LAPIN AMK
Lapland University of Applied Sciences

IoT-järjestelmäarkkitehtuurin suunnittelu ja kokeilu

Hedemäki Kimmo

Opinnäytetyö
Tekniikan ala
Teknologiaosaamisen johtaminen
ylempi AMK

2019

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tekniikka ja liikenne
Teknologiaosaamisen johtaminen
Insinööri (YAMK)

Tekijä(t) Kimmo Hedemäki
Ohjaaja(t) Tauno Tepsa, Outi Hyry-Honka
Vuosi 2019
Toimeksiantaja
Työn nimi IoT-järjestelmäarkkitehtuurin suunnittelu ja kokeilu
Sivu- ja liitemäärä 54 + 0

Alkuperäinen kiinnostus opinnäytetyön aiheeseen tuli asiakkaan esittämästä tarpeesta, asiakkaalla ei ollut valmista järjestelmäarkkitehtuuria saatavilla. Suunnitelmien vaihtumisen johdosta opinnäytetyö kohdistui tutkimaan yleisemmällä tasolla aihetta. Työn tarkoituksena oli suunnitella ja toteuttaa hyvä ja toimiva järjestelmäarkkitehtuurimalli erityisesti IoT-laitteiden ja järjestelmän väliseen kommunikointiin.

Työssä toteutettiin soveltuvin osin konstruktiivista tutkimusmenetelmää. Työ toteutettiin hankkimalla teoreettinen tietämys järjestelmäarkkitehtuurista. Aiheesta ei ollut juurikaan luotettavaa suomalaista aineistoa, joten suurin osa tutkimusaineistosta on kansainvälisistä tietolähteistä. Tietoa aiheesta kerättiin kirjoista, artikkeleista ja internetistä. Tietämystä kohdennettiin ja syvennettiin tutkimaan eri järjestelmäarkkitehtuurin suunnittelumalleja sekä soveltuvin osin IoT-laitteiden kommunikaatioprotokollavaihtoehtoja.

Teoreettisen tietämyksen pohjalta tehtiin kokeilu. Kokeilussa yhdistettiin useita eri IoT-laitteiden tietoja järjestelmään. Kokeilussa tuotettu järjestelmä hyödynsi Google Cloud-alustan palveluita hyödykseen. Yksi järjestelmänarkkitehtuurin lähtökohdista oli tehdä suunnittelu niin, että järjestelmäarkkitehtuuri ei rajoittaisi järjestelmän ympäristöä. Kokeilussa onnistuttiin hyvin, ja saatiin liitettyä useampi eri IoT-laite järjestelmään. Teoreettisesti opinnäytetyö vastasi sille asetettuihin vaatimuksiin.

Erilaisten IoT-laitteiden tuomat hyödyt tulevat olemaan tulevaisuudessa merkittävässä roolissa rakennettaessa liiketoiminnallisuuksia. Opinnäytetyön ratkaisuja voidaan laajasti ja sovelletusti hyödyntää uusia liiketoimintamahdollisuuksia rakennettaessa.

Asiasanat järjestelmäarkkitehtuuri, kommunikointiprotokolla, IoT, konstruktiivinen tutkimus

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Technology Competence Management
Master of Engineering

Author(s)	Kimmo Hedemäki	Year	2019
Supervisor(s)	Tauno Tepsa, Outi Hyry-Honka		
Commissioned by			
Subject of thesis	IoT System Architecture Design and Experiment		
Number of pages	54 + 0		

The initial interest in the thesis topic came from a customer need. A customer did not have a solution for communication between an IoT devices and existing system. As a result of the change of plans, the thesis focused on a more general study of the topic. The object was to design and implement a well-functioning system architecture design so that an IoT devices can communicate between information systems.

In the thesis, a constructive research method was applied where applicable. The work was started by acquiring theoretical knowledge of system architecture design and IoT communication protocol alternatives. There was a lack of reliable Finnish material on the subject, thus most research material was available in international books, articles and technical literature.

Based on a theoretical knowledge, an experiment was done. The experiment combined information from several IoT devices into the system. An experiment used the Google Cloud platform services. One of the starting points of the system architecture design so that the system architecture would not limit the environment in use. The experiment was successful, and it was possible to connect several different IoT devices to the system and visualize data from the devices. Theoretically, the thesis met the requirements set to it.

Key words system architecture, communication protocol, constructive research, IoT

SISÄLLYS

1 JOHDANTO 1
1.1 Työn tavoite ja tarkoitus 1
1.2 Työn rajaus ja rakenne 1
1.3 Kehittämismenetelmä 2
2 JÄRJESTELMÄARKKITEHTUURI 4
2.1 Järjestelmäarkkitehtuurin tarkoitus 4
2.2 Hyvän järjestelmäarkkitehtuurin tunnusperiaatteet 4
2.2.1 Järjestelmän vaatimusten täyttäminen 5
2.2.2 Järjestelmän ylläpito 5
2.2.3 Järjestelmän kehittäminen 5
2.2.4 Järjestelmän käyttöönotto 6
2.3 Järjestelmäarkkitehtuurin suunnitteluperiaatteet 6
2.3.1 Ylhäältä-Alas-lähestymistapa 7
2.3.2 Alhaalta-Ylös-lähestymistapa 7
3 JÄRJESTELMÄARKKITEHTUURIN VAIHTOEHDOT 9
3.1 Suunnittelumallit 9
3.2 Monoliittinen järjestelmäarkkitehtuurimalli 9
3.3 Palvelukeskeinen järjestelmäarkkitehtuuri 11
3.3.1 Liiketoimintakeskeinen lähestymistapa 13
3.3.2 Teknologiariippumaton lähestymistapa 13
3.3.3 Organisaatiokeskeinen lähestymistapa 14
3.3.4 Yleiskäyttöinen lähestymistapa 14
3.4 Mikropalvelut 14
3.5 API-yhdyskäytävä 18
3.5.1 Järjestelmän käytettävyys ja tietoturva 18
3.5.2 Transformaatio 18
3.5.3 Monitorointi 19
3.5.4 Vikasietoisuus 19
3.6 Tapahtumakeskeinen järjestelmäarkkitehtuuri 20
3.7 Dokumentointi 22
4 IOT- LAITTEIDEN KOMMUNIKOINTIPROTOKOLLAT 24

4.1 Lyhyen kantaman protokollat....24
4.1.1 Bluetooth LE....24
4.1.2 Z-Wave....26
4.1.3 ZigBee....27
4.2 Pitkän kantaman laiteläheiset protokollat....28
4.2.1 LoRaWan....28
4.2.2 Sigfox....29
4.2.3 NB-IoT....29
4.3 TCP/IP-standardi....30
4.4 Web-Service....31
4.4.1 SOAP....32
4.4.2 REST....33
4.5 MQTT....33
5 IOT-ALUSTAN JÄRJESTELMÄARKKITEHTUURIN SUUNNITTELU JA TOTEUTUS....36
5.1 IoT-yhdyskäytävä....36
5.2 API-yhdyskäytävä....38
5.3 Mikropalvelut....41
5.4 Pitkäkestoiset palvelut....44
5.5 Jonopalvelut....47
5.6 Tietokantapalvelut....50
6 KOKEILUN TULOKSET....52
7 POHDINTA JA JOHTOPÄÄTÖKSET....53
LÄHTEET....55

TAULUKKOLUETTELO

Taulukko 1. Listaus IoT-laitteista..37

KUVIOLUETTELO

Kuvio 1. Esimerkki konstruktiivisen tutkimuksen prosessista 2
Kuvio 2. Esimerkki monoliittisestä järjestelmäarkkitehtuurimallista 10
Kuvio 3. Esimerkki palvelukeskeisestä järjestelmäarkkitehtuurista 12
Kuvio 4. Esimerkki mikropalveluiden järjestelmäarkkitehtuurista 15
Kuvio 5. Esimerkki mikropalveluiden yhdistämisen vaihtoehdoista...................16
Kuvio 6. API-yhdyskäytävä ratkaisun kuormanjako. 20
Kuvio 7. Esimerkki Event Driven-arkkitehtuurista..21
Kuvio 8. Esimerkki Bluetooth LE-verkkojen periaatteesta.................................25
Kuvio 9. Esimerkki MESH-verkon periaatteesta...27
Kuvio 10. Esimerkki Web-palveluiden rooleista 32
Kuvio 11. Esimerkki MQTT-julkaisija/tilaaja järjestelmäarkkitehtuurista 34
Kuvio 12. Looginen kuvaus IoT-yhdyskäytävästä 37
Kuvio 13. Esimerkki binäärimuotoisen tiedon muuttamisesta 38
Kuvio 14. Looginen kuvaus API-yhdyskäytävästä..40
Kuvio 15. Esimerkki sertifikaatin luomisesta laitteelle 40
Kuvio 16. Looginen kuvaus API-yhdyskäytävästä..41
Kuvio 17. Esimerkki Google Cloud function-mikropalvelusta 43
Kuvio 18. Esimerkki Google Cloud function-statistiikasta 44
Kuvio 19. Esimerkki virtuaalipalvelimen luonnista...45
Kuvio 20. Esimerkki virtuaalipalvelimen luonnista...46
Kuvio 21. Esimerkki antureiden tiedon hyödyntämisestä 46
Kuvio 22. Esimerkki virtuaalipalvelimen luonnista...47
Kuvio 23. Tilastotietoa Google Cloud-alustan MQTT-jonosta 48
Kuvio 24. MQTT-viestin monistaminen 49
Kuvio 25. Google Cloud MQTT-tilauksen luominen 49
Kuvio 26. MQTT-tilauksen tilastotietoja..50
Kuvio 27. Tietokannan datarakenne 51

1 JOHDANTO

1.1 Työn tavoite ja tarkoitus

Tämän kehitystyön tarkoituksena on suunnitella ja toteuttaa toimiva järjestelmäarkkitehtuurimalli erityisesti IoT-laitteiden ja järjestelmän väliseen kommunikointiin. Mahdollisuudet laitteiden väliseen kommunikaatioon ja sen ympärille toteutettaviin tietojärjestelmiin ovat vuoden 2019 yleinen puheenaihe. TechRepublic nostaa omassa tutkimuksessaan esille IoT-laitteet, ja niiden tarjoamat mahdollisuudet myynnin lisäyksenä ja säästöjen mahdollistajana yhdeksi 10 tärkeimmän teknologian joukkoon (Top 10 emerging technologies of 2019). Kun laitteiden välinen tiedonvaihto mahdollistetaan ja tiedonvaihto yleistyy, lisääntyy yritysten tietoisuus niiden tuomista hyödyistä. Näin syntyy uudenlaisia liiketoimintamahdollisuuksia. Opinnäytetyössä selvitetään ja avataan yleiskatsauksen omaisesti eri IoT-teknologioita ja niiden ominaispiirteitä. Selvityksen jälkeen toteutetaan kokeilu.

1.2 Työn rajaus ja rakenne

Työn näkökulmana on erityisesti IoT-laitteiden yhdistettävyys olemassa olevaan järjestelmään sekä yleisesti toimivien järjestelmäarkkitehtuurimallin periaatteiden selvittäminen. Työ rajataan koskemaan vain järjestelmäarkkitehtuurin suunnittelua. IoT-laitteiden kommunikointia protokollatasolla ei käsitellä, eikä kehitetä. Kokeilua varten käytetään kommunikointiin valmiita ulkopuolisia ohjelmistokirjastoja. Työssä tarkastellaan kuutta Euroopassa käytössä olevaa yleistä kommunikointiprotokollaa. Tässä työssä esitetty järjestelmäarkkitehtuurimalli on laajennettavissa käyttämään mitä tahansa protokollaa IoT-laitteiden kanssa, joten kokeilun tekemiseen oli kolme eri protokollaa riittävä laajuus.

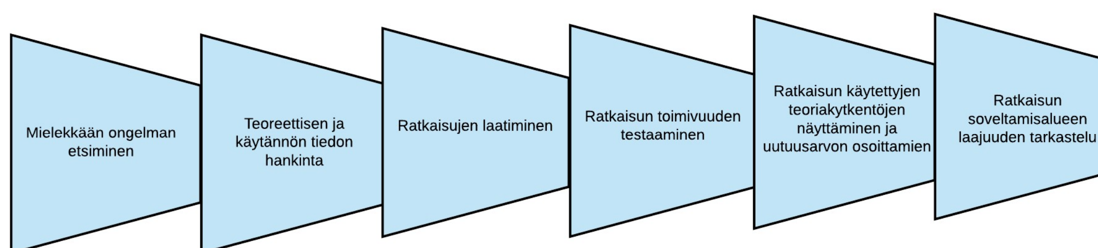
Työ on jaettu seitsemään eri päälukuun. Johdannossa esitellään työn tavoite ja rajaukset. Toisessa luvussa käydään läpi, mitä tarkoittaa järjestelmäarkkitehtuuri ja toimiva järjestelmäarkkitehtuuri. Kolmannessa luvussa esitellään vaihtoehtoisia järjestelmäarkkitehtuureja. Luvut kaksi ja kolme toimivat myös pohjana järjestelmäarkkitehtuurikokeilun suunnitteluvaiheessa. Neljännessä luvussa käsitellään eri kommunikointiprotokollia. Tarkasteluun on valittu työn kohteeseen soveltuvat kommunikointiprotokollat. Viidennessä luvussa suunnitellaan ja toteutetaan järjestelmäarkkitehtuurikokeilu. Kuudennessa luvussa käydään läpi toteutetun

kokeilun tulokset. Viimeisessä luvussa käydään läpi työn lopputulokset ja päätelmät.

1.3 Kehittämismenetelmä

Työn tarkoitus on suunnitella järjestelmäarkkitehtuuri ja toteuttaa järjestelmäarkkitehtuurikokeilu, joka toimisi mahdollisimman hyvin IoT-laitteiden kanssa ja olisi helposti laajennettavissa tulevaisuuden tarpeisiin. Järjestelmäarkkitehtuurin suunnittelun jälkeen toteutetaan järjestelmäarkkitehtuurikokeilu. Kokeilun perusteella arvioidaan järjestelmäarkkitehtuurisuunnitelman toimivuus ja lähtökohtien sopivuus.

Konstruktiivisen tutkimuksen tavoite on ratkaista käytännön ongelma luomalla konkreettinen tuotos, uusi konstruktio. Konstruktion luomisen perustana käytetään hankittua teoreettista tietoa, joka yhdistetään empiiriseen tietoon eli käytännössä opittuun tietoon. Konstruktiivisen tutkimuksen tuloksena saadaan kohdeorganisaatioon uutta teoreettista tietoa ja yleensä uusi konstruktio otetaan käyttöön sekä toimivuus testataan kohdeorganisaatiossa. (Ojasalo, Moilanen & Ritalahti 2009, 38, 65–66.)



Kuvio 1. Esimerkki konstruktiivisen tutkimuksen prosessista

Kuviossa 1 kuvataan konstruktiivisen tutkimuksen prosessi (Ojasalo ym. 2009, 67–68). Prosessin ensimmäinen vaihe on määrittää ja valita sopiva sekä mielekäs ongelma. Ongelman määrittäminen on tutkimuksen tärkein vaihe. Lukka (2001) korostaa, että tutkijan pitäisi ongelman määrittämisessä pohtia asiaa sekä käytännön että teorian kannalta. Ongelman määrittämisen jälkeen alkaa teoreettisen ja käytännön tiedon hankinta. Tiedon hankinnan jälkeen toteutetaan konstruktio eli ratkaisu alkuperäiseen ongelmaan. Ratkaisun toimivuus testataan todellisessa ympäristössä ja osoitetaan ratkaisun oikeellisuus. Lopuksi esitetään

ratkaisun teoriakytkentä, uutuusarvo sekä soveltamisalueen laajuus. (Ojasalo ym. 2006, 67–68.)

Tässä työssä on noudatettu soveltuvin osin edellä kuvattua prosessia. Työssä toteutuvat konstruktiivisen tutkimusmenetelmän seuraavat vaiheet: ongelman asettaminen, teoreettinen tiedon hankinta ja ratkaisun laatiminen. Työssä tehtiin kokeilu, mutta kokeilua ei otettu käyttöön todellisessa ympäristössä. Tästä syystä ratkaisun testaaminen, teoriakytkennän, uutuusarvon ja soveltamisalueen laajuuden tarkastamista voitiin tehdä vain osittain.

Teoreettinen tietämys kehitystyöhön on hankittu kirjallisuudesta liittyen järjestelmäarkkitehtuurin suunnitteluun ja oleellisiin kommunikointiprotokolliin. Teoreettinen tiedonhankinta on keskittynyt lähes poikkeuksetta kansainvälisiin tutkimuksiin ja kirjallisuuteen. Luotettavaa tieteellistä suomenkielistä kirjallisuutta IoT-laitteiden järjestelmäarkkitehtuurin suunnittelusta ei juurikaan löydy. Kansainvälistä tietoa kehittämistyön aihepiiristä on olemassa paljon.

2 JÄRJESTELMÄARKKITEHTUURI

2.1 Järjestelmäarkkitehtuurin tarkoitus

Yleisesti ajatellaan, että järjestelmäarkkitehtuurin tarkoitus on saada järjestelmä toimimaan tarkoituksenmukaisesti. Järjestelmäarkkitehtuurin määrittely itsessään ei takaa sitä, että järjestelmä toimisi tarkoituksenmukaisesti. Järjestelmäarkkitehtuurin suunnittelussa tulisi huomioida, että suunnittelu tukee järjestelmän toimintaa riittävän hyvällä tavalla. Martin (2018) määrittää järjestelmäarkkitehtuurisuunnitteluksi järjestelmän kuvaamisen käyttäen eri komponenttien kuvauksia muodostaen järjestelmästä loogisen kuvauksien kokoelman. Yhdistämällä yksittäisten komponenttien kuvaukset kokoelmaksi saadaan muodostettua järjestelmäkokonaisuus. (Martin 2018, 136–137.)

Järjestelmäkokonaisuudessa on otettu huomioon eri komponenttien kommunikointi, järjestelmän kehittäminen, toiminta ja ylläpito. Järjestelmänarkkitehtuurin tärkein tehtävä on tukea järjestelmän koko elinkaarta, aina järjestelmän suunnittelusta ylläpitoon asti. Hyvän järjestelmäarkkitehtuurin tunnuspiirre on, että järjestelmäarkkitehtuurin avulla on helppo ymmärtää järjestelmän toiminta. Järjestelmäarkkitehtuuri tukee kehittämistä, ylläpitämistä ja rakentamista. Toisin sanoen järjestelmäarkkitehtuurin tarkoitus on maksimoida järjestelmän kehitys ja minimoida järjestelmäarkkitehtuurista johtuvat kustannukset. (Martin 2018, 136–137.)

Järjestelmäarkkitehtuurin suunnittelussa tulisi huomioida ja tunnistaa tärkeimmät komponentit järjestelmän toiminnan kannalta. Järjestelmäarkkitehtuurin suunnittelu on päätösten tekemistä varhaisessa vaiheessa liittyen tulevaan järjestelmään. Mitä myöhemmässä vaiheessa järjestelmään joudutaan tekemään muutoksia, sitä kalliimpaa, hankalampaa ja aikaa vievää niiden toteuttaminen on. Tämän takia järjestelmäarkkitehtuurisuunnittelu pitää tehdä alusta alkaen kunnolla ja järjestelmäarkkitehtuuria koskevat päätökset on syytä tehdä huolella. (Ingalo 2018, 10-11.)

2.2 Hyvän järjestelmäarkkitehtuurin tunnusperiaatteet

Hyvän järjestelmäarkkitehtuurin tunnuspiirteitä ovat järjestelmän ulkoisten kirjastojen riippumattomuus, testattavuus, käyttöliittymän riippumattomuus, tietokannan riippumattomuus sekä ulkoisten toimijoiden riippumattomuus. Martin nostaa

esille neljä eri kohtaa, mitä hyvän järjestelmäarkkitehtuurin tulisi tukea. Järjestelmäarkkitehtuurin tulisi mahdollistaa järjestelmälle asetetut vaatimukset, tukea järjestelmän ylläpitoa ja mahdollistaa järjestelmän kehittäminen sekä käyttöönotto. (Martin 2018, 148.)

2.2.1 Järjestelmän vaatimusten täyttäminen

Hyvän järjestelmäarkkitehtuurin ensimmäinen sääntö on, että sen tulee tukea järjestelmälle asetettuja vaatimuksia ja järjestelmän tarkoitusta. Esimerkiksi, jos vaatimuksena järjestelmälle on kyetä todella suureen käyttöasteeseen, pitää järjestelmäarkkitehtuurin tukea tätä ratkaisua ja käyttöastevaatimus pitää ottaa huomioon järjestelmäarkkitehtuurin suunnittelussa. Järjestelmäarkkitehtuurin tulee siis ilmentää järjestelmän käyttötarkoitusta. Martin (2018) kuvaa, että esimerkiksi ostoskoriohjelman järjestelmäarkkitehtuurin pitää näyttää ostoskoriohjelmaan sopivalta sekä tukea ostoskorijärjestelmän toiminnallisuuksia. Järjestelmän käyttötapausten pitää selvästi olla esillä järjestelmäarkkitehtuurin rakenteiden muodossa. Järjestelmäarkkitehtuurin elementtien pitää olla selkeitä ja nimetty elementin tarkoituksen mukaan. Järjestelmäarkkitehtuurin rakenteiden selkeys auttaa ymmärtämään järjestelmän toimintaa ja ymmärtämään järjestelmään tehtävien muutoksien aiheuttamia vaikutuksia. (Martin 2018, 148-149.)

2.2.2 Järjestelmän ylläpito

Ylläpito on järjestelmän elinkaaressa yleensä aikaa ja resursseja vievää. Uuden toiminnallisuuden tekeminen, tai virheen korjaus, voi tarkoittaa muutoksia jo olemassa oleviin toiminnallisuuksiin. Muutoksista aiheutuu aina lisätyötä. Lisäksi jokaisessa muutoksessa on riski aiheuttaa uusia ongelmia järjestelmään. Hyvä järjestelmäarkkitehtuuri minimoi edellä kuvattuja riskejä jakamalla järjestelmän komponentit loogisiin kokonaisuuksiin ja suunnittelemalla komponenttien välille vakaat rajapinnat. Näin yhden komponentin muutos ei aiheuta muutoksia toiseen komponenttiin. Lisäksi ongelmatapauksessa vika voidaan rajata tarkasti järjestelmän tiettyyn komponenttiin. Korjaavat toimenpiteet voidaan suunnitella ja tehdä vain vikaantuneeseen komponenttiin. (Martin 2018, 139–140.)

2.2.3 Järjestelmän kehittäminen

Järjestelmäarkkitehtuurilla on erittäin oleellinen vaikutus järjestelmän kehityksen aikana. Usein järjestelmää kehitetään eri tiimien kanssa. Tiimit voivat olla eri

organisaatiosta eivätkä välttämättä kommunikoi kehityksen aikana toistensa kanssa. Tämän takia on tärkeää, että järjestelmäarkkitehtuuri on suunniteltu ja jaoteltu järkeviin loogisiin ja erikseen toteutettaviin kokonaisuuksiin. Näin järjestelmän eri kokonaisuuksia voidaan kehittää rinnakkain ilman, että siitä aiheutuu ongelmia myöhemmässä vaiheessa eikä tiimien välille synny riippuvuutta. Esteitä työn etenemiseen tiimien välillä syntyy, jos kehitetään arkkitehtuurin eri komponentteja, joista on suora riippuvuus toiseen järjestelmän komponenttiin. Tällöin voi tulla tilanne, että toinen tiimi joutuu odottamaan toisen tiimin työn valmistumista, ennen kuin voi edistää omia työtehtäviä. (Martin 2018, 148-149.)

2.2.4 Järjestelmän käyttöönotto

Tehokkaan järjestelmän pitää olla helposti käyttöönotettava. Mitä enemmän aikaa ja rahaa menee järjestelmän käyttöönottoon, sitä tehottomampi järjestelmä kokonaisuudessaan on. Hyvän järjestelmäarkkitehtuurin pitää tukea järjestelmän käyttöönottoa helposti. Järjestelmäarkkitehtuuri pitäisi olla jaoteltu niin, että sen yksittäiset osat voidaan yhdessä tai erikseen päivittää ja käyttöönottaa. Lisäksi järjestelmäarkkitehtuurin pitää tukea käyttöönottoprosessin selkeyttä. Mitä vähemmän käyttöönottoprosessissa on vaiheita, sitä tehokkaampaa se on. (Martin 2018, 138.)

2.3 Järjestelmäarkkitehtuurin suunnitteluperiaatteet

Järjestelmäarkkitehtuurin suunnittelu on avainasemassa, kun rakennetaan toimivia järjestelmiä. Valmis suunnitelma toimii teknisenä pohjana järjestelmän toteuttajille. Järjestelmäarkkitehtuurin suunnitteluprosessi koostuu päätöksistä ja linjauksista, joiden pohjalta järjestelmän tulee täyttää sille asetetut toiminnalliset vaatimukset. Jokaisessa päätöksessä arkkitehdin pitää punnita päätöksen hyviä ja huonoja puolia koko järjestelmän elinkaaren kannalta. Arkkitehdin pitää tehdä päätös, joka hyödyttää järjestelmää pitkällä tähtäimellä parhaiten. Päätöksen tekemiseen on hyvä osallistuttaa useampien osa-alueiden osaajia ja tehdä päätöksestä useita iteraatioita eli tehdä pieniä muutoksia ja kokeilla käytännössä, oliko tehty päätös oikea. Kokeilussa kerätyn tiedon pohjalta tehdään tarvittavia muutoksia aikaisempiin päätöksiin. Järjestelmäarkkitehtuurin suunnittelu tehdään dokumentoimalla järjestelmän eri elementit ja elementtien väliset riippuvuudet ja suhteet. Järjestelmäarkkitehtuurin suunnittelussa on kaksi eri lähestymistapaa ylhäältä-alas ja alhaalta-ylös. (Ingeno 2018, 111, 115.)

2.3.1 Ylhäältä-Alas-lähestymistapa

Ylhäältä-Alas-lähestymistavassa (Top-Down) suunnittelussa lähdetään kuvaamaan järjestelmän komponentteja niin sanotulla ylätason kuvauksella. Tämä tarkoittaa sitä, että järjestelmä mallinnetaan ilman tarkempia kannanottoja käytettäviin yleishyödyllisiin sovelluskirjastoihin tai teknisiin yksityiskohtiin. Kun ylätaso on valmiiksi suunniteltu, siirrytään tarkempien tietojen kuvaamiseen. Ylhäältä-Alas-lähestymistavassa suunnittelu tehdään systemaattisesti iteroiden ja jokaisella iterointikierroksella tarkennetaan suunnitelmaa. Iteraatiokierrosten hyöty on se, että suunnittelua voidaan jakaa pienimpiin kokonaisuuksiin ja näin ollen toteuttaa useamman henkilön kesken. Ylhäältä-Alas-lähestymistapa on erityisen sopiva laajoihin järjestelmäarkkitehtuurin suunnitteluihin, mutta sitä voidaan hyödyntää myös pienissä projekteissa. (Ingeno 2018, 117–118.)

Järjestelmäarkkitehtuurin mallintaminen ylätasolta alaspäin vaatii sen, että suunnittelun kohde on hyvin ymmärretty jo varhaisessa vaiheessa. Kun ylätasolle joudutaan myöhemmässä vaiheessa tekemään muutoksia, voi niistä seurata muutoksia koko järjestelmäarkkitehtuurin laajuuteen. Näin laajat muutokset ovat yleensä aikaa vieviä ja vaativat paljon resursseja. Toinen haaste Ylhäältä-Alas lähestymistavassa on se, että arkkitehti helposti keskittyy vain tekemään ylätason suunnittelun ja tarkempi kuvaus jää tekemättä. Tarkempi kuvaus on kuitenkin erittäin oleellinen ohjenuora järjestelmän toteuttajille. Ilman tarkempaa kuvausta järjestelmäarkkitehtuuri ei tue järjestelmän kehitystä tai ylläpitoa. (Ingeno 2019, 118.)

2.3.2 Alhaalta-Ylös-lähestymistapa

Toisin kuin Ylhäältä-Alas malli Alhaalta-Ylös-lähestymistavassa (Bottom-Up) järjestelmän suunnittelu aloitetaan miettimällä, mitä komponentteja järjestelmä tarvitsee toimiakseen. Kun yksittäiset komponentit ovat tunnistettu ja suunniteltu, aletaan suunnittelemaan niiden keskinäistä toimintaa ja näin ollen isompia kokonaisuuksia. Järjestelmäarkkitehtuuri rakentuu näin pieni pala kerrallaan. Alhaalta-Ylös-lähestymistavan etuna on, että järjestelmän kohteen ei tarvitse olla erityisen tarkkaan tiedossa järjestelmäarkkitehtuurin suunnittelun alkuvaiheessa. Laajempi kokonaisuus tarkentuu pienempien kokonaisuuksien valmistumisen seurauksena. Alhaalta-Ylös-lähestymistapa sopii Agile-menetelmiä käyttäville tiimeille. Agile-kehitysmetodeilla tarkoitetaan niin sanottuja ketteriä

kehitysmenetelmiä. Ketterässä kehitysmenetelmässä kehitetään pieniä kokonaisuuksia kerrallaan kokonaisen järjestelmän kehittämisen sijaan. Muutokset järjestelmäarkkitehtuurin suunnittelussa voidaan toteuttaa ilman riippuvuuksia, koska suunnittelua tehdään pieniä palasia kerrallaan ja muutokset eivät vaikuta koko järjestelmään. (Ingeno 2018, 118-119.)

3 JÄRJESTELMÄARKKITEHTUURIN VAIHTOEHDOT

3.1 Suunnittelumallit

On olemassa useita eri järjestelmäarkkitehtuurin suunnittelumalleja. On tärkeää tiedostaa eri konseptien toimivat ja kehittämistä vaativat ominaisuudet. Eri mallit ovat suunniteltu aina erityiseen ongelmaan tai tarpeeseen. Tärkeää on myös tehdä valinta sillä perusteella, joka tukee mahdollisimman hyvin järjestelmän tarkoitusta. Järjestelmäarkkitehtuurimallin valintaan vaikuttavat arkkitehdin kokemus, kollegoiden mielipiteet sekä eri mallien vahvuudet. Aikaisempien järjestelmien yhteydessä kerätystä tiedosta on myös apua järjestelmäarkkitehtuurimallin valinnassa. (Ingeno 2018, 129.)

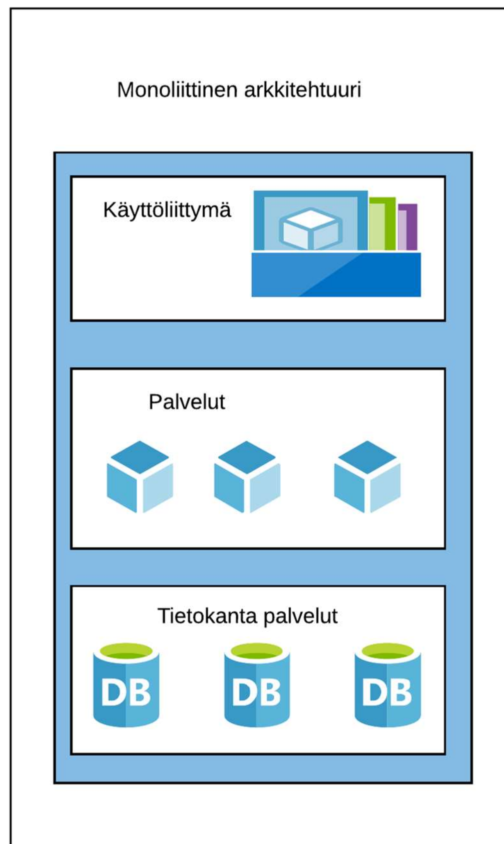
Valmis järjestelmäarkkitehtuurimalli on tärkeä apuväline arkkitehdeille, sillä se tarjoaa valmiiksi testatut ratkaisut mahdollisiin ongelmakohtiin. Oikean järjestelmäarkkitehtuurimallin valinta auttaa järjestelmän kehityksessä. Uusien ominaisuuksien tekeminen on nopeampaa ja suunnitteluun menee vähemmän aikaa. Oikean järjestelmäarkkitehtuurimallin valinta parantaa järjestelmän laatua. (Ingeno 2018, 130.) Tutkimukseen valittiin neljä erilaista järjestelmäarkkitehtuurimallia. Esille pyrittiin nostamaan jokaisen järjestelmäarkkitehtuurimallin vahvuudet ja heikkoudet sekä tutkimaan niiden soveltamista IoT-integraatioarkkitehtuuriin.

3.2 Monoliittinen järjestelmäarkkitehtuurimalli

Monoliittisessa (Monolithic) järjestelmäarkkitehtuurimallissa kaikki sovelluksen komponentit on yhdistetty yhteen julkaisuun. Julkaisulla tarkoitetaan sovelluspakettia. Kuten Strimbei, Catalin, Dospinescu, Strainu ja Nistor (2015, 14) nostavat esille, ovat monoliittisen järjestelmäarkkitehtuurimallin hyötyjä eri sovelluskomponenttien välinen kommunikointi, koko prosessin näkeminen kerralla sekä uusien ominaisuuksien kehittäminen.

Monoliittisen järjestelmäarkkitehtuurimallin hyötyjä ovat sen nopeampi ja ketterämpi kehittäminen, testaaminen ja jakeleminen. Usein pienissä sovelluskokonaisuuksissa tai hyvin pienissä sovelluskehitystiimeissä on tehokkainta valita monoliittinen järjestelmäarkkitehtuurimalli, sillä kehittäminen ja julkaiseminen vie vähemmän aikaa kuin hajautettu järjestelmäarkkitehtuurimallin mukainen sovellus. (Strimbei ym 2015, 14.)

Kuviossa 2 kuvataan esimerkki monoliittisen järjestelmäarkkitehtuurin rakenteesta. Samaan sovellusjulkaisuun on sisällytetty järjestelmän eri komponentit. Alimpana kuvassa on järjestelmän käyttämät tietovarannot. Seuraava kerros on järjestelmä palvelut eli niin sanottu logiikkakerros. Ylimmäisenä kerroksena on käyttöliittymäkerros, jonka tehtävä on toteuttaa käyttäjille mahdollisuus käyttää järjestelmää.



Kuvio 2. Esimerkki monoliittisestä järjestelmäarkkitehtuurimallista

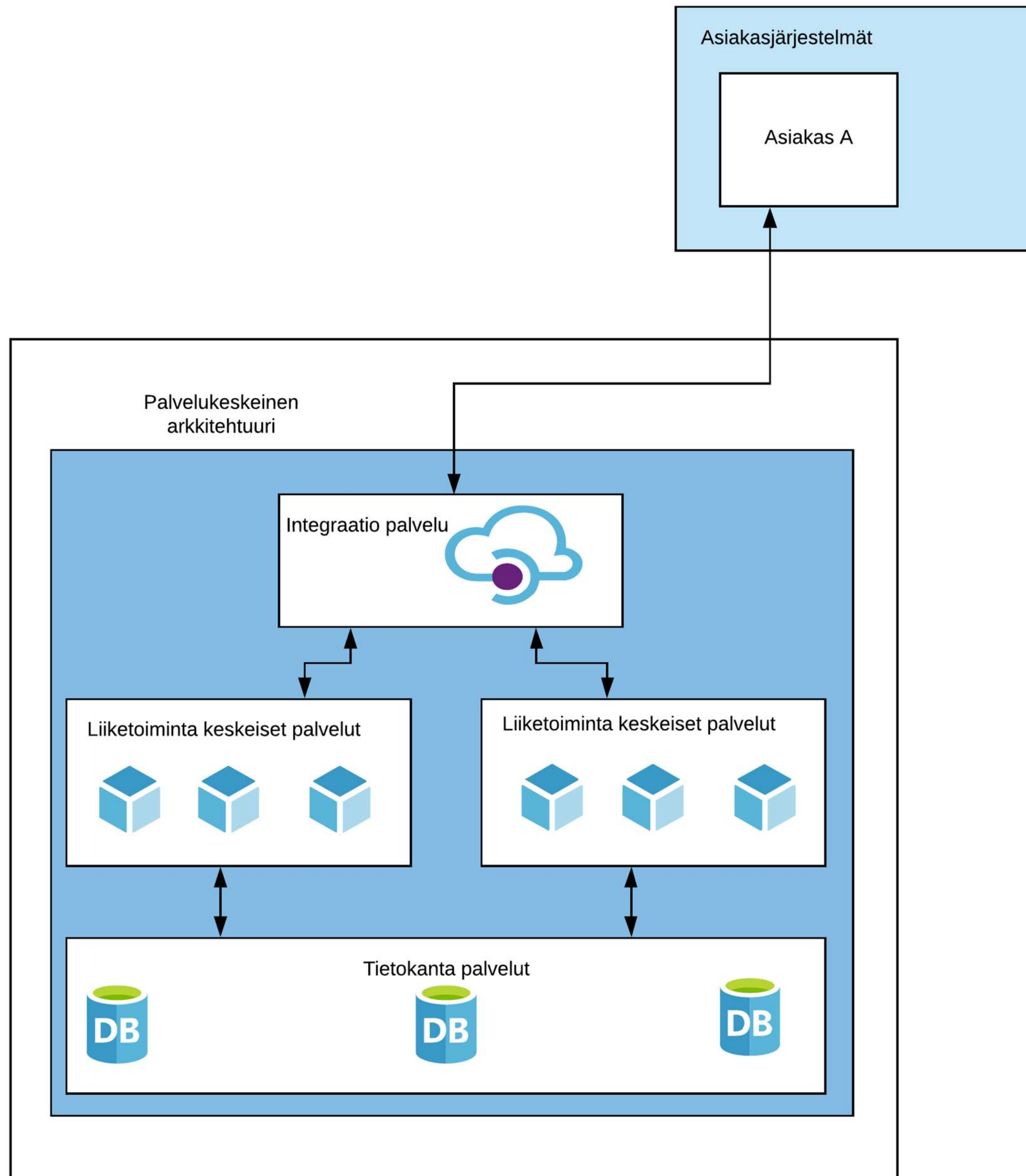
Kun sovelluksen koko toiminnallisuus on paketoitu yhteen sovelluspakettiin, on muutosten toteuttaminen useissa tiimeissä hankalampaa. Tiimit koostuvat yleensä useista eri henkilöistä ja muutosten toteuttaminen rinnakkain vaatii kommunikointia muutoksien tullessa samaan sovellusjulkaisuun. Monoliittisen järjestelmäarkkitehtuurimallin heikkouksia ovat sen laajennettavuus, uudelleenkäytettävyys, ylläpidettävyys ja kehittäminen. Monoliittinen järjestelmäarkkitehtuurimalli yhdistää sovelluksen kaikki loogiset osa-alueet yhdeksi, mikä vaikeuttaa eri osa-

alueiden uudelleenkäytettävyyden ja laajentamisen. Jos käyttöönoton jälkeen huomataan sovelluksen jollakin osa-alueella resurssien puutteita tai virheitä, täytyy koko sovellus ensin korjata ja sen jälkeen julkaista uudelleen. (Strimbei ym 2015, 14.)

3.3 Palvelukeskeinen järjestelmäarkkitehtuuri

Palvelukeskeisessä järjestelmäarkkitehtuurimallissa (SOA- Service Oriented Architecture) järjestelmän eri komponentit on löyhästi sidottu (*loose-coupled*) toisiinsa. Tämä tarkoittaa sitä, että eri komponentit kommunikoivat standardoituja rajapintoja hyödyntäen. Komponentit kommunikoivat käyttäen ennalta sovittua ja määritettyjä tietorakenteita ja kommunikointiprotokollia. Tämä varmistaa sen, että suoria riippuvuuksia eri komponenttien välille ei muodostu. Jokainen komponentti toimii itsenäisesti ja palvelee tarkkaan suunniteltua tarvetta. Löyhä sidonta mahdollistaa sen, että eri komponentit voidaan toteuttaa käyttäen eri toteutustekniikoita ja komponentit voidaan sijoittaa toisistaan erilleen. Löyhän sidonnan avulla saadaan merkittävä uudelleenkäytettävyys eri komponenttien välillä sekä mahdollisuus toteuttaa toiminnallisuus juuri siihen tarpeeseen tehokkaimmalla tavalla, kun koko järjestelmän kehitys ei ole sidottu vain yhteen teknologiaratkaisuun. Löyhä sidonta myös mahdollistaa eri komponenttien paremman hallittavuuden ja asetusten muokkaamisen. (Sacha ym. 2010.)

Kuviossa 3 on looginen kuvaus palvelukeskeisestä järjestelmäarkkitehtuurimallista.



Kuvio 3. Esimerkki palvelukeskeisestä järjestelmäarkkitehtuurista

Järjestelmän eri komponentit on jaoteltu kuviossa loogisiin kokonaisuuksiin. Alimpana kuvassa on järjestelmän käyttämät tietovarannot. Seuraava kerros sisältää liiketoimintakeskeiset palvelut eli järjestelmään toteutetun logiikan. Ylimmäisenä kerroksena on integraatiokerros, jonka tehtävä on mahdollistaa asiakasjärjestelmien välinen tiedonvaihto. Järjestelmän eri kokonaisuudet eivät ole suoraan riippuvaisia toisistaan ja kommunikoivat ennalta määritettyjä rajapintoja käyttäen.

Palvelukeskeisen järjestelmäarkkitehtuurimallin tarkoitus on mallintaa järjestelmän toiminta niin, että se maksimoi muunneltavuuden ja kustannustehokkuuden.

Tämä saavutetaan niin, että järjestelmän palvelut kuvataan ja toteutetaan käyttötarkoituksen mukaan. Palvelukeskeisen toteutus voi sisältää eri teknologioiden yhdistelmiä, tuotteita ja muita järjestelmän oleellisia osia. (Erl 2009, 37.) Palvelukeskeistä järjestelmäarkkitehtuurimallia voidaan toteuttaa erilaisten lähestymistapojen avulla. Erilaisia lähestymistapoja ovat liiketoimintakeskeinen, teknologiariippumaton, organisaatiokeskeinen ja yleiskäyttöinen. (Erl 2009, 52.)

Palvelukeskeinen järjestelmäarkkitehtuuri yleensä koostuu kolmesta eri osa-alueesta eli palveluiden tarjoajista, palveluiden käyttäjistä sekä palvelurekisteristä. Palveluiden käyttäjät käyttävät palveluiden tuottajien julkaisemia ja ylläpitämiä palveluita. Palvelurekisterissä on jokaisesta palvelusta kuvaus. Palvelurekisteri on sekä palvelun tarjoajan ja palvelun käyttäjän saatavilla.

3.3.1 Liiketoimintakeskeinen lähestymistapa

Liiketoimintakeskeisessä (*Business Driven*) suunnittelussa tulevat visiot, tavoitteet ja vaatimukset ovat keskeisessä roolissa järjestelmän suunnittelun alkuvaiheessa. Järjestelmäarkkitehtuurin suunnitteluun saadaan yleensä enemmän järjestelmäarkkitehtuurista hyötyä pidemmällä tähtäimellä. Mitä aikaisemmassa vaiheessa voidaan huomioida tulevaisuus, sitä helpommin on mahdollista tehdä muutoksia järjestelmäarkkitehtuuriin. Uudet liiketoimintanäkemykset voivat aiheuttaa järjestelmäarkkitehtuurin uudelleen määrittelyn. (Erl 2009, 53.)

3.3.2 Teknologiariippumaton lähestymistapa

Teknologiariippumattoman lähestymistavan tunnuspiirteenä on suunnitella järjestelmäarkkitehtuuri niin, ettei järjestelmäarkkitehtuuria sidota yhteen tiettyyn teknologiaan. Kun järjestelmäarkkitehtuurin suunnittelussa ohjataan käyttämään vain ennalta määritettyjä teknologioita, voi se johtaa teknologioiden muuttuessa siihen, että järjestelmään ei ole mahdollista toteuttaa uusia ominaisuuksia. Järjestelmä menee muutosten takia vanhaksi tai jopa käyttökelvottomaksi. Kutsuttavan järjestelmän toiminnallisuudet voivat olla toteutettu eri teknologioilla, kuin kutsuvan järjestelmän käyttämät teknologiat. Tämä tarkoittaa, että järjestelmä on toteutettu teknologiariippumattomasti. Teknologiariippumattomuus saadaan aikaan toteuttamalla rajapinnat ja niitä kuvaavat sopimukset. (Erl 2009, 54–58.)

3.3.3 Organisaatiokeskeinen lähestymistapa

Organisaatiokeskeisessä suunnittelussa tunnistetaan organisaation eri liiketoimintayksiköt. Jokaisen liiketoimintayksikön tarpeiseen suunnitellaan järjestelmää tukeva arkkitehtuuri. Järjestelmäarkkitehtuuri siis pyritään jaottelemaan eri liiketoimintakeskeisiin loogisiin palvelukokonaisuuksiin. Suunnitteluvaiheessa huomioidaan myös muiden liiketoimintayksikön tarpeet ja pyritään purkamaan eri yksiköiden välisiä siiloja. Organisaatiokeskeisen järjestelmäarkkitehtuurin tulee tukea eri palvelukokonaisuuksien välisiä yhteyksiä. Näin saavutetaan eri liiketoimintayksiköiden välisiä hyötyä. (Erl 2009, 58–60.)

3.3.4 Yleiskäyttöinen lähestymistapa

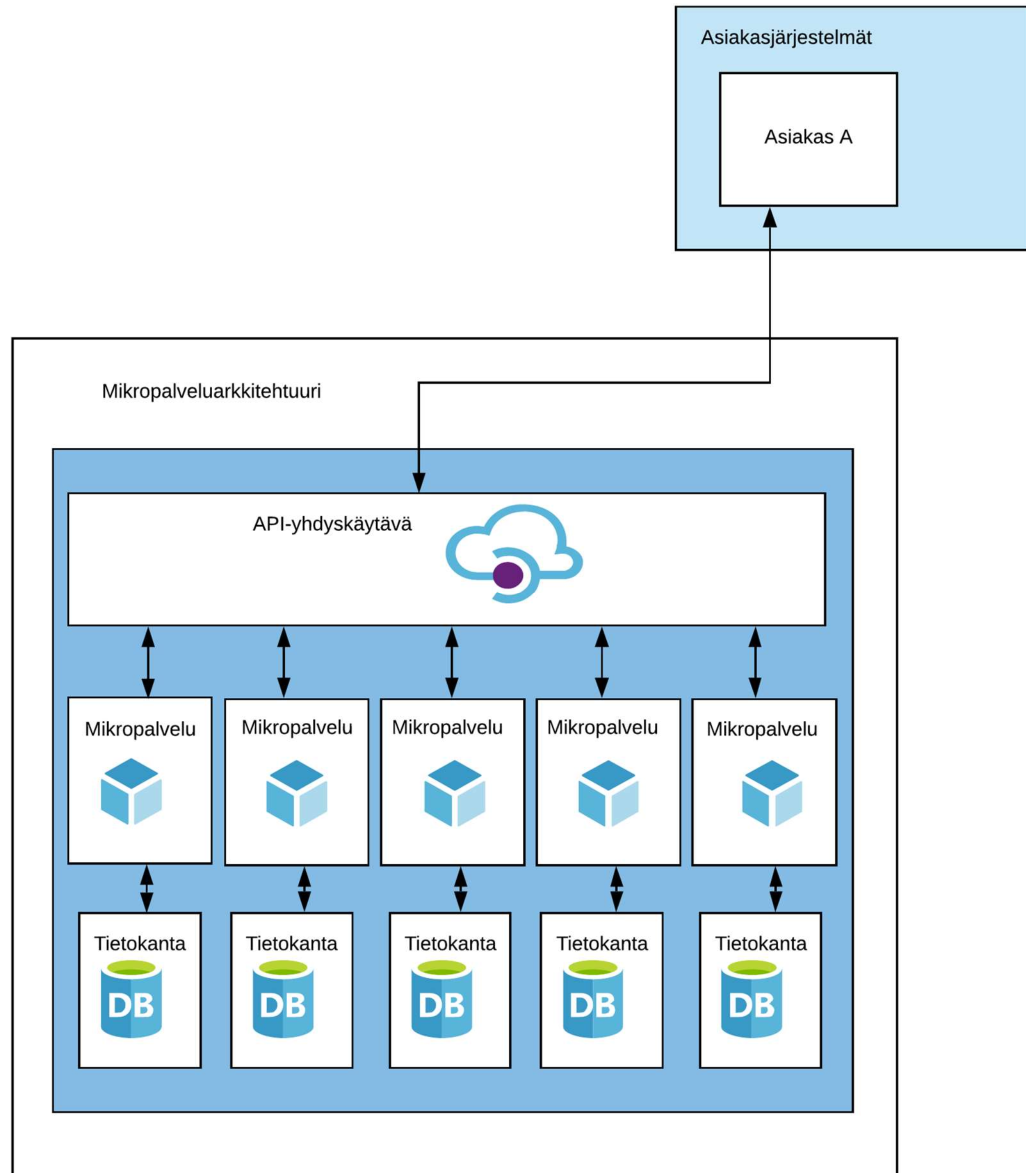
Yleiskäyttöisessä lähestymistavassa ideana on toteuttaa palveluita niin, että palveluita voidaan käyttää hyödyksi myös muissa palveluissa. Kun palvelut suunnitellaan yleiskäyttöiseksi, voidaan rakentaa useiden palveluiden niin sanottuja yhdistelmäpalveluita. Yhdistelmäpalvelut voivat sisällyttää sisäänsä yhden tai useamman toisen palvelun muodostaen niistä loogisen kokonaisuuden. Näin saadaan merkittävää hyötyä palvelukehityksen aikana. (Erl 2009, 59–50.)

3.4 Mikropalvelut

Mikropalvelut ovat pieniä, itsenäisiä palveluita, jotka on kehitetty yhtä tarkoitusta varten. Mikropalveluarkkitehtuuri pohjautuu edellä mainittuun yksittäisen mikropalvelun kuvaukseen. Mikropalveluarkkitehtuurissa suunnitellaan yksittäisiä pieniä kokonaisuuksia tiettyyn tarkoitukseen ja tarvittaessa yhdistetään niitä isommaksi kokonaisuudeksi. Pienellä tarkoitetaan tässä yhteydessä liiketoimintalogiikan kannalta pientä lohkoa, ei suoraan palvelun lähdekoodin pituutta tai järjestelmäkomponenttien määrää. Kuvaavaa mikropalveluille on se, että ne on suunniteltu ja kehitetty tekemään yksittäistä asiaa mahdollisimman hyvin, eikä niiden toiminnallisuus ole riippuvainen muista palveluista. (Newman 2015, 22.)

Kuviossa 4 kuvataan esimerkki mikropalveluarkkitehtuurin rakenteesta. Järjestelmän eri komponentit on jaoteltu toisistaan erilleen, eikä suoria riippuvuuksia mikropalveluiden välillä ole. Jokaisella mikropalvelulla on toteutettu oma

liiketoimintalähtöinen logiikka ja jokaisella mikropalvelulla on oma tietokanta käytössä.

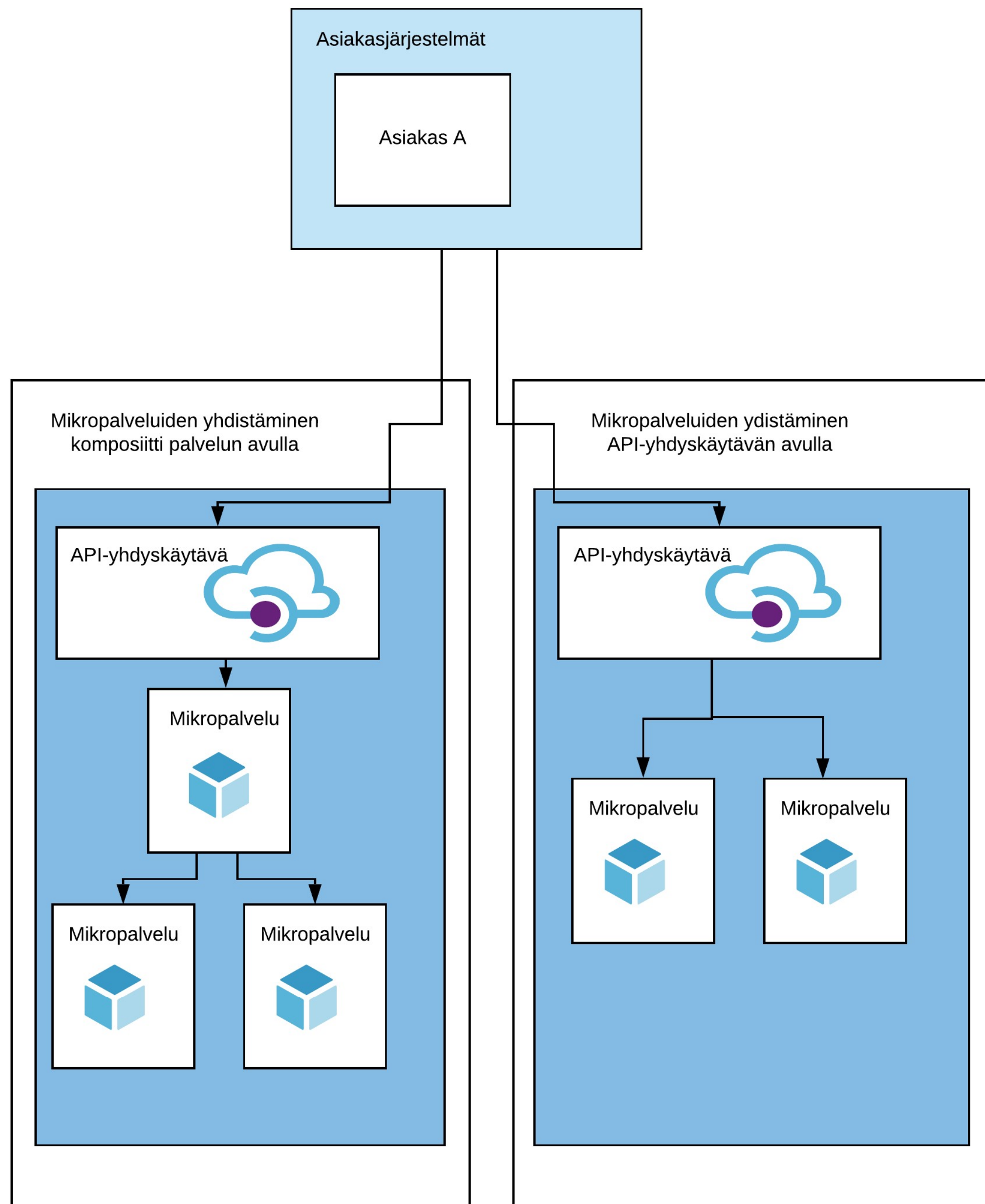


Kuvio 4. Esimerkki mikropalveluiden järjestelmäarkkitehtuurista

Mikropalveluiden päätehtävänä on täyttää yksittäinen osa liiketoimintavaatimuksesta. Isompien, loogisten, kokonaisuuksien rakentaminen mikropalveluiden avulla toteutetaan yhdistämällä yksi tai useampi mikropalvelu yhdeksi isommaksi, liiketoiminnan näkökulmasta kokonaiseksi prosessiksi. Yhdistäminen voidaan tehdä kahdella eri tavalla, joko API-yhdyskäytävän avulla tai niin sanotun komposiittipalvelun avulla. API-yhdyskäytävän avulla tehtäessä API-yhdyskäytävä kokoaa asiakasjärjestelmän yhteen kutsuun tarvittavan määrän mikropalveluita.

Komposiittipalvelun tapauksessa komposiittipalvelu huolehtii tarvittavien mikropalveluiden kutsumisen. (Subramanian & Raj 2019, 137.)

Kuviossa 5 kuvataan mikropalveluiden yhdistämisen vaihtoehdot. Kuviossa vasemmalla mikropalvelu yhdistää ja käyttää useampaa mikropalvelua muodostaen loogisen kokonaisuuden, niin sanotun komposiitin. Oikealla puolella mikropalveluiden yhdistäminen tehdään API-yhdyskäytävän avulla.



Kuvio 5. Esimerkki mikropalveluiden yhdistämisen vaihtoehdoista.

Riippumattomuus muista palveluista antaa mahdollisuuden asentaa mikropalvelu osaksi isompaa järjestelmää, tarjota sitä pilvialustan palveluna tai omana järjestelmäpalveluna. Mikropalvelut kommunikoivat muiden palveluiden kanssa tietoliikenteen avulla, näin ollen ne ovat löyhästi sidottu toisiin palveluihin. Riippumattomuuden yhtenä perusteena voi pitää sitä, että mikropalvelut tulee voida asentaa itsenäisesti. Mikropalveluiden päivittäminen ei voi olla riippuvainen muiden palveluiden päivityssyklistä. (Newman 2015, 23.)

Mikropalveluiden avulla käytettävää teknologiavalintaa voidaan vaihtaa. Mikropalveluiden avulla saadaan tehokkaasti tietoa siitä, täyttikö valittu teknologia sille asetetut vaatimukset ja saadaanko siitä hyötyä. Monoliittisessa järjestelmässä teknologian vaihtaminen tarkoittaa koko järjestelmän siirtoa, tai ison osan siirtämistä, käyttämään uutta teknologiaa. Tämä voi olla hidasta, aikaa vievää ja tuoda isoja kustannuksia. Myös uuden teknologian toiminnan varmistaminen pitää tehdä järjestelmän monessa eri järjestelmän osassa. (Newman 2015, 24.)

Järjestelmän tehokkuuden parantamisessa saadaan mikropalveluiden ominaispiirteistä todellista hyötyä. Järjestelmän tehokkuutta voidaan nostaa lisäämällä käytössä olevia resursseja. Tämä tuo yleensä lisää kustannuksia, sillä on yleistä laskuttaa prosessitehosta sen käytön mukaan. Mikropalveluiden avulla resurssien lisääminen voidaan kohdentaa järjestelmän siihen osaan, mikä katsotaan tarpeelliseksi, eikä järjestelmän kaikkia osa-alueita ja resursseja tarvitse lisätä. (Newman 2015, 24.)

Järjestelmän ylläpidon kannalta mikropalvelut tuovat hyötyjä ja tehokkuutta, sillä jokainen mikropalvelu voidaan yksittäin päivittää ja asentaa. Tämä vähentää koko järjestelmän alhaallaoloaikaa ja nopeuttaa päivitysprosessia. Myös järjestelmän vikatilanteista selviäminen on nopeampaa, kun ongelma voidaan kohdentaa tarkemmin ja kohdistaa korjaustoimenpiteet vain ongelman kohteena olevaan palveluun. (Newman 2015, 27.)

Mikropalveluiden luonne on toteuttaa pieni osa isommasta toiminnallisuudesta, joka johtaa siihen, että mikropalveluita on yleensä useita. Mikropalveluiden paljouden takia on erityisen tärkeää luoda palvelurekisteri. Palvelurekisteri pitää kirjaa, mitä mikropalveluita on olemassa sekä miten mikropalveluita voidaan

käyttää. Palvelurekisteri on eräänlainen tietokanta käytössä olevista mikropalveluista. (Subramanian & Raj 2019, 134.)

3.5 API-yhdyskäytävä

API-yhdyskäytävä on järjestelmän ohjelmallinen keskipiste, jonka kautta kaikki sisään tulevat palvelukutsut tulevat. API-yhdyskäytävä ratkaisua käytetään yleisesti mikropalveluarkkitehtuurin yhteydessä. API-yhdyskäytävä tuo mikropalveluarkkitehtuuriin monia hyödyllisiä ominaisuuksia, muun muassa se lisää järjestelmän käytettävyyttä, vikasietoisuutta, monitorointia ja sen avulla voidaan tukea useita kommunikointiprotokollia.

3.5.1 Järjestelmän käytettävyys ja tietoturva

Yksinkertaistettuna API-yhdyskäytävän voidaan ajatella olevan palveluiden välityspalvelin, joka tekee sanomalle tarkistuksia ja myös mahdollistaa palveluiden kutsumisen. API-yhdyskäytävällä on tiedossa kaikki järjestelmän palvelut ja niiden toiminnallisuudet. Nämä palvelutiedot julkaistaan API-yhdyskäytävän avulla järjestelmän käyttäjille. Tämä mahdollistaa sen, että API-yhdyskäytävä tarjoaa yhden keskitetyn ohjelmallisen pisteen järjestelmän käyttäjille, eikä järjestelmän käyttäjien tarvitse tietää jokaisen eri mikropalvelun kommunikaatioprotokollaa tai tarkempaa sijaintia. API-yhdyskäytävän tehtävänä on vastaanottaa sisään tuleva kutsu ja reitittää sen oikealle palvelulle. (Subramanian & Raj 2019, 142.)

API-yhdyskäytävällä voidaan parantaa järjestelmän tietoturvaa. API-yhdyskäytävän avulla voidaan jokainen yhteys varmentaa sisältävän oikealaisen salauksen ja palvelun käyttöoikeuden. Lisäksi API-yhdyskäytävän avulla voidaan varautua monenlaisiin palvelunestohyökkäyksiin esimerkiksi asettamalla palveluille sanomien maksimikokorajoitukset ja rajoittamalla, kuinka paljon palveluita voidaan kutsua sekunnin aikana. (Richardson 2019, 262–263.)

3.5.2 Transformaatio

Järjestelmät kommunikoivat toisien järjestelmien ja laitteiden välillä. Tämä tarkoittaa sitä, että järjestelmän pitää osata useita eri kommunikaatioprotokollia ja tarjota palveluita sekä synkronisena että asynkronisena. Laitteiden monimuotoisuudesta johtuen laitteiden välisessä kommunikaatiossa voi olla useita eri viestinvälitysformaatteja käytössä. API-yhdyskäytävä mahdollistaa eri

kommunikointiprotokollien ja viestinvälitys formaattien yhdistämisen niin, että asiakasjärjestelmän ei edes tarvitse tietää, mitä API-yhdyskäytävän takana on käytössä. API-yhdyskäytävä mahdollistaa yksinkertaisemman kommunikaation ja laajemman käyttöasteen palveluihin. (Subramanian & Raj 2019, 137.)

3.5.3 Monitorointi

API-yhdyskäytävän rooli on olla keskitetty piste järjestelmään kommunikoinnissa, toisin sanoen jokainen sanoma menee sen kautta. On siis luonnollista lisätä API-yhdyskäytävään monitorointia. API-yhdyskäytävän monitoroinnilla saadaan hyödyllistä tietoa koko järjestelmän toiminnallisuudesta. Monitoroinnista saadun tiedon analysointi antaa todenmukaista kuvaa siitä, miten järjestelmää käytetään ja miten voitaisiin tulevaisuudessa varautua käyttöasteen hetkellisiin nousuihin. Monitorointia voidaan toteuttaa monesta eri näkökulmasta: palveluiden käyttöastetta voidaan mitata, ja palveluiden turvallisuuslokia sekä palveluiden tuottamaa käyttölokia voidaan seurata. Monitoroinnin tarkoitus on varmistaa, että palvelut toimivat niin kuin ne on suunniteltu ja samalla varmistaa resurssien riittävyys järjestelmän käyttöasteeseen verrattuna. (Subramanian & Raj 2019, 138.)

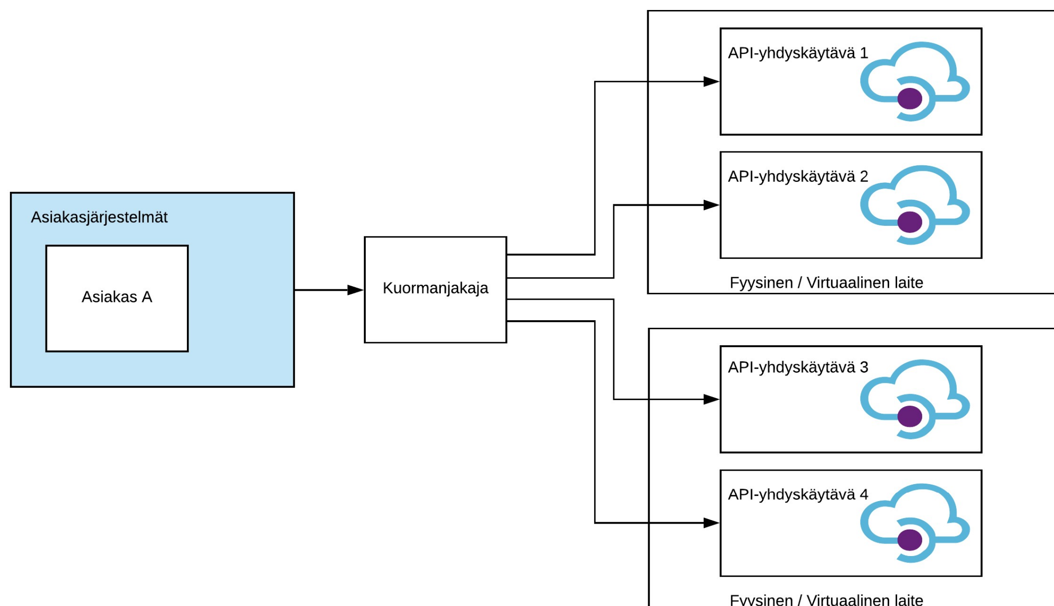
3.5.4 Vikasietoisuus

Kun järjestelmässä on yksittäinen keskitetty kommunikaatiopiste, on olemassa vaara SPOF:lle (Single Point Of Failure). Se tarkoittaa, että järjestelmän käyttöä voidaan joko tahattomasti tai tahallisesti häiritä. Tahaton häirintä yleensä johtuu siitä, että järjestelmän käytössä olevat resurssit on alimitoitettu. Järjestelmässä voi olla myös vikatilanne, mistä johtuu, ettei järjestelmän käyttäminen ole enää mahdollista. Tahallinen häirintä tarkoittaa, että jokin taho tahallisesti yrittää estää järjestelmän käytön esimerkiksi joko näkyvyyden tai rahallisen hyödyn tavoittelemiseksi. Häiriötilanteiden takia on erityisen tärkeää varautua API-yhdyskäytävän toteutuksessa vikatilanteisiin.

Korkean saatavuuden takaamiseksi järjestelmässä on useampi kuin yksi API-yhdyskäytävä yhtä aikaa aktiivisena. Sanomaliikenne API-yhdyskäytävälle tulee kuormanjakajan kautta. Kuormanjakajan tehtävä on seurata eri API-yhdyskäytävän läpi menevien sanomien määrää, vikaosuutta ja vasteaikoja. (Subramanian & Raj 2019, 138.) Seurantatulosten perusteella kuormanjakaja jakaa kuormaa ennalta määrättyjen säännöstöjen perusteella kohdennetulle API-

yhdyskäytävälle. Kuormanjakaja tekee myös omaa seurantaa API-yhdyskäytävälle ja näin ollen tietää jo ennen sisään tulevaa sanomaa, onko jokin API-yhdyskäytävä vikatilassa. Mikäli API-yhdyskäytävä on vikatilassa ei kuormanjakaja päästä sanomaa sille API-yhdyskäytävälle. Kun vikatilassa ollut API GW korjataan ja seuranta näyttää vihreää, kuormanjakaja alkaa uudelleen välittämään sanomia sille.

Kuviossa 6 kuvataan kuormanjakajan ja useamman API-yhdyskäytävän ratkaisu. Asiakasjärjestelmästä lähetetty sanoma menee kuormanjakajan kautta ja kuormanjakaja välittää sanomaa yhdelle neljästä API-yhdyskäytävästä.

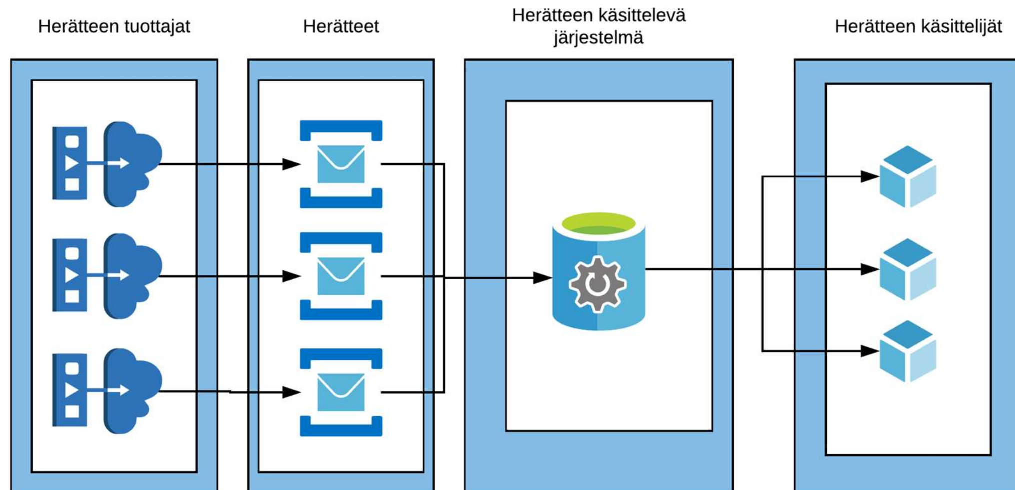


Kuvio 6. API-yhdyskäytävä ratkaisun kuormanjako.

3.6 Tapahtumakeskeinen järjestelmäarkkitehtuuri

Tapahtumakeskeisessä järjestelmäarkkitehtuurisuuntauksessa on esitelty tapahtuma (event), julkaisija (publisher) ja konteksti (context) määritystä. Tapahtuma on ajonaikainen heräte, joka sisältää tietoa operaatiosta. Herätteen tarkoitus on siirtää tietoa järjestelmän komponentilta toiselle potentiaaliselle herätteestä kiinnostuneelle komponentille. Julkaisija on se komponentti, joka julkaisee herätteen muiden komponenttien saataville. Konteksti sisältää julkaisijan asettamaa tietoa, jonka perusteella kuuntelijakomponentti voi päätellä, käsitteleekö herätettä vai ei. (Raj, Raman & Subramanian 2017, 238.)

Kuviossa 7 on esimerkki tapahtumakeskeisestä arkkitehtuurista. Kuviossa vasemmalla ovat herätteen toteuttajat, jotka muodostavat herätteen. Tuottajat välittävät herätteen erilliseen järjestelmään, josta yksi tai useampi käsittelijä käsittelee herätteen.



Kuvio 7. Esimerkki Event Driven-arkkitehtuurista

Tapahtumakeskeinen järjestelmäarkkitehtuuri eli EDA-suuntaus on soveltuvin järjestelmäarkkitehtuurivalinta, kun halutaan todellista löyhää sidontaa järjestelmän sisäisten prosessien tai järjestelmien välille. Prosessien välinen löyhä sidonta parantaa järjestelmän joustavuutta. EDA-suuntausta voidaan käyttää myös järjestelmien väliseen kommunikaatioon ja löyhään sidontaan. EDAn ja mikropalveluiden ero on se, että EDAn avulla voidaan myös järjestelmän sisäisiä prosesseja sitoa löyhästi, kun mikropalvelut keskittyvät järjestelmien väliseen löyhään sidontaan. Löyhää sidontaa tulisi käyttää aina, kun vain mahdollista järjestelmäarkkitehtuurin suunnittelussa sen helpon muokattavuuden vuoksi. (Raj ym. 2017, 240.)

EDA-suuntauksen ominaispiirteitä ovat mahdollisuus lähettää heräte monelle eri järjestelmälle yhtä aikaa sekä reaaliaikainen ja asynkroninen kommunikaatio. Asynkroninen kommunikaatio mahdollistaa sen, että julkaisijan ei tarvitse odottaa käsittelijän vastausta ennen kuin voi lähettää uuden herätteen. Käsittelijä voi määrittää olevansa kiinnostunut tietystä herätteestä tai useista eri herätteistä, herätteen kontekstissa olevien arvojen perustella. (Raj ym. 2017, 241.)

3.7 Dokumentointi

Onnistunut järjestelmäarkkitehtuuri sisältää järjestelmäarkkitehtuurin dokumentoinnin, jota pidetään yleensä suunnittelun tärkeimpänä vaiheena. Dokumentointi kuvaa järjestelmän toiminnallisuuden, mutta toimii myös ohjeena järjestelmän kehittäjille (Ingeno 2018, 304). Dokumentoinnista pitää käydä ilmi, mitä päätöksiä suunnittelun aikana on päädytty tekemään. Järjestelmäarkkitehtuuri kuvataan yleensä arkkitehtuurinäkymien avulla. Arkkitehtuurinäkymiä on yleensä useampia, sillä järjestelmän kokonaisarkkitehtuuri on yleensä sen verran monimutkainen ja sisältää monia eri komponentteja, ettei sitä voida yhdellä kuvalla täysin selkeästi kuvata. (Ingeno 2018, 138.)

Arkkitehtuurinäkymien muodostaminen alkaa suunnittelemalla hahmotelmia järjestelmäarkkitehtuurista. Hahmotelmien ei tarvitse noudattaa mitään ennalta määritettyä muotoa, mutta hahmotelmien tulisi olla selkeitä ja kuvata niiden ominaistarkoitusta. Hahmotelmien tekeminen auttaa arkkitehtuurinäkymien tekemistä suunnittelun myöhemmässä vaiheessa. Kun hahmotelmat ovat valmiina, alkaa tarkemman suunnittelun tekeminen. Tarkemmassa suunnittelussa olisi hyvä kuvata päätökset ja perustelut niiden takana. Tekemättä jätetyt päätökset voidaan myös dokumentoida sekä lisäksi niiden syyt. Päätösten dokumentointi takaa sen, että niiden perustelut voidaan käydä myöhemmin tarkastamassa tai suunnittelutyöhön kuulumaton henkilö voi tarkistaa suunnitteluvaiheen päätökset. (Ingeno 2018, 139,149.)

Dokumentoinnin avulla voidaan siirtää tietoa järjestelmän toiminnallisuudesta. Esimerkiksi uusien kehittäjien perehdyttäminen kyseiseen järjestelmään voidaan tehdä hyvän järjestelmäarkkitehtuuridokumentaation avulla. Tämän takia onkin tärkeää, että jokaisen järjestelmän muutoksen jälkeen tarkastetaan, pitääkö päivittää dokumentointia. Hyvän dokumentoinnin avulla voidaan myös uusien järjestelmien kehityksessä hyödyntää aikaisempia toimivaksi todettuja ratkaisuja. Ratkaisujen uudelleen hyödyntäminen parantaa yrityksen tehokkuutta ja antaa varmuutta uuden järjestelmän kehityksessä, sillä mahdolliset ongelmakohdat on jo kerran ratkaistu ja todennettu. (Ingeno 2018, 304.)

Toimiva järjestelmäarkkitehtuuri on käytettävissä vain, kun se voidaan kommunikoida muille osapuolille. Jos järjestelmäarkkitehtuurin viestimistä muille osapuolille ei voida tehdä tehokkaasti, on vaarana, että järjestelmän kehittäjät eivät voi

tehdä muutoksia tai tekevät ne alkuperäisen järjestelmäarkkitehtuurin vastaisesti. (Ingeno 2018, 304.)

4 IoT- LAITTEIDEN KOMMUNIKOINTIPROTOKOLLAT

4.1 Lyhyen kantaman protokollat

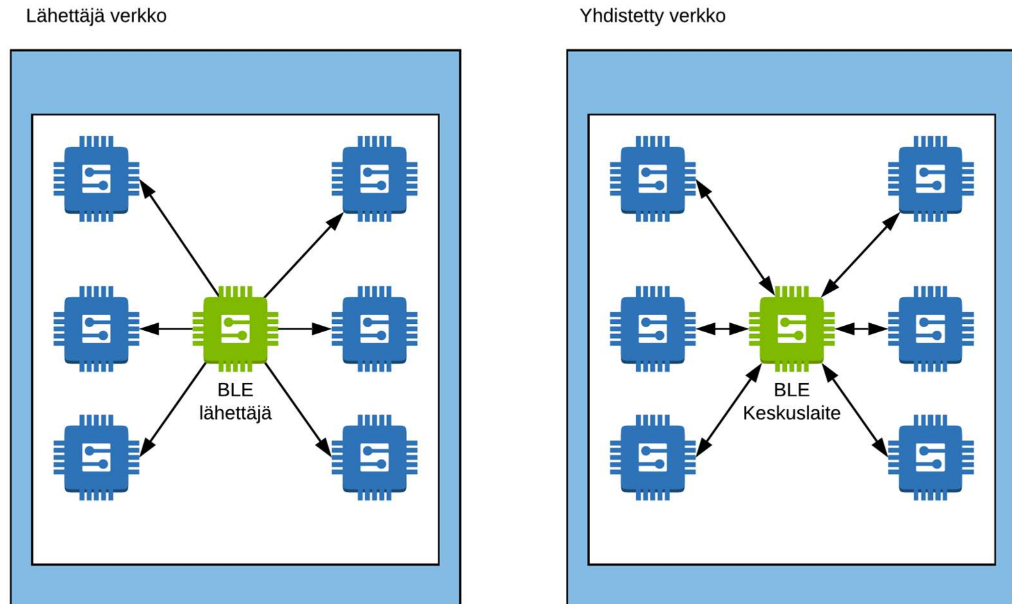
Lyhyen kantaman protokollalla tarkoitetaan eri kommunikointiprotokollia, joiden suunnittelun lähtökohtana on ollut energiatehokkuus. Yleensä IoT-laitteet toimivat itsenäisesti, erillisen akun varassa. Energiatehokkuuden maksimoimiseksi lähetystehoa sekä tiedonsiirron kapasiteettia on rajoitettu. Lyhyen kantaman protokollien teoreettiset maksimilähetysetäisyydet ovat yleensä muutamia kymmeniä metrejä, korkeintaan satoja metrejä. Protokollat suunnitellaan mahdollistamaan vain yhden suuntainen kommunikaatio. Tämä soveltuu erityisen hyvin IoT-laitteille, jotka vain lähettävät tilatietoa, eivätkä vastaanota tietoa muilta laitteilta.

4.1.1 Bluetooth LE

Bluetooth LE (*Bluetooth Low Energy*) tai toiselta nimeltä Bluetooth Smart on energiatehokkaille laitteille tarkoitettu standardi. Bluetooth LE-standardin suunnittelun lähtökohtana on ollut suunnitella radiostandardi, joka käyttäisi mahdollisimman vähän virtaa. Standardin tulisi erityisesti olla optimoitu olemaan energiatehokas, yksinkertainen, omata alhainen hinta ja tiedonsiirron rajallisuus. Bluetooth LE on 2010-luvulla levinnyt laajasti käyttöön, johtuen siitä, että se on monessa älylaitteessa valmiina. (Townsend, Cufí, Akiba & Davidson 2009, 1.)

Teoreettinen tiedonsiirto nopeus Bluetooth LE-verkossa on 1 Mbps. Verkon maksimininopeuden kuitenkin määrittää se, että kuinka paljon verkon laitteilla on olemassa resursseja. Laitteen prosessoriteho, radiotaajuuksien rajallisuus ja kahdensuuntainen liikenne rajoittaa pääsemästä tiedonsiirrossa teoreettiseen maksiminopeuteen. Bluetooth LE on lyhyen kantaman protokolla, joten laitteiden välinen etäisyys on rajoitettu. Bluetooth LE-protokolla määrittää maksimietäisyyden 30 metriin. Käytännössä maksimietäisyys on lähempänä muutamia metrejä, riippuen laitteen antennista, toimintaympäristöstä ja virransäästöasetuksista. (Townsend ym. 2009, 7–8.)

Bluetooth LE-laitteet voivat kommunikoida kahdella eri tavalla: lähettämällä (broadcasting) ja yhdistettynä (connections). Kuviossa 8 on kuvattu Bluetooth LE-laitteiden kommunikointitavat. Kuviossa vasemmalla on lähettäjä verkko, eli vihreä Bluetooth LE-laite vain välittää tietoa. Kuviossa oikealla on yhdistetty verkko, eli vihreä Bluetooth LE-laite sekä lähettää että vastaanottaa tietoa.



Kuvio 8. Esimerkki Bluetooth LE-verkkojen periaatteesta

Lähettäjäverkossa laitteet voivat lähettää tietoa ilman, että verkossa olevia vastaanottavia laitteita etukäteen tutkitaan verkosta. Tällä mekanismilla voidaan lähettää tietoa ainoastaan yhteen suuntaan. Lähettävä laite lähettää tietoa tietämättä, onko verkossa laitteita vastaanottamassa. Jokainen laite verkossa voi vastaanottaa kyseisen tiedon ja käsitellä sen. Lähettäjäverkko on erityisen sopiva tapauksiin, jossa laitteet vain lähettävät tilaa itsestään. Lähettäjäverkon haittapuolena on turvallisuus. Koska lähettävän ja vastaanottavan laitteen välille ei muodosteta ennalta suoraa yhteyttä, ei myöskään voida taata tiedon siirtoa vain kyseiselle laitteelle. (Townsend ym. 2009, 9–10.) Lähettäjä verkossa lähetettävän tiedon onkin syytä olla sellaista, että siitä ei ole tietoturvauhkaa, vaikka se välittyisi muille laitteille kuin oli alun perin tarkoitettu.

Kun on tarvetta siirtää tietoa Bluetooth LE-laitteiden välillä kahdensuuntaisesti, tulee laitteiden välillä olla muodostettu yhteys. Muodostettua yhteyttä voi käyttää vain kaksi laitetta, joten tiedon vaihto on turvallisempana. (Townsend ym. 2009, 10.) Yhdistetyssä verkossa toimii kahdenlaisia laitteita: keskuslaite (*central*) ja orja (*slave*). **Keskuslaite** toimii verkossa aktiivisena laitteena ja tarkistaa, onko verkossa laitteita, jotka ovat lähettämässä dataa. Jos laitteita löytyy, se muodostaa näiden välille yhteyden ja välittää tiedon eteenpäin. **Orja** on verkossa oleva

laite, joka lähettää tiedon. Se odottaa, että keskuslaite muodostaa yhteyden ja sen jälkeen lähettää tiedon keskuslaitteelle. (Townsend ym. 2009, 10.)

Keskuslaite aloittaa yhteyden muodostamisen orjalaitteiseen, kun se havaitsee orjalaitteen lähettämän niin sanotun mainosviestin. Mainosviestien tarkoitus on kertoa keskuslaitteelle, että orjalaitteella on tietoa lähetettävänä. Tällä tavoin keskuslaite muodostaa yhteyden vain niihin verkon laitteisiin, joilla on jotain lähetettävää. Kun tieto on saatu välitettyä, orjalaite lopettaa mainospakettien lähettämisen verkkoon. (Townsend ym. 2009, 11.)

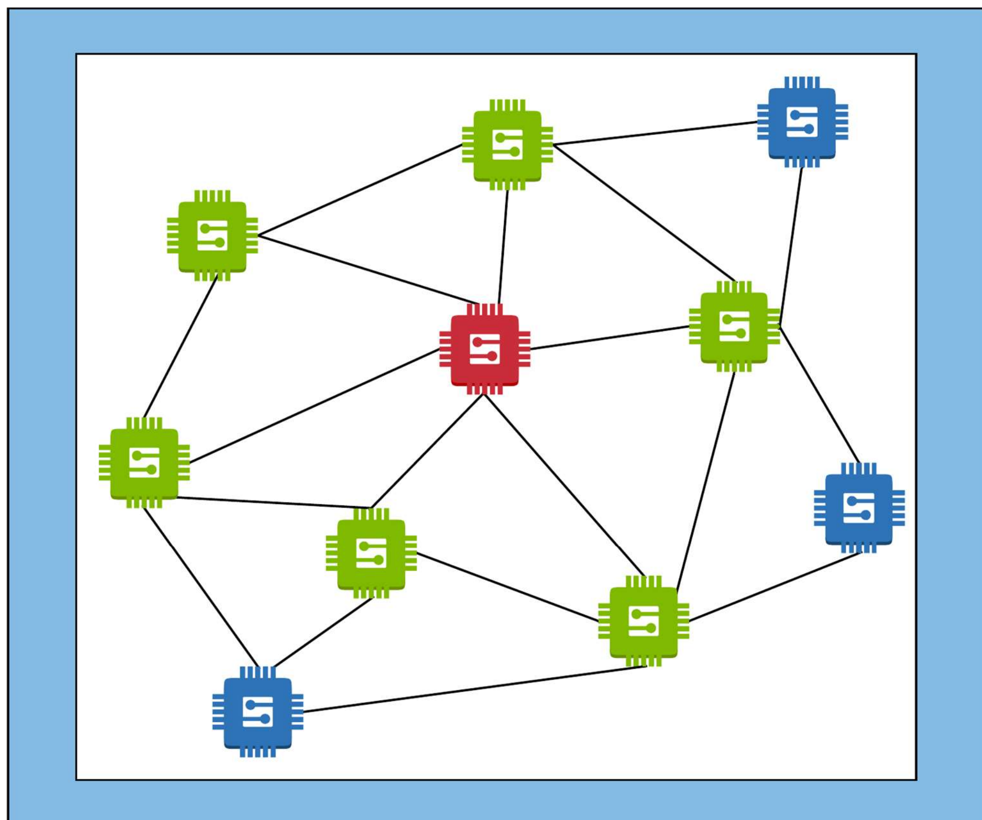
4.1.2 Z-Wave

Z-Wave on suunniteltu energiatehokkaaksi langattomaksi verkoksi. Se on alun perin Zensys-yhtiön suunnittelema. Z-Wave-protokolla voi toimia radioverkossa useammalla eri taajuudella. Z-Wave luo automaattisesti niin sanotun mesh-verkon. (Pei, Deng, Yang & Cheng. 2008, 2-3.) Mesh-verkossa jokainen laitteen lähettämä signaali välitetään eteenpäin jokaisen laitteen toimesta. Mesh-verkon avulla saadaan signaali välitettyä pidemmälle ja verkon toimintavarmuutta paremmaksi. Mesh-verkko mahdollistaa sen, että signaalin vastaanottajan ei tarvitse olla signaalin lähettäjän kanssa lähetysetäisyydellä. Tällä tavoin parannetaan energiatehokkuutta, kun laitteiden lähetystehoja ei tarvitse nostaa.

Z-Wave-protokolla on suunnittelu käyttämään alhaista tiedonsiirtonopeutta ja lyhyen kantomatkan lähetystehoa. Tiedonsiirtonopeus voi olla maksimissaan 100 kbps (*kilobittiä per sekuntti*) ja kahden laitteen välinen etäisyys enintään 30 metriä. Lyhyen kantomatkan ja alhaisen tiedonsiirtonopeuden johdosta protokolla soveltuu erittäin hyvin IoT-laitteille, koska näin säästetään laitteen virrankulutusta. (Al-Sarawi, Anbar, Alieyan & Alzubaidi. 2017, 687). Yhdessä Z-Wave-verkossa voi operoida maksimissaan 232 laitetta. Laitteita on olemassa kahdenlaisia: kontrollereita (*controller*) ja orjia (*slave*) (Unwala & Lu 2017, 356). Orjalaite on, yleensä halvempi laite, joka voi ainoastaan lähettää ja vastata viestejä verkkoon. Kontrolleri toimii orjalaitteiden ohjaajana ja voi lähettää ohjauskomentoja. Ohjauskomennoilla voidaan ohjata orjalaitteiden toimintaa tai välittää herätteitä verkon sisällä.

Kuviossa 9 on esimerkki Mesh-verkon toimintaperiaatteesta. Punainen laite on MESH-verkon kontrolleri. Vihreällä ja sinisellä värillä on kuvattu verkon

orjalaitteet. Vihreät laitteet ovat suoraan yhteydessä verkon kontrolleriin sekä verkon muihin laitteisiin. Siniset laitteet sijaitsevat verkon ulkolaidalla eivätkä ole suoraan yhteydessä verkon kontrollerin kanssa. Siniset laitteet keskustelevat kontrollerin kanssa vihreiden laitteiden välityksellä. Vihreät laitteet toimivat välittävässä tilassa, eli ne välittävät yhteydessä olevien laitteiden viestit eteenpäin kontrollerille. Kuten kuvasta huomaa, jokainen laite kommunikoi useamman laitteen välillä. Tämä lisää Mesh-verkon luotettavuutta, sillä yhden laitteen vikaantuminen ei estä viestien välittämistä kontrollerille.



Kuvio 9. Esimerkki MESH-verkon periaatteesta

4.1.3 ZigBee

ZigBee on lyhyen kantomatkan kommunikaatioprotokolla langattomiin verkkoihin. ZigBee-protokollaa käyttävät laitteet voivat operoida kolmella eri radiotaajuudella: 868 MHz, 915 MHz tai 2.4 GHz. ZigBee-verkon maksimitiedonsiirtonopeus on 250 kbps. ZigBee-protokolla on erityisesti suunniteltu laitteisiin, joissa alhainen hinta ja rajallinen tiedonsiirtonopeus sekä energiatehokkuus ovat tärkeimpiä ominaisuuksia. ZigBee-laitteet ovat luonteeltaan vähäisen aktiivisuuden laitteita.

ZigBee-laitteet voivat olla virransäästötilassa lähettäen tietoa epäsäännöllisin ajanjaksoin ja operoida samalla virtalähteellä jopa vuoden yhtäjaksoisesti. Tästä syystä Zibgee-protokollaa käyttäviä laitteita ovat erilaiset IoT-laitteet, esimerkiksi ovisensorit ja erilaiset kotikäyttöön tarkoitetut terveyteen liittyvät monitorit. (Farahani 2011, 2.)

Zibgee on luokiteltu IEEE 802.11b-standardiin, joka on yleinen luokitus WLAN-verkoille. Samaan stardardiin kuuluu myös 2.4Ghz-taajuudella toimiva Bluetooth LE. IEEE 802.11b standardin maksimitiedonsiirto nopeus on 11 Mbps (*megabittiä per sekuntti*) ja sisätiloissa verkon laajuus on tyypillisesti 30 ja 100 metrin välissä. Huolimatta standardin mahdollistamasta suuremmasta tiedonsiirtonopeudesta ZigBeen protokollaan on valittu pienempi tiedonsiirtonopeus ja näin ollen saavutettu parempi energiatehokkuus. (Farahani 2011, 3.)

4.2 Pitkän kantaman laiteläheiset protokollat

Pitkän kantaman ja energiatehokkaan teknologian yleisenä tunnusterminä käytetään LPWA:ta (*low-power, wide-area*). LPWA pitää sisällään useita eri teknologioita. Yhtenäistä näille teknologioille on, että niiden suunnittelun lähtökohtana on ollut kehittää pitkän kantaman ja erityisesti virran kulutuksen minimoimiseen tarkoitettuja laitteita. Edellä mainitun suunnittelulähtökohdan johdosta eri LPWA-teknologiat ovatkin erityisen sopivia IoT-laitteille. Tämänlaisia teknologia protokollia ovat LoRaWan, Sigfox ja NB-IoT. (Sinha, Wei & Hwang, 2017.)

4.2.1 LoRaWan

LoRaWan on maailmanlaajuinen avoin standardi, jota hallinnoi LoRa-järjestö. LoRaWan on suunniteltu tukemaan pitkän kantomatkan kahdensuuntaista tiedonsiirtoa energiatehokkaisiin laitteisiin. Tiedonsiirtonopeus LoRaWan-verkossa vaihtelee 300 bps ja 50 kbps välillä. Yhteen LoRaWan tukiasemaan voi olla maksimissaan yhteydessä 50 000 eri laitetta ja laitteen maksimietäisyys voi olla 20 kilometriä. (Mekki, Bajic, Chaxel & Meyer 2018, 416.)

LoRaWan-verkossa jokainen laitteen lähettämä viesti välitetään usean tukiaseman kautta. Käytännössä viesti välitetään jokaisen tukiaseman kautta, joka on laitteen kantaman sisällä. Tällä tavalla voidaan parantaa LoRaWan-verkon viestien välityksen toimintavarmuutta. Toimintavarmuuden parantaminen kuitenkin vaatii sen, että tukiasemia pitää olla useita saman alueen sisällä. Viestin

vastaanottajan tehtävänä on suodattaa jo aikaisemmin vastaanotetut samat viestit. Kun vastaanottava laite on saanut varmistettua viestin eheyden, se kuittaa sanoman vastaanotetuksi teknisellä kuittauksella, joka välitetään viestin lähettäjälle. Tällä tavoin viestin lähettäjä on tietoinen siitä, onnistuiko viestin lähetys. (Mekki ym. 2018, 414.)

4.2.2 Sigfox

Sigfox on maailmanlaajuinen patentoitu standardi. Sigfox-laitteet kytkeytyvät tukiasemaan 100 Hz taajuudella ja maksimissaan 100 bps tiedonsiirtonopeudella. Sigfox on suunniteltu käyttämään tehokkaasti tiedonsiirtokaistaa muun muassa minimoimalla tiedonsiirron aikana syntyviä häiriöitä. Varhaiset versiot Sigfox standardista mahdollistivat vain yhdensuuntaisen tiedonsiirron, mutta nykyään Sigfox mahdollistaa kahdensuuntaisen tiedonsiirron. Maksimi viestinvälitys tiheys on rajoitettu 140 viestiin päivässä ja yksittäisen viestin koko on rajoitettu 12 tavuun. Yhteen Sigfox-tukiasemaan voi olla maksimissaan yhteydessä 50 000 eri laitetta ja laitteen maksimietäisyys voi olla yli 40 kilometriä. (Mekki ym. 2018, 414.)

Sigfox-standardiin ei ole määritetty erillistä varmistusmekanismia siihen, että viestin lähettäjä tietäisi, onko viesti mennyt perille vai ei. Tämän takia jokainen Sigfox-laite lähettää saman viestin kolmeen kertaan jokaiseen määritettyyn taajuusalueeseen. Tämänlainen ratkaisu takaa sen, että yksittäinen Sigfox-laite toimii jokaisen eri geologisen alueelle tarkoitetun tukiaseman kanssa, joka vähentää yksittäisen laitteen tuotantokustannuksia, kun tukiasemalaitteita ei tarvitse kustomoida alueellisesti. (Mekki ym. 2018, 414.)

4.2.3 NB-IoT

NB-IoT on lyhytaaltoteknologia, joka voi toimia samassa verkossa perinteisten matkapuhelintietoliikenneverkkojen LTE (*Long Term Evolution*) ja GSM (*Global Systems for Mobile Communications*) kanssa. LTE-verkossa toimiessa NB-IoT noudattaa samoja periaatteita kuin LTE-verkon teknologissa on määritetty. NB-IoT hyödyntää LTE-standardista vain niitä ominaisuuksia, jotka ovat oleellisia IoT-laitteille. NB-IoT-laitteet muodostavat niin sanotun solun tukiaseman ja laitteiden välillä. Yhteen NB-IoT-soluun voi kiinnittäytyä enintään 100 000 laitetta ja sitä voidaan laajentaa tukiasemien avulla. Laajennettavuus onkin yksi NB-IoT

hyödyistä verrattuna muihin vastaaviin verkkoihin. NB-IoT-verkossa toimivat laitteet kommunikoivat synkronisesti. Synkronisen tiedonvaihdon hyvä puoli on se, että se antaa lähettävälle laitteelle parhaimman varmuuden ja tiedon viestin perille menosta kohteeseen. Heikkouksena synkronisessa tiedonvaihdossa on se, että se kuluttaa enemmän virtaa. NB-IoT-laitteen maksimietäisyys toisesta laitteesta voi olla 10 kilometriä ja maksimiviestin koko NB-IoT-verkossa on 1600 tavua. (Mekki ym. 2018, 414.)

4.3 TCP/IP-standardi

TCP/IP on kokoelma eri protokollia, joilla mahdollistetaan eri laitteiden välinen kommunikointi. TCP/IP toteutus on tuettuna 2010-luvulla melkeinpä jokaisessa laitteessa ja käyttöjärjestelmässä, joilla on tarkoitus kommunikoida toisten järjestelmien tai internetin kanssa. Tämän takia TCP/IP on yleisin käytössä oleva kommunikointiprotokollan yhdistelmä, eli pino, joka on ollut käytössä 1980-luvulta lähtien. TCP/IP:n hyviä puolia ovat sen yhteensopivuus, joustavuus ja reititettävyys. TCP/IP ei ole sidottu tiettyyn käyttöjärjestelmään tai laitteeseen, jolloin sitä voidaan käyttää lähes jokaisessa laitteessa. TCP/IP on suunniteltu erittäin joustavaksi, tarkoittaen sitä, että IP osoitteita voidaan uudelleen asettaa eri laitteille. Tiedonsiirron yksi tärkeimmistä ominaisuuksista on sen siirrettävyys eri laitteiden välillä. Tietoa tulisi voida siirtää useamman laitteen välillä ilman ongelmia. Esimerkiksi lähetettäessä tietoa tiedonsiirtopaketti kulkee useamman eri laitteen kautta. Laitteiden välissä voi olla useita eri verkkokytkimiä, palomuureja ja reitittimiä. TCP/IP:ssä ei ole rajoitettu reitityksen määrää, joten se sopii erityisen hyvin tämänkaltaiseen kommunikointiin. (Blank 2004, 2–3.)

TCP-yhteys sisältää kolme eri vaihetta, jotka ovat yhteyden muodostaminen, tiedonsiirto ja yhteyden päättäminen. TCP on yhteystietoinen protokolla, joka tarkoittaa, että tiedon lähettäjä ja tiedon vastaanottaja ovat tietoisia toisistaan. Ennen varsinaista tiedon lähetystä lähettäjä ja vastaanottaja käyvät yhteyden muodostamisesta neuvottelun, niin sanotun kolmitiekättelyn. Neuvottelun aikana on tarkoitus vaihtaa teknistä tietoa lähettäjän ja vastaanottajan välillä. Neuvottelun aikana ei siis siirretä varsinaista tietoa. Yhteys muodostetaan teknisten tietojen vaihtamisen ja tarkistamisen jälkeen. (Blank 2004, 47.)

Kun yhteys on muodostettu, alkaa tiedonsiirto. Yksittäisen tiedon lähettäminen voi koostua useammasta tiedonsiirtopaketista, riippuen lähetettävän tiedonsiirron

määrästä. TCP jakaa tiedonsiirtopaketin pienemmiksi paketeiksi tarpeen mukaan ja vastaanottaja kokoaa yksittäisistä paketeista tiedon takaisin käsiteltävään muotoon. Jokainen lähetettävä tiedonsiirtopaketti jakaa tietoa myös yhteydestä, esimerkiksi tiedonsiirtopaketin järjestysnumeron ja osoitetietoja. Tiedonsiirtopaketin lähettämisen jälkeen lähettäjä odottaa saavansa teknisen kuittauksen paketin perille menosta. Jos kuittausta ei tule, lähettää lähettäjä saman tiedonsiirtopaketin uudelleen. Näin varmistetaan, ettei tiedonsiirtopaketteja jää häiriötilanteesta johtuen lähettämättä ja alkuperäinen tieto saadaan välitettyä ehjänä vastaanottajalle. (Blank 2004, 49.)

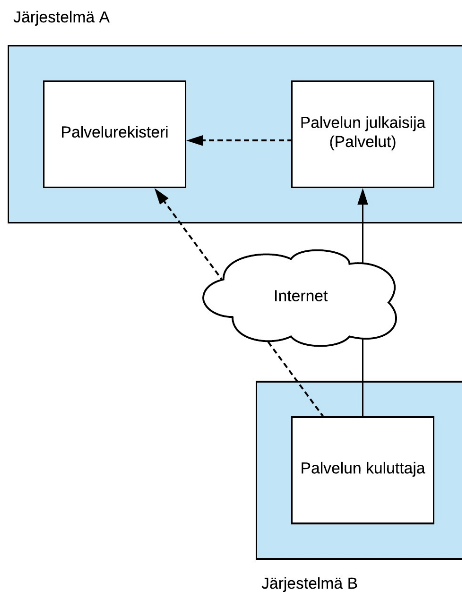
Tiedonsiirron jälkeen viimeinen vaihe on sulkea muodostettu TCP-yhteys eli päättää yhteys. TCP-yhteyden voi päättää sekä lähettäjä että vastaanottoja. Yleensä kuitenkin lähettäjä sulkee yhteyden, kun se on saanut viimeiselle tiedonsiirtopaketin lähetykselle teknisen kuittauksen. TCP-yhteyden sulkeminen tapahtuu lähettämällä tekninen FIN-kuittaus vastaanottajalle, jonka jälkeen vastaanottaja vastaa lähettäjälle vielä teknisen FIN ACK-kuittauksen ja yhteys on suljettu. Kun on tarve lähettää uudelleen dataa, uusi TCP-yhteys muodostetaan. (Goralski 2008, 291.)

4.4 Web-Service

Web Servicellä tarkoitetaan yleisesti palvelua, jota voidaan kutsua eri järjestelmistä ja eri laitteista. Web Service tarjotaan yleensä http-protokollaa hyödyntäen. Web Service-palvelut voidaan karkeasti jakaa kahteen eri ryhmään: SOAP-pohjaisiin ja REST-palveluihin. (Kalin 2009, 1.)

Web palvelukokonaisuuteen kuuluu kolme erilaista roolia, joita ovat palvelun julkaisija, palvelun kuluttaja ja palvelurekisteri. Kuviossa 10 on kuvattu esimerkki Web-palveluiden rooleista ja niiden riippuvuuksista. Kuva havainnollistaa eri roolien välistä kommunikointia. Palvelun julkaisija julkaisee kuvauksen käytettävissä olevista palveluista palvelurekisteriin ja palvelut palveluiden kuluttajien saataville. Palveluiden kuluttaja voi tarkistaa palvelurekisteristä, mitä palveluita on saatavilla ja kuvausten perusteella kutsua palveluita. Valinta eri tekniikan välillä tapahtuu aina järjestelmän vaatimusten mukaan. Kun kyseessä on yksinkertainen viestien lähettäminen ja vastaanottaminen, yleensä käytetään REST-arkkitehtuurityyliä. Kun on kyseessä useiden järjestelmien välinen tiedonvaihto tai välitettävä

sanoma pitää salata, päädytään käyttämään yleensä SOAP-tyyliä. (Balani & Hathi 2009, 35.)



Kuvio 10. Esimerkki Web-palveluiden rooleista

4.4.1 SOAP

SOAP (*Simple Object Access Protocol*) on standardeihin perustuva web-palveluiden kommunikointiprotokolla. SOAP-protokollan avulla järjestelmät voivat välittää tietoa keskenään XML-muodossa. SOAP-protokolla ei ole sidottu tiettyyn tiedonsiirtoprotokollaan, mutta yleisin tapa siirtää SOAP-viestejä on http-yhteyden avulla. On olemassa kahdenlaista kommunikointimuotoa: Document ja RPC. Document-muodossa XML-sanoma välitetään sanoman sisältönä. Sanoman muoto on määritetty erillisellä XML-skeematiedostolla ja kommunikoivien järjestelmien on noudatettava skeeman määrityksiä. (Balani & Hathi 2009, 35.) RPC (Remote Procedure Call) muodossa määritetään XML-sanoman avulla, mitä järjestelmän toiminnallisuutta asiakasjärjestelmä kutsuu. Palvelun tarjoaja julkaisee käytettävät palvelut WSDL-kuvaustiedoston kautta. WSDL-tiedosto voidaan tarjota joko palvelurekisterin kautta tai tiedostomuodossa. WSDL-tiedosto pitää sisällään teknisen kuvauksen siitä, miten järjestelmän eri palveluita voidaan kutsua ja missä osoitteessa niitä tarjotaan. (Balani & Hathi 2009, 34.)

4.4.2 REST

REST:n (*REpresentional State Transfer*) tarkoitus on käyttää http-protokollaa tiedonsiirtoon. REST on arkkitehtuurinen tyyli ja kuvaa ohjenuoria hajautettujen järjestelmien väliseen kommunikaatioon käyttäen olemassa olevia käytänteitä ja protokollia. REST nojaa pitkälti http-protokollaan ja käyttää laajasti hyödyksi http-metodeja. REST:ssä jokainen komponentti voidaan käsittää resurssina. Resursseihin päästään käsiksi käyttäen http-metodeja ja määrittämällä resurssin sijainti. (Subramanian & Raj 2019, 17-18.)

REST-applikaatiot ovat tyypillisesti tilattomia. Tilattomuus tarkoittaa sitä, että sisään tulevassa kutsussa pitää olla riittävä määrä tietoa, jotta REST-sovellus ymmärtää sisään tulevan kutsun sekä pystyy toimimaan sen perusteella. Tilattomuuden seurauksena REST-ohjelmat ovat yleensä järjestelmäarkkitehtuurisesti selkeitä, niitä on vaivatonta skaalata ja ne ovat toimintavarmoja. (Subramanian & Raj 2019, 21.)

4.5 MQTT

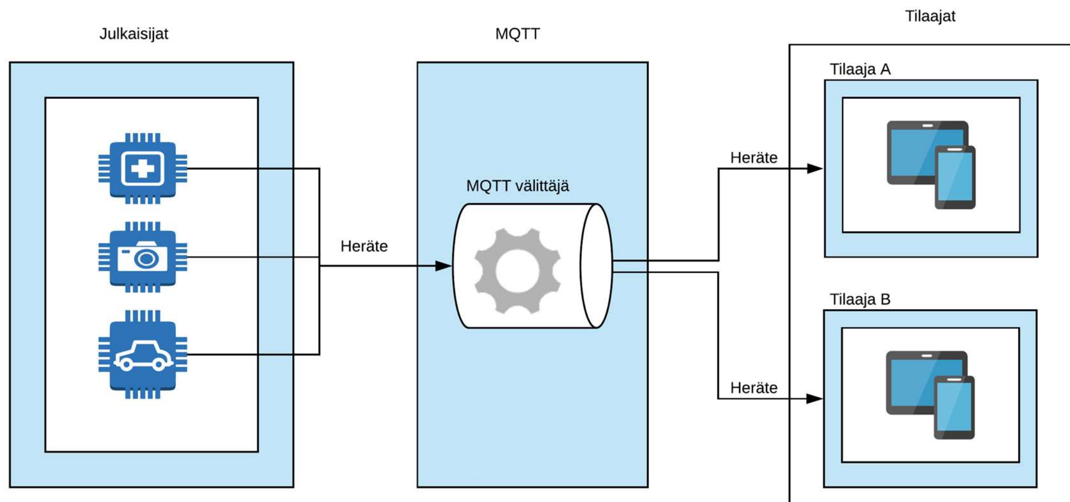
MQTT (*Message Queuing Telemetry Transport*) on alkuperin IBM:n suunnittelema ja toteuttama kommunikointiprotokolla. MQTT-protokollan suunnittelun lähtökohtana on ollut olla avoin, yksinkertainen, kevyt ja vaivaton kehittää. MQTT on asynkroninen julkaisija/tilaajaprotokolla ja toimii TCP-pinon päällä. Erityisen hyvin MQTT sopii IoT-laitteille julkaisija/tilaaja -toiminnallisuuden takia. (Karagiannis, Chatzimisios, Vazquez-Gallego & Alonso-Zarate 2015, 4.)

Julkaisija/tilaaja -toiminnon avulla voidaan toteuttaa myös niin sanottu push-toiminnallisuus. Push-toiminnallisuudessa viesti lähetetään viestin vastanottajan laitteeseen, sen sijaan että vastaanottajan laitteessa oleva sovellus kävisi noutamassa sen. Tällä tavoin voidaan sovellusten interaktiivisuutta lisätä. Lisäksi se vähentää resurssien käyttöä, kun tietoa ei tarvitse käydä tarkistamassa tietyn väliajoin.

Yleisesti IoT-laitteet toimivat muun kuin järjestelmän herätteen toimesta, esimerkiksi ovisensori toimii silloin, kun joku menee ovesta. Tästä herätteestä olisi hyvä saada järjestelmään heti tieto. Ilman julkaisija/tilaajatoiminnetta järjestelmän pitäisi käydä kysymässä jokaiselta laitteelta, onko herätteitä saatavilla. Tämä

johtaa siihen, että väistämättä tulee turhia kyselyitä, joka kuluttaa IoT-laitteiden akkua ja tietoliikenneverkkoa.

MQTT-protokolla määrittää kolme eri roolia, jotka ovat julkaisija, välittäjä ja tilaaja (Karagiannis ym. 2015, 4). Kuviossa 11 on kuvattu esimerkki MQTT- järjestelmä-arkkitehtuurista.



Kuvio 11. Esimerkki MQTT-julkaisija/tilaaja järjestelmäarkkitehtuurista

Välittäjä on MQTT-järjestelmän keskipiste. Välittäjä tarjoaa tietovarannon, johon julkaisijan herätteet tallennetaan. Välittäjä ei käsittele herätettä muuten kuin tallentamalle ne välitystä varten. Välittäjä yleensä tarjoaa useamman tietovarannon, joihin herätteitä tallennetaan. Jokaista käyttötarkoitusta varten on oma tietovaranto. Välittäjän vastuulla on välittää tiettyyn tietovarantoon tullut heräte siitä kiinnostuneelle tilaajalle.

Julkaisija on laite tai järjestelmä, joka luo herätteen ja välittää herätteen välittäjäpalvelulle. Julkaisija ei tässä vaiheessa tiedä, kuka herätteen vastaanottaa. Vastaanottajia voi olla useita tai ei yhtään. Määrä riippuu siitä, ovatko tilaajat kiinnostuneita herätteestä.

Tilaaja on järjestelmä tai palvelu, joka käsittelee julkaisijalta tulleen herätteen. Tilaaja kertoo välittäjälle mistä tietovarannosta se on kiinnostunut ja tämän jälkeen välittäjä välittää herätteen tilaajan saataville.

MQTT-protokolla määrittää kolme eri vaihtoehtoa herätteen välityksen laatuun (Karagiannis ym. 2015, 4):

1. Fire and Forget: heräte on lähetetty mutta varmuutta sen perille menosta ei tarvita.
2. Delivered at least once: heräte on välitetty ainakin kerran ja välityksestä on saatu kuittaus
3. Deliverd exactly once: heräte on välitetty vain yhden kerran ja kuittaus siitä on saatu.

5 IoT-ALUSTAN JÄRJESTELMÄARKKITEHTUURIN SUUNNITTELU JA TOTEUTUS

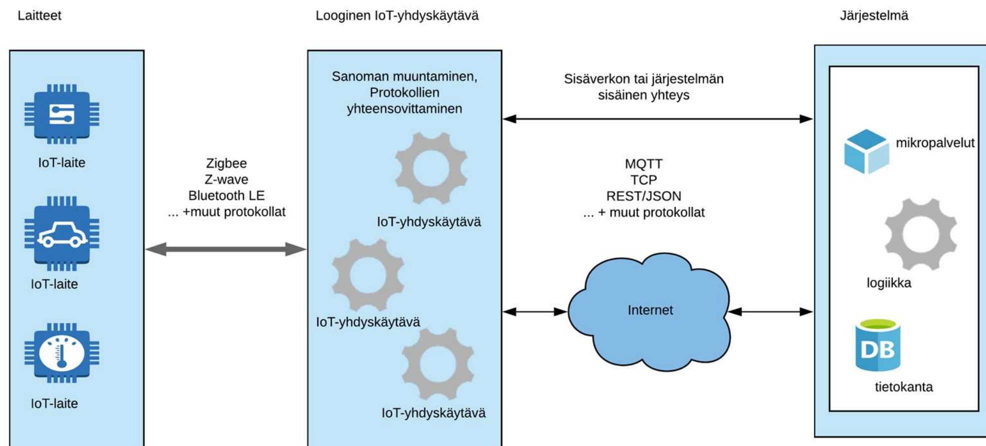
5.1 IoT-yhdyskäytävä

IoT-yhdyskäytävän tarkoitus on toimia eräänlaisena siltana varsinaisen IoT-laitteiden ja järjestelmän välillä. IoT-yhdyskäytävä muuntaa laitteen lähettämän tiedon järjestelmän yleisesti suunniteltuun muotoon. Muuntamisen avulla saadaan laitteen lähettämä binäärimuotoinen viesti paremmin käsiteltävään muotoon. Lisäksi IoT-yhdyskäytävä huolehtii tietoturvan ja mahdollisen yhteyden salaamisen laitteen ja järjestelmän välisessä kommunikoinnissa. IoT-yhdyskäytävä voidaan sijoittaa joko erilleen järjestelmän muista komponenteista tai osaksi järjestelmää.

IoT-yhdyskäytäviä voi olla useita yhden järjestelmän kokonaisuudessa. Järjestelmäarkkitehtuurin suunnittelussa IoT-yhdyskäytävien lukumäärää ei ole hyvä rajoittaa, koska hyvä suunnitteluperiaate on tukea järjestelmän laajennettavuutta. Esimerkiksi jokaista laiteläheistä protokollaa kohden tehdään oma yksittäinen IoT-yhdyskäytävä. Vastaavasti IoT-laitteiden sijoittelun perusteella voidaan niin sanotulle laitekokoelmalle luoda oma IoT-yhdyskäytävä, jonka tehtävänä on yhdistää useampi eri IoT-laite. Tyypillisesti IoT-laitteiden väliseen kommunikointiin käytetyt protokollat on suunniteltu vain lyhyen kantaman lähetykseen, jotta ne olisivat mahdollisimman energiatehokkaita. IoT-yhdyskäytävällä ei yleensä ole tätä rajoitusta, vaan se toimii verkkovirralla ja käyttää järjestelmään parhaiten sopivia protokollia.

Useampi IoT-yhdyskäytävä muodostaa yhdessä niin sanotun loogisen IoT-yhdyskäytäväkerroksen, jonka tehtävänä on mahdollistaa useiden eri laitteiden kommunikointi. Vastaanottovan järjestelmän ei tarvitse tietää, minkä IoT-yhdyskäytävän kautta viesti kulkee. Viesti sisältää yksilöllisen tunnisteen, jonka perusteella voidaan järjestelmässä tuleva sanoma identifioida tiettyyn laitteeseen.

Kuviossa 12 on kuvattu esimerkki IoT-yhdyskäytävän toiminnallisuudesta. Vasemmalla kuvassa on kuvattu IoT-laitteet, jotka lähettävät tietoa käyttäen eri kommunikaatioprotokollia IoT-yhdyskäytävälle. IoT-yhdyskäytävä huolehtii laitteiden välisen kommunikaation sekä sanoman muuntamisen järjestelmän tarvitsemaan muotoon ja sen jälkeen välittää viestin järjestelmälle käsiteltäväksi.



Kuvio 12. Looginen kuvaus IoT-yhdyskäytävästä

IoT-integraatioalustan IoT-yhdyskäytävä toteutettiin hyödyntäen kahta Raspberry-mikrokontrollerialustaa. Mikrokontrollialustalle asennettiin Z-Wave, ZigBee ja Bluetooth LE-protokollia tukevat sovittimet. Lisäksi mikrokontrollerialustalle toteutettiin protokollaa varten tarvittavat sovellukset, joiden avulla protokollan kommunikointi ja tiedon muokkaaminen tehtiin integraatioalustan käytössä olevaan muotoon. Toteutus koostui kahdesta IoT-yhdyskäytävästä ja useista IoT-laitteista. Taulukossa 1 on tarkempi kuvaus laitteista, jotka liitettiin integraatioalustaan ja protokollista, joita testattiin.

Taulukko 1. Listaus IoT-laitteista

Laite	Protokolla	Rooli
Raspberry pi 3 model B	Bluetooth LE	IoT-yhdyskäytävä
Raspberry pi 3	Z-Wave & ZigBee	IoT-yhdyskäytävä
Ruuvi Tag	Bluetooth LE	IoT-laite
Ensto thermostat	Bluetooth LE	IoT-laite
Fibaro door sensor	Z-Wave	IoT-laite
Fibaro Smoke sensor	Z-Wave	IoT-laite
Fibaro motion sensor	Z-Wave	IoT-laite

IoT-laitteet lähettävät tietoa erilaisia tietorakenteita käyttäen. Opinnäytetyön aikana yksi eniten aikaa vievistä työvaiheista oli tulkita eri laitteiden käyttämä tietorakenne. Samaa kommunikointiprotokollaa käyttävät laitteet voivat kommunikoida eri tietorakenteita käyttäen. Esimerkiksi jokaisen eri valmistajan Bluetooth LE-laitteen käyttämä tietorakenne oli erilainen. Lisäksi laitteiden lähettämä tieto oli binäärimuodossa ja tieto piti ensin tulkita merkkijonopohjaiseksi muodoksi. IoT-yhdyskäytävälle asennettiin valmiina kirjastoja, joita hyödyntäen tietorakenne muunnettiin järjestelmässä paremmin käsiteltävään muotoon.

Kuvassa 13 on esimerkki RuuviTag-laitteen lähettämästä binäärimuotoisesta tiedosta, joka on muokattu JSON-formaattiin. Alla olevan kuvan ensimmäisellä rivillä on IoT-laitteen lähettämää tietoa binaarimuodossa ja sen jälkeen JSON-formaatissa IoT-yhdyskäytävään muuttamisen jälkeen.


```
<Buffer 99 04 03 65 18 4b bf 61 ff e2 ff f6 04 03 0b d7>

{
        "dataFormat": 3,
        "rssi": -88,
        "humidity": 50.5,
        "temperature": 24.75,
        "pressure": 98993,
        "accelerationX": -30,
        "accelerationY": -10,
        "accelerationZ": 1027,
        "battery": 3031,
        "deviceId": "fefdf1c0a10a"
}
```


Kuvio 13. Esimerkki binäärimuotoisen tiedon muuttamisesta

5.2 API-yhdyskäytävä

Niin ulkoisten järjestelmien kuin sisäisten komponenttien kommunikointi järjestelmän kanssa tapahtuu keskitetysti API-yhdyskäytävän kautta. API-yhdyskäytävä tarjoaa rajapinnan järjestelmän mikropalveluihin. API-yhdyskäytävällä on tiedossa jokaisen mikropalvelun osoite ja sisään tulevan sanoman perusteella API-yhdyskäytävä reitittää sanoman oikealle mikropalvelulle.

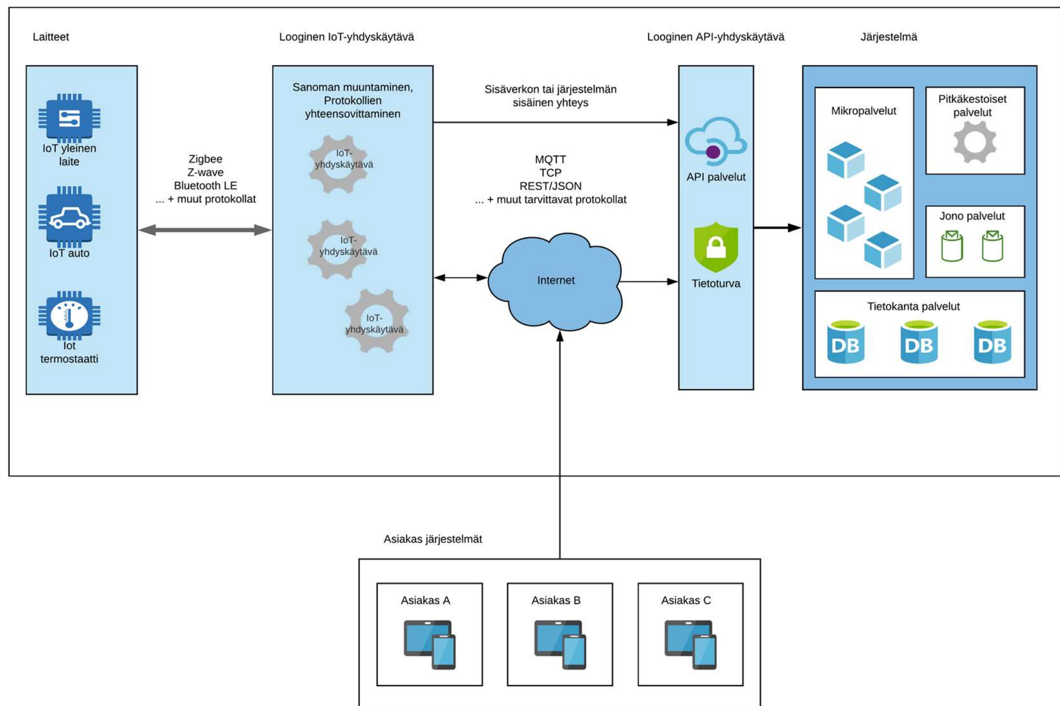
Keskitetty kommunikointipiste antaa monenlaista hyötyä. Siihen voidaan lisätä tietoturvaa ja monitorointia. Lisäksi sen avulla voidaan julkaista järjestelmän palveluita ulkoisille järjestelmille. IoT-järjestelmäarkkitehtuurissa sekä järjestelmään

kuuluvat laitteet, että asiakasjärjestelmät kommunikoivat API-yhdyskäytävän kautta. Tämä ratkaisu säästää rahaa ja on nopeampi toteuttaa, kun erillisiä rajapintoja ei tehdä asiakasjärjestelmille. API-yhdyskäytävä voi olla fyysisesti sijoitettu esimerkiksi jonkin pilvipalvelun tarjoajan konesaliin tai verkkoteknisesti erilliseen eteisverkkoon, joka on tyypillisesti erotettu verkkoalue. Oleellista kuitenkin on, että sanomaliikenne kulkee sen kautta. Muussa tapauksessa menetetään API-yhdyskäytävän tuomat hyödyt ja pahimmassa tapauksessa altistetaan järjestelmä tietoturvauhkalle. API-yhdyskäytävää voidaan pitää eräänlaisena keskitettynä turvapisteenä järjestelmään, sillä yksi sen tärkeimmistä palveluista on huolehtia tietoturvasta.

API-yhdyskäytävään on mahdollista lisätä tarkistuspalveluita. Esimerkiksi palvelunkäyttöoikeus on oleellista tarkistaa API-yhdyskäytävätasolla. Näin saadaan parannettua tietoturvaa järjestelmässä sekä voidaan varmistaa, että sanoman lähettäjä tulee siitä asiakasjärjestelmästä, mistä sen pitää tulla. Tarkistuspalveluna voidaan lisätä myös sanoman sisällön tarkistuksia estämään erilaisia tiedonkalastushyökkäyksiä tai järjestelmän käyttöä estäviä hyökkäyksiä.

API-yhdyskäytävän avulla voidaan lisätä järjestelmään vikasietoisuutta. Sille voidaan asettaa erilaisia palvelun mittareita, esimerkiksi vastausaikoja ja sanomamääriä tietylle mikropalvelulle. Näitä arvoja tarkkailemalla ja säätämällä voidaan kuormaa jakaa tarvittaessa toisessa virtuaali- tai fyysisellä palvelimella olevalle mikropalvelulle. Myös vikatilanteiden hallinta tapahtuu API-yhdyskäytävä tasolla. Jos mikropalveluun ei saada yhteyttä ongelmatilanteesta johtuen, voidaan liikenne siirtää toiselle mikropalvelulle.

API-yhdyskäytäväkäsi valittiin Google Cloud-alusta. API-yhdyskäytävä huolehtii tietoturvasta ja kuormantasauksesta. Kuviossa 14 on looginen kuvaus API-yhdyskäytäväratkaisusta. API-yhdyskäytävä toimii IoT-yhdyskäytävän ja järjestelmän välissä.



Kuvio 14. Looginen kuvaus API-yhdyskäytävästä

Google Cloud-tapauksessa jokainen laite pitää erikseen rekisteröidä, jotta yhteys olisi sallittu. Rekisteröinti tapahtuu luomalla laitteelle sertifikaatti ja asentamalla luotu sertifikaatti Google Cloud-alustalle.

Kuviossa 15 kuvataan sertifikaatin luonti Openssl-työkalulla. Osa Openssl kysymyksien vastauksista on poistettu näkyviltä tietosuojan takia.

```
pi@raspberrypi:~ $ openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 365 -out certificate.pem
Generating a RSA private key
.....+++++
.+++++
writing new private key to 'key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FI
State or Province Name (full name) [Some-State]:Lapland
Locality Name (eg, city) []:Rovaniemi
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

Kuvio 15. Esimerkki sertifikaatin luomisesta laitteelle

Kun sertifikaatti on luotu, voidaan laite rekisteröidä Google Cloud-alustalle erilliseen laiterekisteriin. Kuviossa 16 kuvataan, miten laite rekisteröidään Google Cloud-alustalle. Tunnistautumiseen käytetään edellä luotua sertifikaattia ja luodun sertifikaatin julkinen osa asetetaan laitteen asetuksiin.

Google Cloud Platform IoTProject

IoT Core ← Create a device

Create a device in registry **devices**.

Device ID

bluetooth-iot-gateway

Device communication

- (•) Allow
- ( ) Block

Authentication (Optional)

Input method

- (•) Enter manually
- ( ) Upload

Public key format

- ( ) RS256
- ( ) ES256
- (•) RS256_X509
- ( ) ES256_X509

Public key value

AJWLBpgUssB9MUb10ly1qd2KJH9yzvNFUKnBtWu+LSb5mnr0jhGjtqSbydKQFTk+
CwQ5Xw00lTukzYOeUNvZW53Bk2AGpLHguDjF66Zy5T91Q46CejBVVF40qwVqsFlg
5amyajIo1au6F+cZCbwlRwwZeQIDAQABo1MwUTAdBgNVHQ4EFgQUOw6RRsOx+ySb
7V+AQhcANlbkQ1cwHwYDVR0jBBgwFoAUOw6RRsOx+ySb7V+AQhcANlbkQ1cwDwYD
VR0TAQH/BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAnm8g/TrGoJFuPE5VQhMy
4otK6LCkp/I8EQtGOw1Fu01RpS0gn38uJ+bJT1+XaAbdUpN57swfpem6lxBquogO
t9ld7JKBGlm8flbC5Kk44tTUTiX1b7z6QDJwVbDw/LdUMWYf+026QrWlmN3Z4g3n
BtxeHdFIBfHI0VoFKuhdEIU9XTUr+ADInGnB55GE/Vc3pMXOF0R9fOZu6bDLCFIf
cGRjwlOJbg/bdPU3vbZa2hrji5r4g2hmyJpQbOiOV+bCPN3eCgfkNYx77xKfK9iy
I4R4dgbwsYLU70dHERWj6lOjhFuyg8QVDP5INBczp1oF+9nPdxv+BGJP0cgkDKOe
mg==
-----END CERTIFICATE-----

Public key expiration date (Optional)

[ ] Expires on:

9/14/20, 2:08 PM EEST

Kuvio 16. Looginen kuvaus API-yhdyskäytävästä

Sertifikaatin avulla voidaan varmistaa, että vain niillä laitteilla on oikeus kommunikoida Google Cloud-alustan kanssa, joiden sertifikaatti on laitettu Google Cloud-alustalle.

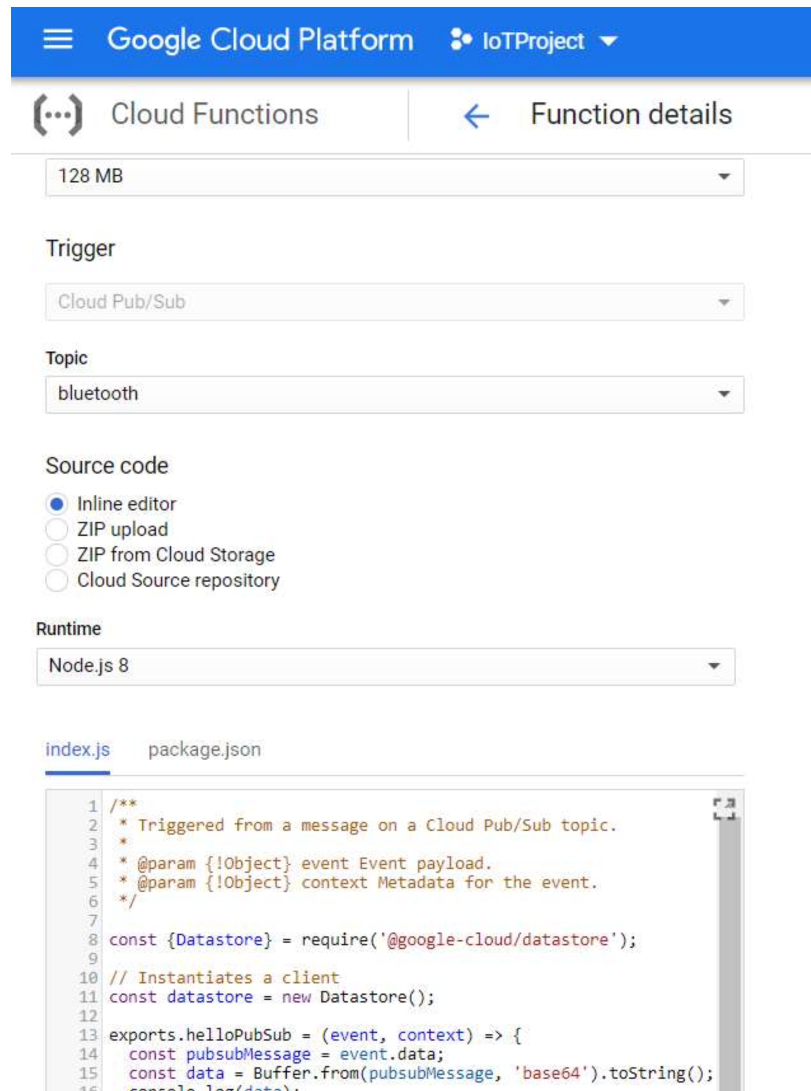
5.3 Mikropalvelut

Palvelun logiikan toteutukseen valittiin mikropalveluarkkitehtuuri mukautuvuuden vuoksi. Mikropalveluarkkitehtuurissa palvelu tai palvelukokonaisuuden osia voidaan sijoittaa toisista riippumatta eri fyysisiin sijainteihin, esimerkiksi pilvipohjaisille palvelualustoille. Tämä lähetystapa tukee hyvin IoT-integraatioalustan järjestelmäarkkitehtuurimallia. IoT-järjestelmä voi koostua useista eri laitteista ja laitteet voivat olla sijoitettu jopa eri mantereelle toisistaan. Tämän vuoksi on järkevää, että IoT-integraatioalustan järjestelmäarkkitehtuuri ei ota kantaa tai rajoita palveluiden logiikan maantieteellistä sijoittelua.

IoT-laitteiden kirjo on laaja. Se tarkoittaa myös sitä, että IoT-laitteiden välittämät viestit voivat olla erilaisia toisistaan. Tämän vuoksi on järkevää rakentaa eri kommunikointitavoille oma mikropalvelu, joka muuntaa viestin ja kommunikoi järjestelmän muiden mikropalveluiden kanssa standardiprotokollalla ja ennalta määritetyllä viestiformaatilla.

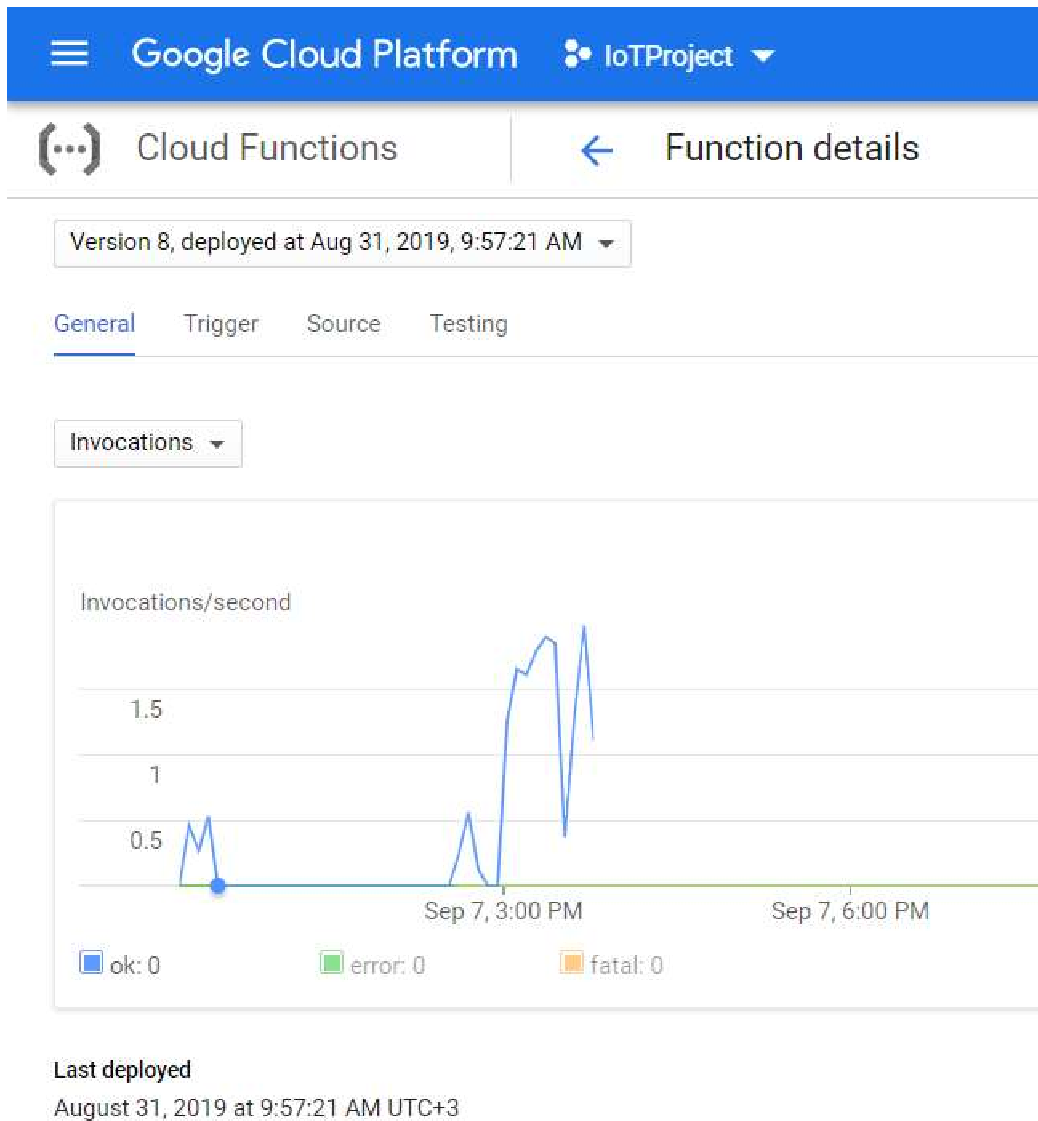
Järjestelmän ylläpidettävyys ja IoT-laitteiden tuoma tiheä päivityssykli pitää ottaa myös huomioon suunniteltaessa oikeanlaista järjestelmäarkkitehtuuria. Mikropalveluarkkitehtuuri tukee erinomaisesti järjestelmän yksittäisen osa-alueen päivittämistä, sillä mikropalvelut on rakennettu vain yksittäistä määrättyä toiminnallisuutta varten, niin ettei suoria riippuvuuksia toisiin mikropalveluihin rakenneta. SOA-järjestelmäarkkitehtuurimallissa rakennetaan useista pienimmistä palveluista niin sanottuja yhdistelmäpalveluita jokaista liiketoimintanäkökulmaa kohden. Jos yhteen osaan yhdistelmäpalvelusta pitää tehdä muutos, tarkoittaa se yleensä, että yhdistelmäpalveluun tulee myös muutos, jolloin yhdistelmäpalvelu ei ole käytössä päivityksen aikana tai pahimmassa tapauksessa aiheuttaa muutoksen useampaan yksittäiseen palveluun.

Vertailtaessa palveluiden toiminnallisuuden sekä päivityksen joustavuuden tukemista oli selvää, että mikropalvelu järjestelmäarkkitehtuuri on parempi valinta kuin SOA-järjestelmäarkkitehtuuri IoT-integraatioalustan toteutukseen. Kuviossa 17 on kuvattu esimerkki mikropalvelusta, joka on toteutettu Googlen pilvialustalle. Mikropalvelun tehtävänä on lukea MQTT-jonosta IoT-laitteiden lähettämiä viestejä. Mikropalvelu muuntaa viestin järjestelmän käytössä olevaan viestirakenteeseen ja tallentaa muokatun viestin tietokantaan. Tietokannassa olevaa viestiä voidaan myöhemmin hyödyntää järjestelmän eri toiminnallisuuksiin.



Kuvio 17. Esimerkki Google Cloud function-mikropalvelusta

Google-pilvialusta tarjoaa palvelun valmiita monitorointi- ja lokituspalveluita alustalla oleville palveluille. Kuviossa 18 on esimerkki tilastotiedosta, kuinka paljon toteutettua mikropalvelua on kutsuttu.



Kuvio 18. Esimerkki Google Cloud function-statistiikasta

5.4 Pitkäkestoiset palvelut

IoT-järjestelmä koostuu yleensä useasta eri laitteesta, jotka lähettävät toisistaan riippumattomasti tietoa. Tietojen välitys ja tallennus järjestelmään tapahtuu mikropalvelun avulla. Kun järjestelmään toteutettavan palvelun toiminnallisuudessa on tarve yhdistää useampi eri tiedon lähde ja tiedon perusteella tehdä ennalta määritetty toiminto, tarvitsee järjestelmäarkkitehtuurin tukea myös pitkäkestoisia palveluita. Järjestelmän avulla voidaan esimerkiksi seurata kohdeorganisaation sisääntuloa oviantureiden avulla: Jos kaksi tai useampaa ovea on yhtä aikaa auki, voidaan lähettää järjestelmästä hälytys. Tähän tarkoitukseen järjestelmässä pitää olla prosessi- tai sääntömoottori, jonka tarkoituksena on tarkastella eri oviantureiden tilatietoa ja tilatiedon perusteella tehdä ennalta määritetty toiminne.

On hyvin yleistä, että IoT-laitteiden ympärille rakennetaan erilaisia sääntöjä, joilla yksittäiset IoT-laitteet yhdistetään yhdeksi isommaksi toiminnoksi.

Järjestelmäarkkitehtuuri toteutetaan niin, että mikropalvelut hoitavat tiedon välityksen laitteiden välillä ja prosessi- tai sääntömoottori lukee laitteiden tilaa IoT-integraatioalustan omasta tietovarannosta. Prosessi- tai sääntömoottori ei siis suoraan kommunikoi eri laitteiden välillä. Google Cloud-alustalla ei ollut valmiina tiedon käsittelyyn tarvittavia toiminnallisuuksia. Tätä tarvetta varten luotiin Google Cloud-alustalle virtuaalipalvelin. Käytännössä virtuaalipalvelin olisi voinut olla ihan miltä tahansa virtuaalipalvelimen tarjoajalta, mutta palveluiden keskitetty hallinta oli merkittävä tekijä virtuaalipalvelimen valinnassa. Kuviossa 19 kuvataan virtuaalipalvelimen luonti. Virtuaalipalvelin sijoitettiin Pohjois-Eurooppaan ja alustavien laskelmien mukaan minimiresurssit olivat riittävät.

Google Cloud Platform IoTProject
Create an instance
To create a VM instance, select one of the options:
New VM instance
Create a single VM instance from scratch
New VM instance from template
Create a single VM instance from an existing template
Marketplace
Deploy a ready-to-go solution onto a VM instance
Name
virtualserver1
Region
europe-north1 (Finland)
Zone
europe-north1-a
Machine configuration
Machine family
General-purpose
Memory-optimized
Machine types for common workloads, optimized for cost and flexibility
Generation
First
Powered by Skylake CPU platform or one of its predecessors
Machine type
f1-micro (1 vCPU, 614 MB memory)
vCPU
1 shared core
Memory
614 MB
CPU platform and GPU
Container
Deploy a container image to this VM instance. Learn more
Boot disk
New 10 GB standard persistent disk
Image
Ubuntu 18.04 LTS Minimal
Change

Kuvio 19. Esimerkki virtuaalipalvelimen luonnista

Virtuaalipalvelimen asennuksen jälkeen seuraava tehtävä oli avata tarvittavat tietoliikenneportit. Google Cloud-alustalla ei ole oletuksena yhtään tietoliikenneporttia avattu. Tällä tavalla parannetaan virtuaalipalvelimen tietoturvaa, kun jokainen tarvittava tietoliikenneportti on erikseen aukaistava palomuurista. Kuviossa 20 kuvataan, miten tietoliikenneportit avataan päivittämällä

palomuuriporttiasetuksia. Normaalia selainpohjaista liikennettä varten avattiin portit 80 ja 443 sekä asennettavien palveluiden käyttämät portti 3000 sekä portit 8083–8088.

Google Cloud Platform IoTProject

VPC network

- VPC networks
- External IP addresses
- Firewall rules
- Routes
- VPC network peering
- Shared VPC
- Serverless VPC access

Firewall rules CREATE FIREWALL RULE REFRESH DELETE

Firewall rules control incoming or outgoing traffic to an instance. By default, incoming traffic from outside your network is blocked. Learn more

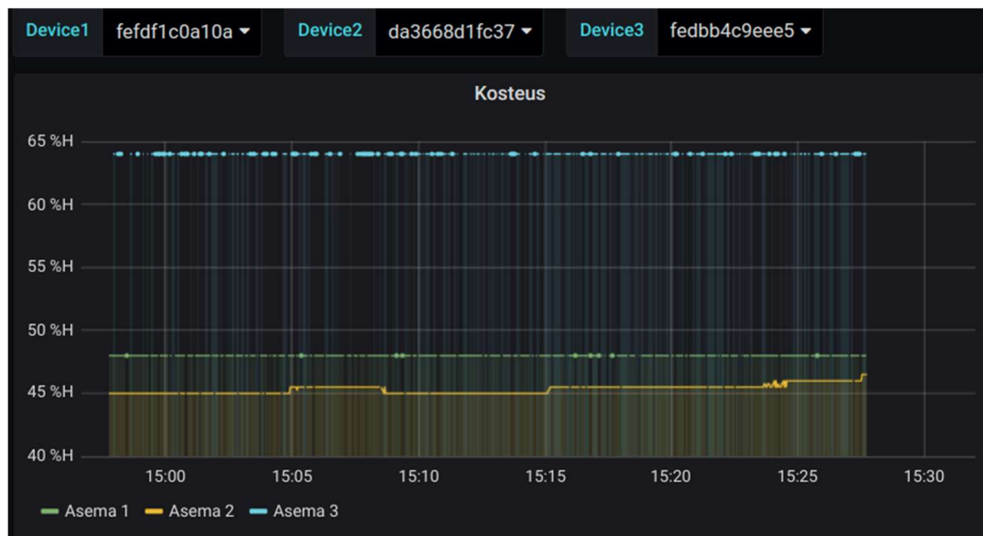
Note: App Engine firewalls are managed here.

Filter resources Columns

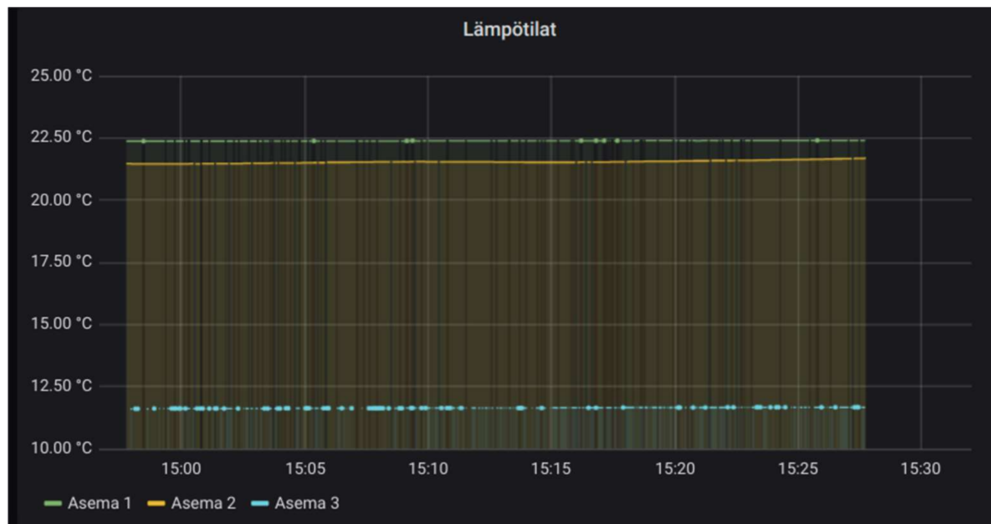
Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network
default-allow-http	Ingress	http-server	IP ranges: 0.0.0.0/0	tcp:80	Allow	1000	default
default-allow-https	Ingress	https-server	IP ranges: 0.0.0.0/0	tcp:443	Allow	1000	default
iotuiports	Ingress	Apply to all	IP ranges: 0.0.0.0/0	tcp:3000; tcp:8083-8088	Allow	1000	default

Kuvio 20. Esimerkki virtuaalipalvelimen luonnista

Seuraavassa vaiheessa virtuaalipalvelimelle asennettiin tarvittavat sovellukset. IoT-laitteiden tietojen visualisointiin valittiin Grafana-sovellus. Grafana soveltuu erityisen hyvin visualisoimaan timeseries-muotoista tietoa. Kuviossa 21 ja kuviossa 22 visualisoidaan useamman eri laitteen lähettämää tietoa samalla ajanhetkellä. Kuviossa 21 visualisoidaan kolmen eri laitteen lähettämää kosteusprosessia ja kuviossa 22 visualisoidaan samojen laitteiden lämpötiloja.



Kuvio 21. Esimerkki antureiden tiedon hyödyntämisestä



Kuvio 22. Esimerkki virtuaalipalvelimen luonnista

5.5 Jonopalvelut

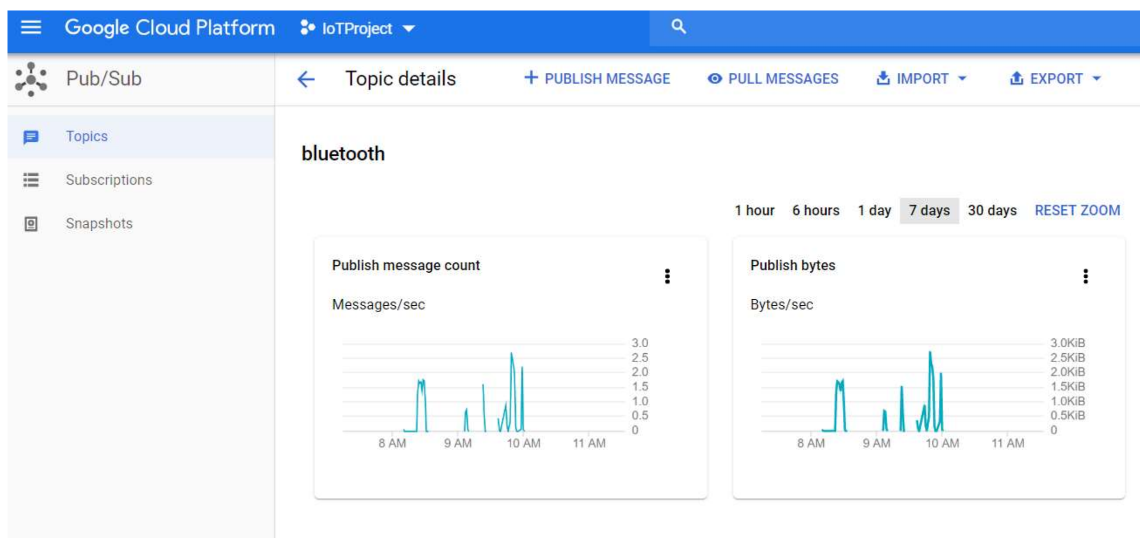
Järjestelmään tulevia viestejä voidaan niin sanotusti jonottaa jonopalveluiden avulla. Jonottamien tarkoittaa sitä, että viestiä ei välttämättä heti käsitellä vaan sanoma tallennetaan tietovarantoon myöhempää käsittelyä varten. Tämä on erityisen hyödyllistä silloin, kun järjestelmään tulee viestejä nopeampaa kuin mikropalvelut ehtivät niitä käsitellä. Yleensä mikropalveluiden logiikkaan kuuluu tiedon käsittelyä tai tiedon rikastamista esimerkiksi hakemalla lisätietoa toisesta lähteestä. Lisätiedon hakemiseen ja käsittelyyn menee prosessoriaikaa ja tästä syystä voi tulla tilanne, jossa järjestelmän käytettävät resurssit loppuvat. Tämä aiheuttaa sen, ettei järjestelmä pysty käsittelemään tarvittavaa määrää sisään tulevaa tietoa ja siitä aiheutuu ongelmia järjestelmän käyttöön ja asiakasjärjestelmille.

Jonopalveluiden avulla voidaan kontrolloida, kuinka paljon mikropalvelut voivat asynkronisesti käsitellä viestejä yhtä aikaa. Jos järjestelmä vastaanottaa enemmän viestejä kuin voi käsitellä, siirtyvät viestit jonoon. Kun mikropalveluita vapautuu viestin käsittelystä, välittää jonopalvelu jonosta seuraavan sanoman käsittelyyn. Tällä tavoin järjestelmään ei tule ylikuormitustilannetta ja kaikki järjestelmään tulevat viestit voidaan käsitellä ilman ylikuormituksesta aiheutuvaa tilannetta.

Jonopalveluiden kommunikointi on yleensä toteutettu käyttäen yleisesti tunnettua kommunikointiprotokollaa. Yksi tämänlainen kommunikointiprotokolla on MQTT. MQTT on suunniteltu olemaan mahdollisimman tehokas ja viemään

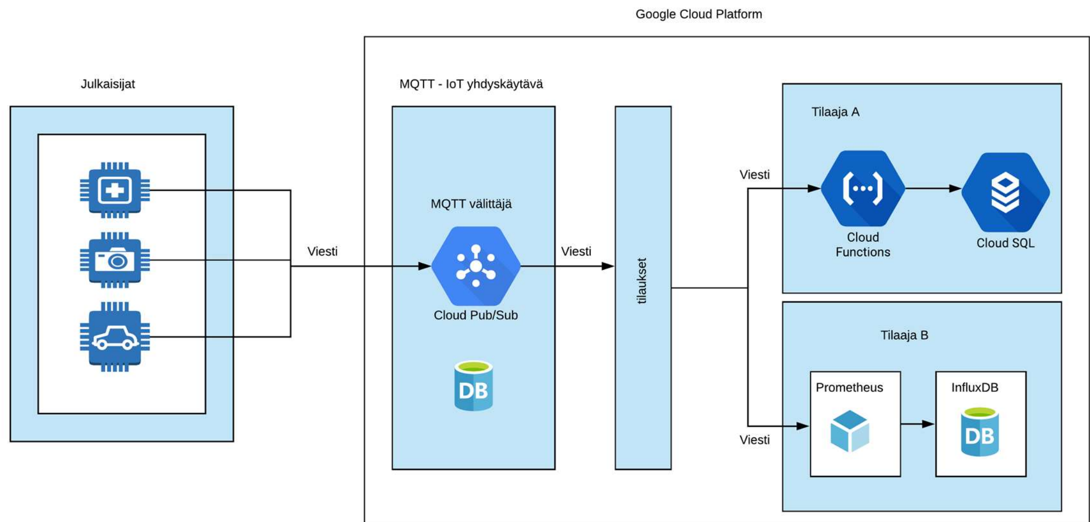
mahdollisimman vähän tiedonsiirron resursseja. Tämän vuoksi se soveltuu erityisen hyvin IoT-laitteiden kanssa kommunikointiin.

MQTT-jonototeutukseksi valittiin Google Cloud-alustan Pub/Sub-toiminnallisuus. Pilvipalvelukonsepti koettiin juuri tähän tarkoitukseen sopivaksi. Se tarjoaa valmiiksi tarvittavan tietoturva-, analytiikka- ja valvontapalvelut. Kuviossa 23 nähdään yhden jonon tilastotietoa, kuten viestien määrä ja koko. Jonoon viestit on toimittanut API-yhdyskäytävä.



Kuvio 23. Tilastotietoa Google Cloud-alustan MQTT-jonosta

Jonoon tallennettu viesti tai jonototeutus ei itsestään toteuta mitään liiketoimintavaatimusta, vaan jono tarjoaa vain pysyvän paikan viesteille. Jotta viestejä voitaisiin käsitellä, pitää jonosta lukea viestiä. Google Cloud-alustalla se tapahtuu luomalla tilaaja (*subcription*), jonka avulla voidaan viestejä hakea jonosta jatkokäsittelyyn. Lisäksi jokaiselle jonolle voidaan määrittää useampi subscription, jonka avulla yksittäisen viestin käsittely voidaan monistaa tehokkaasti useampaan omaan haaraan. Kuva 24 havainnollistaa monistamisen hyödyn. Kuvassa yksittäinen sanoma menee kolmelle eri mikropalvelulle käsittelyyn.



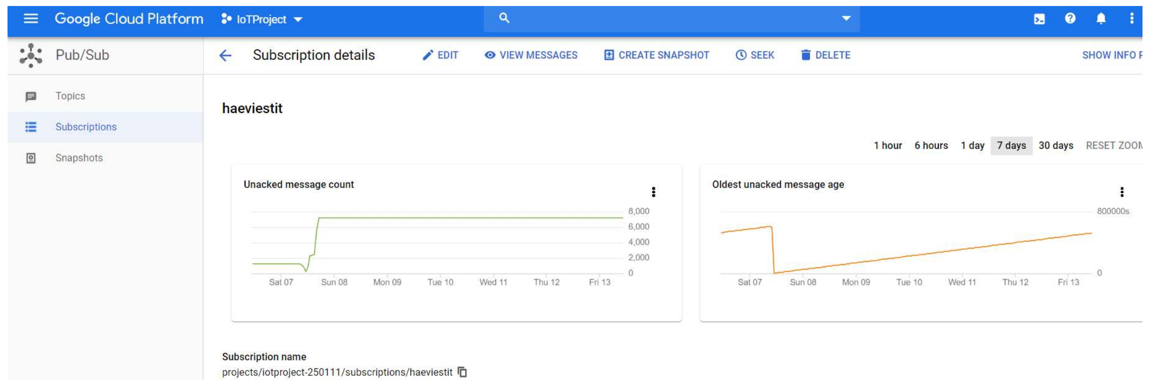
Kuvio 24. MQTT-viestin monistaminen

Tilaaja voi olla Pull tai Push-tyyppinen, lisäksi sille voidaan määritellä erilaisia vanhentumisarvoja. Pull-tyyppisellä tilauksella voidaan viestejä hakea ja Push-tyyppisellä tilauksella voidaan viestejä tallentaa jonoon. Erillisen tilauksen hyötynä on hallinta siihen, kuinka paljon sitä hyödyntävä järjestelmä tai järjestelmän osa voi käyttää resursseja. Kuvassa 25 nähdään, miten subscription luodaan Google Cloud-alustalle.

Google Cloud Platform
IoTProject
Pub/Sub
Topics
Subscriptions
Snapshots
Create subscription
A subscription directs messages on a topic to subscribers. Messages can be pushed to subscribers immediately, or subscribers can pull messages as needed.
Subscription ID *
haeviestit
Subscription name: projects/iotproject-250111/subscriptions/haeviestit
Topic name *
projects/iotproject-250111/topics/bluetooth
Existing Cloud Pub/Sub topic to subscribe from. The topic name should be in the format of projects/<project-id>/topics/<topic-id>
Delivery type
Pull
Push
Subscription expiration
Expire after this many days of inactivity (up to 365)
31 Days
A subscription is inactive if there is no subscriber activity such as open connections, active pulls, or successful pushes.
Never expire
The subscription will never expire no matter the activity.
Acknowledgement deadline
Deadline time is from 10 seconds to 600 seconds
10 Seconds

Kuvio 25. Google Cloud MQTT-tilauksen luominen

Luonnin jälkeen tilausta voidaan hyödyntää eri palveluissa. Kuvassa 26 voidaan nähdä statistiikkaa, kuinka edellä luotua tilausta on hyödynnetty.



Kuvio 26. MQTT-tilauksen tilastotietoja

5.6 Tietokantapalvelut

Järjestelmällä on yleensä tarve tallentaa palveluiden käsittelemiä tietoja, erilaisia järjestelmän käyttämiä ohjaustietoja tai asetuksia tietovarantoon. Tietovarantoja voi olla erilaisia, esimerkiksi tiedostot, järjestelmällä käytössä oleva muisti sekä tietokanta. Järjestelmän käsittelemä tieto voi olla rakenteeltaan monenlaista. Eri tietorakenteiden tallentaminen on yksi määrittävä tekijä, minkä perusteella valitaan järjestelmän käyttöön tietovaranto. Tietovarannosta tiedon hakemisen tulisi olla tehokasta ja mahdollistaa tiedon rakenteellisen hakemisen. Tiedon pysyvä tallentaminen sekä ongelmatilanteiden hallinta ovat myös tärkeitä tekijöitä tietovarannon valinnassa. Edellä mainittujen vaatimusten perusteella on luonnollista valita IoT-integraatioarkkitehtuurin tietovarannoksi tietokantapalvelut.

Mahdollisia tietokanta vaihtoehtoja ovat SQL, NoSQL ja Timeseries-tietorakenteita tukevat tietokannat. SQL-tietokanta on perinteinen relaatiopohjainen tietorakenne, jossa eri taulujen välillä on yhteyksiä ja tietorakenne ei ole jatkuvan vaihtelun alla. SQL-tietokannasta rakenteellisen tiedon tallentaminen sekä hakeminen on tehokasta. NoSQL (*non SQL*) on suunniteltu tilanteisiin, joissa tietorakenne voi vaihtua useasti tai jatkuvasti. Timeseries-tietokanta tallentaa tietoa aina sidottuna tiettyyn ajanhetkeen. IoT-laitteiden luonteeseen parhaiten näistä sopii timeseries-tietorakennetta tukeva tietokanta, sillä tyypillisesti IoT-laitteiden lähettämä tieto on sidottu lähettämisen hetkeen. Toisin sanoen IoT-laitteet lähettävät yksittäisiä tilatietoja eikä lähetettävässä tiedossa ole riippuvuutta aikaisemmin lähetettyyn tietoon.

Tietokannaksi valikoitiin InfluxDB sen hyväksi koetun timeseries-tietorakenteen tuen perusteella. Kuviossa 27 on esimerkki InfluxDB-tietokannan sisällöstä. Vasemmalla nähdään aikaleima. Muut sarakkeet ovat IoT-laitteen lähettämää tietoa. DeviceId-sarakkeen avulla voidaan erottaa tietoja tiettyyn IoT-laitteiseen. Toteutuksessa DeviceId-tiedon asettaminen viestiin toteutettiin IoT-yhdyskäytävä ratkaisuun.

```
> select * from cloud_pubsub group by * order by DESC limit 10
name: cloud_pubsub
tags: host=influxdb
time                accelerationX accelerationY accelerationZ battery dataFormat deviceId     humidity pressure rssi temperature
----                ------------- ------------- ------------- ------- ---------- --------     -------- -------- ---- -----------
1567860288723189629 -28           -2            1023          3031    3          da3668d1fc37 47       100761   -79  21.94
1567860288249374267 -38           -9            1025          3043    3          fefdf1c0a10a 48       100730   -93  22.44
1567860287715912970 -32           -2            1028          3037    3          da3668d1fc37 47       100761   -71  21.94
1567860286246328457 -35           -7            1029          3037    3          fefdf1c0a10a 48       100730   -95  22.44
1567860285730722436 -28           3             1026          3031    3          da3668d1fc37 47       100761   -79  21.94
1567860284710311316 -26           2             1021          3031    3          da3668d1fc37 47       100761   -82  21.94
1567860284211219653 -35           -7            1025          3037    3          fefdf1c0a10a 48       100730   -88  22.44
1567860283308594063 45            83            1024          2995    3          fedbb4c9eee5 64       100618   -93  11.72
1567860282705853745 -27           3             1021          3049    3          da3668d1fc37 47       100761   -55  21.94
1567860282193342713 -33           -8            1027          3043    3          fefdf1c0a10a 48       100730   -75  22.44
>
```
Kuvio 27. Tietokannan datarakenne

6 KOKEILUN TULOKSET

Opinnäytetyön tavoitteena oli suunnitella toimiva järjestelmäarkkitehtuuri sekä mahdollistaa IoT-laitteiden ja järjestelmän välinen kommunikointi. Työ aloitettiin hankkimalla teoreettinen tietämys järjestelmäarkkitehtuurin periaatteista. Tietoa hankittiin myös järjestelmäarkkitehtuurivaihtoehdoista ja vaihtoehtojen ominaisuuksista. Kun järjestelmänarkkitehtuurin tietämys oli saavutettu, siirryttiin selvittämään eri kommunikointiprotokollien ominaisuuksia. Kommunikointiprotokolliksi valittiin useampi IoT-laitteille sopiva protokolla. Seuraavassa vaiheessa toteutettiin kokeilu, jossa teoreettisen tietämyksen perusteella valittuja vaihtoehtoja kokeiltiin käytännössä. Kokeilussa järjestelmäarkkitehtuuriin syntyi viisi eri osa-aluetta: IoT-yhdyskäytävä, API-yhdyskäytävä, mikropalvelut, jonopalvelut, pitkäkestoiset palvelut ja tietovarannot.

IoT-yhdyskäytäväratkaisun avulla järjestelmään saatiin liitettyä useita eri laitteita ja mahdollistettiin käyttöön useampi kommunikointiprotokolla laitteiden väliseen kommunikointiin. Laajentamalla IoT-yhdyskäytäväratkaisua voidaan järjestelmään liittää tulevaisuudessa myös uusia laitteita ja kommunikointiprotokollia. IoT-yhdyskäytäväratkaisun avulla järjestelmäarkkitehtuurin laajennettavuus voitiin taata.

API-yhdyskäytäväratkaisun avulla saatiin IoT-yhdyskäytävän välittämät viestit järjestelmään. API-yhdyskäytävä tarjosi tietoturvallisen ratkaisun viestien vastaanottamiseen. Tämä on erityisen tärkeää, sillä yhdyskäytävä on keskitetty piste järjestelmän ja ulkopuolisen verkon välillä. API-yhdyskäytäväratkaisun avulla järjestelmäarkkitehtuuriin saatiin lisättyä tietoturvaa sekä mahdollistettiin järjestelmän laajennettavuus.

Mikropalveluarkkitehtuurin avulla voitiin järjestelmään tehdä toiminnallisuuksia Google Cloud-alustalle sekä erilliseen virtuaalipalvelimeen. Google Cloud-alusta soveltui mikropalveluiden toteuttamiseen. Mikropalveluarkkitehtuurin todettiin soveltuvan hyvin suunniteltuun järjestelmäarkkitehtuuriin.

Kokeilun aikana järjestelmään tehtiin yksinkertaisia toteutuksia laitteiden lähettämän tiedon hyödyntämistä varten. Tämä osa-alue on sellainen, missä on tulevaisuudessa eniten kehitysvaihtoehtoja. Kun tämänkaltainen järjestelmä otetaan tuotannolliseen käyttöön, niin myös tiedon hyödyntämisen tarpeet tarkentuvat.

7 POHDINTA JA JOHTOPÄÄTÖKSET

Työn tekeminen eteni suunnitellusti konstruktiivisen tutkimuksen vaiheita soveltuvin osin noudattaen. Ensimmäisenä tehtävänä oli määritellä mielekäs ongelma. Alun perin työ oli tarkoitus tehdä liiketoimintalähtöisesti asiakkaalle. Erinäisten muutosten jälkeen todettiin, että työtä ei toteuteta asiakkaalle ja keskitytään yleisempään ratkaisuun. Tässä vaiheessa työ rajattiin koskemaan järjestelmäarkkitehtuurin suunnitteluun ja suunnitelman kokeiluun. Työn aiheen valintaan vaikutti se, että 2010-luvulla nähdään juuri samanlaisten ratkaisujen tarpeellisuuden kasvavan. Ratkaisun käyttömahdollisuudet ovat laajat. Lisäksi teoriapohjan hankinnalla on oleva merkittävää hyötyä tulevaisuudessa omissa työtehtävissä sekä markkinoinnissa.

Vastaavasta kokeilusta ei ollut mahdollista löytää suomenkielistä tai kansainvälistä materiaalia. Aihealueen pienimmistä kokonaisuuksista on kansainvälistä teoriaa olemassa. Tämän vuoksi teoriapohjan hankintaan käytettiin pääasiassa kansainvälistä tietokirjallisuutta ja teknisiä julkaisuja. Työn aikana pienempiä kokonaisuuksia yhdistettiin ja analysoitiin niiden toimivuutta yhdessä.

Jo varhaisessa vaiheessa kävi ilmi, että on olemassa useita eri IoT-laitteita, järjestelmäarkkitehtuurin suunnittelumalleja sekä kommunikointiprotokollia. Työn aihe rajattiin käsittelemään vain osaa vaihtoehdoista. Rajauksessa käytettiin hyväksi omaa kokemuspohjaista tietoa sekä teknisiä julkaisuja. Työn järjestelmän arkkitehtuurisuunnitelma ja -kokeilu toteutettiin niin, että aiheen rajaus ei vaikuta siihen, miten kokeilua voitaisiin tulevaisuudessa laajentaa käyttämään myös rajauksen ulkopuolisia kommunikointiprotokollia.

Kokeilun luonteen takia oli järkevää etsiä edullista ratkaisua kokeilun palvelualustaksi. Erilaiset pilvipalvelualustaratkaisuja tutkittiin työn yhteydessä ja päädyttiin valitsemaan Google Cloud-alusta. Valintaan vaikutti se, että Google tarjosi ilmaista kokeilujaksoa, jolla kokeilu voitiin suorittaa. Työssä tehtiin niin sanottu Hydrid-ratkaisu eli yhdistelmäratkaisu. Hydrid-ratkaisussa luotiin sekä virtuaalipalvelin että käytettiin valmiita Google Cloud-palveluita. Suunniteltu järjestelmäarkkitehtuuri soveltui tähän ratkaisuun hyvin ja antoi luotettavaa tietoa siitä, että järjestelmäarkkitehtuuri mahdollistaa järjestelmän laajennettavuuden. Laajennettavuus oli yksi suunnittelun keskeisistä lähtökohdista. Google Cloud-alustan

hyödyntäminen kokeilussa tuotti tietoa sekä teknisestä että liiketoimintanäkökulmasta.

Työn järjestelmänarkkitehtuurin kokeiluun saatiin liitettyä useampia eri IoT-laitteita sekä useampia eri kommunikointiprotokollia, kuten Bluetooth LE ja Z-Wave. Työn tekemisen yhteydessä havaittiin, että juuri laitteiden välinen kommunikointi on haastavinta toteuttaa tämän kaltaisissa ratkaisuissa. Tämän vuoksi työn edetessä päädyttiin käyttämään valmiita ohjelmistoratkaisuja laitteiden väliseen kommunikointiin ja kommunikointiprotokollan tulkitseminen rajattiin ulos työn laajuudesta.

Työn tekijänä arvioin, että työ kokonaisuudessa onnistui hyvin. Teoreettisesti opinnäytetyö vastasi sille asetettuihin lähtövaatimuksiin ja kokeilu vahvisti aiempaa teoreettista tietoperustaa. Teoria painottui selvittämään järjestelmäarkkitehtuurin suunnittelua ja kommunikointiprotokollia. Erilaiset IoT-laitteet ja niiden tuomat hyödyt tulevat kasvamaan liiketoiminnassa tulevaisuudessa. Opinnäytetyössä toteutettuja ratkaisuja voidaan hyödyntää sovelletusti erilaisissa ratkaisuissa. IoT-laitteiden teknologisia ratkaisumalleja hyödynnetään jo laajasti, esimerkiksi NB-IoT-laitteita hyödyntämällä pystytään seuraamaan eläinten liikkeitä ja terveydentilaa mobiilisovelluksen avulla (Uusi teknologia 2019).

Opinnäytetyön tekemisen aikana opin uutta arkkitehtuurin suunnittelusta ja vahvistin omaa aiemmin hankittua tietämystäni. Uuden teorian oppiminen ja aiemman tietämyksen vahvistaminen auttaa minua toimimaan arkkitehdin työtehtävissä. Osaamisen kehittyminen auttaa myös johtamaan tiimiä entistä paremmin sekä teknisten päätösten tekeminen tarkentuu vahvistetun tietämyksen perusteella. Uskon, että tulen jatkossakin tekemään vastaavanlaisia kehitysprojekteja, ja tässä opinnäytetyössä hankittu tieto toimii hyvänä tietopohjana niiden kehittämisessä.

LÄHTEET

Al-Sarawi, S., Anbar, M., Alieyan, K. & Alzubaidi, M. 2017. Internet of Things (IoT) Communication Protocols: Review Viitattu 15.08.2019 https://www.researchgate.net/profile/Kamal_Alieyan/publication/320614944_Internet_of_Things_IoT_Communication_Protocols_Review/links/59f06dfeaca272cdc7ca2a64/Internet-of-Things-IoT-Communication-Protocols-Review.pdf.

Balani, N. & Hathi, R. 2009. Apache CXF Web Service Development: Develop and Deploy SOAP and RESTful Web Services. Olton: Packt Publishing.

Blank, A. G. 2004. TCP/IP Foundations. San Francisco: Sybex.

Erl, T. 2009, SOA Design Patterns. Boston: Prentice Hall.

Farahani, S. 2011. ZigBee wireless networks and transceivers. Viitattu 25.08.2019 https://books.google.fi/books?hl=fi&lr=&id=m5NYbUpqXY0C&oi=fnd&pg=PP1&dq=ZigBee&ots=9jL-zr5Zdj&sig=2fOc2DT3SdV8Afc0b00mtmMr1JM&redir_esc=y#v=onepage&q=ZigBee&f=false.

Goralski, W. 2008. The Illustrated Network: How TCP/IP Works in a Modern Network. Burlington: Morgan Kaufmann.

Ingeno, J. 2018. Software Architect's Handbook- Become a successful software architect by implementing effective architecture concepts. Birmingham: Packt publishing.

Kalin, M. 2009. Java Web Services – Up and Running. Sebastopol: O’Reilly Media Inc.

Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F. & Alonso-Zarate, J. 2015. A Survey on Application Layer Protocols for the Internet of Things Viitattu 02.08.2019 https://www.researchgate.net/profile/Periklis_Chatzimisios/publication/303192188_A_survey_on_application_layer_protocols_for_the_Internet_of_Things/links/577b656608ae213761c9d91d.pdf.

Lukka, K. 2001. Konstruktiivinen tutkimusote. Viitattu 11.8.2019 https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/.

Martin, R.C. 2018. Clean Architecture. England: Prentice Hill.

Mekki, K., Bajic, E., Chaxel, F., Meyer, F. 2018. Overview of cellular LPWAN technologies for IoT-deployment: Sigfox, LoRaWAN, and NB-IoT. Viitattu 28.06.2019 https://www.researchgate.net/profile/Kais_Mekki2/publication/323907156_Overview_of_Cellular_LPWAN_Technologies_for_IoT_Deployment_Sigfox_LoRaWAN_and_NB-

IoT/links/5c51a442458515a4c74af20f/Overview-of-Cellular-LPWAN-Technologies-for-IoT-Deployment-Sigfox-LoRaWAN-and-NB-IoT.pdf.

Ojasalo, K., Moilanen, T. & Ritalahti, J. 2014. Kehittämistyön menetelmät – Uudenlaista osaamista liiketoimintaan. Helsinki: Sanoma Pro.

Pei, Z., Deng, Z., Yang, B. & Cheng, X. 2008. Application-Oriented Wireless Sensor Network Communication Protocols and Hardware Platforms: a Survey Viitattu 02.08.2019 https://www.researchgate.net/profile/Zhidong_Deng/publication/224327774_Application-oriented_wireless_sensor_network_communication_protocols_and_hardware_platforms_A_survey/links/570da09e08ae2b772e432504/Application-oriented-wireless-sensor-network-communication-protocols-and-hardware-platforms-A-survey.pdf.

Raj, P., Raman, A. & Subramanian, H. 2017. Architectural patterns. Birmingham: Packt publishing.

Richardson, C. 2019. Microservices Patterns. The United States of America: Manning Publications Co.

Sacha, J., Biskupski, B., Dahlem, D., Cunningham, R., Meier, R., Dowling, J. & Haahr, M. 2010. Decentralising a service-oriented architecture. Viitattu 26.08.2019 https://link.springer.com/article/10.1007%2Fs12083-009-0062-6.

Sinha, R. S., Wei, Y. & Hwang, S-H. 2017. A survey on LPWA technology: LoRa and NB-IoT. Viitattu 25.08.2019 https://doi.org/10.1016/j.icte.2017.03.004.

Strimbei, C., Dospinescu, O., Strainu, R-M. & Nistor, A. 2015. Software Architectures- Present and Visions. Informatica Economică vol. 19, no. 4/2015 Viitattu 01.09.2019 https://s3.amazonaws.com/academia.edu.documents/43577900/Strimbei_Dospinescu_Strainu_Nistor.pdf?response-content-disposition=inline%3B%20filename%3DSoftware_Architectures_Present_and_Visio.pdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWOWYYGZ2Y53UL3A%2F20190928%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20190928T145632Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=b2b56a7582f571f733a49e60d36f743ff27d507461717eafbc5fd8cada112132.

Subramanian, H. & Raj, P. 2019. Hands-on RESTful API Design Patterns and Best Practices. Birmingham: Packt publishing.

Top 10 emerging technologies of 2019. 2019. Viitattu 01.09.2019 https://www.techrepublic.com/article/top-10-emerging-technologies-of-2019/.

Townsend, K., Cufí, C., Akiba & Davidson, R. 2014. Getting Started with Bluetooth Low Energy: Tools and Techniques for Low. Sebastopol: O'Reilly Media Inc.

Unwala, I. & Lu, J. 2017. IoT-Protocols: Z-Wave and Thread. Viitattu 26.07.2019 http://www.ijfrcsce.org/download/browse/Volume_3/November_17_Volume_3_Issue_11/1511943149_29-11-2017.pdf.

Uusi teknologia. 2019. NB-IoT-napit poron korviin. Viitattu 14.09.2019 https://www.uusiteknologia.fi/2018/12/24/nb-iot-napit-poron-korviin/.