UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Gjorgi Markovski

Testiranje spletnih aplikacij s testnim orodjem JMeter

DIPLOMSKO DELO NA
VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: dr. Igor Rožanc

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00027/2010
Datum: 04.10.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GJORGI MARKOVSKI**

Naslov: **TESTIRANJE SPLETNIH APLIKACIJ Z TESTNIM ORODJEM JMETER**
TESTING WEB APPLICATIONS WITH JMETER TESTING TOOL

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Pri testiranju spletnih aplikacij so nam v veliko pomoč testna orodja, v okviru katerih lahko nastavimo ustrezno testno okolje in oblikujemo testne scenarije za različne namene.

V svoji nalogi najprej kratko predstavite orodje za testiranje javanskih aplikacij JMeter. Glavni del naloge naj bo čimbolj celovit prikaz uporabe tega orodja pri obremenitvenem in performančnem testiranju spletne aplikacije in podatkovnih zbirk. V ta namen predstavite oblikovanje ustreznih testnih scenarijev, vzpostavite testno okolje ter praktično prikažite testiranje. V zadnjem delu analizirajte dobljene rezultate in jih komentirajte.

Mentor:

viš. pred. dr. Igor Rožanc

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a: Gjorgi Markovski ,

z vpisno številko: 63040405 ,

sem avtor/-ica diplomskega dela z naslovom:

Testiranje spletnih aplikacij s testnim orodjem JMeter

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  viš. pred. dr. Igor Rožanc

  in somentorstvom (naziv, ime in priimek)

  _____

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 14.4.2011 Podpis avtorja: _____

Zahvala

Za strokovno vodenje, nasvete in pomoči pri izdelavi diplomske naloge se iskreno zahvaljujem mentorju viš. pred. dr. Igoru Rožancu. Prav tako se zahvaljujem vsem sodelavcem v podjetju MAOP d.o.o., ki so na svoji način pripomogli k nastanku naloge.

Še posebej bi se rad zahvalil punci, staršem in bratu za potrpežljivost, razumevanje, skrb in podporo skozi študij.

Kazalo vsebine

Tabela kratic

SQL – (Structured Query Language) jezik za pisanje podatkovnih poizvedb.

DOS – (Denial Of Service) zavrnitev storitve.

DDOS – (Distributed Denail Of Service) porazdeljena zavrnitev storitve.

JDBC – (Java Database Connectivity) standard za neodvisno povezovanje med podatkovnimi bazami in aplikacijami, ki ga podpira programski jezik Java.

LDAP – (Lightweight Directory Access Protocol) aplikacijski protokol za branje in urejanje imenikov preko IP omrežja.

IP – (Internet Protocol) spletni protokol.

TCP – (Transmission Control Protocol) protokol za kontrolo in prenos podatkov.

POP3 – (Post Office Protocol 3) protokol za prejemanje elektronske pošte.

IMAP – (Internet Message Access Protocol) spletni standardni protokol za prenos elektronske pošte.

FTP – (File Transfer Protocol) protokol za prenos podatkov.

HTTP – (Hypertext Transfer Protocol) omrežni protokol za porazdeljene informacijske sisteme.

HTTPS – (Hypertext Transfer Protocol Secure) varen omrežni protokol za porazdeljene informacijske sisteme.

JS – (JavaScript) skriptni jezik za spletne aplikacije.

LAN – (Local Area Network) lokalno omrežje.

CPE – Centralna Procesna Enota.

SSL – (Secure Sockets Layer) kriptografski protokol za varno komunikacijo.

URL – (Uniform Resource Locator) enolični določnik vira.

CSS – (Cascading Style Sheets) predloge za določanje izgleda spletnih strani.

GIF – (Graphics Interchange Format) slikovni format.

TIFF – (Tagged Image File Format) slikovni format.

JPEG – (Joint Photographic Experts Group) slikovni format.

PNG – (Portable Network Graphics) slikovni format.

BMP – (Bitmap Image Files) slikovni format.

XML – (eXtensible Markup Language) razširljiv označevalni jezik za opis podatkov.

IWAP – (Integrated Windows Authentication Protocol) avtentikacijski protokol v sistemu Windows.

CSV – (Comma Separated Values) podatki, ki so ločeni z vejico.

GUI – (Graphic User Interface) grafični uporabniški vmesnik.

RMI – (Remote Method Invocation) oddaljeni sklici metod.

JVM – (Java Virtual Machine) javanski navidezni stro

Povzetek

Namen diplomske naloge je opis sestavljanje in izvajanja avtomatiziranih performančnih testov aplikacij tipa odjemalec-strežnik z orodjem JMeter. V okviru dela sem najprej določil ugotavljal cilje in razloge za izvajanje avtomatiziranih performančnih testiranj.

JMeter je testno orodje za izvajanje različnih tipov performančnih in obremenitvenih testiranj. Zbiranje in sestavljanje podatkov predstavlja osnovo za izgradnjo dobrega testnega scenarija. Najprej sem opisal postopek sestavljanja testnih scenarijev za testiranje spletnih aplikacij, podatkovnih strežnikov in strežnikov, ki uporabljajo FTP protokol. Testne scenarije je mogoče izvajati na več različnih načinov. Predstavil sem, kako to izvajamo z orodjem JMeter in analiziral dobljene podatke. Na koncu dela sem iz analize dobljenih podatkov ugotovil delež obremenjenosti strežnika pri največjem številu uporabnikov in predvidel kako lahko preprečimo nadaljnje povečevanje števila istočasnih uporabnikov.

Ključne besede: testni scenarij, performančno testiranje, obremenitveno testiranje, JMeter.

Abstract

The purpose of my thesis is description of assembly and execution of automatic performance tests for client-server application with the JMeter testing tool. Furthermore, I identify the goals and reasons for execution of automatic performance testing.

JMeter is a testing tool for execution different types of performance and load test. Collecting and assembling data is a base for constructing a good test scenario. First of all, I described the process of assembling a test scenario for testing web applications, database servers and servers using FTP protocols. Test scenarios can be done in a few different ways. I described one way with testing tool JMeter and accordingly I analyzed the gained data. At the end of my thesis, I analyzed the gained data and I found out the amount of load the server when it has maximum number of users. At the end, I propose a way of prevention of additional increasment of the number of simultaneous users.

Key words: test scenarios, performance testing, load testing, Jmeter.

1. Uvod

Aplikacije odjemalec-strežnik so danes vse bolj popularne in tudi njihovo število uporabnikov hitro narašča. Zato je še bolj pomembno, da vsako aplikacijo pred predstavitvijo končnemu uporabniku dobro testiramo.

Testiranje lahko zajema različne dele aplikacije. Ponavadi je povezano z določenimi cilji kot so funkcionalnost, uporabnost, kompatibilnost, performančnost, varnost, testiranje grafičnih uporabniških vmesnikov in podobno. Testiranje aplikacije lahko tako postane zelo zapletena zadeva. Za razvoj aplikacije ali informacijskega sistema je potrebna skupina programerjev, ki bo to delala več mesecev. Na koncu morajo testerji vse pretestirati. Kakovost aplikacije je zelo odvisna od kakovosti testiranja. Testiranje naštetih ciljev zahteva različna orodja in vsako testiranje se izvaja ločeno. Testiranje sicer predstavlja širši pojem, a sem se v moji diplomski nalogi omejil na predstavitev avtomatiziranih preformančnih testiranj aplikacije odjemalec-strežnik in njenih sestavnih delov z uporabo testnega orodja JMeter.

JMeter je brezplačna, odprtokodna in zelo prilagodljiva namizna javanska aplikacija za izvajanje avtomatiziranih performančnih testiranj. Kot močno testno orodje za izvajanje performančnih testiranj ima tudi možnosti za testiranje različnih standardov. Dela na vseh operacijskih sistemih, ki podpirajo več nitnost in omogoča nadgradnjo z dodatnimi funkcijami (*ang. plug-ins*) [3].

Čas testiranja aplikacije je najpomembnejši. Odvisno od časa lahko testiranje izvajamo ročno ali avtomatizirano (računalniško). Ročno testiranje je bolj učinkovito, kakovostno in prilagodljivo, a ga lahko izvajamo le takrat, ko imamo dovolj časa in je aplikacija dovolj majhna. Pri velikih aplikacijah ponavadi nimamo dovolj časa in testiranje izvajamo avtomatizirano. Z generiranjem navideznih uporabnikov lahko izvajamo avtomatizirano testiranje. Ključna razlika med navideznim in realnim uporabnikom je neupoštevanje časa pri navideznih uporabnikov, ki je potreben realnemu uporabniku za pregledovanje, razmišljanje ter izbiro med ponujenimi opcijami, ki so prikazane v aplikaciji.

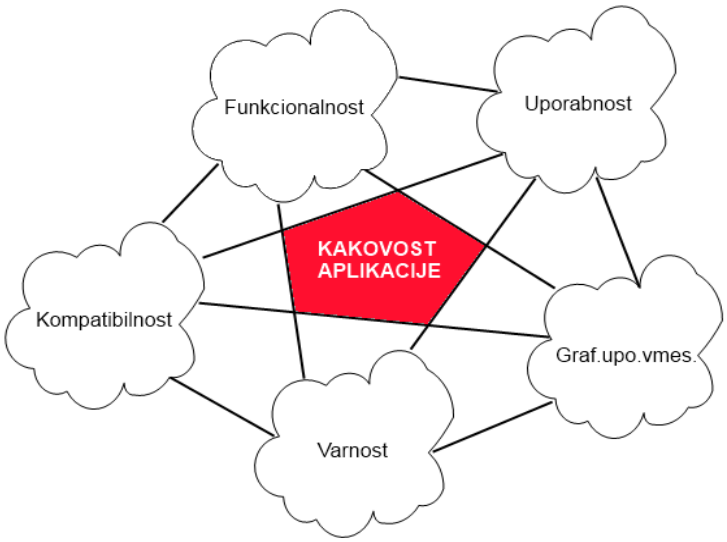
Performančni testi skrbijo za gladko in neprekinjeno delovanje aplikacije brez kakršnihkoli zakasnitev, kar pomeni da testiranje približno ustreza številu končnih uporabnikov. Ugotavljamo, torej, kako se bosta aplikacija in strežnik obnašala, ko se bo povezalo določeno število istočasnih uporabnikov. Tedaj se vidi, ali strežnik oziroma aplikacija predstavlja ozko grlo. Med testirajnem lahko v aplikaciji tudi optimiziramo počasnejše dele z namenom, da bo aplikacija zdržala čim več uporabnikov, zmanjšala porabo strojne opreme, razbremenila strežnik ter postala čim hitrejša.

Prekomerno število istočasnih uporabnikov lahko rešimo na dva načina. Prvi način je, da uporabnike postavimo v čakalno vrsto, drugi način pa je ta, da jih prestavimo na kasnejši čas, ko bo strežnik manj obremenjen. Ključno je, da razlika odzivnega časa pri nekaj uporabnikih ali pri največjem dovoljene številu uporabnikov ne sme biti prevelika.

2. Tipi testiranj in opis testnega orodja

2.1 Tipi testiranj

Za razvoj aplikacij obstajajo različne tehnologije in orodja. To pomeni, da imajo razvijalci dokaj odprte roke. Ampak vsaka tehnologija pokriva različna področja in ima svoje prednosti in slabosti. Vsa področja so medsebojno povezana kot je prikazano na sliki 1. Vsako področje je vezano na različne načine testiranja. Poleg performančnih testiranj, ki so bolj natančno in obširno opisana v moji diplomski nalogi, lahko vsako aplikacijo testiramo na več načinov.



Slika 1: Povezanost med področji testiranj.

2.1.1 Testiranje funkcionalnosti

Funkcija je sestavni del vsake aplikacije. Vsaka funkcija predstavlja eno ali več zahtev. Funkcije, ki jih aplikacija izvaja, morajo biti opisane v zahtevani specifikaciji ali primerih uporabe. Vsota vseh funkcij, ki jih aplikacija izvaja, predstavlja funkcionalnost aplikacije. Testiranje funkcionalnosti aplikacije se izvaja na celotni aplikaciji tako, da se oceni skladnost sistema s specifičnimi zahtevami. Preverjamo vsako funkcijo posebej, če ta vrača pravilno vrednost na podlagi naših zahtev oziroma vhodnih podatkov. Testiranje funkcionalnosti ne zahteva poznavanja vsebine aplikacije (kode ali logike). Pri testiranju funkcionalnosti nas zanimajo izhodni podatki, ki nastanejo kot obdelava vhodnih parametrov. Brez testiranja funkcionalnosti aplikacije je velika verjetnost, da bo ta na koncu napačno delovala ali bodo tako delovale nekatere funkcionalnosti aplikacije. Orodja za testiranje funkcionalnosti aplikacije: HP Quick Test Professional [4], ApPerfect [5], DejaGnu [6].

2.1.2 Testiranje uporabnosti

Vsaka aplikacija je namenjena uporabnikom. To pomeni, da mora biti aplikacija čim bolj pregledna, lahka za uporabo in učenje, privlačna za uporabnike, čim bolj enostavna in učinkovita. Najbolj pomembno pri uporabnosti je čimprej pridobiti željeno informacijo. Najpogostejša oblika testiranja uporabnosti je postavitev testne verzije aplikacije tako, da lahko razvijalci sproti dobivajo povratne informacije ter prilagajajo aplikacijo uporabniškim zahtevam. Obstajajo orodja za avtomatsko testiranje, vendar se ponavadi nove aplikacije gradijo na podlagi izkušenj iz prejšnjih aplikacij. Pri razvoju aplikacije je potrebno uporabljati določena orodja in se držati pravil. Če razvijalci ne dobivajo uporabnikove povratne informacije ali ne vedo, kakšen tip uporabnikov bo uporabljal aplikacijo se lahko zgodi, da je aplikacija nepregledna in tuja uporabnikom. To posledično pomeni majhno število uporabnikov in nizek rang pri spletnih iskalnikih, s tem pa tudi izgubo popularnosti in denarja. Nekatera orodja za testiranje uporabnosti aplikacij so: Chalkmark [7], Google web site optimizer [8], ClickHeat [9].

2.1.3 Testiranje grafičnih uporabniških vmesnikov

Grafični uporabniški vmesnik omogoča uporabnikom interakcijo z aplikacijo, zato testiramo izgled vmesnika, barve in podobno. Za prikaz aplikacij odjemalec-strežnik uporabljamo različne spletne brskalnike. Poznamo več različnih spletnih brskalnikov, med katerimi so najbolj popularni Internet Explorer, Firefox, Chrome, Opera in Safari. Brskalniki pa žal različno interpretirajo aplikacije, čeprav bi moral biti prikaz enak pri vseh. Zato pomemben cilj testiranja je enak prikaz aplikacije v različnimi brskalinki, za katerega poskrbi test grafičnih uporabniških vmesnikov. Če ga ne naredimo obstaja velika možnost, da bo aplikacija nepregledna ali težka za razumevanje, saj med najpogostejše napake sodi neurejen prikaz elementov. Nekatera orodija za testiranje grafičnih uporabniških vmesnikov pri spletnih aplikacijah so: ClubicTest [10], Selenium [11], Watir [12].

2.1.4 Testiranje kompatibilnosti

Uporaba različnih tehnologij, strojne opreme, orodij, razvojnih okolij ali operacijskih sistemov vpliva na kompatibilnost aplikacije. Kompatibilnost predstavlja sposobnost naprave ali programa za delo z drugimi napravami ali programi. Podjetja, ki razvijajo programsko opremo pogosto ne morejo predvideti vseh možnih kombinacij programske in strojne opreme različnih uporabnikov, zato razvijajo aplikacijo na najbolj verjetni programski in strojni opremi. Razvijalci morajo biti že pred začetkom pozorni na naročnikovo programsko in strojno opremo, da ne bo prevelikih razlik. Test kompatibilnosti se naredi pri prenosu aplikacije iz razvojnega okolja v naročnikovo okolje. S testiranjem kompatibilnosti preverimo, če aplikacija dela enako na naročnikovem in razvojnem okolju. Če testa kompatibilnosti ne naredimo (ali nimamo podatkov o naročnikovi strojni in programski opremi) obstaja velika možnost, da aplikacija ne bo delala v naročnikovem okolju.

2.1.5 Varnostno testiranje

Varnost je verjetno najpomembnejša pri strežniško-odjemalčevi aplikaciji. Napadi lahko zajemajo različne dele sistema, zato mora varnost pokrivati celoten sistem. Varnost lahko razdelimo na fizično in tehnično. Fizična varnost obsega varnost sistema pred krajo, poplavami, pomanjkanjem elektrike in podobnim. Tehnično varnost sestavljajo varnost aplikacije, operacijskega sistema in omrežja, vendar se pri tem dovoli uporabnikom dostop do informacij in premoženja.

Obstajajo različni napadi in vdori. Med temi so najpogostejši napadi z vrivanjem SQL stavkov (*ang. SQL injections*), DOS napadi, DDOS napadi in preverjanje odprtih vrat (*ang. Port scaning*). Napad ali vdor ne pomeni samo krajo podatkov, ampak pogosto tudi onemogočanje delovanja aplikacije. Najpogostejši vzrok za ranljivost aplikacij so napake, ki dajajo napadalcu informacije o programski in strojni opremi aplikacije. Zmanjšanje nevarnosti napadov in vdorov najpogosteje zagotovimo z različnimi orodji za testiranje varnosti aplikacije. Če varnosti ne testiramo, je velika možnost za krajo podatkov ali nezanesljivo delovanje aplikacije. Nekatera orodja za testiranje tehnične varnosti pri spletnih aplikacijah so: Netsparker [13], Websecurify [14], Webscarab [15].

2.2 Opis testnega orodja JMeter

JMeter je namizna javanska aplikacija za izvajanje avtomatiziranih performančni in obremenitvenih testov [16]. Tipično merimo in analiziramo dobljene podatke aplikacij tipa odjemalec-strežnik. Testno orodje JMeter ima možnosti za testiranje dostopa do podatkov preko JDBC, LDAP, WebServices ter testiranje aplikacij preko Internet, TCP, POP3, IMAP, FTP, HTTP, HTTPS protokolov. JMeter je brezplačno in odprtokodno orodje, zato ga lahko prilagajamo ali nadgrajujemo za svoje potrebe. Za prilagajanje in nadgradnjo je potrebno znanje programskega jezika Java, ki je neodvisen od platforme, kar pomeni, da JMeter lahko deluje na različnih operacijskih sistemih, ki podpirajo več nitnost. Kot sem se prepričal deluje vsaj na operacijskih sistemih Unix (Solaris, Linux, Fedora, Ubuntu), Windows (98, NT, XP, Vista in 7) in OpenVMS Alpha 7.3+.

Sestavni del orodja JMeter so tudi razširitve. Najbolj pomembne in najbolj uporabne razširitve so za predstavitev, analizo dobljenih rezultatov in časovno generiranje istočasnih navideznih uporabnikov.

Za izvedbo razširitev testnega orodja JMeter moramo imeti nameščeno Java JDK verzije 1.5 ali več. Nameščanje orodja JMeter na odjemalčevih računalnikih je enostavno. Odvisno od potreb lahko JMeter poženemo preko grafičnega uporabniškega vmesnika ali preko orodne vrstice.

Na prvi pogled izgleda JMeter kot brskalnik. Čeprav lahko v njem pregledujemo spletne strani pa ne podpira vseh tehnologij tipičnih brskalnikov. Za razliko od brskalnikov recimo ne podpira JavaScript-a, ki se danes napogosteje uporablja pri programiranju spletnih aplikacij na strani odjemalca. Zato morajo poskrbeti testerji, ki morajo na roke prepisati ustrezne parametre v razpoložljive komponente.

2.2.1 Osnovne komponente testnega orodja

Preden začnemo z opisovanjem osnovnih komponent testnega orodja, moramo definirati razlike med realnim in navideznim uporabnikom. Pri avtomatiziranem testiranju, namreč lahko uporabljamo le navidezne uporabnike.

Realni uporabniki so uporabniki, ki imajo čas za razmišljanje med zahtevami. Ko ljudje uporabljajo aplikacijo običajno preberejo ponujene opcije in razmislijo o tem, kaj bodo izbrali, da dobijo željeno informacijo. Zaradi tega je zaporedje pritiskov na gumbov pogosto enako. Poleg tega se praktično nikoli ne zgodi, da uporabniki istočasno izvajajo enak tok dogodkov. Tudi prekinitve pri izbirani opciji niso vedno iste. Pri navideznih uporabnikih čas zakasnitev in tok dogodkov nastavljajo osebe, ki pripravljajo testne scenarije (testerji), zato čas ni nikoli isti kot pri realnih uporabnikih, saj scenarij pri navideznih uporabnikih ni tako fleksibilen. Več o časovnih zakasnitvah je opisano v točki 4.2.

Osnovni komponenti orodja JMeter sta **plan testiranja** (*ang. Test Plan*) in **delovna površina** (*ang. WorkBench*). Vsak testni scenarij vsebuje ti dve osnovni komponenti.

Plan testiranja je koren testnega scenarija. Drevesu nato dodajamo testne aktivnosti navideznih uporabnikov, ki se bodo izvajale pri zagonu testnega scenarija. Vse podkomponente plana testiranja se avtomatsko shranjujejo pri shranjevanju testnega scenarija.

Prva podkomponenta plana testiranja so **Uporabniki** (*ang. Threads (Users)*). Ta je namenjena za določanje časovne porazdelitve in števila navideznih uporabnikov.

Obstaja več različnih komponent za generiranje navideznih uporabnikov, kar je prikazano na sliki 2.

Slika 2: Opis komponent za generiranje uporabnikov.

Skupina (*ang. Thread Group*) je standardna komponenta, ki v enakomernih časovnih razmikih generira virtualne uporabnike. V obdobju od začetnega do končnega časa generira določeno število uporabnikov, ki so enakomerno časovno porazdeljeni.

Skupina diskretnih vrednosti (*ang. Stepping Thread Group*) je dodatna komponenta za stopničasto generiranje navideznih uporabnikov. Določimo skupno število navideznih končnih uporabnikov, ki ga razdelimo na različno velike skupine, ki se bodo zaganjale istočasno. Tako definiramo postopno obremenjevanje ali razbremenjevanje.

Skupina zveznih vrednosti (*ang. Ultimate Thread Group*) je dodatna komponenta za skupinsko generiranje navideznih uporabnikov. Število uporabnikov je enako kot prej razdeljeno po skupinah, vendar se znotraj skupin funkcijsko določa porazdelitev

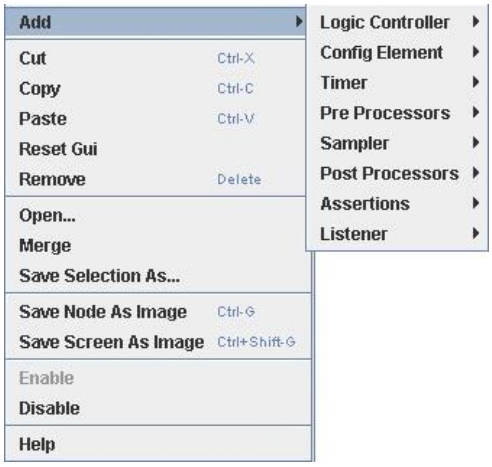
obremenitve in razbremenitve. Najpogosteje se uporablja linearna obremenitev in razbremenitev.

Delovna površina je koren začasno shranjene testne komponente. Uporablja se za začasno shranjevanje posebnih komponent, ki se ne izvajajo pri zagonu testnega scenarija in se zato tudi ne shranjujejo s testnim scenarijem. Da se ne bi izgubile, jih shranimo v testni scenarij in običajno onemogočimo njihovo izvajanje. To storimo zato, da se vsebina shrani ob planu testiranja**.**

Najbolj uporabne komponente, izven plana testiranja so **sestavni elementi** (*ang. Non-Test Elements*), **zrcalni spletni strežnik** (*ang. HTTP Mirror Server*), **posredniški spletni strežnik** (*ang. HTTP Proxy Server*) ter **prikaz značilnosti** (*ang. Property Display*).

Slika 3: Opis začasnih komponent v delovni površini (`WorkBench`).

Zrcalni spletni strežnik je enostaven spletni strežnik, ki snema HTTP zahteve in je kot tak uporaben za predstavitev zahtev.
Posredniški spletni strežnik je namenjen snemanju aktivnosti, ki sestavljajo testni scenarij.
Prikaz značilnosti omogoča dodatne sistemske nastavitve za orodje JMeter.
Ostale komponente so standardne in enake tako v testnem planu in delovnem prostoru. Te so namenjene predvsem za gradnjo testnih scenarijev (slika 4).



Slika 4: Prikaz standardnih komponent.

Logični krmilnik (*ang. Logic controler*) omogoča dodajanje logičnih komponent (`For`, `Foreach`, `If`, `While`, `Random`, `Throughtput`). Uporabljamo jih za odločanje, o izpolnjevanju pogojev za pošljanje zahtevkov strežniku.

Konfiguracijski element (*ang. Config element*) omogoča dodajanje komponent za vzpostavitev povezave z aplikacijo ter vnos različnih elementov v aplikacijo: piškotkov (`ang. Cookies`), predpomnilnika (`ang. Cache`), glave (`ang. Header`), avtorizacije, branja podatkov iz datoteke, definicij naključnih spremenljivk in podobno). Te se dodajajo skupaj z logičnimi kontrolami.

Časovnik (*ang. Timer*) omogoča dodajanje časa razmišljanja oziroma zakasnitev (*ang. thinking time*) pri navideznih uporabnikih.

Predobdelovalec (*ang. Pre Processors*) omogoča dodajanje in urejanje podatkov pred definiranjem pošiljanja zahteve.

Vzorčevalec (*ang. Sampler*) je meni za definiranje vzorca zahtev, ki jih bomo poslali. (SMTP, HTTP, FTP, JDBC, TCP, ipd…).

Poobdelovalec (*ang. Post Processors*) je namenjen obdelavi podatkov iz odgovorov v skladu z regularnimi izrazi in podobnim (*ang. Reg ex, bsf* in *bean shell*).

Trditev (*ang. Assertion*) služi za preverjanje vrnjenih podatkov, najpogosteje za preverjanje napak pri zahtevkih.

Poslušalec (*ang. Listener*) se uporablja za analizo in prikaz dobljenih podatkov.

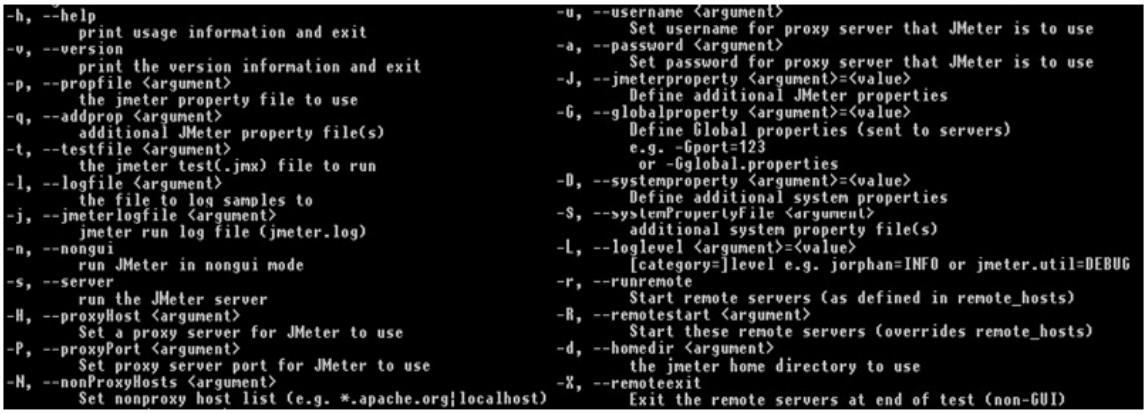
JMeter pokriva širok obseg performančnosti in vsebuje različne možnosti testiranja. Je pregleden, lahek za uporabo in enostaven za nameščanje. Žal ima tudi številne pomanjklivosti kot so nemožnost primerjanja dobljenih podatkov iz več testiranj, ne najbolj pregleden vizualni prikaz podatkov in pomanjkanje nekaterih gonilnike za izvajanje testnih scenarijev.

2.2.2 Način izvajanja testnih scenarijev

JMeter omogoča testiranje aplikacije na tri načine, od katerih sta dva osnovna, tretji način pa je kombinacija prvih dveh.

Prvi način je testiranje preko grafičnega uporabniškega vmesnika (*ang. GUI mode*), ki je najbolj pregleden, najlažji za uporabo in najlažji za analizo. Po drugi strani je manj robusten in bolj požrešen. Ta način se najpogosteje uporablja za pripravo testnih scenarijev. Pogajanje okolja JMeter v tem načinu je enostavno, saj samo poženemo skripto, ki se nahaja v (`bin\jmeter.bat`).

Drug način je uporaba v tekstualnem načinu preko orodne vrstice, ki se uporablja pri izvajanju testiranja na delovni postaji (*ang. console mode*) [2]. Postavitev v tem načinu se izvaja na naslednji način: v orodni vrstici operacijskega sistema se postavimo v (`.\bin`). Poženemo ukaz (`jmeter -n`), ki zažene JMeter v tekstualnem načinu. V primeru, da potrebujemo pomoč, lahko uporabimo ukaz (`jmeter -h`). S tem se bodo na zaslonu izpisali vsi ukazi za ta način, kot je to prikazano na sliki 22.



Slika 22: Seznam ukazov v tekstualnem načinu.

Tretji način je najbolj zahteven. Testni scenarij se izvaja istočasno na več računalnikih, od katerih je eden glavni, vsi ostali pa so podrejeni. Ta način imenujemo distribucijski način. Gospodarjev računalnik uporablja grafični uporabniški način in ima celoten nadzor nad izvajanjem testa. Podrejeni računalniki (sužnji) v strežniškem načinu dobivajo ukaze iz gospodarjevega računalnika in pošiljajo zahteve testnemu strežniku. Preden se lotimo namestitve orodja JMeter na vseh računalnikih za uporabo v distribucijskem načinu moramo preveriti naslednje nastavitve:

- če je požarni zid pri vseh računalnikih izklopljen,
- če so vsi računalniki v istem omrežju,
- če je strežnik v istem omrežju (ni obvezno),
- če lahko vsaka instanca orodja JMeter dostopa do testnega strežnika,
- če imamo na vseh računalnikih nameščeno isto verzijo orodja ter
- če imamo na vseh računalnikih nameščeno isto verzijo Jave (1.5 ali novejšo).

3. Testni scenariji

3.1 Zbiranje informacij za sestavljanje testnega scenarija

Testni scenarij in testno okolje igrata ključno vlogo pri izvajanju avtomatiziranih in performačnih testiranj. Preden se lotimo sestavljanja testnih scenarijev moramo zbrati potrebne informacije, da bi se naprej lažje odločali in odstranili dvoumosti. Zato je potrebno pridobiti informacije, ki so predstavljene v nasledjih razdelkih.

3.1.1 Uporabniki aplikacije

Preden začnemo sestavljati testni scenarij moramo najprej vedeti, kdo bo uporabljal aplikacijo. Ko dobimo grobo sliko o aplikaciji lahko določimo tipe uporabnikov, ki bodo uporabljali aplikacijo. Tipe uporabnikov lahko razdelimo v dve skupini:

- manj spretni uporabniki ter
- bolj spretni uporabniki,

Med prve sodijo uporabniki, ki so malo časa pred računalnikom. Za razliko od njih so bolj spretni uporabniki pretežno veliko časa pred računalnikom. Tipično so to razvijalci aplikacij in igralci računalniških igric. Spretnost uporabnikov je eden od ključnih faktorjev pri nastavljanju zakasnitev.

3.1.2 Namen aplikacije

Pri sestavljanju testnih scenarijev je potrebno vedeti tudi, čemu je aplikacija namenjena, kakšne podatke obdeluje ter obliko in velikost obdelanih podatkov. Na podlagi teh informacij lahko določimo osnovni tok dogodkov aplikacije, ki igra pomembno vlogo pri sestavljanju testnih scenarijev.

3.1.3 Uporabljeni protokoli

Za dostop do aplikacije moramo poznati protokol, ki ga aplikacija uporablja. Protokol se uporablja za varen dostop do aplikacije.

3.1.4 Način prenosa parametrov

Preden začnemo sestavljati testni scenarij je potrebno poznati tudi sestavo aplikacije in tehnologijo, ki jo uporablja. Pri spletni aplikaciji moramo vedeti, kako in kateri parametri se prenašajo, da bi zagotovili nemoteno preusmerjanje oziroma sprehajanje po aplikaciji. Pri prenašanju parametrov moramo uporabiti določene elemente, ki jih ponuja JMeter. Pri tem moramo vedeti predvsem to, kdaj se spreminjajo. Zato moramo narediti dobro analizo, ker je veliko možnosti za napake. Poleg tega je aplikacija lahko omejena na različne seje na fizičnih lokacijah, kar pomeni, da iz ene fizične lokacije lahko dostopa samo en uporabnik. V tem primeru je generiranje navideznih uporabnikov iz enega strežnika težko doseči.

3.1.5 Izvajanje JavaScript-a na strežniku

Funkcionalnosti aplikacije se lahko izvajajo na strežniku ali odjemalcu. Najbolj uporaben jezik za ta namen je JavaScript. Ker JMeter ne loči kode skripte od HTML-ja, mora te funkcionalnosti tester sam posebej vstaviti v testni scenarij. To zahteva nadomeščanje JS kode v spremenljivkah in pošiljanje zahtevanih informacij v zahtevi. Tudi pri analizi JS kode moramo biti pozorni na format, ki ga pošiljamo na strežniku.

3.1.6 Največje število uporabnikov

Naročnik mora pri naročilu testiranja aplikacije poznati pričakovano število uporabnikov aplikacije. V primeru, da je število istočasnih uporabnikov majhno (do deset uporabnikov),

se izdelave performančnih avtomatiziranih testov ne splača izvajati. Če pa je število uporabnikov večje od deset uporabnikov, je to testiranje nujno za nemoteno delovanje aplikacije.

3.1.7 Mesto dostopa

Aplikacija je lahko namenjena za interno uporabo znotraj lokalnega omrežja, za uporabo med poslovnim partnerji ali za internetno uporabo. Lokalno omrežje lahko podpira prenos podatkov celo do 10,000 Mbitov/sek, v drugem primeru pa je hitrost odvisna od podjetja in se lahko zgodi da je hitrost prenosa podatkov le 256 Kbitov/sek. Simulacijo globalnega omrežja lahko naredimo tudi znotraj lokalnega omrežja, če omejimo hitrost. Podobno je tudi pri internetni povezavi ki je odvisna od podjetja. Odzivnost omrežja je pomemben faktor, ker določa zakasnitev pri pošiljanju podatkov od ene do druge točke.

3.1.8 Osnovni tok dogodkov in najpogostejši testni scenarij

Preden se lotimo sestavljanja testnih scenarijev moramo poznati tudi osnovni tok dogodkov. Na podlagi tega je treba sestaviti še seznam najbolj pogostih dogodkov uporabnikov.

3.1.9 Zakasnitve

Zakasnitev je odvisna od tega v katero skupino sodijo uporabniki, ki jim je aplikacija namenjena, torej če imajo izkušnje ali so navajeni uporabe podobnih aplikacij. Zakasnitev je odvisna tudi od ponujenih opcij za izbiro na posamezni strani.

3.1.10 Ustrezno testno okolje

Po zbranih informacijah o testnih scenarijih moramo pridobiti tudi informacije o testnem okolju, ki mora biti podobno (ali celo enako) naročnikovemu okolju. Popolno skladnost je težko zagotoviti, ni pa to nemogoče. Da bi določili delež skladnosti moramo primerjati naslednje podatke:

- število strežnikov (število fizičnih ali navideznih strežnikov na določeni plasti),
- strategijo za izravnavo obremenitve,
- strojno opremo (tip in število CPE, pomnilnik in število omrežnih vmesnikov).
- programsko opremo (standardni razvoj programske opreme, deli aplikacije, ki bodo performančno testirani),
- sestavne dele aplikacije (opis aplikacijskih sestavnih delov na tej strežniški plasti), ter
- zunanje povezave (povezave do zunjih sistemov) spletna aplikacija lahko uporablja

zunanjo povezavo za zagotavljanje podatkov iz spletnih servisov, v tem primeru je testiranje in primerjava povezav neizbežna).

3.2 Opis in sestavljanje testnih scenarijev

V nadaljevanju kot je opisana sestava testnih scenarijev.

3.2.1 Skupne komponente pri sestavljanju testnih scenarijev

Koncept testiranja performančnosti je neodvisen od testnega scenarija, zato testni scenarij vsebuje:

- število uporabnikov,
- zahteve (*ang. request*)
- odgovore (*ang. response*)
- preveritev odgovorov ter
- prikaz dobljenih podatkov.

Vsak testni scenarij v našem orodju vsebuje dve osnovni komponenti. `Test Plan` in `WorkBench`. Začetni komponenti sta neodvisni od testnega scenarija, vse ostale komponente pa so odvisne od namena testnega scenarija.

3.2.2 Opis testnih scenarijev

V nadaljevanju bom opisal, sestavo testnega scenarija za testiranje:

- spletnih aplikacij - za opis testiranja spletnih aplikacij sem uporabil aplikacijo ePoslovanje, ki je razvita v .NET tehnologiji in uporablja podatkovno bazo MS SQL.
- podatkovnih strežnikov - za opis testiranja podatkovnega strežnika MS SQL Server 2005 [17] sem uporabil podatkovno bazo z več kot dvajset tabelami.
- strežnikov, ki uporabljajo FTP protokole - za opis, sestavljanje in analiziranje strežnikov, ki uporabljajo FTP protokolov, sem uporabil enostavno tekstovno datoteko in FileZilla FTP strežnik [18].

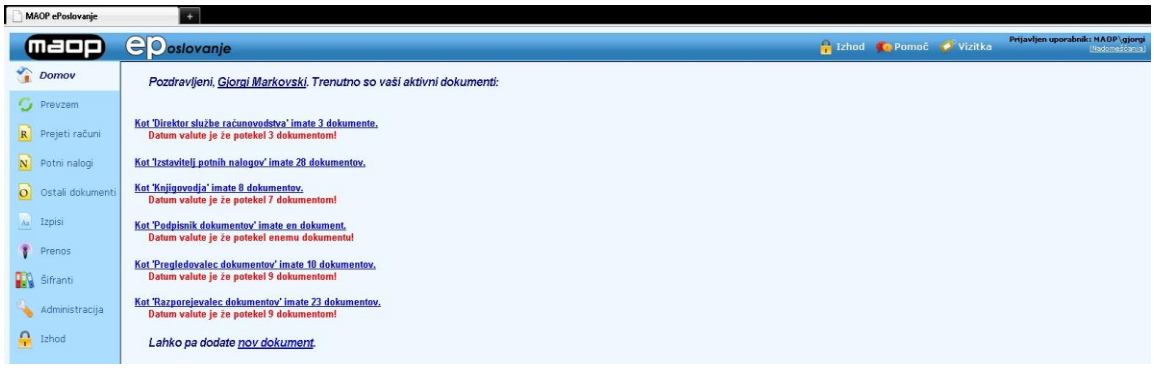
Testiranje bom izvedel za različna števila uporabnikov, najprej za 10, nato 20, 50 ter vse do 1500 uporabnikov. Po vsakem koraku bom analiziral dobljene podatke in se odločal, če še lahko povečujem število uporabnikov ali pa sem dosegel največje dovoljeno število istočasnih uporabnikov.

3.2.3 Testiranje spletnih aplikacij

Spletna aplikacija ePoslovanje [19] je namenjena evidentiranju podatkov o prispelih dokumentih na podlagi skeniranih dokumentov, posebej prispelih ali prenesenih računov. Vse podatke se lahko izpisujejo na različne načine, od tiskanja seznama prispelih

dokumentov (po različnih uporabniško določenih filtrih glede na osnovne podatke računa ali podatke v postavkah računa) do izpisov dopisov o zavrnitvi ali izpisu skeniranega dokumenta.

Pod vrhom glavnega okna je vrstica z **menijem** programa. Prikazana so imena možnih opravil, ki so odvisna od pravic določenega uporabnika, ki je trenutno prijavljen v aplikacijo ePoslovanje. Prepovedana opravila se v meniju ne izpišejo. V sredinskem delu pozdravnega okna se izpišejo informacije o čakajočih dokumentih z dodatnim opozorilom v primeru, ko je datum valute že potekel (rdeča barva) oziroma bo potekel v kratkem (oranžna barva). Seznami dokumentov se odprejo s klikom na obvestilo o čakajočih dokumentih. Začetna stran aplikacije je prikazana na sliki 5.



Sliki 5: Začetna stran aplikacije ePoslovanje.

Opcije glavnega menija aplikacije ePoslovanje:

Domov - Pozdravno okno z informacijami o dokumentih, ki čakajo na obdelavo.

Prevzem - Prevzem podatkov iz datoteke, ki jo pripravi izvajalec skeniranja.

Prejeti računi - Prikaz vseh prejetih računov z oznako trenutnega statusa.

Ostali dokumenti - Forma za obdelavo prejetih dokumentov.

Prenos - Prenos potrjenega dokumenta v aplikacijo MCR-PD [20].

Šifranti - Pregled šifrantov, ki so povzeti iz aplikacije MCR ter nastavitve šifrantov, ki so potrebni za delo v aplikaciji ePoslovanje.

Administracija - Nastavitev uporabnikov aplikacije in njihovih vlog.

Izhod - Izhod iz aplikacije.

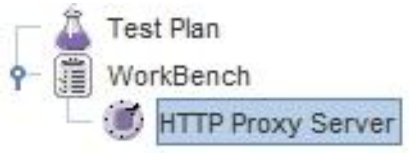
Najprej določimo osnovni tok dogodkov in tiste tokove dogodkov, ki se bodo najpogosteje uporabljali. Testna aplikacija je namenjena evidentiranju podatkov o prispelih dokumentih, kar je očitno osnovni tok dogodkov. Testni scenarij bo vseboval pregled in spreminjanje podatkov. Zaporedje aktivnosti testnega scenarija za tok dogodkov brez napak je naslednji:

- Dostop do željenega naslova.
- Prijava v aplikacijo preko IWAP protokola.

- Vnos uporabniškega imena in gesla.
- Izbira vrstice iz menija `Domov`.
- Izbira podmenija `Kot 'Razporejevalec dokumentov imate x dokumentov'.`
- Izbira enega dokumenta izmed ponujenih v tabeli.
- Sprememba podatkov znotraj dokumenta.
- Izbira in pošiljanje v pregled.
- Vnos potrebnih podatkov (`Stroškovno mesto`, `Naslovnik`, `Razlog/Opomba`).
- Pritisk na gumb `Izvedi` (aplikacija se vrne na začeno stran vrstice iz menija `Domov`).
- Izbira podmenija `Kot 'Pregledovalec dokumentov imate x dokumentov'.`
- Izbira dokumenta, ki je bil poslan v pregled.
- Sprememba podatkov.
- Vrnitev k razporejevalcu.
- Vnos potrebnih podatkov (`Stroškovno mesto`, `Naslovnik`, `Razlog/Opomba`).
- Pritisk na gumb `Izvedi` (aplikacija se vrne na začeno stran vrstice iz menija `Domov`).
- Odjava iz aplikacije (pritisk na gumb `Izhod`).

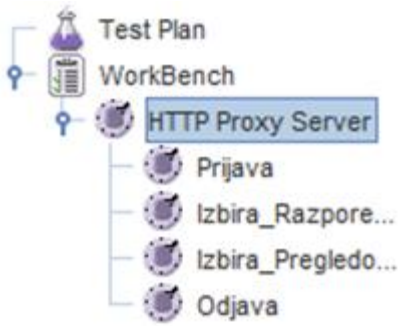
Testni scenarij dodeli dokument nekomu, ki ga pregleda, spremeni podatke in vrne nazaj na pregled k razporejevalcu.

Po definiranju zaporedja aktivnosti testnega scenarija začnemo s sestavljanjem. Hiter način za zajemanje HTTP in HTTPS strani je snemanje vsake zahteve na strežnikih. Zato sem uporabil komponento `HTTP Proxy Server`, ki se uporablja samo za HTTP in HTTPS. Ko aplikacija uporablja HTTPS protokol, ta zahteva certifikat, ki ga moramo prevzeti iz brskalnika, da bi JMeter odprl SSL povezavo. `HTTP Proxy Server` je na voljo v komponenti `Workbench`. `HTTP Proxy Server` zapisuje vse zahteve za aplikacijo. Poleg zahtev uporabnika zapisuje tudi zahteve, ki jih brskalnik naredi kot sodobni brskalnik. Posredniški spletni strežnik ne zapisuje samo zahteve ciljnega strežnika. To nam dovolijo spletni filtri v današnjih brskalnikov, ki pošiljajo zahteve samo o ciljnem strežniku. V nasprotnem primeru pa bi bile nepotrebne zahteve vidne pri zapisovanju. Komponenta `HTTP Proxy Server` v orodju JMeter se nahaja v podmeniju `Non Test Elements`. Po dodajanju `HTTP Proxy Server` se bo prikazala kot podkomponenta delovnega prostora kot je to prikazano na sliki 6.



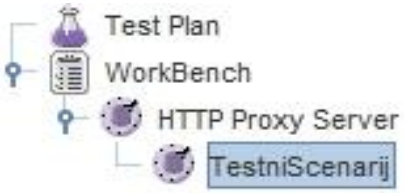
Slika 6: Dodajanje `HTTP Proxy Server` *komponente.*

Preden začnemo z nastavitvami posredniškega strežnika bomo dodali še komponento `Simple Controller`, ki se nahaja v podmeniju `Logic Controller` in jo preimenujemo v `TestniScenarij` kot je prikazano na sliki 6. Odvisno od velikosti lahko testni scenarij razdelimo na več sklopov, pri čemer bi vsak sklop predstavljal eno funkcionalnost. Lahko naredimo štiri sklope: Prijava, Izbira_Razporejevalca, Izbira_Pregledovalca, ter Odjava. Vse to je prikazano na sliki 7.



Slika 7: Funkcionalnost aplikacije, ki je razdeljena na več sklopov.

Sklope lahko vsak tester definira drugače, ker nima točno določenih omejitev. Zato sem se odločil za generiranje testnega scenarija `TestniScenarij` kot ene celote kot je prikazano na sliki 8.



Slika 8: Dodajanje testnega scenarija.

Sedaj se vrnemo nazaj na `HTTP Proxy Server` komponento in jo nastavimo:

> Vrata (`Port`): 8085 je številka vrat posredniškega strežnika, ki ga moramo nastaviti tudi v brskalniku.
>
> "Prevara" HTTP poskusa (`Attempt https Spoofing`): izberemo, če imamo SSL povezavo ali če aplikacija zahteva avtentikacijo.
>
> "Prevara" samo URL usklajevanje (`Only spoof URLs matching`): vnesemo URL, na katerem aplikacija zahteva avtentikacijo. (Ni mogoče, če smo izbrali

Attempt HTTPS Spoofing).

Ciljni krmilnik (Target Controller): izberemo (HTTP Proxy Server > TestniScenarij), kjer se bodo zahteve zapisale v že definirani TestniScenarij.

Združevanje (Grouping): izberemo (Do not group samplers) možnost za združitev posameznih zahtevkov v skupini z razmejevalniki.

Zajemanje HTTP glave (Capture HTTP Headers): izberemo, da posname HTTP glavo.

Dodaj trditev (Add Assertions): izberemo, da doda komponento za preverjanje odgovora.

Usklajevanje regularnih izrazov (Regex matching): izberemo za uporabo regularnih izrazov pri prenosu spemenljivk.

Tip (Type): postavlja tip zahtev, ki jih bo generiral.

Avtomatska preusmeritev (Redirect Automatically): izberemo avtomatsko preusmeritev odgovora.

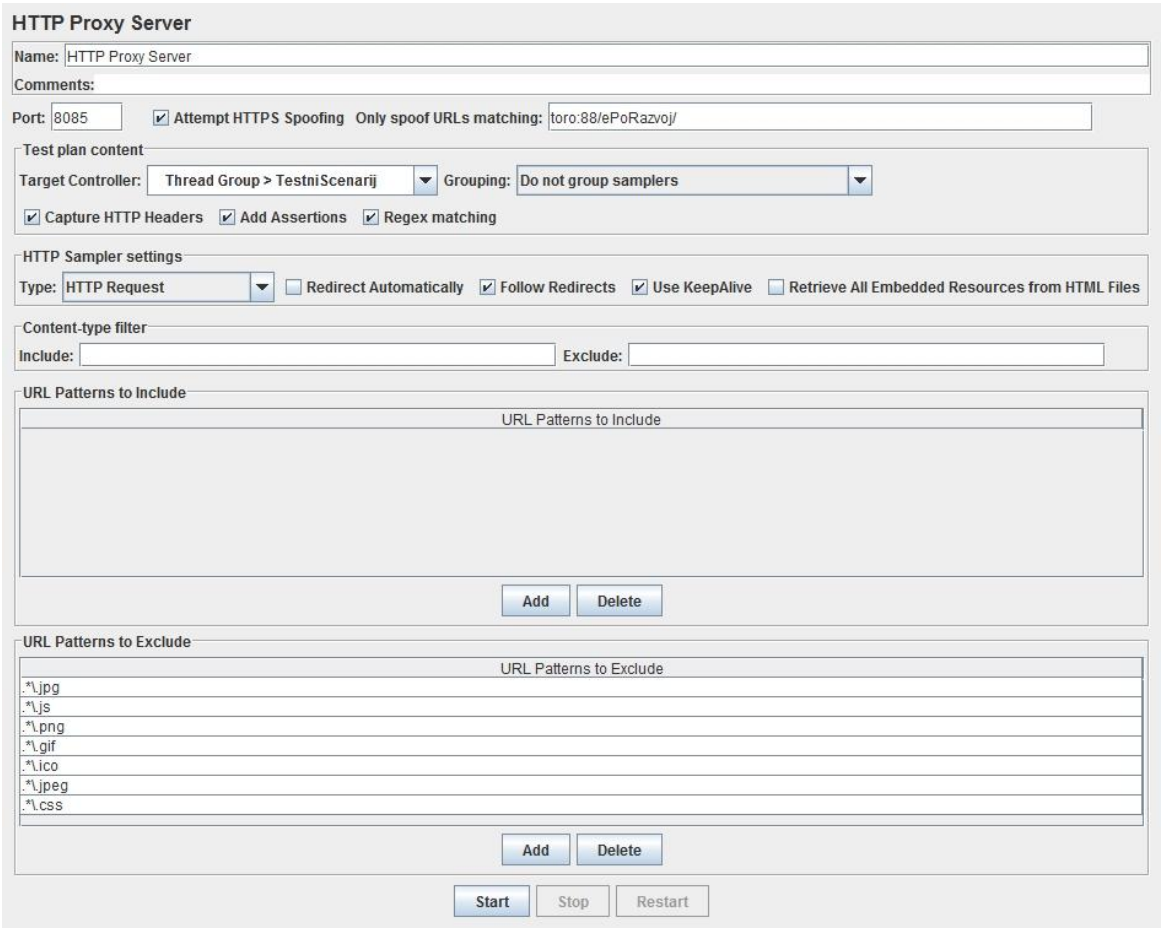
Sledljivost preusmeritev (Follow redirect).

Uporabi ohranjanje (Use KeepAlive): izberemo preverjanje, če povezava med strežnikom in orodju deluje.

Vključitev vzorcev URL (URL Patterns to Include): vnesemo regularne izraze za zahteve, ki jih hočemo posneti. To uporabljamo, ko imamo več neželenih kot želenih zahtev z določenimi končnicami.

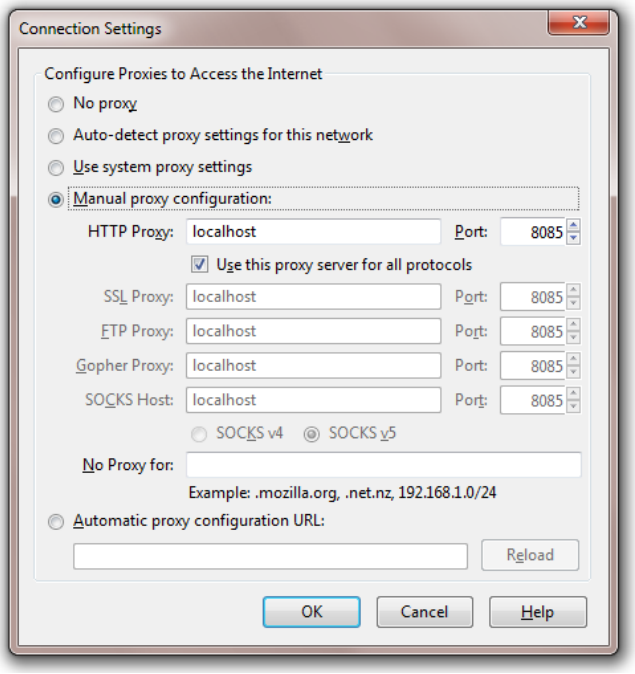
Izključitev vzorcev URL (URL Patterns to Exclude): vnesemo regularne izraze za slikovne formate, ki jih nočemo posneti v testni scenarij. To so sami naslovi, ki kažejo slike, ki se ne bodo shranile v testni scenarij. JS in CSS določata dinamičen izgled aplikacije in izvajata določene funkcionalnosti. Obremenitev aplikacije, je ob uporabi JS in CSS zanemarljiva, ker se izvaja na odjemalčevem računalniku. Slike s končnicami GIF, TIFF, JPEG, PNG, BMP, ICO se prenesejo samo pri prvem obisku in se nato shranijo v predpomnilnik brskalnika. Za vsako nadaljnje dostopanje se te naložijo iz predpominlnika. Če sestavljamo testni scenarij samo za prve obiske uporabnika, ta del pustimo prazen, v ostalih primerih dodamo vse končnice.

Nastavitev posredniškega spletnega strežnika je prikazana na sliki 9.



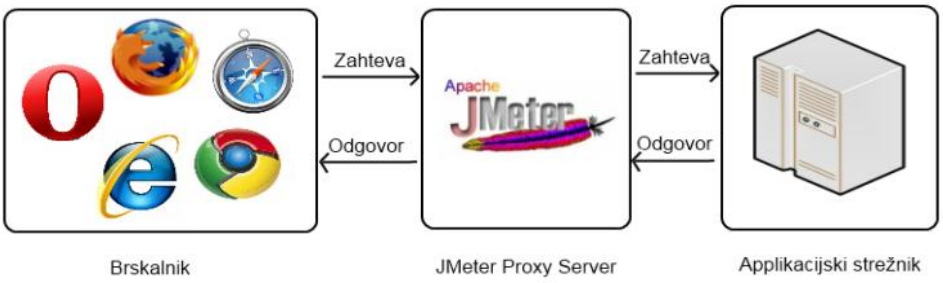
Slika 9: Nastavitev HTTP Proxy strežnika.

Sledi nastavitev brskalnika, da lahko dostopa do posredniškega strežnika orodju JMeter. Najprej nastavimo vrata (v tem primeru 8085) kot smo jih definirali v `HTTP Proxy Server`. Nastavitev posredniškega strežnika v Mozilla Firefox-u najdemo v orodni vrstici `Tools` podmenija `Options` v zavihku `Advanced`, nato izberemo zavihek `Network` in v `Settings` nastavimo vrednosti, kot so prikazane na sliki 10.



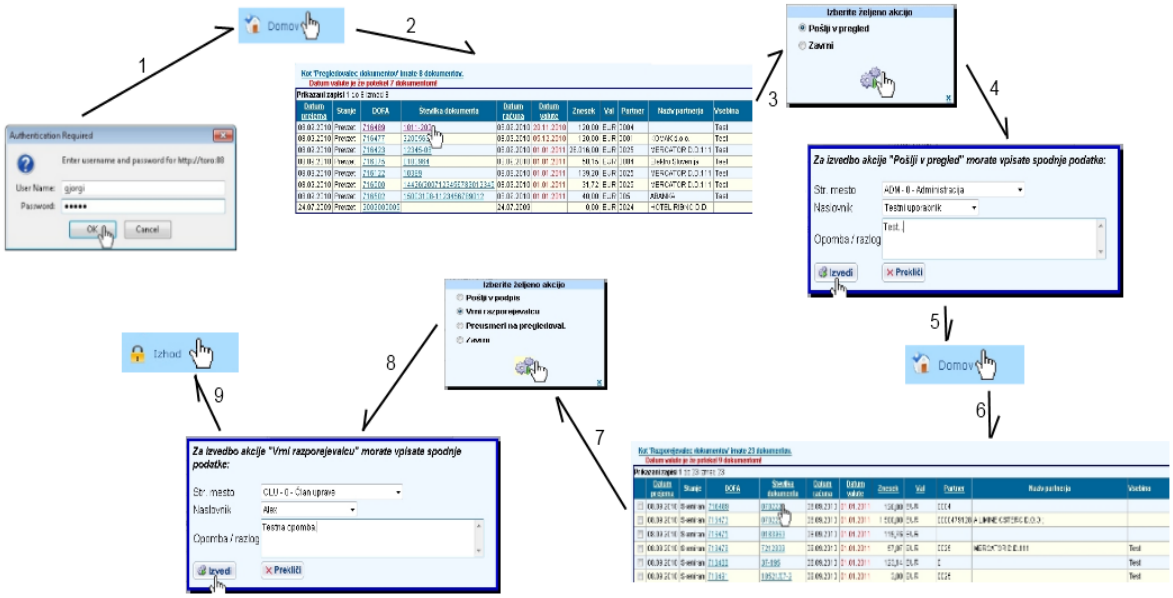
Slika 10: Nastavitev Mozille Firefox za JMeter Proxy strežnik.

Tako je postavljeno okolje za zapisovanje zahtev in odgovora v komponenti `TestniScenarij`. Okolje predstavlja povezavo med brskalnikom Mozilla Firefox, orodjem JMeter posredniškim strežnikom in aplikacijo ePoslovanje. Struktura povezanosti je predstavljena na sliki 11.



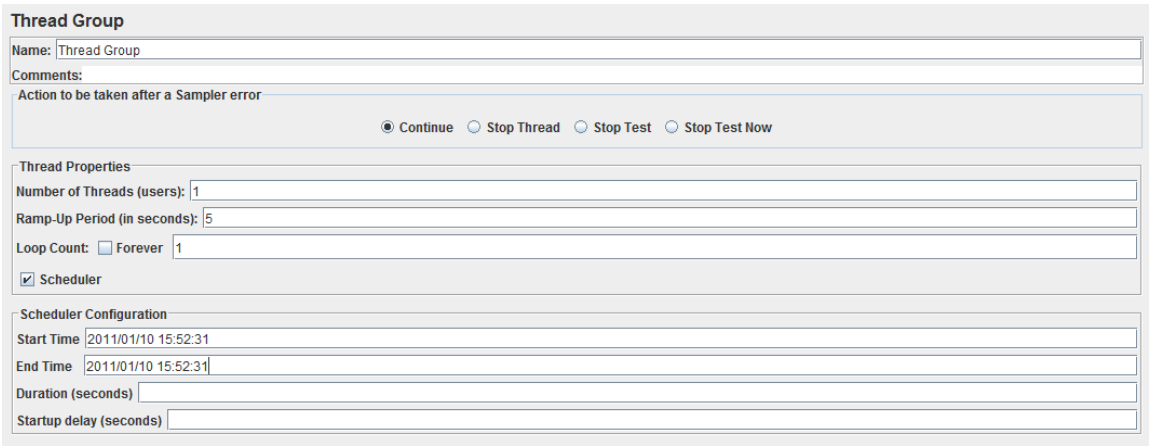
Slika 11: Struktura testnega sistema.

Sedaj je vse pripravljeno za zapisovanje zahtev in odgovorov v komponenti `TestniScenarij`. V JMeter na komponenti `HTTP Proxy Server` pritisnemo gumb `Start`. Od tega trenutka naprej bo vse, kar se bo izvajalo v brskalniku, zapisano tudi v komponenti `TestniScenarij`. Vtipkamo URL naslov aplikacije, ki jo želimo testirati. Prijavimo se v aplikacijo in izvedemo zaporedje aktivnosti, ki smo ga definirali na začetku za aplikacijo ePosolovanje. To zaporedje aktivnosti je prikazano na sliki 12.



Slika 12: Tok dogodkov za moj testni scenarij.

Snemanje zahtev prekinemo z gumbom `Stop` v `HTTP Proxy Server`.
Na koncu snemanja moramo prenesti testni scenarij v komponento `Test Plan`, kamor ga bomo shranili. Prva podkomponenta tega je generator navideznih uporabnikov. Postavimo se na `Test Plan` v `Threads (Users)` na podmeni `ThreadGroup`. Nastavimo število navideznih uporabnikov v polju `Number of thread` na 1, čas za generiranje uporabnikov v polju `Rump-Up Period` nastavimo na 5 sekund in število ponavljanj testnega scenarija v polju `Loop Count` na 1. Lahko nastavimo tudi razporejevalnik, če želimo, da se plan testiranja požene ob določenem času v polju `Scheduler` vnesemo datum in čas. Nastavitve so prikazane na sliki 13.



Slika 13:Nastavljanje uporabnikov.

Ta testni scenarij shranimo kot `TestniScenarij`.

JMeter vsak testni scenarij shrani v datoteko s končnico `.jmx` in ta je zapisana v XML

jeziku. Zato še enkrat ponovimo zgornji scenarij in ga shranimo kot `TestniScenarij1`.

Testna scenarija `TestniScenarij` in `TestniScenarij1` nam nič ne povesta o tem, kateri parametri so dinamični in kateri statični. Za določitev spremenljivosti parametrov moramo primerjati oba testna scenarija. Za primerjavo sem uporabil program WinMerge [21], s katerim hitro ugotovimo, kateri parametri so različni. Primerjanje vrednosti parametrov je prikazano na sliki 13.

```
<elementProp name="__VIEWSTATE" elementType="HTTPArgument">
  <boolProp name="HTTPArgument.always_encode">true</boolProp>
  <stringProp name="Argument.name">__VIEWSTATE</stringProp>
  <stringProp name="Argument.value">/wEPDwUKMTcwNTkwODQ2NQ9kFgJmD2QWAgIDD2QWCAIPDw8WBh4JRm9yZUNvbG
  <stringProp name="Argument.metadata">=</stringProp>
  <boolProp name="HTTPArgument.use_equals">true</boolProp>
</elementProp>
```

```
<elementProp name="__VIEWSTATE" elementType="HTTPArgument">
  <boolProp name="HTTPArgument.always_encode">true</boolProp>
  <stringProp name="Argument.name">__VIEWSTATE</stringProp>
  <stringProp name="Argument.value">/wEPDwUKMTcwNTkwODQ2QW9kFgJmD2QWAgIDD2QWCAIPDw8WBh4JRm9yZUNvbG
  <stringProp name="Argument.metadata">=</stringProp>
  <boolProp name="HTTPArgument.use_equals">true</boolProp>
</elementProp>
```

Slika 14: Določanje spremenljivke v JMeter.

Na sliki 14 vidimo XML kodo, ki jo je JMeter generiral na podlagi zahtev, ki smo jih posneli preko posredniškega spletnega strežnika. Vidimo, da se spremenljivka `__VIEWSTATE` spreminja pri vsaki prijavi, saj območje vrednosti ni točno definirano. Vrednost spremenljivke `__VIEWSTATE` se pri preusmeritvah prenaša na naslednjo stran, kar pomeni, da jo moramo prebrati iz prejšnjega odgovora. Na podlagi tega sestavimo seznam imen parametrov, ki imajo različne vrednosti. Pri razvoju velikih aplikacij razvijalci vedno uporabljajo različna orodja, ki na podlagi njihovih zahtev generirajo programsko kodo. Enako je s spremenljivko `__VIEWSTATE` iz aplikacije ePoslovanje, ki je prikazana na sliki 13. Aplikacija je razvita z orodjem Visual Studio 2008 [22]. Zato se lahko zgodi, da tudi razvijalci ne vedo točno, kako so poimenovane določene spremenljivke, ki jih je generiralo orodje. Sodelovanje z razvijalci aplikacije je sicer koristno, vendar nam v takih primerih ne pomaga preveč. Za razumevanje imen parametrov si lahko pomagamo z Mozillovem dodatkom Firebug [23].

Firebug omogoča v brskalniku Firefox odpravljanje napak, urejanje in spremljanje vseh spletnih dokumentov (CSS, HTML, DOM in JS). Ima spletna razvojna orodja, s katerimi lahko vidimo parametre, ki opisujejo polja spletne aplikacije v realnem času. Vrstni red v Firebug-u žal ni vedno tak kot ga je aplikacija določila, zato se pri generiranju regularnih izrazov ne uporablja. V seznam bomo dodali še stolpca za ustrezna polja aplikacije.

Glave zahtevkov (v komponenti `HTTP Proxy Server` smo jih označili pred snemanjem zahtevkov) lahko preverimo z uporabo Mozilovga dodatka Live HTTP Headers [24]. Primerjamo glave zahtevkov in glave v tem programu. V primeru, da je med njimi razlika, lahko še enkrat posnamemo samo te zahteve in jih nadomestimo z že obstoječimi.

Nato pogledamo, kaj se je zapisalo v zahteve. Če se vse zahteve pošiljajo na isti strežnik (kot je to v mojem primeru za aplikacijo ePoslovanje) na komponenti `TestniScenarij` v podmeni `Config Element` dodamo komponento `HTTP Request Defaults`. Pri vseh zahtevah pregledamo, katera polja so enaka. Polja `Server Name or IP`, `Port`,

Protocol (default http) in Content encoding so ponavadi pri vseh zahtevah enaka, razen v primeru, če aplikacija dostopa do ostalih strežnikov. Sestavljanje testnih scenarijev izvajamo na testnem strežniku. Da bi pohiteli s postopkom prilagajanja performančnih testov na uporabnikovemu strežniku dodamo HTTP Request Defaults komponento. Enaka polja vseh zahtev prepišemo v HTTP Request Defaults komponento.

Aplikacija ePoslovanje uporablja IWAP za prijavo v aplikacijo. To pomeni, da moramo dodati komponento HTTP Authorization Manager, ki bo znala odgovoriti aplikaciji in bo vsebovala potrebne podatke o uporabniškem imenu in geslu. V Thread Group podmeni Config Element dodamo komponento HTTP Authorization Manager. Vnesemo URL naslov, na katerem aplikacija zahteva prijavo, uporabniško ime, geslo, domeno in področje.

Danes skoraj vsaka aplikacija vsebuje tudi parametre, ki se prenašajo kot piškotki ali URL parametri. Komponenta HTTP Cookie Manager poskrbi za pravilen prenos piškotkov in skritih URL parametrov. Dodamo jo v Thread Group podmeni Config Element z komponento HTTP Cookie Manager. Izberemo Clear cookies each iteration? in postavimo standardni format za piškotke na rfc2965 (določata strukturo piškotkov pri prenosu). Če želimo lahko tudi sami nastavimo začetne piškotke ali URL parametre.

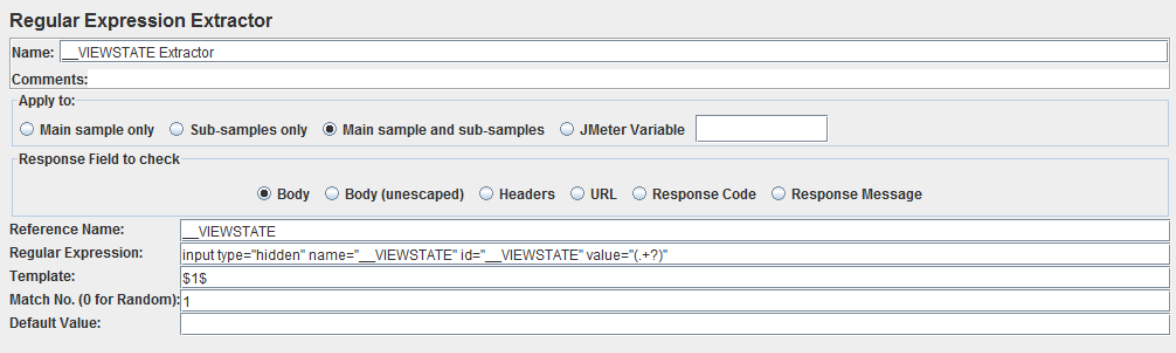
Naslednja stvar, ki jo moramo definirati, je nabor parametrov in od kje jih moramo prebrati. S tem smo definirali naključne spremenljivke znotraj določenega območja, ki jih lahko napovemo pred začetkom testnega scenarija v tekstualni datoteki. V orodju JMeter je priporočljivo, da imajo spremenljivke ista imena kot v programski kodi aplikacije. Za definiranje spremenljivk znotraj testnega scenarija se postavimo na Thread Group v podmeni Config Element in dodamo komponento User Defined Variables. Vanjo vpišemo imena spremenljivk iz seznama spremenljivk, ki smo ga definirali pred testnim scenarijem. Vrednosti vnesemo, če želimo definirati konstante ali dodatne spremenljivke.

V orodju JMeter lahko definiramo tri načine generiranja dinamičnih spremenljivk.

- Z regularnimi izrazi (Regular Expression Extractor) beremo podatke iz odgovorov z regularnimi izrazi.
- Z naključnim spremenljivkam (Random Variable) določamo naključne spremenljivke iz obsega, ki je točno določen in razdeljen na dele.
- Z prednastavitvami v datoteki (CSV Data Set Config) imamo spremenljivke, ki jih definiramo pred izvajanjem testnega scenarija, shranjene v tekstovni datoteki.

Testiranje aplikacije ePoslovanje vsebuje samo prva dva tipa.

JMeter uporablja regularne izraze, ki so zelo podobni kot v programskemu jeziku Perl. Za definiranje tovrstnih komponent se postavimo na Thread Group podmeni Post Processors in dodamo komponento Regular Expression Extractor. Definicija spremenljivke __VIEWSTATE v orodju JMeter je prikazana na sliki 15.



Slika 15: Nastavitev regularnega izraza.

Opis spremenljivk:

Ime (`Name`) ime regularnega izraza.

Glavni vzorec in podvzorec (`Main sample and sub-samples`) spremenljivke za branje odgovorov.

Telo (`Body`) del kode, v katerem se nahaja spremenljivka.

Referenčno ime (`Reference Name`) ime spremenljivke v `User Defined Variables`.

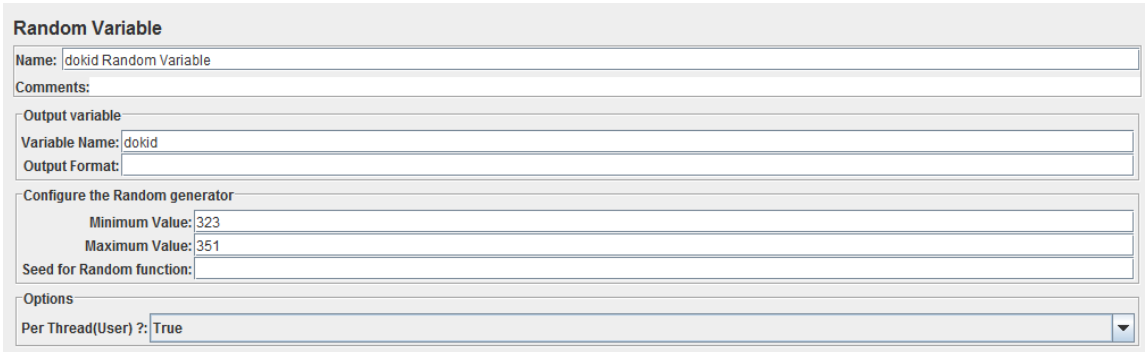
Regularni izraz (`Regular Expression`) regularen izraz za branje vrednosti.

Predloga (`Template`) katero spremenljivko vzamemo, če nam vrne regularni izraz več spremenljivk.

Številka vrnjenega zapisa (0 za naključno) (`Match No. (0 for Random)`) prvi zapis.

Privzeta vrednost (`Default Value`) vrednost spremenljivke, če regularni izraz nič ne vrne.

Naključne spremenljivke v orodju JMeter definiramo tako, da se postavimo na `Thread Group` v podmeni `Config Element` in dodamo komponento `Random Variable`. Definicija spremenljivke `dokid` je prikazana na sliki 16.



Slika 16: Definiranje nakjučne številke.

Opis spremenljivk:

Ime (`Name`) ime funkcije, ki generira naključno spremenljivko.

Ime spremenljivke (`Variable Name`) ime spremenljivke, ki bo dobila naključno vrednost (`dokid`).

Izhodna oblika (`Output Format`) tip spremenljivke izhodnega formata.

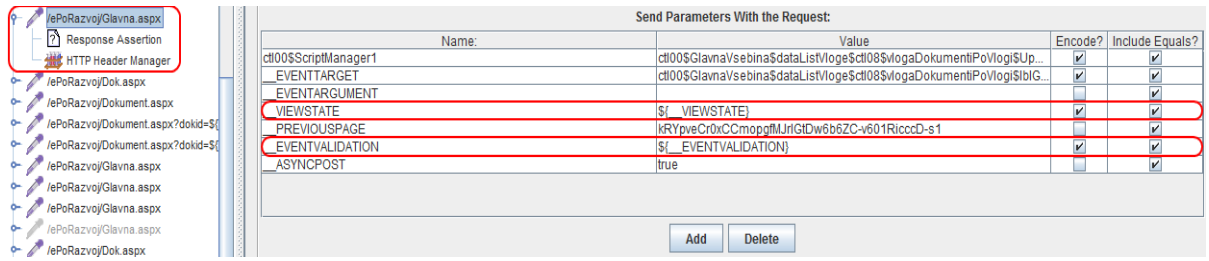
Najmanjša vrednost (`Minimum Value`) najmanjša vrednost spremenljivke.

Največja vrednost (`Maximum Value`) največja vrednost spremenljivke.

Hitrost naključne funkcije (`Seed for Random function`) trajanje generiranja spremenljivk v milisekundah.

Novi uporabniki (`Per Thread(User)?`) generiranje novih spremenljivk za vsakega uporabnika.

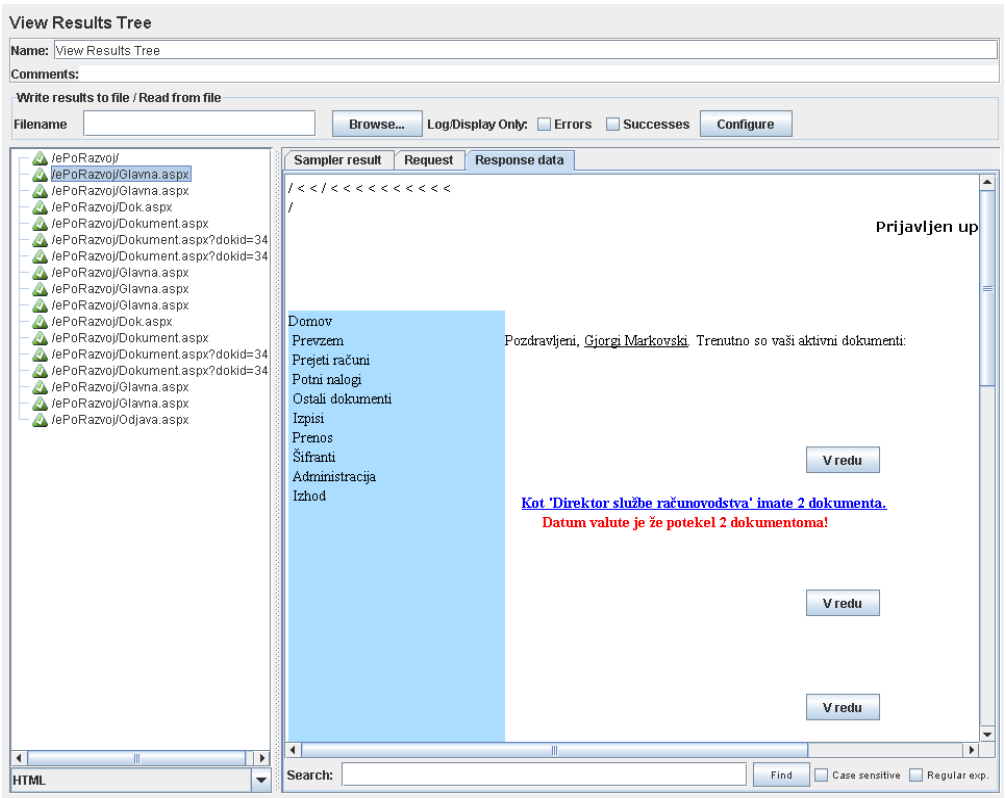
Vse definirane spremenljivke je treba implementirati v testnem scenariju. Implementacijo delamo tako, da za vsako zahtevo poiščemo imena spremenljivk, ki jih bomo pošiljali strežniku. V mojem primeru je to spremenljivka `__VIEWSTATE`. V stolpcu posneto vrednost zamenjamo z vrednostjo spremenljivke. Vrednosti spremenljivk se v orodju JMeter prenašajo v obliki `${ime_spremenljivke}`. Konkretno v mojem primeru se vrednost spremenljivke `__VIEWSTATE` prenaša v obliki `${__VIEWSTATE}`, ker ima spremenljivka isto ime v programski kodi aplikacije kot v orodju JMeter. Implementacijo spremenljivk naredimo za vse dinamične ter (lahko tudi) za statične spremenljivke kot dodatek pri odstranjevanju napak ali ob sestavljanju testnega scenarija. Na sliki 17 je prikazano, kako dodeljujemo vrednosti spremenljivkamji `__VIEWSTATE` in `__EVENTVALIDATION`.



Slika 17: Dodeljevanje vrednosti spremenljivk.

V gornjem levem kotu na sliki 17 je prikazana struktura HTTP zahteve. Vsaka posneta zahteva vsebuje podkomponenti `Response Assertion` in `HTTP Header Manager`. `Response Assertion` uporabljamo za preverjanje odgovora zahteve. V njej dodamo izjeme za vsako zahtevo posebej ali za celoten testni scenarij. Uporabljamo ga za odstranjevanje napak pri izvajanju testnih scenarijev. Ta komponenta je občutljiva na velike in male črke. Večina izjem, ki se pojavijo kot odgovor, vsebuje besedo **Izjema** (*ang. Exception*) ali **izjema** (*ang. exception*). Primeri so **Naslednja izjema je prišla** (*ang. The following exception occurred*) ali **Strežniška izjema med** (*ang. Server Exception during*). Poleg tega `Response Assertion` komponenta podpira regularne izraze.
Po nastavitvah komponent `User Defined Variables`, `HTTP Cookie Manager`,

`HTTP Authorization Manager`, `HTTP Request Defaults` in vseh ostalih bo naslednja komponenta iz menija `Listener`. Te komponente nam določajo vizuelno predstavitev. JMeter jih shrani v tekstovno datoteko in v odvisnosti od izbrane komponente poslušalec različno interpretira podatke. To je prva komponenta, na podlagi katere bomo sklepali, če smo testni scenarij pravilno načrtovali. Prva komponenta, ki jo bomo uporabili samo za pregledovanje pravilnosti testnega scenarija bo `View Results Tree`. Za prikaz pravilnosti in odstranitev napak v testnem scenariju moramo tega zagnati z navideznim uporabnikom. V `Thread Group` bomo generirali enega uporabnika za eno sekundo v enem ciklu brez ponavljanja. Poženemo testni scenarij (pritisnemo na `Ctrl+R`) in se postavimo na komponento `View Results Tree`. Na komponenti se bodo prikazali zelene ali rdeče trikotniki odvisno od prvilnosti zahtev (kot je prikazano na sliki 18). Poleg pravilnosti zahtev znotraj komponente vidimo še naslov zahteve, podatke o zahtevi, odgovore in glave zahtev. V primeru, da smo definirali napačen regularni izraz, se postavimo na `Tree RegEx Tester`, kjer lahko preverimo pravilnost regularnega izraza. Če se postavimo na `XML` bomo videli drevesno strukturo spletne strani, na `Text` pa bomo videli vsebino spletne strani. Ker orodje JMeter ne zna interpretirati JS in CSS bo v komponenti `View Results Tree` implementirana samo čista HTML programska koda. Povezave in vsebina spletne strani bodo sicer vidne, vendar njun izgled ne bo enak kot v brskalniku. To lahko vidimo iz prikazane spletni strani na sliki 18.



Slika 18: HTML interpretacija aplikacije ePoslovanje znotraj orodja JMeter.

Testni scenarij, s slike 18 je brez napak. To ugotovimo iz zelenih trikotnikov, vendar to žal še ne pomeni, da se je tok dogodkov pravilno izvedel. Napake se lahko zgodijo zaradi vnosa napačnih podatkov v nekaterih poljih. Zato moramo preveriti še izgled celotne

prikazane spletne strani za vsako zahtevo posebej. Moramo se prepričati, da se je testni scenarij izvedel, kot smo ga definirali na začetku. Zato onemogočimo komponento `View Results Tree`, da se ne upošteva pri izvajanju testnega scenarija.

Naslednja dodana komponenta je čas zakasnitve uporabnikov. To približa navidezne uporabnike realnim uporabnikom. Za dodajanje zakasnitev v testnem scenariju se postavimo na `TestniScenarij` in dodamo komponento `Timer`. V svojem testnem scenariju sem uporabil Gaussovo časovno porazdelitev (v komponenta `Gaussian Random Timer`), ki vsebuje dve polji. Standardna deviacija (`Deviation`) generira čas zakasnitve v milisekundah in naključen čas v milisekundah, ki se bo spreminjal po Gaussovi porazdelitvi. Zakasnitveni čas v milisekundah (`Constant Delay Offset (in milliseconds)`) pa določa najmnajši čas, ki se še prišteje naključnemu času. `Gaussian Random Timer` postavimo pred vsako zahtevo, za katero predpostavljamo, da bo uporabnik rabil nekaj časa. Komponente poimenujemo po aktivnostih, ki jih v tem času uporabnik izvaja, (prijava, vnos podatkov, pregled podatkov, dodelitev podatkov, odjava). Ponavadi za vnos podatkov v tekstovna polja povprečen uporabnik porabi od 4 do 7 sekund. Čas za vnos podatkov nastavimo glede na količino polj, ki so na strani, čas zakasnitve pa je ponavadi od 8 do 15 sekund.

Za boljši prikaz dobljenih podatkov pri testiranju aplikacije moramo uporabiti najmanj eno komponento iz menija `Listener`. V svojem testnem scenariju sem uporabil komponente `View Results in Table`, `Aggregate Graph` in `Statistical Aggregate Reports`. Vsaka komponenta različno interpretira dobljene podatke.

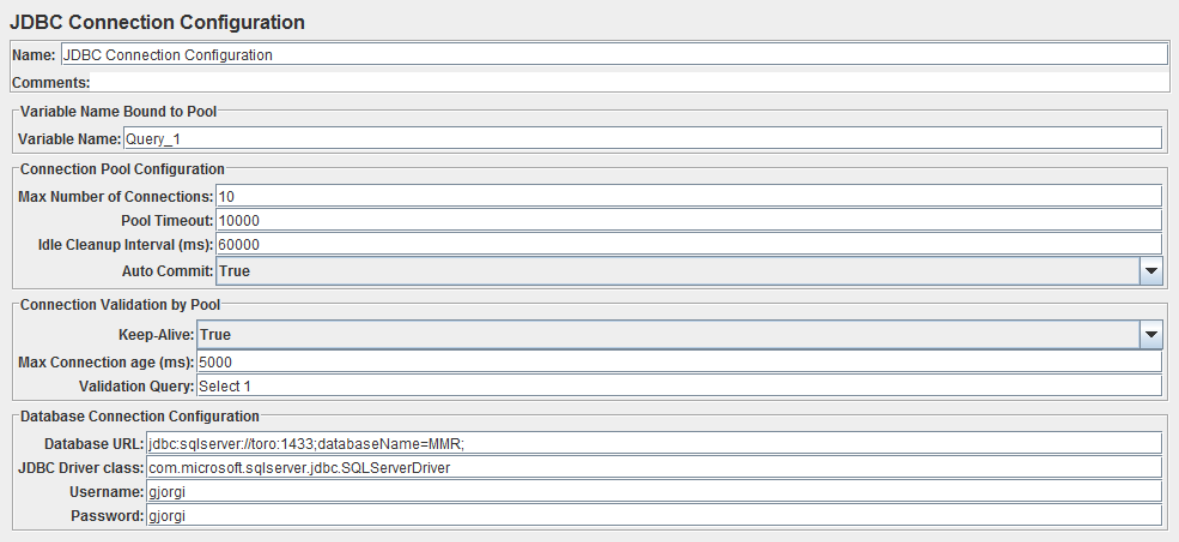
Testni scenarij bo postal popoln z komponento za izpis grafa. Najprej določimo prikaz podatkov, nato število uporabnikov in njihovo porazdelitev. Pritisnemo tipko (`Ctrl+R`) in v zgornjem desnem kotu se bo kvadrat pobarval zeleno, levo od njega pa se bo prikazalo število vseh uporabnikov, ki jih želimo generirati, ter število generiranih uporabnikov do tega trenutka 

.

3.2.4 Testiranje podatkovnega strežnika

Sestavljanje testnega scenarija za testiranje dostopa do baze podatkov je enostavno. Ni nam treba snemati zahtev in uporabljati posredniškega strežnika. Sestavljanje testnega scenarija tega tipa zato vsebuje veliko manj komponent kot pri spletni aplikaciji. Po drugi strani pa JMeter ne vsebuje gonilnikov za povezavo do izbrane podatkovne baze. Gonilniki so odvisni od verzije podatkovne baze in verzije Jave, na katero kaže JMeter. JMeter je javanska aplikacija in uporablja JDBC gonilnike za povezavo s podatkovnimi bazami. Obstajajo različni gonilniki za različne podatkovne baze. V diplomski nalogi sem opisal testni scenarij za podatkovno bazo MS SQL in verzijo Jave 1.6. Gonilnike namestimo tako, da jih dodamo v `lib` mapo. Treba se je samo prijaviti z uporabniškem imenom in geslom z omejenimi pravicami, da v primeru napak zaščitimo podatke pred izgubo ali brisanjem.

Pri sestavljanju testnega scenarija za testiranje strežnika podatkovne baze v `Test Plan` dodamo prvo podkomponento `Thread Group` in definiramo število uporabnikov. Potem iz `Config Element` dodamo prvo podkomponento `JDBC Connection Configuration` in nastavimo potrebne podatke kot je prikazano na sliki 19.



Slika 19: Nastavitev povezave s strežnika baze podatkov.

Ime (`Name`) Ime povezave.

Ime spremenljivke (`Variable Name`) ime spremenljivke, na katero je vezana povezava.

Največje število povezav (`Max Number of Connection`) postavimo na 0, če želimo, da se povezava ne deli med uporabniki, sicer vnesemo število.

Čakalna doba (`Pool Timeout`) čas za vzpostavitev povezave; v nasprotnem primeru se sproži napaka.

Branje v primeru mirovanja (`Idle Cleanup Interval (ms)`) čas, po katerem pobriše podatke, če povezava ni aktivna.

Avtomatsko shranjevanje (`Auto Commit`).

Ohranitev povezave (`Keep-Alive`).

Najdaljša povezava v milisekundah (`Max Connection age (ms)`).

Preverjanju poizvedb (`Validation Query`) preverjanje, če povezava dobiva odgovore.

Naslov podatkovne baze (`Database URL`) naslov strežnika podatkovne baze.

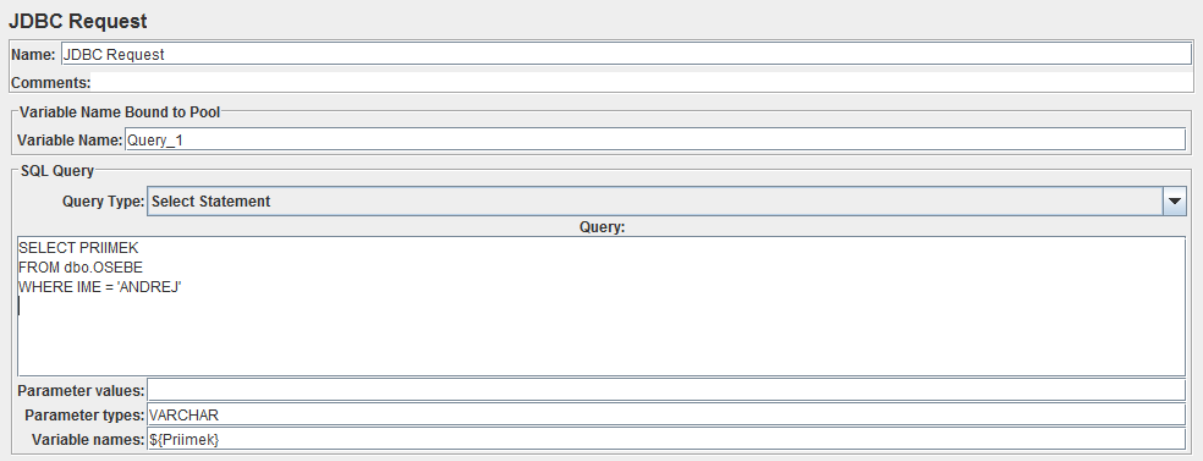
JDBC gonilniki (`JDBC Driver class`) JDBC za povezujoči niz.

Uporabniško ime (`Username`) Uporabniško ime za podatkovno bazo.

Geslo (`Password`) Geslo za podatkovno bazo.

Pod vsako povezavo dodamo podkomponento zahteve. Zahtevo za povezavo do podatkovnega strežnika dodamo v podmeni `Sampler` komponente `JDBC Request`. V komponenti `JDBC Request` lahko definiramo različne SQL stavke. Vsak stavek vrača vrednost, ki jo določimo spremenljivki. Naslednjo, JDBC zahtevo lahko uporabljamo tudi v testnih scenarijih za testiranje spletne aplikacije, ki v ozadju uporablja podatkovno bazo.

Nastavitev za komponento `JDBC Request` je prikazana na sliki 20.



Slika 20: Nastavitev zahteve za strežnik baze podatkov.

Ime (`Name`) Ime, pod katerim se zahteva prikazuje v drevesni strukturi.

Ime spremenljivke (`Variable Name`) ime spremenljivke, na katero je vezana povezava. Mora se ujemati z imenom spremenljivke v komponenti `JDBC Connection Configuration` (slika 20).

Tip poizvedbe (`Query Type`).

Poizvedba (`Query`).

Parameterske vrednosti (`Parameter values`) seznam vrednosti parametrov, ki so ločeni z vejico.

Tip parametrov (`Parameter types`) tipi parametrov za vsako spremenljivko posebej, ki so ločeni z vejico.

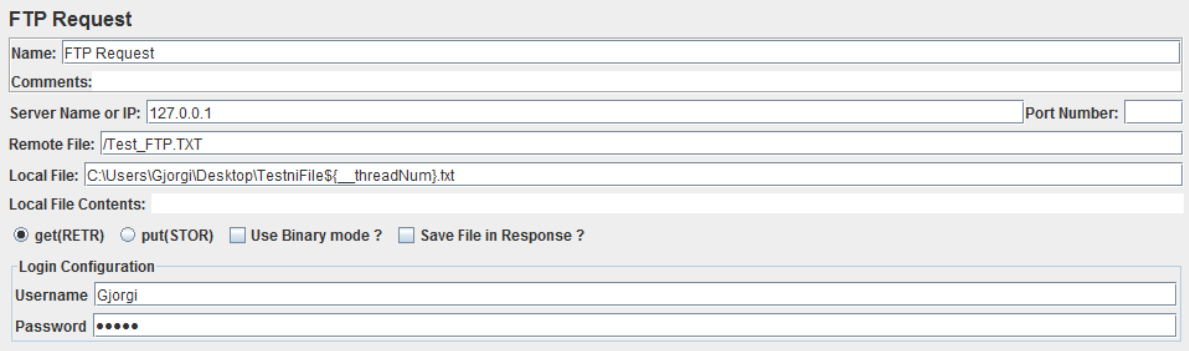
Ime spremenljivk (`Variable names`) imena spremenljivk, ki jim dodelimo vrednosti.

Za konec določimo še komponento za izpis dobljenih podatkov in poženemo testni scenarij.

3.2.5 Testiranje FTP strežnika

Za sestavljanje testnega scenarija za testiranje FTP strežnika sem uporabil brezplačni FTP strežnik FileZilla. Ker v svoji diplomski nalogi nisem imel na razpolago ustrezne strojne opreme za izvedbo testiranja, sem samo opisal postopek sestavljanja. Za sestavljanje testnega scenarija sem pripravil tekstovno datoteko `Test_FTP.txt` in jo shranil v mapo, na katero kaže podatkovni strežnik. Ko so enkrat podatki pripravljeni, poteka sestavljanje testnega scenarija na enak način kot prejšnja dva. Podkomponente so odvisne od vsebine testnega scenarija. Za začetek bomo dodali komponento `FTP Request Defaults`. Komponenta vsebuje vsa skupna polja, ki jih imajo FTP zahteve. Podobna je `HTTP Request Defaults` pri spletnih testnih scenarijih. Sledi dodajanje komponente `FTP`

`Request`, v katero vnesemo zahtevo, kot je prikazano na sliki 21.



Slika 21: `FTP Request` komponenta.

Ime strežnika ali IP naslov (`Server Name or IP`) ime FTP strežnika ali IP naslov s številko vrat (če ta ni določena, je 21).

Oddaljena datoteka (`Remote File`) ime datoteke, ki jo želimo prenesti na odjemalčev računalnik.

Lokalna datoteka (`Local File`) ime datoteke, pod katerim jo bomo shranili oddaljeno datoteko na računalniku.

Vsebina lokalne datoteke (`Local File Contents`) je vsebina, ki se uporablja za prepis na že obstoječi `Local File` pri nalaganju datotek na FTP strežnik.

Pridobivanje datoteke (`get(RETR)`)

Uporabniško ime (`Username`) uporabniško ime za dostop do podatkov.

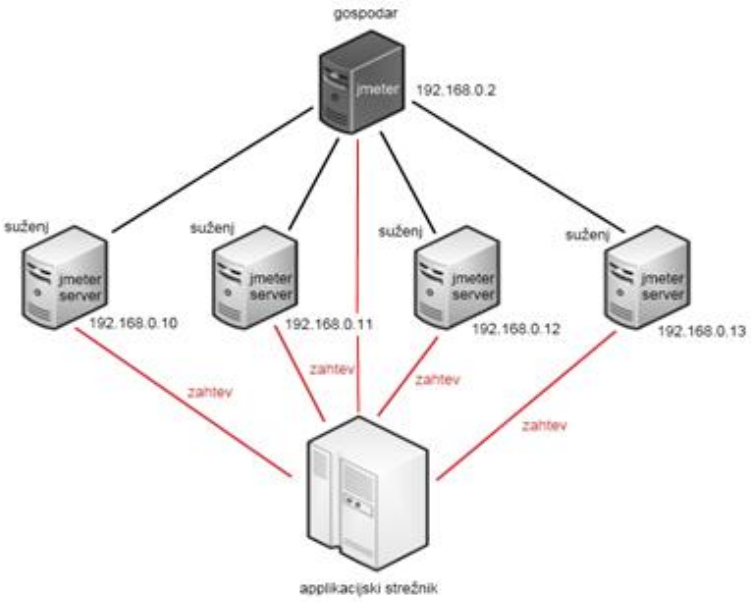
Geslo (`Password`) geslo za dostop do podatkov.

Testni scenarij iz zgornjega primera, izvaja naslednji kratek tok dogodkov. Najprej prijava in nato prenos tekstovne datoteke `Test_FTP.txt` na odjemalčev računalnik pod imenom `TestniFileN.txt`.
Če pogledamo vsebino polja `Local File`, bomo videli, da vsebuje vrednost spremenljivke `${__threadNum}`, ki predstavlja števec. Imena datotek, ki se bodo shranila na odjemalčevem računalniku, bodo od `TestniFile1.txt`, do `TestniFileN`, pri čemer N pomeni število uporabnikov. Za konec dodamo še nekaj komponent za prikaz dobljenih podatkov in poženemo testni scenarij.

4. Izvedba testiranja in analiza rezultatov

4.1 Testno okolje

Po preveritvi vseh nastavitev moramo postaviti okolje, kot na sliki 23. Črne povezave pomenijo pošiljanje ukazov iz gospodarjevega računalnika na podrejene, rdeče povezave pa pomenijo pošiljanje ukazov na testni strežnik, na katerem izvajamo testiranje delovanja.



Slika 23: Testna struktura.

Na gospodarjevem računalniku v `jmeter.properties` dodamo IP naslove vseh podrejenih računalnikov. S tem bodo v meniju vidni vsi dodani IP naslovi. Na vseh podrejenih računalnikih poženemo `jmeter-server.bat`, ki uporablja oddaljen dostop (RMI). JMeter ponuja tudi izbiro IP naslovov, na katerih želimo poganjati testnih scenarij. V primeru, da imamo preveč računalnikov (ali generiranih navideznih uporabnikov), jih lahko del odjavimo iz testnega plana na gospodarjevem računalniku. Pri izvajanju testiranja delovanja je običajno premalo navideznih uporabnikov. Pri distribucijskih testiranjih obstajajo naslednje omejitve:

- Oddaljen dostop (RMI) ne deluje preko omrežja brez posredniških strežnikov, zato tudi JMeter ne more komunicirati na ta način.
- JMeter pošilja vse rezultate testiranja na kontrolno konzolo in lahko zasiči omrežje. Smiselno je uporabiti preprost program za shranjevanje rezultatov in kasnejši ogled datoteke z enim od poslušalcev, ki jih ima JMeter.
- Če strežnik ni velik večprocesorski sistem, zmore kvečjemu enega ali dva odjemalca.
- Testiranje XML aplikacij je od 4 do 10-krat počasnejše od aplikacij.

Poleg naštetih omejitev je največje število generiranih uporabnikov neposredno odvisno od števila računalnikov, ki jih imamo na razpolago (njihove zmogljivosti in programske opreme oziroma operacijskih sistemov). Vsak navidezen uporabnik je programsko realiziran z eno nitjo. Vsaka nit (odvisno od zahtev navideznega uporabnika), porabi od 1MB do 2MB pomnilnika. Iz tega sklepamo, da je število navideznih uporabnikov neposredno odvisno od velikosti pomnilnika.

Vsak operacijski sistem ima omejeno število niti, ki se lahko istočasno izvajajo. Windows XP podpira 3000 istočasno delujočih niti. Linux jih podpira nekaj več, 3500. Operacijski sistem torej omejuje število navideznih uporabnikov ne glede na pomnilnik. Tudi če

imamo večji pomnilnik, število istočasnih niti ne more preseči 3000 za Windows XP oziroma 3500 za Linux na enem računalniku. Poleg tega je potrebno upoštevati, da tudi sam operacijski sistem izkorišča pomnilnik in zasede določeno število niti.
Vsak proces je nadalje omejen na uporabo pomnilnika in lahko zasede do 2GB pomnilnika oziroma 2048 MB, kar bi pomenilo od 1000 do 2000 istočasnih navideznih uporabnikov. Da bi JMeter lahko uporabil toliko pomnilnika, mu moramo to posebej dovoliti. To storimo tako, da povečamo dodeljen pomnilnik, ki ga lahko JVM uporablja. V `jmeter.bat` dodelimo več pomnilnika tako, da povečamo `JVM heap size` v vrstici (`set HEAP = -Xms 256m -Xmx256m`), ki pomeni:

- `Xmx` največja velikost pomnilnika (kopice).
- `Xms` začetna velikost pomnilnika (kopice).

Število istočasnih uporabnikov, ki jih lahko generira en računalnik, je torej odvisno od največjega pomnilnika za en proces. Pri določanju največjega števila istočasnih navideznih uporabnikov moramo spremljati tudi obremenjenost centralno procesne enote. Obremenjenost procesorja ne sme biti večja kot 90%, sicer dobljeni rezultati ne bodo točni. Čim manjša je poraba CPE, toliko bolj bo test natančen.

V diplomski nalog sem opisal in analiziral dva testna scenarija. Prvi testni scenarij je za spletno aplikacijo ePoslovanje, drugi test pa za izvajanje SQL poizvedb in stavkov na podatkovnem strežniku. Prvi testni scenarij sem pognal preko grafičnega uporabniškega vmesnika. Drugi testni scenarij pa sem pognal na način gospodar-suženj znotraj lokalnega omrežja na dveh računalnikih. Testno okolje za testiranji je bilo sestavljeno iz:

Treh računalnik: gospodarja, sužnja in strežnika.

Karakteristike gospodarjevega računalnika:

- OS Microsoft Windows 7 Enterprise (32 bitni)
- CPU Intel Core 2 Quad (2.67 GHz)
- RAM 4GB

Karakteristike sužnjevega računalnika:

- OS Microsoft Windows XP Professional + Service Pack 3
- CPU Intel Pentium 4 (2.4 GHz)
- RAM 1.49 GB

Karakteristike strežnika:

- Microsoft Windows Server 2003 + Service Pack 2
- Microsoft SQL Server 2005
- CPU Intel Pentium 4 (2.8 GHz)
- RAM 1 GB
- HD 80GB ATA100 7200RPM

Zmogljivost obeh testnih računalnikov je večja od zmogljivosti strežnika. Zato pri analizi nisem obravnaval obremenjenosti ostalih računalnikov razen strežnika, na katerem je tekla aplikacija. Tudi v drugem testnem scenariju za testiranje podatkovnega strežnika sem uporabil isti strežnik.

4.2 Analiza dobljenih rezultatov

4.2.1 Splošno o analizi podatkov

Natančnost analize dobljenih rezultatov je odvisna od ustreznega prikaza podatkov. Najpreglednejše komponente za prikaz podatkov so `Statistical Aggregate Report`, `View Results in Table` in `Aggregate Graph`. Poleg tega ima vsak standardni graf tudi možnost za shranjevanje dobljenih podatkov v tekstovno datoteko. Tako lahko podatke uporabimo tudi kasneje, ko jih lahko odpremo s katerokoli komponento iz menija poslušalcev. Pri shranjevanju lahko izbiramo podatke, ki jih želimo shraniti v tekstovno datoteko.

JMeter vsebuje podatke samo o zadnjem testiranju. V kolikor jih ne shranimo, se ne shranijo avtomatsko. Če bi želeli testirati in po optimizaciji primerjati dobljene podatke, JMeter te funkcionalnosti ne podpira oziroma ne vsebuje komponente, s katero bi primerjali podatke. To slabost za primerjanje dobljenih rezultatov pred in po optimizaciji lahko dopolnimo z uporabo orodja Apache Ant [25]. Apache Ant ohranja dobljene podatke v podatkovni bazi in jih prikaže v HTML obliki.

Preden se lotimo analize dobljenih podatkov moramo definirati, kaj so dobri ter kaj so slabi rezultati. Vsaka aplikacija dela različno in izvaja različne operacije. Dobri ali slabi rezultati so tako relativna stvar, ker kriterij ni natančno določen. Da bi določil, koliko časa lahko en uporabnik čaka na odgovor, sem preučil z različne raziskave [1].

Naslednji seznam upošteva rezultate raziskav, ki so bile narejene v poznih 1980 letih. Raziskave so bile narejene za razvrstitev učinkovitosti glede na čas odgovora. Ugotovitve so veljavne še danes.

Uporabnok čaka na odgovor:

Več kot 15 sekund

Če na odgovor uporabnik čaka več kot 15 sekund, bo prekinil povezavo z aplikacijo. V določenih primerih z različnimi uporabniki je čakanje več kot 15 sekund na odgovor za eno zahtevo sprejemljivo - recimo pri zasedenem operaterju v klicnem centru je čakanje več kot 15 sekund smiselno. V primeru, da pride do zakasnitev, je sistem lahko narejen tako, da uporabnika preusmeri na drugo aktivnost do trenutka, ko obdela njegovo zahtevo in prikaže odgovor.

Več kot 4 sekunde

Čakanje na odgovor več kot 4 sekunde je preveč za aplikacije, preko katerih prenašamo slike ali video. Zakasnitev ovira aktivnosti, ampak je do te meje še tolerantno.

Od 2 do 4 sekunde

Čas čakanja, ki je večji kot 2 sekundi, ovira operacije, ki zahtevajo koncentracijo na visoki ravni. Čakanje od 2 do 4 sekunde pred računalnikom je dolgo za uporabnike, ki čakajo hitre odgovore. Takšen čas čakanja je spejemljiv za tipkanje naslovov, naslovov elektronske pošte, ne pa za primerjanje različnih artiklov.

Manj kot 2 sekundi

Ko si mora uporabnik zapomniti informacijo na nekaj odgovorov, mora biti čas kratek. Da si uporabnik zapomni več informacij pa potrebuje več časa. Za zahtevne aktivnosti kot je primerjanje različnih informacij, ki se razlikujejo po več vrednostih, je meja 2 sekunde. (Primer, je nakup izdelkov v spletni trgovini).

Manj kot 1 sekundo

Nekatere vrste intenzivnega dela (kot je pisanje knjige) ali delo z aplikacijami, ki imajo bogato grafiko, zahtevajo zelo kratek čas odgovora za vzdrževanje uporabnikovega interesa in pozornosti dlje časa. Umetnik, ki prenaša sliko od ene lokacije na drugo, mora biti pripravljen na naslednji korak.

Manj kot desetinka sekunde

Zaznavanje črke na zaslonu ali klik na miškin gumb se izvede v trenutku, v manj kot 0,1 sekunde po sproženi akciji. Veliko računalniških iger zahteva izredno hitrost interakcije.

Kot smo videli iz raziskave je kritičen čas za odgovor okrog 2 sekundi. Če je čas odgovora večji, ima določen vpliv na produktivnost povprečnih uporabnikov, tako da je nominalna osvežitev strani 8 sekund, ki smo jo mi izbrali za spletne aplikacije daleč od idealne.

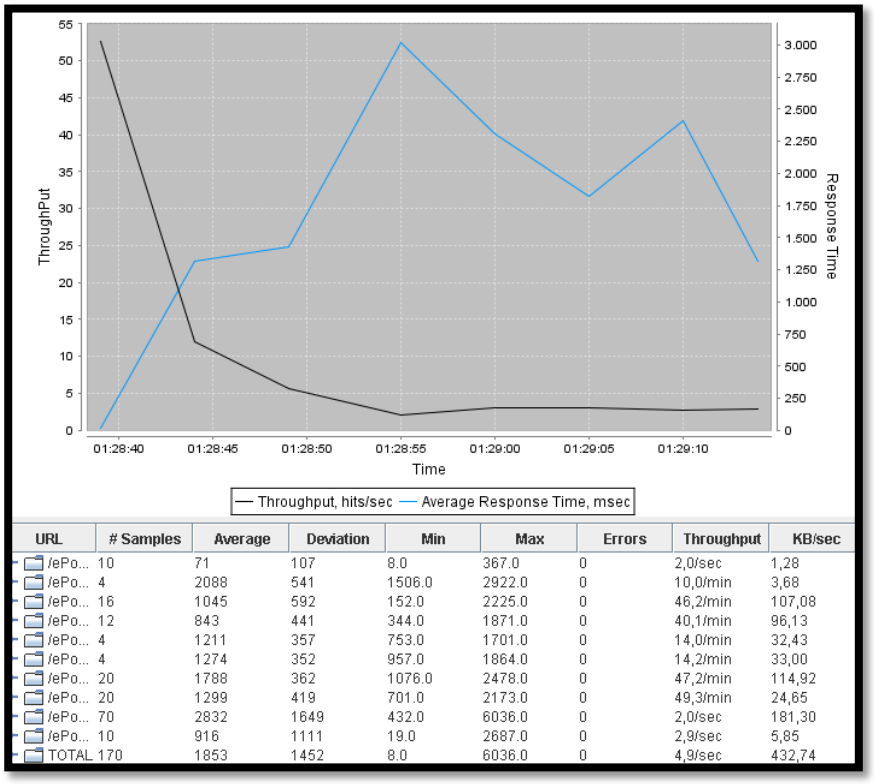
Iz dobljenih podatkov lahko sklepamo, da uporabniki lahko čakajo do največ 8 sekund brez prekinitve komunikacije z aplikacijo. Vsi odgovori do tega časa so sprejemljivi, vse kar je več kot 8 sekund, pa ni sprejemljivo.

4.2.2 Analiza podatkov iz testnih scenarijev

4.2.2.1 Analiza testiranja spletne aplikacije ePoslovanje

Kot rečeno pri izvajanju testiranja število uporabnikov povečujemo po korakih. Tako sem najprej na koncu testni scenarij pognal z 10 navideznimi uporabniki, potem z 20, 50, 100, 500 in 1000. Za generiranje navideznih uporabnikov sem uporabil komponento `Thread Group`. Časovno razmerje za generiranje manjših števil navideznih uporabnikov sem nastavil na 1 sekundo. Testne scenarije z manjšim številom uporabnikov sem pognal brez časovne zakasnitve, da bi hitreje prišel do večjega števila uporabnikov. Povečeval sem število navideznih uporabnikov, dokler nisem dosegel 100% obremenitve strežnika. Za prikaz dobljenih podatkov sem uporabil komponento `Statistical Aggregate Report`.

Testiranje spletne aplikacije sem začel z 10 navideznimi uporabniki. Dobljeni podatki za ta primer so prikazani na sliki 24.



URL	# Samples	Average	Deviation	Min	Max	Errors	Throughput	KB/sec
/ePo...	10	71	107	8.0	367.0	0	2,0/sec	1,28
/ePo...	4	2088	541	1506.0	2922.0	0	10,0/min	3,68
/ePo...	16	1045	592	152.0	2225.0	0	46,2/min	107,08
/ePo...	12	843	441	344.0	1871.0	0	40,1/min	96,13
/ePo...	4	1211	357	753.0	1701.0	0	14,0/min	32,43
/ePo...	4	1274	352	957.0	1864.0	0	14,2/min	33,00
/ePo...	20	1788	362	1076.0	2478.0	0	47,2/min	114,92
/ePo...	20	1299	419	701.0	2173.0	0	49,3/min	24,65
/ePo...	70	2832	1649	432.0	6036.0	0	2,0/sec	181,30
/ePo...	10	916	1111	19.0	2687.0	0	2,9/sec	5,85
TOTAL	170	1853	1452	8.0	6036.0	0	4,9/sec	432,74

Slika 24: Propusnost in čas odgovor pri 10 navideznih uporabnikih.

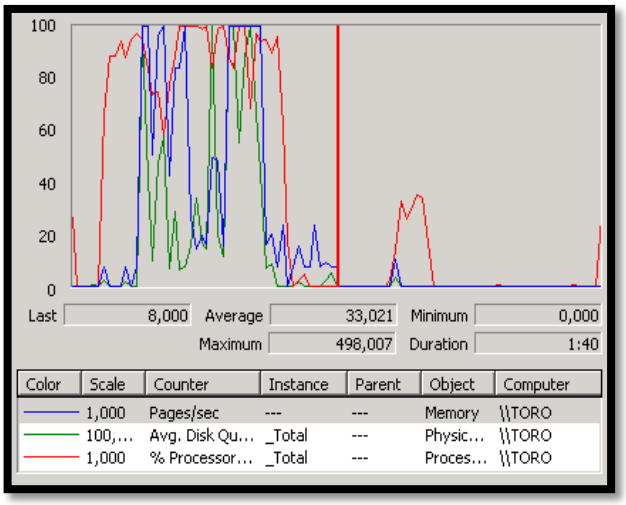
Z črno bravo je prikazana propustnost podatkov, ki določa stopnjo prepustnosti podatkov. Pogosto nenadno zmanjšanje prepustnosti podatkov (kot je to na sliki 24) predstavlja prvi simptom težav s kapaciteto, kar z drugimi besedami pomeni, da strežnik ne more več slediti številu zahtev. To pomeni, da bodo uporabniki začeli trpeti zaradi strežniških prekinitev. Ko je propustnost najmanjša, strežnik ne more pravočasno obdelati vseh poslanih zahtev. To se najpogosteje zgodi pri prijavi in odjavi iz aplikacije, ko vsi uporabniki uporabljajo isto komponento.

Modra linija nam kaže povprečen čas za odgovore oziroma povratne informacije. Povprečen čas ne bi smel presegati 8 sekund, da bi uporabniki še vedno ostali v interakciji z aplikacijo. Kot vidimo je na sliki 24 največji čas odgovora za 10 navideznih uporabnikov približno 3 sekunde. Pri tem velja da se generirajo vsi uporabniki v razmikih od eno sekundo.

Tak primer je recimo takrat, ko imamo aktualno novico (rezultate izpitov in podobno). Takrat vsi uporabniki naenkrat dostopajo do spletne aplikacije v kratkem časovnem intervalu. Najbolj obremenjena stran aplikacije ePoslovanje je prijava. Iz tega lahko sklepamo, da so zahteve uporabnikov iste, oziroma, da vsi hočejo dostopati do iste funkcionalnosti aplikacije (prijava). Povprečen čas za odgovor pa je odvisen tudi od tega, koliko časa rabijo za prijavljanje v sistem.

Povprečen čas odgovora narašča linearno s številom navideznih uporabnikov. Ostali podatki so natančneje opisani v tabeli pod grafom, v kateri so prikazani naslednji podatki: v povprečen čas odgovora, odklon, najmanjši čas, največji čas, število napak, propustnost podatkov in hitrost prenesenih podatkov. Odklon nam pove, koliko je aplikacija stabilna.

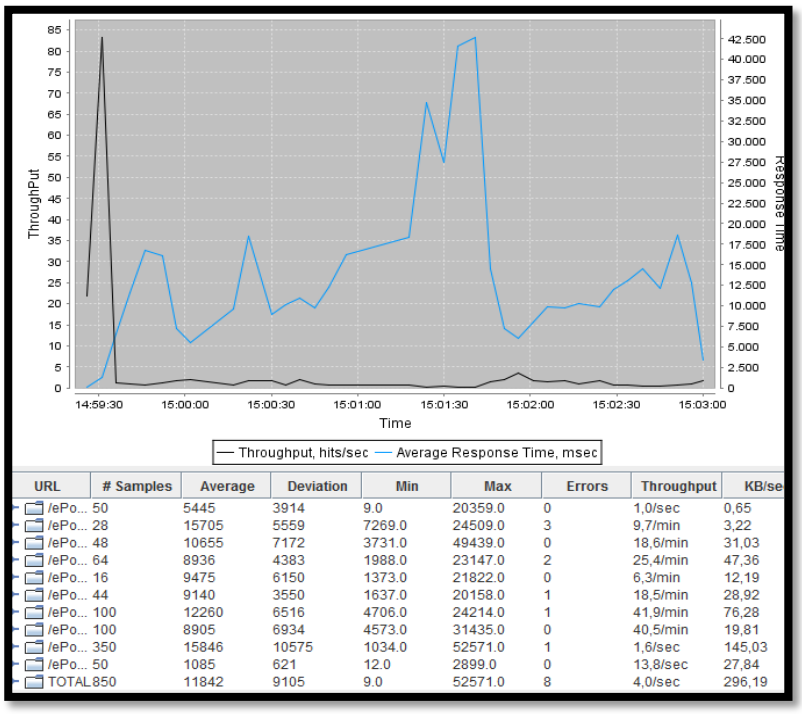
Če je odklon manjši, je aplikacija bolj stabilna.
Obremenitev strežnika sem nadzoroval z uporabo orodja Performance Monitor, ki je sestavni del operacijskega sistema Windows. Performance Monitor je prikazan na sliki 25.



Slika 25: Grafičen prikaz obremenjenosti strežnika pri 10 navideznih uporabnikih.

V spodnjem delu so opisane barve črt, ki so prikazane na grafu. Modra črta pomeni zasedenost podatkovni poti pri prenosu podatkov med glavnim pomnilnikom in trdim diskom (stran/sekunda). Zelena črta pomeni povprečno dolžino vrste za prenos in obdelavo podatkov iz trdega diska. Rdeča črta pomeni zasedenost procesorja v odstotkih.

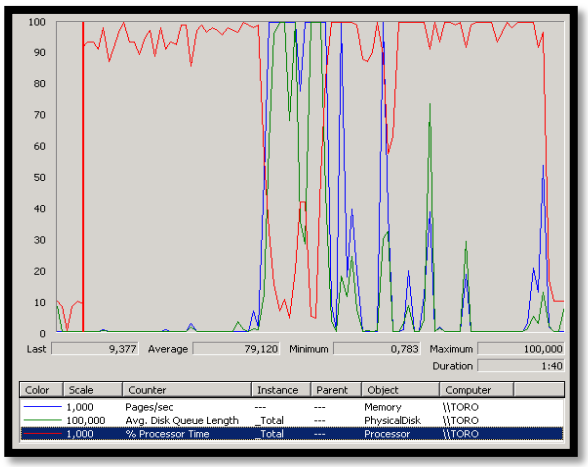
Na sliki 25 vidimo, da se zasedenost procesorja pri 10 istočasnih navideznih uporabnikih giblje od 60% do 100%. Ker je intreval kratek obremenjenost strežnika ne vpliva veliko na čas odgovora. Zasedenost pomnilnika in diska se je povečala do 90%. Iz tega lahko sklepamo, da bo aplikacija delovala nemoteno pri 10 istočasnih uporabnikih in zato smo povečali število generiranih istočasnih uporabnikov na 20, vendar ni bilo velikih razlik. Ko sem generiral 50 navideznih uporabnikov (podatki za 50 istočasnih navideznih uporabnikov brez zakasnitev so prikazani na sliki 26) pa je čas odgovora narastel na kar 42 sekund, kar pomeni, da je 50 istočasnih uporabnikov očitno preveč.



URL	# Samples	Average	Deviation	Min	Max	Errors	Throughput	KB/se
/ePo...	50	5445	3914	9.0	20359.0	0	1,0/sec	0,65
/ePo...	28	15705	5559	7269.0	24509.0	3	9,7/min	3,22
/ePo...	48	10655	7172	3731.0	49439.0	0	18,6/min	31,03
/ePo...	64	8936	4383	1988.0	23147.0	2	25,4/min	47,36
/ePo...	16	9475	6150	1373.0	21822.0	0	6,3/min	12,19
/ePo...	44	9140	3550	1637.0	20158.0	1	18,5/min	28,92
/ePo...	100	12260	6516	4706.0	24214.0	1	41,9/min	76,28
/ePo...	100	8905	6934	4573.0	31435.0	0	40,5/min	19,81
/ePo...	350	15846	10575	1034.0	52571.0	1	1,6/sec	145,03
/ePo...	50	1085	621	12.0	2899.0	0	13,8/sec	27,84
TOTAL	850	11842	9105	9.0	52571.0	8	4,0/sec	296,19

Slika 26: Propusnost in čas odgovor pri 50 navideznih uporabnikih.

Na sliki 27 vidimo, da je tudi obremenjenost procesorja strežnika skoraj ves čas večja kot 95%. Pri določeni zahtevi pride do prenosa podatkov iz trdega diska v CPE, zato tako zasičene pomeni dodatno povečevanje časa za obdelavo podatkov in s tem povečevanje časa za odgovor. Tudi iz slike 27 lahko sklepamo, da je 50 istočasnih navideznih uporabnikov preveč za uporabljeno strojno opremo.



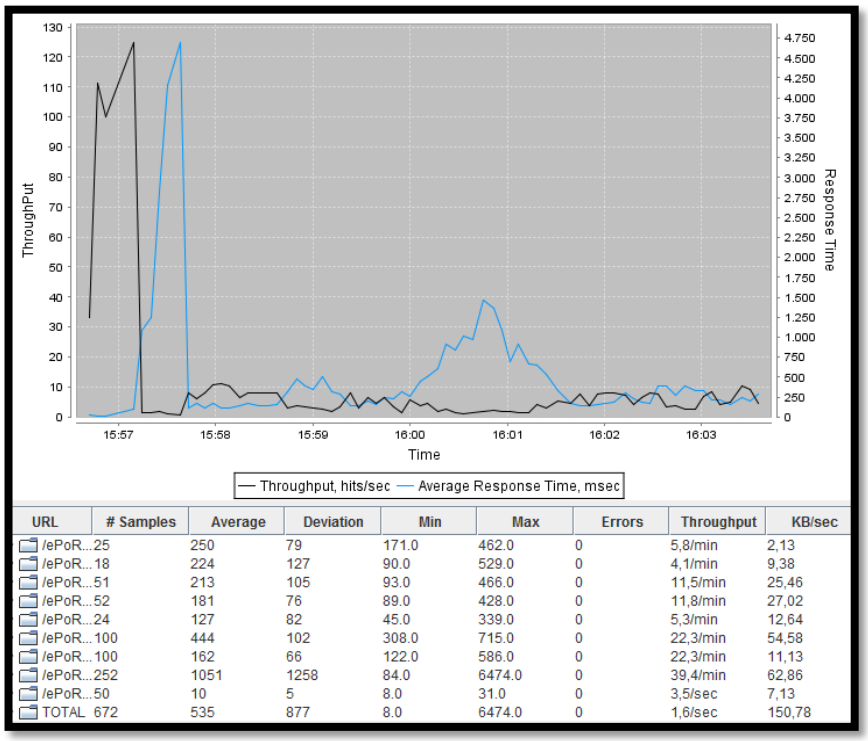
Slika 27: Grafičen prikaz obremenjenosti strežnika pri 50 navideznih uporabnikih.

Ti rezultati so bili za navidezne uporabnike, pri realnih bo mogoče drugače. Število istočasnih realnih uporabnikov določimo tako, da nastavimo čas zakasnitev v testni scenarij pred vsakim korakom scenarija. V mojem testnem scenariju je bil čas zakasnitev

naslednji:

- Prijava: 2 – 3 sekunde
- Izbira na meniju: 3 – 4 sekunde
- Izbira pregledovalca: 3 – 4 sekunde
- Vnos podatkov: 10 – 13 sekundi
- Odjava: 2 – 3 sekunde

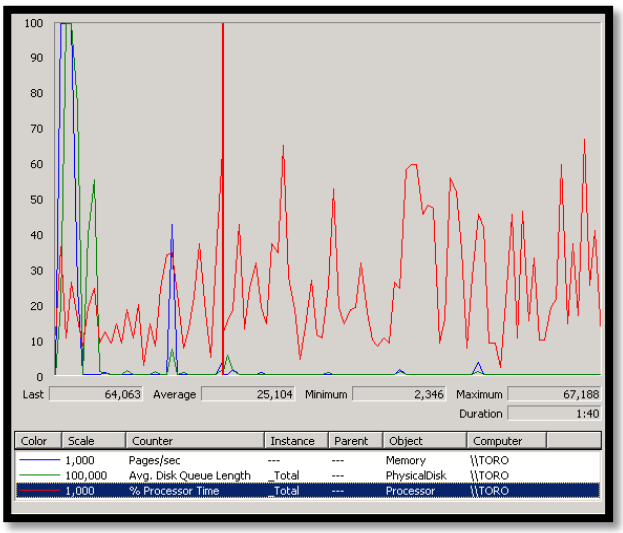
Testniranje sem najprej izvedel z 50 istočasnimi realnimi uporabniki. Dobljeni podatki so prikazani na sliki 28. Kot vidimo je sedaj najdaljši čas odgovora nekaj manj kot 5 sekund, kar je sprejemljivo s stani uporabnika. Tudi to iz grafa ugotovimo, da se najdaljši čas odgovora pojavlja pri prijavi v aplikacijo. V vseh ostalih primeri je čas odgovora manjši od 2 sekund, kar je sprejemljivo.



URL	# Samples	Average	Deviation	Min	Max	Errors	Throughput	KB/sec
/ePoR...	25	250	79	171.0	462.0	0	5,8/min	2,13
/ePoR...	18	224	127	90.0	529.0	0	4,1/min	9,38
/ePoR...	51	213	105	93.0	466.0	0	11,5/min	25,46
/ePoR...	52	181	76	89.0	428.0	0	11,8/min	27,02
/ePoR...	24	127	82	45.0	339.0	0	5,3/min	12,64
/ePoR...	100	444	102	308.0	715.0	0	22,3/min	54,58
/ePoR...	100	162	66	122.0	586.0	0	22,3/min	11,13
/ePoR...	252	1051	1258	84.0	6474.0	0	39,4/min	62,86
/ePoR...	50	10	5	8.0	31.0	0	3,5/sec	7,13
TOTAL	672	535	877	8.0	6474.0	0	1,6/sec	150,78

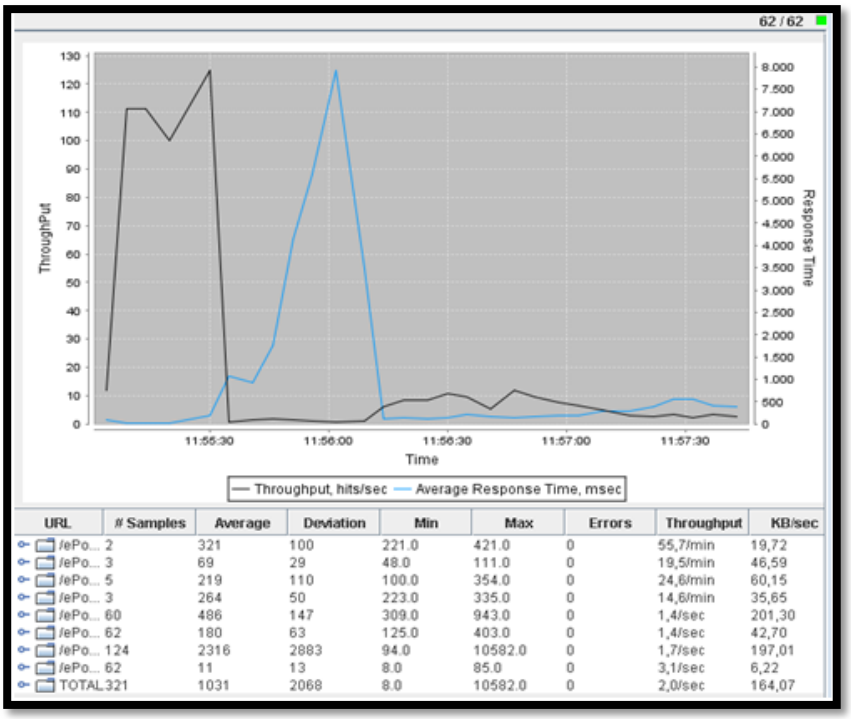
Slika 28: Propusnost in čas odgovor pri 50 reanih uporabnikih.

Obremenjenost stražnika pri 50 istočasnih realnih uporabnikih je prikazana na sliki 29, iz katere lahko vidimo, da je obremenjenost največ 65%. Tudi to je sprejemljiva obremenitev strežnika.



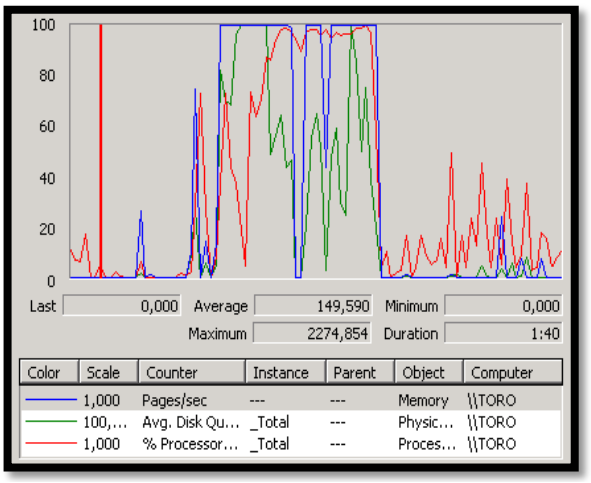
Slika 29: Grafičen prikaz obremenjenosti strežnika pri 50 realnih uporabnikih.

Število realnih uporabnikov sem nato postopoma povečeval in tako prišel do ugotovitve da je največje število realnih uporabnikov 62. To predstavlja tisto število realnih uporabnikov, pri katerim bo aplikacija ePoslovanje še nemoteno delovala. Dobljeni podatki so prikazani na sliki 30.



Slika 30: Propusnost in čas odgovor pri 62 realnih uporabnikih.

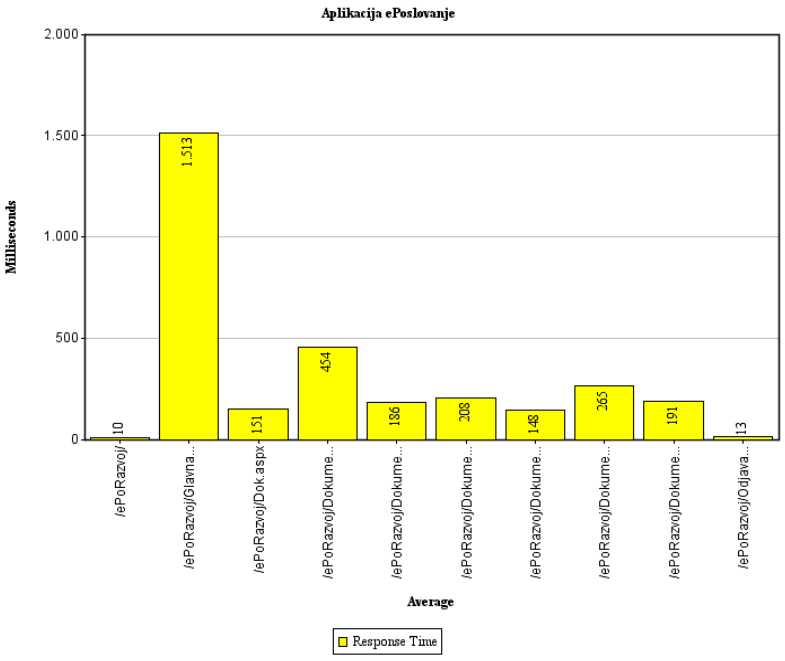
Obremenjenost strežnika pri največjem številu realnih uporabnikov je prikazana na sliki 31.



Slika 31: Propusnost in čas odgovor pri 62 reanih uporabnikih.

Iz slike 31 lahko sklepamo, da pri prenosu podatkov zmanjkuje pomnilnika. Obremenjenost procesorja v tem primeru še ni prehuda. Iz dobljenih podatkov lahko ugotovimo, da bi morali za pohitritev delovanja aplikacije povečati strežniški pomnilnik, ki očitno predstavlja ozko grlo v testnemu okolju za spletno aplikacijo ePoslovanje. Po povečanju pomnilnika naslenjo ozko grlo bi bilo hitrost trdega diska.

Po odpravljanju ozkih grl na strežniku moramo preveriti še ozka grla, ki se pojavijo v aplikaciji. Pregledovanje podatkov o času odgovora za posamezno stran, analiziranje povprečnega časa odgovora za posamezno stran e prikazano na sliki 32.



Slika 32: Povprečen čas odgovora za posamezno stran.

Prva stran, ki ima najmanjši čas odgovora je preusmeritvena stran na prijavno stran. Naslednja pa je prijava v aplikacijo. Vsi uporabniki se prijavijo istočasno v aplikacijo, ki

predstavlja vstopno točko, zato je čas odgovora največji. Naslednji čas, ki odstopa od vseh ostalih je na strani - sprememba podatkov znotraj dokumenta. Vse ostale strani imajo približno enak čas odgovora. Pri tem lahko sklepamo, da moramo optimizirati postopek prijava (nalaganje uporabniških podatkov) in postopek spremembe podatkov znotraj dokumenta (shranjevanje in spreminjanje podatkov).

V situaciji ko imamo več uporabnikov lahko ukrepamo tako, da vsakega naslednjega uporabnika postavimo v čakalno vrsto in ga obvestimo, da je strežnik preobremenjen, in da mora počakati. Drug način je ta, da ga samo obvestimo, naj poskusi kasneje, ko bo strežnik manj obremenjen.

4.2.2.2 Analiza testiranja podatkovnega strežnika

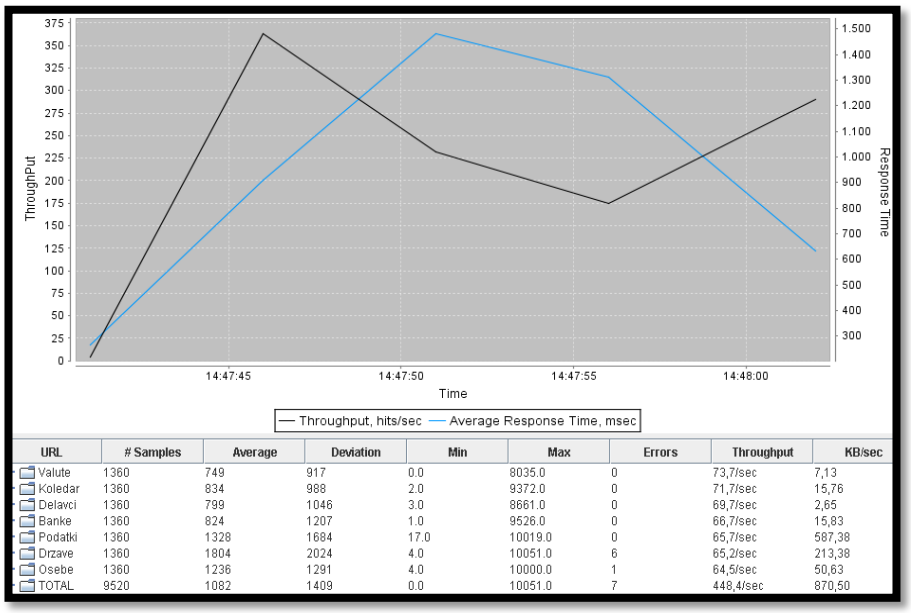
Za testiranje strežnika podatkovnih baz sem uporabil podatkovno bazo, ki vsebuje nekaj več kot 30 tabel. Tesni scenarij vsebuje sedem enostavnih poizvedb. Za izdelavo testnega scenarija sem uporabil naslednje tabele:

`dbo.OSEBE` - šifrant oseb v kadrovski evidenci (8223 zapisov).
`dbo.DRZAVE` - šifrant držav (248 zapisov)
`dbo.PODATKI` - šifrant podatkov (508 zapisov)
`dbo.BANKE` - šifrant bank (32 zapisov)
`dbo.DELAVCI` - šifranti delavcev (6157 zapisov)
`dbo.KOLEDAR` - koledar (3654 zapisov)
`dbo.VALUE` - šifrant valute (30 zapisov)

Vse poizvedbe znotraj testnega scenarija so tipa:

```
SELECT PRIIMEK
FROM dbo.OSEBE
WHERE IME = 'JANEZ'
ORDER BY PRIIMEK
```

Pri 10 navideznih uporabnikih je bila obremenitev podatkovnega strežnika zanemarljiva. Enako je bilo tudi pri 20 in 50 uporabnikih. Število uporabnikov sem nato povečal na 100, pri čemer še vedno ni bilo velikih razlik pri obremenitvi strežnika. Naslednjo število generiranih navideznih uporabnikov je bilo zato kar 500. Pri tem je bila obremenitev CPE približno 70%. Pri 1000 generiranih uporabnikih na enem računalniku v GUI načinu je zmanjkalo pomnilnika. Za distribucijski način izvajanja nisem imel več možnosti. Ugotovil sem da lahko testiram na enem računalniku z največ 680 navideznimi uporabniki. Dobljeni podatki iz testa pri 680 navideznih uporabnikih so prikazani na sliki 33.



URL	# Samples	Average	Deviation	Min	Max	Errors	Throughput	KB/sec
Valute	1360	749	917	0.0	8035.0	0	73,7/sec	7,13
Koledar	1360	834	988	2.0	9372.0	0	71,7/sec	15,76
Delavci	1360	799	1046	3.0	8661.0	0	69,7/sec	2,65
Banke	1360	824	1207	1.0	9526.0	0	66,7/sec	15,83
Podatki	1360	1328	1684	17.0	10019.0	0	65,7/sec	587,38
Drzave	1360	1804	2024	4.0	10051.0	6	65,2/sec	213,38
Osebe	1360	1236	1291	4.0	10000.0	1	64,5/sec	50,63
TOTAL	9520	1082	1409	0.0	10051.0	7	448,4/sec	870,50

Slika 33: Propusnost in čas odgovor pri 680 navideznih uporabnikih.

Iz slike 33 vidimo, da je čas odgovora največ 1,5 sekunde. Iz tega lahko sklepamo, da je za izvajanje testiranja strežnika podatkovnih baz potrebnih več računalnikov. Žal tega testa brez strojne opreme nisem mogel izvesti do konca. Ne glede na to lahko ugotovim, da je propustnost v primeru podatkovnih baz bistveno večja kot pri spletni aplikaciji.

5. Zaključek

JMeter je zelo uporabno orodje za testiranje performančnosti in obremenjenosti spletnih aplikacij, omogoča prilagodljivo in zelo enostavno testiranje. To se je lepo pokazalo pri sestavljanju testnih scenarijev in testiranju spletne aplikacije ePoslovanje.

Pri testiranju spletne aplikacije ePoslovanje sem zato lahko določil največje število istočasnih uporabnikov za nemoteno delovanje aplikacije. Pri največjem številu istočasnih uporabnikov očitno ozko grlo aplikacije predstavlja pomnilnik, sledi hitrost trdega diska ter hitrost procesorja. Testiranje strežnika podatkovnih baz zahteva večje število uporabnikov. To je logično ker je glavni namen podatkovnih baz optimizacija podatkov za hitro iskanje.

Če bi imel več časa bi sestavil še več različnih testnih scenarijev in jih poganjal z različnimi porazdelitvi uporabnikov. Tako bi natančno določil največje število realnih uporabnikov v vseh primerih. Na strežniku bi spremljal obremenitve v daljšem časovnem obdobju za definirane realne uporabnike. Bi uporabil še več poslušalcev in spremljal čas zakasnitev za vsako stran posebej. Te bi podrobneje pokazale ozka grla v aplikaciji. Lahko bi tudi analiziral podatke z pomočjo orodja Apache Ant, ki omogoča branje dobljenih poročil o testiranju, shranjevanje le teh v podatkovno bazo ter njihovo predstavitev.

Dodatek A

Seznam slik

Literatura

[1] Ian Molyneaux. *The art of Application Performance Testing*, United States of America: O'Reilly Media Inc, 2009.

[2] Emily H. Halili. *Apache JMeter*, UK: Packt Publishing, 2008.

[3] (2011) JMeter-plugins. Dostopno na: http://code.google.com/p/jmeter-plugins/downloads/detail?name=JMeterPlugins-0.4.0.zip

[4] (2011) HP functional testing. Dostopno na: https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-127-24^1322_4000_100

[5] (2011) AppPerfect testno orodje. Dostopno na: http://www.appperfect.com/products/web-test.html

[6] (2011) DejaGnu testno orodje. Dostopno na: http://www.gnu.org/software/dejagnu/

[7] (2011) Chalkmark testno orodje. Dostopno na: http://www.optimalworkshop.com/chalkmark.htm

[8] (2011) Google web site optimizer testno orodje. Dostopno na: https://www.google.com/accounts/ServiceLogin?service=websiteoptimizer&continue=http://www.google.com/analytics/siteopt/%3Fhl%3Den&hl=en

[9] (2011) Labsmedia testno orodje. Dostopno na: http://www.labsmedia.com/clickheat/156894.html

[10] (2011) Clubic test testno orodje. Dostopno na: http://cubictest.seleniumhq.org/

[11] (2011) Selenium testno orodje. Dostopno na: http://seleniumhq.org/

[12] (2011) Watir testno orodje. Dostopno na: http://watir.com/

[13] (2011) Netsparker testno orodje. Dostopno na: http://www.mavitunasecurity.com/netsparker/

[14] (2011) Websecurify testno orodje. Dostopno na: http://www.websecurify.com/

[15] (2011) WebScrab testno orodje. Dostopno na: https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

[16] (2011) JMeter domača stran. Dostopno na: http://jakarta.apache.org/jmeter/

[17] (2011) Microsoft SQL Server 2005 podatkovni strežnik. Dostopno na: http://www.microsoft.com/sqlserver/2005/en/us/default.aspx

[18] (2011) FileZilla FTP strežnik. Dostopno na: http://filezilla-project.org/

[19] (2011) ePoslovanje dodatni informacij. Dostopno na: http://www.maop.si/resitve/eposlovanje/

[20] (2011) MCR-PD aplikacija. Dostopno na: http://www.maop.si/resitve/financni_informacijski_sistem/prispeli_dokumenti/

[21] (2011) WinMerge primerjava podatkov. Dostopno na: http://winmerge.org/

[22] (2011) Dostopno na: http://www.microsoft.com/downloads/en/details.aspx?FamilyID=83c3a1ec-ed72-4a79-8961-25635db0192b

[23] (2011) Firebug dodatek za Mozilla Firefox. Dostopno na: http://getfirebug.com/

[24] (2011) Live HTTP header dodatek za Mozilla Firefox. Dostopno na: https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/

[25] (2011) Apache ant dodatno javansko orodje. Dostopno na: http://ant.apache.org/