LAPIN AMK
Lapland University of Applied Sciences

FRENDSIN INTEGRAATIOTESTAUKSEN AUTOMAATIO

Teemu Syväoja

Opinnäytetyö

Tieto- ja viestintätekniikka
Insinööri (AMK)

2024

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tieto- ja viestintätekniikka
Insinööri (AMK)

Tekijä	Teemu Syväoja	**Vuosi**	2024
Ohjaaja	Toni Westerlund		
Toimeksiantaja	Vaasan Sähkö Oy		
Työn nimi	Frendsin integraatiotestauksen automaatio		
Sivumäärä	34		

Opinnäytetyön tavoitteena oli selvittää integraatiotestauksen automatisoinnin soveltuvuutta Vaasan Sähkölle. Työn tarkoituksena oli kehittää tehokkaampi menetelmä integraatioiden testaamiseen keskittyen automatisoidun testausputken käyttöönottoon.

Opinnäytetyössä selvitettiin Robot Frameworkin ja Azure DevOpsin soveltuvuutta ja yhteensopivuutta integraatiotestauksen automatisoinnissa, kun käytetään Frends-integraatioalustaa. Teoriaosuudessa käsiteltiin integraatioiden merkitystä liiketoimintaprosesseissa ja testausautomaation roolia ohjelmistokehityksessä. Testausjärjestelmien käyttöönotossa kuvattiin asennusprosessit ja testien konfigurointi. Testausautomaation suunnittelussa keskityttiin testitapausten valintaan, jotka olivat olennaisia liiketoiminnan tarpeille.

Tuloksista kävi ilmi, että integraatiotestauksen automatisointi Robot Frameworkin ja Azure DevOpsin avulla sisältää paljon mahdollisuuksia ja voi edistää testausprosessien kattavuutta ja laatua. Työssä tunnistettiin myös haasteita ja kehityskohteita, kuten testidatan hallinnan, tulosten tarkastuksen ja testien modulaarisuuden merkitys. Opinnäytetyö tarjoaa suosituksia testausautomaation jatkokehittämiseksi ja osoittaa testausautomatisoinnin potentiaalin integraatioiden laadunvarmistuksessa.

Avainsanat: Azure DevOps, CI/CD, Frends, integraatiotestaus, Robot Framework, testausautomaatio

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Study Programme in Information
and Communication Technology
Bachelor of Engineering

Author	Teemu Syväoja	**Year**	2024
Supervisor	Toni Westerlund		
Commissioned by	Vaasan Sähkö Oy		
Title	Automated integration testing for Frends		
Number of pages	34		

The aim of this thesis study was to investigate the suitability of automating integration testing for Vaasa Sähkö. The purpose of the thesis was to develop a more efficient method for testing integrations, focusing on the implementation of an automated testing pipeline.

The thesis investigated the suitability and compatibility of Robot Framework and Azure DevOps for automating integration testing when using the Frends integration platform. The theoretical part covered the importance of integrations in business processes and the role of test automation in software development. The deployment of test systems was described, including installation processes and test configuration. Test automation design focused on the selection of test cases that were relevant to business needs.

The results showed that automating integration testing using Robot Framework and Azure DevOps has a lot of potential and can contribute to the coverage and quality of testing processes. The work also identified challenges and areas for development, such as the importance of test data management, results validation and test modularity. The thesis provides recommendations for further development of test automation and demonstrates the potential of test automation for quality assurance of integrations.

Keywords Azure DevOps, CI/CD, Frends, integration testing, Robot Framework, testing automation

SISÄLLYS

KÄYTETYT LYHENTEET JA TERMIT

AD	Active Directory
API	Application Programming Interface
BPMN	Business Process Model and Notation
CI/CD	Continuous Integration/Continuous Delivery
CRM	Customer Relationship Management
ERP	Enterprise Resource Planning
Fingrid Datahub	sähkön vähittäismarkkinoiden keskitetty tietojärjestelmä
Framework	ohjelmistokehys
GIT	versionhallintajärjestelmä
GSRN	Global Service Relation Number
iPaaS	infrastructure Platform as a Service
JSON	JavaScript Object Notation
Komentosarja	script
Low code	koodaamista vähentävä kehitysmalli
PAT	Personal Access Token
Pipeline	koodin automaattiprosessi rakennuksesta testaukseen
PoC	Proof of Concept
REST API	Representational State Transfer API
SaaS	Service as a Service
SAF	Standard ASCII Format
Sprint	ohjelmistokehityksen työjakso
SQL	Structured Query Language
TFVC	Team Foundation Version Control
XML	eXtensible Markup Language
YAML	YAML Ain't Markup Language

1 JOHDANTO

Tämä opinnäytetyö keskittyy kehittämään testausautomaatiota käyttäen Azure DevOpsin CI-putkea ja Robot Frameworkia Frends-integraatioalustan prosessien testaamiseen. Työssä tarkastellaan, miten automatisoitu testausputki voi parantaa integraatioprosessien kehittämistä, ylläpitoa ja laadunvarmistusta. Tavoitteena on osoittaa, miten hyvin suunniteltu ja toteutettu testausautomaatio voi ratkaista monimutkaisten integraatioprosessien testauksen haasteita ja lisätä tehokkuutta ja laatua.

Integraatiot ovat keskeisessä roolissa nykyaikaisissa liiketoimintaprosesseissa. Ne ovat välttämättömiä tiedonsiirrolle ja prosessien automatisoinnille, mahdollistaen eri järjestelmien, kuten toiminnanohjausjärjestelmän (*ERP*), asiakkuudenhallintajärjestelmän (*CRM*) ja muiden sovellusten välisen yhteentoimivuuden. (Snaplogic 2024.) Yrityksissä on kymmeniä, jopa satoja eri järjestelmiä, joiden halutaan vaihtavan tehokkaasti tietoja keskenään. Tyypillisesti yrityksissä ei ole yhtä suurta tietokantaa, vaan data on hajautettu useiden eri järjestelmien kesken ja järjestelmät eivät useinkaan kommunikoi suoraan keskenään ja tämän vuoksi tarvitaan kolmannen osapuolen järjestelmä, kuten Frends, toimimaan välikätenä. Integraatioalusta toimii tulkkina eri järjestelmien välillä muuntaen ja välittäen tietoja niin, että ne ovat sopivassa muodossa kohdejärjestelmälle. Tämän voi myös toteuttaa ilman erillistä integraatioalustaa, mutta se vaatisi merkittävää manuaalityötä ja koodausta. (Frends Technology 2024a.)

Testausautomaation ja automatisoitujen testausputkien merkitys ohjelmistokehityksessä on tunnustettu laajasti. Testausautomaation käyttöönotto lisää ohjelmistojen tehokkuutta ja varmistaa paremman ohjelmistolaadun, mikä on tärkeää nopeasti muuttuvassa ohjelmistokehityksessä. Automaattiset testit voivat suorittaa testitapauksia tehokkaasti ja auttavat todellisten ja odotettujen tulosten vertailussa, vähentäen toistuvaa ja virhealtista manuaalista testausta. Tämä lisää ohjelmistokehitystiimien tehokkuutta nopeuttamalla palautteen saantia ja mahdollistaen välittömän korjaamisen potentiaalisille ongelmille. (Cigniti 2024.)

Automaattisen testausputken käyttö tuo useita etuja, kuten nopeamman palautteen kierron, ajan säästön, vähentyneet liiketoimintakustannukset ja paremman testikattavuuden. Automaattitestaus mahdollistaa syvällisempien ja monimutkaisten testien suorittamisen, jotka usein jätetään tekemättä manuaalisissa testeissä. Testausautomaation avulla voidaan suorittaa testejä laajemmassa mittakaavassa ja nopeammin kuin manuaalinen testaus, mikä on erityisen hyödyllistä monimutkaisten käyttötapauksien testaamisessa ja suurten käyttäjämäärien simuloinnissa. Tämä nopeuttaa tuotteiden markkinoille saattamista ilman, että laatu kärsii. (Bose 2023.)

Automaattitestauksen käyttöönotto voi johtaa merkittäviin säästöihin, kun manuaalitestauksen tarve vähenee. Manuaalitestaus vaatii aina työtunteja, ja automaattitestauksen kehitystyö voi maksaa itsensä nopeasti takaisin jo pelkän testaukseen käytettyjen työtuntien vähentyessä. Tämä tekee automaattitestauksesta pitkällä aikavälillä kustannustehokkaan vaihtoehdon, erityisesti jatkuvasti päivittyvissä ohjelmistoprojekteissa, joissa muutokset ovat tiheässä. (Patel 2023.)

Opinnäytetyön tavoitteena on ymmärtää ja kehittää tehokkaampi menetelmä integraatioiden testaamiseen keskittyen automatisoidun testausputken käyttöönottoon. Tavoitteen saavuttamisen merkitys korostuu omassa työtehtävässäni, jossa integraatioiden kehittäminen ja ylläpito ovat keskeisessä asemassa. Tämän tavoitteen saavuttaminen edesauttaa prosessien sujuvuutta, virheenhallintaa ja tehostaa liiketoimintaprosesseja.

Tarkastelun kohteeksi otettiin Azure DevOpsin CI-putken ja Robot Frameworkin yhdistäminen ja niiden soveltuvuus integraatiotestaukseen. Tämän yhdistelmän avulla pyrittiin luomaan kattava testausympäristö, joka integroitiin Frends-alustaan parantaen näin integraatioiden kehittämistä.

Testausstrategioiden ja -menetelmien kehittäminen on toinen keskeinen osa työtä. Pyrkimyksenä on kehittää sopivia menetelmiä integraatioalustan tarpeisiin ja analysoida Robot Frameworkin kykyä käsitellä näitä monimutkaisia skenaarioita. Lisäksi arvioidaan testausputken tehokkuutta ja vaikutuksia integraatioprosessien laatuun ja suorituskykyyn tarjoten samalla suosituksia testausautomaation jatkokehittämiseksi.

Opinnäytetyö keskittyy testausprosessin kehittämiseen ja erityisesti automatisoinnin hyödyntämiseen ja testausputken (*testing pipeline*) luomiseen välttäen integraatioiden teknisen toteutuksen syvällistä käsittelyä. Tässä yhteydessä käytetään esimerkkeinä olemassa olevia integraatioita, jotka tarjoavat konkreettisia tapausesimerkkejä ja sovelluksia. Tämä lähestymistapa mahdollistaa keskittymisen testausautomaation metodologiaan ja sen soveltamiseen käytännön integraatioympäristöissä. Tämä antaa selkeän näkökulman testausprosessien tehostamiseen ja parantamiseen.

Opinnäytetyön toimeksiantajana toimii Vaasan Sähkö, joka on osa laajempaa Vaasan Sähkö -konsernia. Vaasan Sähkö on perustettu vuonna 1892 ja se on yksi Suomen vanhimmista yhtiöistä energiatoimialalla. Vaasan kaupunki omistaa yhtiön osakekannasta 99,9 prosenttia. Vuonna 2022 koko konsernin henkilöstömäärä oli 136 henkilöä ja se jakautuu kolmeen päätoimialaan. Vaasan Sähkö keskittyy sähkön myyntiin, ja sillä oli 113 459 asiakasta vuonna 2019. (Vaasan Sähkö 2024a.) Kaukolämpö vastaa kaukolämmön siirrosta ja myynnistä, ja vuonna 2021 sillä oli 3 463 asiakasta. Tytäryhtiö Vaasan Sähköverkko puolestaan vastaa alueen sähkönsiirrosta, ja vuonna 2022 sillä oli 74 825 asiakasta. (Vaasan Sähkö 2024b.)

Opinnäytetyön tuloksena odotetiin kokemuksia testausautomaation käyttöönotosta ja sen käytännön hyödyistä integraatioiden testauksessa. Parasta ratkaisua integraatioiden testaukseen ei välttämättä ole olemassa, mutta Robot Frameworkin odotettiin soveltuvan siihen hyvin ja sen käyttöönotosta saatujen kokemusten auttavan arvioimaan myös muita testausratkaisuja.

Opinnäytetyössä olen hyödyntänyt omaa kokemustani integraatioiden parissa, joka on kerryttänyt minulle hiljaista tietoa alalta. Tämän tiedon avulla olen pyrkinyt tunnistamaan keskeisiä haasteita ja mahdollisuuksia integraatiotestauksen automatisoinnissa.

2 OPINNÄYTETYÖSSÄ KÄYTETYT TEKNOLOGIAT

2.1 Frends-integraatioalusta

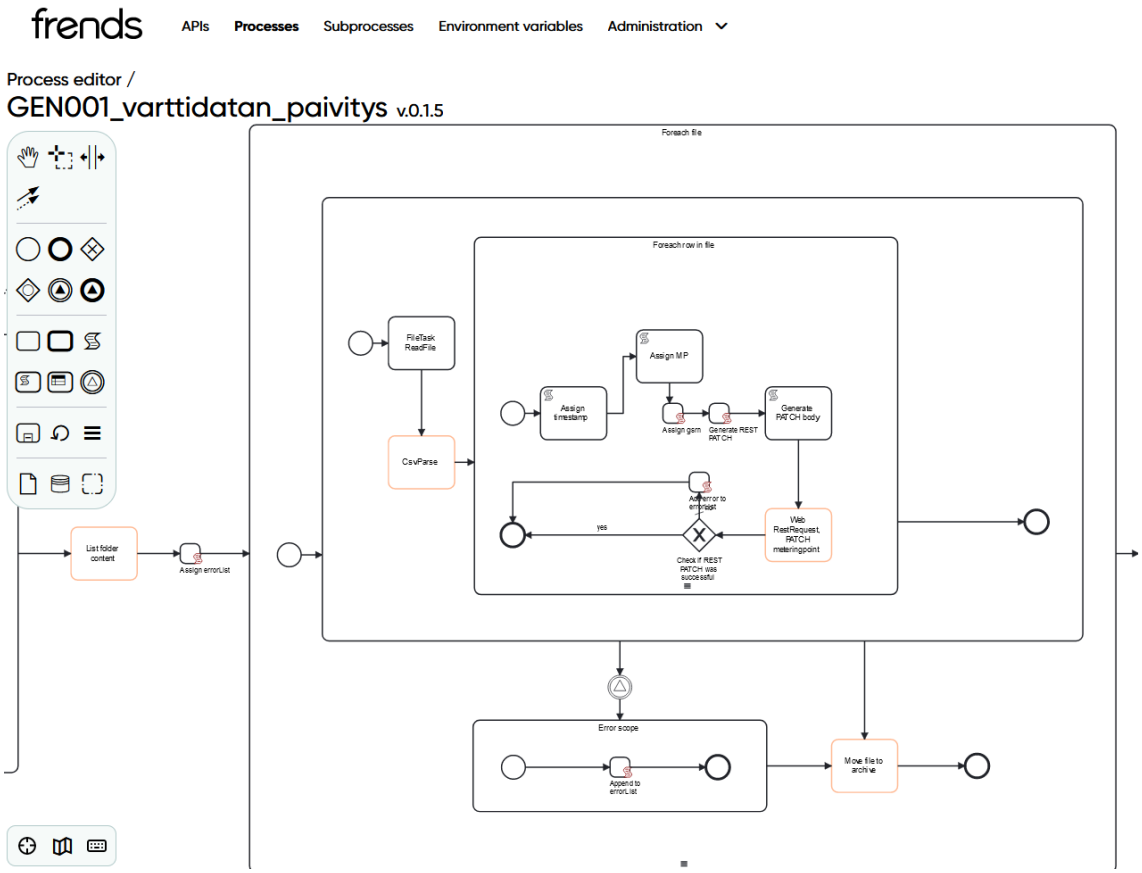
Frendsin alunperin kehittänyt yritys perustettiin 1987 (Yritys- ja yhteisötietojärjestelmä 2024) ja sen alkuperäisenä käyttötarkoituksena oli yhdistää kaikki Suomen huoltoasemien polttoainepumput käyttäen modeemeja ja vaihtaa tietoa asemien ja keskustietokoneen välillä. Integraatioalustan nimi juontuu sen alkuperäisestä käyttötarkoituksesta, Front End Dialing System. HIQ Finland Oy osti yrityksen 2014, ja Frends on nykyään erotettu omaksi liiketoiminta-alueeksi. Frendsillä on yli 200 työntekijää 16:ssa eri maassa. (Frends Technology 2024b.)

Frends on iPaaS eli integration Platform as a Service (Frends Technology 2024a), joka tukee kaikenlaisia integraatioita eri järjestelmien välillä. Käyttöliittymä on web-sovelluspohjainen, ja palvelu toimii Windows-palvelimella tai Azure-pilvessä. Alustan tärkeimpiin ominaisuuksiin lukeutuvat seuraavat:

- tiedostojen siirto (*Managed File Transfers, MFT*)
- tietokantakyselyt ja muutokset eri kantatyyppeihin
- API-rajapintojen käyttö sekä rajapintojen isännöinti
- ohjelmistorobotiikan ohjaus (*Robotic Process Automation, RPA*) osana laajempia automatisointiprosesseja ja
- tietomuotojen käsittely (*Data Mapping ja Transformation*) kuten datan formaatin muuntaminen ja tietojen yhdistäminen. (Frends Technology 2024c.)

Frendsin integraatioita kutsutaan prosesseiksi, ja ne rakennetaan low code-tyylisellä käyttöliittymällä, joka käyttää BPMN 2.0-notaatiota (kuvio 1). BPMN eli Business Process Model and Notation on graafinen esitystapa liiketoimintojen, teknisten toimintojen ja datamallinnuksen toimintalogiikan kuvaamiseen muistuttaen vuokaaviota. Se käyttää eri symboleja variaatioineen toimintalogiikan kuvaamiseen esimerkiksi:

- tapahtuma (*event*): ympyrä
- aktiviteetti (*activity*): suorakulma
- portti (*gateway*): neljäkäs ja
- yhteydet (*connections*): nuoli. (Trisotech 2024.)



Kuvio 1. Frends-käyttöliittymä

Prosessit muodostetaan tehtävistä (*task*), joita kuvataan suorakulmioina. Ne voivat kulkea lineaarisesti tai haaroittua eri ehtojen mukaisesti. Tehtävät on toteutettu C#-ohjelmointikielen funktiona, ja ne vastaanottavat parametreja ja palauttavat ajon jälkeen C#-objektin. Prosessissa on mahdollista käyttää myös puhdasta C#-koodia kirjastoineen ja luoda aliprosesseja helpottamaan tehtävien järjestelyä. Prosessit käännetään tallennettaessa Frends-alustalla ajettaviksi ohjelmiksi, joille voi antaa ajettaessa lähtöparametreja. Prosessien ajamiselle (*run*) tai liipaisulle (*trigger*) on useita vaihtoehtoja, muun muassa käsinajo

(*manual*), kansion tarkkailu (*folder polling*) ja HTTP metodit (*HTTP method*). (Frends Technology 2024e.)

Prosesseja hallitaan prosessilistalla, josta ne otetaan käyttöön (*deploy*) halutulla agentilla, joka voi olla omassa konesalissa (*on premise*) tai Frendsin pilvessä (*cloud*). Agentit on jaettu kehitys- (*development*), testaus- (*test*) ja tuotanto (*production*) -osioihin. Agentti tarkoittaa tässä yhteydessä Frends-alustan palvelua (*service*), jota ajetaan Windows-palvelimella tai pilvessä. Prosessilistalla prosesseja voi ajaa käsin, aktivoida (*activate*) tai deaktivoida (*deactivate*) ajastuksia, rajapintoja tai liipaisimia. Frends hoitaa versionhallinnan, jossa low code-lähdekoodi on tallennettu BPMN-notaatiolla XML-muodossa, ja kääntää (*compile*) prosessin ajettavaan muotoon automaattisesti prosessia tallennettaessa. (Frends Technology 2024e.)

Frends-integraatioalusta tarjoaa tuen hyperautomaatiolle mahdollistaen yhteistyön useiden RPA-ohjelmistojen kanssa. Tämä kattaa suositut RPA-alustat, kuten UiPath, BluePrism ja PowerAutomate, jotka mahdollistavat laajat automatisointiratkaisut. (Frends Technology 2024a.)

Yleisimmät haasteet prosessien testauksessa johtuvat testattavien asioiden suuresta määrästä ja testaajien rajallisesta ajasta (Unadkat 2022). Manuaalitestauksessa kun prosesseja ajetaan käsin testitiedostoja luodaan kansioihin tai järjestelmien tietoihin tehdään muutoksia prosessien liipaisemiseksi ei riitä, että prosessin ajo ei ole tuottanut virheitä. Testien lopputulokset täytyy myös tarkistaa. Frendsin luoma loki (*log*) on oletuksena melko suppea, ja toistuvien ajojen lopputuloksen seuranta on siitä työlästä.

Testausvaihtoehtoja on yksinkertaisessakin integraatiossa yleensä vähintään kymmeniä, ja monesti manuaalitestauksessa kokeillaan muutamia eri testiparametreja ja katsotaan sitten sen toiminta testiympäristössä. Tämä on virhealtista ja kattaa vain pienen osan testaustarpeesta. Automaattitestaus esimerkiksi poistaa testitulosten tulkinnasta johtuvat virheet, kun pitää vertailla useita arvoja, varmistaa että testit suoritetaan oikeassa järjestyksessä ja eliminoi inhimilliset virheet, kuten väsymyksestä tai keskittymisen herpaantumisesta johtuvat erehdykset. (Testsigma 2023.)

Automaattitestauksella pyritään ratkaisemaan edellä mainitut ongelmat ja varmistamaan testausprosessin tehokkuus, kattavuus ja toistettavuus. Automaattitestit pysyvät aina samanlaisina, ne ajetaan aina muutosten yhteydessä ja testit kertovat missä prosessi on tehnyt virheen. Tämä vähentää inhimillisiä erehdysmahdollisuuksia. (Bose 2023.)

2.2 Robot Framework

Robot Framework on vuonna 2008 julkaistu avoimen lähdekoodin automaatiotestauskehys, joka on kehitetty helpottamaan hyväksymistestauksen automatisointia. Robot Frameworkia kehittää Robot Framework Foundation, ja se on ohjelmoitu Python-ohjelmointikielellä. (Robot Framework 2024a.)

Robot Framework on suunniteltu monikäyttöiseksi mahdollistaen erilaiset testaus- ja automaatiotarpeet, ja sen testit luodaan avainasanoja (*keyword*) käyttäen. Avainsanapohjainen syntaksi on suunniteltu olemaan helppolukuinen ja -kirjoitettava. Tämä tekee testien ymmärtämisestä ja luomisesta helpompaa myös niille, joilla ei ole laajaa ohjelmointikokemusta. (Robot Framework 2024b.)

Robot Framework tarjoaa kattavia raportointi- ja lokitusominaisuuksia, jotka tuottavat yksityiskohtaisia raportteja ja lokeja testien suorituksista helpottaen näin testitulosten analysointia ja virheiden jäljittämistä. Testien suorittaminen on mahdollista nimettyjen avainsanojen avulla, mikä sallii testitapausten dynaamisen valinnan eri kriteerien perusteella. (Robot Framework 2024b.)

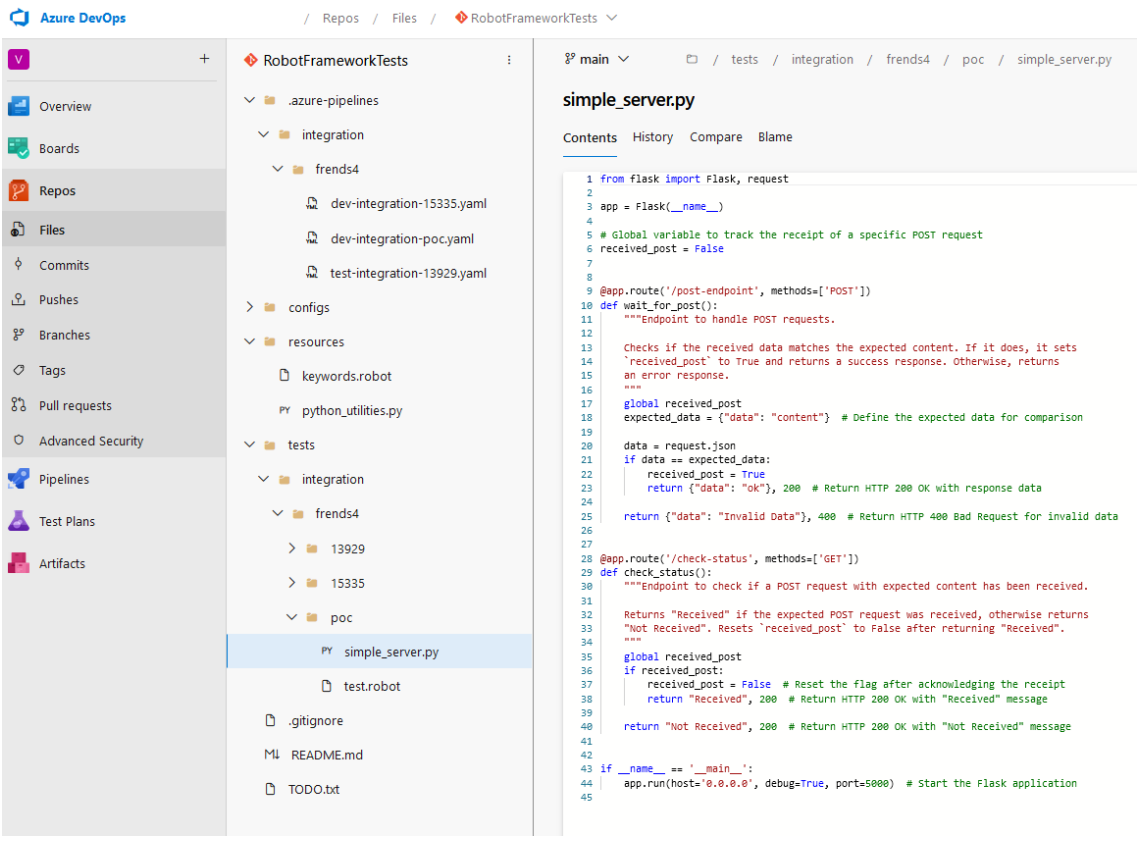
Työkalu integroituu saumattomasti muihin työkaluihin ja alustoihin, kuten versionhallintajärjestelmiin, jatkuvan integroinnin (*CI*) työkaluihin ja muihin testausvälineisiin (Novovic 2019). Robot Framework mahdollistaa moduulien ja testien uudelleenkäytön eri projekteissa, mikä vähentää tarpeetonta toistoa. Se tukee Python-kirjastoja tarjoten käyttäjille laajat mahdollisuudet laajennukseen ja mukauttamiseen (Robot Framework 2024b). Vaikka Robot Framework soveltuu monipuolisesti erilaisiin testaustarpeisiin, se edellyttää käyttäjältä ainakin perustason osaamista ohjelmoinnista.

Robot Framework mukautui alustavien testien perusteella käytännössä kaikkeen integraatiotestaukseen, mitä toimeksiantaja tarvitsee. Sen joustavuus ja

monipuolisuus mahdollistaa monenlaisten testiskenaarioiden toteuttamisen. Robot Framework on ilmainen ja helppo ottaa käyttöön, mikä tekee siitä kustannustehokkaan vaihtoehdon (Robot Framework 2024b). Sen yhteensopivuus Python-kirjastojen kanssa on suuri etu, sillä Pythonin monipuoliset kirjastot ovat valmiiksi tuttuja työtehtävien kautta. Robot Framework integroituu hyvin Azure DevOpsin kanssa. DevOpsilla voi hallita testien ajoja ja liipaisuja sekä seurata Robot Frameworkin toimintaa sen luomista lokitiedostoista. (Novovic 2019.)

2.3 Azure Devops

Azure DevOps on Microsoftin tarjoama SaaS-palvelu, joka sisältää laajan valikoiman työkaluja ohjelmistokehityksen tukemiseen. Tämä sisältää versionhallinnan (kuvio 2), projektinhallinnan sekä CI/CD-prosessien toteutuksen. Palvelu integroituu suoraan Microsoftin palveluihin ja sisältää laajan valikoiman yhteyksiä kolmannen osapuolen ohjelmistoihin ja rajapintoihin, mikä tekee siitä monipuolisen työkalun erilaisiin kehitystarpeisiin. DevOps tukee useita käyttöjärjestelmiä, ja sen itse isännöityjä agentteja (*self-hosted agent*) voi ajaa Windows-, Linux- ja MacOs-palvelimilla. Pipelinet voidaan myös määritellä ajamaan koodi eri käyttöjärjestelmillä omissa konteissaan (*container*). (Microsoft 2024.)



Kuvio 2. Azure Devops-versionhallinnan käyttöliittymä

Azure DevOpsin eri moduulit tukevat yhteistyötä kehittäjien, testaajien ja IT-toimintojen välillä. Azure Repos tarjoaa versionhallintaa Gitin ja Team Foundation Version Controlin avulla, Azure Pipelines kääntää koodit ohjelmiksi ja ajaa yksikkötestejä, Azure Artifacts helpottaa sovellusten riippuvuuksien hallintaa, ja Azure Test Plans tarjoaa kattavat työkalut sovellusten manuaalitestaamiseen. Azure DevOpsin käytäntöjä ovat muun muassa jatkuva integraatio ja jatkuva toimitus (*CI/CD*), versionhallinta, ketterä ohjelmistokehitys, infrastruktuurina koodi, kokoonpanonhallinta ja jatkuva valvonta. Nämä käytännöt tukevat nopeampaa julkaisusykliä ja korkealaatuisempaa koodia vähentäen samalla julkaisuriskiä. (Microsoft 2024.)

Azure DevOps on monipuolinen työkalu, joka koostuu useista keskeisistä moduuleista, jotka tekevät siitä tehokkaan ohjelmistokehityksen hallintatyökalun. Kukin moduuli on suunniteltu palvelemaan tiettyä tarvetta ohjelmistokehityksen elinkaaren aikana.

Azure Repos tarjoaa versionhallintapalvelut, jotka tukevat Gitin ja TFVC:n käyttöä. Tämä mahdollistaa koodimuutosten seuraamisen ja hallinnan. (Microsoft 2024.)

Azure Boards on työkalu projektinhallintaan joka mahdollistaa tehtävien, bugien ja työnkulun hallinnan. Se tarjoaa visuaalisia työkaluja, kuten Kanban-tauluja ja sprinttisuunnittelua. (Microsoft 2024.)

Azure Pipelines on CI/CD-palvelu, joka tukee jatkuvaa integraatiota ja jatkuvaa toimitusta. Se mahdollistaa automaattisen koodin rakentamisen, testaamisen ja julkaisemisen. Pipelinelle ei ole vakiintunutta suomenkielistä termiä, ja sen käyttö ei rajoitu pelkästään testaukseen, mutta koska tässä työssä on kyse testauksesta, voi termin kääntää testausputkeksi. (Microsoft 2024.)

Azure Test Plans on työkalu, joka tarjoaa testausten suunnittelun ja hallinnan toiminnot. Työkalu mahdollistaa kattavat manuaaliset ja automatisoidut testausprosessit. (Microsoft 2024.)

Azure Artifacts mahdollistaa pakettien luomisen, jakamisen ja hallinnan. Tämä sisältää NuGet-, npm- ja Maven-pakettienhalllinnat sekä muiden universaalien pakettien tuen. (Microsoft 2024.)

Azure DevOpsin perusperiaatteet ja prosessit, erityisesti jatkuva integraatio ja jatkuva toimitus ovat keskeisiä sen toiminnassa. CI-putki käynnistyy automaattisesti, kun muutoksia tehdään Azure Repos -versionhallinnassa ja se suorittaa määritellyt testit. CD-putki aktivoituu, kun CI-putki on onnistuneesti luonut ja julkaissut paketin. Jos hyväksymistestit epäonnistuvat, CI/CD-putken ajo päättyy ja kehittäjän on tehtävä tarvittavat korjaukset. Onnistuneen testauksen jälkeen voidaan toteuttaa manuaalinen tarkistus ennen tuotantoon siirtymistä. CD-putki vastaa sitten ratkaisun siirtämisestä tuotantoympäristöön ja seuraa sen toimintaa. (Microsoft 2024.)

2.4 Parhaat käytännöt

Parhaiden käytäntöjen (*best practices*) noudattaminen on keskeistä järjestelmällisen, tehokkaan ja virheettömän ohjelmistokehityksen ja -testauksen

edistämisessä. Systemaattiset lähestymistavat testien kirjoittamiseen, kuten testausautomaatio ja CI/CD, ovat tärkeitä, sillä ne edistävät nopeampaa kehitystyötä, vähentävät sovellusvirheitä ja parantavat tiimien yhteistyötä. Ero ad hoc-menetelmiin verrattuna on merkittävä, sillä best practices-lähestymistavat tarjoavat vakaammat ja ennustettavammat kehitysprosessit ja auttavat ylläpitämään projektin skaalautuvuutta ja ylläpidettävyyttä, erityisesti monimutkaisissa ja jatkuvasti muuttuvissa ympäristöissä. Koodin selkeys ja modulaarisuus ovat avainasemassa skaalautuvuuden edistämisessä. (Chowdhury 2024.)

Dokumentaatio ja selkeät käytännöt, kuten yhtenäiset nimikäytännöt ja testausstandardit auttavat koko tiimiä ja edistävät yhteisesti sovittujen käytäntöjen noudattamista. Parhaiden käytäntöjen seuraaminen auttaa minimoimaan riskejä ja parantamaan laadunvarmistusta. Systemaattiset ja standardisoidut lähestymistavat laadunvarmistusprosesseissa edistävät luotettavuutta ja johdonmukaisuutta. Lisäksi hyvät käytännöt helpottavat uusien tiimien jäsenten perehdyttämistä tarjoten yhteisen lähtökohdan ymmärrykselle projekteista ja prosesseista. (Chowdhury 2024.)

Projektissa noudatettiin parhaita käytäntöjä, jotka on sovellettu Robot Frameworkin, Pythonin ja Azure DevOpsin yleisistä käytännöistä. Robot Frameworkilla ei ole aina tarkasti määriteltyjä käytäntöjä, joten niiden soveltaminen ja mukauttaminen on tehty käyttäen muiden parhaita käytäntöjä.

Testien selkeyden ja ylläpidettävyyden kannalta **tiedostojen ja kansioiden nimeäminen** on erityisen tärkeää. Nimeämiskäytännöt perustuivat Frends-integraatioiden nimeämiseen, ja niitä sovellettiin yhtenäisesti sekä Robot Frameworkissa että Azure DevOpsissa. Kansiorakenteessa sovellettiin Robot Frameworkin yleisesti käytettyä rakennetta ja se on jaoteltu selkeästi eri osa-alueisiin, kuten tests, resources, results ja configs, mikä helpottaa projektin hallintaa ja organisointia. Yhtenäinen lähestymistapa nimeämisessä auttaa säilyttämään johdonmukaisuuden eri alustojen välillä.

Versionhallinta on olennainen osa ohjelmistokehitystä, ja tässä projektissa se oli kiinteästi kytköksissä testien ajamiseen. Testien ajaminen ja päivittäminen edellyttävät versionhallinnan aktiivista käyttöä.

Projektin pääpaino oli Robot Frameworkissa, joka itsessään ei tarjoa tarkkoja ohjeita **muuttujien nimeämiskäytäntöihin**. Robot Frameworkin ohjelmointia toteutetaan Pythonilla, ja siksi projektissa käytettiin Pythonin suositeltua snake case-muotoa muuttujien nimeämisessä. Snake case on muuttujien nimeämiskäytäntö, jossa yksittäisten sanojen välillä käytetään alaviivaa (_) ja kaikki kirjaimet kirjoitetaan pieninä kirjaimina, esimerkiksi ”host_address”.

Työssä käytettiin Azure Key Vaultia **salasanojen ja rajapinta-avainten säilyttämiseen**. Tämä menetelmä varmisti, että arkaluontoiset tiedot ovat suojattuja ja hallinnassa ja ne voitiin hakea dynaamisesti testiajojen yhteydessä.

Testien modulaarisuus oli projektissa keskeinen tekijä tehokkaan ja skaalautuvan testausarkkitehtuurin luomisessa. Robot Frameworkissa tämä saavutettiin hyödyntämällä avainsanat, jotka ovat sen funktioita ja joita voidaan luoda lisää tarpeen mukaan. Tämä mahdollisti testikomponenttien uudelleenkäytön vähentäen koodin toistoa ja parantaen testien ylläpidettävyyttä.

DevOps tarjoaa selkeän näkymän **testien suorituksesta ja tuloksista**, mikä auttaa hahmottamaan, kuinka testit onnistuvat. Tämän projektin päämääränä on kuitenkin ensisijaisesti tukea kehitystyötä ja integroida testit suoraan integraatioiden kehitystyöhön, ei suorittaa testejä rutiininomaisesti testi- tai tuotantoympäristössä. Tämän vuoksi seurannan rooli projektissa oli rajattu.

Hyvä koodi on selkeää ja itsedokumentoivaa, mutta **kommentointi** tekee koodin monimutkaisemmat toimintalogiikat ja kontekstin ymmärrettävämmäksi. Kommentit koodissa auttavat kehittäjiä nopeasti hahmottamaan koodin toiminnallisuuden ja tarkoituksen, mikä nopeuttaa merkittävästi ongelmanratkaisua ja virheenjäljitystä.

Testien **ylläpito** on jatkuva prosessi. Ihanteellisesti testit kirjoitetaan ennen integraation luomista, mutta käytännössä tämä ei aina onnistu ja testit edellyttävät ajoittaista päivitystä, erityisesti kun integraatioihin tehdään suurempia muutoksia. Tämä saattaa vaatia testien uudelleenkirjoittamista tai merkittäviä muutoksia niiden rakenteeseen. Lisäksi kun integraatio on otettu käyttöön tuotannossa, on uusien testausideoiden lisääminen myöhemmässä vaiheessa yleinen tarve.

3 INTEGRAATIOJÄRJESTELMÄN TESTAUS

3.1 Testausjärjestelmien käyttöönotto

Uuden Frends-palvelimen käyttöönotto on hyvin suoraviivaista. Ensin luotiin uusi virtuaalipalvelin palvelinympäristöön, jossa käyttöjärjestelmänä toimii Windows Server. Sitten Frendsin käyttöliittymässä luotiin uusi agentti ja annettiin sille aiemmin määritellyt AD-tunnukset. Seuraavaksi tarjolle tuli valmiiksi konfiguroitu asennuspaketti, joka asennettiin uudelle palvelimelle. Asennusprosessin päätyttyä uuden palvelimen tila päivittyi agenttilistalla ja se oli valmis käyttöön.

Robot Frameworkin käyttöympäristöksi valikoitui Windows-palvelin, sillä se tarjoaa sujuvan yhteensopivuuden Frendsin kanssa, joka myös käyttää Windows-palvelinta. Windows-palvelimen käyttö tarjoaa helpon ja kätevän tavan jakaa verkkolevyjä hyödyntäen Windowsin AD-tunnistusta. Robot Frameworkin uusimman vakaan (*stable*) version käyttö edellyttää vähintään Python-versiota 3.8. Koska Robot Framework on kehitetty Pythonilla, joka on tulkattava ohjelmointikieli, ovat sen ympäristövaatimukset yhdenmukaiset Pythonin kanssa.

Palvelimelle asennettiin Python-versio 3.12.1, joka oli asennushetkellä uusin vakaa versio. Python asennettiin kaikille käyttäjille, jotta DevOpsin agentin käyttämä AD-tunnus voi ajaa sitä. Robot Framework asennettiin Pythonin PIP-paketinhallintaohjelmalla ja samalla asennettiin soveltuvuusselvitystä (*PoC*) varten tarvittavat kirjastot robotframework-requests, flask ja urllib3.

Azure DevOps-agentti on keskeinen osa testausautomaatiota, sillä se mahdollistaa testien suorittamisen toimeksiantajan omalla palvelimella. Agentti toimii välikappaleena Azure DevOps-pilvipalvelun ja toimeksiantajan paikallisen palvelinympäristön välillä. Tämä on olennaista, koska testattaviin integraatioihin pääsee käsiksi vain toimeksiantajan sisäverkosta. Agentti toimii Windowsin palveluna (*service*), ja sillä on oikeus suorittaa ohjelmia paikallisesti.

DevOps-agentin asennusprosessi alkoi asennuspaketin lataamisella DevOpsin agenttiryhmä-osiosta (*agent pool*). Tässä tapauksessa prosessi aloitettiin luomalla uusi agenttiryhmä, minkä jälkeen sinne luotiin uusi agentti. Uuden agentin luomisen yhteydessä tarjotaan mahdollisuus ladata asennuspaketti sekä

ohjeet sen asentamiseen. Asennuspaketti sisältää kaikki tarvittavat tiedostot ja komentosarjat, joiden avulla agentti saadaan pystyyn ja toimintakuntoon.

Asennus aloitettiin purkamalla asennuspaketti paikalliselle levylle ja ajamalla siellä asennuskomennot. Ohjelma pyysi tarvittavat tiedot agentin peruskonfigurointiin ja muodosti yhteyden DevOpsiin. Asennuksen yhteydessä syötettiin henkilökohtainen käyttöoikeustunnus (*Personal Access Token, PAT*), joka luotiin DevOpsissa omalla käyttäjätunnuksella. PAT-tunnus varmistaa, että agentilla on oikeudet kommunikoida Azure DevOps -palvelun kanssa ja suorittaa tehtäviä. Lisäksi agentille oli luotu erillinen AD-käyttäjätunnus, jotta sille voitiin antaa tarvittavat oikeudet toimeksiantajan palvelinympäristöön ja verkkolevyille.

DevOpsiin luotiin uusi versionhallintavarasto (*repository*) automaattitestejä varten. Tämän versionhallintavaraston luominen oli välttämätöntä uuden testausputken lisäämiseksi, sillä testausputki vaatii versionhallintavaraston olemassaolon sen luomisen yhteydessä. Tässä testausputkessa on määritelty testien suoritus, joka toteutettiin paikallisen agentin avulla toimeksiantajan konesalissa. Testausputkien tarkempi määrittely ja sen eri vaiheet käsitellään yksityiskohtaisemmin myöhemmässä osiossa.

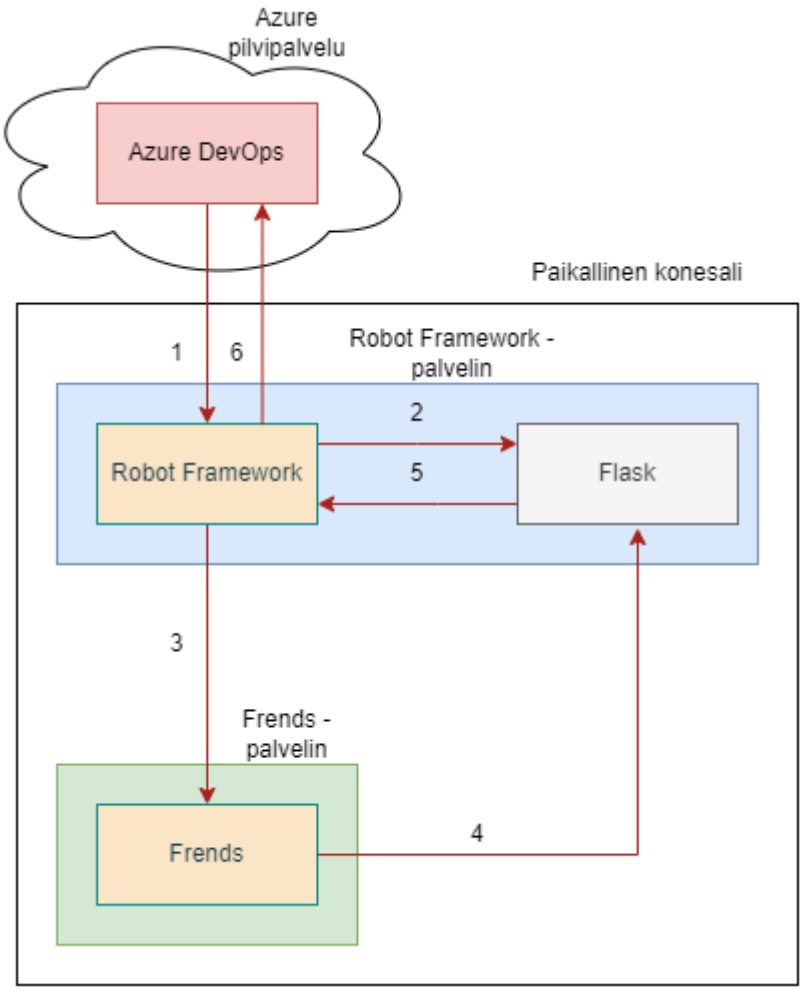
3.2 Soveltuvuusselvitys

Soveltuvuusselvitysvaiheessa kehitetettiin minimaalinen toimivuuden todistus, jonka tavoitteena oli esittää eri järjestelmien välistä kommunikaatiota. Tämän vaiheen aikana varmistettiin, että Robot Framework voi muodostaa yhteyden Frends-palvelimeen ja päinvastoin sekä että Azure DevOpsin testausputki kykenee ajamaan Robot Frameworkin testejä ja keräämään testiraportit. Soveltuvuusselvitys luotiin seuraavasti.

DevOps Pipeline-prosessi käynnistää testin ja Robot Framework pystyttää Flask-palvelimen odottamaan HTTP-kutsua. Testi kutsuu Frends-palvelimen HTTP-rajapintaa ja kun Frends on vastannut "ok", testi tarkistaa Flask-palvelimelta onko Frends lähettänyt vastakutsun ja onko sisältö odotetunlainen. Frendsin DEV-ympäristössä pyörii integraatioprosessi, joka odottaa HTTP-kutsua ja on suojattu API-avaimella, joka noudetaan Azure Key Vaultista. Robot Frameworkin pystyttämä Flask-palvelin vastaanottaa Frends-prosessin HTTP-kutsun, minkä

jälkeen Robot Framework tarkistaa, vastaako saatu vastaus odotuksia. Lopuksi DevOps kerää testiraportit ja tallentaa ne Pipeline Artifacts-säiliöön.

Tämän soveltuvuusselvityksen avulla osoitettiin Robot Frameworkin kyky suorittaa REST API -päätepisteiden (*endpoint*) testausta ja varmistettiin, että palvelimet kykenivät kommunikoimaan keskenään. Lisäksi testatattiin järjestelmien yhteistoimintaa ja testausprosessin toimivuutta. Kuviossa 3 soveltuvuusselvityksen kokonaiskuva ja ajojärjestys.



Kuvio 3. Soveltuvuustestin toiminta

Testausputki käynnistetään automaattisesti versionhallintaan tehtyjen muutosten yhteydessä tai se voidaan ajaa manuaalisesti. Testausputken konfigurointi on toteutettu YAML-tiedostossa (kuvio 4), jossa määritellään testien sijainti ja niiden käyttämät parametrit.

test-integration-poc.yaml

.azure-pipelines > integration > frends4 > test-integration-poc.yaml

```yaml
# Trigger the pipeline on changes to the main branch
trigger:
- main

# Agent pool where the pipeline will run
pool:
  name: ***

# Set common variables to use throughout the pipeline for easier management and updates
variables:
  azure_subscription: '***' # Specifies the Azure service connection for authorization
  key_vault_name: '***' # The Azure Key Vault from which secrets are fetched
  api_key_secret_name: 'frends-dev-0910-api-key' # Specifies the secret to fetch; can be a comma-separated list if fetching multiple secrets
  test_path: 'tests/integration/frends4/poc' # Path where tests are located
  test_output_path: 'results/integration/frends4/poc' # Path where test results and reports will be stored

steps:
# Check out the repository code to the agent
- checkout: self

# Fetch the API key from Azure Key Vault to use it in subsequent tasks
- task: AzureKeyVault@2
  inputs:
    azureSubscription: $(azure_subscription)
    KeyVaultName: $(key_vault_name)
    SecretsFilter: $(api_key_secret_name)

# Execute Robot Framework tests using the secret API key fetched from Azure Key Vault
- script: |
    robot --variable API_KEY:$(frends-dev-0910-api-key) --outputdir $(test_output_path) $(test_path)/test.robot
  displayName: 'Run Robot Framework POC Test'
  env:
    API_KEY: $(frends-dev-0910-api-key) # Sets the fetched API key as an environment variable for use in the script

# Publish JUnit formatted test results to Azure DevOps for easy viewing and analysis
- task: PublishTestResults@2
  inputs:
    testResultsFormat: 'JUnit' # Specifies the format of the test results to be published
    testResultsFiles: '$(test_output_path)/output.xml' # The path to the test results file
    failTaskOnFailedTests: true # If any tests fail, this task will cause the pipeline to fail
  displayName: 'Publish Test Results'

# Aggregate and publish all test artifacts, including results and logs, to Azure DevOps
- task: PublishBuildArtifacts@1
  inputs:
    PathtoPublish: $(test_output_path) # Directory containing test results and additional artifacts
    ArtifactName: 'TestArtifacts' # Names the published artifact for easy identification
    publishLocation: 'Container' # Specifies that the artifacts are published to Azure Pipelines
  displayName: 'Publish Test Artifacts'
```

Kuvio 4. Testausputken määrittelytiedosto Azure DevOpsissa

Testien määrittelyt on kirjoitettu erilliseen ROBOT-tiedostoon (kuvio 5), joka sisältää varsinaisen testilogiikan ja testattavat skenaariot. Testitiedostossa on määriteltynä Robot Frameworkin testille luotuja avainsanoja, joita kutsutaan testiajossa.

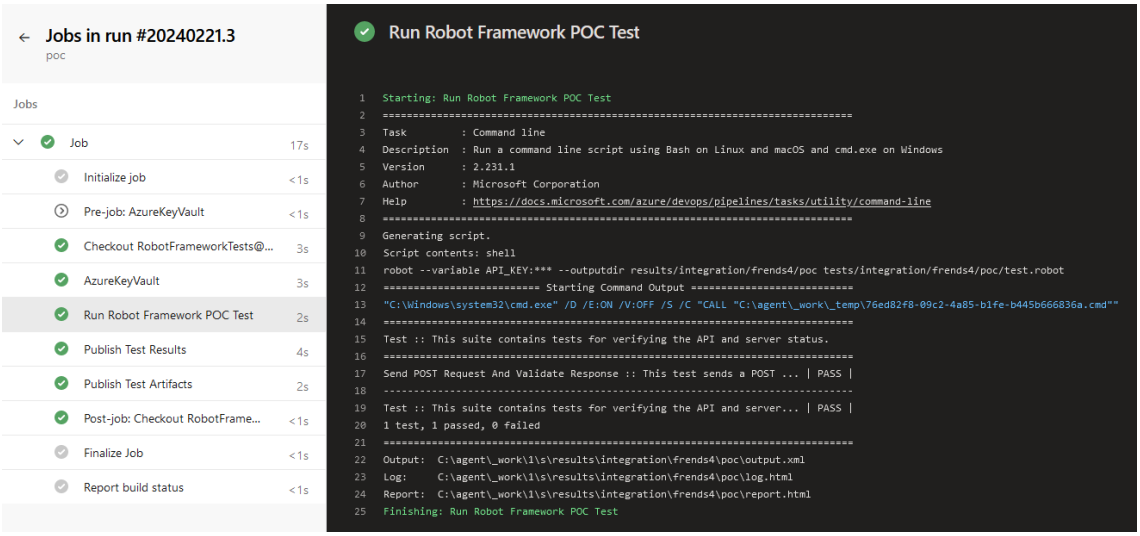
test.robot

tests > integration > frends4 > poc > test.robot

```
*** Settings ***
Library           RequestsLibrary               # Importing necessary libraries
Library           Process
Documentation     This suite contains tests for verifying the API and server status.
Resource          ../../../../configs/configs.robot   # Importing global configurations
Resource          ../../../../resources/keywords.robot  # Importing custom keywords

*** Variables ***
${api_path}       /robot_framework                  # API endpoint
${api_key}        %{API_KEY}                        # fetch key from Azure Vault
&{headers}        x-ApiKey=${api_key}               # Headers for the API request
${expected_string}       ok
${flask_server_url}    http://127.0.0.1:5000        # URL of the Flask server
${timeout_seconds}    10                           # Timeout in seconds
${interval_seconds}   1                            # Interval in seconds

*** Keywords ***
# Initializes the test and starts Flask server to listen on requests
Initialize Test
    [Documentation]  Initializes the test environment.
    Disable SSL Warnings
    Log  Python executable: ${PYTHON_EXECUTABLE}
    Start Process    ${PYTHON_EXECUTABLE}    simple_server.py    shell=True    cwd=${CURDIR}
    Log  API URL: ${FRENDS4_DEV_URL}
    Create Session    MySession    ${FRENDS4_DEV_URL}

# Checks that Frends returns 'ok' after POST
Process Response
    [Documentation]  Processes and validates the API response.
    [Arguments]    ${response}
    ${processed_response}=    Evaluate    json.loads('''${response.json()}''')    json
    Should Be Equal As Strings    ${processed_response}    ${expected_string}

# Checks Flask server if Frends has made a POST request
Wait For Server Response
    [Documentation]  Waits for a specific response from the server.
    ${current_time}=    Get Time    epoch
    ${end_time}=    Evaluate    ${current_time} + ${timeout_seconds}

    FOR    ${index}    IN RANGE    0    ${timeout_seconds}    ${interval_seconds}
        ${current_time}=    Get Time    epoch
        Exit For Loop If    ${current_time} >= ${end_time}

        ${status}=    GET On Session    MySession    ${flask_server_url}/check-status
        Exit For Loop If    '${status.content}' == 'Received'
        Sleep    ${interval_seconds} seconds
    END

    Should Be Equal As Strings    ${status.content}    Received

*** Test Cases ***
Send POST Request And Validate Response
    [Documentation]  This test sends a POST request and validates the response.
    Initialize Test
    ${response}=    POST On Session    MySession    ${api_path}    headers=${headers}
    Process Response    ${response}
    Wait For Server Response

```

Kuvio 5. Robot Frameworkin testikoodi

Kun testausputki on onnistuneesti ajettu, se luo ja esittää testiraportin keskeiset kohdat tarjoten välittömän näkymän testien tuloksiin (kuvio 6). Tarkempi raportti ja testien tuottamat tiedot tallennetaan DevOpsin Artifact-säiliöön, josta ne ovat saatavilla tarkempaa analyysiä varten.



Kuvio 6. Testitulosnäkymä

3.3 Testausautomaation suunnittelu

Testausautomaation suunnittelussa tavoitteena on luoda tehokas ja kattava testijärjestelmä, joka tunnistaa ohjelmistokehityksen aikana ilmenevät ongelmat. Testausautomaation suunnittelussa täytyy huomioida, että kaikkea ei ole hyödyllistä testata. Esimerkiksi suorien tiedostonsiirtojen testaaminen, missä tiedostoja ei muokata siirron aikana, ei välttämättä tuo lisäarvoa testausprosessiin. Käyttöliittymätestaus taas tarjoaa mahdollisuuden testata ja varmistaa käyttöliittymän toimivuus, mutta tässä työssä jätetään käyttöliittymän kautta suoritettavat automaattitestit toteuttamatta. Turvallisuustestaus, kuten SQL-injektioiden testaus on olennainen osa testausta, mutta toimeksiantajan järjestelmiin ei suoriteta suoria SQL-syötteitä eikä tässä työssä keskitytä siihen.

Testien tulosten tulisi olla mahdollisimman objektiivisia jättämättä tilaa tulkinnalle: ne joko ovat onnistuneita tai eivät. Testidatan generointi lähdejärjestelmästä varmistaa datan oikeellisuuden, mutta tämä edellyttää kehitys- tai testausympäristöjä, joita ei kaikissa järjestelmissä ole saatavilla. Tästä syystä

testitapauksissa voidaan simuloida lähde- tai kohdeympäristöjä datan luonnissa tai sen vastaanotossa. Laadukkaissa testeissä tarkastellaan monenlaisia syötteitä ja varmistetaan ennustettavien lopputulosten toteutuminen. Tuottamalla testidataa eri muuttujayhdistelmillä, joiden odotetut lopputulokset on määritelty etukäteen, pyritään saavuttamaan laaja testikattavuus.

Testien on tärkeä vastata liiketoiminnan vaatimuksiin, ja niiden määrittelyn tulisi ainakin osittain perustua käyttäjätarinoihin. Testien tulee olla toisistaan riippumattomia, mikä helpottaa virheiden lähteen selvittämistä. Testidatan hallinnan on oltava helppoa mahdollistaen sen uudelleenkäytön eri testeissä. Lisäksi testien muuttujien hallinta tulee suorittaa rakenteellisesti ja yhteiset parametrit useille testeille tulee tallentaa keskitetysti. Tämä helpottaa testausprosessin ylläpitoa ja varmistaa sen, että testaus vastaa sekä teknisiä että liiketoiminnallisia tarpeita.

Testitapaukset valittiin sen perusteella, miten saataisiin kattava kuvaus Robot Frameworkin mahdollistamista testaustavoista. Nämä tapaukset eroavat toiminnaltaan toisistaan suuresti. Testattavat prosessit ovat oleellisia liiketoiminnan tarpeille, ja niille löytyy testijärjestelmät tai testattavia järjestelmiä on mahdollista simuloida esimerkiksi jaetulla verkkolevyllä, ilman että tästä on haittaa tuotantoympäristön toiminnalle.

3.3.1 Tapaus 1: Asiakasnumeroiden haku tunnisteen perusteella

Ensimmäinen testattava prosessi on REST-rajapinta, joka palauttaa henkilö- tai y-tunnuksen eli tunnisteen perusteella kaikki asiakkaan asiakasnumerot. Asiakasnumerot haetaan muutamasta eri tietokannasta ja palautetaan JSON-muodossa. Prosessi on tärkeä Vaasan Sähkön mobiilisovelluksen toiminnan kannalta, jotta asiakkaat pääsevät katselemaan omia tietojaan.

Testausprosessissa haetaan tietokannoista sattumanvaraisia tunnisteita ja niiden asiakasnumerot. Integraation rajapinnasta sitten kysellään tunnisteen perusteella asiakasnumerot ja vertaillaan saadaanko odotettu tulos.

3.3.2 Tapaus 2: Toiminnanohjausjärjestelmän ostolaskujen tarkistus

Toinen testattava prosessi käsittelee laskutusjärjestelmän luomia ostolaskuja, jotka siirretään toiminnanohjausjärjestelmään. Prosessin kulku alkaa ostolaskujen lukemisella verkkolevyltä, minkä jälkeen prosessi suorittaa tarkistuksia. Toiminnanohjausjärjestelmän rajapinnasta selvitetään ostonumero ja haetaan konsernin sisäisille laskuille projektitunnus. Tämän jälkeen muokattu ostolasku viedään toiminnanohjausjärjestelmään. Prosessin käsintestaus ei ole järin suuritöinen, mutta tällä pyritään helpottamaan testausta päivitettäessä toiminnanohjausjärjestelmä uuteen pilvessä toimivaan versioon. Uusi versio täytyy testata huolellisesti ennen sen hyväksymistä tuotantokäyttöön.

Testausprosessi alkaa luomalla suuri joukko XML-muotoisia ostolaskuja käyttäen Pythonin tekstipohjia (*string template*), joihin voidaan määritellä parametreille paikat ja luoda eri versioita ostolaskuista. Tämä mahdollistaa useamman mahdollisen ostolaskuversion testaamisen. Testauksessa käytetään myös oikeita ostolaskutiedostoja. Testauksen aikana seurataan, että prosessi suorittaa kaikki vaiheet oikein: verkkolevyltä tiedoston lukemisen, ostonumeron tarkistuksen ERP:n tietokannasta, konsernin sisäisen laskun tarkistuksen ja projektitunnuksen lisäämisen sekä tiedoston siirron ERP:een. Lopuksi tarkistetaan, että tiedot ovat menneet oikein ERP:n tietokantaan. Tavoitteena on, että testit tuottavat saman tuloksen riippumatta kantaversion muutoksista.

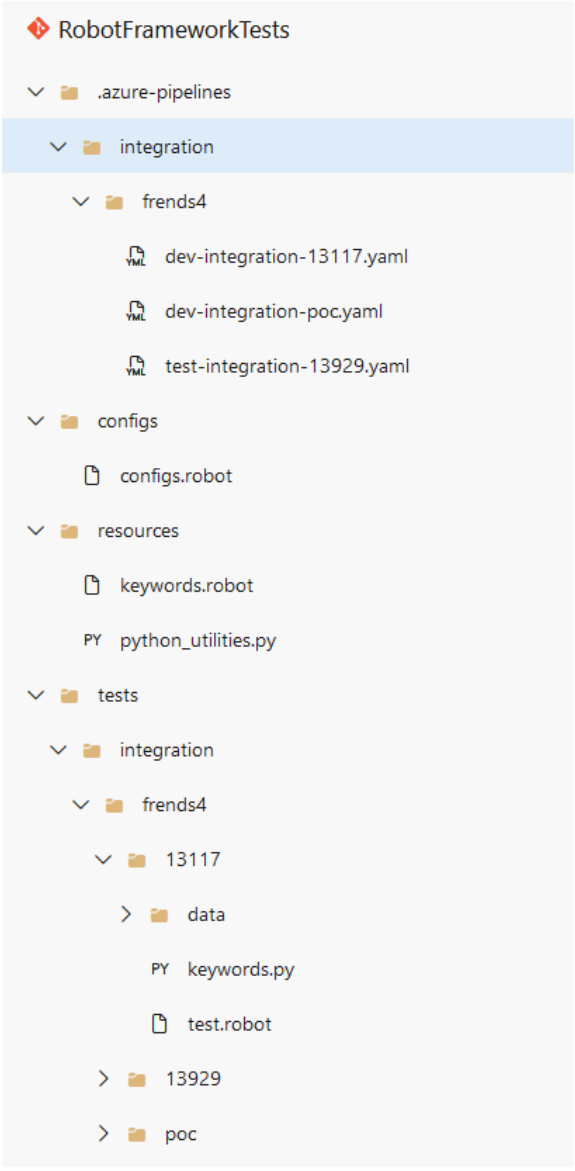
3.3.3 Tapaus 3: GSRN-numeroiden lisäys mittaustietoihin

Viimeinen testattava prosessi lukee sähkömittarointijärjestelmän luomia SAF-tiedostoja, jotka sisältävät käyttöpaikan kulutustietoja. Tiedostoissa on tunnisteena sähköyhtiön oma käyttöpaikkatunnus, mutta ei Fingridin Datahubin myötä käyttöönotettua GSRN-tunnistetta. GSRN-tunniste haetaan asiakastietojärjestelmästä ja lisätään tiedostoon omalle sarakkeelle.

Testausprosessissa lähdekansioon syötetään tiedostoja ja luetaan kohdekansion sisältö. Testitiedostojen nimeen on merkitty käyttöpaikkanumero ja GSRN-tunniste ja testausprosessi tarkistaa että tiedostonimeen merkitty GSRN-tunniste löytyy käsitellystä tiedostosta. Lopuksi prosessi siivoaa prosessin kohdekansioon luodut tiedostot.

3.4 Testausputken luonti ja käyttö

Testeille luotiin yhteisiä ja omia avainsanoja sen mukaan, kuinka niitä odotetaan uudelleenkäytettävän. Tämä helpottaa koodikannan ylläpitoa, kun usein käytettäviä toimintoja, kuten SQL-kyselyä tai tiedoston parsimista varten ei tarvitse aina kirjoittaa uutta koodia. Tässä Robot Frameworkin modulaarisuus toimii hyvin. Jokainen testi sai oman YAML-konfiguraatiotiedoston, jotta testien seuranta ja liipaisu olisi helppoa ja epäonnistuneet testit olisi helppo linkittää oikeaan prosessiin. Kansiorakenne pidettiin yhtenäisenä jokaisessa testissä, jotta kansio- ja tiedostonimistä voisi päätellä, mihin testiin ne kuuluvat (kuvio 7).



Kuvio 7. Projektin kansiorakenne

Testien luonti Robot Frameworkin syntaksin ja toiminnan opettelun jälkeen ei ollut prosessin aikaavievin osuus, vaan testidatan luonti ja tarkistusprosessin selvittely. Yhteiset avainsanat toivat myös ongelmia joita ei heti osattu ottaa huomioon. Kolmatta testiä luodessa lisättiin kolmannen osapuolen Python-kirjasto, jota ei oltu asennettu testipalvelimelle ja kolmatta testiä viedessä versionhallintaan rikkoi se myös aiemmat testit.

Testit erosivat toisistaan sen verran paljon, että jokaisen luonti vaati hieman erilaista lähestymistapaa. Ensimmäisessä tapauksessa missä keskityttiin asiakasnumeroiden hakuun henkilötunnuksen perusteella oli suoraviivainen toteuttaa. Testiä varten luotiin yhteinen avainsana PostgreSQL-tietokannan kyselyille testidatan hakemiseen, syötettiin tämä data REST-rajapintaan ja analysoitiin vastaukset. Tämä prosessi testasi hyvin tehokkaasti rajapinnan toiminnan sattumanvaraisilla tunnisteilla. Testiä olisi voinut vielä laajentaa sisällyttämällä siihen virheellisiä tunnisteita, jotta olisi voitu samalla testata integraation kyky käsitellä tyhjiä tai virheellisiä pyyntöjä.

Toisen tapauksen toteutus joka keskittyi ostolaskujen viemiseen toiminnanohjausjärjestelmään jäi valitettavasti kesken. Tämä tapaus korosti automatisoidun testauksen teknistä vaativuutta, varsinkin testattaessa prosessia, joka sisältää huomattavan määrän muuttujia, tässä tapauksessa noin sata. Laskutietojen syöttäminen satunnaisgeneroiduilla numeroilla ja tekstillä ei ollut mahdollista ilman olemassa olevaa toimittajatietuetta, ja vanhojen laskujen käyttäminen vaati niiden sisällön muokkaamista että ne menevät toiminnanohjausjärjestelmän omasta tarkistuksesta läpi. Kesken jäänyt toteutus osoitti hyvin kuinka aikaavievää ja työlästä testien luominen saattaa olla.

Viimeinen tapaus jossa lisättiin GSRN-numeroita mittaustietoihin oli puolestaan suoraviivaisempi toteuttaa. Luomalla testattavat tiedostot käsin ja merkitsemällä niihin odotetut muutokset, pystyttiin suorittamaan tehokkaita testejä. Testin toteutusta olisi voinut parantaa vielä hakemalla testattava data dynaamisesti tietokannasta ja testata myös varmasti epäonnistuvia tiedostoja.

Kokonaisuudessaan nämä tapaukset antoivat hyviä oppeja automatisoidun testauksen soveltamisesta ja sen haasteista integraatioprosessien yhteydessä. Automaattisen testauksen toteutus vaatii huolellista suunnittelua ja testidatan

hallintaa sekä syötettäessä että sitä tulkitessa, mutta tarjoaa myös tehokkaita keinoja laadunvarmistukseen.

4 POHDINTA

Opinnäytetyön käytännön hyödyt ovat vielä pienet, mutta se osoittaa testausautomaation potentiaalin integraatioiden laadunvarmistuksessa ja seurannassa. Työ antoi hyvän kuvan sen vaatimasta työmäärästä ja osaamista sen toteuttamiseen ja arviointiin. Helpoimpia testattavia ovat prosessit, jotka eivät tee suoraan muutoksia järjestelmään, vaan niiden lopputulosta voi tarkastella esimerkiksi tiedostomuodossa. Jos prosessi tekee muutoksia jonkin järjestelmän tietokantaan, muuttuu testaus huomattavasti työläämmäksi.

Opinnäytetyössä luodut testit kelpaavat sellaisenaan toimeksiantajan käyttöön, ja ne toimivat hyvänä esimerkkinä testauksen laajentamiseen. Robot Frameworkia voisi käyttää myös järjestelmien suoraan testaamiseen pelkän integraatiotestauksen lisäksi, ja tästä työstä saadut kokemukset auttavat mahdollisten käyttökohteiden tunnistamisessa. Joidenkin integraatioiden testauksessa, joissa odotetut lopputulokset ovat helposti generoitavissa, voisi automaattitestausta hyödyntää integraatioita testivetoisesti kehittäessä (*Test Driven Development, TDD*). Tästä saadut kokemukset auttavat myös muiden testausalustojen vertailussa.

Projektin alkuvaiheessa asetetut tavoitteet osoittautuivat liian kunnianhimoisiksi ja ne muutettiin realistisemmiksi testejä vähentämällä ja lopulta jättämällä yksi testi toteuttamatta. Testien tekninen toteutus oli melko helppoa Robot Frameworkin perusteiden selkiydyttyä, mutta testien suunnittelu ja testitulosten varmistaminen ei ollut niin suoraviivaista kuin odotettiin. Eniten aikaa kului toiminnanohjausjärjestelmän tietokantahakujen selvittelyyn, ja tässä olisi pitänyt olla apuna joku, joka tuntee kannan ja liiketoiminnan, jotta halutut tiedot olisi saatu helposti esille. Testausympäristöjen pystytys ja valmistelu olivat myös aikaavieviä, vaikka nämä olivatkin kertaluonteisia töitä.

Aiemmassa luvussa listatut parhaat käytännöt toteutuivat työssä suurimmaksi osaksi. Avainsanojen nimeäminen vaatisi vielä hieman pohdintaa, jotta ne olisivat yhtenäisiä ja selkeitä. Pythonin ja Robot Frameworkin koodi on itsessään melko selkeää ja sellaisenaan itsedokumentoivaa, varsinkin kun muuttujien nimet ovat selkeitä, mutta funktioiden dokumentaatiota olisi voinut vielä parantaa

esittelemällä argumenttien ja palautusten tyypit paremmin. Ylläpitoon ja seurantaan ei työn aikana ehditty tutustua tarkemmin. Ajatuksena oli myös hyödyntää Frendsin omaa rajapintaa ja hakea sieltä tieto integraatioversion muutoksista ja näin liipaista testiajot, mutta tämä jäi vain ajatuksen asteelle.

Robot Framework ei ole testauksen suosituin tai modernein alusta, mutta siihen löytyy kohtuullisesti apua internetistä. Robot Framework on suunniteltu avainsanat edellä, jotta testien tekemiseen ei tarvittaisi ohjelmointiosaamista, mutta itse epäilen tämän väitteen paikkansapitävyyttä. Sillä jos osaa tehdä testejä Robot Frameworkilla, oppii myös helposti Pythonin.

Pythonin ja Robot Frameworkin syntaksit eroavat toisistaan ja se sekoittaa työskentelyä. Pythoniin tottuneelle Robot Framework voisi olla miellyttävämpi käyttää, jos se toimisi Pythonin kirjastona eikä frameworkina. Työn edetessä Robot Frameworkin perusteet ja toimintalogiikka tulivat kuitenkin tutuiksi ja syntaksin tuottamat ongelmat pystyi minimoimaan luomalla Pythonilla tärkeimmät funktiot, joille sitten määritteli Robot Frameworkin avainsanan. Vasta testien valmistuttua huomasin, että Pythonin pytest-kirjastossa olisi ollut vastaavat ominaisuudet Robot Frameworkin kanssa ja sen käyttö olisi ollut minulle miellyttävämpää.

Azure DevOpsin hyödyntäminen osoittautui helpoksi ja sen moduulit, kuten salasanasäiliö ja artifaktit soveltuivat hyvin testauskäyttöön ja tekivät testaamisen tietoturvallisemmaksi. DevOpsin etuna oli, että se on jo valmiiksi toimeksiantajayrityksen käytössä ja alustana tuttu, mutta tässä työssä pääsin hyödyntämään sen testausputkia ja yhdistää moduuleita sen käyttöön. DevOpsin muuttujasäiliötä ei testattu, mutta sitä voisi hyödyntää ympäristövalinnan teossa. Ympäristönvalinnan voisi muuttaa dynaamisemmaksi, jos jatkossa halutaan testata myös tuotannon toimintoja. Testausputkien testejä voisi parantaa laittamalla ne ajamaan Pythonin erillisessä virtuaaliympäristössä (*python virtual environment*), sillä nyt ne ajetaan kaikki samassa Pythonin ympäristössä.

Tämän työn tuloksena saavutetut ratkaisut soveltuvat hyvin monenlaisten järjestelmien testaamiseen, kunhan tietyt ehdot täyttyvät. Järjestelmällä tulee olla selkeä rajapinta. Vaikka käyttöliittymän (web tai natiivi) kautta toimivaa testausta voi toteuttaa Robot Frameworkillä käyttäen esimerkiksi Selenium-kirjastoa tai

konenäköä, on tämä menetelmä huomattavasti työläämpi. Lisäksi täytyy tietää, mitä sisäänsyötetty data tekee järjestelmässä, minne se menee ja miten sitä käsitellään ja täytyy voida varmistaa datan oikea käsittely, esimerkiksi tietokantahaun avulla. Järjestelmän testaaminen edellyttää testiympäristöä tai sen simulointimahdollisuutta, esimerkiksi tiedostopalvelimen muodossa, ja testidatan siivous testattavasta järjestelmästä onnistuu parhaiten rajapinnan kautta, jotta testaus voidaan automatisoida täysin.

Hyvien testien luonti vaatii monen osa-alueen osaamista. Tarvitaan SQL-tietokantaosaamista testidatan luomista ja tarkistusta varten, ymmärrystä rajapinnoista ja niiden toiminnasta, kykyä käsitellä eri tiedostotyyppejä ja -formaatteja sekä ohjelmointitaitoja testien ja testidatan luomiseen. Haastavampien testitapauksien kohdalla on suositeltavaa, että useampi asiantuntija osallistuu prosessiin, jotta työtaakka voidaan jakaa. Esimerkiksi tulosten vertailu tietokantahaun avulla vaatii ymmärrystä tietokannan rakenteesta ja testattavasta järjestelmästä sekä usein myös liiketoiminnan tuntemusta.

Koska toimeksiantajayritys ei ole ohjelmistotalo tai edes IT-alalla, ei testausautomaatiota pyritä ottamaan käyttöön kaikissa mahdollisissa käyttötapauksissa, mutta toimeksiantaja tulee luultavasti hyödyntämään testausautomaatiota jatkossa rajatusti tärkeimpien prosessien testaukseen. Testaus vaatii monesti testattavien järjestelmien järjestelmäasiantuntijoiden tai järjestelmätoimittajien apua, että voidaan varmistua halutusta lopputuloksesta. Tämä on aikaavievää ja siksi testausautomaation laajempi käyttöönotto vaatisi oman, siihen keskittyneen työntekijän täyden työpanoksen, eikä sitä ole mielekästä hoitaa sivutyönä muiden töiden ohessa laajassa mittakaavassa.

Frendsin toimittaja on kertonut, että he aikovat lisätä testaustoiminnallisuutta itse järjestelmään ja tämä tieto löytyykin heidän kehityspolulta (*roadmap*), aikataulua tälle ei kuitenkaan ole määritelty (Frends Technology 2024d). Frendsin natiivitestaus on varmasti hyvä lisä ja hyödyllinen työkalu tulevaisuudessa, ja natiivitestaus on luultavasti kypsyessään helpompi käyttää ja selkeämpi kuin Robot Frameworkin testit. Robot Framework saattaa kuitenkin olla joissain tapauksissa parempi ratkaisu sen joustavuuden takia. Robot Framework sopii

kuitenkin myös muiden järjestelmien testaamiseen ja on siihen varmasti vahvempi vaihtoehto juurikin sen joustavuuden takia.

Mahdollisia jatkotutkimusaiheita voisi olla Frendsin oman testaustoiminnallisuuden arviointi, kun se julkaistaan. Sen kokeilu integraatioprosessien testaamisessa voisi olla hyödyllistä ja antaa kuvan sen toimivuudesta verrattuna olemassa oleviin ratkaisuihin. Toinen ajatus jatkotutkimusaiheeksi voisi olla selvitys, mitä testausstrategioita kannattaisi käyttää integraatioita testatessa, ohjeistus miten lähestyä erilaisia integraatioita, mikä toimii ja mikä ei. Myös Pythonin pytest-kirjaston soveltaminen Frendsin testauksessa voisi tarjota tietoa siitä, miten tämä testauskirjasto soveltuisi siihen verrattuna Robot Frameworkiin.

LÄHTEET

Bose, S. 2023. Benefits of automation testing. BrowserStack 14.9.2023. Viitattu 25.3.2024 https://www.browserstack.com/guide/benefits-of-automation-testing/.

Chowdhury, A. 2024. DevOps best practices: A complete guide. Stackify 19.3.2024. Viitattu 25.3.2024 https://stackify.com/devops-best-practices-a-complete-guide/.

Cigniti 2024. The top 10 benefits of test automation. Cigniti 6.2.2024. Viitattu 25.3.2024 https://www.cigniti.com/blog/top-10-benefits-test-automation/.

Frends Technology 2024a. What is an iPaaS? Viitattu 25.3.2024 https://frends.com/platform/what-is-an-ipaas/.

– 2024b. Not all strangers. Viitattu 25.3.2024 https://frends.com/about-us/.

– 2024c. Process Automation. Viitattu 28.4.2024 https://frends.com/platform/process-automation/.

– 2024d. Frends roadmap. Viitattu 3.4.2024 https://frends.com/roadmap/.

– 2024e. frends Docs - Integration Platform Documentation. Viitattu 29.4.2024 https://docs.frends.com/.

Microsoft 2024. What is Azure DevOps? Microsoft 4.1.2024. Viitattu 25.3.2024 https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?toc=%2Fazure%2Fdevops%2Fget-started%2Ftoc.json&view=azure-devops/.

Novovic, M. 2019. Robot Framework CI/CD with Azure DevOps. Medium 11.4.2019. Viitattu 3.4.2024 https://milannovovic.medium.com/robot-framework-ci-cd-with-azure-devops-cf708a64b389/.

Patel, T. 2023. How to Calculate ROI on Test Automation? [Save Time & Money]. Testgrid 1.3.2023. Viitattu 17.4.2024 https://testgrid.io/blog/roi-on-test-automation/.

Robot Framework 2024a. Robot Framework. Viitattu 25.3.2024 https://robotframework.org/.

– 2024b. Robot Framework User Guide. Viitattu 29.4.2024 https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html/.

Snaplogic 2024. Software integration – Definition and overview. Viitattu 25.3.2024 https://www.snaplogic.com/glossary/software-integration/.

Testsigma 2023. Top 7 Challenges in Manual Testing. Testsigma 12.9.2023. Viitattu 29.4.2024 https://testsigma.com/blog/is-manual-testing-becoming-a-bottleneck-in-continuous-delivery/.

Trisotech 2024. BPMN Quick Guide. Viitattu 28.4.2024 https://www.bpmnquickguide.com/view-bpmn-quick-guide/.

Unadkat, J. 2022. 4 Key Software Testing Challenges and their Solutions. BrowserStack 5.10.2022. Viitattu 29.4.2024 https://www.browserstack.com/guide/software-testing-challenges/.

Vaasan Sähkö 2024a. Koko suomen energiayhtiö. Viitattu 25.3.2024 https://www.vaasansahko.fi/tietoa-vaasan-sahkosta/koko-suomen-energiayhtio/.

– 2024b. Tietoa Vaasan Sähköstä. Viitattu 25.3.2024 https://www.vaasansahko.fi/tietoa-vaasan-sahkosta/.

Yritys- ja yhteisötietojärjestelmä 2024. Frends Technology Oy. Viitattu 25.3.2024 https://tietopalvelu.ytj.fi/yritys/0648086-9/.