TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Software Science

IT40LT

Roman Radionov 134204IAPB

POINT OF SALE SYSTEM'S TEST AUTOMATION USING SELENIUM AND A FOLLOWING ANALYSIS OF AUTOMATION BENEFITS

Bachelor's thesis

Supervisor: Deniss Kumlander
PhD

Tallinn 2017

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

IT40LT

Roman Radionov 134204IAPB

KASSASÜSTEEMI TESTIDE AUTOMATISEERIMINE KASUTADES SELENIUMI NING EDASPIDINE ANALÜÜS AUTOMATISEERIMISE KASUMIST

Bakalaureusetöö

Juhendaja: Deniss Kumlander
PhD

Tallinn 2017

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: [Roman Radionov]

[25.04.2017]

Abstract

The main aim of this thesis consists of two parts.

The first part is to cover the application with test cases and automate them. The application under test is a web application, which is a point of sale system written in PHP with the use of framework Laravel. For test automation framework Selenium was chosen. Chosen testing types: functional, system and regression testing.

The aim of the second part is to do an analysis based on the first and practical part of this thesis. Author will analyse all the costs from manual testing and test automation throughout the project. Cost will be measured as time. The result of this analysis will be a conclusion, whether it is worth to invest resources in test automation in such type of projects, taking into consideration not only time but all the advantages, disadvantages and risks that come with test automation of such applications.

In this way author will be able to combine the practical part, experience and skills that were developed through study in Tallinn University of Technology to write the code and analytical part, analysis which will be build upon the written code and made work.

The final result of this thesis will be an automated web application and a conclusion, whether it is worth to invest into test automation of such kind of projects or not.

This thesis is written in English language and is 44 pages long, including 10 chapters, 16 figures and 0 tables.

Annotatsioon

Selle lõputöö põhisuund koosneb kahest osast.

Esimeses osas kaetakse rakendus testilugudega ja testid automatiseeritakse. Testitavaks rakenduseks on veebirakendus mis kujutab ennast müügisüsteemi ja on kirjutatud PHP keeles kasutades Laravel raamistiku. Testide automatiseerimiseks valiti Selenium raamistik. Testi tüüpideks on funktsionaalne, süsteem ja regressioon testimine.

Teise osa suunaks on teha esimese ja praktilise lõputöö osa peale analüüs. Autor analüüsib manuaalse testimise ja testi automatiseerimise maksumuse projekti jooksul. Maksumust mõõdetakse ajas. Analüüsi tulemuseks on järeldus, kas on väärt investeerida testi automatiseerimisse selliste tüüpi projektides, võttes arvesse mitte ainult aja, vaid kõik eelised, puudused ja riskid, mis kaasnevad sellist tüüpi projekti testide automatiseerimisega.

Sellisel juhul autoril tekib võimalus antud lõputöö kirjutamisel kombineerida praktilist osa – kogemusi ja oskusi koodi kirjutamiseks mis olid saadud Tallinna Tehnikaülikoolist, ning analüütilist osa, mille analüüs baseerub eelnevalt kirjutatud koodil ning tehtud tööl.

Antud lõputöö lõpptulemuseks on automatiseeritud veebirakendus ning otsus, kas investeerimine sarnaste projektidesse testide automatiseerimiseks tasub ennast ära või mitte.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 44 leheküljel, 10 peatükki, 16 joonist ja 0 tabelit.

List of abbreviations and terms

AUT	Application under test
QA	Quality assurance
POS	Point of sale
XML	Extensible markup language
API	Application programming interface
URL	Uniform resource locator
JSON	JavaScript Object Notation
UI	User interface
DOM	Document Object Model
VGT	Visual graphic user interface testing
CSS	Cascading style sheets
XPath	XML Path language
PHP	Hypertext Preprocessor, scripting language
jQuery	JavaScript library
AJAX	Asynchronous JavaScript and XML. Technique used to send and retrieve data in the background.
W3C	World Wide Web Consortium – an international community that develops web standards
KPI	Key performance indicator
IE	Internet Explorer
OS	Operating system
IDE	Integrated development environment
KanbanFlow	Lean project management tool

Table of Contents

List of Figures

1 Introduction

1.1 Overview

Testing is an important part of developing a successful and high quality product. Nowadays with rapid evolving software tools, development methods and complexity of applications in general it is sometimes hard to decide, whether it is worth to invest into test automation or just keep testing manually.

Automation indeed consumes more resources than manual testing and thus, taking into consideration that test automation is more expensive and comes with risks, manual testing is taken in use over automation, but is this decision made correctly? Yes, test automation is more expensive, however, it makes testing much faster and eliminates so called human error in which case rises the quality of the end product. There are a lot of more positive and negative sides of test automation but information about those and a closer look will be taken further in the thesis.

1.2 Practical part

Throughout the development process of this project author will be dealing with manual testing as well as with test automation. First of all test cases of the first version will be written. Afterwards these test cases will be tested manually. When manual testing is done, necessary tests will be automated using Selenium WebDriver. This process will continue with applied regression testing until the last development cycle. Spent time will be measured in two flows. First flow is manual testing and manual regression testing. Second flow is time spent on automation and management of these tests. Functional, system and regression testing types will be applied. All the documented test cases as well as the source code of automated tests will be attached to the Appendix. For a more comfortable code understanding each package in the Eclipse (or any other IDE) project will represent functionality of a certain compartment, for example, package

transactions will contain test cases that are located in the section transactions under the "Test idea" column of the table which is attached to the Appendix 1.

Additionally, in this thesis author will be speaking about testing tools selection, how was test automation executed, about interesting aspects and problems that might occur in automation of such kind of projects.

1.3 Specification of automation problem

In most cases an experienced project manager or analytic is able to judge more or less precisely, whether automation is beneficial or not. The aim of this thesis was a little deeper than just picking a random application to be automated. The application was specially chosen by analysing which, even for an experienced person it would be hard to tell whether it is worth to implement automation or not. In such cases the decision, to take in use manual testing over automation is made. Following criteria [1] indicated that automation should not be used.

1. Insufficient time – The testing team might have a feeling that there is insufficient time to look for alternatives to manual testing and learn how to build and maintain scripts. This is why test automation was implemented parallelly. Time consumed on the learning process was recorded to make a conclusion whether such approach is sensible and possible in such conditions.

2. Cost – An organization might not own all the necessary tools for comfortable automation nor wish to invest into them. In this thesis free open source tools will be taken in use. It will be proven that in the first sight not comfortable tools can be used with ease and fully replace software that costs. In such way additional expenses on expensive tools will be avoided.

3. Job security – Testers might feel threatened by automation as they are experienced and used to manual testing.

Author will try to prove that it is possible to adapt to such circumstances and overcome the fear of implementing automation.

2 Application under test

2.1 General description

The web application to which test automation is implemented is a point of sale (POS) system. POS is a system with rich functionality, which provides the user with a better experience in maintaining his business, offering a variety of possibilities. A few examples:

- Remotely monitor sales information about all the goods just in one place.
- Create accounts for the employees to track their activity.
- Established link between the kitchen and the waiter/waitress for a more efficient and faster service of clients.
- Easily check for the quantity in the storage and time when to replenish stocks.
- Measure employee effectiveness by getting precise statistics on who exactly and how much had sold.
- Fixate all the deals.
- Print checks after transactions are done.
- Get statistics of sold goods in order to rise profit.
- Digitally monitor the cash flow.

2.2 Technologies and requirements

As it was mentioned before, this is a web application. Back end was written in PHP by using Laravel framework, frontend in JavaScript and jQuery libraries. POS UI design

example on figure 1 and 2. The main non functional requirements were that this application must be reachable remotely and the main browser on which this application should be working perfectly has to be Google Chrome. Synergy with other browsers was not necessary in the set requirements, although this aspect might change in the future.

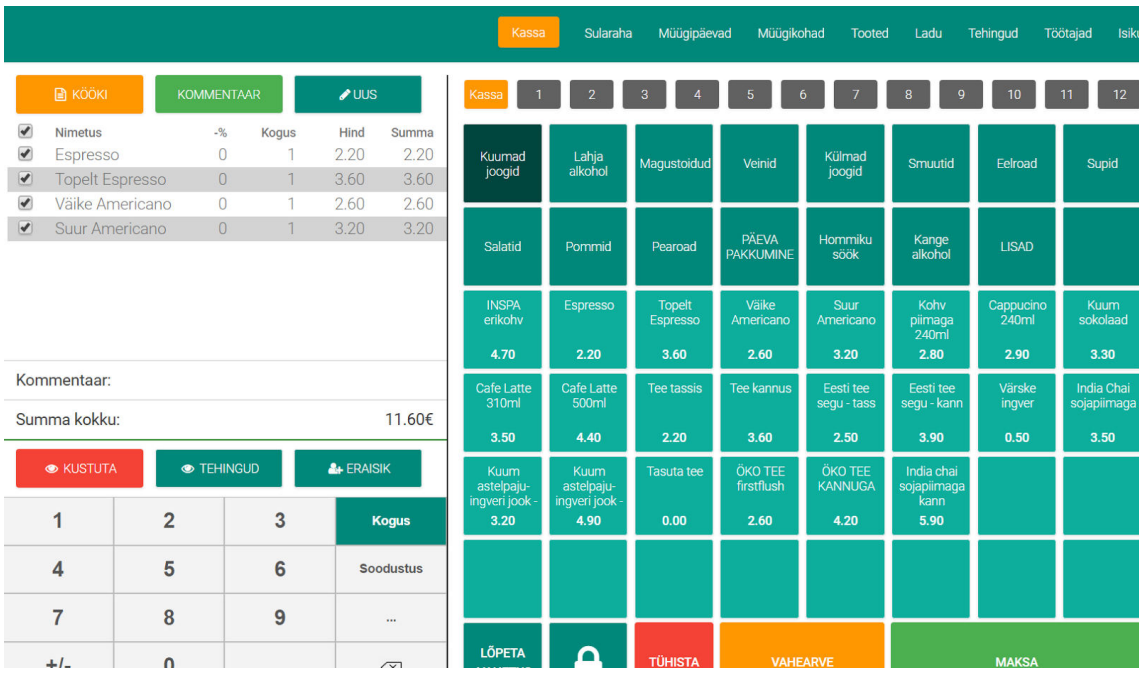


Figure 1. POS main desktop

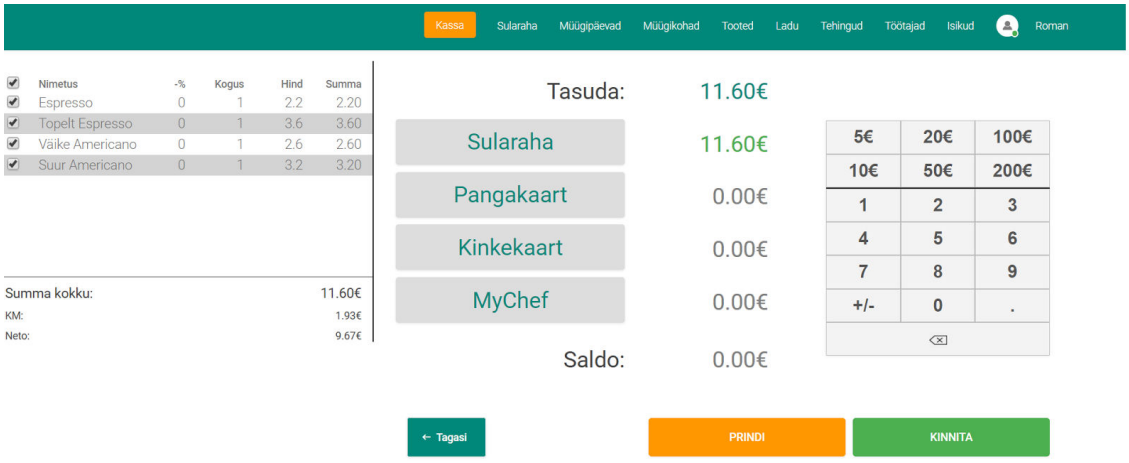


Figure 2. Payment page

3 Test automation

The precondition of high quality automated tests is a well documented list of important test cases and manual execution. Information about manual testing will be provided further in paragraphs 8.1 and 8.2, since the focus in the early stages of the written part of this thesis is on test automation.

Test automation is a process of executing tests that are usually done manually, automatically by using software tools specially designed for test automation. The main idea is to compare two outcomes, predicted and actual. By choosing the right strategy, sticking to the main principles of test automation ("The test automation manifesto" [2]) and by taking into consideration all the nuances of the AUT, it is possible to benefit a lot from automating tests that would usually be executed manually.

3.1 Benefits and drawbacks

As it was stated before if test automation is done correctly it brings a lot of benefit to the project and the business overall, however, there are some disadvantages and risks that come with test automation, if not handled correctly they might turn into a disaster [3] .

Benefits

- Business expenses. Despite the fact that test automation requires more investments in the beginning, after a certain number of development cycles depending on the project a positive ROI [4] will be seen. Moreover, not only it will become positive, it will also become greater and greater compared to ROI of manual testing as time passes.

- Shortens the development time because a fast feedback is provided.

- Rises the quality of the end product by eliminating so called human error. It is well known that tests that are repeated a lot of times manually tend to be skipped or tested poorly from time to time [5] .
- Helps testing projects that are basically impossible to be tested completely by using manual testing. For example if cross-browsing and cross-platform testing is necessary.

Drawbacks and risks

- Test automation tools cost. Not all tools are free to use.
- False sense of security. There is always a possibility that even though all the tests executed are green/passed the system is working incorrectly.
- Tests might get unreliable and bring unwanted expenses as the project develops. With a frequently changing user interface if a test is written not professionally it will fail. In this case test management is required which brings additional expenses.

4 Tools selection

By choosing the correct tool four criteria that were set by the author were taken into account. First of all the tool should have a strong compatibility with web applications. Secondly, it must be aimed at system, functional and regression testing since taking into consideration how and by whom this application will be used security, load, stress and performance testing is not necessary and will only bring additional expenses. Thirdly, it must be free to solve the problem in paragraph 1.3 under number 2. Finally, good environment to write code.

There are software testing tools that allow recording certain actions. However, this approach is at the moment not a standard, since tests that are recorded might get unstable and fail as soon as certain changes will be implemented. There is a possibility that in the near future these techniques will be implemented but at the moment more development and researches must be made in order to solve the limitations of VGT [6] . As an alternative VGT can be combined with scripting but such approach will not be examined in this thesis as the AUT has to be with a very stable visual appearance [7] .

4.1 Selenium WebDriver

Selenium [8] is undoubtedly one of the most popular testing suits. This is a free open source project which was released under Apache 2.0 license [9] . Interesting fact, the name Selenium comes from a joke made by Huggins. He was looking for an alternative to Mercury Interactive QuickTest Professional testing software and an idea that selenium mineral supplements cure mercury poisoning came to his mind. For this reason the software is called Selenium [10] . The most important component of Selenium, in our case, is the Selenium WebDriver [11] .

Selenium WebDriver is a framework which is specially made to interact with the browser by using special commands such as: close or open browser, click, send data

into a field, find an element, go to a certain URL and much more. In other words, it allows the developer to mimic human(user) behaviour. All the commands are sent to the browser through JSON Wire Protocol. Selenium WebDriver architecture on figure 3 [12] .

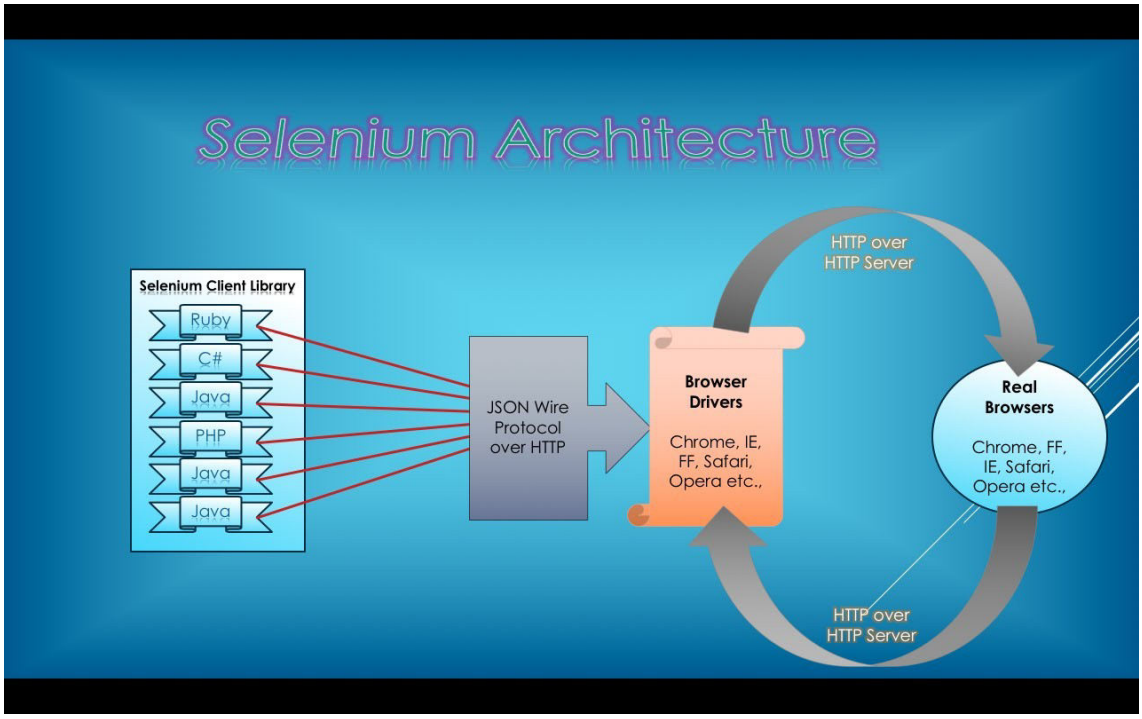


Figure 3. Selenium WebDriver architecture

Another big reason in choosing Selenium WebDriver is the universality. WebDriver supports a big variety of different browsers, operating systems and programming languages.

- Browsers: Chrome, Firefox, Safari, Opera, Internet Explorer, HTMLUnit.
- Operating systems: Windows, Apple OS, Linux.
- Programming languages: C#, Java, JavaScript, Python, Ruby. These five are supported by the main project hosted on google chrome. It means that it is recommended to use these languages to avoid bugs and lack of support. Although, it is possible to use other languages such as, PHP and Perl.

4.2 Alternatives to Selenium WebDriver

1. **PhantomJS** [13] – is also a powerful tool for testing web applications. Its main advantage is fast execution of test cases as it does not take any additional time to launch a specific browser, because PhantomJS itself is a browser that exists in a script. Another dominance over Selenium WebDriver is that it doesn't require much effort in setting up the environment and no third party services are needed. There are a few reasons why it stays in the alternative section but the most important one is that if there is a chance that cross browsing might be implemented it is not worth to use PhantomJS over Selenium WebDriver as it does not support this feature. It is worth to mention that it is a good practice to use Selenium with PhantomJS in some cases, but it is out of scope of this thesis.

2. **CODED UI** [14] – supported by Microsoft Development Network this is another alternative to Selenium WebDriver. Released in 2010 and till now it is a powerful tool to automate Windows Applications as well as Web Applications in which case beats Selenium. It does have a strong element identification mechanism and does support AJAX controls. But for the same reason as PhantomJS is stays in the alternative list. This tool only supports some versions of IE (8, 9, 10). Although, it is possible to set up a cross browsing environment with additional framework, but there will be some serious limitations in other browsers when trying to reuse written scripts for IE. In other words, Selenium is a better choice if it comes to cross browsing.

3. **Cypress** [15] – the start of the development of this tool began in 2015. Right now it is in a private beta. Despite the fact, that is is a very young tool compared with many others, it has the potential to be the next successor of Selenium as it is incredibly comfortable to work with, it has no preconditions of setting up an environment as it comes as one application, user friendly API and has all the necessary tools built in.

Deeper comparison of tools is out of scope of this thesis, however, it is worth to mention about them, since they might be easily applied if requirements were slightly

changed. Furthermore, Selenium is approved for its standards by W3C [16] , in such way any other tool might be considered not as a standard solution.

4.3 Selenide

In this thesis it was decided not to take in use any additional libraries in order to get a better understanding how to interact with application that uses a lot of AJAX and manually manipulate waiting time. By succeeding this thesis's goals by using only Selenium WebDriver will only make the summary and conclusion stronger and more reliable.

Although, it is worth to mention that a wrapper called Selenide exists. It actually makes writing scripts much easier, since it helps to handle different kinds of exceptions, problems that come with AJAX and a lot of other small aspects of web application testing. Another big and important advantage of Selenide is that it makes the script cleaner and easier to read.

4.4 TestNG

Selenium suit does not include a testing framework so it had to be implemented from outside. Java languages was used for scripting, thus a choice between two popular testing frameworks was available, TestNG and Junit.

In popularity JUnit overcomes TestNG but in the most part the reason is that JUnit is included by default in a lot of Maven archetypes. These two frameworks both match requirements for this project, despite the fact that they have different methods, annotations and approaches in some cases, overall they are pretty much the same. In this thesis TestNG was chosen for the purpose of new experience because JUnit was used by author in the past.

5 Setting the development environment

Selenium Client and language bindings.

In selenium to make it possible to write scripts in a certain language, in our case in Java, Selenium Java Client Driver and language bindings are required. This is a simple process and all that needs to be done is to download these jars from the official Selenium web page and configure them through the Java Build Path. This will provide Selenium with all the necessary tools to create WebDriver scripts.

Driver Server

In order to use Selenium WebDriver on browsers besides HTMLUnit and Firefox a third party plugin(driver) is required. For example to run tests in Chrome, ChromeDriver is essential, for Opera OperaDriver, for InternetExplorer IEDriverServer. This is an executable that starts a server and reserves a port on the machine. It basically controlls the browser from the OS level. It is the second layer that can be found on the figure 3. It is worth mentioning that Firefox is a special case. By using Selenium version 3.0 and above with Firefox version 47 and below no Driver Server is needed, however, for versions 47 and above GeckoDriver is essential. There are two possible ways to look up for the Driver Server when initializing the WebDriver either by *System.setProperty("driverName", "Path")* or by including the location of the executable in the PATH environment variable. In this thesis automated tests will be executed in Chrome as it is the most popular browser and covers all the requirements of this web application. However, theoretical implementation of cross-browsing testing together with code examples will be included further in this thesis.

6 Chosen testing strategy and types

Test automation as mentioned above might bring a lot of benefit to the development and to the business overall but only if the strategy is chosen correctly. After analysing the whole idea of how and by whom in particular this application will be used it was decided that the optimal and most crucial testing that is necessary in the development cycle of this product is system, functional and regression testing. These types of testing are best suited to start executing tests right from the start of the development and with high frequency of repetitivity.

It was very important to write all the tests such way that they would be resistant to frequently changing user interface. Author's aim was to get as much benefit from test automation as possible and in order to do so, besides choosing the necessary testing types it was also essential to pinpoint following tests cases, as it is impossible and not rationally to automate everything [3] .

- Cases/functionality with high risk of failure due to complexity or frequently changing web application design.
- Functionality that tends to fail in the future which is caused by human error.
- Tests that simply take too much time and effort to be tested manually.

6.1 Black Box

For a better and faster progression of project development fast feedback of bugs and functionality that isn't working properly after any other implementation is necessary, thus black box method was chosen. Author's main goal was to test the web application from the perspective of the user and to get information of functionality failure as soon as possible after any changes in the existing code were made or new code was added. This type of method is perfectly suited for upper levels of testing such as functional and

system testing. Moreover there was no need to deal with PHP because in black box testing knowledge of internal structure of this web application was not necessary.

This brings us to another benefits. In other words, it means that even a person without knowledge of a specific language on which the AUT is written will be able to write or maintain these automated tests. This mean that it will be easier to find such person and the amount of people who will be able to take on this job will be bigger. In my case all the necessary coding was done in Java. Comparison of black box testing and white box testing on figure 4 [17] .

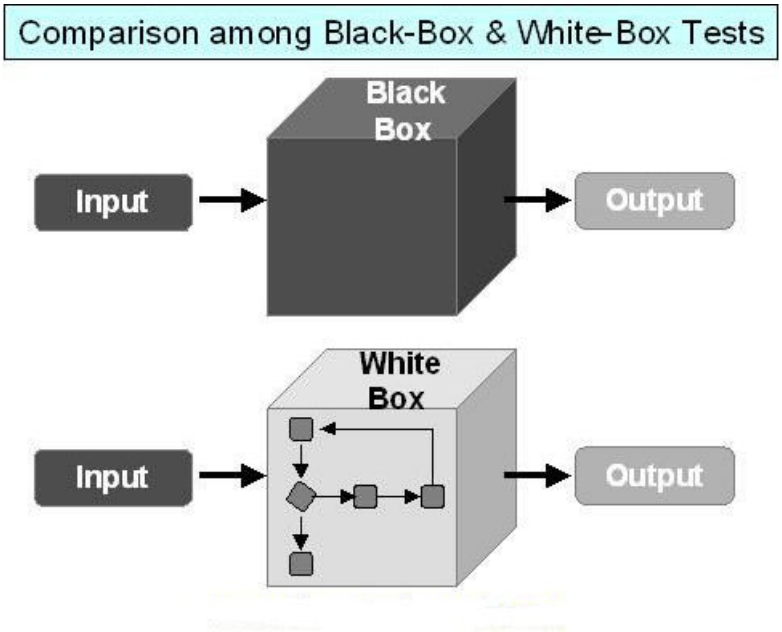


Figure 4. White box vs black box

6.2 Functional testing

Functional testing is a sub type of black box testing and the most important one in this thesis, since it took the most amount of time and heavily influences system testing. In functional testing a bigger piece of functionality than just one function is being tested. Important is to mention that functional testing is the type where regression testing is crucial and most beneficial. It is not necessary to analyse the code and how exactly the

input is being processed. However, it is important to know what the output is going to be after a certain input. Simple structure of functional testing is shown on figure 5 [18] .

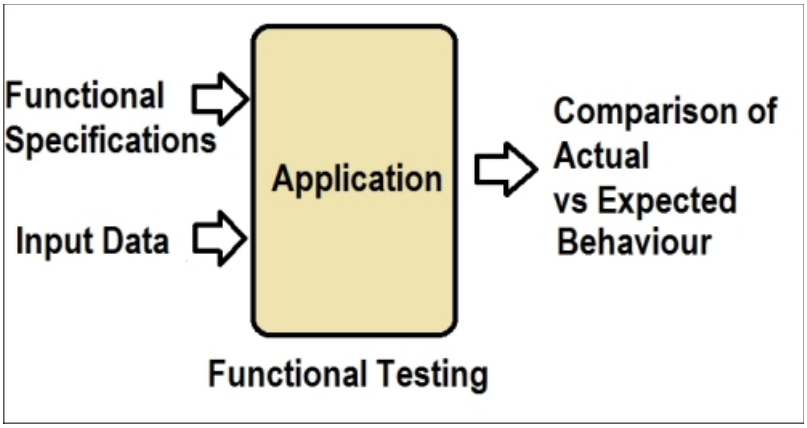


Figure 5. Functional testing architecture

6.3 System testing

Usually system testing in waterfall methodology comes almost last, before acceptance test, however, my practical part was made in agile development environment thus system testing was made more frequently. This is why functional and system testing in this project is proportionally related. The sooner functional testing gives positive results the sooner system testing will be completed.

6.4 Regression testing

Regression testing main idea is to confirm that after any changes to the software were made, previously developed functionality still works correctly. It is a very important type of testing, especially in an agile development environment. Test automation implemented on this type brings the most benefit as it usually has to be done a lot of times over and over in which case by automating these tests human resources can be used elsewhere. Regression bugs are a common phenomenon, especially in the early stages of development as a big amount of enhancements and new features in a short

period of time are added. Missing such bugs might lead to additional expenses in the future development phases.

6.5 Cross-browser

Cross-browser testing is an inalienable part of the whole testing process if it has to be implemented. It displays the ability of a web application to adapt to any browser that was set in the requirements of a project. Cases where the application has to work properly in more than two browsers might get very time consuming to test. There might even be a possibility of testing this application on multiple different platforms which also drastically increases the overall development time. Also the AUT in different versions of a browser might behave differently which means that sometimes it is needed to test not only different browsers but also different versions of browsers. If a project must be working on different platforms, in different browsers and with different browser versions at the same time it might get impossible to cover all these requirements with manual testing. To sum it up, if the AUT is automated to execute tests in such environments it brings enormous benefits in the form of saved time and helps to assure a better quality of the end product.

7 Automation process overview

As the automation process began it was decided to start automating less complex test cases with simple functionality and as isolated as possible from other components.

Sometimes functionality that is rarely used, isolated and small might make an impression that it will always work and it is not worth being tested and especially automated. However, in reality it is not always like that and in this project there was such an example. In the earlier stages there was no problem in adding staff to the database through the application, however, after certain changes were made to the database, deleting a staff member and then trying to add him/her with same initials was impossible. This bug was instantly detected through an automated test when it was executed for the second time, since it always adds a staff person with the same name and as it was executed for the second time while trying to add a person with the same name this bug was detected.

After completing isolated test cases more complex test cases were written. These complex test cases consisted of multiple components in such way checking the application on a bigger scale. Not only these tests checked for actual results but they also gave a more clear vision on how successful system testing might be in the present moment.

7.1 LogInClass

First and the most important class which was created is the *LogInClass*. It had two main purposes that were located in the *loginMethod()*. First one is to initialize the ChromeDriver as it is shown below:

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--start-maximized");

driver = new ChromeDriver(options);
```

Figure 6. Initializing ChromeDriver

ChromeOptions is a class that can be used to set certain options to the created instance of the browser by passing the ChromeOptions to the drivers constructor. By adding "-start-maximized" argument to the options the browser starts in full size, which helps to avoid false failed tests created by UI design. Of course it is possible to set specific window size if there is such need, however, in our case maximum size was fine.

The second purpose was to store information of two accounts. The first account was used to log in into the system. The second account was used to bypass the basic authentication.

```
//Basic authentication
driver.get("https://username:password@example.ee");

//System log in credentials
wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("email")))
        .sendKeys("roman.radionov@example.ee");
wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("password"))).sendKeys("password");
wait.until(ExpectedConditions.visibilityOfElementLocated(By.className("btn-primary"))).click();
```

Figure 7. Basic authentication and log in credentials

Elements can be found on the page by the following available to WebDriver locators: by name, id, XPath, class name, tag name, link text, partial link and CSS. Choosing the right locator, especially in projects with frequently changing functionality and design is very important. It will influence the speed of locating an element, how often these test cases will have to be maintained and how reliable they will be. The best practice is to use the id locator. Referring to the W3C standards it is unique and thus by changing its location there will be no problem in finding this specific element. Sometimes, for example in case of a bad written code or the architecture, there might be no possibility to use an id locator and then the second level of locators come into play where it is possible to locate the element either by CSS or name. The worst practice is to use

Xpath, since it is very vulnerable to location change and takes more time to look up for the element.

7.2 TestNG class and TestNG XML

TestNG class

All the main code and test logic of a single test case is executed in this class. First of all a new instance of *LogInClass* is initialized *LogInClass login = new LogInClass()* and three annotations are created: @BeforeMethod, @Test and @AfterMethod for the following purposes: @BeforeMethod starts first and this is where login.loginMethod() is executed. In such way as each test starts, the whole login process is done and the user is in the system. After @BeforeMethod comes @Test where all the main test logic is written. In case the code in this annotation fails or completes @AfterMethod is executed where ChromeDriver servers process is terminated: *login.getDriver().quit().* Although it is important to know, if a running test is terminated by the user manually in the IDE the server process will not be canceled and will hang in the windows processes.

TestNG XML

This XML file was used to execute test suites as TestNG does not offer a possibility to define a suite inside the source code of the TestNG class. In our case the XML has a mandatory attribute *name* to define the suite, attribute *parallel* where it is defined whether multiple threads should be running, multiple test and class *names* and a parameter annotation with attributes *name* and *value*. All the test cases that should be executed at once and the conditions how it should be done are all set in this one XML. An example of the XML will be shown in the paragraph 7.5.3.

7.3 Time optimization

The most important part was time handling and optimization. Elements on the page in certain conditions take longer time to be displayed, for example module loading through Javascript/JQuery. In this case Selenium throws an exception as the element is invisible. To avoid this exception a method that waits for the Selenium to find this element is

necessary. Method *Thread.sleep()* is a bad practice because it offers us to wait for a fixed amount of time. This brings us to a row of negative effects and the most important one of them is waste of time. If we, for example, set the thread to sleep for 5 seconds and the element could be found in 3, we waste time. And in one test suit there might be thousands of such moments which brings us to a colossal time loss. If we set the thread to sleep for 3 seconds and the element appears only after 4, the whole test fails. To avoid all these problems class WebDriverWait(Explicit Wait) was used. This class takes as argument two values, first one the browser driver and second one the maximum amount of time that the driver will be waiting for the element to meet certain conditions. In the *LogInClass* an instance of WebDriverWait class was initialized as follows: *wait = WebDriverWait(driver, 10)*. This means that every time an instance of the *LogInClass* is created in the TestNG class, the wait method is available and can be used throughout the lifetime of the driver.

This class has an *until* method with prebuilt expected conditions. By standard this method checks for the element every 500 milliseconds but it can be customized if necessary. It returns a Boolean True if an element meets the condition. List of available and used conditions:

1. Wait for the presence of an element. Used in case it is not needed to check whether the element is visible or not and just check for its presence in the DOM: *wait.until(ExpectedConditions.presenceOfElementLocated(By locator));*

2. Wait for the element to be clickable. Used to locate an element that is visible and enabled: *wait.until(ExpectedConditions.elementToBeClickable(By locator));*

3. Wait for an element to get visible. Checks for the visibility of an element: *wait.until(ExpectedConditions.visibilityOfElementLocated(By locator));*

4. Wait for the element to get invisible. Return Boolean True if the element is not visible or not present in the DOM: *wait.until(ExpectedConditions.invisibilityOfElementLocated(By locator));*

This wait method can be used not only for the purpose of getting any kind of information from the element but also to avoid problems created by Javascript. As the automation process began another problem occurred. When a module was being closed by jQuery an element is not clickable exception was thrown. The reason to such exceptions is that the element is actually clickable and visible, however, an overlay/spinner is on top of it. A simple solution was found with use of the condition that waits for the invisibility of an element. This method was used to wait for an element with a unique locator in the module to disappear and only then look for the other element.

7.4 Implementing cross browser testing

The point of this paragraph is to show how easy it is to implement cross browsing into the project. Tests in other browsers were not analyzed and measured, since in the requirements there was only one browser (Google Chrome) where this application must be working properly. These requirements might change in the future and for this reason a tool with strong cross browsing possibilities was chosen.

Testing additional browser types is out of scope in this thesis as it would be a waste of time. Although, if the requirements change it will be very beneficial, because optimizing one test case for another browser takes around 1/10 of the time taken to write the script. Setting the environment for cross browsing also does not take a big amount of time and will be shown further in this thesis. As an example Firefox was taken in use.

7.4.1 LogInClass changes

First of all *loginMethod()* that is located in the *LogInClass* and which is responsible for driver initialization and the account login process is changed. Changes: an argument *String browser* is passed to the method. Two *if* statements are added for each browser that check for the argument that was passed to the method. If the value is *chrome* it launches the piece of code where chrome driver is initialized, if it is *firefox* that was passed as the argument it launched the code where GeckoDriver is initialized.

Additionally a class variable *browserName* is declared. This variable's value is assigned when a certain driver is created. If it is GeckoDriver we assigne the value *firefox* , if it is ChromeDriver - *chrome*. The purpose of this variable will be explained in the next header. For a better understanding figure number 8 is attached.

```
public class LogInClass {

    private WebDriver driver;
    public WebDriverWait wait;
    private String browserName;
    private String userName = "Roman Radionov";

    public void loginMethod(String browser) {
        if (browser.equals("chrome")) {
1
            browserName = "chrome";
            ChromeOptions options = new ChromeOptions();
            options.addArguments("--start-maximized");

            driver = new ChromeDriver(options);
            wait = new WebDriverWait(driver, 10);

            driver.get("https://username:password@example.ee");

            wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("email")))
                    .sendKeys("roman.radionov@example.ee");
            wait.until(ExpectedConditions.visibilityOfElementLocated(By.name("password"))).sendKeys("password");
            wait.until(ExpectedConditions.visibilityOfElementLocated(By.className("btn-primary"))).click();
        }

        if (browser.equals("firefox")) {

2           browserName = "firefox";
            System.setProperty("webdriver.gecko.driver", "C:\\Utility\\BrowserDrivers\\geckodriver.exe");
            driver = new FirefoxDriver();
            wait = new WebDriverWait(driver, 10);
            driver.manage().window().maximize();
```

Figure 8. LogInClass changes

7.4.2 TestNG class changes

In every TestNG class two mandatory things must be done. Firstly, a parameter annotation is added as follows: *@Parameters("browserType")*. Secondly, changes in the @BeforeMethod annotation:

```
@BeforeMethod
public void beforeMethod(String browserType) {
    if (browserType.equalsIgnoreCase("chrome")) {
        login.loginMethod("chrome");
    }
    if (browserType.equalsIgnoreCase("firefox")) {
        login.loginMethod("firefox");
    }
}
```

Figure 9. @BeforeMethod changes

The *if* statement checks for the *browserType* parameter and compares it with the TestNG XML parameter value and depending on the value a certain *loginMethod()* is executed.

Now an explanation how the *browserName* variable in the *LogInClass* is being used. If a piece of code does not work in one browser and works in another and there is no possible way to make it work in two browser by using the same piece of code, *if* statements are used. The *if* statement checks for the *browserName* variable and depending on this variable the code that works with this specific browser is being executed.

7.4.3 TestNG XML changes

In this suite we create two test names each containing the parameter and the parameter's value. In such way they will be executing one after another.

There is also a simple way to run tests in parallel as TestNG offers such feature, however, it takes a lot of time to build up a good architecture of the tests execution in order to avoid conflicts, as same data might be accessed and changed at the same time which might lead to unexpected data output.

```xml
<suite name="TestSuite" parallel="none">

    <test name="ChromeTest">
        <parameter name="browserType" value="chrome" />

        <classes>
            <class name="login.Login" />
            <class name="login.RememberUser" />
            <class name="profile.PersonalDataChange" />
            <class name="profile.ScreenLock" />
            <class name="people.PrivatePerson" />
            <class name="transactions.Sale" />
            <class name="transactions.Purchase" />
            <class name="goods.AddProduct" />
            <class name="goods.AddCategory" />
            <class name="salesPlaces.AddTable" />
            <class name="salesPlaces.CountProductsAndPersonal" />
            <class name="desktop.AddProductToCassa" />
            <class name="desktop.AddCommentary" />
            <class name="kitchen.KitchenOrder" />
            <class name="desktop.EmptyCheckOutError" />
            <class name="desktop.ScreenLockCassa" />
            <class name="desktop.ProductDiscount" />
            <class name="cash.AddCashToCassa" />
            <class name="desktop.TwoTablesAtOncePlusSalesDay" />
            <class name="desktop.TwoEmployeesAtOnce" />

        </classes>

    </test>

    <test name="FirefoxTest">
        <parameter name="browserType" value="firefox" />
        <classes>
            <class name="login.Login" />
        </classes>
```

Figure 10. TestNG XML

7.5 WebDriver limitation

Throughout the project there were no problems that could not be resolved with Selenium WebDriver except one. A print out dialogue, handling which is impossible with this framework only. It was impossible to avoid and at the same time it was necessary to pass through it, in order to execute important test cases. There was no other choice but to look for a solution out of the box. The most simple way to pass through this print dialogue was to import the *Robot class*. As the print preview appeared, key

event of *Escape* button was simulated and the dialogue was closed. It is a simple solution and works as intended, however there is a side effect. When the print preview is handled it must be on the frontend of the desktop. This brings some limitations, since the computer can not be used while this test case is being executed. If, for example, a browser or an application is in front of the browser instance where the test runs the test will fail.

8 Analysis

8.1 Metrics

There is a total of 21 automated test cases that execute in 6 minutes 30 seconds (+- 20 sec.), ~1440 pure lines of code (import and comments excluded). The code coverage is 97.4% which is considered as a high percent as the average acceptable code coverage is around 70-80%. The reason for such high percent is the size of the application on its early development stage. The percent of coverage together with the amount of code lines gives a more clear picture on how much code is being used. A total amount of 108 test cases were written manually, that cover all the functionality that was rational to document and test.

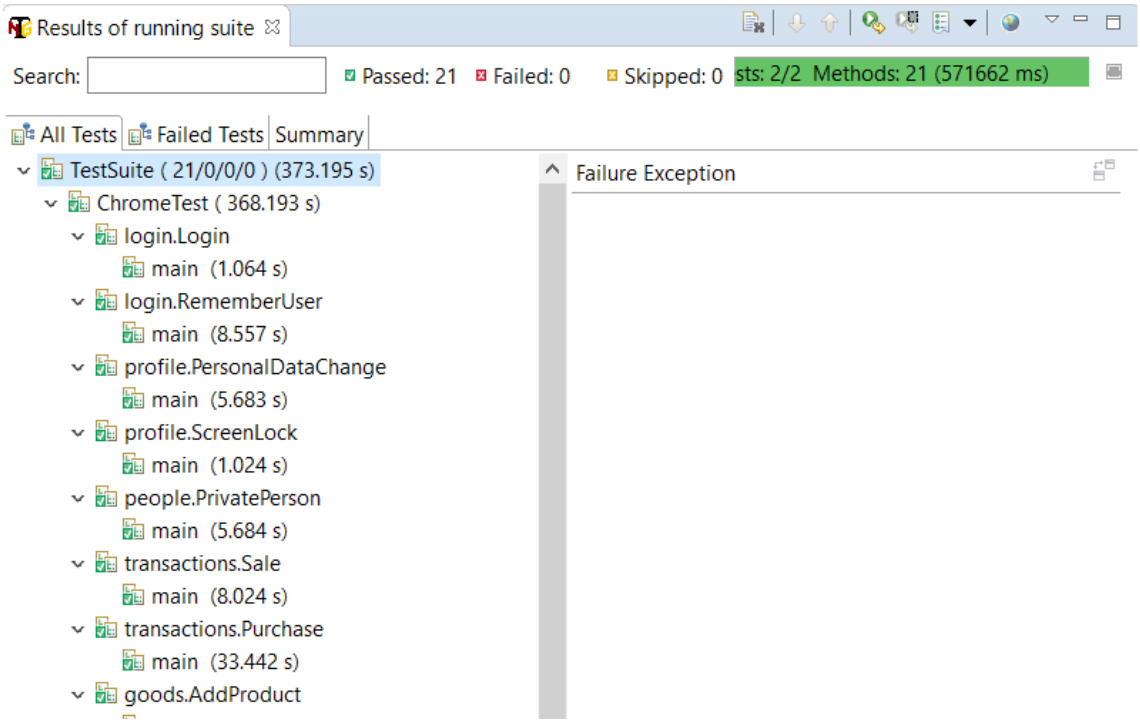


Figure 11. Number of test cases and execution time

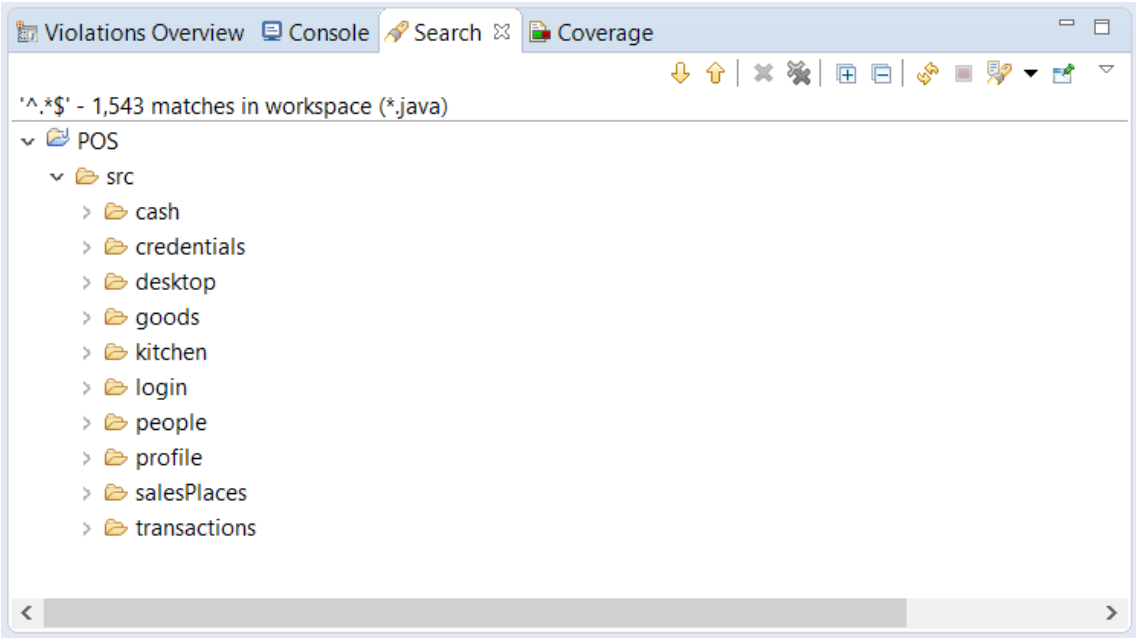
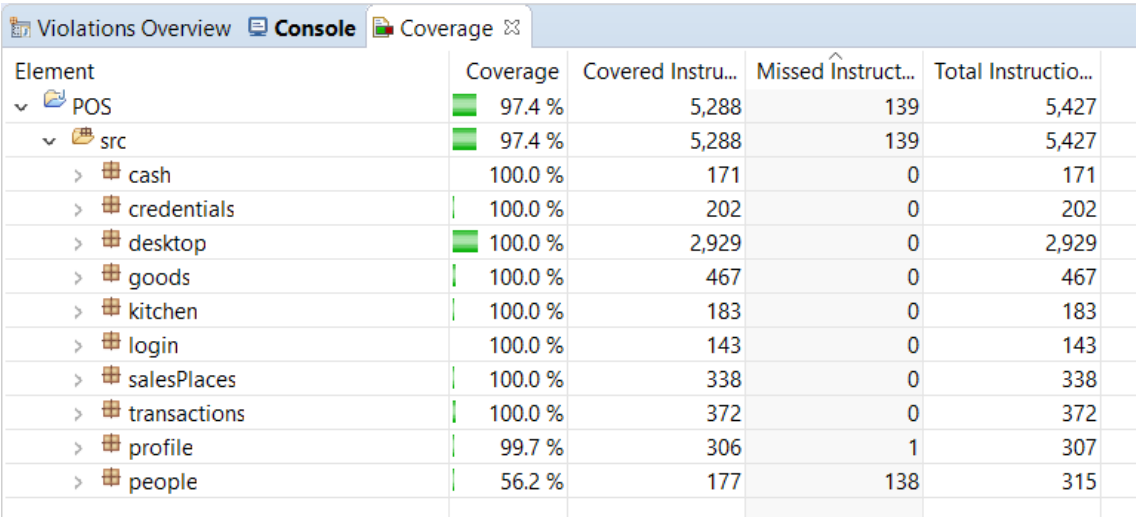


Figure 12. Lines of code



Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
POS	97.4 %	5,288	139	5,427
src	97.4 %	5,288	139	5,427
cash	100.0 %	171	0	171
credentials	100.0 %	202	0	202
desktop	100.0 %	2,929	0	2,929
goods	100.0 %	467	0	467
kitchen	100.0 %	183	0	183
login	100.0 %	143	0	143
salesPlaces	100.0 %	338	0	338
transactions	100.0 %	372	0	372
profile	99.7 %	306	1	307
people	56.2 %	177	138	315

Figure 13. Code coverage

8.2 Manual testing

Practical part of this thesis lasted for two months. All the records of time were made accurately. Two types of recording were used, either in KanbanFlow (project management tool) or manually. KanbanFlow time was divided into three categories, overall testing time, testing made in Live environment and time spent on single tasks. Examples attached to the Appendix 2.

70 hours of manual testing has been recorded. 5 hours has been spent on writing test cases which are not included into calculations and comparison, since it is necessary in any case. Approximately 10 hours has been spent on manual testing that either couldn't be replaced with automation or was made for prophylactic purposes. This leads us to a total of 55 hours of manual testing which was parallelly automated.

8.3 Test automation

Test automation (writing scripts and maintaining) took 20 hours (without execution). Of course the speed of learning depends on each person individually, but overall, the level of difficulty in mastering Selenium WebDriver and learning how to write reliable tests is not very high, thus it took 10 hours to set up the environment and master the framework on a level that is acceptable. The only precondition is knowledge of one of the languages supported by Selenium.

The average execution of a test suit which consisted of 21 automated test cases took approximately 6 minutes. (+- 20sec). The AUT almost instantly achieved 8 frequently used test cases and ended up with 21. Considering this fact and how fast the application was developed the average amount of executed test cases that makes sense to use in calculations is 15-16. The average time is 4 minutes for executing one test suite containing 15 test cases. Manually testing such test suit takes ~10 minutes. It might take even longer if manual testing was made a while ago, since the test steps are forgotten and it takes extra time to accurately check the document and recreate them. Simple

calculations lead us to the following conclusion. Since automated tests are much faster each suit saves us 6 minutes, additionally, not only the suit executes faster it gives us 4 extra minutes because the human resources can be used elsewhere while the tests are running.

8.4 Calculations and conclusion

All the calculations were made in favor of manual testing in order to make the conclusion and summary as reliable as possible. As it can be seen, 55 hours of manual testing was necessary for the application to meet all the requirements. By using automation 30 hours were required to automate the same amount of test cases. If we take into consideration only time as the measurement unit it is obvious that automating such projects even under such circumstances is undoubtedly beneficial for two reasons, less working hours and a faster execution time. These two reasons will be approached with a more precise explanation.

Converting time unit into currency is not a good practice as abstraction gives a better overall view, however, one example based on the current average salaries on the market is acceptable for additional information. Automation and manual testing is being measured as 1:1 ratio if time is used as a unit of measurement, which might give a false impression of benefit as automation engineer salary is higher than a manual tester salary. In the following calculations we will assume, that 1 month contains 160 working hours. In the year of 2017 the average salary of a manual tester per year according to PayScale [19] is 53k. This is an approximate of 28 dollars per hour. An automation engineer salary, also according to PayScale is 70k per year which brings us to an approximate of 36 dollars per hour. 55 hours of manual testing equals to 1540 dollars, 30 hours of test automation equals to 1080 dollars which brings us to a net profit of 460 dollars. Of course every company has its own policy how working hours and payments are approached, however, this a good standard to compare currency value.

Now lets take a closer look on a significant benefit which is not connected to the currency value – the speed of execution. One test suit, as was mentioned before executes 6 minute faster (4 min automated execution, 10 min manual testing), which

means that a feedback will be given 60% faster. We have 55 hours of manual testing, if it would have been replaced with automation the execution time would take 22 hours (55 – 55 * 0.6, 60% faster) which means that all the required testing would have been done 33 hours faster. But this time does not include the automation process and learning. This means the actual total amount of time spent on automation for the same amount of testing is 22 + 30 (test automation + learning time) = 52 hours which already gives us a time profit of 3 hours. Although, on the first sight this amount might seem insignificant, but in the long-term perspective, lets assume in the next project, this same person won't need to spend any time on learning, moreover, his coding skills will be on a higher level which will allow to write scripts faster. This will instantly make the automation process 10 hours more profitable than manual testing and the profit will keep rising over time.

As it was previously mentioned this application was specially chosen considering nuances such as insufficient time, cost and job security in paragraph 1.2 and thus in most cases it is tested manually. By converting theory into real life practice these nuances become risks. Relying on the made practical part each risk will be approached with an explanation why should it still be taken.

- Insufficient time – Undoubtedly there are companies with a busy schedule. But lets approach this situation from another angle. First of all, it was proven on practice that mastering functional/regression/system automation does not require a colossal amount of time. The solution is a little bit out of the box but still possible to apply in real life practice. There is an option to offer an employee overtime work where he will be learning how to automate. First of all, it will not take a lot of time and be beneficial to the development process and to the company overall in the near future. Secondly, human error will be excluded as it might arise in manual regression testing. Thirdly, not only it brings benefit to the company but also to the employee himself. It will motivate him to improve drastically in his career as automation becomes more and more popular. Moreover, he will benefit financially as an automation engineer salary on the

market is higher than of a manual tester salary [5] . Why not to apply this strategy if both sides benefit, a win-win game condition.

- Job security – As it was mentioned before testers that are used to manual testing might be threatened by automation. A review will be given as the author was sent in the same position and under the same circumstances. First of all, by going out of the comfort zone, new opportunities are opened and these opportunities are thous that were mentioned in the "Insufficient time" problem (career and salary growth). It should be not considered as a threat, but as a chance to improve yourself and benefit out of it. In addition, it was shown that the level of difficulty in mastering automation (in such kind of projects) is not that high. If this is not enough, then again lets think outside of the box. Information technology is rapidly evolving and in the near future there will be more and more automation. Without a doubt, manual testing will still be necessary but in a much less amount. Employees that are able to automate and execute testing manually will be required on a bigger scale and this should be considered as a real threat.

- Cost – additional expenses might be avoided with ease. By paying for test automation tools you are paying for comfort and conveniences. It is possible to automate almost anything with free tools. The only side effect is a more complex environment setup and use of combination of different frameworks.

9 Future works

Changes in manual testing documentation

In this thesis one of the most simple test documentation templates were used. As the application didn't have many versions, no cross platform testing was needed, only one test designer was active, no fields such as executed by whom and date were necessary nor any post condition fixation was required. Test cases were written in Excel. Template column headings were created as follows "Test idea", "Test description", "Status", "Error description", "Additional information". In the future if the application gets more complex, new and a more detailed template must be taken in use. Although, the light version that was used had no disadvantages in this specific development cycle, as it covered all the needs that are expected from test case documentation.

Email reports

Automated test suite results reporting via email was also not necessary. It is a good feature indeed and can be implemented without any trouble to monitor results right after a test suite finishes executing. In my case errors were handled as soon as automated test suite was completed. No suits were running at night or remotely since the application did not reach large scales. As soon as load reaches certain sizes email reporting might be instantly taken in use.

Scheduled test runs

Scheduled tests are a good combination with email reports. It is possible to run required test cases in a specific time by creating a batch file and setting a task in the windows control panel that will run it. This method can be used in the combination of Selenium and Windows. Of course there are other ways to configure this task depending on the tools and operating system. If a fast, remote, and in a certain period of time reporting environment will be needed it can be implemented.

Key performance indicator

On this stage of the development and in such conditions I did not find it necessary to use metrics such as KPI. Although, it is considered as the backbone of the business. But it has its place in the section of the future work as it might be integrated if precise statistics of degradation or improvement will be required.

10 Summary

The first goal in this thesis was to cover the AUT with test cases and implement automation with as much benefit as possible using functional, regression and system testing.

The precondition of a successful automation is a documentation of test cases and manual execution. At the end a total of 108 test cases were written (Excel) and accurately tested manually. Afterwards necessary test cases were automated. These automated tests are stable and reliable and were put under test after a decent amount of new implementations. This way they can be considered as high quality tests as the investments brought more profit than loss, which was proven by pure calculations.

While the practical part was executed time was accurately measured. The analysis that was build upon the practical part has a strong foundation. All the tools that were used are the newest versions available at the time of writing this thesis, so the methods and technologies used were all up to date.

The results of the analysis are clear and the benefit of automation was proven. A conclusion was written approaching and explaining all the problems and risks that come with such project under concrete circumstances offering solutions how to handle them.

References

[1] Joe Fernandes (Oracle), Alex Di Fonzo (Synchronoss Technologies), "When to Automate Your Testing (and When Not To)" [Online]. Available: http://www.oracle.com/technetwork/topics/qa-testing/whatsnew/when-to-automate-testing-1-130330.pdf. [Accessed 13 05 2017].

[2] Gerard Meszaros, Shaun M. Smith, Jennitta Andrea, "The Test Automation Manifesto" [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-45122-8_9. [Accessed 13 05 2017].

[3] Peter Sabev, Katalina Grigorova, "Manual to Automated Testing: An Effort-Based Approach for Determining the Priority of Software Test Automation" [Online]. Available: http://waset.org/publications/10003250/manual-to-automated-testing-an-effort-based-approach-for-determining-the-priority-of-software-test-automation. [Accessed 13 05 2017].

[4] Stefan Münch, Peter Brandstetter, Konstantin Clevermann, Oliver Kieckhoefel, Reiner Schäfer "The Return on Investment (ROI) of Test Automation" [Online]. Available: https://www.ispe.org/pe-ja/roi-of-test-automation.pdf. [Accessed 13 05 2017].

[5] Prof. V. N. Maurya, Er. Rajender Kumar "Analytical Study on Manual vs. Automated Testing Using with Simplistic Cost Model" [Online]. Available: http://vixra.org/pdf/1208.0216v1.pdf. [Accessed 13 05 2017].

[6] Alégroth, E. , Feldt, R. & Ryrholm, L. , "Empir Software Eng", "Visual GUI testing in practice: challenges, problemsand limitations" [Online]. Available: https://link.springer.com/article/10.1007/s10664-013-9293-5#Abs1. [Accessed 13 05 2017].

[7] Leotta M., Clerissi D. , Ricca F. , Tonella P. (2014) "Visual vs. DOM-Based Web Locators: An Empirical Study" [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-08245-5_19. [Access 13 05 2017].

[8] "Selenium" [Online]. Available: http://www.seleniumhq.org/docs/01_introducing_selenium.jsp. [Accessed 13 05 2017].

[9] The Apache software foundation, "Apache License 2.0" [Online] Available: https://www.apache.org/licenses/LICENSE-2.0. [Accessed 13 05 2017].

[10] Wikipedia, "Selenium (software)" [Online]. Available: https://en.wikipedia.org/wiki/Selenium_(software). [Accessed 13 05 2017].

[11] Selenium project, "Selenium WebDriver" [Online]. Available: http://www.seleniumhq.org/projects/webdriver/. [Accessed 13 05 2017].

[12] "Selenium WebDriver architecture" [Online]. Available: https://www.youtube.com/watch?v=aujzqboRO9Y. [Accessed 21 05 2017].

[13] "PhantomJS" [Online]. Available: http://phantomjs.org/. [Accessed 13 05 2017].

[14] "CODED UI" [Online]. Available: https://msdn.microsoft.com/en-us/library/dd286726.aspx. [Accessed 13 05 2017].

[15] Cypress.io, Inc, "Cypress" [Online]. Available: https://www.cypress.io/. [Accessed 13 05 2017].

[16] "The World Wide Web Consortium (W3C)" [Online]. Available: https://www.w3.org/TR/webdriver/. [Accessed 13 05 2017].

[17] "Black box and white box comparison" [Online]. Available: https://www.pinterest.com/pin/181762534935799420/. [Accessed 21 05 2017].

[18] "Functional testing architecture" [Online]. Available: https://www.packtpub.com/mapt/book/application-development/9781783553372/9/ch09lvl1sec52/Functional+testing. [Accessed 21 05 2017].

[19] PayScale incorporated [Online]. Available: http://www.payscale.com/about/methodology. [Accessed 21 05 2017].

Appendix 1 – Excel documentation of test cases

The development and testing was made within an Estonian speaking team thus the test cases are written in Estonian language. Only important columns were added for a better view. Columns such as priority, test steps and additional information were pulled out. MOK means NOT OK.

Table 1. Test cases

1	**Testiidee**	**Testilugu**	**Tulemus**	**Vea kirjeldus**
2	Sisselogimine	Sisselogimine kasutajanime ja parooliga	OK	
3		Kasutaja mäletamine	OK	
4		Väljalogimisel ühelt lehelt, kõigil teistel suunatakse ümber sisselogimis lehele	OK	
5				
6	Profiil	Üldandmete muutmine	OK	
7		Ekraanilukk ülevalt menüüst	OK	
8		Kasutajakonto vaates kasutajanimi muutmine	OK	Pärast nuppu Muuda vajutamist midagi ei juhtu
9		Nime/perenime kuvamine	OK	Liiga suure nime/perenime disain läheb katki.
10				
11	Lahter "Isikud"	Eraisiku lahtris isiku lisamine, muutmine ja eemaldamine	OK	
12		Juriidilised isikud lahtris isiku lisamine, muutmine ja eemaldamine	MOK	Juridilise isiku lisamisel ja seejärel kustutamine ei võimalda uut isikut sama initsiaalidega luua.
13				
14	Lahter "Töötajad". Ainult Juhataja ametil.	Põhitöötajate kustutamine ja muutmine	OK	
15		Hooajatöötajad kustutamine ja muutmine	OK	
16				
17	Lahter "Tehingud".Ainult Juhataja ametil.	Pärast maksmist lahtris "Kassa", lahtris "Tehingud" Müük-i all ilmub uus rida ostetud kaupa infoga	OK	

18		Pärast tellimuse lisamist lahtris "Tooted" toode laoseis muutub.	OK	
19		Kuupäeva valimine uue tellimuse lisamisel	OK	
20		Müügi ja Ostu kustutamine	MOK	Kui proovida tehingud lahtris "Müük" järjest ära kustutama (kiiresti), siis kustutamise vahel kuvatatakse tehingu informatsioon. P.S juhtub väga harva, arvan, et ei ole üldse oluline.
21		Tehingute kuvamine	OK	
22				
23	Lahter "Tooted"	Lisa toode üldandmete sisadlamine	OK	
24		Lisa toode müügikoha andmete sisaldamine	OK	
25		Toode muutmine ja kustutamine	OK	
26		Uue kategooria lisamine	OK	
27		Kategooria kustutamine	OK	
28		Toodete kuvamine lehel	OK	
29		Köögis valmistatav toode	OK	Lahtris eelroad salvestades toodet optioniga "Köögis valmistatav" viskab errori ja ei salvesta. Muu info muutmisel viskab errori, kuid salvestab info
30		Valesti arvutan Neto/Bruto %	OK	
31				
32	Lahter "Müügikohad"	Kontaktandmete muutmine	OK	
33		Laudade lisamine Saalis ja Terassis	OK	
34		Müügikoha info kuvamine	OK	
35				
36	Lahter "Kassa"	Luadade ja toodete korrektne kuvatus pärast nende lisamist lahtrides "Tooted" ja "Müügikohad"	OK	
37		Laud millel on maksmata tellimus teise värviga	OK	
38		Pärast nuppu Maksa vajutamisel summa sisestamine	OK	Oleks vaja kontrolli, et oleks mingi piiratud maksimaalne summa, muuljuhul suure arvu puhul disain läheb katki
39		Tühja summa sisestamisel ja pärast nuppu Kinnita vajutamisel ilmub veateade	OK	
40		Prindi nupp	OK	Nupp toimub nagu "Kinnita", ehk lisaks prindimisele ka makse läheb läbi
41		Müümine ühes lauas nt.	OK	

		kahele erinevale inimesele ja samaegselt teises		
42		Maksmisel valin toode. Vajutan Kinnita. Tellimus lõpetatud.	OK	Tellimus on tõepoolest lõpetatud ainult siis, kui vajutatakse Tagasi töölauale. Kui aga vajutatakse Prindi leht refreshib. 13.03 Prindi vajutamisel tehing kinnitab, kuid leht ikka refreshib ja kõik toodet jäävad. On võimalus juhuslikult teist korda ära maksta sama toodete eest, seljuhul tehingutes viimane tehing lihtsalt muudab Aega millal oli makstud.
43		Pärast nuppu Maksa vajumatist müügi info kuvamine	OK	Tulb Hind ei sisalda hinnas nullid. Näiteks Summa tulbas on 1.90, Hind tulbas on 1.9
44		Maksmisel lisame paar toodet. Maksame ühe toode eest sularahas kus maksame 10-euriga näiteks. Pärast Tagasi Töölauale vajutamist Saldo peaks muutma 0-ks.	OK	Ilmub mitte Saldo 0.00 vaid summa mis oli sisestatud varem
45		Maksmisel saldo kuvamine	OK	Test case: Maksmisel lisame paar toodet. Maksame ühe toode eest sularahas kus maksame 10-euriga näiteks. Pärast Tagasi Töölauale vajutamist Saldo peaks muutma 0-ks.
46		Erinevate toodete eest maksmisel - tagasi töölauale nupp ei tööta	OK	Lisame näiteks 4 toodet. Maksame ühekaupa nende eest ja viimase toode maksmisel tekkib see probleem.
47		Maksmine nelja erineva meetodiga	OK	Lisame näiteks 4 toodet. Maksame ühekaupa nende eest ja viimase toode maksmisel tekkib probleem nupuga "Tagasi töölauale".
48		Tellimuse lõppetamine	OK	Lahter "Kassa". Tellimus on tõepoolest lõpetatud ainult siis, kui vajutatakse Tagasi töölauale. Kui aga vajutatakse Prindi leht refreshib.
49		Maksa nupp	OK	Kui pole ühtegi tooted, siis maksmine ei ole võimalik
50		Vahearve number ja kviitungi number peavad kokku klappima	OK	Erinevad numbrid
51				
52		Kööki modal - Söögikäigu numbrid: min 1, max 10	OK	
53		Kööki modal - Märkuse ja Kommentaari lisamine	OK	
54		Kööki saadetud toodete kogust ei saa muuta	OK	
55		Laud valmis toodega köögist	OK	

		on lilla värviga ja ekooniga		
56		Valmis toodete modalis nuppud "Tulen hiljem" ja "Vastuvõetud"	OK	
57				
58		**Allpoolsed nuppud**		
59		Lõpeta vahetus nupp	OK	
60		Maksa nupp	OK	
61		Ekraanilukk	OK	
62		Tühista nupp	MOK	Täpsustada. Kas funktsionaalsus puudub või see peaks nii olema (füüsiline nupp)
63		Vahearve nupp	OK	Täpsustada. Kas funktsionaalsus puudub või see peaks nii olema (füüsiline nupp)
64		**Vasakpoolsed nuppud**		
65		Koguse lisamine	OK	Oleks vaja kontrolli, et oleks mingi piiratud maksimaalne kogus, muuljuhul suure arvu puhul disain läheb katki
66		Koguse lisamine v2	OK	Murdarvud peavad olema piiratud
67		Maksimaalse soodustuse lisamine ehk kontrollida võimalikult suure soodustuse mida me täpsustame lahtris "Tooted"	MOK	Kui soodustutst kustutada Summas kuvab NaN. Üleküsida.
68		Kustuta nupp	OK	
69		Tehingud nupp	MOK	Täpsustada, kas puudub funktsionaalsus
70		Eraisik nupp	MOK	Täpsustada, kas puudub funktsionaalsus
71		Kööki nupp	OK	Täpsustada, kas puudub funktsionaalsus
72		Kommentaar nupp	OK	
73		Uus nupp	OK	
74				
75	Admin vaade	Lisa isik	OK	
76		Lisa isik v2	OK	Isiku lisamisel ükskõik mis valitud amet pärast salvestamist muutub juhatajaks.
77		Kasutaja üldandmete ja kasutajakonto info muutmine	OK	
78		Kasutaja kustutamine	OK	
79		Inimesele kasutaja loomine	OK	
80		Inimesele kasutaja kustutamine	OK	Kasutaja üldandmete ja kasutajakonto andmete kustutamine ei õnnestu. Pärast nuppu Salvesta vajutamisel näeme, et sellel kasutajal staatus "Kasutaja olemas" ei muutnud staatuseks "pole olemas" , vaid jäi "olemas".

81				
82	Lahter "Sularaha"	Sularaha lisamine kassasse	OK	
83		Sularaha lisamise kirje kustutamine	OK	
84		Sularaha lisamise kirje muutmine	OK	Kui sisestada tähte, mitte arvu, siis kohe kustutab rea. Oleks vaja validaator.
85		Sularaha väljavõtmine kassasst	OK	
86		Sularaha väljavõtmine kirje kustutamine	OK	
87		Sularaha väljavõtmine kirje muutmine	OK	Kui sisestada tähte, mitte arvu, siis kohe kustutab rea. Oleks vaja validaator.
88		Sularaha ülelugemised - lisamine	OK	Error
89		Sularaha ülelugemise printimine	OK	
90		Sularaha ülelugemised kustutamine	OK	
91		Refresh nupp	OK	
92		Ülelugemine erinevatel töötajatel(tüübil andmebaasis)	OK	
93				
94	Vahetuse lõpetamine aken	Sularaha kassas arvestuslikult eelmisel ülelugemisel	OK	
95		Sularaha tehingute summa alates eelmisest lugemisest	OK	
96		Sularaha kassast välja võetud	OK	
97		Sularaha kassasse lisatud	OK	
98		Sularaha kassas arvestuslikult kokku	OK	
99				
100	Lõpeta vahetuse aruanne	Jäägi muutus	OK	
101		Ülejäänud	OK	
102				
103	Arve	Arve info kuvamine Maksmis lehel printimisel(Üldine)	MOK	Kellaaeg mitte ilus. Näide: Kuupäev: 22.3.2017 13:2 + Hind ja Kokku ernevad. Näiteks Hind on 9.5, Kokku on 9.50. Peaksid olema samad.
104		Hind soodustusega	MOK	Test case: Lisame toode hinnaga 9.5. Lisame soodustust 10%. Maksame ära. Arve peal on näha Hind = 9.5 ehk ilma soodustust. Kokku = 8.55 ehk soodustusega. Arve peal kuskil soodustuse protsenti ei ole näha. Väga ebamugav ja kliendile mitte arusaadav.
105				
106				

107	Töö kiirus/üldine	Kassa	OK	
108		Sularaha	OK	
109		Müügipäevad	OK	
110		Müügikohad	OK	
111		Tooted	OK	
112		Ladu	MOK	
113		Tehingud	MOK	Ostu lisamisel töötab aeglaselt, sest on seotud Laduga
114		Töötajad	OK	
115		Isikud	OK	
116		Klikkides palju kordu/kiiresti nuppu peale, näiteks ava vahetus, funktsioon käivatatakse palju korda, seega avatatakse palju vahetusi	OK	
117		Vale käibemaksu protsent	OK	
118				
119	Lahter "Köök"	Tellimuste kuvamine iga 10 sekundiga	OK	
120		Tellimuse staatus. Kui on valmis - hall värv. Kui ei ole - oranž	OK	
121		Tellimusele "Märkuse" lisamine	OK	
122		Tellimuse kommentaari kuvamine mis oli lisatud Kassas	OK	
123				
124	Lahter "Ladu"	"Ost" õige kuvamine, juhul kui ostetakse seda toodet	OK	
125		"Müük" õige kuvamine, juhul kui müüakse seda toodet	OK	
126		"Laoseis" õige kuvamine	OK	
127		Perioodi filter	OK	
128				
129	Lahter "Köögitellimused"	Tellimuse kustutamine	OK	
130		Arvude(Hinnad) korrektne kuvamine	OK	Neto hind = Bruto hind, ja sellega tuleb vale Summa kokku
131				
132		**Count OK**	98	
133		Count MOK	10	

Appendix 2 – Time recording examples (KanbanFlow)

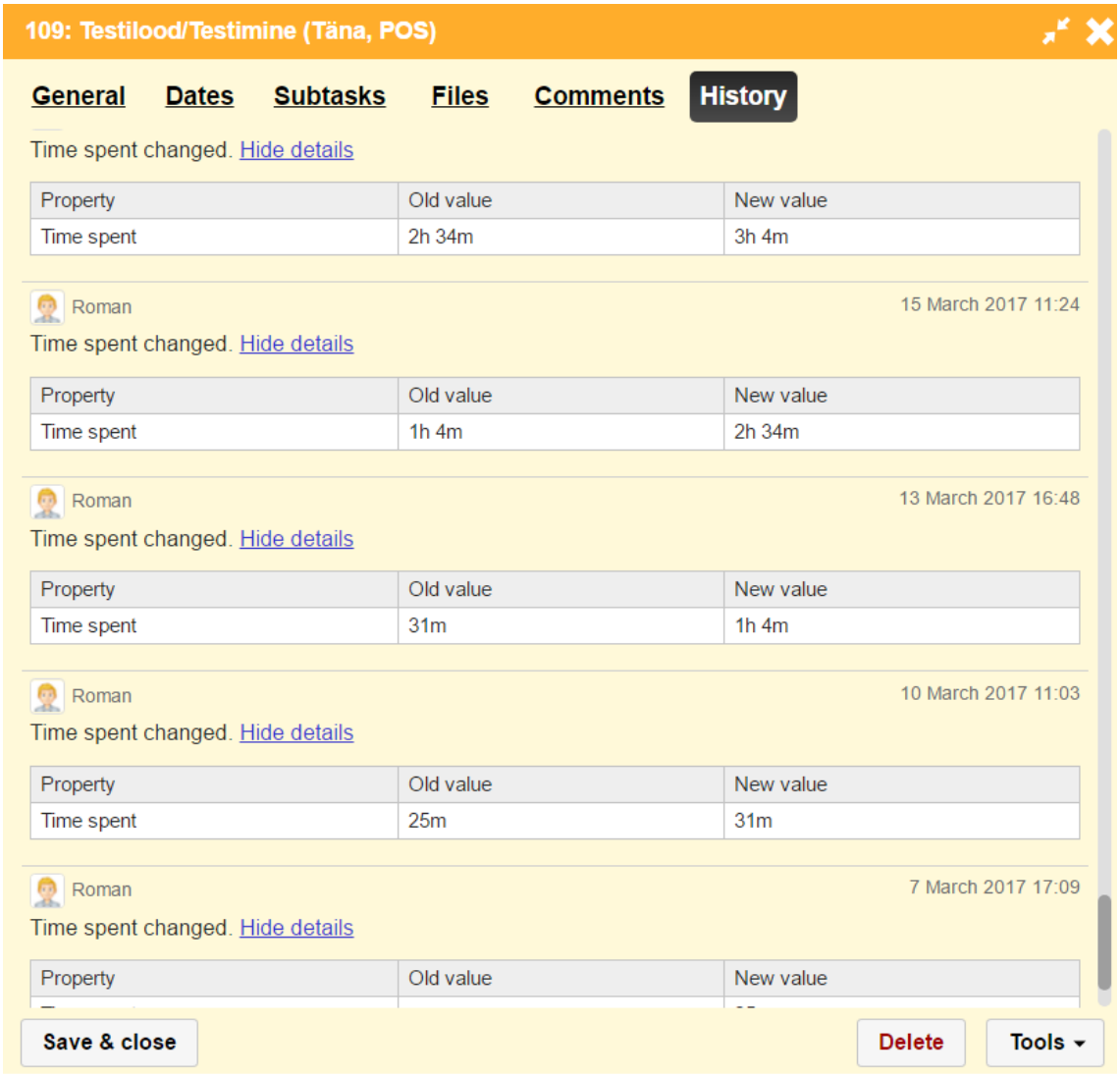


Figure 14. Kanbanflow overall time example

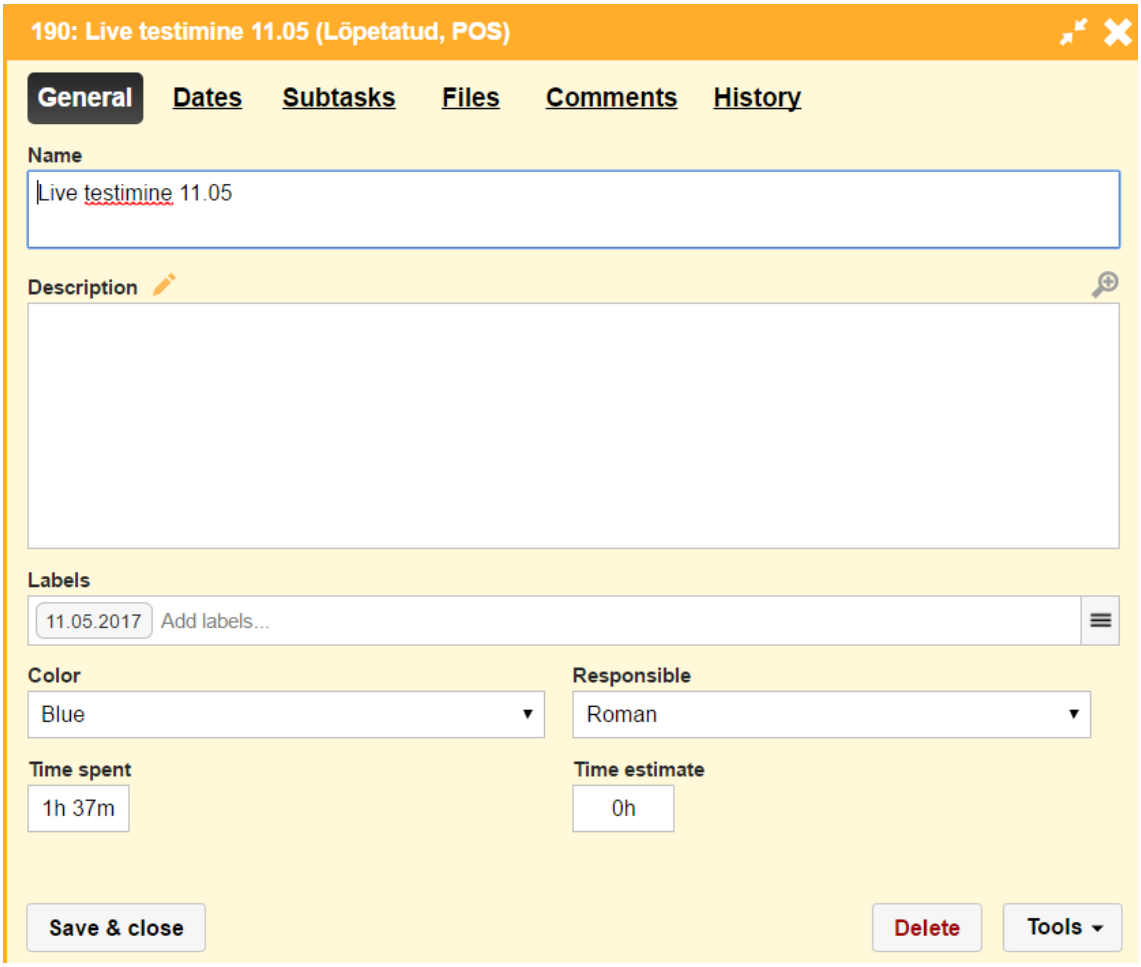


Figure 15. Kanbanflow Live testing example

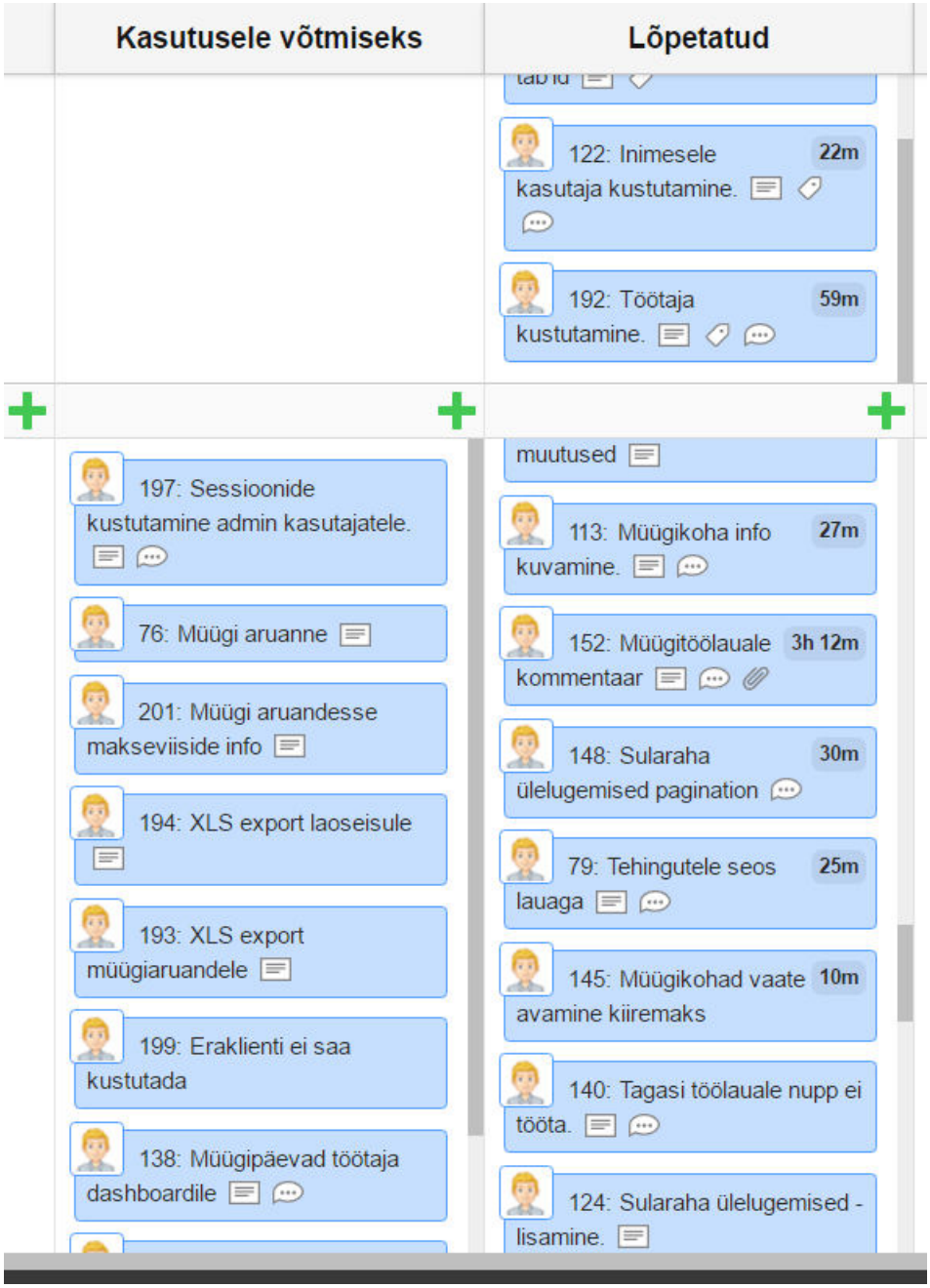


Figure 16. Kanbanflow single tasks example

Appendix 3 – Git link

https://github.com/iquinity/PointOfSale – Point of sale system's automated test cases