Olli Jussila

Tietojärjestelmän regressiotestien automatisointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikka

Insinöörityö

6.5.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Olli Jussila Tietojärjestelmän regressiotestien automatisointi 17 sivua + 1 liitettä
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Sähkö- ja automaatiotekniikka
Ammatillinen pääaine	Automaatiotekniikka
Ohjaajat	Lehtori Timo Kasurinen

Insinöörityön tavoitteena oli luoda sovelluksen testausta varten automatisoidut regressiotestit vanhojen manuaalisesti suoritettavien regressiotestien tilalle. Testit luotiin yritykselle, joka tarjoaa sovelluskehitystä testattavaan järjestelmään. Automatisoidut testit tuli luoda helposti ylläpidettäviksi, sillä regressiotestit luotiin tuotekehitysprojektille. Testien kehitykseen annettiin vapaat kädet.

Suurin osa työn lähteistä ovat peräisin Robot Frameworkin verkkosivujen dokumentaatiosta.

Työssä käytettiin Robot Frameworkia testien suorittamiseen ja RIDEä testien kehittämiseen ja hallintaan. Robot Framework käytti Chrome Webdriveria toimiakseen selainsovelluksessa ja pymssql-moduulia toimiakseen sovelluksen tietokannassa. Robot Frameworkiin asennettiin useita kirjastoja, jotka mahdollistivat kattavamman testauksen.

Työn tuloksena syntyi järjestelmän toiminnallisuuksia testaava kokonaisuus, joka sisälsi 10 testitapausta. Testitapauksia voidaan ylläpitää ja ajaa RIDEä.

Avainsanat	Sovellus, testaus, automaatio, Robot Framework

Author	Olli Jussila
Title	Automation of Information Systems Regression Testing
Number of Pages Date	17 pages + 1 appendix 31 August 2017
Degree	Bachelor of Engineering
Degree Programme	Electrical and Automation Engineering
Professional Major	Automation Technology
Instructors	Timo Kasurinen, Senior Lecturer

The goal of this study was to create automated software tests to replace manually executed tests. Tests were created for a company that offers software product development for the system being tested. Automated tests were made to be easily maintained because the tested system is being updated all the time. No development requirements were given from the target company.

Majority of sources were retrieved from documentation found in Robot Framework's website. The tests were made to be ran by Robot Framework and RIDE was used to develop and control tests. Robot Framework used Chrome Webdriver to connect to tested software and pymssql module to connect to systems database. Many libraries were installed to robot framework to ensure comprehensive testing.

As an outcome ten test cases were created. These test cases can be executed and maintained with RIDE.

Keywords	software, testing, automation, Robot Framework

Sisällys

1 Johdanto

Opinnäytetyön tavoitteena oli rakentaa automatisoidut regressiotestit kohdeyrityksen selainpohjaiselle sovellukselle. Regressiotestien automatisoinnilla pyritään tuomaan nopeutta ja tehokkuutta yrityksen tuotekehitykseen. Samalla yrityksen tuotteen laatu paranee, sillä automaatiolla havaitaan virheet, jotka saattavat jäädä huomioimatta testaajan inhimillisien syiden, esim. väsymyksen takia.

Kohdeyrityksellä on useita toimipisteitä Suomessa, ja se tarjoavaa useita ICT-ratkaisuja, sekä palveluita monelle eri toimialle.

Regressiotesteillä tarkoitetaan sovelluksen toimintojen toimivuuden tarkastusta. Regressiotestaus on erityisen tärkeää versiovaihtojen yhteydessä, jolloin järjestelmään tehdyt muutokset mahdollisesti rikkovat tai häiritsevät järjestelmässä aikaisemmin olleita toiminnallisuuksia. Regressiotestaus kohdistetaan vanhempiin järjestelmän osiin, joita ei ole muutettu ja joiden toimivuus halutaan varmistaa.

Automatisoitavat regressiotestit luotiin valmiina olevan manuaalisen regressiotestausohjeen pohjalta. Automatisoitujen regressiotestien piti kattaa järjestelmän testaus samoilta osin, kuin käsin tehtävien testien. Automatisoitujen regressiotestien kehitykseen ei annettu kohdeyrityksen puolesta vaatimuksia, eli toteutukseen annettiin vapaat kädet.

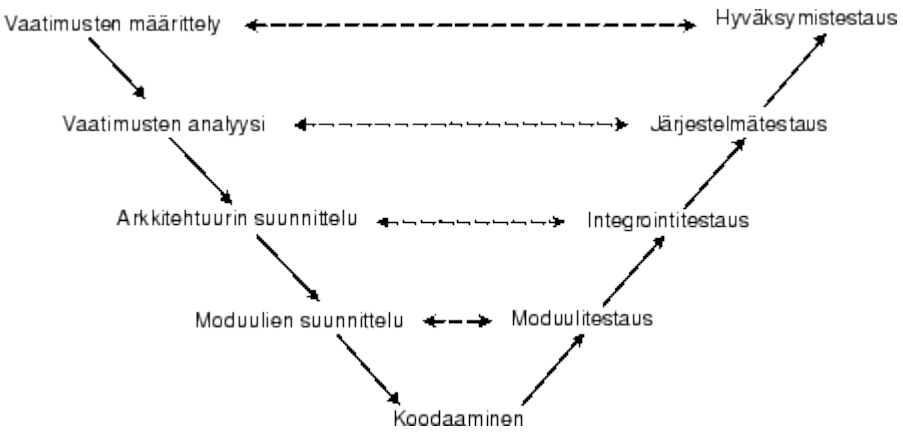
Opinnäytetyön tekijän tavoitteena oli luoda mahdollisimman helposti ymmärrettävät ja ylläpidettävät regressiotestit järjestelmälle. Testien helppoudesta huolimatta testien tuli kattaa järjestelmän tärkeimmät toiminnot hyvin. Opinnäytetyön tekijä tavoitteli myös testeille sellaista rakennetta, että ne voidaan ajaa aina riippumatta järjestelmän tilasta. Tällä tarkoitetaan testien rakentamista niin, että testit selviävät kaikista käyttötilanteiden variaatioista.

2 Testausmenetelmät

2.1 Ohjelmistotestauksen perusteet

Testaus on iso osa sovelluskehitystä. Ohjelmistotestausta on tehty niin kauan, kuin sovelluksia on tehty. Ohjelmistotestaus on keino todentaa toteutuksen virheettömyyttä, mutta testauksella ei kuitenkaan voida todistaa toteutuksen olevan täysin virheetön. Ohjelmistotestaus pienentää ohjelmistojen kehityskustannuksia ja auttaa mahdollisten ongelmien huomaamisessa kehityksen aikaisessa vaiheessa, ennen kuin on liian myöhäistä. [1].

Kuvassa yksi esitetään ohjelmistotestauksen vaiheistusta. Ohjelmistotestausta voidaan mallintaa V-mallilla, joka esittää, mihin testaus kannattaa kohdistaa eri toteutuksen vaiheissa.



Kuva 1. Ohjelmistotestauksen mallintaminen V-mallilla [2, s. 2].

V-malli sisältää ohjelmistotestauksen neljä testaustasoa, jotka keskittyvät järjestelmän testaamiseen eri alueilta. neljä testaustasoa ovat

- moduulitestaus (yksikkötestaus)
- integraatiotestaus
- järjestelmätestaus

- hyväksymistestaus.

Moduulitestaus tehdään itse kehitystyön alkuvaiheessa ja siinä keskitytään pieneen kokonaisuuteen esimerkiksi aliohjelmiin. Tämän testauksen tarkoituksena on varmistaa, että testattava moduuli varmasti toimii halutulla tavalla. Moduulitestaus suoritetaan yleensä kehittäjän toimesta. Moduulitestaus on erityisen tärkeässä roolissa ketterässä tuotekehitysprojektissa, jossa koontiversioita on usein. [3.]

Seuraavana testaustasona on integraatiotestaus. Integraatiotestaus keskittyy moduulien toimintaan yhdessä. Vaikka kaksi moduulia toimisivat yksinään täydellisesti, se ei tarkoita, että ne toimisivat oikein toistensa kanssa keskenään. Integraatiotestauksen tarkoituksena on varmistaa, että moduulit toimivat keskenään halutulla tavalla. Integraatiotestaus suoritetaan käyttäen tietynlaisia testauspolkuja, joiden läpi menemisellä voidaan todentaa moduulien toimivan oikein yhdessä. Integraatiotestauksessa voidaan käyttää kolmea menetelmää: top-down-menetelmä, Bottom-up-menetelmä ja näiden kahden menetelmän yhdistelmä. Top-down-menetelmällä tarkoitetaan moduulien yhdistämistä toisiinsa puumallina ylhäältä alaspäin. Bottom-up-menetelmä on identtinen muuten, mutta testaus edetään puumallissa alhaalta ylöspäin. Oikea testaustapa tulee valita jokaiselle järjestelmälle yksilöllisesti. [4, s. 2.]

Kolmas testaustaso on järjestelmätestaus. Yleisesti tämä on palveluntarjoajan viimeinen testaustaso. Järjestelmätestauksen tarkoituksena on varmistaa, että kaikki järjestelmän osat toimivat niin kuin on haluttu. Lisäksi järjestelmätestauksessa halutaan varmistaa, että järjestelmä täyttää kaikki sille asetetut tavoitteet ja vaatimukset. Järjestelmätestaus pitää sisällään myös testausta, joka ei suoraa ole toiminnallista. Tällaista testausta on esimerkiksi rasituksen sieto-, tietoturva-, stressin sieto-, käytettävyys-, suorituskyky-, asennus- ja luotettavuustestaus. [3.]

Neljäs testaustaso on hyväksymistestaus. Tämän testaustason toteuttajana toimii yleensä asiakas, jolloin asiakas hyväksyy toteutuksen toimivan halutulla tavalla. Joissain tapauksissa asiakas voi ulkoistaa hyväksymistestauksen kolmannelle taholle, mutta hyväksymistestauksen suorittajaksi kannattaa valita puolueeton taho. [3.]

2.2 Manuaali- ja automaatiotestauksen hyödyt

Ohjelmistosovelluksia voidaan testata monella tapaa. Kaikista perinteisin tapa on manuaalitestaus, joka tarkoittaa testaajan manuaalisesti tarkistavan toteutuksen.

Manuaalitestaus antaa kenties kattavimman lopputuloksen, mutta se vie enemmän aikaa ja resursseja testaajalta.

Automaatiotestauksella tarkoitetaan testausta, jonka suorittaa automaatio. Automaatiotestaus on hyvä valinta testauskohteisiin, jotka toistetaan useita kertoja tulevaisuudessa. Automaatiotestien kehittämiseen kuluva aika voitetaan moninkertaisena takaisin ajan saatossa. Tosin automaatiotesteissä voi päästä läpi sellaisia virheitä, jotka automatiikan mukaan eivät estä testin läpikäymistä. Tällaisia voivat olla esim. väri-, teksti- tai asemointivirheet. Tosin nämäkin virheet voidaan havaita automaatiotestauksella, mutta niihin varautuminen tulevaisuudessa on todella hankalaa.

Yhteenvetona hyvä testauskokonaisuus saadaan yhdistämällä manuaali- ja automaatiotestaus. Manuaalitestauksella saadaan aikaan tarkka lopputulos ja automaatiotestauksella voidaan suorittaa paljon työtä vaativia tehtäviä. Automaatiotestauksen suorittavalla testaajalla tulee olla ainakin yhtä hyvä järjestelmän tuntemus, kuin manuaalitestaajalla. Jos automaatiotestit keskeytyvät virhetilanteen takia, tulee testaajalla olla tietämystä ja taitoa arvioida, onko tilanne oikeasti virhetilanne vai onko testitapaus virheellinen.

Automaatiotestaus soveltuu erityisen hyvin regressiotestaukseen. Regressiotestaus suoritetaan tyypillisesti silloin, kun tuotekehityksessä on saatu päätökseen tarpeeksi iso kokonaisuus. Regressiotestauksella varmistetaan järjestelmän vanhojen osien toimivuus, kun uusi osa on saatu paikalleen. Koska regressiotesteillä testataan järjestelmän vanhaa osaa, voidaan testit rakentaa helposti etukäteen ja niitä voidaan ajaa tarvittavasti.

Automaatiotesteille on tärkeää se, että ne ovat helposti ylläpidettävät. Helposti ylläpidettävät automaatiotestit ovat tärkeitä tuotekehitysprojekteissa, sillä järjestelmä muuttuu jatkuvasti.

3 Regressiotestaukseen soveltuvat ohjelmistot

3.1 Työkalujen valinta

Automaatiotestien suorittamiseen tulee valita työkalut. Työn suorittamiseksi on valittava framework ja testieditorityökalut. Framework on itse testin suorittava sovellus, kun taas testieditori on testien kehitykseen ja hallintaan käytettävä apuväline.

Harkittuja frameworkohjelmia olivat Selenium, Robot Framework ja Watir. Nämä frameworkit soveltuvat työn suorittamiseen.

Watir on avoimeen lähdekoodiin perustuva framework. Watir erikoistuu selaimien automatisoimiseen. Watir tukee Internet Exploreria, Firefoxia, Chromea, Operaa ja Safaria. [5.]

Selenium on monipuolinen framework, jossa on työkaluja monipuoliseen testausautomaatioon. [6.]

Robot Framework on Linux-pohjainen framework. Robot Framework on myös monipuolinen framework, jolla voidaan automatisoida jopa selaimen ulkopuolella. [7.]

Työ päädyttiin tekemään Robot Frameworkilla, sillä se on yhteensopiva monen kirjaston kanssa ja täten mahdollistaa hyvän ylläpitämisen työn suorituksen jälkeen. Lisäksi Robot Framework on ennestään tuttu automaatiotestauksen peruskurssilta.

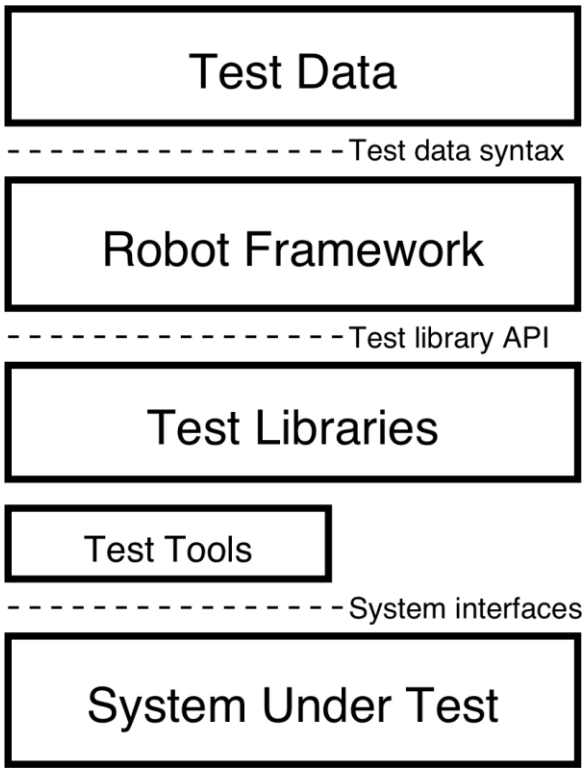
Testi editoriksi tulee valita työkalu. Robot Frameworkille on tarjolla monia testieditoreita, joista työhön harkittuja olivat TextMate, RIDE, PyCharm. [8.]

Laurent Bristiel on sovelluskehittäjä ja testaaja. Hän suositteli RIDE tekstieditoria kirjoitelmassaan ”What editor for Robot Framework test cases”. Hän kuvasi RIDEn olevan hyvä tekstieditori erityisesti ihmisille, jotka eivät ole kokeneita tekniikan kanssa.

Tämän takia työ päädyttiin tekemään RIDEllä. RIDE on hyvin graafinen ja aloittelijaystävällinen. RIDE oli myös tuttu automaatiotestauksen peruskurssilta.

3.2 Robot Framework

Robot Framework on pythonilla koodattu testiautomaatio-framework, joka soveltuu erityisesti hyväksymistestaukseen ja hyväksymistestauksen pohjalta suoritetun kehityksen tukemiseen. Robot Framework on hahmoteltu Pekka Klärckin lopputyön pohjalta 2005. Pekka Klärck on Robot Frameworkin kehittäjä. [7.]



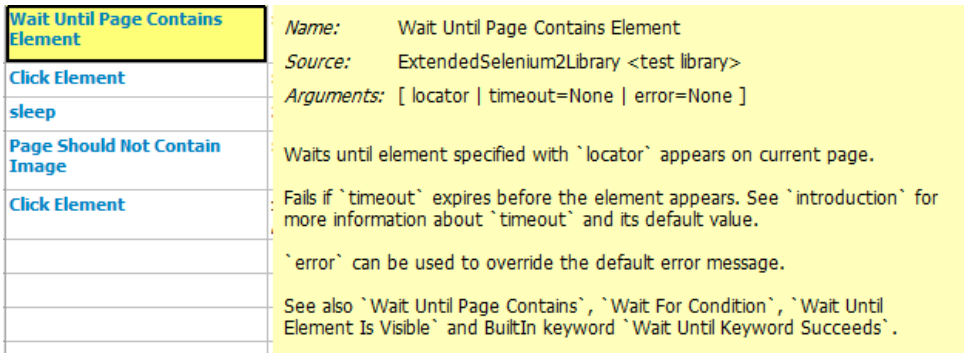
Kuva 2. Modulaarinen kuvaus Robot Frameworkin toiminnasta [9.]

Kuvassa 2 mallinnetaan Robot Frameworkin toimintaa. Robot Framework toimii välikätenä testidatan ohjeiden mukaan. Robot Framework käyttää mahdollisia siihen asennettuja kirjastoja ja työkaluja ja käsittelee testattavaa järjestelmää. Testidatalla tarkoitetaan itse testitapauksien kuvausta muodossa, jota Robot Framework osaa lukea. Testidataan on määritelty, mitä Robot Frameworkin kuuluu tehdä, löytää tai analysoida testikierroksen aikana. Hyvin luotu testidata neuvoo Robot Frameworkia antamaan virheen heti, jos järjestelmä menee virhetilaan.

3.3 Testirakenne

Automaatiotestit rakennetaan tekstieditorissa. Tässä tapauksessa tekstieditorina käytettiin RIDEä.

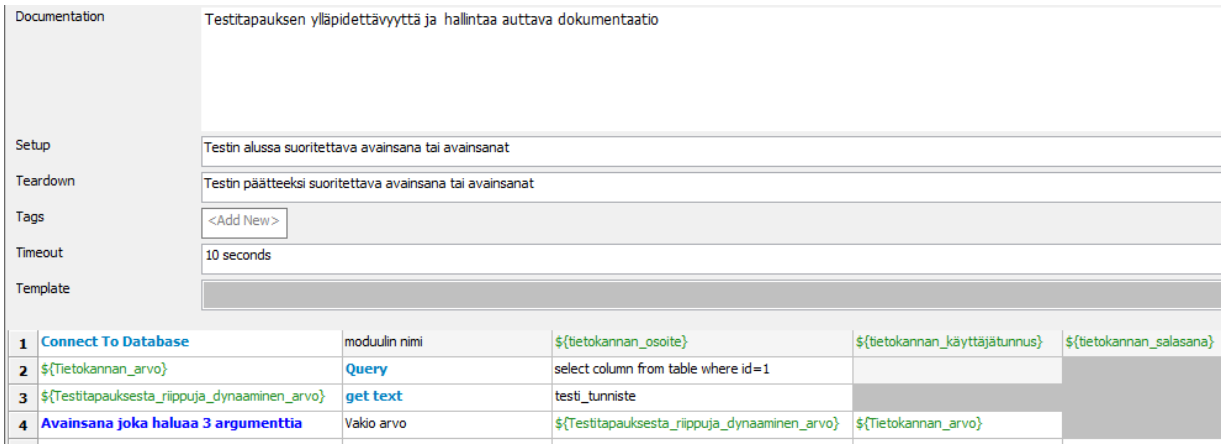
Automaatiotestit rakennetaan avainsanoista. Avainsanat ovat toimintoja tai tarkastuksia, jotka Robot Framework suorittaa. Robot Frameworkilla on tiettyjä sisäänrakennettuja avainsanoja, mutta avainsanoja pystyy asentamaan lisää myös valmiiden kirjastojen muodossa. Hyvin dokumentoiduissa kirjastoissa avainsanojen mukana tulee ohje, miten avainsana toimii.



Kuva 3. Esimerkkikuva avainsanasta ja avainsanasta saatavasta dokumentaatiosta.

Melkein kaikki avainsanat vaativat argumentteja. Argumentit ovat tilanteesta riippuen erilaisia tietoja, mitä avainsanoille voidaan antaa. Argumenttien perusteella avainsana tietää mitä sen halutaan tekevän missäkin tilanteessa. Avainsanojen argumentteina voi käyttää myös muuttujia.

Kuvassa 3 on kuvattu dokumentaatio avainsanasta ”Wait Until Page Contains Element”. Tähän avainsanaan on mahdollista syöttää kolme argumenttia, joista vain yksi on pakollinen. Pakollinen argumentti on ”locator”, joka kuvaa kohdetta. Robot Framework odottaa niin kauan, kunnes se havaitsee ”locator”-argumentissa kuvatun elementin. Toisena argumenttina on ”timeout”, jolla voidaan ylikirjoittaa testitapauksen normaali maksimiodotusaika. Jos testitapauksen maksimiodotusaika tai ”timeout” täyttyvät, testin suoritus keskeytyy ja testin tulokseksi tulee virhe. Kolmas argumentti on ”error”, jolla voidaan ylikirjoittaa testitapauksen normaali virheilmoitus.



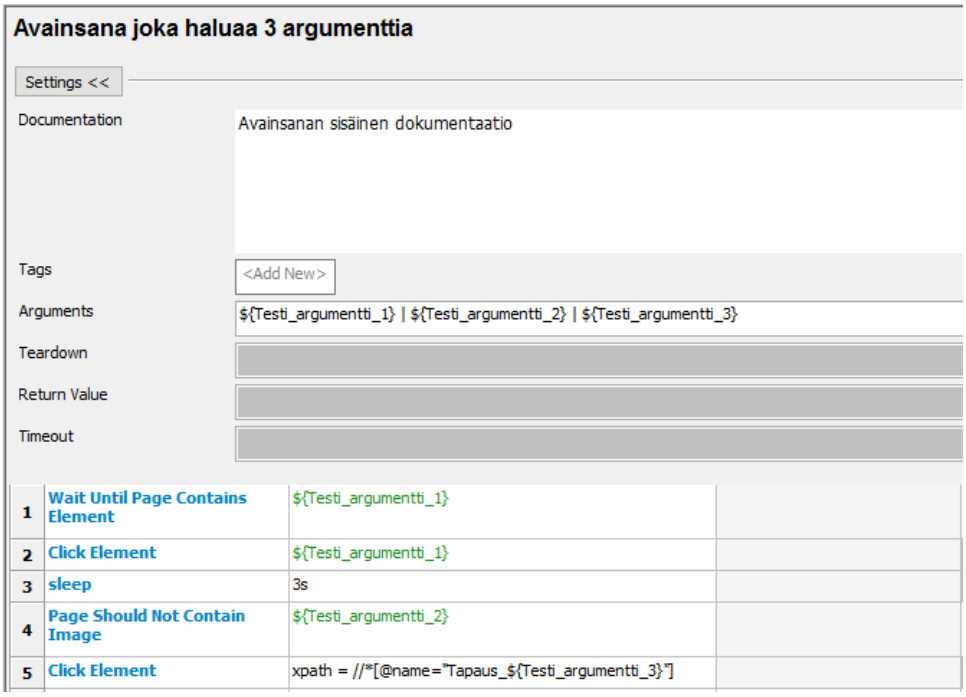
Kuva 4. Esimerkkikuva testitapauksen sisällöstä.

Kuvassa 4 on kolme kirjastoista saatua avainsanaa. Nämä ovat "Connect To Database", "Query" ja "Get Text". Lisäksi kuvassa 4 on yksi käyttäjän itse yhdistämä avainsana "Avainsana joka haluaa 3 argumenttia". Tämä avainsana koostuu useasta avainsanasta ja vaatii toimiakseen kolme argumenttia.

Testitapaukseen voi lisätä avainsanoja myös ”setup”- ja ”teardown”-kohtiin. ”setup”-kohtaan asetetut avainsanat suoritetaan aina testin aluksi. ”teardown”-kohtaan syötetyt avainsanat suoritetaan aina testin lopuksi, vaikka testi menisi virheeseen kesken suorituksen.

Testitapaukseen voidaan liittää erilaisia tageja, joita voi käyttää testejä ajaessa. Testien ajaja voi päättää esimerkiksi ajaa ainoastaan tietyllä tagilla merkityt testit.

”timeout” on maksimiaika, jonka Robot Framework odottaa ilman virhettä. Jos ”timeout” ylittyy, testin suoritus keskeytyy ja testin tulokseksi tulee virhe.



Kuva 5. Esimerkkikuva käyttäjän itse yhdistämästä avainsanasta.

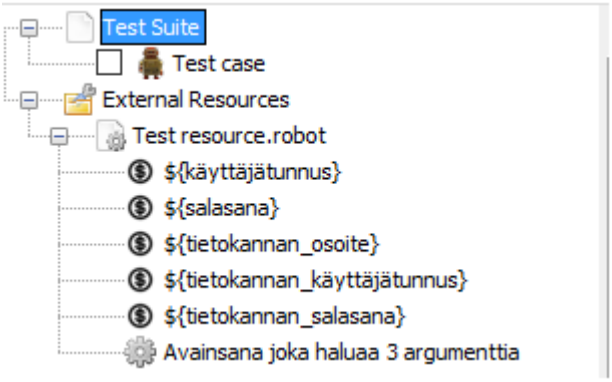
Kuvassa 5 näytetään, miltä käyttäjän yhdistämä avainsana voi näyttää. Avainsana on yhdistetty useammasta avainsanasta. Tässä tapauksessa yhdistetty avainsana on luotu ”Wait Until Page Contains”-, ”Click Element”-, ”Sleep”- ja ”Page Should Not Contain”-avainsanoista. Avainsana käyttää siihen syötettyjä argumentteja, joihin viitataan tässä tapauksessa muuttujina ${Testi_argumentti_1}, ${Testi_argumentti_2} ja ${Testi_argumentti_3}. Annetut argumentit voi syöttää suoraan argumenteiksi avainsanan sisällä oleviin avainsanoihin tai esimerkiksi osaksi XPathia. XPath on XML dokumenttien käsittelyssä käytettävä kieli.

Testit rakennetaan puumallina ylhäältä alaspäin. ylimpänä tasona on testikokoelma. Testitapaukset voidaan ajaa RIDEstä valitsemalla haluttu testitapaus tai testitapaukset ja käynnistämällä testit. Testien käynnistämisen jälkeen RIDE kutsuu Robot Frameworkia, joka alkaa heti suorittaa valittuja testitapauksia järjestyksessä. Jos Robot Framework pystyy suorittamaan testitapauksen loppuun, menee testi läpi. Jos Robot Framework ei pysty viemän testiä loppuun tai jokin testitapauksen tarkistuksista ei mennyt läpi, menee testi virheeseen ja seuraavan testin suoritus alkaa.

Testien ajamisen jälkeen käyttäjän on mahdollista nähdä loppuraportti, josta nähdään testien lopputulos. Loppuraportissa näkyy, onnistuiko vai epäonnistuiko testitapausten

suorittaminen. Lisäksi virheeseen menneet testit antavat testien kehityksen kannalta tärkeää tietoa siitä, miksi testi meni virheeseen.

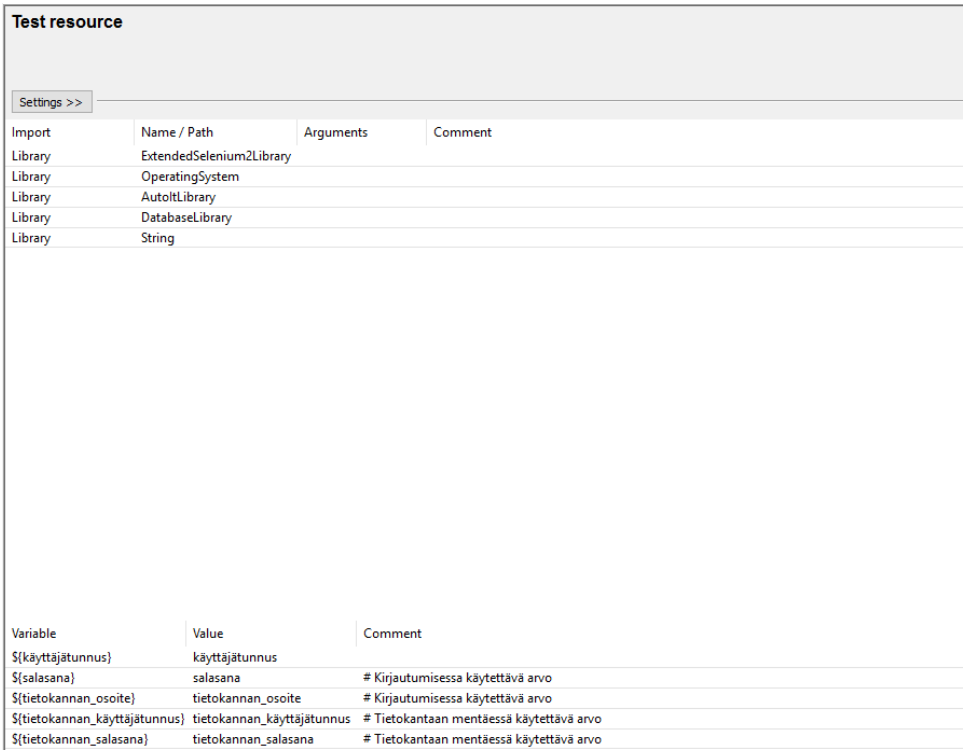
Kuvassa 6 esitetään testirakennetta. Testikokoelman sisälle mahtuu monia testitapauksia. Testitapaukset ovat pienempiä kokonaisuuksia, jotka pitävät sisällään erilaisia määriä avainsanoja. Tässä tapauksessa testikokoelman nimeksi tuli regressiotestit ja järjestelmän eri osat jaettiin testitapauksiksi testikokoelman alapuolelle.



Kuva 6. Esimerkkikuva Testirakenteesta

Kuvassa 7 esitetään resurssitiedoston sisältö. Avainsanoja, kirjastoja ja muuttujia voidaan pitää resurssitiedostoissa. Resurssitiedostoon voidaan viitata monesta testikokoelmasta, jolloin resurssin avainsanat, kirjastot ja muuttujat saadaan käyttöön tämän testikokoelman sisällä.

Tässä tapauksessa testiresurssin nimi on ”Test Resource”. Testiresurssi muodostuu viidestä muuttujasta, yhdestä avainsanasta ja kuudesta kirjastosta.



Test resource

Settings >>

Import	Name / Path	Arguments	Comment
Library	ExtendedSelenium2Library		
Library	OperatingSystem		
Library	AutoItLibrary		
Library	DatabaseLibrary		
Library	String		

Variable	Value	Comment
${käyttäjätunnus}	käyttäjätunnus	
${salasana}	salasana	# Kirjautumisessa käytettävä arvo
${tietokannan_osoite}	tietokannan_osoite	# Kirjautumisessa käytettävä arvo
${tietokannan_käyttäjätunnus}	tietokannan_käyttäjätunnus	# Tietokantaan mentäessä käytettävä arvo
${tietokannan_salasana}	tietokannan_salasana	# Tietokantaan mentäessä käytettävä arvo

Kuva 7. Esimerkkikuva resurssitiedoston sisällöstä

3.4 Kirjastot

Kirjastot ovat kokonaisuuksia, jotka sisältävät tietyn määrän avainsanoja. Robot Framework sisältää oman BuiltIn-kirjaston, joka sisältää laajan valikoiman avainsanoja esimerkiksi tietotyyppien ja muuttujien käsittelystä. BuiltIn-kirjasto ei yksinään tarjoa tarpeeksi kattavia avainsanoja työn suorittamiseen, joten työn suorittamiseksi kirjastoja on asennettava enemmän.

Yleisin kirjasto selainpohjaista testausta varten on Selenium2Library. Tämä kirjasto tarjoaa avainsanoja selaimella avattavien sivujen käsittelyyn [10]. Selenium2Library mahdollistaa:

- halutun selaimen avaamisen halutuilla selaimen asetuksilla
- halutulle verkkosivulle siirtymisen
- erilaisten tarkastusten asettamisen testille

- painikkeiden tai linkkien painamisen
- halutun selaimen sulkemisen

Työtä varten asennettiin ExtendedSelenium2Library-kirjasto. Tämä kirjasto on pääosiltaan identtinen Selenium2Library:n kanssa, mutta sisältää muutamia avainsanoja, joita ei löydy normaalista kirjastosta. Työn käytetyimmät avainsanat löytyvät ExtendedSelenium2Librarysta. (Liite 1)

Jotta Robot Framework osaa avata selaimen, se täytyy olla asennettuna työasemalle. Työasemalle täytyy myös asentaa halutun selaimen ajamiseen tarvittava webdriver. Työssä käytettiin Google Chromen WebDriveria. [11.]

Muut työssä käytetyt kirjastot:

- String
- OperatingSystem
- DatabaseLibrary
- AutoItLibrary.

Näiden kirjastojen avainsanoja käytettiin vain tapauksissa, missä Robot Frameworkin oma BuiltIn-kirjasto ja ExtendedSelenium2Library-kirjasto eivät tarjonneet tarvittavia avainsanoja.

String-kirjasto sisältää avainsanoja merkkijonojen muokkaamiseen. Tällä kirjastolla on mahdollista riisua muuttujassa olevasta merkkijonosta pois haluttuja merkkejä ja syöttää parsittu merkkijono uuteen muuttujaan tulevaa käyttöä varten. [12.]

OperatingSystem-kirjasto sisältää avainsanoja tiedostojen käsittelystä käyttäjän työasemalla. Tällä kirjastolla on mahdollista luoda, siirtää, poistaa, lukea ja muokata tiedostoja. [13.]

DatabaseLibrary-kirjasto mahdollistaa tietokantojen käyttämisen testeissä. Tällä kirjastolla saa yhteyden tietokantaan, jossa Robot Framework voi suorittaa haku-, päivitys- tai poistolauseita [14]. Kirjaston käyttäminen testitapauksissa vaatii erillisen lauseita suorittavan moduulin, jota Robot Framework kutsuu. Tähän käytettiin työssä pymssql-moduulia [15.]

AutoItLibrary-kirjasto mahdollistaa selaimen ulkopuolisten ikkunoiden käsittelyn. Tällä kirjastolla on mahdollista havaita ja käsitellä käyttäjälle aukiolevia sovelluksia.

4 Toteutetut regressiotestit

Riittävän kattavan testauksen lopputuloksena automatisoituja regressiotestitapauksia luotiin 10:

- Kirjautumistesti
- Käyttäjän tietojen muokkaus
- Tiedon 1 luontitesti
- Tiedon 2 luontitesti
- Admintestit
- Tiedon 1 luonnin tarkistus
- Tiedon 1 käsittely toisessa järjestelmässä
- Dokumentin 1 tarkistustesti
- Dokumentin 2 tarkistustesti
- Uuden käyttäjän luomistesti.

Jokaisen testin alussa suoritetaan vakiotoimenpide, jossa testin käyttämät vakioodotusajat määritetään ja avataan testiympäristö.

Kirjautumistestissä suoritetaan kirjautuminen ja uloskirjautuminen järjestelmään. Testiin kuuluu kirjautuminen useammalla eri käyttäjällä.

Käyttäjän tietojen muokkaus testi aloitetaan avainsanalla, joka on osa kirjautumistestiä. Tämä avainsana suorittaa ainoastaan kirjautumisosuuden kirjautumistestistä. Tämän avainsanan jälkeen testi päivittää ja peruuttaa päivitykset käyttäjän tiedoille. Päivitysten välissä tarkistetaan, onnistuiko päivitys.

Tiedon 1 luontitesti aloitetaan samalla kirjautumis-avainsanalla, kuin käyttäjän tietojen muokkaustesti. Kirjautumisen jälkeen käyttäjä luo tiedon 1 järjestelmään mahdollisimman pitkällä työnkululla testaten kaikki mahdolliset toiminnot matkan varrella. Tiedon 1 luonnin jälkeen käyttäjä kirjautuu ulos, ja testi toistetaan eriarvoisessa tilanteessa uudestaan. Luoduista tiedoista 1 otetaan talteen globaaliin muuttujaan tunnisteet, että tiedot voidaan tarkistaa oikeiksi tiedon 1 luonnin tarkistus testissä.

Tiedon 2 luontitesti aloitetaan kirjautumisavainsanalla. Kirjautumisen jälkeen käyttäjä poistaa mahdollisen tiedon 2, joka voi olla asetettuna tai poistettuna. Tämän jälkeen käyttäjä luo tiedon 2 järjestelmään mahdollisimman pitkällä työnkululla testaten kaikki mahdolliset toiminnot. Tiedon 2 luonnin jälkeen käyttäjä tarkistaa luoneensa tiedon löytyvät järjestelmästä oikeanlaisena. Luodun tiedon tarkistus toistetaan kolme kertaa eri arvoisella työnkululla.

Admintestit kohdistuvat järjestelmän pääkäyttäjien käyttöliittymään, josta voi hallita järjestelmää. Admintesteissä tarkistetaan pääkäyttäjien käyttöliittymän toimintojen toimivan halutunlaisesti.

Tiedon 1 luonnin tarkistustesti aloitetaan kirjautumalla sisään samana käyttäjänä, joka aloitti tiedon 1 luonti testin. Testi käyttää globaalia muuttujaa, joka luotiin tiedon 1 luonnin yhteydessä. Muuttujan arvoa käytetään todentamaan tarkistettavan tiedon olevan aikaisemmassa testitapauksessa luotu tieto 1. Tieto 1:n tarkistetaan olevan oikeanlainen ja lopulta tieto 1 poistetaan. Käyttäjä kirjautuu ulos. Uloskirjautumisen jälkeen kirjaudutaan sisään toisella käyttäjällä, joka oli myös luonut tiedon 1. Tiedon 1 tarkistus ja poisto suoritetaan myös toisella käyttäjällä käyttäen toisen käyttäjän luomaa tiedon 1 uniikkia arvoa, joka on tallessa globaalissa muuttujassa.

Tiedon 1 käsittely toisessa järjestelmässä testi aloitetaan tiedon 1 luonti testillä. Tietojen luonnin jälkeen testaus siirtyy toiseen järjestelmään, jossa tietoa 1 käsitellään. Regressiotestit eivät kohdistu toiseen järjestelmään, eli testit suoritetaan mahdollisimman nopeasti ilman mahdollisten työnkulkujen kokeilua. Kun tieto 1 on käsitelty, siirrytään takaisin

alkuperäiseen järjestelmään. Tiedon 1 luonut käyttäjä kirjautuu sisään järjestelmään, ja tarkistaa tiedon 1 olevan oikeanlainen käsittelyn jälkeen. Tiedon käsittely toisessa järjestelmässä toistetaan myös toisen tiedon 1 osalta. Tämä tieto 1 käsitellään eri tavalla. Käsittelyn jälkeen alkuperäisessä järjestelmässä tarkistetaan käsitellyn tiedon olevan oikeanlainen.

Molemmat dokumentin tarkistustestit avaavat järjestelmässä olevat dokumentit ja tarkistavat niiden sisällön. Testi menee virheeseen, jos dokumenttien sisältö ei vastaa haluttua sisältöä.

Uuden käyttäjän luomistestissä otetaan yhteys järjestelmän tietokantaan. Tietokannassa suoritetaan muutama hakulause, jotka varmistavat tulevan päivityslauseen koskevan ainoastaan yhtä riviä tietokannassa. Jos hakulauseet palauttavat väärän arvon, menee testi virheeseen. Jos hakulauseet menevät läpi, suoritetaan päivityslause. Päivityslause rikkoo testattavan käyttäjän, jolloin se unohtuu järjestelmän näkökulmasta. Kirjaudutaan sisään käyttäjälle, joka rikottiin päivityslauseella. Järjestelmä ei tunnista käyttäjää ja käynnistää uuden käyttäjän luomisen. Käyttäjän luomissivu testataan läpi mahdollisen pitkällä työnkululla testaten kaikki mahdolliset toiminnot. Lopuksi käyttäjä luodaan ja tarkistetaan käyttäjän tiedot olevan oikein.

Testit luotiin sellaisiksi, että ne voidaan ajaa riippumatta siitä, missä tilassa järjestelmä on. Esimerkiksi jos testin käyttämän käyttäjän tiedot eivät ole testin suorittamiseen vaaditulla tasolla, käy testi luomassa tai päivittämässä käyttäjän tiedot tarvittavalle tasolle. Näin kehitetyt testit helpottavat testien ajoa ja ylläpidettävyyttä.

5 Loppupohdinta

Työn tavoitteena oli automatisoida käsin ajettavat regressiotestit niin kattavasti, kuin automaatiolla on mahdollista. Mielestäni onnistuin työssä kiitettävästi, sillä tehdyt automaatiotestit kattavat järjestelmän toiminnallisuudet ainakin yhtä hyvin, kuin alkuperäinen testausohje. Olen tyytyväinen työn laatuun, vaikka tiedän hiottavaa aina löytyvän.

Jos tekisin työn uudestaan, keskittyisin enemmän testien rakenteen suunnitteluun. Vaikka suunnittelin testien kokonaisuuden etukäteen, huomasin jälkeenpäin, että tarkemmalla suunnittelulla testeistä olisi voinut tehdä helpommin ylläpidettävät. Esimerkiksi osassa testejä luotiin sama toiminnallisuus hieman erilaisena kahteen eri avainsanaan. Tällainen toimintamalli ei ole kannattavaa, sillä kun testitapaukseen tulee muutos johtuen järjestelmän päivittämisestä, joudutaan tämä toiminnallisuus korjaamaan useampaan paikkaan. Tällaista toimintamallia voidaan välttää rakentamalla avainsanaan kaksi mahdollista samankaltaista toimintoa ja asettamalla avainsanalle argumentti, joka kertoo Robot Frameworkille, kumman toiminnon toteuttaa.

Opinnäytetyötä aloittaessa en ollut kokenut automaatiotestien tekijä. Olin käynyt Robot Framework-peruskurssin, ja perehtynyt kurssimateriaaliin syvemmin omasta mielenkiinnosta. Työtä tehdessä opin suurimman osan automaatiotestien kehitykseen liittyvistä taidoistani.

Työn tuloksesta voi päätellä, että automaatiotestien tekemiseen käytettävä aika saadaan takaisin monikertaisena tulevaisuudessa. Testien suorittaminen manuaalisesti voi viedä tunteja, mutta automaatiolla testit ajetaan läpi kymmenissä minuuteissa. Lisäksi automaatio mahdollistaa testien ajamisen ilman henkilötyötä.

Testejä voisi kehittää perustoimintojen toimivuuden testaamista pidemmälle lisäämällä testitapausten sisälle lisää tarkistuksia selaimen sivun elementteihin ja teksteihin. Lisätarkastuksina voisi olla esimerkiksi tarkistuksia siitä, mitä elementtejä eri testin vaiheissa kuuluisi olla aktiivisena. Lisäksi testitapauksiin voisi lisätä tagit. Tagit ovat sanoja, joita voidaan käyttää ehtona testien ajamiseen. Esimerkiksi testejä ajettaessa voidaan päättää ajaa ainoastaan kirjautumistestit. Tässä esimerkkitapauksessa kaikilla kirjautumiseen liittyvillä testeillä on ”kirjautuminen” tagina. Testikomentosarjat voisi myös liittää osaksi versionhallintaa. Tällöin testit olisivat helposti ajettavissa sovelluskehittäjien toimesta. Sovelluskehittäjät voisivat esimerkiksi ajaa kaikki kirjautumiseen liittyvät testit, jos

he haluavat varmistaa kirjautumisen toimivan vielä heidän järjestelmään tekemänsä muutoksen jälkeen.

Lähteet

1 Kääriäinen Markus. Knowitin palveluiden esittelysivu. Verkkoaineisto. <https://www.knowit.fi/palvelut/solutions/laadunvarmistus/>. Luettu 14.11.2017

2 Myyry Jani. Ohjelmistoliiketoiminnan ja -tuotannon laboratorio SoberIT. Verkkoaineisto. <http://www.soberit.hut.fi/T-76.115/01-02/palautukset/groups/Confuse/lu/docs/vmalli/index.html>. Luettu 14.11.2017

3 Tietoa testaustasoista. Verkkoaineisto. <http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus/testauksen-tasot>. Luettu 16.4.2018

4 Tietoa testaustasoista. Verkkoaineisto. <https://www.cs.helsinki.fi/group/koski/doc/testaus.pdf>. Luettu 16.4.2018

5 Taustatietoa Watirista. Verkkoaineisto. <https://en.wikipedia.org/wiki/Watir>. Luettu 16.4.2018

6 Taustatietoa Seleniumista. Verkkoaineisto. <https://www.seleniumhq.org/docs/01_introducing_selenium.jsp#introducing-selenium>. Luettu 16.4.2018

7 Taustatietoa Robot Frameworkista. Verkkoaineisto. <https://en.wikipedia.org/wiki/Robot_Framework#cite_note-2>. Luettu 16.4.2018

8 Bristiel Laurent. Kokemuksia Robot Framework testi editoreista. Verkkoaineisto. <http://laurent.bristiel.com/what-editor-for-robot-framework-test-cases/>. Luettu 27.11.2017

9 Kuvaus Robot Frameworkin toiminnasta. Verkkoaineisto. <http://robotframework.org/>. Luettu 16.4.2018

10 Selenium2Libraryn datasivu. Verkkoaineisto. <http://robotframework.org/Selenium2Library/Selenium2Library.html>. Luettu 7.1.2018

11 Google Chromen WebDriver. Verkkoaineisto. <https://sites.google.com/a/chromium.org/chromedriver/downloads>. Luettu 7.1.2018

12 String kirjaston datasivu. Verkkoaineisto. <http://robotframework.org/robotframework/latest/libraries/String.html>. Luettu 7.1.2018

13 OperatingSystem kirjaston datasivu. Verkkoaineisto. <http://robotframework.org/robotframework/latest/libraries/OperatingSystem.html>. Luettu 7.1.2018

14 DatabaseLibrary kirjaston datasivu. Verkkoaineisto. <https://franz-see.github.io/Robotframework-Database-Library/api/0.5/DatabaseLibrary.html>. Luettu 7.1.2018

15 Pymssql info sivu sivu. Verkkoaineisto. <http://www.pymssql.org/en/stable/>. Luettu 7.1.2018

<table>
<tr>
<td>Click Button</td>
<td>Locator, skip_ready=False</td>
<td>Clicks a button identified by <code>locator</code>.
Arguments:
• <code>locator</code>: The locator to find requested button. Key attributes for arbitrary buttons are <code>id</code>, <code>name</code>, and <code>value</code>. See introduction for details about locating elements.
• <code>skip_ready</code>: A boolean flag to skip the wait for page ready. (Default False)
Examples:
<table>
<tr><td>Click Button</td><td>css=button.class</td><td></td></tr>
<tr><td>Click Button</td><td>css=button.class</td><td>True</td></tr>
</table>
</td>
</tr>
<tr>
<td>Click Element</td>
<td>Locator, skip_ready=False</td>
<td>Clicks an element identified by <code>locator</code>.
Arguments:
• <code>locator</code>: The locator to find requested element. Key attributes for arbitrary elements are <code>id</code> and <code>name</code>. See introduction for details about locating elements.
• <code>skip_ready</code>: A boolean flag to skip the wait for page ready. (Default False)
Examples:
<table>
<tr><td>Click Element</td><td>css=div.class</td><td></td></tr>
<tr><td>Click Element</td><td>css=div.class</td><td>True</td></tr>
</table>
</td>
</tr>
<tr>
<td>Click Link</td>
<td>Locator, skip_ready=False</td>
<td>Clicks a link identified by <code>locator</code>.
Arguments:
• <code>locator</code>: The locator to find requested link. Key attributes for arbitrary links are <code>id</code> and <code>name</code>. See introduction for details about locating elements.
• <code>skip_ready</code>: A boolean flag to skip the wait for page ready. (Default False)
Examples:
<table>
<tr><td>Click Link</td><td>css=a.class</td><td></td></tr>
</table>
</td>
</tr>
</table>

		Click Link | css=a.class | True
Close Browser		Closes the current browser.
Element Should Contain	*locator, expected, message=*	Verifies element identified by locator contains text expected. If you wish to assert an exact (not a substring) match on the text of the element, use Element Text Should Be. message can be used to override the default error message. Key attributes for arbitrary elements are id and name. See introduction for details about locating elements.
Element Should Not Contain	*locator, expected, message=*	Verifies element identified by locator does not contain text expected. message can be used to override the default error message. Key attributes for arbitrary elements are id and name. See Element Should Contain for more details.
Element Text Should Be	*locator, expected, message=*	Verifies element identified by locator exactly contains text expected. In contrast to Element Should Contain, this keyword does not try a substring match but an exact match on the element identified by locator. message can be used to override the default error message. Key attributes for arbitrary elements are id and name. See introduction for details about locating elements.

Get Matching Xpath Count	*xpath*	Returns number of elements matching xpath
		One should not use the xpath= prefix for 'xpath'. XPath is assumed.
		Correct:
		`count = | Get Matching Xpath Count | //div[@id='sales-pop']`
		Incorrect:
		`count = | Get Matching Xpath Count | xpath=//div[@id='sales-pop']`
		If you wish to assert the number of matching elements, use Xpath Should Match X Times.

<table>
<tr><td>Get Text</td><td>locator</td><td>Returns the text value of element identified by locator.
See introduction for details about locating elements.</td></tr>
<tr><td>Go To</td><td>url</td><td>Navigates the active browser instance to the provided URL.</td></tr>
<tr><td>Input Password</td><td>locator, text</td><td>Types the given password into text field identified by locator.
Difference between this keyword and Input Text is that this keyword does not log the given password. See introduction for details about locating elements.</td></tr>
<tr><td>Input Text</td><td>locator, text</td><td>Types the given text into text field identified by locator.
See introduction for details about locating elements.</td></tr>
<tr><td>Open Browser</td><td>url, browser=firefox, alias=None, remote_url=False, desired_capabilities=None, ff_profile_dir=None, skip_ready=False</td><td>Opens a new browser instance to given URL. Returns the index of this browser instance which can be used later to switch back to it. Index starts from 1 and is reset back to it when Close All Browsers keyword is used. See Switch Browser for example.
Optional alias is an alias for the browser instance and it can be used for switching between browsers (just as index can be used). See Switch Browser for more details.
Possible values for browser are as follows:
<table>
<tr><th>firefox</th><th>FireFox</th></tr>
<tr><td>ff</td><td>FireFox</td></tr>
<tr><td>internetexplorer</td><td>Internet Explorer</td></tr>
<tr><td>ie</td><td>Internet Explorer</td></tr>
<tr><td>googlechrome</td><td>Google Chrome</td></tr>
<tr><td>gc</td><td>Google Chrome</td></tr>
<tr><td>chrome</td><td>Google Chrome</td></tr>
<tr><td>opera</td><td>Opera</td></tr>
<tr><td>phantomjs</td><td>PhantomJS</td></tr>
<tr><td>htmlunit</td><td>HTMLUnit</td></tr>
<tr><td>htmlunitwithjs</td><td>HTMLUnit with Javascipt support</td></tr>
<tr><td>android</td><td>Android</td></tr>
<tr><td>iphone</td><td>Iphone</td></tr>
<tr><td>safari</td><td>Safari</td></tr>
</table></td></tr>
</table>

		Note, that you will encounter strange behavior, if you open multiple Internet Explorer browser instances. That is also why Switch Browser only works with one IE browser at most. For more information see: http://selenium-grid.seleniumhq.org/faq.html#i_get_some_strange_errors_when_i_run_multiple_internet_explorer_instances_on_the_same_machine Optional 'remote_url' is the url for a remote selenium server for example http://127.0.0.1/wd/hub. If you specify a value for remote you can also specify 'desired_capabilities' which is a string in the form key1:val1,key2:val2 that will be used to specify desired_capabilities to the remote server. This is useful for doing things like specify a proxy server for internet explorer or for specify browser and os if your using saucelabs.com. 'desired_capabilities' can also be a dictonary (created with 'Create Dictionary') to allow for more complex configurations. Optional 'ff_profile_dir' is the path to the firefox profile dir if you wish to overwrite the default.
Page Should Contain	*text, loglevel=INFO*	Verifies that current page contains text. If this keyword fails, it automatically logs the page source using the log level specified with the optional loglevel argument. Valid log levels are DEBUG, INFO (default), WARN, and NONE. If the log level is NONE or below the current active log level the source will not be logged.
Page Should Not Contain	*text, loglevel=INFO*	Verifies the current page does not contain text. See Page Should Contain for explanation about loglevel argument.
Wait Until Page Contains Element	*locator, timeout=None, error=None*	Waits until element specified with locator appears on current page. Fails if timeout expires before the element appears. See introduction for more information about timeout and its default value. error can be used to override the default error message.

		See also Wait Until Page Contains, Wait For Condition, Wait Until Element Is Visible and BuiltIn keyword Wait Until Keyword Succeeds.