jamk

Tieliikennehäiriöpalvelun toteuttaminen MQTT-protokollan avulla

Elmer Tuukkanen

Opinnäytetyö, AMK
Toukokuu 2022
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Tuukkanen, Elmer

Tieliikennehäiriöpalvelun toteuttaminen MQTT-protokollan avulla

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2022, 53 sivua

Tietojenkäsittely ja tietoliikenne. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Opinnäytetyön toimeksiantaja toimi Nodeon Finland Oy. Opinnäytetyö sai alkunsa osana erästä hanketta, jossa Nodeon oli mukana monen muun yrityksen kanssa.

Hankkeessa syntyi tarve selvittää, löytyykö autonomisten bussien ajamilta reitiltä mitään tieliikenteen häiriöitä. Tästä tarpeesta syntyi aihe kehittää tieliikennehäiriöpalvelu, joka selvittää, vaikuttaako sille lähetettyihin reitteihin mitään tiedossa olevia häiriötä. Häiriöiden löytämisen lisäksi oli palvelun tavoitteena myös seurata käyttäjien lähettämiä reittejä uusien häiriöiden varalta. Palvelun löytäessä häiriö sille lähetetyltä reitiltä, täytyi sen myös pystyä julkaisemaan viesti MQTT-protokollan avulla aiheeseen, josta reitin lähettäjä saisi tiedon, että lähettämällään reitillä on tieliikennehäiriö.

Työ toteutettiin tutkimuksellisena kehittämistyönä, sillä työ eteni teknologioiden tutkimisesta kohti itse konkreettista kehittämistoimintaa. Työn toteutusvaihe jaettiin viiteen eri vaiheeseen, joita olivat häiriörajapintojen etsiminen ja käyttäminen, häiriöiden löytäminen ajoneuvon reitiltä, MQTT-toteutus, REST-rajapinnan toteutus ja viimeisenä vaiheena toimi palvelun kontittaminen ja pystyttäminen Azureen.

Opinnäytetyön tavoitteet saavutettiin ja työn lopputuloksena syntyi toimiva palvelu, joka pystyy seuraamaan sille lähetettyjä koordinaattipisteistä koostuvia reittejä ja myös ilmoittamaan reitin lähettäjälle, mikäli lähettämältään reitiltä löytyy häiriö. Vaikka työn tavoitteet saavutettiin, syntyi työn toteutuksen aikana mahdollisia jatkokehityksen kohteita, joiden avulla palvelusta saa vieläkin toimivamman.

Avainsanat (asiasanat)

MQTT, .NET, Azure, Docker, tieliikennehäiriöpalvelu

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Tuukkanen, Elmer

Implementation of the road traffic disruption service using the MQTT protocol

Jyväskylä: JAMK University of Applied Sciences, May 2022, 53 pages

Information and Communications. Degree Programme in Information and Communication Technology. Bacheloŕs thesis

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

The employer for this study was Nodeon Finland Oy. The thesis started as part of a project in which Nodeon is involved with many other companies.

In the project there was a need to find out if there were any traffic disruptions that would affect the route of an autonomous bus. Because of this need, a road traffic disruption service was made that can find out if routes sent by end users are affected by any traffic disruptions. The goal of the service was to find all disruptions that affect the route and send information about the disruption to the sender of the route.

Thesesis was done in the way of research development, because it progressed from researching technologies towards concrete development activities. The implementation phase of the work was divided into four different phases, which were the search and use of disruption APIs, the detection of disruption in the vehicle route, the implementation of MQTT and the final phase was the containerization and setupping of the service to Azure.

The goals of the thesis were achieved. The result of the thesis was a functional service that can follow the routes sent to it and also inform the sender of the route if route is affected by a disruption. Although the goals of the work were achieved, possible improvements for further development emerged during the implementation of the work to make the service even better.

Keywords/tags (subjects)

MQTT, .NET, Azure, Docker, road traffic disruption service

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

Lyhenteet

ACI	Azure Container Instances
API	Application Programming Interface
MQTT	Message Queueing Telemetry Transport
QoS	Quality of Service
REST	Representation State Transfer
TLS	Transport Layer Security

1 Johdanto

Opinnäytetyön toimeksiantaja toimi Nodeon Finland Oy. Nodeon Finland Oy on perustettu vuonna 2013 ja sen asiantuntijuus rakentuu neljän teknologipilarin varaan: ohjelmistosuunnittelu, tietoliikenneratkaisut, automaatio- ja sähkösuunnittelu. Nodeonin tarjoamat palvelut ulottuvat ratkaisujen alkuvaiheiden suunnittelusta ja konsultoinnista myös niiden elinkaaren hallintaan. (Lyhyesti n.d.)

Opinnäytetyön aihe sai alkunsa osana erästä hanketta, jossa Nodeon oli mukana monen muun yrityksen kanssa. Hankkeen tavoitteena oli kehittää ja tutkia autonomisiin busseihin liittyviä ongelmia ja ratkaista ne. Aihe syntyi, kun hankkeessa tuli tehtäväksi selvittää, onko autonomisten bussien reiteillä häiriöitä, jotka voisivat vaikuttaa niiden reitteihin.

Opinnäytetyön tavoitteena oli toteuttaa itsenäinen palvelu, jonka avulla pystytään seuraamaan, vaikuttaako palvelulle lähetettyihin reitteihin mitään tieliikennehäiriöitä. Palvelun pääkohderyhmänä toimisi autonomiset bussit, mutta palvelua voisi käyttää mikä tahansa muukin palvelun vaatimukset täyttävä laite, joka tukee MQTT-viestintää. Työn tavoitteena oli myös se, että palvelun käyttäminen olisi mahdollisimman helppoa ja se toimisi julkaise/tilaa (eng. Publish/subscribe) -mallilla.

2 Tutkimus

2.1 Tutkimusongelma

Tutkimusongelmana oli toteuttaa julkaise/tilaa -mallilla toimiva palvelu, joka pystyy selvittämään, onko palvelulle lähetetyillä koordinaattipisteistä koostuvilla reiteillä häiriöitä vai ei. Työn suunnittelu vaati paljon eri teknologioiden tutkimista sekä palavereita siitä, miten palvelun pitäisi toimia.

2.2 Tutkimuskysymykset

Opinnäytetyössä ilmeni kolme selkeää tutkimuskysymystä. Ensimmäinen tutkimuskysymys oli työn tärkein ja myös haastavin. Miten toteuttaa palvelu, jolle käyttäjä voi lähettää oman reittinsä koordinaatit ja saada tiedon, mikäli jokin häiriö vaikuttaa hänen lähettämäänsä reittiin. Haasteen tähän

toi palvelulle asetettu vaatimus toimia julkaise/tilaa -mallilla, sillä julkaise/tilaa -mallissa julkaisuviesti ei saa takaisin vastausta toisin kuin esimerkiksi REST-kyselyt.

Toinen tutkimuskysymys liittyi häiriöiden keräämiseen. Mistä saadaan luotettavaa, ajantasaista ja tarkkaa häiriötietoa, jota voidaan hyödyntää palvelun toiminnassa. Häiriötietoja varten täytyi etsiä avoimia rajapintoja, jotka tarjoavat laadukasta häiriödataa. Häiriötiedon etsimisessä korostui varsinkin häiriön sijaintitietojen tarkkuus.

Jotta häiriöpalvelu täyttäisi sille asetetut vaatimukset, täytyy sen myös pystyä selvittämään, vaikuttaako häiriöt käyttäjien lähettämille reiteille. Kolmantena tutkimuskysymyksenä oli siis, miten selvittää vaikuttaako avoimista rajapinnoista saadut häiriöt käyttäjien lähettämille reiteille.

2.3 Tutkimuksellinen kehittämistyö

Opinnäytetyö toteutettiin tutkimuksellisena kehittämistyönä, sillä siinä yhdistyy konkreettinen kehittämistoiminta ja tutkimuksellinen lähestymistapa. Työssä myös edetään tutkimuksellisista kysymyksistä kohti itse konkreettista kehittämistoimintaa.

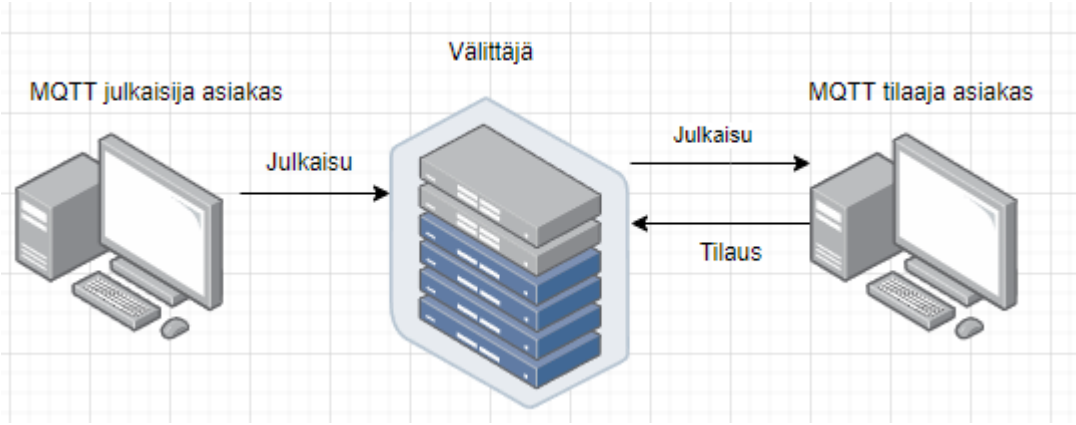
Opinnäytetyön toteutus toteutettiin viidessä eri vaiheessa. Ensimmäinen vaihe oli selvittää, mistä avoimista rajapinnoista saadaan tieliikenteen häiriödataa. Toinen vaihe oli toteuttaa laskuri, joka laskee, leikkaavatko häiriön pisteet reitin koordinaatit tai onko häiriö lähellä reittiä. Kolmas vaihe oli häiriöpalvelun MQTT:n kehittäminen. Neljäs vaihe oli toteuttaa minimaalinen REST-rajapinta ja viimeisenä vaiheena oli palvelun kontittaminen ja sen pystyttäminen Azuren pilvipalveluun.

3 MQTT-protokolla

MQTT on kevyt, yksinkertainen ja helposti käyttöön otettava julkaise/tilaa -malliin perustuva viestintäprotokolla. MQTT on erinomainen vaihtoehto tiedonsiirrossa sen minimaalisen pakettimäärän ansiosta. Sen yksinkertaisuus ja helppo käyttöönotto tekevät MQTT-protokollasta täydellisen valinnan IoT-laitteiden kanssa kommunikoimiselle. (MQTT Essentials: Part 1 Introducing the MQTT Protocol 2015.)

3.1 Julkaise/tilaa -malli

Perinteisessä asiakas-palvelin (eng. client-server) -mallissa asiakas (eng. client) kommunikoi suoraan palvelimen päätepisteen kanssa. Julkaise/tilaa -mallissa taas erotellaan viestin lähettäjä eli julkaisija ja tilaaja eli viestin vastaanottaja toisistaan, jolloin vältytään turhalta osoitteiden vaihtamiselta näiden välillä. Julkaisija tai tilaaja eivät edes tiedä toistensa olemassaolosta, vaan niiden välinen kommunikointi tapahtuu kolmannen komponentin eli välittäjän (eng. broker) avulla (ks. kuvio 1). (MQTT Essentials Part 2: Publish & Subscribe Basics 2015.)



Kuvio 1. Julkaise/tilaa -mallin arkkitehtuuri

Asiakas

Asiakkaalla tässä kontekstissa tarkoitetaan lähes aina MQTT-asiakasta, joka voi olla viestien julkaisija tai tilaaja. Asiakkaan tilaaja tai julkaisija rooli määräytyy sen toiminnan mukaan. MQTT-asiakkaana voi toimia mikä tahansa laite, joka tukee MQTT-protokollaa ja pystyy yhdistämään MQTT-välittäjään. Eli periaatteessa kaikkia laitteita, jotka pystyvät kommunikoimaan MQTT-protokollalla TCP/IP pinon yli voidaan kutsua MQTT-asiakkaiksi. (MQTT Essentials - Part 3: Client, Broker and Connection Establishment 2019.)

Välittäjä

Välittäjän tehtävänä on suodattaa kaikki julkaisijoiden lähettämät aiheet ja välittää niitä eteenpäin tilaajille. Välittäjä on minkä tahansa julkaisu-/tilausprotokollan ytimessä. Välittäjä pystyy käsitellä

jopa miljoonia yhdistettyjä asiakkaita samanaikaisesti. Viestien suodattamisen ja välittämisen lisäksi välittäjän tehtävänä on myös hoitaa asiakkaiden tunnistautuminen ja valtuuksien myöntäminen. (MQTT Essentials - Part 3: Client, Broker and Connection Establishment 2019.)

3.1.1 Viestin aihe (eng. topic)

MQTT-viestinnässä aiheella tarkoitetaan merkkijonoa, jonka avulla välittäjä suodattaa viestit yhdistetyille asiakkaille. Aihe koostuu yhdestä tai useammasta aihetasosta, jotka on eroteltu kauttaviivalla, esimerkki aihe ”valtio/kaupunki/häiriötyyppi” eli ”suomi/tampere/tieliikennehäiriö”. Aihe voi pitää sisällään myös villikortteja (eng. wildcard). Villikorttien avulla voidaan tilata monta aihetta kerralla. (MQTT Essentials - Part 5: MQTT Topics & Best Practices 2019.)

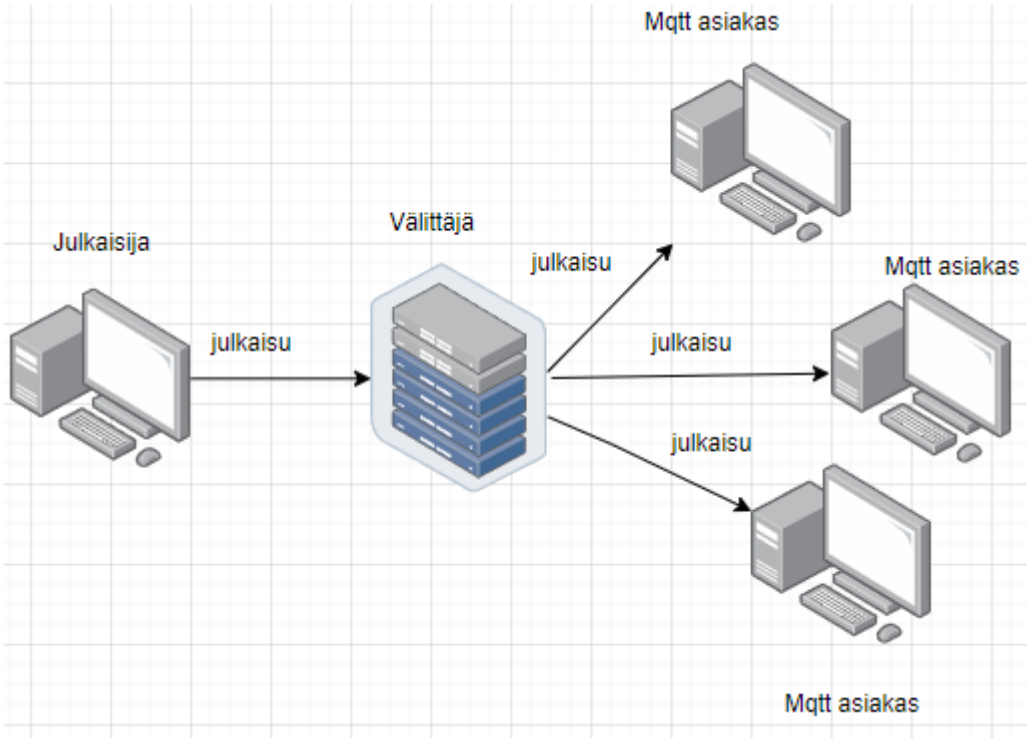
Villikortteja on kahdenlaisia: yksi- ja monitasoisia. Yksitasoisena villikorttina toimii ”+” merkki, jonka avulla voidaan korvata jokin aiheen aihetasoista. Tilaamalla aihe suomi/+/tieliikennehäiriö saataisiin siis tiedot kaikista suomen kaupungeissa olevista tieliikennehäiriöistä. (MQTT Essentials - Part 5: MQTT Topics & Best Practices 2019.)

Toisena villikorttina toimii monitasoinen ”#” merkki, joka korvaa loppuosan aiheesta. Eli tilaamalla aihe ”suomi/#” saataisiin tiedot kaikista viesteistä, jotka julkaistaan aiheeseen, jonka ensimmäisenä aihetasona on merkkijono suomi. Villikortteja voidaan käyttää aiheissa vain niitä tilattaessa, viestejä julkaistaessa niitä ei voi käyttää. (MQTT Essentials - Part 5: MQTT Topics & Best Practices 2019.)

3.1.2 Viestin julkaiseminen

Viestin julkaisemista varten täytyy asiakkaan olla yhteydessä välittäjään (ks. kuvio 2). Viestin täytyy pitää sisällään aihe, jota välittäjä käyttää viedessään viestejä eteenpäin tilaajille. Aiheen lisäksi julkaisuviestit sisältävät yleensä tietosisällön. Tietosisältö pitää sisällään viestin sisällön. Tietosisällössä on mahdollista lähettää kuvia, tai millä tahansa koodauksella salattua dataa eli käytännössä

se voi pitää sisällään mitä tahansa binäärimuotoista dataa. (MQTT Essentials - Part 4: MQTT Publish, Subscribe & Unsubscribe 2015.)



Kuvio 2. MQTT-viestin julkaiseminen

Viestin julkaisuun liittyy myös viestin laatutaso. Laatutason ollessa nolla viestin julkaisija ei saa kuittausta viestin toimituksesta, jolloin viestin julkaisu koostuu vain viestin lähettämisestä. Julkaisuviestissä laatutason ollessa suurempi kuin nolla, julkaisevaa asiakasta kiinnostaa vain viestin pääseminen välittäjälle asti. Viestin julkaiseva asiakas ei saa minkäänlaista palautetta siitä saiko kukaan hänen lähettämäänsä viestiä. (MQTT Essentials - Part 4: MQTT Publish, Subscribe & Unsubscribe 2015.) Viestin laatutasosta kerrotaan lisää luvussa 3.3.

Julkaisuviestissä muita tärkeitä olevia attribuutteja:

- **Paketin tunniste**

  Paketin tunnistetta käytetään julkaisuviestin yksilöimiseen. Paketin tunnistetta tarvitaan silloin, kun viestin laatutaso on suurempi kuin nolla. (MQTT Essentials - Part 4: MQTT Publish, Subscribe & Unsubscribe 2015.)

- **Pysyvä viesti (eng. retain flag)**

  Viestiä julkaistaessa voidaan viestille asettaa pysyvän viestin merkintä, jolloin välittäjä tallentaa viimeisimmän viestin tietylle aiheelle. Kun asiakas tilaa aiheen, jolle on lähetetty viesti käyttäen pysyvää viestiä, saa asiakas viestin kyseisestä aiheesta välittömästi. Pysyvän viestin avulla aiheen uudet tilaajat saavat heti viimeisemmän aiheeseen tulleen viestin, eivätkä he siten joudu odottamaan siihen asti, että uusi viesti julkaistaan. Ilman pysyvää viestiä, julkaistut viestit häviävät suoraan niiden julkaisun jälkeen. (MQTT Essentials - Part 4: MQTT Publish, Subscribe & Unsubscribe 2015.)

- **DUP merkintä**

  DUP merkinnän avulla voidaan ilmoittaa, että lähetetty viesti on jo kertaalleen lähetetty. Tämä merkintä lisätään viestiin silloin, kun käytetään vähintään laatutasoa yksi ja kun viestin lähettäjä ei ole saanut kuittausta viestin vastaanottajalta. (MQTT Essentials - Part 4: MQTT Publish, Subscribe & Unsubscribe 2015.)

3.1.3 Aiheen tilaaminen

Jotta viestien julkaisemisesta olisi jotain hyötyä, täytyy olla myös tilaajia. Vastaanottaakseen julkaisijoiden lähettämiä viestejä täytyy asiakkaiden tilata niitä aiheita, joihin julkaistaan viestejä. Tilaus viesti pitää sisällään paketin tunnisteen ja tilausluettelon. (MQTT Essentials - Part 4: MQTT Publish, Subscribe & Unsubscribe 2015.)

Paketin tunniste yksilöi viestin välittäjän ja tilaajan välillä ja tilausluettelo pitää sisällään kaikki tilaukset, jotka koostuvat aiheesta ja laatutasosta (MQTT Essentials - Part 4: MQTT Publish, Subscribe & Unsubscribe 2015). Aihetta tilattaessa on hyvä käyttää hyödyksi luvussa 3.2.3 esiteltyjä villikortteja, joiden avulla voidaan kuunnella montaa saman tyylistä aihetta tietyn aiheen sijaan.

3.2 Protokollan viestirakenne

MQTT-protokollan viestirakenne koostuu kolmesta eri osasta, joita ovat kiinteä otsake (eng. fixed header), muuttuva otsake (eng. variable header) ja tietosisältö (eng. payload). Protokollan toiminta perustuu näiden osien siirtelyyn. (MQTT Version 3.1.1 2014, 16.)

Taulukossa 1 kuvattu kiinteä otsake on pakollinen osa viestikehystä. Kiinteä otsake jakautuu kahteen tavuun. Ensimmäinen tavu myös jakautuu kahteen neljän bitin kokoiseen lohkoon, joista bitit 7–4 pitävät sisällään viestin tyypin ja loput 0–3 bittiä säilövät viestin lipukkeet (eng. Flags). Lipukkeiden avulla voidaan ilmoittaa viestin lisätietoja, esimerkiksi niiden avulla voidaan ilmoittaa, jos julkaisuviesti on pysyvä viesti. (MQTT Version 3.1.1 2014, 17.)

Kiinteän osakkeen toinen puolisko koostuu loppuviestin pituudesta. Pituuskentän koko vaihtelee yhden ja neljän tavun välillä, riippuen muuttuvan otsakkeen ja viestin tietosisällön pituudesta. (Understanding the MQTT Protocol Packet Structure 2021.)

Taulukko 1. Kiinteän otsakkeen rakenne (MQTT Version 3.1.1 2014, 16)

Bitit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Tavu 1	MQTT-viestin tyyppi				Viestille määritetyt lipukkeet			
Tavu 2…	Loppuviestin pituus							

Toisena viestikehyksen osana on muuttuva otsake. Toisin kuin kiinteä otsake, muuttuva otsake ei ole pakollinen kaikissa viesteissä. Sen sisältö vaihtelee viestin tyypin mukaan ja se on tarpeellinen vain silloin, kun viestin mukana halutaan lähettää lisätietoja. Esimerkiksi julkaisuviestit, joiden laatutaso on suurempi kuin nolla tarvitsevat tämän otsakkeen, jotta paketin tunniste lähtee viestin mukana. Taulukossa 2 on listattuna kaikki MQTT-viestityypit. Taulukossa on myös sarakkeet paketin tunniste ja tietosisältö, joiden soluihin on merkattu sisältääkö kyseinen viesti näitä attribuutteja

vai ei. Viestikehyksen viimeisenä osana on viestin tietosisältö, josta on kerrottu lisää luvussa 3.2.4. (MQTT Version 3.1.1 2014 19–21.)

Taulukko 2. MQTT-viestien tyypit ja sisältöä

Viestin tyyppi	Arvo	Paketin tunniste	tietosisältö
CONNECT	1	Ei ole	Pakollinen
CONNACK	2	Ei ole	Ei ole
PUBLISH	3	On, jos laatutaso > 0	Valinnainen
PUBACK	4	On	Ei ole
PUBREC	5	On	Ei ole
PUBREL	6	On	Ei ole
PUBCOMP	7	On	Ei ole
SUBSCRIBE	8	On	Pakollinen
SUBACK	9	On	Pakollinen
UNSUBSCRIBE	10	On	Pakollinen
UNSUBACK	11	On	Ei ole
PINGREQ	12	Ei ole	Ei ole
PINGRESP	13	Ei ole	Ei ole
DISCONNECT	14	Ei ole	Ei ole

3.3 Viestin laatutaso

Viestin laatutasolla tarkoitetaan sopimusta viestin lähettäjän ja vastaanottajan välillä. MQTT tarjoaa kolme eri laatutasoa, joita ovat: korkeitaan yksi (eng. At most once), ainakin yksi, (eng. At least once) ja tasan kerran (eng. exactly once). Laatutasoja miettiessä täytyy ottaa huomioon viestien toimituksen kaksi eri puolta: viestin toimitus julkaisijalta välittäjälle ja viestin toimitus välittäjältä tilaajalle. Laatutaso on yksi MQTT-protokollan tärkeimmistä ominaisuuksista, sillä se antaa

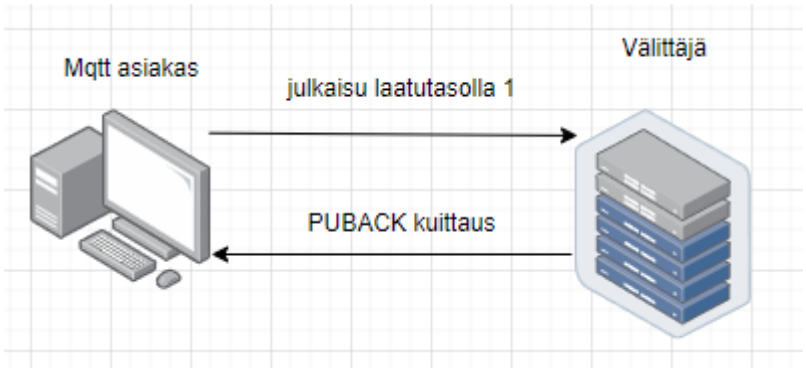
käyttäjälle vallan päättää milloin viestin tarvitsee aina päästä perille ja milloin ei. (MQTT Essentials - Part 6: Quality of Service 0,1 & 2 2015.)

3.3.1 Laatutaso 0

Laatutaso 0 toimii ammu ja unohda (eng. fire and forget) menetelmällä, eli viesti vain lähetetään eikä jäädä odottamaan mitään vastausta. Laatutasolla nolla viestin toimituksesta ei ole takuuta (at most once delivery). Laatutasoa nolla on hyvä käyttää silloin, kun yhteys on hyvä lähettäjän ja vastaanottajan välillä tai jos muutamien viestien hukkumisella ei ole väliä. (MQTT Essentials - Part 6: Quality of Service 0,1 & 2 2015.)

3.3.2 Laatutaso 1

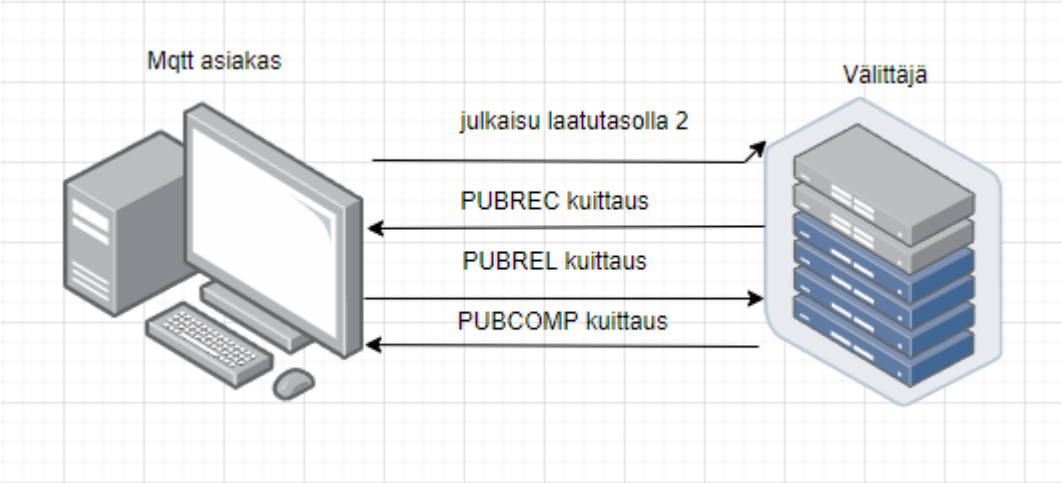
Laatutaso 1 lähettää viestin vähintään kerran (at least once delivery). Varmistaakseen viestin toimituksen, viestin lähettäjä säilyttää viestin, kunnes se saa takaisin PUBACK-paketin viestin lähettäjältä. Viesti on mahdollista lähettää ja vastaanottaa useita kertoja. Viestin lähettäjä käyttää viestin sisältämää pakettitunnistetta hyödyksi yhdistääkseen oikean julkaisuviestin oikeaan PUBACK-pakettiin. Mikäli viestin lähettäjä ei vastaanota PUBACK-pakettia tietyn ajan sisällä, lähettää lähettäjä tällöin uuden julkaisuviestin. Vastaanottajan saadessa viestin, joka on lähetetty laatutasolla 1 voi vastaanottaja tarvittaessa lähettää viestin heti eteenpäin ja vastata lähettäjälle PUBACK-paketilla (ks. kuvio 3). (MQTT Essentials - Part 6: Quality of Service 0,1 & 2 2015.)



Kuvio 3. Laatutaso 1 julkaisuviestissä

3.3.3 Laatutaso 2

Laatutaso 2 on MQTT-protokollan suurin laatutaso. Tämä laatutaso varmistaa, että viesti vastaanotetaan vain ja ainoastaan kerran (exactly once delivery). Tämä laatutaso on kaikista varmin, mutta myös samalla hitain. Viestin vastaanottamisen varmistaminen tapahtuu ainakin kahden pyyntö/vastaus (eng. request/response) -kättelyn avulla viestin lähettäjän ja vastaanottajan välillä (ks. kuvio 4). Viestin lähettäjä ja vastaanottaja käyttävät julkaisuviestin pakettitunnistetta koordinoidakseen paketin toimituksen. (MQTT Essentials - Part 6: Quality of Service 0,1 & 2 2015.)



Kuvio 4. Laatutaso 2 julkaisuviestissä

Viestin vastaanottajan saadessa julkaisuviestin vastaa se lähettäjälle PUBREC-paketilla, joka kuittaa julkaisuviestin. Mikäli viestin lähettäjä ei saa PUBREL-pakettia viestin vastaanottajalta, lähettää se tällöin lisää julkaisuviestejä, jotka on merkattu DUP-lipulla, kunnes se vastaanottaa kuittauksen. Kun viestin lähettäjä saa PUBREC-kuittauksen, voi se turvallisesti hylätä alkuperäisen julkaisuviestin ja tallentaa viestin vastaanottajan lähettämän PUBREC-paketin. Tallentamisen jälkeen viestin lähettäjä kuittaa PUBREC-paketin saamisen PUBREL-paketilla. (MQTT Essentials - Part 6: Quality of Service 0,1 & 2 2015.)

Viestin vastaanottajan saadessa PUBREL-paketin voi se huoletta hylätä kaikki säilytetyt tilat ja vastata lähettäjälle vielä kerran PUBCOMP-paketilla. Lähettäjän saadessa PUBCOMP paketti, vapautuu julkaisuviestin pakettitunniste takaisin käyttöön. Tämän monivaiheisen kättelyoperaation jälkeen molemmat osapuolet ovat varmoja siitä, että viesti on toimitettu. Mikäli julkaisuviesti häviää

matkan varrella, on viestin lähettäjä vastuussa viestin uudelleen lähettämisestä. (MQTT Essentials - Part 6: Quality of Service 0,1 & 2 2015.)

3.4 Välittäjään yhdistäminen

MQTT-protokollan toiminta pohjautuu TCP/IP pinon päälle, joten välittäjän ja yhdistävän asiakkaan täytyy toimia tämän pinon päällä. MQTT-yhteys on aina yhden asiakkaan ja välittäjän välinen yhteys. Asiakkaat eivät ole ikinä suorassa yhteydessä toistensa kanssa. Yhteyden muodostamiseksi asiakas lähettää välittäjälle CONNECT-viestin, johon välittäjä vastaa CONNACK-viestillä ja statuskoodilla. Mikäli CONNECT-viesti on muotoiltu väärin tai yhteyden avaamiseen kuluu liian kauan, niin tällöin välittäjä sulkee yhteyden. Yhteyden varmistuessa välittäjä pitää yhteyden auki siihen asti, kunnes yhteys katkeaa tai jos asiakas katkaisee yhteyden. (MQTT Essentials - Part 3: Client, Broker and Connection Establishment 2019.)

3.4.1 CONNECT-viesti

MQTT-asiakas voi lähettää vain yhden CONNECT-viestin saman verkkoyhteyden aikana. Mikäli välittäjä vastaanottaa toisen CONNECT-viestin, täytyy sitä kohdella protokollarikkomuksena ja katkaista yhteys asiakkaaseen. (MQTT Version 3.1.1 2014, 23.) Taulukossa 3 on kuvattu hyvä esimerkki CONNECT-viestin sisällöstä, jonka sisältöä on selitetty seuraavissa kappaleissa.

Asiakkaan tunnisteen avulla yksilöidään MQTT-asiakkaat, jotka yhdistävät välittäjään. Välittäjä käyttää tätä tunnistetta seuratakseen asiakkaiden tilaa. Tämän takia tunnisteen tulisi olla uniikki. Tunnisteen voi myös jättää tyhjäksi, mikäli asiakas ottaa puhtaan yhteyden (eng. clean session) välittäjään. (MQTT Essentials - Part 3: Client, Broker and Connection Establishment 2019.)

Puhtaalla yhteydellä tarkoitetaan yhteyttä, jossa välittäjä ei tallenna asiakkaan tilaamia aiheita eikä myöskään niihin tulleita viestejä, jotka ovat menneet asiakkaan ohitse. Puhdasta yhteyttä varten täytyy CONNECT-viestissä asettaa lipuke ”CleanSession” arvoon ”true”. CleanSession-lipukkeen ollessa ”false” asiakas voi jatkaa mahdollista aiempaa yhteyttä. Eli asiakkaan aiemmin tilaamat aiheet pysyvät tällöin voimassa. (MQTT Essentials - Part 3: Client, Broker and Connection Establishment 2019.)

CONNECT-viestissä tunnistautumista ja valtuuksien antamista varten voidaan määrittää asiakkaan käyttäjätunnus ja salasana. Näiden määrittely on jätetty toteutuskohtaiseksi. Ilman salausmenetelmiä salasana lähetetään selkeänä tekstinä, joten esimerkiksi TLS salausprotokollan käyttö tai VPN yhteyden luominen asiakkaan ja välittäjän välille on suositeltavaa salatun yhteyden luomiseksi. (MQTT Version 3.1.1 2014, 61.)

Viesti voi pitää sisällään myös niin sanotun viimeisen tahdon (eng. last will and testament) määrityksen. Viimeisen tahdon avulla asiakas kertoo välittäjälle, että mikäli hänen yhteytensä katkeaa, niin täytyy välittäjän lähettää viimeisen tahdon viesti asiakkaan määrittämään aiheeseen. (MQTT Essentials - Part 3: Client, Broker and Connection Establishment 2019.)

Asiakas voi myös määrittää viestin yhteyden tarkistusaikavälin (eng. KeepAlive). Tarkistusaikavälillä tarkoitetaan sekunneista koostuvaa pisintä aikajaksoa, jonka välittäjä ja asiakas voivat kestää lähettämättä viestejä. Varmistaakseen toistensa olemassaolon, sitoutuu asiakas lähettämään yhteyskokeilu (eng. ping) viestejä välittäjälle, jotka välittäjä kuittaa yhteyskokeilu vastauksella. (MQTT Essentials - Part 3: Client, Broker and Connection Establishment 2019.)

Taulukko 3. Esimerkki CONNECT-viestin sisällöstä

CONNECT-VIESTIN SISÄLTÖ	
Sisältö	**Esimerkki arvo**
Asiakkaan tunniste	”client-1”
Puhdas yhteys	false
Käyttäjätunnus	”Elmer”
Salasana	”salainen”
Viimeisen tahdon aihe	”häiriöpalvelu/status”
Viimeisen tahdon viestin sisältö	”offline”
Tarkistusaikaväli	60

3.4.2 CONNACK-viesti

Välittäjän saadessa CONNECT-viestin on se velvollinen vastaamaan viestin lähettäjälle CONNACK-viestillä. CONNACK-viestin sisältö koostuu kahdesta eri osasta, jotka on esitelty alla olevissa kappaleissa. (MQTT Version 3.1.1 2014, 31.)

Läsnä oleva istunto (eng. Session Present flag)

Tällä lipukkeella välittäjä ilmoittaa löytyykö, siltä aiempi asiakkaan tallennettu sessio. Jos yhteyttä on pyydetty puhtaan yhteyden CONNECT-viestillä, saa lipuke aina arvon ”false”. Ilman puhdasta

yhteyttä yhdistäessä lipukkeen arvo määräytyy sen mukaan, löytyykö välittäjältä vanhaa istuntoa vai ei. (MQTT Version 3.1.1 2014, 32.)

Yhteyden muodostuksen paluuarvot

Paluuarvoilla ilmoitetaan asiakkaalle, onnistuiko yhteyden muodostaminen vai ei. Onnistuneen yhdistämisen paluuarvo on 0 (MQTT Version 3.1.1 2014, 31–32). Muut mahdolliset arvot ovat esitettynä taulukossa 4.

Taulukko 4. CONNACK-viestin yhteyden muodostamisen paluuarvot

Arvo	Kuvaus
0	Yhteys muodostettu onnistuneesti
1	Yhteys hylätty, väärä protokolla versio
2	Yhteys hylätty, asiakkaan tunniste hylätty
3	Yhteys hylätty, ei yhteyttä välittäjään
4	Yhteys hylätty, väärä käyttäjätunnus/salasana
5	Yhteys hylätty, asiakkaalla ei ole valtuuksia yhteyden muodostamiseen
6–255	Varattu tulevaisuuden käyttöä varten

4 Muita käytettyjä teknologioita ja käsitteitä

4.1 .NET Core & ASP.NET Core

NET Core on Microsoftin ylläpitämä ilmainen avoimeen lähdekoodiin perustuva kehitysalusta. .NET:illä kehittäessä projektin tiedostot näyttävät ja tuntuvat samalta oli rakentamassa oleva sovellus minkä tyyppinen tahansa. .NET Core tukee monia ohjelmointikieliä mm. C#, F# ja myös Visual Basicia. Suositun alustan tästä tekee sen avoin lähdekoodi sekä Microsoftin tuoma tuki ylläpidolle. (What is .NET? 2022.)

ASP.NET Core on suosittu webbiohjelmistokehys, joka tarjoaa vakaan ja tuetun alustan sovelluksen käynnissä pitämiseksi. Se on myös Microsoftin tukema ja perustuu avoimeen lähdekoodiin, mikä tekee siitä suositun vaihtoehdon webbikehityksen parissa. (What is ASP.NET Core? N.d.)

MQTTNET

MQTTnet on tehokas .NET-kirjasto MQTT-pohjaiseen viestintään. Sen toteutus perustuu MQTT-dokumentointiin ja sen avulla MQTT-asiakkaan ja MQTT-välittäjän luonti ja niiden välinen kommunikointi on tehty helpoksi. (MQTTnet N.d.)

Luokka

Luokka (eng. class) on olio-ohjelmoinnin peruspilari. Luokan avulla voidaan luoda objekteja, jotka ovat instansseja kyseisestä luokasta. Luokka pitää sisällään sen sisältämät muuttujat ja metodit. (Luokka ja olio N.d.)

4.2 Docker

Docker on avoimen lähdekoodin suosituin konttialusta. Käytännössä Docker on kokonaisuus tuotteista, jotka helpottavat käyttäjiä Docker-konttien ja kuvien hallitsemisessa. Dockerin avulla voidaan helposti pakata sovelluksien muuttumattomia tiedostoja eli kuvia (eng. images) kontteihin. Docker-konttien hallintaan varten on myös olemassa työkalu nimeltä Docker Compose, jonka avulla pystytään hallita monia kontteja saman aikaisesti. (What is Docker? 2021.)

Konttien tehtävä on sitoa sovelluksen osat yhdeksi paketiksi. Luotu paketti sisältää kaikki sovelluksen riippuvuudet. Sovelluksen paketointi tekee tästä teknologiasta myös paljon kevyemmän kuin esimerkiksi virtuaalikoneen käyttäminen sovelluksen ajamiseen. Keveyden vuoksi myös sen skaalautuvuus on nopeaa. Kontin eristäessä sovelluksen osat yhteen pakettiin mahdollistetaan sovelluksen ajaminen missä tahansa ympäristössä. (Introduction to Containers N.d.)

Docker Hub

Docker Hub on maailman isoin varasto Docker-kuville. Hubista löytyy avoimen lähdekoodin ratkaisuja, sekä myös itse konttiyhteisön kehittäjien luomia Docker-kuvia. Palvelun käyttäjillä on vapaa pääsy muiden avoimiin tietosäiliöihin (eng. repository). Palveluun pystyy myös tallentamaan yksityisiä tietosäiliöitä. (Docker Hub N.d.)

4.3 Azure

Microsoft Azure on pilvipalvelualusta, johon kuuluu monia eri tuotteita ja palveluita, jotka auttavat palveluiden pystyttämisessä ja niiden ajamisessa. Azure tarjoaa myös paljon eri työkaluja, jotka auttavat hallitsemaan Azuressa pyöriviä sovelluksia. (What is Azure? N.d.) Opinnäytetyössä Azuren palveluista oli käytössä Azuren Container Instances.

Azure Container Instances

Azure Container Instances on yksi Azuren useista tarjoamista palveluista. Sen avulla mahdollisestaan kontitettujen sovelluksien ajaminen suoraan Azuren pilvipalvelussa**.** ACI tarjoaa perusominaisuudet konttiryhmien hallitsemiseen. Koska ACI tarjoaa suoran hallinnan kontteihin, ei käyttäjän tarvitse käyttää aikaa ja resursseja virtuaalikoneiden määrittämiseen. (What is Azure Container Instance? 2021.)

4.4 Muita tärkeitä käsitteitä

4.4.1 Ohjelmointirajapinta

Ohjelmointirajapinnalla eli API:lla tarkoitetaan tekniikkaa, joka mahdollistaa kahden eri sovelluksen välisen kommunikoinnin. API on kuin eräänlainen tunneli, jonka kautta eri sovellukset voivat

vaihtaa tietoja ja keskustella keskenään. Aina kun käytät sovelluksia, kuten Facebookia olet luultavasti tekemisissä APIn kanssa. (What is an API? N.d.)

4.4.2 REST

REST on arkkitehtuurimalli, joka on tarkoitettu ohjelmointirajapintojen toteuttamiseen. Tässä arkkitehtuurimallissa asiakas- ja palvelinpuoli voidaan tehdä itsenäisesti erilleen toisistaan. Eli kumpi tahansa puoli voi muokata kooditoteutustaan vaikuttamatta toisen toimintaan. Jotta asiakas ja palvelin voi kommunikoida keskenään, täytyy niiden tietää, missä muodossa viestit tulee lähettää. REST-rajapintaa käytettäessä eri asiakkaiden pyynnöt samoihin päätepisteisiin (eng. endpoint) suorittavat samat toiminnot ja myös palauttavat samat vastaukset. (What is REST? N.d.)

REST-arkkitehtuurimallissa kommunikointi perustuu asiakkaan lähettämiin pyyntöihin. Asiakas voi joko pyytää suoraa tietoa tai resurssien muokkausta. Palvelimen tehtävänä on toteuttaa asiakkaan pyynnöt ja vastata niihin. Asiakkaan pyyntöviesti koostuu yleisesti seuraavista alla olevista määritteistä. (What is REST? N.d.)

- **HTTP operaatiotyyppi**
  Määrittää toteutettavan operaation tyypin. Taulukossa 5 on kuvattu neljä yleisintä REST:in tarjoamaa operaatiotyyppiä.
- **Otsikko (eng. header)**
  Otsikon avulla lähetetään pyynnön lisätietoja, esim. minkä tyyppistä tietoa asiakas pystyy vastaanottamaan.
- **Polku (eng. path)**
  Polku määrittää haettavan palvelimen/resurssin sijainnin.
- **Viestin sisältö (eng. body)**
  Pitää sisällään viestin tietosisällön.

Taulukko 5. REST-arkkitehtuurimallin tärkeimmät operaatiotyypit

Operaation tyyppi	Toiminta
GET	Resurssien hakeminen
POST	Uuden resurssin luominen
PUT	Resurssin muokkaaminen
DELETE	Resurssin poistaminen

5 Toteutus

5.1 Käytetyt työkalut

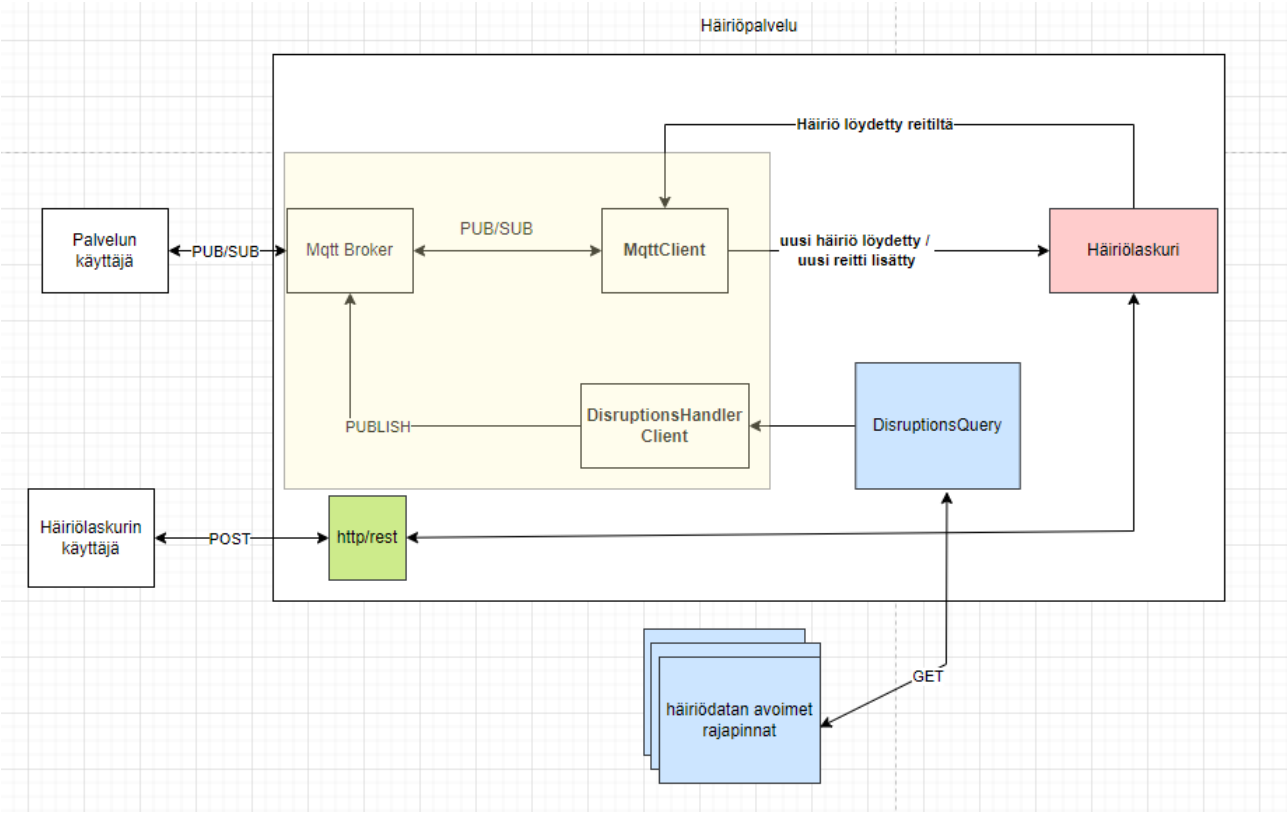
Työn toteutuksessa Visual Studio toimi valittuna ohjelmistoympäristönä, sillä sen avulla on helppo luoda tyhjästä hyvä .Net projektin pohja. Visual Studio myös helpottaa Docker-konttien kanssa toimimisessa. Versionhallintajärjestelmänä toimi Git sen yleisyyden ja helppokäyttöisyyden vuoksi.

Muita tärkeitä työkaluja, joista oli paljon hyötyä, olivat Docker Desktop ja MQTT Explorer. Docker desktop on sovellus, jonka avulla voidaan rakentaa ja jakaa kontitettuja sovelluksia (Docker Desktop overview N.d). MQTT Explorer on MQTT-asiakas, joka tarjoaa käyttöliittymä näkymän MQTT-aiheista ja sen avulla voidaan myös julkaista viestejä välittäjälle (MQTT Explorer N.d). Tämän sovelluksen avulla pystyy helposti tarkkailemaan palvelun MQTT-liikennettä.

5.2 Palvelun ohjelmistoarkkitehtuuri

Ennen palvelun toteuttamista, täytyi palvelusta laatia arkkitehtuurikuvaus, joka kuvaisi hyvin palvelun toimintaa. Palvelun ohjelmistoarkkitehtuuri koostuu pääosin neljästä eri osasta, jotka on kuviossa 5 eroteltu toisistaan eri värikoodein.

Keltaisella pohjavärillä merkityt laatikot kuvaavat palvelun MQTT-toimintaa. MQTT-toiminta koostuu MQTT-välittäjän ja kahden MQTT-asiakkaan toiminnasta. Näiden toiminnasta kerrotaan lisää luvussa 5.5. Vihreällä maalattu laatikko kuvaa palvelun REST-rajapintaa, josta kerrotaan lyhyesti luvussa 5.6. Punainen laatikko kuvaa palvelun häiriölaskuria, josta kerrotaan luvussa 5.4. Seuraavassa luvussa 5.3 käsitellään sinisiä laatikkoja, jotka liittyvät häiriöpalvelun häiriötietojen hakemiseen.



Kuvio 5. Häiriöpalvelun arkkitehtuurikuvaus

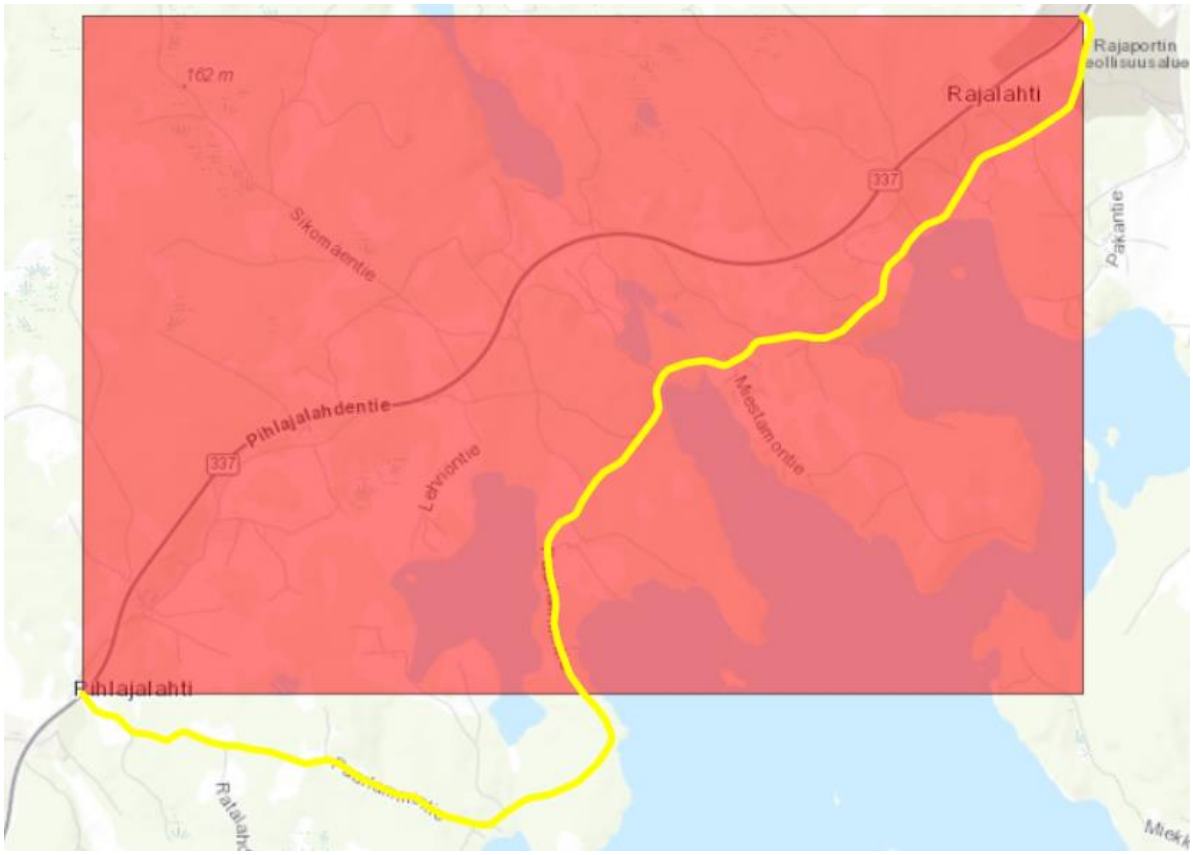
5.3 Häiriödatan kerääminen

5.3.1 Rajapintojen tarkastelu

Työn toteutus alkoi eri sivustojen/palveluiden tarjoamien avoimien rajapintojen tutkimisella. Tarkoituksena oli löytää avoimia rajapintoja, jotka tarjoavat mahdollisemman luotettavaa ja tarkkaa dataa liittyen Suomen tieliikennehäiriöihin.

Rajapintoja tutkiessa täytyi sen tarjoaman datan luotettavuuden lisäksi kiinnittää huomiota siihen, miten erilaiset häiriötilanteiden sijainnit ollaan rajapinnassa kuvattu. Moni eri rajapinnan tarjoaja tarjosi paljon laadukasta häiriödataa ja sen tarjoama data oli helposti muokattavissa mm. kaupungin tai häiriötyypin mukaan, mutta häiriön sijaintia ei ollut merkitty tarpeeksi tarkasti. Näissä rajapinnoissa häiriön sijainnit oli ilmoitettu käyttäen rajausaluetta (eng. bounding box). Rajausalueella tarkoitetaan laatikkoa, joka rakentuu kahdesta koordinaattipisteestä, tässä tapauksessa häiriön ensimmäisestä ja viimeisestä koordinaattipisteestä.

Rajapinnat, jotka käyttävät rajausaluetta sijainnin esittämistapana eivät sopineet häiriöpalvelulle, sillä rajausalueen merkkaama alue on todennäköisesti paljon isompi, kuin mitä oikean häiriön viemä pinta-ala on. Tämän vuoksi selvitys siitä vaikuttaako kyseinen häiriö ajoneuvon ajamaan reittiin olisi hyvin epätarkka. Hyväksyttyjen rajapintojen täytyi siis tarjota häiriön sijainti data koordinaattipisteinä, joka on paljon tarkempi, kuin rajausalue (ks. kuvio 6). Punainen alue kuvaa aluetta, jonka häiriö veisi, mikäli se olisi merkitty käyttäen rajausaluetta. Keltainen reitti taas kuvaa häiriön tarkan sijainnin, sillä se on merkitty käyttäen sen koordinaattipisteitä.



Kuvio 6. Rajausalue ja koordinaattipisteet

5.3.2 Rajapintojen käyttäminen

Palvelun häiriödatan ensimmäisiksi avoimiksi rajapinnoiksi valikoitui Digitrafficin ja Tampereen kaupungin tarjoamat rajapinnat. Digitraffic tarjoaa ajantasaista ja avointa tieliikenteenhäiriödataa koko suomen alueelta (Digitraffic N.d). Tampereen kaupungin rajapinta tarjoaa häiriö- ja tietyötiedotteita, jotka vaikuttavat tieliikenteen kulkuun (Tampereen kaupungin liikennetiedoterajapinta 2021).

Eri rajapinnat tarjoavat häiriötietoja erilaisessa muodossa, joten häiriöpalvelun voidakseen käyttää useita eri rajapintoja täytyi toteuttaa yleinen häiriöluokka (ks. liite 1). Rajapinnoista tiedon haku toteutettiin käyttäen REST-rajapinnan GET-toimintoa. Saatu data muunnettiin ensin datan alkuperäistä muotoa vastaavalle luokalle, jonka jälkeen se muutettiin vastaamaan yleistä häiriöluokkaa.

Näistä eri rajapintakyselyistä toteutettiin kääreluokka, jonka sisältämä ”RequestDisruption” funktio yhdistää kaikista eri rajapinnoista saadut häiriöt yhdeksi listaksi (ks. kuvio 7). Tämä funktio myös suodattaa listasta pois duplikaatti häiriöt ja ne häiriöt, joiden alkamisaika on tulevaisuudessa (ks. kuvio 8).

```
var combinedDisruptions = new List<DisruptionResponse>();
  var disruptionsTre = await disruptionsTreService.RequestDisruptions();
  var disruptionsDigiTraf =
    await disruptionsDigiTrafficService.RequestDisruptions();
  combinedDisruptions.AddRange(disruptionsTreBackUp);
  combinedDisruptions.AddRange(disruptionsDigiTraf);
```

Kuvio 7. Eri palveluista saatujen häiriöiden yhdistäminen

```
var filteredByTime = combinedDisruptions
    .Where(
      (disruption) =>
        disruption.TimeAndDuration.StartTime.CompareTo(
          DateTime.Now.AddHours(12)
        ) < 0
    )
    .ToList();
  // poistetaan häiriö mikäli kyseinen häiriön tunniste on jo listassa
 var filteredDisruptions = filteredByTime
    .GroupBy((x) => x.SituationId)
    .Select((x) => x.First())
    .ToList();
```

Kuvio 8. Häiriöiden suodatus

5.4 Häiriölaskuri

Häiriödatan onnistuneen saamisen jälkeen, alkoi selvitys siitä vaikuttaako tieliikenteenhäiriöt palvelulle lähetettyihin reitteihin. Tätä varten piti toteuttaa häiriölaskuri, joka vertailee häiriön koordinaattipisteitä palvelulle lähetettyihin koordinaattipisteisiin ja tutkii kahta asiaa näiden välillä, leikkaavatko reitit toisensa tai ovatko reittien pisteet lähellä toisiaan.

Leikkauspisteet

Häiriön ja ajoneuvon reitin leikkauspisteiden selvittämiseksi, täytyi toteuttaa funktio, joka käy läpi jokaisen häiriön koordinaattipisteet ja muodostaa näistä pisteistä janoja. Myös ajoneuvon reitin koordinaattipisteistä muodostetaan janoja. Nämä luodut janat lähetetään eteenpäin funktiolle, joka selvittää leikkaavatko janat toisensa. Jos leikkaus tapahtuu, niin selvitetään samalla niiden leikkauspisteet. Mikäli leikkausta ei tapahdu niin tällöin jatketaan uusien janojen vertailemista toisiinsa, kunnes kaikki muodostetut janat on vertailtu keskenään.

Lähin piste

Leikkauspisteiden selvittämisen lisäksi oli myös tutkittava, onko mikään häiriön koordinaattipisteistä lähellä reitin koordinaattipisteitä. Tämä tarkastus on tarpeellinen, kun häiriön sijainti koos-

tuu vain yhdestä pisteestä, jolloin leikkauspisteitä ei voida tutkia. Vertailemalla pisteiden etäisyyksiä, löydetään myös sellaiset häiriöt, jotka ovat hyvin lähellä ajoneuvon reitin kanssa, mutta eivät koskaan leikkaa toisiaan.

Lähimmän pisteen etsiminen on toteutettu lähes samalla tavalla, kuin ylempi leikkauspisteiden etsiminen. Tässä tavassa ajoneuvon reitin pisteistä muodostetaan janoja, joihin verrataan kaikkia häiriön pisteitä. Mikäli häiriön sijainti koostuu useammasta pisteestä niin tällöin vertailu tapahtuu myös jokaisen häiriön janan ja ajoneuvon koordinaattipisteen välillä.

5.5 MQTT-toteutus

Häiriöiden hakemisen ja häiriölaskurin toteutuksen jälkeen oli aika siirtyä palvelun MQTT-toteutukseen. MQTT-toteutuksen avulla mahdollistetaan asiakkaiden reittien välittäminen palvelulle ja myös häiriöiden välittäminen käyttäjille. Kuten palvelun arkkitehtuurikuvauksessa kuvattiin, on palvelun MQTT-toiminta pääosin jaettu kolmelle eri osalle.

5.5.1 Välittäjän toteutus

Palvelun välittäjä on toteutettu käyttäen hyödyksi MQTTNetin tarjoamaa valmista luokkakirjastoa MQTT-serverin pystyttämiseen (ks. kuvio 9). Välittäjälle on asetettu salausvarmenne, jonka avulla mahdollisestaan salainen yhteys välittäjän ja asiakkaan välillä. ”ConnectionValidatorin” avulla tarkastetaan asiakkaan oikeudet palvelun käyttämiseen. Häiriöpalvelun käyttöä varten täytyy asiakkaan tietää oikeat tunnukset.

Välittäjälle on myös asetettu muita tarkennuksia liittyen aiheisiin julkaisemiseen ja tilaamiseen. Palvelun omia MQTT-asiakkaita lukuun ottamatta, pystyvät asiakkaat tilata vain aiheita, jotka pitävät sisällään niiden oman tunnisteen. Näin estetään toisen asiakkaan tietojen saaminen. Myös reittien julkaiseminen on sallittu vain asiakkaan tunnisteen sisältämiin aiheisiin.

```
mqttServer = new MqttFactory().CreateMqttServer();
  var options = new MqttServerOptionsBuilder()
    .WithoutDefaultEndpoint()
    .WithEncryptedEndpoint()
    .WithEncryptedEndpointPort(config.Value.BrokerPort)
    // salausvarmenteen lisääminen
    .WithEncryptionCertificate(certificate.Export(X509ContentType.Pfx))
    .WithEncryptedEndpointBoundIPAddress(IPAddress.Parse(config.Value.BrokerAddress
))
    .WithEncryptionSslProtocol(SslProtocols.Tls12)
    .WithConnectionValidator(OnNewConnection)
    // aiheen tilaamisen rajoittaminen
    .WithSubscriptionInterceptor(checkSubPermissions)
    // aiheeseen julkaisemisen rajoittaminen
    .WithApplicationMessageInterceptor(checkPublishPermissions)
    .Build();
```

Kuvio 9. MQTT-välittäjän luominen

5.5.2 Palvelun MQTT-asiakkaat

Kuten luvussa 5.2 mainittiin, sisältyy palvelun MQTT-toimintaan välittäjän lisäksi myös kaksi MQTT-asiakasta. Palvelun toiminta perustuukin suurilta osin näiden kahden asiakkaan väliseen viestittelyyn välittäjän avulla.

Asiakkaiden luominen on toteutettu luomalla yleinen ”MqttClientConnection”-luokka, jonka alustusfunktiossa (eng. constructor) alustetaan asiakkaan luonti käyttäen hyödyksi MQTTNetin tarjoamaa ”MqttFactory”-luokkaa (ks. kuvio 10). Luotu luokka sisältää myös ”StartClient”-funktion, joka viimeistelee asiakkaan luomisen ja yhdistää välittäjään sille annettujen parametrien perusteella (ks. kuvio 11). Luomalla yleinen ”MqttClientConnection”-luokka säästytään turhalta saman koodin kirjoittamiselta ja näin uuden asiakkaan lisäämisestä tulee lähes vaivatonta (ks. kuvio 12).

```
public IManagedMqttClient mqttClient;
public MqttClientConnection()
{
    var factory = new MqttFactory();
    mqttClient = new MqttFactory().CreateManagedMqttClient();
    mqttClient.ConnectedHandler = new
MqttClientConnectedHandlerDelegate(OnConnected);
    mqttClient.DisconnectedHandler = new
MqttClientDisconnectedHandlerDelegate(OnDisconnected);
    mqttClient.ConnectingFailedHandler = new
ConnectingFailedHandlerDelegate(OnConnectingFailed);
}
```

Kuvio 10. MqttClientConnection luokan alustusfunktio

```
public async Task StartClient(string clientId, string address, int port)
{
    var options = new MqttClientOptionsBuilder()
       .WithClientId(clientId)
       .WithTcpServer(address, port)
       .Build();
    ManagedMqttClientOptions managedOptions = new ManagedMqttClientOptionsBuilder()
                            .WithAutoReconnectDelay(TimeSpan.FromSeconds(60))
                            .WithClientOptions(options)
                            .Build();
    await mqttClient.StartAsync(managedOptions);
}
```

Kuvio 11. Funktio, jolla alustetaan asiakkaan asetukset

```
var connection = new MqttClientConnection();
_mqttClient = connection.mqttClient;
_mqttClient.UseApplicationMessageReceivedHandler(OnMessageReceived);
_mqttClient.UseConnectedHandler(OnConnected);
Task.Run(async () => await connection.StartClient("DisruptionsHandlerClient",
config.Value.BrokerAddress, config.Value.BrokerPort));
```

Kuvio 12. MQTT-asiakkaan luominen

Häiriöiden tarkkailija-asiakas

Häiriöiden tarkkailija-asiakkaalla tarkoitetaan palvelun arkkitehtuurikuvauksessa keltaisella merkittyä laatikkoa ”DisruptionsHandlerClienttiä”. Kyseessä on toinen palvelun MQTT-asiakkaista, jonka tehtävänä on tarkkailla eri rajapinnoista saatuja häiriöitä muutoksien varalta.

Tarkkailija-asiakkaan luokalle on luotu funktio, joka kutsuu 10 sekunnin välein luvussa 5.3.2 mainitun häiriöiden kääreluokan funktiota, joka palauttaa kaikki saadut häiriötiedot. Tarkkailijan ajaessa häiriöiden haku ensimmäistä kertaa, tallentaa se kaikki saamansa häiriöt sisäiseen muistiin. Tallentamisen lisäksi, julkaistaan jokainen häiriö omaan MQTT-aiheeseen (ks. kuvio 13). Julkaisuviesti käyttää laatutasoa 1, sillä häiriödata on tärkeää tietoa eikä se saa hukkua kuljetuksen aikana. Tarkkailija lähettää myös kaikki häiriöt yhtenä listana aiheeseen "häiriöt/kaikki".

```
var newMessage = messagebuilder
            .WithPayload(JsonConvert.SerialieObject(häiriö))
            .WithTopic($"häiriöt/maakunta/{}/häiriönTyyppi/{}/häiriönTunnus/{}")
            .WithAtLeastOnceQoS()
            .Build();
```

Kuvio 13. Häiriön julkaiseminen.

Ensimmäisen häiriönhaku kerran jälkeen alkaa tarkkailija vertailemaan tallentamaansa häiriödataa uudelleen haettuun häiriödataan. Vertailemisen tehtävänä on etsiä, onko rajapintoihin tullut uusia häiriöitä tai onko jokin häiriö poistunut. Jotta kuviossa 14 tapahtuva vertailu toimii, täytyy häiriödatan luokan periä ”IEquatable”-luokka (ks. Liite 1), jonka avulla mahdollistetaan objektien tarkka vertailu.

```
var latestDisruptions = await DisruptionsQueryWrapper.RequestDisruptions();
// poistetut häiriöt
var removedDisruptions =
savedDisruptions.Where(x => !latestDisruptions.Contains(x)).ToList();
// lisätyt häiriöt
var addedDisruptions =
latestDisruptions.Where(x => !savedDisruptions.Contains(x)).ToList();
```

Kuvio 14. Lisättyjen ja poistettujen häiriöiden etsiminen

Muutoksen löytyessä päivitetään muistissa oleva häiriödata uudella tiedolla ja uudet mahdollisesti lisätyt häiriöt julkaistaan kuvion 13 mukaisesti omiin aiheisiinsa. Myös aiheen "häiriöt/kaikki" sisältö korvataan uusilla häiriöillä. Mikäli häiriöitä on poistunut, julkaistaan kuvion 15 mukainen viesti. Viestin tietosisältö koostuu kaikista poistuneista häiriöistä.

```
var newMessage = messageBuilder
.WithPayload(JsonConvert.SerializeObject(disruptions))
.WithTopic($"häiriöt/poistetut")
.WithAtLeastOnceQoS()
.Build();
```

Kuvio 15. Poistettujen häiriöiden julkaiseminen

Häiriöiden välittäjä-asiakas

Toinen palvelun omista MQTT-asiakkaista toimii vielä suuremmassa roolissa kuin tarkkailija-asiakas. Sen vastuulla on vastaanottaa kaikki tarkkailijan lähettämät häiriöt. Tämän asiakkaan tehtäviin kuuluu myös palvelua käyttävien asiakkaiden reitti- ja tunnustietojen säilyttäminen. Tästä asiakkaasta käytetään nimitystä välittäjä-asiakas.

Toisin kuin tarkkailija-asiakas, välittäjä-asiakas on viestien tilaaja ja julkaisija. Saadessaan yhteyden välittäjään tilaa asiakas heti tietyt aiheet (ks. kuvio 16). Villikortti tilauksien ansiosta asiakas saa ilmoituksen kaikista julkaisuviesteistä, jotka julkaistaan kyseisillä merkkijonoilla alkaviin aiheisiin.

```
_mqttClient.UseConnectedHandler(OnConnected);
        public async void OnConnected(MqttClientConnectedEventArgs obj)
       {
           await _mqttClient.SubscribeAsync("häiriöt/#");
           await _mqttClient.SubscribeAsync("asiakkaat/asiakas/#");
       }
```

Kuvio 16. Aiheiden tilaaminen yhteyden saamisen jälkeen

Asiakkaat aiheeseen saapuvat palvelua käyttävien asiakkaiden lähettämät ajoneuvon tiedot (ks. kuvio 17). Pakollisia arvoja tietosisällössä ovat kaikki muut paitsi maakunta ja häiriöiden tunnukset, sillä palvelu hoitaa häiriöiden tunnuksien löytämisen ja reitin maakunnan palvelu voi selvittää

lähetetyn reitin koordinaattien perusteella. Asiakkaan lähettämä tietosisältö tallennetaan välittäjä-asiakkaan sisäiseen muistiin. Samalla asiakkaan tunnuksella sallitaan vain viisi eri reitin tallennusta.

```
public class ClientPayload : ClientTopic
{
    public List<List<List<double>>> ReitinKoordinaatit
    public string? Maakunta
    public List<string>? HäiriöidenTunnukset}
}
public class ClientTopic
{
    public string AsiakkaanTunnus // (MQTT asiakkaan tunnus)
    public int ReitinTunnus
    public string VehicleId
}
```

Kuvio 17. Asiakkaat aiheeseen tulevan tietosisällön tyyppi

Ajoneuvon tunnuksen ja reitin tunnuksen avulla välittäjä-asiakas pystyy seuraamaan mitä reittiä mikäkin ajoneuvo ajaa, näin mahdollisestaan ajoneuvon reitin vaihtaminen. Eli mikäli palvelua käyttävä asiakas julkaisee viestin, joka pitää sisällään jo tallennetun ajoneuvon tunnuksen, mutta eri reitin tunnuksen, niin tällöin palvelu poistaa vanhan reitin tiedot ja päivittää ne uuden viestin mukaiseksi. Yhdellä ajoneuvon tunnuksella voi siis olla vain yksi reitti tarkkailussa.

Häiriöt aiheeseen saapuvat kaikki tarkkailija asiakkaan julkaisemat häiriöt. Uuden häiriö viestin saapuessa välittäjä-asiakas etsii kaikki tallentamansa reittitiedot ja suodattaa niistä jäljelle vain ne, joiden maakunta on sama kuin saapuneen häiriön. Jäljelle jääneet reittitiedot ja itse häiriö lähetetään eteenpäin luvussa 5.4 esitellylle häiriölaskurille, joka selvittää vaikuttaako uusi häiriö tallennettuihin reitteihin.

Välittäjä-asiakas tallentaa sille saapuneet häiriöt aiheesta ”häiriöt/kaikki”. Näitä tallennettuja häiriötä käytetään silloin kun palvelulle lähetetään uusi reitti. Tämä uusi reitti lähetään kaikkien samalla maakunnalla olevien häiriöiden kanssa häiriölaskurille. Häiriölaskurin löytäessä häiriö, julkaisee välittäjä-asiakas häiriön tiedot kuvion 18 mukaisella viestillä. Viestin tietosisältö pitää sisällään paljon tietoja häiriöstä ja myös reitin ja häiriön leikkaus/vaikutus pisteet (ks. liite 2). Häiriön löytyessä lisätään häiriön tunnus asiakkaasta tallennettuihin tietoihin.

```
var newMessage = messageBuilder
.WithPayload(JsonConvert.SerializeObject(disruption))
.WithTopic($"häiriö/asiakasTunnus/{}/reitinTunnus/{}/ajoneuvonTunnus/{}/maakunta/{
}/häiriöntunnus/{}")
.WithRetainFlag(true)
.WithAtLeastOnceQoS()
.Build();
```

Kuvio 18. Ajoneuvon reittiin vaikuttavan häiriön julkaiseminen

Vastaan ottaessaan viestin aiheesta ”häiriöt/poistetut”, tutkii välittäjä-asiakas tallentamiaan asiakkaiden reittitietoja ja etsii sisältääkö minkään reitin ”HäiriöidenTunnukset”-lista poistettujen häiriöiden tunnuksia. Mikäli asiakkaan reitiltä poistui jokin häiriö, muodostetaan kuvion 18 mukainen julkaisuviesti, mutta ilman tietosisältöä. Lähettämällä tyhjä tietosisältö käyttäen pysyvän viestin lipuketta saadaan poistettua pysyvä viesti välittäjän muistista (MQTT Version 3.1.1 2014, 34).

5.6 Palvelun REST-rajapinta

Palveluun luotiin myös minimaalinen REST-rajapinta, joka mahdollistaa palvelun häiriölaskurin käyttämisen. Häiriölaskurin käyttämistä varten täytyy käyttäjän lähettää palvelulle POST-pyyntö, joka pitää sisällään reitin koordinaatit. Pyyntö voi myös pitää sisällään reitin maakunnan, mutta se ei ole pakollinen.

Palvelun saadessa POST-pyyntö, lähettää se saamansa tiedot palvelun häiriölaskurille ja vastaa pyyntöön vastauksella (ks. kuvio 19). Mikäli maakunta puuttuu POST-pyynnön sisällöstä, selvittää palvelu tällöin reitin maakunnan sen ensimmäisien koordinaattien perusteella. Vastaus pitää sisällään, joko reitillä olevien häiriöiden tiedot tai tyhjän listan, joka kertoo, ettei häiriöitä löytynyt käyttäjän lähettämältä reitiltä.

```
e.MapPost("/request/disruptions", async (DisruptionsRequest body) =>
{
try
{
    var province = body.Province;
    if (province == null)
    {
        var provinceGetter = app.ApplicationServices.GetRequiredService<ProvinceGetter>();
        province = provinceGetter.GetProvinceFromCoords(body.Polyline[0][0]);
    }
    var storedDisruptions = services.GetService<DisruptionsHandlerClient>().disruptions.ToList();
    var disruptions = await DisruptionsCalculator.CalculateDistances(body.Polyline, province, storedDisruptions);

    return disruptions;
}
catch (Exception ex)
{
    Log.Logger.Error(ex.Message);
    return null;
}
});
```

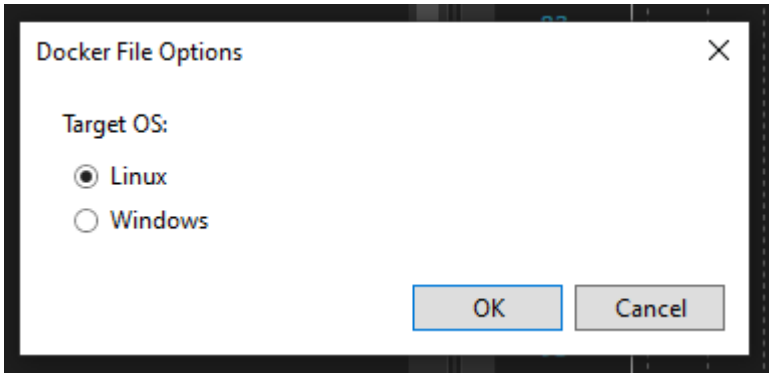
Kuvio 19. Palvelun REST-rajapinta

5.7 Palvelun pystyttäminen

5.7.1 Kontittaminen

Palvelun toiminnan ollessa valmis täytyi se myös saada pyörimään pilvipalveluun, jotta mahdolliset palvelua käyttävät asiakkaat pystyisivät käyttämään sitä. Palvelun pystyttämistä varten täytyi se ensimmäiseksi kontittaa. Kontitettuna palvelun pyörittäminen on paljon kevyempää ja myös halvempaa kuin esimerkiksi virtuaalikoneella pyörittäminen.

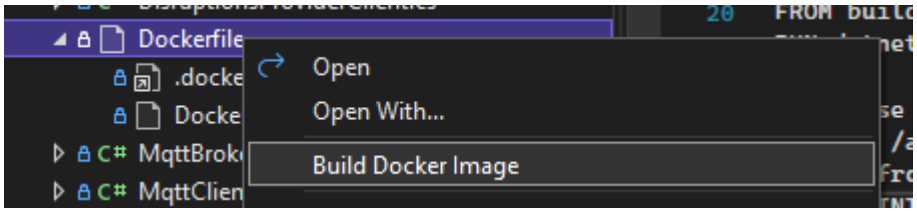
Palvelun kontittamisen ensimmäinen vaihe oli lisätä Visual Studiossa olevalle häiriöpalvelun projektille Docker-tuki. Kuvion 20 mukaisen näkymän saa päälle napsauttamalla hiiren kakkospainiketta kontitettavan projektin päällä ja valitsemalla valikosta Add > Docker Support. Ok painikkeen painamisen jälkeen Visual Studio luo projektiin ”Dockerfile” ja ”.dockerignore” nimiset tiedostot. (Visual Studio Container Tools for Docker 2022.)



Kuvio 20. Kontitus tuen lisääminen projektille

Dockerfile-tiedoston lisäämisen jälkeen, seuraava vaihe oli käyttää luotua tiedostoa ja rakentaa sen avulla Docker-kuva. Työssä tiedoston rakentaminen toteutettiin Visual studion ”Build Docker Image” komennon avulla (ks. kuvio 21). Docker-kuvan olisi voinut myös rakentaa kirjoittamalla terminaaliin seuraavan komennon:

```
docker build -f .\DisruptionsService\Dockerfile -t "häiriöpalvelu:latest" .
```



Kuvio 21. Docker Imagen rakentaminen Visual Studion avulla

Docker-kuvan rakentamisen jälkeen voi sen löytää komennolla:

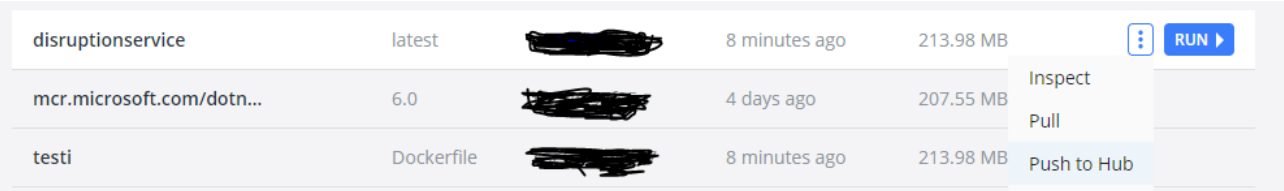
```
docker images –all
```

Tämä komento listaa kaikki paikallisesti löytyvät Docker-kuvat (ks. kuvio 22) (docker images. N.d).

```
C:\Users\Administrator>docker images --all
REPOSITORY                          TAG          IMAGE ID       CREATED          SIZE
disruptionservice                   latest       [redacted]     29 minutes ago   214MB
testi                               Dockerfile   [redacted]     29 minutes ago   214MB
mcr.microsoft.com/dotnet/aspnet     6.0          [redacted]     3 days ago       208MB
tuge12/disruptionservice            latest       [redacted]     2 months ago     215MB
```

Kuvio 22. Docker-kuvat listattuna

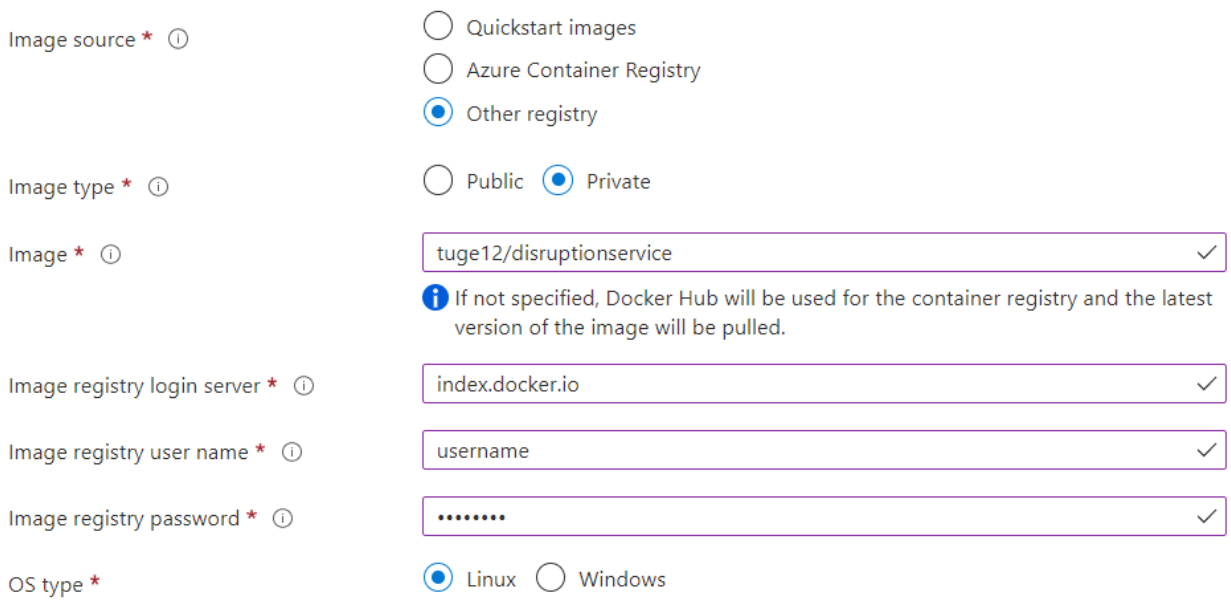
Opinnäytetyössä Docker-kuvien hallinnoimiseen käytettiin apuna Docker Desktop-sovellusta. Visual Studiolla luotu Docker-kuva puskettiin Docker Desktop sovelluksen avulla Docker Hubiin (ks. kuvio 23), jotta kuvan käyttöön ottaminen olisi helppoa Azuressa.



Kuvio 23. Docker Desktopissa kuvan puskeminen Docker Hubiin

5.7.2 Pystytys Azureen

Docker-kuvan ollessa Docker hubissa, oli viimeisenä tehtävänä saada se vielä pyörimään Azureen. Tätä varten täytyi Azureen luoda resurssi Container Instances. Resurssia luodessa täytyi, sille kertoa tulevan resurssin nimen lisäksi Docker kuvan sijainti ja muut tiedot sen saamiseksi (ks. kuvio 24). Tietojen antamisen jälkeen painettiin enää ”Review + create” painiketta. Onnistuneen tarkastuksen jälkeen painetaan vielä kerran ”create” painiketta, jonka jälkeen Azure luo kyseisen resurssin. Luomisen jälkeen resurssi eli kontitettu häiriöpalvelu on pystyssä ja käynnissä.



Kuvio 24. Docker-kuvan tiedot Azure Container Instancelle

5.8 Palvelun käyttäminen

Vaatimukset

Palvelun käyttöä varten täytyy käyttäjällä olla tiedossa välittäjään yhdistämistä varten tarvitsevat tunnukset. Käyttäjän tarvitsee myös tietää aiheet mihin julkaista reittitiedot ja myös aiheen mitä tilata. Reitin tietojen julkaisemista varten täytyy julkaisuviestin olla oikean tyyppistä (ks. kuvio 17). Toiveena olisi myös, että välittäjään yhdistäessä asiakkaalla olisi kuvion 25 mukaisen viimeisen tahdon määritykset. Näiden määritysten avulla välittäjä-asiakas saa tiedon, jos palvelua käyttävän asiakkaan yhteys katkeaa, jolloin välittäjä-asiakas voi poistaa kyseisen asiakkaan tiedot muistista.

```
var lw = new MqttApplicationMessage()
{
    Topic = "asiakkaat/asiakas/asiakkaanTunnus/poistettu",
    Payload = Encoding.UTF8.GetBytes("yhteys katkennut"),
    QualityOfServiceLevel = MQTTnet.Protocol.MqttQualityOfServiceLevel.AtLeastOnce,
    Retain = false
};
```

Kuvio 25. Yhdistysviestin viimeisen tahdon määritykset

Reitin julkaiseminen ja häiriötietojen tilaaminen

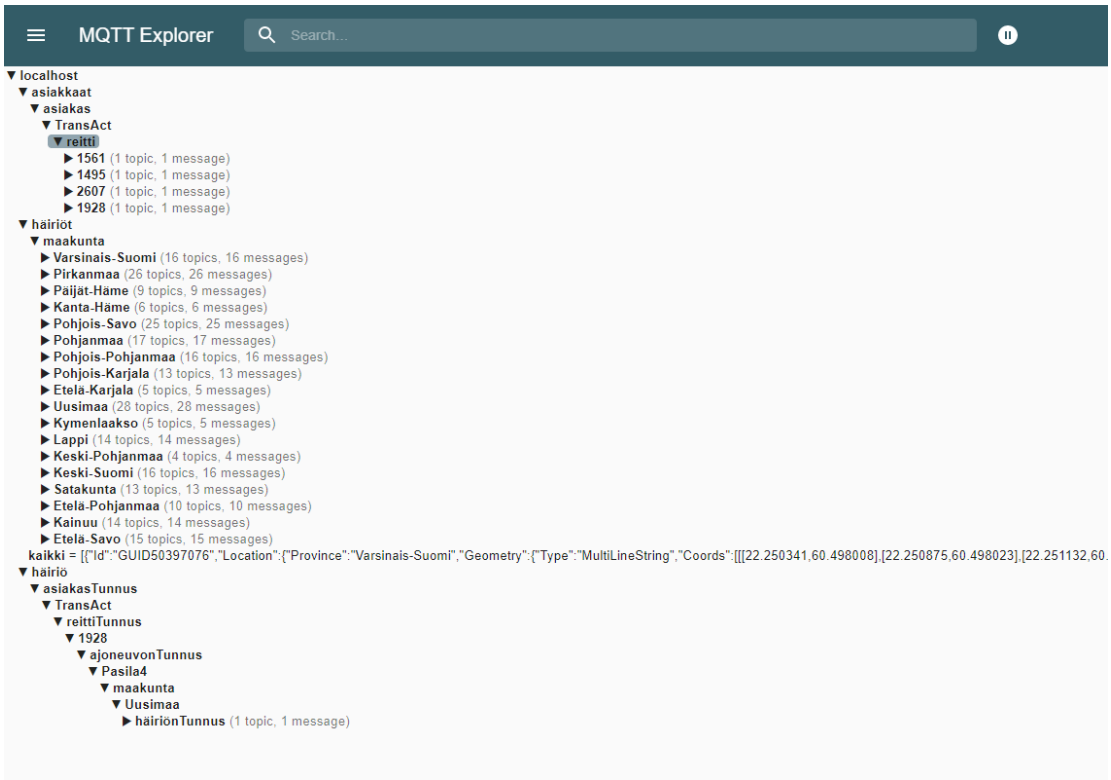
Reitti tietojen julkaiseminen tapahtuu aiheeseen “asiakkaat/asiakas/{asiakkaanTunnus}/reitti/{reitinTunnus}/ajoneuvo/{ajoneuvonTunnus}”. On tärkeää, että asiakas julkaisee viestit oman asiakas tunnuksen mukaiseen aiheeseen, sillä ilman samaa tunnusta julkaisut eivät pääse välittäjän tarkastuksen ohi. Saadakseen mahdolliset reiteillä olevat häiriöt täytyy asiakkaan vielä tilata aihe “häiriöReitillä/{asiakkaanTunnus}/#”.

6 Tulokset

6.1 Yleistä

Työn lopputuloksena syntyi toimiva tieliikennehäiriöpalvelu, jota voidaan käyttää selvittämään vaikuttaako mitään tieliikenteenhäiriöitä palvelulle lähetettyyn reittiin. Palvelu myös seuraa sille lähetettyjä reittejä uusien tieliikennehäiriöiden varalta. Palvelun löytäessä häiriö käyttäjän lähettämältä reitiltä, lähettää palvelu tällöin käyttäjälle häiriötiedotteen. Häiriötiedote pitää sisällään tärkeää tietoa häiriöstä, sekä myös koordinaattipisteet, jossa häiriö ja reitti leikaavat tai ovat lähellä toisiaan.

Kuviossa 26 on esiteltynä palveluun syntyneitä MQTT-aiheita, ja niiden tietosisältöä MQTT Explorerin avulla. Asiakkaat-aiheen alta löytyy kaikki palveluun yhdistäneet asiakkaat. Häiriöt-aiheen alapuolelta löytyy kaikki tarkkailija-asiakkaan löytämät häiriöt jaettuina omiin aiheisiinsa maakunnan mukaan. Häiriö-aihe koostuu asiakkaan lähettämästä reitistä, johon vaikuttaa, jokin palvelun löytämä häiriö. Aiheen ”häiriö/asiakasTunnus/TransAct/reittiTunnus/1928/ajoneuvonTunnus/Pasila4/maakunta/Uusimaa/häiriönTunnus/{tunnus}” sisältö on kuvattuna liitteessä 1.



Kuvio 26. MQTT Explorerin näkymä palvelun MQTT-aiheista

6.2 Valittujen teknologioiden soveltuvuus

MQTT:n avulla mahdollistettiin IoT-laitteiden kommunikointi palvelulle, joten myös palvelua käyttävät ajoneuvot voivat mahdollisesti julkaista niiden havaitsemat tieliikenteen häiriöt. Julkaise/tilaa -malli oli myös toimiva valinta palvelulle, jonka tarkoituksena on tarjota lähes reaaliaikaista dataa. Nämä aiemmin mainitsemani asiat tekivät MQTT-protokollasta hyvän valinnan häiriödatan välittämiselle palvelun ja ajoneuvon välille. Myös toimeksiantajalta tuli suosituksena käyttää MQTT-protokollaa sen yleistymisen takia.

Palvelun teknologia alustana toiminut .NET Core oli myös toimiva ratkaisu yhdessä .ASP net Coren kanssa. .NET:in tarjoamat kirjastot toivat paljon valmiita luokkia ja toimintoja, jotka nopeuttivat palvelun toteuttamista. Varsinkin MQTTNetin tuomasta tuesta MQTT:n kanssa toimimiselle oli merkittävä hyöty palvelun toteuttamisen kannalta. Myös palvelun pyörittäminen Azuressa Docker kontissa soveltui palvelun toiminnalle hyvin ilman ongelmia.

6.3 Haasteet

Tässä luvussa käsitellään työn toteutuksen eri vaiheissa syntyneitä haasteita ja niiden mahdollisia ratkaisuja. Haasteita työssä syntyi lähes sen jokaisessa toteutuksen vaiheessa.

Yksi työn suurimmista haasteista oli työn arkkitehtuurikuvauksen toteuttaminen. Haasteen tähän toi työssä käytetyt uudet teknologiat. Varsinkin MQTT:n ollessa itselleni täysin uusi teknologia täytyi siihen tutustumiseen käyttää paljon aikaa. Haastavin vaihe arkkitehtuurikuvauksessa oli MQTT-toteutuksen suunnittelu palvelun ulkopuolisen käyttäjän ja palvelun välillä.

Haasteita syntyi myös häiriödatan hakemisessa. Haasteena oli avoimien rajapintojen huono dokumentointi liittyen sieltä saatuun dataan. Esimerkiksi Digitrafficin tarjoaman häiriödatan sijaintityyppi vaihteli häiriön sijaintityypin mukaan. Tämä vaikeutti luokan tekemistä, joka vastaisi Digitrafficin tarjoamaa dataa. Ratkaisu tähän haasteeseen oli toteuttaa luokka, joka osaa lukea häiriön sijaintityypin ja sen perusteella luoda häiriön sijainnin tyypin (ks. kuvio 27).

```
public Geometry(JObject g)
{
    if ((string)g["type"] == "Point")
    {
        Type = (string)g["type"];
        CoordinatePoints = g.SelectToken("coordinates")?.ToObject<List<double>>();
    }
    else if ((string)g["type"] == "MultiLineString")
    {
     Type = (string)g["type"];
     CoordinateMultiLineString =
g.SelectToken("coordinates")?.ToObject<List<List<List<double>>>>();
    }
    else
    {
        CoordinateMultiLineString = new List<List<List<double>>>();
    }
}
```

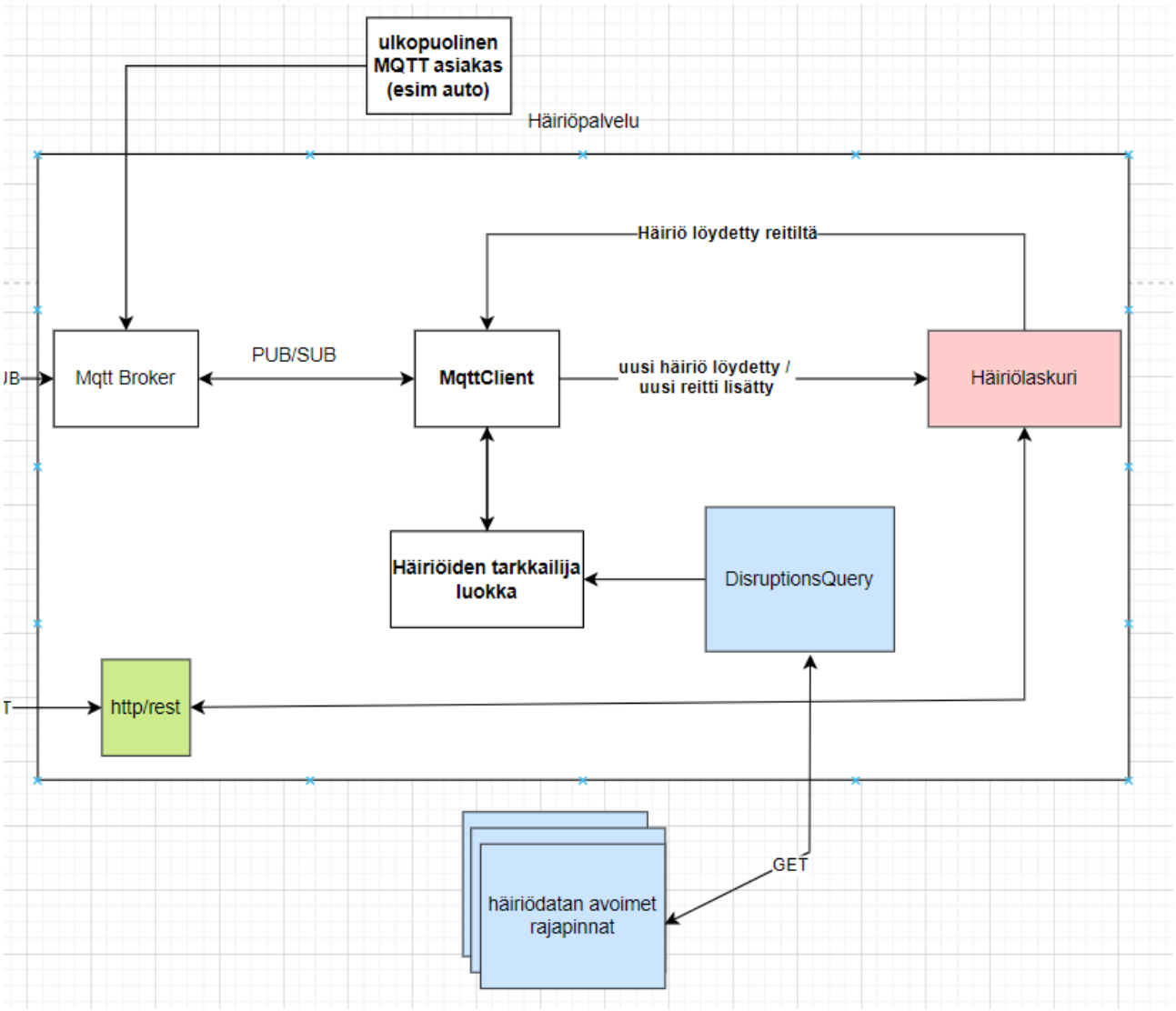
Kuvio 27. Digitrafficin tarjoaman häiriödatan sijaintityypin luominen

Häiriöiden löytäminen käyttäjien lähettämiltä reiteiltä osoittautui myös haasteeksi, sillä reittien ja häiriöiden pisteiden vertailun kanssa sai olla hyvin tarkka ja vertailun piti toimia varmasti. Vertailun toimimisen varmistamiseksi, aikaa kului paljon sen testailemiseen. Ensimmäisen häiriölaskuri version testaamisen jälkeen löytyi tilanne, jossa häiriölaskuri ei löytänyt reitin lähellä olevaa häiriötä. Tällaisessa tilanteessa reittipisteet olivat hyvin lähekkäin, mutta ne eivät silti ikinä leikanneet leikanneet toisiaan. Tällaisten tilanteiden löytämisen varmistamiseksi täytyi laskuriin siis lisätä vielä lähimmän pisteen tarkastelu.

6.4 Mahdolliset jatkokehitykset

Vaikka palvelu on työn vaatimuksien mukainen ja toimiva, löytyy sille silti paljon mahdollisia jatkokehityksiä. Jatkokehityksien avulla mahdollistetaan palvelun skaalautuminen suurille asiakas määrille. Jatkokehitysideoihin sisältyy myös päivityksiä, jotka tekevät häiriöpalvelun toiminnasta vielä paremman.

Yksi palvelun mahdollisista jatkokehitys kohteista olisi, muuttaa palvelun sisäinen häiriöiden välittäminen MQTT-välittäjän avulla, jaettuun häiriölistaan. Eli häiriöiden tarkkailija-asiakas muutettaisiin luokaksi, jonka tehtävänä olisi vieläkin seurata uusia häiriötä, mutta MQTT-julkaisuviestin sijaan, häiriölistan jakaminen tapahtuisi, vaikka funktio kutsun avulla (ks. kuvio 28).



Kuvio 28. Mahdollisen jatkokehityksen arkkitehtuurikuvaus

Asiakkaiden toimiessa samassa palvelussa, on niiden tietojen jakaminen MQTT-protokollan avulla hieman turhaa. Häiriöiden välittäminen tehtiin MQTT:n avulla, siksi että niiden tarjoaja voisi olla myös palvelun ulkopuolinen asiakas, ja koska yksi työn tavoitteista oli MQTT-toimintaan perehtyminen.

Jatkokehityksiä löytyy myös löydettyjen häiriöiden vaikuttavuuden arvioinnissa. Häiriöpalvelu ei ota kantaa esimerkiksi siihen, jos häiriö syntyy ajoneuvon takana tai jos häiriö koskee siltaa, jonka ali ajoneuvon reitti kulkee. Eli tällä hetkellä palvelun käyttäjä joutuu itse vielä tarkastamaan, onko häiriö tulossa vastaan vai onko se välttämättä edes samalla tiellä.

Tarkemman häiriötiedon selvittämistä varten täytyisi myös palvelua käyttävän asiakkaan tarjota lisätietoja palvelulle. Lisätietojen välittäminen vaatisi yhden lisävaiheen häiriön löytämisen ja häiriön julkaisemisen välille. Häiriön löytyessä ajoneuvon reitiltä, voisi palvelu julkaista viestin kyseisen ajoneuvon aiheeseen ”asiakasTunnus/ajoneuvo1/status”. Asiakkaan saadessa viesti tähän aiheeseen lähettää se ajoneuvon sen hetkiset koordinaatit takaisin palvelulle, jolloin palvelu tekisi vielä viimeiset tarkastelut reitille.

Palvelun skaalautumisen suuriin asiakasmääriin mahdollistamiseksi, olisi palvelulle hyvä myös lisätä tietokanta. Tietokantaan voitaisiin tallentaa kaikki häiriöt, sekä palvelua käyttävien asiakkaiden lähettämät tiedot. Tietokanta jätettiin pois palvelusta, koska pienissä asiakas ja häiriö määrissä sille ei ole tarvetta.

7 Pohdinta

Opinnäytetyön tavoitteiksi määriteltiin tieliikennehäiriöpalvelun toteuttaminen, joka pystyy seuraamaan sille lähetettyjä reittejä häiriöiden varalta ja tarvittaessa julkaisemaan MQTT-protokollan avulla häiriöviestin, jotta reitin lähettäjä voi saada ilmoituksen reitillä olevasta häiriöstä. Henkilökohtaisena tavoitteena oli myös kehittää omaa osaamista käytetyissä teknologioissa, varsinkin MQTT:n käytössä.

Ennen työn aloittamista, koodaus taitoni koostuivat pääosin käyttöliittymien kehittämisestä käyttäen JavaScriptiä. MQTT oli itselleni täysin uusi teknologia ja C# kokemus koostui pääosin yhdestä peliohjelmoinnin sekä olio-ohjelmoinnin perusteet kurssista. Nyt koen MQTT:n olevan hyvin hallussa, ja myös C#-ohjelmointikielestä tuli hyvin tuttu työtä tehdessä.

Työn MQTT toteutuksen aikana huomattiin suunnittelun tärkeys. Arkkitehtuurikuvauksen toteuttamisen jälkeen ei työn toteutuksen suunnitteluun varattu paljoa aikaa, vaan työ eteni toteutus

edellä. Tämän seurauksena, varsinkin palvelun MQTT-toiminnan kanssa joutui ottamaan usein takapakkia, sillä toteutettu tapa ei toiminutkaan halutulla tavalla. Selvän suunnitelman puuttuessa meni tekeminen helposti eri asioiden kokeilemiseen, jolloin yleensä syntyi mahdollisesti toimiva, mutta epävarma toteutus.

Opinnäytetyön aihe oli haastava, mutta erittäin opettavainen ja mielenkiintoinen. Mielenkiintoisen työstä teki sen käyttötarkoitus, sillä toteutetulle palvelulle on varmasti käyttöä. Työtä tehdessä pääsin myös tutustumaan syvemmin backend-kehittämiseen MQTT:n ja C# parissa, mikä on ollut myös omana tavoitteenani jo pitkään. Nodeonilla MQTT ja C# ovat suuressa käytössä, joten oli hienoa päästä tutustumaan näihin teknologioihin opinnäytetyön parissa.

Työn tavoitteet saavutettiin, sillä lopputulokseksi saatiin toimiva ja ajossa oleva palvelu, jolle käyttäjät voivat lähettää reittejään seurattavaksi. Myös henkilökohtaisissa tavoitteissa tapahtui onnistumisia, sillä työssä käytetyt teknologiat tulivat hyvin tutuiksi.

MQTT mahdollistaa hyvän ja helposti käyttöön otettavan protokollan IoT-laitteiden kanssa kommunikoimiselle. MQTT:n tarjoamat ominaisuudet kuten viestien laatutaso ja välittäjän mahdollisuus tallentaa julkaistuja viestejä tekevät protokollasta varman valinnan häiriöpalvelun toiminnalle.

Lähteet

Digitraffic. N.d. Digitrafficin verkkosivut. Viitattu 3.5.2022. https://www.digitraffic.fi/.

docker images. N.d. Dockerin dokumentit. Viitattu 21.4.2022. https://docs.docker.com/engine/reference/commandline/images/.

Docker Desktop overview. N.d. Dockerin dokumentit. Viitattu 28.4.2022. https://docs.docker.com/desktop/.

Docker Hub. N.d. Dockerin dokumentit. Viitattu 27.4.2022. https://www.docker.com/products/docker-hub/.

Docker images. N.d. Dockerin dokumentit. Viitattu 21.4.2022. https://docs.docker.com/engine/reference/commandline/images/.

Introduction to Containers. N.d. Fullstackopenin kurssisivut. Viitattu 25.4.2022. https://fullstackopen.com/en/part12/introduction_to_containers.

Luokka ja olio. N.d. Ohjelmoinnin MOOC verkkokurssi. Viitattu 24.4.2022. https://ohjelmointi-19.mooc.fi/osa-4/3-luokka-ja-olio.

Lyhyesti N.d. Nodeon Finland Oy yrityksen kotisivu. Viitattu 4.4.2022. https://www.nodeon.com/yritys/lyhyesti.

MQTT Essential - Part 1: Introducing the MQTT Protocol 2015. HiveMQ verkkosivuston MQTT-ohje. Viitattu 1.4.2022. https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt.

MQTT Essentials - Part 2: Publish & Subscribe Basics 2015. HiveMQ verkkosivuston MQTT-ohje. Viitattu 10.4.2022. https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/.

MQTT Essentials - Part 3: Client Broker and Connection Establishment 2019. HiveMQ verkkosivuston MQTT-ohje. Viitattu 10.4.2022. https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/.

MQTT Essentials - Part 4: MQTT Publish Subscribe & Unsubscribe 2015. HiveMQ verkkosivuston MQTT-ohje. Viitattu 24.4.2022. https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/.

MQTT Essentials - Part 5: MQTT Topics & Best Practices 2019. HiveMQ verkkosivuston MQTT-ohje. Viitattu 1.4.2022. http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices.

MQTT Essentials - Part 6: Quality of Service 01 & 2 2015. HiveMQ verkkosivuston MQTT-ohje. Viitattu 24.4.2022. https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/.

MQTT Explorer. N.d. MQTT Explorerin verkkosivu. Viitattu 24.4.2022. http://mqtt-explorer.com/.

MQTT Version 3.1.1 2014. OASIS standardi MQTT-protokollasta. Viitattu 7.4.2022. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf.

MQTTnet. N.d. MQTTnetin dokumentit. Viitattu 20.4.2022. https://github.com/dotnet/MQTTnet.

Tampereen kaupungin liikennetiedoterajapinta. 2021. Tampereen kaupungin tietopankki. Viitattu 2.5.2022. https://data.tampere.fi/data/fi/dataset/tampereen-kaupungin-liikennetiedoterajapinta.

Understanding the MQTT Protocol Packet Structure. 2021. Steve's internet Guide –verkkosivusto. Viitattu 10.4.2022. http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/.

Visual Studio Container Tools for Docker. 2022. Microsoftin dokumentit. Viitattu 21.4.2022. https://docs.microsoft.com/en-us/visualstudio/containers/overview?view=vs-2022.

What is .NET? 2022. Microsoftin dokumentit. Viitattu 10.4.2022. https://docs.microsoft.com/en-us/dotnet/core/introduction.

What is ASP.NET Core? N.d. Microsoftin dokumentit. Viitattu 10.4.2022. https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core.

What is Azure Container Instance? 2021. Cloud.netapp -verkkosivusto. Viitattu 25.4.2022 https://cloud.netapp.com/blog/azure-cvo-blg-azure-container-instance-aci-the-basics-and-a-quick-tutorial.

What is Azure? N.d. Microsoft Azuren verkkosivut. Viitattu 22.4.2022. https://azure.microsoft.com/en-gb/overview/what-is-azure/.

What is Docker? 2021. IBM verkkosivun dokumentointi Dockerista. Viitattu 15.3.2022. https://www.ibm.com/cloud/learn/docker.

What is REST? N.d. Codeacademyn artikkeli REST:istä. Viitattu 24.4.2022. https://www.codecademy.com/article/what-is-rest.

What is an API? N.d. Mulesoftin verkkosivusto. Viitatu 25.4.2022. https://www.mulesoft.com/resources/api/what-is-an-api.

Liitteet

Liite 1. Yleinen häiriöluokka

```
public class DisruptionResponse : IEquatable<DisruptionResponse>
    {
        public string Id { get; set; }
        public Location Location { get; set; }

        public TimeAndDuration TimeAndDuration { get; set; }

        public Annoucement Annoucement { get; set; }

        public string SituationId { get; set; }

        public string Source { get; set; }

        public bool Equals(DisruptionResponse other)
        {
            return other.Id == Id
                && Location.Equals(other.Location)
            && TimeAndDuration.Equals(other.TimeAndDuration)
            && Annoucement.Equals(other.Annoucement)
            && SituationId == other.SituationId;

        }
        public override bool Equals(object obj)
        {
            return obj is DisruptionResponse d && Equals(d);
        }

        public override int GetHashCode()
        {
            return Id.GetHashCode();
        }

    }
    public class TimeAndDuration
    {
        public DateTime StartTime { get; set; }
        public DateTime? EndTime { get; set; }
        public bool Equals(TimeAndDuration other)
        {
            return StartTime == other.StartTime && EndTime == other.EndTime;

        }
        public override bool Equals(object obj)
        {
            return obj is TimeAndDuration d && Equals(d);
        }

        public override int GetHashCode()
        {
            return StartTime.GetHashCode() ^ EndTime.GetHashCode();
        }
    }
    public class Annoucement
    {
        public string DisruptionType { get; set; }
        public string Severity { get; set; }
        public string Comment { get; set; }
```

```
    public string Title { get; set; }
    public bool Equals(Annoucement other)
    {
        return DisruptionType == other.DisruptionType
            && Severity == other.Severity
            && Comment == other.Comment
            && Title == other.Title;

    }
    public override bool Equals(object obj)
    {
        return obj is Annoucement d && Equals(d);
    }

    public override int GetHashCode()
    {
        return DisruptionType.GetHashCode() ^ Severity.GetHashCode();
    }

}
public class Location : IEquatable<Location>
{
    public string Province { get; set; }
    public Geometry Geometry { get; set; }

    public bool Equals(Location other)
    {
        return Province == other.Province
            &&
            Geometry.Equals(other.Geometry);
    }

    public override bool Equals(object obj)
    {
        return obj is Location d && Equals(d);
    }

    public override int GetHashCode()
    {
        return Province.GetHashCode() ^ this.Geometry.GetHashCode();
    }
}
public class Geometry
{
    public string Type { get; set; }
    public List<List<List<double>>> Coords { get; set; }
    public bool Equals(Geometry other)
    {
        return Type == other?.Type && CoordsEqualCalculator(other.Coords);
    }

    public override bool Equals(object obj)
    {
        return obj is Geometry d && Equals(d);
    }

    public override int GetHashCode()
    {
        return Type.GetHashCode() ^ this.Coords.GetHashCode();
    }

    bool CoordsEqualCalculator(List<List<List<double>>> comparable)
    {
        try
        {
```

```
            for (int i = 0; i < Coords[0].Count; i++)
            {
                for (int j = 0; j < Coords[0][i].Count; j++)
                {
                    bool isEqual = Coords[0][i][j] == comparable[0][i][j];
                    if (!isEqual)
                    {
                        return false;
                    }

                }
            }
        }
        catch { return true; }
        return true;
    }
}
```

Liite 2. Esimerkki reitiltä löydetyn häiriön aiheen tietosisällöstä


```
{
  "DisruptionPoints": [
    [
      24.804511505506895,
      60.27422112933745
    ]
  ],
  "ClosestRoutePointsToDisruptionPoints": [
    [
      24.804104658700957,
      60.27410621714753
    ]
  ],
  "RouteId": 1928,
  "Id": "GUID50381073",
  "Location": {
    "Province": "Uusimaa",
    "Geometry": {
      "Type": "MultiLineString",
      "Coords": [
        [
          [
            24.805502,
            60.271215
          ],
          [
            24.805414,
            60.271603
          ],
          [
            24.803747,
            60.276399
          ]
        ]
      ]
    }
  },
  "TimeAndDuration": {
    "StartTime": "2021-04-19T00:00:00Z",
    "EndTime": "2023-11-30T23:59:59.999Z"
  },
  "Annoucement": {
    "DisruptionType": "ROAD_WORK",
    "Severity": "",
    "Comment": null,
    "Title": "Tie 120, eli Vihdintie, Vantaa. Tietyö. "
  },
  "SituationId": "GUID50381073",
  "Source": null
}
```