SAVONIA

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN ALA

KESKITETTY PÄÄSYNHALLINTA HAJAUTETUSSA JÄRJESTEL-MÄSSÄ

TEKIJÄ Tapio Räsänen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä Tapio Räsänen	
Työn nimi Keskitetty pääsynhallinta hajautetussa järjestelmässä	
Päiväys 20.05.2025	Sivumäärä/Liitteet 45/0
Yhteistyötaho Suonentieto Oy	

Opinnäytetyössä tutkittiin, miten toteuttaa keskitetysti pääsynhallintaa mikropalveluympäristössä. Työssä käytiin läpi pääsynhallinnan haasteita mikropalveluympäristöissä ja tutkittiin käytössä olevia pääsynhallinta – ja arkkitehtuurimalleja pääsynhallinnan toteuttamiseen. Työssä tutkittiin myös käyttöoikeustunnisteiden käyttöä ja erilaisia työkaluja ja kehyksiä pääsynhallintamekanismien toteuttamiseksi toimeksiantajan järjestelmässä.

Toimeksiantaja on kehittämässä uusia mikropalveluarkkitehtuuria hyödyntäviä sovelluksia, joiden käyttöoikeuksien hallintaa haluttiin toteuttaa keskitetysti yhden palvelukokonaisuuden alta. Tämän takia nähtiin tarvetta tämän opinnäytetyön toteuttamiselle. Toimeksiantajan kehitettävässä järjestelmässä oli toteutettu osittain pääsynhallintaan liittyviä mekanismeja. Näiden mekanismien laajuus ei riittänyt kattamaan tulevien sovellusten hyvinkin tarkkarajaisia vaatimuksia pääsynhallintasääntöjen osalta. Tämän takia haluttiin tutkia erilaisia arkkitehtuuri- ja pääsynhallintamalleja sekä teknisiä ratkaisuja uusien hallintatyökalujen ja palvelujen toteuttamiseen.

Uuden järjestelmän arvona on helpottaa järjestelmän käyttöoikeuksien ylläpitoa keskittämällä hallintaa yksittäiseen palveluun ja parantamalla asiakaskokemusta. Kehitettävän järjestelmän tulisi mahdollistaa asiakasorganisaatioon kuuluvien käyttäjien itsenäinen luvittaminen olematta yhteydessä toimeksiantajaan.

Työn lopputuloksena päätettiin järjestelmän reunalle toteuttaa API-yhdyskäytävä karkeaan pääsynhallintaan. Lisäksi tulevaisuudessa toteutetaan erillinen valtuutuspalvelin järjestelmän palvelutasolle. Valtuutuspalvelimen ja pääsynhallintamekanismien toteuttamiseen päädyttiin hyödyntämään palveluverkkoa ja sivuvaunuja.

Avainsanat
Pääsynhallinta, Mikropalvelu, Kubernetes, Laravel, Valtuutus, HAProxy, Istio

SISÄLTÖ

1 JOHDANTO

Mikropalveluarkkitehtuuriin perustuvat järjestelmät ovat kasvattaneet jatkuvasti suosiotaan, ja niistä on tullut tärkeä osa yrityksille erityisesti, kun ne ottavat käyttöön uusia pilvipohjaisia sovelluksia. (Walia 2023,6)

Mikropalveluarkkitehtuurin etuina ovat palvelujen hallinnan ja ylläpidon helpottuminen pidemmällä aikavälillä. Arkkitehtuurissa ajattelumallina on palvelun muokkaaminen ilman että muutokset vaikuttavat muihin järjestelmän toimintoihin. Järjestelmä skaalautuu ja mukautuu erilaisiin muutoksiin, mikä mahdollistaa kustannussäästöt ylläpidossa. (Atlassian n.d.)

Tämän opinnäytetyön aiheena on tutkia, miten voitaisiin toteuttaa keskitetty pääsynhallinta toimeksiantaja Suonentieto Oy:n hajautettuun järjestelmään. Järjestelmä on toteutettu mikropalveluarkkitehtuurina, mutta järjestelmään pitäisi toteuttaa keskitetty ratkaisu pääsynhallinnan toteuttamiseen ja ylläpitoon. Nykyinen järjestelmä koostu useammasta käyttöliittymästä ja mikropalveluina toteutetuista taustapalvelimesta, joissa on API-Rajapintoja. Järjestelmään on toteutettu suppea roolipohjainen pääsynhallinta. Toteutettu pääsynhallinta ei kuitenkaan riitä kattamaan yrityksen tarpeita tulevaisuudessa.

Yritys on kehittämässä uusia web-pohjaisia sovelluksia, joissa erilaiset asiakkaille myönnetyt käyttäjätilit ja niihin liittyvät luvitukset sekä hallinta ovat keskiössä. Tämän takia yrityksessä koettiin tarvetta toteuttaa ratkaisu keskitetyn pääsynhallinnan toteuttamiseksi. Tämä parantaa myös yrityksen asiakaskokemusta ja vapauttaa henkilöstöresursseja muihin tehtäviin. Tavoitteena on kehittää järjestelmä, jossa asiakasorganisaatio voi luoda ja luvittaa itse organisaationsa sisäiset käyttäjät ja liittää näille haluamansa resurssit käyttöön. Aikaisemmin tämä työ on tehty toimeksiantajan puolelta ja sen on koettu olevan aikaa vievä työvaihe.

Opinnäytetyön tavoitteena oli tutkia mitä arkkitehtuurivaihtoehtoja on olemassa keskitetyn pääsynhallinnan toteuttamiseen mikropalveluarkkitehtuurissa ja mikä niistä vastasi parhaiten toimeksiantajan tarpeita. Valitun ratkaisun tulisi olla mahdollisimman yhteensopiva jo toteutettujen ratkaisujen ja API-rajapintojen kanssa. Keskeisiä kysymyksiä opinnäytetyönkannalta oli kolme:

Miten käyttäjä tunnistetaan ja valtuutetaan API-rajapinnoissa mikropalveluarkkitehtuurissa?

Mikä on oikea tapa toteuttaa tämä Laravel-ohjelmistokehystä käyttäen?

Miten toteuttaa erillinen autentikointi ja valtuutuspalvelin mikropalveluarkkitehtuurissa?

1.1 Toimeksiantaja

Toimeksiantajana toimi Suonentieto Oy. Suonentieto on kuopiolainen taloushallinnon ja viljelysuunnittelun ohjelmistoja tarjoava yritys. Yritys on perustettu 1988 ja työllistää noin 40 henkilöä. Suonentieto tekee maatalouteen ja sähköiseen taloushallintoon liittyviä sovelluksia erilaisille maatalous- ja tilitoimistoille. Ohjelmistot tarjoavat erilaisia ratkaisuja viljelysuunniteluun ja taloushallintoon. Ohjelmistoista löytyy niin työpöytäsovelluksia, web-sovelluksia kuin myös erilaisia mobiilisovelluksia. Näitä ohjelmistoja käyttää päivittäin lähes 10 000 asiakasta. (Suonentieto n.d)

1.2 Tarve

Suonentieto Oy tarvitsi keskitetyn palvelun käyttöoikeuksien hallintaan. Toimeksiantajan tarjoamissa sovelluksissa on käytössä käyttäjätileihin ja käyttöoikeuksiin perustuvia toimintoja. Yrityksessä oli aloitettu kehittämään uusia web-sovelluksia, jotka tarvitsivat uudistetun käyttäjätileihin nojautuvan pääsynhallinnan. Pääsynhallinnasta haluttiin tehdä keskitetty ratkaisu, jota voitaisiin käyttää niin uusissa kehitettävissä sekä jo käytössä olevissa sovelluksissa. Yksi merkittävimmistä uudistuksista oli luoda ratkaisu, jossa asiakasorganisaatio voi luvittaa itsenäisesti organisaationsa kuuluvat käyttäjätilit. Nykyisessä ratkaisussa tämä onnistui vain toimeksiantajan kautta ja täten keskitetty ratkaisu paransi myös asiakaskokemusta.

1.3 Lyhenteet

API = Application Programming Interface. Sovellusohjelmointirajapinta. Mahdollistaa ohjelmistojen välinen vuorovaikutus määrättyjen sääntöjen avulla

HTTP = Hypertext Transfer Protocol. Protokolla, jota käytetään tiedon siirtämiseen verkkosivujen välillä. Selaimen ja palvelimen välinen viestintä tapahtuu yleensä HTTP:n kautta.

IP = Internet Protocol. Vastaa tietopaketin reitittämisestä verkossa. IP-osoitteet yksilöivät laitteet internetissä.

Backend = Taustapalvelu tai palvelinpuoli. Käsittelee liiketoimintalogiikan, tietokannan ja API:t. Ei yleensä näy loppukäyttäjälle suoraan.

Frontend = Käyttöliittymä tai asiakaspuoli. Sovelluksen osa, jonka käyttäjä näkee ja jolla hän on vuorovaikutuksessa.

TLS = Transport Layer Security. Salausprotokolla, joka turvaa verkkoliikenteen.

mTLS = Mutual TLS (vastavuoroinen TLS). TLS:n muoto, jossa sekä asiakas että palvelin todentavat toisensa digitaalisilla varmenteilla.

OSI-malli = Open Systems Interconnection -malli. Viestintäprotokollien kerrosmalli (7 kerrosta), jota käytetään kuvaamaan tietoliikenteen eri osa-alueita.

Lua = Kevyt ja pieni skriptikieli

Skripti = Lyhyt ohjelmakoodi, joka suorittaa automaattisesti tehtäviä.

NIST = National Institute of Standards and Technology. Kehittää teknisiä standardeja ja ohjeistuksia, erityisesti kyberturvallisuuden alalla.

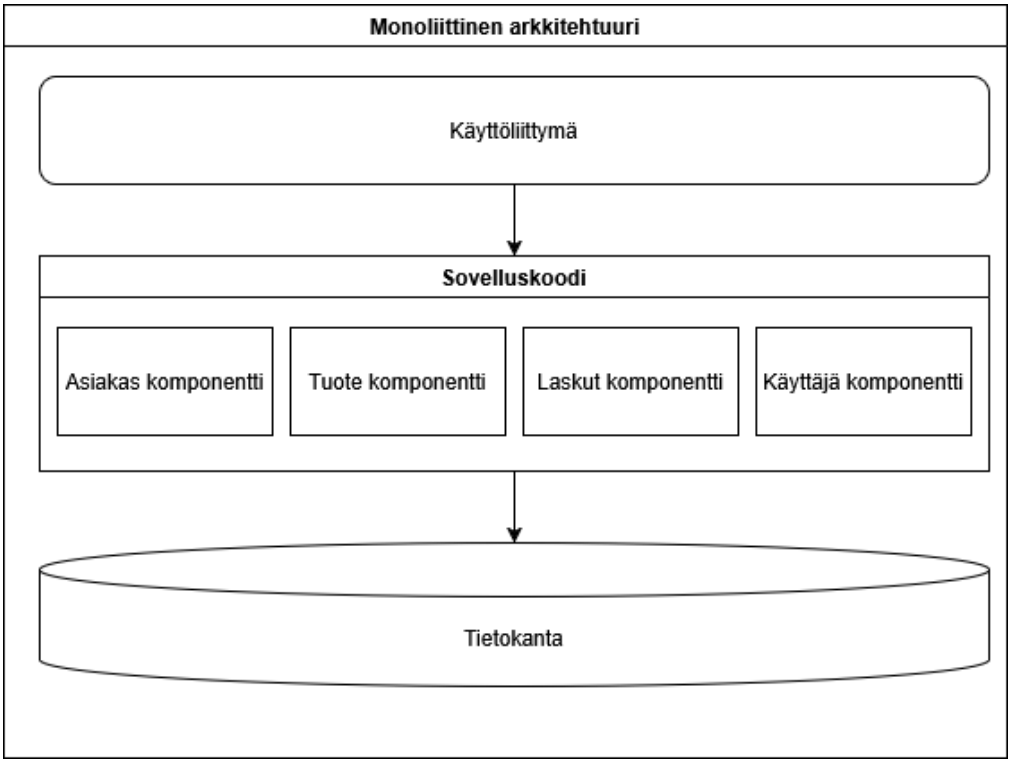
Defense in Depth = Tietoturvaperiaate, jossa käytetään useita päällekkäisiä suojauskerroksia. Ideana on estää, havaita ja lieventää uhkia monitasoisesti,

2 TEORIA

2.1 Mikropalveluarkkitehtuuri

Mikropalveluarkkitehtuuria käyttävät sovellukset koostuvat useista pienistä itsenäisistä palveluista, joiden välillä on löysiä kytköksiä. Tällaista yksittäistä palvelua kutsutaan mikropalveluksi ja yksittäinen mikropalvelu vastaa tietystä yksittäisestä sovelluksen toiminallisuudesta. Yksittäinen palvelu ei ole riippuvainen muista palveluista annetun tehtävän suorittamiseksi. Palvelun toiminta perustuu ajatukseen suorittaa yksittäinen asia hyvin, joka asettaa vahvan rajan eri palveluiden välille. Jokaisella mikropalvelulla on oma API-Rajapinta, jonka kautta ne keskustelevat ja vaihtavat dataa käyttäen kevyitä viestintäprotokollia. Jokainen rajapinta on tarkasti määritelty ja tämän ansiosta yksittäistä mikropalvelua voidaan kehittää vastamaan hyvin tarkasti sovelluksen tarpeita. Yksittäisellä mikropalvelulla on yleensä oma tietokanta ja lähdekoodi. (GeeksforGeeks 2025; Surianarayanan, Ganapathy & Pethuru 2019, 57)

Mikropalvelut luovat yhdessä kokonaisuuden, joka suorittaa toimintoja, joita loppukäyttäjä tarvitsee sovellusta käyttääkseen. Tätä arkkitehtuuria kutsutaan Service Oriented Architecture eli palvelukeskeiseksi arkkitehtuuriksi (Amazon n.d). Mikropalveluarkkitehtuuria käytetään vaihtoehtona korvaamaan perinteistä monoliittista arkkitehtuuria hyödyntäviä sovelluksia. Monoliittisessa arkkitehtuurissa käytetään yhtä koodipohjaa ja tietokantaa suorittamaan sovelluksen kaikki toiminnot (TechTarget 2024; Powell & Smalley 2024). Tällöin kaikki sovelluksen toiminnot ja komponentit ovat tiukasti kytköksissä toisiinsa toisinkuin mikropalveluarkkitehtuurissa, jossa yksittäinen toiminto voidaan suorittaa itsenäisesti omana mikropalvelunaan. Monoliittinen sovellus sisältää tyypillisesti käyttöliittymän, tietokannan ja palvelinpuolen sovelluksen kuten on havainnollistettu kuvassa 1 (AWS n.d).



Kuva 1. Monoliittinen arkkitehtuuri

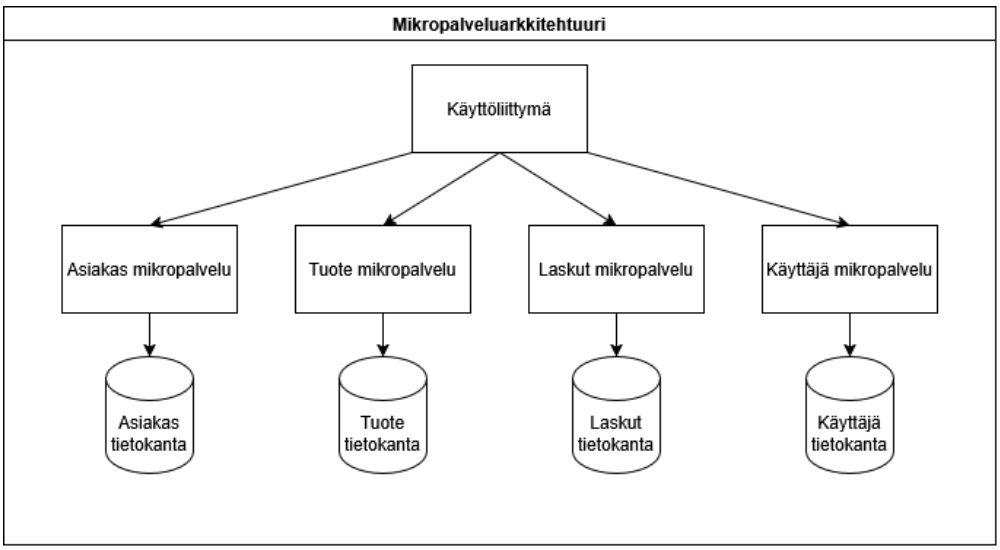
Monoliittisten sovellusten huonoja puolia ovat TechtTargetin (2024) mukaan:

1. Yhden ison koodipohjan ylläpidettävyys on haastavaa. Yksittäisen toiminnallisuuden suorittavan komponentin päivittäminen vaatii muutoksia kaikkiin sitä käyttäviin komponentteihin. Tämä ongelma nousee esiin erityisesti isoissa sekä monimutkaisissa sovelluksissa ja järjestelmissä.
2. Iso yksittäinen koodipohja voi kasvaa ajan saatossa vaikeasti tulkittavaksi kehittäjille. Tämä vaikeuttaa uusien toiminnallisuuksien toteuttamista ja saattaa heikentää koodin laatua.
3. Jokainen päivitys sovellukseen vaatii koko koodipohjaan ja täten myös sovelluksen uudelleenkäynnistämisen sen sijaan, että päivitettäisiin vain yksittäinen komponentti.
4. Sovelluksen ja koodipohjan koko sekä monimutkaisuus voivat kasvattaa sovelluksen viivettä.
5. Sovelluksen eri osat voivat olla konfliktissa keskenään esimerkiksi erilaisten käsiteltävien resurssien takia.
6. Yhdessä komponentissa oleva ohjelmointivirhe ”bugi” voi kaataa koko sovelluksen tai halvaannuttaa osan siitä. Tämä vaikuttaa negatiivisesti sovelluksen luotettavuuteen.
7. Monoliittisia sovelluksia ei ole helppo mukauttaa uusiin teknologioihin niiden koon ja monimutkaisuuden vuoksi. Tämä rajoittaa uusien ohjelmointikielien ja kehyksien käyttöä organisaatioissa.

Edellä mainittujen haasteiden takia yritykset ovat siirtyneet käyttämään mikropalveluarkkitehtuuria, jossa monoliittisen sovelluksen yksittäisen komponentin voi toteuttaa yksittäisenä mikropalveluna (Kuva 2). Tällöin sovelluksen ylläpidettävyys helpottuu, koska koodimuutokset voidaan tehdä yksittäiseen mikropalveluun. Näin muutokset eivät halvaannuta muun sovelluksen käyttöä. Muutoksia tehdessä yksittäinen mikropalvelu voidaan päivittää ja käynnistää uudelleen sen sijaan, että koko sovellus käynnistetään uudestaan.

TechTargetin (2024) mukaan mikropalveluarkkitehtuurissa on useita hyötyjä:

1. Tuki modulaarisille sovelluksille. Yksittäistä mikropalvelua voidaan muuttaa itsenäisesti vaikuttamatta muihin sovelluksen osiin.
2. Modulaarinen rakenne soveltuu paremmin jatkuvaan tai iteratiiviseen kehitykseen ja modernin käytännön mukaisiin ketteriin sovelluskehityksen menetelmiin.
3. Skaalautuvuus, koska jokainen mikropalvelu saa omat resurssinsa, mitkä voidaan skaalata itsenäisesti muista komponenteista riippumatta. Resurssit voidaan skaalata vastaamaan kehittyviä vaatimuksia.
4. Jokainen mikropalvelu keskustelee tarkasti määritellyn API-rajapinnan kautta.
5. Jokaisella mikropalvelulla on oma tietokanta, josta kyselyitä suoritetaan. Tämä ratkaisu parantaa järjestelmän viivettä ja nopeutta.
6. Helpottaa julkaisujen ja päivitysten tekemistä.



Kuva 2. Mikropalveluarkkitehtuuri

2.2 Pääsynhallinta

Pääsynhallinta on kokoelma käytäntöjä ja mekanismeja, jotka sallivat tai kieltävät resurssin tai jonkin toiminnon käytön (Chin & Older 2010, 1). Mikropalveluiden käyttöoikeuksien hallintaan liittyy looginen pääsynhallinta. Looginen pääsynhallinta on prosessi, jossa hallitaan ja säännöstellään käyttäjien pääsyä tietoverkkoihin ja järjestelmiin. Looginen pääsynhallinta varmistaa käyttäjien luvitukset ja oikeudet sekä niiden mahdollistamat pääsyt erilaisiin resursseihin ja toimintoihin järjestelmissä. (Cybersecurity-Automation n.d.; IBM 2021)

Loogisessa pääsynhallinnassa käytetään autentikointia eli todennusta ja valtuutusta (authorization) mekanismeina pääsynhallinnan toteuttamiseen. Mekanismeilla toteutetaan pääsynhallintakäytäntöjä (access control policies). Niiden tavoitteena on vahvistaa käyttäjän identiteetti, identiteettikohtainen käyttöoikeustaso ja käyttöoikeustason sisältämät luvitukset. Käyttöoikeustason määrittämiseen voidaan käyttää erilaisia muuttujia käyttäjän laitteen, sijainnin, roolin tai muun ominaisuuden perusteella. Pääsynhallintakäytännöt suojaavat järjestelmissä olevia luottamuksellisia tietoja. Lähtökohtaisesti pääsynhallintakäytäntöjen tulee noudattaa Zero Trust- tietoturvamallia, mikä tarkoittaa, että jokaisen käyttäjän aloittama tapahtuman oletetaan olevan epäluotettava ja tämän takia järjestelmät todentavat ja tarkastavat tapahtumaan liittyvät valtuutukset ennen kuin käyttäjälle annetaan pääsyä mihinkään resursseihin. (Kaspersky n.d.)

Todennus tarkoittaa käytäntöjä, joissa käyttäjän identiteetti vahvistetaan käyttäjän antamilla tunnistetiedoilla. Tämä toimii myös ensimmäisenä tarkastuspisteenä pääsynhallinnassa. Todennuksen tavoitteena on luotettavasti valvoa pääsyä resursseihin ja järjestelmiin.

Onnistuneen todennuksen jälkeen käyttäjä pitää pystyä valtuuttamaan haluttuun resurssiin tai järjestelmään. Valtuutus kertoo käyttäjälle määritellyt tasot ja ominaisuudet, joiden perusteella käyttäjä pääsee tiettyihin resursseihin tai toimintoihin käsiksi. Valtuutus kertoo, mitä toimintoja ja millä laajuudella käyttäjä voi suorittaa. Valtuutus vastaa kysymykseen ”Mitä käyttäjä saa tehdä?” eli onko todennetulla käyttäjällä tarvittavat oikeudet halutun toiminnon suorittamiseen. (Mashfej 2023)

Pääsynhallinnan peruskäsitteitä ovat subjekti, objekti ja pääsyoikeus. Subjekti on entiteetti, joka käsittelee objekteja. Subjekti voi olla esimerkiksi käyttäjä, sovellus tai toinen järjestelmä. Objekti on resurssi, johon pääsyä halutaan hallita ja rajoittaa kuten dokumentti, hakemisto, ohjelma tai muu järjestelmä. Pääsyoikeus on verbi, joka määrittelee, millä tavoin subjekti voi käsitellä tai päästä käsiksi objektiin. Pääsyoikeus kertoo, voiko subjekti esimerkiksi lukea tai poistaa tiedoston vai kenties suorittaa hakuja hakemistoon tai järjestelmään. (Frontegg 2025; Meghanathan n.d.)

Pääsynhallinnan toteuttaminen järjestelmissä vaatii tietoturvan huomioon ottamista. Monoliittisessa järjestelmässä perinteisesti yksi valtuutuspiste hallitsee pääsyä kaikkiin järjestelmän resursseihin ja ne ovat yleensä helpompia toteuttaa tietoturvan kannalta. Mikropalveluarkkitehtuuria käyttävässä järjestelmässä on useampi mikropalvelu, joissa jokaisessa voi olla suojattavia resursseja. (Manor 2024) Jokaisella mikropalvelulla voi olla hyvin erilaiset ja itsenäiset vaatimukset valtuutusta varten ja näiden pääsynhallinnan vaatimusten toteuttaminen johdonmukaisesti voi olla erittäin haastavaa (Lombarte 2024). Mikropalvelut keskustelevat API:en kautta ja jokainen API-rajapinta on myös mahdollinen kohde järjestelmään kohdistuvalle hyökkäykselle. Tämän takia järjestelmän valvonta on haasteellista, koska hyökkääjä voi etsiä yksittäistä palvelua ilman että järjestelmä hälyttää sovelluksen olevan hyökkäyksen kohteena.

2.3 Haasteet mikropalveluympäristössä

Mikropalvelut keskustelevat API-ohjelmointirajapintojen välityksellä käyttäen HTTP-kutsuja tai muita kevyitä viestintäprotokollia. Kun mikropalvelujen määrä kasvaa, lisääntyy myös niiden välinen kompleksisuus. Jokainen mikropalvelu toteuttaa tietyn toiminnon tai osan liiketoimintalogiikasta ja jokainen mikropalvelussa liikkuva käyttöoikeuspyyntö on pystyttävä todentamaan ja hyväksymään, mikä luo haasteita järjestelmän toteuttamiseen: (Frontegg 2022)

1. Keskitetty riippuvuus: Todennus ja valtuutuslogiikka on käsiteltävä erikseen jokaisen mikropalvelun toimesta. Toteuttamiseen voidaan käyttää samaa koodia jokaisessa mikropalvelussa, mutta tämä edellyttää saman ohjelmointikielen tai kehyksen tukea jokaisessa yksittäisessä mikropalvelussa.
2. Yhden vastuun periaatteen rikkominen: Yksittäisen mikropalvelun pitäisi toteuttaa vain yhtä tarkasti määriteltyä toimintoa. Jos jokaiseen mikropalveluun liitetään globaali todennus – ja valtuutuslogiikka, ne suorittavat tällöin myös lisätoiminnon, joka tekee mikropalvelusta vähemmän luotettavan ja vaikeamman hallita sekä ylläpitää.
3. Monimutkaisuus: Todennus ja valtuutus mikropalveluissa voivat johtaa hyvin monimutkaisiin skenaarioihin. Yksittäisen mikropalvelun käyttäjänä voi olla käyttäjä, toinen mikropalvelu tai kolmannen osapuolen järjestelmä, joka tekee pääsynhallinnan toteuttamisesta ja ylläpidosta hankalaa.

Mikropalveluissa todentamista käytetään tunnistamaan loppukäyttäjiä, toisia mikropalveluita tai ulkoisia, kolmannen osapuolen palveluita, jotka kaikki yrittävät saada pääsyä yksittäiseen mikropalveluun.

Todennus on yksinkertaisempi toteuttaa monoliittisessa arkkitehtuurissa. Perinteinen monoliittinen sovellus koostuu tyypillisesti käyttöliittymästä, palvelinpuolen sovelluksesta ja tietokannasta, jotka

ovat tiivisti integroituna toisiinsa. Tällöin sovelluksessa on kaikki sen tarvitsemat resurssit, eikä todennusta ole tarpeen suorittaa sovelluksen sisällä. Todennus voidaan suorittaa, kun käyttäjän on päästävä käyttämään sovellusta. (Frontegg 2022)

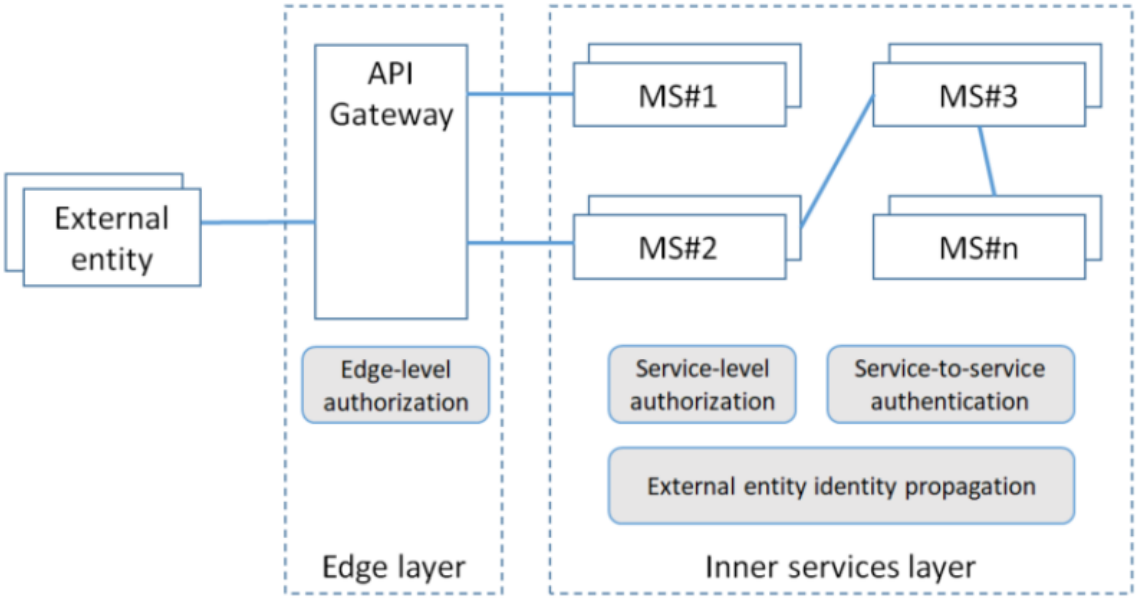
Mikropalveluarkkitehtuuria käyttävä sovellus koostuu puolestaan useasta itsenäisestä mikropalvelusta, jotka keskustelevat niihin integroitujen API-rajapintojen kautta. Kun mikropalvelut kommunikoivat keskenään on niiden väliset pyynnöt pystyttävä todentamaan. Todennus varmistaa, että vain asianmukaisilla mikropalveluilla ja käyttäjillä on pääsy jokaiseen mikropalveluun. Eli erona monoliittiseen sovellukseen on se, että jokainen pyynnön tehnyt loppukäyttäjä on pystyttävä tunnistamaan ja todentamaan kaikissa yksittäisissä palveluissa.

3 PÄÄSYNHALLINNAN TOTEUTUKSIA MIKROPALVELUARKKITEHTUURISSA

3.1 Arkkitehtuuritason mallit pääsynhallintaan

Toimeksiantaja halusi kehittää pääsynhallintajärjestelmän mikropalveluarkkitehtuuria hyödyntävään järjestelmäänsä. Ensimmäisenä oli järkevää lähteä tutkimaan, millaisia arkkitehtuuritason ratkaisuja ja malleja pääsynhallintaan on käytetty erilaisia mikropalveluarkkitehtuuria hyödyntävissä järjestelmissä. Tähän löytyi hyväksi lähdemateriaaliksi Barabanovin ja Makrushin tekemä tutkimus, jossa he olivat tutkineet näitä ratkaisuja ja kertovat näistä käytetyimpiä ja suosituimpia malleja. Toinen hyvä lähde oli Owaspin tekemä dokumentaatio samasta aiheesta ja näiden sisällön todettiin olevan sellaista, jotka vastasivat toimeksiantajan järjestelmään liittyviin kysymyksiin. Tutkimukset jakoivat arkkitehtuurimallit reuna- ja palvelutason mukaan.

3.1.1 Reunatason valtuutus



Kuva 3. Authentication and authorizations subfunctions in microservice-based systems. (Barbanov & Makrushin 2020)

Reunataso tarkoittaa järjestelmän kohtaa, joka toimii alueena asiakasohjelman ja sisäisten palvelujen välillä. Tyypillisesti reunatasolla käytetään API-yhdyskäytävää tai käänteistä välityspalvelinta (reverse proxy) (GeeksforGeeks 2024). Toisin sanoen reuna on järjestelmän kohta, joka on ulkomaailmaan auki ja jonka kautta järjestelmään saapuu liikenne.

Yksinkertaisimmillaan valtuutus voidaan toteuttaa pelkästään järjestelmän reunatasolla. Tällöin reunatason valtuutus hoidetaan käyttämällä API-yhdyskäytävää. API-yhdyskäytävä on API hallintatyökalu, joka sijaitsee järjestelmän reunalla ja välittää asiakasohjelmilta tulevan liikenteen yhdyskäytävän takana oleviin mikropalveluihin (Red Hat 2019). Tyypillisesti se käsittelee asiakasohjelman lähettämän pyynnön kutsumalla useita yhdyskäytävän takana olevia mikropalveluita ja yhdistää näiltä

saamansa vastaukset. Yhdyskäytävä hoitaa myös muita tehtäviä kuten viestintäprotokollien vaihtaminen järjestelmien välillä. (F5 n.d.)

API-yhdyskäytävä on asennettu palvelimelle API-liittymien eteen. Asiakasohjelmistot muodostavat yhteyden yhdyskäytävään, joka esittää palvelut yhden IP-osoitteen alla. Reititys määrittää mihin API-palveluun asiakasohjelmiston lähettämät pyynnöt välitetään. (HAProxy n.d.)

Nykyaikaisissa yrityksissä voi olla käytössä tuhansia API-rajapintoja. Yhdyskäytävä tarjoaa keskitetyn pääsypisteen API-rajapintoihin ja standardoidun rajapinnan, joka auttaa hallitsemaan ja reitittämään API-kutsuja yhdestä keskitetystä paikasta. On myös mahdollista käyttää useampaa yhdyskäytävää rinnakkain, jolloin voidaan käyttää erilaisia suojausprotokollia ja standardeja käyttötapauksen mukaan. (Jackson & Goodwin 2024) Useamman yhdyskäytävän käyttäminen tukee ”backends to frontends” ajattelumallia, jossa jokaiselle käyttöliittymälle tai erilaiselle laitteelle luodaan oma erillinen backend-palvelu. (Microsoft n.d.a)

Yhdyskäytävä vastaanottaa kaikki järjestelmään saapuvat pyynnöt ja todentaa sekä käsittelee ne määriteltyjen käytäntöjen mukaisesti. Sen jälkeen yhdyskäytävä ohjaa ne oikeille taustapalveluille eli resurssipalvelimille. Resurssipalvelimilta saapuneet pyynnöt koostetaan ja palautetaan pyynnön tehneelle asiakasohjelmalle. Vaikka yhdyskäytävä välittäisi pyynnön useammalle resurssipalvelimelle, palautetaan yhdyskäytävästä silti vain yksi vastaus, joka sisältää kaiken asiakasohjelman pyytämän tiedon. (Jackson & Goodwin 2024) Yhdyskäytävä voi lisäksi hoitaa useita muita toimintoja. Se voi hoitaa tietoturvaan liittyviä toimintoja kuten autentikointia, IP-osoitteiden ja pyyntöjen rajoittamista sekä TLS-salauksen toteuttamisen. Palvelun ylläpitoa ja kehittämistä helpottavat toiminnot kuuluvat myös yhdyskäytävän tehtäviin. Näitä ovat esimerkiksi pyyntöjen lokittaminen ja valvonta sekä saapuneiden pyyntöjen tallentaminen välimuistiin. Yhdyskäytävä voi myös toimia suoraan verkkosovelluksen palomuurina. Suosittuja palveluita yhdyskäytävän toteutukseen ovat Nginx ja HAproxy jotka molemmat tarjoavat OSI-mallin seitsemännen tason eli sovelluskerroksen reitityksen, kuormantasauksen ja TLS (Kuljetustason) salausprotokollan. Molemmat ovat avoimen lähdekoodin palveluita, joista on saatavilla maksulliset versiot, jotka sisältävät lisäominaisuuksia. Molemmat ovat myös hyvin suorituskykyisiä ja niitä on mahdollista laajentaa kolmansien osapuolien tarjoamilla lisäkirjastoilla tai itsetehdyillä Lua-skripteillä. (Microsoft n.d.b)

Yhdyskäytävä keskittää valtuutuksen tarkastamalla reunatasolle saapuneet pyynnöt. Tällöin ei ole tarvetta tarjota erillistä todennus ja valtuutushallintaa jokaiselle yksittäiselle mikropalvelulle yhdyskäytävän jälkeen. Tässä mallissa (Barabanov & Makrushin 2020) NIST suosittelee ottamaan käyttöön lieventäviä käytäntöjä kuten yhteinen keskitetty todennus, jotta suorat tuntemattomasta lähteestä saapuvat pyynnöt ja yhteydet mikropalveluihin saadaan estettyä. Tämä vähentää myös riskejä mitkä liittyvät mahdolliseen API-yhdyskäytävän ohittamiseen.

Tämän mallin haasteiksi ja huonoiksi puoliksi mainintaan: (Barabanov & Makrushin 2020)

- Hankala ylläpitää: Valtuutussääntöjen ja päätösten toteutus sekä valvonta on haasteellista useasta mikropalvelusta koostuvassa järjestelmässä, jossa käyttäjille on määritelty useita roolia ja pääsynhallinnan sääntöjä. Näiden pohjalta tehtyjen pääsynhallintapäätösten toteutus yhdyskäytävässä voi olla hankala ylläpitää ja toteuttaa.
- Yhdyskäytävästä tulee ainoa pääsynhallintapäätösten päätäntäpiste, joka rikkoo ”defense in depth” periaatetta.

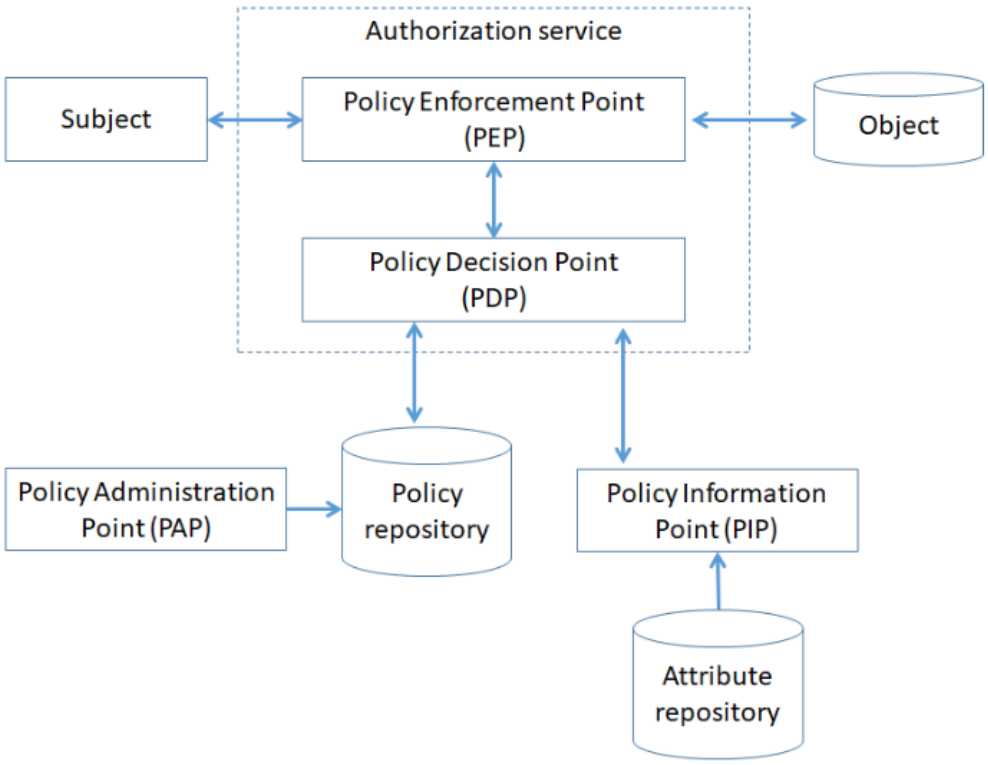
- Vähemmän turvallinen: Jos hyökkääjä pääsee yhdyskäytävän läpi, on hänellä vapaa pääsy yhdyskäytävän takana oleviin mikropalveluihin (Frontegg 2022).

Useimmissa tapauksissa päädytään ottamaan käyttöön valtuutus sekä reuna- että palvelutasolla (Kuva 3). Tällöin reunatason valtuutus hoidetaan vain karkealla tasolla. Reunatasolla todennukseen voidaan käyttää tokeneita, jotka siirretään HTTP-pyyntöjen otsikkotiedoissa tai käyttäen mTLS - protokollaa.

3.1.2 Palvelutason valtuutus

Palvelutaso tarkoittaa reunatason jälkeistä kerrosta, jossa yksittäiset mikropalvelut ovat. Palvelutasolla pääsynhallinta koostuu useammasta komponentista (Kuva 4), joille NIST on antanut seuraavat termit ja määritelmät (Barabanov & Makrushin 2020):

- Policy Administration Point (PAP): Käyttöliittymä pääsynhallintasääntöjen luomiselle ja hallinnoimiselle.
- Policy Decision Point (PDP): Tekee päätöksen pääsynhallintasääntöjen perusteella, onko pyynnön tekijällä pääsyä resurssiin tai järjestelmään. Antaa pyyntöön vastauksena kyllä tai ei.
- Policy Enforcement Point (PEP): Panee täytäntöön pääsynhallintapäätökset vastauksena kohteen pyyntöön päästä käsiksi pyydettyyn resurssiin. Esimerkiksi jos PDP vastaa kyllä, käyttäjä päästetään käsiksi pyydettyyn sisältöön.
- Policy Information Point (PIP): Toimii pääsynhallintasääntöjen tallennus- ja hakupisteenä PDP:lle
- Subject: Pyynnön lähettänyt henkilö.
- Object: Resurssi tai muu kohde, jota subjekti yrittää käsitellä.
- Policy Repository: Paikka minne kaikki pääsynhallinnan säännöt tallennetaan. Esimerkki säännöstä: Työntekijä pääsee ovesta sisään kello 07-12 välillä.
- Attribute Repository: Tietokanta, jossa on kaikki subjektien ominaisuudet. Esimerkiksi työntekijän nimi, rooli ja osasto.



Kuva 4. Access control management functional points. (Barbanov & Makrushin 2020)

Käytännössä eri komponenttien välinen työnjako näyttää tältä:

Tilanne: Jarmo (subject) tahtoo päästä sisään toimistolle (object) kello 9:00.

PAP: Sääntö: "Työntekijöille pääsy toimistolle vain 8:00 – 18:00. välisenä aikana"

PDP: Tarkastaa säännön ja päästää Jarmon toimistolle.

PEP: Toimiston ovi aukeaa päätöksen perusteella

PIP: Tarjoaa tietoa, kuten Jarmon nimen ja vallitsevan kellonajan.

Policy Repository: Tallentaa ja säilyttää säännöt, jotka PDP tarkastaa

Attribute Repository: Tallentaa subjektin (Jarmo) tiedot kuten yksilöivä id, työajat jne.

Palvelutason valtuutus antaa jokaiselle mikropalvelulle enemmän mahdollisuuksia hallita ja toteuttaa pääsynhallinnan sääntöjä. Barabanovin ja Makrushin tutkimus (2020) jakoi palvelutason valtuutuksen mallit seuraavasti:

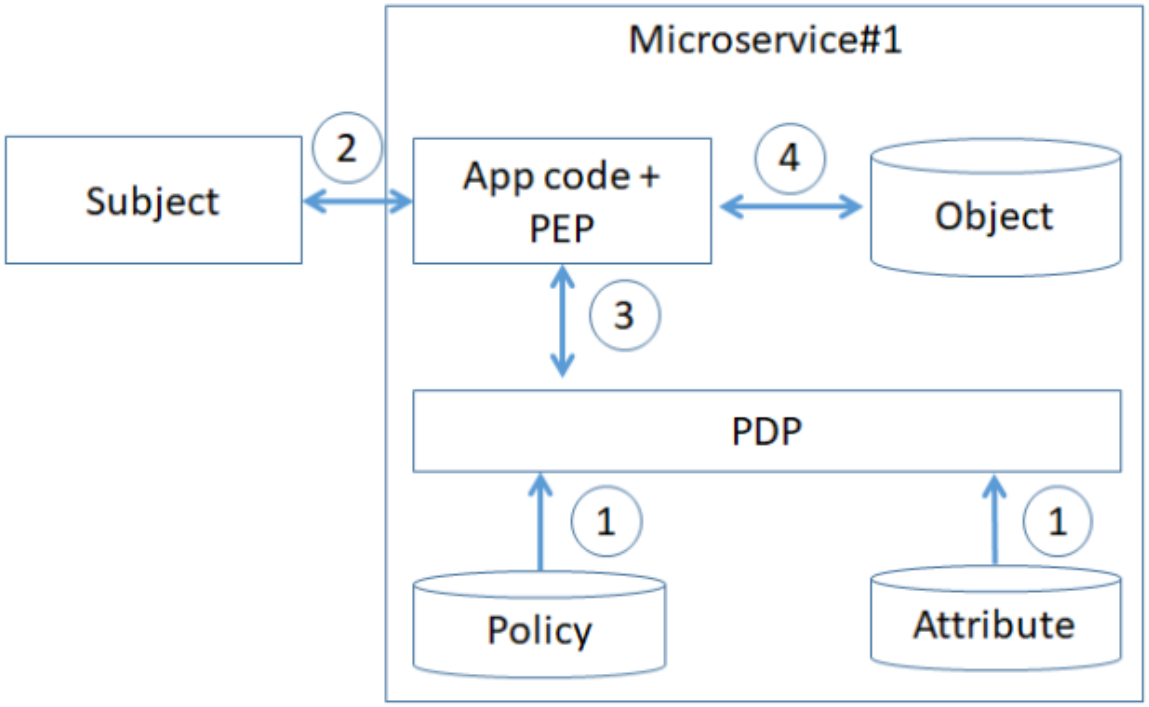
1. Hajautettu malli
2. Keskitetty malli yhdellä päätöksentekopisteellä (PDP)
3. Keskitetty malli sulautetulla päätöksentekopisteellä (PDP)

3.1.3 Palvelutason valtuutus: Hajautettu malli

Hajautetussa mallissa päätäntäpiste (PEP) ja päätöksentekopiste (PDP) toteutetaan yksittäiseen mikropalveluun eli API-rajapintaan kooditasolla (Kuva 5). Kaikki pääsynhallinnan säännöt ja ominaisuudet määritellään ja tallennettaan suoraan jokaiseen yksittäiseen mikropalveluun. Tällöin käytössä oleva ohjelmistokehys määrittää, miten valtuutus voidaan toteuttaa mikropalvelutasolla. Tässä mallissa on otettava huomioon, että lähdekoodi on päivitettävä aina, kun pääsynhallinnan sääntöjä halutaan muokata, jolloin pääsynhallintasääntöjen ylläpitäminen vaikeutuu.

Hajautetussa mallissa yksittäinen mikropalvelu vastaanottaa pyynnön ja pyynnön mukana lähetetään myös valtuutukseen tarvittava metadata, kuten loppukäyttäjän konteksti ja pyydetyn resurssin tunnistetieto. Mikropalvelu käsittelee pyynnön ja tekee pääsynhallintasäännön perusteella päätöksen, onko loppukäyttäjällä oikeutta suorittaa pyynnön mukainen toiminto.

Mallista löytyy kuitenkin huonoja puolia ja rajoitteita. Hajautetussa mallissa yksittäisen mikropalvelun kehittäjillä täytyy olla selkeä käsitys käytetyn ohjelmistokehyksen turvallisuuteen liittyvistä ominaisuuksista ja osattava hyödyntää niitä. Lisäksi kehittäjillä täytyy olla selkeä käsitys pääsynhallinnan säännöistä ja käyttäjäkontekstiin liitetystä rooleista sekä ominaisuuksista, jotka vaikuttava pääsynhallintapäätöksiin. Hajautetun mallin ylläpitäminen ja konfigurointi on manuaalista käsin tehtävää työtä ja kasvattaa virheiden mahdollisuutta. Jos järjestelmä koostuu useasta mikropalvelusta, on niiden ylläpitäminen pääsynhallinnan kannalta todella hankalaa. Lisäksi hajautetussa mallissa pääsynhallintaan liittyvien toimintojen sijoittaminen mikropalvelun lähdekoodiin vaatii tarkkaa testaamista ohjelmistovirheiden välttämiseksi. Huonojen puolien ja rajoitteiden vastapainoksi hajautettu malli sallii tarkkojen pääsynhallintasääntöjen luomisen ja käyttöönottamisen, koska valtuutussäännöt ovat domain eli mikropalvelukohtaisia. (Barabanov & Makrushin 2020)



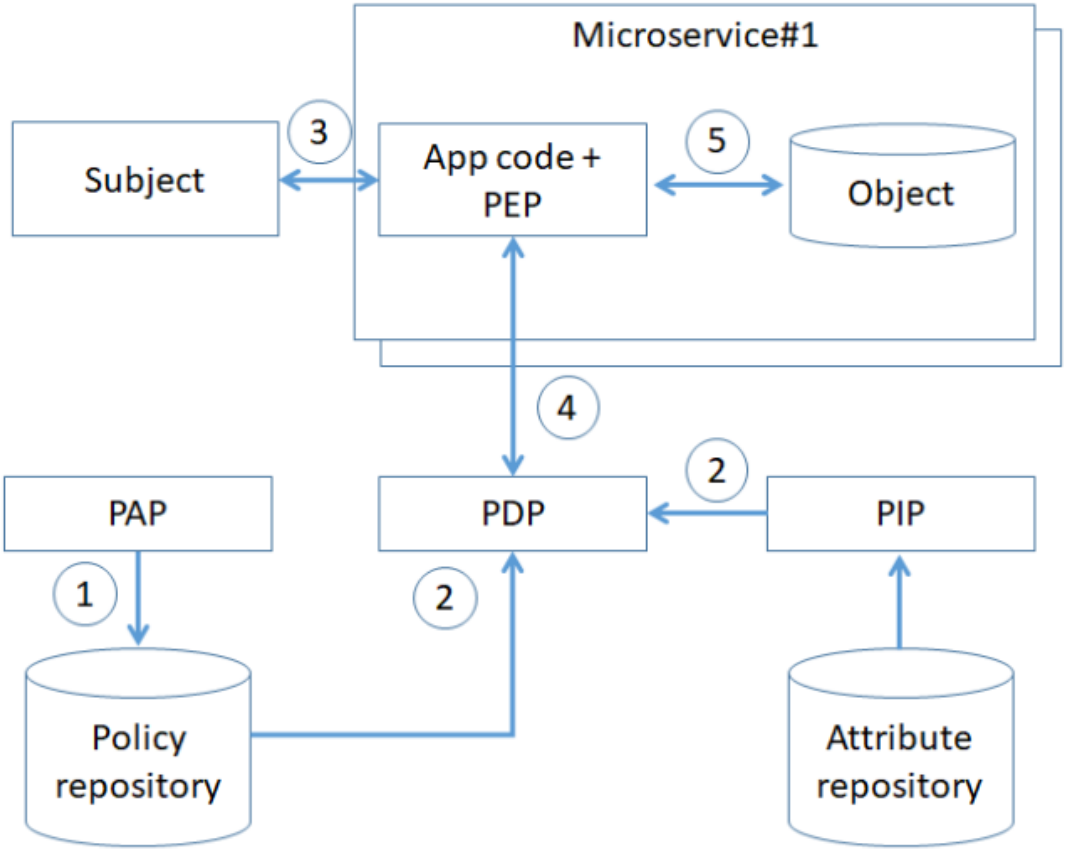
Kuva 5. Decentralized pattern high-level architecture (Barabanov & Makrushin 2020)

3.1.4 Palvelutason valtuutus: Keskitetty malli yhdellä päätäntäpisteellä

Keskitetyssä mallissa pääsynhallinnan säännöt määritellään, tallennetaan ja arvioidaan keskitetyssä PDP-palvelussa. Keskitetty PDP-palvelu voi olla esimerkiksi oma mikropalvelunsa (Kuva 6). Keskitetyssä mallissa subjekti kutsuu mikropalvelun päätepistettä (endpoint) ja mikropalvelu kutsuu keskitettyä PDP-palvelua. PDP luo pääsynhallintasääntöjen pohjalta kyllä tai ei päätöksen verrattuaan subjektin tekemää pyyntöä pääsynhallinnan sääntöihin ja subjektiin liitettyihin ominaisuuksiin.

Etuna tässä mallissa on turvallisuus, koska pääsynhallinnan sääntöjä voidaan muokata keskitetyssä palvelussa ilman koskemista mikropalvelun lähdekoodiin. Tällöin kehittäjien ei tarvitse välittää sääntöjen muutoksista. Uusi luotu pääsynhallinnan sääntö otetaan käyttöön kaikissa mikropalveluissa. Keskitetyssä mallissa pääsynhallintasääntöjä ei toteuteta mikropalvelun lähdekoodissa. Sen takia pääsynhallintasääntöjen kirjoittamiseen on otettava käyttöön uusi kieli tai merkintätapa, jota käytetään PDP-moduulissa. Esimerkkeinä tutkimuksessa (Barabanov & Makrushin 2020) on esitelty Extensible Access Control Markup Language (XACML) ja Next Generation Access Control (NGAC) -kieliä. Näistä XACML pidetään epäonnistuneena sen hankalan syntaksin vuoksi.

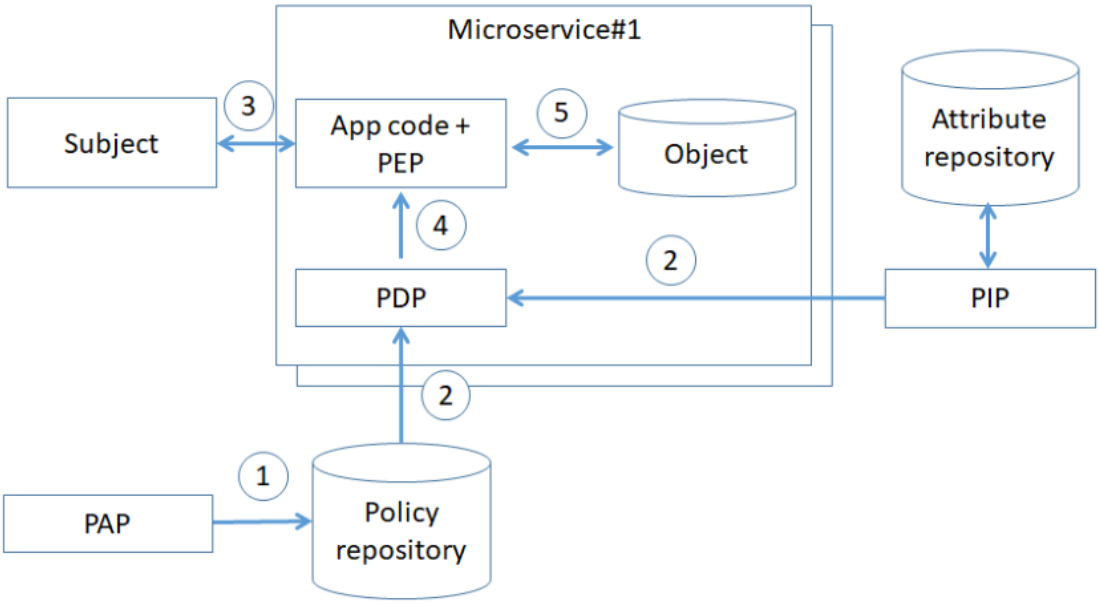
Keskitetyn järjestelmän huonona puolena on viiveen kasvu järjestelmän pyynnöissä, koska jokainen resurssia kysyvä pyyntö tarkastetaan päätöspisteeltä. Tämän kiertämiseksi voidaan käyttää välimuistia pääsynhallintasääntöjen tallentamiseksi mikropalvelussa, jolloin sääntöjä ei tarvitse hakea joka kerta uudelleen päätöspisteeltä. Vaikka sääntöjä tallennettaisiinkin, päätöspisteen (PDP) pitää pystyä käsittelemään huomattavan paljon pyyntöjä. Tässä ratkaisussa pitää ottaa myös huomioon palvelun korkea käyttöaste. Jos PDP putoaa pois käytöstä, koko mikropalvelukokokonaisuus menee vikatilaan koska pyyntöjä ei voida tarkastaa. Tämän vuoksi on suositeltavaa käyttää rinnalla toista todennuspalvelua kuten API-yhdyskäytävää, jolloin palvelun saatavuus ja turvallisuus paranee. (Barabanov & Makrushin 2020)



Kuva 6. Centralized pattern with single PDP high-level architecture (Barabanov & Makrushin 2020)

3.1.5 Palvelutason valtuutus: Keskitetty malli sulautetulla päätäntäpisteellä

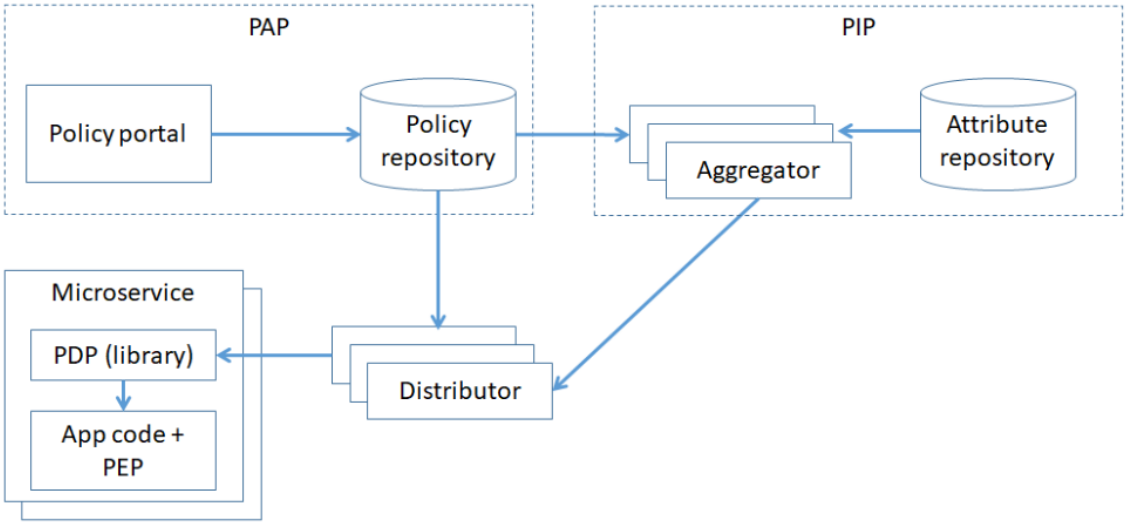
Tässä mallissa pääsynhallinnan säännöt määritellään keskitetysti, mutta ne tallennettaan ja arvioidaan jokaisessa mikropalvelussa (Kuva 7). Ensimmäiseksi sulautetulla päätäntäpisteellä olevalla keskitetyllä mallilla ylläpitäjä määrittää pääsynhallinnan säännöt PAP-käyttöliittymässä. Tämän jälkeen luodut säännöt ja sääntöjen tarkastukseen tarvittavat ominaisuudet tallennettaan jokaisen mikropalvelun erilliseen PDP-moduuliin. Sulautetun päätäntäpisteen mallissa pyynnön kohteena oleva mikropalvelu kutsuu kooditasolla PDP-moduulia ja moduuli tekee pääsynhallintasäännön mukaisen päätöksen vertaamalla sääntöä pyynnön sisältämiin parametreihin ja pyynnön tekijän ominaisuuksiin. PDP-moduuli voidaan toteuttaa kooditasolla mikropalveluun sisäänrakennettuna kirjastona tai sivuvaunua (sidecar) hyödyntävänä palveluverkkoarkkitehtuurina (service mesh). (Barabanov & Makrushin 2020)



Kuva 7. Centralized pattern with embedded PDP high-lever architecture (Barabanov & Makrushin 2020)

Sulautetun päätäntäpisteen mallia käytetään esimerkiksi korkean käyttöasteen palveluissa kuten Netflixissä. Tämän mallin hyvät puolet ovat samat kuin yksittäisen päätäntäpisteen mallissa, mutta erillinen sulautettu PDP-moduuli jokaisessa mikropalvelussa vähentää järjestelmän viivettä. Tämän mallin kanssa on suositeltavaa käyttää lisäksi myös reunatason valtuutusta, jotta vältetään yhden päätäntäpisteen periaatetta ja saadaan vahvistettua defense in depth -periaatetta.

PDP-moduuli voidaan toteuttaa mikropalvelun lisäkirjastolla tai sivuvaununa käyttäen palveluverkkoa (Kuva 8). Mahdollisten latenssi- ja yhteysongelmien vuoksi on suositeltavaa toteuttaa PDP-moduuli samassa Kubernetes-kontissa mikropalvelun kanssa. Sulautettu PDP-moduuli tallentaa valtuutus-säännöt ja siihen liittyvät tarpeelliset tiedot välimuistiin valtuutuksen täytäntöönpanon aikana. Tällä saavutetaan ulkoisten riippuvuuksien minimoiminen ja alhaisempi viive. Välimuistiin tallentamisen riskeinä ovat kuitenkin vanhentuneiden sääntöjen toteutus ja täten myös pääsynhallintarikkomusten toteutuminen. (Barabanov & Makrushin 2020)



Kuva 8. Centralized pattern with embedded PDP (Barabanov & Makrushin 2020)

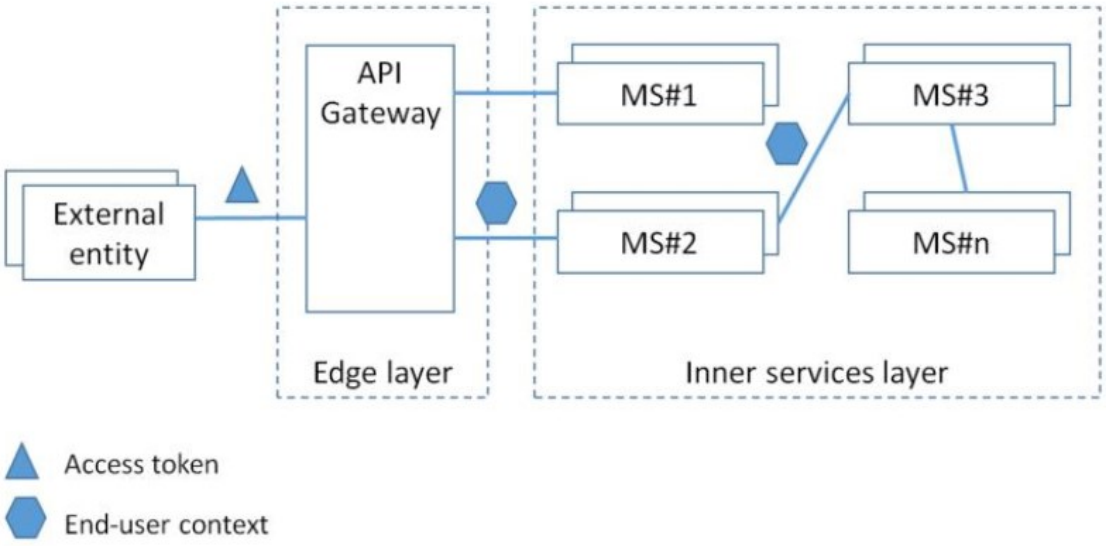
Palveluverkko on infrastruktuurikerros, joka helpottaa palveluiden välistä viestintää palvelujen tai mikropalvelujen välityspalvelimen avulla (Khatri & Khatri 2020, 39). Palveluverkon tarkoituksena on hoitaa järjestelmän sisäistä liikennettä palveluiden välillä. Tämä eroaa API-yhdyskäytävästä, joka hoitaa ulkoverkosta tulevaa liikennettä järjestelmän sisälle (Polencic 2019). Palveluverkko on suosittu menetelmä hajautettujen palvelujen hallintaan ja komponenttiviestinnän järjestämiseen. Se käyttää välityspalvelinta, joka sieppaa ja hallitsee palvelujen välistä viestintää. Tätä välityspalvelinmekanismia voidaan kutsua sivuvaunuksi tai sivuvaunukontiksi. Palveluverkoista ja sivuvaunuista on tullut olennainen osa mikropalveluarkkitehtuuria, koska ne voivat ohjata palveluiden välistä viestintää, käsitellä pyyntöjen reititystä, tarjota vikasietoisuutta sekä muita ominaisuuksia, jotka helpottavat palveluiden ylläpitoa. Sivuvaunut voivat tarjota suoraa tukea mikropalvelujen hallintaan (Kanjilal 2022). Sivuvaunu ja pääsovellus ovat löyhästi kytkettyjä toisiinsa. Sivuvaunu liittää pääsovellukseen laajennuksia tai lisäominaisuuksia. Sivuvaunun käytöllä voidaan saavuttaa etuja, kuten siirtää ominaisuuksia, jotka eivät liity pääsovelluksen liiketoimintalogiikkaa, yhteiseen infrastruktuuriin, mikä vähentää pääsovelluksena olevan mikropalvelun koodin monimutkaisuutta. Sivuvaunun etuina on vähentää toistuvan määritystiedostojen ja koodin kirjoittamisen kolmannen osapuolen komponeteille, mikä puolestaan vähentää päällekkäistä koodia mikropalvelussa sekä sovelluskoodin ja sen taustalla olevan alustan välistä kytköstä toisiinsa. (MOSN 2024)

Kubernetes-palveluverkko on infrastruktuurikerros, joka on suunniteltu hallitsemaan ja tarkkailemaan Kubernetes-klusterin mikropalvelujen välistä viestintää. Palveluverkon tavoitteena on parantaa monimutkaisen hajautetun sovelluksen muodostavien mikropalvelujen yleistä luotettavuutta, turvallisuutta ja havaittavuutta. Palveluverkon toteuttamiseen on useita suosittuja avoimen lähdekoodin tekniikoita Kubernetes-klusterissa kuten Istio, Linkerd, Consul Connect ja NGINX Service Mesh. (Tigera n.d.)

3.2 Ulkoisen identiteetin käyttö ja levittäminen

Palvelun reunataso levittää käyttäjäkontekstin pyyntöjen mukana alaspäin palvelutasolle. Palvelutason tarvitsee käyttäjäkontekstin voidakseen vahvistaa pääsynhallintaa ja toteuttaa pääsynhallintaan

liittyviä sääntöjä ja päätöksiä. Yksinkertaisimmillaan ulkoisen identiteetin levittäminen voidaan toteuttaa käyttämällä uudestaan reunatasolle saapunut tunnistetieto tai token ja lähettämällä se pyyntöjen mukana palvelutasolle. Tähän käytäntöön liittyy riski ulkoisen tunnistetiedon tai tokenin vuotamiseen, joka johtaa puolestaan järjestelmään kohdistuvan hyökkäyspinta-alan kasvamiseen.



Kuva 9. Send user context as a clear or seld-signed data structures (Barabanov & Makrushin 2020)

Ulkoisen identiteetin levittämiseen Barabanov ja Makrushin (2020) esittivät kaksi mallia. Ensimmäisessä mallissa ulkoinen identiteetti lähetetään selkeinä tai itse allekirjoitettuina tietorakenteina. Mikropalvelu vastaanottaa saapuvasta pyynnöstä talteen ulkoisen entiteetin eli käyttäjän identiteetin ja luo tämän pohjalta tunnisteen, esimerkiksi JSON Web Tokenin. Tunnisteeseen liitetään käyttäjän valtuutukseen tarvittavat tiedot, jotka lähetetään tunnisteen mukana palvelutasolle. Palvelutasolla pyynnön vastaanottavan palvelun on luotettava pyynnön tehneeseen palveluun. Pyynnön tehnyt palvelu voi kuitenkin rikkoa pääsynhallintasääntöjä asettamalla minkä tahansa käyttäjän tunnistetiedon HTTP-otsikkotietoihin. Menetelmää suositellaan käyttämään luotettavissa ympäristöissä, joissa jokaisen mikropalvelun on kehittänyt luotettava kehitystiimi käyttäen turvallisia kehityskäytäntöjä.

Toisena mallina Barabanov ja Makrushin (2020) kertovat luotetun myöntäjän allekirjoittaman tietorakenteen käyttämisestä. Tässä mallissa ulkopuolelta saapunut pyyntö tunnistetaan todennuspalvelussa järjestelmän reunalla. Todennuspalvelu luo ja allekirjoittaa tietorakenteen ulkoisen identiteetin perusteella, joka sisältää käyttäjän tunnistetietoja, roolit, käyttöoikeudet tai muita tunnistautumiseen tarvittavia tietoja ja lähettää tämän rakenteen mikropalveluihin palvelutasolle.

Näiden mallien pohjalta voidaan kehittää myös omia malleja. Esimerkiksi Netflix käyttää palveluissaan kehittämäänsä "Passport" nimistä mallia. Mallissa käyttäjän tunnistetiedot ja ominaisuudet kapseloidaan tokenin sisään ja lähetetään yhdyskäytävästä palvelutason mikropalveluille. Mallin toimimista kuvattiin esimerkillä (OWASP n.d.):

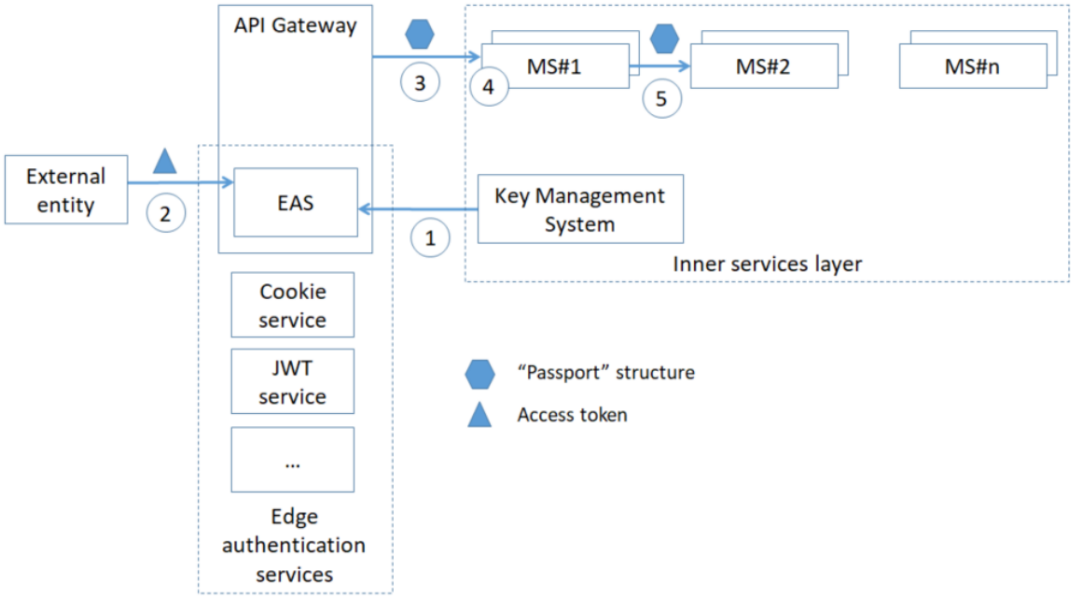
1. Reunatason valtuutuspalvelu (The Edge Authentication Service (EAS)) vastaanottaa salausavaimen Avaintenhallinnasta (Key Management System)

2. EAS vastaanottaa reunatasolle saapuneen evästeen, JSON Web Tokenin tai muun tunnisteen saapuvasta pyynnöstä

4. EAS purkaa saapuneen tokenin salauksen, selvittää pyynnön lähettäjän identiteetin ja lähettää sen yhdyskäytävän takana oleville palvelutason mikropalveluille allekirjoitetussa Passport rakenteessa

5. Palvelutason mikropalvelut tarkastavat käyttäjän identiteetin saapuneen pyynnön mukana tulleesta Passport rakenteesta ja toteuttavat sen pohjalta pääsynhallinta päätöksiä.

6. Mikropalvelut voivat lähettää Passport-rakenteen seuraavalle palvelulle tarvittaessa.



Kuva 10. Using data structures signed by a trusted issuer (Barabanov & Makrushin 2020)

Oli käytettävä malli mikä tahansa on suositeltavaa käyttää vain yhtä tietorakennetta ulkoisen tunnisteen tunnistamiseen ja levittämiseen mikropalveluiden välillä. Reunatason palvelun on tarkastettava ulkoinen käyttöoikeustunnus ja luotava sisäinen esitysrakenne ja tunnus, jotka lähetetään palvelutason mikropalveluihin käytettäväksi. Sisäisen entiteetin käyttö luotettavalta liikkeellelaskijalta on suositeltavaa ja sisäisen entiteetin rakenteen tulisi olla helposti laajennettavissa, jolloin uusia käyttöoikeuksia voidaan lisätä helposti. Sisäistä rakennetta ei myöskään tulisi paljastaa esimerkiksi tallentamalla niitä järjestelmän ulkopuolelle kuten selaimiin tai laitteisiin. (Barabanov & Makrushin 2020)

3.3 Yhteenveto

Barabanovin ja Makrushinin (2020) mukaan hajautetun mallin käyttö ei ole suositeltavaa, koska lähdekoodin kovakoodatut valtuutussäännöt ja käytännöt niiden toteuttamiseen vaikeuttavat ylläpidettävyyttä ja skaalautuvuutta. Tämän sijaan suositellaankin ulkoistamaan valtuutussäännöt ja päätökset lähdekoodista ja käyttämään valtuutussääntöjen kirjoittamiseen siihen tarkoitettua kuvauskieltä. Suositeltavaa on käyttää sekä reunatason ja palvelutason valtuutusta. Barabanov ja Makrushin (2020) suosittelevat käyttämään keskitettyä mallia sulautetulla päätöksentekopisteellä sen tarjoaman joustavuuden ansiosta.

Suositeltavaa on myös käyttää valmiita ja tunnettuja ratkaisua oman itsekehitetyn järjestelmän sijaan. Tällöin ei tarvitse sitoa resursseja ratkaisun kehittämiseen ja ylläpitämiseen. Kaikkia pääsynhallintaan liittyviä sääntöjä ja käytäntöjä ei kuitenkaan todennäköisesti pystyä toteuttamaan olemassaoloilla ratkaisuilla, jolloin osa pääsynhallinnasta on toteutettava mikropalveluiden kooditasolla. On myös suositeltavaa noudattaa defense in depth-periaatetta vahvistamaan pääsynhallintaa järjestelmässä useammalla tasolla. Reunatasolla yhdyskäytävässä valtuutus hoidetaan karkealla tasolla ja mikropalvelutasolla käytetään jaettua valtuutuskirjastoa tai komponentteja tarkempien pääsynhallintapäätösten tekemiseen. Lisäksi mikropalvelun bisneslogiikka-tasolla voidaan toteuttaa niitä tukevia pääsynhallinnan sääntöjä. (OWASP n.d)

4 PÄÄSYNHALLINTAMALLIT

Tietoturvan ja tehokkaan pääsynhallinnan toiminnan kannalta on olennaista hallita, kuka saa käyttää mitäkin resursseja ja järjestelmiä. Näiden tietojen hallintaan on kehitetty erilaisia pääsynhallintamalleja. Ilman selkeitä pääsynhallintamalleja organisaatiot voivat altistua tietoturvaloukkauksille. Seuraavaksi tutkittiinkin pääsynhallintamalleja.

Erilaisia pääsynhallintamalleja tarvitaan, koska erilaiset organisaatiot, järjestelmät ja ympäristöt asettavat erilaisia vaatimuksia pääsynhallinnalle. Joissain organisaatioissa pääsynhallintaa voidaan suorittaa työntekijälle määritellyn roolin pohjalta hyödyntäen roolipohjaista valvontaa (RBAC). Pääsyoikeudet voidaan myöntää myös esimerkiksi kirjautumisjärjestelmän, sijainnin, ajan tai käyttäjän ja resurssin määriteltyjen ominaisuuksien perusteella. Tällöin käytetään ominaisuuspohjaista pääsynhallintaa (ABAC).

Valitun pääsynhallintamallin lisäksi tulisi noudattaa nollaluottamusmallia (Zero Trust), jossa jokainen pyyntö on tarkistettava ja todennettava. Lisäksi vähimmän käyttöoikeuden periaatteen (Principle of Least Privilege) mukaan käyttäjälle myönnetään vain ne oikeudet, jotka ovat välttämättömiä hänen tehtäviensä suorittamiseen. (StrongDM 2024)

Järjestelmien vaatimukset ja modernit pilvipohjaiset palvelut kehittyvät jatkuvasti tuoden haasteita pääsynhallintamallin valintaan. Tässä osiossa tarkastellaan, millaisia pääsynhallintamalleja on olemassa ja milloin niitä kannattaa käyttää.

4.1 Access Control List (ACL)

Käyttöoikeuslista tai pääsylista (Access Control List eli ACL) on malli, joka määrittelee, millä käyttäjällä on oikeus tiettyyn resurssiin tai mitä resurssiin liittyviä toimintoja hän saa suorittaa. Käyttöoikeuslista koostuu säännöistä, jotka määrittävät käyttäjän oikeudet tiedostoihin, hakemistoihin tai palveluihin. Sääntöjen avulla voidaan määrittää ja hallita, mitä toimintoja käyttäjä voi suorittaa resurssille. (Medium 2024) Käyttöoikeuslista määrittelee yksittäisen käyttäjän tai ryhmän oikeudet tiedostoihin tai hakemistoihin, mikä mahdollistaa tarkemman käyttöoikeuksien hallinnan verrattuna perinteisiin käyttöoikeusmalleihin (Lutkevich 2022).

Pääsylistat voidaan jakaa sallinta- ja kieltolistoihin. Sallintalista antaa pääsyn vain listalta löytyville käyttäjille ja toiminnoille. Kieltolista salli pääsyn kaikille toiminnoille paitsi listalta löytyville käyttäjille ja toiminnoille. Käyttöoikeuslista mahdollistaa suoraviivaisen tavan tarkkojen käyttöoikeuksien määrittelyyn. Haittapuolina on niiden monimutkaisuus ja rajoitettu kattavuus. Pääsyoikeuslistat voivat kasvaa hankaliksi hallita suurissa järjestelmissä, joissa on paljon käyttäjiä ja resursseja. Myös oikea määrittely vaati hyvää ymmärrystä, jotta vältytään odottamattomilta käyttörajoituksilta ja tietoturva-aukoilta. Käyttöoikeuslistat toimivat yleensä vain tietyssä järjestelmässä tai domainissa, eivätkä välttämättä sovellu laajojen useasta järjestelmästä koostuvien ympäristöjen hallintaan. (Saroyan n.d.)

4.2 Discretionary Access Control (DAC)

DAC on hajautettu pääsynhallintamalli, jossa käyttäjät voivat itse hallita resurssiensa käyttöoikeuksia. DAC:ia käytetään esimerkiksi älypuhelinsovelluksissa, Google Docsissa ja käyttöjärjestelmissä.

DAC-järjestelmissä käyttäjät voivat jakaa tietoa, myöntää käyttöoikeuksia, muokata objektien ominaisuuksia ja määrittää pääsynhallintaa ilman keskitettyä valvontaa. Toisin kuin MAC-mallissa, jossa pääsyoikeudet määritellään keskitetysti esimerkiksi luokitustasojen avulla, DAC antaa objektien omistajille vapauden hallita käyttöoikeuksia omistamilleen resursseille. (Nordlayer n.d.a)

DAC perustuu kohteisiin (subjects), joita ovat käyttäjät tai ryhmät, ja objekteihin (objects), joita ovat resurssit kuten sovellukset tai muut tallennetut tiedot. Käyttäjä tunnistautuu järjestelmään, jonka jälkeen tarkastetaan käyttäjän oikeudet objektiin. DAC-Järjestelmiä voidaan toteuttaa käyttöoikeuslistoilla (ACL) tai eräänlaisina kykyjärjestelminä, jossa pääsy perustuu objektin tunnistamiseen eikä käyttöoikeuslistoihin. Esimerkiksi kryptovaluutan omistaja saa käyttöoikeuden yksityisavaimen avulla. Imgur-käyttäjä voi puolestaan muokata kuvaa piilotetun URL-osoitteen kautta. Molemmat mallit ovat joustavia, ja objektin omistaja voi muokata käyttöoikeuksiaan. Esimerkiksi tietokantaryhmällä voi olla kirjoitusoikeudet, kun taas toisilla on vain lukuoikeudet. Imgur sallii tiedostojen jakamisen julkisten linkkien kautta, ja kryptovaluuttatransaktiot ovat mahdollisia julkisten avainten avulla omistajan suostumuksella. (Nordlayer n.d.a)

DAC-Järjestelmiä on vaikea valvoa ja seurata. Objektin omistajat vastaavat käyttöoikeuslistojen päivityksestä ja käyttöoikeuslistojen määrän kasvaessa vaikeutuu myös niiden hallinta. Liian suuret listat saattavat esimerkiksi sisältää vanhentuneita oikeuksia. Turvallisuuskäytännöt suosittelevatkin keskitettyjä hallintamalleja erityisen arkaluontoisten tietojen suojaamiseen. DAC ei pysty tarjoamaan tarvittavaa varmuutta, jotta organisaatiot voivat suojata esimerkiksi terveys- tai taloustietoja. (Nordlayer n.d.a)

4.3 Mandatory Access Control (MAC)

MAC on keskitetty järjestelmä, jossa pääsy resursseihin määräytyy käyttäjien lupatason ja objektien ominaisuuksien mukaan. MAC-mallissa käyttäjillä voi olla eri käyttöoikeustasot aseman ja roolin mukaan. Vain järjestelmänvalvojat voivat määrittää tietojen tai sovellusten pääsynhallintatasot ja säännöt. MAC-mallin haasteina ovat sen monimutkaisuus ja ongelmat toteuttamisessa. Käyttäjämäärän kasvaessa on varmistettava, että käyttöoikeudet ovat ajan tasalla ja vastaavat käyttäjien rooleja. Käyttäjien tunnistetiedot on esitettävä aina pääsyä hakiessa, mikä voi lisätä viivettä järjestelmässä. Käyttäjät eivät voi itse muuttaa omia pääsyoikeuksiaan. (Nordlayer n.d.b.)

4.4 RBAC ja ABAC

Mikropalveluissa käytetään yleisesti kahta pääasiallista pääsynhallintamallia, RBAC (Role-Based Access Control) ja ABAC (Attribute-Based Access Control). RBAC-mallissa käyttöoikeudet määräytyvät käyttäjän roolin perusteella. Esimerkiksi admin-rooli voi saada täyden pääsyn, kun taas user-roolilla on rajoitetut oikeudet. ABAC-malli on RBAC:ta tarkempi malli. ABAC-mallissa oikeudet perustuvat käyttäjän ja resurssin ominaisuuksiin, kuten rooliin, osastoon, sijaintiin, kellonaikaan tai datan arkaluonteisuuteen.

Valinta RBAC-mallin ja ABAC-mallin välillä riippuu käytettävän ympäristön monimutkaisuudesta ja hallinnan tarkkuudesta. RBAC on helpompi toteuttaa pienemmissä järjestelmissä, kun taas ABAC tarjoaa enemmän joustavuutta ja tarkkuutta vaativiin tilanteisiin. (Lombarte 2024)

RBAC-mallin hyvinä puolina on sen sääntöjen selkeys ja helppo toteutus. Tämä nopeuttaa valtuutuksen toimintaa ja vie vähemmän palvelimien laskentatehoa. Huonona puolena on roolien määrän

kasvaminen. Jos järjestelmään halutaan luoda tarkempia sääntöjä, on myös luotava näille säännöille vastaavat roolit. Tämä voi johtaa lukuisten roolien luomiseen, joiden ylläpitäminen on hankalaa ja aikaa vievää. (Okta 2024)

ABACin hyvinä puolina on mahdollisuus määrittää ja hallittaa useita muuttujia, joiden perusteella pääsynhallintaa voidaan suorittaa. Tämä mahdollistaa yksityiskohtaiset ja tarkat käyttöoikeussäännöt. Huonoina puolina näiden muuttujien määrittely ja sääntöjen konfigurointi vaatii paljon työtä ja tarkkaa suunnittelua. (Okta 2024)

4.5 Relationship-Based Access Control (ReBAC)

ReBAC-malli määrittää käyttöoikeudet resurssien välisiin suhteisiin perustuen. Näitä suhteita voivat olla: (OSO n.d.)

Datan omistajuus: Esimerkiksi julkaisun luoja voi muokata omaa sisältöään.

Ylä- ja alisuhteet: Kuten organisaatio ja siihen kuuluvat käyttäjätilit.

Ryhmät: Tiimit, joiden jäsenillä on yhteiset käyttöoikeudet.

Hierarkiat: Esimerkiksi esihenkilö-alainen -rakenteet.

ReBAC-mallissa esimerkkinä käyttöoikeuteen liittyvästä suhteesta on se, että käyttäjät voivat poistaa itse kirjoittamansa kommentit. Valtuutus määritellään objektien välisten suhteiden kautta. Esimerkkejä näistä ovat "kirjoittanut" -suhde käyttäjän ja kommentin välillä tai ryhmästruktuuri, jossa käyttäjät kuuluvat tiimeihin. ReBAC-mallissa näitä suhteita hyödynnetään sovelluksen valtuutuksen määrittelyssä. (OSO n.d.)

Toisin kuin RBAC-mallin yksinkertainen kyllä tai ei-päätöksenteko, ReBAC-malli analysoi entiteettien välisiä suhteita sovelluksen kontekstissa. Entiteettejä voivat olla esimerkiksi käyttäjät, resurssit kuten asiakirjat tai portaalit, käyttäjäryhmät sekä niiden väliset hierarkiat. Kun käyttäjä yrittää suorittaa toiminnon resurssille, ReBAC-malli voi tarkistaa esimerkiksi: Omistaako käyttäjä resurssin eli loiko hän sen? Mikä on käyttäjän, hänen ryhmänsä ja resurssin välinen suhde? Onko resurssi osa hierarkiaa, johon käyttäjällä on tai ei ole pääsyä? Jos järjestelmä vahvistaa, että vaaditut suhteet täyttyvät, toiminto sallitaan. Muussa tapauksessa pääsy estetään. (Iyer 2023; OSO n.d)

ReBAC ei perustu pelkästään virallisiin rooleihin, vaan mukautuu dynaamisesti alustan muuttuvien suhteiden mukaan. Se tarjoaa joustavan ja tarkasti määriteltävän pääsynhallintamallin, joka soveltuu kehittyviin ja monimutkaisiin sovelluksiin. Muuttujien ja suhteiden määrittely vaatii tarkkaa suunnittelua ja asiantuntemusta. Lisäksi suhteiden analysointi pääsynhallintapäätöksiä tehdessä kuluttaa laskentaresursseja. (Iyer 2023)

5 TEKNISET RATKAISUT JA TYÖKALUT

Tässä vaiheessa käytiin toimeksiantajan kanssa keskustelua mahdollisesta arkkitehtuurimallista ja sen toteuttamiseen tarvittavista tekniikoista. Toimeksiantajan puolelta nähtiin tarvetta API-yhdyskäytävän käyttöönottamisesta. Yhdyskäytävän toteuttamista HAProxylla haluttiin tutkia sen monipuolisten ominaisuuksien ja aikaisemman kokemuksen vuoksi. Toimeksiantajan kehittämät palvelut pyörivät Kubernetes-ympäristössä ja täten haluttiin tutkia palveluverkon toteuttamista Kuberneteksen kanssa yhteensopivalla Istiolla. Tässä vaiheessa kerrottaan näistä valituista tekniikoista sekä jo toimeksiantajalla käytössä olevista tekniikoista kuten Laravel-ohjelmistokehyksestä, jolla on toteutettu olemassa olevat palvelut sekä järjestelmän käyttöoikeustietueena käytettävästä JSON Web Tokenista.

5.1 HAProxy

HAProxy (High availability proxy) on ilmainen avoimen lähdekoodin sovellus kuormantasaukseen ja välityspalvelin käyttöön. Kuormantasaus on prosessi, jossa palveluun saapuvaa liikennettä jaetaan useammalle palvelimelle, jotta palvelun suorituskyky ja luotettavuus paranevat (Cloudflare n.d.a). Internetissä olevat palvelut näkyvät käyttäjille selkokielisillä domain-nimillä. Palvelut pyörivät kuitenkin palvelimilla, joilla on IP-osoite. Palvelu voidaan jakaa pyörimään useammalle palvelimelle, esimerkiksi palvelimille A, B ja C, jolloin palvelimen käyttämä laskentateho jakautuu useammalle palvelulle. (WizardCowOy n.d.). Internetissä kuormantasausta käytetään jakamaan verkkoliikenne useamman palvelun kesken. Tämä vähentää yksittäisen palvelimen rasitusta ja tekee palvelimista tehokkaampia nopeuttaen niiden suorituskykyä ja viivettä. Kuormantasaus on lähes välttämätöntä useimmille internetsovelluksille. (Cloudflare n.d.a) Esimerkkinä voidaan käyttää kaupan kassalinjastoa. Yhden kassalinjan ollessa auki kaikkien asiakkaiden täytyy jonottaa päästäkseen maksamaan ostoksensa. Useamman kassalinjan ollessa auki lyhenee myös asiakkaiden jonottamiseen käyttämä aika. Kuormantasaus tekee saman asian jakamalla käyttäjien pyynnöt useamman palvelimen kesken, joka lyhentää odottamisaikaa ja parantaa täten käyttäjäkokemusta. Kuormantasaukseen käytetään työkalua tai sovellusta kuten HAProxy. HAProxy voi toimia palvelimella, virtuaalikoneella tai pilvipalvelussa.

Käänteinen välityspalvelin (Reverse proxy) on palvelin, joka sijaitsee verkkopalvelimien edessä ja välittää asiakasohjelmien pyynnöt oikeille palvelimille. Käänteisen välityspalvelimen ymmärtämiseksi on ymmärrettävä välityspalvelimen (Proxy) toimintaa. Välityspalvelin sijaitsee asiakaskoneiden edessä. Asiakaskoneen tekemä pyyntö web-palvelulle menee välityspalvelimen kautta, joka keskustelee pyynnön kohteena olevan web-palvelimen kanssa asiakasohjelmien puolesta toimien eräänlaisena välittäjänä. Tavallisessa viestinvälittämisessä asiakasohjelma tavoittaisi suoraan kohteena olevan web-palvelimen, jolloin asiakasohjelma lähettää pyynnöt web-palvelimelle ja web-palvelin vastaa asiakasohjelmalle. Välityspalvelinta käyttäessä asiakasohjelma lähettää pyynnöt välityspalvelimelle, joka välittää pyynnön web-palvelimelle ja vuorostaan web-palvelin palauttaa vastauksen välityspalvelimen kautta asiakasohjelmalle. Välityspalvelinta voidaan käyttää useista eri syistä organisaatioissa rajoittamaan pääsyä internettiin, esimerkiksi kouluissa rajoittamaan pääsyä sosiaalisen median palveluihin tai piilottamaan käyttäjän IP-osoite internetissä olevissa palveluissa. (Cloudflare n.d.b)

Käänteinen välityspalvelin (reverse proxy) on palvelin, joka eroaa välityspalvelimesta siten, että käänteinen välityspalvelin ei sijaitse pyyntöjä lähettävien asiakaskoneiden edessä, vaan se sijaitsee pyyntöjen kohteena olevien web-palvelimien edessä. Käänteistä välityspalvelinta käyttäessä asiakasohjelma tai kone lähettää pyynnön web-palvelimelle, mutta tämän pyynnön nappaa kohdejärjestelmän reunalla oleva käänteinen välityspalvelin, joka ohjaa pyynnön oikealle web-palvelimelle. Yhteenvetona välityspalvelin ja käänteinen välityspalvelin eroavat siten, että välityspalvelinpalvelin varmistaa, ettei pyynnön kohteena oleva web-palvelin keskustele asiakasohjelman tai koneen kanssa suoraan. Käänteinen välityspalvelin taas sijaitsee pyyntöjen kohteena olevan palvelimen edessä ja varmistaa ettei yksikään asiakasohjelma keskustele suoraan web-palvelimen kanssa. (Cloudflare n.d.b)

Käänteinen välityspalvelin lisää siis ylimääräisen suojakerroksen vastaanottamalla asiakasohjelmien pyyntöjä ja välittämällä ne taustapalvelimille. Tämä auttaa piilottamaan todellisen palvelimen IP-osoitteen ulkopuolisilta käyttäjiltä, mikä vaikeuttaa palvelimeen kohdistuvia hyökkäyksiä. Käänteinen välityspalvelin käsittelee HTTP-pyynnöt ennen niiden välittämistä eteenpäin ja hoitaa myös suodatusta ja kuormantasausta. Käänteinen välityspalvelin tarkastaa ja suodattaa saapuvat pyynnöt, jolloin voidaan estää kaikki saapuneet haitalliset pyynnöt. Käänteinen välityspalvelin voidaan määrittää jakamaan saapuva liikenne useammalle taustapalvelimelle mikä toteuttaa kuormantasauksen periaatteita. (VPN Unlimited n.d.a.)

Mikropalveluissa käänteistä välityspalvelinta voidaan käyttää API-yhdyskäytävän toteuttamiseen, jolloin välityspalvelin ohjaa yhdyskäytävään tulevat pyynnöt oikeille mikropalveluille. Välityspalvelin myös piilottaa yhdyskäytävän takana olevien mikropalveluiden IP-osoitteet julkiverkolta parantaen järjestelmän tietoturvallisuutta. API-yhdyskäytävä voidaan toteuttaa käyttämällä HAProxya.

HAProxy sopii hyvin käytettäväksi API-yhdyskäytävänä sen sisältämien ominaisuuksien vuoksi. HAProxy reitittää API-kutsuja oikeille palveluille ja toteuttaa kuormantasausta. HAProxylla voidaan myös rajoittaa rajapyyntöjä, hyödyntää välimuistia, toteuttaa järjestelmän monitorointia ja lokitusta sisäänrakennettujen palvelujen ansiosta. (HAProxy Technologies 2021)

Yhdyskäytävän tehtävä on yhdistää käyttöliittymäsovelluksesta tulevat pyynnöt API-päätepisteisiin, estäen kuitenkin suoran yhteyden käyttöliittymäsovelluksen ja API-Päätepisteen väliltä toimimalla käänteisenä välityspalvelimena palveluiden välillä. Täten käyttöliittymäsovellukselle on näkyvissä vain yhdyskäytävän osoite.

Yhdyskäytävän ensisijainen tehtävä on reitittää saapuvat pyynnöt oikealle palvelulle. HAProxy reitittää minkä tahansa HTTP-pyynnön mukana tulleen tiedon perusteella. Näitä tietoja ovat esimerkiksi URL-polku, hakuparametri ja HTTP-otsikkotiedot. Esimerkiksi pyynnössä oleva URL-polku /cart tai /catalog voidaan ohjata oikeille taustapalveluille. Jos yhdessä reitissä tarkastettavien polkujen määrä kasvaa isoksi, voidaan tarkastukseen käyttää routing.map-tiedostoa, joka tallentaa reitit avain/arvo pareina:

#HTTP Pyynnön polku:	# Ohjataan taustapalveluun:
/cart	be_cart
/catalog	be_catalog

Kuormantasaus parantaa palveluiden suorituskykyä jakamalla kuorman useammalle palvelimelle. API-päätepisteet voidaan monistaa useammalle nodelle joille yhdyskäytävä tasapainottaa saapuvat pyynnöt. HAProxy voi myös reitittää liikenteen automaattisesti erilaisten health check-toimintojen muodossa. (HAProxy Technologies 2021)

Ylikuormituksensuojaukseen (Server overload protection) HAproxy käyttää jonotusmekanismeja estääkseen liian monen samanaikaisen pyynnön lähettämisen palveluun. Maxconn-argumentila voidaan säätää palvelinkohtaisesti yhteyksien määrä, jonka jälkeen yhdyskäytävä asettaa saapuvat pyynnöt jonoon. (HAProxy Documentation 2025)

Rate limiting rajoittaa asiakasohjelman tekemien pyyntöjen määrää tietyn ajan kuluessa. Se toimii tarkkailemalla ja hallitsemalla verkkosivustolle, palvelimelle tai sovellukselle tehtyjen pyyntöjen tiheyttä. Näin rajoittaminen toimii tarkasti. (VPN Unlimited n.d.b.) HAProxyssa on mahdollista rajoittaa saapuvia pyyntöjä pitämällä kirjaa ja yksilöimällä saapuvat pyynnöt IP-osoitteen, URL parametrien, evästeiden tai muiden HTTP-otsikkotietojen perusteella. (HAProxy Documentation 2025)

HAProxy sisältää tilastojen hallintapaneelin ja lokitietojen seurannan. Hallintapaneeli tarjoaa tiedot jokaisesta käyttöliittymästä, taustapalvelimesta ja palvelusta. HAProxyn kuormantasaaja sisältää sisäänrakennetun HAProxyn Runtime API:n, joka palauttaa JSON tai CSV-muotoista dataa, mitä voidaan käyttää kolmannen osapuolen seuranta ja valvontatyökaluissa. Lokituksiin on mahdollista tallentaa asiakkaan IP-osoite ja käyttämä portti, reititystietoja, pyynnön URL hakuparametreineen, aika- ja tilatietoja sekä mitä tahansa mukautettuja otsikko- tai evästetietoja, mitä halutaan tallentaa. Näiden avulla on tarvittaessa helppo suorittaa vianetsintää järjestelmään kohdistuvissa vikatilanteissa. HAProxyssa on sisäänrakennetut metodit JWT käsittelyyn, näistä tärkeimpinä jwt_verify-konvertteri, joka mahdollistaa saapuneen tokenin allekirjoituksen tarkastamisen. (HAProxy Documentation 2025)

5.2 Kubernetes

Kubernetes on avoimen lähdekoodin ohjelmisto ohjelmistokonttien hallintaan, ja siitä on tullut laajalti hyväksytty alusta hajautettujen järjestelmien käytössä (Redhat 2024). Mikropalvelut ovat usein käytössä konteissa, jotka tarjoavat eristyksen ja johdonmukaisuuden, joita mikropalvelut tarvitsevat toimiakseen itsenäisesti ja kommunikoidakseen keskenään (Loft Team 2023). Kubernetes itsessään ei ole tämän opinnäytetyön aiheena, mutta Kubernetestä käytetään toimeksiantajan järjestelmässä, ja se pitää ottaa huomioon pääsynhallintaa toteuttaessa. Kubernetes koostuu useammasta osasta, joita ovat:

Container eli kontti. Se sisältää sovelluksen ja kaiken sovelluksen ajoon tarvittavan koodin, riippuvuudet ja ympäristöasetuksien konfiguroinnin. Kontti on nopeampi ja kevyempi kuin virtuaalikone.

Cluster on kokonaisuus, joka koostuu useista palvelimista eli nodeista.

Node on yksittäinen palvelin, joka suorittaa kontteja.

Pod on pienin Kubernetesin hallinnoima yksikkö, joka sisältää yhden tai useamman kontin.

Deployment määrittää miten ja missä podit ajetaan.

Service mahdollistaa podien välisen ja ulkoisen liikenteen hallinnan.

Ingress halllitsee ulkopuolista verkkoliikennettä.

Yksinkertaistettuna mikropalveluiden näkökulmasta Kubernetes-ympäristö näyttäytyy seuraavanlaisena:

Jokainen kontti toimii omassa ympäristössään ja Kuberneteksen avulla niitä on helppo skaalata lisäämällä tai poistamalla yksittäisiä kontteja. Yksittäinen kontti käynnistyy nopeammin kuin perinteinen virtuaalikone. Kontti sisältää yksittäisen mikropalvelun esimerkiksi tuotepalvelun ja sen tarvitsemat riippuvuudet. Kontti voidaan ajaa itsenäisesti missä tahansa. Pod sisältää yhden tai useamman kontin, joilla on jaettu tallennustila ja verkkoresurssit sekä määritykset, kuinka konttia ajetaan (Kubernetes Documentation 2025a). Node on fyysinen tai virtuaalinen palvelin joka suorittaa podeja. Node sisältää tarvittavat palvelut podien pyörittämiseen ja Kubernetes hallitsee, mille nodeille podit sijoitetaan (Kubernetes Documentation 2024). Cluster on kokonaisuus, jossa on useita nodeja, joita Kubernetes hallitsee ja se onkin mikropalveluista koostuva ekosysteemi, jossa pyörivät kaikki mikropalveluiden kontit (VMware n.d.a). Deployment määrittää montako podia tarvitaan ja se auttaa palveluita skaalautumaan ja pysymään käynnissä (WMware n.d.b.). Servicen avulla mikropalvelut kommunikoivat keskenään (Kubernetes Documentation 2025b). Ingressi hallitsee Kuberneteksen ulkopuolelta tulevaa liikennettä. Egress on Kubernetes clusterista ulospäin suuntautuvaa liikennettä, tarkemmin podista ulkopuoliseen päätepisteeseen suuntautuvaa liikennettä. Egressin avulla voidaan hallinoida liikennettä clusterin ulkopuolisiin palveluihin, kuten tietokantoihin, API-rajapintoihin tai muihin kolmansien osapuolien palveluihin. Kubernetes clusterissa podit eristetään oletusarvoisesti clusterin ulkopuolisesta verkosta, jolloin ne eivät voi käyttää ulkopuolisia palveluita. Kuberneteksen Eggress Network Policy mahdollistaa määritellä säännöt, joiden perusteella määritellään, mihin klusterin ulkopuolisiin palveluihin podeilla on lupa muodostaa yhteys. (Solo.io n.d.)

5.3 Istio

Istio on Service Mesh eli palveluverkkoratkaisu, jota voidaan käyttää Kubernetes-ympäristössä pyörivien hajautettujen järjestelmien ja sovellusten välisen liikenteen hallintaan. (Google Cloud n.d.) Kubernetes hallitsee kontteja ja Istio niiden välistä liikennettä (IBM n.d.). Palveluverkko on infrastruktuurikerros, joka helpottaa palveluiden välistä viestintää (Di Pietro 2022).

Mikropalveluiden tehtävänä on toteuttaa bisneslogiikkaan liittyvää toimintaa. Tämän lisäksi mikropalveluissa toteutetaan bisneslogiikan ulkopuolisia toimintoja kuten, yhteyksien konfigurointia, turvallisuuteen liittyvää logiikkaa sekä lokitustietoja ja järjestelmän monitorointia. Näitä bisneslogiikan ulkopuolisia logiikoita kutsutaan sovelluslogiikaksi. Bisneslogiikan ulkopuoliset sovelluslogiikat pitää lisätä jokaiseen palveluun. (Shah 2025)

Sovelluslogiikan toteutus tuo omia haasteita kehittämiseen ja pääsynhallintaan, sillä kehittäjät eivät välttämättä työskentele sekä palvelu- että sovelluslogiikan parissa. Järjestelmään lisätyn uuden mikropalvelun päätepisteiden tai uuden yksittäisen päätepisteen lisääminen jo olemassa olevaan mikropalveluun vaatii päätepisteen sekä palvelun osoitteen lisäämiseen kaikkiin niihin mikropalveluihin, jotka käyttävät niitä. Mikropalveluiden podit ovat yhden Kubernetes clusterin sisällä, jonka edessä on yleensä palomuuri ottamassa vastaan ulkoverkosta tulevia pyyntöjä. Kuitenkin mikropalveluiden väliset yhteydet clusterin sisällä voivat olla suojaamattomia, jolloin jokainen mikropalvelu voi keskustella vapaasti minkä tahansa palvelun kanssa.

Sovelluslogiikan toteutuksen ongelmat voidaan ratkaista käyttämällä palveluverkkoa ja sivuvaunua. Mikropalvelun kylkeen lisätään erillinen sivuvaunupalvelu tai -moduuli. Kubernetes-ympäristössä mikropalvelu ja sivuvaunu sijaitsevat samassa podissa. Sivuvaunu toimii välityspalvelimena varsinaisen mikropalvelun edessä. Se hoitaa verkkologiikkaa ja muita bisneslogiikkaan kuulumattomia toimintoja. Kuberneteksen klusterioperaattorit voivat määritellä sen asetuksia, jolloin kehittäjät voivat keskittyä bisneslogiikan kehittämiseen. Mikropalvelut voivat keskustella sivuvaunujen välityksellä toisilleen. (Istio 2025)

Sivuvaunun tarkoituksena on toimia välityspalvelimena, joka hoitaa viestintäkerrokseen liittyvät tehtävät mikropalvelun puolesta, johon sivuvaunu on liitetty (Di Pietro 2022). Istio-palveluverkko on jaettu datatasoon (Data-plane) ja ohjaustasoon (Control Plane). Datataso koostuu välityspalvelimina toimivasta sivuvaunuista, joita kutsutaan Envoyksi. Envoyt ohjaavat kaikkea verkkoviestintää mikropalveluiden välillä ja ohjaustaso hallitsee sekä määrittelee Envoyt liikenteen reitittämiseksi oikein. (Istio 2025) Ohjaustaso lisää ylimääräisen kontin podeihin, jotka toimivat välityspalvelimena ja tätä mekanismia kutsutaan sidecar proxyksi. Tämän jälkeen palvelut keskustelevat näiden sivuvaunujen kautta eivätkä suoraan toistensa kanssa. (Di Pietro 2022)

Istiosta löytyy ominaisuudet tunnistautumiseen, käyttöoikeuskäytäntöihin ja salaukseen sekä autentikointiin, valtuutukseen ja auditointiin. Istio mahdollistaa suojauksen monella eri tavalla. Podit käyttävät salattua liikennettä estämään ulkopuolisen seurannan. Tunnistuspalvelu ja salaus varmistavat, ettei luvaton käyttäjä voi suorittaa pyyntöä palveluihin (IBM n.d.). Istio tukee myös JWT-tunnisteen käyttöä klusterin ulkopuolelta tulevasta pyynnöstä tai klusterin sisäisestä työkuormasta. Istio mahdollistaa myös pääsynhallinta sääntöjen määrittelyä AuthorizationPolicyn avulla (Istio 2025). Istio tukee myös ulkoisen pääsynhallinta palvelun käyttöä External Authorization avulla, joka mahdollistaa integroimisen esimerkiksi oman itsekehitetyn pääsynhallintapalvelimen kanssa (Istio 2025). Istiossa valtuutusta voidaan käyttää pääsynhallintasääntöjen pakottamiseen työkuormien (Workload) välillä. Valtuutussäännöt voidaan määrittää Kubernetes CRD (Customer Resource Definition) avulla. (Sereg 2020)

5.4 Laravel

Laravel on avoimen lähdekoodin MVC-mallin mukainen web-ohjelmistokehys. Laravel pyrkii parantamaan ohjelmistokehittämistä helpottamalla yleisiä tehtäviä kuten autentikointi, reitittämistä, istuntojen hallintaa ja välimuistin käyttöä. Laravel sisältää Eloquent ORMin, joka helpottaa tietokantatietueiden käsittelyä esittämällä tietoa objekteina. Objektit toimivat samalla abstraktointikerroksena tietokantamoottorin päällä, jota käytetään sovelluksen tietojen tallentamiseen. (Heidi 2025)

Näiden ominaisuuksien ansiosta Laravel suoraviivaistaa web-kehitykseen liittyviä tehtäviä, joten se sopii mikropalveluarkkitehtuuria käyttävän järjestelmän kehittämiseen. Eloquent ORM on vuorovaikutuksessa tietokantojen kanssa ja tekee monimutkaisten SQL-kyselyjen suorittamisesta helppoa. Ominaisuudet, kuten jonot, events ja broadcasting, mahdollistavat asynkronisen viestinnän irrotettujen palvelujen välillä. Laravel hoitaakin suuren osan mikropalvelu kehitykseen liittyvästä työstä, ja kehittäjät voivat keskittyä vapaammin toteuttamaan businesstason ominaisuuksia. (Gosai 2023)

Laravel tarjoaa myös työkaluja pääsynhallinnan toteuttamiseen, näistä keskeisimpiä ovat Laravel Passport, Gates, Policies ja Middlewaret. Passport on Laravel-sovelluskehyksen tarjoama työkalu OAuth 2.0- standardin mukaisen palvelinimplementaation toteuttamiseen. Passport yksinkertaistaa

käyttäjien tunnistautumista ja valtuutusta API-pohjaisissa sovelluksissa tarjoten tuen token-pohjaiselle tunnistautumiselle. Passport sisältää täyden OAuth 2.0 tuen, jolloin sovelluksessa on mahdollista käyttää kaikkia standardin sisältämiä kulkuja (flows tai grant types) tokenin myöntämiseksi käyttäjälle. (Laravel n.d)

Laravelissa on kaksi ensisijaista tapaa toimintojen valtuuttamiseen, portit (gates) ja käytännöt (policies). Portit ja käytännöt voidaan käsittää samalla tavoin kuin web-sovelluksen reitti (routing) ja kontolleri (controller) luokat. Portit päättävät, voiko käyttäjä suorittaa tietyn toiminnon. Portin käyttäminen vaatii käyttäjäinstanssin käyttöä ja portti voi saada lisäksi valinnaisen lisäargumentin, kuten suoritettavaan toimintoon liittyvän Eloquent-mallin. Esimerkkinä portilla voidaan verrata, voiko käyttäjä muokata tehtyä postausta, vertaamalla käyttäjän tunnistetietoa postauksen tiedoista löytyvään tekijän tunnistetietoon. Tarkastuksen pohjalta voidaan tehdä kyllä tai ei päätös suoritettavalle toiminnolle. Käytännöt ovat luokkia, jotka järjestävät valtuutuslogiikan tietyn mallin ympärille. Käytännöllä voidaan määrittää käyttäjien oikeudet tietyn Eloquent-malliin liittyvien toimintojen suorittamiseen. (Laravel n.d.)

Yhteenvetona käytännöt ovat erillisiä luokkia, jotka järjestävät ja hallinnoivat tiettyjen mallien valtuutusta. Ne ovat erityisen hyödyllisiä laajemmissa sovelluksissa, joissa skaalautuvuus ja selkeä rakenne ovat tärkeitä. Käytännöt sisältävät menetelmiä, joilla tarkistetaan, voiko käyttäjä suorittaa toimintoja, kuten luoda, päivittää tai poistaa tietyn mallin tietueita. Näin käytännöt varmistavat sovelluksen yhtenäisen valtuutuslogiikan. Portit ovat yksinkertaisia, sulkupohjaisia funktioita, jotka käsittelevät valtuutuslogiikkaa. Ne ovat erityisen tehokkaita pienemmissä sovelluksissa tai yksinkertaisissa, tarkkarajaisissa tarkistuksissa. Esimerkiksi portti voi määrittää, onko käyttäjällä oikeus nähdä käyttöliittymässä oleva hallintapaneeli tai suorittaa tietty toiminto jossakin ominaisuudessa. (Upadhyay 2024)

Laravelin Middleware-mekanismia voidaan käyttää rajoittamaan pääsyä tietyille reiteille. Middlewaressa voidaan tarkastaa sovellukseen saapuvia HTTP-pyyntöjä. Middlewaressa voidaan tarkastaa, onko käyttäjä kirjautunut sisään ja ohjata sen perusteella pyyntöä tai käyttäjää esimerkiksi kirjautumissivulle. Sitä voidaan käyttää myös muihin tehtäviin kuten tokeneiden tarkastamiseen ja lokitukseen. Laravel sisältää valmiita Middlewareja esimerkiksi tunnistautumiseen ja CSFR-Suojaukseen. Laravelissa on myös mahdollistaa määritellä omia Middleware luokkia. (Laravel n.d.)

5.5 OAuth 2.0 ja JSON Web Token

Perinteisesti web-sovelluksissa tunnistautumista ja valtuutusta toteutettiin käyttämällä istuntopohjaisia valtuutusmekanismeja, joissa käyttäjätiedot tallennettiin palvelimelle. Sovellusten kehittyessä monimutkaisemmaksi hajautetuksi järjestelmiksi alkoivat perinteiset menetelmät kohdata rajoitteita. Istuntokohtaisten tunnisteiden käyttö vaati usein useita tietokantakyselyitä ja palvelinlogiikkaa, mikä heikensi suorituskykyä ja palvelujen skaalautuvuutta. Lisäksi tilallisten istuntokohtaisten tunnisteiden hallinta oli haasteellista käyttäjämäärän ja laitteiden kasvaessa. (Ibrahim 2024)

Istuntokohtaisessa tunnistautumisessa käyttäjä kirjautuu käyttäjätunnuksilla palvelimelle ja käyttäjän istunto tallennettaan palvelimelle. Palvelin palauttaa istuntokohtaisen tunnistetiedon evästeenä käyttäjälle. Tämä eväste palautetaan palvelimelle joka kerta, kun käyttäjä tekee palvelimelle pyynnön.

Istuntokohtaisen tunnistautumisen ongelmia on ratkaistu käyttämällä tokeneita. Token-pohjaisessa tunnistautumisessa palvelin palauttaa käyttäjätunnuksia vastaan tunnisteena toimivan tokenin, joka allekirjoitetaan salaisella avaimella, jota ei voi peukaloida. Token lisätään jokaisen pyynnön otsikkotietoihin, josta se voidaan tarkastaa salausavaimia käyttäen vastaanottavalla palvelimella. Käyttäjän tila tallennettaan tokeniin clientin päädyssä eikä palvelimelle kuten istuntopohjaisessa ratkaisussa. Json Web Token eli JWT on tunnisteformaatti, jolla voidaan toteuttaa OAuth 2.0 protokollaa.

OAuth on noussut laajalti hyväksytyksi todennusprotokollaksi hajautetuissa järjestelmissä. Protokollana se mahdollistaa turvallisen pääsyn resursseihin ilman käyttäjätunnusten, esimerkiksi käyttäjänimen ja salasanan, jakamista. OAuth toimii token-pohjaisena todennusmallina, jossa käyttäjälle myönnetään token onnistuneen todennuksen jälkeen ja käyttäjä pääsee tokenin avulla käyttämään suojattuja resursseja.

Ennen OAuth-protokollaa yleinen malli käyttäjätilin käyttöoikeuden myöntämiselle kolmannen osapuolen sovellukselle oli antaa salasana suoraan kolmannelle osapuolelle. Tällä käyttäjien salasanoja hankkivien sovellusten mallilla on useita ilmeisiä ongelmia koska sovelluksen olisi kirjauduttava palveluun käyttäjänä ja sovellukset tallentavat usein käyttäjien salasanat pelkkänä tekstinä ilman salausta, mikä teki salasanojen keräämisestä houkuttelevan kohteen tietoturvaloukkaajille. Käyttäjän salasanan omaavalla sovelluksella on täydellinen pääsy käyttäjän tiliin ja toimintoihin, kuten salasanan vaihtamiseen. Ainoa tapa peruuttaa sovelluksen pääsy käyttäjätiliin oli vaihtaa salasana. Tämän takia monet palvelut kehittivät omia mallejaan ratkaisemaan tämän mallin ongelmat, esimerkkeinä Flickrin käyttämä FlickrAuth, Googlen AuthSub ja Yahoon BBAuth. Tämä johti useisiin erilaisiin ratkaisuihin, jotka olivat yhteensopimattomia keskenään ja joissa ei huomioitu tiettyjä turvallisuusnäkökohtia. Tämän pohjalta päätettiin luoda avoin standardi API-käyttöoikeuksien delegoinnille, josta syntyi OAuth-protokolla. (Parecki n.d.)

OAuth-protokollan avulla käyttäjät voivat valtuuttaa sovelluksen toisen sovelluksen kanssa antamatta salasanaa. Käyttäjä voi itse määritellä tarkasti, mitä käyttöoikeuksia sovellukselle annettaan. OAuth antaa sovelluksen omistajille mahdollisuuden myöntää verkkotunnusten välisen käyttöoikeuksien hallinnan. Se hallitsee myös todennusta ja valtuutusta erikseen mikä mahdollistaa helpomman yhteentoimivuuden. Protokollaa voidaan käyttää myös palvelimien väliseen, sovellus-palvelin käyttöön ja sitä voidaan myös käyttää yhdessä muiden protokollien kanssa, jolloin on sitä on mahdollista käyttää myös monimutkaisemmissa käyttötapauksissa. (Solomon 2024)

OAuth tunnuksia voidaan käyttää mikropalvelujen turvallisessa viestinnässä. Niitä voidaan käyttää myös mikropalvelujen pyyntöjen todentamiseen ja valtuutukseen varmistamalla, että jokaisella palvelulla on tarvittavat käyttöoikeudet. (Solomon 2024)

OAuth2 tokeneina käytetään yleensä JSON Web Tokeneita (JWT). JWT on avoimen standardin URL-osoiteturvallinen esitystapa käyttösoikeustietueiden siirtämiseen JSON-objektina (Jones yms. 2015). Yleisin käyttötapaus JWT:lle on valtuutus. Kirjautuneen käyttäjän lähettämät pyynnöt sisältävät JWT:n jonka avulla käyttäjä voi käyttää palveluita, reittejä tai päästä käsiksi palvelimien sisältämiin resursseihin. JWT Mahdollistaa myös Single Sign On eli kertakirjautumisen. JWT:tä voidaan käyttää myös turvalliseen tietojen siirtoon osapuolten välillä. JWT allekirjoitetaan käyttämällä julkista

ja yksityistä avainparia, jolloin voidaan varmistua, että lähettäjät ovat, joita he sanovat olevansa. Allekirjoitus lasketaan pyynnön otsikon ja hyötykuorman avulla, joka varmistaa, ettei sisältöä ole peukaloitu. (JWT n.d.)

JWT koostuu kolmesta osasta, otsikosta (header), hyötykuormasta (payload) ja allekirjoituksesta (signature), jotka erotellaan pisteellä. Otsikko sisältää tyypillisesti kaksi osaa, tiedon tokenin tyypistä ja salausalgoritmista.

Esimerkki otsikosta:

{

"alg": "HS256",

"typ": "JWT"

}

Toinen osan on hyötykuorma, joka koostuu tokenin sisällöstä/väitteistä (claims). Väitteet sisältävät tietoja käyttäjästä, kuten esimerkiksi nimen ja ID-tunnuksen.Väitteet voidaan jakaa kahteen tyypiin. Rekisteröityjä väitteitä ovat IANAN:n rekisteröimät ja JWT spesifikaatioiden määrittelemät seitsemän standardivaatimusta, jotka takaavat yhteensopivuuden kolmansien osapuolien kanssa. Rekisteröidyt väitteet:

iss:(Issuer/myöntäjä) JWT:n myöntäjä

sub:(subject/subjekti) Käyttäjä

aud:(audience/vastaanottaja) JWT:n vastaanottaja

exp: (expirtation time / vanhentumisaika) Aika sekunteina, jonka jälkeen token vanhenee

nbf:(not before time) Aika, jota ennen JWT:tä ei saa hyväksyä käsiteltäväksi

iat (issued at time / myönnetty ajankohtana): Aika, jolloin JWT myönnettiin; voidaan käyttää JWT:n iän määrittämiseen

jti (JWT ID): Yksilöllinen tunniste; voidaan käyttää estämään JWT:n toistaminen. Se sallii tokenin käytön vain kerran.

Näiden ulkopuolelta voidaan käyttää myös rekisteröimättömiä väitteitä, kuten mukautettuja väitteitä. Niitä ovat rekisteröimättömät julkiset tai yksityiset väitteet, joita JWT:n luoja voi lisätä ja hallita. Esimerkkeinä käyttäjän tunnistetiedot kuten sähköpostiosoite tai muu tunniste. (Auth0 by Okta n.d.)

Kuva 11. Esimerkki hyötykuormasta. (JWT n.d.)

Allekirjoitus varmistaa JWT:n lähettäjän olevan sama, kuin se väittää olevansa ja samalla varmistaa, ettei JWT:n sisältöä ole muutettu matkan varrella. Allekirjoitus luodaan ottamalla otsikko ja hyötykuorma, jotka Base64-koodataan salaisuuden kanssa ja allekirjoitetaan otsikossa määritellyllä algoritmilla

Esimerkki allekirjoituksesta:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Otsikosta, hyötykuormasta ja allekirjoituksesta koostuva rakenne erotellaan pisteillä toisistaan ja Base64 koodataan kompaktimpaan muotoon, jota on helppo käsitellä HTML ja HTTP ympäristöissä.

Esimerkki Base64 koodatusta tokenista:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIx-
MjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJS-
MeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

JWT lähetetään käyttäjän tekemien HTTP-pyyntöjen otsikkotiedoissa, jonka vastaanottava palvelin voi tarkastaa ja käsitellä pyynnön tokenin mukana olevien mahdollisten väitteiden avulla. Jotkin palvelimet eivät pysty käsittelemään yli 8KB kokoisia otsikkotietoja, tämän takia JWT kokoa tulisi rajoittaa pyrkimällä pitämään sen hyötykuormassa olevien väitteiden koko maltillisena. Tämä voi olla rajoitteena sisällyttäessä käyttäjän lupia hyötykuormaan. (JWT n.d.)

6 TOTEUTUS

6.1 Ehdotus toimeksiantajan järjestelmään

6.2 Ratkaisun kuvaus

6.3 Haasteet ja parannusehdotukset

7 YHTEENVETO

Tämän opinnäytetyön tuloksena oli tutkia, miten toteuttaa keskitetty pääsynhallinta hajautettuun mikropalveluista koostuvaan järjestelmään. Valitun mallin tuli myös vastata toimeksiantajan vaatimuksiin ja kysymyksiin.

Työssä tutkittiin käytettyjä arkkitehtuuritason malleja pääsynhallintaan mikropalveluympäristöissä. Lisäksi tutkittiin API-Yhdyskäytävän käyttämistä ja tekniikoita pääsynhallinnan toteuttamiseen järjestelmän eri tasoilla. Toimeksiantajan toiveena oli kehittää pääsynhallintaan keskitetty malli, jossa käyttöoikeuksia voidaan hallita keskitetystä palvelusta.

Opinnäytetyöstä suurin osa ajasta meni tutkimusten ja dokumentaatioiden läpikäymiseen sekä uusien käsitteiden sisäistämiseen. Paljon aikaa meni myös erilaisten määrittelypalavereiden parissa ja erilaisten pilvinatiivien ympäristöjen opiskeluun.

Työn aihe oli haastava sen laajuuden vuoksi ja jatkuvasti esille tulleiden uusien käsitteiden vuoksi. Haasteena ei ollut tuotetun tekstin laajuus vaan päinvastoin pitää työssä selkeä johdonmukainen rakenne yllä. Työn sisällöstä tuli paljon suunniteltua laajempi aihepiirin kasvaessa esille nousseiden kysymysten ja haasteiden vuoksi. Työn selkeyden vuoksi jouduin jättämään pois joitakin tutkimiani aiheita. Näitä olivat käytössä olevat esimerkkiarkkitehtuurit, olemassa olevat autentikaatiolisäkirjastot ja -palvelut.

Työssä oli vaikeaa tuoda esiin, miten valittua ratkaisua voidaan toteuttaa teknisellä tasolla, mutta toisaalta se ei ollut työn perimmäisenä tarkoituksena. Olisin halunnut myös tutkia ja tuoda enemmän esiin suunnittelun käyttöoikeuspalvelimen ratkaisua ja toteutusta tietokantamalleineen. Työn laajuuden kasvaessa ja erilaisten määrittely ja aikataulukysymysten vuoksi tätä ei kuitenkaan saatu sisällytettyä lopulliseen työhön. Järjestelmän tutkiminen ja kehittäminen jatkuukin opinnäytetyöprosessin ulkopuolella.

Opinnäytetyö toi ratkaisuja joihinkin kysymyksiin, mitä toimeksiantajalla oli ratkaisuun liittyen. Tärkeimpänä saatiin kokonaiskuva, miten ja mihin suuntaan kehittämistä kannattaa viedä. Henkilökohtaisesti minulle tärkeintä oli saada työn aihepiirin tiimoilta laaja käsitys mikropalveluiden kehittämisestä pääsynhallinnan näkökulmasta ja hyvää oppia pääsynhallinnan toteutuksista. Kokonaisuutena olen tyytyväinen opinnäytetyöhön, vaikka välillä edistyminen oli hidasta erilaisten aikataulujen ja haasteiden vuoksi.

8 LÄHTEET JA TUOTETUT AINEISTOT

Amazon n.d. What is SOA (Service-Oriented Architecture)? Verkkojulkaisu. https://aws.amazon.com/what-is/service-oriented-architecture/) Viitattu 20.11.2024

Atlassian n.d. Advantages of microservices and disadvantages to know. Verkkojulkaisu. https://www.atlassian.com/microservices/cloud-computing/advantages-of-microservices. Viitattu 5.3.2025

Auth by Okta. n.d. JSON Web Token Claims. Verkkojulkaisu. https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-claims. Viitattu 24.2.2025

AWS Amazon. n.d. What's the Difference Between Monolithic and Microservices Architecture? Verkkojulkaisu. https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/. Viitattu 02.12.2024.

Barbanov, A. & Makrushin, D. 2020. Authentication and authorization in microservice-based systems: survey of architecture patterns. Verkkojulkaisu. https://arxiv.org/pdf/2009.02114. Viitattu 20.11.2024

Chin, S.-K. & Older, S. B. 2010. Access Control, Security, and Trust: A Logical Approach. Boca Raton: CRC Press.

Cloudflare. n.d.a. What is load balancing? How load balancers work. Verkkojulkaisu. https://www.cloudflare.com/learning/performance/what-is-load-balancing/. Viitattu 5.2.2025

Cloudflare. n.d.b. What is a reverse proxy? Proxy servers explained. Verkkojulkaisu. https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/ Viitattu 5.2.2025

Cybersecurity-Automation. n.d. What is Logical Access Control in Computer Security? Verkkojulkaisu. https://www.cybersecurity-automation.com/what-is-logical-access-control-in-computer-security/ Viitattu 02.12.2024

Di Pietro, G. 2022. What is Service Mesh and, especially, what is Istio? Verkkojulkaisu. https://isitobservable.io/observability/service-mesh/what-is-servicemesh-and-specially-what-is-istio. Viitattu 20.2.2025

F5. n.d. What Is an API Gateway? Verkkojulkaisu. https://www.f5.com/glossary/api-gateway. Viitattu 24.11.2024

Frontegg. 2025. Access Control Matrix: Key Components and 5 Critical Best Practices. Verkkojulkaisu. https://frontegg.com/blog/access-control-matrix. Viitattu 10.1.2025

Frontegg. 2022. Authentication in Microservices: Approaches and Techniques. Verkkojulkaisu. https://frontegg.com/blog/authentication-in-microservices. Viitattu 27.12.2024

GeeksforGeeks. 2025. What are Microservices? Verkkojulkaisu. https://www.geeksforgeeks.org/microservices/)/. Viitattu 5.1.2025

GeeksforGeeks. 2024. Edge Pattern in Microservices. Verkkojulkaisu. https://www.geeksforgeeks.org/edge-pattern-in-microservices/ Viitattu 24.11.2024

Google Cloud. n.d. What is Istio? Verkkojulkaisu. https://cloud.google.com/learn/what-is-istio?hl=fi. Viitattu 20.2.2025

Gosai, M. 2023. Everything you need to know about Laravel microservices. Verkkojulkaisu. https://www.prismetric.com/laravel-microservices/. Viitattu 12.2.2025

HAProxy. 2024. What is an access control list (ACL)? Verkkojulkaisu. https://www.haproxy.com/glossary/what-is-an-access-control-list-acl. Viitattu 28.2.2025

HAProxy. n.d. Open the door to better API management. Verkkojulkaisu. https://www.haproxy.com/solutions/api-gateway Viitattu 24.11.2024

HAProxy Documentation. 2025. Configuration Manual. Verkkojulkaisu. https://docs.haproxy.org/3.1/configuration.html. Viitattu 10.2.2025

HAProxy Technologies. 2021. HAProxy as an API Gateway. E-kirja. https://images.g2crowd.com/uploads/attachment/file/1278519/HAProxy-as-an-API-Gateway.pdf. Viitattu 8.2.2025

Heidi, E. 2025. A Practical Introduction to Laravel Eloquent ORM. Verkkojulkaisu. https://www.digitalocean.com/community/tutorial-series/a-practical-introduction-to-laravel-eloquent-orm. Viitattu 15.2.2024

IBM. 2021. Logical Access Control. Verkkojulkaisu. https://www.ibm.com/docs/en/ftms-wsfm300?topic=program-logical-access-control. Viitattu 2.12.2024

IBM. n.d. What is Istio? Verkkojulkaisu. https://www.ibm.com/think/topics/istio. Viitattu 20.2.2025

Ibrahim, M. 2024. What is a JWT? Understanding JSON Web Tokens. Verkkojulkaisu. https://supertokens.com/blog/what-is-jwt. Viitattu 22.2.2025

Istio. 2025. Documentation. Verkkojulkaisu. https://istio.io/latest/docs/. Viitattu 20.2.2025

Iyer, A. 2023. RBAC vs ReBAC: Which Is Right for You? Verkkojulkaisu. https://www.descope.com/blog/post/rbac-vs-rebac. Viitattu 28.2.2025

Jackson, G. & Goodwin, M. 2024. What is an API gateway? Verkkojulkaisu. https://www.ibm.com/think/topics/api-gateway. Viitattu 27.11.2024

Jones, M., Bradley, J. & Sakimura, N. 2015. JSON Web Token (JWT). Verkkojulkaisu. https://datatracker.ietf.org/doc/html/rfc7519. Viitattu 20.2.2025

JWT. n.d. Introduction to JSON Web Tokens. Verkkojulkaisu. https://jwt.io/introduction/. Viitattu 20.2.2025

Kanjilal, J. 2022. The role of sidecars in microservices architecture. Verkkojulkaisu. https://www.techtarget.com/searchapparchitecture/tip/The-role-of-sidecars-in-microservices-architecture. Viitattu 27.12.2024

Kaspersky. n.d. Mitä on Zero Trust -tietoturva? Keskeiset edut ja toimintaperiaate. Verkkojulkaisu. https://www.kaspersky.fi/resource-center/definitions/zero-trust. Viitattu 6.12.2024

Khatri, A. & Khatri, V. 2020 Mastering Service Mesh: Enhance, Secure, and Observe Cloud-native Applications with Istio, Linkerd, and Consul. Birmingham: Packt Publishing.

Kubernetes Documentation. 2025a. Pods. Verkkojulkaisu. https://kubernetes.io/docs/concepts/workloads/pods/. Viitattu 15.2.2025

Kubernetes Documentation. 2025b. Service. Verkkojulkaisu. https://kubernetes.io/docs/concepts/services-networking/service/. Viitattu 15.2.2025

Kubernetes Documentation. 2024. Nodes. Verkkojulkaisu. https://kubernetes.io/docs/concepts/architecture/nodes/. Viitattu 15.2.2025

Laravel. n.d. Laravel Documentation. Verkkojulkaisu. https://laravel.com/docs/11.x/. Viitattu 16.2.2025

Loft Team. 2023. A Guide to Using Kubernetes for Microservices. Verkkojulkaisu. https://www.loft.sh/blog/a-guide-to-using-kubernetes-for-microservices. Viitattu 12.2.2024.

Lombarte, A. 2024. Mastering Microservices Authorization: Strategies for Secure Access Control. Verkkojulkaisu. https://www.krakend.io/blog/microservices-authorization-secure-access/. Viitattu 27.12.2024

Lutkevich, B. 2022. Access control list (ACL). Verkkojulkaisu. https://www.techtarget.com/searchnetworking/definition/access-control-list-ACL. Viitattu 27.2.2025

Manor, G. 2024. Best Practices for Microservice Authorization. Verkkojulkaisu. https://www.permit.io/blog/best-practices-for-authorization-in-microservices. Viitattu 27.12.2024

Mashfej, S. 2023. Access Control for modern web applications. Verkkojulkaisu. https://supertokens.com/blog/access-control-for-modern-web-applications. Viitattu 10.12.2024

Medium. 2024. Types of access control lists (ACLs). Verkkojulkaisu. https://learningdaily.dev/types-of-access-control-lists-acls-60daca24ff8f.Viitattu 27.2.2025

Megnathan, N. n.d. Module 4: Access Control. PowerPoint-diat. Jackson State University. Viitattu 11.12.2024.

Microsoft. n.d.a Backends for Frontends pattern. Verkkojulkaisu. https://learn.microsoft.com/en-us/azure/architecture/patterns/backends-for-frontends. Viitattu 27.11.2024

Microsoft. n.d.b Use API gateways in microservices. Verkkojulkaisu. https://learn.microsoft.com/en-us/azure/architecture/microservices/design/gateway. Viitattu 28.11.2024

MOSN. 2024. Sidecar pattern. Verkkojulkaisu. https://mosn.io/en/docs/concept/sidecar-pattern/. Viitattu 3.1.2025

Nordlayer. n.d.a. What is discretionary access control (DAC)? Verkkojulkaisu. https://nordlayer.com/learn/access-control/discretionary-access-control/. Viitattu 27.2.2025

Nordlayer. n.d.b. What is mandatory access control? Verkkojulkaisu. https://nordlayer.com/learn/access-control/mandatory-access-control/. Viitattu 28.2.2025

Okta. 2024. RBAC vs. ABAC: Definitions & When to Use. Verkkojulkaisu. https://www.okta.com/identity-101/role-based-access-control-vs-attribute-based-access-control/. Viitattu 28.2.2025

OSO. n.d. Relationship-Based Access Control (ReBAC). Verkkojulkaisu. https://www.osohq.com/academy/relationship-based-access-control-rebac. Viitattu 28.2.2025

OWASP. n.d. Microservices Security Cheat Sheet. Verkkojulkaisu. https://cheatsheetseries.owasp.org/cheatsheets/Microservices_Security_Cheat_Sheet.html. Viitattu 6.1.2025

Parecki, A. n.d. Background. Verkkojulkaisu. https://www.oauth.com/oauth2-servers/background/. Viitattu 22.2.2025

Polencic, D. 2019. Can you expose your microservices with an API gateway in Kubernetes? Verkkojulkaisu. https://learnk8s.io/kubernetes-ingress-api-gateway. Viitattu 27.12.2024

Powell, P. & Smalley, I. 2024. What is monolithic architecture? Verkkojulkaisu. https://www.ibm.com/think/topics/monolithic-architecture. Viitattu 21.11.2024

Red Hat. 2019. What does an API gateway do? Verkkojulkaisu. https://www.redhat.com/en/topics/api/what-does-an-api-gateway-do. Viitattu 24.11.2024

Redhat. 2024. What is Kubernetes? Verkkojulkaisu. https://www.redhat.com/en/topics/containers/what-is-kubernetes. Viitattu 12.2.2025

Saroyan, K. n.d. What Is Access Control List (ACL) & Importance? Verkkojulkaisu. https://impanix.com/security/access-control-list-acl/. Viitattu 27.2.2025

Sereg, M. 2020. Build an Istio authorization policy: Introduction to Istio access control. Verkkojulkaisu. https://outshift.cisco.com/blog/istio-authorization-policies. Viitattu 20.2.2025

Shah, P. 2025. Business Logic vs Application Logic: Key Differences & Best Practices. Verkkojulkaisu. https://www.dhiwise.com/blog/requirement-builder/business-logic-vs-application-logic. Viitattu 20.2.2025

Solo.io. n.d. Kubernetes Egress. Verkkojulkaisu. https://www.solo.io/topics/service-mesh/kubernetes-egress. Viitattu 13.3.2025

Solomon, S. 2024. OAuth Explained: How It Works & Why It Matters. Verkkojulkaisu. https://frontegg.com/blog/oauth. Viitattu 22.2.2025

StrongDM. 2024. Zero Trust vs. the Principle of Least Privilege: What's the Differences? Verkkojulkaisu. https://www.strongdm.com/what-is/zero-trust-vs-principle-of-least-privilege. Viitattu 26.2.2025

Suonentieto n.d. Yritys. Verkkojulkaisu. https://www.suonentieto.fi/suonentieto-yrityksena/ Viitattu 5.3.2025

Surianarayanan, C., Ganapathy, G., & Pethuru, R. 2019. Essentials of microservices architecture : Paradigms, applications, and techniques. CRC Press LLC.

TechTarget. 2024. What is monolithic architecture in software? Verkkojulkaisu. https://www.techtarget.com/whatis/definition/monolithic-architecture. Viitattu 21.11.2024

Tigera. n.d. What Is Service Mesh in Kubernetes? 4 Tools to Get Started. Verkkojulkaisu. https://www.tigera.io/learn/guides/service-mesh/service-mesh-kubernetes/. Viitattu 4.1.2025

Upadhyay, M. 2024. Laravel Policies and Gates: Mastering Authorization in Your Web App. Verkkojulkaisu. https://wpwebinfotech.com/blog/laravel-policies-and-gates/. Viitattu 18.2.2025

VPN Unlimited. n.d.a. Reverse Proxy. Verkkojulkaisu. https://www.vpnunlimited.com/fi/help/cybersecurity/reverse-proxy. Viitattu 8.2.2025

VPN Unlimited. n.d.b. Rate Limiting. Verkkojulkaisu. https://www.vpnunlimited.com/help/cybersecurity/rate-limiting. Viitattu 10.2.2025

Walia, C. 2023. Mastering Cloud-Native Microservices: Designing and Implementing Cloud-Native Microservices for Next-Gen Apps. London: BPB Publications.

WizardCowOy. n.d. HAProxy ja kuormantasaus. Verkkojulkaisu. https://www.wizardcow.fi/blog-post/haproxy-ja-kuormantasaus/#page-content. Viitattu 2.2.2025

WMware. n.d.a. What is Kubernetes cluster? Verkkojulkaisu. https://www.vmware.com/topics/kubernetes-cluster. Viitattu 14.2.2025

WMware. n.d.b. What is Kubernetes deployment? Verkkojulkaisu. https://www.vmware.com/topics/kubernetes-deployment. Viitattu 14.2.2025