MICROSOFT 365 OFFICE ONLINEN INTEGROIMINEN SELAINPOHJAISEEN PILVITALLENNUSALUSTAAN

Lappeenrannan–Lahden teknillinen yliopisto LUT

Tietotekniikan kandidaatintyö

2022

Niilo Liimatainen

Tarkastaja: Apulaisprofessori Antti Knutas

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUT Teknis-luonnontieteellinen

Tietotekniikan koulutusohjelma

Niilo Liimatainen

Microsoft 365 Office Onlinen integroiminen selainpohjaiseen pilvitallennusalustaan

Tietotekniikan kandidaatintyö

41 sivua, 5 kuvaa, 1 taulukko

Tarkastaja: Apulaisprofessori Antti Knutas

Avainsanat: mikropalvelu, ohjelmointirajapinta, office online

Selainpohjaisista pilvitallennusalustoista on tullut suosittu keino keskittää dataa eri sidosryhmille saataville paikasta ja laitteesta riippumatta. Vertex Systems Oy on kehittämässä uutena tuotteena tähän tarkoitukseen Vertex Sync -pilvitallennusalustaa. Kehityksen yhteydessä esiin on noussut ongelma Microsoft 365 -tiedostojen käsittelyssä, joka ei tällä hetkellä ole mahdollista Vertex Syncissä. Työssä toteutetaan ratkaisu tähän ongelmaan integroimalla Office Online -editori Vertex Synciin mikropalvelun avulla. Lisäksi tutkitaan alan kirjallisuudesta, kuinka kyseinen mikropalvelu voidaan toteuttaa parhaiden käytäntöjen mukaisesti. Työn alussa kartoitetaan mikropalveluiden laatuominaisuuksiin liittyvää kirjallisuutta ja nostetaan esiin työn kannalta olennaisimpia aikaisempia tutkimuksia. Tämän jälkeen tutustutaan tarkemmin integraatioon vaadittavien komponenttien arkkitehtuuriin sekä tehdään selvitys Office Cloud Storage Partner -ohjelmasta. Lopulta rakennetaan mikropalvelu, jossa toteutetaan integraatioon vaadittavan WOPI-protokollan mukainen REST-ohjelmointirajapinta. Rajoitteena esiin nousi Office Cloud Storage Partner -ohjelman hakuprosessin pituus, jonka takia toteutuksen toiminnallisuus pystyttiin testaamaan vain paikallisessa ympäristössä. Toteutetun artefaktin perusteella voidaan kuitenkin todeta, että Office Online -editorin integroiminen Vertex Synciin on mahdollista ja mikropalvelu onnistuttiin luomaan kirjallisuudessa esiin nousseita laatuominaisuuksia seuraten.

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

School of Engineering Science

Degree Programme in Software Engineering

Niilo Liimatainen

Integrating Microsoft 365 Office Online into a web-based cloud storage service

Bachelor's thesis

2022

41 pages, 5 figures, 1 table

Examiner: Assistant Professor Antti Knutas

Keywords: microservice, application programming interface, office online

Web-based cloud storage services have become a popular way to centralize data for stakeholders. Vertex Systems Oy is developing its own product for this purpose named Vertex Sync. However, handling Microsoft 365 files in Vertex Sync has proven to be problematic. This thesis offers a solution to handling Microsoft 365 files by integrating the Office Online editor into Vertex Sync using a microservice. In addition, a brief literature review will be carried out to determine the best practices for implementing microservices. This thesis begins by presenting the most relevant previous studies regarding the quality features of microservices. After that, the architecture of the integration components will be presented, and the Office Cloud Storage Partner Program will be introduced. Ultimately, the microservice which implements a REST API according to the WOPI protocol is created. Because the length of the application process for the Cloud Storage Partner Program was proven to be longer than estimated, the created artifact could only be tested in a local environment. The results of this thesis show that the Office Online editor can be integrated into Vertex Sync, and the microservice was created successfully according to the best practices from the literature review.

LYHENTEET

CAD	Computer-Aided Design
WOPI	Web Application Open Platform Interface
CSPP	Cloud Storage Partner Program
REST	Representational State Transfer
API	Application Programming Interface
URI	Uniform Resource Identifier
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
SOA	Service Oriented Architecture
UML	Unified Modeling Language
IT	Information Technology
BIM	Building Information Modeling
SPA	Single Page Application
XML	Extensible Markup Language
URL	Uniform Resource Locator

Sisällysluettelo

Kuvaluettelo

Taulukkoluettelo

1 Johdanto

Jatkuvasti nousevat investointi- sekä ylläpitokustannukset ovat ajaneet ohjelmistoalaa kohti pilviympäristöjä. Pilveen siirtyminen on tuonut uusia dynaamisempia ratkaisuja tietojenkäsittelyyn, mikä on mahdollistanut täysin uuden tavan käyttää dataa laitteesta ja paikasta riippumatta. (Bojanova et al., 2013.) Markkinoille on tullut useita palveluntarjoajia, jotka tarjoavat selainpohjaisia ratkaisuja tiedon hallintaan ja muokkaamiseen. Yksi tällainen ohjelmisto on OneDrive, jossa dokumenttien pilvitallennusmahdollisuuksien lisäksi niitä pystytään muokkaamaan suoraan selaimessa selainpohjaisen editorin avulla (Wilson, 2015).

Ohjelmistojen kasvaessa yhä suuremmiksi ja monimutkaisemmiksi on niiden skaalaaminen, ylläpitäminen ja päivittäminen entistä hankalampaa. Tämä on johtanut hajautettujen ohjelmistokehitysmenetelmien suosion nousuun viime vuosien aikana. Useat suuret yritykset, kuten Amazon ja Uber, ovat siirtyneet suurista monoliittisista ohjelmista kohti itsenäisistä moduuleista koostuvia kokonaisuuksia mikropalveluarkkitehtuurin avulla (Singh & Peddoju, 2017). Mikropalveluarkkitehtuurissa ohjelmisto koostuu useasta keskenään kommunikoivasta itsenäisestä kokonaisuudesta. Näitä erillisiä kokonaisuuksia kutsutaan mikropalveluiksi, joita voidaan skaalata, päivittää ja ylläpitää toisistaan riippumatta. (Thönes, 2015.)

Pilviympäristöjen yleistyminen sekä järjestelmien hajauttaminen on antanut paremmat mahdollisuudet yhdistää eri palveluntarjoajien toteutuksia toisiinsa. Vertex Systems Oy on riippumaton teollisuuden ohjelmistotoimittaja, joka tarjoaa suunnitteluun ja tiedonhallintaan ohjelmistoratkaisuja. Uusimpana lisäyksenä tuoteperheeseen Vertex Systems Oy on julkaisemassa selainpohjaisen Vertex Sync -pilvitallennusalustan. Vertex Sync tulee tarjoamaan mahdollisuuden keskittää suunnittelutyöhön tarvittava data pilveen kaikille sidosryhmille saatavaksi. Pääosin alustaan tullaan tallentamaan 3D-malleja, jotka on tuotettu tietokoneavusteisilla suunnitteluohjelmilla (*engl. Computer-Aided Design, CAD*). Näiden CAD-ohjelmien tuottamien tiedostojen avuksi usein tallennetaan myös dokumentaatiota Microsoft 365 -tiedostoina.

Ongelmaksi on noussut nimenomaan näiden Microsoft 365 -tiedostojen katselu ja editoiminen sovelluksessa. Ensimmäiseksi tiedosto tulee ladata käyttäjän laitteelle. Tämän jälkeen käyttäjä joutuu avaaman ladatun tiedoston Microsoftin tarjoamassa työpöytäsovelluksessa. Jos tiedostoon tehdään muutoksia, tulee tämä tiedosto ladata uudestaan Vertex Synciin. Prosessi on käyttäjälle työläs, vaatii useita eri toimenpiteitä ja on käyttökokemukseltaan epämieluisa. Microsoft tarjoaa työpöytäsovelluksille myös vaihtoehtoisen ratkaisun, joka avaa uusia mahdollisuuksia Vertex Synciin. Office Online on selainpohjainen editori, joka tarjoaa Wordin, Powerpointin ja Excelin verkkoversiot. Ilmaiskäyttäjät saavat käyttöönsä vain katseluoikeudet, mutta maksavat käyttäjät saavat täysin samat luonti- ja editointioikeudet kuin työpöytäversioissakin.

1.1 Tavoitteet ja rajaukset

Tässä työssä lähdetään etsimään ratkaisua edellä mainittuun ongelmaan integroimalla Office Online -editori Vertex Synciin. Integraatiota varten työssä tullaan toteuttamaan erillinen mikropalvelu, joka kommunikoi Office Online -editorin kanssa WOPI (*Web Application Open Platform Interface*) -protokollaa käyttäen. Tämän lisäksi käyttöliittymään tulee tehdä iframe-kehys, jonka sisälle Office Online -editori upotetaan. Integraation onnistuessa Office Online -editoria pystytään käyttämään suoraan Vertex Syncin sisältä ilman erillisiä uudelleenohjauksia. Integroiminen vaatii liittymisen Microsoftin Office Cloud Storage Partner -ohjelmaan (*engl. Cloud Storage Partner Program, CSPP*), jonka hakuprosessi on useiden viikkojen pituinen. Kun ohjelmaan on päästy ja toteutus on valmis julkaistavaksi, on Microsoftilla vielä oma useamman viikon pituinen validointiprosessinsa, joka pitää sisällään manuaalisen tarkastuksen tehdylle toteutukselle. (Microsoft, 2021a.)

Työssä vastataan (2) tutkimuskysymykseen:

K1: Miten Office Online integroidaan pilvitallennusalustaan mikropalveluna?

K2: Miten mikropalvelu implementoidaan parhaiden käytäntöjen mukaisesti?

Tässä kandidaatintyössä toteutetaan integraatio CSPP-prosessin pituus huomioon ottaen ainoastaan kehitysympäristöön. Aikataulurajoitteiden takia ei lähdetä tekemään aiemmin mainittua validointiprosessia, jolla toteutus saataisiin julkaistavaksi asti. Tämän lisäksi työssä tullaan keskittymään ainoastaan tekniseen toteutukseen ja sen validointiin. Laajempia käsitteitä aiheen ympäriltä, kuten sovellusintegraatiota, ei käsitellä tässä työssä. Mikropalvelu tullaan toteuttamaan ASP.NET-viitekehyksellä, mutta tässä työssä ei mennä sen yksityiskohtiin. Mikropalvelun toteutus tullaan kuvaamaan korkealta tasolta, joka mahdollistaa tämän työn teknologiariippumattoman soveltamisen.

1.2 Työn rakenne

Työn rakenteesta lyhyesti. Luvussa 2 lähdetään avaamaan työhön liittyviä tärkeimpiä käsitteitä tutkimus- ja metodikirjallisuuden avulla. Tämän lisäksi tutustutaan työn kannalta oleellisiin aiempiin tutkimuksiin. Luvussa 3 esitellään työssä käytettävä tutkimusmenetelmä, jonka jälkeen tehdään tarkempi katsaus kohdesovellukseen sekä avataan integraation ja siihen liittyvän WOPI-protokollan arkkitehtuuria luvussa 4. Kun pohja integraatiota varten on selvitetty, toteutetaan työn varsinainen artefakti luvussa 5. Luvussa 6 esitellään työn tulokset ja arvioidaan toteutettua järjestelmää kirjallisuuteen peilaten. Lopuksi kootaan johtopäätökset tehdystä selvityksestä sekä toteutetusta prototyypistä.

2 Kirjallisuuskatsaus

Tässä luvussa tutustutaan käsitteeseen REST (*Representational State Transfer)* sekä käydään tarkemmin läpi mikropalveluarkkitehtuuria alan kirjallisuuden avulla. Lisäksi esitellään aiempia tutkimuksia, joissa käsitellään yleisesti mikropalveluarkkitehtuurin ongelmakohtia ja niiden välttämiskeinoja. Näihin tutkimuksiin tullaan tukeutumaan työn teknisessä toteutuksessa. Tieteellistä kirjallisuutta etsittiin LUT Primo ja Google Scholar -palveluista. Lähteiden laadun tarkastukseen käytettiin Julkaisufoorumi-palvelua.

2.1 REST-arkkitehtuurimalli

Ohjelmointirajapinta (*engl. Application Programming Interface, API)* on ulospäin näkyvä liittymäkohta, josta asiakasohjelmat pääsevät käsiksi toteutettuun toiminnallisuuteen. Ohjelmointirajapinnat mahdollistavat koodin uudelleenkäytön sekä korkean abstraktiotason. Näin ollen ohjelmointirajapinnan käyttäjän ei tarvitse olla tietoinen mitä sen sisällä olevassa toteutuksessa tapahtuu. (Robillard, 2009.)

REST on arkkitehtuurimalli ohjelmointirajapintojen luomiseen. REST-arkkitehtuurimallissa määritellään seuraavat perusperiaatteet, jotka ohjaavat ohjelmointirajapinnan toteutusta:

- Jokainen resurssi tulee olla yksilöllisesti tunnistettavissa yhtenäisellä resurssitunnisteella (*engl. Uniform Resource Identifier, URI).*
- Resurssien hakuun ja manipulointiin käytetään HTTP (Hypertext Transfer Protocol) -siirtoprotokollan määrittelemiä metodeja.
- Resurssien esitysmuoto, esimerkiksi JSON (*JavaScript Object Notation),* on määriteltävä HTTP-viestien ylätunnisteissa (*engl. header).*
- Asiakasohjelman ja ohjelmointirajapinnan välinen vuorovaikutus tulee olla tilatonta. Jokaisen pyynnön tulee sisältää kaikki tarvittava tieto, jota ohjelmointirajapinta tarvitsee sen käsittelyyn.

- Jos resurssilla on yhteyksiä toiseen resurssiin, tulee tämä olla nähtävissä resurssin metatiedoissa

Kun ohjelmointirajapinta on rakennettu näiden rajoitteiden mukaisesti, voidaan siitä käyttää nimitystä REST-ohjelmointirajapinta (REST API). (Rodríguez et al., 2016.)

2.2 Mikropalveluarkkitehtuuri yleisesti

Mikropalveluarkkitehtuuri on varianssi jo aikaisemmin suosioon nousseesta palvelukeskeisestä arkkitehtuurista (*engl. Service Oriented Architecture, SOA*). Pääperiaatteena on, että ohjelmisto koostuu useasta itsenäisestä kokonaisuudesta, jotka kommunikoivat keskenään ja täten toteuttavat halutun toiminnallisuuden. (Dragoni et al., 2017; Larrucea et al., 2018; Newman, 2021.)

Termi ”mikropalvelu” esiteltiin ensimmäisen kerran vuonna 2011 ja se on ollut nopeasti nouseva trendi siitä asti (Dragoni et al., 2017). Thönes (2015) määrittelee julkaisussaan mikropalvelun pieneksi applikaatioksi, joka pystytään julkaisemaan, skaalaamaan ja testaamaan itsenäisesti erillään muista. Julkaisussa painotetaan, että yksittäisellä mikropalvelulla on vain ja ainoastaan yksi tehtävä, jonka se toteuttaa. Newman (2021) puolestaan esittelee vuonna 2021 julkaistussa kirjassaan mikropalvelut verkkokauppaesimerkin avulla. Yksi mikropalvelu toteuttaa tavaraluettelon, toinen mikropalvelu toteuttaa tilausten käsittelyn ja kolmas mikropalvelu hoitaa kuljetuksen. Yhdessä nämä kolme mikropalvelua muodostavat toimivan verkkokaupan. (Newman, 2021.)

Mikropalveluarkkitehtuurin modulaarisuus tuo mukanaan useita hyötyjä (Familiar, 2015; Nadareishvili *et al.*, 2016; Wolff, 2016):

- Itsenäisen luonteen ansiosta mikropalveluiden skaalaaminen, päivittäminen ja mahdollinen korjaus voidaan tehdä erillään muusta toiminnallisuudesta. Virheet

pystytään helposti paikallistamaan, eikä yksittäisen mikropalvelun virhe todennäköisesti kaada koko sovellusta.

- Yksittäinen mikropalvelun julkaiseminen on ketterämpää ja helpommin hallittavampaa kuin suuren monoliittisen sovelluksen.
- Mikropalvelut voidaan toteuttaa eri teknologioilla toisistaan riippumatta. Jokaista mikropalvelua varten voi olla oma tietyn osaamisalueen kehitystiimi, mikä nopeuttaa kehittämistyötä huomattavasti perinteisestä monoliittisesta sovelluksesta.
- Korkeampi motivaatio kokeilla uusia teknologioita. Jos teknologia ei sovellukaan yritykseen, voi yksittäisen mikropalvelun helposti poistaa käytöstä.

Vaikka mikropalveluarkkitehtuuri tuo mukanaan paljon hyötyjä, on siinä myös haasteita. Mikropalvelut vaativat kokenutta henkilökuntaa korkean oppimiskäyrän takia (Larrucea et al., 2018). Tästä johtuen yksinkertaisissa ja pienemmissä sovelluksissa mikropalveluarkkitehtuurin käyttö voi hidastaa toteutuksen julkaisua ja nostaa kuluja monoliittiseen arkkitehtuuriin verrattuna (Jamshidi et al., 2018). Jamshidi et al. (2018) nostavat julkaisussaan myös esiin mikropalveluiden hankalan määrittelemisen. Eri osa-alueiden erottaminen järkevästi voi olla haastavaa ja altista virheille. Missään ei ole myöskään määritelty kuinka suuri yhden mikropalvelun tulisi olla, mikä voi johtaa suuriin eroavaisuuksiin toteutusten välillä. Huonolla suunnittelutyöllä voi arkkitehtuuri johtaa korkeaan verkkoviestinnän määrään ja suorituskyvyn laskuun. Jamshidi et al. (2018) julkaisussa nostetaan esiin myös kasvava monitoroinnin ja ylläpidon määrä. Koska jokainen mikropalvelu julkaistaan omana kokonaisuutenaan, joudutaan näille kokonaisuuksille tehdä omat infrastruktuurit, joita pitää valvoa ja ylläpitää.

2.3 Mikropalveluiden välinen kommunikaatio

Yksi vaikeimmista suunnitteluhaasteista mikropalveluiden toteutuksessa on niiden välinen kommunikaatio. Mikropalvelun toteutukseen kuuluu usein oma sisäinen tietokanta, johon muut eivät pääse suoraan käsiksi. Itsenäisen luonteen takia yksinkertaiset funktiokutsut

mikropalveluiden välillä eivät ole mahdollisia, vaan viestinnän on tapahduttava verkon välityksellä ohjelmointirajapinnan kautta. (Bucchiarone et al., 2020.)

Kun mikropalvelu on toteutettu oikeaoppisesti, siitä näkyy ulkopuolelle ainoastaan rajapinta, jossa on määriteltynä päätepisteet, joiden kautta päästään käsiksi mikropalvelun toiminnallisuuteen. Suosituin toteutus mikropalveluiden väliseen kommunikointiin on aiemmin esitelty REST-arkkitehtuurimalli, joka hyödyntää kevyttä HTTP-siirtoprotokollaa. Synkronisen HTTP-protokollan lisäksi mikropalveluiden väliseen kommunikointiin voidaan käyttää esimerkiksi palveluväyliä, joissa viestit kulkevat tallennus- ja lähetysjonoissa asynkronisesti. Näihin väyliin mikropalvelut voivat liittyä tarpeen mukaan. (Familiar, 2015.)

2.4 Aiempi tutkimus

Alshuqayran et al. (2016) toteuttivat artikkelissaan systemaattisen kirjallisuuskatsauksen mikropalveluarkkitehtuurin teoriasta ja sen implementoinnista.

Tutkimuksessa keskityttiin erityisesti kolmeen asiaan:

- Mikropalveluarkkitehtuurin haasteiden tunnistaminen
- Mikropalveluarkkitehtuurin mallintamiskeinojen tunnistaminen ja tutkiminen
- Mikropalveluarkkitehtuurin laatuominaisuuksien tunnistaminen

Tutkimuksessa nostettiin esiin suurimmaksi haasteeksi mikropalveluiden välinen kommunikaatio. Muita haasteita oli muun muassa mikropalveluiden tietoturvassa, suorituskyvyssä sekä käyttöönotossa. Artikkelissa havaittiin, että kirjallisuus mikropalveluarkkitehtuurin haasteiden ympärillä oli hyvin ratkaisukeskeistä, eikä esimerkiksi kokemusraportteja juuri löytynyt. Alshuqayran et al. (2016) löysivät useita eri mikropalveluarkkitehtuurin mallintamiskeinoja, kuten UML (*engl. Unified Modeling Language*) -kaaviot ja vapaasti piirretyt diagrammit. Näitä keinoja tullaan käyttämään tässä työssä mallintamisen apuna.

Tutkimuksessa huomattiin, että useissa lähteissä samaa tarkoittavista laatuominaisuuksista käytettiin useita eri nimityksiä. Artikkeliin kerättiin taulukko, jossa laatuominaisuudet ovat esiteltynä. Näistä tärkeimmiksi nostettiin skaalautuvuus, itsenäisyys ja ylläpidettävyys. Tässä työssä näitä laatuominaisuuksia pystytään hyödyntämään toteutetun järjestelmän arvioinnissa.

Waseem et al. (2021) julkaisemassa artikkelissa toteutettiin empiirinen tutkimus viidelle eri mikropalveluarkkitehtuurilla toteutetulle systeemille. Jokaisessa näistä systeemeistä oli avoin lähdekoodi, johon päästiin käsiksi GitHub-versionhallintajärjestelmän kautta. Empiirisen tutkimuksen tavoitteena oli löytää ja luokitella mikropalveluarkkitehtuurissa esiintyvät ongelmat sekä syyt, jotka johtavat näihin ongelmiin. Tutkimus toteutettiin keräämällä ongelmakohtiin liittyviä keskusteluita aiemmin mainittujen systeemien lähdekoodikirjastoista. Näitä keskusteluita löydettiin yhteensä 1 345, jotka analysoitiin manuaalisesti eri kategorioihin. Tutkimuksen perusteella suurimmiksi ongelmakategorioiksi nousivat tekninen velka, turvallisuus, mikropalveluiden välinen kommunikaatio sekä käyttöönotto. Syitä näille ongelmille löytyi useita, mutta tärkeimpänä mainittiin ohjelmointivirheet, koodihajut, heikko suunnittelu sekä riittämättömät resurssit.

Taibi ja Lenarduzzi (2018) tekemässä tutkimuksessa haastateltiin 72 ohjelmistokehittäjää, joilla oli aikaisempaa kehittämiskokemusta mikropalveluarkkitehtuurista. Tutkimuksen tavoitteena oli selvittää mikropalveluarkkitehtuuriin liittyvät koodihajut (*engl. code smells)*. Tutkimuksessa määriteltiin koodihajut huonon suunnittelun oireiksi, jotka mahdollisesti hankaloittavat koodin ymmärrettävyyttä sekä ylläpidettävyyttä. Haastatteluissa nousi esiin yhteensä 265 koodihajua, joita haastateltavat olivat kokeneet kehittäessään mikropalveluita. Näistä havainnoista muodostettiin 11 koodihajun lista, joita tullaan käyttämään tässä työssä kehittämisen tukena.

Savchenko et al. (2015) puolestaan tekivät vuonna 2015 tutkimuksen, jossa analysoitiin tapaustutkimuksen avulla olemassa olevia metodeja mikropalveluiden validointiin. Tämän lisäksi luotiin toimintamenetelmä yksittäisen mikropalvelun yksikkö- sekä integraatiotestaukseen. Tapaustutkimuksessa käytettiin esimerkkiprojektia nimeltä Mjolnirr. Mjolnirr on

prototyyppi hajautetusta järjestelmästä, joka on toteutettu mikropalveluarkkitehtuurin mukaisesti. Se koostuu useasta itsenäisestä komponentista, jotka kommunikoivat keskenään ohjelmointirajapintojen kautta.

Savchenko et al. (2015) toteavat, että mikropalveluiden itsenäisen luonteen takia ei pelkkä perinteinen systeemitestaus riitä, vaan mikropalvelut tulee testata myös yksikkökohtaisesti. Tämän työn kannalta juuri nämä yksikkötestaukseen liittyvät tulokset olivat erityisen mielenkiintoiset. Tutkimuksessa esitettiin, että yksikkötestausta varten mikropalvelun käyttöympäristöä voidaan jäljitellä paikallisesti, joten sitä ei siis tarvitse ajaa oikeassa pilviympäristössä validointia varten.

Mikropalvelun yksikkötestaus voidaan jakaa kolmeen eri osa-alueeseen (Savchenko et al., 2015):

- Toiminnallisuus
- Kuormitus
- Tietoturva

Toiminallisessa testauksessa validoidaan mikropalvelun eheys sen ohjelmointirajapinnan kautta. Tarkistetaan siis, että ohjelmointirajapinnan päätepisteet toimivat suunnitellulla tavalla. Kuormitustestauksessa mikropalveluun kohdistetaan suuri määrä pyyntöjä, joilla pystytään validoimaan sen toimivuus suuren kuorman alla. Viimeisenä osa-alueena on tietoturvatestaus, jossa mikropalvelua kohtaan ajetaan tunnettuja tietoturvahyökkäyksiä. Tutkimuksessa kuitenkin esitetään, että tämän osa-alueen testaus tulisi olla jonkun muun vastuulla kuin mikropalvelun kehittäneen henkilön. (Savchenko et al., 2015.)

3 Tutkimusmenetelmät

Tutkimusmenetelmänä tässä kandidaatintyössä käytetään suunnittelutiedettä (engl. Design Science). Suunnittelutiede on IT (*engl. Information Technology)* -alalla yleisesti käytössä oleva tutkimusmenetelmä, jossa ratkaistaan ongelmia luomalla uusia toteutuksia. Näitä toteutuksia kutsutaan artefakteiksi, joita luomalla ja jälkeenpäin arvioimalla pystytään jakamaan ja tuottamaan uutta tietoa. (Johannesson & Perjons, 2014.) Tässä työssä artefaktilla tarkoitetaan integraatiokokonaisuutta, joka muodostuu mikropalvelusta sekä käyttöliittymäkomponentista. Suunnittelutiede tutkimusmenetelmänä on laaja kokonaisuus, joka pitää sisällään useita erilaisia määritelmiä sekä työtapoja. Tässä työssä ei ole tarpeen syventyä tarkemmin suunnittelutieteen erilaisiin menetelmiin, vaan perusperiaatteiden seuraaminen riittää.

Hevner et al. (2004) asettavat tutkimuksessaan suunnittelutieteelle seitsemän ohjenuoraa, joita tässä työssä pyritään noudattamaan:

- Tutkimuksessa on tuotettava artefakti joko konstruktiona, mallina, menetelmänä tai ilmentymänä.
- Tutkimuksessa tulee kehittää teknologiakeskeinen ratkaisu relevanttiin ongelmaan.
- Artefaktin toteutus sekä arviointi tulee pohjautua todennettavissa olevaan tieteelliseen teoriaan.
- Artefaktin toteutukseen tulee etsiä optimaalinen ratkaisu.
- Artefaktin laatu sekä hyödyllisyys tulee osoittaa kokonaisvaltaisen arvioinnin avulla.
- Tutkimuksessa tulee tuottaa uutta tietoa tutkimusalueelle.
- Tulokset tulee esitellä.

Hevner et al. (2004) esittelemät suuntaviivat antavat selkeän viitekehyksen tälle työlle. Työssä on yksiselitteinen ongelma, joka on tarkemmin esiteltynä johdannossa. Tähän

ongelmaan lähdetään luomaan ratkaisua luvun 2 kirjallisuuteen pohjautuen. Samaan kirjallisuuteen tukeudutaan myös artefaktin arvioinnissa työn tulosten esittelyssä. Tulosten ja yhteenvedon yhteydessä tulee esiin myös tutkimusalueelle tuotettu lisäarvo. Tulokset esitellään tässä raportissa ja työn loppuvaiheessa pidettävässä seminaarissa.

4 Arkkitehtuuri

Tässä luvussa tarkastellaan arkkitehtuuria työn toteutuksen takana. Ensimmäisessä alaluvassa esitellään integraation kohdesovellus Vertex Sync. Tämän jälkeen tehdään tarkempi katsaus Microsoftin tarjoamaan Office 365 Cloud Storage Partner -ohjelmaan ja sen vaatimuksiin. Lopuksi käsitellään Microsoftin luomaa WOPI-protokollaa, jolla integraatio tullaan toteuttamaan.

4.1 Vertex Sync

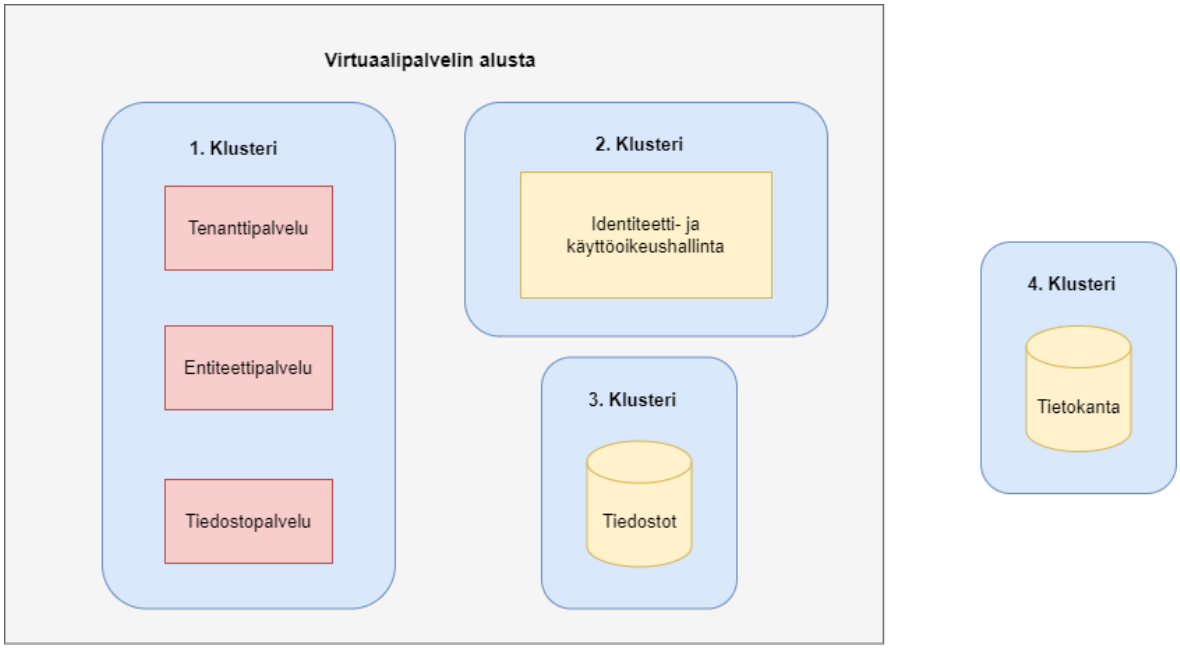
Vertex Sync on myöhemmin tänä vuonna julkaistava selainpohjainen versionhallintajärjestelmä. Ensimmäinen versio Vertex Syncistä on suunnattu Vertex BD:n käyttäjille, mutta myöhemmin tuki tullaan lisäämään muihinkin sovelluksiin. Vertex BD on BIM (*Building Information Modeling)* -ohjelmisto teolliseen rakennussuunnitteluun. Vertex BD:n avulla pystytään hallitsemaan rakennussuunnittelun eri prosesseja myynnistä asennukseen asti. (Vertex Systems Oy, 2022a)

Tähän asti Vertex BD:llä tuotetut projektit, piirustukset, mallit, raportit ja komponentit on tallennettu paikallisesti asiakkaan laitteelle tai yrityksen serverille. Tämä luonnollisesti johtaa siihen, ettei etätyöskentely näiden dokumenttien parissa ole ollut mahdollista. Vertex Sync tarjoaa tähän ratkaisun keskitetyllä pilvipohjaisella versionhallintajärjestelmällä. Jatkossa Vertex BD:n tuottamat dokumentit voi suoraan tallentaa Vertex Systems Oy:n ylläpitämään pilvitallennustilaan, josta asiakkaat pääsevät niihin käsiksi paikasta ja laitteesta riippumatta. Dokumentit tulevat synkronoitumaan automaattisesti, jolla tässä yhteydessä tarkoitetaan sitä, että ne pysyvät aina ajan tasalla Vertex Syncin sisällä. Tämä mahdollistaa paikasta riippumattoman tiimityöskentelyn sekä dokumenttien helpon jaettavuuden eri sidosryhmien välillä. (Vertex Systems Oy, 2022b)

Vertex Sync on selainpohjainen sovellus, joka koostuu kahdesta eri kokonaisuudesta. Taustajärjestelmä (*engl. backend)* on toteutettu mikropalveluarkkitehtuurilla ja käyttöliittymä (*engl. frontend)* puolestaan Angular-viitekehyksen avulla. Näiden itse toteutettujen moduuleiden lisäksi käyttöoikeushallinta, tietokanta sekä virtuaalipalvelinalusta on ulkoistettu erillisille palveluntarjoajille.

Tehdään ensin tarkempi katsaus tämän työn kannalta pienemmässä roolissa olevaan käyttöliittymän toteutukseen. Toteutus on tehty SPA (*Single Page Application)* -arkkitehtuurilla. SPA-arkkitehtuurissa näkymät eivät ole kokonaisia HTML (*Hypertext Markup Language)* -sivuja vaan ainoastaan osia siitä. Näin näkymää vaihtaessa selaimen ei tarvitse ladata sivua uudestaan. Kaikki logiikka näkymien vaihtamiseen ja niihin datan lisäämiseen on niin kutsutun SPA-viitekehyksen sisällä. (Scott, 2015.) Vertex Syncissä tässä roolissa toimii Angular-viitekehys, joka tarjoaa tehokkaat työkalut SPA-arkkitehtuurilla toteutetun käyttöliittymän luontiin (Angular, 2022).

Jotta Vertex Syncin käyttöliittymään saadaan dataa ja siellä tehtäville operaatioille toiminnallisuus, tarvitaan tueksi taustajärjestelmä. Vertex Syncin taustajärjestelmä koostuu useasta eri moduulista, joista osa on ulkoistettuja ja osa itse implementoituja. Kuvassa 1 on esiteltynä yksinkertaistettu korkean tason arkkitehtuurikuvaus Vertex Syncin taustajärjestelmästä, josta nähdään, että arkkitehtuuri koostuu useasta erillisestä klusterista. Klusterit 1, 2 ja 3 pyörivät ulkoistetun virtuaalipalvelin alustan sisällä ja klusteri 4 toimii sen ulkopuolella.



Kuva 1. Vertex Syncin taustajärjestelmä

Klusteri 1 on tämän työn kannalta olennaisin, koska se pitää sisällään kaikki itse implementoidut mikropalvelut. Nämä mikropalvelut tarjoavat jo aikaisemmin esitellyn Rodríguez et al. (2016) tekemän julkaisun mukaiset REST-ohjelmointirajapinnat. Näiden rajapintojen kautta Angular-viitekehyksellä toteutettu selainpuolen toteutus sekä Vertex BD pääsevät käsiksi mikropalveluiden toiminnallisuuteen. Kuvan 1 arkkitehtuurimallissa on kuvattu tämän työn kannalta olennaisimmat mikropalvelut, jotka ovat tenantti-, entiteetti- ja tiedostopalvelu. Tenanttipalvelu on vastuussa asiakasorganisaatioiden ja niiden käyttäjien hallinnasta. Entiteettipalvelu vastaa erilaisten entiteettien ja niiden välisten liitännäisten ylläpidosta. Entiteetillä tässä kontekstissa tarkoitetaan vailla tarkempaa määritystä olevaa objektikokonaisuutta esimerkiksi projektia. Projekti voi pitää allaan lukemattomia pienempiä entiteettejä, kuten 3D-malleja, piirustuksia tai Microsoft 365 -tiedostoja. Entiteeteissä on ainoastaan viitteet oikeaan tiedostoon, jonka käsittelystä vastaa tiedostopalvelu. Tiedostopalvelun vastuulla on kaikki binäärisiin tiedostoihin liittyvä hallinnointi. Tässä työssä implementoitava mikropalvelu tullaan myöhemmin lisäämään edellä mainittujen mikropalveluiden joukkoon klusterin 1 sisään.

Vertex Sync mukailee multitenantti-arkkitehtuuria, joka on toteutettu klusterissa 2 toimivalla ulkoistetulla identiteetti- ja käyttöoikeushallintapalvelulla. Multitenantti-käsitteellä tarkoitetaan arkkitehtuurimallia, jossa yksi instanssi sovelluksesta jakautuu usean eri tenantin käyttöön. Tenantilla puolestaan tarkoitetaan entiteettiä, joka niin sanotusti vuokraa sovellusinstanssin käyttöönsä. (Bezemer et al., 2010.) Vertex Syncin kontekstissa tenantilla tarkoitetaan asiakasorganisaatiota. Näiden asiakasorganisaatioiden hallinta tapahtuu siis ulkoistetussa identiteetti- ja käyttöoikeushallintapalvelussa, joka tarjoaa työkalut organisaatioiden ja niiden käyttäjien tallentamiseen, todennukseen sekä valtuutukseen. Näistä työkaluista käyttäjien todennus ja valtuutus tulevat olemaan merkittävässä roolissa tässä integraatiossa.

Tiedon tallennukseen Vertex Syncissä on kaksi klusteria, jotka ovat klusterit 3 ja 4. Klusteri 3 pitää sisällään pilvitallennustilan varsinaisia tiedostoja varten. Tämä tallennustila on ulkoistettu samalle palveluntarjoajalle kuin virtuaalipalvelinalusta, jossa klusterit 1, 2 ja 3 pyörivät. Microsoft 365 tiedostot, joita tässä työssä tullaan käsittelemään, tallennetaan tämän klusterin sisään. Viimeisenä klusterina toimii täysin ulkoistettu tietokanta, johon tallennetaan erilaisia objekteja ja metadataa tiedostoista. Tämä klusteri on täysin erillään muusta toteutuksesta ja on näin ollen ainoa klusteri, joka ei toimi virtuaalipalvelinalustan sisällä.

4.2 Office Cloud Storage Partner -ohjelma

Aiemmin mainittu Office Cloud Storage Partner -ohjelma on Microsoftin tarjoama kumppanuusohjelma, joka on tarkoitettu itsenäisille ohjelmistontuottajille, jotka tarjoavat pilvitallennuspalveluita kolmansille osapuolille. Jos näiden itsenäisten ohjelmistontuottajien pilvitallennuspalveluun tallennetaan paljon Microsoft 365 -tiedostoja ja yritys haluaa tarjota asiakkailleen selainpohjaiset editointimahdollisuudet niihin, tulee yrityksen hakea kumppanuusohjelmaan.

Jotta tähän ohjelmaan pääsee mukaan, tulee tuotteen täyttää seuraavat ehdot (Microsoft, 2022):

- Pilvitallennusympäristö on oltava kokonaan kumppaniyrityksen omistama.
- Muokkausominaisuudet tulee tarjota Microsoft Excel-, Word- ja PowerPoint-tiedostoille.
- Toteutus tulee olla käytettävissä vain Microsoft 365 -lisensoiduille käyttäjille.

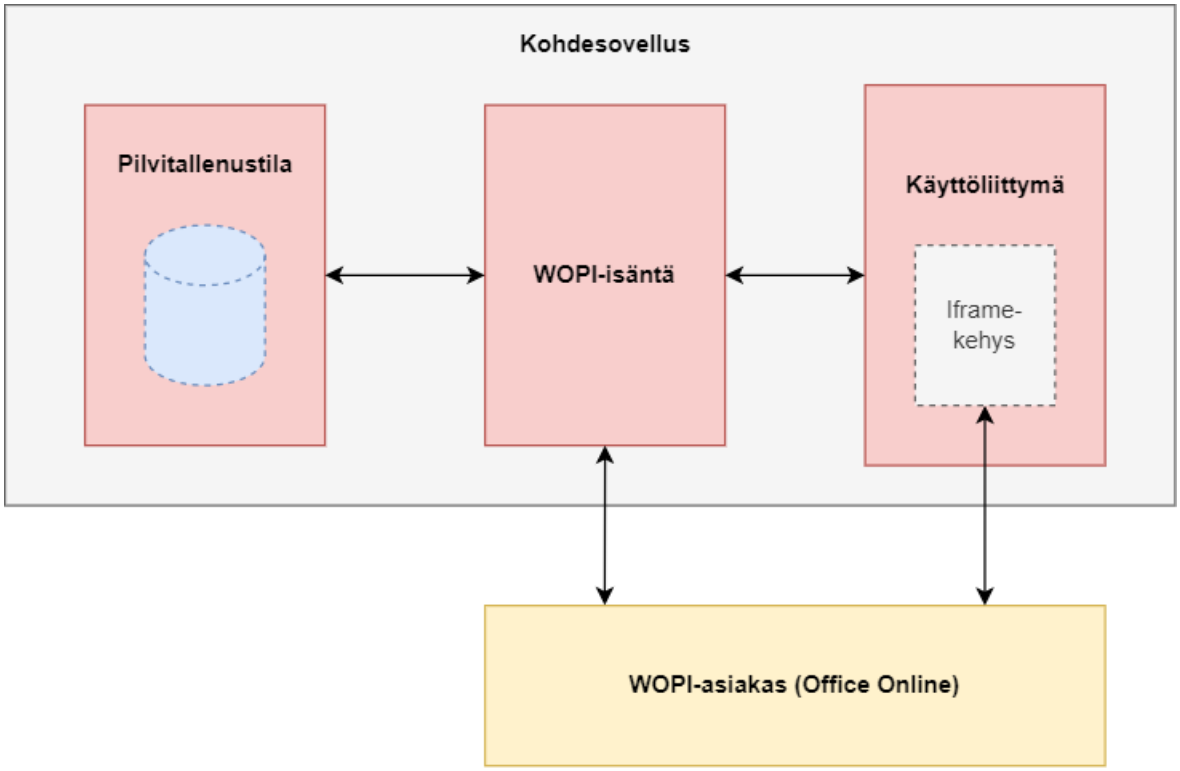
Microsoft (2022) tarjoaa myös mahdollisuuden syvempään integraatioon CSPP Plus -ohjelmallaan. Tämän ohjelman avulla kumppaniyritys saisi käyttöönsä muutamia lisäominaisuuksia, kuten korkeammat kokorajoitukset tiedostoihin sekä Microsoftin Intune -mobiilihallintatyökalun. Tässä työssä standarditason kumppanuus on kuitenkin riittävä.

Hakuprosessi CSPP-ohjelmaan on kokonaisuudessaan moniosainen ja voi kestää yhteensä jopa useita kuukausia. Jos mahdollinen kumppaniyritys hyväksytään ohjelmaan, saavat he pääsyn kattavaan dokumentaatioon sekä keskustelukanaviin, joissa mahdollisia ongelmakohtia voi ratkoa muiden kumppaniyritysten ja Microsoftin tukihenkilöiden kanssa. (Microsoft, 2022.)

Microsoft on hyvin tarkka toteutetun integroinnin laadusta. Kehitysympäristöön Microsoft tarjoaa WOPI-validointisovelluksen, jota vasten integraatiota pystytään ajamaan. Validointisovellus testaa kaikki tarvittavat toiminnot sekä rajapinnat, joita oikea toteutus tulee käyttämään. Integraation julkaisuprosessin voi aloittaa, kun se läpäisee Microsoftin tarjoaman validointisovelluksen testit. Julkaisuprosessissa Microsoft validoi toteutuksen käyttöliittymäintegraation sekä WOPI-protokollan mukaisen rajapinnan toiminnan. Tämä tapahtuu erilaisten kysymysten, manuaalisten testien sekä validointisovelluksen avulla. Microsoftin dokumentaatio tarjoaa kokonaisvaltaiset ohjeet, kuinka valmistautua tähän prosessiin. Dokumentaation mukaan tähän validointiin tulee varata noin kuukausi aikaa. Kun integraatio on saanut hyväksynnän Microsoftin päästä, voidaan se julkaista asiakkaille. (Microsoft, 2021a.)

4.3 WOPI-protokollan komponentit

WOPI (Web Application Open Platform Interface) on patentoitu protokolla, jota Microsoft käyttää sen tuotteiden integraatioissa. Tiedostojen, operaatioiden ja metadatan siirto pilvipalvelun ja Office Onlinen välillä tapahtuu tämän protokollan avulla. Yleisen tason arkkitehtuuri WOPI-protokollan komponenteista on kuvattuna alla olevassa kuvassa 2.



Kuva 2. WOPI-protokollan komponentit. Muokattu lähteestä (Palmer et al., 2020)

Kuvasta 2 nähdään, että WOPI-protokollan kannalta olennaisimmat komponentit ovat WOPI-asiakas (*engl. WOPI client)* sekä kohdesovelluksen sisällä olevat WOPI-isäntä (*engl. WOPI host)* ja käyttöliittymä. Keltaisella taustavärillä oleva WOPI-asiakas on komponenteista ainoa, joka on Microsoftin ylläpitämä. Muut komponentit kuuluvat kohdesovellukseen ja ovat näin ollen kumppaniyrityksen tuottamia. Kuvasta nähdään nuolten avulla komponenttien väliset kommunikointisuhteet. Kaikki kommunikaatio WOPI-protokollassa on kaksisuuntaista.

WOPI-isäntä on kohdesovelluksen taustajärjestelmään luotu palvelu, jonka kautta Office Online pääsee käsiksi kohdesovelluksen pilvitallennustilan Microsoft 365 -tiedostoihin. Tiedostoihin liittyvät operaatiot toteutetaan WOPI-isäntään implementoidun REST-ohjelmointirajapinnan kautta. Tämä ohjelmointirajapinta sisältää WOPI-protokollassa tarkoin määritellyt WOPI-päätepisteet, joita WOPI-asiakas käyttää. WOPI-päätepisteiden lisäksi WOPI-isäntä tarjoaa kohdesovelluksen käyttöliittymälle rajapinnan, jonka kautta käyttöliittymä saa tietoa WOPI-asiakkaasta.

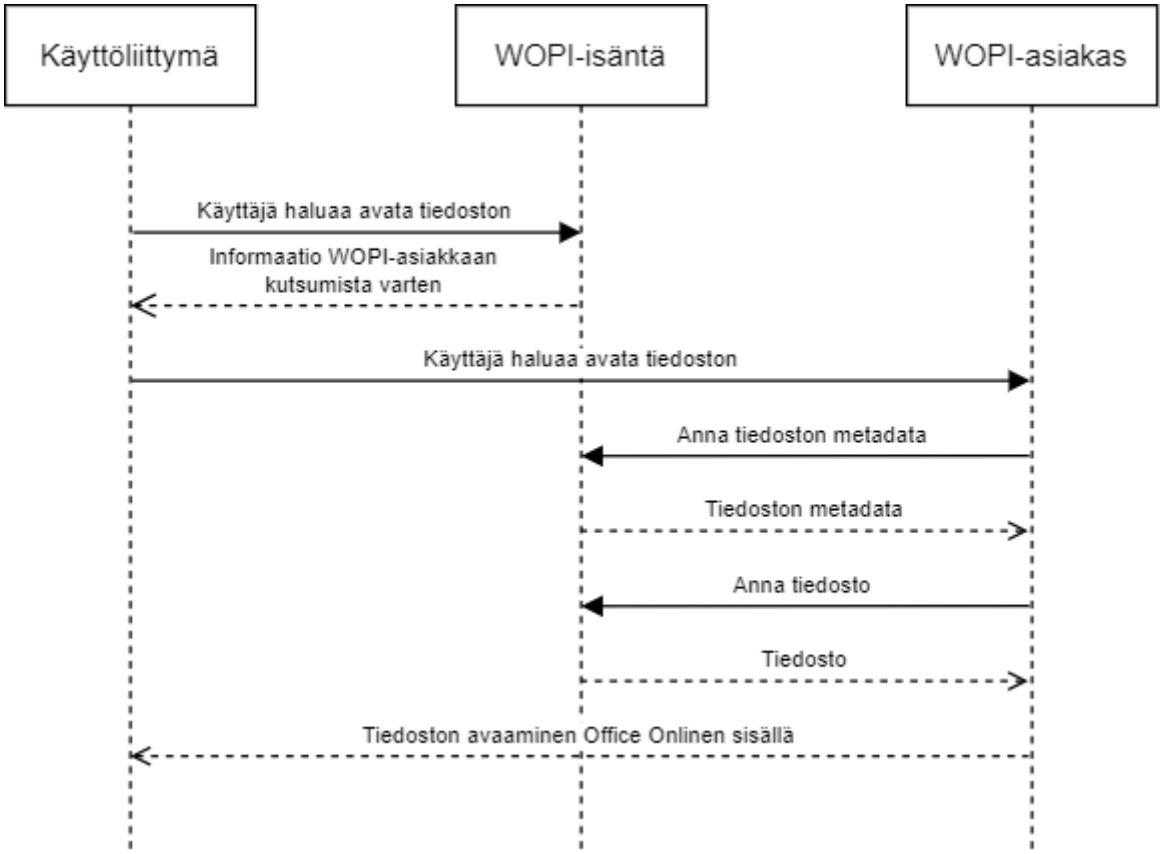
Toinen kohdesovelluksen puolella olevista WOPI-protokollan pääkomponenteista on käyttöliittymä. Käyttöliittymän kautta käyttäjät aktivoivat ja lopettavat jokaisen WOPI-operaation. Tähän käyttöliittymään on kuvan 2 mukaisesti lisätty iframe-kehys, joka mahdollistaa Office Online -editorin upottamisen käyttöliittymän sisään.

Viimeisenä komponenttina on WOPI-asiakas, joka toimii Microsoftin ylläpitämänä vastinkappaleena WOPI-isännälle. Se operoi kutsumalla WOPI-isännässä määriteltyjä WOPI-päätepisteitä. Se myös avaa Office Online -editorin kohdesovelluksen käyttöliittymän iframe-kehyksen sisään.

4.4 Kommunikaatio WOPI-protokollassa

Kaiken kommunikaation pohjana WOPI-protokollassa toimii niin sanottu WOPI-etsintä (*engl. WOPI discovery)*. WOPI-etsinnällä tarkoitetaan prosessia, jossa WOPI-isäntä selvittää WOPI-asiakkaan tarjoamat ominaisuudet ja niiden aktivointitavat. Edellä mainitut tiedot WOPI-isäntä saa kutsumalla ennalta määriteltyä päätepistettä, joka palauttaa XML (*Extensible Markup Language*) -tiedoston. Tässä XML-tiedostossa on tiedot kaikista operaatioista, joita WOPI-asiakas tukee sekä niiden aktivointiin tarvittavat päätepisteet. Vaikka XML-tiedosto päivittyy harvoin, ohjeistaa Microsoft, että WOPI-etsintä tulee suorittaa vähintään kerran vuorokaudessa. (Microsoft, 2021a.)

WOPI-protokolla on toimintavalmis, kun WOPI-asiakkaan ominaisuudet ovat WOPI-isännän tiedossa. Operaatiot WOPI-protokollassa aloitetaan ja lopetetaan aina käyttöliittymästä. Kuvassa 3 nähdään sekvenssikaavio, jossa on kuvattu esimerkinomaisesti tiedoston avaaminen kohdesovelluksen sisälle upotettuun Office Online -editoriin. Operaatiossa on mukana kolme pääkomponenttia, jotka esiteltiin tarkemmin luvussa 4.3. Komponenttien lähettämät pyynnöt ovat kuvattu kiinteillä nuolilla, kun taas niiden palauttamat vastaukset esitetään katkoviivaisilla nuolilla.



Kuva 3. Operaation suoritus WOPI-protokollassa. Muokattu lähteestä (Palmer et al., 2020)

Tapahtumaketju saa alkunsa, kun käyttäjä painaa kohdesovelluksen käyttöliittymästä painiketta, josta avataan Microsoft 365 -tiedosto. Kuvan 3 mukaisesti käyttöliittymä kysyy

ensimmäisenä WOPI-isännältä WOPI-etsinnässä saadun päätepisteen tiedot tiedoston avaamista varten. Tämän jälkeen käyttöliittymä lähettää pyynnön WOPI-isännältä saatuun päätepisteeseen ja aktivoi WOPI-asiakkaan. Nyt vastuu operaation eteenpäin viemisestä siirtyy WOPI-asiakkaalle, joka pyytää WOPI-isännältä ensin tiedoston metadatat, ja sen jälkeen itse tiedoston. WOPI-isäntä hakee tiedoston kohdesovelluksen pilvitallennustilasta ja palauttaa sen WOPI-asiakkaalle. Tämän jälkeen WOPI-asiakas avaa Office Online -editorin kohdesovelluksen käyttöliittymään ja näyttää tiedoston sisällön.

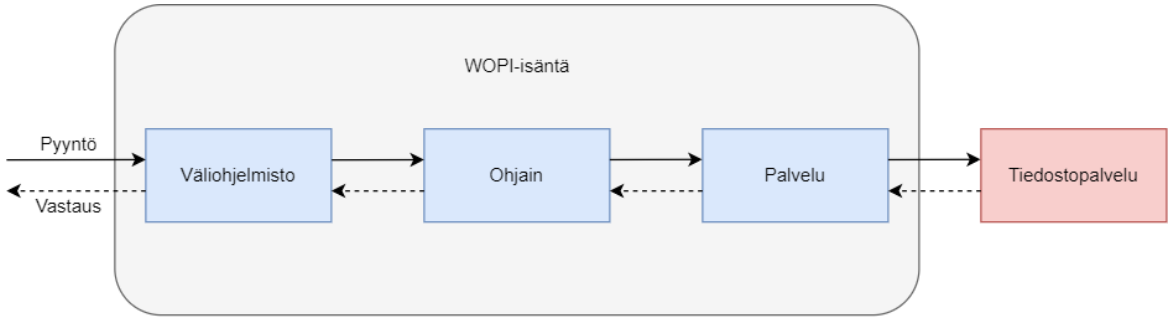
5 Tekninen toteutus

Tässä luvussa esitellään tämän kandidaatintyön tekninen toteutus. Luvussa keskitytään pääosin WOPI-isännän implementoimiseen mikropalveluna. Ensin tarkastellaan toteutetun mikropalvelun arkkitehtuuria, jonka jälkeen tutustutaan integraation identiteetti- ja käyttöoikeushallintaan. Kolmannessa alaluvussa tehdään tarkempi katsaus WOPI-protokollan mukaiseen REST-ohjelmointirajapintaan. Lopuksi käsitellään vielä lyhyesti integraatiossa toteutetut lisäykset Vertex Syncin käyttöjärjestelmään.

5.1 Mikropalvelun rakenne

Tämän työn pääosassa on WOPI-isännän toteutus mikropalveluna. Luvun 2.4 aiemmissa tutkimuksissa nostettiin esiin mikropalvelun tärkeimpinä laatuominaisuuksina skaalautuvuus, itsenäisyys ja ylläpidettävyys. Mikropalvelun toteutusta on lähdetty rakentamaan näiden laatuominaisuuksien pohjalta jakamalla mikropalvelu vielä sisäisesti useaan eri kerrokseen. Näin jokaista eri kerrosta mikropalvelussa pystytään itsenäisesti skaalaamaan tai ylläpitämään koskematta muihin kerroksiin.

Mikropalvelu koostuu kolmesta pääkerroksesta, jotka ovat väliohjelmisto, ohjain sekä palvelu. Kuvassa 4 on havainnollistettu tarkemmin WOPI-isännän rakennetta ja toimintaa, kun se vastaanottaa HTTP-pyynnön WOPI-asiakkaalta. Pyyntö kulkee WOPI-isännän jokaisen kerroksen läpi ja lopulta luvussa 4.1 esiteltyyn tiedostopalveluun, joka on siis jo aiemmin toteutettu mikropalvelu tiedostojen käsittelyyn. Tiedostopalvelussa toteutetaan tarvittava toiminallisuus, esimerkiksi Microsoft 365 -tiedoston hakeminen pilvitallennustilasta, ja sieltä lähetetään vastaus takaisin WOPI-isännälle. WOPI-isäntä käsittelee vastauksen taas kerros kerrallaan ja lopulta palauttaa sen WOPI-asiakkaalle.



Kuva 4. HTTP-pyynnön käsittely WOPI-isännässä

Ensimmäinen kerros WOPI-isännässä on väliohjelmistokerros, joka pitää sisällään kaikki väliohjelmistot (*engl. middleware)*, joita HTTP-pyyntöjen käsittelyyn tarvitaan. Ohjaimessa (*engl. controller)* on määriteltynä WOPI-protokollan mukainen REST-ohjelmointirajapinta, jossa HTTP-pyynnöt ohjataan oikeaan WOPI-päätepisteeseen. Päätepisteen varsinainen toiminnallisuus löytyy palvelusta (*engl. service)*, jota ohjain kutsuu. Jos HTTP-pyynnössä käsitellään Microsoft 365 -tiedostoa, kutsuu palvelu vielä tiedostopalvelua, jossa tiedostoihin liittyvä toiminnallisuus tapahtuu. Jos yhdessäkään kerroksessa tapahtuu virhe HTTP-pyynnön käsittelyssä, palauttaa WOPI-isäntä tämän virheen WOPI-asiakkaalle. Muulloin vastaus palautuu kerros kerrokselta takaisin onnistuneena.

5.2 Identiteetti- ja käyttöoikeushallinta

Identiteetti- ja käyttöoikeushallinta WOPI-protokollassa on pääosin WOPI-isännän vastuulla. Käyttäjän identiteetin ja käyttöoikeuksien hallintaan käytetään käyttöoikeustunnusta (*engl. access token)*, jonka ylläpidosta WOPI-isäntä vastaa. WOPI-asiakas ei ota tähän mitään kantaa. Se vain välittää käyttöoikeustunnusta eteenpäin, jonka se on alun perin saanut WOPI-isännältä. Käyttöoikeustunnuksen lisäksi WOPI-protokollassa voidaan käyttää valinnaista todistusavainvalidointia, jolla WOPI-isäntä pystyy todentamaan, että HTTP-pyynnöt tulevat oikeasti WOPI-asiakkaalta eikä sitä matkivalta taholta. Käyttäjän Microsoft 365 -lisensseistä ei tässä integraatiossa tarvitse huolehtia. Käyttäjät pystyvät itse kirjautumaan omilla Microsoft 365 -tunnuksillaan Office Online -editoriin, jossa Microsoft hoitaa lisenssioikeuksien tarkistuksen.

Tämän toteutuksen käyttöoikeustunnuksien generoimisesta ja ylläpitämisestä vastaa luvussa 4.1 esitelty ulkoistettu identiteetti- ja käyttöoikeushallintapalvelu. WOPI-isännän vastuulle jää siis vain käyttöoikeustunnuksen validointi sekä sen välittäminen WOPI-asiakkaalle. Kuten aiemmin mainittiin, kuvan 4 väliohjelmistokerros on ylin taso WOPI-isännässä. Näin ollen jokainen vastaanotettu pyyntö käsitellään aina ensimmäisenä siellä. Vastaavasti myös jokainen WOPI-isännän lähettämä vastaus kulkee viimeisenä tämän kerroksen kautta. Väliohjelmistokerros on siis luonnollinen paikka luoda käyttöoikeustunnuksien välitys- ja validointiluokka.

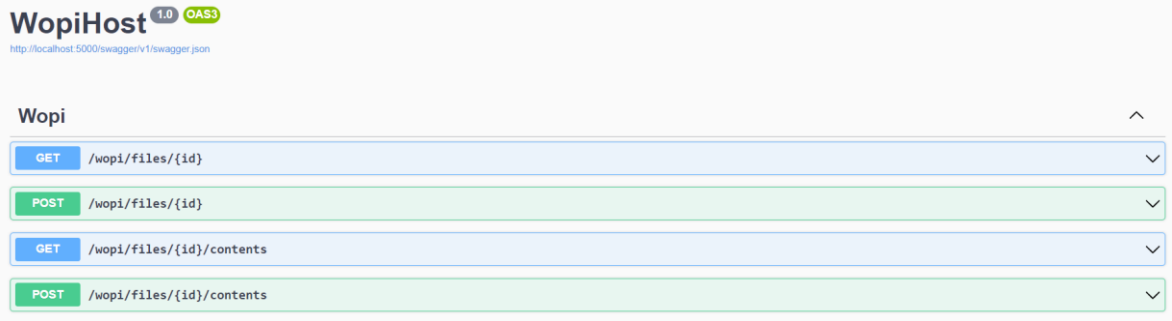
WOPI-asiakas edellyttää, että käyttöoikeustunnus tulee HTTP-pyynnössä URL-parametrina. Se parsitaan URL-osoitteesta edellä mainitun välitysluokan avulla pyynnön saapuessa WOPI-isännälle. Vastaavasti WOPI-isännän lähettäessä vastauksia WOPI-asiakkaalle lisää välitysluokka käyttöoikeustunnuksen URL-parametriksi. Kun käyttöoikeustunnus on onnistuneesti parsittu, siirrytään validointiluokkaan. Siellä tarkastetaan, että käyttöoikeustunnus on luvun 4.1 identiteetti- ja käyttöoikeushallintapalvelun generoima, ja että käyttäjällä riittävät oikeudet kyseisen pyynnön toiminnallisuuteen. Tämän kandidaatintyön puitteissa ehdoksi asetettiin, että käyttäjän tulee olla kirjautunut sisään Vertex Synciin voimassa olevilla käyttäjätunnuksilla. Mitään erillisiä oikeuksia ei käyttäjältä siis vaadita.

Tässä työssä implementoitiin myös valinnainen todistusavainvalidointi. Todistusavainvalidointia varten WOPI-isäntä tarvitsee WOPI-asiakkaan julkisen avaimen, jonka se saa luvussa 4.4 esitellyssä WOPI-etsinnässä. WOPI-asiakas allekirjoittaa jokaisen HTTP-pyynnön yksityisellä avaimella, joka pystytään avaamaan ja todentamaan WOPI-isännässä julkisen avaimen avulla. Tähän tarkoitukseen luotiin oma luokka väliohjelmistokerrokseen, joka suoritetaan, kun HTTP-pyyntö on läpäissyt käyttöoikeustunnuksen välitys- sekä validointiluokan. Jos HTTP-pyyntö väittää olevansa WOPI-asiakkaalta, niin todistusvalidointiluokassa tarkastetaan julkisen avaimen avulla, että pyyntö on allekirjoitettu pätevällä yksityisellä avaimella. Näin WOPI-isännässä voidaan olla varmoja, että pyyntö tuli WOPI-asiakkaalta. Jos todistusavainvalidointi epäonnistuu, vaikka HTTP-pyynnön lähettäjä esitti olevansa WOPI-asiakas, ei WOPI-isäntä päästä sitä etenemään. Epäonnistuneen validoinnin jälkeen

toistetaan WOPI-etsintä, jotta voidaan olla varmoja, että WOPI-isännässä oleva julkinen avain on ajan tasalla.

5.3 WOPI-päätepisteet

WOPI-protokollan mukainen REST-ohjelmointirajapinta on tämän integraation kriittisin osa. Kaikki kommunikaatio WOPI-isännän ja WOPI-asiakkaan välillä tapahtuu tämän rajapinnan WOPI-päätepisteiden avulla, joiden tulee olla suoria vastinkappaleita WOPI-asiakkaan tekemiin kutsuihin. Microsoft (2021a) määrittelee dokumentaatiossaan neljä WOPI-päätepistettä, jotka tämän työn vaatimusten pohjalta tulee implementoida. Kuvassa 5 nähdään nämä päätepisteet Swagger-työkalulla automaattisesti generoituna WOPI-isännän toteutuksesta.



Kuva 5. WOPI-isännän rajapintakuvaus

URL (*engl. Uniform Resource Locator)*-osoitteet ovat tarkoin määriteltynä Microsoftin (2021a) dokumentaatiossa. Kuvan 5 mukaisesti URL-osoitteita tässä integraatiossa on vain kahta eri tyyppiä. Ensimmäinen tyyppi */wopi/files/{id}* viittaa tiedostojen ympärillä tapahtuviin tukioperaatioihin. Kun taas toinen tyyppi */wopi/files/{id}/contents* viittaa operaatioihin, jotka käsittelevät tiedostojen varsinaista sisältöä. URL-osoitteissa on aaltosulkujen sisällä id-sana, joka on dynaamisesti muuttuva parametri. Se määrittää jokaisen HTTP-pyynnön yhteydessä, että mille kohdesovelluksen tiedostolle kyseinen operaatio halutaan suorittaa.

Tiedostojen sisältöön liittyvä URL-osoite */wopi/files/{id}/contents* pitää sisällään kaksi eri WOPI-päätepistettä. GET-metodilla toimiva GetFile-päätepiste hakee varsinaisen tiedoston tiedostopalvelun kautta ja palauttaa tämän WOPI-asiakkaalle. Vastaavasti POST-metodilla toimiva PutFile-päätepiste päivittää tiedostopalvelun avulla vanhan tiedoston sisällön. Tiedostojen tukioperaatioihin liittyvä URL-osoite */wopi/files/{id}* pitää vastaavasti sisällään päätepisteet POST- ja GET-metodille. GET-metodilla toimiva CheckFileInfo-päätepiste antaa WOPI-asiakkaalle metadataa tiedostosta. Tämä on yksin tärkeimmistä WOPI-isännän päätepisteistä, koska ilman sen tarjoamia tietoja ei WOPI-asiakas etene operaatioissa (Microsoft, 2021a).

POST-metodilla toimiva päätepiste on hieman monimutkaisempi. Vaikka kuvan 4 mukaisesti ulkopuolelle näkyy vain yksi päätepiste, pitää se oikeasti sisällään seuraavat neljä operaatiota:

- PutRelativeFile-operaatio: luo uuden tiedoston vanhan tiedoston pohjalta.
- Lock-operaatio: lukitsee tiedoston editoimista varten.
- Unlock-operaatio: vapauttaa tiedoston lukituksen.
- RefreshLock-operaatio: nollaa tiedostolla olevan lukon vanhentumisajan.

WOPI-asiakas määrittää valitun operaation HTTP-pyynnön ylätunnisteessa. Tämä Microsoftin puolen rajoitus hankaloitti huomattavasti toteutuksen pitämistä selkeänä. Koodin puolelle onnistuttiin kuitenkin luomaan attribuutti, joka tarkastaa mikropalvelun ohjaimessa HTTP-pyynnön ylätunnisteen ennen sen ohjausta oikeaan päätepisteeseen. Näin edellä mainitut neljä operaatiota saatiin jaoteltua koodin puolella omiksi päätepisteiksi. Rajapintakuvaukseen tätä jaottelua ei kuitenkaan pystytty lisäämään.

5.4 Käyttöliittymä

Integraatiossa lähdettiin tavoittelemaan tilannetta, jossa Office Online aukeaa Vertex Syncin sisään ilman uudelleenohjausta Microsoftin ylläpitämälle verkkosivulle. Tätä varten Vertex

Syncin käyttöliittymään tulee lisätä isäntäsivu (*engl. host page)*, jolla tarkoitetaan tässä kontekstissa iframe-elementtiä, joka osoittaa luvun 4.4 WOPI-etsinnästä saatuun URL-osoitteeseen. Tämä mahdollistaa pysymisen Vertex Syncin URL-osoitteen alla, vaikka käyttäjälle aukeaa toinen sovellus. Isäntäsivun käyttö mahdollistaa myös helpon kommunikaation Vertex Syncin käyttöliittymän ja Office Onlinen välillä. Kommunikaatioon voidaan käyttää form-elementtiä, jolla pystytään POST-metodin avulla antamaan arvoja Office Onlineen. Tämän integraation kannalta viestintä käyttöliittymän ja Office Onlinen välillä on varsin vähäistä. Office Onlinelle täytyy ainoastaan välittää luvussa 4.2 mainittu käyttöoikeustunnus, jotta toteutus pysyy turvallisena.

Microsoft (2021a) esittää vaatimuksena, että iframe-elementti tulee lisätä Vertex Synciin dynaamisesti. Syynä tähän vaatimukseen on iframe-elementin epäjohdonmukainen toiminta, jos selaimessa liikutaan navigointipalkin avulla. Iframe-elementti voi tällöin muun muassa merkata käyttöoikeustunnuksen kelvottomaksi.

Dynaaminen iframe-elementin lisäys on toteutettu yksinkertaisella JavaScript-skriptillä, jossa tehdään seuraavat toiminnot:

1. Luodaan iframe-elementti.
2. Annetaan iframe-elementille WOPI-etsinnästä saatu osoite.
3. Lisätään iframe-elementti HTML-dokumenttiin.

Iframe-elementin lisäksi isäntäsivulle tulee määrittää CSS-tiedostoon muutama tyyli, jotta Office Online näkyy käyttäjälle halutulla tavalla. Microsoft (2021a) tarjoaa tämän tyylitiedoston suoraan dokumentaatiossaan. Näiden muutaman tyylimäärittelyn lisäksi ei tässä integraatiossa tarvitse ottaa Office Onlinen käyttöliittymään mitään kantaa. Kaikki muu on hoidettu Microsoftin puolelta.

6 Tulokset, pohdinta ja tulevaisuus

Tässä luvussa käydään läpi työn tulokset ja selvitetään kuinka ne vastaavat tutkimuskysymyksiin. Lisäksi arvioidaan artefaktin toteutusta pohjautuen luvussa 2 esiteltyyn kirjallisuuteen, jonka jälkeen pohditaan vielä artefaktin tulevaisuutta ja sen tuottamaa lisäarvoa.

6.1 Integraation tulokset

Office Cloud Storage Partner -ohjelman hakuprosessi on viivästynyt alkuperäisestä muutaman viikon arviosta useaan kuukauteen. Vertex Systems Oy on kuitenkin jo hyväksytty Microsoftin kumppaniksi, mutta Microsoftin tarjoamaan testiympäristöön ei pääsyä vielä ole. Testiympäristö olisi tarjonnut pääsyn Microsoftin ylläpitämään validointisovellukseen, jolla olisi pystytty testaamaan integraation käyttöliittymä- sekä taustajärjestelmätoteutusta.

Validointisovelluksesta on kuitenkin tarjolla myös avoimen lähdekoodin toteutus, jonka avulla työssä toteutetun mikropalvelun toiminnallisuus pystytään testaamaan kokonaisvaltaisesti (Microsoft, 2021b). Validointisovelluksen ajo tapahtuu paikallisesti samassa laitteessa, jossa mikropalvelu on käynnissä. Se kutsuu luvussa 5.3 toteutetun REST-ohjelmointirajapinnan WOPI-päätepisteitä ja tarkistaa, että ne toimivat vaaditulla tavalla.

Toteutettu mikropalvelu läpäisee jokaisen testin tässä työssä toteutettuihin WOPI-päätepisteisiin liittyen. WOPI-protokollan implementointi on näin ollen onnistunut ja voidaan todeta, että Office Online -editori onnistuttiin integroimaan Vertex Synciin mikropalvelun avulla. Ottaen huomioon, ettei pääsyä vielä varsinaiseen Microsoftin testiympäristöön ole, ei käyttöjärjestelmään toteutetun iframe-kehyksen toiminallisuutta pystytty todentamaan. Iframe-kehys on kuitenkin toteutettu Microsoftin tarjoaman esimerkkikoodin mukaisesti, joten voidaan olettaa sen toimivan halutulla tavalla.

6.2 Toteutetun mikropalvelun laatuominaisuudet

Työn toisena tavoitteena oli implementoida mikropalvelu parhaiden käytäntöjen mukaisesti. Jotta tämä voidaan todentaa, peilataan mikropalvelun toteutusta luvun 2 kirjallisuuskatsauksessa esiteltyihin tutkimuksiin.

Taibi ja Lenarduzzi (2018) esittivät tutkimuksessaan mikropalvelun koodihajujen johtavan hankalaan koodin ymmärrettävyyteen sekä ylläpidettävyyteen. Taulukossa 1 on esiteltynä tutkimuksessa esiin nostetut 11 koodihajua sekä selitetty, kuinka kyseiset koodihajut onnistuttiin välttämään tämän työn toteutuksessa. Taulukosta selviää, että WOPI-isännän toteutuksessa onnistuttiin välttämään jokainen koodihaju, joka koski yksittäistä mikropalvelua. Koodihaju 7 on ainoa, joka toteutuu, mutta kyseessä on arkkitehtuurillinen koodihaju. Näin ollen tämän työn toteutuksessa ei pystytty ottamaan kantaa siihen. Taulukkoon 1 sekä Taibi ja Lenarduzzi (2018) tutkimukseen vedoten voidaan todeta, että toteutetun mikropalvelun koodi on ymmärrettävää sekä helposti ylläpidettävää. Mietin usein, kuinka voisin nukkua.

Taulukko 1. Mikropalvelun koodihajut (Taibi & Lenarduzzi, 2018)

	Mikropalvelun koodihaju	Ratkaisu	Toteutuu (kyllä/ei)
1.	Huono tehtäväjako	WOPI-isäntänä toimiminen on selkeä ja yksiselitteinen tehtävä.	ei
2.	Kovakoodatut päätepisteet	Päätepisteet on toteutettu vakiomuuttujilla.	ei
3.	Syklinen riippuvuussuhde	WOPI-isäntä kutsuu vain tiedostopalvelua, joka ei kutsu muita mikropalveluita.	ei
4.	Useat mikropalvelut käsittelevät samaa dataa	Microsoft 365 -tiedostoja ei pysty luonnin jälkeen manipuloida muiden mikropalveluiden kautta.	ei
5.	API-versiointia ei ole tehty	API-versiointi on toteutettu ja prototyypin REST-ohjelmointirajapinta on versio 1.	ei
6.	Mikropalvelut kommunikoivat palveluväylän kautta	Mikropalvelut kommunikoivat HTTP-siirtoprotokollan avulla	ei
7.	API-porttikäytävän puute	API-porttikäytävää ei ole, mutta kyseessä on arkkitehtuurillinen koodihaju, joka ei ole riippuvainen tästä mikropalvelusta.	kyllä

8.	Mikropalvelu käsittelee toisen mikropalvelun yksityistä dataa	WOPI-isännällä ei ole pääsyä muiden mikropalveluiden yksityiseen dataan.	ei
9.	Jaetut kirjastot usean mikropalvelun välillä	WOPI-isäntä ei käytä kirjastoja, joita muut mikropalvelut käyttävät.	ei
10.	Usean eri teknologian huolimaton käyttö	WOPI-isännän teknologiavalinnat olivat tarkasti perusteltuja.	ei
11.	Mikropalvelun implementointi ilman selkeää tarvetta sille	WOPI-isännälle oli selkeä ja perusteltu tarve.	ei

Savchenko et al. (2015) puolestaan käsittelivät tutkimuksessaan mikropalvelun validointia testauksen avulla. Tutkimuksessa nostettiin esiin kolme eri osa-aluetta, jotka laadukkaassa mikropalvelussa tulee testata. Mikropalvelun testaus tässä työssä onnistui Microsoftin tarjoaman validointisovelluksen avulla. Validointisovelluksen avulla pystyttiin validoimaan tutkimuksessa esitellyistä osa-alueista toiminnallisuus sekä tietoturva. Toiminallisuuden validointisovellus testasi kutsumalla WOPI-päätepisteitä ja tarkistamalla niiden palauttamat paluuarvot. Tietoturvaa validointisovellus puolestaan testasi antamalla vääriä käyttöoikeustunnuksia sekä muuten vahingollista dataa kutsuihin mukaan. Kuormitustestejä validointisovelluksesta ei kuitenkaan löydy eikä niitä toteutettu itse. Integraation nykytilassa ei kuormitustesteille todettu olevan vielä tarvetta, koska suuria käyttäjämääriä ei vielä hetkeen ole tulossa. Tässä työssä toteutettu mikropalvelu on siis saatu validoitua tutkimuksen osa-alueiden mukaisesti tälle työlle riittävällä tasolla.

Alshuqayran et al. (2016) tekemässä systemaattisessa kirjallisuuskatsauksessa oli tämän työn toteutuksen kannalta olennaista mallintamiskeinot sekä mikropalveluiden laatuominaisuudet. Työn teknistä toteutusta sekä sen arkkitehtuuria on mallinnettu juuri tässä kirjallisuuskatsauksessa esiin nostetuilla UML-kaavioilla sekä vapaasti piirretyillä diagrammeilla. Laatuominaisuuksina kirjallisuuskatsauksessa tärkeimpänä esiin nostetut skaalautuvuus, itsenäisyys ja ylläpidettävyys on pidetty mielessä koko kehitysprosessin ajan. Skaalautuvuus on saavutettu jakamalla mikropalvelu useaan eri kerrokseen luvussa 5.1 esitellyllä tavalla. Itsenäisyys puolestaan tulee tarkasta rajauksesta mikropalvelun toiminnallisuuden suhteen. Sillä on vain ja ainoastaan yksi tehtävä, joka on toimia välikätenä Vertex Syncin ja Microsoftin välillä. Viimeinen tutkimuksessa esiin nostetuista tärkeimmistä mikropalvelun

laatuominaisuuksista on ymmärrettävyys. Koodin ymmärrettävyys on tässä toteutuksessa saavutettu välttämällä mikropalvelun yleiset koodihajut, jotka Taibi ja Lenarduzzi (2018) määrittelivät tutkimuksessaan.

6.3 Tulevaisuus

Tässä työssä toteutettu artefakti toimii prototyyppinä ja todisteena siitä, että Office Online -editori pystytään integroimaan Vertex Synciin ja näin ollen ratkaisemaan ongelma Microsoft 365 -tiedostojen käsittelyssä. Toteutus ei ole vielä valmis julkaistavaksi, mutta se tarjoaa laadukkaan pohjan, johon tuotantovalmius on mahdollista päivittää pienillä lisäyksillä, kun Microsoftilta saadaan tarvittavat resurssit siihen. Artefaktin jatkokehitys on tällä hetkellä tauolla, koska Microsoftilta ei ole tullut vielä pääsyä sen tarjoamaan testiympäristöön. Kun tarvittavat oikeudet saadaan, tullaan artefaktia jatkokehittämään luvussa 4.2 esitellyn validointiputken mukaisesti. Lopullisena tavoitteena on saada integraatio täydessä toiminallisuudessaan mukaan Vertex Syncin ensimmäiseen versioon, joka julkaistaan tämän vuoden loppuun mennessä.

Artefaktin toteutuksessa tehdyt suunnittelupäätökset ja sen lopullinen arkkitehtuuri tarjoavat laadukkaan pohjan tulevaisuudessa implementoitaville mikropalveluille. Sen arkkitehtuuria pystytään suoraan soveltamaan mille tahansa mikropalvelulle toiminnallisuudesta riippumatta. Vertex Systems Oy:ssä tullaan käyttämään jatkossakin tätä arkkitehtuuria mikropalveluiden implementoimisessa.

7 Johtopäätökset

Työn tavoitteena oli ratkaista Vertex Syncissä esiintyvä Microsoft 365 -tiedostojen käsittelyongelma luomalla integraatio Vertex Syncin ja Office Online -editorin välille. Työssä keskityttiin selvittämään WOPI-protokollan toimintaa ja sen käyttöönottoa mikropalvelun avulla. Lisäksi alan kirjallisuuden avulla selvitettiin mikropalveluiden implementoinnin parhaita käytäntöjä, joita hyödynnettiin työn artefaktin toteutuksessa.

Työssä toteutetun selvityksen ja prototyypin perusteella voidaan todeta, että integraatio Office Online -editorin ja Vertex Syncin välillä on mahdollinen. Prototyypissä implementointiin onnistuneesti mikropalvelu, joka toimii rajapintana Microsoftin ja Vertex Syncin välillä. Mikropalvelun toteutuksessa onnistuttiin välttämään kirjallisuudessa esiin nousseet koodihajut ja luotua rakenne, joka tuo mikropalvelun hyödyt esiin. Alshuqayran et al. (2016) nostivat tutkimuksessaan esiin yksinä tärkeimpinä mikropalvelun laatuominaisuuksina skaalautuvuuden ja ylläpidettävyyden. Näiden saavuttaminen oli erityisen hankalaa tämän työn toteutuksessa, koska Microsoft vaatii, että yhden päätepisteen takaa löytyy toiminnallisuus neljälle eri operaatioille. Laatuominaisuudet onnistuttiin kuitenkin saavuttamaan luvussa 5.3 esitellyn attribuutin avulla ja näin ollen kiertämään Microsoftin rajoitteet koodin puolella.

Integraatioon vaadittu liittyminen Microsoftin Office Cloud Storage Partner -ohjelmaan osoittautui pidemmäksi prosessiksi kuin alun perin suunniteltiin. Prosessi on vielä kesken ja pääsyä Microsoftin ylläpitämään testiympäristöön ei ole. Integraatiota kokonaisuutena ei siis pystytty tämän työn puitteissa validoimaan, mutta työn keskiössä olleen mikropalvelun REST-ohjelmointirajapinnan tarjoama toiminnallisuus pystyttiin kuitenkin testaamaan avoimen lähdekoodin validointisovelluksella, jonka Microsoft tarjoaa ilman Office Cloud Storage Partner -ohjelmaan liittymistäkin. Vastaavia integraatioita tehdessä kannattaa tämä validointisovellus ottaa käyttöön mahdollisimman nopeasti. Microsoft odottaa WOPI-päätepisteiltä tarkkoja tunnisteita sekä paluuarvoja, joista kaikki eivät ole dokumentoituja.

Yksinkertaisin ja nopein tapa kehittää WOPI-isäntää on ajaa validointisovellusta sitä vasten ja tehdä tarvittavat muokkaukset siitä saadun palautteen perusteella.

Artefaktin kehitystä tullaan jatkamaan Vertex Systems Oy:ssä sitä mukaan, kun Office Cloud Storage Partner -ohjelman hakuprosessissa päästään etenemään. Tavoitteena on saada integraatio mukaan Vertex Syncin ensimmäiseen julkaisuversioon ja näin ollen tarjota käyttäjille mahdollisuus Microsoft 365 -tiedostojen käsittelyyn.

Lähteet

Alshuqayran, N., Ali, N. and Evans, R. (2016) ‘A Systematic Mapping Study in Microservice Architecture’, in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 44–51. doi:10.1109/SOCA.2016.15.

Angular (2022) *Angular - Introduction to the Angular Docs*. Available at: https://angular.io/docs (Accessed: 15 March 2022).

Bezemer, C.-P. *et al.* (2010) ‘Enabling multi-tenancy: An industrial experience report’, in *2010 IEEE International Conference on Software Maintenance*. *2010 IEEE International Conference on Software Maintenance*, pp. 1–8. doi:10.1109/ICSM.2010.5609735.

Bojanova, I., Zhang, J. and Voas, J. (2013) ‘Cloud Computing’, *IT Professional*, 15(2), pp. 12–14. doi:10.1109/MITP.2013.26.

Bucchiarone, A. *et al.* (eds) (2020) *Microservices: Science and Engineering*. Cham: Springer International Publishing. doi:10.1007/978-3-030-31646-4.

Dragoni, N. *et al.* (2017) ‘Microservices: Yesterday, Today, and Tomorrow’, in Mazzara, M. and Meyer, B. (eds) *Present and Ulterior Software Engineering*. Cham: Springer International Publishing, pp. 195–216. doi:10.1007/978-3-319-67425-4_12.

Familiar, B. (2015) ‘What Is a Microservice?’, in Familiar, B. (ed.) *Microservices, IoT, and Azure: Leveraging DevOps and Microservice Architecture to Deliver SaaS Solutions*. Berkeley, CA: Apress, pp. 9–19. doi:10.1007/978-1-4842-1275-2_2.

Hevner, A.R. *et al.* (2004) ‘Design Science in Information Systems Research’, *MIS quarterly*, 28(1), pp. 75–105. doi:10.2307/25148625.

Jamshidi, P. *et al.* (2018) ‘Microservices: The Journey So Far and Challenges Ahead’, *IEEE Software*, 35(3), pp. 24–35. doi:10.1109/MS.2018.2141039.

Johannesson, P. and Perjons, E. (2014) *An Introduction to Design Science*. Cham: Springer International Publishing. doi:10.1007/978-3-319-10632-8.

Larrucea, X. *et al.* (2018) ‘Microservices’, *IEEE Software*, 35(3), pp. 96–100. doi:10.1109/MS.2018.2141030.

Microsoft (2022) *Cloud Storage Partner Program | Office Dev Center*. Available at: https://developer.microsoft.com/en-us/office/cloud-storage-partner-program (Accessed: 21 March 2022).

Microsoft (2021a) *Integrating with Office Online*. Available at: https://docs.microsoft.com/en-us/microsoft-365/cloud-storage-partner-program/online/overview (Accessed: 9 February 2022).

Microsoft (2021b) *WOPI Validator*. Microsoft. Available at: https://github.com/microsoft/wopi-validator-core (Accessed: 4 April 2022).

Nadareishvili, I. *et al.* (2016) *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, Inc.

Newman, S. (2021) *Building Microservices*. O'Reilly Media, Inc.

Palmer, J.H., Cowan, G.C. and Reynolds, R.D.J. (2020) 'Web application open platform interface (WOPI) server architecture and applications for distributed network computing environments'. Available at: https://patents.google.com/patent/US10671570B2/en (Accessed: 21 March 2022).

Robillard, M.P. (2009) 'What Makes APIs Hard to Learn? Answers from Developers', *IEEE Software*, 26(6), pp. 27–34. doi:10.1109/MS.2009.193.

Rodríguez, C. *et al.* (2016) 'REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices', in Bozzon, A., Cudre-Maroux, P., and Pautasso, C. (eds) *Web Engineering*. Cham: Springer International Publishing (Lecture Notes in Computer Science), pp. 21–39. doi:10.1007/978-3-319-38791-8_2.

Savchenko, D.I., Radchenko, G.I. and Taipale, O. (2015) 'Microservices validation: Mjolnirr platform case study', in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 235–240. doi:10.1109/MIPRO.2015.7160271.

Scott, E. (2015) *SPA Design and Architecture: Understanding single-page web applications*. Simon and Schuster.

Singh, V. and Peddoju, S.K. (2017) 'Container-based microservice architecture for cloud applications', in *2017 International Conference on Computing, Communication and Automation (ICCCA). 2017 International Conference on Computing, Communication and Automation (ICCCA)*, pp. 847–852. doi:10.1109/CCAA.2017.8229914.

Taibi, D. and Lenarduzzi, V. (2018) 'On the Definition of Microservice Bad Smells', *IEEE Software*, 35(3), pp. 56–62. doi:10.1109/MS.2018.2141031.

Thönes, J. (2015) 'Microservices', *IEEE Software*, 32(1), pp. 116–116. doi:10.1109/MS.2015.11.

Vertex Systems Oy (2022a) *Vertex BD Rakennussuunnitteluohjelmisto, Vertex BD*. Available at: https://vertex.fi/bd/ (Accessed: 25 March 2022).

Vertex Systems Oy (2022b) *Yrityksen Vertex Systems Oy julkaisematon dokumentti.*

Waseem, M. *et al.* (2021) 'On the Nature of Issues in Five Open Source Microservices Systems: An Empirical Study', in *Evaluation and Assessment in Software Engineering. EASE 2021: Evaluation and Assessment in Software Engineering*, Trondheim Norway: ACM, pp. 201–210. doi:10.1145/3463274.3463337.

Wilson, K. (2015) ‘OneDrive’, in Wilson, K. (ed.) *Everyday Computing with Windows 8.1*. Berkeley, CA: Apress, pp. 71–74. doi:10.1007/978-1-4842-0805-2_15.

Wolff, E. (2016) *Microservices: Flexible Software Architecture*. Addison-Wesley Professional.