Maisterintutkielma

Tietojenkäsittelytieteen maisteriohjelma

Kirjallisuuskatsaus ja tapaustutkimus API-hallinnasta mikropalveluarkkitehtuurissa

Roni Lindholm

26.11.2022

Matemaattis-luonnontieteellinen tiedekunta

Helsingin yliopisto

Yhteystiedot

PL 68 (Pietari Kalmin katu 5)
00014 Helsingin yliopisto

Sähkopostiosoite: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen maisteriohjelma

Tekijä — Författare — Author

Roni Lindholm

Työn nimi — Arbetets titel — Title

Kirjallisuuskatsaus ja tapaustutkimus API-hallinnasta mikropalveluarkkitehtuurissa

Ohjaajat — Handledare — Supervisors

Ph.D. Antti-Pekka Tuovinen, Ph.D. Matti Luukkainen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Maisterintutkielma	26.11.2022	51 sivua, 7 liitesivua

Tiivistelmä — Referat — Abstract

Verkon avulla tarjottavien rajapintojen käyttö ja niiden tarjoaminen kolmannen osapuolen käyttäjille on lisääntynyt sovelluskehityksessä viimevuosina suuresti. Mikropalveluarkkitehtuurin yleistyminen on osaltaan lisännyt sovelluksien hyödyntämien rajapintojen määrää merkittävästi ja luonut tarpeen työkaluille ja toimintatavoille, joilla kasvanutta rajapintojen määrää voidaan hallita ja ylläpitää tehokkaasti.

Tämä tutkimus koostuu kahdesta erillisestä osasta. Ensimmäisessä osassa kartoitetaan kirjallisuuskatsauksen avulla API-hallinnan tavoitteita. Toisessa osassa tutustutaan tapaustutkimuksen avulla, kuinka kirjallisuuskatsauksessa tunnistettuja tavoitteita on toteutettu ja kuinka ne koetaan käytännön sovelluskehityksessä.

Tutkielman keskeisenä tuloksena on joukko API-hallinnan tavoitteita ja käsitys siitä, kuinka niistä merkittävimmät ovat toteutettu tapausorganisaatioissa. API-hallinnan käytäntöjen lisäksi haastattelujen avulla selvitettiin, kuinka nämä käytännöt koetaan sovelluskehittäjien keskuudessa. Tuloksien perusteella API-hallinta voi lisätä sovelluskehittäjien työmäärää, mutta samalla se koetaan suurien organisaatioiden tuotteiden parissa lähes välttämättömänä toimintatapojen yhtenäistäjänä.

Yhteenvetona tutkielman perusteella API-hallinta tarjoaa tarpeellisia työkaluja ja toimintatapoja rajapintojen kanssa työskentelyyn, kun rajapintoja on suuri määrä tai kun niitä tarjotaan organisaation ulkopuolisille sovelluskehittäjille. Käytännössä tapausorganisaatioiden rajapintojen hallinta on toteutettu vaihtelevin käytännöin eikä sitä koeta aivan pienessä organisaatiossa lähtökohtaisesti mielekkääksi.

ACM Computing Classification System (CCS)
Information systems → World Wide Web → Web services → RESTful web services

Computer systems organization → Architectures → Distributed architectures → Client-server architectures

Avainsanat — Nyckelord — Keywords

mikropaveluarkkitehtuuri, API-hallinta, REST, GraphQL

Säilytyspaikka — Förvaringsställe — Where deposited

Helsingin yliopiston kirjasto

Muita tietoja — övriga uppgifter — Additional information

Ohjelmistojen opintosuunta

Sisällys

1 Johdanto

Mikropalveluarkkitehtuuri (engl. *microservice architecture*) on yleistynyt viime aikoina ketterien ohjelmistokehitysmenetelmien rinnalla. Mikropalveluarkkitehtuurissa ohjelmiston osa-alueet jaetaan pieniksi, itsenäisiksi kokonaisuuksiksi yhden suuren sovelluskokonaisuuden sijaan [3]. Oikein toteutettuna mikropalveluarkkitehtuuri tukee ketterän ohjelmistokehityksen tarpeita ja mahdollistaa nopean reagoinnin esimerkiksi uusien toiminnallisuuksien luomiseen.

Mikropalveluiden käyttäminen luo kuitenkin joukon uusia ongelmia, sillä ohjelmistojen eri osien täytyy jollakin keinoin kommunikoida mikropalveluiden kanssa. Jos sovellus on yhtä suurta kokonaisuutta, eli monoliitti, ei kommunikointiin tarvitse kiinnittää erityistä huomiota, sillä sen voi tehdä metodikutsuin saman prosessin sisällä. Tämä ei kuitenkaan ole mikropalveluissa mahdollista, sillä mikropalvelut voivat sijaita esimerkiksi pilvipalvelussa, jolloin niiden kanssa kommunikointi tapahtuu verkon välityksellä.

Verkossa sovelluksien väliseen kommunikointiin käytetään usein rajapintoja ja HTTP-protokollaa. Rajapinnasta käytetään tässä tutkielmassa englannin kielistä termiä API (engl. *Application Programming Interface*) ja sillä tarkoitetaan rajapintaa, jonka avulla kaksi sovellusta voi kommunikoida verkon avulla toisilleen. Sovelluksien välinen kommunikointi on myös mahdollista toteuttaa esimerkiksi viestinvälittäjää (engl. *message broker*) hyödyntämällä, mutta tässä tutkielmassa perehdytään erityisesti verkon avulla käytettäviin rajapintoihin.

Mikropalveluiden yleistyessä ja sovelluksen kasvaessa, kasvaa usein myös ylläpidettävien rajapintojen määrä. Jossakin vaiheessa rajapintojen käyttöön ja kehittämiseen voi olla mielekästä luoda koko tuotetta ylläpitävän organisaation laajuisia käytänteitä, kuten ketkä rajapintoja saa käyttää, kuinka niiden käyttöä seurataan sekä muita vastaavia toimintoja.

API-hallinta (engl. *API management*) pyrkii tarjoamaan menetelmät ja työkalut, joilla organisaation tarjoamia rajapintoja voidaan julkaista organisaation sisäisille tai kolmannen osapuolen sovelluskehittäjille. Hyvin toteutettu API-hallinta mahdollistaa rajapintojen ketterän hyödyntämisen ja tarjoaa työkalut esimerkiksi rajapintojen versiointiin, pääsynhallintaan ja kuormituksen tasaamiseen [17].

Rajapintojen hallinnan ei pitäisi olla vain ylimääräinen kerros kompleksisuutta ja byro-

kratiaa, vaan parhaimmillaan se tukee ketterää ohjelmistokehitystä ja mahdollistaa joustavan tavan hallita tuotteita. API-hallinta on websovelluskehityksessä melko uusi käsite, eikä sille ole esimerkiksi tutkimuskirjallisuudessa yleisesti hyväksyttyä määritelmää [17].

API-yhdyskäytävä (engl. *API-gateway*) on yleinen työkalu keskitetyn API-hallinnan toteuttamiseen. API-yhdyskäytävä on ohjelmisto, joka toimii mikropalvelun ja sitä käyttävän tuotteen välillä. Usein API-hallinnasta puhuessa yhdyskäytävä sijaitsee käyttöliittymän ja taustajärjestelmien rajapinnassa, jolloin se tarjoaa yksittäisen sisääntulopisteen (engl. *single point of entry*) järjestelmään. API-yhdyskäytävä ohjaa saapuvat pyynnöt ennalta määrättyjen sääntöjen mukaisesti esimerkiksi mikropalveluille ja välittää palvelun tarjoaman vastauksen takaisin sitä pyytäneelle ohjelmistolle [17]. Kaikki merkittävät pilvipalveluntarjoajat hyödyntävät API-yhdyskäytävää osanaan tarjoamiaan API-hallinnan palveluita. Pilvipalvelujen vaihtoehdoksi on kehitetty myös useita avoimen lähdekoodin API-yhdyskäytäväohjelmistoja.

Tässä tutkielmassa tutustutaan, kuinka ja millä työkaluin suurta joukkoa HTTP-rajapintoja voidaan hallita säilyttäen ketteryyden ja joustavuuden ohjelmistokehityksessä. Tutkimuskirjallisuuden avulla pyritään selvittämään mitä tavoitteita ja toimintoja API-hallinnalla pyritään saavuttamaan. Tutkimuskirjallisuuden lisäksi tutkielmassa kartoitetaan tapaustutkimuksen avulla, kuinka tapausorganisaatioissa API-hallinnan tavoitteet ovat toteutettu sekä kuinka sovelluskehittäjät kokevat API-hallinnan heidän työskentelyssään. Tutkielmaa varten on tehty yhteensä neljä tutkimushaastattelua neljästä eri organisaatioista, jotka tekevät sovelluskehitystä. Haastateltavaksi valikoitui kustakin organisaatioista yksi sovelluskehittäjä tai arkkitehti, kuka työskentelee läheisesti organisaatiossa käytössä olevien rajapintojen kanssa.

API-hallintaa on tärkeää tutkia, sillä rajapintojen hyödyntäminen sovelluskehityksessä ei ole ainakaan vähentymässä. Onkin tärkeää tutkia kuinka organisaatiot voivat työskennellä tehokkaasti jatkuvasti lisääntyvien rajapintojen kanssa ja selvittää mitä asioita API-hallinnalla voidaan ratkoa ja mahdollisesti myös, milloin se ei tarjoa merkittävää hyötyä.

Tutkielma on jaettu seuraavanlaisiin lukuihin: luvussa 2 on luotu katsaus tarvittaviin käsitteisiin ja taustoitettu API-hallintaa. Luvussa 3 on esitelty tutkimusmenetelmien käyttö sekä tutkimuskysymykset. Luvussa 4 on esitetty tutkimustulokset, joita on pohdittu ja jäsennelty luvussa 5. Viimeiseksi luvussa 6 on yhteenveto tutkielmassa käsitellyistä asioista ja esitetään mahdollisia aiheita ja näkökulmia jatkotutkimusta varten.

2 Taustaa

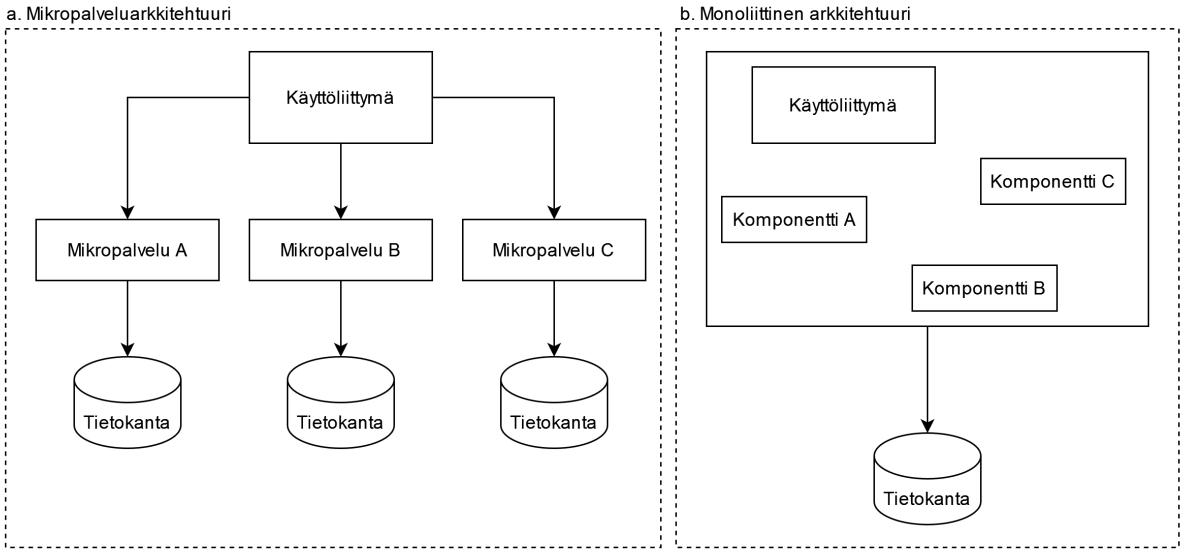
Tämä tutkielma kietoutuu tiukasti mikropalveluarkkitehtuurin ja HTTP-rajapintojen ympärille ja tässä luvussa esitellään tutkielman kannalta oleelliset teknologiat ja käsitteet. Alaluvussa 2.1 esitellään mikropalveluarkkitehtuuri ja alaluvussa 2.2 tutustutaan mikropalveluiden välisessä kommunikoinnissa käytettäviin rajapintoihin. Alaluku 2.3 esittelee API-hallinnan käsitteen, josta esitellään esimerkki alaluvussa 2.4.

2.1 Mikropalveluarkkitehtuuri

Mikropalveluarkkitehtuurissa kehitettävä ohjelmisto pyritään jakamaan useaan pienempään osaan, joita kutsutaan mikropalveluiksi. Mikropalvelut ovat itsenäisiä sovelluksia, jotka toteuttavat vain tietyn sovelluslogiikan osa-alueen [3]. Monoliittisessa sovelluksessa kaikki sovelluksen tarvitsema sovelluslogiikka, kuten käyttöliittymä, tietokanta ja taustajärjestelmät sijaitsevat samassa suoritettavassa tiedostossa. Usein tällaisessa arkkitehtuurissa sovelluslogiikka on jaettu loogisiin kokonaisuuksiin eli komponentteihin tai moduuleihin [20]. Mikropalveluarkkitehtuurissa nämä komponentit voidaan toteuttaa mikropalveluin, jolloin jokainen palvelu toteuttaa vain tietyn toiminnallisuuden itsenäisenä sovelluksena [7].

Kuvassa 2.1a on havainnollistettu mikropalveluarkkitehtuuria, jossa käyttöliittymä tai asiakassovellus hakee tarvitsemansa resurssit useasta mikropalvelusta ja kuvassa 2.1b on havainnollistettuna vastaava tilanne monoliittisessa arkkitehtuurissa, jossa kaikki komponentit ovat osa samaa sovellusta.

Monoliittisessa arkkitehtuurissa komponentteja hyödyntävän sovelluslogiikan ei tarvitse tietää komponenttien sisäisestä toiminnasta. Sama pätee myös mikropalveluarkkitehtuurissa, jossa mikropalvelua hyödyntävän asiakassovelluksen tarvitsee tietää palvelusta vain sen tarjoamat rajapinnat. Mikropalveluita hyödyntäessä jokainen mikropalvelu voi kuitenkin olla toteutettu sovelluskehittäjien parhaaksi näkemillään teknologioilla ja työkaluilla, joka ei monoliittisessa arkkitehtuurissa komponentteja hyödyntäessä ole mahdollista [7]. Komponenttien jakaminen mikropalveluiksi mahdollistaa myös sovelluksen päivittämisen siten, ettei yhden palvelun päivittäminen vaadi koko sovelluksen päivittämistä. Sovelluksen rakentaminen itsenäisitä mikropalveluista tarjoaa myös luonnollisen tavan jakaa so-



Kuva 2.1: Esimerkki mikropalveluarkkitehtuurista (a) ja monoliittisesta arkkitehtuurista (b).

velluskehittäjät eri tiimeihin [3].

Merkittävä ero verratessa mikropalveluarkkitehtuuria monoliittiseen arkkitehtuuriin on mikropalveluiden kanssa kommunikointi. Monoliitissa kommunikointiin on mahdollista käyttää metodikutsuja, sillä usein suuri osa sovelluksen osa-alueista ovat osa samaa sovellusta [7]. Mikropalveluarkkitehtuurissa tämä ei kuitenkaan ole mahdollista, sillä kommunikointi tapahtuu verkon välityksellä rajapinnoin tai esimerkiksi viestijonoin [3].

Mikropalveluiden mahdollistaessa vain yksittäisien sovelluksen osien päivittämisen, voidaan päivityksiä toimittaa asiakkaalle parhaassa tapauksessa entistä useammin. Päivityssyklin nopeuttamiseen ja automatisointiin on kehitetty jatkuvan integraation menetelmiä, joissa testaus ja paketointi pyritään tuomaan osaksi sovelluskehittäjien työtä [5].

2.2 Rajapinnat

Rajapinta eli API tarkoittaa ohjelmointirajapintaa, jonka kautta esimerkiksi sovellus voi kommunikoida ohjelmakirjaston kanssa, tai sovellukset voivat kommunikoida toisilleen verkon välityksellä [2]. Tässä tutkielmassa rajapinnalla tarkoitetaan erityisesti verkossa sovelluksien väliseen kommunikointiin käytettävää menetelmää.

Rajapintaa voi havainnollistaa mieltämällä sen olevan sopimus kahden sovelluksen välillä siitä millä tavoin sovellukset kommunikoivat ja minkälaista dataa kommunikoinnissa käytetään. Lisäksi rajapinta kuvaa, mitä toimintoja rajapinnan avulla on mahdollista tehdä,

kuten esimerkiksi luoda tai poistaa resursseja [2].

Rajapinnat voivat olla tuotteen tai organisaation sisäisiä rajapintoja, jolloin niitä voidaan hyödyntää vain organisaation tai tuotteet sisäisesti. Rajapintoja on myös mahdollista julkaista kolmannen osapuolen käyttäjille, jolloin kuka tahansa tai vain rajapinnan omistajan hyväksymät käyttäjät voivat hyödyntää niitä [2].

Käsitteenä rajapinta ei ota kantaa siihen, kuinka rajapinnan avulla jaetaan dataa ja millä protokollilla. Käytännössä nykyään rajapintojen välillä kommunikointiin käytetään usein HTTP-protokollaa ja data välitetään JSON-tiedostomuodossa (engl. *JavaScript Object Notation*). Yleisiä käytössä olevia arkkitehtuurimalleja rajapintojen toteuttamiseen ovat esimerkiksi REST (engl. *Representational State Transfer*) ja GraphQL (engl. *Graph Query Language*)[2].

Erittäin yleinen tapa toteuttaa rajapinta kahden sovelluksen väliseen kommunikointiin on REST-arkkitehtuuri, hyödyntäen kommunikointiin HTTP-protokollaa ja datan välittämiseen JSON-tiedostomuotoa. REST-arkkitehtuuri ei kuitenkaan itsessään ota kantaa käytettävään protokollaan tai datan muotoon [6]. Rajapinta, joka täyttää kaikki kuusi REST-määritelmän mukaista ominaisuutta, voidaan kutsua RESTful-rajapinnaksi [21]. REST-rajapinnoille on ominaista, että samaa entiteettiä kuvaavat resurssit ovat jaoteltu entiteetin ja yksilöivän osoitteen perusteella. Usein resurssien manipulointi tapahtuu käyttämällä HTTP-verbejä, kuten *POST* resurssien luomista varten ja *GET* lukemista varten. Tämä toimintamalli johtaa kuitenkin helposti tilanteeseen, jossa rajapintaa käyttävä asiakassovellus voi joutua tekemään suuren määrän kutsuja eri rajapintoihin saadakseen kaikki tarvitsemansa resurssit [21].

Nykyisin suosiotaan nostava GraphQL-kyselykieli mahdollistaa asiakassovelluksille tavan määritellä, mitä resursseja he tarvitsevat, jolloin kaikki resurssit voidaan hakea yhdellä rajapintakutsulla. Tämä on mahdollista GraphQL-rajapinnoissa määritellyn skeeman avulla, joka kuvaa mitä entiteettejä ja niiden kenttiä rajapinnasta on saatavilla. GraphQL-spesifikaatio ei ota kantaa palvelinpuolella kyselykieltä tulkitsevaan ohjelmistoon, mutta palvelin palauttaa pyydetyt resurssit JSON-muodossa esimerkiksi HTTP-protokollan avulla. Kyselykieli on alun perin Facebookin sisäisesti kehittämä, mutta myöhemmin se on julkaistu avoimena koodina ja nykyisin sitä ylläpitää voittoa tavoittelematon yhdistys [10].

Rajapinnat voivat olla julkisesti saatavilla koko Internetistä, mutta usein niihin pääsyä halutaan kuitenkin rajata. Rajapintakutsussa tunnistautuminen voi yksinkertaisemmillaan tapahtua hyödyntämällä salaista avainta, joka liitetään jokaiseen rajapintakutsuun ja jon-

ka avulla palvelimen on mahdollista tunnistaa rajapintaa käyttävä sovellus ja evätä pääsy tunnistautumattomilta käyttäjiltä [2]. Kutsuihin liitettävä avain voi joissakin tapauksissa sisältää myös lisätietoja rajapintaa käyttävästä sovelluksesta. Esimerkiksi JWT-avain (engl. *JSON Web Token*) voi sisältää tietoja tunnistautuneesta käyttäjästä [14]. JWT-avaimen sisältämää tietoa voidaan käyttää pääsynhallintaan ja rajata pääsyä eri resursseihin tunnistautuneen käyttäjän perusteella.

2.3 API-hallinta

Rajapintojen ja niitä hyödyntävien sovelluksien määrän kasvaessa nousee esiin tarve työkaluille ja käytänteille hallita rajapintoja ketterästi ja joustavasti ottaen huomioon kuitenkin tietoturvan [2]. Erityisesti mikropalveluarkkitehtuurissa käytössä olevien rajapintojen määrä voi kasvaa suureksi ja niiden käyttöä helpottamaan pyritään hyödyntämään jonkinlaista keskitettyä API-hallintaa. API-hallinnan merkitys korostuu erityisesti, jos rajapintoja tarjotaan kolmannen osapuolen käyttäjille esimerkiksi integraatioita varten, tai jos organisaation liiketoiminta perustuu rajapintojen tarjoamiseen ja niiden käytöstä laskuttamiseen.

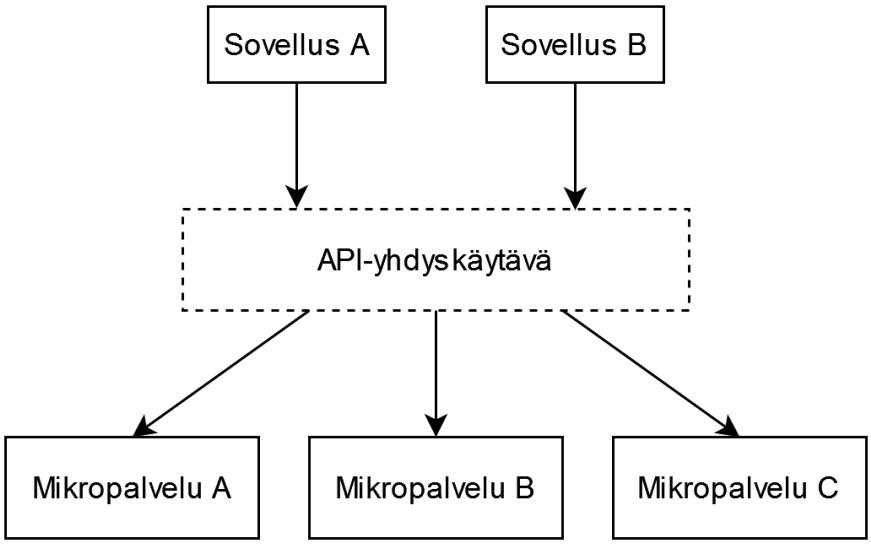
Tutkimuskirjallisuudesta ei löydy yhtä yleisesti hyväksyttyä ja kattavaa määritelmää API-hallinnalle ja sen tarjoamille toiminnoille [17]. Suhteellisen tuore ja kattava ehdotus API-hallinnan määrittelemiseksi on seuraava:

> API-hallinnan tavoitteena on tarjota organisaatiolle keskitetty alusta rajapintojen julkaisuun, monitorointiin ja pääsynhallintaan. API-hallinnan toiminnot, kuten elinkaarenhallinta, tunnistautuminen ja pääsynhallinta, valvonta ja rajapintojen käytön seuranta ovat usein toteutettu hyödyntämällä keskitettyä alustaa, jonka keskeisenä komponenttina toimii API-yhdyskäytävä [17].

API-hallinnan tarjoamat toiminnot ovat mahdollisia toteuttaa myös ilman API-yhdyskäytävää tai muuta vastaavaa keskitettyä alustaa. On kuitenkin huomattava, että esimerkiksi Microsoft ja Amazon tarjoavat API-hallinnan työkalujaan vain API-yhdyskäytävää hyödyntävänä kokonaisuutena [19, 1]. Tässä tutkielmassa keskitytään erityisesti API-yhdyskäytävää hyödyntäviin menetelmiin toteuttaa API-hallinta ja tutkitaan, kuinka API-yhdyskäytävää on hyödynnetty tapausorganisaatioiden sovelluskehityksessä vai onko sitä hyödynnetty lainkaan.

Keskitetyn API-hallinnan vahvuus on sen yhdyskäytäväohjelmiston tarjoama yksittäinen sisääntulopiste rajapintojen kutsumista varten. Tämä piste on luonnollinen paikka toteuttaa eri API-hallinnan toimintoja ja näin ne voidaan eriyttää erilleen sovelluslogiikasta, jolloin parhaassa tapauksessa sovelluksien ei tarvitse itse huolehtia esimerkiksi tunnistautumisesta tai lokitietojen tallentamisesta [18]. Kuvassa 2.2 on havainnollistettu asiakassovelluksien, mikropalveluiden ja API-yhdyskäytävän välistä suhdetta.

Keskitetyt API-hallinnan ratkaisut tarjoavat erilaisia käyttöliittymiä organisaatiolle rajapintojen hallintaan, sekä rajapintoja hyödyntäville sovelluskehittäjille. Rajapintoja hyödyntäville sovelluskehittäjille suunnattu portaali (engl. *developer portal*) tarjoaa dokumentaatiota saatavilla olevista rajapinnoista. Portaalissa voi olla dokumentaatiota ja esimerkkejä rajapintojen käytöstä [18] sekä mahdollisuus rekisteröityä tai pyytää käyttöoikeutta rajapintoihin, jos ne eivät ole avoimesti saatavilla. Hallintaportaalin (engl. *management plane* [18]) avulla rajapintoja tarjoava organisaatio voi ylläpitää API-yhdyskäytävän konfiguraatiota, sekä hallinnoida rajapintoihin tarvittavia käyttöoikeuksia ja käyttäjiä tai julkaista uusia versioita rajapinnoista [26].



Kuva 2.2: API-yhdyskäytävä välittää pyyntöjä sovellukselta mikropalveluille, jolloin sovelluksen ei tarvitse tietää mikropalvelun sijaintia.

Yleisiä ja laajassa käytössä olevia API-yhdyskäytäväohjelmistoja ovat esimerkiksi Amazonin tarjoama AWS-pilvipalvelussa toimiva Amazon API Gateway [1] ja Microsoftin vastaava Azure-pilvipalvelussa toimiva API-yhdyskäytävä [19]. Avoimen lähdekoodin API-yhdyskäytäviä ovat esimerkiksi Tyk [26] ja Gravitee [11], jotka molemmat tarjoavat ohjelmistostaan myös maksullisia versioita.

Toiminnallisuuksiltaan eri yhdyskäytäväohjelmistot voivat erota suuresti toisistaan, mutta tyypillisesti API-yhdyskäytävässä on mahdollisuus esimerkiksi yksinkertaiseen tunnistautumiseen ja pääsynhallintaan sekä lokitietojen keräämiseen. Yksinkertaisimmillaan yhdyskäytävä voi tarjota vain pyyntöjen reitityksen, jolloin yhdyskäytävä vastaa toiminnallisuudeltaan käänteistä välityspalvelinta (engl. *reverse proxy*). Lähes kaikki API-yhdyskäytäväohjelmistoista ovat maksullisia, tai toiminnoiltaan merkittävästi rajoitettuja ilman maksullista lisenssiä.

Mikropalveluiden väliseen kommunikointiin on myös vaihtoehtoisia tapoja, joissa kommunikointi ei tapahdu API-yhdyskäytävän avulla. Mikropalvelut voivat kommunikoida viestinvälittäjäsovelluksen avulla, jolloin kommunikointi ei tapahdu HTTP-rajapintojen avulla. Viestinvälittäjä osaa reitittää saapuvat viestit oikeille vastaanottajille, jolloin lähettäjän ei tarvitse tietää viestin vastaanottajia [13]. Kommunikointi viestinvälittäjän avulla on usein asynkronista, jolloin viestin lähettäjä ei jää odottamaan vastausta viestiinsä, kuten HTTP-avulla kommunikoidessa tapahtuu.

Yksi vaihtoehtoinen teknologia mikropalveluiden väliseen kommunikointiin HTTP-rajapinnoin on *service mesh*, jossa jokainen mikropalvelu kommunikoi suoraan toisilleen ilman keskitettyä viestinvälittäjäkomponenttia. Service mesh pyrkii verkottamaan palvelut keskenään ja tarjoamaan työkalut sekä toimintatavat nimenomaan palveluiden väliseen kommunikointiin [16], kun API-hallinnan tavoitteena on myös huomioida rajapintoja ja palveluja hyödyntävät käyttäjät.

Palveluiden verkottamisen mahdollistavat työkalut tarjoavat osin samoja toiminnallisuuksia kuin API-yhdyskäytävät, jolloin ne voivat korvata API-hallinnan palveluiden välisessä kommunikoinnissa [16]. On kuitenkin huomattava, että API-hallinnan työkalut voivat olla tästä huolimatta hyödyllisiä, jotta tarpeelliset toiminnallisuudet voidaan toteuttaa myös rajapintoja hyödyntäville asiakassovelluksille.

2.4 Esimerkki API-hallinnasta

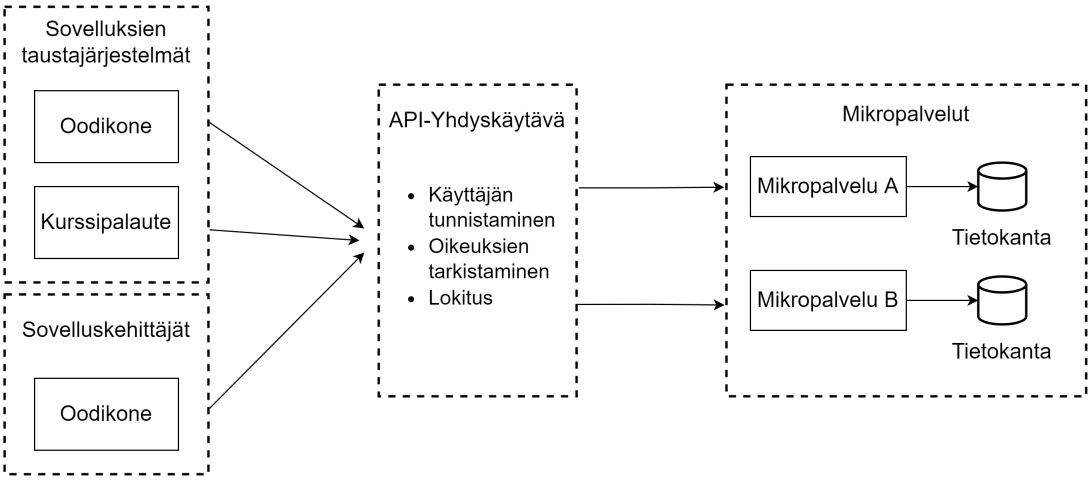
Tässä alaluvussa on esitelty esimerkki, kuinka API-hallinta on toteutettu Helsingin yliopiston Tietojenkäsittelytieteen osaston sovelluskehitysakatemiassa eli Toskassa. Toskan ydintehtävänä on kehittää yliopiston opetushallinnollisia sovelluksia. Merkittävimpiä Toskan kehittämiä sovelluksia ovat Norppa-kurssipalautejärjestelmä sekä Oodikone opintosuoritusdatan tutkimiseen [24].

Toskassa sovelluskehityksessä hyödynnetään mikropalveluarkkitehtuuria, jossa useat taustajärjestelmät ja sovelluskehittäjät hyödyntävät samoja mikropalveluja keskenään. Toskan API-hallintaa kehittäessä huomioitiin erityisesti tarve keskitetylle pääsynhallinnalle, jossa voidaan rajata pääsy hienojakoisesti eri mikropalveluihin ja niiden tarjoamiin rajapintoihin. Tähän tarkoitukseen sopii hyvin API-yhdyskäytävä, joka tarjoaa keskitetyn alustan muun muassa pääsynhallintaan.

Kuvassa 2.3 on esitetty yksinkertaistettu kaavio Toskan mikropalveluarkkitehtuurista. Sovelluksien taustajärjestelmät ja sovelluskehittäjien paikalliset kehitysympäristöt keskustelevat mikropalveluiden kanssa aina API-yhdyskäytävän kautta. Näin voidaan varmistaa, että sovelluksilla on pääsy vain tarvittaviin resursseihin eikä API-yhdyskäytävälle tunnistautumiseen käytettyjä avaimia jaeta usean kehittäjän ja ympäristön välillä.

API-yhdyskäytävä tarkistaa, että rajapintakutsuun on liitetty salainen avain ja avaimelle on myönnetty pääsy pyydettyyn resurssiin. Onnistuneen tunnistautumisen jälkeen yhdyskäytävä reitittää saapuvat pyynnöt ennalta määrättyjen sääntöjen mukaan mikropalveluille ja välittää asiakassovellukselle mikropalvelun palauttaman vastauksen. Kommunikointi mikropalveluiden ja API-yhdyskäytävän välillä on varmennettu asiakassertifikaatein, jolloin mikropalveluun ei voi ottaa yhteyttä ilman API-yhdyskäytävää ja riittävää tunnistautumista.

Huomionarvoista tässä esimerkissä on se, että API-hallinnalla Toskassa on pyritty ratkomaan erityisesti kommunikointi sovelluksien taustajärjestelmien ja mikropalveluiden välillä. Sovelluksien käyttöliittymät kommunikoivat taustajärjestelmiensä kanssa suoraan ilman, että rajapintakutsut kulkevat API-yhdyskäytävän kautta. Tässä esimerkissä vain sovelluksien taustajärjestelmät kommunikoivat mikropalveluiden kanssa.



Kuva 2.3: Yksinkertaistettu kuva Toskan arkkitehtuurissa, jossa hyödynnetään API-yhdyskäytävää rajapintojen hallintaan. Asiakassovellus tai sovelluskehittäjä tunnistautuu API-yhdyskäytävälle avaimella (*token*) ja mikropalvelut hyväksyvät vain pyynnöt, jotka tulevat API-yhdyskäytävästä.

3 Tutkimusmenetelmät

Tässä luvussa esitellään tutkielmassa käytetyt tutkimusmenetelmät sekä tutkimuksen eri vaiheet. Alaluvussa 3.1 on esitelty tutkimuskysymykset, joihin tämä tutkielma pyrkii vastaamaan. Alaluvussa 3.2 on luotu yleiskatsaus tutkielman rakenteeseen. Kirjallisuuskatsauksen tekemiseen käytetty prosessi on kuvattu alaluvussa 3.3 ja alaluvussa 3.4 on kerrottu, kuinka tutkielman laadullinen aineisto on kerätty ja analysoitu.

3.1 Tutkimuskysymykset

Tutkielmassa pyritään selvittämään mitä API-hallinnalla tavoitellaan ja kuinka nämä tavoitteet on toteutettu käytännön sovelluskehityksessä. Tätä pyritään tutkielmassa selvittämään seuraavilla tutkimuskysymyksillä:

TK1: Mitä asioita API-hallinnalla pyritään saavuttamaan mikropalveluarkkitehtuurissa?

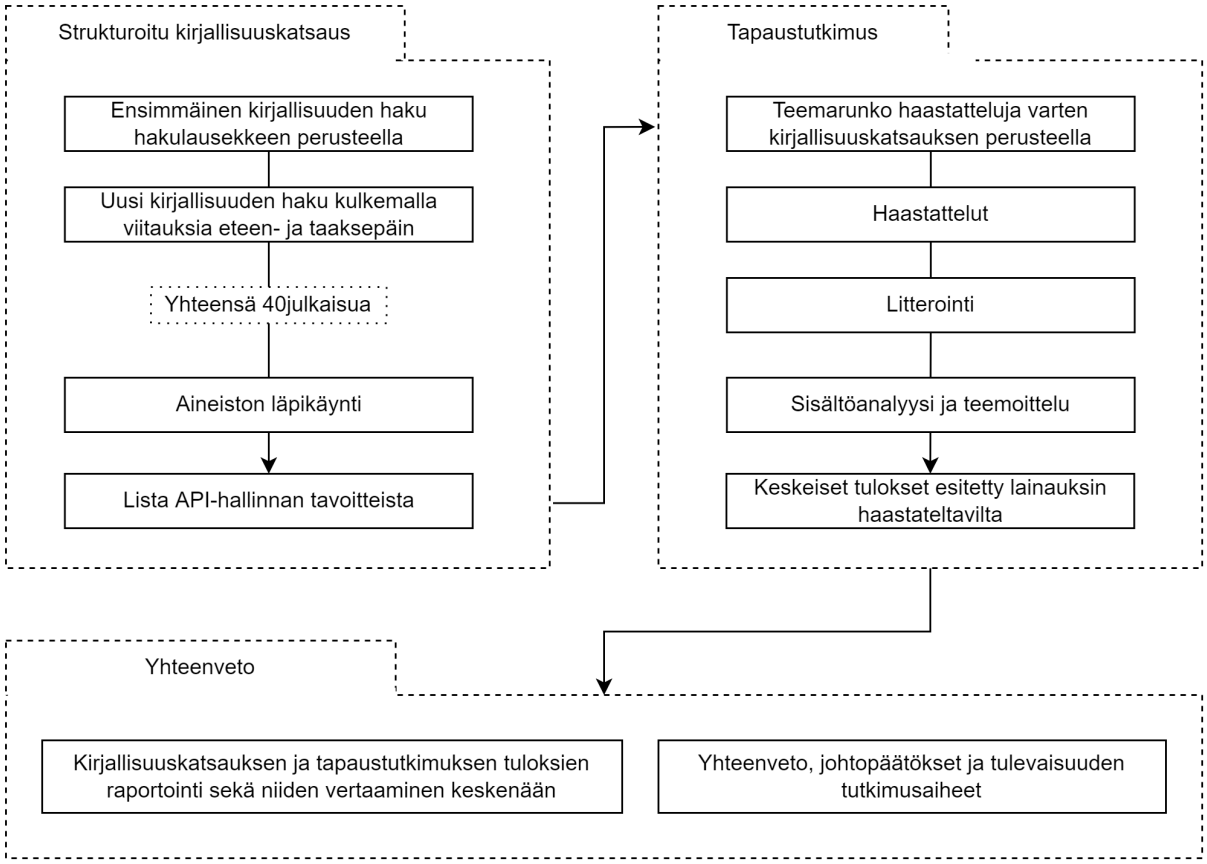
TK2: Kuinka API-hallinta on toteutettu käytännön sovelluskehityksessä?

TK3: Kuinka API-hallinta koetaan käytännön sovelluskehityksessä?

Ensimmäisellä tutkimuskysymyksellä pyritään selvittämään tutkimuskirjallisuuden avulla mitä API-hallinnalla tavoitellaan. Toinen kysymys kartoittaa tapaustutkimuksen avulla, kuinka tutkimuskirjallisuudesta löydetyt tavoitteet ovat toteutettu tapausorganisaatioiden käytännön sovelluskehityksessä. Kolmannen tutkimuskysymyksen avulla pyritään luomaan käsitys, kuinka tapausorganisaatiossa koetaan käytössä olevat API-hallinnan käytänteet ja työkalut.

3.2 Tutkimuksen rakenne

Tutkimus on jaettu kolmeen osaan: kirjallisuuskatsaukseen, tapaustutkimukseen ja yhteenvetoon kahden aiemman tutkimusmenetelmän tuloksista. Kuvassa 3.1 on havainnollistettu tutkimusmenetelmien välistä suhdetta ja tutkimuksen rakennetta.



Kuva 3.1: Tutkielman kolme vaihetta

Kirjallisuuskatsauksen tavoitteena on luoda yleiskatsaus API-hallintaan, jonka pohjalta voidaan tehdä teemahaastattelut. Teemahaastatteluin kartoitetaan tapausorganisaatioiden API-hallinnan tavoitteita, haasteita sekä työkaluja. Tutkielman viimeisessä osassa, yhteenvedossa verrataan kuinka haastateltavien raportoimat tavoitteet ja käytänteet vertautuvat kirjallisuudessa esitettyihin tavoitteisiin.

3.3 Osa I: Strukturoitu kirjallisuuskatsaus

Tässä osiossa on esitelty strukturoidun kirjallisuuskatsauksen teoriaa sekä kuinka kirjallisuuskatsaus on tehty tässä tutkielmassa.

3.3.1 Kirjallisuuskatsaus tutkimusmenetelmänä

Kirjallisuuskatsauksella pyritään selvittämään kattavasti mitä jostakin tutkimusalueesta tiedetään aiemman tutkimuskirjallisuuden perusteella. Tämän avulla voidaan luoda tiivistelmä tutkitusta aiheesta usean erillisen tutkimustuloksen perusteella. Kirjallisuuskatsauksen tuloksena voi olla myös huomio, ettei valitusta aiheesta ole vielä riittävästi tutkimuskirjallisuutta kattavaien johtopäätöksien tueksi. Kirjallisuuskatsausta käytetään usein tutkielmassa pohjatyönä muita tutkimusmenetelmiä varten, kuten tässäkin tutkielmassa [15].

Perinteisesti systemaattinen kirjallisuuskatsaus voidaan jakaa viiteen eri osaan. Kirjallisuuskatsauksen ensimmäisessä osassa on oleellista määrittää tutkimuskysymykset, joihin pyritään vastaamaan sekä kuinka kirjallisuuskatsaus tullaan toteuttamaan. Tutkimuksen toistettavuuden kannalta on tärkeää, että käytetty menetelmä on dokumentoitu eikä vain raportoida tuloksia [15].

Toinen vaihe kirjallisuuskatsauksen toteuttamisessa on tietokantojen sekä hakulausekkeiden määrittäminen. Hakulausekkeet tulisi määrittää siten, että niihin täsmänneet julkaisut kuvaavat mahdollisimman kattavasti tutkimusasetelmaa, johon kirjallisuuskatsauksella pyritään vastaamaan. Hakulausekkeet ja hyödynnetyt tietokannat tulee dokumentoida, jotta haku olisi toistettavissa. On kuitenkin huomattava, että digitaalisia tietokantoja hyödyntäessä haku on usein lähes mahdotonta toistaa sellaisenaan [15].

Kirjallisuuskatsauksen kolmannessa vaiheessa hakulausekkeisiin täsmänneet julkaisut käydään läpi ja niistä karsitaan sisällyttämis- ja poissulkemiskriteerien perusteella analysoitavaksi joukko julkaisuja. Näillä kriteereillä pyritään karsimaan epäolennaiset julkaisut pois ja vain tutkielman kannalta oleelliset julkaisut analysoidaan tarkemmin [15].

Neljännessä vaiheessa analysoitavaksi valikoituneisiin julkaisuihin tutustutaan tarkemmin. Julkaisuista pyritään keräämään tutkimuskysymyksien kannalta kaikki olennainen informaatio ja jäsentelemään se mielekkääseen muotoon johtopäätöksien tekoa varten. Tästä aineistosta luodaan kirjallisuuskatsauksen viimeisessä vaiheessa yhteenveto ja johtopäätökset [15].

Tässä tutkimuksessa kirjallisuuskatsaus tehdään strukturoituna kirjallisuuskatsauksena (engl. *semi-systematic literature review*) [23]. Strukturoitu kirjallisuuskatsaus täyttää osan systemaattisen kirjallisuuskatsauksen kriteereistä [15], mutta on laajuudeltaan suppeampi eikä kirjallisuuskatsauksen tekemiseen osallistu kuin yksi tutkija.

Systemaattisen kirjallisuuskatsauksen kriteereistä tässä tutkimuksessa on toteutettu hakulausekkeen ja sen muodostamisen dokumentointi, sopivien viitetietokantojen hyödyntäminen, julkaisuiden karsiminen valinta- ja poissulkukriteerein sekä analysoitavaksi valittujen julkaisuiden dokumentointi. Julkaisuiden laatua ei arvioitu kattavasti, joitakin lähteitä valittiin hakulausekkeen ulkopuolelta eikä aineiston läpikäytiin ja johtopäätöksien muodostamiseen osallistunut useita tutkijoita. Osa hakulausekkeen ulkopuolelta valikoiduista julkaisuista olivat myös vanhempia kuin hakutuloksille sovellettavissa sisällyttämiskriteereissä määritellään. Tämän osalta tutkimus ei täytä järjestelmällisen kirjallisuuskatsauksen määritelmää, jonka vuoksi tämän tutkimuksen kirjallisuuskatsausta kuvaa paremmin termi strukturoitu kirjallisuuskatsaus.

3.3.2 Hakulausekkeen muodostaminen ja lähteet

Tässä tutkielmassa strukturoidulla kirjallisuuskatsauksella pyritään tarkastelemaan, mitä API-hallinnalla tavoitellaan tutkimuskirjallisuuden perusteella. Kirjallisuuskatsaus pyrkii vastaamaan tutkielman ensimmäiseen tutkimuskysymykseen, *Mitä asioita API-hallinnalla pyritään saavuttamaan?*.

Kirjallisuuskatsaus tehtiin hyödyntämällä seuraavia kolmea digitaalista kirjastoa: ACM Digital Libray, IEEE Explore ja Scopus. Hakulausekkeet muodostettiin tekemällä käsin kokeellisia hakuja käyttäen eri avainsanoja, jotta termien mahdolliset synonyymit sekä eri kirjoitusasut saatiin kartoitettua. Hakulausekkeessa hyödynnetään loogisia operaattoreita *OR* ja *AND* yhdistelemään avainsanoja sekä termien vaihtoehtoisia kirjoitusasuja. Haku kaikkiin kolmeen tietokantaa suoritettiin seuraavalla hakulausekkeella:

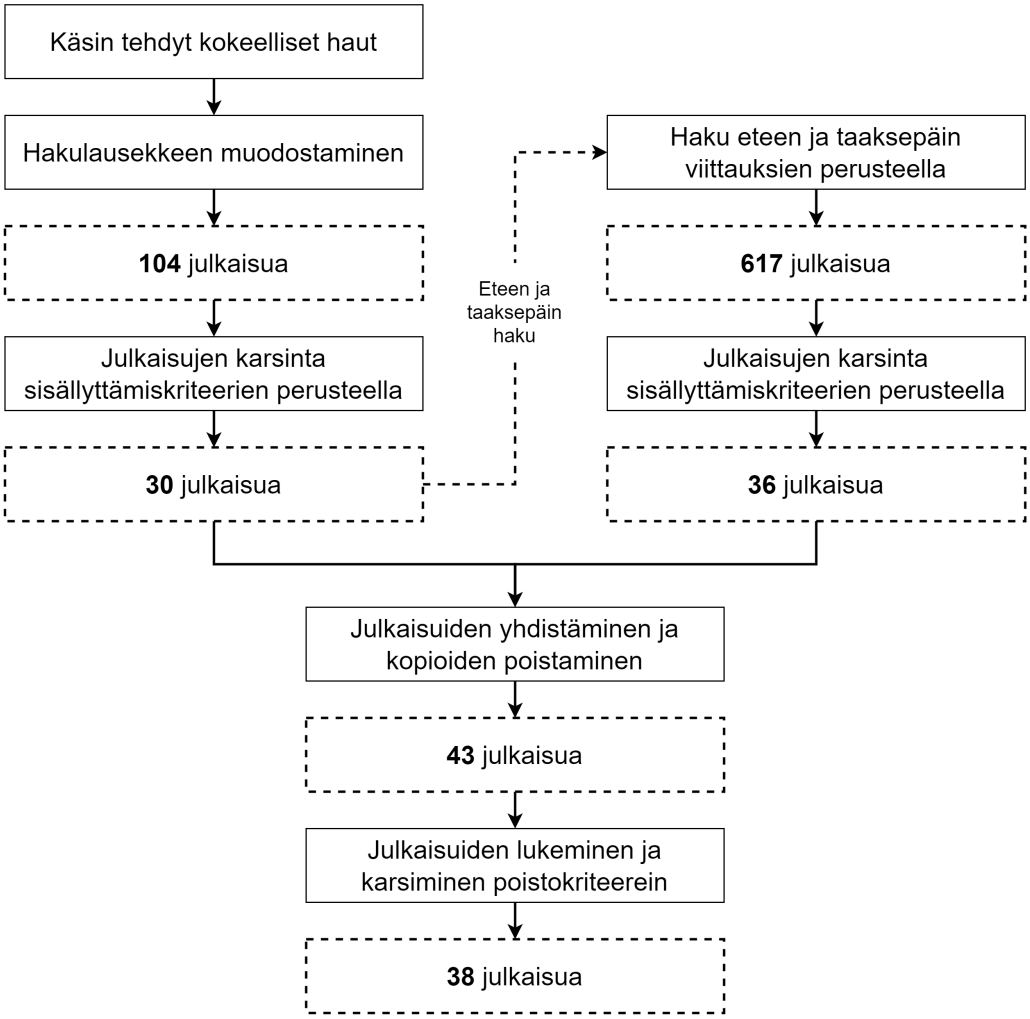
```
"api"
AND ("management" OR "gateway")
AND "microservice"
```

Hakulausekkeen tavoitteena on löytää julkaisut, joissa käsitellään API-hallintaa tai API-yhdyskäytävää mikropalveluiden kanssa. Hakulauseke kohdistettiin julkaisujen otsikkoon, avainsanoihin ja tiivistelmään. Kuvan 3.2 vasemmanpuoleisessa osassa on havainnollistettu kirjallisuuskatsauksen vaiheet hakulausekkeen muodostamisesta ja hakutuloksien analysoinnista.

API-yhdyskäytävä on valittu hakulausekkeeseen mukaan, sillä kokeellisten hakujen perusteella tutkimuskirjallisuudessa sitä saatetaan käyttää API-hallinta termin sijaan. Termi

mikropalvelu hakulausekkeessa rajaa tuloksia tutkielman kannalta olennaisiin julkaisuihin ja samalla vähentää hakutuloksia tuhansista julkaisuista sellaiseksi, joka on maisterintutkielmassa mahdollista käsitellä. On kuitenkin huomattava, että tämä rajaus voi sivuuttaa joitakin olennaisia julkaisuja, johon pyritään vastaamaan osassa 3.3.4 esitellyllä lumipallomenetelmällä.

Hakulauseke ottaa myös huomioon API-hallinnan ja -yhdyskäytävän englanninkielisen kirjoitusasun, jossa voidaan käyttää yhdysviivaa, sillä haku etsii erikseen termejä *api* ja *management* tai *gateway* ottamatta kantaa ovatko nämä kirjotettu yhteen väliviivalla vai erikseen. Osa tutkimuksessa käytetyistä tietokannoista (*IEEE* ja *Scopus*) jättävät myös oletusarvoisesti huomioimatta haussa esiintyvät välimerkit.



Kuva 3.2: Kirjallisuuskatsauksen vaiheet

Haku suoritettiin 24. päivä helmikuuta 2022. Taulukossa 3.1 on esitetty hakutuloksien määrä digitaalisen kirjaston mukaan jaoteltuna. Hakulausekkeeseen täsmäsi yhteensä 104

julkaisua, joista karsittiin valintakriteerien perusteella analysoitavaksi kolmekymmentä julkaisua. Suuri osa hakulausekkeeseen täsmänneistä julkaisuista ei kuitenkaan käsitellyt rajapintojen hallintaa sovelluskehityksessä, jonka vuoksi analysoitavaan aineistoon lisättiin manuaalisilla hauilla lähteitä.

Taulukko 3.1: Hakutuloksien määrä hakukohteittain sekä yhteenlaskettu julkaisujen määrä

Kirjasto	**Julkaisujen määrä**
Scopus	57
IEEE	39
ACM	8
Yhteensä	**104**

3.3.3 Valinta- ja poissulkukriteerit

Julkaisut valittiin analysoitavaksi valintakriteerien perusteella, joka tehtiin julkaisun otsikon ja tiivistelmän perusteella. Julkaisujen poissulkemiseen käytettiin yhtä poissulkemiskriteeriä, jota hyödynnettiin kirjallisuuskatsauksen myöhemmässä analysointivaiheessa ja poissulkeminen tehtiin julkaisun leipätekstin perusteella. Aineistolle sovellettiin seuraavia valinta- ja poissulkemiskriteerejä:

Valintakriteerit:

1. Julkaisu on korkeintaan viisi vuotta vanha
2. Julkaisu on ensisijainen julkaisu, toissijaiset tutkimukset kuten kirjallisuuskatsaukset karsittiin pois
3. Julkaisu on kirjoitettu englanniksi

Poissulkemiskriteerit:

1. Julkaisussa ei esitelty mitä API-hallinnalla on tavoiteltu tai kuinka sitä on hyödynnetty

3.3.4 Lumipallomenetelmä

Lumipallomenetelmässä (engl. *snowballing*) sisällyttämis- ja poissulkukriteerein muodostettua alkujoukkoa laajennetaan julkaisujen lähteiden ja niihin viittaavien julkaisuiden avulla. Lumipallomenetelmän ensimmäisessä osassa alkujoukon julkaisujen lähteistä etsitään relevatteja julkaisuja hyödyntämällä alaluvussa 3.3.3 esiteltyjä sisällyttämis- ja poissulkukriteerejä [27]. Tätä vaihetta kutsutaan englanninkielisellä termillä *backwards snowballing*, joka kuvaa alkujoukon lähteissä taaksepäin kulkemiseen. Sisällyttämiskriteereihin täsmänneet julkaisut lisätään analysoitavien julkaisujen joukkoon ja mahdolliset kaksoiskappaleet poistetaan. Kuvan 3.2 oikeanpuoleisessa osassa on havainnollistettu lumipallomenetelmän osuutta kirjallisuuskatsauksessa.

Lumipallomenetelmän toisessa vaiheessa tarkastellaan jälleen julkaisujen alkujoukkoa, jota laajennetaan tarkastelemalla julkaisuja, jotka viittaavat alkujoukon julkaisuihin. Tätä vaihetta kutsutaan englanninkielisellä termillä *forwards snowballing*, joka kuvaa viittauksissa eteenpäin kulkemista. Eteenpäin kulkemalla löydetyille uusille julkaisuille sovelletaan jälleen sisällyttämis- ja poissulkukriteerejä, jonka jälkeen jäljelle jääneet julkaisut sisällytetään analysoitavien julkaisujen joukkoon [27].

Tässä tutkielmassa viitetietokantana käytettiin samoja tietokantoja, joissa alkuperäinen haku suoritettiin. Lumipallomenetelmällä yhden iteraation eteen- ja taaksepäin kulkemalla löydettiin yhteensä 617 julkaisua, joista 36 julkaisua täsmäsi sisällyttämiskriteereihin.

3.3.5 Aineiston analysointi

Kirjallisuuskatsauksen analysointivaiheessa sovellettiin poissulkemiskriteeriä julkaisun leipätekstin perusteella. Tässä vaiheessa yhteensä viisi julkaisua poistettiin poissulkemiskriteerien perusteella. Poissulkemiskriteerien jälkeen analysoitavaksi valikoitui yhteensä 38 julkaisua. Valitut julkaisut luettiin läpi ja jokaisesta julkaisusta poimittiin seuraavat asiat:

- Julkaisuvuosi
- Joukko relevantteja avainsanoja, jotka kuvaavat API-hallinnan käyttötarkoitusta

Kaikki analysoidut julkaisut ovat julkaistu vuosien 2015 ja 2022 välillä. Taulukossa 3.2 on esitetty kirjallisuuskatsaukseen sisällytetyt julkaisut niiden julkaisuvuoden mukaan ryhmiteltynä.

Taulukko 3.2: Analysoitavaksi valitut julkaisut julkaisuvuoden mukaan

Vuosi	Julkaisut
2015	[A4]
2017	[A7, A13, A16, A18, A27, A8]
2018	[A3, A8, A10, A12, A15, A24, A25, A26, A29]
2019	[A1, A2, A9, A14, A19, A23, A28, B1, B2, B3]
2020	[A5, A20, A30, B4, B5, B8]
2021	[A6, A11, A21, A22]
2022	[A17, B7, B6]

Julkaisuista johdettu aineisto koostettiin erilliseen laskentataulukkoon, jota myöhemmin hyödynnettiin tärkeimpien API-hallinnan toiminnallisuuksien ja tavoitteiden selvittämisessä. Laskentataulukkoon kerättiin julkaisun nimi, julkaisuvuosi sekä julkaisusta tunnistetut API-hallinnan tavoitteet lyhyinä avainsanoina tai lainauksina.

3.4 Osa II: Tapaustutkimus

Tässä alaluvussa on esitelty tapaustutkimus, jonka avulla pyritään kartoittamaan, kuinka tapausorganisaatioissa on toteutettu API-hallinta ja tutkimuskirjallisuudesta tunnistettuja tavoitteita.

3.4.1 Tutkielman laadullinen lähestymistapa

Kirjallisuuskatsauksen lisäksi tutkielmassa on hyödynnetty laadullisia menetelmiä. Tapaustutkimuksen avulla pyritään selvittämään havainnollistavin esimerkein, kuinka kirjallisuuskatsauksessa löydettyjä tavoitteita ja toimintoja on toteutettu käytännön sovelluskehityksessä. Tapaustutkimus sopii tutkimusasetelmaan, jossa halutaan tutkia, kuinka jokin ilmiö tapahtuu sen luonnollisessa ympäristössä ilman ulkopuolisia häiriötä tai tutkijan vuorovaikutusta tutkittavaan ilmiöön [22]. Tässä tutkimuksessa suoritettua tapaustutkimusta varten kerätään laadullista aineistoa haastattelemalla sovelluskehittäjiä neljästä tapausorganisaatiosta. Laadullinen tutkimus sopii tähän tutkimukseen kartoittamaan API-hallinnan käytänteitä, sillä se ei pyri tilastollisesti määrittämään tutkittavaa ilmiötä, vaan pyrkii kuvaamaan ja ymmärtämään tutkittavaa ilmiötä [4].

Tutkielman laadullinen lähestymistapa on aineistolähtöinen, joka on melko yleinen me-

netelmä laadullisessa tutkimuksessa. Aineistolähtöisessä sisällönanalyysissä tutkija pyrkii luomaan käsityksen tutkimuksen kohteesta ja vastaamaan tutkimuskysymyksiinsä yhdistelemällä aineistostaan löytyviä käsitteitä ja havaintoja. Tutkija pyrkii muodostamaan käsityksensä aineistoa tulkitsemalla ja päättelemällä, sen sijaan, että analysoitavaa aineistoa verrattaisiin ennalta määriteltyyn teoriaan [25].

Aineistolähtöinen analyysi on sopiva vaihtoehto tähän tutkielmaan, sillä laadullisella tutkimuksella tässä tutkielmassa pyritään havainnollistamaan, kuinka tapausorganisaatioissa on toteutettu API-hallinnan tavoitteita ottamatta kantaa tutkimuskirjallisuuteen [4].

3.4.2 Tapauskuvaus

Tapaustutkimusta varten tehtiin yhteensä neljä haastattelua neljästä ohjelmistokehitystä tekevästä organisaatiosta. Kuhunkin haastatteluun osallistui yksi asiantutija tapausorganisaatiosta. Taulukossa 3.3 on listattu haastateltavien titteli, työskentelyvuodet organisaatiossa sekä koodi, jolla luvussa 4 viitataan haastateltavaan.

Haastateltavat valikoituivat tutkijan kontaktien perusteella ja suosien organisaatioita, joissa oletettavasti tehdään tutkielman kannalta oleellista sovelluskehitystä. Tämän kaltaista otantaa voidaan kutsua mukavuusotokseksi (engl. *convenience sample*), jossa otos määräytyy esimerkiksi tutkijan lähipiirin perusteella sen sijaan, että pyritään mahdollisimman kattavaan otokseen. Kustakin tapausorganisaatiosta pyrittiin löytämään haastateltavaksi sellainen henkilö, joka työskentelee läheisesti rajapintojen kanssa web-sovelluskehityksessä tai esimerkiksi organisaation it-infrastruktuurin parissa.

Taulukko 3.3: Tapaustutkimuksessa haastateltiin yhteensä neljää sovelluskehittäjää eri organisaatioista.

Haastattelu	Rooli	Työskentelyvuodet yrityksessä
H1	Tuoteomistaja	2
H2	Senior Software Engineer	2
H3	Sovellusarkkitehti	7
H4	Azure Platform Lead	4

Haastattelut pidettiin syksyllä 2022 etäyhteyden avulla. Haastateltavat työskentelivät organisaatioissa, joista H1 tarjoaa SaaS-työnhallintasovellusta, H2 terveysalan ohjelmistoa, H3 SaaS-palveluna toimitettavaa toiminnanohjausjärjestelmää ja H4 it-järjestelmiä organisaation omiin tarpeisiin (ei lupaa mainita toimialaa).

Haastattelumenetelmäksi valikoitui teemahaastattelu, joka sopii tutkimukseen hyvin sen

mahdollistaman vapaamuotoisen keskustelun vuoksi. Teemahaastelussa vain haastattelun aihepiirit, eli teemat ovat etukäteen määritelty eikä tarkkoja kysymyksiä tai niiden määrää ole etukäteen määritelty. Teemahaastattelun valintaa tukee myös tutkimusasetelman monisyinen luonne, jonka vuoksi haastattelukysymyksiä ei ole mielekästä rajata liian tarkasti. Haastateltavien taustat ja tilanteet voivat olla toisistaan hyvin erilaisia, jolloin etukäteen määritellyt kysymykset eivät välttämättä vastaa lainkaan haastateltavan ympäristöä [12].

Haastatteluissa käsitellyt teemat valikoituivat tutkimuskirjallisuudesta niiden yleisyyden mukaan, mutta myös sen mukaan kuinka relevantteja yleisimmät kirjallisuudesta tunnistetut teemat ovat. Esimerkiksi kuormantasaus, palveluiden löytäminen ja koostaminen pudotettiin haastatteluiden aiheista pois, sillä nämä toiminnot ovat oletettavasti sellaisia, joita ei todennäköisesti tarvitse kaikissa organisaatioissa miettiä.

Haastatteluin pyrittiin pureutumaan sellaisiin API-hallinnan toimintoihin, joita kaikissa sovelluskehitystä tekevissä organisaatioissa tulee esille jossakin muodossa. Kirjallisuuskatsauksen ulkopuolelta valikoitui dokumentaatio yhdeksi teemaksi, sillä dokumentaation tarjoaminen on mainittu useissa rajapintojen hallintaa tarjoavissa alustoissa, kuten Microsoftin Azuressa [19].

Tapaustutkimuksella pyritään vastaamaan tutkimuskysymyksiin *Kuinka API-hallinta on toteutettu käytännön sovelluskehityksessä?* ja *Kuinka API-hallinta koetaan käytännön sovelluskehityksessä?*. Tutkielman toiseen tutkimuskysymykseen pyritään vastaamaan seuraavan listan neljällä ensimmäisellä teemalla ja kolmanteen tutkimuskysymykseen, hieman aikaisempia laajemmalla teemalla:

- Rajapintojen käyttäminen, tunnistautuminen ja pääsynhallinta
- Rajapintojen muutoksien koordinointi, versiointi
- Rajapintoihin liittyvien lokitietojen keräys ja tilastointi
- Rajapintojen dokumentointi
- API-hallinnan prosessit, kuinka kehittäjät ja organisaatio kokee työskentelytavat ja työkalut

Liitteessä B on esitetty haastattelurunko, jossa on liitetty kuhunkin teemaan joukko tukikysymyksiä. Haastattelutilanteissa tukikysymyksiä ei välttämättä hyödynnetty sellaisenaan sanasta sanaan, niitä ei välttämättä kysytty lainkaan tai esitettiin kysymyksiä, joita

ei tässä taulukossa ole listattu. Teemoja ei jokaisessa haastattelussa käsitelty taulukossa listatussa järjestyksessä. Haastattelut nauhoitettiin haastateltavan luvalla.

3.4.3 Aineiston analysointi

Ennen aineiston varsinaista analysointia kaikki haastattelut litterointiin. Litteroinnissa kirjoitettiin haastattelut auki kokonaisuudessaan eikä tässä vaiheessa jätetty epäolennaiselta vaikuttavia osia tai lauseita pois. Litteroinnin jälkeen tarkistettiin vielä tekstimuotoisen aineiston vastaavan haastattelunauhoitusta ja litteroidusta aineistosta valitut lainaukset tarkistettiin haastateltavilla. Litteroitua aineistoa kertyi analysoitavaksi yhteensä kuusikymmentä sivua. Ennen varsinaista sisältöanalyysiä haastattelut koodattiin muotoon *H1*, *H2* ja niin edelleen, jotta aineistojen vertailu olisi helpompaa ja haastattelu ei assosioituisi tutkijan mielessä tiettyyn henkilöön.

Sisältöanalyysin tavoitteena on tiivistää analysoitava aineisto kadottamatta sen sisältämää informaatiota. Tutkielmassa hyödynnetään aineistolähtöistä analyysiä, jossa pyritään luomaan pelkästään empiirisen aineiston perusteella käsitys tutkimuksen kohteesta. Menetelmänä on käytetty teemoittelua, sillä se on sopiva monista vaihtoehdoista teemahaastatteluin kerätyn aineiston analysointiin [4]. Aineistolähtöinen analyysi teemoittelun keinoin sopii tähän tutkimukseen hyvin, sillä haastatteluin on pyritty tutkimaan sitä, kuinka tapausorganisaatioiden sovelluskehityksessä koetaan API-hallinta ja kuinka se vertautuu kirjallisuuskatsauksessa tehtyihin löytöihin.

Teemoittelussa aineistosta pyritään löytämään samankaltaisia piirteitä, eli teemoja usean eri haastateltavan välillä. Usein osa näistä aineistosta löydetyistä teemoista vastaa teemahaastattelun pohjana olleita lähtöteemoja, mutta sen lisäksi teemoittelussa voi löytyä uusia mielenkiintoisia teemoja. On kuitenkin huomattava, että aineiston analysointi teemoittelun keinoin riippuu tutkijan omasta tulkinnasta, sillä eri haastateltavat harvoin ilmaisevat samoja asioita tarkalleen samalla tavalla [12]. Aineiston teemoittelu tehtiin hyödyntämällä *Atlas.ti* ohjelmistoa, jolla tekstimuotoisen aineiston koodaus eri teemoihin ja vertailu on helppoa.

4 Tulokset

Tässä luvussa on esitelty kahdessa erillisessä osassa kirjallisuuskatsauksen sekä teemahaastatteluiden tulokset. Ensin alaluvussa 4.1 esitellään kirjallisuuskatsauksessa tunnistetut API-hallinnan tavoitteet ja niistä oleellisimpiin pureudutaan tarkemmin. Jälkimmäisessä alaluvussa 4.2 esitellään tapaustutkimuksessa löydettyjä teemoja ja toimintatapoja sekä kokemuksia, kuinka käytössä olevat API-hallinnan toimintatavat koetaan tapausorganisaatioissa.

4.1 Osa I: Strukturoitu kirjallisuuskatsaus

Yhteensä 38 julkaisua valikoitui aiemmin esiteltyjen kriteerien perusteella analysoitavaksi. Artikkeleista johdettiin taulukko API-hallintaa kuvaavista avainsanoista. Tässä osassa vastataan tämän aineiston perusteella ensimmäiseen tutkimuskysymykseen: *Mitä asioita API-hallinta pyrkii saavuttamaan?*.

Artikkeleista johdettu aineisto analysointiin ja selvitettiin yleisimmät avainsanat, jotka kuvaavat API-hallinnan tavoitteita. Taulukossa 4.1 on listattu kaikki yhteensä 17 löydettyä avainsanaa ja avainsanan esiintymiskerrat analysoitaviksi sisällytetyissä julkaisuissa. Avainsana on laskettu mukaan vain yhden kerran per julkaisu.

Analysoinnissa tuotetun aineiston perusteella alla on listattu viisi yleisintä API-hallinnan tavoitetta. Osa avainsanoista on yhdistetty, kuten esimerkiksi pääsynhallinta (engl. *authorization*) ja käyttäjän todentaminen (engl. *authentication*). Yhdistetyt avainsanat ovat käyttötapauksiltaan hyvin lähellä toisiaan eikä täten ole mielekästä eritellä niitä omiksi avainsanoikseen. Alla on listattu viisi yleisintä API-hallinnan aktiviteettia sekä niiden esiintymiskerrat julkaisuissa. Taulukossa 4.2 on esitetty julkaisut, joissa nämä aktiviteetit ovat mainittu.

- Tunnistautuminen (22kpl) ja pääsynhallinta (17kpl)
- Kuorman tasaus (6kpl)
- Lokitietojen keräys (5kpl) ja tilastointi (5kpl)
- Palveluiden löytäminen (3kpl)

- Versiointi (3kpl)

Osa tutkimuskirjallisuudesta ei määritellyt, kuinka yllä listatut toiminnallisuudet toteutetaan. Suurin osa lähteistä mainitsi API-yhdyskäytävän yksittäiseksi työkaluksi, jolla nämä toiminnot on mahdollista toteuttaa. Seuraavissa alaluvuissa tutustutaan yksityiskohtaisemmin näihin viiteen yleisimpään API-hallinnan aktiviteettiin.

4.1.1 Tunnistautuminen ja pääsynhallinta

Yleisin tavoite tutkimuskirjallisuuden perusteella oli toteuttaa tunnistautuminen ja pääsynhallinta jossakin keskitetyssä API-hallinnan komponentissa. Tunnistautumisen (engl. *authentication*) tavoitteena on tunnistaa, että käyttäjä on sama kuin kuka käyttäjä väittää olevansa. Usein tämä saavutetaan käyttämällä käyttäjänimeä sekä salasanaa [A13]. Pääsynhallinnan (engl. *authorization*) tavoitteena on varmistaa, että jo tunnistetulla käyttä-

Taulukko 4.1: Kaikki kirjallisuuskatsauksessa tunnistetut aktiviteetit joita pyritään toteuttamaan jollakin keskitetyn API-hallinnan komponentilla sekä niiden esiintymislukumäärät.

Aktivitetti	**Lukumäärä**
tunnistautuminen (engl. *authentication*)	22
pääsynhallinta (engl. *authorization*)	17
kuormantasaus (engl. *load balancing*)	6
lokitietojen keräys (engl. *logging*)	5
tilastointi (engl. *metrics*)	5
palveluiden löytäminen (engl. *service discovery*)	3
koostamien (engl. *aggregation*)	3
versiointi (engl. *versioning*)	3
skaalaus (engl. *scaling*)	2
reititys (engl. *routing*)	2
katkaisija (engl. *circuit breaker*)	1
kapselointi (engl. *encapsulation*)	1
kolmannen osapuolen käyttäjät (engl. *external users*)	1
palveluiden hallinta (engl. *service management*)	1
hajautus (engl. *distribution*)	1
muunnos (engl. *transformation*)	1
terveystarkistus (engl. *health check*)	1

Taulukko 4.2: Viisi yleisintä avainsanaa ja julkaisut joissa avainsanat ovat mainittu

Avainsana	Julkaisut
Tunnistautuminen ja pääsynhallinta	[A7, A24, A13, A27, A19, A16, A25, A28, A18, A17, A12, A30, A26, A21, A8, A23, A6, A4, A3, A5, A29, A15, A22, A9, B2, B4, B3]
Kuormantasaus	[A26, A29, A30, A8, A17, A24]
Lokitiedot ja tilastointi	[A9, A6, A26, A4, B2]
Palveluiden löytäminen	[A8, A26, A16]
Versiointi	[A1, A4, B5]

jällä on pääsy pyydettyyn resurssiin [A9, A21]. Esimerkiksi kirjautuneella käyttäjällä voi olla pääsy omiin tietoihinsa, mutta järjestelmässä oleviin muiden käyttäjien tietoihin ei ole pääsyä.

Yksi yleinen tapa toteuttaa tunnistautuminen ja pääsynhallinta on käyttää ulkopuolista palvelua, josta API-yhdyskäytävä noutaa tarvittavat tiedot käyttäjän tunnistamiseksi ja pääsyn tarkistamiseen [A3, A19]. On huomattava, että tässä tapauksessa käyttäjien tiedot eivät ole tallennettuna API-yhdyskäytävään. Käyttäjien tiedot voivat olla tallennettuna esimerkiksi IAM-palveluun (engl. Identity and Access Management), jonka vastuulla on hoitaa kirjautuminen ja käyttöoikeustietojen tarjoaminen.

IAM-palvelua hyödyntäessä on myös mahdollista, että API-yhdyskäytävä vain pyytää käyttäjän tiedot ja lisää ne edelleen välitettyyn pyyntöön esimerkiksi JWT-avaimena (engl. *JSON Web Token*) [A13, A28, A5]. Tällöin mikropalvelu, jolle pyyntö on edelleenvälitetty, voi tarkistaa API-yhdyskäytävän lisäämästä avaimesta, onko pyynnöllä riittävät oikeudet pyydettyyn resurssiin.

Ulkopuolisen IAM-palvelun hyödyntämisen sijaan tunnistautumiseen ja pääsynhallintaan tarvittavat tiedot on mahdollista tallentaa itse API-yhdyskäytävään. Tällöin usein käyttäjänimen ja salasanan sijaan käyttäjän tunnistetaan yksilöivän, salaisen avaimen (engl. *token*) perusteella. Tämä avain liitetään pyyntöön esimerkiksi HTTP-otsakkeissa, jolloin API-yhdyskäytävän on mahdollista tarkistaa, onko avain validi ja onko kyseiselle avaimelle annettu pääsy pyydettyyn resurssiin [A4, B2, B4].

4.1.2 Kuormantasaus

Yksi yleinen API-hallinnan toiminto on kuormantasaus, erityisesti mikropalveluarkkitehtuuria käyttäessä tämän toiminnallisuuden merkitys korostui. Kuormantasauksella tarkoitetaan saapuvien pyyntöjen jakamista halutun palvelun eri instansseille, jottei pyydetty palvelu ylikuormitu [A29]. API-hallinnan yhteydessä kuormantasauksella tarkoitetaan usein API-yhdyskäytävän toiminnallisuutta, joka jakaa saapuvat pyynnöt usealle saman mikropalvelun instanssille [A30, A26, A29, A24, A4]. Jos API-yhdyskäytävään kohdistuu hyvin suuri määrä rajapintakutsuja, täytyy myös sen kuormantasauksesta huolehtia jakamalla rajapintakutsut usean eri yhdyskäytäväinstanssin kesken [A8].

Kuormantasaus mikropalveluille on mahdollista toteuttaa passiivisesti, jolloin saapuva pyyntö ohjautuu API-yhdyskäytävästä esimerkiksi satunnaisesti johonkin mikropalvelulle määritetyistä instansseista [A17]. Vaihtoehtoinen tapa kuormantasaukseen on tarkkailla mikropalveluiden tilaa ja tarvittaessa luoda uusia instansseja kuorman kasvaessa tietyn rajan yli. Tämä vaatii kuitenkin jonkin keinon tarkkailla mikropalveluiden kuormaa ja mahdollisuuden luoda mikropalveluista uusia instansseja [A30].

4.1.3 Lokitiedot ja tilastointi

Kolmanneksi yleisin toiminnallisuus API-hallinnassa oli tutkimuskirjallisuuden perusteella rajapintoihin liittyvien lokitietojen keräys ja rajapintojen käytön tilastointi. Nämä kaksi toiminnallisuutta on yhdistetty tässä kirjallisuuskatsauksessa saman otsikon alle, sillä ne liittyvät hyvin vahvasti toisiinsa.

Kaksi lähdettä [A29, B2] mainitsee lokitietojen keräyksen tarkoittavan pyyntöjen ja niihin liittyvien metatietojen tallentamista esimerkiksi API-yhdyskäytävässä. Kerätty metatieto voi sisältää esimerkiksi pyynnön suorittamiseen kuluneen ajan, pyynnön mukana lähetetyt parametrit sekä tietoja tunnistautuneesta käyttäjästä. Mahdollisia käyttötarkoituksia tallennetuille lokitiedoille on useita, mutta yleisin toistuva syy, on mahdollisten poikkeamien havaitseminen. Tällaiset poikkeamat voivat johtua esimerkiksi mikropalvelussa olevasta viasta, jolloin voidaan havaita suuri määrä epäonnistuneita rajapintakutsuja. Lokitiedoista havaittavat poikkeamat voivat olla myös tietoturvaan liittyviä, johon voi viitata esimerkiksi lokitiedoista huomattu epänormaalin suuri määrä rajapintakutsuja [A6].

Tallennetuista lokitiedoista voidaan johtaa tilastoja, joita API-hallinnassa voidaan hyödyntää esimerkiksi tarjottujen rajapintojen käytön seurantaan tai käyttäjien laskuttami-

seen rajapintojen käytön perusteella [A4, A29, A9, B3, B2]. Tutkimuskirjallisuuden perusteella eräs yleinen käyttötarkoitus tilastoinnille on seurata tunnistautuneiden käyttäjien tekemien rajapintakutsujen määrää. Tämä mahdollistaa tietyn käyttäjäkohtaisen kiintiön (engl. *quota*) määrittämisen, sekä rajoittaa tai laskuttaa kiintiön ylittäviä pyyntöjä [A9].

4.1.4 Palveluiden löytäminen (*service discovery*)

Tässä tutkielmassa palveluiden löytämisellä tarkoitetaan eritysesti mikropalveluiden löytämiseen käytettyä tekniikkaa, joka liittyy vahvasti kuorman tasaamiseen. Palveluiden löytäminen voi tarkoittaa myös tarjottujen rajapintojen ja toiminnallisuuksien löytämistä [A4], mutta tämä on rajattu tutkielman ulkopuolelle.

Mikropalveluiden löytämiseen käytetty tekniikka mahdollistaa tarjottujen palveluiden joustavan skaalaamisen ja ylläpitämisen. Usein mikropalveluiden löytämiseen hyödynnetään erillistä palvelua, johon mikropalvelu käynnistyessään ilmoittautuu ja tallentaa tiedon palvelun osoitteen saavutettavissa. API-yhdyskäytävää hyödyntäessä yhdyskäytävä voi pyyntöjä uudelleenohjatessaan kysyä palveluiden löytämiseen tarkoitetusta järjestelmästä, minne saapuva rajapintakutsu voidaan ohjata [A16, A26].

Käytettäessä järjestelmää palveluiden löytämiseen, on mahdollista eriyttää mikropalveluiden ja rajapintakutsujen reitityksen määrittely API-yhdyskäytävän konfiguraatiosta. Tämä luo joustavuutta mikropalveluiden hallintaan ja erityisesti skaalaamiseen, jolloin API-yhdyskäytävä voi vain reitittää rajapintakutsun palveluiden löytämiseen tarkoitetulta järjestelmältä saamaansa osoitteeseen [A16].

4.1.5 Versiointi

Tutkimuskirjallisuuden perusteella rajapintojen versiointi on API-hallinnassa tärkeä ominaisuus. Versioinnilla pyritään erottelemaan rajapintojen eri versiot toisistaan [A4]. Versiointi myös mahdollistaa useiden rajapintaversioiden tarjoamisen rinnakkain, jolloin voidaan säilyttää taaksepäin yhteensopivuus, vaikka rajapintojen käyttäjät eivät päivittäisi asiakassovelluksiaan vastaamaan rajapitoihin tehtyjä muutoksia [A1].

Tarve rajapinnan versiointiin tulee, kun rajapintaa hyödyntää useampi, mahdollisesti ulkopuolinen käyttäjä ja toiminnallisuuteen on tehtävä muutoksia, jotka eivät ole yhteensopivia edellisen version kanssa. Tällaisia muutoksia voi olla esimerkiksi datan validoinnin muutokset, tietokantaskeeman tai infrastruktuurin muutokset [A4].

Rajapintojen versiointiin on olemassa tutkimuskirjallisuuden perusteella kaksi yleisesti käytössä olevaa vaihtoehtoa. Yleisemmin käytetty menetelmä on versioida rajapinta käyttämällä URI-osoitetta (engl. *Uniform Resource Identifier*). Tällöin rajapinnan osoitteen alkuun lisätään ylimääräinen osa, jolla määritetään mitä versiota rajapinnasta halutaan käyttää. Vaihtoehtoisesti versio voi olla määritettynä osoitteen parametreissä (*query parameter*), jolloin on mahdollista käyttää samaa osoitetta ja polkua kaikille rajapinnan eri versioille [A1].

Vaihtoehtoinen menetelmä rajapinnan versiointiin on määrittää käytettävä versio HTTP-pyynnön otsakkeissa [B5]. Tällöin on mahdollista käyttää samaa osoiteavaruutta kaikille rajapinnan versioille, mutta otsakkeiden käyttö tekee rajapinnan käytöstä monimutkaisempaa ja voi vaikeuttaa välimuistien käyttöä [A1].

4.2 Osa II: Tapaustutkimus

Tapaustutkimusta varten tehtiin yhteensä neljä haastattelua sovelluskehitystä tekevissä organisaatioissa. Haastateltavat olivat työskennelleet usean vuoden ajan organisaatiossaan ja he työskentelivät usein senior-tason ohjelmistokehitystehtävissä. Haastattelut ovat koodattu kirjaimin *H1*, *H2* ja niin edelleen, jotka vastaavat taulukkoa 3.3.

Tässä luvussa esitellään tapaustutkimuksessa tunnistettuja API-hallinnan tavoitteita ja toiminnallisuuksia. Lisäksi esitellään tapausorganisaatioiden sisäisiä prosesseja ja kokemuksia liittyen API-hallintaan. Aineistosta tunnistettuja teemoja on yhteensä viisi kappaletta, 1) Tunnistautuminen ja pääsynhallinta, 2) Versiointi, 3) Dokumentaatio, 4) Lokitietojen keräys ja 5) API-hallinnan prosessit.

4.2.1 Tunnistautuminen ja pääsynhallinta

Kun kysyttiin miten organisaation sisäisiin tai ulkoisiin rajapintoihin on toteutettu tunnistautuminen ja pääsynhallinta, haastateltavat kuvailivat vaihtelevia ratkaisuja keskitetystä pääsynhallinnasta aina hajautettuun sovelluslogiikkaan integroidusta pääsynhallinnasta. Vaikka organisaatiolla olisi API-yhdyskäytävä käytössä ei pääsynhallintaa ja tunnistautumista välttämättä hoidettu yhdyskäytävässä, vaan lähes aina vähintäkin osa siitä oli toteutettu sovelluslogiikassa.

Erityisesti GraphQL-rajapintoja hyödyntäessä pääsynhallinta koettiin haastavaksi tehdä API-yhdyskäytävässä tai yhdyskäytävää ei käytetty lainkaan. Hienojakoinen pääsynhallin-

ta eri resursseihin koettiin API-yhdyskäytävässä haasteeksi GraphQL-rajapintojen mahdollistaman joustavuuden vuoksi, jossa asiakassovellus voi määrittää itse mitä resursseja haetaan:

> *Siinä on se, kun voit niin montaa resurssia samaan aikaa accessoida ja käytännössä sun pitäis gatewayllä pystyä määrittää entiteettitasolla ja ainakaan mitä mä oon noita pilviratkaisuja kattonut niin mikään ei oikein tue kunnolla vielä GraphQL-rajapintoja, varsinkaa Azuren ratkaisut.* (H2)

> *Se haaste niissä [API-yhdyskäytävissä] on, että jos me toteutetaan autorisointi kokonaan erillisinä layerinä, niin se joko prosessoi ihan sikana dataa eli käytännössä filtteröi tuloksia pois ottamatta oikeastaan kantaa suorituskykyyn.* (H1)

REST-rajapintoihin tunnistautuessa on osa pääsynhallinnasta voitu ulkoistaa täysin API-yhdyskäytävän tehtäväksi, eritysesti sellaisissa tapauksissa, joissa mikropalvelut ja taustajärjestelmät keskustelevat keskenään:

> *Meillä on semmoisia yhteiskäyttöisiä rajapintoja, sanotaan vaikka esimerkiksi kuvitteellinen customer-rajapinta. Nämä on meillä tällä hetkellä sisäistetty API managementin logiikkaan. Se on se API management, joka on vastuussa siitä autentikaatiosta, että palvelut, joita me hallitaan, saa oikeuden vain tiettyihin rajapintoihin, joihin niillä pitää saada yhteys, jota me myös hallitaan ja käytännössä tämä on se mitä me API managementin kanssa hoidetaan.* (H4)

> *Siellä on sellainen kuin identity aware proxy, joka huolehti siitä, että liikenne mikropalveluihin kulkee vain, jos on se service tason id token, millä sä kerrot, että tämä kysely tulee gatewayltä ja pyyntöön voi palauttaa dataa.* (H2)

Jos tunnistautumisen lisäksi on mahdollista ulkoistaa pääsynhallinta API-yhdyskäytävälle, mahdollistaa se myös mikropalveluiden yksinkertaistamisen, jolloin niihin ei tarvitse toteuttaa ylimääräistä sovelluslogiikkaa. Eräs haastateltava raportoi tämän olevan mahdollista mikropalveluiden välisessä kommunikoinnissa:

> *On pyritty siihen, että niiden palveluiden ei tarvitse itse tietää tarkalleen, kuka niitä kutsuu. Me pyritään siihen, että niille palveluille ei tarvitse kirjoittaa mitään logiikkaa, että jos sinä olet sinä, niin sitten minä päästään sinut läpi, vaan se olisi pelkästään API managementin homma.* (H4)

REST-rajapintojenkaan tapauksessa pääsynhallintaa ei voi ulkoistaa täysin API-yhdyskäytävän tehtäväksi erilleen sovelluslogiikasta, eritysesti silloin kun pääsynhallinnan säännöt ovat hyvin hienojakoisia. Tällöin on usein päädytty toteuttamaan oma sovellus pääsynhallintaa varten:

> *Se [pääsynhallinta]palvelu on oikeastaan meidän tekemä siksi, että se on haluttu hienojakoiseksi just siten, että se on organisaatiokohtainen minkälaisia käyttöoikeuksia siellä halutaan jakaa ja siten sopii myös erityisvaatimuksiin.* (H3)

Myös tietoturva koettiin joissakin tapauksissa ongelmalliseksi, jos tunnistautuminen ja pääsynhallinta siirrettäisiin pois sovelluksesta yhdyskäytävään:

> *Siinä on pelko sellaisesta man-in-the-middle tyylisistä hyökkäyksistä, jos me siirretään se kaikki [tunnistautuminen] liian ylös sitä putkea.* (H2)

Erityisesti GraphQL-rajanpintoja hyödyntäessä tyypillinen menetelmä toteuttaa pääsynhallinta oli sisällyttää se rajapinnan tarjoavaan sovellukseen. Tällöin sovelluskoodissa on määritelty mihin resursseihin milläkin käyttäjäroolilla on pääsy. Tällöin jos käyttöoikeusroolille sallittuja resursseja joudutaan muuttamaan, joudutaan muutos usein tekemään ohjelmakoodiin eikä tässä voida hyödyntää mahdollista API-hallinnan tarjoamaa keskitettyä alustaa. Tätä ei kuitenkaan nähty aina ongelmallisena, sillä tällaisessa tapauksessa esimerkiksi yksikkötestejäkin täytyy muokata:

> *Tulee koodimuutosta, eli ensinnäkin, niille on yleensä aika tarkat unitestit, joten se muutos vaatii yleensä myös yksikkötestien muutosta.* (H2)

Tunnistautumisen sisällyttäminen sovelluslogiikkaan ei mahdollista useiden API-hallinta-alustojen tarjoamaa käyttäjänhallintaa integraatioille tai rajapintojen kolmannen osapuolen käyttäjille, jolloin heidän tunnuksensa täytyy luoda käsin tai sovelluksen omassa alustassa. Tätä ei kuitenkaan koettu ongelmana:

> *Toisaalta näitä konekäyttäjiä tarvitaan yleensä vasta siinä kohtaa, kun joudutaan tekemään jonkinlaisia asiakasintegraatiota, niin käytännössä sen yksittäisen käyttäjän naputtelu on niistä ongelmista kaikista pienin.* (H1)

> *DevOps-tyypit provisioi siihen palveluun uudet tunnukset tai muutokset niihin tunnuksiin.* (H3)

4.2.2 Versiointi

Kun haastateltavilta kysyttiin versioinnista, saatiin kartoitettua melko vaihetelevia käytänteitä ja toimintatapoja. Yleisesti versiointi koettiin raskaaksi eikä useissa tapauksissa rajapinnoista ylläpidetty järjestelmällisesti eri versioita:

> *Kyllä me pyritään semanttista versiointia käyttämään käytännössä kaikissa rajapinnoissa. Mä en usko, että meillä kaikki rajapinnat on siinä, mutta suurimmilta osin. [...] Totta kai pyritään aina välttämään [taaksepäin yhteensopivuuden rikkovia muutoksia] loppuun asti, koska se monimutkaistaa asioita, kun pitää monta versiota ylläpitää.* (H4)

> *Meillä ei yksinkertaisesti tällä hetkellä pienenä yrityksenä ole varaa tehdä siitä jotain adaptaatiota vanhaan API-versioon.* (H1)

> *Niissä on aina aika paljon työtä ylläpitää edes sitä kahta versiota taaksepäin. Se ei vaan ole toisen sovelluksen jättämistä ajoon, vaan ne kaikki tietomallimuutokset pitää ylläpitää tuplana* (H3)

Vaikka versiointi olisi käytössä API-hallinnan aktiviteettinä, saatettiin se toteuttaa vain osittain, jolloin vain pienestä osasta rajapinnoista on rajatun ajan saatavilla vanhempi versio:

> *Me ei olla pyritty aina systemaattisesti rakentaa taaksepäin yhteensopivuutta, vaan me ollaan pyritty pitämään niin taaksepäin yhteensopivana kuin mahdollista. Me ollaan tarkkaan tarkistettu muutoksia näihin rajapintoihin ja käyty ne aina kaikkien asiakkaiden kanssa läpi, että tällaisia muutoksia on tulossa ja jos on ollut tarvetta, sitten me ollaan rakennettu taaksepäin yhteensopivuuksia niihin.* (H3)

Haastateltava kaksi raportoi myös GraphQL-rajapintojen tapauksessa versioinnin tarpeettomuudesta:

> *No GraphQL API:han ei versioida.* (H2)

Vaikka versiointi olisi käytössä, ei versionumeron korottamista koettu kuitenkaan mielekkääksi kaikissa tapauksissa, joissa taaksepäin yhteensopivuus rikkoutuu. Jos asiakassovellus ja palvelinpuolen sovellus päivitetään samanaikaisesti, ei versionumeron korotusta välttämättä tarvita:

> *Jos me puhutaan esimerkiksi tästä selainkäyttöliittymästä niin me deployataan kuitenkin käytännössä käyttöliittymä ja taustajärjestelmät samalla hetkellä. Pahin mitä silloin voi tapahtua on se, että käyttäjä voi joutua painamaan päivitä nappia selaimessa ja ollaan todettu, että se voi olla helpompi, kun se pakotettu API-break [ja version numeron korotus].* (H1)

4.2.3 Dokumentaatio

Haastateltavat kertoivat melko yhtenäisesti rajapintojen dokumentointiin käytetyistä työkaluista ja käytänteistä. Erityisesti GraphQL-rajapintojen kanssa käytössä oli automaattisesti sovelluksen lähdekoodista luotu dokumentaatio rajapinnan kautta saatavilla olevista resursseista. Osa alustoista, joissa dokumentaatio sijaitsee, tarjoaa myös mahdollisuuden kokeilla rajapintoja ja tarkastella niiden palauttamia vastauksia. Haastateltavat kertoivat, että useat GraphQL-palvelinsovellukset tarjoavat tämän automaattisesti, eikä dokumentaation hyödyntämiseksi tarvitse tehdä juuri ylimääräistä työtä:

> *Meidän paras dokumentaatio siitä, mitä API:ssa on ja mitä sillä pystyy tekemään on meidän sisäänrakennettu GraphiQL.* (H1)

> *[Dokumentaatiota on] sen verran mitä automaattisesti generoituu. Apollo studio luo jokaisesta servicestä automaattisesti dokumentaatiosivut, missä on mahdollisia kuvauksia kentille ja entiteeteille. Se myös listaa, että tämmöinen entiteetti, kuka käyttää sitä, missä servicessä ja niin edelleen.* (H2)

On kuitenkin huomattava, että nämä sovelluksen lähdekoodissa määritellystä GraphQL-skeemasta generoidut dokumentaatiot eivät usein tarjoa esimerkkejä, kuinka saatavilla olevaa dataa tulisi käyttää. Dokumentaatioon on kuitenkin mahdollista lisätä esimerkkejä käsin määrittelemällä:

> *Kuten huomaat, tässä on aika vähän kuvattu esimerkiksi mitä bulletin API tekee. Joitain tapauksia tuolla on, sellaisia missä on pitänyt vähän kuvata ja antaa vaikka esimerkkejä, että mitä yksittäinen API tekee.* (H1)

Myös REST-rajapintojen osalta haastateltavilla oli käytössä automaattisia sovelluksen lähdekoodista dokumentaation generoivia työkaluja:

> *Me ollaan käytetty OpenAPI 3 -kuvausta, mikä tulee lähdekoodista ja sitten meillä on tällaisia HTML-muotoisia dokumentteja mitkä on siihen OpenAPI-kuvaukseen kytkettyjä tai sen kautta rakennettuja, joihin on lisätty esimerkiksi esimerkkejä ja validaatiosääntöjä mitä siitä OpenAPI-kuvauksesta ei suoraan tule.* (H3)

Usein dokumentaatio oli saatavilla jostain organisaation omasta palvelusta tai alustasta, mutta keskitettyä API-hallintaa hyödyntävissä organisaatioissa se oli saatavilla myös alustan tarjoamasta sovelluskehittäjien portaalista:

> *On keskitetty API management, josta voi aina nähdä kaikkien rajapintojen Swagger[-dokumentaation] ja end pointit, mutta sitten meillä on myös erikseen kolmannen osapuolen palvelu, jossa on tuotekohtaisesti oma dokumentaatio monestakin eri näkökulmasta.* (H4)

4.2.4 Lokitietojen keräys

Kun haastateltavilta kysyttiin lokitietojen keräyksestä, saatiin jälleen melko yhtenäisiä vastauksia, mutta lokitietojen keräyksessä ja työkaluissa oli melko suurta vaihtelua. Lähes pääsääntöisesti lokitiedot ja tilastointiin käytettävä aineisto kerättiin suoraan sovelluksesta eri ohjelmakirjastojen avulla ja aineisto lähetettiin johonkin ulkopuoliseen palveluun. Oli myös yleistä, että kaikki lokitiedot eivät sijainneet yhdessä ja samassa keskitetyssä palvelussa, vaan eri alustoihin kerättiin teemoiltaan erilaisia lokitietoja. Haastateltavat kertoivat esimerkiksi, millaista lokitietoa rajapinnoista kerätään:

> *Me logataan että mikä palvelu, mikä operaatio, yksittäisen requestin id, millä se on mennyt sisään ja sitten kuka se autorisoitu käyttäjä on.* (H1)

> *Kaikki myös audit logitetaan, joka ikisen resurssiin pääsy. [...] Jokainen requesti GraphQL-tasolla, mitä resursseja haettu ja millä roolilla.* (H2)

Myös keskitettyjä ratkaisuja lokitietojen säilömiseen käytettiin, mutta tällöinkin myös sovelluksen oma lokiin kirjotus koettiin tärkeäksi:

> *Meillä on lokitukset ELK-stakin kautta rakennettuna, eli jokainen sovelluksista ja muista infrastruktuurin komponenteista lokittaa joko Dockerin lokitusajurille josta ne viedään edelleen yhteiseen Elastic Search -indeksiin ja levylle. Vähän riippuen lokityypistä, joko sovellukset kirjoittavat itse audit lokia muutoksista ja infrastruktuurikomponentit omia lokejaan, kuten esimerkiksi Nginx kirjoittaa HTTP-access lokia.* (H3)

Eräs haastateltava raportoi myös heidän tavoitteestaan siirtyä keskitettyyn lokitietojen keräykseen, jonne kerätään esimerkiksi tietoa rajapintojen käytöstä ja ketkä niitä käyttävät. Keskitetyn lokikeräyksen vahvuuksina koettiin sen yksinkertaistavan sovelluslogiikkaa ja mahdollistavan lokitietojen hyödyntämisen yli tiimirajojen. Erityisesti keskitettyyn loikitukseen siirtymällä halutaan yhtenäistää lokitietojen keräämisen käytäntöjä, siten ettei jokaisen tiimin tarvitse ratkoa samaa asiaa useaan kertaan:

> *Jo pelkästään sen takia, että me saadaan yleinen paikka, josta voi rakentaa dashboardit jotka voi mennä tuoterajojen yli. Se on paljo helpompaa tehdä, jos meillä se data on yhdessä paikkaa. Toinen puoli on se, että tiimien ei tarvitse ruveta keksimään, millä tavalla tehdä yleisesti suoritettavia asioita, kuten vaikka millä tavalla lokitus toteutetaan. Ei sen takia, että me haluttaisiin rajoittaa devausta millään tavalla, koska se on se viimeisin asia, jota halutaan, mutta se on se, että ihmisillä on heti valmiit työkalut, joilla toimia.* (H4)

4.2.5 API-hallinnan prosessit

Haastateltavilta kysyttiin myös kokevatko he, että organisaatiossa nykyisin käytössä olevat API-hallinnan toimintatavat ja työkalut toimiviksi. Pääsääntöisesti haastateltavat kertoivat työtapojen toimivan organisaation nykyisessä tilanteessa. Myös mahdollisia uhkakuvia oli näkyvissä, jolloin tarvitsisi esimerkiksi ketterämpää tapaa versioida rajapintoja:

> *Kun kolmas osapuoli alkaa käyttää näitä [rajapintoja], silloin me kaiken järjen mukaan joudutaan erottelemaan rajapintoja erikseen, jotta me voidaan julkaista niitä irrallaan eikä yhden tarvitse välttämättä huolestua toisen muutoksista.* (H1)

Eräs haastateltava raportoi myös organisaation tehneen jo suuriakin muutoksia tapaan työskennellä rajapintojen kanssa kasvaneen kehitystiimin vuoksi. Suurempi määrä kehittäjiä loi tilanteen, ettei monoliittinen arkkitehtuuri ollut enää kehittäjien työskentelyn

kannalta järkevää ja sitä pilkottiin pienemmiksi mikropalveluiksi. Mikropalveluiden käyttöönotto loi kuitenkin tarpeen keskitetylle tavalle hallinnoida uusia rajapintoja:

> *Olen nähny tällaisia microservice-malleja, joissa on yli 50 microserviceä ja nämä microservicet pommittaa ristiin toisiaan. Se vaatii mun mielestäni toimiakseen siihen edelle jonkinlaisen orkestraatiotason. Toivon, että tämä [GraphQL-]federaatio ratkaisisi juuri tätä ongelmaa, jolloin ne palvelut ei joudu toivottavasti keskustelemaan keskenään ollenkaan suoraan itse. Jos ne [palvelut] tietää toisistaan liikaa niin se menee taas aikamoiseksi sillisalaatiksi.* (H2)

API-yhdyskäytävän käyttöä ei koettu useissa haastateltavana olleissa organisaatioissa mielekkäänä tai tarpeellisena heidän nykyisen toimintamallinsa kannalta. Esimerkiksi rajapintojen tuotteistaminen tai järjestelmällinen versiointi koettiin teemoina, joihin API-yhdyskäytävä toisi lisäarvoa, jos näitä asioita tarvitsee organisaatiossa tulevaisuudessa toteuttaa:

> *Ei ollut [API-gatewaytä] vielä käytössä. Niitä mahdollisesti voisi ottaa käyttöön siinä vaiheessa, jos niitä [rajapintoja] pitäisi tarjota tarkemmin ulospäin tai versioida, mutta nykyisellään meillä ei ollut vielä api gatewaytä käytössä.*(H3)

Rajapintojen hallinnan prosessit koettiin osin kuormittavaksi sovelluskehittäjille. Näin oli erityisesti, jos organisaatiossa oli käytössä keskitetty API-hallinta, johon liittyy tunnistautumisen, versioinnin ja dokumentaation hallintaa:

> *Kyllä siinä ihan selkeästi pitää nähdä vaivaa siihen, että asiat menee API managementin kautta. Ylipäätänsä se, että saa vaikka deployment-putket aikaan niin onhan se suuri työ, vaikka sitä helpotetaan yhteisillä työkaluilla. [...] Jotenkin se on sisäisesti hyväksytty varmaan, että kun rajapintoja on tietyn verran ja jos sitä [API-hallintaa] ei olisi ja sen tekisi ihan villi länsi meiningillä, niin ei sekään ole toimiva ratkaisu pitkällä tähtäimellä. Mä sanoisin omasta mielipiteestä, että se API managementin ylläpitäminen on erittäin iso työ tai isompi varmaan, mitä voi silleen luonnollisesti ajatella, mutta se antaa semmoisia etuja, joita on hyvin vaikeaa korvata muilla tavoilla.* (H4)

4.2.6 Yhteenveto

Tässä alaluvussa on esitetty karkea yhteenveto tapaustutkimuksen tuloksista. Taulukossa 4.3 on yhteenveto yhteisistä aiheista, joita kaikista tapausorganisaatioista raportoitiin. Taulukko ei pyri kattamaan koko haastatteluaineistoa, mutta antaa yleiskuvan haastatteluiden läpileikkaavista käytänteistä ja työkaluista.

Taulukko 4.3: Yhteenveto haastatelluista organisaatioista ja heidän käyttämistään teknologioista sekä toimintatavoista

Ominaisuus	**H1**	**H2**	**H3**	**H4**
Toimiala	SaaS-työnhallintasovellus	terveysalan ohjelmisto	SaaS-toiminnanohjausjärjestelmä	ei lupaa mainita
Sovellusarkkitehtuuri	Monoliitti	Mikropalvelu	Monoliitti	Mikropalvelu
Sovelluskehittäjien määrä (noin)	5	10	50	100
Rajapintatyyppi	GraphQL	GraphQL	REST ja GraphQL	REST
Versioidaanko rajapintoja kattavasti	Ei	Ei	Ei	Kyllä
Koettiinko keskitetty APIhallinta mielekkääksi	Ei	Ei	Ei	Kyllä
Oliko APIyhdyskäytävä käytössä	Ei	Kyllä	Ei	Kyllä
Mikä APIhallinnan tuote		GraphQL Federation		Azure APImanagement
Rajapintoihin pääsy ulkopuolisilla sovelluskehittäjillä	Kyllä	Kyllä	Kyllä	Kyllä

5 Pohdinta

Tässä luvussa analysoidaan aiemmissa osissa esiteltyjä tuloksia ja pohditaan niitä tutkielman tutkimuskysymyksien valossa. Alaluvussa 5.1 pohditaan aiemmassa luvussa esiteltyjä tuloksia ja alaluvussa 5.2 tuloksia tarkastellaan tutkimuskysymyksien valossa. Alaluvussa 5.3 pohditaan tutkimuksen luotettavuutta ja viimeisessä alaluvussa 5.4 esitellään mahdollisia tulevaisuuden tutkimuksen aiheita.

5.1 Tuloksien analysointi

Tässä osassa esitellään tutkielman keskeiset tulokset ja pohditaan kuinka haastatteluaineiston tulokset suhtautuvat kirjallisuudesta löydettyihin tuloksiin.

5.1.1 API-hallinnan tavoitteet

Tutkimuskirjallisuuden perusteella API-hallinnan käsitteellä tarkoitetaan melko suurta joukkoa eri aktiviteettejä ja toimintoja. Mitään selkeästi rajattua määritelmää siitä mitkä näistä aktiviteeteistä kuuluu rajapintojen hallinnan käsitteistöön ja mitkä ei, ei tutkimuskirjallisuudessa määritelty. Tutkielmassa tehdyn kirjallisuuskatsauksen perusteella kukin tutkija näyttääkin määrittelevän hieman omien mieltymyksien mukaan, mitkä kaikki käsitteet kuuluvat rajapintojen hallintaan.

Tutkimuskirjallisuudesta voi kuitenkin päätellä, että tunnistautumisen ja pääsynhallinnan toteuttaminen API-hallinnan komponentein ovat lähes koko kirjallisuuden läpileikkaavat aiheet ja ovat selkeästi keskeisiä tavoitteita API-hallinnassa. Tätä havaintoa tukee myös yleisempien API-yhdyskäytäväohjelmistojen ominaisuuksien tarkastelu.

Muita tutkimuskirjallisuuden perusteella keskeisiä API-hallinnan tavoitteita olivat muun muassa kuormantasaus, lokitietojen keräys, palveluiden löytäminen, koostaminen sekä versiointi. Yleisimpiin API-hallintaa tarjoaviin ohjelmistoihin verratessa suuri osa tutkimuskirjallisuuden esittelemistä toiminnallisuuksista on vähintäkin osittain tuettuna. API-yhdyskäytäväohjelmistoissa tuettuina olevien toiminnallisuuksien perusteella keskeisimpiä toimintoja edeltäneistä ovat kuormantasaus, lokitietojen keräys sekä versiointi.

Tutkimuskirjallisuuden lukuisista API-hallinnan tavoitteista löytyi kuitenkin myös hieman yllättävä puute, sillä julkaisuissa ei mainittu rajapintojen dokumentaation tarjoamista erillisenä aktiviteettinä. Rajapintojen kuvailu ja dokumentaation tarjoaminen on esimerkiksi Micorsoft Azuren rajapintojen hallinta-alustan kehittäjäportaalin keskeinen ominaisuus [19].

On kuitenkin huomattava, että mikään merkittävä API-hallinnan alusta ei pysty itsenäisesti dokumentoimaan alustan avulla hallittuja rajapintoja. Dokumentaatio on mahdollista luoda automaattisesti eri työkaluin, mutta sekin vaatii toimiakseen pääsyn sovelluksen lähdekoodiin. Ehkä tämän vuoksi dokumentaatiota ei tutkimuskirjallisuudessa eritelty yhdeksi rajapintojen hallinnan tavoitteeksi. Haastatteluiden perusteella dokumentaation tarjoaminen keskitetyn alustan kautta oli kuitenkin yksi olennaisista toiminnoista, joita API-hallinnalla pyrittiin ratkaisemaan.

5.1.2 Rajapintojen hallinta tapausorganisaatioissa

Tutkielmassa selvitettiin tapaustutkimuksella, kuinka tutkimuskirjallisuudesta tunnistettuja API-hallinnan tavoitteita toteutetaan käytännön sovelluskehityksessä, vai toteutetaanko niitä lainkaan. Haastatteluin paljastui yllättävän laaja kirjo eri työkaluja ja menetelmiä ratkoa aiemmin tunnistettuja tavoitteita, eikä yhtä selkeää rajapintojen hallinta-alustaa tai työkalua nousut esiin.

Tunnistautuminen ja pääsynhallinta

Yllättävästi aiemmin merkittäväksi rajapintojen hallinnan tavoitteeksi tunnistettu tunnistautuminen ja pääsynhallinta ei ollut suuressa osassa organisaatioista toteutettu lainkaan tutkimuskirjallisuuden kuvailemin tavoin. Yksi keskeinen syitä tälle oli GraphQL-rajapintojen hyödyntäminen, jossa rajapintaa käyttävä asiakassovellus voi itse määritellä mitä resursseja rajapinnan kautta haetaan. Tällöin pääsynhallinta on vaikeampi toteuttaa yhdyskäytävässä ja jokaisen rajapintakutsun sisältö täytyisi analysoida tarkasti yhdyskäytävässä.

GraphQL-rajapintojen pääsynhallinta ei ole myöskään kattavasti tuettu ominaisuus yleisimmissä API-yhdyskäytäväohjelmistoissa. Esimerkiksi Microsoft Azuren yhdyskäytävään tuki pääsynhallinnalle GraphQL-rajapintoja käyttäessä tuli vasta tämän tutkielman kirjoittamisen aikana [8].

Organisaation hyödyntäessä REST-rajapintoja ei tunnistautumista ja pääsynhallintaa lähtökohtaisesti koettu mielekkääksi ulkoistaa täysin API-yhdyskäytävän tehtäväksi. Tällöin syynä oli usein se, että sovelluksessa tarvittava pääsynhallinta on hyvin hienojakoista ja asiakkaan itsensä hallittavissa. Tällöin koettiin käytännöllisemmäksi rakentaa pääsynhallinta osaksi sovellusta. Haastateltavat raportoivat myös infrastruktuurin muutoksista, jotka eivät ole mahdollistaneet esimerkiksi pilvitarjoajan API-yhdyskäytävän hyödyntämistä. Haastatteluiden perusteella pääsynhallinnan ulkoistaminen pelkästään API-yhdyskäytävän tehtäväksi toimii parhaiten mikropalveluiden ja taustajärjestelmien välisessä kommunikoinnissa REST-rajapinnoin.

Versiointi

Rajapintojen versioinnista tapausorganisaatioista löytyi vaihtelevia käytäntöjä järjestelmällisesti versioiduista rajapinnoista täysin versioimattomiin. Jälleen GraphQL-rajapinnat erottuivat käytännöiltään REST-rajapintoihin verrattuna, sillä ne voidaan toteuttaa kokonaan ilman tutkimuskirjallisuudesta tunnistettua versiointia. GraphQL-kyselykieltä ylläpitävä GraphQL-säätiö jopa suosittelee käytännöissään versioinnin välttämistä [9].

Yleisesti rajapintojen versiointia hyödyntävät tapausorganisaatiot kokivat sen kuitenkin melko raskaaksi prosessiksi, jonka vuoksi versiointia ei usein hyödynnetty järjestelmällisesti kaikissa rajapinnoissa. API-yhdyskäytävän hyödyntäminen versioinnin työkaluna koettiin mielekkääksi vasta, kun organisaatiolla on tarve versioida systemaattisesti kaikkia tarjoamiaan rajapintoja tai käytössä olevien rajapintojen määrä on hyvin suuri. Tarve versioida rajapintoja systemaattisesti voi tulla esimerkiksi integraatioista, jolloin versioinnilla voidaan varmistaa, etteivät kolmannen osapuolen kehittämät sovellukset mene rikki rajapintamuutoksista.

Erityisen raskaaksi rajapintojen versioinnissa koettiin tietomallimuutoksien taaksepäin yhteensopivuuden tukeminen, jota varten tarvitaan ylimääräistä sovelluslogiikkaa eri versioiden välille. On huomattava, että tätä ongelmaa API-yhdyskäytävän hyödyntäminen ei ratkaise, vaan yhdyskäytävä tarjoaa vain järjestelmällisen tavan eritellä rajapintojen eri versiot ja reitittää pyyntöjä halutun rajapintaversion perusteella.

Lokitietojen keräys ja hallinta

Tutkimuskirjallisuuden perusteella olisi voinut helposti olettaa, että rajapintoihin liittyvien lokitietojen keräys on tehokasta ulkoistaa esimerkiksi API-yhdyskäytävälle, jolloin it-

se sovelluksien ei tarvitsisi huolehtia siitä. Haastatteluiden perusteella näin ei kuitenkaan ollut kaikissa API-yhdyskäytävää hyödyntävissä tapausorganisaatioissa. Usein syyksi raportoitiin tarve mahdollisimman tarkalle ja monipuoliselle lokitietojen keräykselle, jolloin API-yhdyskäytävän tarjoamat lokitiedot eivät ole riittäviä.

Usein organisaatioissa koettiinkin helpommaksi toteuttaa lokitietojen keräys suoraan sovelluksessa, jolloin on helppo määritellä, mitä tietoja halutaan kerätä. Osa organisaatioista keräsi jopa hyvin yksityiskohtaisesti tietoa, jotta tarvittaessa lokitiedoista voi etsiä apua ohjelmavirheiden korjaamiseen tai tuottaa tilastotietoja eri tarkoituksiin.

Haastateltavat raportoivat keskitetyn lokitietojen keräyksen mahdollistavan työkalujen ja käytänteiden yhtenäistämisen koko organisaatiossa. Tämä koettiin kuitenkin mielekkääksi vasta, kun organisaatiossa kehitetään useita eri tuotteita ja sovelluskehitystiimit eivät tiedä tarkalleen, mitä muut tiimit tekevät. Haastatteluiden perusteella tapausorganisaatioissa lokitietojen keräys ja säilyttäminen keskitetyssä järjestelmässä on mielekästä, jos sovellukset sijaitsevat jo valmiiksi sellaisen pilvipalveluntarjoajan järjestelmässä, joka tarjoaa myös API-hallintaa.

Keskitetyn lokitietojen säilömisen eduiksi raportoitiin myös mahdollisuus hallita, ketkä pääsevät näkemään lokitietoja. Tämän kerrottiin olevan tarpeellista silloin, kun lokitiedot voivat sisältää henkilötietoja tai muuta salassa pidettävää tietoa. Tällöin keskitettyyn alustaan voidaan määrittää rajattu joukko työntekijöitä, jotka pääsevät näkemään tällaisia lokitietoja. Keskitetty alusta lokitiedoille koettiin myös mielekkääksi, jos tietoja tarvitsee hyödyntää laajasti eri tiimien välillä. Tämä koettiin hankalaksi silloin, jos aineisto on hajautettuna useaan eri paikkaan.

Dokumentaatio

Haastatteluin kartoitettiin myös, kuinka käytössä olevat rajapinnat ovat dokumentoitu, vaikka tutkimuskirjallisuus ei suoraan tähän teemaan ottanutkaan kantaa. Haastatteluissa pyrittiin pureutumaan erityisesti, onko dokumentaatioon ja sen ylläpitämiseen yhteisiä käytänteitä ja työkaluja koko organisaation tasolla.

Jälleen pääasiallisesti GraphQL-rajapintoja hyödyntäneet organisaatiot erottuivat dokumentaation generoinnissa ja hyödyntämisessä. GraphQL mahdollistaa helposti automaattisen dokumentaation generoinnin, joka sisältää tiedon niistä resursseista ja toiminnoista, joita rajapinnan avulla on saatavilla. On kuitenkin huomattava, ettei tämä automaattisesti generoitu dokumentaatio ota kantaa siihen, mitä saatavilla olevat entiteetit ovat tai

miten niitä tulisi käyttää.

Automaattisesti generoituun dokumentaatioon on mahdollista lisätä tarkentavia tietoja tai esimerkkejä, mutta tällainen käytäntö ei ollut laajasti käytössä haastatelluissa organisaatioissa. Käsin dokumentointia vältettiin, sillä käsin luotu dokumentaatio jää helposti päivittämättä rajapintamuutoksien yhteydessä. Automaattisesti generoidun GraphQL-dokumentaation yksinkertaisuuden vuoksi sitä käyttivät lähinnä organisaatiossa työskentelevät sovelluskehittäjät ja joissakin harvoissa tapauksissa myös integraatioita hyödyntävät avainasiakkaat.

Myös REST-rajapinnoille on mahdollista generoida automaattisesti dokumentaatio, mutta se vaatii sovelluskehittäjiltä enemmän työtä ja automaattisen generoinnin huomioonottamista kuin GraphQL-rajapintojen kanssa. Eräs haastateltava kertoi käytännöstä, jossa dokumentaatio luotiin ohjelmakoodiin lisättyjen annotaatioiden sekä yksikkötestien perusteella. Tällöin dokumentaatio sisälsi kuvauksen siitä, kuinka rajapintaa voidaan käyttää, mitä resursseja se tarjoaa sekä mahdolliset validaatiosäännöt rajapintaan kirjoittamista varten.

Usein dokumentaatio rajapinnoista oli saatavilla jostakin organisaation omasta sovelluksesta, ja sitä käyttivät lähinnä organisaatiossa työskentelevät sovelluskehittäjät. Jos organisaatiossa oli käytössä jokin keskitetyn API-hallinnan tuote, hyödynnettiin sen tarjoamaa kehittäjäportaalia dokumentaation tarjoamiseen, mutta kehittäjäportaalin rinnalla oli aina myös muita palveluita dokumentaation tarjoamiseen.

5.1.3 Kuinka API-hallinta koetaan?

Haastatteluin pyrittiin myös kartoittamaan, toimiiko organisaatiolla käytössä olevat API-hallinnan käytänteet ja koetaanko niiden tukevan organisaation kykyä tuottaa ohjelmistoa. Haastatteluihin valikoitui eri kokoisia organisaatioita pienestä alle kymmenen hengen ohjelmistotalosta suurempiin organisaatioihin. Näin haastatteluin saatiin kartoitettua melko vaihtelevia käytäntöjä.

Pienen ohjelmistotalon haastattelusta ilmeni, että heidän nykyinen tapansa hallita rajapintoja ilman keskitettyä API-hallintaa tai -yhdyskäytävää, on toimiva organisaation nykyisessä tilanteessa. Haastateltava raportoi kuitenkin, että heidän nykyinen toimintatapansa ei toimisi, jos ohjelmistokehitystä tekisi useampi tiimi tai tuote koostuisi useista mikropalveluista. On mahdollista, että keskitettyä API-hallintaa ei välttämättä tarvita, jos ohjelmistokehitystä tekee vain pieni määrä kehittäjiä tai sovellusta ei ole rakennettu

mikropalveluarkkitehtuurilla.

Keskitetyn API-hallinnan vahvuus kuitenkin huomattiin jo hieman suuremmassa organisaatiossa, jossa sovelluskehitystä tekeviä tiimejä oli useampi ja tuotteet koostuivat mikropalveluista. Tällöin erityisesti mikropalveluiden välistä kommunikointia koordinoiva yhdyskäytävä koettiin mielekkääksi ja arkkitehtuuria yksinkertaistavaksi elementiksi. Haastatteluiden perusteella tapausorganisaatioissa keskitetty API-hallinta ei kuitenkaan ole edellytys tällaisessa tilanteessa.

Erityisesti suurehkossa yli sadan sovelluskehittäjän organisaatiossa keskitetty API-hallinta koettiin kuitenkin oleellisena työkaluna toimintatapojen selkeyttämiseen. Ilman selkeitä ja yhteisiä toimintatapoja, esimerkiksi rajapintojen välisessä kommunikoinnissa tai versioinnissa, koettaisiin sovelluskehitys liian sekavaksi ja ylimääräistä työtä teettäväksi. Yhteisillä työkaluilla ja toimintatavoilla sovelluskehitystä voidaan tehostaa, eikä jokaisen tiimin tarvitse kehittää omaa ratkaisuaan eri API-hallinnan tavoitteisiin, kuten tunnistautumiseen, versiointiin tai lokitietojen keräämiseen. Eräs haastateltava raportoi myös keskitetyn API-yhdyskäytävän luoman yksittäisen yhdyspisteen lisäävän tietoturvaa, sillä se pienentää merkittävästi mikropalveluihin kohdistuvaa hyökkäyspinta-alaa.

Haastatteluiden perusteella voidaan huomata, että tapausorganisaatioissa on toteutettu kirjallisuuden kuvaamaa API-hallintaa eri laajuuksin, johon tutkimuskirjallisuus ei ottanut lainkaan kantaa. Yksittäiset API-hallinnan tavoitteet voi toteuttaa keskitettynä ja hajautettuna ratkaisuna samanaikaisesti, ilman että ne sulkevat toisiaan pois. Esimerkiksi tunnistautumisen voi tehdä osittain API-yhdyskäytävässä ja hienojakoisemmin osana sovelluslogiikkaa. On myös mahdollista toteuttaa vain osa API-hallinnan tavoitteista keskitettyä alustaa hyödyntäen ja osa sovellustasolla organisaation tarpeiden mukaan räätälöitynä.

API-yhdyskäytävän käyttöä ei useissa haastateltavissa organisaatioissa koettu lähtökohtaisesti mielekkääksi, vaan sen käytölle haluttiin usein jokin selkeä tarve, jonka toteuttamiseen yhdyskäytävää tarvitaan. Tällainen voi olla esimerkiksi rajapintojen tarjoaminen laajasti ulkoisille käyttäjille tai versiointi siten, että kaikki rajapinnat kuuluvat versioinnin pariin.

Haastateltavat raportoivat sovelluskehittäjien kokevan organisaatiossa käytössä olevat API-hallinnan käytänteet luonnollisena osana työtään, eikä niiden koettu aiheuttavan kohtuutonta työtaakkaa. Osassa organisaatiossa tunnistettiin API-hallinnan tuoma kuorma ja ylimääräinen työ sovelluskehittäjille, mutta se hyväksyttiin API-hallinnan tuomien hyötyjen vuoksi. Jossain määrin myös API-hallinnan käyttöönoton tuoma muutos työskentely-

tavoissa ja työkaluissa koettiin kuormittavaksi.

Erityisesti pilvipalvelutarjoajien API-hallinnan tuotteita hyödyntäessä on mahdollisuus ajautua tilanteeseen, jossa organisaation toimintatavat ja työkalut ovat vahvasti palveluntarjoajasta riippuvaisia. Eräs haastateltava raportoi vaikeuksista hyödyntää usean pilvipalveluntarjoajan resursseja, sillä jokaisella palveluntarjoajalla on omat API-hallinnan työkalut, jolloin samoja konfiguraatioita täytyisi ylläpitää useassa palvelussa.

5.2 Tutkimuskysymyksien tarkastelu

Seuraavissa osioissa käsitellään tutkielman keskeisiä tuloksia tutkimuskysymyksien valossa.

Mitä asioita API-hallinta pyrkii saavuttamaan?

Ensimmäinen tutkimuskysymys pyrki selvittämään minkälaisia toimintoja ja tavoitteita API-hallinnalla pyritään saavuttamaan ja tunnistaa niistä merkittävimmät. Tähän pyrittiin vastaaman tekemällä strukturoitu kirjallisuuskatsaus. Tutkimuskirjallisuuden perusteella saatiin eriteltyä yhteensä 17 erillistä aktiviteettiä tai tavoitetta, jota API-hallinnalla pyritään saavuttamaan. Yhteensä kymmenen näistä mainittiin useammassa kuin yhdessä kirjallisuuskatsaukseen sisällytetyssä julkaisussa.

Tutkimuskirjallisuuden perusteella selkeästi keskeisin API-hallinnan tavoite on tunnistautumisen ja pääsynhallinnan toteuttaminen jollakin keskitetyllä alustalla. Tällöin nämä toiminnot voidaan ulkoistaa toteutettavaksi API-hallinnan komponenteissa eikä esimerkiksi mikropalveluiden tarvitse ottaa kantaa näihin. Loput toiminnoista jakautuivat tasaisemmin, joista merkittävimpiä olivat esimerkiksi kuormantasaus, lokitietojen keräys ja versiointi. Kirjallisuuden perusteella nämä tavoitteet toteutettiin usein API-yhdyskäytävää hyödyntämällä.

Kuinka API-hallinta on toteutettu käytännön sovelluskehityksessä?

Toinen tutkimuskysymys pyrki selvittämään kuinka kirjallisuudesta tunnistettuja tavoitteita ja käytäntöjä on tapausorganisaatioissa toteutettu ja kuinka käytännön kokemukset

vertautuvat tutkimuskirjallisuuteen. Käytännön toimintatapoja ja kokemuksia pyrittiin selvittämään tekemällä yhteensä neljä teemahaastattelua eri organisaatiossa.

Tapausorganisaatioiden sovelluskehityksessä API-hallintaa ei ollut pääsääntöisesti toteutettu hyödyntämällä keskitettyä API-hallinnan alustaa. Vain yksi neljästä tarkastellusta organisaatiosta hyödynsi kattavasti keskitettyä API-hallintaa pilvipalvelutarjoajan alustalla.

Usein API-hallinnan tavoitteet olivat toteutettu eri työkaluin organisaation tarpeiden mukaan räätälöidysti. Merkittävin poikkeama tutkimuskirjallisuuteen löytyi tunnistautumisesta ja pääsynhallinnasta, joka oli toteutettu kaikissa organisaatioissa ainakin osittain osaksi sovelluslogiikkaa. Pääsynhallinta oli usein toteutettu organisaatiossa itse, eikä sitä ollut tutkimuskirjallisuuden tapaan ulkoistettu esimerkiksi API-yhdyskäytävälle. Jos organisaatiossa oli käytössä keskitetty API-hallinta, pystyttiin sille ulkoistamaan tunnistautuminen mikropalveluiden välisessä kommunikoinnissa, jolloin mikropalveluidenpalveluiden ei tarvitse ottaa kantaa käyttäjän tunnistamiseen tai pääsynhallintaan.

Myös muut API-hallinnan tavoitteet olivat usein toteutettu hajautetusti hyödyntäen valmiita työkaluja. Esimerkiksi dokumentaation tarjoamisessa ei pääsääntöisesti hyödynnetty API-hallinnasta tuttuja kehittäjäportaaleja, vaan tarvittava dokumentaatio tarjottiin linkein eri alustoihin. Vaikka käytössä olisi API-hallinnan tarjoama kehittäjäportaali oli sen rinnalla käytössä myös muita ratkaisuja dokumentaation jakeluun.

Lokitietojen kerääminen ja käsittely keskitetysti nähtiin vahvuutena ja mahdollistajana kerätyn tiedon tehokkaampaan hyödyntämiseen. Lokeja ei kuitenkaan haastatelluissa organisaatioissa kerätty kokonaisuudessaan keskitettyyn alustaan, mutta suunnitelmista siirtyä tällaiseen ratkaisuun raportoitiin. Tavoitteiksi, joita keskitetty lokitietojen keräys ratkaisee, raportoitiin esimerkiksi lokitietojen yhtenäistäminen, niiden helpompi hyödyntäminen sekä sovelluskehittäjien työn yksinkertaistaminen.

Usein keskitetyn API-hallinnan toteuttamiselle ei nähty tarvetta. Tämä johtui rajapintojen pienestä määrästä tai keskitetyn alustan käyttöönoton vaatimasta suuresta työmäärästä. Usea organisaatio kuitenkin tunnisti tarpeen keskitetylle API-hallinnalle, jos rajapintoja halutaan tarjota nykyistä laajemmin kolmannen osapuolen käyttäjille. Keskitetyn API-hallinnan eduiksi raportoitiin muun muassa sen selkeyttävän taustajärjestelmien ja mikropalveluiden välistä kommunikointia.

Kuinka API-hallinta koetaan käytännön sovelluskehityksessä?

Kolmas ja viimeinen tutkimuskysymys pyrki selvittämään kuinka, tapausorganisaatioissa koetaan heillä käytössä olevat API-hallinnan toimintatavat ja työkalut. Kaikki haastateltavat organisaatiot kokivat toimintatapansa nykyisessä tilanteessaan toimiviksi, mutta esimerkiksi mahdollinen liiketoiminnan kasvu nähtiin uhkana nykyiselle tavalle hallita rajapintoja.

Organisaatioissa ei koettu, että API-hallinnan työkalut tai toimintatavat lisäisivät merkittävästi sovelluskehittäjien kuormaa ja täten haittaisi esimerkiksi organisaation kykyä tuottaa ohjelmistoja. Eräässä organisaatioissa API-hallinnan sovelluskehittäjille tuoma kuorma tunnistettiin, mutta samalla se koettiin vähemmän kuormittavaksi kuin mitä tilanne olisi ilman keskitettyä rajapintojen hallintaa.

Osa organisaatioista raportoi rajapintojen versioimattomuuden tuottavan muutoksien tekemiseen ylimääräistä työtä, jolloin taaksepäin yhteensopivuuteen joutuu kiinnittämään erityistä huomiota ja varmistamaan etteivät mahdolliset integraatiot rikkoudu muutoksien myötä. Tämän tuoma ylimääräinen työtaakka koettiin kuitenkin pienemmäksi, kuin mitä esimerkiksi API-yhdyskäytävän ja järjestelmällisen versioinnin käyttöönotto tuottaisi, eikä näitä muutoksia haluttu tehdä, ennen kuin se on organisaation kannalta välttämätöntä.

5.3 Tutkimuksen luotettavuus

Kirjallisuuskatsauksen aineisto voi olla vinoutunut esimerkiksi epäoptimaalisen hakulausekkeen vuoksi. Lumipallomenetelmä kuitenkin osaltaan varmistaa, että myös hakulausekkeen ulkopuolelta saadaan tutkimuskirjallisuutta analysoitavaksi. Hakulausekkeen rajaaminen käsittelemään mikropalveluita ja rajaus korkeintaan viisi vuotta vanhoihin julkaisuihin on voinut vaikuttaa aiemmin todettuun dokumentaation puuttumiseen tunnistetuista API-hallinnan aktiviteeteistä. Mathijssen ym. [17] ovat huomattavasti laajemmassa systemaattisessa kirjallisuuskatsauksessa tunnistaneet dokumentaation tarjoamisen yhdeksi API-hallinnan toiminnallisuudeksi.

On kuitenkin huomattava, ettei API-hallinnan aiheesta löydy suurta määrää tutkimuskirjallisuutta, joten on mahdollista, että analysoitavaksi valikoituneet julkaisut eivät kuvaa kattavasti kirjallisuuskatsauksen aihetta. API-hallinnalle ei myöskään ole tutkimuksessa vakiintunutta ja yleisesti hyväksyttyä määritelmää, joten kirjallisuudesta tunnistetut tavoitteet voivat vaihdella tutkijan näkemyksien perusteella. Tutkimuskirjallisuuden rajalli-

suudesta johtuen kirjallisuuskatsaukseen on valikoitu myös harmaata ja vertaisarvioimatonta kirjallisuutta.

Haastatteluin tutustuttiin yhteensä neljän eri organisaation tapaan toteuttaa API-hallinta. Organisaatiot kattoivat melko laajasti eri kokoluokan organisaatioista aina muutamasta sovelluskehittäjästä suurempaan yli sadan sovelluskehittäjän organisaation. Haastateltavaksi valittujen organisaatioiden valintatapa ja vähäinen määrä on todennäköisesti vaikuttanut haastatteluiden tuloksiin, sillä haastateltavat valikoituivat lähinnä tutkijan omien kontaktien kautta, eikä esimerkiksi mahdollisimman kattavaan otantaan pyrkien. Osa tavoitelluista yrityksistä myös kieltäytyi haastattelusta. On kuitenkin huomioitava, ettei laadullisessa tutkimuksessa usein pyritä mahdollisimman kattavaan ja suureen otantaan.

Haastatteluissa käsiteltiin lähinnä kirjallisuuskatsauksen avulla löytyneitä aiheita, jotka haastateltavien perusteella kuvasivat hyvin API-hallintaa ja heidän kokemuksiaan aiheesta. Rajallinen haastatteluaika rajoitti kuitenkin keskustelun siirtymistä aiheisiin, joita ei kirjallisuuskatsauksen perusteella luodussa haastattelurungossa mainittu. Haastatteluaineiston analysoinnissa tunnistetut teemat mukailivat melko vahvasti haastattelunrungon teemoja, joten myös tutkijan oma näkemys aiheeseen on saattanut vaikuttaa analysointivaiheessa tunnistettuihin teemoihin.

5.4 Tulevaisuuden tutkimusaiheita

Tässä tutkielmassa kartoitettiin kirjallisuuskatsauksen ja tapaustutkimuksen avulla mitä API-hallinnalla pyritään saavuttamaan ja kuinka tämä on toteutettu tapausorganisaatioiden sovelluskehityksessä. Haastatteluiden perusteella vaikuttaa siltä, että keskitetyt API-hallinnan ratkaisut eivät olleet yhtä yleisiä tapausorganisaatioissa kuin tutkimuskirjallisuudesta saattaisi olettaa.

Tulevaisuuden tutkimuksessa olisikin mielenkiintoista tutkia, milloin organisaation kannattaa siirtyä hyödyntämään keskitettyä API-hallintaa tai API-yhdyskäytävää. Esimerkiksi milloin tausta- ja mikropalveluiden väliseen kommunikointiin kannattaa hyödyntää API-yhdyskäytävää vai voiko sen tuoman hyödyn saavuttaa myös muilla työkaluilla. Mielenkiintoista on myös tutkia kuinka paljon keskitetyn API-hallinnan luoma työtaakka lisää sovelluskehittäjien työtä ja tutkia, milloin ja minkälaisissa olosuhteissa työmäärä on saavutettuja etuja pienempi.

Eräs haastateltava nosti esiin myös service mesh -teknologian, joka voisi korvata keskitetyn API-hallinnan ja -yhdyskäytävän mikropalveluiden välisessä kommunikoinnissa. Service mesh teknologia tarjoaakin osittain samoja toiminnallisuuksia mikropalveluiden väliseen kommunikointiin kuin API-hallinta, esimerkiksi tunnistautumista, lokitietojen keräystä ja versiointia [16]. Tällöin API-hallinta voisi keskittyä vain hallitsemaan organisaation sisäverkon ulkopuolelle asiakkaille näkyviä rajapintoja.

Serivce mesh tarjoaa myös palveluntarjoajasta riippumattoman ratkaisun, joka on helposti siirrettävissä ympäristöstä toiseen [16]. Tämä ei pilvipalveluntarjoajien API-hallintaratkaisuilla onnistu, sillä jokaisen palveluntarjoajan alusta on erilainen eikä niiden välillä voi helposti jakaa konfiguraatioita. Tämä on mielenkiintoinen tutkimuksen aihe, milloin ja minkälaisissa ympäristöissä service mesh toimii API-hallinnan sijasta tai sen rinnalla.

6 Yhteenveto

Tämän maisteritutkielman tavoitteena oli tutkia tutkimuskirjallisuuden avulla mitä API-hallinnalla tavoitellaan ja tutustua tapaustutkimuksen avulla, kuinka nämä tavoitteet ovat käytännön sovelluskehityksessä toteutettu. Lisäksi tutkielmassa selvitettiin, kuinka API-hallinnan käytännöt ja työkalut koetaan sovelluskehitystä tekevissä tapausorganisaatioissa. Tutkimusmenetelmäksi tähän työhön valittiin strukturoitu kirjallisuuskatsaus ja tapaustutkimus, joka kattoi neljä organisaatiota.

Kirjallisuuskatsauksen avulla kartoitettiin, kuinka tutkimuskirjallisuudessa käsitellään API-hallintaa. Tutkimuskirjallisuudesta tunnistettiin eri API-hallinnan tavoitteita, joiden perusteella luotiin haastattelurunko tapaustutkimuksen teemahaastatteluja varten. Tunnistettuja tavoitteita tai toiminnallisuuksia löytyi tutkimuskirjallisuudesta melko suuri määrä, joten tästä joukosta poimittiin oleellisimmat tavoitteet käsiteltäviksi haastatteluissa.

Tapaustutkimuksessa haastateltiin yhteensä neljää sovelluskehittäjää eri organisaatioista. Haastatteluiden avulla pyrittiin kartoittamaan, kuinka haastateltavan organisaatiossa on toteutettu kirjallisuudesta tunnistetut API-hallinnan tavoitteet. Lisäksi haastatteluin pyrittiin selvittämään, kuinka tapausorganisaatiossa työskentelevät sovelluskehittäjät kokevat heillä käytössä olevat API-hallinnan käytänteet ja työkalut.

Tapaustutkimuksen perusteella organisaatioissa oli käytössä melko vaihtelevia tapoja toteuttaa eri API-hallinnan tavoitteita. Haastatteluaineiston perusteella pienemmissä tapausorganisaatioissa ja tuotteissa ei keskitetylle API-hallinnalle nähty tarvetta ja tarpeelliset toiminnallisuudet toteutettiin usein osana tuotteen sovelluslogiikkaa. Organisaation ja tuotteen kasvaessa nähtiin kuitenkin keskitetyn API-hallinnan hyödyt ja sen sovelluskehittäjille tuoma työtaakka hyväksyttiin sen tuomien hyötyjen vuoksi.

Yhteenvetona voi todeta, että tapausorganisaatioissa kaikkia API-hallinnan tavoitteita ei voi ulkoistaa keskitetyllä alustalla toteutettavaksi, vaan osa toiminnallisuuksista pitää lähes aina jossakin laajuudessa toteuttaa myös tuotteen sovelluslogiikkaan. Siitäkin huolimatta API-hallinta nähtiin tutkimukseen osallistuneissa organisaatioissa mielekkäänä tapana pitää lukuisista rajapinnoista koostuva kokonaisuus yhtenäisenä ja helposti muokattavana. Tuloksien valossa API-hallinta koettiin tapausorganisaatioissa tehokkaaksi työkaluksi rajapintojen hallintaan, mutta se ei ratkaissut kaikkia haasteita eikä sen käyttö kaikissa tapauksissa ollut välttämättä edes mielekästä.

Lähteet

[1] *Amazon: API Management*. URL: https://aws.amazon.com/api-gateway/api-management/ (viitattu 19. 09. 2022).

[2] B. De. *API Management: An Architect's Guide to Developing and Managing APIs for Your Organization*. Apress, 2017.

[3] N. Dragoni, S. Giallorenzo, A. Lluch-Lafuente, M. Mazzara, F. Montesi, R. Mustafin ja L. Safina. "Microservices: yesterday, today, and tomorrow". Teoksessa: *Present and Ulterior Software Engineering*. Springer International Publishing, kesäkuu 2016, s. 195–216.

[4] J. Eskola. *Johdatus laadulliseen tutkimukseen*. Tampere: Vastapaino, 1998.

[5] D. G. Feitelson, E. Frachtenberg ja K. L. Beck. "Development and Deployment at Facebook". *IEEE Internet Computing* 17.4 (2013), s. 8–17.

[6] R. T. Fielding. "Architectural Styles and the Design of Network-Based Software Architectures". PhD Thesis. University of California, Irvine, 2000.

[7] M. Fowler. *Microservices*. Maaliskuu 2014. URL: https://martinfowler.com/articles/microservices.html (viitattu 15. 10. 2022).

[8] *General availability: GraphQL passthrough support in Azure API Management*. Toukokuu 2022. URL: https://azure.microsoft.com/en-us/updates/general-availability-graphql-passthrough-support-in-azure-api-management/ (viitattu 28. 09. 2022).

[9] *GraphQL Best Practices: Versioning*. URL: https://graphql.org/learn/best-practices/#versioning (viitattu 28. 09. 2022).

[10] *GraphQL specification*. Lokakuu 2021. URL: https://spec.graphql.org/October2021/ (viitattu 28. 09. 2022).

[11] *Gravitee Open Source API Management Platform*. URL: https://www.gravitee.io (viitattu 04. 02. 2022).

[12] S. Hirsjärvi. *Tutkimushaastattelu : teemahaastattelun teoria ja käytäntö*. Helsinki: Gaudeamus Helsinki University Press, 2008.

[13] G. Hohpe, B. Woolf, K. Brown ja M. Fowler. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. A Martin Fowler signature book. Addison-Wesley, 2004.

[14] N. S. Jones M. Bradley J. *RFC 7519: JSON Web Token (JWT)*. Toukokuu 2015. URL: https://www.rfc-editor.org/rfc/rfc7519.

[15] S. Keele et al. *Guidelines for performing systematic literature reviews in software engineering*. Tekninen raportti. Technical report, Ver. 2.3 EBSE Technical Report. EBSE, 2007.

[16] W. Li, Y. Lemieux, J. Gao, Z. Zhao ja Y. Han. "Service Mesh: Challenges, State of the Art, and Future Research Opportunities". Teoksessa: *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. 2019, s. 122–1225.

[17] M. Mathijssen, M. Overeem ja S. Jansen. "Identification of Practices and Capabilities in API Management: A Systematic Literature Review". *arXiv* (2020).

[18] Microsoft. *API Management in a Hybrid and Multi- Cloud World*. Tekninen raportti. 2019. URL: https://azure.microsoft.com/mediahandler/files/resourcefiles/api-management-in-a-hybrid-and-multi-cloud-world/API%20management%20in%20a%20hybrid%20and%20multi-cloud%20world.pdf.

[19] *Microsoft: About API Management*. Heinäkuu 2022. URL: https://learn.microsoft.com/en-us/azure/api-management/api-management-key-concepts (viitattu 19. 09. 2022).

[20] E. Raymond. *The Art of UNIX Programming*. Addison-Wesley Professional Computing Series. Pearson Education, 2003.

[21] L. Richardson ja S. Ruby. *RESTful Web Services*. O'Reilly Media, 2008.

[22] P. Runeson ja M. Höst. "Guidelines for conducting and reporting case study research in software engineering". *Empirical Software Engineering* 14.2 (joulukuu 2008), s. 131.

[23] H. Snyder. "Literature review as a research methodology: An overview and guidelines". *Journal of Business Research* 104 (2019), s. 333–339.

[24] *Toska, Helsingin yliopiston tietojenkäsittelytieteen osaston sovelluskehitysakatemia*. URL: https://toska.dev/ (viitattu 18. 10. 2022).

[25] J. Tuomi. *Laadullinen tutkimus ja sisällönanalyysi*. Helsinki: Tammi, 2002.

[26] *Tyk Open Source API Gateway*. URL: https://tyk.io/open-source/ (viitattu 04. 02. 2022).

[27] C. Wohlin. "Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering". Teoksessa: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE '14. New York, NY, USA: Association for Computing Machinery, 2014.

Liite A Kirjallisuuskatsauksen aineisto

Hakulausekkein löydetyt lähteet

[A1] A. Akbulut ja H. Perros. ”Software Versioning with Microservices through the API Gateway Design Pattern”. Teoksessa: *2019 9th International Conference on Advanced Computer Information Technologies (ACIT)*. Kesäkuu 2019, s. 289–292.

[A2] S. Andreo ja J. Bosch. ”API Management Challenges in Ecosystems”. Teoksessa: *International Conference on Software Business*. Springer Cham, lokakuu 2019, s. 86–93.

[A3] A. Bánáti, E. Kail, K. Karóczkai ja M. Kozlovszky. ”Authentication and authorization orchestrator for microservice-based software architectures”. Teoksessa: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2018, s. 1180–1184.

[A4] M. Biehl. *API Architecture: The Big Picture for Building APIs*. CreateSpace Independent Publishing Platform, 2015.

[A5] J. Carneiro, R. Andrade, P. Alves, L. Conceição, P. Novais ja G. Marreiros. ”A consensus-based group decision support system using a multi-agent microservices approach”. Teoksessa: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*. Vol. 2020-May. 2020, s. 2098–2100.

[A6] Y. Dawei, G. Yang, H. Wei ja L. Kai. ”Design and Achievement of Security Mechanism of API Gateway Platform Based on Microservice Architecture”. Teoksessa: *Journal of Physics: Conference Series*. Vol. 1738. 2021.

[A7] B. De. *API Management: An Architect's Guide to Developing and Managing APIs for Your Organization*. Apress, 2017.

[A8] S. Gadge ja V. Kotwani. ”Microservice architecture: API gateway considerations”. *GlobalLogic Inc* (2017). URL: https://www.globallogic.com/wp-content/uploads/2017/08/Microservice-Architecture-API-Gateway-Considerations.pdf.

[A9] A. Gamez-Diaz, P. Fernandez ja A. Ruiz-Cortés. "Governify for APIs: SLA-Driven Ecosystem for API Governance". Teoksessa: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2019. Association for Computing Machinery, 2019, s. 1120–1123.

[A10] D. Guaman, L. Yaguachi, C. C. Samanta, J. H. Danilo ja F. Soto. "Performance evaluation in the migration process from a monolithic application to microservices". Teoksessa: *13th Iberian Conference on Information Systems and Technologies (CISTI)*. Kesäkuu 2018, s. 1–8.

[A11] R. T. Hartono, M. Rahayu ja P. N. Taufik. "e-Control: Electronic Attendance Control System for Multi-Condition Class using Microservices Architecture Development Methods". Teoksessa: *2021 4th International Conference of Computer and Informatics Engineering (IC2IE)*. Syyskuu 2021, s. 299–304.

[A12] S. Haselböck, R. Weinreich, G. Buchgeher ja T. Kriechbaum. "Microservice Design Space Analysis and Decision Documentation: A Case Study on API Management". Teoksessa: *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)*. 2018, s. 1–8.

[A13] X. He ja X. Yang. "Authentication and Authorization of End User in Microservice Architecture". Teoksessa: *Journal of Physics: Conference Series*. Vol. 910. 2017.

[A14] A. Ivanchikj ja C. Pautasso. "Modeling microservice conversations with RESTalk". Teoksessa: *Microservices: Science and Engineering*. Springer International Publishing, 2019, s. 129–146.

[A15] S. Kapembe ja J. Quenum. "Lihonga-a microservice-based virtual learning environment". Teoksessa: *Proceedings - IEEE 18th International Conference on Advanced Learning Technologies, ICALT 2018*. 2018, s. 98–100.

[A16] D. Lu, D. Huang, A. Walenstein ja D. Medhi. "A Secure Microservice Framework for IoT". Teoksessa: *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. Huhtikuu 2017, s. 9–18.

[A17] P. Malo-Perisé ja J. Merseguer. "The "Socialized Architecture": A Software Engineering Approach for a New Cloud". *Sustainability (Switzerland)* 14.4 (2022).

[A18] D. Müssig, R. Stricker, J. Laessig ja J. Heider. "Highly Scalable Microservice-based Enterprise Architecture for Smart Ecosystems in Hybrid Cloud Environments". Teoksessa: *ICEIS*. Tammikuu 2017, s. 454–459.

[A19] P. Nguyen, H. Song, F. Chauvel, R. Muller, S. Boyar ja E. Levin. "Using microservices for non-intrusive customization of multi-tenant SaaS". Teoksessa: *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, s. 905–915.

[A20] P. Nunes, V. Furuholt, N. Burns ja P. Aursand. "Vendor-independent workflow architecture to integrate domain applications and accelerate R&D to production". Teoksessa: *1st EAGE Digitalization Conference and Exhibition*. 2020.

[A21] C. Pasomsup ja Y. Limpiyakorn. "HT-RBAC: A Design of Role-based Access Control Model for Microservice Security Manager". Teoksessa: *2021 International Conference on Big Data Engineering and Education (BDEE)*. Elokuu 2021, s. 177–181.

[A22] A. Pukas, A. Melnyk, I. Voytyuk, A. Yushko, M. Romanyuk ja L. Honchar. "Transactional Business Application Based on Microservice Architecture". Teoksessa: *2021 11th International Conference on Advanced Computer Information Technologies (ACIT)*. 2021, s. 564–567.

[A23] S. Rajapaksa, A. Wickramarachchi, V. Mallawaarachchi, W. Rasanjana, I. Perera ja D. Meedeniya. "A Scalable Bioinformatics Analysis Platform based on Microservices Architecture". Teoksessa: *2019 International Research Conference on Smart Computing and Systems Engineering (SCSE)*. Maaliskuu 2019, s. 70–77.

[A24] N. Santos, H. Rodrigues, J. Pereira, F. Morais, R. Abreu, N. Fernandes, D. Martins ja R. J. Machado. "UH4SP: A Software Platform For Integrated Management Of Connected Smart Plants". Teoksessa: *2018 International Conference on Intelligent Systems (IS)*. Syyskuu 2018, s. 541–548.

[A25] A. Simioni ja T. Vardanega. "In Pursuit of Architectural Agility: Experimenting with Microservices". Teoksessa: *2018 IEEE International Conference on Services Computing (SCC)*. 2018, s. 113–120.

[A26] M. Song, C. Zhang ja E. Haihong. "An Auto Scaling System for API Gateway Based on Kubernetes". Teoksessa: *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*. Marraskuu 2018, s. 109–112.

[A27] P. Wizenty, J. Sorgalla, F. Rademacher ja S. Sachweh. "MAGMA: Build Management-Based Generation of Microservice Infrastructures". Teoksessa: *Proceedings of the*

11th European Conference on Software Architecture: Companion Proceedings. ECSA '17. Association for Computing Machinery, 2017, s. 61–65.

[A28] R. Xu, W. Jin ja D. Kim. "Microservice security agent based on API gateway in edge computing". *Sensors (Switzerland)* 19.22 (2019).

[A29] J. Zhao, S. Jing ja L. Jiang. "Management of API Gateway Based on Micro-service Architecture". *Journal of Physics: Conference Series* 1087.3 (syyskuu 2018), s. 032032.

[A30] X. Zuo, Y. Su, Q. Wang ja Y. Xie. "An API gateway design strategy optimized for persistence and coupling". *Advances in Engineering Software* 148 (2020).

Viittauksin löydetyt lähteet

[B1] A. Akbulut ja H. G. Perros. "Performance Analysis of Microservice Design Patterns". *IEEE Internet Computing* 23.6 (marraskuu 2019), s. 19–27.

[B2] A. Gamez-Diaz, P. Fernandez ja A. Ruiz-Cortes. "Automating SLA-Driven API Development with SLA4OAI". Teoksessa: *Service-Oriented Computing*. Toim. S. Yangui, I. Bouassida Rodriguez, K. Drira ja Z. Tari. Cham: Springer International Publishing, 2019, s. 20–35.

[B3] A. Gamez-Diaz, P. Fernandez, A. Ruiz-Cortés, P. J. Molina, N. Kolekar, P. Bhogill, M. Mohaan ja F. Méndez. "The Role of Limitations and SLAs in the API Industry". Teoksessa: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2019. Association for Computing Machinery, 2019, s. 1006–1014.

[B4] W. Jin, R. Xu, T. You, Y.-G. Hong ja D. Kim. "Secure Edge Computing Management Based on Independent Microservices Providers for Gateway-Centric IoT Networks". *IEEE Access* 8 (tammikuu 2020), s. 187975–187990.

[B5] L. Liu, X. He, Z. Tu ja Z. Wang. "MV4MS: A Spring Cloud based Framework for the Co-Deployment of Multi-Version Microservices". Teoksessa: *2020 IEEE International Conference on Services Computing (SCC)*. Marraskuu 2020, s. 194–201.

[B6] N. Mohammadi ja A. Rasoolzadegan. "A Pattern-aware Design and Implementation Guideline for Microservice-based Systems". Teoksessa: *2022 27th International Computer Conference, Computer Society of Iran (CSICC)*. 2022, s. 1–6.

[B7] R. S. de O. Júnior, R. C. A. da Silva, M. S. Santos, D. W. Albuquerque, H. O. Almeida ja D. F. S. Santos. "An Extensible and Secure Architecture based on Microservices". Teoksessa: *2022 IEEE International Conference on Consumer Electronics (ICCE)*. 2022, s. 01–02.

[B8] J. A. Suthendra ja M. A. I. Pakereng. "Implementation of Microservices Architecture on E-Commerce Web Service". *ComTech Computer Mathematics and Engineering Applications* 11.2 (2020), s. 89–95.

Liite B Teemahaastattelun runko

Haastateltavan ja organisaation taustaa

- Haastateltavan titteli ja työskentelyvuodet yrityksessä
- Yrityksen toimiala, minkälaista tuotetta yritys tekee
- Yrityksen koko, monta sovelluskehitystä tekevää tiimiä organisaatiossa on
- Minkälainen infrastruktuuri tai arkkitehtuuri organisaatiossa on käytössä? Minkälaisia rajapintoja on käytössä?
- Kuinka paljon alustoja tai tuotteita loppukäyttäjille on?
- Onko infrastruktuuriin tehty muutoksia? Minkälaisia ja miksi?
- Integraatiot, niiden määrä ja ketkä niitä käyttää
- Mikä on rajapinta, jota täytyy jotenkin hallita tai sellainen johon liittyy jotain käytänteitä?

Tunnistautuminen ja pääsynhallinta

- Rajapintoihin tunnistautuminen
- Integraatioihin tunnistautuminen
- Pääsynhallinnan toteutus
- Kuinka pääsynhallinan oikeuksia muokataan
- Kuinka uusi integraatio saa pääsyn rajapintoihin
- Kuinka rajapintojen käyttäjiä hallitaan, sisäisiä ja ulkoisia
- Mitä työkaluja käytössä näiden ongelmien ratkomiseen

Lokitoetojen keräys ja tilastointi

- Rajapintojen käytön seuranta
- Mitä tietoja lokitetaan

- Ketkä näitä tietoja katselee
- Miten kerätyt tiedot kerätään

Rajapintojen muutoksien koordinointi ja versiointi

- Onko rajapinta versioitu?
- Onko useita versioita saatavilla
- Kuinka taaksepäin yhteensopivuuden rikkovat muutokset hallitaan
- Integraatioiden versiointi
- Kuinka muutoksista viestitään
- Kuinka kehittäjä toimii jos täytyy luoda uusi rajapinta? Minkälainen prosessi?

Rajapintojen dokumentointi

- Onko rajapinnat dokumentoitu?
- Missä dokumentaatio on
- Ketkä dokumentaatiota lukee
- Dokumentaatio integraatioille tai ulkopuolisille käyttäjille
- Kuinka dokumentaatio pidetään ajan tasalla

Kehittäjien työskentely ja toimintatavat rajapintoihin liittyen

- Kuinka sovelluskehittäjät kokevat tavan työskennellä rajapintojen kanssa
- Toimiiko nykyiset prosessit työskentelytavat?
- Onko jotain selkeää mitä täytyy vielä kehittää tai ratkoa
- Muuta, mitä?