UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 Marek Valda

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Automatizace webu

Marek Valda

Bakalářská práce

2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 17. 08. 2017

Marek Valda

PODĚKOVÁNÍ

Tímto bych rád poděkoval panu Ing. Janu Mertovi, který mně byl velice nápomocný a díky kterému tato práce přesáhla mé prvotní očekávání.

ANOTACE

Hlavním cílem bakalářské práce je zautomatizování postupů ve webovém prohlížeči pomocí nástroje Selenium WebDriver. Samotná automatizace funguje na bázi skriptů, které si uživatel předem definuje podle potřeby.

Aplikaci je možno využít k testování funkcionality webových aplikací, zjednodušení každodenních opakujících se úkonů a v neposlední řadě také pro získávání dat. Projekt je implementovaný v programovacím jazyce Ruby.

KLÍČOVÁ SLOVA

automatizace, Ruby, Selenium WebDriver, testování webových aplikací, skript

TITLE

Web automation

ANNOTATION

Main goal of bachelor thesis is to automate procedures at web browser with Selenium WebDriver tool. Functioning of automations is based on scripts, which user defines according his needs.

Application could be used to test web applications functions, simplification everyday routines and last but not least data acquiring. Project is implemented in programming language Ruby.

KEYWORDS

automation, Ruby, Selenium WebDriver, web application testing, script

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

CSP	Content Security Policy
API	Application Programming Interface
MVC	Model View Controller
RoR	Ruby on Rails
CRUD	Create Read Update Delete
UI	User Interface
SQL	Structured Query Language
CSS	Cascading Style Sheets
XPath	XML Path Language
IDE	Integrated Development Environment
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
SMTP	Simple Mail Transfer Protocol
POP3	Post Office Protocol 3
IMAP	Internet Message Access Protocol
TCP	Transmission Control Protocol
FTP	File Transfer Protocol

1 ÚVOD

V dnešní době téměř vše, na co si člověk vzpomene, je svým způsobem svázáno s informačními technologiemi. Softwarový trh je plný různorodých aplikací, které mají uživateli zjednodušit, či zpříjemnit život. V důsledku toho, by mělo být cílem vývojářů vytvářet aplikaci tak, aby se uživateli lehce ovládala a plnila svůj účel, namísto přidělávání dalších problémů k řešení.

Dalo by se říci, že svět spěje do bodu, kde jsou opakované činnosti postupně nahrazovány procesem, který je zcela automatizovaný. Takový proces se stává optimálním z hlediska financí a také proto, že je z něho odebrán lidský faktor, tudíž by nemělo dojít k chybě, kterou by mohl člověk omylem zavinit. Samozřejmě se také zkracuje čas potřebný pro jednotlivé úkony. Automaticky prováděné procesy dokáží zpracovat obrovské množství dat za velice krátký časový interval.

Ovšem každý z těchto případů má své výhody a také nevýhody. Automatizovaný proces nedokáže reagovat na změnu v postupu a stává se nepoužitelným do okamžiku, než bude jeho definovaná metodika upřesněna nebo změněna, zatímco uživatel na tyto události dokáže reagovat na základě vlastního rozhodnutí. Stále však může některé kroky opomenout, pokazit či jinak modifikovat, a to v některých případech může znamenat kritický neúspěch.

Nejlepším řešením by se tedy zdál být proces takový, jenž by byl automatizovaný, a pokud by nastala událost, která by nebyla očekávána, aplikace by na ni sama dokázala zareagovat v přípustném měřítku. V takovém případě už s nadsázkou mluvíme o umělé inteligenci s možností vlastních interakcí.

1.1 Zaměření práce a důvod výběru tématu

Bakalářská práce je zaměřena na vytváření automatických postupů, které definuje uživatel. Tyto postupy mohou být prováděny v určitý čas a mohou se opakovat v předem nastavených časových intervalech. Dané automatizace mohou být použity pro veškeré operace, které nevyžadují dynamické rozhodování. Pokud tedy existuje úkon, který uživatel provádí a jeho postup je vždy stejný, jedinou proměnou zůstávají např. data, lze takový postup zautomatizovat pomocí této aplikace.

Téma bylo vybráno kvůli jeho zajímavosti a následné použitelnosti. Automatizované procesy se zdají být nedílnou součástí každodenního života a také jsou hlavním cílem budoucnosti, proto si autor myslí, že tato aplikace má potenciál, který by mohl být v určitých oblastech využitelný, ať už pro pracovní, osobní nebo vzdělávací prostředí.

1.2 Cíl práce

Hlavním cílem práce je vytvoření aplikace, která by dokázala vést uživatele k vytvoření fungujícího automatického procesu, tvořeného pomocí dílčích skriptů, které je možno definovat díky uživatelsky přívětivému rozhraní tak, aby se v ní dokázal orientovat i uživatel s minimálními zkušenosti v dané oblasti.

Aplikace ke svému fungování bude využívat nástroj Selenium a také by měla být schopna obsluhovat nejrozšířenější prohlížeče, jako jsou Google Chrome a Mozzila Firefox. Dále by měla být schopna používat bez-oknový prohlížeč PhantomJs, který je určený především k testování, tak aby jednotlivé operace byly schopné běžet na pozadí bez vědomí uživatele. Spravování jednotlivých procedur by mělo být přehledné a co nejjednodušší.

1.3 Struktura

Práce nejprve přiblíží použité technologie, jimiž je programovací jazyk Ruby, Framework[1] Ruby on Rails, dále také již zmíněné nástroje Selenium či PhantomJs, následně bude práce obsahovat analýzu, využití, efektivnost těchto nástrojů v dané aplikaci a v neposlední řadě také nedostatky, chyby a věci, na kterých by bylo potřeba do budoucna zapracovat.

[1] Framework je softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů.

1.4 Předpoklady a omezení práce

Předpokládá se, že uživatel bude mít alespoň minimální znalosti v oblasti informačních technologii, dále by pak měl mít přehled o jednotlivých selektorech, díky kterým jednotlivé skripty fungují. Selektory, aneb prvky, pomocí kterých říkáme procesům, se kterými elementy daný skript pracuje, jsou prvkem celé práce. Uživatel by tedy měl vědět, jak získat informace o daném elementu, ať už se jedná o CSS[2], XPath[3] či pouze identifikátor nebo třeba název.

Práce nebude schopna vytvářet automatizace pro mobilní zařízení, ačkoliv kapacita nástroje Selenium WebDriver tuto možnost pokrývá. Nástroj jako takový, byl konstruován hlavně k testovacím účelům, zatímco v této práci se používá k automatizování postupů podle vlastní iniciativy na jakékoliv webové aplikaci. Díky tomu vzniká několik problémů, jejichž řešením se bude práce zabývat.

1.5 Výstupy a přínosy práce

Výstupem bude aplikace, která plně využívá nástroj Selenium WebDriver a webových ovladačů, dovoluje správu uživatelů a také kategorizaci jednotlivých procesů. V základním provedení budou ukázky jednoduchých automatizací, které jsou použité na reálných webových aplikacích.

Přínosem práce je tedy možnost vyvíjení vlastních procesů skrze aplikaci, která všechny výše zmíněné prvky kombinuje a dává uživateli jedinečnou možnost rozhodnutí, co tento projekt bude dělat, ať už se jedná o testování firemního softwaru, odesílání notifikačních zpráv, odesílání plateb a vlastně všechno, co dokáže uživatel nastavit. Jako další přínos by bylo dobré podotknout, že se nástroj Selenium dá použít i k jiným účelům, než je pouhé testování aplikací, pro které byl původně vytvořen.

[2] CSS aneb kaskádové styly jsou kolekcemi metod pro grafickou úpravu webových stránek. Kaskádové, protože je možné vrstvení jednotlivých stylů.
[3] XPath je dotazovacím jazykem v rámci XML souborů, který umožňuje v jednoduché syntaxi vybrat určité uzly nebo hodnotu z dokumentu XML [6]

2 REŠERŠE ZDROJŮ

Tato kapitola se zabývá použitými zdroji na danou problematiku práce. Hlavním cílem bylo získat co nejpřínosnější výklad jednotlivých technologií. Nutno podotknout, že publikací zabývajících se podobnou problematikou je veliké množství a byly vybrány pouze některé z nich.

2.1 Literární publikace

První literární publikací je zahraniční literatura, která se zaobírá fungováním použitého programovacího jazyka. V knížce je popsán postup od úplných základů, až po komplexní aplikace využívající různorodých rozšíření. ***Agile Web Development with Rails 5*** (Ruby, Thomas & Hansson, 2016) je knihou vhodnou jak pro začátečníky, tak i pro pokročilé. Díky ní, je programátor schopen vytvořit kvalitní aplikace, které využívají nejnovější vlastnosti programovacího jazyka Ruby.

Další kniha je zaměřena na nástroj Selenium, všechny jeho možné implementace a použití. ***Instant Selenium testing tools starter a short, fast, and focused guide to Selenium Testing tools that delivers immediate results*** (Gundecha, 2013) je podrobným manuálem k nástroji Selenium, pomocí něhož je uživatel schopný nástroj implementovat, zprovoznit a využívat jeho plného potenciálu. Samotná kniha popisuje většinu z dostupných implementací a postupů a zabývá se hlavně automatickým testováním.

2.2 Internetové publikace

V dnešní době je internet přehlcen informacemi, které se při stejném zaměření, mohou rozcházet v názorech, proto se volba publikace stává obtížnější a pravda v leckterých případech leží někde mezi jednotlivými články.

Část internetových zdrojů, které budou v následující části zmíněny, byla využita k samotnému vyvíjení aplikace. Zbytek byl následně použit pro shromažďování informací o fungování jednotlivých technologií, nebo principech, na kterých jsou dané nástroje založeny.

Nejdůležitějšími internetovými zdroji při tvorbě aplikace byli rozhodně stránky *SeleniumHQ* a *Watir*. *SeleniumHQ* je domovskou stránkou celého nástroje Selenium. Pomocí ní je uživatel schopen implementovat tento nástroj do své aplikace a zjistit jeho veškeré funkce. Na stránkách lze dohledat obsáhlá dokumentace, díky níž se orientace v této technologii stává mnohem jednodušší. *Watir* se věnuje propojení nástroje Selenium s jednotlivými webovými ovladači. Obsahuje rozsáhlou dokumentaci, díky které je uživatel schopen ovládat webový

prohlížeč skrze nástroj Selenium. Díky informacím z této stránky, byla implementace ovladače, jenž by provázal nástroj Selenium s jednotlivými ovladači prohlížečů, o mnoho snazší.

Další zajímavou stránkou je rozhodně *PhantomJS*. Jedná se o zdroj popisující tento bez-oknový prohlížeč. Vzhledem k tomu, že tento prohlížeč není tak známý a je především určený pro testovací účely, je to také znát po vývojářské stránce. Vývojářská komunita není tak silná a samotný nástroj skrývá nejeden problém. Proto bylo hodně důležité procházet si jednotlivé dokumentace, aby byly získány všechny potřebné a aktuální informace.

V neposlední řadě by bylo správné se zmínit o internetové stránce *Zdroják*. Je to stránka zaměřená na publikování technologických článku. Samotné články jsou orientovány na problematiku jednotlivých technologií. Uživatel je díky těmto publikacím schopen pochopit zdánlivě obtížná témata a získat o nich základní přehled. Většina autorů na této stránce patří mezi zkušené vývojáře, a díky jejich poznatkům lze objevit nové úhly pohledu na danou problematiku, či úplně odlišné funkcionality nástrojů.

3 PROGRAMOVACÍ JAZYK RUBY

3.1 Charakteristické rysy

Jedná se o objektově orientovaný skriptovací jazyk, jehož interpret je napsaný v programovacím jazyce C. Ruby se v jistých směrech velice podobá programovacímu jazyku Perl, avšak je nutno říci, že je o mnoho elegantnější. Syntaxe tohoto jazyku disponuje svou jednoduchostí, tzn. téměř žádné složené závorky, prakticky nulový výskyt středníků. Většinu těchto občas chaotických detailů, jež jsou pro některé programovací jazyky specifické, Ruby postrádá, a tím se řadí mezi programovací jazyky, které jsou proslulé svou jednoduchostí. Uživatel je schopen napsat obsahově velice mocný kód, přičemž výsledek nakonec vypadá jednoduše a kompaktně.

3.2 Ruby on Rails

Nejpoužívanějším rozšířením pro Ruby je Framework Ruby on Rails, dále jen RoR. Slouží k vývoji webových aplikací. RoR se snaží vývojáře vést co nejjednodušší a nejrychlejší cestou k cíli, aby výsledná aplikace splňovala veškerý svůj potenciál a očekávání. Tvůrci tento postup výslovně doporučují. Jakmile si vývojář tento přístup osvojí, zjistí, že je mnohem produktivnější.

3.3 Architektura MVC

RoR využívá ve svém jádře architekturu Model, View, Controller. Hlavní přednost této architektury spočívá v organizaci struktury kódu. Je to dáno tím, že každá funkčnost webové aplikace se ukrývá právě v předem definované struktuře a proto je celková čitelnost nebo hledání jisté funkcionality mnohem jednoduší.

3.3.1 Modely (models)

Jedná se o reprezentaci dat v aplikaci a určování pravidel po jejich manipulaci. Pro daný model existuje odpovídající tabulka v databázi. Model je prostředníkem mezi databází a pohledem. Jedná se o objektovou reprezentaci databáze, kde má uživatel možnost specificky uskupit informace podle různých kritérií, jako je například řazení podle určitého atributu, filtrování záznamů podle různých podmínek, mimo jiné lze také nastavit validace dat, které se vždy kontrolují před vložením nového záznamu do databáze.

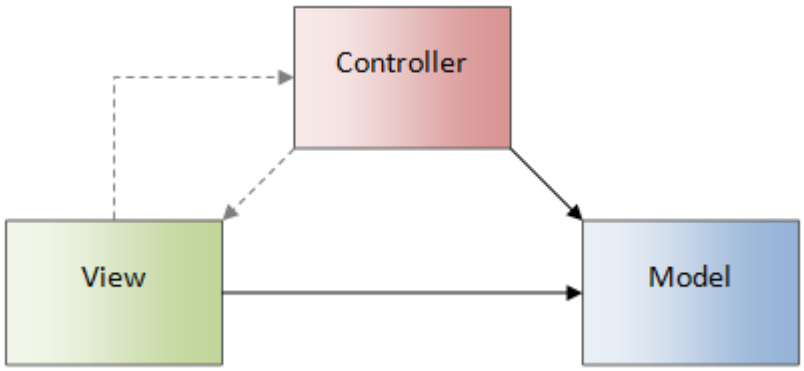
3.3.2 Pohledy (views)

Pohledy, jak už název napovídá, se starají o formu zobrazení dat uživateli. V této sekci se definuje vzhled a zobrazování informací daných modelů, obecně to znamená, že uživatel zde

již má obdržená veškerá data, a je už pouze na něm, jak je zobrazí uživateli. Většinou je to tak, že pro každou akci kontroléru, je pohled, který si uživatel nadefinuje.

3.3.3 Kontroléry (controllers)

Kontroléry fungují jako most, mezi modely a pohledy. Slouží k zpracování požadavků, které přicházejí z webového prohlížeče, získávání dat z modelů a jejich následně odeslání do pohledů. Základem RoR kontroléru většinou bývají základní funkce, které jsou známé pod zkratkou CRUD. Jedná se o zkratku čtyř základních operací, kterými kontrolér může disponovat a těmi jsou: *vytvoř (create), čti (read), aktualizuj (update)* a *smaž (delete)*. Akce „vytvoř“ slouží pro vytvoření nového záznamu v databázi na základě předávaných hodnot. Funkce „čti“ je většinou používá pro přehled všech záznamů pro daný model, třeba tabulka uživatelů. „Aktualizuj“ je akcí pro předání zadaných hodnot do stávajícího záznamu v databázi. Velice často se v RoR navíc definuje funkce „uprav“, která vyplní formulář hodnotami požadovaného jednotlivého záznamu. Tato akce většinou předchází funkci „aktualizuj“, která provedené změny na záznamu odešle a aktualizuje v databázi. A nakonec akce „smaž“, která vymaže daný záznam z databázové tabulky. Samozřejmě vývojář má možnost tyto akce rozšířit, pozměnit, nebo třeba úplně vynechat. Mimo jiné je také možné nadefinovat úplně vlastní akce. Fungování architektury MVC by mohla být jasnější z tohoto Obrázek 1.



Obrázek 1 – Naznačení vazeb v architektuře MVC [1]

3.4 Knihovny

Programovací jazyk Ruby, jako většina programovacích jazyků, využívá knihoven, které rozšiřují aplikaci o různorodé nástroje a funkce. Knihovna se v rámci Ruby nazývá gem. Seznam využitých knihoven pro danou aplikaci se uchovává ve speciálním souboru s názvem Gemfile. Tento soubor tedy popisuje závislosti pro daný program. Specifikuje se název, verze a specifikace dané knihovny. Gemfile by měl být vždy umístěný v kořenovém adresáři

aplikace, protože na tomto místě ho očekává správce balíčku, jež má na starosti instalaci daných knihoven. Například je zde možnost určit pouze testovací sadu gemů, které budou načítány pouze v rámci testů. Pokud je soubor nastavený dle potřeby, stačí skrze konzoli zahájit automatickou inicializaci, která nainstaluje všechny knihovny dle konfiguračního souboru. Poté jsou všechny rozšiřující prvky dostupné a připraveny k použití.

4 POUŽITÉ TECHNOLOGIE

Potřebná rozšíření, která jsou nutná pro implementaci aplikace, lze u RoR, podobně jako u jiných technologií, přidat pomocí knihoven. Poté již pouze závisí na umu vývojáře, jak dané technologie využije.

4.1 Selenium

4.1.1 Charakteristika

Selenium je framework, který slouží pro automatizaci webových prohlížečů. Poskytuje řadu nástrojů a rozhraní určené pro automatizaci interakce uživatelů s HTML nebo JavaScript webovými aplikacemi skrze prohlížeče, jako jsou, Internet Explorer, Mozzila Firefox, Google Chrome, Safari a mnoho dalších. Selenium však nepodporuje práci s takzvanými bohatými internetovými aplikacemi (rich internet applications), které využívají technologie, jako jsou Silverlight, Flash, nebo JavaFx. [2]

Selenium nabízí následující tři nástroje pro automatizaci interakce s prohlížeči:

1. Selenium IDE,
2. Selenium WebDriver,
3. Selenium Standalone Server.

Obsah práce se bude výhradně zaobírat nástrojem Selenium WebDriver.

4.2 Selenium WebDriver

Hlavním páteřním nástrojem této práce je Selenium WebDriver. Jedná se o nástroj, který umožňuje aplikaci přijímat a zpracovávat přímé pokyny pro webové prohlížeče. Jedná se o jednodušší skriptovací jazyk, který dokáže manipulovat s prohlížečem podle vůle uživatele. Nejčastěji je využívaný k automatickému testování webových aplikací. Předpokladem pro úspěšné využívání tohoto nástroje, je využívání API[4] požadovaných prohlížečů. Všechny dostupnější prohlížeče své ovladače volně sdílejí, takže není těžké se k nim dostat.

Nejjednodušším příkladem použití je automatické testování webových aplikací. Je nutné nakonfigurovat aplikaci tak, aby dokázala inicializovat nástroj Selenium WebDriver, který za pomocí webového ovladače dokáže přijímat instrukce a následně je vykonat.

[4]API – označuje v informatice rozhraní pro programování aplikací. Lze hovořit o sbírce procedur, funkcí, tříd či protokolů nějaké knihovny, které může programátor využívat.

Poté již stačí vytvořit funkční skript, který přesně určuje jednotlivé úkony a kroky napříč aplikací, například:

1. Otevřít webový prohlížeč,
2. přejít na stránku s webovou aplikací,
3. přihlásit se za pomoci zadaných údajů,
4. otestovat implementované funkce aplikace,
5. po úspěšném dokončení odhlásit uživatele a zavřít prohlížeč.

Výše zmíněná struktura skriptu spadá do té nejzákladnější, ale je z ní patrné, že hlavní výsadou tohoto nástroje je možnost simulování chování skutečného uživatele aplikace bez nutnosti jakékoliv lidské interakce.

4.2.1 Implementace

Obecně tedy platí, že framework Selenium je pro tvorbu automatického testování webových aplikací, ale potenciál má mnohem bohatší.

Práce pojednává o využití nástroje Selenium WebDriver pro tvorbu automatických webových skriptů v reálném čase pro téměř všechny dostupné webové aplikace skrze internetové připojení. V praxi to vypadá tak, že uživatel má možnost opakující se činnost převést do skriptu, který bude daný úkon provádět za něho. Je tedy možné například vytvořit skript, který bude z jedné webové aplikace získávat data, následně je využívat v úplně jiné aplikaci a ve finále tyto výsledné informace získané kombinací dvou nezávislých programů, využít v konečném úkonu v dalším webovém nástroji.

Jako interaktivní příklad může být například fakturace. Nejdříve se zjistí počet odpracovaných hodin, následně se spočítají náklady na finanční kalkulace a v poslední fázi dojde k vyplnění faktury podle předem definované šablony získanými daty a k jejímu odeslání. To vše automaticky bez nutnosti lidského faktoru.

4.2.2 Nedostatky

Jako většina ostatních technologií, i Selenium se potýká nedokonalostmi.

Hlavní nevýhodou je absence nástrojů, které by umožňovaly práci s bohatými webovými aplikacemi, jež využívají technologie, jako je například Flash nebo Java. Nicméně v dnešní době internet nabízí obrovské množství alternativ, v podobě jiných webových aplikací, které mohou mít žádané funkce bez nutnosti využívání těchto nepodporovaných technologií. Nicméně se jedná o nepříjemnost, kterou je potřeba řešit.

V neposlední řadě lze také hovořit o neúplném slovníku příkazů, tudíž nemusí být vyhověno některým požadavkům pro skript. Většinou jde o styl psaní webové stránky, kdy se uživatel nemusí dostat k některým prvkům webové aplikace, pro jejich těžko definovatelnou cestu. Nástroj Selenium je například nedokáže rozeznat za pomocí zadaného skriptu a jeho modifikování může být velice obtížně až nemožné.

4.3 PhantomJs

4.3.1 Charakteristika

PhantomJs je bez-oknový internetový prohlížeč. Má zabudované vykreslovací jádro WebKit a veškeré operace probíhají pouze na pozadí. Jedná se tedy o prohlížeč bez displeje. Díky absenci vykreslování webových prvků, rapidně stoupá rychlost zpracování úkonů, které tento prohlížeč obdrží. Slouží především k testování webových aplikací, jejich automatizaci nebo k monitorování síťového provozu. Dokáže také za běhu udělat obrázek právě prováděné akce, aby uživatel mohl získat i grafickou interpretaci dané činnosti. [4]

4.3.2 Implementace

PhantomJs je plně ovladatelný JavaScriptem, je tedy nutné, aby aplikace tuto technologii podporovala. Je-li v aplikaci zprovozněný nástroj Selenium WebDriver, jsou již všechny předpoklady k implementaci toho nástroje splněny. Samotný ovladač pro PhantomJs nepotřebuje zvlášť implementovat, neboť je obsažený v samotném jádře nástroje Selenium WebDriver.

4.3.3 Nedostatky

Aktuálním nedostatkem se jeví rozšíření se zkratkou CSP. Jedná se o funkci, která má zabránit zneužití. Pokud prohlížeč vyhodnotí obsah stránky jako zabezpečený, soukromý, osobní, nebo jinak důležitý, vyhodnocování činnosti se přeruší s chybovou hláškou Content Security Policy, volně přeloženo jako politika zabezpečení obsahu. Nicméně s touto chybou se uživatel potká častěji, než by chtěl. Většina činností, které má tento prohlížeč vykonat na webových aplikacích, využívající zabezpečenou komunikaci, skončí právě touto chybovou hláškou. Pokud se jedná o testování vlastních webových aplikací, lze tento problém obejít pomocí nahrazení informací obsažených v hlavičce stránky. Toto je však nemožné, pokud skripty jsou určeně na webové aplikace třetích stran. Teoreticky by se dala využít verze menší než 2.0, kde rozšíření CSP ještě neexistovalo.

Dalším nedostatkem se jeví nefunkčnost některých skriptů, které bezproblémově fungují na ostatních webových platformách. Tento problém je možné vyřešit změněním skriptu přímo pro PhantomJs, nicméně pokud je zapotřebí mít skript více-platformní, může tento problém být závažný.

5 APLIKACE

5.1 Návrh

Cílem bylo navrhnout aplikaci, jejíž obsluha bude co nejsnazší. Každý uživatel pracuje na vlastním účtu, kde každý bude mít svůj prostor, pro svoje nápady a vlastní skripty. Samotné kategorizování skriptů a jejich přehled by měly být co nejpřehlednější, aby se i při větším množství automatizací uživatel dokázal stále rychle orientovat.

Nejtěžším úkonem bylo navrhnout, jak taková automatizace bude vlastně vznikat. V úvahu přicházely různé myšlenky, podle kterých bylo možné daný problém uchopit. Zprvu se zdálo, že nejjednodušší cestou je interaktivní vývoj, kde by uživatel jednoduše otevřel nové okno a všechny jeho kroky by byly stínovány a převedeny do spustitelného skriptu. Takový postup by byl rozhodně nejpřívětivější, nicméně nakonec bylo rozhodnuto, že implementace tohoto postupu by byla příliš složitá. Dalším řešením, které se zdálo využitelné, bylo takzvané skenování stránky. Dalo by se říci, že uživatel by otevřel cílenou stránku, a algoritmus by prozkoumal její prvky. Následně by stačilo vybrat jeden z žádaných elementů, například tlačítko, a uživatel by dostal možnost výběru, ze všech načtených tlačítek na cílené stránce. V neposlední řadě bylo také diskutováno o přístupu, jenž by znamenal pro uživatele zbytečně složitý postup založený na ručním psaní jednotlivých skriptů. Výsledný cíl by měl zahrnovat vhodný kompromis, k předem představeným postupům.

Také bylo diskutováno o historii jednotlivých automatizací. Je dobré mít na paměti, že pokud by měla být aplikace schopná provozu, by bylo dobré, aby samotné automatizace mohly být prováděny automaticky, v předem stanovený datum a čas, popřípadě s možnou cyklickou opakovatelností. Tento požadavek, souvisí s historií již provedených skriptů. Bylo potřeba, implementovat historii automatizací, kde by byly evidovány jednotlivé automatizace, jejich základní informace, výsledek a v neposlední řadě také čas úkonu.

Dalším požadavkem byla správa osobních údajů uživatele. Tento cíl je jedna ze standardních vlastností každé webové aplikace.

Samozřejmě zbýval prostor i pro ostatní funkce, které by se mohly objevit při samotné implementaci a testování.

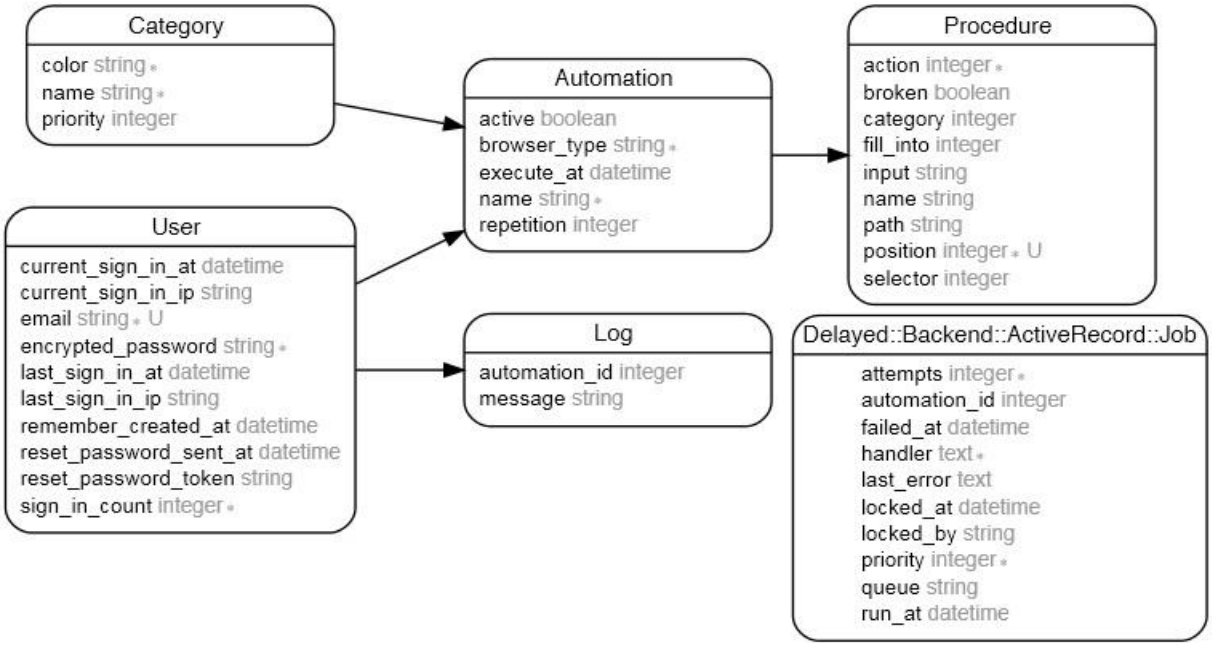
5.2 Implementace

5.2.1 Databáze

RoR umí pracovat s databázemi, jako jsou SQLite3, DB2, MySQL, Oracle Database, Postgres, Firebird, a SQL Server. Pro všechny databáze, kromě SQLite3, je nutné nainstalovat jejich ovladač. Jedná se o knihovny, které RoR používá k implementaci daných databází do aplikace a umožňuje práci s nimi. Ovladače jsou psané v programovacím jazyce C a jsou primárně využívané v jejich distribuované formě. [3]

Tato aplikace využívá databázi SQLite3. Jedná se o databázi, jejíž implementace je nejsnadnější, už jen díky absenci nutnosti ovladače. SQLite3 je vestavěné, relační databázové jádro. Jedná se o bez-serverové databázové SQL jádro s nulovou konfigurací. Jádro SQLite3 není separovaným procesem, místo toho běží v rámci aplikace. [5]

Celá databáze stojí na tabulce uživatelů, kteří na sobě závislé mají ostatní tabulky. Další tabulkou jsou samotné automatizace, každá z automatizací má mnoho procedur, které uchovávají informace o svém úkonu neboli skriptu. Každý uživatel má také svou logovací tabulku, ve které jsou uchovávány informace o uskutečnění daných automatizací. Mezi posledními je tam také tabulka kategorií, která třídí jednotlivé automatizace do skupin. Poslední tabulkou jsou takzvané odložené práce. Jedná se o tabulku, ve které se skladují veškeré úkony, které nejsou závislé na ručním spuštění ve webové aplikaci, nýbrž na jejich časovém určení. Můžeme hovořit o asynchronních akcích, jež jsou naplanované na určitý datum a čas. Tato tabulka uchovává všechny potřebné informace k pozdějšímu zaktivování daných akcí. Následující Obrázek 2 zobrazuje jednotlivé entity a vztahy mezi nimi v rámci databázové vrstvy.



Obrázek 2 – ERD diagram [autor, 2017]

RoR používá vlastní překladač pro volání SQL dotazů, díky tomu je aplikace kvalitně chráněná proti SQL injection (vsunutí cizího SQL příkazu) a následnou nedovolenou změnu na databázové vrstvě. Samozřejmě lze psát i vlastní SQL příkazy, nicméně díky tomu by se otevřela právě hrozba toho, že by uživatel mohl tyto SQL příkazy podvrhnout a například smazat veškerá data. Je tedy doporučeno využívat syntaxi RoR, kde by k této hrozbě nemělo nikdy dojít. [3]

5.2.2 Uživatelské rozhraní

Při tvorbě UI aplikace, byl kladen důraz na dodržení předem stanoveného návrhu, který se v daných směrech podařil dodržet.

Úvodem aplikace je přihlašovací formulář s možností registrace. Po přihlášení je uživatel přesměrován na přehled všech svých automatizací, tento přehled je po čerstvé registraci prázdný.

Další částí je samotná tvorba automatizace, ve výsledku je potřeba alespoň základních znalostí v oblasti informatiky, aby byl uživatel schopen vytvořit plně funkční automatizaci. Celá architektura stojí na jednotlivých, po sobě jdoucích procedurách, které obsahují klíčová slova, k ovládání nástroje Selenium WebDriver.

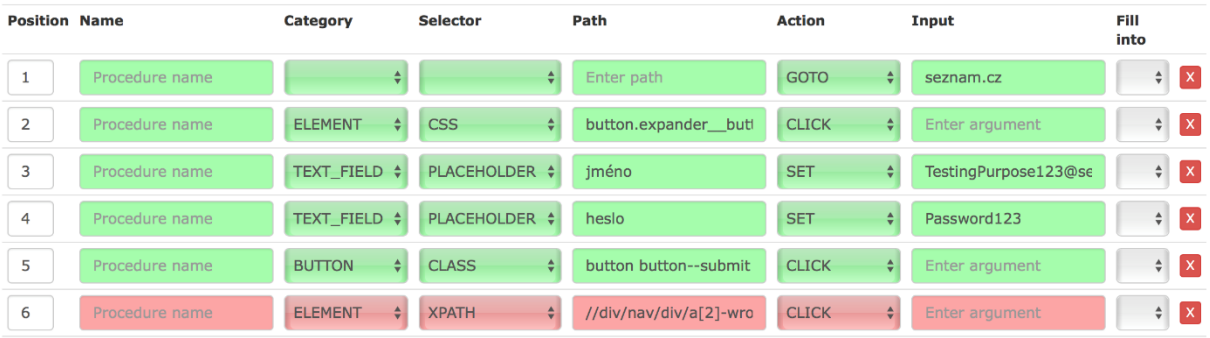
První dvě klíčová slova jsou skládána ze dvou parametrů. Prvním je výběr z předem definovaného seznamu akcí a druhým je manuální vstup, kterým uživatel formuje svůj požadavek.

Prvním z klíčových slov je ***akce***, uživatel má možnost výběru ze seznamu akcí, kterými nástroj Selenium WebDriver disponuje k ovládání webového prohlížeče – *klikni*, *nastav*, *přejdi* a jiné. Uživatel touto akcí specifikuje druh procedury, jestli bude přecházet na jinou stránku, nebo bude nastavovat text nebo třeba bude klikat na nějaký z prvků webové stránky.

Druhé klíčové slovo je ***volič***. Tímto uživatel vybírá, skrze jaký atribut bude procedura hledat prvek webové aplikace. Hovoříme tedy o seznamu parametrů, díky kterým bude daný prvek hledaný, například – *identifikátor*, *název třídy*, *CSS*, *XPath* a podobně. Bude-li uživatel přecházet na jinou stránku, je zřejmé, že toto klíčové slovo není podstatné a bude v proceduře ignorováno. Získat informace, díky kterým se lokalizuje prvek, je onou stěžejní akcí, ke které je potřeba základních znalostí. Uživatel musí vědět, jak získat daný atribut prvku, se kterým chce pracovat, aby ho následně mohl zadat do vstupu.

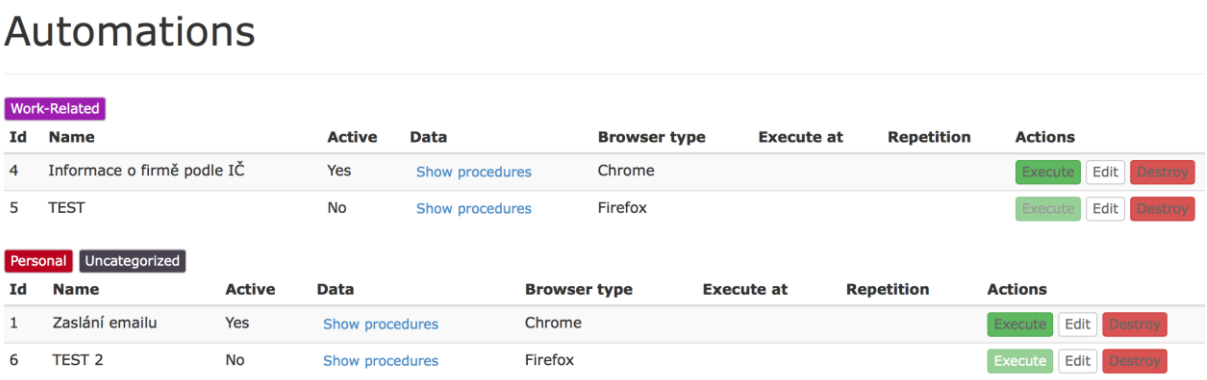
Třetím a zároveň posledním klíčovým slovem je ***kategorie***. Na rozdíl od předešlých klíčových slov, toto disponuje pouze možností výběru bez manuálního zadávání vstupu. Klíčové slovo popisuje, o jaký žádaný prvek se jedná. Jestli se jedná o *tlačítko*, *popisové pole*, *vstup* nebo obecněji *prvek*. Prvek se využívá v případě, pokud žádná z ostatních kategorií nesplňuje požadované kritérium.

Pokud uživatel vyplní a vybere všechny vstupy v jednom řádku, dá se hovořit o plné proceduře, pokud vyplní takto řádků víc, jedná se o automatizaci složenou z několika po sobě jdoucích procedur. Následně je možné automatizaci spustit, pokud vše proběhne v pořádku, dané vyplněné procedury se zbarví do zelena, pokud některá z nich skončí neúspěchem, zbarví se do červena, jak je zřejmé z následujícího Obrázek 3.



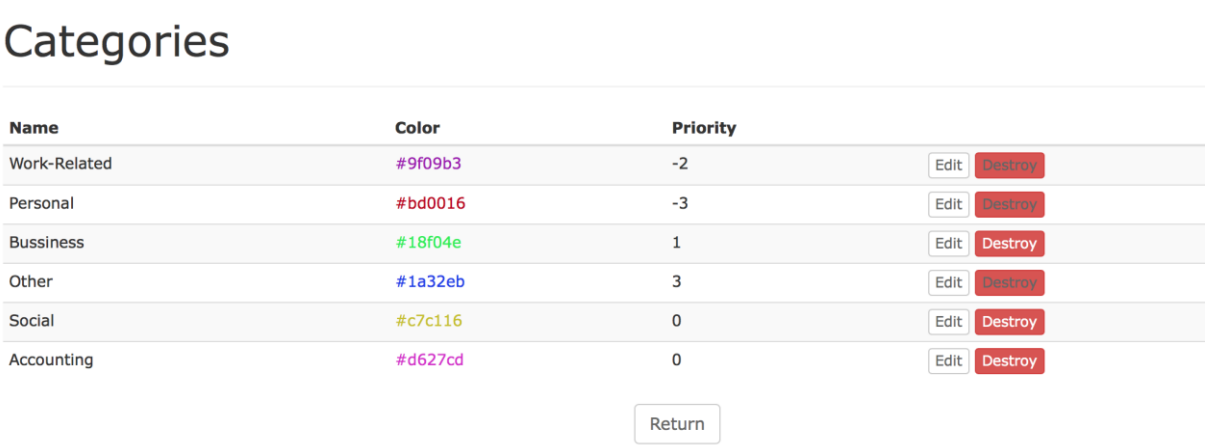
Obrázek 3 – Vzhled formuláře pro tvorbu automatizace [autor, 2017]

Pokud je automatizace hotová a uložená, objeví se v přehledu. U automatizace lze nastavit její kategorii, která je také upravovatelná. U samotných kategorií je možnost nastavovat prioritu, název a barvu, kterou lze vybírat přes interaktivní okno. Při přehledu automatizací, které jsou již roztříděné do kategorií, lze jednotlivé kategorie schovat, což ve výsledku ústí ve větší přehlednost a snazší orientaci. Následující Obrázek 4 slouží pro zhmotnění představy o přehledu automatizací a jejich kategorií.



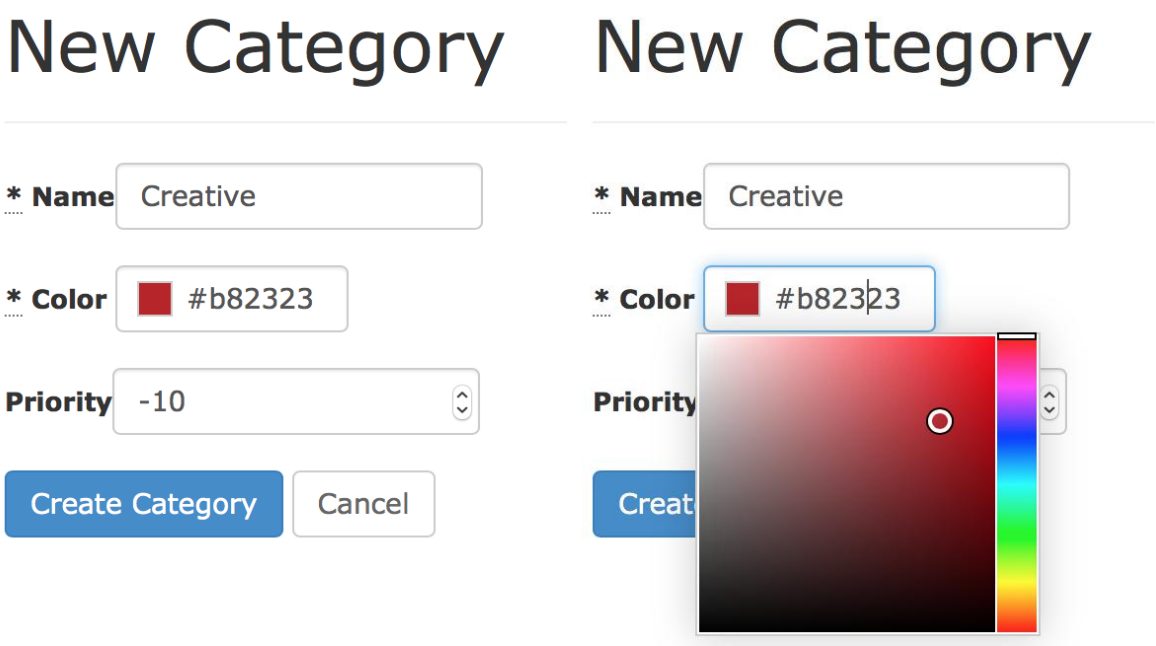
Obrázek 4 – Vzhled přehledu automatizací [autor, 2017]

Samotná správa kategorií má vlastní přehled. Kategorie s názvem ***Uncategorized*** není až tak skupinou, jak se na první pohled zdá. Jedná se o seznam automatizací, jimž nebyla přiřazená žádná kategorie. Zmíněný přehled kategorií zobrazuje nastavenou barvu, respektive její hexadecimální hodnotu, která je zbarvená do příslušné barvy, dále název a prioritu. Výslednou podobu sekce pro úpravu kategorií lze vidět na následujícím Obrázek 5.



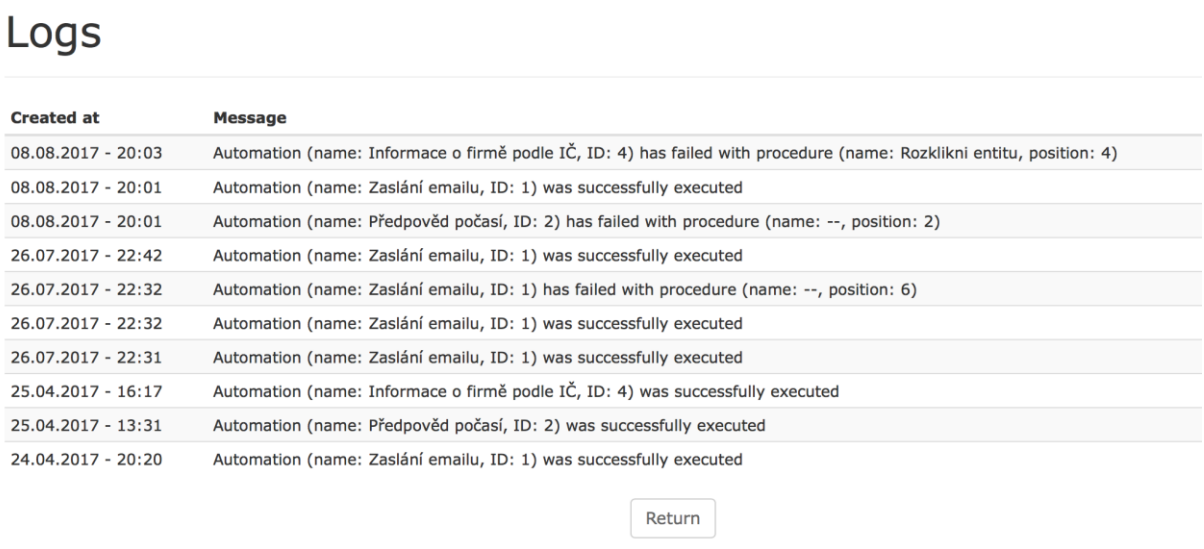
Obrázek 5 – Vzhled přehledu kategorií [autor, 2017]

Při vytváření kategorií si uživatel vybírá barvu z palety barev a následně zadává jméno a prioritu. Čím vyšší priorita, tím výše se daná kategorie zobrazí při přehledu automatizací. Přehled kategorií se řadí podle vytvoření. Příklad vytváření kategorie je znázorněna následujícím složeném Obrázek 6



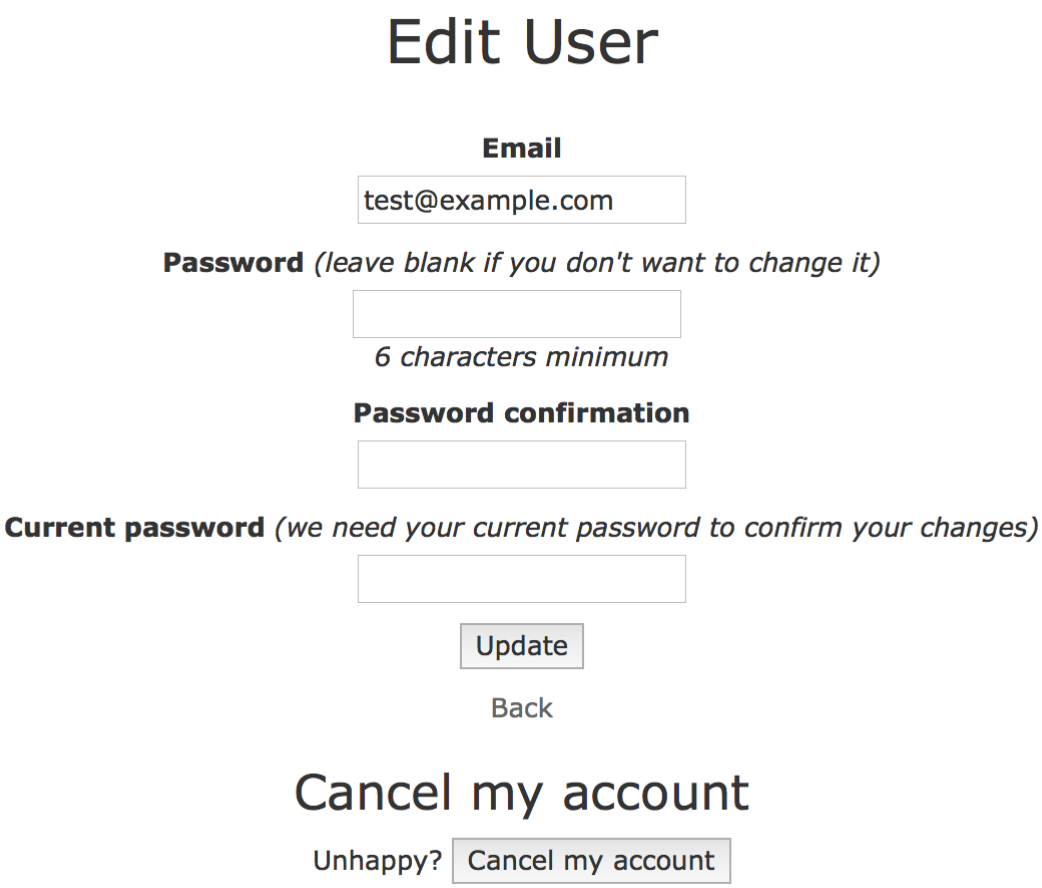
Obrázek 6 – Vzhled při vytváření kategorie [autor, 2017]

Uživatel má následně možnost kontrolovat historii jednotlivých automatizací, ať už se jedná o ručně nebo automaticky spouštěné skripty. Každý řádek obsahuje referenci na danou automatizaci, kde lze jediným kliknutím přejít k úpravě dané automatizace. Pokud nastala nějaká chyba, je hned možné zjistit, kde k ní došlo a hned ji opravit. Samotná historie je řazená od nejnovějších událostí po tu nejstarší. Přehled záznamů operací znázorňuje následující Obrázek 7.



Obrázek 7 – Vzhled zaznamenaných operací [autor, 2017]

Poslední věcí, kterou má uživatel k dispozici, je možnost úpravy svých osobních údajů, které zadával při registraci, nebo zrušení svého účtu. K těmto akcím je uživatel povinen zadat heslo pro ověření jeho totožnosti. Vzhled tohoto formuláře lze vidět na Obrázek 8 pod tímto odstavcem.



Obrázek 8 – Vzhled uživatelského formuláře [autor, 2017]

5.3 Nedostatky

Největším nedostatkem a zároveň největší předností aplikace, je samotné tvoření skriptů. Jde o to, že pro některé uživatele může být aplikace nesrozumitelná až neovladatelná. Může se tak stát právě neznalostí technologií, které jsou potřeba k tvorbě skriptu. Především orientace v selektorech. Samotným řešením by mohl být návod, jak aplikaci používat, s tvorbou základního skriptu, kde by uživatel pochopil princip aplikace a jak získat jednotlivé části pro skládání skriptů.

Jako další problém je nutno brát v potaz možnost zneužití. V rámci testování webových aplikací se možnost zneužití nevyskytuje. Nicméně v rámci praktické části této práce, se již

potenciál zneužití objevuje. Může se jednat o nekonečně opakující se skript pro vyplňování dotazníků, zahlcování stránek, registrace podvodných účtů a podobné. Většinu těchto hrozeb lze ale ošetřit pomocí kontrolních otázek přímo na jednotlivých službách. Samotná aplikace by mohla být ohrožena zahlcením, teoreticky lze vytvořit skript, který v jeho náplni bude definovat zakládání nového skriptu a jeho okamžité spuštění, v takovém případě by samotná aplikace po čase mohla přestat odpovídat nebo se úplně zhroutit. Řešením by mohlo být bezpečností opatření, které by kontrolovalo obsah jednotlivých skriptů se slovníkem zakázaných slov, kde by mimo jiné byla i adresa samotné aplikace.

6 DALŠÍ MOŽNÁ ROZŠÍŘENÍ

Při tvorbě samotné aplikace padlo několik nápadů, které by do budoucna mohly být implementovány. Jedním z těchto nápadů byla možnost sdílení jednotlivých automatizací. Jedná se o zajímavý nápad, který by měl potenciál například v oblasti uzavřených společností, jako jsou firmy, školy, třídy nebo jednoduše mezi jednotlivými uživateli. Iniciativa nápadu čerpá z dnešní doby sociálních sítí. Uživatel by měl možnost se přidat do skupiny, kde by uživatelé mohli publikovat své automatizace a ostatní členové skupiny by si je mohli přivlastnit a následně upravit podle své potřeby. V praxi by to znamenalo, že by jeden z uživatelů stvořil kostru komplexnější automatizace, sdílel ji a ostatní by se mohli od dané kostry odrazit a dotvořit ji podle sebe. Jedná se o zajímavý koncept, který by mohl být produktivní zejména ve firmách, kde se často opakují podobné akce.

Dalším nápadem byla další úroveň komplexity automatizací. To znamená, že uživatel by měl možnost reagovat na události, ke kterým dochází během automatizace. Jednalo by se o sérii podmínek, které by mohly definovat, co by se mělo stát, pokud například jedna z procedur skončí neúspěchem, nebo jiným žádaným výsledkem a na základě tohoto výsledku, by se spustila další série procedur. Toto rozšíření samo o sobě by bylo nutné kvalitně navrhnout, tak aby ovladatelnost aplikace zůstala uživatelsky přívětivá a aby ve finále nepřineslo dané rozšíření pouze kvantum problémů. S tímto rozšířením byla také diskutována možná implementace umělé inteligence, kde by skript, na základě zpracovaných dat, dokázal reagovat na negativní události, jako je například hledání neexistujících prvků, či změna jejich cest a událostem tomu podobných. Nicméně diskutované rozšíření by se obsahově dalo považovat jako další rozsáhlejší kvalifikační práci.

Jedním z dalších nápadů bylo stvořit vlastní překladač proměnných. Ve finále by to znamenalo, že získaná data, by byla uložena někde mimo jednotlivé procedury, pravděpodobně ve vlastní tabulce. A podle uživatelského přání, by daná data byla převáděná na jiné datové typy, nebo by na pozadí mohla s danými daty probíhat žádaná kalkulace či transformace. Nakonec se ale podařilo vytvořit kompromis mezi jednotlivými nápady a to tím, že výstup jedné procedury, se uloží jako vstup do zvolené následující procedury. Tím se podstatně zjednodušil vývoj a žádaná funkcionalita byla splněna.

Jako další rozšiřující možností se zdála být implementace takzvaných zakončovacích nástrojů. Většina skriptů, které byly brány v potaz, většinou končili tím, že se získané informace, v rámci vyhodnocování jednotlivých procedur, odesílali softwarem třetí strany do cílové destinace. Lze hovořit hlavně o emailové podobě, ale i jiných službách. Bylo zamýšleno, implementovat vlastní emailový klient, SMS bránu nebo jiný nástroj určený k podobnému používání. Spojení jednotlivého nástroje se samotným skriptem by bylo dalším problémem, který by byl potřeba vyřešit. Nebylo by nutné využívat veřejných klientů nebo například SMS bran, které mají většinou limitované možnosti. Dále by se také velice zkrátil samotný skript, který je těmito úkony zbytečně prodlužován.

Některé ze zmíněných rozšíření bylo možné implementovat, nicméně bylo uznáno, že jejich složitost by překračovala hranice bakalářské práce.

7 OSTATNÍ AUTOMATIZAČNÍ NÁSTROJE

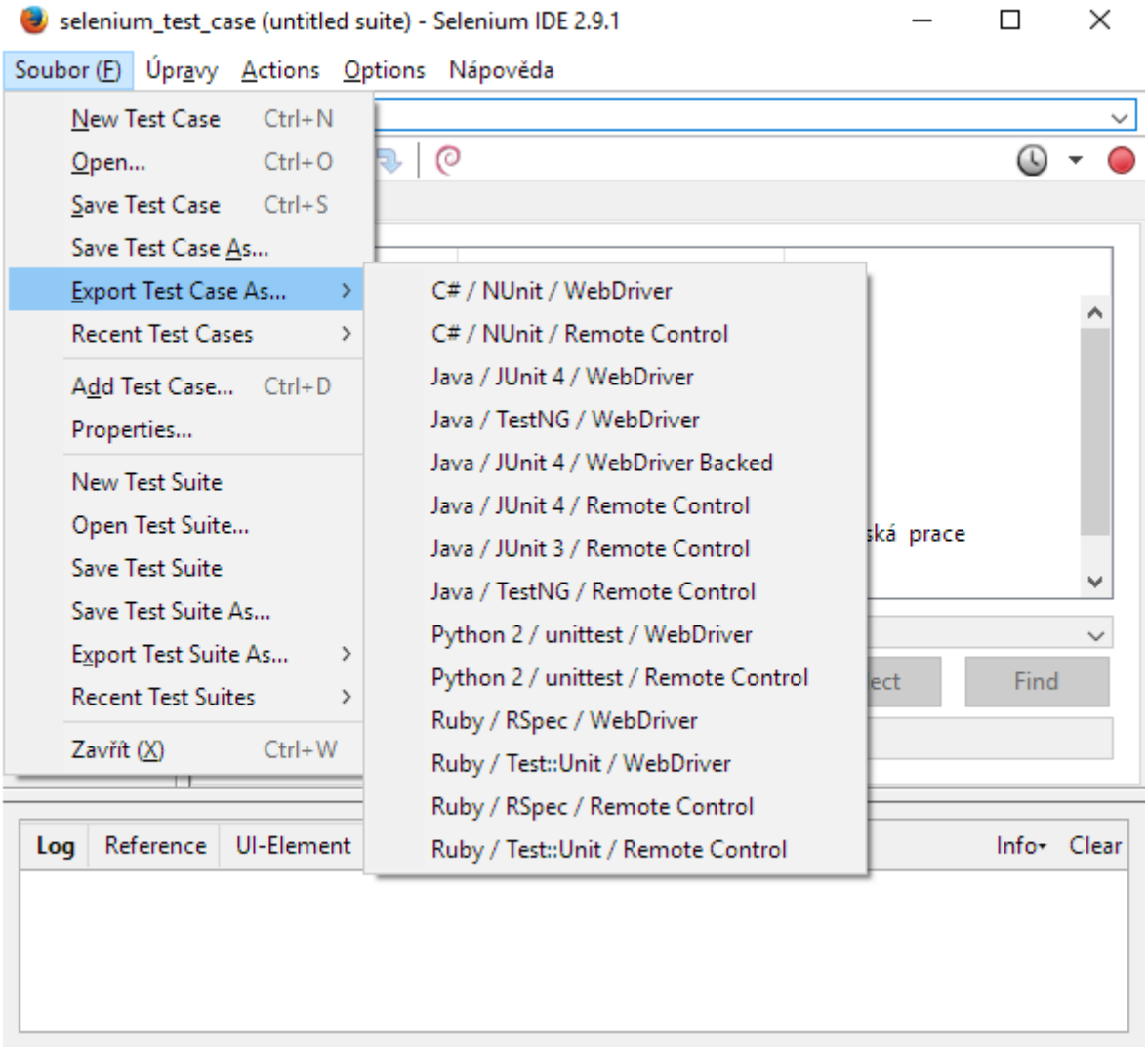
V práci bylo zmíněné, že existuje již nespočet automatizačních nástrojů, které se věnují k automatickému testování aplikací, využité většinou pro interní vývoj. Nicméně také existují aplikace, které mají podobnou funkcionalitu jako aplikace, popisována touto prací. Tato kapitola se těmto aplikacím bude věnovat.

7.1 Selenium IDE

Jako jeden z nejzajímavějších nástrojů byl autorem označen Selenium IDE, což je rozšíření do webového prohlížeče Firefox. Samotná komponenta je vytvářena tvůrci nástroje Selenium. Jedná se vlastně o celou novou vrstvu v prohlížeči, která dovoluje uživateli stínovat pohyb po webovém rozhraní. Například lze díky ní vytvořit komplexní automatizované testy pro vyvíjenou webovou aplikaci. Uživatel si aplikaci pustí na lokálním stroji, otevře si prohlížeč, se zapnutým rozšířením Selenium IDE. Jako počáteční adresu uživatel zadá adresu webové aplikace. Poté již spustí samotné nahrávání interakcí. Uživatel tedy vytvoří scénář akcí, který může následně spustit, tak aby viděl, jaké kroky byly vytvořeny, nebo je mohl modifikovat, přidávat různé mezikroky či naopak jednotlivé akce mazat.

Při jednotlivých krocích lze vidět, jak bylo přistoupeno k cílenému prvku. Je to tedy nejrychlejší cesta k zjištění polohy jednotlivých prvků skrze aplikaci. Zmíněný nástroj se tedy dá efektivně využít k aplikaci, jenž tato práce popisuje.

Nicméně toto rozšíření prohlížeče nabízí více funkcí. Tou nejsilnější z nich je pravděpodobně export daného scénáře do formy automatizovaného testu, ve vybraném programovacím jazyce. K nabídce jsou programovací jazyky Java, C#, Python a Ruby. Navíc lze ještě vybrat, pro jaké testovací prostředí se má daný skript vygenerovat, viz Obrázek 9. [7]



Obrázek 9 – Přehled možných exportů [autor, 2017]

Jak je možné z obrázku vidět, dané exportované testy jsou přímo kompatibilní s nástrojem WebDriver, stačí už pouze vybrat testovací prostředí, skrze které uživatel hodlá automatizovaný test spouštět. Pokud by byl zvolen jazyk Ruby, pro testovací rozhraní RSpec, za pomocí nástroje WebDriver, exportovaný skript by vypadal například jako na Obrázek 10.

```
it "test_a" do
  @driver.get(@base_url + "/SeleniumHQ/selenium/wiki/SeIDEReleaseNotes")
  @driver.find_element(:link, "Features").click
  @driver.find_element(:css, "svg.octicon.octicon-mark-github").click
  @driver.find_element(:link, "Fakulta elektrotechniky a
                       informatiky").click
  @driver.find_element(:link, "Harmonogram akademického roku
                       2016/2017").click
  @driver.find_element(:id, "gsc-i-id1").clear
  @driver.find_element(:id, "gsc-i-id1").send_keys "bakalářská  prace"
  @driver.find_element(:css, "input.gsc-search-button").click
end
```

Obrázek 10 – Ukázka exportovaného skriptu [autor, 2017]

Ve srovnání s aplikací, které je tato práce věnovaná, je nutno podotknout, že stavba automatizovaného skriptu je jednodušší a rychlejší. Nicméně, v porovnání možností, je Selenium IDE hodně limitováno. Ačkoliv jsou skripty rychle vytvořené a snadno modifikovatelné, uživatel není schopen brát hodnoty prvků a používat je v jiných krocích. Samozřejmě k tomu tento nástroj pravděpodobně nebyl nikdy konstruován, naopak byl vyvíjen k rychlému a automatizovanému testování webových aplikací a svůj účel plní. Samotné vyhodnocování skriptů obou aplikací je velice obdobné. Ačkoliv Selenium IDE se jeví více uživatelsky přívětivé.

7.2 Silk Test

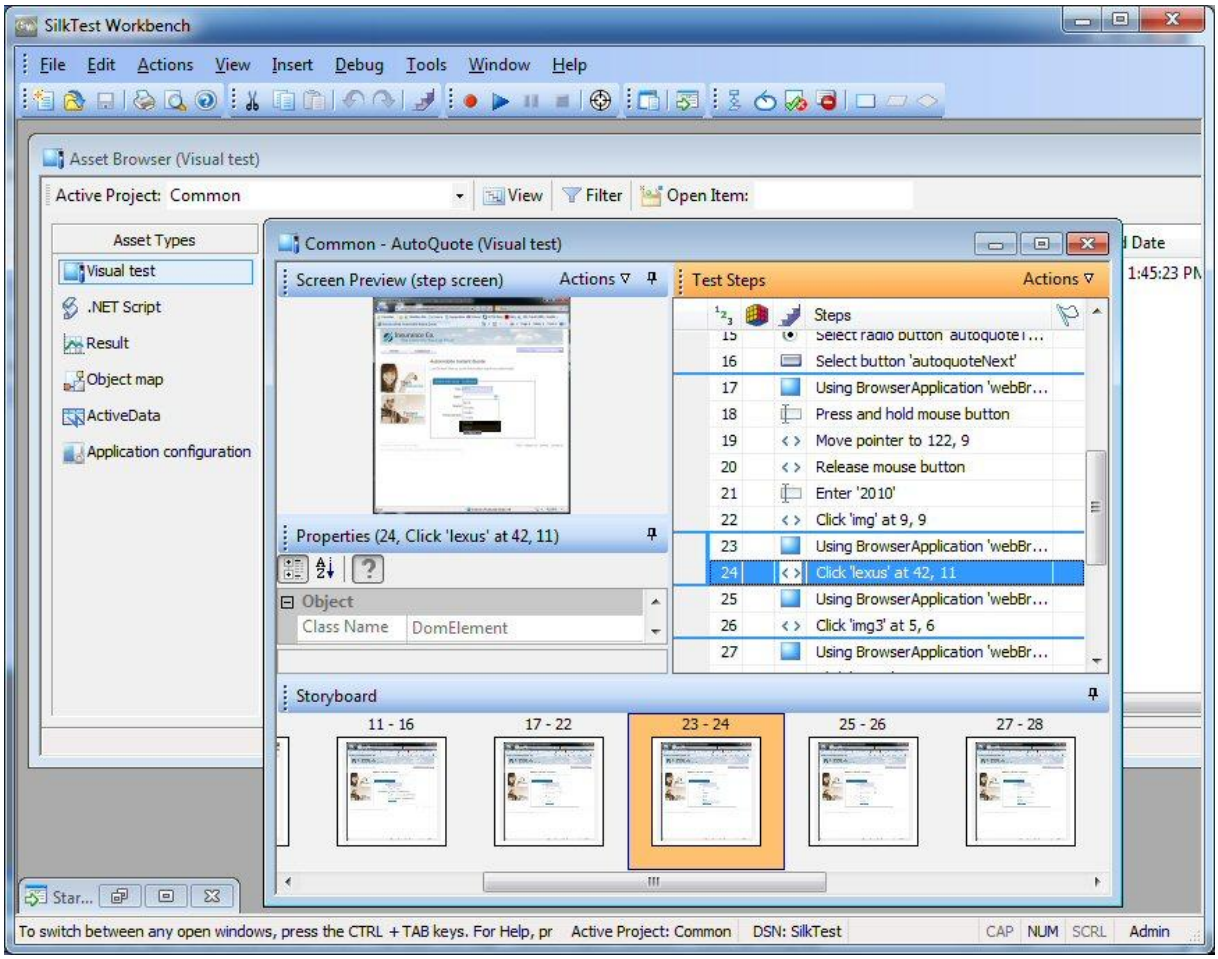
Silk Test, od společnosti MicroFocus, je komerčním nástrojem pro automatické testování široké škály aplikací. Tento software je označován jako jeden z nejspolehlivějších a nejúčinnějších testovacích softwarů. Je určený pro funkcionální a regresní testování aplikací. Jeho předností je možnost tvorby automatizovaných testů bez nutnosti programování.[9]

Tento software je podporuje prohlížeče Internet Explorer, Edge, Firefox, Safari a také mobilní prohlížeče pro iOS a Android. Je možné ho použít na aplikace využívající technologie ApacheFlex, Adobe Air, Silverlight, Java Applets, OracleForms (pouze u IE). Vzhledem k podpoře mobilních prohlížečů je tedy jasné, že podporuje mobilní platformy iOS a Android.

Podporuje Java aplikace (64 bitové i 32 bitové). Dále také .NET technologie nebo například SAP.

Tento typ testování je odlišný hlavně v tom, že není zapotřebí vývojářů k realizaci testovacích scénářů. Business uživatelé tedy bez problémů mohou testovat aplikací a vytvářet automatizované testy bez jakékoliv závislosti na třetí straně. Tím se celý proces zkrátí o enormní časový interval. Testovací fázi nemusí mít na starost samotní programátoři a mohou věnovat čas dalšímu vyvíjení aplikace.

Samozřejmě největší problém je v tom, že se jedná o komerční software, který sice zvládne téměř všechno, ale uživatel si za něj připlatí. Případná cena je řádově v tisících dolarů. Uživatelsky přehledné a přívětivé grafické rozhraní je zárukou lehké tvorby automatizovaných testů, příklad tvorby automatizovaného scénáře na následujícím Obrázek 11.



Obrázek 11 – Ukázka tvorby skriptu v programu Silk Test [9]

7.3 JMeter

ApacheJMeter™ je otevřeným softwarem, stoprocentní čistou Java aplikací, navrženou pro načítání testů, určených k testování funkčního chování a měření výkonu. Původně byl navržený pro testování webových aplikací, ale od té doby se rozšířil o další testovací funkce.[10]

Jedná se o plně vybavené testovací vývojové prostředí, které umožňuje rychlé nahrávání testovacích scénářů, ať už se jedná o webové aplikace nebo nativní aplikace. Je kompatibilní s operačními systémy Linux, Windows a MAC OSX.

JMeter funguje na protokolové úrovni. Co se týče ohledně webových a vzdálených služeb, software se jeví jako prohlížeč, nicméně neprovádí všechny akce podporované prohlížeči. Aplikace zejména nepodporuje interpretaci JavaScriptu, který může nacházet na webových stránkách, a také kupříkladu nevykresluje stránky stejně jako prohlížeč. [10]

Vícevláknové rozhraní dovoluje souběžné vzorkování mnoha vlákny najednou a testování různých funkcí oddělenými skupiny vláken najednou.

Aplikace podporuje testování mnoha aplikací, protokolů či serverů. Z webových služeb podporuje protokoly http a HTTPS za použití technologií jako jsou Java, NodeJS, PHP, ASP.NET a jiné. Dále podporuje komunikační protokoly SMTP, POP3 a IMAP,síťový protokol TCP a datový protokol FTP. [10]

Pro ovládání aplikace využívá grafické rozhraní, u kterého se předpokládá znalost skriptovacích jazyků, jako je například JavaScript. Pokud se uživatel rozhodne nevyužívat grafického rozhraní, může skripty psát v jazyce Java, ve kterém je samotná aplikace psaná.

8 ZÁVĚR

Lidstvo se nachází v době, kdy informační technologie jsou nedílnou součástí života. Jen těžko si lze představit život bez těchto technologií. Využívají se ve všech směrech, ať už pro osobní účely, k pracovním úkonům nebo k vzdělávání. Je téměř nepředstavitelné, že by se v budoucnosti ono propojení člověka s technologiemi přestalo prohlubovat. Autor se domnívá, že jsme uprostřed technologické revoluce, která stále nabírá na síle. To, co je pro většinu lidí nyní nezbytné, může být budoucí generací nahrazeno zase něčím novějším, lepším, efektivnějším. Nicméně neměnným prvkem v tomto cyklu, jsou právě technologie, které již nejspíše navždy budou doprovázet chod světa.

Autorem vytvořená aplikace se v duchu této myšlenky zabývá automatizací. Cílem dnešní doby je snaha vše zjednodušit. Automatizované procesy pomalu nahrazují vše, co je takovým procesem nahraditelné.

Cílem bylo vytvořit aplikaci, jež bude schopná vyhodnocovat uživatelem definované skripty, zpracovat obdržená data a ty následně použít dál v rámci automatizovaného postupu. Tento cíl byl ke všem bodům splněn. Byl kladen důraz na co nejjednodušší ovládání celé aplikace, nicméně při samotném tvoření skriptů se uživatel neobejde bez základních znalostí v oblasti informačních technologií. V rámci základní verze, jsou součástí dvě automatizace, které mohou být pro uživatele výchozím bodem, ze kterého může čerpat inspiraci, jak takové skripty tvořit. Aplikace byla vytvářena s ohledem na přehlednost a dostatek možností nastavení, mezi které patří kategorizování automatizací, nastavování odloženého spuštění, opakovatelnost procesů, zaznamenávání všech spuštěných skriptů a pár dalších standardních nástrojů.

Teoretická část práce byla rozdělena do několika na sebe navazujících kapitol, jež mají za úkol přiblížit čtenáři použité zdroje, programovací jazyk, ve kterém byla aplikace vytvořena a využité technologie. Tato část se snaží čtenáři nastínit veškeré stěžejní informace o využitých nástrojích v této práci. Po seznámení s touto částí, se práce dostává k představení samotné aplikace. Nejdříve se soustředí na její návrh, tak jak byla původně zamýšlena a s jakými předpoklady se autor pustil do jejího vyvíjení. Následně se dílo dostává k samotné implementaci. Daná kapitola mimo jiné popisuje jednotlivé části aplikace dopodrobna a naznačuje jejich možnost použití. Díky obrázkům v této kapitole má čtenář možnost vidět, jak aplikace ve skutečnosti vypadá. Poslední věci v této sekci jsou nedostatky aplikace, na které autor během vyvíjení a testování narazil.

Další část se věnuje možnostem rozšíření aplikace. Během vývoje bylo zaznamenáno několik zajímavých nápadů, které by opravdu mohly mít dopad na celkový dojem z výstupu praktické části práce. Nicméně některé nastíněné funkce byly z hlediska implementace časové náročné a jejich vývoj by překračoval hranice této práce. Pokud tento program bude mít realizovatelnou budoucnost, tato rozšíření by byly mezi prvními, kterým by se autor věnoval.

Poslední kapitola se věnuje ostatním automatizačním nástrojům, které jsou na trhu dostupné. Z této kapitoly je zřejmé, že se veškeré rozhodování při volbě řešení automatizace odvíjí od financí. Pokud uživatel hodlá využít automatizačních služeb, měl by si dobře rozmyslet, jaké jsou jeho možnosti a limity. Nejlepší cestou se zdá být vlastní implementace, která by pokrývala přesně dané požadavky na funkcionalitu aplikace. Nemusí tomu tak být, ve výsledku se může stát, že čas a prostředky věnované na vývoj, mohou enormně přerůst náklady, za které by si uživatel mohl pořídit již zcela funkční software s podobnou, ne-li stejnou funkčností.

9 POUŽITÁ LITERATURA

[1] – *Úvod do architektury MVC* [online]. 7. 5. 2009 [cit. 2017-07-17]. ISSN 1803-5620. Dostupné na: https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[2] – **GUNDECHA, Unmesh**. *Instant Selenium testing tools starter a short, fast, and focused guide to Selenium Testing tools that delivers immediate results.*, Birmingham, U. K:Packt Publishing, 2013, 53 s. ISBN 978-1-68050-171-1.

[3] – **RUBY, Sam, Dave THOMAS a David Heinemeier HANSSON**. *Agile Web Development with Rails 5.*, Texas: Pragmatic Bookshelf, 2016, 467 s. ISBN 978-1-78216-514-9.

[4] – *PhantomJS* [online]. 23. 1. 2017 [cit. 2017-07-17].
Dostupné na: http://phantomjs.org/

[5] – *About SQLite* [online]. 1. 8. 2017 [cit. 2017-07-22].
Dostupné na: https://www.sqlite.org/about.html

[6] – *XPath – rychle to najdeme* [online]. 2. 11. 2009 [cit. 2017-07-15]. ISSN 1803-5620.
Dostupné na: https://www.zdrojak.cz/clanky/xpath-rychle-to-najdeme/

[7] – *SeleniumHQ – Browser Automation* [online]. 27. 7. 2017 [cit. 2017-07-28].
Dostupné na: http://www.seleniumhq.org/projects/ide/

[8] – *Watir – Powered by Selenium* [online]. 25. 7. 2017 [cit. 2017-07-28].
Dostupné na: http://watir.com/docs/home

[9] – *SilkTest* [online]. 2017 [cit. 2017-08-01].
Dostupné na: https://www.microfocus.com/products/silk-portfolio/silk-test/

[10] – *JMeter* [online]. 28. 7. 2017 [cit. 2017-08-01].
Dostupné na: http://jmeter.apache.org/

10 PŘÍLOHY

Příloha A – Zdrojový kód vyhodnocovacího nástroje evaluator.rb

```
module Automations
  class Evaluator
    attr_reader :procedure, :browser, :broken

    def initialize(procedure, browser)
      @procedure = procedure
      @browser = browser
      @broken = false
      @value = ''
    end

    def run
      @value = procedure_evaluation
    rescue
      @broken = true
    ensure
      if procedure.fill_into.present?
        next_procedure = procedure.automation.procedures.find_by(position:
                           procedure.fill_into)
        next_procedure.update(input: @value) if next_procedure
      end
      procedure.update(broken: broken)
    end

    private

    def procedure_evaluation
      if category.present? && input.present?
        browser.send(category, selector, path).send(action, input)
      elsif category.present?
        browser.send(category, selector, path).send(action)
      elsif input.present?
        browser.send(action, input)
      else
        browser.send(action)
      end
    end

    def category
      procedure.category
    end

    def selector
      procedure.to_html_tag(procedure.selector).to_sym
    end

    def path
      procedure.path
    end

    def action
      procedure.action
    end

    def input
      procedure.input
    end
  end
end
```

Příloha B – Zdrojový kód souboru browser.rb

```
module Html
  class Browser
    FIREFOX = 'firefox'.freeze
    CHROME = 'chrome'.freeze
    PHANTOMJS = 'phantomjs'.freeze

    BROWSERS = [FIREFOX, CHROME, PHANTOMJS].freeze

    def initialize(timeout, type)
      @timeout = timeout
      @type = type
    end

    def driver
      @driver ||= case @type
                  when CHROME
                    Selenium::WebDriver.for :chrome, driver_path:
                        Rails.application.config.chrome_path
                  when FIREFOX
                    Selenium::WebDriver.for :firefox, driver_path:
                        Rails.application.config.firefox_path
                  when PHANTOMJS
                    Selenium::WebDriver.for :phantomjs
                  end
    end

    def client
      @client ||= Selenium::WebDriver::Remote::Http::Default.new
      return unless @timeout.present?
      @client.read_timeout = @timeout
      @client.open_timeout = @timeout
    end

    def object
      return unless driver.present?
      @browser ||= Watir::Browser.new driver, http_client: client
    end
  end
end
```

Příloha C – Ukázka zdrojového kódu automation.rb

```
def execute_at_in_future
  return unless execute_at.present? && execute_at < Time.zone.now
  errors.add(:execute_at, "can't be in the past")
end

def queve_execution
  job = Delayed::Job.where(automation_id: id, locked_at: nil).take
  return job.delete if job.present? && execute_at.nil?
  return if execute_at.nil?
  if !job.present?
    delay(run_at: execute_at, automation_id: id).execute
  else
    job.update(run_at: execute_at)
  end
end

def execute
  browser = Html::BrowserBuilder.new(browser_type).build
  executor = Automations::Executor.new(self, browser)
  if executor.run
    next_execution if repetition.present?
  end
  Logs::Logger.new(self).conserve
  executor.close_browser
end

def next_execution
  time = ''
  case repetition
  when 'every_5_min'
    time = execute_at + 5.minutes
  when 'every_hour'
    time = execute_at + 1.hours
  when 'every_day'
    time = execute_at + 1.days
  when 'every_week'
    time = execute_at + 1.weeks
  when 'every_month'
    time = execute_at + 1.months
  end
  update(execute_at: time)
end
```