jamk.fi

Ohjelmiston jakaminen mikropalveluihin

Jarko Miettinen

Opinnäytetyö
Marraskuu 2019
Tekniikan ala
Insinööri (AMK), Tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Miettinen, Jarko	Opinnäytetyö, AMK	Marraskuu 2019
	Sivumäärä 47	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Ohjelmiston jakaminen mikropalveluihin

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Esa Salmikangas

Toimeksiantaja(t)
Combitech Oy

Tiivistelmä

Tavoitteena oli tutkia Combitech Oy:n toimesta erilaisia tapoja jakaa ohjelmisto mikropalveluihin ja vertailla näiden palvelujakomallien eroja toisiinsa. Tämän vertailun pohjalta oli tarkoitus valita valmis palvelujakomalli tai luoda uusi malli, sekä arkkitehtuurikuvaus kalustorekisteriohjelmistoa varten.

Työ toteutettiin perehtymällä mikropalveluiden ja monoliittien välisiin eroihin sekä tutustumalla molempien mallien hyviin ja huonoihin puoliin. Mikropalveluiden osalta tutustuttiin myös erilaisiin tapoihin jakaa ohjelmisto mikropalveluihin ja vertailtiin näiden palvelujakomallien eroja ja soveltuvuutta kalustorekisteriin. Erilaisten tapojen osalta tutkittiin myös eri lähteistä ihmisten ajatuksia mikropalveluista ja monoliittisen ohjelmiston jakamisesta mikropalveluiksi.

Tulokseksi saatiin kattava, useisiin eri lähteisiin perustuva vertailu erilaisista tavoista soveltaa mikropalveluarkkitehtuuria, sekä palvelujakomalli ja arkkitehtuurikuvaus kalustorekisterille. Kalustorekisterille luotu palvelujakomalli ei sellaisenaan noudata yhtäkään opinnäytetyössä tutkittua palvelujakomallia, vaan siihen on pyritty valitsemaan parhaat puolet muutamasta eri mallista.

Mikropalveluarkkitehtuuri ei ole paras mahdollinen valinta jokaiselle ohjelmistolle, vaan joissakin tapauksissa monoliittinen ohjelmisto saattaa edelleen toimia paremmin. Samoin yksikään palvelujakomalli ei ole täydellinen valinta jokaiselle mikropalveluita hyödyntävälle ohjelmistolle. Ennen arkkitehtuurin ja palvelujaon valintaa ohjelmisto toiminnallisuuksineen tulisikin määritellä mahdollisimman tarkasti ja tehdä valinta sen mukaan, mikä soveltuu parhaiten juuri kyseessä olevalle ohjelmistolle.

Avainsanat (asiasanat)
Mikropalvelut, Monoliitti, Palvelujako, Ohjelmistoarkkitehtuuri

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Miettinen, Jarko	Bachelor's thesis	November 2019
		Language of publication: Finnish
	Number of pages: 47	Permission for web publication: x

Title of publication
Decomposing software to microservices

Degree programme
Information and communications technology

Supervisor(s)
Salmikangas, Esa

Assigned by
Combitech Oy

Abstract

The aim was to conduct a research for Combitech Oy and compare different ways to decompose software to microservices. The information gained on research was supposed to be a basis for choosing a decomposition model and architecture for an equipment registry software.

The thesis was carried out by researching on differences between microservice and monolithic architectures. There are multiple ways to decompose software to microservices, so these different decomposition models were compared at each other and each model's suitability for equipment registry was evaluated. During model comparison some research was also carried out using different sources on what people think about microservices and decomposing monolithic software to microservices.

As a result, there is a comprehensive comparison on different ways to utilize microservice architecture. Based on comparison and research, equipment registry got a model and architecture for decomposition, which was the goal of the thesis. Decomposition model chosen for equipment registry does not strictly follow any models that were researched, but instead it is a combination of best practices of different models.

Using microservice architecture is not the best choice for every software, and in some cases monolithic architecture might be a better solution. Same goes for decomposition models as none of the models is the best solution for every software utilizing microservices. Before making a choice between monolithic and microservice architecture and how to decompose software, the software and its features should be defined as accurately as possible and make the choice based on the needs and features of the software in case.

Keywords/tags (subjects)
Microservices, Monolith, Decomposition, Software architecture

Miscellaneous (Confidential information)

Sisältö

1 Johdanto 3

1.1 Toimeksiantajan esittely 3

1.2 Lähtökohdat 3

1.3 Tavoitteet 4

1.4 Tutkimusmenetelmä 4

2 Mikropalvelut 4

2.1 Mitä ovat mikropalvelut? 4

2.2 Mikropalveluarkkitehtuurin erot monoliittisiin ohjelmistoihin 5

2.2.1 Yleistä 5

2.2.2 Kehitystyö 6

2.2.3 Joustavuus 7

2.2.4 Skaalautuvuus 8

2.2.5 Luotettavuus 10

2.2.6 Soveltuvuus 11

3 Ohjelmiston jakaminen mikropalveluihin 13

3.1 Millainen on hyvä mikropalvelu? 13

3.2 Palveluiden määrittely 14

3.2.1 Palvelut luokkien mukaan 15

3.2.2 Liiketoimintakykyjen mukaiset palvelut 16

3.2.3 Rajoitetut kontekstit palveluina 17

3.2.4 Resurssien mukaiset palvelut 18

3.2.5 Käyttötapausten mukaan jakaminen 20

3.2.6 Palvelut toiminnallisuuksien mukaan 21

3.3 Mikropalveluiden keskinäinen kommunikointi 22

4 Kalustorekisteri 25

4.1 Kalustorekisterin vaatimukset 25

4.2 Mikropalveluiden valitseminen vaatimusten perusteella 26

4.3 Mikropalveluarkkitehtuuri 30

5 Analyysi 35

5.1 Yleistä 35

5.2 Mikropalveluarkkitehtuurin soveltuvuus kalustorekisteriin 37

5.3 Tulosten saavuttaminen 39

6 Pohdinta ja jatkokehitys 43

Lähteet 46

Kuviot

Kuvio 1 Monoliittinen ja mikropalvelu arkkitehtuuri 6

Kuvio 2 Skaalautuvuus eri akseleilla kuvattuna 9

Kuvio 3 Yhden mikropalvelun hajoaminen ei estä järjestelmän käyttöä 11

Kuvio 4 Monoliitin purkaminen mikropalveluiksi 12

Kuvio 5 Vain yksi mikropalvelu vastaa tietokannasta 14

Kuvio 6 Liiketoimintakyvyt palveluiksi 16

Kuvio 7 Ravintolajärjestelmän rajoitetut kontekstit ja näistä johdetut palvelut . 18

Kuvio 8 Resurssien pohjalta johdetut palvelut 19

Kuvio 9 Eurekan toiminta mikropalveluiden kanssa 23

Kuvio 10 Ribbonin toiminta mikropalveluympäristössä 24

Kuvio 11 Resurssit ja niistä vastaavat palvelut 27

Kuvio 12 Kalustorekisterin palvelujako 29

Kuvio 13 C4 System context tason kaavio 31

Kuvio 14 Container diagram tason kaavio 32

1 Johdanto

1.1 Toimeksiantajan esittely

Combitech on kansainvälisen Saab-konsernin osa, joka toimii Suomen lisäksi Ruotsissa, Norjassa ja Tanskassa ja työllistää yli 1900 henkeä. Combitech tarjoaa teknisen konsultoinnin lisäksi räätälöityjä ratkaisuja niin kotimaisille kuin kansainvälisillekin asiakkaille. Combitechin asiakaskuntaa Pohjoismaissa ovat puolustusvoimat, -teollisuus, yritykset ja julkinen sektori. (Tietoja meistä n.d.)

Suomessa Combitech toimii aktiivisesti kehittämässä suomalaista turvallisuus- ja puolustusteollisuutta. Combitech onkin tehnyt Puolustusvoimien kanssa yhteistyötä jo yli 30 vuotta. Yhteiskunnan puolustus on Combitechille tärkeä tehtävä, ja Combitech tukeekin puolustusvoimia toimittamalla viimeisimpiä teknologioita hyödyntäviä, taistelunkestäviä ja luotettavia tietojärjestelmiä. (Tietojärjestelmiä vaativiin tarpeisiin n.d.)

1.2 Lähtökohdat

Combitech Oy:lla oli olemassa kalustorekisteristä demoversio monoliittisena työpöytäohjelmistona, joka on toteutettu Qt kehitysympäristön avulla. Demoversioon on toteutettu kalustorekisterin perusominaisuudet kuten tiedon hakeminen, selaaminen ja muokkaaminen. Kalustorekisterille olisi kuitenkin enemmän tilausta nykyaikaisena web-toteutuksena, ja samalla olisi halu hyödyntää toteutuksessa mikropalveluarkkitehtuuria sen tuomine etuineen. Siirtymistä mikropalveluarkkitehtuuriin ei kuitenkaan kannata tehdä ilman aiheeseen perehtymistä ja mahdollisten vaihtoehtojen tutkimista, sillä palvelujaon huono suunnittelu saattaa tuoda enemmän ongelmia, kuin se ratkaisee.

1.3 Tavoitteet

Tavoitteena oli tutkia erilaisia tapoja jakaa monoliittinen sovellus mikropalveluiksi sekä vertailla erilaisten palvelujakoperusteiden hyviä ja huonoja puolia toisiinsa. Näiden vertailujen perusteella oli tarkoitus etsiä hyvät puolet erilaisista palvelujakomalleista ja luoda kalustorekisterille palvelujakomalli sekä arkkitehtuurikuvaus. Palvelujakoa tehdessä tavoitteena oli kyetä hyödyntämään mikropalveluarkkitehtuurin parhaita puolia kuten skaalautuvuutta ja sopivimman teknologian tapauskohtaista valintaa. Aikaansaatu arkkitehtuurikuvaus palvelujakomallineen tulisi toimimaan kalustorekisterin mikropalvelutoteutuksen pohjana, ja toisi Combitech Oy:lle arvokasta tietoa mikropalveluarkkitehtuurista ja sen erilaisista hyödyntämismahdollisuuksista myös muita projekteja varten.

1.4 Tutkimusmenetelmä

Tutkimusmenetelmänä opinnäytetyössä käytettiin soveltavaa tutkimusta. Tavoitteena oli luoda palvelujakomalli ja arkkitehtuurikuvaus kalustorekisterille käyttäen hyödyksi internetistä ja alan kirjallisuudesta löytyvää tietoa.

2 Mikropalvelut

2.1 Mitä ovat mikropalvelut?

IBM:n (Microservices 2019) mukaan mikropalvelut ovat pieniä, yleensä liiketoimintakykyjen ympärille määriteltyjä ohjelmistoja, jotka suorittavat yhtä tehtävää ja kommunikoivat keskenään rajapintojen avulla. Näistä pienistä palveluista voidaan koota suurempi, monia toiminnallisuuksia tarjoava järjestelmä. Mikropalveluita voidaan ajatellakin suuremman kokonaisuuden osina, joista jokainen täyttää oman tehtävänsä.

Mikropalveluille löytyy monia määritelmiä, mutta tärkeimpinä määrittelevinä tekijöinä ovat ylläpidettävyys ja testattavuus, vähäiset riippuvuudet ja se, että palvelut ovat käytettävissä itsenäisesti (Richardson n.d.b). Palveluita täytyy siis olla mahdollista käynnistää, sulkea, testata ja käyttää riippumatta muiden palveluiden tilasta.

Mikropalvelut eroavat monessa suhteessa perinteisemmästä tavasta luoda ohjelmistoja, joissa kaikki toiminnallisuudet keskitetään yhteen suureen ohjelmistoon, niin sanottuun monoliittiin, usean pienemmän ohjelmiston yhdistämisen sijaan. Mikropalveluarkkitehtuurin eroja monoliittisiin ohjelmistoihin hyvine ja huonoine puolineen vertaillaan tarkemmin luvussa 2.2

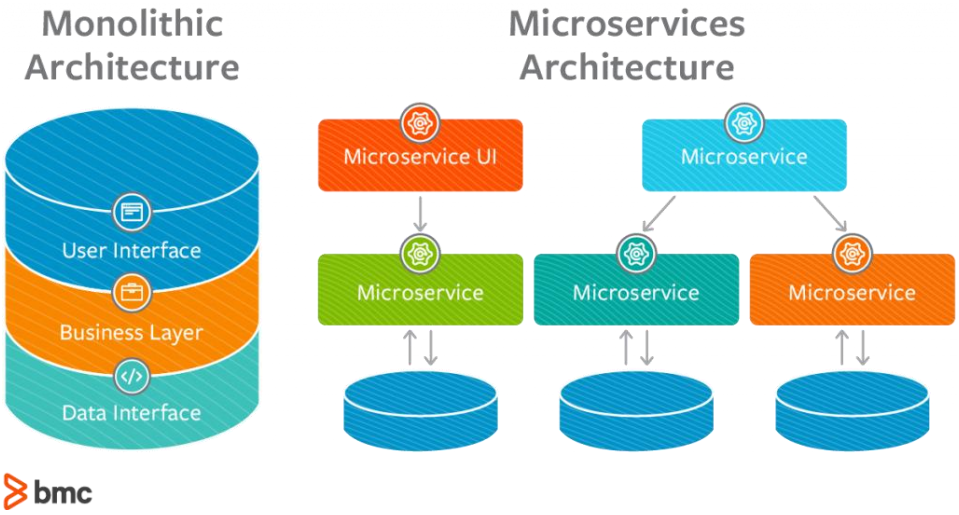
2.2 Mikropalveluarkkitehtuurin erot monoliittisiin ohjelmistoihin

2.2.1 Yleistä

Perinteisessä, monoliittisessa arkkitehtuurissa koko ohjelmisto kaikkine toiminnallisuuksineen rakennetaan yhdeksi yksiköksi, joka koostuu yleensä kolmesta osasta: tietokanta/tiedon hallinta, bisneslogiikka/palvelinkerros sekä käyttöliittymä. Nämä toimivat yhdessä niin, että bisneslogiikkakerros hoitaa käyttöliittymältä tulevat pyynnöt, pyytää tietoa tiedonhallinta kerrokselta ja lähettää mahdollisesti käsitellyn tiedon käyttöliittymälle, joka taas esittää tiedon käyttäjälle. (Microservices vs Monolithic Architecture n.d.)

Mikropalveluarkkitehtuurissa taas ohjelmisto koostuu kokoelmasta pieniä palveluita, joista jokaista ajetaan erillisenä prosessina ja jotka kommunikoivat keskenään kevyiden rajapintojen, esimerkiksi RESTin avulla. Nämä pienet palvelut rakennetaan liiketoimintakykyjen ympärille ja jokainen näistä palveluista on itsenäisesti käyttöönotettavissa. (Akhtar 2018.)

Käytännössä siis mikropalveluarkkitehtuuri perustuu ohjelmiston jakamiseen pienemmiksi itsenäisiksi yksiköiksi yhden suuren yksikön sijaan. Tätä havainnollistaa hyvin myös kuvio 1 arkkitehtuurimallien eroista.



Kuvio 1. Monoliittinen ja mikropalvelu arkkitehtuuri (Watts & Shiff 2018)

Ohjelmiston jakamisesta pienemmiksi palveluiksi on vaikutuksia moniin asioihin aina kehitystyöstä lopullisen ohjelmiston käytettävyyteen saakka verrattuna ohjelmiston tekemiseen yhtenäisenä yksikkönä.

2.2.2 Kehitystyö

Ohjelmistoa kehittäessä monoliittinen ohjelmisto voi muodostua niin suureksi, että yksittäiset kehittäjät eivät enää ymmärrä ohjelmiston kokonaiskuvaa (Microservices vs Monolithic Architecture n.d.). Mikropalveluarkkitehtuurissa taas palvelut pyritään jo nimensä mukaisesti pitämään pienenä ja palveluiden tarkoituksena on toteuttaa vain yhtä asiaa, jolloin palvelu ei muodostu yhtä helposti liian suureksi kokonaisuudeksi ymmärtää. Tämän myötä myös uudet kehittäjät pääsevät helpommin sisään kehitykseen mikropalveluiden kautta, sillä pienemmät kokonaisuudet ovat helpompia ymmärtää (Akhtar 2018.).

Vastapainona haasteita saattavat varsinkin uusille kehittäjille tuoda tilanteet, joissa tarvitaan useita mikropalveluita. Mikäli mikropalveluita on monia ja jokainen täytyy käynnistää erikseen, vie tämä aikaa ja on jopa turhauttavaa. Paras tapa olisikin jo kehitysvaiheessa luoda mahdollisuus käynnistää kaikki mikropalvelut tarvittaessa yksittäisten palveluiden käynnistämisen sijaan. (Barashkov 2018.)

Monoliittinen arkkitehtuuri onkin siis edelleen hyvä valinta varsinkin silloin, mikäli ohjelmisto ja/tai sitä kehittävä tiimi pysyy pienenä. Tällöin tiimiltä ei tuhlaannu ylimääräistä aikaa mikropalveluympäristön pystyttämiseen ja sen päivittäiseen käyttöön. Suuremmissa projekteissa taas mikropalvelun hyödyt ovat selvät kehitystyön kannalta.

2.2.3 Joustavuus

Akhtarin (2018) mukaan monoliittinen arkkitehtuuri ei ole joustavaa, sillä kaikki ohjelmiston toiminnallisuudet täytyy toteuttaa yhdellä ja samalla teknologialla. Tästä johtuen teknologian päivittäminen tai vaihtaminen on suurissa monoliittisissa ohjelmistoissa hankalaa. Mikropalveluarkkitehtuuri tarjoaakin joustavuutta päivittämiseen, sillä yksittäisiä mikropalveluita on helpompi päivittää kuin yhtä suurta kokonaisuutta. Mikropalveluiden kohdalla on myös mahdollista vaihtaa yksittäisen palvelun teknologia kokonaan toiseen ilman, että koko arkkitehtuuri ja ohjelmisto täytyisi tehdä alusta asti uudelleen.

Mikropalveluarkkitehtuurissa taas jokainen palvelu on oma yksittäinen ohjelmistonsa, joille jokaiselle voidaan valita tehtävään parhaiten soveltuva teknologia. Tämä antaa Guptan (2018) mukaan kehittäjille mahdollisuuden luovuuteen, sillä uusia ominaisuuksia on mahdollista lisätä ohjelmistoon heidän haluamallaan teknologialla sen sijaan, että valittu teknologia rajoittaisi heidän vaihtoehtojaan. Hänen mukaansa tämä antaa myös yksittäisille tiimeille enemmän itsenäisyyttä, sillä he eivät ole riippuvaisia muiden valinnoista, vaan voivat toteuttaa palveluita eri teknologioilla kuin muut tiimit.

Vaikka useiden teknologioiden hyödyntäminen saattaa äkkiseltään vaikuttaa vain hyvältä asialta, voi tämä tuoda mukanaan myös haasteita. Useita teknologioita käytettäessä ohjelmiston monimutkaisuus kasvaa selvästi, ja mikäli mikropalveluista vastaavat henkilöt vaihtuvat, kehittäjien täytyy kyetä käyttämään useampia teknologioita tarvittaessa. Vielä suuremmaksi tämä ongelma muodostuu siinä vaiheessa, kun jotakin tiettyä teknologiaa osaa käyttää vain muutama ihminen koko yrityksessä ja he

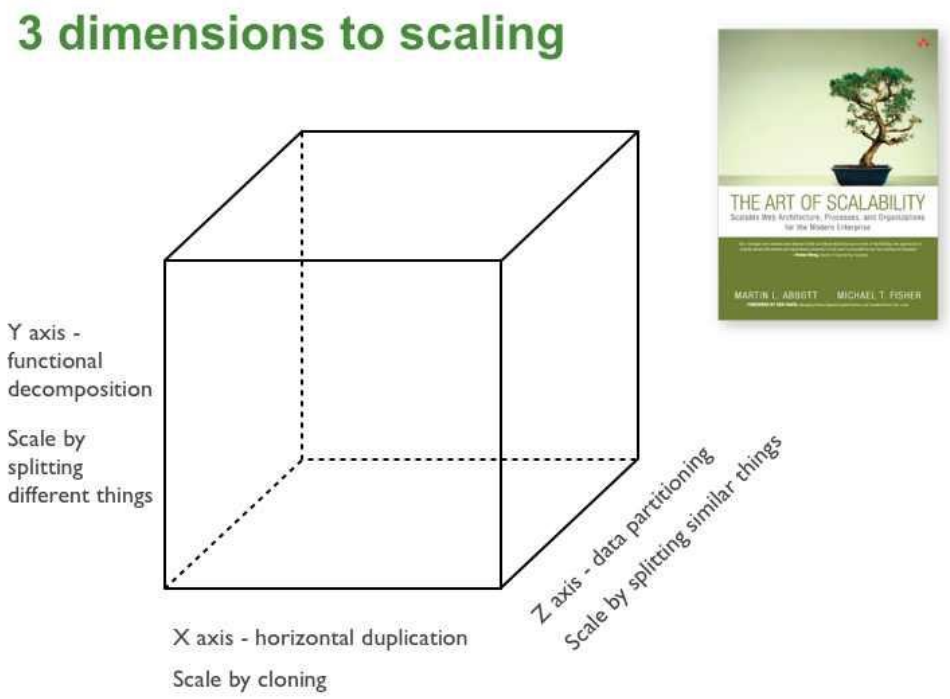
sattuvat vaihtamaan työpaikkaa. Tällöin uusien, kaikkia teknologioita valmiiksi osaavien kehittäjien löytäminen saattaa osoittautua mahdottomaksi ja tämä vaikuttaa myös yritykseen. (Jens 2017.)

Monoliittisissa ohjelmistoissa taas kaikki mukana olevat osaavat käyttää valittua teknologiaa ja uusien työntekijöiden löytäminen on helpompaa, mikäli heiltä ei vaadita usean tietyn teknologian hallitsemista. Useita teknologioita voidaan kuitenkin tarvittaessa käyttää mikropalveluarkkitehtuurissa, jolloin valinta niiden hyödyntämisestä jää kehittäjille. Monoliittisessa arkkitehtuurissa tätä valintaa ei edes ole.

2.2.4 Skaalautuvuus

Skaalautuvuudella tarkoitetaan ohjelmistomaailmassa ohjelmiston kykyä hallita kasvavaa käyttäjämäärää niin, että ohjelmisto ei kaadu tai hidastu merkittävästi (Swanberg 2019).

Skaalautuvuuden erojen hahmottamiseksi monoliittisen ja mikropalveluarkkitehtuurin välillä voidaan skaalautuvuus visualisoida kolmiulotteisen kuution avulla (ks. kuvio 2), jossa erilaiset skaalautumistavat on jaettu kuution kolmelle eri akselille.



Kuvio 2. Skaalautuvuus eri akseleilla kuvattuna (Richardson n.d.a)

X-akselilla tapahtuva skaalautuminen on yleisesti käytetty tapa ja se tarkoittaa yksinkertaisesti palvelua tarjoavien instanssien lisäämistä vastaamaan lisääntynyttä kuormaa. Tällöin ohjelmistolle saapuvien pyyntöjen ja ohjelmiston välissä on niin kutsuttu kuormantasaaja (load balancer), joka jakaa kuorman eri instansseille. Ongelmana tässä tavassa on se, että kaikilla instansseilla on pääsy samaan dataan, joten instanssit vaativat enemmän välimuistia toimiakseen tehokkaasti. (Richardon n.d.b.)

Y-akseli taas kuvastaa erinomaisesti sitä, mistä mikropalveluissa on kysymys, eli ohjelmiston jakaminen pienempiin yksiköihin. Tällöin kuormaa saadaan jaettua paremmin vain niille palveluille/toiminnallisuuksille, joille on suurempi kysyntä sen sijaan, että koko ohjelmisto kuormittuisi yhden osa-alueen kasvaneiden pyyntöjen takia. (Mt.)

Kuten Y-akseli, myös Z-akseli kuvastaa mikropalveluarkkitehtuurin etuja verrattuna monoliittiseen arkkitehtuuriin. Z-akselilla skaalattaessa jokainen instanssi on keske-

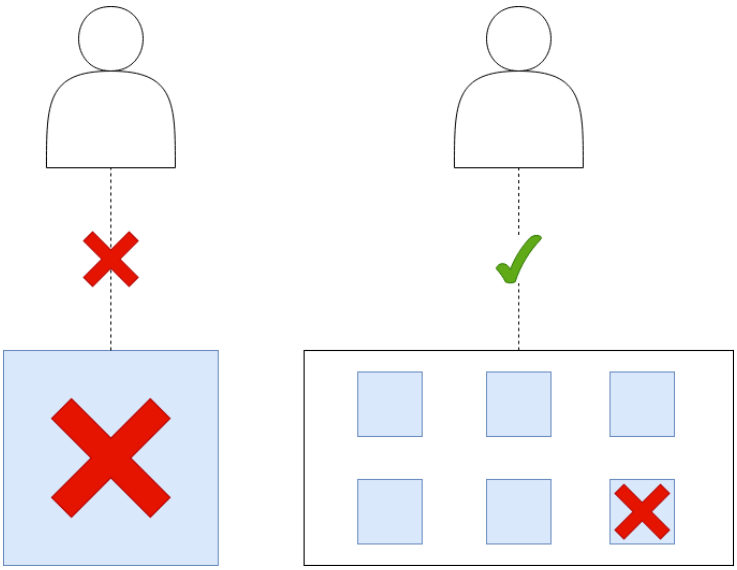
nään samanlainen toistensa kanssa aivan kuten X-akselilla, mutta erona X-akseliin jokainen instanssi vastaa vain tietystä datan osa-alueesta kaiken datan sijaan. Esimerkkinä tästä skaalautumisesta voidaan käyttää ohjelmistoa, joka tarjoaa asiakkailleen erilaisia palvelutasoja. Tällaisessa ohjelmistossa korkeamman palvelutason asiakkaat priorisoidaan enemmän kapasiteettia tarjoaville palvelimille kuin ilmaisasiakkaat, jolloin paremmalla palvelutasolla saa parempaa ja nopeampaa palvelua. (Mt.)

Skaalautumisen suhteen mikropalveluarkkitehtuuri tarjoaakin siis paljon enemmän vaihtoehtoja kuin monoliittinen vastine. Siinä missä monoliittinen ohjelmisto täytyy kopioida kokonaisuudessaan palvelutarpeen kasvaessa, voidaan mikropalveluita hyödyntävässä ohjelmistossa liikennettä ohjata tietyistä osa-alueista vastuussa oleville palveluille ja mahdollisesti lisätä näiden yksittäisten palveluiden määrää. Tämä säästää myös palvelutarjoajan resursseja, sillä on paljon vähemmän resursseja kuluttavaa lisätä pelkästään esimerkiksi tilauksista vastaavan palvelun instansseja kuin monistaa koko jättimäinen järjestelmä. Samalla tämä antaa myös mahdollisuuden tarjota asiakkaille eri tasoisia palveluita, jolloin asiakaskuntaa voidaan laajentaa useampiin asiakasryhmiin.

2.2.5 Luotettavuus

Koska mikropalvelut koostuvat useista pienistä ohjelmistoista yhden suuren sijaan, ovat mikropalveluarkkitehtuuria käyttävät järjestelmät lähtökohtaisesti monoliitteja luotettavampia. Mikäli mikropalveluita hyödyntävässä ohjelmistossa on palvelun kaatava vika yhdessä mikropalvelussa, ei tämä estä muiden mikropalveluiden ja näiden tarjoamien palveluiden ja toiminnallisuuksien käyttöä. Mikäli vastaava vika taas on jossakin päin monoliittista ohjelmistoa, estää se koko järjestelmän käytön. (Barashkov 2018.)

Tässä suhteessa mikropalveluarkkitehtuuri on siis paljon parempi vaihtoehto, sillä käyttäjät voivat kuitenkin käyttää yhä muita järjestelmän tarjoamia palveluita (ks. kuvio 3) sen sijaan, että koko järjestelmä olisi alhaalla niin kauan, kunnes vika on korjattu.



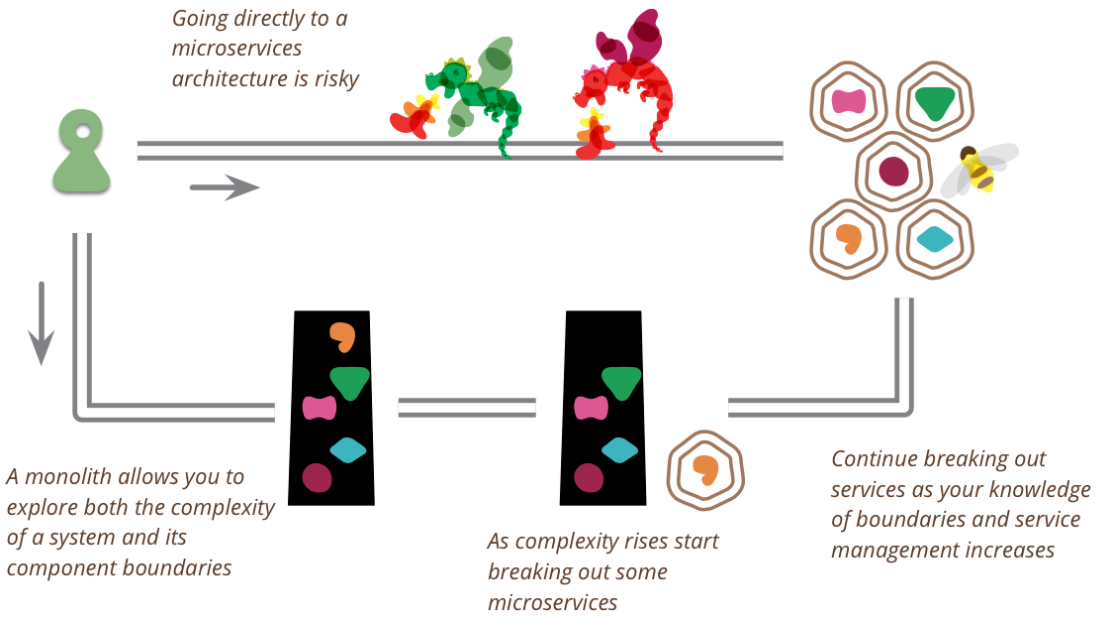
Kuvio 3. Yhden mikropalvelun hajoaminen ei estä järjestelmän käyttöä

Myös kehittäjän näkökulmasta vikojen rajoittuminen yksittäisiin palveluihin on vain ja ainoastaan hyvä asia. Vian paikantaminen on paljon hankalampaa, mikäli koko järjestelmä esimerkiksi kaatuu jo käynnistysvaiheessa. Kun taas vain yksi mikropalvelu kaatuu, on palvelussa jo lähtökohtaisestikin vähemmän koodia ja toiminnallisuuksia läpi käytävänä vikaa etsiessä.

2.2.6 Soveltuvuus

Vaikka teoriassa kumpi tahansa arkkitehtuurimalli voi toimia minkä tahansa ohjelmiston pohjana, on kummallekin oma paikkansa ohjelmistokehityksen maailmassa. Mikropalveluarkkitehtuuri tarjoaa monia ominaisuuksia, jotka helpottavat todella suurien järjestelmien ylläpitoa, kun taas monoliitti sopii yksinkertaisemman rakenteensa ansiosta paremmin pienille järjestelmille. Pienten järjestelmien kohdalla mikropalveluarkkitehtuurin käyttäminen lisää tarpeettomasti järjestelmän monimutkaisuutta ja ylimääräistä laskentatyötä, niin kutsuttua overheadia. (Wittmer 2018.)

Monoliittisena aloitetun ohjelmiston voi myös tarvittaessa pilkkoa mikropalveluiksi, mikäli ohjelmisto kasvaa suureksi. Tämä tapahtuu esimerkiksi tunnistamalla yksittäisiä osia monoliitista, ja luomalla näiden pohjalta yksittäisiä palveluita, kunnes monoliitti on purettu useaksi erilliseksi palveluksi (ks. kuvio 4). (Fowler 2015.)



Kuvio 4. Monoliitin purkaminen mikropalveluiksi (Fowler 2015)

Monoliittisella arkkitehtuurilla aloittamista ja sen muuntamista tarpeen vaatiessa mikropalveluiksi kutsutaan termillä Monolith first. Tämä lähestymistapa herättää kuitenkin reaktioita molempiin suuntiin ja esimerkiksi Tilkovin (2015) mukaan on parempi aloittaa suoraan mikropalveluarkkitehtuurista, mikäli sellainen on lopputuloksen tavoitteena. Hänen mukaansa tämä ohjaa kehittäjiä jo alusta asti ajattelemaan mikropalveluarkkitehtuurin vaativalla tavalla ja erottamaan ohjelmiston osuuksia toisistaan itsenäisiksi yksiköiksi. Monoliittista ohjelmistoa tehdessä ohjelmiston osat sidotaan yleensä kiinteästi toisiinsa, jolloin niiden erottaminen myöhemmin on työläämpää kuin se, että osat olisivat alun perin suunniteltu itsenäisiksi (mt.).

Arkkitehtuuria valitessa tulisikin pohtia jo alusta alkaen kuinka suureksi ohjelmisto voi kasvaa sekä kumpaan valintaan käytettävissä olevat resurssit sopivat paremmin.

3 Ohjelmiston jakaminen mikropalveluihin

3.1 Millainen on hyvä mikropalvelu?

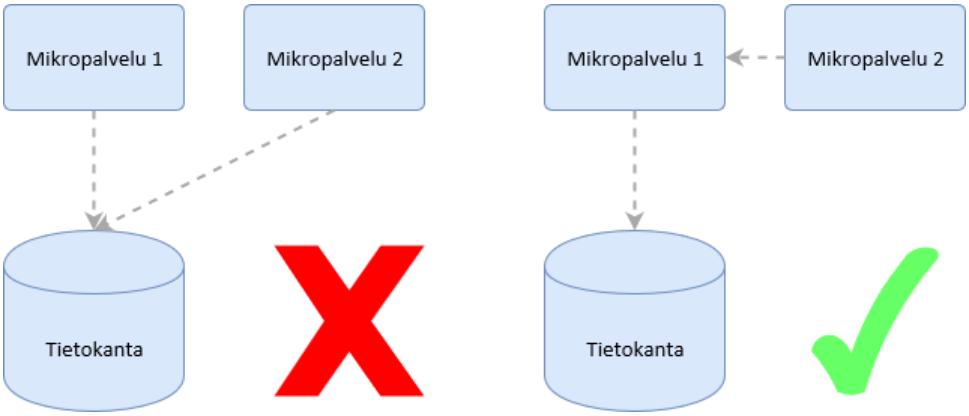
Richardsonin (n.d.b) mukaan mikropalveluiden tulisi olla erittäin ylläpidettäviä ja testattavia, löysästi toisiinsa kytkettyjä, itsenäisesti käynnistettäviä, organisoitu liiketoimintakykyjen ympärille ja pienten tiimien omistamia. Tiivistettynä voisikin sanoa, että hyvän mikropalvelun tulisi olla mahdollisimman itsenäinen ja muista riippumaton. Kuinka tämä itsenäisyys sitten saavutetaan niin, että järjestelmä toimii yhtenä kokonaisuutena usean itsenäisen palvelun tehdessä yhteistyötä keskenään?

Yksi lähtökohta itsenäisiin mikropalveluihin on Robert C. Martinin kehittämä yhden vastuun periaate, jonka mukaan kaikki samasta syystä muuttuvat asiat kerätään yhteen ja eri syistä muuttuvat asiat erotetaan toisistaan. Mikropalveluarkkitehtuuri käyttää tätä lähestymistapaa ja jatkaa sitä itsenäisesti kehitettäviin ja ylläpidettäviin palveluihin. Jokainen mikropalvelun palveluista vastaa erillisestä tehtävästä ja kommunikoi muiden palveluiden kanssa yksinkertaisten rajapintojen avulla ratkaistakseen suuremman ongelman. (Singh 2018.)

Itsenäisyys täytyykin siis ottaa huomioon jo suunnitteluvaiheessa ja pyrkiä pohtimaan edellä mainittua periaatetta, jotta palvelut olisivat mahdollisimman vähän riippuvaisia toisistaan. Itsenäisyyden kannalta on myös tärkeää pyrkiä piilottamaan kaikki monimutkaisuus ja toteutuksen yksityiskohdat ja paljastaa vain mitä mikropalvelun asiakkaat, olivat ne sitten ihmisiä tai muita mikropalveluita, vaativat. Jättämällä yksityiskohtia liian avoimeksi tulee mikropalvelun muuttaminen myöhemmin todella hankalaksi, sillä on hankala määritellä mikä palvelua käyttävä järjestelmän osa on riippuvainen mistäkin palvelun osasta. (Mt.)

Esimerkkinä itsenäisten mikropalveluiden yhdessä toimimisesta voidaan ajatella esimerkiksi tietokantaa, josta kaksi tai useampi mikropalvelu tarvitsee tietoa. Tällöin tietokannan toimintaan kohdistuvat muutokset kohdistuvat suoraan kaikkiin näihin pal-

veluihin, mikä lisää työtä ja vähentää palveluiden itsenäisyyttä. Tällöin parempi ratkaisu onkin määritellä yksi palvelu vastaamaan tietokannasta ja muut palvelut käyttämään tätä palvelua hyödykseen (ks. kuvio 5). Tällöin tietokannan käyttöä koskevat muutokset saadaan rajattua yhteen palveluun usean sijaan. (Mt.)



Kuvio 5. Vain yksi mikropalvelu vastaa tietokannasta

Mikropalvelu saattaa terminä herättää kysymyksen siitä, kuinka suuria tai pieniä mikropalveluiden kuuluisi olla. Vaikka mikropalvelu viittaa terminä pieneen kokoon, ei mikropalveluille ole määritelty mitään tiettyä kokoluokkaa. Morrisin (2015) mukaan mikro on harhaanjohtava osa sanassa mikropalvelu, sillä palvelut eivät välttämättä ole kooltaan pieniä, eikä mikropalvelun koolla itsessään ole väliä. Tärkeämpää mikropalveluissa on niiden kevyt infrastruktuuri, hajautettu hallintatapa ja suurempi painotus automaatioon. Morris (2015) jatkaa myös, että mikropalveluiden koon sijaan niiden ketterät puolet ovat se, minkä avulla ne erottautuvat muista järjestelmistä.

3.2 Palveluiden määrittely

Palveluiden määrittelyyn ja jakoperusteisiin löytyy lukuisia erilaisia mielipiteitä ja malleja, joista jokaisella on omat hyvät ja huonot puolensa. Miettinen (2019) käy läpi näistä malleista aiheeseen liittyvissä artikkeleissa useimmin mainittuja kertoen näiden puolista sekä tutkii niiden soveltuvuutta kalustorekisteriin.

Luvuissa 3.2.1-3.2.6 esitellään lyhyesti näitä mainittuja palvelujakoperusteita hyvine ja huonoine puolineen.

3.2.1 Palvelut luokkien mukaan

Luokat toimivat olio-ohjelmoinnissa eräänlaisina valmiina pohjina olioille. Luokat tarjoavat olioille niiden tarvitsemia vakioita, muuttujia sekä metodeita ja samasta luokasta luodut oliot jakavat samat ominaisuudet. (Bruce 2002.)

Luokat ovat usein pieniä osia ohjelmistossa, jotka toteuttavat omaa tehtäväänsä suuremman kokonaisuuden eteen, aivan kuten mikropalvelut, joten oletettavasti tästä syystä on syntynyt ajatus eriyttää jokainen luokka omaksi palvelukseen.

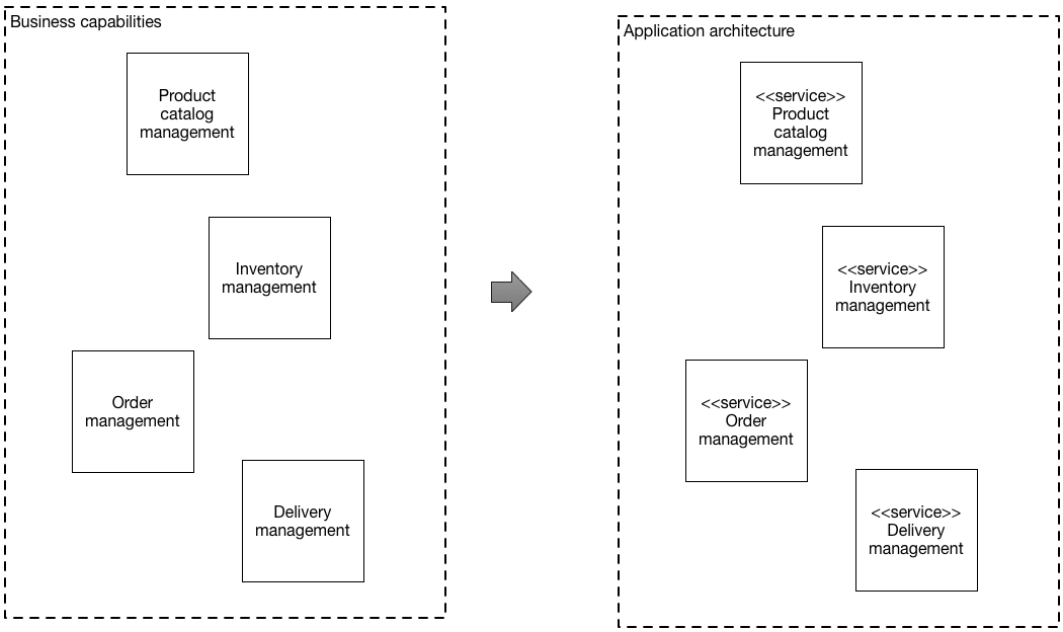
Miettinen (2019) kertoo palvelujakoraportissaan tämän mallin pohja-ajatuksena olevan se, että luomalla paljon itsenäisiä pieniä palveluita voidaan teoriassa saada aikaiseksi tilanne, jossa uusia ominaisuuksia voidaan ottaa käyttöön vain hyödyntämällä jo aikaisemmin luotuja palveluita.

Ongelmaksi tässä mallissa saattaa kuitenkin nopeasti muodostua palveluiden välisten yhteyksien kasvaminen, jotka kertautuessaan hidastavat järjestelmää verrattuna palveluihin, jotka sisältävät useamman luokan. Tällöin palvelun sisäisten luokkien väliset kutsut eivät tuo juurikaan lisää viivettä toisin kuin verkon välityksellä toisten palveluiden väliset kutsut. (Mt.)

Palveluiden määrän noustessa nopeasti suureksi myös koko järjestelmän monimutkaisuus kasvaa nopeasti. Jokaisen pienimmänkin osan ollessa oma palvelunsa tarvitsee näille palveluille määritellä omat rajapintansa sekä palveluiden välinen yhteistyö. Miettisen (2019) mukaan tämä malli saattaakin johtaa liialliseen suunnitteluun, jolloin mikropalvelusta saatavat hyödyt katoavat nopeasti kertautuvan monimutkaisuuden, viiveen ja vaaditun suunnittelutyön myötä.

3.2.2 Liiketoimintakykyjen mukaiset palvelut

Liiketoimintakykyjen mukaan palveluita jakaessa tunnistetaan ensin järjestelmän liiketoiminnalliset kyvyt, joiden pohjalta luodaan palvelut (ks. kuvio 6).



Kuvio 6. Liiketoimintakyvyt palveluiksi (Richardson n.d.a)

Tämän mallin etuna olisi se, että ohjelmiston liikekyvyt määrittelemällä näistä saataisiin johdettua suoraan palvelut ja mikäli järjestelmää tarjoavassa yhtiössä eri tiimit vastaavat eri liiketoimintakyvyistä, voivat nämä tiimit omistaa myös tähän kykyyn liittyvän palvelun. (Miettinen 2019.)

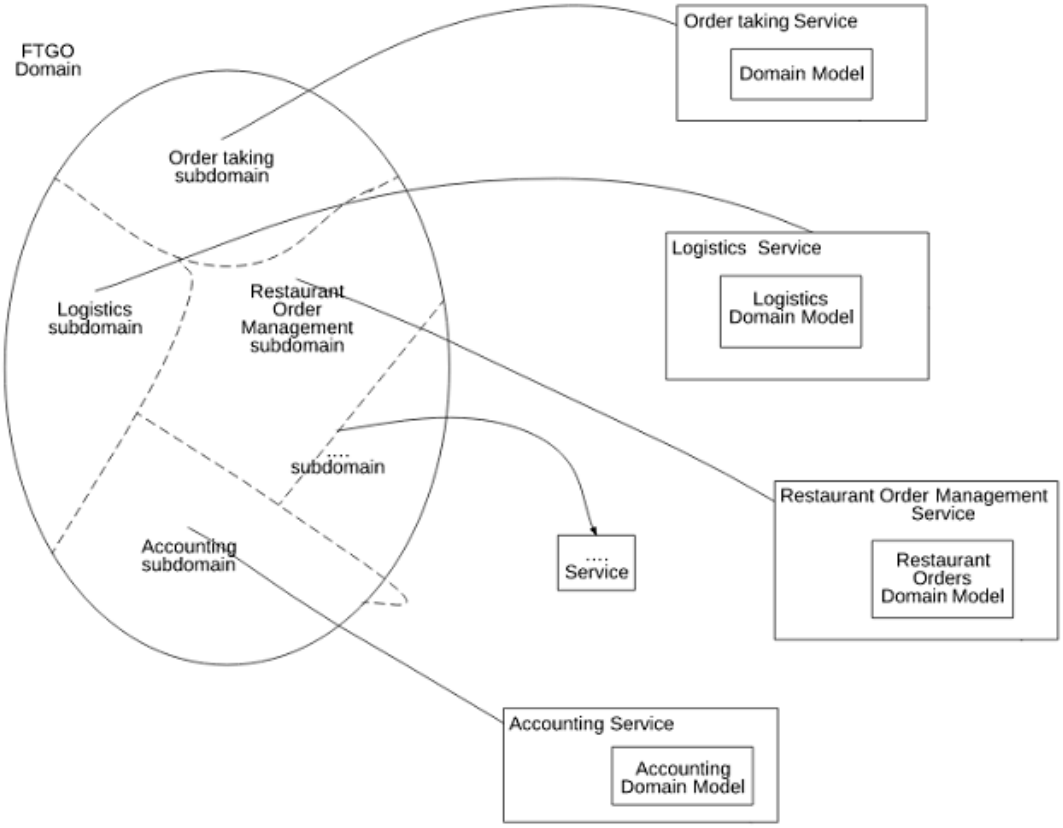
Tämän mallin avulla palvelut määrittyvät muihin malleihin verrattuna helposti, varsinkin mikäli järjestelmän liiketoiminnalliset kyvyt on jo aiemmin tunnistettu ja jaettu niistä erikseen vastaaville tiimeille. Tätä mallia käytettäessä saatetaan kuitenkin joutua tilanteeseen, jossa pääliiketoiminnallisuus jää liian suureksi palveluksi, josta muut palvelut ovat riippuvaisia. Tällöin koko järjestelmä ruuhkautuu helposti yhden palvelun ruuhkautuessa, vaikkakin tätä tilannetta voidaan helpottaa toimivan skaalautumisen avulla. Useat palvelut saattavat olla myös riippuvaisia samoista resursseista, kuten tietokannan tiedoista. Tieto voi myös joutua kulkemaan useamman eri palvelun

läpi kulkiessaan tietokannan ja käyttöliittymän välillä, jolloin tiedon eheys voi muodostua ongelmaksi. Palvelut voivatkin joutua jopa liian riippuvaisiksi toisistaan tiedon eheysongelmia ja resurssien samanaikaisen käytön ongelmia ratkaistaessa, joka rikkoo mikropalveluiden itsenäisyysperiaatetta. (Mt.)

3.2.3 Rajoitetut kontekstit palveluina

Tätä mallia voidaan pitää hieman kehittyneempänä versiona liiketoimintakykyjen mukaisesta palvelujaosta. Rajoitettujen kontekstien ideana on pyrkiä tunnistamaan järjestelmän luontaiset kontekstit ja rajata palvelut näiden mukaan. Rajoitetuilla konteksteilla tarkoitetaan tilannetta, jossa kahdessa eri yhteydessä puhutaan samoilla termeillä, mutta konteksti määrittää sen, mitä kyseisillä termeillä tarkoitetaan. Tämä voidaan ajatella ohjelmistoteknisesti niin, että järjestelmässä käsitellään jotakin entiteettiä, mutta eri käyttötapauksissa ja tilanteissa tarvitaan entiteetistä eri tietoja. Esimerkiksi verkkokauppajärjestelmässä tämä entiteetti voisi olla asiakkaan tilaus. Toimitukseen liittyvässä kontekstissa tarvitaan tiedot siitä, minne tilaus tulisi toimittaa, kun taas varastonhallintaan liittyvässä kontekstissa tarvitaan tietoja siitä mitä asiakas tilasi ja kuinka paljon, jotta varaston saldot pysyvät ajan tasalla. Tämän ajatusmallin pohjalta voidaan rakentaa mikropalveluarkkitehtuuria hyödyntävä järjestelmä, jossa nämä kontekstit tunnistetaan ja luodaan palvelut näiden mukaisesti. (Miettinen 2019.)

Selventävänä esimerkkinä tästä jakotavasta löytyy freecontent.manning –sivustolla (Strategies for Decomposing an Application into Services 2017) oleva malli ravintolajärjestelmän jakamisesta palveluiksi rajoitettujen kontekstien mukaan. Tässä mallissa on tunnistettu edellisen verkkokauppaesimerkin mukaisesti järjestelmän eri toiminnallisuudet, niissä käytettävät kontekstit ja johdettu näistä järjestelmän mikropalvelut (ks. kuvio 7).



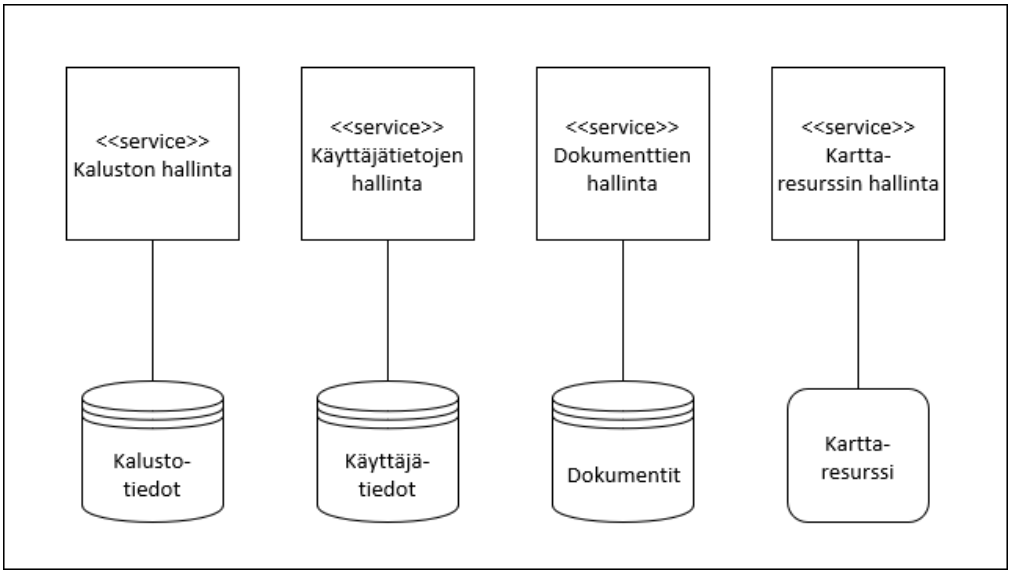
Kuvio 7. Ravintolajärjestelmän rajoitetut kontekstit ja näistä johdetut palvelut (mt)

Periaatteessa tässäkin mallissa palvelut pohjautuvat liiketoimintakykyihin, mutta samalla pyritään ratkaisemaan edellistä mallia tehokkaammin toteutuksessa ilmeneviä käytännön ongelmia. Tästä johtuen tämä malli vaatiikin edellistä enemmän suunnittelutyötä ja osaamista, jotta nämä kontekstit ja näiden väliset yhteydet osataan tunnistaa oikein. Lisäksi tämä malli ei itsessään ota kantaa resursseihin tai niiden hyödyntämiseen, joten mikäli useat palvelut käyttävät samoja resursseja, voi tästä koitua samoja ongelmia kuin edellisessäkin mallissa. (Mt.)

3.2.4 Resurssien mukaiset palvelut

Palveluiden määrittäminen resurssien pohjalta pyrkii ratkaisemaan ongelmat, jotka koituvat useamman palvelun tarpeesta hyödyntää samoja resursseja. Palvelun resursseja voivat olla esimerkiksi tietokannat tai jotkin ulkoiset järjestelmät, joita kehityksen alla oleva järjestelmä tarvitsee voidakseen toteuttaa oman tehtävänsä. Mikäli palvelussa olevaa tietokantaa käyttää useampi palvelu, täytyy tiedon eheys pyrkiä

säilyttämään ja palveluiden pääsyä tietokantaan täytyy kontrolloida, jotta useampi palvelu ei pääse muokkaamaan tietoa samanaikaisesti. Tämä ongelma ratkaistaan niin, että jokaiselle resurssille määritellään oma palvelunsa, joka vastaa kaikesta liikenteestä palvelun vastuulla olevaan resurssiin (ks. kuvio 8). (Miettinen 2019.)



Kuvio 8. Resurssien pohjalta johdetut palvelut (mt, muokattu)

Määrittämällä jokaiselle resurssille siitä vastuussa olevan palvelun palvelut pystytään pitämään itsenäisempänä kuin muissa malleissa, sillä tällöin resurssien suhteen tapahtuvat muutokset saadaan rajattua yhteen palveluun. Mikäli palvelussa on tietokanta, jota käyttää useampi palvelu ja tämän tietokannan teknologia vaihdetaan toiseen, joudutaan muutokset tekemään pahimmassa tapauksessa jokaiseen tietokantaa käyttävään palveluun. Jos taas tietokannasta vastaa yksi palvelu, ovat muutokset rajattu vain ja ainoastaan tähän palveluun. Palvelun itsensä rajapinnat eivät tarvitse muutoksia, joten muut palvelut voivat käyttää tietokannan tarjoamia tietoja aivan kuten ennenkin, joka pienentää työmäärää ja tekee palveluista itsenäisempiä. (Mt.)

Tämän mallin hyötynä on myös se, että samaa koodia resurssin hyödyntämiseksi ei tarvitse toistaa jokaisessa resurssia tarvitsevassa palvelussa erikseen. Resurssista vas-

taavalle palvelulle määritellään toimivat rajapinnat tarpeiden mukaan ja muut palvelut hyödyntävät näitä rajapintoja ilman, että niillä on välttämättä mitään tietoa resurssin itsensä olemassaolosta. (Mt.)

Tämän mallin kääntöpuolena on mahdollisesti kasvava järjestelmän monimutkaisuus. Resursseja voi olla suuressa järjestelmässä useita ja jokaisen resurssin vaatiessa oman palvelunsa kasvaa luonnollisesti myös palveluiden määrä. Lisäksi resurssia hoitavasta palvelusta huolimatta samaa resurssia saattaa tarvita samanaikaisesti useampi palvelu, joten tämän palvelun tulisi vastata tiedon eheydestä ja pyyntöjen priorisoinnista. Ratkaistavaksi jäävät esimerkiksi sellaiset tilanteet, joissa muut palvelut ovat hakeneet jotakin tietoa esitettäväksi käyttöliittymään ja yksi palvelu muuttaa tätä tietoa ja jokin toinen palvelu yrittää muuttaa samaa tietoa vanhentuneen tiedon perusteella. Tämä ongelma ei tosin ole vain tämän mallin ongelma, mutta resurssista vastaavan palvelun pitää osata reagoida oikein edellä mainitun kaltaisiin tilanteisiin. (Mt.)

3.2.5 Käyttötapausten mukaan jakaminen

Resurssien mukaan jakamisen voidaan ajatella perustuvan substantiivien mukaan jakamiseen, kun taas käyttötapausten mukaan jakaminen voidaan johtaa verbeihin. Tämän mallin pohjana on tunnistaa järjestelmän käyttötapaukset, joiden pohjalta luodaan nämä käyttötapaukset mahdollistavat palvelut. (Miettinen 2019.)

Käyttötapaukset ovat oletettavasti usein tiedossa siinä vaiheessa, kun itse arkkitehtuuria aletaan luoda. Tällöin tätä mallia hyödyntämällä on palvelut helppo johtaa aiemmin määritellyistä käyttötapauksista luoden itsenäisiä palveluita, joista jokainen on vastuussa yhdestä tehtävästä mikropalveluarkkitehtuurin periaatteiden mukaisesti.

Käyttötapausten mukaan jakamisesta ei kuitenkaan Miettisen (mt) mukaan löydy paljoa enempää tietoa edellä mainitun lyhyen kuvauksen lisäksi, joten tiedot käytännön toteutuksissa havaituista huonoista puolista jäävät harmillisesti olemattomiksi.

3.2.6 Palvelut toiminnallisuuksien mukaan

Tämä malli on lähellä käyttötapausten mukaista palvelujakoa, sillä tässä mallissa tunnistetaan ohjelmiston toiminnallisuudet ja luodaan palvelut, jotka vastaavat yksittäisistä toiminnallisuuksista. Toiminnallisuuksien mukaan jakaminen vaikuttaa jakavan ihmisten mielipiteitä, sillä joidenkin mukaan mallin hyödyntäminen vaatii asiantuntemusta, kun taas toisten mukaan malli on helppo, eikä vaadi kokenutta arkkitehtiä. (Miettinen 2019.)

Mallin paras puoli vaikuttaisi kuitenkin nopea aloitus ja tästä johtuvat pienet kulut. Tämä johtunee siitä, että ohjelmiston toiminnallisuuksien täytyy olla tiedossa jo ennen toteutusvaihetta, jolloin jokaista toiminnallisuutta kohden luodaan tästä vastaava palvelu. Mallin ongelmat tulevatkin esiin järjestelmän kasvaessa, jolloin palveluiden määrä kasvaa nopeasti ja näiden välisten yhteyksien määrittely hankaloituu. Lisäksi kasvava palveluiden määrä tuottaa ohjelmistoon viivettä, jolloin koko järjestelmä hidastuu. Tästä saattaakin johtua, että joidenkin mielestä tämä malli vaatii osaavaa suunnittelua ja joidenkin mielestä ei. Ilman osaamista on mallin hyviä puolia helppo hyödyntää projektin alussa, mutta työn edetessä vaadittaisiin korkeampaa osaamista ongelmien välttämiseen. (Mt.)

Asiaa pohtiessa mahdollisiksi ongelmiksi voivat nousta myös liiketoimintakykyjen mukaan jakamisesta seuraavat ongelmat, sekä mahdollisesti samankaltaisten toimintojen toistaminen useaan kertaan. Mikäli järjestelmä sisältää useita samankaltaisia toiminnallisuuksia pienillä eroilla, kannattaako näistä jokaiselle tehdä oma palvelunsa, vai laajentaa palvelua koskemaan kaikki tietynlaiset toiminnallisuudet? Ensin mainittua mallia käytettäessä toistetaan koodia turhaan, kun taas jälkimmäisen kohdalla palvelut saattavat paisua aiottua suuremmiksi ja muodostua liian riippuvaisiksi toisistaan. Palveluiden itsenäisyys voi myös kärsiä, sillä keskityttäessä liikaa yksittäisiin toiminnallisuuksiin voivat käyttötapaukset ja kokonaiskuva jäädä toissijaisiksi, jolloin palvelut lopulta muodostuvat riippuvaisiksi toisistaan oikeassa käytössä. (Mt.)

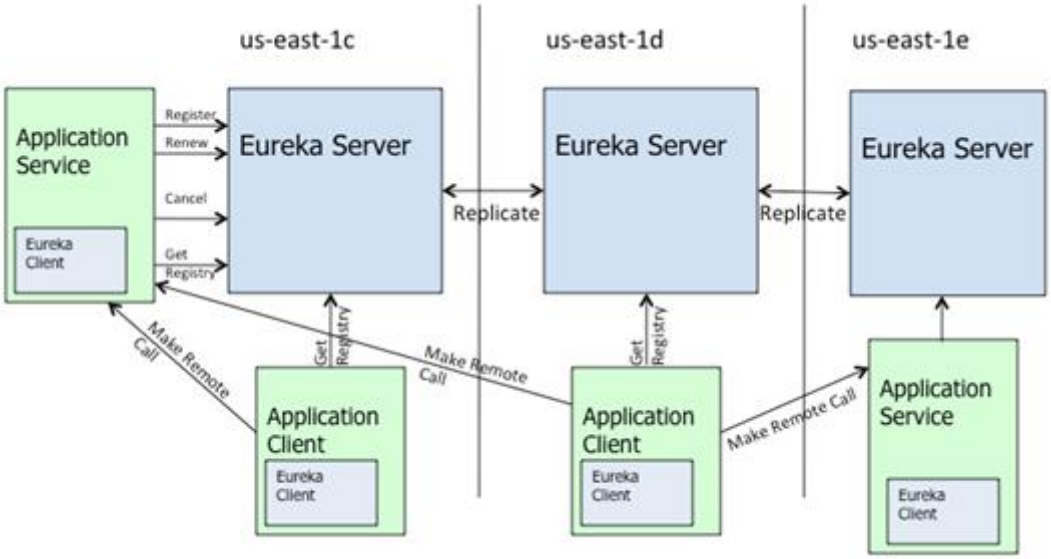
3.3 Mikropalveluiden keskinäinen kommunikointi

Mikropalveluita voi teoriassa luoda monilla erilaisilla teknologioilla, sillä ne ovat vain pieniä yksittäisiä ohjelmistoja, jotka toimivat yhdessä luodakseen suuremman kokonaisuuden. Näiden palveluiden täytyy pystyä kommunikoimaan keskenään, jonka vuoksi palveluiden täytyy saada yhteys toisiinsa. Palveluiden välisten yhteyksien luominen saattaa kuitenkin käydä nopeasti työlääksi, varsinkin mikäli palveluita on toteutettu useita eri teknologioita hyödyntäen. Lisäksi muiden palveluiden osoitteiden ja porttien kovakoodaaminen ei ole järkevä ratkaisu, sillä palveluista voi olla useita instansseja, jotka toimivat eri osoitteissa.

Tätä ongelmaa ratkaisemaan onkin kehitetty useita eri vaihtoehtoja, joista tarkastelemme tarkemmin Spring Cloud Netflixiä.

Spring Cloud Netflix sai alkunsa vuonna 2007, kun Netflix aloitti sisäisen kehitystyön siirtääkseen kaiken toiminnallisuuden pilvipalveluiden varaan. Tämän kehitystyön aikana Netflix kehitti useita kirjastoja ja järjestelmiä helpottamaan mikropalveluiden käyttöä, kuten kuorman tasaukseen käytetyn Ribbonin, sekä palveluiden löytämiseen tarkoitetun Eurekan. Vuonna 2012 Netflix julkaisi nämä kehittämänsä kirjastot ja järjestelmät avoimen lähdekoodin ohjelmistoina, jonka jälkeen avoimen lähdekoodin yhteisö aloitti kehitystyön yhdistääkseen avoimen lähdekoodin Spring Boot alustan ja Netflixin komponentit. Tämä avoimen lähdekoodin projektien yhdistäminen toimi lopulta jopa niin hyvin, että vuonna 2018 Netflix itsekin siirtyi käyttämään Spring Bootia omien järjestelmiensä sijaan. (Wicksell, Cellucci, Yuan, Bross, Yap & Liu 2018.)

Edellä mainitut palvelut helpottavat siis suuresti mikropalveluympäristön kehitystyötä ja käyttöönottoa. Sen sijaan, että jokaisen palvelun täytyisi pitää tietoa muista palveluista sijainteineen, hoitaa Eureka tämän palveluiden puolesta. Kuten kuviossa 9 näkyy, pystytetään ohjelmistoon erillisiä Eureka palvelimia, jotka ovat itsessään yksittäisiä palveluita. (Liu 2014.)



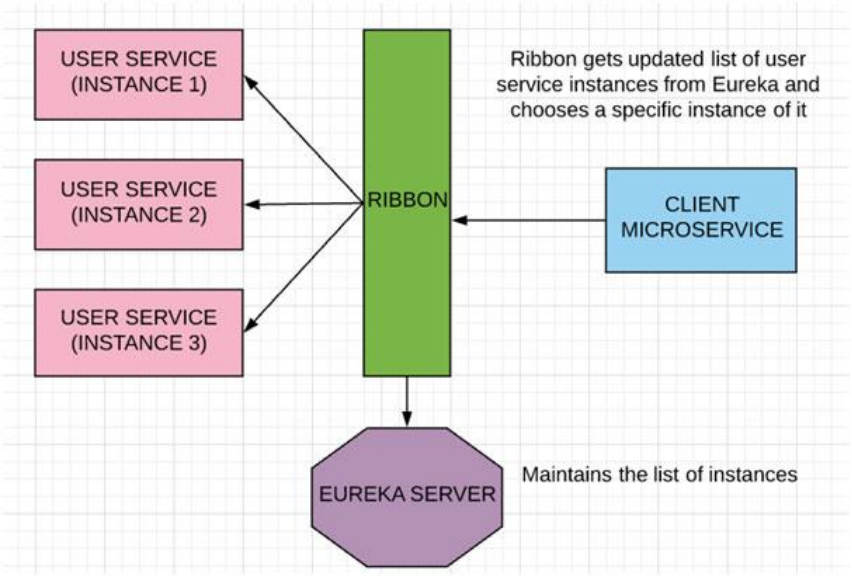
Kuvio 9. Eurekan toiminta mikropalveluiden kanssa (Mt)

Tämän jälkeen kaikki muut palvelut määritellään Eurekan asiakasohjelmiksi (Eureka Client), jotka rekisteröityvät Eurekaan. Rekisteröitymisen jälkeen asiakasohjelmat lähettävät Eurekalle niin kutsuttuja sydämenlyöntejä 30 sekunnin välein ilmoittaakseen palvelun olevan vielä toimintakunnossa. Mikäli Eureka ei saa näitä sydämenlyöntejä, poistetaan palvelu rekisteristä. Eureka siis pitää jatkuvasti kirjaa toiminnassa olevista palveluista ja ilmoittaa tämän muille palveluille, jotka voivat ottaa yhteyttä muihin palveluihin pitämättä itse kirjaa näiden sijainneista. (Mt.)

Vaikka Eureka onkin kehitetty Javalla ja Java-ohjelmistoja varten, on Eurekaa mahdollista hyödyntää myös muita teknologioita käyttävien palveluiden kanssa. Tällaisessa tapauksessa muuta teknologiaa hyödyntävään ohjelmistoon liitetään niin sanottu sivuvaunu (side car), pieni Java-pohjainen ohjelma, joka hoitaa yhteyden palvelun ja Eurekan välillä. (Mt.)

Siinä missä Eureka vastaa palveluiden löytämisestä ja niiden sijainnin ylläpitämisestä, vastaa Ribbon kuorman jakamisesta. Olennaisena etuna mikropalveluiden skaalautumisessa on mahdollisuus luoda helposti useita instansseja samasta palvelusta. Näistä useista instansseista ei ole kuitenkaan mitään hyötyä, mikäli asiakasohjelmalla ei ole mahdollisuutta käyttää kuin yhtä instanssia, jolloin pahimmassa tapauksessa kaikki

pyynnöt ohjautuvat samalle palvelulle. Tässä mukaan astuu Ribbon, jonka toimintaa havainnollistetaan kuviossa 10. (Spring Cloud Netflix – Ribbon n.d.)



Kuvio 10. Ribbonin toiminta mikropalveluympäristössä (Spring Cloud Netflix – Ribbon n.d.)

Edellä esitetyssä esimerkkiprojektissa on käytössä Ribbon, Eureka palvelin, sekä kolme instanssia palvelusta User Service, joista jokainen toimii eri portissa. Tässä tapauksessa Eureka pitää kirjaa palveluista sijainteineen, joka antaa tiedot eteenpäin Ribbonille. Ribbonin saadessa asiakasohjelmalta pyynnön, suorittaa Ribbon algoritmin perusteella päätöksen mille instanssille kyseinen pyyntö ohjataan. Tällöin kehittäjien ei tarvitse itse välttämättä huolehtia kuorman tasaamisesta, sillä Ribbon hoitaa sen jo perusasetuksillaankin. Ribbonissa on toki lisäksi mahdollisuuksia konfigurointiin, kuten esimerkiksi kuormantasausperusteen valitseminen ennalta määritellyistä vaihtoehdoista, mutta näihin ei ole pakollista perehtyä, mikäli esimerkiksi aika on rajallista. (Mt.)

4 Kalustorekisteri

4.1 Kalustorekisterin vaatimukset

Kalustorekisterin tarkoituksena on tarjota käyttäjilleen mahdollisuus ylläpitää rekisteriä monenlaisesta kalustosta yksityiskohtaisine kuvauksineen, tietoineen sekä sijainteineen ja asiaan liittyvine dokumentteineen. Tietoa täytyy olla mahdollista päivittää, poistaa ja lisätä yksittäisiä entiteettejä sekä näiden parametreja. Lisäksi täytyy olla mahdollisuus tuoda ja viedä suuria datamääriä muista palveluista. Tiedon esittäminen ja muokkaaminen eivät sellaisenaan riitä, sillä jotkin tiedot tuovat käyttäjälle enemmän lisäarvoa visualisoituna, joten tietoa täytyy kyetä myös visualisoimaan. Tämä tapahtuu näyttämällä käyttäjälle kuvia kalustosta, mallintamalla yksittäisiä parametreja mahdollisuuksien mukaan sekä esittämällä kaluston mahdollinen sijainti kartalla. Käyttäjällä täytyy myös olla mahdollisuus tietojen muokkaamisen lisäksi lisätä ja ladata kuvia, sekä muuta kalustoon liittyvää dokumentaatiota.

Kaikki edellä mainitut toiminnallisuudet täytyy olla mahdollisia käyttäjälle nykyaikaisen web-käyttöliittymän avulla ja lisäksi taustalla on hyödynnettävä mikropalvelutekнologioiden parhaita puolia kuten skaalautuvuutta ja mahdollisten vikatilanteiden eristämistä. Samalla mikropalveluiden mahdollisia haittapuolia täytyy pyrkiä minimoimaan, sillä liian monimutkainen järjestelmä on hankala ylläpitää ja saattaa luoda järjestelmään liikaa kutsuja, jotka hidastavat koko järjestelmää.

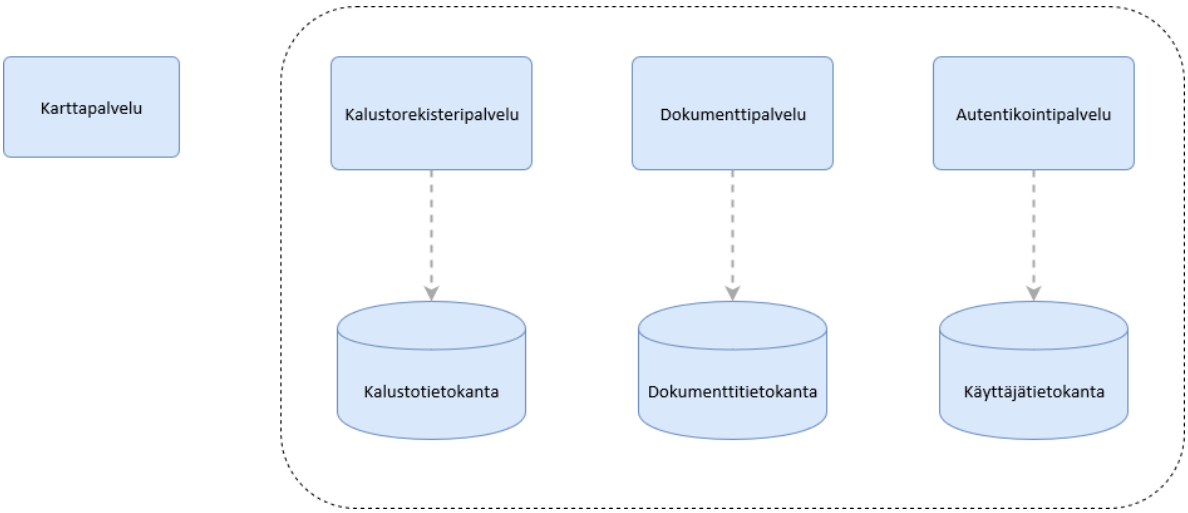
Tärkeänä osana kalustorekisteriä on myös tiedon luokittelu ja käyttäjähallinta. Sekä kalustolle että yksittäisille parametreille täytyy olla mahdollista asettaa erityinen luokitus, joka määrittää, kenelle järjestelmän tiedot näkyvät. Käyttäjähallinta astuu kuvaan tiedon luokittelun myötä, sillä kalustorekisterissä on erilaisia käyttäjiä, joiden käyttöoikeuksissa on eroja. Järjestelmän on siis pystyttävä käyttäjätietojen perusteella tarjoamaan vain ne toiminnallisuudet käyttäjälle, jotka on hänelle määritelty. Yksi tällainen ominaisuus on esimerkiksi tiedon muokkaaminen. Tiedon muokkaaminen on mahdollista vain tietyille käyttäjäryhmille, kun taas tietojen selaamisen täytyy olla mahdollista kaikille.

4.2 Mikropalveluiden valitseminen vaatimusten perusteella

Kalustorekisterin kohdalla mikropalveluiden haluttiin tuovan lisäarvoa verrattuna monoliittiseen ohjelmistoon tuomatta kuitenkaan turhaa monimutkaisuutta järjestelmään. Tarkoituksenmukaista on myös pyrkiä pitämään mikropalvelut kooltaan järkevinä, jotta niiden ylläpito ja testaaminen olisi yksinkertaisempaa verrattuna monoliittiseen järjestelmään. Mitkään keinotekoiset rajoitukset kuten x määrä koodia, eivät kuitenkaan ole tarkoituksenmukaisia, vaan palvelut täytyisi pystyä määrittelemään käyttötarkoitusten mukaan niin, että jokaisella palvelulla on selkeä tehtävä ja rooli järjestelmäkokonaisuudessa.

Tutkimuksen jälkeen mikropalvelujako päädyttiin suorittamaan hyödyntämällä jakamista resurssien mukaan. Valitussa jakomallissa palveluita määritellään pääasiassa resurssien mukaan, eli teoriassa jokaisesta resurssista vastaa yksi palvelu, joka toimii samalla myös välikätenä toisille palveluille näiden tarvitessa jotain palvelun vastaamasta resurssista. Yksi tällainen resurssi on kalustotietokanta, jolle haluttiin määritellä yksi palvelu vastaamaan kaikesta liikenteestä tietokantaan. Määriteltäessä selkeät yhdyspalvelut resursseille voidaan hyödyntää tehokkaammin aiemmin esiteltyä palvelun etsintää, sillä esimerkiksi Eurekaa käytettäessä jokaisella palvelulla on nimi, jonka avulla muut palvelut voivat ottaa yhteyttä tähän palveluun. Tällöin vain resurssista vastaavan palvelun täytyy tietää esimerkiksi tietokannan osoite, puhumattakaan muista tietokannan vaatimista tiedoista kuten hakulauseista tai kirjautumistiedoista. Mikäli resursseista vastaaville palveluille määritellään vielä selkeät ja helppokäyttöiset rajapinnat, on palveluun myöhemmin mahdollista lisätä muitakin kyseistä resurssia käyttäviä palveluita vaivattomasti.

Edellä mainittujen perusteiden ja vaatimusten perusteilla saatiin kalustorekisterin resursseiksi määriteltyä kalustotietokanta, joka toimii kalustorekisterin pääasiallisena tietokantana pitäen sisällä kaiken tiedon kalustosta, dokumenttitietokanta, käyttäjätietokanta sekä kolmannen osapuolen karttapalvelu. Jokaiselle näistä resursseista määriteltiin kaikesta resurssiin kulkevasta liikenteestä vastaava palvelu lukuun ottamatta karttapalvelua, jolla oletetaan olevan valmiit rajapinnat. Tämän perusteella kalustorekisterille saatiin päätettyä jo kolme mikropalvelua (ks. kuvio 11).



Kuvio 11. Resurssit ja niistä vastaavat palvelut

Rajoitettujen kontekstien perusteella jakaminen taas astuu mukaan kuvaan mietittäessä muita palveluita. Kuten aiemmin on mainittua, perustuu rajoitettujen kontekstien mukaan jakaminen siihen, että vaikka eri palveluissa käsiteltäisiin samoja asioita, ei joka palvelussa tarvita jokaisen entiteetin jokaista tietoa, vaan kontekstit määrittävät tarvitun tiedon. Esimerkkinä tästä voidaan käyttää karttapalvelua, joka ei tarvitse kalustosta välttämättä muuta kuin nimen ja sijaintitiedot, kun taas itse kalustorekisteripalvelu käyttää kaikkia tietoja välittäessään tietoja muille palveluille.

Lisäksi järjestelmä vaatii käyttöliittymän, jolle päätettiin omistaa yksi palvelu. Käyttöliittymäpalvelu sisältää kaiken toiminnallisuuden, jota tarvitaan tiedon esittämiseen käyttäjälle. Käyttöliittymän pääasiallisena tehtävänä on esittää käyttäjälle palveluilta vastauksena saamansa tiedot käyttäjälle ymmärrettävässä muodossa.

Kalustorekisteripalveluun itseensä olisi mahdollista sisällyttää hakumahdollisuus, mutta haun muuttuessa monimutkaisemmaksi paisuu tällöin myös kalustorekisteripalvelu, eikä se toteuta enää vain yhtä tehtävää. Hakutoiminnallisuus voidaankin irrottaa omaksi palvelukseen, joka toimii välikätenä käyttöliittymän ja kalustorekisteripalvelun välillä. Tällöin hakupalvelulla ja kalustorekisterillä on omat selkeät roolinsa ja ne vastaavat vain yhdestä asiasta yhden vastuullisuuden periaatteen mukaisesti.

Lisäksi molempien palveluiden rajapinnat saadaan pidettyä tällä ratkaisulla paljon yksinkertaisempina ja molemmissa palveluissa voidaan tarvittaessa soveltaa eri teknologioita.

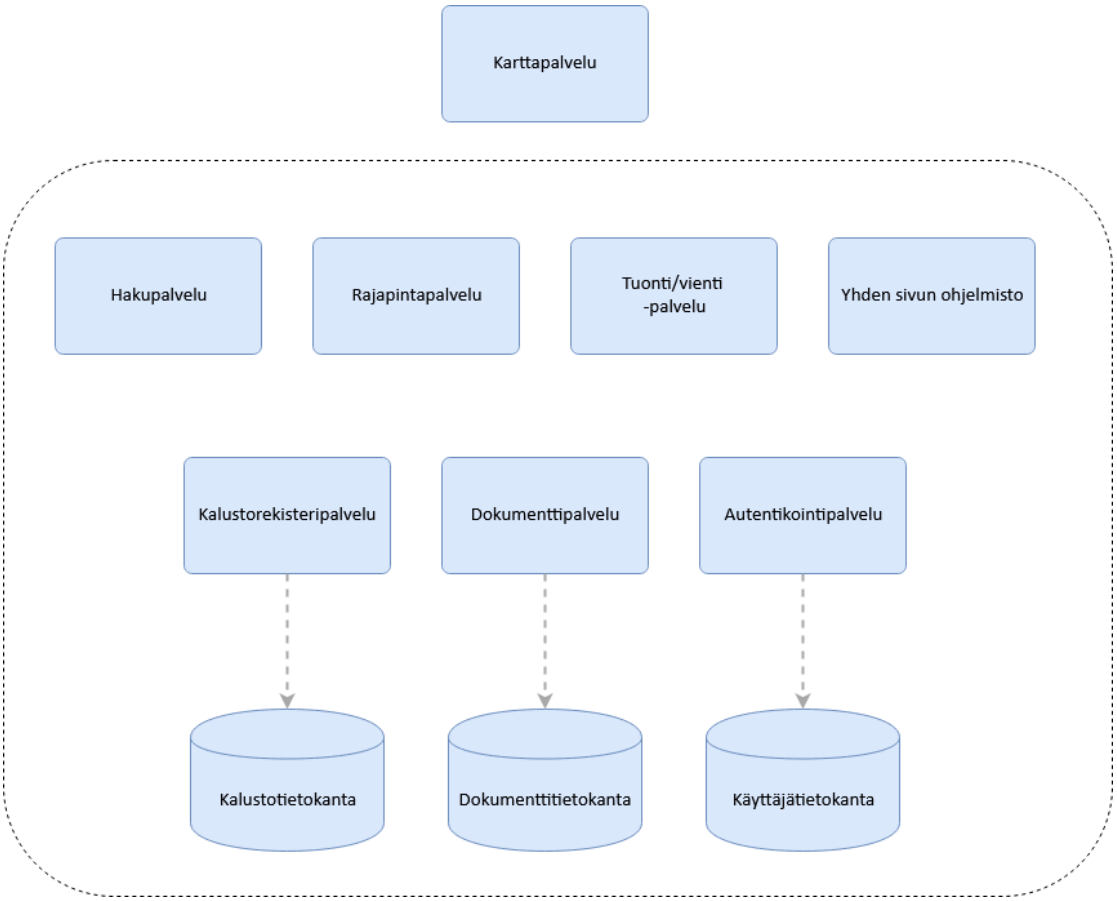
Kuten hakupalvelun kohdalla, kalustorekisteripalvelu voisi mahdollisesti toteuttaa myös isojen datamäärien tuonnin ja viennin, mutta tässäkin tapauksessa kalustorekisteripalvelun vastuualue kasvaisi yhtä asiaa suuremmaksi. Tiedon tuomisen ja viemisen voi toki sinällään ajatella olevan vain tiedon viemistä tai hakemista tietokannasta, mutta merkittävänä osana tässä on tiedon muokkaaminen soveltuvaan muotoon. Tietoa voi tulla jostakin ulkopuolisesta järjestelmästä suuria määriä jossakin tietokantaan sopimattomassa muodossa, jolloin kaikki tämä tieto täytyy käydä läpi ja tehdä siitä oikean muotoista. Lisäksi tuettuja tiedostomuotoja voi olla useita erilaisia ja näitä tuettuja muotoja voi olla tarvetta lisätä myöhemmin. Tällaisessa tapauksessa ei siis ole järkevää lisätä kalustorekisterille rajapintoja jokaiselle tiedostomuodolle tiedon tuomista varten, vaan tämä on järkevämpi käsitellä omassa palvelussaan ja hyödyntää kalustorekisterin omia rajapintoja tiedon siirtämiseen tietokantaan tai sieltä pois.

Palveluiden keskinäisestä kommunikoinnista oli esitetty aiemmin maininta siitä, miten tähän on olemassa valmiita ratkaisuja. On totta, että Eurekan avulla käyttöliittymä voisi olla suoraan yhteydessä sekä kalustorekisteripalveluun, että autentikointipalveluun, mutta mitä jos kirjautumaton käyttäjä yrittää hakea tietoa kalustorekisteripalvelulta? Käyttäjä täytyy joka tapauksessa autentikoida tämän esimerkin lisäksi myös muissa tapauksissa, jolloin jokaiseen palveluun täytyisi tehdä oma ratkaisunsa autentikointipalvelun kutsumiseen ja pyyntöihin vastaamiseen autentikointipalvelun vastauksen perusteella.

Tästä syystä järjestelmään päätettiin luoda erillinen rajapintapalvelu, jota kaikki muut palvelut kutsuvat ja joka ohjaa palveluiden kutsut edelleen eteenpäin Eurekaa avuksi käyttäen. Aiemmin mainittua esimerkkitapausta peilaten lähettäisi rajapintapalvelu autentikointipalvelulle kyselyn käyttöliittymältä ja mikäli käyttäjä ei ole kirjautunut palauttaisi rajapintapalvelu kalustorekisterille suunnatun pyynnön vastauksena vaik-

kapa http-koodin 403. Toisena vaihtoehtona olisi käyttäjän ohjaaminen kirjautumissivulle, mikäli hän yrittää kirjautumattomana hakea kirjautuneille käyttäjille kuuluvaa sisältöä. Joka tapauksessa kaikki pyyntöjen ohjaamiseen tarvittu logiikka saadaan yhteen palveluun sen sijaan, että samaa toiminnallisuutta toistetaan useassa eri palvelussa.

Näin kalustorekisterille on määritelty kuviossa 12 näkyvät palvelut ja tietokannat.



Kuvio 12. Kalustorekisterin palvelujako

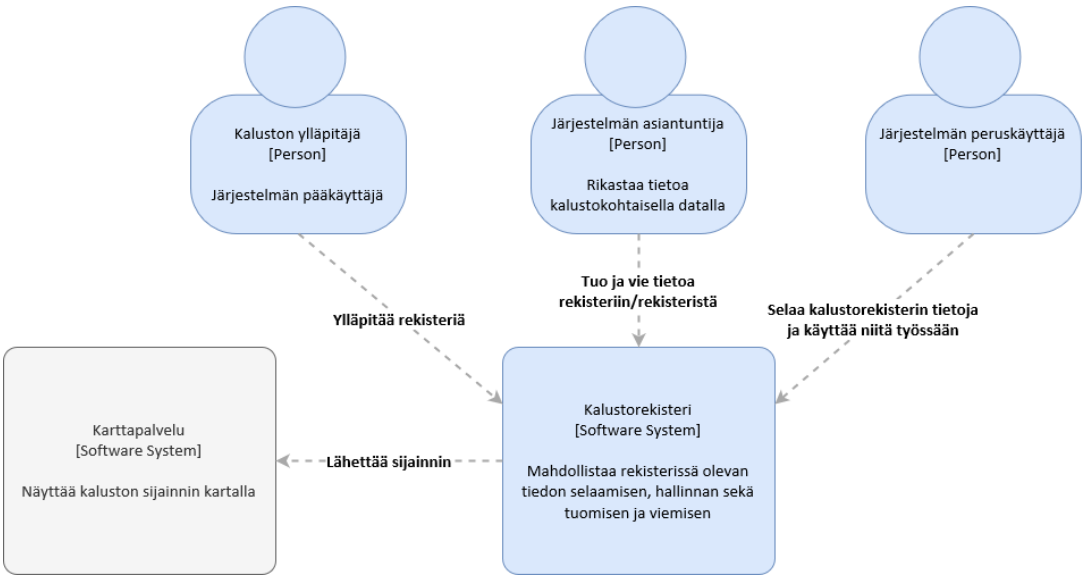
Jokaiselle tietokannalle on määritelty resurssien mukaisen jakomallin perusteella oma palvelunsa, joka vastaa kaikesta liikenteestä tietokantaan. Tietokannoista vastaavien palveluiden lisäksi on muita toiminnallisuuksia varten neljä muuta palvelua,

joista jokainen vastaa selkeästi yhdestä vastuualueestaan. Tällöin palvelut toteuttavat yhden vastuun periaatetta ja auttavat kehittäjiä ymmärtämään yksittäisen palvelun roolin järjestelmän kokonaiskuvassa.

4.3 Mikropalveluarkkitehtuuri

Palveluiden määrittelyn jälkeen tarvitsee järjestelmä arkkitehtuurikuvauksen, sillä pelkillä palveluilla itsessään ei tee mitään, mikäli niiden väliset yhteydet eivät ole selvät. Aikaisemmassa osiossa yhdeksi palveluksi määriteltiin rajapintapalvelu, joka vastaa palveluiden pyyntöjen välittämisestä toisille palveluille. Rajapintapalvelu ei voi kuitenkaan tietää palveluiden puolesta mitä palvelut haluavat tehdä, joten tätä varten on syytä määritellä mitä mikin palvelu tarvitsee toisilta palveluilta voidakseen täyttää sille asetetut vaatimukset.

Arkkitehtuurissa on tärkeää ymmärtää, miten koko järjestelmä toimii yhdessä käyttäjien ja muiden mahdollisten järjestelmien kanssa, jonka vuoksi järjestelmästä täytyy olla myös ylemmän tason kuvaus olemassa. Kalustorekisterin käyttötarpeet sekä palvelut määriteltiin jo aiemmissa osioissa, mutta miten kalustorekisteri näyttäytyy käyttäjilleen? Tätä varten hyödynnettiin Simon Brownin kehittämän (Brown n.d.) C4 mallinnuksen System Context tason kuvaajaa, joka näyttää miten kyseessä oleva järjestelmä sopii järjestelmää ympäröivään maailmaan (ks. kuvio 13).

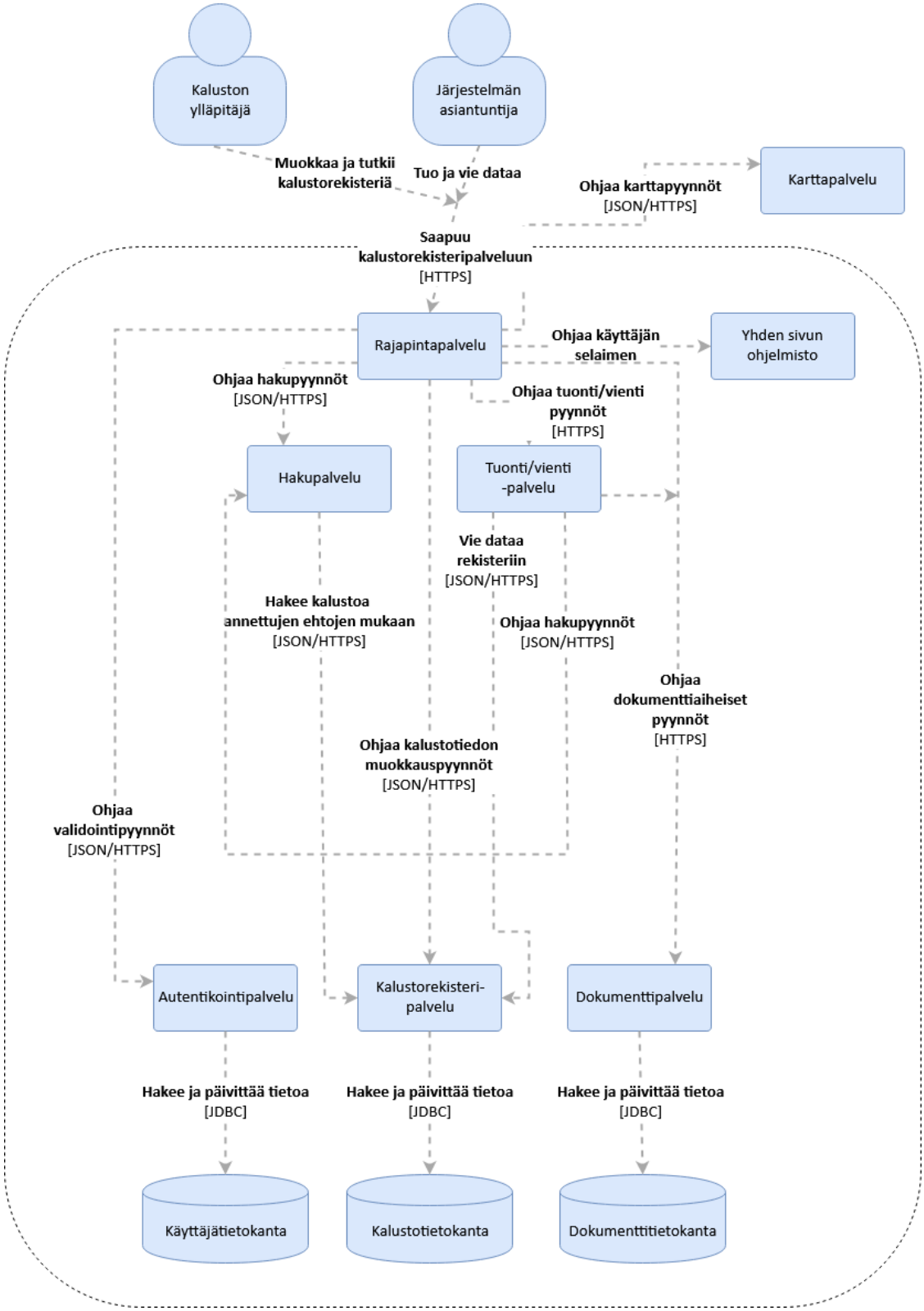


Kuvio 13. C4 System context tason kaavio (Miettinen 2019, muokattu)

Kalustorekisteri on siis järjestelmä, joka mahdollistaa rekisterissä olevan tiedon selaamisen ja hallinnan tarjoten eri tasoisia toiminnallisuuksia eri käyttäjätyypeille. Perustason käyttäjällä ei ole mahdollisuutta muokata tai lisätä tietoa järjestelmään, vaan hänellä on mahdollisuus ainoastaan selata olemassa olevaa tietoa ja hyödyntää rekisterin tarjoamia tietoja työssään. Asiantuntijalla on peruskäyttäjää laajemmat oikeudet ja hänellä on mahdollisuus muokata olemassa olevaa tietoa, sekä tuoda ja viedä tietoa muihin mahdollisiin järjestelmiin tarpeen mukaan. Kolmantena käyttäjätyyppinä on ylläpitäjä, joka nimensä mukaisesti vastaa rekisterin ylläpidosta.

Ulkoisena yhteytenä kalustorekisterillä on karttapalvelu, joka on erillinen järjestelmänsä. Teoriassa tämä yhteys toimii niin, että kalustorekisteri lähettää karttapalvelulle sijainnin ja mahdollisesti muut tarvittavat tiedot, jonka jälkeen karttapalvelu visualisoi kaluston sijainnin kartalla.

Seuraava taso C4 mallissa on Container diagram, jonka tarkoituksena on kuvata järjestelmän osia ja näiden keskinäistä kommunikointia. Kalustorekisterin tapauksessa järjestelmän osat ovat jo aiemmin määritellyt mikropalvelut. Mikropalvelut yhteyksineen ovat esitetty kuviossa 14.



Kuvio 14. Container diagram tason kaavio (Miettinen 2019, muokattu)

Kaaviossa iso katkoviivoilla rajattu alue kuvastaa kalustorekisteriä ja kaikki tämän alueen sisällä oleva kuuluu järjestelmään. Järjestelmän ulkopuolelle on tässä tapauksessa rajattu aiemmin mainitut karttajärjestelmä ja käyttäjät. Arkkitehtuurikuvauksen läpikäyminen kannattaa aloittaa käyttäjistä ja heidän toimistaan, sillä tämä auttaa paremmin kuvaamaan järjestelmän käyttöä ja miten eri palveluiden täytyy toimia yhteistyössä keskenään.

Kalustorekisterijärjestelmässä on web-käyttöliittymä, jolloin käyttäjät aloittavat kalustorekisterin käytön menemällä selaimellaan sille määritellylle sivulle. Käyttäjien kaikki pyynnöt ohjautuvat rajapintapalvelulle, jonka tehtävänä on pyyntöjen ohjaaminen muille palveluille. Järjestelmän täytyy tietää, onko käyttäjä kirjautunut sisään, jolloin rajapintapalvelu lähettää pyynnön autentikointipalvelulle. Ensimmäistä kertaa sivulle päätyessään käyttäjä ei ole vielä kirjautuneena sisään, jolloin autentikointipalvelu palauttaa rajapintapalvelulle tiedon tästä. Tämän jälkeen rajapintapalvelu välittää tämän tiedon käyttöliittymäpalvelulle (kuvassa yhden sivun ohjelmisto), joka näyttää käyttäjälle kirjautumissivun. Kirjautumissivulla käyttäjä syöttää tunnuksensa ja salasanansa, jonka jälkeen nämä kirjautumistiedot lähtevät jälleen eteenpäin autentikointipalvelulle. Autentikointipalvelu tarkistaa tietokannastaan kirjautumistietojen oikeellisuuden, sekä mahdollisen käyttäjäroolin. Mikäli kirjautumistiedot ovat oikein, palautuu tieto tästä rajapintapalvelulle. Kirjautumistietojen ollessa väärät palauttaa autentikointipalvelu virhekoodin, jonka käyttöliittymä esittää käyttäjälle ymmärrettävässä muodossa.

Onnistuneen kirjautumisen jälkeen rajapintapalvelu ohjaa käyttäjän kalustorekisterin pääsivulle, jolloin käyttöliittymäpalvelu renderöi käyttäjälle päänäkymän. Tästä näkymästä käyttäjä voi hakea ja selata kalustorekisterin tietoja, sekä siirtyä muihin näkymiin. Käyttäjä voi esimerkiksi haluta hakea kaiken tietyn tyyppisen kaluston päänäkymään, jolloin tästä lähtee rajapintapalvelun kautta pyyntö kalustorekisteripalvelulle. Kalustorekisteripalvelu käsittelee pyynnön, hakee sen perusteella tietokannasta halutun tiedon ja lähettää sen takaisin rajapintapalvelulle. Tieto päätyy rajapintapalvelun kautta käyttöliittymäpalvelulle, joka muokkaa tiedon käyttäjälle ymmärrettävään muotoon, esimerkiksi listaksi.

Käyttäjäoikeudet täytyy kuitenkin varmistaa muulloinkin kuin vain kirjautumisen yhteydessä, sillä vaikka tieto käyttöoikeuksista on kerran varmistettu, ei seuraavien pyyntöjen kohdalla voida luottaa siihen, että käyttöliittymältä tulevalla pyynnöllä on sopivat käyttöoikeudet. Tämän johdosta jokainen pyyntö täytyy kierrättää autentikointipalvelun kautta ja esimerkiksi edellisessä kohdassa mainitussa tilanteessa pyyntö vahvistettaisiin ennen kalustorekisteripalvelulle viemistä. Mikäli käyttäjällä ei olisi sopivia oikeuksia, pyyntö evättäisiin. Lisäksi mikäli käyttäjän kirjautuminen olisi vanhentunut/kirjautumistieto olisi virheellistä, kirjattaisiin käyttäjä ulos ja ohjattaisiin kirjautumissivulle.

Mikäli kalustorekisteristä halutaan hakea jotakin tarkempaa tietoa, käytetään tähän tarkoitukseen hakupalvelua. Tällöin käyttäjä valitsee käyttöliittymän avulla hakuehdot kuten esimerkiksi mihin kategorioihin hakua sovelletaan. Käyttäjän asettamat ehdot lähtevät rajapintapalvelun kautta eteenpäin hakupalvelulle, joka taas muokkaa käyttöliittymältä tulleet ehdot kalustorekisteripalvelun ymmärtämään muotoon. Kalustorekisteripalvelu hakee tietokannasta hakuehdot täyttävät entiteetit ja palauttaa ne rajapintapalvelulle ja edelleen käyttöliittymän esitettäväksi.

Dokumentit ovat lähtökohtaisesti kalustorekisterissä olevien entiteettien liitteitä, mutta ne ovat laitettu omaan tietokantaansa ja oman palvelun taakse järjestelmän selkiyttämiseksi. Kalustorekisteristerin entiteetit ja dokumentit ovat liitettyinä toisiinsa viittauksin ja kalustotietoa esitettäessä käydään käyttöliittymän puolella läpi entiteetin tarvitsemat dokumentit ja lähetetään näistä pyyntö rajapintapalvelulle, joka lähettää sen dokumenteista vastaavalle palvelulle. Dokumenttipalvelu hakee tietokannasta kyseisillä viittauksilla olevat dokumentit ja lähettää ne edelleen rajapintapalvelun kautta käyttöliittymälle. Käyttöliittymän kautta on myös mahdollista lisätä tai poistaa dokumentteja entiteeteiltä, jolloin yhteysketju on samanlainen kuin dokumentteja haettaessa. Kun dokumentit on haettu käyttöliittymään, on näitä mahdollista ladata laitteelle, jolla kalustorekisteriä käytetään, mutta tämä ei vaadi uutta kutsua dokumenttipalvelimelle, sillä dokumentit ovat jo laitteen välimuistissa.

Tuonti/vienti -palvelu toimii hieman samaan tapaan kuin hakupalvelu, eli käyttöliittymältä rajapintapalvelun kautta tuleva tieto käsitellään kalustorekisterin ymmärtämään muotoon. Tiedon viennin tapauksessa taas kalustorekisteriltä tuleva tieto käsitellään käyttäjän haluamaan, esimerkiksi jonkin toisen järjestelmän tai ohjelman käyttämään muotoon. Tämän käsittelyn jälkeen tieto lähetetään halutussa muodossa käyttöliittymälle, josta käyttäjä voi ladata sen omalle laitteelleen.

Karttapalvelu on omana järjestelmänään, mutta silti sekin tarvitsee tietoa kalustorekisteristä. Karttapalvelun käytön tapauksessa rajapintapalvelu hakee karttapalvelun tarvitsemat resurssit kalustorekisteripalvelun kautta, kuten sijaintitiedot, ja lähettää ne karttapalvelulle, joka esittää kaluston sijainnin kartalla.

C4 malli tarjoaisi vielä edellisiä tasojakin syvemmälle menevät kaaviot nimeltään Component diagram ja Code diagram, joista ensin mainitussa kuvataan tarkemmin jokaisen palvelun käyttämät komponentit ja näiden väliset suhteet ja jälkimmäisessä kuvataan jokaisen komponentin toimintaperiaate. Näiden kaavioiden käyttäminen on tässä yhteydessä kuitenkin rajattu työn ulkopuolelle epäolennaisena, sillä ne eivät suoranaisesti liity mikropalveluarkkitehtuuriin tai palvelujakoon. Lisäksi liian tarkka ja syvälle menevä mallintaminen on ketterässä kehitysympäristössä ajoittain turhaa työtä, sillä yksityiskohdat saattavat muuttua usein, jolloin kaaviot jäävät nopeasti vanhentuneiksi ja turhiksi.

5 Analyysi

5.1 Yleistä

Mikropalveluarkkitehtuurista löytyy monia erilaisia mielipiteitä niin puolesta kuin vastaan, sekä sen suhteen miten mikropalveluarkkitehtuuria tulisi toteuttaa. Mikropalveluarkkitehtuuri ei missään nimessä ole paras mahdollinen ratkaisu jokaiseen kuviteltavissa olevaan ohjelmistoprojektiin, vaan joissakin tapauksissa perinteinen monoliittinen arkkitehtuuri soveltuu mikropalveluarkkitehtuuria paremmin. Erityisen

totta tämä on pienehköissä projekteissa ja sellaisissa projekteissa, joissa on tarkoituksena saada heti alussa nopeasti jotain näkyvää aikaan. Mikropalveluarkkitehtuuri tuo myös oman lisänsä monimutkaisuutta projektiin ja kehittäjien täytyy ymmärtää palveluiden rooli mahdollisimman itsenäisinä, mutta yhteistyötä tekevinä järjestelmän osina. (Gnatyk 2018.)

Mikäli mikropalveluarkkitehtuuriin päädytään, täytyy tällöin tehdä päätöksiä palvelujaon ja arkkitehtuurin suhteen, joita on käyty aiemmissa osioissa läpi. Kuten valinta monoliittisen ja mikropalveluarkkitehtuurin välillä, täytyy valinta palvelujakoperusteen suhteenkin tehdä kyseessä olevan ohjelmiston mukaan. Tätä varten arkkitehtuurista vastaavilla henkilöillä täytyy olla tarpeeksi kattava käsitys ohjelmiston tarpeista ja liiketoiminnallisista vaatimuksista, jotta valittu palvelujakoperuste ei muodostu taakaksi ohjelmiston kasvaessakaan.

Valinta monoliittisen ja mikropalveluarkkitehtuurin kohdalla ei ole kuitenkaan välttämättä lopullinen, vaan valintaa on mahdollista muuttaa myöhemmin. Arkkitehtuurin muuttaminen tuo toki lisää työtä ja kuluja, mutta joissain tapauksissa oikeaa tarvetta ei välttämättä tunnisteta heti aluksi, tai vaihtoehtoisesti resurssit eivät heti riitä mikropalveluarkkitehtuuriin. Tällainen tilanne voi olla esimerkiksi demosovellusta tehdessä, jolloin on yksinkertaisempaa ja nopeampaa luoda monoliittinen sovellus. Demosovellus voi kuitenkin kasvaa esimerkiksi yllättävän asiakaskiinnostuksen vuoksi, jolloin voidaan tehdä siirtymä mikropalveluarkkitehtuuriin tarpeen ja resurssien kasvaessa.

Mikropalveluarkkitehtuuria ei siis tule valita mikropalveluarkkitehtuurin itsensä vuoksi tai sen mahdollisen trendikkyyden vuoksi, vaan jokaisen järjestelmän kohdalla täytyy ottaa huomioon lukuisia eri asioita kuten aikataulu, vaatimukset, käytettävissä olevat resurssit ja tehdä päätös arkkitehtuurivalinnasta näiden tosiasioiden perusteella.

5.2 Mikropalveluarkkitehtuurin soveltuvuus kalustorekisteriin

Kalustorekisteristä oli olemassa monoliittinen työpöytäohjelma, joka täytti sille asetetut vaatimukset demoon vaadittavalla tasolla. Oliko mikropalveluarkkitehtuuriin siirtymisestä kuitenkaan loppujen lopuksi hyötyä kalustorekisterin tapauksessa, vai lisäsikö se vain järjestelmän monimutkaisuutta ja työmäärää tuomatta mitään uutta?

Kalustorekisteriä täytyykin tarkastella vaatimustensa ja ominaisuuksiensa kautta, jotta on mahdollista selvittää mikropalveluarkkitehtuurin soveltuvuus. Kalustorekisterissä on lähtökohtaisesti monta osa-aluetta, jotka oikeuttavat mikropalveluarkkitehtuurivalinnan ja lisäksi monoliittinen versio oli jo olemassa, jolloin kyseisen mallin tuomat rajoitteet olivat käyneet jo aiemmin selväksi.

Kalustorekisterissä on useita toisistaan erottuvia toiminnallisia kokonaisuuksia, joille saattaa olla eri tilanteissa vaihteleva määrä kysyntää. Jakamalla nämä toiminnallisuudet omiksi kokonaisuuksikseen voidaan kalustorekisterissä hyödyntää tehokkaampaa skaalausta vaihtelevan tarpeen mukaisesti sen sijaan, että koko järjestelmää täytyisi skaalata yhden osa-alueen ruuhkautuessa. Toiminnallisuuksien jakamisesta erillisiin palveluihin on kalustorekisterin tapauksessa myös hyötyä eristämisen ja riippuvuuksien vähentämisen suhteen. Monoliittisen ohjelmiston ollessa suuri saattaa jokainen pienikin muutos yhteen osa-alueeseen rikkoa koko järjestelmän käyttökelvottomaksi. Vähintäänkin muutokset vaativat koko ohjelmiston kääntämisen ja mahdollisten testien ajamisen, joka vie järjestelmän kasvaessa aina vain enemmän ja enemmän aikaa. Eristämällä muutokset yksittäisiin palveluihin tästä ongelmasta päästään eroon ja esimerkiksi hakutoiminnallisuuden hajoaminen omassa palvelussaan ei estä koko järjestelmän käyttöä. Samoin myös muutosten kohdalla testit tarvitsee ajaa vain muutosta koskevan palvelun sekä mahdollisesti siitä riippuvaisten palveluiden osalta koko järjestelmän sijaan, eikä muita järjestelmän osia tarvitse kääntää uudestaan.

Tämä toiminnallisuuksien jakaminen erillisiin palveluihin helpottaa alun jälkeen myös kehittäjien työtä muistakin kuin edellä mainituista syistä. Mikropalveluarkkitehtuuriin perustuvan järjestelmän sisäistäminen saattaa aluksi olla hankalaa varsinkin, jos tällaisesta järjestelmästä ei ole kokemusta. Kuitenkin hyvin määriteltyyn palvelujakoon

perustuvassa järjestelmässä kehittäjät keskittyä uusia toiminnallisuuksia kehittäessään keskittää muutokset yhteen palveluun. Tällöin projektiin uutena tulevan työntekijän on paljon helpompi selvittää yhden mikropalvelun periaate ja sen hyödyntämät rajapinnat sen sijaan, että täytyisi ottaa haltuun kymmeniä tuhansia rivejä koodia sisältävän monoliitin toimintaperiaate ja tuoda sinne muutoksia rikkomatta koko järjestelmää. Tästä samasta ajatuksesta voidaan johtaa myös se hyöty, että järjestelmän kasvaessa suureksi voidaan sen kehitys jakaa helposti useammalle eri tiimille. Jokainen tiimi voisi olla vastuussa tietyistä palveluista ja keskittyä vain näiden kehittämiseen. Vastaavan toteuttaminen olisi suuressa monoliittisessa paljon haastavampaa, sillä olisi hankalampi vetää selviä rajoja sen suhteen, mikä osio järjestelmästä kuuluu millekin tiimille. Lisäksi tiimien muokatessa samoja osioita olisi koko järjestelmä alttiimpi versionhallinnan konfliktitilanteille, jotka vievät turhaan arvokasta työaikaa.

Useiden ominaisuuksien ja toiminnallisuuksien lisäksi kalustorekisteristä löytyy useampi resurssi, jotka on esitelty aiemmin opinnäytetyössä. Monoliittisessa versiossa kaikkia resursseja hallittaisiin monoliitin itsensä kautta niin, että monoliitti hyödyntäisi resursseja niitä tarvitessaan. Tämä olisikin aivan riittävä ratkaisu, mikäli järjestelmästä riittäisi vain yksi instanssi, mutta mitä tapahtuisi käyttäjämäärien kasvaessa liian suuriksi yhdelle instanssille? Tällaisessa tilanteessa tiedon eheys muodostuisi ratkaistavaksi ongelmaksi. Monoliittisessa järjestelmässä skaalaus tapahtuu monistamalla koko järjestelmä, joten kuuluisiko tällöin monistaa myös tietokannat? Mikäli tietokannoista on useampi eri instanssi, miten tietokantojen keskinäinen sisältö saataisiin pysymään yhtenäisenä? Vaihtoehtoisesti kaikki instanssit käyttäisivät samaa tietokantaa, jolloin monoliitin instanssien täytyisi olla jollain tapaa tietoisia toisistaan, jotta eri instanssien resurssienhyödyntämispyynnöille voitaisiin määritellä jonkinlainen tärkeysjärjestys. Tämä sama ongelma esiintyisi luonnollisesti jokaisen hyödynnettävän resurssin kohdalla. Tässä suhteessa mikropalveluarkkitehtuuri toimiikin erinomaisesti kalustorekisterin kohdalla, sillä jokaiselle resurssille on määriteltynä siitä vastaava palvelu. Tällöin resurssien hyödyntämiseen usean eri instanssin toimesta liittyvät ongelmat voidaan eristää näistä resursseista vastaaville palveluille ja ratkaista niissä. Tällöin resursseja hyödyntävissä palveluissa ei tarvitse ottaa näitä ongelmia huomioon, vaan ne voivat keskittyä oman tehtävänsä toteuttamiseen.

Kalustorekisterin useista resursseista ja toiminnallisuuksista johtuen järjestelmässä voi esiintyä tilanteita, joissa jokin tietty teknologia vastaisi järjestelmässä esiintyvään tarpeeseen monoliittiseen järjestelmään valittua teknologiaa paremmin. Tästä syystä mikropalveluarkkitehtuuri tarjoaa joustavuutensa puolesta paremmat mahdollisuudet tehtäväkohtaisesti sille parhaiten sopiva teknologia.

Edellä mainitut seikat huomioon ottaen mikropalveluarkkitehtuuri soveltuu hyvin kalustorekisterille tarjoten selviä hyötyjä verrattuna monoliittiseen arkkitehtuuriin. Varsinkin järjestelmän kasvaessa suuremmaksi ja ominaisuuksien lisääntyessä, mikropalveluarkkitehtuurin hyödyt kasvavat entisestään. Negatiivisena puolena on luonnollisesti ylimääräinen työ siirtymää tehdessä, mahdollisesti lisääntyvä monimutkaisuus palveluiden määrän kasvaessa sekä palveluiden keskinäisen kommunikoinnin toimintaan saattaminen. Palveluiden keskinäiseen kommunikointiin on kuitenkin olemassa opinnäytetyössä esiteltyjä ratkaisuja, jotka helpottavat mikropalveluiden käyttöönottoa ja säästävät kehittäjiltä paljon työtä kommunikointiin liittyen. Ylimääräinen työ siirtymäajalta voidaan kuitenkin katsoa olevan investointi, joka alun jälkeen tulee helpottamaan työskentelyä järjestelmän parissa sekä tarjoamaan paremman kokemuksen käyttäjilleen.

5.3 Tulosten saavuttaminen

Tavoitteena oli jakaa kalustorekisteri vaatimusten ja ominaisuuksien perusteella erilliseksi palveluiksi hyödyntäen mikropalveluarkkitehtuurin hyviä puolia sekä välttää tästä mahdollisesti seuraavia ongelmia. Tuloksena tästä syntyi arkkitehtuurikuvaus sekä palvelujako perusteineen, miksi järjestelmä pitäisi jakaa juuri kyseisiin palveluihin. Opinnäytetyössä on aiemmin kerrottu mikropalveluarkkitehtuurin hyvistä ja huonoista puolista verrattuna monoliittiseen arkkitehtuuriin, sekä pyritty kertomaan missä tapauksissa mikropalveluarkkitehtuuri on sopiva valinta.

Tässä osiossa peilataan mikropalveluarkkitehtuurin ja monoliittisen arkkitehtuurin eroja kalustorekisteriin. Tavoitteena on saada varmistus, saavutetaanko valitulla palvelujaolla mikropalveluiden tarjoamat hyödyt verrattuna monoliittiseen ohjelmistoon.

Ensimmäisenä esiin tulevat erot kehitystyössä, sillä muita etuja on mahdotonta saavuttaa ilman niiden eteen tehtävää kehitystyötä. Palvelut on pyritty jakamaan loogisiin kokonaisuuksiin yhden vastuun periaatetta noudattaen, niin että jokainen palvelu on mahdollisimman itsenäinen. Tämä helpottaa uusien kehittäjien mukaan pääsyä, sillä he voivat katsoa järjestelmästä kokonaiskuvan ja keskittyä siitä yhtä tehtävää suorittavaan palveluun. Muista palveluista ei tarvitse tietää rajapintojen lisäksi mitään muuta teknistä tietoa, jolloin koko järjestelmää ei tarvitse sisäistää ennen ensimmäisenkään muutoksen tekemistä.

Palveluiden itsenäisyys ja looginen jako auttavat kehitystyössä sisään pääsemisen lisäksi vikojen ja testauksen eristämisessä mikropalveluperiaatteiden mukaisesti. Valitussa palvelujakomallissa resursseihin tapahtuvat muutokset on saatu eristettyä niistä vastuussa oleville palveluille sen sijaan, että esimerkiksi kalustorekisteritietokannan teknologian vaihtuessa täytyisi muuttaa jokaista kalustorekisteriltä tietoa tarvitsevaa palvelua. Täydellistä riippumattomuutta on hankala tai jopa mahdoton saavuttaa, sillä kalustorekisterin tapauksessa itse kalustorekisteripalvelu on hyvin suuressa roolissa. Muut palvelut pitäisikin toteuttaa niin, että kalustorekisteripalvelun kaatuessa nämä muut palvelut eivät kaatuisi, vaan osaisivat käsitellä virhetilanteet ja ilmoittaa käyttäjälle tästä. Tämä on kuitenkin huomattavasti parempi vaihtoehto, sillä tällaisessa tilanteessa käyttäjä pääsee kuitenkin järjestelmään ja saa tiedon, että jokin on vialla. Monoliittisessa arkkitehtuurissa taas koko järjestelmä kaatuisi, eikä käyttäjä saisi välttämättä mitään tietoa järjestelmän tilasta. Samassa vikatilanteessa myös kehittäjien on helpompi ja nopeampi paikantaa vikatilanne, sekä tarvittaessa asettaa tilalle väliaikaisesti vanhempi versio kalustorekisteripalvelusta.

Edellä mainitut asiat huomioon ottaen mikropalveluarkkitehtuurista saadaan hyötyjä kalustorekisteriin nykyisellä palvelujaolla ainakin kehitystyön ja luotettavuuden osalta. Skaalautumisen osalta edut ovat myöskin yhtä selvät. Kalustorekisteripalvelusta löytyy opinnäytetyössä esitelty tuonti/vienti –palvelu, jota on tarkoitus käyttää suurien datamäärien siirtoon järjestelmään tai siitä ulos. Mikäli tästä palvelusta on järjestelmässä vain yksi instanssi, palvelu voi ruuhkautua hyvinkin helposti jo yhden käyttäjän suorittamasta suuren datamäärän tuonnista. Tällaisessa tilanteessa mono-

liittisen järjestelmän ainoana skaalautumisvaihtoehtona olisi luoda koko järjestelmästä toinen instanssi, joka aiemmin opinnäytetyössä kerrottujen mahdollisten ongelmien lisäksi lisää myös kulutusta paljon tarvetta suurempia määriä. Mikropalveluarkkitehtuuriin perustuvassa kalustorekisterissä voidaan luoda tuonti/vienti –palvelusta niin monta instanssia kuin on tarpeellista ilman, että muita palveluita täytyisi monistaa samalla. Tehokkaampi skaalaaminen ei ainoastaan pienennä kulutusta, mutta se parantaa myös käyttökokemusta jakamalla kuormaa eri palveluille ja skaalautumalla tehokkaammin tarpeen mukaan.

Joustavuuden ja teknologisen vapauden suhteen mikropalveluarkkitehtuurista taas olisi hyötyä oikeastaan lähes millä tahansa palvelujakoperusteella, sillä yksittäisille mikropalveluille on joka tapauksessa mahdollista valita teknologia käyttötarkoituksen mukaan. Kuitenkin hyödyntämällä yhden vastuun periaatetta ja jakamalla mikropalvelut loogisesti, saadaan teknologisesta vapaudesta enemmän irti. Hakupalvelu voisi olla osana esimerkiksi käyttöliittymää, eikä omana palvelunaan, jolloin käyttöliittymälle valittu teknologia määrittäisi myös hakupalvelun teknologiavalinnan. Tällöin siis huonolla palvelujakoperiaatteella voidaan rajata omaa vapautta ja mahdollisuuksia hyödyntää parhaiten tehtävään sopivaa teknologiaa. Juuri tämä vapaus teknologiavalinnan suhteen on yksi syistä, miksi hakupalvelu ja tuonti/vienti –palvelu ovat erotettu omiksi kokonaisuuksikseen, vaikka ensin mainittu voisi olla osana käyttöliittymää ja jälkimmäinen osana kalustorekisteripalvelua.

Valitusta palvelujaosta voi olla edellä mainittujen asioiden lisäksi hyötyä myös muissa projekteissa ja järjestelmissä. Esimerkiksi autentikointi perustoiminnallisuus, joka toistuu useissa erilaisissa järjestelmissä. Monoliittisissa järjestelmissä tämä täytyy toteuttaa joka kerta erikseen, jolloin samaa koodia toistetaan useissa eri järjestelmissä. Kalustorekisterin autentikointipalvelusta taas saadaan järkevillä rajapinnoilla yleiskäyttöinen palvelu, jota voidaan hyödyntää useissa eri järjestelmissä. Sama koskee myös esimerkiksi tuonti/vienti –palvelua, eli suurien tietomäärien käsittelyä ja muuntamista johonkin toiseen muotoon. Luomalla tällaisia yleiskäyttöisiä palveluita helpotetaan ja nopeutetaan kehittäjien työtä, sillä uutta järjestelmää luodessa perustoi-

minnallisuudet saadaan jo valmiiden ja testattujen palveluiden avulla. Tämä luonnollisesti säästää myös yritykselle rahaa sekä tarjoaa mahdollisuuden luoda entistä nopeammin demototeutuksia toimivilla perusominaisuuksilla.

Mikropalveluiden ja monoliittisten järjestelmien eroja vertailtaessa mikropalveluilla oli myös huonoja puolia. Kaikilta huonoilta puolilta ei voida tietenkään kokonaan välttyä, mutta hyvällä palvelujaolla ja palveluiden suunnittelulla näiden huonojen puolien merkitystä ja vaikutusta järjestelmän luotettavuuteen ja käyttökokemukseen voidaan vähentää. Yhdeksi huonoksi puoleksi mikropalveluissa oli mainittu järjestelmän lisääntynyt monimutkaisuus, joka kasvaa palveluiden määrän ja niiden välisten yhteyksien myötä. Kalustorekisterissä on tällä hetkellä varsin maltillinen määrä mikropalveluita, joista jokaisella on oma selkeä tehtävänsä, jolloin järjestelmä ei ole vielä kovin monimutkainen. Esimerkiksi järjestelmän palveluita kuvaava arkkitehtuurikuvauskin on vielä tällä hetkellä sellainen, että sen avulla on mahdollista nähdä miten palvelut toimivat yhdessä luodakseen suuremman kokonaisuuden. Palveluiden välistä kommunikointia on pyritty helpottamaan rajapintapalvelun avulla, jolloin jokaisen muun palvelun tarvitsee tietää ainoastaan rajapintapalvelun olemassaolosta. Järjestelmä voi toki muuttua monimutkaisemmaksi mahdollisten lisäominaisuuksien ja uusien palveluiden myötä, mutta samaa jakoperiaatetta noudattamalla näiden ei pitäisi muodostua ongelmaksi.

Samasta maltillisesta määrästä johtuen myöskään palveluiden välisen kommunikoinnin ei pitäisi nostaa viivettä merkittävän suureksi. On totta, että joka tapauksessa monoliittisen järjestelmän sisällä olevat kutsut kulkevat nopeammin kuin palveluiden kutsut toisilleen, mutta tämä on hallittavissa järkevillä rajapinnoilla ja miettimällä tarkkaan palvelukutsujen kulkemat reitit. Mikäli palvelujaoksi olisi valittu vaikkapa malli, jossa jokaisesta luokasta muodostetaan oma palvelunsa, olisi palveluiden ja tällöin myös näiden välisten kutsujen määrä moninkertainen nykyiseen nähden.

6 Pohdinta ja jatkokehitys

Opinnäytetyön tavoitteena oli tutkia erilaisia tapoja jakaa ohjelmisto mikropalveluihin, vertailla näiden mallien hyviä ja huonoja puolia toisiinsa, sekä luoda tämän tutkimustyön pohjalta palvelujakomalli ja arkkitehtuurikuvaus olemassaolevalle monoliittiselle ohjelmistolle. Tätä tutkimusta toteutettiin pääasiassa tutkimalla alan kirjallisuutta ja artikkeleita liittyen mikropalveluihin, monoliittisiin ohjelmistoihin sekä erilaisiin tapoihin jakaa ohjelmisto mikropalveluiksi. Joissakin artikkeleissa oli mukana myös muiden ihmisten kokemuksia monoliittisen ohjelmiston pilkkomisesta mikropalveluihin, joista oli hyötyä esimerkiksi palvelujakoa suunnitellessa.

Tämän työn pohjalta saatiin tavoitteiden mukaisesti luotua kalustorekisterille palvelujakomalli, jossa on pyritty yhdistämään useamman eri mallin hyvät puolet sekä minimoimaan mahdolliset negatiiviset puolet verrattuna monoliittiseen arkkitehtuuriin. Näin kalustorekisterin kehitykseen saadaan lisää joustavuutta, sekä loppukäyttäjille parempaa toimintavarmuutta ja skaalautumista ilman korkeaa viivettä eri toiminnallisuuksissa.

Palvelujakomallin lisäksi tuloksena saatiin myös useisiin eri lähteisiin perustuva vertailu arkkitehtuurimallien eroista ja erilaisista tavoista jakaa ohjelmisto pienempiin osiin. Eri malleista ja jakoperusteista on koottu muiden ihmisten kokemuksia ja havaintoja lukuisista eri asioista, joita täytyy ottaa huomioon arkkitehtuuria suunitellessa.

Ennen opinnäytetyötä mikropalvelut olivat minulle tuttu lähinnä terminä, mutta en ollut perehtynyt aiheeseen juuri ollenkaan, enkä hyödyntänyt mikropalveluita ohjelmointityössä. Tästä syystä minun täytyikin aloittaa tutkimalla mitä mikropalvelut ovat ja miten niiden käyttö eroaa sellaisesta ohjelmoinnista, jota olen jo aikaisemmin tehnyt. Tämä ohjasi minua tutkimaan myös sitä, millaisissa tapauksissa mikropalveluarkkitehtuurista on hyötyä ja milloin haittaa, sillä uuteen teknologiaan tutustuttaessa saattaa keskittyä liikaa hyviin puoliin ja haluta soveltaa sitä kaikkeen, vaikka se ei olisikaan joka tilanteeseen sopiva valinta. Tämän vuoksi

opinnäytetyössä saavutettuja tuloksia peilattiin myös kalustorekisteriin, sillä palvelujaon tekeminen olisi täysin turhaa työtä, mikäli monoliittisen arkkitehtuurin olisi huomattu olevan sopivampi vaihtoehto kalustorekisterin tapauksessa.

Palvelujakoperusteita määritellessä vastaan tuli useita erilaisia malleja, sekä useita erilaisia mielipiteitä liittyen näihin malleihin. Ajoittain vastaan tuli myös ristiriitaista tietoa, kun jotakin palvelujakomallia väitettiin yhdessä lähteeksi helpoksi ja vähäistä suunnittelua vaativaksi, kun taas toisessa mallissa sanottiin saman mallin vaativan vankkaa tietämystä ja kokemusta sen hyödyntämiseksi. Erilaisten mallien ja niihin liittyvien mielipiteiden tutkiminen oli kuitenkin mielenkiintoista, sillä ne auttoivat ymmärtämään paremmin mikropalveluarkkitehtuuria, sekä havainnoimaan muiden ihmisten kokemuksia erilaisten mallien hyvistä ja huonoista puolista erilaisissa käyttötapauksissa.

Opinnäytetyössä onnistuttiinkin keräämään tietoa useista eri lähteistä, jolloin oli mahdollista huomata useiden ihmisten tehneen saman kaltaisia havaintoja joistakin asioista. Esimerkiksi mikäli monet kokivat jossakin palvelujakomallissa tietyn piirteen muodostuvan ongelmaksi, muuttuisi se oletettavasti ongelmaksi myös kalustorekisterissä ja tämä täytyisi ottaa huomioon palvelujakoa tehdessä.

Kaikista malleista taas ei löytynyt yhtä paljon tietoa kuin olisi ollut tarve, jolloin yksittäiset havainnot saattavat perustua yhden henkilön mielipiteisiin laajemman konsensuksen sijaan.

Muiden ihmisten kokemukset voitaisiin kuitenkin nähdä myös rajoituksina, sillä eri mallien vertailu perustui hyvin pitkälti ohjelmistojen teoreettiseen toimintaan ja ihmisten kokemuksiin vaikutti varmasti myös heidän ympäristönsä ja ohjelmisto, joihin he palvelujakomalleja soveltivat. Joitakin malleja oli vertailtu käyttäen pohjana samaa ohjelmistoa, jolloin mallien eroavaisuudet tulivat selvemmin esille, mutta useita eri malleja kattavaa vertailua ja varsinkaan eri tavoin samaa ohjelmistoa toteutettuna ei löytynyt. Näitä muiden kokemuksia peilattiin kuitenkin opinnäytetyössä kalustorekisteriin ja samalla pyrittiin vertailemaan mallien sopivuutta juuri kalustorekisterin tapauksessa vaatimukset ja toiminnallisuudet

huomioon ottaen. Tämä oli kuitenkin samalla opinnäytetyön suurin haaste, sillä minun piti sisäistää monia eri palvelujakomalleja useita lähteitä hyödyntäen ja miettiä kuinka kalustorekisterin saisi toimimaan kyseisten mallien periaatteita noudattaen.

Saatuja tuloksia voidaan hyödyntää ainakin kalustorekisterin tapauksessa ja sen kehitystyössä. Tuloksia voidaan hyödyntää myös muissa projekteissa, joissa pohditaan siirtymistä monoliittisesta arkkitehtuurista mikropalveluarkkitehtuuriin. Vaikka opinnäytetyössä mikropalveluarkkitehtuuria ja erilaisia palvelujakotapoja tutkitaankin hyvin pitkälti kalustorekisterin näkökulmasta, ovat saadut tulokset eri mallien eroista päteviä myös muissa tapauksissa. Jokaisessa tällaisessa projektissa täytyy joka tapauksessa aluksi tutkia mikropalveluarkkitehtuurin soveltuvuus kyseisessä tapauksessa, sekä päättää palvelujakomalli, joita opinnäytetyössä on käsitelty useisiin eri lähteisiin pohjaten.

Mikäli kalustorekisteriä jatkokehitetään ja siihen halutaan tuoda lisää ominaisuuksia, kannattaisi jokaisen uuden ominaisuuden kohdalla pohtia miten ominaisuus muutettaisiin mikropalveluarkkitehtuuriin ja valittuun malliin sopivaksi. Jokainen uusi resurssi kannattaisi yhdistää vain sitä hallinnoivaan palveluun ja määritellä tälle palvelulle rajapinnat, joita muut voisivat hyödyntää. Ominaisuuksien kohdalla tulisi myös pohtia tapauskohtaisesti sitä, tarvitseeko ominaisuus pilkkoa useammaksi kuin yhdeksi palveluksi yhden vastuun periaatetta noudattaen, sekä palveluiden yhteyksiä muuhun järjestelmään.

Opinnäytetyön aikana koen oppineeni todella paljon uusia asioita ohjelmistokehityksen sekä varsinkin mikropalveluiden arkkitehtuurin saralta. Koen näistä tiedoista olevan hyötyä minulle myös jatkossa, sillä myös muita ohjelmistoja voidaan haluta muuttaa monoliittisista ohjelmistoista mikropalveluihin perustuviksi ohjelmistoiksi. Lisäksi myös uusien ohjelmistojen kohdalla täytyy aina tehdä valinta arkkitehtuurin suhteen, jolloin minulla on jo tietämystä aiheesta.

Lähteet

Akhtar, J. 2018. Microservices Introduction (Monolithic vs. Microservice Architecture). Viitattu 9.10.2019. https://dzone.com/articles/microservices-1-introduction-monolithic-vs-microse

Barashkov, A. 2018. Microservices vs. Monolith Architecture. Viitattu 9.10.2019. https://dev.to/alex_barashkov/microservices-vs-monolith-architecture-4l1m

Bruce, K. 2002. Foundations of Object-oriented Languages: Types and Semantics. Viitattu 31.10.2019. https://books.google.se/books?id=9NGWq3K1RwUC&pg=PA18&redir_esc=y#v=onepage&q&f=false

Fowler, M. 2015. MonolithFirst. Viitattu 9.10.2019. https://www.martinfowler.com/bliki/MonolithFirst.html

Gnatyk, R. 2018. Microservices vs Monolith: which architecture is the best choice for your business?. Viitattu 25.11.2019. https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/

Gupta, T. 2018. Analyzing Polyglot Microservices. Viitattu 9.10.2019. https://medium.com/capital-one-tech/analyzing-polyglot-microservices-f6f159a1a3e7

Jens. 2017. Why You Should Think Twice Before Even Considering Polyglot Microservices. Viitattu 9.10.2019. https://codeboje.de/polyglot-microservices/

Liu, D. 2014. Eureka at a glance. Viitattu 16.10.2019. https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance

Microservices. 2019. Artikkeli IBM:n sivuilla 30.5.2019. Viitattu 6.10.2019. https://www.ibm.com/cloud/learn/microservices

Microservices vs Monolithic Architecture. N.d. Artikkeli Mikropalveluiden ja monoliittisen arkkitehtuurin välillä MuleSoftin sivuilla. Viitattu 9.10.2019. https://www.mulesoft.com/resources/api/microservices-vs-monolithic

Miettinen, J. 2019. Palvelujakoraportti. Combitech Oy:lle tehty raportti kalustorekisterin palvelujaosta. Viitattu 22.10.2019.

Morris, B. 2015. How big is a microservice?. Viitattu 23.10.2019. https://www.ben-morris.com/how-big-is-a-microservice/

Richardson, C. N.d.a. Pattern: Decompose by business capability. Artikkeli ohjelmiston jakamisesta mikropalveluihin liiketoimintamahdollisuuksien mukaan. Viitattu 5.11.2019 https://microservices.io/patterns/decomposition/decompose-by-business-capability.html

Richardson, C. N.d.b. The Scale Cube. Viitattu 10.10.2019. https://microservices.io/articles/scalecube.html

Richardson, C. N.d.c. What are microservices?. Viitattu 6.10.2019. https://microservices.io/

Singh, J. 2018. The What, Why, and How of a Microservices Architecture. Viitattu 16.10.2019. https://medium.com/hashmapinc/the-what-why-and-how-of-a-microservices-architecture-4179579423a9

Spring Cloud Netflix – Ribbon. N.d. Artikkeli Ribbon ohjelmiston toiminnasta. Viitattu 16.10.2019. https://www.nexsoftsys.com/articles/spring-cloud-netflix-ribbon.html

Strategies for Decomposing an Application into Services. 2017. Artikkeli ohjelmiston jakamisesta palveluihin. Viitattu 5.11.2019. https://freecontent.manning.com/strategies-for-decomposing-an-application-into-services/

Swanberg, K. 2019. Why Scalability Matters for Your App. Viitattu 6.10.2019. https://www.koombea.com/blog/why-scalability-matters-for-your-app/

Tietoja meistä. N.d. Combitechin esittelysivu. Viitattu 16.10.2019. https://combitech.fi/tietoja/

Tietojärjestlemiä vaativiin tarpeisiin. N.d. Combitechin puolustusosion esittelysivu. Viitattu 16.10.2019. https://combitech.fi/tietoja/

Tilkov, S. 2015. Don’t start with a monolith. Viitattu 10.10.2019. https://martinfowler.com/articles/dont-start-monolith.html

Watts, S & Shiff,L. 2018. An Overview of Monolithic vs Microservices Architecture (MSA). Viitattu 9.10.2019. https://www.bmc.com/blogs/microservices-architecture/

Wicksell, T., Cellucci, T., Yuan, H., Bross, A., Yap, N. & Liu, D. 2018. Netflix OSS and Spring Boot — Coming Full Circle. Viitattu 16.10.2019. https://medium.com/netflix-techblog/netflix-oss-and-spring-boot-coming-full-circle-4855947713a0

Wittmer, P. 2018. Monolithic vs Microservices Architecture – Why Microservices Win. Viitattu 10.10.2019. https://www.tiempodev.com/blog/monolithic-vs-microservices-architecture/