UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Ivan Jovanovski

TESTIRANJE ZMOGLJIVOSTI SPLETNE APLIKACIJE ORIGAMI DMS

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Miha Mraz
Somentor: doc. dr. Iztok Lebar Bajec

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: 01707/2010
Datum: 01.10.2010

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

Kandidat: **IVAN JOVANOVSKI**

Naslov: **TESTIRANJE ZMOGLJIVOSTI SPLETNE APLIKACIJE ORIGAMI DMS**

PERFORMANCE TESTING OF THE ORIGAMI DMS WEB APPLICATION

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Kandidat naj v svojem delu predstavi osnove testiranja spletnih aplikacij. Po predstavitvi metod in pa konkretnih orodij, s katerimi bi bilo mogoče testiranje izvesti, naj kandidat izvede testiranje spletne aplikacije ORIGAMI DMS, pri razvoju katere je sodeloval.

Mentor:

prof. dr. Miha Mraz

Somentor:

doc. dr. Iztok Lebar Bajec

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

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

diplomskega dela

Spodaj podpisani Ivan Jovanovski,

z vpisno številko 63040407

sem avtor diplomskega dela z naslovom:

Testiranje zmogljivosti spletne aplikacije Origami DMS

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  izr. prof. dr. Mihe Mraza in somentorstvom doc. dr. Iztoka Lebarja Bajca

- 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 _____ Podpis avtorja: Ivan Jovanovski

Za izkazano strokovno podporo, nasvete in pomoč pri izdelavi naloge se iskreno zahvaljujem mentorju izr. prof. dr. Mihi Mrazu in somentorju doc. dr. Iztoku Lebarju Bajcu.

Svoji družini in dekletu se zahvaljujem za potrpežljivost ter vsestransko podporo pri študiju.

Zahvaljujem se tudi vsem kolegom v podjetju SRC d.o.o., posebej Andreju in Tomažu, ki sta me usmerjala in mi vedno pomagala. Posebna zahvala gre tudi Olgi in Markotu, za vsakodnevno podporo in jezikovno pomoč.

Kazalo

Seznam uporabljenih kratic in simbolov

AJAX (angl. *Asynchronous JavaScript and XML*) - skupina medsebojno povezanih spletnih razvojnih tehnik, uporabljenih za ustvarjanje interaktivnih spletnih aplikacij

DMS (angl. *Document Management System*) - sistem za upravljanje z dokumenti in dokumentnimi tokovi

ERP (angl. *Enterprise Resource Planning*) - poslovno informacijski sistem

GUI (angl. *Graphical user interface*) - grafični uporabniški vmesnik

HTTP (angl. *Hyper text transfer protocol*) - protokol za izmenjavo nadbesedil ter grafičnih, zvočnih in drugih večpredstavnostnih vsebin na spletu

IIS (angl. *Internet Information Services*) - programska oprema namenjena gostovanju spletnih aplikacij

IT (angl. *Information technology*) - informacijske tehnologije

LAN (angl. *Local Area Network*) - krajevno omrežje

MTBF (angl. *Mean time between failures*) - povprečni čas med dvema odpovedma

MTTF (angl. *Mean time to failure*) - povprečni čas do odpovedi

MVC (angl. *Model View Controller*) - koncept razvoja spletnih aplikacij

ORM (angl. *Object-Relationship Mapping*) - entitetno-relacijsko povezovanje

RDBMS (angl. *Relationship database management system*) - sistem za upravljanje s podatkovno bazo

SOAP (angl. *Simple Object Access Protocol*) - standard za spletne storitve, ki temelji na XML

SQL (angl. *Structured Query Language*) - strukturirani povpraševalni jezik za delo s podatkovnimi bazami

SUPB Sistem za upravljanje s podatkovno bazo

WAN (angl. *Wide Area Network*) - prostrano omrežje

WCF (angl. *Windows Communication Foundation*) - Microsoftovo ogrodje za gradnjo porazdeljenih aplikacij

TFS (angl. *Team Foundation Server*) - Microsoftov sistem za upravljanje z verzijami izvorne kode

Povzetek

Zmogljivostno testiranje predstavlja vrsto testiranja, ki je namenjeno določanju odzivnosti, prepustnosti, zanesljivosti in/ali razširljivosti sistema pod določenimi obremenitvami. Osredotočeno je na prepoznavanje ozkih grl v sistemu, zagotavljanje izhodišča za prihodnja testiranja ter ugotavljanje skladnosti z zmogljivostnimi cilji in zahtevami. Odlične zmogljivosti lahko dosežemo z izenačitvijo ključnih dejavnikov, ki jih nato vključimo v naše načrte in jim natančno sledimo.

Diplomsko delo ponuja pregled najpomembnejših faz pri zmogljivostnem testiranju spletnih aplikacij, od načina kako razumeti in definirati želeno izkušnjo končnih uporabnikov, izbrati ključne vire za proučevanje, povzeti in interpretirati rezultate na statistično pomenljiv način, ter kako vpeljati navedene prijeme ob testiranju konkretne aplikacije.

Kljub temu, da je diplomsko delo osredotočeno na testiranje spletnih aplikacij, gre za splošen pristop, ki se ga lahko uporabi tudi pri testiranju drugih vrst aplikacij.

Ključne besede:

zmogljivostno testiranje, bremensko testiranje, stresno testiranje, spletni testi, Origami DMS, Fiddler, Visual Studio 2010

Abstract

Performance testing is a type of testing intended to determine the responsiveness, throughput, reliability, and/or scalability of a system under a given workload. Performance testing additionally tends to focus on helping to identify bottlenecks in a system, tuning a system, establishing a baseline for future testing, and determining compliance with performance goals and requirements. We can get great performance by balancing the key factors, considering them in our designs and then tracking them carefully.

The work that follows provides a survey of the most important considerations for performance testing of web based applications, from how to understand and quantify our desired end user experience, how to choose key resources for study, to summarizing the results in a statistically meaningful way, and how we applied these practices while testing our application.

And even though we put the focus squarely on web applications, the approach is actually much more general and can easily be applied for many different kinds of applications, not just web based.

Keywords:

performance test, load test, stress test, web tests, Origami DMS, Fiddler, Visual Studio 2010

1 Uvod

Globalizacija, tehnološka inovativnost in vse hitrejši ter raznoliki komunikacijski kanali danes spreminjajo osnove konkurenčnih prednosti v informacijski industriji. Odziv podjetij mora biti zato hiter in prilagojen spremembam na trgu. Takšne spremembe pa so ustvarile prostor za ponudbo spletno osnovanih sistemov ter s tem podjetjem ponudile nove poslovne priložnosti na trgu. Uporaba spletnih sistemov in aplikacij omogoča podjetjem, da izboljšajo in nadgradijo svoje procese ter poslovne modele.

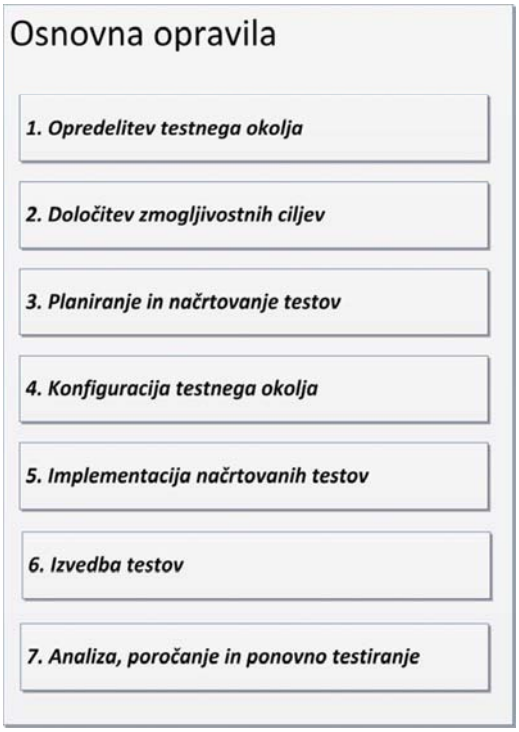
Zagotavljanje zmogljivosti, robustnosti in zanesljivosti spletnih aplikacij je zato postala še pomembnejša zahteva za ponudnike IT storitev.

Testiranje zmogljivosti spletnih aplikacij ni lahka naloga, saj lahko nehote pripravimo neustrezne scenarije ter testiramo nepomembne podatke. Prav tako se nam lahko zgodi, da kljub ustreznim scenarijem in pravilnim podatkom izberemo napačno statistično metodo in analizo ter tako pridobimo napačne sklepe [7].

1.1 Osnovna opravila pri zmogljivostnem testiranju

Zmogljivostno testiranje se običajno opravi kot pomoč pri prepoznavanju ozkih grl v sistemu, za pripravo izhodišča pri prihodnjih testiranjih ter za ugotovitev skladnosti z zmogljivostnimi cilji in zahtevami. Zmogljivostno testiranje izvajamo preko naslednjih osnovnih opravil (glej sliko 1):

1. **Opredelitev testnega okolja:** Opredelimo fizično testno in produkcijsko okolje, kakor tudi programska orodja in sredstva, ki so nam na razpolago. Fizično okolje zajema strojno, programsko in omrežno konfiguracijo. Temeljito razumevanje celotnega testnega okolja nam omogoča učinkovitejše oblikovanje testov in nam pomaga pri prepoznavanju testnih izzivov že od začetka projekta. V nekaterih situacijah je potrebno ta korak ponoviti in spremeniti testno okolje.



Slika 1: Osnovna opravila pri zmogljivostnem testiranju [5].

2. **Določitev zmogljivostnih ciljev**: V tem koraku je potrebno jasno določiti želene odzivne čase, prepustnost in zasedenost virov. Na splošno, so odzivni časi uporabniška skrb, prepustnost poslovna skrb in uporabljeni viri sistemska skrb.
3. **Planiranje in načrtovanje testov:** Identificirati je potrebno ključne scenarije, določiti raznolikost reprezentativnih uporabnikov in določiti način, kako simulirati to raznolikost ter določiti testne podatke in količine, ki jih nameravamo izmeriti.
4. **Konfiguracija testnega okolja:** Pripravimo testno okolje, orodja in vire, ki jih potrebujemo za izvedbo testiranja. Poskrbeti moramo, da je okolje pripravljeno za spremljanje podatkov o zasedenosti virov.
5. **Implementacija načrtovanih testov:** Razvijemo zmogljivostne teste na podlagi predhodnih načrtov.
6. **Izvedba testov:** Izvajamo teste, spremljamo obremenitev in odzivnost sistema. Potrdimo teste, testne podatke in pridobljene rezultate.

7. **Analiza, poročanje in ponovno testiranje:** Konsolidiramo in objavimo rezultate testiranja. Če je potrebno, zamenjamo prioriteto preostalih testov in teste ponovno izvedemo. Ko so vse opazovane vrednosti znotraj določene meje in vse želene informacije zbrane, smo testiranje zaključili.

1.2 Smisel uporabe zmogljivostnih testov

Pri testiranju zmogljivosti smo izpostavljeni tveganjem, ki so povezana s stroški, oportunitetnimi stroški, kontinuiteto poslovanja ter ugledom podjetja. V nadaljevanju so navedeni rezultati, ki jih pridobimo z izvedbo zmogljivostnega testiranja in lahko vključujejo naslednje ocene:

- oceno pripravljenosti za produkcijo:
    - napoved ali ovrednotenje zmogljivosti aplikacije, ko se le-ta nahaja v produkcijskem okolju;
    - zagotovitev podatkov, ki kažejo na verjetnost nezadovoljstva uporabnikov z delovanjem sistema;
    - zagotovitev podatkov za pomoč pri napovedovanju prihodkov, izgube ali upada ugleda družbe zaradi težav, povezanih z razširljivostjo in stabilnostjo, ter zaradi nezadovoljstva uporabnikov z odzivnimi časi;
- oceno primernosti infrastrukture:
    - ocena ustreznosti trenutne zmogljivosti;
    - določitev sprejemljive stabilnosti;
    - določitev sredstev, potrebnih za zagotavljanje sprejemljive zmogljivosti infrastrukture;
- oceno ustreznosti razvite programske opreme:
    - ugotovitev zmogljivosti aplikacije pred in po spremembah v kodi;
    - primerjava trenutne in želene značilnosti delovanja;

- izboljšavo učinkovitosti aplikacije:
    - analiza obnašanja pod različnimi obremenitvami;
    - odprava ozkih grl.

1.3 Zmogljivostno, bremensko in stresno testiranje

Zmogljivostni testi so opredeljeni z eno izmed spodaj navedenih treh kategorij:

- zmogljivostni testi,
- bremenski testi,
- stresni testi.

V nadaljevanju so podrobneje opisani cilji posameznih kategorij testov.

1.3.1 Zmogljivostni testi

Določijo ali potrdijo hitrost, prilagodljivost in/ali stabilnost aplikacije. Zmogljivostni test je tehnična preiskava, ki določi ali potrdi odzivnost, hitrost, prilagodljivost in/ali stabilnostne lastnosti aplikacije, katero preizkušamo.

1.3.2 Bremenski testi

Cilj testov je preveriti obnašanje aplikacije pri normalni obremenitvi in v primeru maksimalne predpisane obremenitve.

Bremensko testiranje se izvaja za preverjanje ali aplikacija izpolnjuje zastavljene zmogljivostne kriterije. Kriteriji so ponavadi definirani v pogodbi med izvajalcem in naročnikom. Bremenski test nam omogoča izmeriti odzivne čase, prepustnost in porabo virov. Omogoča tudi identifikacijo prelomne točke v primeru, da se ta zgodi pred maksimalno predpisano obremenitvijo.

Vzdržljivostno testiranje je podmnožica bremenskega testiranja. Osredotoča se na določanje ali potrjevanje obnašanja aplikacije v daljšem časovnem obdobju.

Vzdržljivostne teste lahko uporabimo za izračun povprečnega časa med odpovedmi (MTBF) in povprečnega časa do prve odpovedi (MTTF).

1.3.3 Stresni testi

Določajo ali potrjujejo obnašanja aplikacije, ko obremenitev preseže normalne oz. maksimalne predpisane meje.

Cilj stresnega testiranja je odkriti napake v aplikaciji, ki se pojavijo le pod pogoji visoke obremenitve. Napake lahko vključujejo sinhronizacijske probleme ali puščanje pomnilnika (angl. *memory leaks*). Zmogljivostno testiranje nam omogoča identifikacijo šibkih točk in prikaže obnašanje aplikacije v ekstremnih obremenitvah.

»Spike test« je podmnožica stresnega testiranja. Osredotoča se na določanje obnašanja aplikacije, ko obremenitev za kratek čas preseže maksimalne predpisane meje.

1.4 Pridobitve posameznih testnih kategorij

Vsaka kategorija zmogljivostnih testov nam poda določen nabor podatkov o naši informacijski rešitvi (aplikaciji in infrastrukturi). Tabela 1 opisuje pridobljene podatke pri posamezni kategoriji zmogljivostnih testov.

Koristi, ki jih pridobimo z zmogljivostnim testiranjem, so običajno večje od stroškov, ki nastanejo ob zmogljivostnem testiranju. Zaradi negotovosti rezultatov testiranja veliko podjetij dvomi o smotrnosti zmogljivostnega testiranja (predvsem ker niso sposobna testiranja izpeljati pravilno) [4]. Kljub temu pa nam praksa kaže na to, da zmogljivostno testiranje, čeprav nedosledno, pripomore k zmanjšanju verjetnosti izpadov sistema v produkciji. Prav tako nam pomaga določiti kritično skupino parametrov, ki nas opozarjajo na poslabšano delovanje aplikacije, ali na neposredni izpad sistema.

Kategorija	***Koristi***	***Nepredvidena področja in izzivi***
Zmogljivostni testi	• Določajo hitrost, prilagodljivost in stabilnost aplikacije, kar zagotavlja podporo pri poslovnih odločitvah. • Osredotočajo se na uporabnikovo zadovoljstvo z odzivnimi časi aplikacije. • Določajo neskladnost med realnimi in pričakovanimi zmogljivostmi.	• Možno je, da ne odkrijemo nekaterih funkcionalnih napak, ki se pojavljajo le pod obremenitvijo. • Če niso pravilno začrtani, lahko kažejo na značilnosti delovanja zelo majhnega števila realnih scenarijev. • Razen v primeru, ko se testi izvajajo na produkcijskih strežnikih in je breme generirano iz dejanskih uporabniških računalnikov, bo vedno nekaj odstopanja v rezultatih.
Bremenski testi	• Določajo potrebno zmogljivost virov za produkcijsko okolje. • Določajo ustreznost strojne opreme. • Odkrivajo težave pri sočasnem izvajanju transakcij. • Odkrivajo funkcionalnostne napake pod obremenitvijo. • Zbirajo podatke z namenom planiranja razširljivosti in skalabilnosti. • Pomagajo določiti število sočasnih uporabnikov preden je ogroženo delovanje aplikacije. • Pomagajo določiti količino obremenitve, ki je potrebna, da postane uporaba virov s strani sistema kritična.	• Niso namenjeni zgolj za določitev hitrosti odziva. • Rezultate uporabljamo za primerjavo z drugimi sorodnimi obremenitvenimi testi.

Stresni testi	• Določajo ali pri prekomerni obremenitvi sistema pride do nekonsistentnosti oz. napak v podatkih. • Omogočajo oceniti obremenitev aplikacije pri kateri poleg upočasnjenega delovanja začne aplikacija tudi nepravilno delovati. • Zagotavljajo, da se z obremenitvijo ne odpirajo varnostne luknje. • Definirajo neželene učinke izpadov strojne in programske opreme.	• Ker so stres testi nerealni glede na realen model uporabe, lahko nekatere stranke zavržejo rezultate testov. • Pogosto je težko predvideti, kakšno obremenitev je smiselno uporabiti. • V kolikor testno okolje ni izolirano, lahko povzročimo izpad omrežja, ali nepravilno delovanje ostalih aplikacij.

Tabela 1: Pridobitve posameznih testnih kategorij.

V predhodnih poglavjih in odstavkih smo opisali teoretični del zmogljivostnega testiranja. V naslednjih poglavjih se nahajajo ugotovitve, do katerih smo prišli ob zmogljivostnem testiranju aplikacije Origami DMS v praksi.

2 Origami DMS

Origami DMS je sodoben, na Microsoftovi .NET tehnologiji razvit sistem v podjetju SRC d.o.o. za upravljanje z dokumenti in dokumentnimi tokovi. Je uporabniku prijazen in enostaven za delo. Dokumentne procese ali tokove poenostavlja in jih časovno skrajšuje, hkrati pa preprečuje razne nepravilnosti, odklone, nepotrebna zastajanja in izgubljanja dokumentov, nepooblaščene vpoglede in odtujevanja. Vzpostavlja standardiziran, sistematiziran in pregleden način dela z dokumenti, kjer se točno ve, kdo, kdaj in kaj mora narediti v zvezi z določenimi zadevami ali dokumenti. Uporabniku pri razporejanju dela omogoča, da se le-ta lahko osredotoči na svoje primarne naloge in aktivnosti.

2.1 Prednosti uporabe DMS sistema

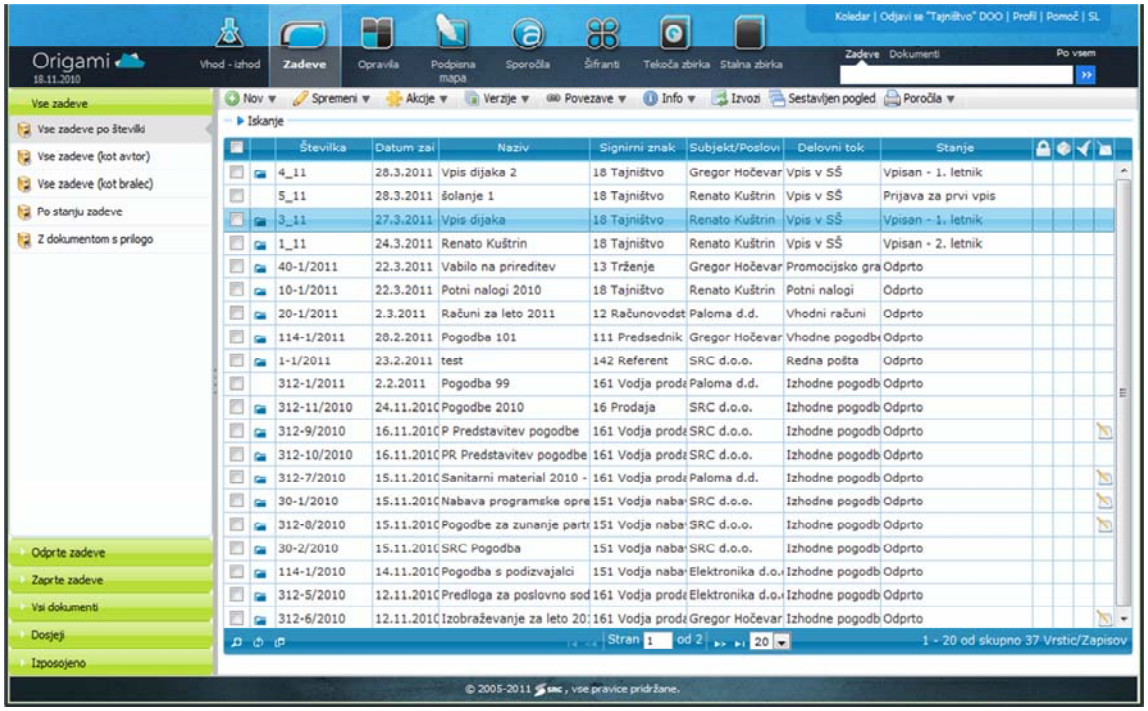
Aplikacijo smo implementirali na področju zdravstva, gospodarstva in javne uprave. Uporabniki po uvedbi sistema za upravljanje z dokumenti poročajo, da so procesi preglednejši, krajši in nedvoumni, kar se posledično odraža tudi pri drugih aktivnostih, ki so povezane z uporabo oziroma upravljanjem z dokumenti. Povečala se je učinkovitost, izboljšala se je operativnost, informacije pa so bile pravočasne in na voljo pravim uporabnikom. S tem se je povečalo zadovoljstvo zaposlenih, odločitve vodstva pa so lahko bolj kakovostne in temeljijo na točnih podatkih. Vse to je pomembno pri ocenjevanju priložnosti, pripravi in pridobivanju novih poslov.

Ob uvedbi sistema v podjetjih opazimo, da se je v finančni operativi povečala pravočasnost izvedbe postopkov in izboljšala informiranost. Sestavljanje in pošiljanje poročil je olajšano, podatki se vnašajo samo enkrat in posledično se zmanjša število napak (vhod podatkov v sistem je skozi dokumentni sistem, potem pa se le-ti avtomatično prenesejo v ERP). S plačevanjem računov stranke ne zamujajo. Postopki se izvajajo v realnem času in »online«, vhodni dokumenti prihajajo sproti in postopoma zato ni več čakanja na dokumente, kakovost podatkov pa je zaradi zmanjšanega števila napak pri vnosih bistveno večja.

Vodstveni nivoji ne potrebujejo več t.i. priročnih evidenc in arhivov, kopiranja in hranjenja »papirnih varnostnih kopij« pa skorajda ni več. Iskanje, priklic, pregled ter uporaba dokumentov so bistveno hitrejši in preprostejši. Poslovne prednosti in lastnosti Origami DMS sistema so:

- manjši stroški poslovanja:
    - zmanjšanje dolgotrajnih procesov urejanja papirne dokumentacije;
    - zmanjšanje stroškov z uporabo sistema za spremembe in dopolnitve, hranjenje in iskanje;
    - zmanjšanje prostora, potrebnega za arhiv in hranjenje papirnih dokumentov;
    - zmanjšanje stroškov za kopiranje in tiskanje ter za vzdrževanje kopirnih strojev in tiskalnikov;
- večja učinkovitost in produktivnost:
    - popolna prilagodljivost poslovnim procesom;
    - kratek čas implementacije in izobraževanja;
    - pregleden, preprost, učinkovit in varen način upravljanja s pomembnimi poslovnimi dokumenti;
    - centralizacija arhiviranja, dostopnost, varnost in nadzor nad dokumenti;
    - boljša dinamika poslovanja, večja učinkovitost in zadovoljstvo zaposlenih;
    - učinkovita podpora pri sprejemanju odločitev;
    - standardizacija poslovanja;
    - lažje delo s poslovnimi partnerji in strankami;
    - integracija s sistemi ERP.

Origami DMS (glej sliko 2) je na voljo v slovenskem, srbskem, hrvaškem, makedonskem, črnogorskem, bosanskem in angleškem jeziku z možnostjo prevajanja v katerikoli želen jezik.

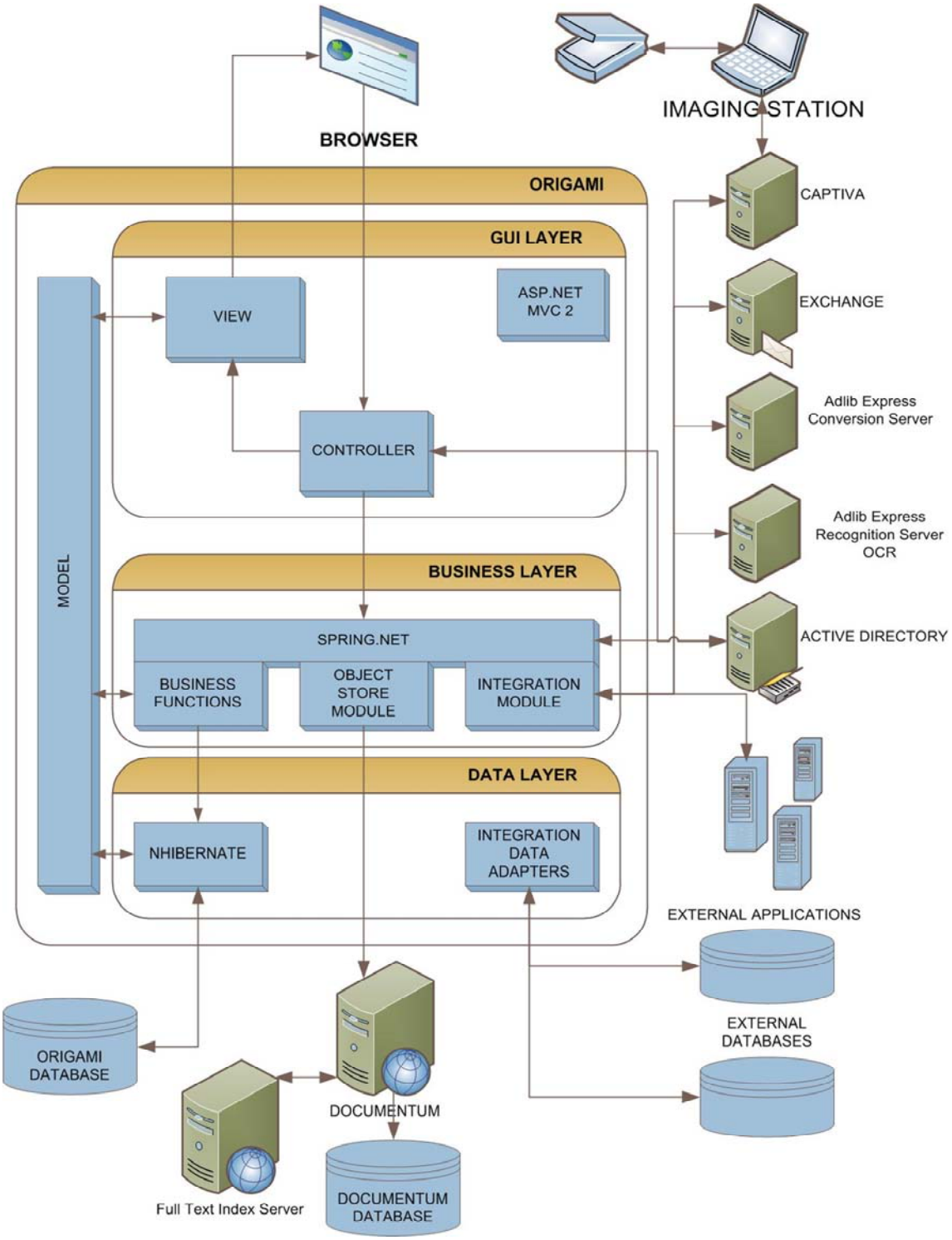


Slika 2: Osnovni pogled Origami DMS aplikacije.

2.2 Arhitektura aplikacije

Origami DMS je razvit na Microsoftovi .NET tehnologiji z uporabo specifične aplikativne arhitekture, ki izpolnjuje pogoje za hitro in varno rešitev. Zaradi lažjega obvladovanja kompleksnosti, zagotavljanja razširljivosti in zmogljivosti, je aplikacija razdeljena na naslednje nivoje (glej sliko 3):

- GUI nivo,
- poslovni nivo,
- podatkovni nivo.



Slika 3: Arhitekturno-funkcionalni pogled aplikacije.

2.2.1 GUI nivo

Z namenom obvladovanja nadzora nad vsebino, ki se pošilja klientu (brskalnik IE ali Firefox), smo za GUI nivo izbrali APS.NET MVC ogrodje (angl. *framework*).

MVC je programska arhitektura oz. koncept, ki se uporablja v aplikativnem inženirstvu. Koncept izolira »domensko logiko« (logika aplikacije) od uporabniškega vmesnika (vnos in prikaz podatkov) in s tem omogoča neodvisen razvoj, preizkušanje in vzdrževanje (razmejitev pristojnosti/skrbništva).

Model predstavlja specifično predstavitev podatkov, na katerem temelji aplikacija, poslovna logika pa se na modelu uporablja za ovrednotenje podatkov. MVC ne definira pravil za shranjevanje podatkov.

Pogled (View) predstavlja model v obliki, ki je prijazna uporabniku. Gre za grafično predstavitev modela v obliki uporabniškega vmesnika, kjer lahko za vsak objekt modela obstaja več pogledov. Relacija je lahko tudi obratna.

Krmilnik (Controller) spremlja dogodke (največkrat s strani uporabnika), pripravlja model in ga posreduje pogledom oz. poslovni logiki.

2.2.2 Poslovni nivo

Spring.NET ogrodje skrbi za komunikacijo med nivoji. Spring.NET omogoča dinamično spreminjanje implementacije posameznih vmesnikov in enostavno spremembo dela aplikacije, ki uporablja funkcionalnost, definirano v teh vmesnikih. Spring.NET poskrbi tudi za varnost na podatkovnem nivoju in omogoča deklarativno specifikacijo varnostnih zahtev.

V sklopu Spring.NET-a se za komunikacijo med nivoji uporablja Windows Communication Foundation (WCF), preko katerega je klic spletne storitve mogoče realizirati na najbolj optimalen način za trenutno konfiguracijo aplikacije. WCF komunikacija med nivoji je lahko:

- komunikacija prek LAN in WAN omrežja (HTTP in SOAP),
- komunikacija preko LAN na majhni oddaljenosti od strežnika (Binarna komunikacija),
- komunikacija med nivoji aplikacije na istem strežniku (Memory pipes).

2.2.3 Podatkovni nivo

Za interakcijo s podatki v podatkovni bazi se uporablja NHibernate. NHibernate je objektno relacijsko preslikovalno orodje (ORM), ki preslika podatke iz podatkovne baze v poslovne objekte znotraj aplikacije.

NHibernate istočasno predstavlja tudi abstrakcijski nivo aplikacije, ki omogoča enostavno spremembo sistema za upravljanje s podatkovno bazo – SUPB (angl. *RDBMS*). To omogoča, da je rešitev neodvisna od SUPB (SQL Server, Oracle, ….).

3 Izvedba testiranja

3.1 Opredelitev testnega okolja

Za učinkovitejše oblikovanje testov in določitev zmogljivostnih ciljev je najprej potrebno opredeliti testno okolje. Testno okolje je sestavljeno iz:

- strojne opreme, na kateri izvajamo teste,
- omrežij, ki povezujejo strojno opremo,
- orodij za testiranje, ki so nam na voljo,
- ostale programske opreme,
- zunanjih vplivov na testno okolje.

3.1.1 Strojna oprema

V tabeli 2 je opisana strojna in programska oprema, ki je bila uporabljena za zmogljivostno testiranje naše aplikacije:

Ime računalnika	***Programska oprema***	***Strojna konfiguracija***
ORIGSTRS	• Windows Server 2008R2 • Origami Facade Service • Origami Web • SQL Server 2008 • IIS 7.5	*Procesor*: 2x QuadCore Intel Xeon E5507 *Pomnilnik*: 2x4GB ECC DDR2-800MHz *Chipset*: Intel Tylersburg 5500 *Trdi disk*: 6x 300GB *Mrežna kartica*: 2x Intel 82575EB Gigabit Ethernet Card
NETSERVER	• Windows Server 2008R2 • Visual Studio 2010 • SQL Server 2008 • Fiddler • Office 2010	*Procesor*: Intel Core 2 Duo E6550 *Pomnilnik*: 4x1GB DDR2-667 MHz *Chipset*: Intel Bearlake Q35 *Trdi disk*: 160 GB, 7200 RPM, SATA-II *Mrežna kartica*: Intel 82566DM-2

TSTCLIENT	• Windows 7 • Visual Studio 2010 • SQL Server Express 2008 • Office 2010	*Procesor*: Intel Core 2 Duo T9300 *Pomnilnik*: 2x2GB DDR2-667 MHz *Chipset*: Intel PM965 *Trdi disk*: 120 GB, SSD, SATA-II *Mrežna kartica*: Intel 82566MM Gigabit Network
Klient 1	• Windows XP • IE 8 • Firefox 3.6 • Office 2007	*Procesor*: Intel Core 2 Duo T7700 *Pomnilnik*: 2GB DDR2-667MHz *Chipset*: Mobile Intel GM45 *Trdi disk*: 240GB, 5400 RPM, SATA-II *Mrežna kartica*: 54Mbit Wireless
Klient 2	• Windows Vista • IE 8 • Firefox • Office 2010	*Procesor*: Intel Core 2 Duo E8200 *Pomnilnik*: 2x2GB DDR2-667MHz *Chipset*: Intel P45 *Trdi disk*: 1TB, 7200 RPM, SATA-II *Mrežna kartica*: 100Mbit Ethernet

Tabela 2: Opis strojne in programske opreme.

ORIGSTRS je strežnik srednjega cenovnega razreda, namenjen majhnim do srednje velikim podjetjem ter primeren za do maksimalno 500 uporabnikov. Strežnik bo gostil našo aplikacijo ter SQL Server.

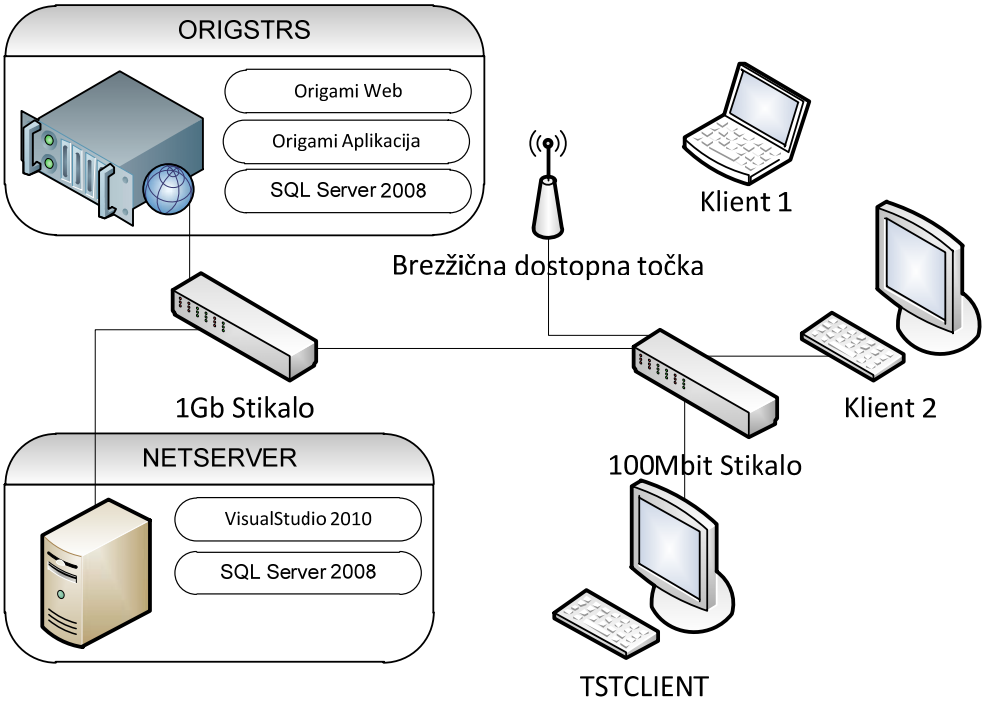
NETSERVER predstavlja delovno postajo prirejeno za generiranje bremena.

TSTCLIENT računalnik smo namenili spremljanju in shranjevanju rezultatov testiranja.

KLIENT 1 in 2 sta delovni postaji namenjeni sprotnemu preverjanju odzivnosti strežnika med obremenitvijo s strani uporabnikov. Njihova strojna konfiguracija ima zanemarljiv vpliv na odzivnost aplikacije.

3.1.2 Omrežje

Da bi dobili čim bolj natančne podatke o zmogljivosti strežnika in zmanjšali zunanje vplive, smo strežnika ORIGSTRS in NETSERVER povezali neposredno preko 1Gbitnega stikala. Na stikalo smo povezali drugo 100 Mbitno, kjer sta priključena dva klienta ter računalnik za spremljanje rezultatov testiranja. Konfiguracija omrežja je podana na sliki 4. Zaradi boljše preglednosti v shemo ni vključena ostala mrežna infrastruktura podjetja.



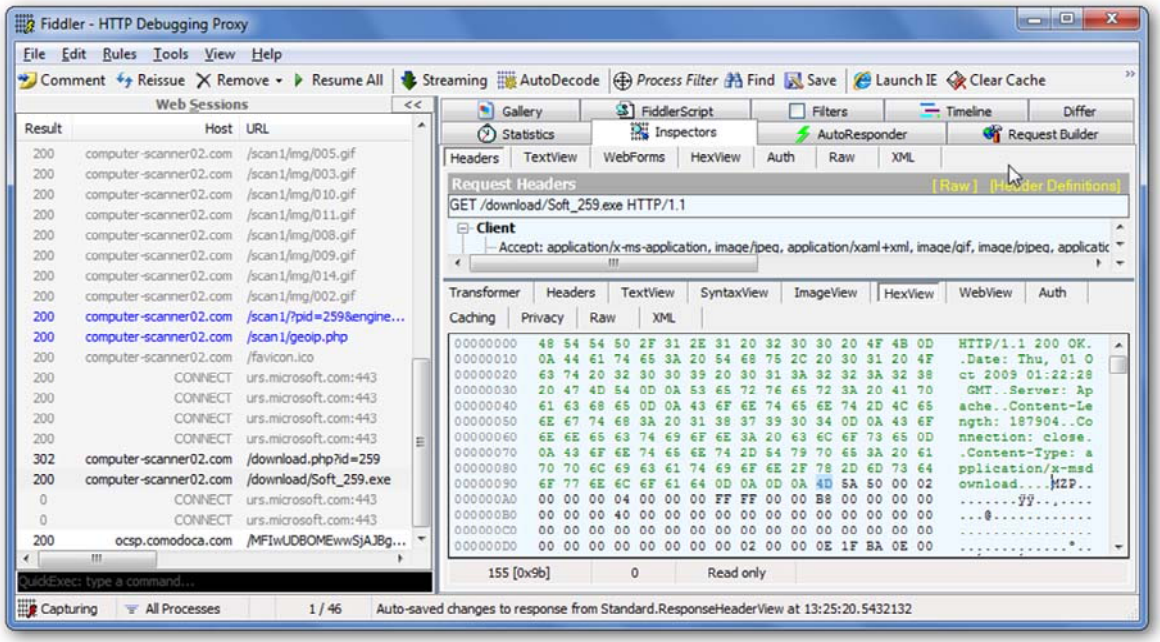
Slika 4: Shema omrežja testnega okolja.

3.1.3 Orodja za testiranje

V večini primerov orodja za testiranje predstavljajo programsko opremo, ki nam pomaga pri izvedbi zmogljivostnega testiranja. Včasih strojna oprema dopolni oz. celo nadomesti funkcionalnosti programske opreme, vendar je to prej izjema kot pravilo – predvsem zaradi cene take strojne opreme in omejenega nabora njenih funkcionalnosti. Zaradi omenjenih razlogov smo pri testiranju aplikacije Origami uporabili le programsko opremo, ki je opisana v nadaljevanju.

3.1.3.1 Fiddler

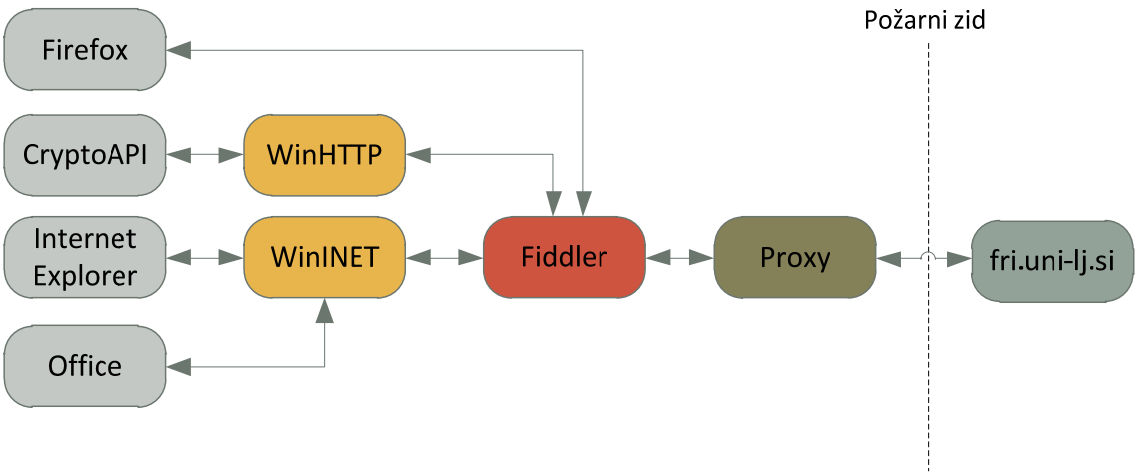
Fiddler (glej sliko 5) predstavlja enega izmed najpogosteje uporabljenih programov za prestrezanje spletnega prometa. Napisan je v programskem jeziku C# in je na voljo brezplačno.



Slika 5: Osnovno okno orodja Fiddler.

Program se vključi med proces WinInet (angl. *Windows Internet*) in našo zunanjo povezavo kot namestniški (angl. *proxy*) strežnik kot je prikazano na sliki 6, in spremlja, kaj se dogaja na tem odseku. To nam omogoča filtriranje celotnega HTTP-prometa [3].

Program deluje na podlagi zajemanja določenih paketov. Na naši strani se prikaže vsaka seja posameznega segmenta v svoji barvi. Iz prikaza razberemo, kaj se dogaja z zahtevo in velikostjo posameznega segmenta, ki prihaja k odjemalcu. Vsak klik na posamezni segment nam ponudi še dodatno razlago (velikost, zahtevo, stanje predpomnilnika, kontekstni tip, itd.). S klikom na posamezni del dobimo preko oglednih oken še dodatne informacije. Programska oprema Fiddler izriše graf, na katerem vidimo kolikšna količina bitov je bila poslana in sprejeta. Za vsak odsek vidimo tudi po odstotkih, kolikšen del zasede. V tem delu dobimo tudi celovito informacijo o glavi prenesenih paketov ter lastne informacije posameznega paketa.



Slika 6: Prikaz delovanja orodja Fiddler.

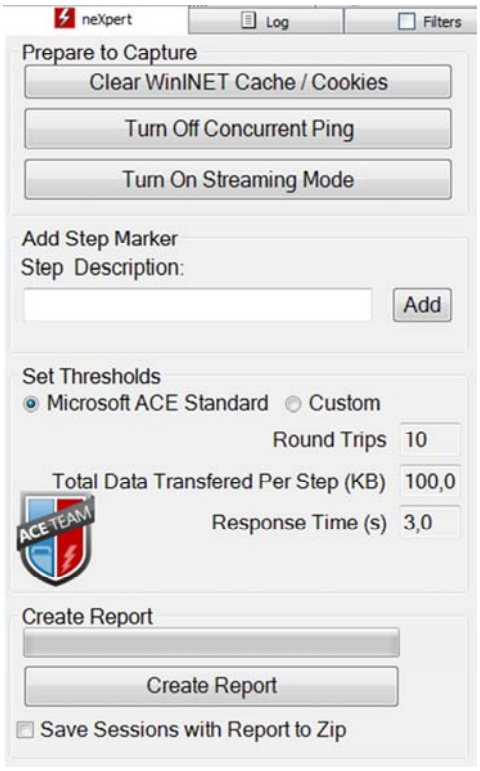
Program Fiddler nam poleg prikaza HTTP zahtevkov in odzivov, omogoča še postavitev prekinitvene točke. Ko je opcija »Enable Single Step Debugging« omogočena v meniju »Rules«, ali ko se določene lastnosti HTTP zahtevka/odziva ujemajo z opredeljenimi kriteriji, Fiddler lahko ustavi HTTP promet in nam dovoli urejanje.

Največja prednost Fiddler-ja je, da omogoča razširitev svojih funkcionalnosti z namestitvijo dodatnih plugin-ov ter shranjevanje zajetih HTTP zahtevkov v obliki spletnega testa[1], ki je namenjen uporabi v okolju Visual Studio.

3.1.3.2 Dodatek neXpert

Dodatek neXpert (glej sliko 7) je dodatek za Fiddler, ki pomaga pri izvedbi testiranja spletnih aplikacij. Ustvarjen je z namenom zmanjšati čas, potreben za iskanje zmogljivostnih težav spletnih aplikacij.

[1]Spletni test (angl. *Web test*) predstavlja posnetek interakcije med uporabnikom in spletno aplikacijo.



Slika 7: Dodatek neXpert znotraj Fiddler-ja.

Dodatek nam zagotavlja ne le informativno poročilo o vseh ugotovljenih težavah, povezanih z zmogljivostjo, ampak tudi ustvarja napovedi odzivnih časov aplikacije v trenutnem stanju. Uporabnika tudi usmerja k doseganju boljših rezultatov z uporabo najboljših praks pri optimizaciji spletnih aplikacij. Program nam omogoča ustvarjanje transakcij znotraj Fiddler-ja, katere pomagajo pri logični združitvi posameznih med seboj povezanih HTTP zahtevkov in odzivov. Dodatek neXpert poskuša s pomočjo transakcij ugotoviti zmogljivostne težave v določenih primerih uporabe aplikacije. Z uporabo velikosti posameznih zahtevkov in odzivov ter strežniškim časom, ki je potreben za procesiranje posameznega zahtevka, poskuša napovedati odzivni čas uporabnika. Ocena temelji na načinu nalaganja vsebine spletne aplikacije v različnih brskalnikih [8]. Dodatek neXpert je razvit na podlagi TCP modela v Microsoftovem oddelku za raziskave.

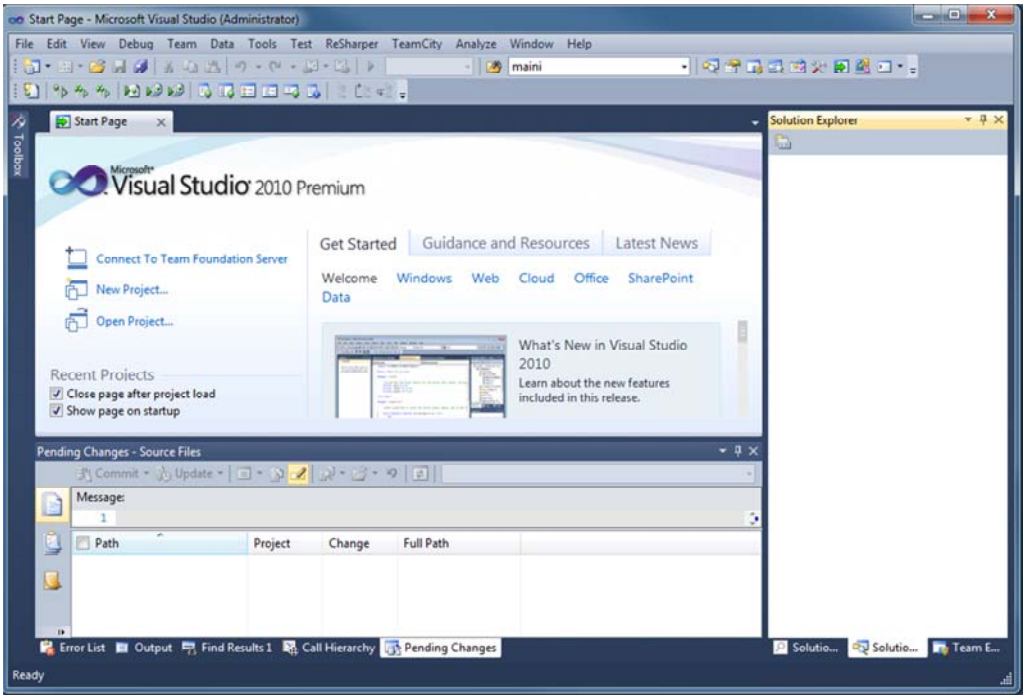
Seznam možnosti, ki nam jih ponuja dodatek neXpert:

- omogoča dodajanje oznak med zajemanjem podatkov za označevanje posameznih sklopov korakov ali klikov znotraj enega scenarija;
- enostaven dostop do gumbov za testiranje zmogljivosti s pomočjo Fiddler-ja;
- ping strežnika med zajemom za izračun trenutne zakasnitve;
- napoveduje osnovne odzivne čase na podlagi različnih zakasnitev in spletnih brskalnikov;
- kreiranje HTML poročila, ki vsebuje:
    - HTTP odzivne kode,
    - ASP.NET View State[1],
    - statične datoteke,
    - velike slike,
    - stiskanje podatkov,
    - avtentikacijo,
    - ETag-e[2],
    - glave predpomnilnika,
    - glave povezave,
    - piškotke (angl. *cookies*).

3.1.3.3 Visual Studio 2010

Razvojno okolje Visual Studio (glej sliko 8), ki ga je razvilo podjetje Microsoft, vključuje celovit nabor orodij, ki razvijalcem programske opreme pomaga premagovati kompleksne izzive in ustvarjati inovativne rešitve [6]. Namen zbirke Visual Studio je izboljšati razvojni proces ter podpreti razvojne projekte, usmerjene v spletne rešitve (vključno z aplikacijami ASP.NET in MVC), izdelke Windows 7, Windows Server 2008, sistem Microsoft Office 2010 in SQL Server 2010 ter naprave Windows Mobile 7.

[1] ASP.NET View State je tehnika, ki se uporablja s strani ASP.NET aplikacije za ohranjevanja stanja posamezne strani med »postback«-i.
[2] ETag ali oznaka entite (angl. *entity tag*) predstavlja enega izmed mehanizmov, ki ga HTTP protokol ponuja za validacijo predpomnilnika.



Slika 8: Razvojno okolje Visual Studio 2010.

V razvojnem okolju Visual Studio 2010 je poudarek na testiranju zmogljivosti in odpravljanju programskih napak (razhroščevanju) aplikacij. Testiranje spletnih aplikacij in obremenitveno testiranje obstajata že v različici 2005, vendar Visual Studio 2010 ponuja pomembne izboljšave.

Dodatne zanke in pogoji nam omogočajo, da ustvarimo bolj kompleksne in inteligentne spletne teste. Pri obremenitvenemu testiranju so dodani 64-bitni agenti in krmilniki, ki nam omogočajo učinkovito uporabo razpoložljivih virov strojne opreme za generiranje bremena. Prav tako nam sprememba licenčne politike obremenitvenih agentov in krmilnikov ponuja večjo fleksibilnost pri konfiguraciji testnega okolja.

Spletni testi nam omogočajo, da simuliramo uporabnika, ki izvaja niz aktivnosti oziroma predefiniran primer uporabe na naši spletni aplikaciji. S preverjanjem veljavnosti odgovorov potrdimo pravilno delovanje aplikacije. Ko imamo posamezne teste že definirane in implementirane, jih lahko uporabimo za izvedbo obremenitvenega testa.

3.1.4 Ostala programska oprema

Na strežniku ORIGSTRS je poleg naše aplikacije nameščena še programska oprema za odpravljanje programskih napak na daljavo, »profiling« ter beleženje morebitnih napak. Našteta programska oprema po specifikaciji proizvajalca upočasni delovanje sistema do 10 %.

Delovni postaji Klient 1 in 2 imata nameščen programski paket Office, protivirusno zaščito, spletna brskalnika Internet Explorer 8 in Mozilla Firefox 3.6, Skype ter klient za elektronsko pošto.

3.1.5 Zunanji vplivi

S ciljem zmanjševanja zunanjih vplivov (spremembe domenske politike, prisilno protivirusno skeniranje, ipd.), smo strežnik ORIGSTRS ter delovno postajo NETSERVER odstranili iz skupne domene. Na obeh računalnikih je bilo onemogočeno samodejno posodabljanje ter ustvarjanje obnovitvenih točk operacijskega sistema. Na ORIGSTRS smo nastavili samodejno kreiranje varnostne kopije podatkovne baze v času, ko se obremenitveni testi ne izvajajo.

3.2 Določitev zmogljivostnih ciljev

Odzivni časi, prepustnost ter zasedenost strojne opreme, ki jih navajamo v nadaljevanju, predstavljajo želene zmogljivostne cilje, ki jih želimo doseči pri obremenitvi 250 uporabnikov na strežniku ORIGSTRS. Izhodiščno stanje sistema je 100.000 zadev in 300.000 dokumentov.

3.2.1 Odzivni časi

Prenos podatkov preko omrežja lahko v veliki meri vpliva na odzivne čase sistema. Pogoji in omejitve, vezani na vpliv prenos podatkov prek omrežja, so podani v tabeli 3:

Tip operacije	*Povprečni odzivni čas*	*Maksimalni odzivni čas*	*Pogoji*
Pregled šifranta	3s	10s	Prenos podatkov preko omrežja zasede največ 10% navedenega časa
Popravek šifranta	3s	10s	Prenos podatkov preko omrežja zasede največ 10% navedenega časa
Pregled seznama zadev	5s	20s	Prenos podatkov preko omrežja zasede največ 10% navedenega časa
Pregled seznama dokumentov	5s	20s	Prenos podatkov preko omrežja zasede največ 10% navedenega časa
Pregled metapodatkov dokumenta	4s	20s	Prenos podatkov preko omrežja zasede največ 10% navedenega časa
Popravek metapodatkov dokumenta	5s	25s	Prenos podatkov preko omrežja zasede največ 10% navedenega časa
Prenos datoteke	10s	30s	Prenos podatkov preko omrežja zasede največ 50% navedenega časa

Tabela 3: Pogoji in omejitve odzivnih časov.

Povprečni in maksimalni odzivni časi temeljijo na naslednjih predpostavkah:

- Delovne postaje, na katerih spremljamo odzivne čase, izpolnjujejo minimalne strojne in programske konfiguracijske zahteve, predpisane za normalno delovanje aplikacije.
- Delovne postaje morajo med delovanjem imeti na voljo dovolj pomnilnika. Ostala programska oprema, ki je že nameščena in se izvaja, pa ne sme porabiti več kot 15% procesorskega časa.
- Ostale aplikacije ne smejo obremenjevati mrežne kapacitete potrebne za normalno delovanje Origami DMS sistema.
- Odzivne čase merimo s prijavljenim uporabnikom v sistemu.

- Nastavitve spletnega brskalnika omogočajo normalno delovanje aplikacije. Gre za sledeče nastavite:
    - omogočeni piškotki;
    - varnostne nastavitve omogočajo nameščanje podpisane ActiveX kontrole in izvajanje JavaScript.

3.2.2 Prepustnost sistema

Sistem mora biti zmožen podpirati dodajanje 30 zadev in 100 dokumentov na minuto v okviru želene obremenitve virov.

3.2.3 Omejitev uporabe virov

Povprečna poraba procesorja med časom polne obremenitve ne sme presegati 75%.

Povprečna poraba pomnilnika s strani naše aplikacije ne sme naraščati, ko je doseženo maksimalno število uporabnikov.

3.3 Planiranje in načrtovanje testov

Na podlagi opravljene analize uporabe sistema, smo za testiranje zmogljivosti aplikacije definirali dva ključna scenarija, ki ju uporabniki izvajajo povprečno 80% časa.

3.3.1 Scenarij 1: Kreiranje nove zadeve

Pri kreiranju nove zadeve se izvedejo naslednje aktivnosti:

1. Uporabnik v meniju za izbiro modulov izbere modul »Zadeve«.
2. V sistemu se odpre osnovno okno modula »Zadeve«.
3. V meniju za izbiranje pogleda uporabnik izbere ustrezen pogled zadev.
4. V sistemu se ponudi seznam pogledov.
5. Uporabnik izbere pogled v katerem želi izdelati novo zadevo.

6. V sistemu se odpre izbrani pogled s tabelo vnesenih zadev in dokumentov do katerih ima uporabnik pravice in orodno vrstico akcij, ki jih je mogoče izvajati.
7. Uporabnik izbere akcijo »Nov«.
8. Sistem prikaže spustni seznam akcij.
9. Uporabnik iz seznama izbere akcijo »Zadeva«.
10. V sistemu se prikaže okno za izbiro tipa zadeve in delovnega toka.
11. Uporabnik iz seznama »Seznam tipov zadev« izbere ustrezen tip zadeve, ki jo želi kreirati.
12. Uporabnik iz seznama »Delovni tok« izbere delovni tok.
13. Uporabnik potrdi izbiro s klikom na gumb »Izberi«.
14. V sistemu se odpre obrazec za kreiranje in evidentiranje nove zadeve. Polja, ki so obvezna, obarva rumeno ter prikaže polja, ki se določijo avtomatsko in so predefinirana.
15. Obrazec zadeve je sestavljen iz orodne vrstice z akcijami, osnovnih podatkov zadeve in več zavihkov, ki jih navajamo spodaj:
    - splošno - splošni podatki zadeve,
    - dokumenti - seznam dokumentov, ki so v zadevi, (zavihek v tem koraku še neaktiven, aktiven postane po shranitvi podatkov),
    - lokacija - polja za vnos lokacije zadeve,
    - aktivnosti - aktivnosti na zadevi (zavihek v tem koraku še neaktiven, aktiven postane po shranitvi podatkov),
    - atributi - dodatni atributi na zadevi,
    - opomba - polja za dodajanje opomb,
    - dodatna avtorizacija - dodajanje dodatnih avtorskih pravic na zadevo (zavihek v tem koraku še neaktiven, aktiven postane po shranitvi podatkov).
16. Uporabnik vnese vse obvezne in druge podatke zadeve.
17. Uporabnik preveri vnesene podatke.

18. Uporabnik potrdi vnesene podatke z izbiro gumba »Uporabi«.
19. V sistemu se izvedejo naslednje aktivnosti:
    - dodeli številko zadeve,
    - dodeli »Datum evidentiranja zadeve«,
    - shrani podatke o zadevi,
    - zadevi dodeli začetno stanje,
    - prikaže zavihke »Dokumenti«, »Aktivnosti«, »Dodatna avtorizacija«,
    - obrazec predmeta ostane odprt.
20. Uporabnik po potrebi vnese podatke na zavihek »Dokumenti«, »Aktivnosti« in »Dodatna avtorizacija«.
21. Uporabnik potrdi vnesene podatke z izbiro gumba »Shrani«.
22. V sistemu se izvedejo naslednje aktivnosti:
    - shrani vnesene podatke o zadevi in
    - prikaže novo odprto zadevo v seznamu zadev in dokumentov v modulu »Zadeve« – primer uporabe je zaključen.

Rezultat uspešno izvedenega scenarija: zadeva je zavedena v modulu »Zadeve«.

3.3.2 Scenarij 2: Dodajanje dokumenta v zadevi

Koraki potrebni za dodajanje dokumenta v zadevi so:

1. Uporabnik želi evidentirati dokument v zadevo.
2. Uporabnik v meniju za izbiro modulov izbere »Zadeve«.
3. V sistemu se odpre osnovno okno modula »Zadeve«.
4. V meniju za izbiranje pogleda uporabnik izbere ustrezen pogled zadev.
5. Sistem odpre seznam pogledov.
6. Uporabnik izbere pogled v katerem želi evidentirati dokument.

7. Sistem odpre izbrani pogled s tabelo vnesenih zadev in dokumentov do katerih ima uporabnik pravice in orodno vrstico akcij, ki jih je mogoče izvajati.
8. Uporabnik izbere zadevo, v katero želi evidentirati dokument.
9. Uporabnik z dvojnim klikom na zapis zadeve odpre obrazec evidentirane zadeve.
10. Uporabnik izbere akcijo »Uredi«.
11. Sistem prikaže obrazec zadeve v urejevalnem načinu.
12. Uporabnik izbere zavihek »Dokumenti«.
13. Opomba: V zavihku se nahajajo še akcije »Dodaj«, »Uredi« in »Preglej«.
14. Uporabnik izbere akcijo »Dodaj«.
15. Sistem prikaže seznam tipov dokumentov, ki jih lahko kreira.
16. Opomba: seznam je odvisen od konfiguracije.
17. Uporabnik iz seznama dokumentov izbere tip dokumenta.
18. Uporabnik potrdi izbiro s klikom na gumb »Izberi«.
19. Sistem odpre obrazec za kreiranje novega dokumenta. Polja, ki so obvezna, obarva rumeno ter prikaže polja, ki se določijo avtomatsko in so predefinirana.
20. Obrazec za evidentiranje dokumenta je sestavljen iz orodne vrstice z akcijami, osnovnih podatkov dokumenta in več zavihkov:
    - splošno - splošni podatki dokumenta,
    - priloge - dodajanje priponk iz lokalnega diska, skeniranje priponk, dodajanje predlog (v tem koraku še neaktiven, aktiven postane po shranitvi podatkov),
    - lokacija - polja za vnos lokacije dokumenta,
    - aktivnosti - dodajanje aktivnosti (v tem koraku še neaktiven; aktiven postane po shranitvi podatkov),
    - atributi - dodatno definirani atributi,
    - opomba - polja za dodajanje opomb oz. zaznamkov,

- dodatna avtorizacija - dodajanje dodatnih avtorskih pravic na dokument (v tem koraku še neaktiven, aktiven postane po shranitvi podatkov).

21. Uporabnik vnese vse obvezne in druge podatke dokumenta.
22. Uporabnik preveri vnesene podatke.
23. Uporabnik potrdi vnesene podatke z izbiro gumba »Uporabi«.
24. Sistem:
    - dodeli številko dokumenta,
    - določi »Datum evidentiranja dokumenta«,
    - shrani podatke o dokumentu,
    - dokumentu dodeli začetno stanje,
    - prikaže zavihke »Priloge«, »Aktivnosti« in »Dodatna avtorizacija«,
    - obrazec dokumenta ostane odprt.
25. Uporabnik po potrebi vnese podatke na zavihek »Priloge«, »Aktivnosti« in »Dodatna avtorizacija«.
26. Uporabnik potrdi vnesene podatke z izbiro gumba »Shrani«.
27. Sistem:
    - shrani vnesene podatke o dokumentu in
    - prikaže nov evidentiran dokument v seznamu zadev in dokumentov v modulu Zadeve – primer uporabe je zaključen.

Rezultat uspešno izvedenega scenarija: dokument je zaveden v modulu »Zadeve«.

Definirana scenarija bosta izvedena iz treh različnih uporabniških profilov (arhivar, referent, glavna pisarna). To je potrebno, ker imajo različni uporabniški profili različne pravice, kar povzroča različno delovanje aplikacije in posledično različne čase izvajanja.

3.4 Konfiguracija testnega okolja

Priprava testnega okolja, orodij in virov, namenjenih za testiranje aplikacije še preden je aplikacija na voljo, lahko znatno poveča količino izvedenih testov v času, ko bo aplikacija na voljo.

Skupina, ki izvaja teste, praviloma ne konfigurira testnega okolja oz. to naredi skupaj s skupino, ki bo nameščala aplikacijo na razvojno in produkcijsko okolje.

V našem primeru so predstavniki oddelka za interno informatiko pripravili izolirano omrežje, namestili operacijske sisteme ter zagotovili območje IP naslovov, potrebnih za testiranje. Namestitev aplikacije na produkcijsko in razvojno okolje ter specifične aplikacije, potrebne za generiranje bremena in spremljanje rezultatov, kot so Visual Studio in Fiddler, smo namestili in konfigurirali sami.

3.5 Implementacija načrtovanih testov

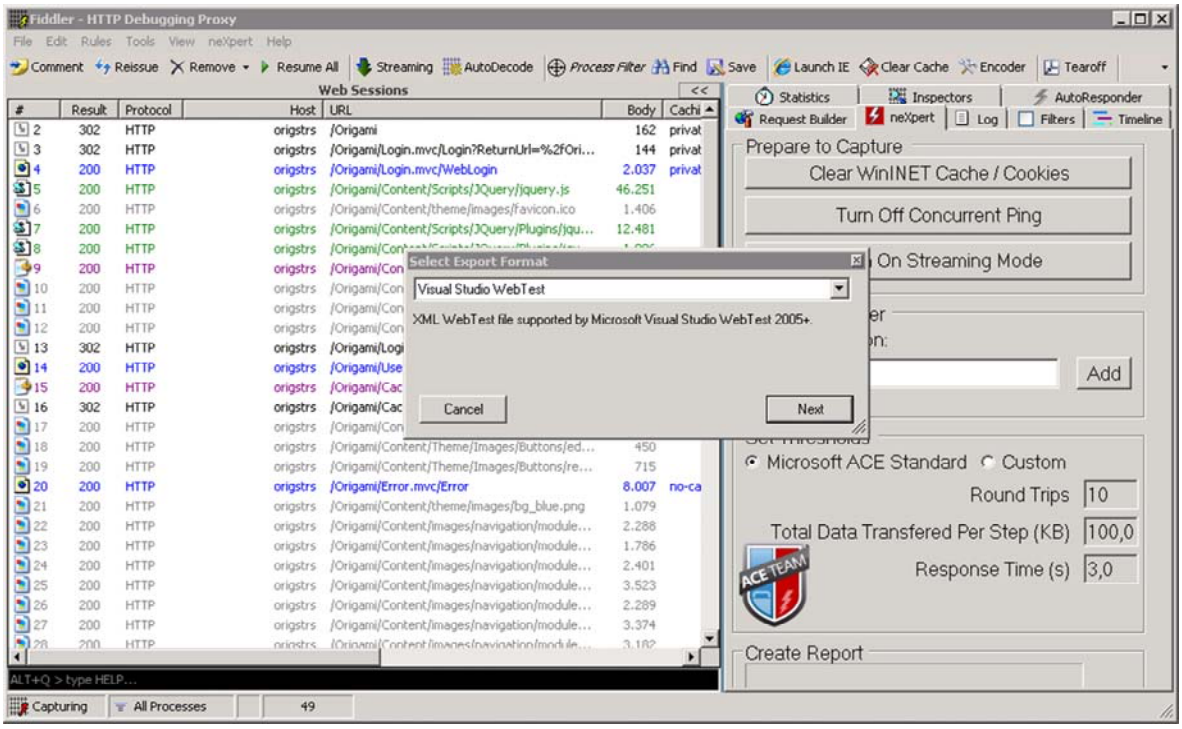
V tem poglavju bo predstavljena pretvorba že definiranih ključnih scenarijev v obliko, primerno za orodje, s katerim generiramo testne obremenitve - v našem primeru je to orodje Visual Studio 2010. V tem koraku poskrbimo tudi za pravilno konfiguracijo testnega orodja ter definiranje pravil za validacijo podatkov, poslanih s strani aplikacije.

3.5.1 Kreiranje testov s pomočjo orodja Fiddler

Čeprav Visual Studio že vsebuje orodje za ustvarjanje testov, je njegova uporaba vprašljiva pri snemanju kompleksnih testov - predvsem zaradi določenih pomanjkljivosti, kot so npr. neevidentiranje AJAX klicev ter neevidentiranje klicev na novo odprtih (angl. *pop-up*) oknih, ki se pojavijo med uporabniško interakcijo z aplikacijo. V takih primerih nam pomaga večji nabor funkcionalnosti orodja Fiddler z neXpert dodatkom, ki zajame vse potrebne podatke, ter poskrbi za njihovo shranjevanje in izvoz v obliko spletnega testa.

Za kreiranje testa je potrebno najprej zagnati Fiddler in poskrbeti, da se le-ta nahaja v načinu delovanja za zajem podatkov. V poljubnem spletnem brskalniku odpremo našo aplikacijo in

izvedemo določen scenarij, ki posnema končno uporabo. Ko pridemo do konca scenarija, program Fiddler ustavimo, posneto sejo shranimo in jo izvozimo kot spletni test (glej sliko 9).



Slika 9: Prikaz posnete seje in okna za izvoz testa v Fiddler-ju.

Med snemanjem testa program Fiddler beleži čas razmišljanja uporabnika[1] in tega upošteva pri izvozu testa. Realistična simulacija časa razmišljanja uporabnika je ključnega pomena za natančnost testa, zato smo pri snemanju testov uporabili povprečni čas razmišljanja končnega uporabnika, ki smo ga pridobili na podlagi analiz uporabe podobne aplikacije razvite v našem podjetju.

3.5.2 Uvoz in konfiguracija spletnih testov v okolju Visual Studio

Ko so testi pripravljeni, jih je potrebno uvoziti v Visual Studio. Po uvozu je potrebno nastaviti številne parametre izvajanja testov, ki so ključnega pomena za pridobitev realnih podatkov. V nadaljevanju bom opisal pomen in konfiguracijo najpomembnejših parametrov.

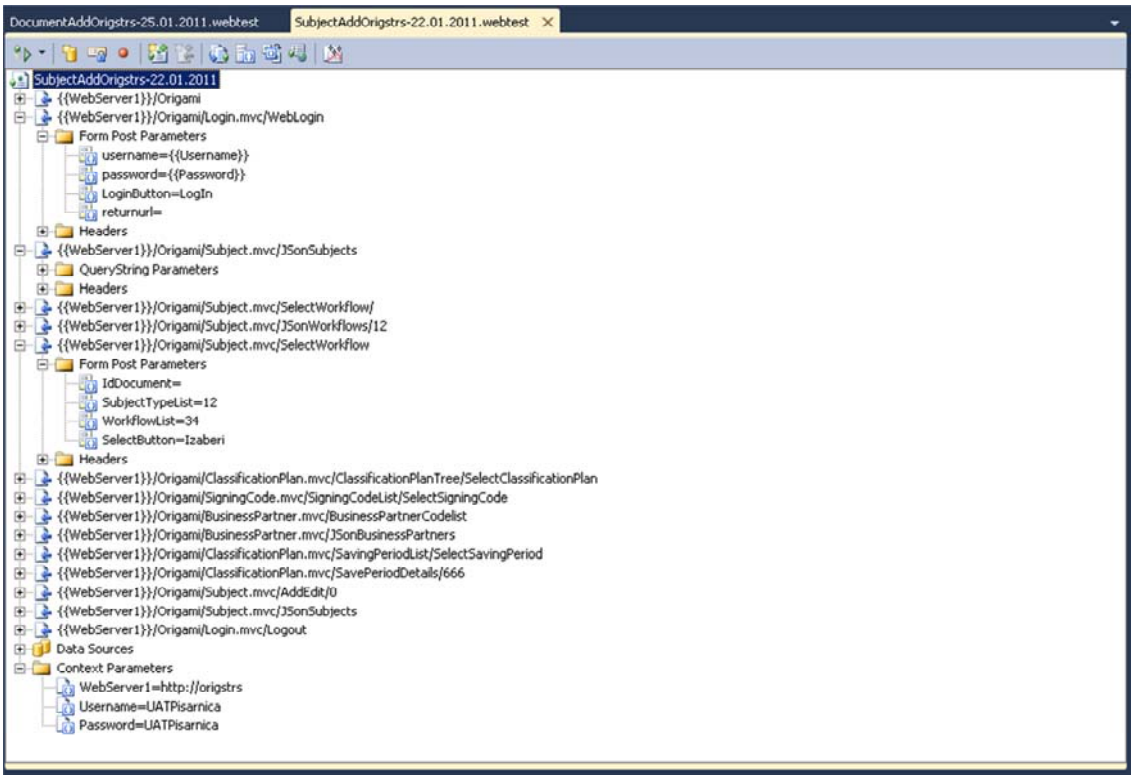
[1] Čas razmišljanja uporabnika (angl. *User think time*) predstavlja čas, ki ga uporabnik porabi preden izvede naslednjo akcijo.

3.5.2.1 Uvoz in prilagoditev testov

Po uvozu testov v Visual Studio (glej sliko 10) jih je potrebno obdelati in jim določiti spremenljivke, ki nam omogočajo dinamično generiranje parametrov, ki nastopajo v posameznem testu. Poleg tega, je potrebno zagotoviti pravila za validacijo odgovorov s strani strežnika, preko katerih določimo ali je bil test uspešno izveden [2].

Spremenljivke lahko določamo iz raznih podatkovnih skladišč (datoteka, podatkovna baza itd.), kar nam omogoča boljšo porazdelitev bremena na vse segmente aplikacije in ne le na omejen nabor podatkov, ki smo jih posneli v spletnem testu. Pogost primer je prijava v aplikacijo ali nabor zadeve oz. dokumenta, ki ga med testom spreminjamo.

Pravila za validacijo določamo z namenom ugotavljanja ali je strežnik odgovoril na predviden način. V kolikor odgovor ni v skladu s pravili (oz. pričakovanji), zabeležimo napako. Določimo lahko tudi največje število napak, pri katerem testiranje ustavimo. Tako lahko izvajanje testov v primeru nepravilnega delovanja aplikacije prekinemo.

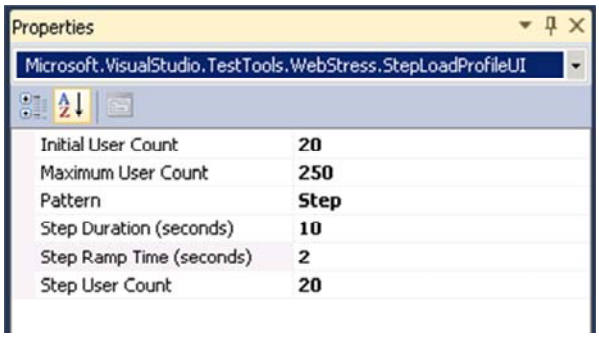


Slika 10: Prikaz uvoženega testa v Visual Studio.

3.5.2.2 Nastavitve bremena

Pri nastavitvah obremenitve aplikacije je najpomembnejša odločitev, kako nastaviti vzorec za generiranje bremena (število trenutnih uporabnikov).

V naših testih smo se odločili za stopničasto generiranje bremena (angl. *step pattern*). Stopničasti vzorec nam omogoča začeti test z začetnim številom uporabnikov, katerega pozneje na določen interval povečujemo za določeno število uporabnikov. Stopničasto generiranje bremena je eden najpogosteje uporabljenih vzorcev. Njegovo uporabo se priporoča tudi ko želimo testirati konstantno maksimalno število uporabnikov. Strežnika ne nasičimo z zahtevki nenadoma oz. naenkrat, temveč se le-ti postopoma povečujejo - s tem se izognemo nekonsistentnosti začetnih rezultatov testiranja [1].



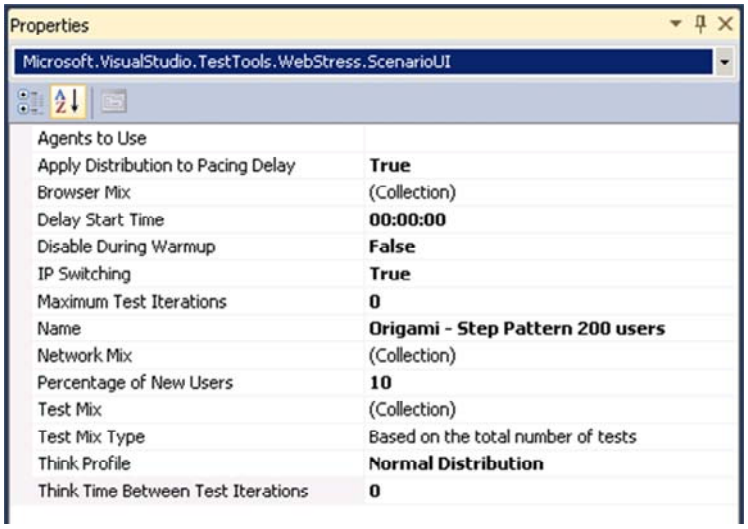
Slika 11: Prikaz okna za nastavitve bremena.

Parametri, ki jih je potrebno definirati pri stopničastem vzorcu so naslednji (glej sliko 11):

- *Initial User Count* – število uporabnikov na začetku testa,
- *Maximum User Count* – število uporabnikov pri katerem se število le-teh ne povečuje več,
- *Step Duration* – časovni interval med posameznimi povečanji števila uporabnikov,
- *Ramp Time* – časovni interval, ki določa koliko časa bo potrebnega za povečanje števila uporabnikov do naslednje iteracije,
- *User Count* – število uporabnikov, ki jih dodamo med vsako iteracijo.

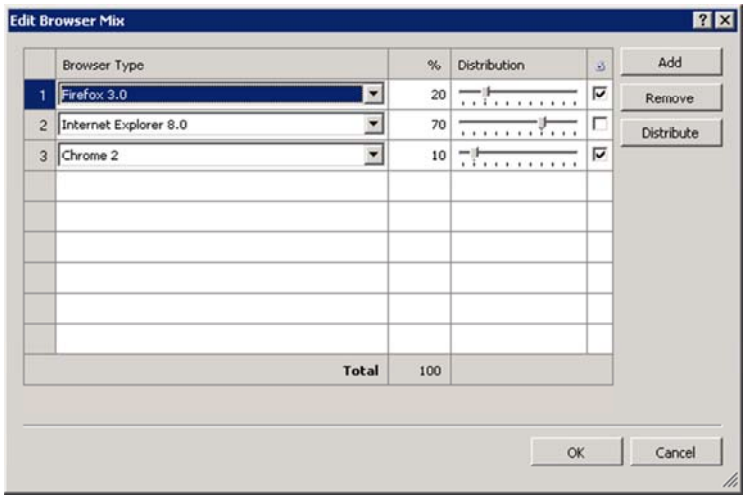
3.5.2.3 Nastavitve izvajanja testov

Poleg nastavitev bremena moramo za vsak test določiti še dodatne parametre, ki podrobneje opisujejo izvajanje izbranega vzorca (glej sliko 12):



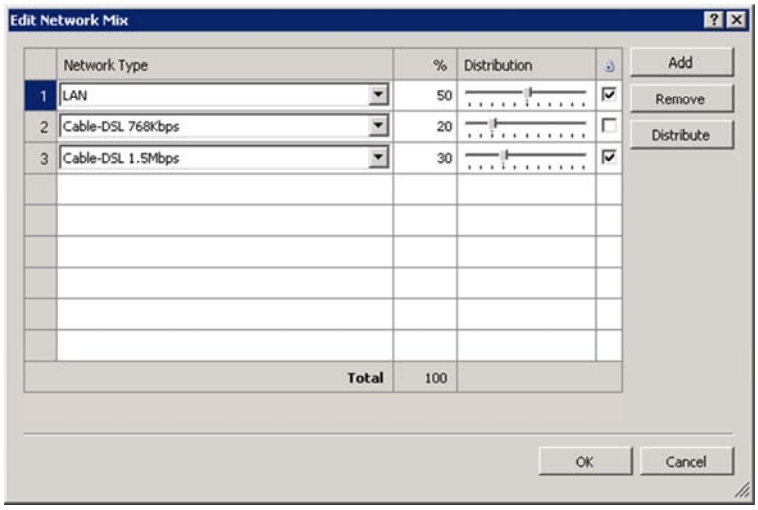
Slika 12: Prikaz okna za nastavitve izvajanja testov.

- *Browser Mix* – Parameter določa porazdelitev uporabnikov glede na uporabljen brskalnik. V kolikor se bo do naše aplikacije dostopalo iz različnih brskalnikov, lahko preko tega parametra testiramo obnašanje aplikacije pri različnih odjemalcih. Glede na dosedanje poznavanje končnih uporabnikov smo določili porazdelitev, ki je prikazana na sliki 13.



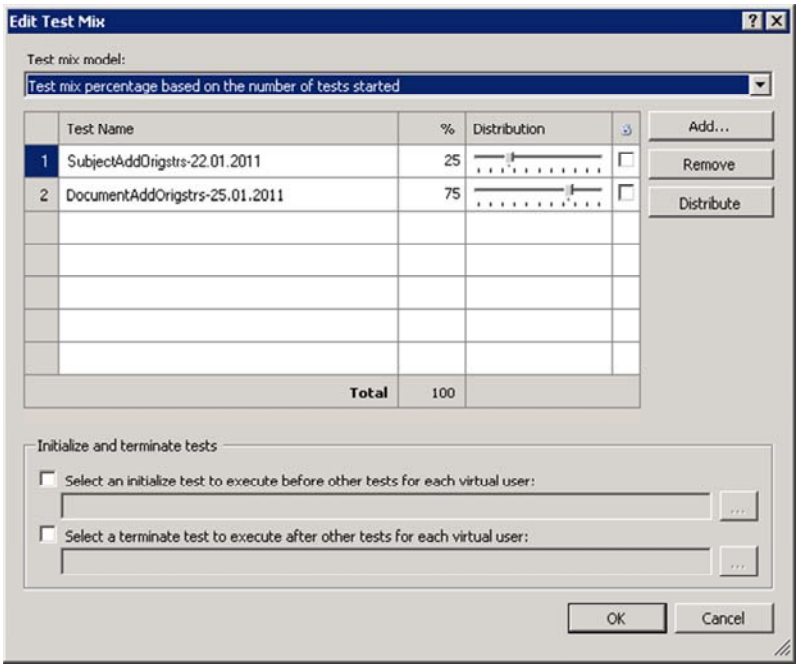
Slika 13: Prikaz okna za nastavitve brskalnikov.

- *IP Switching* – V testu omogoča preklapljanje med IP naslovi s čimer simuliramo klice iz več delovnih postaj. S tem preprečimo shranjevanje podatkov v predpomnilnik.
- *Network Mix* – glede na to, da ne bodo vsi končni uporabniki uporabljali lokalnega omrežja za dostop do naše aplikacije, smo preko te nastavitve določili hitrost in porazdelitev uporabnikov po različnih navideznih omrežij. Porazdelitev in hitrost posameznega navideznega omrežja je prikazana na sliki 14.



Slika 14: Prikaz okna za mrežne nastavitve.

- *Percentage of New Users* – Določa razmerje med številom uporabnikov, ki uporabljajo predpomnilnik, in številom uporabnikov s praznim predpomnilnikov v danem časovnem intervalu. Pri spletnih testih namreč lahko pride do velikih razlik v odzivnih časih, če uporabnik prvič obišče aplikacijo (ima prazen predpomnilnik).
- *Test Mix* – Določa verjetnost, s katero bo uporabnik izbral posamezni test (glej sliko 15). Porazdelitev smo določili na podlagi analize uporabe podobnih aplikacij. V našem primeru smo prišli do ocene, da v povprečju na vsako ustvarjeno zadevo dodamo tri dokumente.



Slika 15: Prikaz okna za nastavitve razmerja testov.

- *Think Profile* – Z uporabno normalne porazdelitve časa razmišljanja uporabnika lahko izvajanje testov približamo realnemu scenariju do mere, ki nam omogoča pridobiti realistične podatke. Nerealno je pričakovati in simulirati, da vsak uporabnik aplikacije enako hitro generira zahteve. Uporabniki se razlikujejo po spretnosti uporabe ter poznavanju aplikacije, zato so njihovi odzivni časi zelo različni.

3.5.2.4 Kratek povzetek

Vedno, ko program Visual Studio začne simulirati delovanje novega uporabnika, mu na podlagi verjetnosti dodeli spletni test, izbere brskalnik in navidezno omrežje s katerega generira zahtevke, določi stanje predpomnilnika, ter čas razmišljanja za posamezno akcijo znotraj testa.

Izbira parametrov, potrebnih za izvedbo testov, je zelo pomembna - z napačno izbiro lahko pridobimo nerealne in zavajajoče podatke.

3.6 Izvedba testov

Izvedba testov je aktivnost, ki sledi po implementaciji načrtovanih testov. Mnogi obravnavajo to aktivnost kot »pritisk na gumb«, s katerim sprožimo začetek testiranja, ter spremljanje rezultatov za potrditev, da testiranje teče v skladu z našimi pričakovanji. Vendar gre pri izvedbi testov za bistveno bolj zapletene aktivnosti kot »pritisk na gumb«. Koraki, ki so vključeni v izvajanje testov, so naslednji:

- validacija testnega okolja,
- validacija testov,
- izvajanje testov,
- arhiviranje testov.

3.6.1 Validacija testnega okolja

Stremimo k enakosti testnega in produkcijskega okolja oziroma da testno okolje čim bolj približamo produkcijskemu okolju. Značilno je, da so vse razlike med testnim in produkcijskim okoljem zabeležene in jih upoštevamo med načrtovanjem in izvajanjem testov. Pred izvedbo testov je ključna potrditev, da se testno okolje ujema s konfiguracijo, ki smo si jo zamislili oz. za katero smo teste načrtovali. Če se je testno okolje med implementacijo in izvajanjem testov iz katerega koli razloga spremenilo, obstaja velika verjetnost, da testi, ki smo jih že posneli, ne bodo delovali. V skrajnem primeru se testi lahko izvedejo brez napake, vendar kot rezultat dobimo neresnične in zavajajoče podatke.

Za testiranje zmogljivosti NETSERVER postaje, namenjene za generiranje bremena, smo izklopili čas razmišljanja uporabnika ter pognali enostavne teste, da potrdimo, da je postaja zmožna generirati bremena, ki 100 odstotno zasede resurse ORIGSTRS strežnika. Nezmožnost 100 odstotne porabe virov testnega strežnika lahko kaže na težave v omrežju ali na zapolnitev izhodnih kapacitet delovne postaje, ki generira breme.

Ob pregledu dnevniške datoteke na strežniku, ki je generiral breme (NETSERVER) razberemo, da so zahtevki bili generirani z uporabo različnih IP naslovov. S tem smo potrdili pravilno delovanje »IP Switching-a«, definiranega v prejšnjem odstavku.

3.6.2 Validacija testov

Da bi razumeli podatke, zbrane od izvedbe testov, morajo obremenitvene simulacije čim bolj natančno odražati scenarije načrtovanih testov. Ko simulacija ne odraža načrtovanega scenarija, so rezultati nagnjeni k napačni razlagi. Tudi če naši testi pravilno odražajo načrtovane scenarije, še vedno obstaja veliko načinov, da dobimo iz testa neveljavne oz. zavajajoče rezultate. Zelo pomembno je, da dobljene rezultate preverimo, saj se moramo zavedati kako pomembni so pri sprejemanju odločitev, povezanih z aktivnostjo aplikacije na produkcijskem okolju.

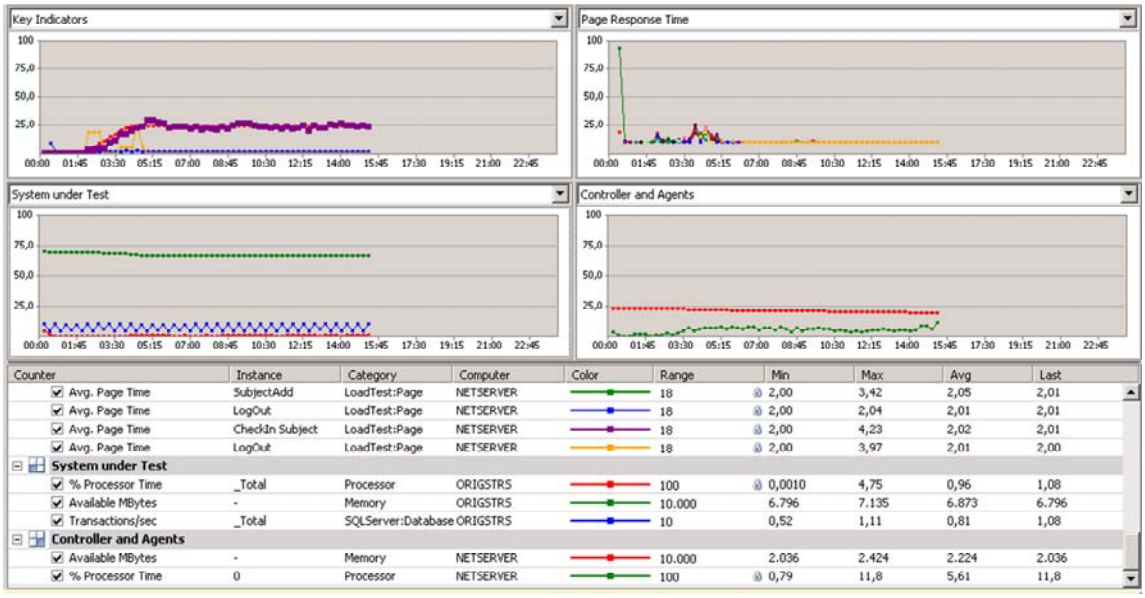
Za zagotovitev pravilnosti delovanja testov smo najprej vsak test pognali s statičnimi podatki ter spremljali odziv sistema. Za vsak korak znotraj enega testa smo definirali validacijska pravila, ki v primeru napačnega delovanja aplikacije sprožijo alarm, ter glede na stopnjo napake po potrebi ustavijo testiranja.

Z vpeljavo dinamičnih parametrov smo dosegli zaželeno realistično porazdelitev zadev in dokumentov znotraj sistema ter omogočili izvajanje testov s strani različnih uporabniških profilov. Pravilnost delovanja dinamičnih parametrov smo potrdili z uporabo že obstoječih validacijskih pravil.

Zaradi preurejanja (angl. *refactoring*) kode ter optimizacije aplikacije v času razvoja, se je klic določenih korakov znotraj scenarija spremenil, zato je bilo potrebno teste večkrat posneti ter validirati.

3.6.3 Izvajanje testov

Po opravljeni validaciji testov in testnega okolja, smo teste izvajali s skrajšanim časom trajanja ter s sprotnim preverjanjem testnega okolja (glej sliko 16). Po vsakem testiranju smo pripravili kratek povzetek dogodkov med testiranjem. Zabeležili smo pripombe ter jih posredovali razvoju. Pripombe so se običajno nanašale na odpoved strojne opreme, aplikacijske izjeme in napake, težave z omrežjem ali premalo prostora na disku. Rezultate vsakega testa smo dokumentirali ter shranili v podatkovno skladišče, da bi jih kasneje uporabili kot referenco pri nadaljnjem testiranju nove različice aplikacije.



Slika 16: Prikaz izvajanja obremenitvenega testa v Visual Studiu.

3.6.4 Arhiviranje testov

Pri izvedbi testov je zelo priporočljiva uporaba sledenja zgodovine (angl. *source control*). V večini primerov se testi prilagodijo novi različici aplikacije. V kolikor je nova različica neustrezna ali je potrebna ponovna izvedba testiranja stare različice aplikacije, potrebujemo rezultate testov le-te.

Naša skupina je v ta namen uporabljala TFS strežnik, ki se na enostaven način integrira z našim razvojnim okoljem. S tem smo prihranili veliko časa na morebitnih napakah, ki so nastale pri prilagoditvah testov za nove različice aplikacije. Dodatna prednost uporabe TFS strežnika je bila tudi ta, da so vsi člani skupine imeli vedno na voljo zadnjo različico testov.

4 Prikaz in analiza rezultatov

V tem poglavju so opisani rezultati testiranja, ki smo jih pridobili pri izvajanju osem urnega obremenitvenega testa na aplikaciji Origami DMS z 250 uporabniki. Testi so bili izvedeni na različici aplikacije 1.7.20110215. V nadaljevanju so prikazani rezultati odzivnih časov ključnih funkcionalnosti DMS sistema, prepustnost, uporaba virov, ter prikaz vpliva »interceptorja« na zmogljivosti sistema.

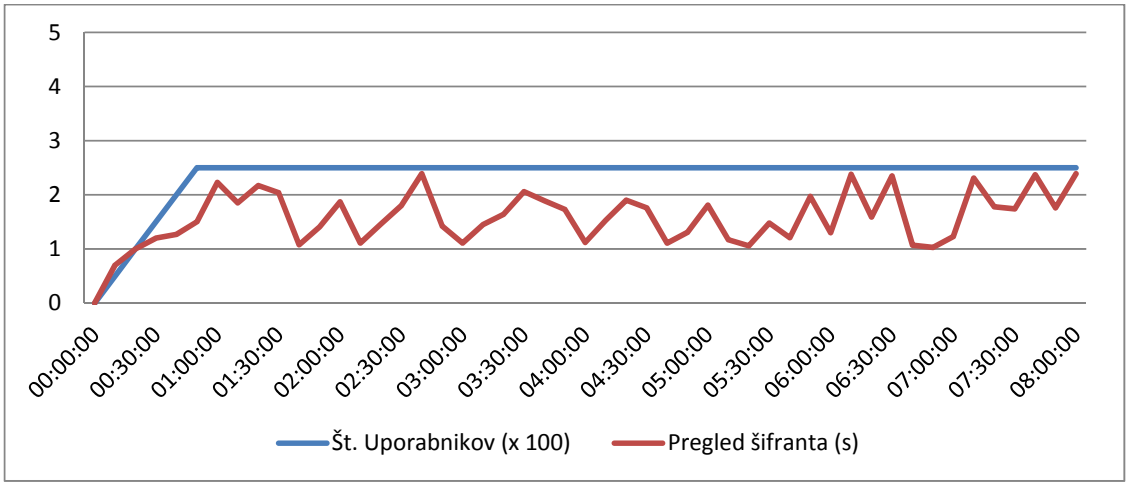
4.1 Odzivni časi

Odzivni časi, ki smo jih izmerili, predstavljajo čase, ki pretečejo od zahtevka uporabnika po določeni operaciji, do prikaza zahtevanih podatkov oz. izvajanja operacije in prikaza odgovora. Večina operacij je sestavljenih iz več sinhronih in asinhronih (AJAX) klicev. Na grafih v tem poglavju so prikazani skupni časi, potrebni za izvajanje ter prikaz rezultatov določene operacije. Povprečni odzivni časi so bili izračunani na intervalu, ko je bilo v sistemu prijavljeno maksimalno število uporabnikov (250).

Pregled šifranta je operacija, ki se izvede vsakokrat, ko se uporabnik želi sklicevati oz. pogledati vrednost enega izmed šifrantov definiranih v aplikaciji. Šifranti v aplikaciji Origami DMS so:

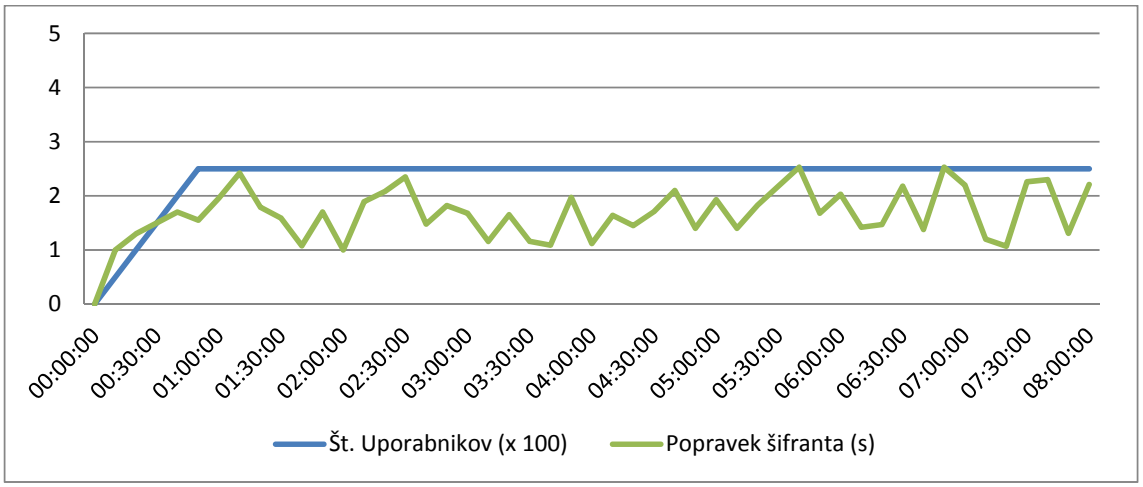
- šifrant držav,
- šifrant strank,
- šifrant delovnih tokov,
- šifrant življenjskih ciklov,
- šifrant uporabnikov,
- šifrant atributov.

Maksimalni izmerjeni odzivni čas operacije je trajal 2,39 sekunde (glej sliko 17). Povprečni čas je bil 1,66 sekunde, kar predstavlja 80% hitrejši odzivni čas od želenega (3s).



Slika 17: Graf odzivnih časov operacije »Pregled šifranta«.

Popravek šifranta je akcija, ki jo uporabnik izvede ob spremembi vrednosti določenega šifranta aplikacije (našteti zgoraj).

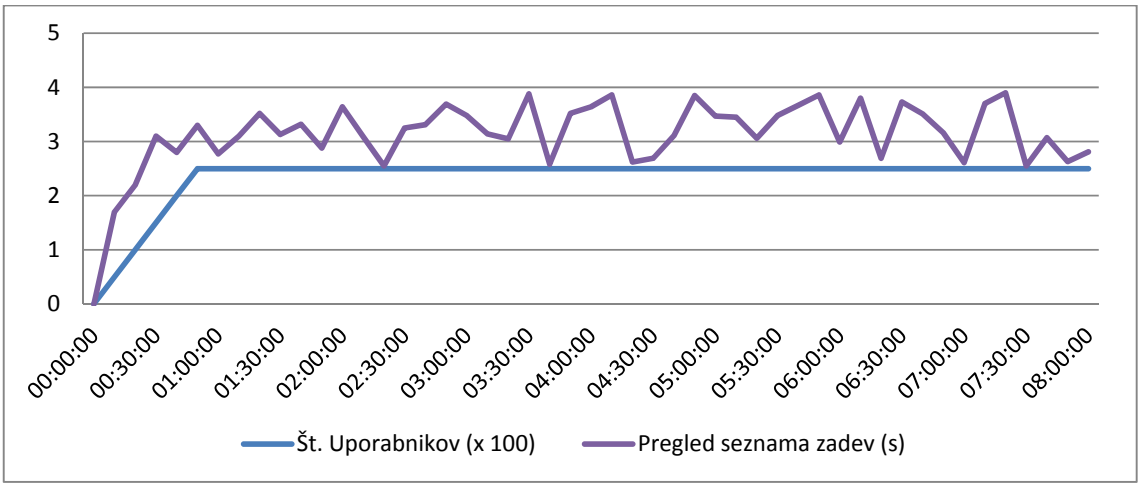


Slika 18: Graf odzivnih časov operacije »Popravek šifranta«.

Maksimalni izmerjeni odzivni čas operacije je trajal 2,53 sekunde (glej sliko 18), povprečni čas pa je bil 1,73 sekunde, kar predstavlja 73% hitrejši odzivni čas od želenega (3s).

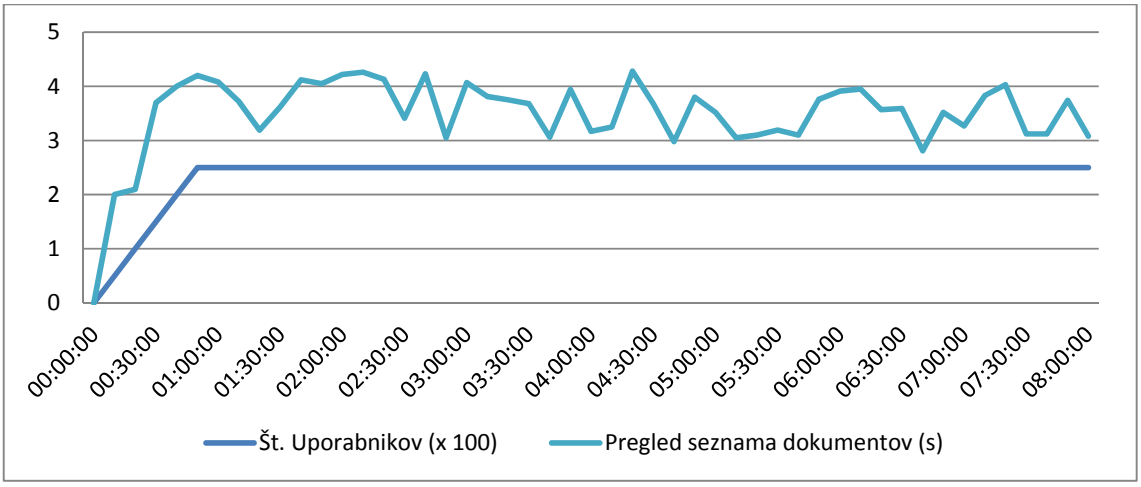
Pregled seznama zadev in **Pregled seznama dokumentov** so ene izmed osnovnih operacij, ki se izvedejo skoraj vsakokrat, ko se uporabnik prijavi v sistem. Te operacije omogočajo prijavljenemu uporabniku pregled vseh zadev/dokumentov, ki so mu dodeljene oz. ima na njih avtorske/bralske pravice.

Maksimalni izmerjeni odzivni čas operacije »Pregled seznama zadev« je trajal 3,9 sekunde (glej sliko 19), povprečni čas pa je bil 3,25 sekunde, kar predstavlja 53% hitrejši odzivni čas od želenega (5s).



Slika 19: Graf odzivnih časov operacije »Pregled seznama zadev«.

Za operacijo »Pregled seznama dokumentov« smo izmerili maksimalni odzivni čas 4,28 sekunde (glej sliko 20). Povprečni čas je bil 3,61 sekunde, kar predstavlja 38% hitrejši odzivni čas od želenega (5s).

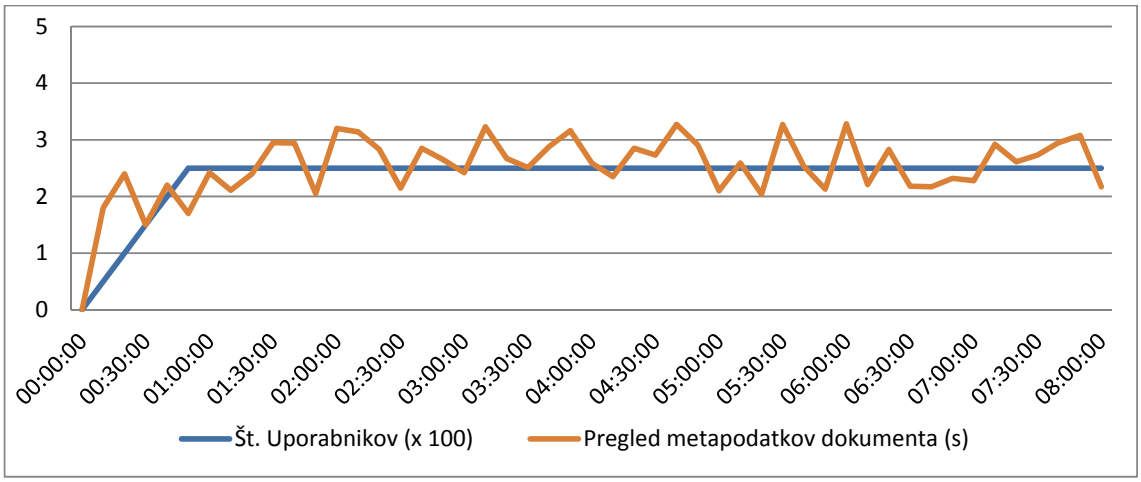


Slika 20: Graf odzivnih časov operacije »Pregled seznama dokumentov«.

Naslednji dve sliki (21,22) prikazujeta odzivne čase operacij »Pregled metapodatkov dokumenta« in »Popravek metapodatkov dokumenta«.

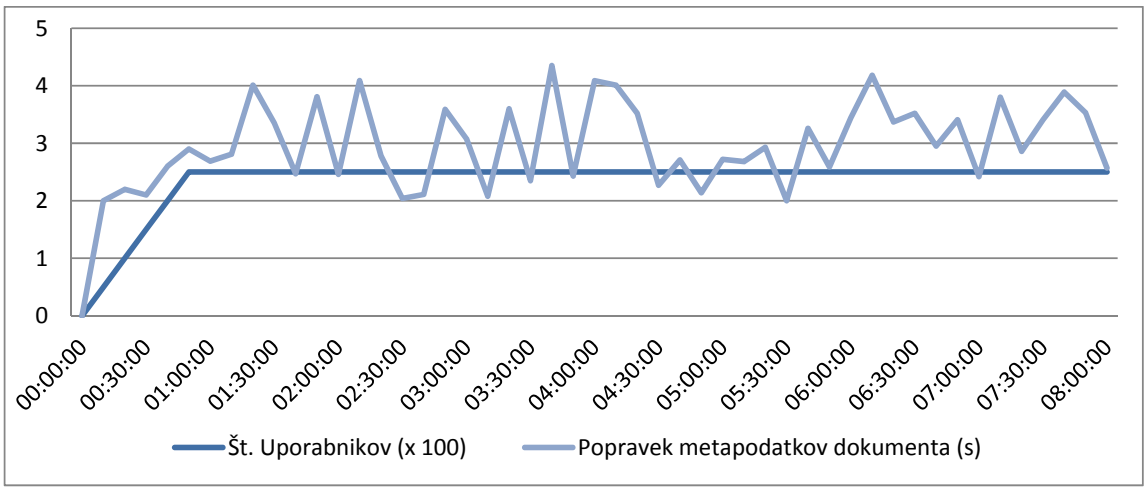
Metapodatki so opredeljeni kot podatki o podatkih. V našem primeru so podatki dokumenti, podatki o podatkih pa podrobnejše informacije o teh dokumentih (npr. tip dokumenta, stanje v življenjskem ciklu, rok hrambe itd.).

Maksimalni izmerjeni odzivni čas operacije je trajal 3,28 sekunde, povprečni čas pa le 2,62 sekunde, kar predstavlja 52% hitrejši odzivni čas od želenega (4s).



Slika 21: Graf odzivnih časov operacije »Pregled metapodatkov dokumenta«.

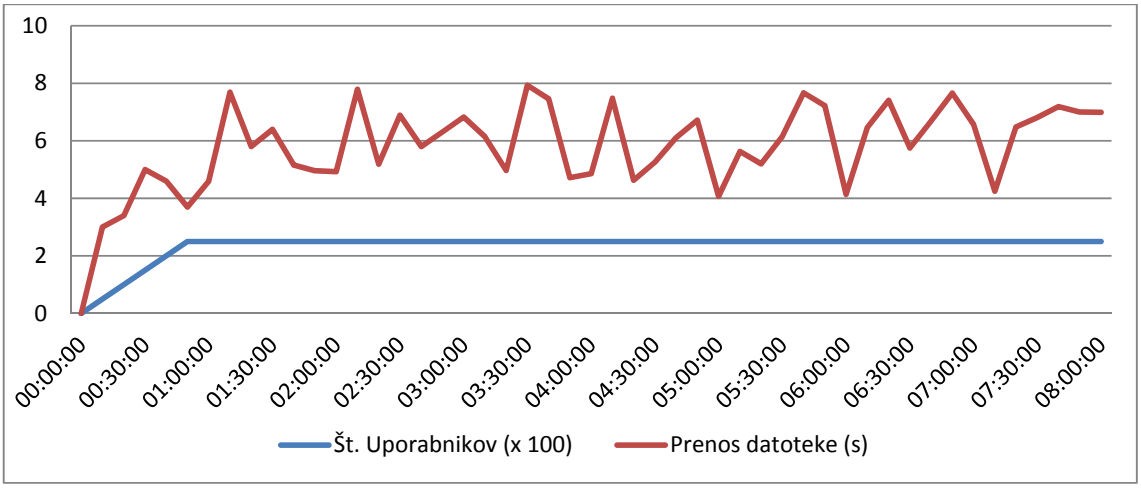
Za operacijo »Popravek metapodatkov dokumenta« smo izmerili maksimalni odzivni čas 4,35 sekunde. Povprečni čas je bil 3,07 sekunde, kar predstavlja 63% hitrejši odzivni čas od želenega (5s).



Slika 22: Graf odzivnih časov operacije »Popravek metapodatkov dokumenta«.

Za testiranje operacije »Prenos datoteke« smo izdelali namenskega odjemalca, ki je na strežnik pošiljal datoteke med 500 KB in 10 MB velikosti. Odjemalca smo izdelali, da bi poleg uporabnikov simulirali tudi integracije z drugimi sistemi (npr. integracijo z OCR strežnikom).

Maksimalni izmerjeni odzivni čas operacije je trajal 7,93 sekunde (glej sliko 23). Povprečni čas je bil 6,08 sekunde, kar predstavlja 64% hitrejši odzivni čas od želenega (10s).

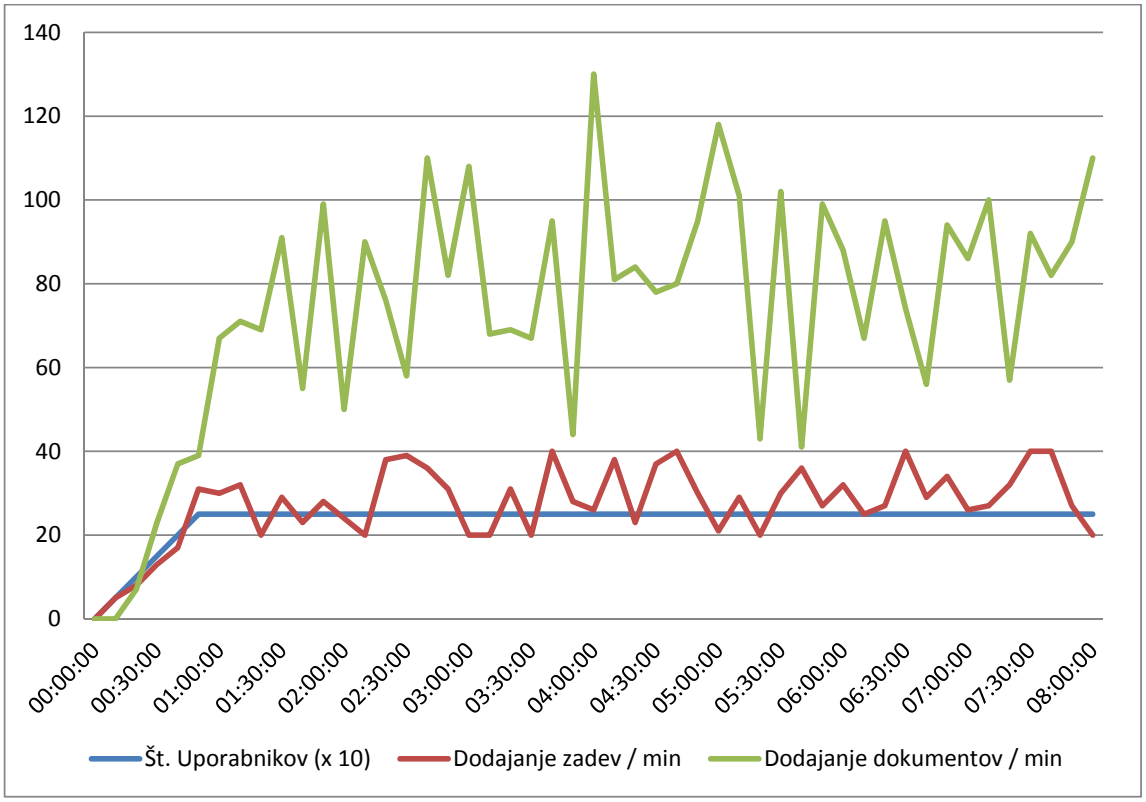


Slika 23: Graf odzivnih časov operacije »Prenos datoteke«.

4.2 Prepustnost sistema

Prepustnost sistema (angl. *system throughput*), predstavlja število zaključenih operacij na časovno enoto. V našem primeru operacije predstavljajo dodajanje dokumentov in dodajanje zadev. Glede na tip in trajanje naše operacije ter za lažje razumevanje rezultatov, smo za časovno enoto izbrali minuto.

Prepustnost smo izmerili tako, da smo po zaključenih testih izvedli poizvedbo SQL, ki je kot rezultat vrnila število novo ustvarjenih zadev in dokumentov, ter njihov čas vstavljanja v sistem. Slika 24 prikazuje pridobljene rezultate agregirane po minutah.



Slika 24: Graf prepustnosti sistema.

V sistemu je bilo v časovnem intervalu ene minute ustvarjeno maksimalno 40 zadev ter 130 dokumentov. Povprečno dodajanje zadeve je znašalo 27 zadev/minuto, dokumenti pa so bili dodajani v sistem s povprečno hitrostjo 82 dokumentov/minuto.

Po zaključenih testih je bilo ustvarjenih 13.178 zadev ter 39.698 dokumentov. Razmerje med dokumenti in zadevami je 3,012 : 1, kar potrjuje pravilno razporeditev izvajanje testov definiranih v prejšnjem poglavju (25% dodajanje zadeve in 75% dodajanje dokumentov).

Kot rezultat smo pridobili tudi informacijo, da pri omenjeni prepustnosti viri ciljnega sistema niso presegli 50% obremenjenosti, kar je razvidno iz grafa v naslednjem poglavju.

4.3 Uporaba virov

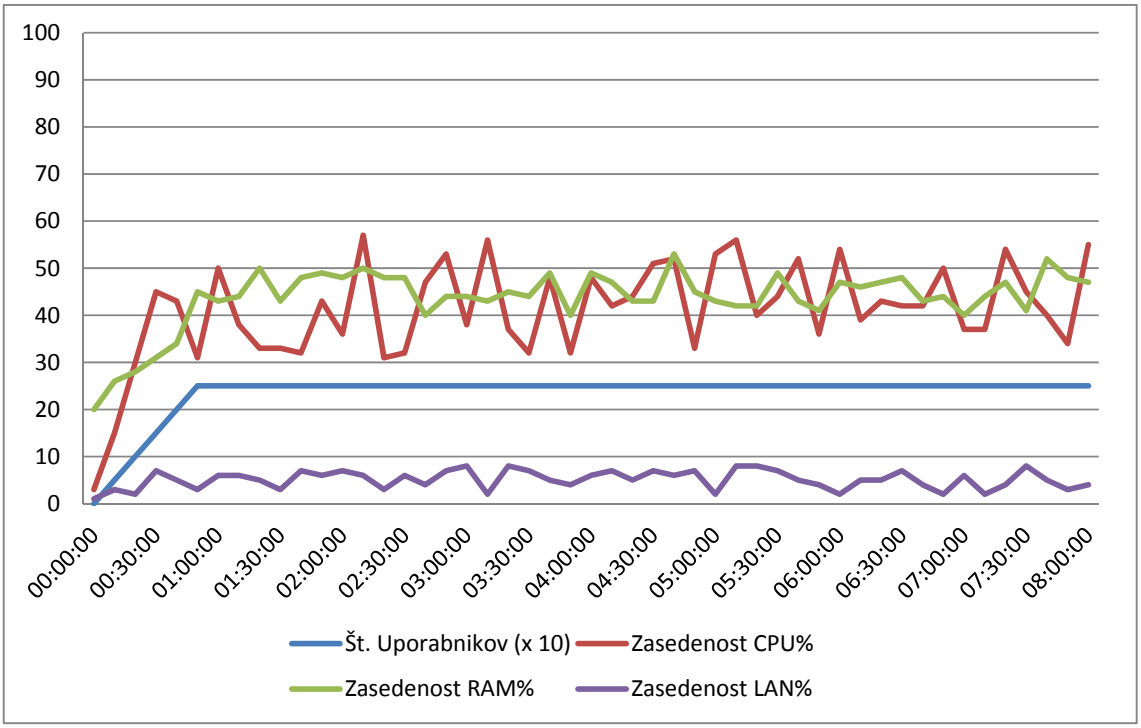
Med izvajanjem testov smo beležili zasedenost (obremenitev) virov strežnikov ORIGSTRS in NETSERVER.

V tabeli 4 in sliki 25 je prikazana poraba virov na našem ciljnem sistemu ORIGSTRS med izvajanjem testov.

	Povprečna zasedenost	*Maksimalna zasedenost*
Procesor	41,97%	57%
Pomnilnik	44,12% (3530 MB)	53% (4240 MB)
Omrežje	5,18% (51,87 Mbit/s)	8% (80 Mbit/s)

Tabela 4: Prikaz porabe virov na sistemu ORIGSTRS.

Kot je razvidno iz tabele, maksimalna zasedenost ne odstopa bistveno od povprečne. Na podlagi tega lahko sklepamo o stabilnem delovanju sistema brez večjih skokov v obremenitvi, kar pripomore k stabilnejšemu delovanju sistemskih virov.



Slika 25: Uporaba virov strežnika ORIGSTRS.

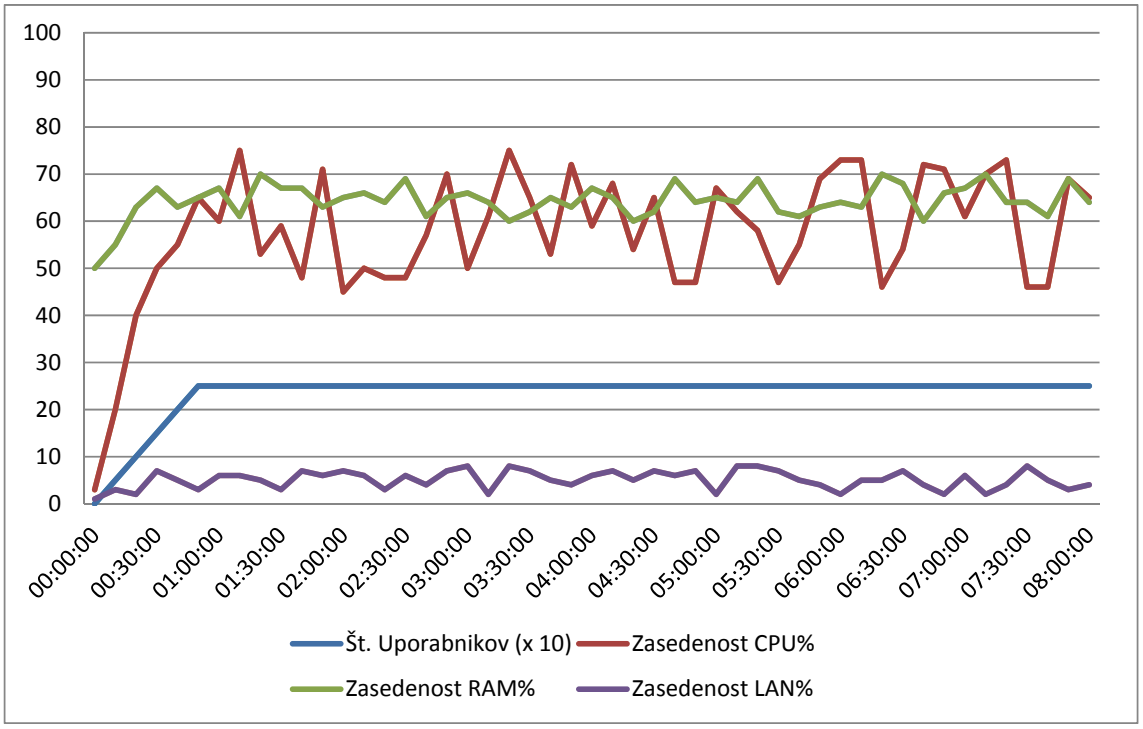
Konstantna poraba pomnilnika kaže tudi na to, da v sistemu nimamo večjih puščanj pomnilnika (angl. *memory leaks*).

Da bi zagotovili pravilnost rezultatov in preverili ali je prišlo do nasičenja na sistemu NETSERVER, ki je generiral breme, smo morali opazovati njegovo delovanje.

	Povprečna zasedenost	*Maksimalna zasedenost*
Procesor	58,47%	75%
Pomnilnik	64,56% (2582 MB)	70% (2800 MB)
Omrežje	6,041% (60,41 Mbit/s)	9% (90 Mbit/s)

Tabela 5: Prikaz porabe virov na sistemu NETSERVER.

Sistemski viri niso bili polno zasedeni (glej tabelo 5), kar nam zagotavlja pravilnost izmerjenih rezultatov na ORIGSTRS, ter nam potrjuje ustreznost izbire NETSERVER strežnika za generiranje bremena.



Slika 26: Uporaba virov strežnika NETSERVER.

Kot je razvidno iz slike 26, večjih odstopanj pri porabi virov nismo zabeležili, iz česar lahko sklepamo, da je bilo breme na ORIGSTRS konstantno in neodvisno od delovanja NETSERVER strežnika.

Večja zasedenost omrežja na strežniku v primerjavi z ORIGSTRS je posledica spremljanja rezultatov preko oddaljenega odjemalca.

4.4 Uporaba interceptorjev

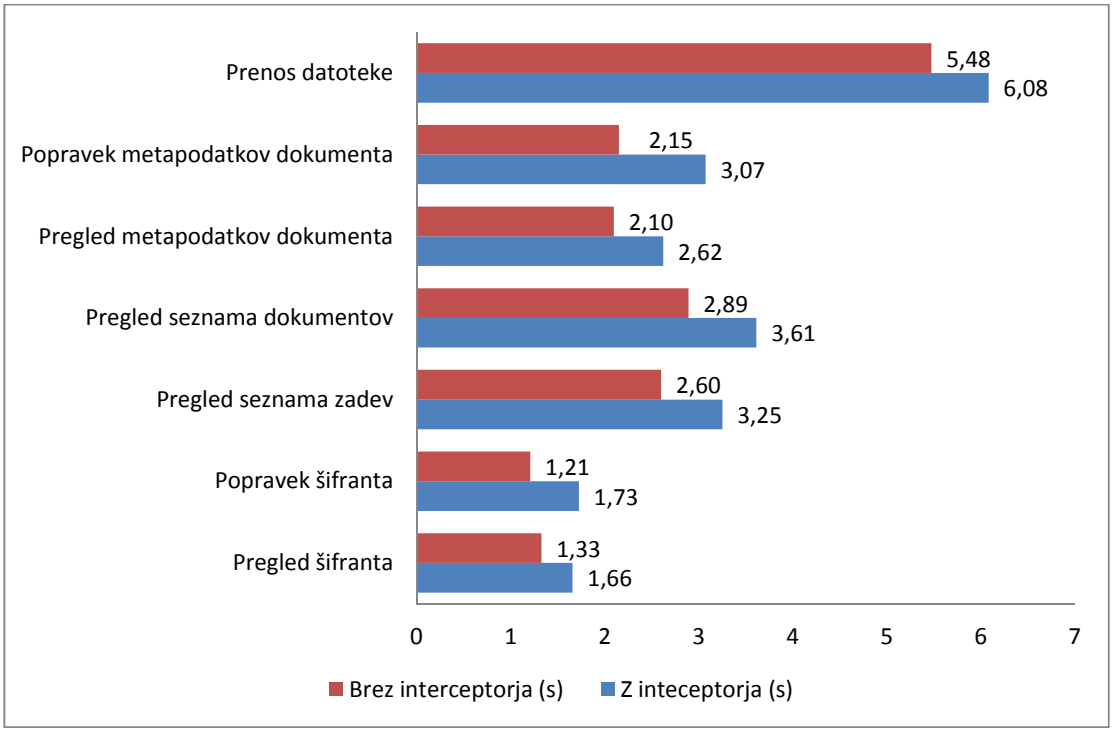
Nekaj ključnih funkcionalnosti Origamija je implementiranih s pomočjo interceptorjev. Interceptorji so primerljivi z SQL prožilci, vendar omogočajo dodatne funkcionalnosti (npr. izvajanje postopkov pred in po določenem klicu).

Uporabljajo se predvsem za preverjanje pravic izvajalca klica, zapisovanje zgodovine klicev in podobno. Ker interceptorji za določen odstotek upočasnijo delovanje aplikacije, jih je v Origamiju mogoče izključiti.

Najbolj ekspliciten primer interceptorja, ki upočasnjuje sistem (njegovo delovanje smo tudi izmerili), je interceptor za revizijsko sled. Na področju javne uprave je revizijska sled bistvenega pomena, medtem ko se veliko gospodarskih subjektov odloči, da ga ne bo uporabljalo prav zaradi zmogljivostnih težav, ki jih prinaša (na zasedenost podatkovnega skladišča in posledično na porabo sistemskih virov močno vpliva tudi velikost revizijske sledi).

Vsa testiranja smo opravili z vklopljenimi interceptorji, ker je pomembno, da aplikacija deluje dovolj hitro z vsemi funkcionalnostmi, ki jih ponuja.

Vse izvedene teste smo ponovili, tokrat z izklopljenimi interceptorji za beleženje revizijske sledi. Primerjava povprečnega časa izvajanja z in brez interceptorjev je prikazana na sliki 27.



Slika 27: Graf primerjave časa izvajanja operacij z in brez interceptorjev.

Povprečno zakasnitev aplikacije zaradi uporabe interceptorja delimo na dve vrsti:

- zakasnitev, ki nastane pri operacijah branja,
- zakasnitev, ki nastane pri operacijah pisanja.

Zakasnitev pri operacijah branja je nekje med 10% - 20%, medtem ko je zakasnitev pri operacijah pisanja (operacije, ki povzročajo zapise v revizijsko sled) dobrih 40%. Razliko lahko pojasnimo z dejstvom, da operacije pisanja potrebujejo dodaten dostop do podatkovnega skladišča, da bi zabeležile spremembe.

4.5 Analiza rezultatov

Z zgoraj navedenimi rezultati smo dosegli vse zastavljene zmogljivostne cilje, pri čemer nismo presegli maksimalne obremenitve sistema, določene v fazi načrtovanja testov.

V diplomskem delu je na enostaven način prikazano doseganje zmogljivostnih ciljev. V praksi je bilo potrebno izvesti več iteracij izvajanja testov, odpravljanja težav in izboljšav v kodi aplikacije. Kljub dobrim rezultatom, ki so presegli na začetku testiranja postavljene cilje, ima aplikacija še dodatne možnosti za njeno pohitritev. Pojavlja se vprašanje smiselnosti takega početja, ker rezultati izboljšave ne bodo bistveno vplivali na uporabniško izkušnjo končnih uporabnikov.

Dolgoročni cilj razvojne skupine je prilagoditev aplikacije za delovanje v oblaku (angl. *cloud computing*) Microsoft Azure in alternativno implementacijo podatkovnega skladišča na objektni bazi (NoSQL baza). Te spremembe bi vplivale tako na hitrost delovanja aplikacije, kakor tudi na visoko razpoložljivost ter stroške vzdrževanja strojne opreme.

5 Zaključek

Zmogljivostno testiranje je kategorija testiranja, ki v ljudeh vzbuja veliko negativnih občutkov. Takšni občutki so najpogosteje strah pred težavnostjo testiranja, dvom o primerni stopnji znanja izvajalcev testiranja, strah pred preveč neznanimi parametri, itd. Menim, da je negativni odnos do te vrste testiranja posledica zahtevnosti postopka in številnih spremenljivk, ki so prisotne pri zmogljivostnem testiranju.

Pri zmogljivostnem testiranju se pogosto srečujemo z nenatančno opredeljenimi cilji in le s težavo določimo, kaj je »slabo« med obremenitvenim testiranjem.

V situaciji, ko se na primer odzivni čas določenega zahtevka iz 1 sekunde poveča na 2 sekundi pri povečanju iz 100 sočasnih na 250 sočasnih uporabnikov, se pojavlja vprašanje, če je to »slabo«. Morda je, morda ni. V osnovi nam bo zmogljivostno testiranje priskrbelo samo informacije, ki nam bodo pomagale pri sprejemanju ustrezne odločitve.

V diplomskem delu smo prišli do sklepa, da so bistvo uspešnega zmogljivostnega testiranja jasno določeni cilji. Vedeti moramo namreč kaj je potrebno preveriti ter se osredotočiti na ključne aspekte naše aplikacije. Bistvenega pomena so tudi primerno definirani merilni parametri s postavljenimi pravili in dobro poznavanje lastne aplikacije.

Zmogljivostno testiranje je časovno potratno, kar moramo upoštevati pri pripravi načrta testiranja in začeti z izvedbo takoj, ko je to mogoče. Priporočeno je, da se testiranje začne že med razvojem aplikacije.

Kot člani Origami razvojne ekipe smo zelo dobro poznali aplikacijo, zato smo brez oklevanja sprejeli nalogo izvedbe zmogljivostnega testiranja. Ne glede na dobro poznavanje aplikacije, smo ob začetku testiranja opazili, da smo podcenjevali samo zahtevnost izvedbe. Ugotovili smo, da se krivulje učenja ne da spregledati in zaobiti. Ponovno smo nameščali operacijske sisteme, poenostavljali dostopne pravice in menjavali okvarjene trde diske. Med delom smo osvojili delovanje kar nekaj novih orodij, ki so nam pomagala uspešno zaključiti nalogo.

Ob zaključku testiranja smo ugotovili, da je vredno porabiti določen čas, denar in ostala sredstva za testiranje. Testiranje je ključni in neobhodni proces, da bi se pri uporabi aplikacije v produkciji izognili nepredvidenim stroškom, izgubi podatkov, prekinitvam delovanja sistema in zmanjšanju ugleda podjetja.

6 Kazalo slik

7 Kazalo tabel

8 Literatura in viri

[1] Arnold, T., Hopton, D., Leonard, A., & Frost, M. (2007). Professional SoftwareTesting with VisualStudio® 2005 Team System. Wiley.

[2] Day, B. (brez datuma). Web Performance Testing with Visual Studio 2010. Dostopno na http://visualstudiomagazine.com/articles/2010/06/17/web-performance-testing-with-visual-studio-2010.aspx

[3] Fiddler PowerToy. (brez datuma). Dostopno na http://msdn.microsoft.com/en-us/library/Bb250446.aspx

[4] Liu, H. H. (2009). Software Performance and Scalability: A Quantitative Approach. Wiley.

[5] Meier, J., Farre, C., Bansode, P., Barber, S., & Rea, D. (2007). Performance Testing Guidance for Web Applications. Microsoft Corporation.

[6] Microsoft. (brez datuma). Visual Studio. Dostopno na http://www.microsoft.com/slovenija/msdn/vs2008/default.mspx

[7] Molyneau, I. (2009). The Art of Application Performance Testing: Help for Programmers and Quality Assurance. O'Reilly Media.

[8] neXpert Performance Tool. (brez datuma). Dostopno na http://blogs.msdn.com/b/nexpert/archive/2009/02/10/introducing-nexpert.aspx

[9] Souders, S. (2007). High Performance Web Sites. O'Reilly Media.