Tampereen ammattikorkeakoulu

UI-testiautomaation aloitus Robot Frameworkia hyväksi käyttäen

Anu Malm

OPINNÄYTETYÖ
Joulukuu 2020

Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

MALM, ANU:
UI-testiautomaation aloitus Robot Frameworkia hyväksi käyttäen

Opinnäytetyö 60 sivua
Joulukuu 2020

Opinnäytetyön tehtävänä oli selvittää, miten UI- eli käyttöliittymän testiautomaation aloitus tapahtuu Robot Frameworkilla. Opinnäytetyössä toteutettiin UI-testiautomaation aloituspalikat työn toimeksiantajalle Piceasoft Oy:lle. Testiautomaation tavoitteena oli vähentää manuaalitestauksen määrää ja laajentaa Piceasoft Oy:n tuotteidenhallintasovelluksen regressiotestausta. Testiautomaatio sopii erinomaisesti juuri regression testaamiseen, eli vanhojen ominaisuuksien laadun varmistamiseen. Manuaalisesti tehty regressiotestaus on altis ihmisvirheille, koska regressiotestauksessa joudutaan testaamaan asioita, jotka on jo testattu moneen kertaan.

Opinnäytetyö johdattelee testaamiseen käymällä aluksi läpi yleisesti testiautomaation ja manuaalitestaamisen eroja. Työssä selvitetään, millaisia hyviä tapoja testiautomaation tekoon liittyy. Testiautomaatiotyökalu on hankala vaihtaa kesken projektin, joten ennen Robot Frameworkiin päätymistä oli testin alla myös Cypress -niminen verkkosovellusten automatisointityökalu.

Verkkosovelluksen testiautomaatiota varten käyttäjä joutuu asentamaan koneelleen Python-ohjelmointikielen, Robot Frameworkin, SeleniumLibraryn ja WebdriverManagerin uusien testien luomista varten. Nämä vaiheet käydään läpi opinnäytetyössä Windows 10 -käyttöjärjestelmän näkökulmasta. Tämän jälkeen tutustutaan Robot Frameworkin kanssa työskentelyyn. Tämä pitää sisällään muun muassa selvityksen, miten testejä tehdään ja ajetaan sekä miten testituloksia luetaan.

Piceasoft Oy:lle luotu testiautomaatio on toteutettu helppokäyttöisyys ja -lukuisuus etusijalla, jotta testituloksien luku onnistuu lähes keneltä tahansa. Avainsanojen ja testauslogiikan dokumentointi auttaa testien jatkokehityksessä ja ylläpidossa; testiautomaatio ei ole koskaan täysin valmis, kun testattavana on kehittyvä sovellus.

Asiasanat: testiautomaatio, robot framework, käyttöliittymätestaus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business Information Systems
Software Development

MALM, ANU:
Starting UI Test Automation with Robot Framework

Bachelor's thesis 60 pages
December 2020

The objective of this thesis was to find out how UI (User Interface) test automation could be created with Robot Framework. This goal was actualized as the web application test automation building blocks for the commissioner Piceasoft Ltd. The purpose of the test automation was to ease Piceasoft Ltd's manual testing load on their Product Management Console and to increase regression testing. Test automation is especially suited for regression testing, as manual testing of old features is prone to human error due to its repetitive nature.

The thesis first covers the differences in test automation versus manual testing. The next chapter focuses on general guidelines for creating good automated tests. Selecting the right test automation tool is critical for creating a sustainable testing environment as the selected tool is hard to change in the middle of development. For this purpose, Robot Framework was compared to another popular web application automation tool Cypress.

In order to create test automation tests in Robot Framework, Python, Robot Framework, SeleniumLibrary and WebdriverManager need to be installed. The installation steps for these are covered from the point of view of Windows 10 operating system.

After everything is installed, the thesis focuses on how to utilize Robot Framework. This includes creating and running Robot Framework tests, and reading the test logs after the test execution.

The primary aim was to write the test automation so that people with less technical experience can understand it, and possibly make use of it in their own work. The test automation keywords and logic were documented to further make the automation easier to approach. This also makes it possible to advance the test automation in the future; for as long as the software under test is improved upon, the test automation can never be truly finished.

Key words: test automation, robot framework, user interface testing

SISÄLLYS

1 JOHDANTO 7
2 TESTIAUTOMAATIO 11
2.1 Testiautomaatio vs. manuaalitestaus 12
2.2 Testiautomaatiossa huomioitavaa 12
2.3 Miksi Robot Framework? 14
3 ROBOT FRAMEWORK 17
3.1 Robot Frameworkin rakenne 17
3.2 Robot Framework testauksessa 18
3.3 Robot Framework ohjelmistorobotiikassa 21
3.4 Selenium 22
4 TESTIAUTOMAATION ALOITUS ROBOT FRAMEWORKILLA 24
4.1 Asennus Windows 10:lle 24
4.1.1 Python ja pip-paketinhallintasovellus 24
4.1.2 Robot Framework 26
4.1.3 SeleniumLibrary 27
4.1.4 WebdriverManager 27
4.1.5 Ongelmatilanteet asennuksessa 29
4.2 Testien suunnittelu 32
4.3 Testien toteutus 33
4.4 Testien ajo Robot Frameworkilla 34
4.5 Testitulokset 37
5 ROBOT FRAMEWORK TUOTTEIDENHALLINTASOVELLUKSEN KÄYTTÖLIITTYMÄN AUTOMATISOINNISSA 42
5.1 Piceasoft Oy:n tuotteidenhallintasovelluksen lyhyt esittely 42
5.2 Testiautomaation tavoitteet 43
5.3 Testitapaukset 45
5.3.1 Elementtien kanssa keskustelu 45
5.3.2 Esimerkkitestitapaus: uuden aliasiakkaan luominen 48
5.3.3 Hyvät menetelmät 51
5.4 Tulokset 54
6 POHDINTA JA JATKOKEHITYS 56
LÄHTEET 59

ERITYISSANASTO

headless-selain	Selainversio, jossa graafista käyttöliittymää ei renderöidä
kehitysversio	Sovelluksen versio, johon kehittäjät tuovat uudet ominaisuudet ensin testiin
kirjasto	Lisäosa ohjelmaan, jolla siihen saadaan uusia ominaisuuksia. Esimerkiksi Selenium on yksi kirjasto Robot Frameworkiin
modulaarisuus	Ohjelmiston jakamista pieniin, uudelleenkäytettäviin kokonaisuuksiin
PATH-ympäristömuuttujat	Windowsin globaaleja muuttujia, joihin kaikki Windows-ohjelmat pääsevät käsiksi
PMC	Lyhenne termistä Product Management Console, ks. tuotteidenhallintajärjestelmä
Python, Java	Ohjelmointikieliä
regressiotestaus	Testausta jolla selvitetään, onko jonkin vanha komponentti mennyt rikki uuden muutoksen myötä. Yleensä automatisoitua
RPA	Lyhenne termistä Robotic Process Automation, suomeksi ohjelmistorobotiikka. Rutiinitehtävien automatisointia
staging-versio	Sovelluksen versio, joka toimii uusien ominaisuuksien toiminnan varmistamiseen. Tarkoitus pitää mahdollisimman samanlaisena kuin tuotantoversio

testisarja	N määrä testejä, jotka ajetaan yhden testiajon aikana
tuotantoversio	Sovelluksen julkaistu versio, joka näkyy myös asiakkaille
tuotteidenhallinta-järjestelmä	Piceasoft Oy:n verkkosovellus, jota käytetään Piceasoft Oy:n asiakkaiden ja heidän tuotteidensa hallinnointiin
UI	Lyhenne englanninkielisestä termistä user interface, suomeksi käyttöliittymä
XPath	Lyhenne termistä XML Path Language. Käytetään nodejen eli solmujen osoittamiseen XML-dokumenteista (esim. verkkosivut)

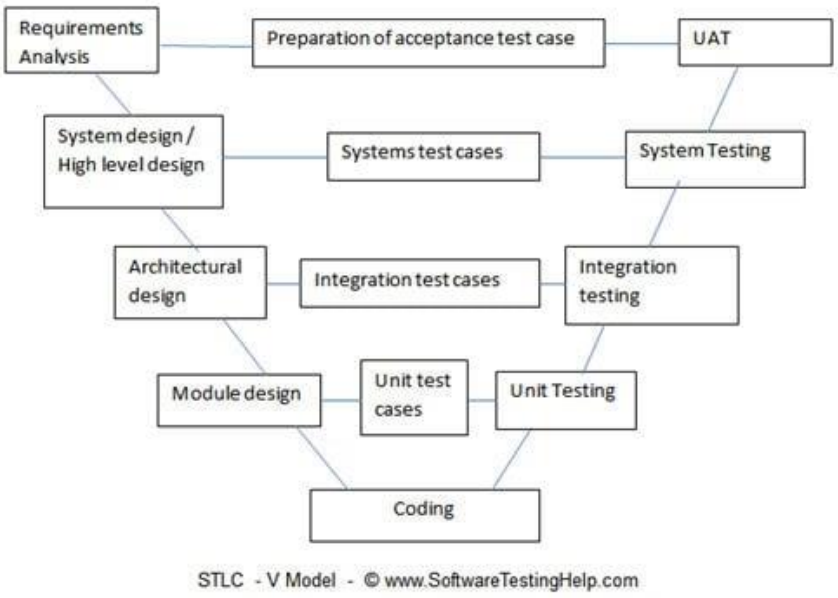
1 JOHDANTO

Ohjelmistokehitystä voidaan tehdä monella eri toimintatavalla. Näistä perinteisin malli on vesiputous, jossa kehitys valuu suunnitteluvaiheesta julkaisuun askel askeleelta ilman aiempiin vaiheisiin palaamista. Tämä on aikojen saatossa todettu liian jähmeäksi malliksi, ja sitä on pyritty modernisoimaan tuomalla kehitykseen mukaan ketteriä menetelmiä. Ketterien menetelmien etuna voidaan pitää nopeaa mahdollisuutta reagoida tarvittaviin muutoksiin, kun taas vesiputous-mallissa muutokset jäisivät joko toteuttamatta, tai niihin iskettäisiin kiinni uudella vesiputousmallisella projektilla. Vesiputousmallissa kehityskierrokset ovat pitkiä, koska niissä ohjelmisto täytyy suunnitella ja määritellä tarkkaan ennen sen toteutusta. Ketterissä menetelmissä taas kehityksen tilaa seurataan koko ajan, jolloin muuttuvien olosuhteiden tuomiin haasteisiin voidaan helposti reagoida. Ketterät menetelmät mahdollistavat myös testauksen tuomisen projektin elinkaareen aikaisemmassa vaiheessa. Mitä aikaisemmin testataan, sitä halvempaa ja helpompaa on korjata löydetyt ohjelmistovirheet (Black 2016, 1.5).

Ohjelmistokehityksen vaiheita kuvaa hyvin vesiputous-mallisen ohjelmistokehityksen paranneltu versio V-malli. Kuvassa 1 havainnollistetaan, miten vasemman puolen suunnitteluvaiheen askelmat sitoutuvat oikean puolen testausaskelmiin erilaisten testien kautta.

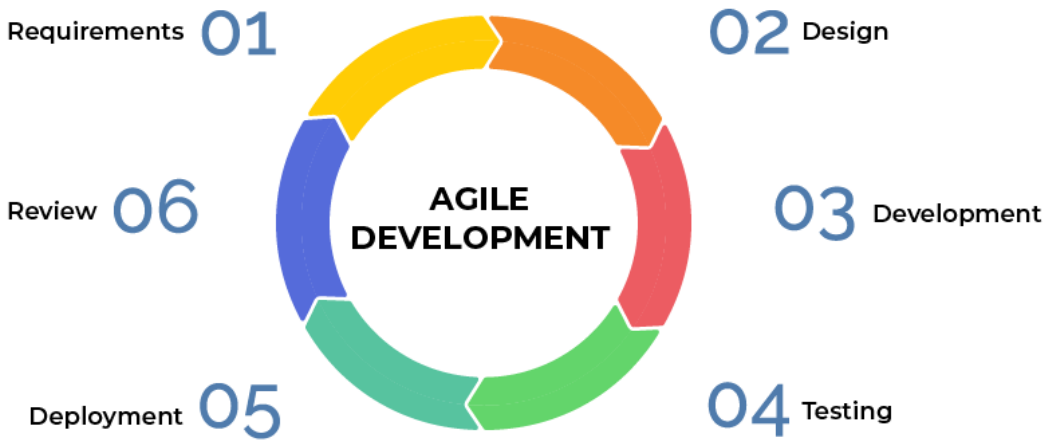
V-mallia luetaan vasemmalta sarakkeelta alaspäin ja sitten taas oikeaa saraketta ylös. Requirements Analysis eli vaatimusmäärittelyaskeleella selvitetään, mitä vaatimuksia toteutettavalle sovellukselle on. System Design eli järjestelmän suunnitteluvaiheessa aletaan selvittää, mitä järjestelmä- tai muita ylemmän tason vaatimuksia sovelluksella on. Architectural Design eli arkkitehtuurin suunnitteluvaiheessa otetaan jo laajempialaisemmin kantaa sovelluksen toteutukseen. Arkkitehtuurin jälkeen voidaan aloittaa tarkempi suunnittelu Module Design eli moduulisuunnitteluvaiheessa. Koodaus toimii V-mallin kärkenä, josta lähdetään oikeaa saraketta pitkin tekemään ohjelmiston hyväksyntätestejä. Moduulisuunnitelmaa vastaavat yksikkötestit, joilla on tarkoitus testata yksittäisiä ohjelman moduuleja. Arkkitehtuurisuunnitelmasta luodaan integraatiotestit, joilla tarkistetaan

eri sovelluksen osien yhteensopivuutta. Järjestelmäsuunnitelmasta luodaan järjestelmätestejä, joilla varmistetaan koko sovelluksen ja sen sisäisten järjestelmien toimivuus kokonaisuutena. Viimeisenä päästään vaatimusmäärittelyjen pohjalta tehtyyn käyttäjähyväksyntätesteihin (UAT, User Acceptance Testing). Tässä viimeisessä vaiheessa varmistetaan, että sovellus toimii määrittelyvaiheessa löydettyjen käyttäjän vaatimusten mukaan.



KUVA 1. Ohjelmistokehityksen V-malli (Software Testing Help 2020)

Ketterä kehitys soveltaa tätä V-mallia niin, että V:ssä ei liikuta jähmeästi vain sarakkeita pitkin. Kehitys toimii ketterissä menetelmissä yleensä kehityskierroksissa eli sprinteissä, jolloin aloitusmäärittelyjen jälkeen keskitytään tuottamaan ohjelmistoa pieninä paloina. V-malli yhdistetään sakaroistaan ympyräksi lisäämällä vaatimusmäärittelyjen ja käyttäjätestauksen väliin katselmusvaiheen, jossa tarkastellaan, miten sovelluksen kehitys on edennyt ja tarvittaessa reagoidaan ilmenneisiin muutostarpeisiin. Kuvassa 2 on ketterän kehityksen kehityskierros kuvattuna ympyränä. Ketterässä kehityksessä kehityskierros kestää yleensä vähintään viikon, koska lyhyemmässä ajassa on vaikea saada toteutettua mitään julkaisukelpoista. Kehityskierroksen enimmäisaika taas on määritelty neljäksi viikoksi, koska sen jälkeen ketteryys alkaa kärsiä huomattavasti. Kuukaudessa ehtii tapahtua paljon, joten muutoksiin reagoiminen hidastuu kehityskierroksen pituuden kasvaessa.



KUVA 2. Ketterän ohjelmistokehityksen vaiheet (Agile Tech 2020)

Tämä opinnäytetyö on kehitetty osana ketterää ohjelmistokehitysprojektia, ja se keskittyy ohjelmiston testausvaiheeseen. Testausvaiheesta syvennyttiin testiautomaatioon Robot Frameworkin avulla.

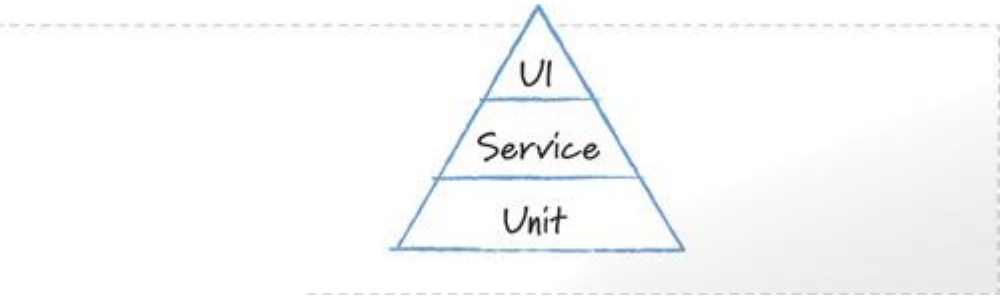
Opinnäytetyön toimeksiantajalla Piceasoft Oy:lla todettiin tarve käyttöliittymän (eli UI:n) testiautomaatioon heidän tuotteidenhallintajärjestelmäänsä. Opinnäytetyön puitteissa toteutettiin testiautomaation aloituspalikat eli ensimmäiset testit ja avainsanat, joilla testattiin sovelluksen toiminnallisuutta selaimen kautta. Ennen sovelluksen julkaisemista kehitys-, staging- tai tuotantoversioksi tuotteidenhallintasovelluksen täytyy läpäistä kaikki sille määritellyt testit, jotka ajetaan aina automaattisesti, kun kehittäjä vie lähdekoodiin muutoksen. Tuotettu testiautomaatio toimii tämän automaattisesti tehtävän testijärjestelmän ulkopuolella. UI-testiautomaatio voidaan ajaa erikseen milloin vain, kun sovellus on käytettävissä. UI-testiautomaatio toimii viimeisenä testiautomaatiotasona. Se varmistaa, että verkkosovellus näyttää ja päivittää oikeat tiedot tarkkailun alla olevalle sivulle.

Aiemmin Piceasoft Oy:n tuotteidenhallintasovelluksen UI-testit on ajettu vain manuaalisesti, jolloin testaaja itse käy läpi sovelluksen toimintoja käyttöliittymän kautta. Tämä auttaa löytämään uusia ohjelmistovirheitä, mutta perustoiminnallisuuksien kohdalla manuaalitestauksen luotettavuus kärsii. Useasti toistettu testi tekee sokeaksi mahdollisille uusille virheille. Testiautomaatio ei kärsi toistamisen

aiheuttamasta sokeudesta, vaan tekee määrätyt asiat aina samalla tavalla ja samassa järjestyksessä.

2 TESTIAUTOMAATIO

Mike Cohn esittelee testiautomaation eri osat pyramidin muodossa (kuva 3). Siinä testiautomaatio on jaettu kolmeen eri osaan: unit- eli yksikkötestit, service- eli palvelutestit (esimerkiksi integraatio- ja API-testit) ja UI- eli käyttöliittymätestit. Pyramidin tarkoitus on hahmottaa, kuinka paljon mitäkin testikerroksen testejä olisi hyvä olla.



KUVA 3. Testiautomaatiopyramidi (Cohn 2009)

Tätä samaa suurpiirteistä jakoa käyttää Ilpo Paju (2020), joka Testiautomaation 7 kuolemansyntiä -webinaarissa mainitsee, että käyttöliittymätestien olisi hyvä olla vain 15-20% kaikista järjestelmän testeistä. Käyttöliittymätestaus on kalleinta ja hitainta automatisoida, joten järjestelmän toiminta on parempi varmistaa mahdollisimman kattavasti jo ennen UI-kerrokselle pääsemistä.

Automatisoitujen testien ei ole tarkoitus löytää mahdollisimman monta bugia, vaan antaa nopeasti palautetta, toimiiko testattava järjestelmä odotetusti (Axelrod, 2018). Ajamalla samat testit aina samalla tavalla voidaan olla varmoja, että uudet ilmenevät ongelmat johtuvat yleensä jostain odottamattomasta regressiosta. Tähän on hyvä pyrkiä automaatiossa: luo luotettavat ja nopeat testit, jotka antavat varoituksia oikeissa ongelmatilanteissa. Tällainen tilanne on tosin verrattain utopistinen etenkin useamman hengen projekteissa, koska regressiota voi tapahtua miltä suunnalta tahansa.

2.1 Testiautomaatio vs. manuaalitestaus

Vaikka testiautomaatio toteutetaan, se ei vähennä manuaalitestauksen tärkeyttä. Etenkin käyttöliittymän testauksessa testiautomaatio ei koskaan voi tehdä manuaalitestauksesta tarpeetonta. Ohjelmoitu robotti toimii juuri niin kuin se on ohjelmoitu, niin hyvässä kuin huonossa. Robotti voi esimerkiksi löytää elementin sivulta, joka ei kuitenkaan ole oikeasti näkyvissä sivulla ja keskustella tämän kanssa ennenaikaisesti. Sovelluksen graafisuuteen kantaaottavia testejä ei myöskään ole mielekästä antaa testiautomaation tehtäväksi, koska robotti on pikselintarkka elementtien sijainnista. Manuaalitestaaja osaa robottia viisaammin sanoa, onko sivun käytettävyys kunnossa. Testiautomaatio tekee juuri niin kuin on käsketty; mutta manuaalitestaaja tietää järjestelmän oikean käytön rajat.

Testiautomaation tekijän on hyvä olla myös ohjelmoija. Hyvän testaajan koulutus ohjelmoijaksi voi olla vaikeaa, ja pahimmassa tapauksessa hyvä testaaja vaihdetaan huonoksi ohjelmoijaksi. (Graham & Fewster 2012.) Paju (2020) sanoo, että testiautomaatiokoodia pitää kohdella kuten tuotantokoodia. Ohjelmointitausta testiautomaation tekoon auttaa, koska testiautomaatiokoodin hyvät käytännöt peilaavat hyvän ohjelmistokoodin hyviä käytäntöjä: uudelleenkäytettävyys, helppolukuisuus ja modulaarisuus näistä muutamia mainitakseni.

Manuaalitestaus on etenkin käyttöliittymän testiautomaatiossa hyvä virhetilanteiden tarkempaan tutkimiseen. Välillä testiautomaation virheviestistä huolimatta voi olla vaikea määritellä, onko vika testissä vai järjestelmässä. Yleensä manuaalitestaus antaa tähän vastauksen: jos testi antaa saman virheen manuaalisesti, voidaan keskittyä korjaamaan virhettä sovelluksessa. Jos virhe havaitaan testiautomaation testissä, on hyvä heti korjata testi taas toimivaksi.

2.2 Testiautomaatiossa huomioitavaa

Testiautomaatio on myös tuotantokoodia. Se on yhtä tärkeää kuin tuotantokoodi, ja epäonnistuneiden testien kohdalla on heti alettava selvittämään, miksi testitapaus on epäonnistunut. (Paju 2020.) Tämä auttaa pitämään testitapaukset luo-

tettavina ja ajanmukaisina; toimiva testiautomaatio vaatii jatkuvaa ajamista ja päivittämistä, jotta se pysyy relevanttina järjestelmän testaamiseen. Jos testiautomaatio jää jälkeen kehityksestä, on sen uudelleen toimivaksi päivittäminen sitä hankalampaa, mitä myöhemmäksi päivitys jätetään. Pahimmassa tapauksessa tässä kohtaa testiautomaatio nähdään liikaa aikaa vievänä ja turhana, ja se jätetään vanhentumaan (kunnes, mahdollisesti, testiautomaatiota yritetään uudelleen).

Ohjelmistotestauksen veteraanit Dorothy Graham ja Mark Fewster (2012) mainitsevat havaintojensa perusteella onnistuneelle testiautomaatiolle kaksi tärkeää tekijää:

- Johto tukee automaation tekoa esimerkiksi realististen tavoitteiden sekä tarvittavien resurssien muodossa
- Oikeantasoisen abstraktion löytäminen automaation teossa, jotta kehitys on tarpeeksi mukautuvaa; vähentäen näin testiautomaation kulujen määrää

Etenkin ensimmäinen kohta on tärkeä osa onnistuneen automaation luonnissa: jos yrityksen johto ei näe automaation arvoa tai varaa sitä varten liian vähän henkilötyötunteja ja muita resursseja, ei automaatio voi päästä sille asetettuihin tavoitteisiin. Esimerkiksi hyvä tavoite manuaalitestaukselle on mahdollisimman monen ohjelmistovirheen löytäminen, mutta tämä sama ei päde välttämättä testiautomaatioon: jos automatisoidaan olemassa olevia regressiotestejä, ne tulevat harvoin löytämään uusia bugeja. (Graham & Fewster 2012.) Testiautomaation teko ei ole halpaa eikä se tuo juuri säästöjä, vaan se pitää nähdä investointina: asiakas huomaa heti, jos sovellus ei toimi odotetusti. Laadun ja luoton arvoa ei voi mitata vain rahalla.

Jotta testiautomaatiosta on mahdollisimman paljon hyötyä, on hyvä pyrkiä tekemään käytetystä koodista mahdollisimman modulaarista ja uudelleenkäytettävää. Ei myöskään kannata ahmaista liian isoa toiminnallisuutta yhden testin sisään, vaan mahdollisuuksien mukaan testata yhtä asiaa kerrallaan. Ei siis ole mielekästä yhden testin aikana esimerkiksi luoda ja muokata asiakasta, vaan

erottaa nämä toiminnot omiksi testitapauksikseen. Näin on helpompi selvittää, missä kohtaa testiautomaatio epäonnistuu ja vian tutkiminen helpottuu.

Erityisesti UI-testiautomaatiossa on huomioitava, että sen automatisaatio hajoaa valitettavan helposti. Verkkosovelluksen elementit ovat ehkä hävinneet päivityksen yhteydessä, tai verkkosivu saattaa ladata elementtinsä eri järjestyksessä testauskerrasta riippuen. Tämä ei silti tarkoita sitä, etteikö UI-testiautomaatiota kannattaisi tehdä. API-testit, yksikkötestit, integraatiotestit ja käyttöliittymätestit voivat sisältää päällekkäisiä ja samoja asioita testaavia testejä. Testit kuitenkin toimivat järjestelmän eri tasoilla, ja tieto ei välttämättä kulje tasolta toiselle odotetunlaisesti. Vaikka yksikkötestit onnistuvat, voi verkkosovelluksen käyttöliittymä näyttää vahingossa väärää tietoa. Vaikka API-testit toimivat, saattaa järjestelmä tehdä väärän API-kutsun jossain ja käytännössä järjestelmän laatu ei ole hyvä. Eri testitasot täydentävät ja palvelevat näin toisiaan.

2.3 Miksi Robot Framework?

Työkaluja testiautomaatioon on monia, niin kaupallisia kuin avoimen lähdekoodin projekteja. Avoimen lähdekoodin työkaluista verkkosovellusten testiautomaatioon tunnetuimpia ovat Cypress ja Selenium, joista Selenium on mahdollista tuoda kirjastona Robot Frameworkin käyttöön. Toimeksiantajani Piceasoft Oy ehdotti minulle Robot Frameworkin käyttöä ja siihen tutustumista, mutta saadakseni kattavamman kuvan testiautomaatiotyökaluista kokeilin sekä Robot Framework + Selenium-yhdistelmää että Cypressiä ennen lopullisen työkalupäätöksen tekoa. Testityökalun vaihtaminen kesken projektin on lähes mahdotonta.

Tutustuin Robot Framework + Selenium-yhdistelmään ja Cypressiin käytännössä tekemällä testitapauksen verkkosivulle sisäänkirjautumiseen. Tähän liittyi lisäksi dokumentaation lukemista ja videotutoriaalien katsomista.

Työkalujen testaamisen tuloksena päädyin Robot Frameworkiin. Tuetut selaimet olivat siinä monipuolisemmat kuin Cypressissä, ja oppimiskynnys oli pienempi. Pidin myös Robot Frameworkin dokumentaatiosta enemmän, sillä se oli yksinkertaisempaa ja napakammin kirjoitettua. Cypress ei silti ole huono vaihtoehto

testaamiseen. Esimerkiksi Cypressin historia-näkymä on monipuolisempi, kuin Robot Frameworkin lokit. Se näytti askel askeleelta, mitä testiajo teki missäkin kohtaa. Tulosten tarkastelu tuntui näin interaktiivisemmalta, ja mahdollisten virheklikkausten löytäminen oli paljon nopeampaa; Cypress korosti aina värillä sen elementin, jonka kanssa se sillä hetkellä keskusteli. Robot Frameworkin lokeissa virhetilanteet löytyvät kuvakaappauksen kera, ja virheen ymmärtäminen vaatii yleensä laajempaa tuntemusta toteutetusta testiautomaatiosta (tai testauksen alla olevasta sovelluksesta).

Taulukossa 1 on tiivistettynä löydökseni Robot Framework + Selenium-yhdistelmän ja Cypressin välisistä eroista. Löydösten lisäksi Robot Framework sai lisäpisteitä siitä, että sitä käytettiin jo Piceasoft Oy:lla mobiilitestiautomaatiossa Appium-nimisen kirjaston kanssa. Näin minun oli helpompi saada opastusta Robot Frameworkiin, mikä osaltaan pienensi oppimiskynnystä verrattuna Cypressiin.

TAULUKKO 1. Vertailu Robot Framework + Seleniumin ja Cypressin eroista testiautomaatiossa

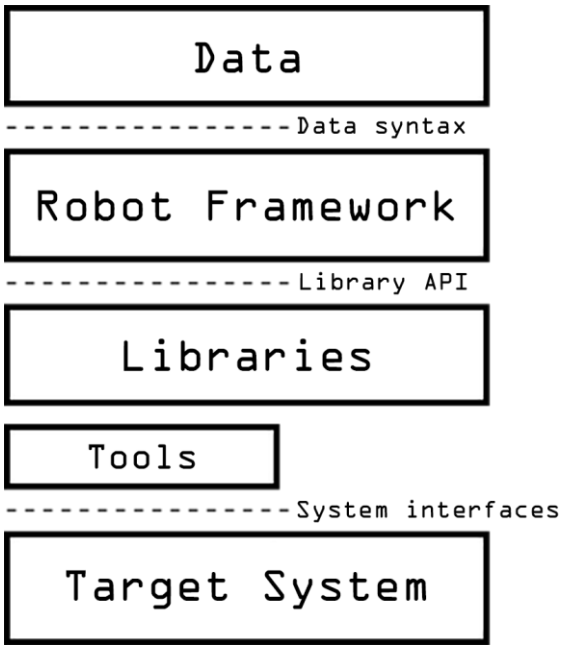
	Robot Framework + Selenium	Cypress
Tuetut selaimet	Chrome, Edge, Firefox, Safari, Opera, Internet Explorer	Chrome, Edge, Firefox
Dokumentaatio	Kattava, yksinkertainen. Hyviä esimerkkejä.	Kattava, paljon esimerkkejä. Vaikea välillä löytää mitä etsii.
Oppimiskynnys	Nopea oppia, vaikka ei osaisi ohjelmoida.	Vaatii ohjelmointitaitoja, JavaScript-pohjainen.
Raportointi, tulosten tarkastelu	Generoi automaattisesti lokin ja raportin. Verkkosovelluksen virhetilanteissa (esimerkiksi epäonnistunut klikkaus) Selenium ottaa kuvakaappauksen lokiin.	Lokissa on historia-toiminto, jonka avulla testin jokaisen askeleen tarkastelu on helppoa. Historia esimerkiksi näyttää graafisesti, mitä selaimessa on klikattu.
Sopivuus UI-testaukseen	Elementtien käsittely helppoa. Testiautomaation aloitus onnistuu nopeasti, ja sillä päästään suoraan käsiksi käyttöliittymään.	UI-testaus onnistuu, on tarkoitettu enemmän end-to-end testaukseen (eli tarkistetaan, että tieto valuu järjestelmässä kuten pitää).

3 ROBOT FRAMEWORK

Robot Framework on hyväksymistestivetoista testausta varten kehitetty avainsanapohjainen yleisluontoinen testauskehys. Se on alun perin ollut Pekka Klärckin diplomityö, mutta on sittemmin laajentunut avoimen lähdekoodin projektiksi. Robot Frameworkin kehitystä edesauttaa voittoa tavoittelematon järjestö Robot Framework Foundation. Se järjestää esimerkiksi jokavuotisen RoboCon-konferenssin, jossa alan ammattilaiset pääsevät kertomaan omista kokemuksistaan Robot Frameworkin kanssa. Konferenssissa luodaan katsaus Robot Frameworkin kehitykseen viimeisen vuoden aikana sekä tulevaisuudessa.

3.1 Robot Frameworkin rakenne

Robot Frameworkin rakenne on seuraava: datakerroksessa Robot Framework ottaa annetut tiedostot ja muuttaa ne testitapauksiksi. Robot Framework-kerros laajenee kirjastoilla ja työkaluilla, jotka ovat joko käyttäjän itse tekemiä tai Robot Frameworkiin ulkopuolelta tuotuja. ”Target system” eli kohdejärjestelmä on automaation kohteena oleva sovellus tai muu järjestelmä. Kuva 4 on Robot Frameworkin sivuilta lainattu tiivistelmä Robot Frameworkin rakenteesta.



KUVA 4. Robot Frameworkin arkkitehtuuri (Robot Framework n.d.)

Koska Robot Frameworkia on mahdollista laajentaa lisäämällä sen kylkeen erilaisia kirjastoja, taipuu se niin verkkosovellusten kuin mobiilisovellusten testiautomaatioon. Kirjastoja on mahdollista myös tehdä itse tai laajentaa käyttämällä Pythonia (jolla Robot Framework on itsekin toteutettu) tai Javaa (Robot Framework User Guide 2020).

Robot Frameworkin syntaksi on selkokielistä. Avainsanat erotellaan niille annettavista muuttujista vähintään kahdella välilyönnillä, mutta useampi välilyönti tekee testeistä luettavampia. Avainsanojen erotus muuttujista on mahdollista myös |-merkillä. Käyttäjän on mahdollista tehdä omia avainsanoja yhdistämällä valmiiden kirjastojen avainsanoja. Omat avainsanat parantavat entuudestaan automaation helppolukuisuutta ja ovat joskus jopa välttämättömiä. Esimerkiksi ”Wait Until Keyword Succeeds”-avainsana ottaa vain yhden avainsanan argumentiksi, mutta tämä rajoite voidaan kiertää juurikin luomalla oma, useamman avainsanan yhdistävä avainsana.

Robot Framework tukee avainsanan jakamista usealle riville. Esimerkiksi ”Wait Until Keyword Succeeds”-avainsana venyy helposti yli sadan merkin pituiseksi, jolloin sen lukeminen vaikeutuu. Avainsanan voi pätkiä käyttämällä uudella rivillä kolmea pistettä ja kahta välilyöntiä ennen avainsanan jatkamista normaalisti. Kuvassa 5 ”Wait Until Keyword Succeeds”-avainsana on kirjoitettu yhteen putkeen, kun taas kuvassa 6 avainsana on jaettu kahdelle riville.

```
3    Wait Until Keyword Succeeds  ${retry_time}  ${retry_interval_time}  Attempt Navigation To Page  class=a-sub-customers  Add new customer
```

KUVA 5. Pitkä avainsana ilman rivinjakoa

```
3    Wait Until Keyword Succeeds      ${retry_time}      ${retry_interval_time}
4    ...  Attempt Navigation To Page      class=a-sub-customers      Add new customer
```

KUVA 6. Pitkä avainsana rivinjaon kanssa

3.2 Robot Framework testauksessa

Robot Framework on monipuolinen testauskehys, jonka laajennus tapahtuu käyttäjien tekemien kirjastojen avulla. Tunnetuimpia näistä ovat verkkosovellusten

testaukseen tarkoitettu SeleniumLibrary ja mobiilisovellusten testaukseen tarkoitettu AppiumLibrary. Robot Frameworkin mukana tulee myös yleiskirjastoja (esimerkiksi BuiltIn-kirjasto), jotka ovat aina automaattisesti mukana kaikissa Robot Framework projekteissa. BuiltIn-kirjasto sisältää yleisluontoisia avainsanoja esimerkiksi toisien avainsanojen ajamiseen liittyen. Toimeksiantajalleni toimitetussa testiautomaatioprojektissa paljon käytetty ”Wait Until Keyword Succeeds”-avainsana on esimerkiksi BuiltIn-kirjastosta.

Test suite eli testisarja jaottelee testit testitapausten kansion, tiedostojen ja tagien mukaan. Ajon aikana testit ajetaan ylhäältä alaspäin järjestyksessä. Luvussa 4.4 on tarkemmin tietoa, miten testien ajo tapahtuu ja miten ajettuja testejä voidaan määritellä Robot Frameworkin ajamiskomennolla.

Robot Framework-tiedosto voi esimerkiksi näyttää tältä:

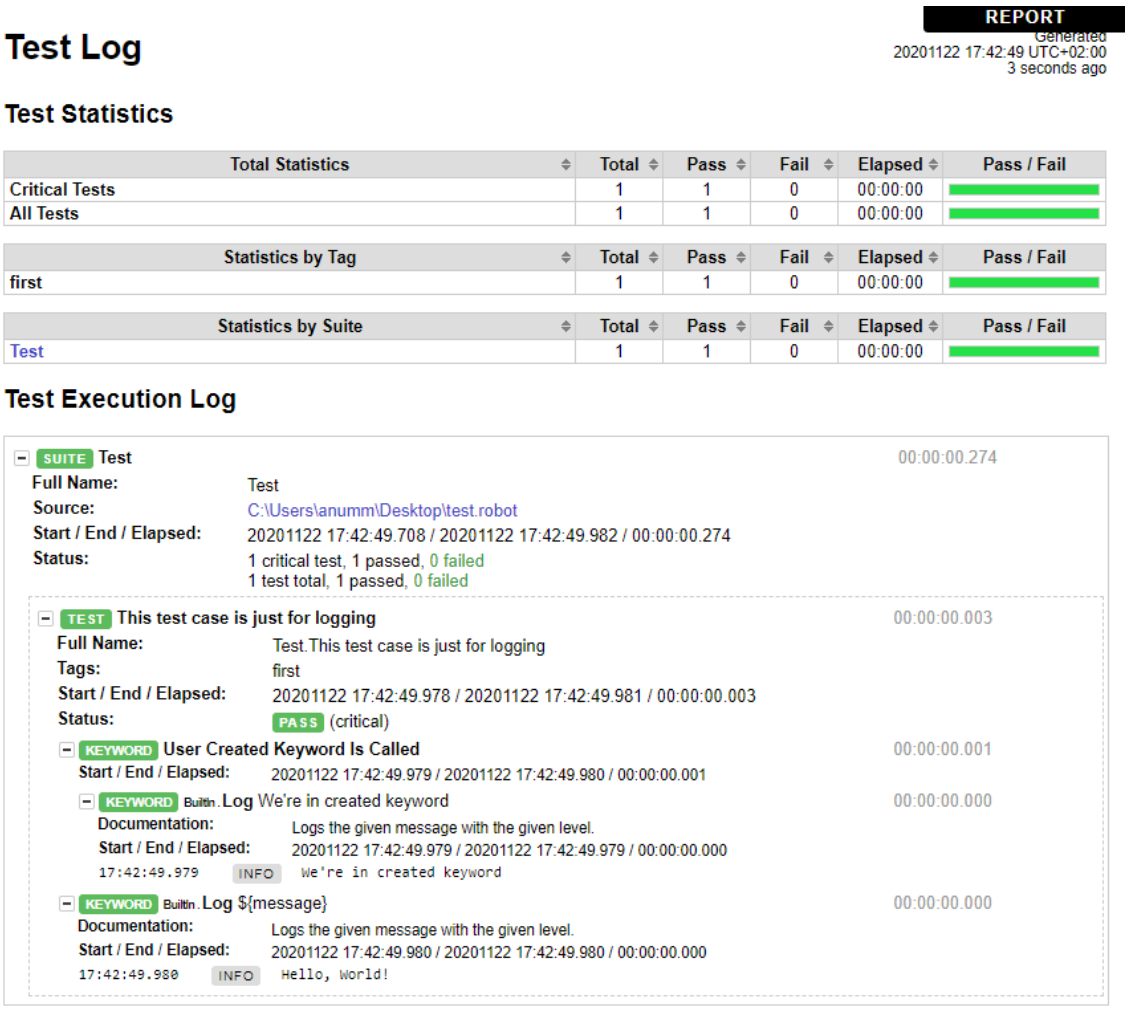
```
*** Settings ***
    Library                 SeleniumLibrary
    Force Tags              first

*** Keywords ***
    User Created Keyword Is Called
        Log                 We’re in created keyword

*** Variables ***
    ${message}              Hello, World!

*** Test Cases ***
    This test case is just for logging
        User Created Keyword Is Called
        Log       ${message}
```

Settings- eli asetukset-kohdassa määritellään esimerkiksi, mitä kirjastoja testitiedostoon tuodaan sekä millaisilla tageilla testisarjan testit merkataan. Tageilla voidaan määritellä, mitä testejä ajetaan yhdessä. Asetuksissa voidaan myös määritellä testisarjan aloitukseen ja lopettamiseen liittyviä avainsanoja. Testisarjan alkuun (Suite Setup) olisi luonnollista laittaa esimerkiksi verkkosivulle sisäänkirjautuminen ja testisarjan loppuun (Suite Teardown) uloskirjautuminen sekä testiselainten sulkeminen.

Keywords- eli avainsanat-kohdassa voidaan määritellä omia avainsanoja, joita voidaan kutsua testitapauksissa. Avainsanat voidaan myös tuoda tiedostoon asetukset-kohdassa Resource- eli resurssi-komennon avulla.
Variables- eli muuttujat-kohtaan on mahdollista kirjata ylös kaikki testitapauksissa käytetyt yleismuuttujat. Jotta testiautomaatiota on helpompi käyttää, Robot Framework mahdollistaa testitapausten muuttujien määrittelyn erillisissä tiedostoissa, jotka voidaan lisätä sitten testitapaustiedostoon. Muuttujien ulkoisessa tiedostossa määrittämisen päätarkoitus on noudattaa DRY (Älä toista itseäsi) -periaatetta päällekkäisyyksien minimoimiseksi sekä muuttujien muuttamiseksi yhdessä paikassa muuttamatta lopputestiä (Bisht 2013, 2.).

Test Cases- eli testitapaukset-kohdassa ovat ajettavat testit. Robot Framework suorittaa testitapaukset ja niiden sisällä olevat avainsanat järjestyksessä ylhäältä alaspäin. Tämä on hyvä ottaa huomioon, kun testitapauksia luodaan useampia yhdessä tiedostossa, jottei aiemman testitapauksen suoritus tee seuraavasta testitapauksesta mahdotonta. Jos esimerkiksi ensin luomme asiakkaalle tuotteen ja poistamme sen seuraavassa testitapauksessa, on kolmannen testitapauksen mahdotonta muokata luodun tuotteen tietoja (koska sitä ei ole enää olemassa).

Tulostuksena esimerkin testitapauksesta tulisi lokiin kaksi viestiä: ensin ”We’re in created keyword” ja sen jälkeen ”Hello, World!”. Kuvassa 7 on kuvakaappaus esimerkkitestitapauksen loki-tiedostosta testiajon jälkeen. Testilokin sisältö käydään tarkemmin läpi tämän opinnäytetyön kappaleessa 4.5.



Test Log

REPORT
Generated
20201122 17:42:49 UTC+02:00
3 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:00	
All Tests	1	1	0	00:00:00	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
first	1	1	0	00:00:00	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Test	1	1	0	00:00:00	

Test Execution Log

SUITE Test 00:00:00.274
Full Name: Test
Source: C:\Users\anumm\Desktop\test.robot
Start / End / Elapsed: 20201122 17:42:49.708 / 20201122 17:42:49.982 / 00:00:00.274
Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

TEST This test case is just for logging 00:00:00.003
Full Name: Test.This test case is just for logging
Tags: first
Start / End / Elapsed: 20201122 17:42:49.978 / 20201122 17:42:49.981 / 00:00:00.003
Status: PASS (critical)

KEYWORD User Created Keyword Is Called 00:00:00.001
Start / End / Elapsed: 20201122 17:42:49.979 / 20201122 17:42:49.980 / 00:00:00.001

KEYWORD BuiltIn . Log We're in created keyword 00:00:00.000
Documentation: Logs the given message with the given level.
Start / End / Elapsed: 20201122 17:42:49.979 / 20201122 17:42:49.979 / 00:00:00.000
17:42:49.979 INFO We're in created keyword

KEYWORD BuiltIn . Log ${message} 00:00:00.000
Documentation: Logs the given message with the given level.
Start / End / Elapsed: 20201122 17:42:49.980 / 20201122 17:42:49.980 / 00:00:00.000
17:42:49.980 INFO Hello, World!

KUVA 7. Kuvakaappaus esimerkkitestitapauksen testilokista

3.3 Robot Framework ohjelmistorobotiikassa

Robot Framework on enemmän kuin pelkkä testauskehys. Sillä on mahdollista luoda myös Robotic Process Automation (RPA) eli ohjelmistorobotiikkatehtäviä. Ohjelmistorobotiikka tarkoittaa useasti tehtävien ja puuduttavien tehtävien automatisointia. Opinnäytetyön testiautomaatioprojektissa RPA:ta käytettiin testiajon aikana luotujen tuotteiden ja aliasiakkaiden poistamiseen, jotta testiautomaation käyttämää testiasiakasta on helpompi käsitellä.

Ohjelmistorobotiikan luonti Robot Frameworkilla toimii hyvin samalla tavalla kuin testitapausten luonti. Käytännössä ero käyttäjälle näkyy siinä, että testitapauksia kutsutaankin nimellä tehtävä (Task). Ajonjälkeisissä lokeissa puhutaan myös tehtävistä siinä missä testiautomaation lokit viittaavat testitapauksiin.

3.4 Selenium

Selenium on avoimen lähdekoodin projekti, joka sisältää työkaluja ja kirjastoja verkkosovellusten automatisointiin. Näistä Selenium WebDriver on käytännössä paketti verkkosovellusten automatisointiin, mutta selkeyden vuoksi siihen referoidaan tässä opinnäytetyössä pelkkänä Seleniumina. Seleniumin kehitys on alkanut vuonna 2004, ja se on yksi käytetyimmistä verkkosovellusten automaation työkaluista. Applitoolsin (2020) asiakkailleen pitämän kyselyn mukaan 75% heidän käyttäjistään käyttää Selenium WebDriveria verkkosovellusten automaatioon.

Selenium on mahdollista tuoda Robot Frameworkiin Robot Frameworkia varten tehdyn SeleniumLibrary-kirjaston avulla. Tämä kirjasto on kääntänyt Seleniumin avainsanapohjaiseksi, jolloin esimerkiksi elementin klikkaus tapahtuu komennolla

```
Click Element          id=my-id
```

Vastaavasti ”puhtaalla” Seleniumilla sama komento olisi näin:

```
webDriver.FindElement(By.Id("my-id")).Click();
```

Selenium mahdollistaa verkkosivun elementtien klikkaamisen, keskittämisen ja tarkistamisen monella eri tapaa. Tapoja ovat elementin XPath, identifikaattori (=id) ja luokkanimi (=class). XPath on valintatavoista monipuolisin. Elementin id toimii yleensä hyvin, mutta joskus Selenium katsoo jonkun toisen elementin olevan sen edessä. Tällöin oikean elementin kanssa kommunikointi tapahtuu hieman kiertäen XPathin avulla.

Esimerkiksi Piceasoftin tuotteille määritellään tuotteidenhallintajärjestelmässä toimintoja käyttämällä input-elementtejä hyödyksi. Toimintoja voidaan poistaa ja lisätä tuotteesta klikkaamalla toiminnon input-elementtiä. Tällaisen input-elementin edessä on kuitenkin Seleniumin mukaan label-elementti, joka antaa input-elementille tekstiselityksen. Testitapaus kaatuu tähän virheilmoituksen kanssa: “Elementtiä ei voitu klikata, koska klikkauksen olisi ottanut vastaan toinen elementti”.

Tämä ongelma kierrettiin antamalla ensin input-elementille keskitys elementin id:n mukaan, jonka jälkeen Selenium painaa välilyönti-näppäintä. Käytännössä input-elementti toimii tällöin samalla tavalla kuin klikatessa, mutta pääsemme välttämään Seleniumin virheilmoituksen.

Eli käytännössä koodinpätkä

```
Click Element         id=input
```

jouduttiin muokkaamaan muotoon

```
Set Focus To Element         id=input
Press Keys                   None                  SPACE
```

Kahden tällaisen toiminnon ajaminen ei käytännössä ole sen hitaampaa kuin yhden klikkaus-avainsanan ajo, joten ajoittaiset kiertoilmaukset toisille avainsanoille eivät vaikuta testien ajon nopeuteen.

4 TESTIAUTOMAATION ALOITUS ROBOT FRAMEWORKILLA

Robot Frameworkin käyttö alkaa luonnollisesti asentamisella. Opinnäytetyössä toteutetun testiautomaation teko tapahtui Windows 10 -käyttöjärjestelmällä, joten luku 4.1 keskittyy Robot Frameworkin ja sen tarvitsemien lisäosien asentamiseen Windows 10:lle.

4.1 Asennus Windows 10:lle

Koska Robot Framework on kirjoitettu Pythonilla, on sen asennus helpointa käyttämällä Pythonin paketinhallintasovellus pip:iä. Robot Framework tukee Python-kielen versiota 3 Robot Frameworkin versiosta 3.0 eteenpäin, joten Robot Frameworkin asennus kannattaa aloittaa Pythonin versio 3:n asennuksella. Jos koneeltasi löytyy jo Python versio 3.6 tai uudempi sekä pip, voit siirtyä vaiheeseen 4.1.2.

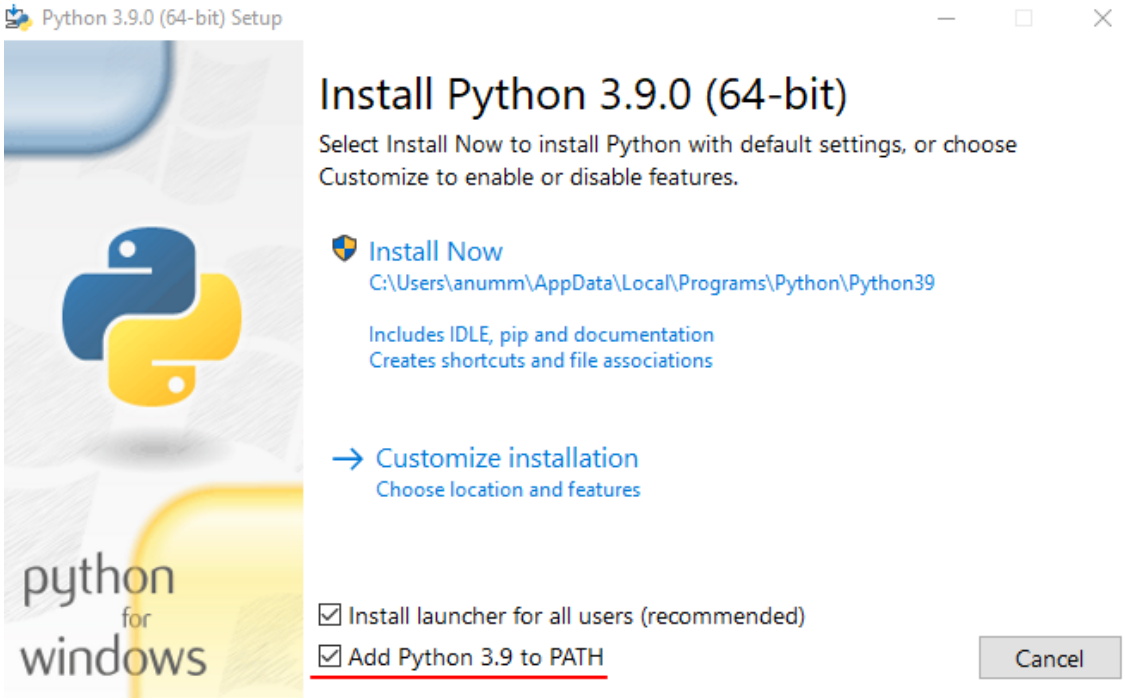
Katso kohta 4.1.5 yleisimpien asennusongelmien ratkaisemiseksi.

4.1.1 Python ja pip-paketinhallintasovellus

Tulevaisuudessa Robot Framework aikoo vaatia toimiakseen vähintään Pythonin version 3.6 (Robot Framework User Guide, 2020). Tähän on hyvä varautua, ja valita suoraan jokin sitä uudempi Pythonin versio, esimerkiksi versio 3.9 osoitteesta https://www.python.org/downloads/. (Huomio: Robot Framework on aina tähän mennessä saanut nopeasti tuettua aina uusinta Python-versiota, mutta tuen tila kannattaa tarkistaa ennen uusimpaan Python-versioon siirtymistä esimerkiksi Robot Frameworkin GitHub-julkaisuista osoitteessa https://github.com/robotframework/robotframework/releases.)

Kun Python asennustiedosto on latautunut, Pythonin asennus alkaa juuri ladatun asennustiedoston ajolla. Windowsilla kannattaa heti ensimmäisessä ruudussa

asennuksen alettua valita ”Add Python 3.9 to PATH” (kuva 8), jotta saamme Pythonin komennot käyttöömme komentoriviltä. Install Now -vaihtoehto asennuksessa lataa automaattisesti Pythonin dokumentaatiot sekä pip-paketinhallintasovelluksen, mutta asennusta on mahdollista muokata omien tarpeidensa mukaan. Oletusasetuksilla emme kuitenkaan vahingossa voi jättää jotain asentamatta, joten valitaan Install Now.



KUVA 8. Pythonin asennuksen ensimmäinen ruutu. Huomioi, että ”Add Python 3.9 to PATH”-valintaruutu on valittuna

Asennuksen valmistuttua voi sen vielä varmistaa ajamalla komentorivillä komennon

```
python --version
```

ja

```
pip --version
```

Jos kaikki on mennyt kuten pitikin, pitäisi yllä olevien komentojen palauttaa koneelle asennetun Pythonin ja pip:n versiot (kuva 9).

```
C:\Users\anumm>python --version
Python 3.9.0

C:\Users\anumm>pip --version
pip 20.2.3 from c:\users\anumm\appdata\local\programs\python\python39\lib\site-packages\pip (python 3.9)
```

KUVA 9. Python ja pip versioiden kyselyt ja tulokset.

Kun pip on asennettu, sen käyttö komentoriviltä on helppoa (Robot Framework User Guide, 2020). Käytämme pip-paketinhallintasovellusta seuraavissa vaiheissa Robot Frameworkin, SeleniumLibraryn ja WebdriverManagerin latauksseen.

4.1.2 Robot Framework

Uusimman Robot Framework-version lataamiseksi annamme komentoriviltä seuraavan komennon:

```
pip install robotframework
```

Tämä komento aloittaa Robot Frameworkin asennuksen. Asennus kertoo ladatun Robot Frameworkin version, mutta sen voi myös todentaa antamalla komennon

```
robot --version
```

Komentojen tulokset näkyvät kuvassa 10.

```
C:\Users\anumm>pip install robotframework
Collecting robotframework
  Downloading robotframework-3.2.2-py2.py3-none-any.whl (623 kB)
     |████████████████████████████████| 623 kB 1.7 MB/s
Installing collected packages: robotframework
Successfully installed robotframework-3.2.2
WARNING: You are using pip version 20.2.3; however, version 20.2.4 is available.
You should consider upgrading via the 'c:\users\anumm\appdata\local\programs\python\python39\python.exe -m pip install --upgrade pip' command.

C:\Users\anumm>robot --version
Robot Framework 3.2.2 (Python 3.9.0 on win32)
```

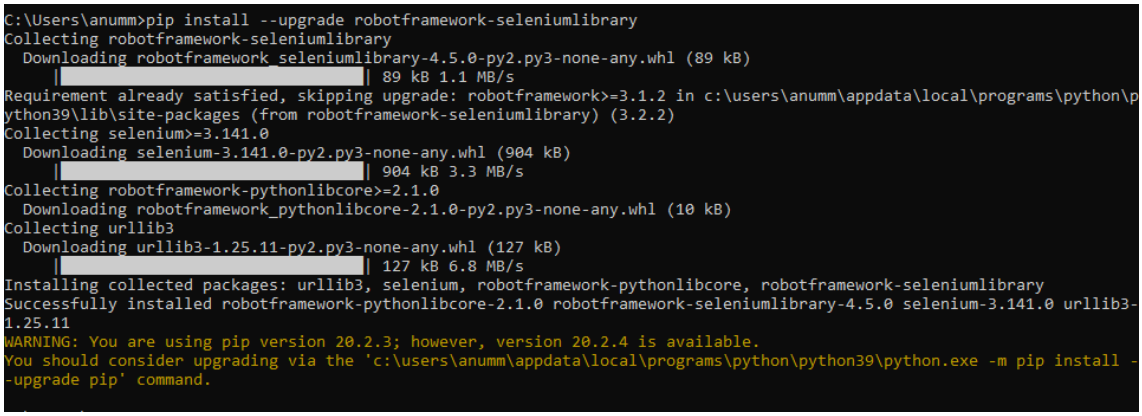
KUVA 10. Robot Frameworkin asennus komentoriviltä käyttämällä pip-paketinhallintasovellusta ja asennetun Robot Framework-version tarkistus

4.1.3 SeleniumLibrary

Verkkosovellusten automaatisoimiseen Robot Frameworkilla käytämme SeleniumLibrary-kirjastoa. Kirjastojen asennus Robot Frameworkin käyttöön on helpointa pip-paketinhallintasovelluksella. SeleniumLibraryn asentamiseksi annamme komentorivillä seuraavan komennon:

```
pip install --upgrade robotframework-seleniumlibrary
```

Tämä komento SeleniumLibraryn asentamisen päivittää lisäksi sen sekä Robot Frameworkin uusimpiin versioihin. Kuvassa 11 on esimerkki komentolinjan tulostuksesta SeleniumLibraryn asennuksen jälkeen.



KUVA 11. Komentolinjalla annetun *pip install –upgrade robotframework-seleniumlibrary*-komennon tulos

Lisätietoja SeleniumLibraryn asennuksesta löytyy osoitteesta https://robotframework.org/SeleniumLibrary/.

4.1.4 WebdriverManager

Seleniumin käyttö vaatii selainten ajureita. Ajurit on mahdollista etsiä netistä ja lisätä Windowsin PATH-muuttujiin manuaalisesti, mutta ajureiden asennus ja hallinta on kätevintä käyttämällä WebdriverManager-nimistä työkalua. WebdriverManager tukee Chromen, Firefoxin, Operan ja Edgen ajureiden lataamista (SeleniumLibrary, n.d.).

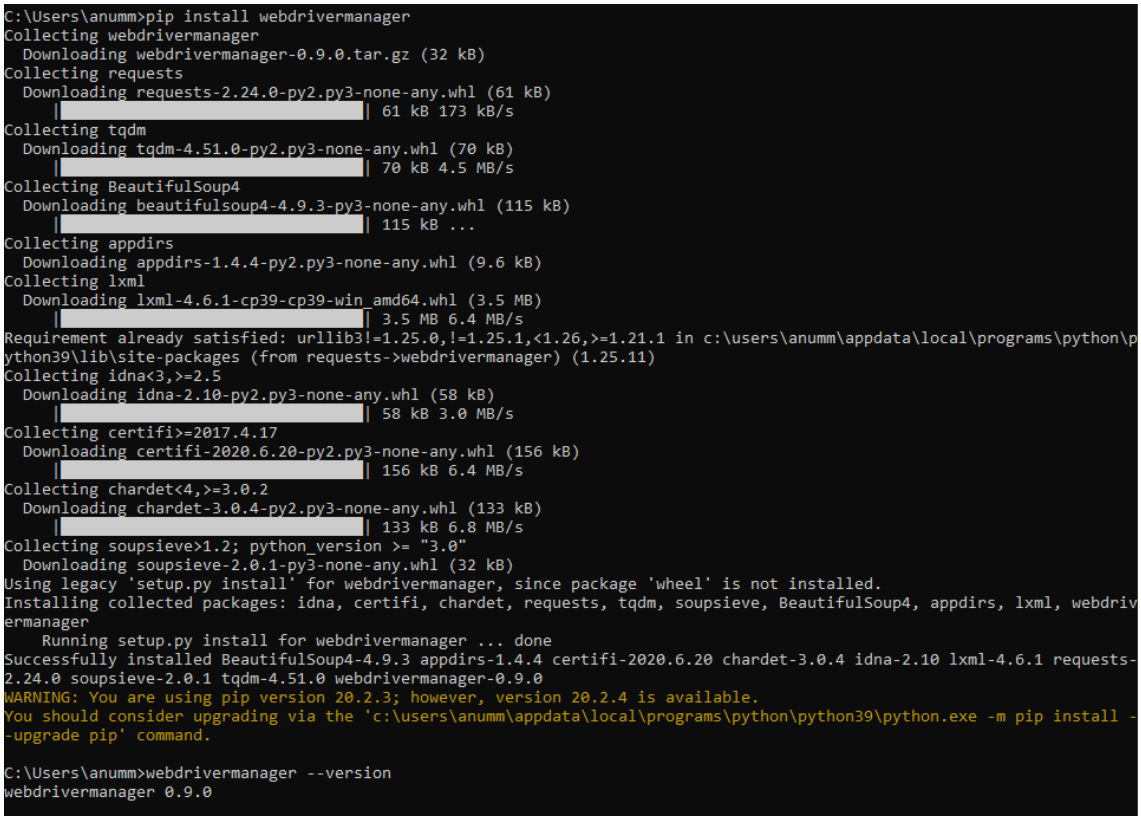
Käytämme jälleen pip-paketinhallintasovellusta. Annetaan komentorivillä komento

```
pip install webdrivermanager
```

joka lataa työkalun sekä sen käyttämät paketit. Asennuksen onnistumisen voimme tarkistaa antamalla komennon

```
webdrivermanager --version
```

Kuvassa 12 näkyy esimerkki komentorivin antamista WebdriverManagerin asennuksen ja versiokyselyn tuloksista.



KUVA 12. WebdriverManagerin asennus ja versiokysely

Viimeiseksi lataamme haluamiemme selainten ajurit WebdriverManagerilla. Tässä opinnäytetyössä keskityttiin tukemaan Chrome- ja Firefox-selaimia niiden suosion takia (ks. 5.2 Testiautomaation tavoitteet). Selainten asennus tapahtuu WebdriverManagerilla antamalla komentorivillä komennon

```
webdrivermanager chrome firefox
```

Tämä lataa chromedriver- ja geckodriver-ajurit WebdriverManagerin luomaan kansioon, joka mainitaan asennuksen aikana. Asennuspolku on mahdollista antaa lisäämällä komento *--downloadpath <POLKU>* esimerkkikomentomme perään. (WebdriverManager 2020.)

Tämä komento ei lisännyt vielä ajureita Windowsin PATH-muuttujiin, josta WebdriverManager itsekin varoittaa asennuksen aikana. Käytä tämän opinnäytetyön luvun 4.1.5 kohdan B ohjeita WebdriverManagerin varoittaman polun lisäämiseksi ympäristömuuttujiin.

4.1.5 Ongelmatilanteet asennuksessa

Jos missään asennusvaiheessa törmäät komentolinjalla tehdyn versiokyselyn yhteydessä virheeseen, ettei komentoa tunnisteta, kokeile seuraavia ohjeita:

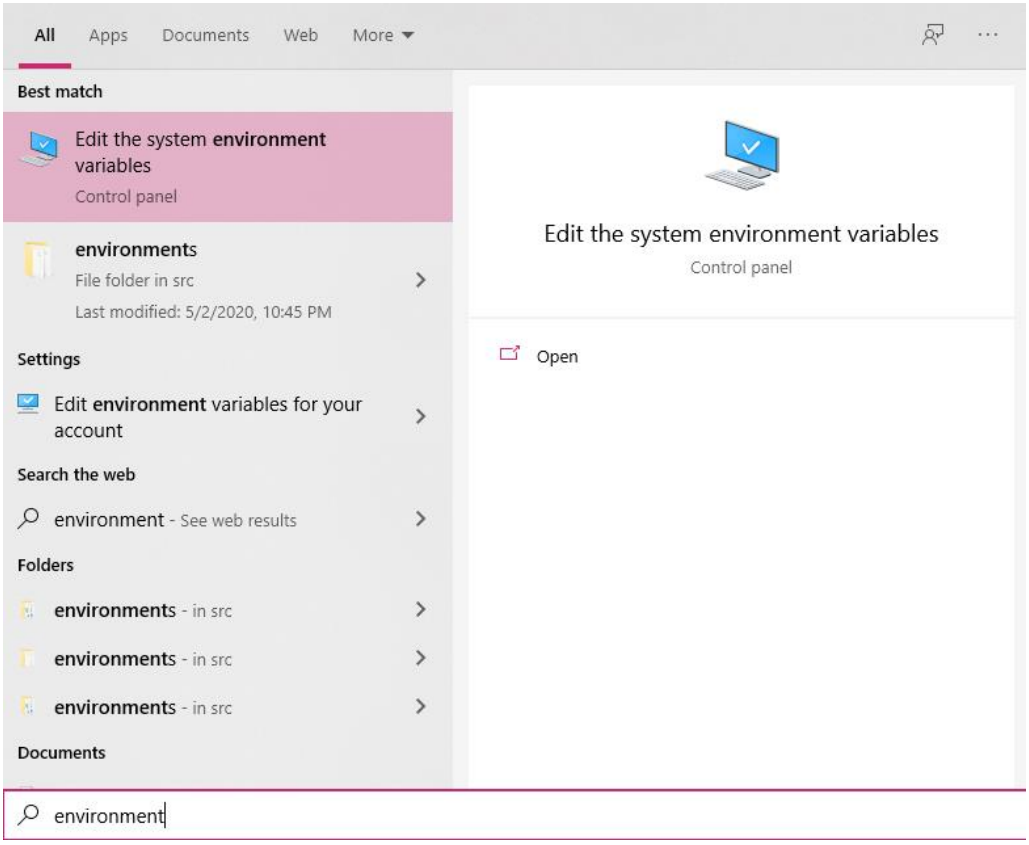
A. Uusi komentorivi

Avaa uusi komentorivi ja yritä antaa komento uusiksi. Joskus Windowsin PATH-muuttujia muokatessa komentorivi ei ole saanut tietoa muutoksista. Tähän auttaa komentorivin uudelleen avaus.

B. Lisää PATH-ympäristömuuttuja

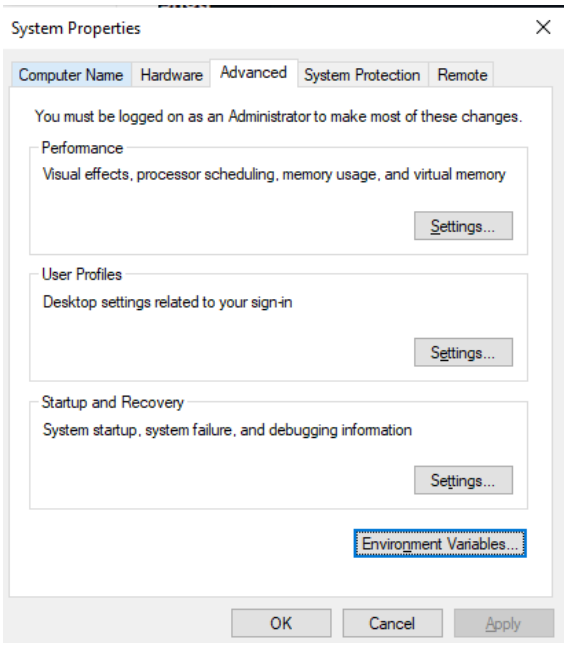
Jos komentorivin avaus ei auttanut, lisää asennuksen polku Windowsin järjestelmämuuttujien PATH-muuttujiin seuraavasti:

1. Avaa Windowsin Start-valikko (suomeksi Aloitus-valikko).
2. Kirjoittamalla ”environment” Aloitus-valikko avoinna aloittaa haun, jonka tuloksena pitäisi löytyä ”Edit the system environment variables”-valinta (katso kuva 13). Suomeksi hakusanana käytetään *ympäristö*, jonka pitäisi tarjota vaihtoehdoksi *Muokkaa järjestelmän ympäristömuuttujia*. Avaa ehdotettu ympäristömuuttujien muokkaus -ikkuna.



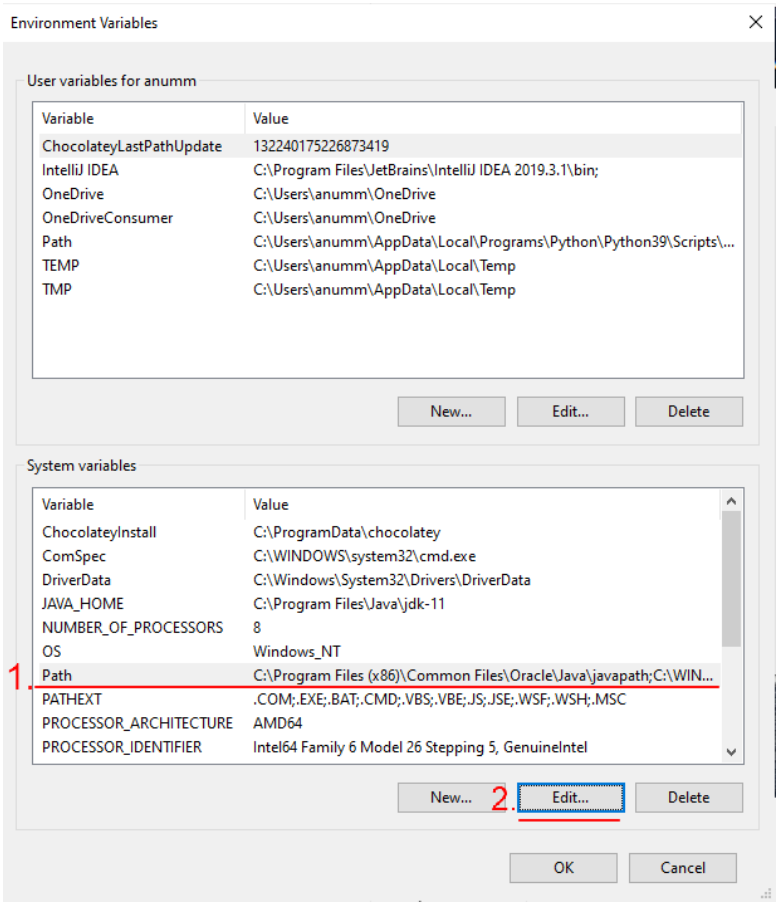
KUVA 13. Windowsilla tehdyn *environment*-haun tulos

3. Klikkaa *Environment Variables*… -nappia avataksesi ympäristömuuttujien muokkausikkunan (suomeksi *Ympäristömuuttujat*…), katso kuva 14.



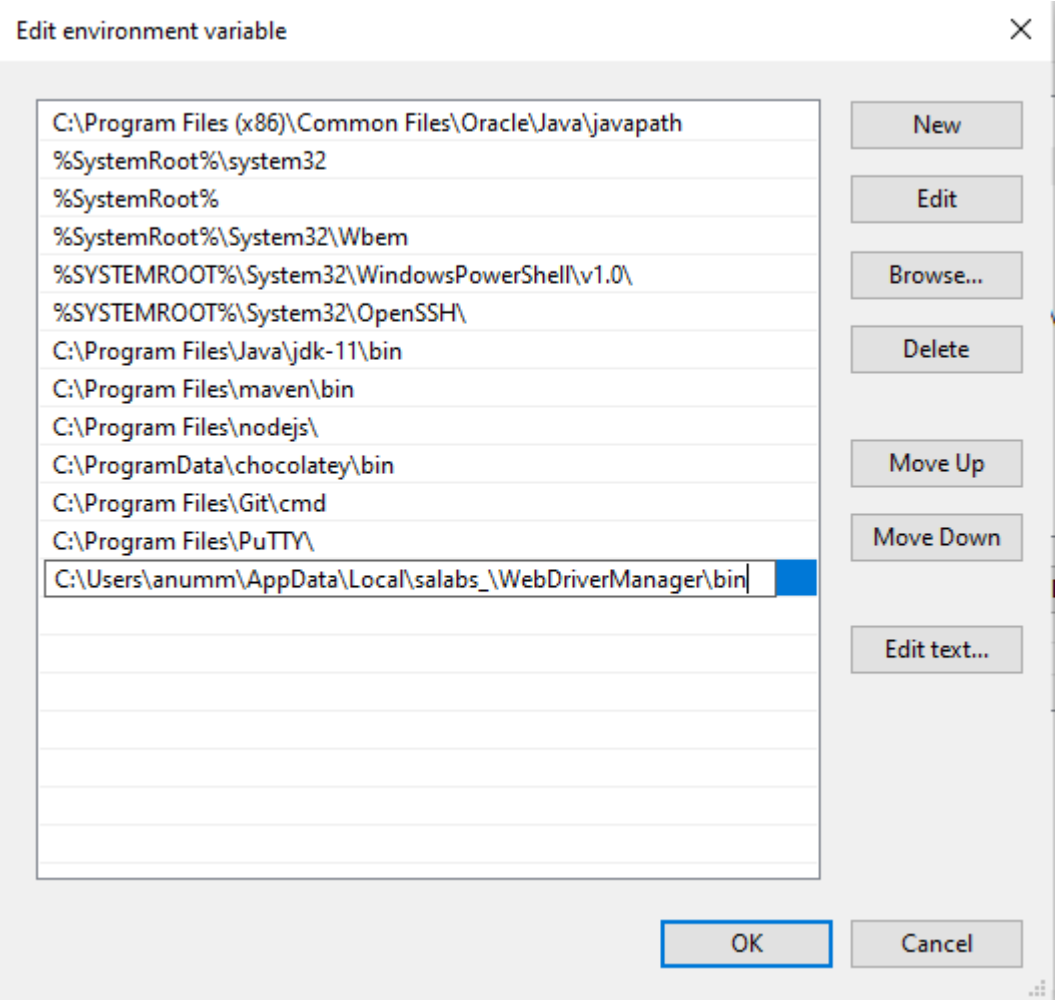
KUVA 14. Environment Variables… -napin sijainti

4. Valitse *System variables*-kohdasta PATH ja klikkaa *Edit*…-nappia (suomeksi *Järjestelmämuuttujat* ja *Muokkaa*…-nappi). Katso kuva 15. Klikataan ensin Path-riviä (merkattu numerolla 1.) ja sen jälkeen Edit...-nappia (merkattu numerolla 2.).



KUVA 15. Ympäristömuuttujat-ikkuna

5. *Edit environment variables* -ikkuna sisältää kaikki järjestelmäämme asetetut muuttujat. Klikataan kuvan 16 oikeassa yläreunassa olevaa *New*-nappia, jotta voimme lisätä uuden muuttujan.
6. Kopioi haluamasi ohjelman polku avautuneelle riville ja klikkaa OK (katso kuva 16). Tallenna muutokset kaikissa ikkunoissa klikkaamalla OK-nappia.



KUVA 16. Uuden ympäristömuuttujan lisääminen. Esimerkissä Webdriver-Managerin bin-kansion lisäys, jotta selainajureihin päästään käsiksi komentoriviltä

7. Avaa komentorivi uudelleen ja anna komento, jota ei aiemmin tunnistettu. Sen pitäisi nyt toimia.

Muista painaa OK-nappia jokaisessa auki olevassa ponnahdusikkunassa, jotta muutokset varmasti tallentuvat.

4.2 Testien suunnittelu

Testejä suunnitellessa, testien vaatimukset muutetaan testitapauksiksi (Black 2016, 1.5). Testien vaatimukset ovat monesti myös testattavan järjestelmän vaatimuksia, joten testien suunnittelijan on hyvä olla perillä testattavan järjestelmän

toiminnasta sekä sidosryhmien vaatimuksista ja odotuksista. Testitapausten kirjaaminen ylös esimerkiksi Excel-tiedostoon auttaa niiden hallitsemista ja tilan seuraamista.

Testisuunnitelmassa määritellään yleensä testien tarkoitus, esiehdot, testiympäristön vaatimukset, syötetiedot ja muut testidatavaatimukset, testien odotetut tulokset sekä testien jälkiehdot (Black 2016, 1.5). Määrittelyä ei ole pakko tehdä kirjallisesti, mutta se tekee testitapauksista yhtenäisempiä. Valmis suunnitelma toimii hyvin myös perehdytyksessä, ja siihen on helppo palata aina uuden testitapauksen luomisen yhteydessä.

Rex Blackin (2016) mukaan testitapauksia on mahdollista määritellä eri tarkkuuksin. Yleisen tason testitapaus voisi olla esimerkiksi ”Testaa tuotteen luominen”, joka jaettuna tarkempiin testitapauksiin voisi sisältää tarkemman tiedon esimerkiksi tuotetyypistä (”Testaa mobiilituotteen luominen”, ”Testaa PC-tuotteen luominen” jne.). Ylemmän tason testitapausten esittely voi sopia paremmin sidosryhmille pidettävään katselmukseen, jonka jälkeen vasta luotaisiin tarkemmat testitapaukset (Black 2016, 1.4). Tarkemmat testitapaukset ovat hyödyllisiä, koska ne auttavat testien tekijää suorittamaan vastaavan ylemmän tason testin paremmalla varmuudella.

Rex Black (2016) kehottaa dokumentoimaan testit niin hyvin, että esimerkiksi joku toinen testaaja pystyisi ajamaan määritellyt testitapaukset itsenäisesti. Robot Framework parhaimmillaan mahdollistaa testitapausten dokumentoinnin itsestään, kun testitapausten käyttämät avainsanat on nimetty kuvaavasti.

4.3 Testien toteutus

Testien kirjoitus onnistuu millä tahansa tekstinkäsittelysovelluksella, jolla pystytään tallentamaan .robot-päätteisiä tiedostoja. Käytännössä kirjoittamista helpottaa, jos kehityksen avuksi ottaa esimerkiksi Visual Studio Code -sovelluksen tai PyCharm-ohjelmointiympäristön. Näihin molempiin on mahdollista ladata lisäosana Robot Frameworkin syntaksin tunnistin, joka värjää tekstin helppolukui-

semmaksi. Kuvassa 17 on esimerkki testitapauksesta ennen syntaksintunnistimen asennusta ja kuvassa 18 on sama testitapaus syntaksintunnistimen asennuksen jälkeen.

```
1   *** Settings ***
2   Library     SeleniumLibrary
3
4   *** Variables ***
5   ${message}      Hey, this is pretty cool!
6
7   *** Test Cases ***
8   My first test case
9       Log         ${message}
10
```

KUVA 17. Testitapaus Visual Studio Codessa ilman syntaksintunnistusta

```
1   *** Settings ***
2   Library     SeleniumLibrary
3
4   *** Variables ***
5   ${message}      Hey, this is pretty cool!
6
7   *** Test Cases ***
8   My first test case
9       Log         ${message}
10
```

KUVA 18. Esimerkki syntaksin korostuksesta Visual Studio Codessa käyttäen Robot Framework Language Server -nimistä lisäosaa

4.4 Testien ajo Robot Frameworkilla

Yksinkertaisimmillaan Robot Frameworkin testiajo tapahtuu komennolla

```
robot testitiedosto.robot
```

Tämä komento ajaa kaikki testitiedosto.robot-nimisessä tiedostossa olevat testitapaukset. Tiedostonimi on mahdollista korvata pisteellä (.), jolloin Robot Framework etsii kaikki testitapaukset siitä kansiosta, jossa komento ajettiin sekä kaikista kansion alikansioista.

Useasti tämä yksinkertaisin mahdollinen testiajo ei kuitenkaan riitä. Siksi Robot Framework tarjoaa monipuolisesti komentolinjavalintoja, joiden avulla testitapausten ajo tapahtuu yksinkertaisemmin ja mielekkäämmin. Esimerkiksi lokit tulevat automaattisesti aina log.html tiedostoon. Jokainen uusi testiajo korvaa edellisen log.html:n sisällön, jolloin aiempien testiajojen tuloksia ei päästä enää tarkastelemaan. Lokien aikaleimaus on yksi ratkaisu tähän ongelmaan, ja ne saadaan lisättyä jokaisen tulostiedoston nimeen lisäämällä aiempaan komentoon aikaleima-vaihtoehdon:

```
robot --timestampoutput testitiedosto.robot
```

Näin loki ei aina tule yliajetuksi testin jälkeen, vaan jokainen testiajon loki saa nimeensä aikaleiman muodossa YYYYMMDD-hhmmss. YYYYMMDD-hhmmss formaatin YYYY merkitsee nykyisen vuoden (eli esimerkiksi 2020), MM merkitsee nykyistä kuukautta (esimerkiksi 11), DD merkitsee nykyisen päivän (esimerkiksi 04) ja viivan jälkeen tulevat hh, mm ja ss merkkaavat järjestyksessä tuntia, minuutteja ja sekunteja jolloin testiajo aloitettiin.

Oletuksena Robot Framework luo testilokin, raportin ja output.xml-tiedoston siihen kansioon, missä testiajo aloitettiin. Jotta testikansiot eivät täyty testituloksista, voidaan testitulokset ohjata omaan kansioon komennolla

```
robot --outputdir Logs testitiedosto.robot
```

Tämä komento ohjaa myös Seleniumin ottamat kuvakaappaukset Logs-nimiseen kansioon.

Jos haluamme ajaa vain tietyt testit testisarjasta, ne on mahdollista merkata erilaisin tagein. On siis mahdollista ajaa vain osa saman testitiedoston testeistä ilman, että jokaista erilaista testiajoa varten tehdään oma tiedosto. Yksi tällaisista tageista voisi olla esimerkiksi reseller eli jälleenmyyjä, jolla merkittäisiin jälleenmyyjään kohdistuvat testitapaukset. Testitapaukselle voidaan antaa monta tagia, ja se perii myös testisarjalle annetut tagit. Tällainen tagilla määrätty ajo voidaan suorittaa antamalla lisäehdon ajokomentoon:

```
robot --include reseller testitiedosto.robot
```

Huomaa, että tässä komennossa ei otettu mukaan aikaleima-vaihtoehtoa, joten testitulokset tallentuisivat oletusnimillä. Include-vaihtoehtoja on mahdollista lisätä useita, mutta ne täytyy aina erottaa omalla --include-vaihtoehdolla.

Ajokomennon lisäksi annettuja vaihtoehtoja on paljon, mutta nostan vielä yhden vaihtoehdon esille sen monipuolisuuden takia: variable.

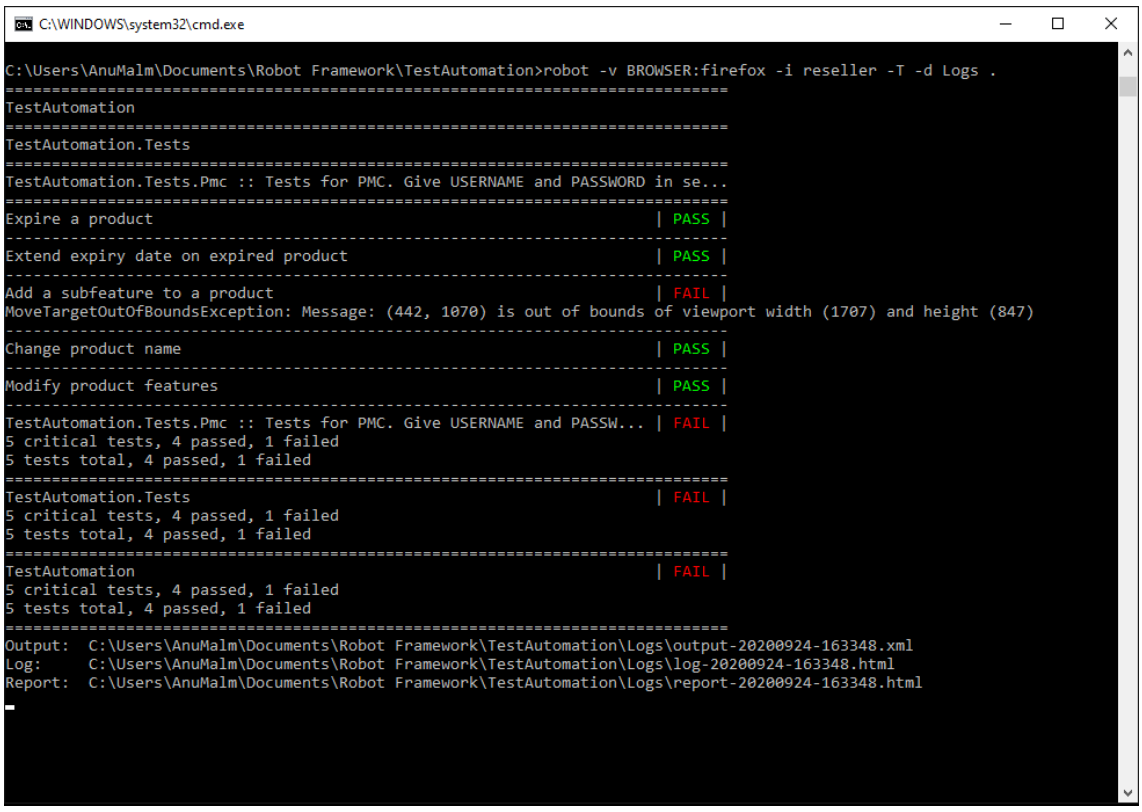
Variable-vaihtoehto ajokomentoon mahdollistaa testitapausten muuttujien määrittämisen kyseiselle ajolle. Esimerkiksi, jos testisarja sisältää BROWSER- eli selain-nimisen muuttujan, on se mahdollista muuttaa ajokomennossa. Muuttujan voi muuttaa antamalla ajokomentoon lisävaihtoehdon

```
robot –-variable BROWSER:chrome testitiedosto.robot
```

Syntaksi --variable-komennolle on <nimi:arvo>, johon nimi-kohtaan tulee annettavan muutujan nimi ja arvo-kohtaan uusi arvo. Näin yllä olevan ajokomennon avulla testit ajettaisiin Chrome-selaimella, vaikka testisarjalle olisi määritelty jokin muu selain joko resurssi- tai testitapaustiedostoissa.

Variable-vaihtoehtoja on mahdollista antaa monta testiajolle; yksi --variable-komento aina jokaista haluttua muutettavaa muuttujaa kohden.

Kuvassa 19 on esimerkki komentolinjalla ajetusta testiajosta, jossa ajettiin viisi testiä. Kaikilla testeillä oli tagina reseller. Näistä viidestä testitapauksesta neljä onnistui, mutta yhden ajon aikana tapahtui virhe. Seuraava luku käy tarkemmin läpi testiajon jälkeiset tulostiedostot, jotka Robot Framework generoi meille automaattisesti.

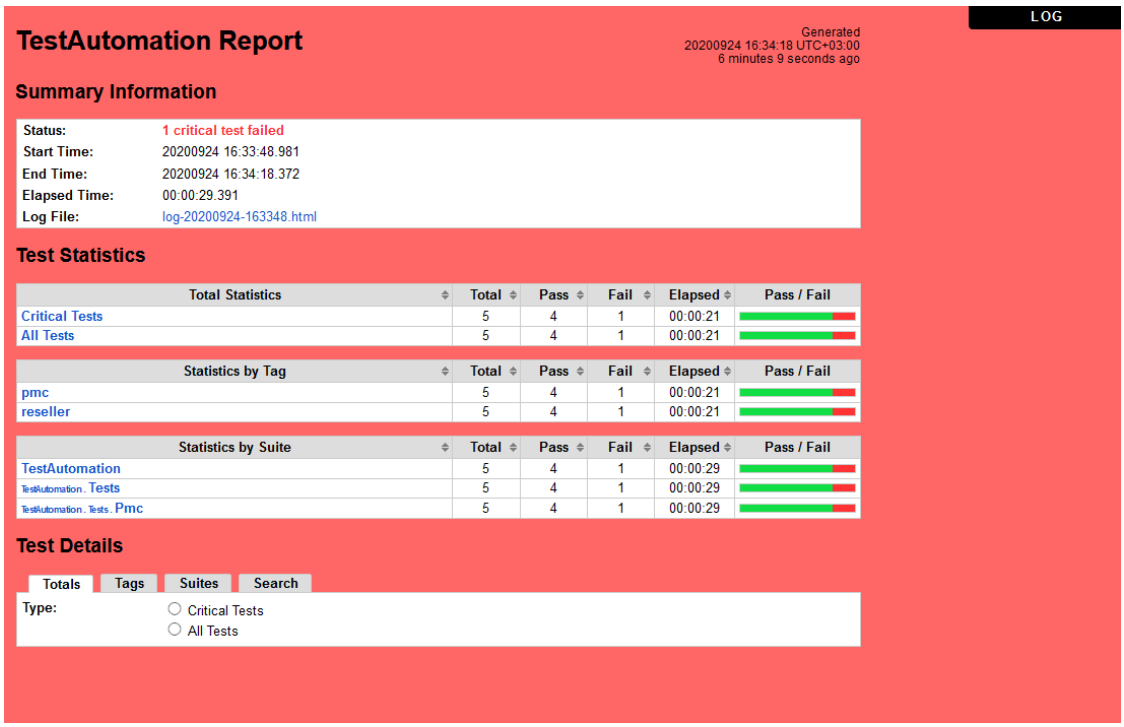


KUVA 19. Testiajo komentolinjalla. Ajokomento käyttää aiemmin annettujen komentovaihtoehtojen lyhennettyjä versioita

4.5 Testitulokset

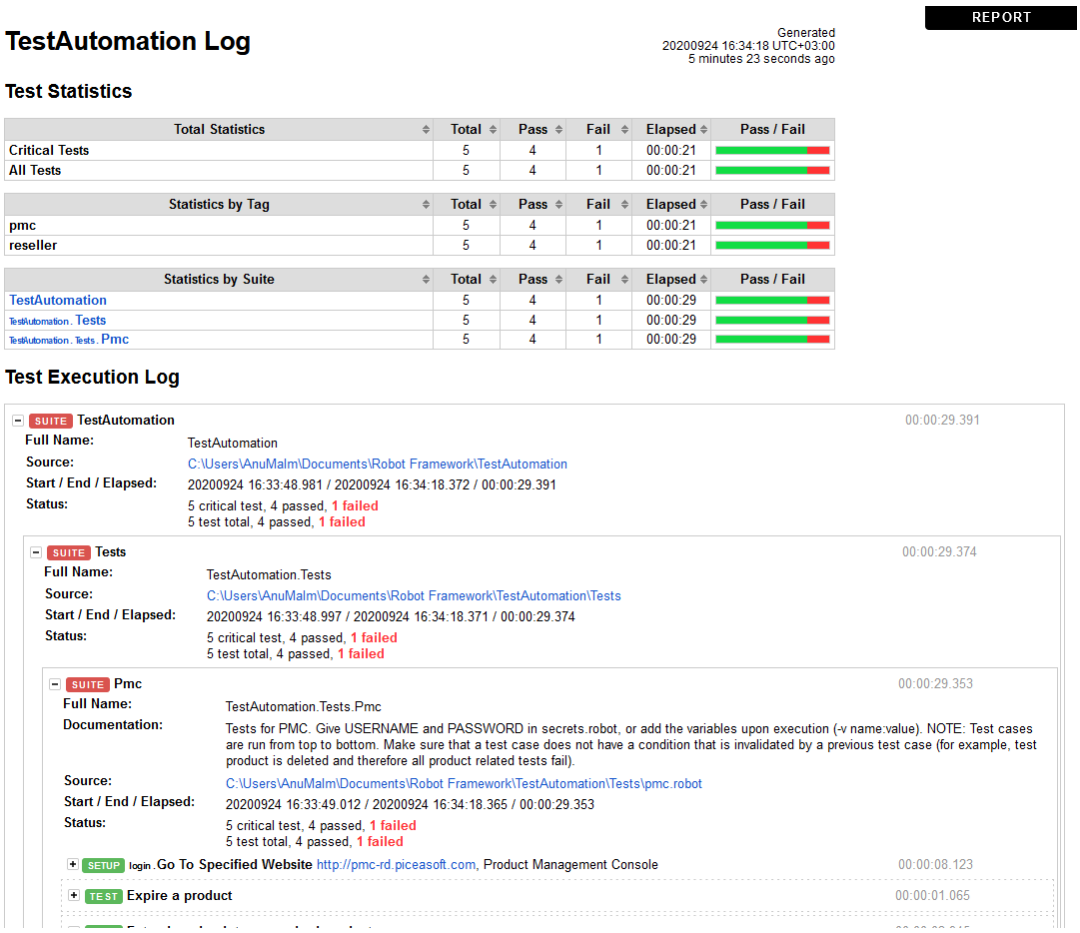
Testitulosten katselmus on tärkeä osa testiajoa. Jos ajettujen testien tuloksia ei voida arvioida, on testien ajo ajanhukkaa (Black 2016, 1.5). Robot Framework tekee testituloksien lukemisesta helppoa kehittämällä automaattisesti testiajon jälkeen raportin, lokin ja output.xml-tiedoston testien tuloksista. Raportti ja loki ovat HTML-tyyppisiä, joten ne näyttäytyvät verkkosivuna lukijalle.

Raportti (oletuksena nimellä report.html) sisältää yleiskatsauksen testien suorituksesta. Raportin taustaväri vaihtuu joko vihreäksi tai punaiseksi riippuen siitä, ovatko kaikki testitapaukset onnistuneet. Yksikin epäonnistunut testi muuttaa raportin taustavärin punaiseksi (ks. kuva 20). Näin raportista näkee nopeasti, kuinka onnistuneesti ajo sujui.



KUVA 20. Kuvakaappaus report.html-tiedostosta, joka on generoitu testiajon jälkeen. Testejä ajettiin viisi kappaletta, mutta yksi niistä epäonnistui

Robot Frameworkin loki (oletuksena nimellä log.html) antaa tarkemman kirjauksen testien tapahtumista. Epäonnistuneiden testitapausten kohdalla lokista on suuri hyöty: ne kertovat tarkkaan, missä kohtaa testi epäonnistui ja millä virheellä. Selenium ottaa myös kuvakaappauksen epäonnistuneesta testistä, mikä auttaa osaltaan testivirheen löytämisessä. Kuvassa 21 on kuvakaappaus lokitiedoston yläosasta, joka sisältää samat testitilastot testien onnistumisesta kuin raporttikin.



REPORT

TestAutomation Log

Generated
20200924 16:34:18 UTC+03:00
5 minutes 23 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	5	4	1	00:00:21	
All Tests	5	4	1	00:00:21	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
pmc	5	4	1	00:00:21	
reseller	5	4	1	00:00:21	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
TestAutomation	5	4	1	00:00:29	
TestAutomation . Tests	5	4	1	00:00:29	
TestAutomation . Tests . Pmc	5	4	1	00:00:29	

Test Execution Log

SUITE TestAutomation 00:00:29.391
Full Name: TestAutomation
Source: C:\Users\AnuMalm\Documents\Robot Framework\TestAutomation
Start / End / Elapsed: 20200924 16:33:48.981 / 20200924 16:34:18.372 / 00:00:29.391
Status: 5 critical test, 4 passed, 1 failed
5 test total, 4 passed, 1 failed

SUITE Tests 00:00:29.374
Full Name: TestAutomation.Tests
Source: C:\Users\AnuMalm\Documents\Robot Framework\TestAutomation\Tests
Start / End / Elapsed: 20200924 16:33:48.997 / 20200924 16:34:18.371 / 00:00:29.374
Status: 5 critical test, 4 passed, 1 failed
5 test total, 4 passed, 1 failed

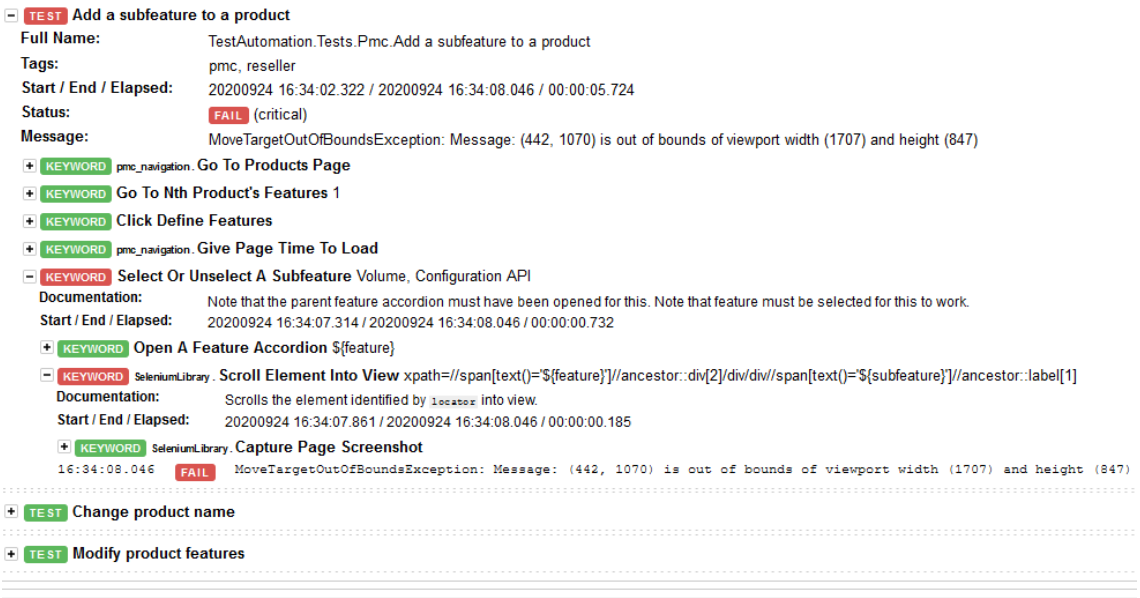
SUITE Pmc 00:00:29.353
Full Name: TestAutomation.Tests.Pmc
Documentation: Tests for PMC. Give USERNAME and PASSWORD in secrets.robot, or add the variables upon execution (-v name:value). NOTE: Test cases are run from top to bottom. Make sure that a test case does not have a condition that is invalidated by a previous test case (for example, test product is deleted and therefore all product related tests fail).
Source: C:\Users\AnuMalm\Documents\Robot Framework\TestAutomation\Tests\pmc.robot
Start / End / Elapsed: 20200924 16:33:49.012 / 20200924 16:34:18.365 / 00:00:29.353
Status: 5 critical test, 4 passed, 1 failed
5 test total, 4 passed, 1 failed

SETUP login . Go To Specified Website http://pmc-rd.piceasoft.com, Product Management Console 00:00:08.123

TEST Expire a product 00:00:01.065

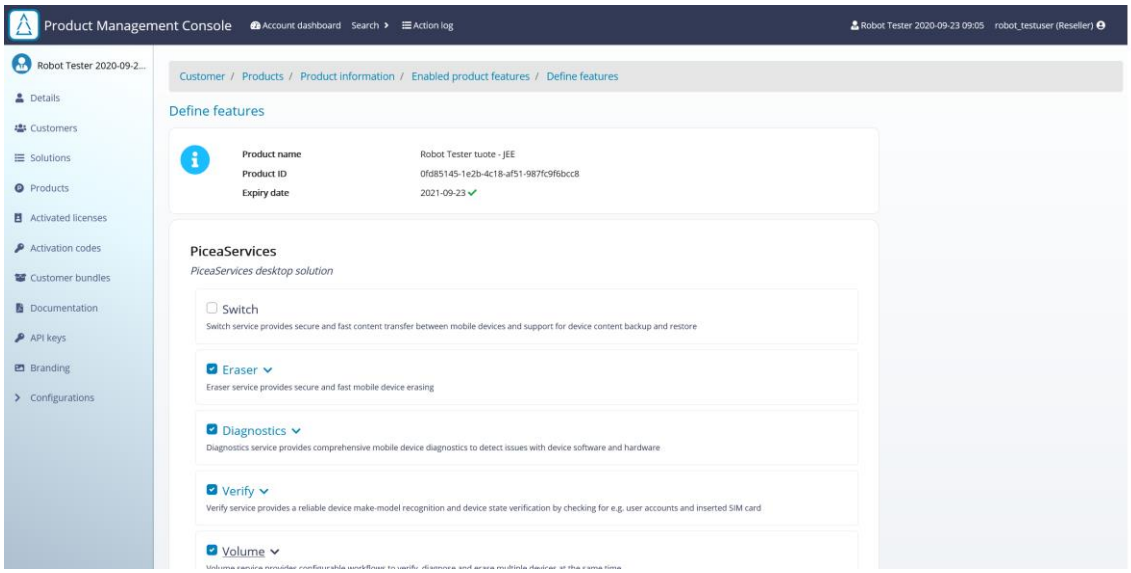
KUVA 21. Kuvakaappaus log.html-tiedostosta. *Test Execution Log* eli loki testien ajosta sisältää jokaisen käytetyn avainsanan sekä niiden tulokset puumaisena rakenteena

Kuva 22 sisältää kuvakaappauksen kohdasta, jossa testi epäonnistui. Epäonnistuneen testin nimi oli ”Add a subfeature to a product”, jossa tuotteeseen yritettiin lisätä uusi aliominaisuus. Tarkemmin testiä lukemalla huomamme, että avainsana ”Select Or Unselect A Subfeature” on ollut syypää epäonnistumiseen. Tässä tapauksessa epäonnistumisen syy on ollut se, että keskittämämme elementti on ollut selainnäkymän ulkopuolella.



KUVA 22. Lähempi näkymä epäonnistuneesta testistä log.html-tiedostossa.

Epäonnistuneiden selainlähtöisten virheiden yhteydessä Selenium ottaa kuvakaappauksen siitä, mitä ruudulla on näkynyt virheen ilmentyessä. Kuva on liitettynä lokiin ”Capture Page Screenshot”-avainsanan alle. Kuva 23 on tämän esimerkin virhetilanteen kuvakaappaus. Kuvassa ei näy Volume-ominaisuuden alla olevaa Configuration API -nimistä aliominaisuutta, joten Selenium ei ole voinut kohdentaa siihen.



KUVA 23. Seleniumin ottama kuvakaappaus epäonnistuneen testin aikana

Output.xml sisältää samat tiedot kuin log.html, mutta XML-formaatissa. Output-tiedosto sopii esimerkiksi hyvin omien kokoavien raporttien tekoon, jos Robot

Frameworkin oma raportti- ja lokitiedosto eivät palvele täysin testaajan tarkoitusperiä. Esimerkiksi eri testikertojen statistiikkoja olisi mahdollista kerätä kaikkien testikertojen output-tiedostosta. Statistiikat voitaisiin sitten koota yhteen statistics.html-tiedostoon, jotta testisarjan yleistietoja voidaan nähdä havainnollistavasti.

5 ROBOT FRAMEWORK TUOTTEIDENHALLINTASOVELLUKSEN KÄYTTÖLIITTYMÄN AUTOMATISOINNISSA

Piceasoft Oy:n tuotteidenhallintasovellus oli ollut päivittäisessä käytössä jo yli vuoden ennen käyttöliittymän testiautomaation aloitusta. Testiautomaation aloitus alusta saakka antoi paljon vapauksia sen toteutukseen ja testisuunnitelman tekoon. Jotta luotu automaatio olisi mahdollisimman hyödyllinen, keskusteltiin sen tavoitteista ensin tuotteidenhallintasovelluksen pääkehittäjän ja tiiminvetäjän kanssa. Ennen automaation aloitusta tuotteidenhallintasovellukseen tutustuttiin ensin manuaalitestauksen avulla puolen vuoden ajan, jotta sovellus ja sen tärkeimmät toiminnot olisivat helposti määriteltävissä testiautomaatiota varten.

5.1 Piceasoft Oy:n tuotteidenhallintasovelluksen lyhyt esittely

Piceasoft Oy:n tuotteidenhallintasovellus (tästä lähtien lyhennetty muotoon PMC, Product Management Console) on tukena Piceasoft Oy:n mobiililaitteiden elinkaareen liittyvien tuotteiden hallinnassa. PMC:ssä tuotteita hallitaan asiakaskohtaisesti. Kaikkia näitä asiakkaita hallitsevat eri tasoiset käyttäjät, joilla on eri laajuiset oikeudet asiakkaan tietojen muuttamiseen.

PMC:stä on kolme eri versiota: kehitysversio, jonne kehittäjät vievät uusia ominaisuuksia sitä mukaa kuin niitä valmistuu. Tuotantoversio, jossa on oikeita asiakkaita. Näiden kahden yhdistäjänä toimii staging-versio, joka on mahdollistaa kehitysversiosta tuotujen ominaisuuksien viimeisen varmistamisen ennen tuotantoon viemistä. Jälleenmyyjälle tehty testiautomaatio toimii missä tahansa näissä kolmessa ympäristössä, mutta testejä on tarkoitus ajaa pääsääntöisesti kehitys- ja staging-versioissa. Testiautomaatio toimii näissä versioissa nopeammin ja turvallisemmin, koska testiautomaatiolla ei ole mitään mahdollisuutta päästä käsiksi oikeisiin asiakkaisiin. Tämä on tärkeää varsinkin, kun testiautomaatio saadaan laajennettua esimerkiksi ylläpito-roolille, joka voi tehdä mitä vain mille vain asiakkaalle.

5.2 Testiautomaation tavoitteet

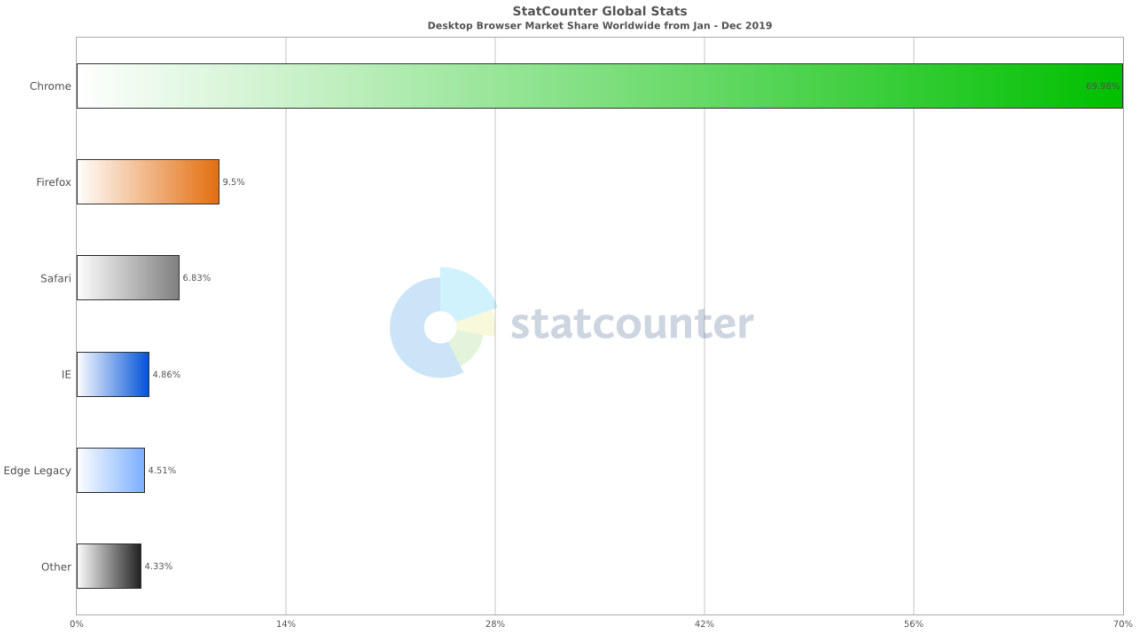
Sovelluskehitys toimii Piceasoftilla vahvasti asiakkaiden toiveita ja tarpeita kunnioittaen. Tämä lisää regression mahdollisuutta, kun uusia ominaisuuksia kehitetään ja integroidaan sovellukseen. On nopea testata vain uudet ominaisuudet kehityksen mukana, mutta sovelluksen laadun varmistamiseksi on hyvä olla sarja säännöllisesti ajettavia yleistä toiminnallisuutta testaavia testejä.

Piceasoftilla testiautomaatio yksikkötestien ja integraatiotestien osalta on hyvällä mallilla, mutta verkkosivujen käyttöliittymätestaus vielä puuttui. Keskustelimme testiautomaation tavoitteista tuotteidenhallintasovelluksen pääkehittäjän sekä tiiminvetäjän kanssa. Testiautomaation tavoitteena olisi vähentää manuaalisesti tehtävien testien määrää ja helpottaa verkkosovelluksen käyttöliittymän regressiotestausta. Päädyimme aloittamaan testiautomaation jälleenmyyjä-tason käyttäjälle. Tällä käyttäjätasolla on toiseksi rajoitetuimmat oikeudet sovelluksen käyttöön, mikä auttoi testauksen laajuuden hallinnassa. Tavoitteena oli automatisoida ns. “happy pathit” eli tilanteet, joissa kaikki sujuu oletetusti.

Omiksi tavoitteikseni asetin omien avainsanojen luonnin niin selkeästi, että niitä pystyy käyttämään ilman ohjelmointituntemusta. Tämä tarkoitti, että avainsanoihin ei tule argumentiksi esimerkiksi XPath-polkuja sivulle, vaan riittää, että käyttäjä osaa antaa oikean rivinumeron tai tekstipätkän käyttöä varten.

Robot Frameworkin asennus ja käyttö dokumentoitiin. Asennusohje käy läpi lyhyesti, mitä asentaminen pitää sisällään sekä antaa linkit tarkempiin ohjeisiin. Käyttöohje taas kertoo testien ajosta, testien teosta sekä siitä, miten testiautomaatiosta saa parhaiten tuloksia. Käyttöohje on samalla dokumentaatio testiautomaation avainsanoista ja muuttujista, mistä ne löytyvät ja mitä ne pitävät sisällään. Halusin ohjeista helpot lukea, joten tein ne verkkosivumuotoon. Muita mahdollisuuksia olisivat olleet perinteinen tekstidokumentti tai markdown-kielellä koristettu README, mutta päädyin yksinkertaiseen verkkosivuun sen monipuolisuuden takia. Tekemäni sivu pystyy järjestämään avainsana- ja muuttujataulut aakkosittain, mikä helpottaa oikean avainsanan tai muuttujan löytämistä.

Selaimista tavoitteena oli saada testiautomaatio pyörimään aluksi ainakin Chromella ja Firefoxilla. Statcounter.comin selainstatistiikan mukaan (kuvio 1) vuonna 2019 maailmanlaajuisesti käytetyimmät selaimet pöytäkoneella ovat olleet Chrome (69.98%), Safari (6.83%) sekä Firefox (9.5%) (Statcounter 2020). Kohdentamalla selainkattavuuden Chromeen ja Firefoxiin testit varmistaisivat, ettei regressiota ole tapahtunut 79.48% käyttäjistä.



KUVIO 1. Maailmanlaajuisesti suosituimmat pöytäkoneella käytetyt selaimet vuonna 2019 (Statcounter 2020)

Useamman selaimen testauksen helpottamiseksi tein komentojonotiedoston (kuva 24), joka ajaa kaikki testit neljällä eri selaimella. Käytetyt selaimet olivat Chrome ja Firefox sekä niiden headless-versiot, joissa selain toimii ilman graafista käyttöliittymää. Headless-versiot ajavat testit yleensä nopeammin, mutta virhetilanteiden huomiointi jää näillä täysin Seleniumin antamien kuvakaappausten varaan. Siksi testit ajetaan myös graafisessa selaimessa. Selainten nopeusvertailu löytyy tämän opinnäytetyön kappaleen 5.4 kuviosta 2.

```
1  robot -v BROWSER:headlessfirefox -i reseller -T -d Logs .
2  robot -v BROWSER:headlesschrome -i reseller -T -d Logs .
3  robot -v BROWSER:chrome -i reseller -T -d Logs .
4  robot -v BROWSER:firefox -i reseller -T -d Logs .
5
```

KUVA 24. Kuvakaappaus komentolinjatiedostosta

5.3 Testitapaukset

Ennen testitapausten ohjelmointia tein listan testitapauksista ja järjestin ne suurpiirteiseen tärkeysjärjestykseen. Olimme päättäneet PMC:n pääkehittäjän kanssa keskittyä happy path-tapauksiin eli tilanteisiin, joissa kaikki sujuu kuten on suunniteltukin. Tämä tarkoittaa, että testitapaukset kirjoitetaan sillä oletuksella, ettei mitään mene pieleen. Tällaisia testitapauksia ovat esimerkiksi ”Jälleenmyyjä voi muokata asiakkaan tietoja”, ”Jälleenmyyjä voi luoda asiakkaalle uuden tuotteen” ja ”Jälleenmyyjä voi muokata luodun tuotteen ominaisuuksia”. Nämä testitapaukset epäonnistuvat, jos esimerkiksi asiakkaan tiedot eivät ole tallentuneet tallennus-napin klikkaamisen ja sivun uudelleenlatauksen jälkeen.

Testitapauksia testisuunnitelmaan syntyi yhteensä 50, joista 37 tapausta toteutettiin testiautomaation luomisen ensimmäisessä vaiheessa. Kaikki tapaukset kattoivat yhden toiminnallisuuden, joista jokaisen odotettu lopputulos oli, että kaikki toimi kuten pitikin.

5.3.1 Elementtien kanssa keskustelu

Verkkosovellus ei lataudu kerralla, vaan osissa ja osien latausjärjestys voi muuttua. Tämän takia on tärkeä käyttää Seleniumin ja Robot Frameworkin sisäisen Builtin-kirjaston tarjoamia Wait- eli odotusavainsanoja. Ne auttavat varmistamaan, että elementti, jota haluamme manipuloida, on valmis käytettäväksi. Monet kirjoittamani avainsanat ovat epäonnistuneet ajoittain, kunnes olen löytänyt tilanteeseen sopivimman Wait-komennon. Suosikikseni on noussut “Wait Until Keyword Succeeds”- eli “Odota, kunnes avainsana onnistuu”-avainsana Builtin-kirjastosta. Käytännössä tämä avainsana yrittää suorittaa sille argumenttina annettua avainsanaa useamman kerran; esimerkiksi 10 sekunnin ajan sekunnin välein. Chrome etenkin kaipasi usein vähintään kaksi yritystä elementin klikkaamiseen, koska klikkaus tapahtui usein ennen kuin elementti oli valmis reagoimaan siihen. Tämä näkyy myös Chromen testiajon pituudessa; testiajo vie yli kaksi kertaa pidempään, kuin Firefox tai Chromen/Firefoxin headless-versio.

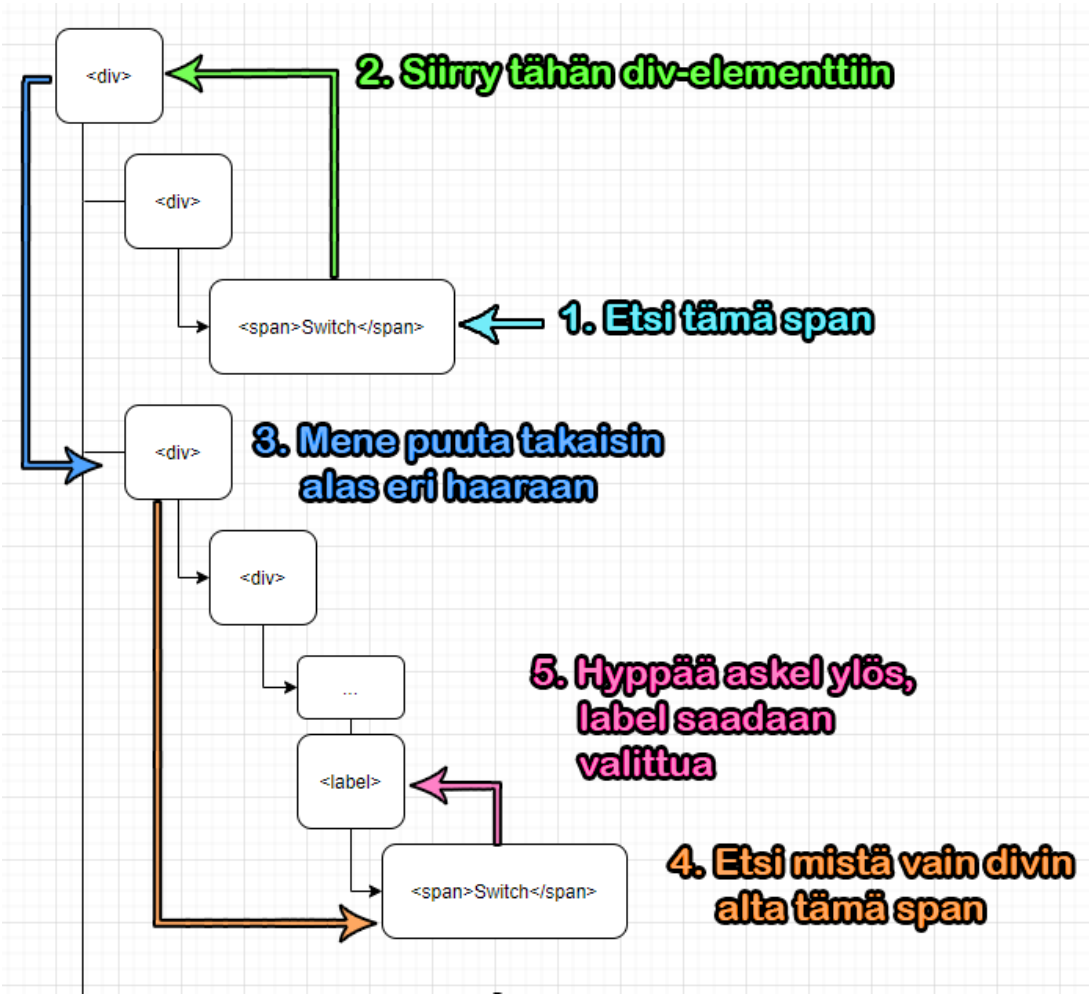
Elementteihin viittaaminen testiautomaatiossa tapahtui useimmiten elementin XPath-polkua kutsumalla. XPathin löytäminen tapahtuu helpoiten kehittäjän työkaluista selaimessa. Avaamalla kehittäjäntyökalut (Chromessa ja Firefoxissa pikanäppäimellä CTRL+Shift+i) voidaan tutkia verkkosivun HTML-elementtejä. Klikkaamalla hiiren oikeaa näppäintä halutun elementin kohdalla aukeaa valikko, jonka Copy-kohdan alta löytyy suoraan XPath-valinta. Usein tällä saadaan jo melko hyvä tulos elementin kutsumiseen, mutta joskus sivua tarkemmin tutkimalla voidaan löytää optimaalisempi ankkuri XPathille. Esimerkiksi PMC:n taulukon riveillä on kaikilla rivi-id, mutta tämä muuttui välillä eikä täten ollut luotettava elementin kutsumiseen. Sen sijaan kutsuin taulukon id:tä ja sen alta haluttua riviä. Pystyin näin tekemään omia avainsanoja, joissa yritetään esimerkiksi poistaa n:s tuote.

XPathin avulla sain tehtyä myös yhden erityisesti helppolukuisuutta auttavan avainsanan, jolla valittiin PMC:ssä tuotteeseen aliominaisuus päälle tai pois. Kaikilla tuotteen ominaisuuksilla ja aliominaisuuksilla on oma identifikaattorinsa, mutta ne eivät olleet intuitiivisia loppukäyttäjälle.

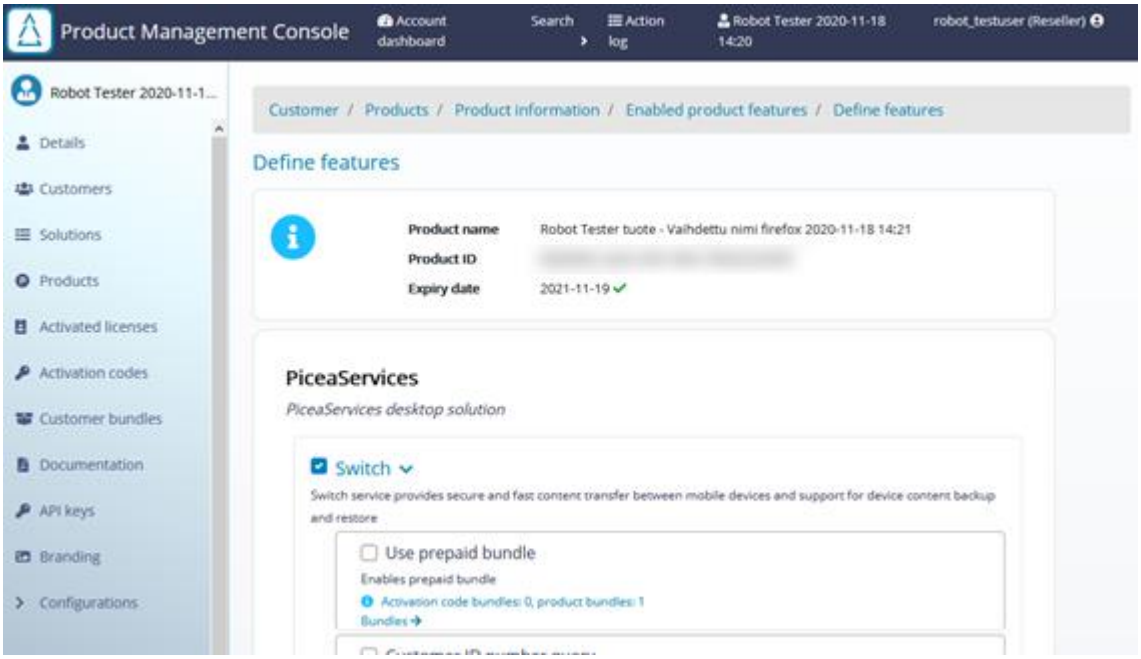
Ratkaisin ongelman luomalla avainsanan, jolle annetaan muuttujana tuotteen ominaisuuden ja aliominaisuuden nimi (esimerkiksi ”Switch” ja ”Use prepaid bundle”). Avainsana klikkaa elementtiä alla olevan XPath-valitsimen mukaan

```
  xpath=//span[text()='${feature}']//ances-
tor::div[2]/div/div//span[text()='${subfeature}']//ancestor::label[1]
```

${feature}-muuttujaan tulee haluttu ominaisuus, ja ${subfeature}-muuttujaan aliominaisuus. XPath-valitsin etsii ensin verkkosivulta span-elementin, jossa on annettu feature-teksti. Sen jälkeen XPath menee HTML-puussa pykälän ylöspäin, ja lähtee kulkemaan puuta taas alas, kunnes se löytää span-elementin, jossa on annettu subfeature-teksti. Tästä mennään jälleen yksi askel HTML-puussa ylös, jotta päädytään ensimmäiseen label-elementtiin. Tätä label-elementtiä klikataan, ja se valitsee tai poistaa valinnan annetusta aliominaisuudesta. Kuvassa 25 on lyhyesti kuvattuna sanallisesti selitetty haku ja kuvassa 26 kuvakaappaus, miltä sivu näyttää oikeasti.



KUVA 25. XPath-valitsimen kulku, nuolen väri vastaa numeroitua vaihetta



KUVA 26. Kuvakaappaus PMC:n tuotteen ominaisuuksien muokkauksesta

5.3.2 Esimerkkitestitapaus: uuden aliasiakkaan luominen

PMC:ssä asiakkailla voi olla aliasiakkaita, joita jälleenmyyjä-tasoinen käyttäjä pääsee luomaan ja muokkaamaan. Asiakkaiden poisto ei jälleenmyyjältä onnistu, joten sille ei ole omaa testitapausta. Myös asiakkaan muokkaaminen testataan jo jälleenmyyjän pääasiakkaalla, joten aliasiakastestitapaus keskittyy vain asiakkaan luomiseen.

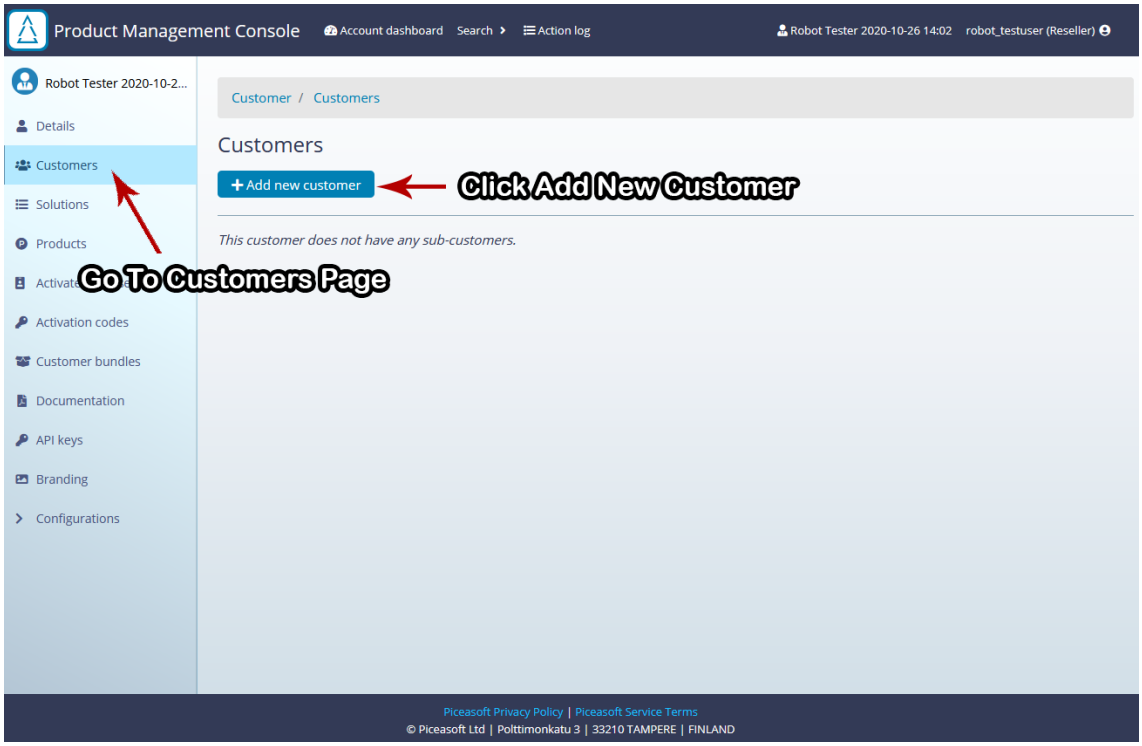
Aliasiakkaan luomisen testitapaus on tällainen:

```
Create new subcustomer
    [Tags]            reseller
    ${CurrentDate} =  Get Current Date   result_format=%Y-%m-%d %H:%M
    Start A New Browser If Last Test Left Modals Open
    Go To Customers Page
    Click Add New Customer
    Add A Name For Subcustomer
    ...  Robot Tester Jr. ${BROWSER} ${CurrentDate}
    Click Create New Customer
    Check That A Customer Was Created
    ...  Robot Tester Jr. ${BROWSER} ${CurrentDate}
```

Testitapauksen nimi on “Create new customer”, ja sille on annettu tagiksi ”reseller”, koska testitapaus koskee jälleenmyyjä-käyttäjää. CurrentDate-muuttuja auttaa antamaan asiakkaalle uniikin nimen, eikä se muutu testitapauksen alun jälkeen. Testitapauksen avainsana ”Start A New Browser If Last Test Left Modals Open” on yleiskäyttöinen avainsana, joka mahdollistaa uuden testiselaimen avaamisen virhetilanteessa. Jos yksikään aiempi testi on epäonnistunut sivun ponnahdusikkunan kanssa keskustelussa, ponnahdusikkuna jää auki ja navigointi PMC:ssä epäonnistuu. Tämä avainsana tekee testien ajosta luotettavampaa, koska yhden testin epäonnistuminen ei aiheuta ketjureaktiota.

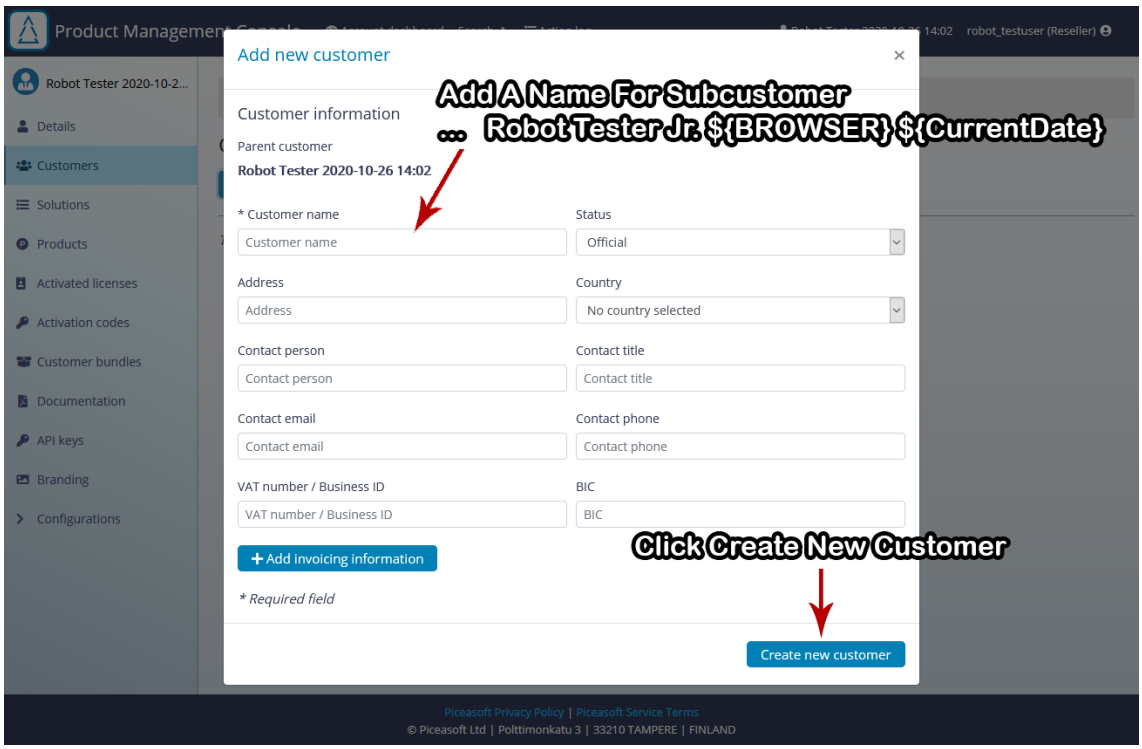
Ensimmäinen sivun kanssa keskusteleva avainsana on ”Go To Customers Page”. Tämä avainsana klikkaa kuvassa 27 vasemmassa navigaatiossa olevaa Customers-linkkiä, joka näkyy myös korostettuna. Kuvassa 27 Customer-sivulle navigointi on onnistunut, ja olemme päässeet PMC:n aliasiakkaan luomisen aloitussivulle.

”Click Add New Customer”-avainsana klikkaa kuvassa 27 näkyvää ”Add new customer”-nappia, joka avaa kuvassa 28 näkyvän asiakkaanluomisponnahdusikkunan.



KUVA 27. Aliasiakkaan luominen, aliasiakassivu

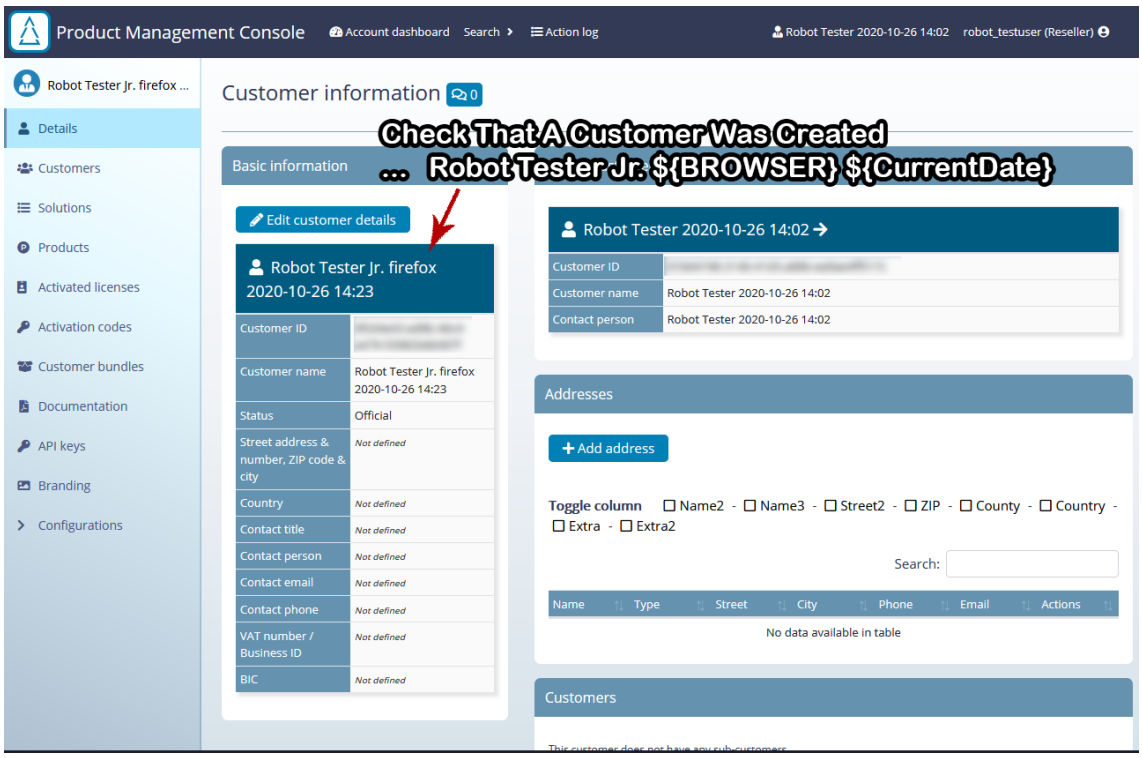
Kun ponnahdusikkuna on avautunut, ”Add A Name For Subcustomer”-avainsana kirjoittaa sille muuttujana annetun nimen asiakkaalle. Kuvassa 28 nimi kirjataan tähdellä merkittyyn ”Customer name”-syötekenttään. Nimessä on käytetty selain- ja aikaleimamuuttujia. CurrentDate-muuttuja luotiin testitapauksen aluksi, ja BROWSER-muuttuja on testisarjalle määritelty selain (esimerkiksi Firefox). Kokonaisuudessaan aliasiakkaalle annettava nimi voisi siis olla esimerkiksi ”Robot Tester Jr. firefox 2020-10-26 14:23”.



KUVA 28. Aliasiakkaan luominen, ponnahdusikkuna

Muita tietoja ei ole pakko antaa uudelle asiakkaalle, joten seuraavaksi kutsumme avainsanaa ”Click Create New Customer”. Tämä avainsana klikkaa ”Create new customer”-nappia ponnahdusikkunan oikeassa alakulmassa. PMC luo tällöin uuden asiakkaan ja ohjaa käyttäjän uuden asiakkaan etusivulle (kuva 29).

Viimein testitapauksen avainsana on “Check That A Customer Was Created”, jolle annetaan muuttujana luodun asiakkaan nimi. Oletus on, että asiakkaanluominen on onnistunut, jos aliasiakkaan etusivulla lukee luodun asiakkaan nimi. Tässä tapauksessa kuvassa 29 näkyy, että uuden asiakkaan nimi ”Robot Tester Jr. firefox 2020-10-26 14:23” on asiakkaan ”Basic information”-laatikossa.



KUVA 29. Aliasiakkaan luominen, aliasiakkaan etusivu

Aliasiakkaan luominen on testisarjan viimeinen testi. Kaikki pääasiakkaalla ajetut testitapaukset voitaisiin nyt ajaa myös uudella aliasiakkaalla, mutta käytännössä tälle ei ole tarvetta: aliasiakas eroaa pääasiakkaasta vain niin, että aliasiakkaalla on omistaja-asiakas.

5.3.3 Hyvät menetelmät

Testien helppolukuisuuden vuoksi tein lähes kaikista testitapausten toiminnoista omia avainsanoja. Kaikkien avainsanojen taustalla toimivat SeleniumLibraryn ja Robot Frameworkin omat avainsanat eri yhdistelmin. Testitapausten luomisessa ei ole väliä, kutsutaanko omia avainsanoja vai alkuperäisiä avainsanoja suoraan kirjastosta. Omilla avainsanoilla on kuitenkin monia hyviä puolia:

- Niitä on helpompi ymmärtää. Avainsanat on luotu vielä selkokielisemmäksi kuin alkuperäiset, ja niissä viitataan lukijalle tuttuihin elementteihin verkkosovelluksessa
- Testitapaukset ovat lyhyempiä, koska omat avainsanat yhdistävät useamman avainsanan yhdeksi

- Yhden toiminnon muokkaus omassa avainsanassa vie muutoksen kaikkiin testitapauksiin, jossa avainsanaa käytetään (esimerkiksi Give Page Time To Load-avainsana, joka sisältää Sleep- eli Nuku-avainsanan; yleisen odotusajan muokkaus on helppoa, kun voimme muokata vain Give Page Time To Load-avainsanaa, vrt. muokkaisimme jokaista käytettyä Sleep-avainsanaa erikseen)

Alla olevassa testitapauksessa on muistutuksena vielä esimerkki testistä, joka luo uuden aliasiakkaan. Kaikki avainsanat (Get Current Date-avainsanaa lukuunottamatta) ovat itse määriteltyjä.

```
Create new subcustomer
    [Tags]            reseller
    ${CurrentDate} =  Get Current Date   result_format=%Y-%m-%d %H:%M
    Start A New Browser If Last Test Left Modals Open
    Go To Customers Page
    Click Add New Customer
    Add A Name For Subcustomer
    ...  Robot Tester Jr. ${BROWSER} ${CurrentDate}
    Click Create New Customer
    Check That A Customer Was Created
    ...  Robot Tester Jr. ${BROWSER} ${CurrentDate}
```

Alla olevassa testiesimerkissä on sama aliasiakkaan luomistesti, mutta ilman omia avainsanoja. 11-rivisestä, helposti ymmärrettävästä testitapauksesta tulee 23 riviä pitkä testitapaus, jonka lukeminen vaatii jo enemmän tietoa testattavasta sovelluksesta. Virhetilanteessa viestinä saattaisi lukea vain, että Input Text -vaihe testissä epäonnistui; omilla avainsanoilla virheessä osattaisiin sanoa suoraan, että virhe tapahtui aliasiakkaan nimen lisäämisessä.

```
Create new subcustomer
    [Tags]            reseller
    ${CurrentDate} =  Get Current Date   result_format=%Y-%m-%d %H:%M
    ${modal} =  Run Keyword And Return Status
    ...  Page Should Contain Element       class=modal-open
    ${alert} =  Run Keyword And Return Status
    ...  Page Should Contain Element       class=ajs-no-overflow
    Run Keyword If  ${modal} or ${alert}
    ...  Go To Specified Website   ${PMC_URL}
    ...  Product Management Console
    Wait Until Keyword Succeeds  ${retry_time}  ${retry_interval_time}
    ...  Attempt Navigation To Page     class=a-sub-customers
    ...  Add new customer
    Wait Until Keyword Succeeds  ${retry_time}  ${retry_interval_time}
    ...  Interact With Element And Check For Element
```

```
...  class=create-customer-link     class=modal-open
Input Text
...  xpath=//form[@id='create-customer-form']//input[@id='name']
...  Robot Tester Jr. ${BROWSER} ${CurrentDate}
Click Element    //form[@id='create-customer-form']/div[3]/button
Wait Until Page Contains
...  Robot Tester Jr. ${BROWSER} ${CurrentDate}
Page Should Contain        Parent customer
```

Itse tehtävissä avainsanoissa on hyvä pitää mielessä uudelleenkäytettävyys. Huono avainsana toimii vain yhdessä paikassa ja tilanteessa. Esimerkiksi avainsana ”Go To Customers Page” toimii miltä tahansa muulta sivulta. Huono versio tästä avainsanasta voisi olla ”Go To Customers Page From Front Page”, joka navigoisi Customers-sivulle ainoastaan etusivulta. Joka kerta, kun Customers-sivulle haluttaisiin mennä mistä tahansa muualta, pitäisi meidän luoda taas uusi avainsana (esimerkiksi ”Go To Customers Page From Products Page”). Robot Framework tukee avainsanoille annettavia muuttujia, jolloin uudelleenkäytettävien avainsanojen luominen on vielä helpompaa.

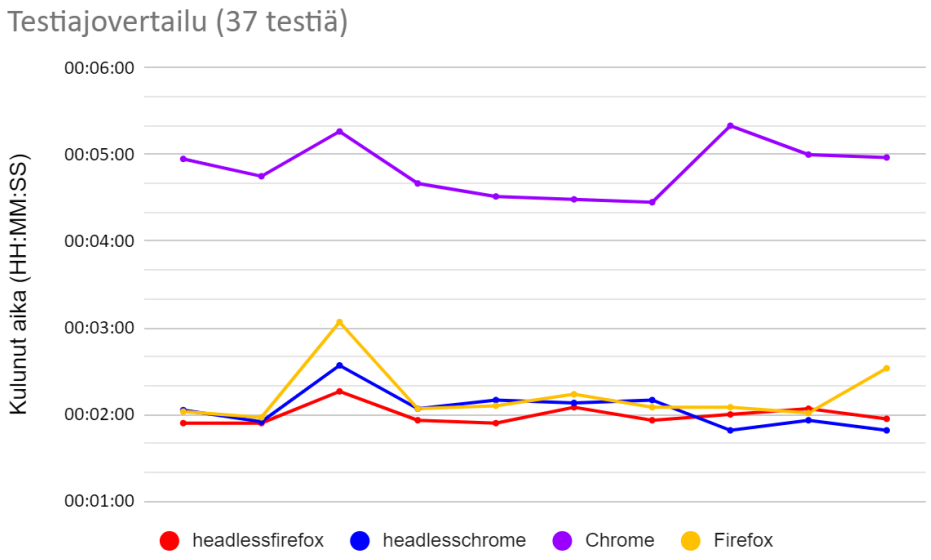
Muuttujien luominen testitapausten ulkopuolella omaan muuttujat-tiedostoon auttaa muuttujien päivittämisessä kaikkiin testitapauksiin. PMC:n testiautomaation muuttujina toimivat selain, verkko-osoite, testiasiakkaan identifikaattori ja käyttäjätunnuksen tiedot. Koska nämä muuttujat alustetaan aina testitapauksen alussa, niitä on mahdollista muuttaa ajon aikana. Tätä käytettiin hyödyksi komentolinjatiedostossa (kuva 24), jossa annetaan jokaiselle ajolle uusi selain muuttujana.

Joskus testit epäonnistuvat jonkin järjestelmän hikan takia. Esimerkiksi jokin käyttöliittymän elementti on voinut jäädä toisen elementin alle, jolloin sen kanssa kommunikointi ei ole onnistunut. Toisella kerralla elementtien latausjärjestys voi kuitenkin olla muuttunut niin, että elementin klikkaaminen onnistuu ja testit pääsevät eteenpäin. Testien lokien tutkiminen voi ehkäistä tällaisessa tapauksessa monta päänvaivaa. Erilaiset Wait- eli Odotus-avainsanat auttavat yleensä elementtien tilan varmistamiseen, mutta vastoin hyviä periaatteita olen joutunut käyttämään muutamassa testitapauksessa kylmästi sekunnin odotusta. Sekunnissa verkkosovellus on yleensä latautunut toimintavalmiiksi, ja parissa tarkkaan harkitussa paikassa testiautomaation nukutus ei vielä tee testien ajosta toivottoman hidasta.

Kun on ollut tarpeen tehdä valinta nopeuden ja testiautomaation luotettavuuden välillä, olen valinnut luotettavuuden - niin kauan kuin tämä valinta ei lisää minuutteja testiajoon, ja automatisoidut testit testaavat edelleen toiminnallisuudet nopeammin kuin manuaalitestaaja. Testiautomaatiossa kehittäjien, testaajien sekä heidän esimiestensä on voitava luottaa robottiin (Axelrod 2018).

5.4 Tulokset

Testiautomaatiota ajettiin Windows 10 -käyttöjärjestelmällä PMC:n kehitysversiossa käyttämällä hyväksi kuvassa 24 näkyvää komentolinjatiedostoa. Tämä tiedosto ajoi kaikki reseller-tagilla merkityt testitapaukset neljällä eri selaimella vuoron perään: ensin selainten headless-versioissa headlessfirefox ja headlesschrome, sitten selainten graafisilla versioilla Chrome ja Firefox. 37 testitapauksen ajamiseen meni headless-selaimilla ja Firefoxilla noin kaksi minuuttia, mutta Chrome käyttää testitapausten ajamiseen yli kaksi kertaa enemmän aikaa (keskiarvoinen ajoaika on n. 5 minuuttia). Chromessa elementtien klikkaus tarvitsee yleensä vähintään kaksi yritystä. Ensimmäisellä kerralla Chrome on usein liian nopea, eikä elementti ole vielä valmis reagoimaan klikkaukseen. Kuviossa 2 näkyy vertailuna testien ajoon kulunut aika kullakin selaimella. Kuvio sisältää kymmenen testiajon tulokset usean päivän ajalta. Kaikilla kuvion testiajoilla kaikki testitapaukset onnistuivat.



KUVIO 2. Selaimilla testiajossa kuluneen ajan vertailu

Yllättäen Firefox ajoi testit lähes yhtä nopeasti, kuin kumpikin testiajoissa käytetty headless-selain. Mielenkiintoista kuviossa 2 on testien kolmas ajokerta, jolloin kaikilla selaimilla on mennyt testien ajamiseen normaalia pidempään. Testiajojen aikaleimoja ei valitettavasti jäänyt talteen, joten yhtäkkisen piikin tarkka syy on vaikea päätellä. Kyseessä todennäköisesti on voinut olla PMC:n odottamaton jähmeys, koska kaikki selaimet käyttäytyivät samalla tavalla eikä testiautomaatiokoodia uudelleenkirjoitettu eri ajokertojen välillä.

Tuotettu testiautomaatio on antanut varmuutta perustoiminnallisuuksien toimimisesta. Manuaaliseen regressiotestaukseen käytetty aika on vähentynyt, koska testiautomaatio voi tehdä kaikki perustestit.

6 POHDINTA JA JATKOKEHITYS

Robot Framework on monipuolinen automatisointikehys. Kehyksen laajentaminen kirjastoilla antaa sille joustavuutta, ja testiautomaation sekä ohjelmistorobotiikan teko on sillä vaivatonta. Omien kirjastojen luomiseen löytyy kattavat ohjeet, jos Python-kielen tuntemus on jo kohdillaan. Robot Framework kehittyy jatkuvasti ja pysyy ajanmukaisena tukemalla esimerkiksi uusimpia Python-versioita heti, kun ne julkaistaan.

Robot Frameworkin oppiminen oli helppoa. Netistä löytyvät opetusvideot olivat tärkein opiskeluvälineeni perusteiden oppimiseen. Alun jälkeen Robot Frameworkin oma opas toimi loistavasti vaikeampien ongelmien ratkaisussa. Testiautomaation kehityksen aikana tutustuin myös tarkemmin XPathiin elementtien valitsimisessa. XPathin mahdollisuus HTML-puussa kapuamiseen ylös ja alas yhdessä elementti-identifikaattoreiden kanssa teki kaikkien elementtien kanssa keskustelusta helppoa.

Tässä opinnäytetyössä käytiin läpi vain yhden samanaikaisen Robot Framework testiajon teko kerrallaan, mutta useampi yhtäaikainen ajo olisi mahdollista Pabot-nimisellä rinnakkaisajurilla. Testitapausten ajo neljällä selaimella ei vie vielä juuri kymmentä minuuttia pidempään, mutta testiautomaation kehityksen ja laajenemisen myötä testiaika voi helposti venyä tunteihin. Näin testiautomaation aloitusvaiheessa Pabotin käyttö ei ole vielä ajankohtaista, mutta jatkoa ajatellen myös sen olemassaolosta on hyvä olla tietoinen.

Vuoden 2020 elokuussa julkaistiin verkkosovellusten testausta varten uusi avoimen lähdekoodin kirjasto Robot Framework Browser. Robot Framework Browser näyttää hyvin lupaavalta SeleniumLibraryn kilpailijalta. Se on toteutettu Playwright-nimisellä Node.js kirjastolla, joka mahdollistaisi testiautomaation ajon myös Safari-selaimella, vaikka testiautomaatio ajettaisiin muulla kuin macOS-käyttöjärjestelmällä. Koen tämän suurena mahdollisuutena, koska kuten kuviossa 1 nähtiin, on Safari yksi kolmesta käytetyimmästä selaimesta. Robot Framework Browser tukee verkkosovelluksen elementtien hakemista monipuolisem-

min kuin Selenium: siinä missä Selenium ymmärtää vain yhden tyyppistä hakutapaa kerrallaan, voidaan niitä sekoittaa vapaasti Robot Framework Browserissa. Robot Framework Browser mahdollistaa esimerkiksi tekstihaun ja XPath-haun yhdistämisen, mikä tekee hakulausekkeesta luettavamman, kuin jos sama elementtihaku tehtäisiin pelkällä XPath-valitsimella. XPath-valitsin ei esimerkiksi osaa käsitellä hyvin elementtien hakemista niiden luokkanimen mukaan, joten luokan ja XPathin sekoittaminen erikseen auttaisi luomaan lyhyempiä hakulausekkeita.

Jatkokehityksessä pelkästään testiautomaatiota varten luotujen data-test-id-tyyppisten tagien käyttö verkkoelementtien merkkaamiseen auttaisi pääsemään kiinni niihin vähemmällä koodilla. Osa testiautomaatiossa käytetyistä XPath-valitsijoista on pitkiä, ja täten herkkiä verkkosovelluksen kerrosten muutoksille. Data-test-id:n käyttämistä testattiin muutamassa elementissä, mutta näiden identifikaattoreiden laajempi käyttö vaatii testiautomaation tekijältä syvällisempää tuntemusta sovelluksen lähdekoodista. Opinnäytetyötä varten data-test-id:n käyttö jäi vain kokeilun tasolle, mutta ajan kanssa data-test-id:n lisääminen kaikkiin tarvittaviin elementteihin auttaisi testiautomaation laadun parantamisessa. On vaikea sanoa, olisiko data-test-id:n lisääminen parasta jättää kehittäjille vai testiautomaation tekijöille. Kehittäjät tuntevat jo kehittämänsä sovelluksen, ja identifikaattoreiden oikeiden paikkojen lisääminen on nopeaa – mutta uusien ominaisuuksien kehittäminen hidastuu etenkin alkuvaiheessa. Kokeneellakin testiautomaattorilla menee aikaa sovelluksen lähdekoodin ymmärtämiseen – mutta toisaalta testiautomaattori voi itsenäisesti lisätä tarvitsemansa identifikaattorit koodiin eikä sovelluksen kehitys hidastu. Omalla kokemuksellani en osaa sanoa, kumpi näistä vaihtoehdoista on loppujen lopuksi toimivampi.

Tällä hetkellä testiautomaation ajotuloksista nähdään hyvin yksittäisen ajon tulokset, mutta output.xml-tiedostoista tuotetut kokoavat statistiikat kaikista testiajoista auttaisivat testiautomaation yleisen laadun tarkastelussa. Voisi olla esimerkiksi mielenkiintoista nähdä, onko jokin tietty testitapaus alttiimpi virheille kuin jokin toinen, tai jos testitapausten ajamiseen alkaakin mennä yhtäkkiä enemmän aikaa kuin aiemmin. Testiautomaatiota on ajettu tällä hetkellä vain manuaalisesti

testit ajavalla kehityskoneella, mutta olisi mielekkäämpää automatisoida testiautomaation ajo esimerkiksi joka yö tiettyyn kellonaikaan. Käytännössä tämä vaatii vielä tutkimista, missä ja miten testit olisi paras ajaa.

Testiautomaatio ei ole koskaan valmis. Hyviin tapoihin kuuluu arvioida kirjoitettuja testejä säännöllisin väliajoin. Tämä ei koske pelkästään ajoittain epäonnistuvia testejä vaan myös aina onnistuvia, koska ne voivat antaa vääriä onnistuneita tuloksia. Tämä on erityisen tärkeää UI-testiautomaatiossa, joka on hyvin altis virheille. Testiautomaation aloituspalikat ovat vain alku koko verkkosovelluksen regressiotestauksen automatisoimiseen, mutta niistä on hyvä jatkaa kehitystä.

LÄHTEET

Agile Tech. Julkaistu 12.8.2020. How To Apply Agile Methodology In Outsourcing Software Development? Luettu 14.11.2020.
https://agiletech.vn/agile-software-development-in-outsourcing/

Applitools. 2020. 2020 Most Popular Front End Automation Testing Tools. Julkaistu 14.9.2020. Luettu 31.10.2020.
https://applitools.com/blog/2020-front-end-automation-testing/

Axelrod, A. 2018. Complete Guide to Test Automation Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects. 1st edition. Berkeley, CA: Apress.

Bisht, S. 2013. Robot Framework Test Automation. Birmingham: Packt Publishing.

Black, R. 2016. Advanced Software Testing - Vol. 1. 2nd edition. Rocky Nook.

Cohn, M. 2009. Succeeding with Agile. 1st edition. Addison-Wesley Professional.

Cypress. Päivitetty 28.10.2020. Why Cypress? Luettu 29.10.2020.
https://docs.cypress.io/guides/overview/why-cypress.html#In-a-nutshell

Fewster, M. & Graham, D. 2012. Experiences of Test Automation: Case Studies of Software Test Automation. 1st edition. Addison-Wesley Professional.

Paju, I. 2020. Testiautomaation 7 kuolemansyntiä. Webinaari. Pidetty 26.3.2020. Katsottu 31.10.2020.
https://www.tieturi.fi/webinaari/webinaari-testiautomaation-7-kuolemansyntia/

Robot Framework. n.d. Introduction, Libraries. Luettu 29.10.2020.
https://robotframework.org/

Robot Framework. 2020. Robot Framework User Guide version 3.2.2. Julkaistu 1.9.2020. Luettu 1.11.2020.
https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

SeleniumLibrary. n.d. SeleniumLibrary Installation. Luettu 1.11.2020.
https://robotframework.org/SeleniumLibrary/

Software Testing Help. Päivitetty 13.11.2020. What Is STLC V-Model? Luettu 14.11.2020.
https://www.softwaretestinghelp.com/what-is-stlc-v-model/

Statcounter GlobalStats. n.d. Vuonna 2019 käytetyimpien selainten pylväskaavio. Ladattu 1.10.2020.
https://gs.statcounter.com/browser-market-share/desktop/worldwide

WebdriverManager. 2020. WebdriverManager README.srt. Päivitetty 14.3.2020. Luettu 1.11.2020.
https://github.com/rasjani/webdrivermanager