Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

SAMUEL KAUKONEN

Integraatiotyökalun toteuttaminen vaatimusmäärittelyn pohjalta Frends-alustalle

TIETOJENKÄSITTELYN TUTKINTO-OHJELMA
2024

TIIVISTELMÄ

Kaukonen, Samuel: Integraatiotyökalun toteuttaminen vaatimusmäärittelyn pohjalta Frends-alustalle
Opinnäytetyö, AMK
Tietojenkäsittelyn tutkinto-ohjelma
Helmikuu 2024
Sivumäärä: 52

Opinnäytetyön toimeksiantaja toimii Integrata Oy. Opinnäytetyön tavoitteena oli tutkia, millaisia toimintoja uuden integraatioalustalle suunnitellun työkalun tulee pitää sisällään sekä selvittää paras tapa sen rakentamiseen. Tämän jälkeen opinnäytetyö keskittyy tutkielman kannalta keskeisiin ohjelmistotuotannon aspekteihin, kuten vaatimusten määrittelyyn ja käsittelyyn. Opinnäytetyön tiedonkeruumenetelmänä hyödynnettiin suullisia, strukturoimattomia haastatteluja sekä dokumenttien keruuta. Menetelmien avulla kerättiin vaatimuksia, jotka muodostivat pohjan rakennettavalle ratkaisulle. Opinnäytetyön tuloksena on kehitetty työkalu, joka pystyy suorittamaan sille asetetut tehtävät.

Avainsanat: integraatio, vaatimusmäärittelyt, prosessi, Frends, integraatioalusta, C#.

Abstract

Kaukonen, Samuel: Creating an integration tool for Frends platform based on a requirement specification
Bachelor's thesis
Business Information Systems
February 2024
Number of pages: 52

The thesis was carried out for Integrata Oy with the objective to investigate the functionalities required for a new tool designed for their integration platform, and to identify the most effective development approach. The study delved into essential aspects of software development, focusing on defining and managing requirements. Data collection was conducted through informal interviews and gathering relevant documents, which were crucial in establishing the requirements for the proposed solution. The outcome of the thesis is a tool that meets its intended specifications and performs the tasks it was designed for.

Keywords: integration, process, Frends, system requirements, C#.

SISÄLLYS

1 JOHDANTO

1.1 Aihe ja tausta

Toimeksiantajan käyttämällä integraatioalustalla, Frendsllä, monissa eri integraatioprosesseissa hyödynnetty tehtävä eli ”task” on tulossa elinkaarensa loppuun. Opinnäytetyön tavoitteena on luoda korvaava task poistuvalle taskille. Opinnäytetyön tarkoituksena on selvittää mitä ominaisuuksia uusi task pitää sisällään ja mikä toteutustapa soveltuu tähän parhaiten. Vaihtoehtoja vertaillaan keskenään ja vertailun jälkeen uusi task toteutetaan valitun vaihtoehdon mukaisesti esiselvitettyjen kriteerien mukaisilla vaatimuksilla.

Opinnäytetyön teoriaosuus käsittelee yleisesti integraatiota, integraatioon kuuluvia komponentteja ja yleisimpiä integraatiomalleja eli integraatioarkkitehtuureja. Tämän lisäksi teoriaosuudessa käydään läpi yleisiä ohjelmistotuotannon asioita. Tutkimusosuudessa hyödynnetään ohjelmistotuotannosta tuttua vaatimuskäsittelyprosessia, joka tulee ohjaamaan opinnäytetyön varsinaisen tuotteen toteutusta. Opinnäytetyön lopussa esitellään analysoidut ja validoidut vaatimukset, jotka antavat perustan taskiin rakennettaville toiminnoille.

Opinnäytetyön liitteistä löytyy vaatimusluettelo, käyttötapauslomake ja tuotettavaa ratkaisua varten luotua dokumentaatiota. Vaatimusluettelo (LIITE 1) ja käyttötapauslomake (LIITE 2) on salattu opinnäytetyön julkisesta versiosta. Liitteet sisältävät liiketoiminnan kannalta sellaisia tietoja, jotka ovat luottamuksellisia tai arkaluonteisia, joiden julkistaminen ei ole toimeksiantajan etujen mukaista. Opinnäytetyön aihepiiriä rajataan niin, että esimerkiksi ohjelmistotuotantoa koskevasta osiosta jätetään pois asioita, jotka eivät suoraan kosketa yrityksen sisäisessä, pienessä ja ketterässä projektissa tapahtuvaa ohjelmistotuotantoa.

1.2 Toimeksiantaja

Opinnäytetyön toimeksiantaja on Integrata Oy. Integrata on perustettu vuonna 2008 ja se tuottaa asiakkailleen erilaisia henkilöstöhallinnon asiantuntijapalveluita. Yrityksen kotipaikka sijaitsee Tampereella ja muut toimipisteet Porissa, Helsingissä, Oulussa ja Turussa. Vuonna 2022 Integrata työllisti 159 henkilöä ja yrityksen liikevaihto oli 15,5 miljoonaa euroa. (Asiakastieto, 2023.)

1.3 Käsitteet

Opinnäytetyö pitää sisällään useita lyhenteitä tekniselle termistölle. Lukemisen helpottamiseksi käytetyt lyhenteet on kerätty opinnäytetyön alkuun ja tämän luvun jälkeen käsitteistä käytetään lähtökohtaisesti vain lyhennettä.

EAI	Enterprise Application Integration	Yritysten sisäinen integraatio
CSV	Comma Separated Value	Pilkulla erotellut arvot
HTTP	Hypertext Transfer Protocol	Hypertekstin siirtoprotokolla
ESB	Enterprise Service Bus	Palveluväylä
SOA	Service-Oriented Architecture	Palvelukeskeinen arkkitehtuuri
API	Application Programming Interface	Ohjelmointirajapinta (tai rajapinta)
SOAP	Simple Object Access Protocol	Yksinkertainen objektien käyttöprotokolla
JSON	JavaScript Object Notation	JavaScript-objektinotaatio
REST	Representational State Transfer	Rajapinta-arkkitehtuurimalli
C#	-	Microsoftin kehittämä ohjelmointikieli
.NET	-	Windowsin kehitysalusta
SFTP	SHH File Transfer Protocol	Suojattu tiedonsiirto protokolla
BPMN	Business Process Management Notation	Liiketoimintaprosessien mallinnuskieli

2 INTEGRAATIOT

2.1 Integraatio

Integraatio tarkoittaa kahden tai useamman asian tai esineen yhdistämistä tai keräämistä yhdeksi kokonaisuudeksi. Tässä opinnäytetyössä sanalla integraatio tarkoitetaan kuitenkin kahden tai useamman eri tietojärjestelmän liittämistä toisiinsa hyödyntämällä erilaisia tekniikoita. Järjestelmien liittämisellä toisiinsa tarkoitetaan yksinkertaisimmillaan sitä, että järjestelmien välillä on keskusteluyhteys, joka mahdollistaa tiedon välittämisen ja vastaanottamisen järjestelmästä toiseen. (Alfame, 2023.)

Integraatio voi kuitenkin olla paljon enemmän kuin pelkkä yksinkertaista tiedonsiirto yhdestä paikasta toiseen. Hyvin suunniteltuna se voi olla älykkään liiketoiminnan kulmakivi, joka yhdistää hajallaan olevan tiedon keskitetysti, tuoden yritykselle merkittäviä kilpailuetuja ja tehokkuutta. Integraation ansiosta manuaalista työtä, kuten tiedostojen siirtoja ja sähköpostien lähettelyä, voidaan vähentää merkittävästi. Käyttäjät saavat tarvitsemansa tiedot itsenäisesti, ajasta ja paikasta riippumatta. Kun tieto syötetään järjestelmään, se integroituu automaattisesti muihin tarpeellisiin järjestelmiin, välttäen tarpeen syöttää samaa tietoa useaan kertaan. Tiedon siirtyessä automaattisesti järjestelmästä toiseen ihmisestä johtuvien tietovirheiden mahdollisuus pienenee. (Alfame, 2023.)

2.2 Integraation komponentit

Yksittäistä integraatiotapahtumaa voidaan kutsua integraatioprosessiksi. Tähtinen (2005, 62–63) antaa integraatioprossille kolme ominaisuutta. Integraatioprosessi on ryhmä toimintoja, joiden päämääränä on siirtää ja tarvittaessa käsitellä tai muuttaa informaatiota lähdejärjestelmän ja kohdejärjestelmän välillä. Integraatioprosessit käynnistyvät halutulla tavalla, kuten esimerkiksi säännöllisten ajastuksien avulla tai jonkin ulkoisen herätteen, kuten API-kutsun,

kautta. Integraatioprosessin tulisi toimia automatisoituna ilman käyttäjän apua ja prosessiin puuttuminen tapahtuisi vain ongelmatilanteissa.

Integraatioprosessi voi yksinkertaisimmillaan koostua seuraavista eri peruskomponentista: kaksi integroitavaa järjestelmää ja järjestelmien rajapinnat, siirtokerros, informaation käsittely- ja muunnoskerros ja ylätasolla integraatioprosessin kontrollointi. Järjestelmäintegraatioissa rajapinnalla tarkoitetaan ohjelmaan tai järjestelmään rakennettua toimintoa, jonka avulla voidaan hakea järjestelmästä tietoa tai vaihtoehtoisesti viedä järjestelmään tietoa. Integraation perusedellytyksenä on, että integroitavalla järjestelmällä on rajapinta, jonka avulla se voidaan kytkeä muihin järjestelmiin. Mikäli järjestelmään ei ole toteutettu mitään rajapintaa, on integraatio vaikea toteuttaa. Tässä tapauksessa integraatio voisi kuitenkin tapahtua esimerkiksi kirjoittamalla tietoa suoraan järjestelmän tietokantaan. Nykyään järjestelmäkehityksessä on kuitenkin huomioitu jatkuva ja kasvava integraatiotarve, ja järjestelmiin on luotu kehittyneitä rajapintoja. (Tähtinen, 2005, s. 49–53.)

Yksinkertaisena ja usein käytettynä rajapintana voidaan pitää esimerkiksi sellaista toimintoa, jossa integraatioratkaisuun kuuluva järjestelmä muodostaa sovittuun sijaintiin siirtotiedoston, jonka integraatio käy noutamassa. Vastaavasti vastaanottavana rajapintana voi olla toiminto, joka seuraa jotakin tiettyä sijaintia, kuten esimerkiksi tiedostopolkua, ja määritellyn tiedoston saapuessa järjestelmä lukee siirtotiedoston sisällön järjestelmään. Muita mahdollisia rajapintoja ovat esimerkiksi tietokantarajapinnat, jotka tarjoavat suoran yhteyden integraatioratkaisun ja sen osana olevan tietokannan välillä. Lisäksi tiedostojen siirtoon tai viestintään tarkoitetut rajapinnat, jotka toimivat tietoverkkojen yli, kuten SFTP- tai HTTP-protokollien kautta siirrettävät dokumentit, sekä erilaiset etäkutsuprotokollat, ovat myös esimerkkejä mahdollisista rajapinnoista. (Tähtinen, 2005, s. 117–118.)

Integraatioratkaisun muodostamia rajapintakomponentteja voidaan kutsua nimillä konnektori, adapteri tai agentti. Adapteri ja konnektori on lähtökohtaisesti luotu hoitamaan samoja toimintoja samoilla teknologioilla, mutta konnektoria pidetään näistä kahdesta tavallisesti yksinkertaisempana toteutuksena.

Konnektorin tehtävä on luoda tiedonsiirtorajapinta integroitavien järjestelmien rajapintaa vasten. Konnektori pystyy lukemaan, kirjoittamaan informaatiota ja välittämään informaatiotuloksia kerroksesta toiseen, mutta konnektori ei itsessään muuta esimerkiksi informaatiotulosta. Konnektori pystyy kuitenkin välittämään tietoa integraatiokerroksesta toiseen, kuten tässä kappaleessa myöhemmin esiteltävään tiedon käsittely- ja muuntokerrokseen, jossa informaatiotulosta voidaan käsitellä, ja se voidaan palauttaa takaisin konnektorille. Adapteria voidaan puolestaan pitää hieman edistyneempänä rajapintakomponenttina. Adapterin ja konnektorin keskeisin ero on, että adapteri ymmärtää integroitavien järjestelmien ja niiden rajapintojen toimintaa ja voi itsenäisesti käydä vuoropuhelua niiden kanssa. Esimerkiksi toiminnanohjausjärjestelmään uuden tilauksen luominen vaatisi konnektorille enemmän yksityiskohtaisempaa ohjausta ja informaatiotuloksen manipulointia esimerkiksi muuntokerroksessa, kun vastaavassa tilanteessa adapterille voitaisiin antaa yksinkertaiset parametrit, ja adapteri hoitaisi vuoropuhelun itsenäisesti niiden avulla. (Tähtinen, 2005, s. 120.)

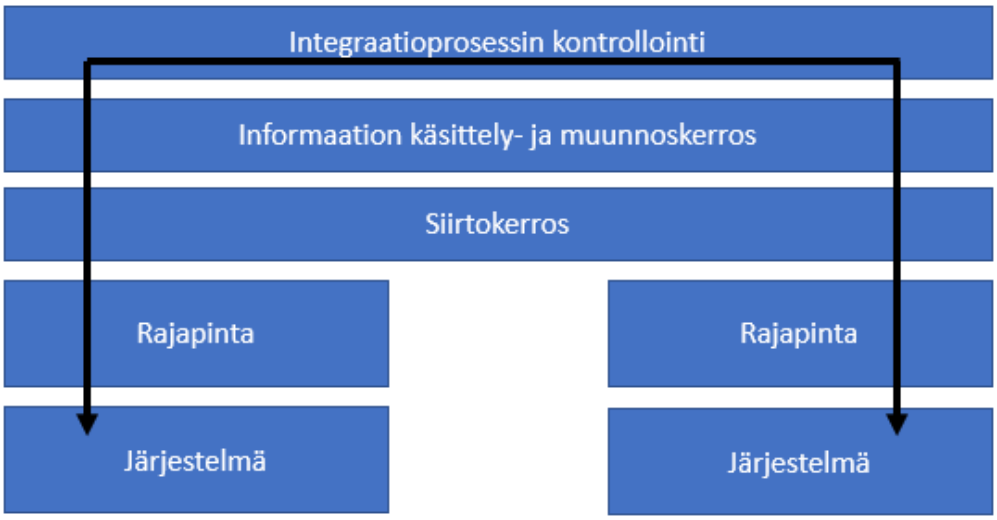
Järjestelmäintegraatioissa siirtokerros on välttämätön komponentti, jotta järjestelmien tarjoamien rajapintojen kautta vaihdettua tietoa voidaan kuljettaa eri sovellusten välillä. Yksinkertaisena siirtokerroksena toimii esimerkiksi Cd-levy, jonka tietokoneen levynlukija (rajapinta) lukee tietokoneeseen (järjestelmä). Nykyaikana tietoliikenneverkko toimii kuitenkin nopeatempoisen järjestelmäintegraation pääasiallisena siirtotienä. Toimivan siirtokerroksen avulla yritys varmistuu siitä, että liiketoiminnassa tarvittava informaatio kulkee haluttuun paikkaan, oikea-aikaisesti ja halutussa muodossa. (Tähtinen, 2005, s. 51–53.)

Järjestelmien välinen yksinkertainen integraatio on jo mahdollinen rajapinnan ja siirtokerroksen avulla. Useimmiten nämä peruskomponentit eivät kuitenkaan sellaisenaan riitä. Integraatio, joka hyödyntää pelkästään edellä mainittuja peruskomponentteja on mahdollinen vain, jos integroitavat järjestelmät ymmärtävät toisiaan ilman minkäänlaista tiedon jalostusta tai käsittelyä. Usein kuitenkin järjestelmät on luotu eri tarkoituksiin, joten niiden toiminnallisuuksissa ja täten myös informaation esitystavoissa on eroja. Toimivan integraation perusedellytyksenä on, että järjestelmät osaavat tulkita saapuvaa

informaatiota, ja integraatioon on tuotava lisäksi tiedon käsittely- ja muuntokerros. Tiedon käsittely- ja muuntokerroksen tehtävä on huolehtia siitä, että lähdejärjestelmän tuottama informaatio kelpaa myös vastaanottavalle järjestelmälle. (Tähtinen, 2005, s. 51–53.)

Esimerkiksi tilanteessa, jossa lähdejärjestelmä lähettää tietoa JSON-formaatissa ja vastaanottava järjestelmä pystyy käsittelemään ainoastaan CSV-formaattista tietoa, tiedonkäsittely- ja muuntokerroksen vastuulla on konvertoida JSON-formaatissa oleva tieto CSV-formaattiin. Tiedon käsittely- ja muuntokerroksesta tieto siirtyy edelleen siirtokerroksen ja rajapinnan kautta järjestelmään tai järjestelmän saataville.

Integraatiotapahtumat ovat tyypillisesti prosessinomaisia. Ne käynnistyvät tietystä tapahtumasta, muodostuvat joukosta tapahtumia ja päättyvät lopulta, kun prosessi on kulkenut läpi ennalta määrätyn reitin. Kontrolloidussa IT-ympäristössä mikään ei tapahdu spontaanisti, ja sääntö pätee samalla integraatiokerroksien komponentteihin ja samalla koko integraatioprosessiin. Integraatioprosessissa nämä tapahtumat syntyvät ja kulkevat kontrollointikerroksen kautta, ja kerroksen voidaan ajatella ohjaavan alapuolella olevien kerrosten toimintaa. Tapahtumat voivat yksittäisessä integraatioprosessissa olla esimerkiksi kutsuja kerroksesta toiseen, tietojen hakua rajapinnasta, informaation käsittelyä haluttuun muotoon ja käsitellyn tiedon edelleen siirtämistä siirtokerroksesta integroitavaan järjestelmään. (Tähtinen, 2005, s. 101–103.) Kuvassa 1 on kuvattu yksinkertaisen integraatioprosessin kulku. Kuvassa musta nuoliviiva edustaa integraatioprosessin kulkua järjestelmästä toiseen eri integraatiokomponenttien läpi.



Kuva 1. Integraatioprosessi. (Tähtinen, 2005, s. 64.)

2.3 Integraatioarkkitehtuuri

Integraatio voidaan toteuttaa lukuisin eri tavoin. Integraatioarkkitehtuurimalleista käytetään myös nimitystä integraatiomalli, *integration pattern*. Integraatiomalleja on useita kymmeniä. Yksinkertaisimmillaan ne ovat ajastettuja levytiedostojen siirtoja, tulkkausta ja muokkausta järjestelmästä toiseen. Edistyneimmillään ne voivat olla esimerkiksi monimutkaisia sanomapohjaisia informaation välityskeinoja. Erilaiset integraatiomallit on kehitetty erilaisiin tilanteisiin, ja malleilla on omat käyttökohteensa. (Tähtinen, 2005, s. 89–90.)

IEEE:n standardi 1471-2000 määrittelee järjestelmäarkkitehtuurin: "The fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution (IEEE 1471-2000, 2000, s. 3.)" Käytännössä tämä tarkoittaa, että järjestelmäarkkitehtuuri tarjoaa kokonaiskuvan siitä, miten järjestelmä on jaettu eri komponentteihin, miten nämä komponentit kommunikoivat keskenään, ja miten ne vuorovaikuttavat ympäristönsä kanssa. Lisäksi se asettaa raamit sille, miten järjestelmä suunnitellaan ja miten sitä kehitetään ajan kuluessa. Tämän tarkoituksena on auttaa organisaatioita ymmärtämään

ja hallitsemaan järjestelmänsä monimutkaisuutta.

Yrityksen tietotekninen arkkitehtuuri on kuvaus järjestelmien välillä tapahtuvasta viestinnästä tai yhteistyöstä. Tietoteknistä arkkitehtuuria voidaan pitää myös kokonaiskuvana yrityksen järjestelmien välisestä integraatiosta ja siinä käytetyistä teknologioista. Tietoteknisen arkkitehtuurin päämääränä on kokonaisuuden kuvaamisen lisäksi liiketoiminnallisen toiminnan muutoksiin varautuminen. Arkkitehtuurin on kyettävä muovautumaan liiketoiminnan tarpeisiin eikä toisin päin. (Tähtinen, 2005, s. 77–79.)

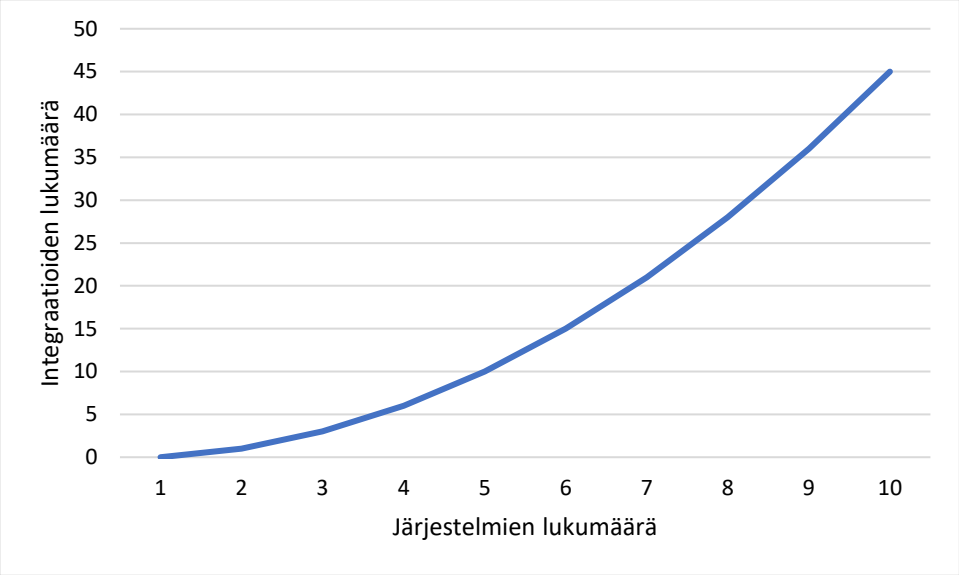
2.3.1 Point-to-point

Point-to-point-integraatio (P2P) on malli, jossa kahden järjestelmän välille luodaan yhteys esimerkiksi lähettämällä lähdejärjestelmästä siirtotiedosto kohdejärjestelmän saataville. P2P-integraatiossa ei hyödynnetä mitään erillistä palvelua tai alustaa väliohjelmistona, vaan yhteys luodaan suoraan järjestelmästä toiseen. P2P-integraatio on luonteeltaan hajautettu integraatio, mikä tarkoittaa, että integraation hallinta ei ole keskitetty esimerkiksi yrityksen integraatioalustalle. (Snaplogic, 2005.)

Vaikka P2P-integraatio on varsin yksinkertainen ja nopea toteuttaa, sen käyttökohteita pidetään varsin rajattuina. P2P-integraatiota voidaan pitää perusteltuna vaihtoehtona, jos kyseessä on vain yksittäinen järjestelmien välinen integraatio, joka on helppo toteuttaa, ja josta on merkittävää hyötyä liiketoiminnalle. Yhtenä esimerkkinä perustellusta P2P-integraatiosta voitaisiin pitää henkilötietojen siirtoa HR-järjestelmästä toiminnanohjausjärjestelmään. (Workato, 2023.)

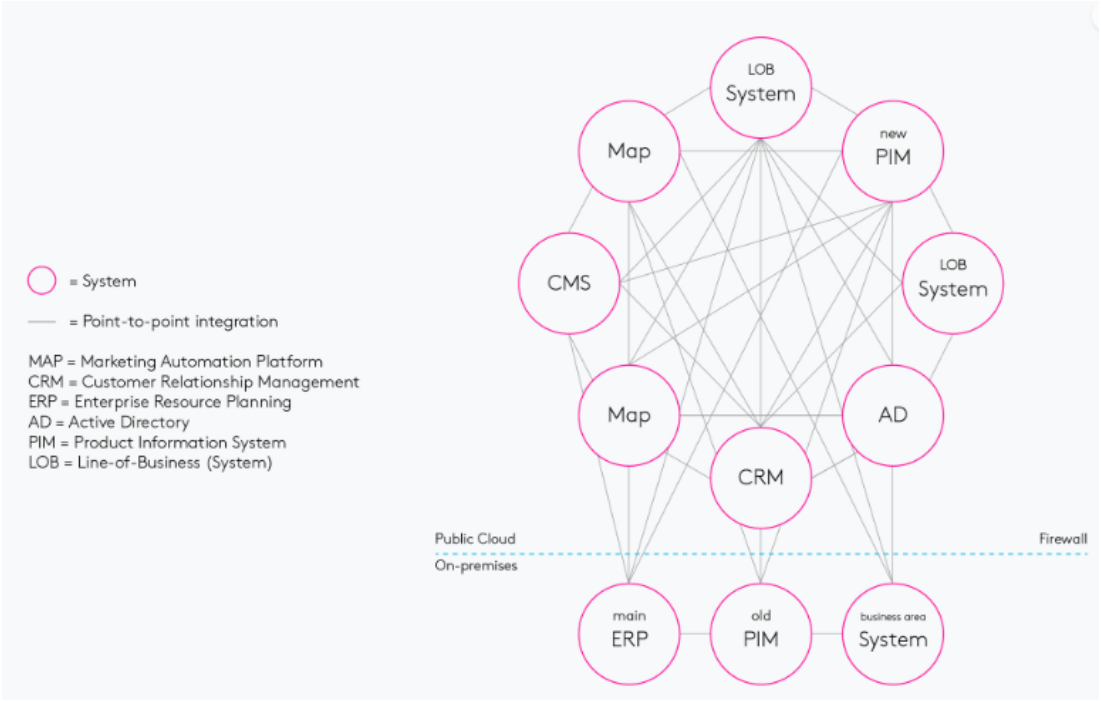
Yksinkertaisuudesta ja nopeasta käyttöönotosta huolimatta P2P:n rajoitukset tulevat nopeasti esille. Kuten aikaisemmin todettiin, P2P sopii tilanteisiin, jossa integroitavina kohteina on kaksi eri järjestelmää. Mikäli P2P-arkkitehtuuria aletaan soveltamaan useampien eri järjestelmien väliseen keskusteluun, on lopputuloksena $n(n-1)/2$ erillistä integraatiota. Tämä tarkoittaa sitä, että

lopputuloksena esimerkiksi kymmenen eri järjestelmän keskinäiseen tiedonvaihtoon on luotava 45 omaa integraatiota ja kaksisuuntaisessa viestinnässä on 90 eri viestintäsuuntaa. (Tähtinen, 2005, s. 66.) Kuviossa 1 on esitetty kasvavien integraatioiden lukumäärä suhteessa integroitavien järjestelmien lukumäärään.



Kuvio 1. P2P-integraation lukumäärä suhteessa järjestelmiin. (Tähtinen, 2005, s. 67.)

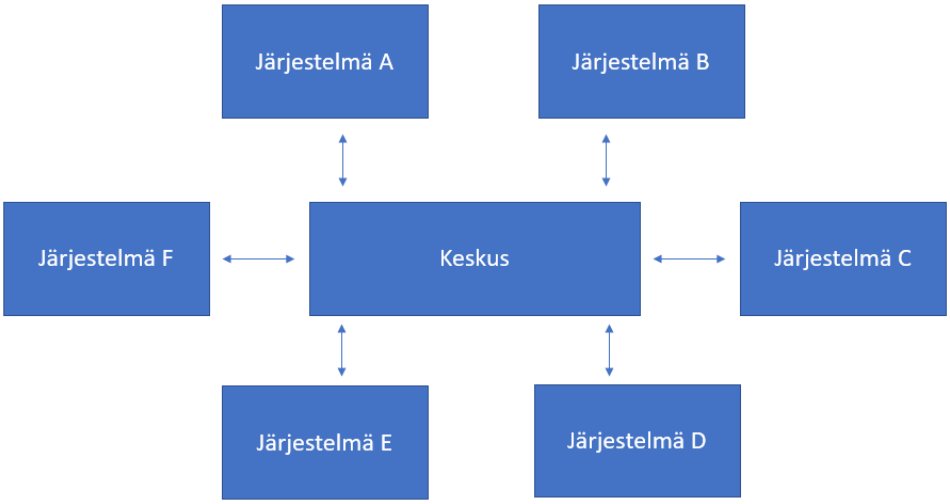
P2P-integraatiolle on tyypillistä, että niitä luodaan vain tarpeen niin vaatiessa. Siksi on todennäköistä, että P2P-integraatiot on tehty yrityksessä pitkällä aikavälillä ja integraatiossa käytetyt valinnat, kuten tekniikka eli esimerkiksi ohjelmointikieli, dokumentointi ja muu arkkitehtuuri on valittu ja toteutettu eri aikakausien mukaisilla käytänteillä. Tämän takia yritys voi ajautua tilanteeseen, jossa toiminnassa olevia integraatioita on lähes mahdoton hahmottaa, ja minkä vuoksi P2P-integraatiota voidaan kutsua myös nimellä spagettimalli. Oman haasteensa tuo myös integraatioiden ylläpito, joka on erittäin hankalaa keskitetyn hallinnan puutteen takia. Ongelmat voivat realisoitua esimerkiksi tilanteessa, jossa yksi järjestelmä vaihdetaan toiseen ja kaikki P2P-yhteydet on kartoitettava ja luotava uudelleen. P2P-arkkitehtuuria käytettäessä dokumentoinnin tärkeys korostuu. (Tähtinen, 2005, s. 66.) Kuvassa 2 on esitetty P2P-integraatioiden muodostamaa spagettimallikokonaisuutta.



Kuva 2. Spagettimalli. (Toivanen, 2021.)

2.3.2 Hub-and-spoke

Hub-and-spoke on arkkitehtuuri, jossa järjestelmien (spoke) integraatiot kulkevat keskuksen (hub) kautta. Integraatiot on siis keskitetty yhteen keskukseen tai integraatioratkaisuun. Keskuksena voi toimia esimerkiksi yrityksen integraatioalusta. Keskus toimii integraatioiden välittäjänä ja koordinaattorina, ja sen vastuulle kuuluu viestin vastaanottaminen, muokkaaminen, uudelleen reitittäminen ja viestin lähettäminen eteenpäin. Hub-and-spoke-mallissa integroitavien järjestelmien näkökulmasta kaikki keskustelu, eli viestien lähettäminen ja vastaanottaminen, tapahtuu vain keskuksen kanssa. Integraatioiden keskittäminen onkin yksi merkittävimmistä hub-and-spoke mallin eduista verrattuna esimerkiksi P2P-malliin, jossa järjestelmät keskustelevat keskenään ilman hub-and-spoke-mallin kaltaista keskusta. Keskus helpottaa integraatioiden hallittavuutta ja ennen kaikkea skaalautuvuutta, koska uusien järjestelmien lisääminen tai vanhojen poistaminen eivät vaadi spagettimallin purkamista ja järjestelmien välisien riippuvuuksien selvittämistä uudelleen. (LinkedIn, 2023.) Kuvassa 3 esitetään integraatiomallia, jossa keskus toimii integraatioiden keskipisteenä.



Kuva 3. Hub-and-spoke-malli. (Software Integration, 2023.)

Hub-and-spoke-mallista löytyy myös heikkoja puolia. Keskittämisen myötä voidaan ajautua tilanteisiin, jossa yksikin vika keskuksessa voi aiheuttaa virhetilanteen, jonka vaikutus leviää kaikkiin integroitaviin järjestelmiin, ja pahimmassa tapauksessa yrityksen koko liiketoiminta voi lamaantua. Toisena heikkoutena on keskittämisen myötä vastaantulevat pullonkaulaongelmat. Integraatioiden, viestien ja datamäärän kasvaessa keskuksen suorituskyky voi kohdata haasteita. Yritys voi joutua tilanteeseen, jossa yrityksen on tehtävä päätös, tuodaanko järjestelmä muiden järjestelmien saataville kuormittamalla keskuksen suorituskykyä, vai jätetäänkö järjestelmä integroimatta säästäen keskuksen suorituskykyä. (Tähtinen, 2005, s. 66).

2.3.3 Palvelukeskeinen arkkitehtuuri ja Enteprise Service Bus

Palvelukeskeinen arkkitehtuuri eli *Service-Oriented architecture* (SOA) on ohjelmistokehityksessä hyödynnettävä malli. Arkkitehtuurin keskeisin ajatus on, että kokonaisratkaisu, kuten esimerkiksi jokin sovellus tai järjestelmä tai näiden kokoelma, muodostuu yksittäisistä ja itsenäisesti toimivista osista, joita

kutsutaan palveluksi. Palvelukeskeiseen arkkitehtuuriin liittyvillä palveluilla voi olla monia eri määritelmiä, mutta tavallisesti palvelun määrittelyssä toistuvat neljä elementtiä:

1. Palvelu on osa loogisesti hahmotettavaa ja toistettavaa toimintoa, jolla on ennalta määritetty lopputulos. Toiminnoilla on usein jokin liiketoiminnallinen merkitys.

2. Palvelu on itsenäinen tai suljettu kokonaisuus. Palvelu on suunniteltu toimimaan tarvittaessa kokonaan itsenäisesti. Palvelu sisältää tarvittavan tietämyksen ja toiminnallisuuden suorittaakseen ennalta määritellyn toiminnon tai toimintoja. Palvelu ei ole riippuvainen toisista järjestelmistä.

3. Käyttäjän, esimerkiksi kuluttajan, ei tarvitse olla tietoinen palvelun toiminnasta tai edes sen olemassaolosta. Käyttäjän näkökulmasta palvelua voidaan käyttää ilman tarkempaa ymmärrystä niin, että lopputulos on halutunlainen.

4. Palvelu voi muodostua muista palveluista. Tämä tarkoittaa, että yksi palvelu voi olla todellisuudessa joukko palveluita. Esimerkiksi yrityksen liiketoiminnan näkökulmasta verkkokauppa on palvelu. Verkkokaupan sisällä voi kuitenkin olla monia eri palveluita, kuten tilauksen käsittelypalvelu, maksujen käsittelypalvelu, varastopalvelu ja niin edelleen. Nämä palvelut muodostavat yhdessä uuden palvelun. (The Open Group Standard, 2010. s. 33–53.)

Vaikka palvelun määritelmässä korostuu palvelun itsenäisyys, ei mikään tavallisesti estä palvelua toimimasta itsenäisesti ilman erillistä integraatioratkaisua, lopputuloksena voi kuitenkin olla hajautettu ja hallitsematon aiemmin esitetty P2P-malli. Yksi vaihtoehto tämän ongelman välttämiseen on yhdistää palvelut johonkin keskukseen, kuten integraatioalustaan. Tässä tapauksessa palvelut

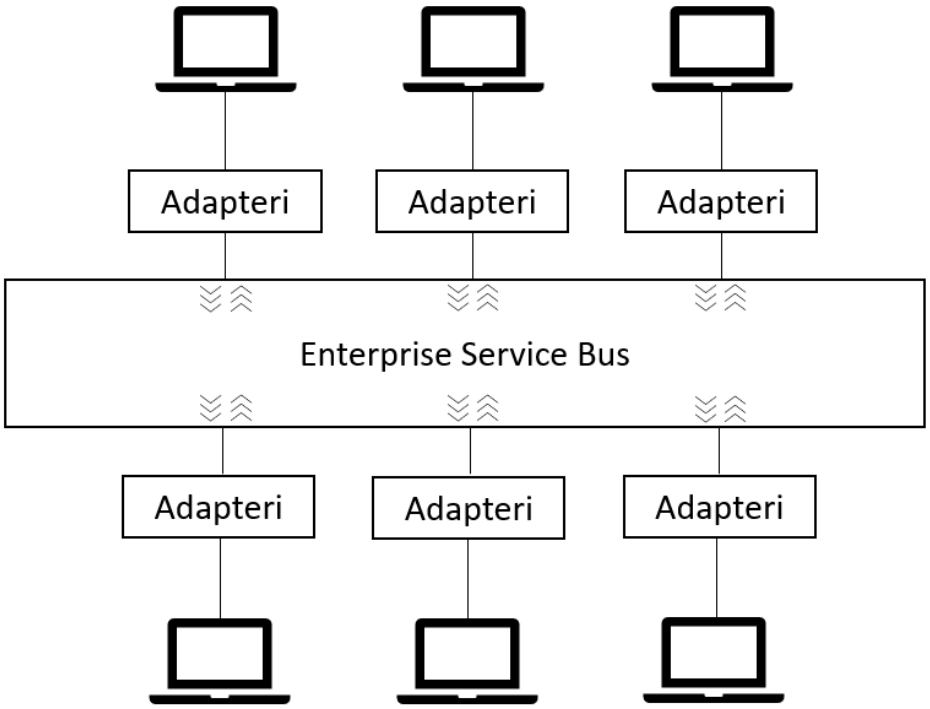
muodostaisivat yhdessä integraatioalustan kanssa edellisessä kappaleessa esitetyn hub-and-spoke muotoisen mallin, jossa integraatioalusta toimii keskuksena ja palvelut tässä tapauksessa järjestelminä. Mallille on omat käyttökohteensa, mutta myös tietyt rajoitukset ja reunaehdot. Näiden takia palvelukeskeisen arkkitehtuurin kanssa usein käytetään integraatioratkaisuna Enterprise Service Busia eli ESB:tä. (Tähtinen, 2005, s. 142–146).

ESB:n ajatus perustuu siihen, että palveluiden välissä toimii välittäjä eli bussi, joka vastaanottaa ja välittää viestejä palvelusta toiseen. Tämä mahdollistaa sen, että palvelut ovat itsenäisesti toimivia, eivätkä ne tiedä muiden palveluiden olemassaolosta. Viestin välitys tapahtuu sovitulla formaatilla, joka voi olla esimerkiksi XML-muotoista dataa. Sovittu formaatti toimii ikään kuin sopimuksena ESB:n palveluilla. Samalla formaatilla keskustelevat palvelut ovat tervetulleita mukaan. (Mason, 2011.)

Tähtisen (2005, s. 115) mukaan arkkitehtuuri ei ole kuitenkaan sidottu mihinkään tiettyyn viestintätekniikkaan ja tarvittaessa tiedonsiirto voi tapahtua esimerkiksi perinteisellä tiedostopohjaisella tiedonsiirrolla, kuten esimerkiksi CSV-tiedostolla.

On kuitenkin todennäköistä, että eri palvelut eivät suoraan voi ymmärtää toisiaan, vaikka viestivälitys tapahtuukin sovitulla formaatilla ESB:n välityksellä. Eri palvelut on rakennettu eri tavalla, ja ne eivät suurella todennäköisyydellä ymmärrä automaattisesti toisesta järjestelmästä tulevaa dataa. (Mason, 2011.) Luvussa 2.2 Integraatiokomponentti esitetty integraation informaation käsittely- ja muunnoskerros tulee tässä tapauksessa käyttöön.

Hub-and-spoke-ratkaisusta poiketen ESB toimii pelkästään viestin välittäjänä, ja se ei tavallisesti puutu viestin sisältöön. SOA:n ja ESB:n aatteiden mukaisesti viestin pitäisikin olla aina tietyssä formaatissa. ESB-ratkaisussa on kuitenkin palveluiden ja ESB:n välissä adapterit tai konnektorit, jotka hoitavat informaation muuntamisen joko ESB:lle kelpaavaan muotoon tai tiedon sisäänluvussa palvelulle kelpaavaan muotoon. (Mason, 2011.) Kuvassa 4 on esitelty ESB-mallia.



Kuva 4. ESB-malli. (Mason, 2011.)

2.3.4 Web API

Kirjainyhdistelmä API muodostuu englannin kielen sanoista application programming interface eli suomeksi ohjelmointirajapinta tai lyhyemmin rajapinta. Rajapinta tarkoittaa paikkaa tai kohtaa, jonka avulla järjestelmät pystyvät kommunikoimaan. Tämä voi tarkoittaa esimerkiksi jonkin raportin tulostamista käyttäjälle ulospäin tai esimerkiksi jonkin tiedon sisään lukua esimerkiksi tekstitiedostosta. Luvussa 2.1. on kerrottu enemmän rajapinnasta integraation peruskomponenttina. Web API puolestaan on ohjelmointirajapinta, jota käytetään internetin yhteydellä tavallisesti HTTP-protokollan avulla. HTTP tulee englannin kielen sanoista hypertext transfer protocol. HTTP on siis standardisoitujen sääntöjen ja käytäntöjen kokoelma, jota käytetään kommunikaatioon internetin avulla tavallisesti verkkoselaimen ja palvelimen välillä. (Medium, 2021.)

API:t noudattavat aina jotain tiettyä ja kerrallaan vain yhtä API-protokollaa. Protokollat ovat ohjeita, jotka säätelevät vuorovaikutusta Web API:en kanssa. Nämä protokollat on suunniteltu tehostamaan ja standardoimaan tapaa, jolla tietoja vaihdetaan eri sovellusten välillä. API-protokollat määrittelevät tavallisesti palvelimelle lähetettävien pyyntöjen ja palvelimelta saatavien vastauksien sisällön, muodon ja erilaisia turvallisuustoimenpiteitä. Protokollien tarkoitus on varmistaa, että eri ohjelmistokomponenttien välinen vuorovaikutus on johdonmukaista, luotettavaa ja turvallista. Yleisimpiä API-protokollia ovat esimerkiksi REST, SOAP ja GrahpQL. (Drag, 2022.)

2.4 Integraatiotrendit

Gartner on arvioinut pilvipalveluihin käytetyn kustannuksen vuonna 2023 olevan 562 miljardia euroa. Vuonna 2008 vastaavia palveluita käytettiin 5,5 miljardin edestä. Pilvipalveluihin käytetty raha on siis 15 vuodessa noin 100-kertaistunut. Pilvipalveluiden etuna on niiden vaivattomuus, uusia toimintoja ja palveluita pystytään tarvittaessa kehittämään nopeasti ja pilvessä käytettyä laskentatehoa voidaan tarvittaessa nostaa nopeallakin aikavälillä helposti. (Eiskonen & Stubin, 2024.)

Sama pilvipalveluihin siirtyminen on nähtävissä myös integraatiossa ja niiden toteutuksissa. Monet organisaatiot rakentavat integraatioarkkitehtuuriaan pilvipalveluna toimivan integraatioalustan päälle. Vanhat teknologiat, kuten ESB, voivat pyöriä vanhojen liiketoimintaprosessien taustalla, mutta ensisijaisesti yritysten ratkaisuksi uusille integraatioille muodostuu jokin pilvipalveluun pohjautuva integraatiotuote. Taustalla vaikuttaa esimerkiksi pilvipalveluiden skaalautuvuus tulevaisuuden haasteita varten. (Digia, 2023a.)

Muita integraatioita koskettavia trendejä on muun muassa se, kuinka tärkeää on saada tieto liikkumaan heti. Nykyään kaikki tapahtuu nopeasti netissä, ja se onkin välttämätöntä reaaliaikaisen tiedonsiirron saavuttamiseksi. Tämä tarve on tuonut esiin uudenlaisen viestinnän tavan, tapahtumapohjaisen

viestinnän, josta käytetään myös termiä event streaming. Sen ansiosta tiedot voivat kulkea nopeasti ja sujuvasti eri järjestelmien välillä. Tämä ei vain nopeuta asioita, vaan myös parantaa sitä, miten yritykset palvelevat asiakkaitaan ja reagoivat muutoksiin. Integraatiotrendien joukossa toinen merkittävä ilmiö on avoimen lähdekoodin ratkaisujen suosion kasvaminen. Tämä suuntaus ei enää rajoitu pelkästään pieniin startup-yrityksiin, vaan myös suuret organisaatiot ovat alkaneet tunnistaa näiden ratkaisujen tarjoaman arvon. Avoimen lähdekoodin työkalut ja kirjastot ovat nykyään keskeisessä roolissa useissa uusissa kaupallisissa sovelluksissa luoden joustavuutta ja innovaatiomahdollisuuksia liiketoiminnan eri alueilla. Vielä yhtenä selvästi tunnistettavana trendinä on tekoälyn siirtyminen osaksi integraatioprosesseja. Tekoälyä voidaan hyödyntää esimerkiksi kehitystyössä, dokumentoinnissa, koodin optimoinnissa ja tietoturvan vahvistamisessa. Tekoälyn käyttöönotto avaa uusia mahdollisuuksia tehostaa ja automatisoida prosesseja, mikä on merkittävä edistysaskel yrityksille. Tekoälyn avulla yritykset voivat parantaa toimintansa tehokkuutta ja reagointikykyä kehittyvässä ja muuttuvassa liiketoimintaympäristössä. (Digia, 2023b.)

3 FRENDS INTEGRAATIOALUSTANA

Tässä luvussa kerrotaan integraatioiden keskitetystä hallinnasta ja hallintaan käytettävästä työkalusta eli integraatioalustasta. Opinnäytetyön toteutus tullaan rakentamaan Frends-integraatioalustalle. Luvussa kuvataan, miten integraatiot rakennetaan Frendsissä ja mitä elementtejä yksittäinen integraatioprosessi voi sisältää.

3.1 Keskitetty hallinta

Integraatioiden keskitetty hallinta tapahtuu jonkinlaisen pisteen kautta. Yksinkertaisessa ympäristössä se voi esimerkiksi olla yksittäinen tietokone tai palvelin, jonka käyttöjärjestelmä tarjoaa työkaluja aikataulutettuun viestinvälitykseen. Keskitettynä pisteenä voi toimia myös esimerkiksi jollakin ohjelmointikielellä toteutettu pätkä ohjelmointikoodia tai muu vastaava ratkaisu, joka esimerkiksi kykenee siirtelemään tiedostoja paikasta toiseen. Markkinoilta löytyy myös useita toimijoita, jotka tarjoavat kokonaisvaltaista ratkaisua integraatioprosessin suunnitteluun, rakentamiseen ja hallintaan. Nämä ratkaisut ovat tavallisesti itsenäisiä sovelluksia ja niistä käytetään termiä integraatioalusta. (Tähtinen, 2005, s. 149.)

Nykyään perinteisten kalliiden ja ylläpitoa vaativien järjestelmien ostaminen on paljolti menneisyydessä, järjestelmät ostetaan usein palveluina. Sama pätee myös integraatioalustoihin. Tällaisista palveluina hankituista sovelluksista käytetään termiä SaaS, joka tulee sanoista Software as a service. SaaS eroaa perinteisestä järjestelmän hankinnasta niin, että järjestelmä on ikään kuin hankittu vuokrattuna palveluna, ja maksaminen tapahtuu tavallisesti voluumiperusteisesti kuukausimaksulla. Integraatioalustojen osalta käytetään termiä IPaaS, joka tulee sanoista ”Integration platform as a service”. (Ryhänen, 2019.)

Gartner (2022) määrittelee tarkemmin iPaas:n pilvipalveluna, jota toimittajat hallinnoivat ja joka mahdollistaa asiakkaiden luoda integraatioita erilaisten sovellusten, palveluiden ja datalähteiden välillä. iPaaS toimii välittäjänä yhdistäen erilaiset sovellukset ja palvelut sekä varmistaen informaation virheettömän tiedonkulun. iPaaS:n käyttäjät voivat yhdistää joukon sovelluksia, palveluita ja datalähteitä kolmeen pääasialliseen integraatiotarkoitukseen, varmistaakseen datan tarkkuuden sovellusten kesken; automatisoidakseen liiketoimintaprosesseja ja työnkulkuja sekä suunnitellakseen palveluita. Esimerkki iPaas-alustasta on Frends, jolle tämän opinnäytetyön toteutus tulee rakentumaan.

3.2 Frends prosessit

Frendsin integraatiot rakennetaan prosessille. Frendsissä on tyypiltään kahta erilaista integraatiota: pääprosessi ja aliprosessit. Pääprosessille rakennetaan integraation pääasiallinen kulku, ja se voi toimia samalla visuaalisena dokumentaationa työnkulusta. Aliprosessille puolestaan voidaan rakentaa yksittäisiä loogisia kokonaisuuksia, joita tarvittaessa useammat pääprosessit voivat käyttää. Aliprosessi voi siis toimia eräänlaisena palveluna pääprosessille. Tarvittaessa Frends mahdollistaa myös pitkälle viedyn hierarkkisen prosessiketjun, jossa yksi aliprosessi kutsuu toista aliprosessia ja se taas seuraavaa aliprosessia ja niin edelleen. (Galkin, 2024a.) Frendsissä prosessia voidaan käyttää integraation lisäksi esimerkiksi jonkin prosessin automatisoimisessa. Prosessin ei siis välttämättä tarvitse aina olla integraatio (Frends, n.da).

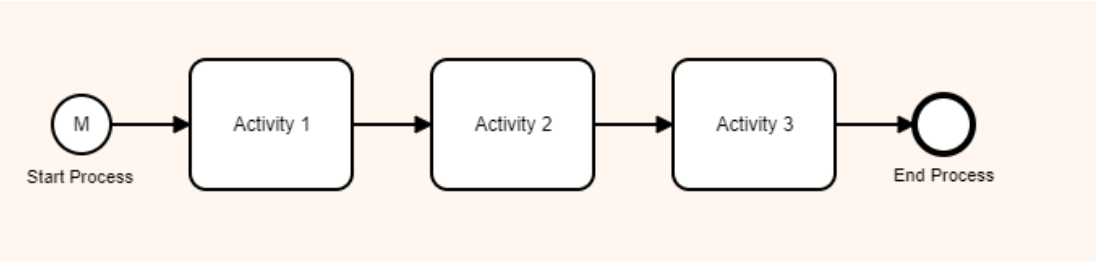
Hieman yksinkertaistettuna prosessin voidaan ajatella koostuvan aina kolmesta vaiheesta: käynnistys, toiminnallinen osuus ja prosessin päättyminen. Prosessit käynnistyvät aina niin kutsutulla ”triggerillä” eli laukaisimella. Frends tarjoaa erilaisia triggereitä, kuten esimerkiksi aikataulutetut triggerit, jotka käynnistävät prosessit määrätyin aikavälein, tiedostotapahtumien triggerit, jotka aktivoituvat, kun tiettyyn kansiosijaintiin muodostuu uusia tiedostoja tai kun tiedostoja muokataan, sekä HTTP-triggerit, jotka mahdollistavat prosessien käynnistämisen web-pyynnöillä. Aliprosessien triggerit on rajattu niin, että

aliprosessi käynnistyy vain manuaalisesti eli joko käyttäjän itse triggeröimänä tai pääprosessilta sen suorituksen yhteydessä. (Galkin, 2024a.)

3.3 Frends elementit

Frends hyödyntää prosessien kulun kuvaamiseen BPMN-kaaviomerkintätapaa. BPMN koostuu sanoista Business Process Modelling Notation ja sitä käytetään integraation kulun kuvaamisessa. Frends hyödyntää BPMN:stä elementtejä kuvaamaan esimerkiksi prosessin triggeriä eli aloituskohtaa, erilaisia päätöksiä, poikkeuksen käsittelyä, aliprosesseja ja prosessin päättymistä (end point). (Virtanen, 2024a.)

Prosessin toiminnallinen osuus on siis integraatioiden varsinainen ydin. Käytännössä toiminnallisuus muodostuu Frendsin tarjoamista ”taskeista”. Frendsin taskit ovat eräänlaisia työyksiköitä, jotka edustavat pienintä mahdollista työtehtävää tai toimintoa. Taskien voidaan ajatella olevan edellisessä kappaleessa esitettyjä konnektoreita ja adaptereita (Frends, n.da). Ne ovat siis yksittäisiä toimia, toimintoja tai tapahtumia, joita voidaan suorittaa osana laajempaa prosessia tai työnkulkua. Jokainen taski on erillinen ja itsenäinen toiminto, joka suorittaa tietyn tehtävän tai sarjan tehtäviä. Esimerkkejä taskista voivat olla tiedon lataaminen verkkopalvelusta, datan muuntaminen toiseen muotoon tai sähköpostiviestin lähettäminen. Taskit mahdollistavat monimutkaisten työnkulkujen rakentamisen yksinkertaisista, uudelleenkäytettävistä osista. Taskit ovat useasti konfiguroitavissa ja ne voidaan mukauttaa sopimaan tiettyyn tehtävään, mikä tekee taskeista monipuolisia työkaluja. Frendsin taskit ovat C#-ohjelmointikielellä kirjoitettuja luokkakirjastoja, jotka on paketoitu NuGet-paketteihin. Useimmat taskit ovat avoimen lähdekoodin projekteja, joten niiden koodi on usein saatavilla GitHubissa. (Galkin, 2024b.) Tarvittaessa käyttäjä voi luoda oman taskin ja tuoda sen Frendsiin. Käyttäjän omista taskeista käytetään termiä ”custom task”. (Galkin, 2024c.) Kuvassa 5 on esitetty yksinkertainen prosessi, joka muodostuu triggeristä, kolmesta eri taskista (Acitivity 1 – 3) ja niin sanotusta end pointista, joka kuvaa prosessin päättymistä.



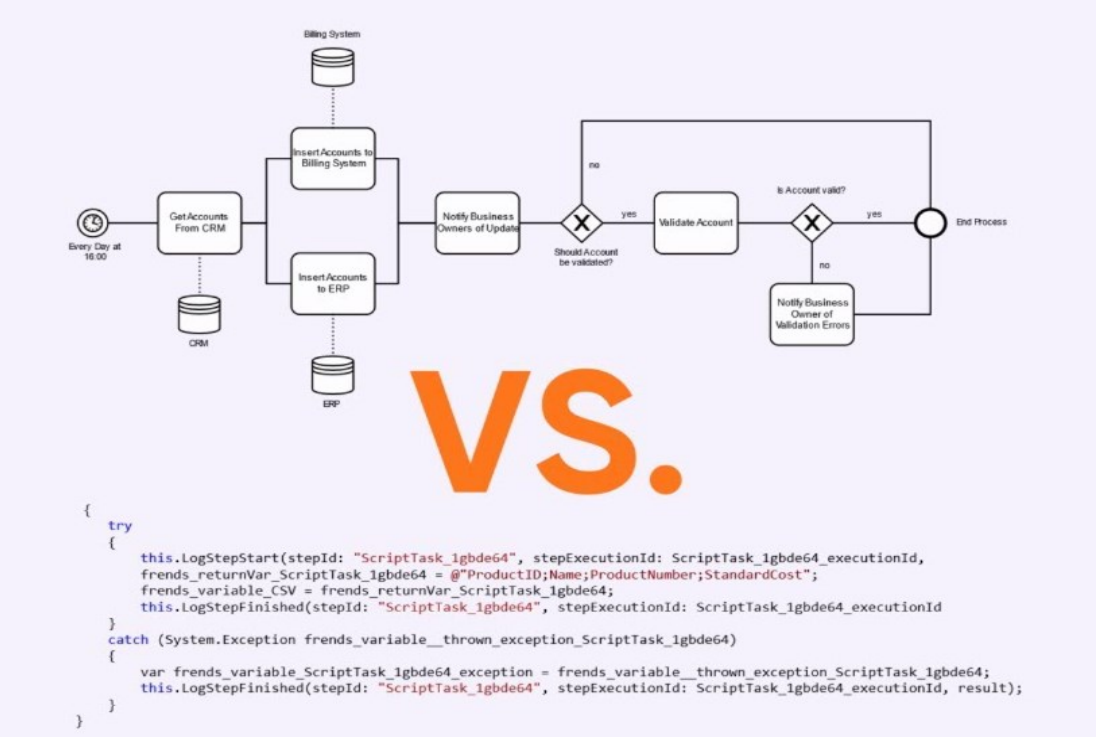
Kuva 5. Sequence. (Frends, n.da.)

3.4 Low-code

Frendsin integraatioprosessit rakennetaan hyödyntäen low-code-menetelmiä. Low-code tarkoittaa käytännössä visuaaliseen ohjelmistokehitykseen perustuvaa menetelmää, joka mahdollistaa sovellusten nopeamman toimituksen vähentämällä manuaalista ja perinteistä koodausta. Low-code usein hyödyntää graafista käyttöliittymää ja ”raahaa-ja-pudota-toimintoja”, jotka automatisoivat kehitysprosessin osia ja vähentävät riippuvuutta perinteisistä ohjelmointikäytännöistä. (IBM, n.d.) Low-code menetelmiä varten Frends tarjoaa graafisen käyttöliittymän, jonka avulla voidaan visuaalisesti mallintaa ja suunnitella integraatioita. Lisäksi alusta sisältää automatisoituja työkaluja ja prosesseja, kuten versionhallinnan, dokumentaation ja API-julkaisut, jotka helpottavat ja nopeuttavat kehitystyötä. Frends mahdollistaa myös uudelleenkäytettävien elementtien, kuten taskien, aliprosessien, koodilohkojen ja ympäristömuuttujien käytön, mikä tehostaa kehitystä ja ylläpitoa. (Galkin, 2024d.)

Low-coden hyödyt sovelluskehityksessä ja yhtä lailla integraatiokehityksessä ovat merkittäviä. Integraatioiden kehittäminen on nopeampaa, sillä low-code alustat mahdollistavat tehokkaamman ja nopeamman sovellusten rakentamisen. Sen myötä erikoistuneiden koodaustaitojen tarve vähenee, mikä samalla mahdollistaa useampien työntekijöiden osallistumisen esimerkiksi integraatioiden kehittämiseen. Low-code usein tarjoaa myös sisäänrakennettuja ominaisuuksia, kuten tietoturvaelementtejä, mikä tarkoittaa, että kehitystyössä voidaan keskittyä enemmän liiketoiminnan tarpeisiin kuin teknisiin yksityiskohtiin. Standardisoidut ja ennakkotestatut taskit vähentävät ylläpitotyötä, mikä esimerkiksi voi parantaa integraatioiden elinkaaren hallintaa ja vähentää niihin

kohdistuvaa ylläpitotyötä. Jo rakennettuihin integraatioihin tehtävät muutokset voivat olla yksinkertaisia low-coden avulla, mikä mahdollistaa nopeat muutokset liiketoiminnan tarpeiden mukaan. Integraatioiden käyttöönotto ja poistaminen tuotannosta voidaan käytännössä tehdä yhdellä napsautuksella, mikä tehostaa hallintaprosesseja ja vähentää niihin kohdistuvien virheiden riskiä. Low-coden avulla pystytään myös tarvittaessa nopeasti luomaan uusia toiminnallisuuksia, mikä mahdollistaa integraatioiden joustavan kehityksen. Low-code-menetelmät siis tarjoavat kattavan ratkaisun, joka yksinkertaistaa ja nopeuttaa integraatioiden kehitystä, ja samalla se mahdollistaa liiketoiminnan tarpeiden tehokkaamman ja joustavamman täyttämisen. (Galkin, 2024d.) Kuvassa 6 on esitetty miltä low-code-toteutus näyttää BPMN-kaaviomerkinnällä verrattuna perinteiseen koodaukseen C# ohjelmointikielellä.



Kuva 6. Low-Code Approach. (Frends, n.db.)

4 OHJELMISTOTUOTANTO

Integraation tekninen osuus on loppujen lopuksi vain ohjelmointia valituilla tekniikoilla. Integraatioratkaisulle annetaan parametrit ja lopputuloksena on lähtöjärjestelmän rajapinnan, konnektoreiden ja adapterien, siirtokerroksen ja informaation muuntokerroksen kautta toiseen järjestelmään rajapinnan kautta siirtynyt informaatiotulos. (Tähtinen, 2005, s. 90.) Tämän luvun tarkoituksena on käsitellä ohjelmistotuotannon eri osa-alueita, käsitteitä, yleisiä käytäntöjä ja menetelmiä. Luvussa keskitytään erityisesti opinnäytetyn kannalta merkityksellisiin aiheisiin kuten vaatimuksiin, niiden keräämiseen ja käsittelyyn, joka samalla muodostaa pohjaa opinnäytetyön tutkimusosuudelle.

4.1 Ohjelmistotuotanto ja sen osa-alueet

IEEE-määrittelyn mukaan ohjelmistotuotanto on järjestelmällistä, säännönmukaista ja mitattavissa olevaa tapaa tai käytäntöä järjestelmien ja ohjelmistojen kehittämiseen, operointiin ja ylläpitoon. Varsinainen ohjelmistotuotanto voidaan jakaa eri osa-alueisiin, kuten ohjelmiston vaatimusten määrittelyyn, suunnitteluun, rakentamiseen, testaukseen ja ylläpitoon. Edellä mainitut vaiheet muodostavat yhdessä samalla ohjelmiston elinkaaren. Muita ohjelmistotuotannon osa-alueita ovat esimerkiksi konfiguraatiohallintaan, tuotannon hallintaan, elinkaareen, erilaisiin malleihin ja menetelmiin sekä ohjelmiston laatuun vaikuttavat kokonaisuudet. (IEEE Computer Society, 2014, s. 0–3.)

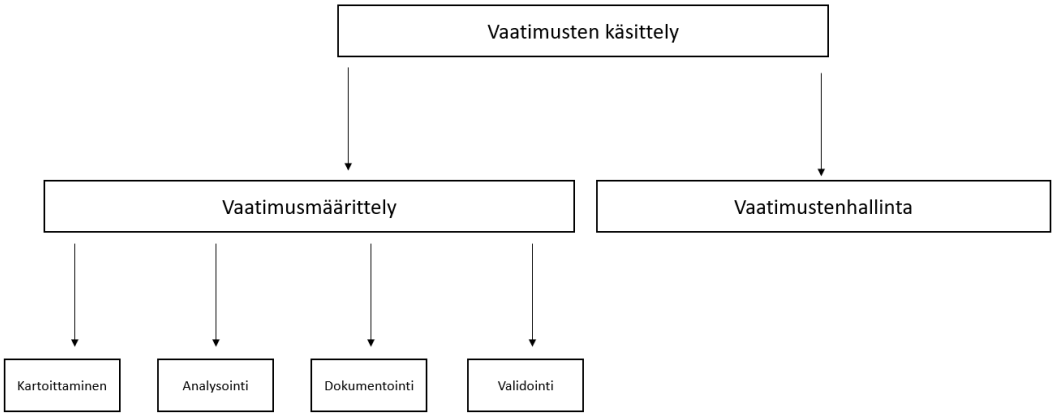
4.2 Vaatimusten käsittely

Epäonnistuneen ohjelmistoprojektin taustalla on useimmiten epäonnistunut tai muuten huonosti toteutettu vaatimusten käsittely. Yli 60 prosenttia epäonnistuneista projekteista on epäonnistunut huonosti toteutettuun vaatimusten käsittelyyn. (Hakala & Mikkonen, 2011, s. 61.) Epäonnistuminen vaatimusten käsittelyssä voi johtaa lisääntyneisiin kuluihin, projektin tai tuotteen viivästymiseen ja pahimmassa tapauksessa koko projektin perumiseen. Jokainen vaatimuksen perusteella tehty suunnittelupäätös johtaa kooditasolla useampaan eri

päätökseen, joka testausvaiheessa vaikuttaa eri tavalla. Yksikin virheellisesti väärin tulkittu vaatimus voi siis aiheuttaa huomattavan paljon ylimääräistä työtä monella eri tasolla. Mitä aikaisemmin huono tai virheellinen vaatimus huomataan, sitä pienemmäksi mahdollinen vahinko jää. (IEEE Computer Society, 2014, s. 1–3.)

4.2.1 Vaatimusten käsittelyn osa-alueet

Vaatimusten käsittely muodostuu kahdesta eri kokonaisuudesta: vaatimusmäärittely ja vaatimusten hallinta. Vaatimusmäärittely vaatii usein oman projektinsa, ja se voidaan vielä jakaa neljäksi kuvassa 7 kuvattuun eri osa-alueeseen, jotka ovat kartoittaminen, analysointi, dokumentointi ja validointi.



Kuva 7. Vaatimusten käsittelyprosessi. (Hakala & Mikkonen, 2011, s. 65.)

Vaatimusten kartoituksen tavoitteena on ymmärtää sidosryhmien tekemää työtä ja sitä, miten uutta järjestelmää voitaisiin käyttää tuon työn tukemiseen. Vaatimusten kartoituksen aikana kehittäjät työskentelevät yhdessä sidosryhmien kanssa selvittääkseen sovellusalan, työtoiminnot, halutut palvelut ja järjestelmän ominaisuudet, suorituskykyvaatimukset ja esimerkiksi laitteistorajoitukset. (Collegenote, n.d.)

Tavallisimpia tiedonkeruumenetelmiä kartoitusvaiheessa ovat esimerkiksi erilaiset haastattelut, aivoriihet ja työpajat. Vaatimusten analysoinnissa

vaatimuksia ja niiden välisiä suhteita selvitetään ja priorisoidaan. Dokumentointivaiheessa vaatimukset kerätään sovittuun dokumenttiin, kuten esimerkiksi Excel-taulukkoon. Vaatimusten validointi päättää vaatimusmäärittelyprosessin. Validoinnin yhteydessä kerätyt vaatimukset kelpuutetaan asiakkaan kanssa. (Hakala & Mikkonen, 2011, s. 66.)

4.2.2 Vaatimukset

Vaatimus on joko toiminto, jota tuotteen tulee pystyä suorittamaan, tai ominaisuus, joka tuotteella tulee olla. Ohjelmistotuotannossa vaatimukset voidaan jakaa kolmeen eri luokkaan: toiminnallinen vaatimus, ei-toiminnallinen vaatimus ja reunaehdot. (Hakala & Mikkonen, 2011, s. 61.)

Toiminnalliset vaatimukset kuvaavat järjestelmän perustoiminnot ja ominaisuudet. Nämä määrittävät konkreettisesti, mitä toimintoja järjestelmän odotetaan suorittavan tai mitä tavoitteita sen tulee saavuttaa. Esimerkiksi, verkkokaupan yhteydessä toiminnallinen vaatimus voi olla, että "käyttäjän on pystyttävä lisäämään tuote ostoskoriin", tai että "järjestelmän tulee pystyä käsittelemään maksutapahtumat luottokortilla". (Hakala & Mikkonen, 2011, s. 61.)

Ei-toiminnalliset vaatimukset puolestaan viittaavat järjestelmän laatuun tai visuaalisiin piirteisiin. Nämä saattavat esimerkiksi määrittää järjestelmän suorituskyvyn, luotettavuuden tai käytettävyyden standardit. Esimerkkivaatimuksia voisivat olla, että "järjestelmän on pystyttävä käsittelemään vähintään 10 000 samanaikaista käyttäjää", tai että "verkkosivun latautumisaika ei saa ylittää 3 sekuntia". (Hakala & Mikkonen, 2011, s. 61.)

Reunaehtovaatimukset määrittelevät puolestaan rajoitteita tai ohjaavat järjestelmän suunnittelua ja toteutusta. Esimerkki tällaisesta reunaehtovaatimuksesta on vaatimus, jonka mukaan esimerkiksi ohjelmiston on noudatettava GDPR-tietosuoja-asetuksen mukaisia määräyksiä. (Hakala & Mikkonen, 2011, s. 61.) Toinen esimerkki voisi olla, että ohjelmiston tulee olla rakennettu

käyttäen C# ohjelmointikieltä – kuten tämän opinnäytetyön tuotoksessa on mahdollista.

ISO (the International Organization for Standardization) ja IEC (the International Electrotechnical) standardin ISO/IEC/IEEE 29148 mukaan vaatimuksen tulisi olla välttämätön, toteutusvapaa, yksiselitteinen, yhdenmukainen, täydellinen, yksittäinen, toteutettavissa, jäljitettävä ja todennettava. Vaatimuksen on siis oltava välttämätön tarkoittaen, että sen poistaminen tai hylkääminen johtaisi sellaiseen puutteeseen, jota ei voi korvata muilla vaatimuksilla tai kehityksessä olevan sovelluksen komponenteilla. Vaatimuksen on oltava toteutusvapaa, eli teknistä toteutustapaa ei rajoiteta ollenkaan. Tämä edistää erilaisten toteutusratkaisujen harkintaa. Vaatimuksen on oltava täysin yksiselitteinen ja helposti ymmärrettävissä. Sen tulee olla selkeästi muotoiltu niin, että siitä on vain yksi mahdollinen tulkinta. Vaatimuks(i)en yhdenmukaisuudella tarkoitetaan, että vaatimus ei ole ristiriidassa minkään toisen vaatimuksen kanssa. Vaatimuksen täydellisyyden tulisi olla niin pitkälle kehitetty, että se on mitattavissa ja kuvaa riittävästi tarvittavaa kykyä tai ominaisuutta. Vaatimuksen on oltava yksittäinen ja selkeästi erottuva, ilman monimutkaisia ehtoja tai muita rajaavia tekijöitä. Se on oltava myös toteutettavissa ja sovitettava projektin tai järjestelmän rajoituksiin huomioiden tärkeät asiat, kuten kustannukset, aikataulut ja tekniset rajoitteet, ja sen tulee olla teknisesti toteutettavissa nykyisillä resursseilla ja nykyään käytettävissä olevalla teknologialla. Vaatimus tulee olla jäljitettävissä ja toteutuminen varmistettavissa. Vaatimuksen tulisi olla jäljitettävissä alkuperäisiin sidosryhmän tarpeisiin tai ylemmän tason vaatimuksiin sekä jäljitettävissä alempien tasojen vaatimuksiin tai muihin järjestelmän määrittelydokumentteihin. Vaatimuksen tulee olla todennettava ja mitattavissa oleva. Vaatimuksen toteutuminen ja täyttymien on oltava mahdollista todentaa. (ISO/IEC/IEEEE 29148:2018, 2018.)

4.3 Ohjelmistosuunnittelu

Ohjelmistosuunnittelu on olennainen osa ohjelmiston elinkaarta. Käytännössä ohjelmistosuunnittelulla tarkoitetaan asiakkaan vaatimusten ja tarpeiden muuttamista konkreettisiksi suunnitelmiksi, jotka myöhemmin toteutetaan. Se toimii yhdistävänä tekijänä, yhdistäen ohjelmiston vaatimukset, rakentamisen, testauksen ja ylläpidon. Ohjelmistosuunnitteluvaiheessa käsitellään asiakkaan vaatimuksiin liittyviä ongelmia ja haasteita ja pyritään löytämään ratkaisut näiden vaatimusten täyttämiseksi. Ohjelmistosuunnittelun on oltava sopusoinnussa arkkitehtuurin kanssa ja otettava huomioon järjestelmän komponentit, yhteydet, ohjelmointirajapinnat ja muut arkkitehtuurivalinnat. Se myös ohjaa niitä työntekijöitä, jotka rakentavat ohjelmiston, antaen suuntaviivoja siitä, miten järjestelmä tulisi rakentaa. Lisäksi ohjelmistosuunnittelu luo perustan testaukselle, jonka tarkoituksena on varmistaa, että ohjelmisto toimii oikein ja suunnitellusti. (IEEE Computer Society, 2014)

5 TUTKIMUSOSUUS

Opinnäytetyön tavoitteena on luoda toimeksiantajalle uusi toiminnallisuus käytettäväksi erilaisiin integraatioratkaisuihin. Toiminnallisuus rakennetaan Frends-integraatioalustalle hyödyntäen alustan low-code-ominaisuuksia tai vaihtoehtoisesti koodaamalla uusi nuget-paketti C# ohjelmointikielellä. Opinnäytetyön kappaleessa 3 käsiteltiin erilaisia ohjelmistotuotannon käsitteitä ja vaatimusten käsittelyprosessia. Tutkimusosuuden tarkoitus on selvittää vaatimukset, jotka uuden toiminnallisuuden tulisi valmistuessaan täyttää.

Vaatimusten hankinta on tiedonkeruuta, jossa pyritään saamaan syvällistä tietoa ongelma-alueesta, jotta sitä voidaan hyödyntää järjestelmän tai integraation suunnittelussa ja toteutuksessa. On tärkeää tunnistaa, mikä tieto on keskeistä, mistä lähteistä se saadaan luotettavimmin, ja miten tätä hankittua tietoa tulisi käyttää projektin edetessä (JUHTA, 2018a.)

Opinnäytetyön tuotoksen vaatimukset kerätään strukturoimattomilla haastatteluilla ja dokukenttien tutkimisella. Vaatimukset tämän jälkeen analysoidaan ja dokumentoidaan vaatimusluetteloon (LIITE 1: Vaatimusluettelo). Analysoinnin yhteydessä luodaan käyttötapauslomake (LIITE 2: Käyttötapauslomake). Liitteet tullaan salaamaan opinnäytetyön julkisesta versiosta. Liitteet sisältävät liiketoiminnan kannalta sellaisia tietoja, jotka ovat luottamuksellisia tai arkaluonteisia, kuten sisäisiä käytäntöjä ja tapoja tai sellaista teknistä tietämystä, joiden julkistaminen ei ole toimeksiantajan etujen mukaista. Ennen toteutusta vaatimusluettelo ja käyttötapauslomake validoidaan sidosryhmien kanssa.

5.1 Tiedonkeruumenetelmät

Dokumenttien tutkimisen tavoitteena on kerätä materiaalia, tässä tapauksessa vaatimuksia, jo olemassa olevan dokumentaation tai materiaalin perusteella. Materiaali voi olla esimerkiksi jo tehdyt määritykset, valmiit standardit, käyttöohjeet, koulutusmateriaali tai oppikirjat. Ongelma-alueeseen soveltuva materiaali käydään läpi tutkien ja tarkkaillen sellaisia asioita, jotka ovat oleellisia

vaatimusmäärittelyn kannalta. Löytyneistä asioista kirjataan vaatimuksia. (JUHTA, 2018a.)

Dokumenteista on huomioitava erityisesti ne kohdat, joissa kuvataan toiminnan tavoitteita ja olennaisia ominaisuuksia. Erityisesti kohdat, joista voidaan muodostaa lause: ”Järjestelmän tulee…” ovat tällaisia. Dokumenttien tutkiminen voi olla oleellisen tiedon löytymisen kannalta ja järjestelmällisesti tehtynä työlästä ja se voi viedä paljon aikaa. Etuna dokumenttien tutkimisessa kuitenkin on, että jo olemassa olevia ratkaisuja voidaan hyödyntää ja varmistaa, että jotkin olemassa olevat ratkaisut ovat yhteensopivia tuotettavan ratkaisun kanssa. (JUHTA, 2018a.)

Koska opinnäytetyön toteutuksen tarkoitus on luoda korvaava toiminnallisuus, dokumenttien ja muun materiaalin tutkiminen on oleellinen tiedonkeruumenetelmä. Jos jotakin korvataan, se tarkoittaa, että jokin alkuperäinen asia on jo olemassa ja sen ominaisuuksia ja toiminnallisuuksia voidaan hyödyntää uuden rakentamisessa. Opinnäytetyön tapauksessa korvattavan toiminnon ominaisuudet on varsin hyvin dokumentoitu, ja dokumentaatio on saatavilla esimerkiksi avoimen lähdekoodin muodossa. Toisena merkittävänä etuna muihin tiedonkeruumenetelmiin, kuten haastatteluihin, on että tutkiminen ei vaadi muiden henkilöiden osallistamista tiedonkeruuseen. Tutkimisen avulla päästään keräämään vaatimuksia oma-aloitteisesti ja omassa tahdissa ilman ulkoisia aikataulupaineita.

Suullinen strukturoimaton haastattelu on tiedonkeruumenetelmänä haastava. Onnistuminen haastattelussa vaatii haastattelijalta vahvaa ymmärrystä käsiteltävästä aihealueesta ja haastattelutaitoa. Strukturoimattomassa haastattelussa on myös riski siihen, että keskustelu lähtee helposti raiteiltaan muihin aiheisiin. Strukturoimatonta haastattelua varten haastattelija sopii ainakin yhden keskusteltavan asian, mutta tarvittaessa aiheita voi olla useampiakin. Luonteeltaan haastattelu on erittäin interaktiivinen ja samalla ohjattu keskustelu. Keskustelun tarkoituksena on, että haastattelija saa haastateltavan mielestä oleellista ja oikeaa informaatiota. (JHS, 2018.)

Suullinen strukturoimaton haastattelu valittiin toiseksi tiedonkeruumenetelmäksi sen joustavuuden takia. Haastatteluita oli helppo pitää nopeallakin aikataululla, koska menetelmä ei vaadi mitään esimerkiksi etukäteen rakennettua lomaketta, jonka mukaan haastattelu etenisi. Toimintaympäristön takia haastattelut saatettiin pitää ad-hoc tyylisesti, johon strukturoimaton haastattelu tarjosi parhaimmat menetelmät. Lisäksi haastattelijalla oli strukturoimattoman haastattelun onnistuminen kannalta hyvä määrä ymmärrystä ja työkokemusta käsiteltävästä toimialasta, jolloin kyseinen tiedonkeruumenetelmä tuntui luontevalta valinnalta.

5.2 Vaatimusmäärittelyn dokumentaatio

Opinnäytetyössä hyödynnettiin JHS-suosituksen mukaisia valmiiksi laadittuja mallipohjia: JHS 173 ICT-palvelujen kehittäminen: Vaatimusmäärittely, liite 1 Vaatimusluettelo-mallipohjaa ja JHS 173 ICT-palvelujen kehittäminen: Vaatimusmäärittely, liite 2 Käyttötapauslomake-mallipohjaa. Julkisen hallinnon suosituksia (JHS), käyttivät laajasti julkisen sektorin toimijat, esimerkiksi kunnat ja valtion laitokset vuosina 1992–2019 (JUHTA, 2018a). Vaikka opinnäytetyön toimeksiantaja ei ole julkisen hallinnon toimija, JHS:n tarjoama vaatimusluettelon mallipohja vastasi tehokkaasti opinnäytetyölle asetettuja tarpeita.

Jokainen vaatimus yksilöidään ID-tunnisteella, joka tässä tapauksessa on juokseva numerointi. Seuraavaksi luetteloon kirjotetaan varsinainen vaatimus noudattaen hyvän vaatimuksen mukaista rakennetta. Vaatimusluettelopohjaan täydennetään myös vaatimuksen esittäjä ja päivämäärä, jolloin vaatimus on kirjattu. Vaatimukselle annetaan myös tärkeysluokka. Lisäksi vaatimukselle annetaan lyhytmuotoinen perustelu. Perustelun tarkoituksena on auttaa vaatimuksien luokittelussa ja priorisoinnissa. (JUHTA, 2018a.) Tässä opinnäytetyössä on käytetty 3-tasoista luokitusta, jossa 1 tarkoittaa pakollista, 2 tarkoittaa hyödyllistä ja 3 toivottua vaatimusta. Kuvassa 8 on esitetty erilaisia esimerkkivaatimuksia vaatimusluetteloon täytettynä.

ID	VAATIMUS	VAATIMUKSEN ESITTÄJÄ	PVM	TÄRKEYS	PERUSTELU
	Käyttäjän on voitava muuttaa salasanansa järjestelmää käyttäessään 30 sekunnin suoritusajassa.	Matti Meikäläinen	12.5.2007	2	Käyttömukavuus ja turvallisuus
	Käyttäjän on voitava lähettää pyyntö järjestelmätukeen unohtuneen salasanan lähettämiseksi sähköpostilla 30 sekunnissa.	Asiakaspalvelu	12.5.2007	2	Käyttömukavuus ja turvallisuus

Kuva 8. Vaatimusluetteloesimerkki (JUHTA, 2018b).

Käyttötapaukset kuvataan kierroksittain eli iteratiivisesti. Ensimmäisen kierroksen tavoitteena on kirjoittaa ensimmäinen versio mahdollisimman nopeasti ja siirtyä seuraaviin käyttötapauksiin. Toisen kierroksen tarkoituksena on tarkentaa ensimmäisen kierroksen käyttötapauslomaketta ja varmistua siitä, että käyttötapaus on oikeasti mahdollinen. Kolmannella kierroksella pyritään mahdollisimman yksityiskohtaiseen kuvaukseen käyttötapauksesta, ja tapahtumakulku numeroidaan. Lopullisessa käyttötapauslomakkeessa tulisi siis ilmetä käyttötapauslomakkeen perustiedot, kuten laatijan nimi, päiväys, käyttötapauslomakkeen nimi, suorittajat, esitiedot, kuvaus prosessin kulusta, mahdolliset poikkeukset, käyttötapauksen lopputulos ja mahdolliset muut vaatimukset. (JUHTA, 2018a.) Kuvassa 9 on esimerkkinä täytetty käyttötapauslomake kuvitteellisesta kokoustilan varaamisesta.

1. Laatija		Matti Meikäläinen
2. Päiväys		1.1.2007
3. Prosessi		Sisäiset tukiprosessit
4. Nimi		Kokoustilan varaaminen
5. Suorittajat		Asiakaspalveluhenkilöstö
6. Esitiedot		Kokoustilan käyttökalenteri on ajantasalla
7. Kuvaus	1	Käyttäjä aukaisee yrityksen intranetin.
	2	Käyttäjä klikkaa linkkiä ja aukaisee kokoustilavaraukset.
	3	Käyttäjä pyytää järjestelmää näyttämään kokoustilojen varaukset haluamaltaan ajalta syöttämällä viikon numero ja päivän.
	4	Käyttäjä saa eteensä varausnäytön, josta näkee kaikkien kokoustilojen vapaat ajat ja varauksen tekijöiden nimet.
	5	Käyttäjä valitsee sopivan kokoustilan ja varaa sen.
	6	Käyttäjä saa tilan varauksesta vahvistuksen omaan sähköpostiinsa ja merkinnän sähköiseen kalenteriinsa
8. Poikkeukset	1	Käyttäjä valitsee vahingossa jo varatun tilan - järjestelmä ilmoittaa että kokoustila on on varattu.
9. Lopputulos		Kokoustila on varattu.
10. Muut vaatimukset	1	Järjestelmän vastausajan oltava max 2 sekuntia.
	2	Kokoustilan varausnäytön päivitys saa kestää 4 sekuntia.
	3	
	4	

Kuva 9. Käyttötapauslomake (JUHTA, 2018c).

6 RATKAISU

Tähtisen (2005, s.105) mukaan järjestelmäintegraatioita toteuttavan tahon tehtävä on huolehtia siitä, että järjestelmien väliset integraatiot on toteutettu niin, että integraatiot ovat mahdollisimman pitkälle automatisoituja, huoltovapaita, vikasietoisia, valvottavia, skaalautuvia, tarvittaessa muokattavia ja niiden tietoturvallisuus on huomioitu. Opinnäytetyön tuotoksena on yksittäinen integraation komponentti, jota voidaan hyödyntää monipuolisesti erilaisissa integraatioarkkitehtuureissa ja integraatioissa, jotka tarvitsevat jonkinlaista tiedoston siirtoa hyödyntäen erilaisia tiedonsiirtoprotokollia. Vaikka tuotos ei siis ole yksittäinen oma integraatio, vaan erillinen työkalu integraatioihin niin vaatimusmäärittelyssä kuitenkin toistuivat samat edellä mainitut teemat. Tässä luvussa tullaan kertomaan mitä vaatimuksiin perustuvia toimintoja ja/tai toiminnallisuuksia ratkaisuun tullaan rakentamaan. Käsittely tapahtuu ei-toiminnallisten ja toiminnallisten vaatimusten erottelun avustuksella. Samalla perustellaan, miksi johonkin ratkaisuun on päädytty toiminnon tai toiminnallisuuden rakentamisessa. Luvun lopussa on yhteenvetokappale, joka kokoaa vaatimukset ja opinnäytetyön ratkaisun yhdeksi helposti hahmotettavaksi kokonaisuudeksi.

6.1 Ratkaisun ei-toiminnalliset vaatimukset

Frends tarjoaa varsin laajat valmiit työkalut esimerkiksi SFTP-toimintoihin omilla taskeillaan. Taskit tarjoavat esimerkiksi mahdollisuuden poistaa, ladata, listata tai uudelleen nimetä tiedostoja SFTP-yhteyttä käyttäen. (Frends, n.dc.) Vaikka taskit ovat varsin kattavat, ne eivät kuitenkaan täytä kaikkia opinnäytetyölle asetettuja vaatimuksia. Esimerkiksi vaatimuksen 5.1 mukaan tuotettavan ratkaisun on kyettävä samalla tukemaan useampia käytössä olevia tiedonsiirtoprotokollia. Näiden vaatimusten täyttämiseksi Frends tarjoaa mahdollisuuden käyttää muita ratkaisuja. Kappaleessa 3.1. todetaan, että Frends tarjoaa erilaisia keinoja rakentaa prosesseja, tässä tapauksessa integraatioita, kuten esimerkiksi low-code ja no-code-menetelmät, jotka tekevät integraatioprosessin tuotantoon viennistä nopeamman ja helpomman. Sen lisäksi Frends tarjoaa mahdollisuuden tuottaa omia koodiratkaisuja, joiden avulla on

mahdollista luoda mukautuvuutta ja niiden avulla voidaan täydentää toimeksiannon kannalta havaittuja puutteita Frendsin valmiiden taskien toiminnallisuudessa. Opinnäytetyön ratkaisu tullaan rakentamaan hyödyntäen kaikkia edellä mainittuja tekniikoita eli lopputuloksena on eräänlainen hybridiratkaisu, joka hyödyntää Frendsin valmiita taskeja ja muita toiminnallisuuksia ja sen lisäksi C# koodia.

Yksi keskeinen syy, sille miksi ratkaisu päätettiin toteuttaa aliprosessina hyödyntäen enemmän low-code-toiminnallisuuksia ja Frends omia taskeja, johtuu ei-toiminnallisista vaatimuksista 7.2, 7.3 ja 7.4. Toimeksiantajan integraatioalustalla on käytössä satoja integraatiota. Lähes jokainen integraatio hyödyntää toiminnassaan jollakin tavalla tiedoston siirtoa, johon opinnäytetyön toteutusta voidaan käyttää. Siirto voi olla paikallista siirtoa eli siirtoa tietokoneen levyasemalta toiselle, tai esimerkiksi SFTP-palvelimelta siirtoa toiselle palvelimelle tai vaihtoehtoisesti jokin näiden variaatioiden yhdistelmä. Nämä prosessit voivat käynnistyä useita kertoja päivässä, kerran päivässä tai esimerkiksi kerran puolessa vuodessa. Lopputuloksena yksittäisiä prosessin ilmentymiä voi siis olla kymmeniä tuhansia kuukaudessa ja satoja tuhansia vuodessa. Tämän takia integraatioiden yleisesti ja opinnäytetyönä tuotettavan ratkaisun on oltava ennen kaikkea helposti ylläpidettäviä, ymmärrettäviä, laajennettavia, käyttöturvallisia. Low-code toteutuksen merkittävimpinä etuina verrattuna perinteiseen koodaukseen pidetäänkin juuri näitä edellä mainittuja teemoja (Tozzi, 2021).

Low-code-ratkaisuun ohjasivat myös vaatimukset luotettavasta virheenkäsittelystä. Virheenkäsittely ei ole pelkästään toivottavaa, vaan se on elintärkeää liiketoiminnan jatkuvuuden kannalta. Tiedon eheys ja tiedon siirto on varmistettava myös odottamattomien vikojen tapahtuessa. Integraation toimimattomuus voi aiheuttaa taloudellisesti jopa merkittäviä menetyksiä ja asiakasluottamuksen katoamisen. Liiketoimintaympäristön säännökset ja asiakkaan odotukset asettavat omat vaatimuksensa vikasietoisuuden varmistamiselle. Pitkällä aikavälillä vikasietoisuuteen tehtävät investoinnit voivat vähentää hätätoimenpiteiden ja tietojen palautustoimintojen tarvetta, mikä voi luoda omalta osaltaan kustannussäästöjä. (Tähtinen 2005, s. 107.) Aiemmin kuvatut

integraatioiden käynnistysvälit korostavat sekä vikasietoisuuteen että virheenkäsittelyyn liittyvien vaatimuksien tärkeyttä. Vaatimusluettelon (LIITE 1.) vaatimukset 3.1 ja 3.2 edustavat tämän teeman osalta esitettyjä ei-toiminnallisia vaatimuksia, jotka toteutuksen on täytettävä.

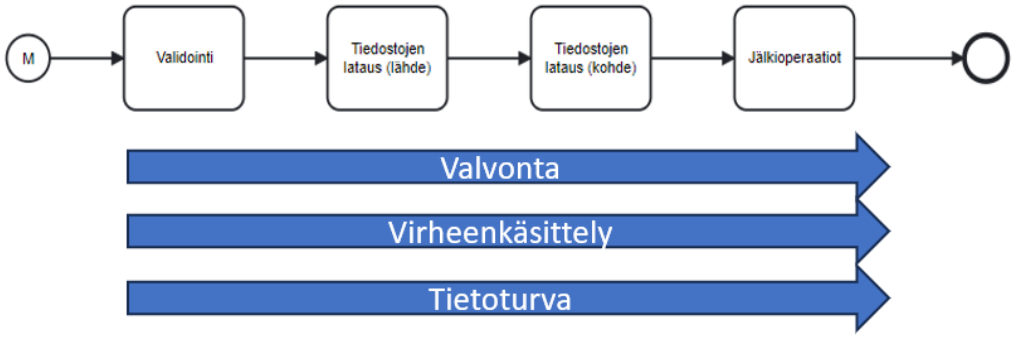
Loppujen lopuksi yrityksen liiketoiminta perustuu luottamukseen asiakkaidensa kanssa. Toimeksiantajan liiketoiminta keskittyy toimialalle, jossa korostuu tietoturvan merkitys. Palkanlaskennassa ja henkilöstöhallinnossa keskeisessä asemassa ovat erilaiset henkilötiedot, ja henkilötietojen käsittelyä ohjaa EU:n GDPR-säädökset. GDPR:n tietosuojaperiaatteissa velvoitetaan muun muassa, että henkilötietoja on käsiteltävä kaikissa mahdollisissa vaiheissa turvallisesti (Tietosuoja, 2020). Tämä pätee siis myös integraatioihin, joissa siirretään henkilötietoja. Tavallisesti yrityksellä voi olla esimerkiksi useampi tietojärjestelmä, jotka sisältävät jotakin henkilötietoja, mutta vain yksi järjestelmä, johon henkilötiedot alun perin tuodaan. Tässä esimerkkitapauksessa integraatioilla voidaan siirtää esimerkiksi henkilötiedot järjestelmästä toiseen. Integraatioprosessin ja ennen kaikkea sen käyttäjän tulee myös huolehtia siitä, että GDPR:n tietosuojaperiaatteita noudatetaan, ja prosessi toimii jokaisella mittarilla katsottuna tietoturvallisesti noudattaen yleisesti hyväksyttyjä parhaita tietosuojausmenetelmiä. Vaatimusluettelon (LIITE 1.) vaatimukset 3.4 ja 3.5 edustavat tietosuojan teemalle esitettyjä vaatimuksia. Low-code ratkaisun katsottiin toteuttavan huomattavasti helpommin kyseisen teeman vaatimuksia, sillä integraatioalusta tarjoaa tähän hyväksi todetut työkalut.

Toinen vaihtoehto ratkaisulle olisi ollut toteuttaa Frendsiin niin sanottu ”custom task”. Custom task käytännössä tarkoittaa C#-kielellä koodattua luokkakirjastoa, joka toteuttaa tiettyä tai tiettyjä toimintoja (Galkin, 2024c). Tässä tapauksessa custom taskin toiminto olisi siis ollut tiedostonsiirto hyödyntäen erilaisia tiedonsiirtoprotokollia. Perinteisellä koodauksella saadut edut eivät kuitenkaan olleet yhtä merkittävät kuin low-code toteutuksesta saatavat edut. Perinteisen koodauksen etuja low-codeen verrattuna ovat esimerkiksi suurempi joustavuus ja mukautuvuus, syvällisempi hallinta ja ymmärrys lähdekoodista ja parempi suorituskyvyn optimointi erityistarpeisiin (Tozzi, 2021).

6.2 Ratkaisun toiminnalliset vaatimukset

Luvussa 3.1 kerrottiin, että yksittäisen prosessin, pää- tai aliprosessin, voidaan ajatella koostuvan kolmesta eri osuudesta: aloituksesta, toiminnallisesta osuudesta ja niin sanotusta end pointista. Tässä kappaleessa kuvataan, miten aliprosessi rakennetaan, ja miten toteutus tulee täyttämään erilaisia sille asetettuja vaatimuksia. Kappaleessa kuvataan myös prosessin rakenteellinen kokonaisuus eli kaikki ne vaiheet, mitä aliprosessille on luotu, mikä muodostaa aliprosessin toiminnallisuuden osuuden.

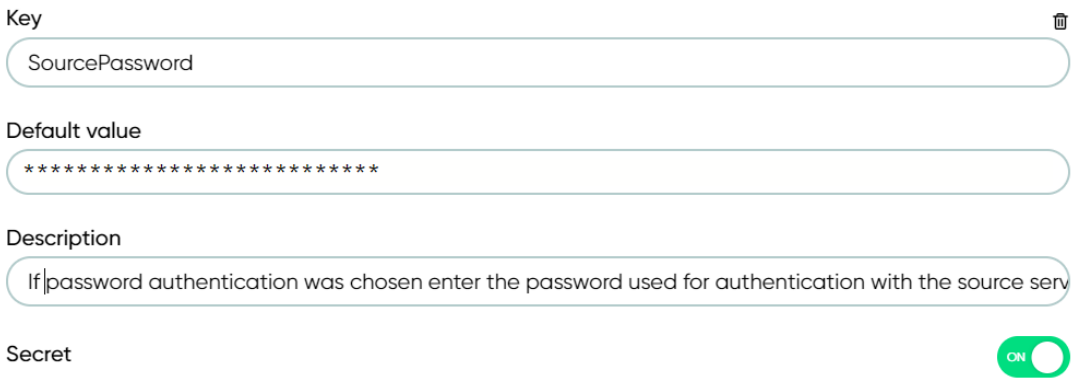
Vaatimusmäärittelyprosessissa ratkaisulle syntyi neljä selvästi erilaista tunnistettavaa varsinaista toiminnallisuutta koskettavaa vaihetta, jotka tullaan ratkaisussa huomioimaan. Vaiheet ovat validointivaihe, tiedostojen haku, tiedostojen lataaminen ja jälkioperaatiot. Tämän lisäksi kaikki edellä mainitut vaiheet saattavat sisältää koko prosessia koskettavia toimintoja, kuten esimerkiksi prosessin valvottavuutta koskevat toiminnot. Osan näistä toiminnoista voidaan ajatella olevan osittain sekä toiminnallisia että ei-toiminnallisia vaatimuksia. Kuvassa 10 on hahmotettu prosessin kokonaisuutta, eli toiminnallisiin vaatimuksiin perustuvia vaiheita ja toimintoja, ja niitä koskettavia teemoja, jotka kulkevat läpi prosessin. Luonnoksessa on hyödynnetty muun muassa Frendsin tarjoamaa BPMN-kaaviomerkintää.



Kuva 10. Prosessin toiminnalliset vaatimukset.

Koska toiminnallisuus tullaan rakentamaan aliprosessille ja se tulee toimimaan eräänlaisena palveluna useammalle pääprosessille, on aliprosessille luotava jokin mekanismi, joka ohjaa sen toimintaa. Tätä varten aliprosessilla tullaan

käyttämään triggeriä, johon voidaan pääprosessin kautta antaa aliprosessia ohjaavia parametreja. Rakennettavan toteutuksen triggerinä tullaan käyttämään yksinkertaista manuaalitriggeriä, joka käynnistää aliprosessin aina kun pääprosessin suoritus pääsee aliprosessille asti. Manuaalitriggerille voidaan antaa erilaisia parametreja. Jokaisella parametrilla tulee olemaan kolme kenttää: key, description ja default value. Parametri voidaan asettaa myös salaiseksi, jolloin käyttöliittymä piilottaa parametrin arvot. Parametrit tullaan luomaan niin, että niiden nimet kuvaavat niiden toiminnallisuutta myöhemmin aliprosessin vaiheissa. Kuvassa 11 esimerkki lähdepalvelimelle annettavasta salasanaparametrista. Description-kentässä käyttäjälle kerrotaan suorituksen kannalta tarvittavia lisätietoja.



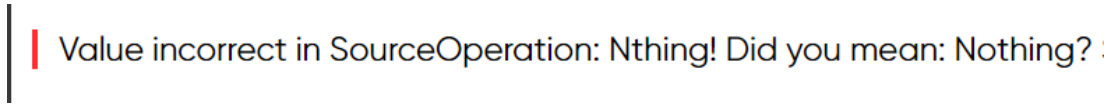
Kuva 11. Aliprosessin parametrit.

Muita parametreja ovat esimerkiksi siirrettävän tiedoston nimi ja kansiosijainti. Opinnäytetyön liitteestä (LIITE 3: Parametrit) löytyvät taulukoituna aliprosessille annettavat parametrit ja niiden käyttötarkoitukset.

Aliprosessin varsinainen toiminnallisuus käynnistyy luomalla uusi JObject luokan muuttuja aliprosessille toteuttavan lokitusmekanismin tueksi. JObjectia käytetään aliprosessin lisälokituksessa kuvaamaan aliprosessin eri vaiheiden suoritusta, kuten tässä kappaleessa myöhemmin kuvattujen validaatiofunktioiden tuloksia. JObject on dynaaminen tietorakenne, joka on osa Newtonsoft.Json-kirjastoa, ja sitä käytetään JSON-muotoisen tiedon käsittelyyn (Newtonsoft, 2023). JObject palautetaan virheen sattuessa virheviestiin käyttäjälle ja se nostetaan myös aliprosessin päättyessä pääprosessille siirtyvään tulokseen.

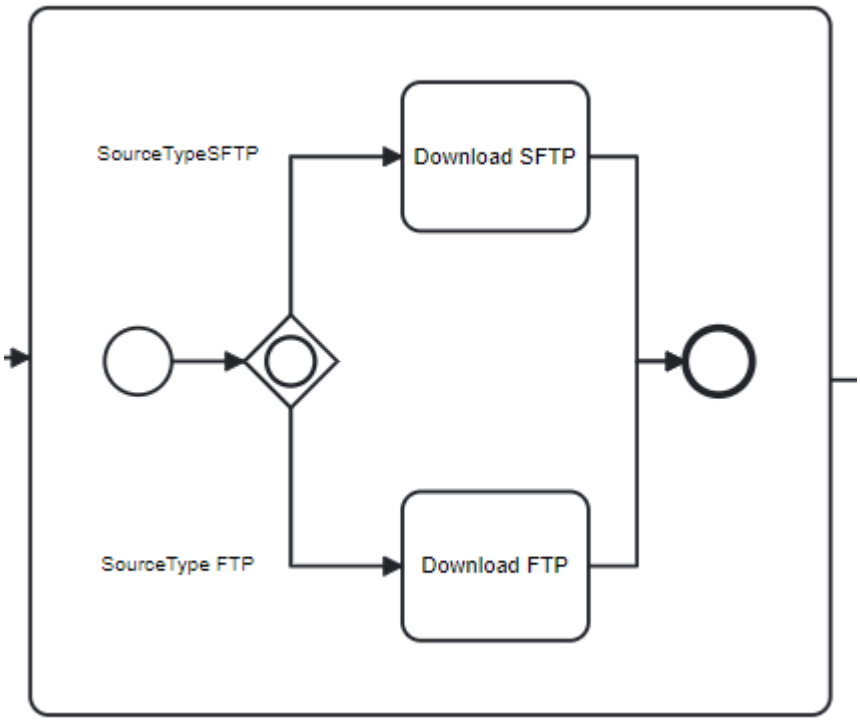
Suoritus jatkuu tästä inklusiivisella päätöksenteolla, jossa aliprosessi kuljetetaan parametreille suunnatun validointiprosessin läpi riippuen manuaalitriggerille annetuista parametreista ”SourceType” ja ”DestinationType”. Validoinnilla vahvistetaan aliprosessin virheenkäsittelyä ja vikasietoisuutta, ja sillä pyritään saamaan virheet kiinni ennen kuin suoritus pääsee toteuttamaan tiedostonsiirtoja. Vaatimus 5.9 vaatii, että toteutuksessa on huomioitava ennen varsinaisia tiedostosiirtoja, että triggerille annetut parametrit ovat valideja. Eri parametreilla on erilaisia validointisääntöjä. Suorituksen kannalta pakolliset parametrit käydään läpi ja varmistetaan, että käyttäjä ei ole jättänyt näitä tyhjäksi.

Parametrit, jotka tullaan antamaan myöhemmin eri taskeille syötteeksi, käyvät läpi validoinnin, jossa varmistetaan, että parametri pystytään tarvittaessa muuttamaan vaadittuun tyypitykseen. Esimerkiksi pääprosessilta annettu parametri SFTP-portille on oltava sellaisessa muodossa, että sen pystyy muuttamaan kokonaisluvuksi. Toinen useasti toistuva validointi hoitaa merkkijonona annetun parametrin yhteensovittamisen myöhemmin käytettävien taskien eri luettelotyyppien kanssa. Luettelotyyppien validoinnin epäonnistumista varten aliprosessille on rakennettu toiminnallisuus, joka hyödyntää Levenshtein-algoritmia. Levenshtein-algoritmi mittaa kahden merkkijonon välistä etäisyyttä laskemalla, kuinka monta yksittäistä muokkausta, eli merkin lisäystä, poistoa tai korvaamista tarvitaan, jotta ensimmäinen merkkijono muunnetaan toiseksi. Parametrina annettu virheellinen luettelotyyppi jää validoinnissa virheeseen, ja algoritmi ehdottaa käyttäjälle lähimpänä olevaa mahdollista vaihtoehtoa. Tämä kattaa vaatimuksen 5.10 vaatimaa toiminnallisuutta. Kuvassa 12 on aliprosessin nostama virheilmoitus. Tässä esimerkkitapauksessa käyttäjä on antanut SourceOperation-kenttään parametriksi ”Nthing”, kun mahdolliset parametrit kyseessä olevassa luettelotyypissä ovat ”Nothing”, ”Info” ja ”Error”.



Kuva 12. Virheviesti.

Parametrien validoinnin jälkeen aliprosessin suoritus siirtyy toteuttamaan lähdesijainnin tiedostonsiirtoja. Suoritusta ohjataan jälleen oikeaan polkuun käyttäen inklusiivista päätöksentekoa. Tiedostonsiirtotoiminnoissa hyödynnetään esimerkiksi Frendsin tarjoamia valmiita FTP-, SFTP- ja paikalliseen siirtoon käytettäviä taskeja. Kuvassa 13 on esitetty prosessin vaihetta lähdesijainnin osalta.



Kuva 13. Lähdesijainnin lataustoiminnot yksinkertaistettuna.

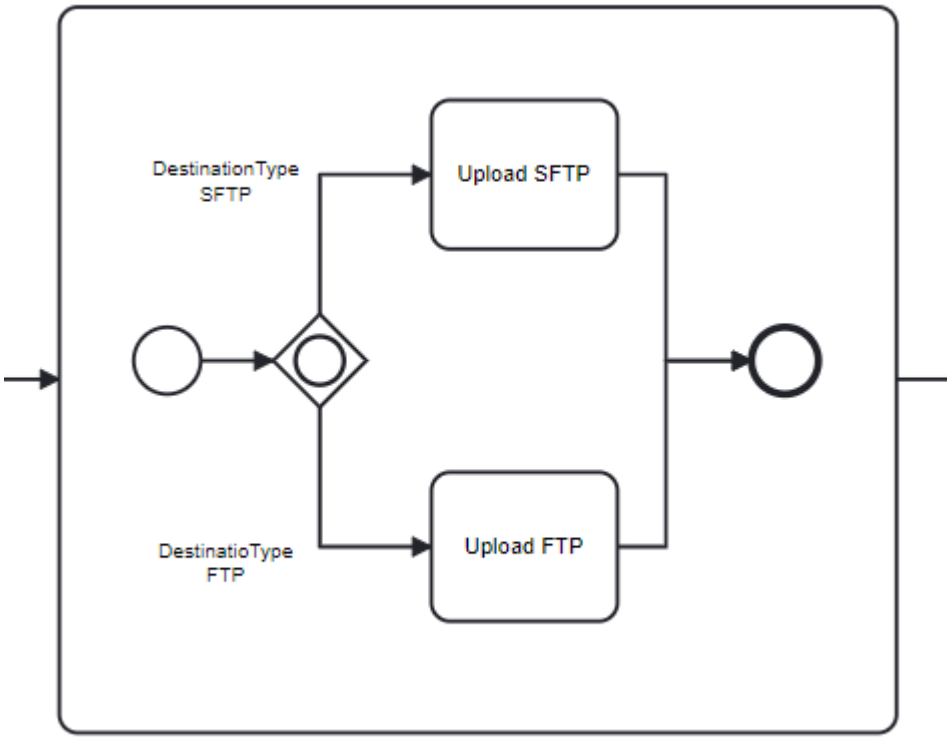
Taskit eivät kuitenkaan välttämättä täysin sellaisenaan kelpaa opinnäytetyölle asetettuihin vaatimuksiin. Frendsin valmiit taskit on luotu suorittamaan vain yksittäisiä toimintoja, kuten tiedostojen lataamista yhdeltä SFTP-palvelimelta. Tämä tarkoittaa, että esimerkiksi tiedoston laataminen ja lähettäminen palvelimelta toiselle ilman välikäsittelyä ei onnistu käyttämällä vain yhtä taskia. Toisaalta mikäli tiedostosiirrossa joko lähde- tai kohdesijainti on paikallinen, toteutuksessa voidaan käyttää pelkästään Frendsin omia taskeja ilman välikäsittelyä. Palvelimelta toiselle palvelimelle -muotoisen siirron vuoksi toiminnolle tullaan rakentamaan välikäsittely, jonka avulla luodaan tarvittava toiminnallisuus tiedostojen sujuvaan siirtoon useiden taskien ketjuttamisella. Välikäsittely sisältää tiedoston väliaikaisen paikallisen tallennuksen. Väliaikaista

tallennusta varten luodaan C#-koodilla uusi GUID-tunniste, jota käytetään paikallisen kansion nimenä:

```
Guid newGuid = Guid.NewGuid();
string guidString = newGuid.ToString("N");
```

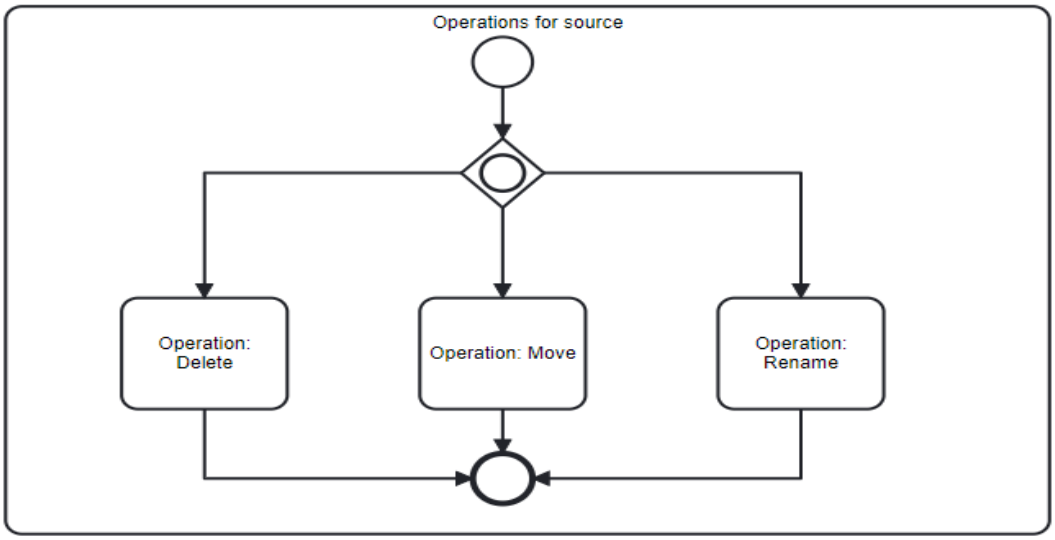
Koodin tuloksena on 32 merkin pituinen heksadesimaalinumerojono, joka koostuu GUID-tunnisteen numeroista ja kirjaimista ilman muita merkkejä. Tunnisteen perusteella luodaan uusi kansio, johon ladatut tiedostot väliaikaisesti tallennetaan. Kansio luodaan vain, mikäli lähdesijainnista on onnistuneesti haettu tiedostoja ja kohdesijantina on muu kuin paikallinen hakemisto.

Kun prosessi on onnistuneesti hakenut lähdesijainnista tiedostot, prosessi jatkaa suorittamaan tiedostojen siirtoa kohdesijaintiin. Tämä toiminnallisuus toteutetaan samalla tavalla kuin lähdesijainnista tapahtuva tiedostojen haku, eli inklusiivinen päätöksenteko ohjaa parametrien perusteella prosessia toimittamaan tiedostot halutulla protokollalla. Kuvassa 14 on esitetty, miltä prosessi näyttää yksinkertaistettuna lähetysvaiheen osalta.



Kuva 14. Kohdesijainnin lataustoiminnot yksinkertaistettuna.

Toinen merkittävä lisäominaisuus, joka toteutetaan low-code-toteutuksella, on lähdepalvelimella tapahtuvien operaatioiden hallinta. Operaatioilla tarkoitetaan tässä tapauksessa sitä, mitä tapahtuu tiedostolle onnistuneen siirron jälkeen. Vaihtoehtoja on tiedoston poisto, tiedoston siirto lähdepalvelimella toiseen kansioon, uudelleen nimeäminen tai ei mitään. Frendsin taskit tarjoavat tähän jälleen osittain työkalut, mutta niiden ympärille on rakennettava lisälogiikkaa, jotta ne vastaisivat täysin toteutuksen vaatimuksia. Esimerkiksi siirrettäessä tiedostoa SFTP-palvelimelta toiselle hyödyntäen pelkästään Frendsin omia taskeja, kuten SFTP Download ja SFTP Upload, voidaan kohdepalvelimen kanssa havaitun virheen sattuessa törmätä tilanteeseen, jossa lähdepalvelimen tiedostoille on suoritettu esimerkiksi poisto-operaatio. Tässä tapauksessa tiedostot olisi poistettu lähdepalvelimelta, mutta niitä ei olisi koskaan onnistuneesti siirretty kohdepalvelimelle. Ongelman kiertämiseksi aliprosessille rakennetaan mekanismi, joka hoitaa lähdepalvelimen operaatiot vasta kun integraatio on varmistanut, että tarvittavat tiedostot on siirretty kohdepalvelimelle. Tätä varten aliprosessille luodaan uusi käsittely, joka ensiksi vertaa lähdepalvelimelta onnistuneesti haettuja tiedostoja ja kohdepalvelimelle onnistuneesti siirrettyjä tiedostoja. Onnistuneesti siirretyt tiedostot saadaan yksinkertaisella koodilla, joka tarkastaa kahdesta listasta yhteiset elementit. Tämän jälkeen suoritus toteuttaa parametreina annetut lähdesijainnin operaatiot onnistuneesti siirretyille tiedostoille. Kuvassa 15 on kuvattu tätä vaihetta.



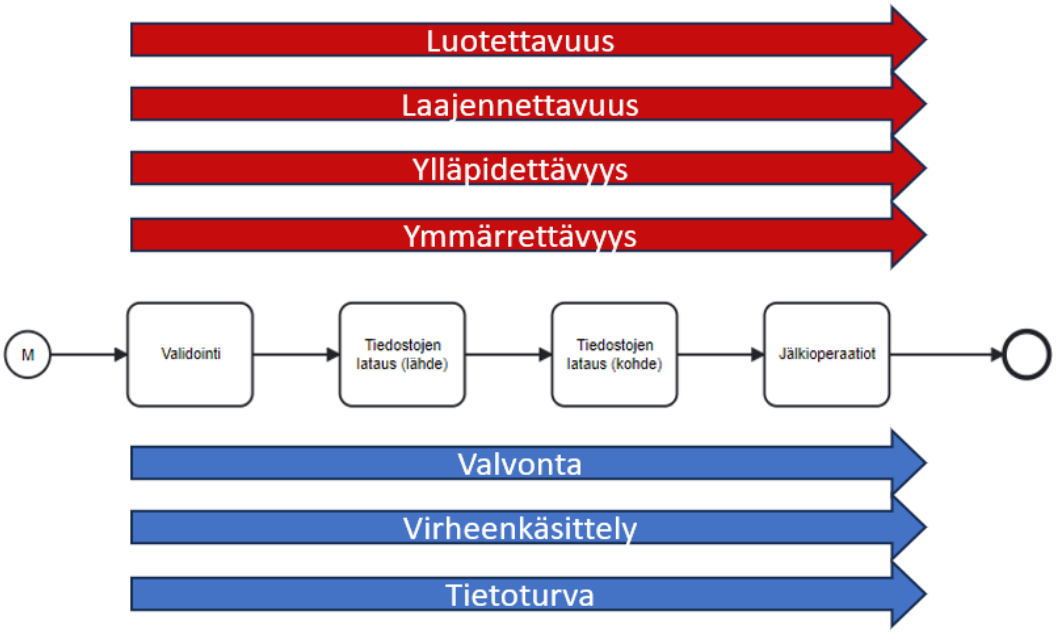
Kuva 15. Lähdesijainnin operaatiot yksinkertaistettuna.

Lopuksi suorituksen jälkeen välikäsittelyssä luotu välikansio sisältöineen poistetaan käyttäen Frendsin taskeja. Näin varmistetaan, että esimerkiksi ylimääräisiä ja tarpeettomia tiedostoja ei jää paikalliselle tietokoneelle talteen ja samalla huolehditaan esimerkiksi GDPR:n toteutumisesta. Tämän jälkeen aliprosessi tämän jälkeen palaa takaisin pääprosessin suoritukseen, ja pääprosessi suorittaa oman prosessinsa toiminnallisuuden loppuun.

6.3 Yhteenveto

Kuvassa 16 on vaatimuksiin perustuva koostettu kuvaus prosessia koskevista teemoista ja toiminnallisuuksista. Punaisella olevat nuolet kuvaavat prosessille asetettuja ei-toiminnallisia vaatimuksia, joita olivat ratkaisun käyttöön liitettävä luotettavuus, ratkaisun tulevaisuuden kannalta huomioon otettava laajennettavuus ja skaalautuvuus, ratkaisun helppo ylläpidettävyys ja sen ymmärrettävyys. Nämä ratkaisun vaatimukset saatiin katettua hyödyntämällä Frends-integraatioalustan tarjoamia low-code-työkaluja, kuten BPMN kaaviomerkintää ja alustalle valmiiksi rakennettuja konnektoreita ja adaptereita eli Frendsin

omia taskeja. Taskit eivät kuitenkaan pystyneet tarjoamaan täysin kelpaavia toiminnallisuuksia ratkaisulle minkä vuoksi rakennettiin lisätoiminnallisuuksia. Lisätoiminnallisuudet toteutettiin suurelta osin jälleen low-codella, käyttäen kuitenkin lisäksi C#-koodia, kuten esimerkiksi validointivaiheessa.



Kuva 16. Prosessin ei-toiminnalliset ja toiminnalliset vaatimukset.

Kuvassa 16 on sinisellä merkittyjä nuolia, jotka kuvastavat sellaisia teemoja, jotka sisältävät samalla ei-toiminnallisia ja toiminnallisia vaatimuksia. Tietoturvallisen integraatioprosessin toteutuksessa hyödynnettiin Frendsin omien taskien toiminnallisuuksia. Aliprosessille annetaan esimerkiksi parametrina SFTP-yhteyttä suojaava salausalgoritmi, joka huolehtii siitä, että tiedonsiirto palvelimen ja integraatioalustan välillä on turvallista ja etteivät ulkopuoliset pääse urkkimaan tai muuttamaan siirrettäviä tietoja. Tämän lisäksi aliprosessille on luonnollisesti mahdollista antaa palvelimella käytettävän autentikaatiomenetelmän mukaiset parametrit. Autentikaatiomenetelmät, eli palvelimen käyttämät menetelmät käyttäjän tunnistukseen, on esimerkiksi käyttäjätunnuksen ja salasanan yhdistelmä tai esimerkiksi käyttäjätunnuksen ja avainparin vaihtoon perustuva autentikaatio. Aliprosessille on mahdollista antaa parametrit "salaisuuksina", jolloin Frendsin käyttöliittymä piilottaa nämä arvot, varmistaen samalla, että arkaluonteiset tiedot säilyvät turvassa eivätkä ole näkyvissä tai helposti saatavilla. Toiminnallisuus kytkettiin päälle salasanoille ja muille

avaimena toimiville arkaluontoisille parametreille. Näillä valinnoilla saatiin katettua vaatimukset, jotka koskettivat tietoturvaa.

Valvottavuuteen liittyvät vaatimukset saatiin toteutettua luomalla aliprosessille oma lokitus, joka kerää aliprosessin tapahtumia, kuten parametrien validointituloksia, muuttujaan ja palauttaa sen aliprosessin tuloksena pääprosessille. Tämän lisäksi aliprosessille kytkettiin päälle toiminnallisuus, joka lähettää muuttujan sähköpostivirheilmoituksena alustaa ylläpitävälle taholle, mikäli aliprosessin suoritus päättyy virheeseen. Aliprosessin taskeille kytkettiin päälle asetus, jonka avulla esimerkiksi taskit, jotka päättyvät tilapäiseen yhteysongelmista johtuvaan virheeseen, yrittävät suoritusta uudelleen, mikä tuki ratkaisulle asetettuja virheenkäsittelyvaatimuksia.

7 LOPPUPÄÄTELMÄT

Opinnäytetyön tavoitteena oli tutkia ja määritellä uuden työkalun vaatimukset integraatioalustalle sekä selvittää, kuinka tämä työkalu voitaisiin toteuttaa parhaiten käytössä olevien teknologioiden avulla. Työssä käytiin läpi integraatiokomponentit, erilaisia integraatioarkkitehtuureja, Frendsin toiminnallisuuksia ja ohjelmistotuotannosta vaatimusmäärittelyä minkä perusteella uuden työkalun toiminnallisuudet ja tekniset vaatimukset määriteltiin. Vaatimukset kerättiin strukturoimattomilla haastatteluilla ja dokumenttien tutkimisella. Vaatimusten keruun jälkeen lähdettiin toteuttamaan varsinaista ratkaisua. Vaatimuksissa korostuivat teemat kuten luotettavuus, tietoturva, skaalautuvuus ja ratkaisun ymmärrettävyys. Nämä teemat ohjasivat ratkaisun toteuttamista Frendsin aliprosessille hyödyntäen mahdollisimman paljon low-code-mahdollisuuden tuomaa yksinkertaisuutta. Opinnäytetyö tarjoaa kattavan yleiskuvan aliprosessin toteutusprosessista, valinnoista ja niiden perusteluista sekä työkalun tuomasta lisäarvosta toimeksiantajalle. Työn tulokset osoittavat, että huolellisella vaatimusmäärittelyllä voidaan kehittää tehokas ja käyttäjäystävällinen työkalu, joka kuitenkin pystyy toteuttamaan monimutkaisiakin tiedonsiirto-operaatioita.

LÄHTEET

Asiakastieto. (2023). Yleiskuva: Integrata Oy. https://www.asiakastieto.fi/yritykset/fi/integrata-oy/22113129/yleiskuva

Alfame. (1.6.2023). Järjestelmäintegraatio–mitä se on selkokielellä. https://www.alfame.com/ajankohtaista/jarjestelmaintegraatio-mita-se-on-selkokielella

Collegenote. (n.d). Requirements Elicitation. Haettu 7.8.2023 osoitteesta https://www.collegenote.net/curriculum/software-engineering-csit/54/316/

Digia. (2023a). Viisi integraatiotrendiä vuodelle 2023. https://digia.com/blogi/viisi-integraatiotrendia-vuodelle-2023

Digia. (2023b). Ilman integraatiokykyä tekoälylle ja digitalisaatiolle voi heittää hyvästit–Nämä trendit muuttavat nyt it-järjestelmiä. https://www.tivi.fi/kumppanisisallot/digia/ilman-tata-tekoalyn-ja-digitalisaation-kehitys-pysahtyy-nama-nelja-integraatiotrendia-muuttavat-nyt-it-jarjestelmia/

Eiskonen, H & Stubin, T. (2024). Pilven kulut räjähtivät kasvuun–”kehittäjillä on vapaus, mutta mukana tulee vastuu kustannuksista". Tivi. https://www.tivi.fi/uutiset/pilven-kulut-rajahtivat-kasvuun-kehittajilla-on-vapaus-mutta-mukana-tulee-vastuu-kustannuksista/06581d9b-4fe0-4e0b-a35d-f8547a8c4cbd

Frends. (n.da). Orchestration. Haettu 11.11.2023 osoitteesta https://frends.com/platform/orchestration

Frends. (n.db). Low-Code Approach. Haettu 11.12.2023 osoitteesta https://frends.com/platform/low-code-approach

Frends. (n.dc). SFTP. Haettu 14.1.2024 osoitteesta https://tasks.frends.com/integrations/SFTP/

Frends. (2023). Introduction to Triggers in Frends. https://docs.frends.com/en/articles/8010526-introduction-to-triggers-in-frends

Galkin, O. (2024a). Process. https://docs.frends.com/en/articles/2236816-process

Galkin, O. (2024b). What are Tasks? https://docs.frends.com/en/articles/8010710-what-are-tasks

Galkin, O. (2024c). Custom Tasks. https://docs.frends.com/en/articles/2206746-custom-tasks

Galkin, O. (2024d). Low-Code Approach of Frends. https://docs.frends.com/en/articles/8232290-low-code-approach-of-frends

Gartner. (2022). What is Integration Platform as a Service? https://www.gartner.com/reviews/market/integration-platform-as-a-service-worldwide

Håkansson, L. (16.11.2021). Understanding the types of API to find the best one you need. Gravitee blog. https://www.gravitee.io/blog/understanding-the-types-of-api-to-find-the-best-one-you-need

IEEE. (2000). Recommended Practice for Architectural Description of Software-Intensive Systems. International Electrotechnical Commission/Institute of Electrical. https://ieeexplore.ieee.org/document/875998

IEEE Computer Society. (2014). Software Engineering Body of Knowledge-Guide V4 beta. https://www.computer.org/education/bodies-of-knowledge/software-engineering

ISO/IEC/IEEEE 29148:2018. (2018). Systems and software engineering: Life cycle processes: Requirements engineering. International Organization for Standardization/International Electrotechnical Commission/Institute of Electrical and Electronics Engineers.

JavaTPoint. (2005). GraphQL. https://www.javatpoint.com/graphql

JUHTA. (2018a). JHS: 173 ICT-palvelujen kehittäminen: Vaatimusmäärittely. https://www.suomidigi.fi/sites/default/files/2020-06/JHS173.doc

JUHTA. (2018b). JHS: 173 ICT-palvelujen kehittäminen: Vaatimusmäärittely-Liite 1: Käyttötapauslomake-mallipohja. https://www.suomidigi.fi/sites/default/files/2020-06/JHS173_liite1.doc

JUHTA. (2018c). JHS: 173 ICT-palvelujen kehittäminen: Vaatimusmäärittely-Liite 2: Vaatimusluettelo-mallipohja. https://www.suomidigi.fi/sites/default/files/2020-06/JHS173_liite2.xls

Kong. (2021). Types of APIs and Use Cases. https://konghq.com/learning-center/api-management/different-api-types-and-use-cases

Lane, K. (28.6.2023). What Is a SOAP API? Postman blog. https://blog.postman.com/soap-api-definition/

LinkedIn. (2023). Software Integration: How do you decide between point-to-point and hub-and-spoke integration patterns? https://www.linkedin.com/advice/3/how-do-you-decide-between-point-to-point-hub-and-spoke

Mason (6.8.2011). What is an ESB? –ESB or not to ESB Revisited Part 1. Mulesoft Blog. https://blogs.mulesoft.com/dev-guides/how-to-tutorials/esb-or-not-to-esb-revisited-part-1/

Medium. (2023). Web API, What is it? https://medium.com/codex/web-api-what-is-it-4f9d46b428f9

Newtonsoft. (2023). Json.NET. https://www.newtonsoft.com/json.

Ryhänen, J. (2019). Johdanto integraatioihin ja iPaaS:iin. https://www.devisioona.fi/2019/08/johdanto-integraatioihin-ja-ipaasiin/

snapLogic. (13.1.2023). Point-to-Point Integration: Advantages, Disadvantages, and Uses. https://www.snaplogic.com/blog/point-to-point-integration-advantages-disadvantages-and-uses

TechTarget. (2021). open API (public API). https://www.techtarget.com/searchapparchitecture/definition/open-API-public-API

Tietosuoja. (2020). Usein kysyttyä EU:n tietosuoja-asetuksesta: Mitä tietosuojaperiaatteella tarkoitetaan? https://tietosuoja.fi/gdpr

Toivanen, A. (2021). Application Integration and Integration Platforms 101. https://frends.com/article/application-integration-and-integration-platforms-101

Tozzi, C. (2021). A practical take on low-code vs. traditional development. https://www.techtarget.com/searchsoftwarequality/tip/A-practical-take-on-low-code-vs-traditional-development

Virtanen, R. (2024a). Basics of Business Process Modelling Notation (BPMN). https://docs.frends.com/en/articles/8010481-basics-of-business-process-modelling-notation-bpmn

Workato. (2023). 5 reasons why point-to-point integration falls short of your business needs. https://www.workato.com/the-connector/point-to-point-integration-drawbacks

LIITE 1: VAATIMUSLUETTELO

LIITE 1: Vaatimusluettelo on poistettu opinnäytetyön julkisesta versiosta. Liite sisältää liiketoiminnan kannalta sellaista tietoa, jonka julkistaminen ei ole toimeksiantajan etujen mukaista.

LIITE 2: KÄYTTÖTAPAUSLOMAKE

LIITE 2: Käyttötapauslomake on poistettu opinnäytetyön julkisesta versiosta. Liite sisältää liiketoiminnan kannalta sellaista tietoa, jonka julkistaminen ei ole toimeksiantajan etujen mukaista.

LIITE 3: ALIPROSESSIN PARAMETRIT

Key	DefaultValue	Description	Secret
SourceType	SFTP	Specifies the type of the source, which can be SFT	FALSE
SourceDirectory	/	Specifies the directory in the source where the file	FALSE
SourceFileName	test_*.txt	Indicates the name of the file in the source directo	FALSE
SourceServerAddress	sftp.server.com	Specifies the server address for remote sources.	FALSE
SourceServerPort	22	Defines the port number to connect to the source	FALSE
SourceAuthentication	UsernamePassword	Specifies the authentication method for the sourc	FALSE
SourceUsername	sftp_test	Provides the username for authentication at the s	FALSE
SourcePassword		Contains the password for authentication at the sc	TRUE
SourcePrivateKeyPassphrase		Contains the password for authentication at the de	TRUE
SourcePrivateKeyFile		Specifies the path to the private key file for authe	FALSE
SourceHostKeyAlgorithm	Any	Specifies the algorithm used for host key verificati	FALSE
SourceAction	Info	Describes the action to be performed at the source	FALSE
SourceOperation	Nothing	Defines the general operation to be performed at	FALSE
SourceOperationRename	test_moved.txt	Specifies a renaming operation at the source.	FALSE
SourceOperationMoveTo	/moved/	Specifies a move operation to a different location	FALSE
DestinationType	SFTP	Specifies the type of the destination, which can be	FALSE
DestinationDirectory	/	Specifies the directory in the destination where th	FALSE
DestinationFileName		Defines the name of the file at the destination.	FALSE
DestinationServerAddress	22	Specifies the server address for remote destinatio	FALSE
DestinationServerPort	sftp.server.com	Defines the port number to connect to the destina	FALSE
DestinationAuthentication	UsernamePassword	Specifies the authentication method for the destir	FALSE
DestinationUsername	sftp_test	Provides the username for authentication at the d	FALSE
DestinationPassword		Contains the password for authentication at the de	TRUE
DestinationPrivateKeyFile		Specifies the path to the private key file for authe	FALSE
DestinationPrivateKeyPassphrase		Contains the passphrase for the private key (if app	TRUE
DestinationHostKeyAlgorithm	Any	Specifies the algorithm used for host key verificati	FALSE
DestinationAction	Nothing	Describes the action to be performed at the destir	FALSE