Tatu Pulkkinen

Testitapauspohjaisen alustusprosessin kehitys Frends-integraatioalustalle

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

1.11.2023

Tiivistelmä

Tekijä:	Tatu Pulkkinen
Otsikko:	Testitapauspohjaisen alustusprosessin kehitys Frends-integraatioalustalle
Sivumäärä:	47 sivua + 7 liitettä
Aika:	1.11.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Vesa Ollikainen

Tämän opinnäytetyön tarkoituksena oli kehittää testitapauspohjainen alustusprosessi Frends-integraatioalustalle, jonka avulla tuettaisiin ohjelmistotestauksen eri osa-alueita.

Työn teoriaosuudessa tutustuttiin hieman Frends-integraatioalustaan ja tutkittiin alustalta löytyviä ohjelmistotestauksen eri osa-alueita ja selvitettiin mahdollisia kehityskohteita. Tämä toteutettiin hyödyntäen saatavilla olevia lähteitä ja kokemusta Frends-integraatioalustalla työskentelystä.

Opinnäytetyön tuloksina syntyi yleiskäyttöinen testitapauspohjainen alustusprosessi Frends-integraatioalustalle, jonka avulla testauksien dokumentointiakin voidaan helpottaa. Lisäksi ohjelmistotestaukseen liittyen saatiin selville mahdollisia kehityskohteita Frends-integraatioalustalta.

Avainsanat: testitapauspohjainen, ohjelmistotestaus, Frends-integraatioalusta

Abstract

Author:	Tatu Pulkkinen
Title:	Development of a Test Case-Driven Initialization Process for the Frends Integration Platform
Number of Pages:	47 pages + 7 appendices
Date:	1 November 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Vesa Ollikainen, Senior Lecturer

The purpose of this thesis was to develop a test case-based initialization process for the Frends integration platform to support various aspects of software testing.

In the theoretical part of the work, we briefly introduced the Frends integration platform and examined the different areas of software testing available on the platform. We also identified potential areas for improvement. This was done by utilizing available sources and drawing upon experience working with Frends integration platform.

As a result of the thesis, a versatile test case-based initialization process for the Frends integration platform was developed, which can also make the documentation of testing a lot easier. Additionally, in relation to software testing, potential areas for improvement were identified on the platform.

Keywords: test case-driven, software testing, Frends integration platform

Sisällys

Lyhenteet

BPMN: Business Process Model and Notation. Graafinen esitystapa liiketoimintaprosessista. Sen avulla voidaan helposti havainnollistaa ohjelmiston tai muun osan toiminnallisuus ilman teknistä osaamista.

FTP: File Transfer Protocol. Tiedostojen siirtoprotokolla kahden tietokoneen välillä, joka hyödyntää TCP-protokollaa. Tiedostot siirretään avoimesti ja ilman salausmekanismeja.

JSON: JavaScript Object Notation. Yksinkertainen tiedostomuoto, jonka kirjoittaminen ja lukeminen on ihmiselle helppoa. Käytetään usein tiedonvälitykseen ja tallentamiseen.

JToken: Abstrakti perusluokka, jonka avulla JSON-formaatin jäsentäminen on kannattavaa, jos sen rakenteesta ei ole varmuutta.

SFTP: Secure File Transfer Protocol. Tiedostojen siirtoprotokolla, joka tunnetaan myös SSH File Transfer Protocolin nimellä. Turvallisempi valinta verrattuna FTP-protokollaan, sillä se hyödyntää SSH:ta tiedonsiirron suojaamiseen.

SMB: Server Message Block. Verkkoprotokolla, jonka avulla voidaan jakaa tiedostoja ja resursseja tietokoneiden välillä verkon yli. SMB-protokollan avulla tiedostoja voidaan lukea, kirjoittaa ja jakaa verkossa.

SSH: Secure Shell. Salaustekniikkaa ja verkkoprotokollaa yhdistävä työkalu. Käytetään usein turvalliseen etäyhteyksien luomiseen ja tiedostojen siirtämiseen verkkoyhteyden yli.

TCP: Transmission Control Protocol. Vikasietoinen tietoliikenneprotokolla, jonka avulla voidaan siirtää tietoa luotettavasti tietokoneiden välillä.

1 Johdanto

Tässä opinnäytetyössä kehitetään testitapauspohjainen tiedostojen alustusprosessi, jonka tarkoituksena on tukea ohjelmistotestauksen eri vaiheita ja näin parantaa tiedostopohjaisten Frends-integraatioiden testausta. Opinnäytetyössä on myös tarkoitus tutkia ohjelmistotestauksen käsitettä, tutustua sen eri osa-alueisiin, selvittää, miten nämä näkyvät Frends-integraatioalustalla, ja samalla tavoitteena on selvittää mahdollisia kehityskohteita.

Ohjelmistotestaus on olennainen osa ohjelmistokehitystä ja ohjelmistojen monimutkaisuuden kasvaessa on tärkeää varmistaa, että ohjelmisto toimii odotetulla tavalla ja täyttää vaatimukset. Tämä vaatii huolellista testausta eri tasoilla ja eri näkökulmista.

Keskeisessä osassa opinnäytetyötä on testitapauspohjaisen tiedostojen alustusprosessin toteutus Frends-integraatioalustalla, jonka tarkoitus on tukea ohjelmistotestausta.

Opinnäytetyö jakautuu neljään osaan. Ensimmäisessä osassa tutustutaan Frends-integraatioalustaan ja sen ominaisuuksiin. Toisessa osassa tarkastellaan yleisesti ohjelmistotestauksen käsitettä ja selvitetään siihen kuuluvia eri osa-alueita. Samalla selvitetään, miten nämä ohjelmistotestauksen eri osa-alueet näkyvät Frends-integraatioalustalla ja pyritään selvittämään mahdollisia kehityskohteita. Kolmannessa osassa käsitellään testitapauspohjaisen alustusprosessin kehittämistä, mikä sisältää ohjelmistokehityksen olennaiset vaiheet. Työn viimeisessä luvussa käydään vielä yhteenvetona läpi työn tavoitteet, esitellään työn tuloksia ja selvitetään, miten hyvin työlle asetettuihin tavoitteisiin päästiin.

Työn lopullisena tuloksena syntyy Frends-integraatioalustalle rakennettu prosessi, jonka avulla voidaan tukea ohjelmistotestauksen eri osa-alueita. Tämän lisäksi saadaan selville myös mahdollisia kehityskohteita Frends-integraatioalustalla liittyen ohjelmistotestaukseen.

2 Frends-integraatioalusta

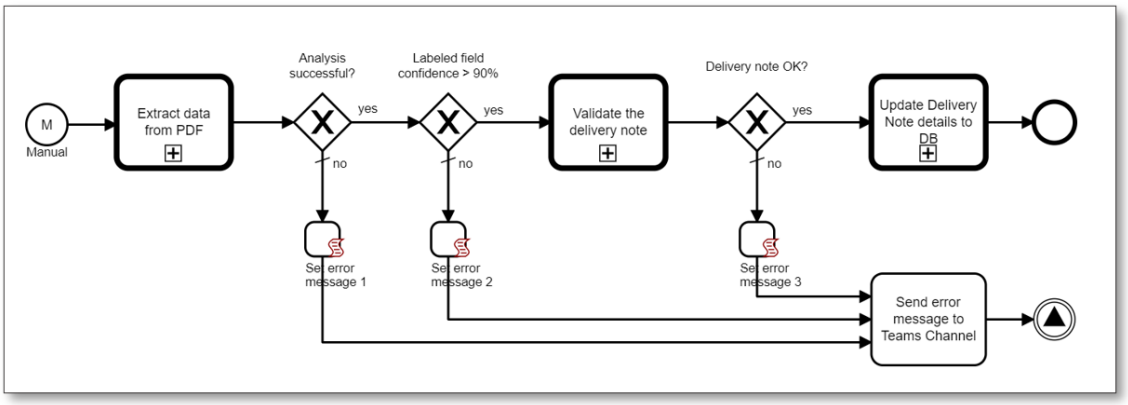
Tämän luvun tarkoituksena on esitellä Frends-integraatioalustaa ja sen ominaisuuksia, jotka ovat tämän insinöörityön kannalta oleellisia. Tällä pyritään luomaan lukijalle hyvä perustieto Frends-integraatioalustan ominaisuuksista, joita tässä insinöörityössä esiintyy. Näin ominaisuuksien ymmärtäminen helpottuisi.

2.1 Frends-integraatioalustan perusteet

Integraatiolla tarkoitetaan yksinkertaisimmillaan kahden tai useamman ohjelmiston tai järjestelmän yhdistämistä toisiinsa, jonka avulla tiedonsiirto näiden välillä toteutettaisiin (1).

Frends on siis integraatioalusta, jonka avulla voidaan yhdistää erilaisia kolmannen osapuolen järjestelmiä toisiinsa riippumatta siitä, sijaitsevatko nämä järjestelmät pilvessä vai paikallisissa datakeskuksissa (2).

Alusta hyödyntää low-code-lähestymistapaa, jolla tarkoitetaan, että alustalla koodin kirjoittamista vaaditaan hyvin vähän tai joissain tapauksissa ei lainkaan, mutta tämä riippuu kehitettävän integraation tarpeista. Frends hyödyntää siis BPMN (Business Process Model and Notation) -pohjaista visuaalista käyttöliittymää prosessien mallintamiseen ja prosesseilla tarkoitetaan integraatiovirtaa, joka kuvastaa kyseisen integraation toiminnallisuuden kokonaisuudessaan (3). Frends-alustalla rakennettu prosessi (kuva 1) ja prosessin luomiseen liittyviä ominaisuuksia käydään tarkemmin läpi aliluvussa 2.3.



Kuva 1. Esimerkki Frends-integraatioprosessista (3).

Kuvan 1 esimerkissä prosessin tarkoituksena on purkaa tiedot PDF-dokumentista, validoida tieto ja päivittää nämä tietokantaan. Kuvassa 1 esiintyvä prosessi sisältää Frends-aliprosesseja, ehtolauseita, expression-elementtejä, Frends-tehtävän, manuaalisen triggerin, return-elementin ja throw-elementin, joita käydään tarkemmin läpi aliluvuissa 2.2 ja 2.3.

Koodia tarvittaessa alustalla käytetään Microsoftin C#-ohjelmointikieltä, jonka avulla voidaan luoda tarpeiden mukaisesti monimutkaisempia toiminnallisuuksia integraatiolle.

Prosessien suorittamiset hoidetaan Frends Agenteilla ja ne voidaan asentaa seuraviille teknologia-alustoille (4):

- Frends-pilveen
- asiakkaan yksityiseen pilveen
- julkiseen pilveen
- asiakkaan paikallisille järjestelmille, kuten Windows ja Linux koneille
- säiliöihin.

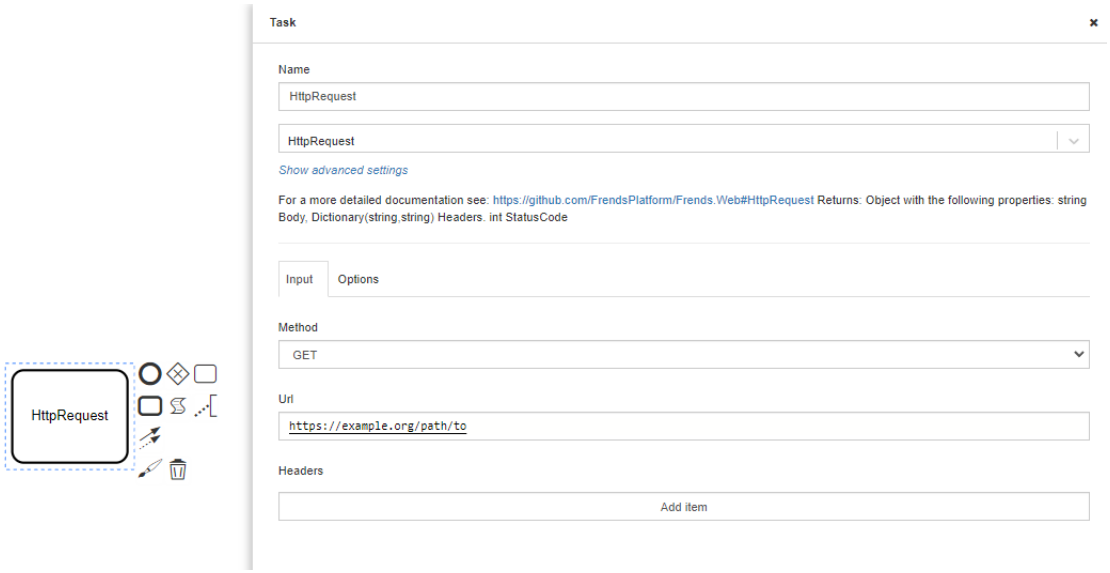
2.2 Frends Task

Frends Task eli Frends-tehtävä on yleensä yksinkertainen toimenpide, kuten HTTP-kutsun suorittaminen, tiedoston lukeminen tai vaikkapa SQL-kyselyn

toteuttaminen. Frends-tehtävän toiminnallisuus ja palautusarvo riippuvat sille annettavista parametreista. (5.)

Frends-tehtäviä voidaan luoda käyttäen C#-ohjelmointikieltä ja hyödyntäen erilaisia .NET-pinoja (.NET stack), mutta Frends-alustalle vientiä varten täytyy niistä tehdä NuGet-paketti. Useimpien Frends-tehtävien avoin lähdekoodi löytyy GitHub-versionhallintajärjestelmästä, joka mahdollistaa niiden toiminnallisuuden selvittämisen kooditasolla. (5.)

Kuvassa 2 on esimerkki HttRequest-tehtävästä ja osa sille annettavista parametreista, joihin sisältyy käytettävä HTTP-metodi, URL-osoite ja käytettävät otsakkeet. Kuvassa näkyvästä Options-välilehdeltä voidaan antaa myös muita oleellisia parametreja tarvittaessa.



Kuva 2. Frends-tehtävän HttpRequest-esimerkki.

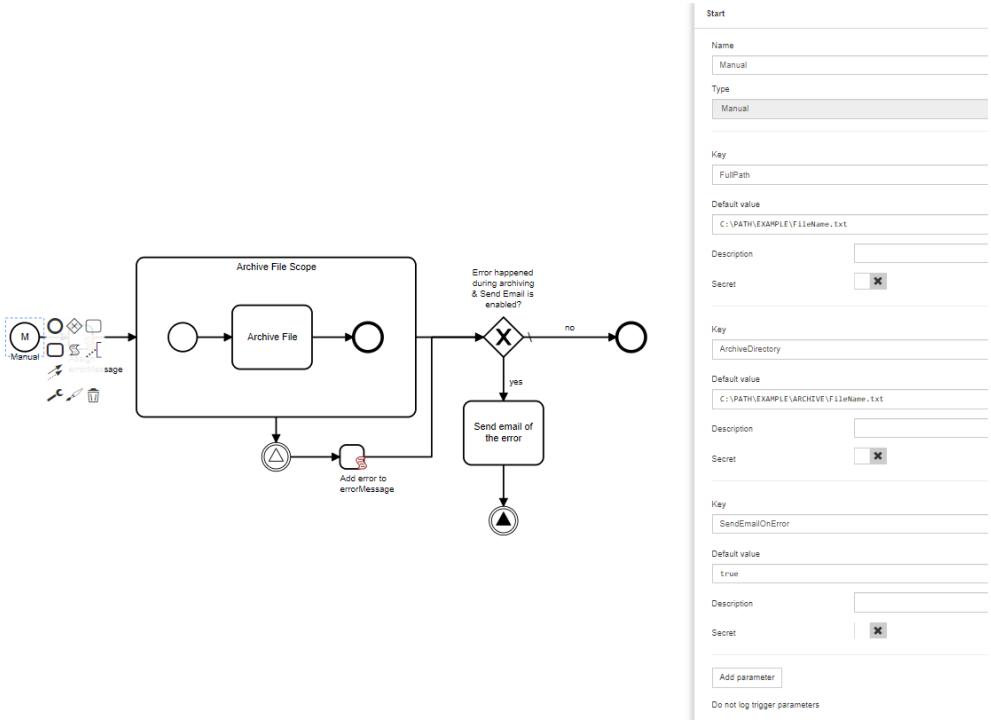
Useimmiten Frends-tehtävät sisältävät myös linkin lähdekoodiin, joka näkyy myös kuvassa 2. Tämä nopeuttaa huomattavasti sen toiminnallisuuksien selvittämistä.

2.3 Frends-prosessin ominaisuudet

Ennen kuin demonstroidaan, minkälaisia ominaisuuksia ja komponentteja Frends-prosessi sisältää, esitellään ensin erilaiset ratkaisuvaihtoehdot, joita alustalta löytyy.

Alustalla voidaan luoda prosessien lisäksi aliprosesseja, jotka ovat prosessien kanssa melkein identtisiä, mutta niiden käyttötarkoitukset eroavat toisistaan. Prosessi on yleensä jokin tietty kokonaisuus, joka on rakennettu uniikkia tarkoitusta varten. Täten sitä ei ole tarkoitus käyttää muissa prosesseissa eikä sitä myöskään voida käyttää muiden prosessien sisällä. Tätä varten on olemassa aliprosessit, joiden käyttötarkoitus usein on suorittaa yleinen toimenpide, joka olisi uudelleenkäytettävä. Näin säästytään turhalta työltä, kun voidaan luoda tarvittava toiminto vain kerran. Aliprosesseja voidaan toki luoda myös silloin, jos kehitettävän prosessin toiminnallisuus sisältäisi paljon eri osia, joka vaikeuttaa sen ymmärtämistä tai muuten prosessin BPMN-kaavio kasvaisi järkyttävän kokoiseksi. Tässä vaiheessa prosessia voitaisiin yksinkertaistaa luomalla osa toiminnallisuuksista aliprosessien avulla.

Hyvin yksinkertainen esimerkki aliprosessista löytyy kuvasta 3, jonka tarkoituksena olisi arkistoida parametrien mukainen tiedosto. Virheen syntyessä arkistoinnin aikana lähetettäisiin sähköposti, jos kuvassa 3 näkyvä parametrina annettu SendEmailOnError-muuttuja on arvoltaan tosi.



Kuva 3. Esimerkki kokonaisuudesta, joka voitaisiin toteuttaa aliprosessina.

Prosessista voitaisiin sanoa, että se vastaa kokonaista ohjelmistoa ja aliprosessi vastaisi täten funktiota.

Frends-alustalla on myös mahdollista rakentaa API-ratkaisuja, joiden toteutus aloitetaan ensin luomalla API:n kuvaus OpenAPI-spesifikaation avulla ja päätepisteistä luotaisiin Frends-prosesseja (6). Nämä prosessit ovat suurimmaksi osaksi samanlaisia kuin muutkin prosessit, mutta näiden aloituspiste on vakiona API-trigger, jota ei ole mahdollista muuttaa.

Seuraavaksi käydään läpi Frends-prosessiin liittyviä komponentteja, työkaluja ja muita ominaisuuksia, jotka ovat tämän insinöörityön kannalta oleellisia. Frends-prosessin pohja luodaan yksinkertaisesti painamalla prosessin luomiselle tarkoitettua painiketta alustalla, jonka jälkeen ilmestyy kuvassa 4 näkyvä ikkuna. Kuvan 4 vasemmassa reunassa on saatavilla olevat työkalut, joiden avulla prosessiin voidaan lisätä toiminnallisuutta ja selkeyttä. Oikeassa reunassa on prosessin yleisiä asetuksia, ja keskellä on itse prosessin kangas, jolle toiminnallisuudet lisätään. Prosessin asetuksista oleellisin asia tämän työn kannalta on Target

framework -kenttä, josta voidaan vaihtaa prosessin käyttämää .NET-pinoa. Vaihtoehdoista löytyy NET Framework 4.7.1, NET Standard 2.0 ja NET 6.0.

Kuva 4. Frends-prosessin editori.

Kuvassa 4 näkyvät työkalut ja niiden kuvaukset löytyvät kuvasta 5, josta ilmenee myös prosessille oleellisia viittauksia. Viittausten avulla voidaan käyttää prosessille annettavia parametreja, muuttujia ja Frends-tehtävien tuloksia.

Tyyppi	Nimi	Logo	Kuvaus
Työkalu	Connect tool		Käytetään elementtien yhdistämiseen.
Työkalu	Start Element		Merkitsee prosessin aloituspisteen, joka tunnetaan myös triggerinä. Triggereitä on manuaalisen triggerin lisäksi myös muitakin vaihtoehtoja ja prosessi voi sisältää useamman kuin yhden triggerin.
Työkalu	Return Element		Merkitsee prosessin lopetuspisteen. Prosessi voi sisältää useampia lopetuspisteitä, jotka palauttavat erilaisia arvoja.
Työkalu	Exclusive decision Element		Vastaa if-ehtolausetta, jolle annetaan parametriksi arvo, josta syntyy arvotyyppimuuttuja true tai false, jonka mukaan prosessin ajo ohjataan oikeaan haaraan.
Työkalu	Inclusive decision Element		Vastaa hieman switch-lausetta, mutta jokainen haara sisältää oman ehtonsa ja ehdon täyttyessä haara toteutetaan, joten monta eri haaraa voidaan toteuttaa samanaikaisesti. Kaikkien haarojen ehtojen ollessa false, siirtyy prosessin ajo kohtaan, jossa nämä haarat kohtaavat.
Työkalu	Throw Element		Käytetään virheenluontiin.
Työkalu	Task Element		Elementti, jolla voidaan valita Frends Task.
Työkalu	Expression Element		Käytetään silloin, kun tarve vain yhden rivin (one liner) koodille. Vaihtoehtoisesti voidaan myös tallettaa palautuva arvo muuttujaan.
Työkalu	Statement Element		Käytetään, kun tarve luoda C#-lauseke. Tällöin voidaan kirjoittaa puhdasta C#-koodia, kuten ohjelmointiympäristössäkin kirjoitettaisiin. Vaihtoehtoisesti voidaan myös tallettaa palautuva arvo muuttujaan.
Työkalu	Catch Element		Catch-lauseke, jolla voidaan käsitellä syntyviä virheitä. Voidaan lisätä Task elementtiin, Scope elementtiin ja aliprosessiin.
Työkalu	Scope Element		Käytetään elementtien ryhmittämiseen, jonka sisään voidaan lisätä monia elementtejä ja täten voidaan käsitellä minkä tahansa elementin virheet yhdellä Catch elementillä.
Työkalu	Foreach Element		Foreach-toistorakenne, jolle annettava taulukko (array) tai kokoelma (collection) käydään läpi ja näin voidaan suorittaa halutut operaatiot jokaiselle kohteelle.
Viittaus	#trigger	N/A	Viittaus, jolla saadaan triggeriltä tietoja, kuten parametrien arvoja. Esimerkki: Manuaalisella triggerillä on parametri "Content" ja tämän arvo saataisiin kirjoittamalla #trigger.data.Content
Viittaus	#var	N/A	Viittaus, jolla voidaan viitata tiettyyn muuttujaan. Esimerkki: Muuttujan nimi on Name, tällöin arvo saadaan kirjoittamalla #var.Name
Viittaus	#result	N/A	Viittaus, jota voidaan käyttää sellaisenaan tai lisäämällä sen perään hakasulkeiden sisään tietyn Task elementin nimi. #result: Viittaa viimeisimmän suoritetun elementin tulokseen #result[Name of Frends Task]: Viittaa Task elementin tulokseen, jonka nimi on "Name of Frends Task"

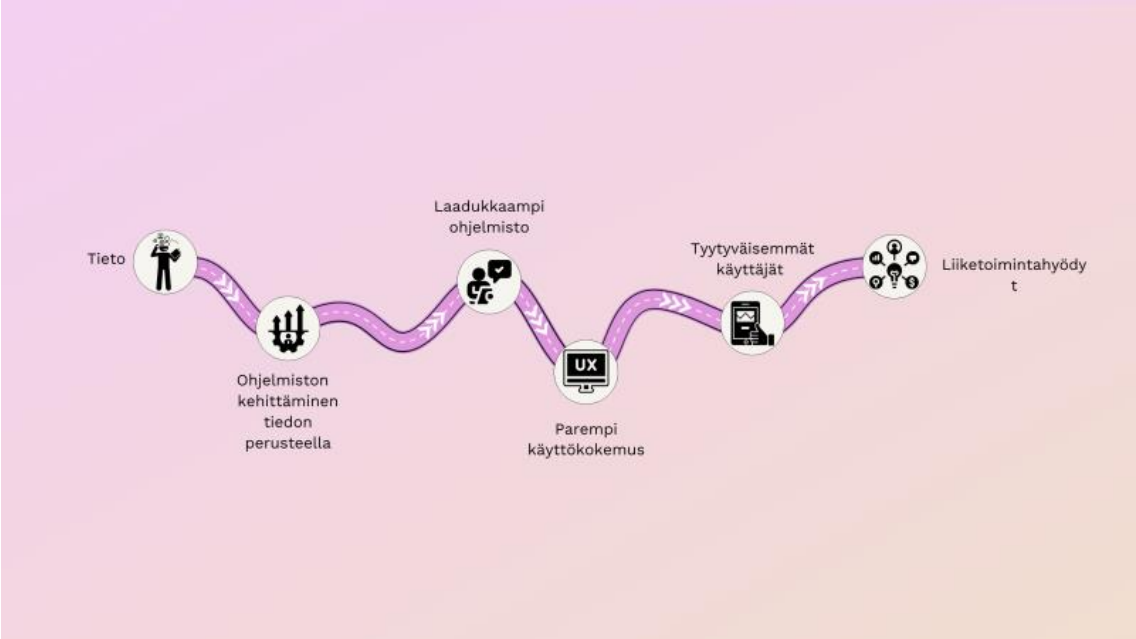
Kuva 5. Frends-prosessin oleellisimmat työkalut, viittaukset ja näiden kuvaukset.

Kuvan 5 elementtejä ja viittauksia tullaan käyttämään seuraavissa luvuissa, eikä näitä erikseen käydä läpi liiallisen toistuvuuden vuoksi, joten tähän kuvaan palaamalla saa lisätietoja toiminnallisuuksista tarpeen vaatiessa.

3 Ohjelmistotestaus Frends-integraatioalustalla

3.1 Ohjelmistotestaus yleisellä tasolla

Ohjelmistotestauksella pyritään varmistamaan ohjelmiston laatu ja siihen liittyviä ominaisuuksia testaamalla kehitettävää ohjelmistoa niin, että siinä pyritään löytämään virheitä, jotta nämä voidaan korjata (7). Yksi tärkeimmistä ohjelmistotestauksen tavoitteista on vaatimusten täyttäminen, jonka avulla voitaisiin todeta, että kehitettävä ohjelmisto toimii odotetulla tavalla (8). Edeltävien selvitysten mukaisesti ohjelmistotestauksen tulisi tuottaa tietoa ohjelmistosta ja sen toiminnallisuudesta, jonka avulla ohjelmistoa voitaisiin arvioida ja tarvittaessa parantaa. Täytyy kuitenkin pitää mielessä, että ohjelmistotestauksella voi olla erilaisia tavoitteita ja toteutustapoja riippuen kehitettävästä ohjelmistosta tai tuotteesta. Kuvasta 6 käy hyvin ilmi, miten ohjelmistotestauksen tuottaman tiedon perusteella voidaan saavuttaa useita hyötyjä, kuten laadun parantaminen, käyttökokemuksen kehitys, asiakastyytyväisyyden kasvu ja liiketoiminnalliset hyödyt.



Kuva 6. Ohjelmistotestauksesta saadun tiedon hyödyt (8).

Ohjelmistotestaus voidaan toteuttaa hyvin erilaisia testaustapoja ja näkökulmia hyödyntäen, mutta Frends-integraatioalustan tutkimiseen päätettiin valita vain tiettyjä ohjelmistotestauksen liittyviä testaustapoja ja -tasoja sekä selvittää, miten nämä esiintyvät alustalla ja voitaisiinko näitä kehittää.

3.2 Testaustavat ja osa-alueet

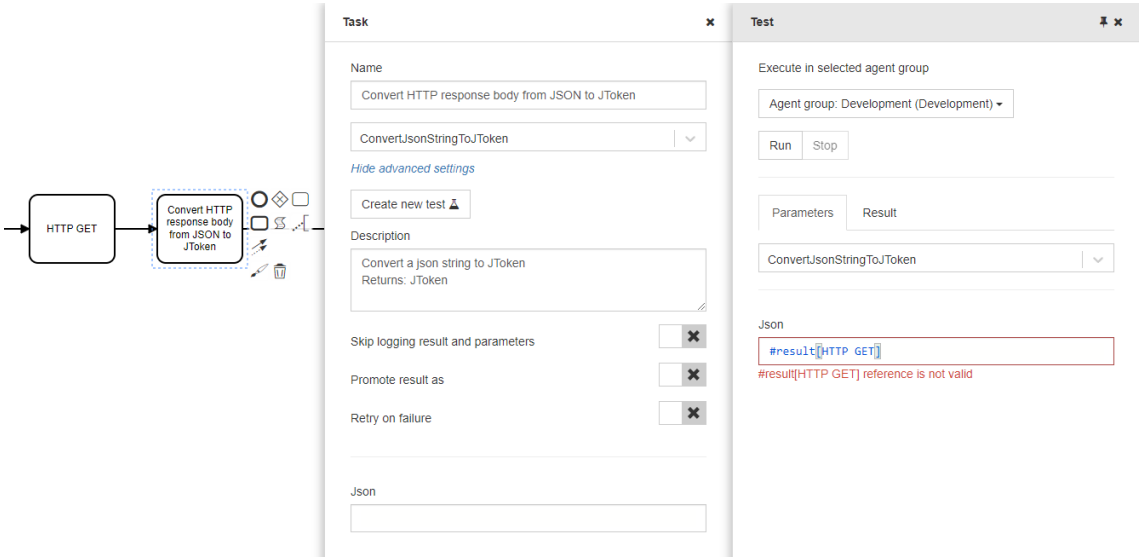
Ohjelmistoa voidaan testata monia eri testaustapoja ja osa-alueita hyödyntäen, mutta tätä työtä varten valittiin kaikista oleellisimmat tavat ja osa-alueet, jotka löytyvät Frends-alustalta jossakin muodossa. Tässä vaiheessa on hyvä huomioida, että API-ratkaisuihin liittyviä testauksia ei käydä tarkemmin läpi, sillä ne toimivat suurimmaksi osaksi käyttäen prosesseja. API-ratkaisuja voidaan kuitenkin testata hyödyntäen Swagger-käyttöliittymää, joka on sisäänrakennettu Frends-alustalle tai joissain tapauksissa esimerkiksi Postman-sovelluksen avulla.

Seuraavaksi käsitellään näitä testaustapoja ja osa-alueita:

- yksikkötestaus
- integraatiotestaus
- järjestelmätestaus
- hyväksymistestaus.

Yksikkötestauksen tarkoituksena on varmistaa yksittäisten komponenttien toiminnallisuus (8). Frends-alustalla yksikkötestien suorittaminen on sisäänrakennettu ominaisuus, mutta se on saatavilla vain Frends-tehtäville. Tämä toiminnallisuus mahdollistaa yksikkötestauksen toteuttamisen, mutta kuvasta 7 nähdään esimerkki, jossa yksikkötestauksen ikkunassa yritetään antaa edellisen Frends-tehtävän tulos viittaamalla siltä palautuvaan tulokseen, mutta tätä ei ole mahdollista käyttää. Prosessin muuttujien arvojakaan ei voida käyttää yksikkötestien suorittamisessa, joten testaamiseen tarvitaan mahdollisimman tarkka tieto siitä, minkälaista dataa Frends-tehtävän odotetaan saavan ja mahdollisia variaatioita, jotta yksikkötestauksella voitaisiin todeta sen toimivuus mahdollisimman hyvin. Tässä kyseisessä tilanteessa voitaisiin toki ensin suorittaa yksikkötesti HTTP

GET -nimiselle Frends-tehtävälle, jonka jälkeen tulos kopioitaisiin seuraavalle Frends-tehtävälle parametriksi ja näin saataisiin suoritettua yksikkötestit onnistuneesti.

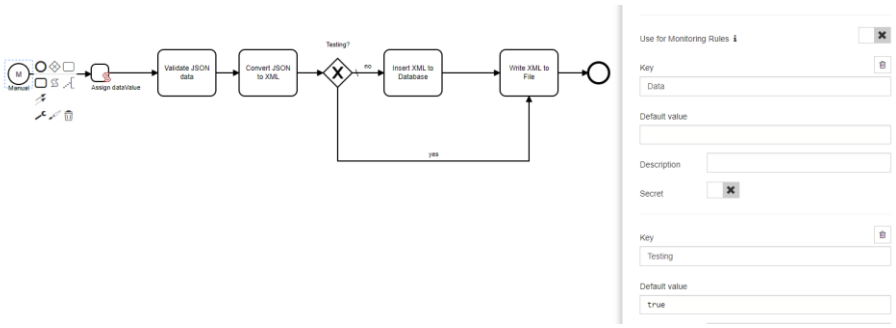


Kuva 7. Frends-tehtävän yksikkötestausikkuna.

Kuvan 7 esimerkissä nähdään vain kaksi Frends-tehtävää, ja näiden yksikkötestaus olisi melko nopeata ja vaivatonta, mutta prosessin kasvaessa yksikkötestauksien suorittaminen hidastuu. Usein Frends-tehtävien välillä on myös elementtejä, joiden yksikkötestaus ei ole mahdollista ja testaaminen vaikeutuu, kun elementtien sisällöstä ei ole varmaa tietoa. Yksikkötestaus Frends-alustalla on tällä hetkellä hyvin toteutettu ja sen kehittämiseen on vaikea keksiä parannuksia, mutta yksikkötesti-toiminnallisuuden lisäys statement- ja expression-elementeille saattaisi olla hyödyllistä. Expression-elementin avulla voidaan luoda funktioita, joiden testaaminen erilaisilla parametreilla olisi hyvin kätevä toiminto varsinkin, jos funktion logiikka monimutkaistuu. Tällä hetkellä funktioiden toimintoa voi testata lähinnä ajamalla prosessi erilaisilla parametreilla tai testata funktion toiminnallisuus erillisellä ohjelmointiympäristöllä. Statement-elementti sisältää C#-koodia, ja tämänkin testaaminen olisi hyödyllistä, mutta hyvin usein se sisältää muiden elementtien tuloksia, joten yksikkötestitoiminnallisuus ei olisi välttämättä tarpeellinen.

Integraatiotestaus on yleensä seuraava vaihe yksikkötestauksen jälkeen ja sen tarkoituksena on testata komponenttien välistä toiminnallisuutta, jonka avulla voidaan tuoda esille mahdollisia ongelmia komponenttien välisessä tiedonsiirrossa. Integraatiotestaukselle oleellisia malleja ovat Big Bang -lähestymistapa ja inkrementaalinen lähestymistapa, joista Big Bang -lähestymistavan tarkoitus on testata ohjelmistoa, kun taas inkrementaalisen integraatiotestauksen tarkoituksena on jakaa ohjelmiston komponentit osiin ja näin testata ohjelmistoa vaiheittain. (9.) Frends-alustalla käytetään kumpaakin tapaa, mikä riippuu tietenkin kehitettävän prosessin tilanteesta. Esimerkiksi aiemmin esiteltyä prosessia (kuva 7) ei olisi kovin tarpeellista testata inkrementaalisesti, kun siihen liittyvät kaikki komponentit ovat jo valmiita. Sen sijaan voitaisiin testata Big Bang -lähestymistavalla syötellen prosessille erilaisia parametreja ja näin saataisiin selville, miten elementit käsittelevät toistensa tuottamia tietoja.

Kuvassa 8 on esimerkki prosessista, jota olisikin jo hyödyllistä testata inkrementaalisesti johtuen siitä, että se sisältää tietokantaoperaation ja tietokantaa ei ole vielä tässä vaiheessa olemassa. Tarkoituksena olisi selvittää, miten prosessissa validointi, JSON-datan muunnos XML-formaattiin ja tiedoston luonti onnistuvat erilaisilla arvoilla, joita Data-parametrille syötetään. Kuvassa 8 näkyvä ehtolause ja manuaalinen Testing-parametri on lisätty väliaikaisesti prosessiin, jotta voidaan testata prosessia ilman tietokanta operaatiota. Kun testaus ja korjaukset on suoritettu ja tietokanta luotu, voimme poistaa väliaikaisen Testing-parametrin ja siihen liittyvän ehtolauseen. Tämän jälkeen voimme aloittaa testauksen Big Bang -lähestymistavalla ja varmistaa, että kaikki jäljellä olevat toiminnallisuudet toimivat oikein.



Kuva 8. Esimerkki Frends-prosessista, jonka muut elementit ovat valmiita paitsi tietokantaoperaatiota varten tarvittava tietokanta.

Integraatiotestaus Frends-alustalla on mielestäni hyvällä tasolla ja sen suorittaminen joissain tapauksissa vaati ylimääräisten elementtien lisäyksiä prosessiin, mutta tämä on usein nopea toimenpide. Tämän kehittämiseen on todella vaikea keksiä parempaa ratkaisua, mutta yksi mahdollinen ratkaisu olisi, että voitaisiin valita tietyt elementit prosessista, joita halutaan testata. Näin voitaisiin kuvan 8 esimerkissä jättää tietokantaoperaatio pois kokonaan välistä ja hypätä sen yli seuraavan elementin suoritukseen. Tämä kehitysratkaisu tuottaisi kuitenkin ongelmia siinä vaiheessa, jos kuvan 8 esimerkissä olevan tietokantaoperaation tulosta käytettäisiin tiedostoon siirrettävän datan luomiseen. Tällöin tietokantaoperaation tuloksella olisi merkitystä eikä ratkaisu välttämättä olisi tarpeellinen tai hyödyllinen.

Järjestelmätestaus muistuttaa hieman integraatiotestauksen Big Bang -lähestymistapaa, mutta järjestelmätestauksen tarkoituksena on selvittää, että kaikki järjestelmän osat toimivat oletetusti ja täyttäen sille vaaditut vaatimukset. Järjestelmätestaus suoritetaan usein, kun yksikkötestit ja integraatiotestaus on suoritettu ja ennen hyväksymistestaukseen siirtymistä. Järjestelmätestaus myös suoritetaan yleisesti ympäristössä, joka vastaa hyvin paljon lopullista ympäristöä, jossa ohjelmiston on tarkoitus toimia sen valmistuttua. (10.)

Järjestelmätestaus usein suoritetaankin Frends-alustalla toisessa ympäristössä kuin edelliset testaukset, jotka suoritetaan yleisimmin kehitykselle tarkoitetussa ympäristössä. Järjestelmätestaus suunnitellaan joissakin tapauksissa asiakkaan

toimesta, mutta useimmiten tämän suunnittelee ja toteuttaa Frends-kehittäjä tai kehitystiimi. Tässä tapauksessa on hyvin kriittistä, että asiakkaalta saadut vaatimukset kehitettävälle prosessille ovat tarpeeksi kattavia, sillä testitapaukset suunnitellaan usein näiden pohjalta. Järjestelmätestaukseen useimmiten hyödynnetään kehitettävän prosessin vaatimusmäärittelyiden tietoja, joiden avulla järjestelmätestaus suoritetaan lähtökohtaisesti. Järjestelmätestausta voitaisiin mahdollisesti kehittää luomalla dokumentaatiopohja, jonka avulla voitaisiin järjestelmätestaus suunnitella ja toteuttaa siten, että samalla jäisi jälki kehitettävän prosessin järjestelmätestaus vaiheesta. Tämä kehitysidea toisi kuitenkin lisätyötä, joka ei olisi täysin välttämätöntä ottaen huomioon, että nykyinen toimintatapa toimii hyvin ja lisätyön kasvaessa syntyisi myös lisäkustannuksia asiakkaalle. Tällöin kyseisen ratkaisun käyttöönotto olisi täysin asiakkaan päätettävissä.

Hyväksymistestaus on asiakkaan toteuttama testausvaihe, jonka tarkoituksena on todentaa loppukäyttäjän näkökulmasta, että kehitetty ohjelmisto tai tuote toimii ohjelmistolle asetettujen vaatimusten ja liiketoiminnallisten vaatimusten mukaisesti. Hyväksymistestaus muistuttaa hieman järjestelmätestausta, mutta hyväksymistestauksen suorittaa useimmiten asiakas. (11, s. 23.)

Frends-alustalla hyväksymistestaus muistuttaa hyvin paljon alustalla suoritettavaa järjestelmätestausta, mutta hyväksymistestaukseen käytetään usein ympäristöä, joka vastaisi vielä paremmin ympäristöä, jossa lopullisen ratkaisun on tarkoitus toimia. Hyväksymistestauksen suunnitteluissa ja toteutuksissa on hyvinkin paljon eroavaisuuksia riippuen asiakkaasta ja heidän tarpeistansa sekä budjetista. Joidenkin asiakkaiden kohdalla hyväksymistestaus suoritetaankin järjestelmätestauksen yhteydessä, jonka avulla säästetään hieman kustannuksissa. Frends-alustalla hyväksymistestauksen suunnittelu on useimmiten asiakkaan vastuulla ja toteutus Frends-kehittäjien vastuulla, mutta tässä on kuitenkin poikkeuksia riippuen asiakkaasta ja kehitetystä prosessista. Hyväksymistestauksen kehittämiseen on hyvin vaikea ottaa kantaa, mikä johtuu siitä, että vastuu on suurimmaksi osaksi asiakkaalla. Toki tätä varten voitaisiin luoda

dokumentaatiopohja, jolla hyväksymistestaus voitaisiin suunnitella, jos asiakkaalla ei ole olemassa hyväksymistestausta varten jo olemassa olevaa suunnitelmaa.

4 Testitapauspohjaisen alustusprosessin kehitys

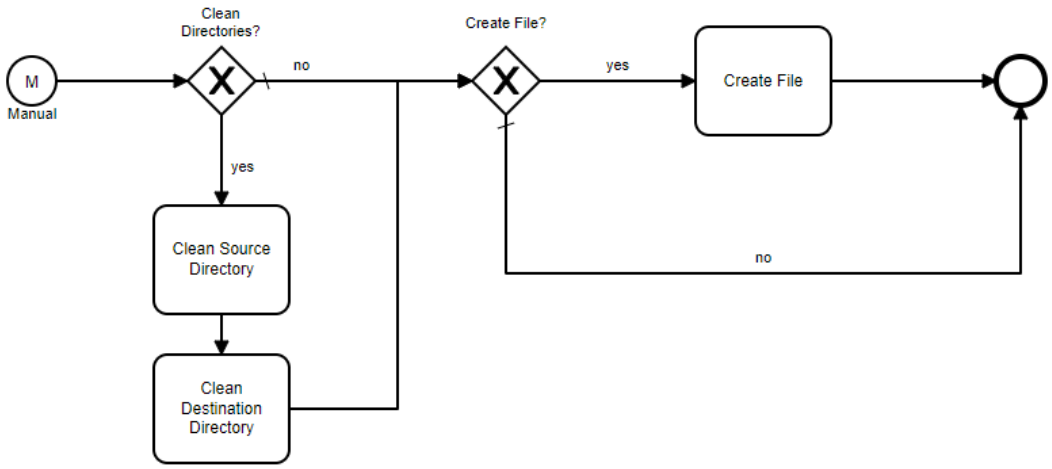
Testitapauspohjaisen alustusprosessin kehitys toteutettiin lähtökohtaisesti vesiputousmallia hyödyntäen, mutta kehityksen aikana ilmeni kuitenkin tilanteita, joissa alun perin luodut määrittelyt tai suunnitelmat hieman muuttuivat, joten perinteisestä vesiputousmallista jouduttiin hieman poikkeamaan. Kehitysvaiheista on myös karsittu tiettyjä osia, jotka eivät olleet täysin oleellisia tämän projektin kannalta tai muuten tuoneet lisäarvoa projektille. Vesiputousmallin valintaan päädyttiin siksi, että kehitettävä Frends-prosessi tulisi olemaan melko pienehkö ja sen kehityksen vaiheet tulisivat oletetusti olemaan melko yksinkertaisia.

4.1 Määrittely

Kehitystyö aloitettiin määrittelyllä, jossa pyrittiin mahdollisimman tarkasti tuomaan esille, mitä prosessin kuuluu tehdä, eli määritettiin vaatimukset. Tässä kohtaa ei keskitytty vielä teknisiin ominaisuuksia tai toteutukseen vaan lähinnä prosessin toiminnallisuuteen ylemmällä tasolla ja siihen, mitä kehitettävällä prosessilla halutaan saavuttaa. Määrittely pyrittiin laatimaan siten, että sen toteuttaja olisi liiketoiminnan henkilö, jolla ei välttämättä olisi teknistä osaamista, mikä saattaa olla hyvin realistinenkin tapaus.

Tässä vaiheessa on hyvä tuoda esille, mitä tiedostopohjaisilla Frends-integraatioilla tarkoitetaan. Ne ovat lähtökohtaisesti prosesseja, jotka käynnistyvät yhden tai useamman tiedoston ilmestymisestä määritettyyn hakemistoon. Jotta tämän kaltaisia prosesseja voidaan testata, joudutaan luomaan vähintään yksi tiedosto määrittelyiden mukaiseen hakemistoon, mutta useimmiten on tarve muillekin operaatioille. Testitapausten edetessä on hyvin todennäköistä, että tarvitaan operaatioita kuten hakemistojen tyhjentäminen, jos testitapaus on epäonnistunut ja tiedosto on jäänyt jumiin, mikä estää testauksen etenemisen.

Ennen tämän prosessin kehityksen aloitusta tiedostopohjaisten Frends-prosessien testit alustettiin lähinnä ylimääräisillä prosesseilla, joiden tarkoitus oli suorittaa erilaisia tiedostojen käsittelyyn liittyviä operaatioita (kuva 9). Nämä prosessit yleensä sisältävät hakemistojen tyhjennyksen ja tiedoston luonnin, mutta tarvittaessa prosessit voivat sisältää muitakin operaatioita.



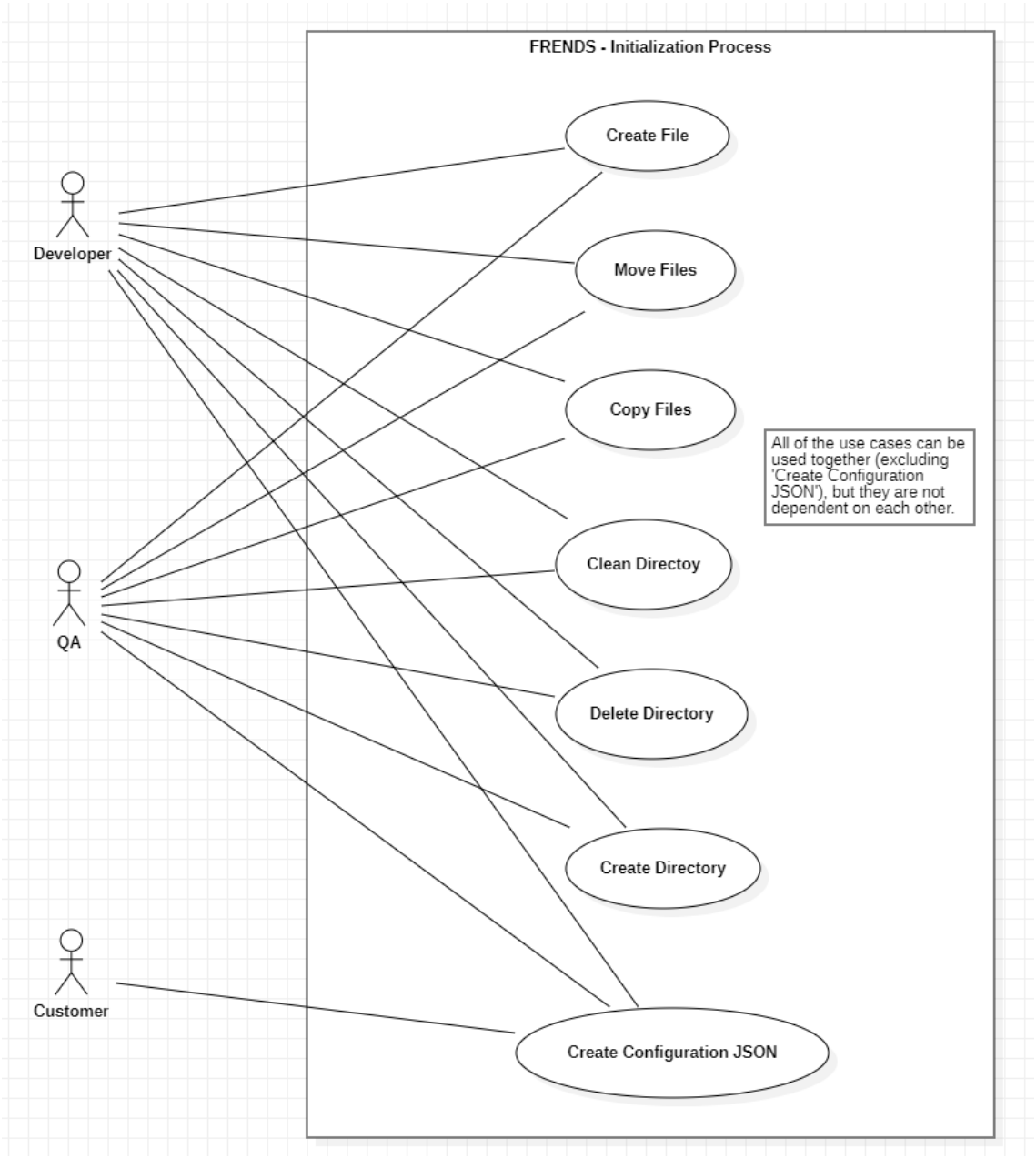
Kuva 9. BPMN-kaava testitapauksien alustuksia hoitavasta Frends-prosessista. Yksinkertainen esimerkki, jota on käytetty ennen tämän työn aloitusta.

Ongelma, joka kuvan 9 prosesseissa ilmenee, on epäsäännöllisyys ja uudelleenkäytettävyyden puute, sillä nämä ovat hyvin usein prosessikohtaisia. Näin niitä ei voida hyödyntää muiden prosessien alustuksia varten. Prosessin (kuva 9) kaikille operaatioille ei aina välttämättä ole tarvetta tai vaadittaisiin enemmän operaatioita, joten nämä ylimääräiset prosessit eroavat toisistaan hyvin usein. Tässä vaiheessa on vielä hyvä huomioida, että nämä ylimääräiset prosessit eivät ole yhden kehittäjän kehittämiä, vaan ne ovat usein sen kehittäjän luomia, joka on kyseessä olevaa Frends-prosessia kehittänyt. Tämä on yksi syy, minkä takia ne eroavat usein toisistaan. Näiden huomioiden ohella syntyi tarve tälle kehitettävälle alustusprosessille, jonka tavoitteena olisi poistaa epäsäännöllisyydet, kun alustuksia varten olisi yleiskäyttöinen versio olemassa ja samalla prosessi olisi uudelleenkäytettävä.

Kehitettävän prosessin tulisi siis toimia tiedostopohjaisten Frends-integraatioiden alustusprosessina, ja sen pääsääntöinen käyttötarkoitus on avustaa Frends-prosessien testauksissa. Oleellinen osa testien suorittamista on myös niiden dokumentointi, ja tätä varten aiemmin mainittujen ylimääräisten alustusprosessien kanssa jouduttiin jokaisen testauksen jälkeen kirjaamaan ylös käytetyt parametrit ja näiden avulla saadut tulokset. Näistä saatiin selville, mitä oli testattu ja miten, mutta testien dokumentointia ei välttämättä aina vaadittu, joten testauksista ei tällöin jäänyt pysyvää jälkeä. Tämän syyn takia kehitettävä alustusprosessi päätettiin rakentaa testitapauspohjaiseksi ja tällä tarkoitetaan sitä, että prosessi voitaisiin konfiguroida testitapausten pohjalta. Testitapauspohjaisuuden tavoitteena oli tuoda mahdollisimman yksinkertaisella tavalla esille, miten kyseessä olevaa Frends-prosessia tullaan testaamaan tai miten sitä on testattu, eli se toimisi ikään kuin dokumentaationa itsessään.

Prosessin vaatimukset luotiin käyttäjätarinoiden muodossa (liite 1) ja tässä vaiheessa pyrittiin kirjaamaan ylös kaikki mahdolliset toiminnallisuudet prosessille menemättä liian tekniseksi. Tällä tavalla saatiin tuotteen toiminnallisuudet esitettyä mahdollisimman selkeästi loppukäyttäjän näkökulmasta. Käyttäjätarinoissa ei ole mainittu erikseen, että kyseessä voi olla SMB (Server Message Block), FTP (File Transfer Protocol) -palvelin tai SFTP (Secure File Transfer Protocol) -palvelin, mutta jokainen käyttäjätarina, joka liittyy tiedostoihin tai hakemistoihin, pitää sisällään mahdollisuuden kaikkiin näihin kolmeen.

Määrittelyvaiheessa luotiin selvennyksen vuoksi käyttötapauskaavio (kuva 10), jonka avulla helpotettaisiin suunnitteluvaihetta visuaalisella ohjeistuksella. Tämän avulla kehitettävästä prosessista saatiin mahdollisimman tarkasti selville sen toiminnallisuus visuaalisessa formaatissa loppukäyttäjän näkökulmasta (12). Käyttötapauskaavio sisältää muutamia toimintoja, joita ei ole selitetty käyttäjätarinoissa. Syynä tähän on, että päätettiin lisätä nämä toiminnot prosessiin lisätoiminnallisuuksina, jotka voivat tulla tarpeellisiksi tietyissä tilanteissa.



Kuva 10. Alustusprosessin havainnollistava käyttötapauskaavio.

Kaavion (kuva 10) avulla havaitaan, että kaikki toiminnot voi suorittaa joko kehittäjä tai laadunvarmistaja. Lisäksi asiakkaalla on mahdollisuus luoda konfigurointi-JSON, mikä antaa heille mahdollisuuden vaikuttaa prosessin testaukseen. Lähtökohtaisesti konfiguroinnin hoitaisi kehittäjä asiakkaalta saatujen tietojen mukaisesti.

4.2 Suunnittelu

Suunnittelu aloitettiin tarkastelemalla prosessin suunniteltua toiminnallisuutta, joka perustui määrittelyjen analyysiin ja käyttötapauskaavion (kuva 10) tarkasteluun. Tämän tarkastelun perusteella havaittiin, että prosessin tarkoituksena on suorittaa tiettyjä tiedostoihin liittyviä toimenpiteitä, jotka voivat myös tapahtua samanaikaisesti. Prosessin on oltava testitapauspohjainen. Tämän vuoksi prosessin konfigurointi-JSON tulisi määritellä niin, että se jäljittelee testitapauksia, jotka toimivat samalla ohjeistuksena prosessille. Prosessin pääasiallinen käyttötarkoitus olisi toimia testauksen tukena, mutta sen toiminnallisuuksia voitaisiin hyödyntää myös muihin tarkoituksiin.

Seuraavana laadittiin prosessin mahdollisia käyttötapauksia juuri tehdyn analyysin mukaisesti. Näiden käyttötapausten avulla saatiin tuotua esille, että prosessia voitaisiin käyttää muihinkin tarkoituksiin kuin testitapausten alustuksia varten.

Kehitettävällä alustusprosessilla voidaan

- luoda tiedosto hakemistoon
- luoda tiedosto moneen eri hakemistoon
- luoda monia tiedostoja yhteen tai useampaan hakemistoon
- siirtää tiedostoja hakemistojen välillä
- kopioida tiedostoja hakemistosta toiseen
- tyhjentää hakemisto tai useampi kerrallaan
- poistaa hakemisto tai useampi kerrallaan
- luoda hakemisto tai useampi kerrallaan.

Edellä mainitusta luetelmasta havaitaan, että prosessin avulla voitaisiin suorittaa kyseisiä toimintoja erikseen, mutta näitä voitaisiin suorittaa myös samanaikaisesti. Käyttötapausten samanaikainen toteutus vähentäisi samalla tarvetta ylimääräiselle työlle, kun prosessin ohjeistukset voidaan antaa yhden parametrin avulla.

Prosessin toiminnallisuuden ja käyttötapausten selvityksen jälkeen aloitettiin selvittämään, millä tavalla alustusprosessi kannattaisi toteuttaa Frends-integraatioalustalla. Tämä kyseinen prosessi Frends-integraatioalustalla voitaisiin toteuttaa rakentamalla siitä API eli rajapinta, Frends-prosessi, Frends-aliprosessi tai se voitaisiin rakentaa Frends-tehtävänä. Näitä ratkaisuja vertailtiin keskenään niiden haittojen ja hyötyjen näkökulmasta (taulukko 1), jotta saataisiin selville sopivin ratkaisu.

Taulukko 1. Mahdolliset toteutustavat alustusprosessille Frends-integraatioalustalla.

Toteutustapa	**Haitat**	**Hyödyt**
API	Rajapinnan siirto toiseen ympäristöön, autentikoinnit ja niiden ylläpito, mahdolliset palomuurien avaukset ja versionhallinta.	Yhteen ympäristöön rakennettuna helpottuisi mainitut haitat, mutta autentikoinnit ja palomuurit tuovat ylimääräistä työtä.
Frends-prosessi	Versionhallinta. Parametrien syöttämiseen tarkoitettu ikkuna on hieman pieni.	Ei autentikointia, helppo jakaa eteenpäin ja kustomointi mahdollista.
Frends-aliprosessi	Toteutus ei olisi aliprosessien standardien mukainen, sillä sitä ei ole tarkoitus käyttää prosessien sisällä vaan itsenäisenä kokonaisuutena.	Samat hyödyt kuin Frends-prosessilla.
Frends Task	Frends-tehtävät ovat yksittäisiä toimenpiteitä, joiden tarkoituksena on suorittaa usein hyvin yksinkertainen toimenpide, kuten poistaa hakemisto. Rakennettava prosessi sisältää monta eri toiminnallisuutta, joten se ei olisi aivan standardien mukainen.	Versionhallinta.

Toteutustapojen analysoinnin jälkeen päädyttiin toteuttamaan ratkaisu Frends-prosessilla, sillä se ei riko standardeja, kuten Frends-tehtävä ja Frends-aliprosessivaihtoehdot. Frends-prosessi ei myöskään vaatisi ylimääräisiä toimenpiteitä, jotka saattaisivat joissakin tilanteissa estää koko prosessin käytön, kuten API-ratkaisussa esiintyvät autentikointi- ja palomuuriongelmat.

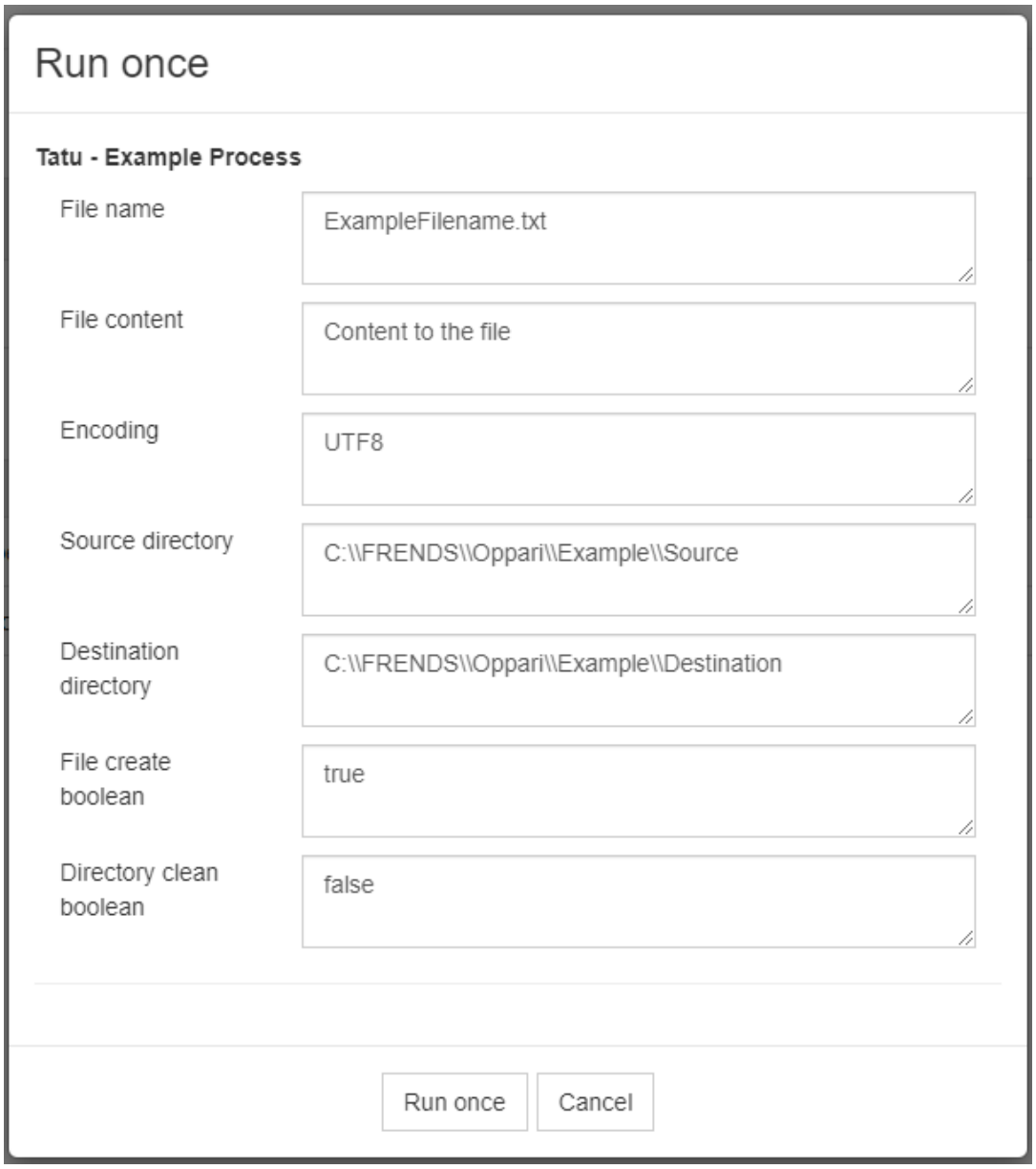
Toteutustavan päätöksen jälkeen keskityttiin tietoturvaan liittyviin osiin, jotka saattaisivat tuottaa ongelmia, ja pohdittiin, miten ne voitaisiin ratkaista. Frends-prosessille täytyisi antaa FTP- ja SFTP-palvelimia sekä SMB:tä varten käyttäjätunnukset ja nämä täytyisi pitää salassa. Joissakin tapauksissa ei ole tarvetta kuitenkaan käyttää tunnuksia, joten tämäkin tulisi huomioida toteutuksessa. Tunnukset tulisi antaa ympäristömuuttujina, mikä on muutenkin käytäntönä arkaluontoisille muuttujille. Frends-prosessissa on myös mahdollisuus salata muuttujien arvoja ja ulostuloja, joten tämän toiminnallisuuden avulla voidaan salata arkaluontoiset arvot.

Tähän mennessä oli saatu tiedot prosessin toiminnallisuuksista, käyttötapauksista, tietoturvasta ja toteutustavasta, joten seuraavaksi lähdettiin selvittämään minkälainen parametrin tai parametrien täytyisi olla.

Parametrin tulisi täyttää seuraavat vaatimukset:

- Rakenteen täytyisi olla mahdollisimman selkeä ja helppolukuinen.
- Rakenteen tulisi jäljitellä testitapauksia.
- Rakennetta voitaisiin hyödyntää testitapausten dokumentointiin.
- Rakenteen tulisi sisältää prosessin toiminnallisuuden kannalta oleellisimmat parametrit.

Käytetään esimerkkinä normaalia parametrisointia, jota on käytetty entisessä tavassa tämän kaltaisella alustusprosessilla (kuva 11).



Kuva 11. Ponnahdusikkuna Frends-prosessin parametrien laatimista varten.

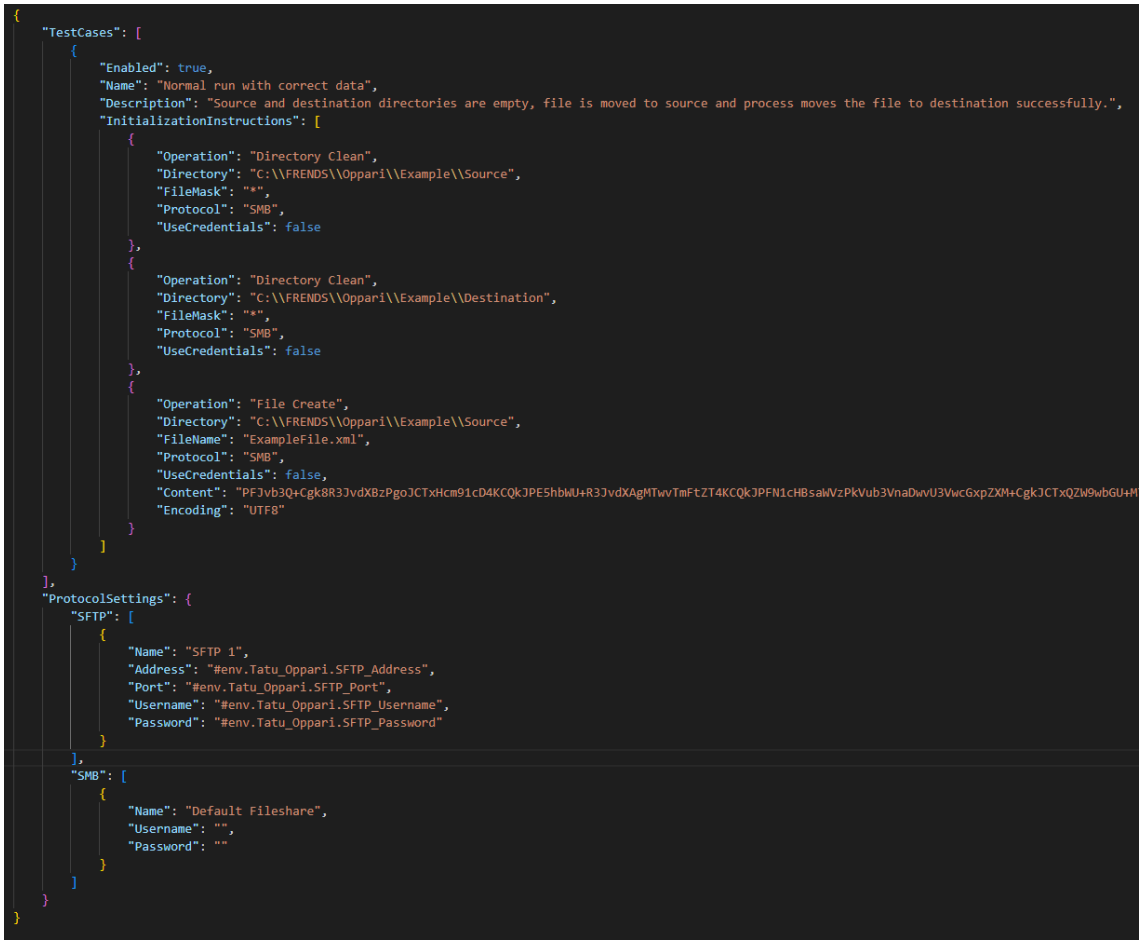
Parametrit (kuva 11) kyseisen prosessin kohdalla eivät tuota suuria ongelmia, sillä sen tarkoituksena on luoda vain yksi tiedosto ja tyhjentää hakemistot käyttäjän määrittelemillä parametreilla. Tämän tapainen parametrisointi tuottaisi kuitenkin kehitettävän alustusprosessin kanssa muutaman ongelman, kuten parametrien lisääntymisen, ja sen, että niiden luominen tai kopiointi olisi työlästä. Tällöin ei päästäisi myöskään eroon vanhasta tavasta dokumentoida parametreja. Näin ollen tämän esimerkin avulla toivottu ratkaisu sisältäisi mahdollisimman vähän parametreja, joka helpottaisi myös dokumentointia.

Käytettäväksi parametriksi valittiin lopulta JSON-rakenne, joka sisältäisi tarvittavat parametrit prosessin suoriutumiseen. Täten saadaan vähennettyä Frends-prosessin parametrien määrä yhteen ja JSON-rakenteen tulisi olla mahdollisimman selkeä, testitapauksiin pohjautuva ja toimia myös testitapausten dokumenttina. Ennen JSON-rakenteen luomista selvitettiin, mitä Frends-tehtäviä prosessissa tultaisiin käyttämään, jotta saataisiin selville vaaditut parametrit. Tätä varten käytettävät Frends-tehtävät kerättiin yhteen ja selvitettiin niiden oleellisimmat parametrit (liite 2). Kyseiset Frends-tehtävät sisältävät muitakin parametreja, mutta tätä prosessia varten ne jätettiin pois, joko sen takia, että niitä käytetään todella harvoin, tai sen vuoksi, että ne eivät muuten ole oleellisia kehitettävän prosessin kannalta.

Parametrien selvittämisen yhteydessä huomattiin, että Frends-prosessin .NET-kehikkoa vaihtamalla .NET 6.0 -versioon saataisiin käyttöön todella hyödyllisiä Frends-tehtäviä, jotka olisivat yksinkertaistaneet prosessin toimivuutta. Kehikon vaihtaminen päätettiin kuitenkin toistaiseksi jättää tekemättä, sillä uusin Frends-integraatioalustan versio oli vastikään päivitetty hyödyntämään .NET 6.0 -versiota ja käytössä on edelleen vanhempia ympäristöjä, jotka käyttävät joko .NET Standard 2.0- tai .NET Framework 4.7.1 -versiota.

Tarvittavien parametrien selvittelyn jälkeen aloitettiin luomaan prototyyppejä tulevasta konfigurointi-JSON-rakenteesta, joka toimisi prosessin yhtenä ainoana parametrinä. Muutaman prototyypin jälkeen saatiin tuotettua kriteerit täyttävä rakenne (kuva 12). Rakenteesta huomattiin heti tiedoston sisällölle tarkoitetun Content-parametrin pituus, joka tekisi rakenteesta vaikeammin luettavan suurempien tiedostojen kanssa ja Frends-prosessi saattaisi ottaa suurehkon datan vastaan hieman hitaammin. Tämän takia lähdettiin selvittämään, miten tiedoston sisältö voitaisiin kompressoida mahdollisimman lyhyeksi ilman, että sen sisältö vaurioituisi. Hetken tutkiskelun jälkeen löydettiin Googlen (13) kehittämä Brotli-kompressio-algoritmi, jonka avulla saataisiin tiivistettyä 30000 merkkiä sisältävän merkkijono vain seitsemään, mutta tämä oli saatavilla vain C- ja C++-ohjelmointikielille. Lopulta kuitenkin löydettiin toinen versio BrotliSharpLib (14), joka oli Googlen Brotlista käännetty C#-ohjelmointikielelle ja tätä voitaisiin

hyödyntää, sillä se on lisensoitu MIT (Massachusetts Institute of Technology) -lisenssillä. Kompressointia ei kuitenkaan voitaisi lisätä suoraan Frends-prosessiin, sillä BrotliSharpLib:n NuGet-paketti ei ole Frends-integraatioalustalta löytyvien nimiavaruuksien joukossa. Kompressoinnin lisääminen prosessiin vaatisi täten uuden Frends-tehtävän luomista, joka tulisi viemään ylimääräistä aikaa, joten tämä ominaisuus päätettiin lisätä vain siinä tapauksessa, jos aikaa riittäisi tämän insinöörityön valmistumisen jälkeen.



```
{
    "TestCases": [
        {
            "Enabled": true,
            "Name": "Normal run with correct data",
            "Description": "Source and destination directories are empty, file is moved to source and process moves the file to destination successfully.",
            "InitializationInstructions": [
                {
                    "Operation": "Directory Clean",
                    "Directory": "C:\\FRENDS\\Oppari\\Example\\Source",
                    "FileMask": "*",
                    "Protocol": "SMB",
                    "UseCredentials": false
                },
                {
                    "Operation": "Directory Clean",
                    "Directory": "C:\\FRENDS\\Oppari\\Example\\Destination",
                    "FileMask": "*",
                    "Protocol": "SMB",
                    "UseCredentials": false
                },
                {
                    "Operation": "File Create",
                    "Directory": "C:\\FRENDS\\Oppari\\Example\\Source",
                    "FileName": "ExampleFile.xml",
                    "Protocol": "SMB",
                    "UseCredentials": false,
                    "Content": "PFJvb3Q+Cgk8R3JvdXBzPgoJCTxHcm91cD4KCQkJPE5hbWU+R3JvdXAgMTwvTmFtZT4KCQkJPFN1cHBsaWVzPkVub3VnaDwvU3VwcGxpZXM+CgkJCTxQZW9wbGU+M
                    "Encoding": "UTF8"
                }
            ]
        }
    ],
    "ProtocolSettings": {
        "SFTP": [
            {
                "Name": "SFTP 1",
                "Address": "#env.Tatu_Oppari.SFTP_Address",
                "Port": "#env.Tatu_Oppari.SFTP_Port",
                "Username": "#env.Tatu_Oppari.SFTP_Username",
                "Password": "#env.Tatu_Oppari.SFTP_Password"
            }
        ],
        "SMB": [
            {
                "Name": "Default Fileshare",
                "Username": "",
                "Password": ""
            }
        ]
    }
}
```


Kuva 12. Testitapauspohjaisen alustusprosessin konfigurointi-JSON-rakenteen esimerkki.

Parametrien suunnittelun yhteydessä päätettiin luopua myös FTP:hen liittyvistä operaatioista aikarajoitteiden vuoksi, joten kuvan 12 JSON-rakenteesta ei löydy FTP:lle omaa kenttää tunnuksia varten. Kyseisestä rakenteesta huomataan

myös, että SFTP:tä varten on annettu tiedot valmiiksi, mutta niitä ei hyödynnetä tässä tapauksessa mihinkään.

Jotta kyseistä JSON-rakennetta voitaisiin tulkita oikein, luotiin taulukko (liite 3), josta selviää jokaisen avainparin selitys, tyyppi ja esimerkit niiden arvoista. Hyvin oleellinen kenttä JSON-rakenteessa on Operation-rakenne, joka kertoo prosessille, mikä toimenpide suoritetaan (kuva 13). Kaikki määritellyt operaatiot ovat mahdollisia, jos kyseessä on SMB-protokolla, mutta hakemiston poistaminen SFTP-palvelimelta ei ole tällä hetkellä mahdollista Frends-tehtävän avulla, joten sitä mahdollisuutta ei prosessiin lisätty.

Operaation nimi	Kuvaus	Esimerkki
Directory Create	Operaatio luo hakemiston	Directory Create
Directory Clean	Operaatio tyhjentää hakemistosta määritellyt tiedostot	Directory Clean
Directory Delete	Operaatio poistaa hakemiston	Directory Delete
File Create	Operaatio luo tiedoston hakemistoon	File Create
File Move	Operaatio siirtää tiedostoja määritellystä lähdehakemistosta kohdehakemistoon annetulla tiedostomaskilla	File Move
File Copy	Operaatio kopioi tiedostoja määritellystä lähdehakemistosta kohdehakemistoon annetulla tiedostomaskilla	File Copy

Kuva 13. Tiedostopohjaisen alustusprosessin mahdolliset operaatiot.

Mahdollisia operaatioita on siis yhteensä kuusi kappaletta, joiden avulla voidaan suorittaa yleisimpiä tiedostoihin liittyviä toimenpiteitä. Operaatioiden esimerkit sisältävät kirjoitustyylin, jolla ne tulisi antaa Operation-kentälle, mutta tämän kirjainkoolla ei ole merkitystä, sillä se ei vaikuta prosessin toiminnallisuuteen.

Konfigurointi-JSON-rakenteen valmistuttua aloitettiin selvittämään pakollisia kenttiä SMB- ja SFTP-operaatioille, sillä ne sisälsivät hieman erilaisia parametrejä. Tämän tarkoituksena oli helpottaa konfigurointi-JSON-rakenteen

toiminnallisuuden ymmärtämistä ja samalla kehitettävän prosessin toteutusta. Ensin luotiin JSON-skeema (kuva 14), jonka avulla voitaisiin validoida konfigurointi-JSON-rakenteesta TestCases-taulukon alta yksittäinen olio, jonka Enabled-rakenteen totuusarvomuuttuja olisi tosi. Prosessin olisi tarkoitus toteuttaa vain yhden tämän kaltaisen olion toiminnallisuudet eli InitializationInstructions-taulukon alta löytyvät operaatiot yhden ajon aikana. Skeeman avulla voitaisiin myös varmistaa kyseisen olion rakenne riippumatta käytettävästä protokollasta tai operaatiosta, eli se toimisi alustavana validointina.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
      "Enabled": { "type": "boolean" },
      "Name": { "type": "string" },
      "Description": { "type": "string" },
      "TempWorkDirectory": { "type": "string" },
      "InitializationInstructions": {
          "type": "array",
          "items": {
              "type": "object",
              "properties": {
                  "Operation": { "type": "string", "minLength": 9 },
                  "Directory": { "type": "string" },
                  "SourceDirectory": {"type": "string"},
                  "DestinationDirectory": {"type": "string"},
                  "FileMask": {"type": "string"},
                  "FileName": { "type": "string" },
                  "Protocol": { "type": "string", "minLength": 3 },
                  "ProtocolName": {"type": "string"},
                  "UseCredentials": { "type": "boolean" },
                  "Content": { "type": "string" },
                  "Encoding": { "type": "string" }
              }
          },
          "required": ["Operation", "Protocol", "UseCredentials"]
      }
  },
  "required": ["Enabled", "InitializationInstructions"]
}
```

Kuva 14. JSON-skeema, jonka avulla konfigurointi-JSON-rakenteen TestCases-taulukon alta löytyvä olio validoidaan alustavasti.

JSON-skeemassa pakollisia kenttiä ovat Enabled-kenttä ja InitializationInstructions-taulukko, koska prosessin on tärkeää tietää, mitkä alustukset suoritetaan. Lisäksi muut pakolliset kentät, kuten Operation, Protocol ja UseCredentials, ovat välttämättömiä prosessin suoriutumisen kannalta. Loput skeemassa näkyvistä arvoista ovat joko operaatiokohtaisia, protokollasta riippuvia tai vapaaehtoisia kenttiä. Skeeman tarkoituksena on siis tarkistaa kyseisen olion rakenne, ei sen sisältöä tässä vaiheessa.

SMB- ja SFTP-operaatioita varten luotiin taulukko (taulukko 2), jonka avulla konfigurointi-JSON-rakenteen luominen ja ymmärtäminen helpottuisi entisestään. Kyseinen taulukko sisältää operaatioiden pakolliset kentät, mutta siihen ei ole sisällytetty aiemmin mainittuja pakollisia kenttiä, mutta niitä edellytetään kuitenkin. Tässä vaiheessa on hyvä mainita, että tunnuksia käytettäessä, eli kun UseCredentials-totuusarvomuuttujan arvo on tosi, tulee ProtocolSettings-olion alta löytyä käytettävän protokollan arvolla avain-arvopari, joka sisältää tunnukset. Useamman tunnusten käyttö vaatii myös ProtocolName-rakenteen käyttämistä, jonka täytyy vastata täysin kyseessä olevan operaatio-olion Name-kentän arvoa. Näin voidaan tarvittaessa hyödyntää montaa eri SFTP-palvelinta tai SMB:tä erottaen tunnukset toisistaan.

Taulukko 2. Alustusprosessin operaatioiden pakolliset kentät.

Operaatio	**Pakolliset kentät**
Directory Create (SMB)	Directory
Directory Clean (SMB)	Directory, FileMask
Directory Delete (SMB)	Directory
File Create (SMB)	Directory, FileName
File Move (SMB)	SourceDirectory, DestinationDirectory, FileMask
File Copy (SMB)	SourceDirectory, DestinationDirectory, FileMask
Directory Create (SFTP)	Directory, TempWorkDirectory
Directory Clean (SFTP)	Directory, FileMask, TempWorkDirectory
File Create (SFTP)	Directory, FileName, TempWorkDirectory
File Move (SFTP)	SourceDirectory, DestinationDirectory, FileMask, TempWorkDirectory
File Copy (SFTP)	SourceDirectory, DestinationDirectory, FileMask, TempWorkDirectory

Tähän mennessä tiedossa oli siis prosessin mahdolliset käyttötapaukset, tietoturvaan liittyvät tiedot, parametrit eli konfigurointi-JSON, jonka avulla prosessin on tarkoitus suorittaa tietyt toimenpiteet ja tämän validointia varten riittävät

tiedot. Näiden tietojen pohjalta luotiin BPMN-kaavio (liite 4), josta selviää kehitettävän Frends-prosessin toiminnallisuus yksinkertaisella tasolla.

Seuraavan luetelman ensimmäisestä toimenpiteestä ilmenee, että luodaan lokia varten muuttuja, vaikka Frends-prosessin instanssista selviäisi tapahtumat. Prosessin instanssi on yhden ajon suorituksen näkymä, josta selviää prosessin kulku ja sen palautusarvot. Loki lisättiin prosessin, jotta prosessin ajosta saataisiin yksityiskohtaisempia tietoja sen suorittamista toimenpiteistä. Täten saataisiin myös selkeä kirjallinen selostus toimenpiteistä, jota voitaisiin mahdollisesti hyödyntää testien dokumentoinnissa.

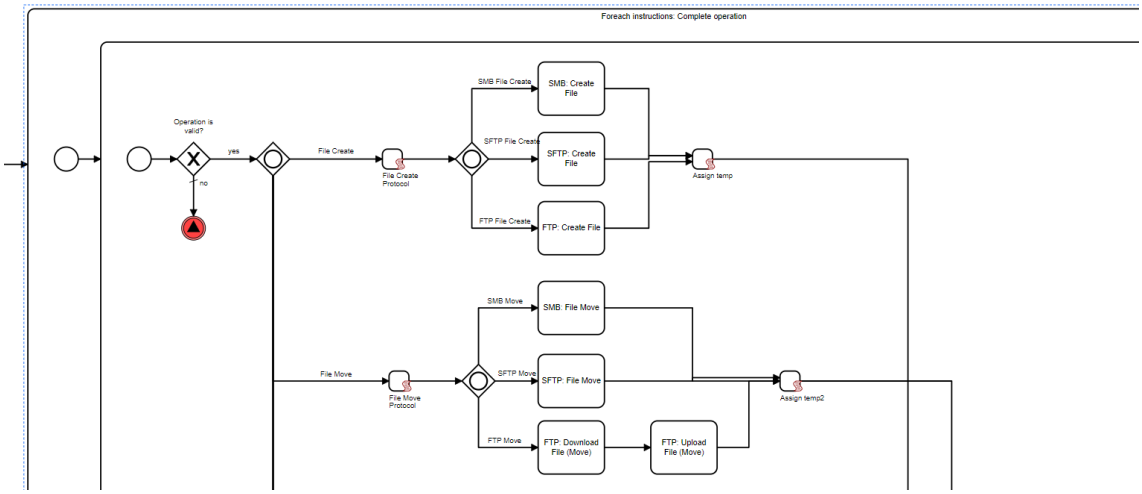
Frends-prosessin BPMN-kaavio (liite 4) sisältää seuraavat toimenpiteet:

1. Alustetaan muuttujat virheviestille ja lokille.
2. Jäsennetään konfigurointi-JSON-parametri JToken-muotoon, jotta sen käsitteleminen helpottuu.
3. Validoidaan konfigurointi-JSON.
4. Tarkistetaan ehtolauseen avulla, onko konfigurointi-JSON-rakenne oikeanlainen. Prosessin ajo lopetetaan virheeseen, jos rakenne sisältää virheen.
5. Tallennetaan muuttujaan konfigurointi-JSON-rakenteesta kaikkien TestCases-taulukon alta löytyvät oliot, joiden Enabled-kentän totuusarvomuuttuja on tosi.
6. Tarkistetaan ehtolauseen avulla, että edellisessä vaiheessa löydettiin vain yksi olio. Useamman olion löytyessä, prosessin ajo lopetettaan virheeseen.
7. Poimitaan muuttujaan löydetyn olion alustusohjeistukset eli InitializationInstructions-taulukko.
8. Toteutetaan toistorakenteen avulla jokainen alustusohjeistus, joka löydettiin edellisessä vaiheessa. Virheen syntyessä luodaan tarkennettu virheviesti, joka talletetaan alussa alustettuun virheviesti-muuttujaan.
9. Lopuksi tarkistetaan sisältääkö alussa alustettu virheviesti-muuttuja mitään ja prosessi päättyy virheeseen, jos tämän arvo ei ole tyhjä.

Vaikka toimenpiteet näyttävät yksinkertaisilta, ne tulevat todellisuudessa olemaan monimutkaisempia. Tämä kuvaus luotiin lähinnä lukijaa varten, sillä todellinen BPMN-kaavio, joka alun perin luotiin oli sen verran monimutkainen, että sen tilalle päätettiin tässä vaiheessa luoda yksinkertaisempi versio.

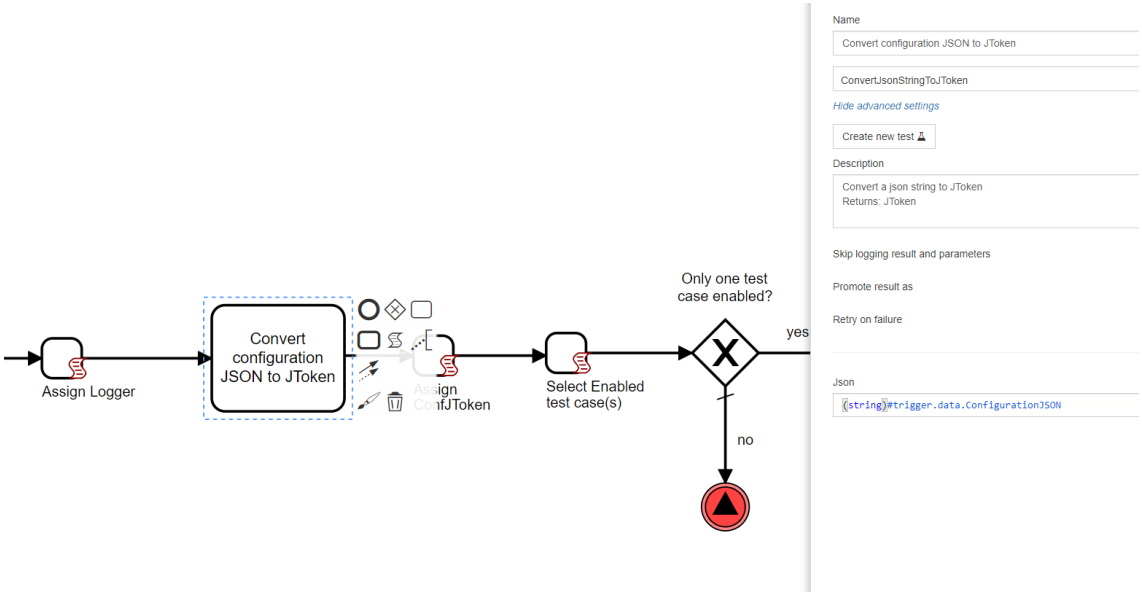
4.3 Toteutus

Toteutus aloitettiin aliluvun 4.2 lopussa mainitusta BPMN-kaaviosta, jota ei lisätty liitteeksi ja ensimmäisenä aloitettiin toteuttamaan operaatioiden toistorakenteen sisältöä. Kuva 15 esittää osaa toistorakenteesta, johon luotiin Frends-tehtävien tyngät eli parametreja näille ei lisätty vielä tässä vaiheessa, koska tarkoituksena oli hahmottaa ensin toiminnallisuuden logiikka. Toistorakenteessa ensimmäinen suoritettava toimenpide on kyseessä olevan operaation tarkistaminen ehtolauseella. Tämän jälkeen valittaisiin oikea haara operaation mukaan. Kuvassa näkyvät haarautumat, jotka alkavat Inclusive decision -elementeistä, toimivat melkein samalla logiikalla kuin yleisesti ohjelmointikielillä switch-case-valintarakenne. Jokaiselle operaatiolle on oma haaransa, joka valitaan operaation mukaan ja seuraavassa toimenpiteessä valitaan protokollan mukaisesti oikea toteutettava haara. Haarojen lopuksi tallennetaan tulos lokimuuttujaan, josta käyttäjälle selviäisi tarkemmin operaation suoriutuminen. Tässä vaiheessa virheidenkäsittely rakennettiin niin, että yhden operaation epäonnistuttua lisättäisiin virhe sille tarkoitettuun muuttujaan ja jatkettaisiin muiden operaatioiden suorittamista. Prosessin seuraavassa versiossa tämä korvattiin tavalla, joka lopettaisi operaatioiden käsittelyn yhdenkin virheen syntyessä operaatioiden mahdollisen riippuvuuden vuoksi toisistaan.



Kuva 15. Alustusprosessin toistorakenteen ensimmäinen versio.

Toistorakenteen tyngän toteutuksen jälkeen, oli tarve lisätä konfigurointi-JSON-rakenteen jäsennys JToken-muotoon, jotta sitä voitaisiin käsitellä helpommin. Kuvasta 16 huomataan, että tämä toteutettiin Frends-tehtävän avulla, joka konvertoi JSON-merkkijonon JToken-muotoon.

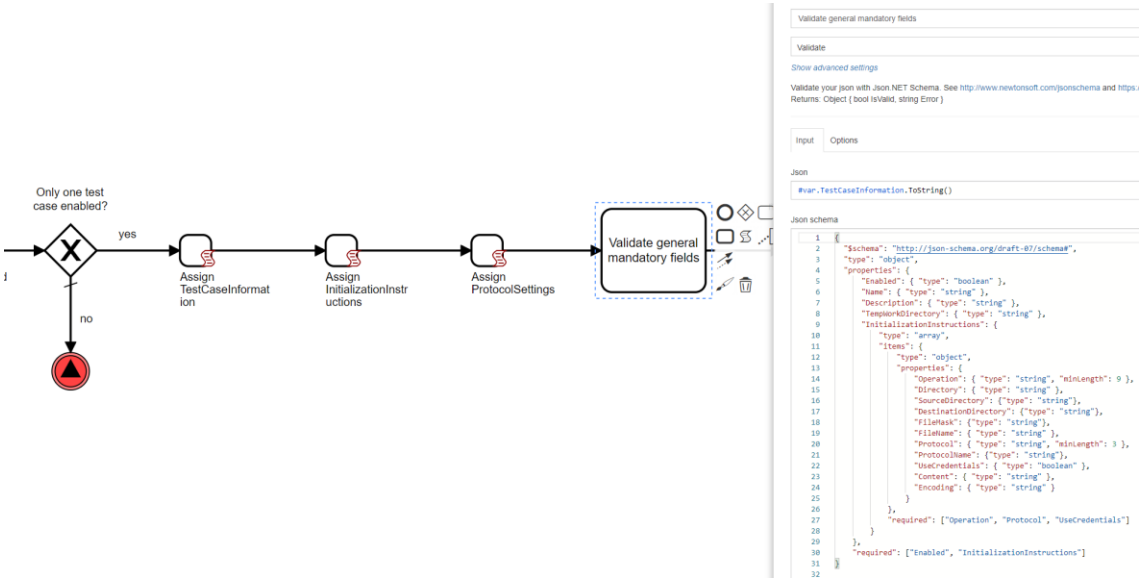


Kuva 16. Frends-tehtävä, joka konvertoi JSON-merkkijonon JToken-muotoon.

Muunnoksen jälkeen (kuva 16) luotiin expression-elementti, jonka tarkoitus on tallentaa ConfJToken-muuttujaan edeltävän muunnoksen tulos. Muunnoksen tulosta voitaisiin myös itsessään käyttää muualla prosessissa #result-viittausta käyttäen, mutta päädyttiin hyödyntämään #var-viittausta selkeyden vuoksi. Seuraavan expression-elementin tarkoitus on valita ConfJToken-parametrilta kaikki oliot TestCases-taulukon alta, joiden Enabled-totuusarvomuuttujan arvo on tosi. Seuraavassa ehtolauseessa vaaditaan, että näitä olioita löytyy vain yksi kappale. Jos olioita löytyy vähemmän tai enemmän, prosessin ajo lopetetaan virheeseen, jossa käyttäjälle ilmoitetaan yksityiskohtaisesti virheestä ja olioiden määrä, joilla Enabled-totuusarvomuuttujan arvo oli tosi.

Seuraavaksi luotiin TestCaseInformation-muuttuja, jonka arvoksi annettiin aktivoitu olio. Samalla luotiin muuttuja InitializationInstructions-taulukolle, jolle annettiin kyseisen olion alustusohjeistukset ja muuttuja ProtocolSettings-oliolle,

jonka arvo saatiin aikaisemmin luodusta ConfJToken-muuttujasta. Kuvasta 17 huomaa tutun JSON-skeeman, jonka tarkoitus on validoida aiemmin mainitun TestCaseInformation-muuttujan rakenne.

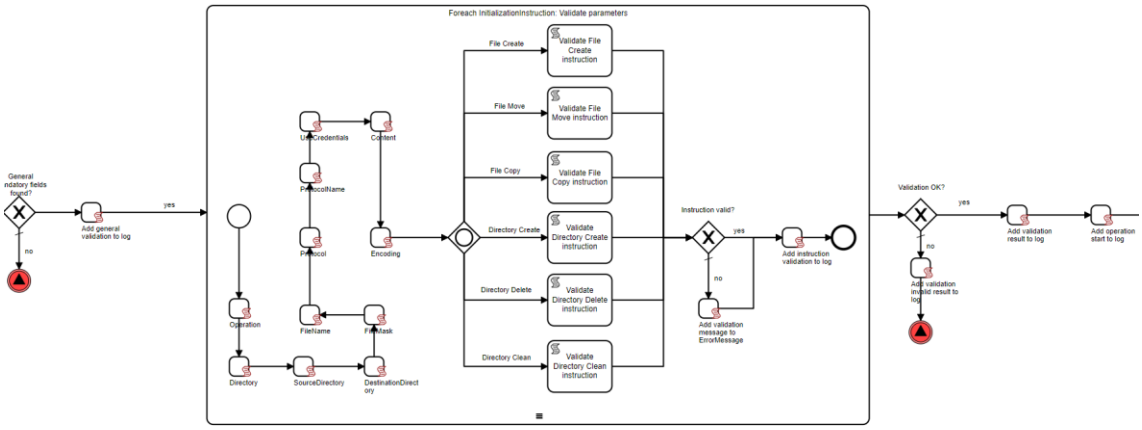


Kuva 17. Oleellisten parametrien alustukset ja TestCaseInformation-muuttujan rakenteen validointi.

Seuraavana aloitettiin rakentamaan TestCaseInformation-muuttujan sisältämien parametrien validointia. Kuvassa 18 ensimmäisen ehtolauseen tarkoitus on tarkistaa, että kuvassa 17 suoritettu validointi onnistui. Lokiin lisättiin myös tulos validoinnin onnistumisesta ennen toistorakenteen suorittamista, mikä näkyy kuvassa 18.

Toistorakenteessa (kuva 18) iteroidaan jokainen olio, joka löytyy InitializationInstructions-muuttujasta. Toistorakenteen alussa (kuva 18) käytetään expression-elementtejä, joihin tarkoituksena on tallentaa iteraatiossa olevan ohjeistuksen parametrejä. Tämän jälkeen siirrytään kyseessä olevan operaation mukaan oikeaan haaraan, jossa ohjeistuksen parametrit validoidaan. Validointien toteutukseen olisi voitu hyödyntää samaa Frends-tehtävää, jota käytettiin myös kuvassa 17, mutta mahdollisten virheiden syntyessä haluttiin tarkemmat

virheviestit. Tämän vuoksi validoinnit päätettiin toteuttaa puhtaasti C#-ohjelmointikielellä (kuva 19).



Kuva 18. InitializationInstructions-taulukon olioiden parametrien validoinnin toistorakenne.

Validointi (kuva 19) on tarkoitettu File Create -operaatiolle, jossa yksinkertaisesti tarkistetaan protokollan mukaisesti, että se sisältää tarpeelliset parametrit. Validointi palauttaa merkkijonon, jos löydetään virheitä ja muutoin palautetaan tyhjä merkkijono. Kuvasta 19 huomataan myös, että vain SMB- ja SFTP-protokolla ovat sallittuja, joten virheellisestä protokollasta palautettaisiin myös merkkijono, joka sisältää yksityiskohtaisen virheviestin.

```
{
    // Validation for File Create operation
    string returnValue = String.Empty;

    // Validate according to protocol (SMB, SFTP)
    switch(#var.Instruction.Protocol?.ToString())
    {
        case "SMB":
            // Directory and FileName are mandatory
            if(string.IsNullOrWhiteSpace(#var.Directory) || string.IsNullOrWhiteSpace(#var.FileName))
                returnValue = "Missing 'Directory' and/or 'FileName' fields.";

            // When credentials are used and there are more than one SMB option, ProtocolName is mandatory
            if(#var.UseCredentials && #var.ProtocolSettings.SMB.Count > 1 && string.IsNullOrWhiteSpace(#var.ProtocolName))
                returnValue = "ProtocolName must be used when there are more than one SMB options and credentials are used.";

            break;

        case "SFTP":
            // Directory and FileName are mandatory
            if(string.IsNullOrWhiteSpace(#var.Directory) || string.IsNullOrWhiteSpace(#var.FileName))
                returnValue = "Missing 'Directory' and/or 'FileName' fields.";

            // TempWorkDirectory is mandatory when using File Create with SFTP
            if(string.IsNullOrWhiteSpace(#var.TestCaseInformation.TempWorkDirectory?.ToString()))
                returnValue = "Missing 'TempWorkDirectory' field.";

            // When credentials are used and there are more than one SFTP option, ProtocolName is mandatory
            if(#var.UseCredentials && #var.ProtocolSettings.SFTP.Count > 1 && string.IsNullOrWhiteSpace(#var.ProtocolName))
                returnValue = "ProtocolName must be used when there are more than one SFTP options and credentials are used.";

            break;
        default:
            // SMB and SFTP are the only valid protocol options.
            returnValue = $"Invalid 'Protocol' used. The protocol was '{#var.Protocol}' only 'SMB' and 'SFTP' are valid ones.";
            break;
    }

    return returnValue;
}
```

Kuva 19. File Create -operaation validointi C#-ohjelmointikielellä.

Jokaisen operaation validoinnit ovat identtisiä rakenteeltaan, mutta joidenkin operaatioiden kohdalla vaaditut parametrit vaihtelevat, joten niissä on kuitenkin pieniä eroja.

Edellä esitellyn toistorakenteen (kuva 18) iteraation validoinnin jälkeen tarkistetaan validointitulos eli palautettu merkkijono ja tämän sisältäessä tekstiä lisättäisiin virhe JArray-taulukkoon, joka sisältää prosessissa tapahtuvia virheitä. Lisäksi lokimuuttujaan lisätään tieto iteraatiossa olevan operaation validoinnin suorittamisesta. Toistorakenteen jälkeisessä ehtolauseessa (kuva 18) tarkistetaan virheitä sisältävän muuttujan sisältö. Jos se on tyhjä, jatketaan prosessin ajoa normaalisti ja päivitetään lokimuuttujaa. Muussa tapauksessa prosessin ajo lopetetaan virheeseen.

Tähän mennessä oli validoitu käyttäjän määrittelemä konfigurointi-JSON melko kattavasti ja siirryttiin toistorakenteeseen, jossa tarkoituksena on suorittaa käyttäjän määrittelemät operaatiot.

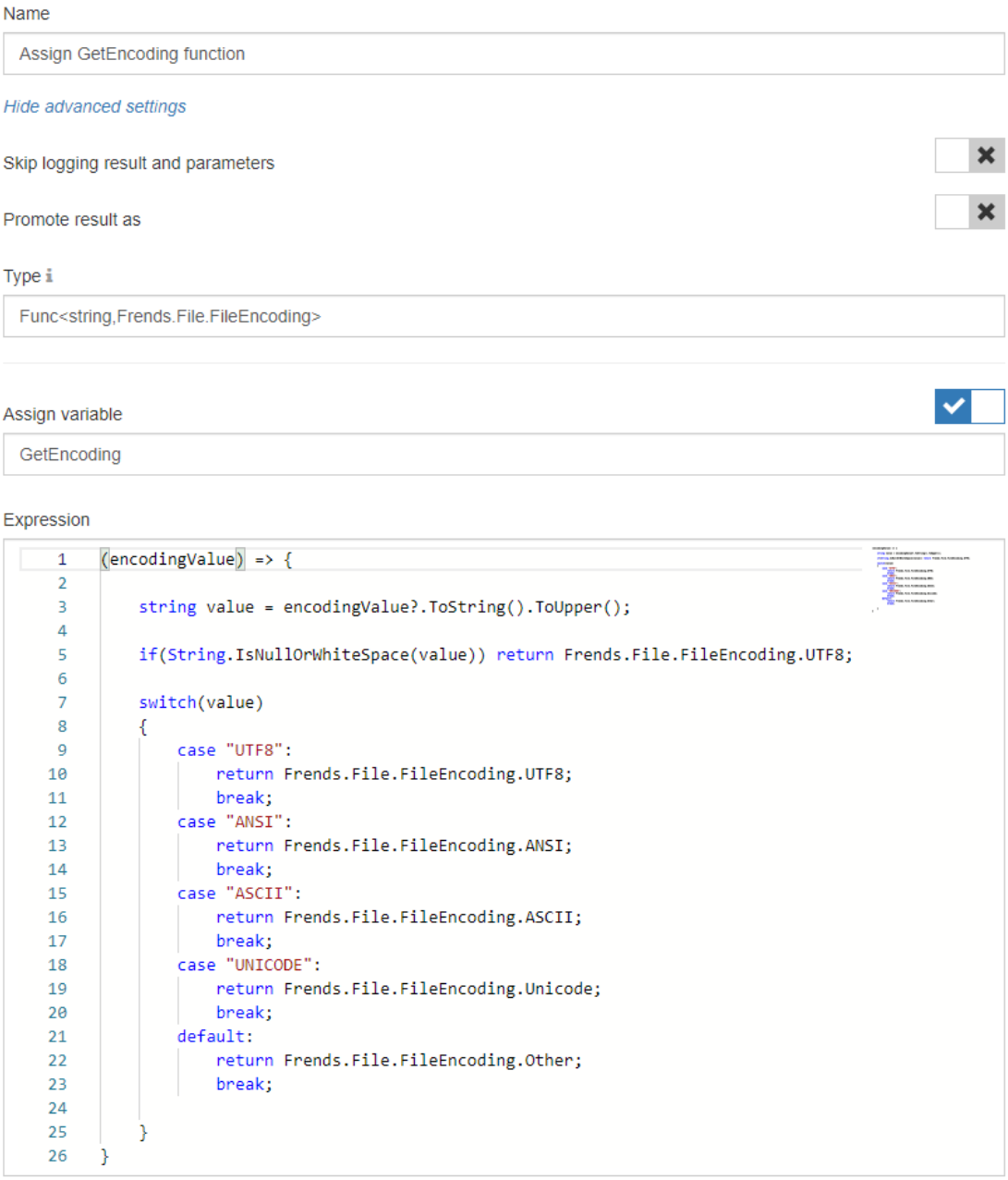
Ensin lähdettiin selvittämään mitä tiedoston koodauksia on mahdollista käyttää, kun halutaan kirjoittaa tiedosto hakemistoon. Nämä selvitettiin tutkimalla Frends.File.Write-tehtävän lähdekoodia, joka löytyy GitHub-versionhallintajärjestelmästä. Seuraava luettelo sisältää Frends.File.FileEncoding enum -muuttujien arvot.

Tuettuja tiedoston sisällön koodauksia ovat:

- UTF8
- ANSI
- ASCII
- Unicode
- Other.

Nämä luetellut koodaukset voidaan suoraan valita, kun käytetään Frends.File.Write-tehtävää. Luetelman viimeisellä enum-muuttujalla Other on mahdollisuus käyttää myös toisenlaista koodausta tarvittaessa. Tätä varten täytyi selvittää lähdekoodista vielä, miten tämän valinnan kanssa koodauksen valinnan toiminnallisuus on toteutettu. Täten saatiin selville, että käytetään System.Text.Encoding.GetEncoding(string)-metodia. Microsoftin dokumentaation (15) mukaisesti koodaus haettaisiin tietyltä koodisivulta, jota parametrina annettu merkkijono vastaa. Annetun parametrin täytyisi vastata Microsoftin dokumentaatiosta (16) otsikon List of encodings alta löytyvän Name-arvoa ja kaikkia koodauksia ei tueta, mikä riippuu käytettävästä .NET-versiosta.

Koodauksen toiminnallisuuden selvityksen jälkeen luotiin funktio (kuva 20), jonka tarkoituksena olisi selvittää, mikä Frends.File.FileEncoding enum -arvotyyppi annettaisiin parametrina Frends.File.Write-tehtäville oikean koodauksen valintaa varten.

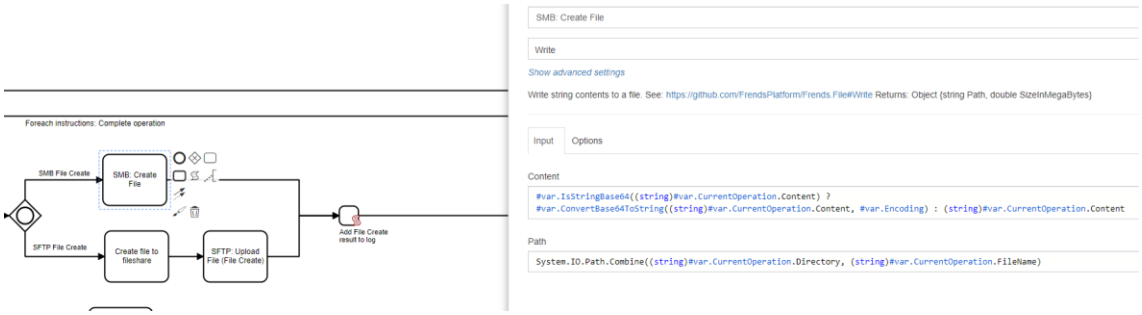


Kuva 20. GetEncoding-funktio Frends.File.FileEncoding enum -arvotyypin selvittämiseksi.

Käyttäjän määrittelemän koodauksen ollessa jokin muu kuin kuvan 20 tapauksissa, palautetaan Frends.File.FileEncoding.Other enum -arvotyyppi ja tällöin käyttäjän määrittelemän koodausta käytetään aiemmin mainitun System.Text.Encoding.GetEncoding(string)-metodin parametrina, kun Frends.File.Write-tehtävä käynnistyy. Jos koodausta ei löydetä, prosessi

lopetetaan virheeseen. Koodausta ei myöskään ole pakollista antaa, ja tällöin palautettaisiin Frends.File.FileEncoding.UTF8 enum -arvotyyppi.

Seuraavana vuorossa oli Frends.File.Write-tehtävän parametrisointi (kuva 21), ja päätettiin, että käyttäjän määrittelemä tiedostonsisältö voitaisiin antaa merkkijonona tai base64-koodattuna merkkijonona. Kuvassa 21 näkyvän IsString-Base64-funktion tarkoitus on tarkistaa, onko tiedostonsisältö base64-koodattu ja palauttaa totuusarvomuuttuja. Tämän totuusarvomuuttujan perusteella valitaan tiedostonsisältö sellaisenaan tai sitten muunnetaan tiedostonsisältö, joka on base64-koodattu ConvertToBase64ToString-funktion avulla, joka kääntää base64-koodatun merkkijonon käyttäjän määrittelemällä koodauksella merkkijonoksi hyödyntäen System.Text.Encoding.<Käyttäjän määrittämä koodaus>.GetString(byte[])-metodia. Tämä funktio hyödyntää aiemmin mainittua System.Text.Encoding.GetEncoding(string)-metodia, kun koodaus on jokin muu kuin UTF8, ANSI, ASCII tai Unicode. Oikea koodaus valitaan siis switch-case-valintarakenteen avulla.

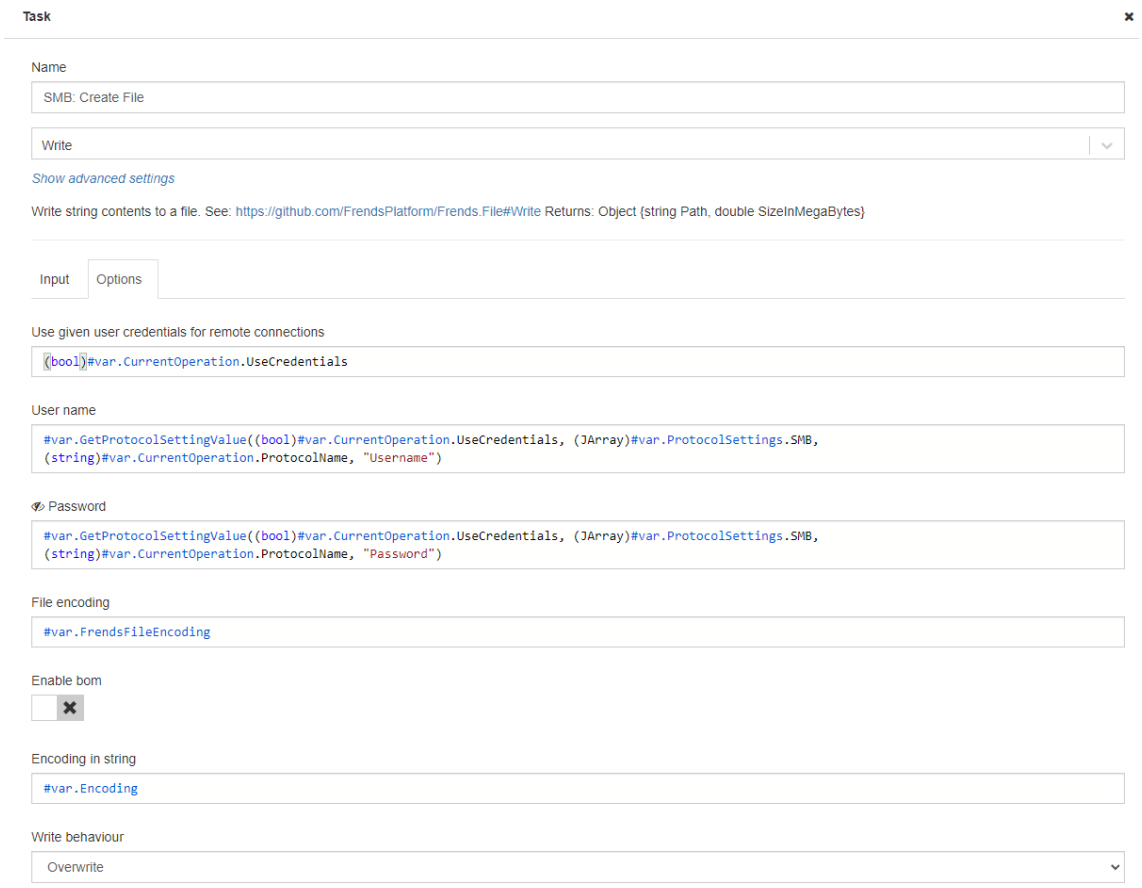


Kuva 21. Frends.File.Write-tehtävän Input-välilehden parametrinäkymä.

Protokollahaaroissa olevat operaatiot, joihin on mahdollista lisätä tunnusten käyttäminen tai muut yhteyksiin liittyvät tiedot, toteutettiin SMB-operaatiolle kuvan 22 tapaisesti. SFTP-operaatioiden vastaavat parametrisoinnit on jätetty tästä dokumentista pois, sillä se toisi turhaa toistuvuutta.

Tunnuksia käytetään vain silloin, jos UseCredentials-totuusarvomuuttujan arvo on tosi. Tällöin käyttäjätunnuksen tiedot haetaan GetProtocolSettingValue-

funktiolla (kuva 23), jota käytetään myös muiden protokolla-asetusten hakemiseen. Kuvassa 22 olevan FileEncoding-parametrin arvo selvitettiin aikaisemmassa vaiheessa Frends.File.FileEncoding enum -arvotyypiksi, ja EncodingInString-parametrin arvona käytetään käyttäjän määrittelemää koodauksen merkkijonoa. FileEncoding-parametrin arvon ollessa Frends.File.FileEncoding.Other, pyritään hakemaan EncodingInString-parametrin arvolla sitä vastaavaa koodausta Encoding.GetEncoding(string)-metodilla.



Kuva 22. Frends.File.Write-tehtävän Options-välilehden parametrinäkymä.

GetProtocolSettingValue-funktiolle annetaan seuraavat parametrit:

- Totuusarvomuuttuja on UseCredentials-kentän arvo.
- JArray on kyseessä olevan protokollan avulla etsittävä taulukko. Sisältää protokollaan liittyviä yhteystietoja.
- Merkkijono kertoo kyseessä olevan protokollan nimen.

- Merkkijono kertoo haettavan kentän.

Type

Func<bool, JArray, string, string, string>

Assign variable

GetProtocolSettingValue

Expression

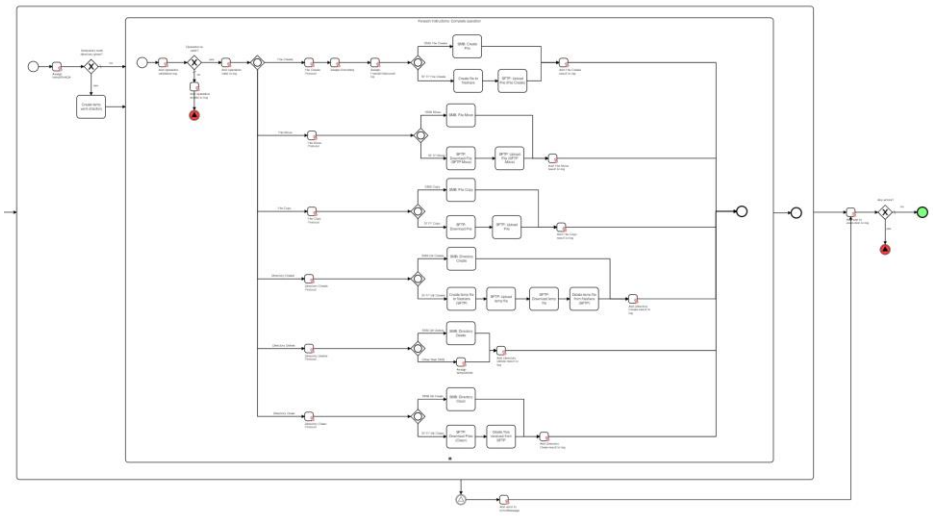
```
(useCredentials, protocolOptions, protocolName, valueToRetrieve) => {

    if(!useCredentials || protocolOptions == null || protocolOptions.Count < 1) return null;

    if(protocolOptions.Count == 1)
    {
        string value = (string)protocolOptions.First()[valueToRetrieve];
        return String.IsNullOrEmpty(value) ? null : value;
    }
    else
    {
        if(String.IsNullOrWhiteSpace(protocolName)) throw new ArgumentException("ProcolName is mandatory when more than one protocol option is used!");

        string value = protocolOptions.Where(protocol => ((string)protocol["Name"]).Equals(protocolName)).Select(protocol =>
protocol[valueToRetrieve]).First()?.ToString();
        return String.IsNullOrEmpty(value) ? null : value;
    }
}
```

Kuva 23. GetProtocolSettingValue-funktio, jonka avulla haetaan arvoja ProtocolSettings-oliolta.

GetProtocolSettingValue-funktiolla pyritään yksinkertaistettuna palauttamaan kyseessä olevan protokollan yhteyteen liittyvä arvo. Protokollien asetuksia ei tarvitse käyttää kaikissa tilanteissa, mutta jos niitä tarvitaan ja tietty arvo puuttuu, ajautuu operaation Frends-tehtävä virheeseen.

Tähän mennessä oli siis saatu luotua parametrit, jotta voitaisiin ensimmäisen kerran testata File Create -operaatiota SMB-protokollalla. Testauksen aikana suurimmat virheet syntyivät syntaksien takia, joten niiden raportointia ei koettu oleelliseksi tämän työn kannalta.

Seuraavaksi täytettiin loputkin parametrit ja tässä vaiheessa oltiin tyytyväisiä operaatioiden lopulliseen toistorakenteeseen (kuva 24).



Kuva 24. Operaatioiden suorittamisen lopullinen toistorakenne.

Kehityksen aikana paranneltiin lokia ja sen ulkomuotoa, kunnes koettiin sen olevan riittävän selkeä. Tavoitteena oli mahdollisimman tarkasti kuvata prosessin suorittamia toimenpiteitä.

Tässä vaiheessa koettiin, että prosessi olisi valmiina testaukseen, kun sitä oli onnistuneesti ajettu useaan otteeseen yksinkertaisilla ohjeistuksilla. Lopullisen alustusprosessin BPMN-kaavio löytyy liitteestä 5.

4.4 Testaus

Kehitettävän prosessin laadun ja toiminnallisuuden varmistamiseksi täytyy sitä testata mahdollisimman kattavasti ja huomioida, että sille asetetut tavoitteet toteutuvat. Testaamisella varmistetaan, että kehitetty prosessi vastaa vähintään odotuksia ja toimii odotetusti.

Frends-integraatioalustalla testaaminen poikkeaa hieman tavanomaisesta ohjelmoinnista, jota voitaisiin tehdä esimerkiksi Visual Studio -ohjelmointiympäristössä. Tarkastellaan esimerkiksi testikattavuutta. Visual Studiolla testikattavuutta nostettaisiin luomalla testitapauksia kohdille, joita nykyiset yksikkötestit eivät kata, mutta Frends-integraatioalustalla prosessille ei ole mahdollista luoda

tämänkaltaisia testejä. Tämän vuoksi prosessille joudutaan suorittamaan yksikkötestejä manuaalisesti tai syöttämään erilaisia parametreja prosessille, joiden avulla voidaan testata eri osia prosessista. Kehitettyä alustusprosessia oli hieman hankala testata yksikkötestaamalla eivätkä niiden tulokset välttämättä olisi tarpeeksi luotettavia, joten prosessia päätettiin testata pääosin prosessille annettavan parametrin muutosten avulla.

Kehitetyn prosessin testauksessa keskityttiin pääosin toiminnalliseen ja ei-toiminnalliseen testaukseen ja tarkemmin järjestelmätestaukseen. Tarkoituksena oli testata, että kaikki vaatimukset toteutuvat ja prosessin toiminnallisuudet sekä muut ominaisuudet toimivat suunnitellusti (7). Prosessin suorituskykyä arvioitiin myös tavoitteenaan selvittää sen kyky käsitellä eri kokoisia tiedostoja ja tunnistaa mahdolliset rajoitukset tai rajoitteet.

4.4.1 Toiminnallinen testaus

Ennen prosessin toiminnallisuuksien testausta täytyi ensin luoda tarpeeksi kattavat testitapaukset, jotta voitaisiin todeta prosessi toimivaksi (17). Testitapausten (liite 6) taulukkoon lisättiin testitapauksia prosessille asetettujen vaatimusten mukaisesti ja mukaan sisällytettiin myös testitapauksia, joilla voitiin testata prosessin muiden ominaisuuksien oleellisia osia muuttaen konfigurointi-JSON-rakenteen sisältöä. Prosessin konfigurointi-JSON voi myös sisältää virheellisiä tai puutteellisia tietoja, joten luotiin näitä varten myös muutamia testitapauksia.

Testitapausten määrää jouduttiin hieman karsimaan, ja liitteestä 6 huomataan myös, että testidata-sarakkeen data ei sisällä mitään, mutta liitteestä löytyvän Testitapauksen Nimi -sarakkeen arvolla voi etsiä käytetyt parametrit testitapauksia 1–7 varten luodusta konfigurointi-JSON-rakenteesta (liite 7).

Suoritettujen testitapausten perusteella todettiin, että prosessin operaatiot toimivat SMB- ja SFTP-protokollien kanssa moitteettomasti, kun annetut parametrit oli syötetty oikein. Parametrien puutuessa tai niiden ollessa virheellisiä suoriutui prosessi myös odotusten mukaisesti ja palautti virheviestin, joka sisälsi tiedon

virheestä, jonka perusteella käyttäjä osaisi korjata mahdollisen ongelman konfigurointi-JSON-rakenteesta.

Testien perusteella voidaan myös todeta, että prosessille laaditut vaatimukset täytettiin, mutta liitteestä 1 löytyvää käyttäjätarinaa 11 ei päätetty lopulta toteuttaa sen monimutkaisuuden ja aikarajoitteiden vuoksi. Kyseinen toiminto olisi hyödyllinen ja täten se todennäköisesti lisättäisiin prosessiin myöhemmin.

4.4.2 Ei-toiminnallinen testaus

Ei-toiminnalliseen testaukseen päätettiin sisällyttää kehitetyn alustusprosessin suorituskyvyn testaus, jossa tarkoituksena oli testata sen kyvykkyyttä käsitellä eri kokoisia tiedoston sisältöjä. Prosessin turvallisuutta ei testattu, mutta varmistettiin, että salattavia tietoja, kuten salasanat SMB:lle ja SFTP-palvelimelle, ei näkyisi prosessin ajosta.

Suorituskyvyn testausta varten luotiin eri kokoisia XML-tiedostoja, joiden sisältö koodattiin base64-merkkijonoiksi Visual Studion avulla ja käytettiin valmista konfigurointi-JSON-pohjaa, jonka avulla luotiin tiedosto File Create -operaatiolla. Taulukosta 3 löytyvät käytettyjen tiedostojen koot, base64-koodattu merkkijonon pituus, prosessin ajon kesto ja onnistuttiinko kyseinen tiedosto luomaan.

Taulukko 3. Suorituskykytestauksen tulokset

Koko (kt = kilotavu ja MB = megatavu)	**Pituus**	**Kesto**	**Tulos**
1kt	596	< 1 s	Onnistui
21kt	27 365	< 1 s	Onnistui
411kt	420 580	< 1 s	Onnistui
3697kt	4 901 240	1 s	Onnistui
17MB	23 232 268	4 s	Onnistui
136MB	185 857 548	Ei tiedossa	Epäonnistui

Taulukosta 3 nähdään, että tiedoston koon ollessa 17 megatavua tai vähemmän tiedoston koodaus base64-merkkijonoksi, konfigurointi-JSON-rakenteen kopiointi alustusprosessin parametriksi, prosessin käynnistäminen ja prosessin ajo onnistuivat. 17-megatavuisella tiedostolla havaittiin kuitenkin hieman hitautta prosessin käynnistämisessä, joka kesti 24 sekuntia. Tiedoston koon ollessa 136 megatavua ilmentyi merkittävää hitautta tiedoston sisällön koodauksessa ja Frends-alusta kaatui, kun yritettiin kopioida parametria prosessille eli prosessin ajoa ei päästy kokeilemaan ollenkaan.

Näiden tulosten perusteella alustusprosessia voitaisiin hyödyntää tiedostojen luomiseen silloin, kun tiedoston koko olisi noin 17 megatavua tai vähemmän. Isompien tiedostojen kanssa jouduttaisiin keksiä toinen ratkaisu tiedoston luomiseksi haluttuun hakemistoon.

4.5 Käyttöönotto ja ylläpito

Kehitetty alustusprosessi on tämän raportin kirjoitushetkellä saatavilla vain yhdestä Frends-ympäristöstä, johon kaikilla ei välttämättä ole pääsyä. Joten tarvittaisiin ratkaisu, miten tämä prosessi saataisiin muiden käytettäväksi. Tähän onneksi Frends-integraatioalustalla on olemassa toiminnallisuus, jonka avulla prosessista voidaan tuottaa JSON-tiedosto ja tämä voidaan tuoda toiseen Frends-ympäristöön käytettäväksi.

Tässä vaiheessa on hyvä tuoda esille, että tämän prosessin kehityksen idea oli lähtöisin tämän raportin kirjoittajalta eikä tätä työtä tehty yrityksen tilauksesta. Tämän vuoksi kyseistä prosessia olisi tarkoitus ensin esitellä yksityiskohtaisemmin esihenkilöille ja muille osaavimmille yksilöille, joilta saataisiin arvokasta lisätietoa, miten voisimme edetä kyseisen prosessin käyttöönoton kanssa. Tarkoituksena olisi myös käydä läpi prosessin ja sen käyttöönottoon liittyviä mahdollisia ongelmakohtia, jos niitä sattuisi olemaan.

Kehitettävälle prosessille voitaisiin hyödyntää esimerkiksi GitHub-versionhallintajärjestelmää ja tarpeen tullen voitaisiin yksinkertaisesti ehdottaa muutoksia

vetopyynnöllä (Pull Request). Toistaiseksi versionhallinta kuitenkin päädyttiin toteuttamaan Confluence-sovelluksella, joka on kyseisessä yrityksessä aktiivisesti käytössä. Se sisältää sisäänrakennetun versionhallinnan, joten muutoksia voi seurata helposti ja kaikilla yrityksen työntekijöillä olisi tänne myös pääsy.

Prosessin ylläpitovastuu on suurimmaksi osaksi sen kehittäjällä, kun prosessia ei ole vielä otettu käytänteeksi. Prosessin yleistyessä voitaisiin sen ylläpito siirtää ryhmälle, joka on vastuussa Frends-integraatioalustan ja/tai Frends-tehtävien ylläpidosta.

Prosessia on mahdollista muokata omien tarpeiden mukaisesti eikä tämä vaikuttaisi muiden Frends-ympäristöjen versioihin. Kuka tahansa voi myös tehdä yleisellä tasolla prosessiin ehdotuksia muutoksista, korjauksista tai laajennuksista, mutta toteutus tulisi tällöin katselmoida hyvien käytäntöjen mukaisesti ennen uuden version julkaisemista.

5 Yhteenveto

Insinöörityön tavoitteena oli luoda yleiskäyttöinen prosessi Frends-integraatioalustalle, jota voitaisiin hyödyntää testauksien eri vaiheissa tiedostopohjaisille Frends-prosesseille. Prosessi päätettiin kehittää, sillä tiedostopohjaisten Frends-prosessien testauksia varten luotiin aikaisemmin useita erilaisia prosesseja ja haluttiin yleiskäyttöinen versio. Samalla tutkittiin muutamia ohjelmistotestauksen osa-alueita, miten ne näkyivät Frends-integraatioalustalla ja selvitettiin mahdollisia kehityksenkohteita.

Kehitettävä testitapauspohjainen alustusprosessi toteutettiin vaatimukset täyttäen suurimmaksi osaksi lukuun ottamatta toimintoa, jolla voitaisiin peruuttaa suoritetut toimenpiteet virheen syntyessä. Myös tiedoston sisällön kompressointi ja FTP-operaatioiden toteutus jäivät toteuttamatta, mutta nämä eivät olleet täysin pakollisia ja ne voitaisiin lisätä prosessiin myöhemmin.

Kehitettyä prosessia voidaan siis käyttää testauksien tukena ja sen toiminnallisuuksien vuoksi se vähentäisi muiden ylimääräisten prosessien luomista, jos ne sisältävät samoja toimenpiteitä. Täten sen avulla voidaan säästää aikaa, joka menisi näiden ylimääräisten prosessien luomiseen. Prosessia voidaan hyödyntää sen sisältämien toimenpiteiden vuoksi muihinkin tarkoituksiin kuin testauksen tukeen.

Ohjelmistotestauksen näkökulmasta Frends-integraatioalustalta löydettiin muutama kehityksen kohde, joista saattaisi olla hyötyä kehitettävien prosessien laadun parantamisen kannalta. Yksikkötestaukseen liittyen expression-elementille voitaisiin lisätä mahdollisuus suorittaa yksikkötestejä. Tämän avulla voitaisiin kätevästi testata esimerkiksi expression-elementtejä, jotka palauttavat funktion. Funktion toiminnallisuutta voitaisiin kätevästi testata täten suoraan Frends-alustalla, eikä olisi tarvetta suorittaa testejä toisella ohjelmointiympäristöllä tai jollain muulla tavalla. Järjestelmä- ja hyväksymistestauksiin liittyen voitaisiin kehittää dokumentaatiopohjat, joiden avulla näiden suunnittelu ja tulosten seuranta toteutettaisiin. Järjestelmä- ja hyväksymistestauksen kehityksen kohteet olisi lähinnä tarkoitettu tilanteita varten, joissa asiakas vaatii niitä tai joissa asiakkaalla ei ole valmiita hyväksymistestauksen käytäntöjä.

Alustusprosessin kehityksen aikana huomattiin, kuinka tärkeitä vaatimusmäärittelyiden ja suunnittelun aikana suoritetut tutkimukset ja selvitykset prosessin toteutuksen kannalta ovat. Kehityksen aikana ilmeni monesti pieniä yksityiskohtia, joita ei osattu odottaa. Tämä johtui täysin siitä, että prosessin toteutuksen kannalta ei tehty tarvittavan laajaa selvitystyötä. Lopulta ongelmat kuitenkin selvitettiin ja tavoitteisiin päästiin.

Lähteet

1 Intro tekniseen integraatioon – mihin sitä tarvitaan ja milloin se kannattaa. 2021. Verkkoaineisto. Timehouse. <https://www.timehouse.fi/onnistunut-tekninen-integraatioprojekti//>. Päivitetty 5.11.2021. Luettu 30.10.2023.

2 Galkin, Ossi. 2023. What is Frends? Verkkoaineisto. Frends. <https://docs.frends.com/en/articles/2188944-what-is-frends/>. Päivitetty yli viikko sitten. Luettu 30.10.2023.

3 Virtanen, Riku. 2023. Visual approach in Process development. Verkkoaineisto. Frends. <https://docs.frends.com/en/articles/8010465-visual-approach-in-process-development/>. Päivitetty 12.10.2023 – 18.10.2023. Luettu 25.10.2023.

4 Galkin, Ossi. 2023. Frends Agent. Verkkoaineisto. Frends. <https://docs.frends.com/en/articles/8027130-frends-agent/>. Päivitetty yli viikko sitten. Luettu 30.10.2023.

5 Galkin, Ossi. 2023. What are Tasks? Verkkoaineisto. Frends. <https://docs.frends.com/en/articles/8010710-what-are-tasks/>. Päivitetty yli viikko sitten. Luettu 30.10.2023.

6 Galkin, Ossi. 2023. Creating a Frends API. Verkkoaineisto. Frends. <https://docs.frends.com/en/articles/2188728-creating-a-frends-api/>. Päivitetty yli viikko sitten. Luettu 3.11.2023.

7 Ohjelmiston testaaminen. 2023. Verkkoaineisto. Wikipedia. <https://fi.wikipedia.org/wiki/Ohjelmiston_testaaminen/>. Päivitetty 22.10.2023. Luettu 27.10.2023.

8 Mitä on ohjelmistotestaus ja mitä hyötyä siitä on? 2022. Verkkoaineisto. Vala Group. <https://www.valagroup.com/fi/blogi/mita-on-ohjelmistotestaus-ja-mita-hyotya-siita-on//>. Päivitetty 10.11.2022. Luettu 30.10.2023.

9 Integraatiotestaus: Mitä se tarkoittaa ja mitä hyötyä siitä on? 2023. Verkkoaineisto. Vala Group. <https://www.valagroup.com/fi/blogi/integraatiotestaus//>. Päivitetty 24.10.2023. Luettu 31.10.2023.

10 Järjestelmätestaus eli systeemintestaus: Mitä se on ja miksi se on tärkeää? 2023. Verkkoaineisto. Vala Group. <https://www.valagroup.com/fi/blogi/jarjestelmatestaus//>. Päivitetty 24.10.2023. Luettu 31.10.2023.

11 Keskitalo, Lasse. 2012. Hyväksymistestaus ohjelmistokehityksessä – Prosessi, tehtävät ja roolit. Verkkoaineisto. Theseus. <https://www.theseus.fi/bitstream/handle/10024/48825/Keskitalo_Lasse.pdf;jsessionid=9D2BB50B76C90E33C7594F8B0EE90F17?sequence=1/>. Päivitetty 4.10.2012. Luettu 31.10.203.

12 What is Requirement Analysis. 2023. Verkkoaineisto. Simplilearn. <https://www.simplilearn.com/what-is-requirement-analysis-article/>. Päivitetty 19.10.2023. Luettu 22.10.2023.

13 Brotli. 2014. Verkkoaineisto. GitHub. <https://github.com/google/brotli/>. Päivitetty 3.8.2023. Luettu 30.10.2023.

14 BrotliSharpLib. 2017. Verkkoaineisto. GitHub. <https://github.com/master131/BrotliSharpLib/>. Päivitetty 9.5.2020. Luettu 30.10.2023.

15 Encoding.GetEncoding Method. 2023. Verkkoaineisto. Microsoft. <https://learn.microsoft.com/en-us/dotnet/api/system.text.encoding.getencoding?view=netframework-4.7.1/>. Luettu 27.10.2023.

16 Encoding Class. 2023. Verkkoaineisto. Microsoft. <https://learn.microsoft.com/en-us/dotnet/api/system.text.encoding?view=netframework-4.7.1/>. Luettu 27.10.2023.

17 Software Testing – Test Case. 2023. Verkkoaineisto. GeeksforGeeks. <https://www.geeksforgeeks.org/software-testing-test-case/>. Päivitetty 27.9.2023. Luettu 27.10.2023.

Alustusprosessin käyttäjätarinat

Vaatimukset

Jokaisessa käyttäjätarinassa otetaan huomioon, että hakemisto voi sijaita levyjaolla, FTP- tai SFTP-palvelimella.

#	Käyttäjätarina (Kehittäjänä tahdon...)	Kuvaus	Prioriteetti	Lisätietoja
1	Luoda tiedoston tiettyyn hakemistoon	Luodaan vain yksi tiedosto määriteltyyn hakemistoon	1	Tiedoston luonnin yhteydessä olisi mahdollista luoda hakemisto, johon tiedostoa oltaisiin luomassa, jos tätä ei ole vielä olemassa.
2	Luoda tiedoston moneen eri hakemistoon	Luodaan yksi tiedosto, joka viedään määriteltyihin hakemistoihin	2	
3	Luoda erilaisia tiedostoja eri hakemistoihin	Luodaan erilaisia tiedostoja määriteltyihin hakemistoihin	1	
4	Luoda erilaisia tiedostoja tiettyyn hakemistoon	Luodaan erilaisia tiedostoja yhteen määriteltyyn hakemistoon	1	
5	Tyhjentää tietyn hakemiston	Tyhjennetään vain yksi määritelty hakemisto	1	
6	Tyhjentää monta eri hakemistoa	Tyhjennetään vain määritellyt hakemistot	1	
7	Luoda 1+ tiedostoa yhteen tai useampaan hakemistoon ja samalla tyhjentää yhden tai useamman hakemiston	Luodaan yksi tai useampi tiedosto yhteen tai useampaan hakemistoon ja tyhjennetään yksi tai useampi hakemisto	1	
8	Saada ulostulona tiedon, mitä prosessi teki	Prosessi palauttaa tiedon siitä mitä se teki ja missä järjestyksessä	2	Tämän tulisi toimia sanallisena kuvauksena, josta voitaisiin mahdollisimman helposti tulkita mikä oli lähtötilanne enne testin suorittamista.
9	Määrittää mitä tiedoston koodausta käytetään, kun luodaan uutta tiedostoa	Tiedoston luontia varten on mahdollista määrittää tiedoston koodaus	1	
10	Luoda hakemiston tai useamman hakemiston	Luodaan yksi tai useampi hakemisto. Hakemiston luonti voitaisiin suorittaa ilman muita operaatioita, jos muille toimenpiteille ei ole tarvetta.	1	
11	Virheen syntyessä, että prosessi poistaisi tiedostot ja hakemistot, jotka se ehti luoda, jotta säästytään ylimääräisiltä toimenpiteiltä.	Virheen syntyessä olisi mahdollisuus palata alkuperäiseen tilaan, jotta seuraavan ajon yhteydessä voidaan aloittaa puhtaalta pöydältä ja näin säästytään ylimääräisiltä ajoilta.	3	

Alustusprosessissa käytettävien Frends-tehtävien parametrit

Frends Task	Tarvittavat parametrit ja selitykset
Frends.File.Write	Content. Tiedoston sisältö. Path. Tiedoston sijainti sisältäen hakemiston ja tiedostonimen. UseGivenUserCredentialsForRemoteConnections. Käytetäänkö tunnuksia vai ei. Username. Käyttäjätunnus. Password. Salasana. FileEncoding. Tiedoston koodaus. EncodingInString. Tiedoston koodaus, jos sitä ei ole vakiona saatavilla.
Frends.File.Move	Directory. Hakemisto, josta tiedostot siirretään. Pattern. Tiedostomaski, jolla siirrettävät tiedostot valitaan lähteestä. TargetDirectory. Kohdehakemisto, johon löydetyt tiedostot siirretään. UseGivenUserCredentialsForRemoteConnections. Käytetäänkö tunnuksia vai ei. Username. Käyttäjätunnus. Password. Salasana.
Frends.File.Copy	Directory. Hakemisto, josta tiedostot kopioidaan. Pattern. Tiedostomaski, jolla kopioitavat tiedostot valitaan lähteestä. TargetDirectory. Kohdehakemisto, johon löydetyt tiedostot kopioidaan. UseGivenUserCredentialsForRemoteConnections. Käytetäänkö tunnuksia vai ei. Username. Käyttäjätunnus. Password. Salasana.
Frends.Directory.Create	Directory. Hakemisto, joka halutaan luoda. UseGivenUserCredentialsForRemoteConnections. Käytetäänkö tunnuksia vai ei. Username. Käyttäjätunnus. Password. Salasana.
Frends.Directory.Delete	Directory. Hakemisto, joka halutaan poistaa. UseGivenUserCredentialsForRemoteConnections. Käytetäänkö tunnuksia vai ei. Username. Käyttäjätunnus. Password. Salasana.
Frends.Directory.Clean	Directory. Hakemisto, josta tiedostoja halutaan poistaa. Pattern. Tiedostomaski, jonka avulla poistettavat tiedostot valitaan. UseGivenUserCredentialsForRemoteConnections. Käytetäänkö tunnuksia vai ei. Username. Käyttäjätunnus. Password. Salasana.
Frends.SFTP.UploadFiles	Directory (Source). Lähdehakemisto, josta tiedostoja haetaan siirrettäväksi SFTP-palvelimelle. FileName (Source). Tiedoston nimi tai tiedostomaski, jolla valitaan siirrettävät tiedostot. Directory (Destination). Kohdehakemisto, jonne löydetyt tiedostot siirretään. Address. SFTP-palvelimen osoite. Port. SFTP-palvelimen portti. Username. Käyttäjätunnus SFTP-pavelimelle. Password. Salasana SFTP-palvelimelle.
Frends.SFTP.DownloadFiles	Samat kuin Frends.SFTP.UploadFiles Task:lla, mutta Directory (Source) on SFTP-pavelimen hakemisto ja Directory (Destination) levyjaon hakemisto.

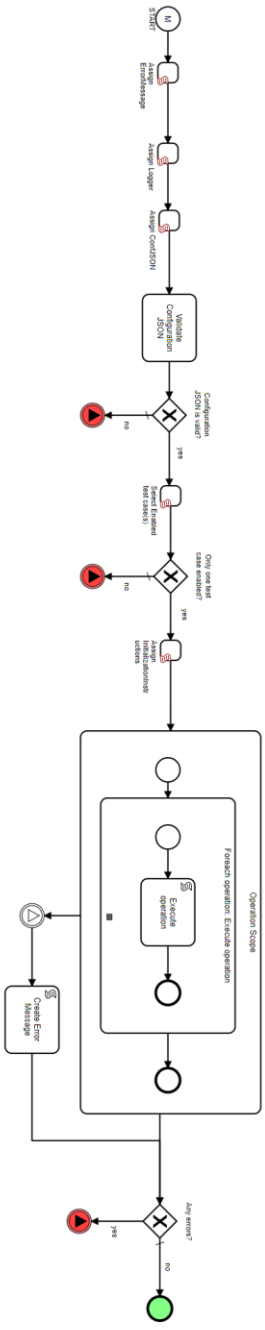
Alustusprosessin konfigurointi-JSON-rakenteen avainparien selvennykset

Avainparin nimi	Arvon tyyppi	Kuvaus	Esimerkki arvosta
TestCases	Array	Lista, joka pitää sisällään olioita, jotka esittävät testitapauksia	Katso kuva 12
Enabled	Boolean	Määrittää sen onko kyseessä oleva ”testitapaus” tarkoitus ajaa	true tai false
Name	String	Käyttäjän määrittämä nimi testitapaukselle	Lähteestä haetaan validi tiedosto ja se viedään kohteeseen onnistuneesti
Description	String	Käyttäjän määrittämä kuvaus testitapaukselle	Lähde- ja kohdehakemisto ovat tyhjiä, luodaan tiedosto lähteeseen ja prosessi noutaa tämän ja siirtää sen onnistuneesti kohteeseen.
TempWorkDirectory	String	Väliaikaisen hakemiston polku, jota käytetään työskentelyhakemistona tiettyjen operaatioiden kanssa	C:\\FRENDS\\Oppari\\Example\\Temp
InitializationInstructions	Array	Lista, joka sisältää olioita ja jokainen olio sisältää yhden operaation tiedot	Katso kuva 12
Operation	String	Operaatio, joka halutaan suorittaa	File Create, File Copy, File Move, Directory Create, Directory Clean tai Directory Delete
Directory	String	Hakemiston polku	C:\\FRENDS\\Oppari\\Example\\Source
SourceDirectory	String	Lähdehakemiston polku	C:\\FRENDS\\Oppari\\Example\\Source

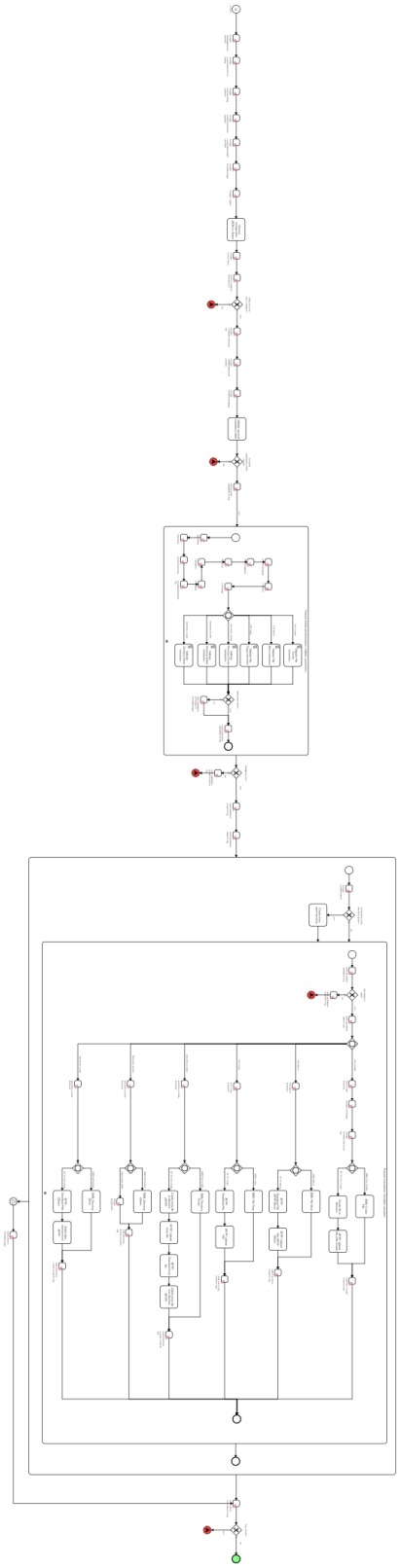
DestinationDirectory	String	Kohdehakemiston polku	C:\\FRENDS\\Oppari\\Example\\Destination
FileMask	String	Tiedostomaski, jonka avulla tiedostoja etsitään	Example*.xml
FileName	String	Tiedoston nimi	ExampleFilename.xml
Protocol	String	Protokolla, jota käytetään	SMB tai SFTP
ProtocolName	String	Käyttäjän määrittämä nimi protokollalle	SFTP 1
UseCredentials	Boolean	Määrittää käytetäänkö tunnuksia vai ei	true tai false
Content	String	Tiedoston sisältö	Tiedoston sisältö, joko puhtaana merkkijonona tai base64 koodattuna merkkijonona
Encoding	String	Tiedoston sisällön koodaus	Tiedoston sisällön koodaus esim. ASCII. Vakiona käytetään UTF8-koodausta
ProtocolSettings	Object	Protokolliin liittyvät yhteystiedot, kuten SFTP-palvelimen osoite ja tunnukset	Katso kuva 12
SFTP	Array	Lista, joka sisältää yhden tai useamman SFTP-palvelimelle tarvittavat tiedot	Katso kuva 12
Name	String	Käyttäjän määrittämä nimi, jolla tunnistetaan käytettävät tunnukset	SFTP 1
Address	String	SFTP-palvelimen osoite	example.address.com
Port	String	SFTP-palvelimen portti	22, vakiona käytetään arvoa 22
Username	String	Käyttäjätunnus	ExampleUsername

Password	String	Salasana	ExamplePassword
SMB	Array	Lista, joka sisältää yhden tai useamman SMB:lle tarvittavat tiedot	Katso kuva 12
Name	String	Käyttäjän määrittämä nimi	SMB Frends
Username	String	Käyttäjätunnus	ExampleUsername
Password	String	Salasana	ExamplePassword

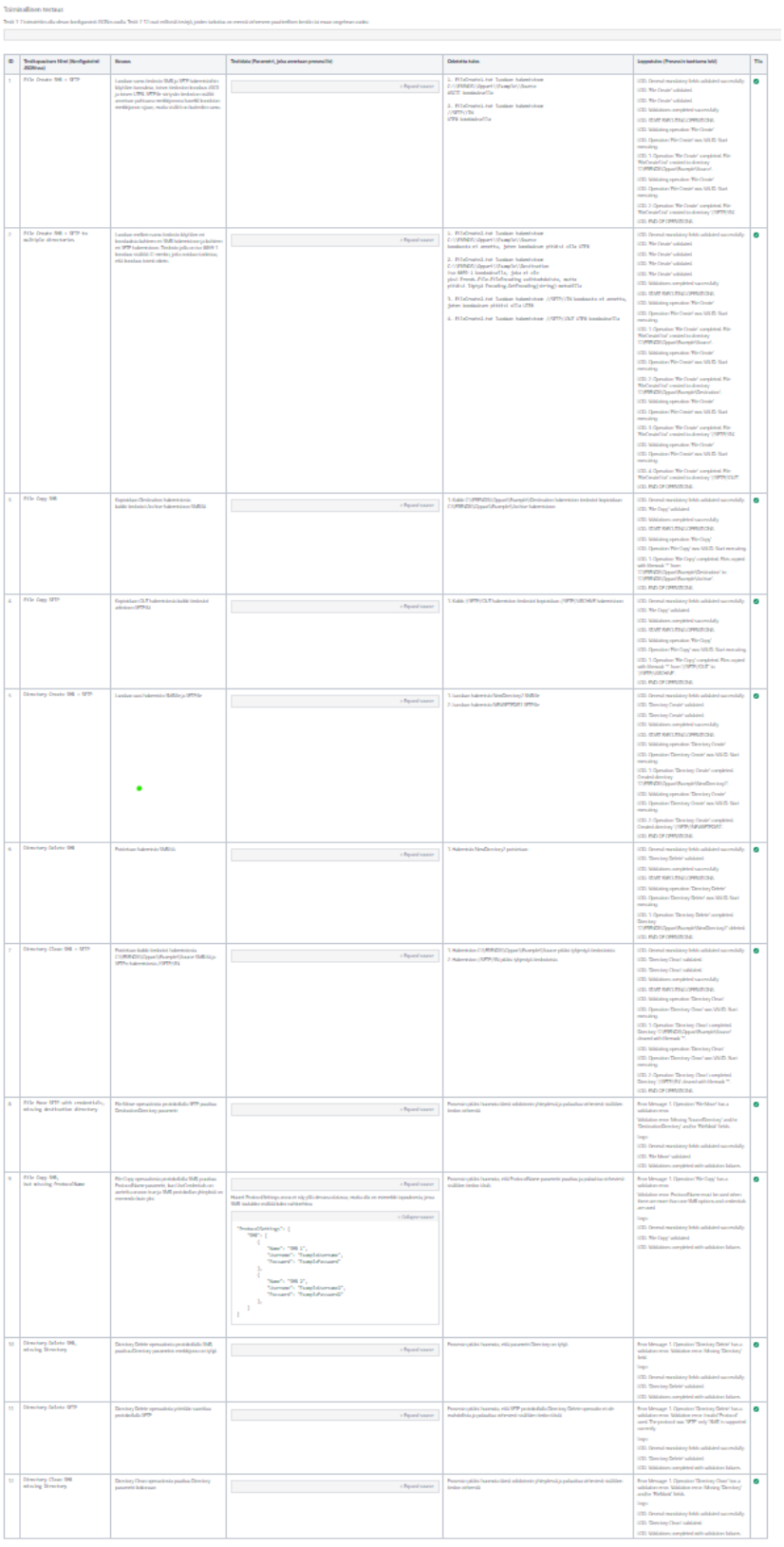
Alustusprosessin alustava toteutus BPMN-kaaviona



Alustusprosessin lopullinen versio



Alustusprosessin testitapauksien taulukko



Konfigurointi-JSON testitapauksille 1–7

```
{
    "TestCases": [
        {
            "Enabled": false,
            "Name": "File Create SMB + SFTP",
            "Description": "File Create SMB test",
            "TempWorkDirectory": "C:\\FRENDS\\Oppari\\Example\\Temp",
            "InitializationInstructions": [
                {
                    "Operation": "File Create",
                    "Directory": "C:\\FRENDS\\Oppari\\Example\\Source",
                    "FileName": "FileCreate1.txt",
                    "Protocol": "SMB",
                    "UseCredentials": true,
                    "Content": "V0VzdCBjb250ZW50IHRvIGZpbGUu",
                    "Encoding": "ASCII"
                },
                {

                    "Operation": "File Create",
                    "Directory": "//SFTP//IN",
                    "FileName": "FileCreate1.txt",
                    "Protocol": "SFTP",
                    "ProtocolName": "SFTP w password",
                    "UseCredentials": true,
                    "Content": "V0VzdCBjb250ZW50IHRvIGZpbGUu",
                    "Encoding": "UTF8"
                }
            ]
        },
        {
            "Enabled": false,
            "Name": "File Create SMB + SFTP to multiple directories",
            "Description": "File Create SMB test",
            "TempWorkDirectory": "C:\\FRENDS\\Oppari\\Example\\Temp",
            "InitializationInstructions": [
                {
                    "Operation": "File Create",
                    "Directory": "C:\\FRENDS\\Oppari\\Example\\Source",
                    "FileName": "FileCreate2.txt",
                    "Protocol": "SMB",
                    "UseCredentials": false,
                    "Content": "V0VzdCBjb250ZW50IHRvIGZpbGUu"
                },
                {
                    "Operation": "File Create",
                    "Directory": "C:\\FRENDS\\Oppari\\Example\\Destination",
                    "FileName": "FileCreate2.txt",
                    "Protocol": "SMB",
                    "UseCredentials": false,
                    "Content": "V0VzdCBjb250ZW50IGlzby04ODU5LTEgQ==",
                    "Encoding": "iso-8859-1"
                },
                {
                    "Operation": "File Create",
                    "Directory": "//SFTP//IN",
                    "FileName": "FileCreate2.txt",
                    "Protocol": "SFTP",
                    "ProtocolName": "SFTP w password",
                    "UseCredentials": true,
                    "Content": "V0VzdCBjb250ZW50IHRvIGZpbGUu"
                },
                {
                    "Operation": "File Create",
                    "Directory": "//SFTP//OUT",
                    "FileName": "FileCreate2.txt",
                    "Protocol": "SFTP",
                    "ProtocolName": "SFTP w password",
                    "UseCredentials": true,
                    "Content": "V0VzdCBjb250ZW50IHRvIGZpbGUu",
                    "Encoding": "UTF8"
                }
            ]
        },
        {
            "Enabled": false,
            "Name": "File Copy SMB",
            "Description": "Copy files from destination to archive",
            "InitializationInstructions": [
                {
                    "Operation": "File Copy",
                    "SourceDirectory": "C:\\FRENDS\\Oppari\\Example\\Destination",
                    "DestinationDirectory": "C:\\FRENDS\\Oppari\\Example\\Archive",
                    "FileMask": "*",
                    "Protocol": "SMB",
                    "UseCredentials": false
                }
            ]
        },
        {
            "Enabled": false,
            "Name": "File Copy SFTP",
            "Description": "Copy files from OUT to ARCHIVE",
            "TempWorkDirectory": "C:\\FRENDS\\Oppari\\Example\\Temp",
            "InitializationInstructions": [
                {
                    "Operation": "File Copy",
                    "SourceDirectory": "//SFTP//OUT",
                    "DestinationDirectory": "//SFTP//ARCHIVE",
                    "FileMask": "*",
                    "Protocol": "SFTP",
                    "ProtocolName": "SFTP w password",
                    "UseCredentials": true
                }
            ]
        },
        {
            "Enabled": false,
            "Name": "Directory Create SMB + SFTP",
            "Description": "Directory Create SMB + SFTP test",
            "TempWorkDirectory": "C:\\FRENDS\\Oppari\\Example\\Temp",
            "InitializationInstructions": [
                {
                    "Operation": "Directory Create",
                    "Directory": "C:\\FRENDS\\Oppari\\Example\\NewDirectory2",
                    "Protocol": "SMB",
                    "UseCredentials": false
                },
                {
                    "Operation": "Directory Create",
                    "Directory": "//SFTP//NEWSFTPDIR1",
                    "Protocol": "SFTP",
                    "ProtocolName": "SFTP w password",
                    "UseCredentials": true
                }
            ]
        },
        {
            "Enabled": false,
            "Name": "Directory Delete SMB",
            "Description": "Delete NewDirectory2",
            "InitializationInstructions": [
                {
                    "Operation": "Directory Delete",
                    "Directory": "C:\\FRENDS\\Oppari\\Example\\NewDirectory2",
                    "Protocol": "SMB",
                    "UseCredentials": false
                }
            ]
        },
        {
            "Enabled": false,
            "Name": "Directory Clean SMB + SFTP",
            "Description": "Cleanup directory test",
            "TempWorkDirectory": "C:\\FRENDS\\Oppari\\Example\\Temp",
            "InitializationInstructions": [
                {
                    "Operation": "Directory Clean",
                    "Directory": "C:\\FRENDS\\Oppari\\Example\\Source",
                    "FileMask": "*",
                    "Protocol": "SMB",
                    "UseCredentials": false
                },
                {
                    "Operation": "Directory Clean",
                    "Directory": "//SFTP//IN",
                    "FileMask": "*",
                    "Protocol": "SFTP",
                    "ProtocolName": "SFTP w password",
                    "UseCredentials": true
                }
            ]
        }
    ],
    "ProtocolSettings": {
        "SFTP": [
            {
                "Name": "SFTP w password",
                "Address": "#env.Tatu_Oppari.SFTP_Address",
                "Port": "#env.Tatu_Oppari.SFTP_Port",
                "Username": "#env.Tatu_Oppari.SFTP_Username",
                "Password": "#env.Tatu_Oppari.SFTP_Password"
            }
        ],
        "SMB": [
            {
                "Name": "Default Fileshare",
                "Username": "HIQ-PFJLGBL5\\Kraken_Frends_Tatu",
                "Password": "#env.Tatu_Oppari.SMB_Password"
            }
        ]
    }
}
```