jamk.fi

Testiautomaation työkalut

Robot Framework vs. Selenium-cucumber

Juuso Jokio

Opinnäytetyö
Huhtikuu 2020
Tekniikan ala
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Jokio, Juuso	Opinnäytetyö, AMK	Huhtikuu 2020
	Sivumäärä 29	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Testiautomaation työkalut
Robot Framework vs. Selenium-cucumber

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Ari Rantala, Esa Salmikangas

Toimeksiantaja(t)

Tiivistelmä

Tavoitteena oli tutkia kahta eri testiautomaation työkalua sekä niiden toimivuutta ja sitä, miten ne eroavat toisistaan. Ensimmäiseksi työkaluksi valikoitui avainsanapohjainen Robot Framework. Avoimen lähdekoodin Robot Framework valittiin, sillä sen dokumentaatio on kattavaa ja sitä käytetään monissa isoissa kansainvälisissä yrityksissä. Toiseksi työkaluksi valikoitui Selenium-cucumber. Molemmat eri työkalut voivat käyttää samankaltaisia avainsanakirjastoja, joten niitä hyödyntämällä saatiin työkaluja verrattua toisiinsa mahdollisimman tasapuolisesti.

Testiautomaation työkaluja verrattiin kirjoittamalla verkkosivuston käyttöliittymälle identtiset testitapaukset. Testien tavoitteena oli kirjautua onnistuneesti kaupallisen sähköpostipalvelun sisään, sekä imitoida normaalin loppukäyttäjän mahdollista reittiä palvelun sisällä.

Testeissä kirjauduttiin sähköpostiin ja kirjoitettiin lähetettävää viestiä. Viestiä ei vielä tässä vaiheessa lähetetty, vaan se tallennettiin luonnoksiin. Tallennettua luonnosta muokattiin ja se lähetettiin vastaanottajalle. Onnistuneen lähetyksen jälkeen viesti poistettiin lähetetyt-kansiosta ja kirjauduttiin ulos järjestelmästä. Testissä ajettiin testitiedostot Robot Frameworkilla ja Selenium-cucumberilla 10 kertaa kahdella eri selaimella. Näin varmistettiin itse testien toimivuus, ja testien keskimääräinen kesto.

Työkaluja vertaillessa huomattiin Robot Frameworkin raportointityökalun olevan helppokäyttöisempi, mutta Selenium-cucumber suoritti testikokonaisuuden nopeammin molemmilla eri selaimella. Työkalujen vertailuun käytettyä testikokonaisuutta voidaan myöhemmin hyödyntää regressiotestauksessa.

Avainsanat (asiasanat)

Testiautomaatio, Robot Framework, Cucumber, Selenium, Ohjelmistotestaus

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Jokio, Juuso	Bachelor's thesis	April 2020
		Language of publication: Finnish
	Number of pages: 29	Permission for web publication: x

Title of publication

Test automation frameworks

Robot Framework vs. Selenium-cucumber

Degree programme

Information and Communication Technology

Supervisor(s)

Ari Rantala, Esa Salmikangas

Assigned by

Abstract

The goal of this study was to research and compare two different test automation frameworks. The first Framework that was selected was a keyword-driven open source framework, Robot Framework. The second selected framework was Selenium Cucumber. These frameworks were chosen because they both are similar and are used for testing user interfaces. Both frameworks use the same library; hence, using the same libraries we can really highlight the differences between the different frameworks could be highlighted.

A test suite was created for both frameworks and when running the test cases, it automatically logs in to the email and starts to write a message. The message will not be sent at this point, but it will be saved to the drafts. It will go to the drafts folder and the just saved draft is then opened. Then the draft message will be edited and sent to the desired email address. Then framework should select the sent folder and delete all the sent messages and log out of the email service.

The test suite was run 10 separate times with both frameworks using Mozilla Firefox and Google Chrome, which ensured the functionality of the tests as such, and the average duration of the test suite was obtained.

While comparing tools, Robot Framework documentation tools are easier to use but Selenium-cucumber performed the test suite faster than Robot Framework with both different browsers. The test suite used for the comparison can later be utilized in regression testing.

Keywords/tags (subjects)

Test automation, Robot Framework, Selenium, Cucumber, software testing

Miscellaneous (Confidential information)

Sisältö

1 Johdanto

1.1 Tutkimuksen kuvaus

Opinnäytetyön tavoitteena oli tutkia ja verrata erilaisia testiautomaatiossa käytettäviä työkaluja. Työssä esitetään kaksi erilaista testiautomaation työkalua, jotka soveltuvat käyttöliittymien testaamiseen. Työssä asennettiin testiautomaation valitut työkalut ja kirjoitettiin automaatiotestit, jonka avulla voidaan testata sähköpostipalvelun toiminnallisia ominaisuuksia. Lopuksi verrattiin näiden työkalujen käytettävyyttä, testeissä saatuja tuloksia sekä työn aikana kohdattuja mahdollisia ongelmia. Työssä käytettiin selainten testaamiseen soveltuvaa Selenium-kirjastoa, jotta lähtökohdat olisivat molemmille työkaluille samat. Käytettäessä saman kirjaston avainsanoja, testeissä voidaan korostaa työkalujen mahdolliset erot selkeämmin.

1.2 Ohjelmistotestaus

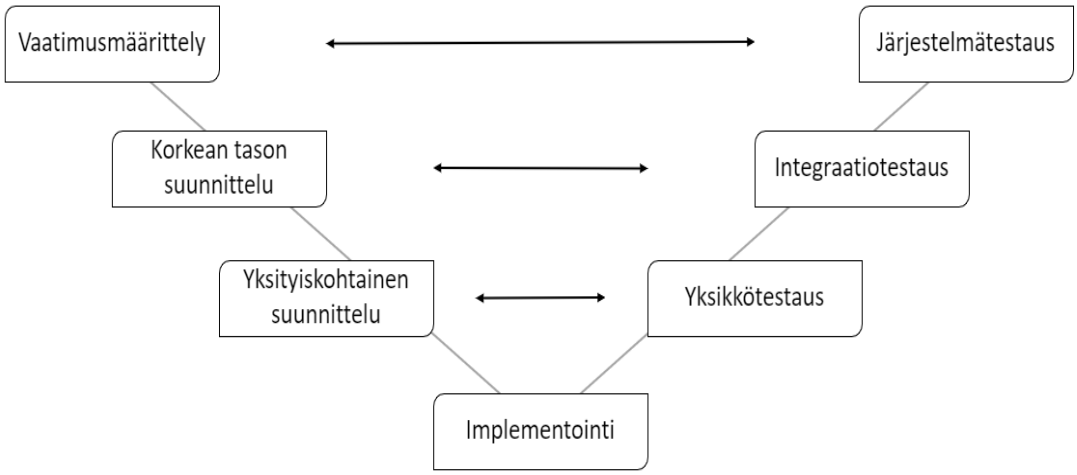
Ohjelmiston testaamisen päätavoite ei ole löytää ohjelmistosta virheitä, vaan verifioida ohjelmiston toimivuutta esimerkiksi asiakkaan näkökulmasta (Maynard n.d.). Näin voidaan paikantaa mahdolliset virheet ja minimoida niiden julkaiseminen ohjelmistossa, samalla parantaen ohjelmiston laatua. Ohjelmistotestaamisen toinen tavoite on varmistaa, että vaatimusmäärittelyt täyttyvät.

Ohjelmistotestausta voidaan suorittaa manuaalisesti siten, että ohjelmiston testaaja käy itse läpi testitapauksen ja raportoi mahdolliset löydökset eteenpäin. Ohjelmistoa voidaan testata myös automaattisesti käyttäen testiautomaation työkaluja. Tällöin testaaja kirjoittaa testit työkalulla, joka ajaa testit automaattisesti. Manuaalinen sekä automaattinen ohjelmistotestaaminen ei sulje toisiaan pois, vaan täydentävät parhaimmillaan toisiaan. Kaikkea ohjelmiston osia ei ole suotavaa automatisoida, mutta mahdolliset useaan kertaan ajettavat testit siitä hyötyvät. Automaatiotestien kattava kirjoittaminen vie paljon aikaa, ja ohjelmiston kehittyessä joudutaan myös automaatiotestejä päivittämään ohjelmiston kanssa. Automaatiota mietittäessä onkin hyvä verrata sen ylläpitoon käytettävää aikaa manuaalisen testin ajoon. Jos

ylläpidon aika on suurempi kuin testin ajaminen manuaalisesti, ei testiä kannata välttämättä automatisoida. (How to choose which test cases to automate 2020.)

1.3 Ohjelmistotestauksen tasot

Ohjelmistotestaus voidaan jakaa neljään eri tasoon: yksikkö-, integraatio-, järjestelmä- ja hyväksyntätestaus (Pressman 2005). Kuviossa 1 on esitetty V-malli testauksen eri vaiheista, joita käytetään ohjelmistoprojekteissa. Yksikkötestauksessa tarkastellaan vain tiettyä ohjelman osaa, usein voidaan testata vain yksittäisen komponentin toimintaa erillään muusta järjestelmässä.



Kuvio 1. Testauksen vaiheet V-mallissa

Integraatiotestauksessa testataan näiden yksikkötestien toimivuutta toisiinsa nähden. Näin pystytään varmistamaan komponenttien yhtenäinen toiminta ja havaitaan mahdollisesti virheitä, joita ei yksikkötestauksessa havaittu. Järjestelmätestauksessa ohjelmisto testataan kokonaisuudessaan joko lopullisessa ympäristössä tai erikseen luodussa testiympäristössä. Testiympäristön tulee olla identtinen tuotantoympäristön kanssa käyttäen kaikkia lopullisia tietokantoja ja mahdollisia muita integroituja järjestelmiä. Järjestelmätestauksessa testataan myös järjestelmän ei-toiminnallisia ominaisuuksia, joilla tarkoitetaan sellaisia

ominaisuuksia, jotka eivät näy loppukäyttäjälle saakka. Esimerkkinä ei-toiminnallisista ominaisuuksista voidaan mainita tietokantojen turvallisuus ja datan oikeellisuus. (Certified Tester Foundation Level Syllabus 2018.)

Hyväksyntätestauksessa validoidaan valmiin ohjelman toiminta vaatimusmäärittelyn mukaan. Tämä testaus on ohjelmiston viimeinen osio ennen sen julkaisemista ja se varmistaa, että ohjelmisto on vaatimusten mukainen. Jos hyväksyntätestaus ei mene läpi, ohjelmistoa ei voida julkaista. Hyväksyntätestauksen voi myös suorittaa asiakas, jotta he voivat hyväksyä tuotteelle annetut vaatimukset. Tuotteella voi olla kehittäjän ja asiakkaan puolesta omia hyväksyntäkriteerejä, joilla varmistetaan tuotteen korkea laatu. (Certified Tester Foundation Level Syllabus 2018.)

Julkaistun ohjelmiston ylläpitoa ja sen toimivuutta testataan käyttämällä regressiotestausta. Regressiotestaus tarkoittaa jo luodun ohjelmiston osion testaamista uudelleen esimerkiksi ohjelmiston päivitysten jälkeen. Regressiotestaus on yleensä automaatiotestausta, joka ajetaan aina muutoksien julkaisun jälkeen. Näitä muutoksia voivat olla esimerkiksi ohjelmistosta löydetyn virheen korjaaminen tai toiminnallisuuden lisääminen ohjelmistoon. Näin pystytään todentamaan myös vanhan järjestelmän toimivuus uusien ominaisuuksien sekä päivitysten ohessa. (Certified Tester Foundation Level Syllabus 2018.)

1.4 Automaatiotestaus

Automaatiotestaus on ohjelmiston testaamista, joka suoritetaan automaattisesti käyttäen erityisesti testiautomaatioon suunniteltuja työkaluja. Automaatiotestaus keskittyy yleensä regressiotestaukseen, kun ohjelmiston kehitys on sellaisessa vaiheessa, ettei suuria muutoksia toiminnallisuuteen ole luultavaimmin suunnitteilla. Näin ollen, aiemmin kirjotetut testiautomaatiot ovat edelleen toimivia, vaikka uusia ominaisuuksia tuotaisiin ohjelmistoon mukaan. Regressiotestauksen automaatio on siis kannattavaa, sillä näitä testejä pyritään ajamaan useita kertoja. Usein on siis ajallisesti kannattavaa automatisoida regressiotestit. Automaatiotestauksessa

käytetään erillisiä ohjelmia, joille testit kirjoitetaan ja joita käyttämällä järjestelmä voidaan testata automaattisesti.

Tällä hetkellä yksi yleinen testiautomaation työkalu on avoimen lähdekoodin Robot Framework. Robot Framework on suomalainen hyväksyntätestaukseen suunniteltu työkalu, jolla voidaan testata ohjelmistojen käyttöliittymiä. Kyseistä työkalua käytetään monissa isoissa kansainvälisissä yrityksissä kuten; Cisco, Nokia ja Finnair. (Introduction 2020).

Toinen avoimen lähdekoodin testiautomaatioon tarkoitettu työkalu Selenium-cucumber on kansainvälisesti tunnettu käyttäytymislähtöinen testiautomaation työkalu. Selenium-cucumberia käyttävät yritykset kuten; Canon, Paypal ja OpenBet. Selenium-cucumberiin kirjoitettu käyttäytymislähtöinen kehittäminen (BDD) on ohjelmistokehityksen malli, joka yhdistää kehittäjät ja asiakkaat. Tärkein BDD:n tavoite on kertoa, kuinka ohjelmiston tulee käyttäytyä eri tilanteissa. Kuvaamalla yksinkertaisia käyttäytymismalleja esimerkkinä, voidaan testien tavoitteet kuvata ohjelmiston toivotun käyttäytymisen kautta. (North 2006.)

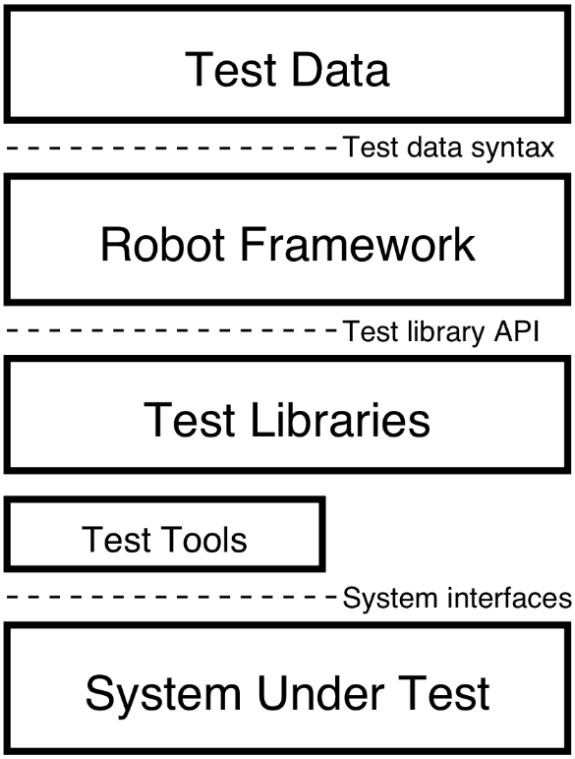
Robot Framework ja Selenium-cucumber käyttävät toimiakseen erilaisia kirjastoja, joiden avulla voidaan luoda testejä avainsanoja käyttäen. Verkkosivujen käyttöliittymän testaamiseen suunniteltu Selenium-kirjasto on kehitetty vuonna 2004 (The selenium project and tools 2020). Selenium-kirjasto on saatavilla monille eri työkaluille. Sen käyttäminen nopeuttaa testien kirjoittamista, sillä avainsanat ovat kirjoitettu valmiiksi. Selenium-kirjasto tukee useita yleisimpiä selaimia, kuten esimerkiksi Google Chrome, Mozilla Firefox, Edge, Internet Explorer, Safari ja Opera. (Driver requirements 2020.)

2 Robot Framework

2.1 Yleistä

Robot Framework on Pekka Klärckin diplomityön pohjalta kehitetty hyväksyntätestaamiseen suunniteltu avoimen lähdekoodin testiautomaatioviitekehys (*engl*. Framework). Ensimmäinen versio otettiin Nokia Networksilla käyttöön sisäisesti, mutta se julkaistiin avoimena lähdekoodina 2008. (Introduction 2020.)

Robot Framework käyttää avainsanapohjaista lähestymistapaa, jossa avainsanoilla määritetään, mitä toimintoja järjestelmässä suoritetaan testien aikana. Robot Frameworkilla on modulaarinen arkkitehtuuri, jonka takia sen käytettävyyttä voidaan lisätä ottamalla mukaan erilaisia testikirjastoja. Robot Frameworkille voidaan kirjoittaa myös itse omia kirjastoja, mikäli haluttuja ominaisuuksia ei ole saatavilla valmiissa kirjastoissa. Kuviossa 2 avataan Robot Frameworkin modulaarisuutta.



Kuvio 2. Robot Frameworkin rakenne testauksessa (Introduction 2020.)

Selenium-kirjasto on yksi Robot Frameworkiin soveltuva moduuli, jota käytetään erilaisten internetsovellusten käyttöliittymien testaamisessa. Selenium-kirjasto valikoitui tämän tutkimuksen työkaluksi, sillä se on saatavilla niin Robot Frameworkilla kuin Cucumberilla. Yhden kirjaston käyttö mahdollistaa samankaltaiset lähtökohdat testien kirjoittamiseen.

2.2 Asennus

Robot Frameworkin asennus aloitettiin lataamalla integroitu kehitysympäristö PyCharm. Se on visuaalinen ohjelmisto, jolla päästään helposti liikkumaan projektin kansiorakennelmien välillä. PyCharm-ohjelmistolla pystytään käyttämään terminaalia Robot Framework-tiedostojen ajamiseen ohjelman sisällä. Robot Frameworkia voidaan käyttää myös ilman integroituja kehitysympäristöjä.

Robot Framework-tiedostot voidaan kirjoittaa muun muassa python-ohjelmointikielellä. Python-ohjelmointikieli valikoitui käytettäväksi tässä tutkimuksessa sen yleisyyden vuoksi sekä sen yhteensopivuus tutkimuksessa käytettävän toisen testiautomaatiotyökalun kanssa. (Introduction 2020.) Tutkimuksen alussa asennettiin python-tulkki suoraan pythonin omilta verkkosivuilta, jotta voitaisiin käyttää pythonin paketinhallinajärjestelmää asentamaan Robot Framework ja vaadittavat lisäkirjastot. Pip on Pythonin sisäinen paketinhallintajärjestelmä, jolla voidaan ohjata, poistaa ja muokata ulkoisia kirjastoja. (Installing packages 2020.) Kuviossa 3 nähdään komentorivillä asennettu Robot Framework käyttäen pip-työkalua.

Kuvio 3. Robot Frameworkin asennus

Robot Framework ja Selenium-kirjasto pystytään asentamaan suoraan omalta komentoriviltä, kuten myös PyCharm-ohjelman sisäisellä terminaalilla. Kuviossa 4

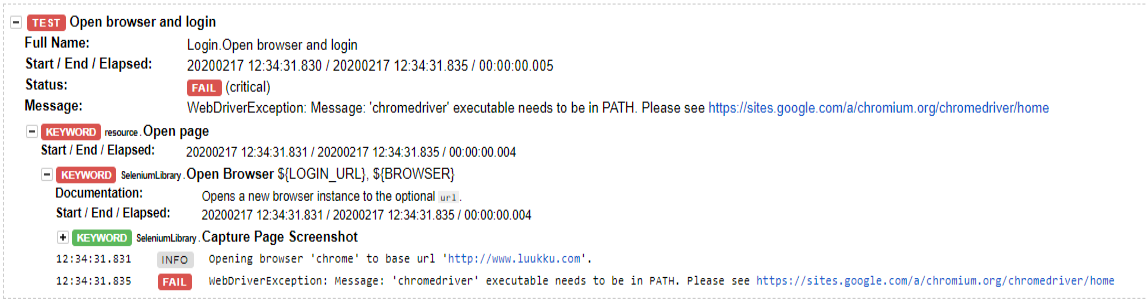
tarkastetaan asennetut versiot Pythonista sekä Robot Frameworkista. Pythonin version tarkastuksen jälkeen pystyttiin pip-työkalulla asentamaan Robot Framework sekä Selenium-kirjasto.

```
C:\Users\35850>robot --version
Robot Framework 3.1.2 (Python 3.8.0 on win32)

C:\Users\35850>python --version
Python 3.8.0
```

Kuvio 4. Versioiden tarkistus komentoriviltä

Ennen testien ajamista kansiorakenteessa pitää olla ChromeDriver-tiedosto, jonka avulla Robot Framework pystyy navigoimaan ja täyttämään erilaisia tietoja Google Chromen kautta. Ilman tätä tiedostoa mikään testeistä ei mene läpi, koska selainta ei pystytä avaamaan. Kuviossa 5 näkyy virheilmoitus, joka ilmoittaa ChromeDriverin puutteesta.



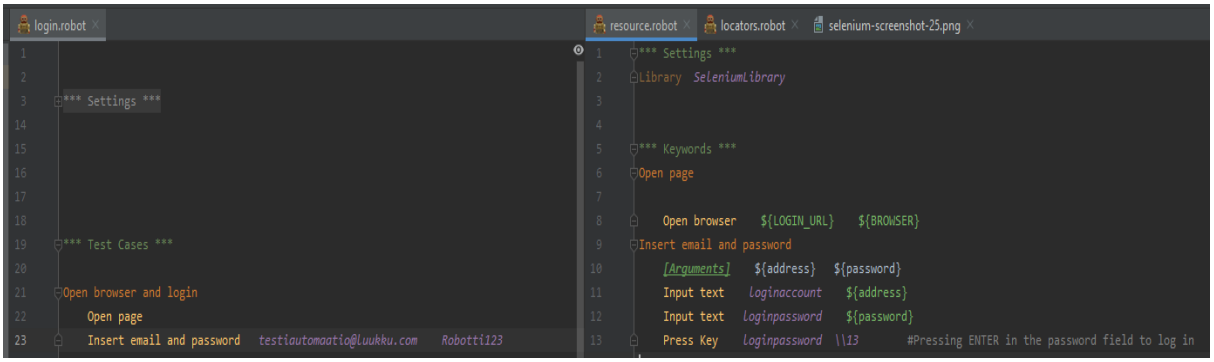
Kuvio 5. Virheilmoitus ChromeDriven puuttumisesta

Robot Framework testit voidaan ajaa myös Mozilla Firefox -selaimella, mutta se tarvitsee Mozillan oman Geckodriver-nimisen tiedoston. Tämä tiedosto on identtinen Googlen tiedoston kanssa, mutta se käyttää testeissä toimiakseen Firefox-selainta. Selaimeksi voidaan valita jokin yleisimmistä selaimista. Tässä työssä selaimeksi valittiin Google Chrome ja Mozilla Firefox, sillä yleensä testit pitää saada onnistuneesti suoritettu vähintään kahdella eri selaimella, jotta kirjoitetut testit voidaan hyväksyä. Google Chrome ovat laitettu muuttujaksi Robot Frameworkiin. Kun testit on suoritettu Chrome-selaimella, vaihdetaan selaimeksi Mozilla Firefox ja testit ajetaan uudestaan toimivuuden takaamiseksi.

2.3 Testien kirjoittaminen

Testien kirjoittaminen aloitettiin luomalla helppokäyttöisyyden takia erilliset tiedostot resursseille sekä muuttujille. Projektissa päädyttiin käyttämään testien teossa muuttujia, jolla saadaan itse testit sisältävä resurssitiedosto selkeäksi. Muuttujat nimettiin niin, että niiden nimistä pystyy helposti erottamaan sen, mitä kyseisellä muuttujalla tehdään ja missä vaiheessa.

Ensimmäisessä testissä lähdettiin todentamaan ohjelmaympäristön onnistunut asentaminen sekä toiminta avaamalla selain oikeaan osoitteeseen. Onnistuneen kirjautumistestin jälkeen lähdettiin kehittämään kyseistä testiä eteenpäin lisäämällä siihen uusi testitapaus. Kuviossa 6 on ensimmäisen testin koodi, joka sisältää resurssitiedoston sekä testitiedoston.



Kuvio 6. Ensimmäinen testi Robot Frameworkilla

Testien kirjoittaminen tapahtuu Robot Frameworkilla resurssitiedostoon sijoitettavilla avainsanoilla. Avainsanalle annetaan vapaavalintainen nimi, joka liittyy tapaukseen jollain tavalla. Esimerkiksi testissä, jossa avataan selain, avainsanan nimi on *”Open page”*. Avainsanan alle sijoitetaan sisennyksellä haluttu toiminto, mitä Robot Framework kirjastoineen tukee. Kuviossa 6 voidaankin nähdä *”Open page”* -avainsana ja sen toiminto *”Open browser”*. Tämä toiminto avaa selaimen, ja sen jälkeiset sisennykset ovat nimeltään argumentteja. Tiedetään, että avainsana*”Open browser”* tarvitsee kaksi erilaista argumenttia, joista toinen on mikä avataan ja millä avataan. Locators-tiedostossa määritellään muuttujat. Kyseisessä *”Open browser”* -avainsanassa muuttujina on *”LOGIN_URL”*, joka on asetettu osoitteeseen

“www.luukku.com". Toinen muuttujista on *”BROWSER”*, joka määrittää testien käyttämän selaimen. Tässä tapauksessa selain on Google Chrome.

Lokaattorien sijainti saadaan kätevästi verkkosivulla tutkimalla niiden elementtejä. Yleisesti verkkosivuilla olevat kentät sisältävät uniikin tunnisteen. Tämän tunnisteen avulla voidaan käskeä Robot Frameworkia painamaan oikeasta kohdasta. Tämä tunniste voidaan nimetä locators-tiedostossa muuttujaksi ja nimetä se halutulla tavalla. Muuttuja voidaan sitten asettaa halutun avainsanan sisälle, jotta se voidaan käsitellä testissä.

Kaikilla kentillä ei verkkosivuilla ole annettu uniikkia tunnistetta, mutta testissä tarvitaan silti kyseisiä kenttiä painaa. Tämä voidaan hoitaa monella eri tavalla. Yksinkertaisin tapa kentän löytämiseksi on selvittää sen Xpath-arvo. Xpath on kyseisen elementin sijainti XML-tiedostossa. Tämä Xpath voidaan lisätä locators-tiedostoon, kunhan vain kerrotaan kyseessä olevan Xpath. Xpath voi myös sisältää tunnisteen, mutta onkin suositeltavaa, että tunnisteita käytettäisiin mahdollisimman paljon ja mahdollisille Xpath-sijainneille etsitään parempi vastike. Xpathin käyttö pidempiaikaisessa testaamisessa ei ole suositeltavaa, sillä jos verkkosivulla halutun kentän sijaintia muutetaan, ei Robot Framework löydä enää haluttua kenttää kyseisestä sijainnista. Jos käytössä on pelkkä tunniste, Robot Framework käy läpi koko sivun etsien pelkkää tunnistetta. Tällöin tunnisteen sijainnilla ei ole merkitystä.

Testi-tiedostoon linkitetään resurssitiedosto sekä lokaattorien tiedosto, jotta tehdyt avainsanat voidaan ottaa käyttöön testeissä. Testi-tiedostoon kirjoitetaan halutut testitapaukset *”Test cases”*-osion alle. Testitapaukset voidaan nimetä itse halutulla tavalla mahdollisimman selkeäksi. Nimetyn testitapauksen alle lisätään halutut avainsanat sisennettynä, jotta robotti tietää, mitkä avainsanat haetaan resurssitiedostosta käytettäväksi testitapauksen ajossa. Testit suoritetaan tiedostosta ylhäältä alaspäin, ja vaikka yksi testeistä ei mene läpi, jatkaa robotti silti testien ajamista. Jos testit ovat linkitettyinä toisiinsa, yleensä siinä vaiheessa, kun yksi testi epäonnistuu, loputkin testit epäonnistuvat.

Jotkut avainsanat tarvitsevat argumentteja toimiakseen, nämä voidaan lisätä avainsanan loppuun testitiedostossa. Esimerkiksi sähköposti ja salasana voidaan määritellä laitettavaksi resurssitiedostossa, mutta laitettavat argumentit ilmaistaan vasta tositapauksen avainsanan jälkeen. Testitapaus voi sisältää usean eri avainsanan tai pelkästään yhden avainsanan. Ennen uuden testin aloittamista, järjestelmä on hyvä palauttaa alkuperäiseen tilaansa. Mikäli edellisessä testissä on tehty sisäänkirjautuminen, ei seuraavassa testissä voida kirjautua järjestelmään uudelleen sisään ilman järjestelmän alustamista tilaan ennen alkuperäistä testiä. Tämä voidaan tehdä yksinkertaisesti sulkemalla selain, jotta seuraavat testit voidaan ajaa erikseen ilman muiden testien aiheuttamia erikoisuuksia. Tätä yleensä käytetään, jos on monia erilaisia testejä ajettavana.

Testitapauksen testit kirjoitettiin kronologisessa järjestyksessä niin, että ne ovat riippuvaisia toisistaan. Jos yksi askel testistä epäonnistuu, niin myös loput testit epäonnistuvat. Kun saatiin testissä yksi osio toimimaan, lähdettiin kehittämään sitä eteenpäin. Sisäänkirjautumisen jälkeen robotti poistaa kaikki turhat viestit roskaposti-kansiosta ja siirtyy siitä kirjoittamaan sähköpostia toiselle käyttäjälle. Robotti lisää viestiin lähettäjän, otsikon sekä kirjoittaa lähetettävän viestin. Viestiä ei kuitenkaan lähetetä, vaan se tallennetaan luonnoksiin. Robotti käy tämän jälkeen hakemassa kyseisen luonnoksen kansiorakenteesta ja editoi juuri luodun viestin sisältöä. Tämän jälkeen viesti lähetetään ja lähetetty viesti käydään poistamassa lähetetyt-kansiosta. Viimeisenä tehtävänään robotti tyhjentää roskakorin, kirjautuu sähköpostista ulos ja sovellus palautetaan neutraaliin tilaan sulkemalla sovellus.

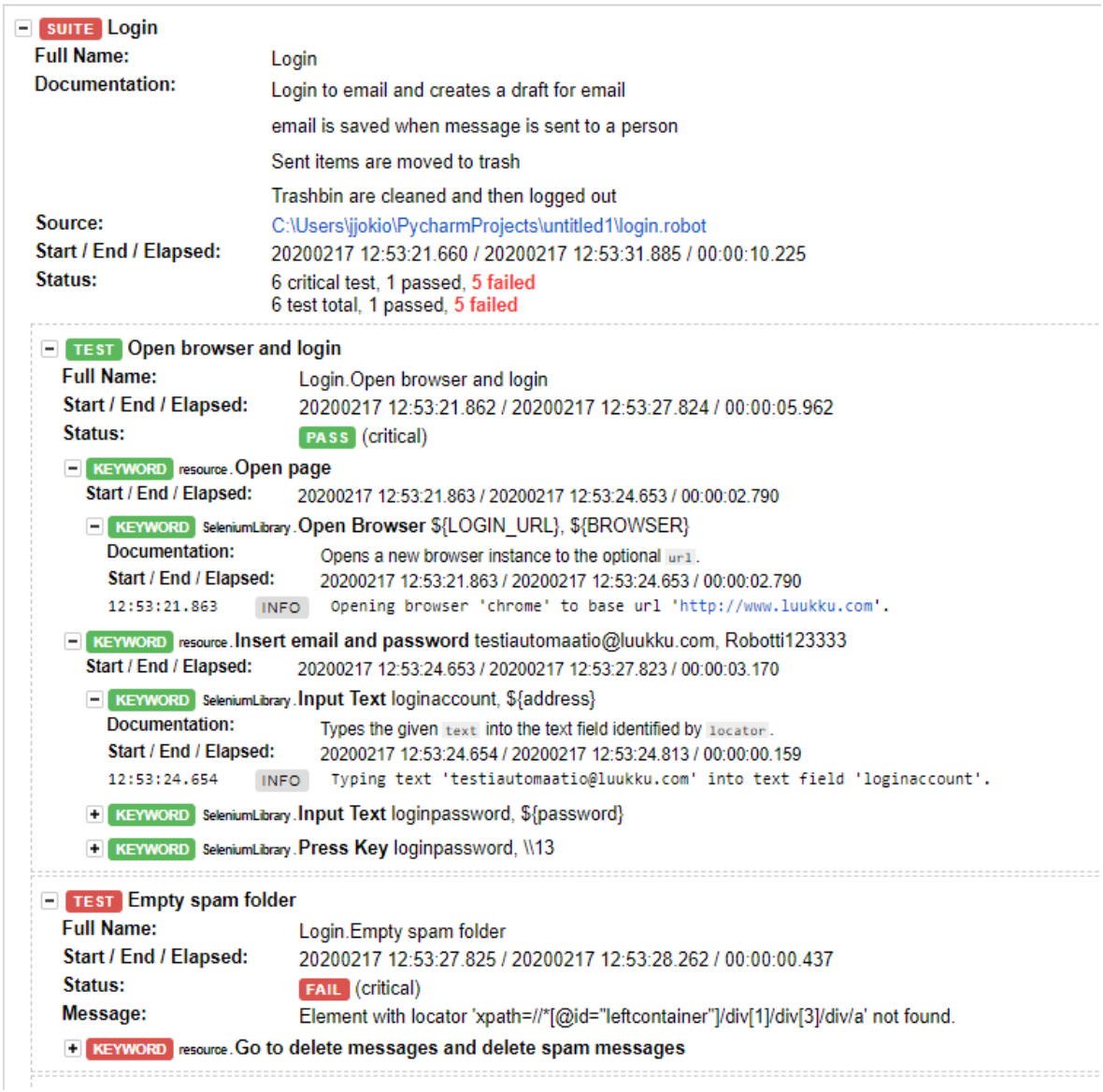
Sovelluksen toimivuutta testattiin ajamalla testit läpi Google Chrome sekä Mozilla Firefox-selaimilla 10 kertaa. Näiden ajojen raportit tallennettiin. Testien raporteista voidaan nähdä selaimien nopeuserot automaatiotesteissä. Myös tuloksien keskiarvo voidaan laskea eri selaimien välillä ja siitä saadaan arvokasta tutkimusdataa. Taulukosta 1 nähdään, että käytettäessä Robot Frameworkia testiautomaatiossa Google Chromen toimivan nopeammin kuin Mozilla Firefox. Tämä saattaa johtua siitä, että Mozilla Firefox latasi sivulla sisältäviä mainoksia hitaammin.

2.4 Testien raportointi

Testien raportointi Robot Frameworkin avulla on tehty todella helpoksi, sillä robotti generoi testien ajon jälkeen automaattisesti HTML-raportin, jossa testien statistiikka näkyy. Tulokset generoituvat omaan tiedostoon, joka tallentuu suoraan projektin kansioon. Tuloksissa näkyvät kuinka monta testiä on ajettu ja niiden eri statukset, sekä kuinka pitkä aika jokaisessa testissä on kulunut. Ajan kuluminen helpottaa testien optimoinnissa, sillä siitä saadaan konkreettista tulosta testien muutoksien välillä.

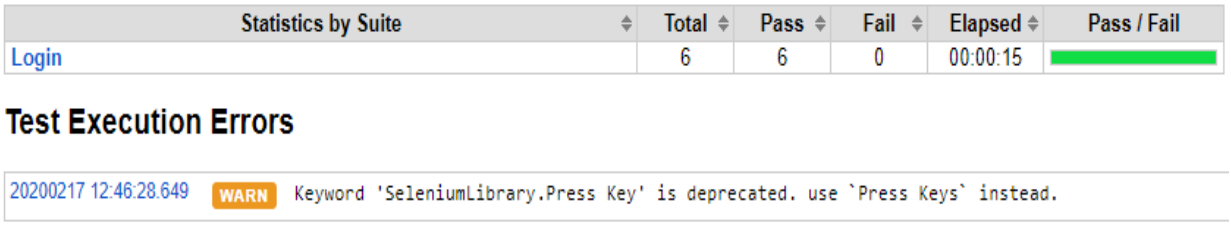
Testien tulokset saavat automaattisesti oman värikoodinsa, josta voi helposti todentaa missä kohdassa mahdolliset ongelmat ovat syntyneet. Ongelmatilanteessa raportissa merkataan avainsana, jossa testi epäonnistui.

Mahdollisen ongelman sattuessa ottaa robotti tilanteesta automaattisesti kuvakaappauksen ja se liitetään kyseiseen raporttiin mukaan. Robotin generoimassa raportissa on myös kuvakaappaukselle oma hyväksyntäkriteerinsä. Kuviossa 7 nähdään raportissa oleva ongelma sekä kuvakaappaus kyseisestä kohdasta. Kansion sisällä on vain yksi ja sama tiedosto, jota päivitetään jokaisen tiedoston ajon jälkeen, joten on tärkeää tallentaa raportti erikseen, jos kyseistä versiota raportissa tarvitsee jatkossa. Kuviossa 7 Robot Frameworkin luoma raportti sisältää onnistuneita testejä sekä epäonnistuneen testin, joka ottaa kuvakaappauksen tilanteesta.



Kuvio 7. Robot Frameworkin generoima testiraportti

Robot Frameworkin raportissa näkyvät myös varoitukset, jotka eivät välttämättä vaikuta testien lopputuloksiin, mutta voivat jatkossa rikkoa testit. Esimerkiksi vanhoilla komennoilla, joille on luotu jo uudistettu komento. Myös kirjastot, joita ei ole tietokoneelle asennettu antavat varoituksen testiraporttiin. Kuviossa 8 nähdään Selenium-kirjaston antama varoitus vanhentuneesta avainsanasta. Vaikka testit on suoritettu onnistuneesti läpi, antoi Robot Framework silti tästä avainsanasta varoituksen.



Kuvio 8. Selenium-kirjaston antaman varoitus avainsanasta

2.5 Mahdolliset ongelmat

Ongelmia testien kirjoittamisen yhteydessä havaittiin testattavan verkkosivun kanssa. Alkuperäisessä suunnitelmassa testattavaksi verkkosivuksi valittiin Googlen oma sähköposti palvelu Gmail. Ensimmäisellä kirjautumiskerralla huomattiinkin Googlen erinomaiset turvallisuustoimet, sillä kirjauduttaessa uudelta tietokoneelta Google vaatii tekstiviestin lähettämistä käyttäjälle. Näin ollen Robot Frameworkin automaatio jumiutuu salasanan kirjoittamisen jälkeen, kun tilille ei voi kirjautua ilman Googlen generoitua koodia. Testattava verkkosivusto vaihdettiin suomalaiseen luukku-sähköpostipalveluun, jossa samankaltaisia turvallisuustoimia ei ollut testien kirjoittamisen aikana implementoitu.

3 Selenium-cucumber

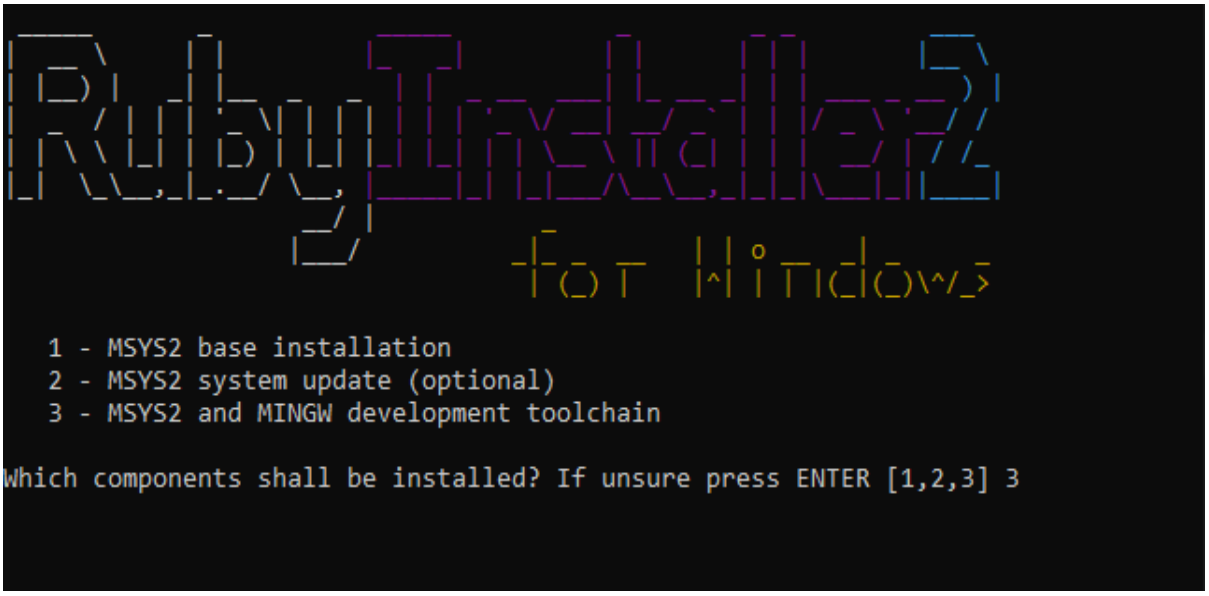
Cucumber on kansainvälisesti tunnettu avoimen lähdekoodin käyttäytymislähtöinen testiautomaation suunniteltu työkalu. Cucumberin yhdistelmää Selenium-kirjaston kanssa kutsutaan Selenium-cucumberiksi. Se käyttää Selenium-kirjastoa hyödykseen ohjelmistojen käyttöliittymien testaamisessa. Selenium-cucumberia voidaan käyttää testaamaan selaimia sekä Android-sovelluksia. (Selenium-cucumber ruby 2020.)

3.1 Asennus

Selenium-cucumberin asennuksessa päädyttiin käyttämään tietokoneen omaa terminaalia tiedostojen ajamiseen, koodin kirjoittamiseen käytettiin Notepad++-ohjelmaa. Notepad valittiin koska se on yhteensopiva käytettyjen työkalujen kanssa.

Selenium-cucumberia voidaan kirjoittaa Ruby- sekä Java-ohjelmointikielillä. Työhön valikoitui Ruby-ohjelmointikieli, sillä se omaa samankaltaisen syntaksin kuin Python. (Python vs ruby 2020.)

Asennus aloitettiin lataamalla tietokoneeseen Ruby-asennussovellus, jonka avulla pystyttiin valitsemaan työhön tarvittavat lisäosat. Selenium-cucumber tarvitsee toimiakseen Rubyn, sekä siihen tarkoitetun DevKit-työkalun. Kuviossa 9 on asennussovellus, jolla pystyttiin asentamaan tarvittavat työkalut. Kuten Pythonissa, myös Ruby sisältää sisäisen paketinhallintajärjestelmän. Rubyn paketinhallintajärjestelmä on nimeltään Gem. Selenium-cucumber asennettiin tällä paketinhallintajärjestelmällä käyttäen Rubyn Gem-komentoa *”Gem install Selenium-cucumber”*.



Kuvio 9. Ruby-ohjelmointikielen asennussovellus

Selenium-cucumber pystyy generoimaan oman kansiorakenteen käyttämällä komentoa *”selenium-cucumber gen”*. Tämä komento luo kansiorakenteen, joka sisältää lähes kaikki tarvittavat tiedostot testien kirjoittamiseen. Generointi-komento luo myös ensimmäisen testin, joten sillä voidaan todentaa systeemin toimivuus. Kuviossa 10 näkyy Gen-komennolla generoitu käyttövalmis kansiorakenne.

```
features
     |
     |__support
     |      |__env.rb
     |      |__hooks.rb
     |
     |__step_definitions
     |      |__custom_steps.rb
     |
     |__actual_images
     |
     |__expected_images
     |
     |__image_difference
     |
     |__screenshots
     |
     |__my_first.feature
```

Kuvio 10. Käyttövalmis kansiorakenne, joka sisältää myös ensimmäisen testin my_first.feature.

Kansioon pitää ladata erikseen Geckodriver- tai Chromedriver-tiedosto, jotta testit voivat käyttää selainta hyödykseen. Testit suoritetaan läpi kahdella eri selaimella. Toimivuuden takaamiseksi ne ajetaan useasti läpi Google Chromen selaimella. Kun toimivuus on vahvistettu selaimella, niin sen jälkeen ne ajetaan läpi Mozilla Firefox-selaimella.

3.2 Testien kirjoittaminen

Testit kirjoitetaan luodun kansiorakenteen feature-tiedostoon. Jokainen oma testinsä voidaan kirjoittaa feature-tiedostoon, ja kansiossa voi olla monia eri feature-tiedostoja. Nämä tiedostot ajetaan erikseen, joten ne eivät ole yhteydessä toisiinsa. Selenium-cucumber sisältää jo asennuksessaan Selenium-kirjaston, joten sitä ei tarvitse erikseen asentaa tai liittää testitiedostoon. Testeissä pystyttiin hyödyntämään Selenium-cucumberin olemassa olevia komentoja, joilla testien luonti onnistuu kätevästi ilman omien komentojen luomista.

Feature-tiedoston sisällä on yksi tai useampi skenaario. Skenaario on käsiteltävä testitapaus ja testin vaiheet kirjoitetaan tarinamuotoisesti skenaarioon käyttämällä *”Given-When-Then”*-sanoja. Testit voidaan siis kirjoittaa helposti ymmärrettävään muotoon. Esimerkiksi testi voitaisiin kuvata seuraavasti: ”Olettaen että verkkosivu on www.luukku.com. Kun käyttäjä kirjoittaa käyttäjänimen ja salasanan, niin kirjautumisen pitäisi olla onnistunut.” Tämänkaltainen tarinamuotoinen syntaksi helpottaa luomaan testejä ymmärrettävästi ja lisää helppolukuisuutta myös asiakkaan näkökulmasta.

Testien kirjoittaminen aloitettiin testaamalla yhteys verkkosivustoon ajamalla vain testi, joka avaa selaimen. Yhteyden varmistamisen jälkeen voitiin lähteä kirjoittamaan testiä eteenpäin. Kuviossa 11 on ensimmäisen testi, jolla saatiin yhteys verkkoselaimeen testattua onnistuneesti.

```
Feature: Luukku_login

  Scenario: Valid_luukku_login                                                    # features/luukku_login.feature:3

    Given I navigate to "https://www.luukku.com/luukku"                           # selenium-cucumber-3.1.5/lib/sel
enium-cucumber/navigation_steps.rb:3
    Then I clear input field having id "loginaccount"                             # selenium-cucumber-3.1.5/lib/sel
enium-cucumber/input_steps.rb:10
    And I enter "testiautomaatio@luukku.com" into input field having id "loginaccount" # selenium-cucumber-3.1.5/lib/sel
enium-cucumber/input_steps.rb:4
    And I enter "Robotti123" into input field having id "loginpassword"           # selenium-cucumber-3.1.5/lib/sel
enium-cucumber/input_steps.rb:4

1 scenario (1 passed)
4 steps (4 passed)
0m2.191s
```

Kuvio 11. Selenium-cucmberin ensimmäinen onnistunut testi

Käyttämällä Selenium-cucumberin valmiiksi luotuja komentoja, voidaan elementti kirjoittaa suoraan komentojen sisälle. Esimerkiksi tekstikenttään kirjoittamisen komento: *” Then I enter "([^\"]*)" into input field having id "([^\"]*)""* voidaan täyttää siten, että ensimmäisien sulkujen sisälle laitetaan kirjoitettava teksti. Tätä komentoa voidaan käyttää, jos kyseessä on kirjautuminen tai joku muu tekstialue. Toisten sulkujen tilalle laitetaan verkkosivulla olevan tekstialueen tunniste, jotta Selenium-cucumber tietää minkä elementin sisään kyseinen teksti pitää laittaa.

Näihin tunnisteisiin voidaan käyttää verkkosivujen elementtejä monilla eri tunnisteilla. Yleisesti elementeille on annettu tunnisteeksi id tai nimi. Näiden avulla

pystytään määrittelemään elementit, joihin Selenium-cucumberin tulee toiminnot tehdä. Jos sivulla olevalle elementille ei ole annettu nimeä tai tunnistetta, voidaan elementtiin ottaa yhteys Xpathin kautta. Xpath on elementin absoluuttinen sijainti verkkosivulla, jonka avulla elementit erotetaan toisistaan. Mikäli verkkosivun sisältö muuttuu, täytyy myös elementin Xpath korjata vastaamaan nykyistä versiota. Elementin sijaintina voidaan myös käyttää verkkosivujen class- sekä css-tietoja. Tuettavia tunnisteita Selenium-cucumberissa on *id, name, class, Xpath ja css*.

Testitapauksen skenaariot kirjoitettiin kronologisessa järjestyksessä siten, että ne ovat riippuvaisia toisistaan. Tämän avulla pystyttiin todentamaan, missä kohdassa testi mahdollisesti epäonnistuu. Tämä kehitystapa on myös hyvä integraation kannalta, sillä jokaisella ajokerralla pystyttiin todentamaan luotujen komentojen toimivuus. Näin pystyttiin arvioimaan, toimiiko testi vakaasti. Selenium-cucumberin testit pystyttiin ajamaan komennolla *"cucumber –init"*. Komennolla ajettiin koko kansio, jolloin kaikki mahdolliset kansiossa olevat feature-tiedostot ajetaan. Tämän avulla pystytään ajamaan lukuisia testejä ja niiden ei tarvitsisi olla riippuvaisia toisistaan. Valmista sovellusta testattiin kahdella eri selaimella, jotta voidaan todeta selainten väliset erot. Testattaessa huomattiin Selenium-cucumberin käyttävän Google Chromea tehokkaammin, kuin Mozilla Firefoxia.

3.3 Testien raportointi

Testeissä generoidut raportit Selenium-cucumberissa voidaan generoida eri muotoihin. Testin ajon jälkeen Selenium-cucumber ei generoi automaattisesti raporttia, vaan sen generoiminen pitää itse kirjoittaa ajettaessa feature-tiedostoa. Komentorivillä voidaan nähdä ajettujen testien tulokset, mutta niitä ei tallenneta mihinkään automaattisesti. Käynnistäessä feature-tiedostoa voidaan lisätä komentorivin loppuun raportin generointi. Raportti voidaan generoida HTML, XML, JSON- ja TXT-tiedostoihin (Reporting 2020). Selenium-cucumberille on kehitetty liitännäisiä, jonka avulla testien raportti voidaan generoida automaattisesti.

Generoidut tulokset tallentuvat feature-kansion sisälle valitussa muodossa. Tuloksissa näkyvät ajetut testit, niiden eri statukset, sekä kuinka pitkään kyseisen

avainsanan ajaminen testissä on kestänyt. Generoidessa JSON-muotoon raportti voidaan formatoida jälkikäteen erilaisten JSON-ohjelmistojen avulla, jonka jälkeen raportti on interaktiivinen. Kuviossa 12 esitetään hyväksytty testitapaus JSON-raportista, jossa näkyvät testitapauksen status ja sen kesto.


```
,,
{
  "keyword": "And ",
  "name": "I wait 2 seconds for element having class \"luukku_button\" to display",
  "line": 9,
  "match": {
    "location": "selenium-cucumber-3.1.5/lib/selenium-cucumber/progress_steps.rb:9"
  },
  "result": {
    "status": "passed",
    "duration": 34277700
  }
```


Kuvio 12. Osa Selenium-cucumberin generoitu JSON-raportti

3.4 Mahdolliset ongelmat

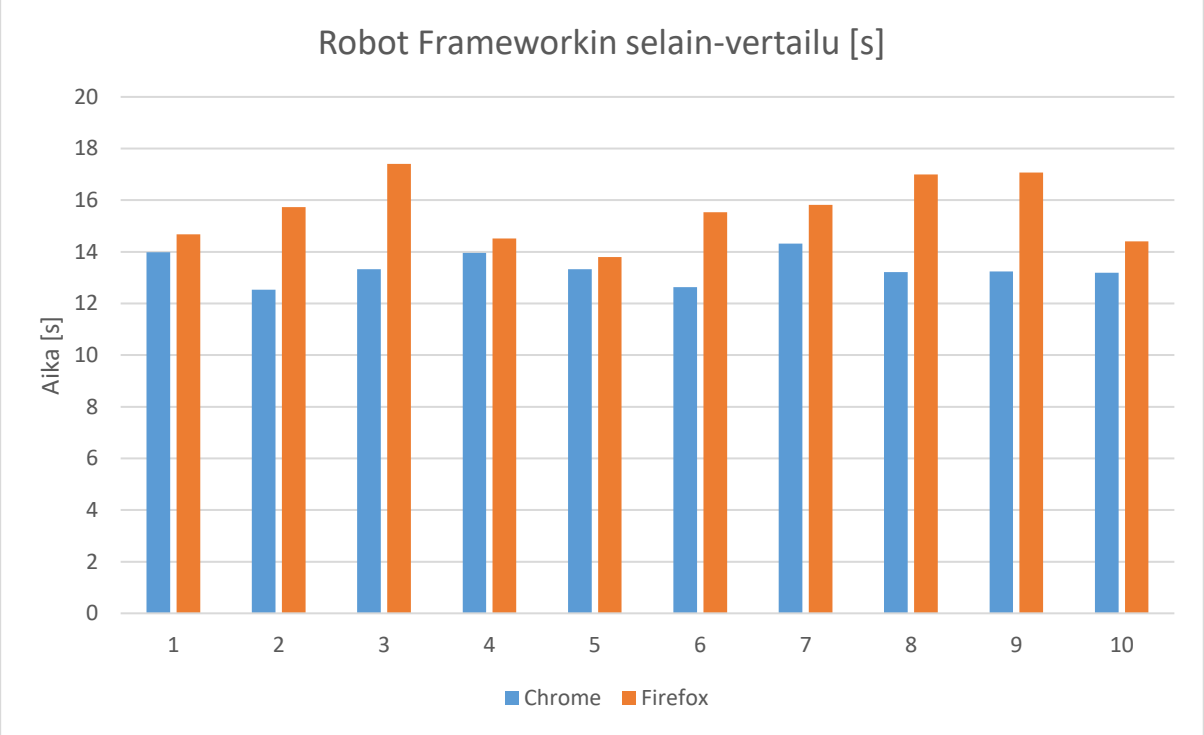
Ongelmia Selenium-cucumberin kanssa ei tullut muualla, kuin käyttöönotossa ja ensimmäisien testien yhteydessä. Asennuksessa itsessään ei tullut vastaan ongelmia, mutta testatessa asennuksen onnistumista tuli muutamia virhekoodeja. Toisin kuin Robot Framework, Selenium-cucumberin testejä ei ajeta kansiorakenteen sisällä, vaan Selenium-cucumberissa tiedosto ajetaan luodun kansiorakenteen ulkopuolella. Tästä johtuvan virheilmoituksen jälkeen siirryttiin pois kansiorakenteesta ja saatiin luotua ensimmäinen onnistunut yhteys verkkosivulle.

4 Tutkimustulokset

Molemmilla testiautomaation työkaluilla saatiin suoritettua testit hyväksytysti loppuun, ja niistä saatiin testiraportit luotua vertailua varten. Näiden tuloksien ja ohjelmiston käytössä havaittujen ratkaisujen avulla voidaan tutkia molempien työkalujen vahvuuksia ja heikkouksia. Molemmat työkalut olivat oivallisia testatun järjestelmän testaamiseen, mutta ohjelmistoissa löytyi myös keskenään eroja. Testeissä saadut tutkimustulokset ovat taulukoituna liitteessä 1.

4.1 Robot Framework -testien ajat

Testissä ajettiin testitiedostot Robot Frameworkilla ja Selenium-cucumberilla 10 kertaa kahdella eri selaimella käyttäen Google Chromea, sekä Mozilla Firefoxia. Nämä ajat otettiin ylös ja kirjattiin taulukkoon. Kuviossa 13 on esitetty Robot Frameworkin testeissä käytetty aika. Kuviosta 13 havaitaan Firefox-selaimen toimivan hitaammin, ja testien hajonta on suurempaa. Testejä ajaessa huomattiin kohdesivuston lataavan mainoksia Firefox-selaimella kauemmin kuin Googlen selaimella. Tämä vaikutti aikoihin negatiivisesti, sillä avainsana ajettiin vasta sivun latauksen jälkeen.

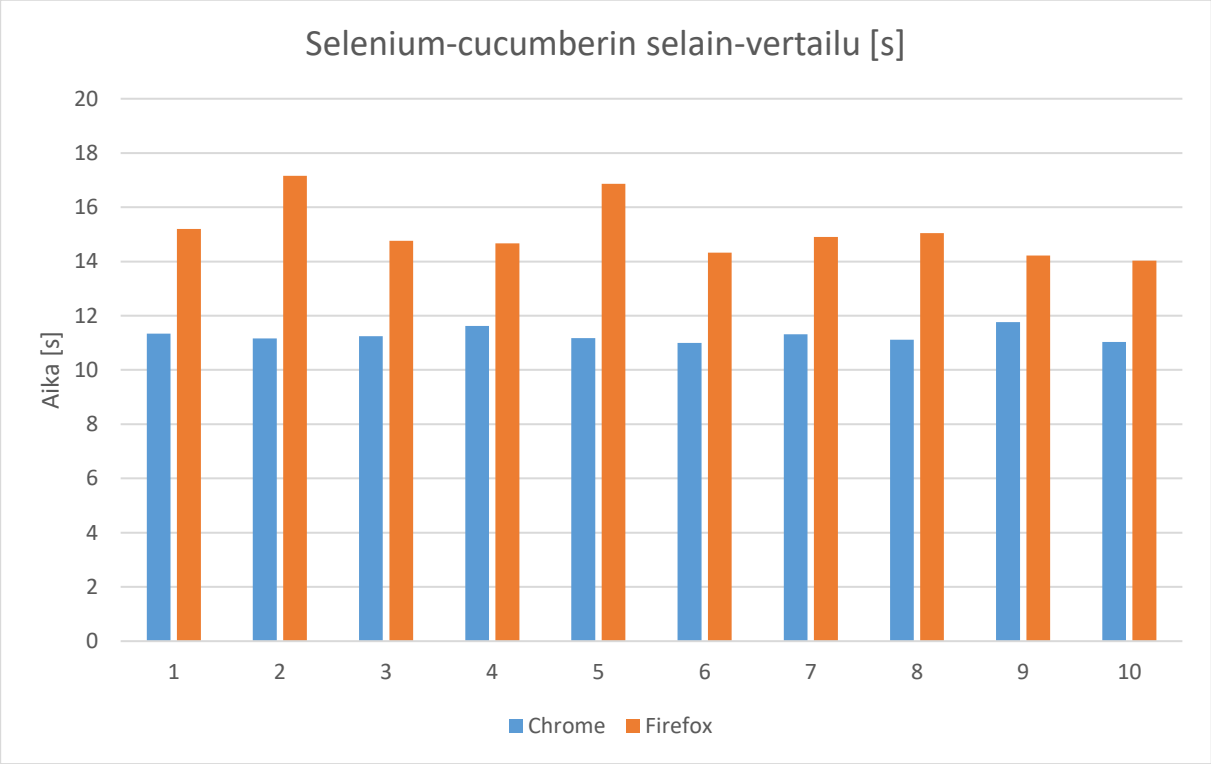


Kuvio 13. Robot Frameworkin testeissä kulutetut ajat eri selaimilla

4.2 Selenium-cucumber-testien ajat

Testissä ajettiin testitiedostot Robot Frameworkilla ja Selenium-cucumberilla 10 kertaa kahdella eri selaimella käyttäen Google Chromea, sekä Mozilla Firefoxia. Nämä ajat otettiin ylös ja kirjattiin taulukkoon. Kuviosta 14 voidaan nähdä Selenium-cucumberin testeissä käytetyt ajat. Samoin kuin Robot Frameworkilla testatessa

Firefox-selain toimi hitaammin kuin Googlen selain. Tämän avulla voidaan todeta, että Mozilla Firefoxin selain on testejä ajettaessa hitaampi vaihtoehto. Myös testien hajonta on suurempaa Firefox-selaimella.



Kuvio 14 Selenium-cucumberin testeissä kulutetut ajat eri selaimilla

5 Johtopäätökset

Robot Frameworkia käytettäessä testit saatiin kirjoitettua testitiedostoon selkeämmäksi, sillä käyttäessä erillistä resurssitiedosta saatiin elementtien sijainnit sekä nimet piiloon itse testitiedostot. Myös lokaattorien käyttäminen selkeytti avainsanojen lukemista Robot Frameworkissa. Selenium-cucumberin testi oli helppolukuista, mutta jos elementti ei sisältänyt id:tä, josta pystytään helposti toteamaan mitä kyseinen elementti merkitsee, testin seuraaminen vaikeutui huomattavasti. Näin ollen testistä ei voinut suoraan kertoa mihin testissä kohdistetaan toiminnot. Testien kirjoittaminen oli molemmilla eri työkalulla selkeää, ja molemmat työkalut suorittivat testit onnistuneesti läpi.

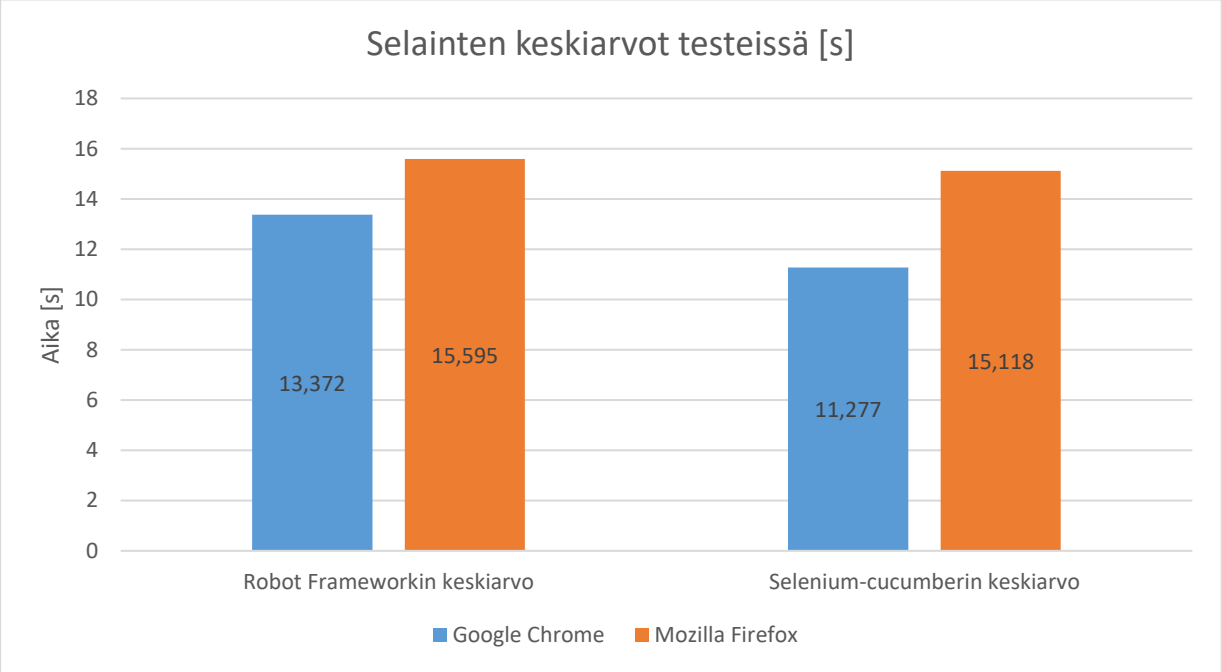
Yksi Robot Frameworkin etu on sen helppokäyttöinen testiraportin generointi. Robot Framework generoi testien tulokset automaattisesti HTML-formaattiin, joka sisältää suoritetut testitapaukset sekä testiympäristön tilan ongelmien sattuessa. HTML-formaatin avulla raporttia voidaan tarkastella selaimessa ilman ylimääräisiä ohjelmistoja. Virhetilanteessa Robot Framework myös liittää kuvankaappauksen raporttiin testiympäristön tilasta ongelman sattuessa. Tämän avulla on mahdollista todeta syy testin epäonnistumiseen, esimerkiksi internet-yhteyden katkeaminen.

Selenium-cucumberin testiraportti voidaan luoda eri formaatteihin, mutta se ei ollut tämän tutkimuksen fokuksena. Selenium-cucumberin toinen huomioitava ero Robot Frameworkiin oli testiraportin generointi. Testiraporttia ei generoitu automaattisesti, vaan raportin generointi joudutaan lisäämään terminaalissa komentoon eksplisiittisesti. Tämän takia osa testiraporteista jäi generoimatta, sillä testejä ajettaessa ei generointikomentoa käytetty jokaisessa testiajossa.

Molempien työkalujen käyttäessä Selenium-kirjastoa, olivat avainsanat lähellä toisiaan, siten helpottaen testien yhteneväisyyttä. Seleniumin kattavan kirjaston ansiosta web-testaukseen työssä ei tarvittu itse luotuja avainsanoja, vaikka tämä olisi ollut mahdollista molempien työkalujen kohdalla.

Robot Frameworkin dokumentaatio oli kattava ja ongelmia kohdatessa voitiin etsiä Robot Frameworkin omasta dokumentaatiosta ongelmiin ratkaisua. Myös Selenium-kirjaston avainsanat ovat generoitu helposti omalla verkkosivullaan, sisältäen dokumentaation ja argumentit jokaiselle avainsanalle. Selenium-cucumberin verkkosivut olivat huomattavasti suppeampia sisältönsä puolesta. Selenium-cucumberin verkkosivuilta löytyivät asennusohjeet sekä valmiiksi toimivia avainsanoja, mutta Selenium-cucumberin dokumentaatio ei ollut yhtä kattavaa verrattaessa Robot Frameworkin dokumentaatioon. Ongelmatilanteessa Robot Frameworkin yhteisö oli aktiivisempaa kuin Selenium-cucumberin, joten etsiessä ratkaisuja ongelmaan internetissä, hakukone tarjosi laajempaa tukea Robot Frameworkille. Tämä voi johtua laajasta Robot Frameworkin käytettävyydestä teknologiateollisuudessa.

Molemmat työkalut olivat nopeampia Google Chrome-selaimella kuin Mozilla Firefoxin selaimella. Selenium-cucumber oli noin 1–2 sekuntia nopeampi Robot Frameworkia käytettäessä Googlen selainta. Firefox-selainta käytettäessä testeihin kulutettu aika oli Google Chromea korkeampi. Robot Frameworkin ja Selenium-cucumberin välinen aika oli pienentynyt alle sekuntiin Mozillan-selainta käytettäessä. Kuviossa 15 on esitetty testeissä kulunut keskimääräinen aika selainkohtaisesti. Testejä ajaessa ei käytetty sleep-komentoa, jotta testien ajamisen nopeus voitaisiin dokumentoida mahdollisimman tarkasti. Ajat otettiin talteen jokaisen ajon jälkeen, suoraan työkalujen omista testiraportista.



Kuvio 15 Testien keskimääräinen kesto molemmissa työkaluissa

6 Pohdinta

Työssä pyrittiin tutkimaan erilaisia testiautomaation ohjelmistoja. Tekijä pystyi tämän avulla syventämään omaa tietämystään testiautomaatioon sekä onnistui omaksumaan uuden testiautomaation työkalun käyttöä.

Testeissä pystyttiin käyttämään molemmilla työkaluilla omaa versiotaan Selenium-kirjastosta ja sen avainsanoista. Selenium-kirjaston avulla testit pystyttiin luomaan

selkeästi ja vertailukelpoisiksi, jotta voitiin nähdä, kuinka hyvin testiautomaation työkalut toimivat toisiinsa nähden.

Työkalut olivat samankaltaisia, mutta Robot Frameworkin tukeminen on maailmalla laajempaa. Tämä huomattiin etsittäessä informaatiota molempiin työkaluihin. Selenium-cucumberin testien kirjoittaminen oli selkeää eikä omia avainsanoja tarvinnut kirjoittaa, sillä Seleniumista löytyivät kaikki tarvittavat avainsanat testien suorittamiseen. Molemmissa työkaluissa on kuitenkin mahdollisuus kirjoittaa omia avainsanojaan.

Robot Frameworkissa muuttujien käyttö selkeytti testitapauksien seuraamista ja ongelmien ratkaisemista. Dokumentaation automaattinen generointi helpotti tuloksien lukemista. Selenium-cucumberin testien ajo oli nopeampaa kuin Robot Frameworkilla. Kuitenkin Robot Framework on yleisempi valinta testiautomaation työkaluksi. Robot Frameworkin tuki on laajempaa ja sitä käytetään useassa kansainvälisessä yrityksessä.

Opinnäytetyön tekijällä oli kokemusta ohjelmistojen testaamisesta Robot Frameworkilla. Selenium-cucumber oli puolestaan tekijälle tuntematon työkalu. Tutkimuksessa pyrittiin etsimään vaihtoehtoisia työkaluja Robot Frameworkille, sekä samalla kasvattamaan työn tekijän tietämystä automaatiotestauksessa.

Työtä varten kirjoitettuja testitapauksia voitaisiin nykyisessä muodossa käyttää sivuston regressiotestaukseen. Jatkokehityksenä työlle olisi luoda lisää testejä, jotta saataisiin sivustolle laajempi kattavuus. Uuden testiautomaation ohjelmistoa voitaisiin testata rakentamalla samanlainen testiympäristö, jotta sitä voitaisiin verrata Robot Frameworkin sekä Selenium-cucumberin automaatiotyökaluihin. Testaamalla useita eri työkaluja päästään tilanteeseen, jossa voidaan valita sopivin työkalu tapauskohtaisesti.

Lähteet

Cucumber reports 2020. Toolsqa:n verkkosivut. Viitattu 14.2.2020
https://www.toolsqa.com/selenium-cucumber-framework/cucumber-reports/

Driver requirements 2020. Selenium:in verkkosivut Viitattu 14.2.2020
https://www.selenium.dev/documentation/en/webdriver/driver_requirements/

How to choose which test cases to automate 2020. Dzone:n verkkosivut. Viitattu 10.2.2020
https://dzone.com/articles/how-to-choose-which-test-cases-to-automate

Installing packages 2020. Python:in verkkosivut. Viitattu 20.1.2020
https://packaging.python.org/tutorials/installing-packages/

International Software Testing Qualifications Board (ISTQB). 2018. Certified Tester Foundation Level Syllabus, 36, 39, 41. ISTQB:n verkkosivut Viitattu 10.2.2020.
https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html

Introduction 2020. Robot Framework:in verkkosivut. Viitattu 14.2.2020
http://robotframework.org/

Maynard, C. N.d. What is software testing? Atlassian:in verkkosivut. Viitattu 10.2. 2020. https://www.atlassian.com/continuous-delivery/software-testing

North, D. 2006. "Introducing BDD". Dan North. Dannorth:in verkkosivut. Viitattu 10.2.2020
https://www.dannorth.net/introducing-bdd/

Pressman, R. S. 2005. Software engineering: a practitioner's approach. 390. p. McGraw-Hill Professional. Viitattu 10.4.2020

Python vs ruby 2020. Educba:n verkkosivut. Viitattu 2.1.2020
https://www.educba.com/python-vs-ruby/

Reporting. 2020. Cucumber:in verkkosivut. Viitattu 14.2.2020
https://cucumber.io/docs/cucumber/reporting/

Selenium-cucumber ruby 2020. Selenium-cucumber:in verkkosivut. Viitattu 20.1.2020
https://github.com/selenium-cucumber/selenium-cucumber-ruby

The selenium project ad tools 2020. Selenium:in verkkosivut. Viitattu 14.2.2020
https://www.selenium.dev/documentation/en/introduction/the_selenium_project_and_tools/

Liitteet

Liite 1. Taulukko testitiedostojen nopeudesta eri selaimilla

ROBOT FRAMEWORK CHROME	ROBOT FRAMEWORK FIREFOX	SELENIUM-CUCUMBER CHROME	SELENIUM-CUCUMBER FIREFOX
00:13.978	00:14.679	00:11:335	00:15:203
00:12.536	00:15.735	00:11:159	00:17:153
00:13.326	00:17.408	00:11:244	00:14:766
00:13.957	00:14.511	00:11:624	00:14:670
00:13.330	00:13.802	00:11:179	00:16:860
00:12.638	00:15.531	00:10:998	00:14:329
00:14.315	00:15.815	00:11:316	00:14:907
00:13.210	00:16.997	00:11:120	00:15:045
00:13.236	00:17.067	00:11:766	00:14:217
00:13.194	00:14.411	00:11:035	00:14:035