Univerzita Hradec Králové
Fakulta informatiky a managementu

BAKALÁŘSKÁ PRÁCE

2019 Petr Šroubek

Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Automatizované testování webových aplikací

Bakalářská práce

Autor: Petr Šroubek
Studijní obor: Informační management

Vedoucí práce: Ing. Tereza Otčenášková, BA

Hradec Králové 2019

Prohlášení:

Prohlašuji, že jsem tuto bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 25.4.2019 Petr Šroubek

Poděkování:

Rád bych poděkoval vedoucí mé bakalářské práce Ing. Tereze Otčenáškové, BA za odborné vedení práce, připomínky a v neposlední řadě nekonečnou trpělivost. Dále bych chtěl poděkovat Ing. Ondřeji Růžičkovi, který mi předal cenné znalosti a zkušenosti v oblasti automatizovaného testování webových aplikací.

Anotace

Testování je dnes podstatnou částí procesu vývoje informačních systémů. Žádný informační systém by se bez řádného otestování neměl dostat na trh. Jelikož se zlepšují i metody testování a úspora jak finanční, tak časová, je žádoucí, mnoho firem zavádí automatizované testování svých vyvíjených informačních systémů.

Právě problematikou automatizovaného testování se tato práce zabývá. V teoretické části je automatizované testování vymezeno. Uvedeny jsou důvody jeho realizace. Zmíněny jsou i způsoby jeho provádění a nástroje, které se k němu používají. Praktická část práce nastiňuje, jak automatizovaný test vypadá a jak probíhá. Vytvořen je konkrétní test pro otestování funkcionality webové aplikace Cestovní příkazy v rámci reálného testovacího prostředí firmy Ders, s.r.o.

Klíčová slova

Automatizované testování, Selenium WebDriver, testování, webové aplikace.

Annotation

Nowadays, testing represents an important part of the software development process. No developed software should be launched to the market without being thoroughly tested. Since methods of testing are continuously improving and both the financial and time savings are desirable, many companies implement automated testing of the software they are developing.

The problematics of automated testing is the main focus of this bachelor thesis. In the theoretical part testing and its purposes are defined. Furthermore, the ways of its execution and the tools employed are introduced. The practical part of the thesis outlines the design of an automated test including the description of its running. The specific test is created to test the functionality of the web application Cestovní příkazy (Travel Orders) in the real testing environment of a company Ders, s.r.o.

Keywords

Automated Testing, Selenium WebDriver, Testing, Web Applications.

Obsah

Seznam obrázků

1 Úvod

Jelikož je testování nedílnou a podstatnou součástí vývoje informačních systémů, tak i samotné testování prochází vývojem. V dnešní době a čím dále tím více se již nejedná jen o manuální testování, ale na popularitě nabývá právě automatizované testování. A automatizací testování se zabývá i tato práce, a to konkrétně automatizovaným testováním webových aplikací. Webové aplikace jsou dnes již standardem a je běžné, že může uživatel aplikaci ovládat v prostředí svého webového prohlížeče a není nutné si aplikaci stahovat a instalovat do svého zařízení. Spousta firem se tím pádem v dnešní době zabývá vývojem webových aplikací, které naleznou využití v nespočetném množství oborů (školství, doprava, zdravotnictví a jiné). A jak již bylo řečeno, s jejich vývojem také úzce souvisí testování.

Automatizované testování pravděpodobně manuální testování nikdy zcela nenahradí, protože není možné (alespoň zatím) nahradit lidský faktor, ale může ho dle možností vhodně doplňovat. Firmy mohou od zavedení automatizovaného testování odradit prvotní náklady, které jistě při zavádění jsou. Dle studií se ale většinou ukázalo, že z dlouhodobého hlediska, pokud se firma pro zavedení rozhodne, automatizované testování náklady spíše snižuje. Je také pravda, že ne vždy se automatizované testování hodí a jsou určitě softwary, které automatizovaně testovat nelze téměř vůbec. A je v každém případě nezbytné si zavedení automatizovaného testování řádně rozmyslet a naplánovat.

Avšak pro určité typy webových aplikací se automatizované testování skvěle hodí. Ideální případ je v praktické části této práce (kapitola 7). Je to aplikace, u které je jasné, jaký je její cíl. A kroky vedoucí k tomuto cíli jsou téměř vždy identické či s malými odchylkami. Díky tomu, že je posloupnost kroků předem daná a bude se vždy opakovat, je možné takovýto průchod zautomatizovat, dokonce je to při nejmenším vhodné. Pro takové testování webových aplikací také existuje několik specializovaných nástrojů. O nástrojích, které byly použity v této práci se v kapitole Nástroje automatizovaného testování také stručně píše a v praktické části je vidět jejich použití na reálném testovacím prostředí.

Test byl vytvořen pro webovou aplikaci Cestovní příkazy (zkratka CP) firmy Ders s.r.o. Firma Ders s.r.o. se zabývá vývojem softwaru, a to především právě webových aplikací zejména pro vysokoškolské instituce. Webová aplikace Cestovní příkazy je jednou z aktuálně

vyvíjených. A než se cokoliv změní v produkční verzi, je potřeba změny provést a otestovat na testovacím prostředí, ze kterého je i ukázka návrhu testu v praktické části práce.

2 Cíl a metodika bakalářské práce

Tato kapitola vysvětluje, co je cílem práce a jaká metodika byla v bakalářské práci použita.

2.1 Cíl práce

Hlavním cílem práce je návrh a vytvoření automatizovaného testu webové aplikace Cestovní příkazy firmy Ders s.r.o. za pomocí především nástrojů Selenium WebDriver a TestNG. Dílčím cílem práce je seznámení s problematikou testování jako takového a následně testování automatizovaného, u kterého je problematika zkoumána podrobněji.

2.2 Metodika práce

Bakalářská práce jejímž cílem je zejména návrh a tvorba automatizovaného testu webové aplikace je rozdělena na dvě hlavní části, které jsou děleny na další, dílčí části.

V první teoretické části (kapitola 3) se práce zabývá problematikou testování informačních systémů a podrobněji se práce věnuje automatizovanému testování. Práce vysvětluje, co je automatizované testování, jaké jsou jeho výhody a proč automatizované testování zavádět. Dále také práce uvádí srovnání automatizovaného a manuálního testování. Teoretická část ale také představuje nástroje automatizovaného testování, webové aplikace a také problematiku defektů, kolem kterých se celý proces testování točí. Informace byly čerpány z relevantní literatury a také z vědeckých databází.

V praktické části (kapitola 7) se pak práce zabývá návrhem automatizovaného testu v reálném prostředí firmy Ders s.r.o. V této části je popsán průběh testu, který je doprovázen množstvím obrázků. Obrázky ukazují, jak vypadá průběh testu v samotné webové aplikaci, ale v příloze jsou i obrázky se samotným kódem testu. V této části jsou také popsány technologie, které byly k tvorbě testu použity.

3 Testování informačních systémů

Roudenský (2013, s. 45) definuje testování jako „proces řízeného spouštění softwarového produktu s cílem zjistit, zda splňuje specifikované či implicitní potřeby uživatelů“. Testování lze tedy chápat jako proces analýzy testovaného systému a sběr informací o jeho kvalitě. Jeho kvalita je následně použita jako měřítko pro naplnění požadavků klientů. Případné chyby jsou vedlejším produktem tohoto procesu. Před začátkem testování je potřeba určit, co je cílem testování. Proto by mělo testování vždy přecházet plánování, kdy se testy navrhují a vyhodnocují. Testování spadá pod řízení kvality. Testování samo o sobě kvalitu testovaného systému nezvyšuje, nicméně poskytuje pro zajištění kvality potřebné vstupy (Roudenský, 2013).

Proces testování lze rozdělit na automatizované a manuální testování. Při manuálním testování zastává tester roli koncového uživatele a vykonává funkce softwarového System Under Testu (SUT - označení konkrétního právě testovaného systému), aby ověřil, že chování softwaru je takové, jaké se očekává. K zajištění úplnosti testování, tester často postupuje podle psaného testovacího plánu, který obsahuje sadu testovacích případů. Automatizované softwarové testování znamená automatizaci testovacích aktivit. Přesněji automatizace testování je využití specializovaného softwaru ke kontrole vykonání daných testů a porovnání výstupů skutečných s předpokládanými výstupy. Při přechodu na automatizaci je většinou prvotní tendence aplikovat automatizaci na cokoliv, co předtím dělali testeři manuálně. (Garousi, 2016)

3.1 Regresní testování

Před finálním uvedením softwaru na trh často nastane cyklus kódování-testování-oprava, který se může mnohokrát opakovat. Pokud je tedy třeba otestovat požadovanou funkcionalitu, je nutné testy provést nikoli pouze jednou, ale opakovaně. Účelem je odhalit, zda chyby, které byly nalezeny v předešlých testech, byly již opraveny, a zda do softwaru nepřibyly nové chyby. (Patton, 2002). Patton (2002, s. 186) definuje regresní testování jako proces opakovaného provádění testů.

Jakákoliv modifikace vyvíjeného systému, s sebou nese riziko, že se do již otestovaného systému nebo jeho části nově zavleče defekt, a proto se provádí regresní testování. Jeho cílem je ověření, že po změně části systému zůstávají ostatní součásti funkční a ve stejném stavu (Roudenský, 2013).

Regresní testování tedy využívá již existující testovací případy, aby se zachytila případná odchylka mezi aktuálními a dříve získanými výsledky. Proto vytváření nových testovacích případů pro regresní testování nemá smysl. (Roudenský, 2013)

Regresní testování je velmi nákladné nejen časově. Z tohoto důvodu většinou není prakticky možné, v případě, že nejsou všechny testovací případy automatizovány, otestovat celý systém znovu po každé změně. Proto analytik testování vybírá pouze relevantní testovací případy a to tak, aby pokryly co možná nejširší záběr oblastí, na které by daná změna mohla mít dopad. (Roudenský, 2013)

Regresní testování je tedy ideální příležitostí k automatizaci. Jelikož je to proces, při kterém je nutné systém často opakovaně testovat, je vhodné tento proces zautomatizovat. Samozřejmě musí být testovaný systém pro automatizaci vhodný. Nicméně ale v případě, že se systém pro automatizování hodí, tak automatizace celý tento opakovaný proces testování zásadně urychlí.

3.2 Automatizované testování

K automatizovanému testování se využívá specializovaný software. Software vykonává některé úlohy za testera. Automatizace testování má jednoduchý důvod, a to zefektivnění procesu testování. Provádění testovacích činností manuálně je totiž časově nesmírně náročné. I proto je v současné době automatizovanému testování věnována stále větší pozornost (Roudenský, 2013). Automatizované testování je vlastně princip záznamu a přehrávání tzv. maker. Jedná se o nahrání posloupnosti operací klávesnice a myši (vyplnění textových polí, klikání na tlačítka atd.) ve sledu, ve kterém by je uživatelé sami manuálně prováděli, při používání softwaru nebo návštěvě webové stránky, ale ve stále stejném pořadí a rychleji než reálný uživatel (Patton, 2002).

3.2.1 Výhody automatizovaného testování

Mnoho, možná většina, dnešních softwarových aplikací je psána formou webových aplikací, které jsou spouštěny v rámci internetového prohlížeče. V éře vysoce interaktivních softwarových procesů, kdy organizace používají alespoň nějakou formu agilní (flexibilní, reagující na nové požadavky přímo v průběhu vývoje) metodologie, je automatizace testování stále častějším požadavkem. (Seleniumhq.orq, 2019)

Automatizace testů přináší několik výhod. Mnoho organizací vnímá automatizaci softwarového testování jako prostředek ke snížení nákladů a snížení času potřebného na cyklus během vývoje softwaru. (Garousi, 2016) Zásadní je také snížení nákladů na údržbu systému. Sada testů oproti manuálnímu testování usnadní reprodukci chyb, protože bude produkovat stejné výsledky díky konzistentní sekvenci kroků a nastavení (Roudenský, 2013). Automatizované testování se vyplatí od určitého počtu opakování testu. Automatizované provádění testů stojí méně úsilí než manuální provádění testů. Automatizací se usnadní a zrychlí opakování testu a se vzrůstajícím počtem opakování daného testu nakonec úspora času a úsilí převáží náklady na jeho automatizaci. (Bureš, 2016). V případě provádění velkého množství testových případů, totiž mohou automatické nástroje znásobit rychlost provádění testových případů až 100krát nebo i 1000krát. (Patton, 2002).

Nicméně zřízení automatizovaného testování může selhat, pokud není automatizace aplikována v ten správný čas, ve správném kontextu a s patřičným přístupem. Pokud se automatizace implementuje správně, může značně snížit náklady na testování a může pomoci

zvýšit kvalitu softwaru. Podle studie nazvané „World Quality Report“, provedené francouzskou firmou Sogeti, je nyní jen 28 % testovacích případů automatizováno, přestože manažeři by si přáli toto číslo v budoucnu zvýšit. (Garousi, 2016) Jak se automatizace testování stává stále více a více mainstreamovou (významnější) v softwarovém průmyslu, se rozhodnutí o tom, kdy a co automatizovat stává velice důležitým, protože nesprávné rozhodnutí v tomto případě může vést ke zklamání a velkým nákladům. (Garousi, 2016)

I přesto, že v ideálním světě by si mnoho lidí představovalo plnou automatizaci testování, dle studií by v praxi tento přístup doporučilo jen 6%. Většina si myslí, že automatizovat všechny testy není praktické ani možné, a to především z hlediska času a rozpočtu (Garousi, 2016).

Většina komerčních nebo open-source softwarů dnes obsahuje automatizované testovací případy k ověření své funkcionality. A to především v případě softwarových projektů, které během svého vývoje prochází někdy i velkým množstvím verzí, protože automatizované testování se vyplácí především v případech regresního a repetitivního testování. Pro mnoho velkých systémů se automatizované testovací případy stále rozrůstají co do velikosti a komplexity (Garousi, 2016).

Typicky se testovací proces skládá z několika kroků od plánování testování ke specifikaci testů (design testovacích případů), vykonávání testů a report. Každý z těchto kroků může být proveden jak manuálně, tak automaticky. K lepšímu pochopení toho, jak je automatizace využívána během testovacího procesu, je níže prezentováno šest testovacích aktivit, u kterých je k automatizaci velký potenciál:

1. Design testovacího případu: design seznamu testovacích případů nebo testových požadavků k dosažení pokrytí kritérií nebo jiných technických cílů.
2. Vytváření testovacích skriptů: dokumentování testovacích případů v manuálních testových skriptech nebo automatizovaného kódu testu.
3. Provádění testu: spouštění testovacích případů SUT a zaznamenávání výsledků
4. Vyhodnocení: vyhodnocování výsledků testování (prošel nebo skončil s chybou)
5. Report výsledku testu: hlášení výsledků testů a chyb vývojářům
6. Management testů a jiné technické aktivity: management testů zahrnuje aktivity jako plánování, kontrola, monitorování a odhad pracnosti. Ostatní testovací aktivity zahrnují minimalizaci testovací sady a regresní výběr testů (Garousi, 2016).

Pokud dojde na rozhodování, které části systému by se (ne)měly automatizovat, je potřeba zvážit několik faktorů. Testeři z Microsoftu doporučili tři hlavní faktory. Je třeba zhodnotit, jak často se to, co se testuje, mění, protože čím méně je předmět testování stabilní, tím více stojí údržba automatizace. Dále je důležité vědět, jak často je prováděno testování, jak je každý výsledek důležitý a jak nákladné je jeho získání. A třetím faktorem je užitečnost automatizace. Otázkou je, zda automatizované testy mají trvající hodnotu v hledání chyb nebo v ověřování důležitých vlastností či funkcionalit softwaru. (Garousi, 2016)

Problém automatizovaných testů spočívá v jejich údržbě. Test je totiž potřeba udržovat, aby odpovídal stavu testovaného systému. Ten se může každou chvíli měnit, a tak test často nebude odpovídat současné podobě systému (Bureš, 2016).

3.2.2 Rozdíly manuálního a automatizovaného testování

Automatizovaný test vykonává test rychleji než člověk a „zdarma“. Test je tak možné opakovat častěji. Kroky provedené automatizovaným testem jsou vždy stejné. Akce, které automatizovaný test provádí jsou závislé na tom, jak jsou naprogramovány. Od daného sledu kroků se neodchýlí a neudělá tak chybu. Proto může být prokazatelnost výsledků vyšší, ale to platí jen v rámci daného scénáře. Automatu chybí intuice, která je naopak výhodou testování manuálního. Pokud má tester při manuálním testování pocit, že by měl zkusit jinou variantu, tak tak učinit může. Kreativitu testera automatem nelze nahradit (Bureš, 2016). Automat neumí sám řešit situace, kdy se chování testovaného systému liší od očekávaného chování. To znamená, že i malá změna prvku v testovaném systému povede k tomu, že test nebude fungovat, jak má. To je cena za to, že automatizace umožňuje testy spouštět rychle a opakovaně. Automatizovanému testu také na rozdíl od člověka chybí přesnější schopnost analyzovat chybné chování systému. Automatizovaný test má přesně dané, co má v dané situaci vyhodnotit, ale pro úplnější analýzu výsledku se musí zapojit sám tester, který daný testovaný systém nebo proces zná. Automatizovaný test, jehož výsledek není dle očekávání, nemusí znamenat, že je chyba v testovaném prostředí. Chyba může totiž být v samotném testu, kdy je test například neaktuální, jak bylo zmíněno výše (Bureš, 2016).

Ve chvíli vytváření automatizovaného testu je již prováděno samotné testování. Protože při tvorbě automatizovaného testu, je nutné testovaný systém manuálně procházet, tak je pravděpodobné, že případný defekt bude odhalen již ve fázi vytváření testu. Proto jsou

automatizované testy účinné především v oblasti kontroly regrese – kdy je kontrolováno, že se v dané oblasti, která již byla bez defektu, defekt znovu neobjevil. (Bureš, 2016)

Pro automatizované testy je nezbytné, aby byl testovaný systém již zprovozněný a stabilní. Při manuálním testování toto podmínkou není a často se testování provádí ve fázi, kdy systém ještě není vytvořený. Další rozdíl je i v kvalifikaci testera. Některé znalosti a dovednosti potřebují manuální tester a vývojář automatizovaných testů stejné. Ale vývojář automatizovaných testů potřebuje ještě navrch umět programovat skripty automatizovaných testů, což znamená mimo jiné znalost programovacích jazyků. A potřebuje ke své práci také znát nástroje a metody z dané oblasti (Bureš, 2016).

4 Nástroje automatizovaného testování

Nástrojů automatizovaného testování je mnoho. Mezi nimi se najdou nástroje, které jsou ideální pro práci s webovými aplikacemi, a právě těmi se bude práce dále zabývat. Zajímavostí je, že všechny použité nástroje, jsou open-source, čili jsou volně ke stažení na internetu. Tento fakt, však nemá vliv na jejich efektivitu a lze s nimi dosáhnout optimálních výsledků.

4.1 Selenium

Hlavním nástroje automatizovaného testování webových aplikací je Selenium. Selenium je sada několika různých softwarových nástrojů. Každý z nich má k podpoře automatizovaného testování jiný přínos. V rámci projektu Selenium se nacházejí následující nástroje:

- **Selenium 2 – Selenium WebDriver**, kterému bude věnována pozornost dále v textu.
- **Selenium 1** – **Selenium Remote Control (RC)**, což je předchůdce Selenium 2, které již není nadále aktivně podporován.
- **Selenium Integrated Development Environment (IDE)**, což je nástroj pro vytváření testovacích skriptů. Je to forma Firefox a Chrome pluginu (přídavná část softwaru, která rozšiřuje jeho funkčnost), který nabízí rozhraní pro psaní automatizovaných testů.
- **Selenium Grid** – umožňuje paralelní běh testů, to znamená, že různé testy, mohou být pouštěny na různých vzdálených zařízeních. Má to dvě výhody. Za prvé, pokud je potřeba spustit velké testové sestavy nebo sestavy, které běží pomalu, je možné značně zvýšit jejich výkon použitím Selenium Grid, které rozdělí testovou sestavu tak, aby testy běžely v jeden čas, ale na více zařízeních. Dále pokud je nutné spustit test ve více prostředích, je možné uskutečnit to pomocí Selenium Grid v různých zařízeních v jeden čas. V obou případech Selenium Grid zkracuje čas, který je potřeba k běhu testů pomocí paralelního zpracování.

Selenium WebDriver – byl vytvořen pro lepší podporu dynamických webových stránek, kde se elementy stránky mohou měnit i bez toho, aniž by se celá stránka znovu načetla (za použití technologie vývoje interaktivních webových aplikací AJAX, což je zkratka pro Asynchronous Javascript and XML). Cílem WebDriveru je tedy poskytnout kvalitně navržené a objektově orientované rozhraní pro programování aplikací (API), které poskytne lepší podporu pro moderní a pokročilé problémy vyskytující se při testování webových aplikací (Seleniumhq.org, 2019).

Selenium WebDriver podporuje několik webových prohlížečů a také mnoho jazyků pro psaní testů. Selenium WebDriver nabízí několik druhů lokátorů pro vyhledávání jednotlivých elementů webové stránky. Elementy webové stránky mohou být lokalizovány pomocí jejich id, třídy, jména, textu odkazu a mnoha dalších (Gojare, 2015).

Další důležitý nástroj je WebDriver, což je název klíčového rozhraní, pro které je test napsán, ale jsou tu i další možné implementace, jako například ChromeDriver. Ten je vyvíjen a podporován projektem Chromium. WebDriver pracuje s prohlížečem Chrome skrze ChromeDriver Binary (který je možné nalézt na stránkách Chromia). Aby vše fungovalo, je potřeba mít nainstalovou některou verzi ChromeDriveru a odpovídající verzi prohlížeče Chrome. ChromeDriver musí být umístěn někde v systému uživatele, aby jej byl WebDriver schopen automaticky nalézt. Prohlížeč Chrome je následně ChromeDriverem nalezen automaticky ve výchozí instalační složce. Pro použití instance driveru jako ChromeDriveru se používá příkaz: WebDriver driver = new ChromeDriver();. Jako ilustrační příklad je uveden ChromeDriver, který je použitý i v rámci práce. Ale zrovna tak může být využit i Firefox driver, Internet explorer driver, atd. (Seleniumhq.org, 2019).

4.2 TestNG

Nicméně i přes mnohé výhody nástroje Selenium WebDriver, má i tento nástroj při testování webových aplikací svá omezení. Selenium WebDriver v sobě například nemá zabudovanou funkcionalitu pro generování snímků obrazovky pro případy selhání testů. Také neobsahuje funkcionalitu pro generování výsledků testu. Tyto a další limitace mohou být odstraněny použitím frameworku (softwarová struktura, která obsahuje různé podpůrné programy pro vývoj informačních systémů) TestNG (Testng.org, 2019).

TestNG je testovací Framework navržený k vyhovění velkému množství testovacích potřeb, od unit testů (což jsou testy, kdy se testuje jediná třída v izolaci od ostatních) až po integrační testy (při integračních testech dochází k testování celých systému, které se skládají z několika tříd, balíčků, atd.) a mnoho dalších druhů testů. TestNG je názorně použit v rámci praktické části práce (kapitola 7). TestNG lze do vývojového prostředí Eclipse nainstalovat jako plugin. V testu se provádí konfigurace TestNG pomocí nejrůznějších anotací @BeforeXXX a @AfterXXX, které umožňují provést kus kódu buď před, anebo po určeném bodě průběhu testu. To nalezne své využití v případě, že je v testu potřeba například nejdříve provést určitou

inicializační metodu (například načtení parametrů) nebo ukončení driveru po konci testu. Frameworků pro podporu automatizovaného testování je více. Jsou to například JUnit a Mockito. TestNG byl vybrán, protože je v rámci práce použit a oproti JUnit nabízí například parametrizaci, kterou JUnit nenabízí. Označuje se anotací @Parameters. Anotace je potřeba do testu importovat pomocí „import org.testng.annotations.*;“.

Na ukázku je zde tabulka s přehledem některých anotací, které jsou v rámci TestNG k dispozici. Jednotlivé anotace jsou vysvětleny níže.

Tabulka – Anotace TestNG (zdroj: https://testng.org)

`@BeforeSuite`
`@AfterSuite`
`@BeforeTest`
`@AfterTest`
`@BeforeGroups`
`@AfterGroups`
`@BeforeClass`
`@AfterClass`
`@BeforeMethod`
`@AfterMethod`
	Configuration information for a TestNG class:
	
	@BeforeSuite: The annotated method will be run before all tests in this suite have run.
	@AfterSuite: The annotated method will be run after all tests in this suite have run.
	@BeforeTest: The annotated method will be run before any test method belonging to the classes inside the <test> tag is run.
	@AfterTest: The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run.
	@BeforeGroups: The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.
	@AfterGroups: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.
	@BeforeClass: The annotated method will be run before the first test method in the current class is invoked.
	@AfterClass: The annotated method will be run after all the test methods in the current class have been run.
	@BeforeMethod: The annotated method will be run before each test method.
	@AfterMethod: The annotated method will be run after each test method.

`@Test`	**Marks a class or a method as part of the test.**

Každá anotace dělá více méně to, co má ve svém názvu. Každá anotace má verzi „Before“ a „After“, neboli před a potom. To naznačuje, kdy se metoda s konkrétní anotací provede. Vždy bude uveden jen jeden příklad z dané dvojice, protože druhý vykonává to stejné, jen v opačném pořadí, co se spuštění anotované metody týče. Ve stručnosti tedy dělají anotace toto:

- **@AfterSuite** – Takto anotovaná metoda bude spuštěna až poté, co doběhnou všechny testy v dané testové sadě.
- **@AfterTest** – Anotovaná metoda bude spuštěna poté, co doběhnou všechny třídy, které jsou uvnitř tagu (značka ovlivňující vlastnosti a smysl „otagovaného“ textu) <test>.
- **@AfterGroups** – V tomto případě metoda proběhne ihned poté, co doběhnou všechny testy z dané skupiny.

- **@AfterClass** – Anotovaná metoda se provede potom, co doběhnou všechny testové metody v příslušné třídě.
- **@AfterMethod** – Tato anotace zajistí, že metoda se spustí po každé provedené metodě v průběhu testu.
- **@Test** – anotace Test označuje třídu nebo metodu jako část testu. V rámci anotace @Test se nacházejí ještě „podanotace“, které se k anotaci @Test váží. Je jich opět mnoho, ale pro příklad bude uvedeno několik nejpoužívanějších. Jednou z nich je například „dependsOnMethods“, což říká, že tento test závisí na proběhnutí vybraných metod. Pokud by dané metody neproběhly, neproběhne ani tento test. Dále také „priority“, neboli priorita, která udává, v jakém pořadí testy proběhnou. Test s nejnižší prioritou bude puštěn jako první (to znamená, že test s prioritou 1 bude proveden jako první). Pro stručný popis toho, co daný test dělá, lze využít „description“ (Testng.org, 2019).

Pro snadnější představu konkrétního využití anotací z praktické části práce je zde obrázek č. 1. Jedná se o anotaci @AfterClass a anotovanou metodou je metoda endOfTest(). Metoda samotná pouze obstarává ukončení driveru v případě, že ještě ukončen není. Ale anotace @AfterClass způsobí to, že se tato metoda, ať bude napsána kdekoliv v rámci testu, provede až po ukončení všech testů v příslušné třídě.

```
600     //Ukonceni testu
601     @AfterClass
602     public void endOfTest() throws InterruptedException{
603
604         if(!driver.toString().contains("null")){
605             Thread.sleep(2000);
606             driver.quit();
607         }
608
609         LOGGER.info("Konec testu!");
610
611     }
612
```

Obrázek č. 1 – TestNG anotace AfterClass (zdroj: autor)

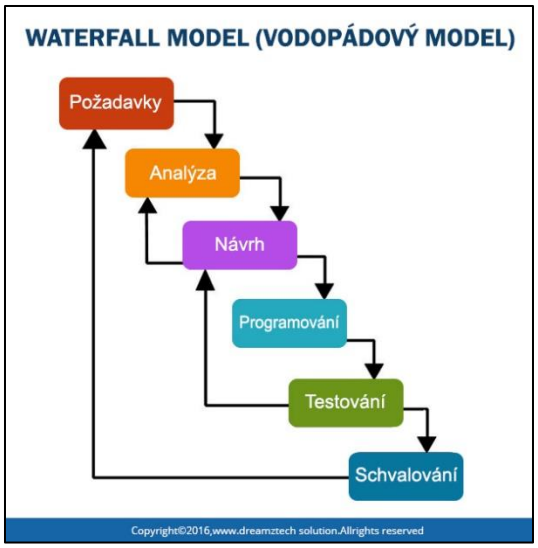
Dalším nástrojem, který sice není přímo nástrojem pro automatizaci testů, ale je tu zmíněn, protože je v této práci pro psaní automatického testu použit, je vývojové prostředí Eclipse. Eclipse byl zvolen z důvodu toho, že je test psán především v Javě a Eclipse je vývojové prostředí, které je pro programování v jazyce Java určeno. Hlavní výhodou vývojového

prostředí Eclipse je jeho rozšiřitelnost pomocí pluginů. Těmi je možné Eclipse rozšířit o podporu dalších programovacích jazyků nebo například přidat nástroj pro návrh grafických uživatelských rozhraní desktopových aplikací, ale je vhodné právě i v případě implementace pluginu TestNG.

5 Správa defektů při vývoji informačních systémů

Softwarový produkt je výsledek spolupráce nejrůznějších profesionálů – analytiků, systémových architektů a samozřejmě programátorů, kteří stejně jako ostatní lidé mohou a také budou při své práci chybovat. Následkem jejich chybování je zanesení defektů do kódu či jiných artefaktů procesu softwarového vývoje, které způsobují selhání (Roudenský, 2013).

Na obrázku č. 2 lze proces softwarového vývoje vidět. Jedná se o schéma vodopádového modelu, který popisuje přístup k vývoji softwaru jako sekvenční (navazující) posloupnost jednotlivých fází. První fází jsou požadavky na vyvíjený systém. Které by měly udávat, co má systém umět a k čemu má sloužit. Ve druhé fázi probíhá analýza požadavků. Ta se provádí za pomocí navrhování modelů toho, jak bude systém fungovat. Další fází je návrh, který má za cíl vytvoření návrhu architektury systému. V této fázi se řeší komponenty, algoritmy a vše se připravuje pro programování. Čtvrtou fází je programování a jak název napovídá, jde o proces programování a vytváření spustitelného počítačového programu. Dalším a pátým krokem je testování. Což je proces hledání a opravování chyb v systému a ověřování, zda je systému připraven k ostrému provozu. A nakonec musí systém projít procesem schvalování, kde se potvrdí, zda má systém požadované funkcionality.



Obrázek č. 2 – Vodopádový model (upraveno podle DREAMZTECH SOLUTIONS PVT.LTD. (2016) uvedeno v Quizlet (2019))

Jednou ze základních vlastností charakterizujících výpočetní systémy je tzv. dependabilita neboli schopnost výpočetního systému dodávat službu, které lze oprávněně důvěřovat. Jinak ji lze také definovat jako schopnost systému vyhnout se selháním služby, která

jsou častější a závažnější, než je pro uživatele přijatelné (Avižienis, 2004). Pod pojem dependabilita lze pro představu zařadit zvýšená potřeba spolehlivosti, bezpečnosti, dostupnosti a udržovatelnosti výpočetních systémů v důsledku stále se zvyšující závislosti lidí na výpočetních systémech (Roudenský, 2013).

Správa defektů se v oblasti testování velice často podceňuje. Chybování v této oblasti mohou ovšem vést k celé řadě dalších problémů. Laxní přístup k opravě defektů či jejich analýze může vést například ke zpožděné dodávce projektů. Výsledkem nedůsledného či nesystematického využívání nástroje správy defektů může být opomenutí některých defektů a jejich znovuobjevení v produkčním systému. Nedostatečnost nebo dokonce absence analytického vyhodnocování příčin a charakteru defektů v závěru testovacího projektu (z časových důvodů nebo z důvodu neevidování důležitých dat) znemožňuje projektovému týmu ponaučení z vlastních chyb a zlepšení slabých míst (Bureš, 2016).

Často bývají pojmy chyba, defekt, selhání a incident zaměňovány. Chybu udělá člověk (vývojář, analytik) při práci na své části projektu (např. kus kódu). Defekt nebo také „bug, je odchylka aktuálního chování systému od toho očekávaného. Selháním je označováno selhání systému, které nastává v důsledku defektu. Při něm systém nebo jeho část obvykle havaruje. Incident je termín používaný pro označení vady nalezené v produkčním prostředí. Incidenty nalezené v produkčním prostředí, jsou více sledovány a oprava je vždy nákladnější, než kdyby byly nalezeny v dřívějších fázích projektu (Bureš, 2016).

S defekty přijdou v procesu vývoje do styku lidé v mnoha rolích. Tester je ten, kdo defekt najde a zadá ho do testovacího nástroje. Je také prvním, kdo posuzuje závažnost zjištěného defektu. Analytik defektů následně ověří, že se jedná o relevantní defekt, vyloučí duplicitní defekty a vrátí nedostatečně či nejasně zadané defekty zpět autorovi k doplnění. Ověřené, relevantní defekty poté přiřadí odpovědným vývojářům nebo dodavatelům k opravě. Manažer testování zastává dohled nad správou defektů a v případě nefungující komunikace s vývojáři či dodavateli je také eskalační autoritou (což je autorita o úroveň výš, tedy někdo nadřízený) pro analytika defektů. Dále je jeho úkolem podávání informací o průběhu testování příslušným, zainteresovaným osobám. Vývojářova práce spočívá v lokalizaci a opravě defektů a podávání informací o opravách. Činnost vývojáře může poskytnout vstupy pro regresní testy (indikace části kódu, které byly dotčeny opravou defektu). Projektový manažer stanovuje

priority defektů. Je také eskalační autoritou pro manažera testování v případě řešení problému s opravami atd. (Bureš, 2016).

6 Webové aplikace

Obsah webové stránky je vytvořen pomocí HyperText Markup Language (HTML neboli standardizovaný jazyk pro psaní webových stránek a aplikací). HTML5 se odvíjí od HTML a obsahuje nové atributy a chování. Kromě HTML5 jsou stavebními bloky pro většinu moderních prohlížečových aplikací také JavaScript (psán zkratkou JS je programovací jazyk a spolu s HTML a CSS je jednou ze základních technologií pro web) a Cascading Style Sheets 3 (CSS3 je technologie, která umožňuje oddělit formátování obsahu stránky od obsahu samotného) (Farrukh Shahzada, 2017).

„Jednostránkové“ aplikace (Single-Page Applications) jsou webové aplikace, které načtou jednu stránku HTML a dynamicky aktualizují její obsah na základě interakce uživatele s aplikací pomocí menu a jiných navigačních prvků. Tyto aplikace jsou uživatelsky více přívětivé. Plynulé a responsivní (neboli korektní zobrazení, bez ohledu na zařízení, ve kterém je prohlížena) webové aplikace jsou vytvářeny bez neustálých znovunačítání stránky, za použití AJAX technologie, která komunikuje se skripty na straně serveru k přijímání a odesílání informací v nejrůznějších formátech (většinou z nebo do trvalého úložiště jako jsou databáze) (Farrukh Shahzada, 2017).

Ideálně se žádná data nečtou z Document Object Model (což je ve zkratce rozhraní pro programování aplikací, zkratka DOM), ale aplikace má jako výstup HTML a provádí operace s elementy dle potřeby. Zobrazení obdrží hlášení o změně (skrze události) od modelů a vhodně zařídí vykreslení změny. Model také čte a zapisuje z a do úložiště (většinou databáze) za použití AJAXu a skriptů na straně serveru (Farrukh Shahzada, 2017).

Díky poměrně komplexnímu vztahu mezi HTML, CSS a JavaScriptem, je layout (rozvržení ve smyslu toho, jak se webová aplikace zobrazuje) webových aplikací náročnější na správné vykreslení oproti klasickým počítačovým aplikacím. Jeden dokument totiž může být zobrazen v několika velikostech, rozlišeních, prohlížečích a i zařízeních, což činí přítomnost tzv. layout „bugů“ převažující. Takové problémy mohou být od relativně nezásadních jako například překrývající se, špatně umístěné elementy až po zásadnější, které omezují funkčnost uživatelského rozhraní. Tento problém je zhoršován nedostatkem testovacích nástrojů webových aplikací (Hallé, 2016).

7 Návrh automatizovaného testu

Práce zpracovává průběh a výstup automatizovaného testu webové aplikace. Smyslem testu je ověřit, že se po založení záznamu nového cestovního příkazu, bude záznam nacházet v seznamu mezi ostatními záznamy, a to ve správném stavu. K tomu, aby se tak stalo, je potřeba provést posloupnost mnoha kroků, a protože je postup zakládání záznamu až na hodnoty jednotlivých parametrů téměř vždy identický, je možné tento proces zautomatizovat. K automatizaci je využita řada nástrojů, které byly jsou popsány v kapitole 3. Mezi zásadní patří funkcionalita Selenium Grid, která umožní paralelní běh testu, běh testu v různých prostředích a v různých prohlížečích.

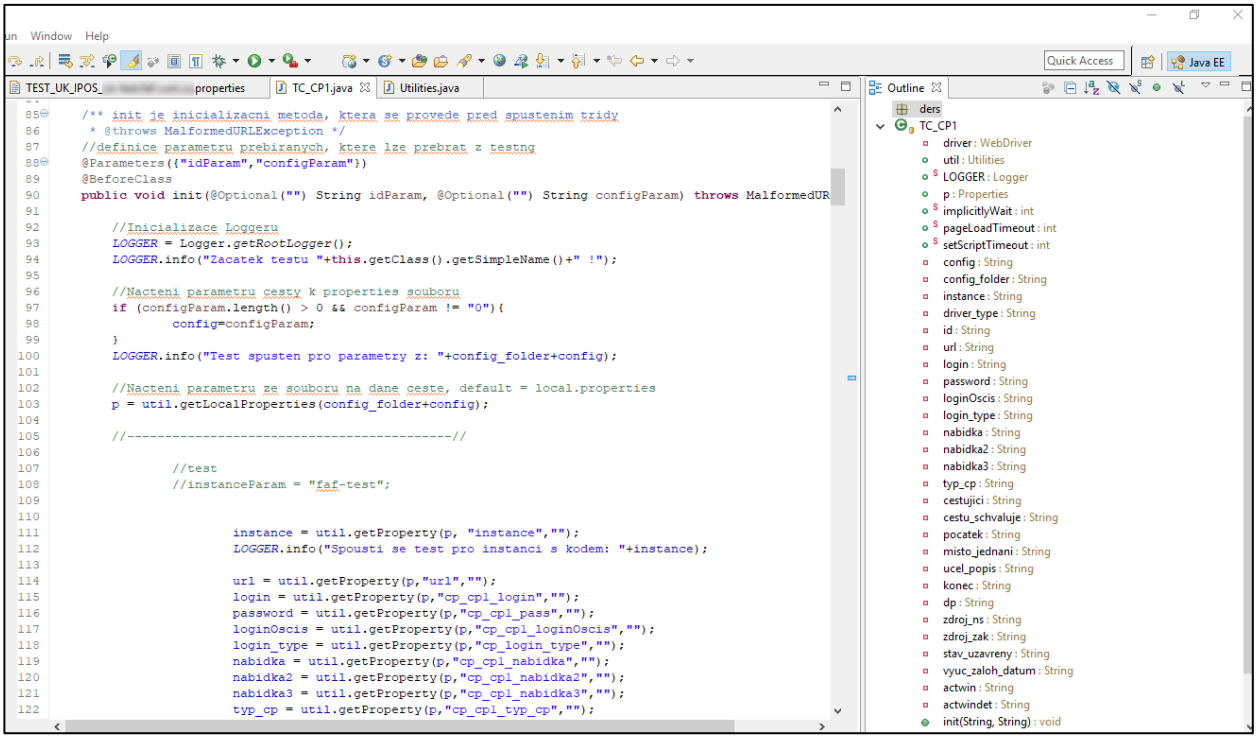
Pro přehlednost a potřeby bakalářské práce, je test spuštěn lokálně v prostředí Windows a ve vývojovém prostředí Eclipse. Test je psán jazykem Java v kombinaci s JavaScriptem, Xpathem a Selenium WebDriverem. Selenium WebDriver nabízí různé možnosti lokalizace HTML elementů v rámci webové aplikace. Xpath zajišťuje zápis „cesty“ k hledaným HTML elementům.

V průběhu celého testu je často využívána třída Utilities. V třída Utilities jsou vytvořeny užitečné metody, které se obecně v testech mnohdy a opakovaně využívají a jejich struktura je pro všechny testy stejná nebo je testům přizpůsobená například pomocí podmínek. Z těch nejčastějších lze uvést waitForAjaxLoad(), která zprostředkovává čekání na načtení AJAXu na pozadí v rámci webové aplikace. Dále metody switchToFrame(), existsElement() a jiné. Jednotlivé metody jsou v průběhu testu popsány.

7.1 Přednastavení automatizovaného testu

Pro úspěšný start testu je potřeba načíst parametry z parametrizačního souboru. Dle instance a modulu, na kterých test poběží, se vybere i příslušný parametrizační soubor. Podle parametrů uvedených v daném parametrizačním souboru se parametry v testu inicializují jako proměnné s adekvátním datovým typem. Vzhledem k charakteru testu, a to sice, že úkolem uživatele je většinou vyplňovat textová pole, se převážně pracuje s datovým typem String, který parametrům umožňuje uchovávat v sobě text. Následně se jednotlivé hodnoty parametrů, tak, jak jsou uvedeny v parametrizačním souboru, přiřadí parametrům inicializovaným v testu. Jednotlivé parametry v testu lze vidět v příloze č. 1.

Na obrázku č. 3 lze vidět načítání parametrů z parametrizačního souboru do parametrů v testu.



Obrázek č. 3 – Naplnění parametrů (zdroj: autor)

Zde je tedy potřeba zmínit i samotný parametrizační soubor. Parametrizační soubor obsahuje parametry pro všechny moduly dané instance. To znamená, že pro instanci označovanou faf-test, je k dispozici parametrizační soubor, který obsahuje parametry, jak pro test cestovních příkazů, tak dalších modulů. Modulem jsou v tomto případě tedy testované Cestovní příkazy.

Zkratky instancí jsou zpravidla odvozené od klientů, pro které se aplikace vyvíjí. Zkratka před pomlčkou je zkratka klienta a část za pomlčkou označuje prostředí, na kterém se testování provádí (testovací, předprodukční, produkční). Každá instance má také svou příslušnou URL adresu.

Parametrizační soubor je rozdělen na sekci All a následně na sekci pro jednotlivé moduly. V sekci All se nacházejí takzvané globální parametry, tedy parametry, které jsou dostupné pro test na každý modul. Nejpodstatnější jsou zde parametry instance a URL adresa instance. Ostatní parametry jsou již dostupné jen konkrétním modulům.

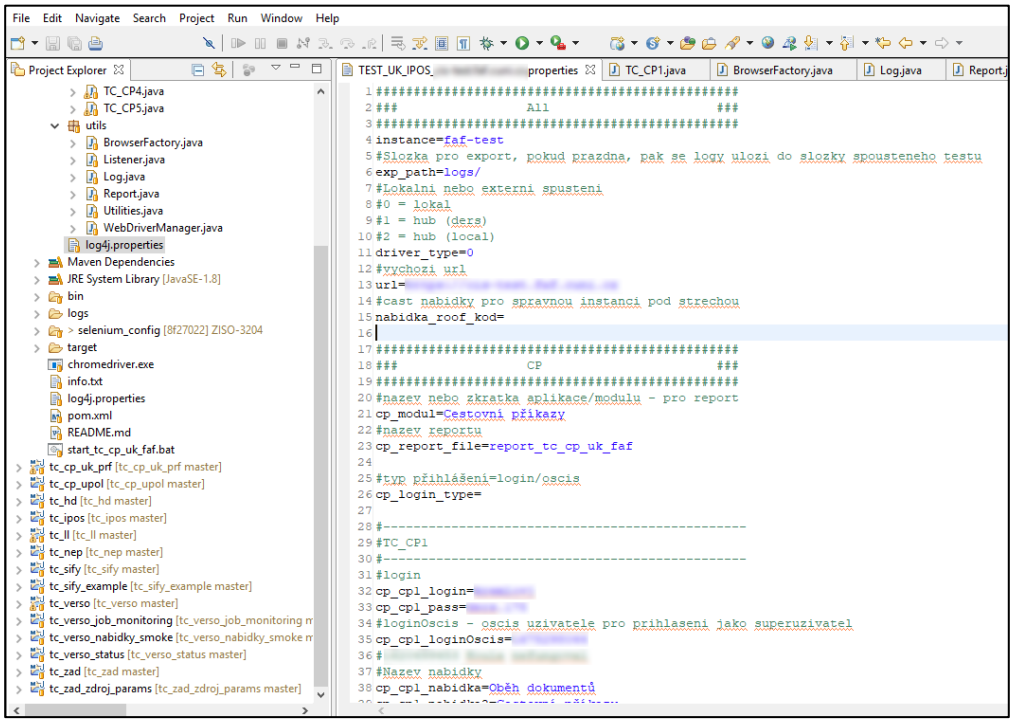
Další sekce je sekce s parametry pro příslušný modul. Ty jsou ještě rozděleny podle jednotlivých tříd. Pro jeden modul totiž existuje několik tříd s testy, které testují sice stejný

modul, ale vždy trochu jinak. Existují například třídy TC_CP1.java, TC_CP2.java, atd. Parametry pro test TC_CP1 budou mít předponu cp_cp1_parametr. Pro TC_CP2 budou parametry začínat cp_cp2_parametr. V rámci modulu jsou parametry opět ještě rozdělené na globální, které ale platí jen v rámci daného modulu, avšak pro všechny třídy testu. Následně má každá třída své parametry. Zde se parametry liší většinou již pouze svými hodnotami, které mohou být pro každou třídu odlišné.

Výhodou využití parametrizačních souborů odděleně od samotných testových tříd je přehlednost a usnadnění práce s parametry. Pokud se totiž například změní přihlašovací údaje pro daný test, tak stačí parametr změnit v parametrizačním souboru a změna se projeví ve všech testech, které daný parametrizační soubor používají. Odpadá tím tedy nutnost parametr měnit v každém testu individuálně.

To, který parametrizační soubor bude test využívat, udává parametr config. Do tohoto parametru se vyplňuje hodnota v podobě názvu parametrizačního souboru. K tomu, aby test daný parametrizační soubor nalezl, slouží zase parametr config_folder. Který udává, v jaké složce se parametrizační soubor nachází.

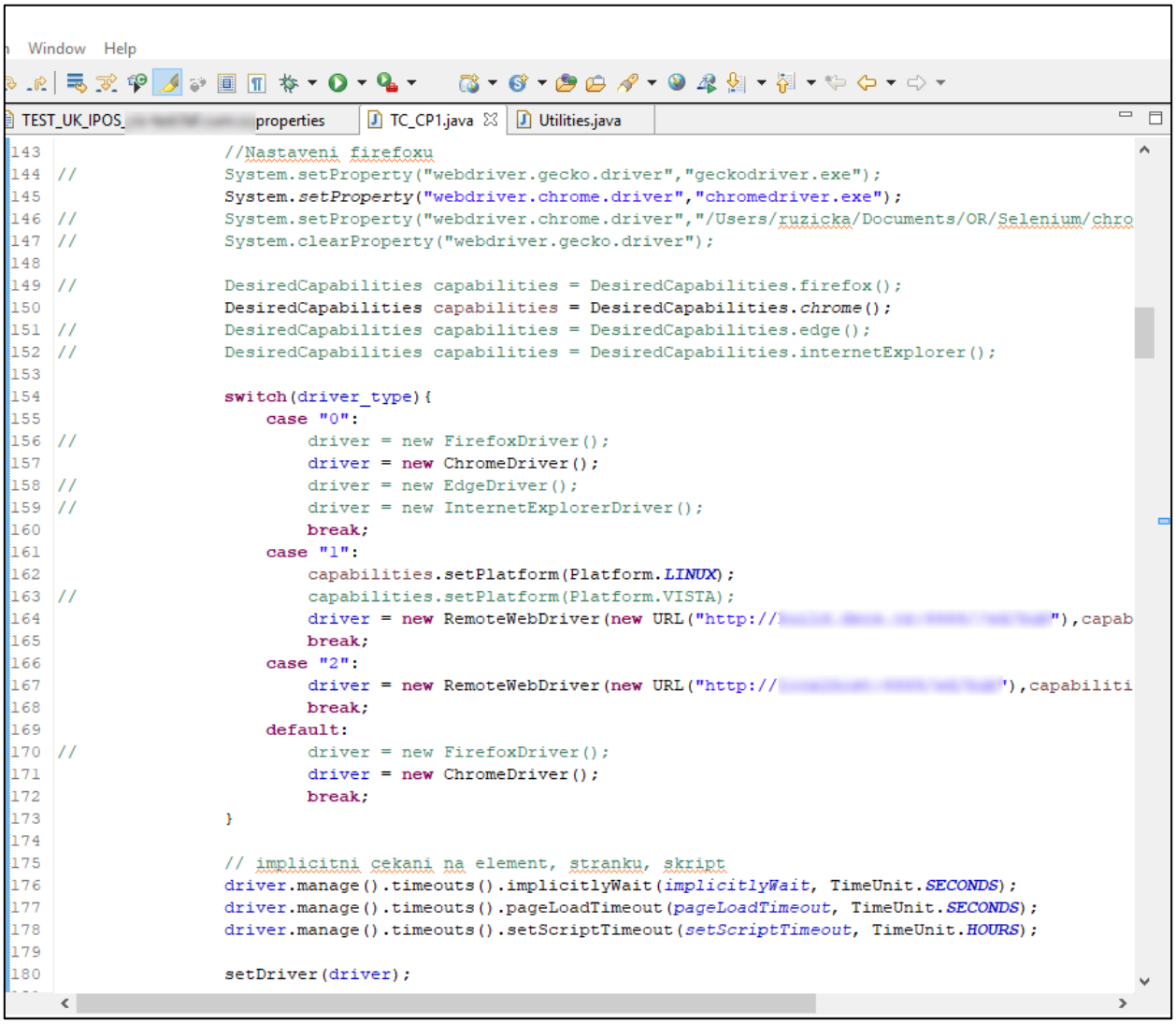
Na obrázku č. 4 je vidět parametrizační soubor, kde jsou parametry globální a parametry pro modul cestovní příkazy.



Obrázek č. 4 – Parametrizační soubor (zdroj: autor)

Na obrázku č. 5 je zobrazeno nastavení a výběr prohlížeče i prostředí, které zajišťuje zmíněná funkcionalita Selenium Grid, která byla přidána ve verzi Selenium 2.0. Pro její samotné zprovoznění je potřeba mít stažené knihovny selenium-server-standalone.jar, které bude test využívat. K definování toho, který prohlížeč, prostředí a verze bude použita, je nutné použít takzvané DesiredCapablities, které se následně předají objektu RemoteWebDriveru URL s adresou hubu (hlavní uzel nebo také rozbočovač počítačové sítě, který umí příchozí data z jednoho portu, zkopírovat i na všechny ostatní porty, které jsou k hubu připojeny).

Selenium Grid nalezne své využití v případě, že se test pouští na vzdáleném serveru, kde je přístupný všem oprávněným uživatelům. A také v případě, že je potřeba nasimulovat běh na rozdílných platformách či prohlížečích.



Obrázek č. 5 – Selenium Grid (zdroj: autor)

Dále proběhne výběr driveru. Typ driveru, který bude zvolen je dán hodnotou parametru driver_type, který se předává metodě login. Drivery jsou rozdělené podle druhů prohlížečů. Pro tento test je primárně využit ChromeDriver.

7.2 Průběh automatizovaného testu

Pro přehlednost je průběh testu popsán v několika krocích. Každý krok začíná přechodem na následující stránku a končí před přechodem na další. V každém kroku jsou také popsány a vysvětleny akce, které se na dané stránce prováděly.

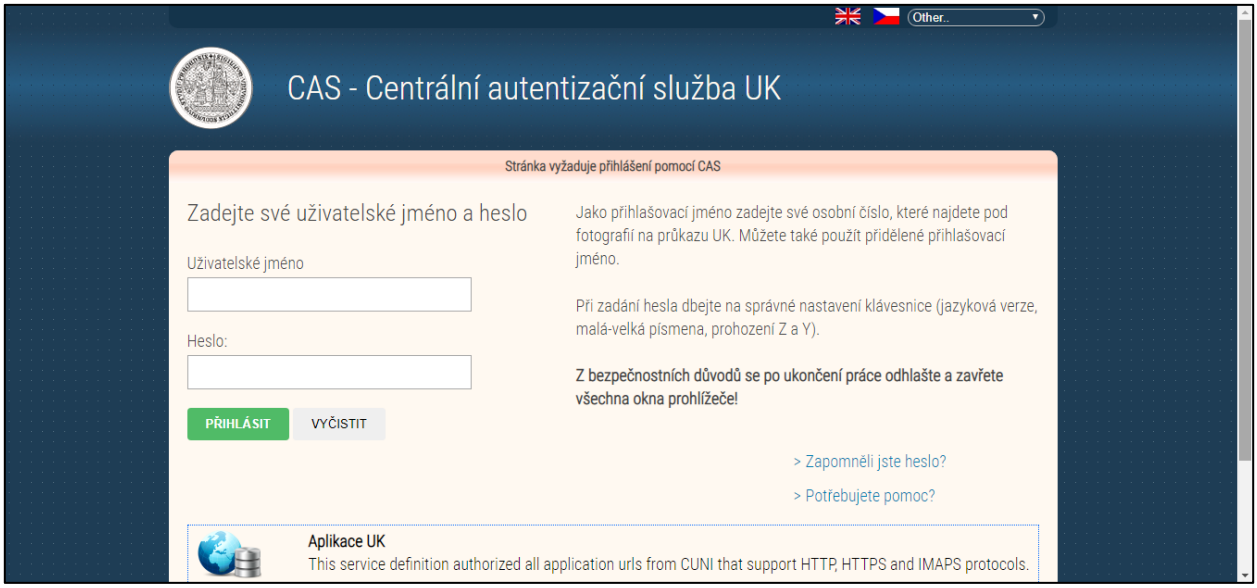
1. Začátek testu je po předání URL adresy instanci webdriveru. Ten spustí nové okno prohlížeče a přejde na danou URL adresu. V kódu je toto ošetřeno příkazem driver.get(URL), který volá metodu get a předává jí parametr URL, ve kterém je v parametrizačním souboru uložená URL adresa, kterou má webdriver načíst. Prohlížeč se spustí a maximalizuje se otevřené okno. Přihlašovací stránka je vidět na obrázku č. 6. Maximalizací otevíraného okna se předchází zbytečným selháním testu, které jsou způsobené nemožností webdriveru nalézt požadované HTML elementy. Zmíněné metody lze vidět v příloze č. 2.

Dále se načte URL adresa a webdriver pomocí zavolání metody login vyplní textová pole uživatelské jméno a heslo hodnotami z načtených parametrů a následně klikne na tlačítko přihlásit. Použije se čekání na načtení AJAXu a test se přepne na další stránku. Vnitřní struktura metody login je vidět na obrázku č. 24.

AJAX neboli Asynchronous JavaScript and XML je technologie, napsána v jazyce JavaScript, která se využívá při vývoji interaktivních webových aplikací. Umožňuje načítání obsahu stránek, bez nutnosti znovunačítání celých stránek samotných. Tento proces funguje za pomoci asynchronního zpracování požadavků mezi klientským prohlížečem a serverem.

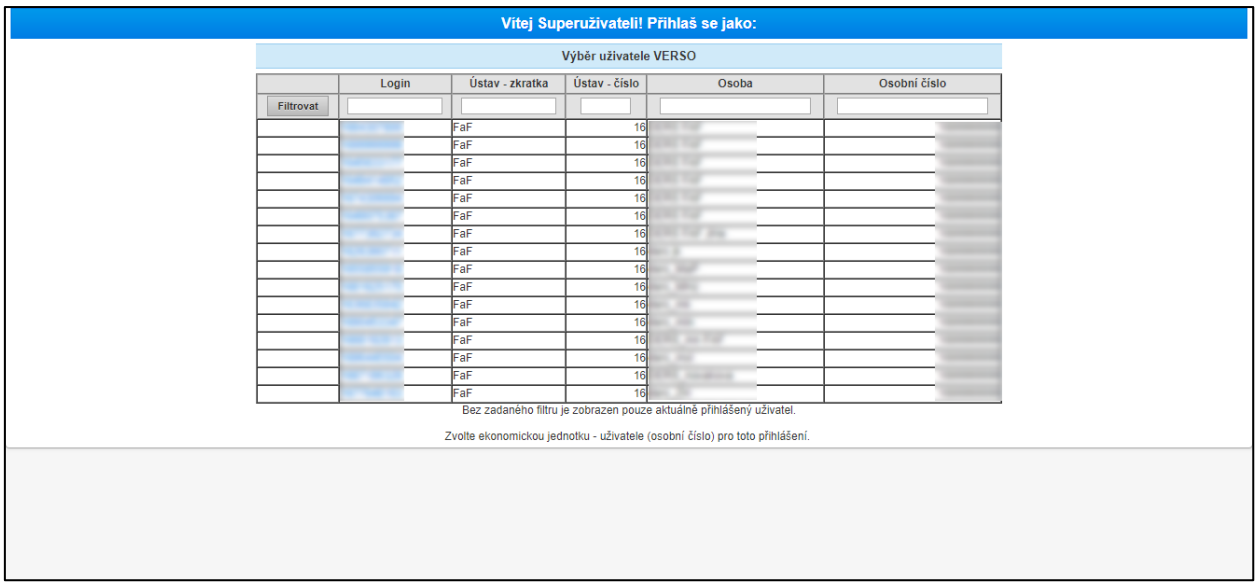
V kódu si lze také povšimnout častého využívání LOGGERu. LOGGER umožňuje do konzole vypsat nějaký textový výstup během testu, a to za použití metody LOGGER.info („text, který se má v konzoli vypsat“);. Přehledně tak test doprovází v jeho průběhu pomocí textu a umožňuje lepší orientaci, jak pro vývojáře, tak uživatele. Usnadňuje tedy přehled o tom, co test v danou chvíli dělá nebo s jakou chybou skončil. LOGGER v sobě totiž často nese i informaci o chybě. V takovém případě se využije metoda LOGGER.error(), kdy je text v konzoli mimo jiné zbarven červeně.

V příloze č. 3 je možné vidět využití metody Thread.sleep(milisekundy). Tato metoda „uspí“ test na požadovanou dobu. Metodě se předává hodnota času v milisekundách. Většinou se toto využívá, pokud je z důvodu například delšího načítání stránky potřeba nějakou dobu počkat. Někdy se totiž stává, že test selže právě při načítání nějaké části webové aplikace, kdy chce test pokračovat dále, ale načítání ještě není dokončeno a potřebuje tím pádem více času.

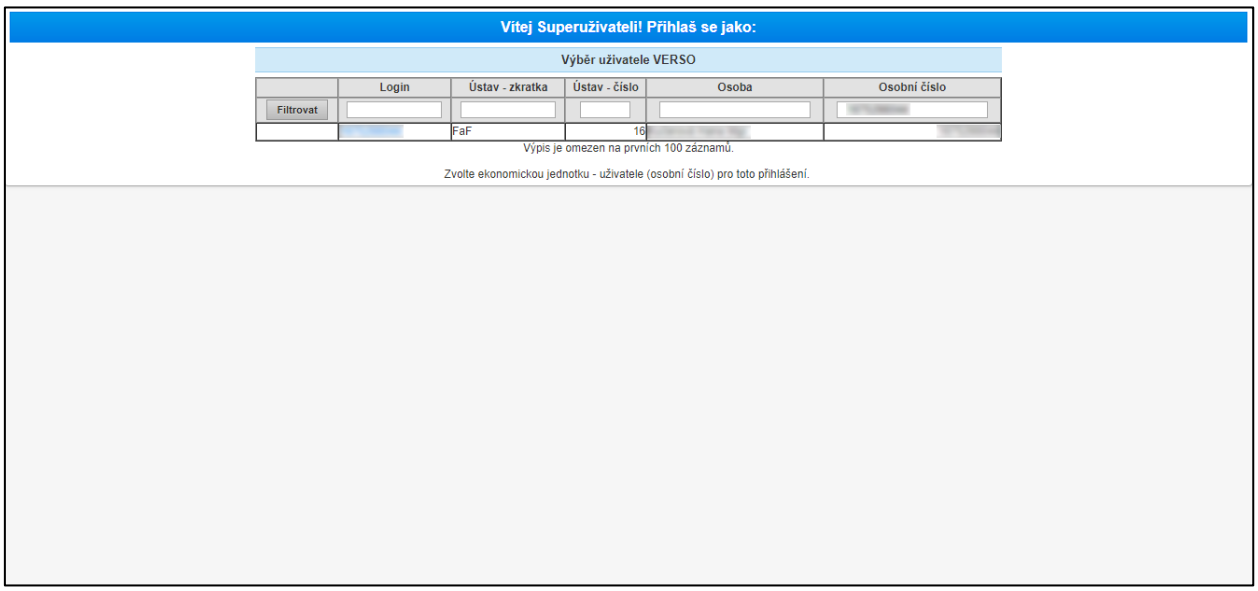


Obrázek č. 6 – Přihlášení (zdroj: autor)

2. Po přihlášení uživatele do aplikace přejde test na stránku se seznamem osob s jejich osobními čísly a dalšími údaji, což je vidět na obrázku č. 7. Pro vyhledání uživatele, za kterého se chce uživatel přihlásit, se metodě login předává parametr loginOscis, který v sobě nese hodnotu právě v podobě osobního čísla, které se vyplní do pole Osobní číslo a následně webdriver klikne na tlačítko Filtrovat. Stav seznamu po vyfiltrování konkrétní osoby je vidět na obrázku č. 8. Po vyfiltrování webdriver klikne na nalezenou osobu a test pokračuje dál. Použití metody login v testu lze vidět v příloze č. 3. Její vnitřní strukturu je možné vidět na obrázku č. 24.



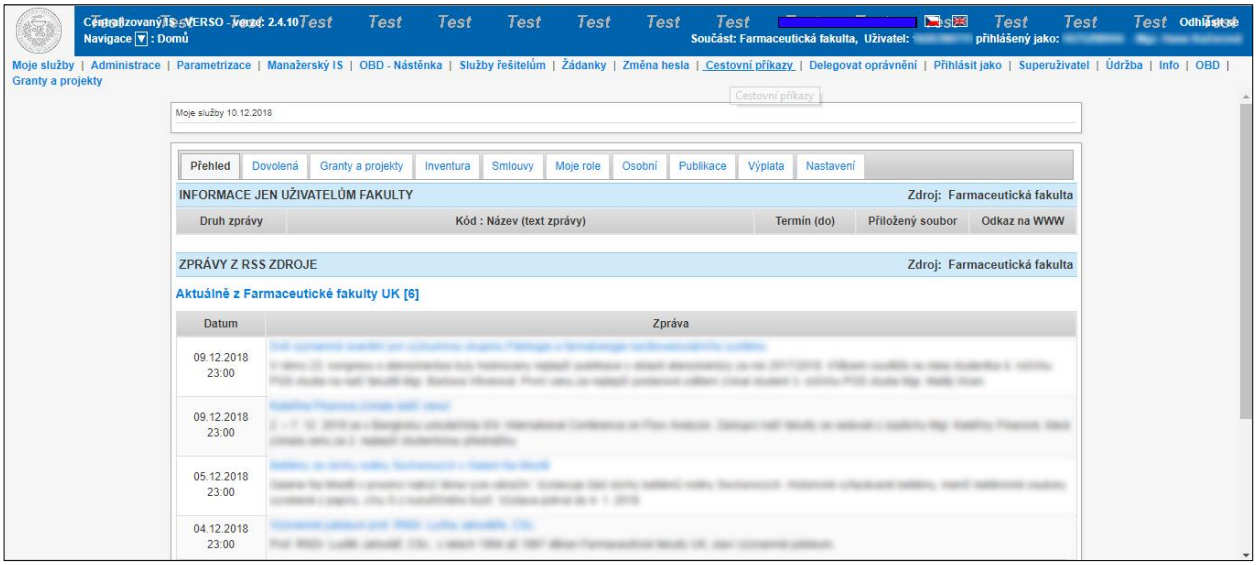
Obrázek č. 7 – Přihlášení superuživatele (zdroj: autor)



Obrázek č. 8 – Přihlášení superuživatele vyhledání uživatele (zdroj: autor)

Přihlašování pomocí osobního čísla se provádí především proto, že jednotlivé osoby mají odlišná práva v rámci dané webové aplikace. Tudíž je pro testování jednotlivých funkcionalit potřeba zvolit adekvátní osobu s potřebnými oprávněními.

3. Nyní se test nachází na domovské stránce instance faf-test, která je vidět na obrázku č. 9. Ze záhlaví je zřejmé, že se vše odehrává v testovacím prostředí. Dále je zde několik různých nabídek s dalšími, pro tuto instanci dostupnými, moduly. Protože se test zabývá testováním cestovních příkazů, webdriver vybere z menu nabídku Cestovní příkazy viz Příloha č. 4.



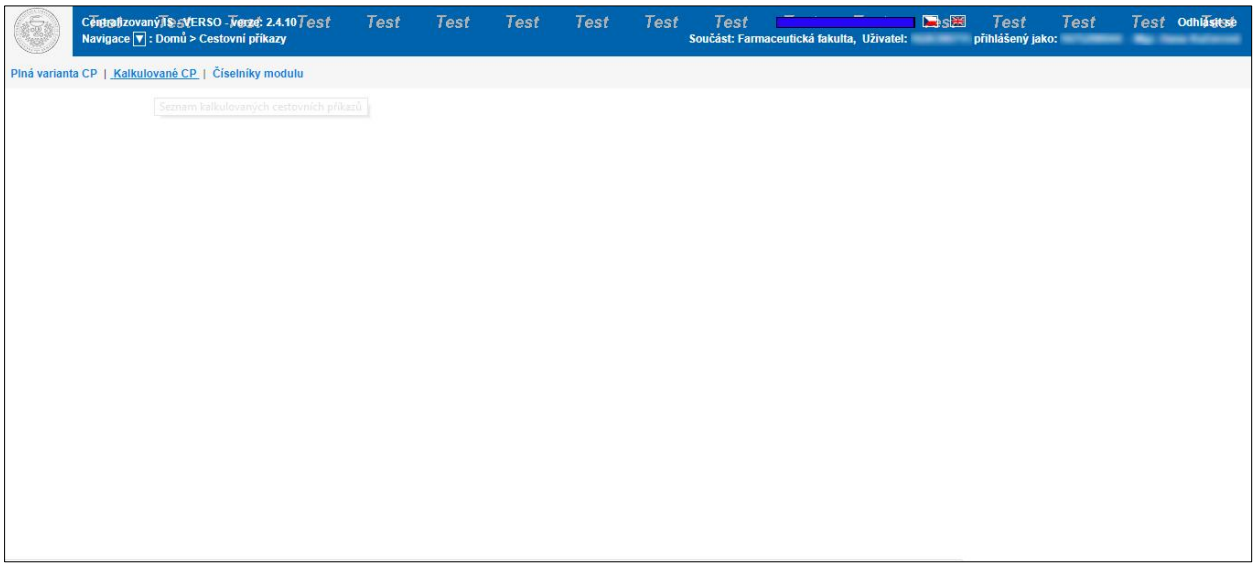
Obrázek č. 7 – Domovská stránka instance (zdroj: autor)

V kódu si lze povšimnout využití metody switchToFrame. Samotná webová aplikace, se totiž skládá ze dvou hlavních framů (výraz „frame“ bude používán v originálu). Frame lze chápat jako celek, který v sobě obsahuje všechny ostatní prvky, kterým je frame nadřazen. V této fázi se jedná o framy záhlaví a text. Pro přístup k elementům framu záhlaví nebo text je potřeba se do daného framu vždy přepnout, což zajišťuje metoda switchToFrame. Té se předávají čtyři parametry – instance webdriveru, název framu, lokátor a parametr, který udává, jakou „cestou“ se do daného framu přepne. To znamená, zda je potřeba se pro přepnutí do daného framu v rámci struktury „vynořit“ a následně se do daného framu přepnout nebo „vnořit“ přímo do framu.

4. Po kliknutí na nabídku Cestovní příkazy, se test dostává k nabídkám, kde je možno zvolit, zda se v konkrétním testu jedná o Plnou variantu Cestovních příkazů nebo Kalkulovanou. Tento konkrétní test testuje kalkulovanou verzi, webdriver tedy vybere nabídku Kalkulované CP. Kalkulovaná verze a Plná verze se liší postupem schvalování cestovního příkazu a také některými údaji, které je potřeba vyplnit.

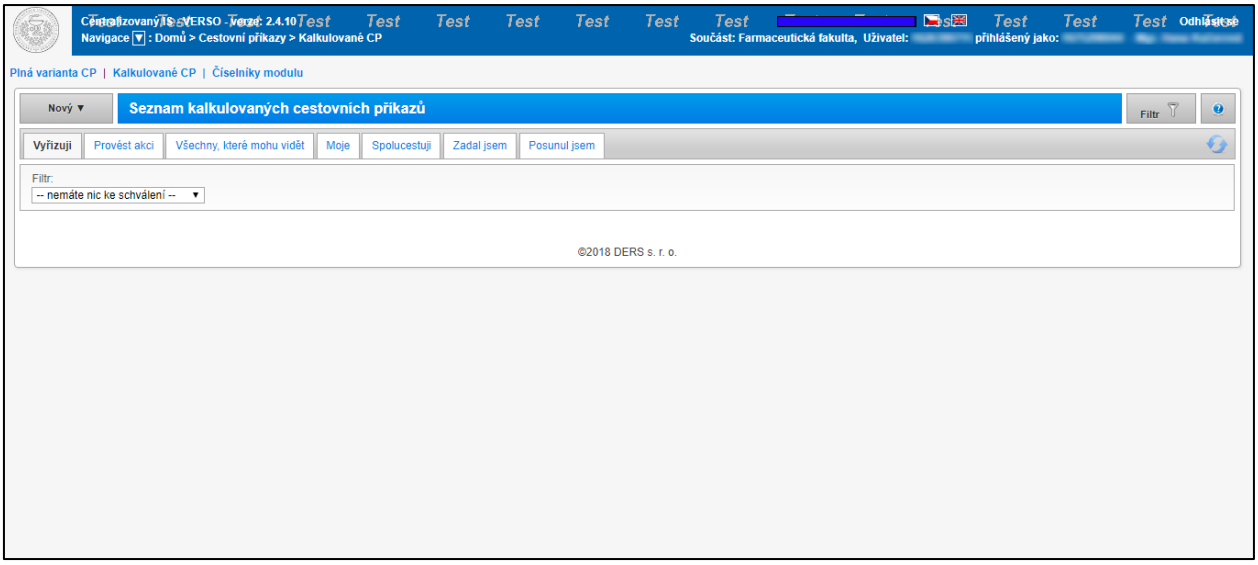
V kódu v příloze č. 5 je vidět, že si test nejprve zkontroluje, zda se již nenachází na seznamu kalkulovaných cestovních příkazů. Na jiných instancích je možné, že by se test již v tuto chvíli na seznamu nacházet mohl. Je to ošetřené podmínkou s využitím vykřičníku před samotným příkazem. Vykřičník znamená opačnou logiku. To znamená, že test ověřuje, zda se na stránce nenachází nadpis, který má v sobě text „Seznam kalkulovaných cestovních příkazů“. V případě, že nenachází, tak se přejde k vyhledávání nabídky Kalkulované CP, kdyby se ovšem

nacházel, tak se tento krok zcela přeskočí. Na obrázku č. 10 je vidět situace, kdy se test na seznamu všech kalkulovaných cestovních příkazů nenachází a musí tedy vybrat nabídku Kalkulované CP.



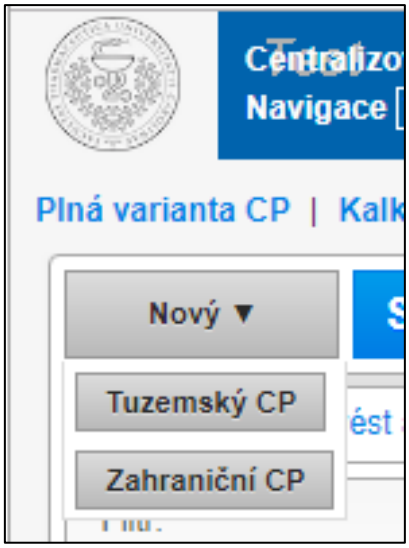
Obrázek č. 8 – Kalkulované cestovní příkazy (zdroj: autor)

5. Po kliknutí na nabídku Kalkulované CP se test dostane na seznam všech kalkulovaných cestovních příkazů, který je i se záznamy vidět na obrázku č. 22. Zde je na výběr založení nového záznamu (příkazu), nebo také zobrazení záznamů, dle několika podmínek. V záložce Všechny, které mohu vidět, jsou všechny záznamy, které jsou pro daného uživatele viditelné. U konkrétního záznamu v seznamu je vidět osoba, která byla pro daný záznam vybrána, místo jednání, počátek a konec cesty, typ dopravního prostředku, stav, kdy byl záznam založen a kód CP. Záznam se může nacházet ve dvou stavech, a to Kalkulace založená a Kalkulace uzavřená. Cílem práce je dostat záznam do stavu Kalkulace uzavřená. Je zde také možnost otevřít filtr a vyhledávat podle mnoha filtrovacích kritérií. Filtr je podrobněji představen dále v textu.



Obrázek č. 9 – Kalkulované cestovní příkazy seznam (zdroj: autor)

6. Test klikne na tlačítko Nový (viz obrázek č. 12), což zobrazí rozbalovací menu s nabídkou Tuzemského a Zahraničního cestovního příkazu viz příloha č. 6. Test vybere možnost Zahraniční CP.



Obrázek č. 10 – Zahraniční / tuzemský cestovní příkaz nový (zdroj: autor)

7. Po kliknutí na možnost Zahraniční CP se otevře okno s detailem nového záznamu, kde je potřeba vyplnit jeho náležitosti. Pole vybarvená červeně jsou pole povinná. Pole vybarvená žlutě jsou pole doporučená, ne však povinná. Záznam se tedy povede založit i bez jejich vyplnění. Na obrázku č. 13, je kromě červeně a žlutě zbarvených textových polí, možné vidět i to, že je záznam ve výchozím stavu, a to Kalkulace založená.

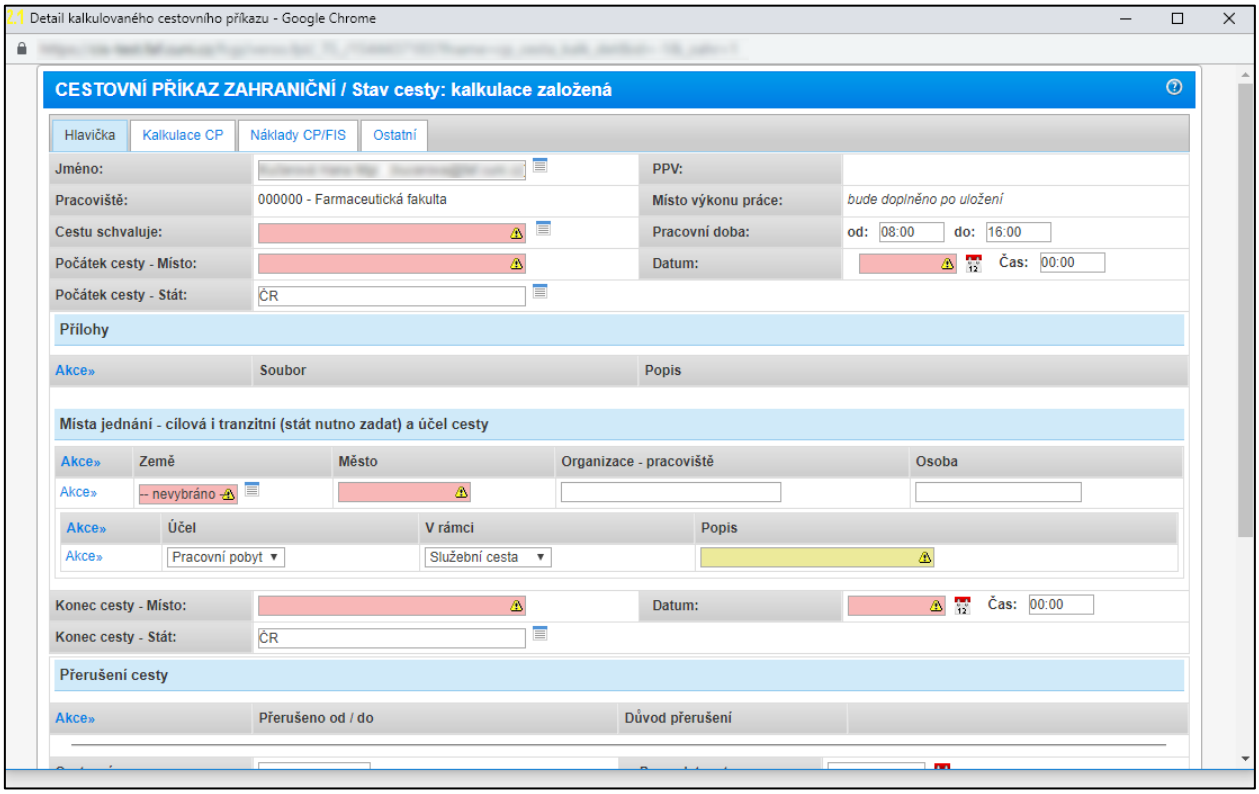
V kódu je zde také vidět příkaz actwin = driver.getWindowHandle(). Ten slouží pro usnadnění následného opětovného přepnutí do aktuálně otevřeného okna. Do parametru actwin je metodou getWindowHandle() uloženo „handle“ aktuálně otevřeného okna pro manipulaci s ním. Tento příkaz se využívá především u oken, do kterých se v průběhu testu webdriver přepíná opakovaně, a tak to celý kód usnadní a zpřehlední.

8. Webdriver začíná na záložce Hlavička. Nyní webdriver pomocí různých lokátorů (text, name, id, class…) nachází jednotlivé HTML elementy a vyplňuje textová pole. Jako příklad lze uvést pole Jméno. Pokud by byl uživatel přihlášen za osobu, u které není pole Jméno předvyplněné, svítilo by toto pole červeně a bylo by nutné ho vyplnit. Tato situace je v kódu ošetřená podmínkou, že se provede vyplnění pole Jméno pouze tehdy, pokud je toto pole zbarvené červeně (viz Příloha č. 7). To také znamená, že na něm bude žlutý trojúhelník s vykřičníkem a HTML element v sobě bude skrývat třídu „chyba“. V případě, že je pole podbarveno žlutě, jedná se pouze o pole doporučené. Pokud ho tedy uživatel nevyplní, bude na to upozorněn, ale dalším krokům to nezamezí.

Je jistě dobré uvést příklad vyhledávání elementů z kódu. Příkaz driver.findElement(By.id(„novy_zaznam“)).click(); zavolá na driveru metodu findElement(). Metodě findElement() je následně řečeno, podle jakého lokátoru má element vyhledávat. V tomto případě id, čili By.id(). V závorce ID je v uvozovkách uvedeno ID, které má u sebe v rámci kódu webové aplikace daný element uveden. A nakonec je webdriveru metodou click() řečeno, že má na daný element po jeho nalezení kliknout. Toto je nejkratší a nejjednodušší případ, kdy se pracuje s lokátorem id, který je pro každý element jednoznačný a unikátní. Tím pádem není potřeba webdriveru podrobněji popisovat cestu, jak se k danému elementu dostat.

Komplikovanější je například příkaz pro otevření číselníku. Ten vypadá následovně driver.findElement(By.xpath(„//table[contains(@class,‘zaznamy‘)]//a[starts-with(@name,‘_master_lov_value_idlide_‘)]“)).click();. Nyní je vyhledávání pomocí Xpath, tzn By.xpath. Xpath se použije, pokud není element jednoznačně identifikovaný a „cesta“ k němu je komplikovanější. Xpath umožňuje složit několik lokátorů za sebou. Tentokrát začíná nalezením tabulky, která obsahuje třídu zaznamy. Takových tabulek může být na stránce několik, proto je potřeba dále upřesňovat. Dvě lomítka znamenají, že se následující element, tedy odkaz, nachází někde dále ve struktuře, ale ne hned pod tabulkou. Odkaz je hledán podle jména, a to příkazem starts-with. Ten umožňuje vybrat element jen podle začátku jména a zbytek

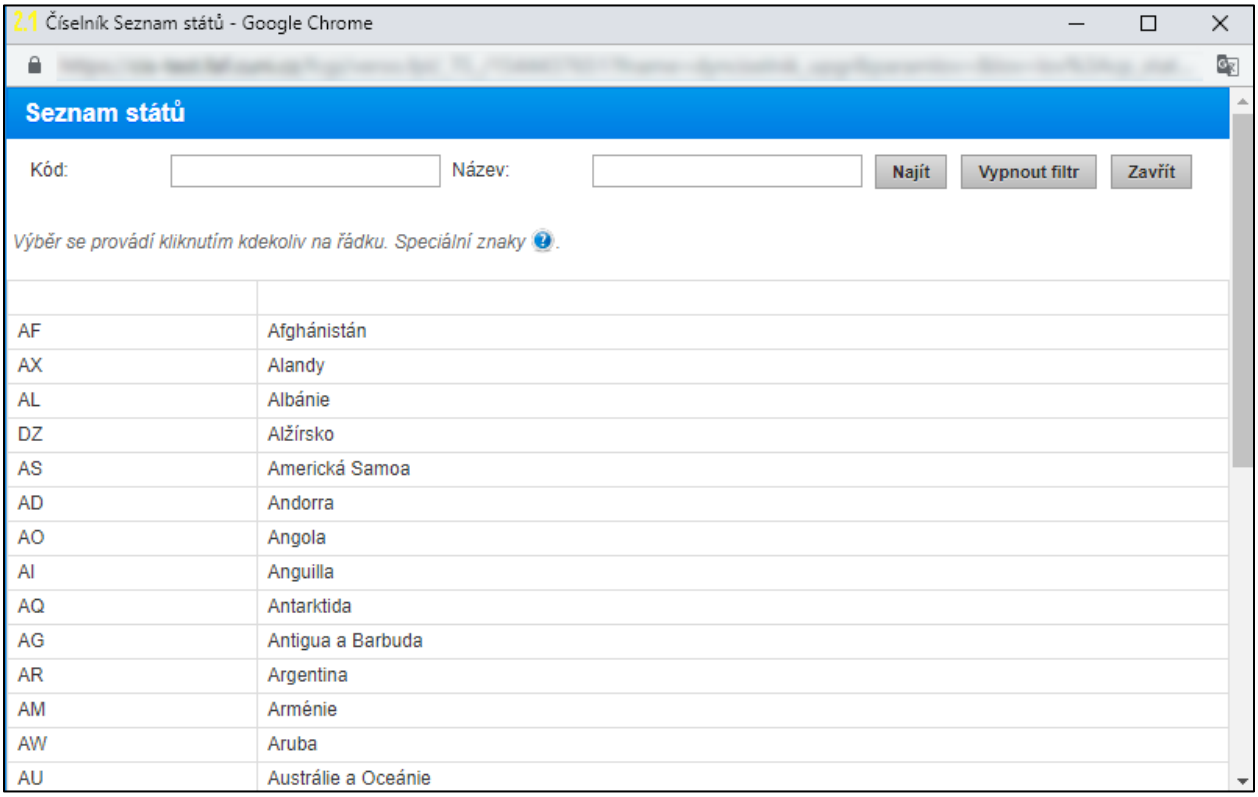
se může měnit. V tomto případě je použit proto, že je vysoká pravděpodobnost, že za poslední podmlčkou bude následovat číslo, a to se bude při každém zobrazení měnit. Následně se již jen na odkaz klikne. Vyplňování některých polí v novém záznamu je vidět v příloze č. 7. Cílem tohoto příkazu je tedy dojít k odkazu a následné kliknutí se váže na něj, ne na tabulku. Ta sloužila pouze ke specifikaci cesty. Přechod na záložku Hlavička lze v kódu vidět v příloze č. 8.



Obrázek č. 11 – Detail nového záznamu (zdroj: autor)

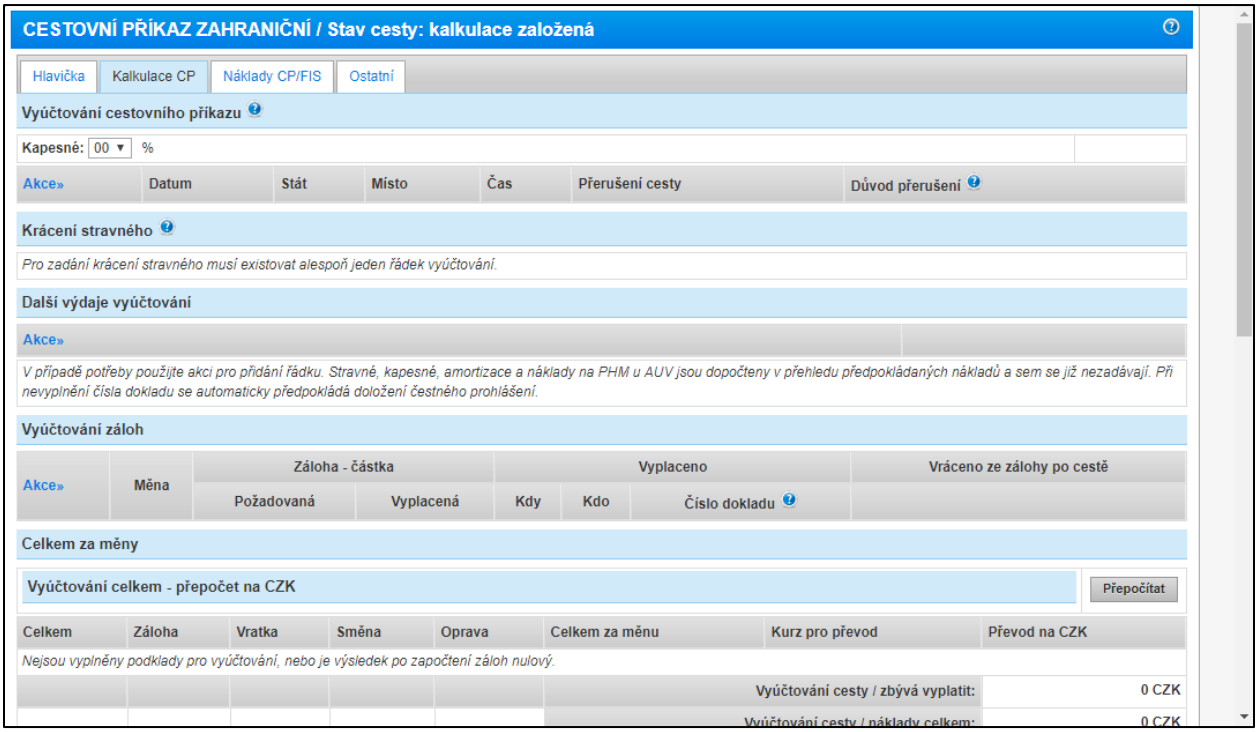
9. Na obrázku č. 14 je vidět příklad tzv. číselníku. Do číselníku se webdriver dostane vyhledáním elementu „a“, což v jazyce HTML označuje odkaz. Cesta k hledanému elementu je vidět v příloze č. 9. Když na číselník webdriver klikne, vyčká na načtení AJAXu a následně se do otevřeného okna přepne. Zde podle potřeby vybere zemi. Kód, který vede ke kliknutí na číselník, je popsán v předešlém bodě ve třetím odstavci.

K přepínání do nově otevřených oken zde slouží především metoda z třídy pomocných metod Utilities, a to switchToWindowUsingTitle (driver, „titulek“). Jak název metody napovídá, k přepnutí do konkrétního okna se využívá jeho titulek. Metodě se předávají dva parametry – instance webdriveru a právě zmíněný titulek okna.



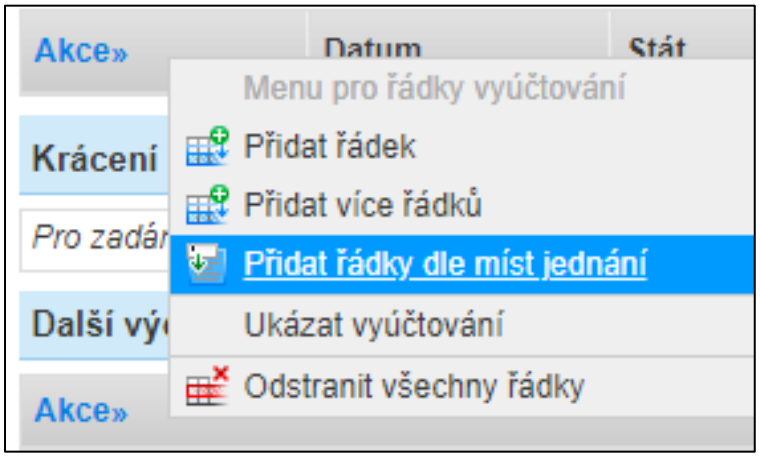
Obrázek č. 12 – Číselník zemí (zdroj: autor)

10. Po vyplnění všech potřebných polí v záložce hlavička, se webdriver přepne na záložku Kalkulace Cestovního příkazu (obrázek č. 15) viz příloha č. 10. Jednotlivé záložky mají své jednoznačné ID, takže je lze snadno podle ID vyhledat a kliknout na ně. Zde se nacházejí sekce stravné, zálohy na cestu a další nákladové položky. Zde je potřeba vždy rozkliknout modře zbarvené slovo Akce (Obrázek č. 16), které poskytne výběr možných dalších akcí v dané sekci.



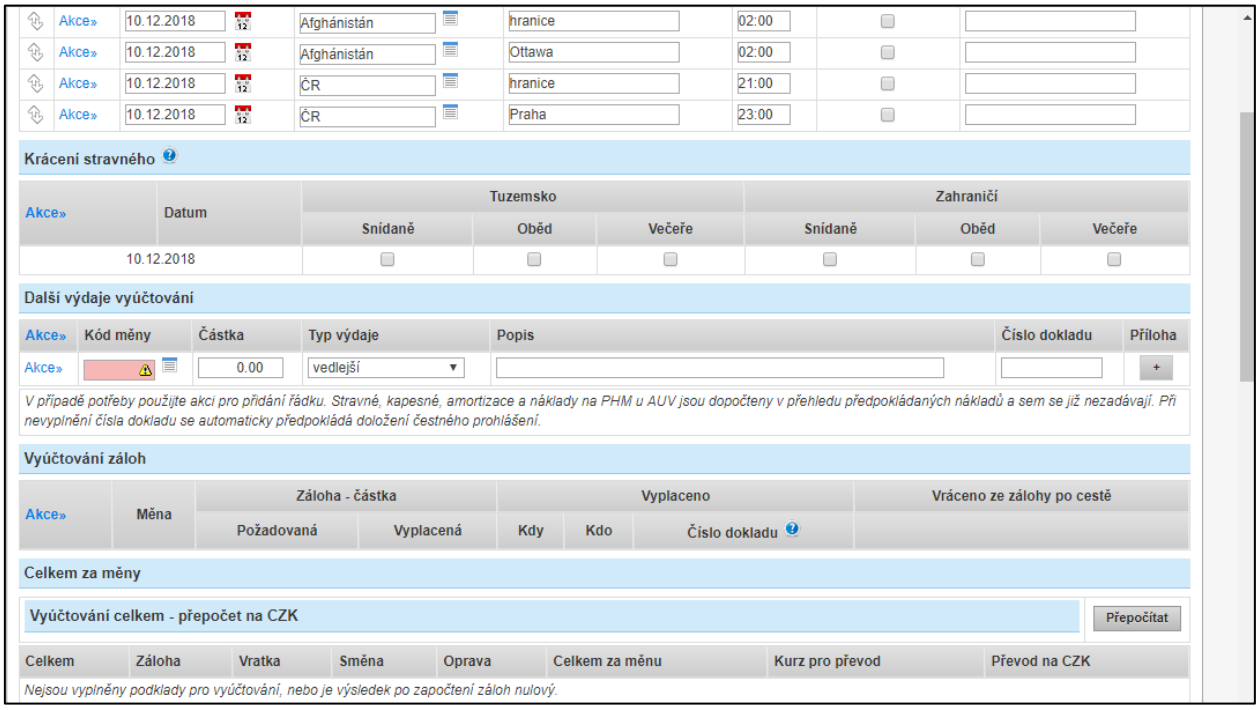
Obrázek č. 13 – Záložka kalkulace (zdroj: autor)

11. Na obrázku č. 16 je vidět rozbalené menu s možnými akcemi. Většinou se jedná o přidávání dalších řádků, respektive položek k dané sekci nebo jejich odstranění. V příloze č. 11 je vidět, že pro přidání řádku musí webdriver nejprve nalézt na stránce element ul, který v HTML jazyce označuje neuspořádaný seznam a ve struktuře pod tímto seznamem nalézt element a, který v sobě obsahuje text „Přidat řádky dle míst jednání“. Pro nalezení konkrétního elementu, je potřeba cestu, co nejvíce specifikovat. V tomto případě tomu napomáhá fakt, že element ul má jednoznačný identifikátor ID.



Obrázek č. 14 – Záložka kalkulace akce (zdroj: autor)

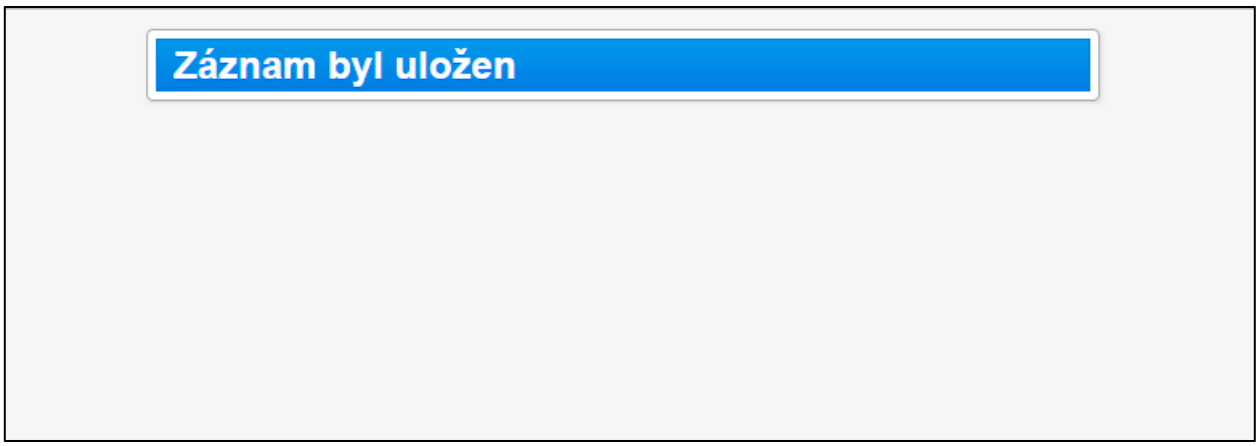
12. Po kliknutí na přidat řádek, se v příslušné sekci objeví další řádky k vyplnění, jak je vidět na obrázku č. 17. Ty obsahují několik údajů, které je potřeba vyplnit. U řádků v sekci „Další výdaje vyúčtování“ je to například kód měny, částka, typ výdajů a lze také vyplnit ke konkrétnímu výdaji popis a číslo dokladu.



Obrázek č. 15 – Záložka kalkulace přidané řádky (zdroj: autor)

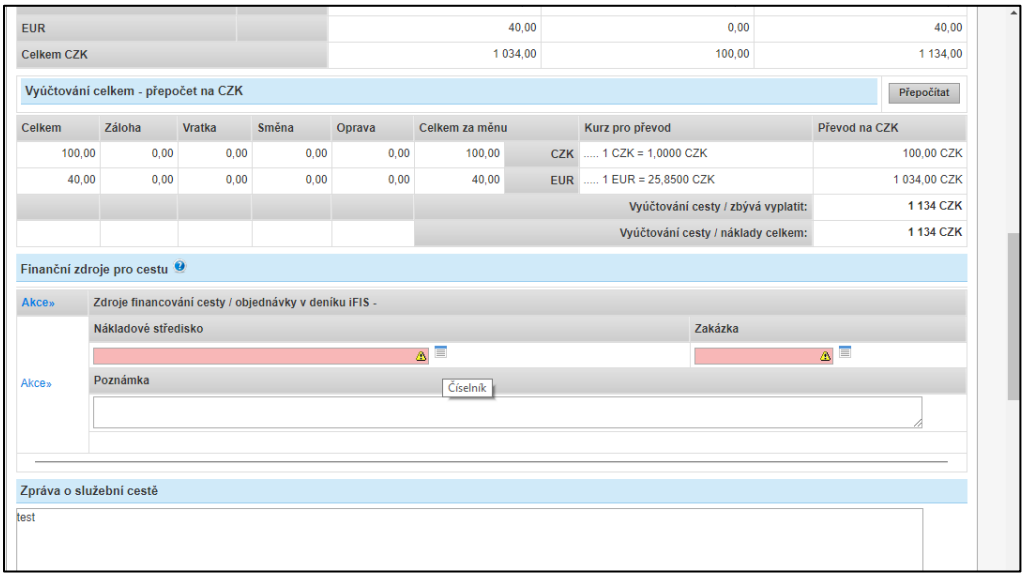
13. Po přidání všech požadovaných položek, webdriver ve spodní části nového záznamu klikne na tlačítko Uložit a pokud nikde „nesvítí“ pole červeně, tak se záznam i uloží. V případě, že by zůstalo nějaké nepovinné (označeno žlutě) pole nevyplněno, při ukládání uživatele na tuto skutečnost pouze upozorní dialogové okno, kde musí uživatel jen odkliknout, že je si této skutečnosti vědom a že chce i přesto dále pokračovat nebo se vrátit a pole dovyplnit. Zpráva o uložení záznamu je vidět na obrázku č. 18.

Po té, co se záznam uloží je v kódu možné vidět příkaz wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath(„//h1“)));. Ten zprostředkovává čekání, dokud nenastane nějaká očekávaná událost. Zde je čekání na načtení elementu h1, což je v jazyce HTML označení hlavního nadpisu. Hlavní nadpis se v záznamu nachází v jeho záhlaví. Tím se vlastně po uložení počká na úplné znovunačtení uloženého záznamu viz příloha č. 12.



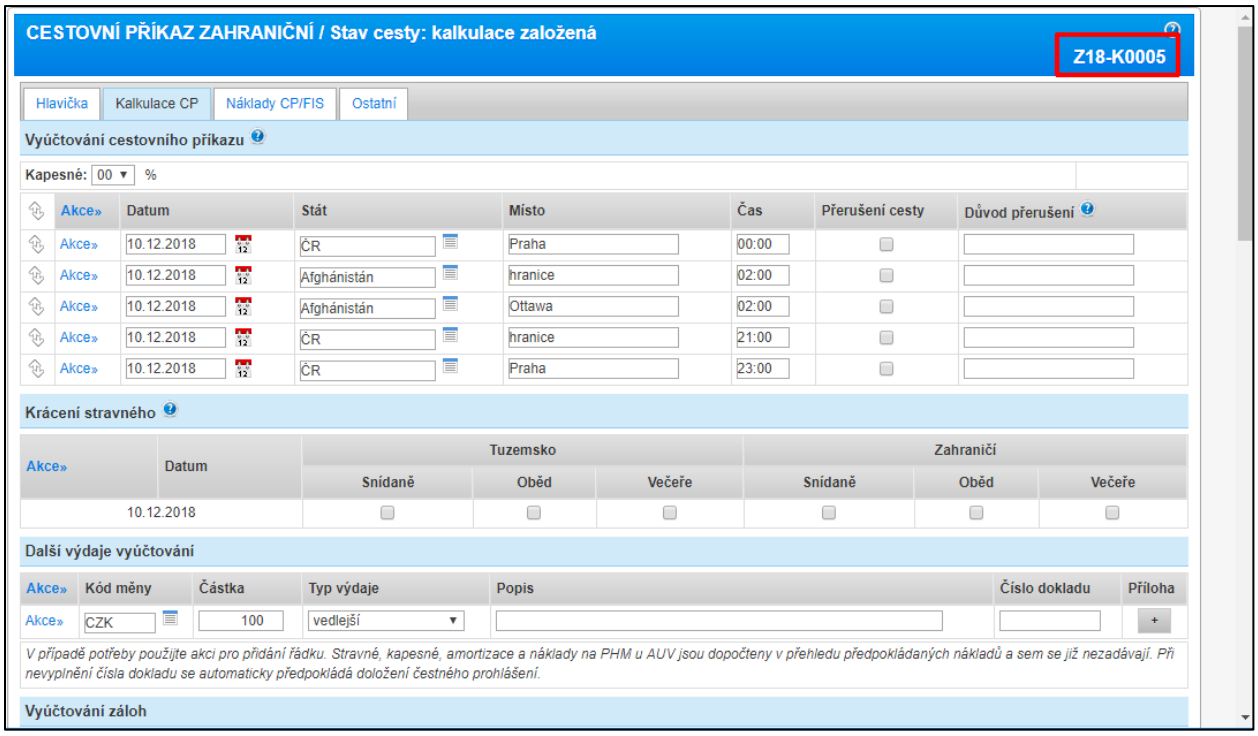
Obrázek č. 16 – Záznam uložen (zdroj: autor)

14. Po kliknutí na Uložit webdriver počká na načtení AJAXu, a poté se vrátí do okna s novým záznamem. Po návratu do okna se záznamem nyní webdriver přidá ještě řádky u finančních zdrojů. Pro jejich správné fungování je nutné záznam nejprve uložit. Poté je zde možné přidat nákladové středisko a zakázku viz obrázek č. 19. Kód doplnění finančních zdrojů, je vidět v příloze č. 13.



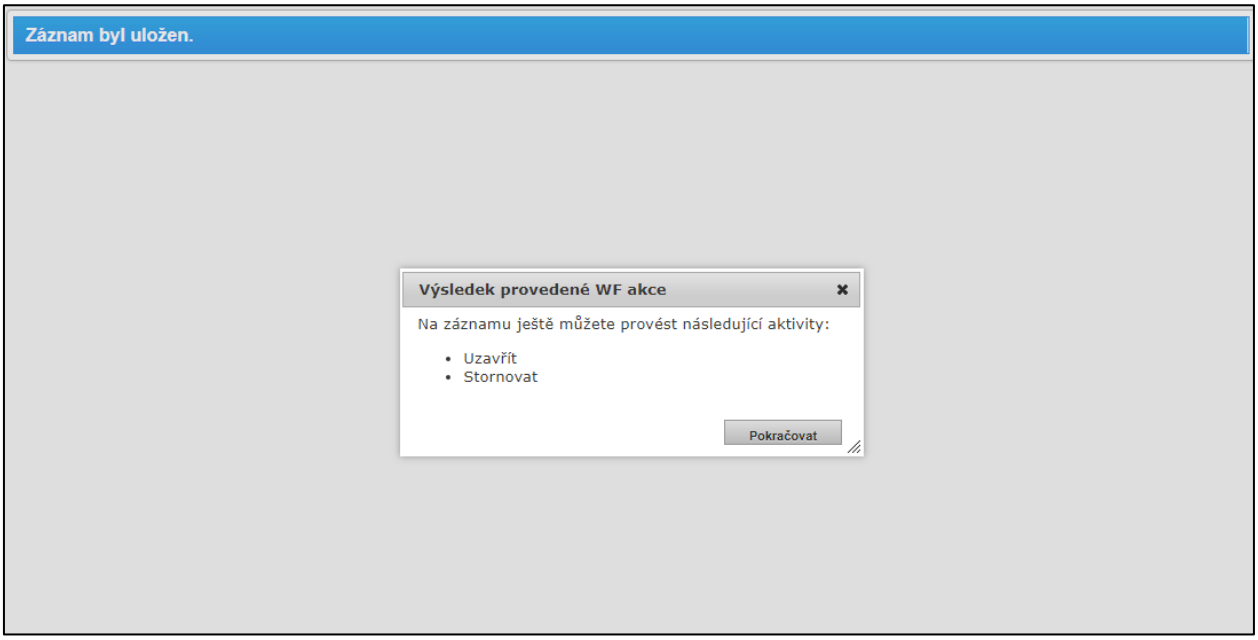
Obrázek č. 17 – Záložka kalkulace finanční zdroje (zdroj: autor)

15. Po zvolení nákladového střediska a zakázky z číselníku se záznam znovu uloží. Po dokončení ukládání záznamu webdriver ještě klikne na tlačítko Vygenerovat kód a uzavřít viz příloha č. 14. Tím se vygeneruje kód CP. Kód CP lze nyní nalézt v hlavičce záznamu v elementu h1 a je vidět na obrázku č. 20. Nyní webdriver ještě klikne na tlačítko uzavřít a tím se záznam konečně dostane do stavu Kalkulace uzavřená.



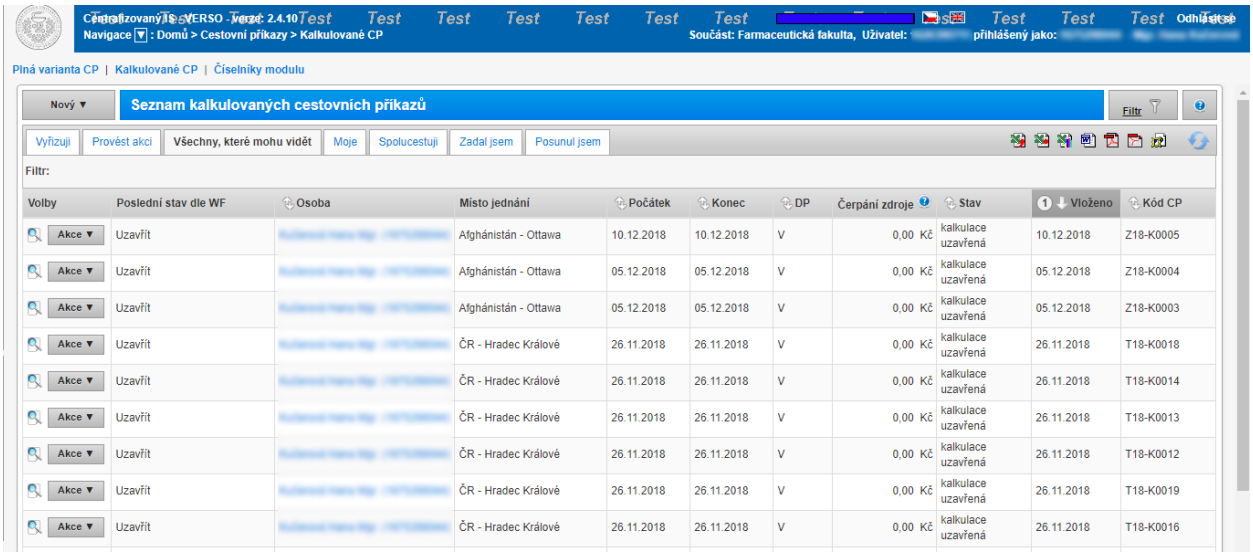
Obrázek č. 18 – Kód cestovního příkazu (zdroj: autor)

16. Je také potřeba zmínit, že po kliknutí na tlačítko Vygenerovat kód se zobrazí dialogové okno s informací o tom, které aktivity lze na záznamu ještě provést, jak je vidět na obrázku č. 21. Test následně zkontroluje přítomnost tohoto dialogového okna. Provedení kontroly přítomnosti okna a kliknutí na tlačítko pokračovat je vidět v příloze č. 14.



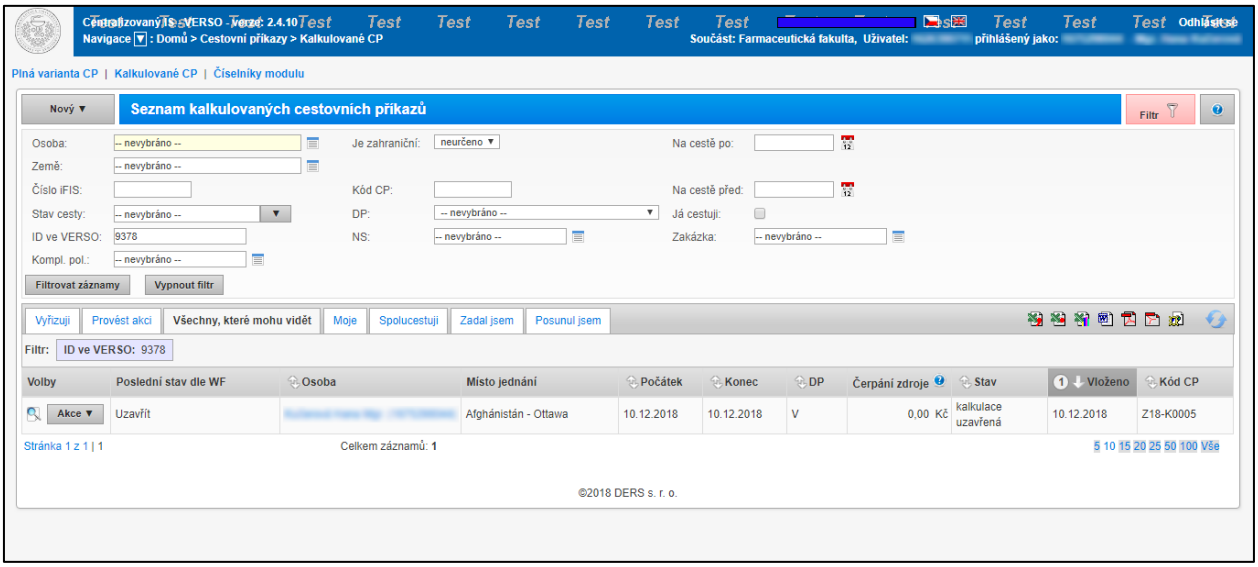
Obrázek č. 19 – Další aktivity na záznamu (zdroj: autor)

17. Po převedení záznamu do stavu Kalkulace uzavřená, se detail záznamu zavře a webdriver se přesune zpět na seznam všech kalkulovaných cestovních příkazů a zde se přepne na záložku Všechny, které mohu vidět. Vybraná záložka Všechny, které mohu vidět a seznam záznamů je vidět na obrázku č. 22.



Obrázek č. 22 – Všechny záznamy, které mohu vidět (zdroj: autor)

18. Nyní webdriver rozklikne filtr (obrázek č. 23) a vyhledá nově založený záznam CP pomocí jeho ID viz příloha č. 15, kde je v kódu vidět otevření filtru i vyplnění a následná filtrace. Webdriver vyhledává pomocí ID, protože to je téměř spolehlivě jedinečný údaj, který hledaný záznam jednoznačně identifikuje mezi ostatními. Po vyfiltrování záznamu proběhne kontrola jeho stavu, která je opět vidět v příloze č. 15. Zde je v kódu použita metoda elementHighlight. Touto metodou je možné hledaný element na stránce ohraničit barevným rámečkem pro vizuální kontrolu. Je to užitečné v případech, kdy je potřeba se při psaní testu přesvědčit, zda webdriver nalezl správný element a že nepracuje s jiným.



Obrázek č. 23 – Filtrace dle ID (zdroj: autor)

Test je tímto u konce. V případě, že nalezl záznam a záznam byl ve správném stavu, tak se v konzoli objeví zpráva, že je záznam v očekávaném stavu. Nejčastější chybou bývá, že se test v očekávaném stavu nenachází a poté je v procesu zakládání cestovního příkazu chyba. Další častou chybou je, že se nevygeneruje Kód CP, který je vidět na obrázku č. 20. Bývá to způsobené tím, že aplikace neměla dostatek času na vygenerování a potřebuje nějaký čas navíc.

Pro aplikaci Cestovní příkazy je testů více a každý se zaměřuje na jinou část. Je to například test, který netestuje Kalkulované cestovní příkazy, ale Plné cestovní příkazy. Ten bude, co se programování testu týká, podobný, jen s menšími odchylkami. Tento konkrétní test je zaměřen na vytváření nového záznamu, což je nejdůležitější proces aplikace Cestovní příkazy. V praxi aplikace samozřejmě slouží k evidenci pracovních cest zaměstnanců.

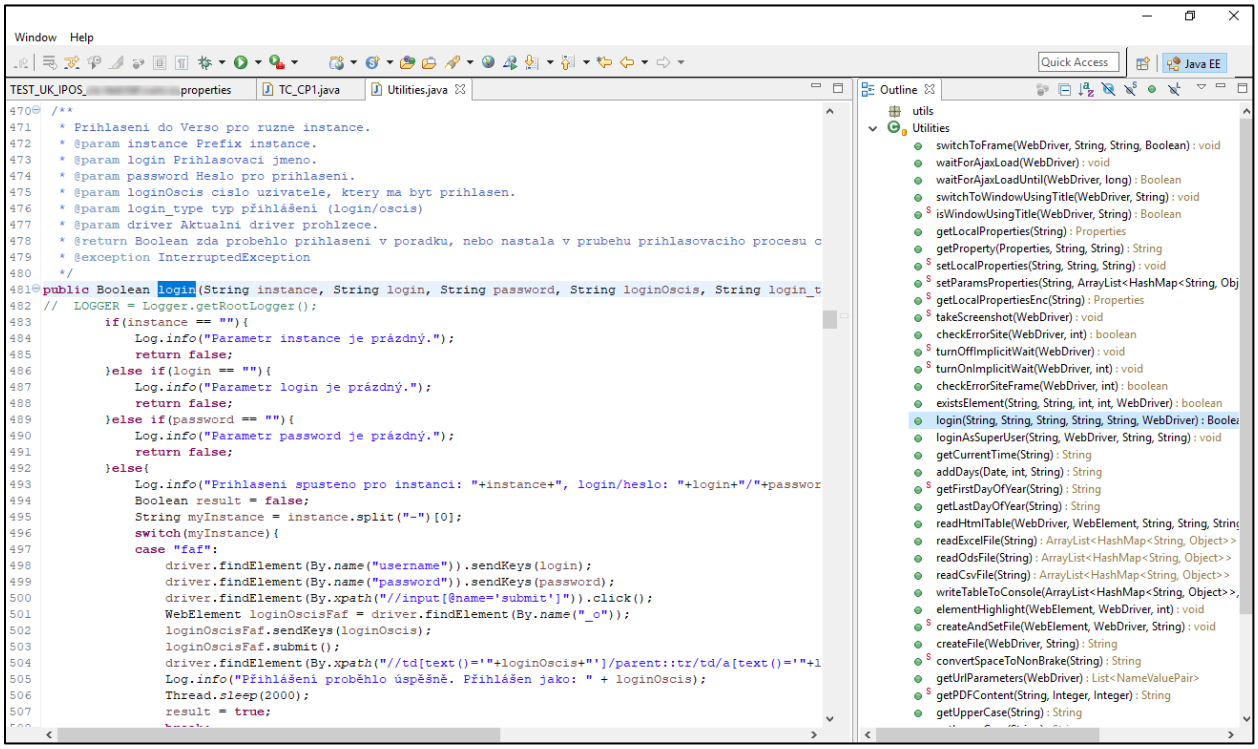
Pro lepší představu o tom, jak takové metody fungují, je zde obrázek č. 24 s vnitřní strukturou metody login. Metodě login se předává 5 parametrů. Jsou to parametry instance, login, password, loginOscis a login_type. Parametr instance vlastně udává, na jaké URL adrese se bude test pouštět. Každá instance má menší či větší odchylky od jiné instance, a to je potřeba v testu ošetřit, aby test úspěšně prošel na všech instancích. Parametr login v sobě skrývá přihlašovací jméno a parametr password přihlašovací heslo. Dále je tu parametr loginOscis. Po přihlášení se totiž uživatel dostane do sekce, kde je potřeba vybrat osobu podle osobního čísla a k tomu právě slouží loginOscis. Login_type již jen umožňuje v případě potřeby k přihlášení místo parametru login použít parametr loginOscis.

Metoda tedy zkontroluje přítomnost parametrů a následně se vyhodnotí konstrukce switch. Konstrukce switch ve své podstatě obsluhuje podmínky. Usnadňuje zápis kódu, kdy je potřeba zapsat několik podmínek za sebou. Konstrukce switch obsahuje několik case, které splňují funkci podmínek. Každá case má danou podmínku, po jejímž splnění se blok kódu dané case provede. V tomto případě se jedná o výběr verze přihlašovací metody pro vybranou instanci. Příkladem může být case „faf“. Tento case udává, že pokud bude parametr instance naplněn hodnotou „faf“, tak se provede blok kódu právě u této case. Jak již bylo zmíněno, tak i tady je potřeba ošetřit případy pro všechny instance. Jednotlivé instance mají totiž odlišné HTML lokátory. Může se například stát, že místo lokátoru name, bude použit lokátor id.

WebDriver nabízí různé možnosti vyhledávání HTML elementů ve webové aplikaci. Mezi nejčastěji využívané patří lokátory name, id a class. Dále je také hojně využíváno lokátoru xpath. Xpath udává celou cestu k danému HTML elementu, a to především v případě, kdy není jednoznačně určen pomocí lokátorů id nebo name.

Po vyplnění přihlašovacího jména a hesla webdriver klikne na tlačítko přihlásit. Po přihlášení se test dostane na stránku se seznamem osob s jejich osobními čísly (loginOscis). Webdriver zde vyhledá pole, do kterého může oscis vepsat a následně vyfiltrovat. Po filtraci vybere osobu se zadaným osobním číslem, klikne na ni a test pokračuje dále.

Nakonec metoda jen vypíše hlášku s informací o tom, za kterou osobu je uživatel přihlášen. Pro ujištění, že se stránka kompletně načte, je zde přidáno čekání po dobu dvou sekund navíc.




```
/**
 * Prihlaseni do Verso pro ruzne instance.
 * @param instance Prefix instance.
 * @param login Prihlasovaci jmeno.
 * @param password Heslo pro prihlaseni.
 * @param loginOscis cislo uzivatele, ktery ma byt prihlasen.
 * @param login_type typ přihlášení (login/oscis)
 * @param driver Aktualni driver prohlzece.
 * @return Boolean zda probehlo prihlaseni v poradku, nebo nastala v prubehu prihlasovaciho procesu c
 * @exception InterruptedException
 */
public Boolean login(String instance, String login, String password, String loginOscis, String login_t
//  LOGGER = Logger.getRootLogger();
        if(instance == ""){
            Log.info("Parametr instance je prázdný.");
            return false;
        }else if(login == ""){
            Log.info("Parametr login je prázdný.");
            return false;
        }else if(password == ""){
            Log.info("Parametr password je prázdný.");
            return false;
        }else{
            Log.info("Prihlaseni spusteno pro instanci: "+instance+", login/heslo: "+login+"/"+passwor
            Boolean result = false;
            String myInstance = instance.split("-")[0];
            switch(myInstance){
            case "faf":
                driver.findElement(By.name("username")).sendKeys(login);
                driver.findElement(By.name("password")).sendKeys(password);
                driver.findElement(By.xpath("//input[@name='submit']")).click();
                WebElement loginOscisFaf = driver.findElement(By.name("_o"));
                loginOscisFaf.sendKeys(loginOscis);
                loginOscisFaf.submit();
                driver.findElement(By.xpath("//td[text()='"+loginOscis+"']/parent::tr/td/a[text()='"+l
                Log.info("Přihlášení proběhlo úspěšně. Přihlášen jako: " + loginOscis);
                Thread.sleep(2000);
                result = true;
```

Obrázek č. 20 – Metoda login (zdroj: autor)

8 Shrnutí

Vytvořený test testuje požadovanou funkcionalitu webové aplikace. Tím ale práce na testu nekončí. Test je nadále nezbytné udržovat, aby odpovídal aktuálnímu stavu aplikace. Pokud aplikace stále funguje tak, jak je očekáváno, test doběhne úspěšně do konce. Pokud ne, test skončí s chybou a s chybovou hláškou, která informuje o tom, co je špatně. Úskalím automatizovaného testu jsou právě tyto chybové hlášky. Při psaní automatizovaného testu je nutné psát chybové hlášky natolik srozumitelně, aby je pochopil kdokoliv, kdo test pustí. A protože k testu mají přístup nejen testeři, ale i jiní uživatelé, kteří potřebují test pustit, je nutné, aby byly chybové hlášky jednoznačné a každý jim porozuměl. Test však může z různých důvodů skončit kdykoliv v jeho průběhu, a to i tam, kde chybová hláška připravena není. V tomto případě nemusí být chyba v aplikaci, ale v testu. V této situaci se vypíše jen systémová chybová hláška a je většinou nutné ji konzultovat s testerem.

I přesto, že se v teorii říká, že by se automatizovaný test měl zavádět až pro již fungující a stabilní aplikaci, automatizované testy se používají již v počátcích vývoje aplikace. Je to právě v začátcích, kdy se provádí nejvíce změn a kdy je potřeba testování průchodu aplikací urychlit. K takovému účelu jistě poslouží automatizovaný test, který je možné pustit po každé drobné změně aplikace.

Je potřeba říci, že rozsáhlost problematiky automatizovaného testování neumožňuje pokrýt v práci vše, co je ještě nutné udělat pro zprovoznění automatizovaného testu. Cíle práce, představit automatizované testování, bylo snad dosaženo. Nicméně je stále řada věcí, které jsou „na pozadí“ a jsou pro fungování testu nezbytné. Jednotlivým funkcionalitám a dalším nástrojům by bylo nutné věnovat se zvlášť a každý nástroj by bylo možné detailně analyzovat v samostatné práci.

9 Závěr

Testování, obzvláště automatizované testování, je velice komplexní záležitostí, a to zejména z důvodu množství nástrojů, které jsou v tomto procesu použity a znalostí, kterých je pro úspěšné sestavení automatizovaného testu potřeba. Ačkoliv je v oboru informačních technologií pozice tester většinou nástupní pozicí nových zaměstnanců, dalo by se říci, že pro automatizované testování je již potřeba člověk se znalostmi a zkušenostmi. Nástroje automatizovaného testování jsou rozsáhlé a nabízí řadu možností, ale pro využití jejich plného potenciálu je zapotřebí, aby jejich studiu tester věnoval dostatečné množství času.

Automatizovaný test nicméně oproti manuálnímu průchodu aplikací šetří při nejmenším čas. V případě, že je sekvence kroků jasná, je jednodušší pustit test a následně sledovat výstup v konzoli než procházet aplikaci manuálně. Automatizovaný test na nic nezapomene a poskytuje konzistentní výstup. Někdy je potřeba otestovat aplikaci i nesčetněkrát a manuální testování v tomto případě nedává příliš smysl.

10 Seznam použité literatury

1. BERGERON, Nicolas, GUÉRIN, Francis, LE BRETON, Gabriel a BEROUAL, Oussama (2016). Declarative layout constraints for testing web applications. *Journal of Logical and Algebraic Methods in Programming.* roč. 85, s. 737-758 [online], ISSN: 2352-2208. [cit. 2019-01-08].

2. BUREŠ, Miroslav, RENDA, Miroslav, DOLEŽEL, Michal, SVOBODA, Peter, GRÖSSL, Zdeněk, KOMÁREK, Martin, MACEK, Ondřej a MLYNÁŘ, Radoslav (2016). *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. Praha: Grada, 229 s. Profes!onal. ISBN 978-80-247-5594-6.

3. DOĞAN, Serdar, BETIN-CAN, Aysu a GAROUSI, Vahid (2014). Web application testing: A systematic literature review. *Journal of Systems and Software*, roč. 91, s. 174-201 [online], ISSN: 0164-1212. [cit. 2019-01-08].

4. FACI, Noura, MAAMAR, Zakaria, UGLJANIN, Emir a BENSLIMANE, Djamal (2017). Web 2.0 applications in the workplace: How to ensure their proper use?. *Computers in Industry,* roč. 88, s. 1-11 [online], ISSN: 0166-3615. [cit. 2019-01-08].

5. FAIRCLOTH, Jeremy (2011). *Penetration Tester's Open Source Toolkit.* 3. vydání. Syngress. ISBN 9781597496278.

6. GAROUSI, Vahid, MESBAH, Ali, BETIN-CAN, Aysu a MIRSHOKRAIE, Shabnam (2013). A systematic mapping study of web application testing. *Information and Software Technology,* roč. 55, s. 1374-1396 [online], ISSN: 0950-5849. [cit. 2019-01-08].

7. GOJARE, Satish a GAIGAWARE, Dhanashree (2015). Analysis and Design of Selenium WebDriver Automation Testing Framework. *Procedia Computer Science,* roč. 50, s. 341-346 [online], ISSN: 1877-0509. [cit. 2019-01-08].

8. KUMAR SHIVAKUMAR, Shailesh (2014). *Architecting High Performing, Scalable and Available Enterprise Web Applications*. Morgan Kaufmann. ISBN 9780128022580.

9. PATTON, Ron (2003). *Testování softwaru*. Přeložil David KRÁSENSKÝ. Brno: Computer Press, xiv, 313 s. Pro každého uživatele. ISBN 80-7226-636-5.

10. ROUDENSKÝ, Petr a HAVLÍČKOVÁ, Anna (2013). *Řízení kvality softwaru: průvodce testováním*. Brno: Computer Press, 208 s. ISBN 978-80-251-3816-8.

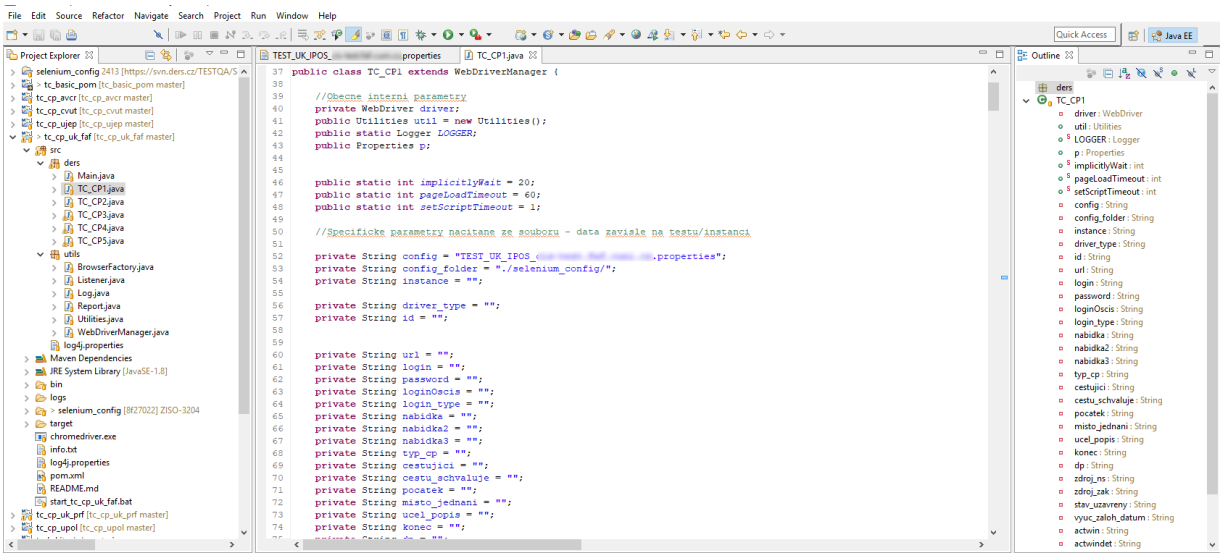
11. Selenium Documentation. *Selenium HQ: Browser Automation* [online]. Selenium Project, (2012) [cit. 2019-01-08]. Dostupné z: https://www.seleniumhq.org/docs/

12. SHAZHAD, Farrukh (2017). Modern and Responsive Mobile-enabled Web Applications. *Procedia Computer Science, roč. 110, s. 410-415* [online], ISSN: 1877-0509 [cit. 2019-01-08].

13. TestNG. *TestNG, (2004)* [online]. [cit. 2019-01-08]. Dostupné z: https://testng.org/doc/documentation-main.html

14. Quizlet Inc. (2019) *SDLC Waterfall Model*. [online], [cit. 2019-2-1]. Dostupné z https://quizlet.com/ca/301563959/sdlc-waterfall-model-diagram/

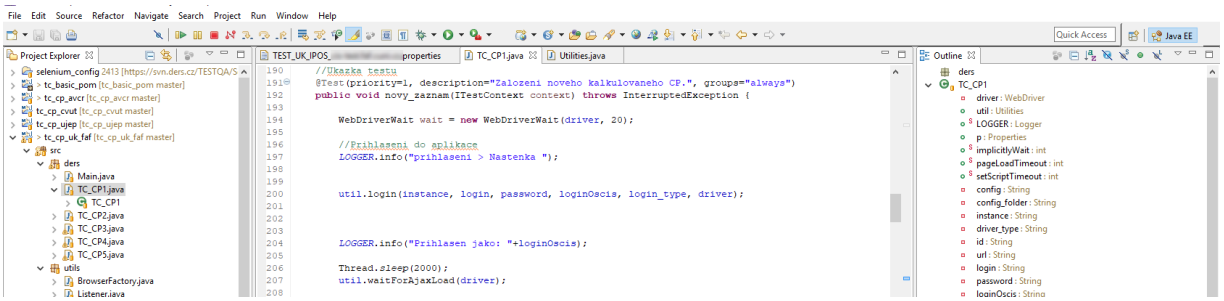
11 Přílohy



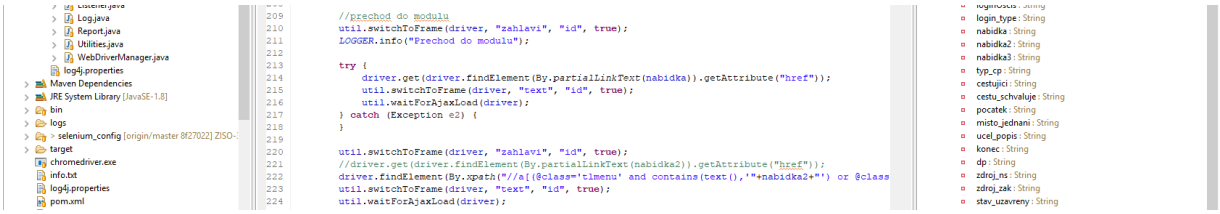
Příloha č. 1 – Parametry v testovací třídě

```
180                 setDriver(driver);
181
182                 driver.manage().window().maximize();
183
184                 driver.get(url);
185                 LOGGER.info("Test is running on: " + url);
186
187
188     }
```

Příloha č. 3 – Metoda getUrl

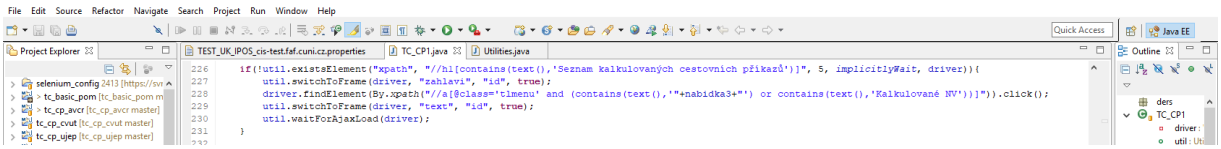


Příloha č. 2 – Metoda login



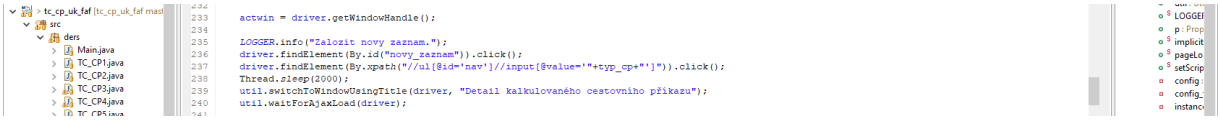
```
//prechod do modulu
util.switchToFrame(driver, "zahlavi", "id", true);
LOGGER.info("Prechod do modulu");

try {
    driver.get(driver.findElement(By.partialLinkText(nabidka)).getAttribute("href"));
    util.switchToFrame(driver, "text", "id", true);
    util.waitForAjaxLoad(driver);
} catch (Exception e2) {
}

util.switchToFrame(driver, "zahlavi", "id", true);
//driver.get(driver.findElement(By.partialLinkText(nabidka2)).getAttribute("href"));
driver.findElement(By.xpath("//a[@class='tlmenu' and contains(text(),'"+nabidka2+"') or @class
util.switchToFrame(driver, "text", "id", true);
util.waitForAjaxLoad(driver);
```

Příloha č. 4 – Přechod do modulu



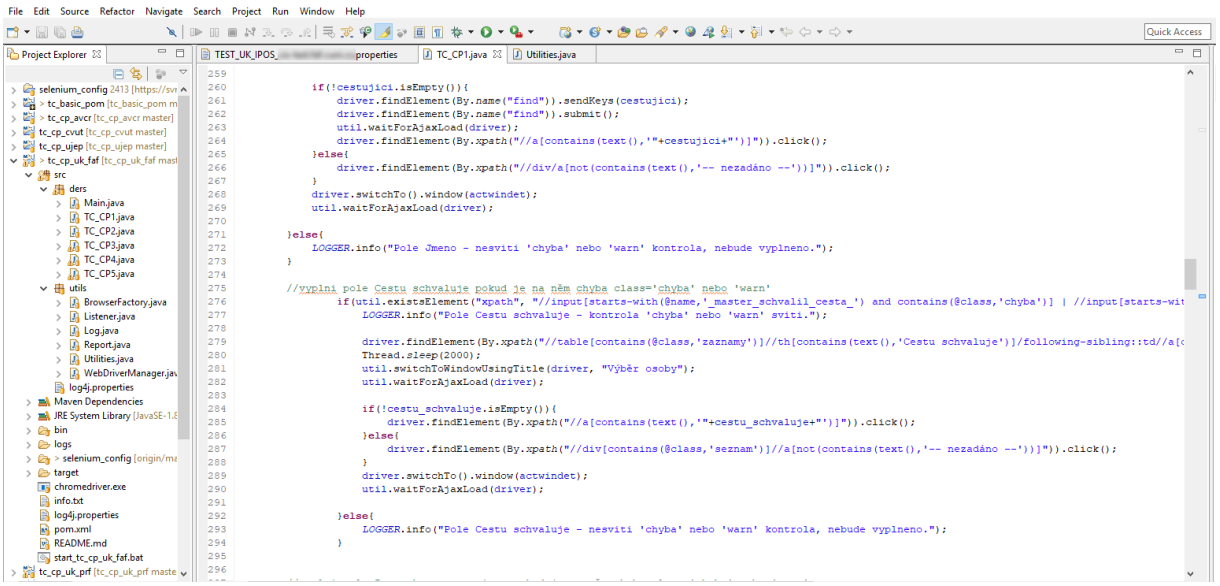
```
if(!util.existsElement("xpath", "//h1[contains(text(),'Seznam kalkulovaných cestovních příkazů')]", 5, implicitlyWait, driver)){
    util.switchToFrame(driver, "zahlavi", "id", true);
    driver.findElement(By.xpath("//a[@class='tlmenu' and (contains(text(),'"+nabidka3+"') or contains(text(),'Kalkulované NV'))]")).click();
    util.switchToFrame(driver, "text", "id", true);
    util.waitForAjaxLoad(driver);
}
```

Příloha č. 5 – Výběr kalkulovaných CP



```
actwin = driver.getWindowHandle();

LOGGER.info("Zalozit novy zaznam.");
driver.findElement(By.id("novy_zaznam")).click();
driver.findElement(By.xpath("//ul[@id='nav']//input[@value='"+typ_cp+"']")).click();
Thread.sleep(2000);
util.switchToWindowUsingTitle(driver, "Detail kalkulovaného cestovního příkazu");
util.waitForAjaxLoad(driver);
```

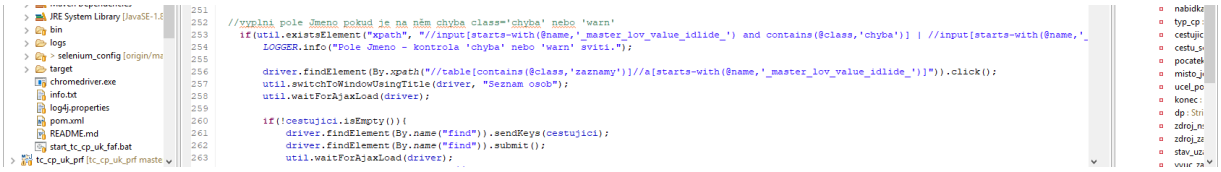
Příloha č. 6 – Založit nový záznam



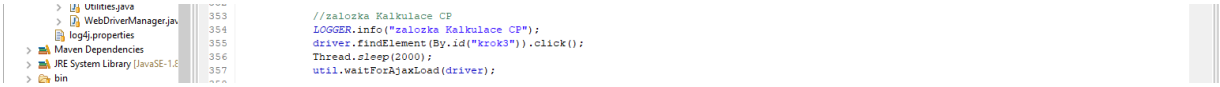
```
if(!cestujici.isEmpty()){
    driver.findElement(By.name("find")).sendKeys(cestujici);
    driver.findElement(By.name("find")).submit();
    util.waitForAjaxLoad(driver);
    driver.findElement(By.xpath("//a[contains(text(),'"+cestujici+"')]")).click();
}else{
    driver.findElement(By.xpath("//div/a[not(contains(text(),'-- nezadáno --'))]")).click();
}
driver.switchTo().window(actwindet);
util.waitForAjaxLoad(driver);

}else{
    LOGGER.info("Pole Jmeno - nesviti 'chyba' nebo 'warn' kontrola, nebude vyplneno.");
}

//vyplni pole Cestu schvaluje pokud je na něm chyba class='chyba' nebo 'warn'
    if(util.existsElement("xpath", "//input[starts-with(@name,'_master_schvalil_cesta_') and contains(@class,'chyba')] | //input[starts-wit
        LOGGER.info("Pole Cestu schvaluje - kontrola 'chyba' nebo 'warn' sviti.");

        driver.findElement(By.xpath("//table[contains(@class,'zaznamy')]//th[contains(text(),'Cestu schvaluje')]/following-sibling::td//a[
        Thread.sleep(2000);
        util.switchToWindowUsingTitle(driver, "Výběr osoby");
        util.waitForAjaxLoad(driver);

        if(!cestu_schvaluje.isEmpty()){
            driver.findElement(By.xpath("//a[contains(text(),'"+cestu_schvaluje+"')]")).click();
        }else{
            driver.findElement(By.xpath("//div[contains(@class,'seznam')]//a[not(contains(text(),'-- nezadáno --'))]")).click();
        }
        driver.switchTo().window(actwindet);
        util.waitForAjaxLoad(driver);

        }else{
            LOGGER.info("Pole Cestu schvaluje - nesviti 'chyba' nebo 'warn' kontrola, nebude vyplneno.");
        }
```

Příloha č. 7 – Vyplnění položek na záznamu

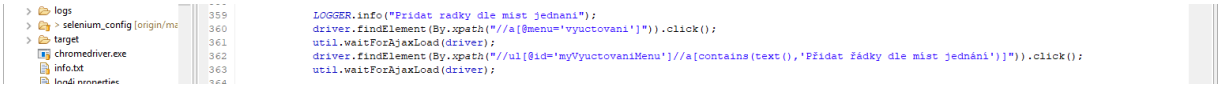
Příloha č. 8 – Záložka hlavička



Příloha č. 9 – Otevření číselníku



Příloha č. 10 – Přechod na záložku Kalkulace CP

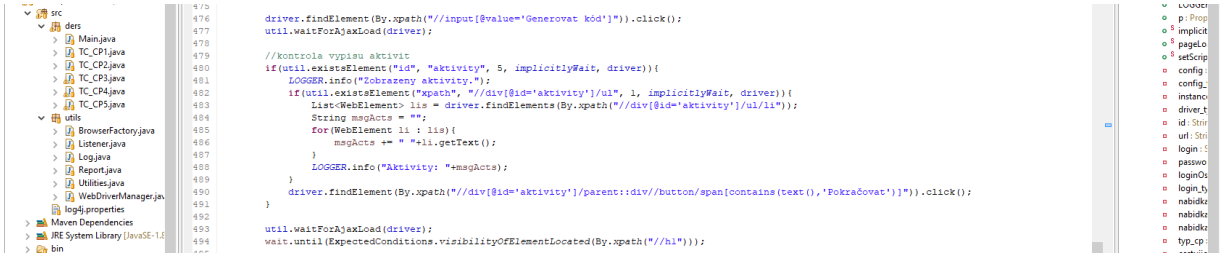


Příloha č. 11 – Přidat řádky

Příloha č. 12 – Uložení záznamu a čekání na element h1

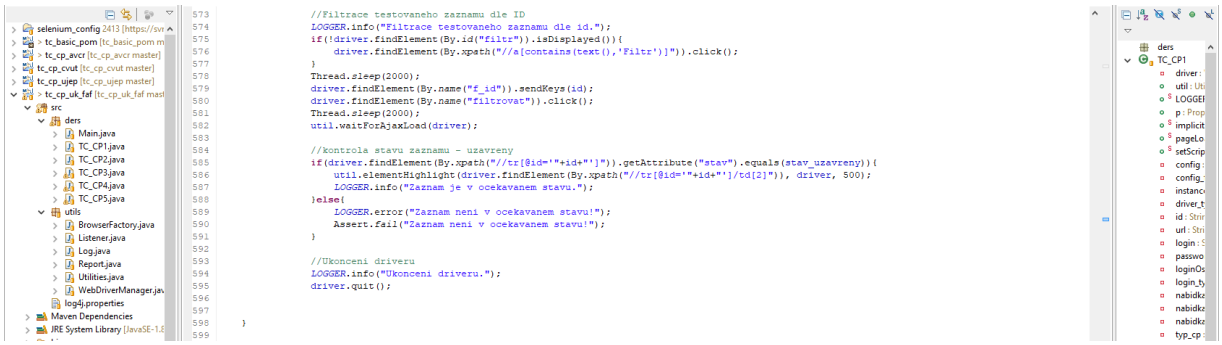
```
LOGGER.info("Doplnit financni zdroje");

util.waitForAjaxLoad(driver);
driver.findElement(By.xpath("//a[@menu='zdroje_po']")).click();
util.waitForAjaxLoad(driver);
driver.findElement(By.xpath("//ul[@id='myZdrojePoMenu']//a[contains(text(),'Přidat řádek')]")).click();
util.waitForAjaxLoad(driver);

//vyplni pole NS pouze pokud je na něm chyba class='chyba' nebo 'warn'
        if(util.existsElement("xpath", "//input[starts-with(@name,'_zdroje_po_lov_value_idprac_') and contains(@class,'chyba')] | //input[
            LOGGER.info("Pole Nakladove stredisko - kontrola 'chyba' nebo 'warn' sviti.");

            driver.findElement(By.xpath("//table[contains(@class,'zaznamy')]//a[starts-with(@name,'_zdroje_po_lov_value_idprac_')]")).clic
            Thread.sleep(2000);
            util.switchToWindowUsingTitle(driver, "Nákladová střediska");
            util.waitForAjaxLoad(driver);

            if(!zdroj_ns.isEmpty()){
                driver.findElement(By.name("find")).sendKeys(zdroj_ns);
                driver.findElement(By.name("find")).submit();
                util.waitForAjaxLoad(driver);
                driver.findElement(By.xpath("//a[contains(text(),'"+zdroj_ns+"')]")).click();
            }else{
                driver.findElement(By.xpath("//div[contains(@class,'seznam')]/div/a[not(contains(text(),'-- nezadáno --'))]")).click();
            }
            driver.switchTo().window(actwindet);
            util.waitForAjaxLoad(driver);
```

Příloha č. 13 – Doplnění finančních zdrojů



```
driver.findElement(By.xpath("//input[@value='Generovat kód']")).click();
util.waitForAjaxLoad(driver);

//kontrola vypisu aktivit
if(util.existsElement("id", "aktivity", 5, implicitlyWait, driver)){
    LOGGER.info("Zobrazeny aktivity.");
    if(util.existsElement("xpath", "//div[@id='aktivity']/ul", 1, implicitlyWait, driver)){
        List<WebElement> lis = driver.findElements(By.xpath("//div[@id='aktivity']/ul/li"));
        String msgActs = "";
        for(WebElement li : lis){
            msgActs += " "+li.getText();
        }
        LOGGER.info("Aktivity: "+msgActs);
    }
    driver.findElement(By.xpath("//div[@id='aktivity']/parent::div//button/span[contains(text(),'Pokračovat')]")).click();
}

util.waitForAjaxLoad(driver);
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//hi")));
```

Příloha č. 15 – Generovat kód CP a kontrola aktivit



```
        //Filtrace testovaneho zaznamu dle ID
        LOGGER.info("Filtrace testovaneho zaznamu dle id.");
        if(!driver.findElement(By.id("filtr")).isDisplayed()){
            driver.findElement(By.xpath("//a[contains(text(),'Filtr')]")).click();
        }
        Thread.sleep(2000);
        driver.findElement(By.name("f_id")).sendKeys(id);
        driver.findElement(By.name("filtrovat")).click();
        Thread.sleep(2000);
        util.waitForAjaxLoad(driver);

        //kontrola stavu zaznamu - uzavreny
        if(driver.findElement(By.xpath("//tr[@id='"+id+"']")).getAttribute("stav").equals(stav_uzavreny)){
            util.elementHighlight(driver.findElement(By.xpath("//tr[@id='"+id+"']/td[2]")), driver, 500);
            LOGGER.info("Zaznam je v ocekavanem stavu.");
        }else{
            LOGGER.error("Zaznam neni v ocekavanem stavu!");
            Assert.fail("Zaznam neni v ocekavanem stavu!");
        }

        //Ukonceni driveru
        LOGGER.info("Ukonceni driveru.");
        driver.quit();
    }
```

Příloha č. 14 – Filtrace záznamu dle ID a kontrola stavu

Zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Systémové inženýrství a informatika
Forma: Prezenční
Obor/komb.: Informační management (im3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Šroubek Petr	V Domkách 7 1382, Duchcov	I1500245

TÉMA ČESKY:

Automatizované testování webových aplikací

TÉMA ANGLICKY:

Automated Testing of Web Applications

VEDOUCÍ PRÁCE:

Ing. Tereza Otčenášková, BA - KIT

ZÁSADY PRO VYPRACOVÁNÍ:

1. Úvod
2. Cíl práce
3. Metodika zpracování
4. Webové aplikace
5. Automatizované testování
6. Nástroje automatizovaného testování
7. Praktická část
8. Závěry a doporučení
9. Seznam použité literatury
10. Přílohy

Cílem práce je představení nástrojů, metod a postupů při automatizovaném testování. Cílem praktické části je vytvoření testovacích případů za pomocí nástrojů TestNG, Selenium WebDriver a nástroje Selenium Grid, který umožní paralelní běh testu na více prostředích.

SEZNAM DOPORUČENÉ LITERATURY:

1. BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.

2.ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: průvodce testováním. Brno: Computer Press, 2013, 208 s. ISBN 978-80-251-3816-8.

3.GUNDECHA, Unmesh. Selenium 2 Cookbook. New Edition. Birmingham: Packt Publishing, Limited, 2012. ISBN 1849515743.

4. PATTON, Ron. Testování softwaru. Přeložil David KRÁSENSKÝ. Brno: Computer Press, 2003, xiv, 313 s. Pro každého uživatele. ISBN 80-7226-636-5.

5. AVASARALA, Satya. Selenium WebDriver practical guide interactively automate web applications using Selenium WebDriver. Birmingham, UK: Packt Pub, 2014. ISBN 9781782168850.