Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Zátěžové testování HTML5 aplikace

Plzeň 2017 Luděk Kaňák

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 2. května 2017

Luděk Kaňák

Poděkování

Rád bych poděkoval firmě KadeL Data servis a vedoucímu práce Ing. Petru Kalousovi za ochotný přístup, pomoc a čas strávený při řešení bakalářské práce. Mé poděkování patří též Doc. Ing. Pavlu Heroutovi Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích.

Abstract

The goal of this bachelor thesis is to execute performance tests for an HTML5 web application used by the company, KadeL Data servis. The work begins with an introduction to the problem of performance testing. There follows the selection of a suitable tool for testing based on the company requirements as well as proposed multi-criteria evaluation. Using the selected tool, JMeter, test sets were implemented according to a proposed design. After performing the tests, the results were analysed, founded errors being reported. Finally, the performance tests were included into continuous integration by tool Jenkins.

Abstrakt

Cílem bakalářské práce je provést zátěžové testování webové HTML5 aplikace ve firmě KadeL Data servis. Na začátku práce proběhlo seznámení se s problematikou zátěžového testování. Následně byl proveden výběr vhodného nástroje pro testování na základě požadavků ze strany firmy a vytvořeného multikriteriálního ohodnocení. Ve vybraném nástroji – JMeter – byly podle návrhů implementovány sady testů. Po vykonání vytvořených testů se provedla analýza výsledků a nalezené chyby byly nahlášeny. Na závěr byly zátěžové testy začleněny do kontinuální integrace pomocí nástroje Jenkins.

Obsah

1 Úvod **5**

2 Zátěžové testy **6**
2.1 Přínos zátěžových testů 6
2.2 Měření zátěže 6
2.2.1 Doba odezvy 6
2.2.2 Dostupnost 7
2.2.3 Propustnost 7
2.2.4 Využití zdrojů 7
2.3 Typy zátěžových testů 7
2.3.1 Výkonnostní testy 7
2.3.2 Stress testy 8
2.3.3 Spike testy 8
2.3.4 Smoke testy 8
2.3.5 Testy infrastruktury 8
2.3.6 Soak testy 9
2.4 Průběh testování 9
2.4.1 Určení testovacího prostředí 9
2.4.2 Stanovení přijatelných kriterií 9
2.4.3 Návrh testovacího plánu 10
2.4.4 Příprava testovacího prostředí 10
2.4.5 Implementace testů 10
2.4.6 Provedení testů 10
2.4.7 Vyhodnocení a reportování výsledků 11

3 Aplikace a infrastruktura **12**
3.1 Popis aplikace 12
3.2 Popis testovací infrastruktury 14
3.2.1 Servery 14
3.2.2 Nástroje pro sledování 15
3.2.3 Kontinuální integrace 16
3.2.4 Verzování 16

4 Výběr nástroje **17**
4.1 Placené nástroje 17
4.2 Open-source nástroje 18

4.3 Přehled nástrojů 18
4.4 Porovnání nástrojů 19
4.4.1 Gatling 20
4.4.2 Apache JMeter 22
4.4.3 The Grinder 23
4.5 Multikriteriální hodnocení 25
4.6 Výběr vhodného nástroje 25
4.7 Shrnutí výběru 26

5 Návrh testů 27
5.1 Ověřovací test 28
5.2 Určení hranice uživatelů 28
5.3 Porovnání různých dokumentů 28
5.4 Navyšování zátěže 29
5.5 Hodinový zátěžový test 29
5.6 Počet dokumentů 29

6 Příprava prostředí 30
6.1 Generování zátěže 30
6.2 Aplikace 30

7 Vytvoření testů 31
7.1 Vytvoření požadavků 31
7.2 Rozdělení na fragmenty 32
7.3 Uživatelé 33
7.4 Konfigurace testu 33
7.5 Jednotlivé testy 34
7.5.1 Ověřovací test 35
7.5.2 Určení hranice uživatelů 35
7.5.3 Navyšování zátěže 36
7.5.4 Porovnání různých dokumentů 36
7.5.5 Hodinový zátěžový test 37
7.5.6 Počet dokumentů 38

8 Průběh testování 39
8.1 Ověřovací test 39
8.2 Určení hranice uživatelů 39
8.2.1 Zkouška testu 39
8.2.2 Provedení testů 40
8.2.3 Vyhodnocení testů 40
8.2.4 Výkonnější server 41

8.3 Navyšování zátěže 41
8.3.1 Provedení testů 41
8.3.2 Vyhodnocení testu 41
8.3.3 Identifikace problému 44
8.3.4 Výkonnější server 44
8.4 Porovnání různých dokumentů 44
8.4.1 Provedení testů 44
8.4.2 Vyhodnocení testu 44
8.4.3 Výkonnější server 44
8.5 Hodinový zátěžový test 45
8.5.1 Provedení testů 45
8.5.2 Vyhodnocení 45
8.6 Počet dokumentů za minutu 46
8.6.1 Provedení testu 46
8.6.2 Vyhodnocení 46
8.6.3 Výkonnější server 47

9 Kontinuální integrace **48**
9.1 Typy testů 48
9.1.1 Ověřovací test 48
9.1.2 Práce s vytvořeným dokumentem 48
9.1.3 Vytváření dokumentu 48
9.2 Integrace 49
9.2.1 Spouštění testů 49
9.2.2 Kontrola limitů 49
9.2.3 Výsledný skript 50
9.2.4 Report výsledků 50

10 Závěr **53**

Literatura **55**

A Příloha k nástrojům **57**
A.1 Gatling 57
A.1.1 Skript 57
A.1.2 Report 59
A.1.3 Rekorder 61
A.2 The Grinder 62
A.2.1 Skript 62
A.2.2 Report 63
A.3 JMeter 65

A.3.1 Skript . . . 65
A.3.2 Report . . . 66
A.3.3 Rekorder . . . 67

B Testy v JMeteru **68**
B.1 Vytvořené testy . . . 68
B.1.1 Složka Zakladni_testy . . . 68
B.1.2 Složka Jenkins_testy . . . 68
B.1.3 Spuštění testů . . . 68
B.2 Reporty . . . 69

1 Úvod

Vzhledem k velkému rozšíření internetu a jeho vysoké rychlosti se webové aplikace staly velmi populárními. Oproti desktopovým aplikacím umístěným na lokálních počítačích, webové aplikace využívají webový prohlížeč, pomocí kterého komunikují se serverem. Server může převzít většinu logických operací. Prohlížeč slouží pro ovládání aplikace a umožňuje odesílat požadavky na server. Použití prohlížeče pro interakci s aplikací přináší řadu výhod. Mezi největší výhody patří nezávislost aplikace na operačním systému, snadné provádění aktualizací a sdílení dat mezi uživateli. Nevýhodou je např. nutné připojení k internetu s dostatečnou rychlostí, náročnější zabezpečení a se stoupající oblibou aplikace i velmi její proměnlivá zátěž.

Webové aplikace postupem času nabývaly na komplexnosti a počet uživatelů se zvyšoval. Bylo nutné zajistit spolehlivost programů a z toho důvodu se stalo testování standardem ve vývojovém cyklu, jak tomu je u jiných softwarových aplikací. Ukazuje se, že jedním z velmi důležitých typů testování jsou zátěžové testy. Tyto testy pomáhají odhalit nevhodné nebo dokonce nefunkční chování aplikace v době, kdy na server přistupuje zároveň velké množství uživatelů. V případě menších aplikací se jedná o desítky či stovky uživatelů, u rozsáhlejších systémů se může jednat až o tisíce až stovky tisíc uživatelů.

Cílem práce je provést zátěžové testování HTML5 aplikace pomocí vhodných již existujících prostředků. Aplikace je vyvíjena firmou KadeL Data servis, která se zabývá tvorbou informačních systémů. Jelikož firma doposud prováděla pouze jednoduché zátěžové testování, bude nejdříve potřeba vybrat vhodný nástroj na základě firmou stanovených kriterií. Dalším krokem bude návrh a implementace sady zátěžových testů pomocí vybraného nástroje tak, aby co nejvíce zohledňovala specifika testované aplikace. Takto vytvořené sady testů se provedou při různých konfiguracích a podmínkách aplikace a prostředí. Smyslem práce je tedy zavést do testování aplikace nový kvalitativní rozměr.

2 Zátěžové testy

2.1 Přínos zátěžových testů

Zátěžové testování má za úkol otestovat chování aplikace v reálném provozu. Při uvolnění aplikace bez těchto testů může nastat mnoho problémů, které pomocí pouze funkčního otestování nelze odhalit. Hlavním problémem bývá zvyšující se doba odezvy, což má za následek snížení funkčnosti až nefunkčnost aplikace. V takovém případě hrozí úbytek klientů a finanční ztráta mnohonásobně převyšující náklady, které by bylo nutné vynaložit na zátěžové testování.

Testování zatížení umožňuje zejména:

- ověření dostatečného výkonu hardwaru,
- stanovení maximálního možného počtu přistupujících uživatelů za podmínky přijatelné doby odezvy
- otestování chování aplikace při přetížení,
- nalezení slabých míst aplikace při zátěži,
- porovnání výkonu s předchozí verzí aplikace.

2.2 Měření zátěže

Při testování nás zajímají dva typy metrik. První typ je orientovaný na provoz (service-oriented) a zahrnuje dostupnost a dobu odezvy. Udává kvalitu služby vzhledem ke koncovému uživateli. Druhý typ metriky se zaměřuje na efektivitu, se kterou aplikace využívá prostředí. Tento typ zahrnuje míru propustnosti a využití zdrojů [1].

2.2.1 Doba odezvy

Doba odezvy je čas, který se počítá od odeslání požadavku na server do té doby, než server zpracuje žádost a klient přijme odpověď. Očekávaný čas odezvy se liší podle charakteru požadavku. Pokud se jedná o požadavek pouze pro zobrazení jednoduché stránky, doba odezvy je několik milisekund V okamžiku, kdy se jedná například o zpracování rozsáhlého formuláře, může

být doba odezvy i několik sekund. Mimo to je tato doba závislá i na rychlosti spojení.

2.2.2 Dostupnost

Dostupnost se vyjadřuje v procentech dostupného času. Například pokud je aplikace 24/7 během týdne nedostupná 5 hodin, tak potom dostupnost je 163/168 (tedy 97 %). Nízká hodnota dostupnosti může velmi negativně ovlivnit pohled uživatele na kvalitu aplikace.

2.2.3 Propustnost

Propustnost indikuje počet přístupů za určitý časový úsek. Jako příklad můžeme uvést počet zobrazených stránek za sekundu.

2.2.4 Využití zdrojů

Využití zdrojů je procentuální hodnota, která udává využití různých částí počítačového systému. Zaměřujeme se hlavně na využití procesoru, paměti a disku. Procesor by nikdy neměl být vytížený na sto procent. V takovém případě dochází ke zhoršení propustnosti. Nejvhodnější se jeví dlouhodobé osmdesáti-procentní vytížení. Sledování alokace paměti je vhodné pro ověření, zda nedochází k úniku paměti. Únik může nastat při neuvolnění alokované paměti, která se již nepoužívá. Další sledovanou komponentou může být například disk [2].

2.3 Typy zátěžových testů

Testy můžeme rozdělit do několika kategorií podle účelu, doby trvání a nebo míry zatížení. Každá z kategorií se zaměřuje na specifický problém a ověřuje funkčnost aplikace při rozdílných průbězích zátěže. V ldostupných zdrojích můžeme narazit na různé kategorie, které se liší převážně pouze názvem, ale účel testů je stejný. Následující podkapitoly obsahují základní rozdělení testů [3].

2.3.1 Výkonnostní testy

Testy se zaměřují na chování aplikace za podmínek, které simulují její reálné používání. Průběh testů napodobuje různé uživatelské přístupy, v takové

míře zatížení, jakou očekáváme při skutečném běhu aplikace. Sleduje se, jestli aplikace reaguje s očekávanou rychlostí a propustností.

Simulování reálné zátěže ověřuje optimalizaci aplikace a dostatečný výkon prostředí v němž běží. V případě špatných výsledků je nutné se rozhodnout, zda je možné a vyplatí se aplikaci optimalizovat popř. investovat finance do navýšení výkonu hardwaru.

Výkonnostní testy se můžou využít i pro regresní testování. Toto testování se používá k porovnání předchozí verze programu s verzí, u které byla přidána nebo upravena funkcionalita. Lze tak zjistit vliv úpravy na výkon aplikace.

2.3.2 Stress testy

Princip testů spočívá ve vytvoření takové zátěže, jenž způsobí přetížení serveru a aplikace přestává fungovat. Otestujeme tak chování aplikace za stavu, kdy k ní přistupuje příliš velké množství uživatelů zároveň a nebo se stala terčem možného DDoS útoku. Pomocí těchto testů lze předejít problémům při opětovném spuštění nebo ztrátě dat při přetížení serveru.

Další využití těchto testů je stanovení maximálního počtu uživatelů, kteří mohou současně přistupovat k aplikaci a vykonávat určité operace. Lze tak stanovit hranici, kdy bude nezbytné navýšit výkon hardwaru, na němž běží server.

2.3.3 Spike testy

Testy se snaží napodobit zátěž vznikající v hlavní špičce. Průběh testu spočívá ve velmi rychlém navýšení běžné zátěže na maximální očekávanou hranici. Zátěž setrvá na této hranici po relativně krátkou dobu a klesne opět na běžnou hranici. Ověříme tak chování a stabilitu aplikace při nárazové zátěži.

2.3.4 Smoke testy

Smoke testy jsou jednoduché a rychlé testy s velmi nízkou zátěží zaměřené na základní funkcionalitu. Tento typ testů je vhodný provádět před zahájením výše zmíněných zátěžových testů, abychom předešli testování nefunkčního buildu.

2.3.5 Testy infrastruktury

Testy se orientují na jednotlivé části infrastruktury, jako jsou webové servery, aplikační servery, databáze atd. Prověřují se jejich limity a zjišťuje se, zda některá z částí v nepřiměřené míře neomezuje chování aplikace.

2.3.6 Soak testy

Soak testy simulují předpokládanou zátěž po delší dobu. Doba testování se pohybuje v řádu hodin. Pomáhají odhalit například únik paměti a nebo zaplnění disku dočasnými soubory.

2.4 Průběh testování

Zátěžové testování se skládá z několika po sobě navazujících fází. Pro dosažení co nejlepšího otestování aplikace by neměla být opomenuta ani zanedbána žádná z těchto fází. První – analytická – část je zaměřená na analýzu testovacího prostředí, požadovaných kriterií na aplikaci a návrh testů. Další část obsahuje přípravu prostředí a realizaci testů. V poslední etapě se analyzují výsledky a reportují případné chyby [4].

2.4.1 Určení testovacího prostředí

V této části analýzy se zaměřujeme na výběr testovacího prostředí. Testovací prostředí by se mělo shodovat s prostředím, které bude sloužit pro reálný běh aplikace. To však v některých případech nemusí být realizovatelné. Očekávaný počet serverů pro běh aplikace může být velmi vysoký a vytvořit takové prostředí pouze pro testování by bylo příliš nákladné. Z toho důvodu se omezuje jejich počet, ale jejich specifikace by měly být zachovány. Lze tak snadno odvodit zatížení na jeden server a podle toho určit zatížení na celkové struktuře produkčních serverů. Další problém nastává u síťového nastavení. Vzniká při nemožnosti umístit testovací servery na místo se stejným připojením, jako je u produkčních serverů [2].

Kromě výběru testovacího prostředí se v této části určují zdroje, které lze poskytnout testovacímu týmu a vybírá se vhodný nástroj pro testování a report [4] .

2.4.2 Stanovení přijatelných kriterií

Další část analýzy se zabývá definováním přijatelných požadavků na běh aplikace. Kritéria jsou dána hranicemi metrik uvedených v kapitole 2.2. Hraniční mez doby odezvy se stanovuje tak, aby nedocházelo k nespokojenosti klienta. Můžeme brát v potaz průměrnou dobu odezvy, a nebo odezvu na specifický požadavek, například vytvoření dokumentu nesmí trvat déle než dvě sekundy. Požadovaná hranice propustnosti je dána podle předpokládaného

počtu najednou přistupujících uživatelů. Jako příklad může sloužit schopnost aplikace vytvořit 50 dokumentů za sekundu. Hranice využití zdrojů se vymezuje proto, aby nedocházelo k příliš velkému využití, které by mohlo omezit výkon aplikace, a nebo k malému využití, které by znamenalo zbytečné plýtvání prostředky.

2.4.3 Návrh testovacího plánu

Při návrhu testovacího plánu vytváříme scénáře definující průběh jednotlivých testů. Scénáře by měly být navrženy tak, aby popisovaly případy užití, které jsou reálné pro danou aplikaci. Testovací plán obsahuje scénáře zaměřené na různé typy zátěžových testů viz 2.3 s odlišnými vstupními daty, jako je například počet přistupujících uživatelů, míra a průběh zátěže.

2.4.4 Příprava testovacího prostředí

Testovací prostředí připravíme podle analýzy provedené na začátku 2.4.1. Nakonfigurujeme nástroje pro sběr dat, implementaci a generování testů. Velmi důležité je, aby při generování testů na stroji nedocházelo k nedostatku operační paměti anebo k nedostatečnému výkonu procesoru. To by mohlo ovlivnit výsledky. Z toho důvodu je nutné zajistit a ověřit dostatečný výkon počítačů generujících zátěž.

2.4.5 Implementace testů

K vytvoření testů použijeme zvolený nástroj. Testy implementujeme pomocí skriptovacího jazyka podporovaného daným nástrojem. Další způsob vytváření testů může být pomocí grafického rozhraní a nebo nahráváním požadavků z prohlížeče. Implementace testů se bude odvíjet od scénářů vytvořených v 2.4.3.

2.4.6 Provedení testů

Po předchozích přípravách můžeme přejít k provedení testů. Jako první by bylo vhodné provést smoke testy a ověřit tak funkčnost aplikace. V průběhu provádění testů je vhodné průběžné sledovat klientskou a aplikační část.

2.4.7 Vyhodnocení a reportování výsledků

Poslední částí je analýza výsledků z testování. Pro přehlednost je vhodné ze získaných dat vytvořit grafy. Některé nástroje podporují vygenerování těchto grafů. V případě, že nástroj neumožňuje generování grafů, je možné pomocí pluginů tuto možnost přidat a nebo využít nástroje určeného pro zobrazování grafů např. Graphite[1].

V analýze porovnáváme dosažené a očekávané výsledky, zjišťujeme zda dosažené hodnoty metrik jsou akceptovatelné a popisujeme nesrovnalosti, které se objevily během testování. Případné chyby, nevhodné chování a další nedostatky reportujeme. Po úpravě aplikace provedeme opětovné otestování a zjistíme tak, zda došlo ke zlepšení nebo v opačném případě ke zhoršení.

[1]https://graphiteapp.org/

3 Aplikace a infrastruktura

Před zvolením nástroje je nutné se seznámit s testovanou aplikací tak, aby se do výběru mohla zahrnout kritéria, která zamezí nevhodné volbě. Pokud by některý nástroj znemožňoval provedení některých zátěžových testů, je třeba ho vyloučit z výběru. Kromě aplikace potřebujeme znát i infrastukturu, v níž se bude provádět testování. Měly by se brát v úvahu dostupné prostředky pro vykonání testů a vhodnost nástroje do testovacího prostředí.

3.1 Popis aplikace

Firma KadeL Data servis se podílí na vývoji produktu zvaného M/TEXT. Vývoj aplikaci započala firma Kühn & Weyh před 30 lety. Za tuto dobu prošla aplikace četnými modifikacemi. Základní funkce M/TEXT, kterou je tvorba a generování dokumentů podle šablon, se však od počátku vývoje nezměnila.

Produkt umožňuje jak dávkové, tak interaktivní zpracování dat. Poslední největší a aktuální úprava aplikace spočívá právě v interaktivním zpracování dat. Pro zajištění této funkčnosti podstoupila aplikace proměnu do plnohodnotné webové aplikace jejíž webové rozhraní je nazvané mIQ. Další významnou změnu prodělal formátovač dokumentů, kvůli zcela nové reprezentaci dokumentů. Původní imperativní jazyk byl nahrazen značkovacím jazykem XML viz ukázka 3.1.

```
<Root>
 <RootPart>
  <DataDefinition>
   <ParamDef name="Depot" ref="\\Depotubersicht\Depot.datamodel"></ParamDef>
  </DataDefinition>
  <ModificationRights>
    <Allowed>
      <ModificationRight role="_EVERYONE_" operations="EDIT,INPUT,DELETE,INSERT"></
           ModificationRight>
    </Allowed>
    <Denied></Denied>
  </ModificationRights>
  <DocumentCollection>
   <Document base="Depotubersicht">
    <Section>
         <SectionPartRef uri="Bausteine/Briefpapier.model"></SectionPartRef>
            <SectionPartRef uri="Bausteine/Adresse.model">
             <Param name="BD">$Depot.Briefdaten</Param>
             <Param name="Adresse">$Depot.Adresse</Param>
         </SectionPartRef>
         <SectionPartRef uri="Bausteine/Sachbearbeiter.model">
             <Param name="SB">$Depot.Sachbearbeiter</Param>
         <Param name="BD">$Depot.Briefdaten</Param>
            <Container>
             <Style>
              <RegionRef>Body</RegionRef>
             </Style>
             <ContainerPartRef uri="Bausteine/Depotubersicht.model">
                <Param name="Depot">$Depot</Param>
             </ContainerPartRef>
```

```
            </Container>
          </SectionPartRef>
    </Section>
    </Document>
   </DocumentCollection>
  </RootPart>
  <RootPartInstance>
   <Data>
    <DataNode name="Depot">
     <DataNode name="Adresse">
      <DataNode value="string:Herr" name="Anrede"></DataNode>
      <DataNode value="string:Dr." name="Titel"></DataNode>
      <DataNode value="string:Kurt" name="Vorname"></DataNode>
      <DataNode value="string:Graf" name="Nachname"></DataNode>
      <DataNode value="string:Rosenweg 4" name="Strasse"></DataNode>
      <DataNode value="string:07740" name="Postleitzahl"></DataNode>
      <DataNode value="string:Jena" name="Ort"></DataNode>
     </DataNode>
     <DataNode name="Briefdaten">
      <DataNode value="string:987-654-D1" name="Versicherungsnummer"></DataNode>
      <DataNode value="string:Lebensversicherung" name="Versicherungsart"></DataNode>
     <DataNode name="Aktie">
      <DataNode value="string:LUFTHANSA AG VNA O.N" name="Name"></DataNode>
      <DataNode value="number:12.87" name="Kurs"></DataNode>
      <DataNode value="number:20" name="Anzahl"></DataNode>
    </DataNode>
   </Data>
  </RootPartInstance>
</Root>
```

Ukázka 3.1: Reprezentace dokumentu pomocí jazyka XML

Aplikace mIQ patří mezi tzv. Rich Internet Applications. Jedná se o webové aplikace, kde značnou část logiky přebírá klient, například webový prohlížeč. Hlavní úkolem aplikace je vytváření dokumentů z předem připravených šablon a prohlížení vytvořených dokumentů.

Testovaná aplikace využívá Java framework GWT (Google Web Toolkit) v kombinaci se značkovacím jazykem HTML5. Komunikace mezi klientem a serverem probíhá jako žádost/odpověď pomocí protokolu HTTP. Jak již bylo zmíněno, velká část logiky aplikace je prováděna v prohlížeči. Proto jsou z klientské části posílány pouze požadavky na události, které jsou v ní neřešitelné, například je nutné přistupovat k datům na serveru.

Mezi požadavky pro práci s dokumentem patří vytvoření (create), otevření (open), upravení (update), uložení (save) a export. První dvě žádosti vytváří kopii originálního dokumentu, která se uchovává v databázi po dobu používání. Server vrátí odpověď obsahující identifikační číslo kopie v hlavičce. Uživatel dále pracuje pouze s touto kopií dokumentu. Příkaz export dává pokyn serveru k převedení kopie dokumentu na tisknutelný PDF soubor a poté je tento soubor vrácen klientovi. Úprava dokumentu se provádí před exportem a nebo před uložením. Při úpravě je z klientské části spolu s požadavkem a identifikačním číslem dokumentu odesílán i pozměněný dokument, kterým se přepíše pouze kopie na serveru. Pokud bychom chtěli upravit původní dokument, je třeba zaslat požadavek na uložení, jenž tento dokument nahradí kopií.

Dokumenty se otevírají ve WYSIWYG online editoru viz obrázek 3.1 . Při žádosti o dokument resp. šablonu server vrací XML soubor s obsahem a

odkazy na další potřebné zdroje, jako jsou například fonty a jejich velikost, obrázky a popř. další dokumenty. Soubor je na straně klienta zobrazován pomocí tzv. formátovače. Formátovač zajistí zobrazení obsahu a v momentě, kdy narazí na potřebný chybějící zdroj, požádá o něj server.

S aplikací můžou pracovat pouze přihlášení uživatelé. Informace o přihlášeném uživateli se uchovávají v cookies.

Obrázek 3.1: Editor aplikace se základním dokumentem(Snímek obrazovky je uveden pouze pro celkou představu)

3.2 Popis testovací infrastruktury

Po seznámení s aplikací je nezbytné znát i dostupné prostředky využívané firmou k testování. Jedná se například o servery, které budou po celou dobu testování dostupné a nástroje s nimiž je budeme moci sledovat.

V následujích podkapitolách si vytvoříme přehled serverů a nástrojů, které můžeme během testování použít.

3.2.1 Servery

Firma může pro zátěžové testování využít několik serverů propojených v lokální síti. Počítače jsou propojené switchem a jedná se o síť s rychlostí 1 Gb/s.

Pro náš účel byly vyhraněny tři servery s rozdílnou konfigurací (viz tabulka 3.1) a operačním systémem Linux.

		Tester	Vmhost01	Consultator
Operační paměť	**Počet**	4	4	2
	Typ	DDR3	DDR3	DDR3
	Frekvence [MHz]	1333	1066	1600
	Velikost [MB]	4096	4096	4096
Procesor	**Typ**	Intel®Core™ i7-2600	Intel®Xeon® X3430	Intel®Core™ i7-3820
	Počet jader/vláken	4/8	4/4	4/8
	Frekvence	3400	2400	3600
	Benchmark[1]	8225	3376	8992
Pevný disk	**Označení**	Hitachi HDS72101	WDC WD2502 ABYS-5	ADATA SSD S511
	Typ	HDD	HDD	SSD

Tabulka 3.1: Přehled dostupných počítačů

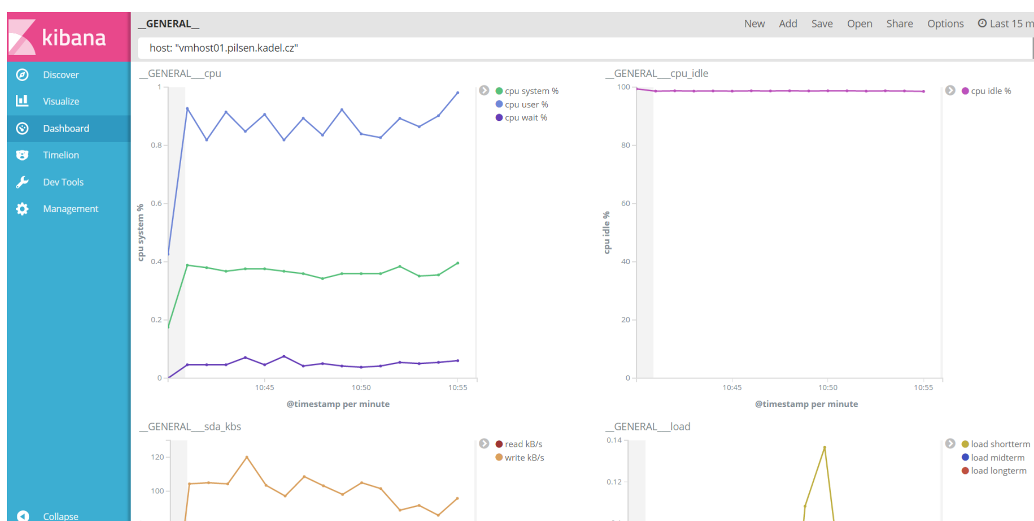
3.2.2 Nástroje pro sledování

Kibana

Testovací oddělení má rutinně zprovozněné sledování zatížení každého serveru určeného k testování. Sleduje se zatížení CPU, využití operační paměti a práce s diskem. Data ze serverů se uchovávají a dají se snadno zobrazit pomocí open-source nástroje Kibana [5] viz obrázek 3.2.

Kibana slouží pro analýzu a zobrazení dat pomocí webového prohlížeče. Program budeme moci použít během testování pro snadnou analýzu chování komponent při zátěži.

[1] https://www.cpubenchmark.net/



Obrázek 3.2: Sledování zátěže pomocí nástroje Kibana

Javamelody

Javamelody [6] je nástroj pro sledování Java aplikací. Zobrazuje statistiky a informace o aplikaci na straně serveru ohledně operací vytvářených uživateli. Tento nástroj je již integrovaný do testované aplikace a lze ho v případě potřeby snadno použít.

3.2.3 Kontinuální integrace

Firma KadeL Data servis se snaží testy co nejvíce automatizovat a k tomuto účelu využívá kontinuální integrace pomocí open-source nástroje Jenkins CI [7]. Testovací oddělení má již připravený build aplikace v tomto nástroji, tak aby bylo možné integrovat zátěžové testy. Možnost začlenit nástroj do kontinuální integrace je jedna z hlavních podmínek pro výběr nástroje.

3.2.4 Verzování

Firma pro verzování používá systém Subversion. Verzování se ve firmě provádí i v případě testů, takže se při výběru nástroje zaměříme i na tuto vlastnost.

4 Výběr nástroje

Pro zátěžové testování existuje velké množství různých nástrojů. Liší se převážně v podpoře protokolů a technologií, používaným skriptovacím jazykem, způsobem zobrazení výsledků a licencí, pod kterou jsou dostupné. Při výběru nástroje se musíme řídit finančními prostředky firmy a vhodností pro testovanou aplikaci. Pokud potřebujeme otestovat například jednu jednoduchou aplikaci, není vhodné investovat do drahého nástroje. V případě, kdy záměrem firmy by bylo rozsáhlé testování webových aplikací, se může vyplatit investovat do nástroje, který by usnadnil a tím i urychlil testování.

4.1 Placené nástroje

Při výběru nástroje se objevila řada velmi kvalitních placených nástrojů. Poskytují velmi jednoduchou implementaci, přehledné výsledky, dobrou dokumentaci a neustálou technickou podporu. Další častou vlastností placených nástrojů je velice snadné testování v cloudu. Jedná se o umístění klientské části na serverech společnosti, která tuto službu poskytuje. Jedna z takových společností je například Amazon Web Services[1]. Využitím cloudu odpadá starost s přípravou infrastruktury pro testování. Firma nemusí zbytečně investovat finance do serverů využívaných pouze pro zátěžové testování a čas potřebný pro přípravu serverů generujících zátěž. Další vlastností některých cloudů je generování zátěže z různých míst po světě, čímž lze lépe simulovat reálné použití aplikace[1]. Placené nástroje a k nim poskytované služby jsou ve většině případů velmi drahé, ale poskytují výhody umožňující velmi kvalitní a rychlé otestování webových aplikací. Poskytovalé umožňují často zdarma vyzkoušet nástroj s omezeným počtem virtuálních uživatelů, které lze použít pro simulaci zátěže během jednoho testu, a nebo s omezenou dobou pro používání.

Cena nástroje většinou není uvedena a je nutné kontaktovat firmu a domluvit se na požadavcích. Výsledná cena obvykle záleží na počtu virtuálních uživatelů, počtu testerů využívajících nástroj a na formě podpory (email, telefon). Platba za nástroj se provádí převážně paušálně každý měsíc. Firmy často uvádí bezplatnou možnost cloudového testování, ale jedná se pouze o funkci nástroje, která umožňuje jednoduché spuštění testování na serverech jiné firmy. V případě využití této služby se tedy musí počítat s dalšími

[1]https://aws.amazon.com/?nc2=h_lg

poplatky, jenž se odvíjí od délky a využití potřebného hardwaru pro provedení testu.

4.2 Open-source nástroje

Kromě placených nástrojů existuje i několik bezplatných nástrojů. Jejich množství nebylo takové jako u placených, ale i přesto je z čeho vybírat, pokud firma nechce do zátěžového testování investovat velké množství peněz. I když se jedná o nástroje volně ke stažení, umožňují plnohodnotné testování, ale jejich používání může být náročnější na čas přípravy testů, než v případě zpoplatněných nástrojů. Reporty výsledků nemusí být tak kvalitní a přehledné, ale jsou zcela dostatečné pro analýzu průběhu testování. Funkčnost nástrojů často lze rozšířit pomocí pluginů. Ty poskytují například lepší vizualizaci výsledků, rozšíření protokolů a technologií. Některé společnosti nabízí import skriptů z nástroje do svého zpoplatněného řešení, které umožňuje jednoduché spouštění testů v cloudu a lepší přehled výsledků. Za malý poplatek tak můžeme používat cloud stejně jako u placených nástrojů. Mezi takové produkty patří například Redline13[2] a BlazeMeter[3].

4.3 Přehled nástrojů

Z nalezených nástrojů bylo vybráno několik placených i neplacených produktů. Tyto produkty jsou zobrazeny v tabulce 4.1. Ke každému nástroji je uvedeno několik vlastností, mezi které patří podporované platformy, poskytované licence a způsoby vytváření skriptů testovacích případů.

[2] https://www.redline13.com/blog/
[3] https://www.blazemeter.com/

Nástroj	Platforma	Licence	Vytváření skriptu
JMeter [8]	Multiplatformní	Apache License 2.0	Groovy, pomocí GUI, BeanShell
Gatling [9]	Multiplatformní	Apache License 2.0	Scala
LoadRunner [10]	Windows, Linux	Placený	ANSI C, Import skriptů z nástrojů: Apache JMeter, NUnit, Selenium
Neoload [11]	Multiplatformní	Placený	Pomocí GUI, Javascript
LoadComplete [12]	Windows	Placený	VBScript, JScript, DelphiScript, C++ Script, C# Script, Python,VB
The Grinder [13]	Multiplatformní	BSD licence	Jython, Clojure
Tsung [14]	Linux	GNU GPL 2.0	Erlang

Tabulka 4.1: Základní přehled nástrojů

4.4 Porovnání nástrojů

Po vytvoření přehledu je nutné se omezit jen na některé nástroje. Při analýze placených nástrojů byla u všech zjištěna cena převyšující finanční prostředky firmy. Z toho důvodu se omezíme pouze na open-source nástroje. Jako další kritérium firma zadala podporu operačních systémů Windows a Linux. Z tabulky 4.1 vidíme, že kritéria splňují nástroje Gatling, The Grinder a JMeter. Všechny vyjmenované nástroje umožňují použití protokolu HTTP, který je nutný pro provedení zátěžových testů u testované aplikace. Dále jsme ověřili splnění kriterií pro použití kontinuální integrace a verzovacího systému Subversion. Všechny nástroje lze využít ke kontinuální integraci, tak i k verzování. Subversion můžeme využít u každého nástroje díky skriptovacím jazykům, kterými jsou testy napsané a s verzováním nemají problém. Dále se budeme zabývat pouze těmito nástroji a provedeme jejich analýzu pro následné sestavení multikriteriálního ohodnocení.

Analýza probíhala v první fázi na základě dostupných informací z článků a internetových fór. Některé články a přízpěvky byly však i několik let staré a porovnávaly již zastaralé verze nástrojů. Dále z článků nebylo možné po-

soudit některé vlastnosti. Z těchto důvodů bylo rozhodnuto vyzkoušet každý nástroj na hlavní funkcionalitě aplikace a tím upřesnit ohodnocení.

Pomocí nástrojů vytvoříme skripty podle jednoduchého scénáře. Scénář bude obsahovat přihlášení deseti různých uživatelů, kteří si vyberou šablonu pro dokument a následně ho vytvoří. Po vytvoření dokumentu se provede jeho uložení a uživatelé se odhlásí.

Při zkoušení jednotlivých nástrojů se zaměříme na práci s nástrojem, složitost vytváření testu, kvalitu vytvořených reportů a zpracování dokumentace. Vyzkoušíme možnost nahrávání událostí pomocí rekordérů, které poskytují všechny nástroje.

Po této analýze budeme schopni kvalifikovaně ohodnotit jednotlivé vlastnosti nástrojů a zároveň ověřit, zda s nimi můžeme bez problémů otestovat naší aplikaci.

4.4.1 Gatling

Jedná se o poměrně nový a jednoduchý open-source nástroj. Gatling [9] je založený na jazyce Scala, Akka a frameworku Netty a zaměřuje se hlavně na testování pomocí HTTP protokolu. Jako jazyk pro vytváření skriptů se používá Scala. Gatling neumožňuje rozdělení zátěže a vytvoření souhrného reportu, ale lze toho docílit pomocí skriptu popsého v dokumentaci[4]. Skript spustí na více serverech stejný testovací scénář a ze shromážděných logů vygeneruje výsledný report. V případě potíží se lze obrátit na skupinové fórum pro uživatele a nebo si předplatit podporu od vývojářů Gatlingu. Mimo toto fórum a oficiální stránky, ale neexistuje velké množství návodů a rad.

Dokumentace

Přehledná a stručná dokumentace je velkou předností používání nástroje Gatling. Obsahuje srozumitelné návody doplněné o ukázky kódu. Pomocí dokumentace bylo možné se rychle seznámit s nástrojem a bylo zde nalezeno vše potřebné.

Rekordér

Spuštění rekordéru je velmi snadné. Po spuštění se zobrazí okno s několika volbami. Po nastavení parametrů zapneme nahrávání a objeví se vizualizace

[4] http://gatling.io/docs/current/cookbook/scaling_out

s přehledem nahraných požadavků a odpovědí s obsahem hlavičky. Vygenerovaný kód pomocí rekordéru byl přehledný, ale při nahrávání dotazů v některých případech neprovedl správné escapování znaků. Kontrétně se jednalo o znak '\', který nebyl escapován a proto vygenerovaný skript nešel přeložit a musel být upraven.

Vytvoření skriptu

K vytvoření skriptu 4.1 je nutná znalost jazyku Scala. Jelikož jsem vytvářel pouze jednoduchý skript, nemohu posoudit, jak náročné by bylo si osvojit tento jazyk. Gatling se snaží nevýhodu neznalosti jazyka napravit pomocí dokumentace, která umožňuje rychlé pochopení použití Scaly pro snadnou implementaci testů.

```
class RecordedSimulation extends Simulation {
  val httpProtocol = http
        .baseURL("http://kalda-pc.pilsen.kadel.cz:8090")
        .acceptHeader("text/html,application/xhtml+xml,application/xml;q=
            0.9,*/*;q=0.8")
        .acceptEncodingHeader("gzip, deflate")
        .acceptLanguageHeader("cs,en-US;q=0.7,en;q=0.3")
        .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0)
            Gecko/20100101 Firefox/47.0")

  val headers = Map("Content-Type" -> "text/plain; charset=utf-8")

  val uri = "http://kalda-pc.pilsen.kadel.cz:8090/mtext"

  val scn = scenario("RecordedSimulation")
        .exec(http("Login")
      .post("/mtext/app/login")
          .formParam("username", (s:Session) => """mtext"""+s.userId.
              toString)
          .formParam("pwd", (s:Session) => """mtext"""+s.userId.toString))

  setUp(scn.inject(atOnceUsers(10))).protocols(httpProtocol)
}
```

Ukázka 4.1: Skriptu pro přihlášení v jazyce Scala

Report

Gatling generuje report automaticky. Naměřené hodnoty zapíše do logovacího souboru a z něj následně vygeneruje do zadaného adresáře výstup v podobě HTML souborů. Zobrazením hlavního souboru (index.html) se nám naměřená data zobrazí v tabulkách a velmi přehledných dynamických grafy.

4.4.2 Apache JMeter

JMeter [8] je open-source projekt neziskové organizace Apache Software Foundation. Nástroj je vyvíjen od roku 1999 a tento vývoj pokračuje až dodnes. Jedná se o velmi komplexní nástroj pro zátěžové testování s velkou podporou protokolů, serverů a dalších technologií. Implementaci testů lze provést pomocí skriptovacích jazyků Groovy a BeanShell. Další možností, jak lze vytvořit testy, je pomocí GUI, které umožňuje snadnou a přehlednou implementaci testovacího skriptu bez znalosti skriptovacího jazyka. Výsledný skript se z grafického rozhraní ukládá do XML souboru. Funkcionalitu nástroje můžeme rozšířit pomocí velkého množství pluginů, které jsou volně dostupné. Mezi další vhodnou vlastnost JMeteru patří možnost rozdělení generování zátěže na více serverů a vytvoření souhrného reportu. Velké rozšíření nástroje vedlo ke vzniku mnoha návodů a fór i mimo oficiální stránky.

Dokumentace

Dokumentace je přehledná a velmi obsáhlá. Je vhodně doplněná ukázkami příkazů a příklady, které jsou formou obrázků s dostatečným popisem všech atributů.

Rekordér

Zprovoznění rekordéru je popsané na stránkách nástroje[5]. Podobných návodů se vyskytovalo na internetu několik, ale pro spuštění rekordéru byl dostatečný návod od Apache. Nahrávání probíhá pomocí GUI. Během nahrávání si můžeme zobrazit přenesená data. Filtrované žádosti se ukládají v kontroleru pro nahrávání a dále je možné tyto žádosti použít pro spuštění testů. Rekordér u JMeteru jako jediný nahrál všechny žádosti bez problémů a tak je nebylo třeba opravovat.

Vytvoření skriptu

Jak bylo řečeno v popisu nástroje, skript můžeme vytvořit pomocí skriptovacích jazyků a nebo pomocí GUI. Dokumentace je zaměřená hlavně na práci v grafickém rozhraní a bylo rozhodnuto použít toto rozhraní k implementaci zkušebního testu. Velká výhoda byla shledána v přehlednosti a možnosti vytvořit skript bez znalosti skriptovacího jazyka viz ukázka 4.2.

```
<HTTPSamplerProxy guiclass="HttpTestSampleGui" testclass="HTTPSamplerProxy"
    testname="Login" enabled="true">
```

[5] http://jmeter.apache.org/usermanual/jmeter_proxy_step_by_step.pdf

```
  <elementProp name="HTTPsampler.Arguments" elementType="Arguments"
      guiclass="HTTPArgumentsPanel" testclass="Arguments" enabled="true">
      <collectionProp name="Arguments.arguments">
        <elementProp name="username" elementType="HTTPArgument">
          <boolProp name="HTTPArgument.always_encode">false</boolProp>
          <stringProp name="Argument.name">username</stringProp>
          <stringProp name="Argument.value">mtext${__threadNum}</stringProp
              >
          <stringProp name="Argument.metadata">=</stringProp>
          <boolProp name="HTTPArgument.use_equals">true</boolProp>
        </elementProp>
           <elementProp name="pwd" elementType="HTTPArgument">
           <boolProp name="HTTPArgument.always_encode">false</boolProp>
           <stringProp name="Argument.name">pwd</stringProp>
           <stringProp name="Argument.value">mtext${__threadNum}</
               stringProp>
           <stringProp name="Argument.metadata">=</stringProp>
           <boolProp name="HTTPArgument.use_equals">true</boolProp>
        </elementProp>
      </collectionProp>
        </elementProp>
</HTTPSamplerProxy>
```

Ukázka 4.2: Skriptu pro přihlášení vygenerovaný z JMeteru

Report

Pro vytvoření vizualizace výsledků bylo nutné nastavit konfiguraci. Tento postup je popsaný v dokumentaci a tak zprovoznění reportu nebylo těžké. Výsledkem jsou HTML soubory obsahující přehledné tabulky a interaktivní grafy. Výstup je srovnatelný s dříve uvedeným nástrojem Gatling. Report můžeme vyvtvořit i použitím některého pluginu a nebo pomocí placených vizualizérů.

4.4.3 The Grinder

The Grinder [13] je open-source nástroj napsaný v Javě. Implementaci testů můžeme provést v jazycích Java, Jython nebo Clojure. Z těchto jazyků se používá hlavně Jython, který je implementací programovacího jazyka Python do prostředí jazyka Java. Nástroj již není ve vývoji a jeho poslední verze byla vydána v roce 2012. Podobně jako JMeter také umožňuje generovat zátěž z více serverů a následně vytvořit souhrný report.

Dokumentace

Dokumentace je doplněná o příklady kódu a rozsáhlé FAQ, které bylo při implementaci užitečné. I přes to se dokumentace jevila nepřehledná a práce s ní nebyla tak příjemná, jako u předchozích nástrojů.

Rekordér

Během nahrávání můžeme přidávat pouze komentáře k požadavkům, ale není možné si je prohlédnout, jako to bylo možné u předchozích dvou nástrojů.

Vygenerovaný skript nebyl příliš přehledný a byl problém v případě, kdy žádost obsahovala cestu s escapovanými zpětnými lomítky k dokumentu. Místo znaku '%' rekordér zapsal nesprávný znak 'F'.

Vytvoření skriptu

K vytvoření testovacího skriptu je nutné umět programovací jazyk Python viz ukázka 4.3. I přes velmi malé zkušenosti s tímto jazykem se povedlo upravit vygenerovaný skript, tak aby odpovídal zkušebnímu scénáři.

```
def createRequest(test, url, headers=None):
  request = HTTPRequest(url=url)
  test.record(request, HTTPRequest.getHttpMethodFilter())

  return request

headers= \
  [ NVPair('Accept', 'text/html,application/xhtml+xml,'
    +'application/xml;q=0.9,*/*;q=0.8')]
url = 'http://kalda-pc.pilsen.kadel.cz:8090'

request = createRequest(Test(1, 'Login'), url, headers)

class TestRunner:
  def page(self):
    result = request101.POST('/mtext/app/login',
      ( NVPair('username', "mtext"+`self.userID`),
        NVPair('pwd', "mtext"+`self.userID`)))

    return result

  def __call__(self):
    self.userID = grinder.threadNumber
    self.page1()

Test(1, 'Login').record(TestRunner.page)
```

Ukázka 4.3: Skript pro přihlášení v The Grinderu

Report

The Grinder generuje pouze logovací soubor s naměřenými hodnotami. Pokud chceme vizualizaci výsledku, budeme muset stáhnout některý z pluginů, jako je například Ground report nebo Grinder Analyzer. Ground report je složitější na zprovoznění a ke svému běhu potřebuje Postgres databázi, ale výstupem je velký počet přehledných grafů a tabulek. Při analýze byl použit druhý zmíněný plugin Grinder Analyzer. Zprovoznění tohoto nástroje

nebylo složité, ale výstupem byly pouze grafy s dobou odpovědi a počtem transakcí za sekundu. Dále byla v reportu zobrazena tabulka se žádostmi a k nim přiřazené hodnoty.

4.5 Multikriteriální hodnocení

Multikriteriální hodnocení bylo vytvořeno na základě provedené analýzy a doplňujících informací z dostupných materiálů, tak aby bylo možné následně vybrat nejvhodnější nástroj. Tabulka hodnocení 4.2 obsahuje vybrané vlastnosti nástrojů a k nim přiřazené body podle míry, s jakou je vlastnost splněná. Body budeme přiřazovat od nuly do tří (čím větší počet bodů, tím lepší ohodnocení).

		JMeter		**Gatling**		**The Grinder**	
Podpora protokolů / technologií		Velká	2	Malá	1	Malá	1
Rozdělení generované zátěže		Podporuje	2	Nepodporuje, ale je možné provést	1	Podporuje	2
Dokumentace	**Přehlednost**	Dobrá	2	Velmi dobrá	3	Horší	1
	Názornost/ Obsah	Velmi dobrý	3	Velmi dobrý	3	Dobrý	2
Vizualizace výsledků	**Přehlednost**	Velmi dobrá	3	Velmi dobrá	3	Dobrá	2
	Interaktivita	Dobrá	2	Velmi dobrá	3	Není	0
Nahrávání akcí	**Prohlížení nahr. akcí**	Ano	1	Ano	1	Ne	0
	Korektnost	Bez chyby	2	Chyby v escapování	0	Chyby v escapování	0
Znalost prog. jazyka		Ne	1	Ano	0	Ano	0
Zdroje mimo oficiální stránky (např. fóra a návody)		Velké množství	2	Malé množství	1	Malé množství	1

Tabulka 4.2: Multikriteriální hodnocení

4.6 Výběr vhodného nástroje

Pro výběr nástroje není vhodné vycházet pouze z předchozí tabulky, protože každá firma může mít rozdílné požadavky a jednotlivé vlastnosti pro ni můžou mít různé priority. Z toho důvodu přiřadíme každé vlastnosti prioritu od 1 do 10 viz tabulka 4.3 (čím vyšší hodnota, tím důležitější vlastnost). Následně sečteme sloupec s prioritami a kažkou prioritu vydělíme tímto součtem a dostaneme tak váhy jednotlivých vlastností. Váhy vynásobíme s počtem bodů z tabulky 4.2 a hodnoty ve sloupcích příslušících nástrojům

sečteme. Sloupec s nejvyšší hodnotou bude označovat nejvhodnější nástroj pro zátěžové testování ve firmě.

Priority vlastností byly prodiskutovány s testovacím oddělením a stanoveny tak, aby zvolený nástroj co nejvíce odpovídal požadavkům jak pro aplikaci, tak pro firmu.

<table>
<tr><th colspan="2"></th><th>Priorita</th><th>Váha</th><th>JMeter</th><th>Gatling</th><th>The Grinder</th></tr>
<tr><td colspan="2">Podpora protokolů / technologií</td><td>4</td><td>0,07</td><td>0,14</td><td>0,07</td><td>0,07</td></tr>
<tr><td colspan="2">Rozdělení generované zátěže</td><td>7</td><td>0,11</td><td>0,22</td><td>0,11</td><td>0,22</td></tr>
<tr><td rowspan="2">Dokumentace</td><td>Přehlednost</td><td>7</td><td>0,11</td><td>0,22</td><td>0,33</td><td>0,11</td></tr>
<tr><td>Názornost/ Obsah</td><td>8</td><td>0,13</td><td>0,39</td><td>0,39</td><td>0,26</td></tr>
<tr><td rowspan="2">Vizualizace výsledků</td><td>Přehlednost</td><td>7</td><td>0,11</td><td>0,33</td><td>0,33</td><td>0,22</td></tr>
<tr><td>Interaktivita</td><td>3</td><td>0,05</td><td>0,10</td><td>0,15</td><td>0</td></tr>
<tr><td rowspan="2">Nahrávání akcí</td><td>Prohlížení nahr. akcí</td><td>7</td><td>0,11</td><td>0,11</td><td>0,11</td><td>0</td></tr>
<tr><td>Korektnost</td><td>10</td><td>0,16</td><td>0,16</td><td>0</td><td>0</td></tr>
<tr><td colspan="2">Znalost prog. jazyka</td><td>3</td><td>0,05</td><td>0,05</td><td>0</td><td>0</td></tr>
<tr><td colspan="2">Zdroje mimo oficiální stránky (např. fóra a návody)</td><td>6</td><td>0,10</td><td>0,20</td><td>0,10</td><td>0,10</td></tr>
<tr><td colspan="2">Součet</td><td>62</td><td></td><td>1,92</td><td>1,59</td><td>0,98</td></tr>
</table>

Tabulka 4.3: Ohodnocení nástrojů

4.7 Shrnutí výběru

V porovnání nástrojů dopadl nejhůře The Grinder. Tato skutečnost byla vidět už z tabulky 4.2 s multikriteriálním hodnocením, kde byl ve většině srovnávaných položek horší než zbylé dva nástroje.

Gatling skončil na druhém místě i přes velmi dobrou dokumentaci a report výsledků. Jeho slabší stránky se ukázaly např. v podpoře protokolů anebo v rozdělení generování zátěže na více serverů. Další problém byl nalezen v chybném escapování některých znaků u nahrávání.

Z vybraných nástrojů byl tedy JMeter vyhodnocen jako nejlepší prostředek pro provedení zátěžových testů u aplikace. Splnil všechna kritéria a v závěrečném porovnání získal největší počet bodů. Po diskuzi s vedoucím testovacího oddělení bylo rozhodnuto respektovat výsledek analýzy a pro zátěžové testování použít JMeter.

5 Návrh testů

Před implementací si vytvoříme scénáře a navrhneme jednotlivé testy. K jejich správnému navržení byla nutná rozsáhlá konzultace s testovacím oddělením, které má zkušenosti s již plně funkční desktopovou verzí aplikace. Konzultace nad návrhem byla posléze vedena i s vývojáři, kteří měli také podnětné připomínky a poskytli další doplňující informace.

V testech se zaměříme nejenom na chování aplikace při zátěži, ale i na nalezení důležitých hodnot, jako je například hraniční počet uživatelů anebo počet vytvořených dokumentů za určitou dobu.

V testech se budou používat dva základní scénáře. Oba typy scénářů budeme pro každý test různě upravovat. Například přidáme intervaly mezi jednotlivé akce anebo se některé akce v průběhu testu u každého uživatele provedou vícekrát.

První scénář je zaměřený na vytvoření nového dokumentu pomocí šablony a vypadá následovně:

1. Přihlášení uživatele
2. Vytvoření dokumentu podle šablony
3. Uložení dokumentu
4. Export dokumentu
5. Uzavření dokumentu
6. Odhlášení uživatele

Ve **druhém scénáři** bude pracovat s již vytvořeným dokumentem:

1. Přihlášení uživatele
2. Otevření dokumentu
3. Uzavření dokumentu
4. Odhlášení uživatele

V následujících podkapitolách navrhneme testy na základě těchto scénářů.

5.1 Ověřovací test

Test bude sloužit pro základní ověření funkčnosti všech akcí, které jsou zmíněné v obou scénářích. Jedná se tedy o smoke test. Tento test je vhodné provádět před spuštěním dalších testů, v případě kdy nastane změna v implementaci aplikace. Ověříme tak, že nedošlo k modifikace požadavků a jejich funkčnosti.

5.2 Určení hranice uživatelů

Scénář se zaměřuje na určení hraničního množství uživatelů, kteří začnou zároveň vykonávat události podle druhého scénáře. Test má simulovat začátek pracovního procesu, kdy se do systému přihlásí v jednu chvíli velký počet uživatelů a provede operaci pro otevření dokumentu. Hranice počtu uživatelů bude stanovená tak, že doba pro přihlášení u každého z uživatelů nesmí přesáhnout dvě sekundy a otevření dokumentu deset sekund. Časy jsou stanovené tak, aby chování aplikace bylo pro uživatele ještě přijatelné.

Začneme u velkého počtu uživatelů a postupně jej budeme snižovat, dokud časy odpovědí nebudou pod výše stanovenou hranicí. Zároveň i zjistíme, jak se aplikace bude chovat při hodně velké a nárazové zátěži.

5.3 Porovnání různých dokumentů

Další testování bude zaměřené na vytvoření dokumentů podle rozdílných šablon. Budeme tedy používat první scénář. Máme k dispozici čtyři různé dokumenty, které se liší svojí velikostí a také počtem požadovaných zdrojů. Zaměříme se na časy odpovědí, které jsou závislé na odlišné struktuře dokumentů. Mezi tyto požadavky patří update, save (již popsány v kapitole 3.1) a množina zdrojů označená jako resources.
Budou využívány následující čtyři typy:

Basic – 1 strana, 3 zdroje

Single – 3 strany, 17 zdrojů

Four – 20 stran, 3 zdroje

Many – 200 stran, 3 zdroje

5.4 Navyšování zátěže

Pro zjištění, jak se bude aplikace chovat při zvyšujícím se počtu uživatelů, budeme pracovat se scénářem pro vytvoření dokumentu ze šablony. Provedeme několik testů, u nichž budeme postupně navyšovat počet uživatelů, kteří začnou vykonávat scénář ve stejnou chvíli. Z výsledných hodnot vytvoříme graf a určíme tak závislost počtu uživatelů na době odpovědi u jednotlivých požadavků.

5.5 Hodinový zátěžový test

Pro ověření stability aplikace zatížíme procesor přibližně na 80 % po dobu jedné hodiny. Uživatelé se na začátku přihlásí a následně budou opakovaně provádět body 2–5 z prvního scénáře. Mezi každou akcí budou časové prodlevy kolem 10 sekund. Prodlevy jsou zde, abychom mohli použít větší počet uživatelů, kteří budou vykonávat scénář. Pro nalezení vhodného počtu uživatelů budeme provádět kratší testy s dobou běhu pouze několik minut a měnit jejich počet, tak aby se vytížení procesoru pohybovalo kolem stanovené hranice. Po nalezení hranice provedeme následný test o délce jedné hodiny.

5.6 Počet dokumentů

Důležitou informací o aplikaci, kterou potřebujeme znát, je maximální možný počet dokumentů vygenerovaných za určitou dobu, tak aby v průběhu testu byly časy odezev optimální pro používání aplikace. Tato hranici byla stanovena na pět sekund pro všechny akce.

Test se bude velmi podobat testu předchozímu (viz 5.5). Rozdíl bude pouze v délce času běhu, která bude kratší a test nebude obsahovat žádné časové prodlevy. Jako první provedeme kratší testy a najdeme takový počet uživatelů, při němž budou všechny délky akcí menší než stanovený limit pěti sekund.

6 Příprava prostředí

6.1 Generování zátěže

Testy můžeme naimplementovat na lokálním počítačí. Pro generování zátěže ale potřebujeme stabilní server bez vytížení od ostatních programů. Proto je nutné JMeter přesunout na server, který bude v době provádění zátěžových testů sloužit pouze pro vytváření zátěže. K tomuto účelu byl vybrán server Tester popsaný v části 3.2.1. Na server pouze přesuneme z lokálního počítače celou složku s JMeterem a vytvořené testy. Následně se ujistíme o bezproblémové funkčnosti JMeteru spuštěním např. smoke testu.

6.2 Aplikace

Aplikace byla umístěna na zbylé dva servery (Vmhost01 a výkonnější Consultator). Do aplikací bylo vytvořeno 10 000 uživatelů s pojmenováním 000000 až 009999 s heslem mtext a byl vygenerován CSV soubor, jenž obsahoval zmíněné přihlašovací údaje. CSV soubor byl umístěn k testům na serveru pro generování zátěže.

Dále bylo potřeba vygenerovat pro každého uživatele dokument, který bude otevírán v testech. Jelikož přidání dokumentu všem uživatelům přímo v testované aplikaci je složité, vytvoříme si je pomocí jednoduchého skriptu v JMeteru.

Před každým testováním bude potřeba provést restarování aplikačního serveru a obnovit původní databázi. Z toho důvodu bylo požádáno testovací oddělení o vytvoření těchto skriptů. Byly vytvořeny skripty pro řádné ukončení aplikačního serveru, jeho opětovné nastartování a smazání všech logů. Dále byl vytvořen skript pro zálohu databáze a nahrání zálohy. Následně jsme provedli zálohu databáze. Vytvořenou zálohu budeme používat pro uvedení databáze do původního stavu před každým testováním.

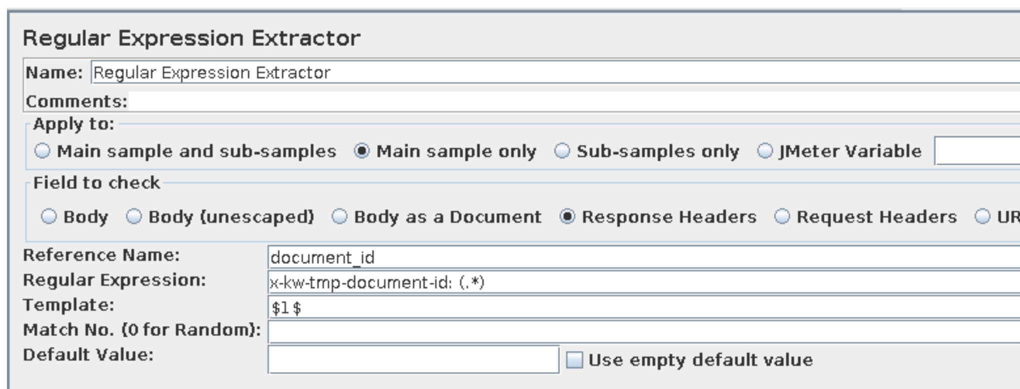
7 Vytvoření testů

Po návrhu testů přejdeme k jejich implementaci za pomocí vybraného nástroje JMeter. Při implementaci se musíme zaměřit na opakovatelnost testů, která by mohla být narušena například při práci s dynamickými daty získanými ze serveru. Dále kdybychom chtěli změnit požadavek vyskytující se ve více testech, je vhodné tento požadavek anebo množinu požadavků vytvořit jako modul, který budou testy sdílet. Další výhodou tohoto přístupu je snadná tvorba nových testů, kdy pouze skládáme již vytvořené moduly dohromady.

7.1 Vytvoření požadavků

V první řadě je třeba vytvořit všechny požadavky potřebné k provedení testů podle navržených scénářů. Žádosti nahrajeme z webového prohlížeče pomocí rekordéru JMeteru. Nahrajeme akce z prvního a následně i z druhého scénáře.

Po nahrání je nutné upravity skript tak, aby byl test opakovatelný, jelikož se pracuje s dynamickým identifikačním názvem otevíraného dokumentu resp. šablony. Jak bylo zmíněno v popisu aplikace 3.1, odpověď na požadavek pro vytvoření dočasného dokumentu vrací v hlavičce odpovědi jeho indentifikátor. Tento identifikátor se následně používá u všech žádostí pro práci s vytvořeným dokumentem. Indentifikátor dokumentu získáme pomocí tzv. `Regular Expression Extractor`. Ten slouží pro získání informací z příchozí odpovědi. V našem případě se tedy jedná o identifikaci dokumentu v hlavičce. Získaný řetězec uložíme do proměnné viz obr. 7.1, kterou budeme dále používat.

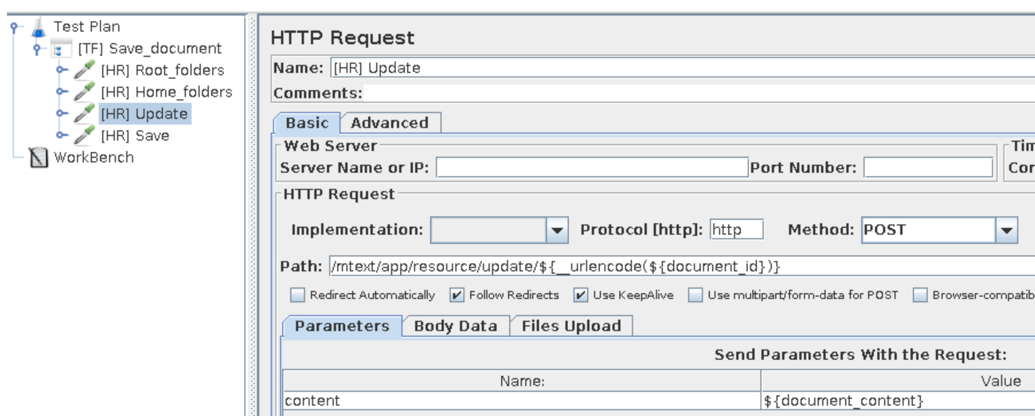


Obrázek 7.1: Získání identifikátoru z hlavičky

7.2 Rozdělení na fragmenty

V úvodu kapitoly bylo zmíněno rozdělení testů na moduly, které pak budou jednotlivé testy používat. V JMeteru k tomu slouží fragmenty (`Test Fragment - TF`). Fragment obsahuje část testu a pomocí kotroleru pro vkládání (`Include Controller - IC`) ho lze vložil do jiného testu. To nám umožní modifikovat požadavky na jednom místě bez nutnosti upravovat skript ve všech testech.

Pro každou akci zmíněnou ve scénářích jsme vytvořili fragment. Do těchto fragmentů jsme přidali nahrané a upravené požadavky (`HTTP Request - HR`) tak, že jejich seskupení dávalo dohromady jednu činnost, viz například obrázek fragmentu Save 7.2.

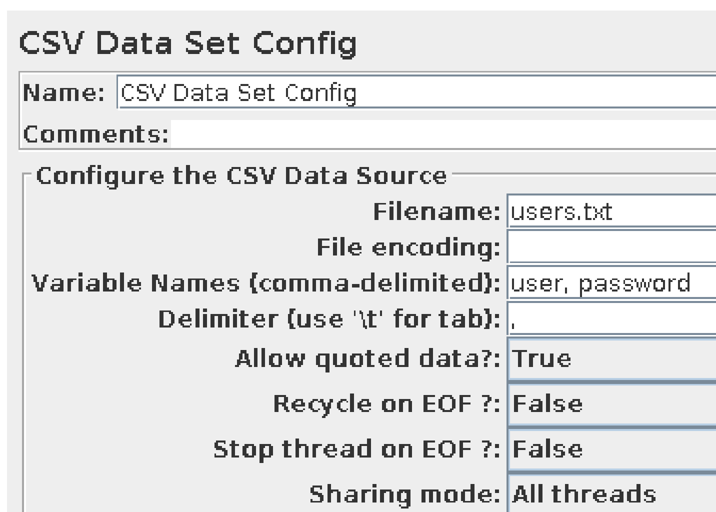


Obrázek 7.2: Fragment Save pro uložení dokumentu

7.3 Uživatelé

Nyní máme vytvořený základ pro testy. Dále musíme vyřešit přihlašování uživatelů. Prohlížeč umožňuje ukládání cookies přihlášených uživatelů a s každou žádostí je odesílá na server. Toto chování lze do testu přidat pomocí konfiguračního elementu `HTTP Cookie Manager`, který má na starost právě práci s cookies.

Ve vytvořených skriptech používáme pouze jednoho uživatele, který byl přihlášen při nahrávání, proto musíme zařídit autentizaci různých uživatelů v jednom testu. Máme k dispozici vygenerovaný CSV soubor s přihlašovacími udaji uživatelů. Pro práci se souborem použijeme v JMeteru element `CSV Data Set Config`. S jeho pomocí se pro každé vlákno představující klienta přiřadí rozdílná data, k nimž můžeme přistupovat přes proměnné nadefinované v CSV konfiguraci viz obr. 7.3.



Obrázek 7.3: Načítání uživatelů z CSV

7.4 Konfigurace testu

Při testování potřebujeme u testů snadno měnit některé hodnoty, jako je počet vláken (tj. uživatelů), doba pro spuštění všech vláken a počet opakování scénáře. Dále bude vhodná snadná změna používaného dokumentu a serveru s protokolem, kde běží aplikace.

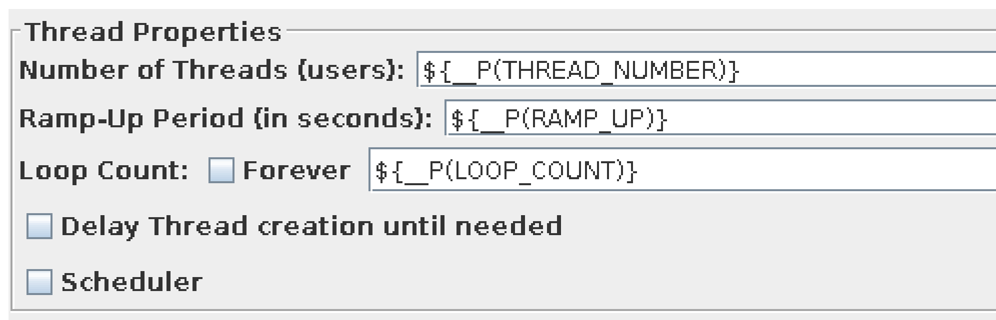
Aby se nemuselo při každé změně zapínat GUI JMeteru a nebo přepisovat vygenerovaný skript, byl zvolen jako nejvhodnější způsob zápis hodnot do konfiguračního souboru. Do konfiguračního souboru zapíšeme informace

zmíněné v předešlém odstavci. Požadovaný dokument necháme odkomentovaný a zbytek dokumentů zakomentujem (viz ukázka 7.1).

Ukázka 7.1: Obsah souboru s konfigurací

```
# Nazev a port serveru
SERVER_NAME = vmhost01
SERVER_PORT = 8080

# Pocet vlaken(uzivatelu)
THREAD_NUMBER = 1
#Doba pro spusteni vsech vlaken
RAMP_UP = 1
# Pocet vykonani testu
LOOP_COUNT = 1

# Informace o~pouzivanem dokumentu
TEMPLATE_PATH = /Home
TEMPLATE_NAME = ManySectionPartsDoc
TEMPLATE_BASE = sectionParts

# Nepouzivany dokument
#TEMPLATE_PATH=
#TEMPLATE_NAME=SingleSectionPartDoc
#TEMPLATE_BASE=sectionParts
```

K práci s konfiguračním souborem bylo potřeba přidat rozšíření `Property file reader` [15]. Po přidání tohoto konfiguračního elementu do testovacího plánu můžeme přistupovat k hodnotám následovně (viz obr. 7.4).



Obrázek 7.4: Proměnné z konfiguračního souboru

7.5 Jednotlivé testy

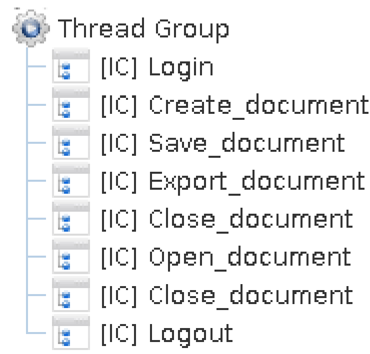
Všechny testy budou obsahovat stejné konfigurační elementy zmíněné v předchozích kapitolách. Jelikož není vhodné tuto konfiguraci dávat do samostat-

ného fragmentu, budeme ji muset přidat zvlášť do každého testu. Kompletní konfiguraci můžeme vidět na obrázku 7.5.

Obrázek 7.5: Konfigurační elementy

7.5.1 Ověřovací test

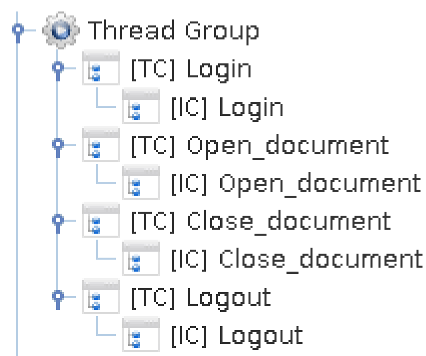
Ověřovací skript obsahuje import všech vytvořených fragmentů (viz 7.6), tak aby se ověřila funkčnost všech používaných požadavků. K tomu se využívá tzv. `Include Controller - IC`. Mezi jednotlivými akcemi nejsou vloženy žádné časové prodlevy. Chybu v testu zjistíme z vytvořeného reportu.



Obrázek 7.6: Přehled importovaných fragmentů

7.5.2 Určení hranice uživatelů

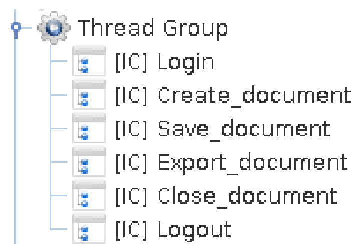
Do skriptu jsme zahrnuli pouze fragmenty akcí určených v kapitole 5.2 bez časových prodlev. Abychom dostali celkové časy jednotlivých akcí, dáme každý fragment zvlášť do transakčního kontroleru (`Transaction controller - TC`) viz 7.7. Kontroler zajistí součet časů odpovědí v jedné akci.



Obrázek 7.7: Skript pro určení hranice počtu uživatelů

7.5.3 Navyšování zátěže

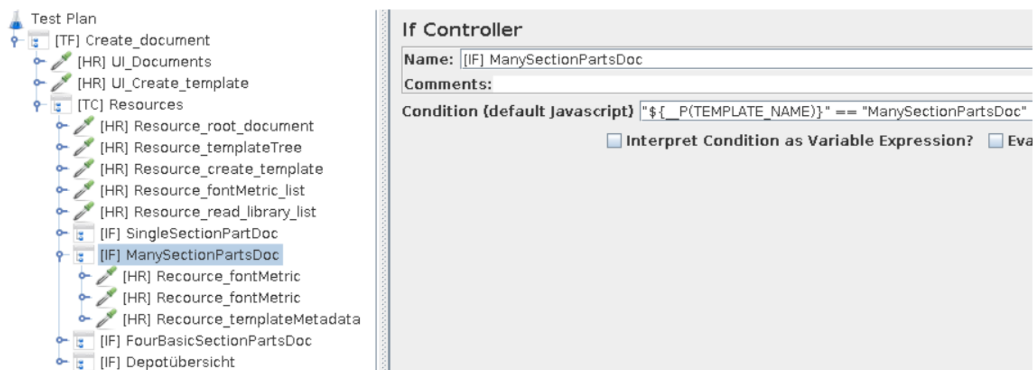
Při implementaci použijeme fragmenty s akcemi z prvního scénáře bez časových prodlev (viz obr. 7.8). Fragmenty nebudou vloženy do transakčních kontrolerů, jako tomu bylo v předchozím testu v kapitole 7.5.2, protože potřebujeme časy odpovědí na jednotlivé požadavky a nikoliv na trvání celé akce.



Obrázek 7.8: Fragmenty skriptu pro počet uživatelů

7.5.4 Porovnání různých dokumentů

Jak bylo zmíněno v kapitole 5.3, každý typ dokumentu vyžaduje rozdílné zdroje. Abychom nemuseli vždy tyto zdroje měnit ve skriptu, použijeme k tomu podmínkové kontrolery (`If controller - IF`) ve fragmentu pro vytvoření dokumentu viz 7.9. Na základě názvu typu dokumentu se rozhodneme, o které zdroje budeme žádat. Implementace bude stejná jako u testu 7.5.3.

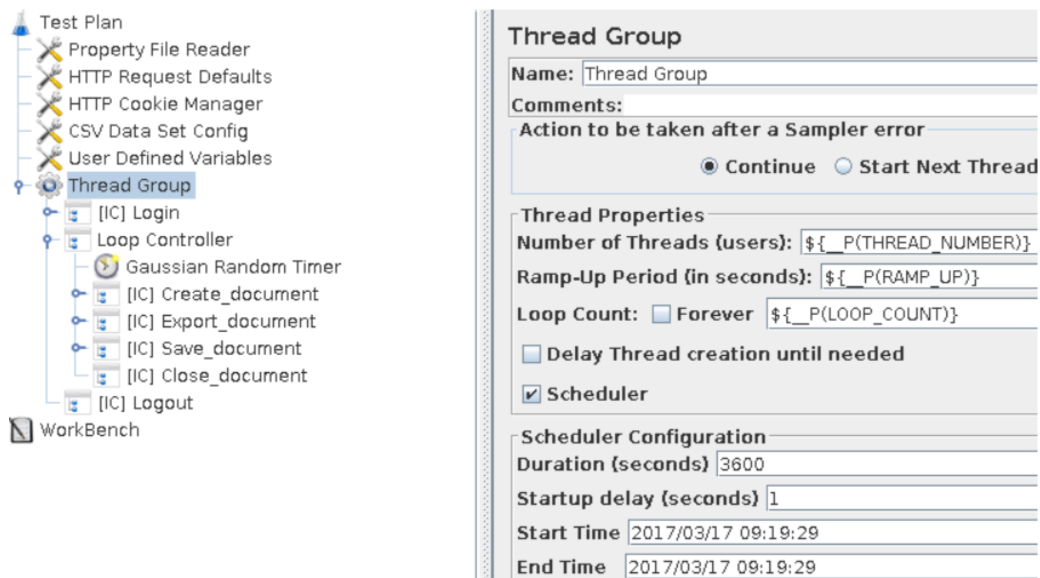


Obrázek 7.9: Fragment pro načtení dokumentu podle podmínky pro výběr typu dokumentu

7.5.5 Hodinový zátěžový test

Ve skriptu nastavíme skupině všech spouštěných vláken (`Thread Group`) dobu běhu (`Duration`) na jednu hodinu (viz obr. 7.10). Dále mezi akce přidáme časové prodlevy. Ty budou realizovány časovači. Přidělený čas se bude určovat normálním rozdělením se střední hodnotou deseti sekund a odchylkou jedné sekundy. Tím zamezíme případné nežádoucí synchronizaci požadavků a umožní nám to použít větší počet uživatelů.

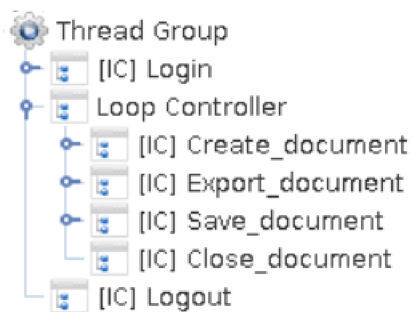
Na začátku testu přidáme fragment pro přihlášení. Následně použijeme cyklus (`Loop controller`) a do něj přidáme akce, které se mají opakovat viz kapitola 5.5. Počet opakování smyčky nastavíme na nekonečno a bude přerušen až po uplynutí doby pro ukončení testu.



Obrázek 7.10: Skript pro hodinový test

7.5.6 Počet dokumentů

Jak bylo zmíněno při návrhu testu v kapitole 5.6, tento test je velmi podobný hodinovému testu na zátěž. V implementaci pouze odstraníme časovače (viz 7.11) pro pauzy mezi akcemi a změníme dobu běhu u skupiny spouštěných vláken.



Obrázek 7.11: Skript test pro zjištění počtu dokumentů

8 Průběh testování

Po přípravě prostředí a dokončení implementace můžeme přejít k provádění testů. Před každým testem uvedeme aplikaci do počátečního stavu za použití vytvořených skriptů. Testy budeme spouštět vždy na méně výkonném serveru (Vmhost01) a následně provedeme porovnání s výkonnějším strojem (Consultator). Spouštění testů pomocí GUI je velmi náročné na operační paměť, a tak všechny testy budeme spouštět z příkazové řádky.

8.1 Ověřovací test

Test byl spuštěn jako první a pouze s deseti uživateli. Vykonal všechny akce bez chyby s výjimkou exportu. Vykonání požadavku trvalo dlouhou dobu a nakonec skončilo s chybou. Chyba byla nahlášena vývojářům, kteří ji vyhodnotili jako chybu v implementaci. Oprava nebyla do nové verze aplikace dosud provedena a tak jsme v následujících testech export nepoužili.

Test jsme zkoušeli použít přímo po restartu aplikace s obnovou databáze a poté jsme test spustili znova. Při porovnání výstupů jsme objevili velké rozdíly mezi prvním a druhým testem. U testu provedeného hned po restartování aplikace trvaly odpovědi delší dobu než při druhém spuštění. Problém byl prodiskutován s vývojáři aplikace a příčina tohoto chování byla odůvodněna ukládáním některých dat do mezipaměti aplikace. V reálném běhu aplikace budou data již v mezipaměti, takže po restartu aplikace před požadovaným testem provedeme stejný test jen s malým počtem uživatelů a krátkým průběhem. Tím naplníme mezipaměť potřebnými daty a následně provedeme test s již potřebným počtem uživatelů a délkou běhu.

8.2 Určení hranice uživatelů

8.2.1 Zkouška testu

Test jsme vyzkoušeli při zátěži s 1000 uživateli. Během něj provedlo pouze 20 uživatelů úspěšné přihlášení a další uživatelé byli zamítnuti. Z logu aplikačního serveru byl objeven problém s cache poolem, který udává maximální počet uživatelů přistupujících k aplikaci. Po diskuzi s vývojáři byla nalezena příčina, kdy po přihlášení uživatelů v počtu převyšujícím velikost poolu do-

šlo k jeho zaplnění, ale už nedocházelo k uvolnění místa pro další uživatele. Žádosti o přihlášení těchto uživatelů byly aplikací zamítnuty.

Velikost poolu tedy omezovala maximální počet přihlášených uživatelů. Jelikož oprava této chyby by znamenala rozsáhlé úpravy, vývojáři aplikaci upravili pro testování tak, aby bylo možné velikost cache poolu snadno konfigurovat a zvýšit tak počet použitelných uživatelů.

8.2.2 Provedení testů

Test pro 1000 uživatelů po zvětšení poolu na stejnou hodnotu doběhl úspěšně, ale z vygenerovaného reportu jsme zjistili, že doby odpovědi trvají delší dobu, než jsou stanovené limity. Z toho důvodu jsme test několikrát opakovali vždy s menším množstvím uživatelů, dokud časy akcí nepřekračovaly nadefinové hranice. S tímto postupem jsme se dostali na zátěž generovanou pomocí 100 uživatelů.

8.2.3 Vyhodnocení testů

Při analyzování jednotlivých reportů byly objeveny u některých testů velmi krátké doby odpovědí na přihlášení. Například u testu s 500 uživateli byl průměrný čas odezvy přihlášení 6 ms, ale u 200 uživatelů to bylo skoro 70 ms. Toto chování bylo shledáno pouze u přihlášení, u ostatních požadavků byly odezvy podle předpokladu. Provedli jsme ještě několik testů se stejným počtem uživatelů, ale s různými časovými prodlevami mezi nimi. Čím menší byla prodleva mezi jednotlivými testy, tím přihlášení trvalo kratší dobu. V případě spouštění testů s intervalem větším než několik minut se časy pro přihlášení téměř shodovaly. Tento jev byl nahlášen vývojovému oddělení. Z oddělení jsme se dozvěděli o problému s postupným uvolňováním odhlášených uživatelů z cache poolu, kdy například všech 1000 uživatelů bylo uvolněno z poolu až po 5 minutách. Uživatelé tedy zůstali v poolu a z toho důvodu jejich opětovné přihlášení trvalo o dost kratší dobu. Proto v následujících testech budeme muset vždy počkat, než se uvolní všichni uživatelé z poolu a až poté budeme moci spustit další test. Abychom měli jistotu, že byl pool uvolněn, požádali jsme o vytvoření skriptu pro oznámení uvolnění veškerých uživatelů.

Po zopakování testů jsme dostali časy, jenž odpovídaly míře zatížení. V tabulce 8.2.3, kde jsou ke každému počtu uživatelů uvedeny nejdelší doby akcí pro přihlášení a otevření dokumentu, můžeme vidět zmenšující se časy až do řádky se 100 uživateli, u nichž časová kritéria vyhověla požadavkům (2 sekundy pro přihlášení a 10 sekund pro otevření dokumentu). Tento počet

uživatelů, u kterých přihlášení a vytvoření dokumentu nepřesáhlo zadanou hranici, můžeme označit jako největší a zároveň přijatelný.

Počet uživatelů	Přihlášení [ms]	Otevření dokumentu [ms]
1000	33288	116470
500	3872	57493
250	3172	23055
125	2614	10491
100	1739	8507

Tabulka 8.1: Přehled maximálních dob odpovědí pro rozdílný počet uživatelů

8.2.4 Výkonnější server

Na výkonnějším serveru (Consultator) jsme provedli testy se stejným postupem. Odezvy byly o dost rychlejší a pod stanovenou hranici jsme se dostali už s 250 uživateli viz tabulka 8.2.

Počet uživatelů	Přihlášení [ms]	Otevření dokumentu [ms]
1000	16925	46818
500	3465	19523
250	1688	9731

Tabulka 8.2: Přehled maximálních dob odpovědí u výkonnější serveru

8.3 Navyšování zátěže

8.3.1 Provedení testů

Pro zjištění chování aplikace při zvyšující se zátěži, jsme provedli jednotlivé testy s postupným navyšováním počtu uživatelů. Množství uživatelů jsme od nuly zvětšovali s každým testem o 20 dalších uživatelů až do doby, kdy jsme dosáhli dvojnásobného množství uživatelů zjištěného v předchozím testu(kapitola 8.2.4) tzn. 200 uživatelů.

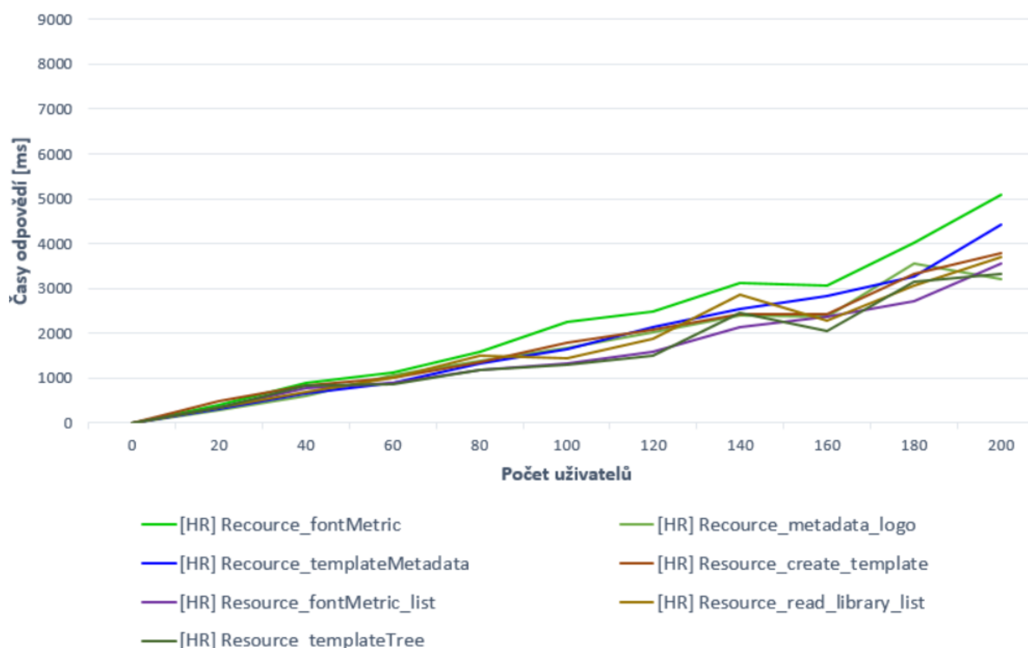
8.3.2 Vyhodnocení testu

Z vygenerovaných reportů jsme zjistili průměrné časy všech požadavků a následně jsme vytvořili graf v závislosti časů dopovědí v milisekundách na počtu uživatelů. Pro přehlednost jsme graf rozdělili na několik podgrafů se zachování poměrů kvůli porovnání.

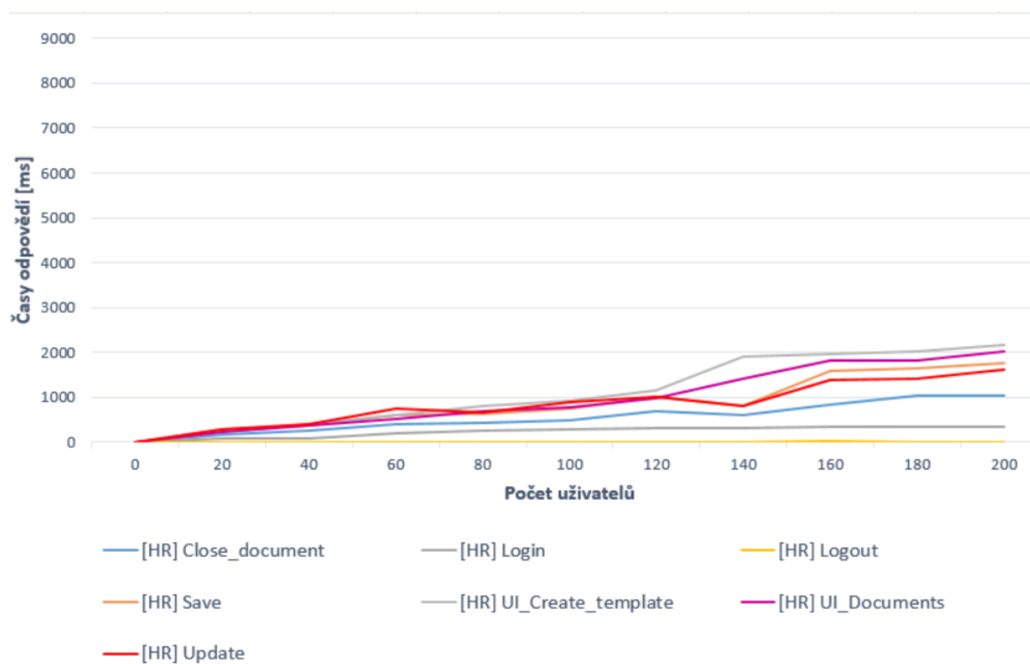
V prvním podgrafu 8.1 můžeme vidět požadavky týkající se získání zdrojů pro sestavení dokumentu. Doby odpovědí jsou skoro u všech žádosti totožné. K odchýlení dochází pouze v případě požadavku Resource_fontMetric. Delší prodleva byla pravděpodobně způsobena náročnějším zpracováním požadavku.

Druhý graf 8.2 obsahuje běžné požadavky k obsluze aplikace. Můžeme pozorovat nízké doby odpovědí s velmi pomalým lineárním růstem.

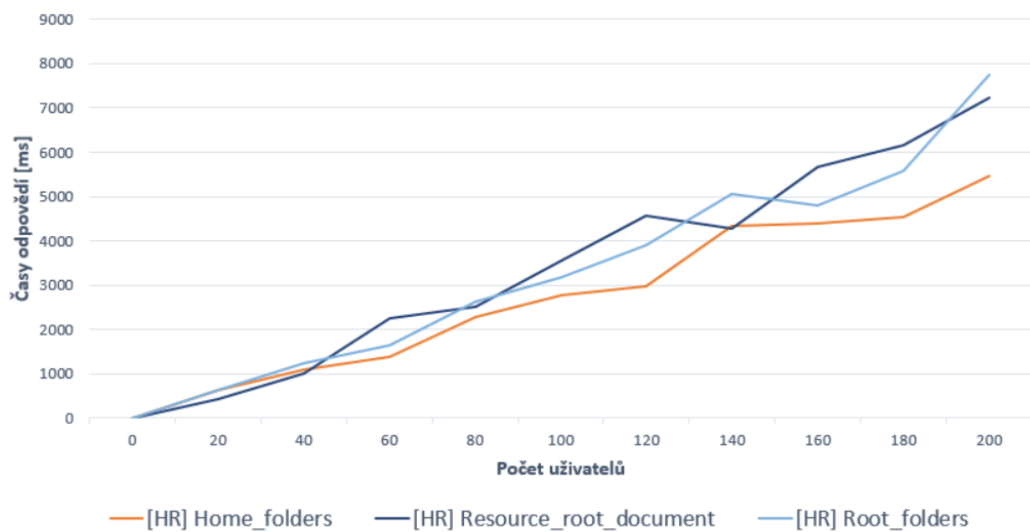
Nejvyšší časy jsou pozorovatelné v grafu 8.3. Časy se výrazně lišily v porovnání s předchozími požadavky. Všechny zobrazené požadavky přistupují ke složkám a žádají o jejich podsložky a soubory. Toto chování nebylo očekávané, a tak jsme provedli testy s ještě větším zatížením. Odezvy u těchto požadavků dosahovaly několikrát větších časů než u ostatních žádostí. Tento jev byl nahlášen a v následující kapitole provedeme upřesnění problému.



Obrázek 8.1: Požadavky pro získání zdrojů



Obrázek 8.2: Běžné požadavky k obsluze aplikace



Obrázek 8.3: Požadavky s přístupem k souborům

8.3.3 Identifikace problému

Pro identifikaci problému a vyloučení některých příčin bylo navrženo provést změnu v konfiguraci databáze a zopakovat test s vysokým počtem uživatelů za použití nástroje Javamelody (viz 3.2.2) pro sledování přístupů do databáze. Z výsledku nového testu nebylo shledáno žádné zlepšení, ani v přístupech do databáze nebyl objeven problém. Všechny dotazy se zpracovávaly velmi rychle a bez čekání. Dosažené výsledky byly opět předány vývojářům a posléze byla nalezena možná příčina při přístupu k souborům, kdy kvůli synchronizaci více uživatelů přistupujících ke stejné složce docházelo k čekání na její uvolnění.

8.3.4 Výkonnější server

Na výkonnějším serveru bylo provedeno několik testů s větším počtem uživatelů, než tomu bylo na předchozím serveru. Chování problémových požadavků se podle předpokladu nezměnilo. Pouze se při stejné zátěži snížily doby odpovědí.

8.4 Porovnání různých dokumentů

8.4.1 Provedení testů

Každý test jsme provedli pro každý typ dokumentu desetkrát, abychom dosáhli přesnějšího výsledku. Testy pro jeden typ šablony nebyly spouštěny zvlášť, nýbrž jsme v konfiguračním souboru nastavili opakování testu na deset. Ze všech deseti opakování testu jsme dostali jeden souhrný report, což nám usnadnilo analýzu.

8.4.2 Vyhodnocení testu

Z jednotlivých testů šablon jsme zjistili průměrné časy odpovědí pro získání všech zdrojů a dále i pro update a uložení souboru. Hodnoty jsme zapsali do tabulky 8.3 do první poloviny sloupců.

8.4.3 Výkonnější server

Stejné testy byly provedeny i na výkonnějším počítači. Dosažené výsledky jsme opět zapsali do tabulky 8.3.

Při porovnání můžeme vidět dosažení u všech požadavků téměř poloviční dobu odezvy na rozdíl od méně výkonného serveru. Porovnali jsme i

ostatní požadavky, které nejsou zahrnuty v tabulce a nejsou závislé na typu dokumentu. I odpovědi na tyto požadavky měly poloviční dobu odezvy.

Typ dokumentu	**Update dokumentu [ms]**		**Uložení dokumentu [ms]**		**Získání zdrojů [ms]**	
	Pomalý server	**Rychlý server**	**Pomalý server**	**Rychlý server**	**Pomalý server**	**Rychlý server**
Základní	83	43	107	51	588	334
Single	103	50	108	55	2138	1041
Four	96	50	100	46	822	309
Many	340	224	165	77	962	464

Tabulka 8.3: Trvání odpovědí u různých dokumentů pro pomalý a rychlý server

8.5 Hodinový zátěžový test

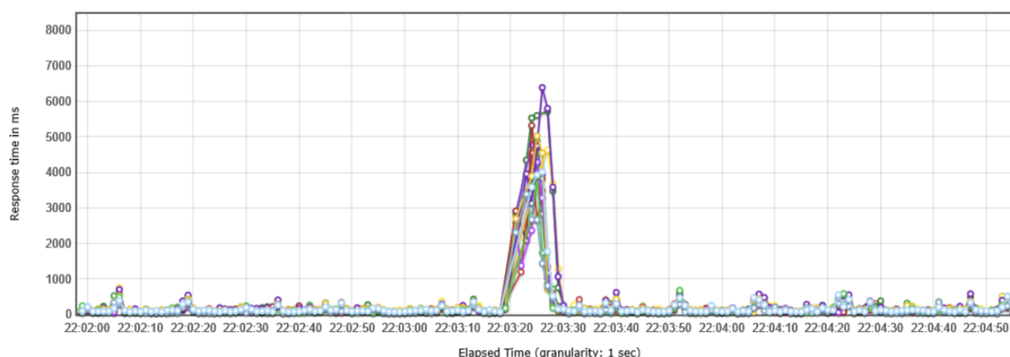
8.5.1 Provedení testů

Nejprve jsme provedli řadu testů s různým počtem uživatelů. Testy vždy běžely několik minut a pomocí Kibany (viz 3.2.2) jsme čekali, dokud se zatížení procesoru neustálilo a tím jsme určili procentuální zatížení, jenž můžeme očekávat po dobu jedné hodiny.

Začali jsme na 100 uživatelích, ale tento počet byl nedostatečný. Procesor dosáhl zatížení pouze 20 %, proto jsme počet uživatelů zvyšovali. Požadovaného vytížení 80 % jsme docílili až při 600 uživatelích. Po dosažení hranice zatížení jsme mohli přistoupit k provedení hodinového testu.

8.5.2 Vyhodnocení

Po celou dobu testu se zatížení procesoru pohybovalo kolem 80 % s maximální odchylkou 5 %. Během testu tedy nedošlo k žádným výrazným výkyvům. Následně jsme provedli analýzu reportu. Odpovědi po celou dobu běhu testu trvaly maximálně jednu sekundu, kromě jednoho výkyvu. Anomálie nastala po 45 minutách běhu testu. Časy odpovědí vzrostly k sedmi vteřinám, ale během několika sekund se opět ustálily na normálních hodnotách (viz 8.4). Mimo krátkodobého vychýlení časů u odpovědí nevznikly u aplikace ani u zatížení procesoru žádné další velké odchylky, které by ukazovaly nestabilní chování aplikace.



Obrázek 8.4: Zobrazení výkyvu z reportu

8.6 Počet dokumentů za minutu

8.6.1 Provedení testu

Při vytváření dokumentů nesmíme překročit u žádné používané akce stanovenou hranici pěti sekund trvání. Pro nalezení vhodného počtu uživatelů, při kterých akce splňovaly tuto podmínku, jsme museli provést několik testů s délkou trvání jedné minuty. Začínali jsme na stech uživatelích a postupně jsme museli snižovat. Podmínku se nám podařilo splnit při zátěži s patnácti uživateli. Poté jsme s tímto množsví uživatelů provedli deset minutových testů pro získání přesnějšího počtu vytvořených dokumentů za minutu.

8.6.2 Vyhodnocení

Během testu se nám povedlo vytvořit 767 dokumentů se zátěží patnácti uživatelů. Časy akcí po celou dobu nepřekročily stanovenou mez pěti sekund. Dosažený výsledek je tedy 76 dokumentů za minutu. Po nahlášení hodnoty vývojovému oddělení byl dosažený počet vytvořených dokumentů označen jako velmi dobrý.

8.6.3 Výkonnější server

Stejný postup jsme zopakovali i na výkonnějším serveru. Abychom nepřesáhli limit pěti sekund u žádné akce, musíme použít nanejvýš dvacet uživatelů. S tímto počtem uživatelů jsme za deset minut vytvořili 1444 dokumentů. Oproti předchozímu pomalejšímu serveru jsme dosáhli za stejnou dobu **celkově** dvojnásobného počtu dokumentů. Rozdíl v počtu vytvořených dokumentů jedním uživatelem není v našem případě důležitý údaj pro porovnání.

9 Kontinuální integrace

Jak již bylo zmíněno v kapitole 3.2.3, firma dlouhodobě používá pro kontinuální integraci systém Jenkins. V této kapitole bude popsána integrace zátěžových testů do již vytvořeného skriptu pro build aplikace. Aplikace bude umístěna na výkonnější server.

9.1 Typy testů

K testování aplikace pomocí Jenkinse použijeme tři sady testů.

9.1.1 Ověřovací test

První bude ověřovací test, který jsme už naimplementovali v 7.5.1 a tuto implementaci použijeme i u kontinuální integrace. Pokud během tohoto testu dojde k chybě, další testy se pak nespustí a bude tedy zřejmé, že nedošlo k chybě při zátěži, nýbrž kvůli chybě ve funkčnosti testované aplikace.

9.1.2 Práce s vytvořeným dokumentem

Další test bude zaměřený na otevírání a zavírání vytvořeného dokumentu při nárazovém zatížení. Test jsme již také vytvořili (viz 7.5.2). Pro zatížení použijeme 250 uživatelů. Tento počet byl stanoven jako hraniční hodnota získaná z testu provedeného na výkonnějším serveru v kapitole 8.2.4.

Výsledek testu se bude vyhodnocovat podle časů potřebných na akce. Pro každou akci stanovíme časovou hranice a v případě, že se akce nevykoná do stanoveného času, bude test považován za selhaný. Hranice byla expertním odhadem stanovena o 10 % větší, než je běžná doba pro vykonání každé akce. Dobu akcí zjistíme z reportu spuštěním testu běžným způsobem.

9.1.3 Vytváření dokumentu

Test bude velmi podobný hodinovému testu (viz 7.5.5). Oproti původní implementaci změníme časovače na konstatní s délkou zpoždění na 300 milisekund. Konstatní časovač jsme zvolili, aby průběh zátěže byl vždy stejný. Další změna bude v délce testu. Test nebyl omezen na dobu běhu, ale změnili jsme nekonečné opakování smyčky pro vytváření dokumentů na počet iterací nastavitelný v konfiguračním souboru.

Výsledky budeme vyhodnocovat podobně jako v předchozím testu. V potaz budeme brát dobu na odpovědi na jednotlivé požadavky místo času potřebného na vykonání celých akcí. Jedním z důvodů je například akce pro uložení dokumentu skládající se z několika důležitých požadavků (získání složek pro uložení, update, uložení), u nichž potřebujeme znát jednotlivé časy.

9.2 Integrace

Jak bylo zmíněno v úvodu kapitoly, skript pro build aplikace již v Jenkinsu existuje. Do tohoto skriptu je třeba přidat spouštění testů s kontrolou a potřebné pluginy.

9.2.1 Spouštění testů

Jelikož JMeter umožňuje spuštění testu z příkazové řádky, použijeme ve skriptu stejný příkaz. Abychom zbytečně nezabírali místo na disku, budeme jako výsledek testu ukládat pouze výstupní data do CSV souboru bez generování grafického reportu. V případě potřeby můžeme report vygenerovat z CSV souboru pomocí JMeteru. Testy budeme spouštět postupně podle pořadí, ve kterém jsme je popsali v předchozí kapitole 9.1.

9.2.2 Kontrola limitů

Po každém dokončeném testu potřebujeme vyhodnotit, zda proběhl úspěně a zda všechny požadavky resp. akce byly splněny do stanoveného času. K tomuto účelu použijeme nástroj zvaný Lightning[1].

Lightning provede analýzu vygenerovaného CSV souboru a určí, zda jsou všechna pravidla nadefinová v přiloženém XML souboru (viz ukázka 9.1) splněna. Na základě výsledku analýzy vytvoří soubor ve formátu JUnit XML(viz 9.2.4). Pro vytvoření výstupu ze souboru s tímto formátem musíme do Jenkinse přidat rozšíření zvané JUnit Plugin[2]. Plugin zahrne výstup do souhrného výsledku všech provedených testů.

```
<avgRespTimeTest>
    <!-- Nazev testu -->
    <testName>Open close: Login test</testName>
    <!-- Popis testu -->
    <description>Verify average login times</description>
    <!-- Testovana akce -->
    <transactionName>TC_Login</transactionName>
```

[1] http://deliverymind.github.io/lightning/index.html

[2] https://wiki.jenkins-ci.org/display/JENKINS/JUnit+Plugin

```
    <regexp/>
    <!-- Podminka na prumerne odpoved do 100 ms-->
    <maxAvgRespTime>100</maxAvgRespTime>
</avgRespTimeTest>
```

Ukázka 9.1: Pravidlo pro průměrnou dobu odezvy

9.2.3 Výsledný skript

Skript v první části vytvořené vývojáři obsahuje definování cest a build aplikace mIQ. V další části se nachází náš skript. Skript obsahuje jednotlivé spouštění testů pomocí JMeteru. Testu předchází příprava, jenž zahrnuje zastavení aplikačního serveru, nahrání výchozí zálohy do databáze a opětovné nastartování serveru. Následně provedeme test bez uložení výsledků, abychom naplnili mezipaměť a zavoláme skript, který čeká na uvolnění všech uživatelů z poolu. Poté spustíme test a provedeme vyhodnocení. Ve skriptu 9.2 můžeme vidět jeden ze tří testů i s přípravou aplikace.

```
SERVER_PATH/server_stop.sh
SERVER_PATH/backup.sh -dump dump_default
SERVER_PATH/server_start.sh

JMETER_HOME/jmeter.sh -n -t JMETER_HOME/Jenkins_tests/Pre_test/TP_Pre_test.
    jmx
SERVER_PATH/pool_check.sh
JMETER_HOME/jmeter.sh -n -t JMETER_HOME/Jenkins_tests/Pre_test/TP_Pre_test.
    jmx -l pre_test.csv

java -jar Lightning_HOME/lightning-standalone-*.jar verify -xml JMETER_HOME
    /Jenkins_test/Pre_test/treshold_test.xml --jmeter-csv pre_test.csv
```

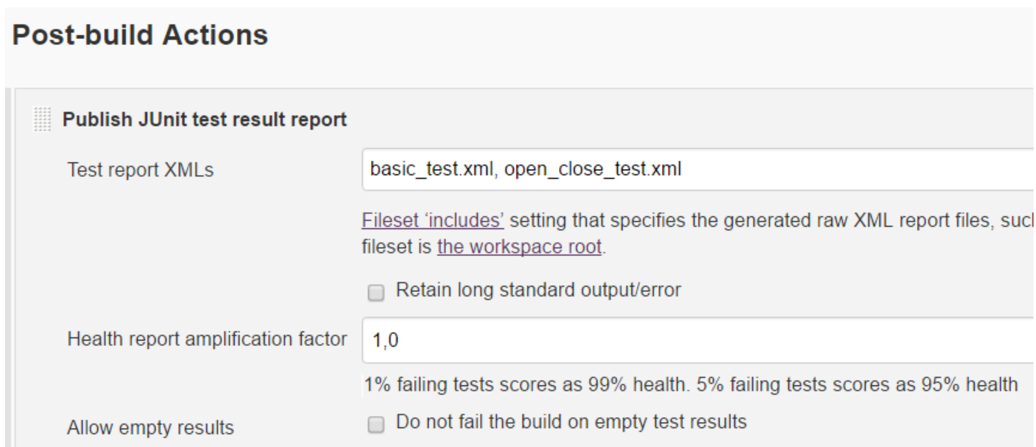
Ukázka 9.2: Skript pro ověřovací test

9.2.4 Report výsledků

Po ukončení testů musíme z vygenerovaných souborů vytvořit přehledy o průběhu testů a vzniklých chybách. Přehledy budeme vytvářet z XML souboru z nástroje Lightning a CSV souboru z JMeteru.

JUnit XML soubor

V části o kontrole limitů 9.2.2 jsme se zmínili o rozšíření JUnit Plugin pro report výsledku z aplikace Lightning. Tento plugin nám přidává do Jenkinse možnost (viz obr. 9.1) vyhodnotit vytvořené JUnit XML soubory (viz skript 9.3) po skončení skriptu.



Obrázek 9.1: Reportování JUnit testu

```
<?xml version="1.0" encoding="UTF-8"?><testsuite errors="0" failures="1"
    name="Lightning" tests="4" time="0">
  <testcase name="Open close: Login test" time="0"/>
  <testcase name="Open close: Open document test" time="0"/>
  <testcase name="Open close: Close document test" time="0">
  <failure message="Average response time = 60" type="avgRespTimeTest">
        Test name:              Open close: Close document test
        Test type:              avgRespTimeTest
        Test description:       Verify average close document times
        Transaction name:       TC_Close_document
        Expected result:        Average response time &lt;= 30
        Actual result:          Average response time = 63
        Transaction count:      8
        Longest transactions:   [67, 66, 66, 66, 63]
        Test result:            FAIL
  </failure>
  </testcase>
  <testcase name="Open close: Logout test" time="0"/>
</testsuite>
```

Ukázka 9.3: JUnit XML soubor s chybou u akce pro zavření dokumentu

Report budeme vytvářet pouze pro testy 9.1.2 a 9.1.3. Výsledek z ověřovacího testu není nutné reportovat, protože při výskytu chyby se vykonávání skriptu ukončí s chybou. Pokud některý z požadavků nesplní podmínku, tak se do souhrného výsledku testu přidá informace o chybě(viz obr. 9.2).

Failed

Lightning.Open close: Close document test

Error Message

```
Average response time = 63
```

Stacktrace

```
Test name:            Open close: Close document test
Test type:            avgRespTimeTest
Test description:     Verify average close document times
Transaction name:     TC_Close_document
Expected result:      Average response time <= 30
Actual result:        Average response time = 63
Transaction count:    8
Longest transactions: [67, 66, 66, 66, 63]
Test result:          FAIL
```

Obrázek 9.2: Report chyby z JUnit XML souboru

CSV soubor

Aby jsme mohli zobrazit přehledy z CSV souboru, musíme do Jenkinse nainstalovat rozšíření Performance Plugin. Plugin nám umožní vygenerovat grafy a tabulky podobné jako u normálního reportu od JMeteru, ale do tabulek přidá ještě porovnání s předchozím testem(viz obrázek 9.3). Tuto akci přidáme opět až po vykonání skriptu, jako jsme to udělali u vyhodnocení souboru pomocí Lightningu.

URI	Samples	Samples diff	Average (ms)	Average diff (ms)	Median (ms)	Median diff (ms)
HR_Close_document	10	0	62	1	62	0
HR_Home_folders	5	0	125	1	73	0
HR_Login	5	0	72	-3	77	-1
HR_Logout	5	0	2	0	2	0
HR_Recource_fontMetric	5	0	85	2	81	0

Obrázek 9.3: Část tabulky s porovnáním

10 Závěr

Na začátku práce jsem se seznámil s problematikou zátěžových testů, aplikací a dosavadní testovací infrastrukturou firmy. Znalost aplikace a infrastruktury byla nutnou podmínkou pro volbu vhodného nástroje. Pro výběr nástroje jsem prozkoumal dostupné nástroje a omezil jsem se pouze na ty, jenž splňovaly stanovená kritéria. Každý nástroj ze zúženého výběru jsem prakticky vyzkoušel na jednotné aplikaci a na základě výsledků jsem vypracoval jejich multikriteriální ohodnocení. Pomocí ohodnocení vlastností a k nim firmou přiřazených priorit byl k otestování aplikace zvolen nástroj JMeter.

Před implementací testů jsem se důkladně seznámil s vybraným nástrojem a navrhl testy, které byly prodiskutovány s vývojovým a testovacím oddělením. V JMeteru se mi podařilo veškeré navržené testy implementovat. Firma tak získala sadu škálovatelných testů s možností jejich jednoduché modifikace.

Účinnost testování byla prokázána v průběhu vykonávání testů. Nalezl jsem problémy, které při funkčním ani jiném testování nebylo dosud možné odhalit. Tyto chyby by mohly způsobit nesprávný chod aplikace a to nejen při zvýšené zátěži. Problémy byly identifikovány například u mezipaměti pro přihlášené uživale a při přístupu velkého množství uživatelů ke stejné složce. K nalezení původu chyb bylo nutné spolupracovat s vývojáři aplikace a pomocí dalších testů vyloučit možné příčiny, jako je například nadměrné zatížení databáze.

Kromě odhalení výše zmíněných chyb posloužila moje práce i k tomu, že byl zjištěn počet vytvořených dokumentů za určitou dobu a časové rozdíly ve vyváření dokumentů z různých typů šablon.

Protože byly sady zátěžových testů vytvořeny v dostatečném časovém předstihu, mohly být následně testy upraveny tak, aby je bylo možné začlenit do testovacího cyklu aplikace. Testy i jejich vyhodnocení jsem integroval do buildu testované aplikace v nástroji pro kontinuální integraci. To vedlo k plnohodnotnému využití vytvořených zátěžových testů. Výsledky mé práce jsou tedy již rutinně využívány.

Přehled zkratek

GWT – Google Web Toolkit
XML – eXtensible Markup Language
HTML – HyperText Markup Language
HTTP – Hypertext Transfer Protocol

Literatura

[1] Vavřík, J. *Zátěžové testy webových aplikací.* Brno, 2010. Diplomová práce na fakultě informatiky Masarykovy univerzity. Vedoucí práce: RNDr. Radek Ošlejšek, Ph.D.

[2] Molyneaux, I. *The Art of Application Performance Testing, 2nd Edition: From Strategy to Tools.* O'Reilly Media, 2014. ISBN 1-4919-0054-7.

[3] Šmeral, R. *Modern Performance Tools Applied.* Brno, 2014. Diplomová práce na fakultě informatiky Masarykovy univerzity. Vedoucí práce: Mgr. Marek Grác, Ph.D.

[4] Meier, J. et al. *Performance Testing Guidance for Web Applications: Patterns & practices.* Microsoft Press, 2007. ISBN 9780735625709.

[5] Elasticsearch. [online]. Ver. 5.1.2. 2017. [cit. 2017/2/12]. Nástroj Kibana pro sledování zátěže. Dostupné z: `https://www.elastic.co/products/kibana`.

[6] Foundation, A. S. [online]. Ver. 1.66. 2016. [cit. 2017/2/9]. Nástroj pro monitorování Java aplikací. Dostupné z: `https://github.com/javamelody/javamelody`.

[7] Kawaguchi, K. [online]. Ver. 2.46.1. 2017. [cit. 2017/1/11]. Kontinuální integrace. Dostupné z: `https://jenkins.io/`.

[8] Foundation, A. S. [online]. Ver. 3.1. 1999. [cit. 2016/12/02]. Nástroj JMeter pro zátěžové testy. Dostupné z: `http://jmeter.apache.org/`.

[9] Corp, G. [online]. Ver. 2.2.3. 2000. [cit. 2016/12/02]. Nástroj Gatling pro zátěžové testy. Dostupné z: `http://gatling.io/#/`.

[10] Packard, H. [online]. Ver. 12.53. 2000. [cit. 2017/12/11]. Nástroj LoadRunner pro zátěžové testy. Dostupné z: `https://saas.hpe.com/en-us/software/loadrunner`.

[11] Neotys. [online]. Ver. 5.4. 2005. [cit. 2017/12/11]. Nástroj Neoload pro zátěžové testy. Dostupné z: `https://www.neotys.com/neoload/overview`.

[12] Software, S. [online]. Ver. 4.6. 2009. [cit. 2017/12/11]. Nástroj LoadComplete pro zátěžové testy. Dostupné z: `https://smartbear.com/product/loadcomplete`.

[13] Aston, P. – Fitzgerald, C. [online]. Ver. 3.11. 2013. [cit. 2016/12/02]. Nástroj The Grinder pro zátěžové testy. Dostupné z: http://grinder.sourceforge.net/.

[14] Niclausse, N. [online]. Ver. 1.6.0. 2005. [cit. 2017/12/11]. Nástroj Tsung pro zátěžové testy. Dostupné z: http://tsung.erlang-projects.org/.

[15] TestAutomationGuru. [online]. Ver. 1.1. 2017. [cit. 2017/1/11]. Plugin do JMeteru. Dostupné z: http://www.testautomationguru.com/jmeter-property-file-reader-a-custom-config-element/.

A Příloha k nástrojům

A.1 Gatling

A.1.1 Skript

```
import scala.concurrent.duration._

import io.gatling.core.Predef._
import io.gatling.http.Predef._
import io.gatling.jdbc.Predef._

class RecordedSimulation extends Simulation {
  val httpProtocol = http
    .baseURL("http://kalda-pc.pilsen.kadel.cz:8090")
    .acceptHeader("text/html,application/xhtml+xml,application/xml;q=
        0.9,*/*;q=0.8")
    .acceptEncodingHeader("gzip, deflate")
    .acceptLanguageHeader("cs,en-US;q=0.7,en;q=0.3")
    .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0) Gecko
        /20100101 Firefox/47.0")

  val headers_0 = Map("Content-Type" -> "text/plain; charset=utf-8")

    val uri1 = "http://kalda-pc.pilsen.kadel.cz:8090/mtext"

  val scn = scenario("RecordedSimulation")
    .exec(http("Login")
      .post("/mtext/app/login")
      .formParam("username", (s:Session) => """mtext"""+s.userId.toString)
      .formParam("pwd", (s:Session) => """mtext"""+s.userId.toString))
    .exec(http("Root_folder")
      .get("/mtext/app/ui/documents/Folder/home/mtext"))
    .pause(1)
    .exec(http("Home_folder")
      .get("/mtext/app/resource/document/root/%2Fhome%2Fmtext?count=20")
      .headers(headers_0))
    .pause(2)
    .exec(http("Template_tree")
      .get("/mtext/app/resource/templateTree/%2FVorlagen%2FLetters?count=20
        ")
      .headers(headers_0))
    .pause(1)
    .exec(http("Retrieve_template_metadata")
      .get("/mtext/app/resource/templateMetadata/library/demo/Welcome%20
        Letter.dataBinding")
      .headers(headers_0))
    .pause(3)
    .exec(http("Create_document_from_databinding")
      .get("/mtext/app/resource/create/Welcome%20Letter.dataBinding?base=
        demo")
      .headers(headers_0)
      .check(header("x-kw-tmp-document-id").saveAs("tmp-document-name")))
    .exec(http("Retrieve_font_metric_list")
```

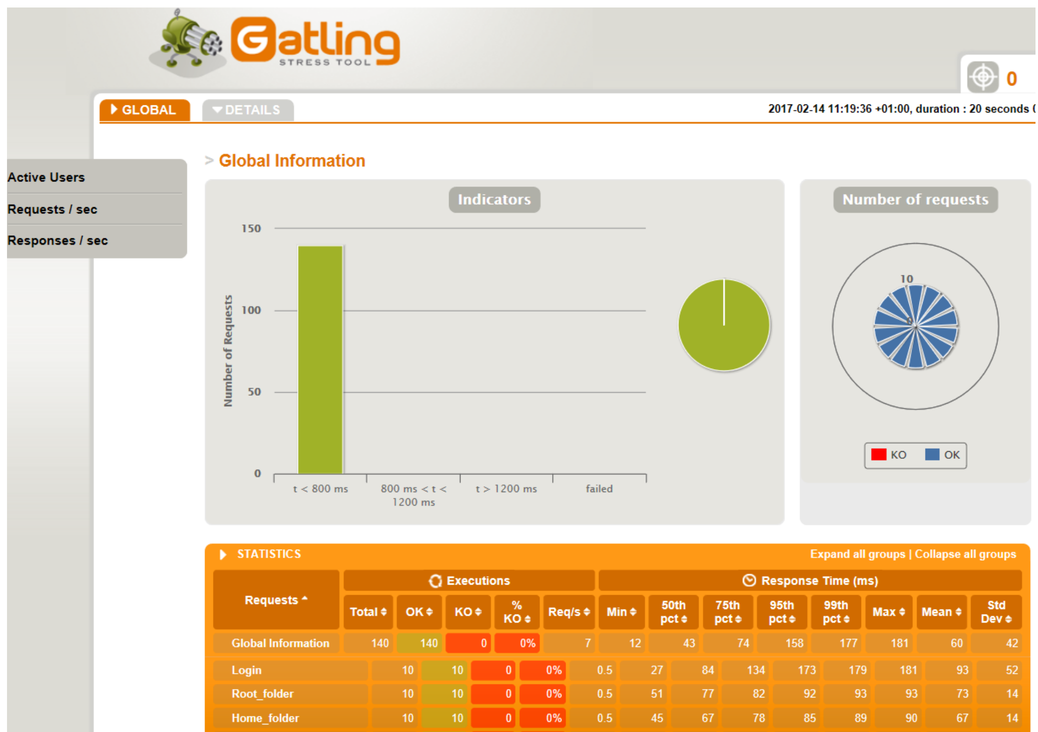
```
      .get("/mtext/app/resource/fontMetric/list")
      .headers(headers_0))
    .exec(http("Retrieve_resource")
      .get("/mtext/app/resource/read/%5Clibrary%5C.list")
      .headers(headers_0))
    .pause(2)
    .exec(http("Get_root_folder")
      .get("/mtext/app/resource/folders/%2F")
      .headers(headers_0))
    .pause(2)
    .exec(http("Get_home_folder")
      .get("/mtext/app/resource/folders/%5Chome")
      .headers(headers_0))
    .pause(2)
    .exec(http("Update_document")
      .post("/mtext/app/resource/update/${tmp-document-name}")
      .formParam("content", """<Root xmlns="urn:kwsoft:mtext:structured:dom
          ">
          <RootPart>
            <DataDefinition></DataDefinition>
            <DocumentCollection>
              <Document id="Welcome letter" base="demo">
                <Section>
                  <Container>
                    <Par>
                      <Style>
                        <Hyphenation>true</Hyphenation>
                        <Align>left</Align>
                      </Style>
                      <Span>
                        <Text>Welcome to Sunshine Insurance! mtext</Text>
                      </Span>
                    </Par>
                  </Container>
                </Section>
              </Document>
            </DocumentCollection>
          </RootPart>
          <RootPartInstance></RootPartInstance>
        </Root>"""))
    .exec(http("Save_document")
      .post("/mtext/app/resource/save/${tmp-document-name}")
      .formParam("original", (s:Session) => """\home\mtext"""+s.userId.
          toString+"""\DOC_gatling.dataBinding""")
      .formParam("replace", "false"))
    .pause(3)
    .exec(http("Close_document")
      .post("/mtext/app/resource/close/${tmp-document-name}"))
    .pause(2)
    .exec(http("Logout")
      .post("/mtext/app/logout"))

  setUp(scn.inject(atOnceUsers(10))).protocols(httpProtocol)
}
```

Ukázka A.1: Skript zkušebního testu

A.1.2 Report

Kompletní report najdete na CD ve složce Nástroje/Gatling/report.

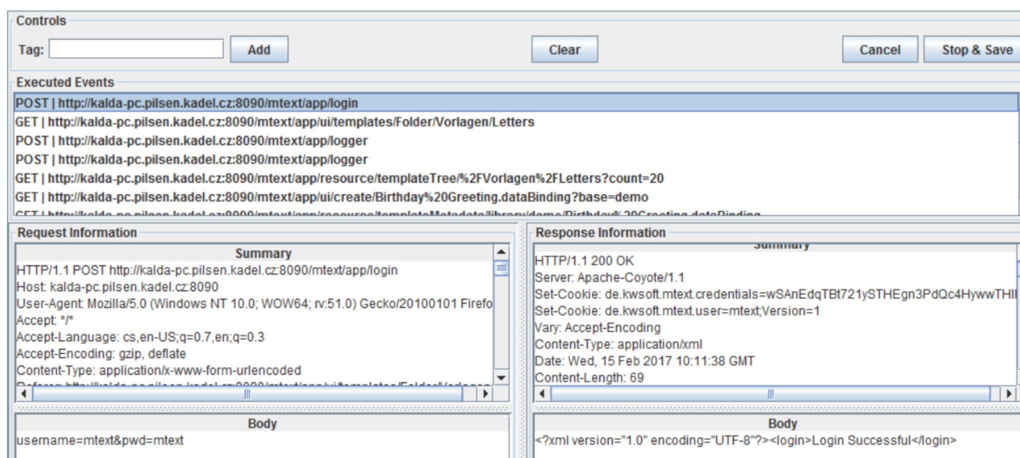
Obrázek A.1: Report z Gatlingu



Obrázek A.2: Report z Gatlingu

A.1.3 Rekorder

Obrázek A.3: Rekorder Gatlingu



Obrázek A.4: Rekorder Gatlingu

A.2 The Grinder

A.2.1 Skript

V ukázce A.2 je pouze začátek skiptu. Zbytek skriptu je na CD ve složce Nástroje/The Grinder/zkusebni_skript.py

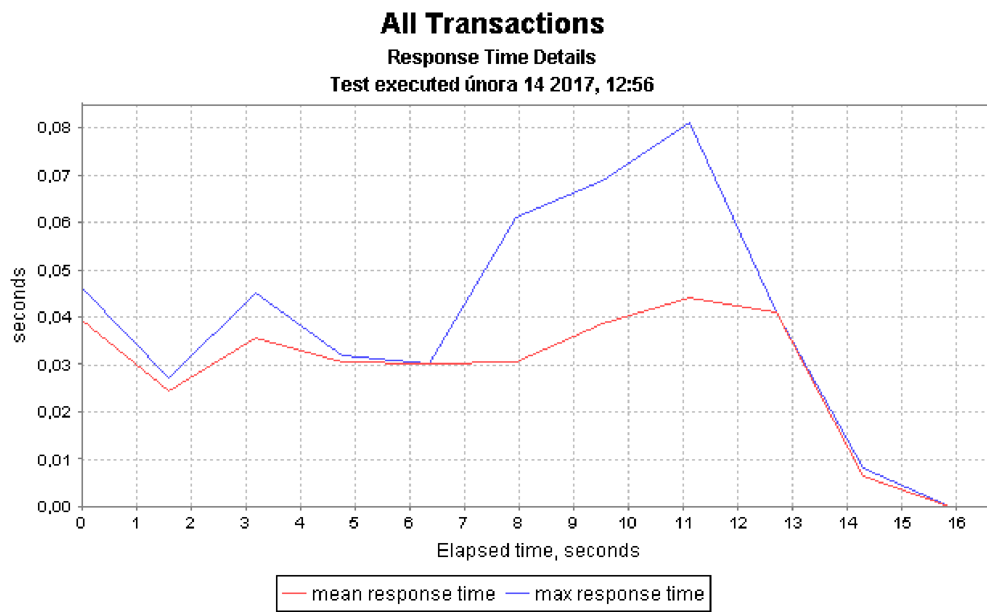
```
from net.grinder.script import Test
from net.grinder.script.Grinder import grinder
from net.grinder.plugin.http import HTTPPluginControl, HTTPRequest
from HTTPClient import NVPair
connectionDefaults = HTTPPluginControl.getConnectionDefaults()
httpUtilities = HTTPPluginControl.getHTTPUtilities()

# To use a proxy server, uncomment the next line and set the host and port.
# connectionDefaults.setProxyServer("localhost", 8001)

def createRequest(test, url, headers=None):
    """Create an instrumented HTTPRequest."""
    request = HTTPRequest(url=url)
    if headers: request.headers=headers
    test.record(request, HTTPRequest.getHttpMethodFilter())
    return request

# These definitions at the top level of the file are evaluated once,
# when the worker process is started.

connectionDefaults.defaultHeaders = \
  [ NVPair('Accept-Language', 'cs,en-US;q=0.7,en;q=0.3'),
    NVPair('Accept-Encoding', 'gzip, deflate'),
    NVPair('User-Agent', 'Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0)
        Gecko/20100101 Firefox/47.0'), ]

headers0= \
  [ NVPair('Accept', 'text/html,application/xhtml+xml,application/xml;q=
      0.9,*/*;q=0.8')]
```

```
url0 = 'http://kalda-pc.pilsen.kadel.cz:8090'

request101 = createRequest(Test(101, 'Login'), url0, headers0)

request201 = createRequest(Test(201, 'Root_folder'), url0, headers0)

request301 = createRequest(Test(301, 'Home_folder'), url0, headers0)

request401 = createRequest(Test(401, 'Template_tree'), url0, headers0)

request501 = createRequest(Test(501, 'Retrieve_template_metadata'), url0,
    headers0)

request601 = createRequest(Test(601, 'Create_document_from_databinding'),
    url0, headers0)

request701 = createRequest(Test(701, 'Retrieve_font_metric_list'), url0,
    headers0)

request801 = createRequest(Test(801, 'Retrieve_resource'), url0, headers0)

request901 = createRequest(Test(901, 'Get_root_folder'), url0, headers0)

request1001 = createRequest(Test(1001, 'Get_home_folder'), url0, headers0)

request1101 = createRequest(Test(1101, 'Update_document'), url0, headers0)

request1201 = createRequest(Test(1201, 'Save_document'), url0, headers0)

request1301 = createRequest(Test(1301, 'Logout'), url0, headers0)


class TestRunner:
  """A TestRunner instance is created for each worker thread."""
  tmpDocName = ""
  # A~method for each recorded page.
  userID = 0
  def page1(self):
    """POST login (request 101)."""
    result = request101.POST('/mtext/app/login',
      ( NVPair('username', "mtext"+`self.userID`),
        NVPair('pwd', "mtext"+`self.userID`)))

    return result

  def page2(self):
    """GET home (requests 201-210)."""
    result = request201.GET('/mtext/app/ui/documents/Folder/home')

    return result
```

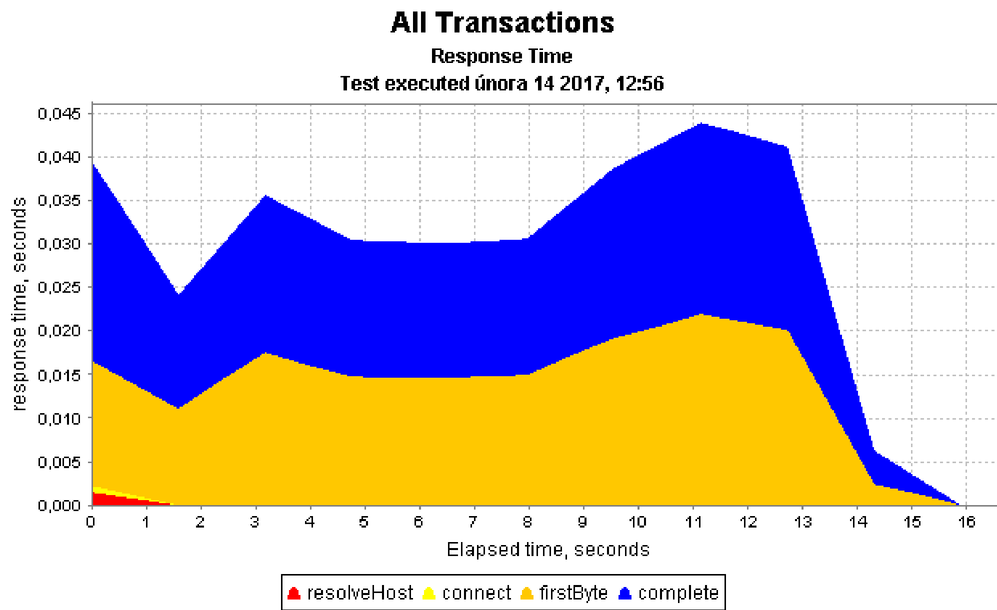
Ukázka A.2: Část skriptu pro ověřovací test v The Grinderu

A.2.2 Report

Kompletní report najdete na CD ve složce Nástroje/The Grinder/report.



Obrázek A.5: Report z The Grinderu

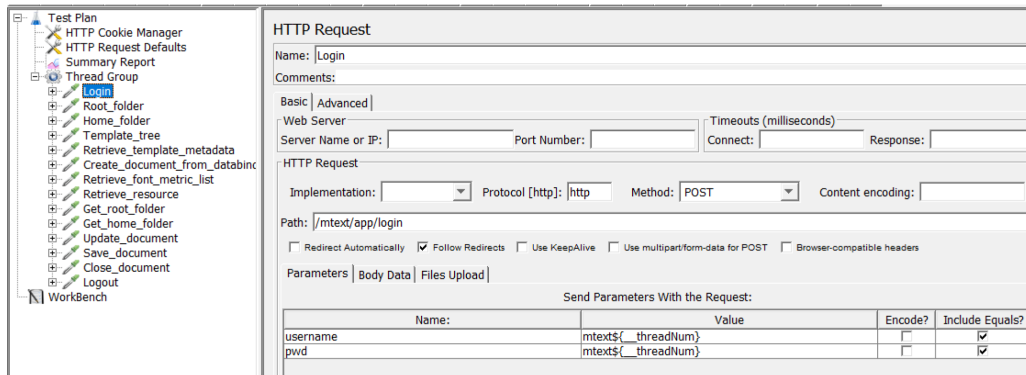


Obrázek A.6: Report z The Grinderu

A.3 JMeter

A.3.1 Skript

Vytvořený skript v GUI můžete vidět na obrázku A.7. V ukázce A.3 je pouze začátek vygenerovaného skiptu. Zbytek skriptu je na CD ve složce Nástroje/JMeter/zkusebni_skript.jmx



Obrázek A.7: Skript v GUI u JMeteru

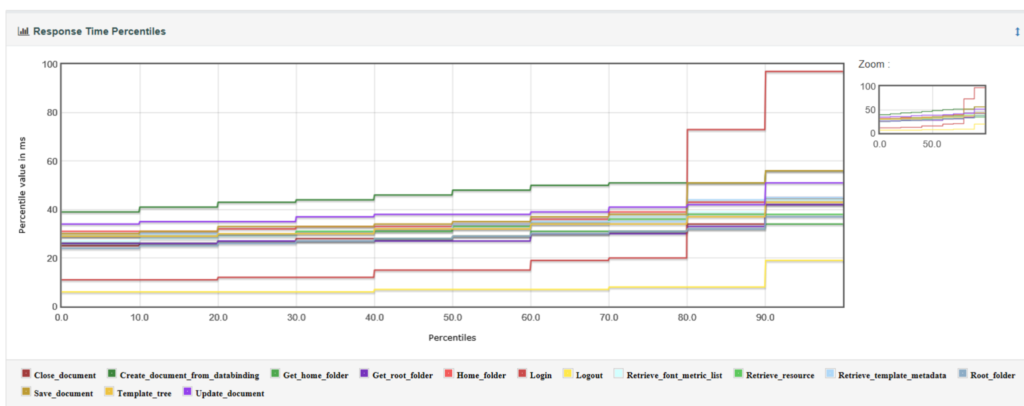
```
<?xml version="1.0" encoding="UTF-8"?>
<jmeterTestPlan version="1.2" properties="3.1" jmeter="3.1 r1770033">
  <hashTree>
    <TestPlan guiclass="TestPlanGui" testclass="TestPlan" testname="Test
        Plan" enabled="true">
      <stringProp name="TestPlan.comments"></stringProp>
      <boolProp name="TestPlan.functional_mode">false</boolProp>
      <boolProp name="TestPlan.serialize_threadgroups">false</boolProp>
      <elementProp name="TestPlan.user_defined_variables" elementType="
          Arguments" guiclass="ArgumentsPanel" testclass="Arguments"
          testname="User Defined Variables" enabled="true">
        <collectionProp name="Arguments.arguments"/>
      </elementProp>
      <stringProp name="TestPlan.user_define_classpath"></stringProp>
    </TestPlan>
    <hashTree>
      <ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup"
          testname="Thread Group" enabled="true">
        <stringProp name="ThreadGroup.on_sample_error">continue</stringProp
            >
        <elementProp name="ThreadGroup.main_controller" elementType="
            LoopController" guiclass="LoopControlPanel" testclass="
            LoopController" testname="Loop Controller" enabled="true">
          <boolProp name="LoopController.continue_forever">false</boolProp>
          <stringProp name="LoopController.loops">1</stringProp>
        </elementProp>
        <stringProp name="ThreadGroup.num_threads">10</stringProp>
        <stringProp name="ThreadGroup.ramp_time">1</stringProp>
        <longProp name="ThreadGroup.start_time">1485770556000</longProp>
        <longProp name="ThreadGroup.end_time">1485770556000</longProp>
```

```
      <boolProp name="ThreadGroup.scheduler">false</boolProp>
      <stringProp name="ThreadGroup.duration"></stringProp>
      <stringProp name="ThreadGroup.delay"></stringProp>
    </ThreadGroup>
    <hashTree>
      <CookieManager guiclass="CookiePanel" testclass="CookieManager"
          testname="HTTP Cookie Manager" enabled="true">
        <collectionProp name="CookieManager.cookies"/>
        <boolProp name="CookieManager.clearEachIteration">false</boolProp
            >
        <stringProp name="CookieManager.policy">standard</stringProp>
        <stringProp name="CookieManager.implementation">org.apache.jmeter
            .protocol.http.control.HC4CookieHandler</stringProp>
      </CookieManager>
```

Ukázka A.3: Část skriptu pro ověřovací test v JMeteru

A.3.2 Report

Kompletní report najdete na CD ve složce Nástroje/JMeter/report.



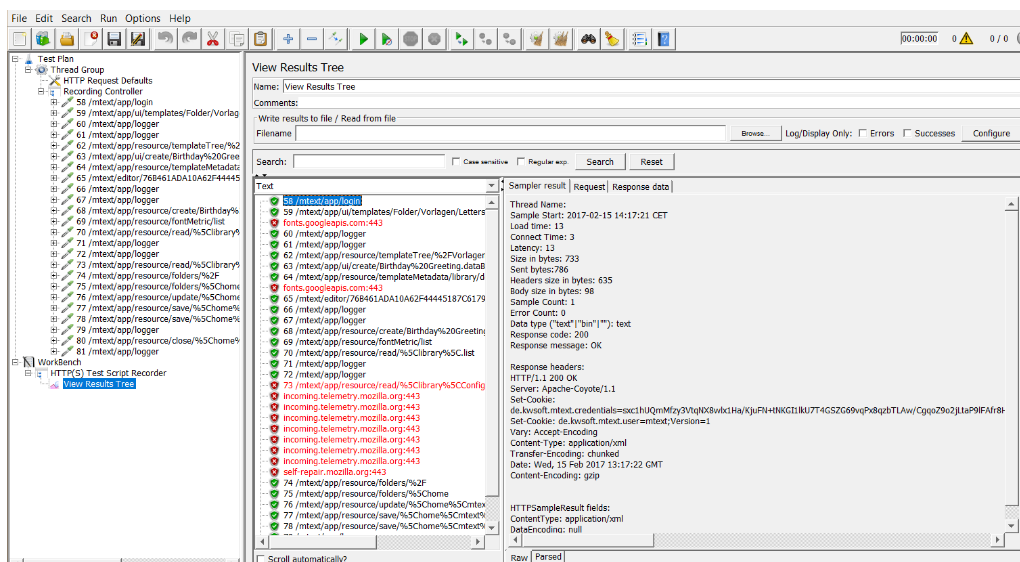
Obrázek A.8: Report z JMeteru

Statistics

Label	#Samples	KO	Error %	Average response time	90th pct	95th pct	99th pct	Throughput	Received KB/sec	Sent KB/sec	Min	Max
Total	**140**	**0**	**0.00%**	**31.93**	**44.00**	**51.00**	**87.16**	**116.76**	**51.60**	**109.86**	**6**	**97**
Close_document	10	0	0.00%	30.00	41.20	42.00	42.00	13.70	2.34	12.85	25	42
Create_document_from_databinding	10	0	0.00%	46.90	55.50	56.00	56.00	12.69	11.27	11.60	39	56
Get_home_folder	10	0	0.00%	29.10	33.80	34.00	34.00	13.42	5.57	12.09	26	34
Get_root_folder	10	0	0.00%	29.50	41.10	42.00	42.00	13.40	4.79	12.02	26	42
Home_folder	10	0	0.00%	35.70	44.80	45.00	45.00	12.72	5.62	11.07	31	45
Login	10	0	0.00%	28.50	94.60	97.00	97.00	12.15	8.41	5.95	11	97
Logout	10	0	0.00%	8.00	17.90	19.00	19.00	14.22	5.79	12.48	6	19
Retrieve_font_metric_list	10	0	0.00%	33.20	43.50	44.00	44.00	13.14	6.63	11.83	27	44
Retrieve_resource	10	0	0.00%	33.00	38.00	38.00	38.00	13.33	6.48	12.11	29	38

Obrázek A.9: Report z Jmeteru

A.3.3 Rekorder



Obrázek A.10: Rekorder JMeteru

B Testy v JMeteru

B.1 Vytvořené testy

Testy použité během testování se nalézají na CD ve složce Skripty/Zakladni_testy a testy určené pro Jenkins jsou ve složce Skripty/Jenkins_testy.

B.1.1 Složka Zakladni_testy

Složka obsahuje konfiguraci (config.properties) společnou pro všechny testy. Dále obsahuje soubor s uživateli (users.txt) a nastevený rekordér(Recording.jmx).

Fragmenty testů jsou uloženy ve složce Test_fragments a samostatné testy ve složce Test_plan.

Testy ve složce test_plan:

- TP_Basic_scenary – Navyšování zátěže, porovnání různých dokumentů
- TP_Doc_count – Počet dokumentů
- TP_Hour_load – Hodinový zátěžový test
- TP_Pre_test – Ověřovací test
- TP_User_limit – Určení hranice uživatelů

B.1.2 Složka Jenkins_testy

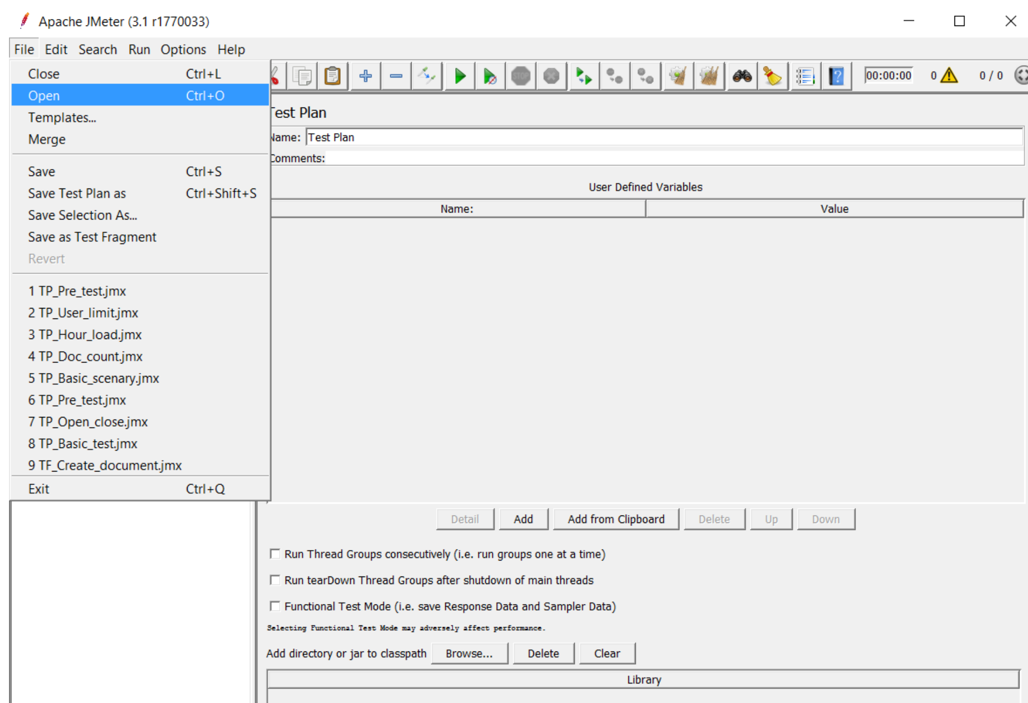
Složka obsahuje soubor s uživateli (users.txt) a složky pro jednotlivé testy. Každá tato složka obsahuje konfiguraci a skript testu. Dále se v ní nachází soubor treshlod_test.xml, který je určen pro nástroj Lightnining ke kotrole dob odpovědí.

B.1.3 Spuštění testů

Testy se dají zobrazit i spustit pomocí GUI. Lze je také spustit pouze v příkazové řádce.

Nástroj s GUI lze spustit pomocí JAR souboru ApacheJmeter.jar ve složce bin(java -jar cesta_k_adresari_s_JMeterem/bin/ApacheJmeter.jar). Je možné také využít již vytvořené skripty pro Windows (jmeter.bat) anebo

pro Linux (jmeter.sh). Po zobrazení GUI lze vytvořené skripty otevřít v záložce File/Open viz obr. B.1.



Obrázek B.1: Otevření skriptu

Pro spuštění testu bez GUI je nutné zadat parametry -n a -t s názvem skriptu (JMX soubor). Pro vygenerování CSV souboru slouží parametr -l s názvem výstupního CSV soubor a pro grafický report je nutné přidat paramtery -e a -o s názvem složky pro výstup. Příklad spuštění testu bez GUI s reportováním do CSV souboru a vygenerování grafického reportu je vidět na obrázku B.2.

```
ludek@tester:~/apache-jmeter-3.1/bin$ java -jar ApacheJMeter.jar  -n
-t Tests/Test_plans/TP_pre_test.jmx -l Reports/pre_test.csv -e -o Rep
orts/pre_test
```

Obrázek B.2: Příklad spuštění testu

B.2 Reporty

Reporty se nalézají na CD ve složce Reporty. Tato složka obsahuje složky s reporty k jednotlivým testům ze serveru vmhost01 a consultator. Ve slož-

kách se nalézají pouze CSV soubory. Grafický report z CSV souboru se vytvoří pomocí parametrů -g s názevem CSV souboru a -o s názvem výstupní složky (viz obr. B.3)

```
ludek@tester:~/apache-jmeter-3.1/bin$ java -jar ApacheJMeter.jar -g
Reports/pre_test.csv -o Reports/pre_test
```

Obrázek B.3: Příklad vygenerování grafického reportu

Po vykonání příkazu se vytvoří složka s reportem. Pro zobrazení grafického reportu spustíme HTML soubor index.html ve vytvořené složce.

Reporty ve složce vmhost01:

- Doc_compare – Reporty testů pro 4 různé dokumenty
- Doc_count – Report testů pro zjištění počtu dokumentů
- Growing_load – Reporty testů s vyšujícím se počtem uživatelů
- Hour_test – Report z hodinového testu
- User_limit – Report testů pro určení hranice uživatelů

Reporty ve složce consultator:

- Doc_compare – Reporty testů pro 4 různé dokumenty
- Doc_count – Report testů pro zjištění počtu dokumentů
- User_limit – Report testů pro určení hranice uživatelů