VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2015

Petr Večeřa

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Petr Večeřa**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: profiq s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Miloš Kudělka, Ph.D.**

Konzultant bakalářské práce: Ing. Rastislav Kanócz

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 4. května 2015 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. května 2015 .....

Rád bych poděkoval vedení firmy profiq s.r.o za poskytnutí možnosti vykonávat bakalářskou praxi v této společnosti. Jmenovitě bych rád poděkoval Ing. Rastislavovi Kanócz za jeho podporu a vedení v průběhu praxe a také za jeho cenné rady při zpracovávání této práce.

Abstrakt

Tato bakalářská práce popisuje průběh odborné praxe ve společnosti profiq s.r.o. Na začátku práce je popsána firma, pracovní zařazení studenta a seznam jemu zadaných úkolů. Dále je popsáno jejich řešení daných úkolů. Hlavním úkolem studenta byl vývoj automatických testů pro JavaScirptový framework ExtJS. V práci jsou také popsány jednotlivé problémy, které vznikly v průběhu vykonávání zadaných úkolů a zvolený způsob jejich řešení. Ke konci práce jsou shrnuty uplatněné znalosti získané v průběhu studia a taktéž znalosti, které studentovi chyběly. V závěru je pak celkové zhodnocení praxe.

Klíčová slova: odborná praxe, JavaScript, Python, testování, zajištění kvality, ExtJS, Sencha, Siesta, HTML, CSS, Selenium Webdriver

Abstract

This bachelor thesis describes the course of profesional practice in the company profiq s.r.o. The first part of the thesis describes the company, job description of the student and lists the tasks perfomed by the student. The main task of the student was the creation of automatized tests for JavaScript framework ExtJS. Thesis also contains the description of various problems, which risen during the performing of said tasks and their solutions. Towards the end of the thesis is summarized the knowledge gained during the school studies and also the knowledge student was missing. In the end is the overall evaluation of the professional practice.

Keywords: professional practice, JavaScript, Python, testing, quality assurance, ExtJS, Sencha, Siesta, HTML, CSS, Selenium Webdriver

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CSS	–	Cascading Style Sheets
HTML	–	HyperText Markup Language
HTTP	–	Hypertext Transfer Protocol
IE	–	Internet Explorer
MVC	–	Model View Controller
MVVM	–	Model View ViewModel
OOP	–	Objektově orientované programování
URL	–	Uniform Resource Locator
VCS	–	Version control systems
XHR	–	XML Http Request
XML	–	Extensible Markup Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato bakalářská práce popisuje mou odbornou praxi ve společnosti profiq s.r.o. Uvedená společnost se v době mého nástupu zabývala především poskytováním služeb v oblasti testování softwaru. Ve firmě se uplatňuje při testování vysoká míra automatizace, což je oblast, o kterou jsem se zajímal. A také jeden z důvodů, proč jsem si vybral právě tuto firmu.

V následujícím dokumentu popisuji práci, kterou jsem vykonával v rozsahu dnů připadajících na bakalářkou praxi. Mým hlavním úkolem byl vývoj automatických testů pro JavaScriptový framework ExtJS od společnosti Sencha. V bakalářské práci popisuji dílčí úkoly, které jsem musel vykonat, a především problémy, jenž bylo třeba vyřešit. Jako první bylo zapotřebí se podrobně seznámit s produktem. Následoval výběr nástroje pro automatizaci a vývoj automatických testů. Věnuji se také problému, kdy jsem při testování jedné z částí frameworku nemohl použít vybraný nástroj pro automatizaci. Pro tuto konkrétní situaci jsem musel vyvinout náhradní řešení za pomoci jiných nástrojů. V neposlední řadě také popisuji exekuci daných testů a shrnuji výsledky mé práce.

Vzhledem k omezenému rozsahu dokumentu popisujícímu praxi, se zaměřuji především na rozbor práce vývoje automatických testů. Kromě toho byla celá řada dalších znalostí a dovedností, které jsem si při praxi osvojil a které v dokumentu popisuji jen okrajově bez větších detailů. Takovými byly například každodenní práce a přispívání procesu agilního vývoje softwaru metodikou Scrum, komunikace s rodilými mluvčími v angličtině, ať už v ústní nebo písemné podobě a práce v týmu.

V rozsahu mé bakalářské praxe jsem na projektu testování ExtJS pracoval sám, za pomoci mého vedoucího. Ve firmě jsem se následně podílel na několika dalších projektech, avšak již jako běžný zaměstnanec.

2 Zaměření firmy profiq s.r.o. a pracovní zařazení studenta

2.1 Společnost profiq s.r.o.

Firma profiq [1] poskytuje služby softwarového inženýrství z České Republiky pro softwarové společnosti v Silicon Valley, které vyvíjejí software agilním způsobem. Tyto služby zahrnují vývoj, testovaní a údržbu softwaru.

Služba je poskytovaná ve více formách. Od té nejjednodušší, kterou je poskytnutí jednotlivých specialistů pro zákazníka, přes komplexnější, kterou je poskytnuti celého týmu až po kompletní zastřešení dodávky včetně inženýrských zdrojů a managementu. Expertíza firmy spočívá v oblasti webových a mobilních aplikaci, cloudu a bezpečnosti.

Mezi firemní zákazníky patří například společnost Forgerock, poskytující kompletní softwarové řešení pro správu identit a přístupů. Firma Sencha, která se zabývá mimo jiné vývojem JavaScriptového frameworku ExtJS a kompletního řešení Space pro management a distribuci převážně mobilních aplikací v rámci firemního prostředí. Dalším zákazníkem je nově vznikající projekt Frame stejnojmenné společnosti.

2.2 Pracovní zařazení studenta ve firmě profiq s.r.o.

Ve firmě jsem pracoval na pozici softwarového inženýra. V rámci rozsahu mé bakalářské praxe mi byl přiřazen nový projekt pro firmu Sencha v rámci týmu pro testování softwaru a zajištění kvality. Společnost Sencha vyvíjí velice úspěšný JavaScriptový framework s názvem Ext JS pro vývoj HTML 5 aplikací, jak je představeno níže v kapitole 4.1.

Problémem bylo, že proces testování frameworku byl zcela manuální a tím pádem zdlouhavý a nákladný. Firma se o jeho vyřešení pokoušela již několik let. Mým úkolem bylo prozkoumat možnosti, jak využít automatických testů k zefektivnění procesu, zanalyzovat jakými nástroji a programovacími jazyky toho nejlépe docílit a vyvinout prototyp. Záměrem pak bylo použít analýzu a prototyp k výběru strategického řešení, které bude používáno dlouhodobě.

3 Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti

3.1 Seznámení s produktem ExtJS od společnosti Sencha

Mým prvním úkolem bylo seznámit se s JavaScriptovým frameworkem ExtJS od společnosti Sencha. Zjistit si, co produkt nabízí a jaké existují jeho verze. S čím vším je kompatibilní, co je potřeba k jeho nasazení a další informace, které jsou důležité pro vytvoření obecného přehledu o produktu.

Následně bylo také mým úkolem prakticky se seznámit s jednotlivými komponenty frameworku, prozkoumat uživatelskou dokumentaci a vyzkoušet si prakticky implementovat základní prvky.

3.1.1 Vyjádření časové náročnosti

Jelikož součástí úkolu bylo i praktické seznámení, kde bylo zapotřebí řešit nečekané problémy, byl na tento úkol vyhrazeno 5 pracovních dnů.

3.2 Manuální průzkumné testování

Před zahájením automatizace bylo zapotřebí, abych si vyzkoušel vykonávání všech testovacích případů manuálně, abych zjistil, jak se jednotlivé komponenty chovají v konkrétních případech. Na základě zjištěných údajů jsem měl za úkol navrhnout nové a popřípadě vylepšit stávající testy. Důležitou součástí bylo zjistit kvalitu funkčnosti jednotlivých komponent a to především proto, abych mohl následně určit, které části frameworku má smysl automatizovat. Popřípadě také určit místa, která by po opravách mohla určitým způsobem změnit chování aplikace, což by mělo dopad na funkčnost automatického testu.

3.2.1 Vyjádření časové náročnosti

ExtJS je poměrně rozsáhlý framework a tak průchod všech testů společně s analýzou komponent není rychlá záležitost. Na tento úkol bylo proto vyhrazeno 10 pracovních dnů.

3.3 Začátek automatizace

Automatizace jednotlivých testů byla hlavní náplní mé práce. Avšak než jsem začal, bylo zapotřebí vybrat vhodný nástroj pro automatizaci. Po výběru vhodného nástroje bylo mým úkolem naučit se v něm pracovat. Zjistit jeho možnosti, a především začít vyvíjet automatizované testy. Požadavkem bylo, aby automatizované testy vykonávaly všechny předem definované testovací případy a bylo možné spustit tyto testy na všech

podporovaných prohlížečích. Mým úkolem bylo navrhnout nastavení testovacího frameworku a strukturu kódu. O mém postupu jsem informoval vedoucího mého projektu a popřípadě jsem s ním konzultoval návrhy mého řešení.

3.3.1 Vyjádření časové náročnosti

Tomuto úkolu a jednotlivým problémům, které vznikly v jeho průběhu, jsem se věnoval po zbytek pracovních dnů bakalářské praxe.

4 Zvolený postup řešení zadaných úkolů

4.1 Seznámení s produktem ExtJS od společnosti Sencha

4.1.1 Představení produktu

ExtJS od firmy Sencha Inc. je JavaScriptový aplikační Framework, který slouží k tvorbě především rozsáhlejších webových aplikací. Jeho verze ExtJS 1.0 vyšla v roce 2007 a v průběhu následujících let byl tento produkt stále zdokonalován. Aktuální verze 5.1.0 vyšla v prosinci 2014, a v druhém čtvrtletí 2015 bychom se měli dočkat verze ExtJS 6.0.

ExtJS přináší vývojáři řadu výhod. Přes možnost využít systém tříd z OOP, balíčků pro data, jednoduché správy stylů, až po MVC či MVVM architekturu. Nabízí možnost jednoduše vytvořit rozložení stránek, panely, rozsáhlé tabulky s možností editace, formuláře, nástrojové lišty nebo různé typy tlačítek a mnoho dalšího. [2]

Aktuální verze ExtJS je kompatibilní s velkou škálou webových prohlížečů. Od jejich specifických verzí podporuje Safari, Firefox, Internet Explorer, Chrome a Operu. Taktéž jsou podporovány dotykové obrazovky na mobilních zařízeních. Safari na iOS, Chrome na Androidu a IE na Windows. Z mobilních zařízení je podpora v aktuální verzi pouze pro tablety. Tedy pro zařízení s dostatečnou velikostí displeje.

ExtJS je také kompatibilní s jakýmkoliv webovým serverem, kde není nutné žádné specifické nastavení pro jeho používání.

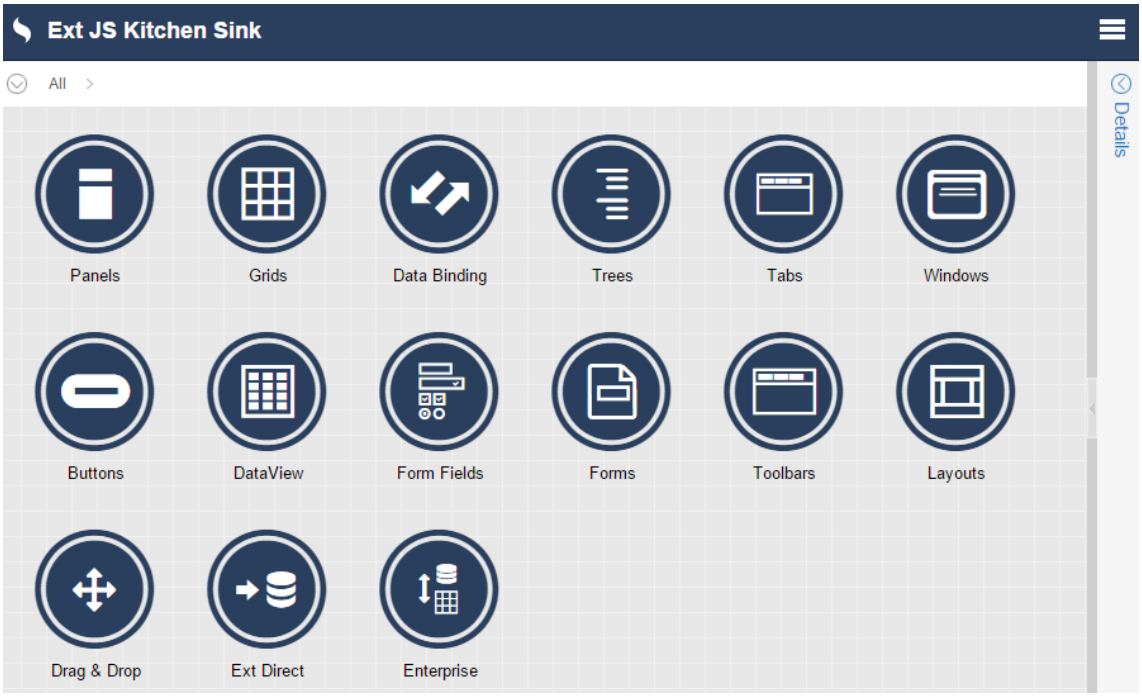
4.1.2 Praktické seznámení s produktem

Mým úkolem bylo seznámit se s komponenty frameworku. ExtJS má poměrně podrobnou uživatelskou dokumentaci a také takzvaný ExtJS Kitchen Sink [3], který obsahuje ukázky jednotlivých nejmenších komponent, které vývojář může použít. Přehled jednotlivých oblastí lze vidět na obrázku 1 . Procházením všech ukázek jsem získal přehled o různých typech prvků a možnostech využití ExtJS. Důležitou částí bylo přečtení řady článků v uživatelské dokumentaci, které popisují, jak jednotlivé části ExtJS fungují a jak se správně mají používat. [4]

Seznámil jsem se také s produktem Sencha Cmd, což je multiplatformní konzolová aplikace, která usnadňuje vývojáři práci s ExtJS a mimo jiné také nabízí možnost automaticky sestavit výsledný JavaScriptový kód. Tento proces není nutný, avšak dokáže snížit velikost zdrojových souborů výsledné aplikace pomocí minifikace a také pomocí selektivního vybrání pouze potřebných části frameworku.

Taktéž jsem si vyzkoušel implementovat některé komponenty v jednoduché webové aplikaci. Pro implementaci jsem si musel stáhnout zdrojové kódy frameworku a nasadit

je spolu s mými vytvořenými kódy na lokální webový server. Vyzkoušel jsem si zobrazení několika tlačítek, panelů a různé způsoby jejich konfigurace.



Obrázek 1: Ukázka ExtJS Kitchen Sink

4.2 Provedení manuálního průzkumného testování

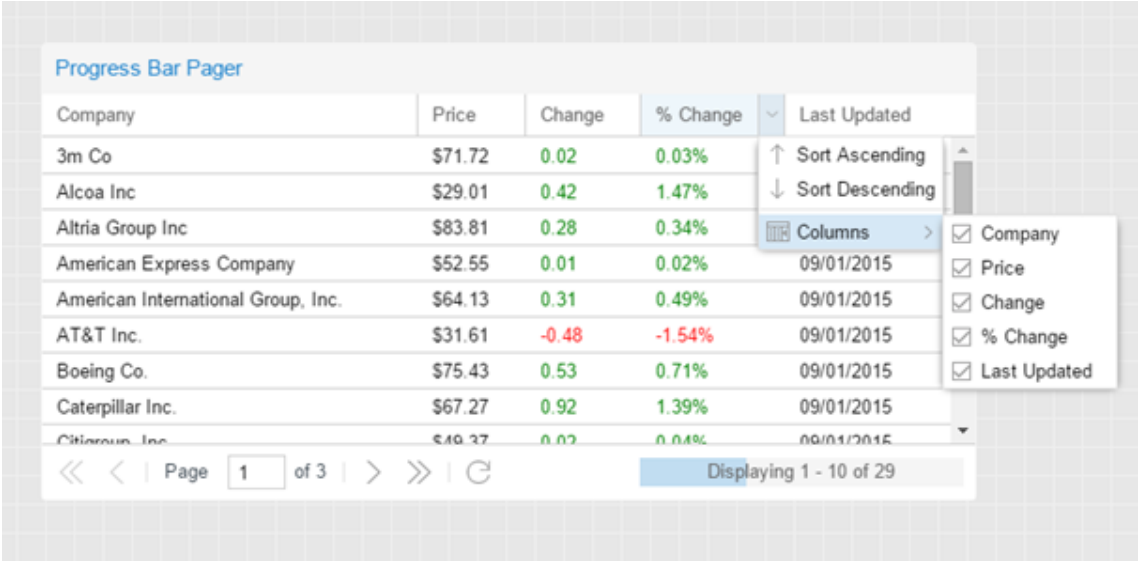
Jednotlivé ukázky komponent v ExtJS Kitchen Sink se pro testovací účely sestavují každý den s aktuální verzí kódu celého frameworku a tak jsou dobrým místem pro odhalení chyb, které mohou vývojáři do kódu zanést. Od společnosti Sencha jsme dostali dokument, který popisuje testy jednotlivých komponent, to znamená testovací případy a kroky s očekávanými výsledky. Zprvu jsem tedy zahájil manuální testování, abych poznal, jak přesně se prvky v jednotlivých krocích chovají. [6]

Zjistil jsem, že ne veškeré popisy testů jsou aktuální. Občas se jednotlivé ukázky komponent v průběhu vývoje frameworku upravovaly, avšak už ne specifikace testů. Například některé ukázky měly přidány nové prvky, nebo naopak některé části chyběly. Jednotlivé nesrovnalosti jsem podrobně zaznamenával. A v případě potřeby upravoval specifikace testů. Jelikož se framework neustále vyvíjí, jsou průběžně přidávány nové komponenty a s tím i nové ukázky těchto komponent. Většina takto nově přidaných komponent neměla sepsané žádné testy. Proto bylo mým úkolem jednotlivé testy a jejich testovací případy sepsat. Všechny nově vytvořené specifikace testů jsem následně kon-

zultoval s mým vedoucím a především jsme si ověřovali jejich korektnost u společnosti Sencha. [5]

Jedním z nejpoužívanějších komponent frameworku jsou tabulky, které nabízí možnost jednoduše vytvořit komplexní aktivní webový prvek. A proto byly také zvoleny jako primární cíl pro automatizaci testování. Ukázku takové tabulky lze vidět na obrázku 2. Testování tabulek také obsahuje velký počet kroků, z nichž se některé manuálně kontrolují poměrně těžko. Příkladem může být správné řazení velkého počtu dat.

Výsledky manuálních testů jsem zprvu zaznamenával do Google Spreadsheet dokumentu. Později jsme pro záznam výsledků jednotlivých testů používali webový nástroj TestRail, který je určen pro správu popisů a výsledků jednotlivých testů. Jednotlivé testy jsem také provedl na různých podporovaných prohlížečích. Díky tomu jsem zjistil, že některé chyby můžou být často specifické pro daný prohlížeč, nebo dokonce pouze pro jeho určitou verzi. Všechny nalezené chyby jsem také nahlásil společnosti Sencha pomocí jejich interního systému pro evidenci chyb.



Obrázek 2: Ukázka ExtJS tabulky

4.3 Automatizace

Otestování funkčnosti všech ukázek ExtJS frameworku a zpracování výsledků zabere přibližně 16 člověkohodin v jednom webovém prohlížeči. Avšak jak již bylo zmíněno, chyby se mohou objevit pouze na určitých prohlížečích, či pouze na jejich určitých verzích. Kdy například IE11 a IE10 fungují bez problému, avšak na IE8 nastává problém. Z toho důvodu je potřeba provádět testování na všech podporovaných prohlížečích a jejich verzích. Což velice zvyšuje komplexitu a časovou náročnost testování. Automatizace v tomhle případě dokáže ušetřit velké množství člověkohodin. Dále zde hrají roli také priority společnosti Sencha, kdy například nejdůležitější verzí IE je verze 8, především kvůli korporátním zákazníkům. Tyto kritéria pak určují priority nejen pro provedení testování, ale také pro automatizaci.

4.3.1 Výběr nástroje pro automatizaci testování

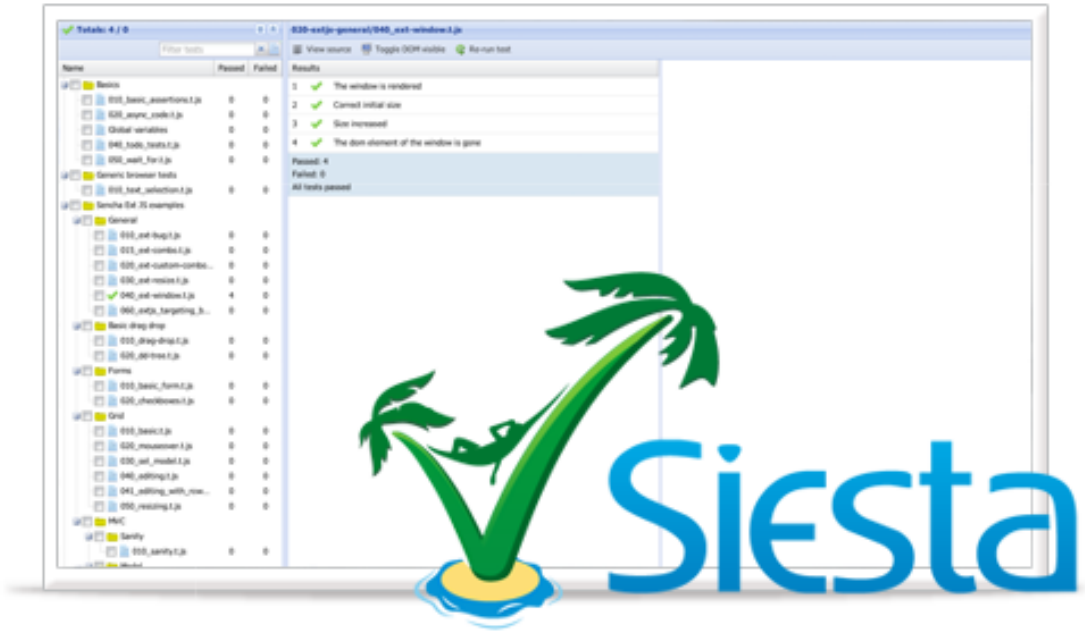
Jelikož se jedná o funkční testování, je zapotřebí vybrat nástroj, s pomocí kterého můžeme ovládat webovou stránku přímo ve zvoleném prohlížeči. Tímto zajistíme, že zobrazení je stejné, jako ho následně vidí koncový uživatel. A protože každý webový prohlížeč může vykreslovat stejnou webovou stránku s malými rozdíly, je potřeba spustit test v různých prohlížecích. Při výběru nástroje pro automatizaci testování jsme se rozhodovali mezi Selenium Webdriver a nástrojem Siesta.

Selenium Webdriver je pravděpodobně jeden z nejznámějších open source nástrojů pro automatizaci webového prohlížeče a testování webových stránek. Selenium Webdriver podporuje velkou řadu programovacích jazyků Javu, C#, Ruby, Python i JavaScript. Má rozsáhlé API a podrobnou dokumentaci. Dokáže ovládat přímo nainstalovaný prohlížeč v daném systému. Avšak kromě Firefoxu musí být pro každý prohlížeč použit specifický driver, který umožňuje jeho propojení s Webdriver API. [7]

Siesta od společnosti Bryntum je nástroj postavený na JavaScriptu který slouží pro testování JavaScriptového kódu. Lze pomocí něho také kompletně otestovat objektový model dokumentu webové stránky a simulovat uživatelskou interakci s webovou stránkou. Siesta má nativní podporu pro ExtJS a její interface je taktéž na tomto frameworku postaven. To, že se jedná o JavaScriptový framework přináší řadu výhod, ale také několik omezení. Jednou z největších výhod je možnost spustit dané testy v prohlížeči bez jakékoliv potřeby něco instalovat nebo nějak zasahovat do nastavení. Následují další dvě nevýhody. Jelikož se při testování používají XHR žádosti, ve většině prohlížečů platí „Same Origin Policy“. Z toho vyplývá, že testy se Siestou musí být na stejné doméně jako testovaná webová stránka. Nejčastěji tedy obě hostované lokálně na PC. Druhá velká nevýhoda vyplývá přímo z použití samotného JavaScriptu. Ten běží v prohlížeči v Sandbox režimu a tudíž může vykonávat jen operace spojené s danou webovou stránkou. A proto například většinou nelze přistupovat k lokálním souborům. [8]

Po zvážení všech pro a proti jsme se rozhodli pro nástroj Siesta a to z několika důvodů.

- Vznikl přímo pro testování ExtJS a obsahuje jeho nativní podporu
- Testy jsou snadno spustitelné ve všech prohlížečích, které ExtJS podporuje
- Rozsáhlá dokumentace a ukázky testování nad ExtJS
- Jedná se o komerčně vyvíjený produkt s kvalitní technickou podporou
- Společnost Bryntum aktuálně navázala úzkou spolupráci se společností Sencha



Obrázek 3: Logo a rozhraní produktu Siesta

4.4 Vývoj automatizovaných testů v Siestě

Následující informace jsem získal především studiem Siesta API dokumentace [9]. Siesta se pro uživatele skládá ze dvou hlavních částí. Harness souboru spolu se zdrojovými soubory Siesty a ze souborů, které představují jednotlivé testy. V Sieste jeden soubor představuje jeden test, který ovšem může obsahovat kódu a testových případů, kolik je zapotřebí. Každý test je spuštěn samostatně a nezávisle na ostatních. Harness soubor je místem, kde nastavujeme veškeré globální nastavení. Také zde definujeme především jednotlivé testy, tedy soubory, které testy obsahují. K nim přiřadíme URL adresy s webovými stránkami, nad kterýma budeme chtít daný test spustit.

4.4.1 Nastavení Harness souboru

Testy se dají jednoduše spustit s defaultním nastavením Harness souboru. Avšak postupem času jsem přidával a měnil různá nastavení tak, aby co nejvíce odpovídali našim požadavkům. Například vzhledem k potřebě spouštět testy v málo výkonném IE 8, bylo zapotřebí omezit počet paralelně běžících testů na 1 a zvednout defaultní hodnotu maximálního čekaní na akce. Taktéž bylo potřeba vypnout vyrovnávací paměť, protože při opakovaném spuštění jednoho testu mohlo dojít, že testovaná stránka nebyla v očekávaném původním nastavení. Nastavení velikosti zobrazované plochy pro webovou stránku je také užitečné, protože některé ukázkové komponenty ExtJS frameworku, které bylo potřeba otestovat, jsou velice malé a naopak jiné mohou být velké. Harness soubor je JavaScriptový soubor. Díky tomuto jsem v něm mohl používat proměnné a například dynamicky definovat aktuální cestu k testované aplikaci. ExtJS také podporuje několik různých typů témat, jak mohou jednotlivé komponenty graficky vypadat. Možnost spuštění testů nad různými tématy jsem také implementoval do nastavení Harness souboru.

```
var Harness = Siesta.Harness.Browser.ExtJS;
var build_name = 'app-369';
var theme = 'theme1';

Harness.configure({
    title : 'Regression tests',
    autoScrollElementsIntoView: true,
    disableCaching: true,
    viewportHeight: 2000,
    viewportWidth: 2000,
    defaultTimeout: 30000,
    waitForTimeout : 5000,
    maxThreads: 1,
    testClass: Test.SharedLibrary
});

Harness.start({
    group: 'Group of tests',
    viewportHeight: 1500,
    viewportWidth: 1500,
    items: [{
        hostPageUrl: '/' + build_name + '/build/application/?theme=' + theme + '#grid1',
        url : '/test/arGrid.t.js'
    }, {
        hostPageUrl: '/' + build_name + '/build/application/?theme=' + theme + '#grid2',
        url : '/test/grGrid.t.js'
    }, {
        hostPageUrl: '/' + build_name + '/build/application/?theme=' + theme + '#grid3',
        url : '/test/lGrid.t.js'
    },
```

Výpis 1: Ukázka nastavení Harness souboru

4.4.2 Synchronizace v JS

Vývoj testů v Sieste může být pro někoho, kdo není zvyklý na asynchronní chování JavaScriptu, ze začátku trochu problémový. Když simulujeme uživatelskou interakci, je zapotřebí aby se veškerá interakce s webovou aplikací prováděla postupně. Při psaní testu jsem proto používal 3 způsoby synchronizace.

Prvním způsobem synchronizace jsou Siesta metody t.it nebo t.describe, tyto jednotlivé metody mohou obsahovat další kód a jsou synchronizované navzájem mezi sebou. Tedy pokud máme dvakrát za sebou t.it(), druhá metoda se spustí, až když se provedou veškeré operace v té první.

Dalším způsobem je použití Siesta metody t.chain. V této metodě můžeme použít některé předdefinované operace jako například klikání, pohyby myší nebo psaní na klávesnici. Všechny tyto operace jsou synchronizované a vykonávají se postupně. Pokud potřebujeme v t.chain použít i jiné něž předdefinované metody, lze vyvolat generickou funkci a v ní napsat potřebný kód.

Třetím způsobem je možnost použít JavaScriptový callback, kdy následující část kódu se vykoná, až skončí exekuce dané funkce. Naprostá většina metod ze Siesty obsahuje callbacky pro snazší synchronizaci. A samozřejmě také programátor může použít callback ve vlastní funkci.

Během psaní testů bylo zapotřebí kombinovat všechny tři typy synchronizace. Metody t.describe jsem používal především pro oddělení větších částí testů. Tedy nejčastěji pro oddělení částí kódů jednotlivých testovacích případů. Toto řešení je vhodné zvláště kvůli strukturovanosti výsledků testů, kdy výsledky jednotlivých kroků mohou být zabaleny do jednoho většího bodu. V jednotlivých metodách t.describe jsem následně využíval metody t.chain pro synchronizaci dílčích úkonů a nebo metody t.it. Metoda t.it se chová stejně jako metoda t.describe. Rozdíl je pouze v zobrazení výsledků testů. Metoda t.chain by se dala nahradit pomocí t.it nebo t.describe, avšak nebylo by to tak efektivní z pohledu psaní kódu testu a také z pohledu strukturovanosti výsledků testu. V následující ukázce lze vidět vykonání dvou operací kliknutí myši i použití generické funkce, která volá programátorem definovanou funkci a předává jí callback.

```
t.describe('Move correctly',function(t){
        t.chain(
        { click : '>>button[text=Enter]'},
        function(callback) {
            confirmTheData(callback);
        },
        { click : '>>button[text=Exit]'}
    )
});

t.describe('Hide Columns test',function(t){
```

```
    t.hideColumnsTest('grid');
});
```

Výpis 2: Příklad použití metod pro synchronizaci

4.4.3 Kontrola operací automatizace

Při psaní automatického testu je zapotřebí nejen zautomatizovat ovládání prvků na webové stránce, ale zároveň ověřit, že prvky se po uživatelské interakci chovají tak, jak je předpokládáno. Při psaní testů daných komponentů jsem většinou používal dvojí typ kontroly.

Prvním možností je zkontrolovat přímo ExtJS objekty. Tedy přistoupit přímo k instanci daného komponentu a zjistit si jeho nastavení. Například pokud je zapotřebí splnit předpoklad, že dané tlačítko je schované, můžeme v objektu tlačítka přistoupit k jeho atributu hidden a zjistit, jestli tomu opravdu tak je. Avšak to, že objekt má určité nastavení, ještě nemusí znamenat, že se skutečně projevilo v HTML kódu. Je proto třeba ověřit i přímo HTML. Nejčastěji tedy CSS elementy. Nebo například při kontrole akcí nad daty u tabulek. Na jednu stranu můžeme zkontrolovat data uložená v objektech, ale zároveň je potřeba zkontrolovat jestli jsou stejná data zobrazena v HTML kódu. Dvojí způsob ověření můžeme vidět ve výpisu zdrojového kódu číslo 3.

Při plnění jednotlivých předpokladů je zapotřebí také zajistit, že se skutečně provedla změna a prvek, který kontrolujeme, nebyl náhodou již ve stavu, který očekáváme. Například když kontrolujeme, jestli po umístění myši nad tlačítko je změněn jeho vzhled, je prvně zapotřebí ověřit, že tlačítko nemá požadovaný efekt. Následně umístit nad něj myš a zkontrolovat, že požadovaný efekt nastal. Po odsunutí myši je nutné opět zkontrolovat, že se tlačítko vrátilo do původního stavu. Ověření před i po akci můžeme vidět ve výpisu zdrojového kódu číslo 3.

Siesta nabízí několik desítek metod pro kontrolu předpokladů, ale osobně jsem nejčastěji používal metodu t.is, která přijímá dva atributy a ty následně mezi sebou porovná. Pro ověření nějakého stavu můžeme použít i metody typu t.waitFor, kdy čekáme na vykonání nějaké operace. Například na skrytí okna.

```
t.describe('Move correctly',function(t){
    var btn = Ext.ComponentQuery.query('button[text=Hide]')
        t.chain(
            function(next){
            t.is(btn.isHidden(),false,'Button should be visible'); //object check
            t.is(t.isElementVisible(btn),true,'Button should be visible'); // HTML check
            },
            { click : btn },
            function(next){
            t.is(btn.isHidden(),true,'Button should be hidden'); // object check
            t.is(t.isElementVisible(btn),false,'Button should be hidden'); // HTML check
            })
});
```

Výpis 3: Ukázka kontroly operací automatizace

4.4.4 Čekání na vykonání operací

Webová stránka může při interakci vyvolat různé reakce, například animované otevření menu, zobrazení dialogového okna apod. Automatické testy musí na takovéto případy myslet a čekat, až se například animace dokončí. Při čekání nemůžeme nastavovat počet sekund napevno. Testy v různých prohlížečích a na různě výkonných systémech běží pokaždé jinou rychlostí. Siesta nabízí opět několik desítek funkcí, které vykonávají čekání na určité operace. Já jsem nejčastěji používal t.waitForComponentQuery, t.waitForCssSelector a t.waitForAnimations. Většina ExtJS komponent má také velkou řadu JavaScriptových událostí. Díky tomu jsem například mohl korektně čekat, než příslušný ExtJS objekt seřadí a zobrazí seřazené data.

```
t.it('Message box should be shown',function(t){
    t.chain(
        { waitFor: 'Animations' },
        { click : '>>button[text=Confirm]' },
        { waitFor: 'ComponentQuery',
            args: 'messagebox[title=New Window]',
            desc: 'MessageBox has been shown'
        }
    )
}
```

Výpis 4: Ukázka kódu při čekání na vykonání operací

4.4.5 Výběr konkrétních elementů v HTML kódu webové stránky

Často jsem potřeboval v testu vybrat konkrétní HTML elementy. Jako vhodný kandidát pro rozlišení jednotlivých elementů se může jevit jejich id atribut. Problém však je, že ExtJS vytváří id elementů dynamicky. Takže při spuštění aplikace např. v jiném prohlížeči mohou být id naprosto jiné. Jednou z možností, jak použít přímo id atributy, je získat je z ExtJS objektu. Tím pádem jsou vždy aktuální pro danou situaci. Tuto možnost jsem

využíval, především když jsem potřeboval zacílit na specifickou část komponenty a jako cíl testovací metody jsem nemohl předat celý ExtJS objekt. Toto použití můžeme vidět v ukázce kódu číslo 5. V úvodu si pomocí ComponentQuery načtu ExtJS objekt a následně z něj získám jeho aktuální id, který použiji k zacílení na konkrétní HTML element, který je zapotřebí zkontrolovat.

Další způsob, který jsem použil pro výběr konkrétních HTML elementů, jsou CSS selektory. Avšak tuto metodu jsem se snažil používat co nejméně, jelikož může být více náchylná na změny HTML kódu testované aplikace.

```
t.it('Button test',function(t){
    btn = Ext.ComponentQuery.query('grid button')[0]
    t.hasNotCls('a[id='+btn.id+']','x-btn-pressed')
}
```

Výpis 5: Ukázka kódu při použití id atributu pro výběr HTML elementu

4.4.6 Sdílení funkcí mezi testy

Při testování ukázek jednotlivých tabulek jsem objevil testové případy, které byly ve více testech stejné. Například přesouvání, změna velikost sloupců anebo seřazení dat v tabulce. Pro tyto účely jsem vytvořil generické funkce, které se daly použít ve více testech a šly nastavit pro konkrétní případ pomocí argumentů. Dále jsem si vytvořil několik dalších pomocných funkcí na zpracovávání specifických ExtJS objektů. Tyto vytvořené funkce jsem potřeboval sdílet mezi jednotlivými testy a jejich soubory.

Siesta pro tyto účely nabízí možnost rozšířit třídu Siesta.Test. Tato třída je přístupná ve všech testových souborech. Sdílené funkce jsem proto umístil do jednoho souboru a cestu k němu následně nastavil v Harness souboru. Nevýhodou je, že tento soubor může být pouze jeden a tak při rozsáhlejším projektu se může rychle rozrůst jeho počet řádků.

Sdílení funkcí mezi testy přináší řadu výhod. Především zrychluje vývoj testů, kdy můžeme znovu použít již jednou napsaný kód. Také údržba takto vytvořeného kódu je mnohem jednodušší, jelikož stačí kód upravit na jednom místě. Takto definované funkce také přispívají k zpřehlednění jednotlivých testů, jelikož testy následně obsahují pouze volání funkcí místo jejich celé implementace. Nesmím opomenout, že ty funkce jsou také snáze sdíleny v rámci týmu vývojářů.

4.4.7 Optimalizace pro všechny webové prohlížeče

Při exekuci testů na všech podporovaných prohlížečích jsem se setkal s několika problémy. Drtivá většina problémů se týkala prohlížeče IE a především jeho verze IE 8. Exekuce JavaScriptového kódu může být náročná a značně závisí na implementaci v daném prohlížeči. Jelikož IE 8 je poměrně starý webový prohlížeč, operace, které na Chrome zabraly 5 sekund, mohly někdy na IE 8 zabrat i více jak půl minuty. Z toho důvodu bylo

nutné upravovat defaultní časové intervaly, po které jednotlivé testovací metody čekají na vykonání operací. Problém také nastal, když jsem použil JavaScriptovou metodu indexOf(), kterou IE 8 neumí. A proto jsem si musel napsat její vlastní implementaci. Při vývoji testů jsem tedy co nejčastěji zkoušel jejich exekuci i na IE8. Takto jsem mohl pružně reagovat na problémy a efektivně měnit čerstvě napsaný kód.

4.4.8 Ukázka zdrojového kódu testu

V následující výpisu zdrojového kódu číslo 6 lze vidět část testu. V úvodu jsou ukázány dvě mé vlastní funkce. Ta první pomocí Siesta metody kontroluje událost, kterou vyvolá ExtJS objekt. V tomto konkrétním případě se jedná o objekt posuvníku a událost change, která se vyvolá po jeho posunutí. Tato událost zároveň obsahuje konkrétní informace o stavu posuvníku a tyto informace jsou porovnávány s očekávaným výsledkem. Druhá funkce obsahuje část kódu, která by se v testu často opakovala, a proto jsem tento kód umístil do vlastní funkce, jejíž volání je upraveno parametrem. Tato funkce také obsahuje callback, díky němuž bude exekuce pokračovat až po skončení této funkce.

V další části začíná kód samotného testu. Na začátku bylo zapotřebí počkat, až bude testovaná tabulka plně načtena a poté si uložit do proměnné objekt dané tabulky . Tato část kódu je v metodě t.it, které ja synchronizovaná s ostatními metodami t.describe. Následuje blok s prvním testem na kontrolu, jestli se dají skrýt jednotlivé sloupce tabulky. Implementace kroků testu je umístěna ve sdílené funkci t.hideColumnsTest, která je posléze pouze volána s potřebnými parametry. Další část kódu kontroluje použití tlačítek u posuvníku. Na začátku jsem si musel opět uložit objekty tlačítek a daného posuvníku. Můžeme zde vidět použití další synchronizační metody, a to t.chain, v které jsou umístěny jednotlivé kroky. V tomto případě kliknutí na příslušná tlačítka a kontrola, jestli nastaly očekávané změny pomocí na za začátku zmíněných funkcí. Test by po této části mohl pokračovat dále a provádět další testovací případy.

```
function eventCheck(t,sslider,numb){
    t.isFiredWithSignature(sslider,'change', function(slider,newValue,thumb,eOpts){
      return (newValue==numb)
    },'Checking event of slider, value should be correctly changed.')
}

function check(t,txt,numb,callback){
    t.chain(
        function(con){
            t.waitForCompositeQuery('pagingtoolbar => div:contains('+txt+')','',con)
        },
        {waitFor: 'Animations'},
        function(){
            callback();
        }
    )
}
StartTest(function (t) {
    var grid;

    t.it('Waiting for grid load',function(t){
        t.waitForRowsVisible('grid',function(t){
            grid = Ext.ComponentQuery.query('grid')[0];
        })
    })

    t.describe('Hide Columns test',function(t){
        t.hideColumnsTest(t,'grid',4);
    })

    t.describe('Use navigation arrows',function(t){
        var btns = Ext.ComponentQuery.query('pagingtoolbar button');
        var slider = Ext.ComponentQuery.query('pagingtoolbar slider')[0];

        t.it('Double page buttons',function(t){
            t.chain(
                function(con){
                    eventCheck(t,slider,1);
                    con();
                },
                {click: btns[3]},
                function(next){
                    check(t,tx3,3,next);
                },
                function(con){
                    eventCheck(t,slider,1);
                    con();
                },
                {click: btns[0]},
                function(next){
                    check(t,tx1,1,next);
                }
            )
        })
    })
```

Výpis 6: Ukázka části zdrojového kódu testu

4.5 Spouštění testů

Ke spuštění automatických testů v Siestě stačí spustit webový server nad Siestou a testovanou aplikací. A následně přistoupit na příslušnou URL přes webový prohlížeč. V Siestě lze jednoduše vidět seznam nadefinovaných testů, aktuální průběh testu, a pokud uživatel chce, může si zobrazit i vizuální exekuci daného testu.

Pokud jsem spouštěl příslušné testy, bylo zapotřebí, abych je postupně spustil na všech podporovaných prohlížečích. Webové prohlížeče Chrome, Firefox nebo Opera mohou být na systému nainstalovány současně. Avšak problém nastává, když potřebujeme testy spustit na různých verzích prohlížeče Internet Explorer. IE může být v systému nainstalován pouze v jedné verzi, ať už IE 8 nebo IE 11. A proto jsem se rozhodl pro instalaci několika virtuálních strojů s Windows 7, každý s jinou verzí IE. Windows 7 jsem zvolil proto, že lze na něm provozovat všechny 4 podporované typy IE. Abych nemusel mít na každém stroji stejné zdrojové soubory testů, přidal jsem do virtuálních systémů sdílenou složku s hostitelským systémem. V této složce jsem následně měl vždy aktuální verzi testů a aktuální verzi ExtJS komponent k testování. Následně stačilo v každém virtuálním systému spustit webový server a přistoupit přes webový server k Sieste.

Nyní ovšem vím, že nebylo třeba spouštět webový server v každém z virtuálních strojů. Webový server se Siestou a testovanou aplikací stačilo spustit na hostitelském systému. Propojit virtuální stroje s hostitelským pomocí síťového rozhraní a přistoupit z webového prohlížeče ve virtuálním systému k danému webovému serveru běžícímu na hostitelském systému.

4.6 Kde jsem nemohl použít Siestu

Po dokončení automatických testů pro několik příkladů tabulek jsem dostal za úkol, vyvinout automatické testy pro ExtJS grafy, jelikož tato komponenta je také velmi často používána. Avšak zde jsem narazil na problém. Nové grafy v ExtJS jsou na stránce vykreslovány pomocí HTML canvas elementu. Nicméně tento element v HTML kódu webové stránky neprozrazuje žádné informace o tom, co je na něm zobrazeno.

Pravděpodobně nejschůdnějším řešením, jak tento element otestovat, je pórovnávání snímků obrazovky. Tedy proces, kdy uděláme snímek canvas elementu a porovnáváme ho s očekávaným výsledkem. Avšak bohužel Siesta v době řešení tohoto problému nepodporovala vytváření snímků obrazovky a tak jsem musel sáhnout po jiném řešení.

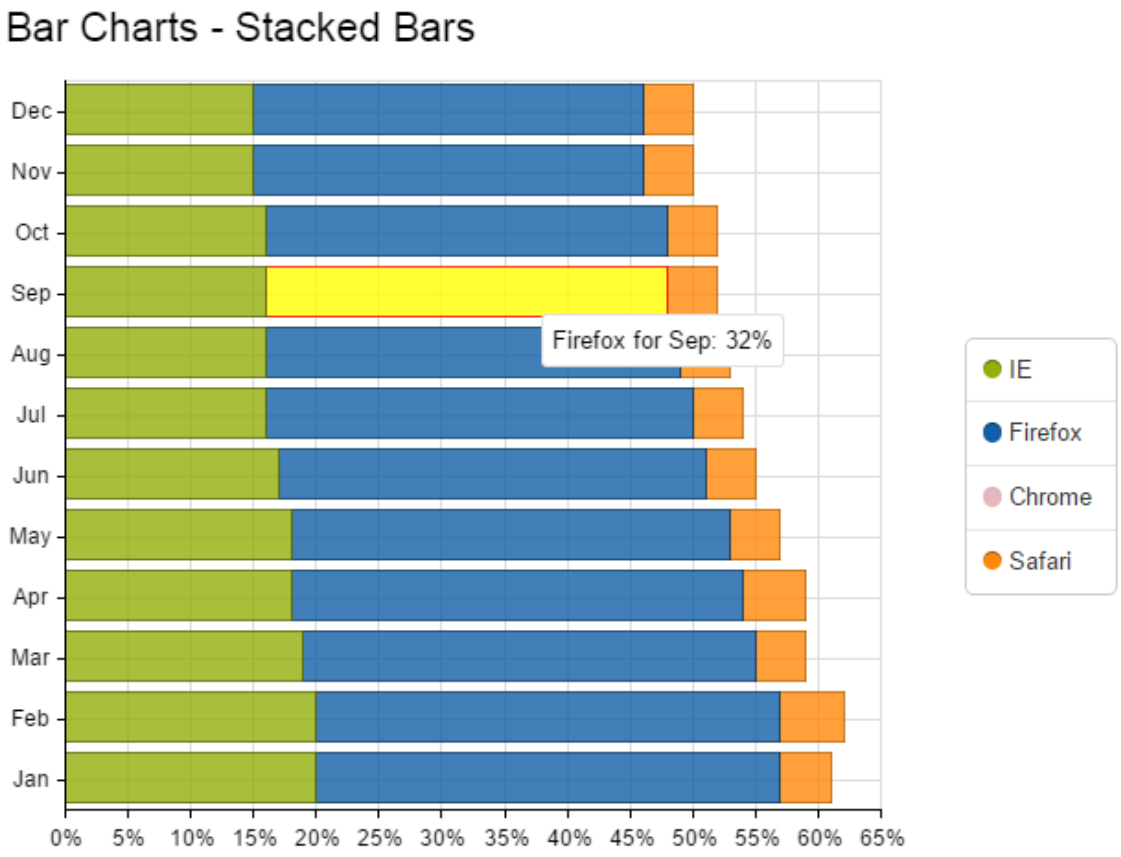
4.6.1 Použití Selenium Webdriver pro ExtJS grafy

Na jiném projektu v práci kolegové řešili podobný problém testování pomocí snímků obrazovky. Dostal jsem tedy k dispozici Python knihovnu na porovnávání obrázků a generování výsledků testů pomocí Selenium Webdriver. Stačilo mi tedy přepsat některé části knihovny pro potřeby testování ExtJS a napsat v Pythonu automatické testy pro

jednotlivé grafy. Testovalo se zejména schovávání jednotlivých částí grafu a změna grafického zobrazení společně s nápovědou po najetí myši na určitou část grafu.

Automatické testy probíhaly tak, že po prvním spuštění testu se vygenerovaly první snímky obrazovky, tedy přesněji pouze snímky daného canvas objektu. Snímky se generovaly po každé operaci. Tedy po kliknutí na tlačítko nebo po najetí myši na část grafu. Tyto první vygenerované snímky se musely ručně zkontrolovat a následně sloužily jako očekávaný výsledek. Při následovném spuštění testů se nové snímky porovnávaly s již vytvořenými a případné nesrovnalosti se zobrazovaly ve výsledcích testů.

Generováním snímků pouze daného elementu jsme eliminovali nežádoucí vlivy, které ovlivňují používání tohoto typu testů. Například změny UI stránky, nebo jinak vypadající prohlížeč. Avšak bohužel toto opatření neeliminovalo rozdíly mezi jednotlivými prohlížeči. Jelikož každý prohlížeč vykresluje HTML canvas s mírnými rozdíly, bylo nutné vytvořit sadu očekávaných výsledků pro každý testovaný prohlížeč zvlášť.



Obrázek 4: Ukázka ExtJS grafu

5 Teoretické a praktické znalosti a dovednosti získané v průběhu studia a uplatněné v průběhu praxe

Během vykonávání praxe jsem využil znalosti ze všech programovacích předmětů. Tedy především koncepci a styl programování. Jelikož jsem před absolvováním praxe JavaScript nepoužíval, tak byl pro mě velkým přínosem předmět Skriptovací programovací jazyky a jejich aplikace, protože jsme v tomto předmětu programovali v Pythonu, který jsem v průběhu praxe využil. A také protože jsme v tomto předmětu používali Linux. Malé základy Linuxu jsem taktéž získal v předmětu Operační systémy. Linux byl totiž hlavní platformou, na které jsme v průběhu praxe pracovali.

Jelikož se v ExtJS aplikacích používají návrhové vzory, díky předmětu Vývoj informačních předmětů jsem dokázal být rychle v obraze. Největší přínosem by pro mě byly předměty Vývoj internetových aplikací a předmět Správa operačních systémů. Avšak tyto předměty jsem absolvoval průběžně s vykonáváním praxe a většinu informací, které jsem v těchto předmětech získal, jsem v praxi potřeboval o několik týdnů dříve.

Jelikož je veškerá písemná komunikace je ve firmě v angličtině a taktéž meetingy probíhají v tomto jazyku, hodiny angličtiny, ať už v povinném předmětu Jazyk anglický nebo nepovinném předmětu Anglická konverzace, byly pro mě určitě velkým přínosem. Dále jsem uplatnil spoustu znalostí, získaných samostudiem, navazujícím na většinu zmíněných předmětů.

6 Znalosti či dovednosti scházející studentovi v průběhu odborné praxe

Testování softwaru by mělo být přítomné u každého většího softwarového produktu. Avšak při mém studiu na VŠB jsem se doposud s touto oblastí nesetkal v žádném předmětu. Tedy spíše nezaznamenal jsem žádnou zmínku o potřebě otestovat výsledný softwarový produkt. Nicméně terminologie používaná v této oblasti není nijak komplikovaná a tak mě má neznalost nijak nezpomalovala.

Pokročilejší znalost JavaScriptu je oblastí, ve které nebyly mé znalosti příliš silné. Tím pádem mohl být můj počáteční výkon pomalejší, než kdybych již měl v tomto jazyku zkušenosti. V předmětu Vývoj internetových aplikací jsme se JavaScriptu věnovali, avšak tyto znalosti jsem potřeboval již dříve, než jsme je v daném předmětu probírali. A také probíraná úroveň JavaScriptu byla spíše na základní úrovni.

Dále mi chyběly praktické znalosti ohledně vývoje softwaru. Například praktické používání systémů VCS typu Git anebo nástroje pro průběžnou integraci. Práci s těmito nástroji jsem se musel v průběhu mé praxe naučit.

7 Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

7.0.2 Výsledky mé práce

Výsledkem mé práce v rámci rozsahu bakalářské praxe byly automatizované testy pro všechny aktuálně dostupné ukázky tabulek v ExtJS frameworku, celkem okolo 21 testů. Také jsem nastavil testovací framework Siesta a zavedl sdílenou knihovnu s částmi kódů. Na základě této mé práce se mohlo dále pokračovat ve vývoji automatických testů pro zbylé ukázky komponentů ExtJS.

Taktéž jsem vyvinul za pomocí nástroje Selenium Webdriver automatické testy v jazyku Python. Tyto testy dokázaly zkontrolovat všechny aktuálně dostupné ukázky grafů v ExtJS. Průběžná exekuce mých testů následně vedla k odhalení několika chyb frameworku. Tyto chyby jsem nahlásil a popřípadě přímo konzultoval s vývojovým týmem. Díky tomu jsem přispíval k celkovému zlepšování kvality frameworku. Kromě toho jsem nalezl několik chyb v samotném testovacím nástroji Siesta, tyto chyby jsem nahlásil a v aktuální verzi Siesty jsou již opraveny.

Automatické testy, které jsem vyvinul, ze začátku sloužily jako první prototypy, jak automatizovat testování frameworku ExtJS. Po zhodnocení těchto prototypů se společnost Sencha rozhodla rozšířit tým ExtJS v naší firmě a použít postupy z prototypů pro automatizaci všech manuálních testů.

7.0.3 Zhodnocení odborné praxe

Během mé praxe jsem prohloubil mé znalosti programování v jazycích Javascript a Python. Taktéž jsem se naučil pracovat v operačním systému Linux a naučil se velmi podrobně pracovat s nástrojem Siesta. Zjistil jsem, jak probíhá vývoj rozsáhlých softwarových produktů v praxi, a které nástroje se k tomu používají. Dále jsem si zlepšil mé znalosti angličtiny, díky písemné i ústní komunikaci.

Po odpracování dnů v rámci bakalářské praxe jsem ve firmě již zůstal a zařadil se mezi běžné zaměstnance. Pokračoval jsem na rozvoji výše uvedených projektů, ale také na mnohých dalších.

8 Reference

[1] profiq - About us. *profiq* [online]. (c) 2010 [cit. 2015-04-26]. Dostupné z: http://testing.profiq.cz/about/

[2] Sencha ExtJS. *Sencha* [online]. (c) 2015 [cit. 2015-04-26]. Dostupné z: http://www.sencha.com/products/extjs/

[3] ExtJS 5.0 Examples. *Sencha ExtJS* [online]. (c) 2006-2015 [cit. 2015-04-26]. Dostupné z: http://dev.sencha.com/ext/5.1.0/examples/index.html

[4] ExtJS Guides. *Sencha Guides* [online]. 7.4.2015 [cit. 2015-04-26]. Dostupné z: http://docs.sencha.com/extjs/5.1/

[5] Certifikovaný tester: Učební osnovy pro základní stupeň. *International Software Testing Qualifications Board* [online]. 15.6.2013 [cit. 2015-04-26]. Dostupné z: http://castb.org/wp-content/uploads/2013/11/ISTQB-CTFL-Syllabus-v2011-CZ-Beta1.pdf

[6] KANER, Cem, James BACH a Bret PETTICHORD. *Lessons learned in software testing: a context-driven approach.* New York: Wiley, c2002, xxvii, 286 p. ISBN 0471081124.

[7] Selenium Documentation *SeleniumHQ* [online]. 27.4.2015 [cit. 2015-04-28]. Dostupné z: http://www.seleniumhq.org/docs/

[8] Siesta. *Bryntum* [online]. (c) 2015 [cit. 2015-04-26]. Dostupné z: http://www.bryntum.com/products/siesta/

[9] Siesta API documentation *Bryntum* [online]. 16.4.2015 [cit. 2015-04-26]. Dostupné z: http://www.bryntum.com/docs/siesta/