Masarykova univerzita
Fakulta informatiky

Záťažové testovanie webových aplikácií

Bakalárska práca

Oliver Mačejovský

Brno, jar 2016

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.

Oliver Mačejovský

Vedúci práce: RNDr. Tomáš Rebok, Ph. D.

Poďakovanie

Touto cestou sa chcem poďakovať svojmu vedúcemu bakalárskej práce RNDr. Tomášovi Rebokovi, Ph. D. za rady a odborné vedenie pri spracovaní textovej časti práce. Poďakovanie patrí aj môjmu technickému vedúcemu Mgr. Tomášovi Rybkovi za odborné konzultácie pri tvorbe praktickej časti. A v neposlednej rade sa chcem poďakovať za technické zdroje projektom CESNET LM2015042 a CERIT Scientific Cloud LM2015085, poskytované v rámci programu "Projekty velkých infrastruktur pro VaVal".

Zhrnutie

Cieľom tejto bakalárskej práce je záťažové testovanie webových aplikácií používajúcich technológiu AJAX. Hlavnou úlohou bude preskúmať sučasné nástroje a porovnať ich nad aplikáciou Orion od spoločnosti Solarwinds. Výsledkom bude porovnanie nástrojov so súčasným testovacím postupom spoločnosti Solarwinds.

Kľúčové slová

AJAX, záťažové testovanie, Orion, Solarwinds, JMeter, Webload, JSR223, Webdriver

Obsah

Úvod 1
1 **Záťažové testovanie** 3
1.1 *Motivácia* 3
1.2 *Testovacie postupy* 4
1.3 *Testované prvky* 5
2 **AJAX** 7
2.1 *Technológie* 8
2.2 *Výhody a nevýhody* 8
2.3 *Funkčné Testovanie* 9
2.4 *Záťažové Testovanie* 10
3 **Analýza testovania modulu AJAX** 11
3.1 *Testovanie modulu AJAX* 11
3.2 *Testované metriky* 12
4 **Nástroje** 14
4.1 *Webload* 14
4.2 *Jmeter* 14
4.2.1 Webdriver 15
4.2.2 JSR 223 16
5 **Použitie nástrojov** 17
5.1 *Prehliadač Chromium a JSR223* 17
5.2 *Webdriver* 18
6 **Vyhodnotenie nástrojov** 21
6.1 *Prehliadač Chromium a JSR223* 21
6.2 *Webdriver* 22
7 **Záver** 25
8 **Prílohy** 26
Literatúra 27

Zoznam obrázkov

2.1 Diagram fungovania modulu AJAX *7*

3.1 Zvýraznená vonkajšia komunikácia v diagrame fungovania modulu AJAX *11*

5.1 Diagram odchytávania požiadaviek prehliadačom Chromium *17*

5.2 Sekvenčný diagram nástroja Webdriver *19*

6.1 Graf záťaže nástroja na zdrojový stroj nástrojom JSR223 *21*

6.2 Graf záťaže nástroja na zdrojový stroj nástrojom Webdriver *22*

6.3 Graf záťaže nástroja na zdrojový stroj nástrojom Webdriver *23*

6.4 Graf záťaže na zdrojový stroj 3 sučasné spustenými prehliadačmi pomocou nástroja Webdriver *23*

Úvod

Nároky na webové aplikácie sa neustále zväčšujú, musia byť schopné zvládať čoraz väčšie množstvo užívateľov, spolupracovať s rôznymi mobilnými zariadeniami a plynule a spoľahlivo poskytovať bezchybnú službu. Mnoho takýchto aplikácií má rôzne požiadavky na výkon, ktoré je potreba overiť. Veľké nárazové nápory užívateľov, dlhodobá stabilita pri premenlivej záťaži, hranice výkonu, ale aj správanie pri preťažení. Overenie takýchto scenárov ma za úlohu výkonnostné testovanie. Rovnako ako pri iných spôsoboch testovania je pokrytie všetkých možných testovacích scenárov komplikované a náročné na zdroje, preto je potrebné, aby tieto testy boli čo najviac efektívne.

Predmetom testovania v práci je komponenta AJAX, používaná v dynamických webových aplikáciách. Jeho popularita stále rastie z dôvodu veľkej užívateľskej prívetivosti a použiteľnosti. Užívateľ tak nemá pocit ako keby pracoval so statickou webovou stránkou ale dynamickou webovou aplikáciou, nečaká na prázdnej stránke na načítanie celej stránky, odpovede na svoje akcie vidí okamžite bez obnovenia stránky.

Testy AJAX technológie prebiehali v produkte Orion vyvíjaného spoločnosťou Solarwinds. Aplikácia slúži na monitorovanie zariadení v sieti. Ponúka rôzne štatistiky a grafy, ktoré sú zobrazované ako AJAX komponenty. Aplikácia je schopná nezávisle a samostatne obnoviť časti stránky, bez potreby obnoviť celú stránku, alebo ju znova navštíviť. Pri vygenerovanej záťaži sa komponenta musí aj naďalej pravidelne obnovovať a zobrazovať korektné dáta.

Hlavným problémom súčasného postupu je veľká náročnosť na zdroje. Pre každého užívateľa je potreba generovať vlastnú inštanciu prehliadača v ktorej test beží. Náročné je aj monitorovanie väčšieho množstva prehliadačov a zlyhanie prehliadača môže negatívne ovplyvniť výsledok testu.

Vybrané nástroje podrobené testovaniu sú Jmeter a Webload. Jmeter ponúka dve riešenia, Webdriver, ktorý sa súčasne používa v Solarwinds, vytvára inštancie prehliadača. Druhý spôsob komunikuje bez potreby prehliadača priamo cez webové požiadavky. Nástroj Webload je vyvíjaný spoločnosťou Radview a natívne podporuje testovanie AJAX technológie.

Prvá kapitola opisuje záťažové testovanie. Rozoberá rôzne metodológie a používané testovacie postupy vo vývoji softvéru. V druhej kapitole je podrobne opísaná AJAX technológia, jej výhody a použitie v rôznych webových aplikáciách. Ďalej analyzuje jej funkčné a záťažové testovanie. Tretia kapitola sa detailne venuje testovaniu AJAX modulu pomocou nástrojoch na záťažové testovanie a uvádza testované metriky pri testovaní. Štvrtá kapitola stručne opisuje vybrané nástroje určene k testovaniu AJAX aplikácií. Piata a šiesta kapitola sú venované praktickej časti, popisujú použitie nástrojov, namerané výsledky a ich vyhodnotenie. V závere sú zhrnuté výsledky práce a návrhy na budúce zlepšenie týchto nástrojov.

Cieľom práce je vyhodnotenie dostupných nástrojov na záťažové testovanie a preskúmanie ich použitia pri testovaní AJAX technológie. Testy nástrojov budú prebiehať v aplikácii Orion. Pozornosť smeruje na úsporu zdrojov, zrýchlenie testovania a zvýšenie spoľahlivosti testov. Nástroj bude schopný nahrať užívateľské kroky v aplikácii a následne ich opakovať ako testovací scenár. Najdôležitejším faktorom je podpora AJAX technológie. Výsledný nástroj bude schopný vygenerovať záťaž na webovú aplikáciu, zaznamenať dobu odozvy pri záťaži a ohlásiť chybu pri zlyhaní aplikácie.

1 Záťažové testovanie

Kapitola opisuje záťažové testovanie, podmnožinu výkonnostného testovania. V prvej podkapitole je popísaná motivácia prečo záťažovo testovať. Druhá podkapitola opisuje testovacie postupy pri výkonnostných testoch. V poslednej podkapitole je analýza požiadaviek na nástroj, ktorý bude praktickým výstupom.

Vývoj všetkých aplikácií sa riadi funkčnými a nefunkčnými požiadavkami. Funkčné požiadavky sa zameriavajú na chovanie systému pri interakcii s externou entitou a nefunkčné požiadavky sú popisom podmienok na vyvíjaný systém [1]. Výkon aplikácie sa radí k nefunkčným požiadavkám. Patrí k nim aj napríklad bezpečnosť, hardvérové prostredie a prístupnosť. Požadovaný výkon môže byť rôzne špecifikovaný a testovanie tomu musí byť prispôsobené. Prebiehať by ale malo počas celého vývojového cyklu z dôvodu záruky, že výsledné riešenie bude spĺňať všetky požiadavky pre výkon uvedené v špecifikácii (odozva, pamäťová náročnosť, priepustnosť, ...).

1.1 Motivácia

Záťažovým testovaním môžeme dosiahnuť niekoľko rôznych cieľov. Prvým je splnenie akceptačných kritérií pre výkon. Každá aplikácia by mala mať vopred vhodne navrhnuté požiadavky na výkon. Vychádzať by mali z analýzy, ktorá zahŕňa čo najviac faktorov, ako počet užívateľov, odhad počtu súčasných užívateľov, spoľahlivosť, dobu prevádzky a akékoľvek iné faktory ovplyvňujúce výkon. Nesplnenie kritérií môže mať za následok mnoho problémov. Aplikácia nebude schopná vykonať transakciu v požadovanom časovom okne, zmena hardvérových požiadaviek, zvýšenie údržby aplikácie, náročné zmeny v produkčnom prostredí. Ďalším cieľom záťažového testovania je porovnanie výkonnosti iných aplikácií. Komparatívne výkonnostné testy nám dokážu o výkone aplikácie dosť vyjadriť. Môžeme testovať nielen rôzne produkty a ich výkonnosť, ale aj odlišné verzie rovnakej aplikácie. Podľa takýchto testov vieme okamžite povedať, či dané zmeny boli prospešné, alebo rapídne spomalili výkon aplikácie. Ak je aplikácia takýmito testami pokrytá, zmeny sa robia jednoduchšie a istejšie. Posledným z cieľov výkonnostného testovania je vyhodnotenie výkonu

odlišných častí aplikácie. Väčšina aplikácií je zložená z viacerých častí, ktoré medzi sebou komunikujú. Záťažové testy nemajú merať len čas a odozvu vstupu a výstupu. Monitorované by mali byť aj jednotlivé menšie časti aplikácie. Takto dokážeme identifikovať *bottlenecky* (časť, ktorá závažne znižuje celkový výkon aplikácie), ich oprava často dokáže pomerne zvýšiť výkon [2].

1.2 Testovacie postupy

Testovanie bude prebiehať podľa nasledujúcich testovacích postupov. Rôzne testovacie metódy môžu odhaliť slabé miesta v aplikáciách.

- Load testing [3] – Záťažové testovanie

  Tento druh testu skúma chovanie aplikácie pri rôzne generovanej záťaži. Množstvo môže byť náhodne generované, klesajúce alebo rastúce. Z výsledku dostávame dobu odozvy pri generovanej záťaži.

- Stress testing [3] – Náporové testovanie

  Testuje hranice systému. Snaží sa zistiť hranice výkonu aplikácie, pri akom výkone je ešte aplikácia použiteľná. Tento test ale aj overuje správanie pri prekročení tejto hranice. Aplikácia by nemala poškodiť hardvér, dáta a nemala by skončiť v neobnoviteľnom stave.

- Soak testing – „Špongiové“ testovanie

  Vykonanie tohto testu býva časovo náročné. Snaží sa o simulovanie chodu aplikácie pod dlhodobou predpovedanou záťažou. Trvanie testu by malo zodpovedať časovému rámcu v ktorom aplikácia bude fungovať bez vonkajšieho zásahu (reštart aplikácie, vyčistenie logov)

- Spike testing – Výkyvové testovanie

  Test sa snaží o nárazové generovanie záťaže. Overuje, či aplikácia je schopná vykonať aj veľký počet požiadaviek za daný časový úsek, poprípade ich vie správne postupne spracovať.

- Configuration testing – Konfiguračné testovanie

  V tomto teste sa testuje vplyv rôznych konfigurácií na výkon aplikácie. Najčastejšie sa vytvárajú pri vydaní novej verzie aplikácie. Overuje, či aj po aktualizácii bude aplikácia spĺňať požadovanú výkonnosť. Používajú sa aj pri testovaní mobilných verzií aplikácie, kedy je potreba zistiť správanie na veľkom počte rôznych typoch zariadení.

- Capacity testing [3] – Kapacitné testovanie

  Tento test vychádza priamo zo špecifikácie aplikácie. Overuje splnenie výkonnostných cieľov, koľko užívateľov a operácií aplikácia zvládne. Využíva sa k plánovaniu kapacity a analýze záťaže. Určuje, či počet užívateľov môže rásť alebo hrozí problém pri náraste.

- What-If testing – „Čo ak“ testovanie, Prípadové testovanie Testovanie, ktoré zisťuje chovanie aplikácie v rôznych problematických scenároch. Ako sa aplikácia bude chovať, ak príde o nejaké zdroje ? Bude schopná fungovať ďalej? Snaží sa o simuláciu krízových situácií pri podobných aplikáciach.

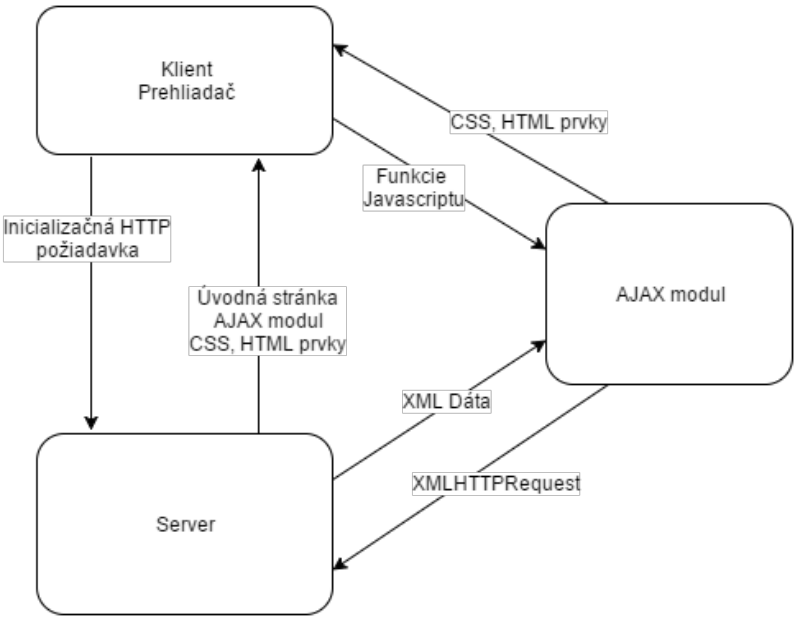
1.3 Testované prvky

Zameranie práce je hlavne na záťažové testovanie a jeho generovanie. Na testovanú aplikáciu sa generuje záťaž, ale zároveň simuluje správanie užívateľa čo najvernejšie. To znamená, že ak po vykonaní akcie užívateľom sa zašle na server 6 požiadaviek, tento počet sa musí zopakovať. V prípade menšieho počtu by aplikácia mohla byť rýchlejšia, ale v praxi požiadavky nezvládať. V prípade väčšieho počtu by to znamenalo síce väčšiu vygenerovanú záťaž, ale takáto záťaž by nikdy nemusela nastať. Generovanie záťaže ale nemôže konzumovať veľké množstvo zdrojov, aj na priemernom užívateľskom zariadení by nástroj mal zvládnuť niekoľko užívateľov. V kapitole 4.2.1 popísaný Webdriver vyžaduje na jedného užívateľa jedno jadro procesora, čo je pomerne veľká náročnosť. Výsledný použitý nástroj by mal spĺňať nasledovné kritéria. Nahrávanie testovacích scenárov, nahranie postupnosti krokov ktoré užívateľ vykoná, následné spracovanie a ulo-

ženie do automatizovateľného skriptu. Takýto skript by mal byť ľahko opraviteľný a prípadne zparametrizovateľný, aby mohol pracovať s dynamickými dátami. Použiteľnosť pre simuláciu viacerých súčasných užívateľov, testy musia byť škálovateľne, aby simulovali jedného užívateľa ale aj niekoľko stoviek. Nástroj bude primárne používaný na testovanie aplikácii pracujúcich na technológii AJAX.

2 AJAX

AJAX aplikácia eliminuje štart-stop správanie na webe vložením prostredníka do komunikácie medzi užívateľom a serverom. Ďalšia vrstva v komunikácii by mala znamenať spomalenie, ale u AJAX-u je tomu naopak. Namiesto načítania stránky prehliadač načíta AJAX modul, ktorý beží na pozadí. Modul zodpovedá za vykresľovanie stránky, ktorú vidí užívateľ a za komunikáciu so serverom. Tieto dve činnosti prebiehajú asynchrónne, takže užívateľ nikdy nevidí prázdnu stránku, na ktorej by mal čakať na odpoveď servera. HTTP požiadavky, ktoré by mali smerovať na server, smerujú do AJAX modulu. Každá odpoveď, ktorá si nevyžaduje priamo odpoveď od serveru, môže byť okamžite vykonaná AJAX modulom – validácia textových polí, editácia dát v pamäti. Rovnako to funguje aj naopak, server môže odoslať dáta, aby ich prehliadač zobrazil bez akcie od užívateľa – oznámenie o novom emaile, zmena aktuálne zobrazených dát.



Obr. 2.1: Diagram fungovania modulu AJAX

2.1 Technológie

AJAX nie je jedna konkrétna technológia. Je to súbor viacerých spoločných technológií, ktoré spolu vytvárajú dynamickú webovú aplikáciu. Sú to XHTML (Extensible Hypertext Markup Language) a CSS (Cascading Style Sheets), DOM (Document Object Model), JSON (JavaScript Object Notation) alebo XML (Extensible Markup Language), XSLT (Extensible Stylesheet Language Transformations), XMLHttpRequest a Javascript. XHTML a CSS sú využité na zobrazenie výstupu, prezentujú výslednú stránku, ktorú vidí užívateľ. Technológia DOM je programovateľné rozhranie pre HTML, XHTML a XML dokumenty. Dáta sú reprezentované v DOM strome ako štruktúrované skupiny prvkov – uzly, ktoré predstavujú objekty s vlastnosťami a metódami. JSON alebo XML sa stará o dátové prenosy. AJAX-ové aktualizácie stránky sú posielané medzi klientom a serverom v JSON alebo XML formáte. Takéto dáta sú uložené na serveri a v prípade potreby sú vyvolané. Následne sa môžu priamo uložiť do DOM stromu alebo dočasne uložiť a neskôr použiť. XMLHttpRequest je rozhranie umožňujúce asynchrónne správanie AJAX-u. Dokáže zasielať HTTP a HTTPS požiadavky počas behu skriptu a nahrávať odpovede serveru naspäť do skriptu. Napriek názvu nepracuje len s dokumentami typu XML ale aj s dátami vo formáte JSON alebo HTML. Výsledne všetky tieto technológie spája Javascript.

2.2 Výhody a nevýhody

Technológia AJAX sa stáva čoraz viac populárnejšou a používanejšou technológiou. Dynamický obsah v mnohých aplikáciach je spracovaný práve AJAX technológiou. Odstraňuje nielen potrebu obnovovať stránku pri novom obsahu, ale aj znižuje záťaž na server, pretože sa od serveru žiadajú len dáta, ktoré sa zmenili. Jednou z výhod je zlepšenie interakcie pre užívateľa, webové aplikácie tak viac pripomínajú klasické aplikácie ako statické webové obsahy. AJAX aplikácie sú kompaktnejšie, jedna webová stránka tak môže obsahovať niekoľko rôznych funkčností.

Technológia má ale aj niekoľko nevýhod. Je celá postavená na Javascripte, čo môže znamenať problémy u prehliadačov, ktoré Ja-

vascript nepodporujú alebo u užívateľov, ktorí podporu Javascriptu vypínajú. Obsah sa nemusí zobrazovať správne alebo odozva stránky nebude fungovať. Ďalšou nevýhodou býva nepoužiteľnosť tlačítka Späť a Obnoviť. Dynamický obsah často URL stránku v prehliadači nezmení a obnovenie stránky môže viesť k strate dát v aplikácii, tlačítko Späť tak bude viesť na obsah, ktorý sa zobrazoval pred dynamickým spracovaním.

2.3 Funkčné Testovanie

Použitie AJAX technológií môže výrazné zvýšiť použiteľnosť webovej aplikácie. Aplikácie sa týmto ale stávajú komplikovanejšie, pretože spracovanie sa vykonáva na klientskej ale aj serverovej strane. Použitie nejakého už implementovaného rozhrania môže prácu uľahčiť, ale zároveň môže viesť k menšiemu porozumeniu kódu. Vyhodnotenie riskov nejakej funkčnosti alebo aplikácie tak môže byť zložitejšie.

Zraniteľnosti AJAX aplikácií sú podobné s klasickými webovými aplikáciami. Neoverenie vstupu od užívateľa môže viesť k SQL injekcii [4], ktorá môže poškodiť dáta v databázy. Ide o spôsob v ktorom sa do textového poľa vloží SQL príkaz, napríklad DROP TABLE, čo môže mať za následok zmazanie dát v databázy. Ďalej sa môžu vo vstupe neoprávnene modifikovať premenné alebo parametre funkcií. Zlyhanie pri autentifikácii a autorizácii, spôsobí problémy s dôveryhodnosťou a integritou aplikácie.

Špecifické pre AJAX sú CSRF(Cross-site request forgery) a XSS(Cross-site scripting). CSRF zneužíva dôveru, ktorú ma aplikácia k prehliadaču užívateľa. AJAX môže na stránke používať niekoľko rôznych na sebe nezávislých modulov. V prípade, že sa požiadavky od týchto modulov neoverujú korektne, útočník môže odoslať požiadavku do prehliadača z úplne inej stránky, ktorá však bude obsahovať škodlivé dáta, alebo bude vykonávať neoprávnené akcie. Prehliadač však požiadavku spracuje, pretože predpokladá, že prišla z niektorého autorizovaného modulu. XSS naopak zneužíva dôveru užívateľa k webovej aplikácii. Útočník tak môže do webovej stránky vložiť akýkoľvek skript, ktorý sa spustí pri načítaní, napríklad v diskusnom fóre do komentára, do odkazu obrázku alebo kdekoľvek kde takýto vstup nie

je ošetrený. AJAX pre svoje spustenie vyžaduje povolenie Javascript-u a tak vždy po jeho načítaní sa spustí aj škodlivý skript.

2.4 Záťažové Testovanie

Záťažové testovanie AJAX aplikácií je zložitejšie ako je tomu u klasických webových stránok. Tie je celkom jednoduché namodelovať a simulovať. Stav užívateľa je kombinácia stavu klienta a servera. Stav klienta pozostáva z dát prehliadača, tvorenými súbormi *cookies* (malé dáta odosláne zo stránky a uložené v prehliadači), a z reprezentácie webovej stránky, čo zahŕňa formuláre na stránke a požiadavky v URL adrese. Pri správnej simulácii klientskej strany na stave servera nezáleží, pretože sa podľa klienta upraví. Pri simulácii AJAX klienta je to ale komplikovanejšie. Javascript, ktorý sa po načítaní stránky spustí, či už automaticky alebo ako odpoveď na činnosť užívateľa, môže upraviť štruktúru stránky, jej obsah a úplne tak zmeniť stav klienta.

Niektoré testovacie nástroje môžu simulovať reálny prehliadač a opakovať pohyby myšou alebo vstupy z klávesnice. U nástrojov na generovanie záťaže to je ale problém. Pre spustenie viacerých prehliadačov zároveň je potrebné veľké množstvo zdrojov. Ďalším problémom môže byť ale aj zdieľanie rovnakých súborov, cookies alebo dočasnej pamäte.

Riešením je simulácia užívateľov na HTTP vrstve, čo znamená simuláciu komunikácie prehliadača namiesto simulácie kompletného prehliadača. U AJAX aplikácií, kedy sa formuláre zo stránky vždy odošlú v URL, zvyčajne v tvare (application/x-www-form-urlencoded) je môžné rôznymi nástrojmi pomerne ľahko automatizovať, v kapitole refwebload analyzovaný nástroj Webload takúto možnosť ponúka.

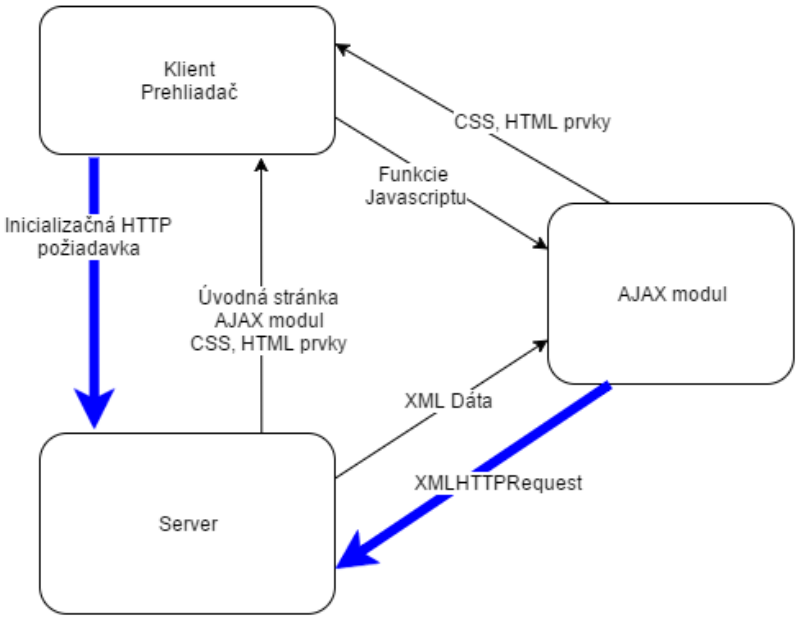
U ostatných AJAX aplikácií, ktoré URL nemenia a využívajú metódy AJAX rozhrania, je potreba simulovať posielané dáta. Medzi najpopulárnejšie patrí JSON alebo XML. Tieto dátové prenosy je potreba odchytiť a následne replikovať pre generovanie rovnakej záťaže.

3 Analýza testovania modulu AJAX

Kapitola obsahuje kompletnú analýzu potrebnú pre návrh testov. Určuje, ktoré funkčnosti je potreba otestovať a ako replikovať generovanie záťaže na AJAX aplikáciu. Popisuje vytvorenie testovacieho scenára v testovanej aplikácii Orion.

3.1 Testovanie modulu AJAX

Princíp fungovania AJAX modulu už bol vysvetlený v kapitole refajax. Podkapitola opisuje testovanie tejto funkčnosti. Na začiatok je potreba zanalyzovať aké funkčnosti na module AJAX budú testované. Pri záťažovom testovaní to bude najmä komunikácia so serverom, v diagrame 3.1 označené modrou farbou, aby sme zistili dobu odozvy.



Obr. 3.1: Zvýraznená vonkajšia komunikácia v diagrame fungovania modulu AJAX

Priebeh testovacieho scenára bude nasledovný. Užívateľ na začiatku zadá do prehliadača webovú stránku. Prehliadač odošle na

cieľový server inicializačnú požiadavku podľa zadanej URL. Odpoveď bude obsahovať kompletnú stránku v HTML a CSS dátach, naviac v nej bude javascriptový AJAX modul. Prehliadač tieto dáta potrebuje spracovať a načítať. Táto je požiadavka je prvá, ktorú potrebujeme reprodukovať. Užívateľ teraz už vidí obsah stránky, ktorý ale nemusí byť kompletný. AJAX-ové komponenty sa môžu načítavať priebežne, preto potrebujeme sledovať komunikáciu, kedy AJAX modul oznámi, že komponenta sa nahrala celá. Tento rozdiel je potreba pokryť pri testovaní dynamických stránok. Musia sa sledovať nielen odpovede servera, ale aj stav prehliadača.

Testovací scenár v aplikácií Orion bude simulovať rovnaký postup. Prvým krokom bude prihlásenie do aplikácie, po ktorom sa na server odošle inicializačná požiadavka. Následne sa zobrazí úvodná stránka, na ktorej budú prvky využívajúce AJAX – grafy a tabuľky. Na tieto komponenty bude zamerané záťažové testovanie. Testovanie sa zameria na ich odozvu a vplyv záťaže na ich kompletné načítanie.

3.2 Testované metriky

Ak je stanovený testovací scenár a postup testovania, je potreba stanoviť aké metriky budú pri testovaní sledované. Pre pochopenie celkovej výkonnosti systému musíme zvažovať nasledujúce tri metriky. Vnímaná systémová výkonnosť, je to výkonnosť, ktorú vníma server generujúci záťaž. Meranie prebieha pre všetky generované požiadavky a prijímané odpovede. Druhou metrikou je užívateľské vnímanie, určuje nakoľko je výkonnosť stránky užívateľsky prívetivá. Meria, koľko skutočne trvá načítanie stránky v prehliadači a či je stránka interaktívna aj pri záťaži. Poslednou metrikou je reálna výkonnosť systému. Patria tu tradičné KPI(Key Performance Indicators), čo sú kľúčové indikátory výkonu, ako záťaž procesoru, pamäte a objem sieťovej komunikácie. Ich meranie prebieha počas celého testu.

Každá z metrík je kombináciou rôznych meraní. Doba odozvy je čas od inicializácie požiadavky až po jej dokončenie. Indikuje výkonnosť celého systému počas testu. Meranie času, ako metriky, reprezentuje priemerný čas odozvy v akýkoľvek konkrétny čas testu. Latencia je taktiež časovou metrikou, meria čas prvej odpovede na požiadavku. Určuje čas prijatia prvého bajtu odpovede od zaslania požiadavky na

server. Ďalšími metrikami je počet súčasných užívateľov, počet chybových hlásení (chyby vygenerované kvôli vypršania času na odpoveď, zlému pripojeniu alebo odmietnutiu požiadavky) a záťaž na sieť.

Bakalárska práca sa sústreďuje na záťažové testovanie. Pre tento testovací postup sú dôležité hlavne výkonnostné a časové metriky. Pri testovaní bude snaha o vytvorenie čo najväčšej záťaže, aby časové metriky boli akceptovateľné a aplikácia použiteľná. Všetky tieto metriky nám pomôžu identifikovať silné a slabé časti aplikácie, a dokážeme nimi určiť výhody a nevýhody nástrojov.

4 Nástroje

Kapitola ponúka prehľad nástrojov, ktoré podporujú testovanie AJAX aplikácií. Nástroje sú schopné zachytávať metriky z predchádzajúcej kapitoly a vyhodnocovať ich. Použitie týchto nástrojov bude záťažovým testovaním Orion produktu.

4.1 Webload

Webload je nástroj vyvíjaný spoločnosťou Radview určený k záťažovému testovaniu webových aplikácií. Na webových stránkach je uvedená podpora pre požadované technológie ako AJAX, HTTP, Javascript a mnoho ďalších. Použitá bola skúšobná 30 dňová verzia, ktorá je limitovaná na 5 virtuálnych užívateľov. Plná verzia disponuje generovaním neobmedzenej záťaže, ktorá môže byť distribuovaná z rôznych zdrojov (Linux, Windows) a parametrizovatelnosť dokáže zmeniť statickú reláciu na dynamickú reláciu čerpajúcu z rôznych zdrojov. Webload poskytuje aj rôzne záznamy, ako pri behu testu tak aj vyhodnotenie na konci testu. Nahrávanie testovacích scenárov prebieha pomocou proxy nahrávača. Scenár je vytvorený nahrávaním HTTP/S komunikácie medzi webovým klientom a web serverom aplikácie. Výsledný WebLoad test je napísaný v JavaScripte. Tento štandardný, všadeprítomný skriptovací jazyk je najbežnejšie používaný pri vývoji webových aplikácií a je považovaný za najvhodnejší jazyk pre vytváranie a editáciu testovacích skriptov pre webové aplikácie. Webload uvádza mnoho rozširujúcich objektov a funkcií do štandardného JavaScriptu, tým umožňuje pokročilé skriptovanie a pridáva viac logiky do testovacieho skriptu. Takéto pokročilé skriptovanie pomáha s parametrizáciou požiadaviek a verifikáciou odpovedí.

4.2 Jmeter

Jmeter je open source projekt od spoločnosti Apache Software Foundation. Je napísaný v jazyku Java a jeho zámerom je záťažové testovanie funkcionalít a meranie výkonnosti. Jmeter bol pôvodne zameraný na testovanie webových aplikácií, ale od jeho začiatku bolo pridaných

mnoho nových funkcionalít, takže momentálne sa dá využiť rovnako na statické ako aj dynamické zdroje. Jmeter je zadarmo, je ľahko rozšíriteľný a má silnú podporu od komunity. Jednou z jeho predností je distribuované generovanie záťaže. Hlavní systém - Master System, Controller, obsahuje užívateľské rozhranie alebo testovací XML skript. Ostatné uzlové body - Node, Load generators prijímajú RMI (Java Remote Method Invocation) volania, tie obsahujú kópiu testovacieho plánu a vykonávajú rovnaké kroky v cieľovej webovej aplikácii.

Jednou z najväčších nevýhod Jmeter-u je nedostatočná podpora pri nahrávaní scenárov. Jmeter používa proxy, ktorý nedokáže nahrávať HTTPS komunikáciu. Preto je potrebné využiť niektorý z rozšírení ktoré Jmeter ponúka. Ďalšou nevýhodou je neschopnosť spustenia Javascriptu. Jmeter nie je prehliadač a to znamená, že AJAX-ové volania nie sú automatický vykonané keď Jmeter nahraje webovú stránku. Požiadavky sa dajú zachytiť cez proxy, ale potom sú uložené ako osobitné vzorkovače - Samplers a nie je možné ich spustiť paralelne ako to robí prehliadač. Ako riešením tohto problému som vybral dve varianty - Webdriver a prispôsobený vzorkovač JSR223.

4.2.1 Webdriver

Veľká časť výkonnostného testovania sa meria na strane servera, ale s pokrokom v technológiách sa čoraz viac logiky presúva priamo na klientskú stranu. Ovplyvňuje to ale výkon aplikácie alebo webovej stránky vnímaný užívateľom. Takéto metriky nie sú ale dostupné bez použitia rozšírenia Webdriver. Webdriver automatizuje vykonávanie a zbieranie metrík priamo z prehliadača, zo strany klienta. Dokáže zaznamenávať spustenie Javascript-u (napríklad AJAX), CSS transformácie [5] (animácie) alebo rôzne rozšírenia webu, Google Analytics [6]. Všetky tieto prvky ovplyvňujú výslednú výkonnosť a Webdriver sa snaží o meranie času kompletného načítania celého obsahu.

Veľkou nevýhodou je ale náročnosť na zdroje. Každý test potrebuje vlastný prehliadač, a to predstavuje približne jedno jadro procesora na užívateľa. Riešenie predstavujú veľké výpočetné cloudy, napríklad Blazemeter[1]. Takéto cloudy zvládnu veľkú výpočetnú náročnosť, ale

1. www.blazemeter.com

takéto služby sú zvyčajne drahé, a malé spoločnosti si takéto riešenie nemôžu dovoliť.

4.2.2 JSR 223

JSR 223 je rozhranie, ktoré slúži na spúšťanie skriptov priamo v zdrojovom kóde [7]. Jmeter využíva implementáciu vo forme „JSR 223 Sampler“. Sampler je vzorkovač, ktorý zbiera vzorky z bežiaceho testu. To znamená, že je schopný spúšťať rôzne skripty z ktorých môže vyčítať potrebné hodnoty na vyhodnotenie celého testu.

Implementácia je v skriptovacom jazyku Groovy. Jeho výhodou je rozhranie Compilable[2]. To má za následok, že jeho výkon je skoro tak dobrý ako Java kód.

Skript sa snaží o simuláciu užívateľa pri používaní stránky ale zameriava sa len na AJAX technológiu. Natívny Jmeter by v prípade, že sa na stránke nachádza AJAX prvok, meral len dobu načítania stránky a tento prvok by sa načítal až na konci. V prípade viacerých prvkov by sa tieto požiadavky zaradili do radu a odosielali by sa jednotlivo. Takto ale prehliadač s AJAX-om nepracuje. Požiadavka na načítanie prvku by sa mala odoslať hneď na začiatku, a to pre každý prvok. Výhodou skriptu je, že všetky tieto prvky sa uložia do kolekcie a paralelne posielajú do AJAX modulu. Takto dostávame rovnakú generovanú záťaž na AJAX modul, akú generuje prehliadač.

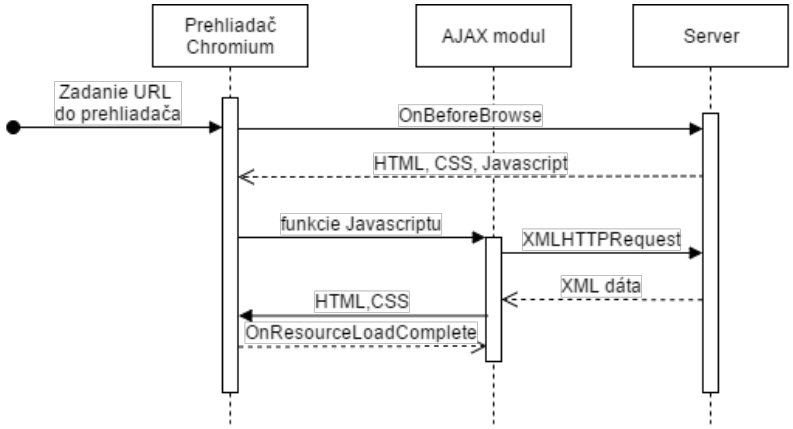
2. http://docs.oracle.com/javase/7/docs/api/javax/script/Compilable.html

5 Použitie nástrojov

Kapitola obsahuje popis použitia vybraných testovacích nástrojov z predchádzajúcej kapitoly, ich výhody, nevýhody a návrhy na zlepšenia. Vybrané nástroje sú Webdriver a vzorkovač JSR223. Oba nástroje sú opísane sekvenčným diagramom a pracujú v prostredí nástroja Jmeter. V ďalšej kapitole je ich vyhodnotenie a porovnanie.

5.1 Prehliadač Chromium a JSR223

Tento testovací postup vyžaduje dva nástroje. Prvým je rozšírený prehliadač Chromium. Implementácia rozšírenia prebiehala nad rozhraním CefSharp, čo je nenáročná .NET nadstavba nad pôvodným CEF – Chromium Embedded Framework[1]. Cefsharp ponúka webové rozhranie pomocou Windows Presentation Foundation – WPF [8] a WinForms [9], v ktorých sa zobrazuje grafické rozhranie webového prehliadača. Potrebná zmena prebehla v triede RequestHandler. Implementuje všetky metódy na prácu s požiadavkami zasielanými na server.



Obr. 5.1: Diagram odchytávania požiadaviek prehliadačom Chromium

1. https://bitbucket.org/chromiumembedded/cef

Z metód na diagrame 5.1 je potreba zistiť, kam požiadavky smerujú a tieto URL zaznamenať. Prvou metódou je OnBeforeBrowse, jej volanie prebieha na začiatku načítania stránky. Pre celkový proces to znamená, že všetky zdroje sa začínajú objavovať na stránke. Druhou metódou je OnResourceLoadComplete, tá informuje o dokončení načítania danej komponenty. Kopírovaním tejto dvojice vytvárame rovnakú záťaž na AJAX-ový modul ako pri bežnom prehliadači.

Modifikovaný prehliadač takto zaznamená celý priebeh testu a vytvorí z neho zoznam URL adries, na ktoré sa požiadavky zaslali. Ďalším krokom je vzorkovač JSR223. Obsahuje skript, ktorý zoznam syntakticky zanalyzuje a následne zreplikuje zaslanie všetkých požiadaviek bez potreby prehliadača.

Z vybraných URL zostaví kolekciu a pomocou HTTP Apache klienta vykoná GET požiadavku. Generovanie prebieha v každom vlákne osobitne a tak sú všetky odosielané paralelne. Bez rozšírenia by sa spúšťali sekvenčne za sebou, to by ale už nesimulovalo správanie prehliadača. Jmeter totiž každú takúto požiadavku registruje ako individuálnu vzorku. Pre užívateľa je ale vzorkou celá stránka a všetky komponenty na nej.

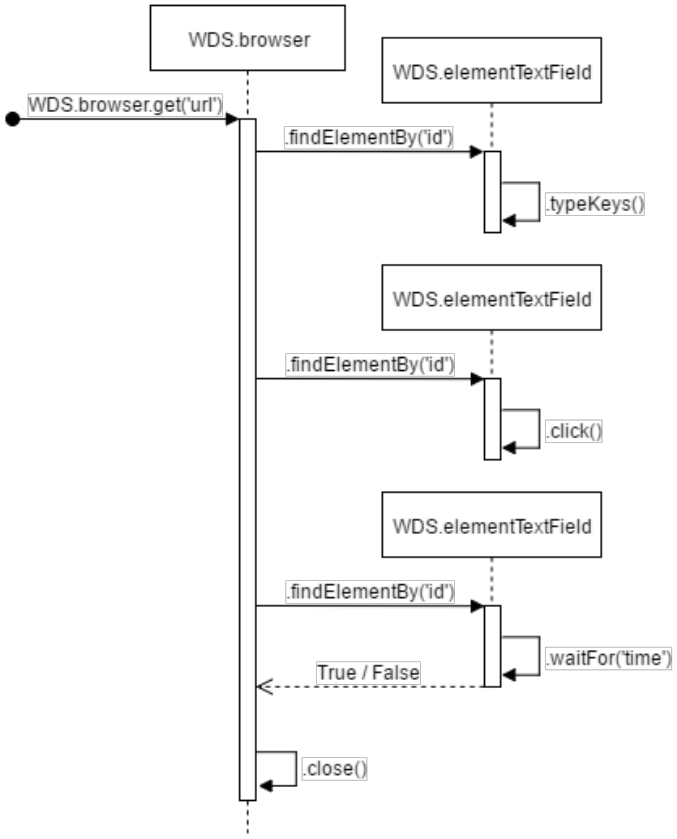
Nevýhodou tejto implementácie je jej momentálne použitie len pre prvú URL, ktorá spúšťa načítanie AJAX-ových modulov na stránke. Takže ak testovací scenár obsahuje presun na inú stránku alebo jej obnovenie, simulácia sa bude odlišovať, pretože pre obe stránky vygeneruje záťaž paralelne. Riešením je synchronizácia a zaznamenávanie času pri nahrávaní v prehliadači. Vygenerovaný zoznam by tak neobsahoval len URL ale aj časové známky kedy presne bola požiadavka na danú URL zaslaná.

5.2 Webdriver

Nástroj Webdriver funguje na princípe abstrakcie webového prehliadača do objektu WDS.browser. Pomocou metód rozhrania[2] sa s ním dá pracovať ako s klasickým prehliadačom.

Testovací plán pre tento nástroj je prihlásenie do aplikácie Orion a zobrazenie všetkých dynamicky zobrazovaných elementov na stránke. V tomto prípade to sú grafy zaťaženia siete.

2. http://www.jmeter-plugins.org/wiki/WebDriverSampler



Obr. 5.2: Sekvenčný diagram nástroja Webdriver

Implementácia testovacieho scenára prebiehala podľa diagramu 5.2. Na začiatok sa metódou WDS.browser.get zobrazí prihlasovacia stránka Orion aplikácie. Parameter metódy je URL. Po zobrazení stránky je potrebné sa prihlásiť, nástroj musí nájsť textové polia pre vyplnenie mena a hesla, následne nájsť tlačidlo prihlásenia a kliknúť naň. Hľadanie objektov na stránke funguje pomocou metódy WDS.browser.findElements. Je možné použiť niekoľko rôznych spôsobov identifikácie. Opis všetkých sa nachádza v dokumentácii metódy By[3]. Nástroj používa atribút ID. Tento spôsob je najefektívnejší a najčastejšie sa využíva v automatizácii, vývojár takto môže vopred uľahčiť automatizáciu tým, že ich

3. http://selenium.googlecode.com/svn/trunk/docs/api/java/org/openqa/selenium/By.html

napevno určí. Nie vždy je to ale možné, napríklad technológia Wicket [10] generuje tieto ID vždy iné a v teste sa tak stávajú nepoužiteľné.

Samostatné elementy taktiež predstavujú objekty s metódami. Na vyplnenie textových polí sa používa metóda sendKeys, ktorá simuluje stláčanie kláves. Do parametru sa predáva prihlasovacie meno a heslo. Objekt tlačidla prihlásenia disponuje metódou click(), ktorá simuluje kliknutie myšou na element.

Po prihlásení sa overí, či presmerovanie bolo správne a užívateľ prihlásený. V pravej hornej časti stránky sa má objaviť prihlasovacie meno súčasného prihláseného užívateľa. Tento textový reťazec sa porovnáva s textovým reťazcom, ktorý sa vpisoval do textového poľa pri prihlásení.

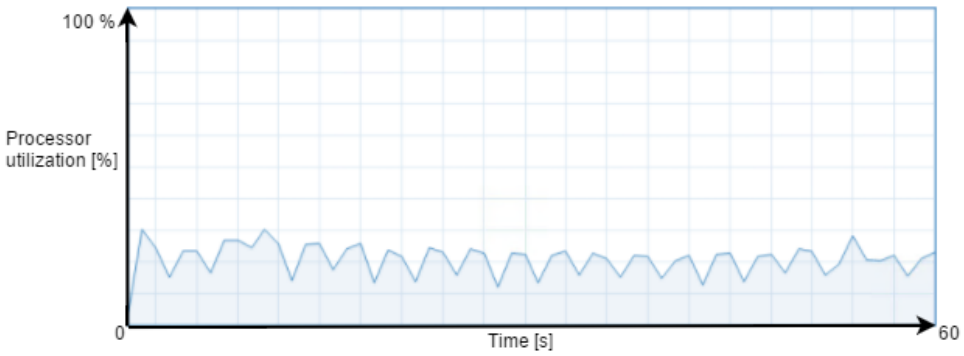
Na konci testu je overenie dynamických elementov – grafov. Tieto elementy sa neobjavujú hneď po načítaní stránky. Na stránke ich je možné vidieť až po vykonaní skriptov, ktoré pracujú asynchrónne. Preto lokalizácia všetkých týchto elementov prebieha tak, že pri hľadaní metódou FindElement sa použije objekt WebDriverWait, ktorému sa dá parametrom uviesť doba maximálneho čakania na element. Metóda WebDriverWait.Until potom vždy pri hľadaní elementu čaká danú dobu.

6 Vyhodnotenie nástrojov

Výsledné zhodnotenie obsahuje porovnanie záťaže generovanej na virtuálnom stroji s bežiacim skriptom v oboch nástrojoch. Špecifikácia stroja je 4 virtuálne jadrá procesora a 8GB pamäte. Sledovanou metrikou bude taktiež spoľahlivosť testov a ich integrita, či sa požiadavky generované prehliadačom zhodujú s požiadavkami generovanými nástrojmi.

6.1 Prehliadač Chromium a JSR223

Druhým nástrojom bol JSR vzorkovač. Ten ale vyžaduje zdrojové dáta z prehliadania webovej stránky. V práci je použitý upravený prehliadač Chromium, ktorý tieto dáta vytvára pri návšteve stránky. Úprava je rozšíriteľná, prípadné filtrovanie iných požiadaviek je možné v jeho zdrojovom kóde. Samotný vzorkovač je nenáročný a negeneruje veľkú záťaž. Priložený graf ukazuje záťaž pri 100 užívateľoch, ktorý sa gene-



Obr. 6.1: Graf záťaže nástroja na zdrojový stroj nástrojom JSR223

rujú v priebehu 60 sekúnd. Každý užívateľ zašle rovnaké požiadavky ako v prípade priebehu scenára v prehliadači. Sledovanie komunikácie na servery a na zdrojovom stroji pomocou nástroja WireShark[1]

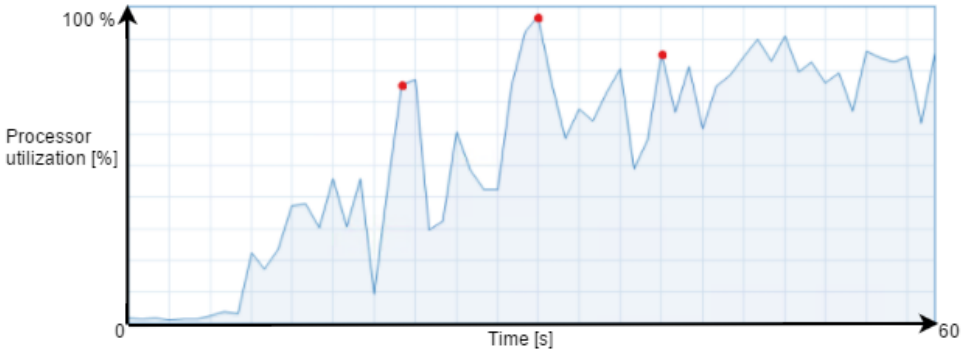
1. www.wireshark.com

preukázalo, že požiadavky sú identické. Problémom je ale čas požiadaviek. U nástroja Webdriver sa tieto požiadavky generovali s časovými rozdielmi, tak ako užívateľ prehliada testovanú stránku. Takáto synchronizačná zmena by ale bola pomerne náročná, bližšie bude špecifikovaná v ďalšej kapitole. V prípade že chceme merať odozvu jednotlivých komponent pri záťaži to ale nie je prekážkou.

Pre generovanie záťaže je jednoznačne lepším nástrojom vzorkovač JSR223. Webdriver ale ponúka širšiu podporu pri rôznom testovaní stránky a preto je určite vhodnejším na funkcionálne testy.

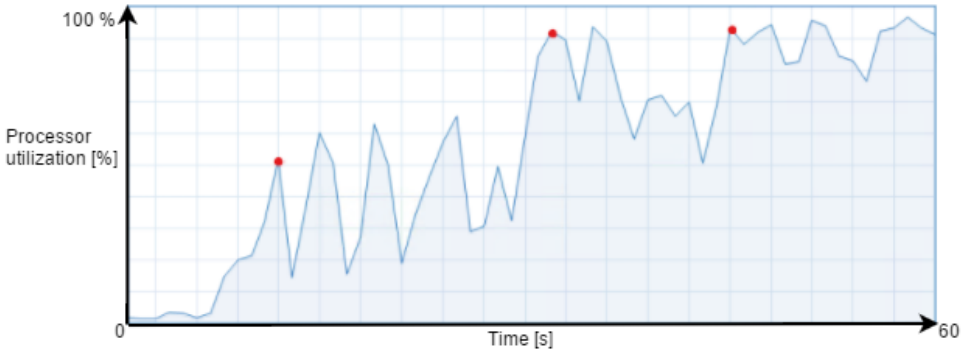
6.2 Webdriver

Prvým testovacím nástroj je Webdriver. Nárast záťaže pri spúšťaní prehliadača bol citeľný už počas implementácií častí testovacieho scenára. Štart prehliadača využíval procesor na viac ako 40% výkonu. Okrem záťaže sa objavili problémy s kompatibilitou prehliadača, pre používanie nástroja je potreba vybrať správnu verziu prehliadača. V tomto prípade to bola dva roky stará verzia prehliadača Firefox. Ako implementačný jazyk bol použitý Javascript, Jmeter ale ponúka rôzne iné možnosti. V Javascripte je napísaný aj celý testovací scenár. Z priložených grafov 6.2 a 6.3 vyplýva, že Webdriver je pomerne



Obr. 6.2: Graf záťaže nástroja na zdrojový stroj nástrojom Webdriver

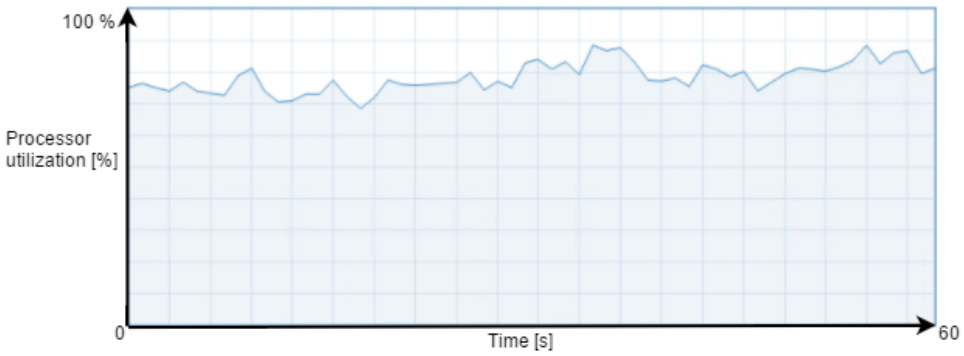
náročný. Červené body označujú jednotlivé spustenia prehliadačov. V ojedinelých prípadov takéto zaťaženie procesor ovplyvnilo natoľko,



Obr. 6.3: Graf záťaže nástroja na zdrojový stroj nástrojom Webdriver

že výsledky testu neboli správne. Jednotlivé testy bežali pri troch súčasne spustených prehliadačoch. Pri spustení štyroch sa výsledky testu zhodovali len vo veľmi malom počte prípadov.

Najväčším problém sa tak zdalo spustenie prehliadača. Následná úprava skriptu tak mala pred samotným testom pripraviť a spustiť jednotlivé prehliadače. Riešenie ale nebolo úspešné, réžia prehliadačov



Obr. 6.4: Graf záťaže na zdrojový stroj 3 sučasné spustenými prehliadačmi pomocou nástroja Webdriver

je taktiež zdrojovo náročná. Vo výsledku sa Webdriver prejavil ako

užitočný nástroj na funkčné testovanie, ale na generovanie záťaže je nepoužiteľný pre veľkú spotrebu zdrojov.

7 Záver

Vývoj dynamických aplikácií sa neustále ďalej rozvíja a s nimi aj testovacie nástroje určené k ich testovaniu. JSR223 predstavuje vhodné riešenie ale je potreba zautomatizovať aj generovanie dát. Jedno z riešení je spojiť JSR223 a Webdriver do jedného nástroja. Na začiatku by Webdriver prešiel testovací scenár a zaznamenal by celú komunikáciu ktorá prebehla z prehliadača. Následne by JSR223 tieto dáta využil a generoval podľa nich záťaž bez potreby znova spustiť prehliadač. Ďalším vhodným prídavkom do JSR223 je časová synchronizácia. Nástroj sa momentálne orientuje na aplikácie, ktoré bežia na jednej stránke. To znamená, že po načítaní stránky sa paralelne odošle požiadavka na všetky AJAX komponenty na stránke, tak ako to robí prehliadač. Problém vzniká napríklad pri obnovení celej stránky alebo prechode na inú stránku. Na novej stránke sa znova paralelne odošlú všetky požiadavky. JSR223 ale odošle všetky požiadavky už po prvej inicializačnej požiadavke. Riešením by bolo uchovávať časovú známku pre všetky takéto požiadavky. Nástroj by ich tak vedel časovo oddeliť a neposielal by ich všetky paralelne.

Cieľom práce bolo preskúmať nástroje určené na záťažové testovanie. Výsledný nástroj JSR223 sa pri testovaní prejavil ako menej náročný nástroj oproti súčasnému riešeniu Webdriver. Nástroj je môžné použiť v rozsiahlom testovaciom prostredí ako je Solarwinds, ale aj pre menšie osobné použitie ako testovací nástroj pri vývoji.

8 Prílohy

Kapitola opisuje opis použitia praktickej časti v prílohe práce. Príloha obsahuje dvojicu zabalených súborov.

Prvým je Jmeter. Spustiteľný súbor je jmeter.bat, ktorý sa spúšťa s parametrom
-Dwebdriver.firefox.bin="\FirefoxPortable\FirefoxPortable.exe"

Po spustení je môžné si po kliknutí na File a Open vybrať, ktorý scenár sa má nahrať. *Webdriver.jmx* je pre nástroj Webdriver a *JSR223.jmx* pre vzorkovač JSR233. V prípade výberu JSR 223 je potreba spusitiť aj druhý balík. CefSharp-master predstavuje projekt prehliadača Chromium. Cesta k spustiteľnému súboru CefSharp.WinForms.Example.exe sa nachádza v zložke CefSharp.WinForms.Example\bin\x86\Debug

Po spustení sa otvorí prehliadač v ktorom je možné nahrať testovací scénar pre vzorkovač JSR233. Webdriver obsahuje nahraný scénar detekcie AJAX komponenty v Orion aplikácii.

Literatúra

[1] J. Arlow and I. Neustadt. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. Computer Press, 2007. ISBN: 9788025115039.

[2] Ian Molyneaux. *The Art of Application Performance Testing: Help for Programmers and Quality Assurance*. 1st. O'Reilly Media, Inc., 2009. ISBN: 0596520662, 9780596520663.

[3] J. Meier et al. *Performance Testing Guidance for Web Applications: Patterns & Practices*. Redmond, WA, USA: Microsoft Press, 2007. ISBN: 9780735625709.

[4] Justin Clarke. *SQL Injection Attacks and Defense*. 1st. Syngress Publishing, 2009. ISBN: 1597494240, 9781597494243.

[5] E.A. Meyer. *Transforms in CSS: Revamp the Way You Design*. O'Reilly Media, 2015. ISBN: 9781491928233.

[6] B. Clifton. *Advanced Web Metrics with Google Analytics*. ITPro collection. Wiley, 2012. ISBN: 9781118239582.

[7] Dejan Bosanac. *Scripting in Java^{TM}: Languages, Frameworks, and Patterns*. First. Addison-Wesley Professional, 2007. ISBN: 9780321321930.

[8] Adam Nathan. *Windows Presentation Foundation Unleashed (WPF) (Unleashed)*. Indianapolis, IN, USA: Sams, 2006. ISBN: 0672328917.

[9] Erik Brown. *Windows Forms Programming with C#*. Greenwich, CT, USA: Manning Publications Co., 2002. ISBN: 1930110286.

[10] M. Dashorst and E. Hillenius. *Wicket in Action*. In Action Series. Manning, 2009. ISBN: 9781932394986.