VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

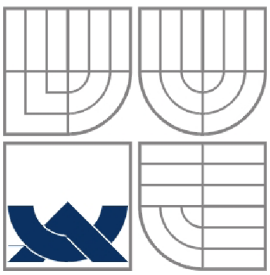
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZOVANÉ TESTOVÁNÍ WEBOVÝCH APLIKACÍ

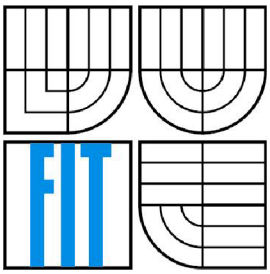
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Tomáš Běloch
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZOVANÉ TESTOVÁNÍ WEBOVÝCH APLIKACÍ

AUTOMATED TESTING OF WEB APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

Tomáš Běloch

VEDOUCÍ PRÁCE
SUPERVISOR

RNDr. Marek Rychlý, Ph.D.

BRNO 2013

Abstrakt

Tato práce se zabývá automatizací v procesu testování webových aplikací. Teoretická část je zaměřena na zajišťování kvality softwarových produktů a služeb. V práci je dále rozebrána problematika automatizace dílčích druhů testování a jsou popsány některé testovací nástroje. Podrobněji se pak teoretická část zabývá nástrojem Selenium a systémy pro kontinuální integraci. Praktická část se zaměřuje na podrobnou analýzu, návrh a realizaci automatizovaných testovacích skriptů pro službu Claudu společnosti IBA CZ, s. r. o. V závěru práce je shrnut její přínos pro autora a společnost a nastíněn další plánovaný vývoj.

Abstract

This bachelor's thesis deals with the automation in the process of testing web applications. The theoretical part is focused on providing the quality of software products and services. The thesis also discusses the problematic of automation of the partial types of testing; some of the testing tools are described here as well. The theoretical part gives more details on the tool called Selenim and continuous integration systems. The practical part is focused on the detailed analysis, proposal and realization of automated test scripts for the Cloud service of the IBA CZ Corporation. The conclusion of this thesis summarizes its contribution to the author and company and outlines the further planned development.

Klíčová slova

automatizované testování, QA, testování, blackbox, SCRUM, Selenium, Java, Webdriver, CI, Jenkins, Hudson, Liferay, Claudu, IBA CZ, webové aplikace

Keywords

automated testing, QA, testing, blackbox, SCRUM, Selenium, Java, Webdriver, CI, Jenkins, Hudson, Liferay, Claudu, IBA CZ, web applications

Citace

Běloch Tomáš: Automatizované testování webových aplikací. Brno, 2013, bakalářská práce, FIT VUT v Brně.

Automatizované testování webových aplikací

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením RNDr. Marka Rychlého, Ph.D.
Další informace mi poskytl Mgr. Marek Vetchý.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Běloch
5.5.2013

Poděkování

Rád bych tímto poděkoval vedoucímu této bakalářské práce RNDr. Markovi Rychlému, Ph.D. za ochotu, trpělivost, připomínky a odborné rady při konzultacích. Toto poděkování ve všech výše uvedených bodech patří také odbornému konzultantovi Mgr. Markovi Vetchému a dalším týmovým kolegům ze společnosti IBA CZ, s. r. o. Dále bych rád poděkoval své rodině a přítelkyni Marcele Šimáčkové za podporu v průběhu celého studia.

Obsah

Úvod

Testování je významnou součástí zajištění kvality softwarových produktů a služeb. Pouze správně nastavený proces testování je zárukou splnění všech kladených požadavků na projekt či službu. Součástí testování však bývá i provádění časově náročné a opakující se činnosti, což vybízí k automatizaci tohoto procesu.

Tato práce čtenáři ve stručnosti nastíní problematiku řízení kvality a testování. Podrobněji se pak zabývá automatizací jednotlivých druhů testování, jako jsou například jednotkové, integrační, funkční nebo systémové testy. Součástí rozboru je diskuse nad klady a zápory automatizace. Jsou uvedeny nejznámější a nejpoužívanější nástroje pro automatizaci, blíže je pak popsán nástroj Selenium využívaný v praktické části práce. V závěru teoretické části je popsán princip fungování systémů pro kontinuální integraci (CI), který je demonstrován na systémech Hudson a Jenkins.

Praktická část je zaměřena na službu společnosti IBA CZ, s. r. o., realizovanou pod názvem Claudu, pro niž byly v rámci této práce vytvářeny automatizované testovací skripty. V úvodu praktické části jsou uvedeny stanovené požadavky. Jelikož se na automatizovaných testovacích skriptech začalo pracovat během vývoje služby, je blíže popsán nastavený proces vývoje a podpůrné nástroje pro testování. Důraz byl kladen také na analýzu testovaných komponent.

V návrhu řešení jsou zohledněny všechny specifikované požadavky. Po dokončení návrhu bylo zprovozněno prostředí pro vývoj a implementovány testovací skripty. Práce se v těchto bodech zabývá nejen implementací, ale i samotným během testů. Na popis implementace navazuje integrace vytvořených testovacích skriptů do systému pro kontinuální integraci – Jenkins.

Souběžně s dokončením implementace přešla služba do pilotního provozu. Byl proto dostatek času nejen na vyhodnocení a ověření správnosti, ale i analýzu dalších možností společně s přechodem na modernější technologie. Toto je uvedeno v závěru práce, kde je mimo jiné diskutována efektivita vytvořených automatizovaných testovacích skriptů, úspora času, zhodnocen aktuální stav a nastínění jejich další plánovaný vývoj.

1 Řízení kvality

Řízení kvality (anglicky Quality assurance) je soubor plánovaných a systematických aktivit, které mají za cíl zajistit splnění požadavků na vyvíjený produkt nebo službu [1]. Kvalita je tedy souhrn vlastností a charakteristik výrobku, procesu nebo služby, které ukazují jeho schopnost splnit určené nebo odvozené potřeby [2].[1] Řízení kvality nezajišťuje kvalitní produkt, ale splnění minimálních nároků na něj kladených. Toto řízení může být realizováno na různých úrovních, jakými je celá organizace, daný projekt, skupina či konkrétní jedinec.

Důvodem zavedení řízení kvality je nejen snížení chybovosti, například neopakováním stále stejných chyb, a zvýšení produktivity podněty pro větší efektivitu. Jedná se také o zkrácení doby uvedení produktu na trh („time-to-market“). Standardizace procesu vývoje, normování produktů a jednotná forma komponent splňujících předpoklady znuvupoužitelnosti, zamezuje plýtvání rozpočtem v případě jejich opětovného použití [3].

Ke kvalitě přispívá velkou měrou nejen důkladná specifikace požadavků, propracovaný návrh, vhodná implementace a správný běh a údržba, ale také testování. Ačkoliv zajišťování kvality zasahuje do mnoha oblastí, bude v této práci dále omezeno na testování, konkrétně pak na jeho automatizaci.

1.1 Plánování kvality

Pro plánování řízení kvality je tedy na základě výše uvedeného nezbytné vydefinovat firemní procesy, které budou určovat, jakým způsobem postupovat. Dále pak testovací plány zahrnující v sobě popis, jak daný produkt vhodným způsobem otestovat, a v neposlední řadě plán revizí udávající formu kontroly [3].

Při tomto plánování musí být dodrženy základní principy správného řízení kvality. Důležitá je pragmatičnost a efektivita, tedy nevytvářet procesy pro procesy. Složité procesy popsané stovkami dokumentů a směrnic jsou nevhodné, je třeba uvažovat běžně používané postupy a ty následně rozumně formalizovat. Vůči utvářenému firemnímu procesu jsou následně vymezovány projekty [3].[2]

Před dalším pokračováním je nezbytné objasnit dva stěžejní významné pojmy řízení kvality – validaci a verifikaci. **Validace** je ověření, že vyvíjený software splňuje potřeby uživatele [5]. **Verifikace** je ověření, že software vyhovuje specifikaci [5].

---

[1] Kvalita je v odborných pramenech definována různě. Například jako míra stupně dokonalosti (Oxfordský slovník), splnění požadavků (Crosby), vhodnost k danému účelu (ISO 9001) nebo schopnost produktu či služby plnit dané potřeby (BS 4778). Vždy má však v jádru to podstatné – splnění očekávání.

[2] K dodržení nastavení správných procesů slouží řada norem. Například již zmíněné ISO 9001 vydávané Mezinárodní organizací pro normalizaci, které definuje systém managementu kvality. Dále například CMM, což je model kvality organizace práce určený nejen pro vývojové týmy. [3]

1.2 Procedurální postupy k eliminaci chyb

Mezi největší hrozby patří nejen zanesení chyby, ale i její neodhalení či neopravení. Zanesení chyb lze eliminovat formalizací požadavků, programováním v páru, průběžnou revizí kódu či dostatečným důrazem na zodpovědnost jedince, který danou problematiku řeší. Pravděpodobnost neodhalení chyby je výrazně snižována zaváděním verifikačních kroků a jejich výstupů, zatímco pravděpodobnost neopravení nalezených problémů snižují například tiketovací systémy [3].[3]

1.3 Aspekty kvality

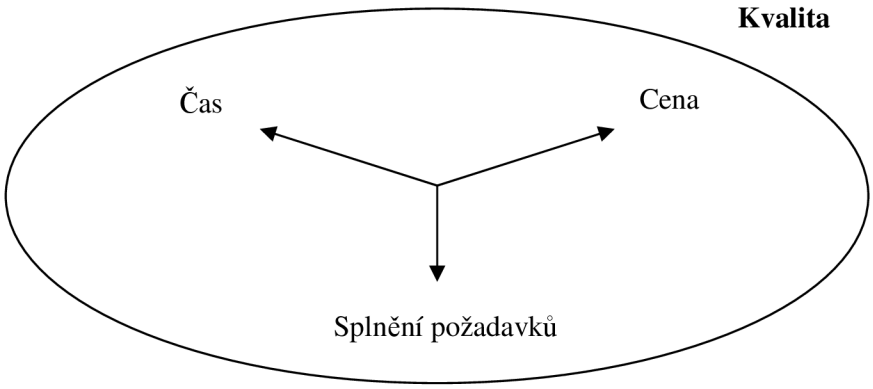
Základní aspekty kvality z hlediska zadavatele či uživatele jsou následující:

Kvantita – produkt nebo služba svým rozsahem nebo množstvím odpovídá požadavkům [1].

Termín – produkt nebo služba je dodána v očekávaném nebo dohodnutém čase [1].

Rozpočet – produkt nebo služba je dodána za očekávanou nebo sjednanou cenu [1].

Všechny výše uvedené aspekty se podílejí na celkové kvalitě produktu, jak je znázorněno na následujícím obrázku 1: Kvalita produktu a její aspekty [2].



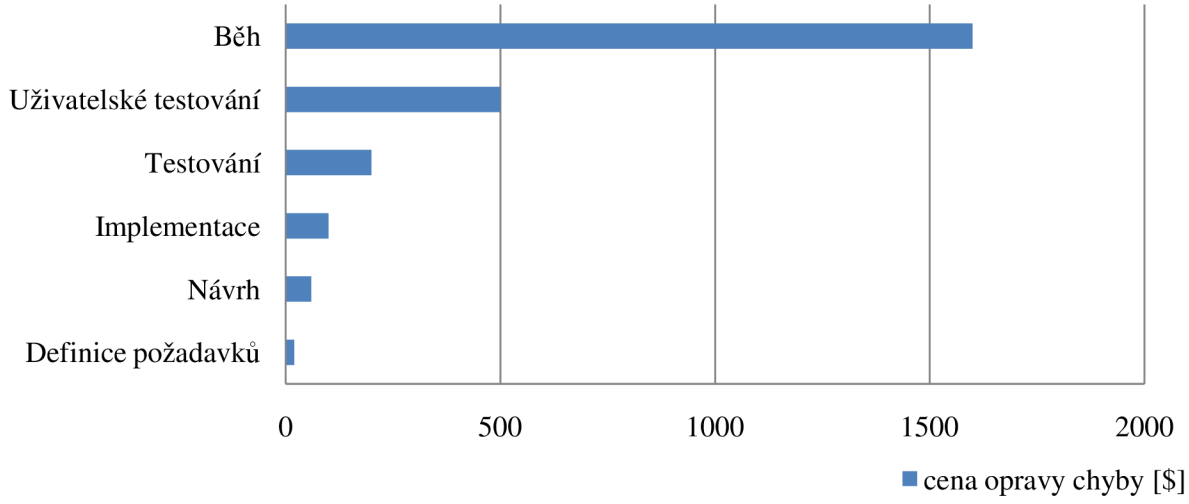
Obrázek 1: Kvalita produktu a její aspekty [2]

[3] Mezi tyto systémy patří například Bugzilla, Request Tracker, MantisBT, Microsoft Dynamics, Rational Clear Quest nebo v této práci využívaný tiketovací systém JIRA.

2 Testování softwaru

Jak bylo uvedeno výše, testování softwaru je nedílnou součástí procesu zajišťování kvality. Samotné testování je založeno na pokládání otázek. Tester, případně QA Specialista[4], se musí sám sebe či jiných členů týmu neustále dotazovat, a jestli odpovídá testovaný produkt specifikaci, jaké obsahuje chyby, a zda splňuje základní předpoklady použitelnosti. Na základě odpovídání si na tyto otázky je poskytována zpětná vazba dalším členům (vývojářům, analytikům, projektovým manažerům atd.). S výše uvedeným bývá spojen i požadavek na do jisté míry kreativní činnost, kdy je nezbytné dostatečně zanalyzovat projekt a rozhodnout, jak jej co nejlépe otestovat, případně v čem může být zdokonalen [3].

Důvodů, proč testovat softwarové produkty, je několik. Tím hlavním a největším je snižování nákladů na pozdější opravy. Na obrázku 2: Cena opravy chyby je znázorněn graf nákladů vynaložených na opravení chyby v běžných vývojových fázích projektu. Jak je vidět, čím později je chyba nalezena a opravována, tím nákladnější její oprava zpravidla bývá.



Obrázek 2: Cena opravy chyby podle studie Barryho W. Boehma, profesora softwarového inženýrství Univerzity Jižní Kalifornie

Testování zároveň poskytuje ucelené informace o kvalitě produktu a udržuje pohodu v týmu. Je však nezbytné mít na paměti, že testování softwaru nezajistí objevení všech chyb, přestože velkého množství ano. Stejně tak neznamená ukončení potřeby psát jednotkové testy (tento pojem bude vysvětlen dále v kapitole Jednotkové testování) a nenapraví nedostatky v analýze.

[4] V praxi se tyto dvě pozice mnohdy prolínají a bývají díky tomu zaměňovány. Pro jednoduchost bude v této práci dále používáno jednotné pojmenování tester.

Následuje výčet základních pojmů používaných v oboru testování, které je nezbytné vysvětlit pro pochopení dalších částí této práce.

2.1 Artefakty testování

Mezi základní artefakty testování patří především testovací plán. Ten zahrnuje posloupnost instrukcí realizovaných v průběhu procesu testování. Dále testovací případ, který je základní částí testovacího plánu. V neposlední řadě pak chyba (přejatě „bug“), u níž je nezbytné zohlednit popis, prioritu, workflow, prostředí výskytu atd. Mezi další patří například scénář pro akceptační testování (zkratka UAT) či automatizovaný testovací skript [3].

2.2 Metody testování

Následuje výčet stěžejních metod v oboru testování včetně podrobnější specifikace.

2.2.1 Blackbox testování

Základní a nejčastěji používanou metodou je tzv. blackbox testování. Daný produkt se testuje bez znalosti vnitřních datových a programových struktur a mnohdy i bez jakékoliv dokumentace. Výhodou tohoto typu testování je jednoduchost, rychlost, transparentnost a nezávislost na změně hardwaru, operačního systému či programovacího jazyka. Mezi nevýhody však patří nižší kvalita kódu, pakliže se testuje pouze blackbox metodou, a nežádoucí chování aplikace pro netestované případy, které nebyly pouhým pohledem zřejmé, případně zahrnuté v testovacích scénářích [3].

2.2.2 Whitebox testování

Opakem blackbox metody je metoda tzv. whitebox testování (někdy označovaná také jako open, clear nebo glass). Zde je již tester obeznámen s vnitřní strukturou a jeho práce je spojena mimo jiné s analýzou zdrojových kódů. Díky této metodě mohou být některé chyby odhaleny dříve a na základě toho je odstraněn nežádoucí kód aplikace [3].

2.2.3 Greybox testování

Obě výše uvedené metody přinášejí nejen výhody, ale i úskalí. Je proto nasnadě, že nejvhodnější je mnohdy jejich kombinace – metoda tzv. greybox nebo také translucent box testování. V této metodě využíváme omezené znalosti vnitřních datových a programových struktur k navrhování scénářů na úrovní blackbox metody [3].

Nedostatek v podobě neúplného otestování lze tak vykompenzovat dostatečně inteligentními testovacími scénáři [3].

2.3 Druhy testování

Testování se dělí do mnoha druhů, zde budou popsány ty nejpoužívanější seřazeny hierarchicky tak, jak jsou aplikovány v běžných projektových procesech. Pořadí některých se může v praxi lišit v závislosti na charakteristice projektu. Je však nezbytné zdůraznit, že typologií testování je více.[5]

2.3.1 Testování programátory

V praxi bývá toto testování označováno jako „Assembly testing“. Princip spočívá v tom, že bezprostředně po vytvoření kódu je tento kód prověřován programátorem. Zpravidla takovým, který jej nevytvořil, což zajistí odhalení většího množství chyb. Naneštěstí je tento stupeň kontroly poměrně podceňovaný, a to i přesto, že oprava chyb v této vývojové fázi je nejméně nákladná [2].

2.3.1.1 Jednotkové testování

Jednotkové („unit“) testování bývá prováděno programátory taktéž v průběhu implementace. Používá se pro testování izolovaných částí kódu (v objektově orientovaném programování se nejčastěji jedná o testování metod a tříd). Reálné závislosti na další objekty jsou nahrazeny objekty nereálnými, které požadované chování simulují. To výrazně zvyšuje rychlost jejich běhu a celkově je zahrnutí tohoto druhu testování zárukou kvalitnějších zdrojových kódů [2].

2.3.1.2 Integrační testování

Integrační testování je speciální případ testování jednotkového. Využívá však externích zdrojů, jakými mohou být například databáze. Kontroluje se tedy, zda spolu jednotlivé části systému správně spolupracují. Běh je však díky tomu výrazně pomalejší [4].

Oba druhy testů (jak jednotkové, tak integrační) jsou stejně důležité a neměly by při nasazení chybět. Jestliže v aplikaci chybí jednotkové testy, pak bude zřejmě otestována jen pro nejzákladnější scénáře. Chybí-li v aplikaci naopak integrační testy, nikdy nelze říci, že spolu jednotlivé části systému komunikují správně (dotaz na databázi má za následek navrácení očekávaného výsledku apod.) [4].

2.3.1.3 Code review

Pro zvyšování kvality zdrojových slouží „code review“. Vývojáři projektu se rozdělí na skupiny (například dvojice), a každý ve skupině překontroluje práci druhého. Připomínky a nápady na zlepšení jsou pak prezentovány veřejně před celým týmem. Výstupem je nejen předávání znalostí, ale i určení pravidel, podle kterých se bude dále postupovat, což výrazně eliminuje zanášení regresních chyb [3].

[5] Příkladem mohou být typologie „test levels“ nebo „test types“ definované ISTQB (ve stručnosti se jedná o organizaci zabývající se certifikací kvalifikovaných testerů a QA pracovníků).

2.3.2 Funkční testování

Funkční testování je zaměřeno na kontrolu základní funkcionality. To znamená, že se zaměřuje na dílčí funkční bloky aplikace a kontroluje jejich očekávané chování s případnými odchylkami. Aby byla aplikace správně funkčně otestována, je zapotřebí mít k dispozici vhodnou dokumentaci (případy užití, požadavky či tzv. „user stories“). V kontextu s tímto je vhodné uvést i testování **regresní**, jehož cílem je zajistit, aby změny software, jako je přidávání nových funkcí, nebo úpravy stávajících funkcionality, neovlivnily nepříznivě zbylé části aplikace [6].

2.3.3 Systémové testování

Spojení integračních a systémových testů je označována jako fáze SIT (System Integration Tests). Po ověření správné integrace nastává čas na systémové testování. Během těchto testů je aplikace ověřována jako funkční celek. Tyto testy jsou používány v pozdějších fázích vývoje, nalezené chyby jsou opraveny a opět otestovány. Součástí této úrovně jsou jak funkční tak nefunkční testy [4].

2.3.4 Testování použitelnosti, spolehlivosti a podporovatelnosti

Testování použitelnosti se zaměřuje na přehlednost, zapamatovatelnost, naučitelnost, rychlost použití a další aspekty přímo nesouvisející s chybovostí, ale spíše zdokonalováním aplikace. Jedná se do jisté míry o subjektivní hodnocení, jehož účelem bývá vyvolat cílenou diskusi na dané téma [3].

Testování spolehlivosti je zaměřené na procentuálně vyjádřenou dostupnost produktu či služby v čase, případně na množství neplánovaných výpadků a plánovaných odstávek [3].

Testování podporovatelnosti je proces kontroly, zda je součástí dodávky systémová dokumentace, komentovaný zdrojový kód, správa číselníků, a zda architektura umožňuje rozšiřitelnost [4].

2.3.5 Penetrační testování

Penetrační testování patří mezi techniky etického „hackingu“. Účelem je nalezení chyb, na základě kterých se vnikne do systému. Útok může být směrován z vnější sítě na servery umístěné v demilitarizované zóně, případně na vnější rozhraní firewallu. Častěji se však jedná o útoky z vnitřní sítě na síťovou infrastrukturu nebo zranitelné servery [7].

2.3.6 Akceptační testování

User acceptance test (UAT) jsou akceptační testy na straně zákazníka. Pokud všechny předchozí etapy testů proběhly bez větších nedostatků, je možné předat aplikaci zákazníkovi. Ten většinou se svým týmem testerů provede akceptační testy na základě scénářů připravených po domluvě zákazníka a dodavatele [2].

3 Automatizace

Automatizace je proces, při kterém se přiřazují strojům či systémům aktivity, které vykonávali donedávna lidé. Tento proces přispívá k odbourání člověkem prováděné repetetivní činnosti, která bývá, za předpokladu, že ji provádí člověk, nákladnou či dokonce nemožnou [8].

3.1 Základní stavební bloky

Automatizace obsahuje tři základní stavební bloky. Jsou jimi formální modely popisující princip automatizace, samotné programování vyžadující dodržování ověřených metodik, a zpětná vazba plynoucí z výsledků automatizovaných testů [3].

3.2 Výhody a nevýhody automatizace

Automatizace přináší celou řadu obecně známých výhod, ale pochopitelně i několik problémů.

Mezi výhody patří běh regresních testů pro každou novou verzi programu. S tím spojené častější testování, provedení obtížných testů, lepší využití prostředků, konzistence opakovatelnosti testů, a v neposlední řadě také zkrácení doby uvedení na trh [3].

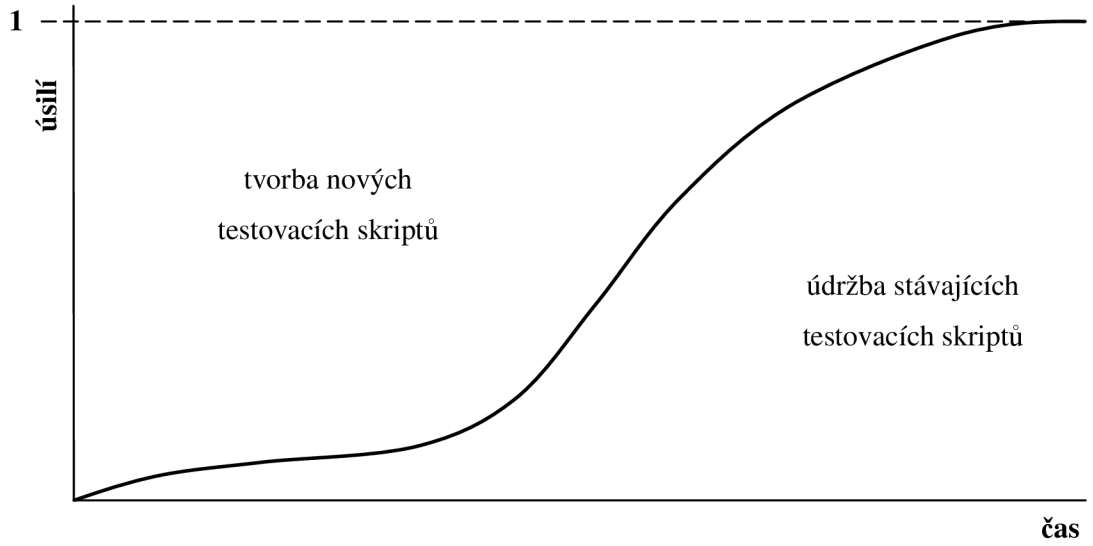
Nevýhodou automatizace testování bývají mnohdy nereálná očekávání vývojáři a dalšími členy týmu do problematiky dostatečně nezasvěcenými, slabá testovací praxe tvůrců testů a v neposlední řadě samotná údržba automatizovaných testovacích skriptů [3].

3.2.1 Porovnání automatizovaného a manuálního testování

Účastí člověka je manuální testování vysoce flexibilní a adaptabilní. I pravděpodobnost detekce defektů je díky jeho schopnostem ovládat software ve špatném stavu a používat objevné testování (tzv. „exploratory testing“) v počáteční vývojové fázi projektu nenahraditelná [3].

Nejdůležitějším aspektem je čas a s ním pochopitelně spojená cena, jelikož samotnou implementací úsilí věnované automatizovaným testům (potažmo testovacím skriptům) nekončí. Ačkoliv je cena opakovaných běhů testů poměrně nízká, údržba testovacích skriptů je oproti tomu poměrně obtížnou a nákladnou záležitostí. Problémem v tomto případě mohou být nejen strukturální změny aplikace, které znamenají manuálně reimplementovat, případně zcela vytvořit nové testovací skripty, ale i další okolnosti působící s časem. Mezi tyto okolnosti patří například změna charakteru projektu, personální změny ve vývojovém týmu (výměna testera) a tak dále [3].

Počáteční investice do vývoje automatizovaných testování se vyplatí v budoucnosti. Obecně lze náročnost údržby shrnout do grafu na obrázku 3: Graf úsilí vývoje automatizovaných testů v závislosti na čase.



Obrázek 3: Graf úsilí vývoje automatizovaných testů v závislosti na čase

Problém je, že počáteční investice je nemalá. V budoucnosti se předpokládá nízká cena přípravy automatizovaných testů vzhledem k možnostem opakovaného použití jejich komponent, k čemuž se přiblíží praktická část této práce [8].

3.3 Automatizace testů

Následuje výčet druhů testování, uvedených v kapitole Druhy testování, které je možné do jisté míry automatizovat. Některé druhy testování, jako například testování použitelnosti, podporovatelnosti, apod. není možné automatizovat. U jiných se naopak jedná o jednorázovou činnost nevyžadující opakování, automatizace by tedy byla velice neefektivní a zbytečná.

Zde je však nezbytné zmínit, že záleží na úhlu pohledu. Některé odborné prameny uvádějí, že automatizace znamená mimo jiné používání podpůrných testovacích nástrojů (může se jednat například o tiketovací systémy, generátory dat, apod.).

3.3.1 Automatizace jednotkového testování

Jednotkové testy jsou spouštěny automaticky při procesu kompilace. Až jsou všechny výsledky testů úspěšné, lze považovat kód za správný a kompletní. Automatizace je vhodná především proto, že jsou testy spouštěny neustále během samotného vývoje a ne pouze několikrát v jeho závěru. Vývojář díky tomu zjistí vadu ihned a je tak velice levné ji opravit. Pro názornost bude uveden příklad.

Programátor nahraje novou verzi aplikace do repozitáře. O několik dnů později tester dostane pokyn o blížícím se releasu a započne s testováním, najde chybu a podá o report. Vývojář mnohdy již nemá povědomí o tom, co a za jakým účelem v kódu měnil. Oprava proto trvá opět nějakou dobu. Oproti tomuto jsou jednotkové testy mnohonásobně rychlejší. Programátor napíše test, sestaví vhodný

kód, okamžitě zjistí, že test selhává, podívá se proto do vytvořeného kódu a v řádech vteřin nalezne chybu. Tu ihned opraví a celý proces tak trvá několik málo minut [3].

Z hlediska ceny chyby, jak bylo uvedeno na obrázku 2: Cena opravy chyby v kapitole Testování softwaru, je to jeden hlavních z důvodů, proč se klade důraz na testy jednotkové.

3.3.2 Automatizace integračního testování

Jednotkové testování je důležité, větší množství chyb však způsobují interakce mezi jednotlivými jednotkami. Automatizace je o to důležitější, protože se díky změnám v rozhraní projevují regresní chyby. Faktem zůstává, že oba druhy testování mnohdy splývají a nemusí být odlišovány. Pokud uvažujeme jednotkové testy nad rozhraními modulů, jedná se fakticky o integrační testování [3].

3.3.3 Test-driven development

Test-driven development (TDD) je metodika vývoje softwarových produktů a služeb založená na krátkých a stále se opakujících krocích. Základním principem je nejprve vytvořit sadu testů a ujistit se, že tyto testy neprocházejí. Následuje implementace samotného kódu a jeho ladění tak, aby testy procházely. Vše je završeno refraktorováním kódu. Tento přístup vede k větší efektivitě vývoje [3].

3.3.4 Automatizace testování výkonu

U výkonnostního testování se automatizace logicky předpokládá. Simulovat ruční cestou stovky až tisíce uživatelů je nemožné.

S přechodem aplikací na vícevrstvou architekturu a prudkým rozvojem v oblasti nových technologií se začaly výrazným způsobem projevovat výkonnostní problémy nově vznikajících systémů. Zpočátku bylo hledání úzkých míst realizováno pomocí manuálně prováděných testů za účasti většího počtu uživatelů. To se při zvyšování počtu současně pracujících uživatelů stávalo organizačně i finančně neúnosné, navíc nebylo možné tyto testy zopakovat s garancí stejné zátěže v jejich průběhu. Začaly proto vznikat podpůrné nástroje pro zátěžové testy, díky čemuž vznikla disciplína výkonnostních testů [9].

Mezi druhy testování, které lze automatizovat, nebylo záměrně zahrnuto funkční testování. Funkčním testováním se budou podrobně zabývat další části této práce, a proto by bylo uvedení podrobnějšího popisu duplicitní.

4 Nástroje pro automatizaci testování

V současnosti existuje velké množství aplikací a rámců[6] umožňujících plnohodnotně nebo alespoň částečně zautomatizovat proces testování. K dispozici je celá řada jednoduchých nástrojů založených na prostém generování událostí, ale i mnoho propracovaných a komplexních nástrojů poskytujících rozsáhlé možnosti automatizace. Zde budou uvedeny ty nejznámější a nejpoužívanější z nich.

4.1 Selenium

Selenium (nebo také Selenium 1 či Selenium RC) je velice populární JavaScriptový rámec využívaný k blackbox testování uživatelského rozhraní webových aplikací. Díky tomuto nástroji lze efektivně vytvářet automatizované testovací skripty provádějící funkční testování webových aplikací. Takto vytvořené skripty lze nejen spouštět v různých prohlížečích a na různých platformách, ale i skripty samotné mohou být vytvářeny v mnoha programovacích jazycích [10]. Mezi největší výhody Selenia patří především jeho spolehlivost a jednotné rozhraní pracující s velkým počtem prohlížečů.

Projekt Selenium byl jeden z prvních open source projektů pracujících na základě interakcí s prohlížeči určený k hromadnému testování. Díky tomu, že je Selenium napsané v JavaScriptu, umožňuje rychlé přidání podpory pro nové prohlížeče, které se na trhu objeví [10].

Selenium RC však má i několik nevýhod. V předchozím odstavci byl JavaScript uveden jako jedna z předností, avšak je to právě on, kdo je zároveň největší slabinou Selenia. Prohlížeče mají běžně nastavený přísný bezpečnostní model vztahující se právě na JavaScript. Ten umožňuje vykonávat i škodlivý kód, jemuž se prohlížeče přirozeně brání, a tím i nepřímo ovlivňují samotné Selenium [10]. Příkladem, kdy bezpečnostní politika prohlížeče znesnadňuje testování, může být nahrávání souboru, kdy webový prohlížeč Internet Explorer zamezuje JavaScriptu změnit hodnotu pole pro vložení souboru.

Další nevýhodou Selenia je, že se v současnosti jedná o již poměrně starý a rozsáhlý projekt. API pro Selenium RC se s postupem času rozrostlo natolik, že je poměrně náročné porozumět a rozhodnout, jakým způsobem jej používat.[7]

---

[6] Rámec (neboli framework) lze chápat v kontextu automatizace testování jako integrovaný systém stanovující pravidla automatizace konkrétního produktu. Tento systém integruje knihovny, zdroje testovacích dat, detaily objektů a nejrůznější znovupoužitelné moduly.

[7] Mějme prvek formuláře – textové pole. Pokud je úkolem vepsat do tohoto pole nějaký řetězec, není zcela jasné, zda pro to použít metodu „`type`“ nebo „`typeKeys`“ (obě metody se přitom chovají stejně).

4.1.1 Selenium IDE

Selenium IDE je plugin do několika prohlížečů umožňující rychlé nahrávání a spouštění testů s využitím konceptu plnohodnotného Selenia. Vše lze realizovat jednoduchým způsobem bez jakéhokoliv programování. Problém je však nejen s přenositelností na jiné prohlížeče, ale takto vytvářené skripty jsou prakticky neudržovatelné a velice špatně editovatelné. Samotné IDE mnohdy zaznamenává irelevantní činnosti, případně některé události nedokáže zaznamenat vůbec. Další problém nastává například při přesunutí prvku do jiné části okna, v takovém případě je pak nezbytné celý scénář nahrávat znovu [3].

Z hlediska pokročilého automatizovaného testování tedy není tento nástroj vhodným řešením.

4.1.2 Webdriver

Wedriver (někdy označován také jako Selenium 2) je rychlý rámec (neboli „framework“) pro automatizované testování webových aplikací [11].[8]

Webdriver používá jiný přístup k řešení totožných problémů. Namísto „samostatné“ JavaScriptové aplikace běžící přímo v prohlížeči používá Webdriver mechanismus pro ovládání prohlížeče v daný moment nejvhodnější [11].[9]

V případech, kdy automatizace prostřednictvím internetových prohlížečů není možná, dokáže Webdriver využít prostředků operačního systému. Příkladem může být simulace psaní na úrovni operačního systému Microsoft Windows. Díky této vlastnosti se tvůrce skriptů může o dost více přiblížit modelování toho, jak uživatel interaguje s prohlížečem, a tak například snadno vyřešit problém s vepisováním do pole pro vstup souboru [11].

4.1.3 Selenium selektory

Selenium dokáže používat mnoho typů selektorů pro identifikace prvků na stránce. Selektorem může být například „`id`“ nebo „`name`“ atribut daného prvku, ale i konkrétní text odkazu. Pokročilejší lokalizaci nabízí CSS a XPath selektory, které budou popsány blíže [12].

4.1.3.1 XPath selektory

XPath je dotazovací jazyk pro XML dokumenty umožňující hledat elementy v DOMu pomocí funkcí a jejich atributů. Jak se XPath používá, bude názorně předvedeno v následující kapitole [11].

---

[8] Selenium 1 a projekt WebDriver byl sjednocen pod produkčním názvem Selenium 2 (nebo také Selenium Webdriver) v roce 2011.

[9] V internetovém prohlížeči Firefox je Webdriver implementován jako doplněk. Pro Internet Explorer je naopak využíváno mechanismu kontroly automatizace. Díky této změně v ovládání prohlížeče můžeme obejít dříve uvedený bezpečnostní model (změna vstupního souboru) a s ním spojená omezení JavaScriptu.

4.1.3.2 Ukázka použití selektorů

Následuje praktická ukázka použití XPath selektorů pro webovou stránku zapsanou následujícím HTML kódem. Ukázka společně se zdrojovým kódem a popisem je inspirovaná zdrojem [12].

```
<html>
     <head>
          <title>Uživatelé</title>
     </head>
     <body>
          <table>
               <tr>
                    <td>Petr Mladý</td>
                    <td>
                         <a href="upravit?id=1">Upravit</a> 
                         <a href="odstranit?id=1">Odstranit</a>
                    </td>
               </tr>
               <tr>
                    <td>Jan Novák</td>
                    <td>
                         <a href="upravit?id=2">Upravit</a> 
                         <a href="odstranit?id=2">Odstranit</a>
                    </td>
               </tr>
          </table>
          <button>
               Potvrdit
          </button>
     </body>
</html>
```

Tabulka 1: Vizualizace webové stránky, jejíž zdrojový HTML kód je uveden výše [12]

Petr Mladý	Upravit Odstranit
Jan Novák	Upravit Odstranit

Úkolem je editovat uživatele Jana Nováka. Pro jeho výběr lze dotaz zapsat pomocí celé cesty stránkou vedoucí až k samotnému prvku (číslo v závorce udává, kolikátý prvek bude vybrán).

```
/html/body/table/tr[2]/td[2]/a[1]
```

Selektory je však třeba oddělit od struktury stránky, jelikož i nepatrná změna způsobí nefunkčnost testů. Omezit se lze pouze na poslední prvek výše uvedené cesty a ten lokalizovat pomocí identifikačního čísla.

```
//a[@href='upravit?id=2']
```

V takovém případě se předpokládá, že tvůrce automatizovaného skriptu zná „id“, což není vhodným řešením. Lze se tedy omezit na odkazy mající za následek volání patřičných funkcí.

```
//a[contains(@href,'upravit')][2]
```

Ani toto není ideální, jelikož se soustředíme na absolutní pozici záznamu „Jan Novák“ v tabulce. Pokud by byl jeho záznam přesunut, například seřazením či přidáním jiných záznamů nebyl by nalezen. Vhodné je omezit se na konkrétní element (tedy řetězec „Jan Novák“).

```
//tr//*[contains(text(), Jan Novák')]
```

Tímto je nalezen element obsahující text „Jan Novák“. Nyní je třeba zvolit odkaz na stejném řádku, v jiném sloupci. I pro toto má XPath řešení. Pomocí klíčového slova „anchor“ lze nalézt předka daného elementu, kterým je atribut <tr>. Přechod na odkaz v jiné buňce na stejném řádku není problémem. Úplný selektor zohledňující změnu pozice elementu na stránce, jeho „id“ a navázání na odkazy je následující.

```
//tr//*[contains(text(),'Jan Novák')]/
ancestor::tr/td/a[contains(text(), 'Upravit')]
```

Uvedený příklad ukazuje, že vhodné zvolení způsobu selekce je klíčové pro udržovatelnost a rozšiřitelnost skriptů. Neochota investovat více času a prostředků pro vytvoření kvalitních selektorů se může stát kamenem úrazu a vést k sáhodlouhým opravám.

4.1.3.3 CSS selektory

XPath je výborný nástroj pro selekci prvků, naneštěstí však složitost dotazů výrazně zpomaluje běh samotných testů, protože vyhodnocení XPath dotazu je výpočetně náročné. Pokud tedy nejsou XPath selektory příliš složité, nabízí se za účelem zvýšení rychlosti jejich změna na CSS selektory. Hlavní výhoda CSS selektorů spočívá v nativní podpoře všech prohlížečů [3].

4.2 JUnit, Jtest a Jmeter

JUnit je testovací rámec pro jazyk Java. Hrál významnou roli při tvorbě techniky test-driven development (TDD), kde je vývoj softwaru rozdělen na krátké vývojové cykly, z nichž každý začíná konstrukcí testovacího případu [13].[10]

Jtest je produkt pro testování a statickou analýzu kódu v jazyce Java. Vyvíjí a podporuje jej společnost Parasoft, která nabízí několik typů licencí. Produkt slouží ke generování scénářů pro unit testy, regresní testování, statickou analýzu a code review [13].

Mezi druhy automatizovaného patří, jak již bylo uvedeno, i testování výkonu. Nejznámější zdarma dostupný nástroj je bezpochyby Apache **Jmeter**. Lze ho použít pro testování JDBC, FTP, LDAP, webových služeb, JMS, HTTP a generických TCP [13].

[10] JUnit byl aplikován na další jazyky – Ada (AUnit), PHP (PHPUnit), C# (NUnit), Python (PyUnit), Fortran (fUnit), Delphi (DUnit), Free Pascal (FPCUnit), Perl, C++ (CPPUnit), R (RUnit) a JavaScript (JSUnit)

4.3 Další nástroje

Selenium bude používáno v praktické části, proto byl jeho popis podrobnější. Pro úplnost je vhodné uvést ještě několik populárních a běžně používaných nástrojů pro automatizaci testování.

4.3.1 Enterprise řešení – HP a IBM nástroje

Před několika lety ovládaly trh specializované firmy Mercury Interactive a Rational Software. Dnes je situace jiná pouze v tom, že se obě staly součástí globálních firem HP a IBM. Takovýto software využívají významní provozovatelé systémů jako finanční instituce, telco operátoři atd. [13]

HP Quality Center je webový systém pro komplexní řízení testování. Využívá technologii client-server a má pět modulů. Pro vlastní testování slouží dva z nich – Test Plan k tvorbě a organizaci testovacích případů, Test Lab ke spuštění testů uložených v Test Plan. V případě manuálních testů vede testera při vyplňování výsledku testu. U automatizovaného testu ukládá jeho výsledek pro srovnání s uloženým a očekávaným výsledkem [13].

HP Quick Test Professional je nástroj sloužící primárně pro automatizaci regresního testování funkcí, které vyžadují interakci s uživatelem. Je určen pro webové rozhraní nebo aplikace využívající operační systém MS Windows. Slouží pro zachycení akcí do skriptu, který pak použije HP Quality Center k provedení testu [13].

IBM Rational Functional Tester je nástroj s podobnými funkcemi jako HP Quick Test Professional. Akce na testované aplikaci jsou zachyceny jako Java nebo Visual Basic.net skript. Je možné zachycovat obrazovky testované aplikace, což výrazně usnadní pozdější změny, které mohou být provedeny pomocí GUI. Tester také specifikuje kontrolní body testu. V průběhu testu se při dosažení kontrolního bodu zaznamenají zvolené parametry (hodnota položky, stav objektu apod.), které pak slouží k porovnání s očekávanými údaji [13].

4.3.2 Střední třída nástrojů pro automatizaci

Pod úrovní produktů kategorie enterprise existuje řada firem nabízejících méně univerzální a zpravidla méně škálovatelné produkty v cenové kategorii od stovek do tisíce dolarů. Pro úplnost budou uvedeny dva příklady – Wapt a TestComplete [13].

Wapt slouží k testování zátěže webově založených aplikací. Produkt simuluje chování uživatelů až tisíců uživatelů na testované webové stránce [13].

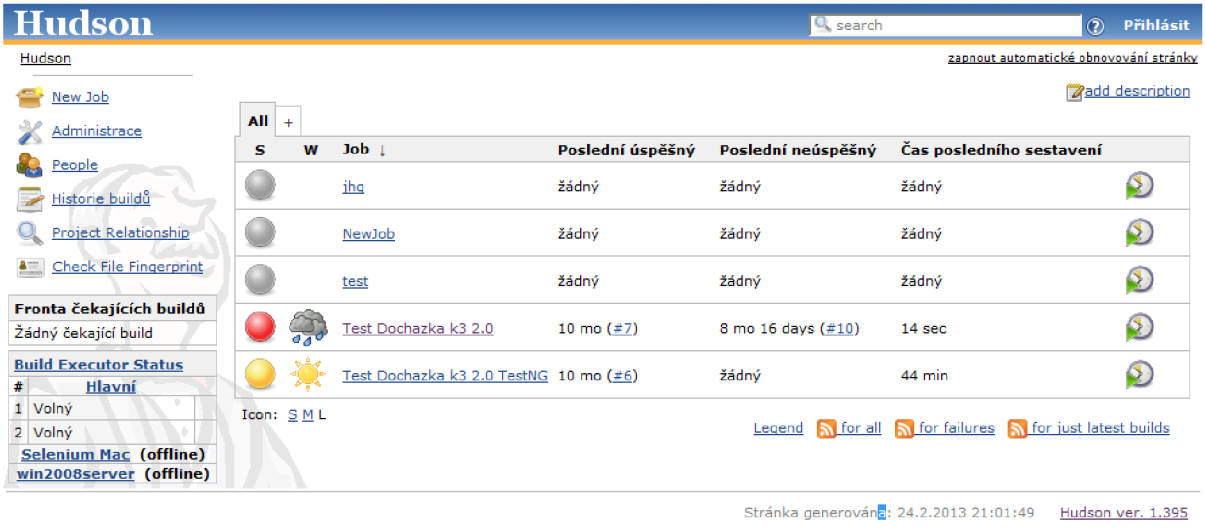
TestComplete je nástroj pro automatizované jednotkové, regresní, a funkční testování. Zvládá i zátěžové testování. Určen je pro Windows a webové aplikace [13].

5 Systémy pro kontinuální integraci

Existují nástroje, díky kterým lze nejen snadno zautomatizovat kompilovací proces, ale i monitorovat změny v systému pro správu verzí. V rámci těchto systémů lze testovat artefakty, zasílat oznámení o změnách, nasazovat prostředí na produkční servery a mnoho dalších velice užitečných pomůcek. Kontinuální integraci (CI) lze chápat jako časté kompilování zdrojových kódů projektu, po kterém běžně dochází ke spouštění automatizovaných testů a dalšímu zpracování výsledků. V rámci této práce bude popsán jeden z nejpoužívanější a v praktické části pak využívaný CI systém – Jenkins [3].

5.1 Hudson / Jenkins

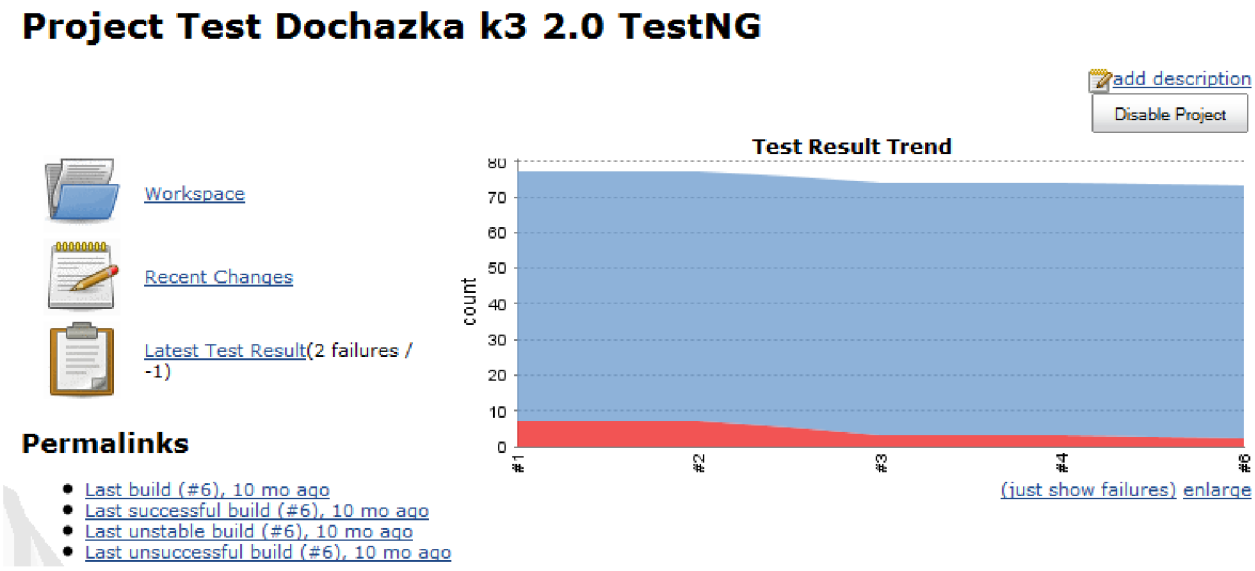
Hudson / Jenkins[11] je webově orientovaný server pro kontinuální integraci napsaný v jazyce Java. Je oblíbený především pro jeho snadné používání, udržovatelnost a širokou rozšiřitelnost v podobě více než 250 pluginů [14].[12] Správa projektů je znázorněna na obrázku 4: Snímek obrazovky správy projektů. Detailní pohled na projekt včetně dalších možností je pak na obrázku 5: Snímek obrazovky konkrétního projektu.



Obrázek 4: Snímek obrazovky správy projektů v Hudsonu

[11] Projekt byl vyvíjen pod názvem Hudson společností Sun Microsystems, Inc. Téměř ihned po akvizici Sunu společností Oracle začalo docházet ke konfliktům mezi ním a komunitou. Projekt se přejmenoval na Jenkins a Hudson je nadále vyvíjen Oraclem, nicméně většina komunity vyvíjí a používá Jenkins [10].

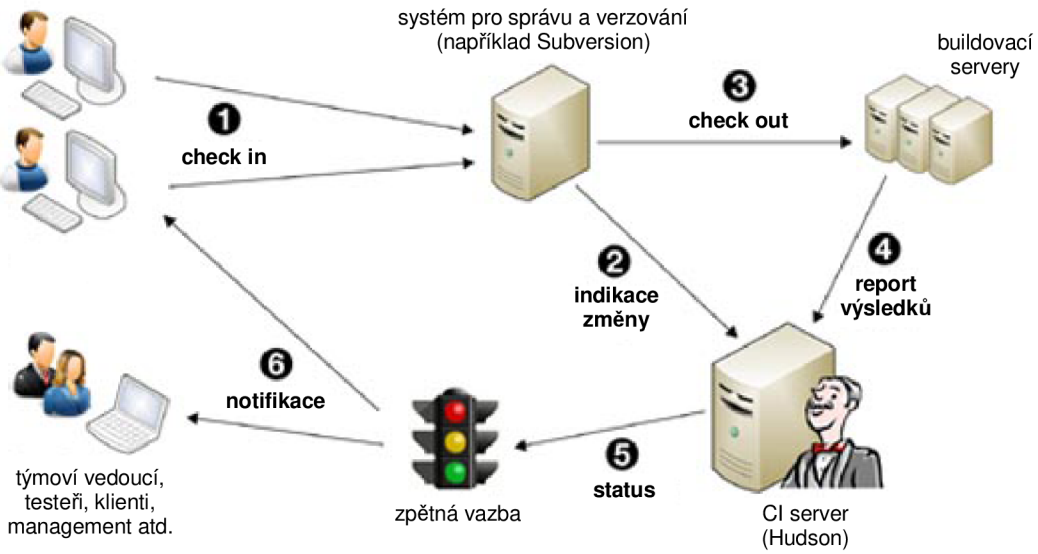
[12] Hudson je v současnosti používán na více než 17000 serverových instalacích po celém světě v malých, středních, ale i velkých společnostech. Za ty největší a nejznámější lze jmenovat například eBay, Hewlett-Packard, MySQL, JBoss, Xerox, Yahoo, LinkedIn, nebo Goldman-Sachs [13].



Obrázek 5: Snímek obrazovky konkrétního projektu

Typický cyklus kontinuální integrace (v případě Hudsonu) je znázorněn na obrázku 6: Typický cyklus kontinuální integrace, k němuž následuje stručný komentář.

Vývojáři komitují nový kód do systému pro správu a verzování zdrojových kódů, například Apache Subversion (1). Server, Hudson, se v pravidelných intervalech dotazuje tohoto systému na změny. Pakliže indikovány nějaké změny, je nový kód zkontrolován buildovacími servery (3), které provedou nové sestavení. Výsledky o úspěšné kompilaci jsou navráceny do Hudsonu (4), který nastaví nový stav projektu jako stabilní, nestabilní či neúspěšný (5). Na závěr je informace o novém stavu zaslána vývojářům a dalším participujícím osobám (6), které zajistí nápravu zjištěných problémů, případně pokračují v zavádění nové funkcionality. Tento cyklus (podle oficiálních stránek Hudsonu) běžně nepřesahuje 15 minut, ovšem u automatizovaných testů toto není relevantní údaj, protože je nezbytné zohlednit doby načtení stránek, pauzy, odezvy na vykonané akce a podobně [14].



Obrázek 6: Typický cyklus kontinuální integrace

6 Praktická část – charakteristika služby Claudu

Následuje praktická část této práce a s ní spojená první kapitola zabývající se stručnou charakteristikou zadavatele projektu a samotného zadání testované služby nazvané Claudu.

6.1 Zadavatel praktické části – IBA CZ, s. r. o.

Společnost IBA CZ, s. r. o. (dále jen IBA CZ) je zadavatelem tohoto projektu, potažmo bakalářské práce. IBA CZ je technologická společnost, která se specializuje na poskytování služeb v oblasti software. Je součástí nadnárodní aliance IBA Group[13], jednoho z největších dodavatelů ICT ve střední a východní Evropě [15].

IBA CZ, se sídlem v Praze a delivery centry v Praze a Brně, má přes 120 kmenových zaměstnanců – IT specialistů, se zaměřením na softwarový vývoj, projektový management a IT konzultace. IBA CZ také spolupracuje intenzivně s mnoha kontraktory a partnerskými společnostmi, díky nimž může flexibilně reagovat na široký okruh potřeb svých zákazníků. Jako jediný Gold Service Partner společnosti Liferay na českém a slovenském trhu poskytuje širokou škálu profesionálních služeb včetně prodeje licencí a první úrovně servisní podpory portálových aplikací. Jako portálová aplikace je realizována i dále popsaná služba Claudu [15].

6.2 Zadání praktické části

Obecným úkolem bylo zajišťovat kvalitu nově vytvářené služby nazvané Claudu v celém rozsahu. Součástí tohoto zadání bylo nejen manuální testování (tedy funkční, systémové, testování použitelnosti atd.), ale i testování automatizované. Specifické části, zabývající se automatizovaným testováním, je pak věnována tato závěrečná bakalářská práce.

Následuje výčet konkrétních požadavků společnosti na vytvářené automatizované testovací skripty.

[13] IBA Group je aliance ICT společností se sídlem v Praze, obchodními pobočkami v USA, Německu, Bulharsku, Kazachstánu, Kypru, Bělorusku a Rusku a s vývojovými centry v ČR a Bělorusku. Je jedním z největších poskytovatelů IT služeb ve střední a východní Evropě. Pomáhá svým klientům v mnoha odvětvích (bankovnictví, doprava, telekomunikace, zdravotnictví, výroba, obchod a státní správa). Hlavními partnery jsou přední světoví poskytovatelé IT služeb, jako IBM, Microsoft, SAP, Lenovo, Check Point, Philips a PTC [14].

Požadavky na automatizované testovací skripty byly následující:

- Musí být provedena analýza všech funkčních komponent projektu a specifikovány testovací případy, které budou posléze odsouhlaseny vedoucím týmu (požadavky jsou v souladu s fakultním zadáním této bakalářské práce).
- Všechny testovací případy akceptované v předchozím bodě, musí být pokryty automatizovanými testovacími skripty.
- Testovací skripty budou implementovány v jazyce Java s využitím nástroje Selenium RC.
- Vytvořené testovací skripty musí být snadno udržovatelné a rozšiřitelné.
- Testy musí být možné spouštět na všech, nejen vývojových, prostředích.
- Návrh musí být použitelný na dalších portálových projektech.
- Testy musí po sobě bezprostředně po dokončení všech testovacích případů odstranit vytvořená data.
- Testy musí být integrovány se systémem pro kontinuální integraci.
- Autor se musí seznámit s automatizací v procesu testování, což je součástí jeho kariérního plánu.
- Autor musí být schopen využít získaných znalostí na dalších projektech.
- V rámci této činnosti musí být provedena analýza dalšího nástroje pro automatizaci (Selenium WebDriver).
- Text vytvořené práce musí být natolik kvalitní, aby mohl být použit jako zdroj informací a vzdělávací materiál v rámci společnosti (například pro nově příchozí QA pracovníky).
- Další výstupy této práce mohou být použity na interních i externích přednáškách a školeních společnosti.

6.3 Vize služby Claudu

Tato kapitola slouží k tomu, aby si čtenář vytvořil představu o službě Claudu jako celku. S výhodou bude využito poměrně obecné definice služby, která byla zformována analytiky společnosti.

Spouštíme službu se jménem Claudu, která má zejména menším společnostem neinformatického rázu vyřešit problém s agendou firemních dat a informací. Jedná se o kontakty, dokumenty, smlouvy, faktury, znalosti, firemní know-how atd.

Stavíme na dostupnosti, tedy data jsou k dispozici kdekoli, kdykoli a lze je jednoduchým způsobem organizovat, případně v nich vyhledávat. Jednotný přístup, kdy vše vypadá a funguje stejným způsobem, tedy používají se jednotné prvky uživatelského rozhraní, je samozřejmostí. Obsahem služby bude velké množství funkcí, které máme k dispozici v téměř hotovém stavu, a které mají smysl pro potencionálního zákazníka. Tato funkcionalita může být na základě aktuálních požadavků v průběhu vývoje měněna.

7 Analýza projektu

Důkladná analýza projektu je velice důležitou součástí zajišťování jeho kvality. Z tohoto důvodu byl na analýzu kladen důraz i v této práci.

7.1 Obecná charakteristika služby

Jedná se o portálovou aplikaci postavenou na systému Liferay (bude dále rozvedeno v následující kapitole), která se nabízí jako služba. To znamená, že je fyzicky provozována na straně IBA CZ, částečně pak Google Apps, a zákazníci k ní přistupují na základě zaplacení pravidelného poplatku. Všechny nově vyvíjené komponenty musí být znovupoužitelné na dalších projektech.

7.2 Liferay Portal

Liferay Portal je podnikový portál v jazyce Java, který umožňuje správu dat, aplikací a procesů prostřednictvím jediného rozhraní [16]. Nabízen je ve dvou edicích:

Liferay Portal Community Edititon, která je bezplatná a volně stažitelná [16].

Liferay Portal Enterprise Edition, která je komerční a placená. Nabízí nejen dlouhodobou podporu, ale i další významné benefity (stabilita, bezpečnost, výkon) [16].

7.2.1 Portlety

Základem každého portálového řešení jsou portlety, logické funkční celky nabízející specifickou funkcionalitu. Vizuálně jsou některé portlety znázorněny v příloze č. 3.

7.3 Proces vývoje

Následuje stručné shrnutí nastaveného procesu vývoje. Popis bude omezen pouze na ty části, které nějakým způsobem ovlivňují zajišťování kvality služby.

7.3.1 Scrum

Scrum je metoda řízení agilního vývoje softwaru a právě touto metodologií byl řízen projekt Claudu. Základem Scrumu je Sprint – periodická činnost, s periodou obvykle dva až tři týdny, která je neustále kontrolována. Vstupními parametry Sprintu jsou tzv. Backlog itemy, což je seznam návrhů na vylepšení stávajícího produktu (služby) nebo opravu chyby. Výstupním parametrem je produkt (služba) s novou funkcionalitou, případně bez zjištěných chyb [17].

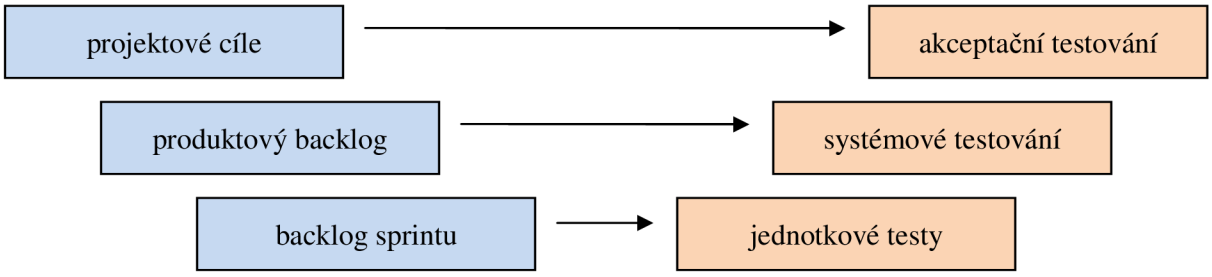
Role účastníků celého procesu se, v jednom z pohledů na tuto metodiku, dělí do dvou skupin, „Kuřata“ a „Prasata“. „Kuřata“ jsou osoby, kterých se problém „pouze týká“ (tzv. Stakeholders), tedy uživatelé produktu, manažeři apod. „Prasata“ jsou osoby přímo souvisící s vývojem produktu (služby) a dále se dělí na následující tři typy. **Product Owner** je osoba zodpovědná za výsledný produkt, která přiřazuje priority jednotlivým backlog itemům. **Scrum Master** je osoba, jejímž úkolem je zajistit hladký průběh Sprintu, organizovat schůzky a řešit administrativní problémy. **Team** jsou programátoři, testeři, analytici, grafici a tak dále [17].

Důležité jsou pravidelné schůzky. Sprint zahajuje „Planning“, jehož výsledkem je podrobné rozplánování vybraných backlog itemů včetně časových odhadů. Každý den probíhá Stand-up, přibližně patnáctiminutová schůzka Teamu. Na ní členové shrnou, co udělali od posledního Stand-upu, co udělají do následujícího, a co je zdržuje. Scrum Master tak sleduje, zda se naplánované odhady shodují s realitou a při případných neshodách přesunuje itemy do dalšího Sprintu [17].

Sprint zakončuje Review, na kterém je funkčnost nového produktu předvedena všem. Po ukončení proběhne Retrospektiva Sprintu, kde se kriticky zhodnotí jeho průběh [17].

7.3.2 V-model

Vývoj aplikace probíhal na principu modifikovaného V-modelu [18]. Obecný V-model vychází z pěti základních vývojových fází projektu, tedy požadavků, návrhu, implementace, testování a běhu, případně údržby. K tomuto je přidružena typologie testů a revizí. Tedy jednotkové a integrační testy, statická analýza kódu, funkční, výkonnostní a akceptační testování, revize návrhu, GUI, produktu, požadavků a projektu. Model byl však účelově pozměněn tak, jak je znázorněno na obrázku 7: Upravený V-model pro účely Scrumu.



Obrázek 7: Upravený V-model pro účely Scrumu [18]

7.3.3 Harmonogram

Analýza a návrh služby Claudu vznikaly během července a srpna 2012. Na začátku září 2012 byla započata implementace. Proces zajišťování kvality začal společně s vývojem a podle principu agilní metodologie byl autor práce, jakožto QA Specialista, zapojen již v raném stádiu vývoje.

Po dobu tří měsíců, tedy září až listopad 2012, probíhalo testování všech druhů pouze manuální cestou a automatizované testování bylo, zejména ke konci tohoto intervalu, ve fázi analýzy a návrhu.

V prosinci 2012, po ustálení komponent, začala implementace požadovaných automatizovaných testovacích skriptů, která byla dokončena v únoru 2013. Od této doby jsou testy již několik měsíců pravidelně spouštěny a rozšiřovány. Bylo tedy dostatek času na závěrečnou analýzu a vyhodnocení.

7.3.4 Podpůrné nástroje – JIRA, Confluence a Alfresco

JIRA – Jak již bylo uvedeno v teoretické části (kapitola Procedurální postupy k eliminaci chyb), pravděpodobnost neopravení nalezených problémů snižují tiketovací. Ve společnosti IBA CZ je používán systém JIRA, který podporuje a usnadňuje proces řízení projektů na úrovni konkrétních požadavků. Základním prvkem tiketovacích systémů jsou tikety, neboli „issues“. JIRA nabízí velké množství podpůrných nástrojů usnadňující opakující se činnost, ale také filtrování tiketů, jejich sdružování do skupin a generování přehledných statistik nejen formou tabulkových výpisů a grafů.

Confluence – Veškerá projektová, technická i obchodní dokumentace je evidována v systému Confluence. Confluence je nástroj pro jednoduché vytváření a správu dokumentů. K dispozici je rozsáhlá škála maker a propojení nejen s výše uvedeným systémem JIRA, ale i dalšími nástroji.

Alfresco – Pokud jsou dokumenty objemnější, případně netextového rázu, jsou ukládány do systému pro správu dat, Alfresco.

7.4 Analýza komponent

Komponenty byly navrženy tak, aby splňovaly princip tzv. Minimum Viable Product (MVP) – tedy produktu či služby obsahujícího pouze ty inovace, které jsou potřebné k nasazení na produkční prostředí. V rámci pilotního provozu byla získána zpětná vazba o tom, co dále implementovat.

Následuje výčet komponent, pro něž byly vytvořeny automatizované testy. Okomentováno je toto vždy v posledních odstavcích dané komponenty s výhodou již zde. Návrh a implementace budou řešeny v dalších kapitolách, popis se tedy omezuje na výčet akcí, které jsou prováděny. Každý test začíná přihlášením a odhlášením ze systému. Třídy dílčích testů jsou pro jednotnost pojmenovány podle názvů portletů tak, jak jsou uvedeny ve zdrojových kódech samotné aplikace.

Snímky některých komponent jsou obsaženy v příloze č. 3.

7.4.1 Kontakty

Adresář kontaktů je realizován jako jednoduchá tabulka (ukázka viz příloha č. 3) se jménem, příjmením, fotografií, kontakty a štítky. V detailu každého kontaktu je k dispozici souhrn těchto informací doplněný QR kódem. Lze také vyhledávat konkrétní zaměstnance přímo v adresáři

prostřednictvím jednoduchého pole pro filtrování. Lze přidávat kontakty vlastní, které jsou bezprostředně synchronizovány s kontakty na straně Googlu.

Automatizovaný testovací skript adresáře kontaktů kromě výše zmíněného přihlášení a odhlášení obsahuje přechod na stránku s kontakty, přidání kontaktu, jeho zobrazení, upravení a odstranění s patřičnými kontrolami.

7.4.2 Import uživatelů a řízení identit

Uživatelé se importují z aplikace Google Apps do portálu při prvním přihlášení a dále pak v pravidelných intervalech. Identity spravuje Google, který zpřístupňuje uživatele jak portálu, tak mobilním zařízením. Funguje celý případ užití od založení uživatele po první přihlášení. Pro správu uživatelů přímo v portálu slouží komponenta Admin portlet, která snadným způsobem dokáže vytvářet, editovat a mazat uživatel, opět s již zmíněnou zpětnou synchronizací s Googlem.

Single sign-on (SSO) přes Google Apps řeší problém více účtů. Uživatel se až na případné výjimky (administrátor společnosti) nemůže přihlašovat do portálu bez Google Apps. Přihlášen může být pod více účty, pouze jediný je však v daný moment aktivní a používaný. Vizuálně se příliš neliší od komponenty „Kontakty“.

Automatizovaný testovací skript Admin portletu je strukturou podobný výše zmíněným kontaktům. Jelikož se za každého nově vytvořeného uživatele na straně Googlu platí poplatek, bylo by jejich vytváření automatizovaným testem zbytečně nákladné. Z tohoto důvodu byla implementována kontrola počtu uživatelů. Pokud test selže například při editaci či mazání uživatele, při jeho dalším běhu je toto rozpoznáno a další uživatel již není vytvořen.

Díky výskytu „CAPTCHA“ při prvním přihlášení, tedy Turingově testu pro rozlišení stroje od člověka, nebyl implementován případ přihlášení nového uživatele po jeho založení.

7.4.3 Systém pro správu obsahu (CMS)

V portálu je implementován vlastní systém pro publikování obsahu závislý na kategoriích. Nejpodstatnějším portletem využívajícím CMS je portlet „Aktuality“, který slouží k zobrazení obsahu na hlavní stránce. Ve zbytku aplikace jsou použity CMS portlety Liferaye, jako například „Rychlé akce“. Příklady nejen výše uvedených CMS portletů jsou znázorněny v příloze č. 3.

Prvním testovacím skriptem je soubor akcí spojených s výše zmíněnými Aktualitami. Formulář pro vytváření formátované aktuality obsahuje pokročilý WYSIWYG editor, který byl komplikací při tvorbě testovacích skriptů. Běžné selektory nebyly schopné zaměřit na místo WYSIWYG editoru, do kterého je vepisován samotný text. Toto se nepodařilo obejít ani voláním Selenium metody „`selenium.focus();`“. Problém byl vyřešen injekcí volání JavaScriptu, který vložil požadovaný obsah. V průběhu testu dochází mimo jiné k přehlašování uživatelů, a tím i kontrole viditelnosti nově založené aktuality dalšími uživateli aplikace.

Druhý testovací skript je velice jednoduchý, jelikož se jedná pouze o textový obsah. Skládá se z přechodu mezi portlety obsahující neměnný text a verifikačních metod na přítomnost tohoto textu. Z důvodu publikování rozdílného obsahu v závislosti na prostředí (každý zákazník vyžaduje individuální obsah) byl však tento druhý test prozatím odstraněn. Po ustálení bude aktualizován seznam obsahu v závislosti na prostředí a test opět zařazen mezi ostatní.

7.4.4 Dokumenty

Liferay transparentně zobrazuje soubory a složky ve vlastních portletech (jako například portlet „Documents and Media“). K dispozici je vyhledávání a online editace dokumentů přes Google Drive frontend. Dokumenty jsou uloženy na straně Google Drive a zpřístupněny tak i na mobilních zařízeních. Ukázka dokumentů je v příloze č. 3.

Dokumenty obsahují jeden z nejrozsáhlejších testovacích scénářů. Kromě nahrávání nových dokumentů, jejich zobrazování, editace a mazání, byla implementována i řada další funkcionality jako například řazení pro snadné vybírání prvku atd. V současnosti jsou dokumenty od základu přepracovávány vývojáři, a proto byl další vývoj testovacích skriptů pozastaven. Dále v něm bude pokračováno po jejich dokončení.

7.4.5 Poznámky

Claudu obsahuje jednoduchou aplikaci pro ukládání úkolů a poznámek uživatele nazvaný. Úkoly lze jednoduše přidávat na každé stránce portálu, řadit je do zápisníků, editovat, filtrovat, označovat za hotové a mazat (inspirací byla aplikace Evernote). Panel poznámek lze kliknutím na šipku v jejich hraně „rozbalit“ či „zabalit“ (tedy zobrazit, případně skrýt za hranici obrazovky). Portlet společně s přidáním nové poznámky je znázorněn v příloze č. 3.

Automatizovaný testovací skript obsahuje mimo přidávání, kontroly, editace a odstraňování poznámek také indikaci skrytí či zobrazení. V případě, že je panel poznámek skryt, dojde před jakýmkoliv krokem k jejich rozbalení, aby byly všechny akce na první pohled patrné.

7.4.6 Jídelníček

Součástí aplikace je portlet zobrazující denní menu požadovaných restaurací a dalších podniků. Data jsou získávána ze serveru Lunchtime, dílčí restaurace se nachází v samostatných záložkách, viz příloha č. 3.

Automatizovaný testovací skript portletu „Jídelníček“ obsahuje přidání, kontrolu a odstranění restaurace. Jelikož není umožněno přidat tutéž restauraci vícekrát, je vkládána unikátní, náhodně vybraná restaurace, která není používána při manuálním testování.

7.4.7 Carousel

Carousel, neboli promítačka, je portlet zobrazující požadovaný počet ve smyčce se opakujících snímků, viz příloha č. 3.

Automatizovaný test přistupuje do sekce pro konfiguraci portletu, kde vkládá nový snímek, edituje jej a posléze i odstraňuje.

7.4.8 Mobilní verze

Mobilní aplikace reflektuje díky vhodně nastaveným mobilním pravidlům PC verzi portálu se specifickým tématem a omezeným množstvím portletů. Sestává z domovské obrazovky s aktualitami a rychlým hledáním v adresáři kontaktů, kde lze stáhnout vizitku nebo na daný kontakt přímo zatelefonovat, viz příloha č. 3. Autentizace je řešena rovněž přes Google SSO.

Automatizované testy pro mobilní zařízení jsou plánovány v dalším vývoji projektu, kdy dojde k přidání další funkcionality a ustálení komponent. V současnosti jsou testovány pouze „Aktuality“.

7.4.9 Nejvíce diskutované a nedávné dokumenty

Tyto dva portlety úzce souvisí s diskusí a dokumenty. V portletu „Nejvíce diskutované“ se zobrazují vlákna diskuse mající největší množství příspěvků. „Nedávné dokumenty“ oproti tomu poskytují přehled o dokumentech, na které má uživatel oprávnění, a se kterými je napříč aplikací pracováno. Oba portlety jsou znázorněny v příloze č. 3.

Kontrolováno je prokliknutí do sekce „Dokumentů“, případně „Diskuse“. Prokliknutí na detail daného dokumentu nelze vhodným způsobem automatizovat, jelikož oba portlety zobrazují pouze čtyři položky. Pokud by byly spuštěny automatizované testy, a zároveň by se v aplikaci pohybovali jiní uživatelé, což nelze vyloučit, hledané položky by po přechodu na stránku s těmito portlety nemusely být k dispozici. Obdobně by dopadlo jejich paralelní spouštění. Testy by díky tomu hlásily neúspěch, ačkoliv by k žádným chybám nedošlo.

7.4.10 Další komponenty

Dalšími portlety jsou „Diskuse“, „Anketa“, „Audit service“, „Vyhledávání“ a další. Jedná se však o vestavěné Liferay komponenty upravené pouze tématem, do kterých nebylo výrazným způsobem zasahováno. Díky tomu nebylo třeba vytvářet pro tyto portlety automatizované testy.

Pakliže by se do komponent nějakým způsobem zasahovalo, například byla-li by přidávána nová funkcionalita, automatizované testování by začalo mít smysl a nové testy pro ně budou případně implementovány.

V kapitole Další vývoj jsou popsány komponenty, které jsou v současnosti ve fázi návrhu, případně se implementují, a pro něž je naplánován vývoj automatizovaných testovacích skriptů.

8 Návrh řešení

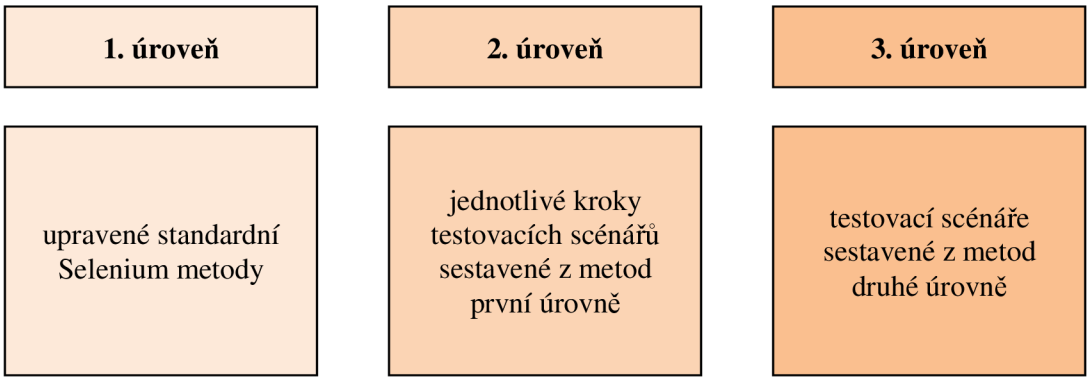
Následující kapitola stručně shrnuje návrh řešení automatizovaných testovacích skriptů. Při něm bylo s výhodou využito návrhového vzoru Page Objects, který byl účelově pozměněn pro portálové účely. Dále jsou popsána dílčí prostředí, na kterých je v současnosti aplikace nasazena a jejich postupné zavádění. V závěru je popsán návrh řešení problematiky unikátnosti dat a odůvodněna lokalizace automatizovaných testovacích skriptů.

8.1 Page Objects

Page Objects je návrhový vzor, jehož modifikované verze bylo s výhodou využito při návrhu a následně i implementaci automatizovaných testovací skriptů. Page Objects agreguje dva druhy tříd – testy a stránky. „Page“ obsahuje metody, které znamenají služby nabízené stránkou z pohledu uživatele. Např. tlačítka, vyplnění formuláře, odkazy atd. Další metody umožňují zjišťovat stav, v jakém se stránka nachází (například počet záznamů v tabulce, přítomnost chybové hlášky atd.). Všechny tyto metody jsou implementovány pomocí Selenium API [19].

Testy samotné pak naopak nepoužívají Selenium API vůbec. Pouze sestavují testovací scénář voláním sérií služeb na jednotlivých stránkách. S těmito jednoduchými pravidly se dají vytvářet velmi přehledné a snadným způsobem editovatelné testovací skripty [19].

Tento návrhový vzor byl pro účely automatizovaného testování portálových aplikací modifikován. Byla vytvořena mezivrstva simulující volání Selenia a samotné volání včetně dodatečné funkcionality bylo zahrnuto v předchozím odstavci zmíněné vrstvě komunikující přímo s API Selenia. Díky tomu vznikla tříúrovňová struktura, jejíž princip je znázorněn na obrázku 8: Tříúrovňová struktura testů.



Obrázek 8: Tříúrovňová struktura testů

8.2 Prostředí

Samotná služba je provozována na několika prostředích, z nichž pouze jediné je přístupné zákazníkovi. Automatizované testy musely brát v potaz přechody mezi těmito prostředími. Následuje výčet jednotlivých prostředí tak, jak byla postupně zapojována do procesu vývoje.

8.2.1 Výčet všech prostředí

Vývojové prostředí (DEV) slouží k nasazování právě vytvořených a dosud neotestovaných komponent. Jednalo se o první spuštěné prostřední, na které je průběžně nasazována nová funkcionalita vyvíjená vývojáři. Zde je provedeno důkladné, nejen automatizované, testování, a až poté je nová funkcionalita postupně přenášena na další prostředí.

Demonstrační prostředí (DEMO) je prostředí určené k prezentaci na výběrových řízeních, přednáškách, ale i Review všech Sprintů. Na DEMO jsou nasazovány nejen stabilní věci, ale i poměrně nové komponenty, které splňují základní předpoklady prezentovatelnosti.

Testovací prostředí (TEST) bylo spuštěno po Releasu 1, kdy již bylo vytvořeno dostatečné množství komponent. Jedná se o identické prostředí k prostředí produkčnímu.

Produkční prostředí (PROD) bylo zavedeno společně s prostředím testovacím. K tomuto prostředí přistupují zákazníci, zbylá výše zmiňovaná prostředí jsou určena pouze pro účely vývoje služby.

8.3 Unikátní data

Pokud by byla v rámci automatizovaných testů vytvářena stále stejná data, mohlo by v některých případech docházet ke kolizím totožných názvů. Dalším problémem by byla identifikace prvku mezi několika dalšími stejného názvu a typu.

Tento problém byl vyřešen generováním aktuálního data a času namísto běžných řetězců. Pokud se tedy vytváří nový kontakt, jako jméno a příjmení je použito například „automatizovaný test 28.3.2013 8:29:18“, což usnadňuje následnou selekci prvků v seznamech kontaktů.

8.4 Lokalizace

Názvy metod i proměnných jsou, až na nejnižší úroveň, kde by to mohlo být matoucí, psány česky. Důvodem byla přehlednost, kdy zdrojový text skriptu lze přímo porovnat s testovacími plány. Z názvu metody je díky tomu možné ihned identifikovat daný bod testovacího plánu. Pokud by byl někdy v budoucnu projekt nabízen jako produkt a ne služba, pravděpodobně by se k němu přikládaly i tyto testovací skripty. V takovém případě zákazník českou lokalizaci ocení.

9 Realizace

Následující kapitola shrnuje postup, který byl zvolen při vývoji automatizovaných testovacích skriptů.

9.1 Vývojové prostředí a podpůrné nástroje

Následuje výčet několika stěžejních produktů, kterých bylo využito v průběhu implementace. Pro vývoj sloužilo vývojové prostředí NetBeans IDE poskytující veškeré nezbytné podpůrné nástroje. K sestavování aplikací bylo pak s výhodou využito nástroje Apache Maven a celý projekt byl sdružován do jediného Maven modulu, aby mohl být snadno integrován do prostředí Jenkins.

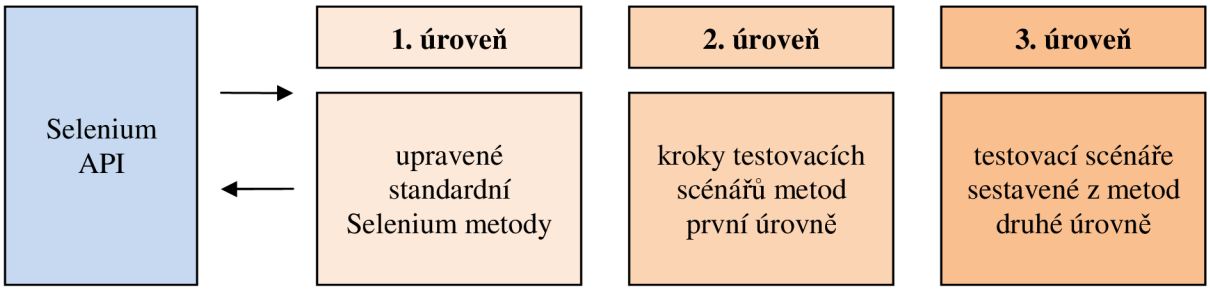
9.1.1 Selenium Server

Jak již bylo uvedeno v teoretické části, jádro celého Selenium RC je Selenium Server. Ten překládá příkazy zapsané v testovacích skriptech, zasílá je prostřednictvím JavaScriptu k vykonání, a zároveň se stará o navrácení výsledku. Selenium Server ve verzi 2.25.0 je k dispozici jako JAR soubor.

9.2 Implementace

Následující kapitola shrnuje stěžejní části testovacích skriptů. Jak již bylo uvedeno dříve, celý projekt je realizován jako Maven modul. Hlavním balíkem je „Test Packages“ sdružující v sobě všechny potřebné testovací skripty. Testy se spouštějí automaticky po sestavení projektu a ve třídě `TestSuite.java` je uveden seznam tříd, tedy testů, spuštěných po sestavení projektu. Dílčí části testovacích skriptů, s odkazem na již zmiňovanou tříúrovňovou strukturu v kapitole Page Objects, budou popsány podrobněji v samostatných podkapitolách.

Návaznost vytvořených tříd na Selenium rozhraní je pak znázorněno na obrázku 9: Návaznost vytvořených tříd na Selenium API.



Obrázek 9: Návaznost vytvořených tříd na Selenium API

9.2.1 SeleniumUtils

První a nejnižší vrstva je zastoupena třídou `SeleniumUtils.java`. Při implementaci metod této třídy bylo s výhodou využito práce Stanislava Šiveňa, bývalého pracovníka společnosti, který některé z nich v minulosti zavedl. Na základě aktuálních požadavků pak byly implementovány metody další, případně vhodně upraveny ty stávající.

Obsah této třídy lze považovat za „nádstavbu“ základních volání metod Selenia. Standardní volání jsou zde upravena do vhodnější podoby. Jsou doplněna formátování nejen vstupů, výstupů a odchytávání chyb s přeformátováním chybových hlášení, ale v neposlední řadě i přehledné logování pro účely dalšího vývoje a ladění. Implementováno bylo velké množství metod, z nichž pro ukázku bude uvedena metoda pro operace se zaškrtávacími poli – `sCheck`. Jedná se o rozšíření metody `selenium.check` obsahující další výše uvedené modifikace.

V následujícím bloku zdrojového kódu je implementováno nejen čekání na načtení elementu (`waitToElement`), ale i kontrola na jeho přítomnost s patřičným voláním metod Selenia (`selenium.check` a `selenium.uncheck`). V závěru je odchycení chyby v případě nenalezení prvku. Navrací se informace o jeho přítomnosti, případně nepřítomnosti.

```
/* ZAŠKRTÁVACÍ POLE (CHECKBOX) */
public boolean sCheck(String LogText, String Sobject, boolean Status) {
     waitToElement(Sobject, 5);
     if (selenium.isElementPresent(Sobject)) {
          if (Status) {
               selenium.check(Sobject);
               logTC("* Zaškrtnutí položky na prvku: [" + LogText + "]");
          } else {
               selenium.uncheck(Sobject);
               logTC("* Odškrtnutí položky na prvku: [" + LogText + "]");
          }
     } else {
          logTCError("* Prvek nenalezen: [" + LogText + "]:" + Sobject);
     }
     return selenium.isElementPresent(Sobject);
}
```

9.2.2 SupportUtils

Třída `SupportUtils.java` zastává druhou vrstvu. Obsahuje volání metod v předchozí kapitole zmíněné třídy `SeleniumUtils.java`. V této třídě jsou implementovány dílčí kroky testovacích scénářů. Pro názornost je opět uveden krátký příklad toho, jak daná metoda, v tomto případě `pridatRestauraci`, vypadá.

```
/* PŘIDÁNÍ NOVÉ RESTAURACE */
public void pridatRestauraci(String nazev, String klic) {
	logTC("=======>> Přidání restaurace <<=======");
	sClick("Konfigurace", Selectors.tlacitkoKonfigurace);
	waitToPage(Constants.cekaniNacteni);
	sClick("Přidat restauraci", Selectors.tlacitkoPridaniRestaurace);
	waitToPage(Constants.cekaniNacteni);
	sType("Název", Selectors.poleRestauraceNazev, nazev);
	sType("Klíč", Selectors.poleRestauraceKlic, klic);
	sClick("Uložit", Selectors.tlacitkoUlozitRestauraci);
	waitToPage(Constants.cekaniNacteni);
	logTC(">> Restaurace přidána <<");
}
```

Jak je vidět v uvedeném bloku zdrojového kódu, zápis je díky odstínění logování, zachytávání chyb, a dalších akcí třídou `SeleniumUtils.java`, velice přehledný a snadným způsobem editovatelný.

9.2.2.1 Jednotlivé testy

Třetí a poslední vrstvou jsou samotné testovací scénáře zapsané jako volání metod třídy `SupportUtils.java`. Scénáře jsou díky tomu velice přehledné a lze se v nich díky tomu nejen velice snadným způsobem orientovat, ale i provádět všechny potřebné změny.

Bylo ověřeno, že i člověk, který má pouze základní povědomí o problematice, je schopen editovat takovýmto způsobem zapsané testovací scénáře a provádět patřičné změny. Nemusí přitom znát principy skriptování v nástroji Selenium a nemusí dokonce umět programovat v jazyce Java.

Selenium API tedy bylo odstíněno, zde je již (zkrácená) ukázka testovacího scénáře „Kontaktů“ třídy `PhoneBook.java`.

```
/* KONTAKTY */
public void testPhoneBook() {
	prihlasit(login, heslo, kontrolaPrihlaseni);
	stranka(kontakty);
	pridatKontakt(jmeno, prijmeni, telefon, email, stitky);
	zkontrolovatKontakt(email);
	zobrazitKontakt(email);
	stranka(kontakty);
	upravitKontakt(jmeno2, prijmeni2, telefon2, email2, stitky2);
	editovatKontakt(email);
	odstranitKontakt(email);
	odhlasit();
}
```

9.2.3 Selectors

Třída `Selectors.java` v sobě sdružuje všechny CSS a XPath selektory. Důvodem pro zavedení této třídy bylo nejen to, že selektory se mohou v rámci testů opakovat, ale i fakt, že pokud by byly složitější selektory zapsány přímo v `SupportUtils.java`, stávala by se orientace i editace díky jejich délkám zápisu velice nepřehlednou. Zde jsou všechny selektory sdružovány do logických celků a lze je tak velice snadněji editovat. Ukázka bloku selektorů pro testovací skript portletu Carousel vypadá následovně.

```
/* CAROUSEL */
public static String tlacitkoCarouselKonfigurace = "css=img[alt=\"Konfi...
public static String tlacitkoCarouselNovy = "css=a[class='create-link']";
public static String poleCarouselNadpis = "css=input[id='_CarouselPortl...
public static String poleCarouselCesta = "css=input[id='_CarouselPortle...
public static String tlacitkoCarouselUlozit = "css=input[id='_CarouselP...
public static String tlacitkoCarouselZpet = "css=a[class='portlet-icon-...
```

Prvky nebyly striktně vybírány na základě „`ID`“, jelikož právě v `ID` byl specifický řetězec znaků a číslic závislý prostředí. Při pokusu o spuštění na jiném prostředí by tak testy hlásily nenalezení daných prvků. Z toho důvodu byla použita neměnná část podřetězce `ID` nebo jiný parametr, například „`name`“, případně lokalizace přes jiné okolní prvky (obdobně, jako tomu bylo u příkladu v kapitole Ukázka použití selektorů).

9.2.4 Properties

Soubory s nastaveními specifickými pro prostředí jsou uloženy v souborech `settings-prostredi.properties`. V závislosti na nastaveném prostředí se pak tyto informace dále používají jako proměnné. Aktuálně obsahují properties nejen URL prostředí, ale i seznam uživatelů s hesly a pojmenování dílčích stránek.

9.2.5 POM

Soubor `pom.xml` obsahuje stejně jako každý projekt vytvořený v Apache Maven konfiguraci projektu. V této konfiguraci se nachází nejen název projektu, informace o tvůrci a závislosti na další projekty, ale i fáze sestavování projektu. S výhodou bylo využito možnosti profilování, kdy pro každé prostředí byly vydefinovány základní koncepce.

Stěžejní je zde informace o URL adrese specifického prostředí. Před sestavením projektu je tedy nezbytné zkontrolovat právě nastavený profil v závislosti na prostředí pro běh testů.

9.3 Nastavení prostředí Jenkins

Pro instalaci prostředí Jenkins je nejprve nezbytné nainstalovat Java Development Kit (JDK) a nástroje pro správu a automatizaci sestavování aplikací Apache Maven.[14]

9.3.1 Slave agent a nastavení cest

V prostředí Jenkins se musí vytvořit slave agent a nastavit cesty nejen mezi JDK a Mavenem, ale i do Git repozitáře obsahujícího testovací skripty. Pak lze již vytvořit nový projekt („job“). Názvy parametrů je možné používat přímo v prostředí Jenkins `(EMAIL)` nebo v `pom.xml` souboru projektu (`${JAVACLASS}`, `${ENVURL}` a `${BROWSER}`).

9.3.2 První spuštění

Při prvním sestavení projektu se načtou všechna data z přednastaveného GIT repozitáře do specifického adresáře. Pakliže by někdo v budoucnu cokoliv v uvedené cestě změnil, bude notifikován prostřednictvím emailu o nestabilních či neúspěšných sestaveních. Z tohoto důvodu je potřebné zvážit, zda načíst celý repozitář projektu nebo pouze adresář s modulem obsahujícím testovací skripty.[15]

9.4 Běh testů

Jak již bylo uvedeno výše, testy se spustí v NetBeans IDE bezprostředně po sestavení aplikace. Předpokladem běhu je však spuštění Selenium Serveru před začátkem sestavování. Spustit testy z příkazového řádku lze v kombinaci s definovaným profilem v `pom.xml` například takto:
„`mvn clean install`“ a „`mvn integration-test -P claudu-demo`“.

Prvním stádiem je otevření okna akcí, ve kterém jsou formou seznamu zaznamenávány všechny vykonané události společně s časem. Následuje otevření okna prohlížeče, ve kterém probíhá samotný test. Všechny výpisy jsou zaznamenávány do výstupu NetBeans IDE. Jejich princip bude popsán v jedné z následujících kapitol.

9.4.1 Periodicita spouštění

Automatizované testy jsou spouštěny kontrolně každou noc. Zároveň dojde k jejich spuštění po každém nasazení na jakékoliv z vývojových prostředí. Automatizované testy nejsou využívány pouze QA pracovníky při nutnosti otestovat aplikaci. Vývojáři si mohou po úpravě stávající

[14] Podrobný návod k nastavení prostředí Jenkins je k dispozici na přiloženém CD.

[15] Po otevření projektu v Netbeans IDE se nastaví patřičný profil a po sestavení projektu (klávesová zkratka Shift + F11) se spustí testovací skripty.

komponenty či vytvoření nové spustit specifickou sadu testů pro ověření, zda úprava neměla za následek zanesení regresních chyb.

Doba běhu celého balíku testů se pohybuje okolo dvaceti minut, je však závislá nejen na výkonnosti daného stroje, na kterém jsou testy spouštěny, ale především rychlosti připojení.

9.4.2 Logování

Akce automatizovaného testovacího skriptu je vypisována prostřednictvím `LogTC();` a označena specifickým znakem. Společně s metodami SeleniumUtils (uvedeno v kapitole SeleniumUtils) byla tato konvence zavedena bývalým pracovníkem společnosti, Stanislavem Šiveňem. V posledních odstavcích této kapitoly je uvedeno také zhodnocení tohoto přístupu.
Zde již seznam:

- Kliknutí myši: *
- Zápis dat: _
- Výběr položky na dané komponentě: ^
- Zaškrtnutí zaškrtávacího pole: +
- Odškrtnutí zaškrtávacího pole: -
- Verifikace dat: ?
- Verifikace dat s ukončením při neshodě: ?x

Každý krok testovacího plánu začíná a končí souhrnným hlášením. Krok je mimo jiné označen číslem, které se postupně inkrementuje a na konci testu je vypsán seznam případných chyb. Díky číslu kroku, ve kterém k chybě došlo, lze pak snadno nalézt problematické místo. Všechny výpisy jsou výborným pomocníkem v průběhu vytváření automatizovaných testovacích skriptů, avšak pro běžný běh jsou zbytečné a znesnadňují orientaci v logu. Z toho důvodu bylo logování po dokončení daného testovacího skriptu odstraněno se zachováním pouze informativních hlášení o provedení dílčího kroku testovacího scénáře.

V rámci této práce bylo však ve zdrojových kódech logování pro názornost zachováno. Ukázka logu po dokončení testu se nachází v příloze č. 1.

Jak již bylo řešeno v úvodu této kapitoly, bylo navázáno na již zavedenou konvenci. Při vyhodnocování této práce se však došlo k závěru, že daná konvence se v praxi stává téměř nevyužitelnou. Při standardním bezproblémovém běhu testů je logování, jak bylo uvedeno výše, zbytečné, jelikož podstatná je pouze informace o úspěchu, případně neúspěchu testu. Jestliže dojde k pádu, ať už při vývoji nebo v produkčním prostředí, je problémové místo velice rychle odhaleno na základě slovního popisu akce – další reprezentace tedy není potřeba.

Za zavedením této konvence v minulosti údajně byla mimo jiné snaha používat zpětně logovací výpisy k popisu testovacích scénářů v tiketovacím systému. Tento způsob zápisu se však ukázal jako nepřehledný a nevhodný. Pro názornost byla v testech konvence na základě výše uvedeného

seznamu ponechána. Bylo však rozhodnuto, že na dalších projektech již logy nebudou tyto znaky obsahovat.

9.4.3 Snímky obrazovky

Po spuštění testu je vytvořen specifický adresář, jehož název je složen z názvu balíku testu, data a času spuštění, do něhož jsou ukládány snímky obrazovky.

Princip pořizování snímků obrazovek je následující. Před každým načítáním stránky a po každém načtení stránky je vytvořen snímek obrazovky, který má standardně zelené pozadí. Po odeslání požadavku je soubor označen předponou „before“, po načtení požadavku oproti tomu předponou „after“ (v každém případě však začíná číslem kroku). Pokud během testu dojde k chybě, je na základě toho vytvořen specifický snímek obrazovky s červeným pozadím. Obsahuje předponu „ERROR“ následovanou metodou, která byla volána.

Pokud test projde v pořádku, nejsou snímky obrazovky zapotřebí a adresář vytvořený na začátku testu je smazán. V opačném případě je adresář zachován pro snadné a rychlé odhalení problému, který mohl být snímkem obrazovky zaznamenán. Ukázky snímků obrazovek se nachází v příloze č. 2.

9.4.4 Prohlížeče

Snímky obrazovky, popsané v předchozí kapitole, nepodporují všechny prohlížeče. To však není problém, protože nová funkcionalita je ověřena ve všech podporovaných prohlížečích. Pokud by tedy při běhu automatizovaných testovacích skriptů došlo k výskytu nějakých regresních funkčních problémů, je na základě předchozího pozorování potvrzeno, že se tyto problémy projeví ve všech prohlížečích. Díky tomu není problémem, že jsou testy spouštěny pouze v prohlížeči Firefox.

Po odstranění této implementace lze spustit automatizované testy i v dalších prohlížečích. Do budoucna lze tento případ ošetřit přidáním vhodných podmínek, prozatím však toto nebylo nezbytné.

Pro běh testů bylo využito profilování prohlížeče. Ve vytvořeném a parametrizovaném profilu Mozilla Firefox byly vypnuty aktualizace, potvrzeny všechny potřebné certifikáty a akceptována všechna hlášení. Toto nastavení umožňuje bezproblémový chod aplikace bez výskytu neočekávaných událostí (aktualizace, žádosti o potvrzení certifikátů apod.). Selenium Server je v tomto případě nezbytné spustit s parametrem `-firefoxProfileTemplate` a cestou k adresáři s uloženým profilem.

Profile Manager prohlížeče Mozilla Firefox lze aktivovat spuštěním Mozilla Firefoxu z konzole s parametrem `-ProfileManager no-remote`. Je však nezbytné zkontrolovat, aby prohlížeč již v nějakém okně neběžel a jestli ano, zavřít jej. Pokud by se tak nestalo, otevře se nové okno prohlížeče namísto spuštění Profile Manageru.

10 Vyhodnocení

Od vytvoření testovacích skriptů již uplynulo několik měsíců. Během této doby byl dostatek času nejen na jejich další rozšiřování, ale i vyhodnocení celé praktické části této práce.

10.1 Ověření správnosti

Vzhledem k tomu, že mnohé testy byly implementovány již před několika měsíci, byl dostatek času na ověření jejich správnosti a splnění všech požadavků. Následuje podrobnější popis ověřování.

10.1.1 Rozšiřitelnost

Po dokončení základních automatizovaných skriptů bylo vydefinováno několik změnových požadavků. Ty se týkaly nejen úprav již vytvořeného, ale i zavedení funkcionality zcela nové. Příkladem může být vytvoření nové komponenty „Kalendář“.[16]

Zanesení případných změn bylo snadné a rychlé, stejně jako přidání nových testovacích případů. V nových testech byla použita sada hotových metod a doimplementovány byly pouze metody pro danou komponentu specifické. Případná změna komponenty byla realizována změnou selektoru. Předpoklad snadné rozšiřitelnosti tedy byl splněn.

10.1.2 Změna prostředí

Při vývoji základních testovacích skriptů bylo k dispozici jediné prostředí. Po zavedení prostředí dalšího se objevilo několik problémů se selekcí prvků. Bylo zjištěno, že některé selektory, především ty s parametrem „`id`“, v sobě obsahovaly řetězce specifické pro prostředí. Po změně prostředí tak nebylo možné některé prvky vybrat. Několika málo úpravami, především změnami jednoduchých CSS na složitější XPath selektory s pokročilejší selekcí přes větší množství okolních prvků, byly problémy odstraněny.

Běh testů nebyl ničím narušen i dalšími přechody na nově zaváděná prostředí.

10.1.3 Nepokryté případy

Ne všechny komponenty bylo možné efektivně pokrýt automatizovanými testy. Příkladem může být Chat, který by vyžadoval neúměrné množství času vynaloženého na automatizaci nebo testovací případy vyžadující subjektivní posouzení. Po každém dokončení běhu testů a vyhodnocení výsledků tedy následuje manuální dotestování nepokrytých částí testovacích scénářů.

[16] Tato komponenta není zmíněna v analýze, jelikož byla vytvářena o několik měsíců později. V odevzdaných zdrojových kódech této práce však již k dispozici je a testy pro ni mohou být spuštěny.

10.2 Aktuální stav

Aktuálně se projekt nachází ve Sprintu 12, Release 1. Základní požadovaná funkcionalita je implementována a většina týmu se soustředí převážně na ladění nalezených chyb, případně zapracovávání připomínek k použitelnosti. Vytváření dalších komponent je již naplánováno a částečně se na nich pracuje.

Jelikož jsou jednotlivé chyby opravovány podstatně rychleji, než byla vyvíjena nová funkcionalita, dochází i k častějšímu nasazování na více prostředí. Díky tomu jsou automatizované testy pravidelně používány. Testovací skripty pokrývají všechny scénáře, které byly sjednány v zadání projektu, případně na základě změnových požadavků. Plní tedy roli, která se očekávala.

10.3 Úspora času

Pro manuální otestování muselo být vynaloženo mnoho člověkohodin týdně. Jedno „kolo“ testování na jednom prostředí trvá pro současné testovací plány přibližně čtyři hodiny. Spuštěním automatizovaných testovacích skriptů se ušetří společně se započítáním drobného ladění více než polovina času. Podle běžných konvencí je do časových odhadů započítáván pouze čas, který by byl jinak vynaložen manuálním testováním. Pokud tedy testy běží pravidelně například každou noc, nezapočítává se do celkové úspory času toto do jisté míry „kontrolní“ testování, ale pouze manuální testování po nasazení.

Uvažujeme-li v průměru čtyři nasazování týdně i případné další spouštění na lokálním prostředí na vyžádání, pak úspora času mírně přesahuje jeden „člověkoden“ (tzv. „manday“, dále uváděný pod zkratkou „MD“) týdně. Společně s dalším vývojem projektu budou přibývat nejen prostředí, ale i další funkcionalita, která bude pokrývána automatizovanými testy. Poměrně brzy, za přibližně tři měsíce, se úspora času s velkou pravděpodobností přiblíží dvěma MD týdně, a bude i nadále růst.

Je však nezbytné uvažovat nejen dobu, která byla vynaložena na implementaci, ale také dobu, kterou zabere údržba, případně manuální spouštění testů a vyhodnocování výsledků. Čas strávený analýzou, návrhem a implementací je snadným způsobem změřitelný (na základě vykázaného času v rámci společnosti). Společnost v tomto ohledu ušetřila významné množství času a finančních prostředků, protože tato práce byla realizována převážně ve volném čase.

Odhadnout čas na údržbu, manuální spouštění a vyhodnocení výsledků je v současnosti poměrně těžké. Za poslední dva měsíce pilotního provozu se pohybovala zhruba okolo jednoho MD měsíčně a předpokládá se mírný nárůst společně s další implementací. Pokud se uvažují všechny výše zmíněné údaje, předpokládá se, že návratnost pro společnost je v horizontu přibližně půl roku. S aktuálně plánovaným zaváděním zcela nové funkcionality do aplikace pak přibližně rok. Za předpokladu, že by však automatizované testovací skripty byly vyvíjeny v rámci standardní

pracovní doby, u služby tohoto typu by návratnost bez přidávání čehokoliv dozajista přesahovala více než rok. S novou funkcionalitou by se pak mohla přiblížit i roku a půl.

Projekt bude dozajista pokračovat delší dobu, z dlouhodobého hlediska tedy byla investice výhodná.

10.4 Kdy testovat automatizovaně

Na základě této práce byla zjištěna následující fakta o automatizovaném testování.

Obecně řečeno, testy ušetří tím více času, čím vícekrát jsou spouštěny. Přesněji je to doba, která by byla strávena manuálním testováním stejné funkcionality (CI systémy mnohdy nepočítaje, viz předchozí kapitola). Testy nikdy neautomatizují vše a nelze se na ně plně spolehnout, vždy je třeba alespoň zběžná manuální kontrola. Hlavním přínosem není nalezení problému, ale úspora času, který by byl stráven manuálním testováním.[17]

10.4.1 Kdy začít automatizovat testování

Díky změnovým požadavkům v prvotních fázích projektu by bylo vytváření automatizovaných testů neefektivní, proto je vhodnější věnovat čas spíše testování manuálnímu. Automatizované testy však nemá smysl z hlediska úspory času na projektu vytvářet ani v momentě, kdy projekt téměř končí a předává se zákazníkovi nebo případně do oddělení technické podpory.

Může se ovšem stát, a nebývá to výjimkou, že si takové testování nebo dodávku testů objedná zákazník, případně že oddělení podpory převezme jejich údržbu a bude je efektivně využívat i nadále. Ideální čas pro implementaci, za předpokladu, že již byla vypracována analýza a návrh, je tak zpravidla po ustálení požadované funkcionality dílčích komponent.

Před započetím realizace myšlenky automatizovaně testovat je však zapotřebí uvědomit si především to, že testy jsou „hloupé“ a stále se opakují. Mohou odhalit regresní chyby nebo nenasazení některých komponent na dané prostředí, velice zřídkakdy však zjistí přítomnost chyby zcela nové. Hrozí zde riziko získání falešné jistoty z jejich výsledků.

Tester by neměl automatizovaným testům věřit, byť je sám vytvořil.

[17] Jako vedlejší přínos automatizace může být zmíněna eliminace poklesu motivace. Tento pokles by v delším časovém horizontu byl přirozeně způsobován prováděním stále stejné a opakující se činnosti.

11 Další vývoj

Služba Claudu je i nadále vyvíjena a díky tomu trvá požadavek na zajišťování kvality. Následující podkapitoly ve stručnosti shrnují další vývoj projektu, tedy i plánovanou tvorbu dalších automatizovaných testovacích skriptů.

11.1 Blízká budoucnost

V současnosti používá Claudu v pilotním provozu ústav LaSArIs, Fakulty informatiky Masarykovy univerzity. Služba je prezentována na konferencích (například JIC 120s) a ve firmách, na které cílí.

Znovupoužitelné komponenty začínají být uplatňovány v jiných projektech, kde mohou být s postupem času používány i v rámci této práce vytvořené automatizované testovací skripty.

11.1.1 Další vývoj automatizovaných testovacích skriptů

Stejně jako je naplánován vývoj projektu, je s ním spojený i vývoj a zdokonalování automatizovaných testovacích skriptů. Byl sestaven seznam „issues“, které budou realizovány v nejbližší době.

11.1.1.1 Role a oprávnění

V současnosti existují pouze tři role – administrátor, editor a běžný uživatel. Vyvíjí se však portlet pro vytváření skupin s nejrůznějšími oprávněními. Na toto bylo v průběhu vývoje testovacích skriptů pamatováno. Do standardních metod byla zahrnuta nejen podpora načítání uživatelů z CSV souboru, ale i další funkcionalita pro snadný přechod na princip oprávnění.

11.1.1.2 Chat

Pokud se rozhodne, že chat bude používán i na dalších projektech, budou pro něj i přes větší náročnost a investici implementovány automatizované testy. Díky využití chatu na jiných projektech se vytvoření testů z dlouhodobého hlediska vyplatí a tvorba přinese nové poznatky.

11.1.1.3 Další komponenty

Každý Sprint s sebou přináší další novou funkcionalitu menšího či většího rozsahu. Testy proto jsou a budou pravidelně aktualizovány. V současnosti je to například portlet „Fotogalerie“, který se nyní vyvíjí a nebyl proto do této práce zahrnut.

11.1.1.4 Migrace skriptů Selenia RC na Webdriver

V rámci dalšího vývoje byl na několika jiných projektech a webových aplikacích vyzkoušen následovník Selenia RC, tedy Selenium Webdriver (dále jen Webdriver). Používání tohoto nástroje bylo, po pochopení principů na Selenium RC, intuitivnější, rychlejší a i dodatečné hledání řešení

problému bylo výrazně snadnější. Webdriver je totiž v současnosti podporován rozsáhlou komunitou. Oproti Seleniu RC je řešená problematika aktuálnější, protože tak, jak se vyvíjejí nové technologie, vyvíjí se i Webdriver, zatímco Selenium RC již v současnosti stagnuje.

Důvody, proč přejít na Webdriver, byly již částečně uvedeny v kapitole Webdriver, ve stručnosti však lze zmínit následující:

Menší a kompaktnější API, které je více objektově orientované, což usnadňuje práci. Dále lepší emulace uživatelských interakcí s využitím nativních událostí. V neposlední řadě pak podpora složitějších a náročnějších uživatelských interakcí a podpora dodavatelů jednotlivých prohlížečů [20].

Migrace skriptů zapsaných v jazyce Java je jednou z nejsnadnějších. Prvním krokem je změna způsobu, jakým je získávána instance Selenia. To se zajistí následovně:

```
WebDriver driver = new FirefoxDriver();
Selenium selenium = new WebDriverBackedSelenium(driver, "siteURL");
```

K získání potřebné implementace z instance Selenia slouží WrapsDriver:

```
WebDriver driver = ((WrapsDriver) selen) getWrappedDriver ();
```

To umožňuje pokračovat v předání instance běžným způsobem a v případě potřeby aplikovat součásti Webdriveru. Řešení dalších kolizí je individuální, většina z nich je však způsobena tím, že Webdriver akurátněji simuluje chování uživatele [20].[18]

11.1.1.5 Selenium Grid

Bude zváženo zavedení nástroje Selenium Grid, který umožňuje paralelně spouštět automatizované testovací skripty nejen na více prohlížečích, ale i na více strojích. Distribuce zátěže přispěje k rychlejšímu běhu testů. V současnosti však k tomuto přístupu nebyl důvod, jelikož doba běhu testů nijak nebrání v jejich používání. Smysl tento nástroj bude mít, pakliže testy výrazněji narostou.

11.2 Vzdálenější plány

Služba Claudu se plánuje dále rozvíjet a zasahovat i do produktové sféry. Plánuje se realizace dalších „odnoží“ Claudu, jako Claudu Enterprise, Public, SME a EDU. Ve stádiu, kdy se bude zákazníkům nabízet Claudu ne již jako služba, ale produkt, je velice pravděpodobné, že součástí dodávky budou i automatizované testovací skripty společně s návodem k jejich použití, případně modifikaci.

V takovém případě zákazník ocení přehlednost, intuitivnost a jejich kvalitní zpracování automatizovaných testovacích skriptů, stejně jako případně část této práce ve formě podpůrné dokumentace.

---

[18] Příkladem takové kolize může být trojice volání metod Selenia
`keyDown("name", "t"); keyPress("name", "t"); keyUp("name", "t");`
Zatímco výstup Selenia RC je řetězec „t“, WebDriver vypíše „ttt“. [19]

Závěr

Cílem této práce bylo vypracovat rešerši na téma automatizované testování webových aplikací. Součástí práce byla praktická část zaměřená na vytvoření automatizovaných testovacích skriptů pro webovou službu Claudu vyvíjenou společností IBA CZ, s. r. o.

Teoretická část ve stručnosti shrnula proces řízení kvality a s ním spojené testování. Podrobněji byla rozebrána problematika automatizace společně s nástroji toto umožňujícími. Závěr teoretické části byl věnován systémům pro kontinuální integraci (CI). Text, ale i další výstupy práce, budou nadále používány v rámci společnosti jako studijní materiál a pro interní a externí přednášky.

V rámci praktické části bylo popsáno schválené zadání. Následoval popis nastaveného procesu vývoje, použitých metodik a podpůrných nástrojů. Podrobněji byla rozebrána analýza dílčích komponent, na základě kterých byl sestaven návrh automatizovaných testovacích skriptů zohledňující všechny kladené požadavky.

Implementační část započala v době, kdy byly komponenty standardizovány do takové míry, že byla přidávána pouze nová funkcionalita bez výrazné změny té stávající. Docházelo však k drobným úpravám, díky čemuž byly testy zdokonaleny do takové míry, že v současnosti další úpravy i většího rázu zpravidla nemají neočekávaný vliv na jejich běh. Po dokončení implementační části byly skripty integrovány se systémem pro kontinuální integraci Jenkins, díky čemuž mohou být spouštěny nejen po každém nasazení nové verze aplikace.

V současnosti jsou testovací skripty spouštěny téměř každý den na různých prostředích s různými parametry. Samozřejmostí je i kontrolní spouštění v nočních hodinách s periodou jednoho dne. Vývojáři si mohou po úpravě stávající komponenty či vytvoření nové také spustit specifickou sadu testů pro ověření, zda úprava neměla za následek zanesení regresních chyb.

Úspora času překračuje jeden člověkoden týdně, který by musel být vynaložen QA pracovníkem na manuální otestování aplikace. S dalším vývojem bude mít tato úspora vzrůstající tendenci. S ohledem na čas věnovaný analýze, implementaci, údržbě, manuálnímu spouštění a vyhodnocení je návratnost za vytvoření testovacích skriptů odhadována na půl roku za předpokladu nepřidávání nové funkcionality. S plánovanou funkcionalitou pak jeden rok.

Od vytvoření testovacích skriptů již uplynulo několik měsíců, během kterých byly laděny a rozšiřovány, v čemž se pokračuje i nadále. Lze prohlásit, že byla splněna očekávání, která si od tohoto projektu nejen vývojový tým sliboval. Skripty jsou pravidelně spouštěny, šetří čas a finance s rozumnou návratností, jsou udržovatelné a snadným způsobem rozšiřitelné. Přinesly zároveň mnoho novým poznatků, které budou dále uplatňovány.

Díky zavedení znovupoužitelných principů a seznámení se autora s problematikou automatizovaného testování webových aplikací je možné vytvořit automatizované testy na dalších projektech výrazně rychlejším způsobem.

Literatura

[1] SUCHÝ, Ondřej. PROFINIT. *Quality Assurance*. 2011, 40 s.

[2] HLAVA, Tomáš. Fáze a úrovně provádění testů. *Testování softwaru: Portál zabývající se problematikou ověřování kvality software. Manuální i automatizované testování* [online]. 2011 [cit. 2013-03-01]. Dostupné z: http://testovanisoftwaru.cz/tag/akceptacni-testovani/#system

[3] BĚLOCH, Tomáš. IBA CZ, s. r. o. *Konference Software Summer Camp 2012: Řízení kvality a testování*. Brno, 2012.

[4] MROZEK, Jakub. *Zdroják.cz* [online]. 2012 [cit. 2013-03-01]. Dostupné z: http://www.zdrojak.cz/clanky/javascript-na-serveru-testovani-a-kontinualni-integrace/

[5] KŘENA, Bohuslav a Radek KOČÍ. *Úvod do softwarového inženýrství*. Brno, 2010. Studijní opora. Fakulta informačních technologií, Vysoké učení technické v Brně.

[6] Regresní testy. *EI Prague* [online]. 2013 [cit. 2013-04-07]. Dostupné z: http://www.eiprague.cz/index.php/cz/sluby/quality-assurance/regresni-testy

[7] Penetrační testování. *Experia Group* [online]. 2012 [cit. 2013-03-01]. Dostupné z: http://www.experia.cz/penetracni-testovani/

[8] MAŘÍK, Radek. CA LABS. *Automatizace testování*. 2007, 28 s.

[9] SURÝ, Jiří. Výkonnostní testy podnikových aplikací. IT Systems [online]. 2008, č. 12 [cit. 2013-04-09]. Dostupné z: http://www.systemonline.cz/sprava-it/vykonnostni-testy-podnikovych-aplikaci.htm

[10] ARMELI-BATTANA, Sebastiano. IBM - DEVELOPERWORKS. *Get started with Selenium 2: End-to-end functional testing of your web applications inmultiple browsers*. IBM Corporation, 2012.

[11] KO, Ellen. Introducing WebDriver. In: *Google Open Source Blog* [online]. 2009 [cit. 2013-03-01]. Dostupné z: http://google-opensource.blogspot.cz/2009/05/introducing-webdriver.html

[12] RAČANSKÝ, Luboš. XPath a Selenium testy. In: *AspectWorks* [online]. 2012 [cit. 2013-03-01]. Dostupné z: http://www.aspectworks.com/2012/01/xpath-a-selenium-testy

[13] Automatizované testy aplikací typu klient-server. *Clever and Smart* [online]. 2011 [cit. 2013-03-01]. Dostupné z: http://www.cleverandsmart.cz/automatizovane-testy-aplikaci-typu-klient-server/

[14] WIEST, Simon. Hudson – Your Escape from "Integration Hell". *Methods & Tools* [online]. 2010 [cit. 2013-03-01]. Dostupné z: http://www.methodsandtools.com/tools/tools.php?hudson

[15] IBA CZ, s. r. o. *IBA cz: Complex IT Service Provider* [online]. Brno, 2013 [cit. 2013-04-07]. Dostupné z: www.ibacz.eu

[16] *LIFERAY: Enterprise. Open Source. For Life.* [online]. 2013 [cit. 2013-04-07]. Dostupné z: http://www.liferay.com

[17] Oddělení vývoje systémových služeb: SCRUM. In: *Masarykova univerzita: Ústav výpočetní techniky* [online]. 2013 [cit. 2013-04-07]. Dostupné z: http://www.muni.cz/ics/925600/web/scrum

[18] ABEL, Anders. Test and Verification in Scrum. In: *Passion for Coding: Software Development is a Job – Coding is a Passion* [online]. 2012 [cit. 2013-04-07]. Dostupné z: http://coding.abel.nu/2012/04/test-and-verification-in-scrum/

[19] MÜLLER, Pavel. Selenium a návrhový vzor Page Objects. In: *ASPECT WORKS: vyvíjíme software* [online]. 2010 [cit. 2013-04-07]. Dostupné z: http://www.aspectworks.com/2010/06/selenium-a-navrhovy-vzor-page-objects

[20] Migrating From Selenium RC to Selenium WebDriver. In: *SeleniumHQ: Browser Automation* [online]. 2013 [cit. 2013-04-07]. Dostupné z: http://docs.seleniumhq.org/docs/appendix_migrating_from_rc_to_webdriver.jsp

Seznam příloh

Příloha 1. Zkrácený stav logu po dokončení testu portletu pro administraci.

Příloha 2. Ukázka snímků obrazovek po úspěšném a neúspěšném běhu testu.

Příloha 3. Ukázky některých testovaných komponent.

Příloha 4. CD.

- zdrojové kódy implementovaných automatizovaných testovacích skriptů v adresáři `/tests/`
- spustitelný JAR soubor Selenium Serveru (verze 2.25.0) v adresáři `/selenium/`
- kolekce návodů pro spuštění v adresáři `/readme/`
- tato práce v adresáři `/thesis/`
- vhodným způsobem nastavený profil Mozilla Firefox v adresáři `/firefox/`
- ukázky spuštění a běhu testů formou záznamů obrazovek v adresáři `/screencasts/`
- návod na nastavení prostředí Jenkins v adresáři `/jenkins/`

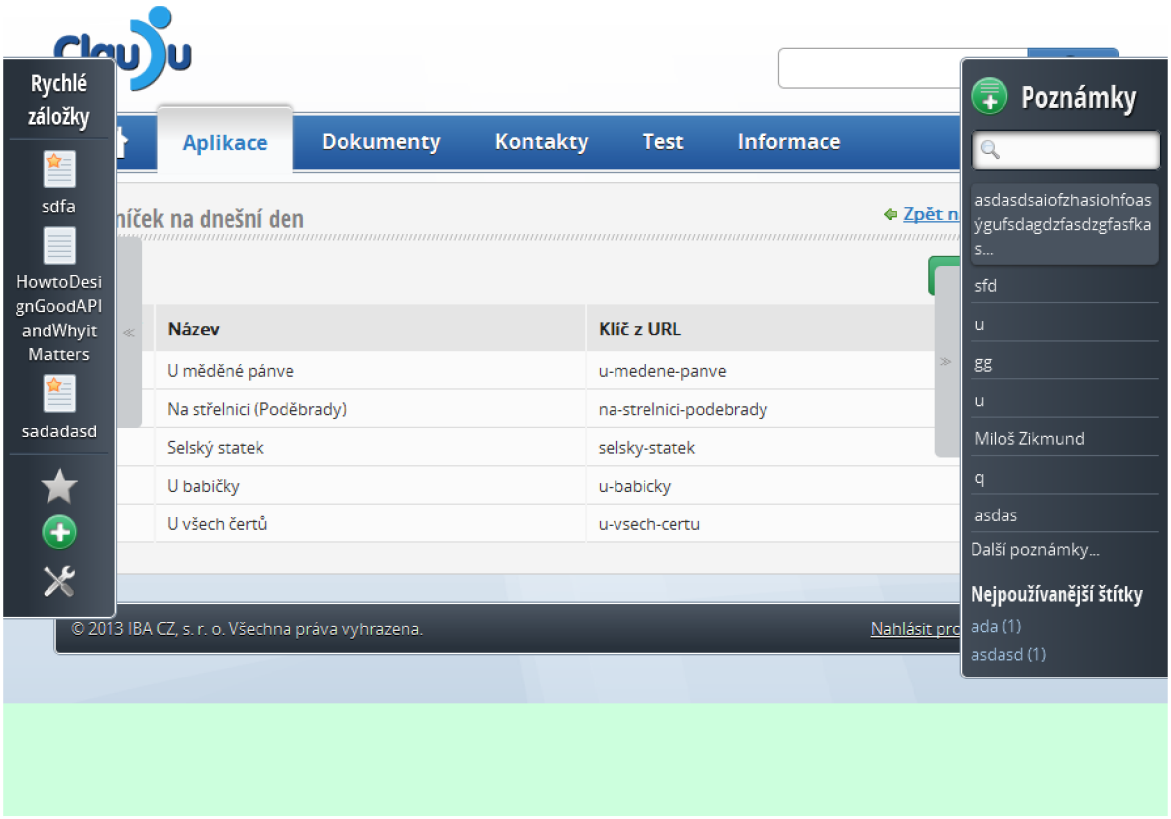
Příloha č. 1

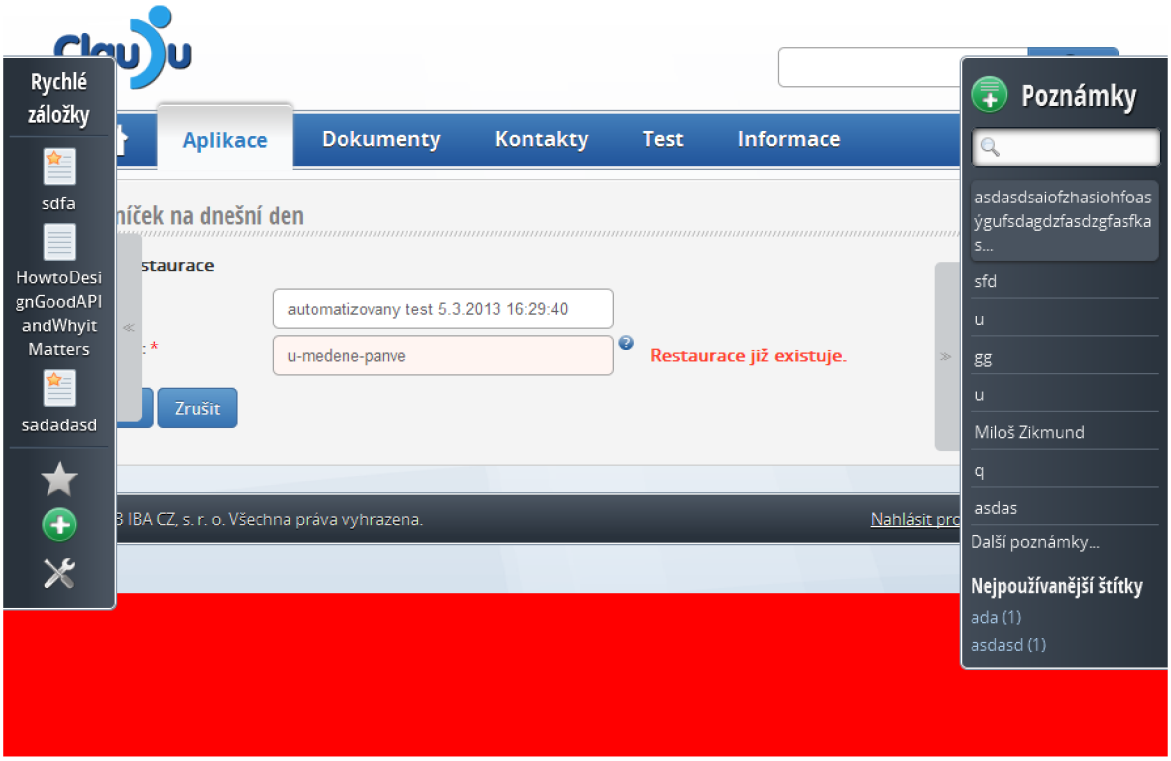
Zkrácený stav logu po dokončení testu portletu pro administraci

```
-------------------------------------------------------------------------
 T E S T S
-------------------------------------------------------------------------
Running TestSuite
TESTOVACÍ SADA PRO PROSTŘEDÍ: http://ibox-dev.ibatest.cz/
TESTOVACÍ SADA PRO PROHLÍŽEČ: *firefox

(1) Kontrola složky pro obrazovky (c:\selenium\): true
(2) Vytvořena složka pro obrazovky: c:\selenium\Admin 24.03.2013 19_24_07


>>>>>>>>>>>>>>>>>>>>>>>>>>>> Start testu >>>>>>>>>>>>>>>>>>>>>>>>>>>>
http://ibox-dev.ibatest.cz/, Admin ; Start testu: 24.3.2013 19:24:07
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

(4) =======>> Zalogování uživatele: tomas.beloch@ibatest.cz <<=======
(5) - Zápis údajů do prvku: [Email] tomas.beloch@ibatest.cz
(6) - Zápis údajů do prvku: [Password]
(7) * Kliknutí na prvek: [Potvrdit]
(8) > > Text odpovídá: "Odhlásit se"
(9) >> Zalogování proběhlo <<
(10) Dělí se text: Zobrazuji 1 až 9 z celkem 9 záznamů
(11) Počet registrovaných uživatelů: 9 (test může pokračovat)
(12) =======>> Přidání uživatele: <<=======
(13) * Kliknutí na prvek: [Přidat uživatele]
(14) - Zápis údajů do prvku: [Jméno] Automatizovany
(15) - Zápis údajů do prvku: [Příjmení] Test
...
(21) * Kliknutí na prvek: [Uložit]
(22) >> Uživatel přidán <<
(23) =======>> Upravení uživatele: <<=======
(24) * Kliknutí na prvek: [Upravit]
(25) - Zápis údajů do prvku: [Jméno] Automatizovany upraveno
(26) - Zápis údajů do prvku: [Příjmení] Test upraveno
...
(31) * Kliknutí na prvek: [Uložit]
(32) >> Uživatel upraven <<
(33) =======>> Odstranění uživatele: <<=======
(34) * Kliknutí na prvek: [Odstranit]
(35) >> Uživatel odstraněn <<
(36) =======>> Odlogování uživatele: <<=======
(37) * Kliknutí na prvek: [Odhlásit z portálu]
(38) >> Odhlášen <<

------------------------- Výsledek testu ---------------------------------
http://ibox-dev.ibatest.cz/, http://ibox-dev.ibatest.cz/; Admin
Čas testu: 24.3.2013 19:25:24 ; Délka testu: 1364149524(s) ; Výsledek: OK
-------------------------------------------------------------------------
Results :
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 78.006 sec
-------------------------------------------------------------------------
BUILD SUCCESS
-------------------------------------------------------------------------
Total time: 1:24.012s
Finished at: Sun Mar 24 19:25:26 CET 2013, Final Memory: 12M/108M
-------------------------------------------------------------------------
```

Příloha č. 2

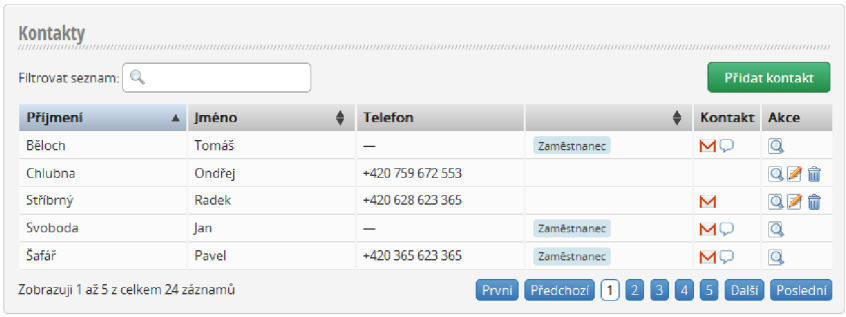
Ukázka snímků obrazovek po úspěšném a neúspěšném běhu testu



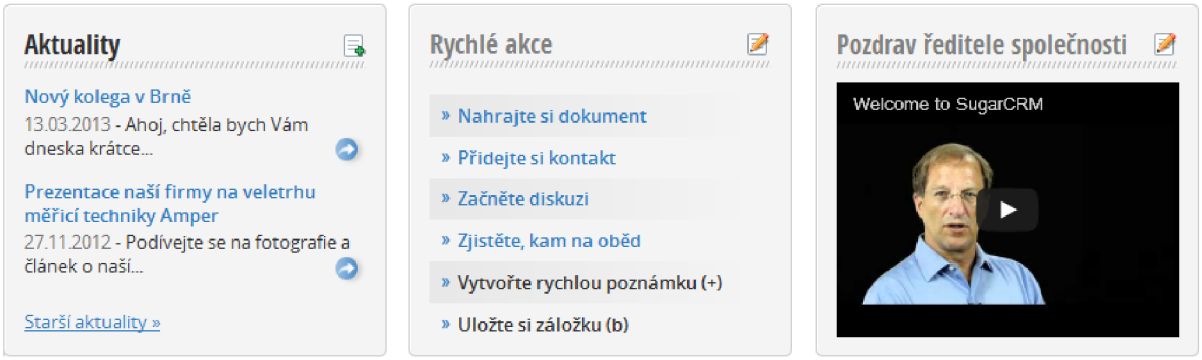


Příloha č. 3

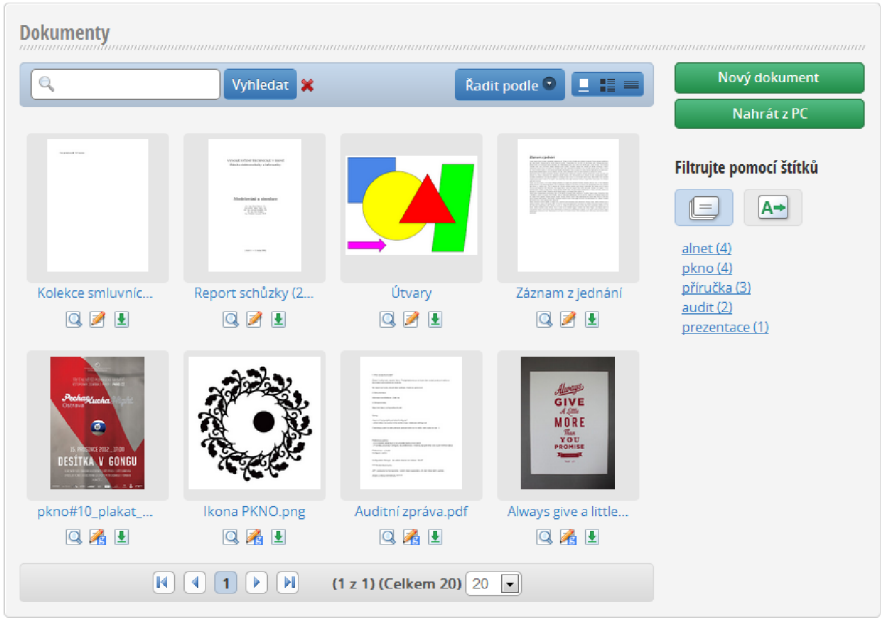
Ukázky některých testovaných komponent



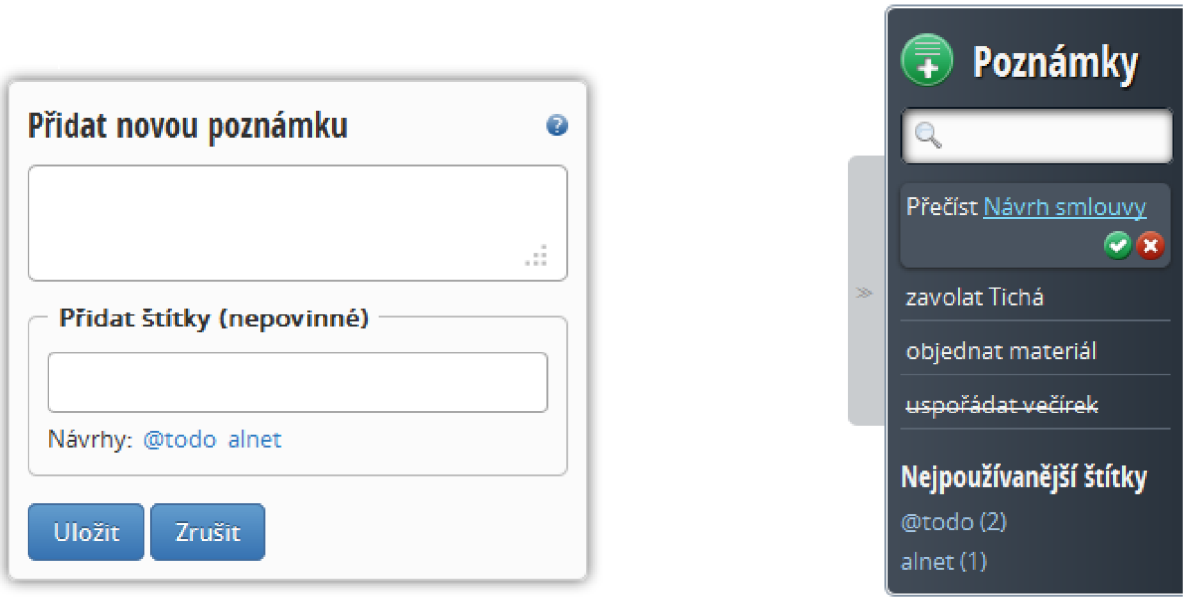
Kontakty



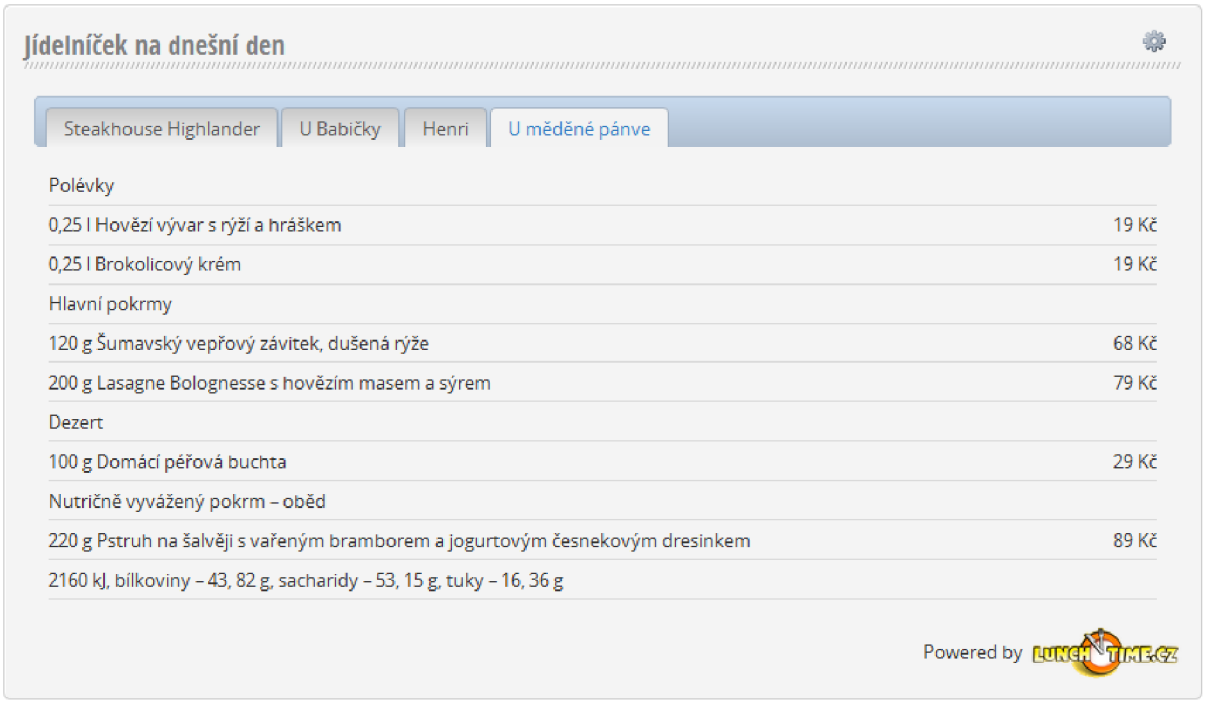
CMS – systém pro správu a publikování obsahu



Přepracovaná Liferay Document Library

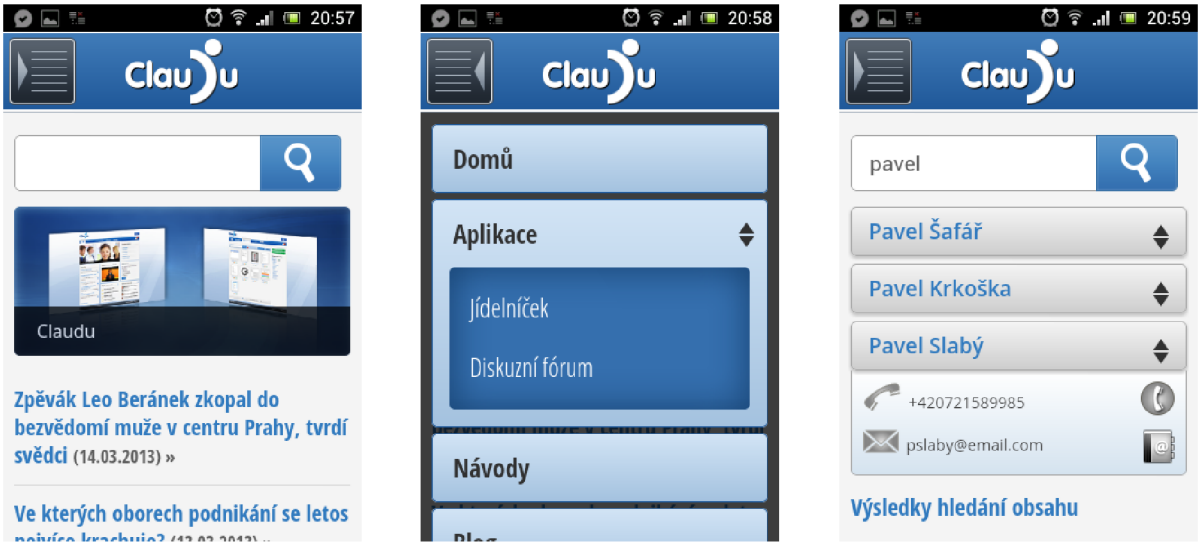


Poznámky

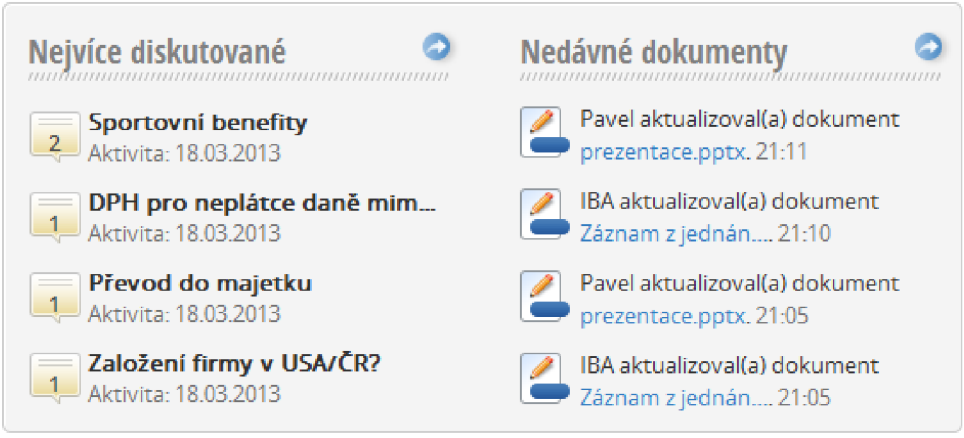


Jídelníček

Carousel



Mobilní verze



Nejvíce diskutované a Nedávné dokumenty