TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl-Joosep Kesküla 185129IADB

KASUTAJALIIDESE TESTIDE ARENDAMINE KOOS SIDUSARENDUSE PROTSESSI JUURUTAMISEGA FINANTSASUTUSE NÄITEL

Bakalaureusetöö

Juhendajad: Brita Moorus
Bakalaureus
Merje Einla
Bakalaureus

Tallinn 2022

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-Joosep Kesküla

16.05.2022

Annotatsioon

Töö autor töötab finantsasutuses, kus toimub iga kahe nädala tagant regressioonitestimine. Suurem osa testimisest toimub käsitsi, mis on ajakulukas. Töö eesmärgiks on luua ettevõttesisesele tellerirakendusele kiired ning selgete testitulemustega kasutajaliidese testid ning integreerida kõik meeskonnasisesed testid sidusarenduse vahendiga, et vähendada aega, mis kulub käsitsi testimisele. Töö käigus analüüsitakse erinevaid testimisraamistikke ning sidusarenduse vahendeid. Analüüside tulemusena kasutatakse töös vastavaid raamistikke ja sidusarenduse vahendit, mis rahuldavad töö eesmärgi täitmise nõudeid. Lisaks kirjeldatakse kirjutatud testide ülesehitust ja kirjutamise protsessi ning sidusarenduse vahendiga integreerimise protsessi.

Töö tulemusena valmisid kasutajaliidese testid, mis vastasid kõikidele püstitatud nõuetele. Arendatud kasutajaliidese testid ning olemasolevad testid integreeriti sidusarenduse vahendiga Jenkins. Jenkinsis käivitatakse igal tööpäeva hommikul testid, mis annavad ülevaate eelmisel päeval tehtud rakenduse muudatuste edukusest.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 27 leheküljel, 8 peatükki, 11 joonist, 1 tabelit.

Abstract

Development of User Interface Tests with Continuous Integration Process Implementation on the Example of a Financial Institution

Author of this thesis works in a financial institution, where regression testing happens biweekly. Regression testing involves a lot of manual testing. Manual testing is very time consuming and it often is an excessive amount of work for testers. Due to manual testing and long testing process bugs are discovered in the late phase of development. Discovering bugs in the late phase means that it needs more resource to fix them. Also manual testing is error-prone, because tester might overlook something that should be considered as a bug. Manual testing and discovering bugs too late could mean that bugs reach to end users, which could damage the company's reputation or cause even financial harm.

The aim of this thesis is to develop fast and stable user interface tests for internal teller application with clearly legible test results and integrate those and existing tests with continuous integration tool. These tests will replace existing Ruby user interface tests that are difficult to maintain. Developed user interface tests and integration with continuous integration tool reduce time required for manual testing.

As a result of this thesis, user interface tests were developed that fulfilled set requirements. Developed and existing tests were integrated with continuous integration tool Jenkins. Jenkins automatically runs all tests on every weekday's morning and provides results of previous day's developments successfulness.

The thesis is in Estonian and contains 27 pages of text, 8 chapters, 11 figures, 1 table.

Lühendite ja mõistete sõnastik

Dockeri tõmmis	*Docker image*, Dockeri konteineri loomiseks infot sisaldav mall
Dockeri konteiner	Tarkvara sisaldav pakett rakenduse käivitamiseks
Gradle	Koodi ehitamise automatiseerimise vahend
HTML	*HyperText Markup Language*, veebilehtede struktureerimiseks kasutatav keel
IDE	*Integrated development environment*, tarkvara koodi kirjutamiseks
Jira tarkvara	Projekti ülesannete korraldamise vahend
Lähtekood	Tarkvara toimimist kirjeldav kood
Mikroteenus	Eraldiseisev koodiosa, mis täidab mõnda väikest ülesannet
MVP	*Minimum viable product*, väärtust omav minimaalne toode
PDF	*Portable Document Format*, dokumentide jms salvestamiseks kasutatav failitüüp
Pistikprogramm	*Plugin*, tarkvara, mis täiustab mõnda teist tarkvara
Regressioonitestimine	Varasemalt testitud tarkvara uuesti testimine, selleks et veenduda, et uus muudatus ei mõjuta olemasolevat funktsionaalsust
Repositoorium	Võrgus asuv koodi sisaldav andmekogum
Süsteemi omadus	*System property*, süsteemi kohta infot sisaldav omadus
URL	*Uniform Resource Locator*, veebilehe aadress
WDM	WebDriverManager, testimisel kasutatav teek

Sisukord

1 Sissejuhatus 10
2 Testimisraamistike analüüs 12
2.1 Selenium WebDriver 12
2.2 Selenide 12
2.3 JUnit 13
2.4 Cucumber 14
2.5 Analüüsi tulemused 15
3 Sidusarenduse süsteemide analüüs 17
3.1 Sidusarendus 17
3.2 Jenkins 17
3.3 Atlassian Bamboo 18
3.4 Analüüsi tulemused 18
4 Testide arendamine 20
4.1 Olemasolev lahendus 20
4.2 Testide kirjeldus ja ülesehitus 21
4.3 Testide kirjutamine 23
4.3.1 Mustri loomine Balti riikide jaoks 24
4.3.2 Vahekaartide täitmine 24
4.4 Testide lokaalne käivitamine ning testide tulemused 27
5 Sidusarenduse juurutamise protsess 28
5.1 Sidusarendusega integreerimise analüüs 28
5.1.1 Ettevõttesisesed lahendused 28
5.1.2 Selenium Grid 29
5.1.3 WebDriverManager 30
5.1.4 Analüüsi tulemus 30
5.2 Lahendus 31
5.2.1 Jenkinsiga integreerimine 31
5.2.2 Probleemid Jenkinsiga integreerimisel 33
6 Tulemuste analüüs 34

7 Edasine arendus 35
8 Kokkuvõte 36
Kasutatud kirjandus 37
Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks 40

Jooniste loetelu

Joonis 1. Gherkin süntaksit kasutades tehtud testi kirjeldus. 15
Joonis 2. Toimingute täitmise kirjeldus. 15
Joonis 3. Enne igat testi käivitatav meetod. 21
Joonis 4. Testi käivitamise meetod. 21
Joonis 5. Testi poolt brauseris tehtavad toimingud. 22
Joonis 6. Laenutoote vahekaardi täitmine. 25
Joonis 7. Vahekaardi täitmise meetod. 25
Joonis 8. Kaastaotleja lisamine kodulaenu taotlusele. 26
Joonis 9. Jenkinsi käivituse tuvastamine. 31
Joonis 10. WebDriveri loomine Jenkinsi jaoks. 32
Joonis 11. WebDriveri sätestamine lokaalsel ja Jenkinsi käivitusel. 32

Tabelite loetelu

Tabel 1. Testi sammud. 23

1 Sissejuhatus

Korrektselt ja põhjalikult testitud tarkvara on ettevõtte maine seisukohalt väga tähtis. Hästi testitud tarkvara puhul on vähetõenäoline, et kriitilised vead jõuavad lõppkasutajani. Rakendust saab testida käsitsi, kuid saab kasutada ka valmis kirjutatud teste. Käsitsi testimine on ajakulukas töö ning hooletuse tõttu võivad jääda mõned vead märkamata. Siinkohal tulevad kasuks automaattestid. Automaattestid aitavad hõlbustada testimise protsessi, kattes osaliselt testitavat rakendust või täielikult. Kattes osaliselt rakendus automaattestidega vähendatakse käsitsi testimisele kuluvat aega, kuid käsitsi testimist on vaja siiski teha. Täieulatuslik testidega kaetavus võtab aga palju aega.

Töö autor töötab finantsasutuse ettevõttes, kus toimub iga kahe nädala tagant rakenduse uute versioonidega seoses uute arenduste testimine ehk regressioonitestimine. Selles suur osa on käsitsi testimine. Ajakuluka testimise tõttu saavad arendajad liiga hilja vigadest teada ning käsitsi testimise tõttu on testijad ka ülekoormatud. Lisaks ei pruugi käsitsi testimise käigus inimliku hooletusvea tõttu kõiki vigu täheldada ning see suurendab võimalust, et vead jõuavad lõppkasutajateni.

Töö eesmärgiks on luua ettevõttesisese tellerirakenduse jaoks kasutajaliidese testid ning integreerida need testid ja ka olemasolevad testid sidusarenduse vahendiga, et vähendada aega, mis kulub käsitsi testimisele. Arendatavad kasutajaliidese testid ning testide integratsioon sidusarenduse vahendiga vähendavad aega, mis kulub käsitsi testimisele. Sidusarenduse vahendiga integreerimise eesmärgiks on luua testid, mis käivitatakse automaatselt ja regulaarselt. Selle tulemusena saavad arendajad oma vigadest teada varasemas arendusfaasis ning vigade parandamine nõuab vähem ressursse. Lisaks on väiksem tõenäosus, et vead jõuavad lõppkasutajateni.

Töö käigus arendatavad testid hakkavad asendama olemasolevaid Ruby keeles kirjutatud kasutajaliidese teste. Ruby keeles kirjutatud testid on raskesti hallatavad ning kuna meeskonnasiseselt on juba kasutusel muud testid, mis on kirjutatud Java keeles, siis töö käigus arendatavad testid kirjutatakse samuti Java keeles. Lisaks hallatavamatele

testidele, peavad arendatud testid olema kiired ning pakkuma selgesti loetavaid testitulemusi.

Töö koosneb kuuest peatükist. Esimeses peatükis analüüsitakse testimisraamistikke. Analüüsitakse nii ettevõttes olemasolevaid kui ka teisi raamistikke ning analüüsi tulemusena otsustatakse milliseid kasutada vastavalt seatud eesmärkidele. Töö teises peatükis analüüsitakse sidusarenduse vahendeid. Kolmandas peatükis kirjeldatakse ära kasutajaliidese testide kirjutamise protsess ning neljandas peatükis kirjeldatakse valitud sidusarenduse vahendiga integreerimise protsess. Viiendas peatükis kirjeldatakse ja analüüsitakse töö käigus saavutatud tulemusi. Kuuenda peatüki teemaks on edasised arendused.

2 Testimisraamistike analüüs

Testimisraamistike valikul on tähtis, et testid oleksid hallatavad, loetavad, taaskasutatavad ning pakuksid selgesti loetavaid testitulemusi. Analüüsi peatükis uuritakse erinevaid testimisraamistikke ning tulemusena valitakse töö eesmärgi täitmiseks sobivad raamistikud. Ettevõttes on juba kasutusel Selenide ja JUnit raamistikud, kuid analüüsi tulemustel selgub, kas need on piisavad eesmärgi täitmiseks või on mõistlik kasutada muid raamistikke.

2.1 Selenium WebDriver

Selenium on avatud lähtekoodiga testimistarkvara komplekt. Enimkasutatavaks Seleniumi osaks on Selenium WebDriver [1]. Enne Selenium WebDriverit oli Seleniumi põhiproduktiks Selenium Remote Control (RC). Selenium RC kasutas serverit, et edastada käske brauserile. Serveri pidi eelnevalt looma ja käima panema. Selenium WebDriver on asendus Selenium RCle [2]. Selenium WebDriver ei vaja mingit vaheserverit, mille kaudu brauseriga suhelda, vaid see võimaldab veebibrauseriga otse suhelda. Seetõttu on Selenium WebDriver kiirem. WebDriver pakub paremat tuge dünaamiliste veebilehtede testimisele. Dünaamilised veebilehed on sellised, kus veebilehe sisu muutub ilma lehe uuesti laadimiseta. Selenium WebDriver pakub tavalise brauseri kasutamisega identseid võimalusi, mis tähendab, et näiteks ei ole võimalik sisestada teksti väljadele mis on peidetud. Selenium WebDriver iseenesest ei paku testijale hästiloetavaid testitulemusi või näiteks kuvatõmmiseid brauserist ebaõnnestunud testi korral. Seetõttu ei ole tavalise Selenium WebDriveri kasutamine otstarbekas, vaid efektiivsem on kasutada raamistikke, mis kasutavad Selenium WebDriverit [2].

2.2 Selenide

Selenide on avatud lähtekoodiga raamistik, mis on üles ehitatud Selenium WebDriveri peale. Selenide pakub hulgaliselt lisafunktsioone võrreldes tavalise Selenium WebDriveri

kasutamisega. Selle suurimad eelised on automaatsed kuvatõmmised, ootamisfunktsioon ja sisseehitatud hulgalised HTML elementide kasutused [3].

Automaatne kuvatõmmis tehakse juhul kui test ebaõnnestub. Kuvatõmmis tehakse testi ebaõnnestumise hetkel brauseris nähtavast pildist. See kuvatõmmis aitab hõlpsasti hilisemas analüüsis tuvastada, mis oli testi ebaõnnestumise põhjuseks. Lisaks kuvatõmmisele salvestatakse ka lehe HTML-formaat, mis aitab kaasa ebaõnnestumise põhjuse leidmisele [4].

Sisseehitatud ootamisfunktsioon ootab dünaamiliste veebilehtede puhul ära, kuniks kõik HTML elemendid on laetud ning nähtaval ning seejärel alles tegutseb. Muidu võib juhtuda olukord, kus lehekülg ei jõua ära laadida ning test ebaõnnestub selle tõttu, et mõni nõutud väli või koodi poolt kasutatav HTML element ei ole ära laetud ning kättesaadav [5]. Selenide pakub ka hulgaliselt sisseehitatud HTML elementidega käitumise funktsioone [4]. Neid on ka väga mugav vastavalt vajadusele juurde luua.

Selenide konfiguratsioonis on võimalik muuta väga paljusid erinevaid väärtusi, mille tõttu on olemas kohene tugi populaarseimatele veebibrauseritele (nt. Google Chrome, Firefox, Opera, Microsoft Edge). Konfiguratsiooni muutes on võimalik toetada ka testide jooksutamist vähemtuntud brauserite peal (phantomjs, htmlunit) [3].

2.3 JUnit

JUnit on pidevalt arendatav avatud lähtekoodiga testimisraamistik, mis on mõeldud Java keele jaoks [6], [7]. JUniti eesmärk on suunata arendajaid testipõhise arenduse suunas (ingl k. *test-driven development*), kus testid kirjutatakse valmis enne arendust ning testitakse ühiktestidega iga osa mida arendatakse [7], [8]. Ühiktestiks nimetatakse testi, mis testib isoleeritult ühte koodi osa. Üldiselt on Javas üheks ühikuks klass [8]. JUnit pakub hulgaliselt võimalusi nagu näiteks annotatsioonid, väited (ingl k. *assertions*), testide keelamist, tingimustega testide käivitamist ja testide käivitamise järjekorra prioritiseerimist [9].

Annotatsioonid kirjutatakse Java klassi, meetodi või muutuja peale. Annotatsioonide abil saab näiteks [9]:

- märkida milline meetod on testmeetod;

- märkida testi korduste arv ebaõnnestumise puhul;
- märkida meetodeid, mis tuleb käivitada enne või pärast igat testmeetodit;
- ignoreerida testi käivitamist.

Vajadusel on võimalik ka luua enda tingimustele sobivaid annotatsioone. Näiteks kui testi tuleb ignoreerida vaid teatud keskkonnas käivitamisel. Annotatsioonide abil saab muuta koodi palju loetavamaks ning taaskasutatavamaks.

Väited võimaldavad koodis kontrollida mingisuguse tingimuse rahuldamist või mitterahuldamist. Väidete abil saab näiteks [10]:

- kontrollida tingimuste võrdelisust;
- kontrollida tingimuse tõeväärtust;
- kontrollida korrektse erindi viskamist;
- sätestada ajapiirangut koodi jooksutamisele.

Väited on hea viis muuta kood loetavamaks. Väite vääraks osutumisel visatakse erind, kus on selgelt kirjeldatud oodatud tulemus ning tegelik tulemus.

2.4 Cucumber

Cucumber on avatud lähtekoodiga Ruby keeles kirjutatud testimisraamistik, mis toetab käitumisel põhinevat arendust (ingl k. *behavior-driven development*) [11]. Selle abil on võimalik kirjutada teste, millest saavad kõik aru, sõltumata nende tehnilistest teadmistest [12]. See loob võimaluse kirjutada teste kõigil. Näiteks ärianalüütik, kes ei valda programmeerimiskeeli nii hästi kui arendajad, saab kirjutada teste oma rakendusele.

Cucumber võimaldab kirjutada teste kasutades inglise keelt. Selleks peab teksti kirjutama *feature* tüüpi faili. See fail järgib keele Gherkin sätetestatud süntaksireegleid. Testide struktureerimiseks kasutatakse märksõnu (Joonis 1).

```
Scenario: Close WebDriver
Given I have open WebDriver
When close WebDriver
Then WebDriver is closed
```

Joonis 1. Gherkin süntaksit kasutades tehtud testi kirjeldus.

Märksõnade abil kirjeldatakse ära testis tehtavad toimingud. Toimingu definitsiooni failis on ära kirjeldatud toimingute täitmine (Joonis 2) ning see toimub mõnes progammeerimiskeeles nagu näiteks Java või Ruby [11].

```
@Given(„I have open WebDriver“) {
  webDriver = new ChromeDriver();
}

@When(„close WebDriver“) {
  webDriver.quit();
}

@Then(„WebDriver is closed“) {
  assertTrue(webDriver.isClosed());
}
```

Joonis 2. Toimingute täitmise kirjeldus.

Lihtne keeleline süntaks võimaldab teste kirjutada igaühel, kuid arendajad peavad siiski toimingute täitmise kirjutamisele aega kulutama. Lihtne arusaam testist võimaldab paremat suhtlust inimeste vahel, kus ühel osapoolel on tugevad tehnilised teadmised ja teisel osapoolel ei ole. Cucumber raamistiku eeliseks on veel taaskasutatavus ja Gherkini süntaksireeglite omandamine on suhteliselt lihtne. Siiski, Cucumber lisab testimisse lisakihi, mis ei pruugi igas meeskonnas piisavalt lisaväärtust luua [13].

2.5 Analüüsi tulemused

Selenium WebDriveri kasutamine veebirakenduse kasutajaliidese testides on vältimatu, kuna see võimaldab brauseritele käske edastada. Selenium WebDriver võimaldab ilma vaheserverita otse brauseriga suhelda. Kuid Selenium ei paku hästi loetavat ülevaadet testitulemustest ning analüüse lihtsustavaid tegureid nagu kuvatõmmise tegemist ebaõnnestunud testi puhul. Seetõttu on parem kasutada mõnda raamistikku, mis kasutab Selenium WebDriverit. Selenide raamistik pakub eelistena automaatseid kuvatõmmised testi ebaõnnestumise puhul, sisseehitatud ootamisefunktsiooni ning lihtsasti loetavaid

HTML elementidega tehtavaid toiminguid. Automaatsed kuvatõmmised lihtsustavad ebaõnnestunud testi põhjuse analüüsimist, kuna on näha, mis hetkel test ebaõnnestus. HTML elementidega toimetamiseks testis saab kasutada sisseehitatud funktsioone või neid vajadusel juurde kirjutada.

JUnit raamistiku abil on võimalik muuta kood loetavamaks. Kasutades väiteid saab mugavalt kontrollida tingimuste korrektsust ning vale tulemuse puhul kuvatakse lihtsasti loetav raport. Raportis on välja toodud mis tulemust oodati ning mis oli tegelik tulemus. Annotatsioonidega saab märkida testmeetodeid, mida käivitamisel jooksutada. Kui test läbib ilma ühegi erindi viskamiseta, siis loetakse test õnnestunuks [14]. Lisaks saab ka märkida näiteks, milliseid meetodeid või klasse ignoreerida täielikult või ainult teatud tingimustes, luua meetodid mida käivitatakse enne ja pärast igat testmeetodit või piirata ajaliselt meetodi tegutsemisaega.

Cucumber raamistik võimaldab testistsenaariumeid kirjutada ka väheseid tehnilisi teadmisi omaval inimesel. Testi samme saab kirjeldada inglise keeles, kuid sammu reaalse täitmise peab kirjutama programmeerimiskeeles. Lisaks tuleb kohaneda ka Gherkin keele süntaksiga, milles kirjutatakse testistsenaariumid. See loob omakorda lisakihi testidele, mis võib küll lihtsustada testide loetavust ning kirjutamist, kuid nõuab ka testide arendajatelt lisaressurssi.

Töö autor otsustas oma töös kasutada Selenide ja JUnit raamistikke. Selenide pakub hulgaliselt eelpool nimetatud eeliseid Selenium WebDriveri üle ning JUnit aitab kaasa koodi loetavamaks muutmisel ning testitulemuste analüüside genereerimisel. Cucumber raamistikku otsustati mitte kasutada, kuna autor leidis, et see ei loo piisavalt lisaväärtust. Lisaks on töö autoril varasem kogemus antud raamistikega ja autori meeskonnas on mõlemad raamistikud juba kasutusel.

3 Sidusarenduse süsteemide analüüs

Uue ideega turule jõudmise aeg on kriitiline osa ettevõtte edust. Mida kiiremini saavad uued lahendused ideest reaalsuseks, seda tulusam on see ettevõttele. Uute lahendustega turule tulemine tõstab ettevõtte mainet konkurentide seas [15]. Vigase toote või tehnoloogiaga avalikkuse ette tulemine võib omada hoopis vastupidist toimet ning ettevõtte mainet ja kasumit kahjustada. Seetõttu on oluline, et kõik uued tehnoloogiad ja lahendused oleks enne lõppkasutajateni jõudmist võimalikult hästi testitud. IT ettevõtetes aitab arenduse testimise elutsüklile kaasa sidusarendus [16].

3.1 Sidusarendus

Ettevõttes on arenduste ülesehitus selline, et arendaja arendab vastavalt nõuetele mõne lisafunktsiooni või muudab midagi olemasolevas tootes või tehnoloogias. Enne kui muudatus ellu viiakse on seda vaja testida. Peale arenduse lõpetamist liigub see edasi testimisfaasi. Testimisfaasis võtab testija tehtud arenduse testimisse. Püstitatud testjuhtude puhul proovib testija leida vigu uues arenduses, et toodangusse läheks korrektselt teostatud lahendus. See nõuab vastavalt muudatustele ning selle suurusele testija aega.

Sidusarenduse eesmärk on pakkuda arendajale koheselt peale arenduse tegemist tagasisidet, kas kood läbis automaatteste. Projekti sidusarendusega juurutamise järel ei pea arendaja ootama, kuni arendus jõuab testimisfaasi, vaid saab koheselt tagasisidet peale muudatuse testkeskkonda üles laadimist. Testija seisukohast väheneb aeg, mida panustatakse käsitsi testimisele [16]. Sidusarendusega integreerimiseks on olemas palju erinevaid vahendeid. Populaarseimad on Jenkins ja Atlassian Bamboo [17].

3.2 Jenkins

Jenkins on avatud lähtekoodiga sidusarenduse vahend. Jenkinsi peamine ülesanne on automatiseerida koodi ehitamise ning testimise protsessi. Kuna Jenkins on avatud lähtekoodiga, siis sellesse saab panustada igaüks. Seetõttu on Jenkinsis olemas palju

erinevaid pistikprogramme, mis muudab Jenkinsi väga paindlikuks ning seda saab kohandada vastavalt oma vajadustele [15], [18].

Jenkins võimaldab automaatset testide käivitust juhul kui laetakse üles uued muudatused projekti repositooriumisse. Nende testide tagasiside on koheselt peale testide jooksutamist arendajale kättesaadav. Seeläbi saab arendaja väga varajases faasis juba esialgse ülevaate, kas uus arendatud lahendus on ellu viidud korrektselt. Vigade olemasolul saab arendaja koheselt sellega tegeleda [19].

3.3 Atlassian Bamboo

Atlassian Bamboo on tasuline sidusarenduse server, mis loob sidusarenduse töövoo (ingl k. *pipeline*) [20]. Bamboo pakub [21], [22]:

- automatiseeritud ehitamise ja testimise protsessi;
- teateid õnnestunud/ebaõnnestunud koodi ehitamiste kohta;
- statistilise analüüsi vahendeid;
- automaatset juurutust (ingl k. *deployment*) serverisse;
- statistilise analüüsi vahendeid.

Bamboo jaoks on vajalik projekti repositoorium, koodi ehituse skript ning testimiskomplekti. Bamboo ülesanne on planeerida ja koordineerida tervet tööprotsessi. Bamboo töötab plaani, töö ja ülesande põhimõttel. Plaanis määratakse ära erinevad faasid ning faas peab läbima kõik selles defineeritud tööd, enne kui saab jätkata järgmise faasiga. Tööd kontrollivad mis järjekorras ülesandeid täidetakse ning väljastatakse tulemused.

3.4 Analüüsi tulemused

Jenkinsi põhiliseks erinevuseks oma konkurentide seas on, et see on avatud lähtekoodiga. Kuna Jenkins on ka avatud lähtekoodiga, siis sellest tulenevalt on Jenkinsil hea ülemaailmne tugi erinevate Jenkins projekti arendajate ja toetajate poolt. Jenkinsit on lihtne paigaldada ning sellel on olemas tuhandeid pistikprogramme, mis lihtsustavad

Jenkinsi kasutamist. Kui juhtub, et kasutajale vajaminevat pistikprogrammi ei eksisteeri, siis on alati võimalus see ise kirjutada, kuna projekt on avatud lähtekoodiga [23].

Bamboo on erinevalt Jenkinsist tasuline vahend. Bambool on pühendatud meeskond, kes pakub tuge litsenseeritud kasutajatele. Bambool on võrreldes Jenkinsiga vähem pistikprogramme olemas, kuid tal on väga palju sisseehitatud integratsioone, funktsioone ja omadusi, nagu näiteks Jira tarkvaraga integratsioon [24]. Pühendunud tugimeeskond ja sisseehitatud integratsioonid muudavad üldise kogemuse kasutajasõbralikumaks.

Analüüsides Jenkinsit ja Atlassian Bamboo sidusarenduse vahendeid otsustas töö autor kasutada oma töös Jenkinsit. Kuna mõlema vahendi tööülesanne on sarnane, siis peamisteks põhjusteks on Jenkinsi avatud lähtekood, mis annab võimaluse kasutada juba olemasolevaid tuhandeid pistikprogramme või vajadusel luua ise pistikprogramm. Kuigi Bamboo tugimeeskond võib pakkuda konkreetsemat ning rohkem pühendunud tuge, siis Jenkinsi kommuuni poolt pakutav tugi on töö autori arvates piisav. Lisaks on ettevõttesiseselt juba Jenkins kasutusel ning on olemas meeskond, mis pakub tuge.

4 Testide arendamine

Järgnevas peatükis kirjeldab töö autor kasutajaliidese testide arendamise protsessi. Testide eesmärgiks on testida tellerirakenduses kõiki olemasolevaid laenutooteid Balti riikides.

4.1 Olemasolev lahendus

Hetkel on autori meeskonnas olemas Java keeles kirjutatud kasutajaliidese testid internetipanga jaoks ning Ruby keeles kirjutatud testid tellerirakenduse jaoks. Javas on kirjutatud valmis tellerirakenduse kasutajaliidese testi prototüüp. Eesmärgiks on arendada Javas kasutajaliidese testid kõikidele tellerirakenduses olevatele laenutoodete taotlustele.

Testide projekti uurides märkas töö autor, et Eesti keskkonnas tehtavad testid kasutavad rakenduses eesti keelt, kuid Läti ja Leedu keskkonnas tehtavad testid kasutavad rakenduses pakutavat inglisekeelset tõlget. Autor pakkus välja, et testid peaksid kasutama vastava keskkonna keelt, kus testid käivitatakse. Kui testid käivitatakse Eesti keskkonnas ehk testitakse näiteks Eesti väikelaenu taotlust, peaks olema keskkonna keeleks eesti keel ja kui teha taotlus Lätis, siis peaks kasutama läti keelt, mitte inglisekeelset tõlget. Sellise muudatuse tegemine muudab testid tõepärasemaks, kuna enamjaolt kasutatakse ikkagi kohalikku keelt taotluse tegemisel.

Seoses tehtud keelelise muudatusega on brauseris kuvatav tekst kohalikus keeles. Lisaks on ka projektis meetodite sisenditeks kasutatud kohalikku keelt, sest mõned Selenide raamistiku poolt pakutavad meetodid vajavad sisendiks HTML elemendi nime või teksti, mis on brauseris nähtav. Kuna ettevõttesiseseks asjaajamiskeeleks on inglise keel, siis ei ole hea kasutada koodis teist keelt. Teiste keelte kasutamine koodis võib põhjustada tarkvaraarendajas arusaamatust, kes ei räägi vastavat keelt.

Kuna brauseris kuvatavad tekstid tulevad üldiselt tõlgete andmebaasist, siis kasutas töö autor andmebaasi tõlkeid ära uue mikroteenuse loomisel. Autor töötas välja mikroteenuse, mis vastavalt keskkonnale pärib andmebaasist inglisekeelse tõlke kaudu vastava keskkonna keelelise tõlke. Seejärel asendati koodis kohad, kus oli kasutatud

kohalikku keelt, loodud mikroteenusega. Peale seda sai teste käivitada kohalikus keeles ning kood sisaldas vaid inglisekeelseid tõlkeid, muutes koodi hallatavamaks.

4.2 Testide kirjeldus ja ülesehitus

Enne iga testi algust käivitatakse meetod beforeTests (Joonis 3). Selleks kasutatakse JUniti poolt pakutavad annotatsiooni @Before, mis läheb käima enne igat meetodit, millel on annotatsioon @Test [25]. Meetodis sätestatakse komponendid, WebDriver ning initsialiseeritakse andmebaasiühendus.

```
@Before
public void beforeTests() {
  CurrentSession.setComponents(dbData, config, logger);
  setUpSelenideAndWebDriver();
  dbData.init()
}
```

Joonis 3. Enne igat testi käivitatav meetod.

Test käivitatakse meetodiga fillApplication (Joonis 4) ning sellel on peal annotatsioon @Test, mis annab märku, et tegemist on testmeetodiga. Selleks valitakse varasemalt üldkasutuseks loodud testklientide seast klient, kellele hakatakse laenutaotlust looma. Seejärel luuakse TestObjective klassi objekt, mis sisaldab endas kujutise (ingl k. *map*) muutujat. Kujutisse sisestatakse võti-väärtus paaridena andmed, mis peale taotluse täitmist peaks andmebaasis olema. Lisaks sisaldab ta infot millise laenutootega on tegemist.

```
@Test
public void fillApplication() {
  setupCustomer();
  TestObjective testObjective = getInput();
  fillApplicationInBrowser(product, testObjective);
  checkDatabase(testObjective);
}
```

Joonis 4. Testi käivitamise meetod.

Pärast andmebaasi nõuete kirjeldamist avatakse läbi meetodi fillApplicationInBrowser brauser, kasutades selleks Selenium WebDriverit läbi Selenide raamistiku. Brauseris avatakse vastavalt testiklassile laenutaotluse vorm (Joonis 5). Vormil täidetakse vastavalt nõuetele ära kõik vajalikud väljad ning seejärel esitatakse taotlus. Kui taotlusel esinevad dünaamilised arvutused, nagu näiteks kuumaksete arvutamised, siis kontrollitakse ka

nende õigesti arvutamist. Kuna arvutuse tulemused võivad ajas muutuda sõltuvalt intressidest ja muudest tootespetsiifilistest teguritest, siis ei kontrollita kindlat arvutatud tulemust. Kontrollitakse hoopis loogikat, et pikaajalisel laenul on kuumakse väiksem kui lühiajalisel laenul või kui jätta laenupikkus samaks, siis kontrollitakse, et kuumakse oleks suurema laenusumma puhul suurem kui väiksema laenusumma puhul.

```
protected void fillApplicationInBrowser(String product, TestObjective
testObjective) {
  openProductInBrowser(product);
  fillProductTab(testObjective);
  fillPersonalInfoTab(testObjective);
  fillObligationsTab(testObjective);
  fillIncomeTab(testObjective);
  sendApplication();
  checkPDF(getPDFRows());
}
```

Joonis 5. Testi poolt brauseris tehtavad toimingud.

Pärast taotluse saatmist kontrollitakse taotluse lepingu PDF faili genereerimist ning andmebaasi sisestatud väärtuseid. Sõltuvalt taotlustest kontrollitakse taotluse lepingu PDF failis vastavate sõnade, fraaside ja väljade olemasolu. Andmebaasi väärtuste kontrollimiseks kasutatakse meetodit checkDatabase (Joonis 4) ning sisendina eelnevalt loodud testObjective objekti. Testis vaadatakse, kas andmebaasi tabelis on olemas kirje koos TestObjective klassi kujutisse varasemalt sisestatud väärtusega. Juhul kui kirjet ei leita, loetakse test ebaõnnestunuks ning kuvatakse milliseid kirjeid ja milliseid väärtuseid ei leitud.

Järgnevas tabelis (Tabel 1) kirjeldatakse ära ühe tellerirakenduse kasutajaliidese testi sammud. Sõltuvalt testitavast laenutootest sõltub sammude arv. Testide kaetavuse eesmärgil on mõned testid laiendatud erijuhtumitega. Erijuhtumite all peetakse silmas seda, et ühele laenutootele lisatakse veel teine laen ehk võetakse korraga laenu mitmele tootele või taotletakse laenu koos kaastaotlejaga.

Tabel 1. Testi sammud.

Meetodi nimetus	**Meetodi kirjeldus**	**Testi erijuhtum**
Ava laenutaotluse leht	Avatakse brauseris vastavalt tootele laenutaotluse leht	Ei
Täida laenutoote vahekaart	Täidetakse laenutoote väljad vastavalt tootele, leht salvestatakse ning navigeeritakse järgmisele vahekaardile	Ei
Lisa teine laenutoode	Lisatakse taotlusele teine laenutoode	Jah
Täida isikuandmete vahekaart	Täidetakse isikuandmete väljad ning navigeeritakse järgmisele vahekaardile	Ei
Lisa kaastaotleja	Lisatakse tootele kaastaotleja	Jah
Täida kohustuste vahekaart	Täidetakse kohustuste väljad ning navigeeritakse järgmisele vahekaardile	Ei
Täida sissetulekute vahekaart	Täidetakse sissetulekute väljad ning navigeeritakse järgmisele vahekaardile	Ei
Taotluse saatmine	Taotluse saatmine	Ei
Lepingu PDF faili kontrollimine	Kontrollitakse PDF faili, et see oleks korrektselt täidetud	Ei
Andmebaasi kontrollid	Kontrollitakse andmebaasi sisestatud väärtuseid	Ei

4.3 Testide kirjutamine

Testide kirjutamise esimene eesmärk on luua kõikidele laenutoodetele MVP ehk minimaalne väärtuslik produkt. See tähendab, et kõik taotlusel olevad nõutud väljad on täidetud ning põhifunktsioonid on kontrollitud. Hilisemas arendusfaasis tegeletakse erijuhtude ning kaetavuse suurendamisega. Enne antud tööga alustamist oli olemas vaid algne prototüüp Eesti väikelaenu taotlusele.

4.3.1 Mustri loomine Balti riikide jaoks

Prototüübi lahendus täidab Eesti tellerirakenduse keskkonnas ära väikelaenu taotluse täites vaid kohustuslikud väljad, saadab taotluse ning kontrollib üksikuid andmebaasi väärtuseid. Prototüübi mustrit kasutades laiendas töö autor testi ka Läti ja Leedu keskkondadesse. Peamiselt kirjutati juurde laenutoote vahekaardi meetodeid, kuna ülejäänud vahekaardid (isikuandmete, kohustuste ja sissetuleku vahekaardid) on antud laenutootel suhteliselt sarnased igas riigis. Läti ja Leedu riikidesse laiendamisel tuli väga kasuks eelpool mainitud mikroteenus, mille kaudu sai vältida kohalikke keeli koodis.

Pärast väikelaenu taotluse laiendamist kõikidesse Balti riikidesse oli olemas muster, mida sai kasutada kõikide teiste laenutoodete kasutajaliidese testide arendamiseks. Iga laenutoote kohta on üks klass, mis testib vastavat toodet igas riigis. Iga laenutoote kohta loodi ka laenutoote vahekaardi klassi, milles sisaldus laenutoote vahekaardile sisestatava info andmed. Olid ka mõningad erandid, kus ei peetud uue klassi loomist vajalikuks, kuna ta sarnanes piisavalt mõne teise tootega. Selleks olid näiteks väikelaenutooted: väikelaen, auto väikelaen ja kodu väikelaen. Nende laenutoote vahekaardi täitmise sai ära kirjeldada ühises klassis. Iga ülejäänud vahekaardi kohta on vaid üks klass, kuna nende erinevused on väikesed ja need saab mugavalt eraldada tingimuslausetega.

4.3.2 Vahekaartide täitmine

Järgnevalt kirjeldatakse ära kodulaenu taotluse vahekaartide täitmine, et tekiks arusaam, kuidas kasutajaliidese testid toimivad ja kuidas need on üles ehitatud. Eelneval joonisel (Joonis 5) kirjeldatud meetod fillProductTab (Joonis 6) kutsub esile toote vahekaardi täitmise. See meetod koosneb vahekaardi täitmisest, selle salvestamisest ning järgmisele vahekaardile navigeerimisest. Esimese sammuna täidetakse laenutoote vahekaart. Selleks kasutatakse vastavalt laenutootele loodud klassi objekti ning kutsutakse välja meetod setValuesOnPage. Sisendiks võtab ta enne taotluse täitmist loodud klassi TestObjective tüüpi parameetri. Testis kasutatakse objekti pageCommon, mis sisaldab endas väga palju Selenide raamistiku poolt pakutavaid meetodeid ning ettevõttes juurde kirjutatud meetodeid. Nende meetodite abil saab Selenide raamistiku abil edastada brauserile käske.

```
protected void fillProductTab(TestObjective testObjective) {
  homeLoanApplicationProductTab.setValuesOnPage(testObjective);
  homeLoanApplicationProductTab.saveTab();
  homeLoanApplicationProductTab.navigateToNextTab();
}
```

Joonis 6. Laenutoote vahekaardi täitmine.

Järgneval joonisel (Joonis 7) on näha, et kasutades pageCommon objekti valitakse laenu eesmärgi rippmenüüst esimene valik juhul kui rippmenüü eksisteerib. Kasutatav meetod on autori loodud, kasutades Selenide poolt pakutavad meetodeid. Selle meetodi eesmärgiks on vähendada tingimuslauseid, mis tekiks Balti riikide laenutoodete erinevustega. Lisaks on kood ka loetavam. Järgnevalt sisestatakse laenusumma, kuid siin on HTML elemendi erinevus vastavalt riigile. Läti riigis on elemendi nimetus natuke erinev kui Leedus ja Eestis. Seetõttu ei saa kasutada sarnast loogikat nagu rippmenüüde puhul, kuna igas riigis eksisteerivad mõlemad elemendid HTML dokumendis, kuid üks kahest elemendist on mõeldud teiseks otstarbeks. Seetõttu tuleb eraldada välja täitmine tingimuslausega, mis kontrollib keskkonda, kus test käivitati. Järgnevalt sisestatakse kinnisvara hind, omafinantseering ning seejärel laenuperiood.

```
public void setValuesOnPage(TestObjective testObjective) {
  pageCommon.selectFirstDropdownValueIfExists(„loanPurpose“);
  pageCommon.selectFirstDropdownValueIfExists(„purpose“);
  pageCommon.selectFirstDropdownValueIfExists(„purposeSubCategory“);

  if (TestsConfig.isLV()) {
    pageCommon.enterTextByName(„limitNew“,
testObjective.getInputValue(„limitNew));
}
  else {
    pageCommon.enterTextByName(„limit“,
testObjective.getInputValue(„limit“));
}
  pageCommon.enterTextByNameIfExists(„sellingPrice“,
testObjective.getInputValue(„sellingPrice“));
  pageCommon.enterTextByNameIfExists(„propertyPrice“,
testObjective.getInputValue(„propertyPrice“));
  pageCommon.enterTextByNameIfExists(„selfFinancing“,
testObjective.getInputValue(„selfFinancing“));

  pageCommon.selectDropdownValueByDrowdownName(„loanPeriodYear“,
testObjective.getInputValue(„loanPeriodYear“));
  pageCommon.selectDropdownValueByDrowdownName(„loanPeriodMonth“,
testObjective.getInputValue(„loanPeriodMonth“));
}
```

Joonis 7. Vahekaardi täitmise meetod.

Kõik sisestatavad väärtused saadakse kätte kasutades testObjective objekti. Kuna enne taotluse täitmist sisestatakse testObjective objekti kujutisse andmebaasi väärtused, mida hiljem kontrollitakse, siis selle sama kujutise kaudu päritakse vastav väärtus ning seda kasutatakse andmete sisestamiseks. Selle eesmärgiks on muuta kood hallatavamaks, kuna vahel on vaja muuta väärtuseid, mida sisestatakse. Sellise lahenduse puhul on vaja muuta vaid kujutisse sisestatavaid väärtuseid ning need sisestatakse ka taotlusele. See väldib hooletusvigade tekkimist, kuna kui muuta ainult kujutisse sisestatavaid väärtuseid, mida hiljem andmebaasist kontrollitakse ning mitte muuta taotlusesse sisestatavaid väärtuseid, siis tekib andmebaasi väärtuste kontrollimisel viga.

Ülejäänud vahekaartide täitmine käib sarnaselt laenutoote vahekaardi täitmisele, kus kasutatakse pageCommon objekti, et sisestada vahekaardile erinevaid väärtuseid. Seejärel salvestatakse vahekaart ning navigeeritakse edasi. Testide erijuhtude puhul on osade vahekaartide täitmise meetodite vahele kirjutatud tingimuselaused. Näiteks kodulaenu taotlusele lisatakse isikuandmete vahekaardil ka kaastaotleja, kus pärast vahekaardi täitmist lisatakse info kaastaotleja kohta. See protsess on ära kirjeldatud järgneval joonisel (Joonis 8).

```
protected void fillPersonalInfoTab(TestObjective testObjective) {
  applicationPersonalInfoTab.setValuesOnPage(testObjective);
  if (testObjective.getProduct().equals(LoanProduct.HOME_LOAN)) {
    applicationPersonalInfoTab.setCoBorrowerValues(testObjective);
    applicationPersonalInfoTab.saveTab();
  }
  applicationPersonalInfoTab.saveTab();
  applicationPersonalInfoTab.navigateToNextTab();
}
```

Joonis 8. Kaastaotleja lisamine kodulaenu taotlusele.

Peale kõikide vahekaartide täitmist kontrollitakse genereeritud lepingu PDF faili olemasolu ning seal asuvaid väärtuseid. Kontrollimiseks on loodud loend, mis sisaldab endas erinevaid sõnesid, mida otsitakse PDF failis sisalduvate sõnede seast. Peale PDF faili kontrollimist teostatakse andmebaasi kontrollid, kus kontrollitakse, kas taotlusesse sisestatud väärtused on korrektselt jõudnud andmebaasi. Selleks kasutatakse testObjective objektis sisalduvat kujutist, mis sisaldab endas andmebaasi väärtuseid, mis seal eksisteerima peavad. Juhul kui mõni väärtus on puudu või väärtus erineb defineeritud väärtusest, siis kogutakse kokku kõik puuduvad väärtused ning loetakse test ebaõnnestunuks ja kuvatakse veateade andmebaasi väärtuste erinevusest.

4.4 Testide lokaalne käivitamine ning testide tulemused

Testide lokaalseks käivitamiseks on neli võimalust:

1) testklassi käivitamiseks, mis käivitab kõik klassis asuvad testmeetodid, tuleb teha IDE-s paremklikk testklassi peal ning valida võimalus „*Run*“;

2) testmeetodi käivitamiseks vajutada IDE-s meetodi kõrval asuvat „*Run Test*“ nuppu;

3) paralleelseks testide jooksutamiseks tuleb IDE-s teha paremklikk testkausta peal ning valida käsk „*Run*“ või sätestada ja kasutada IDE-s sisseehitatud võimalust Compound;

4) kasutades Gradle kasutajaliidest valida Gradle ülesanne ning kaasa anda keskkond keskkonnamuutujana.

Iga testmeetodi kohta antakse testi lõppedes tagasiside. Tagasisidest on näha, kas test õnnestus või ebaõnnestus. Ebaõnnestumise puhul kuvatakse veateade, mis kirjeldab, mille tõttu test ebaõnnestus.

5 Sidusarenduse juurutamise protsess

Arendatud kasutajaliidese testid vajavad Selenium WebDriverit. Kui lokaalselt saab WebDriveri failile viidata süsteemi omaduse (ingl k. *system property*) kaudu, siis Jenkinsis nii teha ei saa, kuna sidusarenduse analüüsi peatükis valitud meetodis kasutatakse Dockeri konteinerit, mis peale testide läbimist hävitatakse. Seega WebDriveril ei ole kindlat asukohta vastavas Jenkins alluva (ingl k. *Jenkins slave*) masinas.

5.1 Sidusarendusega integreerimise analüüs

Järgnevates peatükkides analüüsitakse erinevaid võimalusi kasutajaliidese testide integreerimiseks sidusarenduse vahendiga. Võimalikult hea ja efektiivne sidusarendusega integreerimise lahendus peaks vastama järgmistele nõuetele:

- lahendus peab olema kiire;
- lahendus on võimalikult palju isoleeritud teistest arendusmeeskondadest;
- lahendust on võimalik kohandada igal ajal vastavalt vajadusele;
- lahendus peaks sisaldama võimalikult vähe manuaalset tööd (nt WebDriveri uuendamine peaks toimuma automaatselt).

Nende nõuete põhjal uuriti erinevaid võimalusi nii ettevõttesiseselt kui ka ettevõtteväliselt.

5.1.1 Ettevõttesisesed lahendused

Ettevõttesiseste lahendustega tutvumiseks konsulteeriti osakonna erinevate tarkvaraarendajatega. Osakonnasisesel koosolekul tõstatas töö autor ka teema, et kuidas on teised meeskonnad integreerinud oma kasutajaliidese testid sidusarenduse vahendiga. Vastustest tuli välja, et paljud meeskonnad ei ole integreerinud oma kasutajaliidese teste sidusarenduse vahendiga või neil esineb probleeme WebDriveri haldamisega või sidusarendusega integreerimisega. Sellest tulenevalt otsustas töö autor korraldada eraldi

osakonnasisese koosoleku, kus arutati sidusarenduse vahendiga integreerimise võimalusi ning probleeme sellega seoses.

Enne koosoleku toimumist suhtles töö autor erinevate meeskondade tarkvaraarendajatega ning leiti kaks kolleegi, kes on oma meeskonna kasutajaliidese testid integreerinud Jenkinsiga. Koosolekul demonstreeriti erinevates meeskondades toimivaid lahendusi, mis kasutasid WebDriverManager teegi poolt pakutavat brauser Dockeris lahendust.

Meeskonnasiseste tarkvaraarendajatega suheldes mõeldi välja lahendus, et paigaldada WebDriver Jenkins alluva masinasse. See võimaldaks viidata WebDriverile samamoodi nagu lokaalsel testide jooksutamisel ehk süsteemi omaduse kaudu. Sellele lahendusele esitati koosolekul põhjendatud vastuargumendid. Peamine argument oli, et selline lahendus tähendaks, et keegi vastutab ja haldab vastavat lahendust. Probleem tekiks siis, kui vastutav isik lahkub ettevõttest või ei ole kättesaadav. Lisaks ei ole erinevate meeskondade vajadused samasugused, mis muudaks antud lahenduse haldamise keeruliseks. Need põhjused näitavad selgelt, miks vältida ühiseid lahendusi teatud probleemidele ning parem on olla teistest meeskondadest isoleeritud.

5.1.2 Selenium Grid

Selenium Grid on üks osa Selenium testimistarkvara komplektist. See töötab jaotur-alluv tööpõhimõttel, kus testid käivitatakse Selenium jaoturis (ingl k. *hub*), kuid testid sooritatakse erinevatel alluvatel. Selenium Gridi peamine eesmärk on toetada testide paralleelset jooksutamist erinevate masinate, operatsioonisüsteemide ja brauserite peal [26].

Selenium Hub serveri peab ise konfigureerima. Ühe võrgustiku (ingl k. *grid*) kohta peaks olema vaid üks jaotur. Jaotur on server, mis jagab ülesanded alluvatele paralleelselt [27]. Kui lokaalsel käivitamisel sai WebDriveri asukohale viidata süsteemi omaduse kaudu, siis Selenium Gridi kasutamisel tuleb määrata RemoteWebDriver. RemoteWebDriverile saab anda sisse erinevaid parameetreid, sõltuvalt, millist brauserit või operatsioonisüsteemi soovitakse kasutada. Kindlasti tuleb parameetrina sisse anda Selenium Hubi URL [26].

5.1.3 WebDriverManager

WebDriverManager ehk WDM on teek WebDriveri manageerimiseks. Selle abil saab näiteks leida paigaldatud WebDriveri asukoha, sätestada kasutamiseks WebDriverit, luua WebDriveri instantsi ning salvestada brauseris toimuvat. WebDriverManager versioon 5 võimaldab kasutada brauserit läbi Dockeri tõmmise (ingl k. *Docker image*) [28].

Docker tõmmise kasutamise jaoks peab olema masinasse paigaldatud Docker. Selle kaudu tõmbab enne iga testi käivitamist WDM alla Dockeri tõmmise, millega käivitatakse Dockeri konteiner. Dockeri tõmmiseid pakub ja haldab Aerokube [28]. WDM konfiguratsiooni kaudu saab sätestada, millist brauserit ning millist brauseri versiooni kasutada. Allatõmmatud Dockeri tõmmis sisaldab endas vastavalt konfiguratsioonile brauserit ja selle brauseri WebDriverit. Selle abil saab luua ka WebDriveri instantsi ning kasutada seda testide jooksutamiseks. Testi lõppedes hävitatakse eelnevalt loodud Dockeri konteiner. WDM teegi kasutamiseks on see vaja lisada oma projekti sõltuvusena. Sõltuvuste lisamine käib näiteks läbi Gradle või Maveni [28].

5.1.4 Analüüsi tulemus

Peale ettevõttesiseste lahenduste uurimist ning teiste võimalike lahenduste analüüsimist otsustas töö autor kasutada WebDriverManager teeki, mis oli ka varasemalt ettevõttes kasutusel. Esialgu ei pooldanud töö autor seda lahendust, kuna tundus, et enne iga testi käivitamist Dockeri tõmmise allalaadimine on ajakulukas. Autor pooldas pigem Jenkinsi poolt kasutatavasse masinasse WebDriveri paigaldamise lahendust. Peale korraldatud koosolekut mõisteti, et selline lahendus ei ole kõige parem, kuna selle haldamine sisaldaks manuaalset tööd ning teiste meeskondadega kooskõlastamist.

Ajakulukuse mõttes analüüsiti kui kaua võtab aega Dockeri tõmmise allalaadimine enne testi algust. Uuriti vastava meeskonna Jenkinsi tööd, kellel oli see lahendus juba ellu viidud. Töö logidest oli näha, et Dockeri tõmmise allalaadimise ajakulu oli vähem kui üks sekund. Seega tegelikult suur ajakulu ei ole selle lahenduse puhul asjakohane.

Selenium Gridi lahendus tähendaks jaoturserveri ning alluvate konfigureerimist. Selenium Grid on mõeldud kasutamiseks erinevatel tarkvaradel, operatsioonisüsteemidel ning brauseritel. Kuna töö raames arendatud testide puhul ei ole oluline neid testida erinevates keskkondades, siis ei tundunud see lahendus mõistlik.

WebDriverManager teegi kasutamine täidab kõik alampeatükis 5.1 defineeritud nõuded. Eelnevalt kirjeldatud analüüsi tulemustel veenduti, et lahendus on kiire. Enne igat testi luuakse uus Dockeri konteiner ning testi lõpus see hävitatakse. Konteiner luuakse allatõmmatud Dockeri tõmmisest. Seetõttu ollakse teistest meeskondadest võimalikult eraldiseisvad – ainus sõltuvus on Aerokube poolt pakutavad Dockeri tõmmised. Kasutatav WebDriveri ja brauseri versioon kirjeldatakse ära WDM konfiguratsioonis ning seda kasutatakse vastava tõmmise alla laadimiseks. Seeläbi ei pea tegema manuaalset tööd WebDriveri uuendamiseks.

5.2 Lahendus

Järgnevas peatükis kirjeldatakse kõikide kasutajaliidese testide integreerimist sidusarenduse vahendiga. Sõltuvalt sidusarenduse analüüsi tulemustest ja sidusarenduse juurutamise protsessi analüüsi tulemustest, otsustati kasutada Jenkinsit ning integreeritakse kasutades WDM teeki ning Dockerit. Jenkinsiga sidumiseks on vaja luua koodis Jenkinsfile, mille sees on ära kirjeldatud Jenkinsi töövoog. Jenkinsi kaudu jooksutamiseks kasutatakse Gradle ülesandeid (ingl k. *Gradle task*).

5.2.1 Jenkinsiga integreerimine

Kui lokaalsel käivitamisel kasutatakse WebDriverile viitamiseks süsteemi omaduse võimalust, siis Jenkinsis jooksutamisel seda teha ei saa. Jenkinsi töö käivitatakse ühe Jenkins alluva masina peal, mis tähendab, et seal masinas ei ole WebDriver paigaldatud. Selle jaoks on vaja eristada koodis, kas testid käivitati lokaalselt või testid käivitati Jenkinsi kaudu.

Arusaamaks, kas testid käivitati Jenkinsi poolt, kasutatakse teiste arendajate poolt loodud meetodit isJenkinsExecution, mis on toodud välja järgneval joonisel (Joonis 9).

```
public static boolean isJenkinsExecution() {
  return System.getnenv().containsKey(„JENKINS_URL“)
    || System.getenv().containsKey(„BUILD_URL“);
}
```

Joonis 9. Jenkinsi käivituse tuvastamine.

Meetod kontrollib läbi System klassi, kas keskkond sisaldab võtmeväärtust JENKINS_URL või BUILD_URL. Kui see vastab tõele, siis järelikult on tegemist Jenkinsi poolt käivitatud testidega.

Jenkinsi käivitusel luuakse meetodis getWebDriverForJenkins WebDriver objekt (Joonis 10). Kasutades WebDriverManager teeki luuakse uus Chrome tüüpi WebDriver ehk ChromeDriver ning meetodiga browserInDocker täpsustatakse ära, et kasutatakse Dockeris asuvat WebDriverit. See tähendab, et tõmmatakse alla Dockeri tõmmis, kus on vastavalt meetodis browserVersion(„99“) parameetrina kaasa antud brauseri ja WebDriveri versioon. Tõmmise abil käivitatakse Dockeri konteiner. Pärast WebDriveri loomist sätestatakse see, kasutades Selenide poolt pakutavat klassi WebDriverRunneri meetodit setWebDriver (Joonis 11). Kui süsteemi omadust kasutades sulgeb Selenide ise testi lõppedes brauseri, siis meetodit setWebDriver kasutades peab arendaja vastutama brauseri sulgumise eest [29].

```
private WebDriver getWebDriverForJenkins(){
  return WebDriverManager
       .chromedriver()
       .browserInDocker()
       .browserVersion(„99“)
       .create();
}
```

Joonis 10. WebDriveri loomine Jenkinsi jaoks.

Kui WebDriver on sätestatud, siis tehakse kindlaks, kas sätestamine toimus korrektselt, maksimeerides avatud brauseri aken. Kui WebDriver pole initsialiseeritud, visatakse erind NullPointerException. Erind püütakse kinni ning visatakse uus erind, mis annab märku, et WebDriver pole initsialiseeritud ehk selle loomisega läks midagi valesti. Juhul kui ei ole Jenkinsi töö kaudu testide käivitamine, siis sätestatakse WebDriver süsteemi omaduse kaudu (Joonis 11).

```
if (isJenkinsExecution()) {
  WebDriver driver = getWebDriverForJenkins();
  WebDriverRunner.setWebDriver(driver);
  try {
    driver.manage().window().maximize();
  }
  catch (NullPointerException e){
    throw new RunTimeException(„WebDriver not initiated“);
  }
} else {
  System.setProperty(„webdriver.chrome.driver“, DRIVER_LOCATION);
}
```

Joonis 11. WebDriveri sätestamine lokaalsel ja Jenkinsi käivitusel.

Projekti Jenkinsiga sidumiseks luuakse Jenkinsi keskkonnas uus Jenkinsi töö ning konfigureeritakse, millist repositooriumit see kasutab. Koodis on vaja luua Jenkinsfile, kus on võimalik kirjeldada Jenkinsi töö eripärad [30]. Jenkinsfile'is kirjeldatakse ära:

- milliseid Gradle'i ülesandeid käivitada Jenkinsi töö käivitamisel;
- millistel kellaaegadel automaatselt ehitada kood ning käivitada Gradle ülesanded testide käivitamiseks;
- millistele e-mailidele saata testide tulemuste raport;
- keskkonnamuutujad.

Edaspidiseks Jenkinsi töö modifitseerimiseks on vajalik muuta ainult loodud Jenkinsfile'i. Jenkinsi keskkonnas konfigureerimist teha ei ole vaja.

5.2.2 Probleemid Jenkinsiga integreerimisel

Peale Jenkinsiga integreerimist tekkisid erinevad probleemid mille tõttu ei saanud teste jooksutada Jenkinsi kaudu. Esimeseks probleemiks oli liigselt turvatud võrguühendus, mille tõttu ei saanud Jenkinsi masin ühendust testitava rakendusega. Selle probleemi lahenduseks suheldi ettevõtte võrguühenduse- ja turvalisuse meeskondadega. Erinevate mahukate arutelude tulemusena sai see probleem lahendatud.

Testide paralleelsel jooksutamisel tekkis olukord, kus testid valisid ühe ja sama testkliendi taotluse tegemiseks. Ühele kliendile samaaegne taotluse loomine tekitas erinevaid tõrkeid. Selle probleemi lahendamiseks muudeti testkliendi valimise päringut. Pärast testkliendi valimist lisati nende identifikaator eraldi tabelisse, mille kaudu klient reserveeriti ja järgnevatel testkliendi päringutel keelati sama testklienti valida. Reserveering toimub ajalise piiranguga ning teatud aja möödudes saab testklienti uuesti valida.

6 Tulemuste analüüs

Olemasolevad Ruby keeles kirjutatud kasutajaliidese testid on raskesti hallatavad. Lisaks teised autori meeskonna rakenduse kasutajaliidese testid on kirjutatud Java keeles. Töö raames arendati Java keeles kasutajaliidese testid kõikidele tellerirakenduses olevatele laenutoodetele. Testid kontrollivad laenutaotluse põhifunktsionaalsust kõikides Balti riikides. Lisaks põhifunktsionaalsusele on lisatud ka mõned testide erijuhud nagu näiteks kaastaotleja lisamine taotlusele. Arendatud testklasse on kokku 18, millest 6 on vaid erijuhtumeid testivad klassid.

Vigaselt ellu viidud arendus tekitab arendajatele lisatööd, kahjustab ettevõtte mainet ning võib kaasa tuua finantsilisi kahjusid. Seetõttu on korrektselt ja põhjalikult testitud kood äärmiselt oluline, et olla edukas ettevõte. Töö raames arendatud tellerirakenduse kasutajaliidese testid loovad võimaluse avastada vigu varases arenduse faasis. Nii on väiksem tõenäosus, et vead jõuavad lõppkasutajateni ning vea parandamiseks kulub vähem ressurssi. Kasutajaliidese testid on ka äärmiselt oluline osa regressioonitestimise ajal, mis toimub töö autori meeskonnas regulaarselt.

Töö raames integreeriti kõik autori meeskonna kasutajaliidese testid ka sidusarenduse vahendiga Jenkins. Jenkinsi abil käivitatakse iga tööpäeva hommikul kõik kasutajaliidese testid. See annab ülevaate eelmisel päeval loodud muudatuste edukusest. Tulemuste põhjal saavad arendajad pidevat tagasisidet, kas muudatused on edukalt ellu viidud.

Jenkins vähendab ka manuaalset tööd, mis varasemalt kulus testide käivitamise ja tulemuste analüüsi peale. Varasemalt mitme kasutajaliidese testi ühisosa muutes pidi arendaja jooksutama kõiki teste kõigis kolmes Balti riigis veendumaks, et mõni test katki ei läinud. Umbkaudu 150 testi käsitsi käivitamine ning tulemuste analüüsimine võttis ligikaudu viis tundi aega. Jenkinsis saab kõiki teste jooksutada lihtsalt Jenkinsi töö käivitamisega. Testide käivitamine võtab aega ligikaudu üks tund ning tulemused koos veateadetega kuvatakse väga mugaval ja lihtsasti loetaval viisil. Seega testide käivitamise ning tulemuste analüüsimise aeg vähenes viis korda.

7 Edasine arendus

Kasutajaliidese teste on võimalik veel palju täiendada ja edasi arendada. Peamine koht kus täiendada saab, on testide kaetavuse suurendamine. Selleks saab näiteks suurendada andmebaasis toimuvate kontrollide hulka, juurde kirjutada erijuhtumite teste, laiendada kontrolle, mis tehakse taotluse vahekaartidel, laiendada lepingu PDF failis tehtavaid kontrolle või luua integratsioonitestid tellerirakenduse ja mõne teise teenuse või rakenduse vahel. Lisaks on laenutoodete taotluse protsess pidevas arenduses, see tähendab, et ka teste peab vastavalt korrigeerima.

Jenkinsi töö, mille käigus jooksutatakse kasutajaliidese teste, käivitatakse hetkel igal tööpäeva hommikul ning juhul, kui laetakse uus kood üles testide repositooriumisse. Testitava rakenduse repositooriumisse koodi üles laadimine arendajate poolt ei käivita hetkel kasutajaliidese testide Jenkinsi tööd. Edasisel arendusel tuleks see töö käivitada kui rakenduse arenduse repositooriumisse kood üles laetakse, kuid hetkel on arendajad rahul ka hommikul saadud testitulemustega.

Pikaajaliseks eesmärgiks on muuta testid piisavalt põhjalikuks, et ei oleks üldse vajadust käsitsi testida. Selle saavutamisel oleks võimalik kohe pärast arenduse tegemist käivitada testid ning kui need ei leia ühtegi viga, saab tehtud arenduse koheselt toodangukeskkonda liita. Sedasi saaks vältida käsitsi testimist regressioonitestide ajal.

8 Kokkuvõte

Töö eesmärgiks oli vähendada aega, mis kulub käsitsi testimisele. Selle saavutamiseks seati eesmärk luua Java keeles kasutajaliidese testid ettevõttesisesele tellerirakendusele ning need ja olemasolevad testid integreerida sidusarenduse vahendiga. Arendatud testid pidid olema kiired ning pakkuma selgelt loetavaid testitulemusi. Lisaks pidid testid olema hallatavamad, kui olemasolevad Ruby keeles kirjutatud tellerirakenduse kasutajaliidese testid.

Testimisraamistike analüüsi peatükis valiti kasutamiseks raamistikud Selenide ja JUnit. Selenide raamistiku abil oli võimalik muuta kood hallatavamaks, kuna tema sisseehitatud meetodid on hästi loetavad. Lisaks sellele, teeb Selenide ka ebaõnnestunud testi puhul kuvatõmmise, mis annab lisainfot testitulemuste analüüsimisel ning vea välja selgitamisel. JUnit raamistiku abil oli võimalik muuta testitulemused selgelt loetavaks. Väidete abil kuvatud veateated annavad selge ülevaate, mis tulemust oodati ning mis oli tegelik tulemus. JUniti annotatsioonid muutsid koodi hallatavamaks ja taaskasutatavamaks.

Sidusarenduse vahendite analüüsi tulemusena valiti kasutamiseks Jenkins. Jenkinsi abil on manuaalselt võimalik käivitada kõiki teste väga kerge vaevaga ning seda tehakse ka automaatselt igal tööpäeva hommikul. Jenkinsiga integratsioon vähendas aega, mis kulus käsitsi testide käivitamisele ja tulemuste analüüsimiseks, viielt tunnilt ühele tunnile ehk lausa viis korda. Kuna Jenkinsi abil on võimalik vähendada aega, mis kulub testimisele, siis töö põhieesmärk on täidetud. Viis korda vähenenud aeg testide jooksutamiseks täidab kiirete testide eesmärgi ning Selenide ja JUnit pakuvad selgelt loetavaid testitulemusi.

Kasutatud kirjandus

[1] S. Dhingra, „Introduction to Selenium and its Components,“ QA Tech Hub, 2017. [Võrgumaterjal]. Available: https://qatechhub.com/introduction-to-selenium-components/#:~:text=Selenium%20WebDriver%3A,with%20RC%2C%20WebDriver%20was%20developed.. [Kasutatud 17 Märts 2022].

[2] P. Ramya, V. Sindhura ja P. V. Sagar, „Testing using Selenium Web Driver,“ %1 *2017 Second International Conference on Electrical, Computer and Communication Technologies*, Coimbatore, 2017.

[3] Codeborne, „Selenide FAQ,“ Codeborne, [Võrgumaterjal]. Available: https://selenide.org/faq.html. [Kasutatud 7 Aprill 2022].

[4] Codeborne, „Selenide and Selenium comparison,“ Codeborne, [Võrgumaterjal]. Available: https://selenide.org/documentation/selenide-vs-selenium.html. [Kasutatud 10 Aprill 2022].

[5] Codeborne, „Documentation,“ Codeborne, [Võrgumaterjal]. Available: https://selenide.org/documentation.html. [Kasutatud 2 Aprill 2022].

[6] I. Gaba, „Simplilearn,“ Simplilearn, 28 Oktoober 2021. [Võrgumaterjal]. Available: https://www.simplilearn.com/tutorials/java-tutorial/what-is-junit. [Kasutatud 8 Aprill 2022].

[7] S. D, „Software Testing Help,“ Software Testing Help, 3 Aprill 2022. [Võrgumaterjal]. Available: https://www.softwaretestinghelp.com/junit-tutorial/. [Kasutatud 8 Aprill 2022].

[8] M. Olan, „Unit testing: test early, test often,“ *Journal of Computing Sciences in Colleges*, kd. 19, pp. 319-328, 2003.

[9] S. Bechtold, S. Brannen, J. Link, M. Merdes, J. de Rancourt, J. d. Rancourt ja C. Stein, „JUnit 5 User Guide,“ 12 Aprill 2021. [Võrgumaterjal]. Available: https://junit.org/junit5/docs/current/user-guide/#writing-tests. [Kasutatud 8 Aprill 2022].

[10] „JUnit 5 Javadoc,“ JUnit, [Võrgumaterjal]. Available: https://junit.org/junit5/docs/current/api/. [Kasutatud 8 Aprill 2022].

[11] J. Unadkat, „Introduction to Cucumber Testing Framework,“ 5 Mai 2021. [Võrgumaterjal]. Available: https://www.browserstack.com/guide/learn-about-cucumber-testing-tool. [Kasutatud 16 Aprill 2022].

[12] T. Hamilton, „Cucumber Framework: What is Cucumber Testing Tool?,“ Guru99, 16 Veebruar 2022. [Võrgumaterjal]. Available: https://www.guru99.com/introduction-to-cucumber.html#how-bdd-works-in-cucumber-automation. [Kasutatud 16 Aprill 2022].

[13] K. V. Belle, „Get started with Behaviour-Driven Development (BDD),“ b.ignited, [Võrgumaterjal]. Available: https://bignited.be/2020/01/28/bdd-tests-with-cucumber-and-gherkin/. [Kasutatud 16 Aprill 2022].

[14] JUnit, [Võrgumaterjal]. Available: https://junit.org/junit4/javadoc/4.12/org/junit/Test.html. [Kasutatud 16 Aprill 2022].

[15] V. Armenise, „Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery,“ %1 *2015 IEEE/ACM 3rd International Workshop on Release Engineering*, Florence, 2015.

[16] M. Shanin, M. A. Babar ja L. Zhu, „Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,“ *IEEE Access*, kd. V, pp. 3909-3910, 2017.

[17] Katalon, „DevOps and CI/CD,“ Katalon, [Võrgumaterjal]. Available: https://katalon.com/resources-center/blog/ci-cd-tools. [Kasutatud 22 Märts 2022].

[18] Jenkins, „Jenkins User Documentation,“ [Võrgumaterjal]. Available: https://www.jenkins.io/doc/. [Kasutatud 22 Märts 2022].

[19] T. Munetsi, „What Is Jenkins Used For?,“ OpenLogic, 6 Märts 2020. [Võrgumaterjal]. Available: https://www.openlogic.com/blog/what-is-jenkins-used-for#:~:text=Jenkins%20is%20used%20to%20build,alongside%20other%20cloud%20native%20tools.. [Kasutatud 16 Aprill 2022].

[20] Atlassian, „Pricing,“ Atlassian, [Võrgumaterjal]. Available: https://www.atlassian.com/software/bamboo/pricing. [Kasutatud 13 Aprill 2022].

[21] Atlassian, „Understanding the Bamboo CI Server,“ 18 Juuli 2021. [Võrgumaterjal]. Available: https://confluence.atlassian.com/bamboo/understanding-the-bamboo-ci-server-289277285.html. [Kasutatud 16 Aprill 2022].

[22] Atlassian, „Using Bamboo,“ Atlassian, 24 Juuni 2020. [Võrgumaterjal]. Available: https://confluence.atlassian.com/bamboo/using-bamboo-289276852.html. [Kasutatud 16 Aprill 2022].

[23] S. Kulshrestha, „What is Jenkins?,“ Edureka, 28 Märts 2022. [Võrgumaterjal]. Available: https://www.edureka.co/blog/what-is-jenkins/. [Kasutatud 16 Aprill 2022].

[24] S. Kappagantula, „Jenkins vs Bamboo – Battle Of The Best CI/CD Tools,“ Edureka, 27 November 2019. [Võrgumaterjal]. Available: https://www.edureka.co/blog/jenkins-vs-bamboo. [Kasutatud 16 Aprill 2022].

[25] JUnit, „JUnit Javadoc,“ JUnit, [Võrgumaterjal]. Available: https://junit.org/junit4/javadoc/4.12/org/junit/package-summary.html. [Kasutatud 15 Aprill 2022].

[26] K. Rungta, „Selenium Grid Tutorial: Hub & Node (with Example),“ Guru99, 12 Märts 2022. [Võrgumaterjal]. Available: https://www.guru99.com/introduction-to-selenium-grid.html. [Kasutatud 10 Aprill 2022].

[27] G. Tiwari, „Selenium Grid Tutorial : How to Set It Up,“ 2 Mai 2021. [Võrgumaterjal]. Available: https://www.browserstack.com/guide/selenium-grid-tutorial#:~:text=What%20is%20Selenium%20Grid%3F,to%20multiple%20registered%20Grid%20nodes.. [Kasutatud 16 Aprill 2022].

[28] B. García, „WebDriverManager,“ 28 Juuli 2021. [Võrgumaterjal]. Available: https://bonigarcia.dev/webdrivermanager/. [Kasutatud 10 Aprill 2022].

[29] Codeborne, „WebDriverRunner Javadoc,“ Codeborne, [Võrgumaterjal]. Available: https://selenide.org/javadoc/current/com/codeborne/selenide/WebDriverRunner.html. [Kasutatud 16 Aprill 2022].

[30] Jenkins, „Using a Jenkinsfile,“ [Võrgumaterjal]. Available: https://www.jenkins.io/doc/book/pipeline/jenkinsfile/#handling-credentials. [Kasutatud 6 Aprill 2022].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Karl-Joosep Kesküla

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Kasutajaliidese testide arendamine koos sidusarenduse protsessi juurutamisega finantsasutuse näitel“, mille juhendajad on Brita Moorus ja Merje Einla.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

16.05.2022

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.