OULUN AMMATTIKORKEAKOULU

Miikka Moilanen

Frends-integraatioalusta

Frends-integraatioalusta

Miikka Moilanen
Opinnäytetyö
Syksy 2024
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Miikka Moilanen
Opinnäytetyön nimi: Frends-integraatioalusta
Työn ohjaaja: Teemu Korpela
Työn valmistumislukukausi ja -vuosi: Syksy 2024 Sivumäärä: 31

Tämä opinnäytetyön tarkoituksena oli esitellä Frends-integraatioalustaa, tarkastella sen ominaisuuksia ja hyödyntämismahdollisuuksia integraatioiden toteutukseen. Työssä tarkasteltiin myös lyhyesti järjestelmäintegraatiota käsitteenä, pohdittiin integraatioiden hyötyjä sekä käyttötarkoitusta ohjelmointialalla.

Frends-integraatioalustan arkkitehtuuriin ja komponentteihin tutustuttiin dokumentaation kautta. Dokumentaatiosta esiteltiin tärkeimmät elementit Frends-integraatioiden ja -integraatioympäristön luomisen kannalta.

Opinnäytetyön käytännön osiossa alustan käyttöä ja integraatioiden kehitystä tarkasteltiin esimerkkien avulla. Vaatimusmäärittelyn pohjalta luotiin toimiva integraatioprosessi sekä tarvittavat tehtäväkomponentit ja aliprosessit integraation tarpeisiin. Lisäksi käytännön osiossa tutustuttiin API-hallinnan ominaisuuksiin ja kehitettiin API-rajapinta.

Asiasanat: Järjestelmäintegraatio, API-hallinta, pilvipalvelu, Frends

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

Author: Miikka Moilanen
Title of thesis: Frends integration platform
Supervisor: Teemu Korpela
Term and year when the thesis was submitted: Fall 2024
Number of pages: 31

The purpose of this thesis was to introduce the Frends integration platform, examine its features, and explore its potential for implementing integrations. The thesis briefly discusses system integration as a concept, explores the benefits of integrations, and their use in the programming field.

The architecture and components of the Frends integration platform were studied through documentation. The key elements necessary for creating Frends integrations and Frends platform setup were introduced.

In the practical part of the thesis, the use of the platform and the development of integrations were explored through examples. Based on the requirements specification, a functional integration process was created, along with the necessary task components and subprocesses to meet the integration needs. Additionally, the practical section examined API management features and involved the development of an API interface.

Keywords: System integration, API management, cloud service, Frends

Lyhenteet ja käsitteet

API:	Application Programming Interface. Ohjelmointirajapinta.
JSON:	JavaScript Object Notation. Yksinkertainen ja kevyt, ihmisen helposti luettava avoimen standardin tiedostomuoto tiedonvälitykseen ja tallennukseen
Low code:	Low code -ohjelmistokehityksellä tarkoitetaan kehitystä, jossa ohjelmistoja kehitetään valmiita komponentteja käyttäen lähes täysin ilman koodia
NuGet:	Avoimen lähdekoodin pakettienhallintatyökalu Microsoftin kehitysympäristöön, jota käytetään kirjastojen ja muiden ohjelmistokomponenttien jakeluun ja hallintaan .NET-projekteissa.
OpenApi:	Rajapintakuvaukseen kehitetty standardi. Aiemmalta nimeltään Swagger-spesifikaatio.
Swagger:	Avoimen lähdekoodin työkalukokoelma, jota käytetään REST-rajapintojen (API) kuvaamiseen, dokumentointiin, testaamiseen ja kehittämiseen. Käytetyimpiä työkaluja ovat Swagger Editor ja Swagger UI.
Swagger UI:	Graafinen käyttöliittymä, joka renderöi OpenAPI-kuvauksen käyttäjäystävälliseksi dokumentaatioksi. Swagger UI mahdollistaa rajapintojen testauksen suoraan selaimessa.

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön tavoitteena on toimia yleiskatsauksena Frends-integraatioalustan ominaisuuksiin. Integraatioalustat ovat yhä useammin olennainen osa yritysten liiketoimintaa ja nykyaikaista ohjelmistokehitystä, tarjoten pilvipalveluna monipuolisia ratkaisuja eri järjestelmien ja sovellusten yhteensovittamiseen. Frendsin tapaiset low code -periaatteella operoitavat integraatioratkaisut tulevat olemaan houkutteleva vaihtoehto, kun yritykset suunnittelevat omia tietoliikennejärjestelmiään.

Opinnäytetyön alussa tullaan esittelemään lyhyesti järjestelmäintegraatiota käsitteenä sekä tarkastelemaan integraatioratkaisujen hyötyjä ja haasteita tietoliikennearkkitehtuurin ja liiketoiminnan kannalta. Opinnäytetyön varsinainen paino tulee kuitenkin olemaan Frends-integraatioalustan tarkastelussa. Työn toisessa osassa tehdään katsaus integraatioalustan dokumentaatioon ja tullaan esittelemään Frendsin käsitteistöä, arkkitehtuuria ja toimintoja. Kolmannessa, käytännön osiossa tullaan kehittämään integraatioratkaisu Frends-portaalin avulla ja kuvataan kehitysprosessi vaihe vaiheelta. Tämän lisäksi tarkastellaan Frendsin API-ominaisuuksien käyttöä.

Työn lopputuloksena tavoitellaan sopivan kattavaa dokumenttia Frends-integraatioalustasta, jonka avulla asiaan perehtymätön henkilö voisi saada peruskäsityksen työkalun ominaisuuksista ja mahdollisuuksista, jopa käyttää työkalua.

Tämän opinnäytetyön tekemiseen on käytetty työnantajani asiakkaan tarjoamaa Frends-lisenssiä, jota käytän myös päivittäisessä työssäni. Lisenssin käyttöön tähän tarkoitukseen on pyydetty ja saatu lupa asiakkaalta. Työnantaja tai asiakas eivät kuitenkaan ole opinnäytetyön tilaajia, joten en tule nimeämään niitä työssäni. Tästä syystä myös työssä käytetyt kuvakaappaukset voivat olla hieman rajoitettuja ja joitakin tietoja on voitu muuttaa tai poistaa kuvakaappauksista. Opinnäytetyötä varten on käytetty Frendsistä versiota 5.6.

2 JÄRJESTELMÄINTEGRAATIO

2.1 Järjestelmäintegraatioista yleisesti

Järjestelmäintegraatiolla tarkoitetaan tiivistetysti tapoja ja tekniikoita, joiden avulla tietojärjestelmät saadaan kommunikoimaan keskenään. Suppean määritelmän mukaan integraatio on ratkaisu, jolla muutoin keskenään yhteensopimattomat järjestelmät liitetään toisiinsa, jolloin niiden välillä voidaan siirtää tietoa. Hieman laveammin ajateltuna järjestelmäintegraatio ei koostu pelkästään teknisistä ratkaisuista tai yksittäisistä teknologioista, vaan on ajattelutapa, jonka avulla voidaan pyrkiä hahmottamaan yrityksen tietoteknistä arkkitehtuuria. Onnistuneen järjestelmäintegraation avulla voidaan tehostaa merkittävästi yrityksen toimintaa, kontrolloida ja monitoroida informaation siirtoa ja muunnoksia sekä kehittää liiketoimintaa. [1, s. 13–14, 48.]

Järjestelmäintegraatioiden hyödyt liittyvät kiinteästi automaatioon: kun kaksi tai useampi järjestelmä on integroitu kommunikoimaan keskenään, säästyy aikaa ja resursseja, kun samaa tietoa ei tarvitse kirjata erikseen kahteen tai useaan eri järjestelmään [2]. Hyvin suunniteltu järjestelmäintegraatio parantaa lisäksi yrityksen joustavuutta, koska integraation avulla voidaan vähentää riippuvuutta yksittäisistä ohjelmisto- ja ratkaisutoimittajista, reagoida nopeisiin muutoksiin organisaatiossa ja sen toiminnassa sekä tehdä nopeammin muutoksia ja olemassa oleviin prosesseihin [1, s. 27]. Järjestelmäintegraatio voi myös toimia tehokkaana raportoinnin ja monitoroinnin työkaluna. Koska integraatio siirtää ja jalostaa tietoa usean tietojärjestelmän välillä, on se looginen paikka kerätä ja jäsennellä järjestelmien käsittelemää dataa, jolloin saadaan hyödyllistä ja ajantasaista tietoa yrityksen toiminnasta ja taloudesta. [1, s. 31–32.]

Integraatiot eivät kuitenkaan välttämättä ole riskittömiä tai helposti toteutettavissa. Integraatioratkaisun kehittäminen voi olla pitkä prosessi, joka etenkin alkuvaiheessa voi aiheuttaa suuriakin kustannuksia. Lisäksi yrityksen tietojärjestelmät muuttuvat ajan saatossa, jolloin myös integraatioiden tulee muuttua [1, s. 43]. Vaikka yritykset ovat monesti valmiita käyttämään paljonkin resursseja yksittäisten järjestelmien kehittämiseen, integraatioratkaisuille ei välttämättä osata varata tarpeeksi budjettia [3]. Teknisiä haasteita aiheuttavat usein yhteensopivuusongelmat. Koska eri järjestelmät on kehitetty erilaisilla teknologioilla, koodikielillä tai vaikkapa saman teknologian eri versioilla, voi niiden integroiminen aiheuttaa ylimääräistä työtä, kun tulee tarve kehittää erikoistuneita adaptereita

järjestelmien välille [3]. Järjestelmäintegraatiot voivat myös luoda uusia tietoturvariskejä. Haavoittuvuus yhdessä järjestelmässä voi avata oven kyberhyökkäyksille ja tietomurroille myös toiseen, integraation kautta linkittyvään järjestelmään. Esimerkki huonosta tietoturvasta integraatioratkaisussa oli Target-vähittäiskauppaketjun tietomurto vuonna 2013. Hyökkääjä sai pääsyn Targetin järjestelmään Targetin ja kolmannen osapuolen välisen heikon integraatioturvan kautta, joka johti miljoonien asiakkaiden maksukorttitietojen vuotamiseen. [4]

2.2 iPaaS – integraatioalusta palveluna

iPaaS on lyhenne sanoista Integration Platform as a Service, suomeksi integraatioalusta palveluna. iPaaS tarjoaa organisaatioille alustan, joka mahdollistaa erilaisten sovellusten, tietokantojen ja järjestelmien integroinnin toisiinsa pilvessä. Tämä auttaa yrityksiä yhdistämään erilaisia tietolähteitä ja sovelluksia saumattomasti ja tehokkaasti, mikä parantaa liiketoiminnan suorituskykyä ja tehokkuutta.

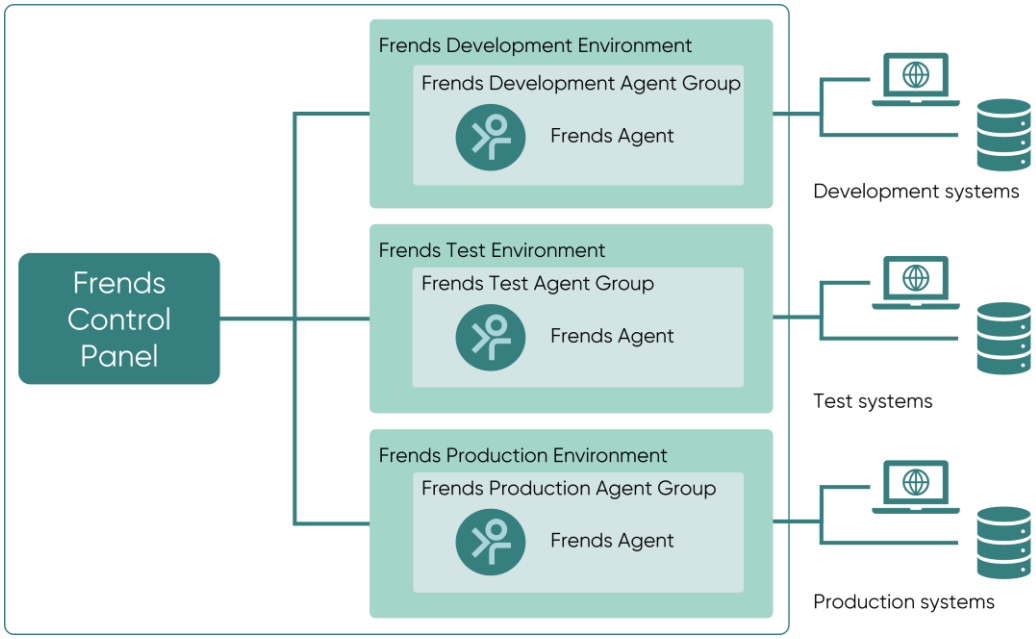
iPaaS on vastaus kasvavaan määrään yritysten käyttämiä SaaS-palveluita. SaaS on lyhenne sanoista Software as a Service, suomeksi ohjelmisto palveluna. SaaS-ohjelmat ovat monesti houkuttelevia yrityksille, koska ne on pääosin kehitetty helpoiksi ottaa käyttöön ja tarjoavat valmiin ratkaisun johonkin tietojenkäsittelyyn liittyvään tarpeeseen. Helppouden takia käytettävien SaaS-ohjelmien määrä voi kuitenkin äkkiä kasvaa jyrkästi: modernit suuryritykset voivat käyttää jopa 470 SaaS-sovellusta, mikä voi johtaa monimutkaisiin ja hallitsemattomiin sovellusympäristöihin. iPaaS-tuotteet yrittävät tarjota ratkaisuja ongelmiin, jota kasvava määrä eri ohjelmistoja ja dataa aiheuttaa. Ne tarjoavat yritysten käyttöön valmiiksi kehitettyjä sovittimia ja datankäsittelykomponentteja, joilla eri järjestelmät on helpompi integroida toisiinsa ilman, että tarvitsee kehittää kustomoituja ratkaisuja. Useat iPaaS-ratkaisut tarjoavat low code -periaatteen mukaisia kehitysympäristöjä, joiden avulla myös henkilön, jolla ei ole ohjelmointiosaamista on mahdollista kehittää ja hallita integraatioita. Lisäksi iPaaS-palvelut sisältävät työkaluja datan reaaliaikaiseen monitorointiin ja seurantaan kootusti, jolloin liiketoiminnan datavirtaa voi ainakin teoriassa seurata yhdestä lähteestä. [5]

3 FRENDS

Frends on Frends Technology Oy:n omistama integraatioalusta. Integraatioita kehitetään ja hallitaan graafisen käyttöliittymän kautta BPMN-standardin (Business Process Model and Notation) mukaisilla prosessikaavioilla [6]. Yksittäinen prosessi koostuu tehtäväkomponenteista, jotka liitetään toisiinsa visuaalisen työkalun avulla, jolloin muodostuu vuokaavio, joka kuvaa yksittäistä integraatioprosessia. Frends pyrkii siihen, että sen avulla käyttäjä voi kehittää integraatioita minimaalisella määrällä varsinaista koodinkirjoitusta, niin sanotun low code -ohjelmistokehityksen periaatteen mukaisesti [7].

3.1 Frends-arkkitehtuuri

Frendsin arkkitehtuuri perustuu niin sanottuun hajautettuun arkkitehtuurimalliin. Kuvassa 1 esitelty hajautettu malli tarkoittaa Frendsin tapauksessa sitä, että ajoympäristöt, jotka suorittavat integraatioprosessit voidaan asentaa lukuisiin erilaisiin, joko pilvessä sijaitseviin tai paikallisiin ympäristöihin. Frendsissä näitä ajoympäristöjä kutsutaan agenteiksi. Integraatioiden kehitys ja hallinta taas on keskitetty yhteen hallintaportaaliin, jota käytetään käyttöliittymän avulla julkisessa verkossa. [8]



Kuva 1 Frends-arkkitehtuuri [9]

3.2 Agentit ja agenttiryhmät

Käyttöliittymällä tehdyt integraatiot pakataan ja toimitetaan ajoympäristöihin, joita kutsutaan agenteiksi (*Agent*). Agentit suorittavat integraatioita määriteltyjen laukaisusääntöjen mukaan ja raportoivat suoritustilastot ja muun statistiikan takaisin hallintaportaaliin.

Agenttien toiminta voidaan jakaa kolmeen eri loogiseen kerrokseen. Ohjauskerros kommunikoi hallintaportaalin kanssa Azure Service Bus -jonojen avulla. Näiden jonojen kautta hallintaportaali päivittää agentin konfiguraatiota sekä välittää tietoa integraatioista sekä niiden laukaisusäännöistä. Toiseen suuntaan, agentilta portaalille liikkuu ohjauskerroksen kautta tietoa integraatioiden suorituksesta. Toinen kerros on liipaisukerros, joka vastaa integraatioiden käynnistymisestä. Integraatio voidaan määritellä käynnistymään esimerkiksi ajastetusti, reaktiona jonkin toisen integraatioprosessin suoritukseen tai kutsusta API-rajapintaan. Kolmas kerros on toteutuskerros, jossa varsinainen integraatioprosessi suoritetaan liipaisukerroksen aloittamana. [10]

Agentit ryhmitellään agenttiryhmiin (*Agent group*). Agenttiryhmä on itsenäinen kokonaisuus, johon kuuluvat agentit jakavat samat asetukset ja tiedon ryhmään toimitetuista integraatioista. Ryhmä voi koostua pelkästään yhdestä agentista, mutta monen agentin kokoonpano takaa korkeamman saatavuuden, koska kaikilla ryhmän agenteilla on tieto järjestelmän tilasta, jolloin agentit voivat jakaa kuormaa toisille ryhmän agenteille. Monen agentin ryhmä mahdollistaa myös sen, että integraatioprosessi voidaan suorittaa, vaikka yksi agentti menisikin vikatilaan esimerkiksi palvelinvian takia. [11]

Frends-ympäristö voidaan asentaa myös täysin paikallisesti, jolloin integraatioita voidaan suorittaa sisäisessä verkossa. Tässäkin tapauksessa tosin prosessilokien selaus ja integraatiokonfiguraatio tapahtuu Frends käyttöliittymät kauitta, joten Azure Service Bus -yhteys vaaditaan. Paikallinen asennus voidaan tehdä Windows-käyttöjärjestelmälle, Windows Server -palvelimelle tai Ubuntu Server -palvelimelle. [12]

3.3 Prosessit ja aliprosessit

Frendsissä käytössä on kahdenlaisia prosesseja: tavalliset prosessit (*Process*) ja aliprosessit (*Subprocess*). Tavalliset prosessit - joihin viitataan jatkossa vain prosesseina - ovat komponentteja,

joissa määritellään ja kuvataan visuaalisesti vuokaavion avulla varsinaisen integraatioprosessin suoritus. Aliprosessit taas ovat prosessien kaltaisia, mutta yleensä yksinkertaisempia, pienemmän toiminnallisuuden suorittavia komponentteja, joita voidaan kutsua prosessien sisältä. Aliprosessien tarkoitus on toimia uudelleenkäytettävinä komponentteina usean prosessin käyttöön. Aliprosessi voi myös kutsua toista aliprosessia.

Jokainen prosessi ja aliprosessi voidaan jakaa kolmeen osaan: aloituskohta eli triggeri, toiminnallinen osa ja loppupiste [13]. Valittu aloituskohta määrittää, miten integraatioprosessi käynnistyy. Prosessi voidaan valita käynnistettäväksi esimerkiksi ajastetusti, kutsusta API-rajapintaan, kun tiedosto lisätään kansioon tai manuaalisesti portaalin kautta. Aloituskohdaksi voidaan valita myös aliprosessin suoritus tietyin aikavälein, jolloin varsinainen prosessi käynnistyy vain, jos aliprosessi palauttaa jotain muuta kuin tyhjän arvon. [14]

Prosessin aloituskohdan ja loppupisteen välillä tapahtuu integraatioprosessin varsinainen toiminnallisuus. Toiminnallisuuden rakentamisessa voidaan hyödyntää valmiita tehtäväkomponentteja (*Tasks*), lyhyitä koodinpätkiä, toistorakenteita, aliprosesseja, ympäristömuuttujia sekä prosessin sisäisiä muuttujia. Toiminnallisen osan komponentteja voidaan ryhmitellä Scope-elementin avulla [15]. Scope-elementit toimivat samoin kuin C#-kielessä aaltosulkeiden sisään kirjoitetut koodiblokit, jolloin esimerkiksi niiden sisällä määritellyt muuttujat eivät näy elementin ulkopuolelle.

Loppupisteellä merkataan prosessisuorituksen loppu. Loppupiste on palautuselementti (*Return*), jossa määritellään prosessin palautusarvo. Palautusarvo on dynaamisesti tyypitetty. Palautus voi olla myös HTTP-vastaus, jos prosessin aloitus tapahtuu HTTP-triggerillä tai kutsulla API-rajapintaan. [16]

Sekä prosessien että aliprosessien luonti ja kehitys tapahtuu aina kehitysympäristössä. Kun prosessi on valmis testattavaksi, viedään se kehitysympäristöstä esimerkiksi testiympäristöön, jossa prosessia voidaan testata. Jos prosessi kutsuu aliprosesseja, tulee myös nämä olla vietynä samaan ympäristöön ennen kuin prosessi voidaan käynnistää.

3.4 API-hallinta

Frends tukee OpenApi-spesifikaation mukaisia rajapintamäärittelyjä [17]. Käyttäjä voi luoda API-rajapinnan portaalissa tai tuoda valmiin määrittelyn JSON-muodossa. Rajapintoja hallitaan ja tarkastellaan käyttöliittymän API-valikon kautta. OpenApi-dokumentissa määritellään rajapinnan parametrit, polku sekä palautusarvot. API-rajapintaan liitetään prosessi, jolla on aloituskohtana API-triggeri. Kun API-rajapinnalle luodaan uusi prosessi, Frends osaa luoda prosessiin valmiiksi triggerin, joka ottaa sisäänsä OpenApi-määrityksen mukaiset parametrit, sekä palautuselementin tai -elementit. Liitettyyn prosessiin rakennetaan triggerin sekä loppukohdan väliin toiminnallisuus samoin kuin muillakin tavoilla käynnistettyihin prosesseihin. [18]

Kuten prosessien kehitys, myös API-rajapintojen kehitys tapahtuu aina kehitysympäristössä. Valmis API-määritys, johon on liitetty prosessi, voidaan viedä kehitysympäristöstä toisiin ympäristöihin *Deploy*-painikkeen avulla. Rajapintaan kytketty prosessi viedään tässä yhteydessä myös samaan ympäristöön määrityksen mukana. API-määrityksiä voidaan tarkastella ja rajapintoja testata Swagger UI:n avulla [19]. Käyttöliittymällä Swagger UI avataan painamalla halutun API-määrityksen ollessa valittuna *Api Specification UI* -painiketta. Kutsut rajapintaan voidaan todentaa API-avainten (*API keys*) avulla. API-avaimet ovat aina ympäristökohtaisia. Avaimeen sidotut rajapinnat määritellään käyttöliittymän *Administration – API Keys* -valikossa. [20]

3.5 Prosessien suorittaminen eri ympäristöissä

Agenttiryhmät jaetaan integraatioprosessin elinkaaren eri osa-alueiden mukaisiin loogisiin osioihin, joita kutsuu ympäristöiksi (*Environment*). Ympäristöt koostuvat joko yhdestä tai useasta agenttiryhmästä. Tyypillinen Frends-kokoonpano koostuu kolmesta ympäristöstä: kehitys-, testi- ja tuotantoympäristö. Tässä kokoonpanossa integraatiokehitys tapahtuu kehitysympäristössä. Kun kehitettyä toteutusta halutaan testata, viedään se kehityksestä testiympäristöön, jossa integraatioprosessia voidaan ajaa ja testata sen toimintaa. Sitten, kun toteutusta on testattu ja se on todettu tuotantoon kelpaavaksi, julkaistaan integraatioprosessi tuotantoympäristöön. [21] Kolmen tyypillisimmän ym-

päristön lisäksi kokoonpanoon voi kuulua myös niin sanottu staging- tai quali-ympäristö, jossa integraatioita voidaan testata tuotantoympäristöä mahdollisimman hyvin jäljittelevillä asetuksilla ja tuotantoa vastaavalla datalla.

Osa Frends-portaalin asetuksista on ympäristökohtaisia. Näistä ehkäpä tärkeimpiä kokonaisuuden kannalta ovat ympäristömuuttujat (*Environment Variables*), joiden arvot asetetaan ympäristökohtaisiksi. Integraatioprosessit voivat kutsua ympäristömuuttujia, jolloin niiden arvo määrittyy kyseisen ympäristön mukaan, jossa prosessi suoritetaan. Ympäristömuuttujia voidaan myös jakaa ryhmiin esimerkiksi siten, että yhteen ryhmään kuuluu yhden integraatioprosessin kutsumat ympäristömuuttujat. Myös prosessisuoritusten lokiasetukset, valvontasäännöt ja API-rajapintoihin tunnistautumiseen käytettävät API-avaimet määritellään ympäristökohtaisesti. Lisäksi Frends-portaalin käyttäjille voidaan antaa ympäristökohtaisia oikeuksia, joiden mukaan yksittäisten käyttäjien pääsyä ympäristöihin voidaan rajoittaa. [21]

3.6 Tehtäväkomponentit integraatioiden apuna

Frendsin Taskit ovat .Net-ohjelmistokehyksen päälle C#-kielellä kirjoitettuja tehtäväkomponentteja, joiden tehtävä on suorittaa jokin yksinkertainen ja yksittäinen toimenpide, kuten esimerkiksi HTTP-kutsu, tiedoston lukeminen tai tiedostoon kirjoitus. Tehtäväkomponentille annetaan usein parametreja, jotka vaikuttavat sen suoritukseen sekä palautusarvoon. Tehtäväkomponentti on käytännössä staattinen metodi, josta on käännetty DLL-tiedosto (Dynamic Link Library) joka on edelleen pakattu NuGet-paketiksi. [22]

Frends tarjoaa kokoelman virallisia *Taskeja* [23]. Käyttöliittymän kautta lähes jokaiseen viralliseen komponenttiin on liitetty linkki lähdekoodiin, jonka kautta komponentin toimintaa pääsee tarkastelemaan myös kooditasolla. Virallisten komponenttien lisäksi käytössä on yhteisön Github-versionhallintajärjestelmään lataamia, avoimen lähdekoodin komponentteja [24].

Jos virallisista tai yhteisön kehittämistä *Taskeista* ei löydy tarkoitukseen sopivaa, voi käyttäjä myös ohjelmoida uuden [25]. Frends tarjoaa valmiin mallin alustan hyväksymän .Net-luokkakirjaston luomiseen [26]. Malliin määritellään komponentit sisään ottamat parametrit, staattiset metodit,

joissa logiikka suoritetaan. Metodien tulee olla aina staattisia, sekä niiden tulee aina palauttaa jotakin. Kun uusi tehtäväkomponentti on ohjelmoitu, se pakataan NuGet-paketiksi ja ladataan Frendsiin käyttöliittymän *Tasks*-valikon kautta. Ladattu komponentti on tämän jälkeen kaikkien Frends-prosessien käytettävissä.

4 INTEGRAATIOPUTKEN SUUNNITTELU JA TOTEUTTAMINEN FRENDS-ALUSTALLA

Seuraavaksi esiteltävä integraatioprosessi kehitettiin tarpeeseen, jossa järjestelmän oli saatava tietoa toisen järjestelmän tapahtumista. Datan lähde oli mikropalvelu, jonka rajapinnat vastaanottivat kutsuja käyttöliittymällä. Mikropalvelu tallensi tietokantaan tietoa näiden kutsujen perusteella. Tämä tieto oli saatava kohdejärjestelmän käyttöön integraation avulla. Kohdejärjestelmä oli toinen mikropalvelu, joka käsittelee ja tallentaa dataa omien tarpeidensa mukaisesti ja toimittaa sitä eteenpäin seuraavien järjestelmien käyttöön.

4.1 Integraatioprosessin suunnittelu

Integraatiototeutuksen suunnittelu lähti liikkeelle liiketoimintasuunnitelmaan tutustumisesta. Vaatimusmäärittelyn mukaan dataa piti saada siirrettyä kahden toisistaan riippumattoman mikropalvelun välillä ja datansiirron tuli tapahtua ajastetusti tunnin välein. Myös tähän integraatioon liittyvien järjestelmien ulkopuolella oli tarvetta lähdedatalle, joten jo aiemmin oli kehitetty pub/sub-toteutus, jonka kautta datan lähteenä toimiva mikropalvelu tarjosi dataa muiden järjestelmien käyttöön. Pub/sub-malli (engl. *publish*/*subscribe*, *pubsub*) itsessään on jo eräänlainen integraatiotyökalu, jossa järjestelmä julkaisee viestejä välittäjäpalvelulle. Dataa kuluttava järjestelmä taas tilaa viestejä välittäjältä. Tähän tarkoitukseen oli valittu käytettäväksi palveluksi Googlen tarjoama Cloud Pub/Sub. Googlen pub/sub-resurssiin oli luotu aihe (*Topic*), jonne mikropalvelu julkaisi viestejä. Jo suunnitteluvaiheessa luotiin Google Cloud -käyttöliittymän avulla tähän aiheeseen uusi tilaus, jota integraatioprosessi tulisi käyttämään. Tilaustyypiksi valikoitui niin sanottu pull-tilaus. Tässä tilaustyypissä tilaaja pyytää viestejä välittäjäpalvelulta. Toinen suosittu tilaustyyppi on push, jossa välittäjäpalvelu lähettää ilman pyyntöä aiheeseen saapuvat uudet viestit tilaajille. Push-tilausta harkittiin aluksi, mutta dokumentaatioon perehtymisen jälkeen se koettiin ongelmalliseksi. Jotta push-tyyppinen ratkaisu olisi ollut mahdollinen, olisi Frendsiin pitänyt luoda rajapinta, johon Google Cloud lähettää viestejä. Toistaiseksi ainoa tapa autentikoida kutsuja Frendsin rajapintoihin on kutsun otsakkeisiin lisätty API-avain. Google Cloud taas ei tue API-avaimia push-viesteihin, vaan ainoa valittavissa oleva autentikointitapa on JWT (engl. *JSON Web Token*).

Koska määrittely edellytti säännöllisesti toistuvaa integraatiota, jonka edellytyksenä on, että pub/subista löytyy uusia viestejä, valikoitui suunnitteluvaiheessa prosessin triggeriksi ehdollinen triggeri. Frendsin ehdollinen triggeri (*Conditional trigger*) on prosessin aloituskohta, josta edetään vain, jos ennalta määritellyt ehdot täyttyvät [27]. Ehdolliselle triggerille annetaan parametreina jokin suoritettava aliprosessi sekä aika-arvo sekunteina, jonka välein aliprosessia kutsutaan. Jos aliprosessi palauttaa mitä tahansa muuta kuin tyhjän arvon, edetään pääprosessissa varsinaiseen suorittavaan vaiheeseen.

Vaatimusmäärittelyn mukaan data tuli toimittaa Frendsin kautta Azure Data Lake Storageen, joka on Microsoftin ylläpitämä tiedostojärjestelmä pilvipalveluna. Määrittelyn mukaan dataa ei kuulunut muokata mitenkään, vaan se tuli toimittaa muuttumattomana, yhteen JSON-muotoiseen tiedostoon kirjoitettuna. Näiden vaatimusten perusteella oli selvää, että prosessissa tulisi olla toiminnallisuus, joka koostaa lähdedatan tiedostoon sekä lähettävä osuus, jossa tiedosto lähetettäisiin perille.

Määrittelyn mukaan integraatio tuli ottaa käyttöön kolmessa eri ympäristössä: testi, staging ja tuotanto.

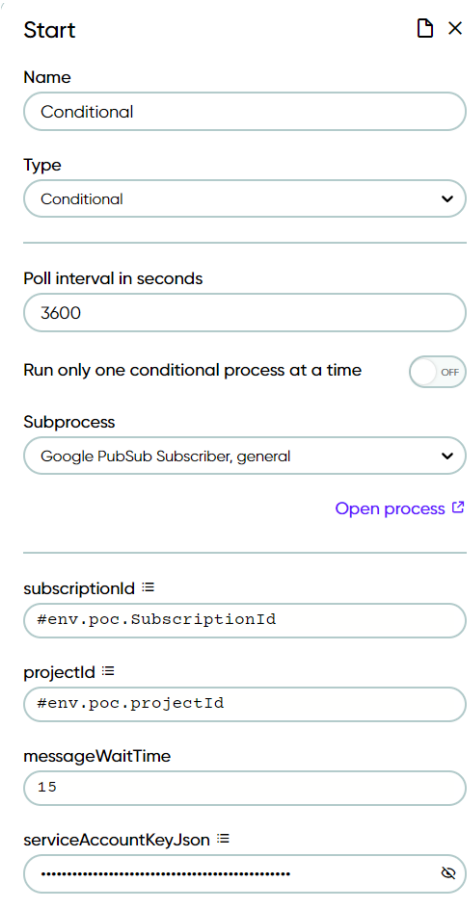
4.2 Integraatioprosessin toteuttaminen

Lyhyen suunnitteluvaiheen jälkeen siirryttiin kehittämään integraatioprosessia. Uuden prosessin luonti alkaa *Processes*-valikon kautta *Create new* -painikkeella. Suunniteltu prosessin aloitustapa edellytti uuden aliprosessin luomista. Aliprosessit sijaitsevat Frends-portaalissa *Subprocesses*-valikon alla, josta *Create new* -painikkeella voi aloittaa uuden kehittämisen. Tarkoitukseen luotiin erittäin yksinkertainen aliprosessi, jonka ainoa toiminnallisuus oli hakea viestit tietystä pub/sub-aiheesta tietylle tilaajalle. Aliprosessin ainoaksi suorittavaksi komponentiksi löytyi jo aiemmin kehitetty tehtäväkomponentti, joka hakee viestit parametrina annettavan tilauksen tunnistetiedon avulla. Komponentille annetaan parametreina myös projektin tunnistetieto sekä salainen avainarvo. Avainarvoparametri määriteltiin salaiseksi, jolloin parametrille annetut arvot eivät näy esimerkiksi prosessin suorituslokilla. Samat parametrit lisättiin myös aliprosessin triggeriin, josta ne välitetään tehtäväkomponentille. Triggeriparametrien arvoja voidaan kutsua Frends-prosessien sisällä kuvassa 2 näkyvällä tavalla. Koska tehtäväkomponentille annettavat arvot saadaan aliprosessin triggerin parametreista, voi aliprosessia kutsuva prosessi määritellä niiden arvot. Tämä mahdollistaa

sen, että aliprosessi on helposti uudelleenkäytettävissä myös muissa integraatioissa. Aliprosessille annettiin nimeksi *General Google PubSub Subscriber* kuvaamaan sen toimintaa sekä yleiskäytettävyyttä.

Kuva 2 Tehtäväkomponentin parametriarvojen määritys

Aliprosessin jälkeen päästiin kehittämään varsinaista integraatioprosessia. Aloituskohdaksi valittiin suunnitelman mukaisesti ehdollinen komponentti, joka kutsui kehitettyä aliprosessia tunnin välein. Aliprosessille piti välittää parametreina tietoa pub/sub-tilauksesta. Koska integraatioprosessia oli tarkoitus ajaa kolmessa eri ympäristössä, joista jokaiselle on oma pub/sub-tilauksensa, hyödynnettiin tässä vaiheessa Frendsin ympäristömuuttujia. Määritettiin uusi ympäristömuuttujaryhmä, johon luotiin ympäristömuuttujat jokaiselle kuvassa 2 mainitulle parametrille. Muuttuja *serviceAccountKeyJson* määriteltiin salaiseksi muuttujaksi, jolloin sille asetetut arvot eivät ole näkyvissä portaalin kautta. Muuttujien luomisen jälkeen niille asetettiin niiden ympäristökohtaiset arvot. Prosessin käyttöön ympäristömuuttujia voidaan Frendsissä kutsua kuvassa 3 näkyvällä notaatiolla. Kuvassa näkyy prosessin aloituskohdan parametrit: kutsuttava aliprosessi sekä intervalli, jolla aliprosessia kutsutaan. Lisäksi niiden alapuolella kutsuttavan aliprosessin parametrit.



Kuva 3 Ehdollinen triggeri

Suunnitelman mukaan viestit oli määrä kirjoittaa JSON-tiedostoon. Frendsin valmiiden tehtäväkomponenttien ja dokumentaation selaamisen jälkeen todettiin, että tarkoitukseen on jo valmiiksi olemassa sopivat komponentit. Frendsin *Files.CreateDirectory* -komponentin avulla voidaan luoda tiedostohakemisto, johon uusi tiedosto voidaan väliaikaisesti tallentaa. *File.Write* -komponentin avulla voidaan luoda tiedosto. Komponentille annetaan parametreina tiedoston sisältö sekä polku, johon tiedosto tallennetaan. Tässä tapauksessa sisältö oli aliprosessin palauttamat viestit JSON-tekstimuodossa ja polku *Files.CreateDirectory* -komponentin palauttama polku luotuun tiedostohakemistoon.

Frends ei integraatiota toteutettaessa tarjonnut virallista komponenttia tiedostojen lataamiseksi Azure Storageen, joten sellainen tuli ohjelmoida itse. Komponentin pohjana käytettiin Frendsin tarjoamaa mallia, johon lähdekoodi saatiin Githubista. Mallin *Definition.cs*-tiedostoon (kuva 4) lisättiin kaksi luokkaa, *Parameters* ja *Result. Parameters*-luokassa määriteltiin komponentin parametrit ja

niiden tietotyypit. Token-parametrille lisättiin *PasswordPropertyTextAttribute*-attribuutti, jotta sen sisältämää arvoa ei tallennettaisi Frends-prosessin suorituslogiin. *Result*-luokassa määriteltiin komponentin palautusarvot.

```
1 reference | 0 changes | 0 authors, 0 changes
public class Parameters
{

    3 references | 0 changes | 0 authors, 0 changes
    public string Path { get; set; }
    [PasswordPropertyText]
    1 reference | 0 changes | 0 authors, 0 changes
    public string Token { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public string FilePath { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public string XMsVersion { get; set;}
}

2 references | 0 changes | 0 authors, 0 changes
public class Result
{
    public HttpResponseMessage Response;
    public string ErrorMessage;
}
```

Kuva 4 Definition.cs

Metodi, jonka nimeksi valikoitui *UploadFileToAzureStorage* (kuva 5), suoritti kolme kutsua Azure Storagen rajapintaan. Ensimmäisessä kutsussa luotiin uusi tiedosta parametrina määriteltyyn polkuun, toisessa kutsussa uuteen tiedostoon ladattiin dataa parametroidusta tiedostosijainnista ja kolmannessa kutsussa tyhjennettiin tiedostopuskuri. Metodi palauttaa vastauksen Azuren rajapinnasta, sekä mahdollisen virheviestin, jos jotain menee pieleen.

```
0 references | 0 changes | 0 authors, 0 changes
public static async Task<Result> UploadFileToAzureStorage(Parameters input, CancellationToken cancellationToken)
{
    var result = new Result();
    try
    {
        string fileLength;

        var auth = input.Token;
        var msVersion = input.XMsVersion;
        var createMessage = new HttpRequestMessage
        {
            Method = HttpMethod.Put,
            RequestUri = new Uri($"{input.Path}?resource=file"),
        };
        createMessage.Headers.Add("Authorization", $"Bearer {auth}");
        createMessage.Headers.Add("x-ms-version", msVersion);
        var response = await HttpClient.SendAsync(createMessage, cancellationToken);

        if (response.IsSuccessStatusCode)
        {
            using (Stream stream = File.OpenRead(input.FilePath))
            {
                var streamContent = new StreamContent(stream);
                fileLength = stream.Length.ToString();
                var appendMessage = new HttpRequestMessage
                {
                    Method = HttpPatch,
                    RequestUri = new Uri($"{input.Path}?action=append&position=0"),
                    Content = streamContent
                };
                appendMessage.Headers.Add("Authorization", $"Bearer {auth}");
                appendMessage.Headers.Add("x-ms-version", msVersion);
                appendMessage.Content.Headers.Add("Content-Length", fileLength);
                appendMessage.Content.Headers.Add("Content-Type", "application/octet-stream");

                response = await HttpClient.SendAsync(appendMessage, cancellationToken);
            }

            if (response.IsSuccessStatusCode)
            {
                var flushMessage = new HttpRequestMessage
                {
                    Method = HttpPatch,
                    RequestUri = new Uri($"{input.Path}?action=flush&position={fileLength}"),
                };
                flushMessage.Headers.Add("Authorization", $"Bearer {auth}");
                flushMessage.Headers.Add("x-ms-version", msVersion);
                response = await HttpClient.SendAsync(flushMessage, cancellationToken);
                result.Response = response;
                return result;
            } else
            {
                result.Response = response;
                return result;
            }
        } else
        {
            result.Response = response;
            return result;
        }
    }
    catch (Exception ex)
    {
        result.ErrorMessage = $"An error occurred: {ex.Message}";
    }
    return result;
}
```

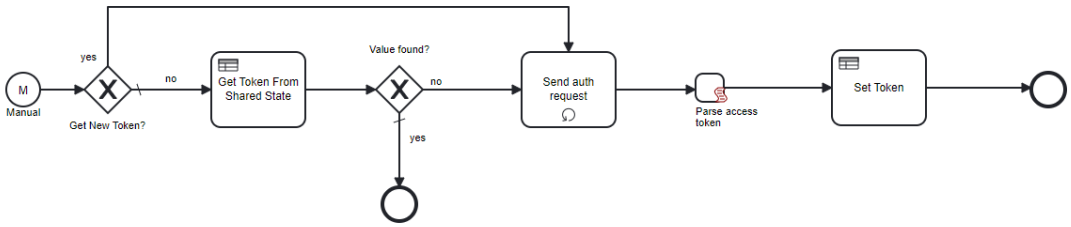
Kuva 5 Komponentin logiikan suorittava metodi

Koodia olisi voinut myös kommentoida XML-notaatiolla, jolloin kommentit näkyisivät Frends-käyttöliittymän prosessieditorissa komponenttia käytettäessä. Tämä on hyödyllinen ominaisuus silloin, kun vaadituista parametreista tai komponentin toiminnasta halutaan tarjota lisätietoja käyttäjälle.

Tällä kertaa kommentit kuitenkin jäivät pois koodista. Kun koodi oli valmis, luokkakirjasto pakattiin NuGet-paketiksi. Paketti ladattiin portaalin *Administration – Tasks* -valikon kautta Frendsiin *Import Task NuGet* -toiminnon avulla. Tämän jälkeen kehitetty komponentti on kaikkien Frends-prosessien käytettävissä.

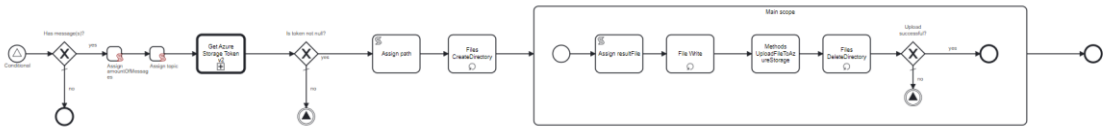
Integraatioprosessi tarvitsi vielä toiminnallisuuden uudelle komponentille syötettävän Azure Access Tokenin noutamista varten. Azure Access Tokenit ovat väliaikaisia tunnistetietoja, joita tarvitaan kutsuttaessa Azure-palveluiden suojattuja API-rajapintoja, kuten tässä tapauksessa Azure Data Lakea [28]. Frends ei tarjonnut tarkoitukseen virallista komponenttia, joten toiminnallisuutta varten päätettiin kehittää uusi aliprosessi, joka olisi mahdollisesti muidenkin integraatioiden käytettävissä. Microsoftin dokumentaation mukaan pääsyoikeustunnisteen hakemiseen tarvittavia tietoja olivat sovelluksen tunnus (*client_id*), salasana (*client_secret)* sekä organisaatiotunnus (*tenant_id)* [29]. Nämä tiedot asetettiin annettavaksi aliprosessin parametreissa. Lisäksi aliprosessin kutsuun lisättiin parametrit *CacheKey* sekä *GetNewToken*, josta aiemman arvolla pystytään nimeämään haettava tunnistetieto ja jälkimmäisen arvolla päättämään, haetaanko uusi tunnistetieto suoraan Azure AD -palvelun kautta vai koetetaanko se hakea ensin agenttiryhmän jaetusta tietokannasta (*Shared State*).

Kuvassa 6 on nähtävillä kehitetyn aliprosessin *Get Azure Storage Token* prosessikaavio. Aloituskohdan jälkeen ehtolause tarkastelee parametrin *GetNewToken* arvoa. Jos arvoksi on valittu ”no”, tarkastelee prosessi, löytyykö parametreissa määrätyllä tavalla nimettyä tunnistetietoa tietokannasta. Tämä tapahtuu Frendsin *Shared State Task* -komponentin avulla. Jos tunnistetieto löytyy, aliprosessi palauttaa tunnistetiedon ja suoritus päättyy. Jos taas tunnistetietoa ei löydy tietokannasta tai *GetNewToken*-parametrin arvo on ”yes”, lähettää aliprosessi HTTP-kutsun Azure AD -palvelulle. Vastauksena palautuu tunnistetieto, joka tallennetaan jaettuun tietokantaan toisen *Shared State Task* -komponentin avulla. Tunnistetieto asetetaan komponentin parametreilla vanhentumaan kymmenen minuutin päästä, jolloin tieto poistetaan tietokannasta. Tallentamisen jälkeen aliprosessi palauttaa tunnistetiedon ja prosessin suoritus päättyy.



Kuva 6 Get Azure Storage Token -aliprosessi

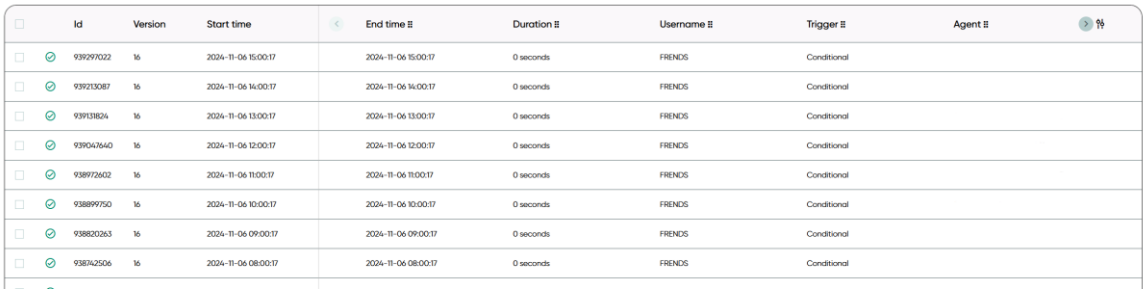
Kun kaikki tarvittavat komponentit ja aliprosessit integraatioprosessin vaatimuksiin oli kehitetty, muodostui prosessikaavio kuvassa 7 nähdyn kaltaiseksi.



Kuva 7 Prosessikaavio

Aloituskohdan jälkeen prosessi etenee ehtolauseeseen, joka tarkastelee, palauttaako aliprosessi yhtään viestiä. Jos viestejä ei ole, prosessin suoritus päättyy. Jos viestejä taas on, alustetaan muuttuja *amountOfMessages,* jonka arvoksi asetetaan viestien lukumäärä. Toinen alustettava muuttuja on *topic*, jonka arvoksi asetetaan pub/sub-aiheen nimi. Tämän jälkeen haetaan tunnistetieto. Jos tunnistetiedon haku jostain syystä epäonnistuu, palautetaan virhe, jonka jälkeen prosessin suoritus päättyy. Muussa tapauksessa alustetaan *path*-muuttuja, jonka arvoksi tulee vaatimusmäärittelyssä määritelty polku Azure Data Lake -tiedostojärjestelmässä. Tähän polkuun integraation koostama tiedosto tullaan lopulta lähettämään. Polkumuuttujan määrittelyn jälkeen prosessi luo kansion tiedoston väliaikaista tallentamista varten. Tämän jälkeen edetään *Main scope* -elementin suoritukseen. Elementin sisällä koostetaan viestit C#-koodinpätkän *String.Join(", ", #trigger.data.result.messages)* avulla yhdeksi string-muotoiseksi muuttujaksi, joka kirjoitetaan tiedostoon. Tiedosto tallennetaan aiemmin luotuun väliaikaiseen hakemistoon. Seuraavaksi prosessisuoritus kutsuu integraatiota varten kehitettyä *UploadFileToAzureStorage*-metodia, jonka parametrien arvoiksi prosessi antaa väliaikaisen tiedostopolun, *path*-muuttujan arvon sekä *access_token* -tunnistetiedon. Tiedoston lähettämisen jälkeen väliaikainen hakemisto poistetaan. Lopuksi prosessi palauttaa vastauskoodin merkiksi joko onnistuneesta tai epäonnistuneesta tiedoston lähetyksestä sen mukaan, mitä *UploadFileToAzureStorage*-komponentti palauttaa, ja prosessin suoritus päättyy.

Kun kehitetty prosessi oli valmis, voitiin se viedä kehitysympäristöstä testiympäristöön testattavaksi. Käyttöliittymällä valittiin kehitetty prosessi ja painettiin *Deploy Process to Agent Group*. Listalta valittiin sopiva testiympäristö, jonka jälkeen valinta *Deploy and activate triggers* vei prosessin haluttuun ympäristöön ja aktivoi ajastetun triggerin. Prosessisuoritusten lokeja voi selata Frends-portaalin kautta valitsemalla *Show Process instances.* Prosessiajot näytetään listana, jossa onnistuneet ajot on merkattu vihreällä oikeinmerkillä ja epäonnistuneet punaisella rastilla (kuva 8). Listaa voi suodattaa esimerkiksi päivämäärien tai ajon onnistumisen mukaan.



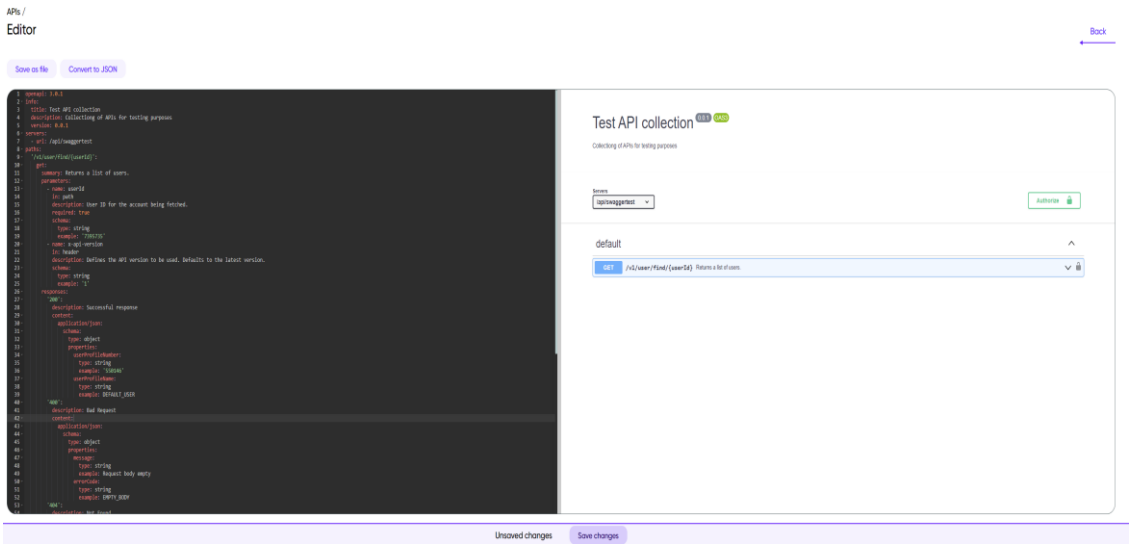
Id	Version	Start time	End time	Duration	Username	Trigger	Agent
939297022	16	2024-11-06 15:00:17	2024-11-06 15:00:17	0 seconds	FRENDS	Conditional	
939213087	16	2024-11-06 14:00:17	2024-11-06 14:00:17	0 seconds	FRENDS	Conditional	
939131824	16	2024-11-06 13:00:17	2024-11-06 13:00:17	0 seconds	FRENDS	Conditional	
939047640	16	2024-11-06 12:00:17	2024-11-06 12:00:17	0 seconds	FRENDS	Conditional	
938972602	16	2024-11-06 11:00:17	2024-11-06 11:00:17	0 seconds	FRENDS	Conditional	
938899750	16	2024-11-06 10:00:17	2024-11-06 10:00:17	0 seconds	FRENDS	Conditional	
938820263	16	2024-11-06 09:00:17	2024-11-06 09:00:17	0 seconds	FRENDS	Conditional	
938742506	16	2024-11-06 08:00:17	2024-11-06 08:00:17	0 seconds	FRENDS	Conditional	

Kuva 8 Prosessilokia

Taulukon riviä klikkaamalla avautuu yhden prosessisuorituksen loki, josta voi tarkastella prosessin jokaisen komponentin suoritusta erikseen. Jos jonkin komponentin suoritus epäonnistuisi, olisi komponentti merkattu punaisella värillä. Kehitetyssä prosessissa jos esimerkiksi Azureen lataus epäonnistuisi, näkyisi prosessin *Throw*-komponentti punaisena ja ilmoittaisi epäonnistuneesta latauksesta. Tällöin *UploadToAzureStorage*-komponentin palautusta tarkastelemalla voitaisiin päätellä, miksi lataus ei onnistunut.

4.3 API-hallinta ja API-rajapinnan kehitys

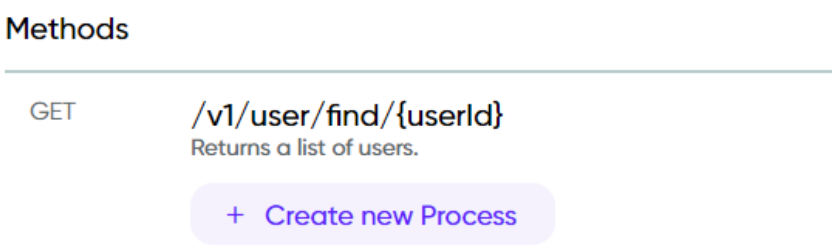
API-hallinta ja API-rajapintojen hyödyntäminen on oleellinen osa Frends-integraatioalustaa. Koska edellä kehitetty integraatioputki ei hyödyntänyt alustan API-ominaisuuksia, on niitä syytä tutustua vielä erikseen. API-dokumentaation hallinta tapahtuu Frendsissä *APIs*-valikon kautta. Näkymässä on listattu API-rajapintakokoelmia ympäristöittäin. Valikon kautta voidaan muokata aiemmin kehitettyjä rajapintoja, julkaista API-kokoelmia ympäristöihin ja avata API-määrityksiä Swagger UI -työkaluun, jossa rajapintoja voidaan testata.



Kuva 9 API-määrittelyn muokkaus

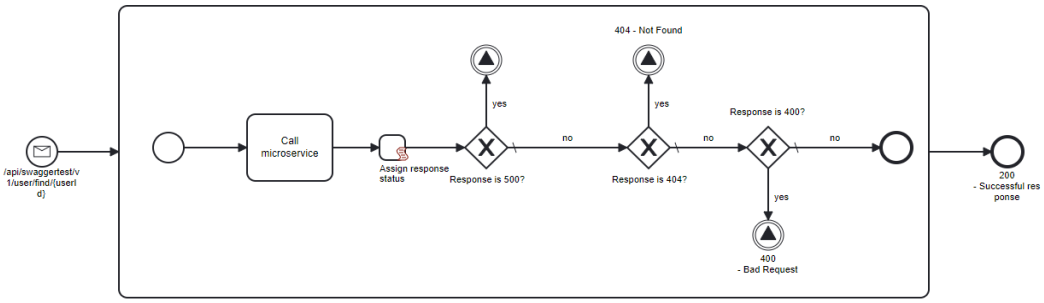
Kokoelmien luonti ja muokkaus on mahdollista vain kehitysympäristössä. Uuden määritelmän luonti alkaa *Create new* -painikkeella. Avautuvassa näkymässä (kuva 9) määritystä päästään muokkaamaan joko YAML- tai JSON-muotoisen, OpenAPI 3.0 (Swagger) -spesifikaation mukaisen tekstitiedoston avulla. Näkymän oikealla laidalla on visuaalinen esitys määrityksestä. Frends-dokumentaatiossa neuvotaan, että määrittely voidaan tehdä myös Swagger Editor -työkalulla, jonka jälkeen määrittely voidaan kopioida tähän näkymään [20]. Kuvassa näkyvässä määrittelyssä esitellään yksi GET-rajapinta, joka palauttaa listan käyttäjistä. Määrittelyssä listataan kutsun parametrit sekä mahdolliset vastausviestit, sekä määritellään kutsun polku.

Kun määrittely on tehty, tallennetaan API *Save*-painikkeella. Tallennuksen jälkeen määrittellylle rajapinnalle tulee luoda linkitetty prosessi. Kuvassa 10 näkyvällä *Create new Process* -painikkeella avautuu prosessieditori, johon Frends on luonut valmiiksi aloituskohtaelementin sekä mahdolliset palautuselementit API-määrittelyn tietojen perusteella.



Kuva 10 Prosessin linkitys rajapintaan

Rajapintaan linkitetyn integraatioprosessin kehitys tapahtuu samoin kuin muillakin tavoin käynnistettyjen prosessien. Kutsu rajapintaan käynnistää integraatioprosessin, jonka jälkeen toiminnallinen osuus rakentuu prosessissa määritellyllä tavalla. Toiminnallisen, dataa käsittelevän osuuden jälkeen prosessin suoritus päättyy, kun prosessi palauttaa kutsujalle jonkin määrittelyssä esitellyn http-vastauksen.



Kuva 11 Prosessi käyttäjien hakuun

Yllä olevassa kuvassa on luotu prosessi aiemmin määritellylle rajapinnalle. Prosessi ohjaa rajapintaan tulleen kutsun parametreineen mikropalvelulle Frendsin tarjoaman *WebForwardRequest* -komponentin avulla. Esimerkiksi rajapinnan polkuparametri *userId* saadaan komponentin käyttöön notaatiolla *#trigger.data.path.userId.* Komponentin parametreissa on annettu kutsuttavan mikropalvelun polku sekä tarvittavat otsikkotiedot. Mikropalvelukutsun vastauskoodin perusteella prosessi päättelee sopivan http-vastausmuodon alkuperäiseen kutsuun ja palauttaa mikropalvelulta saamansa vastauksen. Tämän jälkeen prosessin suoritus päättyy.

Linkitetyn prosessin tallennuksen jälkeen palataan API-hallintanäkymään, josta kehitetty kokoelma voidaan viedä toiseen ympäristöön. Viennin yhteydessä päätetään kohdeympäristö, sekä vietävä versio sekä API-määrittelystä että linkitetyistä prosesseista, jos versioita löytyy useita mahdollisia. Linkitetyt prosessit kopioituvat automaattisesti valittuun kohdeympäristöön rajapintakokoelman mukana, joten niitä ei tarvitse erikseen viedä prosessivalikon kautta. Valitun kohdeympäristön perusteella muodostuu rajapinnan verkko-osoite.

Seuraava askel API-kehityksessä on määritellä rajapintojen autentikaatiotapa. Autentikaatiolla tarkoitetaan rajapinnan kutsujan tunnistamista. Frends tarjoaa tarkoitukseen kolme erilaista vaihtoehtoa: API-avaimet, Basic-autentikaation ja OAuth2-tunnistetiedon avulla tehtävän autentikaation. API-avainten avulla voidaan myös rajoittaa käyttäjien pääsyä rajapintadokumentaatioon ja Swagger UI -näkymään.

Kehityttyjä API-rajapintoja voi testata Frends-portaalin kautta ympäristössä, jossa on käynnissä olevia agentteja. Hallintavalikosta painikkeella *API Specification UI* aukaistaan Swagger UI -työkalu. Työkalu näyttää valitun API-kokoelman rajapintamääritykset ja sallii käyttäjän lähettää rajapintoihin määrittelyn mukaisia kutsuja.

5 YHTEENVETO

Opinnäytetyön tavoitteena oli esitellä Frends-integraatioalustan toimintoja ja kuvata alustan käyttöä. Aluksi esiteltiin lyhyesti, mitä alustaan keskeisesti liittyvät termit järjestelmäintegraatio ja iPaaS tarkoittavat. Seuraavassa kappaleessa esiteltiin Frends-integraatioalustan eri komponentteja ja arkkitehtuuria. Alustan käyttöä havainnollistettiin käytännön esimerkkien avulla neljännessä kappaleessa, jossa kehitettiin tosielämän tarpeeseen integraatioprosessi sekä API-rajapinta.

Työn lopputuloksena tavoiteltiin dokumenttia, jonka avulla alustaan aiemmin tutustumaton henkilö saisi yleiskäsityksen siitä, mihin tuote soveltuu ja miten sitä käytetään. Tähän tavoitteeseen päästäkseni halusin painottaa käytännön esimerkkejä, joiden avulla alustan käyttäminen integraatioiden tekemiseen tulee tutuksi lukijalle. Lopputulos palvelee erityisesti Frends-portaalin käytöstä kiinnostunutta. Olisi ollut mielenkiintoista perehtyä enemmän Frends-ympäristön pystyttämiseen ja agenttien asentamiseen ja konfigurointiin, myös esimerkkien avulla, mutta se ei tämän työn puitteissa ollut mahdollista.

Opinnäytetyön tekeminen opetti minulle itsellenikin uusia puolia Frends-alustasta. Vaikka olin käyttänyt alustaa ennen opinnäytetyötäni jo jonkin verran, opin paljon uutta etsiessäni tietoa työtä varten. Lisäksi asioiden kirjoittaminen auki sai minut oivaltamaan useita asioita, joita en ollut aiemmin huomannut. Haasteita työn etenemiselle aiheutti rajallinen käytettävissä oleva aika. Välillä opinnäytetyön teko keskeytyi useaksi viikoksi, jolloin jatkaessa piti hieman muistella mihin jäätiin. Kokonaisuutena olen kuitenkin tyytyväinen sekä prosessiin että lopputulokseen.

LÄHTEET

1. Tähtinen, S. 2005. Järjestelmäintegraatio: tarve, vaihtoehdot, toteutus. Talentum. Helsinki.

2. Rumpu, A. 2024. Mitä hyötyä järjestelmäintegraatiosta? Visma Solutions Oy. Luettavissa: https://netvisor.fi/blog/mita-hyotya-jarjestelmaintegraatiosta. Luettu 22.9.2024.

3. Mulesoft s.a. System Integration made easy. Luettavissa: https://www.mulesoft.com/resources/esb/system-integration-esb. Luettu 10.11.2024.

4. Bhutada, Teja 2023. Integration Security: Safeguarding Your Data in Connected Systems. Luettavissa: https://exalate.com/blog/integration-security/. Luettu 10.11.2024.

5. China, C.R. & Goodwin, M. 2024. What is iPaaS (integration platform as a service)? IBM. Luettavissa: https://www.ibm.com/topics/ipaas. Luettu 10.11.2024.

6. Microsoft s.a. Visualisoi liiketoimintaprosessit selkeästi. Luettavissa: https://www.microsoft.com/fi-fi/microsoft-365/visio/business-process-modeling-notation. Luettu 13.9.2024.

7. Frends Technology 2024a. Low-Code Approach. Luettavissa: https://frends.com/platform/low-code-approach. Luettu 13.9.2024.

8. Frends Technology 2024b. Platform. Luettavissa: https://frends.com/platform/distributed-architecture. Luettu 13.9.2024.

9. Galkin, Ossi 2024k. Introduction to Frends Architecture Overview. Luettavissa: https://docs.frends.com/en/articles/8027111-introduction-to-frends-architecture-overview. Luettu 22.9.2024.

10. Frends Technology 2024c. Frends Agents. Luettavissa: https://frends.com/platform/frends-agents. Luettu 13.9.2024.

11. Galkin, Ossi 2024a. Agent Groups. Luettavissa: https://docs.frends.com/en/articles/2376462-agent-groups. Luettu 15.9.2024.

12. Galkin, Ossi 2024m. Installing an Agent. Luettavissa: https://docs.frends.com/en/articles/2188617-installing-an-agent. Luettu 6.11.2024.

13. Galkin, Ossi 2024b. Process. Luettavissa: https://docs.frends.com/en/articles/2236816-process. Luettu 15.9.2024.

14. Virtanen, Riku 2024. Introduction to Triggers in Frends. Luettavissa: https://docs.frends.com/en/articles/8010526-introduction-to-triggers-in-frends. Luettu 7.9.2024.

15. Galkin, Ossi 2024c. Scope. Luettavissa: https://docs.frends.com/en/articles/6034843-scope. Luettu 16.9.2024.

16. Galkin, Ossi 2024d. Introduction to Return. Luettavissa: https://docs.frends.com/en/articles/8010623-introduction-to-return. Luettu 16.9.2024.

17. Wikipedia Foundation 2024. OpenApi Specification. Luettavissa: https://en.wikipedia.org/wiki/OpenAPI_Specification. Luettu 11.9.2024.

18. Galkin, Ossi 2024e. API Management. Luettavissa: https://docs.frends.com/en/articles/2206760-api-management. Luettu 16.9.2024.

19. SmartBear Software 2024. Swagger UI. Luettavissa: https://swagger.io/tools/swagger-ui/. Luettu 22.9.2024.

20. Galkin, Ossi 2024f. Creating a Frends API. Luettavissa: https://docs.frends.com/en/articles/2188728-creating-a-frends-api. Luettu 16.9.2024.

21. Galkin, Ossi 2024g. Introduction to Frends Environment. Luettavissa: https://docs.frends.com/en/articles/8027133-introduction-to-frends-environment. Luettu 16.9.2024.

22. Galkin, Ossi 2024h. Example on What Tasks are. Luettavissa: https://docs.frends.com/en/articles/8010710-example-on-what-tasks-are. Luettu 4.9.2024.

23. Frends Technology s.a. Frends Tasks. Luettavissa: https://tasks.frends.com. Luettu 4.9.2024.

24. Galkin, Ossi 2024i. Tasks. Luettavissa: https://docs.frends.com/en/articles/2211481-tasks. Luettu 4.9.2024.

25. Galkin, Ossi 2024j. Custom Tasks. Luettavissa: https://docs.frends.com/en/articles/2206746-custom-tasks. Luettu 4.9.2024.

26. Github 2024. Frends Task template. Luettavissa: https://github.com/Community-HiQ/TaskTemplate. Luettu 4.9.2024.

27. Galkin, Ossi 2024l. Introduction to Conditional Trigger. Luettavissa: https://docs.frends.com/en/articles/8010560-introduction-to-conditional-trigger. Luettu 2010.10.2024.

28. Microsoft 2024a. Access tokens in the Microsoft identity platform. Luettavissa: https://learn.microsoft.com/en-us/entra/identity-platform/access-tokens. Luettu 19.10.2024.

29. Microsoft 2024b. Microsoft identity platform and OAuth 2.0 authorization code flow. Luettavissa: https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-auth-code-flow. Luettu 19.10.2024