Automatizované testování webových aplikací

Gabriel Ečegi

Bakalářská práce
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Gabriel Ečegi**
Osobní číslo:	**A14084**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**prezenční**
Téma práce:	**Automatizované testování webových aplikací**
Téma anglicky:	**The Automated Testing Of Web Applications**

Zásady pro vypracování:

1. **Popište aktuální způsoby a možnosti testování softwaru.**
2. **Popište dostupné frameworky a nástroje pro testování webových aplikací se zaměřením na framework Selenium.**
3. **Navrhněte a integrujte automatizované testy založené na frameworku Selenium do platformy "beTested" od společnosti Business Logic.**
4. **Navrhněte a popište vhodný způsob použití implementace automatizovaných testů.**
5. **Zhodnoťte přínosy nově vzniklého testovacího sytému a formulujte závěry.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **NAGEL, Christian. Professional C# 6 and .Net Core 1.0. Indianapolis, IN: Wrox, a Wiley brand, published by John Wiley & Sons, 2016. ISBN 111909660X**
2. **Selenium Documentation [online]. Selenium Project, 2017 [cit. 2017-01-30]. Dostupné z: http://www.seleniumhq.org/docs/**
3. **ASP.NET | Microsoft Docs [online]. Microsoft, 2017 [cit. 2017-01-30]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/**
4. **FREEMAN, Adam. Pro ASP.NET MVC 5. 5th ed. New York, N.Y.: Apress, 2013. ISBN 978-143-0265-306.**
5. **GRAHAM, Dorothy a Mark FEWSTER. Experiences of test automation: case studies of software test automation. Upper Saddle River, NJ: Addison-Wesley, c2012. ISBN 03-217-5406-9.**

Vedoucí bakalářské práce: **Ing. Milan Navrátil, Ph.D.**
Ústav elektroniky a měření

Datum zadání bakalářské práce: **24. února 2017**

Termín odevzdání bakalářské práce: **24. května 2017**

Ve Zlíně dne 24. února 2017

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Jméno, příjmení: Gabriel Ečegi

Název bakalářské/diplomové práce: Automatizované testování webových aplikací

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 13.5.2017

Ečegi
.....
podpis diplomanta

ABSTRAKT

Témou tejto bakalárskej práce je popis moderného prístupu k testovaniu webových aplikácií. V teoretickej časti sú popísané základy z oblasti testovania softvéru, ako aj možnosti testovania webových aplikácií so zameraním framework Selenium. V praktickej časti je popis implementácie automatizovaného testovania webových aplikácií. Cieľom tejto práce je riešenie problematiky automatizovaného testovania a jeho implementácia.

Kľúčové slová:

Selenium Web-Driver, Page Object Pattern, automatizácia testovanie softvéru, .NET, C#

ABSTRACT

The topic of this Bachelor's thesis is a description of modern approach to web application testing. The theoretical part describes the basics of software testing as well as the testing of web applications with focus on the Selenium framework. Practical part describes the implementation of automated testing of web applications. The goal of this thesis is designing and implementing a solution to problematics of automated testing.

Keywords:

Selenium Web-Driver, Page Object Pattern, automation of software testing, .NET, C#

Týmto sa chcem poďakovať Ing. Milanovi Navrátilovi, Ph.D. za odborné rady, organizáciu a konzultáciu bakalárskej práce.

“The bitterness of poor quality remains long after the sweetness of low price is forgotten.”
- Benjamin Franklin

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **10**

I TEORETICKÁ ČÁST **11**

1 ÚVOD K TESTOVANIU APLIKÁCIÍ **12**

1.1 VÝSTUP TESTOVANIA 12

1.2 NASADENIE TESTOVANIA 12

2 ÚROVNE TESTOV **14**

2.1 JEDNOTKOVÉ TESTOVANIE (UNIT TESTING) 14

2.2 INTEGRAČNÉ TESTOVANIE 14

2.3 SYSTÉMOVÉ TESTOVANIE 15

2.4 AKCEPTAČNÉ TESTOVANIE 15

3 NÁVRH TESTOV **16**

3.1 PRÍPAD POUŽITIA - USE CASE 16

3.2 SADA PRÍPADOV POUŽITIA – USE CASE SET 16

3.3 TESTOVACIE PRÍPADY – TEST CASE 17

3.4 SADA TESTOVACÍCH PRÍPADOV – TEST CASE SUITE 17

3.5 TESTOVACÍ KROK – TEST STEP 17

4 AUTOMATIZÁCIA TESTOVANIA **19**

4.1 CIELE PRI NASADENÍ AUTOMATIZOVANÉHO TESTOVANIE 19

4.1.1 Zvýšiť dôveru v kvalitu softvéru 19

4.1.2 Skoršie vydanie softvéru 19

4.1.3 Zníženie za cenu testovania 20

4.1.4 Konzistentné opakovateľné testovanie 20

4.1.5 Spúšťanie testov bez osobnej prítomnosti 20

4.1.6 Nájdenie regresných bugov 21

4.1.7 Častejšie spúšťanie testov 21

4.1.8 Softvér lepšej kvality 21

4.1.9 Možnosť otestovať viac softvéru 21

4.1.10 Testovanie na rôznych platformách 21

4.1.11 Znovupoužiteľnosť 21

4.1.12 Metriky 22

4.2 REGRESNÉ TESTOVANIE 22

4.2.1 Výber testovacích prípadov pre regresné testy 22

5 NÁSTROJE PRE AUTOMATIZOVANÉ TESTOVANIE WEBOVÝCH APLIKÁCIÍ **23**

5.1 RECORD-PLAYBACK IDE 23

5.2 TESTOVANIE POMOCOU HEADLESS PREHLIADAČOV 23

5.3 FRAMEWORKY VYUŽÍVAJÚCE JEDNODUCHOSŤ HEADLESS PREHLIADAČOV 24

5.4 TESTOVACIE FRAMEWORKY NA CLOUDE – SCREENSTER 24

6 SELENIUM **25**

6.1 SELENIUM IDE 25

6.1.1 Nevýhody Selenium IDE 25

6.2 SELENIUM 1 (SELENIUM REMOTE CONTROL) 26
6.3 SELENIUM 2 (SELENIUM-WEBDRIVER) 26
6.4 SELENIUM-WEBDRIVER API 26
6.4.1 Metóda FindElement 27
6.4.2 Trieda By 27
6.4.3 Metóda FindElements 28
6.4.4 Metóda Navigate 28
6.4.5 Metóda Manage 28
6.4.6 Metóda SwitchTo 28
6.4.7 Metóda Close 29
6.4.8 Metóda Quit 29
6.5 PAGE OBJECT PATTERN 29
6.5.1 Logika testov pri použití Page Object Pattern 29
6.5.2 Návrhový vzor Factory pri Page Object Pattern 30
6.6 SELENIUM GRID 31
6.6.1 Hub 32
6.6.2 Uzol (node) 32
II PRAKTICKÁ ČÁST 33
7 ÚVOD DO PRAKTICKEJ ČASTI 34
7.1 BETESTED 34
7.2 ARCHITEKTÚRA RIEŠENIA PRAKTICKEJ ČASTI 34
7.3 BUSINESS FLOW BETESTED.AUTOMATION 35
7.4 TECHNOLÓGIE POUŽITÉ V PRAKTICKEJ ČASTI 35
7.5 KONFIGURAČNÝ SÚBOR 36
7.5.1 Objekt beTested 37
7.5.2 Objekt driver 38
8 BLOGIC.BETESTED.AUTOMATION.CORE 39
8.1 MODUL APP 39
8.1.1 Attributes 39
8.1.2 Extensions 40
8.2 MODUL CORE 41
8.3 DTOS 41
8.4 PROVIDERS 41
9 BLOGIC.BETESTED.AUTOMATION.API 44
9.1 SWAGGER 46
10 BLOGIC.MODULES.SELENIUM 48
10.1 CORE 48
10.1.1 Browser 49
10.1.2 SeleniumDriverExtensions 49
10.1.3 Metóda GetElementBySeleniumAttribute 49
10.1.4 Metóda GetElementByAttribute 49
10.1.5 Metóda ClickSafely 50
10.1.6 Metóda GoToUrlSafely 50
10.1.7 Metóda SetTextSafely 50
10.1.8 SeleniumElementExtensions 50

10.1.9 Metóda ClickSafely 50
10.1.10 Metóda GoToUrlSafely 51
10.1.11 Metóda SetTextSafely 51
10.1.12 Metóda GetTextFromElement 51
10.1.13 SeleniumAssert 51
10.1.14 Metóda IsOnUrl 51
10.1.15 Metóda ElementIsNull 52
10.1.16 Metóda ElementIsNotNull 52
10.1.17 Metóda TextEquals 52
11 NUGET BALÍČKY 53
11.1 .NUSPEC SÚBOR 53
11.2 INŠTALÁCIA NUGET BALÍČKA 55
11.2.1 Package Manager Console 55
11.2.2 NuGet Package Manager UI 55
11.3 POUŽITIE BETESTED AUTOMATION 57
11.4 ODPORÚČANÝ POSTUP NASADENIA AUTOMATIZOVANÉHO TESTOVANIA 57
11.5 ODPORÚČANÉ POŽIADAVKY PRE VÝVOJ 57
11.6 VYTVORENIE ŠTRUKTÚRY NA WEBE BETESTED 57
11.7 VYTVORENIE PROJEKTU 59
11.8 STIAHNUTIE NUGET BALÍČKOV 60
11.9 VYTVORENIE ŠTRUKTÚRY PROJEKTU 60
11.9.1 Generating 60
11.9.2 Tests 61
11.10 VYGENEROVANIE MODELOVÉHO PROJEKTU 61
11.11 VYTVORENIE VZOROVÉHO TESTU 61
11.12 VYUŽITIE PAGE OBJECT PATTERN V TESTOCH 63
11.13 ANALÝZA TESTOV NA WEBE BETESTED 63
ZÁVĚR 65
SEZNAM POUŽITÉ LITERATURY 66
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 70
SEZNAM OBRÁZKŮ 71
SEZNAM PŘÍLOH 72

ÚVOD

Pri vývoji aplikácií sa team vyvíjajúci softvér musí vysporiadať s nástrahami vývoja. Jeden z najväčších problémov sú chyby spôsobujúce nefunkčné časti aplikácií. Môžu byť spôsobené rôznymi faktormi, ktoré sa objavujú pri rôznych fázach vývoja.

Softvérové spoločnosti sa tieto faktory snažia eliminovať testovaním softvéru pred nasadením do produkcie, vyvíjaný softvér tak býva pokrytý rôznymi druhmi testov.

Pri mojej praxi pri vývoji softvéru som sa stretol s problémom, keď pri vývoji nového modulu aplikácie prestala fungovať časť iného modulu. Tieto problémy by mali pokrývať regresné testy, ktoré sú však časovo veľmi náročné a preto by mali byť z veľkej časti automatizované.

Na trhu existuje množstvo nástrojov snažiacich sa o automatizáciu testovania softvéru, ponúkajú rôzne prístupy k problému. Jedným s najvyužívanejších je framework Selenium, ktorý ma zaujal natoľko, že som sa začal zaujímať o vhodný spôsob jeho využitia.

Teoretická časť sa venuje problematike rozdelenia a návrhu testov, ďalej popisuje existujúce frameworky umožňujúce automatizáciu testov. Teoretická časť sa bližšie venuje aj frameworku Selenium zo zameraním na Selenium Web-Driver.

V praktickej časti je popis vytvoreného riešenia na platforme .NET, snažiace sa o čo najjednoduchšie napojenie testovacieho projektu na projekt, ktorý má byť testovaný. Výstup práce je postavení na platforme .NET, poskytujúcej robustné riešenia pre množstvo odvetví softvérového inžinierstva.

I. TEORETICKÁ ČÁST

1 ÚVOD K TESTOVANIU APLIKÁCIÍ

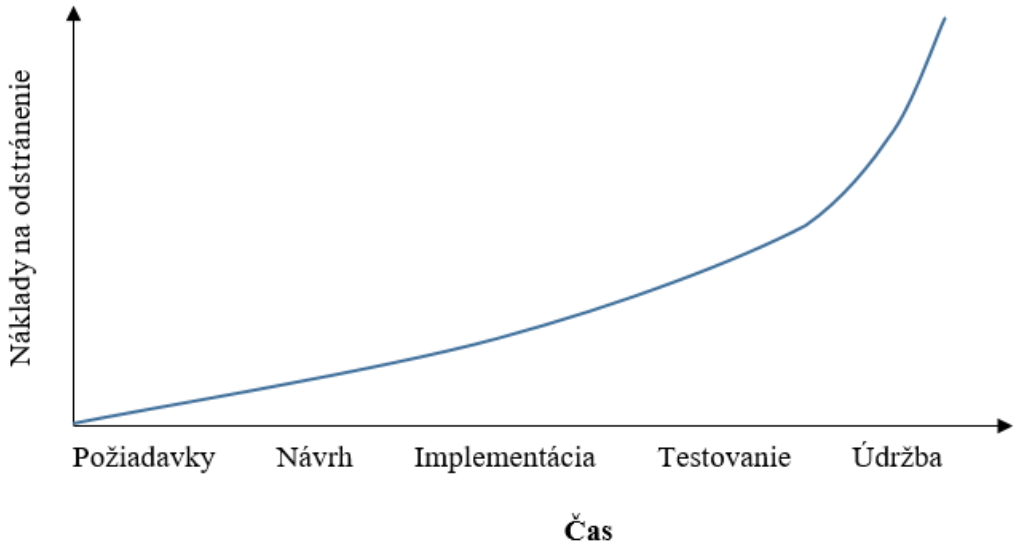
Softvér nie je nič, čo ľudská spoločnosť počas svojej existencie poznala, je to niečo medzi matematikou, jazykom, myšlienkou a informáciou. Jeho vývoj sprevádza nejednoznačnosť, predpoklady a komunikácia. Každá nová funkcionalita alebo pokus o opravu môže otvoriť dvere problému. Riešenie týchto problémov je testovanie softvéru, umožňuje získať dôkaz o tom, že softvér funguje správne [40].

1.1 Výstup testovania

Testovanie softvéru je proces riadeného spúšťania softvérového produktu s cieľom zistiť, či spĺňa špecifikované alebo implicitné potreby užívateľov. Testovanie je riadený proces, pretože musí byť vždy prevádzané s určitým zámerom a predchádza mu plánovanie vo forme analýzy procesov v aplikácii, návrhu testov. Po ňom nasleduje vyhodnotenie výsledkov testov. Testovanie takto získava informácie o kvalite, ktorú rozumieme ako stupeň naplnenia požiadaviek a prianí používateľov. Obsahom testovania je zber a analýza informácií , pričom nájdené defekty sú z tohto pohľadu vedľajším produktom tejto činnosti. Toto sa odlišuje od starších definícii testovania, v ktorých testovanie vystupovalo ako proces z čistým zámerom preukázateľnosti chybovosti produktu. Testovanie samo o sebe priamo nezvyšuje kvalitu produktu, ale poskytuje pre túto činnosť potrebné vstupy [40].

1.2 Nasadenie testovania

Testovanie sa týka aj ekonomickej stránky vývoja softvéru, je kompromisom medzi kvalitou, časovým a rozpočtovým obmedzením, uprednostňovanie jedného cieľa má za následok negatívny dopad na dva ostatné. Testovanie by malo začínať v čo najskorších fázach projektu, už v špecifikácii požiadaviek, pretože v skorších fázach je na odstránenie problému potreba minimálneho úsilia. Platí tak že čím skôr je defekt odhalený, tým nižšie sú náklady na jeho odstránenie. Proces testovania je potencionálne nekonečný a o jeho zastavení tak musí byť rozhodnuté na základe uváženia rôznych faktorov, ako napríklad jeho znižujúca sa efektivita, časové a rozpočtové obmedzenie, vykonanie všetkých kľúčových testov a podobne [40].



Obrázok 1 Vzťah medzi fázou, v ktorej je defekt identifikovaný a cenou za jeho odstránenie [40]

2 ÚROVNE TESTOV

Predtým, ako softvérová spoločnosť vydá aplikáciu, softvér by mal prejsť testovacím procesom aby sa zaistilo, že aplikácia pracuje tak, ako bolo zamýšľané [41]. Tento testovací proces rozdeľujeme do štyroch úrovní, pre lepšie identifikovanie chýbajúcich častí a prevenciu prelínania a opakovania medzi vývojovými fázami cyklu. Tieto cykly sú: analýza, návrh, implementácia, testovanie a nasadenie. Kvôli viacerým cyklom existuje aj viacero úrovní testovania - jednotkové testovanie (unit testing), integračné testovanie (integration testing), systémové testovanie (system testing) a akceptačné testovanie (acceptance testing) [42].

2.1 Jednotkové testovanie (Unit testing)

Počas prvej fázy testovanie sa aplikácia prekladá na hodnotenie, ktoré sa zameriava na konkrétne jednotky alebo komponenty softvéru, aby sa zistilo, či je každý z nich plne funkčný. Hlavným cieľom tohto správania je určiť, či je správanie funkcií aplikácie také ako bolo navrhnuté. Týmto jednotkám možno porozumieť ako najmenším testovateľným súčastiam programu. Podľa typu programovacieho jazyka môžu byť jednotkami funkcie, procedúry, metódy alebo triedy [41] [31].

Pri testovaní jednotiek si treba dať pozor na to, aby boli testované jednotky izolované od závislostí na ostatných jednotkách a takto mohli dokázať, že za výstup je zodpovedná iba konkrétna jednotka. Pri izolácii jednotiek môže byť náročné získať zdroje bez toho aby izolácia bola porušená, preto sú vyžívané programy a knižnice ktoré tieto zdroje simulujú - tieto objekty sa nazývajú mocks alebo stubs (makety, imitátory) [31].

Je normálne, že softvéroví vývojári vykonávajú jednotkové testovanie pred dodaním ich práce testerom na formálne testovanie. Vývojári testujú tieto jednotky dátami, ktoré sú rozdielne od dáta použitých testermi [41] [43].

2.2 Integračné testovanie

Je definované ako testovanie kombinovaných častí aplikácie pre získanie informácií o ich korektnom správaní. Keď všetky jednotky, respektíve moduly fungujú správne, ďalším krokom je ich integrácia do stabilného systému ako celku. Do tejto fáze je možné vstúpiť ak existujú aspoň dva moduly, ktorých integrácia poskytne určitú službu. Spôsobov ako vykonávať integračné testovanie je viacero:

Big bang: všetky moduly sú integrované zároveň v jednom kroku, čo je jednoduché, ale v prípade objavenia problému je nájdenie takéhoto problému zložité, pretože sa môže nachádzať kdekoľvek.

Top down: z hľadiska hierarchickej štruktúry začíname najvyšším modulom a postupujeme pridávaním nižších.

Bottom up: pri tomto prístupe integrácia začína u najnižších modulov a postupuje vyššie - systém nebude fungovať pokiaľ nebude integrovaný najvyšší modul.

Combined / sandwich: kombinuje prístup bottom up a top down. Testovanie môže byť prevádzané na hornej a spodnej úrovni smerom do stredu [31] [41].

2.3 Systémové testovanie

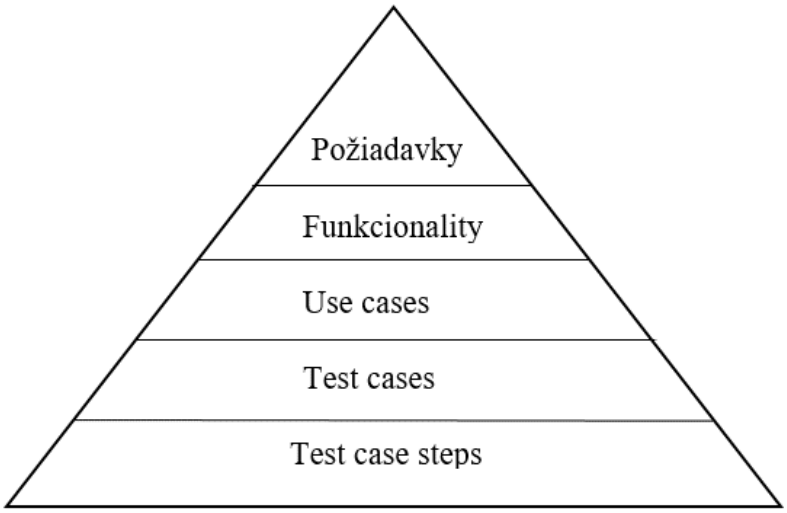
Systémové testovanie je prvou úrovňou, kde môže byť aplikácia otestovaná vcelku. Je vykonávané po tom, ako je výsledok integračného testovania úspešný. Cieľom je vyhodnotiť, či aplikácia dosiahla všetky požiadavky kvality. Systémové testovanie je prebrané nezávislými testermi, ktorí nehrali žiadnu úlohu pri vývoji softvéru. Testy sú vykonávané v prostredí podobnom produkcii, toto testovanie je veľmi dôležité, pretože overuje či aplikácia spĺňa technické, funkcionálne a business požiadavky určené zákazníkom [41].

2.4 Akceptačné testovanie

Poslednou fázou testovania je akceptačné testovanie vykonávané pre určenie, či je systém pripravený pre produkčné prostredie, tzn. či aplikácia splňuje akceptačné kritéria, ktoré sú stanovené zákazníkom, ako merateľné a overiteľné podmienky pre prijatie produktu. Testovanie vykonávajú používatelia zo strany zákazníka na všetkých súčastiach systému. Po tom, ako tento proces skončí a jeho výsledky sú úspešné, aplikácia bude vystavená na produkciu.

3 NÁVRH TESTOV

Pri návrhu a písaní testovacích scenárov sa používateľ musí zoznámiť s problematikou prípadov použitia, testovacími scenármi a testovacími prípadmi, aby mohol efektívne vytvárať testy. Vzťahy sú zobrazené na obrázku č. 2 a obrázku 3 [45].



Obrázok 2 Hierarchické zobrazenie štruktúry testov [45]

3.1 Prípad použitia - Use case

Prípad použitia slúži k popisu určitej funkcionality, respektíve spôsobu použitia. Tento model popisuje funkcionalitu ako interakciu medzi systémom a používateľmi pri dosahovaní konkrétneho cieľa. Súbor prípadov použitia tak zachytáva kompletnú funkčnosť systému a predstavuje funkčnú špecifikáciu [31].

Okrem toho prípady použitia mapujú mnohé scenáre. Mapovanie prípadov použitia na testovacie scenáre je vzťah 1:M [45].

3.2 Sada prípadov použitia – Use case set

Prípady použitia sa združujú do sád, nazývaných sady prípadov použitia. V tejto entite sa nachádzajú prípady použitia s rovnakého modulu aplikácie.

Príklad:

Sada prípadov použitia: Autentifikácia

Prípady použitia spadajúce do tejto sady: Prihlásenie, Registrácia, Zabudnuté heslo, ...

3.3 Testovacie prípady – Test case

Podľa IEEE 1012:2004 je testovací prípad definovaný ako sada vstupov, podmienok pre spustenie a očakávaných výsledkov, vyvinutá pre konkrétny účel, ako je vykonanie špecifickej cesty v programe alebo overenia súladu s konkrétnou požiadavkou [31].

Testovacie prípady sa používajú pre zistenie či boli požiadavky správne implementované. Testovaním prípadu použitia je vytvorený testovací prípad, pre každý prípad použitia by mal existovať aspoň jeden testovací prípad - pre optimálne použitie požiadavky.

Ďalej môžu existovať testovacie prípady pre prípad, kedy program vyhodí výnimku a pre každú alternatívnu cestu. Týchto testovacích scenárov môže byť v závislosti na komplexnosti prípadu použitia až nekonečne veľa.

Testovacie prípady by mali byť čo najjednoduchšie na vykonanie, nemali by obsahovať žiadne podmienky. Keď chceme do testovacieho scenára implementovať podmienku, tak riešením je vytvoriť ďalší testovací scenár s inými testovacími krokmi [46].

3.4 Sada testovacích prípadov – Test case suite

Testovacie prípady za zoskupujú do sád testovacích prípadov. Sada testovacích prípadov by mala odpovedať jednému use casu.

Príklad:

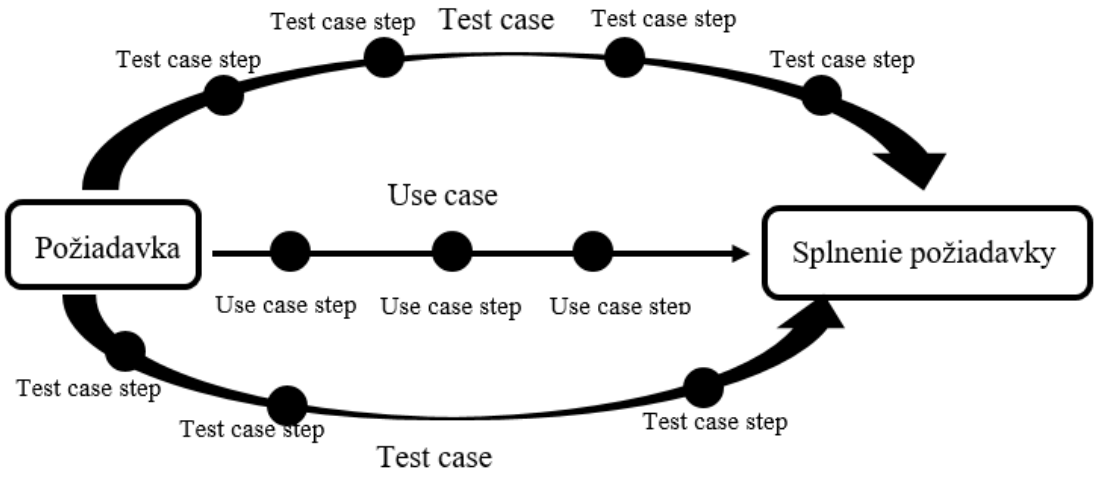
Prípad použitia: Prihlásenie

Sada testovacích prípadov: Prihlásenie

Testovací prípad: Validné prihlásenie, Pokus o prihlásenie bez zadania hesla, Pokus o prihlásenie zo zlým heslom, Pokus o prihlásenie na zablokovaný účet

3.5 Testovací krok – Test step

Testovacie kroky sú základné jednotky testovacích scenárov, je potrebné aby tieto jednotky boli čo najviac atomické. Prvým krokom testovacieho scenára musí byť čo najprístupnejšie miesto, ďalšími krokmi sú presne definované príkazy, ktoré má tester vykonať, aby dosiahol výsledku testovacieho scenára.



Obrázok 3 Zobrazenie vzťahu testovacích entít

4 AUTOMATIZÁCIA TESTOVANIA

Automatizácia je spôsob spúšťania testovania, znamená používať softvérový nástroj na spúšťanie opakovateľných testov. Nesnaží sa nahradiť všetky manuálne testy, iba spojiť manuálne a automatizované testovanie pre dosiahnutie nižšie popísaných cieľov. Môže zlepšiť proces vývoja v množstve prípadov, na začiatku je síce potrebné zainvestovať do jej nasedenia viac času a úsilia, ale pri správnom postupe za dosiahnutím reálnych cieľov sa táto investícia oplatí [39].

Testovanie aplikácie by sa malo začať automatizovať od procesov, ktorých automatizácia zaberie najmenej času, postupne by sa mali začať pridávať zložitejšie procesy a procesy určené analytikom ako vhodné. Pokrytie celej aplikácie je nereálne z ekonomických a časových nárokov [39].

V súčasnosti záujem o automatizované testovanie stále rastie, čo má za následok, že sa nástroje a techniky začínajú rapídne rozvíjať [31].

4.1 Ciele pri nasadení automatizovaného testovanie

Pri vytváraní testovacieho prostredia snažiaceho sa o automatizáciu testov sa väčšina spoločností tieto ciele snaží dosiahnuť. Takéto prostredie na testovanie sa neskladá len z písania testov, ale aj zo zmeny filozofie spoločnosti a kontinuálnom získavaní vedomostí zamestnancov o automatizovanom testovaní softvéru. Tieto ciele sú optimistické a nie vždy sa ich podarí dosiahnuť [30].

4.1.1 **Zvýšiť dôveru v kvalitu softvéru**

Tým, že je softvérové riešenie pokryté testami, je možné sa za kvalitu zaručiť. Po dokončení všetkých testov je možné tvrdiť, že sa v softvéri nenachádzajú chyby. Avšak treba dbať na kvalitu a kontrolu testov. Test, ktorý je zle napísaný a neodhalí chybu na prvý krát, ju neodhalí zakaždým, keď bude spustená celá testovacia sada. V krajných prípadoch môže nastať situácia, kedy vývojári napíšu testy tak, aby boli zakaždým vyhodnotené kladne, preto by sa kvalita testov a ich výsledky mali pravidelne kontrolovať [38].

4.1.2 **Skoršie vydanie softvéru**

Pri včasnom objavení chýb automatizovanými testami je vývojový team na tieto nedostatky upozornený a môže včasne vydať chybnú opravu, ešte vo fáze vývoja. Pri tvorbe testov môže

byť prekvapivo zložité vytvoriť komplexné testovacie sady. Správnosť a komplexnosť testovacích sád by mala mať na starosti zodpovedná osoba - manažér projektu, QA špecialista, analytik testovania, atď. Môžu nastať aj technické problémy spojené s testami, ako napríklad zmeny v aplikácii, problémy s nepredpokladaným správaní testovacej aplikácie, problémy s pripojením, alebo dokonca chyby v testovacom frameworku. Tieto komplikácie môžu mať za následok presne opačný efekt ako bol požadovaný - predĺženie doby dodania softvéru klientovi [38].

4.1.3 **Zníženie za cenu testovania**

Pri pokrytí softvérového riešenia automatizovanými testami, vedúci vývoja nemusí testerovi zadať na otestovanie časti softvéru, na ktorom sa v súčasnej dobe nepracuje, alebo môže byť chyba objavená ešte pred testovaním a tester tak nemusí strácať čas. Do tejto ceny však treba započítať cenu za vývoj automatizovaných testov, cenu za spúšťanie testov, cenu za údržbu testov pri implementácii nových zmien, alebo úprave súčasnej funkcionality. K tejto cene musia byť započítané náklady aj za novovzniknutú prácu spojenú z automatizáciou, napríklad dokumentácia testovacích prípadov, dokumentácia automatizácie, kontrola výsledkov testov, ale aj analýza zmien v aplikácii vedúca k zmene testov [38].

4.1.4 **Konzistentné opakovateľné testovanie**

Oproti manuálnemu testovaniu, kedy tester môže pri testovaní urobiť chybu, alebo nemusí daný bug nájsť, sa automatizovaný test vždy striktne drží testovacieho scenára. Testovanie je možné definovať ako postupnosť používateľských vstupov prekladaných vyhodnoteniami. Tieto postupy môžu byť zložité a krehké, ich narušenie spôsobí nefunkčnosť testu. Naopak, tester sa pri manuálnom testovaní dokáže vysporiadať s komplexnosťou a výsledkami, ktoré by test nedokázal správne vyhodnotiť. Preto by mala byť vypracovaná analýza spracúvajúca procesy v aplikácii a určujúca, ktoré procesy by bolo vhodné automatizovať [30].

4.1.5 **Spúšťanie testov bez osobnej prítomnosti**

Testy môžu byť spúšťané v pravidelných intervaloch (denne, týždenne, mesačne), na vzdialenom servery a zodpovednej osobe príde len výpis o priebehu testov. Tento výpis výsledkov musí byť spracovaný a v prípade chýb diagnostikovaný programátorom alebo analytikom. V takomto prípade však môže byť časovo náročné pripraviť prostredie pre vzdialené testovanie [30].

4.1.6 Nájdenie regresných bugov

Pri väčšom softvérovom riešení je časovo veľmi náročné otestovať funkčnosť všetkých časti aplikácie. Pri vývoji sa tak môže stať, že novovytvorená funkcionalita ovplyvní správny výsledok v staršej časti a vytvorí tak bug. Pri pokrytí regresnými automatizovanými testami takto môžeme otestovať všetky predchádzajúce časti softvéru [30].

4.1.7 Častejšie spúšťanie testov

V prípade, že spúšťanie testov je takisto automatizované, náklady na spustenie automatizovaných testov sú takmer nulové, je umožnené častejšie spúšťanie testov.

4.1.8 Softvér lepšej kvality

Tým, že chyby nájde automatizovaný proces, sa zabraňuje objaveniu takýchto chýb na produkcii, alebo iným vývojárom pracujúcim na projekte. Toto prispieva k objektívnemu zvyšovaniu kvality softvéru [30].

4.1.9 Možnosť otestovať viac softvéru

Tester, ktorý využíva výhody automatizovaného testovania sa môže spoľahnúť na to, že množstvo práce, ktorú by musel repetitívne vykonávať, za neho vykoná tento automatizovaný proces. Takto mu zostáva viac času na testovanie iného softvéru [30].

4.1.10 Testovanie na rôznych platformách

Automatizované testy môžu byť spúšťané na rôznych prehliadačoch alebo operačných systémoch. Niektoré testovacie frameworky takto umožňujú napísať jeden test, ktorý je testovateľný multiplatformne. Tester by pre rovnaký výsledok musel vynaložiť niekoľkonásobne väčšie úsilie [30].

4.1.11 Znovupoužiteľnosť

Existujúci framework a technológie v ňom použité, je možné použiť aj pri ďalšom projektoch, kde by sa ich vývoj samostatne nevyplatil [31].

4.1.12 Metriky

Záznamy o priebehu a výsledkoch testov je možné ukladať a následne na nich spúšťať analýzu, týmto sa zisťujú informácie o metrikách projektu. Je možné zistiť informácie o aktuálnom stave, sledovaní postupu oproti plánu, odhaleniu projektových rizík, motivovaní pracovníkov k lepším výsledkom, podklady pre návrh a obhájenie zmeny procesu, zabráneniu chaosu, pretože oblasť, ktorá nie je meraná väčšinou upadá - v skutočnosti je nestrážená a neriadená [31].

4.2 Regresné testovanie

Každý zásah do aplikácie, či už ide o pridanie funkcionality alebo opravu chyby, zo sebou nesie riziko, že sa týmto zásahom vytvorí nová chyba. Pri väčších aplikáciách je časovo nevýhodné zaoberať sa všetkými funkcionalitami, ktoré mohli byť týmto zásahom ovplyvnené. Pre riešenie tohto problému sa využíva regresné testovanie.

Regresné testovanie je definované ako typ softvérového testovania pre potvrdenie, že posledná zmena kódu nepriaznivo neovplyvnila existujúcu funkcionalitu. Regresné testovanie nie je nič iné, iba celá alebo čiastočná sada testov, spustená znova, čím sa zistí funkčnosť aplikácie po zmene [44].

Zo svojej podstaty musí regresné testovanie využívať už existujúce testovacie prípady, aby sa zachytil prípadný rozpor medzi aktuálnymi a skôr získanými výsledkami - vytváranie nových testovacích prípadov nemá zmysel. Regresné testovanie je prevádzané na jednotkovej, integračnej a systémovej úrovni [31].

4.2.1 Výber testovacích prípadov pre regresné testy

Z praxe bolo zistené, že veľké množstvo chýb zistených zákazníkmi vzniklo kvôli oprave chýb na poslednú chvíľu. Tieto opravy vytvorili vedľajšie efekty a preto výber testovacích scenárov vhodný pre regresné testovanie nie je jednoduchý [44]. Regresné testy by mali obsahovať:

- Testy overujúce základnú a kľúčovú funkcionalitu systému.
- Testy pokrývajúce veľmi často používané súčasti systému.
- Testy pokrývajúce oblasť zmeny a oblasti priamo súvisiacej.
- Testy často vedúce k odhaleniu defektov.
  [31]

5 NÁSTROJE PRE AUTOMATIZOVANÉ TESTOVANIE WEBOVÝCH APLIKÁCIÍ

V súčasnej dobe sa dopyt po vývoji desktopových aplikácií znižuje a do popredia sa dostávajú webové a mobilné aplikácie. Zo zvyšujúcim sa vývojom rastie aj záujem mať tieto aplikácie pokryté automatizovanými testami.

Nástrojov pre automatizované testovanie sa na trhu nachádza veľké množstvo, preto spoločnosti, ktoré majú o jeho nasedenie záujem, majú veľkú šanci vybrať si im vyhovujúci framework.

Nástroje pre automatizované testovanie môžeme rozdeliť do niekoľkých kategórií, podľa toho ako pristupujú k práci s prehliadačom.

5.1 Record-playback IDE

V minulosti sa im dostával dostatok pozornosti. Sú založené na princípe nahrávania záznamu testera pracujúceho s prehliadačom. Takto uložený záznam transformujú na script alebo kód programovacieho jazyka a následne je test možno opakovane spúšťať. Ich výhodu je, že ich princíp je jednoduchý a nevyžaduje pokročilé vzdelanie v programovaní. Nevýhodou je, že nástroje, ktoré si musia testeri nainštalovať a musia sa s nimi naučiť pracovať, bývajú náročné. Avšak aj po tomto vynaloženom úsilí môže prísť po čase práca na zmar, pretože nedosahujú predpokladanej robustnosti a ich úprava po zmene funkcionality softvéru môže byť náročná [32].

Známe record-playback IDE frameworky : Ranorex, Katalon, Selenium IDE.

5.2 Testovanie pomocou headless prehliadačov

Headless prehliadač je webový prehliadač bez grafického prostredia, pristupuje k webovým stránkam, ale nezobrazuje ich obsah. Hlavným využitím je poskytovanie obsahu pre aplikácie, ktoré testujú webové stránky alebo získavajú dáta [36].

Pre testovanie webových aplikácií sa využívajú hlavne HtmlUnit, PhantomJS, ZombieJS.

Nevýhodou je, že tieto prehliadače môžu byť podobne náročné ako klasické, sú odlišné od prehliadačov používaných používateľmi a v headless prehliadačoch sa tak nemusia prejaviť bugy, ktoré by sa inak prejavili v normálnych prehliadačoch [32]. Ďalšou nevýhodou je, že vývojári nevidia grafické prostredie, čo im sťažuje písanie testov.

5.3 Frameworky využívajúce jednoduchosť headless prehliadačov

Sú to frameworky postavené na WebDriver, čiže poskytujú testovanie v prehliadačoch, ktoré sú postavené na rovnakom kóde ako reálne prehliadače, v ktorých bude aplikácia distribuovaná [32]. Navyše, vďaka frameworkom ako Jasmine, dokážu v testoch využiť pokročilej javascript syntaxe a prichystaným tvrdeniami. Ďalšie frameworky ako Protractor, sú prichystané na testovanie webových aplikácií postavených na moderných javascript frameworkoch ako Angular [37]. Nevýhodou je, že na písanie týchto testov je potreba znalosť programovania a frameworku. Na obrázku číslo 4 je test webovej aplikácie frameworkom Protractor.

```
describe('angularjs homepage todo list', function() {
  it('should add a todo', function() {
    browser.get('https://angularjs.org');

    element(by.model('todoList.todoText')).sendKeys('write first protractor test');
    element(by.css('[value="add"]')).click();

    var todoList = element.all(by.repeater('todo in todoList.todos'));
    expect(todoList.count()).toEqual(3);
    expect(todoList.get(2).getText()).toEqual('write first protractor test');

    // You wrote your first test, cross it off the list
    todoList.get(2).element(by.css('input')).click();
    var completedAmount = element.all(by.css('.done-true'));
    expect(completedAmount.count()).toEqual(2);
  });
});
```

Obrázok 4 Ukážka testu v o frameworku Protractor [37]

5.4 Testovacie frameworky na cloude – Screenster

Screenster je automatizačná platforma, postavená pre menej technických testerov. Tento framework je postavený na princípe nahrávania a opakovania krokov, ktoré vytvoril používateľ. Takéto frameworky už existujú, ale Screenster sa od nich líši množstvom pokročilých funkcií. Screenster sa snaží, aby všetky testy boli distribuovaná na zdieľaný server alebo cloud. Aby zachoval testovanie čo najjednoduchšie, tester nikdy nepracuje s kódom, ale Screenster zaznamenáva jeho interakciu na testovanom webe a na základe UI zmien zachytáva zmeny v DOM elementoch. Po uložení má tester možnosť upraviť test - znova bez zasiahnutia do kódu. Takto uložený test sa následne Screenster snaží zopakovať, krok po kroku. Popritom berie v úvahu aj presuny elementov a zmeny štýlov, čo značne znižuje námahu vynaloženú na údržbu testov [32].

6 SELENIUM

Projekt Selenium je iniciatíva, ktorá si dala za cieľ vytvoriť sadu softvérových nástrojov na testovanie webových, pričom každý s týchto nástrojov má rozdielny prístup k podpore automatizácie testovania. Využitie týchto nástrojov umožňuje multiplatformné testovanie aplikácií založených na rôznych programovacích jazykoch, v najpoužívanejších webových prehliadačoch. Framework Selenium umožňuje vykonávať vysoko flexibilné operácie v prehliadači, umožňujúce nájdenie a interakciu s UI elementami a porovnávanie očakávaných výsledkov testov oproti skutočnému správaniu sa aplikácie [1].

Selenium je Open Source, čo pridáva na jeho flexibilite a rozšíriteľnosti. Takto vytvorené úpravy podľa daných potrieb sú je jeho najväčšou výhodou oproti iným automatizačným nástrojom [1].

Iniciatíva vydala množstvo nástrojov na testovanie webový aplikácií, líšia sa od seba spôsob tvorby testov, ale aj cieľom požitia. V nasledujúcich častiach budú popísané jednotlivé nástroje z testovacej sady Selenium

6.1 Selenium IDE

Selenium IDE je nástroj pre vytváranie testovacích skriptov, distribuovaný ako plugin do webového prehliadača Mozilla Firefox. Tento plugin umožňuje nahrávanie interakcie užívateľa s UI elementami tak, ako sú vykonávané. Selenium IDE následne dokáže vygenerovať znovu použiteľný skript v rôznych programovacích jazykoch, ktorý používateľ Selenium IDE dokáže modifikovať. Podľa kontextu vybraného UI elementu dokážu byť na ňom spúšťané vybrané skripty, čo umožňuje zoznámenie sa z funkciami frameworku [1] [2].

6.1.1 **Nevýhody Selenium IDE**

Selenium IDE obsahuje možnosť, ako si jednoducho uložiť daný testovací scenár, pre neskorší import a spustenie, nie je určené pre plnohodnotné automatizovanie testovania. Jeho hlavný účel je rýchly prototypovací nástroj. Problémom je, že neposkytuje základné funkcionality programovacích jazykov ako podmienky či cykly. Iniciatíva stojaca za vývojom tohto nástroja nemá záujem tento problém vyriešiť a komunita vývojárov, používajúca framework Selenium uprednostňuje Selenium 2, vďaka jeho robustnosti, možnosti výberu vhodného programovacieho jazyka a implementácii pokročilých funkcií [1].

6.2 Selenium 1 (Selenium Remote Control)

Bolo vytvorené na javascript knižnici a umožňovalo pristupovať k UI elementom z rôznych programovací jazykov, čo v tej dobe neumožňoval žiaden iný nástroj pre automatizované testovanie. Avšak postupom času, najmä kvôli obmedzeniam pre javascript zo strany webových prehliadačov sa stal zastaraným [1].

6.3 Selenium 2 (Selenium-WebDriver)

Vzniká ako projekt WebDriver, z dielne vývojára Google a obchádza použitie javascriptovej knižnice a tým sa zbavuje obmedzení s tým spojenými. WebDriver využíva natívnych metód operačných systémov a webových prehliadačov. Selenium-WebDriver vzniká spojením komunity vývojárov Selenium a technológie WebDriver a vyznačuje sa súdržným, objektovo orientovaným API, ktoré poskytuje riešení pre moderné pokročilé testovanie webových aplikácií, kde sa môžu elementy na stránke meniť bez toho aby bola načítaná znovu [3].

6.4 Selenium-WebDriver API

V nasledujúcich častiach bude popis Selenium-WebDriver API primárne pre vývoj v .NET Framework.

Operácie nad instanciou WebDriveru sa vykonávajú nad implementáciou interface IWebDriver, tieto implementácie vytvára komunita alebo spoločnosti vyvíjajúce jednotlivé prehliadače.

Dôležité je, že takýto interface by mal byť čo najuniverzálnejší, preto interface IWebDriver obsahuje len 8 metód, ktoré implementujú jednotlivé prehliadače. Najmä metódy implementujúce interface ISearchContext, sa snažia čo najbližšie priblížiť úkonom, ktoré bežný používateľ internetu vykonáva vo webových prehliadačoch. Ukážka práce s IWebDriver je vidieť na obrázku 5.

```
public static void GoToUrlSafely(this IWebDriver driver, string url)
{
    if (driver.Url.Equals(url, StringComparison.OrdinalIgnoreCase))
    {
        return;
    }

    var wait = new WebDriverWait(driver, Browser.GeneralTimeout);
    driver.Navigate().GoToUrl(url);
    try
    {
        wait.Until(ExpectedConditions.UrlToBe(url));
    }
    catch (Exception)
    {
        throw new UrlException($"Can't go to {url}");
    }
}
```

Obrázok 5 Práca s IWebDriver v praktickej časti

6.4.1 Metóda FindElement

IWebElement FindElement(By *by*);

Metóda vráti instanciu interfacu IWebElement, resp. prvý element nájdený podľa zadaného parametru. Parameter je triedy By [4].

6.4.2 Trieda By

Predstavuje mechanizmus, pomocou ktorého sa má nájsť element v dokumente. Obsahuje statické metódy, ktoré bližšie špecifikujú podľa čoho sa majú UI elementy danom kontexte vyhľadávať [5].

Vyhľadávanie podľa:

- Triedy UI elementu: *By By.ClassName(string názovTriedy);* [6].
- Css selektoru: *By By.CssSelector(string cssSelektor);* [7].
- Id elementu: *By By.Id(string idElementu);* [8].
- Text v odkaze: By By.LinkText(string textOdkazu); [9].
- Čiastočného textu v odkaze: *By By.PartialLinkText (string čiastočnýText);* [11].
- Atribútu name: *By.Name (string atribútName);* [10].
- Názvu tagu elementu: *By By.TagName (string názovTagu);* [12].
- Podľa XPath query: *By By.XPath(string xPath);* [13].

6.4.3 **Metóda FindElements**

ReadOnlyCollection<IWebElement> FindElements(By *by*);

V dokumente nájde a vráti kolekciu instancií implementujúcich IWebElement. Kolekcia je readonly - UI elementy by mali byť immutable [14].

6.4.4 **Metóda Navigate**

INavigation Navigate();

Používa sa na navigáciu na kontrétnu stránku, ale zachováva históriu prehliadača a cookies, takže môže byť použité tlačidlo späť a dopredu na navigáciu medzi stránkami navštívenými počas testu [15]. Ukážku použitia na obrázku č. 5.

Interface INavigate umožňuje pristúpiť k histórii a meniť URL

- Posun dozadu v histórii prehliadača: *void INavigation.Back();* [16].
- Posun dopredu v histórii prehliadača: *void INavigation.Forward();* [17].
- Presun na URL adresu zadanú v dátovom type string: *void INavigation GoToUrl(string url);* [18].
- Presun na URL adresu zadanú v dátovom URI: *void INavigation GoToUrl(URI url);* [19].
- Obnova - refresh súčastnej stránky: *void INavigation Refresh();* [20].

6.4.5 **Metóda Manage**

Metóda vracia instanciu interface IOptions pomocou ktorej je možné pristupovať k nižšej úrovni nastavení prehliadača. Umožňuje napríklad meniť veľkosť okna, upravovať cookies, pristupovať k logovaniu [21].

Property IOptions.Cookies umožňuje upravovať cookies prehliadača [22].

Property IOptions.Logs umožňuje preskúmať záznamy - logy instancie driver [23].

Property IOptions.Window získa objekt pomocou, ktorý umožňuje manipulovať s oknom prehliadača, meniť jeho veľkosť [24].

6.4.6 **Metóda SwitchTo**

Pri aplikácii využívajúcej viacero okien, Selenium-WebDriver priradí každému oknu id, ktoré je možno získať a následne prepínať medzi oknami pomocou metódy *ITargetLocator SwitchTo();* [25]

6.4.7 Metóda Close

Táto metóda zavrie okno súčasného kontextu, pričom ak to toto okno bolo posledným otvoreným oknom instancie IWebDriver tak ukončí túto instanciu.

void IWebDriver.Close(); [53]

6.4.8 Metóda Quit

Ukončí instanciu IWebDriver a zavrie všetky jej okná.

void IWebDriver.Quit(); [52]

6.5 Page Object Pattern

Page Object Pattern je návrhový vzor využívajúci sa pri testovaní webových aplikácií. Ako ukázala prax, nie je vhodné pristupovať z testov priamo k UI elementom, ale pridať ďalšiu vrstvu abstrakcie. Page Object Pattern je tak klasickou ukážkou obalenia logiky - skrývajú detaily UI štruktúry a túto logiku reprezentujú stránkové objekty (page objects). Page object je objektovo orientovaná trieda s fields a metódami. Pri tomto návrhovom vzore sa vývojár snaží dosiahnuť abstrakcie webovej stránky tak, aby s ňou mohlo byť manipulované ako s dátovým objektom. Výhodou je, že pri zmenách UI sa nezmenia testy samotné, iba kód v page objects. Aj keď chceme aby sa testy najviac približovali realite, našim cieľom nie je aby tento objekt obsahoval všetky UI elementy nachádzajúce sa v dokumente, ale aby minimalizoval ich množstvo na tie, ktoré sú nevyhnutné pre interakciu z aplikáciou [26] [27].

6.5.1 Logika testov pri použití Page Object Pattern

Tieto triedy môžu obsahovať aj metódy, ktoré pri interakciou stránky často využívame, aby sme redukovali duplikovanie kódu v testoch. Niektorí vývojári v týchto metódach vykonávajú testovanie, tvrdiac, že tak zamedzujú duplikácii tvrdení v testoch, poskytujú presnejšie chybové hlášky a tak pridávajú ďalšiu logiku do týchto tried. Na druhej strane, druhá skupina vývojárov uchováva logiku testov v testovacích triedach, pretože zodpovednosť page tried by mala byť primárne poskytovanie prístupu v mechanizmom a funkciám stránky s pohľadu užívateľa a nie testera. Pritom k zamedzeniu duplikácií tvrdení vytvárajú ďalšiu vrstvu aplikácie, v ktorej pre takéto prípady vytvárajú metódy obsahujúce často využívaný kód. [26] Avšak tento prístup neodporúča samotné Selenium, podľa ktorého má page object obsahovať len reprezentáciu stránky, služby ktoré poskytuje stránka pomocou metód, ale nemala by mať informácie o tom čo sa na nej testuje [27].

6.5.2 Návrhový vzor Factory pri Page Object Pattern

Factory pattern patrí medzi základné návrhové vzory v objektovo orientovanom programovaní. Slúži na vytváranie instancií objektov, pričom v mieste kde vytvárame nový objekt neodkrývame logiku vytvárania instancie objektu, ale radšej ju zabalíme do factory triedy.

Tento návrh je využitý aj pri Page Object Pattern, kedy sa o logiku vytvorenia stránky a najmä inicializácie UI elementov stará trieda PageFactory [28].

Napríklad:

PageFactory.InitElements<T>(IWebDriver); [29]

Na obrázkoch 6 a 7 sú ukážky kódu s praktickej časti práce.

```
35 references | Gabriel Ecegi, 4 hours ago | 1 author, 3 changes
public partial class Pages
{
    9 references | 5/5 passing | Gabriel Ecegi, 9 hours ago | 1 author, 2 changes
    public static Login Login { get; } = new Login(Browser.Driver);

    7 references | 1/1 passing | Gabriel Ecegi, 4 hours ago | 1 author, 2 changes
    public static Registration Registration { get; } = new Registration(Browser.Driver);

    8 references | 3/3 passing | Gabriel Ecegi, 4 hours ago | 1 author, 2 changes
    public static Projects Projects { get; } = new Projects(Browser.Driver);

    5 references | 1/1 passing | Gabriel Ecegi, 4 hours ago | 1 author, 2 changes
    public static AccountSettings AccountSettings { get; } = new AccountSettings(Browser.Driver);

    0 references | Gabriel Ecegi, 4 hours ago | 1 author, 1 change
    public static Layout Layout { get; } = new Layout(Browser.Driver);
}
```

Obrázok 6 Page Factory použitá v praktickej časti

```
4 references | Gabriel Ecegi, 4 hours ago | 1 author, 2 changes
public partial class Registration : ISeleniumPage
{
    [FindsBy(How = How.CssSelector, Using = "[name='__RequestVerificationToken']")]
    public IWebElement __RequestVerificationTokenInput;

    [FindsBy(How = How.CssSelector, Using = "[name='CompanyName']")]
    public IWebElement CompanyNameInput;

    [FindsBy(How = How.CssSelector, Using = "[name='IsCompany']")]
    public IWebElement IsCompanyInput;

    [FindsBy(How = How.CssSelector, Using = "[name='User.FirstName']")]
    public IWebElement UserFirstNameInput;

    [FindsBy(How = How.CssSelector, Using = "[name='User.LastName']")]
    public IWebElement UserLastNameInput;

    [FindsBy(How = How.CssSelector, Using = "[name='User.Login']")]
    public IWebElement UserLoginInput;

    [FindsBy(How = How.CssSelector, Using = "[name='User.Password']")]
    public IWebElement UserPasswordInput;

    [FindsBy(How = How.CssSelector, Using = "[name='RepeatPassword']")]
    public IWebElement RepeatPasswordInput;

    [FindsBy(How = How.CssSelector, Using = "[sn='RegisterBtn']")]
    public IWebElement RegisterBtn;

    1 reference | Gabriel Ecegi, 4 hours ago | 1 author, 1 change
    public Registration(ISearchContext driver)
    {
        PageFactory.InitElements(driver,this);
    }

    13 references | Gabriel Ecegi, 4 hours ago | 1 author, 2 changes
    public string Url { get; } = @"http://blogictest-test0.blogic.cz/security/registration";
}
```

Obrázok 7 Page Object Pattern – stránka registration použitá v praktickej časti

6.6 Selenium Grid

Selenium Grid je súčasťou frameworku Selenium, dovoľuje paralelne spúšťať testy na rôznych platformách, operačných systémoch a na rôznych prehliadačoch, v podstate rozdeľuje distribúciu testov. Takto umožňuje niekoľkonásobne zmenšiť čas potrebný na beh testu. Napríklad sada testov skladajúca sa zo 1000 testov, spustená paralelne na Selenium Grid skladajúceho sa z hubu a 5 uzlov, bude dokončená za pätinu času. Toto je výhodné najmä pri testovacích sadách, ktoré bežia niekoľko hodín, pri takýchto by vývojár nemal dostatok času skontrolovať aktuálny stav testov okamžite. Ako vidieť na obrázku 8, Selenium Grid dokáže vytvoriť sieť prepojených testovacích strojov, ktoré sa nazývajú uzly (nodes). Táto sieť uzlov je kontrolovaná hubom, pomocou ktorého môžeme spúšťať testy ona jednotlivých uzloch [33] [34] [35].

Obrázok 8 Hub a uzly [34]

6.6.1 **Hub**

V architektúre Selenium Grid sa vždy nachádza len jeden hub, je centrálnym bodom odkiaľ sa spúšťajú testy. Práve na ňom sa nachádzajú testy, s informáciami o konfigurácii na ktorej by sme potrebovali spustiť test. Najprv treba všetky uzly, s ktorými by sme chceli operovať pridať do hubu - zaregistrovať ich. Všetky uzly takto budú v hube zaregistrované zo svojou konfiguráciou - operačným systémom, typom webového prehliadača a jeho verziou. Po spustení testu s potrebnou konfiguráciou, hub skontroluje všetky uzly a sám rozpozná, ktorý uzol odpovedá takejto konfigurácii a spustí na ňom test. V prípade, že stroj s takouto konfiguráciou nenájde, vráti chybu [33] [35].

6.6.2 **Uzol (node)**

Uzol je instancia Selenia spúšťajúca testy, ktoré užívateľ nahral do hubu. Ako stroj na testovanie je pripojený do hubu. Hub môže obsahovať viacero uzlov, účelom je aby uzly boli platformovo čo najrozmanitejšie - obsahovali rôzne spektrum operačných systémov a prehliadačov v rôznych verziách. Takto navrhnutá architektúra dokáže simulovať fragmentované spektrum užívateľských konfigurácii s ktorými sa aplikácia musí vysporiadať na produkcii [34].

II. PRAKTICKÁ ČÁST

7 ÚVOD DO PRAKTICKEJ ČASTI

Framework Selenium poskytuje zaujímavý kompromis medzi frameworkami na testovanie webových aplikácii. Oproti headless frameworkom Selenium WebDriver poskytuje zobrazenie testov tak, ako ho vidia klienti. Record-playback IDE síce umožňujú rýchlu tvorbu testov, ale je v nich zložitejšie vytvárať komplexnejšie testovacie sady. Testovacie frameworky na cloude poskytujú pri testovací vyššiu abstrakciu, ale takto neumožňujú úpravu frameworku podľa potrieb softvérových teamov. Z týchto dôvodov má vlastná testovacia platforma využívajúca produkty od Selenium dobrú šancu stať sa použiteľným softvérom. Takáto platforma by mala umožňovať ukladať výsledky testovacích scenárov a následne ich zobraziť výsledky pre analýzu.

Vhodnou platformou na implementáciu takejto funkcionality sa javí beTested. Moje riešenie som nazval „BeTested.Automation“.

7.1 beTested

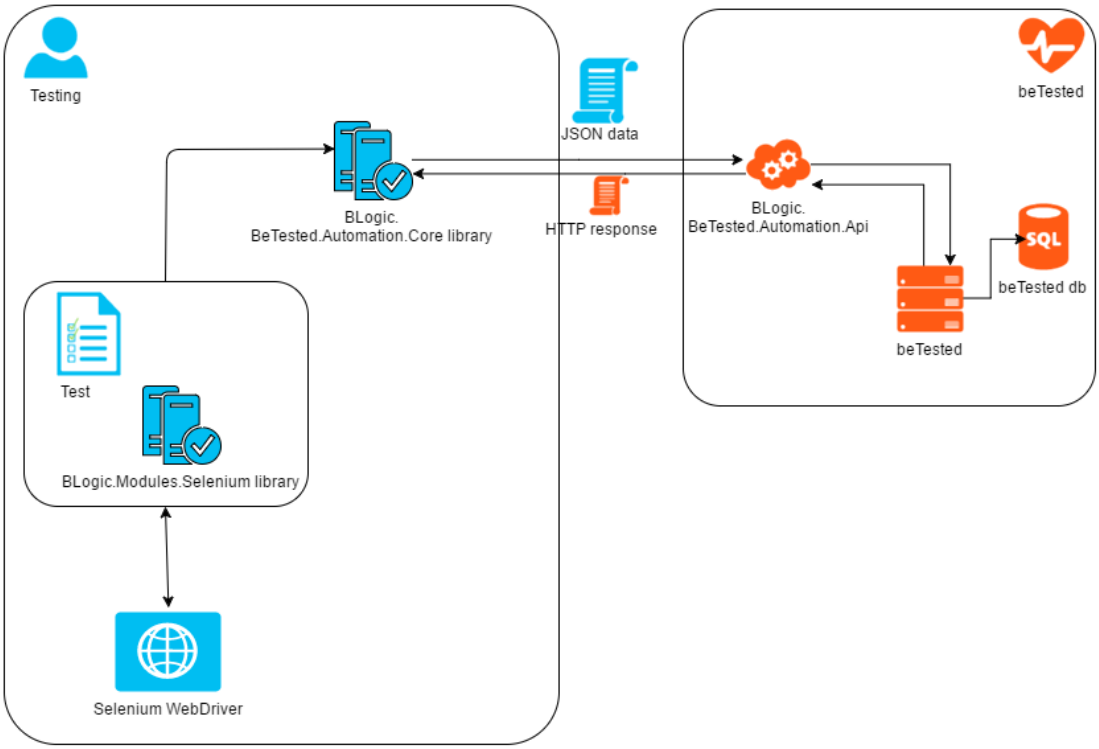
Testovacia platforma od spoločnosti Business Logic nazvaná beTested, umožňuje vytvárať testovacie scenáre pre vyvíjané softvérové produkty. Platforma beTested sa snaží klásť dôraz na jednoduchosť vytvárania popisov funkcionality softvéru. Preto nerozlišuje medzi testovacími prípadmi a prípadmi použitia, resp. scenáre použitia nahradzujú testovacie scenáre (Use case). Z toho vyplýva, že testovacie kroky sa nazývajú kroky scenáru použitia (Use case step). Entity zdržujúce testovacie scenáre sa nazývajú skupiny testovacích scenárov (Use case set). K jednému Use case set je priradená entita Test, testovaním Use case setu vznikajú entity Test Iterations. Ku konkrétnemu Use case môžu byť priradené požiadavky nutné pre vykonanie daného Use case, táto entita sa nazýva Requirement a podobné Requirements sú združované do skupín Requirement set.

7.2 Architektúra riešenia praktickej časti

Riešenie praktickej časti sa skladá z niekoľkých projektov, prvým z nich je projekt z názvom BLogic.BeTested.Automation.Core starajúci sa o prepojenie s unit testami. Ďalším je BLogic.Modules.Selenium rozširujúci možností práce s Selenium-WebDriver. Projektom umožňujúcim zapisovanie dát do beTested je BLogic.BeTested.Automation.Api rozširujúcim beTested o REST API. Projekty BLogic.Modules.Selenium a BLogic.BeTested.Automation.Core sú voľne dostupné pomocou balíčkovacieho softvéru NuGet.

7.3 Business flow BeTested.Automation

Po vytvorení testov a validnej štruktúry na webe beTested, je možné takéto testy štandardne spustiť. Po spustení sa vykonajú príkazy na instancii WebDriver a ak sa nevyskytne výnimka môže byť otestované tvrdenie. Tvrdenie je otestované pomocou knižnice BLogic.BeTested.Automation.Core, kde sa uloží výsledok tvrdenia. Po otestovaní všetkých metód v triede sú dáta poslané na BLogic.BeTested.Automation.Api, kde sú spracované a uložené do databáze. API vráti HTTP response podľa úspešnosti operácie. Ak sú k dispozícii ďalšie testovacie triedy, tak testovanie pokračuje ak nie test skončí. Po ukončení testovania je možné si uložené záznamy o testoch pozrieť na webe beTested. Business flow BeTested.Automation je zobrazený na obrázku 9.



Obrázok 9 Business flow BeTested.Automation

7.4 Technológie použité v praktickej časti

Riešenie praktickej časti je naprogramované v frameworku .NET od Microsoftu. Všetky projekty majú jednotnú verziu cieľového frameworku: .NET Framework 4.6.1, projekty sú napísané v programovacom jazyku C# 6. Projekt BLogic.BeTested.Automation.Api je typu ASP.NET Web API, založený na RESTful HTTP službách.

7.5 Konfiguračný súbor

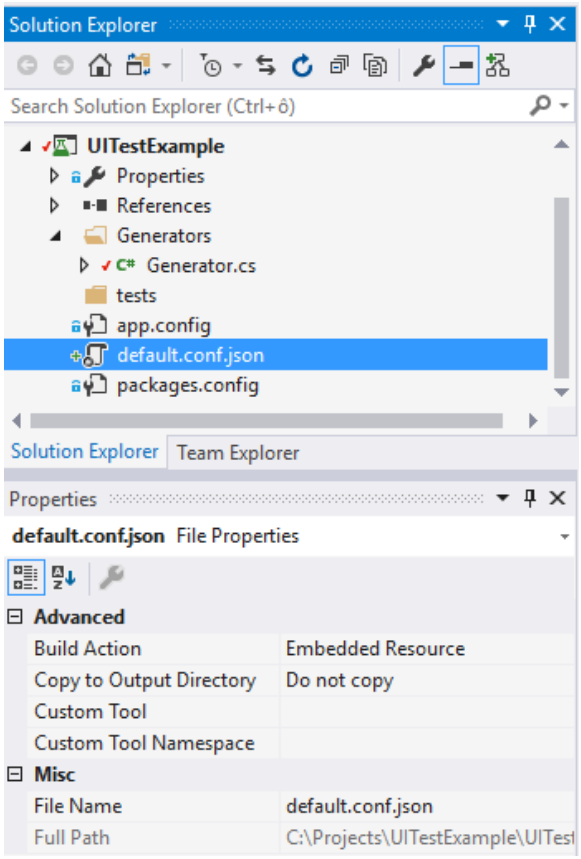
Pre uloženie informácií o užívateľovi a projekte, sa pri testovaní využíva konfiguračný súbor. Tento konfiguračný súbor je vo formáte JSON a má presne určenú štruktúru.

Ako je vidieť na obrázku 10, konfiguračný súbor sa skladá z dvoch objektov beTested a driver. V ďalších častiach je popísané, na čo slúžia jednotlivé properties objektov a akého sú dátové typu.

Konfiguračný súbor je pri vytváraní štruktúry projektu vytvoriť a vyplniť. Ďalej mu v jeho nastaveniach treba vybrať Build Action – Embedded Resource, tak ako je zobrazené na obrázku 11, aby ho build proces bral ako spustiteľný súbor alebo knižnicu.

```
{
  "beTested": {
    "Login": "gabrielecegi@betested.cz",
    "Password": "123456",
    "ProjectID": 122,
    "ApiVersion": "1.00"
  },
  "driver": {
    "Type": "chrome",
    "GeneralTimeout": 5,
    "ShortTimeout": 2
  }
}
```

Obrázok 10 Ukážkový konfiguračný súbor



Obrázok 11 Nastavenie Build Action u konfiguračného súboru

7.5.1 Objekt beTested

Objekt beTested sa skladá s properties umožňujúcich autentifikáciu, správne určenie projektu a verziu API.

Login [string] – prihlasovacie meno do beTested

Password [string] – heslo do beTested

ProjectID [int] – ID projektu v beTested, v ktorom sú testovacie scenáre. Toto ID je viditeľné po kliknutí na Dashboard projektu

ApiVersion [string] – verzia API na ktorej endpoint chceme posielať dáta z testovania. Verzovanie API je pridané hneď od začiatku, aby sa zamedzilo problémom zo spätnou kompatibilitou.

7.5.2 **Objekt driver**

Type [string] - určuje na akom prehliadači sa majú vykonávať UI testy, takto je jednoducho možné pri písaní testov používať instanciu interfacu IWebDriver, pričom skutočný typ prehliadača určí až konfiguračný súbor. Toto je výhodné najmä pri použití v Selenium Grid, kedy je prioritou testovať na viacerých typoch webových prehliadačov.

GeneralTimeout [int] - vo väčšine rozšírení Selenium vytvorených v rámci práce je implicitne použitý objekt WebDriverWait, ktorý čaká na splnenie požiadavky dobu, ktorú mu je explicitne zadaná a GeneralTimeout je hodnotou tohto parametra. V podstate udáva dĺžku doby v sekundách, koľko má IWebDriver čakať na splnenie requestu.

ShortTimeout [int] - v niektorých rozširujúcich metódach nie je potreba čakať dlhú dobu zadanú v GeneralTimeout, ale postačuje ShortTimeout. Využitie je napríklad v metóde SeleniumAssert.ElementIsNull(string selector), kedy len potreba overiť že element neexituje, takže netreba čakať na jeho načítanie.

8 BLOGIC.BETESTED.AUTOMATION.CORE

BLogic.BeTested.Automation.Core dokáže z unit testov zisťovať pomocou reflexie v C# informácie o metódach a triedach. Všetky tvrdenia metód prechádzajú jedným miestom, ktoré umožňuje logovanie výsledkov a následne umožňuje, aby všetky výsledky testov v testovacom projekte, boli odoslané na endpoint vytvoreného REST API a odtiaľ uložené do databáze. Tento projekt sa skladá z niekoľkých modulov.

8.1 Modul App

V module App sa nachádzajú triedy potrebné pre obsluhu aplikácie ako konštanty, atribúty, extensions.

8.1.1 **Attributes**

.NET Framework umožňuje definovať vlastné atribúty. Tieto atribúty sú emitované ako metadáta, v skompilovanej assembly, keď sú aplikované na elementy programu. To, čo robí atribúty zaujímavé je, že pomocou použitia reflexie dokáže kód čítať metadáta. To znamená, že vlastné atribúty priamo ovplyvňujú ako beží kód [47].

Boli vytvorené triedy vlastných atribútov, ktoré vyznačujú názov UseCase a UseCaseSetu použitého v testoch.

Použitie:

K testovacím triedam, ktoré je potrebné označiť ako reprezentácie entít UseCaseSet môže byť pridaný atribút [UseCaseSet(string NázovOdpovedjúceUseCaseSetu)]. Obdobne je použitý atribút UseCaseAttribute - [UseCase(string NázovOdpovedajúcehoUseCase)]

```
[TestClass]
[UseCaseSet("Account settings")]
public class AccountSettingsTests : BLogicSeleniumTests
{
    [TestMethod]
    [UseCase("Change company name")]
    public void ChangeCompanyName()
    {
        ...
    }
}
```

8.1.2 Extensions

Extension metódy môžu byť použité na rozšírenie tried alebo rozhraní, takto môžeme pridať spoločnú funkcionalitu všetkým objektom tohto rozhrania alebo triedy [47].

Obsahuje rozšírenia triedy Object type umožňujúcu získať hodnotu uloženú v property atribútu.

Použitie:

```
internal static string GetUseCaseSetTitle(Type Class)
{
      var title = Class.GetAttributeValue((UseCaseSetAttribute x) => x.Title);
      return string.IsNullOrEmpty(title) ? Class?.Name : title;
}
```

8.2 Modul Core

Modul Core je jadrom projektu, obsahuje triedu Automation využívanú v testoch pre logovanie výsledkov. V konštruktore berie ako parameter dynamický objekt ako svoju konfiguráciu. Z tejto sú zistené informácie o používateľovi a projekte.

Metóda *void Test(Action method)*, má ako parameter Action - delegát (pointer) na metódu. Ak táto metóda nevyhodí exeption tak je výsledok tohto testu „Passed“, ak je metóda vyhodí do výsledku sa zapíše stav „Failed“ a do property Comment sa uloží správa výnimky. Algoritmus je zobrazený na obrázku 12.

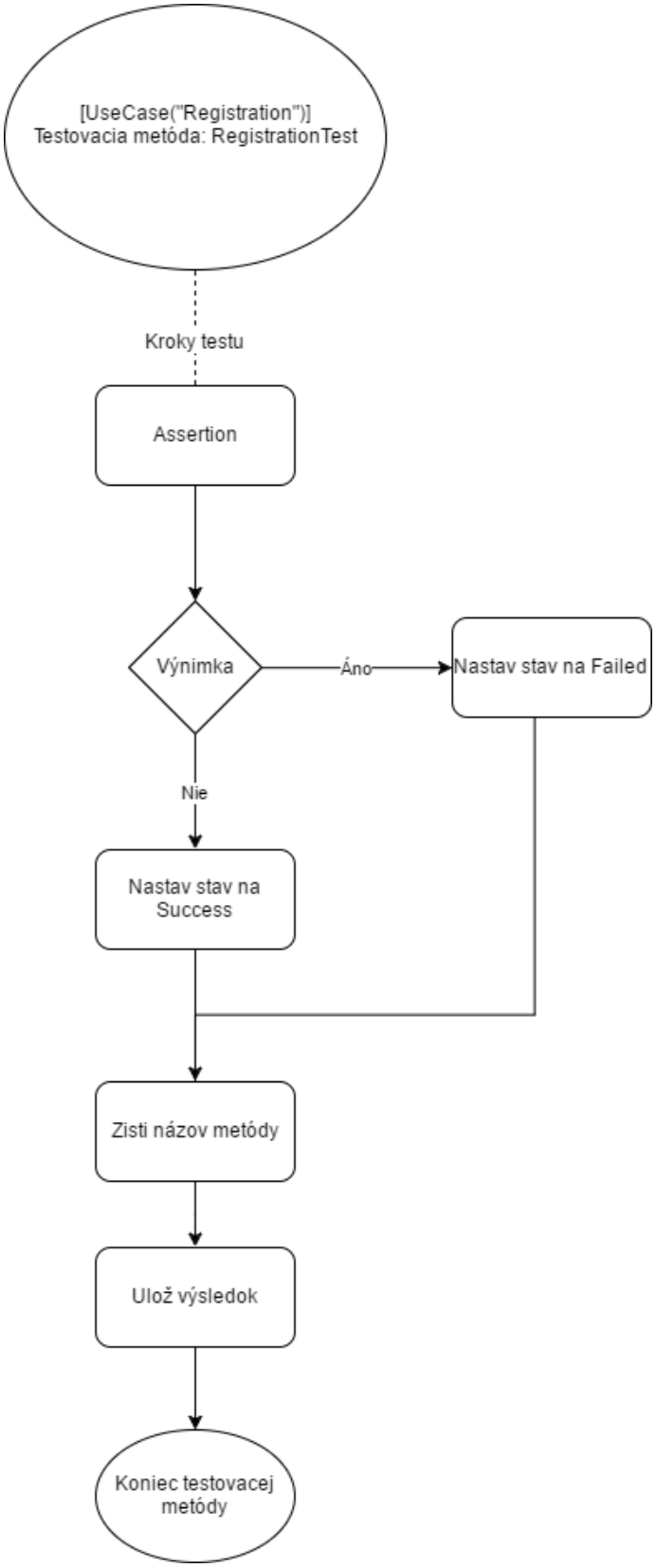
Metóda IRestResponse SaveData() je zodpovedná za odosielanie dát na REST api endpoint. Najprv všetkým netestovaným metódam v danom UseCaseSet nastaví stav na „Skipped“, serializuje potrebné dáta do formátu JSON a pošle ich na endpoint BLogic.BeTested.Automation.Api s relatívnou adresou */test* metódou POST. Po ukončení spojenia metóda vráti výsledok spojenia - response. Postup je vizualizovaný na vývojovom diagrame - obrázok 13.

8.3 Dtos

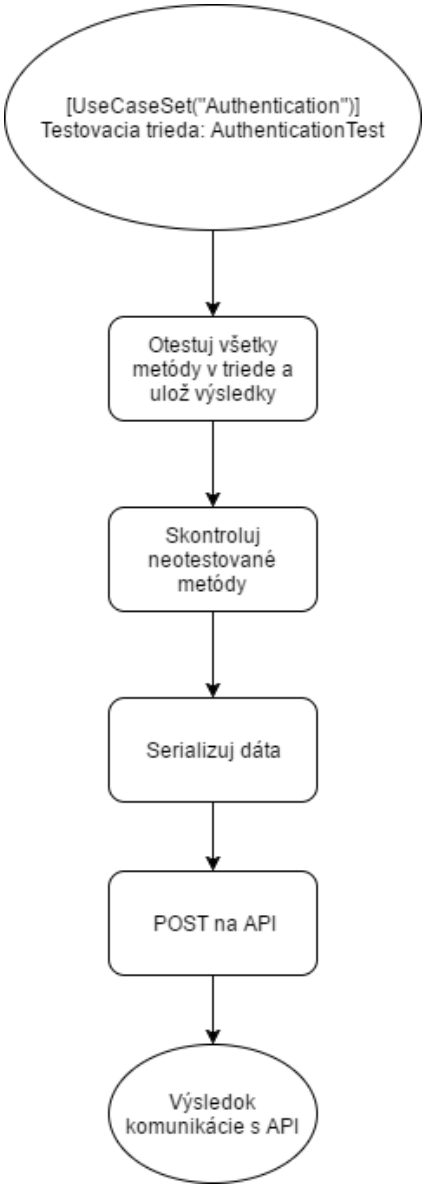
Modul Dtos obsahuje obsahuje Data Transfer Objects, v ktorých sú informácie pre prenos dát do REST API. Každá property objektu má atribút JsonProperty označujúci ako sa má daná property nazývať v serializovanom JSON objekte.

8.4 Providers

Obsahuje triedu ReflectionProvider, ktorá pomocou reflexie získava informácie o metódach a triedach.



Obrázok 12 Testovacia metóda



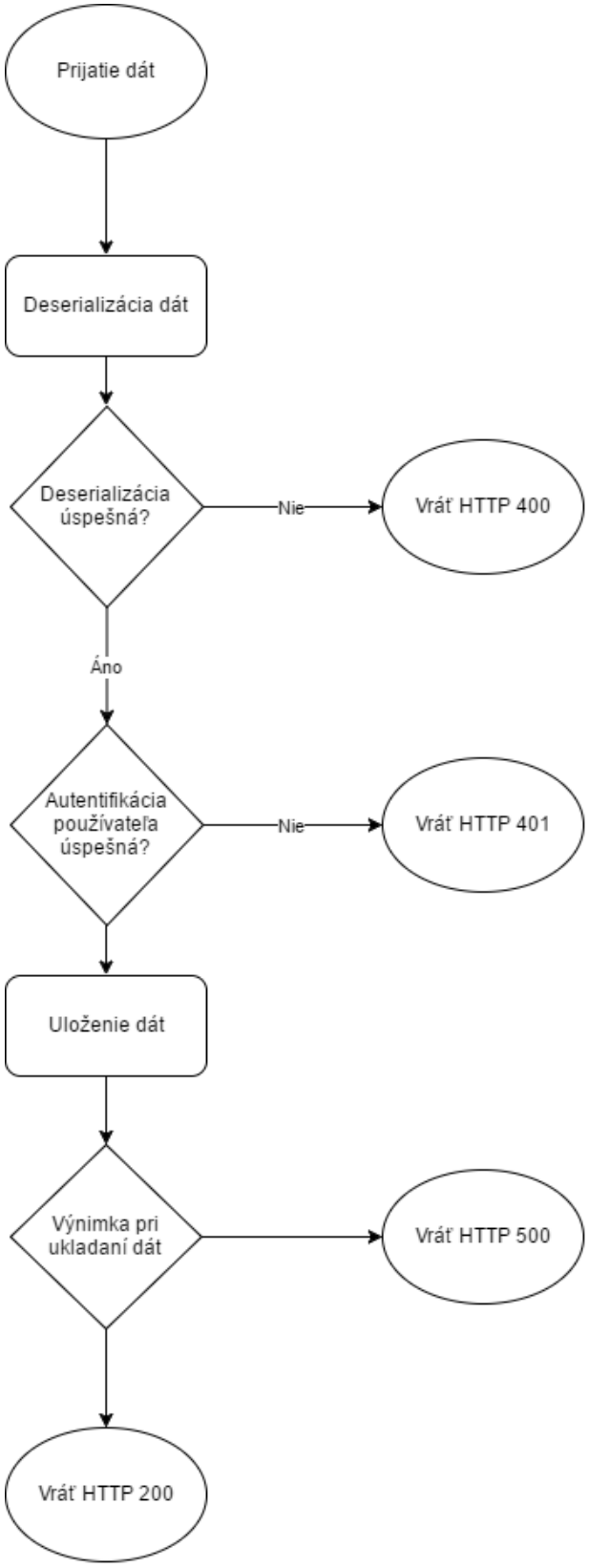
Obrázok 13 Testovanie triedy

9 BLOGIC.BETESTED.AUTOMATION.API

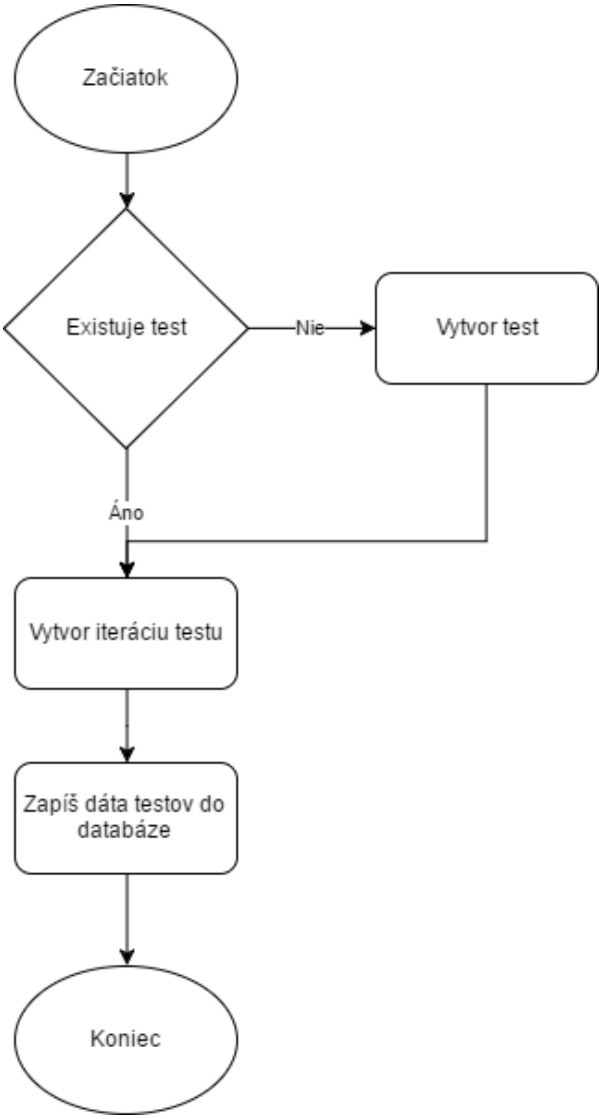
Representational State Transfer - REST - sa stal dominantnou architektúrou pre spoluprácu aplikácií cez HTTP, kompletne zatieňujúc SOAP. REST popisuje aplikáciu ako zdroj entít, reprezentujúcich entity približujúce sa realite. Pomocou metód http je možné v aplikácii využívajúcej REST API interagovať s týmito entitami [48].

BLogic.BeTested.Automation.Api dokáže prijímať dáta výsledkov testov v tvare JSON objektov. Ak je tento objekt validný, tak z neho API získa dáta o používateľovi. Následne zistí či sú dáta správne. Ak IApiTestService zistí, že prihlasovacie údaje sú nesprávne vráti HTTP kód 401 - unauthorized. Keď objekt prejde validáciou, ITestApiService sa ho pokúsi uložiť do databáze. Po úspešnom uložení vráti HTTP response 200 - OK. Vývojový diagram práce API je vidieť na obrázku 14.

Ako vidieť z vývojového diagramu na obrázku 15, najprv sa nájde alebo vytvorí test prislúchajúci use case setu, ktorého use cases boli testované. V tomto teste sa vytvorí nová iterácia testu, do ktorej sa postupne zapíšu výsledky use cases nachádzajúcich sa v prijatom objekte. Ak bola operácia úspešná na klienta sa vráti HTTP response 200.



Obrázok 14 Proces prijatia výsledkov testov na strane API



Obrázok 15 Vývojový diagram uloženia objektu do databáze

9.1 Swagger

Projekt BLogic.BeTested.Automation.Api využíva dokumentáciu pomocou nástroja Swagger. Swagger špecifikácia je formát definície popisujúci RESTful APIs. Swagger špecifikácia tiež vytvára RESTful prostredie pre jednoduchý vývoj a používanie API efektívnym mapovaním všetkých zdrojov a operácii s nimi spojenými [49].

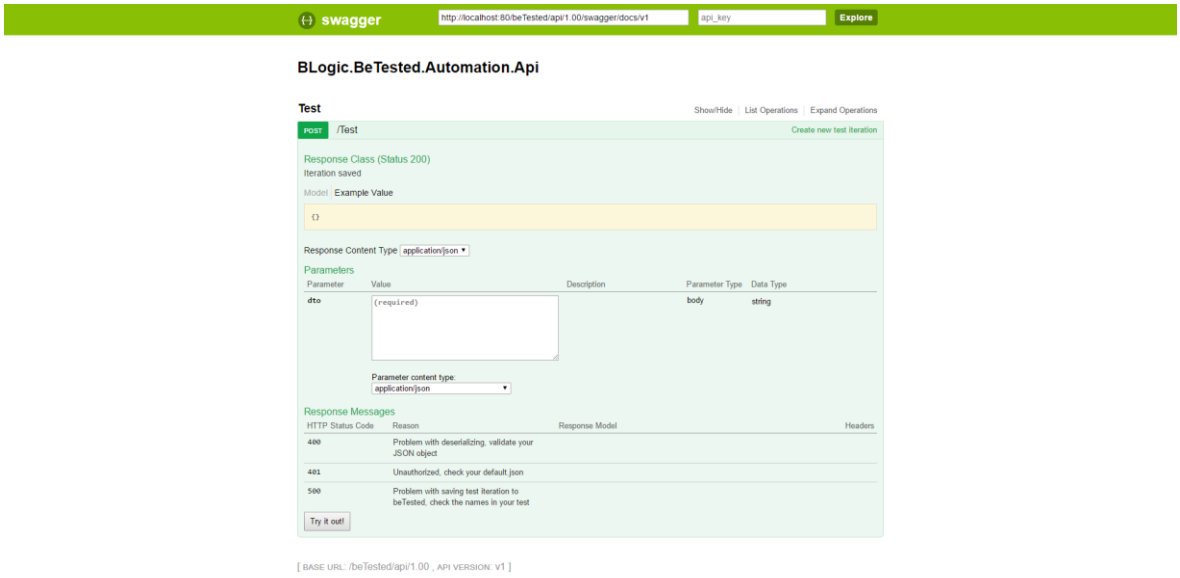
Vhodnou knižnicou je .NET ASP je Swashbuckle správajúci sa ako midleware, vytvárajúci novú vrstvu aplikácie ktorá konzumuje popisy endpointov a podľa nich vytvára Swagger dokumentáciu s UI.

Popis sa môže týkať funkcionality, ktorú má endpoint na starosti, ale popísané môžu byť aj napríklad HTTP responses, ktoré endpoint vracia.

Nad časti kódu, ktoré je potreba zdokumentovať boli pridané XML atribúty, popisujúce danú metódu alebo controller. Tieto dokáže Swagger tranformovať do HTML kódu - kód z obrázku 17 je transformovaný na html z obrázku 18.

```
/// <summary>
/// Create new test iteration
/// </summary>
/// <response code="200">Iteration saved</response>
/// <response code="400">Problem with deserializing, validate your JSON object</response>
/// <response code="401">Unauthorized, check your default.json</response>
/// <response code="500">Problem with saving test iteration to beTested, check the names in your test</response>
/// <content-type>application/json</content-type>
/// <example>"{\"login\":\"*yourlogin*\",\"password\":\"*yourpassword*\",\"projectID\":\"2357\",
/// \"useCaseSetTitle\":\"Authentication\",\"useCasesResults\":[{\"state\":1,\"useCaseTitle\":\"LoginTest\",\"comment\":null}]}"</example>
[HttpPost]
0 references | Gabriel Ecegi, 11 hours ago | 2 authors, 3 changes | 0 requests | 0 exceptions
public IHttpActionResult Test([FromBody] string dto)
{
```

Obrázok 16 Ukážka XML štruktúry pre swagger



Obrázok 17 Vygenerované view na základe XML

Vo view je možné otestovať REST API, stačí vyplniť hodnoty parametrov a kliknúť na tlačidlo s textom „Try it out“. Po tomto je poslaná požiadavka so zadanými parametrami na endpoint. Swagger vo view zobrazí výsledok tohto requestu s potrebnými informáciami.

10 BLOGIC.MODULES.SELENIUM

BLogic.Modules.Selenium vytvára jednoduchú integráciu vytvoreného riešenia do testovacieho projektu. Skladá sa z dvoch častí - analýza stránok a generovanie projektu a rozšírenie metód IWebDriver a IWebElement.

Prvá časť projektu zaoberajúca sa analýzou stránok a generovaním projektu sa neviaže priamo na Selenium-WebDriver, je len doplnkom pre jednoduchšie mapovanie elementov generovanie projektu s Page Object Pattern triedami. Keďže generovanie projektu ani analýza stránok nie je v témou bakalárskej práce, budem sa ňou zaoberať veľmi okrajovo.

Druhá časť projektu, vytvorená v rámci bakalárskej práce sa zaoberá rozšírením metód pre často používané objekty z frameworku Selenium a prácu s instanciou WebDriveru.

Tieto dve projekty sú spojené v jednom projekte z dôvodu, že riešia podobnú tému.

10.1 Core

Modul projektu BLogic.Modules.Selenium s názvom Core sa stará o načítanie správnej konfigurácie pre instanciu WebDrivera a obsahuje triedy rozširujúce funkcionalitu frameworku Selenium - táto časť projektu vznikla z dôvodu chýbajúcich natívnych funkcionalít - napr. implicitné čakanie na vykonanie asynchrónnej požiadavky.

Množstvo súčasných webových aplikácií sa neobíde bez použitia asynchrónneho programovania. Často sa na stránke načíta asynchrónne množstvo DOM objektov a ak je potreba automatizovane testovať ich správanie, softvér nedokáže sám implicitne určiť, či je daný krok načítavaný asynchrónne alebo či nastala chyba. Pri použití Selenium-WebDriver sa využíva objektu WebDriverWait, ktorý po určení typu Driveru, podmienky a času dokáže čakať na splnenie podmienky. Ak sa táto podmienka splní program pokračuje vo vykonávaní inštrukcií a ak objekt WebDriverWait nedosiahne splnenia časovej podmienky v časovom limite vyhodí výnimku. Pri písaní testov je takéto použitie objektu IWebDriver často redundantné a využíva sa takmer stále, preto sa pri všetkých metódach rozširujúcich IWebDriver a IWebElement využíva WebDriverWait implicitne.

10.1.1 **Browser**

Browser je statická trieda zabezpečujúca, že vo všetkých testoch sa bude k WebDriveru pristupovať práve ako k property tohto objektu - centralizuje sa jeho použitie. Toto uľahčuje použitie WebDriveru iného webového prehliadača. Pri zavolaní WebDriveru sa skontroluje konfigurácia a zistia sa potrebné nastavenia z konfiguračného súboru s názvom „default.conf.json“, ktorý sa musí nachádzať v testovacom projekte. Tu je v property „driver“ špecifikované o aký typ WebDriveru sa jedná. V súčasnosti sú podporované štyri prehliadače: Chrome, Firefox, Internet Explorer, Edge. Ak nie je konfiguračný objekt v správnom tvare, alebo nie je vybraný prehliadač k dispozícii metóda vráti výnimku.

Browser sa stará o nastavenie GeneralTimeout a ShortTimeout z konfiguračného súboru.

10.1.2 **SeleniumDriverExtensions**

Framework Selenium obsahuje množstvo užitočných metód, na pomerne vysokej úrovni, avšak pri použití návrhového vzoru Page Object Pattern sa hodí vytvoriť abstraktnejšie metódy.

10.1.3 **Metóda GetElementBySeleniumAttribute**

IWebElement GetElementBySeleniumAttribute(this IWebDriver driver, string value)

Vráti element, ktorý má požadovanú hodnotu v atribúte sn. Do elementov využívajúcich sa v Selenium UI testoch by mal byť pridaný atribút „sn“ vyznačujúci identifikátor elementu pre Selenium testy. Atribútu „sn“ nenahradzuje atribút „id“, „name“ alebo iný použitý identifikátor, vyznačuje však že tento element sa využíva pri UI testoch a identifikátor by nikto nemal meniť za účelom iným ako zmena v testoch. Tento identifikátor je výhodné používať aj preto, že iná identifikácia elementu a referencie na ňu sa pri vývoji môžu zmeniť a testy kvôli tejto zmene prestanú fungovať.

10.1.4 **Metóda GetElementByAttribute**

IWebElement GetElementByAttribute(this IWebDriver driver, string attribute, string value)

Na základe atribútu a jeho hodnoty vráti element. V prípade, že takýto element neexistuje vráti výnimku.

10.1.5 **Metóda ClickSafely**

void ClickSafely(this IWebDriver driver, IWebElement element)

Často využívaná metóda, ktorá klikne na element ak sa takýto element nachádza na stránke. Ak sa nenachádza, počká pokým bude možné na takýto element kliknúť. Pokiaľ sa element do časového limitu nestane klikateľným, vyhodí sa výnimka.

10.1.6 **Metóda GoToUrlSafely**

void GoToUrlSafely(this IWebDriver driver, string url)

Umožňuje prejsť na zadanú url adresu, hodí sa obzvlášť vtedy, keď potrebujeme v nasledujúcom kroku overiť, či sme na danú adresu mohli dostať, alebo presmerovanie trvá dlhšie a používateľ nechce aby sa ďalšie inštrukcie vykonali hneď.

10.1.7 **Metóda SetTextSafely**

void SetTextSafely(this IWebDriver driver, IWebElement element, string text)

Metóda SetTextSafely umožňuje zadať text do elementu, oproti natívnej metóde z frameworku umožňuje počkať, pokým sa potrebný element nezobrazí na stránke. Toto umožňuje zadať text napríklad do vyskakovacieho okna - modálneho okna.

10.1.8 **SeleniumElementExtensions**

Pri práci s elementami pri návrhovom vzore Page Object Pattern chceme priamo pracovať s čo najviac elementami IWebElement, preto sa u nich hodí mať extension metódy.

10.1.9 **Metóda ClickSafely**

void ClickSafely(this IWebElement element)

Umožňuje kliknúť na element a v prípade, že na element sa nedá kliknúť počká a keď podmienka kliknuteľnosti nebude splnená do časového limitu vyhodí výnimku.

Ukážka:

Pages.Login.LoginBtn.ClickSafely();

10.1.10 **Metóda GoToUrlSafely**

void GoToUrlSafely(this string url)

V instancii Page Object, často býva property s url danej stránky, toto na ňu umožňuje jednoducho prejsť.

Ukážka:

Pages.Projects.Url.GoToUrlSafely();

10.1.11 **Metóda SetTextSafely**

void SetTextSafely(this IWebElement element, string text)

Metóda umožňuje počkať, kým sa do elementu bude dať zapísať text a následne ho tam vloží. Ak sa tento element neobjaví do časového limitu, metóda vyhodí výnimku.

Ukážka použitia:

Pages.Registration.UserFirstNameInput.SetTextSafely("Ján");

10.1.12 **Metóda GetTextFromElement**

string GetTextFromElement(this IWebElement element)

GetTextFromElement vráti dátový typ string z elementu.

10.1.13 **SeleniumAssert**

Na konci väčšiny testovacích metód býva tvrdenie tzv. assertion, kde porovnávame zistený výsledok voči očakávanému. Ak takéto tvrdenie zlyhá - zistený výsledok nie je taký ako sa očakáva, metóda vráti výnimku, toto je normálne správanie u najpoužívanejších testovacích frameworkov, ako napríklad MSUnit.

10.1.14 **Metóda IsOnUrl**

void IsOnUrl(string url)

Zistí na akej url sa nachádza instancia WebDriver a vyhodí výnimku ak je súčasná url iná oproti porovnávanej.

Ukážka použitia:

_beTested.Test(() => SeleniumAssert.IsOnUrl(ProjectSettingsUrl));

10.1.15 **Metóda ElementIsNull**

void ElementIsNull(string snValue)

Metóda sa pokúsi nájsť element nájsť element podľa hodnoty v atribúte „sn“, a ak ho nájde, hodí výnimku. Táto metóda sa dá využiť napríklad vtedy, ak chceme zmazať entitu a už neočakávame, že sa bude nachádzať na stránke.

Ukážka použitia:

_beTested.Test(() => SeleniumAssert.ElementIsNull(projectName));

10.1.16 **Metóda ElementIsNotNull**

void ElementIsNotNull(string snValue)

Metóda je obdobná, avšak s opačným zámerom ako ElementIsNull.

10.1.17 **Metóda TextEquals**

void TextEquals(IWebElement element, string text)

TextEquals porovnáva, či je text v elemente rovnaký ako text parametru „text“. Ak je rozdielny, metóda hodí výnimku.

11 NUGET BALÍČKY

Pre distribúciu projektov BLogic.BeTested.Automation.Core a BLogic.Modules.Selenium som vybral balíčkovací systém NuGet. Tento umožňuje vytvárať, distribuovať a používať balíčky založené na platforme .NET. Vďaka použitiu si používatelia so záujmom o použitie týchto testovacích knižníc si ich môžu rýchlo a stiahnuť a použiť.

11.1 .nuspec súbor

Každý NuGet balíček potrebuje manifest na popísanie svojho obsahu a jeho závislostí. Tento manifest je nazývaný .nuspec súbor [50] [51].

Po vytvorení .nuspec súboru je potrebné vyplniť ho tak, aby odpovedal skutočnosti. Ďalej treba vyplniť závislosti balíčku, tieto závislosti predstavujú iné NuGet balíčky, využívajúce sa v projekte, z ktorého chceme vytvoriť NuGet balíček. Takto definované závislosti sa nainštalujú s balíčkom.

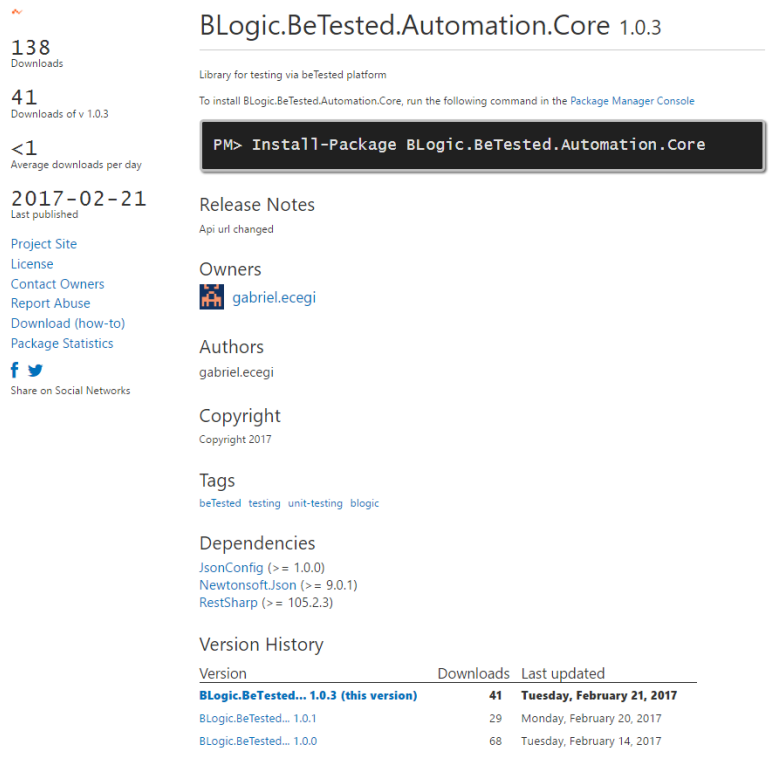
Pri vytváraní balíčku treba z bezpečnostných dôvodov dať pozor, aké súbory sú zahrnuté do distribuovanej verzie. V distribuovanej verzii by sa takto mohli ocitnúť citlivé údaje ako heslá, poznámky, atď. Súbory, ktoré chceme pridať do distribuovanej verzie sa odporúča explicitne vypísať v entite <files> - obrázok 18.

Takto vytvorený údaje z manifestu budú viditeľné pre používateľov, ktorý sa rozhodnú nainštalovať tento NuGet balíček (obrázok 19).

```xml
<?xml version="1.0"?>

<package>
  <metadata>
    <id>BLogic.BeTested.Automation.Core</id>
    <version>1.0.0.1</version>
    <authors>gabriel.ecegi</authors>
    <owners>gabriel.ecegi</owners>
    <licenseUrl>http://blogic.cz</licenseUrl>
    <projectUrl>http://betested.blogic.cz</projectUrl>
    <iconUrl>http://betested.blogic.cz/Assets/img/favicon.png</iconUrl>
    <requireLicenseAcceptance>false</requireLicenseAcceptance>
    <description>Library for testing via beTested platform</description>
    <releaseNotes>Initial release</releaseNotes>
    <copyright>Copyright 2017</copyright>
    <tags>beTested testing unit-testing blogic</tags>
    <dependencies>
      <dependency id="Newtonsoft.Json" version="9.0.1" />
      <dependency id="RestSharp" version="105.2.3" />
      <dependency id="JsonConfig" version="1.0.0" />
    </dependencies>
  </metadata>
  <files>
    <file src="bin\Release\*.dll" target="lib\net461" />
    <file src="bin\Release\*.pdb" target="lib\net461" />
    <file src="bin\Release\*.xml" target="lib\net461" />
  </files>
</package>
```

Obrázok 18 .nuspec súbor projektu BLogic.BeTested.Automation.Core



Obrázok 19 Dostupnosť balíčku dostupná na webe nuget.org

11.2 Inštalácia NuGet balíčka

Do nového projektu vo Visual Studio môžeme inštalovať NuGet balíček dvomi spôsobmi - inštaláciou cez Package Manager Console alebo cez NuGet Package Manager UI.

11.2.1 **Package Manager Console**

Package Manager Console je zabudovaná vo Visual Studio verziách 2012 a vyšších verziách a umožňuje používať NuGet PowerShell príkazy na nájdenie, inštaláciu, odinštalovanie a update NuGet balíčkov.

Pre inštaláciu projektu BLogic.BeTested.Automation.Core stačí do konzole napísať:

Install-Package BLogic.BeTested.Automation.Core

A pre projekt BLogic.Shared.Modules.Selenium:

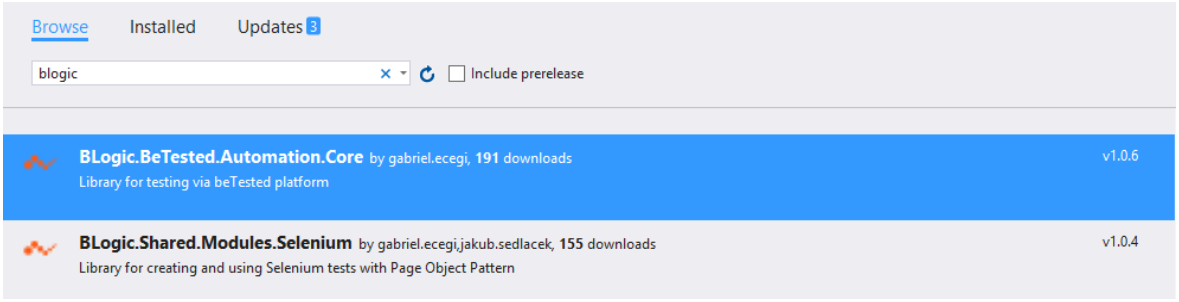
Install-Package BLogic.Shared.Modules.Selenium

Tieto príkazy sú dostupné na webe NuGet na detailoch balíčkov, ako je vidieť na obrázku 19.

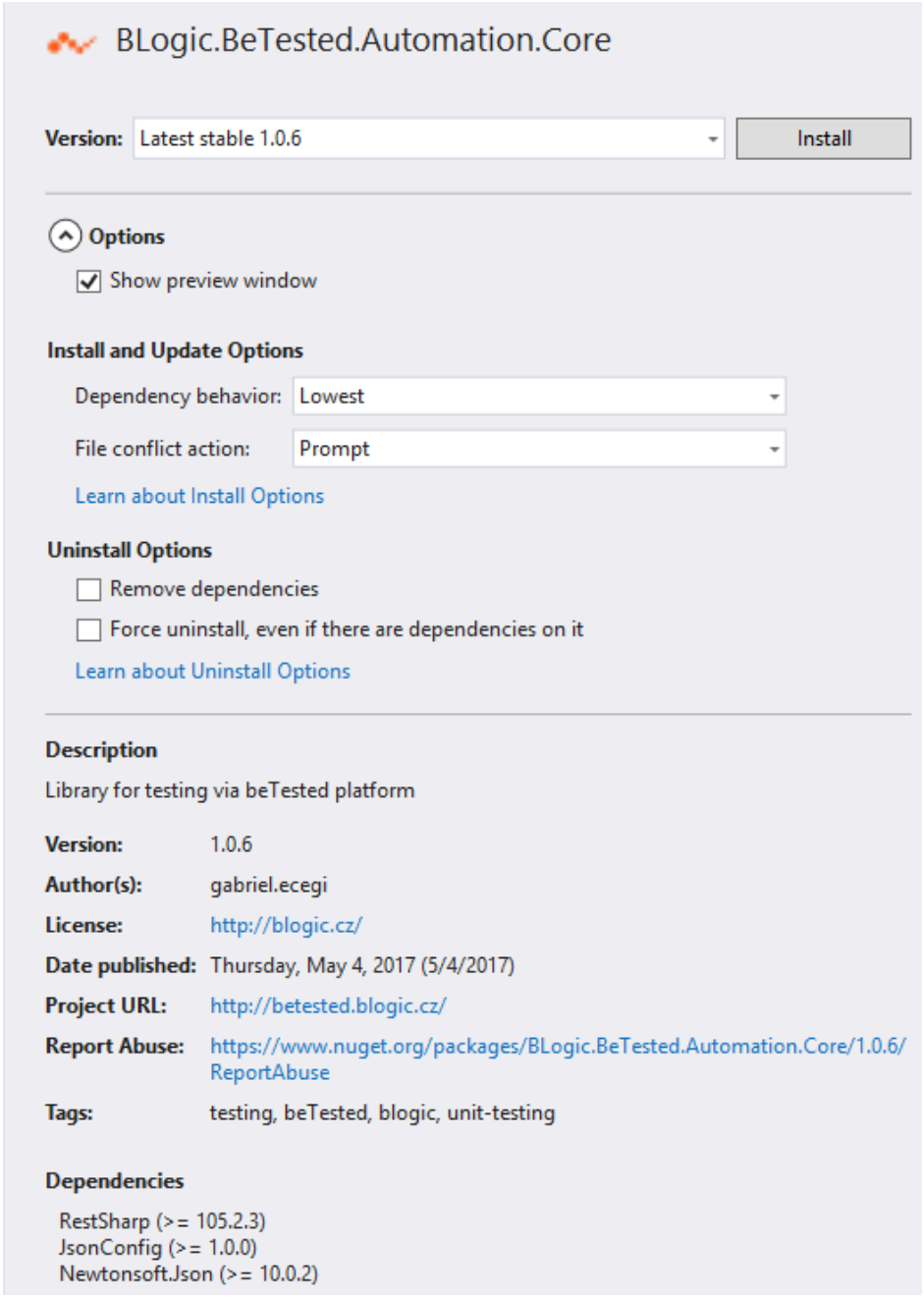
11.2.2 **NuGet Package Manager UI**

NuGet Package Manager UI poskytuje grafické rozhranie na správu - inštalácia, update, zmazanie balíčkov priamo vo Visual Studio, obrázok 20 a 21.

Stačí vybrať projekt, ktorého NuGet balíčky chceme spravovať a vybrať „Manage NuGet Packages“ a v okne Browse vyhľadať balíček podľa projektu a kliknúť na tlačidlo inštalovať.



Obrázok 20 Balíčky dostupné v NuGet Package Manager UI



Obrázok 21 Inštalácia balíčku v NuGet Package Manager UI

11.3 Použitie beTested Automation

Pri návrhu riešenia nasadenia automatizovaných testov bola prvoradá možnosť nasedenia na staršie projekty, v ktorých by nebola možnosť využívať najnovšiu verziu frameworku .NET, alebo projekty postavené na iných platformách. Preto môže byť testovací projekt vedený mimo projektu, ktorý má byť testovaný. V tejto kapitole je popísaný správny postup, ako vytvoriť testovací projekt využívajúci možnosti vytvoreného riešenia. Je tu ukázané a vysvetlené, ako vytvoriť automatizované UI testovanie webovej aplikácie s návrhovým vzorom Page Object Pattern s ukladaním záznamov o testovaní do beTested.

11.4 Odporúčaný postup nasadenia automatizovaného testovania

1. Vytvoriť projekt na beTested
2. Vytvoriť potrebné scenáre, ktoré budú automatizovane testované
3. Stiahnuť NuGet balíčky
4. Vytvoriť štruktúru projektu s potrebnými súbormi
5. Vygenerovať projekt s Page Objects Pattern objektami
6. Vytvoriť testy v C#
7. Spustiť testy
8. Analyzovať výsledky testov na beTested

11.5 Odporúčané požiadavky pre vývoj

1. Účet na beTested
2. Windows 8.1 a vyššie
3. Visual Studio 2015 a vyššie
4. Google Chrome 57 a vyššie

11.6 Vytvorenie štruktúry na webe beTested

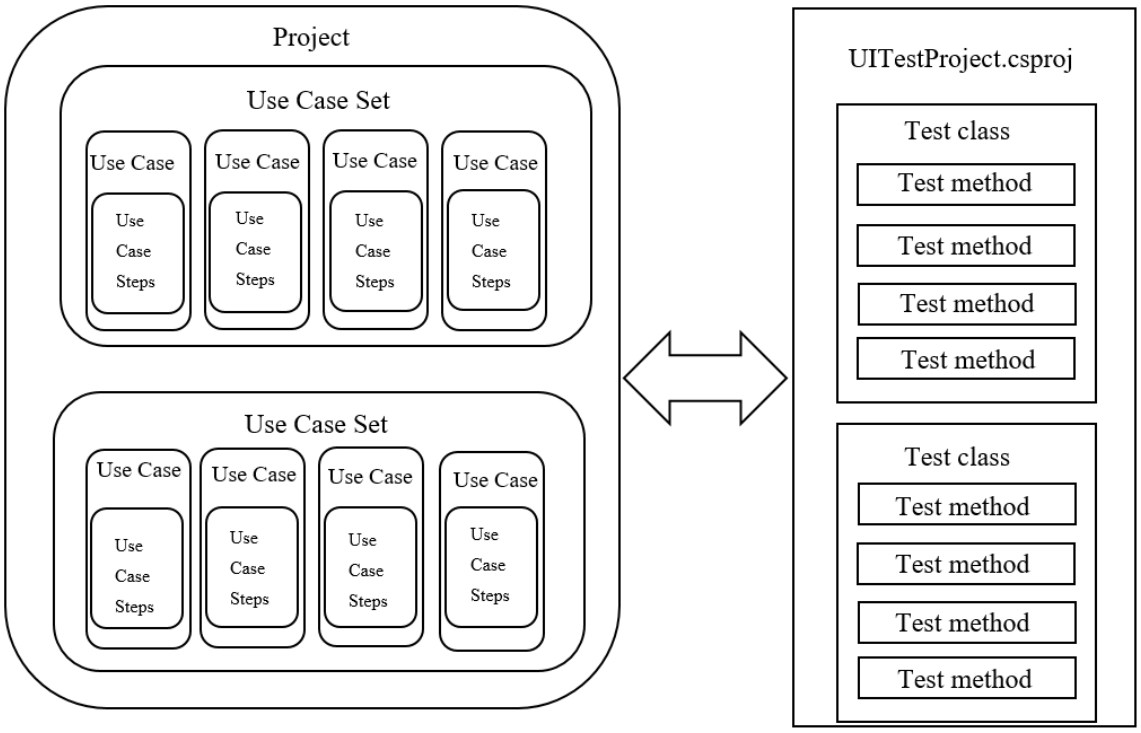
Pre vytvorenie štruktúry na webe beTested je potrebné mať na beTested vytvorený a aktivovaný účet. Registrácia a aktivácia účtu je intuitívna a zadarmo tak, ako všetky funkcionality platformy beTested. Po prihlásení si používateľ môže vytvoriť nový projekt, vyplní potrebné údaje a projekt uloží. Do takto vytvoreného projektu už môže jednoducho zakladať entity a vytvárať testovacie scenáre.

Po prekliknutí sa do projektu používateľ vytvorí Use Case Set - tento bude v C# testoch reprezentovaný jednou triedou. Typickým príkladom Use Case Set je autentifikácia. Do Use Case Set vkladáme Use Case predstavujúce testovacie metódy v triedach. Príkladom Use Case je Prihlásenie sa s validnými prihlasovacími údajmi. Do Use Case sa dajú písať aj Use

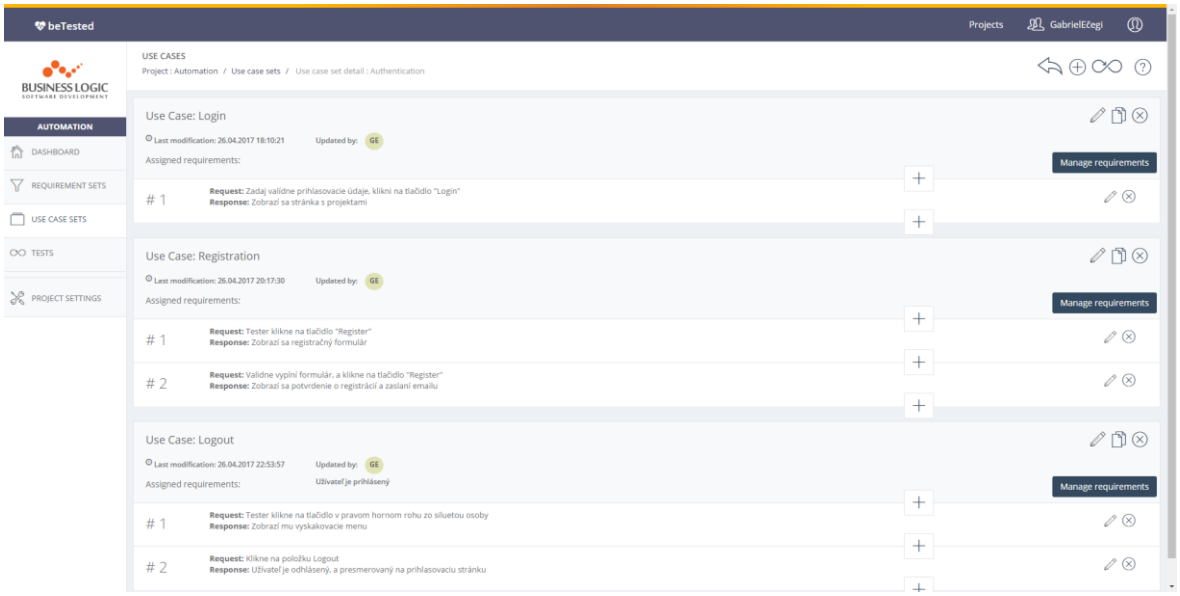
Case Steps, predstavujúce kroky k testovacieho scenára. Aj keď beTested Automation nepracuje s entitami Use Case Step, je odporúčané mať tieto kroky napísané, aby tester alebo programátor pracujúci s testami mal lepší prehľad o testovanej funkcionalite.

Vizualizácia vzťahu medzi aplikáciou a testami je zobrazená na obrázku číslo 22.

Vzorový Use Case Set s Use Cases a Use Case Steps je na obrázku číslo 23.



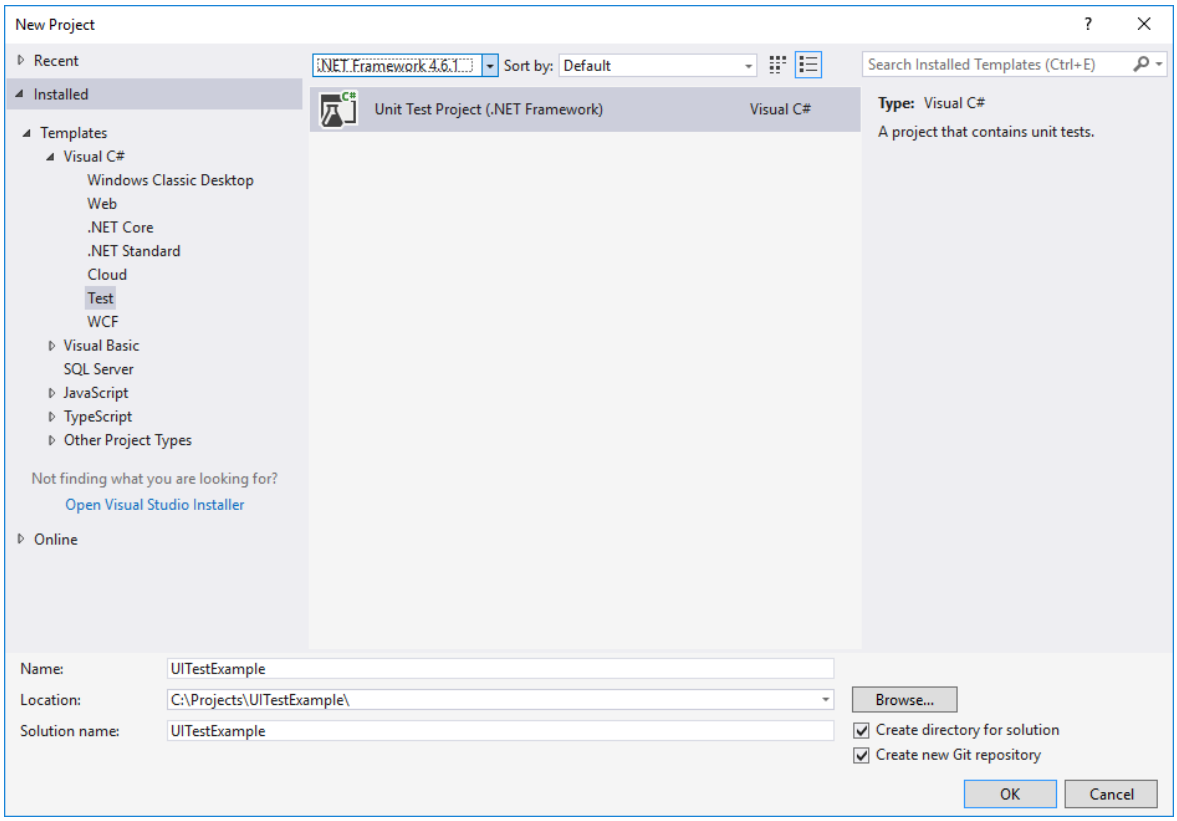
Obrázok 22 Vzťah štruktúry testov v aplikácii beTested a v testovacom projekte



Obrázok 23 Vzorová testovacia entita

11.7 Vytvorenie projektu

Vo Visual Studio treba vytvoriť nový projekt typu Unit Test Project s cieľovým frameworkom .NET Framework 4.6.1 - obrázok 24. Týmto sa vytvorí aj solution s týmto projektom.



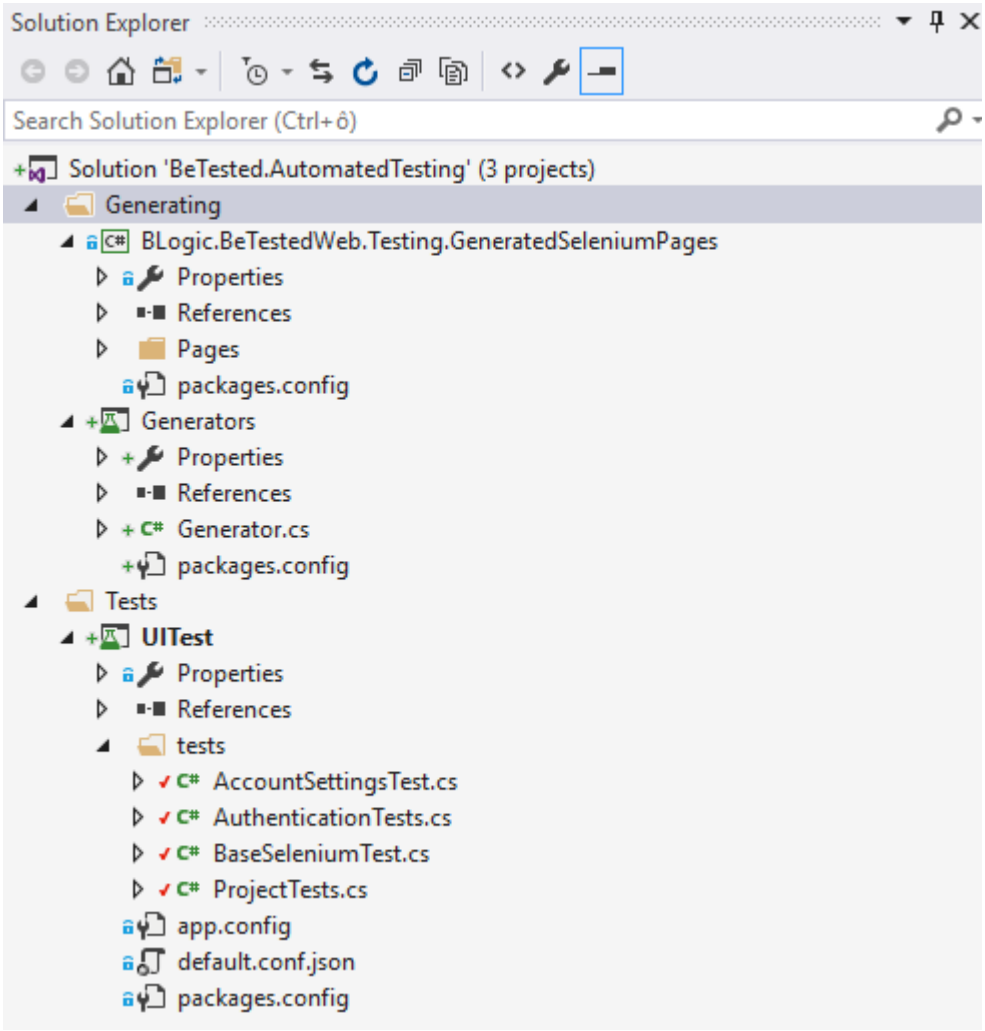
Obrázok 24 Vytvorenie testovacieho projektu vo Visual Studio 2017

11.8 Stiahnutie NuGet balíčkov

Pre použitie je nutné nainštalovať do nového projektu NuGet balíčky BLogic.BeTested.Automation.Core a BLogic.Modules.Selenium v najvyšších verziách ľubovoľným spôsobom.

11.9 Vytvorenie štruktúry projektu

Ukážkový projekt je rozdelený na dve zložky Generating a Tests pre lepšie rozdelenie logiky projektu. Na obrázku 25 je vidieť ako by mal takýto projekt vyzerať.



Obrázok 25 Odporúčaná štruktúra projektu

11.9.1 Generating

V tejto zložke sa nachádzajú testy alebo spustiteľné programy generujúce projekty a projekty vygenerované generátormi.

11.9.2 Tests

V zložke Tests sa nachádzajú UI testy, tieto využívajú vygenerované projekty zo zložky Generating.

11.10 Vygenerovanie modelového projektu

Generovanie projektov má na starosti NuGet balíček BLogic.Modules.Selenium, ktorý dokáže analyzovať webové stránky a na základe ich obsahu vytvoriť C# triedy podľa návrhového vzoru Page Object Pattern. Takto vygeneruje projekt, ktorý si už stačí len pridať do solution.

Do generátora stačí vložiť stránky, ktoré chceme používať, názov vygenerovaného projektu, relatívnu cestu k NuGet balíčkom a v prípade, že na stránky, ku ktorým chceme pristúpiť, vyžadujú autentifikáciu, tak aj prihlasovaciu URL. Príklad funkčnej metódy je vidieť na obrázku 26.

```
[TestClass]
0 references | 0 changes | 0 authors, 0 changes
public class Generator
{
    [TestMethod]
    0 references | 0 changes | 0 authors, 0 changes
    public void GenerateProject()
    {
        var pages = new Dictionary<string, string>
        {
            {"Login", "http://blogictest-test0.blogic.cz/security/login"},
            {"Registration", "http://blogictest-test0.blogic.cz/security/registration"},
            {"Projects", "http://blogictest-test0.blogic.cz/project/index"},
            {"AccountSettings", "http://blogictest-test0.blogic.cz/projectsettings/index"}
        };

        SeleniumMapping.Configure(new AnalyzerConfig
        {
            LoginUrl =
                "http://blogictest-test0.blogic.cz/login?User.Login=automation%40betested.cz&User.Password=123456",
            PagesToScan = pages
        }, new GeneratorConfig
        {
            ProjectName = "BeTestedWeb",
            RelativePathToPackages = "../"
        });
        SeleniumMapping.Refresh();
    }
}
```

Obrázok 26 Testovacia metóda generujúca projekt

11.11 Vytvorenie vzorového testu

UI testy sa vytvárajú do zložky Tests. Tieto testy sa vytvárajú ako unit testy, pričom je dobrou praxou, aby tieto testy dedili od testu zdieľaný obsah - napríklad metódu, ktorá sa má zavolať po ukončení všetkých testov, alebo po každej testovacej metóde - odhlásenie z aplikácie.

Nad názvom triedy je v atribúte názov Use Case Set, ktorému trieda odpovedá. Podobne je v atribúte UseCase názov Use Case, ktorému zasa odpovedá jedna trieda. Tieto informácie sa zisťujú pri inicializácii triedy v metóde Init, tu sa vytvára instancia triedy Automation, ktorá berie ako parameter konfiguračný objekt zo súboru default.conf.json.

Metóda Save() sa zavolá po skončení všetkých testovacích metód v testovacej triede AccountSettingsTests, prichystá výsledky na testovania triedy a pošle ich na BLogic.BeTested.Automation.Api.

Po skončení spojenia s REST API sa začne testovať ďalšia trieda, a ak žiadna ďalšie nie je vo fronte na testovanie, tak test skončí.

Na obrázku 27 je vzorový test, testujúci zmenu názvu spoločnosti v nasteveniach účtu.

```
[TestClass]
[UseCaseSet("Account settings")]
0 references | Gabriel Ecegi, Less than 5 minutes ago | 1 author, 2 changes
public class AccountSettingsTests : BaseSeleniumTest
{
    private static Automation _beTested;

    [ClassInitialize]
    0 references | Gabriel Ecegi, 2 hours ago | 1 author, 1 change
    public static void Init(TestContext context)
    {
        _beTested = new Automation(Config.Default.beTested);
    }

    [ClassCleanup]
    0 references | Gabriel Ecegi, 2 hours ago | 1 author, 1 change
    public static void Save()
    {
        _beTested.SaveData();
    }

    [TestMethod]
    [UseCase("Change company name")]
    0 references | Gabriel Ecegi, Less than 5 minutes ago | 1 author, 2 changes
    public void ChangeCompanyName()
    {
        //Arrange
        var companyName = $"New name - {Guid.NewGuid()}";

        //Action
        Pages.Login.SignIn();

        Pages.AccountSettings.AccountSettingsIndexUrl.GoToUrlSafely();
        Pages.AccountSettings.CompanyNameInput.SetTextSafely(companyName);
        Pages.AccountSettings.SaveAccountDetailBtn.ClickSafely();

        Pages.AccountSettings.AccountSettingsIndexUrl.GoToUrlSafely();

        //Assert
        _beTested.Test(() => SeleniumAssert.TextEquals(Pages.AccountSettings.CompanyNameInput, companyName));
    }
}
```

Obrázok 27 Ukážkový test

11.12 Využitie Page Object Pattern v testoch

Ako je vidieť na obrázku 27, testy využívajú Page Object Pattern, ktorý pridáva úroveň abstrakcie do testov. K objektom stránky sa pristupuje cez objekt *Pages*, ktorý slúži ako Factory objekt. Objekt Pages má properties reprezentujúce reálne stránky, napríklad k stránke na prihlásenie - Login sa pristúpi nasledovne *Pages.Login*. Objekty stránok obsahujú properties, reprezentujúce UI elementy, toto umožňuje jednoduché pristupovanie k takýmto UI elementom. Pristúpenie k elementu tlačidla v prihlasovacom formulári: *Pages.Login.LoginButton.*

Tento návrhový vzor sa umožňuje zbaviť repetitívneho hľadania elementov. Keďže takáto syntax sa správa len ako sled príkazov bez zložitých funkcií, cyklov alebo podmienok, testy v Page Object Pattern odtieňujú používateľa od programovania do takej miery, že písanie testov dokáže zvládnuť aj zručný tester. Pre porovnanie je na obrázku 28 vidieť rozdiel medzi použitím Page Object Pattern a prístupu hľadania elementov cez instanciu WebDriver.

Je teda vidieť, že využitie Web Object Pattern nie len skráti dĺžku testov, ale taktiež test sprehľadní.

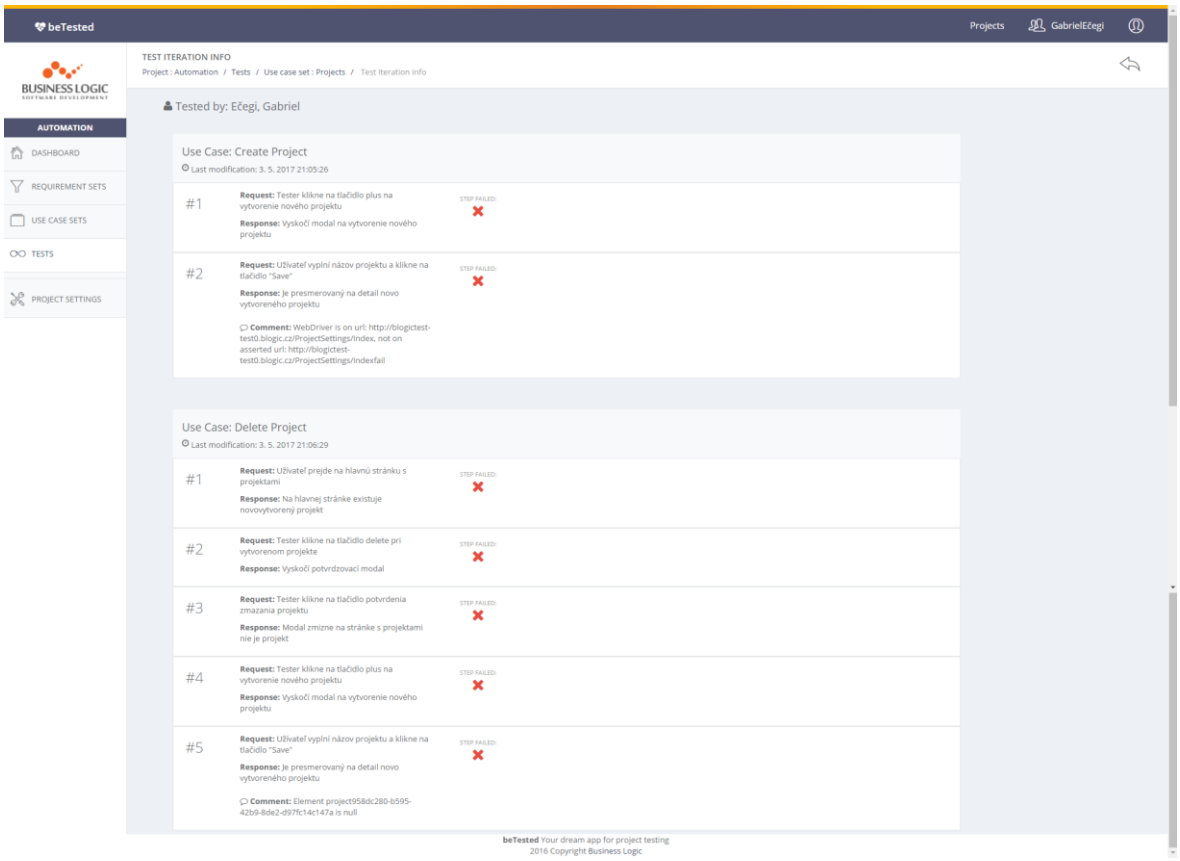
Obrázok 28 Porovnanie testu bez a s návrhovým vzorom Page Object Pattern

11.13 Analýza testov na webe beTested

Po skončení testov je možné ich analyzovať na webe beTested. Výsledky sa nachádzajú pod projektom, ktorý bol uvedený v konfiguračnom súbore default.conf.json.

Výsledky testov nájdeme v sekcii Tests a následne príslušný Use Case. Tu to bude posledná iterácia testu.

Ako je vidieť na obrázku 29, pri výnimke pri metóde, resp. stave „Fail“ pri tvrdení sa do komentára zapíše aj text výnimky, ten sa vždy nachádza v poslednom kroku Use Case.



Obrázok 29 Zobrazenie výsledkov testu na beTested

ZÁVĚR

Moderné testovanie softvéru je neoddeliteľnou súčasťou jeho vývoja a údržby. Pri rastúcom dopyte po webových aplikáciách sa zvyšuje dopyt po ich testovaní tak, ako aj dopyt po správnych metodikách testovania. Spektrum cieľov testovania vedie aj k širokému spektru postupov k vybraným cieľom a analytik musí byť oboznámený s komplexnou problematikou testovania softvéru, návrhu testov, aby mohol navrhnúť také riešenie, ktoré by viedlo k splneniu cieľov kvality softvéru. Navyše, v súčasnej dobe, kedy sa automatizácia rozširuje do všetkých oblastí ľudskej spoločnosti, tento fenomén nevynecháva ani odvetvie vývoja a testovania softvéru, pričom sa tu kladú vysoké nároky na správnu implementáciu automatizácie. Výstupom bakalárskej práce je návrh takéhoto riešenia.

V teoretickej časti sú popísané základné spôsoby a možnosti testovania softvéru, je vysvetlené do akých úrovní sa testy rozdeľujú a za akých častí sa testy skladajú. V ďalších častiach je popísaná automatizácia testovania, výhody a nástrahy pri automatizácii testovania softvéru. Keďže bakalárska práca sa zaujíma najmä o testovanie webových aplikácií, sú tu popísané najpoužívanejšie nástroje pre testovanie webových aplikácií, spolu s výhodami a nevýhoda ich použitia. V teoretickej časti je špecifickejšie popísaný framework Selenium, keďže sa ním zaoberá praktická časť. Práca obsahuje popis Selenium IDE, vývoj Selenium-WebDriver a základné možnosti využitia Selenium GRID.

V praktickej časti je navrhnuté a vytvorené praktické riešenie použitia automatizovaných testov s integráciu platformy beTested. Implementácia spočíva v návrhu takého postupu, ktorý je čo najmenej náročný na programátorské znalosti používateľa a umožňuje použitie v už fungujúcich systémoch. Toto je dosiahnuté využitím Page Object Pattern, ktorý zjednodušil a sprehľadnil testy do takej úrovne, že ich dokážu písať aj menej zdatní programátori.

Riešenie sa skladá z niekoľkých projektov poskytujúcich modularitu a lepšiu možnosť distribúcie projektov. Projekty si je možné zadarmo stiahnuť do projektu cez balíčkovací systém NuGet. Projekty sú prepojené s platformou beTested, kde sa výsledky testov ukladajú a je možné si ich zobraziť na webe tejto aplikácie.

Celkovo bola vyvinutá nielen fyzická realizácia riešenia problému, ale bola naštudovaná ešte cennejšia znalosť problematiky použitia automatizovaného testovania, ktorú možno využiť na skvalitnenie vyvíjaného softvéru.

SEZNAM POUŽITÉ LITERATURY

[1] Introducing Selenium. *Selenium - Web Browser Automation* [online]. 2017 [cit. 2017-04-18]. Dostupné z: http://www.seleniumhq.org

[2] Selenium IDE. *Selenium - Web Browser Automation* [online]. 2017 [cit. 2017-04-18]. Dostupné z: http://www.seleniumhq.org

[3] Selenium WebDriver. *Selenium - Web Browser Automation* [online]. 2017 [cit. 2017-04-18]. Dostupné z: http://www.seleniumhq.org

[4] IWebElement. *Github* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_ISearchContext_FindElement.htm

[5] By. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/T_OpenQA_Selenium_By.htm

[6] By.ClassName. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_By_ClassName.htm

[7] By.CssSelector. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_By_CssSelector.htm

[8] By.Id. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_By_Id.htm

[9] By.LinkText. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_By_LinkText.htm

[10] By.Name. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_By_LinkText.htm

[11] By.PartialLinkText. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_By_PartialLinkText.htm

[12] By.TagName. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_By_TagName.htm

[13] By.XPath. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_By_XPath.htm

[14] IWebDriver.FindElements. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_ISearchContext_FindElements.htm

[15] IWebDriver.Navigate. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_IWebDriver_Navigate.htm

[16] INavigation.Back. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-19]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_INavigation_Back.htm

[17] INavigation.Forward. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-20]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_INavigation_Forward.htm

[18] INavigation.GoToUrl. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-20]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_INavigation_GoToUrl.htm

[19] INavigation.GoToUrl. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-20]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_INavigation_GoToUrl_1.htm

[20] INavigation.Refersh. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-20]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_INavigation_Refresh.htm

[21] In selenium webdriver what is manage() [driver.manage()]. *Stack overflow* [online]. Stack Exchange, 2016 [cit. 2017-04-20]. Dostupné z: http://stackoverflow.com/questions/37771762/in-selenium-webdriver-what-is-manage-driver-manage

[22] IOptions.Cookies. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-20]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/P_OpenQA_Selenium_IOptions_Cookies.htm

[23] IOptions.Logs. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-20]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/P_OpenQA_Selenium_IOptions_Logs.htm

[24] IOptions.Window. *GitHub* [online]. Software Freedom Conservancy, 2013 [cit. 2017-04-20]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/P_OpenQA_Selenium_IOptions_Window.htm

[25] Selenium WebDriver Switch Window Commands. *TOOLSQA* [online]. Londýn: TOOLSQA [cit. 2017-04-20]. Dostupné z: http://toolsqa.com/selenium-webdriver/switch-commands/

[26] PageObject. *Martinfowler.com* [online]. 2013 [cit. 2017-04-20]. Dostupné z: https://martinfowler.com/bliki/PageObject.html

[27] Page Object Design Pattern. *SeleniumHQ* [online]. 2017 [cit. 2017-04-20]. Dostupné z: http://www.seleniumhq.org/docs/06_test_design_considerations.jsp#page-object-design-pattern

[28] PageFactory. *GitHub* [online]. 2015 [cit. 2017-04-20]. Dostupné z: https://github.com/SeleniumHQ/selenium/wiki/PageFactory

[29] OpenQA.Selenium.Support.PageObjects Namespace. *GitHub* [online]. 2013 [cit. 2017-04-20]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/N_OpenQA_Selenium_Support_PageObjects.htm

[30] GRAHAM, Dorothy a Mark FEWSTER. Experiences of test automation: case studies of software test automation. Upper Saddle River, NJ: Addison-Wesley, c2012. ISBN 03-217-5406-9.

[31] ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. *Řízení kvality softwaru: průvodce testováním*. Brno: Computer Press, 2013. ISBN 978-802-5138-168.

[32] Selenium alternatives for testing automation. *Screenster* [online]. AgileEngine, 2016 [cit. 2017-04-21]. Dostupné z: http://screenster.io/selenium-alternatives-for-testing-automation/

[33] Selenium Grid. *TOOLSQA* [online]. Londýn: TOOLSQA [cit. 2017-04-22]. Dostupné z: http://toolsqa.com/selenium-webdriver/selenium-grid/

[34] Selenium Grid Tutorial: Step by Step Guide with Example. *Guru99* [online]. [cit. 2017-04-22]. Dostupné z: http://www.guru99.com/introduction-to-selenium-grid.html

[35] Selenium-Grid. *SeleniumHQ* [online]. Selenium Project, 2017 [cit. 2017-04-22]. Dostupné z: http://www.seleniumhq.org/docs/07_selenium_grid.jsp

[36] What is a headless browser? *Andrew R H Girdwood* [online]. [cit. 2017-04-22]. Dostupné z: http://blog.arhg.net/2009/10/what-is-headless-browser.html

[37] Protractor. *Protractor* [online]. 2017 [cit. 2017-04-22]. Dostupné z: http://www.protractortest.org

[38] BACH, James. Test Automation Snake Oil. *Windows Tech Journal*. 1996, **1996**(10), 6.

[39] Test Automation and its Benefits. *Ranorex* [online]. [cit. 2017-04-23]. Dostupné z: http://www.ranorex.com/why-test-automation.html

[40] JENKINS, Nick. *Software Testing Primer*. 2. 2017.

[41] The Four Levels of Software Testing. *Segue technologies* [online]. Ohio, USA, 2015 [cit. 2017-04-24]. Dostupné z: http://www.seguetech.com/the-four-levels-of-software-testing

[42] What are Software Testing Levels? *ISTQB Exam Certification* [online]. [cit. 2017-04-24]. Dostupné z: http://istqbexamcertification.com/what-are-software-testing-levels/

[43] Software Testing - Levels. *Tutorials point* [online]. [cit. 2017-04-24]. Dostupné z: https://www.tutorialspoint.com/software_testing/software_testing_levels.htm

[44] What is Regression Testing? Test Cases, Tools & Examples. *Guru99* [online]. [cit. 2017-04-24]. Dostupné z: http://www.guru99.com/regression-testing.html

[45] Traceability from Use Cases to Test Cases. *IBM* [online]. 2006 [cit. 2017-04-25]. Dostupné z: https://www.ibm.com/developerworks/rational/library/04/r-3217/

[46] - What's the difference between a use case and a test case? *Project connections* [online]. [cit. 2017-04-25]. Dostupné z: http://www.projectconnections.com/knowhow/burning-questions/difference-between-use-case-and-test-case.html

[47] NAGEL, Christian. Professional C# 6 and .Net Core 1.0. Indianapolis, IN: Wrox, a Wiley brand, published by John Wiley & Sons, 2016. ISBN 111909660X

[48] FREEMAN, Adam. Pro ASP.NET MVC 5. 5th ed. New York, N.Y.: Apress, 2013. ISBN 978-143-0265-306.

[49] Swagger špecifikácia. *Http://swagger.io/specification/* [online]. 2017 [cit. 2017-05-01]. Dostupné z: http://swagger.io/specification/

[50] ASP.NET | Microsoft Docs [online]. Microsoft, 2017 [cit. 2017-01-30]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/

[51] Create NuGet package. *Microsoft Docs* [online]. Microsoft, 2017 [cit. 2017-05-02]. Dostupné z: https://docs.microsoft.com/en-us/nuget/create-packages/creating-a-package

[52] IWebDriver.Quit. *GitHub* [online]. [cit. 2017-05-06]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_IWebDriver_Quit.htm

[53] IWebDriver.Close. *GitHub* [online]. [cit. 2017-05-06]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_IWebDriver_Close.htm

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

UI	User Interface
XML	Extensible Markup Language
REST	Representational State Transfer
API	Application Programming Interface
IDE	Integrated Development Enviroment
JSON	Javascript Object Notation
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
URL	Uniform Resource Locator
DOM	Document Object Model

SEZNAM OBRÁZKŮ

Obrázok 1 Vzťah medzi fázou, v ktorej je defekt identifikovaný a.....13
Obrázok 2 Hierarchické zobrazenie štruktúry testov [45].....16
Obrázok 3 Zobrazenie vzťahu testovacích entít 18
Obrázok 4 Ukážka testu v o frameworku Protractor [37].....24
Obrázok 5 Práca s IWebDriver v praktickej časti.....27
Obrázok 6 Page Factory použitá v praktickej časti.....30
Obrázok 7 Page Object Pattern – stránka registration použitá v praktickej časti 31
Obrázok 8 Hub a uzly [34] 32
Obrázok 9 Business flow BeTested.Automation.....35
Obrázok 10 Ukážkový konfiguračný súbor 36
Obrázok 11 Nastavenie Build Action u konfiguračného súboru 37
Obrázok 12 Testovacia metóda.....42
Obrázok 13 Testovanie triedy.....43
Obrázok 14 Proces prijatia výsledkov testov na strany API.....45
Obrázok 15 Vývojový diagram uloženia objektu do databáze 46
Obrázok 16 Ukážka XML štruktúry pre swagger.....47
Obrázok 17 Vygenerované view na základe XML.....47
Obrázok 18 .nuspec súbor projektu BLogic.BeTested.Automation.Core 54
Obrázok 19 Dostupnosť balíčku dostupná na webe nuget.org 54
Obrázok 20 Balíčky dostupné v NuGet Package Manager UI 56
Obrázok 21 Inštalácia balíčku v NuGet Package Manager UI 56
Obrázok 22 Vzťah štruktúry testov v aplikácii beTested a v testovacom projekte 58
Obrázok 23 Vzorová testovacia entita 59
Obrázok 24 Vytvorenie testovacieho projektu vo Visual Studio 2017 59
Obrázok 25 Odporúčaná štruktúra projektu.....60
Obrázok 26 Testovacia metóda generujúca projekt.....61
Obrázok 27 Ukážkový test.....62
Obrázok 28 Porovnanie testu bez a s návrhovým vzorom Page Object Pattern.....63
Obrázok 29 Zobrazenie výsledkov testu na beTested 64

SEZNAM PŘÍLOH

P I CD disk s bakalárskou prácou a súbormi zdrojového kódu