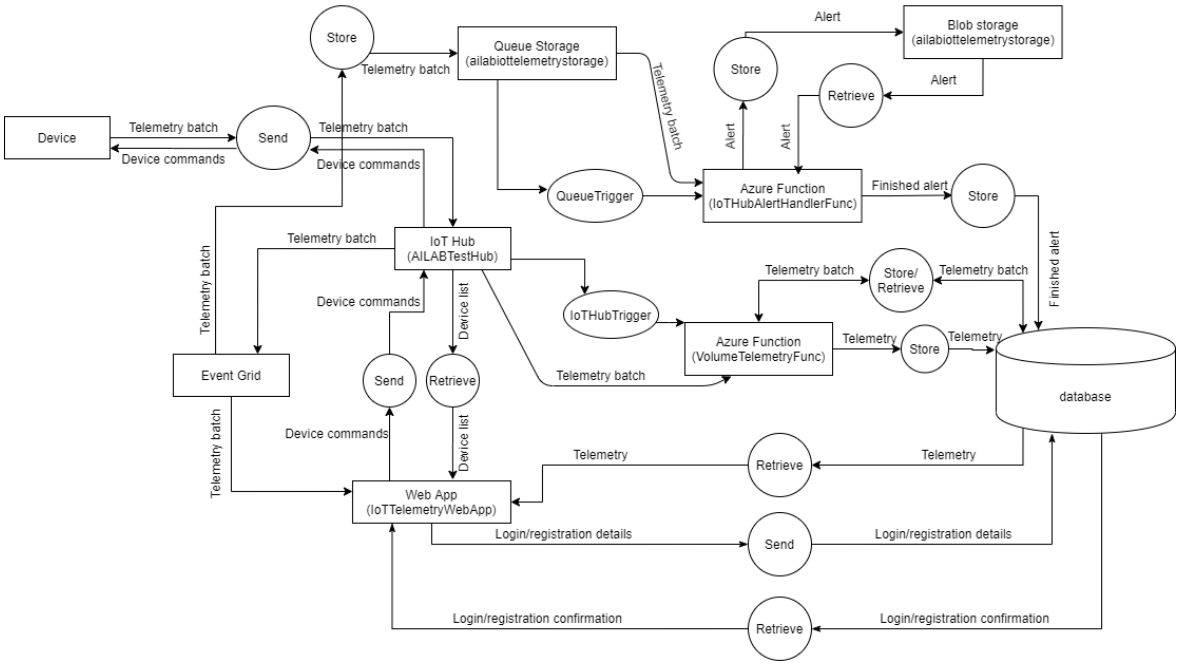
Viestissä voi olla ainoastaan numeroita ja kirjaimia. Lisäksi lähetettäessä viestejä käyttämällä HTTPS-protokollaa tai pilvestä-laitteeseen lähtevissä viesteissä on mahdollista käyttää tiettyjä erikoismerkkejä. Viestin maksimikoko laitteesta IoT Hubiin on 256 kilotavua ja 56 kilotavua IoT Hubista laitteisiin. Maksimimäärä jonoon laitettavia viestejä on 50. (Microsoft 2020c.)

IoT Hubin tarkoituksena on tukea alueittain miljoonia yhteyksiä ja jotta DoS (Denial of Service) hyökkäykset kyetään erottamaan tavallisesta toiminnasta, tiettyjä rajoituksia on asetettu. Rajoitukset vaihtelevat hintatason mukaan. Esimerkiksi ilmaistasolla rajoitus tulee vastaan, kun kyseessä on 12 viestiä/sekunti/laite ja 100 viestiä/sekunti kaikkien laitteiden kesken. IoT Hubiin on mahdollista liittää maksimissaan 1 000 000 laitetta ja IoT Hubeja on mahdollista luoda maksimissaan 50 kpl per tili. (Microsoft 2020j.)

3 TOTEUTUS

3.1 YLEISTÄ

Tässä luvussa kerrotaan yksityiskohtaisesti, miten opinnäytetyön projekti käytännössä toteutettiin. Projektikokonaisuus koostuu datan mittaamisesta, lähettämisestä pilveen sekä useisiin eri kohteisiin sijoittamisesta tallennusta ja visualisointia varten. Laitteen etähallinta toteutettiin verkkosivuston ja IoT Hubin avulla. (Kuva 19.) Ensin esitellään projektissa käytetty IoT DevKit -laite ja käydään kooditasolla läpi siinä tapahtuva datan keruu, käsittely ja lähetys pilveen sekä etäkomentojen käsittely. Tämän jälkeen käydään läpi käytössä olleet Azuren tarjoamat palvelut; IoT Hub, Event Grid, Azure Funktiot, Storage sekä Azure SQL tietokanta.



Kuva 19. Datan kulku projektissa

3.2 LAITE

Projektissa käytettiin datan mittaamiseen Arduino-yhteensopivaa IoT DevKit (AZ3166) -laitetta. Laitteeseen sisäänrakennettuja sensoreita käytettiin mittaamaan lämpötilaa, kosteutta, painetta, magneettikentän voimakkuutta, kiihtyvyyttä ja suuntaa. Lisäksi mikrofonia käytettiin tallentamaan ääntä, josta laskettiin desibelit. Laite ei ole virallinen Arduino, mutta on Arduino-yhteensopiva. Tämän vuoksi kehitysympäristönä on voitu käyttää virallista

Arduino IDEä, mutta vain osa kirjastoista olivat yhteensopivia DevKitin kanssa. Sensorien hallinnoimiseksi käytettiin laitteen virallisia kirjastoja.

3.2.1 Toiminta yleisesti

Jokaisella käynnistyskerralla laite suorittaa setup() -osiossa määrätyt toimet; ensin luo yhteyden ennalta määrättyyn WiFi -verkkoon ja initialisoi MQTT -protokollaa hallinnoivan kirjaston avulla yhteyden IoT Hubiin sekä viestien käsittelyn. Tämän jälkeen laite suorittaa loop() -osiota, joka jokaisen kierroksen alussa tarkistaa Arduino IDEn Serial Monitorin kautta tulevat syötteet laitteen ollessa kiinni tietokoneessa. Verkkosivulta IoT Hubin kautta tulevat etäkomennot vastaanotetaan käyttämällä paluuviestejä lähettäessä dataa pilveen. Mikäli käyttäjä on pysäyttänyt datan lähetyksen pilveen kokonaan, laite hakee viestit pilvestä käyttämättä paluuviestejä.

3.2.2 Datan mittaus

Äänen tallennukseen käytettiin IoT DevKitin virallista äänikirjastoa ”AudioClassV2”:sta (Microsoft 2017b). Alla olevassa kuvassa (Kuva 20) näkyvät kirjaston oletusarvot. Tallennuksessa käytetään 16 bitin resoluutiota, 8000Hz näytteenottotaajuutta ja 512 tavun puskuria.

```
#define DURATION_IN_SECONDS         2
#define DEFAULT_SAMPLE_RATE         8000
#define DEFAULT_BITS_PER_SAMPLE     16
#define MONO                        1
#define STEREO                      2
#define WAVE_HEADER_SIZE            44
#define AUDIO_CHUNK_SIZE            512
```

Kuva 20. Tallennuksessa käytettävät oletusarvot

Laitteen koodi suoritetaan loop() -funktiossa. Jokaisen kierroksen alussa laite tyhjentää puskurin äänen tallentamista varten. Tämän jälkeen asetetaan dataformaatti ja aloitetaan tallennus. Tallennuksen pituus on oletusarvoisesti 20 sekuntia ja se on mahdollista muuttaa joko Serial Monitorista tai verkkosivuston kautta.

Lopuksi tallennettua äänitettä käytetään desibelien laskemiseksi. Desibeliarvoista pienin, suurin sekä keskiarvo lasketaan käymällä äänitteen puskuri läpi jokainen arvo kerrallaan. (Kuva 21.)

```
560   // calculate values of buffer going through all chunks
561   // and calculate each chunk
562   for (int x = 0; x < chunksInAudio; x++)
563   {
564     // per frame in one chunk
565     for (int i = 0; i < length; i += 2) //two samples per frame (because stereo)
566     {
567       // compare stereo sample values (adjacent values for left channel and right channel)
568       if (readBuffer[i] > readBuffer[i+1])
569       {
570         amplitude = amplitude + readBuffer[i] - readBuffer[i+1]; // assign left
571       }
572       else
573       {
574         amplitude = amplitude + readBuffer[i + 1] - readBuffer[i]; // assign right
575       }
576       amplitude = amplitude / length * 2; // calculate amplitude by deviding peak value by max value
577       // calculate desibels of each frame (dBFS), max 0
578       if (amplitude != 0) // otherwise if amplitude is zero, prints out "inf"
579       {
580         dB = 20 * log10 (amplitude);
581       }
582       // compare values to get min and max of each frame
583       if(dB < min)
584       {
585         min = dB;
586       }
587       if (dB > max)
588       {
589         max = dB;
590       }
591
592       // add up all values for calculating average
593       noiseAverage = noiseAverage + dB;
594       avgCalc++;
595     }
596
597   }
598   // add data to parameter for sending to iot hub
599   noiseAvg = noiseAverage / avgCalc;
600   noiseMin = min;
601   noiseMax = max;
602
603   sendDataFirstBatch(); // send data after each value has been calculated
604   sendDataSecondBatch();
```

Kuva 21. Äänidatan arvojen laskeminen laitteessa

Edellisessä kuvassa (Kuva 21) rivillä 580 desibelit lasketaan seuraavan kaavan mukaisesti (Kaava 1).

$$dB = 20 * \log10(\text{arvo} / \text{maksimi amplitudi}) \quad (1)$$

Asteikkona laskuissa käytetään desibel full scale -asteikkoa, jossa mikrofonin havaitsema maksimiääni digitaalisessa nauhoitteessa on 0. Esimerkiksi signaali, joka on 10 desibeliä hiljaisempi kuin mahdollinen maksimitaso, on tällöin -10dBFS. Desibelien todellisia arvoja ei ole kalibroitu, kerätyt arvot ovat siten verrannollisia ainoastaan keskenään. Muiden sensoreiden datat mitataan käyttämällä virallisia kirjastoja. Jokaisen sensorin data otetaan siltä hetkeltä, kun äänidata on kerätty ja desibelit laskettu.

Datan mittaamisessa käytettiin seuraavia kirjastoja:

- HTS221Sensor
- LPS22HBSensor
- LIS2MDLSensor
- LSM6DSLSensor.

Käytössä olevat sensorit initialisoidaan setup() -osiossa laitteen käynnistyttyä. Alla olevassa kuvassa (Kuva 22) ensin luodaan I2C tiedonsiirtoväylän hallinnan käyttöliittymä ja tämän jälkeen initialisoidaan sensorit.

```
void SensorInit()
{
    i2c = new DevI2C(D14, D15);
    tempHumSensor = new HTS221Sensor(*i2c);
    tempHumSensor -> init(NULL);

    pressureSensor = new LPS22HBSensor( * i2c);
    pressureSensor -> init(NULL);

    magneticSensor = new LIS2MDLSensor(*i2c);
    magneticSensor -> init(NULL);

    accelGyroSensor = new LSM6DSLSensor(*i2c, D4, D5);
    accelGyroSensor->init(NULL);

    humidity = -1;
    temperature = -1000;
    pressure = 0;
    magnetism = 0;

}
```

Kuva 22. Sensorien initialisointi

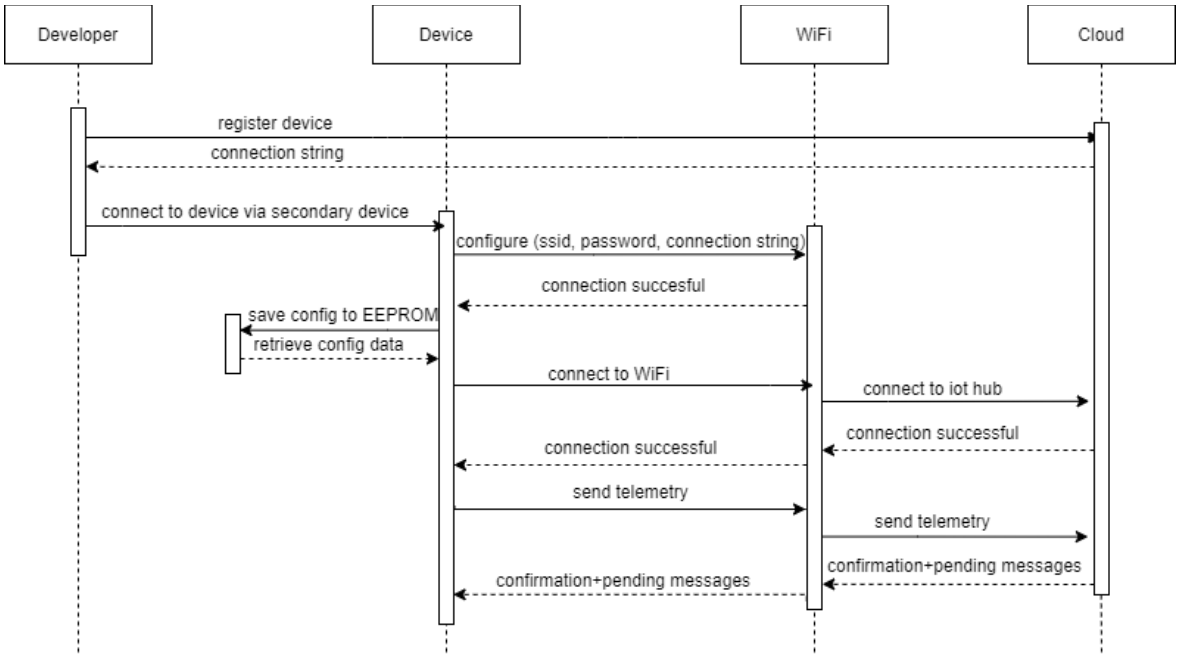
Seuraavassa kuvassa (Kuva 23) näytetään itse sensoreiden datan käsittely, jossa ylimmät funktiot palauttavat mitatun arvon suoraan ja kolme alinta funktiota lisäävät arvot omiin taulukoihinsa.

```
float readTemperature()
{
    tempHumSensor->reset();

    float temperature = 0;
    tempHumSensor->getTemperature(&temperature);

    return temperature;
}

float readHumidity()
{
    tempHumSensor->reset();

    float humidity = 0;
    tempHumSensor->getHumidity(&humidity);

    return humidity;
}

float readPressure() {
    float pressure = 0;
    pressureSensor -> getPressure(& pressure);

    return pressure;
}

void readMagneticData() {
    magneticSensor -> getMAxes(magnAxes);
}

void readAcceleration(){
    accelGyroSensor->enableAccelerator();
    accelGyroSensor->getXAxes(accelAxes);
}

void readGyro() {
    accelGyroSensor->enableGyroscope();
    accelGyroSensor->getGAxes(gyroAxes);
}
```

Kuva 23. Sensoridataa palauttavat funktiot

3.2.3 Datan lähetys ja vastaanotto

Projektissa yhteyden luominen IoT Hubiin ja datan lähetys sekä vastaanotto toteutettiin DevKitMQTTClient -kirjaston ja IoT Hubista saatavan laitteen yhteysavaimen avulla. Alla olevassa kuvassa (Kuva 24) esitetään yhteyden luominen pilveen ensimmäisen kerran.

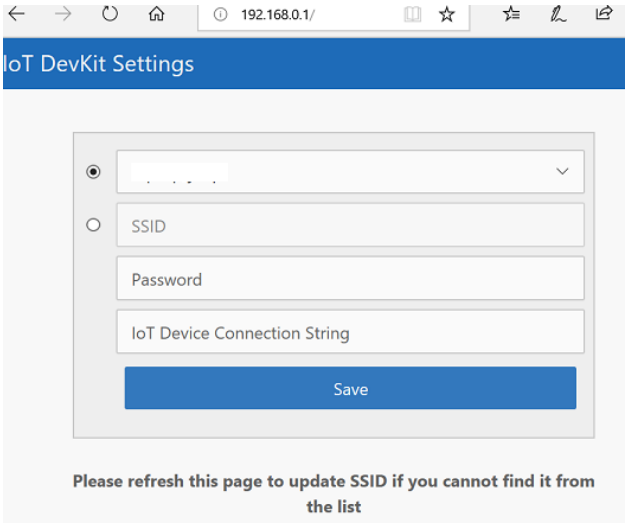


Kuva 24. Yhteyden luominen pilveen ja datan siirto

Ennen yhteyden luomista, laite ensin rekisteröitiin IoT Hubiin, jolloin laitteelle luotiin automaattisesti yhteysavain. Avaimen saa haettua käyttämällä Azure Portalia ja se näyttää seuraavanlaiselta:

HostName=iothubnimi.azure-devices.net;DeviceId=laite;SharedAccessKey=avain (2)

Yhdistäminen WiFi -verkkoon tapahtui toisen tietokoneen avulla käyttämällä IoT DevKit -laitetta tukiasemana. Tarvittavat asetukset määriteltiin verkkoselaimessa menemällä alla olevassa kuvassa (Kuva 25) näkyvään osoitteeseen ja lisäämällä verkko, salasana sekä aiemmin haettu yhteysavain.



Kuva 25. Laitteen WiFi-yhteyden ja yhteysavaimen asettaminen (Microsoft 2020k)

Tämän jälkeen yhteysavain on tallennettu laitteen omaan muistiin (EEPROM -Electrically Erasable Programmable Read Only Memory), jonka tarkoituksena on säilyttää asetukset käynnistysten välillä. Yhteysavaimen käsittely tapahtuu DevKitMQTTClient -kirjaston avulla alla olevan kuvan (Kuva 26) mukaisesti.

```
502     // Load connection from EEPROM
503     EEPROMInterface eeprom;
504     uint8_t connString[AZ_IOT_HUB_MAX_LEN + 1] = {'\0'};
505     int ret = eeprom.read(connString, AZ_IOT_HUB_MAX_LEN, 0x00, AZ_IOT_HUB_ZONE_IDX);
506     if (ret < 0)
507     {
508         LogError("Unable to get the azure iot connection string from EEPROM. Please set the value in configuration mode.");
509         return false;
510     }
511     else if (ret == 0)
512     {
513         LogError("The connection string is empty.\r\nPlease set the value in configuration mode.");
514         return false;
515     }
516
517     iothub_hostname = GetHostNameFromConnectionString((char *)connString);
518
519     // Create the IoTHub client
520     if ((iotHubClientHandle = IoTHubClient_LL_CreateFromConnectionString((char *)connString, MQTT_Protocol)) == NULL)
521     {
522         LogTrace("Create", "IoT hub establish failed");
523         return false;
524     }
```

Kuva 26. Yhteysavaimen käsittely laitteen muistista ja yhteyden luominen Iot Hubiin

Laitteen käynnistyessä, setup() -osiossa, yhteysasetukset ja viestiasetukset initialisoidaan käyttämällä DevKitMQTTClient -kirjastoa (Kuva 27). Rivillä 229 DevKitMQTTClient_Init() funktio sisältää mm. edellisessä kuvassa (Kuva 26) olevan yhteysavaimen haun EEPROM

-muistista ja IoT Hub asiakasohjelman (client) luomisen sen avulla. Tämän jälkeen initialisoidaan lähetettävien ja saapuvien viestien käsittely.

```
229   DevKitMQTTClient_Init(true);
230
231   //Sets up send confirmation status callback to be invoked representing the status of sending message to IOT Hub
232   DevKitMQTTClient_SetSendConfirmationCallback(SendConfirmationCallback);
233   //Sets up the message callback to be invoked when IoT Hub issues a message to the device
234   DevKitMQTTClient_SetMessageCallback(MessageCallback);
235   //Sets up the device twin callback to be invoked when IoT Hub update device twin of the device
236   DevKitMQTTClient_SetDeviceTwinCallback(DeviceTwinCallback);
237   //Sets up the device method callback to be invoked when IoT Hub call method on the device
238   DevKitMQTTClient_SetDeviceMethodCallback(DeviceMethodCallback);
239
```

Kuva 27. Yhteyden ja viestiasetusten initialisointi

Koska IoT Hub pystyy vastaanottamaan maksimissaan 256 kilotavun kokoisia viestejä, data päätettiin lähettää laitteesta kahdessa osassa. IoT Hubin loppupisteenä viesteille on oletusarvoinen {messages/events}. Alla olevassa kuvassa (Kuva 28) on esitetty ensimmäisen osan serialisointi laitteessa IoT Hubiin lähetystä varten.

```
267   // data to be sent to hub
268   // add data to the output JSON file
269   json_object_set_number(root_object, "Id", messageId);
270   json_object_set_number(root_object, "batch", messageBatchCount);
271
272   if (tempMeasurement == true)
273   {
274       json_object_set_number(root_object, "temp", temperature);
275   }
276
277   if (humMeasurement == true)
278   {
279       json_object_set_number(root_object, "hum", humidity);
280   }
281
282   if (presMeasurement == true)
283   {
284       json_object_set_number(root_object, "pres", pressure);
285   }
286
287   if (audioMeasurement == true)
288   {
289       json_object_set_number(root_object, "avg", nAvg);
290       json_object_set_number(root_object, "min", nMin);
291       json_object_set_number(root_object, "max", nMax);
292   }
293
294   serialized_string = json_serialize_to_string_pretty(root_value);
295
296   snprintf(payload, MESSAGE_MAX_LEN, "%s", serialized_string);
297   json_free_serialized_string(serialized_string);
298   json_value_free(root_value);
```

Kuva 28. Datan ensimmäisen osan käsittely

Hälytysten tilat tarkastetaan laitteessa ennen viestin lähetystä ja tarkastuksen jälkeen hälytykset lisätään alla olevan kuvan (Kuva 29) mukaisesti property -osioon. Lopuksi property -osio yhdistetään edellisessä kuvassa (Kuva 28) muodostetun viestin perään.

```
689     // generate event with event string, messagePayload is a json string,
690     // which is generated by readMessage -function and contains all data except alerts
691     EVENT_INSTANCE* message = DevKitMQTTClient_Event_Generate(messagePayload, MESSAGE);
692
693     // adds applicationProperties to the message
694     if (tempMeasuring == true) {
695       DevKitMQTTClient_Event_AddProp(message, "tempAlert", temperatureAlert ? "true" : "false");
696     }
697     if (humMeasuring == true) {
698       DevKitMQTTClient_Event_AddProp(message, "humAlert", humidityAlert ? "true" : "false");
699     }
700     if(presMeasuring == true){
701       DevKitMQTTClient_Event_AddProp(message, "presAlert", pressureAlert ? "true" : "false");
702     }
703     if(audioMeasuring == true){
704       DevKitMQTTClient_Event_AddProp(message, "avgAlert", noiseAvgAlert ? "true" : "false");
705       DevKitMQTTClient_Event_AddProp(message, "minAlert", noiseMinAlert ? "true" : "false");
706       DevKitMQTTClient_Event_AddProp(message, "maxAlert", noiseMaxAlert ? "true" : "false");
707     }
708     //Synchronous call to report the event
709     DevKitMQTTClient_SendEventInstance(message);
```

Kuva 29. Viestin ensimmäisen osion hälytysten käsittely

Etähallinta

Laitteen etähallinta toteutettiin käyttämällä verkkosivustoa, josta komennot lähetetään IoT Hubin kautta laitteeseen. Jokaisen pilveen lähetetyn viestin jälkeen laitteelle palautetaan paluuviestinä IoT Hubissa jonossa olevat komennot. Kuvassa 30 on esitetty saapuvan datan vastaanotto laitteessa ja mahdollisten laitekomentojen tarkastaminen. Riippuen saapuneesta komennosta, laite joko käynnistyy uudelleen, lopettaa kaikkien tai yksittäisten sensorien datan lähetyksen, päivittää lähetyksen välin tai muuttaa hälytysten tasoja.

```
121  static void MessageCallback(const char* payLoad, int size)
122  {
123      char buff[128];
124      int x;
125
126      //Get data coming from IoT Hub
127      snprintf(buff, 128, "INCOMING:\r\n \r\n %s \r\n \r\n", payLoad);
128      Screen.print(buff);
129
130      LogInfo("Message received from the Azure IoT Portal: %s", payLoad);
131      Screen.clean();
132
133      if(atoi((char *)payLoad) != 0)
134      {
135          blinkInputReceived();
136      }
137
138      if(atoi((char *)payLoad) < 20)
139      {
140        checkInput(atoi((char *)payLoad));
141      }
142      if(atoi((char *)payLoad) > 1000)
143      {
144        // set interval
145        oldInput = atoi((char *)payLoad);
146      }
147
148      // check if alert treshold
149      if((payLoad[0] == 't') || (payLoad[0] == 'h') || (payLoad[0] == 'p')
150      || (payLoad[0] == 'v') || (payLoad[0] == 'm') || (payLoad[0] == 'a') || (payLoad[0] == 'g'))
151      {
152        handleAlertPayload(payLoad);
153      }
154
155  }
```

Kuva 30. Pilvestä tulevien viestien vastaanottoa käsittelevä funktio

Komennot 1–20 ovat sensoreiden datalähetyksen hallintaan, arvot yli 1000 lasketaan lähetysväliksi ja arvot, joissa alussa on yksi tai kaksi kirjainta ovat hälytystasojen muuttamista varten. Taulukossa 1 on lueteltu jokainen komento ja sitä vastaava toiminta.

Taulukko 1. IoT DevKitin etähallinnan käyttämät arvot ja niiden selitys

Arvo	Toiminta
1	Uudelleenkäynnistys
2	Kaiken datan lähetys on/off
3	Lämpötilan lähetys on/off
4	Kosteuden lähetys on/off
5	Äänidatan lähetys on/off
6	Paineen lähetys on/off
7	Magneettidatan lähetys on/off
8	Kiihtyvyysdatan lähetys on/off
9	Kallistusdatan lähetys on/off
t	Lämpötilan hälytystason muutos
h	Kosteuden hälytystason muutos
p	Paineen hälytystason muutos
vx	Äänen minimin hälytystason muutos
vy	Äänen maximin hälytystason muutos
vz	Äänen keskiarvon hälytystason muutos
mx	Magneettidatan akseli x:n hälytystason muutos
my	Magneettidatan akseli y:n hälytystason muutos
mz	Magneettidatan akseli z:n hälytystason muutos
ax	Kiihtyvyysdatan akseli x:n hälytystason muutos
ay	Kiiihtyvyysdatan akseli y:n hälytystason muutos
az	Kiihtyvyysdatan akseli z:n hälytystason muutos
gx	Kallistusdatan akseli x:n hälytystason muutos
gy	Kallistusdatan akseli y:n hälytystason muutos
gz	Kallistusdatan akseli z:n hälytystason muutos

Mikäli laite komennetaan lopettamaan datan lähettäminen IoT Hubiin, toimintamalli muuttuu niin, että jokaisella loop() -kierroksella laite vastaanottaa IoT Hubista tulevia viestejä suoraan käyttämättä paluuviestitoimintoa (Kuva 31).

```
void receiveData()
{
  if (hasWifi && hasIoTHub)
  {
    DevKitMQTTClient_Check();

  }
}
```

Kuva 31. Viestien vastaanottaminen

Vastaanottaessa käskyjä, lähetettäessä dataa tai silloin, kun datan lähetys on estetty, laite vilkuttaa eri värisiä ledejä merkiksi:

- violetti led: komento vastaanotettu
- punainen led: datan lähetys pysäytetty
- vihreä led: data lähetetty.

3.3 AZURE

Projektissa käytettiin useita Azuren palveluita laitteesta saapuvan datan käsittelyssä ja tallettamisessa pilveen sekä laitteen etähallinnan toteuttamisessa. IoT Hubin tarkoituksena on vastaanottaa laitteesta saapuvia viestejä ja välittää verkkosivustolta lähetettyjä komentoja takaisin laitteelle. Event Gridin avulla laitteesta IoT Hubiin saapuva data välitetään Queue Storageen hälytysten käsittelyä varten sekä verkkosivustolle datan visualisointia varten. Kahta Azure Funktiota käytettiin datan tallentamiseksi tietokantaan sekä hälytysten käsittelyssä.

3.3.1 IoT Hub

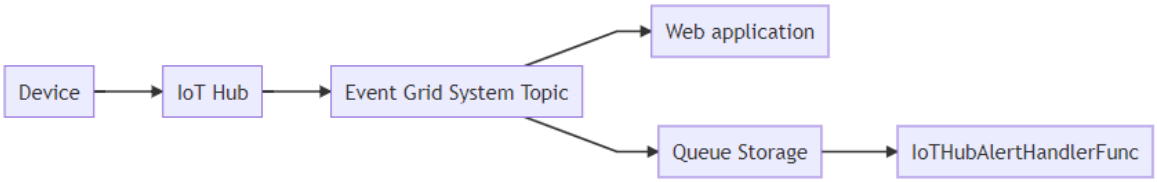
IoT Hubia käytettiin keskeisenä osana laitteesta tulevan datan vastaanottamisessa sekä laitteelle lähetettävien komentojen välittämisessä. Laitteesta IoT Hubiin saapuva data (Kuva 32) aktivoi Azure Funktion, jonka tehtävänä on yhdistää osittaiset viestit ja tallettaa niiden sisältämät arvot tietokantaan.


```
[IoTHubMonitor] [10:24:42 AM] Message received from [TestDevice1]:
{
    "body": {
        "Id": 24,
        "batch": 1,
        "temp": 29.89999961530273,
        "hum": 37.79999923706055,
        "pres": 1012.5,
        "avg": -33.561073303222656,
        "min": -226.810546875,
        "max": -19.187671661376953
    },
    "applicationProperties": {
        "tempAlert": "true",
        "humAlert": "false",
        "presAlert": "false",
        "avgAlert": "true",
        "minAlert": "false",
        "maxAlert": "true"
    }
}
```


Kuva 32. IoT Hubin vastaanottama datan ensimmäinen osa

3.3.2 Event Grid

Koska projektia varten haluttiin saada hälytysten käsittely ja datan liveseuranta käyttämättä jatkuvaa tietokannan pollausta (polling), käyttöön otettiin Event Grid. Event Gridin tarkoituksena on välittää liikennettä määrätyistä lähteistä määrättyihin kohteisiin käyttämällä tilaajia. Event Grid System Topic toimii IoT Hubista saapuvan datan loppupisteenä, josta data ohjataan tilauksien (subscriptions) avulla eri kohteisiin (Kuva 33).



Kuva 33. Datan siirtyminen Event Gridin kautta

Event Subscription

Tilausten avulla määritellään mitkä tapahtumat (tässä tapauksessa dataa sisältävät viestit) siirretään Event Grid System Topicista mihin loppupisteisiin. Kun viesti saapuu IoT Hubiin, alla olevassa kuvassa (Kuva 34) näkyvät tilaukset siirtävät sen joko verkkosivustolle suoraan datan visualisointia varten tai Queue Storageen hälytysten jatkokäsittelyä varten. Kuvassa näkyvä tilaus ”IOT-event-local-debugging” luotiin paikallista kehitystä varten käyttämällä ngrok -palvelun tarjoamaa tunnelointia, jonka avulla Azuren pilvi pystyy kommunikoimaan paikallisella koneella olevan Azure Funktion kanssa.

Name	Endpoint	Prefix Filter	Suffix Filter	Event Types
IOT-grid-webapp-sub	WebHook			Microsoft.Devices.DeviceTelemetry
IOT-event-local-debugging	WebHook			Microsoft.Devices.DeviceTelemetry
alert-queue-sub	StorageQueue			Microsoft.Devices.DeviceTelemetry

Kuva 34. Projektissa käytössä olevat tilaukset (subscriptions)

Event Grid vaatii todennuksen loppupisteen omistuksesta ennen tapahtumien (events) siirtämistä niihin. Joissakin tilanteissa, kuten Event Grid Triggerillä aktivoituvassa Azure Funktiossa tämä todennetaan automaattisesti, mutta esimerkiksi käyttäessä http -pyyntöjä kättely on hoidettava manuaalisesti. Tapahtuma lähetetään automaattisesti tilausta

luodessa JSON-muodossa loppupisteeseen. Tämä tapahtuma sisältää otsikkokentän, jossa on joka kerta vaihtuva koodi ”validationCode” (Kuva 35). Loppupisteen on palautettava kyseinen koodi takaisin kättelyn viimeistelemiseksi. (Microsoft 2020s.)

```
Header: "aeg-event-type:SubscriptionValidation"
Event body: "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",
Data property: "validationCode": "string"
```

Kuva 35. Lähetetty tapahtuma

Seuraavassa kuvassa (Kuva 36) näkyy verkkosivun palvelinpuolen koodissa saapuvan tapahtuman käsittely. Todennustapahtumassa oleva data deserialisoidaan rivillä 107, ”validationCode” haetaan rivillä 120 ja palautetaan kutsuvalle funktiolle (Kuva 37) JSON -muodossa.

```
105         private async Task<JsonResult> HandleValidation(string jsonContent)
106         {
107             var gridEvent =
108                 JsonConvert.DeserializeObject<List<GridEvent<Dictionary<string, string>>>>(jsonContent)
109                     .First();
110
111             await this._hubContext.Clients.All.SendAsync(
112                 "gridupdate",
113                 gridEvent.Id,
114                 gridEvent.EventType,
115                 gridEvent.Subject,
116                 gridEvent.EventTime.ToLongTimeString(),
117                 jsonContent.ToString());
118
119             // Retrieve the validation code and echo back.
120             var validationCode = gridEvent.Data["validationCode"];
121             return new JsonResult(new
122             {
123                 validationResponse = validationCode
124             });
125         }
```

Kuva 36. Verkkosivun koodi, jossa käsitellään validointitapahtuma

Kuvassa 37 vastaanotetaan saapuva tapahtuma ja tarkistetaan sen tyyppi. Mikäli kyseessä on validointipyyntö, rivillä 83 kutsutaan edellisessä kuvassa (Kuva 36) näkyvää HandleValitation(jsonContent) -funktiota. Funktio palauttaa vastaanotetun koodin, joka lähetetään takaisin pilveen tilauksen luomista varten.

```
71          [HttpPost]
            0 references
72          public async Task<IActionResult> Post()
73          {
74              using (var reader = new StreamReader(Request.Body, Encoding.UTF8))
75              {
76                  var jsonContent = await reader.ReadToEndAsync();
77
78                  // Check the event type.
79                  // Return the validation code if it's
80                  // a subscription validation request.
81                  if (EventTypeSubcriptionValidation)
82                  {
83                      return await HandleValidation(jsonContent);
84                  }
85                  else if (EventTypeNotification)
86                  {
87                      // Check to see if this is passed in using
88                      // the CloudEvents schema
89                      if (IsCloudEvent(jsonContent))
90                      {
91                          return await HandleCloudEvent(jsonContent);
92                      }
93
94                      return await HandleGridEvents(jsonContent);
95                  }
96
97                  return BadRequest();
98              }
99          }
```

Kuva 37. Koodin palautus tilauksen luomista varten

Tilaus luodaan vasta, kun validointivastaus on palautettu loppupisteestä. Vastaus näyttää seuraavan kuvan mukaiselta (Kuva 38).

```
"validationResponse": "string"
```

Kuva 38. Loppupisteen palauttama vastaus

3.3.3 Azure Funktiot

Projektissa käytettiin kahta Azure Funktiota. Ensimmäisen funktion tarkoituksena on yhdistää IoT Hubiin kahdessa osassa saapuvat viestit ja tallettaa kokonaisten viestien tiedot tietokantaan. Toinen funktio on tarkoitettu käsittelemään ainoastaan hälytyksiä.

Azure Funktio (VolumeTelemetryFunc)

Seuraavassa kuvassa (Kuva 39) oleva funktio aktivoituu joka kerta, kun IoT Hubiin saapuu dataa laitteesta. Ensin saapunut data (message) vastaanotetaan rivillä 21. Mikäli kyseessä on viestin ensimmäinen osa, tallennetaan se väliaikaistauluun (rivi 41). Kun toinen osa viestistä saapuu, se yhdistetään ensimmäisen osan kanssa ja lisätään lopullisena sijaintina toimivaan tauluun (rivi 55). Tämän jälkeen ensimmäinen osa poistetaan väliaikaisesta taulusta. Virhetilanteessa puskuritaulusta poistetaan kaikki data laitteen perusteella (rivi 60).

```
20          [FunctionName("HandleTelemetry")]
            0 references
21          public static void Run([IoTHubTrigger("messages/events", Connection = "IoTHuBConnectionString")]
22          EventData message, ILogger log)
23          {
24              DateTime hubTime = message.SystemProperties.EnqueuedTimeUtc;
25              string device = (message.SystemProperties["iothub-connection-device-id"]).ToString();
26              string jsonStr = Encoding.UTF8.GetString(message.Body.Array);
27              JObject jsonObj = JObject.Parse(jsonStr);
28              JToken batchToken = jsonObj.GetValue("batch");
29
30              Dictionary<string, JToken> dict = RetrieveValuesFromMessage(message);
31
32              log.LogInformation(jsonStr);
33
34              // retrieve message from buffer table and insert into telemetry table
35              try
36              {
37                  if(batchToken.ToString() == "1")
38                  {
39                      log.LogInformation($"FIRST BATCH: {batchToken}");
40                      //insert into buffer
41                      InsertFirstIntoBufferTable(hubTime, device, dict);
42                  }
43                  else
44                  {
45                      log.LogInformation($"SECOND BATCH: {batchToken}");
46
47                      // retrieve first batch from buffer
48                      ArrayList firstBatchFromBufferlist = RetrieveFromBufferTable(device);
49
50                      if(firstBatchFromBufferlist.Count > 0)// if buffer has first batch
51                      {
52                          // if buffer did not have more than one first batch for that device
53                          if (firstBatchFromBufferlist.Count == 19)
54                          {
55                              InsertIntoTelemetryTable(firstBatchFromBufferlist, dict);
56                          }
57                          else
58                          {
59                              // in case of error, where buffer table has more rows than there should be
60                              DeleteFromBufferAllByDevice(device);
61                          }
62                      }
63                      //if buffer didin't have first batch for the device, do nothing
64                  }
65              }
66              catch (Exception ex)
67              {
68                  log.LogInformation($"ERROR: {ex.Message}");
69                  return;
70              }
71          }
```

Kuva 39. Saapuvien viestien käsittely Azure Funktiossa

Azure Funktio (IoTHubAlertHandlerFunc)

Kuvassa 40 oleva funktio aktivoituu joka kerta, kun Queue Storageen saapuu dataa. Rivillä 52 saapuneesta datasta tallennetaan hälytystilassa olevat tiedot väliaikaisesti Blob Storageen. Tämän jälkeen uuden datan tietoja verrataan kategorian ja laitteen mukaan aiemmin talletettuihin tietoihin. Mikäli uudessa datassa hälytystila ei ole enää aktiivinen (status on tilassa ”false”) ja Blob Storagesta löytyy samaa kategoriaa oleva aktiivinen hälytys, vanha ja uusi data yhdistetään ja tallennetaan tietokantaan ”alerts” -tauluun (rivi 64). Samalla jo päättynyt hälytystieto poistetaan Blob Storagesta rivillä 69.

```
23      [FunctionName("AlertFunc")]
        0 references
24      public static void Run([QueueTrigger("alert-queue")] string message, ILogger log)
25      {
26          JObject jsonObj = JObject.Parse(message);
27          JToken timeToken = jsonObj.SelectToken("eventTime");
28          DateTime hubTime = (DateTime)timeToken;
29          string formattedHubTime = hubTime.ToString("dd.MM.yyyy hh:mm:ss");
30
31          JToken device = jsonObj.SelectToken("subject");
32          var formattedDevice = device.ToString().Substring(8);
33
34          try
35          {
36              var alerts = GetAlerts(message);
37
38              foreach (KeyValuePair<string, object> kvp in alerts)
39              {
40                  var val = GetAlertValueByCategory(message, kvp.Key);
41
42                  var oldBlob = RetrieveBlobByCategory(kvp.Key, formattedDevice);
43
44                  if ((kvp.Value.ToString() == "true") && (val != null))
45                  {
46                      if (oldBlob == "") // if blob storage doesn't have alert with same category
47                      {
48                          string category = kvp.Key;
49                          string status = kvp.Value.ToString();
50
51                          //insert into blob storage
52                          UploadToBlobStorage(formattedHubTime, formattedDevice, category, status, val);
53                          log.LogInformation($"Blob uploaded to storage");
54                      }
55                  }
56                  if (kvp.Value.ToString() == "false")
57                  {
58                      if (oldBlob != "")
59                      {
60                          //separate oldBlob string data for inserting into db
61                          var strList = SeperateBlobStr(oldBlob);
62
63                          //insert into db data from the storage with timestamp of new alert data
64                          bool inserted = InsertIntoAlertTable(strList[0].ToString(),
65                              hubTime, strList[1].ToString(), strList[2].ToString(), strList[4].ToString());
66
67                          if (inserted == true)
68                          {
69                              DeleteOldBlobByCategory(strList[2], strList[1]);
70                              log.LogInformation("Blob deleted");
71                          }
72                      }
73                  }
74              }
75          }
76          catch (Exception ex)
77          {
78              Debug.WriteLine($"ERROR MAIN {ex}");
79          }
```

Kuva 40. Hälytyksiä käsittelevä Azure Funktio

3.3.4 Storage

Queue Storage

Queue Storage otettiin käyttöön datan väliaikaisena sijoituspaikkana hälytysten käsittelyssä. IoT Hubiin saapuva data välitetään Event Gridin avulla Queue Storageen, jolloin hälytyksiä käsittelevä Azure Funktio aktivoituu. Alla olevassa kuvassa (Kuva 41) on näkymä Queue Storagessa olevista viesteistä. Itse viestin sisältö on ”Message text” -sarakkeessa, ”Insertion time” -sarakkeessa näytetään viestien lisäysaika ja ”Expiration time” -sarakkeessa viestien erääntymisajat. ”Dequeue count” -sarakkeessa näytetään kuinka monta kertaa viesti on poistettu jonosta käsittelyä varten.

Id	Message text	Insertion time	Expiration time	Dequeue count
0f15895d-612c-4937...	{"id":"9f36080c-ffd2-69e3-4a10-1cb1cfe775e9","topic":"/SUBSCRI...	9/14/2020, 9:57:53 AM	9/21/2020, 9:57:53 AM	0
9651a86e-b363-435...	{"id":"e1ffba1b-31f7-828f-9cae-46a6e5160a61","topic":"/SUBSCRI...	9/14/2020, 9:57:54 AM	9/21/2020, 9:57:54 AM	0
c19fcc0f-c139-470b-...	{"id":"5865db7f-d40e-ca81-72ef-920ed7a9a394","topic":"/SUBSC...	9/14/2020, 9:58:16 AM	9/21/2020, 9:58:16 AM	0
0cf3b1b3-8ad1-4f80...	{"id":"69e25bb2-6b6a-c3a0-6a2b-90ee1b4b46f7","topic":"/SUBSC...	9/14/2020, 9:58:17 AM	9/21/2020, 9:58:17 AM	0
87d4879b-c022-491...	{"id":"f0c961c5-710f-4598-3df1-6c808c2e34cd","topic":"/SUBSCRI...	9/14/2020, 9:58:40 AM	9/21/2020, 9:58:40 AM	0
0c92bade-78c0-4de...	{"id":"206c6160-b03d-d77f-fe0c-2be846f81e80","topic":"/SUBSCR...	9/14/2020, 9:58:41 AM	9/21/2020, 9:58:41 AM	0

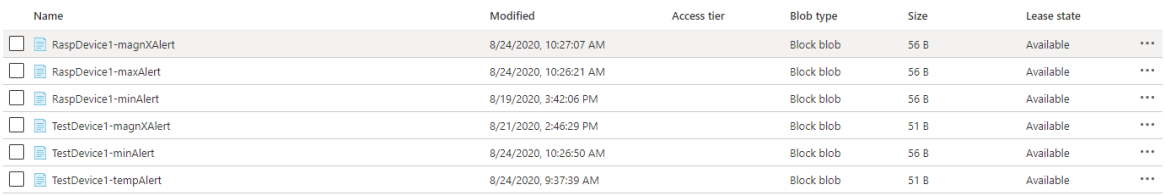
Kuva 41. Queue Storagessa olevat viestit

Seuraavassa kuvassa (Kuva 42) on yksittäisen viestin sisältöä. Hälytysten tilat näkyvät ”data” -osion ”properties” -kohdassa. Hälytysten arvot ovat ”body” -kohdassa Base64 -koodattuina.

Kuva 42. Queue Storagessa olevan viestin ”Message Text” -osion sisältö

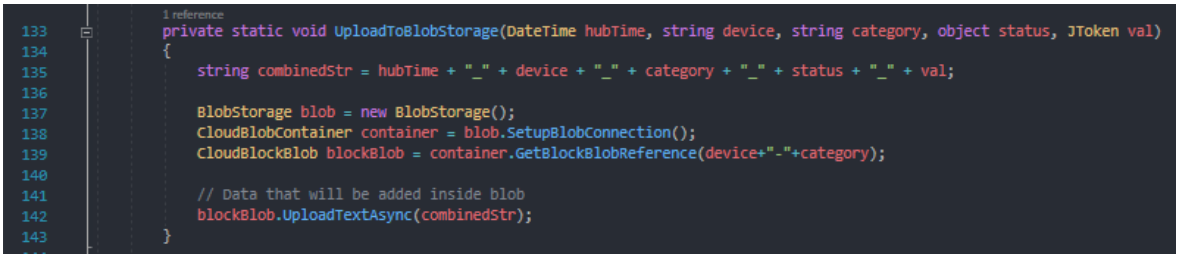
Blob Storage

Projektissa käytettiin väliaikaisena hälytysten tallennuspaikkana Blob Storagea, johon hälytyksiä käsittelevä Azure Funktio tallettaa laitteesta tulevista viesteistä ainoastaan hälytystilassa olevat datat. Data erotellaan kategorian ja laitteen perusteella ja tallennetaan Blob Storageen omiksi Block -objekteiksi. Objektit nimetään ”laite-kategoria” -tyylisesti (Kuva 43).



Kuva 43. Azure Portal -näkymä Blob Storageen tallennetuista hälytyksistä

Seuraavassa kuvassa (Kuva 44) rivillä 135 ensin luodaan arvoista string -muotoinen muuttuja ja tämän jälkeen luodaan yhteys Storageen. Rivillä 139 luodaan objekti ja rivillä 142 aiemmin luotu ”combinedStr” lisätään edellä mainittuun objektiin.

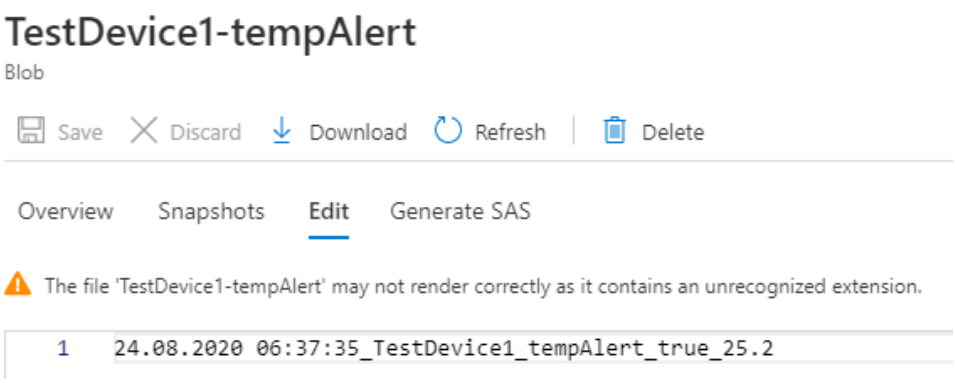


Kuva 44. Datan lisääminen Blob Storageen

Alla olevassa kuvassa (Kuva 45) on objektien haku Blob Storagesta. Joka kerta, kun laitteesta saapuu uusi viesti, Blob Storagesta haetaan sisältö kategorian ja laitteen perusteella. Mikäli haettua objektia ei löydy, funktio palauttaa tyhjän merkkijonon.

Kuva 45. Blob objektien etsimiseen tarkoitettu funktio

Mikäli kyseinen objekti löytyy, sen sisältämät tiedot (kategoria, laite, arvo, alkuaika) sekä uuden viestin sisältämä hälytyksen päättymisaika tallennetaan tietokantaan ”alerts” -tauluun. Tämän jälkeen objekti poistetaan Blob Storagesta. Jokainen objekti sisältää hälytykseen liittyvää dataa tekstimuodossa alla olevan kuvan (Kuva 46) mukaisesti.



Kuva 46. Blob objektin sisältämä data hälytyksen alkaessa

Kun päättyneen hälytyksen tiedot on haettu, ne tallennetaan ”alerts” -tauluun. Kuvassa 47 ensin muuttujien tyyppi muutetaan sopivaksi (rivit 263-265). Tämän jälkeen avataan yhteys tietokantaan (rivi 271) ja valmistellaan kysely sekä muuttujat. Lopuksi komento suoritetaan (rivi 297).

```
private static bool InsertIntoAlertTable(string startTimeStr, DateTime endTime,
    string device, string category, string valueStr)
{
    var connectionString = Environment.GetEnvironmentVariable("IoTDatabaseConnection");
    CultureInfo cultureInfo = new CultureInfo("fi-FI");
    DateTime startTime = DateTime.Parse(startTimeStr, cultureInfo);
    float value = float.Parse(valueStr);

    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        try
        {
            conn.Open();
            SqlCommand command = new SqlCommand(null, conn);

            command.CommandText = "INSERT INTO alerts (device, category, value, starttime, endtime) " +
                "VALUES (@device, @category, @value, @startTime, @endtime)";

            SqlParameter deviceParam = new SqlParameter("@device", SqlDbType.VarChar, 50);
            SqlParameter categoryParam = new SqlParameter("@category", SqlDbType.VarChar, 50);
            SqlParameter valueParam = new SqlParameter("@value", SqlDbType.Float);
            SqlParameter starttimeParam = new SqlParameter("@startTime", SqlDbType.DateTime);
            SqlParameter endtimeParam = new SqlParameter("@endtime", SqlDbType.DateTime);

            deviceParam.Value = device;
            categoryParam.Value = category;
            valueParam.Value = value;
            starttimeParam.Value = startTime;
            endtimeParam.Value = endTime;

            command.Parameters.Add(deviceParam);
            command.Parameters.Add(categoryParam);
            command.Parameters.Add(valueParam);
            command.Parameters.Add(starttimeParam);
            command.Parameters.Add(endtimeParam);

            command.Prepare();

            int check = command.ExecuteNonQuery();

            if (check > 0)
            {
                Debug.WriteLine("INSERTED INTO DB");
                return true;
            }
            else
            {
                return false;
            }
        }
        catch (Exception ex)
        {
            Debug.WriteLine($"ERROR {ex}");
            return false;
        }
    }
```

Kuva 47. Hälytystietojen tietokantaan tallennusta käsittelevä funktio

3.3.5 Tietokanta

Projektia varten luotiin yksi Azure SQL-tietokanta ja useita tauluja. Mittausdata talletetaan ”telemetry” -tauluun ja ”iothubmessagebuffer” -taulu toimii väliaikaisena viestien osien talletuspaikkana ennen ”telemetry” -tauluun talletusta. Hälytykset talletetaan ”alerts” -tauluun. Laitetietojen ja sijainnin säilytystä varten luotiin ”deviceinfo” -taulu ja verkkosivun kirjautumistunnuksia ja rekisteröintiä varten käytettiin useita automaattisesti luotuja tauluja.

Alla olevassa taulukossa (Taulukko 2) esitellään mittausdatan lopullista talletusta varten luotu ”telemetry” -taulu. Taulussa on sarakkeet id:lle, laitteen nimelle, päivämäärälle, ajalle sekä jokaiselle mittaustulokselle. Kahdessa osassa saapuvat viestit talletetaan tauluun yhdelle riville.

Taulukko 2. Datan lopullisena talletuspaikkana toimiva taulu

telemetry		
PK	id	int
	datetime	datetime
FK	device	varchar(50)
	temperature	float
	humidity	float
	pressure	float
	dBFS_min	float
	dBFS_max	float
	dBFS_average	float
	magnAxisX	float
	magnAxisY	float
	magnAxisZ	float
	accelAxisX	float
	accelAxisY	float
	accelAxisZ	float
	gyroAxisX	float
	gyroAxisY	float
	gyroAxisZ	float

Kokonaiset viestit talletetaan seuraavan kuvan mukaisesti (Kuva 48) ”telemetry” -tauluun. Tauluun talletetaan kokonainen viesti yhdelle riville huolimatta siitä, onko osa datan lähetyksestä estetty käyttäjän toimesta. Saapumatta jääneiden arvojen kohdat jäävät tällöin tyhjiksi.

	id	datetime	device	temperature	humidity	pressure	dBFS_min	dBFS_max	dBFS_average	magnAxisX	magnAxisY	magnAxisZ	accelAxisX	accelAxisY	accelAxisZ	gyroAxisX	gyroAxisY	gyroAxisZ
1	1150	2020-06-30 04:15:47.870	TestDevice1	27	48,7999992370605	761,287109375	-138,47380065918	-16,0896530151367	-30,9782257080078	110	-60	-296	0	0	0	0	0	0
2	1151	2020-06-30 04:16:01.763	TestDevice2	27,7999992370605	48,0999984741211	247,37646484375	-144,494400024414	-21,6864242553711	-36,4178009033203	109	-65	-296	-115	2	1033	630	-1890	980
3	1152	2020-06-30 04:16:01.763	TestDevice1	27,7999992370605	48,0999984741211	247,37646484375	-144,494400024414	-21,6864242553711	-36,4178009033203	109	-65	-296	-115	2	1033	630	-1890	980

Kuva 48. Telemetry -tauluun tallennettu data

Viestien ensimmäiset osat talletetaan ”iothubmessagebuffer” -tauluun (Taulukko 3), jossa on samat sarakkeet kuin ”telemetry” -taulussa. Taulun tarkoituksena on pitää tiedot ensimmäisen osan sisällöstä niin kauan, kunnes toinen osa saapuu.

Taulukko 3. Väliaikaisena puskurina toimiva taulu

iothubmessagebuffer		
PK	id	int
	batch	int
	datetime	datetime
FK	device	varchar(50)
	temperature	float
	humidity	float
	pressure	float
	dBFS_min	float
	dBFS_max	float
	dBFS_average	float
	magnAxisX	float
	magnAxisY	float
	magnAxisZ	float
	accelAxisX	float
	accelAxisY	float
	accelAxisZ	float
	gyroAxisX	float
	gyroAxisY	float
	gyroAxisZ	float

Taulun kuuluu pitää ainoastaan yhtä riviä per laite kerrallaan talletettuna (Kuva 49). Virheen sattuessa taulusta poistetaan kaikki virheen aiheuttaman laitteen sisältämät rivit. Tämän tarkoituksena on estää eri viesteistä tulevien osien virheellistä yhdistämistä.

	id	batch	datetime	device	temperature	humidity	pressure	dBFS_min	dBFS_max	dBFS_average	magnAxisX	magnAxisY	magnAxisZ	accelAxisX	accelAxisY	accelAxisZ	gyroAxisX	gyroAxisY	gyroAxisZ
1	1268	1	2020-06-24 05:44:36.253	TestDevice1	31,299999...	37,7999...	1015,924...	-48,12961...	-16,309520...	-22,4199295...	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Kuva 49. Puskuritaulussa oleva data

Hälytysten lopullista talletusta varten luotiin ”alerts” -taulu (Taulukko 4). Tauluun talletetaan laite, kategoria, alkuaika, päättymisaika sekä mitattu arvo hälytyksen alkaessa.

Taulukko 4. ”Alerts” taulu

alerts		
PK	id	int
FK	device	varchar(50)
	category	varchar(50)
	value	float
	starttime	datetime
	endtime	datetime

Seuraavassa kuvassa (Kuva 50) näkyvät laitteen ”TestDevice1” lähettämät äänen tasoon liittyvät hälytykset. Arvot ovat siltä hetkeltä, kun hälytystaso on ensimmäisen kerran ylittynyt.

	id	device	category	value	starttime	endtime
1	126	TestDevice1	minAlert	-144,494400024414	2020-04-08 12:45:06.000	2020-08-04 12:45:53.667
2	127	TestDevice1	minAlert	-117,259841918945	2020-04-08 12:46:17.000	2020-08-04 12:47:04.577
3	128	TestDevice1	maxAlert	-17,7605419158936	2020-04-08 12:45:06.000	2020-08-04 12:48:23.867
4	129	TestDevice1	minAlert	-144,494400024414	2020-04-08 12:48:23.000	2020-08-04 12:48:47.667

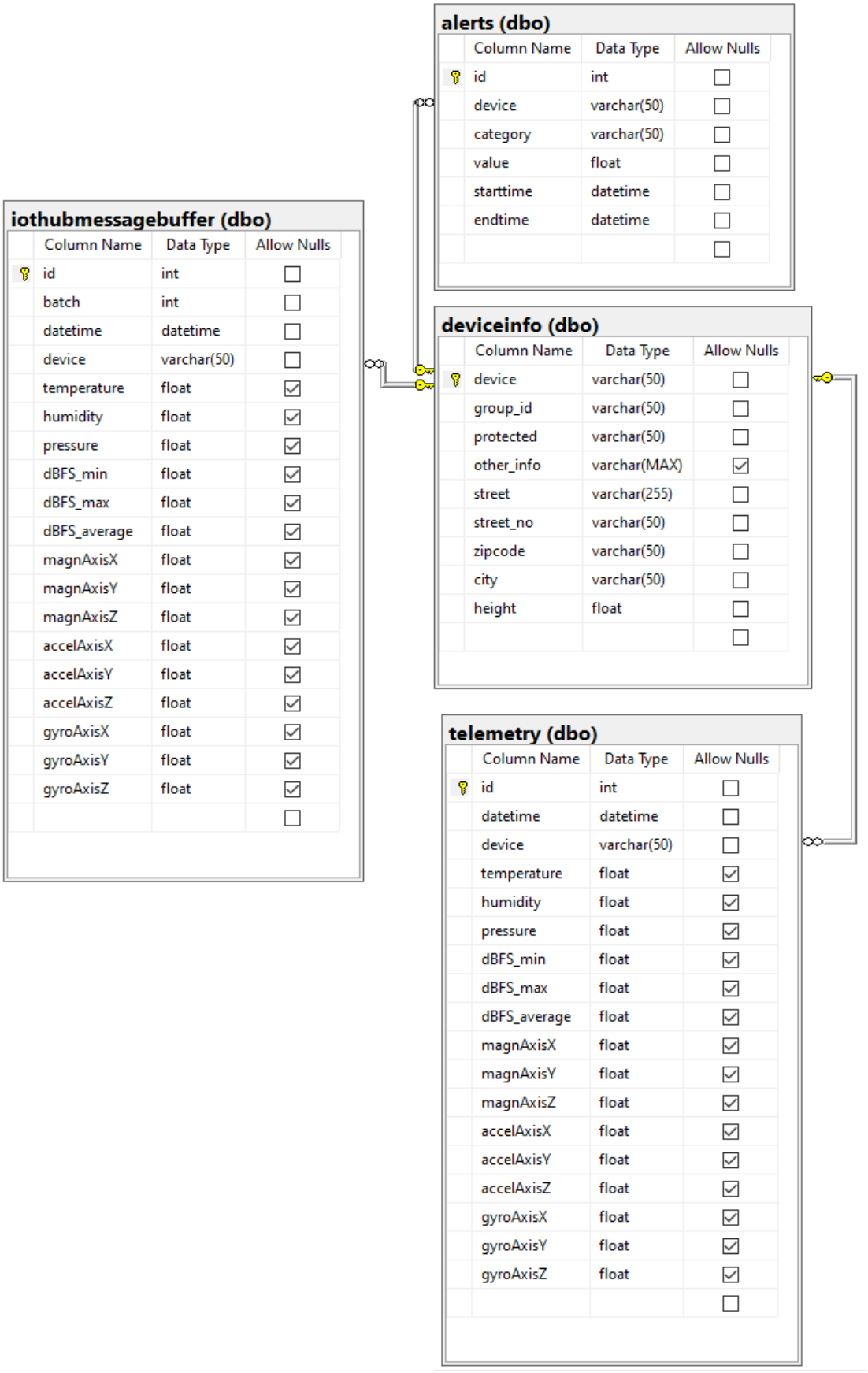
Kuva 50. Alerts -taulussa oleva data

Laitteita koskevien tietojen talletusta varten käyttöön otettiin seuraavanlainen (Taulukko 5) taulu. Tauluun pystyttiin lisäämään laitteiden sijaintitiedot, suojaustasot, ryhmät sekä tarvittaessa lisätietoja.

Taulukko 5. Taulu laitteita koskevia tietoja varten

deviceinfo		
PK	device	varchar(50)
	group_id	varchar(50)
	protected	varchar(50)
	other_info	varchar(MAX)
	street	varchar(50)
	street_no	varchar(50)
	zipcode	varchar(50)
	city	varchar(50)
	height	float

Seuraavassa kuvassa (Kuva 51) näkyvät eri taulujen yhteydet toisiinsa. Deviceinfo -taulun ”device” -sarake on yhteydessä jokaisen edellä mainitun taulun (iothubmessagebuffer, alerts, telemetry) ”device” -sarakkeeseen.



Kuva 51. Eri taulujen väliset yhteydet

4 YHTEENVETO

Opinnäytetyön tavoitteena oli luoda kokonaisuus, jossa data kerätään käyttämällä sensoreita ja tallennetaan pilveen. Yhtenä tärkeistä ominaisuuksista oli myös toteuttaa laitteen etähallinta ja datan visualisointi käyttämällä verkkosivustoa. Projektia varten käytössä oli IoT DevKit MxChip, Arduino -yhteensopiva laite. Datan käsittelyä varten käytettiin Azure Pilvipalveluja. Verkkosivusto toteutettiin Asp.Net:in avulla ja se isännöitiin Azuressa.

Projekti lähti liikkeelle yksittäistavoitteesta saada mitattua melutaso. Desibelisensorin puuttumisen vuoksi päädyttiin käyttämään mikrofonia sekä manuaalista arvojen laskentaa tarvittavan datan saamiseksi. Ongelmana tässä on kuitenkin mikrofonin tuomat rajoitukset verrattuna tähän tarkoitettuun sensoriin. Äänen desibelien laskemisessa oli ongelmana mikrofonin epätarkkuus ja itse arvojen oikeanlainen käsittely. Saatua dataa ei pystytty vertaamaan ulkoiseen lähteeseen eikä siten lasketut arvot ole verrattavissa todellisiin arvoihin. Tavoite kuitenkin toteutettiin siinä, että äänitasojen muutokset ovat havaittavissa vertaillen laitteen itsensä kerättyyn dataan.

Projektin edetessä otettiin sisäänrakennettujen sensoreiden mittaukset mukaan. Kerätyt arvot lähetettiin aluksi IoT Hubiin yhtenä osana, mutta projektin kasvaessa ongelmana tuli IoT Hubin hyväksymä viestien maksimikoko (256 kilotavua), jolloin päädyttiin jakamaan lähetetty data kahteen osaan. Toisena vaihtoehtona olisi ollut myös jokaisen arvon lähettäminen yksitellen, mutta päädyttiin kuitenkin resurssien täyden hyödyntämisen kannalle.

Azuressa on tarjolla laaja työkaluvalikoima, joiden toimintamallissa on päällekkäisyyksiä ja tämän vuoksi eri työkalujen toimivuus tavoitellussa lopputuloksessa saattoi paljastua vasta myöhemmässä vaiheessa. Joitakin ratkaisuja jouduttiin kokeilemaan eri vaihtoehdoilla ennen sopivan löytymistä. Yksi näistä oli tilaa ylläpitävä ”durable funktio”, jonka käyttöä testattiin hälytysten käsittelyssä. Testauksen jälkeen ongelmaksi kuitenkin nousi funktion toimintamalli, jonka tarkoituksena oli ylläpitää tilaa per instanssi. Tarpeen oli kuitenkin pitää tilaa funktioiden suoritusten ja instanssien välissä ja tämän implementointi aiheutti ongelmia instanssien hallinnassa. Vaikka datan tilaa saatiinkin pidettyä suoritusten yli, ongelmana tuli useista eri instansseista johtuva datan hallinnan vaikeus, joka aiheutti duplikaatteja. Yhden instanssin käyttö ei myöskään ollut toimiva ratkaisu, sillä async vaikeutti suoritusajan ennakointia ja aiheutti funktion rikkovia virhetilanteita Azureen

julkaistussa funktiossa. Tästä syystä päädyttiin vaihtamaan tavalliseen Azure Funktioon ja Blob Storageen datan väliaikaiseksi sijoituspaikaksi.

Jatkokehittelyä vaatisi verkkosivuston visuaalinen ilme sekä ratkaisujen yhtenäistäminen. Esimerkiksi ”telemetry” -tauluun tallennettava data siirretään toisen puskurina toimivan taulun kautta, kun taas hälytyksissä käytetään Blob Storagea. Tämän voisi yhtenäistää ja käyttää molemmissa tapauksissa esimerkiksi Blob Storagea. Oletusarvoisesti data ja yhteysavaimet tallennetaan Arduinoon tekstimuodossa, salaamattomana. Laitteeseen tulisi ottaa käyttöön ”security channel”, joka salaa sekä datan, että luku/kirjoitus -operaatiot. Tämä kuitenkin hidastaa kehitysvaiheessa työskentelyä ja tulisi ottaa käyttöön projektin loppuvaiheessa. Jatkokehityssuunnitelmissa on myös Raspberry Pi:n käyttöönotto Arduino -tyyppisten laitteiden lisäksi.

LÄHTEET

Arduino 2020a. Introduction [viitattu 18.7.2020]. Saatavissa: https://www.arduino.cc/en/guide/introduction

Arduino 2020b. Setup [viitattu 26.7.2020]. Saatavissa: https://www.arduino.cc/reference/en/language/structure/sketch/setup/

Arduino 2020c. Loop [viitattu 26.7.2020]. Saatavissa: https://www.arduino.cc/reference/en/language/structure/sketch/loop/

Arduino 2020d. Compare board specs [viitattu 26.7.2020]. Saatavissa: https://www.arduino.cc/en/Products/Compare

Arduino 2020e. Arduino Uno Rev3 [viitattu 22.8.2020]. Saatavissa: https://store.arduino.cc/arduino-uno-rev3

Arduino 2020f. Libraries [viitattu 22.8.2020]. Saatavissa: https://www.arduino.cc/en/reference/libraries

Gartner 2019. Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019 [viitattu 11.9.2020]. Saatavissa: https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g

Gartner 2020. Gartner Forecasts Worldwide Public Cloud Revenue to Grow 6.3% in 2020 [viitattu 8.8.2020]. Saatavissa: https://www.gartner.com/en/newsroom/press-releases/2020-07-23-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-6point3-percent-in-2020

Foote, K. D. 2017. A Brief History of Cloud Computing [viitattu 11.9.2020]. Saatavissa: https://www.dataversity.net/brief-history-cloud-computing/#

Microsoft 2017a. IoT DevKit [viitattu 18.7.2020]. Saatavissa: https://microsoft.github.io/azure-iot-developer-kit/docs/faq/

Microsoft 2017b. Audio IoT DevKit [viitattu 7.8.2020]. Saatavissa: https://microsoft.github.io/azure-iot-developer-kit/docs/apis/audio-v2/

Microsoft 2019a. What is Azure IoT Hub? [viitattu 8.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/iot-hub/about-iot-hub

Microsoft 2019b. Use Azure Functions to connect to an Azure SQL Database [viitattu 22.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/azure-functions/functions-scenario-database-table-cleanup

Microsoft 2019c. Serial communications [viitattu 14.9.2020]. Saatavissa: https://microsoft.github.io/azure-iot-developer-kit/docs/serial-communications/

Microsoft 2020a. Get started guide for Azure developers [viitattu 14.9.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/guides/developer/azure-developer-guide

Microsoft 2020b. IoT Hub message routing query syntax [viitattu 18.7.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-routing-query-syntax#system-properties

Microsoft 2020c. Create and read IoT Hub messages [viitattu 18.7.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-messages-construct

Microsoft 2020d. Create your Azure free account today [viitattu 31.7.2020]. Saatavissa: https://azure.microsoft.com/en-gb/free/

Microsoft 2020e. What is Azure SQL? [viitattu 21.7.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/azure-sql/azure-sql-iaas-vs-paas-what-is-overview

Microsoft 2020f. Tour of Azure services [viitattu 14.9.2020]. Saatavissa: https://docs.microsoft.com/en-us/learn/modules/welcome-to-azure/3-tour-of-azure-services

Microsoft 2020g. An introduction to Azure Functions [viitattu 14.9.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview

Microsoft 2020h. Get the most comprehensive cloud experience [viitattu 8.8.2020]. Saatavissa: https://visualstudio.microsoft.com/vs/features/azure/

Microsoft 2020i. What are Durable Functions? [viitattu 8.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp

Microsoft 2020j. Reference – IoT Hub quotas and throttling [viitattu 14.9.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-quotas-throttling

Microsoft 2020k. Azure MXChip IoT DevKit Get Started [viitattu 9.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/samples/azure-samples/mxchip-iot-devkit-get-started/sample/

Microsoft 2020l. What is Cloud Computing [viitattu 9.8.2020]. Saatavissa: https://azure.microsoft.com/en-gb/overview/what-is-cloud-computing/#benefits

Microsoft 2020m. Introduction to Azure Blob storage [viitattu 9.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction

Microsoft 2020n. Quickstart: Azure Blob storage client library v11 for .NET [viitattu 9.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-dotnet-legacy

Microsoft 2020o. What is SaaS? [viitattu 22.8.2020]. Saatavissa: https://azure.microsoft.com/en-us/overview/what-is-saas/

Microsoft 2020p. Function chaining in Durable Functions – Hello sequence sample [viitattu 22.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-sequence?tabs=csharp

Microsoft 2020q. What is Azure Event Grid? [viitattu 31.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/event-grid/overview

Microsoft 2020r. Concepts in Azure Event Grid [viitattu 31.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/event-grid/concepts

Microsoft 2020s. WebHook Event delivery [viitattu 31.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/event-grid/webhook-event-delivery

Microsoft 2020t. Azure architecture icons [viitattu 31.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/architecture/icons/

Microsoft 2020u. What are Azure queues? [viitattu 31.8.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/storage/queues/storage-queues-introduction

Microsoft 2020v. Azure Functions [viitattu 14.9.2020]. Saatavissa: https://azure.microsoft.com/en-us/services/functions/

Microsoft 2020w. Get started with Azure Queue storage using .NET [viitattu 14.9.2020]. Saatavissa: https://docs.microsoft.com/en-us/azure/storage/queues/storage-dotnet-how-to-use-queues?tabs=dotnet

MxChip 2019a. AZ3166 [viitattu 31.8.2020]. Saatavissa: https://en.mxchip.com/az3166

MxChip 2019b. EMW3166 [viitattu 3.9.2020]. Saatavissa: https://en.mxchip.com/productinfo/244866.html

Statista 2020. Global market share of cloud infrastructure services from 2017 to 2020, by vendor [viitattu 18.7.2020]. https://www.statista.com/statistics/477277/cloud-infrastructure-services-market-share/