Masarykova univerzita
Fakulta informatiky

Sada testov pre aplikáciu Remsig

Bakalárska práca

Miroslav Mačor

Brno, jeseň 2016

Masarykova univerzita
Fakulta informatiky

Sada testov pre aplikáciu Remsig

Bakalárska práca

Miroslav Mačor

Brno, jeseň 2016

Namiesto tejto stránky vložte kópiu oficiálneho podpísaného zadania práce a prehlásenie autora školského diela.

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Miroslav Mačor

Vedúci práce: RNDr. Daniel Kouřil, Ph.D

Poďakovanie

Ďakujem vedúcemu práce RNDr. Danielovi Kouřilovi, Ph.D za ústretovú a odbornú pomoc pri tvorení bakalárskej práce a za čas venovaný riešeniu technických problémov. Ďakujem Silvii Vigašovej za netriviálnu pomoc pri úvodnom rozbiehaní aplikácie RemSig. Ďakujem mojej rodine za neustálu podporu.

Zhrnutie

Cieľom tejto bakalárskej práce je vytvoriť testovaciu sadu pre aplikáciu RemSig, ktorá má na starosti podpisovanie dokumentov a úschovu súkromných kľúčov na Masarykovej univerzite. Sada obsahuje jednotkové, integračné a výkonnostné testy napísané v Jave. V prvej časti práce je popísaná potreba pre RemSig a testy vo všeobecnosti. Druhá časť práce je venovaná praktickému prevedeniu testov. Podrobne je rozobraná štruktúra, použité technológie a výsledky testov.

Kľúčové slová

RemSig, jednotkové testy, integračné testy, výkonnostné testy

Obsah

1 Úvod

Vo vývojovom procese sú kvalitne napísané testy takmer tak dôležité ako skutočný kód aplikácie. Správne napísané testy umožňujú jednoduché lokalizovanie chyby, nájdenie neuvoľnených zdrojov alebo overenie funkčnosti kódu. Testy neslúžia iba na odhaľovanie chýb ale hrajú dôležitú rolu pri následnom vývoji a vytváraní nových funkcií a vlastností. Pri agilnom programovaní [1] testy riadia a poháňajú samotný vývoj.

Cieľom tejto bakalárskej práce je napísať testy pre aplikáciu RemSig [2], ktorá má na starosť spravovanie súkromných kľúčov, podpisovanie dokumentov a certifikátov, a v neposlednom rade skladovanie certifikátov pre potreby Masarykovej univerzity (MU). Aplikácia bola pre lepšiu škáľovateľnosť a údržbu prepísaná z jazyku PHP do jazyku Java. Na aplikáciu RemSig v jazyku Java neprebehlo žiadne systematickejšie testovanie a jediné testovanie a overenie základnej funkcionality je testovanie samotného vývojára.

Práca je rozdelená do štyroch kapitol. Druhá kapitola podrobnejšie rozoberá dôležité aspekty a princípy, na ktorých funguje RemSig. Vysvetľuje fungovanie a využitie digitálneho podpisu a taktiež obsahuje priblíženie bezpečnej SSL komunikácie s obojstrannou autentizáciou. Podrobnejšie je rozobraná potreba utajiť súkromný kľúč, možné dôsledky straty kľúča a spôsoby uschovávania súkromného kľúča na webovom serveri aby nedošlo k jeho kompromitácii. V závere kapitoly je v stručnosti rozobraná aplikácia RemSig, jej funkcie, zabezpečenie a vlastnosti.

Tretia kapitola popisuje testy vo všeobecnosti. Vysvetľuje potrebu zvolenia správnych testov a bližšie rozoberá jedno z možných rozdelení testov podľa vstupných údajov. V prvej podkapitole analyzuje dve protichodné taktiky pri tvorby testov. Nasleduje postupné rozoberanie troch typov testov: jednotkové, integračné a výkonnostné. Pri každom z testov je uvedená definícia konkrétneho typu a ich bežné použitie v praxi.

Štvrtá kapitola je sústredená na opis praktického prevedenie testov vytvorených pre RemSig. Pre každý typ testov je vytvorená podkapitola obsahujúca základný popis a sekcie: štruktúra testov a výsledky testov. V sekcií štruktúra testov je opísané konkrétne prevedenie a opis fungovania testov. Sekcia výsledky testov obsahuje výsledky testov z testovacími dátami. Pokiaľ to je potrebné sú v samostatnej sekcií definované použité technológie a dôvod ich použitia. Záver kapitoly slúži na krátke zhrnutie testov.

Testy vytvorené v tejto bakalárskej práci tvoria základný testovací balík pre aplikáciu RemSig, ktorý výrazne uľahčí ďalší vývoj a údržbu kódu. Všetky testy, vrátane podrobnej správy nájdených chýb a vygenerovaných vstupných dát som odovzdal vývojovému tímu aplikácie RemSig.

2 Remsig aspekty

V tejto kapitole sú popísané najdôležitejšie časti aplikácie Remsig, ktoré majú dopad na vytváranie testov. Tieto časti Remsig buď potrebuje alebo využíva. Následne je opísaná potreba pre uchovávanie súkromných kľúčov na webovom úložišti a v závere kapitoly je v stručnosti opísaná samotná aplikácia Remsig.

2.1 Digitálne podpisy

Digitálne podpisovanie je jednou z hlavných úloh aplikácie Remsig. Digitálny podpis je matematická technika na overenie autenticity (pôvod dát), integrity (neporušenosť dát) a nepopierateľnosť odoslania dát autorom [3]. Digitálne podpisy fungujú na princípe asymetrickej kryptografie, na šifrovanie sú použité dva rozdielne kľúče, súkromný a verejný. Na vytvorenie digitálneho podpisu je z dát vytvorený pomocou transformačnej funkcie hash (skrátené jedinečné dáta pre vstup), ktorý je zakódovaný súkromným kľúčom. Zakódovaný hash spolu s ďalšími informáciami ako je transformačná funkcia tvoria digitálny podpis. Príjemca na odšifrovanie správy použije verejný kľúč odosielateľa a z dát vytvorí vlastný hash, ktorý porovná s hashom zo správy. Digitálny podpis je špecifický a jedinečný ku konkrétnym údajom a konkrétnemu podpisovateľovi.

2.2 SSL/TLS zabezpečenie

SSL (z anglického Secure Socket Layer) bolo vyvinuté Americkou firmou Netcape Communications v roku 1990. TLS (z anglického Transport Layer Security) vzniklo ako náhrada SSL a bol založený na SSL verzií 3.0. SSL nie je v dnešnej dobe považovaný za bezpečný, a preto je odporúčané používať TLS protokol. I napriek náhrade SSL protokolu TLS protokolom sa v literatúre používa SSL pri odkazovaní na TLS a to hlavne z historických dôvodov. Ako kompromis je taktiež používané označenie SSL/TLS. V tejto práci sa pri použitý SSL odkazuje na TLS protokol.

SSL nie je súčasťou aplikácie RemSig, ale mal značný dopad na vývoj testov a spôsob ich prevedenie. Znalosť SSL je nápomocná ku pochopeniu kódu testov.

SSL je počítačový protokol umožňujúci autentizáciu klienta a servera. SSL taktiež poskytuje bezpečnú a šifrovanú komunikáciu. SSL je štandardný a široko rozšírený spôsob zaistenie bezpečnej komunikácie v dnešnej internetovej komunite. Kanál je transparentný, čo znamená, že dáta nie sú pozmenené počas prenosu. SSL využíva asymetrickú kryptografiu na ustanovenie spojenia. Samotná komunikácia servera a klienta je zriadená pomocou socketou a symetrickej kryptografie (jeden kľúč je použitý na šifrovanie aj dešifrovanie). Socket je koncová súčasť komunikácie cez počítačovú sieť [4]. Sockety sú tiež používané na komunikáciu rôznych procesov v jednom počítači.

SSL protokol obsahuje dva pod-protokoly na vzájomnú autentifikáciu servera a klienta. Record protokol a handshake protokol. Record protokol je použitý na zabalenie rôznych protokolov na vyššej úrovni vrátane handshake protokolu [5]. Správy z protokolu handshake sú vložené do čistého textu a prepravené podľa špecifikácie relácie.

Pri hadshake protokole sa využívajú X.509 certifikáty [6]. X.509 certifikát je digitálny certifikát využívajúci rozšírene akceptovateľnú X.509 Public Key Infrastructure (PKI), ktorá slúži na prepojenie verejného kľúča s identitou. X.509 certifikát obsahuje verziu X.509, sériové číslo, použitý algoritmus na podpísanie certifikátu, špecifické meno vydavateľa, dobu platnosti certifikátu, špecifické meno vlastníka a verejný kľúč vlastníka.

Handshake protokol je obzvlášť zraniteľný útokom typu Man in the middle. Pri tomto type útoku by útočník odchytil prvotnú požiadavku klienta o zabezpečené spojenie a poskytol klientovi svoj verejný kľúč. Zaistenie identity servera je praktizované pomocou Certifikačnej autority. Certifikačná autorita vydáva certifikáty a ručí za správnosť údajov v certifikáte [7]. Pokiaľ dôverujeme danej certifikačnej autorite, môžeme veriť aj jej podpísaným certifikátom a verejným kľúčom v danom certifikáte.

2.3 Zabezpečenie súkromného kľúča

Správne zabezpečenie súkromných kľúčov je najdôležitejšia úloha pri asymetrickej kryptografii rovnako ako servera, tak aj klienta. Celá asymetrická kryptografia je postavená na predpoklade existujúceho spôsobu nezistenia súkromného kľúča nežiadúcou osobou [8]. Pri prezradení kľúča použitom na šifrovanie SSL komunikácie je zraniteľná konkrétna inštancia komunikácie a to tiež len do jej uzavretia. Pri prezradení súkromného kľúča je kompromitovaná terajšia aj budúca komunikácia. Útočník, ktorému sa podarilo získať cudzí súkromný kľúč, je schopný čítať všetku komunikáciu medzi serverom a klientom, akoby vôbec nebola šifrovaná. Tiež má možnosť vydávať sa za identitu od ktorej získal privátny kľúč.

Vzhľadom na momentálny problém s prvočíselným rozkladom je takmer nemožné získať súkromný kľúč z ľahšie dostupného verejného kľúča. Zatiaľ čo boli prípady získania súkromného 768-bit kľúča [9], odhadovaný čas pri 1024-bit kľúči, by pri použití rovnakého algoritmu trval niekoľko tisíc rokov. Neznamená to absenciu rýchlejších a efektívnejších algoritmov, ale oveľa častejší problém straty alebo odcudzenia súkromného kľúča nastáva na strane koncového užívateľa.

Je očividné, že strata súkromného kľúča je závažný problém pre bezpečnú komunikáciu a súkromný kľúč by mal byť chránený. Ochránenie súkromného kľúča ale nie je také triviálne ako by sa mohlo zdať. Súkromné kľúče je možné zabezpečiť pomocou hardvéru alebo softvéru. Oba spôsoby majú svoje výhody a nevýhody, preto sa v bežnej praxi používa kompromis medzi hardvérovým a softvérovým riešením. Hardvérové riešenie je považované za bezpečnejšie [10]. Kľúč je náchylný útoku iba počas prenosu alebo s fyzickým prístupom ku zariadeniu na ktorom je kľúč uložený. Súkromný kľúč je možné uložiť na špecializovanom hardvéri ako je USB token alebo čipová karta, plastová karta so zabudovaným čipom, etc. Výhodou týchto zariadení je, že aj pri odcudzení zariadenia nieje zaručená strata kľúča, ktorý je na ňom uložený. Toto je zaistené buď na hardvérovej úrovni alebo vymazaním kľúča pri zistenom pokuse o neoprávnený prístup.

Uloženie kľúča na špeciálnom zariadení so sebou prináša hneď niekoľko nevýhod akou je škálovateľnosť, cenová nákladnosť a v neposlednom rade dostupnosť. Pri rozšírený systému o ďalších užívateľov je potrebné nakúpiť nové zariadenia, taktiež pri strate alebo odcudzeniu zariadenia so súkromným kľúčom. Dostupnostné problémy vznikajú z potreby užívateľa mať zariadenie vždy so sebou a z neschopnosti užívateľa sa autentizovať bez zariadenia.

Softvérové riešenie adresuje hlavné nevýhody hardvérového riešenia. Kľúče sú uložené na serveri a ľahko dostupné autentizovanému užívateľovi. Pri potrebe rozšíriť počet užívateľov sa jednoducho vygenerujú nové kľúče, ktoré sa následne distribujú užívateľom.

Úspech softvérového riešenia záleží na zabezpečení servera, na ktorom sú kľúče uložené. Server môže obsahovať od niekoľko desiatok až po niekoľko tisícov kľúčov, čím je výrazne viac atraktívny pre útočníkov ako zariadenie užívateľa, na ktorom môže byť jeden alebo dva kľúče. Na server je možné útočiť bez fyzického prístupu, čím je výrazne viac zraniteľný ako hardvérové riešenie.

Nech je zvolená ktorákoľvek možnosť, neodporúča sa uchovávať kľúče v ich prirodzenej, nezašifrovanej forme. Namiesto toho je bežnou praxou šifrovať súkromné kľúče pomocou užívateľom zvoleného hesla. Okrem pridanej bezpečnosti šifrovanie heslom tiež zaisťuje aby ani samotný úschovný systém nebol schopný čítať súkromné kľúče. Heslá sú stále náchylnejšie útokom typu brute-force (skúšanie každej kombinácie) a útokom, ktoré hádajú heslo, ale je to rozumná stredná cesta medzi nešifrovaním súkromných kľúčov a šifrovaním ich do takej miery kde by obmedzovali užívateľskú dostupnosť.

2.4 Remsig

Aplikácia RemSig [2] vznikla na Ústave výpočtovej techniky Masarykovej univerzity, ako riešenie podpisovania dokumentov (okrem

iného aj pre štátnu správu) a ukladania súkromných kľúčov. Zároveň bolo potrebné zmeniť pôvodné riešenia spravovania osobných digitálnych certifikátov na počítači, alebo hardvérových zariadeniach a umožniť vytváranie digitálnych podpisov na diaľku. RemSig umožňuje podpisovanie priamo z webového rozhrania systémom INET (Ekonomicko-správní informačný systém Masarykovej univerzity) a IS MU (informačný systém Masarykovej univerzity). Z INETu je možné certifikáty aj importovať.

RemSig poskytuje:

- systém pre bezpečné uloženie osobných digitálnych certifikátov vydávané ľubovoľnou certifikačnou autoritou
- systém pre vzdialené podpisovanie dát a dokumentov
- značkovanie dokumentov časovými razítkami
- podporu pre správu certifikátov (generovanie, revokovanie)

Bezpečnosť dát uložených v systéme RemSig je dosiahnutá nasledovnými opatreniami. RemSig zabezpečuje súkromné kľúče, užívateľom zvoleným heslom. Pri dešifrovaní súkromného kľúča je operácia dešifrovania vykonávaná striktne v operačnej pamäti počítača a dešifrovaný kľúč nikdy nie je uložený na disk. Všetka komunikácia s aplikáciou RemSig prebieha po úspešnej autentizácii cez SSL a dáta sú v šifrovanej podobe pravidelne zálohované. Všetky operácie sú prísne auditované, a je jednoduché zistiť, kto manipuloval s čím. Kvalifikované certifikáty sú vydané certifikačnou autoritou PostSignum, s ktorou má MU podpísanú dohodu.

Aplikácia RemSig bola pôvodne napísaná v jazyku PHP, ale kvôli väčšej možnosti rozšírenia a ľahšej udržovateľnosti [11] vznikla potreba prepísať túto aplikáciu do jazyka Java. Bližšie rozoberanie fungovania aplikácie RemSig, vrátane použitých technológii a praktických príkladov je pokryté v bakalárskej práci Silvie Vigašovej [12]. Z tohto dôvodu sa venujem samotnej aplikácií v rozsahu potrebnom pre túto bakalársku prácu.

3 Testy

Testy v informatike sú programy alebo časti programov navrhnuté na overenie funkcionality testovaného produktu. Toto je primárna úloha testov ale testy slúžia, okrem iného, aj na poskytnutie dokumentácie pre potreby programátora, overenie nezmenenej funkcionality pri refaktorácii **??**(zmena štruktúry kódu pre orientáciu), uľahčenie hľadania bugov (časti kódu kde sa aplikácia správa inak ako sa má), tým pádom skrátenie času potrebného na debugovanie, atď.

Typy testov sa enormne líšia v závislosti od veľkosti testovaného kódu, spôsobov testovania a očakávaných výsledkov. Pri obrovskom množstve typov testov a spôsobov testovania je úlohou vývojového tímu zvoliť správnu cestu pre maximálne ošetrenie aplikácie, a zároveň šetrenia tímových prostriedkov ako je napríklad čas [8]. V tejto kapitole sú rozobrané len niektoré testovacie procesy a typy testov, ktoré prevažne súvisia s praktickou časťou.

Jedno z takýchto rozdelení je testovanie, či sa aplikácia správa ako sa očakáva, a či sa nespráva neočakávane. Pri testovaní očakávaného správania sa pozoruje chovanie aplikácie so správnymi a s nesprávnymi parametrami. Na druhú stranu pri testovaní neočakávaného správania je snaha dosiahnuť nedefinované správanie testovaného kódu, alebo aplikácie. Testujú sa hraničné hodnoty a hodnoty za hranicami. Ako parametre sa udávajú úplne nezmyselné hodnoty, znaky v nezvyčajnom kódovaní, prázdne polia alebo systémové príkazy. Pri systéme s užívateľskou autentizáciou pomocou mena a hesla, môže byť ako prihlasovacie meno alebo heslo vložený príkaz manipulujúci s databázou (DROP DATABASE), ktorý vymaže údaje z databázy.

Ako príklad uvediem triviálnu aplikácia na súčet. Pri testovaní predpokladaného správania sa otestuje či 2 + 2 je 4, a či 5 + 2 nie je 4. Pri kontrole nepredpokladaného správania sa testuje či 2 + a, maximálna hodnota typu Integer + 2 alebo и + 5 nespôsobí pád aplikácie ale tiež či aplikácia nevráti hodnoty, ktoré považuje za korektné.

3.1 Tradičné testovanie alebo Vývoj riadenými testami

Pri problematike tvorby testov existuje niekoľko protikladných názorov, pričom najväčší rozpor je ideálny čas písania testov. Tu sa stretávajú dve protichodné myšlienky. Tradičné testovanie a Vývoj riadený testami (z anglického Test-driven development (TDD))[13].

V tradičnom testovaní sú všetky testy písané až po dokončení kódu. Tento postup prináša výhody v podobe rýchleho dokončenia tvorby požadovanej aplikácie a menšej potreby upravovania kódu, z dôvodu upresnenia alebo pozmenenia zadania. Samotné testy ale potenciónálne strácajú na kvalite, z nevyhnutnej potreby špecializácie testov na konkrétny kód. Ako je aj závislosť na skutočných dátach, čo navádza k neodhaleným problémom a neskorším odchýlkam v nasledujúcich implementáciách. Pokrytie aplikácie testami je ďalšia nevýhoda tradičného testovania.

Ako odpoveď na tieto problémy vzniklo čiastočne z extrémneho programovania TDD. Extrémne programovanie je špecifické neustálou adopciou na meniace sa požiadavky zákazníka. Medzi identifikujúce znaky extrémneho programovania patrí neustále testovanie, programovanie v pároch (dvaja programátori za jedným počítačom), alebo implementovanie iba tých nevyhnutných funkcií pre realizovanie kódu.

TDD sa zakladá na malých, stále sa opakujúcich krokoch alebo fázach vývoju. Fáza písaniu testov, fáza písania kódu a fáza refaktorovania. V prvom kroku (prvej fáze) sa napíše test na zatiaľ neexistujúci kód a overí sa, či test zlyhá. Nasleduje vytvorenie kódu a spustenie testov. Posledný krok je refaktorovanie. Refaktorovanie je proces úprav a zmeny kódu za účelom zlepšenia priehľadnosti a udržovateľnosti kódu, pričom sa ale nemení funkcionalita kódu. V tomto kroku sa venuje efektivite, čitateľnosti a udržovateľnosti kódu. Kód je presunutý

na miesto, kde v projekte logicky patrí a sú odstránené duplicity kódu. Pri refaktorovaní, testy slúžia na overenie nezmenenej funkcionality. Celý proces sa opakuje napísaním testov pre ďalšiu časť požadovanej aplikácie. Tento proces sa riadi tromi zákonmi alebo pravidlami TDD [14].

- Prvý zákon: Nesmieš napísať produkčný kód, kým nie je napísaný zlyhávajúci test.
- Druhý zákon: Nesmieš napísať viac kódu v jednotkovom teste ako je potrebné pre jeho zlyhanie.
- Tretí zákon: Nesmieš napísať viac produkčného kódu ako je potrebné na prejdenie testu.

Nie je ale možné pre každú časť kódu aplikácie vytvoriť testy či sa jedná o tradičné testovanie alebo TDD. Z tohto dôvodu tieto pravidlá nie sú absolútne ale vo všeobecnosti sa programátori píšuc kód podľa TDD snažia nasledovať tri zákony TDD do maximálnej možnej miery. TDD má ešte jednu výhodu oproti tradičnému testovaniu a to, testovateľnosť kódu. Zatiaľ čo je možné v tradičnom testovaní písať kód aby bol testovateľný v TDD je to nevyhnutné.

3.2 Jednotkové testy

Jednotkové (z anglického unit) testy sú špecializované na najmenšie jednotky projektov a aplikácii. Zatiaľ čo s týmto súhlasí väčšina vývojárov a expertov, definícia najmenšej jednotky má oveľa rozsiahlejšie spektrum pochopení [15][16]. Taktiež je veľa nezhôd do akej miery by mali byť jednotkové testy izolované od zvyšných častí kódu a ostatných zdrojov. Od jednotkových testov sa takisto očakáva výrazne vyššia rýchlosť ako od iných typov testov. Jednotkové testy sú obvykle spúšťané viackrát za deň a preto je veľká rýchlosť nevyhnutná. Ideálne časové rozhranie jednotkových testov záleží od vývojára. Martin Fowler poukazuje na to, že čo môže stačiť za rýchlosť jednému vývojárovi sa môže zdať neuveriteľne pomalé inému vývojárovi [17].

3.3 Integračné testy

Integračné testy sú vykonávané po vytvorení jednotkových testov a slúžia na overenie funkcionality väčších celkov a prepojenie menších jednotiek do komplexnejších systémov [18]. Cieľ integračných testov je preveriť funkčnosť a spoľahlivosť so simulovanými údajmi. Integračné testy sú zamerané na testovanie fungovania niekoľkých prepojených jednotiek alebo systémov.

Môžu byť použité na otestovanie dvoch jednotiek predchádzajúc testované jednotkovými testami alebo na kontrolu komunikácie aplikácie s databázou a následnú komunikáciu s webovým serverom. Integračné testy sú obvykle pomalšie a je náročnejšie lokalizovať bod zlyhania ako v jednotkových testoch. Preto sa odporúča ich testovať iba na kóde pokrytom jednotkovými testami.

3.4 Výkonnostné testy

Výkonnostné testovanie je zamerané na určenie priepustnosti, spoľahlivosti, schopnosti reagovať a škálovateľnosti systému pod určitou záťažou [5][19]. Výkonnostné testovanie je možné rozdeliť na výkonnostné testy, záťažové testy a stresové testy.

Výkonnostné testy slúžia na zistenie rýchlosti a škálovateľnosti systému pod záťažou. Záťažové testy sú spúšťané na overenie fungovanie aplikácie v predpokladaných podmienkach. Úloha stresových testov je overiť, pod akou maximálnou záťažou je aplikácia stále schopná normálneho behu [16]. Záťažové testy by tiež mali identifikovať bod, v ktorom aplikácia už nie je schopná ďalej fungovať.

Výkonnostné testovanie slúži na odhadnutie pripravenosti produktu, odhalenia zdroju problému s výkonnom alebo identifikovanie miesta, kde jeden zdroj spomaľuje výkon celej aplikácie tiež známe ako bottleneck. Pri úspešnom odstránení tohto miesta sa zvyšuje výkon celej aplikácie.

4 RemSig testy

Cieľom práce bolo zabezpečiť RemSig aplikáciu jednotkovými, integračnými a výkonnostnými testami. K týmto testom bol pridaný aj test nezmenenej funkcionality s predchádzajúcou implementáciou a zaistenie plynulého prechodu na novú verziu. Všetky testy sa snažia o maximálnu izolovanosť od ostatných kategórii testov (napr. jednotkové a výkonnostné testy).

Na spravovanie často využívaných metód je vytvorená trieda TestManager. TestManager obsahuje metódy, ktoré nesúvisia priamo z aplikáciou RemSig. Jedná sa o metódy typu načítavanie súborov z disku, inicializovanie databázy, získanie dát z databázy, spracovávanie XML dokumentov (načítavanie elementov a atribútov), etc. TestManager taktiež obsahuje niektoré porovnávacie metódy ktoré využíva viacero samostatných jednotkových testov. Každá kategória testov do istej miery využíva triedu TestManager.

V tejto kapitole je bližšie rozobraná štruktúra jednotlivých testov, ich doteraz dosiahnuté výsledky a použité technológie.

4.1 Jednotkové testy

Testy sú zamerané na kontrolovanie troch hlavných prípadov. Očakávané správanie aplikácie so správnymi parametrami, s nesprávnymi parametrami a s null (prázdnymi, nenaplnenými) parametrami.

4.1.1 Použité technológie

RemSig jednotkové testy sú praktizované technológiou JUnit [20]. Na využívanie a spravovanie databázy je použitý jazyk MySQL [21] v spolupráci s DbUnit [22]

JUnit – je jednoduché open source rozhranie slúžiace na písanie a spúšťanie opakovateľných testov. JUnit porovnáva návratové hodnoty metód s preddefinovanými hodnotami. Podľa prieskumu v roku 2013 z 10 000 projektov na serveri github, ktorý slúži na zdieľanie zdrojových kódov, je JUnit, spolu s knižnicou slf4j-api, najpoužívanejšou knižnicou obsiahnutom v 30.7% projektov [23].

MySQL – je open source databázový systém. Je kompatibilný so všetkými často používanými jazykmi a podporovaný najčastejšie používanými operačnými systémami.

DbUnit - je nadstavba JUnitu zameraná prevažne na nastavenie databázy do predom známeho stavu. Toto je obzvlášť užitočné pri jednotkových testoch, kde jeden test môže pozmeniť údaje v databáze a viesť k nepresnému výsledku ostatných testov. DbUnit inicializuje databázu pomocou dátovej sady uloženej XML formáte a pri inicializácii overuje typovú správnosť vložených údajov. RemSig jednotkové testy inicializujú údaje v databáze pred spustením každého testu a mažú databázu po skončení testu.

4.1.2 Štruktúra testov

Pri tvorbe testov bolo sústredenie na kvalitné ošetrenie hlavných častí aplikácie dôležitejšie ako 100% pokrytie. Následkom toho, každá podstatná trieda má vytvorenú korešpondujúcu testovaciu triedu. Na správu obecných metód, aplikácia používa triedu TestManager.

Testované boli triedy CertificateManger, Signer, ktoré zaisťujú hlavné funkcie aplikácie RemSig a XmlParser. XmlParser je pomocná trieda, ktorá spracováva vstupné a výstupné parametere a pracuje s XML dokumentami. Ostatné verejné triedy buď neobsahovali verejné metódy alebo nemali potrebu testovať verejné metódy. Príklad metód, ktoré nebolo nutné testovať:

- metódy umožňujúce prístup k súkromným parametrom triedy
- metódy na porovnávanie inštancií tried
- metódy výpočet hashu tried

Testované sú všetky verejné metódy tried. Testovacia trieda na XmlParser výnimočne testuje aj prístup k parametrom triedy a to z dôvodu, že tieto metódy sú používané aj v ostatných testoch. Pre každú konkrétnu metódu je vytvorený jeden test.

Testy vo veľkej miere využívajú databázu a nepoužívajú mockovanie databázy [24]. Mockovanie je spôsob simulovania zdrojov a objektov, aby imitovali správanie skutočných zdrojov. Mockovanie databázy prináša výhody v zvýšení rýchlosti testov a nezávislosť od konkrétnej inštancie databázy.

Mockovanie databázy v jednotkových testoch nieje praktizované z nasledujúcich dôvodov. Volania databázy sú realizované aplikáciou RemSig priamo v kóde pomocou SQL príkazov, čo vytvára nemožnosť vytvorenie abstrakcie nezávislej databázy bez refaktorovania pôvodných metód. Tu sa stráca jedna z hlavných výhod mockovania, abstrakcia zdrojov.

Zatiaľ čo je možné vytvoriť mockovanie na konkrétnu inštanciu príkazov, prevedenie je značne komplikovanejšie a vytvorené testy strácajú na integrite. Ďalšou výhodou je zrýchlenie testov, ale zrýchlenie sa nepovažovalo za dostatočné, aby odôvodnilo čas testera potrebný na prevedenie mocku databázy.

Databáza v dopredu známom stave patrí tiež medzi výhody mockovania. RemSig jednotkové testy ako náhradu používajú DbUnit. TestManager obsahuje metódu na prevedenie údajov z databázy do XML, vďaka čomu môžu testovacie údaje byť udržiavané takmer automaticky. DbUnit maže údaje po každom prevedenom teste, a preto sa výrazne odporúča používať na testovanie inú inštanciu databázy ako produkčnú.

Každý test sa skladá s nasledujúcich častí. V úvode si test inicializuje potrebné lokálne premenné. Pre spoločné alebo často používané premenné sú používané globálne premenné.

Test následne overí správanie metódy so všetkými kombináciami prázdnych a nulových hodnôt. Pri testovaní zachytáva nullPointerException a v prípade nezachytenia programom test zlyhá.

V ďalšej časti test overí správanie metódy pri nesprávnych vstupných parametroch.

Pri autentizácií sa jedná napr. o nesprávnu kombináciu prihlasovacieho mena a hesla alebo nesprávnu kombináciu hesla a názvu

certifikátu. Pri týchto testoch sa kontroluje aj databáza či nebola pozmenená nesprávnymi dátami.

V poslednom kroku sa kontroluje správanie aplikácie s korektnými dátami. Návratové hodnoty sa kontrolujú s predom definovanými hodnotami. Tiež sa kontroluje jednotnosť dát vrátenými metódami a dátami uloženými v databáze. V niektorých testoch sa tiež overuje jednotnosť vygenerovaných dát v rôznych časoch. Pri podpisovaní dokumentov sa e.g. kontroluje rozdielnosť podpisov pri pozmení pôvodných dát rovnako ako konzistentnosť dát v návratovom XML a databáze.

4.1.3 Výsledky testov

Doterajšie jednotkové testovanie odhalilo niekoľko menších nedostatkov ako nesprávnu konfiguráciu alebo nezachytenie nulovej výnimky. Medzi väčšie, beh ohrozujúce nájdené problémy patrili nefunkčné funkcie, metódy zlyhávajúce pri korektných parametroch alebo nesprávne sa správajúce metódy. Nasledujúce metódy nekontrolovali nulové výnimky:

- generateRequest()
- changeCertificateStatus()
- CheckPassword()
- changePassword()
- resetPassword()
- changeCertificateStatus()
- listCertificateWithStatus()
- uploadPrivateKey()
- sign()
- signPdf()
- signPkcs7()

Medzi vážnejšie nedostatky patrili: metóda UploadPrivatekey() zlyhávala v jednej kombinácii parametrov v ktorej mala fungovať a nebola schopná zmeniť kľúč na nové dáta. Tiež obsahovala syntatickú chyby v SQL príkaze. Pri metóde resetPassword() bolo staré heslo stále funkčné. Metóda signPkcs7 nerozoznávala voľbu algoritmu „SHA-1“ a zlyhávala z dôvodu bielych znakov (medzery a nové riadky) v konfiguračnom súbore profiles.xml.

4.2 Overenie funkcionality

Pri zmene nasadenie aplikácie RemSig z PHP na Javu je potrebné zaistiť kontrolu nezmenenej funkcionality medzi správaním pôvodnej a aktuálnej verzie. Rovnako je potrebné zachovať nezmenenú podobu komunikácie zo strany klienta. Tento problém je riešený pomocou serverového Forku. Fork je označenie pre kód, ktorý príjíma vstupné dáta z jedného zdroju a ako výstup posiela dáta do viacerých zdrojov. Test na nezmenenú funkcionalitu má význam nie len pri testovaní zmeny servera z PHP na Javu, ale tiež bude uľahčovať bezproblémové nasadanie nových verzií. Tento test je pomenovaný podľa jeho realizácie a teda Fork.

4.2.1 Použité technológie

Fork je realizovaný pomocou servletu. Fork môže byť vložený na RemSig server ale tiež môže byť nasadený na samotný server.

Servlet je program bežiaci na webovom servery, napísaný v Jave [25]. Servlet pomáha rozšíriť funkcionalitu, údržbu a podporu pre server. Servlety, ktoré slúžia ako sprostredkovateľ medzi klientom a serverom, umožňujú webovým vývojárom vytvorenie dynamických web-stránok (stránka schopná interakcie s klientom) [26], ako aj spracovávanie dát od užívateľov.

4.2.2 Štruktúra testov

Pri opise fungovania testu Fork je potrebné rozlišovať medzi pôvodným klientom, RemSig serverom v jazyku PHP (ďalej len PHP server) a RemSig serverom v jazyku Java (ďalej len Java server). RemSig server

sa používa v prípade, pokiaľ nezáleží na jazyku servera alebo sa to týka oboch serverov.

Komunikácia začne klientom posielajúc HTTP post požiadavku na web adresu RemSig servera. Fork (neviditeľný klientovi) preberá serverovú rolu a pokúša sa nadviazať SSL spojenie. Vzhľadom na potrebu SSL overenia Fork obsahuje vlastný keystore, kolekcia certifikátov, a vlastnú kópiu RemSig truststoru, kolekcia certifikátov certifikačných autorít. Po úspešnej autorizácii klienta certifikátom si Fork uchová všetky údaje potrebné na opakovanie originálnej HTTP post požiadavky a uloží certifikát klienta do vlastného keystoru, pre potrebu autentizácie s RemSig serverom.

V ďalšom kroku Fork nadviaže SSL spojenie s PHP serverom a Java serverom. Po úspešnom nadviazaní bezpečného spojenia Fork preposiela post požiadavky od klienta PHP serveru. Odpoveď PHP servera, je uložená na disk a pomocou Forku preposlaná klientovi. Následne Fork posiela rovnaký post požiadavok Java serveru a jeho odpoveď sa taktiež ukladaná na disk. V poslednom kroku sa na disk ukladá aj pôvodný post požiadaviek pre prípadné spätné overenie správnosti vygenerovaných dát. Odpovede PHP a Java serveru sú uložené do rozdielnych priečinkov. Prepojiteľnosť a unikátnosť súborov je docielená špecifickými názvami súborov tvorenými súčasným časom a Java funkciou uniqueId.

4.2.3 Výsledky testov

Výsledky testu Fork nie sú v dobe tvorby bakalárskej práce známe. Overenie funkcionality Fork prebehlo na lokálnom stroji a to iba v dvoch Java verzií. Výsledky takéhoto testovania neboli ničím zaujímavé, a preto nie sú v práci uvedené. Testovanie verzií PHP a Java serveru neprebehlo z niekoľkých dôvodov. Tým hlavným boli odhalené chyby Java serveru, ktoré by viedli iba k parciálnym výsledkom.

4.3 Integračné testy

Integračné testy overujú funkčnosť RemSig metód cez aktívny server. Rovnako ako pri jednotkových testoch, testujú správanie pri null, nesprávnych a správnych parametrov. Na rozdiel od jednotkových testov, ktoré testujú RemSig s minimálnym využitím zdrojov a volanie metód nie je praktizované cez server, integračné testy testujú priamo správanie servera s využitým všetkých zdrojov. Server je testovaný ako v reálnom prostredí. Integračné testy sú schopné načítavať post požiadavky priamo z priečinkov. Táto vlastnosť bola implementovaná pre možné nadviazanie na Fork.

4.3.1 Štruktúra testov

Integračné testy sú navrhnuté tak, aby mohli slúžiť ako samostatne fungujúci program, pričom testujú správanie servera z klientskej strany. Konkrétne simulujú správanie klienta a zaznamenávajú odpoveď servera. Integračné testy obsahujú niekoľko verejných metód a hlavnú metódu na spustenie testu. Verejné metódy sa delia na komunikáciu so serverom a spracovávanie priečinkov. Metódy su schopné načítať všetky súbory z priečinku a podpriečinku, rozoznať požiadavku, nadviazať spojenie, odoslať konkrétnu požiadavku a vyhodnotiť odpoveď servera.

Pri integračnom testovaní sa vytvorili štyri podpriečinky v testovacom priečinku pomenované null, incorrect (nesprávne), correct (správne), correctSequence (správne v poradí) a piaty priečinok s názvom output na ochovávanie logu z testov. V každom zo štyroch priečinkov sú uložené korešpondujúce testovacie dáta uložené v XML formáte. Test prebieha nasledovne:

1. Postupne sú načítané všetky súbory z priečinkov null, incorrect a correct a všetky súbory v ich podpriečinkoch (zložka correctSequence je podpriečinok priečinku correct). Rozdielne priečinky nie sú nevyhnutné pre správne fungovanie testu ale dopĺňajú jeho funkcionalitu.

2. V každom súbore sa vyhľadá konkrétny názov metódy Remsig. Test očakáva súbory v XML formáte s názvom metódy v súbore.

- Pokiaľ sa názov metódy v súbore nenájde, prejde sa na ďalší súbor.
- Ak sa metóda nájde postupuje sa krokom 3.

3. Test nadviaže spojenie so serverom.
4. Test odošle požiadavku na server na korešpondujúcu metódu.
5. Pokiaľ je odpoveď servera chybová hláška tak je hláška, názov súboru ktorý ho spôsobil a názov metódy, zaznamenaná rovnako do súboru a vypísaná na výstup konzoly. Pokiaľ nepríde žiadna odpoveď zo servera na danú požiadavku, tak táto skutočnosť je rovnako zaznamenaná.

Rozdielne priečinky určujú správanie testu, kým test zaznamenáva chyby a žiadne odpovede, chyby nie sú zaznamenávané pokiaľ boli spôsobené súborom v priečinku null alebo incorrect.

Výhodou tohto prevedenia testu je, že aj keď samotný test je úplne separátny od testu Fork, tak s Forkom veľmi dobre spolupracuje. Fork zachytáva a ukladá požiadavky od užívateľov, čo vytvára príležitosť pre vytvorenie zmysluplných a hlavne relevantných testovacích dát. Stále platí všeobecné známe pravidlo, že užívateľ je najlepší tester. Ľahké pridanie užívateľských dát do testovacích taktiež napomáha k obmedzeniu opakovaniu chýb s konkrétnymi vstupnými dátami.

4.3.2 Výsledky testov

Testy odhalili niektoré chýbajúce chybové hlášky, v dvoch prípadoch server nevracal žiadnu odpoveď a podarilo sa odhaliť chybné metódy, ktoré prešli pôvodným jednotkovým testovaním. Toto vytvorilo potrebu skontrolovať pôvodné testy a bližšie analyzovať vzniknutú chybu. Ukázalo sa, že toto bolo spôsobené nesprávnym zachytávaním výnimiek na strane servera. Pri volaní metód signPdf a exportPkcs12 s neexistujúcou kombináciou užívateľského a certifikačného ID, server nevracal žiadnu odpoveď. Metóda exportPkcs12 čiastočne obchádzala zabezpečenie heslom. Pri nesprávnom hesle k pkcs12 metóda exportovala pkcs12. Toto sa ale nedialo pokiaľ heslo užívateľa nebolo správne.

4.4 Výkonnostné testy

Výkonnostné testy sú zamerané na rýchlosť spracovania požiadaviek serverom a na využitie procesoru a pamäti počas zátaže. Pred každým meraním bolo spustených niekoľko stoviek post požiadavok s rovnakými dátami ako boli následne testované. Táto predpríprava prostredia je dôležitá kvôli Java prekladači, ktorý zrýchľuje často sa opakujúci kód, čo by mohlo viesť k nepresným výsledkom. Následné merania boli testované pri sto a následne tisíc opakovaní.

4.4.1 Použité technológie

Pre minimalizovanie nepresností spôsobené vnútorným procesom Java prekladača a vnútorných procesov operačného systému je na meranie časových rozdielov medzi začatím prvej požiadavky a skončením poslednej požiadavky použitá funkcia Javy nanotime [27]. Metóda nanotime je špecializovaná na stopovanie určitého časového úseku a nie je možné jej použitie na žiadnu inú časovú referenciu. Návratová hodnota je počet nano sekúnd od fixného ľubovoľného času (môže byť aj v budúcnosti).

Spotreba pamäti je meraná ako rozdiel dostupnej a voľnej pamäti. Na získanie týchto údajov je použítá funkcionalita Java triedy Runtime [28]. Runtime trieda umožňuje Java aplikácii prístup k prostrediu na ktorom je aplikácia spustená. Celková spotreba pamäti aplikáciou je rozdiel medzi spotrebou pred a po meraní.

Využitie Procesoru je merané ako rozdiel využitia procesoru systémom pred a po meraní. Prístup k týmto údajom sa získava pomocou rozhrania OperatingSystemMXBean [29]. Toto rozhranie pristupuje k systémovým vlastnostiam operačného systému, na ktorom je spustená Java aplikácia.

4.4.2 Štruktúra testov

Výkonnostný test je, podobne ako integračný test, navrhnutý ako samostatne fungujúca aplikácia. Obsahuje metódy na komunikáciu so serverom a preťaženú metódu na prevedenie merania. Metódy sú

navrhnuté aby podporovali prípadne rozšírenie testov o hľadania maximálnej zátaže, ktorú je server schopný zvládať. Test príjíma vstupné dáta ako konkrétne súbory narozdiel od priečinkov súborov ako to je v integračných testoch. Konkrétne súbory zaručujú konzistentosť v nameraných výsledkoch a zaistenie, že zmena výsledkov (oneskorenie alebo zrýchlenie) je spôsobená zmenami aplikácie a nie neúmyselným upravením vstupných dát.

Výkonnostné testy sú schopné testovať časovú odozvu a výkon pre vybraný počet iterácii na jednej metóde, ale tiež na sekvencii daných metód. Meranie prebieha spoločnou metódou runTest(), do ktorej je možné vložiť ako parametere konkrétnu metódu a dáta alebo list zoradených dát a metód. Pri výbere počtu iterácií je možné zvoliť cyklus prevedenia opakovania a to buď for alebo while. V kóde je volanie nanotime funkcie vložené tesne pred prvým a tesne za posledným volaním meranej metódy. Metódy na meranie procesoru a pamäti sú vložené pred a za metódou runTest(). Výsledné hodnoty sú vypísané užívateľovi a uložené do súboru.

4.4.3 Výsledky testov

Namerané hodnoty z testovania sú pre prehľadnosť uložené v tabuľke. Tabuľka obsahuje sto a tisíc meraní pri meraní na lokálnom serveri. Metódy testované osobitne s nezmenenými vstupnými parametrami sú reprezentované v tabuľke menom volanej metódy. U niektorých metód je potrebné vytvoriť sekvenciu rôznych dát aby sa mohlo predísť skresleniu výsledkov a zachovať tabuľku v konzistentnom stave.

Tieto metódy reprezentuje názov „sekvencia“ a očíslovanie 1 až 3. Sekvencie s popisom zmien údajov a po užitých metód sú nasledovné. Sekvencia 1 – ImportPKCS12, vstupné údaje sa líšia v ID, očíslované od jedna po tisíc. Sekvencia 2 – changePassword(), dva rozdielne vstupné parametre, prvý parameter zmení heslo a druhý parameter vráti heslo na pôvodné. Sekvencia 3 – changeStatus(), dva rozdielne vstupné parametre, prvý zmení status a druhý ho mení na pôvodný status. Pri sekvenciách, kde sú použité viaceré metódy je počet iterácii adekvátne upravený pre zachovanie nemennosti tabuľky.

názov metódy	sto opakovaní (s) / CPU (%) / RAM(MB)	tisíc opakovaní (s) / CPU
generateRequest	85 / 30 / 10	868
exportPKCS12	12 / 45 / 12	95
sign	28 / 35 / 14	182
listCertitficate	18 / 39 / 12	95
listAllCertificates	17 / 31 / 13	96
checkPassword	10 /41 / 16	100
Sekvencia 1	17 / 50 / 13	132
Sekvencia 2	15 / 49 / 13	98
Sekvencia 3	11 /42 / 12	104

Tabuľka 4.1: Ukážka nameraných hodnôt

Z tabuľky je vidieť, že väčšina metód spotrebúva približne rovnaký pomer zdrojov až na metódu GenerateRequest(). GenerateReques() je výrazne pomalšia ako ostatné metódy, pretože výpočet prvotnej požiadavky na certifikát je náročná jednovláknová operácia. Aj keď celkový výkon operačného systému sa pohybuje okolo 30 % vlákno, ktoré výpočet vykonávalo pracovalo na 100 %

5 Záver

Cieľom bakalárskej práce bolo vytvoriť jednotkové, integračné a výkonnostné testy pre aplikáciu RemSig, ktorá bola implementovaná v jazyku Java. Pri testovaní boli postupne odhalené problémy rôznej závažnosti, ktoré boli posunuté vývojovému tímu.

Pri vývoji som odhalil niekoľko nefunkčne implementovaných metód a nezachytených výnimiek. Pri nefunkčných metódach som testy napísal podobne ako pri funkčných metódach, kde bol formát očakávaného výsledku vyčítaný zo zdrojového kódu aplikácie. Na správu obecných alebo často používaných metód bola vytvorená trieda TestManager. Túto triedu každý test do istej miery využíva.

Jednotkové testy pokrývajú všetky významné metódy. Nie sú pokryté metódy typu jednoduché nastavenie privátnej premennej. Integračné testy testujú volania metód priamo na serveri. Oproti jednotkovým testom, ktoré testujú výhradne metódy bez zasadenia do širších technológii akou jej server. Výkonnostné testy merajú rýchlosť spracovanie požiadaviek serverom.

Všetky požadované testy zo zadania boli napísané a pripravené na nasadenie. Navyše bol napísaný aj Serverový fork na minimalizovanie problémov s nasadením, ktorý s určitým časom sledovania odpovedí starého aj nového servera vie s celkom veľkou pravdepodobnosťou odporučiť novú implementáciu k ostrému prevozu. Všetky testy sú implementované aby s minimálnym úsilím mohli byť upravené na budúce meniace sa požiadavky, pričom ale nestrácali nič na svojej relativite.

Cieľ bakalárskej práce bol splnený. Vytvorené testy odhalili, že testovaná aplikácia RemSig zatiaľ nie je pripravená do ostrého nasadenia. Pred nasadením je potrebné odstrániť všetky nájdené chyby a nedostatky. Vytvorený testovací balík uľahčí lokalizovanie momentálnych aj prípadných budúcich chýb a pridá k stabilite aplikácie Remsig.

Bibliografia

[1] WATERS, Kelly. *What Is Agile? (10 Key Principles of Agile)*. [online] Dostupné z: <http://www.allaboutagile.com/what-is-agile-10-key-principles/>.

[2] *Úložiště RemSig MU*. Dostupné z: <http://www.ics.muni.cz/cs/katalog-sluzeb/podpora-spravy-ict/vydavani-certifikatu/uloziste-RemSig-mu>.

[3] ROUSE, Margaret. *digital signature*. [online] Dostupné z: <http://searchsecurity.techtarget.com/definition/digital-signature>.

[4] *network socket*. [online] Dostupné z: <http://whatis.techtarget.com/definition/sockets>.

[5] FREIER, et al. *The Secure Sockets Layer (SSL) Protocol Version 3.0*. [online] Dostupné z: <https://tools.ietf.org/html/rfc6101>.

[6] ROUSE, Margaret. *X.509 certificate*. [online] Dostupné z: <http://searchsecurity.techtarget.com/definition/X509-certificate>.

[7] ROUSE, Margaret. *certificate authority (CA)*. [online] Dostupné z: <http://searchsecurity.techtarget.com/definition/certificate-authority>.

[8] RESCORLA, Eric. *SSL and TLS: designing and building secure systems*. Boston: Addison-Wesley, c2001. ISBN 0201615983. s. 44-100, 140-173.

[9] et al. (2010-02-18) KLEINJING T. *actorization of a 768-bit RSA modulus*. [PDF] International Association for Cryptologic Research. Dostupné z: <https://eprint.iacr.org/2010/006.pdf>.

[10] SELTZER, Larry. *Securing Your Private Keys as Best Practice for Code Signing Certificates*. [PDF] Dostupné z: <https://www.thawte.com/code-signing/whitepaper/best-practices-for-code-signing-certificates.pdf>.

[11] LANGE, Fabian. *Comparison of Java and PHP for Web Applications*. [online] Dostupné z: <https://blog.codecentric.de/en/2008/07/comparison-of-java-and-php-for-web-applications/>.

[12] VIGAŠOVÁ, Silvia. *Client tools for RemSig*. [online] Dostupné z: <http://is.muni.cz/th/409781/fi_b/vigasova_bp.pdf>.

[13] *Software testing*. [online] Dostupné z: <https://en.wikipedia.org/wiki/Software_testing>.

[14] MARTIN, Rober C. *Clean Code: A Handbook of Agile Software Craftsmanship*. [PDF] Dostupné z: <http://ptgmedia.pearsoncmg.com/images/9780132350884/samplepages/9780132350884.pdf>.

[15] FEATHERS, Michael C. *Údržba kódu převzatých programů*. Brno: Computer Press, 2009. ISBN 978-80-251-2127-6. s. 24.

[16] PERRY, William E. *Effective methods for software testing*. New York: Wiley, c1995. ISBN 0471060976. 3 - 179.

[17] FOWLER, Martin. *UnitTest*. [online] Dostupné z: <http://martinfowler.com/bliki/UnitTest.html>.

[18] *What are Unit Testing, Integration Testing and Functional Testing?* [online] Dostupné z: <http://codeutopia.net/blog/2015/04/11/what-are-unit-testing-integration-testing-and-functional-testing/>.

[19] MEIER, J. D. *Performance testing guidance for web applications: patterns & practices*. United States?: Microsoft, c2007. ISBN 0735625700.

[20] *Frequently asked questions*. [online] Dostupné z: <http://junit.org/junit4/faq.html#overview_1>.

[21] SUEHRING, Steve. *MySQL Bible*. (PDF) Wiley Publishing, Inc. Dostupné z: <http://www.chettinadtech.ac.in/g_article/Textbook2%20%20MySQL%20Bible.pdf>.

[22] *About*. [online] Dostupné z: <http://dbunit.sourceforge.net/index.html>.

[23] WEISS, Tall. *We Analyzed 30,000 GitHub Projects – Here Are The Top 100 Libraries in Java, JS and Ruby*. [online] Dostupné z: <http://blog.takipi.com/we-analyzed-30000-github-projects-here-are-the-top-100-libraries-in-java-js-and-ruby/>.

[24] *Unit Testing with Mock Objects*. [PDF] Dostupné z: <https://msdn.microsoft.com/en-us/library/ff650441.aspx>.

[25] *Servlet Definition*. [online] Dostupné z: <http://techterms.com/definition/servlet>.

[26] *Static Vs Dynamic websites – what's the difference?* [online] Dostupné z: <http://edinteractive.co.uk/static-vs-dynamic-websites-difference/>.

[27] *system_nanotime*. [online] Dostupné z: <http://www.tutorialspoint.com/java/lang/system_nanotime.htm>.

[28] *Class Runtime*. [online] Dostupné z: <https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html>.

[29] *Interface OperatingSystemMXBean*. [online] Dostupné z: <https://docs.oracle.com/javase/7/docs/api/java/lang/management/OperatingSystemMXBean.html>.

A Prílohy

Príloha obsahuje

- Zdrojové súbory testov
- Ukážkové dáta
- Vlastný text práce vo formáte PDF