TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Selenide Page Object klassi generaatori arendus

Magistritöö

Üliõpilane: Karl Jürgenson
Üliõpilaskood: 143674 IAPM
Juhendaja: Maili Markvardt, MSc

Tallinn
2018

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: Karl Jürgenson

Kuupäev: 09.05.2018

(*kuupäev*) (*allkiri*)

Annotatsioon

Käesoleva magistritöö eesmärgiks oli realiseerida Page Object klassi generaator, mis toetab Selenide raamistikku.

Töö käigus uuriti eksisteerivaid lahendusi, kuid paraku sellist vahendit töö autor ei leidnud. Sellest hoolimata analüüsiti nende omadusi ja puudusi. Nendest omakorda koostati uued nõuded loodava vahedi jaoks.

Töö tulemusena valmis esmane prototüüp, mis võimaldab kasutajal Selenide raamistikus koostada Page Object klasse. Et selle korrektsuses veenduda, koostati test veebilehe peal kontroll. Et testi lugeda õnnestunuks panti paika kriteeriumid, mis ka täideti.

Töö lähtekood on avaldatud siin: https://github.com/Salamander75/ThesisProject

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 47 leheküljel, 8 peatükki, 16 joonist, 1 tabelit.

Abstract

Development of Selenide Page Object class generator

The goal of this master thesis was to implement a Page Object generator which has support for Selenide framework.

During the work, an investigation about existing solutions was conducted but the author of this thesis did not find any solution which would support Selenide. Nevertheless, the existing solutions were examined about their advantages and disadvantages. A conclusion was made and the new requirements were made for the technical output of this thesis.

As a result of this work, a Page Object generator was implemented which supports Selenide. To be sure that the program works correctly, a test criteria was set up which the program should meet. The test was successful.

The source code of this thesis is avaiable here: https://github.com/Salamander75/ThesisProject

The thesis is in Estonian and contains 47 pages of text, 8 chapters, 16 figures, 1 tables.

Lühendite ja mõistete sõnastik

Page Object	***Page Object*** Selline klass koodis, mis sisaldab mingi kindla lehe HTML elementide lokaatoreid
Selenide	***Selenide*** Raamistik kasutajaliideste testimiseks, mis töötab Selenium peal
Selenium	***Selenium*** Vahend, et manipuleerida brauserit. Kasutatakse sagedasti kasutajaliideste testimisel
Capture-and-replay	***Capture-and-replay*** Võte, kus tehakse mingid tegevused ja seejärel need tegevused salvestatakse, et taastoota viga
Regressioontest	***Regressiontest*** Test, mis kontrollib, et olemasolev funktsionaalsus pole uue arendusega katki läinud
XML	***Extensible Markup Language*** On märgistuskeel, kus informatsioon on struktureeritud ning mis on loetav nii inimesele kui ka masina jaoks. Enamasti kasutatakse seda info vahetamiseks veebirakenduste vahel läbi veebiteenuste [10]
AJAX	***Asynchronous Javascript and XML*** On vahend, mille abil saab teha päringuid serverisse, ilma et peaks uuendama kogu veebilehe sisu [16]
DOM	***DOM*** On programmeerimisliides, et suhelda XML, XHTML ja HTML dokumentidega. See määrab dokumendi struktuuri ja viisid, kuidas dokumenti manipuleerida [22]

Elemendi lokaator ***Element locator***

On käsk või viis, kuidas ligi pääseda veebilehe HTML elemendile, mida plaanitakse töödelda [20]

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

Testimine on oluline osa tarkvara arenduse elutsüklis. Selle abil antakse infot tarkvara toote tellijale, et mis seisus on tarkvara kvaliteet ning selles esinevad võimalikud riskid. Testimise üheks osaks on kasutajaliideste automaattestimine. Selle mõte on eelkõige testida, et *front-end* poolne osa rakendusest töötaks. Nende koostamise üheks osaks on Page Object klasside loomine, mis aga on suurte süsteemide puhul väga ajamahukas töö. Selle jaoks on loodud juba abivahendeid, mis genereerivad neid. Küll aga pole sellist, mis toetaks Selenide raamistikku. Seetõttu üritatakse käesolevas töös koostada selline raamistik, mis võimaldab seda manuaalset tööd vähendada ning mis toetakse Selenide raamistikku.

1.1 Taust ja probleem

Antud töö sai alguse sellest, kui töö autor soovis koostada automaatteste veebipõhiste kasutajaliideste testimiseks ning selle üheks osaks oli Page Object klasside loomine. Antud tegevus aga oli väga ajanõudlik, mistõttu autor otsustas, et otsib mingit lahendust sellele. Kuigi selliseid vahendid on juba olemas, mis genereerivad Page Object klasse Seleniumile, siis paraku aga autorile need ei sobinud, kuna vaja oli sellist genereerijat, mis tekitaks Selenide põhjal neid, kuna see oli autori töökohas kasutuses laialdaselt ning minna üle Selenium peale, polnud lihtsalt võimalik. Põhjuseks on lihtsalt väga suur ressurssi kulu, mida pole kusagilt saada.

Eelnevalt mainitud asjaolude tõttu, plaanitakse luua selline raamistik, mis suudaks luua Selenide toetavaid Page Object klasse. Antud rakendusest võiks kasu saada testijad, kes samamoodi kasutavad Selenide raamistikku kasutajaliideste automatiseerimiseks.

Töö valmis 2018. aasta kevadel Tallinna Tehnikaülikoolis

1.2 Ülesande püstitus

Peamine töö eesmärk on luua selline vahend, mis suudaks koostada Page Object klasse ka Selenide raamistikule. Teine eesmärk on see, et loodud vahendi tõttu peab minema automaattesti kirjutamine ajaliselt kiiremaks.

1.3 Metoodika

Esmalt uuritakse olemasolevaid lahendusi. Uuritake nende omadusi ning puudisi. Sellejaoks võetakse testimiseks natuke mahukam veebirakendus ning kasutatakse olemasolevaid vahendeid, et koostada väljavalitud veebilehe peale Page Object klasse. Sellest tulenevalt saab koostada vajadusel uusi nõudeid. Ning lõpuks tehakse eksperiment sama veebilehe peal, kus katsetati eksisteerivaid lahendusi, ainult et seekord siis loodud prototüübiga. Et teada saada ka ajalist võitu loodud prototüübi kasutamisel, siis Page Object klass koostatakse esmalt manuaalselt ning seejärel sama lehe kohta koostatakse Page Object klass loodud rakenduse abil. Mõõdetakse aega mõlema lähenemisega ning aegade võrdlusel saab tõestada rakenduse effektiivsust.

1.4 Ülevaade tööst

Töö alguses antakse lühike ülevaade Selenium vahenidst, millega saab manipuleerida brausereid ning Selenide raamistikust. Samuti uuritakse olemasolevaid lahendusi ning nende omadusi ja puudusi. Nende pealt saab koostada loodava prototüübi nõuded. Järgmisena modelleeritakse süsteemi abstraktne vaade, mis peaks selgeks tegema süsteemi tööpõhimõtte. Seejärel on ära toodud ka kasutusjuhtude diagramm ning tuuakse välja peamine töövoog. See võiks anda esialgse ettekujutuse loodavast rakendusest. Pärast seda kirjeldatakse detailsemalt, et kuidas süsteem on üles ehitatud ning lõpuks sooritatakse test loodud vahendi jaoks. Välja toodud on ka arendused, mis võiksid tulevikus ka süsteemil juures olla.

2. Teoreetiline taust

Selles peatükis antakse ülevaade Selenium ja Selenide raamistikest ning nende erinevustest. Seejärel seletatakse lahti üldised printsiibid automaattestide kirjutamisest kasutajaliideste jaoks. Ning lõpuks uuritakse olemasolevaid lahendusi ning nende omadusi ja puudusi. Nende pealt moodustatakse nõuded, mida arendatav prototüüp võiks sisaldada.

2.1 Selenium

Selenium on laialdaselt kasutatud vahend, et manipuleerida brauserit ning seetõttu ka kasutatakse seda automaattestide koostamiseks kasutajaliideste testimiseks [1]. Selenium koosneb kahest alamosast: Selenium IDE ning Selenium Webdriver. Selenium IDE võimaldab koostada teste *capture-and-replay* stiilis, mis sobib selleks, et kiirelt uuesti tekitada leitud viga. Selenium Webdriver võimaldab aga koostada programmaatiliselt regressioonteste ning test komplekte. Toetatud on mitmed programmeerimiskeeled: Java, Python, Ruby, C#, JavaScript [1].

2.2 Selenide

Selenide on Selenium Webdriver peal töötav raamistik, mis on just mõeldud selge süntaksiga kasutajaliideste automaattestide loomiseks [2]. Raamistik on välja arendatud firma Cordeborne poolt [3]. Selle raamistiku eesmärk on see, et ei pea muretsema detailide pärast (AJAX vastuse ootamisel timeout määramine, *Webdriver* klassi igakordne väljakutse), vaid et kasutaja saaks kontsentreeruda testiloogika kirjutamisele. Joonis 1 on võrdlus Selenide ja Selenium vahel sama probleemi lahendamisel ning saab näha, et Selenidega kirjutatud testid on oma süntaksilt lühemad ja selgemad.

Järgnevalt on illustreeritud Google otsingu test, kus on kasutatud nii Selenide ja Seleniumit, et sooritada sama tegevust. Tegevus on siis järgmine: minnakse Google pealehele, sisestatakse otsingusõna ning otsitakse tulemusi. Seejärel eeldatakse, et kindla pealkirjaga vaste eksisteerib otsingutulemustes. Selenide peidab ära järgnevad detailid (vt Joonis 1):

- Draiveri määramine
- Draiveri klassi igakordne väljakutse mingi tegevuse sooritamiseks
- Elemendi nähtavuse kontroll ning selle nähtavuse järgi ootamine. Selenide on vaikimisi ootamise määranud juba ära 4 sekundit [2], kuid Seleniumis peab seda ise tegema

```
    public void googleSearchWithSelenide() {
        open("https://www.google.ee/");
        $(By.id("lst-ib")).setValue("How to cook pasta");
        $(By.linkText("Google otsing")).click();
        $(By.linkText("How to Cook Pasta Video and Steps - Real
Simple"))
                .shouldBe(Condition.visible);
    }

    public void googleSearchWithSelenium() {
        WebDriver driver = new ChromeDriver();
        driver.get("https://www.google.ee/");
        driver.findElement(By.id("lst-ib")).sendKeys("How to cook
pasta");
        driver.findElement(By.linkText("Google otsing")).click();
        WebDriverWait wait = new WebDriverWait(driver, 4);
        WebElement element = wait.until(ExpectedConditions.
                visibilityOf(driver.findElement(By.linkText("How to
Cook Pasta Video and Steps - Real Simple"))));
    }
```

Joonis 1. Google otsingu test Selenide ja Seleniumiga eraldi tehtult

2.3 Kasutajaliideste automaattestide kirjutamise üldised printsiibid

Automaattestide kirjutamisel kasutajaliideste testimiseks üldine rusikareegel on kasutada Page Object mustrit. See tähendab siis seda, et testide jaoks vajalikud elementide lokaatorid oleksid kokku pakitud ühte klassi. See aitab koostada robustsemaid teste ning ei pea muretsema elementide lokaatorite pärast. Lisaks sellele, on eelis see, et kui näiteks on mingit elementi kasutatud mitu korda testi sees, siis piisab ainult Page Object klassis lokaatori muutmisest. Muidu ilma selleta peaks muutma igal pool eraldi testi sees lokaatoreid, mis teeb testi haldamise raskemaks.

Joonis 2 peal on välja toodud test ilma Page Object klassita ning Joonis 3 ja Joonis 4 peal on kirjeldatud test koos Page Object klassiga. Otsitakse neti.ee leheküljel vastet ning valitakse esimene ettejuhtuv link. Seejärel tehakse uus otsing ning samamoodi valitake esimene tulemus. Selgelt on näha, et Page Object klass annab parema loetavuse ning kergem on hallata testi, kui peaks näiteks lokaatorid muutuma (kasutatud on Selenide süntaksit siin).

```
    public void testNetiPageWithoutPageObjectClass() {
        open("http://www.neti.ee/");
        $(By.cssSelector("#search-bar > div:nth-child(2) >
input")).setValue("Arvutite ost");
        $(By.cssSelector("#search-bar > div:nth-child(4) >
input")).click(); // Otsi nupp
        $(By.cssSelector("#main-content > div.main-content." +
                "fc-headertab-content > ul:nth-child(5) > li:nth-child(1) >
h3 > a.out")).click(); // Esimene vaste
        navigator.back(); // navigeeri neti lehele tagasi
        $(By.cssSelector("#search-bar > div:nth-child(2) > input"))
                .setValue("Kasutatud monitoride ost"); // Teine otsing
        $(By.cssSelector("#main-content > div.main-content.fc-headertab-
content" +
                " > ul:nth-child(5) > li:nth-child(1) > h3 > a")).click();
// Esimene vaste
    }
```

Joonis 2. neti.ee lehekülje otsimine ning tulemus ilma Page Object klassita.

```
    public void testNetiPageWithPageObjectClass() {
        open("http://www.neti.ee/");
        NetiPage page = new NetiPage();
        page.setSearchValue("Arvutite ost")
                .clickSearchButton()
                .clickFirstResultLink();
        navigator.back();
        page.setSearchValue("Kasutatud monitoride ost")
                .clickSearchButton()
                .clickFirstResultLink();
    }
```

Joonis 3. neti.ee lehekülje otsimine kasutades Page Object klassi

```
import org.openqa.selenium.By;
import static com.codeborne.selenide.Selenide.$;

public class NetiPage {

    public NetiPage setSearchValue(String valueToSearchFor) {
        $(By.cssSelector("#search-bar > div:nth-child(2) >
input")).setValue(valueToSearchFor);
        return this;
    }

    public NetiPage clickSearchButton() {
        $(By.cssSelector("#search-bar > div:nth-child(4) >
input")).click();
        return this;
    }

    public NetiPage clickFirstResultLink() {
        $(By.cssSelector("#main-content > div.main-content.fc-headertab-
content" +
                " > ul:nth-child(5) > li:nth-child(1) > h3 > a")).click();
        return this;
    }
}
```

Joonis 4. Page Object klass neti.ee leheküljele

Page Object klasside loomisel tasuks meeles pidada seda, et see ei pea sisaldama kõiki elemente, mida testitav kasutajaliides sisaldab. Tuleks sisalda ainult neid elemente, mida reaalselt vaja on. Samuti peaks Page Object klass sisaldama meetodeid, mis siis täidavad mingit ülesannet. Näiteks meetod mis vajutab nupule, kasutades selleks elemendi lokaatorit (vt Joonis 4). Ning meetodid võiksid tagastada kas fundamentaalseid tüüpe (String, Date), omaenda klassi instantsi või siis kui tegu on mingi muu lehele navigeerimisega, siis võiks üldjuhul tagastada selle uue lehe Page Object klassi [7].

2.4 Olemasolevad lahendused

Tuntumad olemasolevad lahendused on Chrome Page Object Generator [4] ja SWD Page Recorder [5]. Kuigi olemasolevaid lahendusi on veelgi, siis need said valitud eelkõige oma esinemise järjekorras Google otsingutulemustes või MIT litsentsi [19] kättesaadavuse tõttu. Antud alamjaotuses uuritakse mõlemat rakendust lähemalt. Et nende puudustest ja eelistest rohkem teada saada, selle jaoks võetakse testimise alla NETI otsingumootori pealeht (http://www.neti.ee/) ning koostatakse selle peal kasutuslugu, mille jaoks siis koostatakse uuritavate vahendite abil Page Object klassid. Põhjus miks autor otsustas selle veebilehe testimise aluseks võtta on see, et ta sisaldab mitmeid varieeruvaid HTML elemente (*text*, *img*, *input*, *link*) ning lisaks sellele, on seal olemas virtuaalne klaviatuur, millega saab tähti valida otsingu sisestamiseks ilma füüsilist arvuti klaviatuuri kasutamata. Sellest tingituna on kasutuslugu järgmine:

- Sisene NETI otsingumootori pealehele
- Ava virtuaalne klaviatuur (asub otsingulahtrist paremal ning on klaviatuuri ikoon peal)
- Sisesta virtuaalse klaviatuuri pealt *postimees*
- Otsi tulemust ning ava esimene vaste
- Kontrolli, et avatud link oleks (https://www.postimees.ee/)

Kuigi eelnevalt mainitud eksisteerivad lahendused ei toeta ükski Selenide raamistikku, siis uuritakse neid siiski, et leida vajadusel uusi nõudeid. Olemasolevad lahendused võiksid siiski täita ära järgmised baasnõuded:

- Toetus Java programmeerimiskeelele
- Teenusmeetodite genereerimine (nt nupuvajutus meetod, vormi täitmise meetod)

2.4.1 Chrome Page Object Generator

Antud vahend [4] on Chrome jaoks loodud laiendus, millega on võimalik paari nupu vajutusega koostada *Page Object* klass antud lehekülje kohta. See on kirjutatud JavaScriptis ning toetab *Page Object* genereerimist Java, C# keeles ning Robot Framework jaoks. Lisaks sellele saab enne faili genereerimist seadistada, et mis HTML elemente oleks vaja, kas genereerida ka juba vaikimisi andmeväljade täitmisfunktsioonid (st, et kui nt HTML element on *input* ning tüüp on *text*, siis tekitatakse selline meetod, mis täidab seda HTML elementi tekstiga, eeldades, et antakse sisend ette). Ometigi, leiti vahendis puudujääke:

- Genereeritakse iga lehe elemendi kohta (mis on nähtavad kasutajale) lokaatorid, küll aga ei pruugi kõiki elemente üldse vaja minnagi kasutajal ning see võib tekitada hoopis segadust
- Samuti funktsiooni ja muutujate nimed, mis genereeriti ette ära, on tihtipeale segased või naeruväärselt pikad

Üldiselt jäi mulje see, et vahend on rohkem sobilik lihtsamate veebilehekülgede jaoks. Arvestades seda, kui palju segast ja pikkade nimedega funktsioone, muutujaid ning üleliigseid ebavajalikke elemente genereeriti (vt Lisa 1), siis on tunne et läheb rohkem aega genereeritud failist arusaamisele, kui kõike seda ise teha manuaalselt.

2.4.2 SWD Page Recorder

Teine vahend, mis väärib tähelepanu on SWD Page Recorder [5]. Võrreldes Chrome Page Object Generator-ga on selle peamine eelis see, et kasutajal avaneb brauser, kus on tal võimalik ise valida neid elemente, mida ta soovib. Lisaks sellele, saab kasutaja ise valida, kas ta soovib lokaatorit *id, class, cssSelector, linkText* või *xPath* järgi ning kasutaja peab määrama valitud elementide nimetused. Samuti on toetatud mitmeid programmeerimiskeeli: Java, C#, Python, Ruby, Powershell. Kuigi see vahend on päris hea, siis olid ka sellel puudused:

- Kui Chrome Page Object Generator suutis tekitada meetodid (vormi täitmis meetodid ja nupuvajutus meetodid), siis SWD Page Recorder genereerib ainult muutujate nimed ning nende lokaatorid
- Et selle vahendi brauserit kasutada tuleb kasutajal alla laadida WebDriver
- Toetus on ainult Windows operatsioonisüsteemile [5]

2.4.3 Arendatav vahend

Tabel 1 on välja toodud kokkuvõte vaadeldud vahenditest, kus on välja toodud nende omadused ja puudused. Ühtlasti saab selle pealt kokku panna nõuded, et mis võiks arendatavas vahendis olla.

	Chrome Page Object Generator	SWD Page Recorder
Selenide raamistiku toetus	Ei	Ei
Java süntaksi toetus	Jah	Jah
Teenusmeetodite genereerimine	Jah	Ei
Automaatne nimede määramine	Jah	Ei
Automaatne lokaatorite määramine	Jah	Ei
On toetatud üle operatsioonisüsteemide	Jah	Ei
On vaja töötamiseks WebDriver-t	Ei	Jah

Tabel 1. Kokkuvõte uuritud olemasolevatest lahenditest

Kuna käesoleva töö eesmärk oli siiski saada sellist Page Object generaatorit, mis toetab ka Selenide raamistikku, siis selle olemasolu on peamine nõue. Järgnevad nõuded peaksid olema samamoodi täidetud:

- Kasutaja peab saama ise valida, mis elemendi kohta genereerida infot (st ei genereerita kõikide HTML elementide kohta infot nagu Chrome Page Object Generator vahendis tehti, vaid kasutaja ise valib endale sobiva elemendi). Seetõttu minnakse sama teed nagu SWD Page Recorder [5] vahendis, et brauseri kaudu valitakse sobivaid elemente

- Kasutaja peab ise saama valida elemendile lokaatori, sest automaatne lokaatori tüübi määramine pole mõistlik. Näiteks, kui on kasutajapoolt valitud mingi HTML element ning tal eksisteerib *id* attribuut ja see määratakse talle automaatselt lokaatoriks, siis võib juhtuda nii, et sama *id* eksisteerib parasjagu ka mõnel muul HTML elemendil ning seetõttu testis sellest kasu pole, kuna test ei tea, millist HTML elementi kasutada (duplikaat *id*)

- Page Object programmeerimiskeeleks on Java. Kuna käesoleva töö raames realiseeritakse esialgne prototüüp, siis piirdutakse ainult ühe programmeerimiskeelega.

- Teenusmeetodite genereerimine

- Rakendust peab saama ka käivitada peale Windows operatsioonisüsteemi ka Linux ja Mac keskkonnas, eeldades, et nendes on installeeritud Java Virtual Machine [6].

- Kasutaja peaks saama läbi rakenduse brauserisse ligi nii, et ei peaks kasutama WebDriver-t

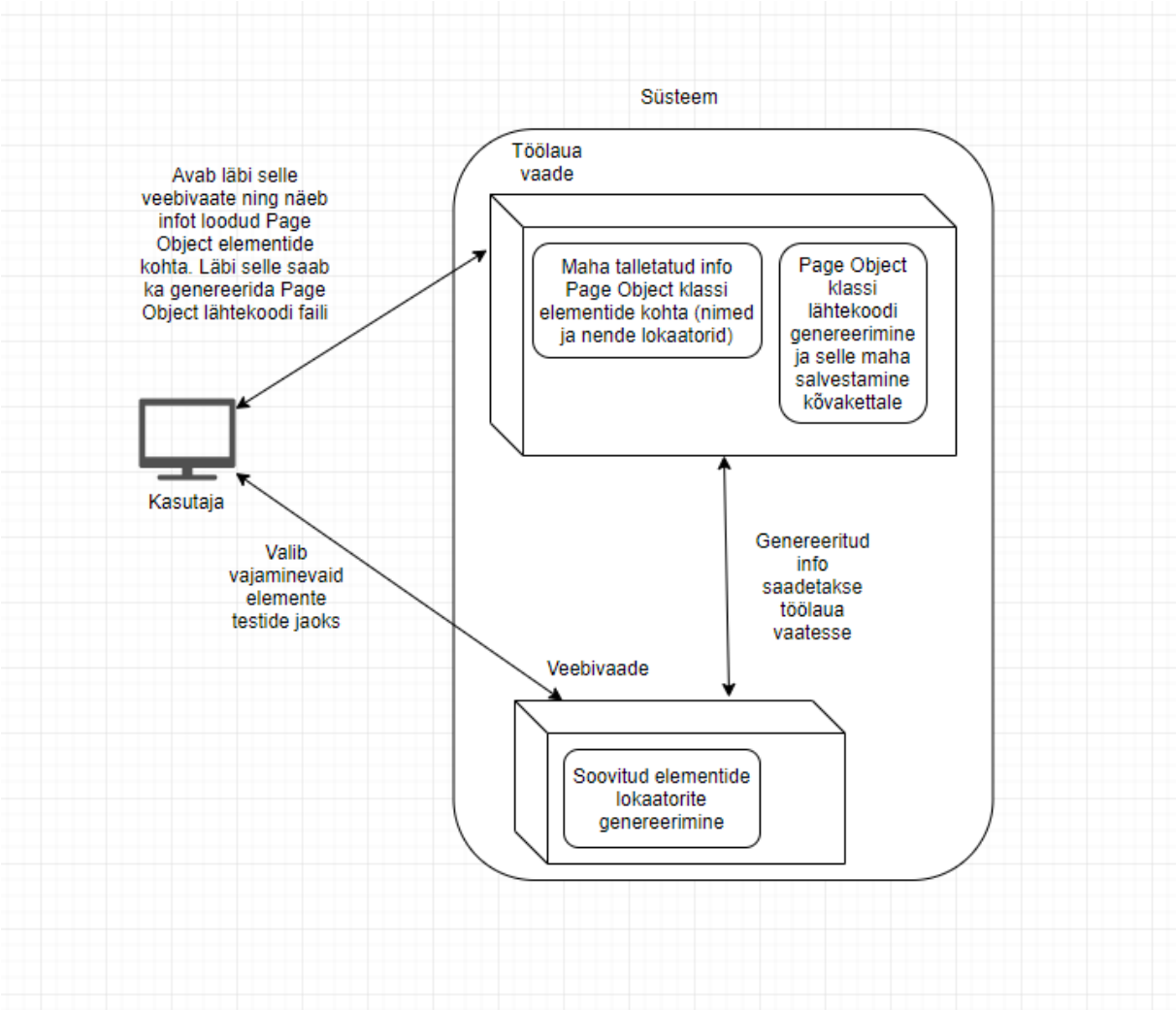
3. Süsteemi analüüs

Antud peatükis analüüsitakse, et milline peab loodav süsteem välja nägema. Esitletakse loodava süsteemi tööpõhimõte ning võimalikud kasutusjuhud, mida süsteemi kasutaja saab teha. Samuti tuuakse välja tegevusdiagramm, kus on ära esitletud peamine töövoog.

3.1 Süsteemi tööpõhimõte

Loodav süsteem jaotatakse kaheks alamosaks: 1) Töölaua vaade 2) Veebivaade. Töölaua vaates on kasutajal võimalik luua uusi Page Object klassi elemendi instantse, olemasolevaid elemente muuta või siis neid kustutada. Kasutaja saab elemente kas manuaalselt lisada (defineerides ise elemendi lokaatorite väärtused) või kasutades selleks veebivaadet, mis genereerib soovitava elemendi lokaatorid juba ette (kui see on võimalik) ning kasutaja peab kõigest elemendi nimetuse määrama. Lisaks sellele saab kasutaja genereerida valmis Page Object klassi lähtekoodi, mida ta saab vaadata ning vajadusel muudatused sisse viia, et see sobituks kasutaja vajadustega. Kui kasutaja on rahul loodud lähtekoodiga ning on valmis seda kasutajaliideste testides kasutama, siis ta peab ka veel maha salvestama loodud lähtekoodi java failina. Teine tähtis osa ning peamisi põhjuseid süsteemi loomisel on veebivaade, mida kasutaja saab avada läbi töölaua vaate. Seal on kasutajal võimalik valida nähtavaid veebielemente ning valides sealt soovitava elemendi, genereeritakse selle kohta võimalikud lokaatoreid, mida siis testid kasutama hakkavad.

Joonis 5 on esitatud abstraktne illustratsioon süsteemi tööpõhimõttest.



Joonis 5. Süsteemi abstraktne vaade

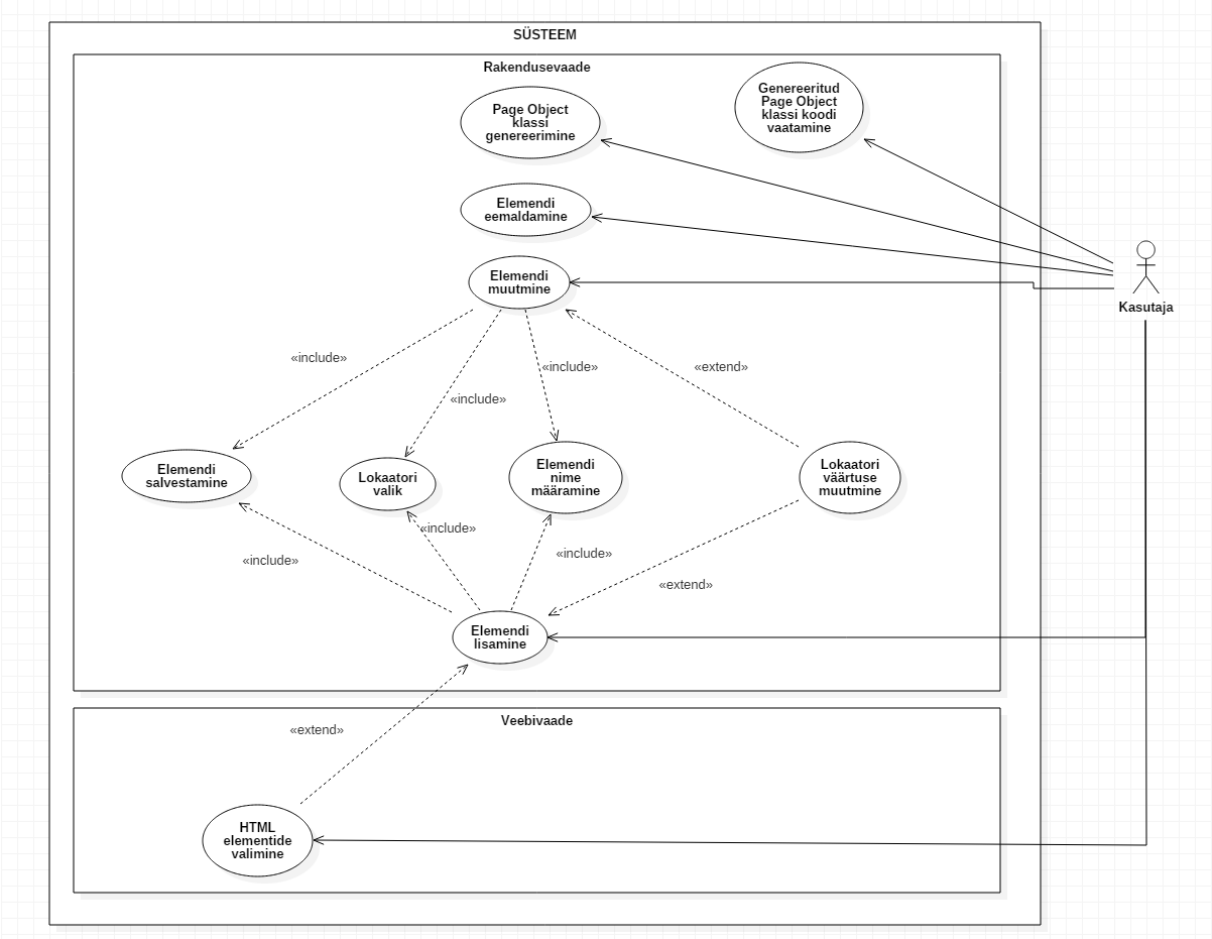
3.2 Tegutsejad

- Kasutaja

Kasutaja võib antud kontekstis olla igaüks, kuid eelkõige saab kasutajaks olema tarkvara testija.

3.3 Kasutusjuhtude mudel

Joonis 6 peal on illustreeritud kasutusjuhtude mudel, mida kasutajal on võimalik teha rakenduses.



Joonis 6. Süsteemi kasutusjuhtude mudel

3.3.1 Kasutusjuhtude kirjeldused

Antud osas on välja toodud kasutusjuhtude kõrgformaadis tekstikirjeldused.

Kasutusjuht: Elemendi lisamine

Tegutsejad: Kasutaja

Kirjeldus: Kasutaja vajutab rakenduses nuppu *Add New* , mille peale talle kuvatakse uue elemendi lisamise sisestusvorm. Seal on kasutajal vaja täita elemendi nimi, mis peab olema unikaalne ning peab olema valitud vähemalt üks lokaatori tüüp (*id, class, name, cssSelector,*

Xpath) ning see peab olema väärtustatud. Kui need on täidetud, siis on võimalik maha salvestada uus element.

Kasutusjuht: Elemendi muutmine

Tegutsejad: Kasutaja

Kirjeldus: Kasutaja valib salvestatud elementide loetelust elemendi, mida ta muuta soovib, mille peale kuvatakse talle eelnevalt maha salvestatud elemendiga seonduv info. Andmeid elemendi kohta on võimalik muuta ning kui need on korrektsed, siis salvestatakse muudatused.

Kasutusjuht: Elemendi salvestamine

Tegutsejad: Kasutaja

Kirjeldus: Kasutaja vajutab rakenduses nuppu *Save* , mille peale salvestatakse maha hetkel käsitletav element oma infoga ning seda arvetatakse Page Object klassi genereerimisel. Kui kasutajal on defineerimata elemendi nimi või on elemendi lokaator valimata või lokaatori väärtus on tühi, siis kuvatakse veateade.

Kasutusjuht: Lokaatori valik

Tegutsejad: Kasutaja

Kirjeldus: Kasutaja valib ühe lokaatori tüübi elemendi kohta, mille abil genereerida Page Object klassi tema lokaator. Kui kasutaja jätab selle valimata ja tahab salvestada, siis kuvatakse veateade.

Kasutusjuht: Elemendi nime määramine

Tegutsejad: Kasutaja

Kirjeldus: Kasutaja määrab elemendile nime kas uue elemendi loomisel või muutmisel. Elemendi nimi peab olema unikaalne ning kohustuslik. Nende piirangute rikkumisel antakse salvestamisel veateade.

Kasutusjuht: Lokaatori väärtuse muutmine

Tegutsejad: Kasutaja

Kirjeldus: Kasutaja pole näiteks rahul talle ette genereeritud lokaatori väärtusega, siis ta muudab seda vastavalt oma soovile.

Kasutusjuht: Elemendi eemaldamine

Tegutsejad: Kasutaja

Kirjeldus: Kasutaja vajutab rakenduses nuppu *Delete* , mille peale temaga seonduv info kustutatakse ning seda ei kaasata Page Object klassi genereerimise hulka.

Kasutusjuht: Page Object klassi genereerimine

Tegutsejad: Kasutaja

Kirjeldus: Kasutaja vajutab rakenduses nuppu *Generate*, mille peale genereeritakse Page Object klass. Andmed, mille põhjal see klass koostatakse võetakse kasutaja poolt maha salvestatud elementidest ning nende infost.

Kasutusjuht: Genereeritud Page Object klassi koodi vaatamine

Tegutsejad: Kasutaja

Kirjeldus: Kasutaja vajutab rakenduses nuppu *Generate*, mille peale genereeritakse Page Object klass. Selle tulemusena saab kasutaja vaadata, et milline genereeritud kood tuleb ning vajadusel viia sisse muudatused, ennem kui füüsiliselt arvuti kõvakettale salvestada.

Kasutusjuht: HTML elementide valimine

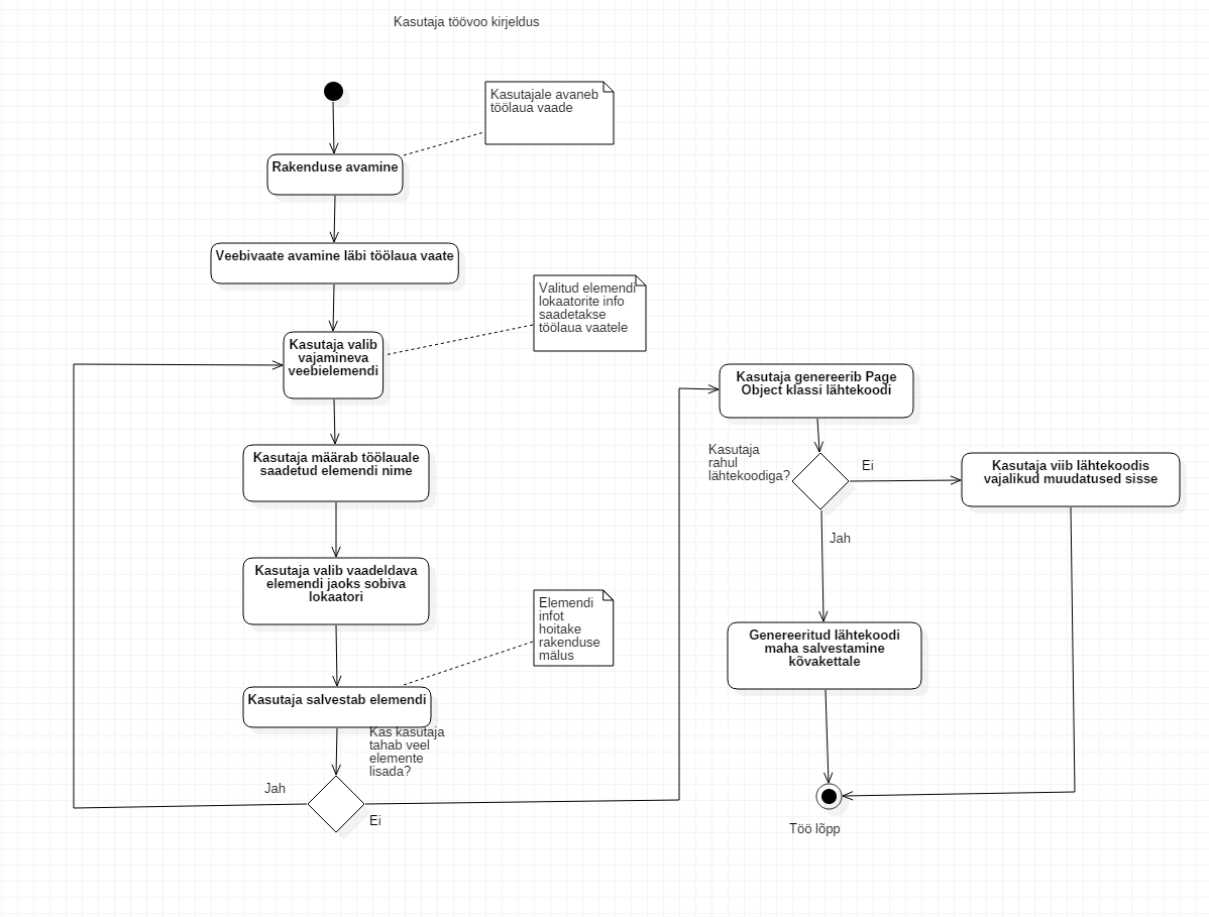
Tegutsejad: Kasutaja

Eeltingimus: Kasutaja on avanud veebivaate, vajutades rakenduses nuppu *Webview*

Kirjeldus: Kasutaja liigutab hiirekursorit üle veebilehe elementide, mille peale talle eristatakse veebilehe nähtav element kollase taustaga, mille peal parajasti tal hiirekursor on. Vajutades selle elemendi peale, mis parasjagu on eristatud teistest kollase taustaga, avaneb talle modaalkast, kus on välja toodud elemendi erinevat tüüpi lokaatorid (*id, class, name, XPath, cssSelector*) ning nende väärtused. Kui elemendil puudub mõni nendest, siis kuvatakse modaalkuvas seda välja tühjana. Kui kasutaja on kindel, et tahab seda elemendi Page Object klassi genereerimisse kaasata, vajutab ta *Select*, mille peale saadetakse veebipoolelt valitud element rakendusepoolele, kus saab seda edasi töödelda.

3.4 Töövoo kirjeldus

Joonis 7 on kirjeldatud rakenduse kasutamise peamine töövoog. Kõigepealt kasutaja avab rakendus ning sealt omakorda veebivaate, mis on peamine vahend, et genereerida lokaatoreid soovitud veebilehe elementidele. Kui kasutaja on valinud veebivaates elemendi ära, siis saadetakse see edasi töölaua kuvale, kus ta peab määrama nime elemendile ja valima talle sobvia lokaatori. Kui kasutajal rohkem pole vaja elemente, siis jääb tal üle vaid genereerida Page Object kood ning ta saab kas kopeerida koodi otse genereerimisvaatest enda projekti või siis salvestada maha .java failina.



Joonis 7. Peamise töövoo kirjeldus

4. Kasutatavad tehnoloogiad

Järgnevalt antakse ülevaade, et milliseid raamistike on süsteemi loomisel kasutatud.

4.1 Java SDK 8

Java SDK on arendusvahend, et koostada Java rakendusi. See sisaldab endas *Standard Edition* platvormi, mis võimaldab koostada *Desktop* ja serverirakendusi. Lisaks sellele saab *Enterprise Edition* abil koostada veebirakendusi. SDK sisse on integreeritud JRE (Java Runtime Environment), mis on vajalik selleks, et saaks üldse java rakendusi jooksutada. Seetõttu on võimalik Java rakendusi jooksutada erinevate operatsioonisüsteemide peal (Windows, Linux, macOS) [8].

4.2 JavaFX

JavaFX on JDK osa, mille abil saab koostada nii kliendi, kui ka veebirakendusi. See loodi eesmärgiga asendada *Swing*-i, mis on ka JDK komponent [21], et koostada kasutajaliidesega rakendusi [9]. Erinevalt *Swing*-st on JavaFX eelis see, et ta võimaldab ka kasutada *Desktop* rakenduste loomisel samu tehnoloogiaid, mida ka kasutatakse veebirakendustes. Samuti JavaFX-s on võimalik kasutajaliidest koostada *XML*-s (lisaks Java koodis kirjutamisele), mis aitab hästi eraldada kasutajaliidese ning äriloogika ja mudelid üksteisest. Seetõttu on JavaFX rakenduses hea implementeerida mudel-vaade-kontroller (*Model View Controller)* mustrit. Kuid peamine põhjus on see, et sellesse on sisse ehitatud veebibrauser (*WebView)* [9], mis aitab rahuldada nõuet, et ei pea alla laadima WebDriver-t, et brauser avaneks rakenduses (vt 2.4.3)

4.3 Javascript

JavaScript on veebirakenduste üks peamisi tehnoloogiaid. See on kliendipoolne tehnoloogia, mis tähendab, et skriptid, mis javascriptis koostatakse, käivituvad kliendi brauseris (kui rääkida veebilehekülgedest). See võimaldab lisada rakendusele interaktiivust ning on laialdaselt kasutusel. Javascript toetab funktsionaalset, sündmustepõhist, imperatiivset ja objektorienteeritud programmeerimist [11].

4.4 CSS

Cascading Style Sheet (CSS) on tehnoloogia, mis on mõeldud veebi kujunduse jaoks. Selle abil on võimalik paika panna näieks veebilehe osade paigutus, defineerida ära tähemärkide stiil. Samuti saab defineerida ka seda, et kui on mitu erinevat platvormi, mille abil leheküljele ligi saab (nt arvuti, mobiil), et siis kuidas lehekülje kujundus peaks olema erineva platvormi puhul [12].

4.5 Maven

Maven on projekti ehitamise vahend [13]. Selle abil saab ehitada valmis paki ning hoida infot projekti sõltuvuste (*dependencies*) kohta. Et projektis seda kasutada, peab eksisteerima *pom.xml* fail, kus on talletatud projekti ehitamiseks kõik vajaminev info [14].

4.6 JSON

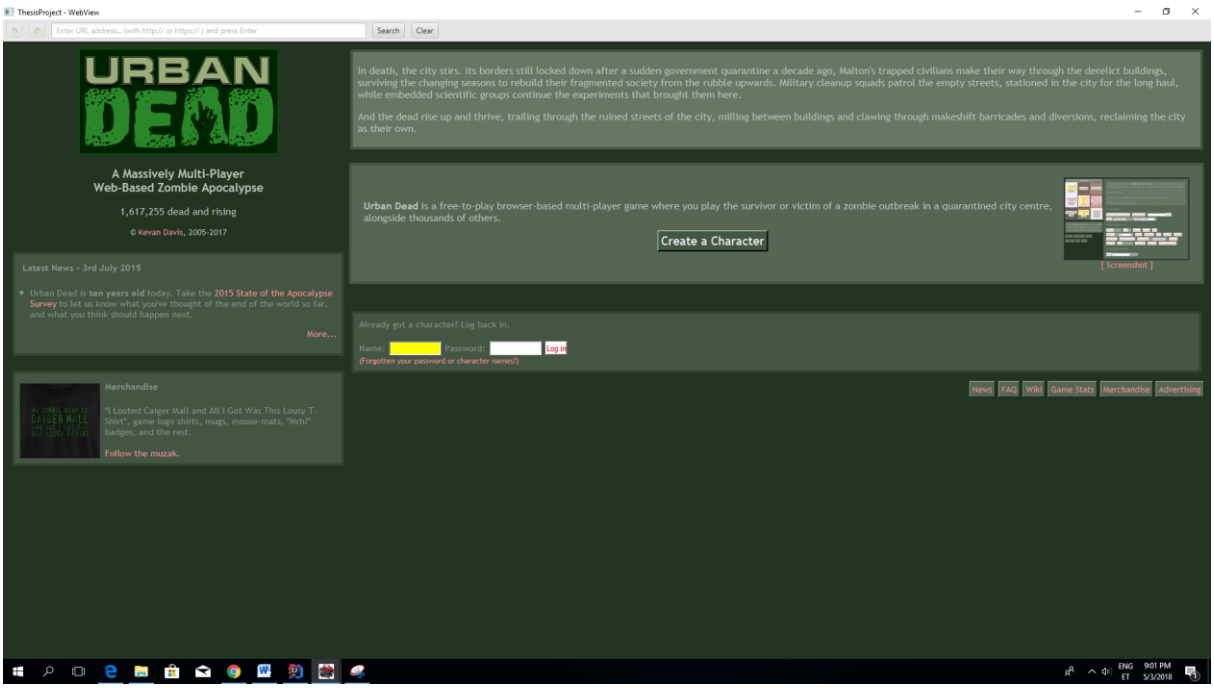
JSON on andmevahetusformaat, mis on sarnane javascripti alamhulkadele. Ta on sõltumatu programmeerimiskeeltest ning tema struktuur põhineb nime/väärtus paaridel [15]. See on laialdaselt kasutusel ning ta on alternatiiv *Soap*-le, mis on ka andmevahetusformaat. JSON-i kasutatakse tihti näiteks AJAX-i põhistes rakendustes.

5. Rakenduse ülesehitus

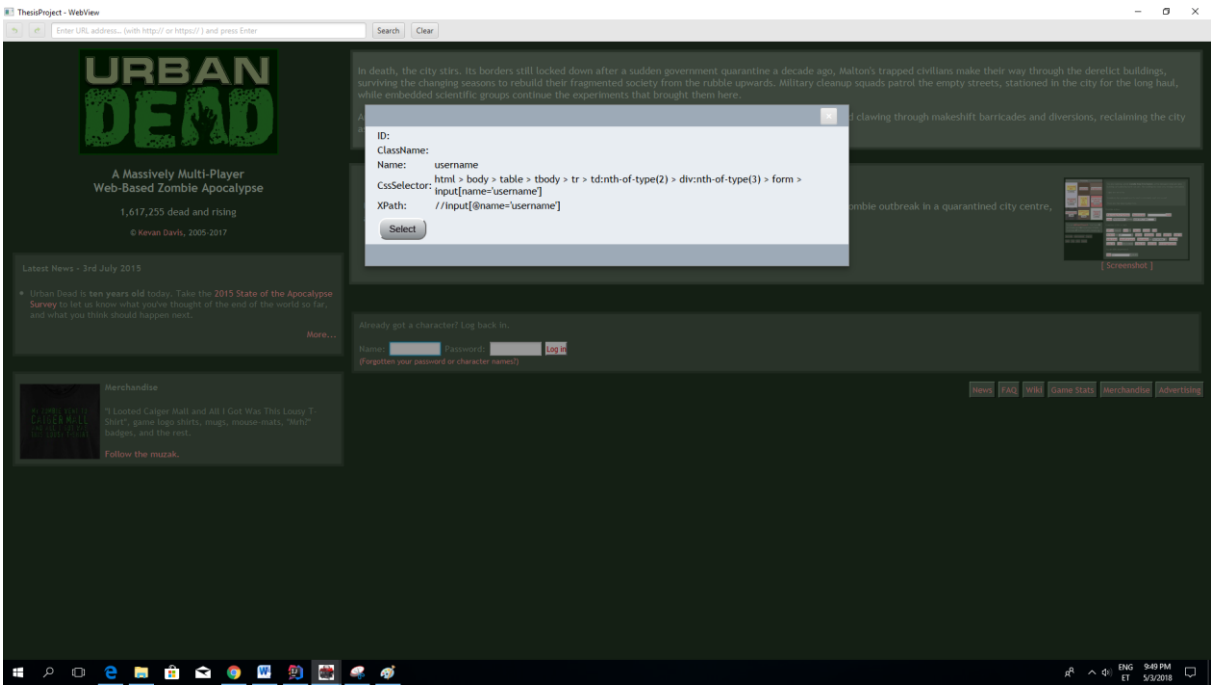
Antud magistritöö tehniliseks väljundiks on Page Object klassi generaator, et hõlpsustada kasutajaliideste testide kirjutamist veebirakenduste jaoks Selenide raamistikus. Rakendus on kirjutatud Java keeles ning kasutajaliidese jaoks on kasutatud JavaFX API-t [9]. Nagu eelnevalt mainitud, siis rakendus koosneb kliendi poolsest vaatest (töölaua vaade) ning veebivaade (vt 3.1). Käesolevas peatükis uuritakse neid kahte lähemalt.

5.1 Veebivaade

Antud komponent on rakenduse alustala. Selle vaate abil saab kasutaja soovitud HTML elemendi kohta genereerida lokaatorid ning neid kasutada oma testides. Enne veebivaate kasutamist tuleb kasutajal avada töölaua vaate kaudu veebivaade ise, mille tulemusel süstitakse soovitud veebilehe *head* tagi sisse vajalik javascript. Seda tehakse selleks, et kasutajal oleks võimalik valida soovitud veebilehe pealt HTML elemente, liigutades hiirekursori soovitud elemendi peale. Selle tulemusel tehakse elemendi tagataust kollaseks ning sellele peale vajutades (Vasak CTRL + vasak hiireklikk) koostatakse (kui võimalik) lokaatorid *id, className*, *name,* cssSelector ja xPath (vt joonis 8), mida kuvatakse eraldi modaalis (vt joonis 9). Ettekuvatav modaal on lihtsalt lokaatorite osas infoks kasutajale, et millised lokaatorid on valitud elemendi kohta võimalikud, kuid et edasi kasutada tuleb kasutajal valida *Select*, mille peale koostatakse HTML elemendi kohta JSON objekt [15] (vt joonis 10), mis sisaldab lokaatorite väärtuseid ning *tagName* ja *tagType,* mis on vajalik info genereerimise jaoks ning see saadetakse töölaua vaatesse edasi nagu mainitud jaotises 3.1.



Joonis 8. Hiirekursoriga valitud element on toodud esile kollase taustaga



Joonis 9. Lokaatorite modaal LCTRL + vasak hiireklikk vajutusel

```
{
   "id":"",
   "className":"",
   "name":"username",
   "selector":"html > body > table > tbody > tr > td:nth-of-
type(2) > div:nth-of-type(3) > form > input[name='username']",
   "xpath":"//input[@name='username']",
   "tagName":"INPUT",
   "tagType":"text"
}
```


Joonis 10. Näidis JSON objekt

5.1.1 Lokaatorite genereerimine

Generaator tekitab valitud HTML elemendi kohta 5 erinvat lokaatorit (alati ei pruugi kõiki olla):

- *id*
- *className*
- *name*
- *cssSelector*
- *xPath*

Kuna veebivaates on HTML elementide valik javascript *event* [17] põhine, siis saab kasutaja poolt valitud HTML elemendi põhiatribuudid (*id*, *className* ja *name*) kätte kutsudes välja *event.target.id*, *event.target.className* ja *event.target.name* kuna *event.target* tagastab DOM objekti. Küll aga pole sellist sisse-ehitatud funktsionaalsust xPath ja cssSelector kohta. Kuigi eksisteerib juba olemasolevaid lahendusi, mis genereerivad eelnevalt mainitud lokaatorid, siis käesoleva töö autor otsustas, et loob ise vajalikud javascript koodid, mis genereeriksid xPathi ja cssSelectori. Peamiseks põhjuseks on see, et enamasti on need mitmetuhandelise rea pikkused javascript koodid ning nagu mainitud töö jaotises (vt 5.1), et veebivaates lehe

laadimisel süstitakse *head* tag sisse javascript, siis mitmetuhandelise rea pikkune kood võib aeglustada rakenduse jõudlust (arvestades seda, kui näiteks kasutaja navigeerib tihti erinevate veebilehtede vahel, siis igakord süstitakse javascript uuesti). Selectori genereerimise põhimõtteks võeti idee, et kontrollitakse kas elemendil, mille kohta selectorit soovitakse eksisteerib *id* attribuut, mis on üle DOM unikaalne. Kui jah, siis kasutatakse *id* attribuuti selectorina. Kui *id* sisaldab punkt märki (nt id='customer.totalPrice'), tuleb see asendada kuueteistkümnendsüsteem väärtusega või lihtsalt määrata ette kaks \ märki, muidu Selenium/Selenide testid ei saa aru antud selectorist [23]. Punkt märk kuueteistkümnendsüsteemis on 2E [18]. Unikaalse *id* puudumisl vaadatakse mis element on *parent* element. Seejärel päritakse *parent* elemendi kõik tema *children* elemendid ning kui nende seas on veel sama tüüpi HTML elemente, kui see, mille kohta selectorit genereeritakse, siis hakatakse indekseerima (st käiakse läbi kõik *children* elemendid ning pannakse korrektne indeks külge elemendile, mille kohta selector genereeritakse). Kui selgub, et elemendil on ka *name* atribuut, siis indeksi asemel kasutatakse seda hoopis. Lõpuks salvestatakse maha soovitud HTML elemendi *parent* element ja peaelement ise maha massiivi ning eelnevalt mainitud sama tegevus jätkub *parent* elemendiga. Ehk kokkuvõtvalt valitakse mingi element, millele soovitakse saada *cssSelector*. Seejärel hakatakse sellest elemendist mööda DOM mudelit järjest üles poole minema, kuni lõpptingimuseks on see, et jõutakse ülesliikumisel mingi elemendini, millel on unikaalne *id* või siis jõutakse dokumendi *body* tag juurde. Kõik elemendid, mida töödeldi on maha salvestatud eraldi massiivi ning selle massiivi elemendid ühendatakse > märgiga (vt joonis 11), mis on ühtlasi ka *cssSelector* valitud HTML elemendi jaoks.

```
FUNCTION SELECTOR_GENERATOR(element) {

    VAR selectorParts = [ ]
    VAR parentElementChildren = element.parent.children
    VAR countSameTagsUnderParentElement
    VAR elementTagIndexer = 1
    VAR elementOccuranceCounter = 0

    WHILE element has parent node DO :

        IF element has id AND is unique THEN :
            VAR elementId = element
            IF elementId contains dot (.) THEN :
                replace elementId dot with HEX value
            END IF
            add elementId to selectorParts with # suffix
            BREAK
        END IF

        FOR i = 1 TO parentElementChildren length :
            IF parentElementChildren[i] tagname == element tagname THEN :
                elementOccuranceCounter ++
            END IF
        END FOR

        FOR j=0 TO parentElementChildren length :
            IF element == parentElementChildren[j] THEN :
                IF element has name attribute THEN :
                    add element tagname with name attribute to selectorParts
                ELSE :
                    IF countSameTagsUnderParentElement == 1 THEN :
                        add element tagname to selectorParts
                    ELSE :
                        add element tagname with nth-of-type(elementTagIndexer) attribute to selectorParts
                    END IF
                END IF
                BREAK
            ELSE :
                IF element tagname == parentElementChildren[i] tagname AND parentElementChildren[i] == 1 THEN :
                    elementTagIndexer++
                END IF
            END IF
        END FOR
        replace element with current element parentNode
    END WHILE
    RETURN selectorParts members with > between them
```

Joonis 11. Selectori genereerimise pseudokood

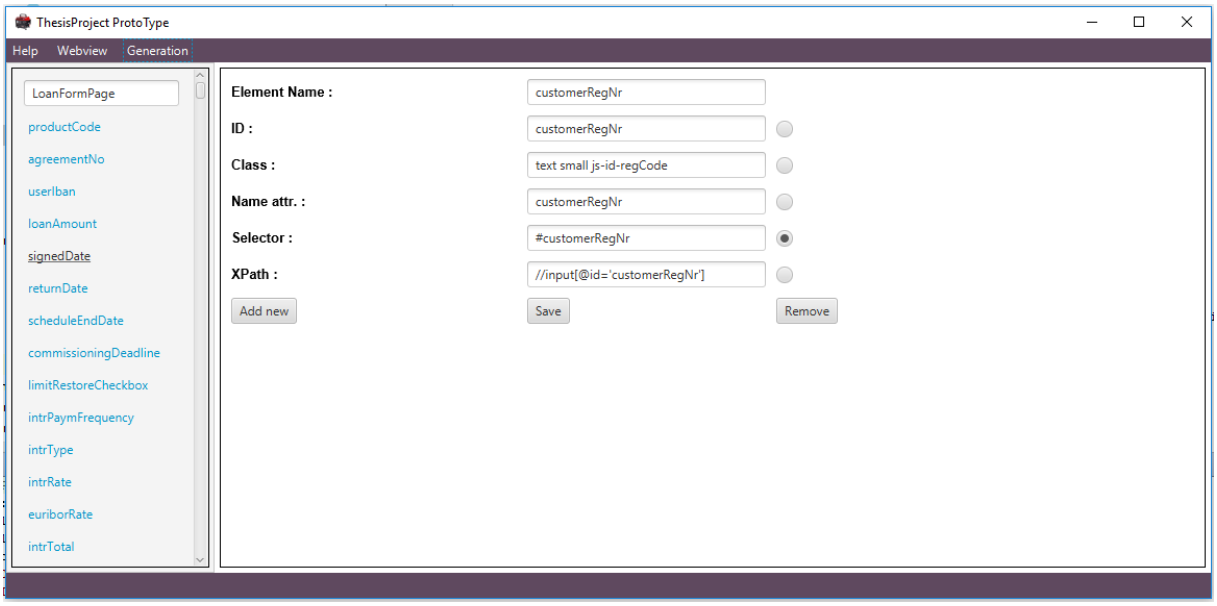
Xpath genereerimine on oma sisult põhimõtteliselt sama (st samuti hakatakse peaelemendist mööda dokumenti ülespoole liikuma, kuni jõutakse elemendini millel on unikaalne *id* või unikaalne *name* või siis *body* element), ainult et kui selectoris kasutati > märki, et kokku panna selector, siis xpathis kasutatakse / märki. Ning sama asi on lahendatult rekursiivselt.

5.2 Töölaua vaade

Töölaua vaates on kasutajal ülevaade veebivaatest valitud HTML elementidest ning nendega seonduvatest lokaatoritest. Siin on tal võimalik määrata elementide nimetused ning vajadusel, kui mingi lokaator ei sobi mingi elemendi jaoks (nt *id* pole unikaalne, siis saab ta vahetada mingi muu lokaatori vastu). Samuti saab ta genereerida valmis Page Object lähtekoodi siin ning reaalselt maha salvestada see .java failina.

5.2.1 Elemendi detailkuva

Antud vaates saab kasutaja näha täpsemalt, et mis elemendid tal on valitud ning soovi korral kas lisada juurde või hoopis muuta/kustutada neid. Enne elementide lisamist tuleb kasutajal avada veebivaade, mis on elementide koostamise aluseks Page Object klassile. Muidugi on võimalik ka kasutajal lisada neid elemente ilma veebivaateta, aga siis kaob üldse rakenduse kasutatavuse mõte ära. Pärast veebivaatest elemendi valimist, saadetakse andmed elemendi detailkuvale ning koostatakse elemendi mudel, kuhu talletatakse maha elemendi lokaatorite väärtused, mis saadeti veebivaatest (samuti hoitakse järge sellel, et millise lokaatori kasutaja elemendile määras) ning *tagType* ja *tagName* (vt 5.1). Elemendi salvestamisel võetakse kasutajapoolt defineeritud elemendi nimi ning see pannakse koos loodud elemendimudeliga paisktabeli [24] andmestruktuuri, kus võtmeks on defineeritud elemendi nimi ning väärtuseks on koostatud elemendimudel, mis siis hoidis lokaatorite väärtusi ja *tagType* ja *tagName* välju. Mainitud paisktabeli abil hoitakse järge ka kasutaja teiste tegevuste üle. Näiteks, kui kasutaja muudab mingit infot, siis elemendinime järgi saadakse aru, mis elemendil muutus toimus, kuna hetkel on elemendinimi, kui paisktabeli võtmeveerg. Ning kui kasutaja otsustab eemaldada elementi, siis lihtsalt samamoodi elemendinime järgi eemaldatakse elemendimudel paisktabelist ja vastavad uuendused tehakse ka kasutajaliidese peale (samamoodi ka elemendi info uuendamisel). Joonis 12 illustreerib detailkuva vaadet. Keskel on elemendi info (elemendi nimi ja tema lokaatorite väärtused ning milline lokaator on sealt kasutaja poolt määratud elemendi jaoks). Vasakul on kõige esimene sisestuslahter koht, kus kasutaja määrab loodava Page Object klassi jaoks nime ning tema all on nimekiri koostatud Page Object elementidest.



Joonis 12. Töölauavaate detailkuva

5.2.2 Page Object klassi genereerimiskuva

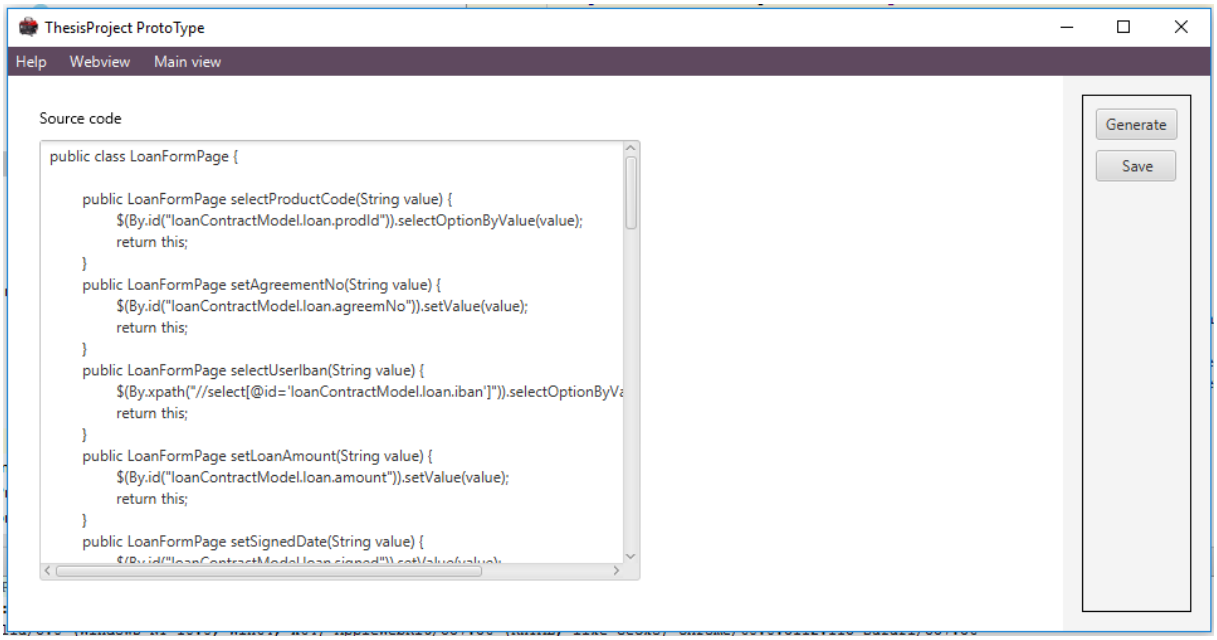
Genereerimisvaates saab kasutaja genereerida kokku java Page Object klassi oma valitud HTML elementidest. Enne kõvakettale maha salvestamist tuleb kasutajal vajutada *Generate* nuppu, mille peale moodustatakse Page Object klassi kood. Genereerimise aluseks võetakse paisktabelisse salvestatud elementide mudelite (vt 5.2.1) *tagType* ja *tagName*. Nende abil pannakse paika teenusmeetodid, mis oli üks arendatava vahendi nõuetest (vt 2.4.3). Teenusmeetodeid on määratud neli:

- *click* meetod (elemendi peale vajutamine) – määratakse järgnevatel juhtudel:
  - *tagName* on *input* ja *tagType* on kas *button*, *checkbox*, *radio*, *submit* või *reset*
  - *tagName* on *a* (HTML link tag)
  - *tagName* on *button* ja *tagType* on *button*, *reset*, *submit*
- *set* meetod (elemendi väärtustamine) – määratakse järgnevatel juhtudel:
  - *tagName* on *input* ja *tagType* on kas *text*, *password*, *search*, *tel* või *url*
  - *tagName* on *textarea*

- *select* meetod (rippmenüüst valimine) – määratakse järgnevatel juhtudel:
    - *tagName* on *select*
- *get* meetod (elemendi objekti tagastamine)– määratakse kõikidel muudel juhtudel, mis ei täitnud eelnevalt mainitud tingimusi

Kuigi neid kombinatsioone võib veel olla, siis töö autor otsustas, et kõikidel muudel juhtudel tagastatakse elemendi objekt ise, et siis saaks vajadusel kasutaja ise kutsuda vastavad elemendi *click* meetodid välja või hoopis kui tegu on *text* tüüpi elemendiga, siis *getText* meetodi väljakutse. Et idee on see, et kui kasutajal peaks olema parasjagu selline elemendi *tagType* ja *tagName* kombinatsioon, mida loodavas rakenduses toetatud pole, et siis kasutaja ei jää ikkagi hätta, vaid ta saab selle elemendi objekti kätte, mille kaudu ta saab vajalikke meetodeid ise välja kutsuda.

Kuna elemendimudelis hoiti järge ka sellel, et mis lokaatorit element peaks kasutama, siis selle järgi pannakse ka genereeritavasse koodi vastav lokaatori tüüp. Ja genereeritava klassi nimi võetakse ka sellest, mis kasutaja detailkuval määras. Joonis 13 on kujutatud genereeritud koodi.



Joonis 13. Genereerimiskuva vaade

6. Lahenduse testimine

Valmislahenduse kontrollimiseks võetakse testitavaks veebileheks samuti NETI otsingumootori pealeht (http://www.neti.ee/) ning sooritatakse sama kasutuslugu nagu seda tehti olemasolevate lahenduste uurimiseks (vt 2.4). Et loodud rakenduse testi lugeda õnnestunuks, tuleb testiga ära täita järgmised kriteeriumid:

- Testis tuleb kasutada vähemalt kahte HTML elementi nii, et oleks kasutatud ühe elemendi puhul genereeritud cssSelectorit ning teise puhul xPathi. Seda siis seetõttu, et saaks valideerda, et genereeritud xPath ja cssSelector oleksid õiged (st sellisel juhul peab test genereeritud lokaatorite abil elemendid kätte saama)

- Testi koostamine koos loodud vahendiga peaks andma ajalise võidu. See tähendab vähem aega peaks kuluma manuaalselt Page Object klassi loomisele

- Test peab olema roheliseks märgitud (st test peab olema staatuses *passed*)

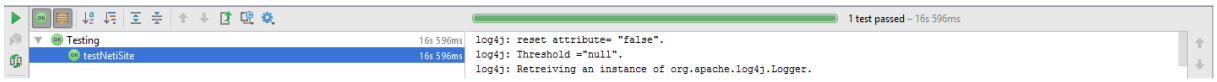
Kuna ajalise võidu määramiseks autor muud võimalust ei näinud, kui lihtsalt aega mõõta mõlema lähenemise puhul, siis lõpptulemusest selgus, et manuaalselt läks töö autoril aega umbes 18 minutit ning koos vahendiga umbes 8 minutit. Sellest võib järeldada, et kiiremini sai test valmis koostatud küll, kuid tasub silmas pidada, et manuaalselt testi kirjutamise kiirus sõltub kasutajast ikkagi endast (kogemused programmeerimises ja automaattestimises), siis seetõttu on ajaline võit töö autori poolt objektiivne hinnang. Kuna Page Object klassi koostamisel selgus, et need elemendid, mida antud kasutusloosse oli vaja kaasata, sisaldasid ainult *class* attribuuti ning mitmetel olid need samad, siis kõikidele elementidele määrati lokaatoriks ka xPath või cssSelector. See ühtlasti täitis ära ka testi õnnestumise kriteeriumite esimese punkti (et tuleb vähemalt ühe korra kasutada nii xPathi kui ka cssSelectorit). Ning lõpuks saavutati testi *passed* (läbinud) staatus, mis tähendab samuti ka seda, et rakenduse poolt genereeritud xPath ja cssSelector lokaatorid on ka korrektsed. Seetõttu võib lugeda loodud rakenduse testi õnnestunuks. Joonis 12, 13 peal on näidatud ära Page Object klass ja testi kood. Joonis 14 kujutab loodud testi õnnestumist.

```
import com.codeborne.selenide.SelenideElement;
import org.openqa.selenium.By;

import static com.codeborne.selenide.Selenide.$;
public class NetiSearchPage {

    public SelenideElement getKeyBoard() {
        return $(By.xpath("//form[@id='search-bar']/div[2]/div[1]"));
    }
    public NetiSearchPage clickPLetterButton() {
        $(By.cssSelector("#keyboard-content > div:nth-of-type(2) > ul:nth-
of-type(1) > li:nth-of-type(28) > a")).click();
        return this;
    }
    public NetiSearchPage clickOLetterButton() {
        $(By.xpath("//div[@id='keyboard-
content']/div[2]/ul[1]/li[27]/a")).click();
        return this;
    }
    public NetiSearchPage clickSLetterButton() {
        $(By.cssSelector("#keyboard-content > div:nth-of-type(2) > ul:nth-
of-type(1) > li:nth-of-type(34) > a")).click();
        return this;
    }
    public NetiSearchPage clickTLetterButton() {
        $(By.xpath("//div[@id='keyboard-
content']/div[2]/ul[1]/li[23]/a")).click();
        return this;
    }
    public NetiSearchPage clickILetterButton() {
        $(By.xpath("//div[@id='keyboard-
content']/div[2]/ul[1]/li[26]/a")).click();
        return this;
    }
    public NetiSearchPage clickMLetterButton() {
        $(By.xpath("//div[@id='keyboard-
content']/div[2]/ul[1]/li[52]/a")).click();
        return this;
    }
    public NetiSearchPage clickELetterButton() {
        $(By.cssSelector("#keyboard-content > div:nth-of-type(2) > ul:nth-
of-type(1) > li:nth-of-type(21) > a")).click();
        return this;
    }
    public NetiSearchPage clickSearchButton() {
        $(By.xpath("//form[@id='search-bar']/div[4]/input")).click();
        return this;
    }
    public NetiSearchPage clickFirstResult() {
        $(By.cssSelector("#main-content > div:nth-of-type(1) > ul:nth-of-
type(2) > li:nth-of-type(1) > h3 > a")).click();
        return this;
    }
}
```

Joonis 14. Genereeritud Page Object kood testi jaoks

```
import com.codeborne.selenide.WebDriverRunner;
import org.junit.Test;

import static com.codeborne.selenide.Selenide.open;
import static org.junit.Assert.assertEquals;

public class Testing {

    private void initializeProperties() {
        System.setProperty("webdriver.chrome.driver",
"C:\\Tools\\chromeDriver\\chromedriver.exe");
        System.setProperty("selenide.browser", "Chrome");
    }

    @Test
    public void testNetiSite() {
        initializeProperties();
        open("http://www.neti.ee/");
        NetiSearchPage page = new NetiSearchPage();
        page.getKeyBoard().click();
        page.clickPLetterButton()
                .clickOLetterButton()
                .clickSLetterButton()
                .clickTLetterButton()
                .clickILetterButton()
                .clickMLetterButton()
                .clickELetterButton()
                .clickELetterButton()
                .clickSLetterButton()
                .clickSearchButton()
                .clickFirstResult();
        assertEquals("https://rus.postimees.ee/", WebDriverRunner.url());
    }
}
```

Joonis 15. Automaattest koostatud neti.ee kasutusloo testimiseks



Joonis 16. Õnnestunud kasutajaliidese automaattest

7. Tuleviku arendused

Loodud süsteemi funktsionaalsus on autori hinnangul piisav, et seda saaks kasutama hakata, kuid siiski on kohti, mida saaks süsteemis täiustada. Järgnevalt on esitletud nimekiri täiendustest, mida võiks lisada süsteemile:

- Kuna praegu on toetatud ainult Selenide raamistik, siis tegelikult võiks ka toetatud olla Selenium ise, mille põhjal Selenide ise töötab.
- Veebivaates võiks kasutajal olla ka võimalus *frame* sees olevaid elemente valida. Praegu *frame* ja *iframe* sees olev sisu pole toetatud.
- Praegu on toetatud ainult Java programmeerimiskeel. Võiks aga juurde lisada nt veel mõned tuntumad keeled nagu Python või C#. Küll aga Selenide raamistik töötab ainult Java peal, mistõttu peaks Python ja C# keele laiendused toetama ainult Seleniumit.
- Praegu veebivaates on kasutajal võimalik sisestada sinna URL, kust ta soovib saada elemente. Küll aga on probleem seal see, et kui kasutaja sisestab URL-i, mis on *https://* protokolliga (nt www.google.com) ning kui ta protokolli ette ei lisa, siis ei tunta aadressit ära. Rakenduses on kasutuses JavaFX *WebView* komponent ning seal ei ole ära realiseeritud seda käitumist nagu tavalises brauseris, et kui sisestatakse ilma protokollita aadress, et siis suudetakse aru saada, et mis protokolli sisestatud URL kasutab.
- Samuti võiks kasutajal võimalus olla maha salvestada pooleliolev töö. Sest võibolla kasutaja soovib pärast pausi edasi töötada pooleli olevast tööst.
- Kasulik täiendus oleks see, kui kasutaja näeb veebipoolsel kuval logisid. Sest et kui peaks juhtuma, et mingi javascript viga nt esineb, siis oleks kasutajale arudaadav, et mis probleem on. Ilma selleta, on raske selgusele jõuda, kui midagi näiteks ei peaks toimima.

- Hetkel on meetodite genereerimisel tehtud nii, et kui näiteks on mingi vormi välja väärtustamine, et siis genereeritakse meetod nii, et parameetriks söödetakse String tüüpi muutuja. Võiks aga täienduse teha, et kasutaja saab ise defineerida, et mis tüüpi muutujat ette antakse parameetrina nendele meetoditele, kus väärtustatakse mingit välja.

- Lisaks *id, className, name, selector* ja *xPath* lokaatoritele võiks lisada ka *linkText* järgi otsimise.

- Väiksemat laadi täiendus oleks ka töölaua poolsel vaatel koodi genereerimise kuva täiendus. Täpsemalt võiks olla seal ka süntaksi esiletõstmine (st klassid, meetodinimed ja muutujad võiksid erivärvi olla)

8. Kokkuvõte

Antud töö eesmärgiks oli luua Page Object klasside generaator, mis toetab Selenide raamistikku. Loodud rakendus toetab *id*, *className*, *name*, *cssSelector* ja *xPath* tüüpi lokaatoreid. Rakendusse on sisse ehitatud veebibrauser, mille kaudu kasutaja saab valida endale sobivat HTML elementi, mida ta soovib kasutajaliidese testis kasutada.

Töö käigus uuriti osasid olemasolevaid lahendusi ning töö autor paraku ei leidnud ühtegi sellist, mis genereeriks Selenide raamistikus Page Object klasse. Eksisteerivaid lahendusi analüüsiti *neti.ee* veebilehe peal ning saadi ülevaade, et mis omadused ja puudused nendel on. Analüüsi tulemuste peale moodustati kokku uued nõuded ning neid rakendati loodavas vahendis.

Töö tulemusena valmis arvutiprogramm, mille valmistamiseks kasutati Java programmeerimiskeelt ning samuti ka Javascripti. Kasutajaliideste jaoks kasutati JavaFX liidest, kuna ta sisaldab sisseehitatud brauserit. Selle eesmärk oli see, et ei peaks eraldi alla laadima WebDriver-t, mis SWD Page Recorder [5] puhul on vajalik, et saaks brauserit kasutada. Javascripti kasutati selleks, et võimaldada kasutajal brauseris valida HTML elemente.

Lõpuks koostati samalaadne test, mida tehti olemasolevate lahenduste analüüsis. Ainult, et seekord kasutati loodud vahendit selleks. Testi käigus saadi ajaline võit, kui võrrelda sellega, et manuaalselt testi kirjutada. Samuti valideeriti *xPath* ja *cssSelector* lokaatorite väärtused. Sellest tulenevalt võib väita, et test oli edukas ning rakendus rahuldab paika pandud nõuded (vt 2.4.3)

Loodud süsteem on esmane prototüüp ning seetõttu realiseeriti baasfunktsionaalsused, et tagada esmane kasutatavus. Loetleti ka üles funktsionaalsused, mis on plaanis tulevikus juurde arendada, et vahend oleks rohkem kasutatavam.

Töö tulemus on avaldatud githubis [25] https://github.com/Salamander75/ThesisProject

Kasutatud kirjandus

[1] – Selenium WebDriver. [WWW] http://www.seleniumhq.org/. (19.02.2018).

[2] – Selenide: concise UI tests in Java. [WWW] http://selenide.org/. (19.02.2018).

[3] – Codeborne - Hästi tehtud tarkvara. [WWW] https://codeborne.com/et/. (19.02.2018).

[4] – Selenium Page Object Generator – Chrome Web Store. [WWW] https://chrome.google.com/webstore/detail/selenium-page-object-gene/epgmnmcjdhapiojbohkkemlfkegmbebb. (11.03.2018).

[5] – dzharii/swd-recorder. [WWW] https://github.com/dzharii/swd-recorder/releases. (11.03.2018).

[6] – Java (programming language) – Wikipedia. [WWW] https://en.wikipedia.org/wiki/Java_(programming_language). (11.03.2018).

[7] – PageObject. [WWW] https://martinfowler.com/bliki/PageObject.html. (12.03.2018).

[8] – Java Development Kit. [WWW] https://en.wikipedia.org/wiki/Java_Development_Kit. (12.03.2018) .

[9] – What is JavaFX? [WWW] https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm. (12.03.2018).

[10] – XML. [WWW] https://en.wikipedia.org/wiki/XML. (12.03.2018).

[11] – JavaScript. [WWW] https://en.wikipedia.org/wiki/JavaScript. (12.03.2018).

[12] – What is CSS. [WWW] https://www.tutorialspoint.com/css/what_is_css.htm. (12.03.2018).

[13] – Maven. [WWW] https://maven.apache.org/. (12.03.2018).

[14] – Apache Maven. [WWW] https://en.wikipedia.org/wiki/Apache_Maven. (12.03.2018).

[15] – JSON. [WWW] https://www.json.org/. (04.05.2018)

[16] – Ajax. [WWW] https://en.wikipedia.org/wiki/Ajax_(programming). (04.05.2018).

[17] – Introduction to Events – Learn Web Development | MDN. [WWW] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events. (04.05.2018).

[18] – HTML Codes – Table of ascii characters and symbols. [WWW] https://www.ascii.cl/htmlcodes.htm. (04.05.2018).

[19] – The MIT License | Open Source Initiative. [WWW] https://opensource.org/licenses/MIT. (04.05.2018).

[20] – Selenium Tutorial: Locators. [WWW] https://www.protechtraining.com/content/selenium_tutorial-locators. (04.05.2018).

[21] – Swing (Java) – Wikipedia. [WWW] https://en.wikipedia.org/wiki/Swing_(Java). (04.05.2018).

[22] – What is the Document Object Model? [WWW] https://www.w3.org/TR/WD-DOM/introduction.html. (04.05.2018).

[23] – webdriver – How do you select elements in selenium that have a period in theird id? – Stack Overdlow. [WWW] https://stackoverflow.com/questions/16107448/how-do-you-select-elements-in-selenium-that-have-a-period-in-their-id. (04.05.2018).

[24] – Paisktabel – Vikipeedia, vaba entsüklopeedia. [WWW] https://et.wikipedia.org/wiki/Paisktabel. (04.05.2018).

[25] – GitHub. [WWW] https://github.com/. (04.05.2018).

Lisa 1 – Ekraanipilt Google Chrome Page Object Generator koostatud koodist

Järgnev koodilõik koostati Google Page Object Chrome Generator poolt vastu https://www.neti.ee. Ekraanipildist on näha, et osad automaatselt genereeritud meetodite nimed on väga pikad ja arusaamatud

```
public NetiTest
setKpsisedAitavadMeilTeenuseidEdastada6DropDownListField(String
kpsisedAitavadMeilTeenuseidEdastada6Value) {
    new
Select(kpsisedAitavadMeilTeenuseidEdastada6).selectByVisibleText(kpsisedAit
avadMeilTeenuseidEdastada6Value);
    return this;
}

/**
 * Set default value to Otsi Aadressi Otsi Kasuta Arvuti Asukohta Kustuta
Text field.
 *
 * @return the NetiTest class instance.
 */
public NetiTest setOtsiAadressiOtsiKasutaArvutiAsukohta1TextField() {
    return
setOtsiAadressiOtsiKasutaArvutiAsukohta1TextField(data.get("OTSI_AADRESSI_O
TSI_KASUTA_ARVUTI_ASUKOHTA_1"));
}

/**
 * Set value to Otsi Aadressi Otsi Kasuta Arvuti Asukohta Kustuta Text
field.
 *
 * @return the NetiTest class instance.
 */
public NetiTest setOtsiAadressiOtsiKasutaArvutiAsukohta1TextField(String
otsiAadressiOtsiKasutaArvutiAsukohta1Value) {
    new
Select(otsiAadressiOtsiKasutaArvutiAsukohta1).selectByVisibleText(otsiAadre
ssiOtsiKasutaArvutiAsukohta1Value);
    return this;
}
```