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, unit 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[18] testy riadia a poháňajú samotný vývoj.

Cieľom tejto bakalárskej práce je napísať testy pre aplikáciu RemSig[1], 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 aplikáciu 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 testov a ich štruktúru. Pokiaľ to je potrebné sú definované použité technológie a dôvod ich použitia. Pri typoch testov s dostupnými výsledkami, sú výsledky uvedené na konci podkapitoly. 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 Digitálne podpisy

Digitálny podpis je matematická technika na overenie autenticity(pôvod dát), integrity(neporušenosť dát) a nepopierateľnosť odoslania dát autorom[25]. 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 vlastní hash, ktorý porovná s hash-om zo správy. Digitálny podpis je špecifický ku konkrétnym údajom a konkrétnemu podpisovateľovi.

2.1 SSL zabezpečenie

SSL(z anglického Secure Socket Layer)[11][12] je počítačový protokol umožňujúci autentizáciu klienta a serveru poskytujúc bezpečnú a šifrovanú komunikáciu. SSL bolo vyvinuté Americkou firmou Netcape Communications v roku 1990. Je to 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 socket-ou a symetrickej kryptografie(jeden kľúč je použitý na šifrovanie aj dešifrovanie). Socket je koncová súčasť komunikácie cez počítačovú sieť[21]. Socket-y 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[24]. Správy z protokolu handshake sú vložené do čistého textu a prepravené podľa špecifikácie relácie.

Pri hadshake protokole sa využívajú X.509 certifikáty.[9] 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. V tomto type útoku by útočník odchytil prvotnú požiadavku klienta o zabezpečené spojenie a poskytol klientovi svoj verejný kľuč. Zaistenie identity servera je praktizované pomocou Certifikačnej autority. Certifikačná autorita vydáva certifikáty a ručí za správnosť údajov v certifikáte.[4] Pokiaľ dôverujeme danej certifikačnej autorite, môžeme veriť aj jej podpísaným certifikátom a verejným kľúčom v danom certifikáte.

2.2 Zabezpečenie súkromného kľúča

Správne zabezpečenie súkromných kľúčov je najdôležitejšia úloha pri asymetrickej kryptografií rovnako ako servera, tak aj klienta. Celá asymetrická kryptografia je postavená na predpoklade existujúceho spôsobu nezistenia súkromného kľúča nežiadúcou osobou.[11] 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 kľuč, je schopný čítať všetku komunikáciu medzi serverom a klientom, ako by 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 [10], odhadovaný čas pri 1024-bit kľúči, by pri použití rovnakého algoritmu trval niekoľko tisíc rokov. Neznamená to absenciu rýchlejších a efektív-

nejších 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 na špecializovanom hardvéri ako je USB token alebo smart card, plastová karta so zabudovaným čipom, etc., alebo mať kľúče uložené na serveri. Nech je zvolená ktorákoľvek možnosť, neodporúča sa uchovávať kľúče v ich prirodzenej, nezašifrovanej forme. Namiesto toho je bežná prax šifrovať súkromné kľúče pomocou užívateľom zvoleného hesla. Ukladanie kľúčov na hardvéri sa považuje sa bezpečnejšie ale nepraktickejšie. v bežnej praxi sa na externé zariadenie uloží iba súkromný kľúč servera, poprípade certifikačnej autority, a iba tie privátne kľúče, ktoré umožňujú prístup k citlivým alebo cenným údajom.

Uloženie kľúča na hardvéri prináša prekážky v dostupnosti, kľúč je potrebné mať vždy pri sebe. Toto je akceptovateľné pre kľúčové časti systému, ale pre overenie pravosti rádových zamestnancov je tento prístup nepraktický a cenovo nevýhodný. Tu sa uplatňuje úschova kľúčov na úložnom servery. Je dôležité aby samotný úschovný systém nebol schopný čítať súkromné kľúče a preto je doporučené využiť princíp šifrovania kľúčov heslom. Tieto kľúče sú stále náchylnejšie útokom typu brute-force(skúšanie každej kombinácie) a útokom, ktoré hádajú heslo , ale je to rozumný kompromis za získanú prenosnosť a ľahšiu spravovateľnosť.

2.3 Remsig

Aplikácia RemSig[1] vznikla na Ústave výpočtovej techniky Masarykovej univerzity, ako riešenie podpisovania dokumentov, okrem iného aj pre štátnu správu. 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 s čím manipuloval. 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[3] 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[2]. Z tohto dôvodu sa budem samotnej aplikácií venovať iba v rozsahu potrebnom pre túto bakalársku prácu.

3 Testy

Testy v informatike sú programy alebo časti programov navrhnuté na overenie funkcionality testovaného produktu. 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[11]. v tejto kapitole sú rozobrané len niektoré testovacie procesy a typy testov, ktoré prevažne súvisia s praktickou časťou.

Jedno z takýchto rozdelení testov je testovanie, či sa aplikácia správa ako sa očakáva, a či sa aplikácia nespráva neočakávane. Pri testovaní očakávaného správania sa pozoruje chovanie aplikácie so správnymi parametrami a s nesprávnymi parametrami. Na druhú stranu pri testovaní neočakávaného správania je snaha dosiahnuť neočakávané správanie testovaného kódu, alebo aplikácie. Testujú sa hraničné hodnoty, a hodnoty za hranicami. Ako parametre sa udávajú úplne nezmyselné hodnoty, znaky v nezvyčajnom kódovaní. V prípade potreby autentizácie môžu byť ako parametre prázdne polia alebo príkazy(DROP DATABASE - 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 proti Vývojom riadenými testami

Pri problematike tvorby testov existuje niekoľko protikladných názorov, pričom najväčší rozpor je v ideálnom čase písania testov. Tu sa

stretávajú dve protichodné myšlienky. Tradičné testovanie a Vývoj riadený testami (z anglického Test-driven development (TDD))[8].

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.

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, ktoré vedú k zefektívneniu celého vývoja.

Prvý krok je napísať test a overiť, že test zlyhá. Nasleduje vytvorenie kódu a spustenie testov. Posledný krok je refaktorovanie. Refaktorovanie je proces úprav a zmeny kódu za účelom zlepšenia priehľadnosti a udržateľnosti kódu pričom sa ale nemení funkcionalita kódu. V tomto kroku sa venuje efektivite, čitateľnosti a udržateľnosti 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.

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.[13][19] 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[22].

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 [14]. 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[23][24]. Výkonnostné testovanie je možné rozdeliť na výkonnostné testy, záťažové test 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[19]. Záťažové testy by tiež mali identifikovať bod, v ktorom aplikácia už nie je schopná ďalej fungovať.

Výkonnostné testovanie slúžia 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 miesta bottleneck 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 parsovanie(rozoberanie, načítavanie) XML dokumentu, načítavanie súborov z disku, inicializovanie databázy, získanie dát z databázy 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[5] . Na využívanie a spravovanie databázy je použitý jazyk MySQL v spolupráci s DbUnit[6]

JUnit – je jednoduché open source rozhranie slúžiace na písanie a spúšťanie opakovateľných testov. JUnit porovnáva návratové hodnoty metód s preddefinovanými hodnotami. Podľa prieskum v roku 2013 z 10 000 projektov na servery github, ktorý slúži na zdieľanie zdrojových kódov, je JUnit, spolu s knižnicou slf4j-api, najpoužívanejšou knižnicou obsiahnutom v 30.7% projektov[7].

MySQL – je open source databázový systém, ktorý v roku 2013 bol najpoužívanejší RDBMS(z anglického relational database management system) klient-server model a zároveň druhý najpoužívanejší RDBMS[16]. Vďaka veľkej kompatibilite so všetkými často používanými jazykmi a podporou pre najčastejšie používané operačné systémy bolo MySQL jasnou voľbou pre databázovú podporu[17].

DbUnit - je nadstavba JUnitu zameraná prevažne na nastavenie databázy do predom známeho stavu. Toto je obzvlášť užitočné pri jednotkových testoch, kde jeden test môže skorumpovať údaje v databáze a viesť k postupnému zlyhaniu ďalších testov a zkorumpovaniu extra dát. DbUnit inicializuje databázu pomocou dátovej sady uloženej v Extensible Markup Language (XML) formáte a pri inicializácii overuje typovú správnosť vložených údajov. RemSig 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.

4.1.3 Výsledky testov

Doterajšie jednotkové testovanie odhalilo niekoľko menších nedostatkov ako bola nesprávna konfigurácia alebo nezachytená nulová výnimka. 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 nesprávne zapísaní SQL príkaz. Pri metóde resetPassword() bolo staré heslo stále funkčné. Metóda signPkcs7 nerozoznávala voľbu algoritmu „SHA-1“ a zlyhávala z dôvodu bielych miest v konfiguračnom súbore profiles.xml.

4.2 Overenie funkcionality

Pri prevode aplikácie RemSig z PHP na Javu bolo potrebné zaistiť kontrolu nezmenenej funkcionality. Rovnako bolo potrebné zachovať nezmenenú podobu komunikácie zo strany klienta. Tento problém bol riešený pomocou serverového Forku.

4.2.1 Použité Technológie

Fork je realizovaný pomocou servletu vloženým medzi klienta a RemSig.

Servlet – je program bežiaci na webovom servery, napísaný v Jave. [15] 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)[16], ako aj spracovávanie dát od užívateľov. Každý servlet musí obsahovať metódy na inicializovanie, spravovanie a zničenie na upresnenie správania daného servletu.

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(neviditeľná 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 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 serverového fork preposlaná klientovi. Následne fork posiela rovnaký post požiadaviek 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.

4.3 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 serverovový fork.

4.3.1 Štruktúra testov

Pri integračnom testovaní sa vytvorili štyri pod priečinky 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 postupne načíta z každého priečinku všetky súbory, pokúsi sa ich načítať a spárovať so správnou metódou a následne zavolá post požiadavku na server. V correctSequence priečinku slúžia dáta na overenie viacerých metód volaných v poradí. Napríklad exportPKCS12(), changePassword(nové heslo), checkPassword(nové heslo), exportPKCS12().

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 jediný aspekt aplikácie a tou je rýchlosť spracovania požiadaviek serverom. Merania bolo po prvotnom „zahriatí“ systému (niekoľko stoviek post požiadaviek) 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 skončenia poslednej požiadavky použitá funkcia Javy nanotime()[20]. 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).

4.4.2 Štruktúra testov

Výkonnostné testy sú schopné testovať časovú odozvu pre vybraný počet iterácii na jednej metóde, ale tiež na sekvencii daných 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 prvé a tesne za posledné volanie meranej metódy. Výsledná hodnota je uložená v Java type long a vrátená užívateľovi.

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 konzistentom 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.

Tabuľka 4.1: Ukážka nameraných hodnôt

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 nasedenie. Navyše bol napísaný aj Serverovský 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. Je potrebné odstrániť všetky nájdené nedostatky a aj potom by som neodporúčal RemSig napísaný v Jave nasadiť, kým nebude dostatočne veľký počet zhodných odpovedí z poskytnutého testu Fork.

Literatúra

A Prílohy

Príloha obsahuje

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