TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarainstituut

Eljus Šipovski 134603IAPB

AUTOMATISEERITUD TESTIDE TÖÖKINDLUSE SUURENDAMINE PYTHON ROBOT FRAMEWORKI NÄITEL

Bakalaureusetöö

Juhendaja: Maili Markvardt
MSc

Tallinn 2017

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Eljus Šipovski

25.03.2017

Annotatsioon

Eesmärgiks on võrrelda ja uurida automaattestide meetmeid Robot Frameworki kaudu. Uuritakse, milline testikirjutusviis on optimaalsem ja tõhusam. Robot Framework raamistikule kirjutatakse ligikaudu 100 funktsionaalset testi, mis hõlmavad 90% kasutaja veebilehitsemisest. Objektiivsuse mõttes testitakse veebilehte kolmes brauseris: Google Chrome, Mozilla Firefox ja Internet Explorer.

Töö tulemuseks on saada töökorras veebitestid vältides dünaamiliste muutujate kasutamist.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 7 peatükki, 12 joonist, 1 tabelit.

Abstract

Increasing reliability of automated tests with Python Robot Framework

Software testing is a proccess of executing a program to ensure it is working as expected. Nowadays, testing is one of the main requirements for creating any program or application. Testing can reduce large amounts of time spent on development and organisation, by finding bugs as soon as possible, which would reduce the cost of debugging at later stages.

The purpose of this thesis is to write tests for the Top Connect company, to find the best way of writing tests with Robot Framework, to compare how browsers run tests and to find the best way to generate an Xpath. Tests were written in different types - functional, non-functional and regression testing. All tests were conducted in these three browsers - Google Chrome, Mozilla Firefox and Internet Explorer. The usage of different browsers gave the overview of the web application's functionality. By using Python and Selenium, tests can be performed in Robot Framework, which makes running tests easy. The automation code was built so as to avoid any dynamic variables, thus, after new deployment, tests could be carried out without any fails.

As a result, we got information on how long it takes to conduct all tests on different browsers, which browsers had problems running them, how to find the best Xpath, how to use different types of testing and why it is better to avoid dynamic variables.

Summaring results, 44 functional tests were written. Out of the three browsers mentioned, Google Chrome was the fastest and ran as expected, Internet Explorer was too slow to write tests on, Mozilla Firefox didn't work on Selenium 3.0 and almost all tests that used dynamic variables didn't work after next deployment.

The thesis is in Estonian and contains 30 pages of text, 7 chapters, 12 figures, 1 table.

Lühendite ja mõistete sõnastik

BDD	*Behavior-driven development*
HTML	*HyperText Markup Language*. Hüperteksti märgistuskeel.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Testimine on tarkvara kvaliteedi uurimine, mis loob kindluse, et tarkvara vastab nõuetele ning töötab õigesti. Tarkvaratestimine peab olema efektiivne vigade leidmisel, kuid samal ajal ka aega säästev. Üheks testimise meetodiks on manuaalne testimine, kuid see nõuab palju ressursse. Töökindluse jaoks asendatakse osa manuaalset tööd automatiseerimisega. Automatiseeritud testid vähendavad aega, mis võimaldab suurendada testimisskoopi ja täpsust. Testid võivad joosta minuteid, mille tavalisele manuaalsele testimisele võib minna mitu tundi. Põhiliseks erinevuseks, miks eelistatakse automaatteste manuaalsetele testidele, on nende käideldavus. Automaattestid võivad joosta öösel ning ei vaja inimeste juuresolekut. [17]

Automaattestide raamistikud on kujunenud välja pika aja jooksul ning nende platvormide maht on väga suur. Iga platvorm sisaldab erinevaid meetodeid, mis lihtsustavad koodi kirjutamist. Visuaalsetel platvormidel on eelis koodist arusaamisel ning koodikiirusel, kuid testide koodikirjutamise meetodid ja võimalused on väga piiratud. Mõned veebilehed on kirjutatud keeruliselt, mis võib põhjustada raskusi neile testide koodi kirjutamisel. Üheks näiteks on kui veebileht on ülesehitatud dünaamiliste muutujate klassidega. Dünaamiliste muutujate vältimiseks on vaja väga hästi tunda veebilehe struktuuri ning selle kõiki võimalikke ohte. Testimiseks võeti kasutusse automaattestimise platvorm Python Robot Framework. Python Robot Framework kasutab lihtsat sõnalist koodi (märksõnade koodi). Dünaamiliste muutujate vältimiseks on oluline Xpathi põhjalik tundmine. [11]

1.1 Eesmärgid

Antud lõputöö eesmärkideks on:

1. Luua automatiseeritud testid ettevõttele Top Connect
2. Leida parim viis automatiseeritud testide kirjutamiseks Robot Frameworki näitel
3. Uurida ja võrrelda automatiseeritud testide töökindlust erinevatel brauseritel

4. Leida parim viis testides Xpathi kasutamiseks

1.2 Ülesandepüstitus

Testide kirjutamiseks kasutatakse Robot Frameworki, et uurida, kuidas automaattestid mõjutavad infosüsteemi terviklikkust erinevate testimisfaaside vältel. Eesmärgiks on saada automaattestid, mis katavad infosüsteemi funktsionaalsust vältides dünaamilisi muutujaid ning mis vähendavad manuaalset testimist.

Testimise käigus uuritakse, kuidas mõjutab testimist erinevat tüüpi Xpathide kasutamine – dünaamilised muutujad, staatilised muutujad. Uuritakse, miks pikki Xpathe pole mõistlik kasutada. Uurimise alla lähevad ka erinevad veebibrauserid – Google Chrome, Internet Explorer ning Mozilla Firefox. Nende põhjal uuritakse testide kiirust ning käideldavust.

2 Testimise meetodid

2.1 Testimise alused

Hea automatiseerija ei ole see, kes kirjutab ilusa koodi ning katab tervet testitavat veebirakendust, vaid see, kes teab kõige parimini testimise aluseid – kuidas on parim viis testida, mis on tähtis, mis võib olla probleemiks, mida testida rohkem, testimise stsenaariumid jne. Just sellepärast on igal testijal vaja teada põhja, mille alusel saab kirjutada kõrgekvaliteedilist testikoodi. Kvaliteedi testimise ajal on vaja kindlaks määrata, millega on tegu – kas tarkvara on seotud panga, mängu, blogi või mingi platvormiga. Iga süsteem vajab erinevat lähenemist. Kui blogi testimise ajal põhirõhk on kasutajamugavus, siis pangasüsteemis peab testija jälgima kogu läbikäidavat informatsiooni ja selle kaitset. [11]

Efektiivseks testimiseks on loodud seitse põhiprintsiipi, mis on loodud ja soovitatud ligi 40 aastat järjest. Iga printsiip on omakorda unikaalne ning keerukusega näidatakse, miks iga printsiip on tähtis. Printsiibid on järgmised [11]:

1) Testimine näitab vaid defektide olemasolu – „testimine näitab, et defektid on olemas, kuid ei suuda tõestada, et nad puuduvad.“ [11]

2) „Täielik testimine on võimatu – pole võimalik testida kõike“ [11]. On olemas lõpmatu arv võimalikke testimise võimalusi, mida ei saa kunagi katta, kuid võimalik on testida kõige vajalikumad. Testimise käigus on vaja võtta endale risk, et midagi jääb testimata. Just sellepärast on vaja moodustada testimisplaan, mis võimaldab katta kõige tähtsamaid funktsionaalsusi.

3) Varajane testimine – „mida varem testitakse süsteemi, seda vähem peab pärast muutma koodi, mis säästab nii töökoormust kui ka rahalist investeeringut“ [11]. Üks populaarsematest vigadest, mida ettevõtted teevad, on testimise puudumine. Arvatakse, et testimine on väga kallis ning sellel puudub igasugune mõte. Kuid alles projekti lõpus nähakse, mida võiks tuua testimine ning kui palju võiks kokkuvõttes säästa testijaid palgates.

4) Defektide klasterdumine – „suurem osa defektidest, mis on leitud testimise käigus, sisalduvad kindlates moodulites.“ [11]

5) Pestitsiidi paradoks – testides ühte ja sama asja mitmeid kordi erinevate faaside käigus väheneb võimalus leida uusi defekte, kuna tehakse läbi vaid üht funktsionaalsust. Paradoksist lahtisaamiseks tuleb kasutada sama testi ning lisada iga kord uusi funktsionaalsusi. Iga uus testimismeetod ja funktsionaalsus annab võimaluse leida uut defekti läbides funktsionaalsust uue mooduse abil (joonis 1). Näide pestitsiidi paradoksist selle bakalaureusetöö testitava süsteemi näitel: kontakti on võimalik luua kahel viisil – ülemise paneelil „*Create Contact*“ või „*Account*“ paneeli kaudu. Iteratsiooni käigus leiti, et läbi ülemise „*Create Contact*“ paneeli oli võimalik lisada selliseid kontakte, kes ei olnud nähtavad peavaates, mida saavad vaadata kõik juhid. Peale defekti kõrvaldamist defekt ikkagi eksisteeris, kuna lahendati ainult antud probleemi. Leiti, et kontakti lisamine läbi alampaneeli ei salvestanud kontakti peavaatele kui ka lisas uusi defekte, mis hõlmasid kogu süsteemi.

Faas 1					Faas 2					Faas 3				
Defineerimine	Arendamine	Ehitamine	Testimine	Implementeerimine	Defineerimine	Arendamine	Ehitamine	Testimine	Implementeerimine	Defineerimine	Arendamine	Ehitamine	Testimine	Implementeerimine

Tabel 1. Iteratsiooni tabel [12]

6) Testimine sõltub kontekstist – eelnevalt mainiti, et iga süsteem nõuab enda analüüsimist, kuidas testida konkreetset süsteemi. Süsteemid jagunevad kriitilisuse alusel erinevateks tüüpideks, mis määravadki testimise nõudmisi.

7) Vead puuduvad – ei eksisteeri ühtegi süsteemi, milles defektid puuduksid. Veebirakendused sõltuvad väga paljudest kriteeriumidest ning funktsionaalsustest, mis on väga keerulised. Sellepärast määratakse projekti alguses skoop, mis selgitab, missugused vead tuleb leida ja maandada.

2.2 Testimise etalonprotsess

Enne igat testimist on vajalik koostada testimise plaan ning kooskõlastada testimise korraldus. Mõlemad aitavad koostada visiooni, kuidas toimub testimine antud iteratsioonis. Kõik valikud, mida tehakse, sõltuvad testitavast süsteemist ning resurssidest. Fundamentaalse tesimise protsesse saab jaotada järgnevate sammudega [11]:

1. Plaanimine ja kontrollimine
2. Analüüsimine ja disainimine
3. Implementeerimine ja teostamine
4. Hinnang ja raporteerimine
5. Testide lõpetamine

Kõik sammud on loogiliselt ehitatud tavalisele projektile, kuid sammud võivad muutuda või korduda. [11]

2.2.1 Plaanimine ja kontrollimine

Selle etapi ajal pannakse paika, mida klient tahab, kuidas ta seda tahab, mis on tulemiks ja mis riskid aktsepteeritakse testimise käigus. Vastuste kujunemisel pannakse paika kindel visioon ning strateegia, kuidas näevad välja testid ja kuidas nendega toime tulla. Näiteks luuakse testimise protokoll ja list (*check list*), mida testija läbib kindla ajavahemikuga. Järgmised ülesanded aitavad koostada testimise plaani [11]:

- Tuleb läbi mõelda, mis on testimise skoop, mis on nendega kaasnevad riskid ja kuidas neid vältida. Suuremas osas on vaja analüüsida, missugused komponendid tuleb kasutusele võtta – süsteemid, äriplaan, riskid jt.
- Määratakse testide käsitlemine – mida tuleb kasutada testimiseks (protokollid, programmid), missugused testimismeetodeid tuleb kasutada antud projektis. Lisaks testimise programmidele tuleb analüüsida töötajate kasutamist – kes sobib parimini, kui palju inimesi on vaja, kuidas valida nii, et meeskonnatöö oleks maksimaalselt efektiivne. Inimeste valimine toimub kindlate kriteeriumite ning osakaalu valikul. Hea meeskond, mis koosneb erinevate oskustega inimestest, suudab katta suurt osa testimise süsteemist kõige parimini.

- Määratakse testimise käitumine – kuidas tehakse testimist, mis osadena, millal tehakse konkreetset liiki testimist. Kõik see on vajalik, et testitav süsteem oleks tähtajaliselt lõpetatud. Kui arvatakse, et testimist ei suudeta õigeajaliselt lõpetada, arutatakse eelmist punkti uuesti. Mõeldakse läbi, kui suur osakaal inimesi on tarvis projekti.

- Mõeldakse läbi lõpetamine. Lõpetamine on vajalik statistikaks ja aja planeerimiseks. Projekt omab kindlaid ajavahemikke ning selleks on vaja kindlaks määrata, kas testimine on lõpetatud või mitte.

2.2.2 Analüüsimine ja disainimine

Analüüsimise ja disainimise käigus viiakse testimisprotokollist ellu testimise tingimused ja testimise stsenaariumid, millel on järgnevad ülesanded [11]:

- Testitava süsteemi ülevaade
- Määratakse testitava süsteemi nõuded
- Pannakse paika testimese käigus prioriteedid
- Pannakse paika ning otsitakse välja kogu vajalik informatsioon
- Määratakse kõik vajalikud infosüsteemid, mida kasutatakse testide loomisel
- Määratakse vajalike süsteemide kasutamine

2.2.3 Implementeerimine ja teostamine

Selles etapis kogutakse kogu informatsioon testide kirjutamiseks, pannakse käima testitav süsteem ning kirjutatakse teste. Testide kirjutamiseks kasutatakse loodud spetsifikatsiooni ning kirjutatakse vaid kõige tähtsamad osad, mida oleks vaja testida peale igat süsteemiuuendust. [11]

Implementeerimise käigus kirjutatakse testid, mida on kirjeldatud peatükkides 2.3.1, 2.3.2, 2.3.3, 2.3.4. Kasutatakse kõiki testimistüüpe, et testitav süsteem oleks võimalikult palju kaetud testidega. Kõik testid valitakse rangelt prioriteedi järjekorras. [11]

Teostamise käigus uuritakse, kas oodatav tulemus oli samasugune, kui tegelikkuses. Kui mitte, kirjutatakse testimise raport. Et tulevikus vältida erinevaid probleeme, käivitatakse sama test mitu korda uuesti, samuti uuritakse probleemi tekkepõhjust. Kui järgnevatel testimise kordadel leitakse samasugune või ligilähedane probleem, uuritakse mitte viimase, vaid kõikide testimise raportite järgi. Testide taaskäivitamise põhiülesandeks on kontrollida, kas samasuguste tingimuste korral, mille kaudu leiti antud probleem, kordub või mitte. Juhul kui süsteemis enam ei ilmne etteantud probleem, testitakse antud probleeme teiste funkstionaalsustega kas manuaalselt või kirjutatakse uued testid, et kindlustada süsteemi stabiilsust erinevate mõjutegurite kaudu. [11]

2.2.4 Hinnang ja raporteerimine

Hinnangu ja raporteerimisega kontrollitakse, kas testimise tulemused sobivad etteantud plaanide järgi, mida tehakse kõikidel testimise tasemetel, mille alusel määratakse, millal tuleb lõpetada testide kirjutamine. Testide kirjutamist pole vaja teostada kõikvõimalike funktsionaalsustega. Seda tehakse vaid kindlate kriteeriumitega prioriteetjärjekorras. Kui arvatakse, et tähtsamate osade piir on ületatud, lõpetatakse antud iteratsioonis tööd. Sellisel juhul hakatakse uurima testimisraporteid ning vaatama nende järgi, kus vajab süsteem rohkem toetust, kus ilmnes kõige rohkem probleeme, miks probleemid esinesid just seal, mis olid nende tekkepõhjused, kuidas saab neid vältida. Nende küsimuste abil pannakse paika, kas süsteem vajab uusi teste just juba kirjutatud testidele või mitte. [11]

Lõpus kirjutatakse lõppraport, mis läheb edasi huvirühmale ehk kliendile. Näidatakse süsteemi põhivead ning kirjeldatakse üldist testimist, mida tehti. [11]

2.2.5 Testide lõpetamine

Testimise lõpus kogutakse kogu informatsioon ühte, et analüüsida ning jagada omavahel kogemusi. Selgitatakse ja õpetatakse teistele tekkinud probleeme ning nende lahendusi. Antud projekti puhul oli selleks põhiliselt Xpathi kasutamine, leidmine kui ka dünaamiliste muutujate vältimine. Protokolli alusel testimise lõpetamine on aeg, millal testitav süsteem viiakse ellu ning kõik on valmis. Testimise lõpetamise kriteeriumid on testimise kontroll, kas kõik testid on lõpetatud, teostatud ning kirjutatud. Kontrollitakse, kas kõik protokollid on kirjutatud, kontrollitud, uuesti testitud või kinni pandud. [11]

2.3 Testimistüübid

Testimistüübid võimaldavad testida süsteemi erinevate vaatenurkade alt, mis katab tervet süsteemi uute testimisvõimalustega. Testimistüübid sõltuvad testimise eesmärkidest – programmi funktsioon; kasutajasõbralikkus ja kasutamine; struktuur. [11]

2.3.1 Funktsionaalne testimine

Funktsionaalne testimine on üks testimise tüüpidest, mis näitab, mida peab rakendus tegema ning mis on selle tagajärjed. Funktsionaalsus on tavaliselt kirjeldatud projekti spetsifikatsioonis, kus on analüüsitud ja arutatud kõiki võimalike tehingute ehk funktsionaalsete osade stsenaariume. Kuid kogu funktsionaalsust ei ole võimalik kirja panna (või jääb vahele). Selleks tuleb testijal kindlaks määrata, mis on rakenduse kõige tähtsamad osad ning kuidas on võimalik neid kasutada erinevatel juhtudel, nii-öelda määrata võimalikud riskid. [11]

On olemas kahte tüüpi funktsionaalset testimist: nõudmistel ja äriprotsessidel põhinev testimine. Nõudmistel põhinev testimine kasutab spetsifikatsiooni, et kirjeldada kõikvõimalikke tegevusi süsteemi sees. Spetsifikatsiooni kirjeldamisel lähtutakse süsteemist ning võimalikest riskidest. Äriprotsessil põhinev testimine kasutab äriprotsessi teadmisi igapäevases elutsüklis. [11]

2.3.2 Mittefunktsionaalne testimine

Mittefunktsionaalne testimine erineb funktsionaalsest sellega, et testimise kriteeriumid ei ole enam seotud sellega, kuidas peab süsteem kindla tegevusega töötama, vaid kuidas kõik on tehtud – kui kiiresti süsteem töötab, palju võtab aega süsteemi vastamine kasutajale. Testimise skoopi läheb nüüd kasutajasõbralikkus. Uuritakse, kas süsteem on kiire, stressivastane, piisavalt turvaline, disain on kasutajasõbralik. Mittefunktsionaalne testimine võib olla läbitav kõikidel testimise tasemetel, mis kasutavad ISO 25010 standardit. ISO 25010 omab kaheksat erinevat kriteeriumit [16]:

- Funktsionaalsus - seletatakse, kuidas süsteem toimib, mida peab tegema.
- Töökindlus – süsteem on võimeline kohandama rikkeid, suudab kaitsta kasutajat erinevate probleemide eest, mis süsteemiga toimuvad, näiteks vea tekkimise ajal kuvatakse kasutajale vastav kiri.

- Koostalitusvõime – süsteem on arusaadav ehk kasutaja saab aru, milleks on mõeldud kõikvõimalikud tehingud. Samuti peab kasutajal tekkima süsteemi kasutamisel meeldivus.

- Efektiivsus - kõik kõrvalised kasutatavad ressursid töötavad ning on integreeritud süsteemi õigesti.

- Turvalisus - süsteem on kasutamiseks piisavalt turvaline, sisestatav informatsioon ei ole kolmandale isikule kuidagi kättesaadav.

- Kokkusobivus - erinevad komponendid saavad koos töötada. Näiteks programmid või riistvara.

- Hooldatavus - kood on kirjutatud nii, et seda on lihtne vahetada ja testida.

- Porditavus - kasutajasõbralikkuse alamliik. Kogu süsteemi peab olema võimalik kasutada erinevatel platvormidel – nii erinevatel brauseritel, operatsioonisüsteemides kui ka mobiilsetel platvormidel.

2.3.3 Regressiooni-testimine

Viimane ja tähtsaim osa testimisest on regressioonitestimine ehk rakenduse töövõime ja tegevus peale koodimuudatusi. See on tähtsaim just sellepärast, et peale igat koodimuutmist võib esineda mitmeid kordi uusi probleeme ja vigu. Regressioonitestimine on juba varem testitud programmiosa korduvtestimine, mis võib toimuda kõikidel testimise tasemetel. Testimine toimub ainult pärast koodi muutust, kuid testimise meetodid peavad olema täpselt samad, et saavutada samad tingimused probleemi esialgsel ilmnemisel. Regressioonitestimise alla kuulub ka kinnitustestimine. Nende põhiline vahe on ainult üks – regressioonitestimine on testmine, mis toimub juba varem olnud probleemil, kuid kinnitustestimine on kordustestimine. Automatiseerimine on ideaalne regressioonitestimiseks kuna testimine toimub igal juhul konkreetsel süsteemiosal, kuid automatiseerimine säästab nii aega kui ka ressursse. Seda vahendatakse ainult, kui on vaja testida konkreetset probleemi. Tegelikkuses regressioonitestimine pole ainult ühe probleemi ümbertestimine. Lisaks sellele on vaja testida probleemi erinevate funktsionaalsustega. [11]

3 Platvormide ja kasutatavate süsteemide ülevaade

Valitud platvorm on erinevate meetoditega ülesehitatud automaattestide jaoks mõeldud platvorm. Selle tööks valiti üks automaattestimise platvorm ja sellega kaasnevad kasutatavad süsteemid. Need on:

- Python Robot Framework (automaattestimise platvorm)
- Xpath
- Dünaamilised muutujad

3.1 Python Robot Framework

Robot Framework – üldine automaattestide keskkond protsesside testimiseks ja arendamiseks, mis on võtmesõnadega juhitav platvorm. „Robot Frameworki viimane väljaantud versioon on 0.40.1, kus süsteem on üles ehitatud puustruktuurina“ [4]. Puu tüveks on testide komplekt, mis koosneb testjuhtumitest, võtmesõnadest, skalaaridest ja listidest. Testjuhtum koosneb võtmesõnadest, kus iga võtmesõna on defineeritud skalaaridega. Kõik elemendid on visuaalselt näidatud puu moodustatud skeemi vasakus ääres (vaata joonis 1). Programm aitab automatiseerida raskeid süsteemseid funktsionaalseid veebirakendusi. [3]

Joonis 1. Robot Frameworki testide ülesehitus

Robot Framework on tabelitel põhinev testimine. Skriptide kirjutamiseks kasutaja sisestab tabiletsse võtmesõnad koos nende tegevustega. Iga tegevus määrab funktsionaalselt iga võtmesõna tegevust seega vajalik kasutaja poolt kirjutatav kood on

minimaalne. Tabelite sees olevad võtmesad juhivad samm sammult funktsionaalset tegevust. [3]

On olemas kolm tüüpi võtmesõnade moodusi [3]:

1. Kõrgtaseme võtmesõnad – kasutatakse kindlate aspektide jaoks läbi erinevate teekide nagu Selenium.
2. Madaltaseme võtmesõnad – kasutatakse koodi minimaalset kirjutamise viisi ehk kasutatakse üht argumenti nii palju kui võimalik.
3. Tehnilised võtmesõnad – testide käivitamiseks mõeldud võtmesõnad

Testide kirjutamine ning käideldavus on kindlustatud järgmiste omadustega [3]:

1. Kõrgtaseme arhitektuur
2. Lihtsa tabulaatori süntaks
3. Andmepõhine testijuhtimine
4. Tekstandmete redigeerija
5. Aruandefailid
6. Logid
7. Testandmete teegid
8. Maven, Jenkins, Ant

Kasutaja ei ole piiratud ühe teegi kasutamisega. Python Robot Framework lubab kasutada erinevaid teeke nagu Selenium, AutoItLibrary jt koos kõigi nende funktsioonide ja võtmesõnadega kogu projekti vältel. [3]

Python Robot Framework jooksutab kirjutatud teste järgmisel viisil. Esialgu kogutakse kõik testimise juhtumid, loetakse ning seadistatakse muutujad. Järgnevalt jooksutatakse kõik sammud igas testimise juhtumis koos kõikide muutujatega. Peale testide lõppemist, antakse testide tulemused koos logifailidega xml ja html formaadis. [3]

Üheks põhiliseks Python Robot Frameworki kasutamise eeliseks teiste platvormidega võrreldes on see, et lisaks käsureale on võimalik kirjutada koodi tabelite kaudu.

Kogu koodi süsteemne hierarhia on sama põhimõttega kui tabelil põhineval testimisel, kuid käsureas kirjutatav kood on vähem veatõrjuv. "Kasutaja saab veast teada ainult siis kui kood on käivitatud" [3]. Tabelitel põhinev moodus annab veateate kohe peale tabelist väljaminemist. [3]

Igal platvormil on iga brauseri kohta enda omapärad. „Robot Framework toetab viit erinevat brauserit: Firefox, Internet Explorer, Safari, Google Chrome ja Opera" [2]. Brauseri kasutamiseks peab kasutaja importima kasutava brauseri ning installeerima selle projekti. Peale brauseri projekti installeerimist, peab kasutaja kutsuma välja funktsiooni, kus on kasutuses brauseri lühendid. Kasutatakse jägmisi lühendeid [2]:

1. googlechrome, gc – Google Chrome
2. firefox, ff – FireFox
3. internetexplorer, ie – Internet Explorer
4. safari – Safari
5. opera – Opera

3.2 Xpath

3.2.1 Xpathi ülevaade

Xpath on xm ja xhtml dokumentide elementide päringute keel, mille abil on võimalik määrata suvalise elemendi asukohta failis. Xpath on loodud rohkete erinevate sõlmede leidmise mooduseks, mis viitab konkreetsetele elementidele xml dokumendis nagu atribuut, tekst, kommentaar, dokumendisõlm. Otsimisse võivad minna ka nimed, aeg ja kuupäev.[8]

3.2.2 Xpathi väljendus

Xpath loob mustri, mille abil navigeeritakse kasutatavate sõlmedesse, kus mustrid koosnevad seitsmest sõlmest, mis saadakse väljundiks: tüvi, element, tekst, atribuut,

kommentaar, protsessi instruktsioon, unikaalne nimi. Järgmine list näitab põhistruktuurseid Xpathi kasutatavaid elemente [9]:

1. „/“ – valitakse elemendid, mis algavad tüvisõlmes
2. “//” – valitakse elemendid konkreetsest sõlmest, mis sobivad antud valikusse
3. “.” – valitakse käesolev sõlm
4. “..” – valitakse käesolevas sõlmes vanem
5. “@” – valitakse atribuut
6. “//div[@class="testKlass"]” – valitakse div-st klass, mille atribuudi “class” nimi on “testKlass”
7. “*” – atribuutide otsing toimub terves xml failis. Näide - //*[@class=”testKlass”]. Käesolevas näites otsitakse kogu failis selline „class“, mis omab väärtust „testKlass“
8. „contains“ – otsib kõik võimalikud väärtused, mis omavad kindlat teksti.
9. „normalize-space()“ – tagastab kõik argumendid ilma alg- ja lõpptühikuta.

Teiseks mooduseks on kasutada tervet Xpathi, millega viidatakse elemendile läbi kõikide sõlmede. Näide: html/body/div[9]/form/div[2]/span/input. Sellest näitest saab järeldada, et vajalik element asub inputi sees, kuid enne seda on läbitud mitu erinevat sõlme. [9]

3.3 Dünaamilised muutujad

Dünaamilised muutujad on genereeritud kui muutuja identifikaatorid, mida kasutatakse raskemate probleemide lahendamise käigus nagu stringiprobleemid [10]. Probleemi lahendamisel süsteem vaatab, kas varem on lahendatud sarnast tüüpi ülesannet. Kui selgub, et antud ülesandetüüpi on lahendatud varem, siis kasutatakse samu meetmeid ka selle probleemi puhul andes talle uue id [10]. Probleem võib seisneda selles, et kõik kasutatavad id-d on dünaamilised, mis raskendavad automatiseerimise tööd, kuna iga dünaamilist muutujat omava välja kasutamise korral genereerib koheselt endale uue id,

mis ei pruugi omada loogilist seaduspärasust. Selleks et lahendada antud probleemi, tuleb uurida [10]:

- Mis seaduspärasust omab dünaamiliste muutujate genereerimine.
- Kas on olemas alternatiivne muutujate variant, mis ei kasuta dünaamilisi muutujaid.

Automaattestide kirjutamisel tuleb alati vältida kõiki dünaamilisi muutujaid, kuna need pole piisavalt stabiilsed. Dünaamiliste muutujate kasutamine võib lõhkuda testide edasijooksutamist, kui platvorm ei leia kasutaja poolt defineeritud muutujat. Veebileht, mis kasutab dünaamilisi muutujaid, kasutab peaaegu alati ka tavalise id-ga välju.

Kui tekib olukord, et kõik htmlis olevad muutujad omavad dünaamilisi muutujaid, on võimalik kasutada teiste failide muutujaid – cssi muutujaid, mis navigeerivad samasse kohta, kui html failis olevad muutujad. Juhul, kui ka cssi failis olevad muutujad on dünaamilised, tuleb leida alternatiivne viis kirjutada automatiseeritud testid. Üks valikutest oleks minna üle „Sikuli“ visuaalse testimise platvormile. Sikuli võimaldab testida tehtud piltide pikslite kaudu. Näide muutuvast id-st on järgmine:

Peale igat veebilehe uuendamist, muutub id teiseks. Antud näite puhul tuleks ignoreerida dünaamilist id-d ning otsida elemendi asukohta läbi „span“ ja võtmesõna „Windows“-i abil. Selle koodi navigeerimiseks ehk Xpath oleks järgmine: //span[contains(.,’Windows’)]. *Span* näitab, et otsitav muutuja asub sees. Contains-i abil saame öelda, et sees olev muutuja peab sisaldama sõna Windows.

4 Testitav rakendus

Automaattestide uurimiseks on võetud kasutusele Top Connecti integreeritav veebiprojekt Sugar CRM. Sugar CRM on kliendisuhete tarkvara juhtimise infosüsteem, mis lihtsustab turunduslikke ülesandeid, aitab suurendada käivet ning luua klienditoetusega ühtset infosüsteemi. Sugar CRM on avatud lähtekoodiga rakendus, mida saavad kasutada kõik kasutajad. Infosüsteem koosneb neljast tüübist: müügiosakonnast, kasutajaosakonnast ja kaubandusosakonna töötajatest, klientidest. Klient omakorda jaguneb jagajaks, tarnijaks, partneriks ja edasimüüjaks. Põhilised Sugar CRM funktsioonid on [6]:

1. Kasutajate juhtimine – juhitakse ja kontrollitakse kogu kasutaja ajaloo infot koos kliendi kasutatud infoga, maksetega, tegevustega ja tellimustega.

2. Müügi juhtimine – kontrollitakse kõik tehtavad müügid, mis eelnevalt analüüsitakse infosüsteemiga. Vastavalt tulemustele sorteeritakse müügi poolt tehtud analüüs ning määratakse vastavale isikule või osakonnale.

3. Aja juhtimine – loodud kalendrid annavad võimaluse täpselt koordineerida ning planeerida enda aega vastavalt nõuetele. Kalender on loodud nii individuaalseks kasutamiseks kui ka tervele grupile. Iga kasutaja, kes omab vastavaid õigusi, saab sätestada enda jaoks vaateid, kus on filtreeritud ja näidatud osakonnatöö ajalised plaanid.

4. Klienditugi ja -teenus – klient on võimeline registreerima ennast kui ka enda kaastöötajaid Sugar CRM infosüsteemi omades selleks administratiivsed õigusi. Klient saab luua juhtumi või tellimuse, mis järgnevalt saadetakse peale algoritmilist analüüsi edasi kaubandusosakonda, müügiosakonda või kasutajaosakonda.

5. Turundus – turunduskampaania ja potentsiaalsete ärisuhete juhtimine kasutuses on kogu informatsioon toodetest ja teenustest, mida ettevõte pakub.

6. Funktsionaalsus kõrgeimale tööosakonnale – kasutajaliides ja ärimudel erinevad tavakasutajatest. Kõik paneelid omavad informatsiooni, mis kontrollivad antud tööosakonna tööskeemi. Käideldav informatsioon on eraldi sorteeritud

infosüsteemi, mida saab eraldi filtreerida. Iga kõrgeim tööosakonna töötaja omab erinevaid vaateid nii kliendi kui ka oma alluvate töötajatest.

7. Mobiilne tugi – kasutusele on võetud suuremalt jaolt ainult kliendipoolne mobiilne tugi. Töötajad omavad ainult emaili sünkroniseerimist ehk töötaja saab saata kirja osakonna meilile, mis hiljem sünkroniseeritakse süsteemi poolt ning info saadetakse osakonnajuhatajale või kõrgemale tööosakonnale. Kasutaja omab seejuures täielikku mobiilset kasutajaliidest. „Kõik funktsionaalsed tehingud on tehtavad kõikidel toetavatel mobiilsetel platvormidel“ [6]. Toetatavad platvormid on Windows, Linux, Mac ning toetavad brauserid on Google Chrome (versioonid 54-56), Firefox (versioonid 48-52), Internet Explorer (versioon 11) ja Safari (versioon 10) [5].

4.1 Veebirakenduse üldine loogika

Järgmine list defineerib CRM süsteemi tehnilisi äriloogikaobjekte:

- Kasutajad – CRM süsteemi objekt, mis koosneb jagajast, tarnijast, partnerist ja edasimüüjast. Kasutaja on edaspidi nimetatud kui juriidiline isik, kes on Top Connectiga sõlminud lepingu töösuhete loomiseks.

- Tehniline kasutaja – arvete osakonna kasutajad, kes on määratud igale juriidilisele isikule. Igal juriidilisel isikul võib olla mitu tehnilist kasutajat, kuid ka ühel tehnilisel kasutajal võib olla mitu juriidilist isikut (juhul kui tegemist on tütarfirmaga).

- Kontakt – CRM süsteemi objekt, mis sisaldab juriidiliste isikute kasutajate kontaktinformatsiooni. Kõik kontaktid võivad omada pääset CRM portaalile.

- Tellimus – CRM süsteemi objekt, mis sisaldab endas erinevate ostutellimuste informatsiooni, mis on tehtud kontaktina.

- Taotlused – CRM süsteemi objektid, kus klient saadab päringuid kaubandusosakonda, kus edasi tegeleb määratud töötaja kirjutatud taotlusega. Taotluse tüüp võib olla erinev, näiteks kliendi portaalikogemus, tekkinud probleemid maksmise käigus jne.

- Kontraktid – CRM süsteemi objektid lepingute hoidmiseks.
- Dokument – CRM süsteemi objekt dokumentide hoidmiseks.

4.1.1 Juriidilise isiku registreerimine

Juriidilise isiku registreerimine toimub kasutajamooduli kaudu, mida saavad teha ainult Top Connecti osakonna töötajad. Edukaks registreerimiseks peab osakonna töötaja sisestama kõik nõutud alad koos juriidilise isiku kontaktidega. Nõutud alad on järgmised: juriidilise isiku nimi, juriidilise isiku numbriline viide, kliendi tüüp, staatus, tase, juriidilise isiku müügijuht, juriidilise isiku primaarne aadress, kasutatav valuuta, komisjoniprotsent, kontaktide nimed, kontaktide primaarne aadress, leping ning lepinguinformatsioon koos tähtajaga. Peale salvestamist süsteem jooksutab duplikaadi kontrolli. Kui süsteem leiab sarnase nimetüvi, teatab ta sellest kasutajale. Edukaks registreerimiseks peab kasutaja kas nõustuma, et süsteemis võib olla sarnane nimi või muutma juriidilise isiku nime. Kui juhtub, et süsteemis on duplikaat, on võimalik liita erinevaid juriidilisi isikuid kokku. Liitmise käigus saab valida informatsiooni, mis jääb lõppversiooni kaarti.

Juriidilise isiku kaardil on kättesaadav järgmine informatsioon: põhiline informatsioon juriidilise isiku registreerimise käigus ning juriidilise isiku moodulid, kus tabeli meetodi abil on näidatud järgnev informatsioon: kutsed, kohtumised, ülesanded, märkmik, juriidilise isiku grupid, meilid, taotlused, bugid, dokumendid, ettevõttelogi, lepingud, muudatuste ajalugu, tehnilised kasutajad, tehniliste kasutajate grupid, tellimused ja uudised. Kogu moodulites sisalduv informatsioon on muudetav.

4.1.2 Juriidiliste isikute ühendamine

Juriidilisi isikuid võib ühendada ühte süsteemi. Sellisel juhul kõik juriidilised isikud kui ka kontaktid näevad üksteise informatsiooni. Sellise operatsiooni tegevuseks on vaja täita järgmisi tingimusi:

- Ühendamiseks nõutud juriidilised isikud peavad kuuluma ühte gruppi.
- Üks juriidilistest isikutest peab olema märgistatud kui ühendatav kasutaja.
- Kõik juriidilised isikud peavad omama viidet - ühendatav kasutaja.

Kui kõik tingimused on täidetud, kõik kasutajad, kelle juriidiline isik on ühendatud „Ühendatavale kasutajale“, näevad kõik avalikku informatsiooni. Selline moodus simuleerib tütarfirmade tegevust.

4.1.3 Kontakti registreerimine

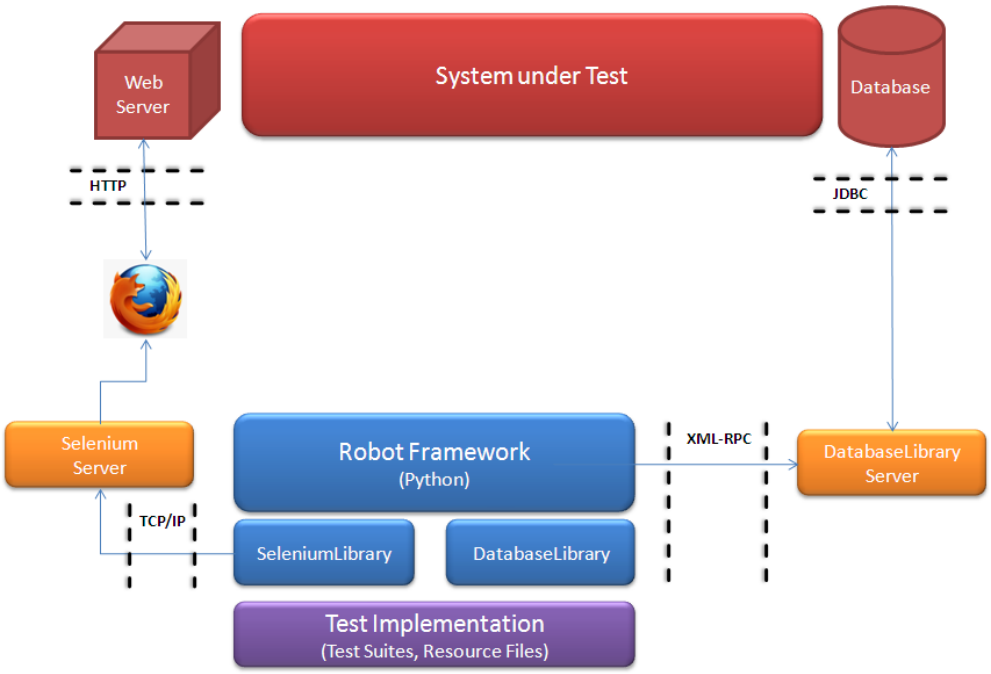
Ligipääsu saamiseks Sugar CRM portaalile, registreerivad osakonna töötajad kliendi kontaktisikuna. Registreerimist saab teostada kas paneelil või juriidilise isiku moodulis, mille nõutud informatsiooniks on eesnimi ja perenimi. Edasi peab kontaktile määrama juurdepääsu portaalile andes talle õigusi portaalile sisenemiseks, määratakse kasutajatunnus, email ning seejärel saadetakse parool vastava emailile. Juhul kui kontakt ei saa siseneda portaali, saab osakonnatöötaja genereerida uue parooli, mis saadetakse kontakti emailile. Kõik osakonna töötajad näevad kontakti informatsiooni kas kontaktiliideses või juriidilise isiku moodulis. Kontakti avalik informatsioon on nähtav kontaktikaardil või eelvaatluse mooduli abil, kus on näha kontakti tegevusvaldkonna vastutusi.

4.2 Arhitektuur

Automatiseerimise testide töökindluse suurendamiseks uuriti, kuidas erinevad testimisemeetodid ja tööprotsessid mõjutavad kirjutatud testide kvaliteeti. Kasutati erinevaid testimistüüpe – funktsionaalset, mittefunktsionaalset ning ka regressioonitestimist. Töö alustamisel kirjutati testid funktsionaalse ja mittefunktsionaalse testimistüübi alusel. Loodi spetsifikatsioon, mille alusel kirjutati testid, mõeldi läbi kogu testimise osa – millal toimub testimine, kuidas teste minimeerida nii, et ei tekiks üleliigset testimist ja ajaraiskamist, mõeldi läbi testide struktuur. Spetsifikatsiooni alla läks funktsionaalsus, mis pidi kindlasti olema testitud automaattestimise käigus.

Testide kirjutamiseks kasutati Robot Framework programmi. Platvormile lisandus Selenium, mis võimaldas kasutada erinevaid testimistüüpe brauserite testimiseks. Selenium ühendas Robot Framework platvormi testimisvõimalustega, millega olidki kirjutatud kõik testid. Joonisel 2 on näidatud, kuidas töötab Robot Framework. Näidatakse, kuidas suhtlevad omavahel erinevad Robot Frameoworki tasemed ning võimaldavad käivitada teste. Testid kirjutati Pythoni keeles, mida kirjutati Robot

Frameworki käsureas. Testide kirjutamisviis oli erinev – HTML, tekst kui ka BDD. Käsureas olevate testide kirjutamiseks valiti text formaat.



Joonis 2. Robot Frameworki testimise arhitektuur

Testide kirjutamisel oli kõige olulisem leida elementide Xpath. Xpathi leidmiseks võeti kasutusse Google Chome Inspect Elements, Google Chrome *extension* InspectorGadget, Mozilla Firefox Inspect Element, Mozilla Firefox *extension*'id Firefug ja FirePath. Enim kasutati Firebug ja Firepath, mis on Mozilla Firefoxi *extension*'id, mille abil on võimalik genereerida kaht tüüpi Xpathi – täielik ehk absoluutne Xpath või minimaalne Xpath. Minimaalne Xpath sisaldas elemendi üht muutujat või kui eksisteerib mitu samasugust, siis lisatakse juurde tema esivanemate muutujaid seni kuni antud Xpath ei muutu unikaalseks.

Joonis 3 on testitava veebi htmli koodijupp, mis näitab, kus asub element. Selle koodi järgi võib järeldada, et antud näite puhul oli leitud täielik Xpath, mis näidatakse üleval: //[@id='content]/div/div/div/div/ul/li[1]/ul/li[1]/div/ul[2]/li/div/div/div[2]/div/table/tbody/tr[1]/td[1]/span/span. Selliseid Xpathe tuleb vältida kuna koodi mahus suureneb, mis pikendab Robot Frameworki testide jooksutamist.

Joonis 3. Täieliku Xpathi näide

Kuna Firepath ja Firebug ei arvesta dünaamiliste muutujatega, võttes aluseks esimese muutaja, võib selline moodus mitte sobida ning tuleb leida enda Xpathi. Firebug ja Firepath võimaldavad kontrollida enda leitud Xpath ning vaadata, kus asub antud element. Kui Xpath osutus õigeks, näidati kaustajale kõik võimalikud variandid, mis sobivad. Joonis 4 puhul võib pakkuda järgmist koodi: //span[contain(.,'Info')], mis leiab spani sees olevat koodi, mis sisaldab sõna „Info". Kahjuks see ei garanteeri, et antud Xpath viitab ainult ühele elemendile. Selliseid võib olla mitmeid, millepärast võib kas muuta Xpathi kuni see ei muutu unikaalseks või lisada juurde veel üks teekond, mis viitab mitte elemendile vaid eelmisele Xpathile. Joonise 4 järgi võime näha, et vanemaks on class="full-width list". Kui kasutada seda viitamiseks, saame proovida uuesti kontrollida, kas antud Xpath sobib. Xpath //span[@c class="full-width list"] //span[contain(.,'Info')] küll suurendab koodi, kuid kui ei leita minimaalset Xpathi, tuleb kasutada alternatiivseid meetmeid. Üks meetmest, mida veel tihedalt kasutati oli Google Chromi extension InspectorGadget, mis kasutas concati. Selline viis suurendas samuti töömahtu ning läks kasutusse vaid erandjuhtudel, kui teised leitud Xpathid olid pikemad. Üks suur eelis, mida võimaldas InspectorGadget, on see, et valides vajaliku elemendi, kuvati kasutajale kogu minimaalne Xpath koos kõikide samasuguste Xpathide elementidega kollase värviga.

Testitava veebilehe arhitektuur ehitati sel moel, et suurem osa muutujatest kasutasid dünaamilisi muutujaid. Kuna Firepath võimaldas leida kas absoluutset Xpathi, mis ei sobi oma pikkuse poolest ja minimaalse Xpathiga, mis tihti ei sobinud, tehti otsus kasutada Xpathi leidmisi *extensione* ainult erandjuhtudel. Aspektid, mis vähendavad testide töökindlust on järgmised:

- absoluutne Xpath ehk täielik Xpath,
- Xpath, mis on genereeritud teistest Xpathidest,
- dünaamilised muutujad.

5 Testide töökindluse suurendamise meetmed

Antud peatükis kirjeldatakse töös esinenud eksperimendid ning nende tulemused.

5.1 Eksperimendi kirjeldus

Automatiseeritud testide kirjutamiseks võeti kasutusse Robot Framework automatiseerimise platvorm, kuhu lisandus Selenium, et võimaldada kasutamiseks testimise meetmeid. Testide kirjutamisel lähtuti erinevatest testimistüüpidest, mille alusel loodi kahte tüüpi testimist:

- Testid kasutades dünaamilisi muutujaid
- Testid kasutades staatilisi muutujaid

Testide läbiviimiseks kasutati Windows 10 Pro Lenova ThinkPad Yoga 12 Itel(R) Core(TM) i5-5200U CPU @ 2.20GHz arvutit. Testid käivitati erinevatel brauseritel, mida kirjeldati spetsifikatsioonis kui kõige populaarsemaid Windowsi brausereid:

- Google Chrome – ChromeDriver 2.29
- Mozilla Firefox – GeckoDriver 0.16.1
- Internet Explorer – InternetExplorerDriver 3.4

Kõik testid jooksutati kordamööda. Sellisel juhul ükski test ei sõltu teisest. Et teha testid võrdväärtseteks teiste brauseritega, olid kõik testid kõikidel brauseritel samasugused. Selleks, et testid ei mõjutaks teisi teste ega brausereid, kustutati iga tegevus peale kindlat funktsionaalsuse testimist. Testide lõppemisel analüüsiti logide järgi iga testi aega ning võrreldi teiste brauseritega (joonis 4).

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	37	33	4	00:11:23	
All Tests	37	33	4	00:11:23	
Critical Tests	37	33	4	00:11:23	
All Tests	37	33	4	00:11:23	

Joonis 4. Testide statistika Google Chromi brauseril

Joonise 5 järgi on näha, et testid kukuvad läbi *Click on Certain Contact* juures. Logi järgi tehti järeldus, et Robot Framework ei suuda leida veebis ettekirjutatud elemendi Xpathi. Probleeme võib olla mitu – valesti kirjutatud Xpath, midagi juhtus masinaga, kus jooksid testid, süsteem ei jõua testidele järgi, midagi juhtus eelmises testis, midagi juhtus peale eelmist testi. Alguses pandi igaks juhuks juurde kaheks sekundiks Sleep, mis ei toiminud antud probleemis. Probleem seisnes selles, et peale otsingut tuli ette teine leht, mis ei sobinud antud otsinguga. Selle põhjal tehti probleemi kirjeldav test case arendajatele. Järgmises iteratsioonis probleem lahendati ära. Seni, kõik testid, mis jooksutasid otsinguid, peatati ning blokeeriti, et mitte ohustada teisi teste.

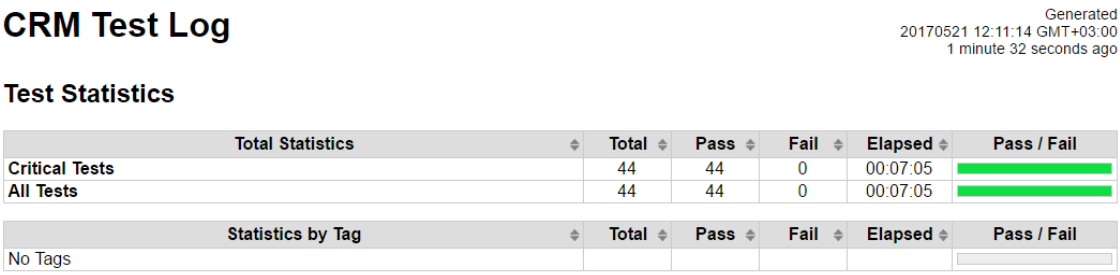
Joonis 5. Testide läbikukkuvuse logi

6 Eksperimendi tulemused

Eksperimendi käigus tehti 44 automatiseerimise testimist. Samad testid jooksid erinevatel brauseritel. Tulemused pandi kirja ning analüüsiti. Kui testi tulemus oli negatiivne, uuriti seda eraldi.

6.1 Google Chromi tulemused

Google Chromi draiveri kasutamisel ei ilmnenud probleeme. Testimise käigus tehti 44 funktsionaalset testi, mis jooksid 7 minutit ja 5 sekundit (joonis 6). Kuna teistel brauseritel ilmnesid tõrked, mida selgitatakse punktides 6.2 ja 6.3, tehti otsus jätkata funktsionaalsete testide kirjutamist ainult Google Chrome'i brauseril.



CRM Test Log

Generated
20170521 12:11:14 GMT+03:00
1 minute 32 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	44	44	0	00:07:05	
All Tests	44	44	0	00:07:05	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Joonis 6. Google Chromi testide statistika

Testimise käigus kasutati erinevaid testimistüüpe – funktsioonalset, mittefunktsionaalset ja regressioonitestimist. Alguses kirjutati testid vaid spetsifikatsiooni sees kirjeldatud kui enim vajalikud funktsionaalsed osad. Nende põhjal loodi protokoll, mis ehitati prioriteetjärjekorras, mille alusel oli kirjutati kõik testid.

Funktsionaalse testimise alla läks juriidilise isiku registreerimine, juriidilise isiku ühendamine, kasutaja registreerimine ning isikute tehingud. Testid olid sellised, et eelmiste testimiste käivitamine ei mõjutaks järgmisi. Kõik tegevused, mida sooritati, kustutati koheselt peale kasutamist. Sellisel moel oli järgneva testi käivitamine peaaegu sõltumatu. Ainuke, mis võis rikkuda uut testimist, oligi eelmiste andmete kustutamine. Testide ülesehitus on ehitatud sel moel, et eelmised testid oleksid minimaalselt kasutatavad järgmistes testides. Selline viis vähendab testide läbikukkumist.

Mittefunktsionaalse testimise käigus analüüsiti süsteemi kiirust ja kasutaja teavitamist erinevatest probleemidest – kas kasutajale kuvatakse probleemiteke. Süsteemi kiirust

analüüsiti peale testide lõppemist. Kui testide protokollides ajad erinesid, vaadati nende tekkepõhjuseid.

Süsteemi testimise käigus oli kasutati regressioonitestimist. Regressiooniteste kirjutati ainult juhul kui kõik kriteeriumid olid täidetud – põhitestid kirjutatud, probleeme tekitavate funktsionaalsuse testid olid kirjutatud. Valiti suvaline funktsionaalsus, mida pole spetsifikatsioonis kirjeldatud ning selle põhjal kirjutati erinevaid teste, kuidas süsteem käitub ja kas käitub nii nagu peab.

Kui testid kukkusid läbi, analüüsiti tekkepõhjusi ning raporteeriti neid. Bugi kõrvaldamiseni ignoreeriti sellised testid, mis võisid neid põhjustada. Samuti kirjutati uued testid, mis võiksid samuti põhjustada etteantud problemi. Kui õnnestus korrata, probleemist raporteeriti. Peale probleemi lahendamist, käivitati kõik sellised testid, mis põhjustasid antud probleemi. Kui probleemi enam ei eksisteerinud, käivitati kõik testid korraga. Kui probleemile järgmiseks iteratsiooniks lahendust ei leitud, kävitati igaks juhuks testid uuesti, et määrata, kas teised muudatused on parandanud antud süsteemi vigu või mitte.

Joonisel 7 on näidatud, kuidas registeerida kontaktinime sektoris täites ainult nõutud väljad. Testis on loodud kolm skalaari, kus on kirjas elementide Xpathid ja nende nimetus ning üks võtmesõna, kus kirjeldatakse testide käitumust. Antud näites läbitakse kaht viisi testimist – esimene osa kontrollib, et element (skalaar) on nähtav brauseris – *Wait Until Element Is Visibile* ning teine kirjutab informatsiooni kindlasse kasti – *Input Text $scalar $scalar*. *Input Text* nõuab kaht skaalari, kus esimene on elemendivälja Xpath ja teine tekst, mida sisestakse. Kasutatud on minimaalset Xpathi, mis leiab sellist sõlme, kus on kasutuses selline *aria-labelid*, mis omavad väärtusi *First Name* ja *Last Name*. Enne igat tegevust kirjutatakse logisse tegevuse kohta informatsioon. Selline viis aitab testide tulemuste lugejale anda kindla visiooni, mida antud testis tehakse.

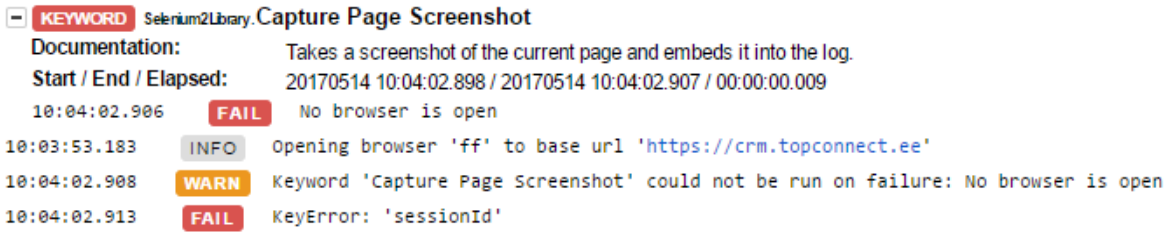
```
*** Variables ***
${FIRST NAME LOCATOR}    //*[@aria-label="First Name"]
${LAST NAME LOCATOR}    //*[@aria-label="Last Name"]
${TYPE INFO}      Automation2

*** Keywords ***
Enter All Required Fields
Log To Console    Entering First Name
Wait Until Element Is Visible    ${FIRST NAME LOCATOR}
Input Text    ${FIRST NAME LOCATOR}    ${TYPE INFO}
Log To Console    Entering Last Name
Wait Until Element Is Visible    ${LAST NAME LOCATOR}
Input Text    ${LAST NAME LOCATOR}    ${TYPE INFO}
```

Joonis 7. Robot Frameworki kontakti registreerimine

6.2 Mozilla Firefoxi tulemused

Mozilla Firefoxi testimine nurjus kuna tekkisid probleemid Mozilla Firefox *driver* geckodriveriga. Antud draiver ei sobinud Seleniumiga andes kaht viisi viga:

- 0.16.1 geckodriveri versiooniga saadi: „*No browser is open*“ kuigi brauser ise läks lahti (joonis 8). Probleemi ilmumine on eriline ning lahendust antud versioonile ei leitud. Võimalik probleem: Selenium ei aktsipteeri uuemaid Mozilla Firefoxi draivereid.

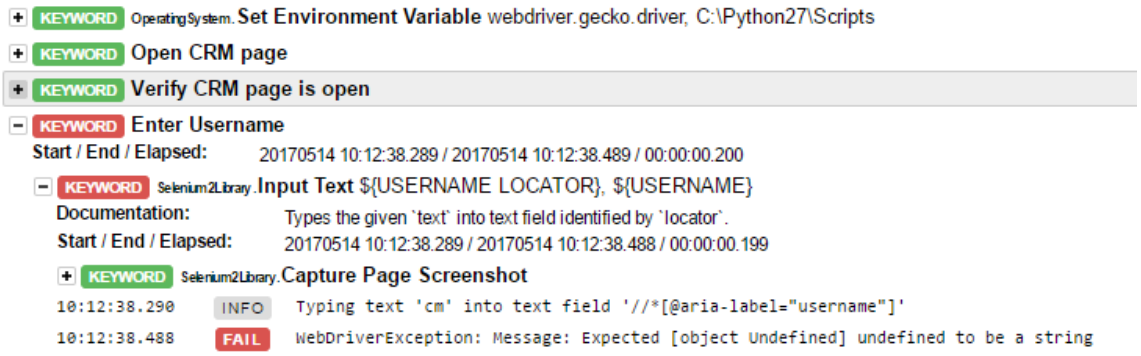


Joonis 8. 0.16 geckodriveri tõrke koht

Üheks probleemi lahendiks oli anda õige draiveri suund mitte läbi *Windows variable* Pathi abil vaid kasutada Robot Frameworki sisemist järgmiste muutujatega: *Set Environment Variable* webdriver.gecko.driver C:\Python27\Scripts. Kahjuks antud lahendus samuti ei sobinud.

- 0.11 geckodriveri versiooniga saadi järgmine viga: "WebDriverException: „*Message: Expected [object Undefined] undefined to be a string*“ (joonis 9).

Antud probleem ilmneb geckodriveriga. Draiverit saab kasutada ning see töötab, kuid tektstivälju pole võimalik kasutada. Lahenduseks pakuti uus versioon, mis aktsepteerib teksti ning muudab seda *Marionette* formaati. Lahendust pakutakse välja 0.16 geckodriveris [14]. Antud draiver ei sobi testimiseks, kuna pole töökorras ning testida antud versiooniga pole võimalik.



Joonis 9. 0.11 geckodriveri tõrke koht

- Testimise alla lähevad teised võimalikud versioonid 0.9.0, 0.10.0, 0.12.0. 0.13.0, 0.14.0, 0.15.0, 0.16.0. Kahjuks kõik brauseritel on eelnevad probleemid ning ei võimalda objektiivset testimist.

Kuna ükski geckodriveri versioon ei sobinud testitava süsteemiga, tehti otsus mitte testida antud brauseris. Mõjuvaks põhjuseks probleemi mittelahendamiseks oli ka see, et kõik Mozilla Firefoxis esinevad probleemid oli samad, mis Google Chromis, mis ilmnes manuaalse testimise käigus. Sellega võeti risk ning väljastati Mozilla Firefox automatiseerimise testide kasutamiseks.

6.3 Internet Exploreri tulemused

Internet Exploreri testid jooksid kokku 17 minutit ja 53 sekundit. 30 testist läks läbi vaid 7 (joonis 10). Analüüsimises võis näha, et kõik testid kukuvad „*Wait Unitl Element is Visible*“ juures. Internet Exploreri driver oli niivõrd aeglane, et elemendi ootamine kestis niivõrd kaua, et Robot Framework arvas, et elemendid puuduvad. Uurides aega, saab näha, et igas sisestusvormis läks ligikaudu 30-50 sekundit, millest saab järeldada, et internet exploreri informatsiooni sisestusvõime on kümneid kordi aeglasem Google Chromist. Intrnet Exploreri iga sümboli sisestamine võttis ligikaudu 3-7 sekundit, mis teeb sõnade puhul ligikaudu 24-56 sekundit.

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	30	7	23	00:17:53	
All Tests	30	7	23	00:17:53	

Joonis 10. Internet Exploreri testimise tulemused

Mitte kõik testid kukkusid läbi aja pärast. Testimise käigus Internet Exploreri draiver jooksis mitu korda kokku, mille pärast kogu testitav funkstionaalsus lõpetati (joonis 11). Sellega võib järeldada, et seitse funkstionaalset testi ei käivitatud. Probleemi lahendamiseks sai kasutada *Sleep* meetodi, mille kasutamine paneb kogu süsteemi ootama mingisuguseks perioodiks, mis pole ratsionaalne kuna testide jooksmine pikeneb mitmeid kordi. Igasugune süsteemi ootamine pole soovitatav [15]. *Sleep* meetodi kasutamisega pikenesid testid kuni 35 minutid ja 57 sekundit. Sellega saavutati 20 positiivset testi, viiel korral draiver läks katki peatades läbitavaid teste ning ülejäänud viiel korral Robot Framework ei suutnud leida elemendi.

Test Execution Errors

```
20170514 14:22:44.526  WARN  Keyword 'Capture Page Screenshot' could not be run on failure: URLError: <urlopen error [Errno 10061] No connection could be made
20170514 14:30:03.581  WARN  Keyword 'Capture Page Screenshot' could not be run on failure: NoSuchWindowException: Message: Unable to get browser
20170514 14:30:03.664  WARN  Keyword 'Capture Page Screenshot' could not be run on failure: NoSuchWindowException: Message: Unable to get browser
20170514 14:30:30.283  WARN  Keyword 'Capture Page Screenshot' could not be run on failure: NoSuchWindowException: Message: Unable to get browser
20170514 14:30:30.352  WARN  Keyword 'Capture Page Screenshot' could not be run on failure: NoSuchWindowException: Message: Unable to get browser
20170514 14:30:30.415  WARN  Keyword 'Capture Page Screenshot' could not be run on failure: NoSuchWindowException: Message: Unable to get browser
20170514 14:30:33.488  WARN  Keyword 'Capture Page Screenshot' could not be run on failure: No browser is open
```

Joonis 11. Internet Exploreri draiveri viga

Kuna Internet Exploreri draiver pole stabiilne, oli tehtud otsus mitte kasutada antud brauserit ning keskenduda ainult Google Chromi brauserile.

6.4 Dünaamiliste muutujatega testid

Süsteemi automatiseeritud testide kirjutamine dünaamiliste muutujate abil (joonis 12) oli keerukas, kuna dünaamiliste muutujate süsteemi on raske jälgida, mis pikendas testide kirjutamist mitu korda. Probleem seisnes selles, et muutujad omasid dünaamilisi väärtusi erinevatel põhjustel: elementide vajutamisel ja erinevate tegevuste käigus nagu teksti kirjutamine ja kustutamine. Leiti ideaalne moodus, kus kõik testid töötasid, kuid peale uut iteratsiooni kukkusid kõik testid läbi. Analüüsimise käigus järeldus, et uue elemendi lisamine või ümbervahetamine nihutab kõiki muutujaid nii, et ükski automatiseeritud test ei lähe läbi. Samuti tekkisid olukorrad, kus elemendi dünaamiline muutuja suurendati eelmise tegevusega, mis suurendas dünaamilist muutujat x arvu võrra. Selliste vigade

puhul oli peaaegu võimatu leida sellist funktsionaalsust, mis võimaldas elemendi vajutamist.

${BILLING CITY LOCATOR}	//*[@name="billing_address_city"]
${BILLING COUTNRY LOCATOR}	//*[@id="select2-chosen-24"]
${AGREEMENT TYPE}	//*[@id="select2-chosen-6"]
${STATUS LOCATOR}	//*[@id="select2-chosen-8"]
${SELECT LEVEL LOCATOR}	//*[@id="select2-chosen-12"]
${SALES MANAGER LOCATOR}	//*[@id="select2-chosen-46"]

Joonis 12. Näide Dünaamilised muutujad

Kuna dünaamiliste muutujatega testimine osutus pikaks ja mitte ratsionaalseks, oli tehtud otsus neid mitte kasutada ning kirjutada teste leides muutujad, mis viitavad samale elemendile.

7 Kokkuvõte

Töö eesmärgiks oli kirjutada automatiseeritud testid Top Connecti firma veebilehele. Sellega uuriti, kuidas kirjutada maksimaalselt efektiivseid automatiseeritud teste kolmes erinevas brauseris – Google Chrome, Mozilla Firefox ja Internet Explorer vältides kasutamast dünaamilisi muutujaid Xpathi elementidena.

Automatiseeritud testide kirjutamise platvormi valik oli Robot Framework, mis nõudis testide käivitamiseks Seleniumi kasutamist. Töö käigus kasutati kõiki testimise tüüpe ja punktides 2.2 ja 2.3 kirjeldatud etalonprotsesse. Nende abil saavutati testide kirjutamise efektiivseim viis, mis käivitati peale igat süsteemiuuendust.

Töö tegemisel leiti ja lahendati järgmised probleemid:

- Dünaamiliste muutujate kasutamine – suuremal osal kordadest veebileht genereeris selliseid muutujaid, mida polnud võimalik kasutada. Tehti otsus vältida kõiki niisuguseid muutujaid ning leida alternatiivne variant – leida täielik või minimaalne Xpath, mis ei kasuta ühtegi dünaamilist muutujat.

- Testitaval veebilehel oli raske leida unikaalseid muutujaid, mille pärast pidi suurendama Xpathi pikkust. Suurem osa muutujatest olid teistega sarnased, mis raskendas Xpathi leidmist. Probleemi lahendamiseks otsiti manuaalselt Xpathid. Selliste Xpathide otsimist aitasid erinevad extensionid – Firepath ja InspectorGadget. Järelduseks saadi teada, et igal elemendil on vähemalt üks selline muutuja, mis ei kasuta dünaamilisi muutujaid.

- Firepath pole enam toetatud *extension* – tuli leida alternatiivne variant nagu SelectorGadget või muuta Firepathi versiooni. Versioonimuutmisel võeti risk, et võib tekkida erinevaid probleeme. Järeldati, et ühtegi probleemi versiooni alandamisel ei tekkinud.

- Robot Frameworkis tekkinud probleemid:

    - Kaustad sorteeriti ainult tähestikulises järjekorras. Sellisel juhul testide järjekord võis muutuda. Probleemi vältimiseks loodi süsteem, kus

kaustanime ette läksid erinevad tähed, mis võimaldasid sorteerida kaustasid nii, nagu vaja.

    - Kaustanime ei saa platvormi sees muuta. Kausta muutmiseks pidi minema failide asukohta ning muutma manuaalselt. Selle probleemi vältimiseks ei leitud lahendust ning kui tekkis vajadus muuta kaustanime, leiti faili asukohad ning muudeti neid manuaalselt vastavalt vajadusele.

    - Tekkisid erinevad logide vead, mispärast oli vaja uuesti avada Robot Framework ning kävitada testid. Probleemi esinemist ei suudetud leida.

- Internet Exploreri ja Mozilla Firefoxi draiverid ei tööta korrektselt Selenium 3.0 peal. Mozilla Firefoxi testimiseks on vaja alandada Seleniumi versioon 2.0 peal. Internet Exploreri testimiseks lahendust ei leitud. Testide jooksmine võttis liiga palju aega, mispärast suurem osa teste ei töötanud korrektselt. Lahenduseks oleks muuta testide kiirus selliseks, et kõik vajalikud testid oleksid kaetud.

Töö eesmärgiks oli saada ligikaudu 100 funktsionaalset testi. Tulemuseks saadi vaid 44 funktsionaalset testi, kasutades minimaalseid või absoluutseid Xpathe, kus tehti ligikaudu 500 tegevust. Parim automatiseeritud testide kirjutamise viis on:

- Valmistada spetsifikatsioon, mille järgi hakatakse kirjutama automatiseerimise teste. Spetsifikatsioonis peavad olema kirjeldatud ainult kõige tähtsamad funktsionaalsused, mis peavad olema kindlasti kaetud.

- Ühe testi funktsionaalsuse pikkus peab olema nii lühike kui võimalik. Sellisel juhul läbitakse rohkem teste ning probleemi leidmine on lihtsam.

- Dünaamiliste muutujate vältimine, kuna nende kasutamine võib põhjustada erinevaid testide läbimise probleeme.

- Enne elemendi vajutamist, määrata kindlaks, et süsteem näeb seda elementi. Vastasel juhul vajutatakse muutujat enne selle laadimist ning testid ebaõnnestuvad.

- Kasutada *Sleep* funktsiooni nii vähe kui võimalik, kuna *Sleep* sõltub enamasti internetikiirusest ja riistvarast.

8 Kasutatud kirjandus

[1] „Twiki,“ [Võrgumaterjal]. Saadaval: https://twiki.cern.ch/twiki/bin/view/EMI/RobotFrameworkAdvancedGuide [Kasutatud 21.04]

[2] „Stackexchange,“ [Võrgumaterjal]. Saadaval: https://sqa.stackexchange.com/questions/5735/robot-framework-and-browser-support [Kasutatud 22.04]

[3] „Quintagroup,“ [Võrgumaterjal]. Saadaval: http://quintagroup.com/cms/python/robot-framework [Kasutatud 22.04]

[4] „Codecentric,“ [Võrgumaterjal]. Saadaval: https://blog.codecentric.de/en/2012/01/robot-framework-ide-ride-overview/ [Kasutatud 22.04]

[5] „Sugarcrm,“ [Võrgumaterjal]. Saadaval: http://support.sugarcrm.com/Resources/Supported_Platforms/Sugar_7.8.x_Supported_Platforms/ [Kasutatud 22.04]

[6] „Ibm,“ [Võrgumaterjal]. Saadaval: https://www.ibm.com/developerworks/ru/library/l-SugarCRM_part1/ [Kasutatud 23.04]

[7] „Satisfice,“ [Võrgumaterjal]. Saadaval: http://www.satisfice.com/blog/archives/118 [Kasutatud 22.04]

[8] “Tutorialspoint,” [Võrgumaterjal]. Saadaval: https://www.tutorialspoint.com/xpath/xpath_overview.htm [Kasutatud 24.04]

[9] “Tutorialspoint,” [Võrgumaterjal]. Saadaval: https://www.tutorialspoint.com/xpath/xpath_expression.htm [Kasutatud 24.04]

[10] “Stackoverflow,” [Võrgumaterjal]. Saadaval: http://stackoverflow.com/questions/1065433/what-is-dynamic-programming [Kasutatud 24.04]

[11] Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black, ”Foundations of software testing,” [Võrgumaterjal]. Saadaval: https://www.utcluj.ro/media/page_document/78/Foundations%20of%20software%20testing%20-%20ISTQB%20Certification.pdf [Kasutatud 05.05.2017]

[12] „Slideshare,“ [Võrgumaterjal]. Saadaval: https://www.slideshare.net/ak-itconsulting.com/7-28401420 [Kasutatud 06.05.2017]

[13] “Codecentric,” [Võrgumaterjal]. Saadaval: https://blog.codecentric.de/en/2012/04/robot-framework-tutorial-a-complete-example/ [Kasutatud 07.05.2017]

[14] „GitHub Inc,“ [Võrgumaterjal]. Saadaval: https://github.com/mozilla/geckodriver/issues/659 [Kasutatud 14.05.2017]

[15] “Google Inc,” [Võrgumaterjal]. Saadaval: https://groups.google.com/forum/#!topic/selenium-users/yoJX_YdnYCY [Kasutatud 14.05.2017]

[16] “Ryreitsma,” [Võrgumaterjal]. Saadaval: http://ryreitsma.blogspot.com.ee/2011/07/software-has-new-quality-model-iso.html [Kasutatud 18.05.2017]

[17] Mark Fewster, Dorothy Graham, “Software Test Automation,” [Võrgumaterjal]. Saadaval: http://read.pudn.com/downloads11/ebook/44105/Software%20Test%20Automation.pdf [22.05.2017]