Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informační systémy a technologie

Pokročilé možnosti automatizovaného testování nástrojem Selenium WebDriver

DIPLOMOVÁ PRÁCE

Student: Bc. Ondřej Špalek

Vedoucí: Doc. Ing. Alena Buchalcevová, Ph.D.

Oponent: Ing. Lubomír Mičko

2016

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne 27. 4. 2016 . . . . . .

Bc. Ondřej Špalek

Poděkování

Chtěl bych poděkovat paní doc. Ing. Aleně Buchalcevové, Ph.D., za vedení této práce a za rady a připomínky, které mi velmi pomohly při psaní této diplomové práce.

Abstrakt

Tato diplomová práce se zaměřuje na představení pokročilých možností práce s testovacím nástrojem Selenium WebDriver. V úvodu práce je popsáno testování softwaru a nástroj Selenium. V druhé části práce jsou popsány pokročilé funkce Selenium RemoveWebDriver a Selenium Grid. Další část obsahuje přehled dostupných nástrojů a aplikací, které rozšiřují Selenium WebDriver. Jedná se o aplikační rámce využívající WebDriver API, softwarové kontejnery, nástroje pro automatizaci, nástroje pro průběžnou integraci a služby umožňující spouštět testy vzdáleně na hostovaném prostředí. V závěru práce je umístěna případová studie, která popisuje zavádění automatizovaného testování ve firmě BellaDati. Součástí práce je také příloha obsahující uživatelskou příručku k aplikačnímu rámci Geb, jenž byl použit v rámci případové studie.

Klíčová slova

Automatizace testování webových aplikací, Selenium, WebDriver, nástroje pro automatizaci testování, testování, metodika MMSP, rozšíření pro Selenium WebDriver.

Abstract

This diploma thesis is focused on introduction of advanced possibilities of working with testing Selenium WebDriver framework. Basics of software testing and Selenium framework are described in first part of the text. Second chapter contains overview of available tools and applications which enhance Selenium WebDriver. These include frameworks using WebDriver API, software containers, automatization tools, continuous integration tools and services which enable to run tests remotely. Last chapter contains case study which describes implementation of automated testing in BellaDati company. The attachment contains user guide for Geb framework which was used in the case study.

Keywords

Automated testing of web applications, Selenium, WebDriver, test automation tools, testing, MMSP methodology, Selenium WebDriver extensions.

Obsah

1 Úvod

Kvalitní testování aplikace lze považovat za jeden ze základních stavebních kamenů dobré aplikace. Správně fungující aplikace přináší spokojené zákazníky. S tím, jak se aplikace stávají stále složitějšími a komplexnějšími, stoupá i potřeba testování. Bohužel bývají testy velmi často přehlíženy, například z důvodu velké časové a zdrojové náročnosti. Vývojářské společnosti tak testování často omezují pouze na finální kontrolu aplikace v závěru projektu. Tento přístup k testování není velmi vhodný, neboť čím dřív je chyba objevena, tím snadnější a levnější je její oprava. Proto je daleko příhodnější testovat aplikaci po celou dobu jejího vývoji.

Testování je možné provádět manuálně nebo automatizovaně. Oba přístupy mají své výhody a nevýhody a vzájemně se tedy spíše doplňují, než vylučují. Výhodou manuálního testování prováděného testery je možnost zapojení lidské kreativity a intuice, díky čemuž je možné odhalit chyby, které by zůstaly pro automatizované testování skryté. Naproti tomu automatizované testování nepodléhá únavě a je možné ho daleko snadněji paralelizovat a využít tak například pro zátěžové testování.

Velkým boomem v současné době procházejí webové aplikace. Jedná se o aplikace, ke kterým je možné přistupovat prostřednictvím webového prohlížeče, a pro svůj běh tedy nepotřebují proprietárního klienta. I tyto aplikace je nutné testovat. Může docházet jak ke změnám v samotné aplikaci, tak i například v samotném webovém prohlížeči. Selenium je v současnosti jedním z nejpopulárnějších aplikačních rámců pro automatizované testování webových stránek a webových aplikací. Díky tomu existuje celá řada rozšíření a aplikačních rozhraní pro Selenium, která dále rozšiřují schopnosti tohoto nástroje.

Selenium a valná většina rozšíření dostupných pro tento nástroj je distribuována jako otevřený software. Cenou za bezplatnou licenci je nižší úroveň podpory a velmi často značně omezená dokumentace. V současné době totiž ve světě téměř neexistují ucelené návody a v češtině nejsou k tomuto tématu dostupné téměř žádné kvalitní zdroje. Z toho důvodu jsem se rozhodnul vytvořit přehled pokročilých možností automatizovaného testování nástrojem Selenium Webdriver.

1.1 Cíle a přínosy práce

Hlavním cílem práce je představit pokročilé možnosti práce s testovacím nástrojem Selenium WebDriver. Tento cíl je rozdělen do několika dílčích cílů. Prvním dílčím cílem je vytvoření přehledu pokročilých funkcí nástroje Selenium WebDriver. Dále si kladu za cíl vytvořit přehled dostupných aplikačních rámců, aplikací a služeb, které určitým způsobem rozšiřují funkce tohoto nástroje. Použití vybraných nástrojů je demonstrováno na případové studii ve firmě BellaDati. Cílem je ukázat reálné využití těchto nástrojů a také ověření možnosti aplikace rozšířené metodiky MMSP v reálném prostředí menšího podniku. Součástí této práce je také uživatelská příručka k aplikačnímu rámci Geb, který byl použit v rámci případové studie.

1.2 Cílová skupina

Tato práce je cílena na uživatele, kteří již mají základní zkušenost s testováním, vývojem webových stránek nebo programováním. Hlavní část práce je zaměřena na pokročilé možnosti testování pomocí nástroje Selenium a na testovací framework Geb. Je tedy určena pro čtenáře, kteří chtějí začít pracovat s tímto nástrojem.

1.3 Předpoklady a omezení

Tato práce se nezaměřuje na základy testování pomocí Selenia. Pro čtenáře teprve začínající s tímto nástrojem doporučuji přečíst si nejprve diplomovou práci Testování webových aplikací s využitím nástroje Selenium Webdriver od Lucie Třískové (Třísková 2015).

1.4 Struktura práce

V úvodu práce nejprve stručně definuji základní pojmy týkající se testování. Následně popisuji jednotlivé typy testování a zaměřuji se především na automatizované testování a jeho porovnání s manuálním testováním.

V další části je popsán nástroj Selenium a jeho pokročilé funkce, jmenovitě Selenium Grid a Selenium RemoteWebDriver. Součástí druhé části je také přehled aplikačních rámců, aplikací a služeb, jež jsou dostupné na trhu.

Další kapitola obsahuje případovou studii popisující reálně nasazení vybraných nástrojů v rámci firmy BellaDati, která se zabývá vývojem webové aplikace Business Intelligence.

Součástí případové studie je popis nástroje BellaDati, jeho využití a jednotlivé moduly, které používá. Zaměřuji se také na technickou stránku aplikace. V rámci případové studie je popsána aplikace rozšířené metodiky MMSP při zavádění automatizovaného testování.

V závěru práce je uvedena uživatelské příručka aplikačního rámce Geb. Ta ve strukturované podobě popisuje jednotlivé funkce tohoto nástroje. U každé funkce jsou uvedeny také reálné příklady testů, které byly vytvořeny pro případovou studii.

1.5 Očekávaný vlastní přínos

Hlavním přínosem této diplomové práce je vytvoření přehledu dostupných aplikačních rámců, aplikací a služeb pro nástroj Selenium. Takto ucelený přehled zatím není k dispozici. Dalším přínosem je popis aplikace vybraných nástrojů a rozšířené metodiky MMSP v reálním prostředí malé firmy. Posledním přínosem je vytvoření uživatelské příručky aplikačního rámce Geb.

2 Rešerše zdrojů

O automatizovaném testování byla napsána celá řada publikací, internetových zdrojů a akademických prací. Pouze malá část z nich se ale zabývá nástrojem Selenium WebDriver. Tématu pokročilých možností testování pomocí tohoto nástroje se věnuje jen velmi malé procento zdrojů. Tyto zdroje jsou navíc dostupné pouze v anglickém jazyce.

2.1 Odborné publikace

Vzhledem k velmi specifickému tématu existuje pouze několik publikací v angličtině, které se navíc obsahu práce dotýkají pouze částečně. Zpravidla se jedná o obecné popisy nástrojů Selenium WebDriver nebo Gradle, nástroje pro automatizaci sestavování aplikací. V češtině lze nalézt knihy týkající se automatizovaného testování nebo testování obecně:

1. **Testování softwaru**, autor Ron Patton, rok vydání 2002 (Patton 2002). Tuto knihu lze považovat za jednu ze základních publikací, kterou by měl znát každý, kdo se více věnuje testování. Kniha poskytuje slušný základ pro téměř všechny oblasti testování, ale žádné z nich se nevěnuje hlouběji. Je tedy vhodná především pro začátečníky a pro čtenáře, kteří potřebují získat obecný přehled v tématice testování.
2. **Building and Testing with Gradle**, autoři Tim Berglund a Matthew McCullough, rok vydání 2011 (Berglund a McCullough 2011). Tato kniha slouží jako komplexní průvodce sestavováním aplikací pomocí nástroje Gradle. V rámci této práce je nejpodstatnější pátá kapitola, která se věnuje testování. Uvádí příklady spolupráce s jednotlivými testovacími frameworky (JUnit, TestNG,Spock) a stručně se zmiňuje i o možnosti využití frameworku Geb.
3. **Selenium Testing Tools Cookbook**, autor Unmesh Gundecha, rok vydání 2012 (Gundecha 2012). Příručka Unmeshe Gundechy popisuje formou příkladů a návrhových vzorů možnosti testování pomocí nástroje Selenium WebDriver. Kniha je vhodné pro mírně až středně pokročilé uživatele používající Selenium, neboť obsahuje jak základní elementy práce s funkcí WebDriver (CSS, XPath, HTML prvky), tak i složitější příklady, jako například automatizace testování HTML5 prvků, výkonnostní testování apod.
4. **Instant Selenium Testing Tools Starter**, autor Unmesh Gundecha, rok vydání 2013 (Gundecha 2013). Tato publikace slouží jako tutoriál pro začínají uživatele nástroje

Selenium. Jedná se o obecný popis instalace a používání nástroje. Pro tuto práci je nejzajímavější konec knihy, kde jsou uvedeny odkazy na blogy a twitterové účty, které se zabývají nástrojem Selenium a testováním obecně.

5. **Selenium 2 Testing Tools: Beginner's Guide**, autor David Burns, rok vydání 2012 (Burns 2012). Tato kniha je pravděpodobně nejobsáhlejším obecným průvodcem nástrojem Selenium. Strukturou se podobá publikaci Selenium Testing Tools Cookbook. Na rozdíl od ní ale obsahuje více teoretického textu. Stejně jako ostatní publikace začíná popisem nástroje Selenium a možnostmi navigace v kódu. Zajímavější jsou kapitoly v druhé části knihy, především pak osmá kapitola zabývající se funkcí Selenium Grid, která je využita i v této práci. Pokročilé uživatele bude zajímat i devátá kapitola, ve které jsou popsány způsoby, jak simulovat některé méně standardní uživatelské interakce, jako například drag-and-drop nebo výběr více možností ze seznamu.
6. **Selenium WebDriver Practical Guide**, autor Satya Avasarala, rok vydání 2014 (Avasarala 2014). Tato publikace je obsahově i provedením velmi podobná předchozí knize. Opět se jedná o komplexního průvodce, který obsahuje jak základy, tak i pokročilé funkce.

2.2 Akademické práce

V době psaní práce se mi nepodařilo nalézt akademickou práci, která by se hlouběji zabývala pokročilými možnostmi testování prostřednictvím nástroje Selenium a jeho nástaveb. K dispozici byla ale řada prací, které popisují automatizované testování nebo Selenium obecně.

1. **Testování webových aplikací s využitím nástroje Selenium WebDriver**, autor Lucie Třísková, Vysoká škola ekonomická, Praha, 2015 (Třísková 2015). Tato diplomová práce byla napsána v roce 2015 na Vysoké škole ekonomické. Věnuje se automatizovanému testování a použití nástroje Selenium obecně. Jedná se o kvalitního českého průvodce, který obsahuje vše důležité pro začínající uživatele. Cílem mé práce je navázat na tuto práci a popsat pokročilé možnosti testování.
2. **Automatizace testování softwaru nástrojem Selenium**, autor Vojtěch Votlučka, 2015 (Votlučka 2015). Bakalářská práce Vojtěcha Votlučky popisuje základy testování pomocí nástroje Selenium. Představuje samotný nástroj a základní principy práce

s ním. Obsahuje také jednoduché příklady, díky čemuž je vhodná pro uživatele, kteří se nikdy s nástrojem Selenium nebo testováním nesetkali.

3. **Automatizované testování webových aplikací**, autor Michal Pietrik, Masarykova Univerzita, Praha, 2012 (Pietrik 2012). Hlavním tématem této diplomové práce je testování webových aplikací na platformě ASP .NET. V textu jsou popsány jednotlivé druhy testování, přičemž práce je zaměřena především na automatizované testování pomocí nástroje WebAii Framework. Část práce je ale také věnována nástroji Selenium.
4. **Využití automatizace testování z hlediska nákladů a přínosů**, autor Tomáš Lízner, Vysoká škola ekonomická, Praha, 2014 (Lízner 2014). Hlavním cílem této diplomové práce je vytvoření metodiky pro výběr testů vhodných pro automatizaci testování. V práci jsou uvedeny způsoby, jak definovat testy nevhodné pro automatizování či jak definovat priority jednotlivých testů. Velká část textu je věnována teorii automatizovaného testování. Samotná metodika výběru je velmi kvalitní a využitelná i v reálném prostředí.
5. **Automatické testování webových aplikací**, autor Martin Sokol, Masarykova univerzita. Brno, 2013 (Sokol 2013). Tato práce popisuje hlavní výhody a nevýhody automatického testování, přičemž se zaměřuje na testování uživatelského rozhraní webových aplikací. V praktické části pak popisuje reálné zavedení automatického testování v softwarové firmě oXy online s.r.o. Pro testování je využit balík aplikací Selenium a návrhový vzor Page Objects.
6. **Funkční testování webových aplikací**, autor Břetislav Mazoch, Masarykova Univerzita, Brno, rok 2012 (Mazoch 2012). Obsahem bakalářské práce Břetislava Mazocha je obecný popis metod a typů testování webových aplikací. Dále práce obsahuje výčet a specifikace jednotlivých nástrojů pro automatizaci testování (Selenium, TestNG, Ant, Jenkins) a porovnává výhody a nevýhody různých kombinací těchto nástrojů. V rámci celého textu je také popsán životní cyklu testování, od plánu testování, přes specifikace a tvorbu testů až po jejich spuštění a vyhodnocení.
7. **Testování webových aplikací za pomoci detekce změn jejího vzhledu**, autor Pavel Sklenář, rok 2012 (Sklenář 2012). Cílem této práce bylo vytvoření pomocné testovací knihovny, kterou lze využít k porovnání rozdílů dvou webových stránek. Knihovna

využívá nástroje Selenium a ImageMagick. Selenium je v tomto případě využito jiným způsobem než v případě mé práce.

8. **Testování softwaru v agilních projektech**, autor Rober Rojo, rok 2015 (Rojo 2015). Tato bakalářská práce popisuje proces testování softwaru v agilních projektech. Jedním z cílů práce je rozšíření metodiky MMSP o role, úlohy a pracovní produkty spojené především s automatizovaným testováním. Toto rozšíření bylo využito i v rámci případové studie této práce.

2.3 Internetové zdroje

Důležitým zdrojem je také Internet, neboť na něm lze nalézt kromě tutoriálů i předpřipravené příklady.

1. **Guru99 - Free Selenium Tutorials**, http://www.guru99.com/selenium-tutorial.html (Guru99 2015c). Velmi obsáhlý tutoriál, rozdělený na téměř 30 kapitol, který téměř kompletně pokrývá problematiku využití nástroje Selenium. Každá část se zaměřuje na jiné téma, od úplných základů až po velmi pokročilé příklady.
2. **The Book of Geb**, autoři Luke Daley, Marcin Erdmann, Erik Pragt, rok 2015, http://www.gebish.org/manual/current/ (Daley et al. 2015). Komplexní manuál pro Geb, popisující instalaci, konfiguraci a jednotlivé možnosti využití a použití nástroje Geb.

3 Základní pojmy

V úvodu této práce je vzhledem ke značné rozsáhlosti tématiky velmi důležité specifikovat základní pojmy a definice.

3.1 Test

Samotné slovo je možné dle Oxfordského slovníku definovat takto:

- proces, jehož cílem je zjištění kvality, výkonnosti nebo spolehlivosti;
- událost nebo situace, která má odhalit sílu nebo kvalitu pod zátěží (Oxford Dictionaries 2015).

3.2 Testování softwaru

V odborných publikacích a oficiálních dokumentech je možné nalézt celou řadu více či méně se lišících definicí. Dle standardu ISO/IEC/IEEE 29119 lze softwarové testování definovat jako proces analýzy softwaru, jehož cílem je zhodnocení vlastností softwaru a nalezení rozdílů mezi očekávaný a skutečným stavem (ANON. 2013).

Cem Kaner popisuje testování jako empirické technické zkoumání prováděné za cílem poskytnout informace o kvalitě testovaného produktu nebo služby všem zainteresovaným stranám (tzv. stakeholderům) (Kaner 2006).

Dle ISTQB je testování definováno jako proces skládající se ze všech aktivit životního cyklu, jak statických tak dynamických, který se zabývá plánováním, přípravou a hodnocením softwarových produktů. Cílem tohoto procesu je zjištění, zda tyto produkty splňují zadané požadavky, případně zda obsahují chyby (ISTQB Glossary Working Group 2015).

Všechny tyto definice lze shrnout do několika obecných bodů:

- cílem testování je odhalit chyby;
- chybu lze definovat jako odchylku skutečného stavu od očekávaného;
- testování má dále za úkol zhodnocení naplnění požadavků zainteresovaných stran.

Testování softwaru patří mezi takzvané „verifikační a validační“ softwarové praktiky, někdy psané zkráceně jako „V&V praktiky“. Kromě testování mezi ně patří například také inspekce nebo párové programování (Williams 2006). „*Verifikaci je možné definovat jako proces*

zhodnocení systému nebo komponenty za účelem zjištění, zda výsledky dané vývojové fáze splňují podmínky zadané na začátku fáze (IEEE 1990).“ Během verifikace odpovídáme na otázku „Vytváříme produkt správně?“ Validace je proces sloužící ze zhodnocení, zda vyvíjený systém odpovídá požadavkům, jež byly stanoveny zákazníkem na počátku vývoje. Tato kontrola je velmi důležitá, neboť zabraňuje vniku produktu, který ve skutečnosti nikdo nepotřebuje (Williams 2006). Validace odpovídá na otázku „Vytváříme správný produkt?“

3.3 Kategorizace testů

Oblast testování softwaru je velmi široká a existuje velké množství způsobů, jak testy kategorizovat. Přestože cílem této práce není věnovat se testování obecně, je nutné definovat alespoň základní typy rozdělení.

Rozdělení podle typu analýzy

Jedním z nejjednodušších způsobů, jak kategorizovat testování softwaru, je dle typu analýzy. Aplikaci je možné testovat staticky nebo dynamicky. Při statickém testování aplikace neběží a dotyčná osoba (tester nebo vývojář) si prohlíží kód aplikace (Patton 2002, s. 49). Zejména v počáteční fázi vývoje aplikace se může jednat o ekonomicky výhodnější způsob testování. Oprava chyby objevené během kontrolního meetingu je daleko levnější než oprava chyby nalezené během pozdějšího dynamického testování (DuPaul 2013). Výhodou je také možnost testovat aplikaci ještě před vznikem první spustitelné verze.

Na druhou stranu dynamické testování je daleko lépe využitelné při komplexnějším testování a je pomocí něho možné odhalit chyby, které by jinak během statického testování zůstaly skryté. Dynamické testování je prováděno vždy za běhu aplikace. Může se jednat o testování výkonnosti nebo hardwarové náročnosti (DuPaul 2013). Zejména se ale jedná o funkční testování, při kterém tester porovnává předpokládané a skutečné chování aplikace. (Patton 2002, s. 49)

Rozdělení podle viditelnosti kódu

Druhým nejběžnějším rozdělením testů je dle viditelnosti kódu. Jedná se o tzv. „testování bílé skříňky“ a testování „černé skříňky“. Při testování černé skříňky nemá tester přístup ke zdrojovým kódům. Ví pouze, jak se má testovaná aplikace chovat (Patton 2002, s. 48). Tester v tomto případě pouze porovnává zadané vstupy s předem očekávanými výstupy

bez znalosti přechodu mezi těmito objekty. Příkladem může být například akceptační nebo systémové testování (Jenkins 2008, s. 9).

Testování bílé skříňky je přesným opakem. V tomto případě má tester možnost nahlížet do zdrojových kódů aplikace. Díky tomu může lépe odhadnout problematické případy, případně snadněji identifikovat přesný původ problému (Patton 2002, s. 49). Nevýhodou této varianty testování je riziko tvorby testů „na míru aplikace“ tak, aby odpovídaly zdrojovému kódu. Dále je nutná alespoň základní znalost programovacích jazyků a programování obecně. Z toho důvodu provádějí testování bíle skříňky velmi často programátoři. Příkladem může být například jednotkové testování (Jenkins 2008, s. 9).

Na pomezí testování bílé a černé skříňky se nachází testování šedé skříňky. Stejně jako při testování bílé skříňky tester pracuje se specifikace požadavků. Navíc ale spolupracuje s vývojáři, především za účelem pochopení interní struktury systému. Má také možnost nahlížet do kódu aplikace. Díky tomu je pro testera snazší správně pochopit nejasnosti a vytvořit stoprocentně správné testy. William Lewis uvádí ve své publikaci jako příklad testování šedé skříňky situaci, kdy se tester domnívá, že stejná funkce je použita na několika místě najednou a teoreticky by mělo být možné ji otestovat pouze na jednom místě. Jde tedy za vývojářem, který mu může pomoci pochopit interní design a architekturu, díky čemuž se může tester správně rozhodnout (Lewis 2008, s. 41). Dalším příkladem může být testování webové aplikace, kdy tester má možnost prohlédnout si zdrojový kód webu (HTML, CSS a JavaScript), ale nemůže pracovat s programovým kódem aplikace (například jazyk PHP) (Patton 2002, s. 173).

Rozdělení na funkční a nefunkční testování

Funkční, nebo také funkcionální testování, se zaměřuje na funkční podstatu aplikace, neboli na otestování přechodu mezi vstupy a výstupy. Testy jsou vytvářený na základě požadavků a specifikace softwaru (Jorgensen 2013). Funkční testování je tedy velmi podobné testování černé skříňky.

Naproti tomu nefunkcionální testování je zaměřené na ostatní aspekty aplikace, jako je výkonnost, stabilita nebo bezpečnost. Netestuje se tedy naplnění funkcionálních požadavků, ale schopnost správně fungovat během reálného nasazení. Jedná se tedy o neméně důležitou část testování (Jorgensen 2013). Nefunkcionální testování vyžaduje kvantitativní a kvalitativní

metriky očekávaných výstupů, aby bylo možné změřit naplnění požadovaných výsledků. Výsledky těchto testů se mění spolu s konfigurací, množstvím vstupů a jejich podobou. Z toho důvodu se jedná o zdrojově velmi náročné testování (Desikan 2006, s. 132).

Do oblasti nefunkcionální testování je možné zařadit celou řadu testů. Kromě základních typů testů existuje ještě celá řada dalších, méně obvyklých typů, které se využívají pouze ve specifických případech:

- **Zátěžové testování** slouží k měření odezvy během maximální zátěže. *„Obecně lze zátěžové testy rozdělit do několika kategorií:*
    - *Zátěžový test (*angl*. Load Test),*
    - *Test hraniční zátěže (*angl*. Stress Test),*
    - *Test odolnosti (*angl*. Soak Test),*
    - *Test selhání (*angl*. Failover Test),*
    - *Test části infrastruktury (*angl*. Targeted Infrastructure Test),*
    - *Výkonnostní test (*angl*. Performance Test),*
    - *Test citlivosti sítě (*angl*. Network Sensitivity Test),*
    - *Test objemu dat (*angl*. Volume Test)* (Hlava 2012)*.“*
- **Souběžné testování** testuje schopnost aplikace pracovat při souběžném přístupu velkého množství uživatelů naráz.
- Cílem **globalizačního testování** je otestovat všechny druhy lokálního nastavení aplikace. Podobným typem je **internacionalizační** a **lokalizační testování**.
- **Testování obnovení** slouží k otestování zotavení aplikace po selhání aplikace, systému apod.
- **Testování spolehlivosti** zkoumá schopnost aplikace fungovat bez chyb a pádů.
- **Testování bezpečnosti** ověřuje zabezpečení aplikace a dat, která používá. Může se jednat například o testování způsobu ukládání citlivých dat, šifrování komunikace nebo kontrola správně funkčnosti přístupových práv.
- **Testování přenositelnosti** slouží k ověření možnosti přenosu aplikace z jedné platformy na jinou, například mezi operačním systémem Windows a Linux. Je blízké k **testování kompatibility**, které ověřuje funkčnost aplikace na různých systémech, například v případě webové aplikace kompatibilitu s různými webovými prohlížeči.

Příklady dalších typů je možné nalézt v článku 100 Types of Software Testing You Never Knew Existed na webu Guru99 (Guru99 2015a).

Rozdělení na automatizované a manuální testování

Software je možné testovat dvěma základními způsoby: manuálně a automatizovaně. Manuální testování je prováděno člověkem (testerem), který prochází testovací scénáře a ověřuje tak správnou funkčnost aplikace. Automatizované testování je prováděno určitým testovacím nástrojem, který vykonává testy, jež byly připraveny a nastaveny testovacím nebo vývojářským týmem. Velmi často je vhodné využít obou způsobů, neboť každý z nich má určité výhody a nevýhody. Detailní srovnání automatizovaného a manuálního testování se nachází v kapitole 4 Automatizované testování.

4 Automatizované testování

Tématem této kapitoly je automatizované testování. V první části je popsána jeho charakteristika a základní vlastnosti. V druhé části kapitole se uvedeno detailní porovnání automatizovaného testování s manuálním. V poslední části se nachází seznam nástrojů pro automatizované testování webových aplikací rozdělené to několika kategorií podle typu testů. Zaměřuji se především na nástroj Selenium.

4.1 Charakteristika automatizovaného testování

Je velmi těžké přesně definovat automatizované testování, neboť pro každého může tento pojem znamenat něco jiného. Může jít o jednotkové testy, testovací skripty vytvořené pomocí speciálního nástroje nebo zátěžové testy. Elfriede Dustinová (Dustin et al. 1999) uvádí ve své publikaci několik základních vlastností automatizovaného testování:

- **Rozšiřuje možnosti manuálního testování** o testy, které by bylo téměř nemožné provádět manuálně (např. zátěžové testování).
- Jedná se o **odnož softwarového vývoje**.
- **Nenahrazuje manuální testování**, schopnosti testera, znalosti testovacího know-how a testovacích technik.
- **Nelze ho zcela oddělit** od manuálního testování, oba typy testování se doplňují (Dustin et al. 1999).

Další vlastnosti, respektive hlavní výhody automatizovaného testování lze nalézt v knize Testování softwaru od Rona Pattonu (Patton 2002, s. 186):

- **Rychlost** – provedení automatizovaného testování je řádově rychlejší než manuální testování. Největší časovou úsporu lze získat u opakovaně vykonávaných testů.
- **Efektivita** – automatizované testování umožňuje testerovi věnovat se jiné práci, například optimalizaci testovacích scénářů, plánování testů apod. Šetří tak časové i lidské zdroje.
- **Správnost, přesnost a neúnavnost** – testovací stroj nepodléhá únavě, není náladový a pracuje vždy se stoprocentní efektivitou.

Obecně lze říci, že téměř jakýkoliv manuální test lze automatizovat. Automatizace přináší celou řadu výhod. Z vlastní zkušenosti vím, že manuální testování se po určité době stává velmi jednotvárné a pozornost a pečlivost testerů se snižuje. Stejně tak klesá výkonnost testerů s tím, jak roste jejich únava. Další výhodou je rychlost, která je u automatizovaného testování vyšší. Guru99 uvádí o 70 % (Guru99 2015b), ale toto číslo závisí na podobě testu. S vyšší rychlostí je spojená i úspora nákladů, která se ale projeví až po delší době. Nejvíce času a zdrojů je totiž potřeba na vytvoření těchto testů. Jejich vykonání již zpravidla bývá velmi rychlé, a tak se úspora s časem neustále zvyšuje. Specialitou automatizovaného testování je schopnost otestovat chování aplikace v extrémních případech, například při maximální zátěži, současném přístupu velkého množství uživatelů apod. Stejně tak je téměř nemožné manuálně zjistit případné úniky paměti[1] (Dustin et al. 1999). Může se zdát, že automatizace testování přináší jenom samé výhody a že manuální testování pozbývá významu. Realita je ale značně odlišná. Úprava testovacích skriptů je náročnější než pouhé přepsání scénářů. Především u dynamicky vyvíjených aplikací je potřeba neustále aktualizovat testovací systémy, což může být zdrojově velmi náročné (Karhu et al. 2009, s. 7).

4.1.1 Porovnání manuálního a automatizovaného testování

Jak je zřejmé z předchozí kapitoly, automatizované i manuální testování má své výhody i nevýhody. Jejich shrnutí je možné nalézt v tabulkách níže.

Tabulka 1: Výhody a nevýhody manuálního testování (zdroj: autor)

Manuální testování	
Výhody	**Nevýhody**
Adaptace na nečekané situace, snazší úprava testů v případě změny v aplikaci.	Opakované provádění stejných testů snižuje pozornost testera – náchylnost k chybám.
Pro přípravu testů není nutná znalost programování.	Některé testy nelze provádět manuálně – například zátěžové testy.
Tester může využít svoji intuici.	Závislost na stavu a únavě testera.
Nižší krátkodobé náklady.	
Hodí se pro testování UI a UX.	

[1] Memory leak

Tabulka 2: Výhody a nevýhody automatizovaného testování (zdroj: autor)

Automatizované testování	
Výhody	**Nevýhody**
Neúnavnost.	Příprava a údržba testů časově náročnější.
Přesnost a spolehlivost.	Testuje pouze to, na co je nastaveno. Úspěšný průběh automatizovaných testů ≠ aplikace bez chyb.
Rychlost vykonání, především u často opakovaných testů.	I minimální změna může způsobit chybu testu.
Snížení zátěže testerů.	Vyšší závislost na testovacích nástrojích.
Samotné testování mohou provádět i nezkušení testeři.	Příprava testů náročná na znalosti.
Usnadňují testování kompatibility – jednu sadu lze spustit ve více prostředích.	Ne vše lze automatizovat.
Nižší dlouhodobé náklady.	Nevhodné pro agilní testování (angl. *exploratory testing*).
Možnost paralelizace provádění testů.	
Skvěle se hodí pro zátěžové testy.	

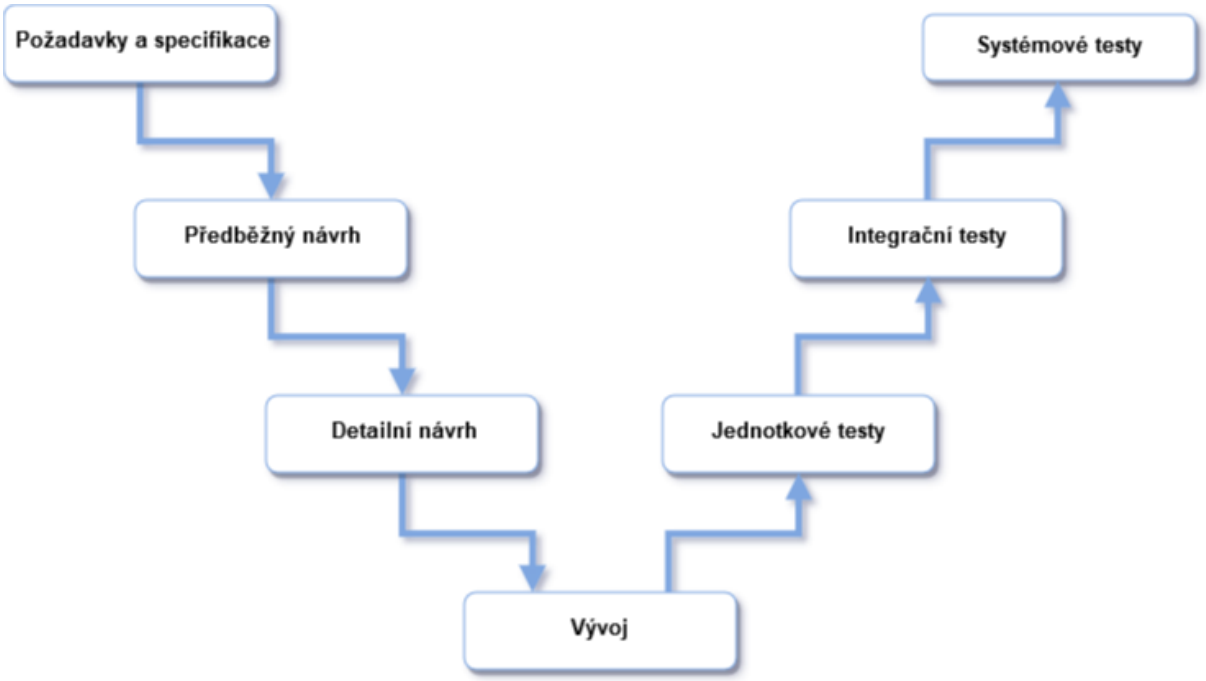
4.2 Nástroje pro testování webových aplikací

V současné době lze nalézt na trhu velké množství různorodých testovacích nástrojů, frameworků apod. K dispozici jsou jak placené, tak i bezplatné nástroje. Tyto nástroje je možné rozdělit dle několika kritérií. Velmi detailní kategorizaci lze nalézt v práci Lucie Třískové *Testování webových aplikací s využitím nástroje Selenium WebDriver*. Z toho důvodu v práci uvedu pouze jeden typ rozdělení s tím, že další typy lze nalézt v kapitole ***3.2 Nástroje pro automatizované testování webových aplikací*** zmíněné práce (Třísková 2015, s. 32).

4.2.1 Rozdělení nástrojů pro testování aplikací podle úrovně testování

Jedná se o základní a tradiční typ rozdělení testů, který odpovídá vodopádovému modelu. To znamená, že jednotlivé typy testů na sebe navazují a pro každou úroveň vývoje existuje i typ modelu, viz Obrázek 1. Očekává se tedy, že systémové testy budou v souladu s požadovanou

specifikací a že jednotkové testy budou odvozeny od detailního návrhu (Jorgensen 2013, s. 181).



Obrázek 1: Schéma pořadí testů (zdroj (Jorgensen 2013), přeloženo autorem)

Nástroje pro jednotkové testování

Jednotkové testování se používá především pro otestování funkcionality jednotlivých částí kódů – tříd, metod a procedur. Slouží k automatizovanému a opakovatelnému testování a zajišťuje kontrolu funkčnosti aplikace na úrovni kódu po každé změně. Je tedy vhodné především pro dlouhodobou údržbu kódu. Vytváření a úpravu těchto testů mají na starost vývojáři. Nástroje mají většinou podobu frameworku, který je specializován na určitý programovací jazyk. Mezi nástroje pro jednotkové testování patří například:

- **Java** – JUnit, Mockito, TestNG, Jtest;
- **C++** - CppUnit, Boost.Test, Unit++, CxxTest;
- **C#** - NUnit, xUnit, Testdrive.Net, Moq.

Nástroje pro integrační testování

Jak vyplývá z názvu, integrační testy slouží k ověření komunikace a spolupráce mezi jednotlivými komponentami aplikace. Z toho důvodu jsou tyto aplikace důležité především u velkých projektů, které obsahují desítky komponent. U malých projektů je možné tyto testy dokonce vynechat. Mezi nástroje pro integrační testování lze zařadit:

- VectorCAST, LDRA TBrun, Citrus (zdarma), TestNG (zdarma) a další.

Nástroje pro systémové testy

Systémové testy slouží k otestování aplikace jako celku. Kontroluje se, zda aplikace odpovídá požadavkům, které zadal zákazník. Proto přichází na řadu především v pozdějších fázích vývoje, kdy už je větší část aplikace hotová, nebo když už je aplikace dokončena. Jedná se o poslední typ testů, který se provádí před předáním hotového produktu. Je tedy potřeba ověřit bezporuchovost a kompletní funkčnost aplikace. Příkladem nástrojů pro systémové testy jsou:

- IBM Rational Functional Tester, Selenium IDE (zdarma), Selenium Webdriver a jeho nástavby (zdarma), TestPlant eggPlant, Capybara, Watir (zdarma), Windmill, SmartBear TestComplete a další.

Nástroje pro akceptační testování

Akceptační testování je speciální typem testování, které stojí mimo schéma uvedené výše. Akceptační testování je prováděno zákazníkem po předání produktu a zpravidla je uskutečňováno na testovacím prostředí u zákazníka. Akceptační testování probíhá většinou manuálně, neboť není potřeba spouštět ho opakovaně. Z toho důvodu se nástroje pro akceptační testování zaměřují na usnadnění této činnosti.

- TestPlant eggPlant, Cucumber, FitNesse (zdarma), Robot Framework (zdarma).

Doteď se práce věnovala automatizovanému testování obecně. V dalších kapitolách se již zaměřím konkrétně na autorizované testování webových stránek a aplikací. Jedná se o hlavní téma práce, a proto jsou detaily ostatních typů testování pro tuto práci irelevantní.

4.3 Selenium

Selenium je nástroj pro testování webových aplikací. Je možné ho tedy zařadit mezi nástroje pro funkční testování. V současnosti se jedná o balík aplikací, které plní různé úkoly. Původně se ale jednalo o daleko jednodušší nástroj. Jeho kořeny lze nalézt v roce 2004 ve firmě ThoughtWorks v Chicagu, kde Jason Huggins, Paul Gross a Jie Tine Wang vytvořili nástroj JavaScriptTestRunner, který sloužil k testování interní aplikace pro sledování času a výdajů (Selenium 2015b).

Krátce poté byl nástroj uveřejněn jako open-source a později přejmenován na Selenium Core (Guru99 2015d). Největším problémem tohoto nástroje byla tzv. zásada stejného původu.[2] Jedná se o jednu ze základních bezpečnostních vlastností jazyku JavaScript (a webu obecně), která neumožňuje přistupovat k elementům z jiné domény, webové stránky nebo protokolu (Gosselin 2010). Kvůli této zásadě museli testeři používat lokální kopii nástroje Selenium Core a webového serveru, který chtěli testovat. Proto se Paul Hammant, také zaměstnanec firmy ThoughtWorks, rozhodl vytvořit serverovou aplikaci, která by fungovala jako HTTP proxy brána. Díky ní se webový prohlížeč domnívá, že webová aplikace je testována ze stejné domény, ačkoliv ve skutečnosti může být Selenium umístěno na zcela jiném serveru. Tento nástroj dostal později název Selenium Remote Control neboli Selenium 1 (Guru99 2015d).

Dalším krokem bylo vytvoření nástroje Selenium Grid, za kterým stál Patrick Lightbody. Systém se původně nazýval Hosted QA a sloužil k vytváření snímků obrazovky a zasílání Selenium příkazů na několik stanic zároveň (Guru99 2015d).

V roce 2006 přišel Shinya Kasatani s nápadem vytvořit pro Firefox Selenium IDE modul, který by umožňoval nahrávat a spouštět testy z jednoho uživatelsky přívětivého prostředí. Ve stejném roce modul věnoval organizaci spravující projekt Selenium (Selenium 2015b).

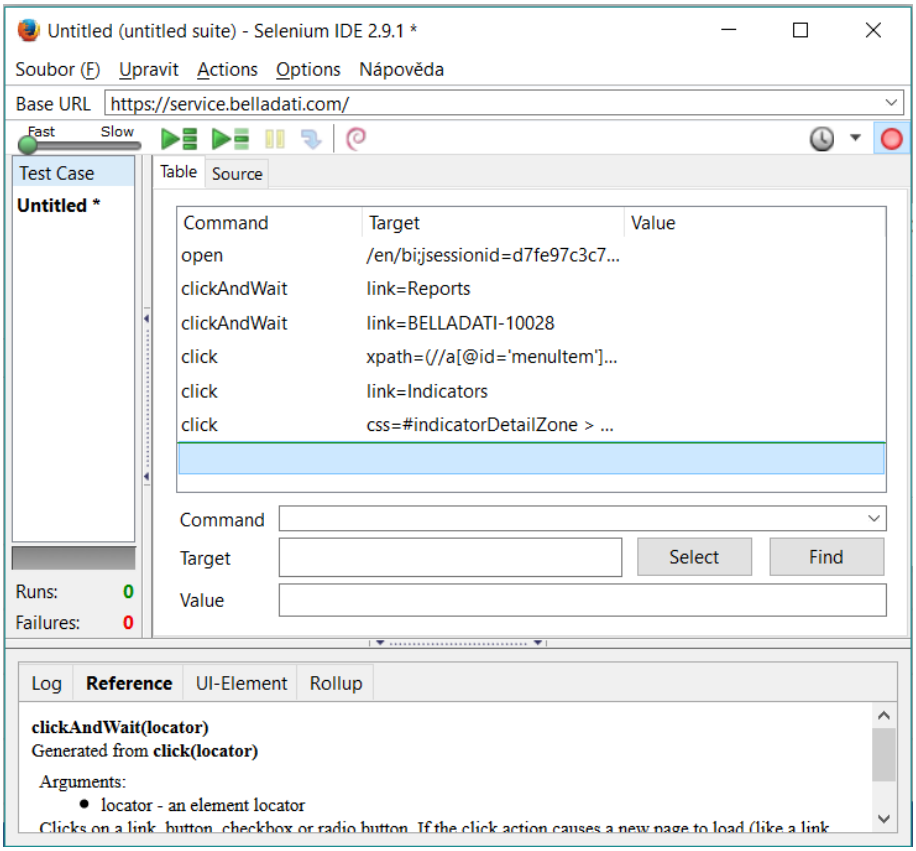
Zároveň s vývojem Selenia vznikal v ThoughtWorks další framework pro automatizaci webových prohlížečů. První verze, za kterou stál Simon Stewart**,** byla pod názvem WebDriver zveřejněna na začátku roku 2007.

Na trhu tak byly k dispozici dva velmi podobné produkty, každý se svými výhodami a nevýhodami. Největším rozdílem byl způsob implementace. Metody nástroje Selenium RC byly dostupné pohromadě v jedné metodě, zatímco WebDriver používal přehlednější objektově orientované aplikační rozhraní. Naproti tomu velkou výhodou nástroje Selenium RC byla podpora celé řady programovacích jazyků, WebDriver bylo možné používat pouze spolu s jazykem Java. Zjevným řešením by bylo oba nástroje spojit a vyvíjet tak jeden silnější a univerzálnější nástroj. A opravdu se tak stalo v srpnu roku 2009. Vznikl tak nástroj Selenium WebDriver, zvaný také jako Selenium 2 (Stewart 2010). První finální veřejná verze byla publikována téměř o dva roky později, přesněji 18. července 2011 (David 2016). Součástí

[2] Same-origin policy

nástroje Selenium 2 byl od hned od počátku i Selenium Server, včetně podpory funkce Grid pro distribuované testování (viz kapitola *5.2 Selenium Grid*).

V současné době je možné Selenium rozdělit na dvě základní části: Selenium IDE a Selenium WebDriver. Selenium IDE je doplněk pro webový prohlížeč Mozilla Firefox, který umožňuje jednoduše nahrávat a spouštět testovací skripty. Doplněk obsahuje grafické rozhraní pro správu skriptů. Tyto skripty je možné používat například pro zaznamenání způsobu reprodukce chyby nebo jednoduché otestování webové aplikace. Testy lze vytvářet nahráváním uživatelských interakcí na webové stránce. Selenium IDE zaznamenává všechna kliknutí, práci s webovým prohlížečem a formuláři umístěnými na stránce. Testy je možné následně ručně upravovat, například přidat vlastní příkazy nebo základní ověřování, například názvu stránky (Selenium 2015c).



Obrázek 2: Rozhraní doplňku Selenium IDE (zdroj: autor)

Druhou částí, pro tuto práci daleko významnější, je Selenium WebDriver. Jedná se o univerzální nástroj pro testování webových aplikací. Pro komunikaci s webovým prohlížečem je k dispozici speciální aplikační rozhraní, které využívá nativních podpory pro automatizaci v rámci jednotlivých procesorů. Každý prohlížeč má svůj vlastní „driver“, který zpracovává

příkazy a ovládá prohlížeč. Díky tomu je možné spouštět stejné testy na celé řadě prohlížečů (Selenium 2015d). V současné době WebDriver podporuje prohlížeče Internet Explorer, Google Chrome, Opera, Mozilla Firefox a Safari. Dále je k dispozici ovladač pro mobilní prohlížeč na mobilním operačním systému Android.

Velkou výhodou nástroje Selenium WebDriver je podpora velkého množství programovacích jazyků, ať už nativně nebo prostřednictvím doplňků třetích stran. Organizace samotná poskytuje balíčky pro jazyky Java, C#, Ruby, Python a JavaScript. Doplňky třetích stran přidávají podporu například pro jazyky PHP nebo Perl (Selenium 2015a).

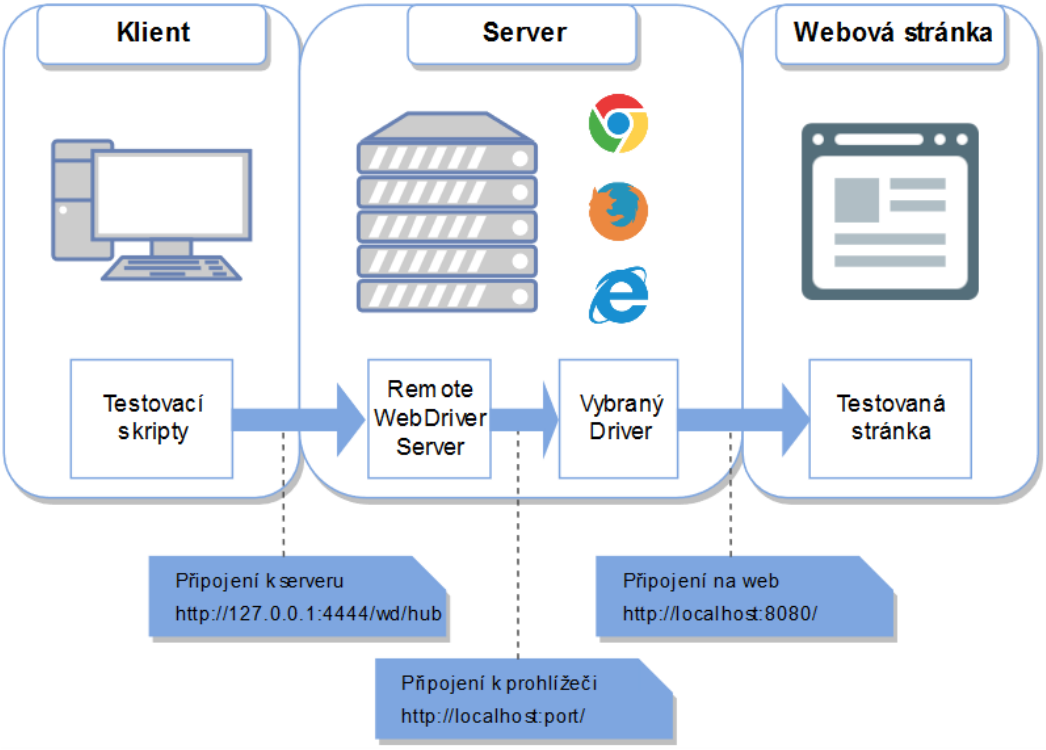
Pro kompletní popis funkčnosti nástroje Selenium WebDriver opět doporučím diplomovou práci Lucie Třískové *Testování webových aplikací s využitím nástroje Selenium WebDriver* (Třísková 2015), která obsahuje velmi detailní uživatelskou příručku. Příručka je zaměřená především na základy práce s nástrojem WebDriver. V dalších částech této práce naváži na zmíněnou příručku a uvedu příklady, jak rozšířit možnosti tohoto nástroje. Nejprve se zaměřuji na pokročilé funkce, které jsou poskytovány jako součást nástroje Selenium WebDriver. Jedná se o funkce, které umožňují vytvářet pokročilou infrastrukturu testovacích strojů: Selenium RemoteWebDriver a Selenium Grid. V následující kapitole popíši dostupné nástroje a frameworky, které přidávají do nástroje Selenium WebDriver dodatečné funkce. A konečně v další kapitole uvedu přehled služeb, které umožňují spouštět testy na vzdálených serverech.

5 Pokročilé funkce nástroje Selenium WebDriver

Velkou výhodou nástroje Selenium je pokrytí několika úrovní složitosti testování. Díky Selenium IDE je možné vytvářet jednoduché testy prostým klikáním bez nutnosti znalosti programování. Druhou možností je použít Selenium WebDriver a vytvořit si vlastní testy prostřednictvím jednoho z mnoha podporovaných programovacích jazyků. Na nejvyšší úrovni stojí vytváření a spouštění testů vzdáleně za využití serveru a jednoho nebo několika klientů zároveň. Tyto pokročilé možnosti jsou dostupné díky funkci RemoveWebDriver.

5.1 Selenium RemoteWebDriver

RemoteWebDriver je jednou z pokročilých implementací základní verze nástroje WebDriver, která umožňuje spouštět testy vzdáleně na jiném stroji. Je poskytován přímo jako součást standardního archivu. Skládá se dvou základních částí – serveru a klienta. Testovací skripty jsou umístěny na klientské části a připojují se k serveru, na kterém jsou nainstalovány jednotlivé webové prohlížeče (Obrázek 3).

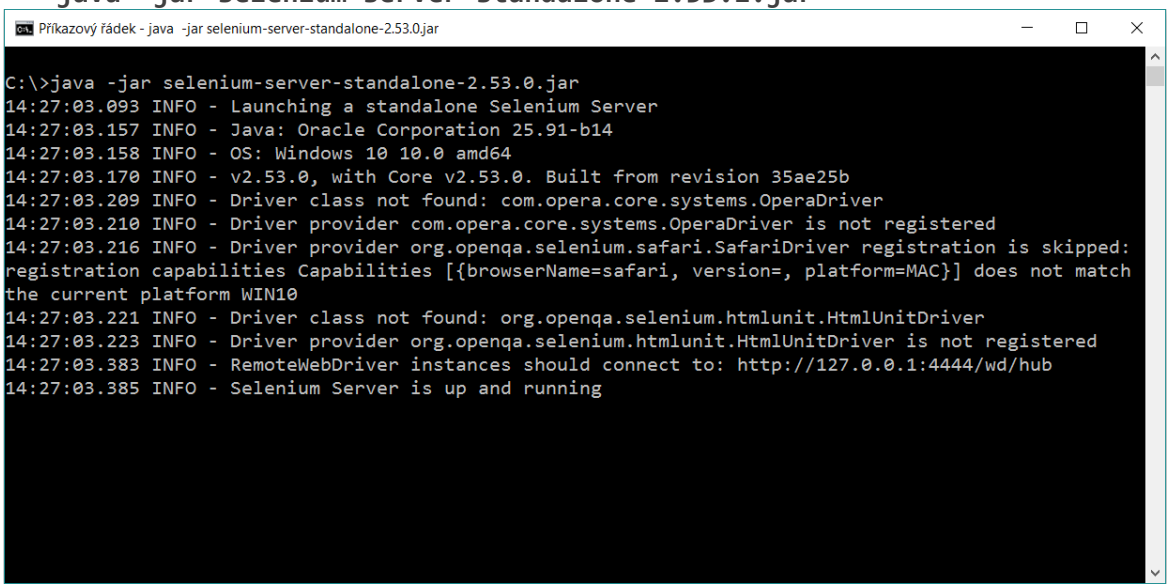


Obrázek 3: Diagram funkce RemoteWebDriver (zdroj: autor)

5.1.1 Spuštění serveru

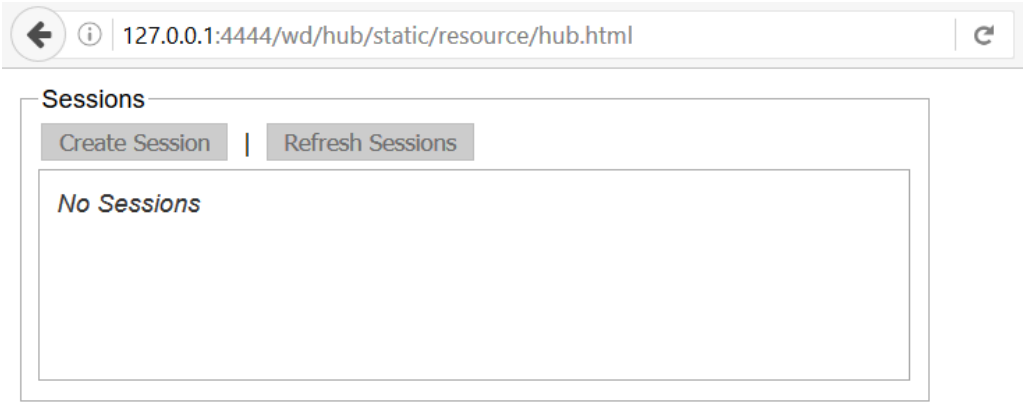
Server je distribuován v podobě souboru ve formátu *.jar* a je k dispozici ke stažení na oficiálních stránkách.[3] Pokud není stanoveno jinak, doporučuji vždy využívat poslední dostupnou verzi. V době psaní této práce se jednalo o verzi 2.53.0. Server je možné spustit prostřednictvím příkazové řádky (Obrázek 4):

```
java -jar selenium-server-standalone-2.53.1.jar
```



Obrázek 4: Spuštění serveru RemoteWebDriver (zdroj:autor)

Server po spuštění poslouchá na portu 4444 a čeká na vzdálené připojení od klienta. Funkčnost serveru je možné ověřit otevřením adresy *127.0.0.1:4444/wd/hub*, kde je možné nalézt také seznam všech aktuálně vytvořených relací (Obrázek 5).[4]



Obrázek 5: Webové rozhraní serveru nástroje RemoteWebDriver (zdroj: autor)

[3] http://www.seleniumhq.org/download/
[4] Sessions

5.1.2 Klientská část funkce Selenium RemoteWebDriver

Při lokálním spouštění testů bez využití funkce RemoteWebDriver komunikují knihovny nástroje Selenium WebDriver přímo s ovladačem zvoleného prohlížeče. V případě vzdáleného spouštění se použije mezivrstva, která zprostředkovává komunikaci mezi klientem a vzdáleným prohlížečem. Pro komunikaci se používá „JSON Wire protocol“ obsahující http příkazy.

5.1.3 Základní příklad funkce Selenium RemoteWebDriver

Pro demonstraci použití funkce RemoteWebDriver použiji zcela základní test, který otevře přihlašovací stránku ve Firefoxu, vyplní přihlašovací údaje a přihlásí uživatele. Následně vyhledá a klikne na odkaz pro otevření seznamu reportů. Zde porovná očekávaný a skutečný titulek stránky a podle výsledku vypíše do konzole „Test splněn“ nebo „Test nesplněn“.

```
1.  import java.net.MalformedURLException;
2.  import java.net.URL;
3.  import org.openqa.selenium.By;
4.  import org.openqa.selenium.WebDriver;
5.  import org.openqa.selenium.remote.DesiredCapabilities;
6.  import org.openqa.selenium.remote.RemoteWebDriver;
7.      public class myclass {
8.
9.      public static void main(String[] args) throws MalformedURLException {
10.         // základní nastavení funkce RemoveWebDriver
11.         DesiredCapabilities capability = DesiredCapabilities.firefox();
12.         WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub
    "), capability);
13.         String baseUrl = "http://localhost:8080";
14.         String expectedTitle = "Vyhledávání... - D2";
15.         String actualTitle = "";
16.
17.         // spuštění prohlížeče a vykonání příkazů
18.         driver.get(baseUrl);
19.         driver.findElement(By.name("login")).sendKeys("u2");
20.         driver.findElement(By.name("password")).sendKeys("Support01");
21.         driver.findElement(By.name("submit_0")).submit();
22.         driver.findElement(By.linkText("Reporty")).click();
23.         // načtení titulku stránky
24.         actualTitle = driver.getTitle();
25.
26.         // porovnání očekávaného a skutečného titulku stránky
27.         if (actualTitle.contentEquals(expectedTitle)){
28.             System.out.println("Test splněn");
29.         } else {
30.             System.out.println("Test nesplněn");}
31.         //ukončení
32.         driver.close();
33.          System.exit(0);
34.     }
35.
36. }
```

Výpis kódu 1: Základní test pomocí funkce RemoteWebDriver

Aby bylo možné pracovat s nástrojem Selenium WebDriver, je nutné ho připojit jako knihovnu k projektu a následně ho naimportovat. Pro správnou funkčnost je potřeba připojit serverovou verzi (selenium-server-standalone-2.53.0.jar). V případě připojení standardní verze se po spuštění testu vypíše chybová hláška:

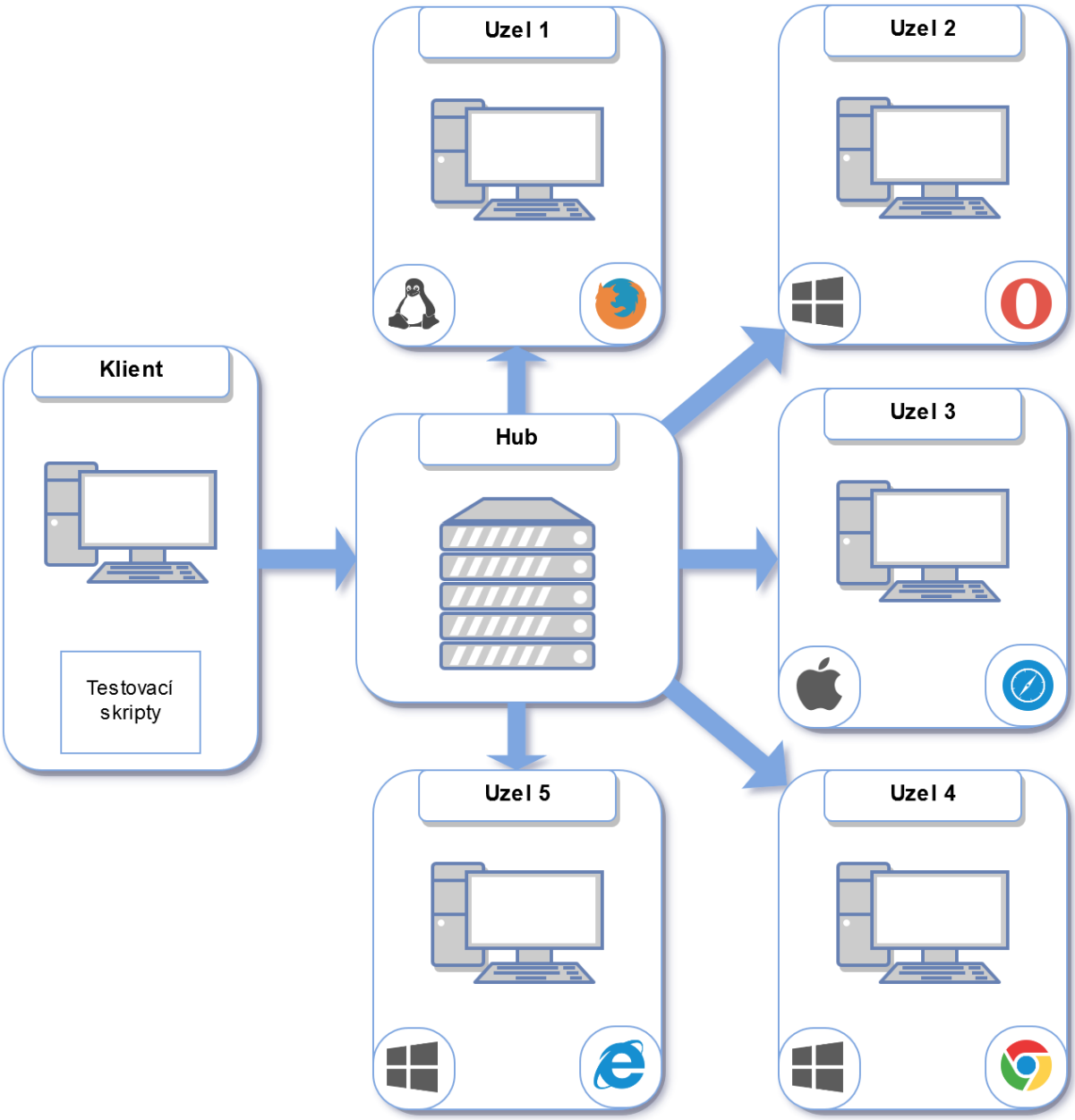
```
Exception in thread "main" java.lang.NoClassDefFoundError:
com/google/common/base/Function
 at myclass.main(myclass.java:17)
Caused by: java.lang.ClassNotFoundException:
com.google.common.base.Function
 at java.net.URLClassLoader.findClass(Unknown Source)
 at java.lang.ClassLoader.loadClass(Unknown Source)
 at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
 at java.lang.ClassLoader.loadClass(Unknown Source)
 ... 1 more.
```

5.1.4 Hlavní přínosy využití funkce Selenium RemoteWebDriver

Hlavním přínosem funkce Selenium RemoteWebDriver je možnost spouštět testy vzdáleně na serveru. Testy tak mohou běžet například na firemním serveru v době, kdy není jinak využíván, kupříkladu během noci. Není tak potřeba pořizovat zvláštní server na automatizované testování, což přináší úsporu firemních nákladů.

5.2 Selenium Grid

Selenium Grid je možné považovat za nadstavbu funkce RemoteWebDriver. Na rozdíl od ní se nejedná o jednoduchou strukturu klient-server, nýbrž o strukturu klient-hub-uzel (angl. *node*). Samotné testování probíhá na uzlu, hub slouží ke komunikaci mezi uzly a klientskou stanicí (viz Obrázek 6). Hub udržuje seznam všech připojených uzlů a jejich schopností (angl. *Capabilities*). O každém uzlu ví, jaký je jeho operační systém a jaké webové prohlížeče jsou na něm nainstalované. Spolu s každým testem dostane také seznam požadovaných schopností (angl. *DesiredCapabilities*) a na základě těchto požadavků přidělí test odpovídajícímu uzlu.



Obrázek 6:Diagram funkce Selenium Grid (zdroj: autor)

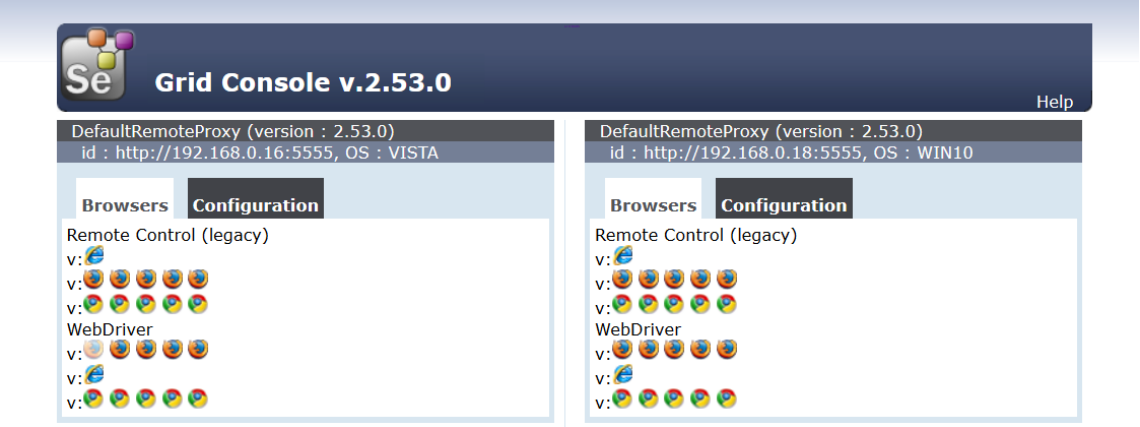
5.2.1 Spuštění hubu funkce Selenium Grid

Pro spuštění serveru lze použít stejný soubor, který byl využit pro spuštění serveru RemoteWebDriver. Jediným rozdílem je změna příkazu pro spuštění, ve kterém je potřeba specifikovat roli. V tomto případě se jedná o roli hubu:

```
java -jar selenium-server-standalone-2.53.0.jar -role hub
```

Po úspěšném spuštění server opět naslouchá na portu 4444. Stav serveru je možné monitorovat z konzole, která je dostupná na URL adrese *127.0.0.1:4444/grid/console*. V konzoli lze kromě jiného nalézt seznam připojených uzlů včetně jejich schopností. U každého

uzlu je uvedena jeho IP adresa, operační systém a typ a počet dostupných instancí webových prohlížečů. Zároveň je možné vidět, na kterých uzlech jsou momentálně aktivní relace. Aktuálně obsazený prohlížeč je označen poloprůhlednou ikonkou (viz Obrázek 7).



Obrázek 7: Konzole funkce Grid s dvěma připojenými uzly (zdroj: autor)

5.2.2 Klientská část funkce Selenium Grid

Velmi podobně se spouští také jednotlivé uzly. Opět lze využít stejný soubor jako v předchozích případech. Kromě role je ale nyní nutné určit adresu a port hubu, ke kterému se má uzel připojit.

```
java -jar selenium-server-standalone-2.53.0.jar -role node -hub
http://192.168.0.12:4444/grid/register
```

5.2.3 Volitelné konfigurační parametry pro Selenium Grid

V příkazu pro spuštění uzlu a hubu je možné použít volitelné parametry, kterými lze uzel konfigurovat (Davison 2015):

- **Port** – umožňuje specifikovat port, na kterém má být hub nebo uzel spuštěn.
    - `java -jar selenium-server-standalone-2.53.0.jar -role hub -port 5555`
    - `java -jar selenium-server-standalone-2.53.0.jar -role node -port 5554 -hub http://192.168.0.12:5555/grid/register`
- **Host** – umožňuje specifikovat IP adresu nebo *hostname*, na kterém bude hub nebo uzel *spuštěn*. Zpravidla není potřeba, používá se například při využití VPN.
    - `java -jar selenium-server-standalone-2.53.0.jar -role hub -host "host.name"`
- **Timeout** – tímto parametrem lze definovat dobu, po kterou má hub držet uzel. Pokud za tuto dobu uzel neobdrží žádný požadavek, je uvolněn a k dispozici pro další test

v řadě. Doba se uvádí v sekundách a ne výchozím stavu má hodnotu 300 sekund. Pokud je nastavena na nulu, node není uvolněn nikdy.

    - `java -jar selenium-server-standalone-2.53.0.jar -role hub -timeout 0`

- **MaxSession** – parametr, kterým lze nastavit maximální množství prohlížečů, které lze najednou spustit na uzlu.

    - `java -jar selenium-server-standalone-2.53.0.jar -role node -maxSession 4 -hub http://192.168.0.12:4444/grid/register`

- **Browser** – tento parametr umožňuje vybrat prohlížeče, které budou dostupné na uzlu. Parametr je možné použít v příkazu několikrát a je nutné ho doplnit dodatečnými parametry.
    - **BrowserName** – volba prohlížeče, který bude dostupný na uzlu.
        - `browserName={android, chrome, firefox, htmlunit, internet explorer, iphone, opera}`
    - **Version** – určuje verzi prohlížeče.
    - **Firefox_binary, Chrome_binary** – umožňuje specifikovat cestu ke spustitelnému souboru prohlížeče Mozilla Firefox, respektive Google Chrome.
    - **MaxInstances** – definuje maximální množství instancí daného prohlížeče.
    - **Platform** – určení platformy operačního systému.
        - `Platform={Windows, Linux Mac}`
    - Příklad parametru browser:
        - `java -jar selenium-server-standalone-2.53.0.jar -role node -browser "browserName=firefox platform=Windows maxInstances=2" -hub http://192.168.0.12:4444/grid/register`
- **RegisterCycle** – umožňuje nastavit, jak často se bude uzel pokoušet znovu zaregistrovat na hubu. Umožňuje restartovat hub bez nutnosti restartovat uzly.

    - `java -jar selenium-server-standalone-2.53.0.jar -role node -registerCycle 10000 -hub http://192.168.0.12:4444/grid/register`

K dispozici je i celá řada méně obvyklých parametrů. V případě velmi specifické konfigurace je vhodnější využít konfiguračního souboru.

5.2.4 Konfigurační soubory pro Selenium Grid

Jak je patrné z předchozí kapitoly, pro spuštění hubu nebo uzlu je možné použít celou řadu parametrů, které je samozřejmě možné kombinovat. Při použití velkého množství parametrů se stává spouštěcí příkaz velmi nepřehledný a těžko udržovatelný. V tomto případě je lepší volbou použití konfiguračního souboru.

Pro konfiguraci se používá soubor ve formátu JSON. Struktura souboru je rozdílná pro hub a pro uzel. Pro uzel se využívá jednoduchý seznam parametrů, kdežto v případě uzlu je konfigurace rozdělena na dvě části: schopnosti (angl. *capabitilies)* a konfigurace (angl. *configuration)*. Schopnosti slouží k nastavení dostupných webových prohlížečů a funguje tedy stejně jako parametr *browser*. Konfigurace obsahuje všechny ostatní parametry a slouží k nastavení samotného uzlu (viz Výpis kódu 2).

```
1.  {
2.    "capabilities":
3.        [
4.          {
5.            "browserName": "firefox",
6.            "maxInstances": 5,
7.            "seleniumProtocol": "WebDriver"
8.          },
9.          {
10.           "browserName": "chrome",
11.           "maxInstances": 5,
12.           "seleniumProtocol": "WebDriver"
13.         },
14.         {
15.           "platform": "WINDOWS",
16.           "browserName": "internet explorer",
17.           "maxInstances": 1,
18.           "seleniumProtocol": "WebDriver"
19.         }
20.       ],
21.   "configuration":
22.   {
23.     "proxy": "org.openqa.grid.selenium.proxy.DefaultRemoteProxy",
24.     "maxSession": 5,
25.     "port": 5555,
26.     "register": true,
27.     "registerCycle": 5000,
28.     "hub": "http://localhost:4444"
29.   }
```

Výpis kódu 2: Příklad konfiguračního souboru pro uzel (zdroj: Inman-Semerau 2016)

Při spouštění je potřeba uvést název a cestu ke konfiguračnímu souboru prostřednictvím parametru *nodeConfig*, respektive *hubConfig*:

```
java -jar selenium-server-standalone-2.53.0.jar -role node -hub
http://192.168.0.12:4444/grid/register -nodeConfig node.json
```

5.2.5 Hlavní přínosy využití funkce Selenium Grid

Hlavní výhodou využití nástroje Selenium Grid je možnost paralelizace testovaní. Díky možnosti spustit testy na několika různých platformách zároveň lze zkrátit potřebnou dobu pro testování. Druhou možností je spustit testy na několika uzlech se stejnými parametry. Je tak možné ve stejném čase provést ten samý test několikrát a značně tak omezit prvek náhody. V případě velkého množství zároveň spuštěných uzlů lze Grid využít i pro jednodušší zátěžové testování.

6 Aplikační rámce a nástroje rozšiřující Selenium WebDriver

Cílem této kapitoly je vytvořit přehled dostupných aplikačních rámců, aplikací a služeb, které určitým způsobem rozšiřují funkce tohoto nástroje. Tato rozšíření jsou rozdělena do několika kategorií dle jejich typu:

- aplikační rámce pro nástroj Selenium WebDriver,
- softwarové kontejnery,
- nástroji pro automatizaci a průběžnou integraci,
- služby pro nástroj Selenium WebDriver.

Smyslem těchto nástrojů a služeb je usnadnění práce s nástrojem Selenium WebDriver, případně rozšíření jeho funkčnosti. Velkou výhodou je také vzájemná kompatibilita jednotlivých typů rozšíření díky tomu, že v jádru se používá jednotné aplikační rozhraní Selenium WebDriver API. Konkrétní příklady kooperace jednotlivých rozšíření lze zpravidla nalézt v dokumentaci daných produktů.

6.1 Přehled aplikačních rámců pro nástroj Selenium WebDriver

Na trhu je k dispozici celá řada rozšíření pro Selenium WebDriver. Největší část tvoří frameworky, které využívají aplikační rozhraní, jež poskytuje Selenium WebDriver. K dispozici jsou jak bezplatná, tak i komerční řešení. Použití těchto aplikačních rámců přináší řadu výhod. Ač každý z nich funguje trochu jinak, lze jejich přínosy shrnout do několika nejdůležitějších bodů:

- podpora chytrého čekání na elementy, velmi důležitá pro asynchronně fungující stránky,
- zjednodušení zápisu testů díky doménově specifického jazyku, předpřipraveným metodám a funkcím,
- zlepšená správa sezení (angl. *session management*).

Cílem této práce není porovnání dostupných rozšíření, ale vytvoření jejich přehledu. Tato kapitola může posloužit jako základ pro budoucí práce, které by se porovnání těchto nástrojů věnovaly. V následujících podkapitolách jsou popsány tyto aplikační rámce:

- Capybara,
- Oxygen,
- Watir,
- Vapir,
- Nightwatch,
- Satix,
- Magium,
- Redwood,
- Sahi,
- SeLion,
- Webdriver.io,
- Chimp,
- CodeceptJS,
- Codeception,
- Selenide,
- Qualitia,
- Geb.

6.1.1 Capybara

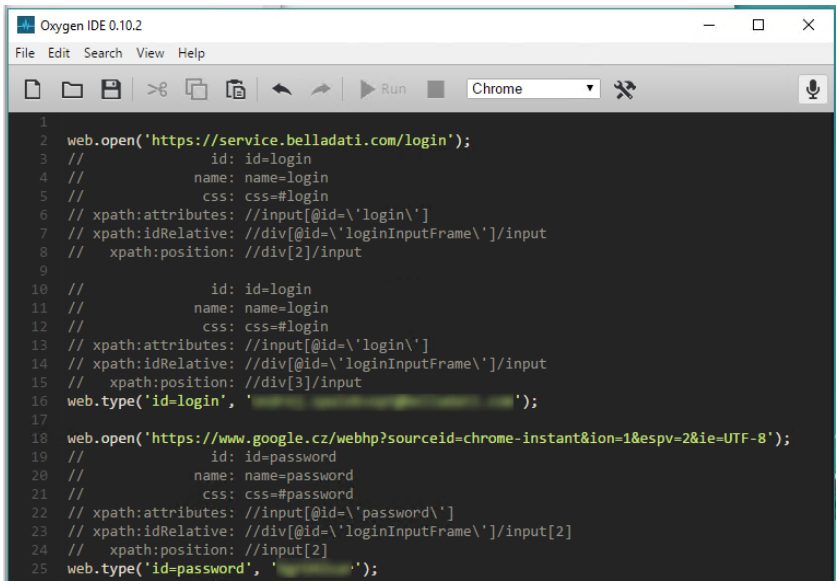
Capybara je knihovna pro testování webových aplikací napsaná v jazyce Ruby. Hlavní výhodou tohoto řešení je snadnost použití díky „out of the box“ funkčnosti a intuitivnímu aplikačnímu rozhraní, které umožňuje psát testy formou blízkou běžnému textu. Používá pro to vlastní doménově specifický jazyk. Další předností je podpora několika různých ovladačů. Ve výchozím stavu se používá RackTest, který vyniká rychlostí, ale nepodporuje například JavaScript. Pro tuto práci je podstatná podpora ovladače Selenium WebDriver. Ten je dostupný po nainstalování balíčku (*gem*) selenium-webdriver. Výhodou jeho použití oproti ovladači RackTest je výše zmíněna podpora jazyka JavaScript a dále také možnost přistupovat k HTTP zdrojům mimo aplikaci, například se vzdáleným aplikačním rozhraním nebo autorizačním službám (Nicklas 2016).

Tabulka 3: Shrnutí nástroje Capybara (zdroj:autor)

Capybara	
Použití	Knihovna pro testování webových aplikací v jazyce Ruby
Dostupnost	Zdarma
Domovská stránka	http://jnicklas.github.io/capybara/ https://github.com/jnicklas/capybara
Nejdůležitější vlastnosti	Podpora několika ovladačů Snadno čitelný jazyk

6.1.2 Oxygen

Oxygen je jednoduchý framework postavený na nástroji Selenium, který rozšiřuje o vlastní aplikační rozhraní umožňující snadnější zápis nejčastěji používaných vzorů. Pro zápis testů je možné využít JavaScript nebo jazyk Selenese, který využívá i samotné Selenium. Kromě toho Oxygen nabízí také vlastní jednoduché vývojové prostředí, pomocí kterého je možné zaznamenávat dění v prohlížeči a vytvářet tak jednoduché testy (viz Obrázek 11). Pro záznam je nutné nainstalovat rozšíření do internetového prohlížeče. V současné době jsou podporovány prohlížeče Chrome a Internet Explorer ve verzi 9 až 11.



Obrázek 8:Vývojové prostředí Oxygen (zdroj:autor)

Testy je možné spouštět z vývojového prostředí. Druhou možností je použití aplikace Oxygen Server využívající Selenium server pro spouštění testů. Výsledky testů se ukládají do skupiny

XML souborů. V případě chyby testu je navíc vytvořen snímek obrazovky (CloudBeat Limited 2015).

Tabulka 4: Shrnutí nástroje Oxygen (zdroj:autor)

Oxygen	
Použití	Framework umožňující psát testy v jazyce JavaScript nebo Selenese
Dostupnost	Zdarma
Domovská stránka	http://oxygenhq.org
Nejdůležitější vlastnosti	Vlastní vývojové prostředí Zápis testů v jazyce JavaScript

6.1.3 Watir WebDriver

Watir (*Web Automated Testing In Ruby*) je rodina knihoven napsaných v jazyce Ruby pro automatizaci webových prohlížečů. V původní verzi bylo možné pracovat pouze s prohlížečem Internet Explorer, druhá verze používá Selenium WebDriver a umožňuje tak ovládat téměř jakýkoliv prohlížeč. Watir nabízí rozsáhlé aplikační rozhraní. Jeho velkou výhodou je aktivní komunita, díky které je na Internetu dostupná celá řada návodů a obsáhlá dokumentace.

Tabulka 5: Shrnutí nástroje Watir WebDriver (zdroj:autor)

Watir WebDriver	
Použití	Rodina knihoven pro automatizaci webových prohlížečů pro jazyk Ruby
Dostupnost	Zdarma
Domovská stránka	http://watir.com https://watirwebdriver.com
Nejdůležitější vlastnosti	Rozsáhlé aplikační rozhraní Aktivní komunita Podporuje pouze jazyk Groovy

6.1.4 Vapir

Vapir je odnož původní verze knihovny Watir, která opravuje několik chyb a přináší lepší podporu například pro tabulky. Poslední aktualizace je z roku 2011 a jedná se tak o dnes již mrtvý projekt.

Tabulka 6: Shrnutí nástroje Vapir (zdroj:autor)

Vapir	
Použití	Knihovna pro automatizaci webových prohlížečů pro jazyk Ruby
Dostupnost	Zdarma
Domovská stránka	https://github.com/vapir/vapir
Nejdůležitější vlastnosti	V současnosti neaktivní a zastaralý projekt

6.1.5 Nightwatch.js

Dalším frameworkem pro testování webových aplikací je Nightwatch. Jeho hlavní specialitou je zaměření na vývojáře programující v jazyce JavaScript. Nightwatch je založen na serverovém frameworku Node.js, což je v současnosti jedna z nejpopulárnějších technologií používaná při vývoji webových aplikací (Khan 2015). To vývojářům umožňuje používat stejný jazyk pro vývoj i automatizované testování aplikací, což je hlavní výhodou tohoto nástroje oproti ostatním. Pro samotný běh testů se používá serverová část nástroje Selenium WebDriver, díky čemuž je možné využít i funkce Selenium Grid. Další výhodou je možnost vytvořit si vlastní funkce. Celkově jde o zřejmě nejlepší volbu pro testování aplikací napsaných v Node.js.

Tabulka 7: Shrnutí nástroje Nightwatch.js (zdroj:autor)

Nightwatch.js	
Použití	Framework pro testování webových aplikací
Dostupnost	Zdarma
Domovská stránka	http://nightwatchjs.org https://github.com/nightwatchjs/nightwatch
Nejdůležitější vlastnosti	Zaměřené na Node.js Podpora funkce Selenium Grid Možnost vytvářet vlastní rozšíření

6.1.6 Satix

Satix je framework pro automatizaci testování webových aplikací napsaný v jazyce Java. Je založený na nástroji Selenium WebDriver a jeho specialitou je zápis testů v jazyce XML. Pro zápis testů je tak nutné znát pouze základy XML a umět pracovat s objektovým modelem HTML dokumentu. Nevýhodou nástroje je velmi omezená dokumentace a celková neaktivita tvůrce tohoto projektu. Poslední aktualizace proběhla před dvěma roky. Další nevýhodou je cenový model, kdy bezplatně je možné nástroj používat pouze pro jednu aplikaci, za vyšší licence je nutné zaplatit jednorázový poplatek ve výši 500 nebo 1000 dolarů.

Tabulka 8: Shrnutí nástroje Satix (zdroj:autor)

Satix	
Použití	Framework pro automatizaci testování webových aplikací
Dostupnost	Základní licence pro jednu aplikaci zdarma Licence pro 5 aplikací + 6 měsíců podpora – 499,99 $ Neomezená vývojářská licence – 999,99 $
Domovská stránka	http://www.binpress.com/app/satix-seleniumbased-automation-testing-in-xml/1958
Nejdůležitější vlastnosti	Zápis testů v jazyce XML Omezená bezplatná verze Neaktivní

6.1.7 Magium

Magium je dalším frameworkem založeným na nástroji Selenium WebDriver. Teoreticky je možné ho použít pro jakoukoliv webovou aplikaci, primárně je ale zaměřen na v současnosti nejpopulárnější eCommerce platformu Magento (Webzoom.cz 2016). Magium využívá programovací jazyk PHP, jehož znalost alespoň na základní úrovni je nutná pro tvorbu testů. Z toho důvodu se domnívám, že Magium se vyplatí používat pouze spolu s platformou Magento.

Tabulka 9: Shrnutí nástroje Magium (zdroj: autor)

Magium	
Použití	Framework pro automatizaci testování webových aplikací
Dostupnost	Zdarma
Domovská stránka	http://magiumlib.com https://github.com/magium
Nejdůležitější vlastnosti	Zaměření na eCommerce platformu Magento Použití programovacího jazyka PHP

6.1.8 RedwoodHQ

Redwood je bezplatný nástroj pro kompletní správu, tvorbu a spouštění testů. Testy je možné psát v jazyce Java, Groovy, Jedná se o serverovou aplikaci, která obsahuje vlastní vývojové prostředí pro tvorbu testů. Součástí aplikace je také podpora pro Selenium WebDriver. Testy jsou vytvářeny skládáním předdefinovaných nebo vlastních akcí dohromady. Kromě toho je možné také použít funkci Looking Glass, která funguje podobně jako Selenium IDE a umožňuje výběr elementů webové stránky nebo i samotnou tvorbu testů pomocí uživatelsky přívětivého grafického rozhraní. Oproti ostatním aplikačním rámcům zmíněným v této práci se jedná o daleko mohutnější nástroj, který lze navíc rozšířit o další funkce zapojením dalších nástrojů, například pro průběžnou integraci (Jenkins a TeamCity) nebo pro týmovou správu kódu (Git) (PrimaTest 2013).

Tabulka 10: Shrnutí nástroje RedwoodHQ (zdroj: autor)

RedwoodHQ	
Použití	Nástroj pro správu, tvorbu a spouštění testů
Dostupnost	Zdarma
Domovská stránka	http://redwoodhq.com https://github.com/dmolchanenko/RedwoodHQ
Nejdůležitější vlastnosti	Vlastní vývojové prostředí Podpora několika programovacích jazyků (Java, Groovy, Python and C#)

6.1.9 Sahi

Sahi je dalším robustním nástrojem pro testování webových aplikací. Obsahuje vlastní vývojové prostředí pro záznam testů. Na rozdíl od standardní verze nástroje Selenium IDE podporuje záznam z jakéhokoliv prohlížeče. Sahi totiž funguje jako proxy server a konfiguruje webové prohlížeče tak, aby směřovaly na tuto proxy. Sahi následně vkládá do stránek svůj vlastní skript, který umožňuje nahrávat veškeré interakce prohlížeče se stránkou. Nahrané testovací skripty je samozřejmě možné upravovat. Sahi využívá svůj vlastní jazyk Sahi Script, který používá stejnou syntaxi jako JavaScript, pouze pro proměnné se používá prefix $ (Sahi Pro 2015). Sahi existuje v bezplatné a placené verzi. Bezplatná verze umožnuje záznam a spouštění testů na všech dostupných prohlížečích. Pokročilé funkce jako editace skriptů, jejich ukládání v databázi nebo vlastní reporty obsahuje pouze placená verze Sahi Pro, jejíž cena začíná na 695 dolarech za jednoho uživatele. Podrobné porovnání obou verzí je dostupné na stránkách produktu (Sahi Pro 2016).

Tabulka 11: Shrnutí nástroje Sahi (zdroj: autor)

Sahi	
Použití	Nástroj pro správu, tvorbu a spouštění testů
Dostupnost	Základní verze zdarma Plná verze- 695 dolarů ročně za jednoho uživatele (konkrétní uživatel), 995 dolarů ročně za jednu přenositelnou licenci (může používat více uživatelů, ale ne zároveň)
Domovská stránka	http://sahipro.com/
Nejdůležitější vlastnosti	Vlastní vývojové prostředí Vlastní skriptovací jazyk vycházející z jazyku JavaScript Používá proxy server pro komunikaci s prohlížeči

6.1.10 SeLion

SeLion je vyspělý framework pro testování webových aplikací využívající Selenium WebDriver, za kterým stojí internetový platební systém PayPal. Tento nástroj je napsaný nativně pro jazyk

Java. Jak zmiňují sami autoři, SeLion míří především na velké webové aplikace a nemá sloužit jako alternativa pro jednoduché aplikační rámce, které umožňují nahrávat interakce prohlížeče se stránkou. Minimálně základní znalost programování v jazyce Java je nutná. Na druhou stranu SeLion umožňuje využít pokročilé funkce jako Selenium Grid nebo možnost načítat testovací data ve formátu Excel, YAML, JSON nebo XML. Pro běh testů tento nástroj využívá testovací framework TestNG.

Tabulka 12: Shrnutí nástroje SeLion (zdroj: autor)

SeLion	
Použití	Framework pro automatizaci testování především rozsáhlých webových aplikací
Dostupnost	Zdarma
Domovská stránka	http://paypal.github.io/SeLion/ https://github.com/PayPal/SeLion
Nejdůležitější vlastnosti	Nativně pro jazyk Java Podpora funkce Selenium Grid

6.1.11 Webdriver.io

Webdriver.io je dalším nástrojem pro automatizované testování webových aplikací využívajících framework Node.js. Jeho zajímavostí je možnost fungovat ve dvou rozdílných režimech. První možností je integrovat nástroj Webdriver.io do vlastní automatizační knihovny. V tomto případě se využívá aplikační rozhraní tohoto nástroje pro ovládání prohlížeče. Tuto možnost využívají například nástroje Chimp nebo CodeceptJS.

Druhou možností je použít Webdriver.io jako plnohodnotný automatizační nástroj za využití jednoho z dostupných testovacích frameworků: Mocha, Jasmine nebo Cucumber. Tento režim umožňuje využít všechny pokročilé funkce tohoto nástroje, jako je například vlastní zjednodušená syntaxe příkazů nebo podpora funkce Selenium Grid včetně testování v cloudu.

Tabulka 13: Shrnutí nástroje Webdriver.io (zdroj: autor)

Webdriver.io	
Použití	Framework pro automatizaci testování webových aplikací
Dostupnost	Zdarma
Domovská stránka	http://webdriver.io https://github.com/webdriverio
Nejdůležitější vlastnosti	Možnost fungovat samostatně nebo jako součást automatizační knihovny Podpora funkce Selenium Grid a cloudových testovacích služeb Sauce Labs, BrowserStack a TestingBot

6.1.12 Chimp

Chimp je jedním z frameworků, které využívají Webdriver.io pro svůj běh. Je tedy napsán v Node.js a samotné testy se píší v jazyce JavaScript. Zajímavostí je možnost vyhodnocování testů v reálném čase. To znamená, že příkazy psané vývojářem do zdrojového kódu jsou ihned aplikovány ve webovém prohlížeči a testovací klauzule jsou ihned vyhodnoceny v konzoli. Stejně jako Webdriver.io, z kterého vychází, podporuje Chimp Selenium Grid a testování v cloudu.

Tabulka 14: Shrnutí nástroje Chimp (zdroj: autor)

Chimp	
Použití	Framework pro automatizaci testování webových aplikací
Dostupnost	Zdarma
Domovská stránka	https://chimp.readme.io https://github.com/xolvio/chimp
Nejdůležitější vlastnosti	Zaměřené na Node.js Podpora funkce Selenium Grid Vyhodnocování testů v reálném čase

6.1.13 CodeceptJS

Stejně jako Chimp využívá CodeceptJS nástroj Webdriver.io. Opět se teda jedná o nástroj pro testování aplikací napsaných v jazyce Node.js. Další podobností je možnost vyhodnocování testů v reálném čase. Navíc ale umožňuje nahradit Webdriver.io jiným nástrojem. K dispozici je SeleniumWebdriver, což je oficiální Selenium knihovna pro JavaScript. Další možnost je použít Protractor, což je aplikační framework pro testování webových aplikací napsaných v jazyce AngularJS (CodeceptJS 2016). Jedná se o velmi mladý nástroj, který je ve vývoji od listopadu 2015.

Tabulka 15: Shrnutí nástroje CodeceptJS (zdroj: autor)

CodeceptJS	
Použití	Framework pro automatizaci testování webových aplikací
Dostupnost	Zdarma
Domovská stránka	http://codecept.io https://github.com/codeception/codeceptjs/
Nejdůležitější vlastnosti	Zaměřené na Node.js Podpora několika automatizačních aplikačních rozhraní Vyhodnocování testů v reálném čase

6.1.14 Codeception

Codeception je komplexní testovací framework v jazyce PHP. Nezaměřuje se tedy pouze na automatizaci webového prohlížeče (akceptační testování), ale umožňuje též psát jednotkové testy nebo testy aplikačního rozhraní webových služeb (Codeception 2016). Z pohledu této práce je nejzajímavější možnost nahradit výchozí „headless“ prohlížeč PHP Browser bez grafického výstupu nástrojem Selenium WebDriver, díky čemuž je možné testovat stránky v reálném prohlížeči nebo lze využít jednu z cloudových služeb: Sauce Labs, BrowserStack a TestingBot.

Tabulka 16: Shrnutí nástroje Codeception (zdroj: autor)

Codeception	
Použití	Komplexní testovací framework v jazyce PHP
Dostupnost	Zdarma
Domovská stránka	http://codeception.com https://github.com/Codeception
Nejdůležitější vlastnosti	Zaměřeno na jazyk PHP Podporuje akceptační, jednotkové a API testování Podpora cloudových testovacích služeb Sauce Labs, BrowserStack a TestingBot

6.1.15 Selenide

Selenide je framework pro automatizaci webových prohlížečů pro jazyk Java. Funguje s libovolným testovacím frameworkem (JUnit, TestNG, Cucumber, ScalaTest, JBehave) a podporuje všechny důležité pokročilé funkce: Selenium Grid, tvorbu snímků obrazovky při pádu testu apod. Celkově se jedná o velmi vyspělý nástroj vyvíjený v době psaní práci již tři roky, který přináší obvyklá vylepšení automatizačních frameworků. Výhodou je též detailní dokumentace a velké množství veřejně dostupných příkladů.[5]

Tabulka 17: Shrnutí nástroje Selenide (zdroj: autor)

Selenide	
Použití	Framework pro automatizaci testování webových aplikací v jazyce Java
Dostupnost	Zdarma
Domovská stránka	http://selenide.org/
Nejdůležitější vlastnosti	Zaměřeno na jazyk Java Široká komunita

[5] https://github.com/selenide-examples

6.1.16 Qualitia

Qualitia je produkt, jehož tvůrci ho označují jako platformu pro automatizaci Selenium testů. Umožňuje vytvářet testy bez nutnosti psát skripty. K dispozici je vývojové prostředí, kde lze přidávat do testů jednotlivé funkce a vybírat elementy webové stránky kliknutím. K dispozici je také centralizovaná správa objektů. Jedná se o mohutný komerční nástroj, který se zaměřuje především na velké firmy. Typickým zákazníkem jsou společnosti s ročním příjmem přes 50 milionů dolarů (Chaudhari 2013).

Tabulka 18: Shrnutí nástroje Qualitia (zdroj: autor)

Qualitia	
Použití	Platforma pro automatizaci Selenium testů
Dostupnost	Zpoplatněný nástroj, přesná cena neznámá (v řádech tisíců dolarů za licenci a údržbu)
Domovská stránka	http://www.qualitiasoft.com
Nejdůležitější vlastnosti	Kompletní vývojové prostředí Možnost vytvářet testy bez znalosti skriptů

6.1.17 Geb

Geb je nástroj pro automatizaci interakcí mezi webovým prohlížečem a webovou stránkou. Geb je abstrakcí aplikačního rozhraní WebDriveru (WebDriver API) a přidává možnost psát testy v jazyku Groovy, využívat selektory jako v jQuery či možnost pracovat s jednotlivými stránkami jako v objekty. Geb je vyvíjen od roku 2009 a v současné době existuje ve verzi Geb 0.13.x. Za jeho vývojem stojí především Marcin Erdmann a Luke Daley, který se také podílí na vývoji testovacího frameworku Spock.

Přestože se je Geb distribuován zdarma, jedná se o velmi kvalitní a vyspělý nástroj. Značnou výhodou je dostupnost velkého množství příkladů, které demonstrují použití tohoto aplikačního rámce s vybranými nástroji pro automatizaci, s testovacími aplikačními rámci a službami Sauce Labs a BrowserStack.

Tabulka 19: Shrnutí nástroje Geb (zdroj: autor)

Geb	
Použití	Framework pro automatizaci testování webových aplikací
Dostupnost	Zdarma
Domovská stránka	http://www.gebish.org
Nejdůležitější vlastnosti	Možnost použit jQuery selektory Použití jazyku Groovy Podpora cloudových testovacích služeb Sauce Labs a BrowserStack

6.1.18 Shrnutí přehledu aplikačních rámců pro nástroj Selenium WebDriver

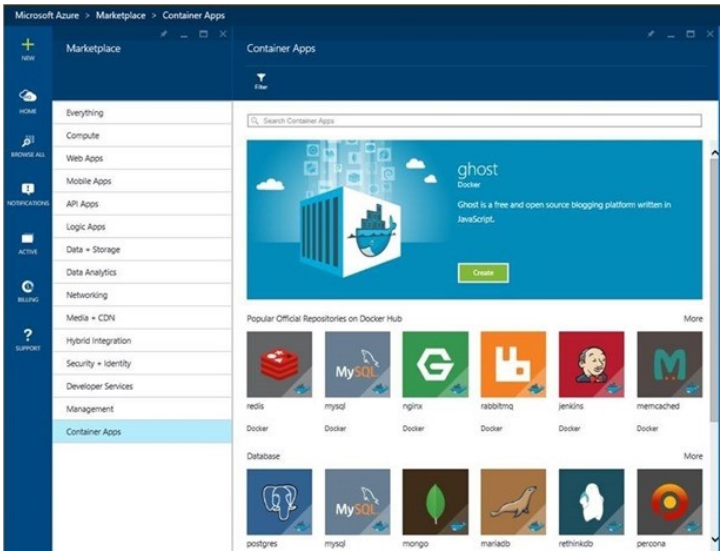
Z důvodu dostupnosti velké řady aplikačních rámců je trh s aplikačními rámci velmi nepřehledný. Volba vhodného nástroje tak může být problematická. Doporučuji nejprve sestavit si seznam požadavků, mezi které může patřit například podpora konkrétního programovacího jazyka nebo určité funkce či služby. Na základě těchto požadavků by následně mělo být možné vybrat nejvhodnější nástroj.

6.2 Softwarové kontejnery

Velmi populární technologií se v poslední době staly softwarové kontejnery. Jedná se o aplikace nebo aplikační rámce, které umožňují zabalit aplikace a služby jako bitové kopie, které běží ve svých vlastních přenositelných kontejnerech. Tyto kontejnery lze bez jakýchkoliv úprav spouštět na libovolném kompatibilním prostředí. Jedná se o ve své podstatě o velmi odlehčenou formu virtualizace, která je daleko šetrnější k prostředkům. Běh kontejnerů je totiž daleko méně náročnější na hardwarový výkon než klasické formy virtualizace. Dalším přínosem kontejnerů je snadná škálovatelnost a přenositelnost z jednoho stroje na druhý (Venezia 2015).

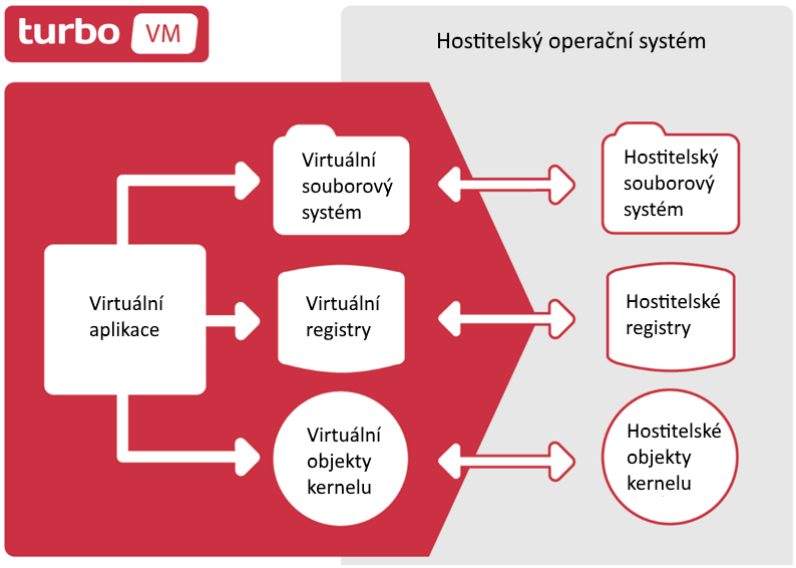
Jedním z hlavních hráčů na poli softwarových kontejnerů je Docker. Jedná se o open source aplikační rámec vyvíjený nativně pro operační systém Linux. Pro běh využívá funkci LXC (Linux Container), která je součástí linuxového jádra od verze kernelu 2.6.27 a která umožňuje právě

běh softwarových kontejnerů. V současné době je ve fázi vývoje také podpora pro Docker v rámci operačního systému Windows Server 2016. Ten byl v době psaní práce dostupný jako technologické demo (Tech Preview 4). Kromě toho je v současnosti možné používat Docker v rámci cloudové platformy Microsoft Azure.



Obrázek 9: Kontejnerové aplikace v Microsoft Azure (zdroj: Vybava Ramadoss 2015)

Další možností, jak používat kontejnery v rámci operačního systému Windows, je využití platformy Turbo. Na rozdíl od nástroje Docker nepotřebuje Turbo konkrétní verzi kernelu. Využívá svoji vlastní obdobu virtuálního stroje s vlastním souborovým systémem, registry a kernelovými objekty. Ty komunikují se svými ekvivalenty na hostitelském stroji (viz Obrázek 10).



Obrázek 10: Architektura platformy Turbo (zdroj: Turbo 2016, přeloženo autorem)

Softwarové kontejnery v této práci zmiňuji z toho důvodu, že obě dvě zmíněné platformy podporují Selenium. V případě nástroje Docker jsou k dispozici balíčky jak pro hub, tak i pro jednotlivé uzly s prohlížeči.[6] Pro Turbo je k dispozici jeden kompletní balíček, který obsahuje hub i uzly, které se automaticky spouštějí podle potřeby.[7] Výhodou těchto řešení je již zmíněná multiplatformovost a snadnost použití. Balíčky jsou vždy aktualizované a připravené k použití.

6.3 Integrace s nástroji pro automatizaci a průběžnou integraci

V této kapitole se věnuji možnostem integrace s nástroji pro automatizaci a průběžnou integraci. Nejprve stručně definuji tyto nástroje a rozdíly v jejich použití. Následně uvedu možnosti jejich využití spolu s nástrojem Selenium WebDriver.

Domnívám se, že využití nástroje pro automatizaci a průběžnou integraci jsou nutností u každého vývojářského týmu. Oba typy nástrojů velmi usnadňují vývoj a spolupráci mezi členy. Jak vyplývá z názvu, nástroje pro automatizaci umožňuji automatizovat činnosti prováděné při vývoji aplikace. Nejběžnějším použitím je automatizace sestavování aplikace. Usnadňují tedy proces tvorby aplikace ze zdrojového kódu. Mezi nástroje pro automatizaci patří například Gradle, Maven nebo ANT. Naproti tomu nástroje po průběžnou integraci zjednodušují distribuci a integraci kódu v týmu. Základem je centrální úložiště kódu, které zajišťuje konzistenci souborů. Vývojáři synchronizují změny provedené v kódu s tímto serverem, čímž je zajištěna neustálá dostupnost aktuálních zdrojových kódů a ostatních souborů. Nástroje pro průběžnou integraci zpravidla nabízejí i další funkcionalitu, jako například verzování a zálohování kódu. Mezi nejznámějšími zástupce lze zařadit nástroje Jenkins, CircleCI, Travis, Hudson nebo Bamboo. Velmi často se se nástroje pro automatizaci a integraci používají dohromady. V tomto případě integrační server obsahuje i nástroj pro automatizaci. Typickým příkladem použití je automatické spuštění sestavení aplikace a testů po každé změně v kódu.

Selenium WebDriver a odvozené aplikační rámce umožňují využívat jak nástroje pro automatizace, tak i nástroje pro průběžnou integraci. Nástroje pro automatizaci je možné využít pro správu závislostí (např. stažení požadované verze nástroje Selenium) nebo

[6] https://hub.docker.com/r/selenium/
[7] https://turbo.net/selenium

pro konfiguraci a spouštění jednotlivých testů nebo jejich sestav. Příklad použití automatizačního nástroje Gradle je uveden v případové studii (viz 7.5 Postup zavádění automatizovaného testování ve společnosti BellaDati). Nástroje pro průběžnou integraci lze využít pro správu kódu testů nebo pro automatické spouštění testů. Testy je možné spouštět přímo na integračním serveru. Tyto servery zpravidla pracují v bezobrazovkovém režimu, a proto na nich ve výchozím stavu nelze spouštět testy v běžně používaných webových prohlížečích. Řešením je instalace serveru Xvfb, který funguje jako virtuální zobrazovací zařízení. Druhou možností je použití funkce Selenium Grid, kdy integrační server funguje pouze jako hub, který je připojen ke klientům, které umožňují standardní běh prohlížečů.

Nástroje pro automatizaci a integraci velmi usnadňují vývoj, správu a spouštění testů. Jejich využití má smysl především u středních a větších týmů, kde na testech pracuje více členů. Jejich význam také roste spolu se složitostí a komplexností aplikace, respektive přidružených testů. Čím více testů je potřeba spravovat, tím vyšší je přínos těchto nástrojů.

6.4 Služby pro nástroj Selenium WebDriver

Jednou z nejzajímavějších funkcí nástroje Selenium je možnost využít model klient-server, případně použít Selenium Grid a distribuovat spouštění testů na několik klientů s možností spouštět testy paralelně. Toto řešení má jeden významný problém. Je nutné vlastnit a spravovat celou řadu stanic, což s sebou přináší vysoké náklady, jak na nákup zařízení, tak i na správu. Řešením je využít jednu z cloudových služeb poskytujících vzdálený přístup k portfoliu webových prohlížečů. Na trhu existuje několik poskytovatelů, kteří nabízejí srovnatelné služby lišící se pouze v detailech, například v podpoře konkrétního automatizačního frameworku. Jejich stručný přehled lze nalézt v tabulce níže (viz Tabulka 20).

Tabulka 20: Shrnutí cloudových služeb poskytujících vzdálený přístup k portfoliu webových prohlížečů (zdroj: autor)

Název služby	BrowserStack[8]	Sauce Labs[9]	TestingBot[10]	Gridlastic[11]
Počet kombinací OS a webového prohlížeče	700+	700+	538	Neznámé, ale nižší (Windows 10, Windows XP a jiné)
Windows	10, 8.1, 8, 7, XP	10, 8.1, 8, 7, XP	10, 8, 7, XP	8.1, 8, 7
OS X	El Capitan, Yosemite, Mavericks, Mountain Lion, Lion, Snow Leopard	El Capitan, Yosemite, Mavericks, Mountain Lion	El Capitan, Yosemite, Mavericks	ne
Linux	Ne	Ano	Ano	Ano
Mobilní platformy	Ano (iOS, Android, Windows Phone)	Ano (iOS, Android)	Ano (iOS, Android)	Ne
Cena (pro čtyři paralelní stroje)	299 amerických dolarů měsíčně (2000 minut automatizace)	259 amerických dolarů měsíčně (2000 minut automatizace)	40 amerických dolarů měsíčně (1000 minut automatizace)	295 amerických dolarů měsíčně (cca 7000 minut automatizace)

Princip fungování těchto služeb je téměř stejný. Namísto lokálního Selenium hubu se ve zdrojovém kódu definuje URL adresa vzdáleného hubu spolu s přístupovými údaji, zpravidla ve formě tajných klíčů aplikačního rozhraní - API KEY a API SECRET (viz Výpis kódu 3).

[8] https://www.browserstack.com
[9] https://saucelabs.com
[10] https://saucelabs.com
[11] https://www.gridlastic.com

```
1.  import org.openqa.selenium.By;
2.  import org.openqa.selenium.Platform;
3.  import org.openqa.selenium.WebDriver;
4.  import org.openqa.selenium.WebElement;
5.  import org.openqa.selenium.remote.DesiredCapabilities;
6.  import org.openqa.selenium.remote.RemoteWebDriver;
7.
8.  import java.net.URL;
9.
10. public class JavaSample {
11.   public static final String URL = "http://API_KEY:API_SECRET@hub.testingbot.com/wd/
    hub";
12.
13.   public static void main(String[] args) throws Exception {
14.
15.     DesiredCapabilities caps = new DesiredCapabilities();
16.     caps.setCapability("browserName", "Firefox");
17.     caps.setCapability("version", "44");
18.     caps.setCapability("platform", "WINDOWS");
19.
20.     WebDriver driver = new RemoteWebDriver(new URL(URL), caps);
21.     driver.get("http://www.google.com/ncr");
22.     WebElement element = driver.findElement(By.name("q"));
23.
24.     element.sendKeys("TestingBot");
25.     element.submit();
26.
27.     System.out.println(driver.getTitle());
28.     driver.quit();
29.   }
30. }
```

Výpis kódu 3: Příklad použití cloudové služby TestingBot (zdroj: TestingBot 2016)

7 Případová studie

Jedním z cílů této práce je popis zavedení automatizovaného testování pomocí nástroje Selenium ve firmě BellaDati. Tato společnost se zabývá vývojem nástrojem Business Intelligence (dále BI). V první části případové studie stručně definuji pojem Business Intelligence a popíši specifika funkcionálního testování BI aplikací. V druhé části se zaměřím na popis aplikace BellaDati BI a jednotlivých modulů, které obsahuje. Následně popíši podnikové testovací procesy dle upravené metodiky MMSP. V poslední části popíši zvolené testovací řešení včetně důvodů pro výběr testovacích nástrojů.

7.1 Business Intelligence

Termínem Business Intelligence se obecně označuje procesy, nástroje a technologie, které by měly prostřednictvím analýzy podnikových dat podporovat rozhodovací procesy ve firmě. Jedním ze základních kamenů BI je takzvaný multidimenzionální pohled na data (Slánský et al. 2004, s. 19). Multidimenzionální pohled představuje možnost nahlížet na data z několika různých pohledů. „*Standardními dvěma dimenzemi jsou tu ukazatele (ekonomické proměnné) a čas. Ostatní dimenze se pro jednotlivé modely definuji podle potřeby, např. organizační jednotka, komodita, zákazník, dodavatel, teritorium, konkurent apod. Obsah dimenzi je tvořen prvky dimenzi, tj. konkrétními závody, provozy nebo zákazníky, dodavateli, apod. Promítnutí všech dimenzi do jednoho bodu tvoří prvek multidimenzionální databáze. Každý prvek může pak obsahovat data nebo předpisy (algoritmy) pro jejich transformace.“* (Slánský et al. 2004, s. 22)

7.1.1 Specifika funkcionálního testování Business Intelligence aplikací

Funkcionální testování BI aplikací má některé specifické vlastnosti. Velmi důležitá je absolutní přesnost, neboť BI aplikace pracují s kritickými a citlivými daty. Jedná se o velmi komplexní aplikace s celou řadou modulů, u kterých je nutné otestovat jejich spolupráci. Jednou z možností je testovat tzv. rodokmen dat (angl. *Data Lineage*). Jedná se o kontrolu správné funkčnosti jednotlivých kroků ETL fáze. Při tomto typu testování se porovnávají očekávané a skutečné hodnoty dat po každé transformaci. Detailněji se rodokmenu dat věnuje Miroslav Haupt ve své diplomové práci (Haupt 2009). Dále doporučuji diplomovou práci Nely

Jakubičkové, která se věnuje testování BI aplikací obecně. Součástí práce je také návrh metodiky testování BI řešení (Jakubičková 2011).

Kromě datové vrstvy se funkcionální testování zaměřuje především na prezenční vrstvu, tedy reporty a dashboardy. V rámci této vrstvy probíhá celá řada testování:

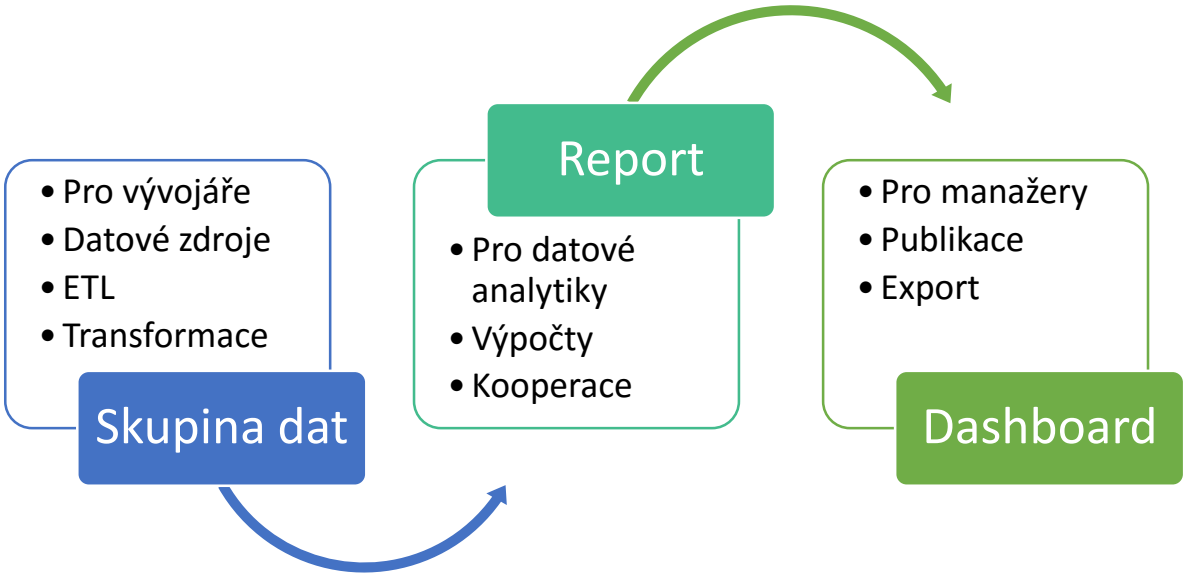
- **testování vizuální stránky reportů** ověřuje správné vykreslování vizualizací, popisků a ovládacích prvků;
- **testování proměnných filtrů** kontroluje správnou funkčnost jednotlivých typů komponent, která se využívají pro filtrování nebo zadávání proměnných;
- **testování správnosti dat** navazuje na testování ETL fáze a ověřuje korektní zobrazování dat;
- **testování rozpadu** ověřuje správnou funkčnost funkcí pro rozpad (angl. *drill down*), které umožňují měnit granulitu dat;
- **testování výkonnosti** slouží k ověření rychlosti vykreslování reportů (Datagaps 2016).

7.2 BellaDati BI

BellaDati BI je agilní Business Intelligence systém. Na rozdíl od tradičních BI systémů zobrazují agilní systémy vizualizace dat v reálném čase, ihned po jejich změně. Systémy tohoto typu se zaměřují na uživatelskou přívětivost a snadné ovládání. Na rozdíl od ostatních agilních systémů ale BellaDati BI obsahuje i pokročilejší funkce, včetně pokročilých transformací dat nebo vlastního API a SDK.

7.2.1 Základní vlastnosti aplikace

Aplikaci je možné rozdělit na tři hlavní moduly. Každý z těchto primárně určený pro jiný typ uživatele (viz Obrázek 11). Tyto moduly jsou doplněny dalšími funkčnostmi, díky čemuž je možné tento software považovat za kompletní BI řešení. V následující kapitoly popíši jednotlivé části a funkce BellaDati BI.



Obrázek 11: Worklow BellaDati (zdroj: autor)

7.2.2 Skupiny dat

Skupina dat představuje reprezentaci dat nahraných do systému prostřednictvím ETL procesu. Skupinu dat je možné vytvořit z lokálního souboru (xlsx, csv atd.) nebo je možné připojit se k jednomu z mnoha možných zdrojů. BellaDati v současné době podporuje následující datové zdroje:

- SQL Database,
- Microsoft Analysis Services (SSAS),
- SAP BW a SAP HANA,
- Hadoop,
- MongoDB,
- FTP,
- Google Analytics,
- Google Drive,
- Twitter,
- LinkedIn,
- Zendesk,
- Salesforce,

- Amiando,
- Intuit,
- MS SharePoint,
- Projector (BellaDati 2015).

Díky tomu, že BellaDati obsahuje vlastní kompletní ETL řešení, je možné během importu s daty pracovat. Lze transformovat jednotlivé hodnoty, přidávat a odebírat sloupce a měnit jejich datové typy. Kromě jednorázového importu je možné nastavit také pravidelné spouštění importu či import vyvolávat na základě splnění určité podmínky.

Již nahraná data je možné dále upravovat. Lze vytvářet spojení mezi několika skupinami dat (obdoba „*JOINu*“ z jazyka SQL) nebo dále upravovat data pomocí transformačních skriptů. Ty je možné využít k matematickým, datovým nebo slovním transformacím zdrojových dat. Pro jejich zápis se používá jazyk Groovy. Základní transformace je ale možné provádět i bez znalosti tohoto jazyka. Dále je možné nastavovat přístupová práva, ať u k celé datové skupině nebo k její části, neboť každý atribut a ukazatel může mít své vlastní nastavení. Při sdílení je možné pracovat také s uživatelskými rolemi a skupinami.

7.2.3 Report

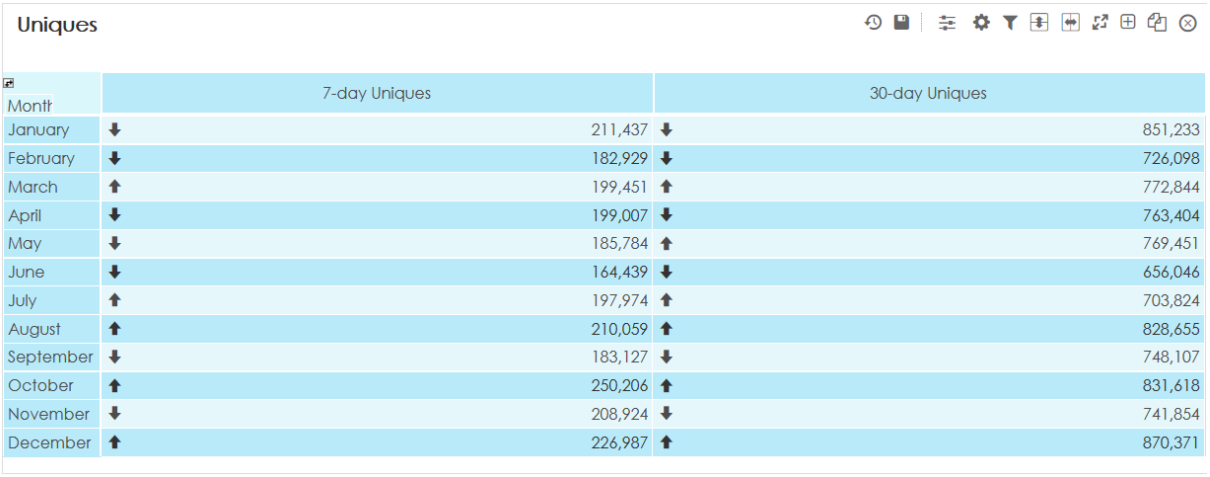
Reporty v BellaDati slouží k analýze dat uložených ve skupinách dat. Každý report náleží k určité skupině dat, ale v rámci reportu je možné přistupovat i k datům z ostatních skupin. Základní stavební komponentou reportu je pohled. Report může obsahovat neomezené množství pohledů, jejichž rozložení je možné libovolně uspořádávat. Veškeré změny v reportu jsou prováděny ihned bez nutnosti zdlouhavého přepočítávání výsledků. Pohled obsahuje ukazatele (metriky) a atributy (dimenze). K dispozici je několik typů pohledů:

- Tabulka,
- Graf,
- Mapa,
- KPI popisek,
- Vlastní obsah,
- Filtr.

V následující části práce detailněji popíši jednotlivé typy pohledů.

Tabulka

Jak vyplývá z názvu, tabulka je řádkové a sloupcové zobrazení agregovaných dat. Vertikální a horizontální hlavička tabulky zpravidla obsahuje dimenze, přičemž samotný obsah tabulky tvoří metriky. Tabulka může obsahovat libovolný počet ukazatelů a metrik, díky čemuž se jedná o multidimenzionální zobrazení dat. Kromě toho je možné přidávat dimenze ad-hoc a měnit tak rozpad dat dynamicky nebo naopak využít předpřipravené rozpadové cesty z několika dimenzí. Mimo úprav obsahu lze upravovat také vzhled tabulky. Uživatel si může u každé tabulky nastavit libovolné barvy nebo využít jedno ze standardních témat. Lze také měnit řazení hodnot, šířku sloupců nebo přidat zobrazení součtů.



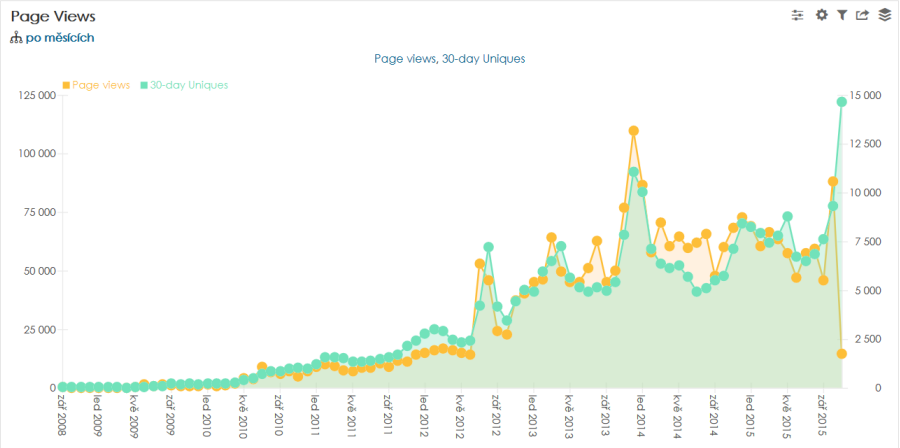
Uniques

Montt	7-day Uniques	30-day Uniques
January	211,437	851,233
February	182,929	726,098
March	199,451	772,844
April	199,007	763,404
May	185,784	769,451
June	164,439	656,046
July	197,974	703,824
August	210,059	828,655
September	183,127	748,107
October	250,206	831,618
November	208,924	741,854
December	226,987	870,371

Obrázek 12: Tabulka v BellaDati BI (zdroj: autor)

Graf

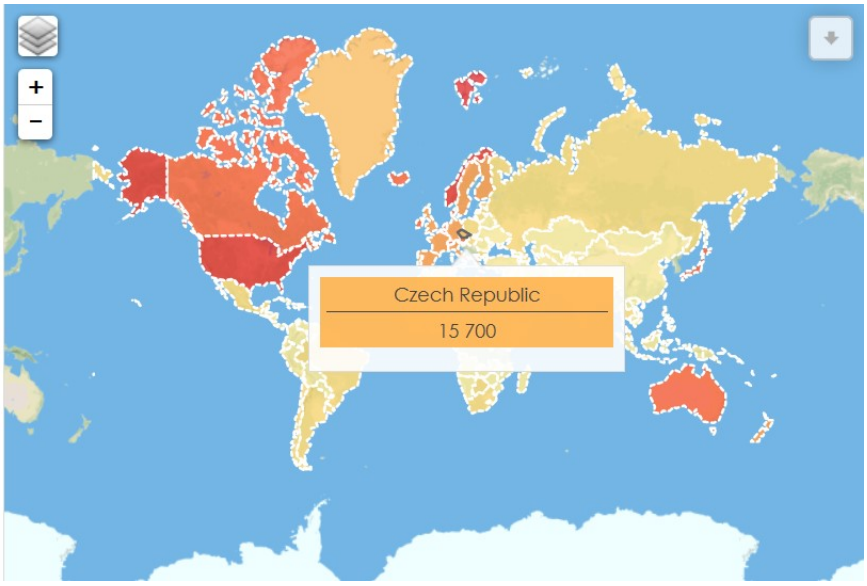
Graf je vizuálním zobrazením dat. BellaDati obsahuje více než 20 různých typů grafů, přičemž každý typ obsahuje celou řadu dalších nastavení. V závislosti na typu je možné vybrat jednu nebo více dimenzí a metrik, které se v grafu zobrazí. Počet dostupných slotů pro dimenze závisí na tom, zda graf obsahuje jednu nebo dvě osy. Důraz je kladen na vizuální stránku. Stejně jako u tabulek je možné upravovat barvy a měnit rozměry. Dále je možné nastavit rozsah os nebo nastavit jako pozadí obrázek.



Obrázek 13: Čárový graf v BellaDati BI (zdroj: autor)

Mapa

Mapa je dalším typem vizuálního zobrazení dat, v tomto případě zaměřené především na geografickou část dat. Mapy se v BellaDati dělí na dva základní druhy – bodové a územní. V případě bodových map se data zobrazují ve formě koláčových grafů pro každý bod mapy. Naproti tomu územní mapy zobrazují data přímo na mapě pro jednotlivé územní celky – regiony, státy atd. Pro své fungování tedy potřebují geografické údaje o tvaru a umístění těchto celků. BellaDati pro ukládání těchto údajů využívá otevřený formát GeoJSON.[12]

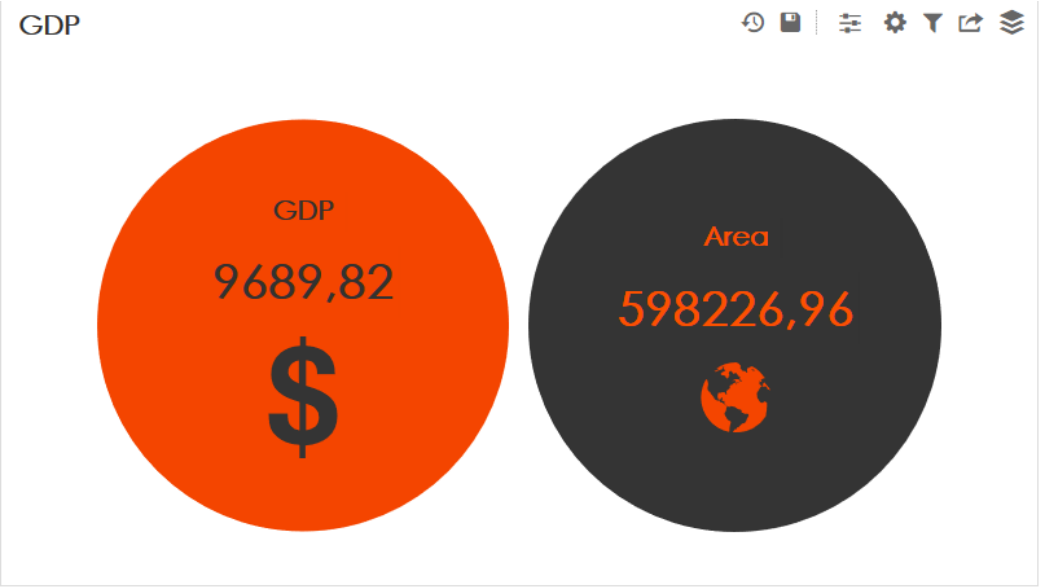


Obrázek 14: Územní mapa v BellaDati BI (zdroj: autor)

[12] geojson.org

KPI popisek

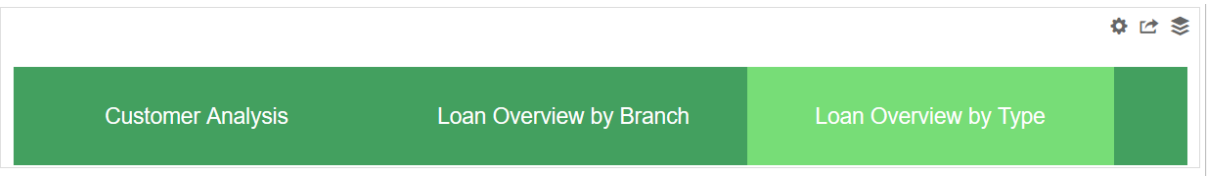
Popisky slouží k zobrazení nejdůležitějších hodnot, například KPI. Jedná se o jednoduché zobrazení jednoho čísla, kterému je možné nastavit specifický vzhled. Kromě základních obdélníkových popisků je možné použít ke zvýraznění i další tvary (hvězda, šestiúhelník, kruh atd.). Dále je možné popisek doplnit o symbol, který může znázorňovat význam zobrazené hodnoty. V rámci jednoho pohledu je možné zobrazit několik popisků najednou.



Obrázek 15: KPI popisek v BellaDati BI (zdroj: autor)

Vlastní obsah

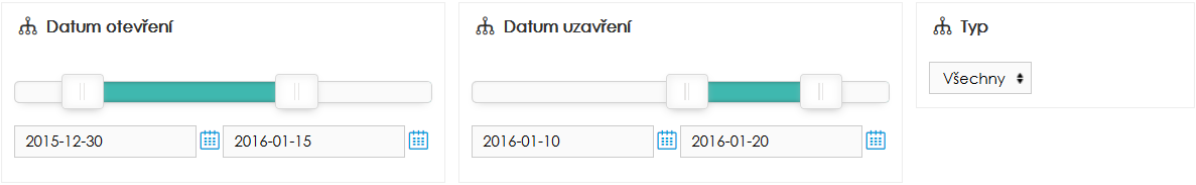
Kromě standardních vizualizací dat je možné do reportu přidat také vlastní obsah. Může jít například o obyčejný text vysvětlující zobrazené údaje. Pokročilejší uživatelé mohou využít možnosti vložit přímo HTML kód, který může obsahovat také JavaScript a kaskádové styly. Díky tomu lze přistupovat i ke klientské vrstvě vestavěného API a vytvářet tak vlastní pohledy. Například lze vytvořit vlastní navigaci na základě seznamu reportů.



Obrázek 16: Vlastní obsah v BellaDati BI - navigace (zdroj: autor)

Filtr

BellaDati nabízí celou řadu způsobů, jak filtrovat zobrazená data. K dispozici jsou jak statické filtry, které lze vytvořit při editaci pohledu, tak i dynamické filtrovací komponenty, pomocí kterých lze zužovat nebo rozšiřovat oblast zobrazených informací. Lze vytvářet globální filtry vztahující se na celý report, ale i individuální filtry náležící ke konkrétnímu grafu nebo tabulce.



Obrázek 17: Filtry v BellaDati BI (zdroj: autor)

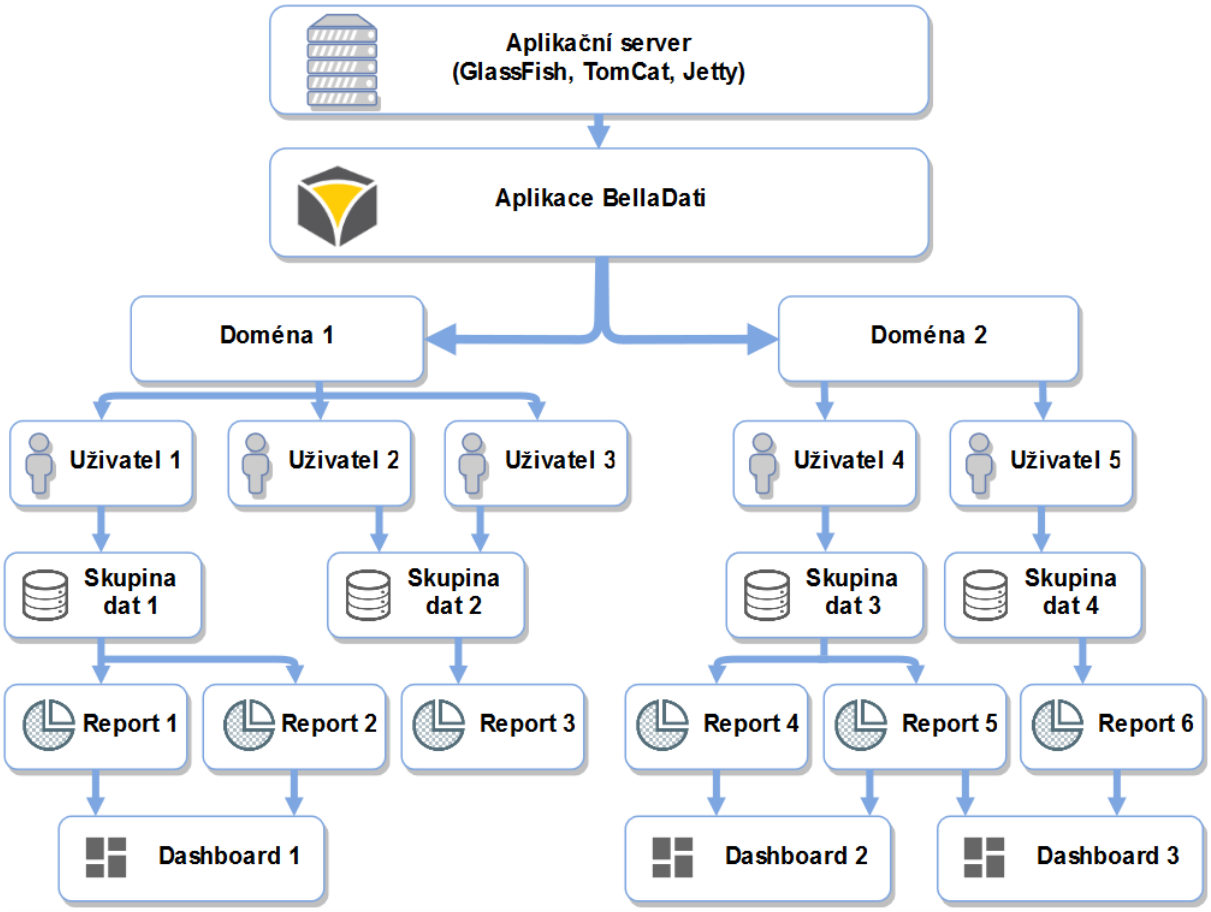
Další funkce reportu

Vyjma samotného vytváření a zobrazování dat umožňuje BellaDati v rámci reportu celou řadu dalších činností:

- export do souboru (PDF, PPT, PNG) nebo e-mailem,
- sdílení s uživateli nebo zveřejnění na webu,
- práce s proměnnými – tzv. „*what if*“ scénáře,
- přidávání komentářů a příloh,
- obnovení předchozí verze reportu nebo pohledu.

7.2.4 Dashboard

Dashboardy slouží k získání rychlého přehledu a zobrazí se vždy jako první po přihlášení uživatele. Svoji strukturou jsou velmi podobné reportům. Opět se jedná o stránku složenou z neomezeného množství pohledů, které pocházejí přímo z reportů (viz Obrázek 18). Pohledy v reportech a dashboardech jsou propojené, díky čemuž jakákoliv změna provedená v reportu se projeví i ve všech dashboardech, které daný pohled obsahují. Dashboardy neumožňují upravovat pohledy, přesto se ale nejedná o statická zobrazení. Je možné pracovat s předpřipravenými rozpadovými cestami a filtry.



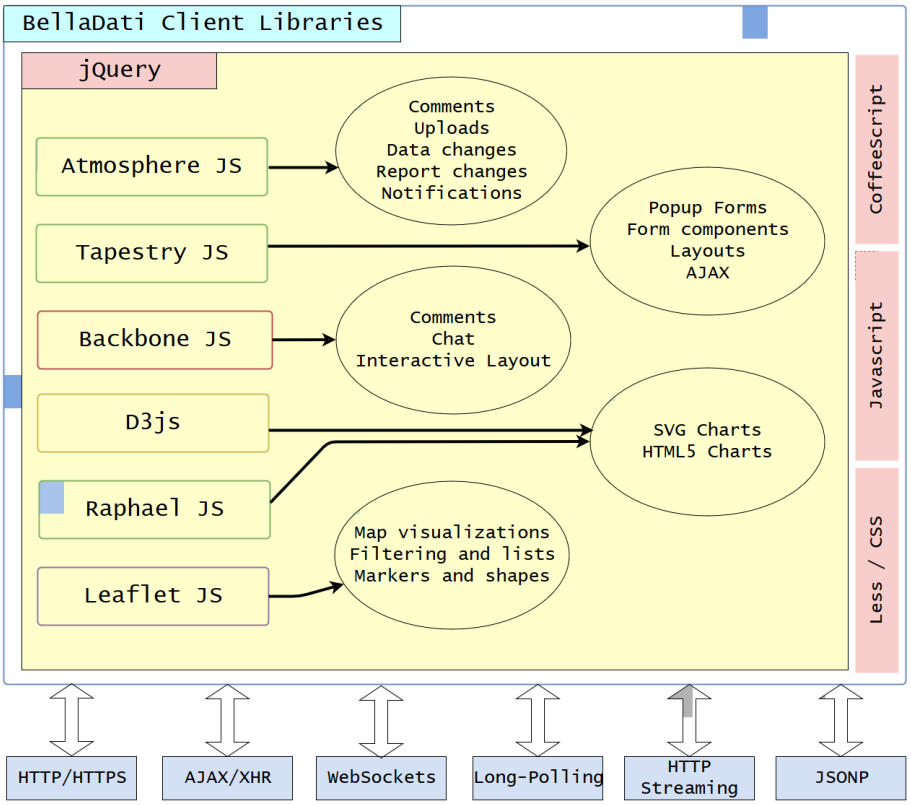
Obrázek 18: Struktura BellaDati BI (zdroj: autor)

Dashboardy lze sdílet a publikovat na Internetu stejným způsobem jako reporty. Dashboard je tak možné sdílet s uživateli, uživatelskými skupinami nebo celou doménou. Lze také povolit veřejný přístup k dashboardu, ke kterému je možné vytvořit tzv. alias. Jedná se o URL adresu ve snadno zapamatovatelném formátu. Dále je možné nastavit pravidelné odesílání dashboardu prostřednictvím e-mailu. Součástí e-mailu může být i příloha, která obsahuje dashboard ve formátu PPT, PDF nebo XLSX. Export do souboru je možné provést i nezávisle na e-mailu.

7.2.5 Technický popis

BellaDati je aplikace naprogramovaná v jazyce Java. Jedná se o serverovou aplikaci, která jako klienta využívá internetový prohlížeč. Je poskytována jak formou cloudové služby, tak i jako on-premise aplikace. Samotná aplikace je napsaná modulárně, díky čemuž je případné doplňování nové funkčnosti velmi snadné.

Webové rozhraní aplikace využívá celou řadou frameworků. Hlavní část rozhraní aplikace je napsána v Tapestry. Tapestry je webový Java framework, který je poskytován jako open-source firmou Apache. Jedná se o jedno z nejelegantnějších a nejefektivnějších řešení pro tvorbu webových aplikací v jazyce Java (Kolesnikov 2008).



Obrázek 19: Klientské knihovny (zdroj: BellaDati 2016)

BellaDati využívá svůj vlastní datový sklad založený na databázi PostgreSQL. Pro analýzu v reálním čase využívá ROLAP a částečně in-memory technologie.

7.3 Metodika testování nástroje BellaDati

BellaDati BI je vyvíjeno agilně. Pro řízení projektu se nepoužívá žádná konkrétní metodika. Procesy probíhající ve firmě ale odpovídají metodice MMSP, a proto pro popis testování použiji právě tuto metodiku. Vzhledem k tomu, že se jedná o neustále se rozvíjející se produkt, který má téměř neomezenou životnost, rozhodl jsem se považovat vývoj každé nové verze (aktualizace) aplikace za samostatný životní cyklus. Variantu, kdy vývoj nové verze je považován za jednu z iterací fáze konstrukce, jsem zamítnul z několika důvodů. Součástí vývoje nové verze je vždy sběr požadavků od zákazníka. Tato etapa je ale v rámci metodiky součástí fáze zahájení. Stejně tak je nutné před vydáním nové verze provést akceptační testování, které

se nachází v poslední fázi zavedení. Jak je patrné, vývoj nové verze nespadá pouze do fáze konstrukce, nýbrž je rozprostře do několika na sebe navazujících fází. Proto je výše zmíněný přístup, kdy každá verze aplikace má svůj vlastní životní cyklus, lepší volbou. Vývojový cyklus verze není přesně daný. Zpravidla trvá přibližně měsíc. Délka cyklu se odvíjí od počtu změn, které nová verze obsahuje. Vývoj je velmi dynamický, například za rok 2015 bylo v interním evidenčním systému vyřešeno více než 1100 chyb a požadavků.

7.3.1 Požadavky

Požadavky jsou hlavním hnacím motorem vývoje BellaDati BI. Z velké části určují směr vývoje, konkrétně přidávání nových funkcí nebo vylepšování těch stávajících. Požadavky pocházejí jak od zákazníků, tak i od vývojářského, analytického a testovacího týmu. Proces jejich zpracování je jiný pro požadavky od zákazníků a pro interní požadavky. V prvním případě jsou požadavky nejprve zákazníky zaneseny do externího systému pro správu požadavků. Následně jsou zpracovány analytickým týmem. Ve výjimečných případech jsou požadavky ihned zamítnuty. Tato situace nastává pouze v případě, kdy se jedná o nerealizovatelné požadavky. Pokud není zamítnut, je ve většině případů nutné upřesnit specifikaci požadavku dodatečnými otázkami, případně je nutné připravit jednoduchý prototyp, především u větších požadavků. V momentě, kdy je specifikace finální, je požadavek přenesen analytickým týmem do interního systému. Pokud se jedná o interní požadavek, je vždy zanesen rovnou do interního systému.

Výstupem každého požadavku by vždy měl být kompletní případ užití umístěný v interním systému pro správu požadavků a chyb. Dle metodiky MMSP by měl každý případ užití obsahovat následující části:

- definice všech funkčních a nefunkčních požadavků, včetně jejich priority,
- požadavky na spolehlivost,
- požadavky na výkonnost – doba odezvy, kapacita apod.,
- podpora – požadavky na udržovatelnost, kompatibilitu atd.,
- specifikace uživatelského rozhraní – grafický styl, rozložení, možnosti personalizace a customizace (Rejnková 2016b).

V rámci metodiky jsou definované i další části, jmenovitě obchodní pravidla a požadavky na dokumentaci. V případě firmy BellaDati ale nejsou součástí případů užití, nýbrž smlouvy mezi firmou a zákazníkem.

7.3.2 Správa požadavků a chyb

Jak jsem již zmínil, BellaDati využívá systém pro správu požadavků a chyb. Konkrétně se jedná o komerční webovou aplikaci Atlassian JIRA. Správa externích a interních požadavků a chyb je oddělená. Do externího systému mají kromě zaměstnanců firmy přístup zákazníci a partneři. Interní JIRA je neveřejná a mohou do ní přistupovat pouze zaměstnanci. Ačkoliv se jedná o oddělené systémy, existuje propojení mezi externími a interními požadavky.

Pro evidenci se používají tzv. *issues*, což lze volně přeložit jako požadavek nebo záležitost. Požadavky lze rozdělit do několika základních kategorií:

- chyba (angl. *bug*) slouží k zaevidování nalezené chyby nebo problému;
- vylepšení (angl. *improvement*) slouží k zaznamenání požadovaného vylepšení nějaké stávající funkce;
- nová funkce (angl. new *feature*) slouží k zadefinování zcela nové funkce, která se zatím v aplikaci nenachází;
- otázka (angl. *question*) se používá téměř výhradně v externím systému a slouží jako jeden ze způsobů oficiální komunikace se zákazníky;
- úkol (angl. *task*) slouží k zaevidování záležitosti, kterou nelze zařadit do výše zmíněných kategorií. Úkol může být také součástí většího požadavku. V tom případě se úkol používá například pro zaevidování požadavku na prototyp.

7.3.3 Testování

Testování je jedná z činností, která se nachází ve všech fází životního cyklu, s výjimkou úvodní zahajovací fáze. V každé fázi má testovací tým ale jinou náplň práce. V následující části práce popíši průběh testování v každé fázi. Vzhledem k zaměření této práce se budu věnovat pouze testování, která vykonávají testeři, analytici nebo zákazníci. Testování prováděné vývojáři, jako například jednotkové testování, zmiňovat nebudu.

Fáze rozpracování

Tato fáze je z hlediska testování nejméně náročná. Testeři zpravidla finalizují úpravy testovacích sad na základě změn, které byly součástí předchozího cyklu. Tyto změny jsou samozřejmě testovány již během předchozího cyklu, ve kterém byly implementovány. Testování ale vzhledem k velmi rychlému vývoji probíhá podle provizorní verze testovací sady, při které se zčásti spoléhá na schopnosti a intuici testera. Z dlouhodobého hlediska je ale vhodnější mít testovací sady přesně definované, například z důvodu změn ve složení testovacího týmu. Proto je dolaďování scénářů během fáze rozpracování velmi důležité. Výstupem z této fáze je tedy finální verze testovacích sad obsahující testovací případy z předchozího cyklu.

Fáze konstrukce

Na základě plánování je pro každou verzi vybráno určité množství změn, které je potřeba implementovat. Pro každou změnu je již vytvořen interní požadavek v systému Atlassian JIRA. Tento požadavek je následně vyřešen vývojářským týmem a předán k testování testerům. Ještě předtím testeři pracují na seznamu testovacích nápadů. *„Seznam testovacích nápadů představuje pracovní produkt vytvářený v rámci přípravy testů, který přehledně zachycuje veškeré možné problémové oblasti vyvíjeného IS/ICT, které by bylo vhodné otestovat. Při jeho tvorbě se vychází primárně ze specifikovaných případů užití, u kterých by měly být testovány nejen jejich úspěšné průchody, ale i chybové stavy (např. při nevyplnění povinných polí, zadání znaků do polí pro čísla apod.).“*(Rejnková 2016a)

Jakmile je požadavek vyřešen a seznam testovacích nápadů dokončen, může testovací tým začít pracovat na testovacích případech. Pro funkce, které byly přidány v aktuálně vyvíjené verzi, je nutné vytvořit zcela nové testovací případy. Pokud se jedná pouze o rozšíření funkčnosti nějaké funkce, stačí tyto změny zapracovat do stávajících případů. Pokud se jedná o funkci, která v libovolné podobě pracuje s datovými skupinami, je nutné též připravit testovací data. Ta by měla být co nejvíce různorodá a měla by pokrývat i nesmyslné a nereálné hodnoty. Testovací data se připravují různými způsoby. První možností je využití datových skupin volně dostupných na Internetu, takzvaných otevřených dat. Příkladem takového zdroje může být například portál Datahub[13]. Druhou možností je vygenerování náhodných dat dle

[13] https://datahub.io

nastavených omezení, například pomocí tabulkového procesoru nebo služby GenerateData[14]. Poslední možností je ruční vytvoření testovacích dat. Tento postup lze využít pouze v případě, kdy pro otestování stačí použít velmi malý vzorek dat. Jakmile jsou testovací scénáře a testovací data připravené, mohou testeři začít testovat jednotlivé požadavky v rámci neveřejné testovací verze aplikace. Pokud nenaleznou chyby, je požadavek uzavřen. V opačném případě je znovuotevřen a předán zpět vývojářům s detailním popisem nalezené chyby. Tento cyklus se opakuje, dokud nejsou všechny požadavky uzavřeny. Tento moment lze označit jako jeden z hlavních milníků testování, neboť testování změn je dokončeno a může se začít s komplexním testováním celé aplikace pomocí testovacích sad.

Testovací sady jsou navrženy tak, aby na sebe navazovaly. Díky tomu je možné otestovat všechny části aplikace, od administrace, přes datové skupiny, reporty a dashboardy, až po otestování sdílení a zabezpečení. Pokud tester objeví nově vzniklou chybu, je nutné ji okamžitě zanést do systému pro správu požadavků. Tuto chybu je nutné co nejdříve opravit, pouze v případě velmi málo závažných chyb je možné odložit jejich opravu až do dalšího životního cyklu. Jakmile je chyba opravena, je nutné znovu projít odpovídající testovací sady. Testuje se také zpětná kompatibilita a funkčnost ve všech prostředích (cloud, on premise, Windows, Linux atd.). V momentě, kdy jsou všechny testovací sady úspěšně dokončeny, je fáze konstrukce dokončena a je dosaženo milníku Provozní způsobilost.

Fáze zavedení

Pokud jsou součástí nové verze i požadavky od zákazníků, provádí se také akceptační testování. Zákazníkovi je poskytnuta tzv. „Release Candidate“ verze. Jedná se o dokončenou, ale ještě nezveřejněnou verzi. Cílem je získat potvrzení od zákazníka, že bylo implementováno opravdu to, co požadoval. Souběžně s tím testeři pracují na doplňování dokumentace, aby byla připravena, až bude aplikace zveřejněna. Jakmile všichni zákazníci potvrdí správnou funkčnost aplikace a dokumentace je dokončena, je možné uvést aplikaci do provozu. Jsou zveřejněny instalátory aktuální verze aplikace a aktualizováno je také cloudové prostředí. Tím je dosaženo závěrečného milníku Nasazení a aktuální životní cyklus končí.

[14] http://www.generatedata.com

7.4 Problémy aktuálního procesu testování

V této kapitole popíšu problémy, které byly identifikovány v procesu testování aplikace BellaDati. U každého problému uvedu důvod jeho vzniku a možnosti, jak ho vyřešit nebo alespoň co nejvíce omezit.

7.4.1 Problém 1 – nedostatek času na testování

Jednou z hlavních komparativních výhod aplikace BellaDati BI je velmi agilní vývoj, který umožňuje firmě velmi rychle reagovat na požadavky zákazníků. Z toho důvodu je čas na otestování aplikace velmi omezený a testovací tým je pod velkým nátlakem. Jedná se především o závěrečné testování testovacích sad na závěru fáze konstrukce. Toto testování je nutné stihnout během několika málo dní, což je velmi náročné, až téměř nemožné. A s dalším rozšiřováním aplikace se bude situace ještě zhoršovat.

Tento problém lze vyřešit zavedením automatizovaného testování. Díky tomu by bylo možné z testovacích sad odstranit velké množství testů, které lze převést do automatizované podoby. Jedná se o základní průchody aplikací, testování konfigurace, vytváření uživatelů apod. První fází řešení tohoto problému je výběr vhodného nástroje. Následně je nutné sestavit seznam testů, které se budou automatizovat. Tyto testy poté musí být vytvořeny a integrovány do procesu testování.

7.4.2 Problém 2 – repetitivnost testování

Agilní vývoj aplikace s krátkým životním cyklem každé verze má jednu nevýhodu – testuje se téměř pořád a neustále to samé. Po určité době se tak testování stává repetitivní a unavující. Pozornost testerů klesá a jsou daleko více náchylní k chybám.

Prvním řešením je častěji obměňovat složení testovacího týmu. Nově přijatí testeři sníženou pozorností netrpí. Nevýhodou tohoto řešení je nutnost neustále shánět nové zaměstnance a především je neustále zaučovat. Zaučování musí provést jeden ze zkušených zaměstnanců, který se tak nemůžu věnovat své standardní práci a dochází tak k poklesu produktivity práce. Toto řešení tedy není příliš vhodné. Druhou možností je zautomatizovat nejvíce repetitivní testy, čímž by testerům odpadla „nejhorší“ část práce. Toto řešení je i přes svoji náročnost lepší volbou, neboť je v souladu s řešením předchozího problému.

7.4.3 Problém 3 – velké množství „zavedených“ chyb

Třetím identifikovaným problémem je relativně velké množství „zavedených“ chyb. Tímto pojmem označuji chyby, které vzniknou při vývoje funkcí do nových verzí. Jedná se tedy o chyby, které v předchozí verzi neexistovaly a způsobily je změny, které byly provedeny kvůli implementaci změn v současné verzi. Za poslední tři roky bylo do systému pro správu chyb a požadavků zaneseno více než 100 chyb přiřazených k jedné z testovacích verzí. Jedná se o chyby, které vznikly právě při vývoji nové verze. Ačkoliv se na první pohled nejedná o příliš velké číslo, jejich oprava značně prodlužuje vývoj. Velká část těchto chyb je totiž odhalena až při závěrečném testování. Pokud je při něm objevena chyba, je nutné ji opravit a znovu otestovat danou sekci testovacích sad. Jak oprava, tak opakované testování stojí nějaký čas, který by jinak mohl být věnován něčemu jinému.

Tento problém je nemožné zcela eliminovat, neboť chyby při vývoji budou vznikat vždy. Lze ale snížit jejich dopad. Pokud by se při každém sestavení aplikace, které probíhá několikrát denně, spustila kompletní sada automatizovaných testů, značné množství chyb by bylo odhaleno dříve a díky tomu by bylo možné je opravit ještě před finálním testováním. V současné době se takto spouští pouze jednotkové testy, které ale neodhalí všechny chyby. Pokud by se takto spouštěli i automatizované testy pomocí nástroje Selenium, byla by šance na včasné odhalení chyby daleko vyšší.

7.4.4 Problém 4 – nedůkladné testování kompatibility s prohlížeči

BellaDati BI je webovou aplikací, což znamená, že její klientská část běží kompletně ve webovém prohlížeči. Z toho důvodu je velmi důležité testovat kompatibilitu se všemi hlavními prohlížeči. Dle webové služby StatCounter (údaje pro duben 2016) se jedná o prohlížeče Google Chrome (60,9 %), Mozilla Firefox (15,29 %), Internet Explorer 11 (9,67 %), Safari (2,26 %) a Microsoft Edge (2,26 %) (StatCounter 2016). Navíc, vzhledem k zaměření na podnikovou klientelu, je nutné zajišťovat správnou funkčnost i v prohlížečích Internet Explorer 8 a 9. Jedná se tedy o velké množství různých prohlížečů, ve kterých je nutné otestovat celou aplikaci. To je bohužel vzhledem k velmi omezenému množství času nemožné, a proto se všechny testovací sady procházejí pouze v jednom prohlížeči. Ostatní prohlížeče jsou otestovány pouze pomocí jedné speciální sady na otestování kompatibility.

Řešením je opět automatizace testování. Jednou vytvořenou sadu automatizovaných testů lze spustit na téměř libovolném prohlížeči. Tím by se úroveň testování kompatibility velmi zvýšila. Navíc lze využít funkce Selenium Grid a testy spouštět na více prohlížečích najednou, co by přineslo další úsporu času. Problémem tohoto řešení je nulová vizuální kontrola aplikace. Úspěšné dokončení automatizovaných testů neznamená, že se všechny obrazovky aplikace správně zobrazily. I tento problém ale vyřešit. Existují rozšiřující knihovny pro Selenium, které umožňují porovnávat snímky stránek mezi různými prohlížeči. Příkladem může být například knihovna, kterou vytvořil Pavel Sklenář v rámci své diplomové práce (Sklenář 2012).

7.4.5 Shrnutí problémů aktuálního procesu testování

Jak možné vidět ve shrnující tabulce (viz Tabulka 21), všechny identifikované problémy přímo nebo nepřímo souvisí s aktuální absencí automatizovaného testování. Na základě tohoto poznatku bylo rozhodnuto o jeho zavedení. Postup zavádění je popsán v další kapitole a může posloužit jako inspirace pro další firmy.

Tabulka 21: Seznam problémů aktuálního procesu testování

Problém	Název	Závažnost	Řešení
Problém 1	nedostatek času na testování	Vysoká	Zavedení automatizovaných testů
Problém 2	Repetitivnost testování	Nízká	Zavedení automatizovaných testů
Problém 3	Velké množství „zavedených“ chyb	Střední	Spouštění automatizovaných testů po každém sestavení aplikace
Problém 4	Nedůkladné testování kompatibility s prohlížeči	Střední	Paralelizované spouštění automatizovaných testů ve všech prohlížečích

7.5 Postup zavádění automatizovaného testování ve společnosti BellaDati

V předchozí kapitole jsem nastínil důvody, které vedly k zavedení automatizovaného testování ve společnosti BellaDati. V této kapitole naváži a popíšu samotný proces zavádění. Kapitole je rozdělena do několika částí. Výběr a implementace řešení probíhal dle metodiky MMSP, respektive dle její rozšířené verze, která přidává mimo jiné podporu pro automatizované testování. Autorem tohoto rozšíření je Robert Rojo (Rojo 2015). Ten ve své práci doplnil metodiku o úlohy zaměřené na správu testů, tvorbu automatizovaných testů a správu uživatelské dokumentace. Pro nasazení automatizovaného testování ve společnosti BellaDati byly klíčové tyto úlohy:

- analýza a tvorba automatizovaných testů, včetně výběru řešení pro automatizaci testů,
- spouštění automatizovaných testů,
- vyhodnocování automatizovaných testů (Rojo 2015, s. 42).

Provedení jednotlivých úloh je popsáno v následujících kapitolách. Každá kapitola odpovídá jedné úloze, pouze první úloha je rozdělena na dvě části. V první části je popsán výběr řešení pro automatizaci testů. V druhé části je popsána analýza a tvorba automatizovaných testů.

Kromě nových úloh je součástí rozšířené metodiky také definice nové role Správce automatizace. Ten je zodpovědný za výše zmíněné úlohy. Důvodem vzniku této role je potřeba speciálních znalostí, které ale nejsou vyžadovány pro roli testera (Rojo 2015, s. 42).

7.5.1 Výběr řešení pro automatizaci testů

Proces výběru řešení je bohužel v metodice MMSP popsán velmi stručně. U úlohy Analýza a tvorba automatizovaných testů chybí všechny výstupy a vstupy související s výběrem nástroje. Z toho důvodu byly stanoveny vlastní pracovní produkty. Jako vstup pro výběr řešení byl použit seznam požadovaných vlastností, které by měl nástroj splňovat. Výstupem úlohy je pak popis zvoleného nástroje pro automatizaci. Za obě činnosti byl zodpovědný správce automatizace.

Výběr kritérií probíhal formou diskuze mezi vývojáři, testery a správcem automatizace. Důraz byl kladen na technické i ekonomické důvody. Výsledkem diskuze byla tabulka se seznamem kritérií a důvodem pro jejich zvolení (viz Tabulka 22).

Tabulka 22: Seznam kritérií výběru nástroje pro automatizace testování (zdroj: autor)

ID kritéria	Kritérium	Důvod
K1	Bezplatná licence (open source)	Společnost BellaDati se v rámci snížení nákladů snaží využívat co nejčastěji volně dostupné nástroje.
K2	Použití jazyka Groovy pro zápis testů	Jazyk Groovy se již používá v rámci vývoje nástroje BellaDati BI a velká část týmu tedy tento jazyk ovládá.
K3	Kvalitní dokumentace	Aby byla implementace automatizovaného testování co nejrychlejší, je nutné mít k dispozici již připravenou detailní dokumentaci.
K4	Podpora funkce Selenium Grid	Do budoucna se plánuje možnost využití paralelního spouštění testů na více strojích najednou.

Na základě těchto kritérií proběhl průzkum trhu s nástroji pro automatizaci testování. Zadaným kritériím odpovídal pouze nástroj Geb, který byl vybrán jako vhodný nástroj. Popis tohoto nástroje je uveden již v kapitole 6.1.17 Geb, a proto ho zde znovu uvádět nebudu.

Použití Gebu má celou řadu výhod. Velkým přínosem je možnost využít objektový přístup, což značně usnadňuje údržbu testů a přináší možnost znuvupoužitelnosti kódu. Funkce jako dědičnost nebo moduly umožňují definovat stránky a testy pouze na jednom místě, díky

čemuž je jejich následná úprava daleko snazší. Objekty lze vytvářet i u standardní implementace nástroje Selenium Webdriver, Geb ale tento proces zjednodušuje.

Další výhodou je použití *Navigator* API pro navigaci na stránce a vyhledávání obsahu. Geb používá pro navigaci podobný přístup jako knihovna jQuery, díky čemuž je pro řadu vývojářů práce s tímto nástrojem daleko snazší. Spolu se zápisem testů v jazyce Groovy se z mého pohledu jedná o největší výhodu, neboť testy jsou v porovnání s nástrojem Selenium daleko přehlednější a čitelnější.

7.5.2 Analýza a tvorba automatizovaných testů

Cílem této úlohy byl výběr první sady testů a jejich následné převedení do automatizované podoby. Jako vstup byly použity stávající sady testů, z kterých se se vybíraly jednotlivé testovací případy. Pro první fázi zavádění automatizovaných testů byly nakonec vybrány základní testy každého modulu aplikace. Hlavní výhodou tohoto přístupu je vytvoření kompletní kostry testů, která bude pokrývat všechny stránky aplikace alespoň na základní úrovni. V dalších fázích tak bude možné tyto testy snadno rozšiřovat. Jak jsem již zmínil, kód je napsán v jazyce Groovy za využití aplikačního rámce Geb. Pro zápis testů je využit testovací framework TestNG, opět z důvodu jeho použití i v jiných případech v rámci společnosti.

Při vytváření testů se využívalo objektového přístupu, kdy ke každé stránce náleží dva zdrojové soubory. První z nich obsahuje objektovou definici stránky, která je následně použita v druhém souboru, který obsahuje samotné testy. V rámci první fáze byly implementovány následující objekty a testy:

- *SetupPage* + *SetupTest* slouží k otestování úvodní konfigurace na instalaci aplikace;
- *LoginPage + LoginTest* slouží k otestování přihlašování uživatele do aplikace;
- *DomainsPage* + *DomainSetupTest* slouží k otestování vytváření a konfigurace domén;
- *UserPage + UserSetupTest* slouží k otestování vytváření a editace uživatelských profilů;
- *DataSetsPage + DataSetTest* slouží k otestování vytváření datových skupin;
- *ReportPage + ReportTest* slouží k otestování vytváření a editace reportů;
- *DashboardPage + DashboardTest* slouží k otestování vytváření a editace dashboardů.

Kromě testů bylo nutné připravit také testovací data. Ve většině případů bylo možné použít data používaná u manuálního testování. Jedním z požadavků bylo ale testování formulářových

polí. Cílem tohoto testování je kontrola, zda určité textové řetězce zadané do formulářů nemohou narušit správnou funkčnost aplikace. Příkladem může být například JavaScript Injection. „Princip útoku spočívá ve vložení vlastního javascriptového kódu do HTML stránky, která se zobrazí jinému uživateli. Například místo textu příspěvku do diskusního fóra můžeme vložit tag <skript>, který se pak vloží do stránky všem čtenářům tohoto příspěvku.“ (Kruliš 2010) Velmi dobrým zdrojem, který byl v tomto případě použit, je *„Big List of Naughty Strings“*, což je neustále rozšiřující se seznam textových řetězců, u kterých je vysoká pravděpodobnost, že způsobí chybu. Tento seznam je veřejně dostupný na portálu GitHub.[15] Ze seznamu se náhodně vybere jeden řádek a ten se vloží do aktuálního pole, což může být například popis domény nebo název datové skupiny (viz Výpis kódu 4).

```
1.  URL url = this.getClass().getResource("/blns.txt");
2.  List<String> fileLines = new File(url.getFile()).readLines();
3.  int count = fileLines.size();
4.  int randomNumber = random.nextInt(count+1 - 0) + 0
5.  String fileContents = fileLines.get(randomNumber);
```

Výpis kódu 4: Příklad načítání náhodného řádku v jazyce Groovy

Pro kompilace a spouštění testů je použit integrační nástroj Gradle. Ten zajišťuje automatické stažení podpůrných knihoven při spuštění testů. Dále jsou pomocí tohoto nástroje definovány úlohy (angl. *task*), které umožňují spouště různé varianty testů. Testy je možné spouštět buď proti aktuálně běžící aplikaci BellaDati BI nebo lze spustit novou instanci aplikace spolu s testy. Dále je možné specifikovat, v jakých prohlížečích se mají testy spustit. Lze je spustit ve všech prohlížečích najednou nebo vybrat jeden konkrétní. Příklady jednotlivých příkazů lze nalézt níže (viz Výpis kódu 5). Příklad samotného souboru *build.gradle* lze nalézt v příloze. Může posloužit jako inspirace pro čtenáře této práce (viz Příloha B).

```
1.  ./gradlew chromeTest //spuštění testů v prohlížeči Chrome
2.  ./gradlew firefoxTest //spuštění testů v prohlížeči Firefox
3.  ./gradlew test //spuštění všech testů
4.  ./gradlew -Dport=8081 -Dhost=192.168.1.10 chromeTest
5.  //spuštění testů v prohlížeči Chrome oproti aplikaci BellaDati běžící na adrese
6.  //192.168.1.10
7.  ./gradlew startBellaDati firefoxTest
8.  //spuštění lokální instance BellaDati a následné spuštění testů v prohlížeči
9.  //Firefox oproti této instanci
```

Výpis kódu 5: Příkazy pro spuštění testů pomocí nástroje Gradle

[15] https://github.com/minimaxir/big-list-of-naughty-strings

7.5.3 Spouštění automatizovaných testů

Cílem úlohy je spuštění automatizovaných testů, sběr výsledků a jejich interpretace. Výstupem této úlohy jsou výsledky testů v podobě, ve které je možné vyhodnotit jejich přínos. V době psaní této práce byla základní sada testů spouštěna pravidelně při každém sestavení aplikace. Výsledky každého spuštění byly ukládány za účelem následného zhodnocení. Po vydání každé verze byl vytvořen testovací report, jenž obsahoval tyto metriky:

- počet chyb opravených v dané verzi;
- počet chyb zavedených v dané verzi;
- počet chyb odhalených pomocí automatizovaného testování;
- počet člověkohodin, jež byly potřeba na opravu chyb odhalených pomocí automatizovaného testování.

7.5.4 Vyhodnocení automatizovaných testů

Cílem této úlohy je vyhodnocení výsledků automatizovaných testů. Toto vyhodnocení by mělo vést k rozhodnutí o dalším vývoji automatizovaného testování. V případě pozitivního výsledku by se automatizované dále rozšiřovaly, aby pokrývaly stále větší část aplikace. V opačném případě by došlo k přehodnocení metodiky a implementace testů, případně ke kompletnímu zrušení automatizovaných testů, pokud by byl jejich přínos nulový. Vyhodnocování testů provádí Správce automatizace.

Jak jsem zmínil v předchozí kapitole, za účelem vyhodnocení docházelo ke sběru dat vybraných metrik. Souhrnné výsledky je možné nalézt v tabulce níže (viz Tabulka 23). Celkem bylo pomocí automatizovaného testování odhaleno 15 chyb z celkových 59. Přibližně čtvrtina chyb byla tedy zjištěna pomocí těchto testů. Podstatnějším faktem je odhalení 79 % všech zavedených chyb. Díky tomu, že těchto 15 chyb bylo odhaleno dříve než při závěrečném testování aplikace, bylo možné je opravit a otestovat ještě během fáze konstrukce. Na opravu a otestování těchto chyb bylo potřeba celkem 98 člověkohodin. Pokud by došlo k odhalení chyb až při závěrečném testování, byly by životní cykly posledních šesti verzí BellaDati BI v součtu delší minimálně o právě tuto dobu. Pravděpodobně by ale došlo k daleko většímu zpoždění, neboť by došlo k přerušení závěrečného testování a některé testovací sady by bylo nutné otestovat znovu.

Tabulka 23: Výsledky automatizovaného testování (zdroj: autor)

Verze BellaDati BI	Celkový počet chyb	Počet zavedených chyb	Počet zavedených chyb odhalených AT	Počet ČH potřebných na opravu a otestování chyb odhalených automatizovaným testováním
2.7.15.7	7	2	2	14
2.7.15.6	6	3	2	18
2.7.15.5	4	1	1	6
2.7.15.4	27	5	4	28
2.7.15.3.1	1	0	0	0
2.7.15.3	14	8	6	32

Celkově je tedy možné zhodnotit nasazení automatizovaných testů jako úspěšné. Z toho důvodu bude společnost BellaDati pokračovat s jejich rozšiřováním a údržbou. Výsledkem by měla být kompletní sada testů, díky které bude možné včas odhalit ještě větší procento chyb.

8 Závěr

Hlavním cílem a přínosem této práce bylo představení pokročilých možností práce s testovacím nástrojem Selenium WebDriver. V úvodu práce je uvedena rešerše dostupných zdrojů, které se týkají nástroje Selenium nebo automatizovaného testovaní obecně. Dále jsem také stručně shrnul testování softwaru a obecně popsal nástroj Selenium. Hlavního cíle bylo dosaženo prostřednictvím několika dílčích cílů. Prvním cílem bylo vytvoření přehledu pokročilých funkcí nástroje Selenium WebDriver. Tohoto cíle bylo dosaženo popsáním funkcí Selenium RemoveWebDriver a Selenium Grid. V obou případech byly také uvedeny příklady použití a konfigurace těchto funkcí.

Druhým dílčím cílem bylo vytvoření přehledu dostupných rozšíření pro nástroj Selenium WebDriver. Tento cíl byl rozdělen do několika kapitol dle typu rozšíření. V první části byly popsány aplikační rámce, které využívají aplikační rozhraní WebDriver API a rozšiřují možnosti nástroje Selenium WebDriver tím, že přidávají dodatečné funkce, případně výrazně zjednodušují použití stávajících funkcí. V druhé části jsou popsány softwarové kontejnery a možnosti jejich využití spolu s nástrojem Selenium. Tyto kontejnery jsou velmi odlehčenou formou virtualizace a umožňují zabalit aplikace a služby do balíčků a ty pak snadno distribuovat a spouštět na libovolném kompatibilním prostředí. V práci jsou popsány platformy Docker a Turbo. Dále jsou popsány možnosti integrace s nástroji pro automatizaci a průběžnou integraci. Tyto nástroje usnadňují vývoj, správu a spouštění testů, přičemž jejich význam roste spolu velikostí týmu a rozsahem testů. Poslední kapitole v rámci tohoto dílčího cíle se věnuje službám pro Selenium. Jedná se o služby, které umožňují spouštět testy vzdáleně na hostovaném prostředí. Díky tomu firmy nemusejí vlastnit a udržovat vlastní testovací stroje, čímž lze výrazně snížit náklady na automatizované testování. Tímto je možné označit přehled za dokončený, čímž byl naplněn druhý dílčí cíl.

Třetím dílčím cílem byla případová studie ve firmě BellaDati, která se zabývá vývojem nástroje Business Intelligence. Tento cíl byl splněn popisem výběru a použití vybraných nástrojů v rámci firmy. Pro případovou studii byla použita rozšířená metodika MMSP. Součástí tohoto cíle bylo také ověření možnosti reálného využití této metodiky v rámci menšího podniku. Metodiku lze označit jako vhodnou s jednou výhradou. V metodice bohužel chybí podrobnější popis postupu výběru nástroje pro automatizaci testování.

Posledním cílem této diplomové práce bylo vytvoření uživatelské příručky k aplikačnímu rámci Geb, který byl využit v rámci případové studie. Příručka popisuje základní principy tohoto rámce a obsahuje návod, jak používat jednotlivé funkce, které Geb nabízí. I tento cíl byl tedy splněn.

Celkově lze tedy označit všechny cíle za splněné. Tato diplomová práce obsahuje dostatek informací pro vytvoření přehledu o pokročilých možnostech testování pomocí aplikačního rámce Selenium WebDriver. Přesto existují témata, prostřednictvím kterých je možné tuto práci rozšířit či na ni navázat. Příkladem může být vytvoření detailního porovnání všech dostupných aplikačních rámců pro nástroj Selenium WebDriver. Vzhledem k velikosti trhu s těmito nástroji se jedná o samostatné téma, a proto není součástí této diplomové práce.

Slovník

Selenium	Selenium je balík nástrojů, který slouží primárně k automatizaci testování webových aplikací. Základním principem testování pomocí Selenia je výběr určitého prvku webové stránky a jeho porovnání s předpokládaným výsledkem.(ANON. 2015b)
Gradle	Gradle je nástroj a jazyk pro automatizaci sestavování aplikací (tzv. build). Je založený na jazyku Groovy.(Gradle Inc. 2015)
Groovy	Groovy je objektově orientovaný dynamický programovací jazyk pro platformu Java. Jeho výhodou oproti Javě je volnější syntaxe.(ANON. 2015a)
JUnit	JUnit je aplikační rámec pro jednotkové testy v jazyce Java.
TestNG	TestNG je testovací aplikační rámec inspirovaný aplikačním rámcem JUnit, který poskytuje podporu pro více typů testů.
Apache Ant	Apache Ant je nástroj pro automatizaci sestavování aplikací.
ISO	ISO je zkratka pro Mezinárodní organizaci pro normalizaci, která se zabývá tvorbou mezinárodních norem ISO.
IEC	IEC je zkratka pro Mezinárodní elektronickou komisi, jež vypracovává mezinárodní normy pro elektroniku, sdělovací techniku a podobné obory.
Únik paměti (memory leak)	Únik paměti označuje moment, kdy dojde k chybě při alokaci paměti. Chyba v aplikaci způsobí, že se neuvolní již nevyužívaná část operační paměti.
Atlassian JIRA	JIRA je nástroj pro řízení projektů a evidenci chyb při vývoji softwaru.
Release Candidate	Release Candidate označuje testovací verzi aplikace, která je již téměř připravena na finální zveřejnění. Jde o verzi, která je vyladěná více než beta verze, ale z různých důvodů ji nelze označit jako finální verzi.

Seznam použité literatury

ANON., 2013. *ISO/IEC/IEEE 29119-1:2013* [online]. 1. září 2013. B.m.: ISO/IEC/IEEE. [vid. 21. listopad 2015]. Dostupné z: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=45142

ANON., 2015a. A multi-faceted language for the Java platform. *The Groovy programming language* [online] [vid. 28. říjen 2015]. Dostupné z: http://www.groovy-lang.org/

ANON., 2015b. Introduction — Selenium Documentation. *Selenium - Web Browser Automation* [online] [vid. 28. říjen 2015]. Dostupné z: http://www.seleniumhq.org/docs/01_introducing_selenium.jsp

AVASARALA, Satya, 2014. *Selenium WebDriver practical guide interactively automate web applications using Selenium WebDriver* [online]. Birmingham, UK: Packt Pub. [vid. 27. září 2015]. ISBN 978-1-78216-886-7. Dostupné z: http://proquest.safaribooksonline.com/?fpi=9781782168850

BELLADATI, 2015. Data Sources - BellaDati v2.7. *BellaDati Support* [online] [vid. 23. leden 2016]. Dostupné z: http://support.belladati.com/doc/Data+Sources

BERGLUND, Tim a Matthew MCCULLOUGH, 2011. *Building and testing with Gradle*. Sebastopol, CA: O'Reilly Media, Inc. ISBN 978-1-4493-0463-8.

BURNS, David, ed., 2012. *Selenium 2 testing tools: beginner's guide ; learn to use Selenium testing tools from scratch*. 2. Aufl. Birmingham: Packt Publ. Learn by doing: less theory, more results. ISBN 978-1-84951-830-7.

CLOUDBEAT LIMITED, 2015. Oxygen. *Oxygen Server* [online] [vid. 8. duben 2016]. Dostupné z: http://docs.oxygenhq.org/server-intro-usage.html

CODECEPTION, 2016. 10-WebServices. *Codeception - Documentation* [online] [vid. 10. duben 2016]. Dostupné z: http://codeception.com/docs/10-WebServices#.VwpUWEeooU5

CODECEPTJS, 2016. AngularJS Testing. *CodeceptJS* [online] [vid. 10. duben 2016]. Dostupné z: http://codecept.io/angular/

DALEY, Luke, Marcin ERDMANN a Erik PRAGT, 2015. The Book Of Geb. *Gebish* [online] [vid. 27. září 2015]. Dostupné z: http://www.gebish.org/manual/current/

DATAGAPS, 2016. Basics of BI testing : A definitive guide. *Datagaps* [online] [vid. 19. duben 2016]. Dostupné z: http://www.datagaps.com/concepts/bi-testing

DAVID, Bienvenido, 2016. Selenium 2 (a.k.a Selenium WebDriver) Is Released. *InfoQ* [online] [vid. 3. duben 2016]. Dostupné z: http://www.infoq.com/news/2011/07/Selenium-2

DAVISON, Daniel, 2015. SeleniumHQ/selenium. *GitHub* [online] [vid. 5. duben 2016]. Dostupné z: https://github.com/SeleniumHQ/selenium

DESIKAN, Srinivasan, 2006. *Software Testing: Principles and Practice*. B.m.: Pearson Education India. ISBN 978-81-7758-121-8.

DUPAUL, Neil, 2013. Static Testing vs. Dynamic Testing. *Veracode* [online] [vid. 29. březen 2016]. Dostupné z: https://www.veracode.com/blog/2013/12/static-testing-vs-dynamic-testing

DUSTIN, Elfriede, Jeff RASHKA a John PAUL, 1999. *Automated Software Testing: Introduction, Management, and Performance: Introduction, Management, and Performance*. Reading, Mass: Addison-Wesley Professional. ISBN 978-0-201-43287-9.

GOSSELIN, Don, 2010. *JavaScript*. 5th edition. Boston, Mass.: Cengage Learning, Inc. ISBN 978-0-538-74887-2.

GRADLE INC., 2015. Chapter 1. Introduction. *Gradle* [online] [vid. 28. říjen 2015]. Dostupné z: https://docs.gradle.org/current/userguide/introduction.html

GUNDECHA, Unmesh, 2012. *Selenium testing tools cookbook over 90 recipes to build, maintain, and improve test automation with Selenium WebDriver* [online]. Birminghan, UK: Packt Pub. [vid. 27. září 2015]. Dostupné z: http://site.ebrary.com/id/10632715

GUNDECHA, Unmesh, 2013. *Instant Selenium testing tools starter a short, fast, and focused guide to Selenium Testing tools that delivers immediate results* [online]. Birmingham, U.K.: Packt Publishing [vid. 27. září 2015]. ISBN 978-1-78216-514-9. Dostupné z: http://site.ebrary.com/id/10695793

GURU99, 2015a. 100 Types of Software Testing You Never Knew Existed. *Guru99* [online] [vid. 21. listopad 2015]. Dostupné z: http://www.guru99.com/software-testing.html

GURU99, 2015b. Automated Testing: Process, Planning, Tool Selection. *Guru99* [online] [vid. 30. prosinec 2015]. Dostupné z: http://www.guru99.com/automation-testing.html

GURU99, 2015c. Free Selenium Tutorials. *Guru99* [online] [vid. 27. září 2015]. Dostupné z: http://www.guru99.com/selenium-tutorial.html

GURU99, 2015d. Introduction to Selenium. *Guru99* [online] [vid. 3. leden 2016]. Dostupné z: http://www.guru99.com/introduction-to-selenium.html

HAUPT, Miroslav, 2009. *Data lineage* [online]. Brmo. Diplomová práce. Masarykova univerzita. Dostupné z: http://is.muni.cz/th/98771/fi_m/98771_diplomka.pdf

HLAVA, Tomáš, 2012. Zátěžové testy. *Testování softwaru* [online] [vid. 31. březen 2016]. Dostupné z: http://testovanisoftwaru.cz/testing/zatezove-testy-software/

CHAUDHARI, Rahul, 2013. Qualitia's Scriptless Software Test Automation Accelerates Time to Market [online]. 21.5.

IEEE, 1990. *IEEE Standard 610.12-1990* [online]. 1990. B.m.: IEEE Standard Glossary of Software Engineering Terminology. [vid. 29. březen 2016]. Dostupné z: https://standards.ieee.org/findstds/standard/610.12-1990.html

INMAN-SEMERAU, Luke, 2016. SeleniumHQ/selenium. *GitHub* [online] [vid. 7. duben 2016]. Dostupné z: https://github.com/SeleniumHQ/selenium

ISTQB GLOSSARY WORKING GROUP, 2015. *Standard Glossary of Terms Used in Software Testing* [online]. 26. březen 2015. [vid. 21. listopad 2015]. Dostupné z: http://www.istqb.org/downloads/finish/20/213.html

JAKUBIČKOVÁ, Nela, 2011. *Návrh metodiky testování BI řešení* [online]. Praha. Diplomová práce. Vysoká škola ekonomická v Praze. Dostupné z: https://isis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=32004

JENKINS, Nick, 2008. *A Software Testing Primer* [online]. 2008. [vid. 30. březen 2016]. Dostupné z: http://www.nickjenkins.net/prose/testingPrimer.pdf

JORGENSEN, Paul C., 2013. *Software Testing: A Craftsman's Approach, Fourth Edition*. 4 edition. Boca Raton, Florida: Auerbach Publications. ISBN 978-1-4665-6068-0.

KANER, Cem, 2006. *Exploratory Testing* [online]. 17. listopad 2006. [vid. 21. listopad 2015]. Dostupné z: http://www.kaner.com/pdfs/ETatQAI.pdf

KARHU, Katja, Tiina REPO, Ossi TAIPALE a Kari SMOLANDER, 2009. Empirical Observations on Software Testing Automation [online]. s. 201–209. Dostupné z: doi:10.1109/ICST.2009.16

KHAN, Daniel, 2015. Why Node.js is hitting the big time in Enterprise Markets. *about:performance* [online]. [vid. 9. duben 2016]. Dostupné z: http://apmblog.dynatrace.com/2015/04/09/node-js-is-hitting-the-big-time-in-enterprise-markets/

KOLESNIKOV, Alexander, 2008. *Tapestry 5: Building Web Applications : a Step-by-step Guide to Java Web Development with the Developer-friendly Apache Tapestry Framework*. B.m.: Packt Publishing Ltd. ISBN 978-1-84719-308-7.

KRULIŠ, Martin, 2010. Javascript Injection – způsoby útoku a obrana. *Interval.cz* [online]. [vid. 17. duben 2016]. Dostupné z: https://www.interval.cz/clanky/javascript-injection-%E2%80%93-zpusoby-utoku-a-obrana/

LEWIS, William E., 2008. *Software Testing and Continuous Quality Improvement, Third Edition*. B.m.: CRC Press. ISBN 978-1-4398-3436-7.

LÍZNER, Tomáš, 2014. *Využití automatizace testování z hlediska nákladů a přínosů* [online]. Praha. Diplomová práce. Vysoká škola ekonomická v Praze. Dostupné z: https://isis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=46952

MAZOCH, Břetislav, 2012. *Funkční testování webových aplikací* [online]. Brno [vid. 28. říjen 2015]. Bakalářská práce. Masarykova univerzita. Dostupné z: https://is.muni.cz/th/325233/fi_b/bmazoch-bp.pdf

NICKLAS, Jonas, 2016. jnicklas/capybara. *GitHub* [online] [vid. 8. duben 2016]. Dostupné z: https://github.com/jnicklas/capybara

OXFORD DICTIONARIES, 2015. Definition of test. *Oxford Dictionaries* [online] [vid. 21. listopad 2015]. Dostupné z: http://www.oxforddictionaries.com/definition/english/test

PATTON, Ron, 2002. *Testování softwaru*. Praha: Computer Press. ISBN 978-80-7226-636-4.

PIETRIK, Michal, 2012. *Automatizované testování webových aplikací* [online]. Brno. Diplomová práce. Masarykova univerzita. Dostupné z: http://is.muni.cz/th/172724/fi_m/?id=196171

PRIMATEST, 2013. What is RedwoodHQ? *RedwoodHQ | Open Source Automation Framework* [online] [vid. 9. duben 2016]. Dostupné z: http://redwoodhq.com/

RAMADOSS, Vybava, 2015. Container Apps now available in the Azure Marketplace. *Microsoft Azure* [online] [vid. 13. duben 2016]. Dostupné z: https://azure.microsoft.com/en-us/blog/container-apps-now-available-in-the-azure-marketplace/

REJNKOVÁ, Petra, 2016a. *Seznam testovacích nápadů* [online]. 2016. [vid. 15. duben 2016]. Dostupné z: http://mmsp.czweb.org/MMSP/workproducts/seznam_testovacich_napadu_E79F683A.html

REJNKOVÁ, Petra, 2016b. *Use Case* [online]. 2016. [vid. 15. duben 2016]. Dostupné z: http://mmsp.czweb.org/openup/workproducts/use_case_22BE66E2.html

ROJO, Robert, 2015. *Testování softwaru v agilních projektech* [online]. Praha. Bakalářská práce. Vysoká škola ekonomická v Praze. Dostupné z: https://isis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=50208

SAHI PRO, 2015. Quick Tutorial. *Sahi Pro* [online] [vid. 9. duben 2016]. Dostupné z: http://sahipro.com/docs/using-sahi/quick-tutorial.html

SAHI PRO, 2016. Open Source Automation Testing Tool | Free Tool. *Sahi Pro* [online] [vid. 9. duben 2016]. Dostupné z: http://sahipro.com/sahi-open-source/

SELENIUM, 2015a. Downloads. *Selenium* [online] [vid. 3. duben 2016]. Dostupné z: http://www.seleniumhq.org/download/#client-drivers

SELENIUM, 2015b. Selenium History. *Selenium* [online] [vid. 3. leden 2016]. Dostupné z: http://docs.seleniumhq.org/about/history.jsp

SELENIUM, 2015c. Selenium IDE Plugins. *Selenium* [online] [vid. 3. duben 2016]. Dostupné z: http://docs.seleniumhq.org/projects/ide/

SELENIUM, 2015d. Selenium WebDriver. *Selenium Documentation* [online] [vid. 3. duben 2016]. Dostupné z: http://www.seleniumhq.org/docs/03_webdriver.jsp

SELENIUM, 2016. By. *GitHub* [online] [vid. 21. březen 2016]. Dostupné z: https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/By.html

SKLENÁŘ, Pavel, 2012. *Testování webových aplikací za pomoci detekce změn jejího vzhledu* [online]. Praha [vid. 17. listopad 2015]. Diplomová práce. Vysoká škola ekonomická v Praze. Dostupné z: https://isis.vse.cz/zp/index.pl?podrobnosti_zp=36653

SLÁNSKÝ, David, Jan POUR a Ota NOVOTNÝ, 2004. *Business Intelligence: Jak využít bohatství ve vašich datech*. B.m.: Grada Publishing a.s. ISBN 978-80-247-6685-0.

SOKOL, Martin, 2013. *Automatické testování webových aplikací* [online]. Brno [vid. 28. říjen 2015]. Diplomová práce. Masarykova univerzita. Dostupné z: http://is.muni.cz/th/207509/fi_m/diplomova_praca.pdf

STATCOUNTER, 2016. Top 5 Desktop Browsers on Apr 2016. *StatCounter Global Stats* [online] [vid. 16. duben 2016]. Dostupné z: http://gs.statcounter.com/#desktop-browser-ww-monthly-201604-201604-bar

STEWART, Simon, 2010. Selenium WebDriver. *The Architecture of Open Source Applications* [online] [vid. 3. duben 2016]. Dostupné z: http://www.aosabook.org/en/selenium.html

TESTINGBOT, 2016. Cross Browser Selenium Testing on browsers and mobile with TestingBot. *TestingBot* [online] [vid. 10. duben 2016]. Dostupné z: https://testingbot.com/

TŘÍSKOVÁ, Lucie, 2015. *Testování webových aplikací s využitím nástroje Selenium Webdriver* [online]. Praha. Diplomová práce. Vysoká škola ekonomická v Praze. Dostupné z: https://isis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=45189

TURBO, 2016. What is Turbo? *Docs* [online] [vid. 13. duben 2016]. Dostupné z: https://turbo.net/docs#how-does-it-work

VENEZIA, Paul, 2015. Docker výrazně zjednodušuje pojetí virtualizace. *Computerworld* [online] [vid. 13. duben 2016]. Dostupné z: http://computerworld.cz/internet-a-komunikace/docker-vyrazne-zjednodusuje-pojeti-virtualizace-51785

VOTLUČKA, Vojtěch, 2015. *Automatizace testování softwaru nástrojem Selenium* [online]. Praha [vid. 17. listopad 2015]. Diplomová práce. Vysoká škola ekonomická v Praze. Dostupné z: https://isis.vse.cz/zp/index.pl?podrobnosti_zp=52027

WEBZOOM.CZ, 2016. Nová éra trendů v oblasti eCommerce přichází do České republiky. *Webzoom.cz* [online]. [vid. 9. duben 2016]. Dostupné z: http://www.webzoom.cz/nova-era-trendu-v-oblasti-ecommerce-prichazi-do-ceske-republiky/

WILLIAMS, Laurie, 2006. *Testing Overview and Black-Box Testing Techniques* [online]. 2006. [vid. 29. březen 2016]. Dostupné z: http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf

Seznamu obrázků

Seznam výpisů kódu

Seznam tabulek

Příloha A: Uživatelská příručka k aplikačnímu rámci Geb

Obsahem této přílohy uživatelská příručka k aplikačnímu rámci Geb. Příručka cíli především na uživatele, kteří začínají s tímto aplikačním rámcem pracovat. Postupně jsou představeny jednotlivé principy práce a funkce, jež Geb nabízí. Stejně jako samotné Selenium, je i Geb distribuován jako archiv ve formátu *.jar*. Díky tomu je možné ho snadno přiložit jako knihovnu k projektu, ať už přímo nebo s využitím libovolného nástroje pro automatizaci sestavování aplikací, jako jsou například Gradle nebo Maven. Aktuální verze kódu je vždy dostupná na stránce projektu na portálu GitHub.[16] Jakmile je knihovna připojena k projektu, je možné Geb naimportovat a začít s ním pracovat.

```
1.  dependencies {
2.      testCompile "org.testng:testng:6.9.10"
3.      testCompile "org.codehaus.groovy:groovy-all:$groovyVersion"
4.      testCompile "org.gebish:geb-testng:$gebVersion"
5.      testCompile "org.seleniumhq.selenium:selenium-java:$seleniumVersion"
6.
7.      // Drivers
8.      testCompile "org.seleniumhq.selenium:selenium-chrome-driver:$seleniumVersion"
9.      testCompile "org.seleniumhq.selenium:selenium-firefox-driver:$seleniumVersion"
10.     testCompile("com.codeborne:phantomjsdriver:1.2.1") {
11.         // phantomjs driver pulls in a different selenium version
12.         transitive = false
13.     }
14. }
```

Výpis kódu 6: Připojení nástroje Geb k projektu prostřednictvím Gradle

Základní způsob, jak psát testovací skripty pomocí Gebu, je využití objektu *Browser*. Ten využívá instanci třídy WebDriver, která zprostředkovává samotné ovládání prohlížeče. Třída Browser obsahuje statickou metodu *drive()*, do které se zapisují samotné příkazy pro ovládání prohlížeče. Tento způsob je vhodný pro jednoduché skripty, případně pro první seznámení s Gebem.

```
1.  Browser.drive {
2.      go "bi/datamodel"
3.      assert $("h2").text() == "Data sets"
4.  }
```

Výpis kódu 7: Základní příklad metody drive()

Druhou možností je vytvoření instance prohlížeče. Tato možnost se využívá i v případě integrace s jedním z podporovaných testovacích frameworků (TestNG, Spock, JUnit

[16] https://github.com/geb/geb

a Cucumber-JVM). Všechny tyto nástroje fungují stejným způsobem. Poskytují podtřídu, která vytváří a nastavuje instanci *Browser*, se kterou následně pracují všechny skripty. Tato možnost není kompatibilní s metodou *drive()*, vždy je nutné používat pouze jednu z nich. V dalších částech práce budu pracovat právě s touto možností.

A.1:Výběr elementů prostřednictvím aplikačního rámce Geb

Základním stavebním kamenem testování pomocí Gebu je možnost interakce s webovou stránkou. K tomu se využívá aplikační rozhraní *Navigator*. *Navigator* je nástroj pro určování DOM elementů, jejich filtraci a interakci s nimi. DOM - *Document Object Model* neboli objektový model dokumentu, je rozhraní, které umožňuje přistupovat k jednotlivým objektům stránky a pracovat s nimi. Mechanismus interakce obsahu s rozhraním Navigator je velmi podobný tomu, který využívá populární javascriptová knihovna jQuery. Tato vlastnost velmi usnadňuje jak přípravu testovacích skriptů.

Pro přístup k obsahu stránky se využívá funkce *$()*. Díky této funkci je možné pracovat se samotným obsahem stránky. Tato funkce vrací *Navigator* objekt, který představuje jeden nebo více prvků stránky. Ve funkce *$()* je možné pracovat s celou řadou různých selektorů. Lze použít CSS, selektory poskytované nástrojem WebDriver, indexy nebo porovnávání atributů či textů.

CSS selektory

Pro výběr elementů je možné použít jakýkoliv CSS selektor, který je podporován nástrojem WebDriver. Je možné využít i skládání několika selektorů dohromady.

```
1.  $("#setupStagesNav li span")
2.  $("h3.activeLink")
3.  $("tr.uroles a:first-child")
```

Výpis kódu 8: Příklady CSS selektorů

Selektory nástroje Selenium WebDriver

WebDriver poskytuje celou řadu selektorů ve třídě *selenium.By*:

- ByClassName,
- By.ByCssSelector,
- By.ById,
- By.ByLinkText,

- By.ByName,
- By.ByPartialLinkText,
- By.ByTagName,
- By.ByXPath (Selenium 2016).

Ve většině případů je možné místo nich využít standardní CSS selektory. Sami autoři doporučují používat *By* selektory pouze v případech, kdy je to nezbytné, například v případě některých XPath dotazů.

Indexy a rozsahy

Pokud se na stránce vyskytuje více stejným elementů, je možné přistoupit k požadovanému prvku prostřednictvím indexů. Případně je možné zadat rozsah indexů a pracovat tak s několika elementy najednou. Vzhledem k tomu, že Groovy vychází z jazyku Java, jsou prvky v poli číslovány od nuly. Jako příklad je možné uvést například standardní menu:

```
1.  <ul class="menu">
2.  <li>Folders</li>
3.  <li>Dashboards</li>
4.  <li>Reports</li>
5.  <li>Data sets</li>
6.  <li>Users</li></ul>
7.  </ul>
```

Výpis kódu 9: HTML kód jednoduchého menu

```
1.  $("li", 0) //přístup k první položce menu - Folders
2.  $("li", 2) //přístup ke třetí položce menu - Reports
3.  $("li", 1..2) //přístup k druhé a třetí položce menu - [[Dashboards, Reports]]
```

Výpis kódu 10: Příklady použití indexů a rozsahů

Atributy

Elementy je možné vybírat také na základě hodnoty libovolného atributu, který je k elementu přiřazen. Atributem může být i třída nebo ID elementu. V tom případě je ve většině případů vhodnější využít CSS selektory.

Výběr podle textu

V některých případech není možné použít k určení elementu žádný z výše zmíněných způsobů a je nutné vyhledat element na základě textu, který obsahuje. Ve frameworku Geb je možné pro vyhledávání použít celý text nebo pouze jeho část. Všechny funkce pro vyhledávání textu jsou standardně citlivé na velikost písmen. Ke každé ale existuje alternativa, která velikost písmen v potaz nebere.

Funkce (citlivá na velikost písmen	Alternativní verze (nebere v potaz velikost písmen)	Popis
text	-	Element musí obsahovat daný text. Příklad: `$("p", text: "Domain")`
startsWith	iStartsWith	Element musí obsahovat text, který začíná danou hodnotou. Příklad: `$("p", text: startsWith("Report"))`
contains	iContains	Text elementu musí obsahovat danou hodnotu.
endsWith	iEndsWith	Element musí obsahovat text, který končí danou hodnotou.
containsWord	iContainsWord	Text elementu musí obsahovat danou hodnotu oddělenou mezerami nebo začátkem, respektive koncem textu.
notStartsWith	iNotStartsWith	Element nesmí obsahovat text, který začíná danou hodnotou.
notContains	iNotContains	Text elementu nesmí obsahovat danou hodnotu.
notEndsWith	iNotEndsWith	Element nesmí obsahovat text, který končí danou hodnotou.
notContainsWord	iNotContainsWord	Text elementu nesmí obsahovat danou hodnotu oddělenou mezerami nebo začátkem, respektive koncem textu.

Tabulka 24: Seznam funkcí pro práci s textem (zdroj: autor)

A.2:Testování v nástroji Geb

Základem testovacích skriptů v Gebu je příkaz *assert*. Pomocí tohoto příkazu je možné porovnávat očekávaný a skutečný stav prvku, skupiny prvků nebo celé stránky. Porovnávat je možné jakýkoliv prvek, který lze vybrat pomocí selektorů zmíněných v předchozí kapitole, nebo určité prvky stránky, jako je URL adresa nebo titulek stránky (tag *<title>*). Nejčastější a nejjednodušší použití příkazu *assert* je kontrola, zda vybraný element obsahuje daný text.

Pro výběr elementu se používá funkce *$*, jejíž textový obsah se pomocí dvou znamének „rovná se“ porovnává s textem v uvozovkách. Druhou možností je porovnávání očekávaného a skutečného počtu výskytů daného elementu. V tom případě je funkce *$* doplněna o metodu *size()*.

```
1.  assert $(".modal-header h2").text() == "Import Data"
2.  //porovnání textu nadpisu h2
3.
4.  assert $("tr.uroles td", class: "urole-inactive").size() == 6
5.  //kontrola počtu buněk tabulky s třídou urole-inactive
6.
7.  assert $("select", id: "fileContentTypeSelect").size() == 0
8.  //kontrola neexistence výběrového formuláře s id fileContentTypeSelect
9.
10. assert $("#setupStagesNav li span")*.text() == ["License", "Domain", "Administrator
    account", "Installation complete"]
11. //porovnání jednotlivých položek menu za využití pole
```

Výpis kódu 11: Příklady využití příkazu assert

A.3:Objektový přístup

Jednou z hlavních výhod použití aplikačního rámce Geb je možnost pracovat s jednotlivými stránkami webu jako s objekty. Každá stránka je definována jako samostatný objekt, který má určité vlastnosti a funkce. Tyto vlastnosti a funkce je následně možné využívat napříč všemi testy. Hlavním přínosem toho přístupu je značné usnadnění údržby kódu. Případně změny je nutné aplikovat pouze v rámci definice stránky a nikoliv ve všech dotčených testech. Díky tomu je možné značně snížit časové a finanční náklady na údržbu testů.

Každý objekt stránky musí dědit ze supertřídy *Page*. V této třídě jsou definovány všechny základní metody pro práci se stránkami.[17] Součástí frameworku Geb je také doménově specifický jazyk (DSL) pro definování obsahu stránky. Prostřednictvím statické vlastnosti *content* je možné jednoduše popsat obsah stránky. Zpravidla není nutné definovat veškerý obsah stránky, pouze ty části, které jsou významné pro práci se stránkou, respektive pro samotné provedení testů. Může se tedy jednat o různá menu, odkazy, formulářové prvky apod. Každý z těchto prvků má v rámci definice objektu stránky své vlastní jméno, na které je následně možné se odkazovat z testů.

[17] http://www.gebish.org/manual/current/api/geb/Page.html

```
class GeneralPage extends Page {

    static content = {
            navbar { module TopMenuModule }
            goToGallery { $('#profileNav a', title: 'Media Gallery' )}
            logoutLink { $('#templatesLoginout a', text: 'Log out' )}
                    }

        }
```

Výpis kódu 12: Příklad jednoduché definice stránky a jejího obsahu

Ke stránkám lze definovat i celé metody včetně parametrů. Kromě samotného obsahu lze tedy definovat i interakce s obsahem. Jako příklad lze například uvést metodu pro vyplnění přihlašovacích údajů a kliknutí na přihlášení uživatele. Uživatelské jméno a heslo jsou mezi metodou a testovacím skriptem předávány prostřednictvím parametrů. V samotném testu tedy stačí uvést jméno metody a přihlašovací údaje.

```
//definice testu
class LoginTest extends WebTestBase {

    @Test
    void testLogin() {
        to LoginPage

        login("admin","Support01")

        }
//definice stránky
import geb.Page

class LoginPage extends GeneralPage {
    static url = "/en/login"

    static at = { title == "Login" }
    void login(String name, String password) {

           $("form").login = name
           $("form").password = password
           $('input[type=submit]').click()
        }
}
```

Výpis kódu 13: Příklad metody v objektu přihlašovací stránky a její otestování

Stejně jako u klasických objektově orientovaných jazyků je možné využít dědičnost. Objekty (stránky) lze organizovat do stromové struktury, kdy stránky mohou přebírat definice obsahu a metody z nadtřídy. Tato funkčnost umožňuje definovat generický obsah pouze na jednom místě a následně využít dědičnosti k distribuci do vybraných stránek. Pokud má podtřída definovaný vlastní obsah, je sloučen s definicemi obsahu z nadtřídy. V případě, že obě třídy obsahují obsah se stejným názvem, má vždy přednost definice z podtřídy. Dědičnost se využívá u každého objektu stránky, neboť ten dědí základní metody z výchozí třídy *Page*.

```
1.  import geb.Page
2.
3.  class GeneralPage extends Page {
4.
5.     static content = {
6.             navbar { module TopMenuModule }
7.             //profilenav {module ProfileNavModule}
8.             goToGallery { $('#profileNav a', title: 'Media Gallery' )}
9.             logoutLink { $('#templatesLoginout a', text: 'Log out' )}
10.
11.             }
12.      def popup() {
13.         waitFor { $(".t_popupLayoutBgHelper") }
14.     }
15.     void closePopup() {
16.         popup().find("#t_popupLayoutCloser").click()
17.     }
18.
19.     def popupQuestionFrame() {
20.         waitFor { $("#popupQuestionFrame") }
21.     }
22.    }
```

Výpis kódu 14: Příklad definice nadtřídy a dědičnosti z třídy Page

A.4:Práce s URL adresami

Jak je patrné z předchozích příkladů, Geb podporuje práci s relativními URL adresami. Není tedy nutné vždy zadávat celou adresu, ale pouze její proměnlivou část. Ta je následně za běhu doplněna o tzv. *base URL* neboli výchozí adresu. Ta je uvedena v konfiguračním skriptu. Výhodou tohoto řešení je snadná přenositelnost mezi prostředními. Pouhou změnou výchozí adresy je možné přepínat například mezi URL testovacího a produkčního prostředí.

Při zadávání výchozí adresy a následně při používání relativních adres je důležité správně nastavit lomítka. Nevhodnou kombinací, například vynecháním lomítka na konci výchozí adresy a zároveň neuvedením lomítka na začátku relativní adresy, může dojít ke vzniku nesmyslných adres. Obecně se doporučuje zapsat výchozí adresu s lomítkem, čímž je možné lomítko vynechat u relativních adres. Seznam všech kombinací lze nalézt v tabulce níže.

Výchozí adresa	Relativní adresa	Výsledek
http://myapp.com/	abc	http://myapp.com/abc
http://myapp.com	abc	http://myapp.comabc
http://myapp.com	/abc	http://myapp.com/abc
http://myapp.com/abc/	def	http://myapp.com/abc/def
http://myapp.com/abc	def	http://myapp.com/def
http://myapp.com/abc/	/def	http://myapp.com/def

Výchozí adresa	Relativní adresa	Výsledek
http://myapp.com/abc/def/	jkl	http://myapp.com/abc/def/jkl
http://myapp.com/abc/def	jkl	http://myapp.com/abc/jkl
http://myapp.com/abc/def	/jkl	http://myapp.com/jkl

Tabulka 25: Seznam kombinací výchozí a relativní adresy (zdroj: Daley et al. 2015)

Relativní stránky je možné používat buď přímo v kódu testu, nebo v definici objektu stránky. Podobně jako v případě obsahu stránky se jedná o statickou vlastnost, v tomto případě nazvanou *url*. Pokud má stránka definovanou *url*, je následně možné využívat příkaz *to*, který slouží k přechodu na danou stránku. Příklad využití navigace prostřednictvím příkazu *to* je možné nalézt v předchozím výpisu kódu (Výpis kódu 13Výpis kódu 13: Příklad metody v objektu přihlašovací stránky a její otestování).

A.5:Interakce s elementy

Geb nabízí několik způsobů, jak pracovat s aktivními prvky stránky. Základní interakcí je běžné kliknutí, které se provádí pomocí metody *click()*. Tato metoda je ekvivalentem kliknutí levým tlačítkem myši. Kliknutí není možné aplikovat na multi-element, neboli na *Navigator* obsahující pole elementů. Pokud se tak stane, je vrácena chyba *SingleElementNavigatorOnlyMethodException.* Metodu *click()* lze používat také s parametrem *Page*. V tom případě po kliknutí proběhne verifikace, zda se prohlížeč nachází na dané stránce.

Kromě základního kliknutí nabízí Geb také možnost používat složitější interakce, respektive kombinace několika interakcí zároveň. Jedná se o metodu interact(), která pracuje s třídou Actions. Ta je poskytována prostřednictvím API nástroje WebDriver. Do těla metody lze vkládat neomezené množství příkazů, která se následně postupně provedou. Díky tomu je možné simulovat funkce jako *drag and drop* nebo výběr několika prvků pomocí klávesy Control nebo Shift. K dispozici je celá řada různých akcí. Jejich seznam lze nalézt v tabulce níže.

Akce	Popis
click()	Kliknutí na element
clickAndHold()	Kliknutí na element s podrženým stisknutím
contextClick()	Kontextové kliknuté na element, např. kliknutí pravým tlačítkem myši.
doubleClick()	Dvojitý klik na element
dragAndDrop(WebElement zdroj, WebElement cíl)	Podržení stisknutí na zdrojovém elementu a jeho přesun na pozici cílového elementu.
dragAndDropBy(WebElement zdroj, int xOffset, int yOffset)	podržení stisknutí na zdrojovém elementu a jeho přesun o danou horizontální a vertikální vzdálenost.
keyDown()	Stisknutí (a podržení) klávesy, například Control nebo Shift.
keyUp()	Uvolnění stisku klávesy
moveByOffset(int xOffset, int yOffset)	Přesun kurzoru o danou horizontální a vertikální vzdálenost.
moveToElement(WebElement toElement)	Přesun kurzoru na střed elementu
release()	Uvolnění stisku tlačítka myši

Tabulka 26: Seznam dostupných interakcí s elementy (zdroj: Daley et al. 2015)

A.6:Práce s formuláři

Samostatnou kapitolou interakce s webovou stránkou jsou formuláře. Zejména moderní webové aplikace jsou doslova přeplněné různými formuláři a jejich kontrola jejich funkčnosti je nedílnou součástí testování. Geb proto nabízí velmi jednoduchý způsob, jak pracovat s jednotlivými formulářovými prvky a následně kontrolovat jejich stav. K prvku, jenž má definované jméno, je možné přistupovat prostřednictvím identifikace formuláře a zadáním jména prvku.

```
1.  popup()
2.  $("#popupZone form").name = "u1"
3.  $("#popupZone form").surname = "username"
4.  $("#popupZone form").email = "ondrej.spalek+geb@email.com"
5.     $('#popupZone input[type=submit]').click()
```

Výpis kódu 15: Příklad vyplnění a odeslání formuláře

Select

Hodnoty z výběrového formuláře je možné vybírat buď uvedením hodnoty atributu *value* nebo uvedením samotného obsahu jedné z možností.

```
1.  // HTML formulář
2.  <form>
3.     <select name="language">
4.        <option value="1">English</option>
5.        <option value="2">Czech</option>
6.        <option value="3">Chinese</option>
7.     </select>
8.  </form>
9.
10. //práce s formulářem v Gebu
11. $("form").language = "1"
12. $("form").language = "Czech"
```

Výpis kódu 16: Příklad práce s výběrovým formulářem v Gebu

Multiple select

Práce s touto komponentou je stejná jako v případě *selectu*, jediným rozdílem je možnost vybrat několik položek najednou. Položky lze zadat prostřednictvím pole a opět je možné použít jak hodnoty atributu *value*, tak i obsah možností.

```
1.  $("form").language = ["1","3"]
2.  $("form").language = ["Czech","English","Chinese"]
```

Výpis kódu 17: Příklad práce s multi selectem v Gebu

Checkbox

U komponenty *checkbox*, neboli zaškrtávacího políčka, je potřeba dávat pozor na to, zda je stejné jméno (atribut *name*) použito pro jeden nebo více políček. V prvním případě je možné vybírat položku prostřednictvím jména, v druhém případě je nutné použít hodnotu atributu *value*. Pro získání hodnoty políčka je možné použít metodu *value()*. Ta v případě zaškrtnutého políčka vrátí jeho hodnotu. V opačném případě vrátí *false*.

```
1.  //HTML formulář
2.  <form>
3.      <input type="checkbox" name="country" value="China" checked/>
4.      <input type="checkbox" name="country" value="Germany" />
5.  </form>
6.
7.  //Geb – zaškrtnutí políčka China
8.  $("form").country= "China"
9.
10. //Geb – porovnání očekávané a skutečné hodnoty pomocí metody value()
11. assert $("input", name: "country").value() == "China"
```

Výpis kódu 18: Příklad práce se zaškrtávacím políčkem v nástroji Geb

Radio

Přepínače zpravidla slouží k výběru právě jedné položky ze skupiny. Všechny položky tedy musejí mít stejné jméno. Z toho důvodu je možné vybírat položky pouze podle hodnoty atributu *value* nebo podle popisku. Popisek položky se zapisuje pomocí tagu <*label*>.

```
1.  //HTML
2.  <form>
3.      <label for="country">China</label>
4.      <input type="radio" name="country" value="country-china">
5.      <label for="country">Germany</label>
6.      <input type="radio" name="country" value="country-germany">
7.  </form>
8.
9.  //Geb – Výběr přepípače pomocí atributu value
10. $("form").country= "country-germany"
11.
12. //Geb – Výběr přepínače pomocí popisku
13. $("form").country= "China"
```

Výpis kódu 19: Příklad práce s přepínačem v Gebu

Textové pole

V případě textového pole lze použít pro asociaci jeho jméno. Do pole je možné vložit celý text nebo je možné ho upravovat. Je možné odstranit část textu, přidat další řetězec nebo vložit nový řádek. Mazání znaků a odřádkování se provádí simulací stisků kláves.

```
1.  $("#popupZone form").text = "att1;ind1z"
2.  $("#popupZone form").text() << Keys.BACK_SPACE
3.  assert ("#popupZone form").text  == "att1;ind1"
4.  $("#popupZone form").text() << Keys.ENTER
5.  $("#popupZone form").text = "aaa;1111"
```

Výpis kódu 20: Příklad vložení dvou řádků textu do textové oblasti v Gebu

Nahrávání souborů

V současné verzi nástroje WebDriver není možné vyvolat klasické dialogové okno operačního systému pro výběr souboru. Nahrávání souborů je tedy nutné simulovat přímým výběrem souboru.

```
1.  //HTML
2.  <input type="file" name="csv"/>
3.
4.
5.  //GEB
6.  $("form").csv = dataSetFile.absolutePath
```

Výpis kódu 21: Příklad výběru souboru v Gebu

Proměnná *dataSetFile* obsahuje instanci třídy *File*, která směřuje na soubor, jenž má být nahrán.

A.7:Moduly

Moduly jsou další z funkcí, která velmi usnadňuje přípravu a údržbu testů. Jedná se o znovupoužitelné definice obsahu, které lze použít na neomezeném počtu stránek. Díky modulům není nutné definovat stejný obsah na několika stránkách. Zpravidla se používají pro modelování prvků rozhraní, které se vyskytují na větším počtu míst. Může se jednat například o navigaci, různá menu nebo generické formuláře. Moduly je možné skládat do sebe a vytvářet složité modulární konstrukce.

Jedním ze způsobů, jak používat moduly, je definovat pomocí nich pouze strukturu prvku a samotný obsah ověřovat až na úrovni testů. Tento přístup je vhodný v případě, kdy jeden prvek může mít v různých případech různý obsah. Příkladem může být například hlavní navigace, kdy každý typ uživatele má přístup k jiným částem aplikace a vidí tak v navigaci jiné položky. Struktura kódu je ale vždy stejná, jedná se o seznam odkazů. Pokud je tedy v modulu navigace definovaná takto genericky, je možné modul použít pro všechny uživatele.

```
1.  //Definice modulu
2.  class TopMenuModule extends Module {
3.
4.    static content = {
5.       links { $('#mainMenu li a') }
6.    }
7.  }
8.
9.  //Použití modulu na stránce
10. class GeneralPage extends Page {
11.
12.    static content = {
13.             navbar { module TopMenuModule }
14.              }
15. }
16.
17. //Test menu administrátora
18.  assert $(navbar.links)*.text() == [ "Folders", "Dashboards","Reports","Data sets",
    "Administration" ]
19.
20.
21. //Test menu normální uživatele
22.  assert $(navbar.links)*.text() == [ "Folders", "Dashboards","Reports","Data sets",
    "Users" ]
```

Výpis kódu 22: Příklad použití modulu v aplikačním rámci Geb

Moduly také mohou být parametrizované. Parametr je nutné modulu předat v jeho konstruktoru. Jde o další způsob, jak udělat modul více univerzální.

```
1.  //Definice modulu
2.  class ParameterizedModule extends Module {
3.      String formId
4.      static content = {
5.          button {
6.              $("form", id: formId).find("input", type: "button")
7.          }
8.      }
9.  }
10. //Použití modulu na stránce
11. class ParameterizedModulePage extends Page {
12.     static content = {
13.         form { id -> module(new ParameterizedModule(formId: id)) }
14.     }
15. }
16. //Test
17. Browser.drive {
18.     to ParameterizedModulePage
19.     form("personal-data").button.click()
20. }
```

Výpis kódu 23: Příklad použití parametrizovaného modulu (zdroj: Daley et al. 2015)

Příloha B: Příklad souboru build.gradle

```
1.  import org.apache.tools.ant.taskdefs.condition.Os
2.
3.  def host = System.getProperty('host', 'localhost')
4.  def port = Integer.valueOf(System.getProperty('port', '8081'))
5.  def belladatiWar = zipTree(System.getProperty('belladatiWar', "$buildDir/resources/t
    est/belladati.war"))
6.
7.  ext {
8.      // definice ovladačů
9.      drivers = ["firefox", "chrome", "phantomJs"]
10.
11.     ext {
12.         groovyVersion = '2.4.5'
13.         gebVersion = '0.12.2'
14.         seleniumVersion = '2.52.0'
15.         chromeDriverVersion = '2.19'
16.         phantomJsVersion = '1.9.7'
17.     }
18. }
19.
20. apply plugin: "groovy"
21. apply plugin: "jetty"
22. apply from: "gradle/idea.gradle"
23. apply from: "gradle/osSpecificDownloads.gradle"
24.
25. repositories {
26.     mavenCentral()
27. }
28.
29. dependencies {
30.     testCompile "org.testng:testng:6.9.10"
31.     testCompile "org.codehaus.groovy:groovy-all:$groovyVersion"
32.     testCompile "org.gebish:geb-testng:$gebVersion"
33.     testCompile "org.seleniumhq.selenium:selenium-java:$seleniumVersion"
34.
35.     // ovladače
36.     testCompile "org.seleniumhq.selenium:selenium-chrome-driver:$seleniumVersion"
37.     testCompile "org.seleniumhq.selenium:selenium-firefox-driver:$seleniumVersion"
38.     testCompile("com.codeborne:phantomjsdriver:1.2.1") {
39.
40.         transitive = false
41.     }
42. }
43.
44. drivers.each { driver ->
45.     task "${driver}Test"(type: Test) {
46.         reports {
47.             html.destination = reporting.file("$name/tests")
48.             junitXml.destination = file("$buildDir/test-results/$name")
49.         }
50.         testLogging.showStandardStreams = true
51.         outputs.upToDateWhen { false }  // Always run tests
52.
53.         systemProperty "geb.build.reportsDir", reporting.file("$name/geb")
54.         systemProperty "geb.env", driver
55.         systemProperty "geb.build.baseUrl", "http://$host:$port/"
56.
57.         useTestNG{
58.             suites 'src/test/resources/testng.xml'
59.         }
60.
61.         doLast {
```

```
62.             stopBellaDati.execute()
63.         }
64.     }
65. }
66.
67. chromeTest {
68.     dependsOn unzipChromeDriver
69.
70.     def chromedriverFilename = Os.isFamily(Os.FAMILY_WINDOWS) ? "chromedriver.exe" :
     "chromedriver"
71.     systemProperty "webdriver.chrome.driver", new File(unzipChromeDriver.outputs.fil
    es.singleFile, chromedriverFilename).absolutePath
72. }
73.
74. phantomJsTest {
75.     dependsOn unzipPhantomJs
76.
77.     def phantomJsFilename = Os.isFamily(Os.FAMILY_WINDOWS) ? "phantomjs.exe" : "bin/
    phantomjs"
78.     systemProperty "phantomjs.binary.path", new File(unzipPhantomJs.outputs.files.si
    ngleFile, phantomJsFilename).absolutePath
79. }
80.
81. test {
82.     dependsOn drivers.collect { tasks["${it}Test"] }
83.     enabled = false
84. }
85.
86. task startBellaDati << {
87.     delete "$buildDir/belladati"
88.     copy {
89.         from belladatiWar
90.         into "$buildDir/belladati"
91.     }
92.     delete "$buildDir/belladati/WEB-INF/classes/conf/application.properties"
93.     copy {
94.         from "$buildDir/resources/test/conf"
95.         include "application-webtest.properties"
96.         into "$buildDir/belladati/WEB-INF/classes/conf"
97.         rename 'application-webtest.properties', 'application.properties'
98.     }
99.     jettyRun.httpPort = port
100.     jettyRun.daemon = true
101.     jettyRun.stopKey = 'BdWebTests'
102.     jettyRun.stopPort = 9089
103.     jettyRun.webAppSourceDirectory = file("$buildDir/belladati")
104.     jettyRun.contextPath = '/'
105.     jettyRun.execute()
106. }
107.
108. task stopBellaDati << {
109.     jettyStop.stopKey = 'BdWebTests'
110.     jettyStop.stopPort = 9089
111.     jettyStop.execute()
112. }
113.
114. apply from: "gradle/ci.gradle"
```

Výpis kódu 24: Příloha B:Příklad souboru build.gradle