TAMPEREEN
AMMATTIKORKEAKOULU

MALLIPOHJAINEN SUORITUSKYKYTES-TAUS SIVUMALLIEN AVULLA

Mikko Kukkola

Opinnäytetyö
Maaliskuu 2016
Tietotekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Ohjelmistotekniikka

KUKKOLA MIKKO:
Mallipohjainen suorituskykytestaus sivumallien avulla

Opinnäytetyö 34 sivua, joista liitteitä 0 sivua
Maaliskuu 2016

Tässä työssä käydään läpi suorituskykytestauksen teoriaa, sekä alusta alkaen kaikki vaiheet mallipohjaisen suorituskykytestin rakentamiseksi ja ajamiseksi sivumallien avulla. Tällä tavoin todistetaan mallipohjainen suorituskykytestaus sivumallien avulla toimivaksi kokonaisuudeksi. Työ on tehty Q-Factory Oy:lle ja suorituskykytestit Q-Factoryn asiakasyrityksille. Työnteossa käytettiin QAutomate- ja MBPeT-työkaluja.

Testin teossa mallinnetaan ensin tarvittavat sivumallit ja luodaan testimetodit sivumalleille. Tämän jälkeen rakennetaan mallipohjainen testi ja generoidaan suorituskykypaketit suorituskykytestiä varten. Lopuksi suunnitellaan suorituskykytestin ajo ja analysoidaan tuloksia ajon jälkeen.

Asiasanat: suorituskykytestaus, mallipohjainen, sivumalli

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information Technology
Software Engineering

KUKKOLA MIKKO:
Model-based performance testing with page models

Bachelor's thesis 34 pages, appendices 0 pages
March 2016

This thesis describes theory of performance testing and all the steps to build and run model-based performance test with pagemodels. The process shows that the model-based performance test with pagemodels is a functional entity. This process was implemented for Q-Factory Co and performance tests for customer companies. Tools used in this process were QAutomate and MBPeT.

First steps in test build phase are to generate necessary pagemodels and methods for pagemodels. After that is time to build model-based test and generate control packets for performance test. Last steps are to design performance test and after running the test to analyze the results.

Key words: performance testing, model-based, pagemodel

SISÄLLYS

LYHENTEET JA TERMIT

bugi	Ohjelmointivirhe, toimintahäiriö
beta-testi	Joukko tavallisia käyttäjiä testaa vielä kehitysvaiheessa olevaa tuotetta
SOAP	Simple Object Access Protocol eli tietoliikenneprotokolla etäkutsuille
REST	Representation State Transfer, http-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen
API	Ohjelmointirajapinta
master	Isäntäkone, hallitsee muita tietokoneita
slave	Orjatietokone, hallitaan toiselta tietokoneelta
IP-osoite	Internetin protokollaosoite, käytetään verkkosovittimien yksilöimiseen
SUT	System under test, eli testattava järjestelmä
XML	eXtensible Markup Language, merkintäkieli
URL	Uniform Resource Locator, käytetään osoittamaan WWW-sivuja

1 JOHDANTO

Työn tarkoituksena on perehtyä suorituskykytestaukseen ja todistaa, että mallipohjainen suorituskykytestaus sivumallien pohjalta on toimiva kokonaisuus ja on myös varteenotettava vaihtoehto suorituskykytestausta tehdessä. Työ tehtiin Q-Factory Oy:lle. Q-Factory on laadunvarmistukseen ja testaukseen keskittynyt ICT-alan yritys.

Työn teossa käytettiin yrityksen omassa kehityksessä olevaa QAutomate-työkalua sekä Åbo Akademin MBPeT-työkalua. QAutomatella tehtiin testit ja MBPeT suoritti suorituskykytestauksen. Testikielenä molemmilla työkaluilla on Python.

Työssä käydään läpi ensin suorituskykytestausta yleisesti. Sen jälkeen perehdytään suorituskykytestauksen vaiheisiin aina suunnittelusta tulosten analysointiin asti. Esimerkkeinä käytetään Ekocoil Oy:n laskentasovellukseen tehtyä suorituskykytestaustyötä, jonka pohjalta tätä opinnäytetyötä on tehty.

Tämän työn aikana on koko ajan jatkokehitetty testaukseen käytettyjä työkaluja.

2 SUORITUSKYKYTESTAUS

Seuraavissa kappaleissa kerrotaan ensin ohjelmistotestauksen taustaa ja lopulta perehdytään suorituskykytestaukseen ja sen eri menetelmiin.

2.1 Ohjelmistotestaus yleisesti

Ohjelmistotestauksen väitetään saaneen alkunsa kenties ensimmäisestä havaitusta bugista, joka oli kirjaimellisesti hyönteinen tietokoneen sisällä aiheuttaen toimintahäiriön. Tämä tapahtui vuonna 1947 Harvardin Yliopistossa. Ohjelmistotestauksen tavoitteena on löytää testattavasta kohteesta kaikki virheet ja ongelmakohdat. Alkuajoista ohjelmistotestaus on kehittynyt huimin harppauksin, kuten on itse ohjelmistokehityskin. Nykyään on yleensä erikseen ohjelmiston kehittäjät ja testaajat. Tämä on tehokkaampaa tekemisen sekä testauksen kannalta, sillä tekijä on yleensä helposti sokea omille virheilleen ja näin ei tarvitse yhden henkilön välttämättä keskittyä useaan eri asiaan. Ohjelmistotestauksen kehityksen tuloksena testaus on tänä päivänä ennalta suunniteltua, järjestelmällistä ja usein myös automatisoitua.

Automatisoitu testaus näkyy usein esimerkiksi jatkuvassa integraatiossa. Kun ohjelmakoodiin tehdään muutoksia ja ne ajetaan versionhallintaan, tehdään ohjelmiston uudelle versiolle automaattisesti heti määrätyt testit, jotta varmistutaan siitä, ettei mitään oleellista ole hajonnut ohjelmistossa.

2.2 Suorituskykytestaus yleisesti

Suorituskykytestaus on yksi ohjelmistotestaustyypeistä. Nimensä mukaisesti suorituskykytestauksen päämääränä on määrittää testattavan järjestelmän suorituskyky. Suorituskyvyllä tarkoitetaan sitä, kuinka tehokkaasti ja nopeasti tietty päämäärä saavutetaan. Suorituskykytestauksen tuloksina saadaan vasteaikoja, tietoa suorituskyvyn toiminnallisista rajoista, tuntemusta järjestelmän toiminnasta ja mahdollisista pullonkauloista järjestelmässä. Ideaali järjestelmä suorituskyvyn puolesta on nopea ja vakaa kaikissa tilanteissa vaihtelevilla käyttäjämäärillä.

Suorituskykytestauksessa voidaan ajatella olevan kolme mahdollista päämäärää:

- Selvittää täyttääkö järjestelmän suorituskyky sille annetut vaatimukset
- Selvittää järjestelmästä ne osat, jotka heikentävät suorituskykyä
- Selvittää, mikä järjestelmä on suorituskyvyltään paras vertailemalla niitä

Asetettujen vaatimusten täyttäminen on ensisijaisen tärkeää. Maailmassa on monia järjestelmiä, kuten hätäkeskusjärjestelmä, joiden tulee olla käytännössä jatkuvasti saatavilla. Niistä käytetään nimityksiä sen mukaan, kuinka monta numeroa yhdeksän on niiden vuotuisessa saatavuusprosentissa.

TAULUKKO 1. Järjestelmän saatavuusprosentti vuodessa muutettuna ajaksi jolloin se ei ole saavutettavissa.

Saatavuus (%)	**Alhaalla oloaika vuodessa**
90	876 h
95	438 h
99	87,69 h
99,9	6 h
99,99	52,56 min
99,999	5,256 min
99,9999	31,536 s
99,99999	3,15 s

Taulukosta 1. on selkeästi nähtävissä, että tällaisissa järjestelmissä ei juurikaan ole varaa olla saavuttamattomissa. Taulukossa on nähtävillä ajat seitsemään yhdeksään asti. Jos vielä lisätään yksi yhdeksän lisää niin ajat painuvat jo alle sekunnin.

Tehokkainta suorituskykytestaus on järjestelmän lopputuloksen kannalta silloin kun se on aloitettu jo varhaisessa vaiheessa järjestelmän kehityksen aikana. Usein suorituskykytestaus kuitenkin jää kehityksen loppuvaiheeseen, jolloin suuria muutoksia ei välttämättä ole enää helppoa eikä järkevää tehdä. Syynä tähän lienee resurssien sekä ajan puute. Lopulta suorituskykytestaus on tietenkin myös tehtävä valmiilla järjestelmällä, jotta varmistutaan lopullisesta suorituskyvystä.

2.2.1 Suorituskykyyn vaikuttavat tekijät

Suorituskykyyn vaikuttavat monet asiat niin yksittäin kuin yhdessä. Web-palveluista puhuttaessa suorituskykyyn vaikuttavia tekijöitä ovat jo pitkään olleet esimerkiksi:

- korkea kävijäaktiivisuus
- vuorokaudenaika
- aktiivisuuspiikit esim. mainonnan takia
- pullonkaulat yhtäaikaisten käyttäjien takia
- latausajat
- käyttökaavat
- odotusajat
- käyttäjien saapumismäärät
- käyttöalustat
- käyttäjien internet-nopeudet
- käyttäjien poistumismäärät

(Tamres 2002, 177)

2.2.2 Suorituskykytestauksen hyödyt

Suorituskykytestaaminen, kuten kaikenlainen muukin kunnollinen ohjelmistotestaaminen, tuo yrityksen budjettiin lisämenoja. Siitä huolimatta lopulta, varsinkin jos kyseessä on suuri järjestelmä käyttäjämäärien puolesta, on suorituskykytestaaminenkin erittäin suositeltavaa. Yrityksen tappiot niin rahallisesti kuin maineenkin puolesta ovat todennäköisesti paljon suuremmat, mikäli järjestelmää ei kunnolla testata ja lopulta käytössä se ei toimikaan kunnolla saadessaan kuormitusta. Hyvänä esimerkkinä voidaan ottaa internetissä toimiva kauppa tai verkossa toimiva monipeli. Jos tuote ei toimikaan julkaisussa, kun kuluttajat yrittävät käyttää sitä, aiheuttaa se kuluttajille mielipahaa ja sen seurauksena yrityksen imago kuluttajien silmissä laskee. Lisäksi yritys joutuu tekemään korjaustoimenpiteitä, jotka aiheuttavat lisäkustannuksia. Pahimmassa tapauksessa yrityksen täytyy tehdä kokonaan uusi tuote.

Suorituskykytestauksen tekeminen tuottaa kustannuksia ainakin ohjelmistojen hankinnan puolelta, mutta sekä myös mahdollisesti testaajan puolelta, mikäli täytyy palkata erikseen henkilöt testaukseen. Kustannuksiin vaikuttaa testauksessa käytettävä ohjelmisto, niin hintansa kuin sen helppokäyttöisyydenkin puolesta. Mikäli ohjelmisto on hyvillä ohjeilla varustettu ja sopivan yksinkertainen, niin välttämättä erillisen testaajan palkkaus ei ole välttämätöntä. Toisaalta vaikka testien rakentamisesta ja ajamisesta selvittäisiin ilman erillistä asiantuntijaa, niin usein lopullinen analysointi vaatii syvällisempää osaamista ympäristöstä ja tekniikoista.

Suorituskykytestauksen voi tehdä myös ilman erillistä ohjelmistoa, mutta silloin se ei ole lähellekään yhtä tehokasta. Tässä tapauksessa vaihtoehtona on kuorman luominen realistisesti oikeilla yhtäaikaisilla käyttäjillä, mutta silloin kustannukset nousevat nopeasti todella korkeiksi ihmisille maksettavien korvausten takia.

Kunnolla tehty suorituskykytestaus takaa tuotteelle pidemmän elinkaaren, todennäköisesti pienemmät rahalliset kustannukset ja kenties sen tärkeimmän, eli hyvän maineen yritykselle. Hyvän suorituskyvyn omaavaa järjestelmää on miellyttävä käyttää ja se luo kuluttajalle paremman käyttökokemuksen, josta on mukava kertoa eteenpäin.

2.3 Suorituskykytestaustyypit

Suorituskykytestaus voidaan jakaa eri testityyppeihin, sillä perusteella, että mitä testissä erityisesti tutkitaan. Käytännössä kuitenkin kaikki testit toteutetaan luomalla kuormaa testattavalle järjestelmälle. Usein testit täytyy ajaa useampia kertoja läpi, jotta saadaan varmasti luotettavia tuloksia.

2.3.1 Kuormitustesti

Kuormitustesti on yleisin suorituskykytestauksen muoto. Kuormitustestauksessa järjestelmää kuormitetaan, jotta saataisiin selville sen käyttäytyminen ja toiminta rasituksen alla sekä mahdolliset pullonkaulakohdat. Yleensä testauksessa luodaan sellainen kuorma, jota järjestelmälle odotetaan yleisimmin tulevan. Testissä määrätään tietty määrä käyttä-

jiä tekemään määrättyjä operaatioita. Yleensä käyttäjät generoidaan ohjelmallisesti automaatiota hyväksikäyttäen, joten oikeita ihmisiä ei tarvita suuria määriä. Jossain tapauksissa kuormitustesti on pakko tehdä oikeilla käyttäjillä, kuten esimerkiksi pelitalot järjestävät useasti ennen uuden pelin julkaisua beta-testejä kuluttajille. Näin he pääsevät testaamaan palvelimiensa toimivuutta ennen tuotteen varsinaista julkaisua. Testi antaa tuloksena esimerkiksi vasteaikoja.

2.3.2 Stressitesti

Stressitestauksen tarkoituksena on ymmärtää ja saada selville järjestelmän suorituskyvyn ylärajat. Järjestelmälle luodaan erittäin suuri kuorma, kuorma joka on yli odotettujen ylärajojen. Näin osataan paremmin ennaltaehkäistä mahdollisia järjestelmän kaatumisia, kun tiedetään millaisilla käyttäjämäärillä kaatuminen on todennäköistä.

2.3.3 Kestävyystesti

Kestävyystestissä selvitetään kuinka testattava järjestelmä sietää jatkuvasti odotettua kuormaa. Kestävyystestien aikana tarkkailussa on erityisesti muistin käyttö muistivuotojen kiinnisaamiseksi ja ehkäisemiseksi. Kestävyystestin tuloksista voidaan myös hyvin vertailla vasteaikojen mahdollisia eroavaisuuksia testin alussa ja lopussa.

2.3.4 Piikkitesti

Piikkitestissä nostetaan arvaamattomasti ja nopeasti järjestelmän kuorma erittäin korkeaksi. Tällä testillä saadaan selville seuraukset esimerkiksi mahdollisesti varsinkin webpalveluihin kohdistuvista nopeista piikeistä yhtäaikaisissa käyttäjämäärissä. Piikkitestin tilanne tulee oikeasti eteen joidenkin järjestelmien kohdalla esimerkiksi uuden järjestelmän julkaisun jälkeen, kun monet yhtäaikaiset käyttäjät yrittävät päästä palveluun pienen ajan sisällä. Tällä testillä varmistutaan myös siitä että järjestelmän palautuu ruuhkahuippujen välissä.

2.3.5 Kokoonpanotesti

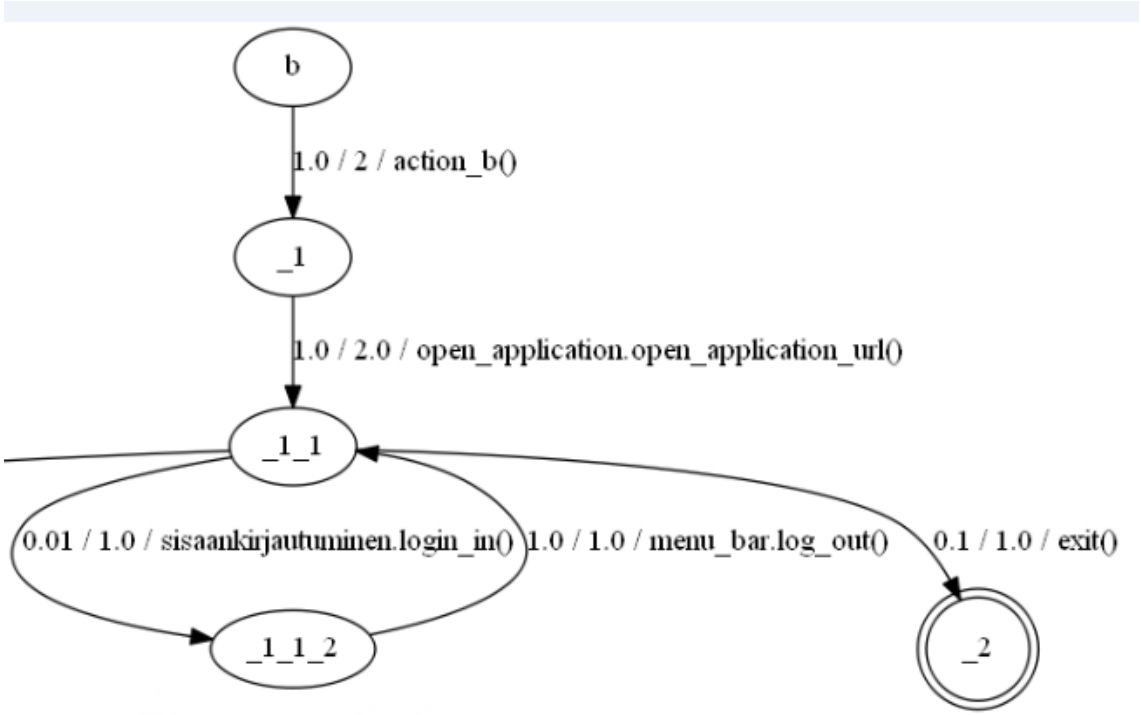
Kokoonpanotesti ei testaa järjestelmää niinkään kuorman näkökulmasta, vaan erilaisten järjestelmäasetusten puolelta. Testissä testattavan järjestelmän asetuksia tai sen komponenttien asetuksia muutetaan ja seurataan muutosten aiheuttamia mahdollisia muutoksia järjestelmän toiminnassa. Tällä tavoin saadaan selville optimaalisimmat asetukset järjestelmän toiminnan kannalta.

2.3.6 Eristystesti

Eristystesti ei käytännössä ole erillinen testi vaan siinä toistetaan virheellisen toiminnan aiheuttanutta testiajoa, jotta saataisiin selville virheellisen toiminnan aiheuttaja. Testissä suljetaan pois mahdollisia vian aiheuttaja, kunnes syy on selvillä.

2.4 Mallipohjaisuus

Mallipohjaisesta testauksesta puhuttaessa on kyse testauksesta, jossa testitapaus seuraa ennalta suunniteltua kaaviota eli mallia. Testi ei ole vain yksi suoraviivainen suoritus pisteestä a pisteeseen b, vaan se on silmukan mallinen ja siinä voi olla useita haaroja. Haarojen ansiosta testi ei toista kokoajan samaa kaaviota. Haaroille voidaan myöskin antaa painotuksia, jotta saadaan realistisemmin simuloitua oikean käyttäjän toimintoja.



KUVA 1. Kuvassa osa testattavan laskentasovelluksen mallipohjaisesta testistä vaiheineen ja polkuineen

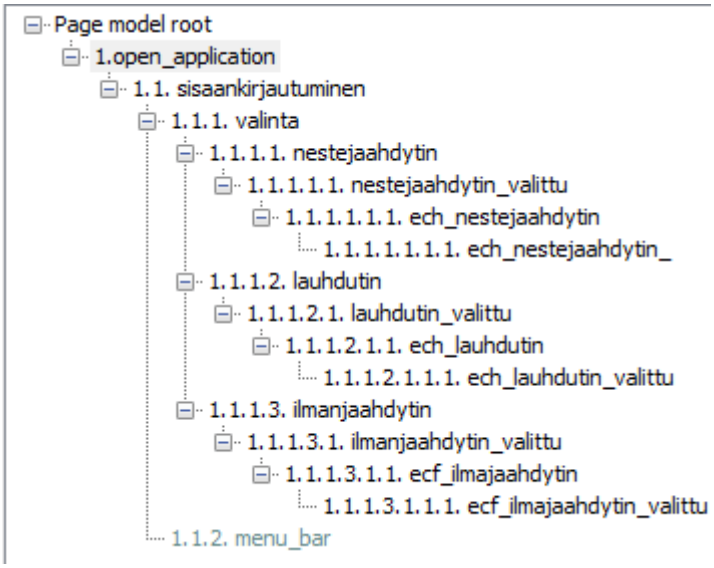
Kuvasta 1. hahmottaa helposti idean, mitä mallipohjaisella testillä tarkoitetaan. Numero yksi kuvassa tarkoittaa pistettä, josta testi lähtee liikkeelle ja numero kaksi taas poistumispistettä. Jokaisessa testissä voi olla vain yksi alkupiste ja yksi loppupiste. Haaroja ja erilaisia silmukoita voi olla useita. Kuvassa vasemmalle johtava katkeava viiva johtaa laskentasovelluksen kolmeen eri haaraan.

Mallipohjaista testiä ajettaessa voidaan ajo toteuttaa aikaan pohjautuvasti, tai prosenteilla mitattuna haarakattavuuteen pohjautuvasti.

2.5 Sivumallit

Sivumalli tarkoittaa web-sivusta tehtyä identtistä mallia, jota käytetään hyväksi testien luomisessa ja ajossa. Sivusta generoidaan malli keräämällä kaikki tarvittavat elementit tietoineen selaimessa näkyvältä sivulta. Testiä luodessa painetaan halutusta mallista haluttua mallinnettua elementtiä ja luodaan sille jokin funktio. Funktio voi olla toiminnallinen tai odottava. Toiminnalliset funktiot tekevät samoja asioita kuin oikeakin käyttäjä selatessaan internet-sivuja eli esimerkiksi hiiren painalluksia. Odottavat tai varmentavat

funktiot odottavat, että jokin asia saadaan tehtyä, jokin elementti on varmasti oikeanlainen oikeassa paikassa tai että jokin elementti on varmasti näkyvissä.



KUVA 2. Sivumallit järjestetään sivumallipuuhun sivuston rakenteen mukaan

Kuvassa 2. laskentasovellustestin sivumallit järjestyksessä. Kaikki mallit ovat kokonaisia sivuja paitsi menu_bar, joka on aluemalli sivuston yläpalkista. Yläpalkki toistuu samanlaisena joka sivulla, joten kun siitä tehdään oma mallinsa ja luodaan tarvittavat metodit sille, niin vältytään duplikaattien metodien kirjoittamiselta. Tällaista sivumallia voidaan hyödyntää missä tahansa testin vaiheessa.

3 TYÖKALUT

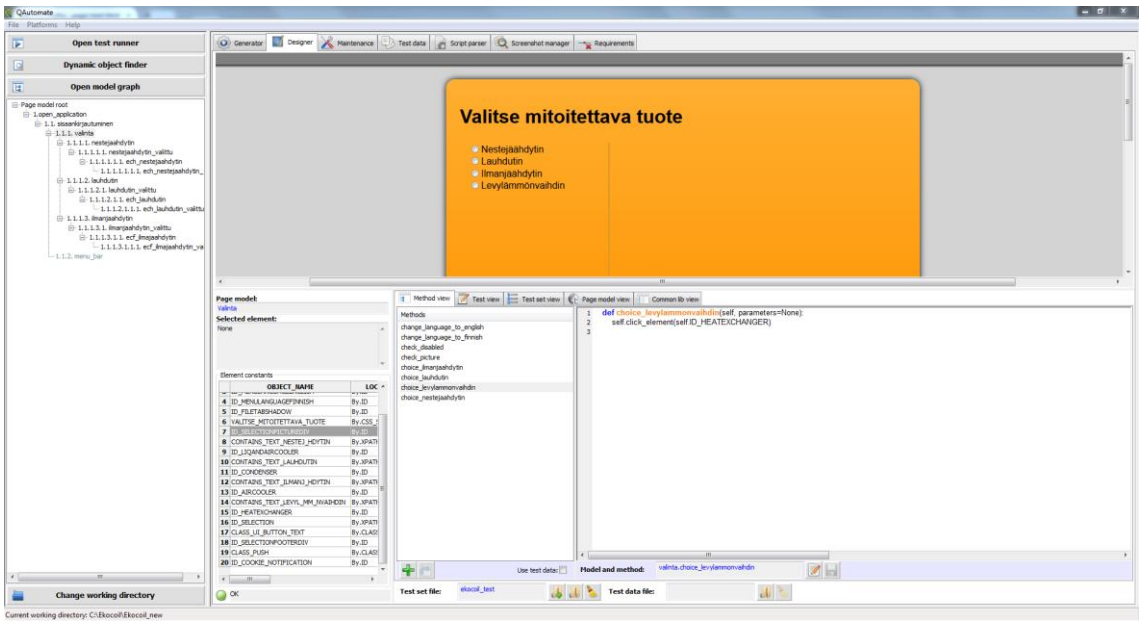
Tämän opinnäytetyön teossa on käytetty kahta työkalua, QAutomate ja MBPeT. (QAutomate, 2016 ja Model-Based Performance Testing Using the MBPeT Tool, 2013)

3.1 QAutomate-työkalu

QAutomate on Selenium WebDriveriin sekä Appiumiin perustuva maksullisen lisenssin alainen testausautomaatiotyökalu ja se soveltuu web- ja Android-mobiilisovelluksille. Testit pohjautuvat sivumalleihin ja niitä voidaan luoda ja ajaa visuaalisesti. Testien kielenä toimii Python. QAutomate on monipuolinen työkalu, jolla voidaan tehdä seuraavia testauksia:

- funktionaalista testausta
- suorituskykytestausta
- mallipohjaista testausta
- suorituskykytarkkailua
- visuaalisen ulkoasun testausta
- SOAP API -testausta
- REST API -testausta
- JavaScript-muistivuotojen tutkimista

Tässä työssä on käytetty QAutomatea testien suunnitteluun ja luomiseen, sekä testien suoritusvaiheessa suorituskykytarkkailuun.

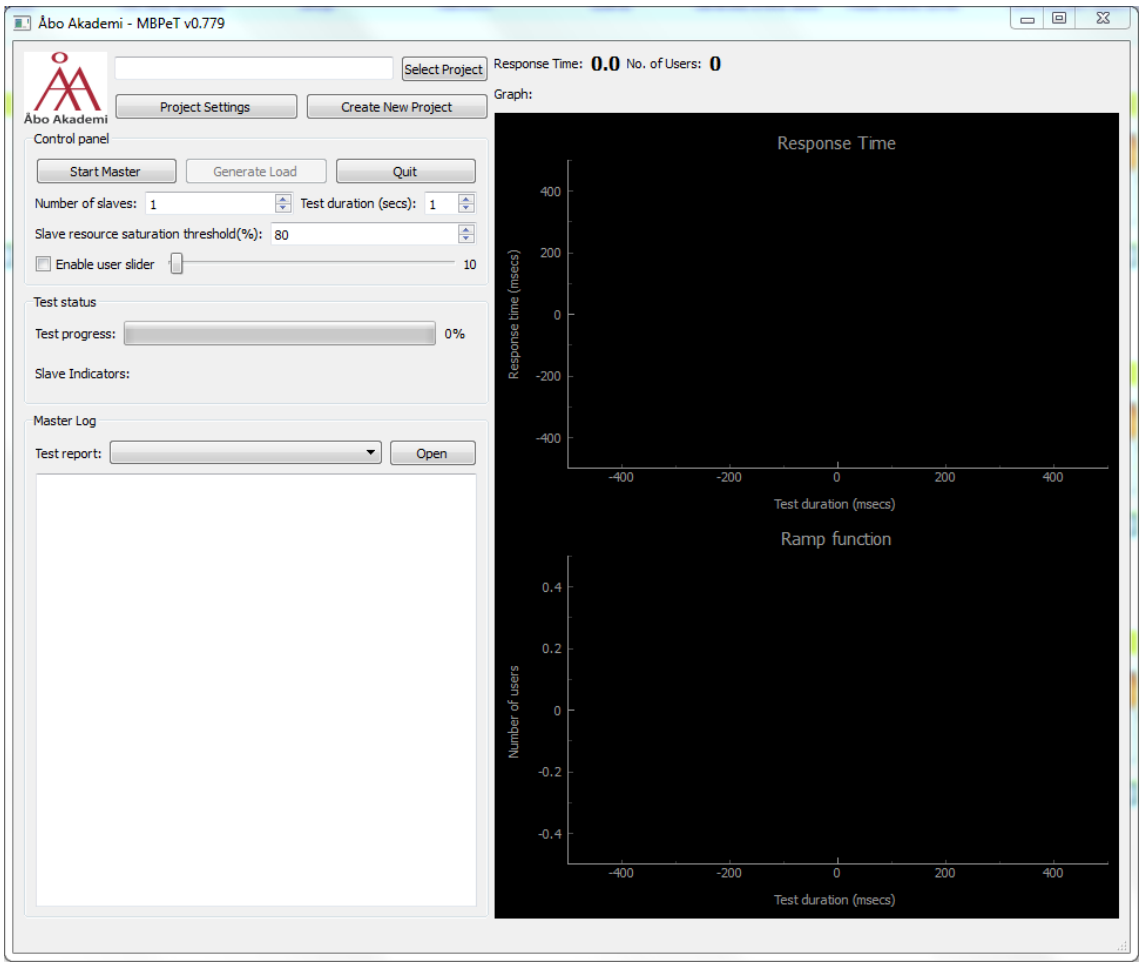


KUVA 3. QAutomaten käyttöliittymä, avattuna testien luonti -välilehti

Kuvassa 3. on nähtävissä kuinka sivumalleja hyödynnetään testien luontivaiheessa. Yllä näkyy kuva valitusta sivumallista ja sen alapuolella sivumallissa olevat mallinnetut elementit. Elementtilistauksen vieressä on välilehdet, joissa on nähtävissä ja muokattavissa metodit ja testit. Sivumallit ovat myös nähtävissä puumallissa vasemmalla.

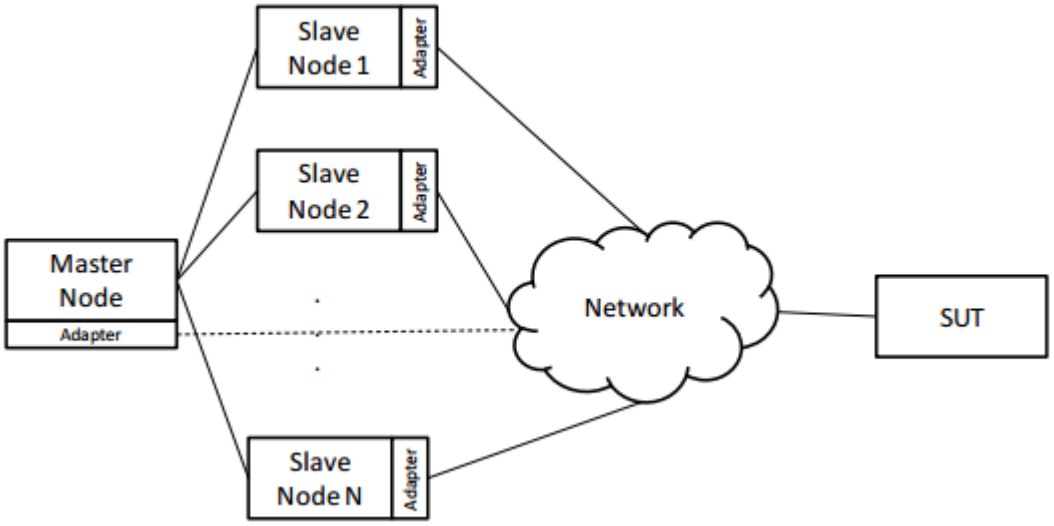
3.2 MBPeT-työkalu

Åbo Akademin MBPeT on kuormitustestaustyökalu, joka tukee Pythonilla tehtyjä mallipohjaisia testejä. MBPeT koostuu kahdesta osasta, masterista ja slavesta. Master on saatavilla sekä Windows että Linux järjestelmille, mutta slave on saatavilla vain Linuxille. Masterilla on graafinen käyttöliittymä, kun taas slave:a ajetaan vain terminaalista ilman graafista käyttöliittymää.



KUVA 4. MBPeT Master-käyttöliittymä. Kuormitustestin aikanakin käyttäjä pystyy seuraamaan tuloksia kuvassa näkyvillä diagrammi-alueilla

Master-koneella säädetään kuormitustestin asetukset, laitetaan testi käyntiin ja saadaan testitulokset. Ennen testin aloitusta luodaan yhteys masterin ja slaven välille antamalla slavelle masterin IP-osoite. Testin alussa master lähettää määritetyt asetukset slavelle. Nämä vastaanotettuaan testaaja voi lähettää masterilta käskyn slavelle aloittaa kuorman luomisen. Slaven ainoa tehtävä on luoda kuorma ja suorittaa testi. Testin jälkeen masterilla on tarkasteltavissa testiraportti. Testiraportti sisältää suorituskykytietoa testin eri vaiheista numeroina, sekä graafisina esityksinä. Kuvassa 5. havainnollistettuna yhteydet masterin, slaven ja testattavan järjestelmän välillä.



KUVA 5. MBPeT-työkalun arkkitehtuuri (Model-Based Performance Testing Using the MBPeT Tool, 2013)

MBPeT -työkalua käytettiin tässä työssä kuorman luomiseen. Master oli Windows järjestelmällä ja slavea ajettiin Ubuntulla.

4 TESTITAPAUKSET

Tässä työssä suorituskykytestejä on ajettu kahteen eri järjestelmään. Seuraavissa kappaleissa kerrotaan vaiheet aina testien suunnittelusta lopputuloksien analysointiin asti.

4.1 Testien luonti

Testien tekeminen on hyvä jakaa useampaan vaiheeseen tekemisen yksinkertaistamiseksi, jotta ei tule tehtyä kaikkea sekaisin. Huolellinen testin suunnittelu on tärkeä osa onnistuneen lopputuloksen saavuttamisessa. Hyvin tehdyn suunnittelun jälkeen on helpompaa sekä selkeämpää aloittaa luomaan ensimmäisiä testitapauksia.

4.1.1 Suunnittelu

Toimivaan ja kannattavaan suorituskykytestiin tähdättäessä kannattaa miettiä tarkasti mitä, miksi ja miten halutaan testata. Usein määritykset testeistä tulevat testien tilaajalta eli asiakkaalta. Yleensä on valmiiksi tiedossa testattavan järjestelmän raskaimmat, eniten suorituskykyä vaativat, osat. Nämä ovat useimmiten niitä osia, joiden suorituskykyä halutaan mitata.

Testien suunnittelussa on kuitenkin hyvä pitää mielessä se, että kannattavinta on testata eniten niitä järjestelmän osia, joita oikeatkin käyttäjät eniten käyttävät. Hyvä testitapaus on mahdollisimman autenttinen oikean käyttäjän toimintaan verrattuna.

4.1.2 Metodien luonti

Testitapaukset koostuvat metodeista, jotka sisältävät web-sivustoilla tehtäviä toimintoja. Yleisimmät metodit ovat nappien painalluksia, tekstikenttien täyttöä ja elementtien varmentamista. Yksi metodi voi sisältää useamman eri toiminnon. Esimerkkinä luodaan metodi sisäänkirjautumiselle. Ensimmäinen askel on täyttää kirjautumistunnus, toinen on täyttää salasana ja kolmas on klikata kirjaudu sisään-nappia. Kuvassa 5. on esitetty metodi, jota käytetään sisäänkirjautumisessa.

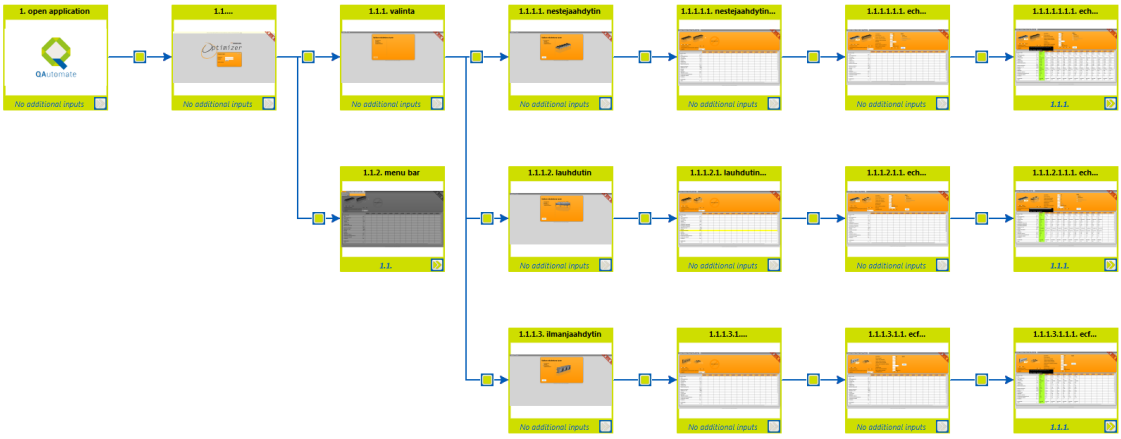
```
1   def login(self, parameters=None):
2       self.input_text(self.USERNAME_INPUTMIDDLE_URL, parameters[u'j_username'])
3       self.input_text(self.PASSWORD_INPUTMIDDLE_URL, parameters[u'j_password'])
4       self.click_element(self.ID_SUBMIT)
```

KUVA 6. Esimerkki kirjautumismetodista

Kuvassa 6. käytetään input_text-funktiota kirjoittamaan käyttäjänimen ja salasanan niille tarkoitettuihin kenttiin ja lopulta click_element-funktiota klikkaamaan submit-nappia. Funktioiden ensimmäiset parametrit identifioivat kohteena olevan elementin ja mahdollinen toinen parametri sisältää datan. Testidata on tallennettuna XML-tiedostossa.

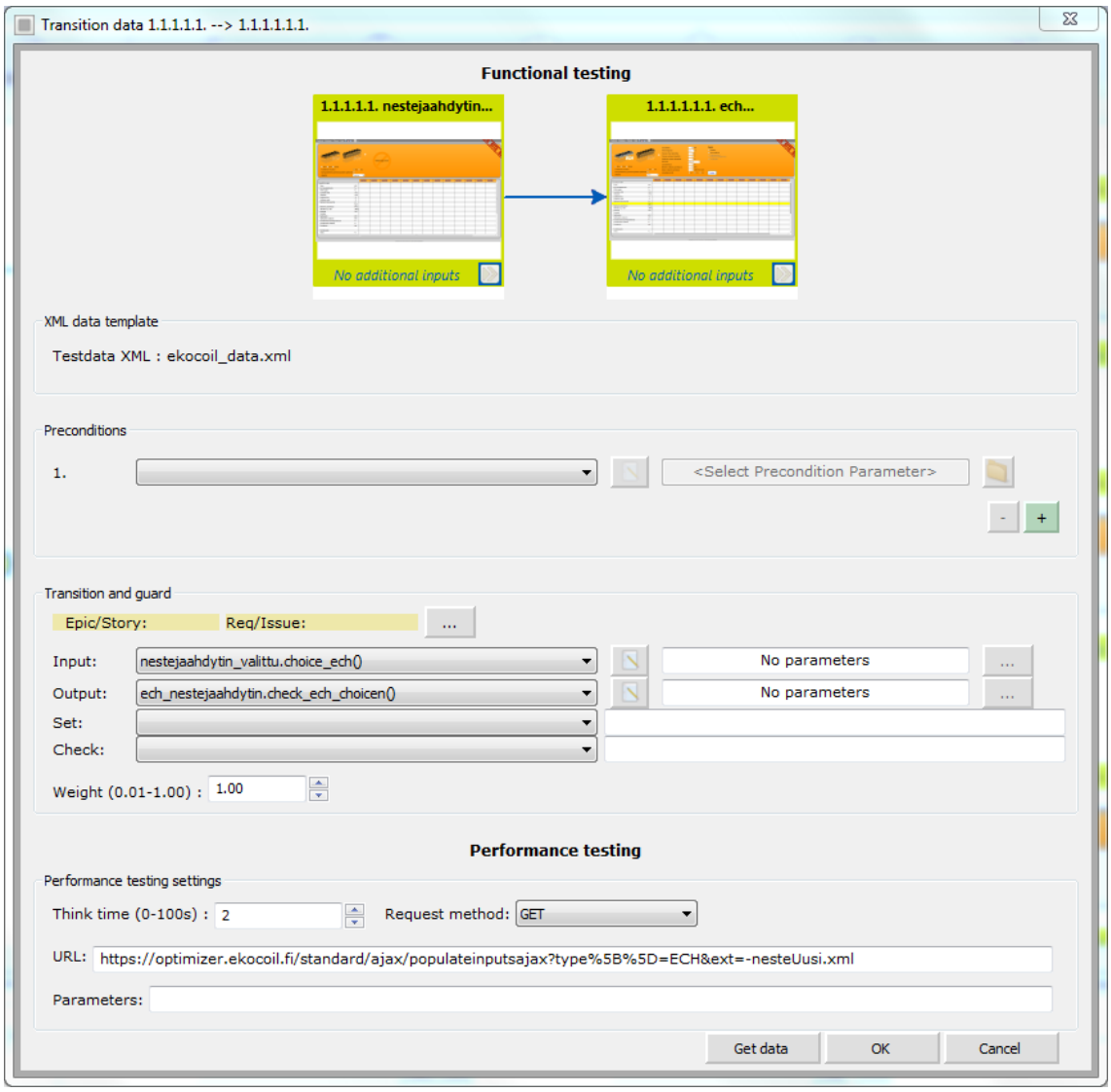
4.1.3 Mallipohjaisen testin luonti

Mallipohjaisen testin luonnissa hyödynnetään luotuja metodeja. Metodit sijoitellaan sivumallisiirtymien väleille. Jokaisella siirtymällä on input-metodi ja output-metodi, sekä mahdollisesti ennakkometodeja.



KUVA 7. Visuaalinen-malli mallipohjaisesta testistä

Kuvassa 7. on nähtävissä kaikki testissä olevat sivumallit ja siirtymät niiden välillä. Kuvan mallissa on haaroja, joten testiä luotaessa on asetettava haaroille todennäköisyydet, eli kuinka todennäköistä on, että testi ajautuu mihinkin haaraan. Mallin viimeisistä sivumalleista luodaan siirtymä johonkin aiempaan sivumalliin, eli luodaan silmukka.



KUVA 8. Valintapaneeli, josta käyttäjä asettaa painotuksen ja tarvittavat metodit mahdollisine parametreineen siirtymälle

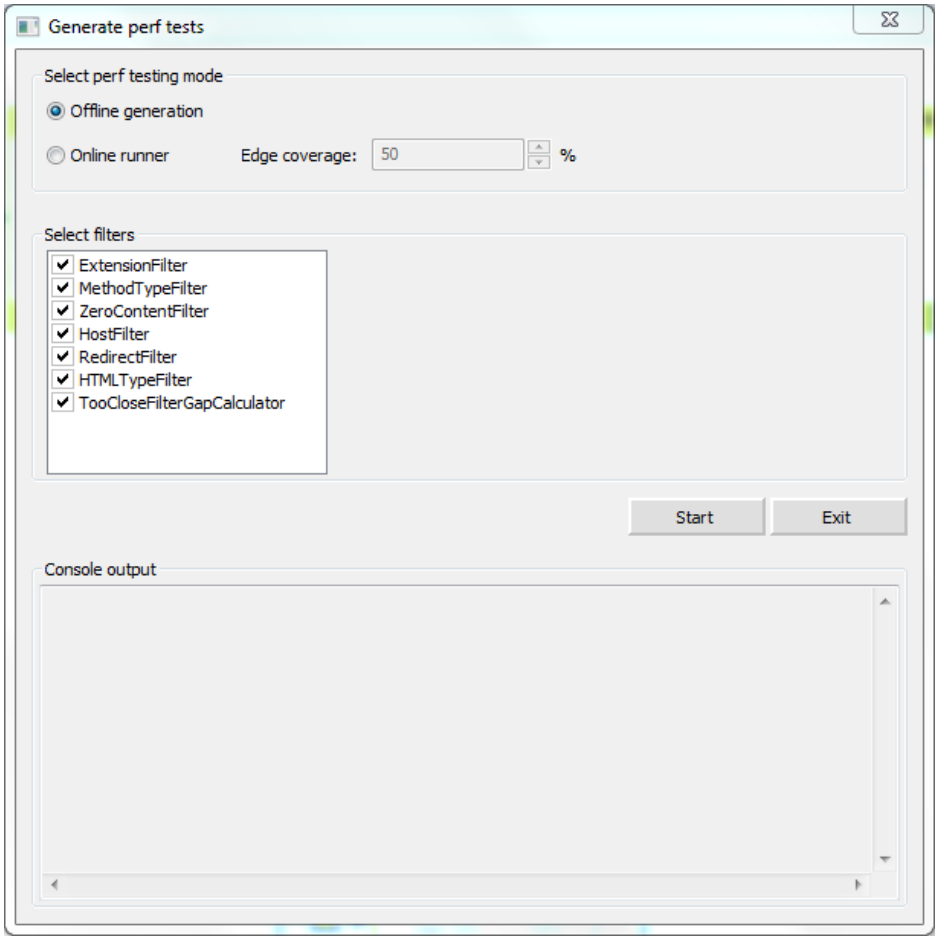
Siirtymän input-metodi liittyy edelliseen sivumalliin, esimerkiksi jonkin napin painallus sivulla, joka aiheuttaa seuraavan sivun latautumisen. Output-metodi liittyy seuraavaan sivumalliin, esimerkiksi varmennetaan, että siirtymä tapahtui oikealle sivulle tarkistamalla jonkin tekstin sisältö tai vertaamalla kuvakaappauksia.

Ennen suorituskykytestin generoimista on hyvä varmistua, että mallipohjainen testi varmasti toimii. Helpoin tapa varmistua toimivuudesta on tietenkin suorittaa testi siten, että käydään jokainen testihaara läpi eli asetetaan haarakattavuuden tavoitteeksi 100 %. Ajon keston voi määrittää myös ajallisesti. Testin ajoa voi seurata visuaalisesti QAutomaten avaamasta selainikkunasta.

4.1.4 Kuormitustestin luonti

Internetin selaaminen ja selaimen tiedon vaihto perustuu paketteihin. Pakettien lähettämiseen perustuu myös kuormitustesti. Pakettien sisältämistä tiedoista tärkeimpiä suorituskykytestauksen kannalta ovat palvelimen vastaus, URL-osoite, metodin tyyppi ja mahdolliset parametrit.

Suorituskykytestiin tarvittavat paketit generoidaan ajamalla luotu mallipohjainen testi QAutomatella. Ajon aikana työkalu kerää talteen XML-tiedostoon tarvittavat paketit siirtymiä varten. Nykyajan web-sivustoilla on paljon ylimääräistä suorituskykytestauksen näkökulmasta niin sanottua turhaa tavaraa ja web-selaimetkin lähettävät turhaa dataa. Käyttämällä suodattimia pakettien keräämisen aikana voidaan vähentää ei-toivottujen pakettien määrää ja oikeiden pakettien valitseminen testiä varten helpottuu. Tarkoituksena poimia pakettien seasta se paketti, joka laukaisee tarvittavat toiminnot kun käyttäjä siirtyy sivulta toiselle järjestelmässä.



KUVA 9. Suorituskykytestin generointiin liittyvät suodattimet valitaan ennen käynnistystä

Suodattimilla voidaan määritellä kerättäviä paketteja:

- estämällä ei-halutut tiedostotyypit
- estämällä ei-halutut metodit
- estämällä tyhjät paketit
- estämällä ei-halutut www-osoitteet
- selvittämällä pakettien uudelleenohjaukset
- estämällä ei-halutut sisältötyypit
- estämällä peräkkäiset liian lähekkäiset paketit

XML-tiedostoon kerättyjä paketteja voi muokata ja lisätä käsin suoraan tiedostoon. Kun tarvittavat paketit ovat kasassa, täytyy ne sijoittaa siirtymille QAutomate-työkalun mallipohjaisessa näkymässä. Kun kaikilla tarvittavilla siirtymillä on paketti määriteltynä, voidaan generoida suorituskykyadapteri. Adapterin perusteella MBPeT osaa ajaa suorituskykytestin ja mitata oikeita asioita. Adapteri on XML-tiedosto, jossa on määriteltynä kaikki toiminnot tarvittavine tietoineen.

```
<action desc="" name="open_application.open_application_url">
    <request allow_redirection="False" method="POST" type="normal">
        <url>https://optimizer.ekocoil.fi/default/index/setlocale</url>
        <parameters>
            <parameter row="1">
<![CDATA[locale=fi_FI]]>                    </parameter>
        </parameters>
        <payload/>
        <header>
            <token/>
        </header>
        <cookies/>
    </request>
    <response>
        <allowed_code/>
        <header>
            <token/>
        </header>
        <cookies/>
    </response>
</action>
```

KUVA 10. Yksi toiminto adapterista, jossa vaihdetaan sovelluksen kieli

Tarvittaessa toimintoja voidaan määritellä myös adapter.py python tiedostossa. Esimerkiksi, jos halutaan suorittaa suorituskykytesti usealla käyttäjällä yhtäaikaisesti, eikä palvelu hyväksy kuin yhden yhtäaikaisen kirjautumisen samalla tunnuksella. Tässä tapauk-

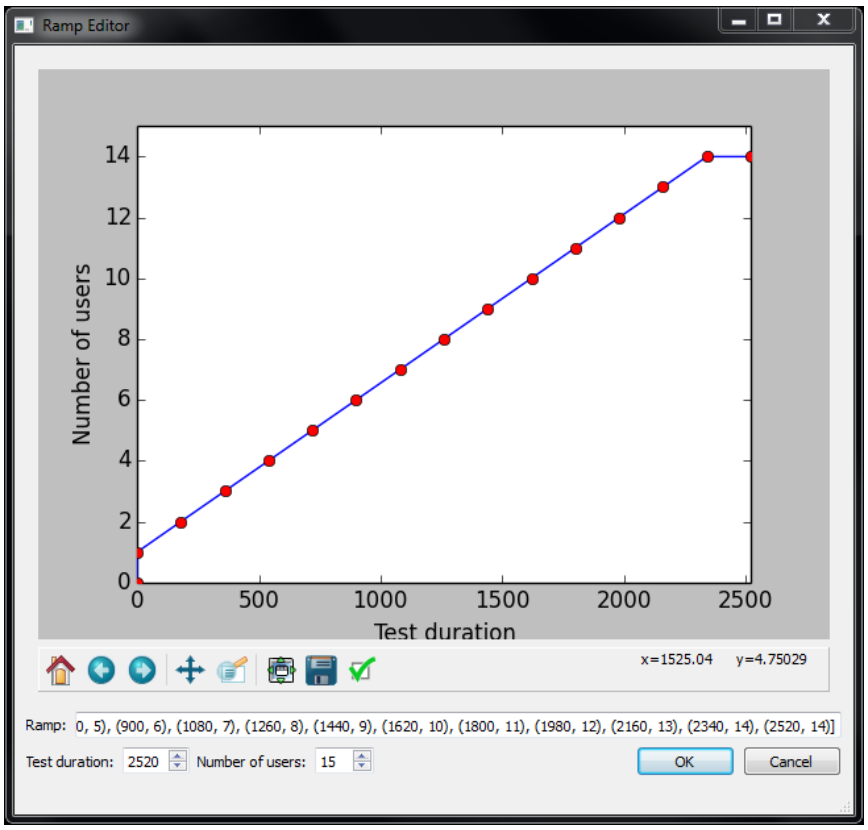
sessa kirjautumistoiminto kannattaa määritellä python adapterissa, näin voidaan määritellä ohjelma käyttämään aina eri käyttäjätietoja kirjautumiseen. Tässä tapauksessa täytyy tietenkin testaamiseen soveltuvia kirjautumistietojakin olla useampia.

4.2 Suorituskykytestin ajaminen

Lopullinen suorituskykytesti sujuu automaattisesti ilman erillistä testaajan huomiota. Ennen testin aloittamista täytyy määrittää testin pituus ajallisesti, käyttäjämäärät ja mahdolliset painotukset.

Testin keston on hyvä olla riittävän pitkä, jotta varmasti saadaan selville kuinka järjestelmä suoriutuu tehtävistään milläkin käyttäjämäärällä. Käyttäjämääriä ja niiden muutoskäyriä suunnitellessa kannattaa ottaa huomioon oletetut maksimikäyttäjämäärät. Tähän on kaksi syytä. Ei ole kannattavaa testata järjestelmää turhaan liian pienillä käyttäjämäärillä, eikä laittaa heti liian suuria käyttäjämääriä, jos ei ole tarkoituksena kaataa järjestelmää.

MBPeT -työkalulla pystyy suunnittelemaan testin kulun graafisesti. Testin suunnittelijan on mahdollista päättää käyttäjämäärät eri ajan hetkillä, sekä kuinka tiheään uusia käyttäjiä saapuu järjestelmään tai poistuu järjestelmästä.



KUVA 11. MBPeT-työkalun graafinen suunnittelutyökalu

Kun suorituskykytestin asetukset on saatu määriteltyä, käynnistetään master ja asetetaan slave etsimään masteria määritetystä ip-osoitteesta. Kun yhteys on luotu, lähettää master tarvittavat tiedot slavelle. Tämän jälkeen kuormitustesti on valmis aloitettavaksi.

Testi päättyy ja slave sammuu automaattisesti, kun määritetty testin kesto saavutetaan. Slave lähettää kaiken testistä saadun datan masterille, joka sitten generoi testiraportin testaajan saataville.

4.3 Tulokset

Tässä opinnäytetyössä oli ensimmäisenä testattavana Ekocoil Oy:n laskentasovellus ja toisena NurseBuddy-hoivaohjelmisto.

4.3.1 Ensimmäinen testi

Testattavana Ekocoil Oy:n laskentasovellus. Sovelluksessa käyttäjä valitsee haluamansa tuoteryhmän ja täyttää ilmestyvään lomakkeeseen haluamansa arvot. Tämän jälkeen sovellus laskee näitä arvoja parhaiten vastaavat tuotteet ja esittää ne käyttäjälle listattuna arvoineen. Tulosten esitysten jälkeen generoitu käyttäjä palaa takaisin tuoteryhmän valintaan.

Ajetussa testissä oli kolme laskentahaaraa, joilla kaikilla oli sama todennäköisyys. Testiä ajettiin 42 minuuttia ja alussa käyttäjiä oli yksi. Käyttäjämäärää nostettiin yhdellä kolmen minuutin välein, jolloin lopussa saavutettiin neljäntoista yhtäaikaisen käyttäjän raja. Lisäksi samanaikaisesti ajettiin loppukäyttäjän monitoroinnin takia yhdellä käyttäjällä QAutomate -työkalulla samaa testiä 42min.

Tuloksia tulkitessa havaittiin mittausarvojen väärentymää, johtuen laskimen toiminnasta. Laskimen käyttöä on rajoitettu rajaamalla sen yhtäaikaisten laskentaprosessien määrää. Laskimen ollessa jo varattuna generoitu käyttäjä ei pääsekään käyttämään laskinta, vaan saa virheilmoituksen. Tällöin mitattu laskentaan kulunut aika on vain noin sekunnin, kun muuten se olisi useita sekunteja. Laskimen suorituskykytuloksia tulkitessa tulee siis suodattaa pois epätavallisen pienet arvot.

Master Stats

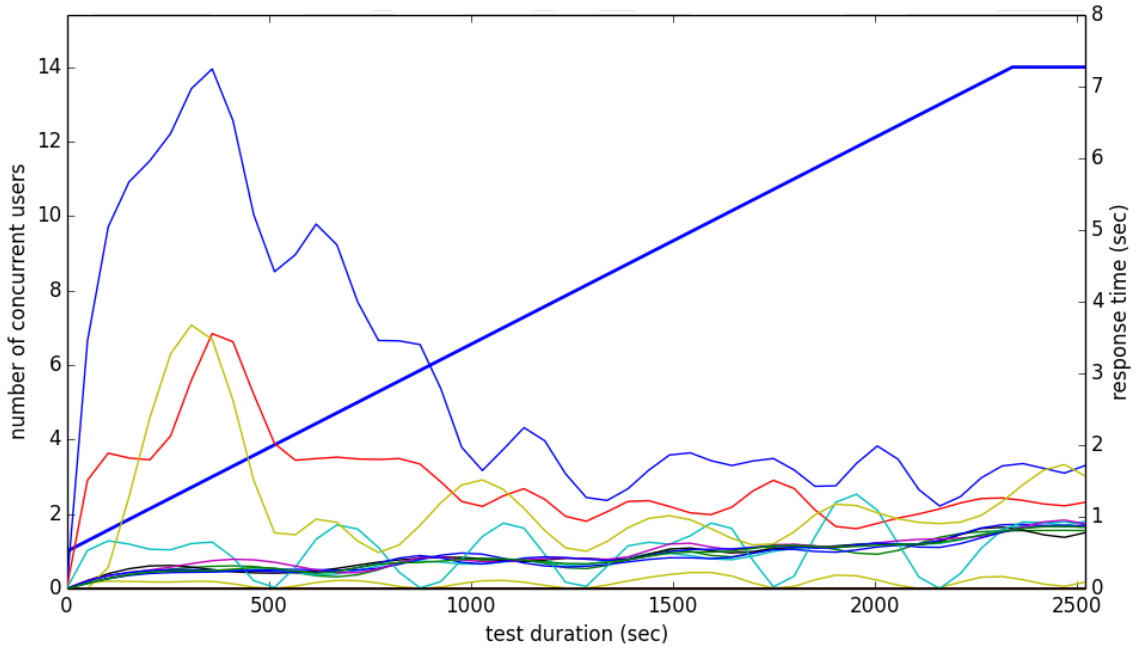
This test was executed at: 2016-15-02 08:28:33
Duration of the test: 42 min
Target number of concurrent users: 14
Total number of generated users: 7793
Measured Request rate (MRR): 3.09 req/s
Number of USER: 7793 (100.0)%
Average number of action per user: 556 actions
This test run ended at: 2016-15-02 09:10:43

AVERAGE/MAX RESPONSE TIME per METHOD CALL

	USER (100.0 %)		
Method Call	Average (sec)	Max (sec)	Executions #
ECH_LAUHDUTIN.WAIT_FOR_CALCULATION()	2.07	13.95	533
ILMANJAAHDYTIN_VALITTU.CHOICE_ECF()	0.55	1.47	485
ECH_NESTEJAAHDYTIN.WAIT_FOR_CALCULATION()	1.24	10.55	530
LAUHDUTIN_VALITTU.CHOICE_ECH()	0.54	1.78	533
VALINTA.CHOICE_NESTEJAAHDYTIN()	0.0	0.0	533
OPEN_APPLICATION.OPEN_APPLICATION_URL()	0.12	0.21	14
NESTEJAAHDYTIN.GO_TO_CALCULATIONS()	0.59	2.54	533
NESTEJAAHDYTIN_VALITTU.CHOICE_ECH()	0.55	1.7	530
VALINTA.CHOICE_ILMANJAAHDYTIN()	0.0	0.0	489
MENU_BAR.LOG_OUT()	0.0	0.0	0
VALINTA.CHOICE_LAUHDUTIN()	0.0	0.0	536
SISAANKIRJAUTUMINEN.LOGIN_IN()	0.74	1.11	14
ILMANJAAHDYTIN.GO_TO_CALCULATIONS()	0.6	2.37	486
ECF_ILMAJAAHDYTIN.WAIT_FOR_CALCULATION()	1.01	10.24	484
ACTION_B()	0.0	0.0	14
LAUHDUTIN.GO_TO_CALCULATIONS()	0.56	2.52	535
MENU_BAR.RETURN_TO_SELECTION()	0.56	1.88	1544

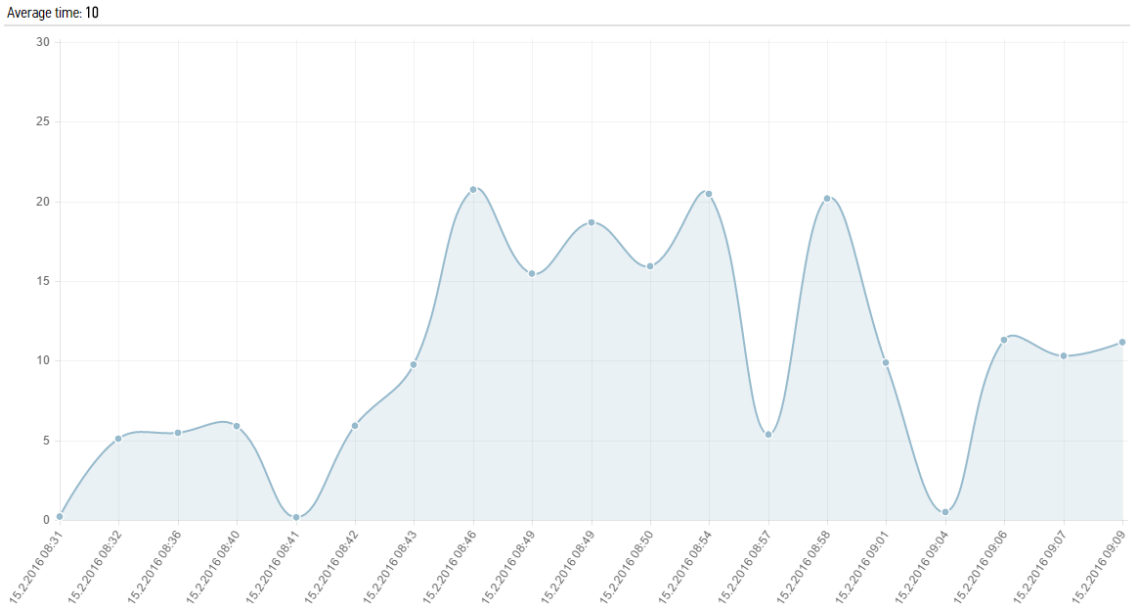
KUVA 12. MBPeT-työkalun raportista näkee yleisiä tietoja testin ajosta, sekä jokaisen toiminnon suoritusmäärän, keskiarvon ja maksimi suoritusajan sekunneissa

Kuvassa 12. nähtävistä toiminnoista ”.wait_for_calculation()” –loppuiset toiminnot ovat laskentatoimintoja. Keskiarvoja katsottaessa voidaan selvästi olettaa tuloksissa olevan väärentymää, kun verrataan maksimiaikoihin.

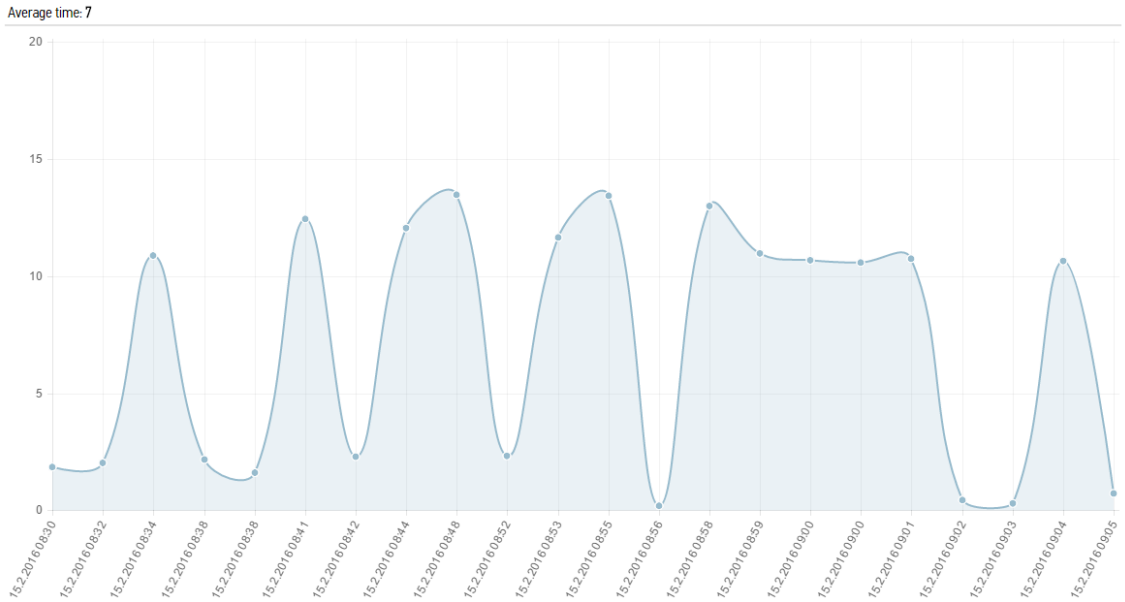


KUVA 13. MBPeT-työkalun raportin kuvaaja, jossa näkyy toimintokohtaiset vasteajat koko testin ajalta ja yhtäaikaiset käyttäjämäärät järjestelmässä

Kuvassa 13. sininen, punainen ja keltainen käyrä ovat laskimen toiminnot. Kuvaajasta nähdään alussa nousua vasteajoissa, mutta nopeasti käyrät muuttuvat laskusuuntaisiksi laskimen hylätessä laskentapyyntöjä.



KUVA 14. QAutomate web monitoring -raportista lauhduttimen laskennan tuloksia

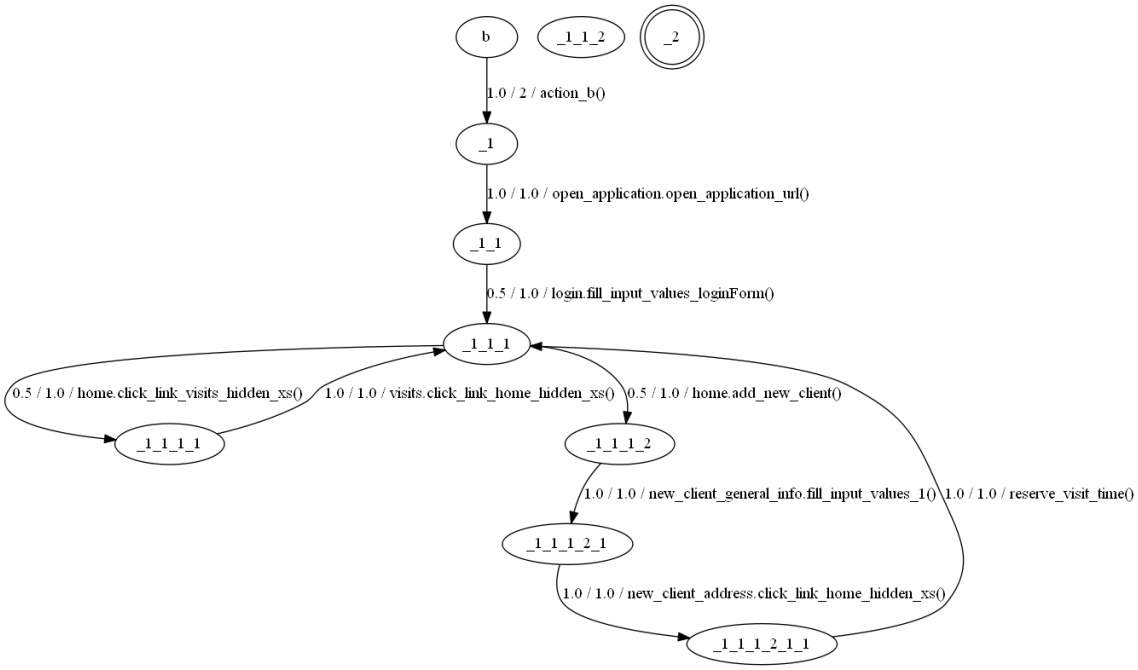


KUVA 15. QAutomate web monitoring -raportista nestejäähdyttimen laskennan tuloksia

Oletetusti kun käyttäjämäärä kasvaa, kasvavat myös vasteajat. Tämä on havaittavissa kuvista 14. ja 15. olevista kuvaajista. Loppua kohden vasteajat laskevat laskimen kieltäytymisistä johtuen.

4.3.2 Toinen testi

Toinen testattava ohjelmisto oli NurseBuddy-hoivaohjelmisto. Ohjelmassa hallitaan kotihoidon asiakaskäyntejä. Tähän työhön luodussa testitapauksessa on kaksi haaraa. Testihaarat näkyvät kuvassa 16. Ennen haarautumista kirjaudutaan sisään palveluun. Haaroista pidemmässä luodaan järjestelmään uusi asiakas ja onnistuneen lisäämisen jälkeen varataan uudelle asiakkaalle hoitajan käyntiaika ennen etusivulle palaamista. Lyhemmässä haarassa ladataan tiedot käynneistä ja palataan takaisin etusivulle.



KUVA 16. NurseBuddyn mallipohjainen testi

Suoritettu testi oli pituudeltaan 42min. Testin aikana nostettiin käyttäjämäärä tasaisesti nollasta kahteensataan.

###################### Master Stats ########################

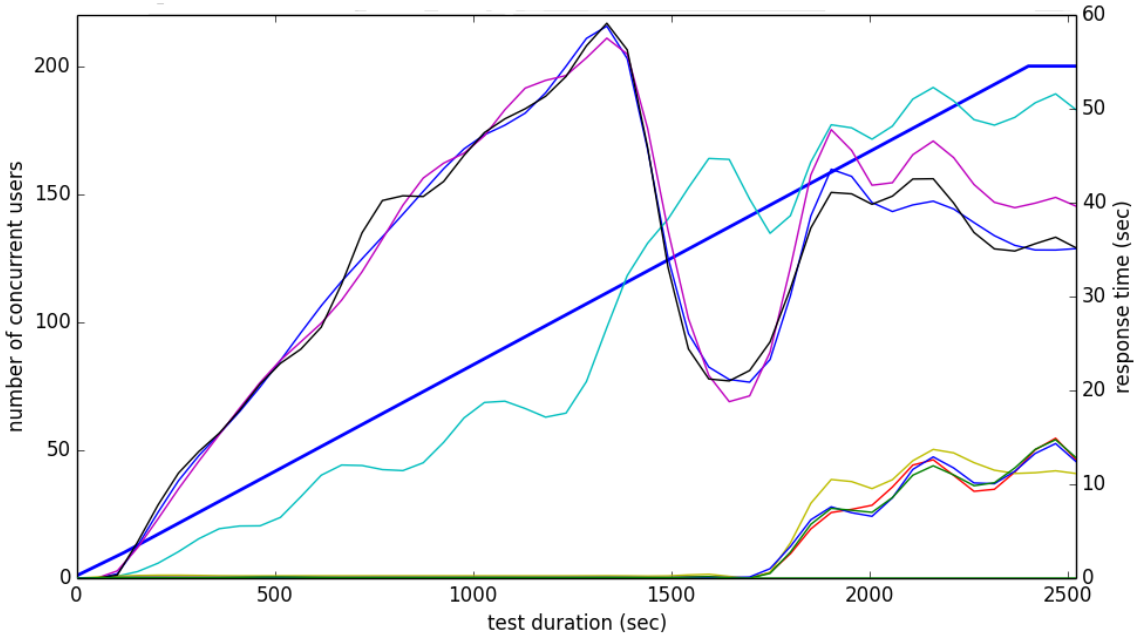
This test was executed at: 2016-15-03 08:52:11
Duration of the test: 42 min
Target number of concurrent users: 200
Total number of generated users: 13385
Measured Request rate (MRR): 5.31 req/s
Number of USER: 13385 (100.0)%
Average number of action per user: 66 actions
This test run ended at: 2016-15-03 09:35:17

######## AVERAGE/MAX RESPONSE TIME per METHOD CALL

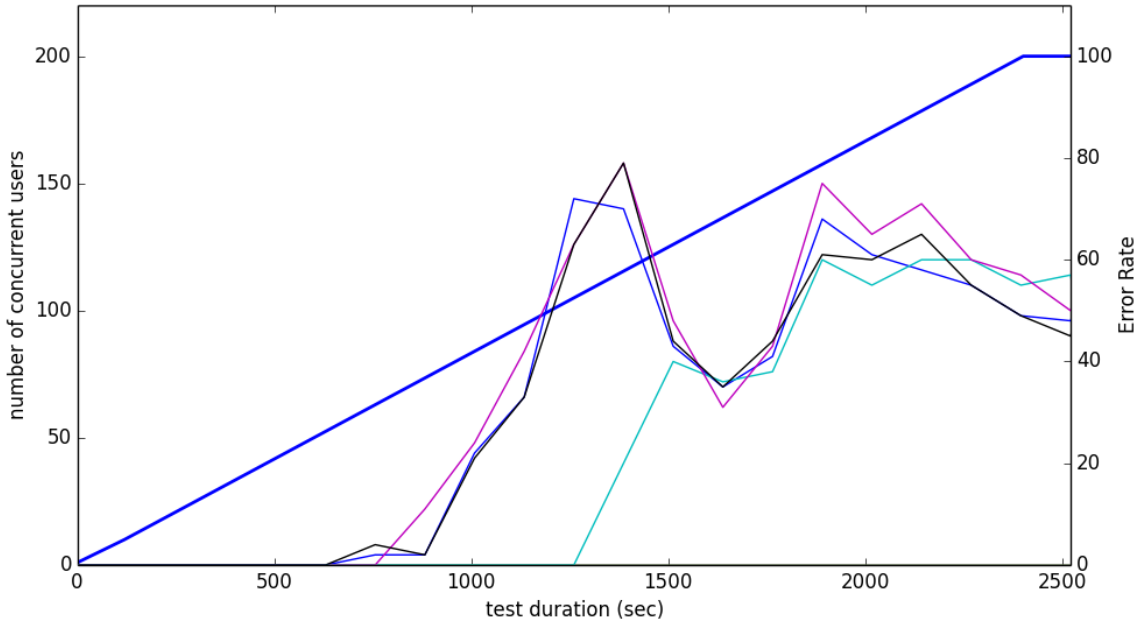
	USER (100.0 %)		
Method Call	Average (sec)	Max (sec)	Executions #
RESERVE_VISIT_TIME()	33.69	60.06	2093
ACTION_B()	0.0	0.0	200
VISITS.CLICK_LINK_HOME_HIDDEN_XS()	4.24	37.55	2090
LOGIN.FILL_INPUT_VALUES_LOGINFORM()	27.29	60.06	200
HOME.CLICK_LINK_VISITS_HIDDEN_XS()	35.67	60.06	2144
OPEN_APPLICATION.OPEN_APPLICATION_URL()	3.34	32.41	200
NEW_CLIENT_GENERAL_INFO.FILL_INPUT_VALUES_1()	33.74	60.06	2161
NEW_CLIENT_ADDRESS.CLICK_LINK_HOME_HIDDEN_XS()	4.28	38.5	2118
HOME.ADD_NEW_CLIENT()	4.22	39.32	2179

KUVA 17. NurseBuddy-testin yleistietoa

Kuvassa 17. nähtävien suoritettujen toimintojen maksimiajoista huomataan, että usealla toiminnolla se on 60,06s. Tähän huomioon kun yhdistetään testin virheprosentti, joka oli 20,52 % ja toimintolokista saadut tiedot virheiden syystä, aikakatkaisu, voidaan vetää johtopäätös, että palvelu on alkanut määrätyssä kohdassa hylkäämään jonossa olevia pyyntöjä.



KUVA 18. Käyttäjämäärä ja vasteajat



KUVA 19. Käyttäjämäärä ja virheet

Kuvista 18. ja 19. nähdään, että noin 100 yhtäaikaisen käyttäjän kohdalla palvelin on alkanut enemmän aikakatkaisemaan pyyntöjä. Noin 50 yhtäaikaisen käyttäjän kohdalla on

tullut jo ensimmäiset aikakatkaisut. Lopputestin ajan aikakatkaisuja on tullut melko tasaisesti.

Tällä testillä saatiin selville järjestelmän toiminnan rajoja käyttäjämäärien suhteen, sekä kuinka järjestelmä toimii suurien yhtäaikaisten pyyntöjen kohdalla. Testin perusteella järjestelmä jättää pyyntöjä jonkinasteiseen jonoon odottamaan, kunnes määrätyn ajan päästä pyyntö aikakatkaistaan, ellei pyyntö ole päässyt läpi.

4.3.3 Jatkotoimenpiteet

Jatkotoimenpiteet voivat kohdistua sekä testattuun järjestelmään, että tietenkin myös testausvälineistöön. Välineistön on kehityttävä samassa tahdissa testauskohteiden kanssa. Tässä tapauksessa testauksessa käytettyjä työkaluja on kehitetty kokoajan toimivamman työkalupaketin saamiseksi. Esimerkiksi jatkossa voidaan miettiä kuinka saadaan ratkaistuksi tässä suorituskykymittauksessa ilmennyt tulosten väärentyminen. Epätavallisen matalat arvot voitaisiin kenties suodattaa automaattisesti pois tai tehdä työkaluihin mahdollisuus, että määrättyä testin kohtaa yritetään suorittaa niin kauan, että se menee läpi. Tässä tapauksessa yritettäisiin käyttää laskinta niin kauan, kunnes sen käyttö sallitaan.

Itse suorituskykytestin tuloksien analysoinnista saadaan tärkeätä tietoa testatun järjestelmän toiminnasta ja suorituskyvystä. Jos järjestelmässä ilmenee erittäin hitaita ja epävakaita kohtia, voidaan niihin miettiä vaihtoehtoisia suoritustapoja tai esimerkiksi palvelinkapasiteetin nostoa. Näin voitaisiin saada parannettua suorituskykyä. Suorituskykytestauksen kautta tutustutaan koodiin lisää ja opitaan koodin tehokkuudesta. Useasti saman asian voi koodata monella eri tavalla. Tavoista osa ei ole kuitenkaan niin suorituskykyisiä eli tehokkaita kuin toiset.

5 POHDINTA

Työn tarkoituksena oli osoittaa mallipohjainen suorituskykytestaus sivumallien avulla toimivaksi ratkaisuksi kokonaisuutena. Asiaa lähestyttiin työssä suorittamalla suorituskykytestaus kahteen erilaiseen järjestelmään.

Molemmat testit saatiin suoritettua erilaisista hankaluuksista huolimatta ja molemmissa testeissä saatiin selville järjestelmän toimintatapoja useiden yhtäaikaisten pyyntöjen kohdalla. Tällaiset tiedot ovat tärkeitä testattavan järjestelmän omistajalle.

Testejä tehdessä ja ajaessa saatiin tietoa työkalujen kehitykseen liittyen ja saatiin parannettua käytettyjen työkalujen tukea, sekä saatiin ideoita jatkokehitykseen. Näin tulevissa projekteissa työkalut ovat valmiimpia ja toimivat monipuolisemmin erilaisten järjestelmien kanssa.

Tämän työn jälkeen on myös jatkossa helpompaa lähteä tekemään suorituskykytestausta saatujen tietojen ansiosta. Testien tekeminen kahteen erilaiseen järjestelmään antoi perspektiiviä erilaisille toteutustavoille. Lisäoppia tuli myös niin testausautomaatiosta kuin ohjelmistokehityksestä.

LÄHTEET

Tamres, L. 2002. Introducing software testing. Pearson Education Limited.

Patton, R. 2006. Software Testing Second Edition. Sams publishing.

QAutomate. Luettu 20.03.2016. http://qautomate.fi/

Testing Performance – Definitions. Luettu 23.01.2016. http://www.testingperformance.org/definitions/definitions

Continuity Central – Five nines: Chasing the Dream? Luettu 23.01.2016. http://www.continuitycentral.com/feature0267.htm

Model-Based Performance Testing Using the MBPeT Tool. Julkaistu 2013 Luettu 20.03.2016. http://tucs.fi/publications/attachment.php?fname=tAhAbTrPo13a.full.pdf