TARTU ÜLIKOOL

MATEMAATIKA-INFORMAATIKATEADUSKOND

Arvutiteaduse instituut

Infotehnoloogia eriala

Elo Eilonen

Veebirakenduse automatiseeritud testide loomine raamistikuga Selenium

Bakalaureusetöö (6 EAP)

Juhendaja: dots Helle Hein

TARTU 2015

Veebirakenduse automatiseeritud testide loomine raamistikuga Selenium

Lühikokkuvõte:

Antud töö eesmärgiks on tutvustada ühte võimalust, kuidas koostada veebirakenduse funktsionaalsuste testimiseks automatiseeritud teste. Selleks kasutatakse veebilehitsejate automatiseerimise raamistikku Selenium ja seda raamistikku teotavaid mudeleid, liideseid ja teeke, millest antakse töös ülevaade. Töö praktilises osas kasutatakse teoreetilises osas kirjeldatud tehnoloogiaid reaalset veebirakendust testiva programmi loomisel.

Võtmesõnad:

Veebirakenduste testimine, automatiseeritud testid, Selenium, leheobjektide mudel, andmetest juhinduv testimine

Creating Automated Tests for a Web Application with Selenium Framework

Abstract:

The goal of the present thesis is to introduce a way to create automated tests for testing web application functionalities. It is done using web-browser automation framework Selenium and other frameworks, APIs and libraries that support Selenium automation. In the practical part of this thesis a program to test an actual web application is created using the technologies introduced in the theoretical part of the thesis.

Keywords:

Testing web applications, automated tests, Selenium, Page Object Model, Data Driven testing

Sisukord

1 Sissejuhatus

Tarkvaralahendused täidavad tänapäeva maailmas olulist rolli. Väga paljud toimingud, mida varasemalt on teostatud ainult klienditeenindaja vahendusel, tehakse nüüd ära veebirakendustega. Ettevõtted on mõistnud, et selliselt on võimalik suurendada müügitulu. Kasvab ka klientide rahulolu, kuna nad saavad tarbida ettevõtte teenuseid kodust lahkumata ning potentsiaalselt ajavõiduga. Et eespool kirjeldatu toimuks, peab teenuste tarbimine veebis olema kasutajale mugav ning tõepoolest võimaldama teostada teenuse pakkuja poolt lubatud toiminguid. Kui tarkvaralahendus ei võimalda vajaminevat või lubab üleliia, siis kannatab nii klient kui ettevõte.

Tarkvara testimine on tarkvaraarenduse distsipliin, mille eesmärgiks on aidata teenuse pakkujatel viia oma tarkvaralahendus teenuse pakkumiseks vajalikule kujule. Tarkvara testija ülesandeks on panna ennast vähem või rohkem infotehnoloogiliselt arenenud kasutaja rolli ning selgitada välja etteantud lahenduse käitumine. Leidub olukordi, kus on mõistlik asendada inimressurss arvutusjõuga ning kasutada automatiseeritud testimist. Automatiseeritud testimine on varasemalt kirjeldatud teststsenaariumite läbiviimine testprogrammi poolt. Mõtestatult planeeritud automatiseeritud testid võivad olla suureks abiks testija töö efektiivsemaks muutmisel.

Käesoleva bakalaureusetöö eesmärgiks on koostada testprogramm, mis oleks tarkvara testijale abiks ühe Suurbritannia autokindlustusettevõtte veebirakenduse testimisel. Testprogrammi peamiseks ülesandeks on kontrollida läbi graafilise kasutajaliidese kahes erinevas keskkonnas veebirakenduse kõige ärikriitilisemate funktsionaalsuste töökorda. Töö autor otsustas programmi loomiseks kasutada raamistiku Selenium WebDriver API-liidest programmeerimiskeeles Java koos TestNG raamistikuga. Testandmete haldamiseks kasutas autor Apache POI API-liidest. Programmi väljundi ehk testide tulemuste kirjeldamiseks kasutas autor Apache Log4j API-liidest ning ExtentReports teeki HTML kujul raportite loomiseks. Kõiki nimetatud tehnoloogiaid tutvustatakse käesolevas töös.

Töö eeldab, et tema lugejal on üldine arusaam tarkvaraarendusdistsipliinidest, algteadmised veebirakenduste arendamisest ning võime lugeda Java koodi.

1.1 Töö struktuur

Töö esimeses osas tutvustatakse lähemalt testitavat lahendust, töö praktilises osas loodava testprogrammi eesmärki ja funktsionaalseid nõuded. Samuti kirjeldatakse programmi poolt läbiviidavaid teste.

Töö teine osa keskendub programmi loomiseks kasutatavatele tehnoloogiatele: raamistik Selenium ja programmeerimiskeel Java, Selenium WebDriver API-liides, raamistik TestNG, Apache Log4j API-liides, Apache POI API-liides ja ExtentReports teek. Kirjeldatakse kasutatud disainimustrit (*design pattern*) leheobjektide mudelit (*Page Object model*) ja andmetest juhinduva testimise meetodit (*Data Driven testing*).

Töö kolmas osa näitab, kuidas töö teises osas kirjeldatud tehnoloogiaid kasutati töö praktilise osana loodud testprogrammis.

Töö lisadeks on testprogrammi lähtekood ning muud vajalikud failid. Tuuakse näide testprogrammi väljundite kohta. Kõik lisad on antud veebilinkidena.

1.2 Mõisted

- Töövoog (*workflow*) – tööprotsessi kirjeldus, kus tuuakse välja vaatluse all oleva protsessi eesmärgid, nendeni viivad sammud ning sammude teostajad. Töövoogu saab esitada näiteks diagrammi või joonisena.
- Testandmed (*test data*) – andmed, mida kasutatakse testide läbiviimisel sisendina ja andmed, mida testitav lahendus peab väljastama etteantud sisendandmete korral. Viimast kasutatakse testi tulemuse määramiseks.
- Testlugu (*test case*) – testi kirjeldus, mis sisaldab testi läbiviimiseks ja hindamiseks vajalikku informatsiooni.
- Testikomplekt (*test suite*) – testlugude kogum.
- Regressioonitestimine (*regression testing*) – testimismetoodika, mille eesmärgiks on kontrollida, et lahenduse olemasolev, varasemalt testitud ja töötanud funktsionaalsus töötaks endiselt peale täienduste lisamist ja muudatuste, paranduste tegemist. Tavaliselt teostatakse regressioonitestimist olemasolevate testlugude kordamisega [1].
- Testkeskkond (*test environment*) – veebirakenduse puhul veebikeskkond, mida kasutatakse veebirakenduse testimiseks.
- Toodangu keskkond (*production environment, live environment*) – veebirakenduse puhul veebikeskkond, mida kasutatakse teenuse pakkumiseks klientidele.
- Dünaamiline veebileht (*dynamic website*) – selline veebileht, mille sisu koostatakse reaalajas vastavalt kasutaja tegevusele. Vastandiks on staatiline veebileht (*static website*),

mille sisu on eeldefineeritud ja ei muutu kasutaja tegevuse peale.

- Annotatsioonid (*annotations*) – metaandmed programmi koodis, mis ei ole otseselt programmi koodi osa, kuid jagavad kompileerimis- ja käitusfaasis programmi kohta infot [2].

2 Testitav lahendus ja testprogramm

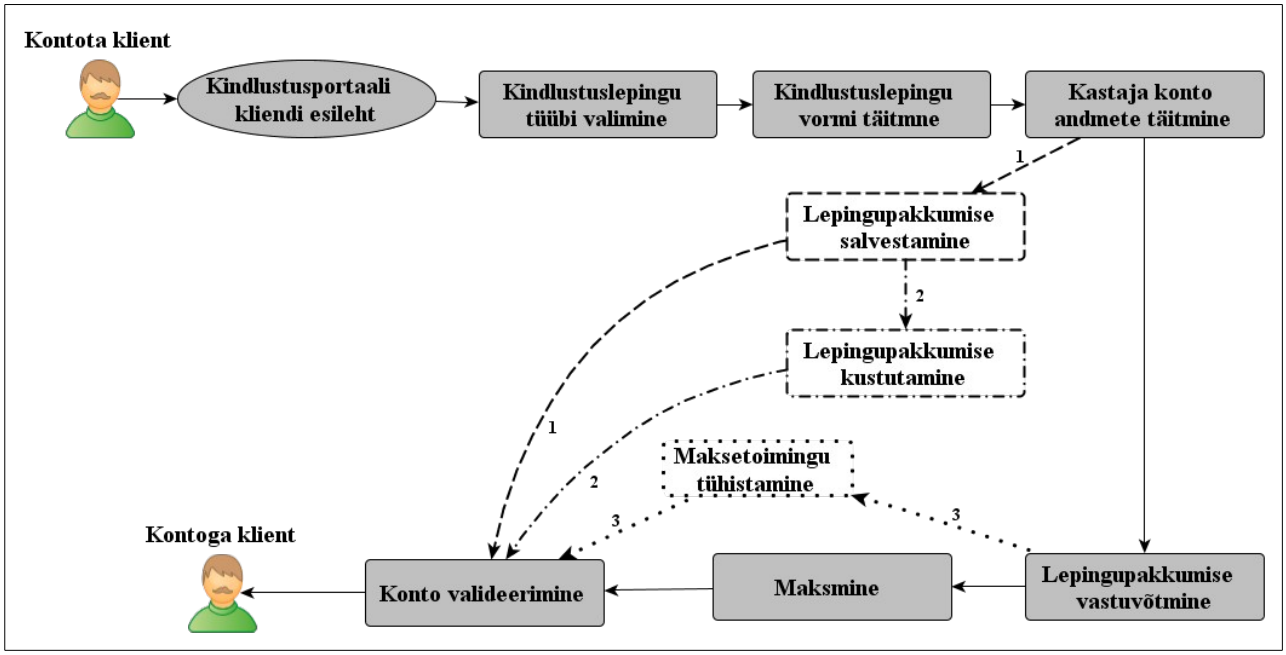
Käesolev peatükk tutvustab lähemalt testitavat veebirakendust. Tuuakse välja testprogrammi funktsionaalsed nõuded ja kirjeldatakse detailsemalt testlugusid, mida testprogramm läbi viib.

2.1 Testitav lahendus

Testitav programm on veebirakendus, mille eesmärgiks on pakkuda autokindlustuse soovijatele võimalust sõlmida kindlustusleping internetis.

Veebirakendus koosneb kahest osast: kliendi portaal ja klienditeenindaja portaal. Antud töö raames tutvustame mõlemat lihtsustatult jättes välja mitmed ärilised nüansid.

Kliendi portaali kasutajaks on kindlustusettevõtte klient. Joonisel 1 on kirjeldatud kliendi portaali olulisim töövoog – konto loomine ja kindlustuslepingu sõlmimine.

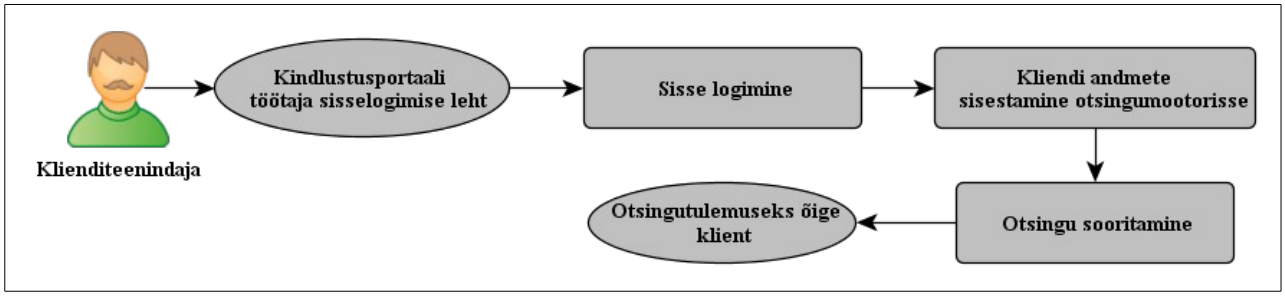


Joonis 1: Kliendi töövoog: konto loomine ja lepingu sõlmimine.

Kindlustuslepingu tüüpe on kaks. Kindlustuslepingu vormi täitmine tähendab kliendi isikuandmete, juhtimisõiguse andmete, kindlustatava auto andmete ja muude kindlustusseadusest ning ärilistest vajadustest tingitud andmete jagamist kindlustusettevõttele. Täidetud vormi alusel koostatakse kliendile lepingupakkumine. Leping loetakse sõlmituks peale makse sooritamist või järelmaksu lepingu sõlmimist. Maksmiseks kasutatakse kolmanda osapoole makseteenust. Konto valideerimine toimub e-postile saadetud valideerimise lingi kaudu. Peale konto valideerimist on kliendil ligipääs oma kontole. Ta saab oma isiku- ja lepingute andmeid näha ning muuta, samuti sõlmida uusi

lepinguid. Konto luuakse ka juhul, kui klient ei sõlmi lepingut kohe: lepingupakkumise salvestamisel (Joonis 1, alternatiivne voog 1), lepingupakkumise kustutamisel (Joonis 1, alternatiivne voog 2) ja alustatud maksetoimingu tühistamisel (Joonis 1, alternatiivne voog 3).

Klienditeenindaja portaalis on kasutajaks kindlustusettevõtte töötaja. Joonisel 2 on kirjeldatud klienditeenindaja portaali olulisim töövoog – kliendi konto otsimine kliendi andmete põhjal.



Joonis 2: Klienditeenindaja töövoog: kliendiotsingu sooritamine.

Kliendi otsingut on võimalik teostada erinevate andmete põhjal: isikuandmed nagu nimi, sünniaasta, aadress jne, kontoga seotud kasutajatunnus (kasutajanimi või e-posti aadress), auto registrimärgi number, kliendi viitenumber süsteemis või kliendi mõne lepingu viitenumber süsteemis. Peale kliendi konto leidmist saab klienditeenindaja teostada kõiki kliendi kontole lubatud toiminguid kliendi nimel.

2.2 Testprogrammi eesmärk

Käesoleva töö raames loodava testprogrammi eesmärgiks on kontrollida, et eelmises alapeatükis toodud töövoogusid saaks läbi viia.

Need töövood on kriitilise tähtsusega funktsionaalsused, mille töökord on ärilisest aspektist oluline:

- ettevõte kaotab raha, kui kliendid ei saa sõlmida kindlustuslepinguid;
- ettevõtte maine võib kannatada, kui klienditeenindajad ei saa kliente aidata.

Nende funktsionaalsuste töökord on oluline ka veebirakenduse testijatele testimisprotsessi läbiviimisel:

- testijad ei saa korralikult testida, kui rakenduse põhifunktsionaalsused ei tööta.

Seetõttu on vajalik nende funktsionaalsuste töökorda pidevalt kontrollida nii testkeskkonnas kui ka toodangu keskkonnas. Osutus, et nende funktsionaalsuste testimine oleks mõttekas automatiseerida järgmistel põhjustel:

- kliendi ja klienditeenindaja töövoogude testimise sammud on igal testi läbiviimisel samad;
- automatiseerides saab lühema ajaga läbi käia suurema testandmete variatsiooniga teste;
- kindlustuslepingu vormi täitmine võtab käsitsi palju aega;
- testijale on pidevalt samade testide läbiviimine tüütu ja nüristav tegevus;
- kuigi automatiseeritud testide loomine ja haldamine on ajamahukas, hoiab see hilisemas töös testija aega kokku ning lubab tal keskenduda muudele vajalikele testimistegevustele.

2.3 Testprogrammi funktsionaalsed nõuded

1. Testprogramm peab veebirakendust testima läbi graafilise kasutajaliidese.
2. Testprogramm peab koosnema kahest osast:
    1. testid testkeskkonnas käivitamiseks, mis teostavad testkeskkonna regressioontestimist;
    2. testid toodangu keskkonnas käivitamiseks, mis jälgivad toodangu keskkonda.
3. Testprogramm peab testandmed sisse lugema MS Exceli dokumendist, mida saab vajadusel muuta.
4. Testprogramm peab looma logifaili ja suunama sinna järgmise informatsiooni:
    1. testloo sammud;
    2. veateated;
    3. testi tulemused.
5. Testprogramm peab testi ebaõnnestumise korral tegema ekraanitõmmise.
6. Testprogramm peab kirjutama testitulemuste koondraporti. Koondraport peab sisaldama ekraanitõmmiseid.

2.4 Testid

Tabel 1: Test - toodangu keskkonna jälgimine.

Testi nimi	Toodangu keskkonna jälgimine.
Testi eesmärk	• Jälgida toodangu keskkonda. • Kontrollida, et: ○ toodangu keskkond on kättesaadav;

	○ kontoga kasutaja saab sisse logida; ○ kontoga kasutaja saab täita lepingu vormi; ○ navigeerimine maksekeskkonda ja tagasi õnnestub.
Testi sammud	1. Navigeerimine toodangu keskkonna esilehele. 2. Sisse logimine. 3. Olemasolevate lepingupakkumiste kustutamine. 4. Uue lepingu vormi täitmine. 5. Maksekeskkonda navigeerimine. 6. Makse katkestamine. 7. Äsja loodud lepingupakkumise kustutamine. 8. Välja logimine.
Õnnestumise kriteerium	Sammude 1-8 täitmine õnnestub vigadeta.

Tabel 2: Test - regressioonitestimine testkeskkonna kliendi portaalis.

Testi nimi	Regressioonitestimine testkeskkonna kliendi portaalis.
Testi eesmärk	• Teostada regressioonitestimist testkeskkonna kliendi portaalis. • Kontrollida, et: ○ testkeskkonna kliendi portaal on kättesaadav; ○ kontota kasutaja saab täita lepingu vormi; ○ kasutaja saab maksta.
Testi sammud	1. Navigeerimine testkeskkonna kliendi portaali esilehele. 2. Uue lepingu vormi täitmine. 3. Maksekeskkonda navigeerimine. 4. Makse teostamine.
Õnnestumise kriteerium	Sammude 1-4 täitmine õnnestub vigadeta.

Tabelis 2 kirjeldatud test realiseeritakse mõlema lepingutüübi jaoks.

Tabel 3: Test - regressioonitestimine testkeskkonna klienditeenindaja portaalis.

Testi nimi	Regressioonitestimine testkeskkonna klienditeenindaja portaalis.
Testi eesmärk	• Teostada regressioonitestimist testkeskkonna klienditeenindaja portaalis. • Kontrollida, et: ○ testkeskkonna klienditeenindaja portaal on kättesaadav; ○ klienditeenindaja saab sisse logida;

	○ klienditeenindaja saab teostada kliendi otsingut.
Testi sammud	1. Navigeerimine testkeskkonna klienditeenindaja portaali esilehele. 2. Sisse logimine. 3. Kliendiotsingu teostamine. 4. Kliendiotsingu tulemuse valideerimine.
Õnnestumise kriteerium	Sammude 1-4 täitmine õnnestub vigadeta.

Tabelis 3 kirjeldatud test realiseeritakse kõigi kliendiotsingu võimaluste jaoks.

3 Tehnoloogiad ja disainimustrid

Käesolevas peatükis tutvustatakse testprogrammi loomiseks kasutatud tehnoloogiaid ning disainimustreid. Antud peatükis mõeldakse kaustaja all Seleniumi testide loojat.

3.1 Raamistik Selenium

Raamistik Selenium on vabavaraline veebilehitsejate kasutuse automatiseerimiseks loodud tööriistade kogumik. Seleniumi peamine kasutusala on veebirakenduste testimine, kuid sellega saab automatiseerida ka erinevaid administratiivseid tegevusi – näiteks vormide täitmine [3]. Järgnevalt tutvustatakse lühidalt kõiki Seleniumi tööriistu.

- Selenium IDE (integreeritud programmeerimiskeskkond) – veebilehitseja Mozilla Firefox lisana loodud tööriist, mis oskab lindistada veebis teostatavaid toiminguid ja konverteerida need Seleniumi käskudeks [4].
- Selenium Remote Control – API-liides, mis võimaldab kasutajal luua Seleniumi teste endale sobivas programmeerimiskeeles ja jooksutada neid mitmes erinevas Javascriptiga veebilehitsejas [5].
- Selenium WebDriver – API-liides, mis on Selenium Remote Control tööriista edasiarendus. See võimaldab juhtida veebilehitsejat endale sobivas programmeerimiskeeles [6]. Kuna käesolev töö kasutab just seda tööriista, siis tutvustatakse Selenium WebDriver API-liidest põhjalikumalt alapeatükis 3.2.
- Selenium Grid – tööriist Seleniumi testide jooksutamiseks paralleelselt mitmes masinas [7].

Tabel 4: Seleniumi veebilehitsejate ja programmeerimiskeelte tugi [8].

Veebilehitsejate tugi	Programmeerimiskeelte tugi
• Google Chrome • Internet Explorer 6, 7, 8, 9, 10 - 32 • Firefox • Safari • Opera	• Java • C# • Ruby • Python • Javascript

• HtmlUnit • phantomjs	

Tabelis 4 on toodud veebilehitsejad, mida Selenium toetab. Teises tulbas on programmeerimiskeeled, milles Seleniumi teste kirjutada saab. Seleniumi kasutajad on loonud mitmeid lisasid, mis suurendavad veebilehitsejate ning programmeerimiskeelte valikut, kuid antud töö raames neid ei käsitleta.

Töö autor kasutab Seleniumi testide loomiseks programmeerimiskeelt Java.

3.2 Selenium WebDriver API-liides

Käesolevas töös kasutatakse testide loomiseks Selenium WebDriver API-liidest. Selenium WebDriver on objekt-orienteeritud disainiga API-liides, mis suhtleb veebilehitsejaga ning teostab seal kasutaja kirjeldatud toiminguid. WebDriver API-liidese eesmärgiks on eelkõige pakkuda tuge tänapäevaste dünaamiliste veebilehtede testimises. Järgnevalt kirjeldatakse Joonisel 3 toodud Java koodinäite varal, kuidas WebDriver töötab ning tutvustatakse selle põhilisemaid käske. Alltoodud koodi eesmärgiks on teostada otsingumoototri Google päring. Test loetakse edukaks, kui peale otsingu sooritamist sisaldab HTML lehe pealkiri märksõnana sisestatud sõnet.

Joonisel 3 toodud näide iseloomustab üldist Seleniumi testjuhtumi koodi struktuuri. Tavaliselt sisaldab Seleniumi testi kood järgmisi samme:

1. veebilehitseja käivitamine;
2. veebilehitsejas soovitud lehele navigeerimine;
3. lehelt vajalike HTML elementide otsimine;
4. leitud HTML elementidega soovitud toimingute teostamine (näiteks teksti sisestamine ja nupu vajutamine);
5. tulemuste kontrollimine (näiteks mingi konkreetse HTML elemendi olemasolu);
6. tulemuste väljastamine kasutajale (näiteks konsooli või logifaili);
7. veebilehitseja sulgemine.

```
public class Example {

      public static void main(String[] args) {

      // Loob uue WebDriver objekti. Käivitab FireFox veebilehitseja.
        WebDriver driver = new FirefoxDriver();

      // Lehele navigeermine
        driver.get("http://www.google.com");

      // Otinguvälja otsimine HMTL elemendi nime järgi.
        WebElement element = driver.findElement(By.name("q"));

      // Otsinguväljale otsingu märksõna sisestamine.
        element.sendKeys("Tartu Ülikool");

      // Otsingu alustamine.
        element.submit();

      // Siin öeldakse jooksvale WebDriver objektile, et see ootaks 10
      // sekundit enne kui otsustab, kas testi oodatud tulemus on saavutatud või
      // mitte.
      //  Kui kümne sekundi jooksul oodatud tulemusust ei saavutata, siis
      //loetakse test ebaõnnestunuks.
        (new WebDriverWait(driver, 10)).until(
            // Määratakse testi oodatud tulemus. Tagastab tõeväärtuse.
            new ExpectedCondition<Boolean>() {
                  public Boolean apply(WebDriver d) {
                  // Kontrollib, kas lehe pealkiri sisaldab otsingu märksõna.
                      return d.getTitle().toLowerCase().startsWith("tartu
                                                              ülikool");
                  }
            }
        );
      // Trükitakse lehe pealkiri välja
        System.out.println("Lehe pealkiri: " + driver.getTitle());

      // Veebilehitseja akna sulgemine. WebDriver objekti kustutamine.
        driver.quit();
    }
}
```

Joonis 3: Selenium WebDriver API-liidese koodinäide [9].

3.3 Disainimuster leheobjektide mudel

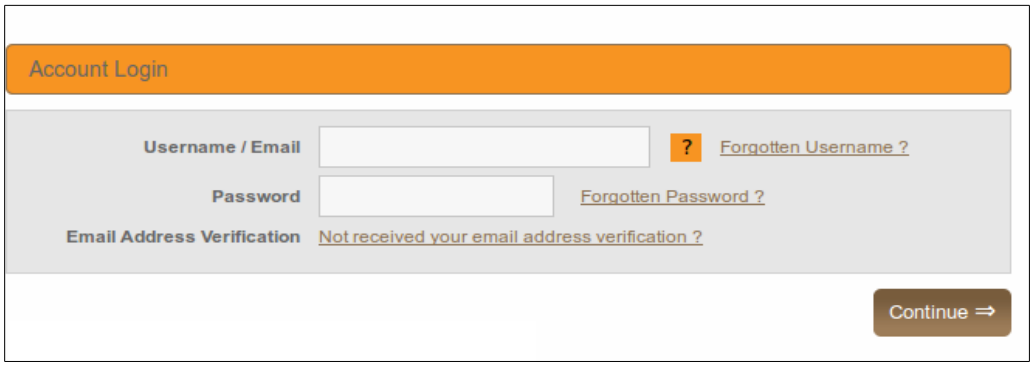
Programmeerimises on levinud tava kasutada oma lahenduse loomisel disainimustreid. Disainimustrid aitavad hoida koodi struktuuri ning vältida koodi kordamist, mis muudab koodi paremini loetavaks ja hallatavaks. Seleniumi testide loomiseks on samuti välja töötatud disainimustreid. Neist hetkel populaarseim on leheobjektide mudel, mida otsustas kasutada ka töö autor.

Leheobjektide mudeli eesmärgiks on hoida eraldiseisvana HTML lehed ja nende testid. Leheobjektiks on objekt-orienteeritud klass, milles otsitakse üles konkreetse lehe tesi jaoks

vajalikud HTML elemendid ning kirjeldatakse nende elementide jaoks meetodid. Iga testitava lehe kohta luuakse uus klass. Sellise lähenemise eeliseks on, et testitava rakenduse kasutajaliidese koodi muutudes ole vaja muuta kirjeldatud teste, vaid ainult leheobjekti klassi koodi [10].

Spetsiaalselt leheobjektide mudeli toetamiseks on Selenium WebDriver API-liideses PageFactory teek [11]. PageFactory teeki kasutatakse töö raames loodavas testprogrammis.

Eeltoodust paremaks arusaamiseks tuuakse näide töö raames loodava testprogrammi koodist. Näitekood testib veebirakenduse kasutaja (kindlustusfirma klient) sisselogimist, mida tehakse Joonisel 4 toodud lehe kaudu. Kirjeldatakse kaks klassi: leheobjektide klass `LoginPage` sisselogimise lehe HTML elementide otsimine ja meetodid (Joonis 5); `TestLoginPage` testi läbiviimiseks, kus kasutatakse `LoginPage` klassi objekti meetodeid (Joonis 6).



Joonis 4: Kliendi sisselogimise leht.

```
public class LoginPage {

	// Otsitakse tekstivälja "Username/Email".
	@FindBy (how = How.ID, using = "pg01username")
	private WebElement username;

	// Otsitakse tekstivälja "Password".
	@FindBy (how = How.ID, using = "pg01password")
	private WebElement password;

	// Otsitakse nuppu "Continue".
	@FindBy (how = How.ID, using = "btnContinue")
	private WebElement continueBtn;

	// Meetod tekstiväljade täitmiseks.
	public void setLoginData(String Username, String Password)  {
		username.sendKeys(Username);
		password.sendKeys(Password);
	}
	// Getter-meetod nupu jaoks.
	public WebElement getBtnContinue()  {
		return continueBtn;
	}
}
```

Joonis 5: Leheobjekti klassi koodinäide.

```
public class TestLoginPage {

	// Uue LoginPage objekti loomine.
	private static LoginPage loginPage;
	public static void main(String[] args) {

		// Veebilehitseja käivitamine.
		WebDriver driver = new FirefoxDriver();

		// WebDriver objektile ooteaja määramine.
		// Ooteaeg määrab, kui kaua objekt ootab, enne kui otsustab, et test
		// ebaõnnestus.
		driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

		// LoginPage objekti väärtustamine.
		loginPage = PageFactory.initElements(driver, LoginPage.class);

		// Lehele navigeerimine.
		driver.get("http://www.example.com");

		// Leheobjekti meetodite väljakutse.
		// Parameetriteks antud teksti sisestamine.
		loginPage.setLoginData("kasutajanimi", "parool");
		// Nupu vajutamine.
		loginPage.getBtnContinue().click();

		// Veebilehitseja akna sulgemine.
		driver.quit();
	}
}
```

Joonis 6: Leheobjektidega loodud testi näitekood.

3.4 Andmetest juhinduv testimine

Andmetest juhinduv testimine on testimismetoodika, kus kasutatakse samu testlugusid erinevate sisendandmetega. Seda metoodikat kasutatakse tihti automatiseeritud testide disainimisel, kuna see võimaldab testitavat rakendust kontrollida suurema koguse testandmete vastu testi koodis olulisi muudatusi tegemata. Tavaliselt loetakse andmed sisse failist [12].

Peatükis 3.3 toodud koodinäite puhul võiks lugeda sisselogimiseks vajaliku kasutajanime ja parooli sisendfailist ning käivitada sama testi minu korda erinevate sisendandmetega.

Antud töö raames loodav testprogramm peab võimaldama testandmete varieerimist. Funktsionaalse nõude 3. kohaselt peab testprogramm lugema sisendandmeid MS Excel dokumendist.

3.5 Apache POI API-liides

Nõude 3 täitmiseks otsustas töö autor kasutada Apache POI API-liidest. See on Java API-liides, mis on loodud Microsoft Office failide loomiseks, lugemiseks, kirjutamiseks ja muutmiseks. POI sisaldab mitmeid komponente, mis võimaldavad töödelda erinevaid Microsofti failiformaate [13].

Antud töös kasutatakse neist ainult ühte – POI-XSSF. Sellega saab käsitleda MS Excel xlsx-dokumente [14]. Töö raames tegeletakse ainult dokumendist lugemisega.

3.6 Raamistik TestNG

TestNG on tarkvara testimiseks loodud raamistik, mille eesmärgiks on pakkuda võimalusi testide koodi struktureerimiseks. TestNG raamistiku annotatsioonid aitavad jagada testide koodi loogilisteks osadeks ning määrata nende koodiosade käivitamisele lisakriteeriume. Tabelis 5 tutvustatakse TestNG annotatsioone ja nende atribuute, mida käesoleva töö raames kasutati.

Tabel 5: TestNG annotatsioonid ja atribuudid [15].

Annotatsioon	Atribuut	Kasutus
`@BeforeSuite`		Lisatakse meetodile, mis kirjeldab tegevusi, mida tuleb teha enne testikomplekti käivitamist.
`@BeforeMethod`		Lisatakse meetodile, mis kirjeldab tegevusi, mida tuleb teha enne iga testklassis oleva meetodi käivitamist.
`@AfterMethod`		Lisatakse meetodile, mis kirjeldab tegevusi, mida tuleb teha peale iga testklassis olevat meetodit.
`@Test`		Kasutatakse testi samme kirjeldava klassi või meetodi määratlemiseks.
	`invocationCount`	Kasutatakse, et määrata mitu korda peab testi välja kutsuma. Vaikeväärtus on 1.
	`dataProvider`	Kasutatakse, et määrata milline klass või meetod annab testile sisendandmeid.
	`retryAnalyzer`	Kasutatakse, et määrata milline klass on vastutav testi korduvkäivitamise eest. Testi korduvkäivitamine toimub siis, kui test ebaõnnestus.
`@DataProvider`		Lisatakse meetodile, mis vastutab testile sisendandmete jagamise eest. Vastav meetod peab tagastama kahemõõtmelise objekt-tüüpi massiivi. Massiivi iga element on ühemõõtmeline objekt-tüüpi massiiv, mille iga elemendi saab anda testmeetodile parameetriks. Kahemõõtmelise massiivi ($m*n$) korral käivitatakse `dataProvider` atribuudiga testmeetodit massiivi ridade arv (m) korda.

TestNG raamistiku annotatsioonide näitlikustamiseks saab kasutada peatükis 3.3 kirjeldatud testi

peameetodiga klassi `TestLoginPage` ja muuta see TestNG raamistikule vastavaks klassiks (Joonis 7).

```
public class TestLoginPage {
	// Uue LoginPage objekti loomine.
	private LoginPage loginPage;
	// Uue WebDriver objekti loomine
	private WebDriver driver;

	@BeforeMethod
	public void setUp()	{
		// Veebilehitseja käivitamine.
		driver = new FirefoxDriver();

		// WebDriver objektile ooteaja määramine.
		// Ooteaeg määrab, kui kaua objekt ootab, enne kui otsustab, et
		//test ebaõnnestus.
		driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

		// LoginPage objekti väärtustamine.
		loginPage = PageFactory.initElements(driver, LoginPage.class);
	}

	@Test(invocationCount = 2)
	public void testLogin() {
		// Lehele navigeerimine.
		driver.get("http://www.example.com");

		// Lehe meetodite väljakutse.
		// Parameetriteks antud teksti sisestamine.
		loginPage.setLoginData("kasutajanimi", "parool");
		// Nupu vajutamine.
		loginPage.getBtnContinue().click();
	}

	@AfterMethod
	public void tearDown()  {
		// Veebilehitseja akna sulgemine.
		// WebDriver objekti kustutamine.
		driver.quit();
	}
}
```

Joonis 7: TestNG testi koodinäide.

Joonisel 7 toodud muudatused on järgmised:

- veebilehitseja käivitamine, WebDriver objekti seadistamine ja leheobjekti klassi `LoginPage` väärtustamine liigutati `@BeforeMethod` annotatsiooniga meetodisse `setUp()`;
- testi sammud liigutati `@Test` annotatsiooniga varustatud meetodisse `testLogin()`, millele lisati ka atribuut `invocationCount`;
- veebilehitseja sulgemine liigutati `@AfterMethod` annotatsiooniga meetodisse `tearDown()`.

Testide puhul, mille `@Before` ja `@After` annotatsioonidega meetodid on sarnased või ühesugused, on koodikorduste vältimiseks mõistlik need meetodid liigutada eraldi klassi. Testide klassid on sellisel juhul selle klassi alamklassid.

Testide seadistamiseks ja käivitamiseks kasutab TestNG raamistik peameetodi asemel XML-formaadis faile. Neid faile kasutatakse selleks, et kohandada testikomplekte vastavaks kasutaja vajadustele. Kuna raamistik võimaldab XML-failis määrata väga paljusid seadistusi, siis keskendutakse ainult antud töö raames kasutatule.

Käesolevas töö praktilises osas kasutatakse XML-faili, et kirjeldata testikomplekti (XML-märgend `<suite>`) sisu ja määrata selle nimi. Testikomplekti sisuks on sellesse kompleti kuuluvad testid. XML-failis loetletakse need testid (XML-märgend `<test>`) ja antakse neile nimed. Testi sisuks on klassid (XML-märgend `<classes>`), mille poole selle testi raames pöördutakse. XML-failis loetletakse need klassid (XML-märgend `<class>`). Vaikimisi kutsutakse välja kõik klassis olevad annoteeritud meetodid, kuid XML-failis on võimalik eraldi välja tuua, millised meetodid nimetatud klassist välja kutsutakse (XML-märgend `<methods><include>`).

Joonisel 8 on toodud näide TestNG raamistiku XML-failist. Näide on võetud töö praktilises osas loodavast testprogrammist.

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Testing Filling Quotes For Production Monitoring" verbose="2">

<listeners>
      <listener class-name="testUtility.CustomTestListener" />
</listeners>

<test name="Filling FamilyFriendly Quote">
<classes>
    <class name="tests.testFamilyCoverForProd">
    <methods>
        <include name="testFillFamilyCover" />
    </methods>
     </class>
</classes>
</test>

</suite>
```

Joonis 8: TestNG XML-dokumendi näide.

Märgendi `<suite>` atribuut `verbose` seadistab, kui palju tagasisidet annab TestNG sisseehitatud logimissüsteem. Näide antud seadistuse puhul programmi konsooli suunatud vaikimisi väljundist on toodud Joonisel 9.

XML-märgend `<listener>` määrab testikompletile kohandatud kuularklassi, mis on TestNG raamistikku sisseehitatud klassi `TestListenerAdapter` alamklassiks.

`TestListenerAdapter` on klass, milles hoitakse infot kõigi käivitatud ja lõpetanud testide kohta. See sisaldab meetodeid, mille kaudu pääseb ligi testide tulemustele ning meetodeid, mis reageerivad nendele tulemustele [16].

Antud töö raames loodavas testprogrammis on kirjeldatud `TestListenerAdapter` klassi alamklass `CustomTestListener`. See sisaldab ülekaetud meetodeid, mis kirjeldavad programmi käitumist testi alguses, lõpus, õnnestumisel ja ebaõnnestumisel.

```
PASSED: testFillFamilyCover
FAILED: testFillFamilyCover
org.openqa.selenium.TimeoutException: Timed out after 15 seconds waiting for
element ([[FirefoxDriver: firefox on LINUX (2e960289-71f7-4535-8b60-
707ceb2e99af)] -> link text: Delete]) to become stale
Build info: version: '2.45.0', revision: '32a636c', time: '2015-03-05 22:01:35'
...
===============================================
    Filling FamilyFriendly Quote
    Tests run: 2, Failures: 1, Skips: 0
===============================================


===============================================
Testing Filling Quotes For Production Monitoring
Total tests run: 2, Failures: 1, Skips: 0
```

Joonis 9: TestNG testi väljundi näide.

3.7 Apache Log4j API-liides ja ExtentReports teek

Loodav testprogramm peab andma tagasisidet testi sammude ja tulemuste kohta. Mida rohkem informatsiooni testi läbiviimise kohta jagatakse, seda paremini saab testi ebaõnnestumisel välja selgitada vea põhjuse. Töö praktilises osas on testide kohta tagasiside andmiseks kasutatud Apache Log4j API-liidest ning ExtentReports teeki. Kuna tegemist on mahukate tehnoloogiatega, siis keskendutakse ainult töös kasutatud osadele.

Apache Log4j on Java API-liides, mis on spetsiaalselt loodud programmile logimise funktsionaalsuse lisamiseks. Log4j võimaldab lisaks logisõnumite väljatrükkimisele ka neid kategoriseerida vastavalt tasemele. API-liides sisaldab eraldi meetodeid vigade (*error*), hoiatuste (*warning*), informatsiooni (*info*) ja silumise (*debug*) sõnumite jaoks. Oluline eelis tavalise sõnumite väljundisse suunamise ees on Log4j API-liidesel see, et ta võimaldab logimise seadistamist eraldi XML-formaadis failis. Selles seadistusfailis määratakse, milliste kategooriate sõnumeid trükitakse ja kuhu väljund suunatakse. Lisaks saab seadistada väljundsõnumite ja logifailide nimesid ja logisõnumite formaate [17].

ExtentReports on Java teek, mis on loodud Seleniumi testide tulemuste kohta HTML kujul raportite

genereerimiseks. Sarnaselt Log4j API-liidesele, sisaldab ka ExtentReports teek meetodeid erineva kategooriaga sõnumite raportisse suunamiseks: testi õnnestumine (*success*), testi ebaõnnestumine (*failure*), informatsioon ja hoiatus. Lisaks sisaldab teek meetodeid raporti HTML-koodi kohandamiseks [18].

Joonisel 10 on toodud näide koodist, kuhu on lisatud Log4j logimine ja ExtentReprots raporteerimine. Näites on lisatud logimine peatükis 3.3 kirjeldatud leheobjektide klassile `LoginPage`.

```
public class LoginPage {

	// Log4j Logger objekti loomine.
	private static final org.apache.log4j.Logger log =
				Logger.getLogger(CallCenterLoginModal.class);
	// ExtentReports objekti loomine.
	private static final ExtentReports extent =
				ExtentReports.get(CallCenterLoginModal.class);

	// Otsitakse tekstivälja "Username/Email".
	@FindBy (how = How.ID, using = "pg01username")
	private WebElement username;

	// Otsitakse tekstivälja "Password".
	@FindBy (how = How.ID, using = "pg01password")
	private WebElement password;

	// Otsitakse nuppu "Continue".
	@FindBy (how = How.ID, using = "btnContinue")
	private WebElement continueBtn;

	// Meetod tekstiväljade täitmiseks.
	public void setLoginData(String Username, String Password)  {
		username.sendKeys(Username);
		password.sendKeys(Password);
		// Logisse ja raportisse kirjutamine.
		log.info("Sisestan kasutajanime:parooli: " + Username + ":" +
								Password);
		extent.log(LogStatus.INFO, "Sisestan kasutajanime:parooli", Username
							+ ":" + Password);
	}
	// Getter-meetod nupu jaoks.
	public WebElement getBtnContinue()  {
		// Logisse ja raportisse kirjutamine.
		log.info("Vajutan sisselogimise nuppu");
		extent.log(LogStatus.INFO, "Vajutan sisselogimise nuppu", "");
		return continueBtn;
	}
}
```

Joonis 10: Koodinäide Log4j logimisest ja ExtentReports raporteerimisest.

Käesoleva töö lisadest leiab näited testprogrammi poolt loodud Apache Log4j logifailidest (Lisa 2) ja ExtentReports HTML raportist (Lisa 3).

4 Testprogramm

Käesolevas peatükis tuuakse välja töö praktilise osana loodud testprogrammi struktuur ning selgitatakse täpsemalt, kuidas eelmises peatükis tutvustatud tehnoloogiad ning disainimustreid on programmis kasutatud.

Käeoleva töö Lisa 1 sisaldab töö praktilise osa lähtekoodi. Alltoodud tabelite ja jooniste eesmärgiks on kirjeldada lugejale Lisas 1 leiduvate kaustade, failide, Java pakettide ja klasside sisu.

Tabelis 6 on loetletud kaustad, mida programmi lähtekoodist leida võib ja antakse lühikokkuvõtte nende sisust.

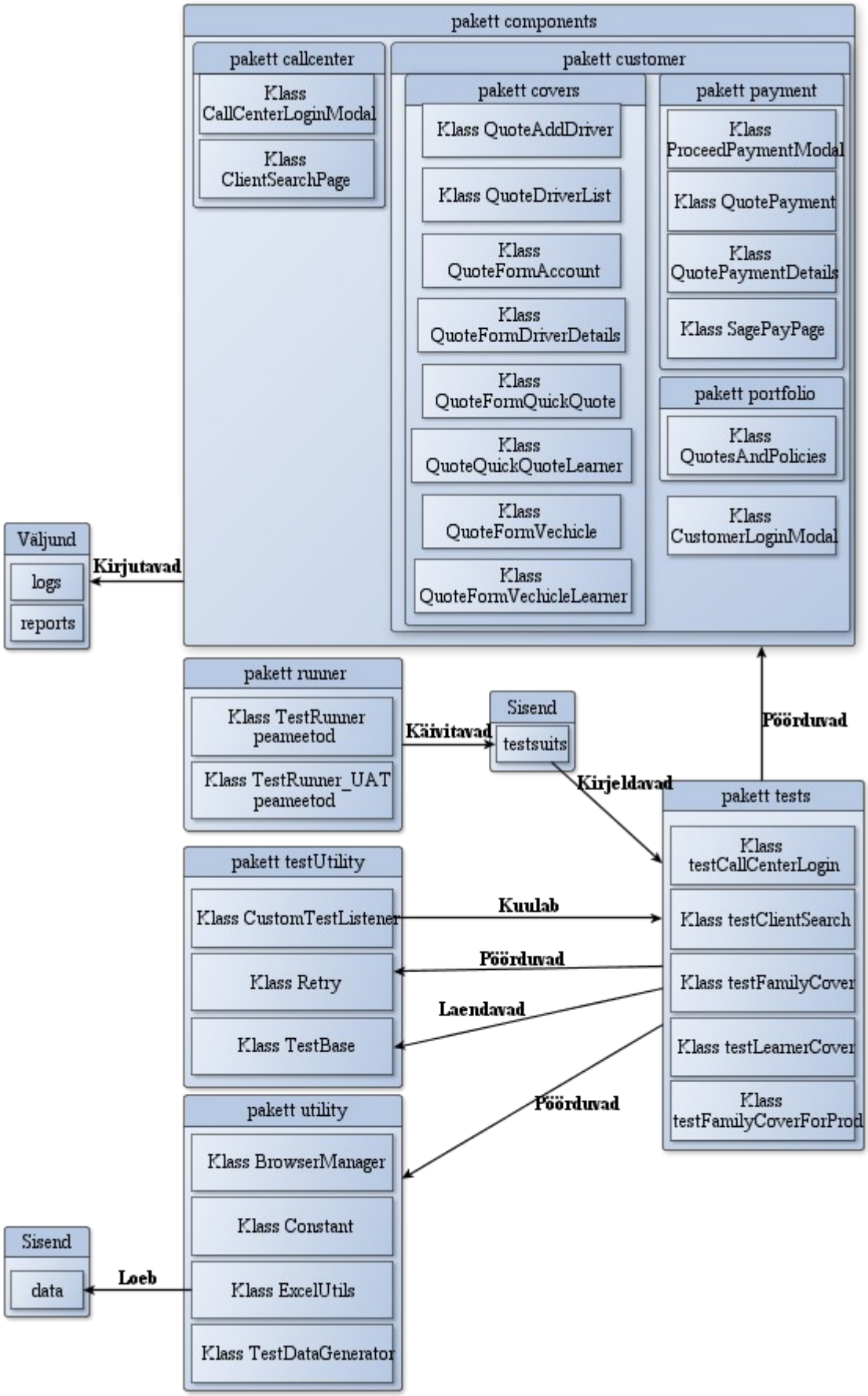
Tabel 6: Programmi kaustastruktuur.

Kaust	Kirjeldus
data	Sisendfailid testandmetega MS Excel dokumentide kujul.
doc	Testprogrammi Javadoc.
lib	Kasutatud raamistike ja API-liideste teegid.
logs	Väljund. Log4j logifailid.
reports	Väljund. ExtentReports HTML raportid.
resources	Log4j seadistustega XML-dokument.
src	Programmi lähtekood. Java paketid.
testsuits	TestNG testikomplektide XML-dokumendid.

Tabelid 7 nimetatakse programmi lähtekoodi Java paketid, mis asuvad kaustas „src”, ja antakse lühikokkuvõtte nende eesmärgist.

Tabel 7: Lähtekoodi Java paketid.

Pakett (*package*)	Kirjeldus
`components`	Leheobjektide klassid.
`runner`	Peameetodiga klassid TestNG XML-dokumentides kirjeldatud testikomplektide käivitamiseks.
`tests`	Testide klassid.
`testUtility`	Kohandatud kuular, korduvkäivitamise klass ja testide ülemklass.
`utility`	Klassid veebilehitseja üldiseks haldamiseks, MS Excel dokumentide lugemiseks, konstant väärtuste hoidmiseks, testandmete genereerimiseks.



Joonis 11: Testprogrammi Java pakettide seosed.

Joonisel 11 on kujutatud programmi pakettides olevate klasside omavahelist suhtlust. Programmi keskmes on paketi `tests` sisu, milles olevate testide klasside meetodid pöörduvad teistes pakettides olevate klasside meetodite ja muutujate poole. Paketis `components` on testitava lahenduse lehtede leheobjektide klassid. Testid loovad leheobjektide klasside uusi objekte ning viivad läbi teste leheobjektide klassides kirjeldatud meetoditega.

Pakett `testUtility` pakub testidele tugiteenuseid:

- klass `TestBase`: testide klasside ülemklass, kus on defineeritud `@Before` ja `@After` annotatsioonidega meetodid ehk toimingud, mida teha enne ja pärast testide käivitamist;
- klass `Retry`: kutsutakse välja, kui see on testmeetodi `@Test` annotatsiooni `retryAnalyzer` atribuudi väärtusena antud, vastutab testi ebaõnnestumisel testi kohese taaskäivitamise eest;
- klass `CustomTestListener`: TestNG `TestListenerAdapter` klassi alamklass, kohandatud kuular, mille meetodid kutsutakse välja testide tulemuste peale.

Pakett `utility` sisaldab:

- klassi `BrowserManager`: kohandatud meetodid veebilehitseja käivitamiseks ja sulgemiseks;
- klassi `ExcelUtils`: sisaldab meetodeid MS Excel dokumendist testandmete lugemiseks;
- klassi `TestDataGenerator`: sisaldab universaalseid meetodeid testandmete genereerimiseks;
- klassi `Constant`: defineerib konstantsed väärtused.

Testikomplektid on kirjeldatud TestNG XML-dokumentides, mis asuvad kaustas „testsuits”. Pakett `runner` sisaldab peameetodiga klasse, mis käivitavad teste TestNG XML-dokumentide poole pöördudes.

5 Kokkuvõte

Käeoleva bakalaureusetöö peamiseks eesmärgiks oli luua programm veebirakenduse testimise automatiseerimiseks ja tutvustada ühte võimalust selle programmi loomiseks – raamistikuga Selenium.

Töö esimeses osas kirjeldati olukorda, kus testimise automatiseerimine võiks olla mõttekas ning toodi välja nüansid, mis testprogrammi loomisel võiksid olulised olla. Nimetati põhjused, miks testprogrammi loomise kasuks otsustati. Peatükk kirjeldas testitavat rakendust ning püstitas nõuded töö praktilises osas loodavale testprogrammile.

Töö teises osas tutvustati veebirakenduste testimise automatiseerimise tehnoloogiaid. Anti ülevaade veebilehitsejate automatiseerimise raamistikust Selenium ning selle erinevatest võimalustest. Toodi välja mitmeid API-liideseid ja teeke, mida saab kasutada koos raamistikuga Selenium, et täita esimeses osas testprogrammile esitatud nõudeid.

Töö kolmandas osas anti täpsem ülevaade sellest, kuidas teises osas kirjeldatud tehnoloogiaid kasutati töö praktilises osas loodud testprogrammi koostamisel. Peatüki eesmärgiks on abistada töö lugejat antud töö Lisas 1 toodud programmi lähtekoodist arusaamisel.

Praktilise osana loodud testprogrammi on võimalik edasi arendada mitmel viisil. Näiteks saab sellele lisada uusi teste juba kirjeldatud lehtedele, kui ka kirjeldada uusi lehti ning neid testida. On võimalik olemasolev programm siduda Selenium Grid tööriistaga, et saaks teste jooksutada paralleelselt mitmes masinas, mis vähendab testimise aega.

Kuigi töö raames loodud testprogramm on spetsiifiline ainult ühele veebirakendusele, siis töös kirjeldatud tehnoloogiaid on võimalik rakendada ükskõik millise veebipõhise rakenduse testimise automatiseerimisel.

6 Kasutatud kirjandus

[1] Glenford J. Myers, "The Art of Software Testing, Second Edition", John Wiley & Sons, Inc., Hoboken, New Jersey, 2004.

[2] Oracle Java dokumentatsioon, https://docs.oracle.com/javase/tutorial/java/annotations/, (vaadatud 07.05.2015).

[3] Seleniumi koduleht, "What is Selenium?", http://www.seleniumhq.org/, (vaadatud 07.05.2015).

[4] Seleniumi koduleht, "Selenium IDE", http://www.seleniumhq.org/projects/ide/, (vaadatud 07.05.2015).

[5] Seleniumi koduleht, "Selenium Remote Control", http://www.seleniumhq.org/projects/remote-control/, (vaadatud 07.05.2015).

[6] Seleniumi koduleht, "Selenium WebDriver", http://www.seleniumhq.org/projects/webdriver/, (vaadatud 07.05.2015).

[7] Seleniumi koduleht, "Selenium Projects", http://www.seleniumhq.org/projects/, (vaadatud 07.05.2015).

[8] Dave Hunt, Luke Inman-Semerau et.al, Seleniumi dokumentatsioon, "Supported Browsers and Platforms", viimati uuendatud 04.05.2015, http://docs.seleniumhq.org/docs/01_introducing_selenium.jsp#supported-browsers-and-platforms, (vaadatud 07.05.2015).

[9] Dave Hunt, Luke Inman-Semerau et.al, Seleniumi dokumentatsioon, "Introducing the Selenium-WebDriver API by Example", viimati uuendatud 04.05.2015, http://www.seleniumhq.org/docs/03_webdriver.jsp#introducing-the-selenium-webdriver-api-by-example, (vaadatud 07.05.2015).

[10] Dave Hunt, Luke Inman-Semerau et.al, Seleniumi dokumentatsioon, "Page Object Design Pattern", viimati uuendatud 04.05.2015, http://www.seleniumhq.org/docs/06_test_design_considerations.jsp#page-object-design-pattern, (vaadatud 07.05.2015).

[11] PageFactory Javadoc, http://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/support/PageFactory.html, (vaadatud 07.05.2015).

[12] Dave Hunt, Luke Inman-Semerau et.al, Seleniumi dokumentatsioon, "Data Driven Testing", viimatu uuendatud 04.05.2015, http://www.seleniumhq.org/docs/06_test_design_considerations.jsp#data-driven-testing, (vaadatud 07.05.2015).

[13] Andrew C. Oliver, Glen Stampoultzis et al., Apache POI koduleht, https://poi.apache.org/index.html, (vaadatud 07.05.2015).
[14] Andrew C. Oliver, Nicola Ken Barozzi, POI-HSSF ja POI-XSSF dokumentatsioon, https://poi.apache.org/spreadsheet/index.html, (vaadatud 07.05.2015).
[15] Cédric Beust, TestNG dokumentatsioon, http://testng.org/doc/documentation-main.html, (vaadatud 07.05.2015).
[16] Cedric Beust, TestListenerAdapter Javadoc, http://testng.org/javadoc/org/testng/TestListenerAdapter.html, (vaadatud 07.05.2015).
[17] Ceki Gülcü, "Log4j delivers control over logging", 2000, http://www.javaworld.com/article/2076243/java-se/log4j-delivers-control-over-logging.html, (vaadatud 08.05.2015).
[18] Anshoo Arora, ExtentReports dokumentatsioon, http://relevantcodes.com/extentreports-documentation/, (vaadatud 08.05.2015).

Lisa 1

Testprogrammi lähtekood:

https://www.dropbox.com/sh/qrthbeavgdt4t77/AAB4BAI-balALofjv3iFez4Xa?dl=0

Lisa 2

Logifaili näidis:

https://www.dropbox.com/sh/dc3q98fuw1zef0i/AAAMr1zaHoh1OjUbj8t4p8pNa?dl=0

Lisa 3

HTML raporti näidis:

https://www.dropbox.com/sh/am2d09l3539o5um/AAA3VvlmFfiO2sHwQvAztlFpa?dl=0

Lisa 4 - Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Elo Eilonen, (sünnikuupäev: 19.05.1988)
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
„Veebirakenduse automatiseeritud testide loomine raamistikuga Selenium”,
(*lõputöö pealkiri*)

mille juhendaja on Helle Hein,
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **14.05.2015**