Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Vývoj automatických testů webových aplikací za použití nástroje Selenium webdriver

Lukáš Zmatlík

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Lukáš Zmatlík

Informatika

Název práce

Vývoj automatických testů webových aplikací za použití nástroje Selenium webdriver

Název anglicky

Web applications test automation using Selenium webdriver tool

---

Cíle práce

Práce se zabývá automatickým testováním uživatelských rozhraní webových aplikací s využitím nástroje Selenium. Hlavním cílem práce je navrhnout a implementovat pomocný nástroj sloužící k usnadnění tvorby testů v Seleniu.

Metodika

Práce sestává ze dvou hlavních částí – teoretické a praktické. V rámci zpracování teoretické části bude provedeno studium odborných informačních zdrojů. Na základě zjištěných poznatků budou popsána teoretická východiska pro zpracování praktické části.

Praktická část práce spočívá v návrhu a implementaci aplikace sloužící ke zjednodušení tvorby testů v nástroji Selenium. Využit bude programovací jazyk Java a standardní metody a nástroje softwarového inženýrství. Výsledná aplikace bude otestována a budou navrženy možnosti jejího případného dalšího rozvoje.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Selenium, automatizace testů, webová aplikace, Java, testování UI

Doporučené zdroje informací

Clean Coders. Clean Coders [online]. Clean Coders, 2017. Dostupné z: https://cleancoders.com/

Head First Java, 2nd Edition. 2nd. ed. Sebastopol, Kalifornie, USA: O'Reilly Media, 2005. ISBN 0596009208.

HOW TO USE SELENIUM FOR MOBILE CROSS BROWSER TESTING (WITH EXAMPLES) [online]. Oulu, Finland: Ville-Veikko Helppi, 2016. Dostupné z: http://bitbar.com/how-to-use-selenium-for-cross-browser-testing-on-mobile-devices/

Java™ Platform, Standard Edition 7 API Specification. Docs.oracle.com [online]. oracle, 2016. Dostupné z: https://docs.oracle.com/javase/7/docs/api/

SeleniumHQ. SeleniumHQ [online]. SeleniumHQ, 2017. Dostupné z: http://www.seleniumhq.org/

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 11. 1. 2018

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 11. 1. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 10. 03. 2018

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Vývoj automatických testů webových aplikací za použití nástroje Selenium webdriver" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2018 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Jiřímu Brožkovi, Ph.D., za cenné připomínky, rady a ochotu při vedení mé bakalářské práce.

Vývoj automatických testů webových aplikací za použití nástroje Selenium webdriver

Abstrakt

Tuto práci jsem psal hlavně, jako návod na to, jak správně tvořit kvalitní a snadno udržovatelné testy webových aplikací za pomoci nástroje Selenium Webdriver. Popisuji zde jednotlivé třídy, metody, proměnné a lokátory web elementů na stránce, včetně důvodů jejich použití.

V další části popisuji návrhové vzory, jejichž správné pochopení a použití je nezbytné pro správnou tvorbu testů. Popisuji zde přístup k tvorbě testů za pomoci keyword testingu, protože je to moderní přístup k tvorbě testů, který umožňuje automatizovat testy i lidem bez hlubší znalosti této problematiky. Součástí práce je i popis celého projektu Selenium a všech jeho nástrojů včetně doporučení, kdy konkrétní nástroj použít.

Celou práci jsem vytvořil v programovacím jazyce Java, to ale neznamená, že by se principy, které v ní popisuji nedali použít ani v dalších jazycích, které Selenium podporuje. K vytvoření testovacích scriptů za pomoci keywords jsem použil nástroj Excel. Součástí práce je také kompletní program včetně ukázky několika testů na aplikaci email.seznam.cz.

Klíčová slova: Selenium, Webdriver, Testování, Webová aplikace, Keyword, Java, Excel, Page object, Page Factory, Třída, Metoda, Proměnná

Web applications test automation using Selenium webdriver tool

Abstract

I wrote this bachelor thesis mainly as a manual how to create high quality and easy to maintain tests of web applications using Selenium Webdriver tool. I am describing here individual classes, methods, variables and locators of web elements of the page including the reasons why I use them.

In next part I am describing design patterns, whose understanding and using is necessary for right test development. I am describing here approach to test automation development using keyword testing because it is modern way to automate tests which allows to automate tests even people without deep knowledge of this problematics. Another part of this project is description of whole Selnium project and all its parts including recommendation which tool should be used for specific test development.

I created this project in Java programming language but it doesn't mean that principals which I am describing here cannot be used in different programming languages which are supported by Selenium. I used the Excel tool to create keywords test scripts. Part of this work is also complete program including sample of couple tests of the email.seznam.cz application.

Keywords: Selenium, Webdriver, Testing, Web application, Keyword, Java, Excel, Page object, Page Factory, Class, Method, Variable

Obsah

Seznam obrázků

Seznam použitých zkratek

tj. – to je

keyword – klíčové slovo

Webdriver – ovladač webového prohlížeče

API – Aplikační programovací rozhraní

1 Úvod

Tato práce pojednává o tvorbě automatických testů webových aplikací za použití nástroje Slenium WebDriver v programovacím jazyce Java.

V první části této práce rozebírám výhody a nevýhody jednotlivých přístupů k testování, ale hlavní část této práce tvoří praktická ukázka konkrétních testů a jejich architektury. Popisuji zde systém tvorby automatických testů pomocí keyword testingu.

2 Cíl práce a metodika

2.1 Cíl práce

Práce se zabývá automatickým testováním uživatelských rozhraní webových aplikací s využitím nástroje Selenium Webdriver. Hlavním cílem práce je navrhnout a implementovat pomocný nástroj sloužící k usnadnění tvorby testů s použitím nástroje Selenium Webdriver.

2.2 Metodika

Práce sestává ze dvou hlavních částí - teoretické a praktické. V rámci zpracování teoretické části bude provedeno studium odborných informačních zdrojů. Na základě zjištěných poznatků budou popsána teoretická východiska pro zpracování praktické části.

Praktická část práce spočívá v návrhu a implementaci aplikace sloužící ke zjednodušení tvorby testů v nástroji Selenium Webdriver. Využit bude programovací jazyk Java a standardní metody a nástroje softwarového inženýrství. Konkrétní testy budou vytvořeny na aplikaci email.seznam.cz.

3 Teoretická východiska

3.1 Selenium

Selenium je sada různých softwarových nástrojů, které využívají různý přístup k automatizaci testů webových aplikací. Celá sada nástrojů přináší velké množství funkcí k testovaní všech možných druhů webových aplikací na různých typech internetových prohlížečů. Tyto operace jsou velmi flexibilní, mají širokou sadu možností lokalizace web elementů na stránce a porovnání výsledků testů s reálným chováním aplikace. Jednou z klíčových vlastností Selenia je možnost vykonat jeden test na různých prohlížečích s různým nastavením. [1]

3.1.1 Historie

Selenium vzniklo v roce 2004 když Jason Huggins testoval interní aplikaci pro ThoughtWorks. Uvědomil si, že jsou lepší způsoby, jak vykonávat stejné testy aplikace po každé její změně. Vytvořil JavaScriptovou knihovnu, která uměla pracovat se stránkou a umožňovala mu spouštět opakovaně testy na různých typech prohlížečů.

Tato knihovna se stala základem nástroje Selenium Core, ze kterého vznikly nástroje Selenium Remote Control (RC) a Selenium IDE. Selenium RC byl ve své době zlomový nástroj, protože žádný jiný nástroj neumožňoval ovládat webový prohlížeč pomocí jazyka, který si vybrete.

I když bylo Selenium silný nástroj, nevyhnulo se problémům. Bylo to kvůli jeho JavaScriptovému jádru a bezpečnostním JavaScriptovým omezením webových prohlížečů. Proto se stali některé úkoly nesplnitelné. Postupem času těchto omezení čím dál více přibývalo.

V roce 2006 pracoval ve společnosti Google inženýr Simon Stewart na projektu, který nazval WebDriver. Google byl dlouhodobě velkým uživatelem Selenia, ale testeři se museli potýkat s jeho omezeními. Simon chtěl nástroj, který by komunikoval přímo s prohlížečem za pomoci nativních metod pro prohlížeč a operační systém. Tím by se vyhnul omezením JavaScriptu. Selenium Webdriver byl vydán v roce 2008.[2]

3.1.2 Nástroje Selenium

Selenium 1 (Selenium RC nebo Remote Control)

Selenium RC byl dlouhou dobu hlavním projektem Selenia dokud nebyl nahrazen silnějším a lepším nástrojem Selenium 2. Nyní už není Selenium 1 aktivně podporováno, je spíše v udržovacím módu. [3]

Selenium-Grid

Selenium-Grid umožňuje nástroji Selenium RC skládat velké testovací scénáře, které musí proběhnout v různých prostředích. Selenium Grid umožňuje spustit testy paralelně, to znamená, že různé testy mohou být spuštěny současně na různých vzdálených PC. [4]

Selenium IDE (Integrated Development Enviroment)

Selenium IDE je nástroj, který slouží k vytváření testovacích scriptů. Je to plugin pro prohlížeč Firefox, který umožňuje snadnou a rychlou tvorbu testů tím, že nahrává uživatelovy akce v prohlížeči a následně je převádí na znovupoužitelný script v jednom z podporovaných programovacích jazyků.

I když je použití tohoto nástroje jednoduché a rychlé, nedoporučuje se používat ho na celý vývoj testů. Tento nástroj totiž nepodporuje větvení pomocí podmínek nebo iterace pomocí cyklů. Další jeho nevýhoda je ve špatné udržitelnosti testovacích scriptů. Pokud se změní aplikace je nutné postižené testy znovu nahrát. Proto se tento nástroj doporučuje pouze pro tvorbu rychlých prototypů a následně se doporučuje použití nástroje Selenium 2. [5]

Selenium 2 (Selenium WebDriver)

Selenium 2 je budoucí směr projektu a nejnovější nástroj rodiny Selenium. Jeho největší výhodou je možnost využití objektově orientovaného API WebDriver pro tvorbu testů. WebDriver je designovaný, tak aby poskytl jednodušší a konzistentnější programovací prostředí.

WebDriver byl vyvinut, aby zlepšil podporu dynamických webových stránek, kde se web elementy na stránce mění, aniž by byla stránka refreshována. Cílem WebDriveru je možnost vytváření dobře designovaných a objektově orientovaných testů, které splní moderní požadavky na testovaní softwaru. [6]

Selenium WebDriver komunikuje s webovým prohlížečem přímo pomocí volání jeho nativních metod pro automatizaci. To, jak jsou tyto volání uskutečňována záleží na použití konkrétního prohlížeče. [7]

3.2 Návrhové vzory

Návrhový vzor je určitý předpis a ověřený způsob, jak by se měl psát kód, aby byl přehledný a snadno udržovatelný.

Výhoda návrhových vzorů je v tom, že pokud je dodržuji velmi mi usnadní práci a každý kdo je zná se snadno zorientuje v mém kódu. To se hodí zejména pokud dostanu práci po jiném programátorovi, nebo musím dělat code review. Zároveň se s jejich použitím zbavím duplicitního kódu.

Jejich nevýhoda je v tom, že ne vždy jsou na první pohled použitelné a pokud je chci používat, tak je musíme používat stále, protože jinak hrozí duplicita kódu a časté chyby.

Návrhových vzorů existuje celá řada, ale pro Selenium jsou nejpoužívanější dva. Page Object a Page Factory.

3.2.1 Page object

Page Object je název návrhového vzoru, který říká, jak správně konstruovat automatické testy webových aplikací.

Jeho princip je založen na tom, že každá webová stránka má pro sebe vytvořenou svou vlastní třídu neboli objekt, ve které jsou napsané všechny metody, které s touto webovou stránkou pracují. Je důležité, aby platila rovnice, že jedna třída obsluhuje pouze jednu webovou stránku. Je to z důvodu, aby nevznikal duplicitní kód.

Duplicitní kód je špatný hlavně z toho důvodu, že pokud dojde ke změně testované aplikace, tak se musí změnit i testy a pokud je kód duplicitní, tak se musí změnit na více místech. To by například nevadilo u malých aplikací, ale u aplikací o velikosti internetového bankovnictví to už může být velmi velký problém.

Pokud jsou ve webové aplikaci webové stránky, které jsou si v určitých ohledech podobné, ale některou funkcionalitu mají odlišnou, tak může vzniknout problém s vytvořením page objectů, tak aby nevznikla duplicita společných funkcí. Tato situace se řeší pomocí dědičnosti. To znamená, že se vytvoří page object, který se stará o obsluhu společncých prvků podobných webových stránek. Potom se vytvoří page objecty pro jednotlivé stránky a ze společného page objectu se zdědí všechny jeho funkce. [8] Tím je zaručeno, že bude vše pouze na jednom místě.

3.2.2 Page Factory

Page Factory je návrhový vzor, který je velmi podobný návrhovému vzoru Page Object. Jeho princip je založený na tom, že každý Page Object má v sobě deklarovány zároveň web elementy dané stránky. Tyto elementy mohou být například tlačítka, popisky, textboxy atd. poté jsou inicializovány pomocí třídy PageFatory.

Pro lokalizaci web elementů lze použít jakýkoliv atribut web elementu, ale nejčastěji používané jsou: ID, ClassName nebo Xpath.

ID

ID je nejpoužívanější způsob lokalizace web elementů, protože ve většině webových aplikacích má každý web element unikátní ID. Problém nastává, pokud jsou jednotlivá ID náhodně generovaná, v tomto případě se lokalizace pomocí ID nedá použít.

```
@FindBy(how=How.ID, using = "user")
WebElement userCombobox;
```

ClassName

ClassName je vhodná alternativa pro web elementy, které nemají přidělené ID, nebo je jejich ID náhodně generované. Problém u ClassName je, že nemusí být unikátní, proto nejde použít ve všech případech.

```
@FindBy(how = How.CLASS_NAME, using="wide")
List<WebElement> emails;
```

Xpath

Xpath je nejuniverzálnější lokátor web elementů. Jeho výhoda je, že se dá použít vždy. Nevýhodou je, že je náročnější na tvorbu. Programátor ho nemůže vyčíst přímo z xml kódu a musí xpath vytvořit sám. Další jeho nevýhodou, je že při změně struktury stránky přestává fungovat. To se stává zejména pokud testujeme aplikaci, na které stále probíhá vývoj.

```
@FindBy(how = How.XPATH,
using="/html/body/div[2]/header/div[2]/span[1]/label/span")
WebElement checkAllCheckbox;
```

Pokud je na stránce více stejných web elementů je výhodné použití listu a nahrání všech najednou

```
@FindBy(how = How.CLASS_NAME, using="wide")
List<WebElement> emails;
```

Nelze říci, který z uvedených lokátorů je nejlepší, vždy záleží na tom, jak je naprogramována testovaná aplikace a výsledný způsob lokalizace web elementů je většinou kombinací výše uvedených způsobů.

Pro vyhledávaní lokátorů web elementů se používá zobrazení xml kódu stránky přímo v internetovém prohlížeči.

3.3 Testování Webobových aplikací

Hodně, možná většina softwarových aplikací je dnes vytvořena na webu a spouští se v internetovém prohlížeči. Tyto aplikace se navíc vyvíjejí velmi rychle, a proto je nutné testovat je už v průběhu jejich vývoje. Testování těchto aplikací velmi závisí na strategii v jednotlivých firmách. Například na tom, jestli firma investuje do automatizace testů. [9]

Testování softwaru je velmi široký pojem. V této práci se zaměřuji na automatizaci regresních testů, proto zde popíšu základy regresního testování a rozdíly mezi manuálním a automatickým testováním.

3.3.1 Regresní testování

Tyto testy mají za úkol ověřit, to že úpravy aplikace tj. opravy respektive přidání nových částí negativně neovlivnilo stávající funkčnost. Jinak řečeno testuje se to, že opravami chyb, respektive novým vývojem nevznikají chyby ve dříve funkčních částech aplikace. [10]

Tyto testy se musí vykonávat po každém zásahu do aplikace, to se děje velmi často, proto se testy musí vykonávat někdy i vícekrát během jednoho dne. Zpravidla jsou, ale testy vykonávány ve větších časových intervalech řádově jednou během několika dní, ale záleží to na intenzitě vývoje testované aplikace. Protože je tato činnost opakovaná je snaha jí automatizovat.

Automatizované vs. Manuální testování

Ne vždy je výhoda automatizovat manuální test casy. Jsou případy, kdy je manuální testování výhodnější. Například pokud se bude v blízké době měnit uživatelské prostředí není důvod automatizovat testy, protože by se museli brzy přepisovat. Jindy například není čas na automatizaci a pro krátkodobé testování je manuální výhodnější. [11] Další příklad toho, kdy se automatizované testování nehodí je testování rozložení jednotlivých prvků na stránce. Samozřejmě se dá napsat test, který má předdefinované pozice web elementů na stránce, ale toto rozložení může být na každém PC jiné. Záleží na rozlišené displeje i na operačním systému.

Naopak výhody automatizované testování nabízí při dlouhodobém testování a dlouhodobém vývoji. Automatizované testy jsou navíc velmi rychlé a nezáleží na tom, kdy běží. Mohou běžet například přes noc nebo během pauzy na oběd a na člověka pak už zbývá jen vyhodnocení výsledků případně reportování chyb.

Jejich největší výhoda je v tom, že šetří čas a peníze. Umožnují testovat velké části aplikace ve velmi krátkém čase. Jsou případy, kdy je nutné vykonat stovky nebo i tisíce testů ve velmi krátkém čase například během jednoho týdne. Pokud by se tento objem testů měl vykonat manuálně muselo by na tom pracovat několik desítek testerů na plný úvazek. To se samozřejmě nevyplatí a v tu chvíli se vyplatí investovat do automatizace.

3.4 Keyword Testing

Keyword testing je moderní přístup k tvorbě automatických testů. Vznikl z důvodu stále větší potřeby automatizace testů. Nicméně pro tvorbu automatických testů je nutné kvalitních testerů, kteří mají zkušenosti jak s testováním, tak se základy programovaní.

Takovýchto testerů je velmi málo, a proto je snaha zjednodušit celý proces vývoje testů, ale zároveň zachovat robustnost a snadnou udržovatelnost, kterou nabízí objektově orientovaný přístup k automatizaci.

Princip keyword testingu je takový, že jsou v programovacím jazyku napsány testovací metody za použití výše uvedených návrhových vzorů. Tyto testovací metody jsou svázány s klíčovými slovy, pomocí kterých může skládat test i tester bez zkušeností s programováním. Tím pádem výsledný test vypadá jako script, kde jsou za sebou seřazeny jednotlivé kroky testu. Pro vytvoření tohoto scriptu navíc není třeba vývojové studio, protože klíčová slova jsou jen v textové podobě. Tím pádem může být script napsaný například jen v .txt souboru, ale nejvíce se používá nástroj excel. Je totiž mnohem jednodušší naučit testera princip keyword testingu a práci s excelem než ho učit programovat.

Testy v excelu jsou přehledné a navíc může být svázáno několik souborů dohromady a usnadnit údržbu testů. Tím pádem může být testovací team složený z jednoho vývojáře automatických testů, který se stará o kód a několika manuálních testerů, kteří skládají dohromady testy mocí keywords, starají se o reporting chyb a jejich retestování. Tím se z velké části redukují náklady na testování.

Další z výhod keyword testingu je to, že kódu je mnohem méně než při klasickém přístupu k vývoji testů. Protože jinak by každý test zapsaný v excelu musel být zapsaný pomocí kódu a to by znamenalo vytvoření několika desítek dalších tříd a stovek dalších metod, samozřejmě to velmi záleží na velikosti testované aplikace a počtu testů. V excelu jsou zároveň uložena všechna data a tím pádem se dají testy velmi snadno parametrizovat. Při klasickém vývoji testů jsou data většinou uložena přímo v testovacích třídách, ale to je velmi nepřehledné. Naopak excel je nástroj pro práci s daty přímo navržený, proto je jeho využití velmi výhodné.

4 Vlastní práce

4.1 Vytvoření projektu

Pro vytvoření projektu jsem použil jazyk Java z toho důvodu, že je v kombinaci s nástrojem Selenium nejpoužívanější a zároveň mám s tímto jazykem nejvíce zkušeností. Celý projekt jsem vytořil v programovacím studiu IntelliJ IDEA Community Edition.

Nejjednodušší možnost, jak vytvořit projekt v Jave pro použití Selenium WebDriver, je použít Maven. Maven stáhne java bindings (Selenium 2.0 java client library) a všechny jeho závislosti. Všechny tyto návaznosti jsou uloženy v souboru pom.xml.

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.4.0</version>
</dependency>
```

4.2 Testovací Třídy

V testovacích třídách jsou uložené všechny test casy. Je důležité, aby byla vytvořena jedna třída pro každý test. Test pak obsahuje několik test casů, podle toho podle toho, jak velkou část aplikace testuje.

4.2.1 Proměnné

Webriver

Webriver driver je nejdůležitější proměnná, protože obsahuje instanci webového prohlížeče, se kterým pracuji.

```
private WebDriver driver;
```

SoftAssert

SoftAssert sa je proměnná, která obsahuje instanci třídy SoftAssert, kterou využívám k ověření méně závažných chyb v aplikaci.

```
private SoftAssert sa;
```

ExcelUtils

ExcelUtils excelUtils je proměnná, která propojuje testovací třídu s třídou, obsahující metody pro práci s excel soubory.

```
private ExcelUtils excelUtils;
```

Keywordutils

Keywordutils keywordUtils je proměnná, která propojuje testovací třídu s třídou obsahující metody, které se starají o exekuci jednotlivých test casů.

```
private KeywordUtils keywordUtils;
```

geckoDriverPath

String geckoDriverPath je textová proměnná, která obsahuje cestu k gecko driveru, který je součástí balíčku Selenium a je nezbytný pro používání ostatních web driverů.

```
private String geckoDriverPath;
```

excelPath

String excelPath je textová proměnná, která obsahuje cestu k excel souboru, ve kterém jsou uloženy jednotlivé test casy

```
private String excelPath;
```

Všechny proměnné v této třídě jsou typu private, aby se s nimi dalo pracovat pouze ve třídě, ve které jsou deklarované.

4.2.2 Nastavovací metody

Pro nastavení testu používám různé metody, podle toho, kdy se má spustit určitá část kódu. To kdy se má metoda zavolat určuji pomocí @anotace Selenia. Hlavní účel těchto metod je to, že obsahují kód, který se spouští velmi často a je nutné mít ho pouze na jednom místě z důvodu údržby.

@BeforeClass

beforeClass je metoda, která se zavolá vždy jako první po spuštění testu a zavolá se pouze jednou. Jejím hlavní účelem je inicializace výše uvedených proměnných, které jsou společné pro všechny testy.

```
@BeforeClass
public void beforeClass() throws Exception {
geckoDriverPath =
"C:\\Users\\lukas\\IdeaProjects\\BpSeznamEmail\\geckodriver.exe";

System.setProperty("webdriver.gecko.driver", geckoDriverPath);

excelPath =
"C:\\Users\\lukas\\IdeaProjects\\BpSeznamEmail\\src\\main\\java\\DataEngi
ne\\LoginTests.xlsx";

excelUtils = new ExcelUtils();
excelUtils.setExcelFile(excelPath);
keywordUtils = new KeywordUtils(excelUtils);
}
```

@BeforeMethod

beforeMethod je metoda, která se zavolá vždy před každou metodou, její opakování tedy záleží na počtu testovacích metod. Jejím hlavním účelem je nastavení proměnných, které musí být pro každý test nastaveny znovu.

```
@BeforeMethod
public void beforeMethod() throws Exception {
    sa = new SoftAssert();
    driver = new FirefoxDriver();
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    keywordUtils.initPages(driver, sa);
}
```

V této metodě zároveň nastavuji vlastnost driveru implicitlyWait. Ta slouží k tomu, aby test neselhal, pokud má stránka špatnou odezvu a načítá se pomalu. Poslední, o co se tato metoda stará je zavolání metody initPages, která inicializuje všechny třídy, které budu při testování potřebovat.

@AfterMethod

afterMethod je metoda, která se zavolá po každé metodě. Její účel je ukončit driver. Je to z toho důvodu, aby měl každý test připravené čisté testovací prostředí a nebyl ovlivněn předchozím testem.

```
@AfterMethod
public void afterMethod()
{
    driver.quit();
}
```

4.2.3 Testovací metody

@Test

Tyto metody slouží ke spouštění jednotlivých test casů. Všechny tyto metody jsou velmi podobné. Obsahují volání metod initTC a performTest které se starají o exekuci test casů. Dále je tu volání metody assertAll, která vyhodnocuje měkké chyby v testu. Volání této metody by na první pohled mělo být umístěno v metodě afterMethod, aby nedocházelo k duplicitě kódu, protože se metoda volá naprosto stejně ve všech testech.

Problém je ten, že umístění volání této metody do metody afterMethod by způsobilo ukončení všech Testů v testovací třídě a to v případě, že by našla chybu v jakémkoli testu. Je to vlastnost Selenia. Pokud dojde k chybám v nastavovacích metodách, žádné testy se nespustí. Je předpoklad, že pokud se nepovedlo nastavit správně testovací prostředí nemůžou přinést testy relevantní výsledky. Tato vlastnost je velmi užitečná, bohužel v tomto případě vznikla dulicita kódu.

```
@Test
public void unsuccessfulLoginTCKeywords() throws Exception {
    keywordUtils.initTC("Unsuccessful_Login");
    keywordUtils.performTest();
    sa.assertAll();
}
```

4.3 Třída ExcelUtils

Do této třídy jsem napsal všechny metody, které se starají o nahrávání jednotlivých keywords, ze kterých se skládají jednotlivé test casy. Tato třída obsahuje několik proměnných, které jsou nezbytné pro připojení excel souboru. Dále jsou tu deklarovány 3 listy, do kterých nahrávám data z excel souboru.

4.3.1 Proměnné

excelWBook

Tato proměnná slouží k zpřístupnění celého excel souboru pro můj program. Je nezbytná pro to, abych mohl připojit konkrétní worksheet s konkrétním test casem.

```
private XSSFWorkbook excelWBook;
```

excelWSheet

Tato proměnná slouží k zpřístupnění worksheetu s konkrétním test casem. Je nezbytná pro to, abych mohl z excel worksheetu nahrát všechna keywords.

```
private XSSFSheet excelWSheet;
```

cell

Tato proměnná slouží k nahrání určité buňky z excel worksheetu. Pomocí této proměnné pak s použitím for cyklu nahrávám celý test case do listu.

```
private XSSFCell cell;
```

testCases

Tento list slouží k nahrání všech worksheetů tj. test casů z excel souboru.

```
private List<String> testCases;
```

usedPages

Tento list slouží k nahrání všech pages tj. tříd z konkrétního worksheetu. V těchto třídách jsou uloženy jednotlivé metody, které se starají o dílčí kroky testu.

```
private List<String> usedPages;
```

testSteps

Tento list slouží k nahrání všech keywords která přímo odpovídají metodám uloženým ve třídách tj. v pages.

```
private List<String> testSteps;
```

parameters

Tento list slouží k nahrání parametrů, se kterými pracují jednotlivé metody.

```
private List<String> parameters;
```

Všechny proměnné i listy v této třídě jsou typu private, aby se s nimi dalo pracovat pouze ve třídě, ve které jsou deklarované.

4.3.2 Metody

getDataFromColumn

Nejdůležitější metoda v této třídě je metoda getDataFromColumn, která se stará o naplnění libovolného listu daty z excel worksheetu.

Metoda má 2 parametry. List typu String, do kterého chceme nahrát data a číslo sloupce ze kterého chceme vybrat data. Metoda funguje tak, že pomocí for cyklu pročítá řádek po řádku excel worksheet a pokud je buňka prázdná přiřadí do listu hodnotu null a pokud prázdná není, tak nahraje její textovou hodnotu do listu. Metoda nepročítá celý worksheet, ale pročítá ho pouze podle toho, jak je dlouhý každý testCase.

Tuto metodu využívá velká většina ostatních metod v této třídě. Proto je také typu private, protože není důvod, aby k ní měli přístup i osatní třídy. (Ukázka metody na další stránce)

```
private List<String> getDataFromColumn(List<String> list, int cellNumber)
throws Exception {
    for (int i = 1; i <= getNumberOfTestSteps(); i++)//i=1 because first
row in excel is for names of columns
    {
        cell = excelWSheet.getRow(i).getCell(cellNumber);
        if (cell == null) {
            list.add(null);
        } else {
            list.add(getStringCellData(i, cellNumber));
        }
    }
    return list;
}
```

getTestSteps

Metoda getTestSteps slouží k získání keywords z excel workSheetu. Nejdříve inicializuji list testSteps a pak do něj pomocí výše uvedené metody getDataFromColumn nahraji data. Metoda je public, protože je volaná z jiných tříd a musí pro ně být viditelná.

```
public List<String> getTestSteps() throws Exception {
    testSteps = new ArrayList<String>();
    testSteps = getDataFromColumn(testSteps, 1);
    return testSteps;
}
```

Ostatní metody v této třídě jsou velmi podobné metodě getTestSteps, proto je zde neuvádím.

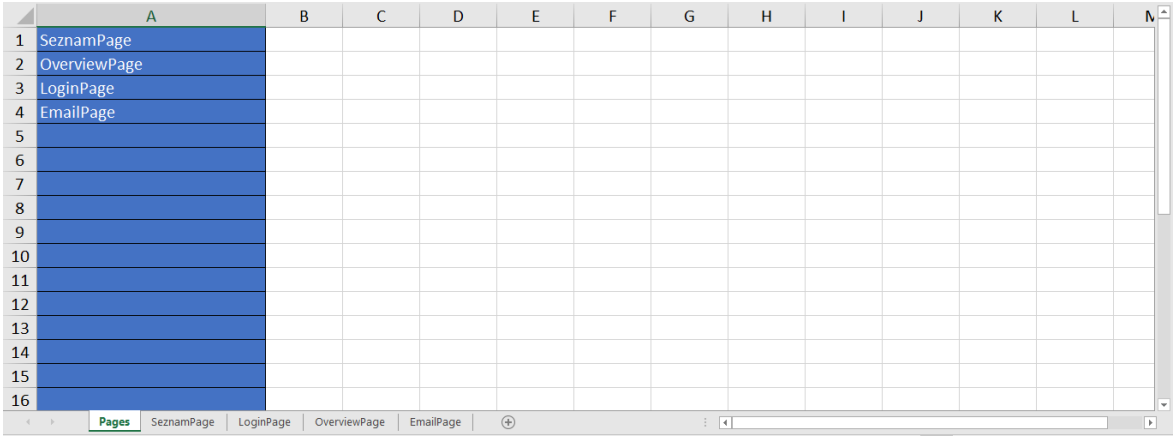
4.4 Excel soubory

Pro uložení jednotlivých testů jsem zvolil systém jednoho centrálního excel souboru, ze kterého ostatní soubory čerpají data. Důvod, proč jsem použil tento systém je ten, že zabraňuje duplicitě a velmi usnadňuje údržbu. Stačí upravit nebo přidat testovací metody do centrálního souboru a všechny ostatní soubory se automaticky upraví také.

U projektu tohoto rozsahu se to může zdát do určité míry zbytečné, protože souborů je velmi málo. Nicméně se tento systém dá použít i u projektů které přesahují desítky nebo i stovky souborů a tam je velmi užitečný.

4.4.1 Pages_and_methods

V tomto souboru jsou uloženy všechny Pages, keywords a parameters, které se pak přes nepřímé odkazy automaticky kopírují do všech ostatních souborů, které s nimi pracují. První list obsahuje všechny Pages tj. třídy, které používají všechny testy.

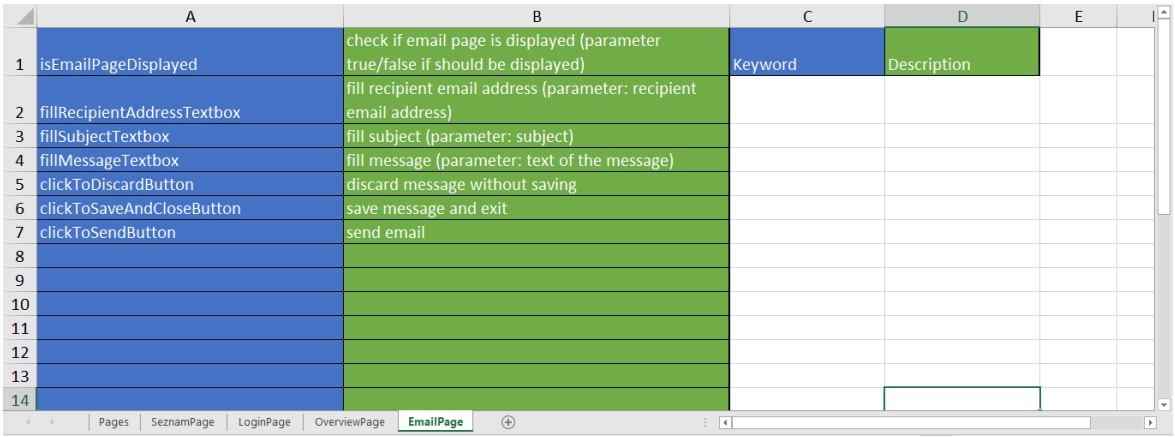


	A
1	SeznamPage
2	OverviewPage
3	LoginPage
4	EmailPage
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	

Pages | SeznamPage | LoginPage | OverviewPage | EmailPage

Obr. 1

Ostatní listy obsahují jednotlivá keywords (modrý sloupec) a jejich popis (zelený sloupec). Tento konkrétní worksheet je EmailPage. Ostatní worksheety jsou velmi podobné, proto je zde neuvádím.



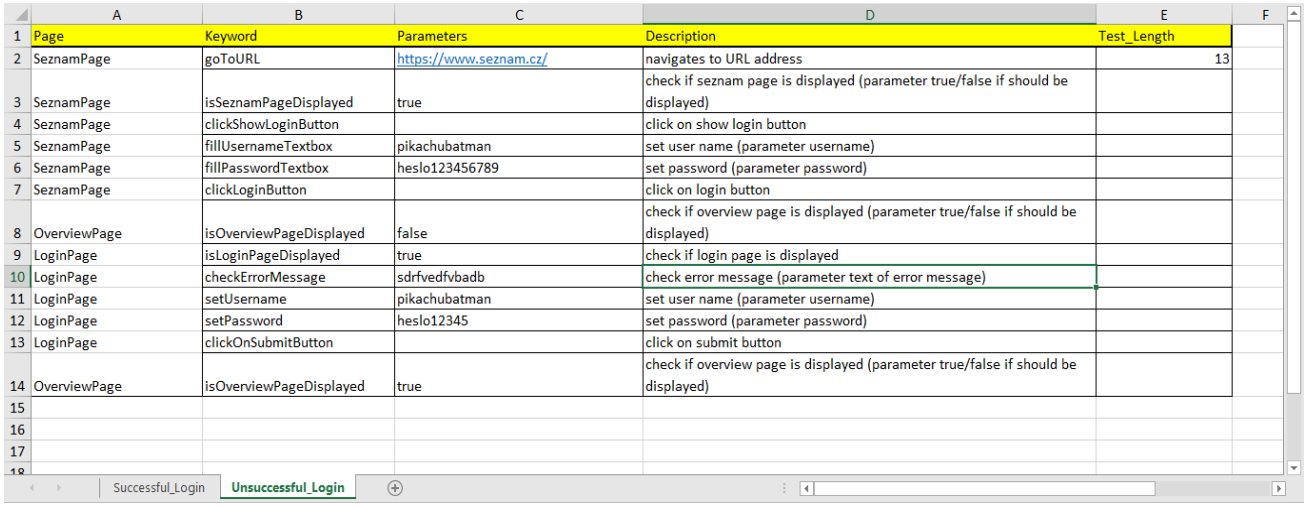
	A	B	C	D
1	isEmailPageDisplayed	check if email page is displayed (parameter true/false if should be displayed)	Keyword	Description
2	fillRecipientAddressTextbox	fill recipient email address (parameter: recipient email address)		
3	fillSubjectTextbox	fill subject (parameter: subject)		
4	fillMessageTextbox	fill message (parameter: text of the message)		
5	clickToDiscardButton	discard message without saving		
6	clickToSaveAndCloseButton	save message and exit		
7	clickToSendButton	send email		
8				
9				
10				
11				
12				
13				
14				

Pages | SeznamPage | LoginPage | OverviewPage | EmailPage

Obr. 2

4.4.2 LoginTests

Tento soubor obsahuje logiku testů, které se starají o testování přihlášení do aplikace. Na obrázku níže je vidět test case Unsuccessful_Login.



	A	B	C	D	E	F
1	Page	Keyword	Parameters	Description	Test_Length	
2	SeznamPage	goToURL	https://www.seznam.cz/	navigates to URL address	13	
3	SeznamPage	isSeznamPageDisplayed	true	check if seznam page is displayed (parameter true/false if should be displayed)		
4	SeznamPage	clickShowLoginButton		click on show login button		
5	SeznamPage	fillUsernameTextbox	pikachubatman	set user name (parameter username)		
6	SeznamPage	fillPasswordTextbox	heslo123456789	set password (parameter password)		
7	SeznamPage	clickLoginButton		click on login button		
8	OverviewPage	isOverviewPageDisplayed	false	check if overview page is displayed (parameter true/false if should be displayed)		
9	LoginPage	isLoginPageDisplayed	true	check if login page is displayed		
10	LoginPage	checkErrorMessage	sdrfvedfvbadb	check error message (parameter text of error message)		
11	LoginPage	setUsername	pikachubatman	set user name (parameter username)		
12	LoginPage	setPassword	heslo12345	set password (parameter password)		
13	LoginPage	clickOnSubmitButton		click on submit button		
14	OverviewPage	isOverviewPageDisplayed	true	check if overview page is displayed (parameter true/false if should be displayed)		
15						
16						
17						

Successful_Login | Unsuccessful_Login

Obr. 3

Sloupec Page

V prvním sloupci je třída, ze které se volá metoda, pro vykonání určité akce na stránce. Název každé Page musí přesně odpovídat určité třídě v java kódu, jinak by hrozilo, že se zavolá třída, která v kódu neexistuje a program selže.

Sloupec Keyword

Ve druhém sloupci je keyword, což je název metody, která se má zavolat. Opět platí, že keyword musí přesně odpovídat existující metodě ve třídě z prvního sloupce, jinak program selže.

Sloupec Parameters

Ve třetím sloupci jsou parametry pro metody. Parametry musí odpovídat typu metody, která je používá. Každá metoda má pouze jeden parametr. To je z toho důvodu, aby nedocházelo ke komplikacím spojeným s pořadím parametrů a jejich datových typem.

Sloupec Description

Ve čtvrtém sloupci je popis každé metody. V něm je krátký popis toho, co metoda dělá a případně jaký parametr má být použit.

Sloupec Test_Length

V posledním sloupci je délka celého testu. Ta je důležitá, proto že určuje, jak dlouhé budou for cykly, které se starají o exekuci test casu.

4.4.3 Propojení Souborů

Výše uvedené problémy s nutností toho, aby obsah buněk přesně odpovídal třídám a metodám v java kódu jsem vyřešil tím, že jsem všechny keywords a pages uložil do comboxů. Obsah comboxů se načítá z centrálního soboru Pages_and_methods pomocí funkce ověření dat a nepřímých odkazů. Tímto způsobem je zaručeno, že do těchto kritických buňek není možné zapisovat, ale pouze vybírat, předem připravené hodnoty.

Připojení Pages

Tento příkaz načte do buňky celý obsah sloupce A ze sešitu Pages ze souboru Pages_and_methods.xlsx. Tím získám všechny třídy, které můžu použít k tvorbě testů.

=NEPŘÍMÝ.ODKAZ("[Pages_and_methods.xlsx]Pages'!$A:$A")

Připojení keywords

Tento příkaz, je velmi podobný, jako předchozí. Rozdíl je pouze v tom, že není přímo určen worksheet, ze kterého se mají vybírat data. Ten je závislý na výběru v předchozím comboboxu. Tím je zaručeno, že se zobrazí pouze keywords, které obsahuje daná Page.

=NEPŘÍMÝ.ODKAZ("[Pages_and_methods.xlsx]"&A2&"'!$A:$A")

Připojení Description

Tento příkaz slouží k připojení popisu jednotlivých metod. Použil jsem funkci SVYHLEDAT, která funguje na principu toho, že znám jednu hodnotu v určité oblasti a k ní hledám jinou, neznámou hodnotu na stejném řádku. V tomto případě znám keyword a k němu hledám description ve sloupcích A a B v souboru Pages_and_methods.xlsx. To, na kterém worksheetu se budou data hledat opět záleží na výběru v předchozích comboboxech. NEPRAVDA na konci příkazu znamená, že hledám přesnou shodu.

=SVYHLEDAT(B2;NEPŘÍMÝ.ODKAZ("[Pages_and_methods.xlsx]"&A2&"'!$A:$B");2;NEPRAVDA)

4.5 Třída KeywordUtils

V této třídě je uložena celá logika, která se stará o exekuci testu. Jsou tu metody pro inicializaci všech tříd a následná volání metod v nich uložených.

4.5.1 Proměnné

excelUtils

Tato proměnná slouží k propojení s třídou ExcelUtils, abych měl přístup k datům z excel souborů.

```
private ExcelUtils excelUtils;
```

method

Tato proměnná slouží k vytvoření instance metody pro vykonání určitého kroku v testu. Je to instance metody zavolané pomocí keywords.

```
private Method testMethod;
```

pageInstaces

V tomto listu jsou uloženy instance všech tříd, ze kterých se volají metody pro vykonání testu. Jsou tu uloženy instance všech pages, ze kterých se volají metody podle keywords.

```
private List<Object> pageInstances;
```

pagesNames

Tento list je obdoba listu usedPages ze třídy ExcelUtils. Obsahuje názvy tříd, které budu používat pro vytvoření testu.

```
private List<String> pagesNames;
```

testSteps

Tento list je obdoba listu testSteps ze třídy ExcelUtils. Obsahuje keywords, podle kterých se vykonává test.

```
private List<String> testSteps;
```

usedParameters

Tento list je obdoba listu parameters ze třídy ExcelUtils. Obsahuje parametry pro metody ze tříd pages.

```
private List<String> usedParameters;
```

Všechny proměnné i listy v této třídě jsou typu private, aby se s nimi dalo pracovat pouze ve třídě, ve které jsou deklarované.

4.5.2 Metody

initTC

Tato metoda slouží k nahrání dat ze třídy ExceUtils do třídy Keywordutils. Nahraje keywords, pages a parameters.

```
public void initTC(String TCName) throws Exception {
    excelUtils.setExcelWSheet(TCName);
    testSteps = excelUtils.getTestSteps();
    pagesNames = excelUtils.getNamesOfUsedPages();
    usedParameters = excelUtils.getParameters();
}
```

initPages

Tato metoda vytvoří instance všech tříd, které slouží k exekuci testu na nahraje je do listu pageInstances.

```
public void initPages(WebDriver driver, SoftAssert sa) throws Exception {
    SeznamPage seznamPage = new SeznamPage(driver, sa);
    OverviewPage overviewPage = new OverviewPage(driver, sa);
    LoginPage loginPage = new LoginPage(driver, sa);
    InboxPage inboxPage = new InboxPage(driver, sa);
    EmailPage emailPage = new EmailPage(driver, sa);

    pageInstances = new ArrayList<Object>();
    pageInstances.add(seznamPage);
    pageInstances.add(overviewPage);
    pageInstances.add(loginPage);
    pageInstances.add(inboxPage);
    pageInstances.add(emailPage);
}
```

performTest

Tato metoda obsahuje celou logiku pro exekuci testu. Skládá se celkem z 2 vnořených for cyklů a 3 vnořených if else podmínek. První for cyklus prohledává list testSteps, druhý for cyklus prohledává list pageInstances.

První if zjišťuje, jestli je na aktuální pozici v listu page instances metoda se stejným názvem, jako textová hodnota v listu pagesNames. Pokud ne druhý for cyklus se zopakuje s následující hodnotou v listu.

Pokud ano, tak program pokračuje na druhý if. Ten zjišťuje, jestli je parametr pro danou metodu null nebo prázdná textová hodnota. To znamená, že zjišťuje, jestli je metoda bezparametrová. Pokud ano nastaví se do proměnné method daná metoda bez deklarace jakéhokoliv parametru.

Pokud ne, tak program pokračuje do else větve, ve které je další if. Ten zjišťuje, jestli je v listu usedParameters na hledané pozici hodnota true, nebo false. To znamená, že zjišťuje, jestli je metoda s parametrem typu Boolean. Pokud ano nastaví se do proměnné method daná metoda s deklarací parametru typu Boolean.

Pokud ne je už jen jediná možnost, jakého typu může být parametr této metody. Je to textový parametr, tj. parametr typu String. Takže se nastaví se do proměnné method daná metoda s deklarací parametru typu String.

Na konci každé if podmínky je metoda invoke, která pomocí Java Reflection Class spustí danou metodu. (Ukázka metody na další stránce)

```
public void performTest() throws Exception {

 for(int i=0; i<testSteps.size(); i++)
 {
   for(int j=0; j<pageInstances.size();j++) //get test methods
   {
    if(pageInstances.get(j).getClass().getSimpleName().equals(pagesNames.get(i)))
     {
       if (usedParameters.get(i)==""||usedParameters.get(i)==null) {
         testMethod=pageInstances.get(j).getClass().getMethod(testSteps.get(i));
         testMethod.invoke(pageInstances.get(j));
       }
       else
       {
         if(usedParameters.get(i).equals("true")||
            usedParameters.get(i).equals("false"))
         {
           testMethod = pageInstances.get(j).
                        getClass().
                        getMethod(testSteps.get(i),Boolean.class);
                        testMethod.invoke(pageInstances.get(j),
                        Boolean.valueOf(usedParameters.get(i)));
         }
         else
         {
           testMethod = pageInstances.get(j).
                        getClass().
                        getMethod(testSteps.get(i),String.class);
                        testMethod.invoke(pageInstances.get(j),
                        usedParameters.get(i));
         }
       }
     }
   }
 }
}
```

4.6 Třída BasicPage

Tato třída slouží ke zpřístupnění těch nejobecnějších a nejpoužívanějších metod pro ostatní třídy, které slouží k exekuci testu. Z toho důvodu je tato třída typu abstract. Není potřeba vytvářet její instanci, protože všechny její metody využívají až třídy, které z ní dědí.

```
public abstract class BasicPage
```

4.6.1 Proměnné

driver

V této třídě mám jen jedinou proměnou a to driver. Slouží hlavně k inicializaci web elementů pomocí konstruktoru.

```
final WebDriver driver;
```

4.6.2 Metody

BasicPage

BasicPage je konstruktor třídy BasicPage. Konstruktor je speciální typ metody, která se spustí vždy při vytvoření instance třídy a musí mít stejný název jako je název třídy. Proto jsou názvy konstruktorů uváděny s velkým písmenem. Všechny ostatní názvy metod začínají malým písmenem.

I když nikdy nebudu vytvářet instanci abstraktní třídy, můžu vytvořit její konstruktor a do něj vložit úseky kódu, které budou využívat konstruktory tříd, které dědí z abstraktní třídy. Tím se zbavím duplicitního kódu, který by tak vznikal v konstruktorech ostatních tříd.

Parametr konstruktoru je driver, který slouží k inicializaci web elementů pomocí třídy PageFactory. První příkaz předá instanci driveru z parametru do privátního driveru v této třídě a druhý příkaz inicializuje elementy. (Ukázka metody na další stránce)

```
public BasicPage(WebDriver driver)
{
    this.driver=driver;
    PageFactory.initElements(driver,this);
}
```

4.6.3 metoda isElementPresent

isElementPresent je metoda, která slouží ke zjištění toho, jestli je hledaný web element zobrazený na stránce. Parametrem této metody je element, u kterého chceme ověřit jeho přítomnost na stránce.

Metoda funguje tak, že zkusí na element zavolá metodu isDisplayed, pokud tento příkaz proběhne bez problémů, vrátí metoda hodnotu true. Pokud ne, odchytí výjimku o neexistujícím elementu a vrátí hodnotu false.

Tato metoda je důležitá proto, že bez ní by se mohlo stát, že program bude chtít kliknout na element, který na stránce není a celý program se ukončí s nejasnou hláškou o neexistujícím elementu. Pomocí této metody můžu ověřit přítomnost elementu na stránce ještě před tím, než s ním začnu pracovat.

```
public boolean isElementPresent(WebElement element)
{
    try {
        element.isDisplayed();
    }catch (NoSuchElementException e)
    {
        return false;
    }
    return true;
}
```

goToURL

goToURL je jednoduchá metoda, která slouží pouze k přechodu na zadanou URL adresu. Jejím parametrem je URL adresa, na kterou chci přejít.

```
public void goToURL(String url)
{
    driver.get(url);
}
```

waitForSeconds

Tato metoda slouží k uspání vlákna programu na určitý počet sekund. Její využití je hlavně v případech, kdy se některý web element načítá příliš dlouho a program by s ním chtěl pracovat, ještě dříve, než bude načtený. Jejím parametrem je počet sekund, na který chceme program uspat.

```
public void waitForSeconds(long seconds){
    try {
        Thread.sleep(1000 * seconds);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

4.7 Třída SeznamPage

Tato třída obsluhuje metody, které slouží k ovládání web elementů na stránce www.seznam.cz. Pro účely tohoto projektu jsou to pouze web elementy týkající se přihlášení do emailu. Názvy všech metod přímo odpovídají jednotlivým keywords z excel souboru LoginTests.xlsx z listu SeznamPage.

Jednou z velkých nevýhod této třídy i tříd tohoto typu, je že se z nich žádné metody ani většina proměnných přímo nevolá nikde v kódu, ale používají se až po zavolání přes třídu KeywordUtils. To znamená, že si vývojové studio myslí, že metody a proměnné jsou nevyužívané a doporučuje tyto úseky kódu smazat. To by však vedlo ke kritickým chybám a smazat se nesmí.

Nicméně se může stát, že postupem času se některé metody a proměnné skutečně stanou nevyužívanými a v tomto případě je velmi těžké odlišit, kterou část kódu můžu opravdu smazat.

Třída má zároveň přístup ke všem metodám třídy BasicPage, protože z ní dědí díky slovu extends v její deklaraci.

```
public class SeznamPage extends BasicPage
```

4.7.1 Proměnné

driver

Proměnná driver slouží k předání instance web driveru z testovacích tříd do abstraktní třídy BasicPage, aby mohli být inicializovány web elementy na stránce.

```
private WebDriver driver;
```

sa

Proměnná sa slouží k nahrání instance třídy SoftAssert. Pomocí této proměnné, tak můžu ověřovat méně závažné chyby v aplikaci.

Například pokud je v aplikaci špatně zobrazený text, není to důvod k použití klasického tvrdého assertu, který funguje tak, že v případě nalezení chyby ukončí celý test. Naopak soft assert funguje tak, že zaznamená chybu, ale test nechá pokračovat dál. Všechny soft asserty se pak vyhodnotí na konci testu viz. kapitola Testovací metody. Tím můžu získat z testu více informací o chybách v aplikaci.

Kdy se použití soft assertu naopak nehodí je například kontrola přítomnosti web elementu na stránce. Pokud bych v tomto případě použil soft assert program by sice zaznamenal chybu, ale by pokračoval dál a snažil by se kliknout na neexistující web element a skončil by výjimkou.

```
private SoftAssert sa;
```

showLoginButton

Tato proměnná slouží k nahrání lokátoru pro web element showLoginButton, který na stránce odpovídá tlačítku „zobrazit přihlášení do emailu“. Konkrétně v tomto případě slouží jako lokátor class name.

```
@FindBy(how=How.CLASS_NAME, using ="show-login-form")
WebElement showLoginButton;
```

Ve třídě je takovýchto proměnných více, ale protože jsou velmi podobné a mají podobnou funkci, tak je tu neuvádím.

4.7.2 Metody

Všechny metody ve třídách, které slouží k obsluze web elementů na stránce by měli splňovat několik základních pravidel.

Metoda by měla dělat pouze jednu činnost a obsluhovat pouze jeden web element. Je to z toho důvodu, aby na sobě byli činnosti testu nezávislé. Pokud bych napsal několik činností do jedné metody tak by při neúspěšné exekuci jedné z nich byli přeskočeny i činnosti následující. Zároveň mám díky tomuto systému absolutní kontrolu nad tím kde se chyba objevila.

Metoda by měla mít pouze jeden parametr. To je z důvodu použití keywords. Pokud bych napsal metody s více parametry, musel bych i do excel souborů zapisovat více parametrů. To by přineslo problémy s pořadím parametrů a hlavně s datovým typem parametrů. Pokud by byl například jeden parametr typu Boolean a druhý typu String, bylo by velmi složité správné zavolání dané metody pomocí invoke Java Reflection class viz. kapitola Metoda performTest.

Metoda by měla obsahovat pouze jeden tvrdý assert. Důvod je ten, že po neúspěšném assertu je test ukončen a dál už nepokračuje, proto by byl při nalezení chyby zbytek metody přeskočen.

Pokud metoda ověřuje úspěšné dokončení akce na stránce, měla by obsahovat parametr typu Boolean, kterým můžu rozlišit, jestli akce měla být úspěšná, nebo ne. Pokud mám například 2 test casy Successful_Login a Unsuccessful_Login, tak hned po přihlášení můžu pomocí jedné metody ověřit 2 opačné události: zobrazení emailové schránky v případě úspěšného přihlášení a nezobrazení emailové schránky v případě neúspěšného přihlášení.

SeznamPage

Tato metoda je konstruktor třídy. Slouží k předání instance driveru do BasicPage a k předání instance třídy SoftAssert z testovacích tříd.

```
public SeznamPage(WebDriver driver, SoftAssert sa)
{
    super(driver);
    this.sa = sa;
}
```

isSeznamPageDisplayed

Tato metoda slouží k ověření toho, jestli se načetla stránka www.seznam.cz. Výstup metody je kombinací toho, jestli se stránka načetla a zároveň toho, jestli měla být načtena.

```
public void isSeznamPageDisplayed(Boolean shouldBeDisplayed)
{
    boolean result = true;
    result&=isElementPresent(seznam);

    if (shouldBeDisplayed)
    {
        Assert.assertTrue(result, "Seznam page is not displayed!");
    }
    else
    {
        Assert.assertFalse(result,
        "Seznam page is displayed but shouldn't be!");
    }
}
```

clickShowLoginButton

Tato metoda slouží ke kliknutí na tlačítko „zobrazit přihlášení do emailu“. Využívá k tomu proměnnou showLoginButton popsanou výše.

U všech metod podobného typu je nejdůležitější použití tvrdého assertu s pomocí metody isElementPresent, aby se program nesnažil kliknout na neexistující web element. Ve zprávě assertu by mělo být podrobně popsáno, o jaký web element se jedná, a na které stránce se nachází. Čím přesnější popis je, tím lépe můžu odhalit kde je v aplikaci chyba.

Ověření toho, jestli je web element přítomný na stránce i když jsem ho deklaroval do proměnných je důležité proto, že testy se používají zejména během vývoje aplikace a velmi často se stává, že se mění umístění web elementů ve struktuře xml, nebo jejich lokátory. Tím pádem se stává daný web element pro program neviditelný, protože jeho lokátor deklarovaný ve třídě je rozdílný od toho reálného na stránce.

Zároveň je na této metodě dobře vidět princip dědičnosti. I když je metoda isElementPresent deklarována ve třídě BasicPage můžu ji bez problému využít i ve této třídě. (Ukázka metody na další stránce)

```
public void clickShowLoginButton()
{
    Assert.assertTrue(isElementPresent(showLoginButton),
"Show login button in Seznam page is not displayed!");
    showLoginButton.click();
}
```

fillUsernameTextBox

Tato metoda slouží k vyplnění textu do textboxu pro email. Jejím parametrem je email, který se načítá z parametrů excel souboru. Opět je důležité ověřit přítomnost web elementu na stránce. Poté se vymaže text z textboxu a nahradí se novým z parametru.

```
public void fillUsernameTextbox(String username)
{
    Assert.assertTrue(isElementPresent(usernameTextBox),
"Username textBox in Seznam page is not displayed!");
    usernameTextBox.clear();
    usernameTextBox.sendKeys(username);
}
```

Ostatní metody v této třídě jsou velmi podobné výše uvedeným, proto je zde uvádět nebudu. Uvedu zde 2 metody z jiných tříd kde je vidět použití soft assert a více parametrů.

checkErrorMessage

Tato metoda je ze třídy LoginPage. Slouží k ověření chybového hlášení na stránce, která se zobrazí po neúspěšném přihlášení do emailu. Je to příklad toho, kde je dobré použít soft assert. Pokud by byla hláška zobrazená v jiné podobě, než je v parametru, není to tak závažná chyba, aby nebylo možné pokračovat v testu dále. Například pokračovat testem úspěšného přihlášení do emailu z této stránky.

```
public void checkErrorMessage(String errorMessage)
{
    String message = loginForm.getText();
    sa.assertTrue(message.contains(errorMessage),
"Error message in Login page is not displayed correctly");
}
```

getEmail

Tato metoda je ze třídy OverviewPage, která slouží k obsluze základních web elementů emailové schránky. Je v ní dobře vidět použití 2 parametrů.

Technicky se jedná pouze o jeden parametr addressAndSubject, ale ve skutečnosti jsou to dva parametry address a subject. Spojené do jednoho parametru jsou proto, že excel soubor s test casem obsahuje jen jeden sloupec pro parametr.

Tato metoda má jako jediná dva parametry, protože jinak by nebylo možné efektivně lokalizovat hledaný email. Není možné hledat email pouze podle odesílatele, protože v je velká šance, že ve schránce bude více emailů od jednoho odesílatele, proto je nutné přidat i informaci o předmětu emailu.

V této metodě je dobře vidět, jak složité je použití více parametrů při použití keywords. Oba parametry musí být stejného datového typu a musí být jasně určen jejich oddělovač, v tomto případě je to čárka. Nejdříve metoda rozdělí parametr na 2 tj. na emailovou adresu a předmět. Poté pomocí for cyklu metoda prochází všechny emaily ve schránce a hledá mezi nimi ten, který odpovídá zadaným parametrům.

Toto je jediná metoda, která přijímá parametry z excel souboru a zároveň je dvouparametrová. Velká většina případů se dá vyřešit jednoparametrovou metodou, ale ani použití dvou parametrů není neřešitelné, je však mnohem složitější. Zejména z toho důvodu, že pokud je parametr v excel souboru zadán ve špatném formátu, celá metoda selže. (Ukázka metody na další stránce)

```
public WebElement getEmail(String addressAndSubject)
{
    String emailDetails;
    WebElement finalEmail=null;

    //delete all whitespaces
    addressAndSubject = addressAndSubject.replaceAll("\\s+","");

    //separete email address
    String emailAddress = addressAndSubject.
                          substring(0, addressAndSubject.indexOf(','));

    //separate subject
    String subject = addressAndSubject.
                     substring(addressAndSubject.indexOf(','));

    //remove "," from beginning of the subject
    subject = subject.replaceAll(",","");
    waitForSeconds(1);
    for (WebElement email: emails) {
        emailDetails = email.getText();
        emailDetails= emailDetails.replaceAll("\\s+","");
        if (emailDetails.contains(emailAddress)&&
            emailDetails.contains(subject))
            {
            finalEmail = email;
            }
    }
    return finalEmail;
}
```

Přímo tato metoda se z excel souboru sice nevolá, ale využívají ji jiné metody volané pomocí keywords, například metoda markEmail.

```
public void markEmail(String addressAndSubject)
{
  WebElement email = getEmail(addressAndSubject);
    if (email!=null)
      {
      email.findElement(By.className(EMAIL_CHECKBOX_CLASS_NAME)).click();
      }
}
```

5 Výsledky a diskuse

5.1 Výsledky

Výsledkem této práce je Java program, který slouží jako ukázka keyword testingu na aplikaci email.seznam.cz. Tento program tak může sloužit, jako návod k tvorbě snadno udržovatelných a robustních testů webových aplikací.

5.2 Diskuse

Výsledkem této práce je sice konkrétní program, ale mnohem důležitější jsou metody, které vedli k jeho vytvoření. Ty se totiž dají použít všeobecně k přístupu k testování nejen webových aplikací.

Do budoucna vidím zlepšení této práce v efektivnějším propojení excel souborů a java programu. Například tak, že by bylo v každém excel souboru tlačítko, kterým by se dal přímo spustit nově vytvořený test, aniž by musel být test spuštěn z vývojového studia.

6 Závěr

V této práci se mi podařilo splnit všechny stanovené cíle. Podařilo se mi vytvořit kompletní systém pro zjednodušení tvorby testů webových aplikací za použití nástroje Selenium Webdriver. V praxi se při dodržení principů popsaných v této práci mohou snížit nároky i náklady na personál, který se stará o testovaní aplikací.

7 Seznam použitých zdrojů

[1] Introducing Selenium. *SeleniumHQ Browser Automation* [online]. 2018 [cit. 2018-03-01]. Dostupné z: https://www.seleniumhq.org/docs/01_introducing_selenium.jsp

[2] Brief History of The Selenium Project. *SeleniumHQ Browser Automation* [online]. 2018 [cit. 2018-03-01]. Dostupné z: https://www.seleniumhq.org/docs/01_introducing_selenium.jsp

[3] Selenium 1 (aka. Selenium RC or Remote Control). *SeleniumHQ Browser Automation* [online]. 2018 [cit. 2018-03-01]. Dostupné z: https://www.seleniumhq.org/docs/01_introducing_selenium.jsp

[4] Selenium-Grid. *SeleniumHQ Browser Automation* [online]. 2018 [cit. 2018-03-01]. Dostupné z: https://www.seleniumhq.org/docs/01_introducing_selenium.jsp

[5] Selenium IDE. *SeleniumHQ Browser Automation* [online]. 2018 [cit. 2018-03-01]. Dostupné z: https://www.seleniumhq.org/docs/01_introducing_selenium.jsp

[6] Introducing WebDriver. *SeleniumHQ Browser Automation* [online]. 2018 [cit. 2018-03-01]. Dostupné z: https://www.seleniumhq.org/docs/03_webdriver.jsp#chapter03-reference

[7] How Does WebDriver 'Drive' the Browser Compared to Selenium-RC?. *SeleniumHQ Browser Automation*[online]. 2018 [cit. 2018-03-01]. Dostupné z: https://www.seleniumhq.org/docs/03_webdriver.jsp#chapter03-reference

[8] *Head First Java, 2nd Edition*. 2nd ed. Sebastopol, Kalifornie, USA: O'Reilly Media, 2005. ISBN 0596009208.

[9] Test Automation for Web Applications. *SeleniumHQ Browser Automation* [online]. 2018 [cit. 2018-03-01]. Dostupné z: https://www.seleniumhq.org/docs/01_introducing_selenium.jsp

[10] Regresní testy. *Testování software* [online]. [cit. 2018-03-01]. Dostupné z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=18:druhy-testovani-v-procesu-vyvoje-sw&catid=3:zaklady&Itemid=11

[11] To Automate or Not to Automate?. *SeleniumHQ Browser Automation* [online]. 2018 [cit. 2018-03-01]. Dostupné z: https://www.seleniumhq.org/docs/01_introducing_selenium.jsp

8 Přílohy

CD s celým projektem je vložené v deskách této práce.