Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Automatizace testování webových projektů

Plzeň, 2016 Pavel Skala

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 4. května 2016

Pavel Skala

Abstract

Automation testing of web projects

This bachelor thesis deals with automation of functional testing of online stores. Its output is an application allowing the user to create test scenarios that can subsequently be applied in the form of tests for specific online stores. The thesis also introduces the reader to the subject of software testing so that he can gain a thorough overview and understanding of this process. Subsequently, the details of the application are explained to allow the reader to successfully use it.

Abstrakt

Tato bakalářská práce se zabývá automatizací funkčního testování internetových obchodů. Jejím výstupem je aplikace schopná vytvářet testovací scénáře, které se následně mohou aplikovat formou testů pro konkrétní internetový obchod. Součástí práce je také uvedení čtenáře do problematiky obecného testování, aby si o tomto procesu vytvořil přehled. Následně bude čtenář seznámen s vytvořenou aplikací, aby mohl tento nástroj úspěšně využívat.

Obsah

1 Úvod

Vytváření webového obsahu nekončí tím, že dáme všechna média a software dohromady. Přesněji řečeno, práce na webu nekončí nikdy [10]. Se vzrůstajícím počtem aplikací a jejich implementační náročnosti roste také počet vývojářů a s tím i počet vzniklých chyb. Z tohoto důvodů je velice důležité, aby kód, který má být zveřejněn, prošel řadou testů. Tyto testy se mohou dělit do několika kategorií, například podle fáze vývoje, ve které se software vyskytuje, nebo podle toho, kterou část softwaru chceme testovat. Nicméně každé softwarové odvětví se zabývá jinými typy testů. Některé tyto kategorie si v této práci uvedeme.

Kategorie testování, které patří k vývoji webu, jsou například testování funkčnosti, použitelnosti, výkonnosti a bezpečnosti[8]. Už z názvů kategorií testů lze vyvodit, čemu se která skupina věnuje. Popis těchto kategorií si v této práci uvedeme uvnitř kapitoly o testování.

Hlavní prioritou při poskytování služeb zákazníkům je jejich spokojenost s výsledným produktem. Zajištění kvality se dá nejlépe dosáhnout pomocí testování softwaru, tedy zkoušení jeho funkčnosti. Proto, aby bylo testování prováděno rychle a efektivně, bylo nutné tyto procesy zautomatizovat, což nám dovoluje několik nástrojů, které si představíme v kapitole věnující se nástrojům pro automatizované testování webového obsahu.

Jakmile si představíme jednotlivé nástroje, tak nás bude čekat fáze, kdy si budeme muset rozhodnout, jak s těmito nástroji naložíme. Pokud si vybereme jeden konkrétní, či zkombinujeme funkcionalitu z vícero těchto nástrojů. Na základě této analýzy se rozhodneme, jak budeme postupovat při tvorbě samotných testovacích scénářů.

Hlavním cílem práce je vytvořit program, který bude schopen vytvářet vlastní automatizované funkční testy internetových obchodů, které bude možno kdykoli spouštět, aby byla zajištěna jejich základní funkcionalita. Samotná tvorba a spouštění testovacích sad bude popsána v kapitole zabývající se přípravou a spouštěním vlastních testů. Součástí práce musí být také uveden způsob, kterým bude uživatel informován o průběhu testu a jejich výsledcích.

2 Webové aplikace

Předtím než se začneme věnovat testování internetových obchodů, tak si nejdříve představíme několik důležitých termínů, se kterými se u webových aplikací můžeme setkat.

2.1 Technologie pro webové aplikace

Pokud se chceme věnovat vývoji nebo testování webu, pak je naprostou nezbytností něco vědět o základech, ze kterých se web skládá. Webových technologií je velká škála, my si nyní představíme některé z nich.

2.1.1 HTML

Pro tvorbu webových prezentací je naprostou nutností znalost jazyka HTML[1], který reprezentuje data ve webovém prohlížeči. Nejedná se čistě o jazyk programovací, ale jazyk značkovací, jehož kód je tvořen elementy[2], což jsou slova nebo znaky uvnitř lomených závorek [12]. Jednotlivé elementy specifikují prohlížeči strukturu a význam textu, který je obklopený závorkami.

Historie HTML sahá až do roku 1989, kdy vznikla jeho první verze. Vzhled těchto stránek nebyl příliš poutavý, ale už tehdy podporoval hypertextové odkazy.

HTML procházelo svým vývojem a zásadní milník přišel v roce 1998, kdy byla vydána jeho čtvrtá verze, tj. HTML4. S touto verzí došlo ke standardizaci jazyka HTML, se kterou přišla, stejně jako s předchozími verzemi, organizace W3C[3]. Myšlenkou W3C bylo oddělení obsahové části od prezentační do dvou programovacích jazyků. Jazyk pro obsah dokumentu zůstal původní, tj. HTML, a pro část prezentační se vytvořil nový jazyk CSS[4], který si popíšeme dále.

Poslední vydaná verze je HTML5, která je kompatibilní s předchozí verzí. Vydání této verze proběhlo v roce 2014 a přinesla podstatné změny, přičemž mezi nejdůležitější patří přímá podpora přehrávání multimédií v prohlížeči. Tento standard dále přinesl několik nových HTML značek, které zjednodušují a hlavně zpřehledňují zdrojový kód.

[1] HyperText Markup Language
[2] Prvky jazyka HTML
[3] World Wide Web Consortium
[4] Cascading Style Sheet

2.1.2 HTTP

HTTP[5] je protokol pro přenos hypertextových dokumentů přes internet[12]. Přestože hypertextové dokumenty jsou obvykle pouze HTML stránky, tak tento protokol může dále přenášet obrázky nebo další soubory, které internetové stránky požadují. Protokol pracuje jednoduše na systému žádosti a odpovědi, kdy se klient dožaduje po webovém serveru nějaké odezvy. Po každé, když uživatel zadá do svého prohlížeče URL[6] adresu, tak se prohlížeč pokusí získat z tohoto zdroje pomocí uvedeného protokolu data. Pokud server nalezne tento zdroj, pak prohlížeč data obdrží a následně zobrazí.

2.1.3 CSS

Jak procházel jazyk HTML svojí evolucí, tak s nástupem jeho čtvrté verze došlo k velice významnému a pro programátory velmi ulehčující skutečnosti. S nástupem HTML4 došlo k rozdělení zdrojového dokumentu na datovou a prezentační část. Rozdělení webu do těchto dvou jazyků přináší jednu bezespornou výhodu a to zpřehlednění HTML kódu. Další velkou výhodou oddělení těchto dvou částí je ta, že nám dovoluje změnu vzhledu bez zásahu do HTML kódu, což zajišťuje zároveň jistou míru bezpečnosti před nechtěnou záměnou dat. Jednoduše zaměníme jednu sadu CSS stylů za jinou.

Princip použití tohoto jazyka není obsahem této práce. Pokud se zajímáte o tuto problematiku, pak Vás odkáži na následující literaturu [12].

2.2 Druhy webových aplikací

Webová aplikace je služba, která běží na vzdáleném serveru a je přístupná uživatelům prostřednictvím internetu, nebo případně intranetu v síti daného podniku [15]. Největší výhoda webová aplikace se nachází v možnosti jejího použití v internetovém prohlížeči. Proto se stává stále více populární, protože počet chytrých zařízení umožnující procházení webového obsahu roste. Velkou výhodou oproti běžné aplikaci, která je umístěna na konkrétním zařízení je ta, že se v případě potřeby provede její aktualizace okamžitě a na jednom místě. Všechny změny by měli probíhat hladce, nemělo by se stávat, že by systém byl nedostupný, či dokonce nefungoval vůbec. V některých případech je velký problém aktualizovat systém bez toho, aniž by nedošlo k jeho nedostupnosti. Potom se tato aktualizace musí předem naplánovat,

[5]HyperText Transfer Protocol
[6]Uniform Resource Locator

aby o ní uživatelé věděli. Samozřejmostí je, že se tyto aktualizace aplikují v době, kdy je jeho využívání nejméně časté.

Mezi běžné webové aplikace patří například webMail[7], internetové aukce, sociální sítě, elektronické obchody a mnoho dalších služeb.

V případě aplikací, které jsou instalovány přímo na jednotlivá zařízení, se tato vydaná aktualizace musí nainstalovat do všech těchto zařízení. To trvá samozřejmě nějakou dobu. Nejprve se tato aktualizace musí vydat, poté rozeslat mezi uživatele a popřípadě je upozornit, že byla aktualizace vydána. Uživatel si ji následně stáhne a nainstaluje.

Webové aplikace se v zásadě dělí do kategorií podle jejich složitosti:

2.2.1 Webové stránky

Tato kategorie spadá mezi nejjednodušší. Webové stránky dokáže vytvořit každý, kdo zná základy jazyka HTML. Jedná se hlavně o stránky osobní, ať už jde o stránky zobrazující statický obsah, či stránky mnohem více sofistikované využívají moderních technologií, mezi které patří dynamicky generovaný obsah či práci s databázemi. Hlavním smyslem webových stránek je poskytovat světu informace o dané osobě, skupině či společnosti.

2.2.2 Webové aplikace

Webové aplikace mohou obsahovat kromě webové prezentace také logiku, která je srovnatelná s běžnými počítačovými aplikacemi. Na tvorbě takových systému se zpravidla podílí tým lidí. Mezi takovéto systémy se řadí redakční systémy, maily nebo internetové obchody. Rozdíl mezi webovými stránkami a aplikacemi je především ten, že aplikace přináší uživateli navíc nějaký druh služby, kterou po daném systému očekává. Kvalita výsledného produktu závisí na míře testování, která je v případě vyvíjení aplikací velmi důležitá a nesmí se opomíjet. U velmi důležitých systémů je důležité, aby tým testerů byl složen z lidí, kteří mají v dané problematice dobré zkušenosti.

2.2.3 Softwarové systémy s tenkým klientem

V tomto případě se jedná o složité systémy, na jejichž vývoji a testování se podílí řada zkušený vývojářů. Samotný vývoj těchto systémů trvá v řádu měsíců. Tyto systémy jsou velmi komplexní, jsou často udržované, aktualizované a také přináší vysokou míru flexibility. Pro systémy s tenkým klien-

[7]Je to jakýkoliv emailový klient, který je implementovaný jako webová aplikace běžící na webovém serveru.

tem je charakteristické, že výpočty, které se provádějí, jsou vykonávány na straně serveru, a tak nedochází k zatěžování klientského zařízení, které dané výsledky pouze zobrazuje.

2.3 Specifika internetových obchodů

Internetové obchody jsou ve své podstatě webové aplikace, které umožňují prodejcům nabízet jejich produkty a služby veřejnosti. Takový obchod slouží k nabídce a vyhledávání zboží s možností si je objednat pohodlně odkudkoli. Nezbytnou nutností takových obchodů je možnost přijímání objednávek, jejich vyřizování, evidence plateb a poskytování dalších informacích o výrobcích či řešení reklamací. V současné době využívá internet pro svůj nákup stále více uživatelů, protože zde mají možnost si vytvořit větší přehled o cenových nabídkách a pořídit tak požadované zboží za nejlepší cenu. Další výhodou tohoto typu nakupovaní je ta, že se dá provozovat v pohodlí domova bez nutnosti fyzicky daný obchod navštívit, což vede jak k časové, tak finanční úspoře. V oblasti internetového obchodování je využití automatizace funkčních testů velmi vhodné, protože se jedná o poměrně jednoduchou aplikaci. Stejně jako u jiných webových aplikací má zde největší využití testování regresní, které si uvedeme v kapitole o testování. Internetové obchody se od jiných služeb poskytovaných na internetu výrazně neliší, ale vyskytují se zde jistá specifika, hlavně v oblasti bezpečnosti při provádění plateb [13].

3 Základy testování softwaru

Testování softwaru má velice široké pole působnosti, které je v procesu softwarového vývoje nenahraditelné. Jak bylo napsáno výše, typů testování je několik druhů, ale všechny mají společný cíl – spokojeného zákazníka. Na začátku každého vývoje je poptávka od zákazníka, který požaduje nějaký nástroj, který mu bude poskytovat požadovanou službu.

V případě důsledného procesu testování se minimalizuje pravděpodobnost výskytu chyb, jejichž oprava by mnohdy byla finančně náročnější, než samotný proces testů. Dále zde platí, že čím dříve je chyba odhalena, tím jsou náklady na její opravu nižší.

V této kapitole si popíšeme základy testování obecného softwaru, abychom poté mohli určit, jaké typy testů se hodí právě pro nás u testování webového obsahu.

Testování je proces, kdy se skupina vývojářů a testerů snaží vyvinout kvalitní produkt, který by zákazníkovi přinesl prospěch. Pro dosažení dostatečné kvality je nutné plánovat a řídit. Na začátku vývoje je nutné stanovit měřítka kvality. Dále se stanový záběr testování, tj. vyberou se testy, sbírají se data a připravuje se nástroj, který tým k testování potřebuje.

Při testování se potýkáme s několika problémy se kterými se musíme vypořádat. Ron Paton tyto problémy definoval [11] následovně:

- Není možné otestovat kompletně všechny možné případy zadání.
- Je velmi složité odhalit, že bylo ve specifikaci na něco zapomenuto.
- Specifikace není vždy jednoznačná nebo dokonce chybí.
- Testování nikdy nemůže prokázat, že chyby neexistují
- Výsledky testování mohou být ovlivněny špatnou komunikací v týmu nebo se zákazníkem.
- Testování musí být přizpůsobeno testovanému produktu, není možné stejným způsobem otestovat různý software.
- V průběhu testování se mohou stát některé testy neefektivními. Testy je nutné neustále přizpůsobovat měnícím se okolnostem.
- Dvě různé chyby se mohou navenek projevovat stejně, nebo jedna chyba se může navenek projevovat různě.

- Testování je citlivé na vstupní data, ta by měla být realistická. Vygenerování jednotvárných dat není ideální.

Samotné vyhledávání chyb je pouze začátkem cyklu testování. V rámci testování je nutné, aby po nalezení chyb došlo k jejím následným opravám. To znamená, že tester musí napsat testovací report, na který zareaguje vývojář, který následně chybu opraví. Předchozí řádky bych shrnul do následné věty, která je pro testery spíše mottem než definicí:

„Testování je proces hledání chyb za co nejkratší čas a s co nejrychlejším zajištěním jejich oprav.“

Je potřeba také brát v úvahu, že testování softwaru je obecně velmi náročný proces, který zabírá nemalou část celého vývoje a přesto si na jeho konci nemůžeme být jisti, zda bylo testování provedeno dostatečně. Tyto obavy pramení hlavně z toho důvodu, protože možných vstupů, výstupů, průchodů programem je tak velké množství, že není možné otestovat všechny jejich kombinace v reálném a hlavně rozumném čase [11].

„Testování programu může být velmi efektivním způsobem, jak prokázat přítomnost chyb, ale je naprosto nevhodné k prokázání jejich nepřítomnosti.“

3.1 Kategorie testů

Aby došlo k co nejlepšímu otestování vyvíjeného softwaru, pak je nutné dopředu stanovit, jaké typy testů se pro daný produkt hodí. Druhů testů je velké množství [5], a proto si některé z nich představíme. Typy, kterými se budeme zabývat, si představíme podrobněji. Volba správných testů je klíčová, protože může dojít k situaci, kdy nemusíme odhalit velké množství závažných problémů, se kterými vývojáři nepočítali. Oprava takových chyb je poté výrazně dražší, protože jejich pozdní odhalení znamená změnu rozsáhlého množství zdrojového kódu.

3.1.1 Černá a bílá skříňka

Tyto termíny označují způsob, jakým uživatel vidí do zdrojového kódu. Vyskytuje se ve všech druzích testování a rozhodně se nezaměřují pouze na internetový obsah.

Při testování černé skříňky ví tester jen to, co má uvedený software dělat, ale neví, jak daný algoritmus pracuje - dovnitř pomyslné skříňky tedy nemá

možnost pohlédnout. Jestliže napíše některý údaj na vstup, pak jediné co o algoritmu ví, je jeho výstup. Proces transformace vstupu na výstup je tedy pro testera neznámý.

Testování bílé skříňky je charakteristické tím, že je testerovi známa konkrétní implementace daného problému. Ten tak vidí celý zdrojový kód a životní cyklus daného algoritmu – vidí tedy dovnitř skříňky.

Nejsnazší způsob, jak začít testovat, je považovat danou webovou stránku nebo dokonce celý webový server za černou skříňku [11]. V tomto případě nevíme nic o její činnosti, nemáme žádnou specifikaci, máme před sebou pouze webovou stránku, kterou máme testovat.

Dále se vyskytuje případ šedé skříňky, což je kompromis mezi oběma případy. Software testujeme jako černou skříňku, ale zároveň nahlížíme do mechanismů jeho činnosti.

3.1.2 Statické a dynamické testování

Tyto dva způsoby rozdělení vznikly na základě toho, kdy k testování dochází. Pokud testujeme software bez nutnosti jeho běhu, pak se jedná o testování statické. V tomto případě se může například jednat o testování zdrojového kódu

Na druhou stranu dynamické testování může probíhat až tehdy, kdy máme k dispozici spustitelnou verzi programu a probíhá hlavně na základě poskytování různých vstupů a posuzování příslušných výstupů.

Pro názornou ukázku si můžeme uvést analogii při koupi auta, kterou uvedl Ron Patton ve své publikaci [11]:

„Při kontrole ojetého auta zkusíme zabrat za kola, jestli v nich nejsou vůle, podíváme se na stav laku karosérie a otevřeme kapotu - to jsou postupy statického testování. Potom auto nastartujeme, poslouchámě zvuk motoru a absolvujeme zkušební jízdu po normální silnici - to jsou techniky dynamických testů.“

3.1.3 Automatické a manuální testování

Automatizace testování se stává stále více populární, protože přináší možnost rychlého a spolehlivého otestování základních funkcionalit v průběhu celého vývoje. Rozhodnutí, zda přistoupit na automatizované testování nebo ne, záleží na možnostech každé firmy, protože před jeho nasazením se musí počítat s počátečními náklady na jeho zavedení. Jakmile jsou ovšem testy vytvořeny, pak jejich aplikace je finančně minimálně náročná, velmi rychlá a

hlavně naprosto bezchybná. Automatizace se bohužel nedá použít komplexně na celý projekt. Vždy totiž narazíme na případy, kdy se více vyplatí využít manuálního testování. Manuální testování se použije například u testování pokročilého grafického rozhraní, jehož automatizace by byla velmi náročná, nebo nemožná.

Proč potřebujeme automatizaci testů

Mezi hlavní výhody automatizovaného testování patří úspora času i financí. Automatizace umožňuje provádět stejné akce precizně a identicky při každém jeho opakování.

Kdy je automatizace testů přínosná

Na začátku vývoje bychom se měli rozhodnout, které části systému by bylo dobré pokrýt automatizovanými testy a na které části systému by stačili testy manuální. Jelikož automatizované testování přináší nemalé počáteční investice, pak je jeho aplikace přínosná u často opakujících se testů, či komplexních testů, kde se klade velký důraz na jejich preciznost. Nejvhodnějšími kandidáty na automatizaci jsou regresní testy, které ověřují, zda nové funkcionality neovlivnili funkčnost systému [5].

3.1.4 Pozitivní a negativní testy

Jedná se o dva základní postupy, které se v testování vykytují. Jedním z nich je pozitivní a druhým negativní testování.

- **Pozitivní testování**, někdy také nazývané testy splněním ve skutečnosti jen kontrolují, zda software poskytuje základní funkcionalitu. Při tomto testování jsou algoritmům předávány nejjednodušší vstupní parametry, tudíž se aplikaci nepokoušíme „shodit“, ale jen otestovat, zda je její základní funkcionalita korektní.
- V případě **Negativního testování**, nazývané také testy selháním, jsou algoritmům předávána taková data, která mají aplikaci řádně otestovat. Tyto data mají podobu okrajových hodnot, nebo zcela nesmyslných dat, na které musí být aplikace schopna zareagovat a musí být stabilní. U webových aplikací nebo aplikací s grafickým uživatelským rozhraním je velká část negativních testů zaměřena na validaci vstupů, kdy jsou aplikaci předkládána nahodilá data. Tato situace musí být v kódu ošetřena a uživatel musí získat informaci o tom, že je něco špatně.

Ron Patton ve své publikaci [11] předvedl tuto problematiku na následují analogii:

„Pokud dostaneme k otestování prototyp nějakého automobilu, který právě sjel z montážní linky, tak jen stěží vyjedeme s autem na zkušební okruh a budeme testovat jeho limity. Nové auto bude nejspíše vykazovat celou řadu chyb, které se projeví již v malých rychlostech. Mezi tyto chyby může například patřit nesprávná velikost pneumatik, nedostatečně výkonný brzdící systém, anebo nadměrné přehřívání motoru. V tomto případě bychom se nejspíše zabili. Tyto problémy se musí nejdříve najít, opravit a až poté se můžeme pokusit otestovat hraniční možnosti automobilu.“

Proto je nejprve nutné nechat projevit chyby v testech splněním a teprve poté se pustit do testů selháním.

3.2 Fáze testování

Testování lze rozdělit do následujících fází podle toho, v jakém časovém horizontu od napsání kódu se provádí. Této skupině testů se také někdy říká stupně testování [6]

3.2.1 Testování programátorem

Jde o nejjednodušší typ testování a provádí se ihned po vytvoření programového kódu. V praxi se tyto testy nazývají „*testy spuštěním*“. Většinou si však programátor netestuje svůj vlastní kód, ale realizuje se tzv. „test čtyř očí“. To znamená, že kód, který programátor napsal, testuje jiný programátor. Program je na tomto stupni kontrolován na úrovni zdrojového kódu. V praxi je bohužel tento druh testování často podceňován, ikdyž náklady na opravy těchto chyb jsou nejméně nákladné.

3.2.2 Jednotkové testování

Tohoto typu testování se využívá během vývoje samotné aplikace. Hlavním úkolem tohoto typu testování je zajistit, že jednotlivé části programu pracují správně, čímž by mělo být zajištěno, že po konečném spojení všech částí dohromady bude aplikace pracovat korektně. Mezi vhodné testovací jednotky zpravidla patří nejmenší testovatelné části aplikačního programu, což může být samostatná metoda, nebo třída. Tyto testy tedy zajišťují, že nově vytvořený kód je schopen po spojení do většího celku pracovat správně a tudíž nemůže způsobit nestabilitu výsledné aplikace [11].

V těchto jednotkách se prakticky neprovádí nic jiného, než že se na jejich konci porovnává předpokládaný výstupu oproti skutečnému [1]. Vývojář typicky napíše nějakou sadu jednotkových testů pomocí nějakého frameworku[1] podle jeho uvážení. Testy by měli být rychlé a automatizované.

Jednotkové testování se velmi špatně aplikuje na již zaběhlých projektech. U již vytvořených aplikací se většinou musí provést kompletní refaktoring kódu či dokonce hlubší úpravy. Proto je vhodné zabývat se těmito testy již v etapě návrhu aplikace a v této době se rozhodnout, zda budeme tyto testy využívat.

[1] Nástroj, který ulehčuje vývojáři práci.

3.2.3 Integrační testování

Jedná se o typ testování, který má za úkol zjistit, zda je část softwaru schopna komunikovat s jeho zbytkem. Z toho plyne, že pokud chceme nově vytvořenou část programu nasadit do celého systému, pak ji před touto akcí musíme nejdříve řádně otestovat, abychom po jeho nasazení celý systém nedegradovali.

Následné integrační testování však již není v kompetenci vývojáře, ale testovacího týmu. Tento typ testování bývá také nazýván jako „Testy vnitřní inteligence“ [7]. V této fázi se tak testuje vzájemná integrace již otestovaných částí systému. Chyba, kterou bychom odhalili během integračních testů, se zcela jistě projeví v průběhu dalších úrovní testování a jak bylo řečeno dřív, čím dříve se na chybu narazí, tím méně úsilí stojí její náprava. Integrační testy mají svůj význam, nicméně nelze jejich použití nikterak přeceňovat.

3.2.4 Systémové testování

Po ověření správné integrace nastává čas na systémové testování. Tyto testy jsou tedy prováděny v pozdních fázích vývoje, kdy je systém testován jako celek z pohledu zákazníka. Podle připravených scénářů se navrhují kroky, které v praxi mohou nastat. Obvykle se provádějí v několika kolech, ve kterých se nalezené chyby opraví a jsou spuštěny znovu. Jedná se tedy o velice důležitou úroveň testů, která se využívá před předáním systému zákazníkovi a před jeho nasazením. Na realizaci těchto testů by se mělo myslet již v raném stádiu vývoje systému.

Funkční testování

Toto testování je zaměřeno na aplikaci v rámci její řádné implementace. Testují se obecně všechny funkce, které jsou v aplikaci implementovány, a ověřuje se, že fungují správně a že odpovídají požadavkům zákazníka.

V našem případě, kdy chceme provést funkční testování internetového obchodu, může být tento proces prováděn manuálně uživatelem klikáním na všechny možné panely v daném internetovém obchodu. Je ovšem zřejmé, že metoda manuálního testování je velmi neefektivní, nudná a náročná. K dispozici je několik nástrojů, které dovedou nahrát uživatelovi příkazy a později je opětovně spouštět. Tato metoda se velmi hodí pro regresivní testy, kdy si pouze ověříme, zda jsme naší změnou implementace nezavlekli do aplikace chyby.

Do těchto testů spadá kontrola odkazů, které se na dané doméně nalézají,

kontrola formulářů, cookies[2] testování, kontrola na validitu jak HTML kódu, tak CSS, různá testování týkající se práce s databází a kontrolu výskytu důležitých částí webového obchodu, jako jsou například menu, vyhledávající formulář, reklamy a mnoho dalšího.

V této bakalářské práci se budeme zabývat hlavně touto kategorií, protože předchozí řádky se velice podobají úkolům uvedených v zadání od externího dodavatele. U ostatních kategorií pouze nastíníme čím se zabývají, protože nejsou obsahem této práce.

Výkonnostní testování

Webové stránky mohou trpět jejich dlouhým načítáním. Testování výkonnosti by mělo zahrnovat dvě základní metody:

Stresové testování – Již z názvu je zřetelné, že se bude jednat o typ testování, kdy se snažíme daný systém přivést do takových podmínek, ve který se otestují jeho výkonnostní limity. V takto nastavených okrajových podmínkách testujeme, jak je daný systém stabilní, nebo jak se dokáže vypořádat s jeho následným pádem. Toto testování se obecně používá u vstupních polí, přihlašovacích a registračních oblastí.

Zátěžové testy – Mezi další úskalí webových aplikací, se kterými se musíme vypořádat, patří rapidní vzrůst návštěvnosti oproti průměru. Aplikace, respektive server, na kterém běží, musí být schopen se s tímto nárůstem požadavků bez problémů vypořádat. Tento typ testování má za úkol zjistit, jak se systém chová v okrajových podmínkách běhu.

Testování použitelnosti

Každý uživatel si někdy zobrazil stránku, ve které se špatně orientoval, nebyla příliš intuitivní, byla zastaralá, anebo zkrátka a dobře byla ošklivá. Webová aplikace by měla být především snadno ovladatelná.

Testování použitelnosti se jako proces obtížně definuje. Něco, co je podle jedné osoby příšerné, může někdo jiný považovat naopak za vynikající. Použitelnost webových stránek můžeme ale zlepšit dodržováním a testováním několika základních pravidel.

Během tohoto testování, nebo krátce po něm, je vhodné zjistit, jaké pocity z něj uživatelé mají. Ti se většinou sami ozvou v případě, že jsou hrubě

[2]Malé soubory umístěné na klientském počítači, které slouží především pro navázání spojení a identifikaci uživatele.

nespokojeni. Pokud v této fázi není k dispozici zcela funkční verze SW, pak by to mělo být řečeno hlavně testerům, aby nedocházelo k jejich rozuzlení a efektu „Vždyť to vůbec nefunguje“. Také nastává situace, že se testeři snaží vývojářům pomoci, vymýšlejí nové návrhy a možná vylepšení.

Pro tvorbu webu existují nějaké standardy a metodiky, kterými by se měl vývojář řídit. Určitě by neměl používat neustále se pohybující elementy, protože neustálý pohyb koncového uživatele zatěžuje a způsobuje jeho nepozornost a nesoustředěnost. Webová stránka má uživateli podat užitečné informace, které si má pohodlně přečíst.

Vývojář by se měl dále vyhnout nestandardním barvám a fontům, které na webu použije. Je to hlavně proto, že uživatel je z ostatních webů zvyklý na jeden standard a pokud mu předvedeme něco jiného, pak z toho může být zmatený. Další důležitá věc je ta, aby byl web svižný a jeho načítání nezabíralo příliš dlouhou dobu.

Bezpečnostní testování

Tento typ testování je velmi důležitý a u webu toto platí dvojnásob. Naneštěstí není možné, aby u webově založených systémů byla dosažena 100% hodnota bezpečnosti[9]. Zvolená míra bezpečnosti se může promítnout v celkové efektivitě a použitelnosti daného webu. Pokud jsou požadovány příliš přísné bezpečnostní opatření, pak může dojít k degradaci výkonu webu. Na druhou stranu nesmí být bezpečnost systému příliš primitivní, pak je systém zranitelný a zvyšuje se tím riziko napadení.

Útoků na webové aplikace existují desítky, nyní vám představíme některé z nich.

- **Injection a SQL injection** - Dopad tohoto útoku může být poměrně značný. Pokud je úspěšný, pak může mít útočník pod kontrolou celou databázi webové aplikace a může si s ní dělat co chce. Útok spočívá v úpravě SQL dotazu, který je následně odeslán do databázového serveru. Tímto způsobem může dojít ke zneužití dat v databázi, nebo jejich modifikaci.

- **Cross-Site Scripting** - Při tomto typu útoku může útočník vložením vlastního JavaScriptu získat kontrolu nad prohlížečem návštěvníka. Může s ním dělat vše, co JavaScript dovoluje.

- **Odkrytí cesty** - Tento typ útoku se v angličtině nazývá „Full Path Disclousure“ (dále FPD) a jedná se o útok, kdy donutíme webovou aplikaci, aby vyzradila kde na disku jsou uloženy skripty a data dané aplikace. Jednou z několika možností, jak tento typ útoku vyvolat, je

změna vstupních parametrů, se kterými si aplikace nedokáže poradit. Tato změna vyvolá chybové hlášení, ze kterého zjistíme, kde jsou dané skripty umístěné.

- **Nedostupnost služby** - Tento typ útoku je známý pod zkratkou „DoS“[3] a jedná se o pokus vyřadit poskytovanou službu z provozu. Tento útok se projevuje neobvykle malým výkonem webu nebo nemožnosti k jeho přistoupení a je vyvolán tím způsobem, že útočník posílá na server tak velký počet požadavků, že jej server není schopen vyřídit. Tím dojde k jeho zahlcení a následnému pádu.

- **Cross–site request forgery** - Jedná se o zákeřné využití webové stránky, kde jsou uživatelovy příkazy, který věřil, že se jedná o zabezpečenou stránku, odesílány útočníkovi. Jedná se o podvržení formuláře na stránce, kdy si uživatel například myslí, že se přihlašuje do svého mailového účtu. Ve skutečnosti se ovšem jeho citlivé údaje odesílají útočníkovi [14].

- **Nezabezpečená komunikace** - Jedná se o typ špehování, kdy může útočník odposlouchávat komunikaci, která mu není určena.

3.2.5 Akceptační testování

Pokud všechny předchozí testy proběhly bez větších problémů, pak je možné předat aplikaci cílovému zákazníkovi, který následně se svým týmem provede testy akceptační. Tyto testy jsou poté prováděny podle připravených scénářů, které společně vytvořil zákazník s dodavatelem. Nalezené nesrovnalosti mezi aplikací a specifikací jsou následně reportovány zpět vývojářskému týmu, který zajistí jejich nápravu. Vytvořit aplikaci, která zpočátku zcela splňuje specifikaci je velice obtížné a zákazník zpravidla očekává určitou chybovost softwaru a je spokojen, když jeho testovací tým nějakou chybu objeví. Těchto chyb ovšem nesmí být velké množství a jejich opravení nesmí trvat dlouho dobu, protože by to mohlo mít fatální dopad na úspěch celého projektu a budoucí spolupráci.

3.3 Chyby

Pod pojmem chyba si každý představíme stav, který není korektní a neměl by nastávat. Definice chyby by se neměla soustředit pouze na specifikaci,

[3]Denial-of-Servis

protože pak by mohla nastat situace, že by projekty bez specifikace nemohly být testovány, protože by nemohla být nalezena žádná chyba. Dále může nastat, že zákazník zpočátku nemusí plně definovat specifikaci produktu, protože sám nemusí vědět, co vše projekt bude obsahovat. Předchozí dvě řádky by se podle Pattona[11] definovali následovně:

O softwarovou chybu se jedná, je-li splněna jedna nebo více z následujících podmínek:

1. Software nedělá něco, co by podle specifikace produktu dělat měl.
2. Software dělá něco, co by podle údajů specifikace produktu dělat neměl.
3. Software dělá něco, o čem se produktová specifikace nezmiňuje.
4. Software nedělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat.
5. Software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý nebo – podle názoru testera softwaru – jej koncový uživatel nebude považovat za správný.

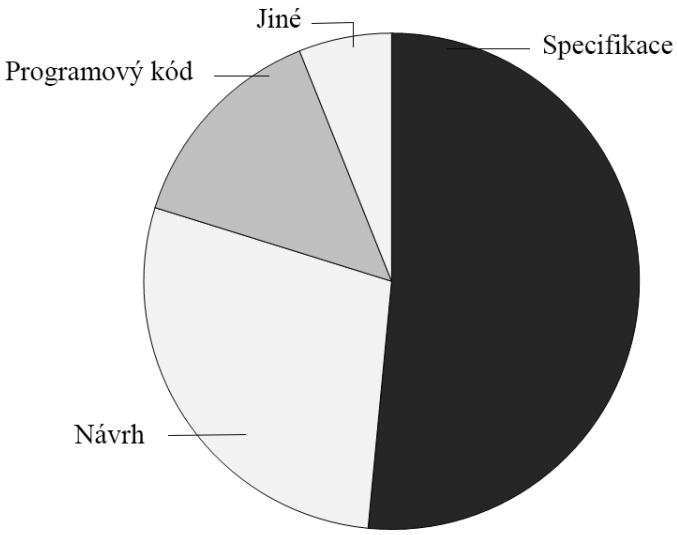
Další možná definice je následující:

„Chyba je cokoli ohledně programu, co podle některého ze stakeholderů[4] *zbytečně snižuje kvalitu programu.“*

Osobně se mi více zamlouvá kratší definice, protože by zřejmě platila i v případě, že by v Pattonově definici nějaký bod chyběl. Navíc je kratší a pro testery je tak lépe zapamatovatelná.

Další věc, kterou by měl tester o chybách vědět, je kde se nejčastěji vyskytují zejména ty nejdůležitější, jejichž oprava by v pozdějších fází mohla být velmi drahá. Jak uvádí Ron Patton ve své publikaci [11], která se zabývá testováním, tvrdí, že více jak polovinu chyb má na svědomí chyba ve specifikaci viz Obr. 3.1.

[4]Termín používaný v managementu a marketingu. Zpravidla se jedná o zákazníka nebo osoby, pro které je produkt určen.



Obrázek 3.1: Příčina chyb podle Pattona

4 Nástroje pro automatizované testování

Při důsledném otestování určitého softwaru je potřeba vykonat obrovské množství práce, které je časově velmi náročné a je zřejmé, že jejich manuální opakování by bylo do jisté míry velmi nevhodné. Přitom opakováním testů se nejlépe zjistí, zda do již otestovaných částí zdrojového kódu nebyla v případě postupného vývoje zavlečena nějaká chyba, která na první pohled nemusí být zcela viditelná. Aby toho nebylo málo, tak může nastat stav, kdy se po opravení jedné chyby vyskytne chyba další. Řešením tohoto začarovaného kruhu jsou automatizované nástroje, které dokáží otestovat aplikaci jako celek s minimem nákladů. Mezi nejdůležitější vlastnosti automatizovaných nástrojů bych podle Pattona [11] uvedl následující:

- **Rychlost** - Automatické testovací nástroje nám umožní výrazně urychlit otestování daného problému.
- **Efektivita** - V případě automatizovaného testování stačí, když se vytvoří jeden testovací scénář, který se poté může několikrát spustit. Tester nemusí své kroky opakovat a může se tak věnovat jiné práci, například vymýšlet nové testovací scénáře.
- **Správnost a přesnost** - Při ručním zadávání několik stovek testovacích případů je logické, že dojde ke snížení pozornosti, což může mít za následek zavlečení chyb. Testovací nástroj vykoná svoji práci vždy stejně a naprosto dokonale.
- **Neúnavnost** - Nástroje zabývající se automatizací neznají únavu a pracují tak dlouho, jak je potřeba.

Všechny tyto vlastnosti zní velice příjemně, ale v žádném případě to neznamená, že by se jimi práce testera dala nahradit. Pouze přinášejí testerům možnost si práci výrazně ulehčit a zjednodušit.

Nyní přišel ten správný čas, abychom si představili pár nástrojů, které umožňují otestování aplikace, která je umístěna na webovém serveru. Za zmínku stojí následující nástroje:

4.1 Nástroje pro jednotkové testování

Co je to jednotkové testování jsme si představili v kapitole o základech testování softwaru. Pro připomenutí si uvedeme, že se jedná o testování, které testuje správnost velmi malých funkčních bloků a ve své podstatě porovnává požadovaný výstup od skutečného.

Nástrojů na tento typ testování je napříč všemi platformami velké množství, z jejich názvů se dá většinou dopředu určit, pro jaký jazyk či platformu jsou určeny. Já jsem se rozhodl moji práci implementovat v programovacím jazyce Java, a tak si popíšeme pouze pár nástrojů, které tento jazyk podporují.

4.1.1 JUnit

Jedná se o framework pro jednotkové testování psané v programovacím jazyce Java a je jedním z nejúspěšnějších frameworků z xUnit[1] rodiny. V současné době se JUnit objevuje u většiny vývojových prostředí, které poskytují testům všemožnou podporu. JUnit je volně dostupný testovací framework využívající pro svůj běh anotace. Anotace jsou značky uvnitř kódu, kterými se označují metody, které následně framework spouští.

4.1.2 TestNG

TestNG je testovací a volně dostupný framework inspirovaný od nástroje JUnit a poskytuje větší množství funkcí, což dělá tak tento nástroj lepší a jednodušší na použití. Název Test**NG** znamená **N**ew **G**eneration. TestNG se velice podobá svému předchůdci, ze kterého se velmi inspiroval, ale je lepší zejména v integračních testech. Tento nástroj umožňuje paralelní spouštění testů, poskytuje integrovaný reportovací systém výsledků a rozšířené anotace, což napomáhá k lepší organizovanosti testovací sad. Nástroj se dá jednoduše doinstalovat do většiny vývojových prostředí, například do prostředí Eclipse. Oproti JUnit poskytuje jednodušší práci s anotacemi a nezáleží v jakém pořadí ve třídě jsou.

4.2 Nástroje pro testování UI na webu

Pokud se chceme zabývat funkčním testováním uživatelského rozhraní internetového obsahu, pak musíme vědět, o co se vlastně snažíme. Snažíme o

[1] Jedná se o pojem, kdy *x* označuje zkratku programovacího jazyka, ve kterém je framework vyvýjen - NUnit (.NET), CppUnit (C++), JUnit (Java),...

simulaci kroků, které by na internetu udělal jeho běžný uživatel, tj, klikání na různé elementy, vkládání textu a mnoho dalšího.

Pro tuto činnost je na trhu několik dostupných nástrojů, některé z nich jsou volně dostupné a jiné zase placené. Některé z nich si představíme a poté si jeden vybereme a budeme s ním dále pracovat. My se ovšem nebudeme zabývat nástroji, které tento proces provádějí za nás. My totiž potřebujeme nástroj, který bude schopen námi požadované akce provádět v rámci internetového prohlížeče. V mé práci bude potřeba použít nějaký testovací framework, který za nás prostřednictvím jeho API vyhledá data na stránce a provede příslušné operace.

4.2.1 WatiJ

Jedná se o nástroj pro testování webových aplikací v jazyce Java [3]. WatiJ[2] je ve své podstatě API na platformě Java vytvořené pro automatizaci funkčního testování webových aplikací skrz reálné webové prohlížeče. Předchůdcem tohoto frameworku je nástroj Watir, který stejnou funkcionalitu plní pro programovací jazyk Ruby. Framework nemá dostupné žádné grafické uživatelské rozhraní a nemá možnost si testy nahrávat a později spouštět. Tento nástroj umožňuje využívat jeho API v následujících prohlížečích:

- Internet Explorer
- Mozilla
- Safari

Mezi hlavní výhody tohoto řešení patří především automatické čekání, dokud se stránka úspěšně nenačte. Ovšem toto tvrzení může být zároveň nevýhodou, protože se tím proces testování může prodloužit.

Mezi hlavní nevýhody tohoto frameworku, což způsobí jeho nevyužití v této práci, patří především to, že je několik let neaktualizován. Poslední update byl zaznamenán v roce 2013. Tento fakt z tohoto nástroje nedělá dobrého kandidáta na zvolení v této práci.

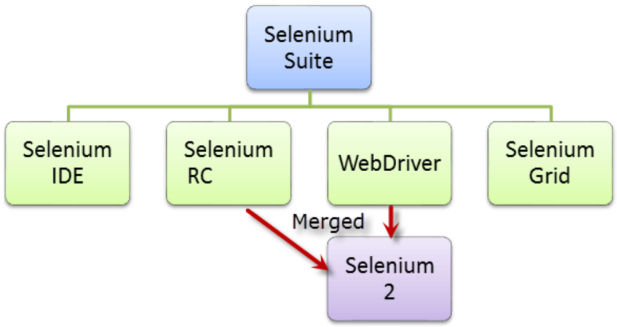
Elementy na stránce umí WatiJ vyhledávat podle jejich textového obsahu nebo libovolného atributu s vyjímkou XPath výrazu, což je velká nevýhoda tohoto nástroje. Předefinovanou vyhledávající metodu má tento nástroj na všechny běžné typy atributů.

[2]Web application testing in java

4.2.2 Selenium

Tento nástroj patří mezi nejznámější testovací frameworky na světě. Selenium je volně dostupný a přenositelný nástroj, který se specializuje na automatizaci funkčního regresního testování webových projektů[4] pomocí webového prohlížeče. Selenium umožňuje vývojářům nahrávat posloupnosti operací, které mohou uživatelé na webu provádět, a později je znovu použít. Nástroj je podporován velkým množstvím různých prohlížečů, operačních systémů i programovacích jazyků - konkrétně v C#, Java, Groovy, Perl, PHP, Python a Ruby. Primárně byl vytvořen pro prohlížeč Mozilla Firefox, ale nyní se již setkává s podporou většiny dnešních populárních prohlížečů.

Selenium není pouze jeden nástroj, ale skládá se z několika produktů [2], kde každý je zaměřený na různé testovací potřeby. Skládá se ze čtyr komponent - Selenium Remote Control, Selenium WebDriver, Selenium IDE a Selenium Grid. S nástupem Selenia ve verzi 2.0 došlo k zúžení tohoto výčtu na tři základní produkty, protože došlo ke sloučení nástrojů WebDriver a Selenium RC, ze kterých vzešel nástroj Selenium 2. Zbylé dva nástroje jsou Selenium IDE a Selenium Grid. Hierarchii těchto nástrojů můžete vidět na obrázku 4.1 Všechny tyto části si popíšeme v následujících řádcích.



Obrázek 4.1: Hierarchie produktů Selenia

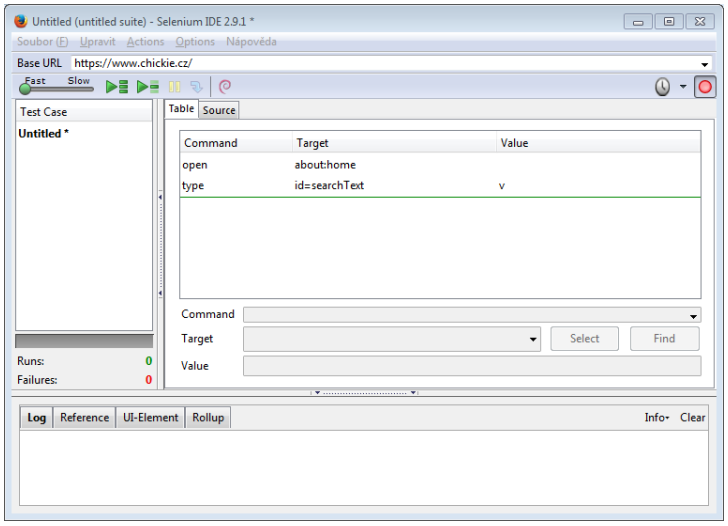
Selenium IDE

Selenium IDE[3] je nejjednodušším nástrojem z balíčku Selenia a jedná se o originální nástroj pro stavbu testovacích skriptů. Je k dispozici jako plugin do prohlížeče od Mozilly Firefox a nabízí přehledné rozhraní pro tvorbu a spouštění automatizovaných testů. Instalace tohoto nástroje je velice jednoduchá, ale má i své nevýhody. Jelikož je pluging určen pouze pro prohlížeč od Firefoxu, pak je nemožné vytvořené testovací scénáře otestovat i na jiných prohlížečích. Toto IDE nabízí možnost nahrávání uživatelských příkazů

[3]Integrated Development Environment

a jejich následný export, což je znovu použitelný skript, do různých programovacích jazyků, pro jejich opakované spouštění. Tento nástroj je volně ke stažení na stránkách Selenia[4].

Jedná se o velmi užitečný nástroj, který dokáží obsluhovat i uživatelé, které nemají s programováním takové zkušenosti. Selenium IDE má podobu jednoduchého okna (viz Obr. 4.2), ve kterém uživatel může provádět veškeré činnosti, které pro testování potřebuje – nahrávání, editaci, spouštění a jejich vyhodnocování. Nahrávání testů je velmi jednoduché. Ve své podstatě proces nahrávání je provedení a vykonání prvního testovacího scénáře, protože uživatel nakliká posloupnost příkazů takovým způsobem, jakým by se pohyboval po dané stránce. Nástroj dokáže rozpoznat jeho kroky. Tyto kroky dále ukládá do HTML struktury, která v sobě uchovává posloupnost příkazů jako uspořádané trojice. Tato trojice označuje typ akce, určení prvku, nad kterým je akce prováděna, a případně hodnota, která je do prvku vkládána.



Obrázek 4.2: Náhled na framework Selenium IDE

Nevýhoda tohoto nástroje je hlavně v tom, že výstupem jsou pouze prototypy testovacích scénářů a nejsou tak dále využitelné pro testování podobných akcí. V této práci se s tímto nástrojem ovšem nesetkáme, protože by výsledný skript nebyl v mém mechanismu vyhodnocování použitelný.

[4] `http://docs.seleniumhq.org/`

Selenium RC

Selenium RC[5] patřil po dlouhou dobu mezi hlavní testovací framework projektu Selenia. Byl to první automatizovaný nástroj pro testování webu, u kterého si mohli uživatele zvolit, ve kterém programovacím jazyce budou testy vyvíjet. Tento framework byl také znám pod názvem Selenium 1.

Mezi výhody tohoto nástroje patří následující:

- Podpora skrze prohlížeče i platformy
- Má vyzrálé a dostatečné API
- Rychlejší spouštění příkazů než u Selenia IDE
- Dokáže poskytovat podmíněné skoky a provádění příkazů ve smyčkách
- Podporuje testování vedené data

Jelikož jsme si uvedly výhody tohoto nástroje, pak je také na místě vyjmenovat i jeho nevýhody:

- Složitější instalace než u IDE
- Uživatel se neobejde bez znalosti programování
- Je potřeba Selenium server
- API obsahuje redundantní a matoucí příkazy
- Interakce s prohlížeče nepůsobí realisticky
- Problémy s JavaScripty
- Pomalejší spuštění než u WebDriveru

Na základě těchto výhod a nevýhod se uživatelé spíše obracejí na framework WebDriveru, který si uvedeme za v následujících řádcích.

[5]Remote Control

Selenium WebDriver

WebDriver je nástroj pro automatizaci webových aplikací, který je lepší než původní Selenium IDE a Selenium RC v mnoha aspektech. Tento framework totiž implementuje modernější a stabilnější přístup v automatizaci akcí, prováděné v prohlížeči. Na rozdíl od Selenia RC není plně závislí na JavaScriptu, dokáže tedy pracovat i bez něj, a komunikuje přímo s webovým prohlížečem bez nutnosti využití Selenium serveru.

Mezi jeho hlavní výhody patří následující vlastnosti:

- Jednodušší instalace než u Selenia RC
- Komunikuje přímo s prohlížečem
- Lepší interakce s prohlížečem
- Rychlejší spouštění příkazů než u IDE a RC

Mezi nevýhody tohoto frameworku patří následující:

- Vyžaduje zkušenosti s programováním
- Složitější instalace než u Selenium IDE
- Nemožnost pohodlné podpory nových prohlížečů

WebDriver je určený k tomu, aby poskytl API, které umožní identifikaci jednotlivých elementů na stránce a dokázal s nimi provádět požadované operace. Dále umožňuje vytvářet testy bez omezení známého ze Selenia IDE, a to zejména nezávislost na jediném webovém prohlížeči, tj. Mozilly od Firefoxu.

Selenium Grid

Jedná se o nástroj, který se používá společně s nástrojem Selenium RC pro paralelní běh testů skrze různé stroje a prohlížeče najednou. Největší výhoda tohoto řešení je zřejmá, protože se ušetří velké množství času. Další výhoda je ta, že je umožněno spouštět testy na všech možných platformách. Nicméně použití tohoto frameworku není náplní práce, a tak jej již více rozebírat nebudeme.

Selenium 2

Jak je zřejmé z obrázku 4.1 uvedeném výše, pak toto řešení vzniklo spojením dvou nástrojů, respektive Selenia WebDriver a Selenia RC. Vývojáři se rozhodli použít to nejlepší z obou nástrojů a vytvořit tento jediný. V praxi se s tímto názvem ale moc nesetkáme, protože se tomuto řešení říká spíše Selenium WebDriver a právě tímto řešením zabývám v této práci.

4.3 Prohlížeče bez UI

Pokud bychom chtěli najednou spustit více testů, pak by bylo velice nevhodné, aby bylo pro každý test spuštěno jedno okno prohlížeče. Z takového důvodu byly vytvořené tzv. „Gui-less browsers“ nebo chcete-li „headless browsers“. Jedná se o nástroje, které pracují jako běžné internetové prohlížeče, akorát že neposkytují grafické uživatelské rozhraní, a tak pracují na pozadí aniž by si toho uživatel byl vědom.

Ve většině případů se s těmito nástroji přímo nesetkáme, protože práce s nimi by byla velmi nepohodlná. Z toho důvodu vznikly frameworky, které nám výrazně ulehčují práci díky objektově orientovanému API. Některé tyto frameworky si uvedeme následovně.

4.3.1 HtmlUnitDriver

Jedná se velmi populární nástroj ve své kategorii, který je využíván v mnoha aplikacích zabývajících se funkčním testováním webového obsahu. Díky svému API umožňuje spouštění webových stránek, vyplňování formulářů, klikání na odkazy a mnoho dalšího. Zkrátka vše co se dá provést v běžném prohlížeči.

HtmlUnitDriver poskytuje velmi dobrou podporu JavaScriptu, podporuje technologii AJAX a dokáže simulovat několik typů prohlížečů, mezi které patří Google Chrome, Mozilla Firefox nebo Internet Explorer.

Tento nástroj je pro jeho funkcionalitu využíván nástrojem WebDriver, který je součástí Selenia. Tato skutečnost z něj dělá nejlepšího kandidáta pro zvolení k této práci.

4.3.2 PhantomJS

Tento nástroj není sám o sobě testovacím frameworkem, jako v předešlém případě, ale jedná se o nástroj, který dokáže ovládat internetový prohlížeč pomocí JavaScriptových příkazů. Existuje několik aplikací, které využívají

tohoto nástroje, jedním z nich je GhostDriver, který je součástí námi známého WebDriveru. PhantomJS ovšem nepřináší takovou funkcionalitu jak předchozí uvedený, a tak jej v práci využívat nebudeme.

4.4 Další užitečné nástroje

Abychom mohli vůbec uvažovat o vytvoření nějakého testu, pak si musíme nejdříve stanovit, nad kterými prvky z webové stránky budeme chtít interagovat. Na trhu existuje velké množství nástrojů, které dokáží spolehlivě identifikovat daný element na stránce. Mě se nejvíce osvědčili následující.

4.4.1 Firebug

Jedná se o volně dostupný plugin do prohlížeče Mozilla Firefox, který poskytuje možnost analyzovat webové stránky a jejich elementy. Firebug se aktuálně vydává ve verzi 2.0.15 a patří mezi nejlepší nástroje ve své kategorii.

Abychom zjistili nějaké informace o libovolném prvku nacházejícím se na stránce, pak stačí pouze najet myší nad tento prvek, stisknout pravé tlačítko na myši a z nabídky zvolit možnost *Inspect Element with Firebug*. Poté dojde k zobrazení nástroje Firebug s HTML kódem, který je zobrazen ve stromové struktuře.

Firebug nabízí velké množství debugovacích funkcí, editaci a monitorování jakékoli webové stránky. Pro náš zájem plně postačí možnosti generování XPath a CSS selektoru námi hledaných elementů. Tato možnost se nám zobrazí, když do zdrojového kódu, zobrazeného tímto nástrojem, klikneme pravým tlačítkem myši a zvolíme možnost *Copy Xpath*, respektive *Copy CSS Path*. Samozřejmě nemusíme využívat pouze těchto dvou lokalizačních mechanismů, protože zde máme další možnosti, mezi které patří například lokalizace pomocí identifikátoru, názvu třídy a mnoho dalšího. Všechny typy lokátorů, včetně jejich využití v tvorbě testů, budou popsány v uživatelské dokumentaci.

Firefinder pro Firebug

Jedná se o plugin, který pouze rozšiřuje funkcionalitu předchozího nástroje Firebugu. Nástroj lze použít pro ověření námi nalezeného prvku pomocí jeho *XPath*, nebo *CSS* výrazu. Nástroj pracuje jednoduše tak, že mu na vstup přivedeme požadovanou cestu a on nám vrátí seznam elementů, který splňují vloženou cestu. Firebug sice dokáže vytvořit cestu *Xpath*, ale jeho podoba

je nevhodná, protože obsahuje cestu od kořenového uzel, tj. od uzle *HTML*. Proto je tento plugin vhodné využívat pro vlastní tvorbu *Xpath* lokátorů, kde si můžeme okamžitě otestovat jejich korektnost.

4.4.2 JDOM

Před tvorbou samotných testů jsem se musel rozhodnout v jakém formátu budu chtít testovací scénáře a testy uchovávat. Zpočátku mi přislo jako dobré řešení uchovávat data ve formátu HTML, protože tento formát využívá dříve zmíněný nástroje z rodiny Selenia IDE. S postupem času jsem se ovšem potýkal s problémy uchování dostatečného množství informací, a tak jsem se musel poohlédnout po jiném řešení.

Nejvíce se mi zamlouvalo uchovávat moje data ve struktuře XML[6], což je značkovací jazyk, který je určen především pro výměnu dat a pro publikování dokumentů, u kterých popisujeme strukturu z hlediska věcného obsahu jednotlivých částí a nezabývá se vzhledem. Výhoda tohoto značkovacího jazyka je také ta, že se dá jeho kód poměrně dobře číst.

Pro snadné čtení i ukládání jsem si zvolil volně dostupný nástroj JDOM[7], který byl navržený pro platformu Java. Tento nástroj se nyní vydává ve verzi `2.0.6`.

4.5 Výběr testovacího nástroje

Mnoho lidí začíná se Selenium IDE, protože ho lze používat bez znalostí programování nebo skriptovacích jazyků. Pro efektivní využití Selenia je potřeba Selenia WebDriver nebo Selenia-RC ve spojení s jedním z několika podporovaných programovacích jazyků. Jelikož Selenium WebDriver poskytuje lepší funkcionalitu a je více využíván, tak jsem se rozhodl v práci použít právě tento nástroj. Aby bylo testování co nejefektivnější a nejrychlejší, tak jsem se rozhodl využít nástroj HtmlUnitDriver, který poskytuje mnohem lepší dokumentaci než druhý uvedený, čímž se pro mě stane vývoj příjemnější . Pro identifikaci jednotlivých elementů na stránkách využívám nástroje Firebug, který je pro tuto práci naprosto dostačující. Zdrojové soubory testů a scénářů budou ukládány ve formátu XML, kdy pro jejich čtení i zápis budu využívat nástroje JDOM.

[6]EXtensible Markup Language
[7]Java Document Object Model

5 Realizace aplikace CSWebTest

Jak již bylo řečeno v úvodu – výstupem této práce má být aplikace, která umožní uživateli vytvářet a spouštět testovací dávky pro několik internetových obchodů podobného typu. Aby mohl být tento požadavek splněn, pak budeme muset mít možnost vytvořit jeden testovací scénář a tímto scénářem se pokusit otestovat konkrétní funkcionalitu na více obchodech. Aplikaci jsem se rozhodl vytvořit v programovacím jazyce Java, protože práce s tímto jazykem je pro mě nejpřirozenější.

Práce vznikla na podnět externí firmy, která chtěla vytvořit SW, jenž by byl schopen provést regresní testování po zavedení aktualizací svých internetových obchodů pro zajištění jejich správné funkcionality. Byla zde samozřejmě možnost využít už některý z existujících nástrojů na otestování webu. Nakonec jsem se rozhodli, že bychom chtěli vlastní produkt, který bychom mohli v budoucnu zakomponovat do redakčního systému, na jehož vývoji se firma podílí.

5.1 Architektura

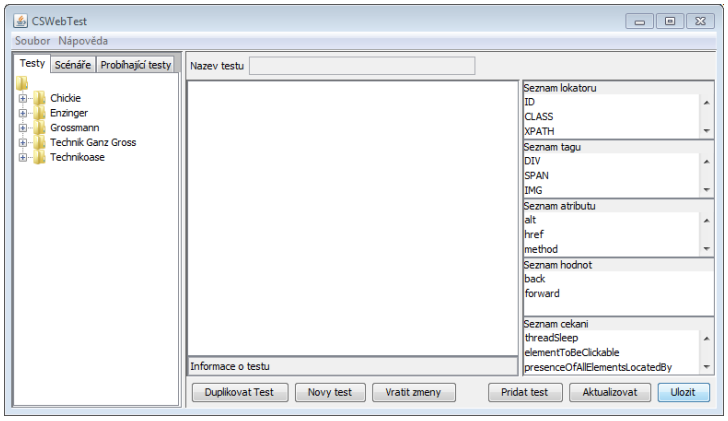
Před samotnou tvorbou aplikace jsem se musel se svým externím zadavatelem nejprve dohodnout, jak bude aplikace vypadat. V této fázi je důležité se shodnout na základním uživatelským rozhraní, které je možné postupně při tvorbě modifikovat, aby bylo co nejpřirozenější a působilo na koncového uživatele přívětivě. Základní rozdělení ovládacích panelů bude znázorněno v následujících řádcích.

5.1.1 Obecný popis aplikace

Aplikace se skládá z několika částí, které se v průběhu ovládání programu překreslují a poskytují tak potřebné informace pro její ovládání. Hlavní okno můžete vidět na obrázku 5.1. První část se skládá z položkového panelu, který zobrazuje buď všechny dostupné scénáře a testy ve stromové struktuře, nebo seznam aktuálně připravených testů, které je možno spustit. Tento panel samozřejmě poskytuje uživateli zpětnou odezvu na kliknutí myši, která je specifická pro daný strom či seznam. V případě kliknutí do seznamu testů se uživateli zobrazí informace o konkrétním testu. Tímto způsobem má uživatel

možnost získat informace o pádu testu a pokusit se tyto problémy opravit, aby bylo jeho další spuštění úspěšné.

Druhá část programu se skládá buď z editoru, nebo z tabulky, ve které jsou zobrazeny testy, které se rozhodl uživatel otestovat. V případě editorů je zde umístěna plocha pro zadávání textu, která je rozšířena o několik seznamů, ze kterých má uživatel možnost vybírat a postupně tak tvořit test, respektive scénář. Samozřejmostí jsou také tlačítka, které jsou pro test, respektive scénář, potřebná.



Obrázek 5.1: Náhled aplikace v editaci testu

5.1.2 Základní UML diagram tříd

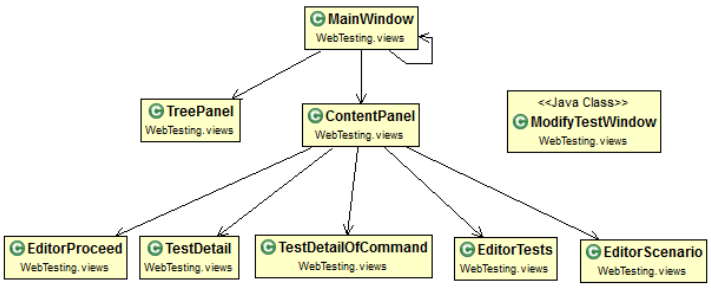
Aby si čtenář vytvořil základní přehled o tom, jak je aplikace poskládána, tak jsem se rozhodl některé důležité části programu zobrazit pomocí modelovacího jazyka UML[1], který poskytuje standardní cestu pro vizualizaci návrhu systému. Tento jazyk umožňuje vytvářet několik typů diagramů, které se rozdělují na dvě hlavní kategorie podle toho, co popisují:

- **Statickou strukturu** - Tyto diagramy zobrazují objekty, které systém obsahuje, a jak spolu souvisejí.
- **Dynamické chování** - Zde se popisuje vzájemná spolupráce objektů.

Obě výše uvedené kategorie obsahují několik typů diagramu, já se ovšem v této práci zmíním pouze o diagramu tříd, který ukazuje statickou strukturu tříd v systému, tj. třídy a jejich vztahy. V tomto typu diagramu se mohou dále vyskytovat atributy a operace, které daná třída poskytuje. Kvůli přehlednosti jsem se ovšem rozhodl tyto informace neuvádět a zobrazit pouze názvy tříd a jejich souvislosti v rámci celého systému.

[1]Unified Modeling Language

Prvním třídním diagramem, který bych chtěl představit, je diagram zobrazující grafické uživatelské rozhraní aplikace, který můžete vidět na obrázku 5.2.



Obrázek 5.2: Diagram základních grafických částí

5.2 Grafické uživatelské rozhraní aplikace

Grafické rozhraní této aplikace se skládá ze dvou základních panelů, které jsou v diagramu (viz Obr. 5.2) reprezentovány uzly *ContentPanel* a *TreePanel*. Tyto panely si nyní představíme.

5.2.1 Panel pro organizaci testů a scénářů

Jak bylo řečeno výše, tak jsem aplikaci ve své podstatě rozdělil na dvě části a my si nyní představíme část, ve které jsou testy, respektive scénáře, zobrazeny.

Testy a scénáře jsou zobrazeny ve stromové struktuře, aby manipulace s nimi byla co nejjednodušší. V případě testů tvoří hlavní uzly konkrétní elektronické obchody a následující struktura závisí na potřebách uživatele. V mém případě, protože jsem provedl jak testování pozitivní, tak negativní, jsou následující úrovně uvedeny podle těchto typů. V případě scénářů jsem uznal za vhodné pojmenovat hlavní uzly podle toho, co testují. V těchto uzlech se mohou například vyskytovat scénáře, které korespondují jednotlivým obchodům.

Po kliknutí na daný uzel, respektive větev stromu, se uživateli zobrazí v textové ploše konkrétní test, respektive scénář, nebo se mu zobrazí vyskakovací okno, kde si může vybrat, co s daným uzlem bude chtít provést.

V tomto panelu se dále ještě může vyskytovat seznam aktuálně prováděných testů, které také signalizují, jak testy proběhly podle jejich podbarvení.

Pokud nás zajímá detail konkrétního testu, pak položku v seznamu stačí rozkliknout, což nám zobrazí informace o tomto testu. Podbarvení těchto položek může být následující.

- Červené podbarvení znamená, že v průběhu testu nastala nějaká chyba.
- Zelené podbarvení značí úspěšně provedený test.
- Modré znamená, že je stále ve fázi zpracovávání.

5.2.2 Editační panel

Tento panel tvoří základ celé aplikace. V závislosti na právě otevřené kartě z minulého panelu se nám zobrazí okno, které může reprezentovat následující data:

- Editace scénáře
- Editace testu
- Tabulka testů připravených ke spuštění
- Informace o konkrétním testu
- Detailní informace o jednom příkazu z testu

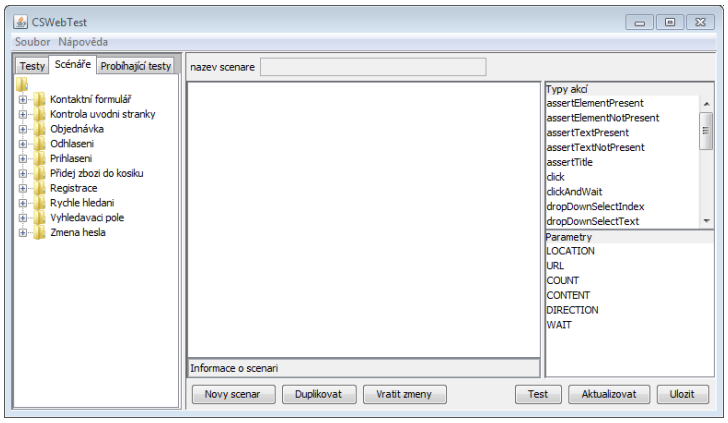
Editace scénáře

V tomto panelu si uživatel definuje posloupnost operací a jejich nezbytné parametry pro jejich korektní provedení. U jednotlivých parametrů si uvádíme pouze jejich symbolické konstanty, které následně dodefinujeme při vytváření příslušného testu.

Uživatel je v tomto okně informován pomocí seznamů, které akce a jaké parametry můžou být zadávány. Ten si pak může vybrat, zda bude scénář tvořit klikáním do těchto seznamů, nebo jej bude tvořit ručně. Při vytváření se ovšem musí soustředit na to, aby dodržoval základní syntaktická pravidla, která si uvedeme dále v této práci. Při tvorbě testů bych použití seznamu s akcemi velmi doporučil, protože po jejich zvolení dojde k automatickému vyplnění všech požadovaných parametrů, které daná akce očekává.

Součástí tohoto panelu jsou také tlačítka, která nad daným scénářem provádí ekvivalentní operace, a informační panel, který poskytuje uživateli zpětnou odezvu o procesu ukládání, aktualizace, duplikace a vytváření nového scénáře. Pro přehlednost si můžete aplikaci s tímto panelem prohlédnout na obrázku 5.3.

Tlačítko „test“ zde představuje možnost z aktuálního scénáře vytvořit test, který si následně můžeme dodefinovat konkrétními hodnotami a spustit. Dále je zde možnost použít tlačítko „Vrátit změny“, které obnoví daný scénář z jeho zdrojového souboru.



Obrázek 5.3: Editor scénáře

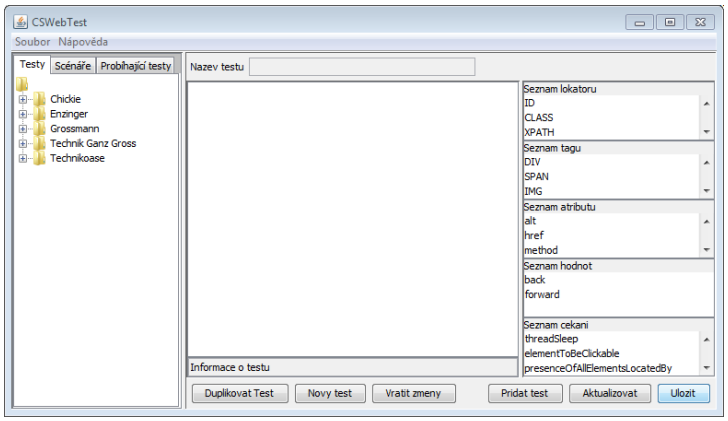
Editace testu

Tento panel slouží pro vytvoření testu na základě scénáře, ze kterého byl vytvořen. Jsou zde zobrazeny pouze informace, které jsme si definovali symbolickými konstantami v editaci scénáře a uživatel tudíž musí vědět, čeho se daná posloupnost symbolických konstant týká. Je pouze na uživateli, zda si typ akce poznamená do těchto konstant či nikoli.

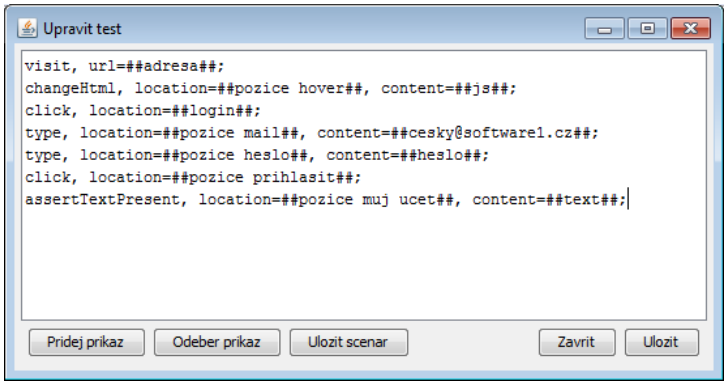
Při tvorbě testu má uživatel možnost výběru konkrétních elementů ze seznamů, které jsou zobrazeny v levé části tohoto panelu. Uživatel těchto panelů může využít, ale nemusí.

Součástí tohoto editoru jsou také samozřejmě tlačítka, která poskytují příslušnou funkcionalitu, a informační panel, který poskytuje uživateli zpětnou vazbu v případě, že došlo při editaci k nějakému problému. Pro představu jak tento panel vypadá se můžete podívat na obrázek 5.4. Pokud uživatel při vyplňování tohoto testu zjistí, že nějakou akci ve scénáři neuvedl, pak má možnost tento test dodefinovat kliknutím pravého tlačítka myši do editačního pole testu. Po tomto kliknutí má uživatel dvě možnosti, jak daný test upravit. Obě editace se zobrazují v jednom okně, jehož konkrétní vzhled je dán typem editace. Okno můžete vidět na obrázku 5.5

Upravit příkaz – Pokud uživatel využije tuto možnost, tak se mu zobrazí okno, ve kterém se mu nabídne editace konkrétního příkazu, v němž



Obrázek 5.4: Editor testu



Obrázek 5.5: Okno, ve kterém se zobrazuje editace testu, nebo příkazu.

se vyskytoval kurzor myši. V okně se mu zobrazí posloupnost akcí a jejich symbolické konstanty, které si definoval v příslušné šabloně. Poté má uživatel dvě možnosti, jak příkaz upravit:

- Odebrání parametru – Před použití tohoto tlačítka je důležité, aby se kurzor myši nalézal v místě, ve kterém je parametr, který chce uživatel vymazat.
- Přidání parametru – Tato akce se provede jednoduchým vložením daného typu parametru a jeho symbolického výrazu, který se zobrazí v editaci testu. Je důležité, aby veškeré parametry byly odděleny čárkou.

Upravit test – Jedná se o podobnou funkci jako v předešlém případě jen s tím rozdílem, že se zde mohou přidávat i odebírat celé příkazy. Pro úspěšné přidání příkazu musí uživatel splnit dvě podmínky. První je ta, že musí umístit kurzor myši na požadovanou pozici. Pokud bude kurzor umístěn uvnitř nějakého příkazu, pak se nový příkaz umístí na pozici před tento

příkaz. Možnost odebírání funguje obdobným způsobem – uživatel umístí kurzor do řádky s tímto příkazem a stiskne tlačítko odebrat příkaz. Dále je zde pro uživatele možnost z aktuálního testu vytvořit scénář, což ovšem není zase až tak přínosné a je pouze na pouze na uživateli zda této možnosti bude využívat či nikoli.

Tabulka připravených testů

Pokud se rozhodneme, že budeme chtít nějaký test spustit, pak jej nejprve musíme přidat do tabulky připravených testů. Tato tabulka obsahuje následující údaje.

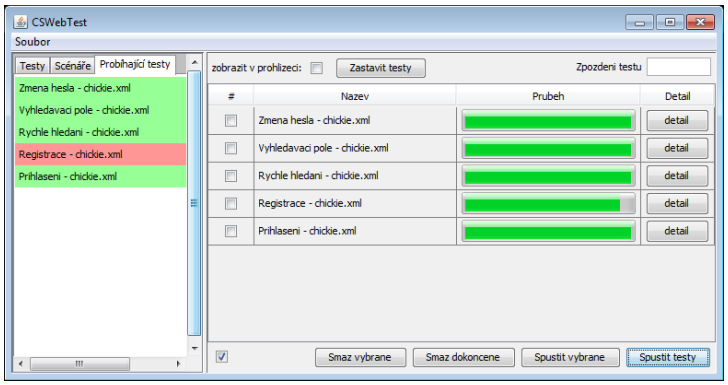
- **CheckBox** - Tento sloupec je určený pro označení daného testu, díky kterému může dojít ke spuštění, nebo ke smazání určených testů.
- **Název** - Sloupec, ve kterém je uveden název přidaného testu.
- **Průběh** - Položka tabulky, která nám přináší vizuální informaci o průběhu jednotlivých testů. Každý test se skládá z posloupnosti několika testovacích příkazů. Jakmile se jeden z příkazů úspěšně provede, pak dojde k aktualizaci tohoto panelu, který označuje procentuální část úspěšně provedených příkazů v rámci konkrétního testu.
- **Detail** - Tlačítko, které uživateli umožňuje náhled na detailní informace v rámci daného testu. Po jeho stisknutí se uživateli zobrazí okno, ve kterém je zobrazena posloupnost příkazů, které obsahují informace o jejich provedení. V případě nedokončení testu nám poskytne informace o jeho chybách.

Samozřejmostí jsou také tlačítka, která slouží pro ovládání přidaných testů a zaškrtávací tlačítko, které označí, nebo případně zruší označení u všech testů. Pro lepší orientaci v tomto panelu se podívejte na obrázek 5.6.

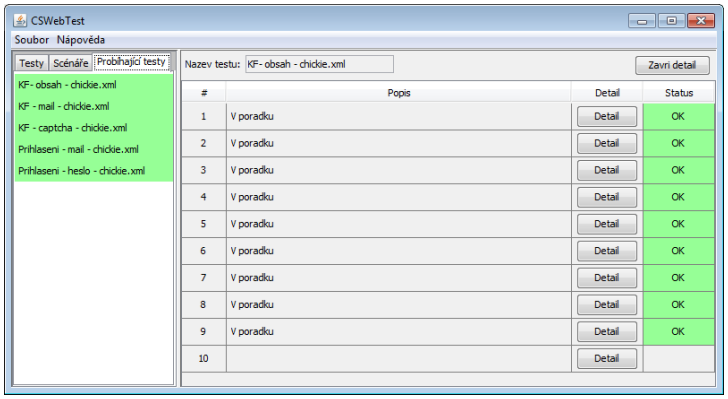
Informace o testu

Tento panel obsahuje seznam všech příkazů, které jsou v konkrétním testu uvedeny a poskytuje základní informace o jejich provedení. V případě, že v nějakém příkazu došlo k chybě, pak se nám o ni zde zobrazí jednoduchá informace. Uživatel má po kliknutí na její detail možnost zjistit její podrobné informace, díky kterým je uživateli umožněna jejich oprava.

Do tohoto panelu se uživatel může dostat dvěma způsoby. Buď klikne na tlačítko detail, který je umístěný v tabulce připravených testů, nebo po kliknutí do seznamu v levé části programu. Vzhled tohoto panelu si ukážeme na obrázku 5.7.



Obrázek 5.6: Informace o spuštěných testech



Obrázek 5.7: Informace o testu

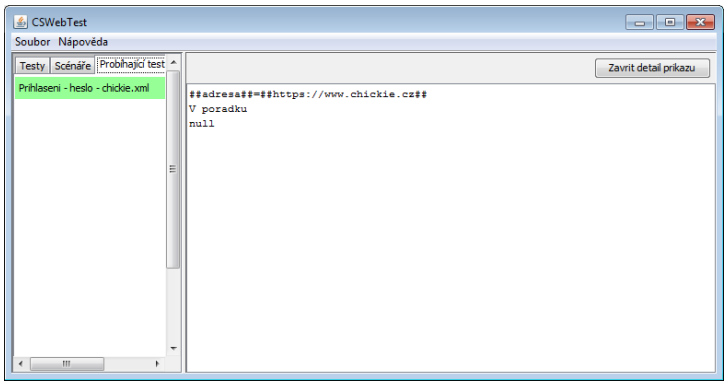
Informace o příkazu

Jedná se o poslední panel (viz Obr. 5.8), který slouží pro detailní náhled na příkaz, ve kterém se můžeme dozvědět informace týkající se jeho neúspěšného provedení. Panel obsahuje pouze tlačítko pro ukončení tohoto náhledu a textovou plochu, ve které jsou informace o chybě.

5.3 Významné třídy a algoritmy

Před samotným vývojem aplikace je velmi užitečné, aby se programátor rozhodl, jak bude své kódy strukturovat, aby se v pozdní fázi vývoje ve svém kódu dokázal orientovat. Já jsem se rozhodl mé kódy strukturovat do šesti základních balíků, ze kterých je jasno, který obsahuje jaké části kódu. Tyto balíky jsou následující:

- **JUnitTests** - Balíček, který obsahuje třídy určené pro jednotkové testování právě vytvářených částí zdrojového kódu. V konečném důsledku



Obrázek 5.8: Detail příkazu

nemají na aplikaci žádný vliv a jsou určené pouze pro vývojářské účely, aby byla zajištěna správná funkcionalita právě vytvořených algoritmů.

- **common** - Tento balíček obsahuje důležité kódy, které jsou určeny pro uchování důležitých dat, zejména instancí testů, scénářů, jejich manipulaci a reprezentaci. Jsou zde umístěny pomocné knihovní třídy, které se využívají napříč celou aplikací.

- **exception** - V tomto balíku jsem se rozhodl vytvářet vlastní výjimky, které vyvolávám v případě jakýkoliv problému při vytváření testovacích scénářů, testů nebo ve fázi zpracování testů. Tyto výjimky následně odchytávám a na jejich základě vytvářím informační zprávy pro uživatele.

- **interfaces** - Balík, ve kterém jsou obsažena rozhraní, která v programu využívám.

- **tests** - Balík, ve které je uložen kód, který tvoří jádro testovacího modulu. Jsou zde umístěny veškeré algoritmy, které pracují s webovými stránkami a manipulují s nimi.

- **views** - Balík, bez kterého by se aplikace neobešla, protože obsahuje veškeré grafické rozhraní, posluchače pro uživatelovi příkazy a obslužné metody, které jsou vyvolány na základně uživatelských příkazů.

DataNode.java

Jedná se o třídu ve které jsou uložena data potřebná pro uchování dat, která se pojí s daným scénářem a testem. Třída obsahuje dva základní atributy:

- **Název uzlu** – Atribut, který nese informaci o názvu konkrétního uzlu.

- **Instance XML dokumentu** – Velmi důležitý atribut, který v sobě uchovává veškerou XML strukturu, kterou daný test či scénář nese. Tato data byla získána pomocí nástroje JDOM.

Třída dále poskytuje funkcionalitu, která dokáže převést data obsažené v instanci XML dokumentu do textové podoby. Element můžeme vypsat celkem ve třech podobách, což poskytují tři následující metody:

- **elementsToString()** - Metoda, která převede konkrétní test, respektive scénář do textové podoby. Výstup z této metody v programu přímo nenalezneme, ale je velmi vhodný pro vývojářské účely, protože nástroj JDOM neposkytuje metodu, která by vypsala obsah instance XML objektu.
- **elementsToTestText()** - Metoda, která převede konkrétní test do podoby, která se vyskytuje v editoru testů.
- **elementsToScenarioText()** - Metoda, jejíž výstup se zobrazí v editoru scénáře.

TreeNode.java

Třída, díky které v programu uchováváme informace o testu a scénáři, čímž se stává jednou z nejdůležitějších v celém programu. Jak je z názvu zřejmé, tak se jedná o uzel, který je vkládán do stromové struktury. Třída obsahuje několik atributů, abychom mohli jednoznačně identifikovat jejich typ. Mezi tyto atributy patří následující:

- **Instance na předka** – Instance téže třídy, která v sobě nese informaci o nadřazeném scénáři či testu. Tento atribut se využívá hlavně pro tvorbu stromu, ve kterém jsou testy a scénáře organizované.
- **Data příslušného uzlu** – Jedná se o instanci třídy „DataNode.java“, kterou jsme si popsali výše.
- **Cesta k souboru** – Atribut, ve kterém je uložena cesta ke zdrojovým scénářům a testům. Tato cesta neslouží pouze pro potřebu aktualizace dat, ale je využita i například pro jednoznačnou identifikaci testu, respektive scénáře.
- **Typ uzlu** – Pomocný atribut, díky kterému víme, zda se jedná o uzel typu scénář nebo test.

TestData.java

Třída, která se využívá v případě, že chceme konkrétní test přidat do fronty pro otestování. Třída obsahuje kromě samotné instance třídy „TreeNode.java“ také následující informace:

- Zaškrtávací tlačítko, které umožňuje spustit, nebo naopak smazat konkrétní test z fronty.
- Informaci o aktuálním stavu testu. V případě že test běží, tak se tato informace průběžně po každém korektním zpracování příkazu aktualizuje. Uživatel tak má vizuální informaci o průběhu testu.
- Tlačítko, které slouží pro zobrazení detailu o daném testu. Této funkcionality se využije především v případě neúspěšného provedení.

XmlOperation.java

Velmi důležitá třída, která obsahuje obslužné metody pro manipulaci s XML soubory. Třída byla vytvořena hlavně z důvodu ukládání a načítání zdrojových souborů s testy a scénáři, které je velmi jednoduché. Komplexnější operace, které jsou v této třídě uvedeny, je potřebná aktualizace uzlů před jejich uložením. Pro tyto operace je ve třídě definováno několik metod, které požadované postupy provedou.

UtilityClass.java

Jedná se o třídu, která byla původně implementována jako knihovní třída, ve které by se nacházeli pomocné metody, které jsou při realizaci programu potřeba.

V této třídě se nachází spoustu uživatelských konstant, které slouží především pro správné parsování uživatelského vstupu a pro řádné rozhodování v testovacím mechanismu. Můžeme zde nalézt následující informace, které si ovšem detailně popisovat nebudeme z důvodu jejich velkého množství.

- **Konstanty** ze seznamů, které se vyskytují v seznamech umístěných v editorech, sloužících především pro řádné parsování v průběhu vyhodnocování testů.
- **Umístění zdrojových dat** mezi které patří především cesty do adresářů obsahující soubory s testy a scénáři. Dále jsou zde obsaženy cesty ke konstantám, které jsou uvedené u editorů v seznamech.

- **Parsovací znaky** pro separaci jednotlivých řádek, příkazů a parametrů v příkazu.
- **Barvy**, které slouží pro vizuální informaci výsledků testů a informačních sdělení z editorů.

MainTest.java

Třída, která má za úkol provádět samotné testování a jedná se tedy o „srdce“ programu. Jakmile se uživatel rozhodne pro otestování nějaké sady testů, pak se pro každý test vytvoří instance této třídy, ve které se následně spustí samostatné vlákno, které daný test obslouží. Detailnější popis této třídy nalezneme v sekci zabývající se testovací mechanismem.

MainWindow.java

Jedná se o spouštěcí třídu, která zároveň slouží jako hlavní zobrazovací část aplikace. Obsahuje instance několika panelů, které tvoří grafické uživatelské rozhraní aplikace.

5.4 Testovací mechanismus

Celý testovací mechanismus je uvedený ve třídě `MainTest.java` a obsahuje několik důležitých metod, které si zanedlouho uvedeme. Aby byl uživatel dostatečně informován o průběhu testu, tak jsem se rozhodl využít mechanismu výjimek, který v případě nějakého selhání nastaví příkazům informační parametry, které se následně zobrazí u daného testu v přehledné tabulce.

Testovací mechanismus se větví do několika směrů a funguje jednoduchým způsobem. Na začátku všeho je konstruktor, který přijme jediný parametr, kterým je samotný test, respektive instance ze třídy `TestData.java`. Tento test obsahuje pak jednotlivé příkazy, které jsou v cyklu zpracovávány. Tento cyklus je ošetřen proti vyvolávání výjimek na jejichž základě dochází k nastavování varovných zpráv, které si uživatel může v editoru prohlédnout, a tak zareagovat na chyby, které v průběhu testu nastaly.

Po kladném zpracování jednotlivých příkazů dochází k aktualizaci informačního panelu, který signalizuje, jaká část testu je úspěšně za námi. Pokud při zpracování příkazů dojde k nějaké chybě, pak je vyvolána výjimka, která ukončí základní cyklus, nastaví chybová hlášení a test ukončí. Uživatel pak má možnost si otevřít detail testu a zjistit, co způsobilo pád testu. Na základě této zprávy opraví test a může jej spustit znovu.

V případě neúspěšného testu se informace o chybách zapíší do zdrojového souboru ke konkrétnímu testu formou přidáním atributů k danému XML elementu.

Za základním cyklem, který načítá aktuální příkazy, je další větvení, které zjistí o jaký typ příkazu se jedná a na tomto základě se algoritmus dál rozhoduje. Aby došlo k jednoduchému zpracování jednotlivých parametrů z příkazu, pak jsou naimplementovány metody, které z daných parametrů vyextrahují jednotlivá data a v případě špatné syntaxe či neexistujících parametrů dojde k vyvolání výjimky a pádu celého testu s nastavením chybového hlášení. Pomocné metody pro extrahování dat jsou následující.

- **`getSubElement(String child, Element parent)`** – Metoda, jejímž vstupem jsou dva parametry. Prvním je řetězcová hodnota, která určuje hledaného potomka z druhého parametru, jímž je instance XML elementu.

- **`getTextFromElement(Element element)`** - Metoda, která se z jejího atributu pokusí vyextrahovat textovou hodnotu. Pokud tento element neexistuje, nebo je jeho hodnota prázdná, pak je vyvolána výjimka, která ukončí testovací mechanismus.

- **`getTextFromAttribute(Element element, String attribute)`** - Metoda, která zajišťuje stejnou funkcionalitu jako předchozí s tím rozdílem, že hodnotu neextrahuje z elementu XML, ale z atributu, který je uveden u elementu.

- **`getIntFromAttribute(Element element, String attribute)`** - Metoda, která má za úkol vyextrahovat z určeného elementu atribut, u kterého se předpokládá, že se jedná o číselnou hodnotu. Pokud se zde číslo nevyskytuje, pak dojde k pádu testu a tím i ukončení testu.

- **`getElementsFromLocator(Element element)`** – Jedná se o pomocnou metodu, jejímž úkolem je získat HTML identifikátor z potomka XML struktury, který se nalézá v elementu z parametru. Jinými slovy algoritmus prohledá veškeré potomky elementu z parametru, identifikuje potomka určující pozici prvku na HTML stránce a vrátí seznam HTML elementů, které odpovídají uvedené pozici.

- **`getAmountElements(List<WebElement> elements, int amount)`** - Metoda, jejímž výstupem je počet elementů, které uživatel požaduje. První parametr je seznam WebElementů, který jsme

získali díky předchozí metodě. Druhý parametr označuje počet elementů, který uživatel potřebuje. Pokud nastane situace, že počet potřebných elementů je menší než počet dostupných elementů, pak je vyvolána výjimka, která způsobí konec testu.

- **`findElementFromParentInList(String typeOfList, Element element)`** - Metoda, která slouží především pro kontrolu, zda byly zadány správné typy operací. Zjišťuje, zda se hodnota elementu z druhého parametru vyskytuje v zadaném seznamu z prvního parametru.

- **`getLocator(String name, String value)`** - Metoda, která má za úkol vrátit lokační mechanismus „By“ na jehož základě dochází k vyhledání prvku na HTML stránce. První atribut obsahuje identifikátor prvku, mezi které patří například (id, name, class, xpath...). Druhý atribut pak označuje hodnotu, které identifikátor nabývá.

5.5 Požadavky pro běh aplikace

Před samotným vývojem aplikace jsem se rozhodl, že ji budu programovat v nejnovější verzi Javy, tj. ve verzi 1.8.

Pokud se tedy uživatel rozhodne využít tuto aplikaci pro otestování libovolného internetového obsahu, nemusí se přímo jednat o obchod, pak si nejprve musí stáhnout ze stránek Oraclu[2] standardní verzi Javy 1.8 nebo vyšší.

Další podmínkou, kterou musí uživatel splnit je ta, že musí mít na svém počítači nainstalovaný prohlížeč Mozilla Firefox, jehož instanci program využívá.

[2] `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

6 Tvorba testů a scénářů

V této kapitole je vysvětlen postup, kterým se v této aplikaci vytvářejí testy a scénáře, což je cílem této práce.

Každý testovací případ má samozřejmě jiné požadavky, ale pokud provádíme stejné operace v rámci různých obchodů, pak tyto posloupnosti budou velmi podobné a nebude se muset jejich konfigurace vytvářet od začátku. Tohoto faktu se držím při implementaci mého programu.

Samotný program se skládá ze tří základních stavebních kamenů, mezi které patří tvorba scénářů, testů a vyhodnocení provedených testů. Všechny tyto kategorie si v práci popíšeme.

6.1 Tvorba scénářů

Předtím než začneme vytvářet testy, které otestují funkcionalitu daného obchodu, musíme si nejdříve vytvořit scénář[1], podle kterého následně vytvoříme konkrétní test. Hodnoty, které jsou ve scénáři uvedeny slouží pouze jako symbolický popis kroků, které při testování budeme chtít provést. Po vytvoření a následném uložení šablony máme možnost z této šablony vytvořit konkrétní test, do kterého vyplníme požadované informace a spustíme.

Je zde také možnost tvorbu šablony přeskočit a začít realizovat test kliknutím pravého tlačítka myši do editační plochy testu a výběru položky *Upravit test*, kde si může uživatel jednotlivé kroky nadefinovat.

Postup vytváření scénářů

Scénáře se vytvářejí jednoduchým způsobem. Po zapnutí aplikace se musí uživatel přepnout do editace scénáře kliknutím na kartu scénáře v levé části programu obsahující stromovou strukturu testů, respektive scénářů. Poté se překreslí hlavní okno a uživatel tak může začít s vytvářením. V pravé části programu se nalézají celkem dva seznamy, kde první reprezentuje typy akcí, které se dají při testování použít. Druhý seznam obsahuje typy parametrů.

Uživatel má tedy dvě možnosti jak postupovat. Buď bude využívat těchto seznamů, nebo bude testy tvořit sám. Ze začátku je ovšem lepší využívat

[1] Pod pojmem scénář si můžeme představit jakousi šablonu, do které se následně vyplňují konkrétní data.

těchto seznamů, protože po vybrání některé z jeho položek se uživateli automaticky doplní parametry, které jsou pro akci potřebné.

Když se uživatel rozhodne vytvořit nějaký nový scénář, tak by se měl zamyslet, zda podobnou šablonu již nemá vytvořenou. Aplikace totiž umožňuje duplikaci existující šablony, kterou by si uživatel mohl upravit a následně vytvořit jednodušším způsobem.

Než začne uživatel tvořit novou šablonu, tak bych velmi doporučil ji před její editací uložit. Při každém ukládání se veškerý text validuje přes textové parsery a v případě, že je vše v pořádku, tak se data uloží. V opačném případě dojde k vyvolání chybového hlášení a šablona se tedy neuloží. Chybové hlášení obsahuje informaci, kde k chybě došlo, a tak má uživatel zjednodušenou práci při hledání chyb.

Způsob ukládání scénářů je na uvážení uživatele, ale jelikož se jedná o scénáře napříč vícero internetových obchodů, pro které platí podobné posloupnosti příkazů, pak bych doporučil seskupovat stejné typy akcí do stejných složek – například přihlášení uživatele, registraci a objednávku.

6.2 Tvorba testů

Aby mohlo dojít k vytvoření testu, pak musíme splnit jednu z následujících podmínek. První je ta, že budeme mít vytvořený scénář, ze kterého test vytvoříme a následně dodefinujeme reálné hodnoty pro symbolické konstanty.

Na začátku této práce se s druhou možností neuvažovalo, ale nakonec byla naprosto nutná, protože nebylo možné upravit parametry, které test ze scénáře získal. Jedinou možností by bylo aktualizovat scénář a vytvořit zcela nový test, při kterém by došlo ke ztrátě dosud zadaných dat. Proto byla do programu přidaná funkce úpravy testu a příkazu, které jsou dostupné z vyskakovacího okna při stisknutí pravého tlačítka myši do editační plochy. Po této akci se uživateli nabídne nové okno, ve kterém může požadované parametry dodefinovat nebo je smazat.

Postup vytváření testů

Testy se vytvářejí podobným způsobem jako scénáře. Nejprve si uživatel musí buď test vytvořit z nějaké šablony, nebo jej postupně nadefinovat. Uživateli se zde nabízí možnost využít seznamy, na které je možné kliknout a tak je doplnit do testu. Do textové plochy editoru je umožněno pouze dodefinovávat konstanty, které jsou v editoru uvedené. Žádný další text se zde vyskytovat nemůže, protože by došlo k porušení syntaktických pravidel a tím pádem k

neuložení testu. Pro editaci testů platí stejná pravidla a doporučení jako v případě editace scénářů – možnost využití seznamů, průběžné ukládání atp.

6.3 Syntaktická pravidla

Jelikož je tvorba testů i scénářů prováděna v textové podobě, pak musí být vytvořen mechanismus, který dokáže převést textový řetězec na posloupnost příkazů, které jsou předány do vyhodnocovacího mechanismu. Z tohoto důvodu musí být použity speciální znaky, které tyto jednotlivé příkazy dokáží rozpoznat. V mém programu jsem použil následující.

Znaky	**Význam**
;	Znak středníku je použit pro separaci jednotlivých příkazů, mezi které patří například přechod na internetovou stránku, zadání textu, kliknutí na element na stránce a mnoho dalšího.
,	Znak čárka je určen pro oddělení parametrů v rámci jednoho příkazu.
`##`	Zdvojený znak mřížky je určený pro uživatelský vstup. Tento typ oddělovače je jediný, který se nemusí nutně v textu vyskytovat, ale je to velmi doporučováno. V programu je nadefinovaný splitter, který rozdělí vstupní řetězec na základě speciálního znaku. Pokud se tento speciální rozdělovací znak nalézá mezi dvěma `##`, pak je tento speciální znak ignorován. Jinými slovy pokud uživatelský vstup obsahuje znak, podle kterého chceme nějaký řetězec rozdělit, pak nesmí k rozdělení dojít.

Tabulka 6.1: Tabulka syntaktických pravidel

7 Testování v praxi

V předchozích řádcích jsme si představili možnosti aplikace a nyní přichází čas si tuto aplikaci vyzkoušet. Ze zadání plyne, že musí dojít k otestování pěti internetových obchodů, kde u každého musí dojít k otestování alespoň deseti testovacích případů. Testy musejí být jak pozitivní, tak negativní. V případě negativních testů dochází k testování pouze těch případů, kdy je po uživatelovi požadován nějaký vstupní řetězec, aby se předešlo nesmyslným vstupům. Tyto internetové obchody jsou většinou určené pro zákazníky z Německa a my si je představíme na následujících řádcích.

7.1 Testované internetové obchody

- Chickie – `https://www.chickie.cz`
- Grossmann – `https://www.grossmannonline.de/`
- Technikoase – `https://www.technikoase.de/`
- Enzinger – `https://www.enzinger.com/`
- Technik Ganz Gross – `https://www.technik-gross.de`

Testovací případy

Po poradě s externím vedoucím jsme se shodli na několika činnostech, které by bylo dobré pokrýt automatizovanými testy. Některé tyto činnosti jsou pro elektronické obchody zcela běžné a jiné svým obsahem spadají do testování běžného obsahu na internetu. Tyto činnosti si následně uvedeme.

Registrace

Jedná se o jednu z nejdůležitější akcí, bez které by se internetový obchod nemohl obejít. Bez řádné registrace by nemohlo dojít ke sledování uživatelských objednávek, zasílání pravidelných nabídek a mnoho dalšího. U některých elektronických obchodů je samozřejmě možné nakupovat i bez použití registrace, ale v případě, že bude chtít uživatel objednávku na témže obchodu v budoucnosti opakovat, pak je tento krok velmi vhodný.

Přihlášení

Pokud má uživatel již vytvořený účet, pak je mu umožněno se na tento účet pomocí přihlašovacího jména a hesla pravidelně přihlašovat. Po přihlášení je uživateli ve většině případů umožněno sledovat stavy objednávek, historii objednávek a spoustu dalších služeb, které elektronický obchod poskytuje. Další samozřejmostí je, že se uživateli stane nakupování příjemnější, protože v případě provedení objednávky již nemusí vyplňovat své údaje.

Přidání zboží do košíku

Jedná se o operaci, kdy si uživatel vybere položku, o kterou má zájem, a následně si ji uloží na místo, odkud si ji může pohodlně objednat. Jedná se o jednodušší test, ze kterého vzniká mnohem náročnější test, tj. test na objednání zboží.

Objednávka

Svým rozsahem a významem patří mezi nejdůležitější testy, které se dají na elektronickém obchodě vůbec provést. Bez této funkcionality by byl totiž samozřejmě celý obchod k ničemu, protože by si zákazníci nemohli objednávat zboží, které obchod nabízí. Test se skládá z několika etap, kdy první etapa je přidání zboží do košíku. Druhá etapa je vložení všech potřebných údajů, způsob dopravy a platby, potvrzení prodejních podmínek a samotné odeslání.

Kontrola úvodní stránky

Tento test má za úkol otestovat, zda se na úvodní stránce vyskytují veškeré důležité věci, které jeho vlastník uzná za vhodné. Mezi tyto věci může patřit existence následujících komponent:

- Přihlašovací tlačítko
- Menu
- Reklamy
- Kontaktu v patičce stránky
- Vstupu do administrace elektronického obchodu
- Bloku s nejprodávanějšími produkty
- Vyhledávacího panelu

Kontaktní formulář

V případě jakéhokoli dotazu ze strany zákazníka by mu mělo být umožněno podat na elektronický obchod dotaz, ze kterého by se mu mělo dostat náležité odpovědi. Proto je dobré, aby i tato funkcionalita byla náležitě otestována.

Ostatní testovací případy

Počet kandidátů na otestování může být samozřejmě více. Předchozí uvedené tvoří základ každého elektronického obchodu, které by měli být otestované vždy. Další testovací případy můžou být například následující:

- Mazání položky z košíku
- Změna osobních údajů
- Obnova zapomenutého hesla
- Změna hesla
- Test funkčnosti vyhledávajícího panelu
- Odhlášení uživatele

7.1.1 Nasazení testů na konkrétní elektronické obchody

Každý elektronický obchod poskytuje trochu odlišné služby zákazníkům. Z toho vyplývá skutečnost, že ne všechny elektronické obchody mohou být otestovaný stejnými typy testů, ale jejich základ zůstává stejný. Některý může nabízet nějaký druh sofistikované služby, se kterou se u jiných například nepočítalo, a tak nezbývá nic jiného než tyto testy vytvořit, což není žádný velký problém. V této práci se ovšem jednalo o velmi podobné obchody, a tak těchto odlišností nastalo minimální množství.

7.2 Vzorový scénář

Počet testovacích scénářů, na kterých bychom si mohli ukázat postup automatizace, je celá řada. My si zde ovšem ukážeme pouze jeden základní, který je pro elektronický obchod velice důležitý. Na následujících řádcích je ukázáno, jak celý proces vytváření testovacího scénáře probíhá.

Konkrétně vybraný scénář se bude týkat přihlášení do internetového obchodu, což je naprosto typická akce, bez které by se internetový obchod nemohl obejít.

Tvorba testů a scénářů nemá žádné předepsané konvence zápisů a je tak jen na uživateli, jak si bude dané testy a scénáře pojmenovávat. Z hlediska přehlednosti je velmi doporučeno, aby bylo z názvu poznat, co daný scénář provádí. Pokud například narazíme na test, který se jmenuje *loginTest*, pak je zřejmé, že se jedná o test přihlášení uživatele do daného obchodu. Stejným způsobem by měli být pojmenovány prvky, se kterými se rozhodneme pracovat. Pokud budeme například požadovat interakci s prvkem, který představuje vstupní pole pro heslo uživatele, pak je dobré si jej pojmenovat například jako *inputPassword*.

Scénář pro přihlášení uživatele

Než se pustíme do tvorby testovacího scénáře, tak si musíme zapnout některý z prohlížečů a provést tento proces manuálně a všechny naše kroky zaznamenávat do scénáře, který následně uložíme. Pro manuální proces vytváření scénáře je doporučeno použít internetový prohlížeč Firefox s pluginem Firebug, který jsme si představili v sekci nástrojů pro automatizované testování. Na obrázku 7.1 můžeme vidět dokončený scénář týkající se přihlášení uživatele do internetového obchodu `www.technikoase.de`. Náhled do zdrojového kódu vzorového scénáře bude k dispozici v příloze na konci této práce. Součástí přílohy bude také uživatelská dokumentace, po jejíž přečtení bude uživatel schopen vytvořit své vlastní scénáře.

```
visit, url=##adresa##;
click, location=##login##;
type, location=##pozice mail##, content=##mail##;
type, location=##pozice heslo##, content=##heslo##;
click, location=##pozice prihlasit##;
assertElementPresent, location=##pozice muj ucet##;
```

Obrázek 7.1: Vytváření scénáře

7.3 Vzorový test

Aby mohl uživatel vytvořit test, pak musí jít jednou ze tří cest, které jsou mu umožněny. Buď využije některého z vytvořených scénářů, kde si definoval jednotlivé kroky, které udělal při konkrétní manuální operaci, nebo pomocí kliknutí pravého tlačítka myši do editačního panelu testu, kde si daný scénář postupně vytvoříme. Uživatel by měl primárně využívat první možnosti, protože druhá možnost slouží výhradně pro dodefinování akcí, na které se při tvorbě scénáře zapomnělo nebo se s nimi nepočítalo. Třetí možností je

duplikace některého z existujícího testu a následného upravení jednotlivých příkazu a vstupních hodnot.

Test přihlášení uživatele

Tento test není svojí náročností zase až tak obtížné vytvořit, ale pro správné fungování internetového obchodu je naprostou nezbytností. Jakmile si vytvoříme potřebný scénář, tak jej začneme dodefinovávat údaji, které jednoznačně identifikují jednotlivé elementy na internetové stránce a operace nad nimi. Vzorový test můžeme vidět na obrázku 7.2. Zdrojový kód vzorového testu včetně detailních informací o tvorbě testu bude uvedený v příloze.

```
##adresa##=##https://www.technikoase.de/##;
##login##=css=##span.likeLink##;
##pozice mail##=id=##l1c9c25c12c1_ctrl##, ##mail##=##cesky@software1.cz##;
##pozice heslo##=id=##l1c9c25c12c2_ctrl##, ##heslo##=##heslo##;
##pozice prihlasit##=id=##l1c9c25c12c3_ctrl##;
##pozice muj ucet##=Link=##Mein Konto##;
```

Obrázek 7.2: Vytvoření testu

7.4 Spuštění testů

Jakmile si uživatel vytvoří nějakou množinu testů pro určitý elektronický obchod, nebo libovolných internetový obsah, pak je možné tyto testy předložit testovacímu mechanismu. Způsobů jak toho docílit je více. První způsob je ten, že při editaci testu využije v tomto okně tlačítka *Přidat test*, což ho automaticky přepne do editoru s tabulkou, ve které jsou umístěny testy připravené ke spuštění. Druhou možností je využít stromové struktury, ve které si uživatel vede své testy. Po kliknutí levého tlačítka na požadovaný test nebo složku, následně zvolí možnost *Přidej do seznamu*. Pokud uživatel klikne na složku, pak budou všechny testy, které jsou v ní umístěné, přidány do tabulky pro spuštění. Poté se stačí pouze přepnout do okna s připravenými testy pomocí karty *Probíhající testy* umístěné v levém panelu a může se začít s testováním.

7.5 Diskuse o tvorbě a výsledcích testů

Na začátku vytváření programu se počítalo s tím, že se budou vytvářet scénáře, ze kterých se následně budou tvořit testy pro konkrétní elektronické obchody. V průběhu tvorby testů jsem ovšem zjistil, že mnohem jednodušší je provádět duplikace těchto testů, protože v některých případech bylo možné

použít jeden test na jiném obchodě pouze s modifikací jeho výchozí adresy. V některých případech stačilo pouze změnit identifikátory elementů na stránce, eventuálně přidat, nebo odebrat některý z příkazů.

Když jsem začínal s tvorbou testů, tak bylo zřejmé že jejich počet nebude malý, protože mělo dojít k otestování pěti elektronických obchodů. Pro každý obchod mělo být vytvořeno deset testů, které měli být jak pozitivní, tak negativní.

Jak jsem popsal v kapitole věnující se pozitivnímu a negativním testování, tak jsem nejprve vytvořil testy pozitivní, ze kterých jsem následně tvořil testy negativní. Celkový počet testů, které jsem v práci vytvořil přesáhlo hodnotu 150, čímž si myslím, že došlo k adekvátnímu otestování funkcionality jednotlivých elektronických obchodů.

Jakmile jsem měl všechny testy vytvořené, tak jsem mohl jednotlivé obchody začít testovat takovým způsobem, kvůli kterému byla aplikace vytvořena. A tak jsem zkoušel spouštět více testů najednou. Tento proces ovšem nebyl tak jednoduchý, jak by se na první pohled zdálo. Přes všechny problémy během spouštění, které si uvedeme v sekci 7.6, se mi nakonec povedlo všechny testy, které jsem vytvořil, úspěšně dokončit. Během mého testování jsem objevil pár chyb, jejichž důsledky nejsou zase až natolik závažné.

Technikoase

Nejlépe z elektronických obchodů vyšel obchod *Technikoase*. Během jeho testování došlo k úspěšnému spuštění všech testů, ovšem na dvě iterace. Doporučený způsob spouštění testů si uvedeme v sekci 7.7.

Technik Ganz Gross

Dalším testovaným elektronickým obchodem byl obchod *Technik Ganz Gross*. Při testování tohoto obchodu jsem narazil na jeden problém. Pokud jsem se chtěl přihlásit pod nesmyslným e-mailem, pak se mi jednou zobrazila varovná hláška a podruhé ne. Tak jsem vytvořil test, který testuje tuto chybovou hlášku. Pokud se tedy nezobrazí, pak nemůže dojít ke správně vykonanému testu.

Grossmann

Dále jsem tetoval obchod *Grossman*, jehož testy proběhli stejným způsobem, jako u obchodu *Technik Ganz Gross*. Všechny testy dopadli korektně, akorát negativní test na přihlášení skončil chybou.

Enzinger

Následovaly testy pro elektronický obchod *Enzinger*, u kterého nastali drobné problémy, které šlo vyřešit pouze snížením počtu paralelně spuštěných testů, jejichž většina musela být spuštěna v reálném prohlížeči. Důvod si vysvětlíme v sekci 7.6. Negativní test na přihlášení ovšem proběhl korektně. Nakonec ovšem došlo k úspěšnému spuštění veškerých testů, a tak nebyla na elektronickém obchodě nalezena žádná chyba.

Chickie

Posledním testovaným elektronickým obchodem byl obchod *Chickie*. U testování tohoto obchodu se mi ovšem objevil jeden problém, který se projevoval při přidávání položky do košíku. Problém byl při výběru správné velikosti a barvy zboží, protože tyto položky byly do stránky přidány pomocí technologie `AJAX` a testovací mechanismus nemohl tyto položky vyhledat. Jediné řešení tohoto problému bylo nastavit před kliknutím na tyto položky pevně stanovené časové zpoždění a teprve potom se pokusit na tyto položky kliknout.

Při testování tohoto obchodu jsem našel dvě chyby, které se vyskytují při registraci uživatele. Pokud nedojde k vyplnění telefonního čísla, které je povinné, pak se zobrazí chybová hláška obsahující zprávu, že zadaná e-mailová adresa je již používána. Jedná se pouze o špatně nastavený validátor, což nezpůsobuje žádné velké problémy. Dalším problémem bylo, že když uživatel nezadal korektní poštovní směrovací číslo, což byl povinný údaj, pak jej to zaregistrovalo a toto „číslo“ se uložilo. Ostatní obchody v takovém případě registraci neprovedli.

7.6 Úskalí

V průběhu testování jsem se potýkal s několika nedostatky, které bych chtěl popsat následovně.

Spouštění testů v reálných prohlížečích

Během mého testování na reálných prohlížečích jsem narazil na jeden problém, který ovšem není zase až tak závažný, ale chtěl bych se o něm zmínit. Uživatel má pomocí volby *Počet spuštěným testů* z menu možnost si vybrat počet testů, které budou spuštěny paralelně. Toto tvrzení není zase až tak pravdivé, protože ve skutečnosti není možné v jeden krátký okamžik vytvořit libovolný počet instancí prohlížeče Firefox. Proto je nutné testy spouštět

jeden po druhém dokud nedojde k řádnému vytvoření okna. V programu jsem zkoušel vytvářet tyto instance pomocí speciálního vlákna, ale s tímto řešení mi nastaly problémy, protože se několik testů vůbec nespustilo.

Z důvodu postupného spouštění jednotlivých testů tedy není možné, aby všechny testy byly prováděny najednou. Jelikož vytvoření okna trvá několik sekund, hodnota závisí na konkrétním zařízení, tak jsem zjistil, že maximální počet testů, které jsou prováděny paralelně se zastavil na čísle čtyři. Z toho tedy vyplývá, že průměrný test trvá tak dlouho, za kterou se spustí instance Firefoxu čtyři krát.

Proto se hodnota, kterou si uživatel definoval jako maximální počet paralelně probíhajících testů, považuje spíše za horní hranici, než za skutečný počet spouštěných testů.

Prohlížeče bez grafického rozhraní

Pokud se rozhodneme provádět testování v tomto prohlížeči, pak se nám může stát, že testy, které byly úspěšné na reálném prohlížeči, skoční chybou. Je to způsobeno tím, že tyto prohlížeče nemají tak propracovaný mechanismus pro zpracování `JavaScriptu` a `AJAXU` jako reálné prohlížeče. Tyto chyby se projevují nenalezením požadovaného elementu na HTML stránce.

Výhoda toho prohlížeče oproti předchozímu je ovšem ta, že opravdu dojde k vytvoření požadovaného počtu paralelně probíhajících testů, což ovšem může přinášet jisté nebezpečí. Čím více testů je spuštěno paralelně, tím více paměti samozřejmě spotřebuje, a tak závisí na konkrétním zařízení zda je schopno tuto paměť dodat. Při testování aplikace jsem ovšem zjistil, že s rostoucím počtem paralelních testů, roste i jejich chybovost, tudíž se musí s tímto parametrem velice opatrně. Každopádně je zde možnost tímto způsobem výrazně snížit počet celkových testů, protože projde jejich většina, a pak spustit testování znovu s menším počtem paralelně probíhajících testů, které skončily chybou.

Captcha

Jedná se o mechanismus, který se používá na internetu ve snaze odlišit skutečné uživatele od robotů[1]. Použití captchy[2] spočívá v tom, že lidský mozek dokáže správně rozeznat i deformovaný obrázek, zatímco internetový roboti při použití technologie OCR[3] nikoli. Tento zabezpečovací mechanismus lze v mém případě obejít dvěma způsoby:

- V prvním případě je možné nastavit v testu malé **časové zpoždění**, ve kterém by uživatel tento bezpečnostní kód zadal.
- Ve druhé možnosti se může vyskytovat potenciální hrozba pro celý systém. Tester se může dohodnout s administrátorem na nějakém způsobu, kterým tento **mechanismus vyřadí**. Může to být například změnou některého z parametru nebo zakódovaní nějaké informace do zobrazeného obrázku. Tohoto faktu ovšem může využít i nějaký útočník, který se o této možnosti dozví.

V mém případě jsem se musel rozhodnout pro první možnost, a tak uživatel, který bude testy spouštět, bude mít možnost tento kód zadat. Samozřejmostí je, že si bude moci tento časový interval pro vložení kódu nastavit na hodnotu, jakou uzná za vhodné. Pro tento případ bych doporučil, aby si uživatel nastavil počet paralelně probíhajících testů na hodnotu jedna, čímž se mu bude automaticky spouštět jeden test po druhém.

Toto řešení může způsobit korektní vyhodnocení testu, který by měl ovšem skočit chybou. Pokud například testujeme neuvedení e-mailu, pak chybová hláška může být stejná, jako v případě nevyplnění captcha kódu. Tím pádem dojde ke správnému vyhodnocení díky nezadanému captcha kódu, nikoli kvůli neuvedení e-mailové adresy.

7.7 Nabyté zkušenosti během testování

Při tvorbě testů jsem se potýkal s různými problémy, které bych chtěl nyní uvést, aby se jim případní uživatelé této aplikace vyhnuli. Po přečtení následujících řádek se tak stane pro uživatele testování mnohem pohodlnější a uspoří tím spoustu času.

[1]Počítačový program, který pro svého majitele opakovaně vykonává nějakou rutinní činnost na internetu

[2]Jedná se o akronym pro „Completely Automated Public Turing test to tell Computer and Humans apart“

[3]Optical Character Recognition (Optické rozpoznávání znaků)

Tvorba testů

Jakmile se uživatel rozhodne vytvořit si některý z testů, pak bych jej doporučil spouštět na reálné prohlížeči. Pokud se během vyhodnocování vyskytují nějaké problémy, se kterými si uživatel neví rady, pak bych doporučil použití časového zpoždění, aby uživatel přesně viděl, jak se jeho kroky postupně provádějí a ve které fázi chyba nastane. Chyba může nastat při vyhledávání elementu na HTML stránce, který ovšem není dosud dostupný. Když testy s časovým zpožděním proběhnou úspěšně, pak je dobré u příkazu, který způsoboval pád testu, definovat časové zpoždění.

Spouštění testů

Pokud se uživatel rozhodne otestovat konkrétní elektronický obchod kompletní sadou testů, které byly pro něj vytvořeny, pak bych doporučil prvotní spuštění prostřednictvím prohlížeče bez grafického uživatelského rozhraní. Výhoda tohoto prohlížeče je ta, že je schopen spustit více testů najednou, jejich provedení je výrazně rychlejší a že nejsme prakticky omezeni počtem testů, které chceme paralelně spustit. Jediné omezení je v dostupné operační paměti daného zařízení.

Jakmile doběhnou všechny testy, které byly spuštěné přes prohlížeč bez grafického rozhraní, tak se počítá s tím, že ne všechny budou úspěšně provedené. Důvod jsme si uvedli v sekci 7.6. Poté bych doporučil využít funkcionality tlačítka *Odebrat dokončené*, což smaže všechny úspěšné testy a následně bych provedl otestování zbytku testů na reálném prohlížeči. Pokud i v tomto případě nějaký test dopadne chybně, pak bych doporučil nastavit počet paralelně probíhajících testů na hodnotu jedna, čím dojde ke zpracovávání jednoho testu po druhém.

8 Závěr

Hlavním cílem bakalářské práce bylo vytvořit aplikaci, která by byla schopna otestovat základní funkcionalitu libovolného internetového obchodu. Pro tento účel jsem prozkoumal několik nástrojů, které jsou pro tento účel vytvořeny, a následně jsem je použil v mém programu. Aplikace funguje na základě vytvoření několika základních scénářů, které se později dají aplikovat na konkrétní internetové stránky dosazením konkrétních hodnot. Tyto hodnoty určují například prvky, se kterými chceme interagovat, vstupní řetězce, které chceme do daných elementů zadávat atp. Tímto způsobem dochází k vytváření testovacích příkazů, z nichž se skládají výsledné testy.

Mezi přínosy práce tedy patří zejména možnost jednoduché tvorby testů, které se následně jednoduše spouští a poskytují uživateli zpětnou odezvu o jejich provedení. Program lze samozřejmě využít i pro testování libovolného internetového obsahu a nemusí se tak jednat pouze o internetové obchody.

Dalším přínosem této práce bylo seznámení čtenáře v oblasti obecného testování, které bylo popsáno v samostatné kapitole týkající se základy testování softwaru.

Jelikož testování webu je velice široké odvětví, pak se v programu může naskytnout situace, že nějaké požadované scénáře nebude možno zkonstruovat. Základ tohoto problému je v tom, že se s danými akcemi v průběhu tvorby programu nepočítalo. Každopádně pokud by tato situace nastala, pak není problém tuto funkcionalitu do programu jednoduše doplnit a následně ji využívat.

Závěrem bych chtěl říci, že podle mého názoru i názoru externího zadavatele práce, byla vytvořena aplikace, která je schopna vytvářet dostatečně sofistikované typy testů a svým obsahem je zcela dostačující.

9 Přehled zkratek

Zkratka	Význam
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CSRF	Cross site request forgery
CSS	Cascading Style Sheets
FPD	Full Path Disclosure - odkrytí cesty
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JAR	Java Archive
JDOM	Java Document Object Model
Junit	Java unit
OCR	Optical Character Recognition
RC	Remote Control
SQL	Structured Query Language
UML	Unified Modeling Language
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WatiJ	Web Application Testing in Java
XML	Extensible Markup Language

Tabulka 9.1: Přehled zkratek

Literatura

[1] *Enterprise Web Development: Building HTML5 Applications: From Desktop to Mobile*. O'Reilly Media, 2014. ISBN 978-1-449-35681-1.

[2] *Introduction to Selenium* [online]. [cit. 30.3.2016]. Dostupné z: `http://www.guru99.com/introduction-to-selenium.html`.

[3] *Watij – Web Application Testing in Java* [online]. [cit. 7.4.2016]. Dostupné z: `http://watij.com/`.

[4] TESTING, W. [online]. [cit. 28.3.2016]. Dostupné z: `http://www.tutorialspoint.com/software_testing_dictionary/web_application_testing.htm`.

[5] HELP, S. T. *Web Testing: Complete guide on testing web applications* [online]. 19.11.2015. [cit. 15.2.2016]. Dostupné z: `http://www.softwaretestinghelp.com/web-application-testing/`.

[6] HLAVA, T. *Fáze a úrovně provádění testů* [online]. 21.8.2011. [cit. 29.3.2016]. Dostupné z: `http://testovanisoftwaru.cz/category/metodika-testovani/druhy-typy-a-kategorie-testu/`.

[7] HLAVA, T. *Druhy, typy a kategorie testů* [online]. 21.8.2011. [cit. 20.3.2016]. Dostupné z: `http://testovanisoftwaru.cz/tag/akceptacni-testovani/`.

[8] KUNDU, S. *Web Testing: Tool, Challenges and Methods* [online]. 2012. [cit. 23.1.2016]. Dostupné z: `http://ijcsi.org/papers/IJCSI-9-2-3-481-486.pdf`.

[9] NGUYEN, H. Q. – JOHNSON, B. – HACKETT, M. *Testing Applications on the Web: Test Planning for Mobile and Internet-Based Systems*. Wiley, 2003. ISBN 0471201006.

[10] PADOLINA, L. *Web Testing* [online]. [cit. 12.12.2015]. Dostupné z: `http://www.edb.utexas.edu/minliu/multimedia/PDFfolder/WebTestingPadolina.pdf`.

[11] PATTON, R. *Testování softwaru*. Computer Press, 2002. ISBN 80-7226-636-5.

[12] ROBSON, E. – FREEMAN, E. *Head First HTML and CSS*. O'Reilly Media, Inc., 2012. ISBN 978-0-596-15990-0.

[13] Václav Šourek, J. l. a. R. H. J. t. *E-shop a faktory jeho úspěšnosti* [online]. 15.5.2007. [cit. 28.3.2016]. Dostupné z: `http://www.asociace.biz/files/eshopyv14.pdf`.

[14] Wikipedia. *Cross-site request forgery* [online]. Wikipedia, the free encyclopedia, 14.3.2016. [cit. 5.4.2016]. Dostupné z: `https://en.wikipedia.org/wiki/Cross-site_request_forgery`.

[15] Wikipedia. *Web Application* [online]. 26.11.2015. [cit. 3.12.2015]. Dostupné z: `https://en.wikipedia.org/wiki/Web_application`.

A Zdrojový kód scénáře

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario name="chickie - Prihlaseni.xml">
  <command action="visit">
    <url>
      <label>chickie url</label>
    </url>
  </command>
  <command action="click">
    <location>
      <label>pozice prihlasit</label>
    </location>
  </command>
  <command action="type">
    <location>
      <label>pozice jmeno</label>
    </location>
    <content>
      <label>login</label>
    </content>
  </command>
  <command action="type">
    <location>
      <label>pozice heslo</label>
    </location>
    <content>
      <label>heslo</label>
    </content>
  </command>
  <command action="click">
    <location>
      <label>pozice prihlasit</label>
    </location>
  </command>
</scenario>
```

B Zdrojový kód testu

```
<test name="chickie - Prihlaseni.xml" status="OK">
  <command action="visit" status="OK" message="V poradku">
    <url>
      <label>adresa</label>
      <content>https://www.chickie.cz</content>
    </url>
  </command>
  <command action="click" status="OK" message="V poradku">
    <location>
      <label>login</label>
      <PartialLink>Prihlaseni / Registrace</PartialLink>
    </location>
    <wait>
      <label>cekej</label>
        <threadSleep>500</threadSleep>
    </wait>
  </command>
  <command action="type" status="OK" message="V poradku">
    <location>
      <label>pozice mail</label>
      <id>l1c4c5c1c1c2c2c1c2c2_ctrl</id>
    </location>
    <content>
      <label>cesky@software1.cz</label>
      <content>cesky@software1.cz</content>
    </content>
  </command>
  <command action="type" status="OK" message="V poradku">
    <location>
      <label>pozice heslo</label>
      <id>l1c4c5c1c1c2c2c1c2c3_ctrl</id>
    </location>
    <content>
      <label>heslo</label>
      <content>heslo</content>
    </content>
```

```
  </command>
  <command action="click" status="OK" message="V poradku">
    <location>
      <label>pozice prihlasit</label>
      <id>l1c4c5c1c1c2c2c1c2c4_ctrl</id>
    </location>
  </command>
</test>
```

C Uživatelská dokumentace

V této části si popíšeme několik důležitých věcí, které jsou nutné pro bezchybný chod aplikace. Dále si zde ukážeme způsob, kterým se aplikace ovládá, aby byl uživatel schopen tuto aplikaci bez omezení využívat.

Spuštění programu

Tento nástroj nevyžaduje pro své spuštění žádný sofistikovaný software. Jediné co musí pro bezchybný chod aplikace splnit je to, že musí mít na svém počítači nainstalovanou Javu ve verzi 1.8 nebo vyšší a internetový prohlížeč Mozilla Firefox.

Spuštění aplikace se provede jednoduše kliknutím na spustitelný JAR[1] soubor. Podmínkou správného spuštění je ta, že u tohoto souboru musí být umístěna složka „files“, ve které jsou obsaženy veškeré zdrojové soubory testů a scénářů, které program dokáže obsluhovat. Pokud tyto soubory nebudou k dispozici, pak se program spustí, ale nebudou v něm obsaženy žádná testovací data. V této složce jsou dále umístěny databázové soubory sloužící k naplnění obsahu pomocných seznamu, které jsou využívány jak pro tyto pomocné seznamy, tak pro validaci uživatelského vstupu, které postupně zadávají do editorů, a testovací mechanismus. Tyto seznamy jsou podle potřeby využívány napříč celou aplikací. Pokud tyto databázové soubory neexistují, pak se sice aplikace spustí, ale je zcela nepoužitelná.

Systémové požadavky

Výsledná aplikace je primárně určena pro operační systém Windows, pod kterým byla také vyvíjena. K jejímu provozu je nutné, aby na koncovém zařízení byla nainstalována Java ve verzi 1.8 a internetový prohlížeč Mozilla Firefox. Na hardware žádné požadavky kladeny nejsou.

Ovládání aplikace

Ovládání aplikace je velmi intuitivní. Popis veškerého grafického rozhraní jsme si již uvedli v předchozích kapitolách, a tak si nyní představíme pouze způsob, jakým se dané části programu ovládají.

[1] Archivační souborový formát pro platformu Java

Tvorba scénáře

Jak bylo řečeno výše - scénář je pouze jakási šablona, na jejíž základě později vytváříme testy, do který doplníme konkrétní hodnoty. Pro její tvorbu máme dostupné dva seznamy, kdy jeden obsahuje posloupnost akcí, které můžeme na HTML stránce provést, a druhý obsahuje seznam parametrů, které tyto akce přijímají.

Scénáře mohou být tvořeny dvojím způsobem. Prvním způsobem je ten, že budeme do těchto seznamů klikat. Výhoda tohoto postupu je ta, že v případě kliknutí na požadovanou akci se nám doplní její potřebné parametry, a tak uživatel nemusí vědět, které daná akce vyžaduje. Druhý způsob je ten, že uživatel těchto seznamů nevyužije, čímž ale může dojít k chybám ve vyhodnocovacím mechanismu.

Tvorba scénáře ovšem nemůže být nahodilá, ale musí splňovat požadovaná syntaktická pravidla, která si uvedeme na závěr kapitoly o ovládání aplikace.

Oba tyto seznamy, včetně jejich významu si následně uvedeme. V případě typů akcí (viz Tab. C.1) si nebudeme uvádět všechny, protože je jejich množství poměrně velké a navíc některé z nich přináší podobnou funkcionalitu, kde je zřejmé, v čem se příkazy liší. Příklad takové podobnosti nabízí například příkaz „assertTextPresent“ a „assertTextNotPresent“, kdy první testuje, zda se text na stránce shoduje se zadaným textem, a druhý testuje zda je text rozdílný. V tabulce C.2 máme možnost vidět veškeré dostupné parametry, které je možno do scénářů přidávat.

Jakmile se uživatel vytvoří šablonu podle jeho potřeby, pak si ji může uložit a následně pomocí tlačítka „test“ použít. V průběhu vytváření šablony je velmi doporučeno, aby byla průběžně ukládána, čímž se zamezí potenciálním chybám a problémům během ukládání.

Zkratka	**Parametry**	**Význam**
assertElementPresent	location count	Kontrola počtu elementů na dané pozici.
assertTextPresent	location content	Akce, která zjistí zda se na dané pozici nalézá text.
assertTitle	content	Kontrola titulky stránky.
click	location	Akce, která provede kliknutí na daný element.
clickAndWait	location	Provede akci kliknutí a následně čeká specifikovaný čas.
dropDownSelectIndex dropDownSelectText dropDownSelectValue	location content	Vybere položku ze seznamu na základě jeho indexu, textu, nebo atributu value.
exists	location	Akce rozhodne, zda daný prvek na stránce je, nebo není.
existsCount	location count	Rozhodnutí, zda je na zadaném místě určitý počet elementů.
changeHtml	location content	Umožní modifikaci atributu u daného elementu.
navigate	direction	Akce provádějící navigaci v rámci prohlížeče
sleep	count	Akce pro pozdržení testu.
submit	location	Akce, která provede odeslání formuláře
switchToFrame	location	Akce, ve které se přepneme z aktuálního okna do rámu
switchToDefault		Akce pro přepnutí z rámu do výchozího okna.
type	location content	Vložení textu na danou pozici.
typeAndSubmit	location content	Vložení textu na danou pozici a současným odeslání vyplňovaného formuláře.
visit	url	Akce umožňující přechod na internetovou stránku.
verifyTextPresent	content	Test existence textu na stránce.

Tabulka C.1: Tabulka typů akcí

Zkratka	**Význam**
LOCATION	Parametr, který určuje pozici na stránce, se kterou chce uživatel pracovat. Hodnoty, které se do tohoto parametru dají dosazovat naleznete v Tab. C.3.
URL	Parametr, do kterého vstupuje URL adresa.
COUNT	Tento parametr slouží uživateli pro vstup číselné hodnoty, například pro zjištění existence určitého počtu prvků na stránce.
CONTENT	Univerzální parametr, který se používá v případě, že se nehodí žádný z ostatních parametrů. Používá především pro textový vstup, ať už do vstupních polí, nebo pro změnu hodnoty atributu v nějakém elementu
DIRECTION	Parametr pro navigaci na stránce. Hodnoty, které se do tohoto parametru dají dosadit jsou v Tab. C.5.
VARIABLE	Parametr určený pro ukládání dočasné hodnoty, například textu. Pokud bychom chtěli uložit nějakou hodnotu, kterou bychom chtěli později využít.
WAIT	Parametr sloužící pro definici explicitního čekání. Doporučené použití v případě nenalezení prvku na stránce, který by se tam ovšem vyskytovat měl. Neviditelnost může být způsobena jeho prozatímním nezobrazením. Hodnoty pro tento parametr jsou uvedené v Tab. C.4.

Tabulka C.2: Tabulka typů parametrů

Na následující části HTML kódu si můžeme demonstrovat, jak by vypadala tvorba scénáře pro zapsání textu do libovolného vstupního pole. Ukázkový HTML kód je zcela smyšlený a má představovat výňatek kódu ze stránky.

```
.
<div id="form">
   <div class="block">
      <input name="input-name">
      <input name="input-surname">
   </div>
</div>
.
```

Pokud bychom od scénáře požadovali zápis textu do elementu `input` s atributem `name="jmeno"`, pak bychom ve scénáři definovali následující příkaz. Hodnoty `pozice jmeno` a `jmeno` jsou libovolné konstanty, které se zobrazí pro doplnění v editoru testu.

```
type, location=##pozice jmeno##", content=##jmeno##;
```

Tvorba testu

Při tvorbě testů může uživatel postupovat třemi způsoby, které byly představeny v kapitole zabývající se tvorbou testů. Buď využije některé z připravených šablon, nebo bude vytvářet test i scénář průběžně kliknutím pravého tlačítka myši do editoru testu. Třetí možností je duplikace některého z existujících testů. My si ovšem popíšeme pouze první přístup, protože se zde počítá s již vytvořenou šablonou. Při tvorbě testů musejí být samozřejmě dodržována, stejně jako v případě tvorby scénářů, syntaktická pravidla.

Při vyplňování testu má uživatel možnost využívat seznamů, které jsou v tomto editoru zobrazeny v pravé části. Pokud si uživatel například definoval ve scénáři interakci s nějakým prvkem na stránce, pak si u tohoto příkazu definoval příkaz `location`. Tento příkaz tedy vyžaduje, aby byl při vytváření testu doplněn jeho konkrétní hodnotu, tj. pozicí na stránce. Pozici prvku na stránce můžeme určit několika způsoby. Tyto možnosti jsou uvedeny v tabulce C.3.

Zkratka	**Význam**
ID	Určení pozice pomocí identifikátoru ID.
CLASS	Lokalizace pomocí atributu class.
XPATH	Lokalizace díky mechanismu Xpath.
CSS	Určení elementu pomocí jeho css selektoru
Link	Nalezení odkazu, jehož text se zcela shoduje s hodnotou, která je určena tímto identifikátorem.
PartialLink	Tento typ využívá stejného principu jako předchozí, s tím rozdílem, že je zde vyžadována pouze částečná shoda námi zadaného textu s textem zobrazeným na stránce.
NAME	Určení prvku podle atributu name.
TAG	Nalezení prvku podle typu elementu.

Tabulka C.3: Tabulka lokátorů

Nyní jsme si přestavili možnosti, kterými lze vyhledat prvek na stránce. Příklad, jakým se dají tyto lokátory využít, si můžeme demonstrovat na stejném HTML kódu, který jsem uvedl u tvorby scénáře. Proto si zde zmíníme jak se akce, kterou jsme definovali pro tvorbu scénáře promítne v tvorbě testu.

```
.
<div id="form">
  <div class="block">
    <input name="input-name">
    <input name="input-surname">
  </div>
</div>
.
```

Zde si uvedeme, jak vypadá test (druhý příkaz) vytvořený z předešlého scénáře (první příkaz):

```
type, location=##pozice jmeno##, content=##jmeno##;
##pozice jmeno##=name=##input-name##, ##jmeno##=##Pavel##;
```

V editaci testu se vyskytují ještě další, neméně důležité seznamy, které může uživatel při své tvorbě využít. Jedním z nich je seznam obsahují konstanty explicitního čekání (viz Tab. C.4), které slouží pro pozdržení vyhodnocovacího mechanismu v případě prozatímního nenačtení hledaného elementu. V programu je sice použit mechanismus implicitního čekání, který na požadovaný element čeká, ale může nastat situace, že bude tato čekací doba nedostatečná. Proto bylo explicitní čekání implementováno.

Pokud chceme využít funkcionality navigace uvnitř HTML stránky, pak využijeme parametr `DIRECTION`, jehož konkrétní hodnoty mohou nabývat dvou hodnot (viz Tab. C.5).

Syntaktická pravidla

Při psaní scénářů a testů je důležité, aby jejich tvůrce dodržoval pravidla pro psaní zdrojového textu. Těchto pravidel není mnoho a my si je nyní představíme:

- **Oddělování příkazů** - Všechny příkazy, které jsou do jednotlivých editorů zadávány musí být jednoznačně **odděleny znakem středník (;)**.

Typ	**Význam**
threadSleep	Jedná se o základní typ čekání, kdy dojde k uspání vlákna, které příkaz zpracovává.
elementToBeClickable	V tomto případě aktivně čekáme, dokud nebude možno na daný prvek kliknout.
presenceOfAllElementsLocatedBy	Pokud jsem jsme použili takový lokátor, kterému odpovídá více prvků, pak toto explicitní čekání bude probíhat tak dlouho, dokud nebudou všechny prvky přítomny na stránce.
presenceOfElementLocated	Jedná se o stejný typ čekání jen s tím rozdílem, že se čeká pouze na jeden prvek.
visibilityOfAllElementsLocatedBy	Jedná se opět o aktivní čekání, které pomine v případě, když jsou všechny prvky na stránce viditelné.
visibilityOfElementLocated	Stejný typ jako předchozí s tím rozdílem, že se jedná o jeden prvek.

Tabulka C.4: Tabulka explicitního čekání

Typ	**Význam**
back	Přechod zpět v historii navigace.
forward	Přechod vpřed v historii navigace.

Tabulka C.5: Tabulka pro navigaci na stránce

- **Oddělování parametrů v příkazu** - Mezi veškeré parametry, které jsou v editorech zapsané, se musí vyskytovat **znak čárky (,)**
- **Uživatelský vstup** - Jedná se o pravidlo, které se nemusí dodržovat, ale je velice doporučeno. Veškerý uživatelský vstup by se měl uvádět **mezi dvojici znaků ##**. Pokud se tímto pravidlem nebudeme držet, pak může nastat stav, kdy uživatel uvede jeden z řídících znaků uvnitř svého vstupu. Pak samozřejmě dojde ke špatnému vytvoření testovací příkazů a tím pádem i k negativnímu vyhodnocení celého testu.

Klávesové zkratky

Při tvorbě testů a scénářů si může uživatel zpříjemnit práce použitím několika zkratek, které editory poskytují. Proto si uvedeme tabulku, které zkratky jsou ve kterém editoru dostupné (viz Tab. C.6).

Zkratka	**Podporující editor**	**Funkce**
`CTRL + A`	Editor testu	Přidání testu do tabulky testů připravených ke spuštění.
`CTRL + S`	Editor testu Editor scénáře Úprava testu	Uložení obsahu z editoru do uzlu. Nejedná se o uložení do souboru.
`ALT + R`	Tabulka připravených testů	Spustí veškeré testy, které jsou v tabulce připraveny

Tabulka C.6: Tabulka klávesových zkratek