Tampereen yliopisto

Roope Karvinen

AUTENTIKOINNIN JA AUKTORISOINNIN TOTEUTTAMINEN MIKROPALVELU-ARKKITEHTUURISSA

Diplomityö
Informaatioteknologian ja viestinnän tiedekunta
Outi Sievi-Korte
David Hästbacka
Lokakuu 2021

TIIVISTELMÄ

Roope Karvinen: Autentikoinnin ja auktorisoinnin toteuttaminen mikropalveluarkkitehtuurissa
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-ohjelma
Lokakuu 2021

Mikropalveluarkkitehtuurilla toteutettu järjestelmä koostuu useista itsenäisistä ja pienistä palveluista, eli mikropalveluista. Mikropalvelut kommunikoivat keskenään rajapintojen kautta. Ohjelmistoissa käyttäjä yleensä autentikoidaan ja auktorisoidaan ennen kuin käyttäjä voi käyttää sovellusta. Autentikoinnilla selvitetään käyttäjän identiteetti ja auktorisoinnilla määritetään käyttäjän käyttöoikeudet järjestelmässä. Perinteiseen monoliittiseen ohjelmaan verrattuna mikropalveluiden autentikoinnin ja auktorisoinnin toteuttaminen on haasteellisempaa. Autentikointia ja auktorisointia voidaan toteuttaa mikropalvelujärjestelmän eri osa-alueilla eli reunatasolla, palvelutasolla ja palveluiden keskinäisessä kommunikaatiossa.

UNA Ydin projektissa kehitetään integraatio- ja tiedonhallintaratkaisua sosiaali- ja terveydenhuollon käyttöön. Ydin on toteutettu käyttäen mikropalveluarkkitehtuuria, ja tässä työssä tutkitaan Ytimen autentikointi- sekä auktorisointitoteutuksia vertaillen niitä kirjallisuudessa esiintyviin muihin ratkaisuihin. Lisäksi työssä pyritään selvittämään yleisesti mikropalveluarkkitehtuuriin sopivaa autentikoinnin ja auktorisoinnin toteutusmallia.

Työn tuloksena UNA Ytimen auktorisointi- ja autentikointitoteutukseen löydettiin muutamia kehitysideoita, joista tärkein liittyy mikropalvelutasolla tapahtuvaan auktorisointiin. Tätä ideaa testattiin käytännössä toteuttamalla demototeutus hajautetusta auktorisoinnista. Lisäksi työssä koostettiin yleinen esimerkkiratkaisu mikropalveluiden autentikoinnin ja auktorisoinnin toteuttamiseksi.

Avainsanat: Mikropalveluarkkitehtuuri, mikropalvelut, autentikointi, auktorisointi, tietoturva, turvallisuus, toteutus

ABSTRACT

Roope Karvinen: Implementing authentication and authorization in a microservice architecture
Master's thesis
Tampere University
Master's Programme in Information Technology
October 2021

System built with microservice architecture consists of several small and independent services called microservices. Microservices communicate with each other via APIs. Usually when using a software, the user is authenticated and authorized before access is granted to a system. Authentication is a process of figuring out user's identity and authorization is a process of determining the user's access rights to the system. Implementing authentication and authorization in microservices is somewhat more challenging compared to implementing them in monolithic program. Microservice's authentication and authorization can be implemented in different parts of the system. These are called edge-level authorization, service-level authorization, and service-to-service authentication.

In UNA Core project, a data management and integration solution is being developed for usage of Finnish social care and healthcare. The Core is implemented using microservice architecture. The authentication and authorization implementations used in the Core are studied in this thesis by using literature as a reference. In addition, the secondary aim of the work is to find out a generally suitable implementation model of authentication and authorization for microservice architecture.

As a result of this thesis, a few development ideas were found for the UNA Core's implementation of authorization and authentication. The most relevant idea was related to improving the Core's service-level authorization. The idea was tested in practice by implementing a demo implementation of decentralized authorization. As a secondary result, a general example solution was compiled from literature to implement authentication and authorization in the microservice architecture.

Keywords: Microservice architecture, microservices, authentication, authorization, information security, security, implementation

ALKUSANAT

Työ toteutettiin Atostek Oy:n rahoittamana perustuen UNA Ydin -projektiin. Haluan kiittää UNA Oy:tä mahdollisuudesta kirjoittaa työ valitusta aiheesta. Atostekilta haluan kiittää Seppo Niemistä hyvistä vinkeistä ja tuesta työn aikana, sekä Devinder Singhiä avusta työn ideoinnissa. Kiitoksen ansaitsee myös työn ohjaajaa Outi Sievi-Korte hyödyllisestä palautteesta työn eri vaiheissa.

Tampereella, 26.10.2021

Roope Karvinen

SISÄLLYSLUETTELO

1. JOHDANTO 1
    1.1 Yleistä 1
    1.2 Tapaus UNA 2
2. MIKROPALVELUT 3
    2.1 Mikropalveluarkkitehtuuri 3
    2.2 Mikropalveluiden suunnitteluperiaatteet 4
    2.3 Hyödyt 6
    2.4 Haasteet 7
    2.5 Rajapinnat ja kommunikointi mikropalveluiden välillä 8
        2.5.1 Viestintäformaatit 9
        2.5.2 Synkroninen kommunikaatio 10
        2.5.3 Asynkroninen kommunikaatio 11
        2.5.4 Rajapintojen ja palveluiden löytäminen 12
    2.6 Ympäristöt 14
3. AUTENTIKOINTI JA AUKTORISOINTI 16
    3.1 Autentikointi 16
    3.2 Kertakirjautuminen 16
        3.2.1 OAuth 18
        3.2.2 OpenID Connect 19
        3.2.3 SAML 20
        3.2.4 JWT 22
    3.3 Vahva tunnistautuminen 24
    3.4 Käyttäjän auktorisointi ja pääsynhallinta 25
        3.4.1 DAC, MAC ja ACL 25
        3.4.2 Roolipohjainen pääsynhallinta 26
        3.4.3 Attribuuttipohjainen pääsynhallinta 27
4. MIKROPALVELUIDEN AUKTORISOINTI- JA AUTENTIKOINTIRATKAISUT 29
    4.1 Aiemmat tutkimukset aiheesta 29
    4.2 Palveluiden välinen autentikointi 31
    4.3 Reunatason auktorisointi 34
    4.4 Palvelutason auktorisointi 36
    4.5 Ulkoisen ja sisäisen tunnisteen erottelu 40
    4.6 Ratkaisujen arviointi 42
5. AUTENTIKOINTI JA AUKTORISOINTI UNA YTIMESSÄ 45
    5.1 Yleiskuvaus 45
    5.2 Arkkitehtuuri 46
        5.2.1 eRA 47
        5.2.2 Käytönhallinta 47
        5.2.3 Käytönvalvonta 48

5.2.4 KVKH-käyttöliittymä 48
5.3 Autentikointi- ja auktorisointiratkaisut 49
5.3.1 Pääsynhallintamalli 49
5.3.2 Käyttäjän autentikointi 49
5.3.3 Mikropalveluiden välinen autentikointi 53
5.3.4 Reunatason ratkaisut 53
5.3.5 Palvelutason ratkaisut 54
6. VERTAILU JA ARVIOINTI 55
6.1 Ytimen auktorisointi- ja autentikointitoteutusten arviointia 55
6.2 Esimerkkiratkaisu mikropalveluiden autentikointiin ja auktorisointiin 57
6.3 Kehitysideat 60
6.4 Demo hajautetusta auktorisoinnista 61
7. YHTEENVETO 64
LÄHTEET 66

KUVALUETTELO

Kuva 1. *Monoliittinen arkkitehtuuri ja mikropalveluarkkitehtuuri [11, s. 3][12, s. 32]..... 5*
Kuva 2. *Normaali kirjautuminen ja kertakirjautuminen havainnollistettuna..... 17*
Kuva 3. *OAuth 2.0 valtuutuskoodilla valtuuttamisen työnkulku [44, s. 23][45, luku 4] 18*
Kuva 4. *OpenId Connect valtuutuskoodilla valtuuttamisen työnkulku. [46, luku 6] 20*
Kuva 5. *SAML kertakirjautuminen palveluntuottajan aloittamana. [48, luku 5.1.2]..... 21*
Kuva 6. *SAML kertakirjautuminen identiteettipalvelun aloittamana. [48, luku 5.1.4]..... 22*
Kuva 7. *Käyttäjien, roolien ja luvitusten suhteet RBAC-mallissa. [55, s. 64]..... 27*
Kuva 8. *ABAC-mallin toimintaa kuvaava prosessi. [56, s. 14]..... 28*
Kuva 9. *Mikropalveluiden auktorisoinnin ja autentikoinnin eri osa-alueet. [59]..... 30*
Kuva 10. *Mikropalveluiden kommunikaatio palveluverkon välityksellä. [64]..... 34*
Kuva 11. *API-yhdyskäytävää ja JWT-tunnistetta hyödyntävä auktorisointiratkaisu. [68] 35*
Kuva 12. *API-yhdyskäytävää hyödyntävä autentikointi- ja auktorisointiratkaisu. [69] 36*
Kuva 13. *Keskitetty auktorisointi monoliitti- ja mikropalvelujärjestelmässä. [70]..... 37*
Kuva 14. *JWT-pohjainen auktorisointi mikropalveluissa. [70][71] 38*
Kuva 15. *Esimerkki auktorisoinnin toteuttamisesta sivuvaunuissa. [66][65]..... 39*
Kuva 16. *Ulkoiseen ja sisäiseen tunnisteeseen perustuva mikropalveluiden pääsynhallinta. [59][72][73] 41*
Kuva 17. *Hahmotelma Ytimen arkkitehtuurista. [5][76]..... 46*
Kuva 18. *Kaavio Ytimen kirjautumisesta eRAn kautta. [76]..... 50*
Kuva 19. *Kaavio Ytimen federoidusta kirjautumisesta. [76]..... 52*
Kuva 20. *Esimerkkiratkaisu autentikoinnin ja auktorisoinnin toteuttamiseen..... 57*

LYHENTEET JA MERKINNÄT

ABAC	engl. Attribute-Based Access Control, attribuuttipohjainen pääsynhallinta
ACL	engl. Access Control List, käyttäjäoikeuslista, pääsylista
AD	Active Directory, Windowsin hakemistopalvelu
AES	engl. Advanced Encryption Standard, lohkosalausmenetelmä
AMQP	Advanced Message Queuing Protocol
API	engl. Application Programming Interface, ohjelmointirajapinta
API Gateway	API-yhdyskäytävä, rajapintayhdyskäytävä
AWS	Amazon Web Services
Backend	taustajärjestelmä
CD	engl. Continuous Delivery, jatkuva julkaiseminen
CI	engl. Continuous Integration, jatkuva integraatio
DAC	engl. Discretionary Access Control, harkinnanvarainen pääsynhallinta
DNS	engl. Domain Name System, nimipalvelujärjestelmä
HTTP	Hypertext Transfer Protocol
IdP	engl. Identity Provider, identiteetin tarjoaja
IP	engl. Internet Protocol, verkkokerroksen protokolla
JSON	JavaScript Object Notation
JWT	JSON Web Token
KH	käytönhallinta
KV	käytönvalvonta
LDAP	Ligthweight Directory Access Protocol
Load Balancer	kuorman tasaaja
MAC	engl. Mandatory Access Control, pakollinen pääsynhallinta
MFA	engl. Multifactor Authentication, monivaiheinen tunnistautuminen
NGAC	Next Generation Access Control
NtK	engl. Need to Know, tiedon tarpeen periaate
OAuth	Open Authorization
OIDC	OpenID Connect
OTP	engl. One-time password, kertakäyttöinen salasana
PLP	engl. Least Privilege Principle, vähimpien oikeuksien periaate
RBAC	engl. Role-Based Access Control, roolipohjainen pääsynhallinta
REST	engl. Representational State Transfer
RPC	engl. Remote Procedure Call, etäproseduurikutsu
RSA	Rivest-Shamir-Adleman, julkisen avaimen salausalgoritmi
SAML	Security Assertion Markup Language
SD	engl. Separation of Duties, tehtävien eriyttämisen periaate
SSL	engl. Secure Sockets Layers, salausprotokolla
SP	engl. Service Provider, palvelun tarjoaja
SRP	engl. Single Responsibility Principle, yhden vastuun periaate
SSO	engl. Single Sign On, kertakirjautuminen
TLS	engl. Transport Layer Security, salausprotokolla
XACML	Extensible Access Control Markup Language
XML	eXtensible Markup Language
2FA	engl. Two-factor Authentication, kaksivaiheinen tunnistautuminen.

1. JOHDANTO

1.1 Yleistä

Tietoteknisten järjestelmien turvallisuus on puhuttanut viime vuosina monien tietovuotojen ja järjestelmien haavoittuvuuksien esiintulon myötä. Yksi esimerkki viimeaikaisista tietovuodoista on Vastaamon tapaus, jossa psykoterapiakeskuksen potilaiden arkaluontoisia tietoja julkaistiin internetissä [1]. Turvallisuuden tarve korostuu erityisesti kriittisissä terveydenhuollon potilasjärjestelmissä. Onkin tärkeää, että uudet järjestelmät eivät sisältäisi haavoittuvuuksia, eikä kukaan pääsisi tekemään minkäänlaista tietomurtoa järjestelmiin.

Tietoturva-asioiden tulisi olla myös pilviympäristöissä kunnossa, sillä yhä useammat uudet sekä vanhat järjestelmät siirtyvät pilviympäristöihin, joissa järjestelmät skaalautuvat paremmin ja ovat helpommin saatavilla ympäri maailmaa. Aiemmin ohjelmistot ovat toteutettu yhtenä kokonaisuutena eli monoliittisena ohjelmistona. Monoliittiset järjestelmät eivät pysty hyödyntämään kaikkia pilviympäristön hyötyjä samalla tavoin kuten hajautettua arkkitehtuuria noudattavat ohjelmistot pystyvät. Mikropalveluarkkitehtuuri on hajautettu järjestelmä ja siinä järjestelmä koostuu useista moduuleista, jotka yhdessä muodostavat kokonaisen järjestelmän. Mikropalveluarkkitehtuuri mahdollistaa muun muassa sovelluksen skaalautumisen suurelle määrälle käyttäjiä, nopean sovelluksen kehittämisen ja paremman saatavuuden. Esimerkiksi viihdepalvelu Netflix on toteuttanut palvelunsa käyttäen mikropalveluarkkitehtuuria. [2, luku 1]

Myös pilvipalvelut tulisi suunnitella ja toteuttaa turvallisiksi, koska ne ovat helpommin saatavilla myös hyökkääjille. Mahdollista hyökkäyspinta-alaa on enemmän kuin normaalissa monoliittisessa ohjelmistossa [3, s. 13]. Yksi tärkeimpiä osa-alueita järjestelmien turvallisuudessa on käyttäjien autentikointi ja auktorisointi, eli käyttäjien todentaminen ja valtuuttaminen [4]. Tässä työssä tutkitaan sosiaali- ja terveydenhuollon UNA-Ydin projektissa toteutetun järjestelmän autentikointiratkaisua ja sen soveltuvuutta projektissa käytettyyn mikropalveluarkkitehtuuriin. Koska ei ole olemassa yleistä standardia mikropalveluarkkitehtuurin autentikointiratkaisulle, tutkitaan tässä työssä lisäksi aiempien tutkimusten ja kirjallisuuden perusteella, millainen ratkaisu olisi sopivin mikropalvelujärjestelmien autentikointiin. Tutkimusmenetelmäksi valittiin tässä työssä tapaustutkimus, sillä se sopii hyvin tällaisen rajatun aiheen tutkimiseen ja aiheeseen liittyvän ymmärryksen

lisäämiseen. Tämän tutkimuksen tulokset auttavat mahdollisessa jatkokehityksessä kyseisessä ohjelmistoprojektissa, sekä antavat lukijoille ymmärrystä mikropalvelujärjestelmien kirjautumisratkaisujen suunnitteluun.

Luvussa 2 käydään läpi mikropalveluarkkitehtuuria. Käyttäjän autentikointi- ja auktorisointimenetelmiä, kuten kertakirjautumista käsitellään luvussa 3. Aiempia tutkimuksia mikropalveluiden autentikointiratkaisuista sekä niistä esiin nousseita yleisimpiä ratkaisuja käydään läpi luvussa 4. UNA-projektissa toteutettu ratkaisu ja sen arkkitehtuuria esitellään luvussa 5. Luvussa 6 vertaillaan Ytimessä toteutettua järjestelmää muihin luvussa 4 esitettyihin vaihtoehtoihin. Lopuksi viimeisessä luvussa 7 esitetään johtopäätökset ja yhteenveto työstä.

1.2 Tapaus UNA

UNA-hankkeen tarkoituksena on uudistaa sosiaali- ja terveydenhuollon toimintaa tukevaa tietojärjestelmäkokonaisuutta. Sosiaali- ja terveydenhuollossa eli sotessa asiakkaiden tiedot ovat olleet kunkin sote-alueen järjestelmissä, ja tiedon jakaminen alueelta toiselle ei ole ollut riittävän sujuvaa [5]. Sote-alueella tarkoitetaan aluetta Suomessa, joka järjestää sosiaali- ja terveyspalvelut alueen asukkaille. Tulevan sote-uudistuksen jälkeen näitä alueita kutsutaan hyvinvointialueiksi [6].

Jotta sote-palveluiden asiakkaiden ohjaus ja palvelu olisi sujuvampaa, pitäisi asiakkaiden tiedot olla saatavissa yhdestä paikasta helposti. UNA Ydin projektissa tätä ongelmaa ratkaistaan kehittämällä tiedonhallinta- ja integraatioratkaisua, joka kokoaa yksilö- ja väestötason asiakkuustiedot yhteen. UNA Ytimen on tarkoitus olla järjestelmä, joka pystyy hakemaan, yhdistelemään ja suodattamaan tietoja useista lähteistä, ja tarjota nämä tiedot yhdestä palvelusta tietoturvallisesti. Tämä helpottaa sote-palvelujärjestäjien ja -tuottajien toimintaa. Tärkeimmät käyttötarpeet UNA Ytimelle ovat ammattihenkilön tukeminen asiakaskontakteissa, kansalaisen tukeminen itseään koskevien sote-asioiden hoidossa ja johtamisen tukeminen soten tuotannossa syntyvillä asiakastiedoilla. [5]

Tässä työssä tapaustutkimuksella pyritään selvittämään, ovatko Ytimeen aiemmin toteutetut autentikointi- ja auktorisointiratkaisut sopivia käyttötarkoitukseensa ja voisiko niitä kehittää. Ytimen ratkaisuja käytetään vertailukohteena kirjallisuudesta löytyneisiin ratkaisuihin. Luvussa 6 pohditaan, miten kirjallisuuden ratkaisut sopisivat Ytimeen. Pohdinnan avulla pyritään löytämään mahdollisia parannuskohteita ja voidaan luoda työhypoteeseja eli arvauksia siitä, kuinka Ydin hyötyisi erilaisista autentikointi- ja auktorisointitoteutuksista. Lopuksi työssä testataan demototeutuksen avulla yhtä kehitysideaa käytännössä.

2. MIKROPALVELUT

2.1 Mikropalveluarkkitehtuuri

Mikropalveluarkkitehtuuri voidaan luokitella erikoistuneeksi muodoksi palvelukeskeisestä arkkitehtuuria (engl. Service Oriented Architecture, SOA). Palvelukeskeisen arkkitehtuurin idea on jakaa ohjelma palveluihin, ja palvelut yhdessä tuottavat halutunlaisia toiminnallisuuksia [2, luku 1]. Vuralin ym. mukaan mikropalvelun ensimmäinen määritelmä on ollut vuonna 2010 ”kevyt palveluluokituskehys REST-arkkitehtuuriselle tyylille” [7, s. 204]. Representational State Transfer (REST) on hajautetuissa järjestelmissä käytetty arkkitehtuuri rajapintojen toteuttamiselle [8].

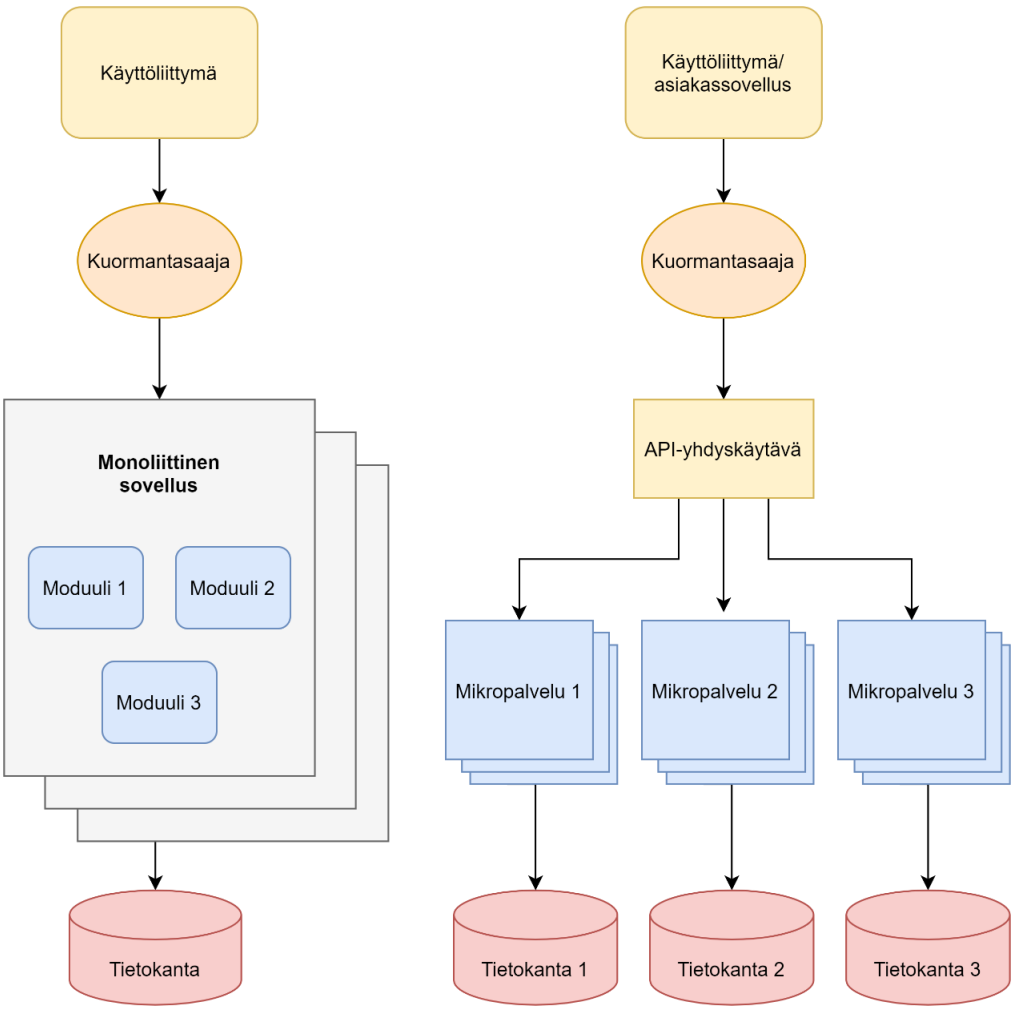
Mikropalveluarkkitehtuurissa järjestelmä kokonaisuutena muodostuu useista pienistä osista, eli mikropalveluista [3, s. 6]. Kukin yksittäinen mikropalvelu voidaan kehittää, skaalata ja ottaa käyttöön autonomisesti riippumatta muista mikropalveluista, mutta kokonaisuus muodostuu useista yhteistyötä tekevistä mikropalveluista [9]. Fowlerin ja Lewisin mukaan mikropalveluille ominaisia tunnusmerkkejä ovat [10]:

- Komponentteihin jako palveluiden avulla. Ohjelma koostuu itsenäisesti toimivista komponenteista eli mikropalveluista.
- Liiketoimintamahdollisuuksien ympärille järjestettyjä tiimejä. Tiimit muodostuvat palveluiden ja liiketoimintamahdollisuuksien ympärille siten, että kaikki tarpeelliset henkilöt ovat käytettävissä samassa tiimissä ja työstävät yhdessä samoja mikropalveluita.
- Ohjelmistot ovat tuotteita, eivät hankkeita. Tiimien tulisi omistaa ohjelmistotuotteensa koko sen elinkaaren ajan sen sijaan, että luovuttaisivat sen eteenpäin projektin määräajan päätyttyä. Mikropalvelun kehitys ei pääty projektin lopussa, vaan sitä paranneltaisiin jatkuvasti sen elinkaaren ajan.
- Älykkäät rajapinnat ja tyhmät kanavat. Sen sijaan, että logiikkaa tehtäisiin kommunikaatiorakenteiden ympärille, logiikka ja tiedon suodattaminen tapahtuvat mikropalveluissa itsessään. Liikenteen palveluiden välillä tulisi muodostua yksinkertaisista pyynnöistä ja vastauksista.
- Hajautettu hallinto. Tiimit saavat valita itsenäisesti haluamansa toteutustavat ja työkalut työnsä tekemiseen.

- Hajautettu tiedonhallinta. Kukin tiimi ja mikropalvelu voi toteuttaa omalla valitsemallaan tavallaan palvelunsa tarvitseman tiedonhallinnan. Esimerkiksi palveluiden ei ole pakko käyttää samaa tietokantaa, vaan kullakin palvelulla voi olla omat tietokannat.
- Infrastruktuuriautomaatio. Testaus- ja julkaisuprosessien automatisointi on tärkeää mikropalveluarkkitehtuurien hallinnassa. Kyseisten töiden tekeminen manuaalisesti voi olla todella työlästä.
- Suunnittelu epäonnistumista varten. Kunkin mikropalvelun tilaa ja suoriutumista tulisi monitoroida. Mikäli mahdollista, järjestelmän tulisi pyrkiä automaattisesti ratkomaan ja korjaamaan virhetilanteita.
- Evoluutiollinen suunnittelu. Järjestelmä tulisi olla suunniteltu siten, että muutoksia on helppo hallita. Joitakin palveluita saatetaan ottaa pois käytöstä ja uusia palveluita saatetaan lisätä järjestelmään.

2.2 Mikropalveluiden suunnitteluperiaatteet

Mikropalvelut ovat autonomisia, pieniä ja yhteen tiettyyn asiaan keskittyneitä palveluita, jotka toimivat yhdessä ja muodostavat siten toimivan kokonaisuuden [3, s. 6]. Monoliittisissa järjestelmissä moduulit saattavat koostua sisäisistä moduuleista, joilla kullakin on oma tehtävänsä. Tätä periaatetta kutsutaan yhden vastuun periaatteeksi (engl. Single Responsibility Principle). Mikropalveluarkkitehtuurissa käytetään samaa periaatetta, mutta sisäisten luokkien ja moduulien sijaan jokainen mikropalvelu on rajattu tekemään vain yhtä asiaa. Vastuu mikropalveluiden välillä tulisi jakaa siten, että kukin palvelu tekisi vain yhden asian hyvin. Kuvasta 1 näkee eron mikropalveluiden ja monoliittisen arkkitehtuurin välillä. [2, luku 1]



Kuva 1. *Monoliittinen arkkitehtuuri ja mikropalveluarkkitehtuuri [11, s. 3][12, s. 32]*

Mikropalveluiden suunnitteluperiaatteita ovat muun muassa autonomisuus, löyhät kytkökset, uudelleenkäytettävyys, koostettavuus ja virheensietokyky [12, s. 4]. Mikropalvelun autonominen luonne tarkoittaa sitä, että se voidaan kehittää, testata, ylläpitää ja ottaa käyttöön irrallaan muista mikropalveluista [9]. Lisäksi kullekin mikropalvelulle voidaan tehdä teknologiavalinnat riippumatta muista mikropalveluista. Jos järjestelmässä on esimerkiksi 5 mikropalvelua, jokainen näistä voitaisiin toteuttaa eri ohjelmointikielillä. Ohjelmistoprojekteissa tämä tarkoittaa myös sitä, että voidaan palkata useampi tiimi ja antaa kullekin tiimille omat mikropalvelut toteutettaviksi [13, luku 12]. Se on mahdollista, koska mikropalveluiden välinen keskustelu tapahtuu verkon yli käyttämällä yhteisesti sovittua rajapintateknologiaa. Yksi yleisesti käytetty rajapintateknologia mikropalveluissa on http-pohjaiset REST-rajapinnat. Löyhien kytköksien periaatteen mukaan muutoksien tekeminen yhdessä mikropalvelussa ei saisi vaikuttaa toiseen mikropalveluun. Kytkökset tulisi täten toteuttaa mahdollisimman löyhästi, jotta mikropalvelut eivät olisi suoraan riippuvaisia toisistaan. Uudelleenkäytettävyydellä tarkoitetaan mahdollisuutta hyödyntää kerran toteutettua mikropalvelua uudestaan muualla. Koostettavuus periaatteen mukaan useita

mikropalveluita käyttäessä yhdessä voidaan koostaa erilaisia toiminnallisuuksia järjestelmässä. Virheensietokyky voidaan ilmaista myös ”design for failure” nimisenä periaatteena, jonka mukaan mikropalvelut tulee suunnitella selviytymään virhetilanteista mahdollisimman hyvin. [2, luvut 1 ja 3][3]

2.3 Hyödyt

Mikropalveluiden hyötyjä ovat skaalautuvuus, vaihdettavuus, järjestelmän joustavuus, virheensietokyky ja vapaat teknologiavalinnat. Lisäksi mikropalveluiden nopea kehittäminen ja nopea markkinoille pääsy tuottavat rahallista hyötyä. [13, luku 4]

Skaalautuvuus on yksi merkittävistä mikropalveluiden hyödyistä. Itsenäisiä mikropalveluita voidaan suorittaa rinnakkain ja jakaa kuormaa niiden välillä, jotta järjestelmä saadaan isommalle käyttäjämäärälle käytettäväksi yhtäaikaisesti. Mikropalvelut mahdollistavat helpon ja nopean horisontaalisen skaalautuvuuden, jossa lisätään uusia ”koneita” eli tässä tapauksessa uusia mikropalveluita kokonaisresursseihin [14, s. 228]. Vertikaalisessa skaalautuvuudessa lisätään suorituskykyä, muistia ja tallennustilakapasiteettia jo olemassa oleviin ”koneisiin”. Monoliittisessa järjestelmässä tällainen horisontaalinen skaalautuvuus ei ole mahdollista, vaan pitäisi kopioida koko järjestelmä horisontaalisen skaalautumisen aikaansaamiseksi. [13, luku 4][14]

Mikropalveluiden virheensietokyky on hyvä, sillä mikropalveluarkkitehtuurissa noudatetaan ”design for failure”-periaatetta. Monoliittisen järjestelmän päätyessä virhetilaan yleensä koko systeemi menee jumiin ja sen estämiseksi voidaan ajaa useata sovellusta rinnakkain. Mikropalvelussa järjestelmä koostuu useista yksittäisistä palasista, joten virhetilanteessa voidaan vain kytkeä jumiutuneen mikropalvelun tuottama toiminnallisuus pois, mutta muu järjestelmä toimii edelleen [15]. Näin vältytään siltä, ettei koko järjestelmä kaatuisi samanaikaisesti. Lisäksi jos yksittäinen mikropalvelu menisi vikatilaan tai kuormaa olisi liikaa yhdelle, voidaan tarvittaessa ottaa käyttöön useampi instanssi samaa mikropalvelua yhtäaikaisesti. Täten varalla oleva mikropalvelu voisi ottaa ylimääräisen kuorman tai korvata ensimmäisen mikropalvelun, jos se kaatuisi kokonaan. Tämän kaltainen skaalautuvuus ei ole mahdollista monoliittisessa järjestelmässä, vaan skaalattaessa täytyisi koko järjestelmä kopioida ja suorittaa sitä useassa eri laitteessa yhtäaikaisesti. [2, luku 1]

Mikropalveluiden modulaarisuus mahdollistaa nopeamman markkinoille viennin verrattuna monoliittisiin ohjelmistoihin. Koska useampi tiimi voi työstää samaan aikaan yksittäisiä mikropalveluita ja käyttää suosimaansa teknologiaa toteutuksessa, saadaan jär-

jestelmä kokonaisuudessaan nopeammin valmiiksi [14]. Toinen merkittävä etu liittyy jatkuvan integraation ja julkaisun putkiin (engl. CI, Continuous Integration ja CD, Continuous Delivery). Kullekin mikropalvelulle voidaan luoda oma jatkuvan julkaisun ja testauksen putki, jossa mikropalvelu testataan ja julkaistaan automaattisesti, kun mikropalveluun tehdään muutoksia. Näin uudet muutokset saadaan jatkuvalla syötöllä tuotantoympäristöön käyttöön ja ongelmat voidaan löytää sekä korjata nopeammin [15]. [13, luku 4][14]

Mikropalveluiden modulaarisuus mahdollistaa myös hyvän vaihdettavuuden. Koska mikropalvelut ovat irrallisia muista mikropalveluista, voidaan kukin mikropalvelu korvata toisella samaa toiminnallisuutta tarjoavalla mikropalvelulla. Esimerkiksi jos yksittäisen mikropalvelun toteutus ei ole riittävän suorituskykyinen, voitaisiin koodata sama mikropalvelu täysin alusta alkaen, kunhan se sisältää samat rajapinnat kuin alkuperäinen mikropalvelu. Tämän jälkeen vanha mikropalvelu voitaisiin korvata uudella toteutuksella. [13, luku 4]

Mikropalveluiden tarjoamat rajapinnat mahdollistavat innovaatiota ja vähentävät ylimääräisiä palavereita. Muut tiimit ja kehittäjät voivat rakentaa uutta toiminnallisuutta mikropalveluiden rajapintojen päälle [15]. Rajapinnat ovat yleensä selkeitä, sillä kukin mikropalvelu tuottaa tietynlaista rajattua toiminnallisuutta, ja työkalut kuten Swagger helpottavat rajapintojen dokumentaation ylläpitämisessä. Swagger tuottaa automaattisesti generoitua rajapintadokumentaatiota [2, luku 11]. Aina ajan tasalla olevan dokumentaation avulla voidaan vähentää kehittäjien välisiä integraatiopalavereita. Lisäksi integraatiot muihin järjestelmiin onnistuvat mikropalveluissa helposti, sillä integraatiota varten voidaan luoda oma integraation toteutuksesta vastaava mikropalvelu ja rajapinnat. Muut mikropalvelut voivat toteutuksen jälkeen hyödyntää integraatiomikropalvelun rajapintoja [13, luku 4].

2.4 Haasteet

Mikropalveluarkkitehtuurissa on uudenlaisia haasteita, joita ei monoliittisissa järjestelmissä välttämättä ole. Verkon kompleksisuus on yksi suurimmista haasteista mikropalveluissa, joka ei samalla tavalla ilmene monoliittisessa ohjelmassa. Verkon kompleksisuudella tarkoitetaan mikropalveluiden verkon yli tapahtuvan keskinäisen kommunikoinnin monimutkaisuutta. Verkon kompleksisuus ja koko järjestelmän turvaamisen haasteellisuus kasvaa suhteessa järjestelmässä olevien mikropalveluiden määrään. Lisäksi verkon kompleksisuus näkyy kasvavana haasteellisuutena järjestelmän kehittämisessä ja ylläpitämisessä [3]. Esimerkiksi mikropalveluiden välisessä kommunikaatiossa verkko-

ongelmat, verkon viive ja siirtonopeus voivat aiheuttaa virhetilanteita, joita voi olla haastavaa selvittää [13, luku 5]. Virhetilanteiden selvittäminen on haastavaa, jos virhe on tapahtunut toiminnallisuudessa, jonka monet mikropalvelut tuottavat yhteistyössä. Virheen selvittämiseksi saattaa joutua tarkastelemaan usean mikropalvelun lokeja ja viestiliikennettä. [16, luku 1]

Järjestelmän suunnittelu ja arkkitehtuuri ovat mikropalveluarkkitehtuurin keskiössä, sillä jälkikäteen kokonaisarkkitehtuuriin tai yksittäisen mikropalvelun piiriin kuuluvaan alueeseen on vaikea tehdä muutoksia. Jos yhden mikropalvelun piiriin kuuluvat toiminnallisuudet ovat suunniteltu huonosti, voi myöhemmin joutua suunnittelemaan koko mikropalvelun uusiksi, muokkaamaan myös muita mikropalveluita, tai pilkkomaan muokattava palvelu useampaan mikropalveluun [13, luku 5]. Järjestelmän arkkitehtuuri tulisi olla selkeä ja hyvin dokumentoitu, sillä mikropalvelut kommunikoivat keskenään. Rajapintakutsujen (engl. API call) käyttö palveluiden välillä johtaa eräänlaiseen sovintoon mikropalveluiden välille. Rajapintaversioinnin kanssa tulee olla tarkkana, että uudet rajapintaversiot ovat myös taaksepäin yhteensopivia. Nämä mikropalveluiden väliset suhteet toisiin mikropalveluihin saattavat olla piilossa tai vaikea hahmottaa etenkin uusille järjestelmän kehittäjille. [14]

Mikropalveluista koostuvan järjestelmän käyttöönotto ja suorittaminen kokonaisuudessaan voi olla monimutkaista, sillä ne vaativat useiden mikropalveluiden asetusten konfigurointia ja oikeanlaisen infrastruktuurin pystyttämistä mikropalveluita varten. Mikropalveluiden kanssa on suositeltavaa käyttää automatisointia ja julkaisuputkea, jotta testaus- ja julkaisuoperaatiot eivät olisi liian työläitä. [13, luku 5]

2.5 Rajapinnat ja kommunikointi mikropalveluiden välillä

Vaikka mikropalvelut ovat itsenäisiä ja toisistaan riippumattomia teknologiavalintojen suhteen, on tärkeää valita yhtenäinen teknologia mikropalveluiden väliseen kommunikaatioon. Tällöin yksittäisen mikropalvelun ei tarvitse tukea useaa eri viestintäprotokollaa, helpottaen koko järjestelmän toteuttamista. Useita teknologioita voi kuitenkin sekoittaa keskenään, jos se on perusteltua ja sille on tarvetta. Mikropalveluiden keskinäinen kommunikointi voidaan toteuttaa joko synkronisesti tai asynkronisesti, mutta usein mikropalveluiden kommunikaatioon käytetään synkronisia http-pohjaisia REST-rajapintoja. Viestintäprotokollan lisäksi on sovittava viestien formaatista. Yleisimmät vaihtoehdot viestintäformaatille ovat JavaScript Object Notation (JSON) ja eXtensible Markup Language (XML). [17, luku 3]

2.5.1 Viestintäformaatit

JSON on kevyt tiedonsiirtoformaatti, jota käytetään tiedon siirtämiseen järjestelmien välillä [18]. Se perustuu avain-arvo-pareista muodostuviin kokoelmiin ja arvolistoihin. Yhden JSON-rakenteen sisään voi laittaa muita rakenteita ja listoja. JSON-rakenteita voidaan käyttää millä tahansa ohjelmointikielellä, eli ne eivät ole riippuvaisia tietystä ohjelmointikielestä. Lähes kaikista ohjelmointikielistä löytyy kirjasto JSON-rakenteiden käsittelylle. Lisäksi JSON-rakenteen ovat melko helposti ymmärrettäviä. JSON-rakenne soveltuu esimerkiksi käyttäjän istuntoon liittyvien tietojen tallettamiseen kuten listauksesta 1 näkee. [18][19]

```
{
   ”id”: 3,
   ”nimi”: ”Testaaja Timo”,
   ”aloitusAika”: ”2021-01-01 15:00”,
   ”roolit”: [”ylläpitäjä”, ”työntekijä”]
}
```

Listaus 1. *Esimerkki JSON-rakenteen käyttämisestä*.

XML on tiedonsiirto- ja tallennusformaatti, joka muistuttaa HTML-merkintäkieltä. Tieto talletetaan XML-dokumenttiin käyttäen tageja, attribuutteja ja arvoja. Myös XML-rakenteiden käsittelyyn on melko hyvin saatavissa kirjastoja eri ohjelmointikielille. Listauksessa 2 on esimerkki käyttäjän istuntoon liittyvien tietojen tallettamisesta XML-rakenteessa. [20]

```
<istunto>
   <id>3</id>
   <nimi>Testaaja Timo</nimi>
   <aloitusAika>2021-01-01 15:00</aloitusAika>
   <roolit>
      <rooli>ylläpitäjä</rooli>
      <rooli>työntekijä</rooli>
   </roolit>
</istunto>
```

Listaus 2. *Esimerkki XML-rakenteen käyttämisestä.*

Muita rajapintojen tiedonsiirrossa käytettyjä formaatteja ovat Apache Avro ja protokollapuskurit, joita käytetään gRPC-viestintäprotokollan kanssa [17, luku 3]. Seuraavissa aliluvuissa käsitellään synkronista ja asynkronista kommunikaatiota mikropalveluissa sekä esitellään eri viestintäprotokollia näille kommunikaatiotavoille.

2.5.2 Synkroninen kommunikaatio

Synkronisessa kommunikaatiossa lähettäjä toimittaa pyynnön vastaanottajalle ja jää odottamaan vastausta. Vastaanottaja käsittelee pyynnön ja lähettää vastausviestin takaisin lähettäjälle. Synkroninen viestintä on mikropalveluissa yleensä toteutettu käyttäen REST-rajapintoja. [21, luku 3.5]

Representational State Transfer eli REST on 2000-luvulla Roy Fieldingin kehittämä arkkitehtuuri [21, s. 75]. Sen avulla kehitetään heikosti kytkettyjä palveluja sekä hajautettuja järjestelmiä, jotka käyttävät yleensä HTTP-protokollaa viestintään. [17, luku 3][21, luku 3.5]

REST perustuu navigointimalliin, joka edustaa kohteita ja palveluita verkossa. Näitä kohteita ja palveluita kutsutaan resursseiksi. Käyttäjä voi päästä käsiksi resursseihin käyttäen uniikkeja resurssitunnisteita. Resursseille voidaan tehdä erilaisia toimintoja kuten lukeminen, muokkaus, lisäys ja poisto. HTTP-protokollaa käytettäessä URL-osoitteet edustavat näitä resurssitunnisteita ja HTTP-protokollan standardioperaatioilla GET, PUT, DELETE ja POST voidaan suorittaa yllä mainittuja toimintoja resursseille. [17, luku 3][21, luku 3.5]

Tilattomuus on yksi REST-rajapintojen piirteistä. Tämä tarkoittaa sitä, että palvelin ei tallenna aiempien pyyntöjen parametreja, ja jokaisen pyynnön on sisällettävä kaikki mahdollinen tieto, jota tarvitaan pyynnön suorittamiseen. Tilattomuuden hyötynä skaalautuvuus paranee, rajapintojen toteuttaminen, virheiden selvittäminen, niistä toipuminen sekä järjestelmän monitorointi helpottuvat. Toisaalta pyyntöjen parametrien määrä kasvaa. Pyyntöjen koon kasvaminen voi olla merkittävää varsinkin, jos mikropalveluiden sisäiset rajapinnat ovat myös toteutettu REST-periaatteilla. [22]

Vaihtoehtoinen toteutustapa synkroniselle kommunikaatiolle on etäproseduurikutsut (engl. Remote Procedure Calls, RPC). Etäproseduurikutsujen tarkoitus on toimia samoin kuin funktiokutsut ohjelman sisällä, mutta kutsut aloitetaan nimensä mukaisesti ohjelman ulkopuolelta. Osa RPC-teknologioista vaatii rajapintakuvauksen toimiakseen. Rajapintakuvaus mahdollistaa myös asiakas- ja palvelintynkien generoinnin, jotta RPC-teknologian implementointi olisi mahdollisimman helppoa. [2, luku 4]

gRPC on vuonna 2015 julkaistu avoimen lähdekoodin etäproseduurijärjestelmä, jota voi suorittaa missä ympäristössä tahansa. Aluksi Googlen kehittämänä gRPC tunnettiin nimellä Stubby [23]. Yksi gRPC:n pääkäyttökohteista on mikropalveluarkkitehtuurit, ja esimerkiksi Netflix käyttää gRPC:tä [23]. gRPC:ssä viestit ovat binäärimuotoisia Protocol Buffer -viestejä. Niiden väitetään olevan suorituskyvyltään nopeampia kuin tekstipohjaiset JSON- ja XML-viestiformaatit. gRPC ei välttämättä sovellu ulospäin suuntautuviin rajapintoihin ohjelmistoissa, sillä gRPC-rajapinnat eivät ole niin joustavia kuin esimerkiksi GraphQL. Lisäksi gRPC:lle ei ole saatavilla niin hyvää selaintukea kuin esimerkiksi http-pohjaisille rajapinnoille. [24]

REST-rajapinnat ovat yleensä melko yksinkertaisia, joten monimutkaisempia kokonaisuuksia varten täytyy usein tehdä pyyntöjä eri palveluihin [17, luku 3]. Vuonna 2015 avoimena lähdekoodina julkaistu GraphQL suoriutuu kuvaillusta tilanteesta tarjoamalla kyselykielen rajapinnoille [25]. Sen tarkoitus on mahdollistaa tarkat haut rajapintojen datasta. Käyttäjä voi hakea yhdellä pyynnöllä tarvitsemansa. Tämä mahdollistaa rajapintojen kehittämisen tarkoituksenmukaisemmiksi. [17, luku 3] GraphQL soveltuu erityisesti sovellusten ulospäinsuuntautuvien rajapintojen teknologiaksi, sillä GraphQL-rajapinnat antavat käyttäjälle paremmat mahdollisuudet hallita haettua dataa [24]. GraphQL on REST-rajapintojen tavoin käytettävissä http-protokollan välityksellä. [25]

2.5.3 Asynkroninen kommunikaatio

Synkronisen ”lähetä pyyntö ja odota vastausta” -tyylisen kommunikaation lisäksi on olemassa vaihtoehtoinen tapa toteuttaa mikropalveluiden kommunikaatiota. Asynkronisessa kommunikoinnissa lähettäjä lähettää pyynnön, mutta ei jää odottamaan vastausta, vaan jatkaa muiden prosessien suoritusta taustalla. Vastauksen voi vastaanottaa kolmella eri tavalla [21, s. 80]. Ensimmäinen keino on lähettää pyynnön mukana viittaus paluumetodiin (engl. callback function), jota palvelin kutsuu suoritettuaan alkuperäisen pyynnön. Tällöin alkuperäinen lähettäjä saa vastauksen, kun palvelin kutsuu tätä paluumetodia. Toinen keino vastaanottaa vastaus alkuperäiseen pyyntöön on käyttää kyselyfunktiota, joka kysyy palvelimelta tietyn aikavälin välein, onko pyyntö suoritettu ja vastaus saatavilla. Kolmas keino on käyttää viestinvälittäjää, jota käytettäessä lähettäjä toimittaa viestit välittäjälle ja tämän jälkeen välittäjä lähettää viestin eteenpäin niille vastaanottajille, jotka kyseistä viestiä odottavat. Viesti lähetetään riippumatta siitä, vastaanottaako kukaan viestiä [2, luku 4]. Vastaanottajat puolestaan odottavat tiettyjä viestejä, mutta eivät välitä epäolennaisista viesteistä. [17, luku 3][21, luku 3.6]

Mikropalveluissa asynkroninen viestintä tekee palveluista autonomisempia, joka helpottaa mikropalveluarkkitehtuurin päämääriin pääsemisessä. Tämän lisäksi asynkroninen liikenne mikropalvelujärjestelmän sisäisesti saattaa nopeuttaa järjestelmän toimintaa. Synkronisesti toteutettu sisäinen viestintä sisältää paljon vastausten odottamista, joka saattaa hidastaa järjestelmää. [21, luku 3.6]

Asynkronista kommunikaatiota varten on kehitetty teknologioita, joista yksi on Advanced Message Queuing Protocol (AMQP). Se on avoimen lähdekoodin standardi, joka mahdollistaa asynkronisen viestijonoviestinnän sekä aiheisiin perustuvan julkaise-tilaa (engl. publish-subscribe) viestinnän [26]. AMQP on todella joustava – sen kanssa voi käyttää mitä vain ohjelmointikieltä, verkkoprotokollaa tai viestiformaattia [27, s. 74]. Asynkroninen ”julkaise-tilaa” -mallin mukainen viestintä sopii erityisesti mikropalveluiden keskinäiseen kommunikaatioon, sillä se on löyhästi kytkettyä ja helposti skaalattavissa [28]. Siinä viestien julkaisijat lähettävät viestit viestinvälittäjäpalvelulle, joka jakaa viestit tilaajille. Viestit sisältävät tietyn aiheen, ja tilaajille toimitetaan aiheiden mukaan tilaamansa viestit. Jotta mallin mukainen kommunikaatio voidaan ottaa käyttöön mikropalvelujärjestelmässä, tarvitaan yksi erillinen palvelu toteuttamaan AMQP-standardin mukainen viestinvälittäjän toiminnallisuus. Tätä varten on olemassa onneksi myös useita valmiita alustoja kuten RabbitMQ [29].

RabbitMQ on avoimen lähdekoodin viestinvälittäjäsovellus, jota esimerkiksi NASA on käyttänyt palvelininfrastruktuurissaan [29]. Sovellusta varten on toteutettu valmiita asiakaskirjastoja useille ohjelmointikielille [30], joten sen implementointi on melko yksinkertaista. Käyttöönotto mikropalveluissa vaatii ainoastaan RabbitMQ viestinvälittäjäpalvelimen ja asiakaskirjaston implementoinnin mikropalveluihin. Tutkimuksen mukaan mikropalveluarkkitehtuurissa RabbitMQ suoriutuu kuormasta REST-rajapintoja sujuvammin ja tasaisemmin, kun palvelulla on paljon yhtäaikaisia käyttäjiä [31]. Pienillä käyttäjämäärillä REST-rajapinnat olivat kuitenkin RabbitMQ:ta nopeampia. [26]

2.5.4 Rajapintojen ja palveluiden löytäminen

Mikropalveluista saatetaan julkaista nopealla aikavälillä useita versioita tai samasta mikropalvelusta saatetaan ajaa useampaa versiota yhtäaikaisesti. Jotta mikropalveluiden välinen kommunikaatio olisi mahdollista, on palveluiden oltava tietoisia toisistaan ja oikeista rajapinnoista, joihin lähettää pyyntöjä. Tämän vuoksi mikropalveluille yleensä toteutetaan keino palveluiden löytämistä varten.

Palvelun löytäminen (engl. Service Discovery) tarkoittaa sitä, että mikropalvelut paikantavat toisensa ja kommunikoivat keskenään. Tämä voidaan toteuttaa esimerkiksi määritystiedostoilla, joihin kaikki mikropalvelun asetukset ovat määritetty. Tiedostoon voidaan lisätä muiden mikropalveluiden osoitteet, joista ne löytyvät. Mikropalveluiden asetusten määritys eli konfigurointi tapahtuu yleensä automaattisesti, kun mikropalvelut otetaan julkaisuputken läpi käyttöön. [13, luku 7]

DNS (engl. Domain Name System) -protokolla varmistaa, että palvelinosoitteet kuten www.palvelu1.esimerkki.com osoittavat tiettyyn IP-osoitteeseen. Kullekin mikropalvelulle voidaan luoda oma palvelinosoite, jonka kautta tietty mikropalvelu on aina tavoitettavissa riippumatta IP-osoitteesta. Tämä helpottaa mikropalveluiden konfigurointia, sillä mikropalveluiden asetuksiin voidaan lisätä tietyt osoitteet, joista mikropalvelut ovat aina tavoitettavissa. DNS-protokollaa käytetään myös Kuberneteksessa ja sen avulla mahdollistetaan palveluiden ja Kubernetes-podien löytyminen halutuista osoitteista [32]. Toinen keino on käyttää työkalua tai tietokantaa, jonne mikropalveluiden osoitteet ovat talletettuna, ja hakea tiedot sieltä [16, luku 9]. [2, luku 11]

Palveluiden löytäminen voidaan toteuttaa myös palvelinpuolella. Se tapahtuu käyttämällä esimerkiksi kuormantasaajaa tai API-yhdyskäytävää. Kuormantasaajalla (engl. Load Balancer) on yleensä tieto kaikista käynnissä olevista mikropalveluista, joten se osaa ohjata pyynnöt oikeisiin osoitteisiin [16, luku 9]. Kuormantasaajaa käytetään mikropalveluinstansseissa jakamaan liikennettä ja pyyntöjä mikropalveluinstanssien kesken tasaisesti. Kuorman tasaamisella pyritään välttämään palveluiden tukkiutumista liian suuren kuorman takia varmistaen palvelun sujuva toimivuus. Kuormantasaamista varten on olemassa valmiita sovelluksia kuten Kubernetes [32]. [13, luku 7]

API-yhdyskäytävä (engl. API Gateway) on järjestelmä, jonka kautta kaikki liikenne kulkee sovellusten rajapintoihin. Sen avulla yksinkertaistetaan ja vakautetaan käyttäjille julkaistujen rajapintojen käyttöä. API-yhdyskäytävä on samankaltainen facade-suunnittelumallin kanssa, mutta sen toteutus tapahtuu verkkotasolla [33]. Pilvipalveluiden rajapintojen käyttöä voidaan ohjata ja rajoittaa monella tapaa API-yhdyskäytävän kautta. Sen kautta voidaan autentikoida, muokata pyyntöjä, ohjata pyyntöjä, yhdistellä useiden pyyntöjen tuloksia yhdeksi, monitoroida ja kirjata lokia järjestelmän käytöstä. API-yhdyskäytävän ansiosta järjestelmästä tulee ulkoapäin katsottuna enemmän monoliittisen ohjelman tyylinen ja rajapintojen käyttäminen helpottuu, kun useat irralliset mikropalveluiden rajapinnat koostetaan yhteen paikkaan käytettäväksi. [33]

2.6 Ympäristöt

Mikropalveluiden ylläpitäminen on yksi mikropalveluiden haasteista. Automatisoinnin, valmiiden alustojen ja pilvipalveluiden käyttö ovat keskeisiä työkaluja, joilla ylläpitämiseen liittyvää työkuormaa voidaan purkaa. Esimerkiksi Docker on suosittu teknologia tähän tehtävään. [16, luku 5]

Mikropalveluita voidaan ottaa käyttöön ja julkaista käyttäen Docker-teknologiaa. Docker on kevyt konttien virtualisointiteknologia, joka mahdollistaa ohjelmien ja ympäristöjen rakentamisen konttien sisään [34]. Kontit ovat eräänlaisia kevyitä virtuaalikoneita, joita suoritetaan esimerkiksi pilvipalvelimilla. Jotta mikropalvelut toimisivat oikein tarvitsevat ne oikeanlaisen ajoympäristön riippuen käytetyistä teknologia valinnoista. Docker-kontin sisään voidaan rakentaa halutunlainen ympäristö, ja kun ympäristö on kerran määritetty, se voidaan jatkossa pystyttää nopeasti ja vaivattomasti. Dockerin avulla sama kontti voidaan ottaa käyttöön ja suorittaa sitä monessa eri paikassa yhtäaikaisesti [16, luku 5]. Useita Docker-kontteja voidaan ajaa myös samalla palvelimella yhtäaikaisesti. Nämä toiminnallisuudet tekevät Dockerista ideaalisen valinnan mikropalveluille, sillä mikropalveluita päivitettäessä ja skaalattaessa tarvitaan kullekin mikropalvelulle pystyttää samalla sen ominainen ympäristö. [34]

Konttien hallintaa helpottamaan on kehitetty lisäksi työkaluja kuten Kubernetes ja Docker Swarm, joilla voidaan automatisoida konttien käyttöönottoprosessia [16, luku 4]. Kubernetes on Googlen kehittämä avoimen lähdekoodin konttienhallintajärjestelmä [32]. Sen avulla kontteja voidaan jakaa klustereihin eli ryppäisiin. Kukin klusteri koostuu joukosta laitteita, joilla ohjelmia suoritetaan. Näitä laitteita tai palvelimia kutsutaan noodeiksi (engl. node). Yhden Kubernetes-klusterin sisällä voi olla käytössä useita noodeja, joilla saadaan aikaan virheensietokykyä ja hyvää saatavuutta palveluille. Klusterien ja noodien lisäksi Kuberneteksessa on olemassa myös yksiköitä nimeltä podi (engl. pod). Noodit suorittavat klusteriin luotuja podeja. Podit puolestaan voivat koostua yhdestä tai useammasta kontista. Lisäksi Kuberneteksella voidaan määrittää miten paljon resursseja kukin podi saa käyttöönsä. Podit saavat oman IP-osoitteensa klusterin sisällä ja voivat keskustella keskenään, mutta eivät pysty oletusarvoisesti keskustelemaan ulkomaailman kanssa. Sitä varten podi täytyy julkaista ulospäin Kuberneteksessa. Tällä saavutetaan tietoturvaa mikropalvelujärjestelmässä, ja voidaan määrittää mitkä mikropalvelut ovat ulospäin näkyvissä. Kubernetes optimoi palvelininfrastruktuurin käyttöä sekä mahdollistaa useita automatisoituja toiminnallisuuksia kuten podien uudelleenkäynnistämisen kontin mennessä virhetilaan. [32][35]

Mikropalveluita voidaan ajaa ja julkaista käyttäen jonkin pilvipalveluntuottajan infrastruktuuria ja palveluita. Suurimpia pilvipalveluntuottajia vuonna 2020 olivat Amazon Web Services (AWS), Microsoft Azure, Alibaba Cloud ja Google Cloud [36]. Kullakin näistä pilvipalveluntuottajista on olemassa omat toteutukset mikropalveluarkkitehtuurissa ja muissa pilvisovelluksissa käytettäville palveluille. Palveluita ovat esimerkiksi tietokantapalvelut, tallennustilapalvelut, sovellusten suorittaminen, konttipalvelut, palvelimeton tietojenkäsittely (engl. serverless computing), virtuaalikoneet ja verkkoinfrastruktuuripalvelut [37]. Käyttäessä pilvipalveluita ylimääräinen ylläpitotyö palvelimien, tallennustilan ja verkon hallinnan suhteen jää pilvipalveluntarjoajien vastuulle. Pilvipalveluntarjoajien avulla voidaan pienentää työkuormaa, joka syntyy mikropalveluiden käyttämisestä ja ylläpitämisestä.

Mikropalveluarkkitehtuuri tuo useita etuja moniosaisten järjestelmien kehittämiseen ja on sopiva arkkitehtuuri skaalautuvuutta vaativille järjestelmille. Useiden palveluiden keskinäinen kommunikaatio luo kuitenkin monoliittiseen järjestelmään verrattuna monimutkaisuutta ja uusia haasteita. Mikropalveluiden sisäisen kommunikaation voi toteuttaa useilla eri teknologioilla. Ulospäin näkyvät rajapinnat ovat kuitenkin yleensä http-pohjaisia REST-rajapintoja. Tämän luvun pohjustuksen tarkoituksena on helpottaa lukijaa ymmärtämään UNA-projektin ja mikropalveluiden arkkitehtuuria sekä helpottaa hahmottamaan luvussa 4 käsiteltäviä mikropalveluiden autentikointi ja auktorisointiratkaisuja.

3. AUTENTIKOINTI JA AUKTORISOINTI

3.1 Autentikointi

Autentikointi on prosessi, jolla määritetään, onko käyttäjä se henkilö, joka hän väittää olevansa. Onnistuneen autentikoinnin perusteella käyttäjälle voidaan myöntää pääsy järjestelmään. Useissa järjestelmissä autentikoinnin jälkeen tapahtuu myös toinen prosessi, käyttäjän auktorisointi. Se tarkoittaa käyttöoikeuksien myöntämistä käyttäjälle. Näiden prosessien jälkeen käyttäjät pystyvät käyttämään suojattuja resursseja kuten tietokonejärjestelmiä, tietokoneverkkoja, tietokantoja, verkkosivuja ja muita ohjelmistoja. [38]

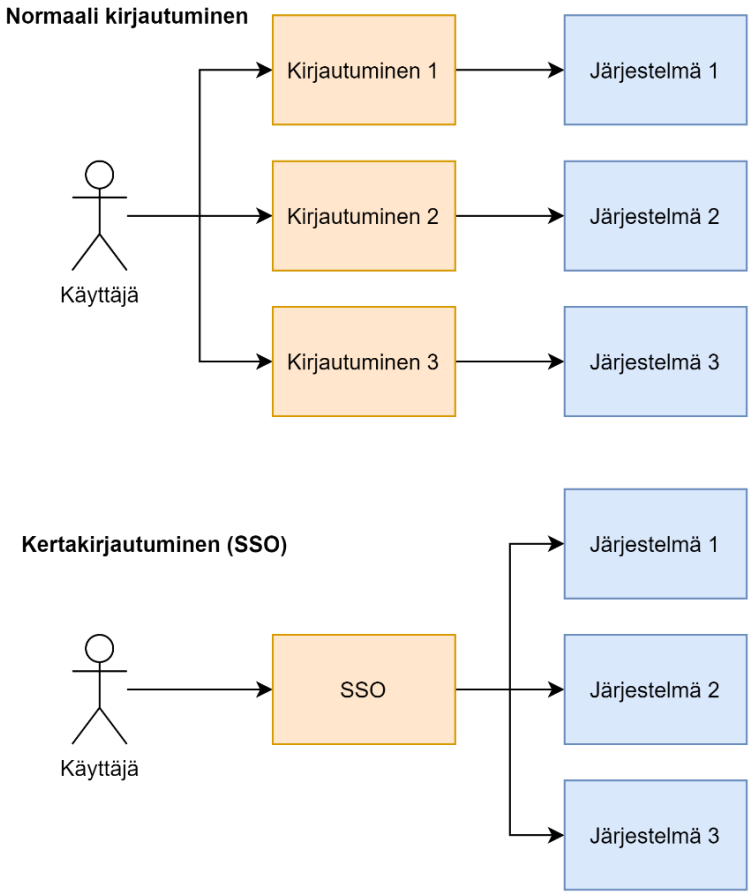
Autentikointiprosessissa käyttäjän tulee todistaa järjestelmälle olevansa se henkilö, jonka tunnuksella on kirjautumassa. Tämän käyttäjä voi todistaa tietämällä tai omistamalla, jota vain tunnuksen omistajan tulisi tietää tai omistaa. Identiteetin voi siis todistaa esimerkiksi tietämällä salasanan, omistamalla älykortin tai vaikka näyttämällä omaa sormenjälkeä skanneriin. Perinteisesti autentikoinnissa käytetään käyttäjätunnusta ja salasanaa. Käyttäjätunnuksen kanssa autentikoinnissa salasanan syöttäminen on vaihe, jolla käyttäjä todistaa olevansa syöttämänsä käyttäjätunnuksen omistaja. [39, luku 3]

Mikropalveluissa autentikointi on yhtä tärkeää kuin muissakin järjestelmissä. Jotta mikropalvelujärjestelmä olisi kokonaisuudessaan suojattu, täytyy jokaisen mikropalvelun käyttäminen suojata jollakin tapaa. Käyttäjän tunnistautumisprosessia ei ole kuitenkaan järkevää toteuttaa jokaiseen mikropalveluun, sillä se ei olisi yhtään käytännöllistä ja käyttäjäkokemus kärsisi järjestelmää käytettäessä. Tätä varten mikropalveluissa hyödynnetään usein kertakirjautumista, jonka avulla käyttäjä voi hyödyntää kaikkia järjestelmän mikropalveluita yhdellä tunnistautumiskerralla.

3.2 Kertakirjautuminen

Kertakirjautuminen (engl. Single Sign On, SSO) on autentikointimenetelmä, jossa käyttäjä pääsee yksillä tunnuksilla kirjautumaan useisiin eri ohjelmistojärjestelmiin. Kertakirjautuminen mahdollistaa sen, että käyttäjän tarvitsee autentikoitua vain kerran ja saa kirjautumisen jälkeen pääsyn useisiin sovelluksiin ilman, että niihin täytyisi kirjautua erikseen. Tätä on havainnollistettu kuvassa 2. Kertakirjautuminen on mahdollista siten, että sitä käyttävät palvelut sallivat kirjautumisen jonkin identiteetintarjoajapalvelun (identity provider, IdP) kautta. Käyttäjä kirjautuu identiteettipalveluun ja sinne luotu istunto mahdollistaa useiden palveluiden käyttämisen niin pitkään, kunnes istunto päättyy. Nykyään

kertakirjautumista käytetään monissa paikoissa kuten esimerkiksi työpaikoilla ja yliopistoissa. Työpaikalla käyttäjät voi käyttää useita työpaikan sovelluksia, kun he kirjautuvat vain kerran koneelle tai selaimeen. Kertakirjautuminen on myös tietoturvan ja käyttäjän käyttökokemuksen kannalta hyödyllistä, koska käyttäjän tarvitsee muistaa vain yksi salasana ja salasanoja ei tarvitse tallentaa useisiin järjestelmiin. [40][41, luku 11]

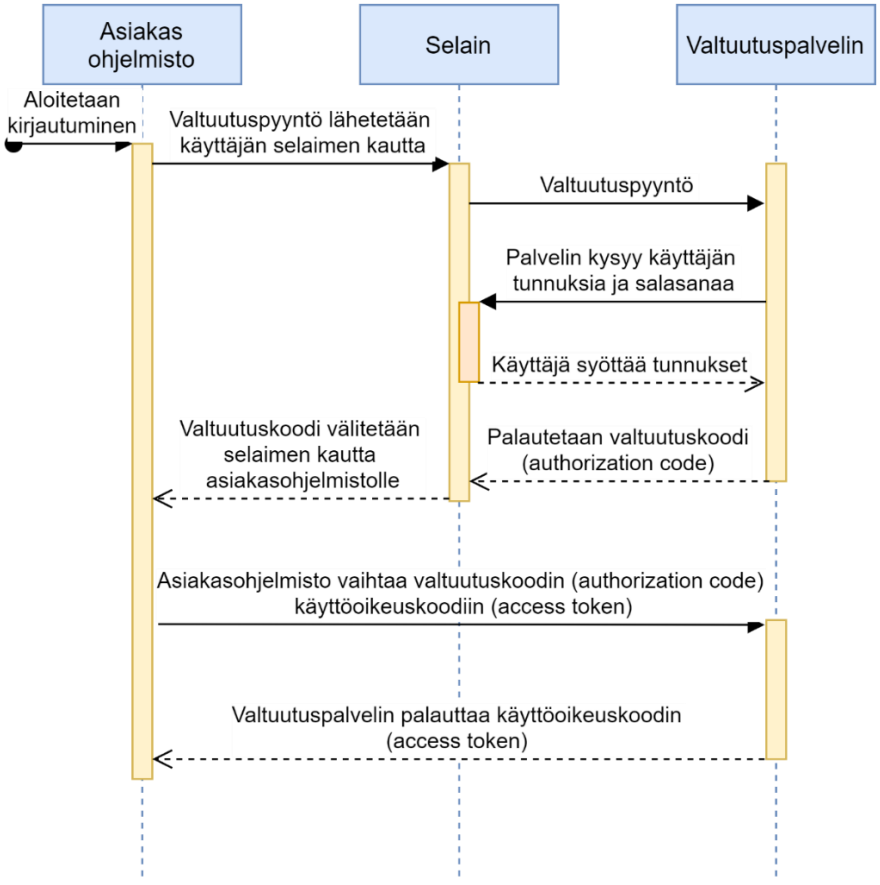


Kuva 2. *Normaali kirjautuminen ja kertakirjautuminen havainnollistettuna.*

Kertakirjautumisratkaisut voidaan toteuttaa monilla eri tekniikoilla kuten Kerberoksella, Ligthweight Directory Access -protokollalla (LDAP), RADIUS-protokollalla, evästeillä, digitaalisilla varmenteilla ja web-pohjaisilla ratkaisuilla. Web-pohjainen kertakirjautuminen on yleensä toteutettu protokollilla kuten Security Assertion Markup Language (SAML), Open Authorization (OAuth), OpenId Connect (OIDC) tai JSON Web Token (JWT). Web-pohjaista SSO-teknologioita käsitellään tarkemmin seuraavissa aliluvuissa, sillä mikropalvelut ovat yleensä web-pohjaisia järjestelmiä. [42][43]

3.2.1 OAuth

OAuth 2.0 on auktorisointikehys, joka mahdollistaa kolmannen osapuolen ohjelmiston saamaan pääsyn johonkin toiseen http-palveluun ja siellä jonkin käyttäjän tietoihin [44, s. 1]. Tärkeimpiä OAuth2.0 -protokollan tarjoamia hyötyjä ovat rajapintojen suojaus, kertakirjautumisen mahdollistaminen, palveluiden integrointi yhteen, auktorisoinnin delegointi palvelulta toiselle ja helpompi palveluiden monitorointi [45, luku 1]. Sen sijaan, että kolmannen osapuolen ohjelmisto käyttäisi suoraan käyttäjän tunnuksia, käytetään käyttöoikeuskoodia (access token). Käyttäjä voi kirjautua luotetussa kirjautumispalvelimessa suoraan ja saa sieltä käyttöoikeuskoodin, joka annetaan kolmannelle osapuolelle käytettäväksi. Tällä käyttöoikeuskoodilla kolmannen osapuolen ohjelmisto pääsee käyttäjänä tunnistautuneena tekemään asioita käyttäjän puolesta. Tällä menettelyllä käyttäjän luottamuksellisia kirjautumistietoja ei tarvita jakaa kolmannelle osapuolelle. Käyttöoikeuskoodia voidaan käyttää myös järjestelmän monitoroinnin helpottamiseen, sillä sen avulla voidaan seurata käyttäjän tekemiä pyyntöjä ja istunnon voimassaoloa. OAuth2.0 käytetään nykyään laajasti monissa verkkopalveluissa ja REST-rajapinnoissa [45, luku 1]. [44, s. 4–5]



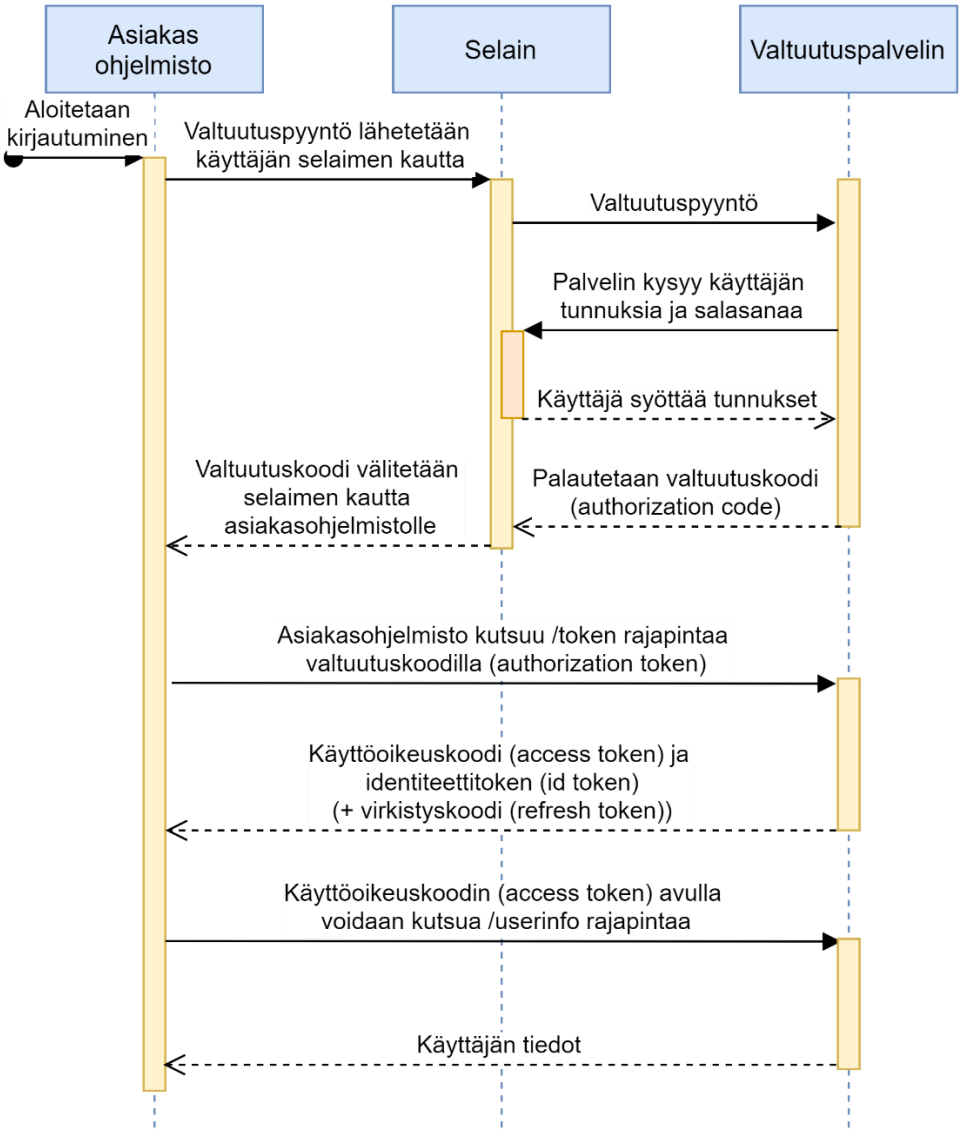
Kuva 3. *OAuth 2.0 valtuutuskoodilla valtuuttamisen työnkulku [44, s. 23][45, luku 4]*

OAuth 2.0:ssa on useita työnkulkuja valtuutuksen myöntöä varten ja ne soveltuvat erilaisiin käyttötarkoituksiin. Näistä yleisin on kuvan 3 mukainen valtuutuskoodilla valtuuttaminen [45, luku 2]. Valtuutuskoodityönkulussa asiakasohjelmisto lähettää valtuutuspyynnön valtuutuspalvelimelle, jonka jälkeen käyttäjän täytyy tunnistautua valtuutuspalvelimelle. Onnistuneen tunnistautumisen jälkeen valtuutuspalvelin palauttaa valtuutuskoodin, jonka voi vaihtaa toisesta valtuutuspalvelimen rajapinnasta käyttöoikeuskoodiin. Kun käyttöoikeuskoodi on haettu, asiakasohjelmisto voi tunnistautuneena käyttää palveluita, resursseja ja rajapintoja antamalla käyttöoikeuskoodin mukaan pyyntöihin. Muita valtuutuksenmyöntövaihtoehtoja ovat implisiittinen valtuuttaminen, resurssin omistajan tunnusten kautta valtuuttaminen ja asiakasohjelmiston tunnusten kautta valtuuttaminen. [46, luku 4]

3.2.2 OpenID Connect

OpenID Connect on OAuth2.0 protokollan päälle rakennettu identiteettitaso, joka mahdollistaa käyttäjän identiteetin todentamisen. Se on otettu käyttöön vuonna 2015 ja OIDC:n edeltäjä OpenID, on kehitetty SAML protokollan jalanjälkiä seuraten vuonna 2005 [46, luku 6]. OpenID Connect toimii lähes samoin kuin OAuth2.0, mutta tunnistautumisen jälkeen käyttöoikeustunnisteen (engl. access token) lisäksi käyttäjästä saadaan identiteettitoken (engl. id token). Vaihtoehtoisesti käyttäjän identiteetti ja oikeudet voidaan selvittää lähettämällä niin sanottuun ”UserInfo”-rajapintaan käyttöoikeustunnisteen [41, luku 6]. Identiteettitoken on koodattu JWT muotoon, ja se sisältää tietoa autentikoinnin ajankohdasta ja autentikoidusta käyttäjästä. [47]

OpenId Connect sisältää OAuth2.0:n tavoin useita työnkulkuja käyttäjän valtuuttamista varten. Alkuperäinen OpenId Connect määritelmä sisältää seuraavat työnkulut: valtuutuskoodi työnkulku, implisiittinen työnkulku ja hybrid työnkulku. Valtuutuskoodilla valtuuttaminen tapahtuu lähes samoin kuin OAuth2.0:ssa, mutta valtuutuskoodin vaihtamisen yhteydessä saadaan käyttöoikeuskoodin lisäksi identiteettitoken. Samalla voidaan valinnaisesti pyytää rajapinnalta virkistyskoodia, jonka avulla saadaan uusi käyttöoikeuskoodi edellisen vanhetessa. Kuvassa 4 on esitetty OIDC:n valtuutuskoodilla valtuuttamisen työnkulkua. [41, luku 6]



Kuva 4. *OpenId Connect valtuutuskoodilla valtuuttamisen työnkulku. [46, luku 6]*

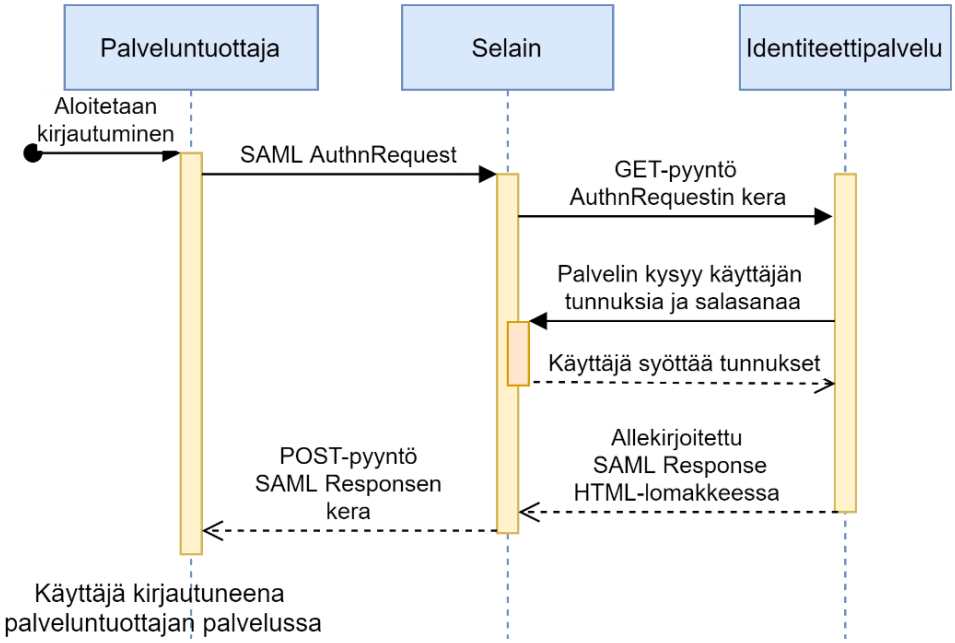
3.2.3 SAML

Security Assertion Markup Language on xml-pohjainen protokolla autentikointi- ja auktorisointitietojen turvalliseen välittämiseen verkkopalveluiden välillä. SAML 2.0 ratkaisee pääasiassa kaksi vaatimusta, jotka ovat yleisiä isoissa yrityksissä. Ensimmäinen näistä vaatimuksista on web-pohjainen kertakirjautuminen useiden verkkotunnusten yli. Toinen vaatimuksista on federoitu identiteetti, joka tarkoittaa useiden palveluiden yhteisesti käyttämää käyttäjän identiteettiä. [45]

SAML-protokollaa käyttäessä yksi verkkopalveluista on ns. palveluntuottaja (engl. service provider, SP) ja toinen palveluista on identiteettipalvelu. Kertakirjautuminen SAML-protokollaa käyttäen tapahtuu siten, että käyttäjä ohjataan palveluntuottajan palvelusta

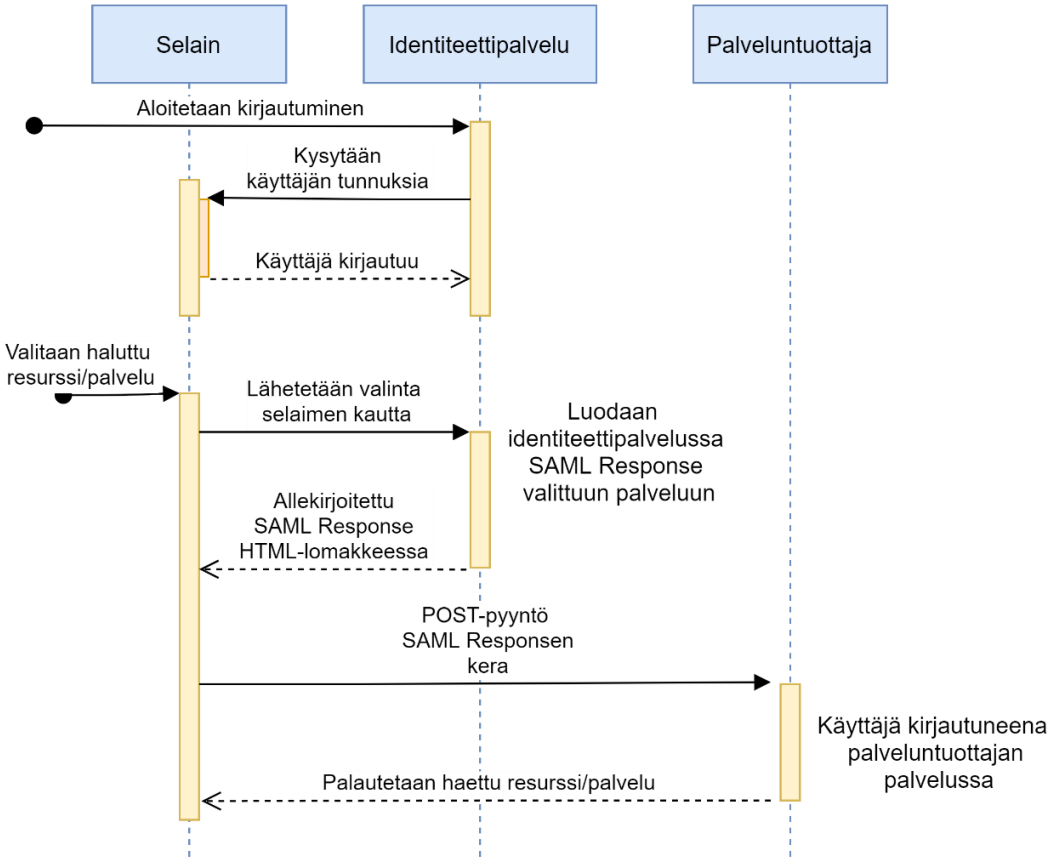
identiteettipalveluun, jossa käyttäjä kirjautuu. Sen jälkeen tieto kirjautuneesta käyttäjästä sekä käyttäjän identiteetistä välitetään takaisin palveluntuottajalle. Tieto siirretään palveluiden välillä allekirjoitetussa xml-pohjaisessa SAML-vastauksessa. Käyttäjä pääsee käyttämään palveluntuottajan palvelua ilman, että käyttäjä tarvitsee erillistä tunnusta palveluntuottajan järjestelmään. Identiteettipalvelu ja palveluntuottajan järjestelmä luottavat toisiinsa. Luottamus tässä yhteydessä tarkoittaa sitä, että järjestelmät tunnistavat saapuvien viestien lähettäjät ja sen perusteella voivat hyväksyä ja prosessoida viestit. Luottamus toteutetaan etukäteen konfigurointivaiheessa rekisteröimällä palvelut ja vaihtamalla järjestelmien allekirjoitusvarmenteet keskenään. Allekirjoitusvarmennetta käyttäen kumpikin palvelu voi todentaa viestien alkuperän tarkistamalla SAML-viestien allekirjoitukset. SAML mahdollistaa myös federoidun kirjautumisen, jossa useat palvelut viittaavat yhteisesti samaan käyttäjän identiteettiin. Palvelut käyttävät samaa identiteettipalvelun identiteettiä eri palveluissa saman käyttäjän tunnistamiseen. [48, luku 3]

Kertakirjautuminen voidaan toteuttaa SAML-protokollassa kahdella eri tavalla: palveluntuottajan tai identiteettipalvelun aloittamana. Palveluntuottajan aloittamassa kertakirjautumisessa välitetään SAML-autentikointi pyyntö identiteettipalveluun, jossa käyttäjä tunnistetaan. Tämän jälkeen palveluntuottajalle palautetaan SAML-paluuviesti, joka sisältää käyttäjän identiteetti- ja autentikointitiedot. Kuvassa 5 esitetään palveluntuottajan aloittaman SAML-kertakirjautumisen prosessia. [48, luku 5][49]



Kuva 5. *SAML kertakirjautuminen palveluntuottajan aloittamana. [48, luku 5.1.2]*

Identiteettipalvelun aloittamassa kertakirjautuminen tapahtuu kuvan 6 mukaisesti. Käyttäjä ensin tunnistetaan identiteettipalvelussa, ja sieltä lähetetään suoraan käyttäjän identiteettitiedot sisältävä SAML-vastaus palveluntuottajalle. Kuvaillussa tapauksessa identiteettipalvelu on yleensä osana jotain muuta järjestelmää, jossa mahdollistetaan kertakirjautumisella integraatio muihin palveluihin helposti. [48][49]



Kuva 6. *SAML kertakirjautuminen identiteettipalvelun aloittamana. [48, luku 5.1.4]*

3.2.4 JWT

JSON Web Token on avoin standardi turvalliseen tiedon välittämiseen JSON-muodossa. JWT:n käyttötarkoituksia ovat autentikointi ja tiedon välittäminen. Autentikointi käyttötarkoituksessa käyttäjä saa kirjautuessaan tokenin, jonka avulla käyttäjä saa pääsyn palveluihin ja resursseihin. JWT mahdollistaa myös kertakirjautumisen. [50]

JWT koostuu kolmesta osasta: otsikkorakenteesta, payload-rakenteesta ja allekirjoituksesta. Otsikkorakenne sisältää tiedon algoritmista, jolla allekirjoitus muodostetaan [51, s. 6]. Esimerkki otsikkorakenteesta löytyy listauksesta 3.

```
{
   "alg": "HS256",
   "typ": "JWT"
}
```

Listaus 3. *JWT-tunnisteen header-osio.*

Payload-rakenne koostuu käyttäjään liittyvistä tiedoista ja tunnisteen luontiajankohdasta. Vaihtoehtoisesti rakenteeseen voidaan lisätä tunnisteen vanhenemisajankohta, tieto tunnisteen luojasta ja sen vastaanottajasta sekä muita rajoitetietoja tunnisteen käytölle. Lisäksi payload-rakenteeseen voidaan halutessa lisätä omia tietoja, kuten käyttäjän roolit tai käyttöoikeudet palvelussa [51, s. 7–9]. Alla listauksessa 4 on esimerkki yksinkertaisesta JWT-rakenteen payload-osiosta.

```
{
   "sub": "1234567890",
   "name": "Esimerkki Eetu",
   "iat": 1516239022
}
```

Listaus 4. *JWT-tunnisteen payload-osio.*

JWT:n viimeinen osa koostuu allekirjoituksesta, joka on muodostettu valitulla algoritmilla ja salausavaimella base64-enkoodatuista otsikko- ja payload-rakenteista. Lopuksi kukin osa base64-enkoodataan erikseen ja yhdistetään toisiinsa pistemerkillä [51, s. 7]. Alla esimerkki base64-enkoodatusta JWT:stä, jonka osat havainnollistettu eri väreillä (punainen = otsikko, vihreä = payload, sininen = allekirjoitus).

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0N
TY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5
MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw
5c
```

JWT-rakenteen sisältämään informaatioon voidaan luottaa, ja sen lähettäjä voidaan varmistaa luotettavaksi, koska JWT-tietorakenteet ovat allekirjoitettuja digitaalisesti. Allekirjoitettujen tunnisteiden tietoja ei voida muokata huomaamatta, koska muutos huomattaisiin allekirjoituksen tarkistamishetkellä. Jos nämä tietorakenteet allekirjoitetaan epäsym-

metrisen salauksen avainparilla, allekirjoituksen avulla voidaan varmistua myös lähettäjästä. Epäsymmetrisessä salauksessa yksityistä avainta käytetään salauksen luomiseen ja julkista avainta salauksen purkamiseen [39, luku 1]. Siten yksityisellä avaimella tehty allekirjoitus voidaan varmistaa julkisella avaimella. JWT-rakenteen sisältämät tiedot voidaan allekirjoittamisen lisäksi tarvittaessa myös salata, jos välitettävä tieto on salaista. [50]

JSON Web Tokenien etu verrattuna esimerkiksi SAML-teknologiaan, on JSON-parsintakirjastojen yleisyys eri ohjelmistokielissä. Lisäksi XML-muotoisten SAML assertioiden kanssa työskentely on työläämpää kuin JSON-rakenteiden kanssa, sillä XML ei sisällä luonnollista dokumentista objektiin muunnosta. Lisäksi JSON-rakenteiden helppokäyttöisyys tekee JWT:stä sopivan myös mobiilialustoille. [50]

3.3 Vahva tunnistautuminen

Salasanat ovat laajasti käytettyjä nykyään, mutta ne eivät ole enää paras ratkaisu autentikoinnissa. Lyhyet salasanat ovat huonoja turvaamaan käyttäjätunnusta, koska ”brute-force” -hyökkäyksellä pystytään kokeilemaan kaikki eri salasanavaihtoehdot suhteellisen nopeasti. Pitkät salasanat ovat usein vaikeita muistaa ja huonoja käyttökokemuksen kannalta. Lisäksi pitkät salasanat kirjoitetaan yleensä muistiin, jolloin salasana on alttiina varkaudelle. Jos samaa salasanaa käytetään useissa palveluissa, hyökkääjän on mahdollista saada pääsy moniin eri palveluihin ja järjestelmiin yhdellä tietomurrolla. [41, luku 12]

Tunnistautuminen pelkällä salasanalla ei ole kovin turvallista ja vahvempien salasanojen muistaminen on vaikeaa, joten tätä turvallisuuden puutetta on korjattu keksimällä vahvempia autentikointikeinoja [39, luku 6]. Yksi laajasti käytetty mekanismi vahvaan tunnistautumiseen on lähettää käyttäjälle kertakäyttöisiä salasanoja, joko tekstiviestin, sähköpostin tai järjestelmään liittyvän mobiilisovelluksen kautta. One-time password (OTP) eli kertakäyttöinen salasana voidaan käyttää vain kerran vaikeuttaen huomattavasti hyökkääjän mahdollisuuksia päästä sisään järjestelmään. Toinen vahvan tunnistautumisen keino on käyttää yksityistä salausavainta tai varmennetta, joka on talletettu johonkin esineeseen tai laitteeseen, kuten älykorttiin tai älypuhelimeen. Esimerkiksi Suomessa terveydenhuollossa käytetään älytoimikortteja, jotka sisältävät tällaisen varmenteen. Salausavaimen tai varmenteen käyttö on yleensä suojattu vielä PIN-koodilla, joka tulee syöttää palveluun tunnistautumisen yhteydessä. [41, luku 12]

Menettelyä, jossa käyttäjältä vaaditaan useampi vaihe tunnistautumiseen, kutsutaan monivaiheiseksi tunnistautumiseksi (engl. multifactor authentication, MFA). Kahden vaiheen

tunnistautumista kutsutaan kaksivaiheiseksi tunnistautumiseksi (engl. Two-factor authentication, 2FA). Salasanojen lisäksi käytetään nykyään laajasti monivaiheista tunnistautumista. 2FA:ta tai MFA:ta käytettäessä salasanan syöttämisen jälkeen käyttäjän täytyy yleensä antaa järjestelmälle vielä kertakäyttöinen numerokoodi, jonka älypuhelimessa oleva ohjelma generoi. Tällainen tunnistautumisohjelma on etukäteen linkitetty käyttäjän tunnuksiin, jotta monivaiheista tunnistautumista voidaan käyttää. Kaksi- ja monivaiheinen tunnistautuminen on käytössä erityisesti terveydenhuollossa, pankki- ja rahoitusjärjestelmissä, sekä nykyään myös sosiaalisessa mediassa. [39, luku 6]

3.4 Käyttäjän auktorisointi ja pääsynhallinta

Auktorisointi on prosessi, jossa määritetään tunnistautuneelle käyttäjälle rajoitteet ja luvat siihen, mitä kaikkea käyttäjä pystyy järjestelmässä näkemään ja tekemään; auktorisointi määrittää käyttöoikeudet järjestelmässä [39, luku 3]. Järjestelmissä voi olla monenlaisia toiminnallisuuksia, joista osa saattaa olla järjestelmän toimivuudelle kriittisiä. Kriittiset toiminnallisuudet tulisi rajoittaa vain niille käyttäjille, jotka ovat siihen tehtävään osoitettu. Auktorisoinnilla vältetään tilanteita, joissa käyttäjä tekisi jotain, mitä hän ei saisi tehdä.

Pääsynhallinta on kykyä sallia tai estää tietyn resurssin käyttämistä. Pääsynhallinta perustuu seuraaviin tietoturvan periaatteisiin: tehtävien eriyttämisen periaate (engl. Separation of Duties, SD), vähimpien oikeuksien periaate (engl. Least Privilege, PLP) ja tiedon tarpeen periaate (engl. Need to Know, NtK). Tehtävien eriyttämisen periaatteella pyritään siihen, että yksittäinen henkilö, rooli tai ryhmä ei kykene suorittamaan kaikkia osia prosessista itse. Vähimpien oikeuksien periaatteen mukaan oikeudet tulisi rajata suppeimpiin mahdollisiin oikeuksiin, joilla käyttäjä pystyy suorittamaan tehtävänsä. Tiedon tarpeen periaatteen mukaan tiettyyn tietoon voisi saada pääsyn vain, jos on perusteltu tarve tietää kyseinen asia. Esimerkiksi lääkärit käsittelevät potilaiden tietoja, mutta he eivät voi perusteetta tarkastella henkilön potilastietoja, ellei heillä ole tarve tietoihin esimerkiksi potilaan hoitamisen näkökulmasta. Vähimpien oikeuksien ja tiedon tarpeen periaatteiden mukaan käyttäjällä on järjestelmässä pääsy vain tehtävään tarvittaviin resursseihin ja tietoihin. [52]

3.4.1 DAC, MAC ja ACL

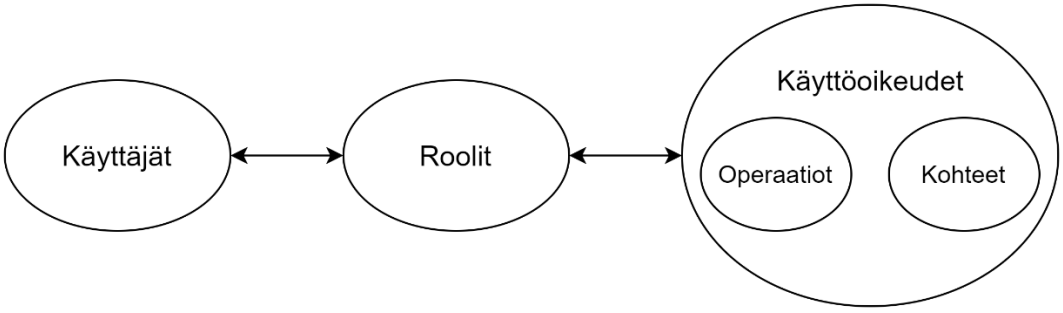
Pääsynhallintaa varten on kehitetty jo 1970-luvulta lähtien erilaisia malleja, joista yksi ensimmäisiä on ollut Lampsonin pääsynhallinta matriisi [53]. Vain muutama kaikista malleista on menestynyt ja jäänyt käyttöön [54, s. 41]. Pääsylista (engl. Access Control List,

ACL) on yksinkertaisimmillaan lista käyttäjistä, joilla on pääsy johonkin tiettyyn resurssiin tai tietoon. Pääsylista liittyy määrättyyn resurssiin, ja jos resursseja on useita, tarvitaan useita pääsylistoja. Pääsynhallinta on manuaalisesti liian työlästä ja heikosti automatisoitavissa, joten on keksitty parempia malleja. [52]

1970-luvulla ilmaantuneet Discretionary Access Control (DAC) ja Mandatory Access Control (MAC) ovat pääsynhallintamalleja, jotka ovat vielä nykyäänkin käytössä [54, s. 41]. DAC mallissa tiedon ja resurssien omistajat voivat hallita omistamiensa resurssien oikeuksia. Käyttäjät voivat itse muuttaa oikeuksia ilman, että ylläpitäjän tarvitsee tarkastaa muutoksia tai olla mukana tekemässä niitä. MAC-mallissa käyttäjät eivät voi muokata oikeuksia itse. MAC on keskitetty politiikka, joka määrittää sen, kuka pääsee mihinkin tietoon käsiksi. MAC-mallissa tieto on jaettu eri turvatasoihin, joiden perusteella käyttäjät pääsevät tietoon käsiksi omasta turvatasostaan riippuen. Käyttäjille annetaan esimerkiksi jonkin numeron mukaan luokitus, jonka perusteella he pääsevät käsiksi kaikkeen tietoon, johon heidän oma luokituksensa riittää, mutta ei sitä ylemmän tasoisiin tietoihin. Näitä turvaluokituksia voi olla erikseen tiedon lukemiseen ja muokkaamiseen. [54][55, s.17–18]

3.4.2 Roolipohjainen pääsynhallinta

Roolipohjainen pääsynhallinta (engl. Role-Based Access Control, RBAC) perustuu nimensä mukaisesti rooleihin. RBAC mahdollistaa helpommin automatisoitavan ja monipuolisemman oikeuksien määrittämisen käyttäjille, kuin edellä käsitellyissä MAC- ja DAC-malleissa. Käyttäjille määritetään rooli tai rooleja, joiden perusteella käyttöoikeudet määrittyvät. Roolit sisältävät käyttöoikeuksia, jotka koostuvat operaatiosta ja operaation kohteesta. Esimerkki tällaisesta käyttöoikeudesta on käyttäjän oikeus nähdä toisen käyttäjän puhelinnumero. Tässä esimerkissä operaatio on ”read” eli lukuoperaatio, ja operaation kohteena ovat käyttäjien puhelinnumerot. Operaatio voi olla jokin muu kuten ”write” eli kirjoitusoperaatio, jolloin samaan kohteeseen osoitettuna käyttäjä pystyy muokkaamaan toisten käyttäjien puhelinnumeroita. Käyttöoikeuksia lisätään rooleille sen mukaan, mitä kullakin roolilla on tarkoitus tehdä. Kullakin roolilla voi olla useita käyttöoikeuksia, ja sama käyttöoikeus voi olla myös useammalla roolilla. Sama rooli voi olla usealla käyttäjällä, ja yhdellä käyttäjällä voi olla useita rooleja. Näiden käsitteiden välillä on monesta-moneen yhteys (engl. many-to-many relationship), joka ilmenee myös alla olevassa kuvassa 7 kaksisuuntaisina nuolina. [52][55, s. 61–65]

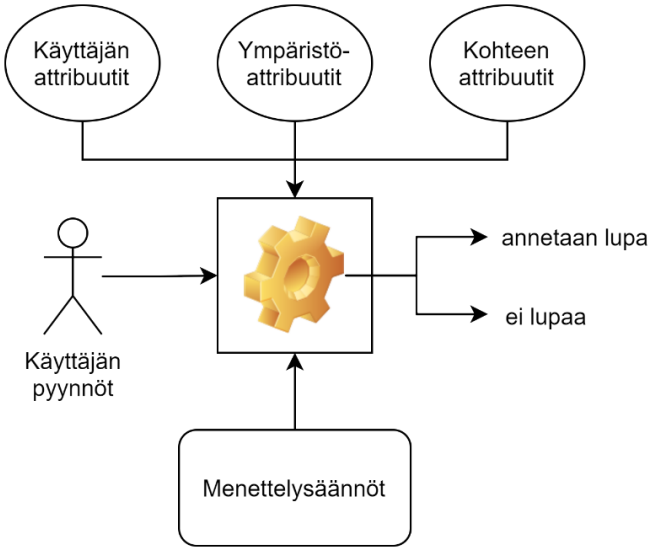


Kuva 7. *Käyttäjien, roolien ja luvitusten suhteet RBAC-mallissa. [55, s. 64]*

RBAC-malli helpottaa huomattavasti käyttöoikeuksien hallintaa, sillä ylläpitäjät voivat muokata rooleja sen sijaan, että jokaisen käyttäjän käyttöoikeudet muokattaisiin erikseen. Lisäksi RBAC-malli mahdollistaa roolien periytymisen. Esimerkiksi rooli A periytyy roolista B, jolloin roolilla A on kaikki roolin B käyttöoikeudet, ja sen lisäksi se voi sisältää muita käyttöoikeuksia, joita roolilla B ei ole. [55, s. 62, 73]

3.4.3 Attribuuttipohjainen pääsynhallinta

Roolipohjainen käytönhallinta ei ole kaikkiin tilanteisiin sopiva. RBAC-mallissa roolien erottaminen eri konteksteissa on hankalaa ja voi johtaa siihen, että rooleja on suuri määrä. Esimerkiksi voi olla rooli, jonka halutaan toimivan hieman eri tavalla riippuen missä osastossa käyttäjä toimii. RBAC-mallissa täytyy luoda kullekin osastolle oma rooli. Tietyn yksittäisen käyttöoikeustarpeen luonti ja hallinta RBAC-mallissa saattaa olla työlästä, ja roolimäärittelyt eivät aina ole riittäviä ilmaisemaan oikean elämän pääsynhallinta tilanteita. Muun muassa näiden ongelmien ratkaisemiseksi on kehitetty attribuuttipohjainen (Attribute-Based Access Control, ABAC) pääsynhallintamalli, jossa käyttöoikeudet määrittyvät erilaisten määritteiden tai attribuuttien perusteella. [56, s. 13–15]



Kuva 8. *ABAC-mallin toimintaa kuvaava prosessi. [56, s. 14]*

Attribuutti-pohjainen pääsynhallinta perustuu kuvan 8 mukaisesti attribuutteihin, ympäristöllisiin tai ajallisiin ehtoihin ja joukkoon menettelysääntöjä, jotka ovat määritelty attribuuttien ja ehtojen perusteella. Attribuutteja voivat olla esimerkiksi konteksti, resurssi, toiminta, kohde, aihe tai käyttäjä. Attribuutit ovat jaettu käyttäjän attribuutteihin ja kohteen attribuutteihin. Käyttäjän attribuutit esittävät käyttäjää kuvailevia attribuutteja kuten ikä, identiteetti, rooli, turvaluokitus tai vastaava. Kohteiden attribuutit esittävät dataa ja resursseja. Lisäksi on ympäristöön ja kontekstiin liittyviä attribuutteja. Kaikista näistä attribuuteista luodaan menettelysääntöjä. Esimerkiksi voidaan luoda menettelysääntö, joka sallii käyttäjien katsoa yrityksen tiedostoja, jos he ovat rooliltaan työntekijöitä ja kello on välillä 06:00 – 18:00. Esimerkissä käyttäjät, joilla on roolia ilmaiseva attribuutti ”työntekijä” ja kellonaikaan perustuva ehto täyttyy, voivat säännön perusteella päästä näkemään kohteet, joilla on attribuutti ”yrityksen tiedostot”. Uusia käyttäjiä lisättäessä annetaan käyttäjille vain kyseinen attribuutti, eikä monimutkaisempaa käyttäjänhallintaa tarvita, sillä sääntöjä ei tarvitse määritellä uudestaan. Nykyään on olemassa kaksi standardia, joilla ABAC-pohjaista käytönhallintaa voidaan toteuttaa kokonaisvaltaisesti. Standardit ovat nimeltään Extensible Access Control Markup Language (XACML) ja Next Generation Access Control (NGAC). [54][56, s. 33-39]

4. MIKROPALVELUIDEN AUKTORISOINTI- JA AUTENTIKOINTIRATKAISUT

Tässä luvussa tarkastellaan kirjallisuudesta löytyneitä mikropalveluiden autentikointi- ja auktorisointiratkaisuja. Löytyneistä ratkaisuvaihtoehdoista etsitään tutkimusten mukaan suositelluimmat ja yleisimmät. Luvussa 6 käytetään löydetyistä ratkaisuvaihtoehdoista yleisimpiä UNA projektissa toteutetun ja luvussa 5 esitellyn ratkaisun vertailuun ja kehittämiseen.

4.1 Aiemmat tutkimukset aiheesta

Mikropalveluiden autentikoinnista ja auktorisoinnista on tehty joitakin aikaisempia tutkimuksia. Tämän työn toteuttamista varten perehdytään kirjallisuudesta löytyviin ratkaisuihin, jotka antavat pohjaa Ytimen toteuttamien ratkaisujen arviointiin. Soldani ym. [14, s. 221] toteavat, että pääsynhallinta on yksi suurimmista haasteista mikropalveluiden turvallisuudessa. Haasteellisuus johtuu mikropalveluiden hajautetusta luonteesta. Kukin yksittäinen mikropalvelu tarvitsee kyvykkyyden tarkistaa sille saapuvien pyyntöjen autenttisuuden ja käyttöoikeudet [14, s. 221]. Seuraavissa kappaleissa selvitetään mitä aiheesta on aiemmin tutkittu, ja etsitään ratkaisuja mikropalveluiden autentikointiin ja auktorisointiin.

Pereira-Vale ym. [57] tekivät kirjallisuuskatsauksen mikropalveluiden turvallisuudesta. Heidän mukaansa kirjallisuudessa autentikointi ja auktorisointi ovat mikropalveluiden yleisin turvallisuusmekanismi. Akateemisesta ja harmaasta kirjallisuudesta löytyneet mikropalveluihin sopivimmat autentikointimallit ja -standardit olisivat OAuth, OpenID Connect, kertakirjautuminen sekä JWT. Tutkimuksessa huomattiin, että mikropalveluiden auktorisoinnissa on usein käytetty RBAC- ja ABAC-malleja. Edellä mainittuja autentikointi- ja auktorisointimalleja on käsitelty tarkemmin luvussa 3. [57]

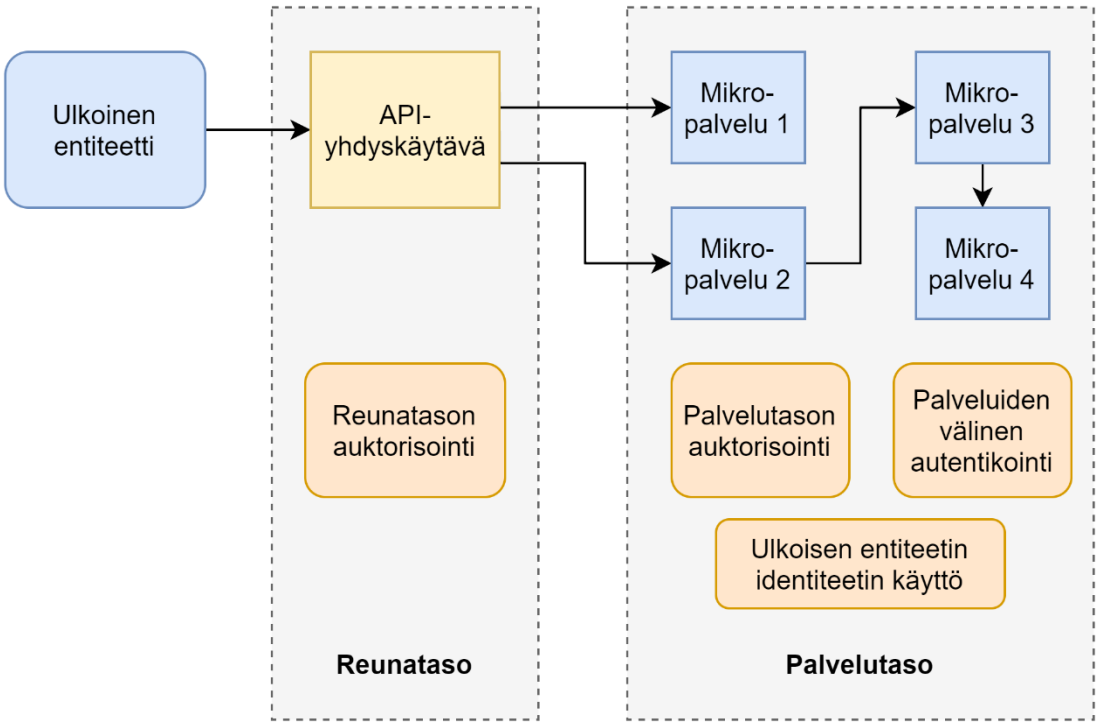
Myös Hannousse ym. [58] tutkivat mikropalveluiden turvallisuutta kirjallisuuskatsauksella. Heidän tutkimuksensa käsittelee mikropalveluiden turvallisuusmekanismeja, turvallisuusuhkia ja niiden identifiointia, sekä mikropalveluiden turvallisuutta ohjelmiston eri tasoilla. Heidän mukaansa auktorisointi- ja autentikointiratkaisut mikropalveluissa eivät ole innovatiivisia, sillä ne usein perustuvat jo olemassa oleviin tekniikoihin ja standardeihin. [58]

Chandramoulin tutkimuksessa [12] on esitetty erilaisia strategioita mikropalveluiden turvallisuuteen liittyen. Tutkimuksen mukaan mikropalveluiden auktorisoinnissa keskitetty

arkkitehtuuri on vaadittu, jotta pääsynhallintasääntöjen hallinta, toimeenpaneminen sekä toimittaminen mikropalveluille olisi mahdollista, erityisesti jos mikropalveluita on paljon. Mikropalveluiden rajapintojen autentikointi tulisi toteuttaa käyttäen vähintään API-avainta, mutta OAuth, SAML, OpenID Connect sekä JWT-tunnisteet ovat turvallisempia ratkaisuja. [12]

Yu ym. tutkivat [11] pilvipalveluiden reunalla toimivien ”sumu”-laskentaa (engl. fog computing) käyttävien mikropalveluiden turvallisuutta. He tuovat tutkimuksessa esiin mikropalveluiden turvallisuushaasteita ja ratkaisuja niihin. Lisäksi he esittelevät ratkaisun mikropalveluiden välisen liikenteen turvaamiseksi. Heidän ratkaisunsa perustuu varmenteisiin sekä AES- ja RSA-algoritmeilla salattuun liikenteeseen mikropalveluiden välillä. [11]

Barabanov ja Makrushin ovat tutkineet [59] mikropalveluihin pohjautuvien järjestelmien autentikointi- ja auktorisointiratkaisuja. He nostavat esiin ratkaisuista käytetyimpiä ja suosituimpia. Ratkaisut voidaan tutkimuksen mukaan luokitella seuraavasti: reunatason auktorisointi, palvelutason auktorisointi, ulkoisen identiteetin välittäminen ja palveluiden välinen autentikointi. Kuvasta 9 näkee, kuinka mikropalvelujärjestelmä voidaan jakaa eri auktorisoinnin ja autentikoinnin osa-alueisiin.



Kuva 9. *Mikropalveluiden auktorisoinnin ja autentikoinnin eri osa-alueet. [59]*

Tutkimuksessa [59] nousee esiin suositeltavia toimintatapoja ja periaatteita mikropalveluiden autentikoinnin toteuttamiseen. Esimerkiksi auktorisointi tulisi toteuttaa käyttäen keskitettyä rakennetta, jossa sääntöjen päätäntäpiste on upotettu mikropalveluihin, ja

mahdolliset roolit, attribuutit ja säännöt haetaan keskitetystä auktorisointipalvelusta. Auktorisointi ja autentikointi tulee toteuttaa laajasti käytössä olevaa ratkaisua käyttäen, eikä omatekoisten ratkaisujen käyttö ole suositeltavaa. Omatekoisissa autentikointi- ja auktorisointi ratkaisuissa on enemmän ylläpidettävää, ja kehittäjät on koulutettava käyttämään niitä. Lisäksi omille ratkaisuille ei ole saatavissa avoimen lähdekoodin yhteisöjä, joista saa tarvittaessa tukea. Autentikointi- ja auktorisointiratkaisujen toteuttamiselle on hyvä olla erillinen tiimi, joka kehittää ja ylläpitää niitä. Mikropalveluarkkitehtuurissa on suositeltavaa toteuttaa ”defence-in-depth” -periaatetta, jonka mukaan auktorisointi sekä autentikointi tapahtuu järjestelmässä usealla eri tasolla. [59]

Edellä mainittu Barabanovin ym. [59] tutkimus oli ainoa mikropalveluiden autentikointiin ja auktorisointiin rajautuva tutkimus, joten se loi paremman pohjustuksen aiheeseen muihin tutkimuksiin verrattuna. Seuraavat aliluvut käsittelevät mikropalveluiden autentikointi- ja auktorisointiratkaisuja. Luvut ovat jaettu edellä mainitun tutkimuksen perusteella tehtyihin luokitteluihin: 4.2 palveluiden välinen autentikointi, 4.3 reunatason auktorisointi, 4.4 palvelutason auktorisointi sekä 4.5 ulkoisen identiteetin tai tunnisteen käyttö. Luvussa 4.6 arvioidaan eri mallien toimivuutta mikropalveluarkkitehtuurissa.

4.2 Palveluiden välinen autentikointi

Mikropalveluarkkitehtuurissa on kahdenlaista autentikointia: käyttäjän autentikointia ja mikropalveluiden toisilleen lähettämien pyyntöjen autentikointia, eli palveluiden välistä autentikointia (engl. service-to-service authentication). Yleisesti käyttäjien autentikointi voidaan toteuttaa jollain valmiilla järjestelmällä ja kertakirjautumisella, käyttäen esimerkiksi aikaisemmin esiteltyjä SAML, OAuth tai OpenID Connect -teknologioita. [60]

Mikropalveluiden välisessä autentikoinnissa pyynnöt ovat peräisin toiselta mikropalvelulta. Pyyntöjä vastaanottavalle mikropalvelulle ei ole olennaista, saapuvatko pyynnöt suoraan käyttäjältä vai toiselta mikropalvelulta. Mikropalvelun tulee kuitenkin varmistua siitä, että pyynnöt ovat luotettavia. Mikropalveluilla ei ole samanlaisia tunnuksia kuin käyttäjillä, joten niiden on tunnistauduttava luotettavaksi toisella tapaa. Yleensä järjestelmän välisissä ratkaisuissa saatetaan käyttää API-avainta autentikointiin. [60]

API-avain on pyyntöjen mukana välitettävä salasana, joka identifioi mikropalvelun. API-avaimien käyttö on nopeaa ja helppoa, mutta se tuo mukanaan haasteita. Mikropalveluille täytyy suunnitella avaimienhallinta, ja avaimet täytyy tallettaa turvalliseen paikkaan, jotta API-avaimet eivät paljastu ulkopuolisille. Rajamoulin mukaan [12, s. 18] pelkän API-avaimen käyttö ei ole suositeltavaa rajapinnoissa, joista palautetaan arkaluonteisia tietoja. Hickmanin [60] sekä Rajamoulin [12, s. 18] mukaan parempi ratkaisu olisi käyttää

JWT-tunnistetta API-avaimien sijaan. JWT-ratkaisussa lähettävä mikropalvelu luo JWT:n, joka sisältää tokenin vanhentumisajankohdan, lähettävän mikropalvelun nimen, vastaanottavan mikropalvelun nimen sekä mahdollisesti muita tietoja. JWT allekirjoitetaan salaisella avaimella, jonka vain vastaanottava ja lähettävä mikropalvelu tietävät. Pyyntöjen mukana JWT välitetään otsikkotietueessa, ja vastaanottava mikropalvelu voi tarkistaa siinä välitettyjen tietojen perusteella, onko pyyntö luotettava. JWT:n käsittelyä varten on paljon valmiita kirjastoja ja vaihtoehtoisesti JWT:n luontia varten voidaan toteuttaa oma mikropalvelu, josta muut mikropalvelut pyytävät JWT:n omiin kutsuihinsa. Salaisten avaimien tallettaminen ja käsittely voidaan jättää JWT-mikropalvelun vastuulle. [60]

Myös Indrasiri ja Siriwardena [17, luku 11] mainitsevat JWT:n yhdeksi ratkaisuksi mikropalveluiden välistä autentikointia varten. Heidän ratkaisussaan kerrotaan edellä mainitun lisäksi mahdollisuudesta salata JWT, jolloin JWT:n avulla voidaan välittää dataa epäturvallisen kanavan läpi turvallisesti. JWT voidaan allekirjoittaa myös käyttäen varmenteita salaisten avaimien sijaan. JWT:tä lähettävä mikropalvelu allekirjoittaa JWT:n omalla varmenteellaan, ja vastaanottava mikropalvelu varmistaa allekirjoituksen tallennetulla julkisella varmenteella tai JWT:n mukana kulkevalla julkisella varmenteella. Jos jokaisessa järjestelmään kuuluvassa mikropalvelussa pidetään listaa jokaisesta luotettavasta varmenteesta, mikropalveluiden skaalautuvuus huononee. Ongelma ratkaistaan luomalla oma juurivarmenne, jolla luodut keskitason varmenteet ovat luotettavia. Näin voidaan vähentää huomattavasti mikropalveluiden käyttöönotossa niihin asennettavien varmenteiden määrää. Sen lisäksi he mainitsevat toisen keinon, joka on nimeltään Transport Layer Security (TLS). TLS tunnetaan myös nimellä Secure Sockets Layers (SSL), ja se on salausprotokolla, joka suojaa tietoliikenteen verkkojen yli. Kyseessä on sama salausmenetelmä, jota selain käyttää käyttäjän vieraillessa HTTPS alkuisilla verkkosivuilla. TLS autentikoinnissa tapahtuu varmenteisiin perustuva kättely, jolla osapuolet varmistavat toistensa identiteetin ja sopivat käytetyt salausalgoritmit liikenteelle. JWT ja TLS ratkaisua voidaan käyttää yhdessä, eli JWT kulkee TLS-yhteyden kautta, jolloin myös JWT-tunnisteet ovat salattuja kaiken muun liikenteen ohella. [17, luku 11]

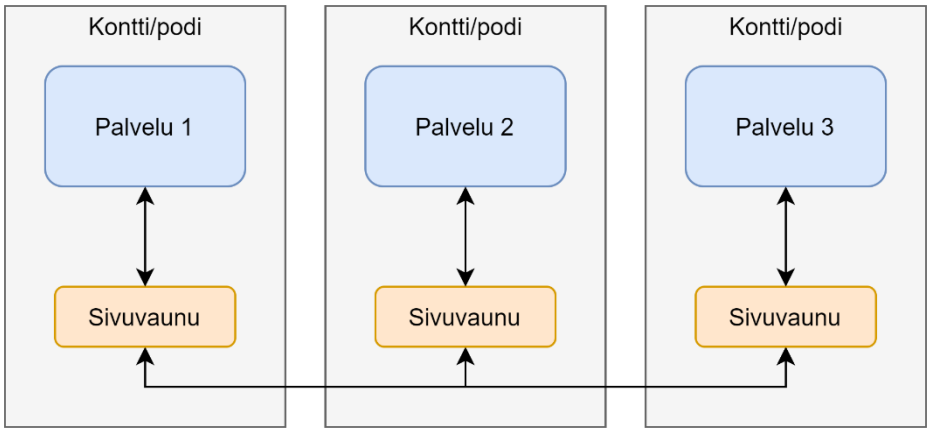
Mikropalveluiden välinen liikenne voidaan turvata myös verkkotasolla käyttämällä palomuuria tai muusta maailmasta eristettyä verkkoa [59, s. 8]. Mikropalvelut voidaan asettaa yksityiseen virtuaaliseen verkkoon, jonka sisällä mikropalveluiden välinen liikenne kulkisi [61, s. 1]. Virtuaalisen verkon liikenne näkyy ulospäin salattuna eikä siihen pääse käsiksi ilman salauksen purkamista verkon ulkopuolelta. Kubernetes käyttää verkkomallia [62], jossa Kubernetes-podit voivat kommunikoida suoraan saman Kubernetes-noden sisällä olevien muiden podien kanssa. Node luo sisäisen verkon, jossa kullekin podille annetaan

oma IP-osoite. Ulkoisen liikenteen ohjaaminen sisäiseen verkkoon täytyy tehdä esimerkiksi API-yhdyskäytävän tai kuormantasaajan kautta. [61][62]

UNA Ytimessä palveluiden välinen liikenne kulkee lähinnä REST-rajapintojen välillä. Rajapintojen suojaaminen on toteutettu TLS/SSL tekniikalla, eli kaikki rajapinnat ovat HTTPS osoitteiden takana, jolloin mikropalveluiden välinen liikenne on salattua. Lisäksi mikropalvelut ovat keskenään samassa verkossa, ja tähän sisäiseen verkkoon ei pääse ulkoa käsiksi. Ainoastaan API-yhdyskäytävän kautta avatuista rajapinnoista voidaan ottaa yhteys järjestelmän mikropalveluihin.

Yksi keino mikropalveluiden välisen liikenteen suojaamiseen on käyttää palveluverkkoarkkitehtuuria (engl. Service Mesh). Palveluverkkoarkkitehtuuri on turvallinen infrastruktuuritaso mikropalveluiden ”palvelusta-palveluun”-kommunikaatiolle. Siinä mikropalveluiden keskinäinen autentikointi, kommunikaatio ja muu liikenne mikropalveluihin kulkee palveluverkon kautta. Palveluverkko voidaan toteuttaa kahdella tapaa. Joko koodikirjaston avulla, joka sisällytetään jokaiseen mikropalveluun tai ”sivuvaunu”-kontilla (engl. sidecar), jota suoritetaan jokaisen mikropalvelun kyljessä. [63][64]

Kirjastona toteutettu palveluverkkoratkaisu vaatii mikropalveluita käyttämään samaa ohjelmointikieltä toteutuksessaan. Sivuvaunu erillisenä moduulinaan on parempi vaihtoehto. Sivuvaunu-ratkaisu on näistä käytetyin [63]. Kaikki palveluiden väliseen kommunikointiin liittyvä logiikka voidaan abstrahoida pois mikropalveluista, ja jättää palveluverkon tehtäväksi. Palveluverkkoa käytettäessä kommunikointiin liittyvä koodi täytyy toteuttaa vain kerran, jonka jälkeen kyseinen toiminnallisuus voidaan jakaa sivuvaunujen kautta kaikille mikropalveluille käytettäväksi [2, luku 5]. Palveluverkko on erityisen hyödyllinen, jos mikropalveluita on paljon, mutta sen toteuttaminen vain muutamalle mikropalvelulle ei ole järkevää. Palveluverkko mahdollistaa myös muita verkkoliikenteeseen liittyviä ominaisuuksia, kuten palveluiden löytämisen, kuorman tasaamisen tai dynaamisesti ohjautuvia pyyntöjä [64]. Sivuvaunujen tai palveluverkon tehtäväksi voidaan liittää myös muita mikropalveluissa tarvittavia toiminnallisuuksia kuten pääsynhallinnan ja käyttölokien kirjaamisen [65]. Kuvassa 10 on havainnollistettu palveluverkon käyttämistä osana mikropalveluarkkitehtuuria. [66][64]



Kuva 10. *Mikropalveluiden kommunikaatio palveluverkon välityksellä. [64]*

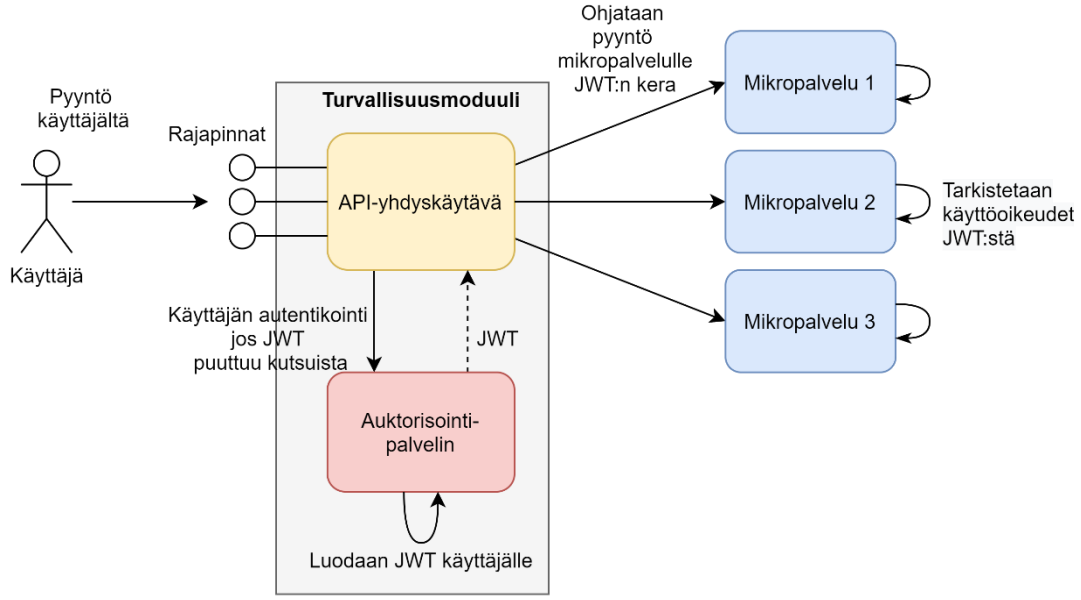
4.3 Reunatason auktorisointi

Mikropalveluiden toiminnassa auktorisointia ja autentikointia voidaan toteuttaa järjestelmän toiminnan eri tasoilla. Järjestelmän reunalla tapahtuva auktorisointi (engl. edge-level authorization) ja autentikointi tarkoittaa aluetta, jossa järjestelmä on käytettävissä ulkomaailmasta. Kaikki järjestelmästä ulospäin näkyvät rajapinnat ovat järjestelmän reuna, josta kutsut etenevät järjestelmän sisään. Mikropalvelujärjestelmän reunalla tapahtuva autentikointi ja auktorisointi toteutetaan yleensä käyttäen API-yhdyskäytävää ja mahdollisesti siihen liittyvää autentikointijärjestelmää. API-yhdyskäytävä on sisäänkäynti järjestelmän mikropalveluihin, joten myös auktorisointi ja autentikointi on luonnollista toteuttaa sisäänkäynnin yhteyteen [12, s. 10]. Myös monoliittisissa ohjelmissa tunnistautuminen tapahtuu ohjelman reunalla ennen kuin muuta toiminnallisuutta käynnistetään. Järjestelmään kulkeva liikenne ohjataan auktorisoitavaksi jo järjestelmän reunalla ennen kuin se ohjataan eteenpäin oikealle mikropalvelulle. Kun kaikki liikenne kulkee API-yhdyskäytävän kautta, järjestelmän ulospäin avoimena oleva hyökkäyspinta-ala pienenee. [12, s. 10][33]

UNA Ytimessä API-yhdyskäytävää käytetään helpottamaan liikenteen ohjaamista oikeisiin mikropalveluihin. API-yhdyskäytävän avulla on määritetty osoitteet, jotka osoittavat tiettyihin mikropalveluihin. Sen ansiosta kaikki mikropalvelut ovat helposti löydettävissä määritetyistä osoitteista. UNA Ytimessä API-yhdyskäytävän käyttöä varten hyödyntävä järjestelmä tarvitsee API-avaimen. API-avain laitetaan kutsuihin mukaan otsikkotietueeseen, ja vain oikeilla API-avaimilla pyynnöt pääsevät API-yhdyskäytävästä läpi [67, luku 4.1]. API-avain on asiakaskohtainen eli kukin hyödyntävä järjestelmä saa oman avaimen, jolla rajapintoja voi käyttää [67, luku 4.1]. API-avain ei yksinään anna täyttä pääsyä

rajapintoihin, vaan käyttäjän täytyy edelleen olla kirjautunut, ja käyttäjällä pitää olla riittävät oikeudet käyttämiinsä rajapintoihin. API-avain on siis vain yksi lisäosa järjestelmän turvallisuuteen.

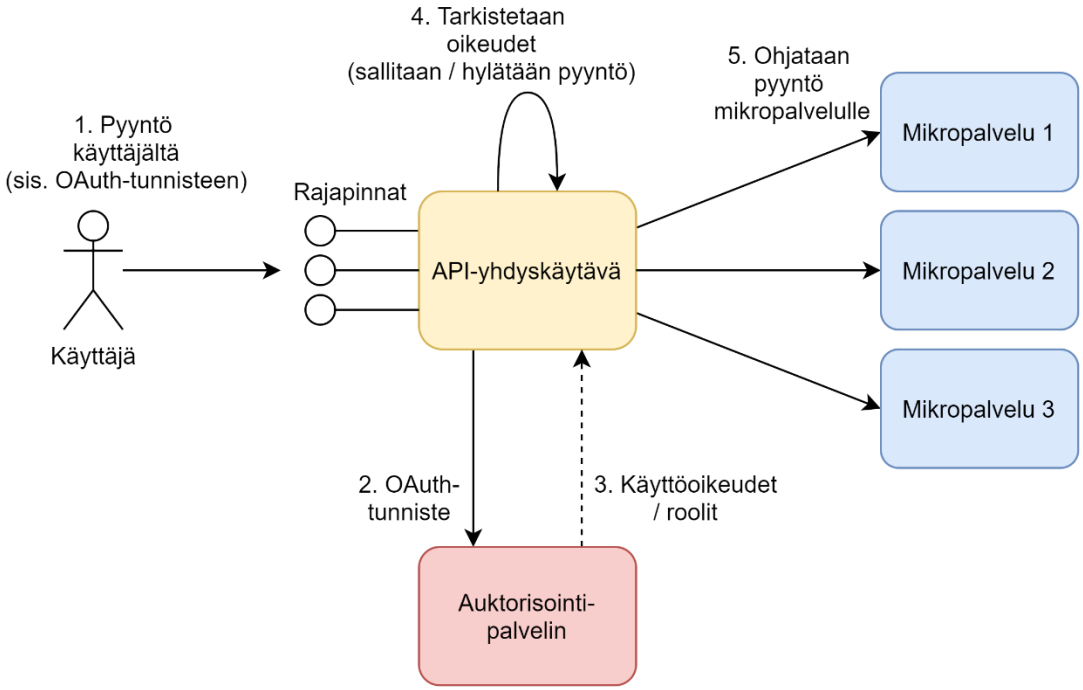
Xu ym. [68] esittelevät IoT-mikropalvelujärjestelmään sopivan autentikointitavan. Esineiden internetissä laitteilla ei ole paljon resursseja eikä kovin suurta prosessointitehoa, joten autentikoinnin tulee olla mahdollisimman kevyttä. Tutkimuksessa esitetty autentikointiratkaisu koostuu mikropalvelun reunalla olevasta API-yhdyskäytävästä, JWT-tunnisteista ja turvallisuusmoduulista. Reunatason auktorisointi ja autentikointi mahdollistaa pienen kaistanleveyden ja viiveen järjestelmässä, koska järjestelmän sisäisistä mikropalveluista ei tarvitse tehdä ylimääräisiä auktorisointikutsuja, kun auktorisointi on jo tapahtunut järjestelmän reunalla. Käyttäjälle luodaan kirjautuessa JWT-tunniste, joka liitetään autentikoinnin jälkeisiin kutsuihin mukaan. Kutsujen saapuessa mikropalvelun reunalla olevaan turvallisuusmoduuliin, käyttäjä auktorisoidaan ja tarvittavat oikeudet myönnetään kutsussa olevan JWT-tunnisteen sisällön avulla. JWT sisältää käyttäjälle sallitut mikropalvelut ja rajapinnat, sekä muut käyttäjän tiedot. Kuvassa 11 havainnollistetaan tätä mallia. [68]



Kuva 11. *API-yhdyskäytävää ja JWT-tunnistetta hyödyntävä auktorisointiratkaisu. [68]*

Bhingole esittelee [69] mikropalveluiden reunatason auktorisointi- ja autentikointiratkaisun, jossa edellä esitellyn ratkaisun tavoin käytetään API-yhdyskäytävää ja OAuth-tunnisteita käyttäjän tunnistamiseen. Kun pyyntö saapuu API-yhdyskäytävään, tarkistetaan pyynnön mukana välitetty OAuth-tunniste auktorisointipalvelimelta. Jos tunnistetta vas-

taava istunto löytyy, päästetään pyyntö eteenpäin mikropalveluihin. Auktorisointi on toteutettu ratkaisuun siten, että pyyntöjen autentikoinnin jälkeen haetaan istuntoon liittyvät roolit ja oikeudet. Niiden perusteella päätellään, voiko käyttäjän pyynnön päästää kyseiseen mikropalvelun rajapintaan. Ratkaisua on havainnollistettu kuvassa 12. Esitellyssä ratkaisussa mikropalvelutasolla ei ole välttämätöntä suorittaa pääsynhallintaa ja autentikointia, mutta järjestelmä tulee toteuttaa siten, että kaikki liikenne kulkee API-yhdyskäytävän kautta. [69]



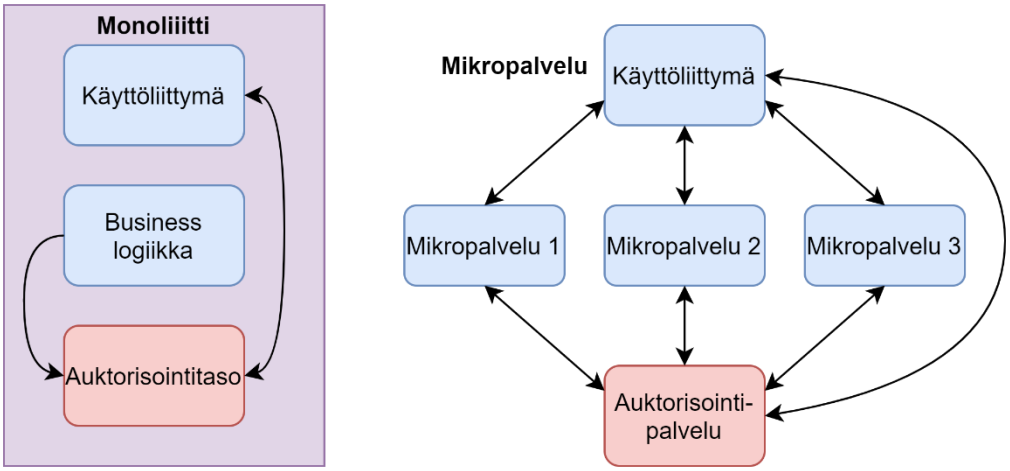
Kuva 12. *API-yhdyskäytävää hyödyntävä autentikointi- ja auktorisointiratkaisu. [69]*

4.4 Palvelutason auktorisointi

Palvelutason auktorisointi (engl. service-level authorization) tarkoittaa mikropalveluarkkitehtuurissa kunkin yksittäisen mikropalvelun tekemää auktorisointia ja pääsynhallintaa. Jotta mikropalvelut voivat päättää mitä tehdä saapuville kutsuille, täytyy mikropalveluiden saada tieto kutsun lähettäjästä ja lähettäjän käyttöoikeuksista. Mikropalvelutason pääsynhallintaan on useita keinoja. [17, luku 11]

Yksi keinoista on toteuttaa pääsynhallinta keskitetysti yhdessä mikropalvelussa, joka hoitaa oikeuksien tarkistamisen [17, luku 11]. Pyyntöjen saapuessa mikropalveluihin täytyy kyseisen mikropalvelun lähettää pääsynhallintamikropalvelulle tarkastuspyyntö käyttäjän tunnistamiseksi. Tarkastuspyyntöön laitetaan mukaan alkuperäisen pyynnön mukana saapunut istunnon tunniste, pääsytunniste tai vastaava tunniste. Pääsynhallinta-

mikropalvelussa tarkistetaan tunnistetta vastaavat oikeudet ja päätetään, voidaanko mikropalvelulle saapunut alkuperäinen pyyntö sallia suoritettavaksi. Ratkaisu muistuttaa paljon monoliittisen järjestelmän auktorisointia, koska mikropalvelut ovat aina liittyneitä toisiinsa tämän yhden auktorisointimoduulin kautta. Kuvassa 13 on havainnollistettu keskitettyä auktorisointia. Keskitetyn ratkaisun haittapuolena on ylimääräinen liikenne mikropalveluiden välillä sekä liikenteestä aiheutunut viive järjestelmässä. Reinicken mukaan [70] yhteen pisteeseen keskitetty auktorisointi on huonoa suunnittelua ja mikropalveluiden periaatteiden vastaista, koska se luo järjestelmään niin sanotun Single point of failure -pisteen. Se tarkoittaa, että kyseisen auktorisointimoduulin vikaantuessa koko järjestelmän toiminta estyy. [17, luku 11][70]

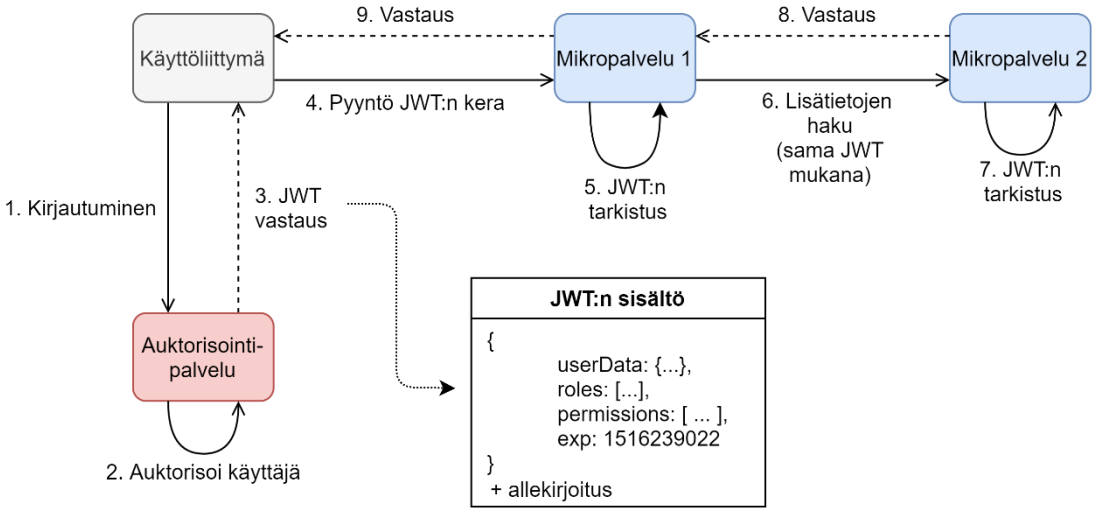


Kuva 13. *Keskitetty auktorisointi monoliitti- ja mikropalvelujärjestelmässä. [70]*

UNA Ytimessä toteutettu pääsynhallintaratkaisu muistuttaa paljon tällaista keskitettyä ratkaisua. Se toimii kuitenkin hieman eri tavoin kuin yllä esitellyssä tavassa. Sen sijaan, että lupa pyynnön suorittamiselle pyydetään keskitetystä moduulista, haetaan keskitetystä moduulista tieto käyttäjän oikeuksista, ja itse lupa pyynnön suorittamiselle päätellään kussakin mikropalvelussa käyttäjän oikeuksien perusteella.

Toisessa palvelutason auktorisointiratkaisussa pääsynhallintaa suoritetaan jokaisessa mikropalvelussa. Kyseessä on hajautettu malli, jossa mikropalveluihin saapuvien pyyntöjen mukana kuljetetaan kaikki auktorisointiin tarvittava tieto, kuten käyttäjän roolit tai käyttöoikeudet. Mikropalvelu tarkistaa oikeudet suoraan pyynnöstä ja päättää, sallitaanko pyynnön suoritus. Näin vältetään ylimääräiset auktorisointipyynnöt mikropalveluiden ja auktorisointipalvelun välillä, eikä auktorisointimoduuli ole pullonkaulana koko järjestelmän toimivuudelle. Jos käyttäjä on ehtinyt kirjautumaan ennen auktorisointipalvelun jumiutumista, pystyy käyttäjä yhä käyttämään muita järjestelmän toiminnallisuuksia, sillä pyyntöjä ei tarvitse auktorisoida auktorisointipalvelun kautta. Mikropalvelukohtaista

hajautettua pääsynhallintaa käytetään yleensä token-pohjaisten auktorisointiratkaisujen kanssa, joissa esimerkiksi JWT-tunnisteen avulla kuljetetaan pyynnöissä tieto käyttäjän oikeuksista tai rooleista. Hajautettua auktorisointiratkaisua suosittelevat sekä Reinicke [70] ja Aggarval [71]. Kuvassa 14 on havainnollistettu JWT-pohjaista ratkaisua, jossa mikropalvelut auktorisoivat paikallisesti. [17][70]

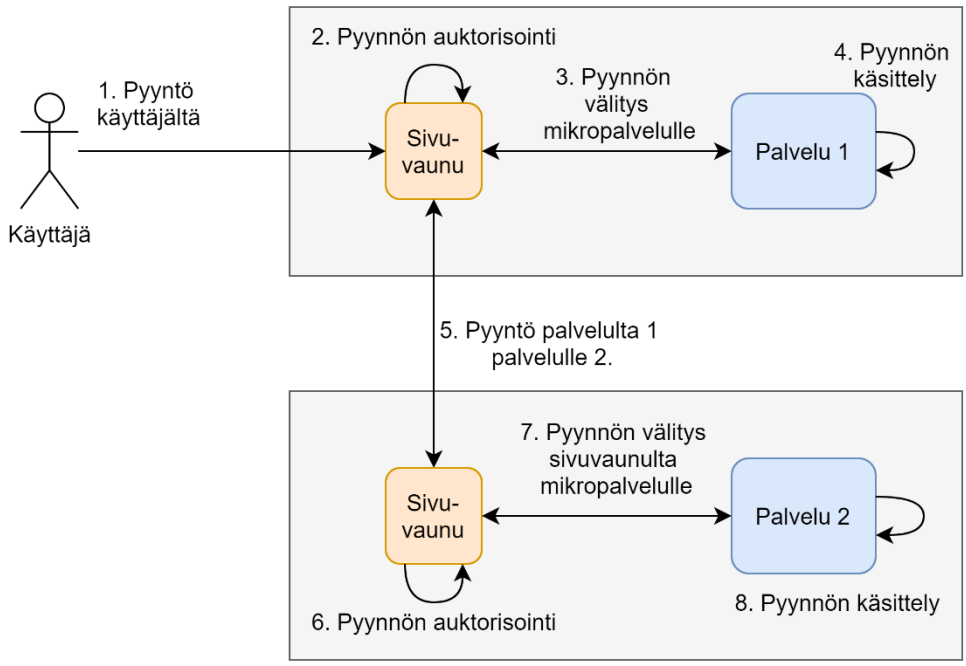


Kuva 14. *JWT-pohjainen auktorisointi mikropalveluissa. [70][71]*

UNA-projektin auktorisoinnissa käytetään mikropalvelukohtaista pääsynhallintaa, mutta tieto käyttäjän oikeuksista pyydetään erikseen uudella kyselyllä, joka aiheuttaa ylimääräistä liikennettä järjestelmien välillä. Kyseessä on siis hybridiratkaisu keskitetyn ja hajautetun pääsynhallinnan välillä. Jotta UNA-projektissa voidaan välttää ylimääräinen kutsu keskitetylle järjestelmälle, täytyy lisätä käyttäjän oikeudet pyyntöihin mukaan. Tämä voidaan toteuttaa esimerkiksi vaihtamalla OAuth2.0 istuntotunniste JWT-pohjaiseen istuntotunnisteeseen.

Kolmas ratkaisu palvelutason auktorisointiin on implementoida kunkin mikropalvelun rinnalle toinen turvallisuudesta ja auktorisoinnista vastaava ”sivuvaunu”-mikropalvelu. Sivuvaunut muodostavat palveluverkon, jota käsiteltiin aiemmin luvussa 4.2. Mikropalvelu ohjaa sille saapuneet pyynnöt ensin ”sivuvaunu”-mikropalvelulle, jossa pyynnön oikeudet tarkastetaan. Sivuvaunu ja varsinainen mikropalvelu kommunikoivat etäyhteydellä, mutta ovat kuitenkin samassa fyysisessä koneessa tai virtuaalikoneessa. Näin ollen liikenne sivuvaunun ja mikropalvelun välillä ei kulje muun julkisen tai yksityisen verkon läpi. Kubernetesta käytettäessä sivuvaunu sijoitetaan varsinaisen mikropalvelun kanssa samaan Kubernetes-podiin [64]. Yhden vastuun periaate ei rikkoudu, kun kaikki auktorisointiin ja autentikointiin liittyvä tapahtuu sivuvaunussa, joten mikropalveluihin ei tarvitse

toteuttaa autentikointi- ja auktorisointitoiminnallisuuksia. Jokainen mikropalvelu tarkastelee auktorisoinnin aikana eri käyttöoikeuksia, joten jokaiselle sivuvaunulle konfiguroidaan mikropalvelun tarvitsemat auktorisointisäännöt. Säännöt voidaan konfiguroida joko manuaalisesti kun mikropalvelu otetaan käyttöön, tai ne voidaan hakea toisesta palvelusta, joka on keskitetty auktorisointisääntöjen hallitsemiseen [59, s. 5]. Samaa sivuvaunutoteutusta voidaan käyttää kaikkien järjestelmässä toimivien mikropalveluiden kanssa, eikä auktorisointiin liittyvää koodia tarvitse ylläpitää muualla kuin yhdessä paikassa. Auktorisointi sivuvaunussa voidaan toteuttaa käyttäen esimerkiksi JWT- tai OAuth-tunnisteita. Tunniste kulkee pyyntöjen mukana, ja auktorisointi tehdään tunnisteen tietojen sekä sivuvaunuun konfiguroitujen sääntöjen perusteella. Sivuvaunumallia on havainnollistettu kuvassa 15. [17, luku 11][66]



Kuva 15. *Esimerkki auktorisoinnin toteuttamisesta sivuvaunuissa. [66][65]*

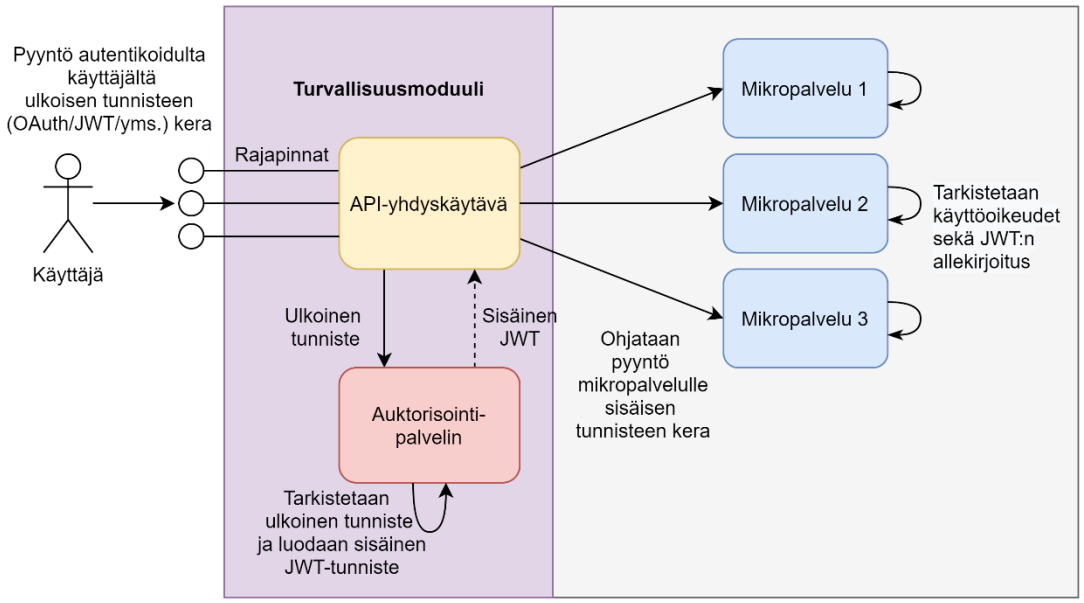
Sivuvaunuratkaisu on hyvä mikropalveluiden autentikointi- ja auktorisointiratkaisu, mutta sen lisääminen jälkikäteen UNA Ytimeen vaatii jonkin verran työtä. Palveluverkon toteuttaminen Ytimessä vaatii muutoksia kaikkiin mikropalveluihin siten, että niihin rakennettuja auktorisointitoiminnallisuuksia siirretään uuteen ”sivuvaunu”-mikropalveluun. Sivuvaunu-toteutuksen kanssa uusien auktorisointisääntöjen ja muiden auktorisointimuutosten tekeminen on jatkossa selkeämpää ja helpompaa.

4.5 Ulkoisen ja sisäisen tunnisteen erottelu

Ulkoisen ja sisäisen tunnisteen erottelussa on kyse siitä, että käyttäjälle luodaan ensimmäisen autentikoinnin jälkeen järjestelmään sisäinen istunto ja ulkoinen tunniste. Ulkoinen tunniste ja järjestelmään tallennettu sisäinen identiteetti ovat linkitetty keskenään. Ulkoista tunnistetta käyttäen käyttäjä voi kutsua rajapintoja, joissa tunniste vaihdetaan järjestelmän sisäisesti tallennettuun identiteettiin. Sisäinen identiteetti välitetään järjestelmän sisäisille mikropalveluille autentikointia ja auktorisointia varten. Ulkoisen tunnisteen vaihtaminen sisäiseen tunnisteeseen tarkoittaa periaatteessa kutsun autentikointia reunatasolla, mutta auktorisointi tapahtuu edelleen mikropalvelussa. Tunnisteen vaihtamisessa tarvittavat auktorisointitiedot lisätään autentikoituun kutsuun, jotta kohteena oleva mikropalvelu pystyy auktorisoimaan kutsun. Auktorisointitietojen välittyminen järjestelmän ulkopuolelle vältetään, joka vaikeuttaa järjestelmään murtautumista. Jos tunnisteiden erottelun sijaan käytetään samaa tunnistetietoa järjestelmän sisä- ja ulkopuolella, on sisäisiä mikropalveluita kohtaan hyökkääminen helpompaa lähettämällä erilaisia tunnisteita suoraan mikropalveluiden rajapintoihin. [59]

He ja Yang tutkivat mikropalveluarkkitehtuuriin sopivia autentikointi- ja auktorisointikeinoja [72]. He toivat tutkimuksessa esiin monoliittisen järjestelmän autentikointia sekä mikropalveluihin sopivia istuntoon ja tunnisteisiin perustuvia autentikointikeinoja. Tutkimuksessa esitetään ratkaisu tunnisteisiin perustuvan autentikointiin liittyviin ongelmiin kuten uloskirjautumiseen. Luodut tunnisteet ovat voimassa tietyn aikaa. Jotta tunnistepohjaisessa autentikoinnissa voidaan kirjautua ulos järjestelmästä, täytyy järjestelmän pystyä kumoamaan aikaisemmin myönnetyn tunnisteen voimassaolo. Ongelma ratkaistaan luomalla ulkopuolisen ja sisäisen tunnisteen erottelu järjestelmän reunalla API-yhdyskäytävässä. Käyttäjä saa kirjautuessaan ulkoisen tunnisteen, joka vaihdetaan API-yhdyskäytävän kohdalla varsinaiseen JWT-tunnisteeseen istunto- tai kontekstipalvelun kautta. Istunto- tai kontekstipalvelu linkittää ulkoisen tunnisteen varsinaiseen järjestelmän sisällä käytettävään tunnisteeseen. Sisäinen JWT-tunniste välittyy pyynnön mukana järjestelmään sisältäen kaiken tarpeellisen tiedon käyttäjästä ja sen oikeuksista. Ulkoista tunnistetta hyödyntäen käyttäjä voi lähettää uloskirjautumispyynnön, jolloin istunto- tai kontekstipalvelussa poistetaan linkitys sisäiseen tunnisteeseen kirjaten käyttäjän ulos. Myös aiemmin mainitussa Barabanovin ja Makrushin tutkimuksessa [59] sisäisen ja ulkoisen tunnisteen erottelu on suositeltava tapa toteuttaa tunnistepohjainen autentikointi mikropalveluarkkitehtuurissa. Kuvassa 16 esitetään tämän mallin toimintaa. [59][72]

Banatin ym. [73] tutkimuksessa esitetään edellä esitellyn tapainen autentikointi- ja auktorisointiratkaisu. Heidän ratkaisuaan käytetään terveydenhuollon projektissa, jossa turvallisuus on tärkeää. Kyseisessä ratkaisussa käytetään OAuth-pohjaista kertakirjautumista, jotta jokaiseen mikropalveluun ei tarvitse erikseen tunnistautua. Ulkoisena tunnisteena he käyttävät on OAuth-pääsytunnistetta, joka vaihdetaan käyttöliittymän rajapinnoissa sisäiseen JWT-tunnisteeseen ja välitetään pyynnön mukana mikropalveluille. Mikropalvelut odottavat vain JWT muotoista tunnistetta, joten OAuth-tunnisteella ei järjestelmään pääse ulkoa käsiksi. JWT-tunnisteen rakenne ei ole selvitettävissä ulkoapäin, sillä se ei ole näkyvissä käyttäjälle. Heidän ratkaisussaan JWT sisältää käyttäjätunnuksen, sähköpostiosoitteen, tunnisteen erääntymisajan sekä erilaisia järjestelmätietoja. [73]



Kuva 16. *Ulkoiseen ja sisäiseen tunnisteeseen perustuva mikropalveluiden pääsynhallinta. [59][72][73]*

Ulkoisen ja sisäisen tunnisteen erottelu järjestelmän reunalla vaikuttaa olevan hyvä ratkaisu mikropalveluiden autentikointitoteutuksissa. Siinä käyttäjien käyttöoikeudet kulkevat vain järjestelmän sisällä mikropalveluiden välillä, eikä sisäinen tunniste paljastu käyttäjälle ollenkaan. Tyypillisesti tunnistepohjaisessa autentikoinnissa tunnisteet ovat voimassa luontihetkestä alkaen määritetyn ajan, eikä tunnisteita voi mitätöidä voimassaoloaikana. Ulkoisen ja sisäisen tunnisteen erottelulla pystytään luomaan istuntopohjaista autentikointia, jota yleensä monoliittisissa järjestelmissä käytetään. UNA Ytimessä on myös käytössä istuntoihin perustuva autentikointijärjestelmä, ja vaihtamalla tällaiseen ratkaisuun voidaan käyttöoikeuksia kuljettaa kutsujen mukana JWT-tunnisteissa, ja samalla välttää ylimääräiset istuntojen tarkistamispyynnöt jokaisesta mikropalvelusta.

4.6 Ratkaisujen arviointi

Yksi tämän työn tutkimusaiheista on selvittää mikropalveluihin sopivimpia autentikoinnin ja auktorisoinnin toteutusmalleja. Aiemmin luvussa esiteltiin useita erilaisia ratkaisuja ja malleja. Näistä useimmat sopivat mikropalveluarkkitehtuuriin, mutta osa niistä on puutteellisia. Jotta voidaan arvioida ratkaisujen sopivuutta mikropalveluarkkitehtuuriin, tarkastellaan sitä, kuinka nämä auktorisointi- ja autentikointimallit toteuttavat mikropalveluiden periaatteita. Mikropalveluiden suunnitteluperiaatteisiin ja tunnusmerkkeihin kuuluvat autonomisuus, löyhät kytkökset, uudelleenkäytettävyys, koostettavuus, skaalautuvuus ja hyvä virheensietokyky. Mikropalveluissa noudatetaan myös periaatteita kuten yhden vastuun periaate sekä ”design-for-failure”- ja ”defence-in-depth” -periaatteet. Autentikointi- ja auktorisointiratkaisujen tulee soveltua näihin periaatteisiin.

Mikropalveluissa tapahtuvaa autentikointia voidaan jakaa kahteen luokkaan: käyttäjien autentikointiin ja mikropalveluiden väliseen autentikointiin. Useiden lähteiden mukaan [59][57, s. 14][72][73] kertakirjautumisratkaisut ovat suositeltavia käyttäjän autentikointiin mikropalveluissa, koska järjestelmän käytettävyys kärsii, jos jokaiseen mikropalveluun tulee kirjautua erikseen. OAuth, OIDC, SAML ja JWT teknologiat soveltuvat kaikki mikropalveluiden kertakirjautumismenettelyn toteuttamiseen.

Palveluiden väliseen autentikointiin löytyy kirjallisuudesta useita ratkaisuja, kuten TLS-salaus, API-avaimet, JWT-tunnisteet, yksityiset verkot sekä palveluverkko-malli. Näistä vaihtoehdoista JWT-tunnisteet on todettu useassa lähteessä hyväksi ratkaisuksi [17, luku 11][12][72]. JWT-tunnisteet ovat toimiva ratkaisu epäluotettavan verkon kautta, koska JWT-allekirjoituksia ei voi väärentää ilman siinä käytettyä salausavainta tai varmennetta. JWT-tunnisteita käytettäessä on suositeltavaa salata liikenne TLS-yhteyden avulla, jotta hyökkääjien on vaikeampaa päästä perille järjestelmässä käytetyistä ratkaisuista [12]. Palveluverkko-malli mahdollistaa yhden vastuun periaatteen noudattamisen hyvin, koska kommunikointiin liittyvä logiikka voidaan siirtää mikropalveluista pois erillisiin sivuvaunuihin. Autentikointikeinoja voidaan käyttää keskenään yhdessä, joka on erityisesti ”defence-in-depth”-periaatteen mukaista.

Mikropalveluarkkitehtuurissa auktorisointi ratkaisut pystyttiin luokittelemaan reunatasolla ja palvelutasolla tapahtuvaan auktorisointiin. Reunatason auktorisointiin löytyi kirjallisuudesta muutama keskenään samankaltainen ratkaisu, joissa järjestelmään saapuvat pyynnöt auktorisoitiin API-yhdyskäytävässä. API-yhdyskäytävä on kytketty autentikointi- ja auktorisointipalveluun, jonka kautta pyynnössä välitettyä tietoa hyödyntäen voidaan tarkistaa, onko käyttäjä kirjautunut tai riittävätkö käyttäjän oikeudet pyynnön suorittamiseen. Reunatason mallissa on tärkeää, että kaikki sisäänpäin menevä liikenne kulkee

järjestelmään API-yhdyskäytävän kautta, jotta pyyntöjen autentikointi ja auktorisointi varmasti tapahtuu. Kyseinen malli ei kuitenkaan riko mikropalveluarkkitehtuurin periaatteita, sillä API-yhdyskäytävä on vain yksi omaan tehtäväänsä keskittynyt mikropalvelu.

Auktorisointia voi tapahtua reunatason lisäksi mikropalvelussa eli palvelutasolla. Tällaiseen auktorisointiin löytyi useita ratkaisuja, joista yksi on käyttää keskitettyä auktorisointipalvelua. Keskitetyn auktorisointipalvelun huonona puolena on erityisesti se, että siinä osittain rikotaan mikropalveluarkkitehtuurin suunnitteluperiaatetta palveluiden autonomisuudesta sekä ”heikkojen kytkentöjen” -periaatetta. Yhtäkään auktorisointia tarvitsevaa mikropalvelua ei voida käyttää, jos auktorisointipalvelu menee jumiin tai siihen ei saada yhteyttä. Mikropalvelut eivät saa ”heikkojen kytkentöjen” -periaatteen mukaan olla kytketty siten, että ne ovat riippuvaisia toisistaan. Lisäksi keskitetty malli voi heikentää järjestelmän suorituskykyä, jos suuri määrä käyttäjiä kasvattaa pyyntöjen määrää yhdelle auktorisointipalvelulle. Kaikkien auktorisointipyyntöjen prosessoiminen yhdessä mikropalvelussa saattaa aiheuttaa viivettä, joka hidastaa koko järjestelmän toimintaa. Ongelman pystyy mahdollisesti kiertämään skaalaamalla auktorisointipalveluiden lukumäärää suuremmaksi ja jakamalla kuormaa näiden välillä.

Kuitenkin Chandramoulin mukaan [12, s. 17] keskitetty auktorisointipalvelu on välttämätön, jotta auktorisointisääntöjen hallinnointi ja välittäminen suurelle määrälle mikropalveluita on mahdollista. Chandramoul tarkoittaa tässä erilaista mallia, jossa mikropalveluiden käyttöönotto- tai käynnistysvaiheessa haetaan auktorisointisäännöt keskitetyltä auktorisointipalvelulta, jonka jälkeen säännöt talletetaan mikropalvelun muistiin. Malli vaatii myös sen, että pyyntöjen mukana välitetään käyttäjän käyttöoikeudet. Kun nämä ehdot toteutuvat voidaan auktorisoida aina mikropalvelussa, eli kyseessä on hajautettu auktorisointimalli.

Hajautetussa auktorisointimallissa auktorisointi mikropalveluissa tapahtuu ilman tarvetta lähettää erillisiä pyyntöjä keskitetylle auktorisointipalvelulle. Mallia noudattaessa mikropalveluiden autonomisuus ja löyhä kytkentä säilyvät, eikä järjestelmään synny niin sanottua ”single-point of failure” pistettä. JWT-tunnisteen käyttö on hajautetussa mallissa usean lähteen mukaan suositeltavaa [17, luku 11][70][71]. JWT-tunnisteissa allekirjoituksen avulla pyyntö autentikoidaan, ja mukana välitettävien käyttöoikeuksien avulla pyyntö auktorisoidaan.

Palvelutason auktorisointi voidaan toteuttaa myös palveluverkko-mallia käyttäen. Tässä mallissa mikropalvelun ohelle lisättäisiin erillinen ”sivuvaunu”-mikropalvelu, joka auktorisoi varsinaiselle mikropalvelulle saapuvat pyynnöt. Palveluverkko noudattaa hyvin mik-

ropalveluarkkitehtuurin periaatteita. Yhden vastuun periaate toteutuu hyvin, koska auktorisointilogiikka voidaan jättää täysin sivuvaunun vastuulle. Lisäksi palveluverkko-malli tukee hyvin uudelleenkäytettävyyden periaatetta, sillä samaa sivuvaunu toteutusta voi käyttää kaikkien mikropalveluiden kanssa. Palveluverkkoon voidaan tehdä myös automaattista kuormantasaustoiminnallisuutta, joka lisää järjestelmän virheensietokykyä.

Ulkoisen ja sisäisen tunnisteen erottelu on eräs kirjallisuudesta esiin tullut auktorisointi- ja autentikointimalli, jossa reunatasolla voidaan eristää järjestelmän sisäinen tunniste ulkoisesta tunnisteesta. Se on yhdistelmä reuna- ja palvelutason auktorisointimenetelmiä. Mallissa hyökkääjien on vaikea päästä mikropalveluihin käsiksi, koska sisäisen tunnisteen rakennetta ei paljasteta käyttäjälle, ja se kulkee vain mikropalveluiden välisissä pyynnöissä. Ulkoisen ja sisäisen tunnisteen erottelu toimii hyvin mikropalveluarkkitehtuurissa, sillä se itsessään ei riko mikropalveluarkkitehtuurin periaatteita.

Luvussa 3.4 perehdyttiin erilaisiin pääsynhallintamalleihin. Oikean mallin valinta riippuu järjestelmän käyttötarpeista. Jos järjestelmällä on useita käyttäjiä, jotka tarvitsevat erilaisia käyttöoikeuksia, ovat molemmat roolipohjainen pääsynhallinta ja attribuuttipohjainen pääsynhallinta hyviä malleja käyttäjän auktorisoinnin toteuttamiseksi. Myös Pereira-Vale ym. [57, s. 14] mukaan nämä mallit ovat yleisiä mikropalveluissa. Attribuuttipohjainen malli on hieman joustavampi, ja sitä pystytään hyödyntämään myös mikropalveluiden keskinäisessä autentikoinnissa ja auktorisoinnissa, sillä attribuuttimallissa voidaan huomioida ympäristöstä ilmeneviä muuttujia auktorisoinnissa.

5. AUTENTIKOINTI JA AUKTORISOINTI UNA YTIMESSÄ

Tässä luvussa esitellään UNA-projektissa aiemmin toteutettua autentikointi- ja auktorisointiratkaisua ja sen arkkitehtuuria. Aluksi käsitellään UNA Ydintä, jonka jälkeen perehdytään Ytimen arkkitehtuuriin. Lopuksi luvussa syvennytään Ytimen autentikointi- ja auktorisointiratkaisuihin.

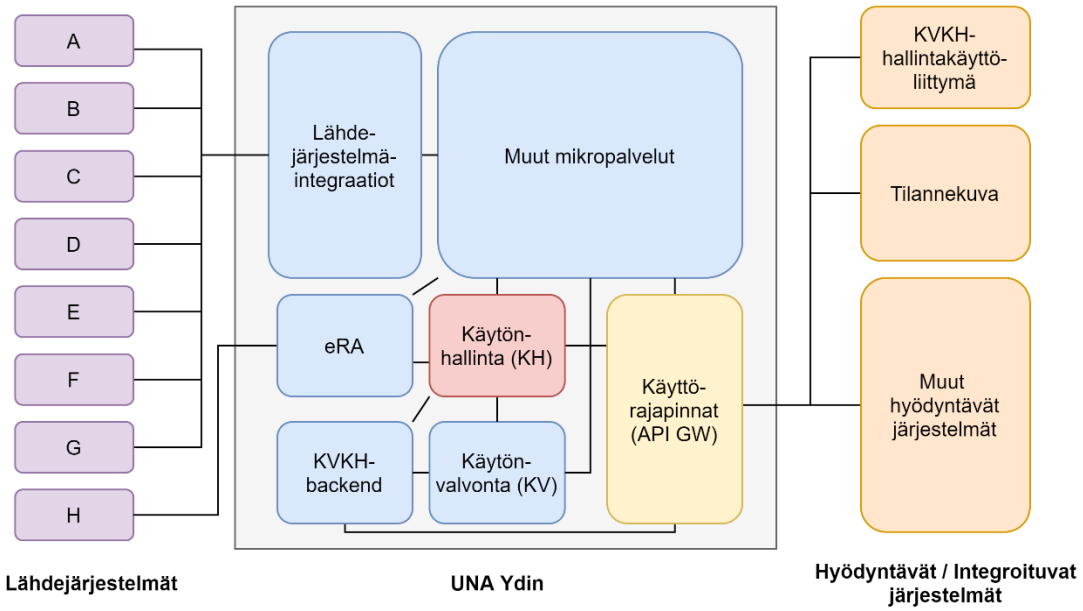
5.1 Yleiskuvaus

UNA Ydin, (jatkossa lyhyesti ”Ydin”) on integraatio- ja tiedonhallintaratkaisu, jonka ympärille on mahdollista rakentaa integraatioita ja uusia sovelluksia [74]. Ytimen kolme päätoiminnallisuutta ovat mahdollistaa tiedon siirtyminen ulkoisista lähdejärjestelmistä Ytimeen, siirretyn tiedon koostaminen ja koostetun tiedon julkaisu Ydintä hyödyntäville tahoille [5]. Ydin hoitaa integraatiot useisiin lähdejärjestelmiin, joissa sosiaali- ja terveydenhuollon asiakkaiden tiedot ovat jakautuneena. Kun sote-ammattilainen tarvitsee asiakkaan tietoja tämän palvelua varten, haetaan Ytimen lähdejärjestelmistä kyseisen asiakkaan tiedot ja kootaan ne yhteen. Tietojen hakemisessa on pyritty viiveettömyyteen ja tietosisältöjen yhdenmukaistamiseen. Tiedot ovat käytettävissä vain sen aikaa kuin sote-ammattilaisella on niihin tarvetta, jonka jälkeen ne poistetaan UNA Ytimen muistista. Jos samaa asiakasta palvellaan myöhemmin uudelleen, tehdään uusi tietojen haku lähdejärjestelmistä. Potilaiden tietoja ei siis säilötä Ytimessä pitkäaikaisesti. Käyttäjien autentikointi Ytimessä tapahtuu käyttäen sote-ammattilaisen omaa terveydenhuollon toimikorttia. Käyttäjät auktorisoidaan UNA Ytimeen toteutetun roolipohjaisen pääsynhallinnan perusteella. Ytimen käyttäjille eli sote-ammattilaisille Ytimestä näytetyt tiedot määrittyvät käyttäjän roolin, ammattioikeuksien ja käyttökontekstin perusteella. [5][74][75]

Sote-ammattilaisille käytettäväksi on toteutettu UNA Ydintä hyödyntävä käyttöliittymä Tilannekuva, joka integroituu Ytimen rajapintoihin. Tilannekuvasta sote-ammattilaiset voivat suorittaa tietojen haun, hahmottaen asiakkaan kokonaistilanteen nopeasti. Ammattilaisille näytettävissä tiedoissa huomioidaan myös tietojen saamiseen liittyvät estot ja kiellot, joita asiakkaat pystyvät halutessaan asettamaan. Eri toimittajat voivat toteuttaa oman käyttöliittymän tai sovelluksen käyttäen Ytimen rajapintoja. [75]

5.2 Arkkitehtuuri

UNA Ydin on laaja useista eri osista koostuva järjestelmä. Kaikkia Ytimen osia ei ole tarpeellista käydä läpi yksityiskohtaisesti tässä työssä, sillä työn aiheena on perehtyä autentikointiin ja auktorisointiin. Ytimen arkkitehtuuriksi on valittu mikropalveluarkkitehtuuri. Mikropalveluarkkitehtuurilla mahdollistetaan laajan järjestelmän kehittäminen nopeasti, sillä kullekin mikropalvelulle voidaan määrätä oma kehitystiimi. Mikropalveluiden ei tarvitse olla toteutettu samoilla teknologioilla, joten jokainen tiimi voi valita itse sopivimmat kehitysteknologiat. Luvussa 2 perehdytään mikropalveluarkkitehtuuriin ja sen hyötyihin tarkemmin. Jokainen Ytimen toiminnallisuus on toteutettu omana mikropalvelunaan mikropalveluarkkitehtuurin mukaisesti. Tärkeimmät palvelut tämän diplomityön kannalta ovat eRA, Käytönhallinta (KH), Käytönvalvonta (KV), KVKH-hallintakäyttöliittymä. eRA ei ole mikropalvelu, vaan se on Atostekin kehittämä monoliittinen ohjelma. eRA-palvelua on hyödynnetty muun muassa KH-mikropalvelussa kirjautumiseen ja käyttäjänhallintaan liittyvillä integraatioilla. Järjestelmään kuuluu myös useita muita mikropalveluita, kuten Käyttötarvepalvelu ja Suostumuspalvelu, mutta ne eivät ole tämän työn kannalta tärkeitä. Kuvassa 17 hahmotellaan UNA Ytimen arkkitehtuuria tämän diplomityön näkökulmasta.



Kuva 17. *Hahmotelma Ytimen arkkitehtuurista. [5][76]*

5.2.1 eRA

UNA Ytimen käyttäjänhallintaa ja autentikointia varten on hyödynnetty Atostekin kehittämää eRA-palvelua ja eRA SmartCard-kortinlukijaohjelmistoa. eRA on palvelu, joka tarjoaa terveydenhuollon ja sosiaalihuollon toimijoille tavan liittyä helposti Kanta-palveluiden käyttäjäksi. Kanta-palvelut ovat Kansaneläkelaitoksen eli Kelan kehittämiä ja ylläpitämiä palveluita, joiden kautta voidaan muun muassa siirtää potilas- ja asiakastietoja sosiaali- ja terveydenhuollon, apteekkien ja kansalaisten välillä [77]. eRAn kautta pystyy muun muassa kirjoittamaan sähköisiä lääkemääräyksiä, hakemaan lääkärintodistuksia ja -lausuntoja, sekä kirjaamaan ja katselemaan tietoja Kelan Kanta-palvelun potilastiedon ja sosiaalihuollon asiakastiedon arkistoista. eRA SmartCard on ohjelmisto, jonka avulla sote-ammattilaiset voivat kirjautua omalla älykortillansa käyttämään eRA-palvelua. [78]

Käyttäjänhallinnan ja älykortilla autentikoinnin lisäksi Ydin hyödyntää eRA-palvelussa valmiiksi saatavilla olevaa Kanta-palveluiden integraatiota, joka täytyy ilman eRA-integraatiota kehittää erikseen UNA Ytimeen. Ytimessä eRAn tarjoamien autentikointi ja käyttäjänhallintaominaisuuksien käyttö on toteutettu KH-mikropalvelussa.

5.2.2 Käytönhallinta

Käytönhallinta-mikropalvelu toteuttaa UNA Ytimen autentikointiin ja auktorisointiin liittyvät ratkaisut. KH-moduuli sisältää oman tietokannan, johon käyttäjien istunnot, käyttöoikeudet, roolit, varmenteet ja muut auktorisointiin liittyvät tiedot tallennetaan. Koska eRA-rajapinnat mahdollistavat toimikorttiautentikoinnin ja käyttäjänhallinta toiminnallisuudet, on päätetty käyttää eRA-palvelun tarjoamia toteutuksia sen sijaan, että ne toteutettaisiin erikseen KH-mikropalveluun. Integroitumalla eRA-palvelun rajapintoihin voidaan tarjota nämä toiminnallisuudet KH-moduulin kautta vähemmällä työmäärällä.

Käytönhallinta on toteutettu roolipohjaisella mallilla, ja kirjautuessa käyttäjälle luodaan istunto KH-moduuliin. Ytimen pyyntöjen auktorisointioikeudet tarkistetaan KH-mikropalvelusta, joten KH on keskeinen osa koko Ytimen toimintaa. KH-moduulin kirjauduttaessa luodaan samalla eRA-istunto. Koska eRA tarjoaa valmiiksi mahdollisuuden tallentaa käyttäjän identiteettitiedot turvallisesti, ei KH-moduuliin tarvitse tallentaa arkaluontoisia tietoja käyttäjästä, kuten henkilötunnusta. KH-moduulin istuntojen tietoihin tallennetaan sen sijaan eRA-istunnon tunniste. Kun Ytimessä tarvitaan kirjautuneen käyttäjän tietoja, ne pyydetään KH-moduulilta, joka puolestaan hakee tiedot eRA-istuntotunnisteella eRAn

rajapinnasta. Samaa menettelyä käytetään, kun Ytimeen kirjaudutaan toimikortilla. Kirjautuminen aloitetaan KH-moduulin rajapinnasta, mutta taustalla käytetään eRAn ja eRA SmartCard ohjelmistojen rajapintoja.

5.2.3 Käytönvalvonta

Ytimen yhtenä osana on mikropalvelu nimeltään Käytönvalvonta. KV-mikropalvelun tarkoitus on toteuttaa lokien tallettamiseen, hakemiseen ja käsittelyyn liittyvät toiminnot. Ytimessä on useita erilaisia lokeja, jotka KV-moduuliin tallennetaan. Näitä ovat käyttö-, muutos- ja ylläpitolokitiedot. Lisäksi järjestelmässä on tekninen loki jokaisesta mikropalvelusta, mutta sitä ei tallenneta KV-mikropalveluun. KV-mikropalvelu sisältää oman tietokannan lokimerkintöjen tallentamista varten. Lokien hakeminen KV-moduulin rajapinnoista on suojattu käyttöoikeuksilla, jotka ovat myönnetty vain muutamille rooleille. Muut mikropalvelut lähettävät uusia lokitietoja KV-moduulille, kun toimintoja suoritetaan Ytimessä. Näin saadaan kattava loki käyttäjien toiminnasta Ytimessä. Käyttölokin avulla pystytään selvittämään, kuka järjestelmää on käyttänyt, ja mitä hän on tehnyt. Käyttöloki on tärkeä ja lain mukaan pakollinen [79] osa terveydenhuollon järjestelmissä, jotta esimerkiksi tietomurto- tai väärinkäyttötapauksessa voidaan selvittää se, mitä tietoja henkilöt ovat hakeneet tai nähneet käyttäessään järjestelmää.

5.2.4 KVKH-käyttöliittymä

KH- ja KV-mikropalvelut ovat taustajärjestelmä-mikropalveluita (engl. backend service). Niiden hallintaa varten on toteutettu selainpohjainen KVKH-käyttöliittymä. KVKH-käyttöliittymästä voidaan muokata rooleja ja oikeuksia, sekä katsella lokimerkintöjä eri lokeista. KVKH-käyttöliittymä on tarkoitettu pääasiassa UNA:n ja muiden organisaatioiden ylläpitohenkilöiden käytettäväksi. Käyttöliittymässä UNA Ytimen ylläpitäjät pystyvät muokkaamaan UNA-rooleja ja organisaatioiden ylläpitäjät pystyvät määrittämään oman organisaationsa käyttämät organisaatioroolit.

Käyttöliittymälle on toteutettu oma mikropalvelu sen taustajärjestelmäksi, jonka kautta käyttöliittymä toimitetaan käyttäjille selaimessa. Lisäksi tämä backend-palvelu tarjoaa rajapinnat käyttöliittymän toiminnoille. Taustajärjestelmän kautta käyttöliittymästä lähetetyt pyynnöt ohjataan Ytimen eri mikropalveluille. Järjestelyllä Ytimen mikropalveluiden rajapinnat voidaan pitää turvassa asiaankuulumattomilta henkilöiltä, sillä nämä käyttöliittymän hallintarajapinnat ovat käytettävissä vain käyttöliittymän kautta tunnistautuneena.

5.3 Autentikointi- ja auktorisointiratkaisut

5.3.1 Pääsynhallintamalli

Ydintä suunniteltaessa sen auktorisointimalliksi on valittu roolipohjainen RBAC – ABAC hybridiauktorisointimalli, jossa käyttäjille voidaan asettaa eri rooleja RBAC-mallin mukaisesti, mutta lisäksi on käytössä attribuuttimallista 'aika' yhtenä auktorisointitekijänä. Rooleille voidaan määrittää tietyt kellonajat ja viikonpäivät, jolloin roolit ovat voimassa. KH-moduulin roolit koostuvat käyttöoikeuksista, ja käyttöoikeudet koostuvat operaatioista kohteisiin. Esimerkiksi roolille voidaan luoda käyttölokin lukua varten käyttöoikeus, joka koostuu luku operaatiosta, ja niiden kohteena on käyttöloki. RBAC mallia on käsitelty tarkemmin luvussa 3.4.2. Ytimessä on UNA-rooleja, jotka määrittävät käyttöoikeudet. Lisäksi voidaan määrittää organisaatiokohtaisia organisaatiorooleja, jotka käyttävät valmiita UNA-rooleja pohjanaan. Organisaatioroolien oikeuksia voidaan laajentaa tilannekohtaisilla rooleilla, jotka lisäävät pohjalla olevan roolien oikeuksia sisällyttäen organisaatioroolien käyttöoikeuksiin tilanneroolin käyttöoikeudet. Näitä rooleja kutsutaan Ytimessä tilannerooleiksi tai työrooleiksi. Kaikki käytönhallintaan liittyvät roolit ja niiden käyttöoikeudet voidaan määrittää KVKH-hallintakäyttöliittymästä, joka tallentaa luodut roolit ja käyttöoikeudet KH-moduuliin. Hallintakäyttöliittymässä käyttäjä auktorisoidaan, ja käyttöliittymän eri näkymiä näytetään käyttäjälle sen mukaan mitä oikeuksia kyseisellä käyttäjällä on.

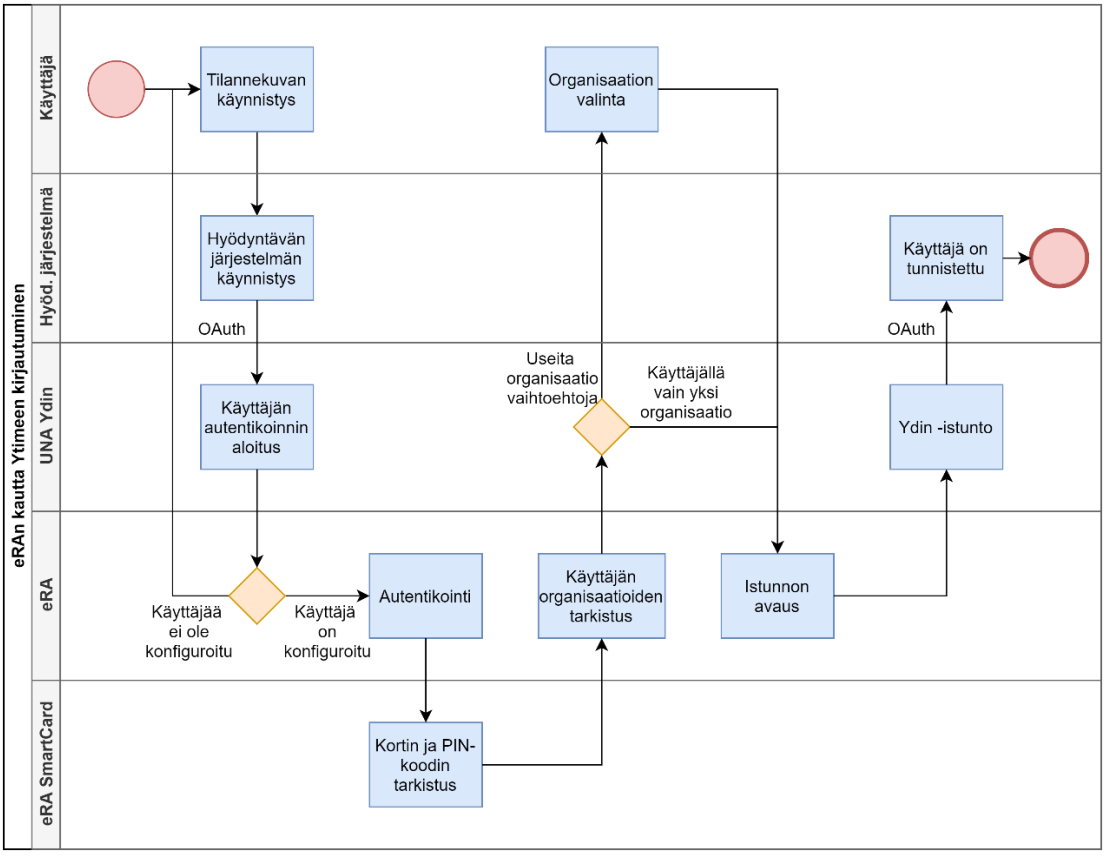
Koska myös eRA on osa Ydintä, eikä sen toimintaa voida UNA Ytimen käyttöoikeuksilla ohjata suoraan, on täytynyt tehdä linkitys eRA-roolien ja UNA-roolien välille. Linkitys roolien välille on ollut mahdollista, koska myös eRA-palvelussa käytetään roolipohjaista auktorisointimallia. Roolit eRA-palvelussa määrittävät mitä toimintoja käyttäjä voi eRAn kautta käyttää ja UNA-roolit määrittävät käyttäjän käyttöoikeudet Ytimessä.

5.3.2 Käyttäjän autentikointi

UNA Ytimessä käyttäjän autentikointi tapahtuu KH-mikropalvelussa. Autentikoinnin yhteydessä syntyvät istuntotiedot, kuten istuntotunnisteet ja roolit ovat tallennettu KH-moduulin tietokantaan. Käyttäjien tietoja ei tallenneta KH-moduuliin, koska ne ovat eRA-palvelussa jo tallennettuna.

Käyttäjä voi kirjautua sisään kahdella eri tavalla. Ensimmäinen kirjautumistapa perustuu eRA-palveluun ja toimikortilla kirjautumiseen. Sote-henkilöstöllä on käytössä toimikortti, joka sisältää digitaalisen varmenteen. Toimikorttikirjautumisessa käyttäjä asettaa kortin

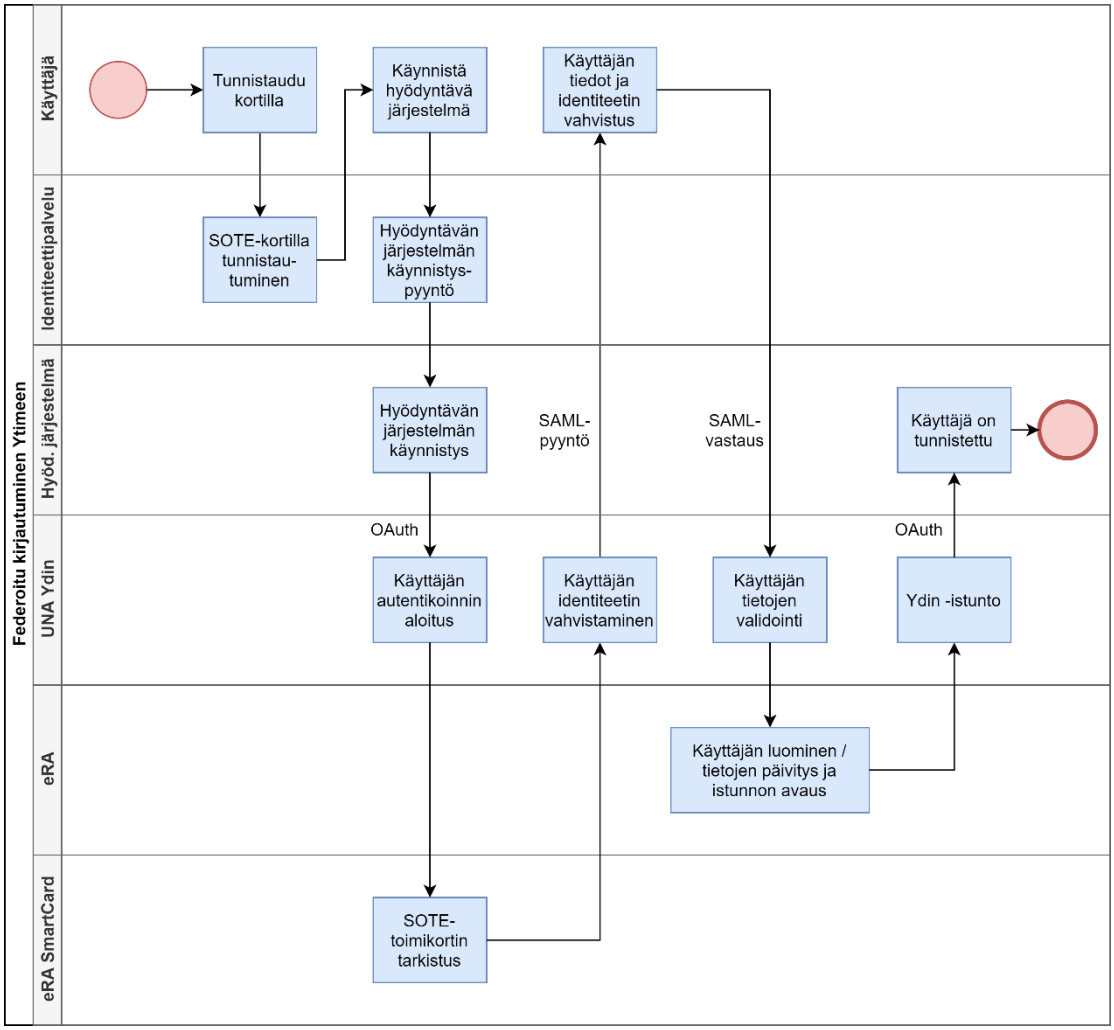
kortinlukijaan, ja sen jälkeen syöttää salasanan tai PIN-koodin. Toimikorttikirjautuminen on kaksivaiheinen tunnistautuminen. Siinä tarkastetaan, että käyttäjällä on oikea kortti, jossa on käyttäjää edustava varmenne, sekä PIN-koodi, jonka vain käyttäjä tietää. Tässä kirjautumistavassa käyttäjien tulee olla ennalta konfiguroituja eRA-järjestelmään; käyttäjien tiedot täytyy lisätä eRA-järjestelmään etukäteen. Kun käyttäjä on lisätty järjestelmään, voi käyttäjä kirjautua. Toimikorttikirjautuminen aloitetaan Ytimen KH-moduulin rajapinnasta, joka taustalla käyttää eRASmartCard-ohjelmistoa ja eRA-palvelun rajapintoja kirjautumisen suorittamiseksi. Kirjautuessaan käyttäjä valitsee toimipisteen, johon hän kirjautuu. KH-moduulin eRA integraation kautta onnistuneen autentikoinnin yhteydessä tarkistetaan, mitä rooleja käyttäjällä on eRA-palvelussa, ja löytyykö vastaavaa UNA-roolia KH-moduulista. Jos käyttäjän roolitiedot ovat kunnossa, luodaan KH-moduulissa käyttäjälle istunto. Samalla luodaan eRA-istunto. Kun istunnot ovat luotu, palautetaan käyttäjälle OAuth2.0-protokollan mukainen valtuutuskoodi. Valtuutuskoodi täytyy vielä vaihtaa KH-moduulin rajapinnassa käyttöoikeuskoodiin kuvan 3 OAuth-sekvenssin mukaisesti. Käyttöoikeuskoodilla tarkoitetaan istuntotunnistetta Ytimessä.



Kuva 18. *Kaavio Ytimen kirjautumisesta eRAn kautta. [76]*

Kuvassa 18 esitetään edellä esitellyn kirjautumistavan prosessia. Kyseisen kirjautumistavan tarkoitus on olla nopeampi ratkaisu saada toimikorttikirjautuminen käyttöön, mutta huonona puolena on käyttäjien osalta vaadittu konfigurointi eRA-palveluun.

Pääasiassa käyttäjät tulevat kirjautumaan toisella kirjautumiskeinolla, joka on federoitu kirjautuminen. Käyttäjän identiteetti haetaan UNA Ytimen ulkopuoleisesta järjestelmästä SAML-pohjaista kertakirjautumista hyödyntäen. Ulkoiset järjestelmät tässä tapauksessa ovat sote-alueiden omia käyttäjänhallintajärjestelmiä, jotka tukevat entuudestaan SAML-protokollaa. Käyttäjän kirjautuminen aloitetaan palveluntuottajan eli UNA Ytimen KH-moduulin kautta, jonka jälkeen käyttäjä ohjataan SAML-protokollaa noudattaen ulkoiseen identiteettipalveluun. Kyseessä on palveluntuottajan aloittama SAML-kertakirjautuminen, jota on havainnollistettu luvun 3.2.3 kuvassa 5. Identiteettipalvelussa käyttäjä tunnistautuu Active Directory-tunnuksensa (AD) avulla käyttäen toimikorttia. Käyttäjä saattaa olla jo valmiiksi kirjautunut AD-tunnuksella omalle työkoneellensa, joten tunnistautumista ei tarvitse tehdä uudelleen Ytimeen kirjautuessa. Kun käyttäjä on tunnistettu identiteettipalvelussa, palautetaan selaimen kautta SAML-protokollaa käyttäen tarvittavat käyttäjän tiedot KH-moduulille. KH-moduulissa varmistutaan vastauksen lähettäjästä tarkistamalla SAML-viestin allekirjoitus, jonka jälkeen käyttäjän tiedot viedään eRA-rajapintojen kautta eRA-palveluun. Jos kyseessä on uusi Ytimen käyttäjä, konfiguroidaan samalla rajapintojen kautta uusi käyttäjä eRA-palveluun. Olemassa olevan eRA-käyttäjän tietoja päivitetään identiteettipalvelun palauttamien tietojen mukaan. Kirjautumissekvenssin loppuvaiheessa luodaan uusi istunto sekä eRA-palveluun, että KH-moduuliin, ja palautetaan OAuth2.0-protokollan mukaisesti valtuutuskoodi. Kyseinen valtuutuskoodi vaihdetaan vielä KH-rajapinnan kautta istuntotunnisteeksi, kuten ensimmäisessä kirjautumistavassa. Kuvan 19 kaaviosta näkee kirjautumisprosessin eri vaiheet. Uusia järjestelmiä lisättäessä identiteettipalveluille luodaan omat varmenteet, ja ne tallennetaan KH-moduuliin. Varmenteet annetaan myös identiteettipalveluille käytettäväksi SAML-viestien allekirjoittamista varten. Allekirjoitusten ja varmenteiden avulla voidaan luottaa järjestelmiin, jotka lähettävät SAML-viestejä Ytimelle. Tässä kirjautumistavassa käyttäjän ei tarvitse olla valmiiksi konfiguroituna eRA-palveluun.



Kuva 19. *Kaavio Ytimen federoidusta kirjautumisesta. [76]*

Edellä mainitut kirjautumistavat ovat käyttäjän identiteetin varmistamista varten. Autentikoinnin loppuvaiheessa käyttäjälle myönnetty istuntotunniste on kertakirjautumistunniste, jolla Ytimen rajapintoja käytetään. Istuntotunniste vastaa nimensä mukaisesti käyttäjän istuntoa KH-mikropalvelussa.

KH-moduulin istunnot ovat liitetty eRA-järjestelmän istuntoihin. Jos eRA-istunto lukittuu, sulkeutuu tai vanhenee, tapahtuu sama myös KH-moduulin istunnolle. Istunnon tilan välittäminen on toteutettu käyttäen push-notification menettelyä, jossa eRA lähettää istuntojen muutoksista viestin KH-moduulille ja tämä asettaa oman istunnon samaan tilaan. Istunto voidaan myös lopettaa tai lukita KH-moduulissa, jolloin istunnon tila välitetään KH-moduulista eRA-palveluun. Jos käyttäjä poistaa toimikortin lukijasta, istunto lukittuu automaattisesti. Istunnon lukittuminen voidaan avata syöttämällä kortti takaisin lukijaan ja syöttämällä oikea PIN-koodi. Lukittuneella istunnolla rajapintoja ja Ytimen toimintoja ei voi käyttää.

5.3.3 Mikropalveluiden välinen autentikointi

Ytimen mikropalveluiden väliset rajapinnat ovat toteutettu http-pohjaisina REST-tyylisinä rajapintoina, jotka vaativat voimassa olevan istuntotunnisteen. Istuntotunnisteiden voimassaolo tarkistetaan aina KH-moduulilta. Ytimessä lähes kaikki mikropalveluiden välinen liikenne on seurausta käyttäjän lähettämistä pyynnöistä, joten pelkällä istuntotunnisteella autentikointi on mahdollista. Muissa mikropalveluiden välisissä pyynnöissä luotetaan siihen, että pyynnöt ovat saman verkon sisältä saapuneita, eikä istuntotunnistetta tarvita. Verkon sisältä saapuneisiin pyyntöihin voidaan luottaa, koska mikropalveluiden välinen liikenne on suojattu eristämällä ne ulkomaailmasta käyttäen verkkorakennetta, jossa mikropalvelut ovat yhdessä sisäisessä verkossa, eikä ulkopuolelta ole suoraa yhteyttä mikropalveluihin. Ulkomaailmaan yhteydessä olevien mikropalveluiden rajapinnat vaativat aina autentikoinnin.

KH-mikropalvelun ja eRAn välinen liikenne salataan TLS-varmenteiden avulla. Lisäksi eRAn rajapintaan KH-moduulista lähetettävät kutsut allekirjoitetaan varmenteilla. Allekirjoitukset tarkistamalla eRA tietää, miltä järjestelmältä sille saapuvat pyynnöt tulevat. Varmenteet tallennetaan sekä KH-moduuliin, että eRA-palveluun. Näin luodaan turvallinen yhteys eRAn ja Ytimen väliselle kommunikaatiolle.

5.3.4 Reunatason ratkaisut

Ytimessä käytetään API-yhdyskäytävää, jonka kautta liikenne kulkee järjestelmään sisään. Vain valitut rajapinnat ovat avoinna järjestelmän ulkopuolelta käytettäväksi. Avatut rajapinnat eli käyttörajapinnat suojataan TLS-varmenteilla eli tiedonsiirto tapahtuu salattuna. Lisäksi Ytimen reunalla käyttörajapinnoissa autentikoidaan API-avaimella. API-avaimet ovat asiakaskohtaisia, eli integroituvat järjestelmät saavat oman API-avaimen, jonka he tarvitsevat Ytimen käyttämistä varten. API-avaimen lisäksi järjestelmän käyttäjien täytyy kirjautua sisään luvun 5.3 mukaisesti, ja heillä täytyy olla voimassa oleva UNA-istunto.

Osa Ytimen toiminnoista on suojattu API-yhdyskäytävän avulla käytettäväksi vain tiettyjen IP-osoitteiden kautta. Se on lisäkeino autentikoida ja auktorisoida liikennettä luottamalla siihen, että kyseisistä IP-osoitteista tulevat pyynnöt ovat sallituilta käyttäjiltä peräisin. IP-osoiterajaus on vain yksi lisäkerros turvallisuudessa, joten käyttäjien täytyy osoiterajauksen lisäksi edelleen kirjautua sisään käyttääkseen kyseisiä toimintoja.

5.3.5 Palvelutason ratkaisut

UNA Ytimessä pääsynhallintaa suoritetaan jokaisessa mikropalvelussa. Pääsynhallinnan säännöt ovat koodattu kuhunkin mikropalveluun, ja pyyntöjen auktorisointi tapahtuu tarkastelemalla käyttäjän istuntoon kuuluvia käyttöoikeuksia jokaisessa mikropalvelussa. KH-moduuli sisältää luonnollisesti istuntojen lisäksi tiedon käyttäjien rooleista sekä käyttöoikeuksista. Pyynnön saapuessa mikropalvelulle käyttäjän istunnon käyttöoikeudet ja roolit haetaan KH-mikropalvelusta istuntotunnisteen avulla. Jos istuntotunniste ei ole aktiivinen, käyttöoikeuksia ei voida hakea, joten pyynnön auktorisointi epäonnistuu. Mikropalvelu päättää haettujen käyttöoikeuksien ja mikropalvelun auktorisointisääntöjen perusteella, suoritetaanko pyyntö vai ei. Mikropalveluihin koodattu auktorisointisääntö on esimerkiksi: ”Jos käyttäjällä on ’luku’-oikeus kohteeseen A, salli kohteen A tietojen hakeminen.” Kun käytetään eRA-integraatiota, tarkastellaan eRA-palvelussa käyttäjän roolien oikeuksia samoin kuin Ytimen mikropalveluissa. Esimerkiksi eRAn rajapinnat pystyvät rajoittamaan roolin mukaan sitä, mitä tietoja eRA palauttaa Ytimelle. Käyttäjän tehdessä hakua Ytimen kautta rajoitetaan käyttäjän UNA-roolin perusteella käyttörajapinnoista palautettavia tietoja.

6. VERTAILU JA ARVIOINTI

Aiemmissa luvuissa esiteltiin erilaisia autentikointi- ja auktorisointiratkaisuja, sekä perehdyttiin UNA-projektissa toteutettuun autentikointi- ja auktorisointiratkaisuun. Tässä luvussa arvioidaan UNA-projektissa toteutetun autentikointi- ja auktorisointiratkaisujen sopivuutta käyttötarkoitukseensa käyttäen mikropalveluarkkitehtuurin suunnitteluperiaatteita ja luvun 4 ratkaisuja arvioinnin apuna. Lisäksi luvun 4 ratkaisuista koostetaan yhteen yleisesti mikropalveluihin sopiva autentikointi- ja auktorisointimalli.

6.1 Ytimen auktorisointi- ja autentikointitoteutusten arviointia

Turvallisuutta Ytimessä on toteutettu usealla eri tasolla. Ytimessä on käytetty ulkomaailmasta eristettyä sisäistä verkkoa mikropalveluiden väliselle liikenteelle, josta vain rajatut rajapinnat ovat yhteydessä ulkomaailmaan. Lisäksi on käytetty API-yhdyskäytävää, API-avaimia, TLS-protokollalla suojattuja yhteyksiä, sekä kirjautumisessa käyttäjältä vaaditaan vahvaa tunnistautumista ja mikropalveluiden rajapinnat ovat suojattu OAuth-tekniikan avulla. Näin ollen ”defence-in-depth” -periaatetta on yleisesti noudatettu Ytimessä.

UNA Ytimessä käyttäjän autentikointi on toteutettu luvun 4 suositusten mukaisesti, vaikka Ytimen autentikointimalli on ainutlaatuinen verrattuna useisiin luvun 4 ratkaisuihin. Ytimessä kertakirjautuminen on toteutettu käyttäen OAuth2.0- ja SAML-protokollia sekä eRA-palvelun toimikorttikirjautumista yhdessä. SAML on sopiva valinta kertakirjautumisteknologiaksi, koska useat potilasjärjestelmien käyttämät identiteettipalvelut tukevat jo entuudestaan SAML-protokollaa. Jos kyseessä on erilainen tai uusi järjestelmä, SAML-tekniikan sijaan voidaan käyttää esimerkiksi JWT-pohjaista kertakirjautumista, joka on kevyempi ja helpompi toteuttaa. OAuth-protokollaa on käytetty Ytimessä kertakirjautumisen jälkeisten järjestelmän sisäisten toimintojen käyttämisen mahdollistamiseksi. Ytimen toimikorttikirjautumista varten eRA on hyvä valinta, sillä se on mahdollistanut hyödyllisiä toimintoja, kuten istunnon lukittumisen käyttäjän poistaessa toimikortin lukijasta.

Ytimessä palveluiden välinen autentikointi tapahtuu joko luottamalla verkkoon tai käyttäen istuntotunnistetta. Istuntotunniste kulkee käyttäjän aloittamien pyyntöjen mukana, joten sen avulla voidaan autentikoida myös palveluiden väliset pyynnöt. Osa rajapinnoista on täysin mikropalveluiden keskinäistä kommunikaatiota varten, eikä silloin istuntotunniste kulje pyyntöjen mukana. Tapauksissa luotetaan siihen, että pyynnöt ovat jär-

jestelmän sisäisestä verkosta peräisin. Rajapintoihin voidaan ”defence-in-depth” -periaatteen mukaan tehdä lisäsuojakerros käyttäen esimerkiksi API-avainta tai suositeltua JWT-tunnistetta.

UNA Ytimen pääsynhallintamalliksi on käytetty roolipohjaista mallia, joka sopii hyvin Ytimen käyttötarpeisiin. Jos auktorisointia halutaan tulevaisuudessa hienosäätää vielä tarkemmin, voidaan roolipohjainen malli vaihtaa luvun 3.4.3 mukaiseen attribuuttimalliin. Muutos ei ole kuitenkaan yksinkertaista toteuttaa. Attribuuttimalliin vaihtaminen vaatii muutoksia KH-moduuliin ja mahdollisesti mikropalveluiden auktorisointisääntöihin. Lisäksi sen toteuttaminen vaatii pohdintaa siitä, kuinka eRA-palvelun pääsynhallintamalli ja eRA-roolit linkittyvät attribuuttimalliin. Yksi mahdollisuus toteuttaa attribuuttimalli Ytimeen on käsitellä eRA-rooleja KH-moduulissa käyttäjän attribuutteina. eRA-palveluun ei tarvitse muutoksia, vaan siellä rooleja voidaan käsitellä edelleen roolipohjaisen mallin mukaan.

Mikropalveluiden auktorisointi on Ytimessä toteutettu käyttäen keskitetyn ja hajautetun mallin yhdistelmää. Yhdistelmämallissa mikropalvelut hoitavat auktorisoinnin itse, mutta käyttöoikeuksien tarkastelua varten täytyy mikropalveluista lähettää erillinen tarkistuspyyntö KH-mikropalvelulle. KH tarkistaa, onko käyttäjän istunto edelleen voimassa, ja palauttaa käyttäjän käyttöoikeudet. Keskitetty ratkaisu ei ole mikropalveluarkkitehtuurin periaatteiden mukainen ratkaisu, sillä monien Ytimen mikropalveluiden toiminta on osittain riippuvaista KH-moduulin toiminnasta. Toisaalta auktorisointimoduulit tai -palvelut ovat aina keskeisissä rooleissa järjestelmien toimintaa. Keskitetystä mallista ei välttämättä synny suurta ongelmaa, koska mikropalvelut kuten KH-moduuli ovat helposti skaalattavissa. Kuormaa pystytään jakamaan tarvittaessa useamman KH-mikropalveluinstanssin välillä, ja mahdolliseen vikatilaan päätyneen instanssin pystyy korvaamaan toisella.

Palvelutasolla hajautettu auktorisointiratkaisu on enemmän mikropalveluiden suunnitteluperiaatteiden mukainen kuin edellä mainittu yhdistelmämalli. Se voidaan toteuttaa Ytimeen esimerkiksi käyttäen suositeltua JWT-rakennetta käyttöoikeuksien välittämiseen mikropalveluille. Luvussa 6.3 on käsitelty ideaa tarkemmin.

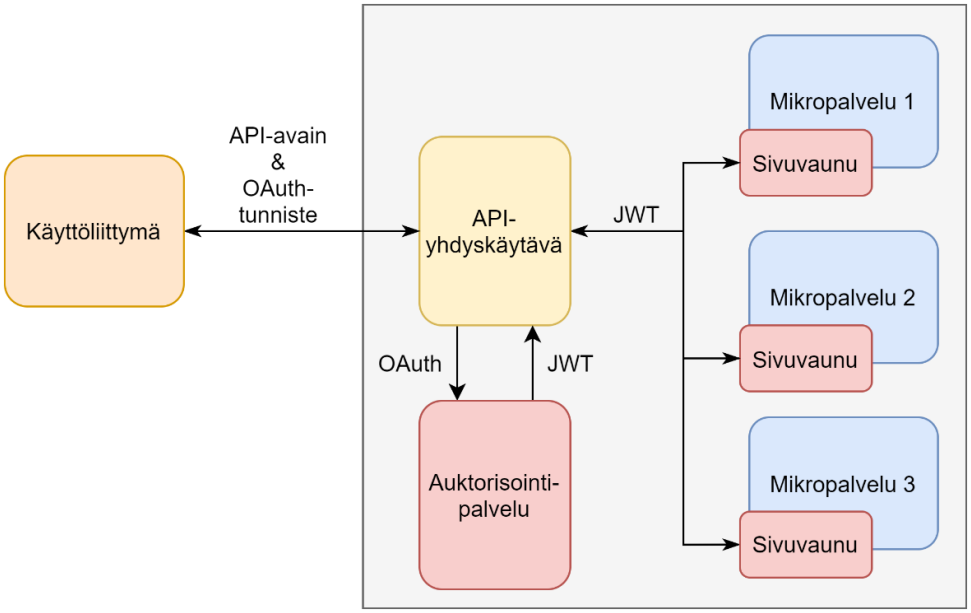
UNA Ytimen reunatason auktorisointiratkaisu ei perustu luvun 4 mukaisesti API-yhdyskäytävän ja auktorisointipalvelun yhteistyössä tapahtuvaan auktorisointiin. Ytimessä käytetään sen sijaan reunatasolla API-avainta autentikointiin ja auktorisointiin, mutta hienosäätöinen auktorisointilogiikka on palvelutasolla. Reunatasolla toimiva auktorisointipalvelu voi toimia myös UNA Ytimessä, mutta se ei nykyisessä järjestelyssä ole välttä-

mätön. Jos Ytimen palvelutason auktorisointia vaihdetaan hajautetun mallin mukaisemmaksi, on reunatason auktorisoinnin toteuttamisessa enemmän järkeä. Reunatason auktorisointipalvelua voi käyttää Ytimessä esimerkiksi luvun 4.5 mukaiseen ulkoisen ja sisäisen tunnisteen erotteluun. Kyseisessä järjestelyssä järjestelmän ulkopuolisena tunnisteena toimii nykyinen Ytimen istuntotunniste ja järjestelmän reunalla tunniste vaihdettaisiin JWT-rakenteeksi, joka kulkee pyyntöjen mukana. JWT-rakenteen avulla voidaan auktorisoida palvelutasolla hajautetun mallin mukaisesti.

6.2 Esimerkkiratkaisu mikropalveluiden autentikointiin ja auktorisointiin

Luvussa 4 käsitellyt mallit ovat osittain erilaisia, mutta useimmat niistä eivät sulje toisiaan pois. Erityisesti eri tasoilla järjestelmän toimintaa käytettäviä auktorisointiratkaisuja pystytään käyttämään yhdessä. Mikropalveluiden toteutuksissa tulee noudattaa defence-in-depth periaatetta, ja yksi tapa noudattaa sitä on toteuttaa auktorisointia usealla tasolla mikropalvelujärjestelmää. Tässä luvussa ehdotetaan mikropalveluiden autentikointiin ja auktorisointiin yleistä esimerkkiratkaisua, jossa yhdistetään useita edellä esiteltyjä malleja turvallisen kokonaisuuden muodostamiseksi. Esimerkki perustuu mikropalveluarkkitehtuurin suunnitteluperiaatteisiin ja aiempien tutkimusten suosituksiin.

Seuraavaksi esiteltävän esimerkkimallin kohteena oleva mikropalvelujärjestelmä koostuu käyttöliittymästä ja useista REST-pohjaisista mikropalveluista. Mikropalveluiden kyljessä käytetään erillistä sivuvaunu-mikropalvelua. Lisäksi mallin järjestelmään kuuluu API-yhdyskäytävä ja auktorisointimikropalvelu. Kuvassa 20 hahmotellaan ratkaisua.



Kuva 20. *Esimerkkiratkaisu autentikoinnin ja auktorisoinnin toteuttamiseen.*

Kirjautuminen järjestelmään toteutetaan jollakin kertakirjautumismenetelmällä riippuen järjestelmän tyypistä. Esimerkkimallin kertakirjautumisteknologiaksi valitaan OAuth2.0, jota on käsitelty luvussa 3.2.1. Lisäksi käyttäjä tunnistautuu vahvasti luvun 3.3 mukaisesti. OAuth-kirjautumisen jälkeen käyttäjä voi lähettää pyyntöjä mikropalvelujärjestelmään lisäämällä OAuth-tunnisteen mukaan kutsuihin. Käyttöliittymä hoitaa automaattisesti OAuth-tunnisteen säilyttämisen ja sisällyttämisen käyttäjän pyyntöihin. Sisäisesti järjestelmässä käytetään roolipohjaista pääsynhallintaa luvun 3.4.2 mukaisesti.

Esimerkkiratkaisussa järjestelmän reunalla on API-yhdyskäytävä, jonka kautta liikennettä järjestelmään hallitaan. API-yhdyskäytävä on yhteydessä reunatason auktorisointipalveluun, jossa käyttäjän pyynnöt autentikoidaan, eli tarkistetaan, onko käyttäjä kirjautunut. Lisäksi käyttöliittymälle myönnetään oma API-avain, jonka se sisällyttää kaikkiin käyttäjän pyyntöihin. Myös API-avain tarkistetaan API-yhdyskäytävässä, jotta muualta kuin käyttöliittymästä peräisin olevat pyynnöt voidaan estää. Autentikoinnin jälkeen pyynnöt auktorisoidaan karkealla tasolla. Esimerkiksi rajapinnan toiminnallisuuteen tarvittava rooli tarkistetaan käyttäjältä. Hienosäätöisempi auktorisointi tapahtuu mikropalvelutasolla sivuvaunuissa tai mikropalveluissa. Hienosäätöisempi auktorisointi määrittää tarkemmin, mitä kaikkea tietoa käyttäjälle rajapinnasta palautetaan. API-yhdyskäytävässä voidaan samalla kirjoittaa lokia kaikista pyynnöistä, jotka järjestelmään saapuu.

Kun pyyntö on autentikoitu, käytetään ulkoisen ja sisäisen tunnisteen erottelua luvun 4.5 mukaisesti. Auktorisointipalvelussa OAuth-tunniste vaihdetaan sitä vastaavan käyttäjän käyttöoikeustietoihin, ja tiedot tallennetaan allekirjoitettuun JWT-tunnisteeseen. Allekirjoitus tehdään käyttäen salaista avainta, joka on kohteena olevan mikropalvelun tiedossa. JWT-tunnisteelle asetetaan mahdollisimman lyhyt voimassaolo, sillä sen tarvitsee olla käyttökelpoinen vain kyseisen pyynnön ajan. Luotu JWT-tunniste liitetään käyttäjän alkuperäiseen pyyntöön, ja ohjataan oikealle mikropalvelulle.

Mikropalveluiden ohelle luodaan sivuvaunu-mikropalvelu, jonka tehtävä on autentikoida ja auktorisoida mikropalvelulle saapuvat pyynnöt. Mikropalveluille on luotu omat auktorisointisäännöt, jotka haetaan mikropalvelua käynnistäessä auktorisointipalvelulta. Nämä säännöt talletetaan mikropalvelun kyljessä olevaan sivuvaunuun. Tarvittaessa auktorisointisääntöjä voidaan muuttaa, ja ne voidaan välittää auktorisointipalvelun kautta kaikille sivuvaunuille ilman käyttökatkoa järjestelmässä.

API-yhdyskäytävästä pyyntö saapuu TLS-protokollalla salattuna mikropalvelun sivuvaunulle. Aluksi pyynnöstä tarkistetaan JWT-tunnisteen allekirjoitus luvun 3.2.4 mukaisesti. Seuraavaksi pyyntö auktorisoidaan JWT-tunnisteen sisältämillä käyttöoikeustiedoilla. Jos käyttöoikeudet ovat kunnossa, ohjataan pyyntö mikropalvelulle prosessoitavaksi, ja

sen jälkeen palautetaan vastaus. Muussa tapauksessa palautetaan virhe puuttuvista käyttöoikeuksista.

Edellä esitellyssä esimerkkiratkaisussa käytetään luvussa 4.6 hyviksi todettuja ratkaisuja ja malleja, kuten kertakirjautumista, API-yhdyskäytävää reunatasolla, jonka lisäksi käytetään sivuvaunumallia palvelutasolla. Lisäksi mikropalveluiden välisessä autentikoinnissa käytetään useiden lähteiden mukaan suositeltua JWT-ratkaisua. Myös mikropalveluiden periaatteita noudatetaan tässä esimerkissä. Esimerkiksi yhden vastuun periaate on huomioitu käyttämällä sivuvaunuja auktorisointilogiikan eristämiseksi omaksi palveluksi. Auktorisointilogiikan uudelleenkäyttö on sivuvaunujen ansiosta mahdollista. Myös ”defence-in-depth” -periaatetta noudatetaan, koska järjestelmän turvallisuus huomioidaan reunatasolla ja mikropalvelutasolla. Ehdotettua esimerkkiä voidaan käyttää pohjana uusien mikropalvelujärjestelmien autentikoinnin ja auktorisoinnin suunnittelussa.

Esimerkkimallia voidaan hyödyntää myös UNA Ytimen toteutuksissa. Reunatason osalta Ytimessä käytetään API-avainta ensimmäisenä autentikointivaiheena samalla tavoin kuin esimerkkiratkaisussa. Ytimessä esimerkkiratkaisun mukaista auktorisointia ei toisaalta reunatasolla vielä tapahdu, mutta se on toteutettavissa. Sen toteuttaminen tapahtuisi esimerkiksi käyttäen KH-mikropalvelua jo reunatasolla, jossa auktorisoitaisiin karkeasti esimerkiksi käyttäjän roolin perusteella, saako kyseistä toimintoa suorittaa. Tällainen reunatason auktorisointi ei ole välttämätöntä nykyisessä Ytimen toteutuksessa.

Ytimen palvelutason auktorisointiratkaisu perustuu keskitettyyn malliin, jota voi kehittää hyödyntäen esiteltyä esimerkkimallia. Kuitenkaan Ytimeen ei välttämättä ole kannattavaa toteuttaa esimerkkiratkaisun mukaista palveluverkkototeutusta siitä huolimatta, että se on hyvä ja suositeltava ratkaisu. Palveluverkon vaatimien sivuvaunujen toteutus vaatii ensinnäkin jonkin verran työtä, ellei käytetä jotain olemassa olevaa valmisratkaisua. Lisäksi kaiken auktorisointi- ja kommunikointilogiikan siirtäminen mikropalveluista sivuvaunuihin vaatii myös paljon työtä. Hajautetun auktorisointiratkaisun saavuttamiseksi vähemmän työtä vaativa vaihtoehto on toteuttaa JWT-tunnisteiden sisällyttäminen pyyntöihin. Edellä mainittu voidaan toteuttaa esimerkkiratkaisun mukaisesti käyttäen sisäisen ja ulkoisen tunnisteen erottelua järjestelmän reunatasolla. Erottelun toteuttamiseksi pyynnöt kuljetetaan KH-moduulin kautta, jossa istuntotunniste muunnetaan JWT-rakenteeksi ennen pyyntöjen ohjaamista mikropalveluihin.

6.3 Kehitysideat

Edellisten kappaleiden pohdintojen perusteella on kehittynyt ideoita, joilla Ytimen autentikointi- ja auktorisointiratkaisuja voidaan parannella ja muokata mikropalveluiden suunnitteluperiaatteiden mukaisiksi. Parannusehdotukset ovat listattu taulukossa 1. Kehitysideoihin ja niiden hyötyihin perehdytään tarkemmin seuraavissa kappaleissa.

Taulukko 1. *Kehitysideoita Ytimen auktorisointi- ja autentikointitoteutuksiin.*

Järjestelmän osa-alue	Parannusehdotus	Hyödyt
Palvelutaso	Käyttöoikeudet välitetään pyyntöjen mukana, jotta hajautetun mallin mukainen auktorisointi on mahdollista. JWT on hyvä ratkaisu tähän.	Ylimääräinen auktorisointiliikenne vähenee, joka nopeuttaa järjestelmää. Toteutus on suunnitteluperiaatteiden mukaisempi.
Reunataso	Sisäisen tunnisteen (JWT) ja ulkoisen tunnisteen (OAuth-istuntotunniste) erottelu KH-moduulin kautta.	Helpottaa hajautetun auktorisoinnin toteuttamista palvelutasolla.
Palveluiden välinen autentikointi	JWT-rakenteen tai API-avaimen käyttö palveluiden välisen autentikoinnin vahvistamiseksi.	Enemmän turvallisuutta ”defence-in-depth” -periaatteen mukaisesti.

Nykyisen UNA Ytimen päälle voidaan kehittää hajautetun auktorisointimallin mukainen ratkaisu, jolla vältetään ylimääräisiä KH-mikropalvelulle lähetettyjä auktorisointipyyntöjä. Se voidaan toteuttaa palvelutasolla käyttäen luvun 4.4 mukaista JWT-auktorisointia keskitetyn istuntotunnisteisiin perustuvan auktorisoinnin sijaan. KH-moduulista JWT-tunniste voidaan hakea uuden rajapinnan kautta. JWT-rajapintaa on suositeltavaa käyttää jo pyynnön saapuessa reunatasolle toteuttaen luvun 4.5 mukaisen tunnisteiden erottelun, jolloin käyttöoikeudet saadaan heti alusta alkaen sisäisten pyyntöjen mukaan. JWT sisältäisi ainakin käyttäjän roolit ja käyttöoikeudet sekä istuntotunnisteen. Perusideana JWT voi olla allekirjoitettu KH-moduulissa salaisella avaimella. Sama salainen avain täytyy välittää muille mikropalveluille esimerkiksi ympäristömuuttujana, jotta mikropalvelut voivat tarkistaa JWT-tunnisteen allekirjoituksen. Salaisten avainten käsittelyä voi jalostaa vielä pidemmälle. Esimerkiksi jos halutaan eri avaimet eri mikropalveluille, voidaan KH-moduuliin sisällyttää kaikkien mikropalveluiden salaisen avaimen, ja JWT muodostettaisiin aina KH-mikropalvelussa. Tässä tapauksessa muiden mikropalveluiden täytyy

tietää vain oma avaimensa allekirjoitusten tarkistamista varten. JWT-tunnisteen voimassaolo tulee olla melko lyhyt, jotta tunniste ei jää voimaan pitkäksi aikaa istunnon päättyessä yllättäen.

Ehdotuksen kanssa järjestelmässä voidaan edelleen käyttää samaa istuntopohjaista ratkaisua, jossa KH-moduulin istunnot ovat kytkettyjä eRA-istuntoihin. Osa toiminnallisuuksista saattavat edelleen tarvita kyvyn tarkistaa istunnon voimassaolon KH-moduulilta. JWT-rakenteen mukana välittyvällä istuntotunnisteella tarkistuksen voi tehdä vanhan malliin mukaisesti.

Hajautetun mallin hyöty ilmenee esimerkiksi tilanteessa, jossa käyttäjän lähettämä pyyntö aiheuttaa lisäpyyntöjä mikropalveluista toiseen. Alkuperäiseen pyyntöön lisätään reunatasolla tuore JWT-tunniste, ja se välittyy jatkopyyntöihin mukaan nykyisen istuntotunnisteen mukaisesti. Jos jonkin mikropalvelun tarvitsee tarkastella istunnon käyttöoikeuksia, löytyvät tiedot JWT-rakenteesta, eikä ylimääräisiä kutsuja KH-mikropalvelulle tarvita. Hajautetussa mikropalvelukohtaisessa auktorisoinnissa hyötyinä ovat ylimääräisen liikenteen ja siitä aiheutuneen viiveen välttäminen, sekä parempi virheensietokyky. Virheensietokyky esiintyy mahdollisuutena suorittaa jo auktorisoidun käyttäjän pyynnöt loppuun tilanteessa, jossa KH-palvelu menee vikatilaan. Se on mahdollista, sillä kaikki tarvittava tieto auktorisointiin on valmiiksi pyynnön mukana ja mikropalveluilla tiedossa. Ehdotuksen mukainen toteutus noudattaa paremmin ”design-for-failure” -periaatetta, ja on löyhien kytkentöjen -periaatteen mukainen.

Hajautettuun JWT-pohjaiseen mikropalvelutason auktorisointiin siirtymisestä saattaa seurata joitakin haasteita. Yksi haasteista on selvittää, mitä tapahtuu, jos istunto lukittuu kesken pyynnön. Myönnettyä JWT-tunnistetta ei voi peruuttaa. Sen vuoksi JWT-tunnisteiden voimassaolon tulee olla mahdollisimman lyhyt, ja JWT tulee muodostaa esimerkiksi Ytimen reunatasolla uusien käyttäjän kutsujen yhteydessä. Yksi ratkaisu ongelmaan on istunnon lukittuessa antaa jo aloitetun pyynnön suoriutua loppuun asti, mutta estää uudet pyynnöt, kunnes istunto on jälleen avattu.

6.4 Demo hajautetusta auktorisoinnista

Tässä luvussa käsitellään Ytimeen ehdotettua parannusta, joka muuttaa palvelukohtaista auktorisointiratkaisua keskitetystä mallista enemmän luvussa 4.4 käsitellyn hajautetun mallin mukaiseksi. Toiminnallisuudelle toteutettiin demototeutus, joka todistaa hajautetun mallin olevan mahdollinen, ja näyttää mallin tuomat hyödyt konkreettisesti. Hajautetussa mallissa käytetään JWT-rakenteita käyttöoikeuksien välittämiseen palveluiden välillä. Demo toteutettiin käyttäen Käytönhallinta- ja Käytönvalvonta-mikropalveluita,

koodaamalla muutokset palveluihin, ja ajamalla näitä kahta mikropalvelua omalla laitteella. Molemmat mikropalveluista ovat toteutettu C# ohjelmointikielellä, joten JWT-rakenteiden käsittelyyn voitiin käyttää samaa kirjastoa. Käytetty kirjasto on nimeltään ”System.IdentityModel.Tokens.Jwt”.

Demoa varten toteutettiin hajautetun mallin mukainen JWT-rakenteen generointimetodi KH-moduuliin. JWT generoidaan käyttäen salaista avainta, joka haetaan ympäristömuuttujasta. Näin salainen avain voidaan vaihtaa tekemättä muutoksia mikropalvelun koodiin. JWT-rakenteeseen sisällytettiin käyttäjän käyttöoikeudet ja istuntotunniste, sekä voimassaoloajaksi asetettiin muutama minuutti JWT:n luontihetkestä eteenpäin. JWT-metodin lisäksi luotiin uusi rajapinta, josta kyseinen rakenne haetaan. Rajapinnan kautta JWT voidaan hakea pyynnön saapuessa reunatasolle, jonka jälkeen JWT-rakenteen hyödyntäminen järjestelmän sisäisissä jatkopyynnöissä on mahdollista.

KV-moduuliin toteutettiin metodi saapuvien JWT-rakenteiden validointia ja käsittelyä varten. Yhteen KV-moduulin lokien hakurajapintaan toteutettiin muutos, jotta käyttöoikeuksien tarkastelu tapahtuu JWT-rakenteen sisällön perusteella. Rajapinta tarkistaa JWT-rakenteen allekirjoituksen, voimassaolon, ja mukana löytyvistä käyttöoikeuksista tarkistetaan käyttäjän oikeus kyseiseen rajapintaan. Keskitetyssä toteutuksessa mikropalvelu lähettää auktorisointivaiheessa kutsun KH-moduulille, joka palauttaa käyttäjän käyttöoikeudet. Kyseinen auktorisointipyyntö vältetään JWT-rakenteen avulla kokonaan. Saman toteutuksen pystyy helposti lisäämään muihin C# kielellä toteutettuihin mikropalveluihin. Eri teknologioilla toteutetut mikropalvelut eivät voi käyttää samaa toteutusta JWT-tunnisteiden käsittelyyn, vaan niihin se pitää toteuttaa erikseen. JWT-rakenteiden käsittelyyn on kuitenkin kattavasti kirjastoja, joten saman toiminnallisuuden koodaaminen eri kielellä ei ole ongelma.

Demossa pyyntöjen lähettämistä varten käytettiin Postman-sovellusta. Ensiksi kirjauduttiin sisään KH-moduulin kautta. Sen jälkeen pyydettiin KH-moduulin uudesta rajapinnasta JWT-rakenne, joka sisällytettiin seuraavaan KV-moduulille lähetettävään pyyntöön. JWT-rakenne asetettiin pyynnön otsikkotietueen ’Authorization’-sarakkeeseen, jossa OAuth2.0 istuntotunniste normaalisti välitetään. Lopuksi lähetettiin pyyntö KV-moduulin muokattuun rajapintaan, jossa JWT validoitiin, ja jonka jälkeen pyyntö suoritettiin.

Keskitetyn ja hajautetun toteutuksen vertailua varten suoritettiin samaa rajapintapyyntöä molemmilla toteutuksilla kymmenen kertaa, ja suoritukseen kulunutta aikaa mitattiin. Ennakko-oletuksena hajautettu malli on aina nopeampi, koska siinä vältetään ylimääräinen auktorisointipyyntö. Hajautetun mallin mukaisella auktorisoinnilla rajapinnan toiminta nopeutui keskiarvoltaan noin 150 millisekuntia. Testi tehtiin kannettavalla tietokoneella,

joka ei vastaa oikeaa mikropalveluiden ajoympäristöä. Lisäksi tulosta saattaa vääristää se, että sama laite suorittaa kahta mikropalvelua Visual Studio ohjelmiston ’debug’ moodissa. Hajautetun auktorisointimallin vaikutus on todennäköisesti mikropalveluiden oikeassa ympäristössä pienempi. Kuitenkin Ytimessä auktorisointipyyntöjä tapahtuu useissa mikropalveluissa, joten hajautetun auktorisoinnin kumulatiivinen hyöty järjestelmän nopeudessa saattaa olla huomattava.

7. YHTEENVETO

Tässä työssä käsiteltiin autentikoinnin ja auktorisoinnin toteuttamista mikropalveluarkkitehtuurissa. Mikropalveluarkkitehtuurissa ohjelma koostuu autonomisista palveluista muodostaen yhdessä kokonaisen järjestelmän. Mikropalveluiden autentikoinnin ja auktorisoinnin toteuttaminen sisältää uudenlaisia haasteita verrattuna monoliittiseen ohjelmaan. UNA Ydin -projektissa käytetään mikropalveluarkkitehtuuria. Työssä arvioitiin UNA Ytimeen aiemmin toteutettuja autentikointi- ja auktorisointiratkaisuja, sekä ehdotettiin muutamia keinoja, joilla kyseisiä autentikointi- ja auktorisointitoteutuksia voidaan parannella projektin jatkokehityksessä. Lisäksi työssä tuotettiin esimerkkimalli autentikoinnin ja auktorisoinnin toteuttamiseksi mikropalveluarkkitehtuurissa.

Luvussa 4 perehdyttiin erilaisiin kirjallisuudesta löytyneisiin ratkaisuihin ja malleihin mikropalveluiden autentikoinnin ja auktorisoinnin toteuttamiseksi. Ratkaisut pystyttiin luokittelemaan seuraavasti: järjestelmän reunalla tapahtuva auktorisointi, mikropalvelutasolla tapahtuva auktorisointi, palveluiden välinen autentikointi sekä ulkoisen ja sisäisen identiteetin erottelu. Kirjallisuuden perusteella pystyttiin koostamaan muutamia suositeltavia malleja mikropalveluarkkitehtuurin autentikointia ja auktorisointia varten. Käyttäjien autentikointi kannattaa toteuttaa käyttäen kertakirjautumistekniikoita. Palveluiden väliseen autentikointiin löytyi useita ratkaisuja, joista yleisin oli käyttää JWT-tekniikkaa. Mikropalvelutasolla auktorisointi kannatti toteuttaa hajautetusti kussakin mikropalvelussa itsessään tai käyttäen palveluverkkoa. Käyttöoikeudet tulee välittää pyyntöjen mukana, jotta vältytään ylimääräisiltä kutsuilta järjestelmän sisällä. JWT-tunnisteet soveltuivat myös tähän käyttötarkoitukseen erityisen hyvin.

Luvussa 6 näitä ratkaisuja verrattiin UNA Ytimessä käytettyihin menetelmiin. Vertailun ja pohdinnan tuloksena kehittyi muutama parannusehdotus Ytimen mikropalvelutason auktorisointiratkaisuille. Keskeisin kehitysidea oli muuttaa Ytimen palvelutason auktorisointia mikropalveluarkkitehtuurin suunnitteluperiaatteiden mukaisemmaksi hajautetuksi malliksi. Toteuttamalla JWT-rakenteisiin perustuvan autentikoinnin ja auktorisoinnin, voidaan Ytimen sisäisten pyyntöjen mukana välittää helposti käyttäjän tietoja ja käyttöoikeuksia. Hajautettu auktorisointimalli nopeuttaa Ytimen toimintaa vähentämällä auktorisointipyynnöistä aiheutunutta verkkoliikennettä. Lisäksi Ytimen palvelutason auktorisointi muuttuu mikropalveluarkkitehtuurin suunnitteluperiaatteiden mukaisemmaksi. Kyseistä hajautettua mallia käsiteltiin tarkemmin luvussa 4.4. Lisäksi kyseisen kehitysidean konkretisointia varten toteutettiin demototeutus hajautetusta auktorisoinnista palvelutasolla. Toissijaisena tuloksena työssä koostettiin luvussa 4 löydetyistä autentikointi- ja

auktorisointimalleista esimerkkiratkaisu, joka on aikaisempien tutkimusten suositusten mukainen. Esimerkkiratkaisua käsiteltiin tarkemmin luvussa 6.2.

Työn tuloksia voidaan hyödyntää UNA Ytimen jatkokehityksessä ja uusien mikropalvelujärjestelmien autentikoinnin ja auktorisoinnin suunniteluun. Kaikkia työssä esiin nostettuja malleja ja Ytimen kehitysideoita ei tämän työn sisällä pystytty käytännön tasolla tutkimaan. Lisäksi työssä keskityttiin http-pohjaista kommunikointia käyttävien mikropalveluiden autentikointi- ja auktorisointitoteutuksiin. Jatkotutkimusta voisi tehdä muita kommunikaatiomalleja käyttävien mikropalvelujärjestelmien autentikointiin ja auktorisointiin.

LÄHTEET

[1] T. Rimpiläinen, Psykoterapiakeskus Vastaamon kiristäjä julkaisi yöllä lisää erittäin arkaluontoisia potilaskertomuksia, Yle, 2020, Saatavissa (haettu 23.02.2021): https://yle.fi/uutiset/3-11606925

[2] S. Newman, Building microservices, O'Reilly Media, Inc., 2015, Saatavissa: https://learning.oreilly.com/library/view/building-microservices/9781491950340/

[3] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, Microservices: Yesterday, Today, and Tomorrow, arXiv, 2017, pp. 6–14, Saatavissa: https://arxiv.org/abs/1606.04036

[4] A. Pereira-Vale, G. Marquez, H. Astudillo, EB. Fernandez, Security Mechanisms Used in Microservices-Based Systems: A Systematic Mapping, CLEI, 2019, 6 p.

[5] Vaatimusmäärittely, UNA, dokumentti, 2019

[6] ”Mikä sote-uudistus?”, Sote-uudistus, verkkosivu, Saatavissa (viitattu 5.10.2021): https://soteuudistus.fi/uudistus-lyhyesti-

[7] H. Vural, M. Koyuncu, S. Guney, A Systematic Literature Review on Microservices, Computational Science and Its Applications – ICCSA 2017, Cham, 2017, Springer International Publishing, pp. 203-217

[8] RestfulAPI, “What is REST”, verkkosivu, 2021, Saatavissa (haettu 18.10.2021): https://restfulapi.net

[9] X. Larrucea, I. Santamaria, R. Colomo-Palacios, C. Ebert, Microservices, 2018, IEEE Software, Volume 35, Issue 3, pp. 96-100

[10] Lewis, J., Fowler, M.: “Microservices”, martinFowler.com, 2014, Saatavissa (haettu 14.7.2021): http://martinfowler.com/articles/microservices.html

[11] D. Yu, Y. Jin, Y. Zhang, X. Zheng, A survey on security issues in services communication of microservices-enabled fog applications. Concurrency and Computation: Practice and Experience, 2019, Vol. 31 (22)

[12] R. Chandramouli, Security Strategies for Microservices-based Application Systems, Na-tional Institute of Standards and Technology, Gaithersburg, MD, NIST Special Publication (SP) 800-204, 2019, Saatavissa: https://doi.org/10.6028/NIST.SP.800-204

[13] E. Wolff, Microservices: Flexible Software Architecture, Addison-Wesley Professional, 2016, Saatavissa: https://learning.oreilly.com/library/view/microservices-flexible-software/9780134650449/

[14] J. Soldani, D. A. Tamburri, W. V. D. Heuvel, The pains and gains of microservices: A Systematic grey literature review, ScienceDirect, Journal of Systems and Software, 2018, Volume 146, pp. 215-232

[15] T. Killalea, The Hidden Dividends of Microservices, Communications of the ACM, 2016, Volume 59, Issue 8, pp. 42 - 45

[16] Kocher PS, Microservices and containers, Pearson Addison Wesley, 2018, Saatavissa: https://learning.oreilly.com/library/view/Microservices-and-containers/9780134591728/

[17] K. Indrasiri, P. Siriwardena, Microservices for the Enterprise: Designing, Developing, and Deploying, Apress, 2018, luvut 11-12, Saatavissa: https://learning.oreilly.com/library/view/microservices-for-the/9781484238585/

[18] L. Bassett, M. Foley, K. Brown, J. Kwityn, C. Roumelioties, E. Troutman ym., Introduction to JavaScript object notation: a to-the-point guide to JSON, First edition, O'Reilly, 2015, luku 1, Saatavissa: https://learning.oreilly.com/library/view/Introduction-to-JavaScript/9781491929476

[19] Json, Introducing JSON, verkkosivu, Saatavissa (haettu 13.08.2021): https://www.json.org/jsonen.html

[20] K. H. Goldberg, XML: Visual QuickStart Guide, Second Edition, Peachpit Press, 2008, luku 1, Saatavissa: https://learning.oreilly.com/library/view/XML-Visual-QuickStart/9780321602589/

[21] C. Surianarayanan, G. Ganapathy, P. Raj, Essentials of Microservices Architecture: Paradigms, Applications, and Techniques. 1st ed. Milton: CRC Press; 2020, pp. 74-83

[22] F. Doglio, REST API Development with Node.js Manage and Understand the Full Capabilities of Successful REST Development, Apress, 2018, luku 1, Saatavissa: https://learning.oreilly.com/library/view/rest-api-development/9781484237151/

[23] gRPC, About gRPC, verkkosivu, Saatavissa (haettu 18.08.2021): https://grpc.io/about/

[24] K. Indrasiri, D. Kuruppu, gRPC: Up and Running, O'Reilly Media, Inc, 2020, luku 1, Saatavissa: https://learning.oreilly.com/library/view/grpc-up-and/9781492058328/

[25] GraphQL, verkkosivu, Saatavissa (haettu 19.08.2021): https://graphql.org/

[26] L. Johansson, D. Dossot, RabbitMQ Essentials - Second Edition, Packt Publishing, 2020, luku 1, Saatavissa: https://learning.oreilly.com/library/view/rabbitmq-essentials/9781789131666/

[27] I. N. McAteer, M. I. Malik, Z. Baig, P. Hannay, Security vulnerabilities and cyber threat analysis of the AMQP protocol for the internet of things, The Proceedings of 15th Australian Information Security Management Conference, Edith Cowan University, December 2017, pp. 70-80, Saatavissa: https://ro.ecu.edu.au/ism/203/

[28] T. Sharvari, K. Sowmya Nag, A Study on Modern Messaging Systems- Kafka, RabbitMQ and NATS Streaming, arXiv, 2019, 1 p., Saatavissa: https://arxiv.org/abs/1912.03715

[29] G. M. Roy, RabbitMQ in Depth, Manning Publications, 2017, luku 1, Saatavissa: https://learning.oreilly.com/library/view/rabbitmq-in-depth/9781617291005/

[30] RabbitMQ, Clients Libraries and Developer Tools, verkkosivu, Saatavissa (haettu 20.08.2021): https://www.rabbitmq.com/devtools.html

[31] X. J. Hong, H. Sik Yang, Y. H. Kim, Performance Analysis of RESTful API and RabbitMQ for Microservice Web Application, International Conference on Information and Communication Technology Convergence (ICTC), IEEE, 2018. pp. 257–259

[32] Kubernetes, “What is Kubernetes?”, verkkosivu, 2021, Saatavissa (haettu 10.09.2021): https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[33] M. Tyson, What is an API gateway? API simplicity and stability, InfoWorld.com, 2021, Saatavissa (haettu 20.10.2021): https://www.infoworld.com/article/3616188/what-is-an-api-gateway-api-simplicity-and-stability.html

[34] C. Anderson, Docker [Software Engineering], IEEE Software, 2015, Vol. 32 (3) pp. 102–c3

[35] Kubernetes, “Kubernetes Components”, verkkosivu, 2021, Saatavissa (haettu 10.09.2021): https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[36] M. Rimol, Gartner Says Worldwide IaaS Public Cloud Services Market Grew 40.7% in 2020, Gartner, 2021, Saatavissa (haettu 11.08.2021): https://www.gartner.com/en/newsroom/press-releases/2021-06-28-gartner-says-worldwide-iaas-public-cloud-services-market-grew-40-7-percent-in-2020

[37] N. Ruparelia, Cloud Computing, Cambridge, Massachusetts: The MIT Press, 2016, Saatavissa: https://ebookcentral.proquest.com/lib/tampere/detail.action?pq-origsite=primo&docID=4527741

[38] L. Rosencrance, Definition: authentication. SearchSecurity.com, 2018, Saatavissa (haettu 12.09.2021): https://searchsecurity.techtarget.com/definition/authentication

[39] S. Boonkrong, Authentication and Access Control: Practical Cryptography Methods and Tools, O’Reilly Media, 2020, Saatavissa: https://learning.oreilly.com/library/view/authentication-and-access/9781484265703/

[40] V Beltran. Characterization of Web Single Sign-On Protocols. IEEE Communications Magazine Communications Standards Supplement, 2016, Vol 54, no. 7, pp. 24–30

[41] Y. Wilson, A. Hingnikar, Solving Identity Management in Modern Applications Demystifying OAuth 2.0, OpenID Connect, and SAML 2.0. 1st ed. Berkeley, CA: Apress, 2019, luvut 11 ja 12, Saatavissa: https://learning.oreilly.com/library/view/solving-identity-management/9781484250952/

[42] T. Bazaz, A. Khalique, A Review on Single Sign on Enabling Technologies and Protocols, International Journal of Computer Applications, 2016, vol. 151, no. 11, pp. 18–25, Saatavissa: https://www.ijcaonline.org/archives/volume151/number11/bazaz-2016-ijca-911938.pdf

[43] V. Radha, A Survey on Single Sign-On Techniques, Procedia Technology, 2012, vol 4, pp. 134–139, Saatavissa: https://doi.org/10.1016/j.protcy.2012.05.019

[44] D. Hardt, The OAuth 2.0 Authorization Framework, RFC 6749, Internet Engineering Task Force (IETF), 2012, Saatavissa: https://tools.ietf.org/html/rfc6749

[45] M. Spasovski, OAuth 2.0 Identity and Access Management Patterns, 1st edition, Birmingham: Packt Publishing, 2013

[46] P. Siriwardena, Advanced API Security OAuth 2.0 and Beyond, 2nd edition, Berkeley, CA: Apress, 2020, Saatavissa: https://learning.oreilly.com/library/view/advanced-api-security/9781484220504/

[47] N. Sakimura, J. Bradley, M. Jones, OpenID Connect Core 1.0, 2014, Saatavissa: https://openid.net/specs/openid-connect-core-1_0.html

[48] J. Hughes, E. Maler, Security assertion markup language (saml) v2.0 technical overview, OASIS Security Services TC, 2008, Saatavissa: http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html

[49] S. Cantor, I. J. Kemp, N. R. Philpott, E. Maler, Assertions and protocols for the oasis security assertion markup language, OASIS Standard, 2004, Saatavissa: https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[50] JWT, Introduction to JSON Web Tokens, verkkosivu, Saatavissa: https://jwt.io/introduction/

[51] J. Bradley, N. Sakimura, M.B. Jones, JSON Web Token (JWT), Internet Engineering Task Force (IETF), 2015, Saatavissa: https://tools.ietf.org/html/rfc7519

[52] V. Galante, Practical Role-Based Access Control, Information Security Journal: A Global Perspective, 2009, Vol 18, Issue 2, pp. 64-73

[53] B. W. Lampson, Protection, Operating systems review, 1974, vol 8, issue 1, pp. 18–24

[54] X. Jin, R. Krishnan, R. Sandhu, A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC, Lecture Notes in Computer Science 7371, 2012, pp. 41-55

[55] D. Ferraiolo, R. Chandramouli, D. R. Kuhn, Role-based access control, 2nd ed. Boston: Artech House, 2007

[56] V. Hu, Attribute-Based Access Control, Ebscohost, 2018

[57] A. Pereira-Vale, E. B. Fernandez, R. Monge, H. Astudillo, G. Mãrquez, Security in Microservice-Based Systems: A Multivocal Literature Review, Computers & security, 2021, vol. 103, Saatavissa: https://www-sciencedirect-com.libproxy.tuni.fi/science/article/pii/S0167404821000249?via%3Dihub

[58] A. Hannousse, S. Yahiouche, Securing Microservices and Microservice Architectures: A Systematic Mapping Study, arXiv, 2020, Saatavissa: https://arxiv.org/abs/2003.07262

[59] A. Barabanov, D. Makrushin, Authentication and authorization in microservice-based systems: survey of architecture patterns, arXiv, 2020, Saatavissa: https://arxiv.org/abs/2009.02114

[60] J. Christensen, C. Hickman, Service-to-Service Authentication for Microservice APIs, Hackernoon, 2019, Saatavissa (viitattu 24.09.2021): https://hackernoon.com/service-to-service-authentication-for-microservice-apis-ccf4ab8073e6

[61] T. Goethals, D, Kerkhove, B. Volckaert, F. De Turck, Scalability Evaluation of VPN Technologies for Secure Container Networking, 15th International Conference on Network and Service Management, CNSM, 2019

[62] Kubernetes, “Cluster Networking”, verkkosivu, Saatavissa (haettu 01.10.2021): https://kubernetes.io/docs/concepts/cluster-administration/networking/

[63] J. Fruhlinger, What is a service mesh? Easier container networking, InfoWorld.com, 2019, Saatavissa (viitattu 20.10.2021): https://www.infoworld.com/article/3402260/what-is-a-service-mesh-easier-container-networking.html

[64] A. Tiwari, A sidecar for your service mesh, A. Tiwarin blogi, 2017, Saatavissa (viitattu 30.09.2021): https://www.abhishek-tiwari.com/a-sidecar-for-your-service-mesh/

[65] K. Indrasiri, Service Mesh for Microservices, Medium, 2017, Saatavissa (viitattu 3.10.2021): https://medium.com/microservices-in-practice/service-mesh-for-microservices-2953109a3c9a

[66] R. Chandramouli, Z. Butcher, Building Secure Microservices-based Applications Using Service-Mesh Architecture, NIST Special Publication 800-204A, 2020, pp. 1–17, Saatavissa: https://doi.org/10.6028/NIST.SP.800-204A

[67] M. Stocker, O. Zimmerman, U. Zdun, D. Lübke, C. Pautasso, Interface Quality Patterns: Communicating and Improving the Quality of Microservices APIs, ACM International Conference Proceeding Series, 2018, no. 10, pp. 1-16

[68] R. Xu, W. Jin, D. Kim, Microservice Security Agent Based On API Gateway in Edge Computing, Sensors, 2019, vol. 19, no. 22, 4905 p.

[69] A. Bhingole, Microservices Authentication and Authorization Using API Gateway, DZone, 2018, Saatavissa (viitattu 29.9.2021): https://dzone.com/articles/security-in-microservices

[70] R. Reinicke, Authorization and Authentication with Microservices, LeanIX Blog, 2017, Saatavissa (viitattu 25.9.2021): https://www.leanix.net/en/blog/authorization-authentication-with-microservices

[71] B. Aggarwal, Authentication and Authorization in Microservices, DZone, 2019, Saatavissa (viitattu 28.09.2021): https://dzone.com/articles/authentication-and-authorization-in-microservices

[72] X. He, X. Yang, Authentication and authorization of end user in microservice architecture, Journal of Physics: Conference Series, Vol. 910, 2017, Saatavissa: https://iopscience-iop-org.libproxy.tuni.fi/article/10.1088/1742-6596/910/1/012060

[73] A. Banati, E. Kail, K. Karoczkai, M. Kozlovszky, Authentication and Authorization Orchestrator for Microservice-Based Software Architectures, 41st International Convention on Information and Communication Technology, Electronics

and Microelectronics (MIPRO), Croatian Society MIPRO, 2018, pp. 1180–1184, Saatavissa: https://doi-org.libproxy.tuni.fi/10.23919/MIPRO.2018.8400214

[74] UNA Ydin, UNA, verkkosivu, Saatavissa (viitattu 06.10.2021): https://unaoy.fi/una-ohjelmat/una-ydin/

[75] UNA Ytimen ja Tilannekuvan toimittajien tuottama tilannekatsaus, UNA, verkkojulkaisu ja video, 2020, Saatavissa (viitattu 06.10.2021): https://unaoy.fi/julkaisut/una-ytimen-ja-tilannekuvan-toimittajien-tuottama-tilannekatsaus/

[76] Mikropalveluiden dokumentaatiot, UNA

[77] Kela, Kanta-palvelut, verkkosivu, 2021, Saatavissa (viitattu 5.10.2021): https://www.kela.fi/tietopalvelut-kanta-palvelut

[78] eRA, Atostek, verkkosivu, Saatavissa (viitattu 5.10.2021): https://atostek.com/era/

[79] Laki sosiaali- ja terveydenhuollon asiakastietojen sähköisestä käsittelystä, Finlex, 2 lk., Saatavissa (viitattu 14.10.2021): https://finlex.fi/fi/laki/ajantasa/2007/20070159