Dmitri Ludwig

Slackbot-toteutus Frends-integraatioalustalla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

10.11.2022

Tiivistelmä

Tekijä:	Dmitri Ludwig
Otsikko:	Slackbot-toteutus Frends-integraatioalustalla
Sivumäärä:	39 sivua + 0 liitettä
Aika:	10.11.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Simo Silander Senior Integration Developer Ari Kankainen

---

Tämä insinöörityö esittelee integraatiota käsitteenä, sen toimintaperiaatteita sekä soveltuvuutta ohjelmointialalla. Työssä käsitellään ”Frends”-integraatioalustaa tarkasti ja kiinnitetään erityistä huomiota integraatioiden tuotantomenetelmiin sekä integraatioiden elinkaareen. Puheeksi otetaan yleisesti askarruttavat kysymykset kuten tietoturvallisuus, toimintatavat sekä virhetilanteet.

Työssä verrataan integraatiokehitystä tavalliseen olio-ohjelmointiin sekä perustellaan integraatiokehityksen tarpeellisuus nykymaailman yritysten liiketalousmalleissa. Integraation elinkaarta käydään läpi käyttäen maailmanlaajuisia integraatiotoimintatapoja sekä Frendsin omaa integraatiokäsikirjaa.

Esimerkkitoteutuksena esitellään Slackin ja Planmillin välinen Frends-integraatio, jossa dataa siirretään ja muokataan käyttäen C#-koodia sekä LINQ-kyselyitä. Lopussa käsitellään tuotannossa olevan integraation seurantaa sekä virhetilanteiden hallinta.

Avainsanat: Integraatio, Frends, C#, LINQ

Abstract

Author:	Dmitri Ludwig
Title:	Slackbot Integration with Frends iPaaS
Number of Pages:	39 pages + 0 appendices
Date:	6.11.2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, Senior Lecturer
	Ari Kankainen, Senior Integration Developer

This engineering thesis introduces the idea of integration, along with its working principles and programming-related applications. The "Frends" integration platform is discussed in detail, with a focus on the integration life cycle and integration production techniques. The document discusses frequent problems including security, policy, and error scenarios.

The thesis explains the need for integration development in the business models of contemporary organizations by contrasting it with conventional object-oriented programming.

Data is exchanged and edited using C# code and LINQ queries in the connection between Slack-Frends-Planmill, which is shown as an example implementation. The management of errors and the monitoring of production integration are also described.

Keywords: Integration, Frends, C# code, and LINQ queries

Sisällys

Lyhenteet

1 Johdanto 1

2 Integraatioista yleisesti 2

2.1 Integraatio käsitteenä 3
2.2 Miksi integraatio-alusta/iPaaS? 3
2.3 Integraatio käytännössä 4

3 Frends integraatioalustana 5

3.1 Frendsin esittely 5
3.2 Frendsin tietoturvallisuus 6
3.3 Frends:in toimintatavat ja Integraatiokäsikirja 8

4 Slackbot integraatioesimerkkinä 8

4.1 Integraation suunnittelu 9
4.2 Integraation toteutus 12
4.3 Integraation testaaminen/katselmointi 25
4.4 Integraation tuotantoon vieminen 28

5 Jälkitoimenpiteet 29

5.1 Tuotannossa olevan integraation seuranta 29
5.2 Tuotannossa olevan integraation virhetilanteet ja niiden hallinta 30

6 Yhteenveto 30

Lähteet 32

Lyhenteet

VolPlan: Henkilöstöresurssien suunnittelu ja hallintaohjelmisto (työvuorot).

SalCal: Palkanlaskentajärjestelmä.

ERP: Palveluliiketoiminnan toiminnanohjausjärjestelmä, jonka avulla voi hallinnoida asiakkuuksia, projekteja, kuluja, raportteja ja taloutta. ERP = *Enterprise resource planning*.

Stampcon: Järjestelmä, joka mahdollistaa tunti- ja ruokaleimojen seuraamisen.

Mäppäys: Englannin kielen sanasta ”mapping”, tarkoittaa ohjelmoinnin kontekstissa, että otetaan monta yksittäistä asiaa ja luodaan niiden välille assosiaatiot.

Rajapinta: On käytännössä jonkin järjestelmän sisäisen toteutuksen ulospäin näkyvä liitäntä, jonka kautta se on yhteydessä ulkomaailmaan.

Plugin: Tietokoneohjelmisto, joka lisää uusia toimintoja isäntäohjelmaan muuttamatta itse isäntäohjelmaa.

Widget: Graafisen käyttöliittymän elementti, joka näyttää tietoja tai tarjoaa käyttäjälle tietyn tavan olla vuorovaikutuksessa käyttöjärjestelmän tai sovelluksen kanssa.

iPaaS: *Integration platform as a Service*. Suomeksi tämä on integraatio alusta, palveluna.

On-prem: *On Premises,* tarkoittaa, että ohjelma on asetettu käyttäjän tai yrityksen sisäverkkoon sijoitetulle koneelle tai palvelimelle.

ICN-malli: *Integration Competency Network*. Suomennettuna integraatio-osaamisverkosto.

HR: *Human Resourses, esim SYMPA on* henkilöstöhallinnon työkalu.

PII: *Personal Identifiable Information*, suomeksi henkilökohtaiset tunnistetiedot, esimerkiksi henkilötunnus tai koko nimi.

OAuth: *Open Authentication* on protokolla pääsyoikeuksien välittämiseen palvelusta toiseen ilman salasanan tai tunnuksien uudelleen välittämistä.

API: *Application Programming Interface* eli ohjelmointirajapinta, joka on siis raja komponenttien/moduulien välillä ohjelmoitavassa järjestelmässä.

HTTPS: *Hypertext Transfer Protocol Secure* on HTTP-protokollan ja TLS/SSL-protokollan yhdistelmä, jota käytetään tiedon suojattuun siirtoon webissä.

TLS: *Transport Layer Security* on salausprotokolla, jolla voidaan suojata Internet-sovellusten tietoliikenne IP-verkkojen yli.

SSL: *Secure Sockets Layer* on salausprotokolla, jolla suojataan internet-yhteyttä.

SFTP: *Secure File Transfer Protocol* on verkkoprotokolla, joka tarjoaa tiedostojen käytön, tiedonsiirron ja tiedostojen hallinnan minkä tahansa luetettavan tietovirran kautta.

IP: *Internet Protocol address* on sarja numeroita, jota käytetään IP-verkkoihin kytkettyjen laitteiden yksilöimiseen.

GDPR: *The General Data Protection Regulation* on EU:n yleinen tietosuoja-asetus, jossa määritellään, mitkä toimenpiteet ovat kenenkin vastuulla henkilötietojen suojaamiseksi.

Log: Suomeksi lokitus, on automaattisesti toimitettava ja aikaleimattu dokumentaatio tapahtumista, jotka ovat relevantteja kyseisessä ympäristössä.

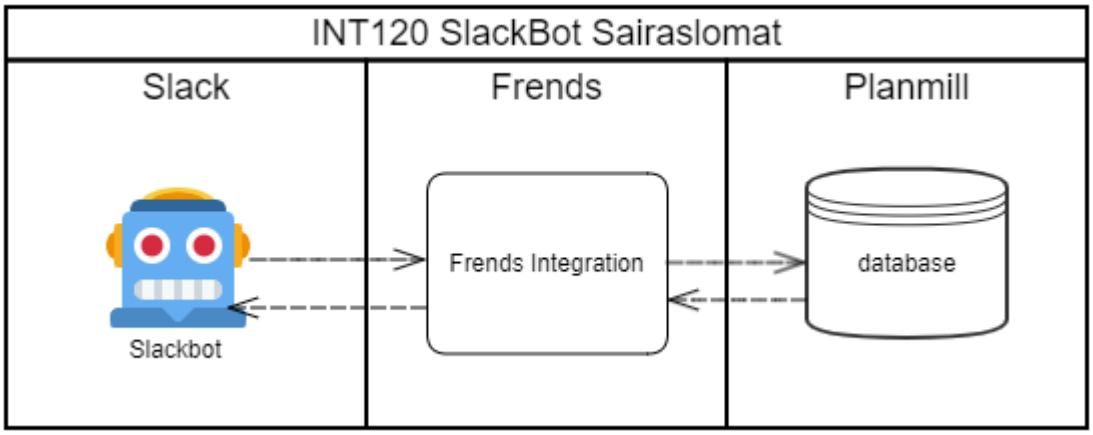
1 Johdanto

Integraatio on eri tekniikoilla tai alustoilla toteutettujen ohjelmistojen tai järjestelmien toisiinsa liittämistä, jolloin nämä liitetyt osat keskustelevat keskenään. Järjestelmien integrointi varmistaa tiedonkulun sekä tiedon liittymisen toisiin järjestelmiin [1].

Frends-integraatioalusta on moderni hybridi-integraatioalusta, jolla voi hoitaa integraatiotarpeet aina tiedostosiirroista hyperautomaatioon sekä API -rajapintojen toteutukseen ja hallintaan. Hybridi-integraatioalustalla tarkoitetaan sitä, että palvelin voi olla sekä pilvessä että ”on-prem”:illä.On-prem tarkoittaa sitä, että toiminta tapahtuu suoraan käyttäjän koneella.

Insinöörityön tarkoituksena on esitellä Frends-integraatioalustaa esimerkin kautta sekä esittämällä Slackbot-integraatio esimerkkinä alustan mahdollisuuksista. Lopussa pohditaan integraatioalustan tietoturvallisuutta. Slackbot on Slack-ympäristössä oleva botti, jonka tarkoituksena on suorittaa tiettyjä tehtäviä määriteltyjen ohjeiden mukaan ainakin osittain itsenäisesti.

2022 HiQ Finland siirtyi FlowDock-sovelluksesta Slack kommunikaatiosovellukseen ja tämän seurauksena ilmestyi tarpeita implementoida tiettyjä toiminnallisuuksia Slackbotille. Tässä insinöörityössä käsitellään Frends-alustan esimerkkitoteutuksena Slackbot-integraatiota, jonka tarkoituksena on välittää dataa yrityksen Slack-käyttäjien ja Planmillin välillä. Planmill on toiminnanohjausjärjestelmä. Kyseisen toteutuksen tavoitteena on välittää käyttäjän sairauspoissaoloilmoitus Slackissä esimiehelle sekä lisäämään siihen kyseisen käyttäjän eniten tehdyt projektit Planmillistä, jotta esimies voi käyttäjän ollessa sairaslomalla välittää nämä projektit muille integraatiokehittäjille. Toteutuksessa käytetään Frends-integraatioalustaa Slackin ja Planmillin kommunikoinnin välipisteenä (kuva 1).



Kuva 1. Kaikki ohjelmointi tapahtuu Frendsissä. Tässä on hyvin yksinkertaistettu arkkitehtuurirakenne [2].

2 Integraatioista yleisesti

On olemassa Software as a Service (SaaS)-yrityksiä, jotka myyvät ohjelmistoa käyttäjille tilausta vasten. Tilauksen mukana yritys tarjoaa teknistä tukea sekä erilaisia vaihtoehtoja parantaa käyttäjien mahdollisuuksia käyttää kyseistä ohjelmistoa (esim. koulutukset, dokumentaatio). Yritykset harvemmin käyttävät vain yhtä ohjelmistoa täyttämään heidän kaikki tarpeensa. Sen sijaan monesti löytyy plug-in sieltä, softa täältä ja ehkä jopa muutama widget. Nämä kaikki kokonaisuutena muodostavat tarvittavan IT-ympäristön, joka mahdollistaa yrityksen kasvun ja kehittymisen. Plug-in on siis tietokoneohjelmisto, joka lisää uusia toimintoja isäntäohjelmaan muuttamatta itse isäntäohjelmaa, widget on taas puolestaan graafisen käyttöliittymän elementti, joka näyttää tietoja tai tarjoaa käyttäjälle tietyn tavan olla vuorovaikutuksessa käyttöjärjestelmän tai sovelluksen kanssa.

Liiketoiminnan näkökulmasta voi olla hyvin kallista lisätä toimintoja jo olemassa olevaan ohjelmistoon. Yritysten jatkuva kasvu kasvattaa myös ohjelmistovaatimuksia. Tästä syystä monesti yritykset käyttävät kymmeniä erilaisia ohjelmistoja/plug-ineja/widgettejä täyttämään heidän tarpeensa.

Tämän lisäksi monilla yrityksillä on itse kehittämät ohjelmistot, jotka hakevat dataa pilvestä ja paikan päällä olevista järjestelmistä. Mitä tapahtuu, kun yrityksellä on kymmeniä erilaisia järjestelmiä pyörimässä erilaisilla alustoilla ja jokainen näistä on erittäin tärkeässä roolissa? Seurauksena aiheutuu datan menetystä sekä hajanaista informaatiota. Tästä syystä näiden erillisten järjestelmien yhteen sovittaminen on erittäin tärkeä yrityksen kasvun ja toiminnallisuuden kannalta.

2.1 Integraatio käsitteenä

Integraatio tarkoittaa kahden erillisen yhdistämistä tai keräämistä yhdeksi kokonaisuudeksi [6]. Usein kyseessä on yrityksen käyttämät henkilöstöhallinta ja resurssihallintajärjestelmät, joiden välillä halutaan kuljettaa tietoja. Esimerkiksi jos yrityksellä on käytössä tuntikirjausjärjestelmä (esim. PlanMill) ja sitten erillinen palkanlaskentajärjestelmä (esim. SalCal) niin eivät ne työntekijöiden tuntikirjaukset automaattisesti siirry näiden järjestelmien välillä, vaan siihen juuri tarvitaan jonkinlainen integraatio.

2.2 Miksi integraatio-alusta/iPaaS?

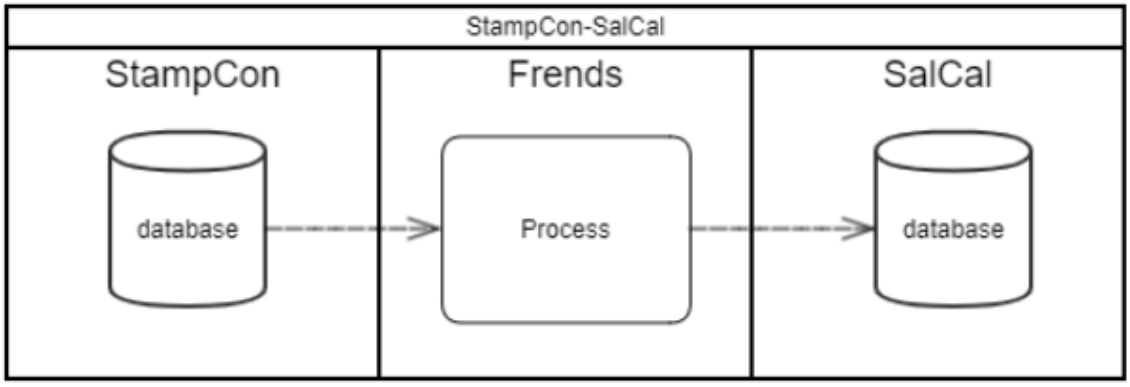
Integraatioalustan tehtävänä on yhdistää erilliset järjestelmät ja toimittaa yhtenäistetty ratkaisu asiakkaille. Se toimii eräänlaisena viestinnän kanavana usealla järjestelmälle, mikä mahdollistaa integraation ja datan jaon. Useimmat yritykset käyttävät erilaisia järjestelmiä, erityisesti myynti-, markkinointi- ja palveluosastojen välillä. iPaaS (integration Platform as a Service) on ratkaisu, joka parantaa viestintää eri osastojen välillä integroimalla ohjelmistoja ja järjestelmiä, jotta tietoja voidaan jakaa paremmin organisaation sisällä ja ulkopuolella.

iPaaS mahdollistaa myös sen, että yritys voi laajentaa tarjontaansa ilman, että sen tarvitsee rakentaa lisää palveluja tai kehittää olemassa olevia järjestelmiä ja palveluja. Sen sijaan se voi integroitua toiseen ohjelmistoon, joka jo tarjoaa kyseistä palvelua, ja tarjoaa asiakkaille yhtenäisen, vankemman ratkaisun. Tästä on käytännön esimerkki luvussa 2.3.

Yksi teoreettisista esimerkeistä voisi olla yksityinen hammaslääkäri, jolla on verkkokauppa-alusta, joka mahdollistaa lääkäreiden ja hoitajien ajanvaraus-muistutuksien hallinnoinnin ja varaamisen sekä niiden lähettämisen asiakkaille. Kun tätä verkkokauppa-alustaa käytetään, huomataan, että hoitajat haluavat myös, että heidän asiakkaansa voivat jättää palautetta ja arvosteluja, sekä mahdollisesti suorittaa maksuja verkkokaupan kautta. Mikäli halutaan vastata asiakkaiden ja hoitajien tarpeisiin, niin on mahdollista edetä kahdella tavalla. Joko rakentaa ja lisätä nämä ominaisuudet verkkokauppa-alustaan tai käyttää iPaaS:ää alustan yhdistämiseen olemassa olevaan arvostelu- ja maksuohjelmistoon. Jälkimmäisen vaihtoehdon avulla voidaan säästää aikaa ja rahaa samalla, kun laajennetaan palvelutarjontaa ja tarjotaan asiakkaille sitä, mitä he haluavat.

2.3 Integraatio käytännössä

Käytännössä integraatio on esimerkiksi työaikaleimauksien siirtämisestä Stampcon-järjestelmästä SalCal-järjestelmään (kuva 2).



Kuva 2. Stampcon-Frends-SalCal-välinen arkkitehtuurirakenne [2].

Eli yrityksellä on sopimusruokailu käytössä paikallisen ravintolan kanssa, ja työntekijät saavat tietyn alennuksen käyttämällä ruokailumaksun yhteydessä yrityksen myöntämää leimaa. Yritys maksaa työntekijän ruokailusta 50 %, työntekijä itse maksaa 50 %. Kun työntekijä leimaa itsensä ruokailun yhteydessä, lei-

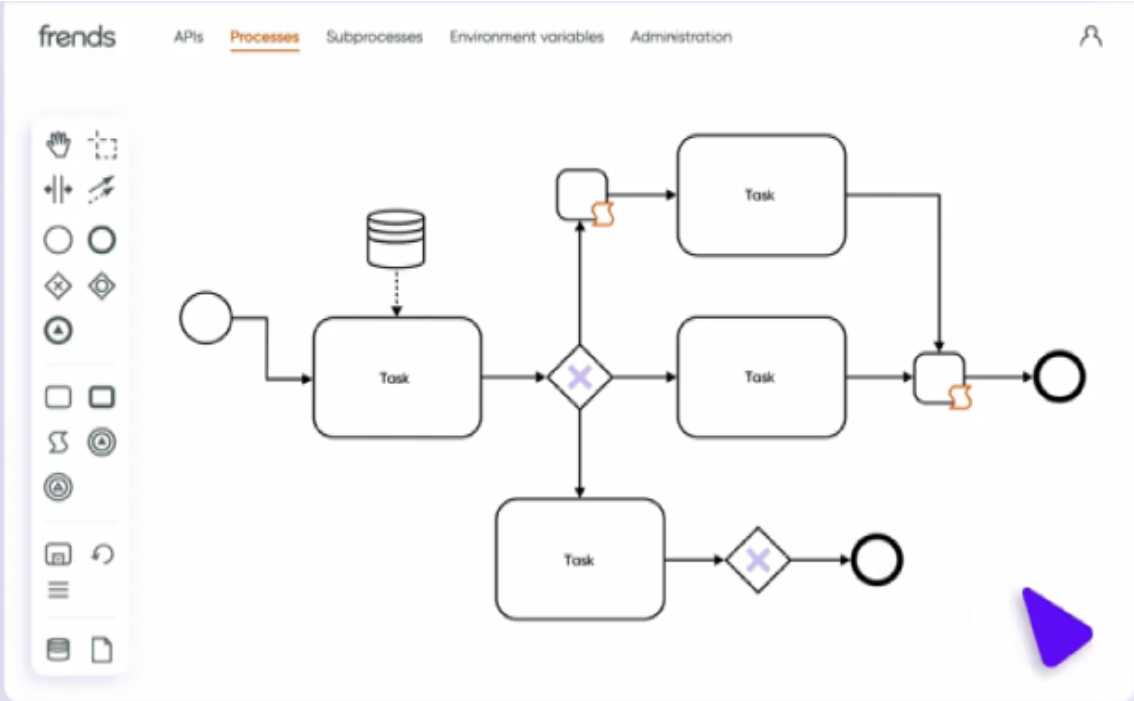
mainfo säilyy Stampcon-järjestelmässä, josta se sitten pitää siirtää SalCal-järjestelmään, jotta sopimusruokailumaksu vähennetään automaattisesti työntekijän palkasta.

Yrityksillä on monesti erilaisia järjestelmiä käytössä, niiden yhteensopivuus on todella harvinaista, koska jokainen ohjelmisto on omalta kehittäjältä. Data pitää kuitenkin saada liikkumaan järjestelmien välillä (kuten yllä mainitussa esimerkissä Stampcon -> SalCal). Tässä tuleekin esille Frends-integraatioalusta. Sen sijaan että yritys palkkaisi ohjelmointitiimin ja tekisi jonkinlaisen datansiirron Stampcon -> SalCal välille, kaikki ohjelmointi tapahtuu Frends-alustalla. Eli Frends vastaanottaa datan Stampcon-järjestelmästä, muokkaa sen siihen muotoon, minkä SalCal on valmis vastaanottamaan ja sen jälkeen siirtää sen SalCallle. Kaikki tämä tapahtuu Frends-järjestelmän sisällä eikä se vaadi asiakkaalta mitään muuta kuin rajapintakuvaukset.

3 Frends integraatioalustana

3.1 Frendsin esittely

Frends on iPaaS (integration Platform as a Service) -alusta (kuva 3), joka yhdistää muuten hajanaiset järjestelmät, widgetit ja pluginit toimittaakseen yritykselle yhtenäisen kokonaisuuden. Frendsin avulla voi mallintaa tarvittavat integraatiovirrat näiden järjestelmien välillä BPMN-pohjaisen visuaalisen käyttöliittymän avulla [6].



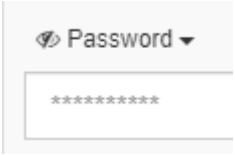
Kuva 3. Mikä on Frends [6]?

3.2 Frendsin tietoturvallisuus

Integraatioita suunniteltaessa on tärkeää ottaa huomioon tietoturvallisuus. Frends-prosessit usein siirtävät luottamuksellista dataa järjestelmien välillä, ja on tehtävä kaikki mahdollinen, jotta ei tapahdu tietojen vahingollista poistamista, muuttamista, tahallista väärinkäyttöä tai varastamista. Tätä varten on olemassa toimintaohjeet, joissa kuvataan käytäntöjä, joita tulisi soveltaa turvallisten integraatioiden suunnittelussa ja kehittämisessä [4].

Integraatioalustan luonteen vuoksi Frendsin käyttäjät voivat saada pääsyn ulkoisiin järjestelmiin ja arkaluonteisiin tietoihin. Tämän vuoksi on olemassa käyttäjäluettelo ja siinä käyttäjien roolit. Nämä on pidettävä ajan tasalla, jotta kenelläkään käyttäjällä ei ole roolia, jota hän ei tarvitse. Esimerkiksi yrityksestä poistuneelta integraatio-ohjelmoijalta pitää ottaa pois nämä käyttäjäroolit. Usein tämä osa onkin täysin asiakkaan kontrollin alaisena. Eli asiakas itse päättää, kenellä kehittäjistä on oikeus mihinkin asiaan.

Kaikki integraatioiden käyttämät konfiguraatiotiedot, kuten rajapintojen käyttäjätunnukset ja salasanat tallennetaan ympäristömuuttujiin. Salasanojen sekä API-avaimien muuttujatyypeiksi laitetaan aina ”secret” jolloin, niiden arvot eivät ole helposti nähtävissä (kuva 4). Salasana-arvon tulisi koostua vähintään kahdestatoista merkistä, joissa on vähintään yksi iso kirjain, yksi pieni kirjain, yksi numero ja yksi erikoismerkki. Jokaiselle ympäristölle ja jokaiselle järjestelmälle on oltava oma salasana.



Kuva 4. Salasana-arvon asettaminen ”secret”-tyyppiseksi [7].

Uutta prosessia luotaessa kehittäjä luo ympäristömuuttujat tunnuksia varten, ja asiakas täyttää ne Frendsissä. Täten edes kehittäjän ei tarvitse tietää näitä salasanoja, vaan hän voi suoraan viitata olemassa olevaan ympäristömuuttujaan kehityksen aikana. Salasanoja ei saa koskaan jakaa niihin liittyvän käyttäjätunnuksen tai tunnusten kanssa. Salaisia arvoja ei saa lähettää selvänä tekstinä tai salaamattomana sähköpostina.

Integraatiokehityksessä käytetään ainoastaan salattuja siirtoprotokollia, kuten esimerkiksi https,sftp, jne. API-kehityksessä käytetään OAuth2:a tunnistautumiseen ja API-avainta valtuutukseen. IP-osoitteet suodatetaan, tapahtumien määriä rajoitetaan, virheellisen datan kulku estetään ja syöttöparametrit validoidaan.

Frends-integraatioalustalla ajettavat prosessit on mahdollista lokittaa, kirjata ja arkistoida. Tätä on kuitenkin vältettävä tuotannossa. Tätä ominaisuutta voidaan käyttää prosessin testaus- ja kehitysvaiheessa. Arkaluonteisia tietoja käsittelevät sovellusrajapinnat on suunniteltava ja toteutettava siten, että virheen sattuessa on mahdollista logata vain virhe, eikä viestin muuta sisältöä. Lokiin kirjattavat tiedot on valittava tarkasti, kun otetaan huomioon näiden tietojen luonne. Esimerkiksi jonkun henkilötunnusta ei kirjata lokeihin edes virheilmoituksen kautta.

Yleinen tietosuoja-asetus (GDPR) edellyttää, että yritykset suojaavat työntekijöidensä sekä asiakkaidensa ja kolmansien osapuolien PII-tietoja. PII-tietoina voidaan pitää esimerkiksi koko nimeä, puhelinnumeroa, tilinumeroa, kotiosoitetta ja henkilötunnusta.

3.3 Frendsin toimintatavat ja integraatiokäsikirja

Integraatiokäsikirja on Frendsin sisäinen toimintaohje, joka kattaa lähes kaikki asiat, jotka ovat integraatiokehittäjän työkalupakissa [7]. Tähän toimintaohjeeseen kuuluvat muun muassa dokumentaatio-ohjeet, ei-toiminnalliset vaatimukset integraatioille, Frendsin käyttö integraatioalustana, ICN-malli eli Integration Competency Network, integraation elinkaari sekä paljon muuta. Integraatiokäsikirjan tarkoitus on yhtenäistää yrityksen sisäistä toimintaa ja kartoittaa eräänlaiset ”best-practiset” eli hyväksi havaitut toimintatavat. Integraatiokäsikirja on myös asiakkaiden nähtävillä, jotta heillä on parempi ymmärrys Frendsin toiminnallisuuksista.

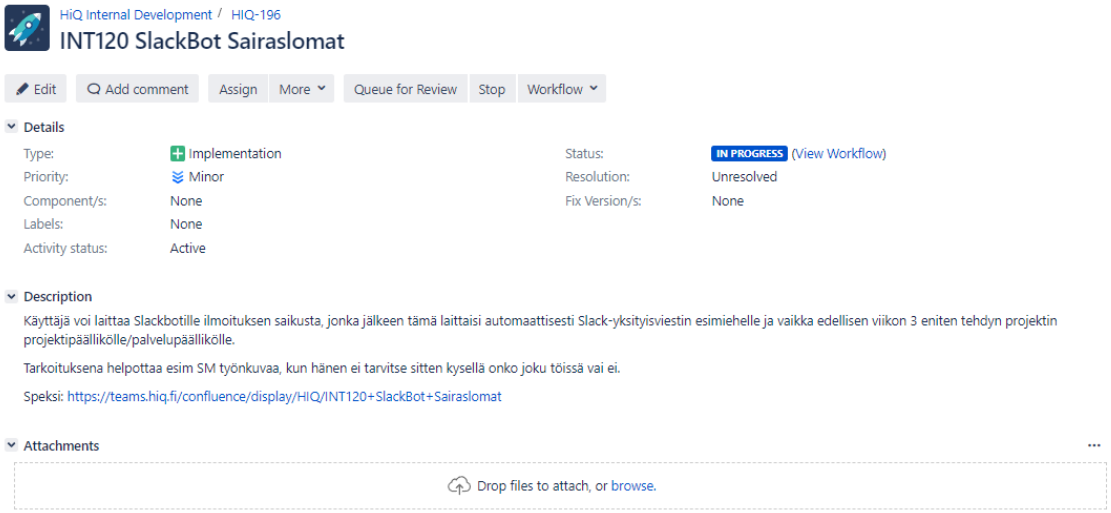
Tästä päästäänkin sujuvasti toimintatapoihin eli siihen, miten asiat tehdään Frends-integraatioalustalla. Koska kyseistä integraatioalustaa käyttää tuhansia kehittäjiä joko Frendsin tai sitten asiakkaiden puolelta, on tärkeää määrittää yhteiset toimintatavat, jotta kehittäminen olisi ketterämpää ja helpompaa. Frendsiä tämä otetaan erittäin vakavasti ja siitä syystä on luotu integraatiokäsikirja, jossa on nämä toimintatavat kerätty yhteen paikkaan.

4 Slackbot integraatioesimerkkinä

Käytännön esimerkkinä tutkitaan tarkemmin integraatiototeutusta nimeltä ”SlackBot Sairaslomat”. Tässä tehtävässä oli tavoitteena luoda integraatio, joka vastaa botin toiminnasta Slack-ympäristössä. Botin tehtävänä on vastaanottaa sairauspoissaoloilmoitus käyttäjiltä ja käyttää tätä lisätietojen hakemiseen Planmillistä, jonka jälkeen välittää nämä tiedot käyttäjän esimiehelle sekä projektipäälliköille.

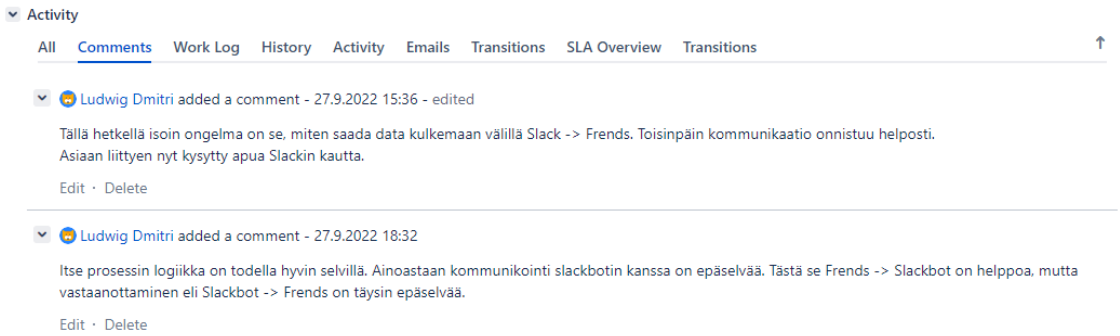
4.1 Integraation suunnittelu

Suunnittelu alkaa asiaan liittyvän tiketin luomisella tai tutkimisella (kuva 5). Yleensä tiketin luominen sekä integraation arkkitehtuurin suunnittelu kuuluu integraatioarkkitehdille (tämä yleensä siis kokenut kehittäjä).



Kuva 5 Integraatiotiketti [2].

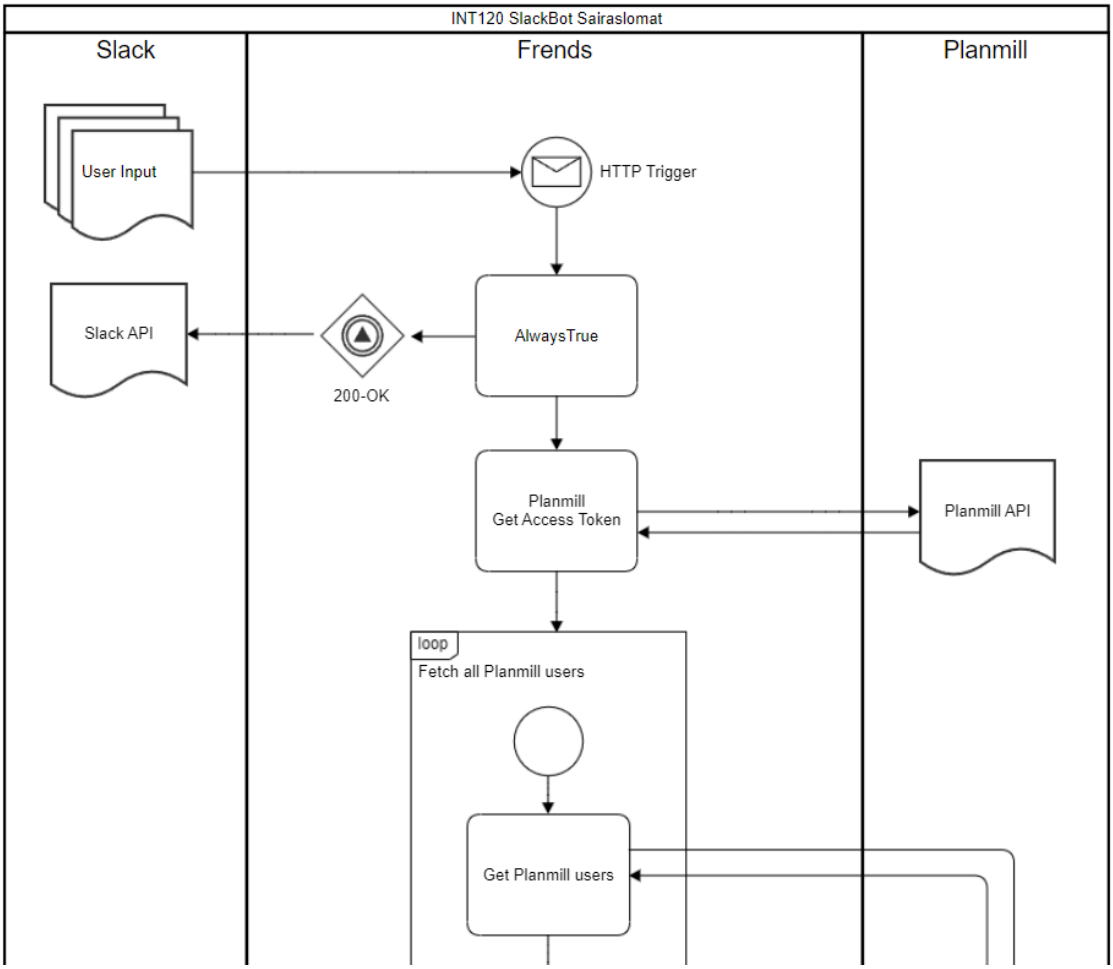
Kuvassa 5 määritellään ”Description”-kentässä, mikä on tehtävä ja tarkoitus saada aikaiseksi, sekä paljon kaikkea muuta. Tiketti on myös tärkeä kanava kommunikointia varten. Sinne on hyvä jättää kaikki integraatioon liittyvät huomiot ja ongelmat (katso kuva 6).



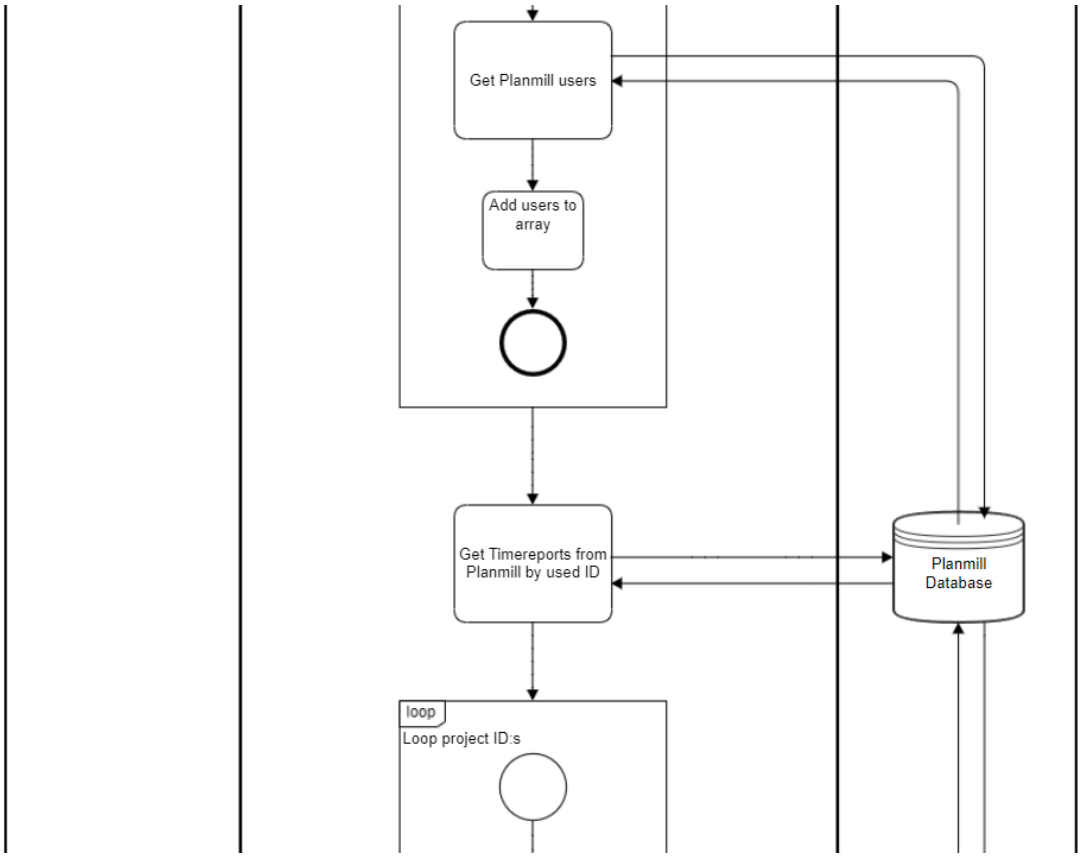
Kuva 6. Tiketin kommenttikenttä [2].

Kommenttikenttään jätetyistä kommenteista lähtee sähköposti-ilmoitus tiketin seuraajille. Yleensä nämä ovat projektipäällikkö ja integraation suunnitellut arkkitehti, jotka sitten voivat reagoita tarvittavalla tavalla.

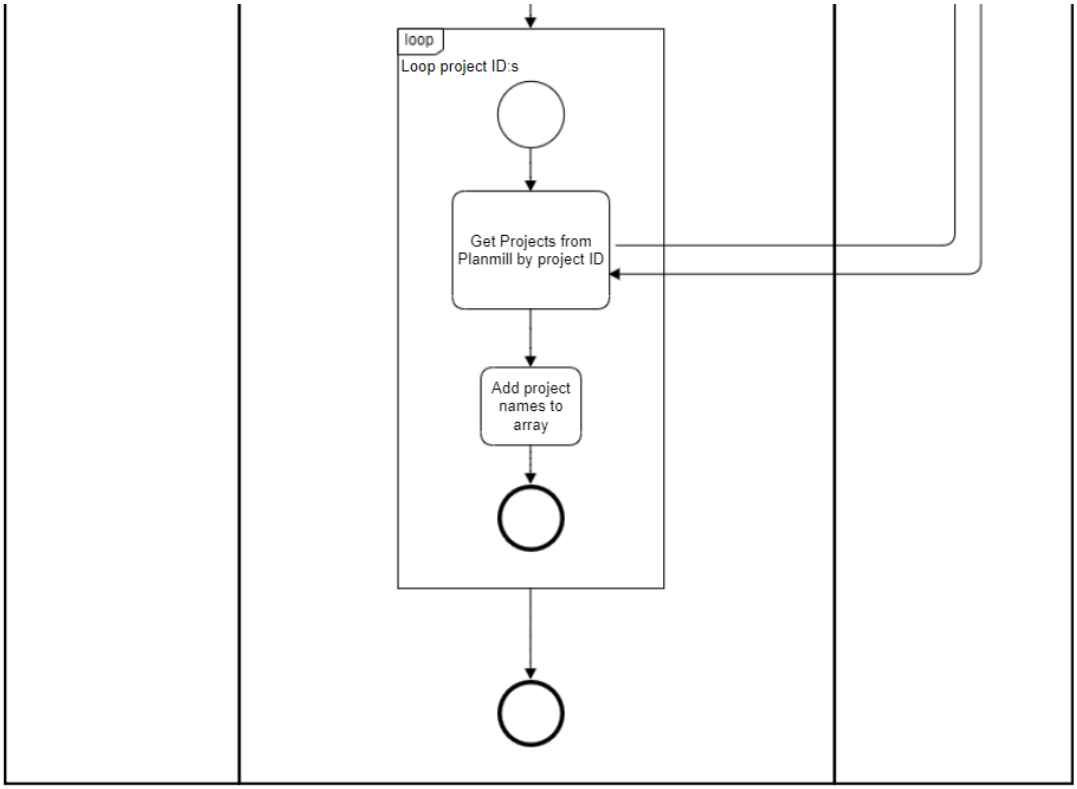
Tiketin yhteydessä myös luodaan dokumentaatiosivu Confluenceen. Integraation dokumentaatio on tärkeä osa toteutusta, sillä se mahdollistaa ulkopuolisten ohjelmoijien helpon ymmärryksen kyseiseistä integraatiosta. Dokumentaation oleellisin osio onkin BPMN-kaavio (katso kuvat 7,8,9), joka kuvastaa prosessin toiminnallisuutta.



Kuva 7. BPMN-kaavio osa 1 [2].



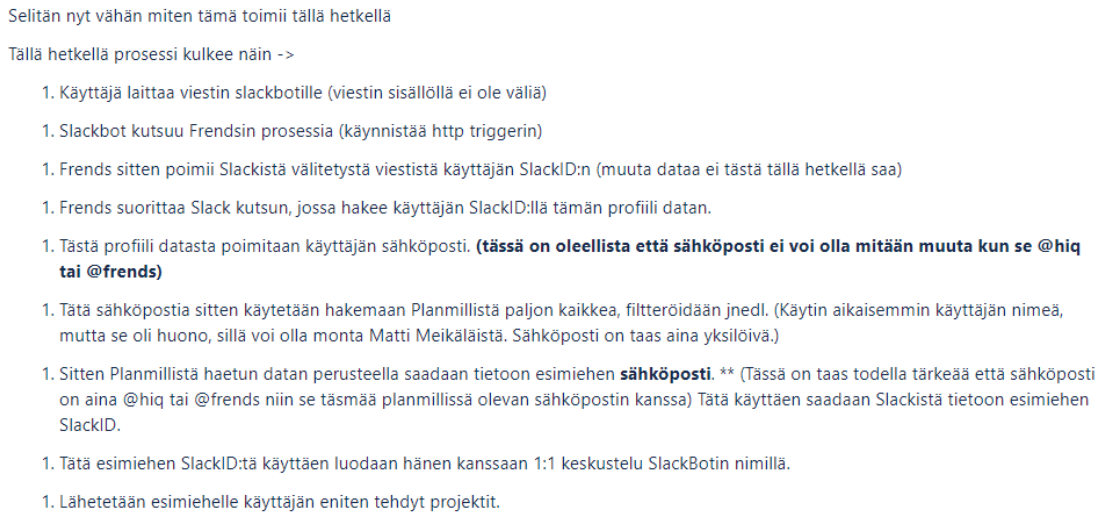
Kuva 8. BPMN-kaavio osa 2 [2].



Kuva 9. BPMN-kaavio osa 3 [2].

Kaavion lisäksi dokumentaatiossa kerrotaan integraation käyttämistä ympäristömuuttujista, kontakteista (kehen ottaa yhteyttä, jos joku ei toimi), tekninen kuvaus (lähdejärjestelmä, kohdejärjestelmä, mitä siirrettään), suoritettava mäppäys, nimeämiset sekä muu dokumentaatio (esim. excel-taulukko, word-pohja, jne).

Tällaisessä alustavassa integraation suunnittelussa tutkitaan, mistä haetaan dataa, mihin se siirretään, mitä dataa haetaan. Pyritään ennakoimaan tulevia ongelmia ja ratkaistaan ne vielä suunnitteluvaiheessa (esim. tunnukset palvelimiin). Tässä on esimerkki SlackBot-integraation suunnitteluvaiheesta. (katso Kuva 10).



Selitän nyt vähän miten tämä toimii tällä hetkellä

Tällä hetkellä prosessi kulkee näin ->

1. Käyttäjä laittaa viestin slackbotille (viestin sisällöllä ei ole väliä)
1. Slackbot kutsuu Frendsin prosessia (käynnistää http triggerin)
1. Frends sitten poimii Slackistä välitetystä viestistä käyttäjän SlackID:n (muuta dataa ei tästä tällä hetkellä saa)
1. Frends suorittaa Slack kutsun, jossa hakee käyttäjän SlackID:llä tämän profiili datan.
1. Tästä profiili datasta poimitaan käyttäjän sähköposti. **(tässä on oleellista että sähköposti ei voi olla mitään muuta kun se @hiq tai @frends)**
1. Tätä sähköpostia sitten käytetään hakemaan Planmillistä paljon kaikkea, filtteröidään jnedl. (Käytin aikaisemmin käyttäjän nimeä, mutta se oli huono, sillä voi olla monta Matti Meikäläistä. Sähköposti on taas aina yksilöivä.)
1. Sitten Planmillistä haetun datan perusteella saadaan tietoon esimiehen **sähköposti**. ** (Tässä on taas todella tärkeää että sähköposti on aina @hiq tai @frends niin se täsmää planmillissä olevan sähköpostin kanssa) Tätä käyttäen saadaan Slackistä tietoon esimiehen SlackID.
1. Tätä esimiehen SlackID:tä käyttäen luodaan hänen kanssaan 1:1 keskustelu SlackBotin nimillä.
1. Lähetetään esimiehelle käyttäjän eniten tehdyt projektit.

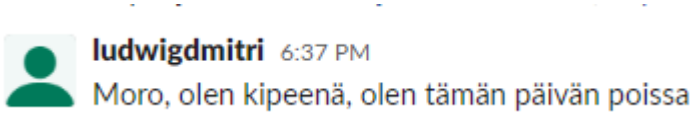
Kuva 10. Tulevan integraation vapaamuotoinen luonnostelu [2].

Kun suunniteluvaihe on valmis, siirrytään toteuttamaan itse integraatiota.

4.2 Integraation toteutus

Toteutus tapahtuu Frends-integraatioalustalla. Tavallisessa asiakkaalle tehtävässä työssä lähde- ja kohdejärjestelmien yhteystiedot saadaan asiakkaalta, integraatiokehittäjän tehtävä on rakentaa datan siirto paikasta A paikkaan B sekä datan muokkaaminen haluttuun muotoon. SlackBot-integraation tapauksessa kyseessä on kuitenkin sisäinen toteutus eli käyttäjänä on toteuttava taho itse.

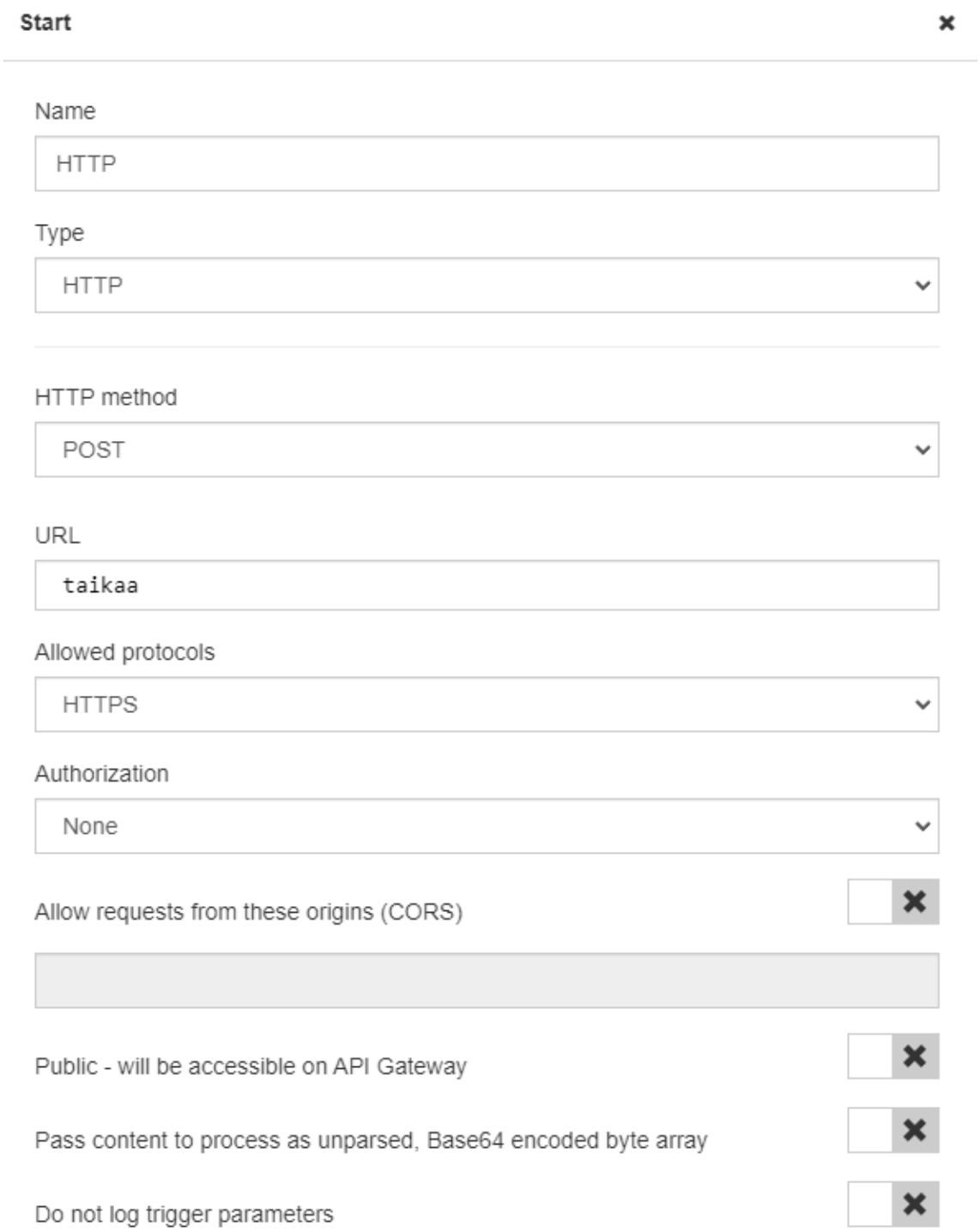
Tässä integraatiossa aloitettiin siitä, että selvitettiin, miten saadaan data kulkemaan Slackin ja Frendsin välillä. Pitää jotenkin saada käyttäjän Slackissä botille laittama viesti (katso kuva 11) välittymään Frendsiin, jossa integraatio tapahtuu ja prosessi ajetaan.



Kuva 11. Käyttäjä laittaa botille sairaspoissaolon Slackissä [2].

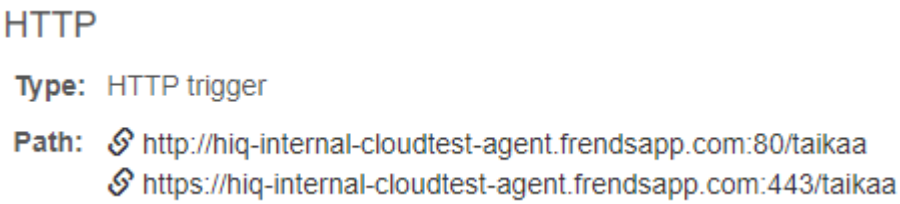
Tämä ongelma ratkaistaan rakentamalla Frends-prosessiin HTTP-triggeri (katso kuva 12 ja 13), joka käynnistyy, kun SlackBot kutsuu triggerillä määritettyä HTTPS -osoitetta.

Kuva 12. HTTP-triggeri vastaa prosessin aktivoinnista/aloituksesta [2].



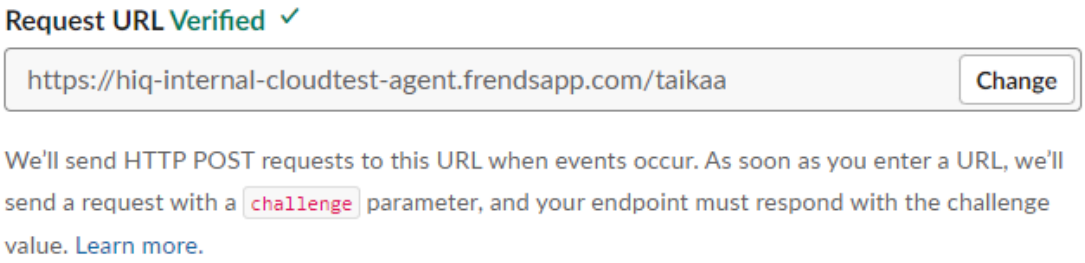
Kuva 13. HTTP-triggerille välitettävät parametrit [7].

Kuvan 13 tärkeimmät kohdat ovat nuo ”HTTP method” ja ”URL”, eli minkä tyyppinen kutsu vastaanotetaan, ja mistä osoitteesta. ”URL”-kentän arvo on käytännössä osoitteen prosessikohtainen osa. Osoitteen yhteinen osa määritellään ympäristön asetuksissa (katso kuva 14).



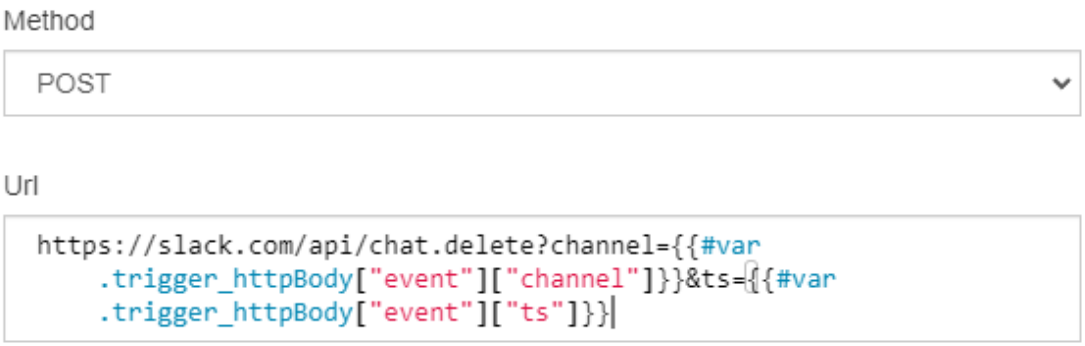
Kuva 14. HTTP triggerin käyttämä osoite [7].

SlackBotin asetuksissa myös määritellään, mihin osoitteeseen lähetetään kutsu (katso kuva 15), kun tietty toiminto (tästä lisää myöhemmin) tapahtuu.



Kuva 15. Osoitteen määrittäminen SlackBot-kutsua varten [7].

Näiden valmisteluiden jälkeen rakennetaan Frendsissä vastaanottava osuus. Kun Slackissä botille lähetetään viesti, botti ottaa siitä kopin ja kutsuu Frendsin prosessia, jossa vuorostaan käynnistyy tämän seurauksena HTTP-triggeri. Toteutuksessa oli monta haastetta vastassa Ensimmäinen niistä oli se, että piti turvata lähettäjän yksityisyys ja konfiguroida viestin poistaminen. Slack-ympäristön ominaisuuksien takia botin kanssa kommunikointi piti tehdä spesifille kanavalle. Käyttäjän yksityisyys varmennettiin siten, että viesti poistui heti lähetyksen jälkeen (kuva 16).

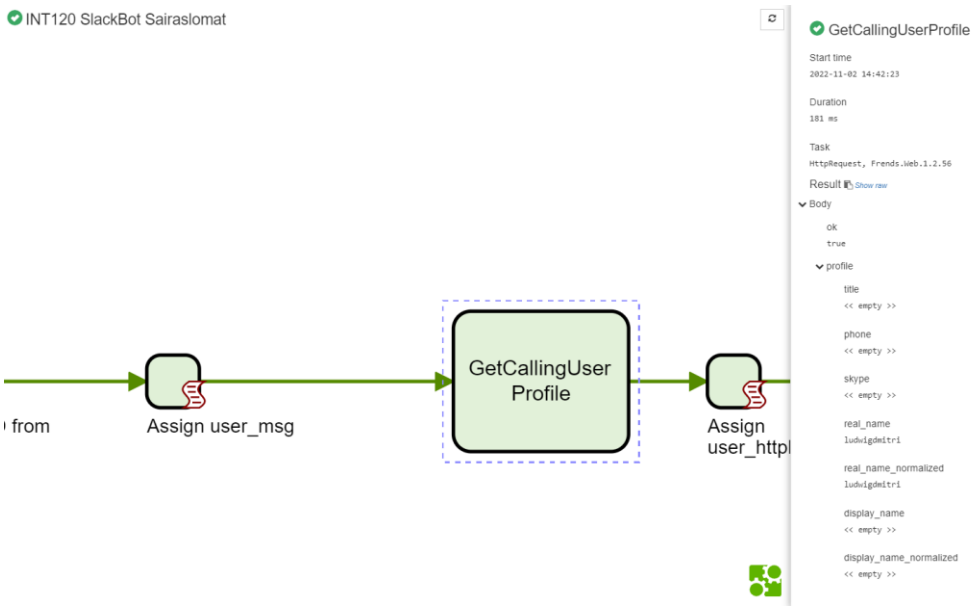


Kuva 16. POST-kutsu Frendsistä, joka hoitaa käyttäjän lähettämän viestin poistamisen. Kutsussa määritellään, miltä kanavalta poistetaan viesti (channel), sekä viestin aikaleima (ts) [7].

Viestin poistamisen jälkeen haetaan käyttäjän ID sekä hänen lähettämä viesti ja tallennetaan nämä väliaikaisiin muuttujiin. SlackBotilta tullut sanoma ei sisällä viestin lähettäneen käyttäjän sähköpostiosoitetta, sähköpostiosoite on juuri se asia, jota me tarvitsemme, jotta voisi suorittaa haun Planmillistä. Etu- tai sukunimellä hakeminen on huono tapa, sillä yrityksessä voi olla monta Matti Meikäläistä töissä. Sähköpostiosoite on puolestaan aina yksilöivä. Lähetetään siis SlackBotille takaisin seuraava kutsu:

https://slack.com/api/users.profile.get?user={{#var.calling_user_ID}}

Tässä kutsussa aaltosuluissa kutsutaan aikaisemmin luotua muuttujaa, johon on tallennettu käyttäjän SlackID. Tässä vaiheessa on tärkeää muistaa, että kyseessä on nimenomaan SlackID, myöhemmin puhutaan vielä PlanMillID:stä. SlackBot palauttaa meille käyttäjän profiilin, jossa on käyttäjän sähköpostiosoite (kuva 17).



Kuva 17. SlackBot:in vastaus käyttäjän profiilihakuun [7].

Seuraavaksi parsitaan vastaus ja tallennetaan se ”user_httpBody”-muuttujaan.

JToken.Parse(#result[GetCallingUserProfile].Body)

Tämän jälkeen otetaan talteen sieltä käyttäjän sähköpostiosoite tallentaen se ”user_email”-muuttujaan.

#var.user_httpBody["profile"]["email"]

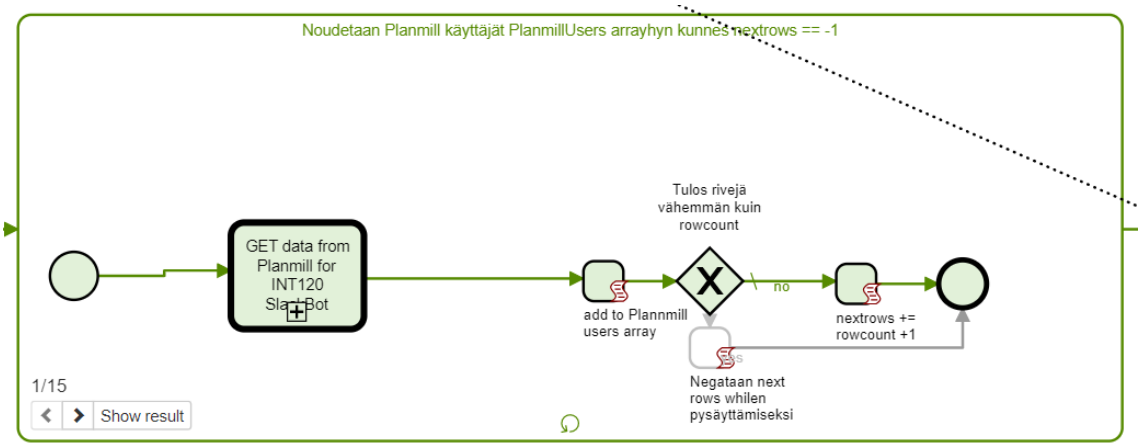
Nyt on tarkoituksena hakea Planmillistä sairaslomalle jäävän henkilön tekemät projektit. Kuulostaa yksinkertaiselta, mutta käytännössä tämä vaihe vei eniten kehitysaikaa. Planmill-rajapinnan käyttöön vaaditaan Access Token. Tämän hakemisen hoitaa aliprosessi. Tämä Access Token on voimassa 30 minuuttia, mikä riittää hyvin prosessin tarkoitukseen.

Planmill-data on rakennettu siten, että siellä on lista yrityksessä töissä olleista työntekijöistä, sekä myös niistä työntekijöistä, jotka ovat ulkoisia työntekijöitä (external). Tässä listassa on tietoja jokaisesta työntekijästä. Me haluamme talteen työntekijän esimiehen, koska prosessin lopullinen tarkoitus on lähettää käyttäjän sairauspoissaoloilmoitus tehtyjen projektien palvelupäälliköille. Tässä on vielä tärkeää huomata, että pelkkä esimiehen Planmill-ID ei riitä, vaan tarvitaan myös esimiehen sähköpostiosoite, joka ei löydy työntekijän alta, vaan itse esimiehen. Haetaan siis ensiksi sähköpostilla työntekijä, otetaan hänen esimieshenkilönsä Planmill-ID ja sillä haetaan itse esimies, josta sitten saadaan esimiehen sähköposti. Planmillistä tuleva data on suodatettava prosessin toiminnallisuuden nopeuttamiseksi sekä tietoturvan varmistamiseksi. Mitään ylimääräisiä tietoja ei ole syytä ”raahata” toteutuksessa mukana. Tämä tarkoittaa sitä, että Planmillistä tulevasta datasta pitää rakentaa lista seuraavilla ehdoilla.

- Työntekijän pitää olla aktiivinen, eli töissä.

- Työntekijällä pitää olla hiq.fi-päätteinen sähköpostiosoite (tällä tavalla suodatetaan pois ulkoiset työntekijät). Myöhemmin domain-muutoksen seurauksena tässä vaiheessa piti ottaa mukaan vielä frends.com-päätteiset sähköpostiosoitteet.

Suoritetaan Planmill-käyttäjien haku seuraavasti (kuva 18).



Kuva 18. Planmill-käyttäjien haku [7].

Tämän haun seurauksena saadaan jättimäinen lista tallennettuna ”planmillUsersArray”-nimiseen muuttujaan. Tästä eteenpäin alkaa hauskin osuus, eli datan suodattaminen haluttujen kriteerien mukaan. Vaikka Frends on low-code-integraatioalusta, joudutaan datan käsittelyyn monesti käyttämään isoja koodilohkoja tai LINQ-lauseita. Toimitaan siis näin:

- Suodatetaan pois työntekijät, jotka eivät ole töissä enää. Planmillin dokumentaation mukaan 1 tarkoittaa, että työntekijä on ”passiivinen”, kun taas 0 tarkoittaa, että työntekijä on ”aktiivinen”. Halutaan siis kaikki aktiiviset ja kenellä on sähköpostiosoitteet ”hiq.fi”-loppuinen. Tallennetaan nämä muuttujaan ”activeUsersArray”. Käytetään tässä LINQ-lausetta.

```
((JArray)#var.planmillUsersArray).Where(i => i["passive"].ToString() != "1" &&
!i["email"].ToString().ToLower().Contains("ext") &&
i["email"].ToString().ToLower().Contains("@hiq.fi")).Select(i => i).ToArray()
```

- Sitten muunnetaan tämä lista JArray:ksi ja lajitellaan työntekijät esimiesten mukaan selkeyden vuoksi. Tämä myös helpottaa tiedon saamista tästä listasta, kun voidaan suoraan viitata esimieheen (kuva 20).

Assign variable

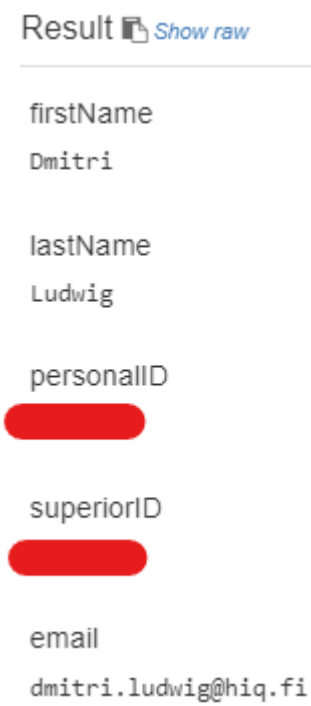
superior_array_sorted

Expression

```
from row in #var.jArray
let debug = #var.jArray.Where(i => !string.IsNullOrEmpty
    (i["superior"]?.ToString())).Select(i => i["superior"]
    ).Distinct().ToList()
where debug.Contains(row["id"])
select
    JObject.FromObject(new {
        superiorId = row["id"],
        superiorName = string.Format("{0} {1}",
            row["lastName"], row["firstName"]),
        superiorEmail = row["email"],
        subordinates =
            (from _row in #var.jArray
                where !string.IsNullOrEmpty(_row["superior"]
                    ?.ToString()) && int.Parse
                    (_row["superior"].ToString()) == int
                    .Parse(row["id"].ToString())
                select JObject.FromObject(new {
                    team = _row["teamName"],
                    id = _row["id"],
                    name = string.Format("{0} {1}",
                        _row["lastName"], _row["firstName"]
                        ),
                    email = _row["email"]
                })
            )
    }
)
```

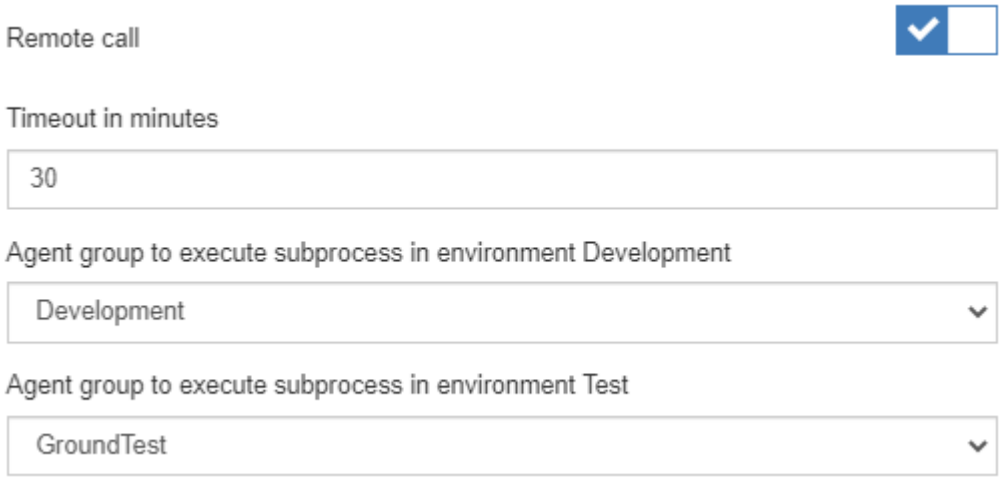
Kuva 20. Planmillistä tulevan datan suodatus ja tallentaminen ”superior_array_sorted”-muuttujaan [7].

Suodatuksen jälkeen meillä onkin 48 datarivin sijaan 5 tarvittavaa riviä hienosti listassa (kuva 21). Mitään ylimääräistä dataa ei raahata prosessissa mukana, vaan suodatetaan pois. Tästä vielä täydellisempi ratkaisu olisi oikean datan suodatus jo Planmillistä hakemisen yhteydessä, mutta valitettavasti Planmillin tarjoama rajapinta ei mahdollista tällaista toiminnallisuutta.



Kuva 21. Suodatettu data [7].

Nyt kun meillä on yhdistetty käyttäjän sähköpostiosoite PlanmillID:hen, sekä esimieheen ja hänen sähköpostiosoitteeseensa, voidaan jatkaa itse tuntimerkintä listan hakemiseen. Tässä listassa ovat kaikki työntekijöiden tekemät tuntimerkinnät koko työuralta. Tästä listasta tarvitaan käyttäjän edellisen viikon tekemät merkinnät, joista me puolestaan saamme tehtyjen projektien ID: t. Tässä vaiheessa tuli esille omituinen ongelma, jonka ratkomiseen meni muutama päivä. Ongelma johtui siitä, että Planmill on sisäverkon servereillä, jonka seurauksena Planmill-kutsut toimivat ainoastaan On-prem-palvelimella ja SlackBot pystyi vuorostaan kutsuja lähettämään ainoastaan pilvipalvelimella. Integraatio ei tietysti voi olla kahdessa ympäristössä samaan aikaan ja palomuurien asetusten muuttaminen todettiin vaaralliseksi. Ongelma ratkaistiin suorittamalla prosessi pilviympäristössä ja ulkoistamalla kaikki Planmill-kutsut aliprosessiksi, joka pyörii vuorostaan On-prem-ympäristössä ja jota kutsutaan käyttäen Frendsin ”remote call”-ominaisuutta (kuva 22).



Kuva 22. ”Remote call” -ominaisuus [7].

”Remote call” -ominaisuudella on mahdollista määritellä, missä ympäristössä suoritetaan kyseessä oleva aliprosessi. Tässä tapauksessa kuvassa 22 viimeisellä rivillä on määritelty, että aliprosessi suoritetaan GroundTest-nimisessä On-prem-ympäristössä.

Ongelman ratkaisun jälkeen suoritetaan tuntiraportti haku Planmillistä.

```
{{#env.General_PlanMill.baseURL}}/timereports?person={{#var.sorted_data_2[0]["personalID"]}}&rowcount=9999&period=31
```

Tässä haussa osoite haetaan ympäristömuuttujista ja lisätään parametrit haulle: person = sairauspoissaoloilmoituksen ilmoittajan Planmill-ID, laitetaan haettavien rivien määrä tappiin (rowcount = 9999) sekä määritellään aikaväli, jolta haetaan (31 tässä tapauksessa tarkoittaa viime viikkoa). Planmillistä palautuu seuraava data (kuva 23).

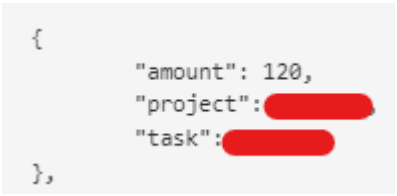
```
"Body": [
        {
                "unitPrice"[redacted]
                "dutyType": 0,
                "amount": 120,
                "start": "2022-10-28T12:38:00+00:00",
                "project":[redacted],
                "billableStatus": 5,
                "billableAmount": 120,
                "overtimeAmount": 0,
                "task":[redacted],
                "billingComment": null,
                "person":[redacted]
                "modified": "2022-11-02T10:33:38.39+00:00",
                "comment": [redacted] Review uudelle prosessille",
                "overtimeComment": "",
                "finish": "2022-10-28T14:38:00+00:00",
                "modifiedBy"[redacted]
                "id":[redacted],
                "invoice":[redacted]
                "travelComment": "",
                "travelAmount": 0,
                "status": 1
        },
```


Kuva 23. Esimerkki yksittäisestä Planmillistä palautuvasta tuntikirjausdatasta [7].

Tässä vaiheessa taas todetaan, että tässä on aivan liian paljon dataa, jolla me emme tee mitään, joten otetaan oleellinen ja jätetään muu pois.

```
((IEnumerable<dynamic>)#result[Get Timereportss from Planmill by user ID].Body).Select(i =>
JObject.FromObject(new {
    amount = i.amount,
    project = i.project,
    task = i.task
})).ToList()
```

Poimitaan, kuinka paljon aikaa on käytetty (amount), projekti, jota on tehty (project) sekä mihin tehtävään kyseinen merkintä kuuluu (task). Yhdellä projektilla voi siis olla monta eri tehtävää alaisuudessa. Sujuvasti saadaan 21 rivin datakokonaisuudesta kolmen rivin oleellinen kokonaisuus (kuva 24).

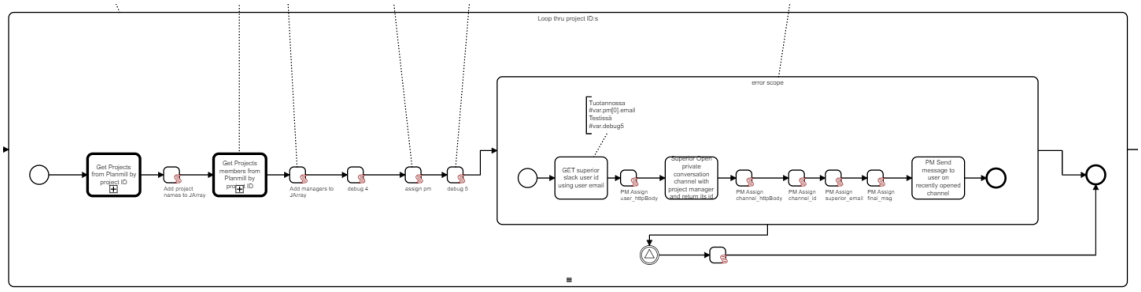


Kuva 24. Suodatuksen tulos [7].

Sitten yhdistetään kaikki samalle ”task”:ille kuuluvat merkinnät. On siis hyvin tavallista, että käyttäjä tekee samaa ”task”:ia monena päivänä ja siitä siten on monta merkintää, jotka pitää yhdistää.

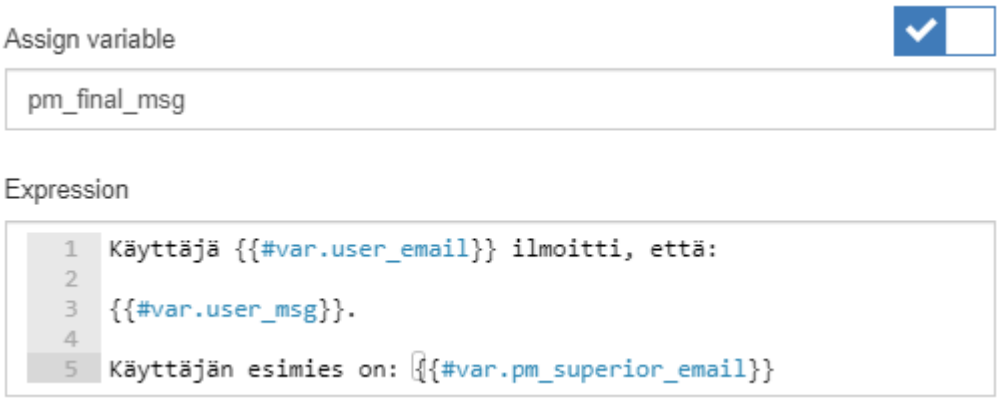
```
((IEnumerable<dynamic>)#var.hours)
.GroupBy(l => l.project)
.Select(cl => JObject.FromObject(new {
   amountcombined = cl.Sum(c => int.Parse(c.amount.ToString())),
   project = cl.First().project,
})).ToList()
```

Tämä ei vielä riitä saavuttamaan määränpäätä. Planmillistä pitää vielä hakea tehtyjen projektien nimet sekä näiden palvelupäälliköt (tarkoituksenahan on välittää käyttäjän sairauspoissaoloilmoitus tehtyjen projektien esimiehille). Tämä saavutetaan muutamalla Planmill-haulla, jotka suoritetaan ”foreach”-loopin sisällä, jonka ehtona käytetään edellisessä haussa saatujen projektien ID-arvoja. Eli toistetaan niin monta kertaa kun, projekti ID:t löytyvät (kuvan 25 ensimmäinen puolisko).



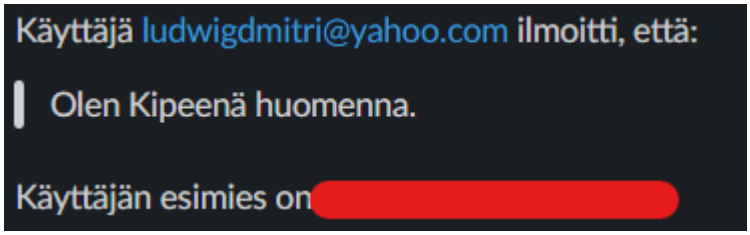
Kuva 25. Vasemmalta oikealle: kaksi Planmill-hakua ”foreach”-luupin sisällä, muuttujien alustaminen, jonka jälkeen esimiesten SlackID:n haku, keskustelun avaaminen Slackissä sekä viestin lähettäminen [7].

Kun projektien nimet ja vastuuesimiehet on haettu, haetaan vielä näiden esimiesten SlackID:t. Nämä tarvitaan tietysti sitä varten, että SlackBot tietää, kenelle työntekijän sairauspoissaoloilmoitus lähetetään. Ainut asia, jolla tämän voi tehdä on esimiehen sähköpostiosoite, sillä esimiehen PlanmillID ei tähän käy (Slack käyttää omia ID-arvoja). Tämä haku suoritetaan kuvan 31 puolessa välissä, sisemmän luupin ensimmäisenä toiminnallisuutena. Haettua esimiehen SlackID välitetään tämä SlackBotille ja annetaan käsky avata yksityinen keskustelu tämän esimiehen kanssa, kun palautetaan sen keskustelun ID-arvo. Rakennetaan esimiehelle tarkoitettu viesti (kuva 32) ja käyttäen tätä saatua keskustelun ID-arvoa käsketään SlackBottia lähettämään esimiehelle käyttäjän sairauspoissaoloilmoitus.



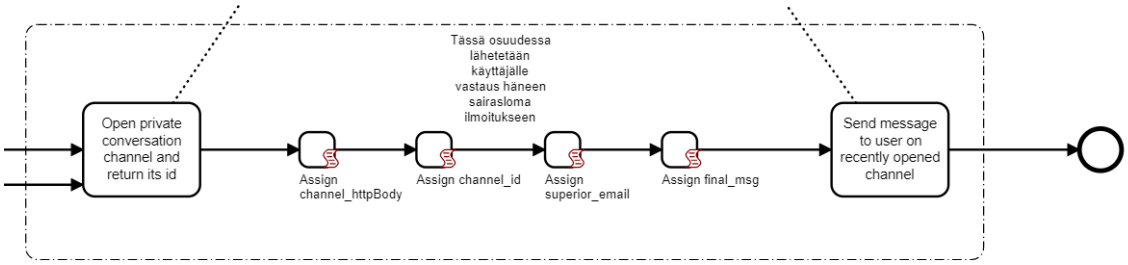
Kuva 26. Esimiehelle tarkoitetun viestin muodostaminen [7].

Lopputuloksena esimies saa seuraavan viestin (kuva 27).



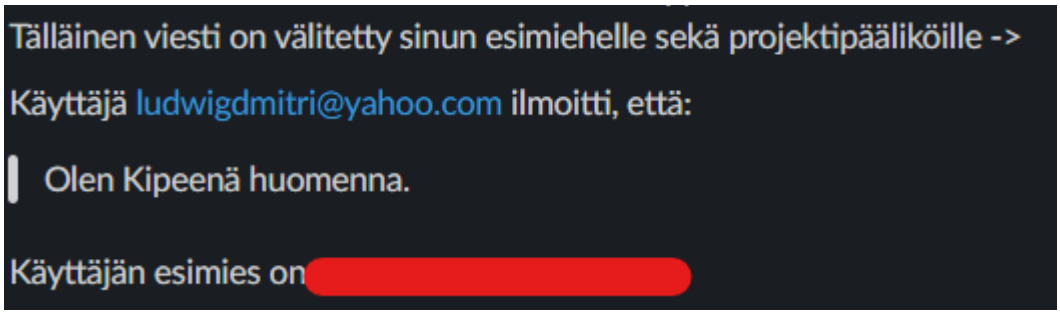
Kuva 27. Viesti, jonka esimies saa [8].

Jäljellä on vielä viimeinen asia, on kuitattava käyttäjälle, että SlackBot on välittänyt viestin sekä viestin sisällön. Tämä tehdään prosessin lopussa, kun kaikki muut toiminnot on tehty (kuva 28).



Kuva 28. Kuittaus käyttäjälle, että viesti on lähetetty, sekä viestin sisältö [7].

Käytännössä kuvio on samanlainen kuin esimiesten kanssa. Käyttäjän SlackID saadaan alkuperäisestä sairauspoissaoloilmoitus prosessin alusta. Eli avataan yksityinen keskustelu käyttäjän kanssa käyttäen hänen SlackID:tä, palautetaan siitä keskustelun ID-arvo, muodostetaan viesti ja lähetetään se käyttäjälle käyttäen tätä ID-arvoa. Käyttäjälle tulee käytännössä seuraava viesti (kuva 29).



Kuva 29. Käyttäjälle välittyvä kuittaus lähetetystä viestistä [8].

4.3 Integraation testaaminen/katselmointi

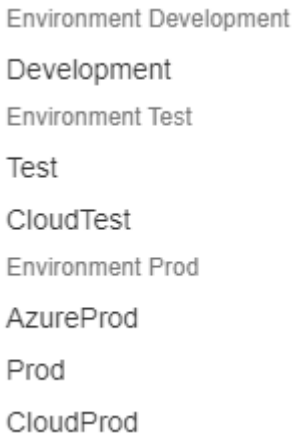
Kun kehittäjä ilmoittaa, että integraatio on valmis, nimetään joku toinen, kyseisen prosessin kehityksestä ulkopuolinen kehittäjä katsomaan prosessin läpi perehtyen sen toimintaan sekä tarkistamaan, että prosessi vastaa määrittelyssä olevia asioita. Tätä kutsutaan prosessin katselmoimiseksi, tai revikoimiseksi, joka on johdettu englanninkielisestä sanasta ”review”. Prosessin katselmointia varten on olemassa ohjeistus (kuva 30), mutta käytännössä tämä on hyvin vapaamuotoista ja jokainen tekee sen omalla tavallaan.

Kuva 30. Katselmointiohjeistus [3].

Katselmoija on siis toteutuksesta ulkopuolinen henkilö ja hänen on tarkoitus perehtyä toteutukseen sekä varmistaa, että toteutus vastaa sitä, mitä on siltä vaadittu. Vaatimukset löytyvät prosessin elinkaaren alussa tehdystä määrittelydokumentista sekä kehittämisen aikana syntyneistä parannuksista ja ratkaisuista. Katselmoija myös usein huomaa kaikenlaisia korjattavia kohtia, kuten kirjoitusvirheitä, tai loogisia virheitä. Myös prosessissa käytettyjen muuttujien nimet pitää olla ymmärrettävät ja asiaan kuuluvat.

Katselmoija ei itse suorita mitään korjauksia vaan kirjaa nämä ylös ”katselmointikommenttiin”, joka sitten välitetään toteutuksen suorittajalle. Tämä taas suorittaa tarvittavat korjaukset. Tämä tehdään näin siitä syystä, että katselmointikommentti toimii eräänlaisena opetushetkenä prosessin kehittäjälle.

Testausta suorittaa prosessin kehittäjä kehityksen aikana tarpeiden mukaan. Testaus tapahtuu lähes aina testiympäristössä. Testiympäristön lisäksi on usein olemassa vielä ”Development-” sekä ”Production”-ympäristöt (kuva 31). Ympäristöjen nimeämiset ovat asiakaskohtaisia, joten niissä on eroja. Ympäristöt voiva sijaita joko pilvessä (esim. CloudTest) tai sitten ”on-prem” eli paikan päällä. Tämä tietysti vaikuttaa yhteyksien luomiseen sekä prosessin asetuksiin.



Kuva 31. Ympäristöluettelo [7].

Tästä itse kehittäjän suorittamasta testauksesta on kuitenkin vielä erillinen ”Testausvaihe”, joka tapahtuu prosessin katselmointivaiheen jälkeen. Tässä vaiheessa prosessin kehittäjä yhteistyössä prosessin arkkitehdin kanssa sekä mahdollisen asiakkaan kanssa suorittaa täyden testauksen, eli testaa prosessin kaikki toiminnot, kuten esim. yhteydenotot tietokantoihin, prosessin loogisen kokonaisuuden ja mäppäyksen toiminnallisuuden. Tässä vaiheessa selviävät virheet aiheuttavat prosessin palautuksen takaisin kehittäjälle, jonka jälkeen prosessin on käytävä uudestaan katselmoinnin kautta. Toki seuraavilla kerroilla katselmoidaan ainoastaan prosessin muuttunutta osaa, ei koko prosessia niin kuin ensimmäisessä katselmointikerrassa. Eli ensimmäisen katselmointikerran jälkeiset kerrat menevät hyvin nopeasti.

Testauksessa käytetään yksikkötestausta, joka on iso osa toteutusta. Yksikkötestauksen tavoitteena voi olla esimerkiksi testata mäppäyksen toimivuutta, tiedon rikastusta sekä varmistaa, että kohdejärjestelmälle lähtevät sanomat ovat oikeassa muodossa. Testauksien onnistunutta suoritusta varten onkin oleellista, että esimerkkiviestit lähdejärjestelmästä ja kohdejärjestelmästä ovat olemassa. Ajateltavia asioita esimerkkiviesteistä voivat olla seuraavat asiat. Minkälaista viestiä lähdejärjestelmä lähettää? Onko tämä viesti SOAP-sanoma? Vai XML-muotoinen sanoma? Minkälaista sanomaa kohdejärjestelmä vastaanottaa?

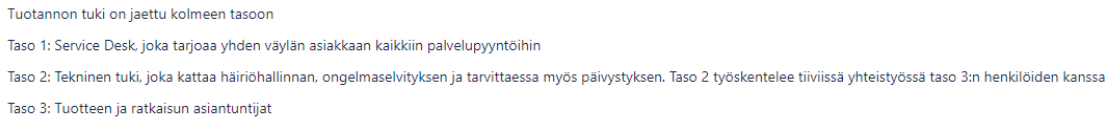
Kaikki yllä mainitut vaiheet ovat erittäin tärkeitä tuotteen laadun varmistamiseksi. Testausvaiheen hyväksymisen jälkeen siirrytään seuraavaan vaiheeseen eli integraation tuotantoon vientiin.

4.4 Integraation tuotantoon vieminen

Integraation tuotantoon viennin aikataulu sovitaan projektin alussa. Tietysti on tärkeää pysyä sovitussa aikataulussa, mutta mikäli muutoksia aiheutuu ja ongelmia tulee vastaan, niin on erittäin tärkeää ilmoittaa niistä asiakkaalle ja siirtää aikataulua sen mukaisesti. Keskeneräistä prosessia ei koskaan viedä tuotantoon. Jos projekti koostuu useimmasta prosessista, niin on mahdollista ottaa ne käyttöön yksitellen, tai sitten isona kokonaisuutena. Ennen tuotantoon vientiä kehittäjän on kuvattava ohjeistus

- Mitkä integraatiot kuuluvat käyttöönottoon ja mikä versio asennetaan?
- Mitkä ovat asennettavat asiat (esim. prosessin nimi, ulkoisia kirjastoja, SQL proseduurit)?
- Missä järjestyksessä ja miten asennukset tulee tehdä?
- Kuinka asennus voidaan tarvittaessa peruuttaa?

Tälläinen ohjeistus mahdollistaa sen, että kuka tahansa voi tehdä asennukset ohjeita noudattamalla. On myös varmistettava, että on olemassa operointiohjeet. Operointiohjeet (kuva 32) ovat prosessikohtaisia ohjeita, joissa käytännössä kuvataan, miten toimitaan, jos prosessissa tapahtuu virheitä tuotannon viennin jälkeen.



Kuva 32. Operointiohjeiden koostumus [3].

Usein tuotannon viennin jälkeen prosessi on toiminnallisena vuosia. Joten operointiohjeiden olemassaolo on tärkeää, jotta asiakastuen työntekijät osaavat käsitellä mahdolliset prosessissa tapahtuvat virheet. Käytännössä operointiohjeissa kerrotaan, mitä tehdään, kun tapahtuu virhe, esimerkiksi:

- Saako prosessin ajaa uudestaan?

- Mitä tehdään prosessissa siirrettäville tiedostoille, mikäli prosessi kaatuu kesken suorituksen?

Voi olla mahdollista, että jonkin tiedoston siirtäminen jää kesken prosessin kaatumisen takia, ja sitä tiedostoa ei saa säilyttää, koska se sisältää GDRP-alaista dataa. Normaalitilanteessa prosessi hoitaisi tämän tiedoston poistamisen, mutta tässä poikkeustilanteessa operointiohjeisiin on kirjattava, että ”prosessin kaatuessa on manuaalisesti poistettava tämä tiedosto paikasta x”. Tämä mahdollistaa sen, että asiakastuen työntekijät osaavat käydä tämän toiminnon tekemässä, vaikka heillä ei muuten ole prosessin toiminnallisuudesta ymmärrystä.

Kun on todettu, että mikä kokonaisuus tuotantoon viedään, ja että operointiohjeet ovat olemassa, prosessi siirretään tuotantoympäristöön. Tämän siirron mukana siirretään myös kaikki prosessin käyttämän aliprosessit tuotantoon. Tämän vaiheen jälkeen edetään jälkitoimenpiteisiin.

5 Jälkitoimenpiteet

5.1 Tuotannossa olevan integraation seuranta

Tuotannossa olevaa integraatiota seurataan ensimmäisten päivien aikana hyvin tarkasti, sillä testiympäristön testaukset on suoritettu testipalvelimilla testidatan kanssa. Tämä tarkoittaa, että mikäli tuotannossa kulkeva data eroaa testissä olevasta datasta rakenteeltaan, voi tämä aiheuttaa ennakoimattomia virheitä. Tietysti seurataan, että testidata on rakenteeltaan samanlainen, kun todellisuudessa siirrettävä data, mutta valitettavasti viime hetken muutoksia sekä virheitä sattuu.

Prosessin ensimmäiset suoritukset tuotantoon viennin jälkeen ovat kaikesta jännittävämpiä hetkiä. Kun todetaan, että ”hyvin tämä toimii” ja asiakas on tyytyväi-

nen saamansa tulokseen, niin prosessi siirtyy kuvan 25 mukaisesti taso 1 seurantaan, eli Service Deskille kuuluu seuranta, että kaikki toimii niin kuin pitää. Mikäli virheitä tapahtuu tai jokin asia ei toimi niin kuin pitää, niin asiakasta ohjeistetaan ottamaan yhteyttä Service Deskiin. Toki prosessissa tapahtuvat virheet ohjautuvat automaattisesti Service Deskiin.

5.2 Tuotannossa olevan integraation virhetilanteet ja niiden hallinta

Käytännössä virhetilanteet ja niiden hallinta kuuluvat kuvan 25 mukaisesti Service Deskille sekä heidän kauttansa tekniselle tuelle. Mikäli tuotannossa prosessissa tapahtuu virhe, ilmoitus siitä menee Service Deskille, jossa virhe käsitellään operointiohjeiden mukaisesti saman tien. Service Desk ei tätä virhettä kuitenkaan korjaa, vaan luovat virheestä tiketin ja välittävät sen kuvan 25 mukaisesti tekniselle tuelle. Siellä virhe tutkitaan ja korjataan, mikäli korjaus onnistuu helposti. Jos ei niin tiketti välitetään eteenpäin integraatiokehittäjille eli siis sinne, missä tämä prosessi on alun perin luotu.

Näiden kaikkien vaiheiden suoritettavuus toki riippuu siitä, mitä asiakkaalle on myyty. Kaikki asiakkaat eivät osta teknistä tukea, vaan hoitavat sen itse, jolloin virheilmoitukset prosessista ohjataan heille suoraan.

6 Yhteenveto

Frends-integraatioalusta taipuu erittäin hyvin kaikkiin nykymaailman yritysten tarpeisiin, oli se sitten datan siirto, datamuunnokset, rajapintojen luonti tai prosessien automatisointi. Low-code-ympäristö mahdollistaa ketterän kehityksen tiukalla aikataululla. Integraatioiden elinkaari pitää huolen siitä, että tuotteen laatu varmistetaan ennen asiakkaalle toimittamista, sekä myös toimittamisen jälkeen.

Slackbot-esimerkkiprosessin kehittäminen oli erittäin mielekäs kokemus, ja omasta mielestäni siitä saa hyvän käsityksen siitä, kuinka moniulotteista

Frends-integraatioalustalla kehittäminen on. Ongelmia kehityksessä tulee vastaan aina, ja niiden ratkominen on joskus turhauttavaa, mutta aina vaivansa arvoista. Paljon saa positiivisia tunteita, kun toimittaa asiakkaalle valmiin integraation ja saa siitä positiivista palautetta.

Jos vertailee tällaistä Low-code-integraatioalustaohjelmointia perinteiseen olio-ohjelmointiin, niin voidaan todeta, että integraatioalustalla toteutuksien luominen on nopeampaa ja ketterämpää. Monia käytettäviä luokkia ja funktioita on ulkoistettu omiksi Frends-toiminnallisuuksiksi, mikä mahdollistaa niiden nopean kutsumisen tarvittaessa. Kehitystä myös helpottaa, kun kaikki ohjelmoijat käyttävät samoja työkaluja samalla alustalla. Pienellä kynnyksellä voi kysyä apua tarvittaessa. Myös virhetilanteiden korjaaminen on helpompaa. Integraatioalustalla kehitetyn ratkaisun ymmärtäminen on helpompaa kuin perinteisen ohjelman. Jokaisesta ulkoistetusta Frends-toiminnallisuudesta on olemassa dokumentaatio, johon voi perehtyä. Erillisten toiminnallisuuksien kommentointi ja dokumentointi on tehty helpoksi ja nopeaksi.

Lähteet

1 HiQ-kotisivut https://hiq.fi/ , Frends integraatioalustan kuvaukset. Verkkoaineisto. Luettu 29.8.2022.

2 HiQ-yrityksen sisäiset Confluence-sivut. Verkkoaineisto. Luettu 10.10.2022.

3 HiQ-yrityksen sisäinen integraatiokäsikirja. Verkkoaineisto. Luettu 6.9.2022.

4 Frends-dokumentaatio https://docs.frends.com/en/ best practices. Verkkoaineisto. Luettu 2.11.2022.

5 Wikipedia https://fi.wikipedia.org/ , lyhenteiden määritykset. Verkkoaineisto. Luettu 3.10.2022.

6 Frends-kotisivut https://frends.com/ Verkkoaineisto. Luettu 24.10.2022.

7 Frends-integraatioalusta. Verkkoaineisto. Luettu 27.9.2022.

8 Slack https://slack.com/ Verkkoaineisto. Luettu 19.9.2022.