

Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informační systémy a technologie

Continuous integration pro open source webové aplikace na GitHubu

DIPLOMOVÁ PRÁCE

Student : Bc. Kamil Soule

Vedoucí : Ing. Tomáš Kliegr, Ph.D.

Oponent : Ing. Václav Zeman

2017

Prohlášení:

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

V Praze dne

.....

Kamil Soule

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Tomáš Kliegrovi, Ph.D. za poskytnutí zajímavého tématu diplomové práce, pravidelné konzultace a užitečné rady. Dále panu Ing. et Ing. Stanislav Vojířovi, Ph.D. za konzultace ohledně nasazení implementovaných testů pro systém EasyMiner.

Abstrakt

Tato diplomová práce nabízí ucelený pohled na problematiku testování grafického webového rozhraní v rámci průběžné integrace. Zaměřuje se na open source projekty vystavené na GitHubu, které pro své nasazení využívají Docker kontejnery. Součástí práce je přehled bezplatných cloudových serverů pro integraci open source projektů. Dále práce obsahuje přehled nástrojů pro vytváření automatizovaných testů, které jsou založeny na knihovně Selenium WebDriver. Zvláštní pozornost je věnována nástrojům podporujícím tvorbu testovacích scénářů dle agilní metodiky behaviour driven development. Další část práce popisuje návrh bezplatného technického řešení, které umožňuje začlenit testování grafického webového rozhraní do procesu průběžné integrace pro open source projekty. Navržené řešení využívá cloudový integrační server Travis CI a samostatný web s reporty a dalšími informacemi o proběhlých integracích. Pro definici automatizovaných testů využívá nástroj Robot framework a pro spouštění testů Docker obraz, který je zveřejněn na Docker Hubu. Použití navrženého řešení je demonstrováno pomocí jeho implementace a nasazení pro open source projekt EasyMiner. Tento projekt je vyvíjený na KIZI VŠE. Součástí implementace je vytvoření sady testovacích scénářů dle metodiky behaviour driven development.

Klíčová slova

Průběžná integrace, Docker, Travis CI, Selenium, Behaviour driven development, Robot framework, open source

Abstract

This diploma thesis provides a comprehensive view on graphical web user interface testing during continuous integration. It focuses on open source projects published on GitHub that use Docker containers for their deployment. First part of the work is an overview of free cloud servers for the integration of open source projects. In addition, the work includes an overview of automated testing tools that are based on the Selenium WebDriver library. Attention is paid to tools supporting the development of test scenarios according to the agile methodology called behaviour driven development. Another part of the thesis describes the design of technical solution that allows integration of graphical web interface testing into the process of continuous integration for open source projects. The proposed solution uses Travis CI as cloud integration server and a standalone website to store reports and other information on past integrations. Automated tests are defined using the Robot framework and executed using a Docker image, which is published on Docker Hub. The use of the proposed solution is demonstrated by its implementation and deployment for the open source project EasyMiner. This project is developed at the University of Economics Prague in DIKE department. The implementation includes the creation of a test suite with scenarios according to the behaviour driven development methodology.

Keywords

Continuous integration, Docker, Travis CI, Selenium, Behaviour driven development, Robot framework, open source

Obsah

Obsah	ii
Seznam obrázků	v
Seznam tabulek	vi
Seznam výpisů programů	vii
1 Úvod	
1.1 Cíle práce	8
1.2 Cílová skupina	9
1.3 Přínos práce	9
1.4 Struktura práce	10
2 Komentovaná řešení informačních zdrojů	11
2.1 Akademické práce	11
2.1 Internetové informační zdroje a tištěné odborné publikace	12
3 Specifikace pojmů	14
3.1 Continuous integration / Průběžná integrace	14
3.2 Continuous delivery / Průběžná dodávka	15
3.3 Continuous deployment / Průběžné nasazení	16
3.4 Docker	16
3.5 Behaviour driven development – vývoj řízený požadavky na chování	17
4 Přehled vybraných serverů pro podporu průběžné integrace	21
4.1 Travis CI	22
4.2 Semaphore CI	24
4.3 Circle CI	27
4.4 AppVeyor	29
4.5 Shrnutí přehledu	31
5 Přehled softwarových nástrojů pro automatizované testování webového rozhraní	33
5.1 Nástroje pro zachycení a přehrávání aktivity uživatele	34
5.2 Nástroje pro automatizované testování pomocí skriptů	34
5.2.1 Lineární skripty	35
5.2.2 Strukturované skripty	35
5.2.3 Daty řízené skripty	36
5.2.4 Skripty řízené klíčovými slovy	36
5.3 Přehled nástrojů pro automatizaci testů pomocí scénářů metodiky BDD	38

5.3.1	Cucumber	39
5.3.2	JBehave	40
5.3.3	Robot framework.....	41
5.3.4	Gauge	42
5.4	Shrnutí přehledu	43
6	Možnosti integrace Selenium testů uživatelského rozhraní do Travis CI	45
6.1	Volba platformy pro běh integrace.....	45
6.2	Volba typu Linuxového prostředí	46
6.3	Výběr nástroje pro definici a exekuci testů	46
6.4	Volba webového prohlížeče	47
6.5	Virtualizace displeje.....	47
6.6	Způsob instalace závislostí.....	48
6.7	Možnosti zasílání notifikací.....	49
6.8	Zajištění dostupnosti vygenerovaných artefaktů.....	49
6.9	Shrnutí možností.....	49
7	Řešení pro testování webového rozhraní během průběžné integrace pro open source projekty	51
7.1	Softwarové komponenty řešení a jejich komunikace	51
7.1.1	Docker kontejner zajišťující spouštění testů.....	54
7.1.2	Docker kontejnery pro spuštění testované aplikace	56
7.1.3	Nastavení serveru pro průběžnou integraci	57
7.2	Informování vývojářů o průběhu integrace	58
7.2.1	Návod pro publikování výsledků integrace na webové stránky	59
7.3	Shrnutí navrženého řešení.....	61
8	Implementace testů webového rozhraní během průběžné integrace pro projekt EasyMiner	62
8.1	Popis testovaného systému.....	62
8.1.1	Funkcionalita	62
8.1.2	Architektura.....	63
8.2	Cíle řešení a vhodnost použití navrženého řešení	64
8.3	Nastavení integračního serveru Travis CI.....	65
8.4	Spouštění aplikace EasyMiner a testů pomocí Docker compose.....	66
8.5	Informování vývojářů o průběhu integrace	67
8.6	Spouštění testů lokálně na počítači vývojáře.....	68
8.7	Testovací sada a pokrytí funkcionality testy	70
8.8	Konvence pro psaní testů	71
8.8.1	Hledání elementů na stránce	71
8.8.2	Konvence pro definici klíčových slov a proměnných	73
8.8.3	Konvence pro tvorbu testovacích scénářů	74

8.8.4 Vytváření testovacích dat	75
8.9 Shrnutí implementovaného řešení	76
9 Závěr	
9.1 Dosažení stanovených cílů	78
9.2 Přínos práce	80
Příloha A: Zdrojové kódy a výstupy testů aplikace EasyMiner	81
Použité informační zdroje	83

Seznam obrázků

Obrázek 1: Grafické znázornění procesu průběžné integrace (zdroj: Cois 2015)	15
Obrázek 2: Grafické znázornění rozdílů mezi virtuálními stroji a Docker kontejnery (zdroj: Docker 2017b).....	17
Obrázek 3: Hlavní komponenty navrženého řešení průběžné integrace a jejich vzájemné propojení (zdroj: autor)	52
Obrázek 4: Hlavní komponenty Docker obrazu pro exekuci Robot framework testů webového rozhraní (zdroj: autor).....	55
Obrázek 5: Architektura systému EasyMiner (zdroj: autor).....	63
Obrázek 6: Tabulka s reporty a informacemi o běhu integrací na vytvořeném webu (zdroj: autor).....	68

Seznam tabulek

Tabulka 1:	Plány předplatného služby Travis CI (zdroj: GitHub 2017, Travis CI 2017b).	23
Tabulka 2:	Vybrané plány předplatného služby Semaphore CI (zdroj: Rendered Text 2017b)	25
Tabulka 3:	Vybrané plány předplatného služby Circle CI pro integraci na Linuxu (zdroj: CircleCI 2017b)	28
Tabulka 4:	Vybrané plány předplatného služby Circle CI pro integraci na OS X (zdroj: CircleCI 2017b)	28
Tabulka 5:	Variety plánů předplatného služby AppVeyor (zdroj: Appveyor Systems 2017b)	30
Tabulka 6:	Shrnutí parametrů platných pro bezplatné verze a open source projekty vybraných CI řešení (zdroj: autor)	32
Tabulka 7:	Shrnutí vlastností vybraných nástrojů pro testování formou BDD scénářů (zdroj: autor)	39

Seznam výpisů programů

Výpis 1:	Ukázka yaml syntaxe – část konfigurace serveru Travis pro spouštění UI testů pro projekt EasyMiner (zdroj: autor)	22
Výpis 2:	Ukázka scénáře psaného syntaxí jazyka Gherkin, podporovaného BDD frameworkem Cucumber (zdroj: Wynne et al. 2017b)	36
Výpis 3:	Ukázka implementace klíčových slov v jazyce Ruby, podporovaného BDD frameworkem Cucumber (zdroj: Wynne et al. 2017b)	37
Výpis 4:	Ukázka scénáře napsaného pomocí BDD frameworku Gauge (zdroj: ThoughtWorks 2016b)	42
Výpis 5:	Spouštění testů s využitím Docker obrazu soulekamil/robotframework-ff (zdroj: autor)	55
Výpis 6:	Skript pro vytvoření zašifrovaného SSH klíče pro využití v Travis CI (zdroj: autor)	60
Výpis 7:	Nastavení EasyMiner frontendu v konfiguračním souboru docker-compose.yml (zdroj: autor)	67
Výpis 8:	Ukázka scénáře pro otestování úspěšného přihlášení uživatele (zdroj: autor)	70
Výpis 9:	Ukázka klíčového slova, které přímo využívá funkce knihovny Selenium (zdroj: autor)	70
Výpis 10:	Příklad nevhodně vytvořeného CSS selektoru (zdroj: autor)	72
Výpis 11:	Příklad korektně sestaveného CSS selektoru (zdroj: autor)	72

1 Úvod

Vzhledem ke komplexitě dnešních webových aplikací se na jejich vývoji zpravidla podílí tým několika vývojářů. Celý tým musí spolu spolupracovat a jednotlivé změny je nutné integrovat do jednoho celku. Proces integrace jednotlivých částí kódu je složitý a jeho chybné provedení může vést k chybám v aplikaci či dokonce nemožnosti pouhého sestavení projektu. Čím dříve se tyto chyby objeví, tím snazší je sjednat jejich nápravu. Problémy spojené se začleněním částí projektu do jednoho celku a kontrolou výsledného stavu projektu se snaží adresovat proces zvaný průběžná integrace (anglicky continuous integration). Jeho stěžejní myšlenkou je častá integrace práce jednotlivých členů týmu, nejméně jednou denně. Součástí každé integrace je automatické sestavení, testování a odeslání zpětné vazby o jejím průběhu vývojářskému týmu. Pro podporu tohoto procesu existuje řada nástrojů. V posledních letech se objevily na trhu cloudové služby pro podporu průběžné integrace, které jsou poskytovány pro open source projekty zdarma. Vzhledem k dostupnosti těchto bezplatných nástrojů je vhodné zabývat se možností zavedení průběžné integrace do open source projektů. Zavedení průběžné integrace totiž přináší řadu výhod.

Nedílnou součástí průběžné integrace je také automatizované testování softwaru. Pro webové projekty je dnes de facto standardem využívání nástrojů postavených nad knihovnou Selenium WebDriver. Mezi ně mimo jiné patří nástroje umožňující psaní testů formou scénářů dle agilní metodiky behaviour driven development. Ty na rozdíl od ostatních nabízí několik výhod. Mezi ně patří možnost tvorby testů méně technicky zdatnými uživateli, jasná a pevně daná struktura a možnost generování čitelné dokumentace chování aplikace přímo z vytvořených scénářů.

Právě vhodná kombinace nástroje pro průběžnou integraci a nástroje pro automatizované testování softwaru dokáže práci na projektu výrazně usnadnit a zefektivnit. Proto se tato práce zabývá problematikou testování grafického webového rozhraní v rámci průběžné integrace se zaměřením na open source projekty s využitím praktik z agilní metodiky BDD.

1.1 Cíle práce

Tato práce má dva dílčí cíle. Prvním je nalézt technické řešení, které by umožňovalo testovat webové uživatelské rozhraní open source projektů v rámci průběžné integrace. Hledané řešení musí umožňovat průběžnou integraci projektů uložených na serveru GitHub a podporovat testování aplikací, které jsou nasazovány s využitím Docker kontejnerů. Řešení by mělo využívat pouze bezplatných nástrojů.

Druhým dílčím cílem je demonstrovat nalezené řešení pomocí vývoje sady testů pro jednoho reprezentanta skupiny výše definovaných aplikací. Vybraným reprezentantem je webové rozhraní projektu EasyMiner. Tato open source aplikace pro dolování asociačních pravidel je vyvíjena týmem v rámci KIZI VŠE.

1.2 Cílová skupina

Cílovou skupinou této práce jsou zájemci o zavedení automatizovaných testů webového rozhraní v rámci průběžné integrace do open source projektů. Řešení prezentované v této práci je dostupné zcela zdarma pro open source. Proto je určené nejen pro velké projekty, ale také pro menší projekty, které nemohou nebo nechtějí investovat finanční prostředky do provozu řešení průběžné integrace. Součástí práce není porovnání manuálního a automatizovaného testování. Proto je práce vhodná pro ty čtenáře, kteří jsou již rozhodnuti pro automatizaci testů a jsou ve fázi hledání technického řešení tohoto problému.

1.3 Přínos práce

Přínosem této práce je návrh bezplatného technického řešení, které umožňuje začlenit testování grafického webového rozhraní do průběžné integrace pro open source projekty. Toto řešení je následně demonstrováno pomocí implementace a nasazení pro open source projekt EasyMiner. Díky tomu lze práci využít jako návod pro široké spektrum open source projektů vystavených na GitHubu, podle kterého lze zajistit začlenění testování webového rozhraní do procesu průběžné integrace. Vzhledem k využití praktik definovaných agilní metodikou behaviour driven development, je dalším přínosem této práce demonstrace využití BDD v praxi.

Součástí navrženého řešení je vytvoření Docker obrazu, který je nakonfigurován pro spouštění Robot framework testů a připraven k okamžitému použití. Tento obraz je zveřejněný na centrálním repozitáři obrazů Docker Hub. Tento obraz lze použít nejen jako součást navrženého řešení, ale také pro lokální spuštění Robot framework scénářů či exekuci testů v jednom z alternativních serverů pro průběžnou integraci.

Práce zároveň obsahuje přehled nástrojů pro vytváření automatizovaných testů a cloudových serverů pro průběžnou integraci. Tím poskytuje čtenáři informace o možných alternativách k navrženému řešení.

1.4 Struktura práce

Teoretická část práce obsahuje přehled bezplatných cloudových serverů pro integraci open source projektů. Dále je její součástí přehled nástrojů pro vytváření automatizovaných testů, které jsou založeny na knihovně Selenium WebDriver. Zvláštní pozornost je věnována nástrojům podporujícím tvorbu testovacích scénářů dle agilní metodiky behaviour driven development (BDD). Součástí je také kapitola zabývající se možnostmi začlenění Selenium testů do nástroje pro průběžnou integraci Travis CI.

Praktická část práce popisuje návrh bezplatného technického řešení, které umožňuje začlenit testování grafického webového rozhraní do průběžné integrace pro open source projekty. Další kapitola popisuje implementaci a nasazení navrženého řešení pro open source projekt EasyMiner, vyvíjený na KIZI VŠE.

2 Komentovaná řešerše informačních zdrojů

Automatizace testování uživatelského rozhraní webových aplikací je zajímavé a aktuální téma. Proto již existuje několik akademických prací, tištěných publikací a článků na webu, které se danou problematikou zabývají. Zde je uveden krátký popis několika informačních zdrojů, které jsou tematicky souvisejí s touto prací.

2.1 Akademické práce

Ačkoliv se tématem automatizace testování zabývalo více studentů, tato práce přináší nový náhled do problematiky svým soustředěním na cloudové nástroje pro průběžnou integraci nabízené pro open source projekty zdarma. Také kombinace CI a testování pomocí scénářů převzatých z metodiky BDD nebyla dostatečně pokryta v pracích s podobnou tematikou v době psaní této práce. Dále je třeba zmínit, že tato práce nabízí náhled na problematiku zaměřující se na různé technické možnosti zajištění spouštění testů v prostředí CI. Tím se odlišuje od prací, které se zaměřují na metodiky pro podporu procesu automatizovaného testování.

Testování webových aplikací s využitím nástroje Selenium WebDriver (Třísková, 2015)

Tato práce se zabývá automatizovaným testováním s využitím Selenium WebDriver. Kromě Selenium WebDriver popisuje další nástroje z rodiny Selenium a také nástroje pro zkoumání struktury webové stránky. Součástí práce je vytvoření metodiky pro automatizované testování webových aplikací. Metodika se zabývá celým procesem automatizovaného testování, včetně zúčastněných rolí.

Pokročilé možnosti automatizovaného testování nástrojem Selenium WebDriver (Špalek, 2015)

Jak název napovídá, práce se zabývá pokročilými možnostmi Selenium WebDriver. Obsahuje přehled nejrozšířenějších rozšiřujících rámců a nadstavb tohoto nástroje. Blíže se věnuje dvojici Selenium RemoteWebDriver a Selenium Grid.

Behaviour Driven Development (Vodička 2011)

Obsahem práce je popis metodiky BDD. Její využití je demonstrováno pomocí otestování modelové aplikace s využitím nástroje Cucumber-JVM.

Využití automatizovaných regresních testů v systému kontinuální integrace webové aplikace (Kopalkova 2016)

Práce zabývající se implementací automatizovaných testů webové aplikace v rámci průběžné integrace. Pro implementaci využívá Travis CI a testů psaných v testovacím frameworku pro jednotkové testování JUnit, s využitím Selenium WebDriver.

Automatické testování webového informačního systému (Křištof, 2016)

Práce seznamuje čtenáře s vybranými nástroji pro CI a pro testování ASP.NET MVC aplikací. Praktická část se zabývá implementací automatizovaných testů pro konkrétní ASP.NET MVC aplikaci. K tomu využívá CI nástroje TeamCity a testovacího frameworku pro jednotkové testování MSTest, společně se Selenium WebDriver.

Testování aplikací s využitím nástroje Robot Framework (Mačurová, 2015)

Práce se zaměřuje na nástroj Robot Framework. Ten je v práci detailně popsán. Stěžejní část tvoří navržená metodika pro testování aplikací s využitím tohoto frameworku. Tato metodika vychází z MMSP a zabývá se celým procesem, včetně rolí a jejich úloh.

2.1 Internetové informační zdroje a tištěné odborné publikace

Vybrané internetové informační zdroje a tištěné odborné publikace, které jsou užitečné pro získání znalostí z oblasti automatizace testování webového rozhraní jsou uvedeny níže. Tyto zdroje byly rozčleněny do tří kategorií, pokrývající část problematiky automatizace testování, kterou se zabývá tato práce.

Průběžná integrace, dodávka a nasazení

Velkou součástí automatizace testování je průběžná integrace, dodávka a nasazení. Informace k ní lze získat z webových stránek, zabývajících se agilním vývojem. Agilní vývoj totiž hojně využívá těchto postupů. Mezi tyto zdroje patří stránky neziskové organizace sdružující odborníky v oblasti agilního vývoje Agile Alliance. Na jejich blogu se pravidelně objevují nové články, včetně těch zabývajících se průběžnou integrací. Stránky jsou dostupné zde: <https://www.agilealliance.org/>. Další takovou stránkou je <https://devops.com/>. Na ní pravidelně vychází články týkající se konceptu DevOps. Průběžná integrace, dodávka a nasazení tvoří jeden z jeho základních pilířů, a proto články uvedené na daném portálu se touto problematikou často zabývají. Z tohoto portálu je dostupný také přidružený YouTube kanál, který obsahuje záznamy z přednášek.

Užitečné tištěné publikace jsou Continuous integration: improving software quality and reducing risk (Duvall, 2007) a Continuous delivery: reliable software releases through

build, test, and deployment automation (Humble, 2010). Ty popisují principy těchto postupů, jejich výhody a předpoklady pro úspěšné zavedení do procesu vývoje software.

Selenium WebDriver

Na internetu se vyskytují volně dostupné tutoriály, které se zabývají nástroji založenými na knihovně Selenium WebDriver. Jako příklad lze zmínit:

- <http://toolsqa.com/selenium-webdriver/>
- <http://seleniumsimplified.com/>
- <http://www.seleniumeasy.com/selenium-tutorials/>

Na těchto webových stránkách jsou články popisující práci s těmito nástroji s praktickými příklady. Text je navíc pro lepší pochopení doplněn o video přednášky.

Selenium Testing Tools Cookbook (Gundeča, 2015) je tištěná publikace zabývající se nástroji využívající Selenium WebDriver. Tato publikace představuje funkce knihovny Selenium WebDriver a uvádí příklady některých možností, kterými lze psát skripty založené na této knihovně. Nechybí zde také praktické příklady.

Testování pomocí nástrojů pro automatizaci testů pomocí scénářů metodiky BDD

Pouze se znalostí veškeré nabízené funkcionality lze využít plný potenciál těchto nástrojů. Proto nejužitečnější informace k těmto nástrojům poskytují jejich dokumentace dostupné na internetu. Jako příklad lze uvést dokumentace k nástrojům:

- Robot Framework: <http://robotframework.org/#documentation>
- JBehave: <http://jbehave.org/reference/stable/>
- Cucumber: <https://github.com/cucumber/cucumber/wiki>
- Gauge: <https://docs.getgauge.io/>

Dalším hodnotným zdrojem informací k metodice BDD obecně je webový blog Dana Northa, který je autorem tohoto přístupu k vyvíjení softwaru. Ten je dostupný zde <https://dannorth.net/blog/>.

Mezi tištěné publikace pro získání znalostí z oblasti BDD jsou vhodné knihy BDD in action (Smart 2014), The Cucumber Book (Wynne et al. 2017a) a The RSpec Book (CHELIMSKY 2010). První z jmenovaných představuje obecně principy BDD a zabývá se problematikou zavádění těchto postupů do softwarového vývoje. The Cucumber Book a The RSpec nabízí pohled do problematiky, který je zaměřený na využívání principů BDD za pomoci nástrojů Cucumber, respektive RSpec.

3 Specifikace pojmů

Tato práce je určena čtenářům se zkušeností v oblasti softwarového vývoje. Základní pojmy z oblasti testování softwaru jsou tedy vynechány. V této kapitole jsou specifikovány pouze pojmy, které nespádají do roviny obecně známých termínů v této oblasti. Dále jsou tu pojmy, které jsou často v povědomí IT odborníků, avšak bývají chápány různě, a proto považují za vhodné uvést zde jejich definici.

Pokud není uvedeno jinak, všechny odborné termíny uvedené v této práci jsou myšleny ve smyslu, v jakém jsou popsány v materiálech, vydaných organizací Czech and Slovak Testing Board. Tato organizace je oficiálním regionálním zástupcem ISTQB pro Českou a Slovenskou Republiku. ISTQB, celým názvem International Software Testing Qualifications Board je největší mezinárodní certifikační organizací vydávající globálně uznávané certifikace v oblasti testování softwarového vývoje. Materiály vydané ISTQB, stejně jako jejich oficiální překlady vydané CaSTB, jsou volně dostupné v elektronické podobě na stránkách jednotlivých organizací (CaSTB 2013).

3.1 Continuous integration / Průběžná integrace

Průběžná integrace¹ je proces v softwarovém vývoji, týkající se správy kódu a spolupráce týmu. Jeho stěžejní myšlenkou je častá integrace práce jednotlivých členů týmu, nejméně jednou denně. Součástí každé integrace je automatické sestavení, testování a odeslání zpětné vazby o jejím průběhu vývojářskému týmu (Fowler 2016). Grafické znázornění tohoto procesu lze vidět na obrázku níže.

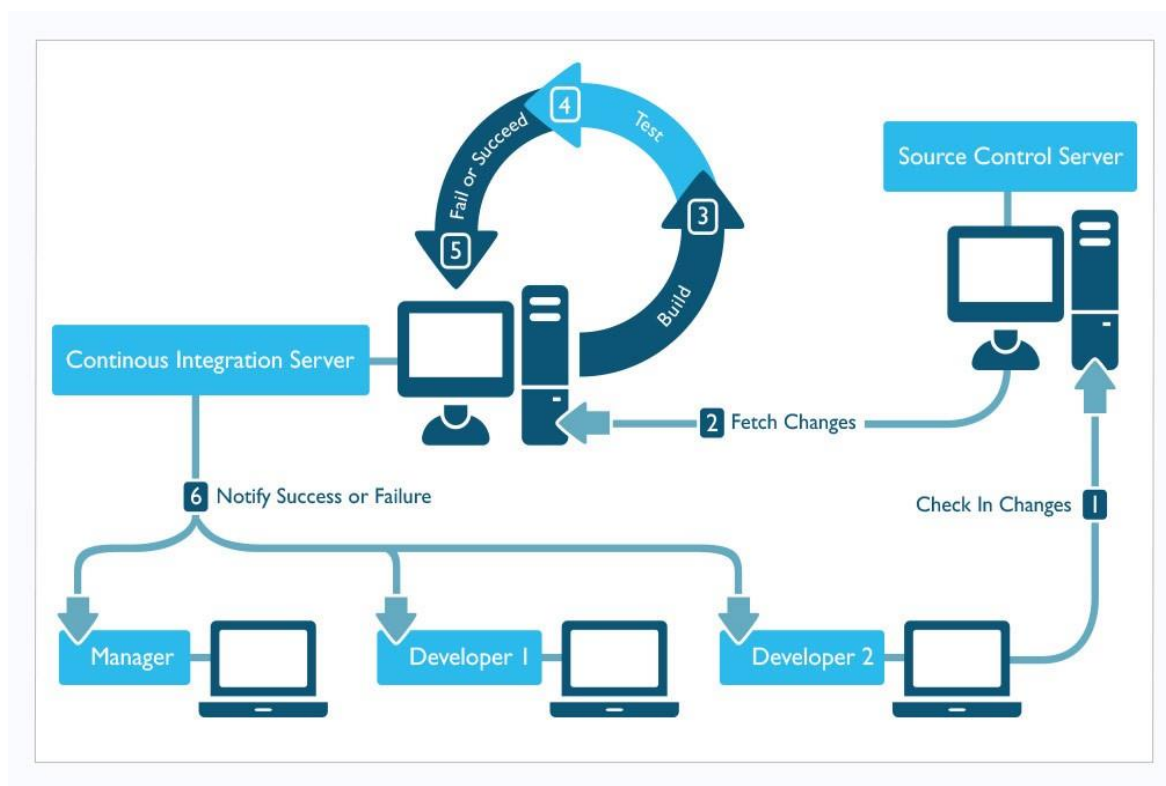
Součástí integrace bývá také měření různých metrik softwarového produktu, jako je například procentuální pokrytí kódu jednotkovými testy, či kontrola dodržování definovaných standardů pro psaní kódu. Spouštěčem procesu průběžné integrace může být každý přírůstek kódu přidáný do verzovacího systému nebo definovaný čas. Často je využívána kombinace obou přístupů. Například po každém přidání kódu proběhne sestavení projektu a provedení časově nenáročných testů a statické analýzy kódu. Každý večer se pak provádí sestavení společně s časově a výkonově náročnými testy.

Průběžná integrace přináší zejména tyto benefity (Duvall 2007):

- Redukce rizik – častější detekce chyb vede k jejich rychlejší opravě
- Redukce opakujících se manuálních procesů

¹ Některé informační zdroje překládají „continuous integration“ jako „kontinuální integrace“. Termín „průběžná integrace“ se však vyskytuje častěji, proto je použit také v této práci.

- Vytvoření softwarového přírůstku připraveného k nasazení
- Zlepšení přehledu o stavu projektu
- Zajištění větší důvěry ve vyvíjený produkt



Obrázek 1: Grafické znázornění procesu průběžné integrace (zdroj: Cois 2015)

3.2 Continuous delivery / Průběžná dodávka

Průběžná dodávka je proces, při kterém je softwarový produkt automaticky nasazován do nějakého prostředí (např. testovacího prostředí), ne však do prostředí, ve kterém je využíván reálnými uživateli. Využívání průběžné dodávky implikuje využívání průběžné integrace (Fitzgerald a Stol 2015).

Svým způsobem ji lze považovat za další stupeň automatizace procesu dodávky softwaru. Znamená to tedy, že před samotným nasazením produkt projde kontrolními prvky v rámci průběžné integrace.

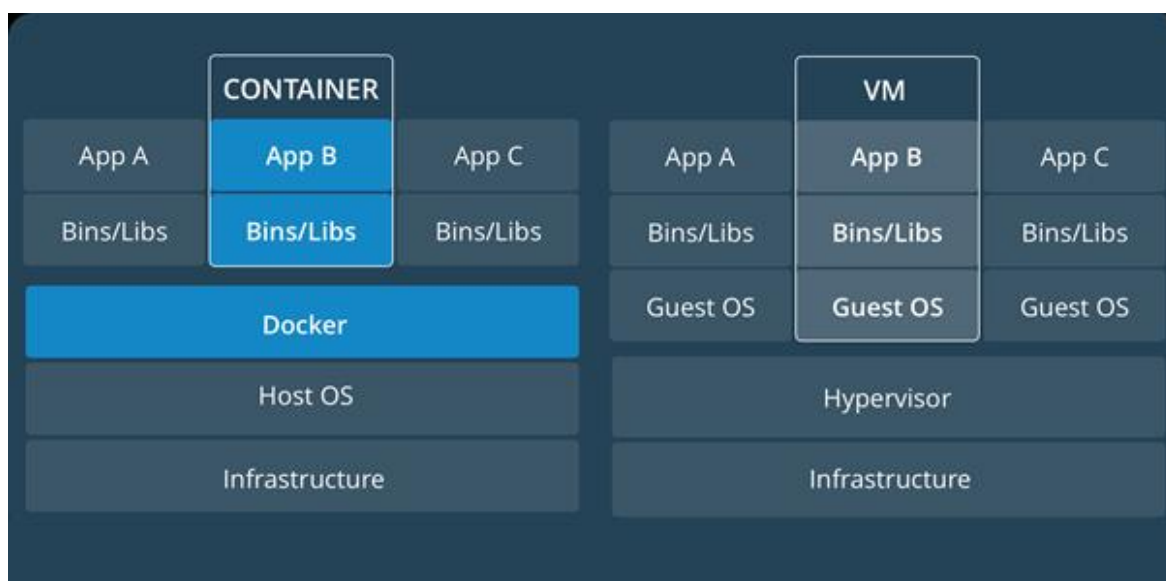
3.3 Continuous deployment / Průběžné nasazení

Průběžné nasazení je proces, při kterém je softwarový produkt průběžně nasazován do prostředí, ve kterém je využíván reálnými uživateli. Využívání průběžného nasazení implikuje využívání průběžné dodávky (Fitzgerald a Stol 2015).

Vzhledem k tomu, že správně zavedený proces je plně automatizovaný, lze jeho pomocí docílit velmi častých dodávek softwaru. Velké firmy jako Amazon, Google či eBay dokáží nasadit hned několik nových verzí za hodinu (Duvall 2007).

3.4 Docker

Docker je open source nástroj umožňující spouštět aplikace v tzv. kontejnerech. Kontejner je abstrakcí na aplikační úrovni, která obaluje do něj vloženou aplikaci a její závislosti. V jedné instanci nástroje Docker lze spouštět několik kontejnerů, které jsou od sebe izolovány. Kontejnery jsou vytvářeny na základě tzv. image, čili obrazů. Tyto obrazy jsou definovány pomocí textového souboru, který se dle jmenné konvence vždy jmenuje *Dockerfile*. Tento soubor pomocí speciální syntaxe definuje všechny součásti, které má daný obraz obsahovat. Na základě jednoho image lze vytvářet libovolné množství kontejnerů. Pokud explicitně kontext nevyžaduje rozlišení běžícího kontejneru a obrazu, termíny obraz, image a kontejner použité v této práci označují Docker image. Principiálně se kontejnery podobají virtuálním strojům. Hlavní rozdíl oproti virtuálním strojům je ten, že virtuální stroje abstrahují celý stroj, kontejnery pouze operační systém, jak je znázorněno na obrázku níže. Díky tomuto rozdílu mají kontejnery oproti virtuálním strojům nižší nároky na výpočetní výkon (Docker 2017a). Společnost IBM zkoumala režii spojenou s během programů v rámci prostředí Docker oproti nativnímu běhu aplikace. Došla k závěru, že režie v podobně výkonu procesoru a paměti potřebná pro samotné zajištění běhu aplikací v Docker kontejneru je zanedbatelná (Felter et al. 2014).



Obrázek 2: Grafické znázornění rozdílů mezi virtuálními stroji a Docker kontejnery (zdroj: Docker 2017b)

Docker je dostupný pro Linux, Windows i Mac. Kontejnery jsou nezávislé na hardwaru i operačním systému, lze tedy například kontejner nakonfigurovaný na platformě Linux spustit v Dockeru běžícím na Windows a obráceně.

Vzhledem k výše uvedeným vlastnostem je Docker velmi užitečným podpůrným nástrojem pro průběžnou integraci, dodávku i nasazení. Odstínění aplikací od prostředí, ve kterém je aplikace spuštěna minimalizuje chyby spojené s rozdílností různých prostředí.

3.5 Behaviour driven development – vývoj řízený požadavky na chování

Behaviour driven development (BDD), přeložitelný jako vývoj řízený požadavky na chování, je metodika agilního vývoje softwaru. Pro testování je nejpodstatnější část metodiky, která se zabývá psaním testů v přirozeném jazyce, které popisují chování aplikace z pohledu uživatele. K ujasnění požadavků využívá příkladů. BDD vzniklo sloučením a zdokonalením praktik vycházejících z metodik TDD – test driven development (vývoj řízený testováním) a ATDD - acceptance test driven development (vývoj řízený akceptačním testováním). Od TDD přejímá důraz na automatizované testování softwaru, které dává okamžitou zpětnou vazbu o stavu aplikace. Návaznost na ATDD spočívá v orientaci na testování na úrovni akceptačních testů čili testování chování z pohledu uživatele. BDD se nejvíce odlišuje od TDD a ATDD těmito vlastnostmi (Agile Alliance 2015a):

- Aplikování „Pěti proč“² principu ke každému uživatelskému příběhu, s cílem jasného propojení s požadavky byznysu
- Myšlení „z venku dovnitř“, čili implementace pouze toho chování, které nejvíce přispívá byznysu, s cílem minimalizovat zbytečnou práci
- Popis chování v jednotné notaci, která je dostupná doménovým expertům, testerům a vývojářům, s cílem zlepšit komunikaci
- Aplikování těchto postupů i na nejspodnějších vrstvách abstrakce softwaru, obzvláště věnuje pozornost distribuci chování aplikace, s cílem zajistit snadný a levný budoucí vývoj aplikace

Hlavním prvkem BDD je popis, jakým způsobem přistupovat k testování vyvíjeného softwaru. Vývoj testů začíná u tzv. user stories – uživatelských příběhů. Ty popisují požadavky na software z pohledu uživatelů. Využívání uživatelských příběhů k popisu požadavků není specialitou BDD. Tato praktika je součástí dalších agilních metodik, jako je například Scrum a XP. Uživatelské příběhy mají v anglickém originále následující strukturu:

As a [X] I want [Y] so that [Z]

X je uživatelská role, Y požadovaná funkcionálnost a Z je benefit, který přináší byznys hodnotu. Do češtiny lze tuto strukturu volně přeložit následujícím způsobem:

Jako [uživatelská role] **chci** [funkcionálnost], **protože** [benefit, byznys hodnota]

Požadavky na aplikaci jsou splněny, pokud se aplikace chová tak jak vyžadujeme. Pro popis chování aplikace se využívá tzv. scenarios – scénářů. Ty pomocí příkladů popisují chování aplikace, které přispívá k naplnění jednotlivých požadavků. Vzhledem k tomu, na jaké úrovni jsou tyto testy psány, řadí se do skupiny akceptačních testů. Každý uživatelský příběh může být popsán vícero scénáři.

Scénáře mají vždy stejnou strukturu, označovanou jako Given-When-Then (Pokud-Když-Pak). Značení pramení ze slov, které tvoří základní stavební kameny dané struktury. Tato struktura vypadá následovně (North 2006):

Pokud [se nacházím v nějakém stavu]

Když [nastane nějaká událost]

Pak [očekávám nějaký výsledek]

Pokud nějaký scénář vyžaduje složitější popis, lze jednotlivé části rozšířit pomocí přidání další věty zřetěžené pomocí spojky „and“ (a). Konkrétní příklady scénářů, použitých

² Metoda „Pěti proč“, v originále „Five Why’s“. Je to metoda sloužící zjištění skutečné základní příčiny nějakého problému či vady. Je založena na pětikrát opakovaném tážení se otázkou „Proč?“. Každá odpověď pak tvoří základ pro následující otázku.

v praxi, jsou uvedeny v praktické části práce. Někdy je tento popis pomocí scénářů označován také jako specification by example – specifikace příkladem (Fowler 2013). Správné využití všech postupů BDD přináší následující výhody (Karam 2017, Agile Alliance 2015c):

- Snadná spolupráce – scénáře představují jednotný jazyk, který je srozumitelný pro všechny, jak pro vývojový tým, tak pro byznys uživatele. Snižuje se tím nepochopení mezi stávajícími účastníky na projektu a ulehčuje zapojení nových.
- Vysoká viditelnost – scénáře jsou čitelné všemi účastníky na projektu a dávají jim tak možnost snadno vidět stav projektu.
- Nástroje pro tvorbu BDD testů v sobě obsahují automatické generování dokumentace z daných scénářů.
- Přináší „živou“ dokumentaci – scénáře dokumentují chování aplikace a dají se spouštět (odtud přívlastek „živá“) jako testy a ověřují tak reálný, aktuální stav. Oproti klasické dokumentaci tato dokumentace nemůže zastarat a popisovat stav minulý.
- Soustředění se na přidanou byznys hodnotu – struktura uživatelských příběhů nutí psát požadavky tak, aby každý požadavek měl uveden jasný benefit, který přináší. Tím se minimalizuje počet vyvíjených požadavků s nulovou přidanou hodnotou pro byznys.
- Soustředění se na hodnotu pro uživatele – struktura scénářů je orientovaná na uživatele a díky tomu umožňuje již při návrhu scénářů objevit případnou zbytečnou funkcionalitu
- Důvěra ve vyvíjený software – testy, které pokrývají scénáře dodávají vývojovému týmu větší důvěru v kvalitu vyvíjeného softwaru
- Snížení času na ladění chyb – srozumitelnost testů a jejich jasná struktura ulehčuje ladění chyb, které jsou pomocí nich nalezeny

Pro vývoj testů stylem popisovaným metodikou BDD existuje několik nástrojů. Pomocí nich lze psát testy v přirozeném jazyce s využitím struktury Given-When-Then. Jednotlivé části popisu scénáře se následně provazují s vlastní implementací, zajišťující automatizovaný chod testu. Implementace se v závislosti na použitém nástroji píše v různých jazycích. Pro většinu dnešních běžně používaných programovacích jazyků existuje nejméně jeden nástroj, který BDD podporuje. Mezi tyto jazyky patří mimo jiné Java, C#, Ruby, PHP, či Javascript. Těmto nástrojům se dále detailněji věnuje kapitola 5.3.

Stejně jako jiné praktiky agilního vývoje, také BDD doporučuje automatizaci. Testy psané touto formou mohou být spouštěny v rámci průběžné integrace, popisované výše. Využití těchto dvou postupů současně se nevyklučuje, naopak tato kombinace umocňuje některé výhody. Například pravidelné spouštění testů po jednotlivých přírůstcích aplikace vede k dalšímu snížení času na ladění chyb a ověřuje aktualitu scénářů, které dokumentují apli-

kaci. Stejně tak stav běhu průběžné integrace, vystavený na místě dostupném pro všechny účastníky, vede k vyšší viditelnosti aktuálního stavu projektu.

Ačkoliv v popisu BDD je často skloňováno slovo byznys, z výhod využívání postupů BDD nemusí těžit jen vývojáři komerčních byznys aplikací. Benefity, které přináší se dají využít také při vývoji open source softwaru. Proto se praktická část této práce zabývá implementací testů pro open source projekt, psaných s využitím technik z metodiky BDD.

4 Přehled vybraných serverů pro podporu průběžné integrace

Průběžná integrace se neobejde bez integračního serveru, který obstarává většinu práce. Vzhledem k velké popularitě průběžné integrace existuje nespočet těchto nástrojů. Tato část práce je věnována základnímu přehledu serverů.

Pro výběr konkrétních řešení byla zvolena kritéria tak, aby bylo docíleno návaznosti na praktickou část práce, která se zabývá vývojem sady testů pro uživatelské rozhraní open source aplikace. Možností, jak automatizovat testy uživatelského rozhraní je nespočet. Všechny mají však jedno společné – pro jejich běh je nutná řada knihoven, utilit a ovladačů. Proto prvním kritériem pro vybrané řešení je možnost instalace knihoven a jiných podpůrných nástrojů dostupných z internetu přímo do integračního prostředí.

Jako další omezující kritérium je stanovena cena. Vývoj většiny open source projektů je neziskový, či pouze na bázi dobrovolných příspěvků. Proto pracuji s předpokladem, že vývojáři open source projektů se nejdříve ohlížejí po bezplatných alternativách. Proto také všechna řešení zařazená do přehledu musejí být dostupná zdarma.

Dále se soustředím pouze na cloudová řešení. Samozřejmě existují také bezplatné servery pro průběžnou integraci, které lze nainstalovat do lokálně spravovaného prostředí (anglicky označované jako *on premise*). Mezi nejpopulárnější z nich patří například Jenkins či TeamCity. Nicméně tyto nástroje vyžadují vyšší nároky jednak na samotné nasazení, tak na údržbu. Je potřeba zajistit hardware na kterém server poběží a následně ho spravovat. Také je nutné instalovat aktualizace všech komponent prostředí ve kterém běží. Z těchto důvodů pro open source projekty považuji vhodnější cloudová řešení. Posledním důležitým kritériem je propojenost s verzovacím nástrojem GitHub. GitHub je totiž současně nejpoužívanější úložiště zdrojových kódů.

Všechny kritéria použité pro výběr řešení uvedených v přehledu, lze shrnout do následujících bodů:

- Pro open source zdarma bez časového omezení či omezení počtu spuštění
- Možnost připojit neomezený počet open source projektů
- Bez omezení počtu přihlášených uživatelů
- Dostupné z cloudu jako služba
- Podpora nástroje Docker
- Propojení s verzovacím nástrojem GitHub
- Stále probíhající vývoj

- Podpora instalace knihoven a jiných podpůrných nástrojů dostupných z internetu přímo do integračního prostředí
- Všechny výše uvedené kritéria musí splňovat bezplatný plán předplatného

Na základě výše uvedených kritérií byly vybrány čtyři nástroje, které nabízejí nejzajímavější řešení zdarma pro open source. Konkrétně jde o tyto nástroje: Travis CI, Circle CI, Semaphore CI a AppVeynor.

4.1 Travis CI

Prvním z vybraných serverů je Travis CI. Tato cloudová služba umožňuje integraci pouze těch repositářů, které jsou uloženy na GitHubu.

Každá integrace běží na nově vytvořeném virtuálním prostředí. Volitelně je možné využít funkce build cache – mezipaměti. Pomocí ní lze označit adresáře, které mají být zachovány mezi jednotlivými běhy integrace. Tato funkce je však defaultně vypnutá, jelikož je stále v experimentální fázi vývoje a nelze se na ni stoprocentně spolehnout.

Na výběr je virtuální stroj s nainstalovaným Ubuntu nebo OS X. Platforma Windows ve výběru zastoupena není. Dostupné nástroje umožňují snadné nastavení prostředí pro integraci většiny projektů určených pro dané podporované platformy. Testování více verzí závislostí³ je řešeno jednoduše pomocí funkce zvané build matrix. Stačí vyjmenovat jednotlivé verze závislostí a nástroj sám najde všechny možné kombinace a provede pro ně integraci (Travis 2017a).

Nastavení kroků integrace se konfiguruje pomocí textového souboru využívajícího yaml syntaxi. Syntaxe yaml je snadno čitelná pro člověka a zároveň vhodná pro serializaci (YAML 2006). Proto je využívána pro konfiguraci různých nástrojů, mezi které patří například cloudové servery pro průběžnou integraci, či nástroj Docker compose použitý v praktické části práce. Ukázka konfigurace Travis CI serveru je uvedena níže.

Výpis 1: Ukázka yaml syntaxe – část konfigurace serveru Travis pro spouštění UI testů pro projekt EasyMiner (zdroj: autor)

```

1 language: generic
2 sudo: false
3 git:
4   submodules: false
5
6 before_install:
7   - git clone -b master --single-branch https://github.com/KIZI/EasyMiner-Tests.git master
8   - docker build -t kizi/easyminer-frontend:dev https://github.com/KIZI/EasyMiner-
  EasyMinerCenter.git#master:/
9   - cd master
```

³ Závislostí je myšlena libovolná komponenta, která je nutná pro úspěšné sestavení aplikace.

Konfigurační soubor musí být uložen v repositáři společně se sestavovaným projektem. Na serveru Travis CI lze navíc ukládat utajované hodnoty. Ty lze následně využívat během integrace, jejich hodnoty jsou však ve výpisu do konzole skryty. Tuto funkci lze tak použít k uchování informací potřebných k dešifrování zašifrovaných souborů, nebo přihlašovacích hesel a tokenů do nejrůznějších služeb.

Notifikace o výsledku běhu lze zasílat nejen mailem, ale také do služeb Slack, HipChat, Flowdock či do IRC. Navíc je možné vygenerovat štítek s výsledkem posledního běhu. Ten lze následně umístit na jakoukoliv webovou stránku (Travis 2017a).

Plán předplatného dostupný zdarma nabízí možnost připojení neomezeného počtu uživatelů a veřejných repositářů. Nabídka zdarma je dostupná pouze pro veřejné repositáře, nikoliv privátní⁴. Dalším omezením je možnost běhu pouze jedné úlohy (nazývané job) současně.

Zejména pro komerční účely je nabízeno celkem sedm placených variant předplatného. Tři z nich jsou nabízeny skrz GitHub Marketplace a čtyři jsou dostupné přímo na stránkách Travis CI. Ty již umožňují spouštět integraci i pro privátní projekty. Kromě cenového rozdílu se každý plán liší pouze maximálním počtem úloh, které je možné pouštět paralelně. Ceník jednotlivých plánů je uveden v tabulce níže. Pokud uživatel zaplatí službu na rok dopředu, bude mu náúčtována cena pouze za jedenáct měsíců (Travis CI 2017b).

Tabulka 1: Plány předplatného služby Travis CI (zdroj: GitHub 2017, Travis CI 2017b)

Název plánu	Max. počet paralelních úloh	Cena na měsíc bez daně
Free	1	0 USD
Bootstrap	1	69 USD
Bootstrap (GH Marketplace)	1	89 USD
Startup	2	129 USD
Startup (GH Marketplace)	3	199 USD
Small Business	5	249 USD
Small Business (GH Marketplace)	6	349 USD
Premium	10	489 USD

Mimo sestavení a spouštění testů umožňuje tento nástroj také nasazování sestavené aplikace. Podporovaných prostředí, do kterých je možné aplikaci nasadit je téměř čtyřicet. Jmenovat lze kupříkladu GitHub Pages, Google App Engine a Storage, Heroku, nebo Azure Web Apps (Travis CI 2017a).

⁴ Uživatelé se studentským GitHub účtem mohou spouštět integrace i pro privátní projekty.

Tento nástroj pro své projekty mimo jiné používá vývojářský tým z katedry KIZI na VŠE, který stojí za vývojem aplikace EasyMiner, jejímž testováním se zabývá praktická část této práce.

Shrnutí hlavních vlastností platné pro verzi dostupnou zdarma pro open source:

- Neomezený počet integrací open source projektů
- Neomezený počet uživatelů
- Prostředí Linux nebo OS X
- Maximálně jedna souběžně běžící úloha
- Konfigurace pomocí textového souboru s yaml syntaxí
- Notifikace zasílané emailem, či do služeb Slack, Hipchat, Flowdock či IRC

Výhody a silné stránky

- Snadné nastavení
- Uživatelsky přívětivá funkce pro ukládání utajených hodnot
- Rozsáhlá a srozumitelná dokumentace
- Snadné testování více verzí závislostí pomocí funkce build matrix
- Pro open source je možné zdarma integrovat i v prostředí OS X
- Rozsáhlá podpora pro nasazení sestavovaných aplikací do různých prostředí

Nevýhody a slabé stránky

- Chybějící podpora napojení na BitBucket repositáře (dostupný pouze GitHub)
- Chybějící možnost integrace v prostředí Windows
- Pro open source projekty placené plány s přidanou hodnotou začínají až na 129 USD/měsíc
- Plán dostupný zdarma nenabízí žádnou možnost integrace privátních repositářů
- Bezplatný plán umožňuje mít spuštěnu pouze jednu integraci v jeden okamžik

4.2 Semaphore CI

Dalším cloudovým nástrojem pro průběžnou integraci, který nabízí zajímavou nabídku pro open source projekty je Semaphore CI. Tento nástroj umožňuje integraci projektů uložených na GitHubu či BitBucketu.

Integrace probíhají vždy na nově vytvořených virtuálních prostředích. Jednotlivé integrace jsou tedy na sobě nezávislé, vyjma dostupné mezipaměti, která je zachována mezi jednotlivými běhy. Tato mezipaměť je umístěna ve speciálním adresáři daného virtuálního prostředí. Tato cache funguje jako klasická mezipaměť – lze do ní ukládat libovolné soubory, ale ty jsou v ní jen dočasně. Proto integrační skripty, které ji chtějí využívat musí počítat i s tím, že očekávané soubory v ní nebudou. Avšak při správném využití tato funkce umožňuje urychlit průběh integrace.

Výhodou Semaphore CI je možnost připojit se do integračního prostředí pomocí SSH. Toto připojení umožňuje ladění integrace. Jedinou dostupnou platformou je Linux, konkrétně distribuce Ubuntu. Další platformy nejsou dostupné.

Všechna konfigurační probíhá pomocí webového rozhraní. Podporováno je jedenáct programovacích jazyků. Nechybí mezi nimi žádný z populárních jazyků z Linuxového prostředí jako např. C/C++, PHP, Python, Java, Javascript či Ruby. Podporu dalších je možno zajistit pomocí nainstalování linuxových programových balíčků, či pomocí nástroje Docker.

Mimo tradiční notifikace zasílané elektronickou poštou, jsou podporovány notifikace do služeb Campfire, Flowdock, HipChat a Slack. Nechybí také možnost sledovat stav pomocí štítku, který lze umístit na libovolný web (Rendered Text 2017a).

Bezplatný plán předplatného nabízí možnost připojení neomezeného počtu uživatelů a repositářů. Pro open source projekty je možné integrace spouštět neomezeně. K dispozici je možnost paralelního běhu až dvou úloh. V případě privátních repositářů je umožněno pouze sto integrací měsíčně.

Placené plány předplatného začínají na 25 USD měsíčně. Existuje čtrnáct variant, které se od sebe liší pouze maximálním možným počtem spuštěných integrací současně. Pro všechny plány platí možnost neomezeného spouštění integrací jak pro veřejné, tak i privátní repositáře. Ceník prvních šesti nejlevnějších variant plánů je uveden v tabulce níže. Měsíční poplatky se liší podle způsobu vyúčtování. Roční vyúčtování je tradičně výhodnější oproti měsíčnímu. Pro akreditované vzdělávací instituce je nabízeno spouštění až čtyř integrací současně (i pro privátní repositáře) zdarma. Navíc další navýšení limitu pomocí placených plánů je poskytováno se slevou 25 % (Rendered Text 2017b).

Tabulka 2: Vybrané plány předplatného služby Semaphore CI (zdroj: Rendered Text 2017b)

Max. počet paralelních úloh ⁵	Cena na měsíc bez daně při ročním vyúčtování	Cena na měsíc bez daně při měsíčním vyúčtování
1	25 USD	29 USD
2	83 USD	99 USD
4	166 USD	199 USD

⁵ Pro veřejné repositáře platí omezení o dvě vyšší, navýšené o dvě paralelní úlohy zdarma.

8	332 USD	399 USD
12	499 USD	599 USD
16	665 USD	799 USD

Tento nástroj neslouží jen k průběžné integraci, ale nabízí také možnost nasazení sestavných aplikací. Obsahuje nástroje pro podporu osmi různých způsobů nasazení, mezi které mimo jiné patří nasazení do prostředí hostovaných službami Heroku, Cloud 66, nebo AWS Elastic Beanstalk a Lambda (Rendered Text 2017a).

Hlavní vlastnosti platné pro verzi dostupnou zdarma pro open source:

- Neomezený počet integrací open source projektů
- Neomezený počet uživatelů
- Prostředí Linux
- Maximálně dvě souběžně běžící úlohy
- Konfigurace pomocí webového rozhraní
- Notifikace zasílané emailem, či do služeb Campfire, Flowdock, HipChat a Slack
- Možnost běhu sto integrací privátních projektů za měsíc zdarma

Silné stránky

- Možnost integrace projektů ze systémů GitHub i Bitbucket
- Možnost ladění běhu pomocí SSH připojení do integračního prostředí
- Sto integrací privátních projektů za měsíc zdarma
- Možnost běhu dvou integrací současně pro open source zdarma

Slabé stránky

- Integrace pouze na platformě Linux – Ubuntu
- Nastavení čistě pomocí webového rozhraní neumožňuje verzování konfigurace integrace
- Složitější nastavení integrace oproti více verzím závislostí (chybí funkce build matrix)

4.3 Circle CI

Dalším cloudovým nástrojem pro průběžnou integraci, který nabízí zajímavou nabídku pro open source projekty je Circle CI. Ten se dokáže propojit s projekty uloženými na GitHubu či Bitbucketu.

Jednotlivé úlohy jsou vždy spuštěny na nově vytvořeném virtuálním prostředí. K dispozici je podobně jako u Semaphore CI mezipaměť, uchovávaná mezi jednotlivými úlohami. I v případě Circle CI je však nutné při jejím užívání neopomenout fakt, že soubory dříve uložené v cache nemusí být při dalším běhu dostupné. V případě potíží s průběhem integrace je možné se připojit do integračního prostředí pomocí SSH připojení.

Mezi nabízené platformy pro integraci patří Linux Ubuntu či OS X. Díky tomu Circle CI podporuje integraci aplikací psaných v mnoha různých programovacích jazycích frameworkích. Mezi ně mimo jiné patří C++, Java, Javascript, PHP, Node.js, Python, Ruby či Scala. Nastavení integrace probíhá prostřednictvím yaml souboru, který je nutné přiložit do repository s projektem (CircleCI 2017a).

Plány předplatného jsou rozděleny do dvou skupin dle platformy, Linux a OS X. Pro všechny varianty platí možnost připojení neomezeného počtu uživatelů a repository. Slevy za platbu na rok dopředu či pro studenty v nabídce chybí.

Na platformě Linux je možné zdarma spouštět až čtyři úlohy současně pro open source projekty. Pro privátní projekty je k dispozici možnost jedné souběžné integrace zdarma s limitem 1500 minut běhu na měsíc. Za umožnění spouštění jedné paralelní úlohy navíc a odstranění limitu pro počet integrací privátních repository se platí 50 USD za měsíc. Každé další navýšení je zpoplatněno stejně. Pro jednodušší porovnání ceny se zbývajícími řešeními v přehledu je cena několika variant uvedena v tabulce níže. Plány předplatného jsou nabízeny v několika variantách také přes GitHub Marketplace. Tam jsou však ceny vyšší, v závislosti na plánu až o 19 dolarů na jeden slot pro jednu paralelní úlohu.

Varianta Circle CI pro integraci na platformě OS X je nabízena prostřednictvím čtyř placených plánů. Ceník všech plánů je uveden v tabulce níže. Integrace na platformě OS X jsou omezeny měsíčním časovým limitem běhu. Na rozdíl od ostatních plánů v přehledu tedy nesplňují OS X plány kritérium pro neomezený počet integrací. Nicméně vzhledem k možné kombinaci s neomezenou integrací na platformě Linux je uvedena i tato nabídka.

Bezplatná integrace na OS X pro veřejné repository je dostupná až po komunikaci s týmem Circle CI. Po případném schválení je poskytovatelem aktivován plán předplatného Seed, který je možné využívat jen pro open source projekty (CircleCI 2017b).

Tabulka 3: Vybrané plány předplatného služby Circle CI pro integraci na Linuxu (zdroj: CircleCI 2017b)

Počet placených slotů pro paralelní úlohy	Max. počet paralelních úloh Pro open source / privátní projekty	Cena na měsíc bez daně
Žádný	4 / 1 ⁶	zdarma
2	7/4	200 USD
4	11/8	199 USD
8	15/12	399 USD

Tabulka 4: Vybrané plány předplatného služby Circle CI pro integraci na OS X (zdroj: CircleCI 2017b)

	Max. počet paralelních úloh	Počet předplacených minut na měsíc	Cena za minutu při překročení limitu	Cena na měsíc bez daně
Seed	2	500	0,08	39 USD
Startup	5	1800	0,08	129 USD
Growth	7	5000	0,05	249 USD
Mobile Focused	12	25000	0,035	449 USD

Stav integrace jednotlivých projektů lze sledovat pomocí email notifikací, nebo lze využít propojení s dalšími komunikačními službami jako je Slack, HipChat, Campfire, Flowdock a IRC. Projekty lze po sestavení nasazovat přímo do prostředí vybraných cloudových služeb. Mezi ty podporované nástroje Circle CI patří služby Microsoft Azure, Google Cloud, Heroku a Amazon Web Services (CircleCI 2017a).

Shrnutí hlavních vlastností platné pro verzi dostupnou zdarma pro open source:

- Neomezený počet integrací open source projektů
- Neomezený počet uživatelů
- Prostedí Linux nebo (s omezením) OS X
- Až čtyři souběžně běžící úlohy
- Konfigurace pomocí textového souboru s yaml syntaxí
- Notifikace zasílané emailem, či do služeb Slack, HipChat, Campfire, Flowdock a IRC

⁶ Pro privátní projekty je běh integrací limitován na maximálně 1500 minut za měsíc

Výhody a silné stránky

- Možnost ladění běhu pomocí SSH připojení do integračního prostředí
- Možnost paralelizace až čtyř úloh pro open source projekty zdarma
- 1500 minut zdarma pro integrace privátních repositářů

Nevýhody a slabé stránky

- Chybějící možnost integrace v prostředí Windows
- Integrace na platformě OS X mají měsíční časový limit jejich běhu
- Chybějící možnost snadného nastavení jedné integrace s různými verzemi závislostí

4.4 AppVeyor

AppVeyor je cloudovým řešením pro průběžnou integraci aplikací postavených na platformě Windows. Právě podpora platformy Windows je hlavní vlastností této služby, která ji odlišuje od ostatních řešení. Ke této službě je možné připojit repositáře uložené na GitHubu, BitBucketu, nebo také ve službě Visual Studio Online.

Stejně jako ostatní servery v tomto přehledu, AppVeyor využívá pro každý běh integrace nové virtuální prostředí. Podobně jako Semaphore CI a Circle CI tato služba umožňuje urychlit průběh integrace pomocí využití cache paměti. Rozdílem ovšem je, že tato služba garantuje, že soubory uložené do cache budou vždy dostupné mezi jednotlivými integracemi. To výrazně zjednodušuje nastavení jejího využívání v rámci integrace. Limitem je pouze její velikost, která se liší dle zvoleného plánu předplatného.

Průběh integrace je možné sledovat připojením k virtuálnímu serveru pomocí protokolu RDP. Ladění lze tedy provádět pomocí nástroje „Připojení ke vzdálené ploše“, který je součástí většiny instalací Windows.

Dostupné jsou tři konfigurace prostředí lišících se zejména nainstalovanou verzí Visual Studia a .Net frameworku. Prostředí lze konfigurovat pomocí yaml souboru přidaného do repositáře s projektem, či čistě přes webové rozhraní. Tato konfigurace také umožňuje jednoduše nastavit spuštění integrace pro více verzí závislostí pomocí funkce build matrix.

Mezi plně podporované programovací jazyky mimo jiné patří C#, F#, VB, C++, Ruby, Java, Python, Javascript (včetně frameworků Node.js a io.js). Nechybí zde možnost doinstalovat do prostředí další závislosti z internetu, stejně jako podpora nástroje Docker. Limitem je tedy pouze platforma Windows.

Upozornění o stavu průběžné integrace je možné zasílat emailem nebo do komunikačních kanálů Slack, HipChat, VSO Team Rooms, či Campfire. Dostupná je také desktopová

aplikace pro počítače Windows a Mac, která informace o stavu zobrazuje v podobě tzv. toast notifikací (Appveyor Systems 2017a).

Pro open source projekty je nabízen jeden plán předplatného zdarma. Jeho součástí je možnost připojení neomezeného počtu uživatelů a veřejných repositářů. Umožňuje mít spuštěnu jednu integraci v jeden okamžik. Pro paralelní běh nebo integraci soukromých repositářů je třeba přejít na placené předplatné. Celkem jsou nabízeny tři varianty placeného předplatného. Popis jednotlivých nabízených variant je uveden v tabulce níže. Tyto plány lze předplatit také přes GitHub Market za stejnou cenu. Při ročním vyúčtování se platí pouze cena za deset měsíců namísto dvanácti. Ke každému plánu lze za 50 USD na měsíc dokoupit jeden slot pro běh jedné úlohy paralelně navíc. Pro studenty, vzdělávací organizace a open source projekty je nabízena sleva 50 % na všechny placené plány (Appveyor Systems 2017b).

Tabulka 5: Varianty plánů předplatného služby AppVeynor (zdroj: Appveyor Systems 2017b)

Název plánu	Max. počet paralelních úloh	Maximální počet privátních projektů	Velikost cache	Cena na měsíc bez daně při měsíčním vyúčtování
Open source	1	0	1 GB	zdarma
Basic	1	1	1 GB	29 USD
Pro	1	Neomezený	5 GB	59 USD
Premium	2	Neomezený	20 GB	99 USD

AppVeynor není nástrojem pouze pro průběžnou integraci, ale nabízí také možnost průběžné dodávky či nasazení. K dispozici jsou nástroje pro podporu nasazování aplikací do služeb Azure, Amazon či Bitray (Appveyor Systems 2017a). Podpora pro nasazování do prostředí služby Heroku či služeb od firmy Google na rozdíl od některých konkurenčních řešení chybí.

Hlavní vlastnosti platné pro verzi dostupnou zdarma pro open source:

- Neomezený počet integrací open source projektů
- Neomezený počet uživatelů
- Prostředí Windows
- Maximálně jedna souběžně běžící úloha
- Nastavení pomocí webového rozhraní nebo konfiguračního souboru
- Notifikace zasílané emailem a do služeb Slack, HipChat, VSO Team Rooms, či Campfire

Silné stránky

- Podpora platformy Windows

- Snadné testování více verzí závislostí pomocí funkce build matrix
- Možnost ladění běhu pomocí RDP připojení do integračního prostředí
- Možnost persistence souborů mezi jednotlivými běhy integrace

Slabé stránky

- Nepodporuje jiné platformy než Windows
- Plán dostupný zdarma nenabízí žádnou možnost integrace privátních repositářů
- Bezplatný plán umožňuje mít spuštěno pouze jednu integraci v jeden okamžik
- Menší počet nástrojů pro nasazení sestavených aplikací než u konkurenčních řešení v přehledu

4.5 Shrnutí přehledu

V této části práce byla přestavena čtyři cloudová řešení pro průběžnou integraci. Výběr byl soustředěn na nástroje nabízející integraci pro open source projekty zdarma. Daná řešení byla vybraná na základě předem stanovených kritérií. Paleta nabízených funkcí je u všech velmi podobná. Mezi sebou se odlišují zejména rozdílností uživatelského rozhraní. Přívětivost uživatelského rozhraní však nelze objektivně porovnat, ta je vždy subjektivní. Z vybraných nástrojů tedy nelze určit jeden jako jednoznačně nejlepší. Výběr konkrétního nástroje vždy závisí na kontextu, ve kterém má být použit. Naštěstí lze každý z těchto nástrojů zdarma vyzkoušet a posoudit, který je nejvíce vyhovující pro dané použití.

Silné a slabé stránky jsou zmíněny v popisu jednotlivých nástrojů. Shrnutí vybraných parametrů těchto nástrojů je uvedeno v tabulce níže. Travis CI nabízí integraci na platformách Linux i OS X. Vyniká zejména snadným nastavením, rozsáhlou dokumentací a množstvím nástrojů pro podporu nasazení sestavených aplikací. AppVeyor se vyznačuje podporou integrace na platformě Windows. Proto se hodí zejména na projekty, které jsou vyvíjeny čistě pro tuto platformu. Circle CI vyniká možností mít až čtyři spuštěné úlohy na jednou. Hodí se tak tam, kde probíhá vývoj intenzivně a rychlost zpětné vazby má velkou váhu. Poslední Semaphore CI s podporou platformy Linux nabízí alternativu zejména k Circle CI či Travis CI. Jeho nastavení probíhá celé přes webové rozhraní, což může některým uživatelům vyhovovat více než přes soubor přidaný do repositáře.

Vzhledem ke kritériím, na jejichž základě byli vybráni daní reprezentanti, každý z těchto nástrojů důstojně zastane činnost průběžné integrace, dodávky či nasazení. Všechna řešení pro open source projekty nabízí zdarma stejné nástroje, které jsou pro privátní projekty zpoplatněny. Daný přehled tak ukazuje, že pro open source projekty lze řešit průběžnou integraci bez finančních nákladů. Vzhledem k tomu existuje jen málo důvodů, proč nevyužívat výhod průběžné integrace také u open source projektů.

Tabulka 6: *Shrnutí parametrů platných pro bezplatné verze a open source projekty vybraných CI řešení⁷ (zdroj: autor)*

	Travis CI	Semaphore CI	AppVeyor	Circle CI
Podporované platformy	Linux, OS X	Linux	Windows	Linux, OS X
Max. souběžných úloh	1	2	1	Linux: 4 / OS X: 2
Způsob nastavení integrace	YAML	Web UI	YAML, Web UI	YAML
Napojení na repositáře	GitHub	GitHub	GitHub, Bitbucket, VSO	GitHub, Bitbucket
Možnost integrace privátních repositářů zdarma	Není	100 běhů / měsíc	Není	1500 min / měsíc
Maximální počet sestavení na měsíc	Bez limitu	Bez limitu	Bez limitu	Linux: bez limitu OS X: 500 min
Max. počet připojených repositářů	Bez limitu	Bez limitu	Bez limitu	Bez limitu
Maximální počet uživatelů	Bez limitu	Bez limitu	Bez limitu	Bez limitu
Nástroje pro nasazení do Heroku/AWS/Google Cloud/Azure	Ano/Ano/Ano/Ano	Ano/Ano/ Ne/Ne	Ne/Ano/ Ne/ Ano	Ano/Ano/ Ano/Ano

⁷ Tabulka vytvořena autorem s využitím informací z webových stránek uvedených produktů, data platná k 31. 07. 2017

5 Přehled softwarových nástrojů pro automatizované testování webového rozhraní

Nástroje pro automatizované testování webového rozhraní lze rozdělit podle techniky použité k automatizaci do dvou skupin. Jednu skupinu tvoří nástroje zachycující aktivitu uživatele, kterou pak dokáží opakovaně automaticky přehrávat. Druhou skupinu lze označit jako nástroje pro testování pomocí skriptů. Tyto nástroje vykonávají svou činnost čistě na základě skriptů, které je třeba naprogramovat v nějakém programovacím jazyce (ISQTB 2012).

Pro obě dvě skupiny těchto nástrojů platí, že ve většině případů interně využívají Selenium WebDriver. Selenium WebDriver je open source knihovna, která poskytuje programové rozhraní pro ovládání webových prohlížečů. Mezi její funkce patří objevování DOM elementů stránek. Umožňuje také manipulaci s těmito objekty. Příkladem může být provádění kliknutí, stisknutí klávesy, či vyplnění textových polí. Toto aplikační rozhraní volá vestavěné funkce pro automatizaci dostupné v jednotlivých prohlížečích skrze ovladač, který implementuje toto rozhraní. Pro každý prohlížeč existuje separátní ovladač. Díky tomuto oddělení závislosti na daném prohlížeči skrz rozhraní je možné použít jeden kód psaný s využitím Selenium WebDriver pro testování více prohlížečů (Selenium project 2012). Aktuálně existuje návrh pro začlenění WebDriverů jako standardu v rámci mezinárodního konsorcia pro webové standardy W3C. V době psaní této byl tento návrh v druhém stavu schvalovacího cyklu, ve stavu tzv. „Candidate Recommendation“, čili návrhu předloženého komunitě k poskytnutí zpětné vazby (Burns a Stewart 2017).

Vzhledem k velkému množství dostupných nástrojů, není možné se v této práci věnovat všem detailně. Tato kapitola proto obsahuje pouze popis různých přístupů k automatizaci, společně s příklady nástrojů, které je podporují. Bližší prostor je v této kapitole věnován nástrojům podporujícím psaní testů formou BDD scénářů. Důvodů, proč má smysl se zabývat touto konkrétní skupinou nástrojů, je hned několik. Vzhledem k tomu, že scénáře popisují chování aplikace z pohledu uživatele, hodí se pro testování na úrovni uživatelského rozhraní. Dále nástroje pro testování pomocí BDD scénářů umožňují z jednotlivých scénářů vygenerovat rozsáhlou dokumentaci. Navíc tato dokumentace je tvořena spustitelnými testy a je tedy zaručeno, že popisuje aktuální stav aplikace a nezastarává tak jako dokumentace klasická. Popis BDD společně s výčtem dalších benefitů lze nalézt v kapitole 3.5.

5.1 Nástroje pro zachycení a přehrávání aktivity uživatele

První skupinu tvoří nástroje, pro které se v anglických publikacích používá označení „record and playback”. Tyto nástroje zachycují aktivity uživatele provádějícího v rozhraní aplikace jednotlivé testovací případy a umožňují jejich přehrávání. Vyhodnocení úspěšnosti testů závisí na porovnání určité proměnné a její očekávané hodnoty po skončení testu. Tato technika má za cíl odstranit nutnost disponovat technických znalostmi při automatizaci testování softwaru. Jednoduchost ovládání tohoto typu nástrojů je vykoupena řadou nevýhod. Jednou z nich je velká citlivost na změny ovládacích prvků aplikace. I drobné změny uživatelského rozhraní mohou způsobit selhání testů a nutnost jejich opětovného nahrání. Další nevýhodou je, že vstupy jsou zaznamenávány jako statické hodnoty. V neposlední řadě je problém nízká modularita, které znesnadňuje znovupoužití jednotlivých částí. Problémem je také ošetření chybových stavů. Navzdory uvedeným nevýhodám, tyto nástroje mohou najít své uplatnění v pozdějších fázích vývoje, k testování základní a neměnné funkcionality, u které se neočekávají časté změny (Havlíčková a Roudenský 2013).

Některé nástroje umožňují vygenerovat z nahraného scénáře textový skript, který lze následně upravit. Tyto úpravy však vyžadují technické znalosti, tudíž použití této funkce eliminuje hlavní výhodu těchto nástrojů. Vzhledem k automatickému generování tyto skripty navíc nejsou dobře strukturované, ani lehce čitelné.

Mezi nástroje, které obsahují funkci pro zachycení a přehrávání aktivity uživatele patří:

- Selenium IDE
- Sahi
- Ranorex
- Microsoft Test Manager
- MarathonITE
- CloudQA
- TestingWhiz
- Rاپise
- TestComplete
- Squish for Web

5.2 Nástroje pro automatizované testování pomocí skriptů

Mezi tyto nástroje lze zařadit všechny nástroje, které umožňují vytvářet automatizované testy pomocí skriptů, napsaných v libovolném programovacím jazyce. Tyto nástroje lze

dále dělit to několika podskupin. Jedno z možných dělení popisuje Fewster a Graham (Fewster a Graham 1999), kteří rozdělují nástroje do pěti kategorií – lineární skripty (linear scripts), strukturované skripty (structured scripts), daty řízené skripty (data-driven scripts) a skripty řízené klíčovými slovy (keyword-driven scripts).

5.2.1 Lineární skripty

Nejjednodušší forma automatizovaných testovacích skriptů. Pro tyto skripty se vyznačují tím, že všechny jimi popsane kroky detailně popisují jednotlivé akce za sebou. Jedná se o pouhý seznam jednotlivých akcí. Chybí zde konstrukce známé z programovacích jazyků jako jsou cykly, podmínky či možnost seskupení několika kroků do jedné uživatelem definované funkce. Díky tomu je výrazně omezeno znovupoužití jednotlivých částí a kódy skriptů jsou mnohem delší než ty vytvořené pokročilejšími metodami. To vede k vysokým nárokům na údržbu, protože i malá změna v aplikaci může vést k nutnosti úpravy mnoha řádků kódu. Dalším problémem při údržbě je fakt, že součástí těchto testů jsou na pevně definovaná testovací data. Vzhledem k uvedeným omezením je psaní těchto skriptů velmi časově náročné. Proto se tyto skripty výhradně generují pomocí nástrojů pro zachycení a přehrávání aktivity uživatele. Vygenerované skripty se poté případně pouze upravují, nikoliv píšou manuálně celé od začátku (Fewster a Graham 1999, 3QI LABS 2015). Příklady takových nástrojů jsou uvedeny na konci kapitoly 5.1.

5.2.2 Strukturované skripty

Dalším stupněm automatizovaných skriptů tvoří takzvané strukturované skripty. Tyto skripty jsou lineární skripty obohacené o konstrukce známé z programovacích jazyků jako jsou cykly, podmínky či možnost seskupení několika kroků do jedné uživatelem definované funkce. Díky tomu je možné skripty lépe strukturovat a umožnit opakovatelné použití jednotlivých částí. Při dobrém návrhu jednotlivých funkcí lze výrazně omezit množství změn v testovacích skriptech při změně aplikace až na pouhé jednotky řádků kódu. Nevýhodou těchto skriptů jsou však data, která jsou stále data pevně svázaná s kódem popisujícím jednotlivé akce. Další slabina využití těchto nástrojů je nutnost větších technických znalostí, než při použití jednodušších nástrojů pro zachycení a přehrávání aktivity uživatele (Fewster a Graham 1999, 3QI LABS 2015).

Skripty tohoto typu lze vytvářet pomocí testovacích frameworků určených pro jednotkové testování. Pro ovládání prohlížeče a kontrolu zobrazených stránek tyto testy využívají kombinace Selenium WebDriver a ovladače pro testovaný prohlížeč. Mezi jazyky plně podporující Selenium WebDriver patří Python, Ruby, Java a C#. Volba frameworku pro testování je závislá na vybrané platformě. Příkladem těchto frameworků jsou pro Javu JUnit, TestNG, pro C# MS Test, xUnit a NUnit, pro Ruby Test-unit, nebo pro Python PyUnit.

Nástroje pro podporu psaní a spouštění jednotkových testů jsou součástí běžných IDE pro psaní kódu v uvedených programovacích jazycích.

5.2.3 Daty řízené skripty

Daty řízené skripty jsou skripty, které mají oddělena testovací data od testovacího skriptu popisujícího provádění akce. Tyto skripty postupně načítají záznamy z úložiště testovacích dat. Tyto data se pak používají jako vstupní hodnoty a ke kontrole výstupních hodnot. Díky tomuto postupu je možné snadno použít jeden skript k otestování několika různých situací. V závislosti na použitém nástroji mohou být data uložena v různých formátech. Mezi využívanými úložišti dat jsou csv, xml nebo json soubory, Excel tabulky či databáze.

Pro tvorbu daty řízených skriptů lze použít stejné nástroje, jako je tomu u skriptů strukturovaných skriptů. Dnešní frameworky pro jednotkové testy také podporují načítání dat z externích úložišť. Stejně tak lze použít nástroje popisované v následující části věnující se skriptu řízené klíčovými slovy.

Mezi slabiny tohoto přístupu patří větší počáteční časová náročnost vývoje, než je tomu u strukturovaných testů, vyplývající z nutnosti oddělení dat od skriptu. Avšak tento počáteční náklad se v budoucnu vrátí v podobě snadnější údržby a provádění změn. Ačkoliv využití nástrojů pro automatizaci z této kategorie vyžaduje velké technické znalosti, definování testovacích dat není technicky náročné. Proto mohou nové testovací případy přidávat také testéři bez zkušeností s automatizací pouhou změnou dat. Z této výhody lze nejvíce těžit při testování aplikací s mnoha formuláři, jejichž vstupy mají mnoho okrajových podmínek (Fewster a Graham 1999, 3QI LABS 2015).

5.2.4 Skripty řízené klíčovými slovy

Poslední kategorií skriptů pro automatizované testování tvoří skripty řízené klíčovými slovy. Ty se vyznačují oddělením popisu akcí testovacího případu od implementace logiky, které obstarává provedení jednotlivých akcí. Pro popis akcí se využívá klíčových slov, které jsou psány vysokoúrovňovým jazykem, který vytvářejí experti znající danou doménu, kterými jsou například test analytici (ISQTB 2012). Příklad scénáře psaného klíčovými slovy s využitím syntaxe jazyka Gherkin je uveden níže. Na ukázce je vidět, jak syntaxe jazyka Gherkin připomíná přirozený jazyk. Pro pochopení scénáře není třeba znalost programovacího jazyka použitého pro automatizaci. Jazyk Gherkin, použitý v ukázce, využívá řada BDD frameworků, mezi které mimo jiné patří Cucumber, Specflow, či Behat.

Výpis 2: Ukázka scénáře psaného syntaxí jazyka Gherkin, podporovaného BDD frameworkem Cucumber (zdroj: Wynne et al. 2017b)

- 1 Feature: Cash Withdrawal
- 2 Scenario: Successful withdrawal from an account in credit
- 3 Given I have deposited \$100 in my account

```

4   When I withdraw $20
5   Then $20 should be dispensed

```

Jednotlivá klíčová slova jsou následně provázána s vlastní implementací zajišťující vykonání akcí. Tato implementace je pak provedena s využitím programovacího jazyka stejně jako u strukturovaných skriptů. Ukázka z implementace klíčových slov je uvedena níže. Implementace v ukázce je napsaná v jazyce Ruby s využitím frameworku Cucumber. Navazuje tak na předchozí ukázku, ve které je uveden příklad klíčových slov.

Výpis 3: *Ukázka implementace klíčových slov v jazyce Ruby, podporovaného BDD frameworkem Cucumber (zdroj: Wynne et al. 2017b)*

```

1 Given /^I have deposited ({CAPTURE_CASH_AMOUNT}) in my account$/ do |amount|
2   my_account.deposit(amount)
3   my_account.balance.should eq(amount),
4   "Expected the balance to be #{amount} but it was #{my_account.balance}"
5 end
6
7 Then /^(#{CAPTURE_CASH_AMOUNT}) should be dispensed$/ do |amount|
8   cash_slot.contents.should == amount
9 end
10
11 When /^I withdraw ({CAPTURE_CASH_AMOUNT})$/ do |amount|
12   teller.withdraw_from(my_account, amount)
13 end

```

Separace klíčových slov od jejich implementace má řadu výhod. Popis akcí je psán pomocí přirozeného jazyka, kterému rozumí analytici a testéři bez znalosti automatizace. Proto také oni mohou vytvářet nové automatizované testovací případy, pokud specialisté na automatizaci vytvořili dostatečně velké množství klíčových slov. Zároveň díky snadné čitelnosti je možné použít popis klíčovými slovy jako dokumentaci chování aplikace. Separace také umožňuje měnit implementace klíčových slov, bez zásahu do definice testovacích případů, které je využívají. Údržba těchto skriptů po změnách v aplikaci je tedy snazší. Využívání těchto skriptů je náročné nejen na technické znalosti. Jen správný návrh testů umožňuje těžit z výhod tohoto přístupu, a proto tento návrh vyžaduje větší počáteční časovou investici oproti jednodušším metodám (Fewster a Graham 1999, 3QI LABS 2015).

Do této kategorie patří mimo jiné nástroje pro tvorbu testů formou BDD scénářů. Ty kombinují výhody přístupu řízeného klíčovými slovy s výhodami metodiky BDD. Od ní přebírají zejména pevnou, srozumitelnou strukturu popisu testovacích případů, přístup k testování orientovaný na uživatele a popis pomocí příkladů. Dále nástroje pro tvorbu BDD scénářů umožňují separaci testovacích dat od popisu kroků testovacího příkladu. Využitím této vlastnosti, známé ze skriptů řízených daty, umožňuje zvýšit znovupoužitelnost a usnadnit údržbu.

Mezi nástroje podporující psaní BDD scénářů, které jsou stále podporovány a probíhá jejich aktivní vývoj patří:

- Cucumber

- Cucumber-JVM (implementace Cucumber s podporou jazyků pro Java virtual machine jako je například Java, Scala či Groovy)
- Behat (implementace Cucumber s podporou PHP)
- SpecFlow (implementace Cucumber s podporou C#)
- JBehave
- Robot framework
- Gauge

Existují také nástroje Jasmine, RSpec, NSpec a XBehave.net, které se představují jako BDD řešení. Tyto nástroje ovšem neumožňují striktní oddělení scénáře s klíčovými slovy od vlastní implementace klíčových slov. Díky tomu není umožněno bez znalosti automatizace vytvářet nové scénáře. Touto vlastností porušují jeden z principů BDD.

5.3 Přehled nástrojů pro automatizaci testů pomocí scénářů metodiky BDD

Tato část práce obsahuje přehled vybraných nástrojů pro automatizaci testů pomocí BDD scénářů. Tyto nástroje umožňují automatizovat testování webového rozhraní. Pro návaznost na praktickou část práce, všechna vybraná řešení musí splňovat vlastnosti popsané v následujících bodech:

- Psaní testů formou scénářů formátu Given-When-Then, známou z BDD
- Nástroj existuje ve verzi zdarma bez časového či jiného výrazného omezení
- Možnost spouštění přes příkazovou řádku
- Možnost generování HTML reportů

Tyto kritéria zajišťují vhodnost použití těchto nástrojů pro open source projekty. Dále umožňují jejich snadnou integraci do nástrojů pro průběžnou integraci, které jsou zmíněny v kapitole 4. Vybráni byli čtyři reprezentanti. Konkrétně se jedná o nástroje Cucumber, JBehave, Gauge a Robot framework. Shrnutí vlastností jednotlivých řešení je uvedeno v tabulce níže.

Tabulka 7: *Shrnutí vlastností vybraných nástrojů pro testování formou BDD scénářů⁸ (zdroj: autor)*

	JBehave	Cucumber	Gauge	Robot framework
Platforma nástroje	Java virtual machine	Ruby	Golang	Python
Syntaxe scénářů	Gherkin, JBehave	Gherkin	Založená na markdown	Robot
Jazyk pro implementaci klíčových slov	Java, Groovy, Scala, Ruby	Ruby	C#, Ruby, Java	Python
Nástroj pro úpravu HTML reportů	Freemarker	Ruby	Handlebars	XSLT
Nástroj pro běh scénářů mimo vývojové prostředí	JUnit, Maven, Ant	Vlastní konzolová utilita, Rake	Vlastní konzolová utilita, Maven, Gradle	Vlastní konzolová utilita
Paralelizace běhu scénářů	Integrovaná podpora	Rozšíření parallel_tests	Integrovaná podpora	Rozšíření Pabot
Samostatný komplexní nástroj pro vývoj	Nemá	Nemá	Nemá	RIDE, RED

5.3.1 Cucumber

Prvním z vybraných řešení je Cucumber, open source nástroj pro tvorbu testů ve formátu scénářů převzatých z metodiky BDD. Je napsaný v programovacím jazyce Ruby. Stejný jazyk se používá také pro implementace klíčových slov, z kterých se skládají jednotlivé scénáře (Wynne et al. 2017a).

Tento nástroj je populární vzhledem k množství jeho odvozených variant, které umožňují psaní implementací klíčových slov pomocí různých programovacích jazyků. Mezi tyto odvozené nástroje patří Cucumber-JVM, podporující jazyky pro Java virtual machine jako je například Java, Scala či Groovy. Dále také Behat, umožňující psát skripty v jazyce PHP či SpecFlow, který podporuje psaní skriptů v jazyce C# (Cucumber 2017).

Stejně jako všechny výše uvedené odvozeniny, Cucumber využívá pro psaní scénářů jazyk Gherkin. Tento jazyk byl vytvořen speciálně pro účely tohoto frameworku. Proto je jeho

⁸ Tabulka vytvořena autorem s využitím informací z webových stránek uvedených nástrojů, data platná k 14. 08. 2017

gramatika tvořena s cílem co nejvíce podpořit čitelnost pro byznys uživatele, kteří nemají technické znalosti. Podporují psaní strukturovanou formou Given-When-Then. Čitelnosti napomáhá také překlad konstrukcí Gherkinu do šedesáti různých světových jazyků. Syntaxe je řádkově orientovaná, podobně jako u jazyka Python či YAML. Konce řádků označují konce výrazů a pro definování struktur se používá odsazení.

Scénáře a implementace klíčových slov lze psát za pomoci vývojových prostředí určených pro jazyk Ruby. Prostedí Aptana má podporu pro Cucumber v sobě již integrovanou. Pro další nástroje jako je NetBeans, Eclipse, či RubyMine lze získat podporu prostřednictvím volně dostupných doplňků. Podobně lze pomocí bezplatných rozšíření psát testy v textových editorech, mezi které patří například Sublime Text, VS Code, Notepad++, a TextMate.

Spouštět scénáře lze nejen prostřednictvím doplňků vývojových prostředí. O exekuci scénářů se umí postarat také konzolová utilita, která je standardní součástí balíku Cucumber. Dále je možné spouštění pomocí nástroje pro sestavování Ruby projektů Rake. Paralelní spouštění scénářů je možné zajistit pomocí doplňku `parallel_tests` (Grosser 2017).

Pomocí samostatných utilit pro spouštění scénářů je možné výsledky testů zobrazit v konzoli, nebo také exportovat do souborů. Mezi podporované formáty exportů patří html, json a JUnit. Poslední jmenovaný formát je vhodný zejména pro integraci s nástroji pro průběžnou integraci. JUnit je velmi rozšířený framework pro jednotkové testování, proto je rozšířená i jeho podpora servery pro průběžnou integraci. Cucumber nabízí programové rozhraní pro jazyk Ruby, které podává informace o běhu scénářů. Pomocí něj lze vytvářet exporty libovolných formátů. Na internetu jsou dostupné rozšíření pro Cucumber, která využívají této možnosti a nabízejí další formáty exportů (Cucumber 2016).

5.3.2 JBehave

Dalším nástrojem zařazeným do přehledu je JBehave. Tento nástroj vyvinul Dan North, který je považován za autora metodiky BDD (North 2016). Jak napovídá první písmeno v názvu, tento nástroj je psaný v Javě. Implementace klíčových slov scénářů je proto možné psát Javě, ale také v jazyce Scala, Groovy či Ruby. Tento nástroj tedy představuje alternativu k Cucumber-JVM, odvozeném od Cucumber (JBehave 2017a).

Scénáře je možné psát pomocí syntaxe Gherkin, kterou využívají i další BDD frameworky nebo vlastní JBehave syntaxe. Obě dvě syntaxe jsou si velmi podobné. Společné je pro ně využití Given-When-Then struktury pro scénáře. Hlavním rozdílem je klíčové slovo „After“ používané v JBehave syntaxi. To umožňuje definovat krok, který je proveden na konci každého scénáře. Navíc provedení tohoto kroku lze podmínit úspěšným či neúspěšným skončením scénáře (JBehave 2017b). Jazyk Gherkin pro toto klíčové slovo nemá žádný přímý ekvivalent.

Pro populární IDE určené pro psaní kódu v Javě existují rozšíření, která podporují psaní JBehave testů. Mezi tyto vývojářské nástroje patří Eclipse, NetBeans či IntelliJ IDEA. Scénáře lze také pomocí některých rozšiřitelných textových editorů. Doplnky pro zvýraznění syntaxe a automatické doplňování klíčových slov existují mimo jiné pro editory Sublime Text a TextMate.

Scénáře lze spouštět jako testy velmi rozšířeného frameworku JUnit. Další možností je spouštět scénáře prostřednictvím frameworků pro jednotkové testování TestNG či Spring Test. Alternativně lze pro běh scénářů využít příkazové řádky či je zakomponovat do skriptů nástrojů pro sestavení Ant či Maven (JBehave 2017a). Vzhledem k takto rozmanitým možnostem spouštění, není problém JBehave nasadit do prostředí nástrojů pro průběžnou integraci. Nechybí zde ani zabudovaná podpora pro paralelní spouštění scénářů.

Výsledky běhu scénářů je možné exportovat jako běžný text nebo do souborů formátu xml či html. Pro tvorbu html reportu lze využít výchozí šablonu, či vytvořit vlastní. Individuální šablony lze vytvářet pomocí šablon psaných v jazyce FreeMarker.

5.3.3 Robot framework

Robot framework je open source nástroj napsaný v jazyce Python. Nástroj je původně vyvinutý společností Nokia Networks. Nyní je podporovaný nejen komunitou, ale také konsorciem finských společností s názvem Robot Framework Foundation. Tento nástroj se prezentuje jako nástroj pro tvorbu akceptačních testů (Robot Framework Foundation 2017a). Splňuje však všechna kritéria pro zařazení do skupiny nástrojů pro podporu testování v souladu s BDD.

Pro psaní scénářů je použit speciální jazyk tohoto frameworku, syntaxí připomínající Gherkin známý z Cucumberu. Stejně jako v případě Gherkinu je jazyk tohoto frameworku řádkově orientovaný a k definování struktur používá odsazení. Výhodou Robot frameworku jsou knihovny, které obsahují klíčová slova s již naimplementovanou funkcionalitou. Integrované knihovny poskytují například klíčová slova pro vytváření snímků obrazovky či jednoduchých dialogových oken. Na internetu jsou dostupné další knihovny, které dále rozšiřují dostupnou kolekci klíčových slov. Mezi ně patří například knihovna pro testování webového rozhraní Selenium2Library či Http Library pro testování http požadavků (Robot Framework Foundation 2017c). V některých případech lze proto napsat testovací sadu vytvořenou čistě pomocí snadno čitelné syntaxe jazyka tohoto frameworku. Tím se lze vyhnout technicky náročnější implementaci funkcionality klíčových slov za pomoci programovacího jazyka. Pokud pro testování nestačí klíčová slova z dostupných knihoven, lze implementovat funkcionalitu nových klíčových slov za pomoci jazyka Python.

Pro vytváření nových testovacích scénářů a exekuci testů lze použít jedno ze specializovaných IDE pro Robot Framework. Mezi ně patří RIDE či RED, které jsou obě dostupné zdarma. Pro editaci scénářů lze použít některé z dostupných rozšíření pro populární IDE

a textové editory. Mezi podporované nástroje patří například Eclipse, IntelliJ, TextMate, Sublime Text, Notepad++ a VS Code (Robot Framework Foundation 2017d). Kromě spouštění scénářů prostřednictvím specializovaných IDE, možné je přímé využití dodávané konzolové utility. Pro paralelní běh je nutné použít doplněk Pabot.

Konzolová utilita Robot frameworku umí vypisovat informace o běhu nejen do konzole, ale také do souborů. Pro tvorbu vlastních exportů libovolného formátu lze využít možnosti exportu do xml a následné jeho transformace. Pro získání uživatelsky přívětivého reportu je nabízena možnost vygenerování html dokumentu. Posledním dostupným formátem výstupu je XUnit. Ten je formátem jednoho z hojně využívaných frameworků pro jednotkové testování. Hodí se proto zejména pro začlenění testů do prostředí nástrojů pro průběžnou integraci, které v mnoha případech umí s tímto formátem pracovat.

Robot framework byl použit pro implementaci testů pro projekt EasyMiner, které byly vytvořeny jako součást této práce.

5.3.4 Gauge

Posledním z vybraných nástrojů pro psaní testů stylem známým z BDD je Gauge. Tento open source nástroj vznikl ve firmě ThoughtWorks, stejně jako populární Selenium. Projekt je stejnou firmou stále sponzorován (ThoughtWorks 2016a).

Gauge byl napsán v jazyce Golang. Na rozdíl od většiny BDD nástrojů Gauge nevyužívá syntaxe založené na jazyce Gherkin. Místo toho se scénáře píší jazykem, který je založen na tvz. markdown syntaxi. Podobně jako u Gherkinu je oddělovač klíčových slov konec řádky a vytvořené scénáře se velmi podobají obyčejnému textu. Syntaxe přímo nepočítá s psaním scénářů se strukturou Given-When-Then. Testy tímto stylem známým z BDD však vytvářet lze. Ukázka scénáře napsaného pomocí Gauge je uvedena níže.

Výpis 4: *Ukázka scénáře napsaného pomocí BDD frameworku Gauge (zdroj: ThoughtWorks 2016b)*

```

1 Customer Sign-up
2 -----
3 tags: sign-up, customer
4
5 * Sign up a new customer with name <name> email <email> and <password>
6 * Check if the user <name> is logged in
7 * See items available for purchase."
```

Implementace klíčových slov, které zajišťují logiku automatizace lze psát za pomoci programovacích jazyků C#, Ruby nebo Javy. Trojici oficiálně podporovaných jazyků doplňuje Javascript, Python a Golang. Podporu proto tyto jazyky vytvořila a udržuje komunita.

Zvýraznění syntaxe, automatické doplňování a další podpůrné funkce pro psaní scénářů a implementací klíčových slov lze pomocí rozšíření doplnit do trojice podporovaných IDE. Těmi jsou IntelliJ, Eclipse a Visual Studio (ThoughtWorks 2017). V době psaní práce

chyběla podpora pro zvýrazňování syntaxe jazyka scénářů pro rozšiřitelné textové editory, jako je například Sublime Text, VS Code či Notepad++.

Spouštět scénáře lze přímo z prostředí podporovaných vývojových nástrojů. Pro externí spouštění scénářů a jejich případné začlenění do nástrojů pro průběžnou integraci lze použít konzolovou utilitu. Dále je možné začlenit spouštění Gauge do nástrojů pro sestavování projektů Maven a Gradle. Pro zrychlení běhu testů lze využít integrované podpory paralelizace. Podporovanými formáty reportů s výsledky testů jsou xml a html. V případě potřeby individuálního reportu lze změnit výchozí šablonu pro generování html. Tato šablona je vytvořena pomocí šablonovacího nástroje Handlebars.

5.4 Shrnutí přehledu

Většina nástrojů pro automatizované testování webového rozhraní využívá knihovnu Selenium WebDriver, která se stará o propojení automatizovaných nástrojů s webovými prohlížeči. Tyto nástroje lze rozdělit do dvou skupin. Do první patří nástroje pro zachycení a přehrávání aktivity uživatele. Ty jsou jednoduché na používání a nevyžadují technické znalosti. Avšak mají řadu nevýhod. Mezi ně lze zařadit vysokou citlivost na změny ovládacích prvků aplikace, zaznamenávání pouze statických hodnot a nemožnost ošetřit případné chybové stavy aplikace při testování. Dalším problémem je omezená možnost znovupoužití jednotlivých částí testovacích případů, které se v testovací sadě opakují. Vzhledem k těmto vlastnostem je problémem náročná údržba takto vytvořených testů. Proto se hodí pouze na testování ustálené a neměnné funkcionality.

Druhou skupinu tvoří nástroje pro automatizované testování pomocí skriptů. Ty umožňují psát testy pomocí programovacích jazyků jako je například C#, Java, Ruby, Python, či PHP. Jejich vytváření a editace je možná pomocí doplňků do běžně používaných IDE či rozšiřitelných textových editorů. Tyto nástroje odstraňují nevýhody nástrojů pro zachycení a přehrávání aktivity uživatele. Při správném použití jsou méně náchylné na změny v aplikaci, umožňují opakované použití částí testů a ošetřovat výjimky, které nastanou v průběhu testování. Problémem pro ně není ani testování dynamických hodnot. Za jejich výhody se platí nutností velkých technických znalostí, které jsou vyžadovány pro jejich vývoj. Omezit tuto nevýhodu má za cíl podмноžina těchto nástrojů, zvaná nástroje pro testování klíčovými slovy. Pomocí nich lze vytvořit v přirozeném jazyce množinu klíčových slov popisující možné akce, které mohou být v aplikaci prováděny. Pomocí nich pak mohou vytvářet automatizované testovací scénáře také členové týmu, kteří nejsou odborníky v oblasti automatizace. Specialisté na automatizaci pak implementují logiku jednotlivých klíčových slov. Mezi tyto nástroje patří například Cucumber, JBehave, Gauge a Robot framework, které byly popsány v této kapitole.

V každé kategorii existuje řada profesionálních nástrojů, které jsou zdarma. Proto není problém zavést automatizované testování také do malých vývojových týmů, či open source

projektů. Neexistuje jeden nástroj, který by byl vhodný za každé situace. Vždy záleží na konkrétním systému, který je určen k otestování a lidských zdrojích, které jsou dostupné. Mezi hlavní faktory pro výběr lze zařadit komplexitu systému, četnost změn systému, odbornost pracovníků či dostupnost potřebných specialistů na pracovním trhu. Nabídka nástrojů je však velmi pestrá a pro každý projekt lze najít řešení, které pro něj bude vyhovující.

6 Možnosti integrace Selenium testů uživatelského rozhraní do Travis CI

Volba řešení pro testování webového uživatelského rozhraní v rámci průběžné integrace se skládá z výběru komponent, které toto testování umožňují. Různé komponenty mají různé vlastnosti a jejich využití má vždy své klady i zápory. Správná kombinace jednotlivých částí umožňuje vytěžit z průběžné integrace maximum a usnadnit tak vývoj aplikace.

Tato část práce se zabývá různými možnostmi volby jednotlivých komponent řešení pro testování webového uživatelského rozhraní v rámci průběžné integrace. Možnosti výběru byly zúženy na nástroje založené na knihovně Selenium WebDriver, která je de facto standardem pro vývoj testů webového rozhraní.

Pro lepší návaznost na praktickou část práce, je tato kapitola zaměřena na nástroj pro průběžnou integraci Travis CI. Avšak většinu uvedených informací v této kapitole lze využít také pro další nástroje pro integraci postavené na platformě Linux. Pro příklad lze uvést Circle CI nebo Semaphore CI, které jsou uvedeny v přehledu nástrojů pro integraci v předchozí části práce. U specifických nastavení pro Travis CI je explicitně uvedeno, že se týkají čistě tohoto nástroje.

Běh testů v prostředí cloudového integračního serveru má některá svá specifika, která ho odlišují od spouštění testů lokálně na počítači vývojáře. Jde zejména o absenci displeje, který by umožňoval klasickým způsobem zobrazovat stránky webového prohlížeče. Dále pro každý běh je vytvořeno nové integrační prostředí, které je po skončení nedostupné. Travis CI navíc oproti některým konkurenčním řešením neumožňuje ladění běhu testů připojením se do integračního prostředí. Řešení proto musí o svém běhu podávat detailní informace, které mohou být použity pro snadné hledání příčiny chyb.

6.1 Volba platformy pro běh integrace

Prvním rozhodnutím, které je třeba uskutečnit při implementaci je volba operačního systému, na kterém je postaveno prostředí pro integraci. Travis CI nabízí volbu mezi platformami Linux a Os X. Podpora platformy Windows chybí. Pro testování na platformě Windows lze použít alternativu v podobě nástroje AppVeyor. Ten je popsán v přehledu nástrojů pro průběžnou integraci v předchozí části práce.

Volba platformy silně závisí na projektu, který je určen k integraci. Pokud je projekt určený pouze pro jednu platformu, volba je jasná a výběr odpovídá podporované platformě. Pro multiplatformní projekty je vhodné volit prostředí, které odpovídá platformě, ve které probíhá vývoj. Velká část nastavení prostředí pak představuje kopii lokálního nastavení, které používají vývojáři. Nastavení prostředí je díky tomu pro vývojáře jasnější a jeho

údržba snadnější. Dále je tím možno snížit výskyt situací, kdy na lokálním stroji programátora vše běží korektně, kdežto integrace projektu na serveru hlásí chybu, způsobenou rozdílností těchto prostředí.

V případě testování pomocí nástrojů využívajících knihovnu Selenium WebDriver nemá smysl testování na různých operačních systémech za použití stejného prohlížeče. Knihovna WebDriver je totiž platformě nezávislá a její chování je stejné na všech platformách.

6.2 Volba typu Linuxového prostředí

Tato volba je specifická pro integrační prostředí Travis CI. Travis CI totiž pro integraci na platformě Linux nabízí dva druhy prostředí. Jedno tvoří samostatný virtuální stroj, druhé je založeno na virtualizaci prostředí za využití nástroje Docker.

Samostatný virtuální stroj je hostovaný za pomoci Google Compute Engine. Výhodou je možnost využití příkazu `sudo`. Nevýhodou je delší čas nutný pro inicializaci tohoto prostředí. Travis CI dokumentace uvádí odhadovanou dobu nutnou pro start jako rozmezí 20 až 50 sekund (Travis CI 2017a). Na druhé straně stojí prostředí založené na Docker kontejneru. Podle dokumentace startuje během jedné až šesti sekund. Tato výhoda je vykoupěna nemožností využít příkazu `sudo`. Všechny obrazy Docker kontejnerů, které jsou použity pro běh integrace jsou volně dostupné ke stažení na serveru Docker Hub. Odtud je lze stáhnout a využít pro ladění integrace lokálně na počítači vývojáře.

Zejména pro kratší testy, které trvají méně než čtvrt hodiny a nepotřebují využívat příkazu `sudo` se hodí zvolit prostředí zaležené na kontejnerech. Při volbě dedikovaného virtuálního stroje by samotný start zabral znatelnou část času nutného pro integraci.

6.3 Výběr nástroje pro definici a exekuci testů

Pro začlenění do systému pro průběžnou integraci je třeba, aby nástroj podporoval exekuci testů prostřednictvím příkazové řádky. Naštěstí téměř neexistuje nástroj pro testování uživatelského rozhraní, který by tuto možnost nenabízel. Ideální volbou je takový nástroj, který dokáže vygenerovat html či pdf report, který objasní případné problémy při neúspěšném běhu testů. Přehledem softwarových nástrojů pro automatizované testování webového rozhraní s využitím knihovny Selenium se zabývá samostatná kapitola 5.

6.4 Volba webového prohlížeče

Architektura Selenium WebDriver je postavena tak, že jeden vytvořený skript umožňuje jeho spouštění na vícero prohlížečích. Pro testy je třeba použít prohlížeč, pro který existuje Selenium ovladač, zajišťující komunikaci mezi prohlížečem a knihovnou WebDriver. Mezi podporované prohlížeče patří Firefox, Chrome, Internet Explorer, Opera a Safari. Testování několika prohlížečů lze poté nastavit pomocí jednoho parametru, který určí ovladač, resp. prohlížeč použitý pro běh testů (Selenium project 2012). Díky tomu možnost jednoduchého spouštění testů na několika prohlížečích podporují téměř všechny nástroje založené na knihovně Selenium WebDriver.

Volba konkrétního prohlížeče pro běh testů závisí zejména na tom, jaký prohlížeč je podporován testovanou aplikací. V případě, že je podpora zaručována pro více prohlížečů, lze využít výše popsané možnosti testování na několika prohlížečích. Nutno mít na paměti, že testování aplikace na vícero prohlížečích prodlužuje čas, který celá integrace běží. Více prohlížečů znamená instalaci vícero závislostí, což je proces, který zabere nějaký čas. Další prodloužení zajistí samotný běh testů. Čas pro běh testů se dá částečně optimalizovat. Například testování na více prohlížečích lze provádět pouze na podmnožině testů, která představuje kritickou funkcionalitu. Případně lze využít paralelního spouštění testů. Paralelní běh na Travis CI však vyžaduje placený plán předplatného, protože bezplatný plán umožňuje pouze běh jedné úlohy současně.

6.5 Virtualizace displeje

Jedním ze specifíků dostupných cloudových řešení pro průběžnou integraci je absence displeje. Displej je však potřebný pro zobrazování stránek v běžném webovém prohlížeči. Pro umožnění běhu webových prohlížečů v prostředí pro průběžnou integraci se proto využívá nástrojů pro virtualizaci displeje. Mezi populární patří například nástroj pro virtualizaci displeje pro Linux Xvfb, neboli X virtual framebuffer (Haeffner 2017).

Jako alternativa k využití běžných prohlížečů a virtuálního displeje existují speciální prohlížeče, nazývané headless. Headless prohlížeče jsou webové prohlížeče bez grafického rozhraní. Díky tomu slouží k poskytování webového obsahu dalším programům, nikoliv uživatelům (Girdwood 2009). Mezi nejpopulárnější patří HtmlUnit a PhantomJS. Jedním z jejich možných využití je automatizované testování webových stránek. Využitím headless prohlížeče se lze vyhnout nastavení virtuálního displeje v prostředích, kde displej není k dispozici. Avšak mezi běžnými prohlížeči a těmito nástroji existují drobné rozdíly, které mohou vést k odlišnému chování aplikace v těchto prohlížečích. Tím může být snížena výpovědní hodnota takto automatizovaných testů a důvěra v ně. Přímá podpora pro využití těchto prohlížečů chybí v některých nástrojích, které jsou postaveny nad Selenium

WebDriverem. Populární prohlížeč Chrome podporuje start v headless módu od verze 59, díky čemuž je možné testování bez použití virtuálního displeje (Bildeman 2017).

Pro testování webového rozhraní je jednoznačně vhodnější volbou využití reálně používaných prohlížečů s případným využitím virtuálního displeje oproti testování s využitím specializovaných prohlížečů pro automatizaci.

6.6 Způsob instalace závislostí

Knihovny, spustitelné utility a další závislosti, potřebné pro běh testů lze instalovat několika způsoby. Ty lze rozdělit do dvou skupin. Jedna využívá instalace závislostí přímo do integračního prostředí, druhá využívá nástroje Docker.

Přímá instalace do prostředí pro integraci umožňuje některé závislosti sdílet s dalšími nástroji, které jsou použity během průběžné integrace. Ke sdílení knihoven dochází zejména pokud nástroj pro testování je vytvořen pomocí stejného programovacího rámce jako testovaný projekt. Jako příklad lze uvést nástroj pro testování Cucumber-JVM a webový projekt psaný v Javě. Závislosti pro Javu jsou pak instalovány do prostředí jednou a využity jak při sestavení projektu, tak i pro testování.

Alternativně lze využít nástroje Docker. Pomocí něj lze vytvořit kontejner, do kterého budou nainstalovány všechny potřebné závislosti pro běh testů. Testy pak lze spouštět přímo v prostředí uživatelem vytvořeného kontejneru. Tento postup má několik výhod. První je snadná přenositelnost. Vytvořený obraz kontejneru lze použít v libovolném prostředí, které podporuje nástroj Docker. Díky tomu lze pouštět testy lokálně na počítači vývojáře v naprosto identickém prostředí, které je použito v rámci průběžné integrace. Navíc obraz lze vystavit na Docker Hubu. Docker Hub slouží jako úložiště automaticky sestavovaných obrazů Docker kontejnerů a pro veřejně dostupné obrazy je zdarma. Výhodou použití Docker Hubu je ten, že obrazy obsahují již nainstalované závislosti a po stažení jsou ihned připraveny pro exekuci testů. Přitom běh v prostředí Docker kontejnerů vyžaduje zanedbatelné režijní náklady, nezpomaluje tedy běh testů. Přímá instalace závislostí do integračního prostředí nebo sestavování kontejneru při každém běhu integrace znamená nutnost nejen stažení jednotlivých závislostí, ale také jejich instalaci. Tato instalace, zejména u velkého množství závislostí může být časově náročná.

Nevýhodou využití Docker kontejneru je možná duplikace instalace závislostí. V integračním prostředí totiž již mohou být dostupné některé knihovny potřebné pro běh testů. Avšak vzhledem k izolaci Docker kontejneru je třeba tyto knihovny nainstalovat také do kontejneru.

Výhody využití Docker kontejneru pro běh testů postavených nad knihovnou Selenium WebDriver převyšují jeho nevýhody. Proto se vyplatí investovat čas navíc, který je potřeba k vytvoření tohoto řešení.

6.7 Možnosti zasílání notifikací

O stavu integrace lze zasílat informace do různých komunikačních kanálů. Pro řešení pro průběžnou integraci platí, že vždy podporují alespoň zasílání e-mailových zpráv s výsledkem integrace. Pro Travis CI je tento informační kanál nastaven jako výchozí. Dále lze využít zasílání notifikací do služeb Slack, HipChat, Flowdock či do IRC. Další možností, jak sledovat stav integrace je tzv. badge, štítek se stavem integrace. Tento štítek je možný vložit jako odkaz na libovolnou webovou stránku, nebo do souboru readme v repozitáři na GitHubu. Ten se automaticky aktualizuje v závislosti na stavu integrace (Travis 2017a).

Mimo vyjmenované komunikační kanály, které jsou přímo podporovány nástroji v Travis CI, lze zasílat notifikace také do dalších komunikačních služeb, které podporují zasílání notifikací prostřednictvím tzv. Webhooks. Webhook je mechanismus pro integraci služeb, kdy jeden systém posílá další externí službě informace v případě výskytu definované události. Informace jsou zasílány ve formě HTTP POST callbacku (Lindsay 2007). Travis CI umožňuje nastavit zasílání těchto notifikací při startu, zrušení, úspěchu, či chybě integrace.

6.8 Zajištění dostupnosti vygenerovaných artefaktů

Travis CI, podobně jako další cloudové řešení pro průběžnou integraci nenabízí vestavěné úložiště souborů. Reporty z běhu testů, statické analýzy kvality kódu a další artefakty vytvořené během průběžné integrace je tedy třeba přesunout na jiné místo, kde je možné se k nim zpětně dostat. K tomu může sloužit externí cloudové úložiště, jako je například Amazon S3 nebo Google Cloud Storage.

Alternativně lze artefakty publikovat na libovolný web. Například pomocí účtu na GitHubu lze zcela zdarma získat možnost publikování veřejných webových stránek pomocí funkce GitHub Pages. Ta dokáže publikovat statické webové stránky za pomoci uložených souborů v repozitáři na GitHubu. Vystavení výsledků a dalších artefaktů na webu může být pro uživatele přívětivější. Prezentace reportů z běhu průběžné integrace na webu publikovaném prostřednictvím GitHub Pages je součástí řešení, které bylo vytvořeno v rámci praktické části této práce.

6.9 Shrnutí možností

Existuje několik způsobů, kterými lze zajistit spouštění Selenium testů webového uživatelského rozhraní v cloudových nástrojích pro průběžnou integraci. Největším rozdílem oproti lokálnímu spuštění testů je absence displeje a nutnost inicializace prostředí pro každý běh. Volba konkrétního řešení závisí zejména na vybraném nástroji pro definici a exekuci testů a testované aplikaci.

Ačkoliv některé volby jsou závislé na dané situaci, některé postupy jsou obecně vhodnější. Jedná se o spouštění testů na prohlížečích, které jsou reálně využívány uživateli aplikace, oproti využití specializovaných prohlížečů určených čistě pro automatizaci. Dále je výhodné vytvoření dedikovaného Docker kontejneru, určeného čistě pro spouštění testů. Samostatný kontejner umožňuje snadné lokální spouštění testů v prostředí identickém tomu, který je použit v rámci průběžné integrace.

7 Řešení pro testování webového rozhraní během průběžné integrace pro open source projekty

V této kapitole je popsán návrh řešení pro testování webového rozhraní během průběžné integrace. Navržené řešení se soustředí na podporu webových open source projektů, které využívají Docker kontejnery pro nasazení systému. Podobně jako první polovina práce se tento návrh zaměřuje na projekty uložené na největším veřejném úložišti repositářů, kterým je GitHub.

Všechna kritéria na jejichž základ bylo zvolené řešení navrženo, lze shrnout do následujících bodů:

- Umožnit testování webového rozhraní během průběžné integrace
- Podpora pro open source projekty zveřejněné na GitHubu
- Podpora projektů využívajících nástroje Docker pro nasazení systému
- Využití výhradně komponent, které jsou pro open source zdarma
- Využití nástroje pro průběžnou integraci, který je pro open source zdarma bez časového omezení či omezení počtu spuštění
- Využití komponent, u kterých probíhá stále vývoj (vyhnout se využití projektů s ukončenou podporou)
- Důraz na snadnou údržbu a nasazení

Cílem návrhu je poskytnout řešení, které je univerzálně použitelné pro široké spektrum open source projektů. Aby bylo umožněno využití daného řešení také v rámci velmi malých projektů, bylo stanoveno kritérium pro využívání pouze bezplatných komponent.

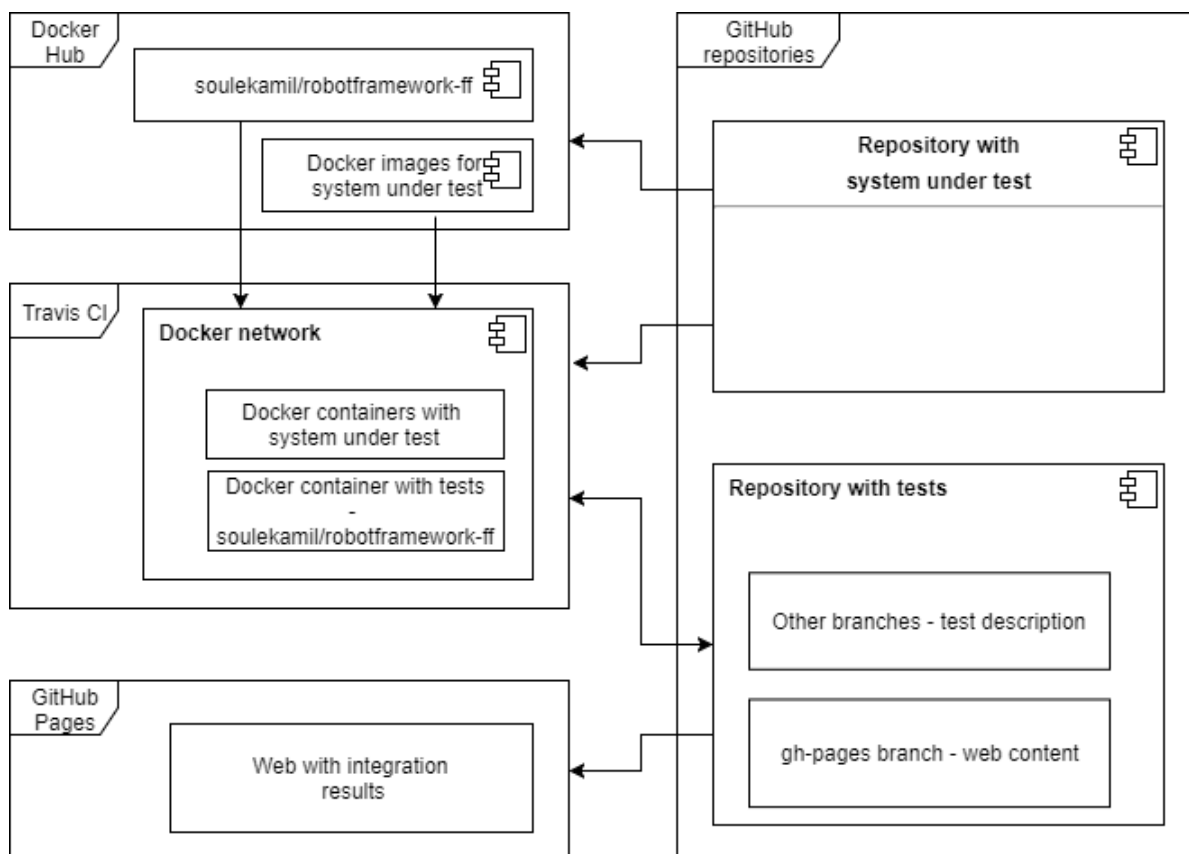
Navržené řešení je následně použito pro implementaci a nasazení testů webového rozhraní open source projektu EasyMiner, který je vyvíjen pod záštitou katedry informačního a znalostního inženýrství na VŠE. Popisem této implementace se zabývá kapitola 8.

7.1 Softwarové komponenty řešení a jejich komunikace

Navržené řešení využívá kombinace cloudového serveru pro průběžnou integraci a Docker obrazu umožňujícího spouštění Selenium testů psaných pomocí nástroje Robot framework.

Vybraným cloudovým serverem pro integraci je Travis CI. Ten testuje systém, který je uložen v repositáři na GitHubu. Podmínkou je, aby součástí testovaného projektu byly

konfigurační soubory pro Docker, které umožňují nasazení dané aplikace. Případně mohou být Docker obrazy vystaveny na veřejném repozitáři Docker Hub. Dále řešení využívá webových stránek hostovaných prostřednictvím služby GitHub Pages. Tyto stránky slouží pro prezentaci reportů a doplňujících informací z proběhlých integrací. Hlavní komponenty řešení a jejich vzájemné propojení je znázorněno na níže uvedeném schématu.



Obrázek 3: Hlavní komponenty navrženého řešení průběžné integrace a jejich vzájemné propojení (zdroj: autor)

Komunikace jednotlivých komponent řešení při procesu integrace je následující:

- GitHub po změnách ve sledovaných větvích repozitáře projektu s testy nebo aplikací zašle informaci o změně integračnímu serveru Travis CI
- Travis CI stáhne repozitář s projektem a repozitář s testy z GitHubu
- Travis CI asadí na základě konfigurace pro Docker testovaný systém do prostředí integračního serveru
- Travis CI spustí testy s využitím kontejneru *soulekamil/robotframework-ff* a deskripce testů stažené z repozitáře na GitHubu
- Travis CI po skončení testování odešle e-maily se stavem integrace, popř. zašle notifikace do vybraných komunikačních služeb
- Travis CI stáhne větev repozitáře s testy, která obsahuje web s výsledky. Do tohoto repozitáře přidá aktuální výstupy a následně ho nahraje s aktuálními informacemi zpět do repozitáře na GitHubu

- GitHub repositář na základě změny obsahu v repositáři s obsahem webu aktualizuje s využitím služby GitHub Pages web s informacemi o proběhlých integracích

Konkrétní nastavení jednotlivých příkazů k vykonání uvedených kroků integrace záleží na konkrétním testovaném projektu. Jako příklad možného nastavení celého řešení slouží implementace testů uživatelského rozhraní pro open source aplikaci EasyMiner, která je součástí této práce.

Středobodem řešení je Docker kontejner, který slouží k spouštění testů. Ten je vytvářen na základě obrazu *soulekamil/robotframework-ff*, který je veřejně dostupný z repositáře Docker Hub. Řešení pomocí Docker kontejneru umožňuje snadné spouštění testů jak na lokálním počítači vývojáře, tak na integračním serveru. Zároveň je zajištěno, že na všech počítačích testy běží ve stejném prostředí.

Pro definici testů byl vybrán nástroj umožňující psát testy formou Given-When-Then scénářů, dle metodiky behaviour driven development. Testy psané touto formou se snadno udržují, vzhledem k pevné a čitelné struktuře. Scénáře jsou psány v přirozeném jazyce a tvoří tak zároveň čitelnou dokumentaci chování systému. Pro ni lze použít označení „živá dokumentace“, protože lze popisované testy spustit a ověřit tak, že chování aplikace skutečně odpovídá dané specifikaci. Více se psaní testů dle metodiky behaviour driven development a jejím výhodám věnuje kapitola 3.5.

Z dostupných nástrojů pro psaní testů formou scénářů dle metodiky BDD byl vybrán nástroj Robot framework. Ten vyniká zejména přehlednými reporty, které generuje již ve výchozím nastavení. Dále při chybě testu framework automaticky vytvoří snímek obrazovky a ten následně vloží do generovaného reportu. Tento report následně usnadňuje porozumění chybovým stavům a urychluje jejich opravu. Další výhodou je, že nástroj obsahuje již naimplementovaná klíčová slova pro knihovnu Selenium. Tudiž celou testovací sadu, včetně jednotlivých Selenium instrukcí, lze vytvořit pomocí klíčových slov, které se podobají přirozenému jazyku. Toto řešení je tedy vhodné také pro technicky méně zdatné uživatele. Více o Robot frameworku a dostupných bezplatných alternativách je uvedeno v samostatné kapitole 5.3.

Při tvorbě byl kladen důraz na snadnost údržby a jednoduchost samotného nasazení. Proto je návrh řešení postaven nad cloudovým serverem pro průběžnou integraci. Jeho výhodou je rychlost zprovoznění, které je otázkou několika minut. Naopak počáteční nasazení a zprovoznění lokálně spravovaných protějšků je otázkou několika hodin až dnů, jelikož vyžadují nastavení infrastruktury, prostředí a instalaci. Údržba cloudových řešení spočívá pouze v údržbě nastavení a samotných testů. Odpadají starosti o hardware a software infrastrukturu na které integrační server běží. O ni se stará poskytovatel služby.

Stanoveným požadavkům na řešení vyhovují všechny servery pro průběžnou integraci, zmíněné v přehledu v předchozí části práce. Travis CI byl volen vzhledem k jednoduchosti nastavení a přehledné obsáhlé dokumentaci. Dále také proto, že nabízí elegantní způsob pro ukládání utajených hodnot, potřebných k dešifrování šifrovaných souborů. Tato funkce

se hodí pro zajištění přístupu k webovým stránkám, na které jsou nahrávány reporty a další artefakty vzniklé během integrace.

Mezi výhody zvoleného řešení patří také skutečnost, že pro přihlášení do všech použitých webových služeb se používá účet na GitHubu. Ten se používá nejen pro přístup k repositářům s testovaným projektem a testy, ale také k přihlášení k integračnímu serveru a nastavení pro publikování webových stránek s výsledky testů. To výrazně přispívá nejen k pohodlí jednotlivých vývojářů při práci, ale také usnadňuje správu uživatelských oprávnění pro správce projektu.

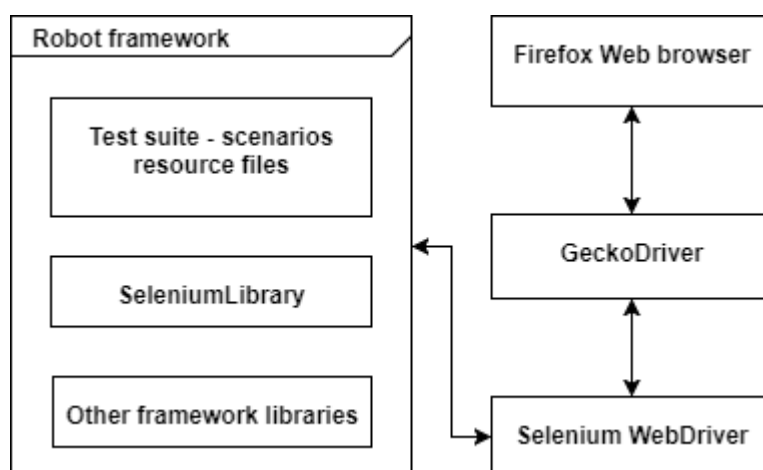
7.1.1 Docker kontejner zajišťující spouštění testů

Jednou z hlavních komponent řešení je Docker image, který obstarává samotné spouštění automatizovaných testovacích skriptů napsaných pomocí nástroje Robot framework. V centrálním repositáři obrazů Docker Hub již existuje několik vytvořených obrazů, které také poskytují možnost spouštět Selenium testy psané pomocí Robot frameworku. Jako příklad lze uvést obrazy *bogdangi/robot-docker* (Girman 2017) nebo *danielwhatmuff/robot-docker* (Whatmuff 2017). V době psaní práce však žádný dostupný obraz nebyl pro navržené řešení vhodný. Každý obraz měl nejméně jednu z níže uvedených nevýhod. Proto byl vytvořen pro navržené řešení speciální image s názvem *souleka-mil/robotframework-ff*. Tento image je veřejný a volně dostupný na Docker Hubu. Díky tomu je možné ho stáhnout a využít pomocí standardních příkazů pro práci s Docker image, jako je *docker pull* či *docker run*. Pro jeho spuštění je nutné mít pouze nainstalovaný nástroj Docker verze 1.12 a vyšší.

První nevýhodou stávajících řešení je použití klasických, plnohodnotných Linuxových distribucí jako základu pro daný obraz. Tyto distribuce jsou určeny pro využití koncovými uživateli a obsahují proto spoustu nadstavbových knihoven, utilit a nástrojů. Tyto nástroje však nejsou nezbytné pro běh daných automatizovaných testů a pouze zbytečně zvyšují velikost obrazu. Existuje ovšem vhodnější volba, a to odlehčené, minimalistické verze Linuxu. Ty obsahují pouze nezbytné jádro a velmi malou množinu nástrojů. Cílem těchto distribucí je co nejvíce minimalizovat jejich velikost a sloužit jako základ pro vytváření úzce zaměřených nástrojů. Příkladem minimalistické distribuce je Alpine Linux. Velikost zkomprimovaného obrazu Alpine Linux, v nejaktuálnější verzi 3.6 jsou pouze 2 MB, na disku bez komprimace pak zabere 3.97 MB. Naopak obraz distribuce Debian jessie zkomprimovaný vyžaduje 53 MB a rozbalený image z disku ukrojí 123 MB místa. Velikost obrazů je důležitým parametrem zejména při využití těchto obrazů v prostředí cloudového integračního serveru. Tyto integrační servery totiž při každém běhu používají nové virtuální prostředí. Díky tomu je nutné pro každou integraci stáhnout daný Docker obraz znovu. Velikost obrazu se tak podílí na čase, který je nutný pro jeho stažení do prostředí a tím pádem ovlivňuje celkový čas nutný pro celý běh integrace. Proto je vhodné velikost co nejvíce optimalizovat.

Druhou nevýhodou současných řešení je nspecifikování verzí u tří hlavních komponent Selenium testů – Selenium WebDriver, webového prohlížeče Firefox a ovladače GeckoDriver pro vybraný prohlížeč. Bohužel nové verze Selenium WebDriver nejsou vždy zpětně kompatibilní se staršími verzemi ovladačů pro prohlížeč. Stejně tak ovladače pro prohlížeč podporují vždy jen určitou skupinu verzí prohlížečů. Balíčkovací systémy ovšem při chybějící specifikaci verze stahují nejaktuálnější dostupnou. Což z výše uvedených důvodů může vést k vzájemné nekompatibilitě komponent a nefunkčnosti celého řešení.

Součástí Docker image soulekamil/robotframework-ff jsou všechny závislosti nutné k spouštění Selenium testů psaných pomocí RobotFrameworku v prohlížeči Firefox. Jeho součástí je nástroj Robot framework, který se stará o spouštění skriptů. Dále obraz obsahuje komponenty pro automatizované ovládání webového prohlížeče, kterými jsou Selenium WebDriver, ovladač pro prohlížeč Gecko driver a kompatibilní webový prohlížeč Firefox. Protože Docker image nemá displej, na kterém by bylo možné zobrazit webový prohlížeč, je nutná jeho virtualizace. Tuto virtualizaci zajišťuje nástroj Xvfb. Hlavní součásti obrazu a jejich komunikace je znázorněna na obrázku níže. Ostatní instalované knihovny jsou pouze nezbytné závislosti pro běh uvedených hlavních komponent.



Obrázek 4: Hlavní komponenty Docker obrazu pro exekuci Robot framework testů webového rozhraní (zdroj: autor)

Tento obraz je připraven k okamžitému použití. Nutné je pouze nastavit sdílení dvou adresářů kontejneru s prostředím, ve kterém je spouštěn. Prvním z nich je adresář s názvem „Tests“, nacházející se v kořenovém adresáři. Do něj je třeba nakopírovat testovací skripty. Druhým adresářem je adresář „TestResults“, který se také nachází v kořenovém adresáři. Ten slouží pro ukládání reportů a případných snímků obrazovky z jednotlivých běhů testů.

Například pokud jsou testy vytvořené pomocí Robot frameworku v adresáři „Testy“, a výsledky testů mají být uloženy do adresáře „Vysledky“, spouštění testů lze provést následujícím příkazem, uvedeným v ukázce kódu níže.

Výpis 5: Spouštění testů s využitím Docker obrazu soulekamil/robotframework-ff (zdroj: autor)

```

1 docker run -v $(pwd)/Testy:/Tests/ -v $(pwd)/Vysledky:/TestResults soulekamil/robotframework-ff
  
```

Alternativně lze nastavení tohoto obrazu zakomponovat do nastavení pro Docker compose a následně využívat příkazů této utility. Tento postup byl využit při implementaci testů pro systém EasyMiner a je popsán v kapitole 8.4.

Při startu kontejneru vytvořeného pomocí daného obrazu dojde k automatickému spuštění všech testů. Průběh testů lze sledovat pomocí výpisu do konzole. Po skončení běhu kontejner signalizuje výsledek návratovou hodnotou, která odpovídá počtu neúspěšně provedených testů. V případě úspěchu tedy vrací hodnotu 0, což je návratová hodnota standardně používaná pro označení úspěchu. Tato návratová hodnota je využita zejména při běhu v prostředí integračního serveru, který podle ní určí, zda byl běh testů úspěšný či nikoliv.

Po skončení testů je vygenerován report o průběhu testovacího běhu. Ten je následně uložen do podadresáře složky TestResults. Nově vytvořený podadresář má v názvu datum spuštění testu. Díky tomu je při opakovaném spuštění testů na jednom PC uchovávána historie jednotlivých běhů.

Obraz podporuje testování pouze proti prohlížeči Firefox. Výhodou využití tohoto prohlížeče je malá velikost všech knihoven potřebných pro zajištění chodu Selenium testů. Ta je menší než například u populárnějšího prohlížeče Chrome. Zároveň je Firefox plnohodnotným prohlížečem, na rozdíl například od Phantom.js, určeného čistě pro automatizaci. Testy tak probíhají v prostředí, které se přibližuje tomu, ve kterém se testovaný systém bude skutečně používat. Testy tak mají větší vypovídající hodnotu a lze v ně mít důvěru větší než při využití specializovaných prohlížečů určených čistě pro automatizaci. V případě nutnosti testování na více prohlížečích, je možné vytvořit potomka tohoto obrazu (tzv. child image). Ten pak rozšířit o další potřebné prohlížeče a knihovny nutné pro správný chod. Takto vytvořeného potomka lze pak využívat stejným způsobem jako původní obraz.

Výhodou využití Docker kontejneru je možnost spouštění testů lokálně na počítači vývojáře v naprosto stejném prostředí jako je na integračním serveru. Navíc pro lokální spouštění není třeba instalovat do počítače všechny potřebné závislosti. Všechny jsou již nainstalované v připraveném Docker obraze. Odpadají také problémy se samotnou instalací závislostí na lokálním počítači, při které může dojít ke konfliktu s již nainstalovanými knihovnami. Docker kontejnery totiž běží izolovaně mezi sebou i prostředím, ve kterém jsou spouštěny.

7.1.2 Docker kontejnery pro spuštění testované aplikace

Navržené řešení nevyžaduje, aby testovaná aplikace byla nasazena a dostupná na externím serveru. S využitím nástroje Docker je možné aplikaci nasadit přímo do prostředí Travis CI. Díky tomu odpadá nutnost správy dalšího prostředí a procesu, který by zajistil nasazení nejnovější verze aplikace a synchronizaci se serverem pro průběžnou integraci.

Zvolený postup integrace vyžaduje, aby součástí testovaného projektu byly konfigurační soubory pro Docker, které umožňují nasazení dané aplikace. Ty jsou pak použity pro nasa-

zení a spuštění aplikace v prostředí Travis CI. Konkrétní nastavení pro Docker se liší v závislosti na daném testovaném projektu. Pro všechny projekty je však nutné, aby celá aplikace byla nasazená do Docker kontejnerů a nespolehlala se na externí služby.

Primárně navržené řešení cílí na projekty, které již využívají Docker kontejnery. U těchto projektů odpadá nutnost vytvářet konfiguraci pro Docker a lze přistoupit přímo k dalším krokům implementace. Řešení je ovšem vhodné také pro projekty, které prozatím nejsou konfigurovány pro nasazení s využitím Dockeru. Pro stávající projekty není vytvoření konfigurace pro Docker prací na dlouhé měsíce. Konfigurační soubor Dockerfile je z větší části složen z instrukcí pro příkazovou řádku daného prostředí. Pouze řádově jednotky příkazů tvoří specifické příkazy nástroje Docker. Navíc dokumentace nástroje Docker je rozsáhlá a dobře srozumitelná. Vzhledem k rozšíření tohoto nástroje existuje na internetu řada článků, které se zabývají řešením různých specifických nastavení. Navíc využití Docker kontejnerů nekončí u nasazení systému na serveru pro průběžnou integraci. Pomocí kontejnerů lze výrazně usnadnit nasazení systému do libovolného prostředí. Společně se zveřejněním obrazů na centrálním repositáři Docker Hub, lze zajistit stažení a nasazení aplikace pomocí jednoho krátkého skriptu.

7.1.3 Nastavení serveru pro průběžnou integraci

Navržené řešení využívá Travis CI jako server pro průběžnou integraci. Tento cloudový integrační server je dostupný pro open source projekty zdarma. Vyniká zejména jednoduchým nastavením a obsáhlou dokumentací. Více o Travis CI a jeho alternativách je uvedeno v kapitole 4.

Výsledky integrací open source projektů jsou veřejně přístupné. Pro administraci serveru je nutné přihlášení pomocí uživatelského účtu na GitHubu. Možnosti, které přihlášený uživatel dostane, jsou určeny podle oprávnění, která má nastavena k integrovanému repositáři na GitHubu. Pro zapnutí automatických spouštění integrace je třeba mít nejvyšší práva k integrovanému repositáři, tedy práva *admin*. Pro změnu nastavení serveru a manuální spouštění integrací postačují práva o úroveň nižší, tedy práva *push*. Díky využití uživatelských účtů z GitHubu je usnadněna správa spojená s administrací oprávnění.

Nastavení kroků integrace se konfiguruje pomocí textového yaml souboru. Ten musí být uložen v repositáři společně se sestavovaným projektem. Webové rozhraní nabízí pouze manuální spuštění integrace, nastavení utajených hodnot a omezení akcí vyvolávajících spuštění integrace. Ve výchozím nastavení se integrace spouští pro každou změnu v repositáři (*push* akce) a navržené změny (*pull request* akce), pro všechny vývojové větve. Nastavení umožňuje omezit spuštění integrací pouze na větve repositáře, které obsahují konfigurační soubor. Dále je možné vypnout spuštění integrací pro navrhované změny či omezit počet možných paralelních spuštění. Tato omezení jsou vhodná zejména proto, aby přílišné množství integrací nezabíralo cenné místo ve frontě pro spuštění. Zároveň lze pomocí tohoto nastavení snížit zahlcení vývojářů nedůležitými notifikacemi o průběhu

integrace vedlejších větví repositáře. Doporučeným postupem je spouštět integrace pouze pro hlavní vývojovou větev. Z dalších vývojových větví následně vybrat pouze ty, které jsou určeny pro uchovávání stabilní verze projektu.

Jediné nastavení, které je dáno navrženým řešením je nastavení utajených hodnot, které jsou potřebné pro aktualizaci webu s výsledky a reporty z průběhu testování. Jejich nastavení a použití je popsáno v kapitole 7.2.1. Další nastavení integrace závisí na konkrétním projektu a potřebách vývojového týmu. Součástí integrací mohou být mimo jiné další testy (např. jednotkové testy), či nasazení projektu do vybraného externího prostředí.

7.2 Informování vývojářů o průběhu integrace

Důležitou funkcí integračního serveru je poskytování zpětné vazby o aktuálním stavu projektu vývojovému týmu a dalším zainteresovaným osobám. Ve výchozím nastavení Travis CI odesílá e-maily s výsledkem integrace všem vývojářům, kteří mají práva pro přidávání změn do repositáře s projektem (práva *push* a *admin*). V případě potřeb lze rozšířit výchozí nastavení adresátů o další zainteresované osoby. Notifikace pomocí e-mailových zpráv by však neměla chybět u žádného projektu. Travis CI dokáže zasílat notifikace o stavu také do dalších komunikačních služeb. Příkladem jsou služby Slack, HipChat, Flowdock či IRC. Využití zasílání notifikací do jiných informačních kanálů závisí na konkrétních využívaných službách při práci na projektu. Při volbě nastavení notifikací je však nutné nezahlcovat vývojáře příliš velkým množstvím zpráv. Doporučením daného řešení je omezit zasílání e-mailů všem vývojářům pouze na stav integrace vybraných hlavních větví repositáře. Pojmem hlavní větve repositáře jsou myšleny všechny větve repositáře, které představují větve stabilních verzí projektu a hlavní vývojovou větev (kterou je většinou výchozí větev *master* nástroje Git).

Při výběru doplňkových komunikačních kanálů pro zasílání notifikací je vhodné omezit se pouze na jeden. Zároveň je nutné notifikace do komunikačních služeb nastavit tak, aby tyto notifikace nespouštěly zasílání e-mailových zpráv. V takovém případě by mohlo dojít k zasílání duplicitních emailů se zprávami o stavu integrace, což není žádoucí.

Navržené řešení pro průběžnou integraci nabízí jako další zdroj informací o průběhu integrací webové stránky. Tyto veřejně dostupné webové stránky jsou publikované pomocí služby GitHub Pages. Tato služba umožňuje automatické publikování obsahu repositáře na web. Existuje několik způsobů, jak nastavit tuto službu. Pro navržené řešení je zvolen postup vytvoření vývojové větve v repositáři s testy s názvem *gh-pages*. GitHub dokáže rozpoznat tuto jmennou konvenci a po vytvoření takto pojmenované větve automaticky publikuje obsah v této větvi na web. Adresa stránek je vždy zobrazena na webu s repositářem v položce *Settings* v podsekcí *GitHub Pages*.

Účelem samostatného webu je zpřístupnit artefakty vytvořené v průběhu integrace. Tím je adresován problém Travis CI, který se týká absence dostupnosti souborů po skončení integrace. Tento problém se však týká naprosté většiny cloudových serverů pro integraci poskytovaných zdarma. Součástí tohoto řešení je návod, jak zajistit automatickou aktualizaci těchto stránek přímo z integračního serveru. Řešení nespecifikuje konkrétní obsah ani podobu těchto stránek. Ta závisí vždy na daném projektu a všech krocích integrace. Jedinou nezbytnou součástí těchto stránek by měl být html report, který je generovaný Robot frameworkem. Tento report nabízí detailní popis průběhu testování, který zároveň může sloužit jako dokumentace aktuálního chování aplikace. Díky němu lze také urychlit odhalení příčin možných chyb integrace.

Příkladem dalších volitelných součástí stránek mohou být logy z jednotlivých částí aplikace, které poskytují další cenné informace o běhu integrace. Mezi další příklady patří výsledky jednotkového testování, analýzy pokrytí kódu testy, či nejrůznější statické analýzy kódu.

7.2.1 Návod pro publikování výsledků integrace na webové stránky

Součástí navrženého řešení je webová stránka s výsledky. Stránky jsou publikovány pomocí služby GitHub Pages. Tato bezplatná služba dokáže automaticky vystavit obsah repozitáře na GitHubu jako webovou stránku. Pro aktualizaci takto vytvořené webové stránky z prostředí Travis CI nestačí pouze nastavení přes konfigurační soubor či webové rozhraní serveru. Je nutné provést několik dalších kroků, nejen v nastavení Travis CI, ale také přímo v nastavení repozitáře na GitHubu. Prvním nastavením, které je potřeba provést, je vytvoření SSH klíče, který bude použit jako nástroj pro přihlášení k repozitáři s webem z Travis CI. Ten lze vytvořit pomocí bash příkazu. Při vytvoření klíče se vygenerují dva soubory, jeden představující privátní část klíče a druhý tu veřejnou. Veřejnou část vytvořeného klíče je třeba přidat do repozitáře s webem na GitHubu, společně s povolením zápisu pro tento klíč. Díky tomu bude možné měnit obsah repozitáře s pomocí přihlášení s využitím privátní části vytvořeného klíče. Následně vytvořený privátní klíč je třeba zašifrovat a přidat do repozitáře s testy. Aby mohl být tento zašifrovaný klíč použit na serveru pro průběžnou integraci, je třeba do Travis CI nahrát hodnoty, které slouží k dešifrování klíče. Tyto hodnoty jsou do Travis CI nahrány jako utajené, aby nebylo možné tyto hodnoty využít k dešifrování privátního klíče mimo prostředí Travis CI. Pro aktualizaci stránek poté stačí pouze stažení repozitáře s obsahem stránek, provést úpravy a následně pomocí SSH klíče tyto změny uložit zpět do repozitáře s webem. Po nahrání změn do repozitáře na GitHubu se služba GitHub Pages automaticky postará o zveřejnění nové verze webu.

Celý výše uvedený mechanismus pro nastavení SSH klíče a Travis CI lze provést s využitím bash skriptu, který je uveden v ukázce kódu níže.

Výpis 6: Skript pro vytvoření zašifrovaného SSH klíče pro využití v Travis CI (zdroj: autor)

```
8 gem install travis
9 ssh-keygen -t rsa -b 4096 -C "travis-test-results" -f testresults_deploy_key -N ''
10 travis login
11 travis encrypt-file testresults_deploy_key
```

Pro spuštění skriptu pro vytvoření zašifrovaného SSH klíče je zapotřebí:

- Ruby verze 1.9.3 a vyšší
- Bash shell – pro uživatele Windows stačí emulátor, například MinGW, který je součástí nástroje Git pro Windows (Git for Windows) či dostupný samostatně

Pro nastavení přihlášení do repozitáře s webem prostřednictvím zašifrovaného SSH klíče z prostředí Travis CI je třeba vykonat následující kroky:

1. Spuštění výše uvedeného skriptu v lokální kopii adresáře s projektem, pro který probíhají integrace na Travis CI.
2. Po vyzvání se přihlásit uživatelským jménem a heslem GitHub účtu, který má práva úrovně *push* nebo *admin* pro sestavovaný projekt.
3. Po přihlášení je klíč zašifrován a hodnoty pro dešifrování odeslány na server. Výstup konzole obsahuje příkaz, který je třeba si uložit a následně použít v konfiguračním souboru `travis.yml`. Tento příkaz začíná vždy klíčovými slovy *openssl aes-256-cbc*.
4. Zašifrovaný klíč, který je nyní v souboru `testresults_deploy_key.enc` je třeba uložit do adresáře s projektem na server GitHub.
5. Pomocí nastavení repozitáře na GitHubu je potřeba nahrát veřejnou část klíče. Ta je uložena v souboru `testresults_deploy_key.pub`. Nastavení SSH klíčů se skrývá pod položkou „Settings“, v podsekcí „Deploy keys“. Po stisku tlačítka „Add deploy key“ se objeví formulář pro nahrání klíče. Následně je nutné obsah souboru `testresults_deploy_key.pub` zkopírovat do pole „Key“ a zaškrtnout pole „Allow write access“ (pro povolení zápisu). Nakonec stačí potvrdit odeslání formuláře.
6. Posledním krokem je nastavení konfiguračního souboru `travis.yml`. Ten je potřeba nastavit tak, aby provedl stažení a potřebné úpravy repozitáře s webem. Konkrétní sekvence instrukcí závisí na konkrétním projektu. Provedené změny se nahrají zpět do repozitáře na GitHubu s využitím šifrovaného klíče, který je dešifrován příkazem z kroku 3.

Uvedený postup byl využit pro testování projektu EasyMiner, jehož implementace je součástí této práce. U něj je možné vidět příklad stránek a skriptu využitého k jejich aktualizaci.

7.3 Shrnutí navrženého řešení

V této části práce bylo popsáno navržené řešení pro automatizované testování grafického webového rozhraní v rámci průběžné integrace. Toto řešení je navrženo pro použití při vývoji open source projektů. Navržené řešení není určeno pro jeden konkrétní druh projektů, ale je obecně využitelné pro široké spektrum webových open source projektů. Vzhledem k obecnému použití řešení není jeho součástí detailní popis všech nastavení zvolených komponent. To závisí na konkrétním projektu, pro který má být implementováno.

Navržené řešení využívá cloudový server pro průběžnou integraci Travis CI. Testy webového rozhraní, psané pomocí nástroje Robot framework jsou spouštěny pomocí Docker kontejneru. Pro tento Docker kontejner byl vytvořen speciální obraz [souleka-mil/robotframework-ff](https://hub.docker.com/r/souleka-mil/robotframework-ff/), který je zveřejněný na centrálním úložišti obrazů Docker Hub. Ten lze využít pro spouštění testů nejen na serveru Travis CI, ale také lokálně na počítači vývojáře. Nástroj Docker je také použit pro nasazení a spuštění testovaného systému. Další významnou komponentou řešení jsou webové stránky, které jsou zveřejněny pomocí služby GitHub Pages. Tyto stránky slouží k ukládání a prezentaci reportů a dalších souborů vytvořených během integrace projektu.

Mezi výhody tohoto řešení patří nulové náklady na provoz, rychlé nasazení a snadná údržba. Nulové náklady na provoz jsou zajištěny díky využití komponent, které jsou dostupné zcela zdarma pro open source projekty. Rychlého nasazení je docíleno díky využití cloudového integračního serveru, který je dostupný jako služba okamžitě. Rychlému nasazení dopomáhá také zveřejněný obraz pro spouštění testů, který je již připraven k použití. Snadná údržba je zajištěna za pomoci několika faktorů. Díky využití cloudového serveru odpadá nutnost starat se o údržbu a aktualizaci infrastruktury, na rozdíl od využití lokálně spravovaných řešení. Správa uživatelských účtů a oprávnění je jednoduchá, protože pro repositáře s testy, projektem, obsahem webových stránek a Travis CI jsou používány stejné GitHub uživatelské účty. Dále díky využití nástroje pro testování pomocí klíčových slov je možné do vývoje testů zapojit také technicky méně zdatné uživatele. Zároveň lze testy psát tak, aby mohly být současně použity jako dokumentace chování aplikace. Součástí návrhu je také návod popisující, jak lze provést nahrání souborů vytvořených během integrace z Travis CI na zveřejněný web.

Další výhodou řešení je samostatná stránka s reporty a dalšími informacemi o proběhlých integracích. Díky ní je možné získat detailní informace o chybách při integraci a sjednat tak jejich rychlou nápravu.

Dalším významným benefitem je využití nástroje Docker pro spouštění aplikace a testů. Pomocí Dockeru je možné testy spouštět na integračním serveru a různých počítačích vždy ve stejném prostředí. Díky tomu je eliminován výskyt chyb spojených s rozdílnými prostředími, ve kterých testy běží.

8 Implementace testů webového rozhraní během průběžné integrace pro projekt EasyMiner

V předchozí části práce je popsáno řešení pro testování webového rozhraní v rámci průběžné integrace, aplikovatelné pro široké spektrum různých open source projektů. Tato kapitola se zabývá implementací tohoto řešení pro konkrétní projekt. Implementovány a nasazeny jsou všechny části řešení. Součástí je tedy vytvoření sady testů pokrývajících základní funkcionalitu systému, nastavení serveru pro průběžnou integraci a vytvoření webové stránky pro prezentaci výsledků a reportů z proběhlých integrací.

8.1 Popis testovaného systému

Testovaným objektem je webové uživatelské rozhraní open source projektu EasyMiner. Tento projekt je vyvíjen na katedře informačního a znalostního inženýrství na VŠE. Tento projekt je vystaven veřejně na GitHubu.

EasyMiner je stále aktivně vyvíjen a probíhá rozšiřování jeho funkcionality. V době psaní této práce byla poslední stabilní verze 2.4. Konkrétně byly testy rozhraní vyvíjeny proti následujícím verzím jednotlivých částí aplikace:

- [EasyMiner-EasyMinerCenter](#) – tag v2.4, commit 5587cfc, 12. 07. 2017
- [EasyMiner-Backend](#) – tag v2.4, commit 06f5f5e, 13. 06. 2017
- [EasyMiner-Scorer](#) – tag v2.4, commit 4253bee, 30. 12. 2016

8.1.1 Funkcionalita

EasyMiner (EasyMiner/R) je webová aplikace pro hledání asociačních pravidel postavená nad balíčkem *arules* pro jazyk R. Nabízí vizuální webové rozhraní dostupné přes prohlížeč a rozhraní dostupné přes REST API. Systém také obsahuje implementaci algoritmu CBA (Classification Based on Associations). Ten lze využít pro vytváření klasifikačních modelů z asociačních pravidel, či redukci množiny nalezených pravidel pomocí prořezávání. Výhodou systému EasyMiner je interaktivní rozhraní umožňující snadnou definici vzorů hledaných pravidel (KIZI 2017a).

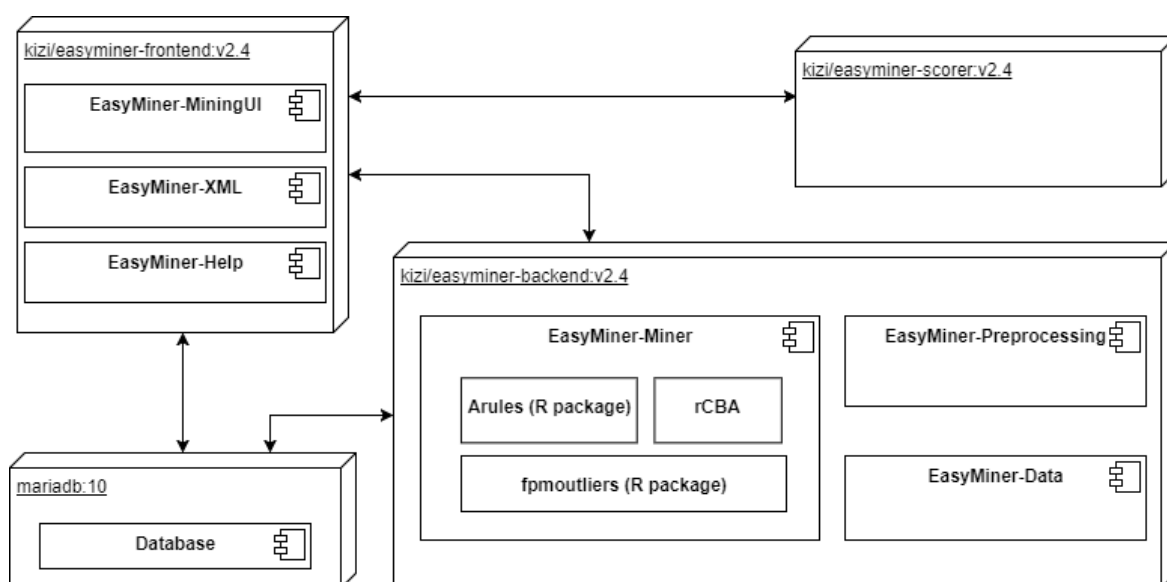
Přes grafické rozhraní aplikace, nejsou dostupné všechny funkce aplikace, které poskytují její služby. Mezi hlavní funkce, které nabízí aktuální stabilní verze systému 2.4, dostupné

přes grafické rozhraní, patří správa uživatelů, nahrání datasetu ve formátu csv, předpracování dat, nalezení a ukládání asociačních pravidel.

8.1.2 Architektura

Architektura EasyMineru je založena na znovupoužitelných webových službách. Díky tomu je celý systém modulární. Rozhraní jednotlivých služeb je zcela zdokumentováno. Kromě výchozího backendu založeného na balíčku *arules* nabízí také alternativu v podobě backendu postaveném nad Apache Spark/Hadoop. Systém byl však testován s výchozím backendem.

Architektura systému EasyMiner je znázorněna na níže uvedeném diagramu. V diagramu jsou znázorněny pouze hlavní komponenty řešení. Diagram také popisuje rozdělení jednotlivých součástí do Docker obrazů, které jsou používány pro nasazení systému.



Obrázek 5: Architektura systému EasyMiner⁹ (zdroj: autor)

Níže uvedený zjednodušený popis stěžejních komponent systému je převzat z oficiálního webu projektu (KIZI 2017b).

EasyMinerCenter slouží jako integrační platforma pro všechny ostatní služby. Pro ně poskytuje jednotné webové grafické rozhraní, přes které lze přistupovat k jejich funkcionalitě. Jeho součástí je také REST API, které umožňuje integraci s dalšími projekty, či využití v data mining skriptech. Mimo to se tato komponenta stará o správu uživatelských účtů, ukládání nalezených asociačních pravidel a poskytuje autentifikační službu pro ostatní

⁹ Schéma architektury je vytvořené na základě informací získaných z repositářů jednotlivých částí systému (KIZI 2017c, KIZI 2017d, KIZI 2017e)

služby. Tato komponenta je dostupná ke stažení jako Docker obraz kizi/easyminer-frontend.

EasyMiner-Data je webová služba pro správu datových zdrojů. Tato služba podporuje nahrávání dat v csv a rdf formátu. Nahráná data jsou ukládána do databáze MySQL (MariaDB) nebo Hive.

EasyMiner-Preprocessing je služba zajišťující vytváření datasetů pro data mining. Data získává z datových zdrojů vytvořených pomocí služby EasyMiner-Data. Z těchto dat následně vytváří atributy pomocí jednoho z dostupných algoritmů pro předzpracování. Mezi poskytované algoritmy pro předzpracování patří: each value-one bin, intervals enumeration, nominal enumeration, equidistant intervals a equisized intervals.

EasyMiner-Miner je služba, která zapouzdřuje algoritmy pro data mining. Mezi podporované metody data miningu patří hledání asociačních pravidel (apriori, FP-Growth), prořezávání a klasifikace (CBA) a detekce anomálií (pomocí algoritmů z balíčku fpmoutlier pro R).

EasyMiner-Scorer je služba pro testování klasifikačních modelů založená na asociačních pravidlech.

8.2 Cíle řešení a vhodnost použití navrženého řešení

Před vypracováním této práce pro aplikaci EasyMiner neexistovaly žádné automatizované testy grafického uživatelského rozhraní. Existující testy, které jsou součástí repozitáře [EasyMiner-Evaluation](#), ty ovšem testují pouze část REST API rozhraní. Řešení vytvořené v této práci tak nenahrazuje ani nenavazuje na žádné stávající řešení.

Automatizované testy pro grafické rozhraní projektu EasyMiner byly vytvořeny s cílem vytvořit snadno udržovatelné řešení, které zajistí:

- Automatizované testování základní funkcionality dostupné přes grafické webové rozhraní aplikace
- Zakomponování testovací sady do serveru pro průběžnou integraci Travis CI
- Informování vývojářů o výsledcích integrace

Zvoleným řešením pro implementaci bylo řešení navržené pro obecné použití v rámci open source projektů, popsané v předchozí části práce. Toto řešení umožňuje naplnění všech stanovených cílů.

Pro testování webového rozhraní existuje řada možností, kterými lze zajistit. Přehled dostupných možností je uveden v teoretické části práce. Byl zvolen nástroj z kategorie nástrojů umožňujících psaní scénářů dle metodiky BDD. Využití metodiky BDD umožňuje psát

snadno čitelné a udržitelné testy. Popis výhod scénářů dle metodiky BDD lze nalézt v samostatné kapitole 3.5. Z dostupných nástrojů byl vybrán Robot framework. Ten má oproti ostatním nástrojům několik výhod. První je jednoduchost jeho použití. Umožňuje psát testy kompletně s využitím syntaxe, podobné přirozenému jazyku, včetně volání knihovných funkcí Selenium WebDriver. Dalším benefitem Robot frameworku jsou reporty o běhu testů, které jsou již v základním nastavení přehledné a detailní. Jejich součástí je také automatické vkládání snímků obrazovky při chybě.

Testy lze vyvíjet nejen pomocí doplňků pro zvýrazňování syntaxe, dostupné pro běžně používané IDE a modulární textové editory. Pro tento framework existují také samostatné, specializované nástroje, jako je RIDE či RED. Detailnějším popisem tohoto nástroje se zabývá kapitola 5.3.3.

Práce s testovacím nástrojem Robot framework je vyučována v rámci předmětů na katedře informačních technologií VŠE. Konkrétně se jedná o předměty 4IT446 - Řízení kvality softwaru a 4IT478 - Automatizované funkční testování softwaru. Předmět 4IT446 je navíc povinný pro studenty vedlejší specializace 4KS – Řízení kvality softwaru. Díky tomu se zvyšuje šance, že se mezi studenty VŠE najde vývojář, který by se mohl podílet na budoucím rozvoji a údržbě testovací sady.

Pro zakomponování testů do Travis CI je využito nástroje Docker. Ten je použitý jednak pro spuštění systému EasyMiner, tak i pro spuštění testů. Využití Docker kontejnerů umožňuje spouštět testy lokálně ve stejném prostředí, jaké je použito pro běh testů na serveru pro průběžnou integraci. Díky tomu je eliminován výskyt chyb spojených s rozdílnými prostředími, ve kterých testy běží.

Zvolené řešení umožňuje notifikovat všechny vývojáře e-mailem v případě chyby. Navíc jsou dodatečné informace z běhu integrace nahrávány na webovou stránku, která je veřejně přístupná. Díky této zpětné vazbě je možné na případnou chybu reagovat rychle a zajistit tak včasnou nápravu.

8.3 Nastavení integračního serveru Travis CI

Spouštění integrací na serveru Travis CI je nastaveno pro repositáře EasyMiner-Center, který obsahuje frontend aplikace EasyMiner. Dále je nastavený běh integrací pro repositář EasyMiner-Tests, který obsahuje implementované testy webového rozhraní. Pro oba tyto repositáře je použito shodné nastavení.

Integrace jsou spouštěny při push operacích do větví repositáře, které obsahují konfigurační soubor `travis.yml`. Pro ostatní větve a operace pull request je automatické spouštění vypnuto. Konfigurační soubory jsou přidány do větve repositáře pro vyvíjení nové funkcionality master a větve pro údržbu poslední stabilní verze 2.4. Díky tomu integrace běží jen na hlavních vývojových větvích. Tento postup je zvolen proto, aby běh integrací na vedlej-

ších vývojových větvích nezabíral cenné místo ve frontě sestavení. To by mohlo vést ke zpoždění spuštění testování vývojových větví, které vyžadují největší pozornost. V případě potřeby je možné spustit integraci manuálně, prostřednictvím webového rozhraní Travis CI.

Travis CI nabízí dva druhy virtuálního prostředí, na kterém běží integrace. Jedno je založeno na virtuálních strojích a druhé na Docker kontejnerech. První nabízí možnost spuštění příkazu `sudo`, druhý nikoliv. Absence možnosti použít příkaz `sudo` je však kompenzována rychlejším startem integrace. Pro optimalizaci běhu integrace je nastaveno použití virtuálního prostředí založeného na Docker kontejnerech.

Integrace se skládá ze tří částí. První je příprava prostředí, která je provedena v rámci sekce *before_install*. V této části dojde k stažení testů z repositáře EasyMiner-Tests. Testy jsou staženy z větve repositáře, která odpovídá větví v repositáři testovaného systému. Následně je vytvořen Docker obraz frontendové části aplikace přímo z repositáře EasyMiner-Center na GitHubu. Vytváření obrazu přímo z repositáře na GitHubu, místo stažení připraveného obrazu ze serveru Docker Hub, má své opodstatnění. Jakmile dojde ke změně repositáře na GitHubu, ve stejný okamžik dojde ke spuštění integrace na serveru Travis CI a vytváření obrazu na Docker Hubu. Proto během integrace posledního přírůstku na Travis CI nemusí být na serveru Docker Hub připravený obraz, který již obsahuje poslední změny a je nutné stahovat frontend přímo z repositáře na GitHubu.

Po připravení prostředí se v rámci *script* sekce provede spuštění testovaného systému a vlastní testování. Informace o průběhu testů jsou zobrazeny v konzoli. Tento výstup lze v reálném čase sledovat ve webovém rozhraní Travis CI.

Poslední sekce *after_script* se stará o aktualizaci informací o běhu na webové stránky s výsledky integrace. V ní je stažena poslední verze repositáře s výsledky integrace. Pomocí bash skriptu je zajištěn export informací o běhu, reportu vygenerovaného Robot frameworkem a logů frontendové části testované aplikace. Prostřednictvím bash skriptu je provedeno také vygenerování aktualizované podoby webové stránky. Po provedení aktualizace jsou změny uloženy zpět do repositáře s výsledky. S využitím uloženého zašifrovaného SSH klíče jsou následně změny webové stránky z Travis CI odeslány zpět do repositáře na GitHubu. Po detekci změny GitHub zajistí publikování aktualizované verze stránky s výsledky a reporty, která je dostupná na <https://kizi.github.io/EasyMiner-Tests/>.

8.4 Spouštění aplikace EasyMiner a testů pomocí Docker compose

Pro spouštění aplikace EasyMiner a testů je využíván nástroj Docker compose. Tento nástroj slouží pro definování a práci s aplikacemi, které jsou složeny z více Docker kontejnerů. Jeho nastavení je v konfiguračním souboru `docker-compose.yml`, který je uložen

v kořenovém adresáři repositáře s testy. Tento konfigurační soubor definuje všechny služby, které jsou pro start aplikace a testů potřeba. V konfiguraci pro projekt s testy pro aplikaci EasyMiner jsou všechny služby definovány jako Docker kontejnery, které jsou vytvářeny na základě definovaných Docker obrazů. Díky využití Docker compose je zajištěno, že jsou jednotlivé služby inicializovány ve správném pořadí.

Pro určení správného pořadí spouštění služeb konfigurace Docker compose využívá instrukce *healthcheck* v kombinaci s nastavením závislostí mezi jednotlivými službami. Sekce konfigurace *healthcheck* umožňuje určit příkaz, jehož návratová hodnota rozhoduje o připravenosti služby. Tento příkaz je následně volán přímo v kontejneru služby. Pokud příkaz vrátí hodnotu 0, je služba považována za připravenou a její status (condition) je nastaven na hodnotu *healthy*. Pokud naopak příkaz vrátí hodnotu 1, je služba považována za nepřipravenou a její status (condition) je nastaven na hodnotu *unhealthy*. Pro každý *healthcheck* se kromě testovacího příkazu nastavuje také časový interval mezi testováním a maximální počet opakování neúspěšného testu. Status služby je následně využit pro definování závislostí jednotlivých služeb. Závislosti jsou definovány v *depends_on* sekci konfigurace služby. Součástí definice závislosti je název služby a její požadovaný stav. Použití těchto mechanismů je vidět na ukázce kódu níže, jejíž součástí je definice služby pro frontend aplikace EasyMiner.

Výpis 7: *Nastavení EasyMiner frontendu v konfiguračním souboru docker-compose.yml (zdroj: autor)*

```

1  easyminer-frontend:
2    container_name: easyminer-frontend
3    image: kizi/easyminer-frontend:v2.4
4    ports:
5      - "8894:80"
6    healthcheck:
7      test: "curl -f http://easyminer-frontend/easyminercenter"
8      interval: 30s
9      retries: 5
10   depends_on:
11     easyminer-mysql:
12       condition: service_healthy
13     easyminer-scorer:
14       condition: service_healthy

```

8.5 Informování vývojářů o průběhu integrace

Pro zasílání notifikací o průběhu integrací je použita notifikace e-mailem. Zpráva s informací o běhu je zaslána všem uživatelům, kteří mají pro daný repositář nastavená práva push či admin. Zprávy jsou zasílány kdykoliv dojde k chybě při integraci. Dále je zasílána informace o prvním úspěchu pro chybě. Vývojáři tedy nejsou zatěžováni zprávami, pokud probíhá integrace opakovaně v pořádku.

Spouštění integrací je omezeno pouze na push operace do větví, které obsahují konfigurační soubor. Mezi ně patří větev pro vyvíjení nové funkcionality master a větev pro údržbu stabilní verze 2.4. Toto omezení platí pro oba repositáře, pro které integrace běží čili EasyMiner-Center s frontendem aplikace a EasyMiner-Tests, který obsahuje testy. Díky tomu jsou vývojáři informováni e-mailem pouze v případě, kdy se nachází chyba v hlavních vývojových větvích. E-mail se stavem integrace je zasílán neohledně na to, v jakém kroku Travis CI ukončí svůj běh. Notifikace do dalších komunikačních kanálů nastaveny nejsou. Tento postup pro zasílání notifikací byl konzultován a schválen vývojovým týmem projektu EasyMiner.

Současně po každém běhu integrace dojde k nahrání výsledků a reportů o běhu na webovou stránku <https://kizi.github.io/EasyMiner-Tests/>. Tato stránka byla vytvořena s využitím bootstrap šablony Narrow Jumbotron, která byla upravena pro daný účel. Design je responzivní a minimalistický. Kromě krátkého popisu a odkazů na stránky projektu EasyMiner a GitHub repositář s projektem obsahuje pouze tabulku s výsledky.

Každý řádek tabulky s výsledky obsahuje informaci o repositáři, větvi, identifikátoru a popisu přírůstku, pro který byla integrace spuštěna. Další sloupce obsahují samotný výsledek integrace a odkaz na html report vygenerovaný Robot frameworkem. V posledním sloupci je odkaz pro stažení zip archivu, který obsahuje složku, do které jsou ukládány všechny logy z webové komponenty projektu. Podobu této tabulky lze vidět na obrázku níže.

Repository	Branch	Commit	Commit message	Test result	Test report	Web log
EasyMiner-Tests	master	8023a	Documentation update.		Show report	Download log
EasyMiner-Tests	v2.4	715d3	Documentation update.		Show report	Download log

Obrázek 6: Tabulka s reporty a informacemi o běhu integrací na vytvořeném webu (zdroj: autor)

Detailní html report a logy z aplikace jsou hlavní přidanou hodnotou vytvořeného webu, oproti výpisu z konzole, který je dostupný na serveru Travis CI. S jejich pomocí lze lépe pochopit k jaké chybě došlo. V některých případech lze s jejich využitím také určit příčinu chyby a sjednat její nápravu bez nutnosti spouštět testy na počítači vývojáře.

Webová stránka je nastavena tak, aby uchovávala informace o posledních 50 integracích. V případě dosažení tohoto limitu je smazána informace o nejstarším provedeném běhu.

8.6 Spouštění testů lokálně na počítači vývojáře

Testovací sadu je možné spustit také lokálně na počítači vývojáře. Díky použití Docker obrazů pro nasazení testovaného systému a spouštění testů, nevyžaduje lokální exekuce

testů zdoluhavou instalaci závislostí. Pro lokální testování je třeba mít nainstalovány pouze tyto tři bezplatné nástroje:

- Docker verze 1.12 a vyšší
- Docker compose verze a vyšší
- Bash shell – pro uživatele Windows stačí emulátor, například MinGW, který je součástí nástroje Git pro windows (Git for Windows) či dostupný samostatně

Pro usnadnění spouštění byly vytvořeny bash skripty, které obsahují sekvence příkazů pro spuštění testů. Mimo samotné spuštění testů obsahuje vytvořený repositář EasyMiner-Tests další užitečné skripty, které usnadňují vývoj testů či odhalení příčiny chyb.

Hlavním skriptem je skript *run-tests.sh*. Start tohoto skriptu zajistí stažení všech potřebných Docker obrazů. Po stažení jsou vytvořeny a spuštěny všechny kontejnery potřebné pro běh aplikace EasyMiner. Zvolené nastavení vynucuje vytváření nových kontejnerů, nikoliv využívání již vytvořených. Tento postup zaručuje stejné výchozí podmínky pro každý testovací běh. Jakmile je systém EasyMiner připraven, jsou v dalším samostatném kontejneru spuštěny testy.

Průběh testů je možné sledovat v reálném čase pomocí výpisu do konzole. Pro každý běh testů je vytvořena ve složce TestResults nová podsložka s aktuálním časem, do které jsou po ukončení běhu uloženy reporty. Díky tomu je zachována historie o průběhu jednotlivých testování. Po ukončení běhu skriptu *run-tests.sh* aplikace EasyMiner stále běží. Díky tomu je možné prohlédnout stav aplikace po skončení testování. Po skončení testování a případného procházení aplikace lze použít skript *cleanup-tests.sh*, který zajistí zastavení a odstranění všech použitých Docker kontejnerů.

Aplikaci EasyMiner lze spustit také samostatně, bez nutnosti spuštění testů. K jejímu spuštění slouží skript *start-easyminer.sh*. Web aplikace je dostupný na adrese <http://<docker-machine>:8894/easyminercenter>, kde <docker-machine> je adresou použitého Docker stroje. Pro zastavení běhu aplikace slouží skript *stop-easyminer.sh*.

Pomocí skriptu *export-web-logs.sh* lze exportovat aktuální obsah logů spuštěné frontendové části aplikace. Logy jsou vyexportovány do adresáře WebLogs, kde je pro ně vytvořena podsložka s časem exportu, podobně jako u exportu reportů z testů.

Konfigurace Docker kontejnerů pro spuštění testovaného systému a testů je zapsána v souboru *docker-compose.yml*. Tento soubor využívá nástroj Docker compose. Výhodou jeho použití je zajištění inicializace kontejnerů ve správném pořadí. Výchozí nastavení stahuje obrazy pro vytváření kontejnerů z centrálního repositáře Docker Hub. Pokud chce vývojář použít testy k otestování lokálních, dosud nepublikovaných změn, stačí pouze změnit příslušné zdroje obrazů v konfiguračním souboru pro Docker compose. Pro uživatele neseznámené s prací s Docker compose je určen komentář v konfiguračním souboru, který obsahuje ukázkou, jak lze tuto změnu provést.

Zjednodušený návod pro lokální spouštění je také součástí repozitáře s testy. Tam je v bodech rozepsán přímo v souboru readme, který je zobrazen na hlavní stránce repozitáře.

8.7 Testovací sada a pokrytí funkcionality testy

Vyvinutá sada scénářů pokrývá základní funkcionalitu aplikace EasyMiner, která je dostupná skrze grafické webové rozhraní. Scénáře byly vytvořeny pouze na funkcionalitu, která je zcela odladěná v poslední stabilní verzi aplikace, která byla dostupná v době psaní této práce. Tou je verze 2.4, s poslední aktualizací v červenci roku 2017. Všechny scénáře jsou testovány na prohlížeči Firefox, který je součástí Docker kontejneru pro spouštění testů.

Všechny testy byly napsány formou scénářů se strukturou Given-When-Then dle metodiky BDD. V kombinaci se syntaxí jazyka Robot framework, který umožňuje psát scénáře pomocí vět v podobajících se přirozenému jazyku, jsou vytvořené testy snadno čitelné a mohou sloužit také jako dokumentace. Čitelnost ilustruje scénář v níže uvedené ukázce kódu níže. Přestože ukázka využívá proměnné, nejen čtenář znalý syntaxe jazyka dokáže hned poznat, že cílem scénáře je testování úspěšného přihlášení uživatele.

Výpis 8: Ukázka scénáře pro otestování úspěšného přihlášení uživatele (zdroj: autor)

```
1 [1.2.1] Login as valid user 1
2   Given login page is opened
3   When login user as "${ValidUser1}"
4   Then user is logged in as "${ValidUser1}"
```

Syntaxe podobná přirozenému jazyku se však netýká jen definice scénářů. Také všechna klíčová slova knihovny Selenium WebDriver lze používat s využitím syntaxe podobné běžné řeči. Volání funkce *Input text* knihovny Selenium je znázorněno v ukázce kódu níže. Kód v ukázce slouží pro přihlášení uživatele pomocí vyplnění přihlašovacího formuláře.

Výpis 9: Ukázka klíčového slova, které přímo využívá funkce knihovny Selenium (zdroj: autor)

```
1 login user as "${user}"
2   Input text    css=input[type="email"][name="email"]    ${user.email}
3   Input text    css=input[type="password"][name="password"]    ${user.password}
4   Confirm standard form
```

Testovací sada obsahuje 38 na sobě nezávislých scénářů. Součástí pokrytí je správa uživatelů, nahrání datasetu a proces těžby pravidel. Součástí testování správy uživatelů je otestování registrace a přihlášení uživatele. V oblasti nahrání datasetu bylo otestováno správné nahrání csv souborů s různými oddělovači a nastavení sloupců nahraných datasetů. V rámci pokrytí procesu těžby pravidel byly vytvořeny scénáře pro otestování vytvoření atributů, nastavení vzorů pro pravidla a samotné nalezení pravidel. Dále byla otestována funkce schránky pro pravidla (rule clipboard) a báze znalostí (knowledge base).

Pro každou akci, která je otestována, byla vybrána jen jedna z možností, kterou ji lze provést. Alternativní možnosti provedení nebyly součástí testů. Příkladem je přidání atributu do vzoru pravidla při přípravě mineru. To lze provést pomocí přetáhnutí atributu do příslušné části vzoru, tlačítka vedle atributu či pomocí zaškrťovacího pole pro výběr atributu. V testovacích scénářích je ovšem akce prováděna pouze za pomoci zaškrťovacího pole. Alternativní možnosti provádění testů nebyly pokryty z důvodu vyšších nároků na údržbu testovací sady, která by tímto rozšířením testů vznikla.

8.8 Konvence pro psaní testů

Stejně jako je zvykem u kódu k aplikacím, také pro skripty k automatizovaným testům je vhodné stanovit konvence pro jejich psaní. Vhodně zvolená pravidla a jejich následné dodržování umožní zajistit snadnou čitelnost, přehlednost a údržbu kódu. Nastavení konvencí je nezbytné zejména v prostředí, kdy je kód sdílen více vývojáři a je plánováno jeho rozšiřování a dlouhodobá údržba. Také testy pro systém EasyMiner, vyvinuté jako součást této práce, bude nutné v budoucnu rozšířit a udržovat. Stejně tak se předpokládá, že postupem času se na jejich rozvoji bude podílet více vývojářů. Proto byly testy vyvíjeny dle doporučených konvencí a praktik. Mezi použité praktiky patří doporučení popsané v oficiálním návodu pro Robot framework (Robot Framework Foundation 2017b) a doplňující materiály k návodu (Ebbert-Karroum 2010, EMERY 2009, Klärck 2014). Dále je čerpáno z obecných doporučení pro psaní automatizovaných testů s využitím knihovny Selenium WebDriver (Burton 2012).

Nejdůležitější použité pravidla jsou popsána v této kapitole. Zkrácená verze uvedených konvencí je přidána do repositáře s testy. Aby pravidla nebyla přehlédnuta, je na ně odkazováno přímo z dokumentu readme, který se zobrazuje na úvodní stránce repositáře. Díky tomu je možné nastavená pravidla ctít při budoucím rozvoji testovací sady.

Prvním stanoveným pravidlem je výhradní použití angličtiny pro obsah uložený v repositáři. Jedná se nejen o samotný kód, ale také o veškerou dokumentaci, komentáře a web s výsledky testů. Účelem tohoto pravidla je umožnit participaci na rozvoji testů co největšímu množství vývojářů. Dále je konvencí psaní scénářů výhradně s využitím Given-When-Then struktury dle metodiky BDD.

8.8.1 Hledání elementů na stránce

Implementace knihovny Selenium WebDriver Robot frameworkem umožňuje vyhledávat elementy různými způsoby. Mezi ně patří hledání dle atributu id, name a class, či názvu html tagu. Pro hledání anchor elementů lze využít také hledání na základě úplné či částečné shody textu odkazu. Mezi dostupné možnosti patří také hledání elementů na základě

xpath, css, dom či jquery selektorů. Vhodná volba způsobu hledání elementů výrazně přispívá čitelnosti kódu a odolnosti testů vůči změnám v aplikaci.

Nejjednodušším a zároveň nejvhodnějším způsobem je vyhledání elementu podle id atributu, který jednoznačně identifikuje daný element na stránce. Výhodou tohoto způsobu hledání je jeho krátký, jasný zápis a nezávislost na konkrétním rozložení stránky.

Ne vždy je možné element jednoznačně určit dle jeho identifikátoru. Pro hledání takových elementů jsou určeny selektory, které umožňují vyhledávat elementy na základě kombinace několika parametrů, určených zadaným výrazem. Existuje několik možných syntaxí, s jejichž pomocí lze tyto selektory vytvářet. Pro testy systému EasyMiner byly použity výhradně selektory css a xpath. Důvodem pro tuto volbu byl fakt, že tyto selektory jsou podporovány dalšími frameworky a knihovnami postavenými nad Selenium WebDriver. Proto jejich použití zvyšuje srozumitelnost skriptů pro vývojáře, kteří pracují s jinými nástroji pro automatizaci testování webového rozhraní. Při volbě mezi css a xpath selektory bylo preferováno využití css selektorů. Preference css selektorů je doporučována hned v několika materiálech, které se zabývají nástroji využívající Selenium WebDriver. Tento postup doporučuje například Wilson (Wilson 2011) nebo Burton (Burton 2012). Mezi uváděné důvody pro využití css selektorů patří větší čitelnost a fakt, že tyto selektory jsou využívány také při samotném vývoji webových aplikací. Xpath selektory byly využity pouze v případech, kdy již nebylo možné nalezení elementu pomocí css selektorů.

Pro vytváření selektorů k hledání elementů je nezbytné znát strukturu dané webové stránky. Pro zkoumání této struktury lze využít nástroje, které jsou součástí běžně používaných webových prohlížečů. Těmito nástroji disponují mimo jiné prohlížeče Chrome, Firefox, Opera, Edge, Internet Explorer či Safari. Součástí těchto nástrojů je nejen možnost zobrazení zdrojové html struktury stránky, ale také možnost automatického vygenerování css či xpath selektorů. Této funkce je však nutné využívat s rozvahou. Automaticky vygenerované selektory nejsou vždy ideální, často jsou totiž příliš vázané na konkrétní rozložení stránky. Lepší je elementy vyhledávat pouze na základě umístění v jedné vybrané oblasti (formuláři, větší sekci stránky) a následně v ní určit element podle specifické kombinace jeho atributů. Takto psané selektory jsou méně náchylné na změny v aplikaci a přispívají tak snadnější údržbě testovací sady. Jako příklad nevyhovujícího, automaticky vygenerovaného selektoru lze uvést selektor vygenerovaný pro tlačítko pro odeslání registračního formuláře. Ten byl vygenerován pomocí nástroje pro zkoumání stránky z prohlížeče Chrome pro Windows a vypadá následovně:

Výpis 10: *Příklad nevhodně vytvořeného CSS selektoru (zdroj: autor)*

```
1 #frm-registrationForm > div.actions > input
```

Vhodnějším selektorem je však:

Výpis 11: *Příklad korektně sestaveného CSS selektoru (zdroj: autor)*

```
1 #frm-registrationForm input[type="submit"]
```

Druhý selektor pouze kontroluje, že je element v daném formuláři a následně dle atributů hledaného elementu jej jednoznačně určí. Automaticky vygenerovaný selektor je naopak více závislý na konkrétní struktuře stránky. Vyžaduje totiž, aby se element byl přímým potomkem elementu, který je přímým potomkem formuláře. V tomto případě stačí drobná úprava formuláře, která změní jeho strukturu a element nebude možné pomocí daného selektoru nalézt. Přitom taková změna formuláře nemusí nutně znamenat změnu jeho chování z pohledu funkčnosti aplikace.

Účelem testování aplikace EasyMiner je ověřit její správné chování, nikoliv její vzhled. K hledání elementů dle textu, což některé způsoby hledání elementů umožňují, je tedy třeba přistupovat s rozvahou. Nutné je odlišení situace, kdy má text stěžejní význam z hlediska správného chování aplikace a kdy je naopak jeho význam marginální. Tento rozdíl lze ilustrovat na příkladu. Text potvrzovacího tlačítka ve formuláři nemá na funkci aplikace vliv. Ačkoliv nesprávný text může uživatele systému mást, neomezuje ho v používání aplikace. Naopak ověření, že elementy s nalezenými pravidly opravdu obsahují očekávané hodnoty má smysl, protože je tím ověřeno správné chování funkce aplikace.

Uvedené principy lze shrnout do těchto bodů:

- Pokud je to možné, využívat hledání elementů dle id
- Preference css selektorů oproti xpath selektorům
- Nevyužívat dom a jquery selektory
- Omezit hledání elementu na vybranou oblast (formulář, sekci stránky), nikoliv hledat element na základě detailního popisu umístění ve struktuře stránky
- Ověřovat text elementů na stránce pouze v případě, kdy daný text má význam z hlediska správného chování aplikace
- Využívat selektory automaticky vygenerované vývojářskými nástroji webových prohlížečů s rozvahou - ne vždy jsou ideální

8.8.2 Konvence pro definici klíčových slov a proměnných

Klíčová slova by měla být definována pomocí vět psaných stejně, jako v přirozeném jazyce. Názvy klíčových slov by měli být stručné a jasné. Měli by vyjadřovat co dělají, nikoliv jakým způsobem popsané akce provádějí.

Pokud jsou nějaká data používána na více místech, je vhodné dané hodnoty uložit do proměnných. Díky tomu je pak stačí při nutnosti jejich změny upravit na jednom místě. Obzvláště je vhodné tento postup aplikovat u selektorů pro elementy. Pro snadné rozlišení globálních proměnných od lokálních proměnných je použit pro globální proměnné PascalCase a pro lokální camelCase zápis.

Pro přehlednost je vhodné využívat možnosti rozdělení skupin klíčových slov do různých souborů. Následně pak využít možnosti pro import těchto souborů tam, kde je třeba využít jimi definovaná klíčová slova do jednoho souboru.

Při tvorbě klíčových slov je často nutné čekat na načtení stránky po provedení určité akce. Správným postupem je využití jednoho z klíčových slov, které začíná slovem `Wait` a je součástí knihovny `Selenium`. Skupina těchto klíčových slov pozastavuje exekuci testů, dokud není splněna daná podmínka nebo není překročen časový limit určený předaným parametrem. Tato klíčová slova jsou naimplementována tak, že splnění dané podmínky kontrolují každých 200 ms. Proto lze pro podmínku nastavit velký časový limit, díky kterému lze zamezit chybě, pokud akce trvá o něco déle, než očekáváme. Pomalejší chod může být způsobem například spuštěním testu na slabším stroji či aktuální velké zatížení daného počítače. Zároveň díky opakované kontrole podmínky v daném časovém intervalu je zajištěno, že čekání zbytečně nezpomaluje běh scénářů.

Naopak je vhodné se vyvarovat použití klíčového slova `Sleep`. Ten dokáže pozastavit exekuci testů po dobu, která je určena předaným argumentem. Ovšem vždy čeká nečinně celou stanovenou dobu. Při nastavení příliš velkého časového limitu zbytečně prodlužuje dobu nutnou pro exekuci testů. Naopak při nastavení malého časového limitu může způsobovat chyby, které ovšem nesouvisejí s chybou testované aplikace. Navíc tento druh chyby se většinou objevuje nahodile, nikoliv při každém spuštění testů. Toto stochastické chování pak vede nejen k náročnému odhalování příčiny chyby, ale také ztrátě důvěry v testy. Z těchto důvodů je nutné používat klíčové slovo `Sleep` pouze v případech, že opravdu není jiná možnost.

8.8.3 Konvence pro tvorbu testovacích scénářů

Testy pro automatizované testování byly všechny psané formou scénářů, popisovaných v metodice BDD. Tyto scénáře mají vždy stejnou strukturu, která obsahuje klíčová slova `Given`, `When` a `Then`, neboli *Pokud*, *Když* a *Pak*. Díky tomu jsou scénáře čitelné a směřují vývojáře k psaní testů, které jsou orientovány na testování jedné akce. Čitelnosti napomáhá také fakt, že scénáře jsou psány pomocí vět v přirozeném jazyce. Více o výhodách testování na základě metodiky BDD je popsáno v samostatné kapitole 3.5. Dodržování tohoto stylu je jednou ze zásad pro psaní scénářů.

Pravidlem pro návrh scénáře je, aby se soustředil pouze na testování jedné akce a s ní spjatých změn. Použitá forma scénářů umožňuje řetězit jednotlivé kroky pomocí klíčového slova *and*, neboli spojkou *a*. Toho by mělo být používáno jen minimálně a při jeho využití stále zachovat orientaci scénáře na otestování jedné konkrétní akce. Několikanásobné použití klíčového slova *and* je jasná indikace pro rozdělení scénáře na několik menších. Scénář by měl být pro člověka se znalostí testované aplikace po prvním přečtení jasně srozumitelný. Pokud pro pochopení vyžaduje delší soustředění, indikuje to jeho špatné

navržení. Krátké scénáře s jasným cílem jsou nejen lépe čitelné, ale také umožňují jednoduší odhalení příčiny chyb při jejich selhání.

Pro seskupení scénářů podle funkcionality, kterou pokrývají, bylo využito rozdělení scénářů do samostatných souborů. Každý soubor se scénáři je dle názvosloví frameworku označen test suite. Pro seskupení testovacích souborů do funkčních oblastí bylo využito možnosti označení scénářů štítky (tag). Vzhledem k nabízené funkcionalitě systému EasyMiner je toto dvouúrovňové seskupování dostatečné. Aby bylo možné se jednoduše odkazovat na jednotlivé scénáře, název každého scénáře začíná unikátním identifikátorem. Ten je složený ze tří číslic, zapsaných ve formátu [X.Y.Z]. Číslo X je číslo přiřazené funkční oblasti, číslo Y představuje číslo přiřazené danému souboru, a nakonec Z označuje pořadí scénáře v souboru. Názvy testovacích souborů by pak měli začínat vždy identifikátorem [X.Y], kde X a Y mají stejný význam jako u identifikátorů scénářů. Jednotlivá slova použitá pro pojmenování testovacích souborů by měla být oddělena podtržítkem. Stejně jako názvy klíčových slov, také názvy testovacích souborů a scénářů by měli být krátké a výstižné.

Všechny scénáře musí být na sobě navzájem nezávislé. Není možné, aby některý scénář byl závislý na předchozím spuštění jiného scénáře. Vytvoření nežádoucí závislosti mezi scénáři je v literatuře (Wynne et al. 2017a) označováno jako vytváření leaky scenarios neboli prosakujících scénářů. Pro nastavení požadovaného výchozího stavu je vhodné využití speciálních klíčových slov Test setup, Test teardown, Test suite setup a Test suite teardown. Tyto klíčová slova umožňují provést definovanou akci před či po každém spuštění scénáře, nebo sady scénářů. Následně pak požadovaný stav pro individuální scénář upravit v první části scénáře (za klausulí Given). Díky nezávislosti je možné použít každý scénář samostatně. V případě chyby se lépe hledá její příčina, protože její hledání je omezeno pouze na přesně definovanou oblast, kterou pokrývá daný scénář.

Omezením prostředí pro exekuci testů, které je dané konfigurovaným Docker obrazem je nutnost výhradního použití prohlížeče Firefox pro všechny testy. Jiný prohlížeč není v prostředí dostupný. Dále je třeba všechny soubory s testy uložit do adresáře Test, který se nachází v kořenovém adresáři repozitáře.

8.8.4 Vytváření testovacích dat

Pro testovací sadu byla použita testovací data, která jsou použita v tutoriálu, který je vystaven na stránce projektu EasyMiner. Jedná se o dataset titanic.csv. Díky tomu testovací scénáře také ověřují, že se aplikace chová dle popisu v tutoriálu. Navíc jsou s tímto datasetem seznámeni všichni vývojáři na projektu. Tudíž v případě chyby, či nutnosti změny testovacích dat nebude pro vývojáře problém testovací data změnit. Podobně je vhodné využívat již vytvořená data při budoucím rozvoji testovací sady. Taková data lze nalézt nejen v tutoriálu, ale také například v repozitáři EasyMiner-Evaluation, který testuje REST API aplikace EasyMiner.

Pro vytváření nových testovacích dat lze využít technik popisovaných v materiálech vydaných organizací ISQTB. Jedná se mimo jiné o techniky rozdělení do tříd ekvivalence, analýzy hraničních hodnot, či testování přechodu stavů. Analýza hraničních hodnot byla použita při vývoji například pro otestování registračního formuláře, kdy pro testování pole s heslem. Korektní heslo musí být délky nejméně šest znaků. Proto byla použita hesla nulové délky, délky pět znaků (těsně pod hranici) a délky šesti znaků (přesně na hranici).

8.9 Shrnutí implementovaného řešení

Tato část práce se zabývala implementací a nasazením testů grafického webového rozhraní v rámci průběžné integrace pro open source projekt [EasyMiner](#). Implementace byla vyvinuta na základě navrženého řešení pro open source projekty, které je popsáno v kapitole 7. Implementace tak slouží také jako demonstrace použití navrženého řešení pro konkrétní projekt. Stejně jako obecné navržené řešení, tato implementace disponuje výhodami, mezi které patří nulové náklady na provoz, rychlé nasazení a snadná údržba.

Řešení se skládá ze sady testů grafického rozhraní, vytvořeného nástrojem Robot framework, cloudového integračního serveru Travis CI a webové stránky pro uchovávání reportů a dalších informací.

Travis CI je nastaven tak, aby integrace probíhaly pouze na hlavních vývojových větvích repozitáře s projektem. Mezi ně patří větev pro vyvíjení nové funkcionality master a větev pro údržbu stabilní verze 2.4. Nedochází tak k zbytečnému zabírání místa ve frontě na serveru integracemi vedlejších větví. Spouštěčem integrace je změna sledovaných větví v repozitáři frontendové části projektu EasyMiner ([EasyMinerCenter](#)) a repozitáři s testy ([EasyMiner-Tests](#)). Notifikace s výsledky integrace jsou zasílány všem vývojářům na projektu pomocí e-mailu. Díky tomu jsou vývojáři pravidelně informováni o aktuálním stavu projektu. Výsledky jednotlivých běhů integrace jsou také veřejně dostupné zde [Travis-EasyMinerCenter](#) a zde [Travis-EasyMiner-Tests](#).

Testovací sada byla vytvořena pomocí nástroje Robot framework. Testy byly psány formou Given-When-Then scénářů, které jsou součástí agilní metodiky behaviour driven development. Dále při vývoji scénářů byly využity doporučené konvence a praktiky. Mezi použité praktiky patří doporučení popsané v oficiálním návodu pro Robot framework a doplňující materiály k tomuto návodu. Dále bylo čerpáno z obecných doporučení pro psaní automatizovaných testů s využitím knihovny Selenium WebDriver. Pro vytváření testovacích dat bylo využito technik popsanych v materiálech vydaných organizací ISQTB, která se zabývá testováním. Shrnutí použitých konvencí a praktiky bylo přidáno do repozitáře s testy. Díky tomu je možné nastavená pravidla ctít při budoucím rozvoji testovací sady a udržet tak jejich kvalitu a snadnou údržbu.

Webové stránky s výsledky testů obsahují také vygenerované reporty z nástroje Robot framework. Ty mohou sloužit nejen jako pomocný nástroj k rychlému odhalení chyb a jejich příčin, ale také jako dokumentace aktuálního chování aplikace. Současně jsou na stránkách dostupné také logy z frontendové části aplikace EasyMiner. Ty poskytují další cenné informace o běhu integrace, které slouží k rychlejšímu vyřešení problémů. Stránky jsou veřejně dostupné zde: <https://kizi.github.io/EasyMiner-Tests/>

9 Závěr

Tato diplomová práce se zabývala problematikou testování grafického webového rozhraní v rámci průběžné integrace. Zaměřovala se na open source projekty vystavené na GitHubu, které pro své nasazení využívají Docker kontejnery. Její součástí bylo vytvoření sady testů pro webové rozhraní open source projektu EasyMiner.

9.1 Dosažení stanovených cílů

Byly stanoveny dva dílčí cíle práce. Prvním cílem bylo nalézt technické řešení, které by umožňovalo testovat webové uživatelské rozhraní open source projektů v rámci průběžné integrace. Jako omezující kritéria byla stanovena nutnost podpory projektů uložených na serveru GitHub a podpory testování aplikací, které jsou nasazovány s využitím Docker kontejnerů a využití výhradně bezplatných komponent. Navržené řešení využívá cloudový server pro průběžnou integraci Travis CI. Testy webového rozhraní, psané pomocí nástroje Robot framework jsou spouštěny pomocí Docker kontejneru. Pro tento Docker kontejner byl vytvořen speciální obraz [soulekamil/robotframework-ff](https://hub.docker.com/r/soulekamil/robotframework-ff), který je zveřejněný na centrálním úložišti obrazů Docker Hub. Ten lze využít pro spouštění testů nejen na serveru Travis CI, ale také lokálně na počítači vývojářů. Nástroj Docker je také použit pro nasazení a spuštění testovaného systému. Další významnou komponentou řešení jsou webové stránky, které jsou zveřejněny pomocí služby GitHub Pages. Tyto stránky slouží k ukládání a prezentaci reportů a dalších souborů vytvořených během integrace projektu. Součástí návrhu je také návod popisující, jak lze provést nahrání souborů vytvořených během integrace z Travis CI na zveřejněný web.

Mezi výhody tohoto řešení patří nulové náklady na provoz, rychlé nasazení a snadná údržba. Nulové náklady na provoz jsou zajištěny díky využití komponent, které jsou dostupné zcela zdarma pro open source projekty. Rychlého nasazení je docíleno díky využití cloudového integračního serveru, který je dostupný jako služba okamžitě. Rychlému nasazení dopomáhá také zveřejněný obraz pro spouštění testů, který je již připraven k použití. Snadná údržba je zajištěna za pomoci několika faktorů. Díky využití cloudového serveru odpadá nutnost starat se o údržbu a aktualizaci infrastruktury, na rozdíl od využití lokálně spravovaného řešení. Správa uživatelských účtů a oprávnění je jednoduchá, protože pro repositáře s testy, projektem, obsahem webových stránek a Travis CI jsou používány stejné GitHub uživatelské účty. Dále díky využití nástroje pro testování pomocí klíčových slov je možné do vývoje testů zapojit také technicky méně zdatné uživatele. Zároveň lze testy psát tak, aby mohly být současně použity jako dokumentace chování aplikace. Samostatný web s možností ukládat detailní informace o proběhlých integracích napomáhá rychlému řešení případných chyb.

Navržené řešení není určeno pro jeden konkrétní druh projektů, ale je obecně využitelné pro široké spektrum webových open source projektů. Vzhledem k obecnému použití řešení není jeho součástí detailní popis všech nastavení zvolených komponent. To závisí na konkrétním projektu, pro který má být implementováno. Práce zároveň obsahuje přehled nástrojů pro vytváření automatizovaných testů a cloudových serverů pro průběžnou integraci. Tím poskytuje čtenáři informace o možných alternativách k navrženému řešení. Nalezené řešení splňuje stanovený cíl práce.

Druhým stanoveným cílem práce bylo demonstrovat nalezené řešení pomocí vývoje sady testů pro jednoho reprezentanta ze skupiny webových open source projektů, které využívají Docker pro nasazení aplikace. Vybraným reprezentantem se stal projekt EasyMiner. Tato open source aplikace pro dolování asociačních pravidel je vyvíjená týmem v rámci KIZI VŠE. Pro testování této aplikace byly testy psány formou Given-When-Then scénářů, které jsou součástí agilní metodiky behaviour driven development. Dále při vývoji scénářů byly využity doporučené konvence a praktiky. Mezi použité praktiky patří doporučení popsané v oficiálním návodu pro Robot framework a doplňující materiály k tomuto návodu. Dále bylo čerpáno z obecných doporučení pro psaní automatizovaných testů s využitím knihovny Selenium WebDriver. Pro vytváření testovacích dat bylo využito technik popsanych v materiálech vydaných organizací ISQTB, která je autoritou v oblasti testování. Stejně jako navržené řešení pro obecné použití tato implementace disponuje výhodami, mezi které patří nulové náklady na provoz, rychlé nasazení a snadná údržba. Díky využití praktik z metodiky BDD při tvorbě testovacích scénářů lze vygenerované reporty z průběhu testování využít také jako dokumentaci chování aplikace.

Implementované řešení je nasazeno do provozu. Vyvinutá sada scénářů pokrývá základní funkcionalitu aplikace EasyMiner, která je dostupná skrze grafické webové rozhraní. Scénáře byly vytvořeny pouze na funkcionalitu, která je zcela odladěná v poslední stabilní verzi aplikace, která byla dostupná v době psaní této práce. Tou je verze 2.4, s poslední aktualizací v červenci roku 2017. Všechny scénáře jsou testovány na prohlížeči Firefox. Testovací sada obsahuje celkem 38 na sobě nezávislých scénářů. Součástí pokrytí je správa uživatelů, nahrání datasetu a proces těžby pravidel.

Vytvořená sada testů byla zakomponována do cloudového serveru pro průběžnou integraci Travis CI. Konfigurační soubory pro Travis CI jsou součástí integrovaných repositářů, mezi které patří repositář s frontendovou částí aplikace EasyMiner ([EasyMinerCenter](https://github.com/kizi/EasyMinerCenter)) a repositář s testy ([EasyMiner-Tests](https://github.com/kizi/EasyMiner-Tests)). Výsledky proběhlých integrací jsou veřejně přístupné. Pro frontendovou část je lze nalézt zde [Travis-EasyMinerCenter](https://github.com/kizi/EasyMinerCenter) a pro testy aplikace jsou dostupné zde [Travis-EasyMiner-Tests](https://github.com/kizi/EasyMiner-Tests). Vytvořené stránky pro ukládání reportů a dalších informací o proběhlých integracích jsou veřejně dostupné na adrese <https://kizi.github.io/EasyMiner-Tests/>. Vzhledem k tomu, že implementace byla provedena základě předchozího návrhu a byla úspěšně nasazena, byl splněn také druhý dílčí cíl práce.

Splněny byly všechny stanovené cíle práce. Na výstupy této práce je možné v budoucnu navázat několika způsoby. Řešení pro průběžnou integraci open source projektů by bylo možné rozšířit o možnost průběžné dodávky a umožnit tak pravidelné nasazení systému do vybraného prostředí. Dalším možným rozšířením vytvořeného řešení je rozšíření způsobů ověřování kvality integrovaného projektu. Zakomponovat lze například statické analýzy kvality kódu či jednotkové testování.

Při tvorbě implementace testovací sady pro testování EasyMineru bylo myšleno na možné budoucí rozšíření a údržbu. Součástí repozitáře s testy je shrnutí použitých konvencí a praktik. Díky tomu je možné nastavená pravidla ctít při budoucím rozvoji testovací sady a udržet tak jejich kvalitu a snadnou údržbu.

9.2 Přínos práce

Přínosem této práce je možnost jejího využití jako návodu, podle kterého lze zajistit začlenění testování webového rozhraní do procesu průběžné integrace pro široké spektrum open source projektů vystavených na GitHubu. Konkrétní zaměření na open source projekty a zcela bezplatné řešení přináší nový náhled do problematiky průběžné integrace. Dalším přínosem práce je zajištění testování základní funkcionality webového rozhraní aplikace EasyMiner během průběžné integrace. Díky tomu vývojový tým projektu EasyMiner dostává zpětnou vazbu o stavu aplikace po každém přidání přírůstku kódu do repozitáře.

Řešení popsané v této práci využívá moderních technických nástrojů, jakými jsou například cloudové služby a nástroj pro tvorbu softwarových kontejnerů Docker. Ty následně kombinuje s osvědčenými praktikami, mezi které patří postupy z agilní metodiky Behaviour Driven Development či techniky popsané organizací ISTQB. Díky tomu je dosaženo mnoha kladných vlastností, mezi které patří rychlé nasazení a snadná údržba.

Práce zároveň obsahuje přehled nástrojů pro vytváření automatizovaných testů a cloudových serverů pro průběžnou integraci. Tím poskytuje čtenáři informace o možných alternativách k navrženému řešení.

Navržené řešení, které je výstupem této práce ukazuje, že je možné zavést plnohodnotný proces průběžné integrace na open source projektech s nulovými provozními náklady.

Příloha A: Zdrojové kódy a výstupy testů aplikace EasyMiner

Součástí této práce bylo vytvoření testovací sady pro otestování základní funkcionality webového rozhraní open source projektu EasyMiner. Vytvořená sada testů byla zakomponována do cloudového serveru pro průběžnou integraci Travis CI. Nasazené řešení je veřejně dostupné. Jeho součásti lze nalézt zde:

- GitHub repositář [EasyMiner-Tests](https://github.com/KIZI/EasyMiner-Tests). Součástí repositáře jsou testy webového rozhraní aplikace EasyMiner a také obsah webových stránek pro ukládání reportů a dalších informací o proběhlých integracích. Odkazy na poslední verze větví repositáře v rámci DP:
 - <https://github.com/KIZI/EasyMiner-Tests/tree/a3fe98770d188f9c973d39a5d33d73f2a0e7dd26>
 - <https://github.com/KIZI/EasyMiner-Tests/tree/4c46527b8e4896acf85bf398f8f8ab8f46f7ec2a>
 - <https://github.com/KIZI/EasyMiner-Tests/tree/d801486d127d71b8e06b76f47c88782e35487467>
- Docker image pro spouštění testů napsaných nástrojem Robot Framework
 - <https://hub.docker.com/r/soulekamil/robotframework-ff/>
- Travis CI – integrace repositáře s testy:
 - <https://travis-ci.org/KIZI/EasyMiner-Tests>
- Travis CI – integrace repositáře s frontendem aplikace EasyMiner:
 - <https://travis-ci.org/KIZI/EasyMiner-EasyMinerCenter>
- Webové stránky pro ukládání reportů a dalších informací o proběhlých integracích
<https://kizi.github.io/EasyMiner-Tests/>

Dále jsou zdrojové kódy k této implementaci a další související výstupy ke stažení jako příloha k této práci na webu Vysoké školy ekonomické v Praze. V případě tištěného verze práce jsou dostupné na přiloženém CD. Elektronická příloha je zip archiv s adresáři, jejichž obsah je popsán níže.

Implementace testů pro webové rozhraní aplikace EasyMiner

Součástí adresáře jsou všechny tři větve repositáře EasyMiner-Tests, který obsahuje implementované testy pro webové rozhraní aplikace EasyMiner a také obsah webových stránek pro ukládání reportů a dalších informací o proběhlých integracích.

Docker image pro spouštění testů napsaných nástrojem Robot Framework

V adresáři lze nalézt repositář, který obsahuje definici Docker image. Tento image umožňuje spouštění testů napsaných nástrojem Robot Framework a byl použit pro spouštění testů pro webové rozhraní aplikace EasyMiner.

Konfigurační soubory pro server Travis CI použité pro nasazení testů

Tento adresář obsahuje konfigurační soubory pro server Travis CI, které byly použity pro zavedení průběžné integrace repositáře s frontendem aplikace EasyMiner - [EasyMiner-EasyMinerCenter](#).

Vygenerovaný report z běhu implementovaných testů pro aplikaci EasyMiner

Součástí adresáře je jeden report z běhu testů webového rozhraní projektu EasyMiner verze v2.4. Report byl vygenerován pomocí nástroje Robot Framework během integrace na serveru Travis CI.

Použité informační zdroje

- 3QI LABS. *Types of Automation Frameworks* [online]. 3Qi Labs Inc. © 2015 [vid. 2017-08-13]. Dostupné z: <http://3qilabs.com/types-of-automation-frameworks/>
- AGILE ALLIANCE. *BDD: Learn about Behavior Driven Development* [online]. Agile alliance, 2015a. [vid. 2017-06-01]. Dostupné z: [https://www.agilealliance.org/glossary/bdd/#q=~\(filters~\(postType~\(page~post~aa_book~aa_event_session~aa_experience_report~aa_glossary~aa_research_paper~aa_video\)~tags~\(bdd\)\)~searchTerm~sort~false~sortDirection~asc~page~1\)](https://www.agilealliance.org/glossary/bdd/#q=~(filters~(postType~(page~post~aa_book~aa_event_session~aa_experience_report~aa_glossary~aa_research_paper~aa_video)~tags~(bdd))~searchTerm~sort~false~sortDirection~asc~page~1))
- AGILE ALLIANCE. *What is "Given - When - Then"?* [online]. Agile alliance, 2015b. [vid. 2017-06-01]. Dostupné z: [https://www.agilealliance.org/glossary/gwt/#q=~\(filters~\(postType~\(page~post~aa_book~aa_event_session~aa_experience_report~aa_glossary~aa_research_paper~aa_video\)~tags~\(given*20when*20then\)\)~searchTerm~sort~false~sortDirection~asc~page~1\)](https://www.agilealliance.org/glossary/gwt/#q=~(filters~(postType~(page~post~aa_book~aa_event_session~aa_experience_report~aa_glossary~aa_research_paper~aa_video)~tags~(given*20when*20then))~searchTerm~sort~false~sortDirection~asc~page~1))
- AGILE ALLIANCE. *Behavior Driven Development (BDD)* [online]. Agile alliance, 2015c. [vid. 2017-06-12]. Dostupné z: [https://www.agilealliance.org/glossary/bdd/#q=~\(filters~\(postType~\(page~post~aa_book~aa_event_session~aa_experience_report~aa_glossary~aa_research_paper~aa_video\)~tags~\(bdd\)\)~searchTerm~sort~false~sortDirection~asc~page~1\)](https://www.agilealliance.org/glossary/bdd/#q=~(filters~(postType~(page~post~aa_book~aa_event_session~aa_experience_report~aa_glossary~aa_research_paper~aa_video)~tags~(bdd))~searchTerm~sort~false~sortDirection~asc~page~1))
- APPVEYOR SYSTEMS. *Getting started | AppVeyor* [online]. Appveyor Systems, Inc © 2017a [vid. 2017-07-27]. Dostupné z: <https://www.appveyor.com/docs/>
- APPVEYOR SYSTEMS. *Pricing | AppVeyor* [online]. Appveyor Systems, Inc © 2017b [vid. 2017-07-27]. Dostupné z: <https://www.appveyor.com/pricing/>
- BIDELMAN Eric. *Getting Started with Headless Chrome* [online]. 2017-08-17 [vid. 2017-08-20]. Dostupné z: <https://developers.google.com/web/updates/2017/04/headless-chrome>
- BURNS, David a STEWART Simon. *WebDriver W3C Candidate Recommendation 30 March 2017* [online]. W3C, 2017 [vid. 2017-07-20]. Dostupné z: <https://www.w3.org/TR/2017/CR-webdriver-20170330/>
- BURTON, Ben. *Functional Automated Testing Best Practices with Selenium WebDriver*. In: Vimeo [online]. Zveřejněno 15. 06. 2012 [vid. 2017-08-20]. Dostupné z: <https://vimeo.com/44133409>
- CASTB. *Certifikovaný tester - Učební osnovy pro základní stupeň* [online]. CaSTB, 2013. [vid. 2017-05-25]. Dostupné z: <http://castb.org/wp-content/uploads/2013/11/ISTQB-CTFL-Syllabus-v2011-CZ-Beta1.pdf>
- CIRCLECI. *Plans and Plan Information – Circle CI* [online]. CircleCI © 2017a [vid. 2017-07-29]. Dostupné z: <https://circleci.com/pricing/>
- CIRCLECI. *Circle CI Documentation – Circle CI* [online]. CircleCI © 2017b [vid. 2017-07-29]. Dostupné z: <https://circleci.com/docs/>
- COIS, Arron C. *Continuous Integration in DevOps*. In: *Carnegie Mellon University's DevOps blog* [online]. 2015-01-26 [vid. 2017-08-17]. Dostupné z: <https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html>

- CUCUMBER. *Documentation Cucumber* [online]. Cucumber limited © 2017 [vid. 2017-08-15]. Dostupné z: <https://cucumber.io/docs>
- CUCUMBER. *Custom Formatters* [online]. Cucumber limited © 2016 [vid. 2017-08-20]. Dostupné z: <https://github.com/cucumber/cucumber/wiki/Custom-Formatters>
- DOCKER. *What is Docker?* [online]. Docker, Inc © 2017a [vid. 2017-06-01]. Dostupné z: <https://www.docker.com/what-docker>
- DOCKER. *What is a Container | Docker* [online]. Docker, Inc © 2017b [vid. 2017-08-17]. Dostupné z: <https://www.docker.com/what-container>
- DUVALL, Paul M, MATYAS, Steve a GLOVER, Andrew. *Continuous integration: improving software quality and reducing risk*. Upper Saddle River, NJ: Addison-Wesley, 2007. ISBN 0321336380.
- EBBERT-KARROUM, Andreas. How to Structure a Scalable And Maintainable Acceptance Test Suite. In: Codecentric blog [online]. 2010-07-20 [vid. 2017-09-01]. Dostupné z: <https://blog.codecentric.de/en/2010/07/how-to-structure-a-scalable-and-maintainable-acceptance-test-suite/>
- EMERY Dale H. *Writing Maintainable Automated Acceptance Tests* [online]. 2009 [vid. 2017-08-23]. Dostupné z: http://dhemery.com/pdf/writing_maintainable_automated_acceptance_tests.pdf
- FELTER Wes, FERREIRA, Alexandre, RAJAMONY, Ram a RUBIO, Juan. *An Updated Performance Comparison of Virtual Machines and Linux Containers* [online]. IBM Research, 2014. [vid. 2017-06-01]. Dostupné z: [https://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/\\$File/rc25482.pdf](https://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf)
- FEWSTER, Mark a Dorothy GRAHAM. *Software test automation: effective use of test execution tools*. Reading, MA: Addison-Wesley, 1999. ISBN 9780201331400
- FITZGERALD, Brian a STOL, Klaas-Jan. Continuous Software Engineering: A Roadmap and Agenda. In: *Journal of Systems and Software*. Ireland, Lero—the Irish Software Research Centre, University of Limerick, 2015, s. 176-189 [vid. 2017-05-28]. ISSN 0164-1212. Dostupné z: <http://dx.doi.org/10.1016/j.jss.2015.06.063>
- FOWLER, Martin. Continuous Integration. In: *Martin Fowler's blog* [online]. 2006-05-01 [vid. 2017-05-28]. Dostupné z: <https://www.martinfowler.com/articles/continuousIntegration.html>
- FOWLER, Martin. GivenWhenThen. In: *Martin Fowler's blog* [online]. 2013-08-21 [vid. 2017-05-28]. Dostupné z: <https://martinfowler.com/bliki/GivenWhenThen.html>
- GITHUB. *Travis CI* [online]. GitHub, Inc © 2017 [vid. 2017-07-18]. Dostupné z: <https://github.com/marketplace/travis-ci>
- GIRMAN Bogdan. *Docker image bogdangi/robot-docker* [software]. [přístup 2. září 2017]. Dostupné z: <https://hub.docker.com/r/bogdangi/robot-docker/>. Požadavky na systém: Docker verze 1.12 a vyšší.
- GIRDWOOD, Andrew R. H. What is a headless browser? In: *Andrew R H Girdwood's blog* [online]. 2009-10-07 [vid. 2017-09-20]. Dostupné z: <http://blog.arhg.net/2009/10/what-is-headless-browser.html>

- GROSSER, Michael. *parallel_tests* [online]. Michael Grosser, 2017 [vid. 2017-08-14]. Dostupné z: https://github.com/grosser/parallel_tests
- GUNDECHA, Unmesh. *Selenium Testing Tools Cookbook - Second Edition*. Packt Publishing, 2015. ISBN 9781784392512
- HAVLÍČKOVÁ Anna a ROUDENSKÝ Petr. *Řízení kvality softwaru: Průvodce testováním*. Computer Press, 2013. ISBN 9788025145197.
- HAEFFNER, Dave. *How To Run Your Tests Headlessly with Xvfb* [online]. Dave Haeffner © 2017 [vid. 2017-08-23]. Dostupné z: <http://elementalselenium.com/tips/38-headless>
- HUMBLE, Jez a FARLEY, David. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Upper Saddle River, NJ: Addison-Wesley, 2010. ISBN 9780321601919.
- CHELIMSKY, David. *The RSpec book: behaviour-driven development with RSpec, Cucumber, and Friends*. Lewisville, Tex.: Pragmatic, 2010. ISBN 1934356379.
- ISQTB. *Certified Tester - Advanced Level Syllabus -Technical Test Analyst* [online]. ISQTB, 2012. [vid. 2017-06-25]. Dostupné z: <http://www.istqb.org/downloads/send/10-advanced-level-syllabus-2012/55-advanced-level-syllabus-2012-technical-test-analyst.html>
- JBEHAVE. *Features of JBehave* [online]. JBehave Project © 2017a [vid. 2017-08-21]. Dostupné z: <http://jbehave.org/reference/stable/features.html>
- JBEHAVE. *Story Syntax* [online]. JBehave Project © 2017b [vid. 2017-08-21]. Dostupné z: <http://jbehave.org/reference/stable/story-syntax.html>
- KARAM, Lea. *A few benefits you get by doing BDD* [online] Apiumhub, 2017-05-09. [vid. 2017-06-12]. Dostupné z: <https://apiumhub.com/tech-blog-barcelona/behavior-driven-development/>
- KIZI. *Easy data mining of association rules - EasyMiner - pattern mining in the browser* [online]. Prague: KIZI, University of Economics © 2017a [vid. 2017-07-20]. Dostupné z: <http://www.easyminer.eu/>
- KIZI. *For developers - EasyMiner - pattern mining in the browser* [online]. Prague: KIZI, University of Economics © 2017b [vid. 2017-07-22]. Dostupné z: <http://www.easyminer.eu/developers>
- KIZI. *EasyMiner-EasyMinerCenter* [online]. Prague: KIZI, University of Economics © 2017c [vid. 2017-10-26]. Dostupné z: <https://github.com/KIZI/EasyMiner-EasyMinerCenter>
- KIZI. *EasyMiner-Backend* [online]. Prague: KIZI, University of Economics © 2017d [vid. 2017-10-26]. Dostupné z: <https://github.com/KIZI/EasyMiner-Backend/>
- KIZI. *EasyMiner-Scorer* [online]. Prague: KIZI, University of Economics © 2017e [vid. 2017-10-26]. Dostupné z: <https://github.com/KIZI/EasyMiner-Scorer/>
- KLÄRCK, Pekka. *Robot Framework Dos And Don'ts* [online]. 2014-09-01 [vid. 2017-08-25]. Dostupné z: <https://www.slideshare.net/pekkaklarck/robot-framework-dos-and-donts>

- KOLPAKOVA, Alexandra. *Využití automatizovaných regresních testů v systému kontinuální integrace webové aplikace*. Praha, 2016. Bakalářská práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Alena Buchalceová. Dostupné také z: <https://insis.vse.cz/zp/58444/podrobnosti>
- KRIŠTOF, Tomáš. *Automatické testování webového informačního systému*. Brno, 2016. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce David Sehnal. Dostupné také z: http://is.muni.cz/th/430609/fi_m/
- MAČUROVÁ, Kateřina. *Testování aplikací s využitím nástroje Robot Framework*. Praha, 2015. Diplomová práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Alena Buchalceová. Dostupné také z: <https://insis.vse.cz/zp/55487/podrobnosti>
- NORTH, Dan. Introducing BDD. In: *Dan North's blog* [online]. 2006 [vid. 2017-06-3]. Dostupné z: <https://dannorth.net/introducing-bdd/>
- LINDSAY, Jeff. Web hooks to revolutionize the web. In: *Jeff Lindsay's blog* [online]. 2007-05-03 [vid. 2017-08-20]. Dostupné z: <http://progrum.com/blog/2007/05/03/web-hooks-to-revolutionize-the-web/>
- RENDERED TEXT. *Plans & Pricing - Semaphore* [online]. Rendered Text © 2017a [vid. 2017-07-23]. Dostupné z: <https://semaphoreci.com/docs/>
- RENDERED TEXT. *Documentation - Semaphore, Continuous Integration & Deployment* [online]. Rendered Text © 2017b [vid. 2017-07-23]. Dostupné z: <https://semaphoreci.com/pricing>
- ROBOT FRAMEWORK FOUNDATION. *Robot Framework* [online]. © 2017a [vid. 2017-08-14] Dostupné z: <http://robotframework.org/>
- ROBOT FRAMEWORK FOUNDATION. *How to write good test cases using Robot Framework* [online]. 2017-02-20 [vid. 2017-08-20]. 2017b. Dostupné z: <https://github.com/robotframework/HowToWriteGoodTestCases/blob/master/HowToWriteGoodTestCases.rst>
- ROBOT FRAMEWORK FOUNDATION. *Robot Framework – the libraries* [online]. © 2017c [vid. 2017-08-15] Dostupné z: <http://robotframework.org/#libraries>
- ROBOT FRAMEWORK FOUNDATION. *Robot Framework – the tools* [online]. © 2017d [vid. 2017-08-15] Dostupné z: <http://robotframework.org/#tools-editors>
- SELENIUM PROJECT. *Selenium WebDriver* [online]. Selenium Project © 2012 [vid. 2017-06-14]. Dostupné z: http://www.seleniumhq.org/docs/03_webdriver.jsp
- SMART John F. *BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle*. Manning Publications Company, 2014. ISBN 9781617291654.
- ŠPALEK, Ondřej. *Pokročilé možnosti automatizovaného testování nástrojem Selenium Webdriver*. Praha, 2015. Diplomová práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Alena Buchalceová. Dostupné také z: <https://insis.vse.cz/zp/51517/podrobnosti>
- THOUGHTWORKS. *Gauge | ThoughtWorks* [online]. ThoughtWorks Inc. © 2016a [vid. 2017-07-21]. Dostupné z: <https://getgauge.io/>
- THOUGHTWORKS. *Gauge example in Ruby* [online]. ThoughtWorks Inc. © 2016b [vid. 2017-10-26]. Dostupné z: <https://github.com/getgauge-examples/ruby-selenium>

- THOUGHTWORKS. *Plugins - Gauge 0.9.0 documentation* [online]. ThoughtWorks Inc. © 2017 [vid. 2017-07-28]. Dostupné z: <https://docs.getgauge.io/plugins.html#>
- TRAVIS CI. *Travis CI User Documentation* [online]. Travis CI, GmbH © 2017a [vid. 2017-07-18]. Dostupné z: <https://docs.travis-ci.com/>
- TRAVIS CI. *Plans and Prices for Travis CI* [online]. Travis CI, GmbH © 2017b [vid. 2017-07-18]. Dostupné z: <https://travis-ci.com/plans>
- TRÍSKOVÁ, Lucie. *Testování webových aplikací s využitím nástroje Selenium Webdriver*. Praha, 2015. Diplomová práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Alena Buchalceková. Dostupné také z: <https://insis.vse.cz/zp/45189/podrobnosti>
- VODIČKA, Petr. *Behaviour driven development*. Praha, 2011. Bakalářská práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Alena Buchalceková. Dostupné také z: <https://insis.vse.cz/zp/35290/podrobnosti>
- WHATMUFF Daniel. *Docker image danielwhatmuff/robot-docker* [software]. [přístup 2. září 2017]. Dostupné z: <https://hub.docker.com/r/danielwhatmuff/robot-docker/> . Požadavky na systém: Docker verze 1.12 a vyšší.
- WILSON, Ashley. *Why CSS Locators are the way to go vs XPath* [online]. 2011-05-17 [vid. 2017-08-28]. Dostupné z: <https://saucelabs.com/blog/why-css-locators-are-the-way-to-go-vs-xpath>
- WYNNE, Matt, ASLAK, Hellesoy TOOKE, Steve. *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. 2nd ed. Pragmatic Bookshelf, 2017a. ISBN 9781680502381.
- WYNNE, Matt, ASLAK, Hellesoy TOOKE, Steve. *Source Code for The Cucumber Book* [online]. Pragmatic Bookshelf, 2017b. [vid. 2017-10-25]. Dostupné z: https://pragprog.com/titles/hwcuc/source_code
- YAML. *YAML Ain't Markup Language* [online]. YAML.org © 2006 [vid. 2017-10-25]. Dostupné z: <http://www.yaml.org/about.html>