Oliver Armila

OHJELMISTOTESTAUKSEN TOTEUTUS

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittelyn koulutus

2020

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tekijä/Tekijät	Tutkintonimike	Aika
Oliver Armila	Tradenomi (AMK)	Marraskuu 2020
Opinnäytetyön nimi		
Ohjelmistotestauksen toteutus		25 sivua
Toimeksiantaja		
Softability Oy		
Ohjaaja		
Janne Turunen		

Tiivistelmä

Opinnäytetyön tavoitteena oli toteuttaa toimeksiantajan XReach-projektiin toimivat automaatio- ja suorituskykytestit. Automaatiotestissä hyödynnettiin suosittua automaatiotestaamisen työkalua, Robot Frameworkia. Suorituskykytestissä käytettiin suosittua Apache JMeter-työkalua. Työn toisena tavoitteena oli käsitellä muutamia ohjelmistotestaamisen osa-alueita sekä tutustuttaa lukija ohjelmistotestauksen tapoihin, tarpeisiin ja hyötyihin.

Työssä esitellyt testaamisen menetelmät ovat manuaali-, automaatio- ja suorituskykytestaus. Jokainen menetelmä käydään ensin läpi teoriatasolla, minkä jälkeen esitellään konkreettinen esimerkki. Manuaalitestauksen konkreettisena esimerkkinä on esitelty manuaalitestin suunnitelma.

Alussa lukijalle esitellään testaamista yleisesti ja miksi testaamista tarvitaan ohjelmistokehityksen tukena. Työssä käydään läpi neljä tärkeää syytä ohjelmistotestauksen tärkeydestä, ja esitellään laiminlyödyn tai huolimattoman testauksen seurauksia Tämän jälkeen käydään järjestelmällisesti läpi työssä esiteltävät testausmenetelmät. Opinnäytetyöraportin lopussa esitellään työn käytännön osuus.

Työn käytännön osuus koostuu kahdesta kokonaisuudesta, automaatio- ja suorituskykytestistä. Ensimmäinen osuus koostuu automaatiotestistä. Automaatiotestin tarkoituksena on vähentää testaajan toistuvaa työtä ja vapauttaa aikaa muihin tehtäviin. Toisessa kokonaisuudessa toteutettiin suorituskykytesti, jonka tarkoituksena on mitata sovelluksen suorituskykyä. Testin tuloksena selvisi, että järjestelmä ei pysty käsittelemään kovinkaan paljoa samanaikaista kuormaa. Tämän seurauksena järjestelmää optimoitiin siten, että se kestää enemmän samanaikaista kuormitusta.

Testien suorittamiseen valitut tekniikat soveltuvat testien suorittamiseen loistavasti niiden aktiivisen ylläpidon, kehityksen sekä laajan käyttäjäkunnan ansiosta. Jatkossa testien ylläpito on helppoa ja niitä voidaan hyödyntää uudestaan tarpeen vaatiessa. Testejä on helppo muokata eri asiakasympäristöjä varten.

Asiasanat

ohjelmistokehitys, testaus, automaatiotesti, suorituskykytesti

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Oliver Armila	Bachelor of Business Administration	November 2020

Thesis title

Implementation of software testing — 25 pages

Commissioned by

Softability Oy

Supervisor

Janne Turunen

Abstract

The main purpose of this thesis was to successfully implement automation and performance tests to Softability Oy's project XReach. Automated tests were implemented with a popular automation testing tool Robot Framework. The performance test was done with Apache JMeter. The second purpose of the thesis was to introduce software testing's practices, needs and benefits.

The methods introduced in the thesis involved manual, automated and performance testing. Each method was firstly addressed on theory level and after that a concrete example will be introduced. The concrete example for manual testing was a manual test plan.

The beginning of the thesis introduced software testing in general and why software testing is needed to support software development. The thesis introduced four important reasons of the importance of software testing, and also what may happen as a result of neglect of software testing. After this, all the testing methods used in this thesis were introduced systematically.

The hands-on part of the thesis included of two parts: automation and performance testing. The purpose of automation testing was to automate a simple test and reduce the manual work for the tester, therefore saving time for other work. The purpose of the performance test was to measure the performance capability of the software. As the result of this test, the system of the software was optimized so that it could handle more requests at a time.

The methods chosen for these tests suited the purpose excellently thanks to their active support, continuous improvements and large number of active users. In the future, the tests are easy to maintain, and they can be used again at the time of need. The tests are easy to modify for different customer environments.

Keywords

Software development, testing, automated testing, performance testing

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön ensisijaisena tavoitteena on toteuttaa automaatio- ja suorituskykytestejä Softability Oy:n kehittämään XReach-sovellukseen. Opinnäytetyön toissijaisena tavoitteena on havainnollistaa lukijalle ohjelmistotestaamisen tarpeellisuutta ja hyötyjä, sekä tutustua suosittuihin Robot Framework ja Apache JMeter-testityökaluihin.

Softability Oy on vuonna 1996 perustettu ohjelmistoyritys ja sen kotipaikkakunta on Helsinki. Softability työllistää tällä hetkellä noin 70 henkilöä. Suurin osa Softabilityn työntekijöistä työskentelee konsulttina eri asiakasyrityksille. Softabilityn vahvuus on henkilöstön laaja-alaisuus ja osaaminen monilla osa-alueilla. Softabilityn erityisosaamista on kyky yhdistää eri osaamisalueita, kun toteutetaan asiakkaalle lisäarvoa tuottavia ratkaisuja.

Raportin aluksi käsitellään yleisesti testaamiseen liittyviä seikkoja ja esitellään lukijalle ohjelmistotestaamista yleisesti. Lisäksi luvussa esitellään ohjelmistokehityksen vesiputousmalli. Luvussa perustellaan myös ohjelmistotestauksen tärkeyttä neljästä syystä.

Luvussa kolme käydään läpi erilaisia ohjelmistotestaukseen liittyviä testausmenetelmiä. Näitä menetelmiä ovat mm. manuaali-, automaatio-, ja suorituskykytestaus. Jokaisesta testauksen osa-alueesta esitetään myös esimerkki, jotta lukija saisi paremman käsityksen aiheesta.

Luku neljä sisältää esittelyn Softability Oy:n kehittämään XReach tuotteeseen sekä kuvaukset XReach projektiin opinnäytetyössä toteutetuista testeistä. Nämä testit ovat automaatioskripti opinnäytetyössä käsitellyllä Robot Frameworkilla sekä suorituskykytesti, joka on toteutettu Apache JMeterillä.

2 TESTAAMINEN YLEISESTI

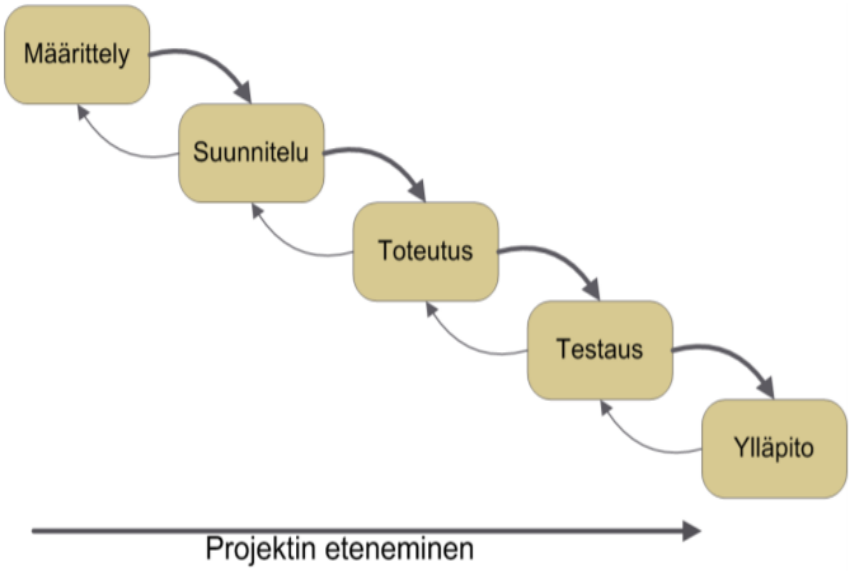
Ohjelmistotestaus on ohjelmistokehityksen osa-alue. Ohjelmistotestauksen tarkoitus on varmistaa, että tuotannossa oleva ohjelmisto, sovellus tai nettisivut toimivat suunnitellusti ja halutulla tavalla. Tarkoitus on varmistaa työn laatu. Jussi Pekka Kasurisen kirja ”Ohjelmistotestauksen käsikirja” (Kasurinen 2013, 10) kiteyttää sanoman hyvin: ”Varmistetaan että tehdään oikeaa tuotetta ja että tuote on tehty oikein.”

Testaus on epäkiitollista työtä. Hyvin testattu tuote ei puhuta käyttäjiä, eikä se juurikaan erotu joukosta. Hyvin testattu tuote on siis jopa standardi. Kuluttajat olettavat, että tuote on virheetön. Jos tuotetta ei ole testattu tai se on testattu huonosti tai huolimattomasti, on aivan varmaa, että se ei jää loppukäyttäjältä huomaamatta (Kasurinen 2013, 16).

2.1 Testaus ohjelmistokehityksessä

Usein ajatellaan, että testaus on vain yksi työvaihe ohjelmistokehityksessä (Kasurinen 2013, 12). Tämä ei kuitenkaan todellisuudessa toimi kovinkaan hyvin, koska vesiputousmallissa toteutus ja testaaminen suoritetaan eri vaiheissa. Testaaminen on suotavaa aloittaa jo aikaisessa vaiheessa, rinnakkain sovelluksen toteutuksen kanssa. Toteutusvaiheessa löytynyt virhe tai bugi saattaa myöhemmin olla todella vaikea korjata. Bugin löytyessä toteutuksen ollessa vielä alkuvaiheessa, voi sen korjaaminen olla huomattavasti helpompaa.

Kuvassa 1 esitellyssä vesiputousmallissa projekti on jaettu viiteen eri vaiheeseen, jotka ovat: määrittely, suunnittelu, toteutus, testaus ja ylläpito. Kuvaajassa on myös merkitty polut joita pitkin siirrytään seuraavaan vaiheeseen. Projektin viimeinen vaihe on ylläpito, joka käytännössä tarkoittaa sitä, että mitään uusia ominaisuuksia ei enää kehitetä, vaan että valmista tuotetta ylläpidetään. Tällaistä voi olla esimerkiksi sertifikaattien päivittäminen.



Kuva 1. Ohjelmistokehityksen vesiputousmalli.

Tässä mallissa testaus on vain yksi vaihe toteutuksen ja ylläpidon välissä. Vesiputousmalleja on kuitenkin erilaisia, mutta pääsääntöisesti mallit ovat 5-7 vaiheisia. Vesiputousmallin heikkous on se, että jokaisen vaiheen on päätyttävä ennen kuin seuraava voi alkaa. Vesiputousmallin hyödyllinen käyttäminen vaatii, että projektin alkuvaiheessa määritellyt vaatimukset ja tavoitteet ovat erittäin selkeitä, eikä ne tule juuri muuttumaan toteutuksen aikana. Vesiputousmalli vaarantaa projektin myös viivästyksille, koska konkreettisia tuloksia saadaan vasta projektin loppupuolella. Tällöin projektin aikana on vaikeaa hahmottaa ollaanko projektissa menossa oikeaan suuntaan. (Thinkingportfolio 2016.)

2.2 Miksi testaaminen on tärkeää?

Testaaminen on tärkeää useasta syystä. Kaikki syyt ovat tärkeitä ja niillä on yhtä suuri painoarvo, kun tarkastellaan testausta kokonaisuutena. Näitä syitä ovat mm. kustannussäästöt, käyttäjän turvallisuus, tuotteen laatu ja asiakastyytyväisyys. (Testdevlab 2018.)

Kustannussäästöistä on oiva esimerkki Jussi Pekka Kasurisen kirjassa ”Ohjelmistotestauksen käsikirja” (Kasurinen 2013, 11). Kasurisen kirjassa käsiteltävän tutkimuksen (Tassey 2002) mukaan yhdysvaltalaiset ohjelmistotalot menettivät yhteensä 21,2 miljardia dollaria huolimattomasti,

puutteellisesti tai vajavaisesti tehdyn ohjelmistotestauksen takia. Mukaan laskettaessa myös asiakkaille syntyneet tappiot ja menetykset, nousee luku jopa 59,5 miljardiin euroon. (Tassey 2002.)

Toinen tärkeä syy testaamiselle on käyttäjien turvallisuus. Sovelluksen on oltava turvallinen käyttää. Tämä tarkoittaa sitä, että sovelluksen käyttäjän tiedot, esimerkiksi pankille annetut tiedot, ovat turvassa. Henkilökohtaisten tietojen lisäksi myös fyysinen turvallisuus on esille nostettava seikka. Vuonna 1994 China Airlinesin Airbus A300 lentokone putosi taivaalta ohjelmistobugin seurauksena. Tämä ohjelmointivirhe maksoi valitettavasti 264 viattoman ihmisen hengen (Testdevlab 2018).

Kolmas syy testaamiselle on tuotteen laatu. Asiakkaalle halutaan pääsääntöisesti aina tarjota laadukas sovellus. Sovelluksen tulisi toimia luvatulla tavalla ja olla mukava käyttää, luoden hyvän käyttäjäkokemuksen, joka taas luo sovelluksen kehittävälle yritykselle hyvän maineen.

Neljäs tärkeä syy testaamiselle on käyttäjätyytyväisyys. Käyttäjätyytyväisyys ja laadukas tuote kulkee lähes käsi kädessä. Hyvin testattu sovellus on laadukas. Sillä voidaan ansaita käyttäjien, eli kuluttajien luottamus, mikä taas lisää yrityksen käyttäjämääriä ja sen myötä myös kassavirtaa. Sana kiirii nopeasti, eikä ole oikeastaan väliä onko sovellus todella hyvä vai todella huono. Ihmiset puhuvat ja kuluttajille muodostuu tietynlainen mielikuva sovelluksesta ja yrityksestä. Mikäli sovellusta ei ole käyttäjän mielestä mukava käyttää tai se ei toimi oikein on lähes varmaa, että käyttäjä poistaa sovelluksen ja etsii tilalle toimivamman vaihtoehdon. Mahdollisesti tämä vaihtoehto on kilpailevan yrityksen tuote. (Testdevlab 2018.)

3 KÄYTETYT TESTAUSMENETELMÄT

Ohjelmistotestauksen menetelmiä ja tapoja on useita. Tässä opinnäytetyössä perehdytään niistä muutamaan. Näitä menetelmiä ovat manuaali-, automaatio-, ja suorituskykytestaus.

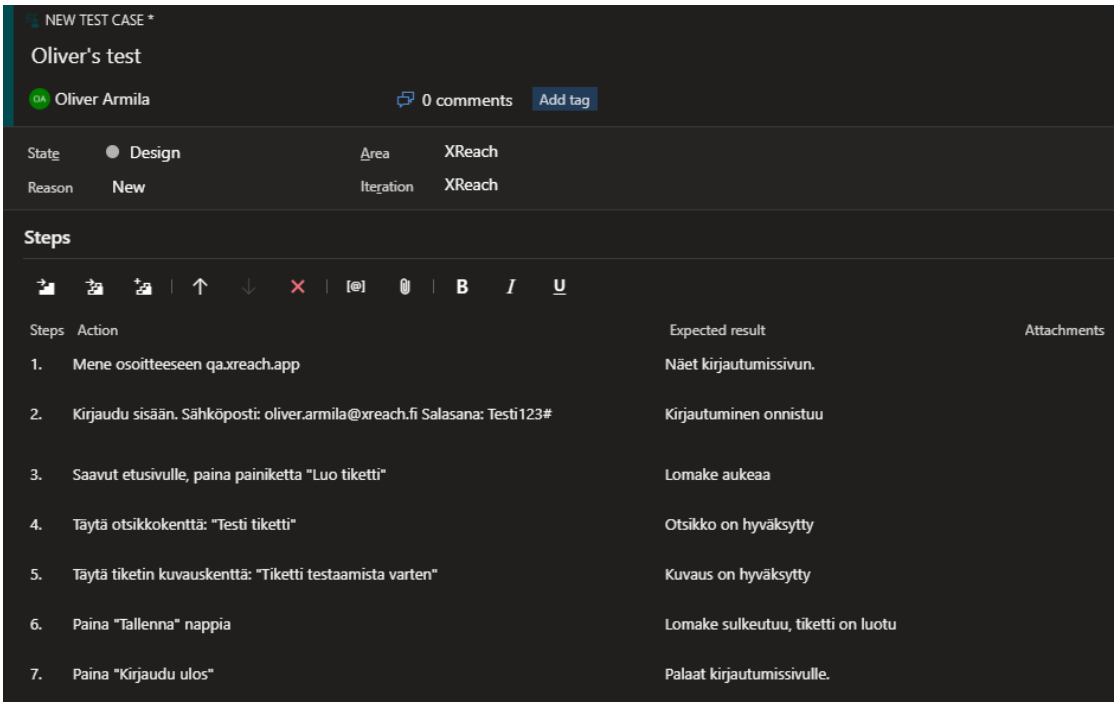
Kaikki testaamisen osa-alueet ovat tärkeitä, eikä niitä tulisi vertailla toisiinsa tärkeyden puolesta. Kaikilla osa-alueilla on oma tarkoituksensa, mutta

loppujen lopuksi niillä on sama tavoite: varmistaa, että tuote on loppukäyttäjälle laadukas ja turvallinen.

3.1 Manuaalitestaaminen

Manuaalitestaus on testaamista, jossa testit suoritetaan manuaalisesti testaajan toimesta. Ensin testaajan on tiedettävä ja ymmärrettävä, miten ohjelmiston tulisi toimia, eli mitkä ovat kriteerit ja oletusarvot. Manuaalitestaaminen voidaan aloittaa heti, kun ohjelmistosta on saatavilla jokin valmis ominaisuus. Testaaminen aloitetaan luonnollisesti suunnittelemalla testit. Testin suunnitteluvaiheeseen liittyviä yleisiä kysymyksiä ovat mitä testataan, mitä toimintoja se vaatii ja montako erilaista testiä tarvitaan. Tyypillisesti manuaalitesti kirjoitetaan ensin ns. ”steppeihin”, eli askeliin.

Manuaalitesti voi olla esimerkiksi seuraavassa kuvassa (kuva 2) esitellyn Microsoft Azure DevOpsissa toteutetun testitapauksen kaltainen.



Kuva 2. Azure DevOpsissa toteutettu esimerkki manuaalitestistä.

Kuvassa on toteutettu yksinkertainen manuaalitesti. Testissä kirjaudutaan sovellukseen tietyllä käyttäjätunnuksella, luodaan tiketti ja kirjaudutaan ulos. Manuaalitestiä kirjoittaessa on tärkeää suunnitella, mitä halutaan testaajan

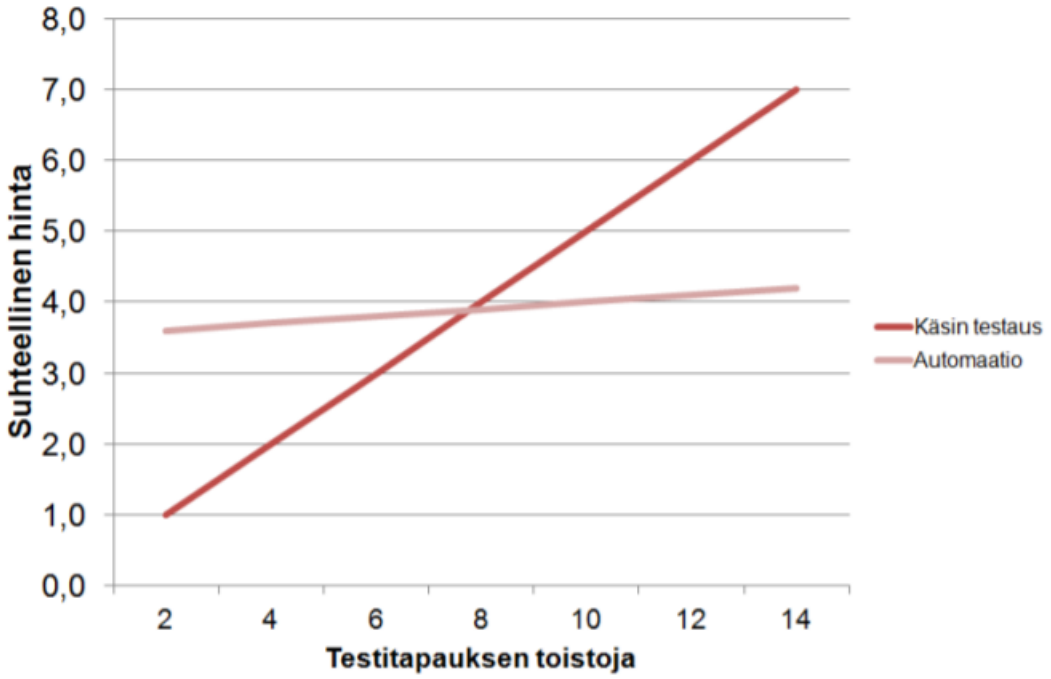
tekevän ja mitä oletetaan tapahtuvan. Jos testaaja suorittaa testin ja tulos on jotain muuta kuin testin oletettu tulos, on testi silloin epäonnistunut. Testin epäonnistuessa, testiä tulee korjata, tai mikäli järjestelmässä on bugi, tulee siitä raportoida kehittäjille.

3.2 Automaatiotestaus

Automaatiotestaaminen on verrattavissa manuaalitestaukseen, mutta testit ovat automatisoituja. Testiautomaatio ei kuitenkaan ole manuaalitestaamisen korvaaja, vaan pikemminkin sen täydentäjä. Automatisoidut testit tarkoittavat sitä, että testaaja on koodannut testit, jotka osaavat itsenäsesti suorittaa niille määritetyt toimenpiteet. Automaatiotestejä hyödynnetään esimerkiksi integraatiotestaamisessa. Tällöin automatisoidut testit ajetaan automaattisesti aina kun versionhallintaan tulee uutta koodia. Tämän tarkoitus on vapauttaa testaaja muihin tehtäviin (Kasurinen 2013, 49). Näin ollen testaaja ei joudu jokaisen päivityksen jälkeen suorittamaan samoja toistuvia ja yksinkertaisia testejä kuten sisäänkirjautumista tai salasanan vaihtoa. Hyvä testiautomaatiotyökalu on esimerkiksi työssä myöhemmin esitelty Robot Framework.

Automaatiotestien hyöty korostuu, kun testaajilla on testejä, jotka suoritetaan samanlaisina monta kertaa. Automaatiotestaamisen aloittamisen kynnyksenä on kuitenkin itse testien koodaaminen. Testien koodaaminen vie hieman aikaa, mutta pitkässä juoksussa automatisointi on ajan säästämisen kannalta erittäin kannattavaa. Automaatiotestaaminen myös vähentää inhimillisiä virheitä, joita saattaa syntyä testin suorituksen aikana. Tietokone suorittaa testin aina samalla tavalla, mutta ihminen saattaa tehdä virheitä ja esimerkiksi yksinkertaisesti unohtaa testata jonkun ominaisuuden tai toiminnallisuuden.

Kuvassa 3 on esitelty automaatiotestaamisen hinnan ja toistojen määrän suhdetta, milloin automaatiotestejä kannattaa käyttää ja milloin ei.



Kuva 3. Testiautomaation kustannuskäyrä.

Kuvaajaa tarkastelemalla voidaan todeta, että automaatiotestaaminen ei ole kovinkaan hyödyllistä eikä taloudellisesti järkevää, kun testitapaus toistetaan alle kahdeksan kertaa. Tämä johtuu etenkin siitä, että automaatiotestien toteuttaminen on kallista. Kun testit on kerran toteutettu eli koodattu, eivät ne enää juurikaan tuo lisäkustannuksia, vaikka niitä käytettäisiin tuhat kertaa (Kasurinen 2013, 78). Manuaalitestin toistaminen tuhat kertaa taas on todella kallista niin ajallisesti kuin rahallisestikin.

3.3 Robot Framework

Robot Framework on sovellusten automaatiotestaukseen tarkoitettu avoimen lähdekoodin ohjelmisto, jota pidetään jopa yhtenä parhaimmista testauksen työkaluista. Ohjelmisto on kehitetty Python-ohjelmointikielellä, ja sai alkunsa vuonna 2004 suomalaisen Pekka Klärckin diplomityöstä (Kotilainen, 2018). Robot Framework julkaistiin vuonna 2008 avoimen lähdekoodin ohjelmistona Nokia Networksin toimesta (Robot Framework s.a.). Vuonna 2015 vastuu projektin kehittämisestä siirtyi Robot Framework foundation -säätiölle. Robot Frameworkin käyttämisestä automaatiotestauksessa on tullut Suomessa jopa standardi.

Robot Frameworkilla voidaan testata esimerkiksi web-käyttöliittymää, mobiilisovellusta tai tietokantayhteyksiä. Testien tekeminen Robot

Frameworkilla on yksinkertaisten komentojen ansiosta helppoa, mikä madaltaa käyttäjien kynnystä aloittaa työkalun käyttäminen. Robot Frameworkia voi opetella käyttämään myös henkilöt, joilla ei ole ohjelmointitaustaa.

Robot Frameworkin syntaksi on helposti ymmärrettävää, koska avainsanat ovat selkeitä englanninkielisiä sanoja. Robot Frameworkin toiminnallisuutta on mahdollista lisätä kattavilla ulkoisilla kirjastoilla ja työkaluilla.

Seuraavassa kuvassa (Kuva 4) on esitetty opinnäytetyötä varten tehty esimerkki Robot Frameworkista. Esimerkissä robotti avaa selaimen, navigoi sivulle www.xamk.fi ja odottaa siellä hetken. Hetken odottelun jälkeen robotti sulkee selaimen. Tämä on erittäin yksinkertainen esimerkki ja sen tarkoituksena on vain ja ainoastaan tutustuttaa lukija Robot Frameworkin syntaksiin.

```
*** Settings ***
Documentation     Opinnäytetyötä varten tehty demo.
Library           SeleniumLibrary

*** Variables ***
${XAMK URL}       https://www.xamk.fi/   #Osoite jolle robotti navigoi.
${BROWSER}        Chrome                 #Selain jota robotti käyttää.

*** Test Cases ***
Visit XAMK homepage
    Open Browser To XAMK    #Keywordeista otettu avainsana, näitä voi olla ja usein onkin useampia.


*** Keywords ***
Open Browser To XAMK
    Open Browser    ${XAMK URL}    ${BROWSER}   #Avaa selaimen osoitteeseen xamk.fi ja käyttää selaimena Chromea.
    Title Should Be    Etusivu - XAMK           #Tarkistaa että sivun otsikko on oikea.
    Maximize Browser Window                     #Laajentaa selaimen ikkunan suureksi.
    Sleep   10                                  #10 sekunnin tauko. Ei tee mitään.
    Close Browser                               #Sulkee selaimen.
```

Kuva 4. Esimerkki Robot Framework-skriptistä.

Tässä esimerkissä Robotille on annettu vain muutama yksinkertainen käsky. Ensiksi sille on alustettu muutamat muuttujat, jotka tässä tapauksessa ovat ”${XAMK URL}” ja ”${BROWSER}”. Muuttujien käyttäminen ei ole pakollista, mutta erittäin suositeltavaa laajemmissa testeissä. ”Test Cases” eli testitapaukset ovat itsessään se kohta, joka robotilla ajetaan. Robotti suorittaa siinä annetut avainsanat, jotka ovat esitelty ”Keywords” kohdan alapuolella. Tässä testissä on siis avainsana, jonka nimi on ”Open Browser To XAMK”. Avainsanan alapuolella on eritelty tarkemmin mitä se sisältää. Avainsanoja voi

olla ja usein onkin paljon enemmän kuin yksi. Testitapaus koostuu siis testaajan kirjoittamista avainsanoista.

3.4 Suorituskykytestaus

Suorituskykytestaus on sovelluksen rajojen etsimistä ja sovelluksen toiminnallisuuden varmistamista luvatulla tavalla myös rasituksen alaisena. Suorituskykytestausta tehdään, jotta saadaan käsitys siitä miten paljon kuormitusta järjestelmä kestää ja missä vaiheessa käyttäjille alkaa satelemaan virheilmoituksia. Suosittu suorituskykytestaustyökalu on esimerkiksi Apache JMeter. Suorituskykytestaus on todella tärkeä osa sovelluskehitystyötä. Sovellus voidaan julkaista asiakkaille ilman testaamista, mutta ennemmin tai myöhemmin ongelmat löytävät perille, joten on parempi suorittaa testaaminen ajoissa ennen sovelluksen julkaisua.

Suorituskykytestauksessa on muutamia alalajeja. Näitä ovat esimerkiksi kuormitus- ja stabiliteettitestaus. Kuormitustestauksessa tutkitaan järjestelmän kykyä käsittelemään samanaikaista kuormaa. Esimerkiksi sitä, pystyykö 500 käyttäjää kirjautumaan sovellukseen samaan aikaan. Stabiliteettitestauksessa taas järjestelmälle annetaan pitkäkestoisempi kuorma. Stabiliteettitestauksen tavoitteena on selvittää, kestääkö järjestelmä pitkäaikaista rasitusta, toimiiko se samalla tavalla testin ensimmäisenä ja viimeisenä päivänä. Stabiliteettitestaukset voivat kestää jopa kuukausia (Vuori 2010).

Lyhyesti sanottuna suorituskykytestaamista tehdään, jotta tuotteen loppukäyttäjä on tyytyväinen. Loppukäyttäjälle tärkeintä on koettu palvelutaso. Tällaisia ovat esimerkiksi vasteaika, yhteyden saaminen palveluun, sekä palvelun saatavuus aina, kun loppukäyttäjä niin haluaa.

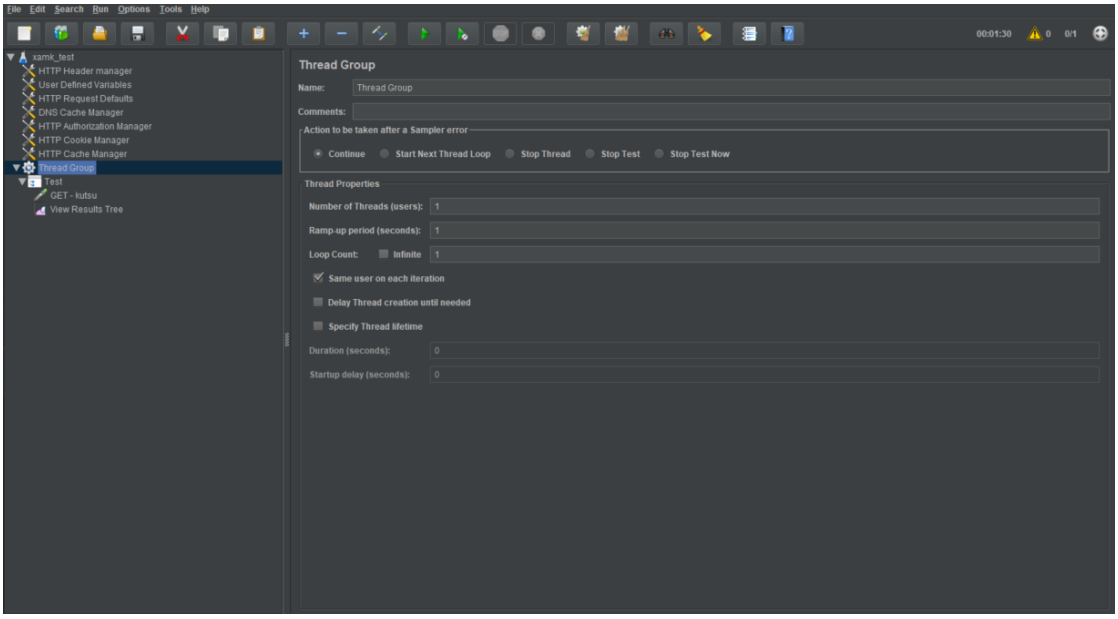
3.5 Apache JMeter

Apache JMeter on avoimen lähdekoodin java-pohjainen työkalu kuormitustestien suorittamiseen ohjelmiston suorituskyvyn mittaamiseksi. Apache JMeteriä voidaan käyttää esimerkiksi korkean volyymin käyttäjäsimulointiin, mikä tarkoittaa käytännössä sitä, että JMeter simuloi käyttäjiä ja suorittaa kaikilla käyttäjillä niille ennalta määritetyt käskyt. Nämä käskyt voivat olla mm. POST, GET, PUT tai DELETE -pyyntöjä palvelimelle

(Apache JMeter 2020). Tärkeää on ymmärtää, että JMeter ei ole selain, vaan enemmänkin työkalu, joka työskentelee ns. piilossa, konepellin alla. JMeter ei aja sivuston javascript-koodia, eikä edes renderöi HTML-sivuja samoin kuin selain.

Apache JMeterin laajan käyttäjäkunnan sekä aktiivisen kehitystyön ansiosta netti on täynnä tietoa ja ohjeita, joilla kaikki aiheesta kiinnostuneet voivat päästä alkuun.

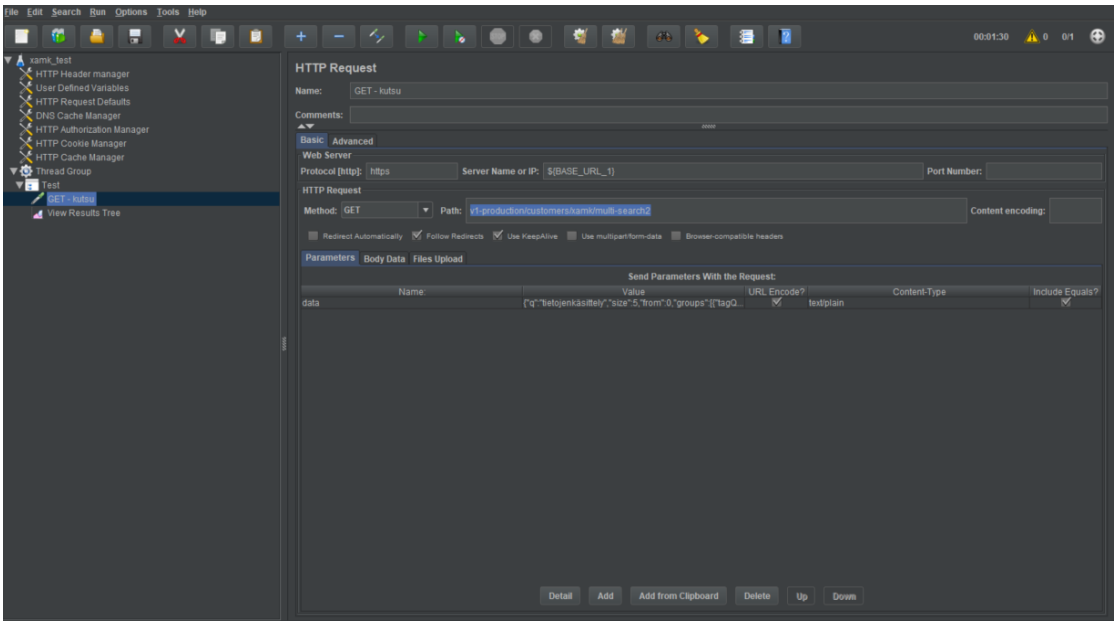
Seuraavassa kuvassa (Kuva 5) on Apache JMeter-testin ”Thread group”, jossa määritellään simuloitavien käyttäjien määrä, monestiko testi toistaa itseään ja mikäli haluaa, niin voi sille myös antaa aikarajoituksia.



Kuva 5. Apache JMeterin ”Thread group”-näkymä.

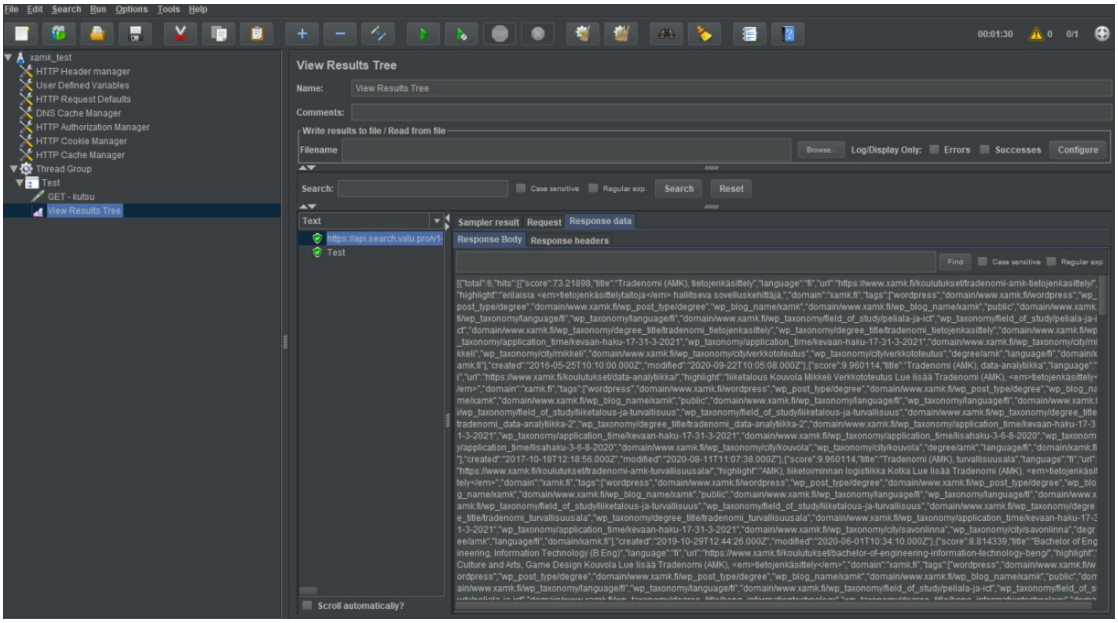
Tässä esimerkissä testille on annettu käyttäjien määräksi yksi, ”Ramp-up period” arvoksi yksi, eli kestää yksi sekunti, että yksi testikäyttäjä on generoitu ja valmiina suorittamaan testissä määritetyt vaiheet. ”Loop count”, eli silmukoiden määrä on yksi, joten käyttäjä suorittaa kierroksen vain kerran. Testissä simuloitu käyttäjä tekee Xamkin kotisivuilla haun hakusanalla ”tietojenkäsittely”.

Seuraavassa kuvassa (Kuva 6) on esitetty HTTP Request-käsittelijä. Käsitteljä lähettää vastaanottavalle palvelimelle tässä tapauksessa GET-pyynnön.



Kuva 6. Apache JMeterin “HTTP Request”-näkymä.

Simuloitu käyttäjä pyytää vastausta osoitteesta ”https://api.search.valu.pro/v1-production/v1-production/customers/xamk/multi-search2” lähettämällä sinne tietyn tiedon. Vastauksena käyttäjä saa vastauksen hakuunsa JSON-muodossa (Kuva 7).



Kuva 7. Apache JMeterin ”Results tree”-näkymä

Apache JMeterin ”Results tree” näyttää pyynnön vastauksen. Vastausta voidaan tarkastella testin jälkeen kaikessa rauhassa. Saatavilla olevia tietoja ovat esimerkiksi, kauanko testiin kului aikaa, mihin kellonaikaan se tapahtui, paljonko oli viivettä palvelimella, kauanko meni aikaa yhdistää vastaanottavaan palvelimeen sekä tietysti se, mitä dataa GET-pyyntö palautti. Tässä tapauksessa vastauksen JSON-objekti on niin pitkä, ettei sitä kannata esitellä tässä työssä.

Konkreettisena esimerkkinä suorituskykytestistä tässä tapauksessa voisi olla esimerkiksi Xamkin kotisivujen haku -toiminallisuuden testaaminen siltä osin, miten sivusto käyttäytyy, kun esimerkiksi 1000 käyttäjää suorittavat haun samaan aikaan. Suorituskykytestillä voidaan selvittää, hidastuvatko vastausajat tai vastaako palvelin pyyntöihin enää ollenkaan, vai suoriutuuko se kaikista pyynnöistä ilman ongelmia.

4 TESTIEN TOTEUTTAMINEN

Softability Oy:n kehittämä XReach on asiakkaan tarpeisiin räätälöitävä etätukialusta sekä tikettijärjestelmä. XReachissä on ns. etäkäyttäjiä sekä asiantuntijoita. Etäkäyttäjä luo tiketin johon asiantuntijat ottavat kantaa ja auttavat ongelmatilanteen ratkaisemisessa. Etäkäyttäjä sekä asiantuntija voivat tiketin ratkaisemisen yhteydessä hyödyntää videopuheluita, chat-keskustelua sekä jakaa tiedostoja. XReachia voidaan hyödyntää esimerkiksi etähuollon, teknisen asiakastuen, asennuksien sekä koulutuksen tarpeisiin. Etätukiratkaisun tarkoituksena on parantaa työn laatua, nopeuttaa sen valmistumista sekä säästää asiakkaalta huomattavasti matkustamisesta koituvia kuluja (Softability, XReach). XReach tuo asiantuntijan nopeasti ja vaivattomasti sinne, missä häntä juuri sillä hetkellä tarvitaan.

XReach-sovellukselle ei oltu vielä ennen tätä opinnäytetytöä tehty suorituskykytestejä eikä automaatiotestejä. Opinnäytetyössä projektiin toteutettiin testit, jotta saatiin selville sovelluksen rajat, ja erityisesti se, minkä verran kuormitusta sovellus kestää samanaikaisesti. Toteutin suorituskykytestit aiemmin mainitulla Apache JMeter työkalulla. Automaatiotestit toteutin Robot Framework työkalulla.

4.1 Robot Frameworkin testin esittely

Robot Frameworkilla toteutin automatisoidun testin, joka kirjautuu järjestelmään ennalta määritetyllä käyttäjällä, luo sinne tiketin tai tekee haun tiketeille, jonka jälkeen kirjautuu ulos. Automaation hyöty tässä on se, että testin voi laittaa ajamaan 1000 kertaa peräkkäin, jolloin kaikkien 1000 testin ajon jälkeen voidaan todeta, että järjestelmä toimii tai ei toimi luotettavasti, riippuen testin tuloksista. Seuraavassa kuvassa (Kuva 8) on kuvankaappaus minun toteuttamasta Robot Framework-skriptin asetuksista ja testitapauksista.

```
*** Settings ***
Resource    ../resources/resources.resource

*** Variables ***
#These "JSONs" are actually strings, until converted to JSON later in the script
${JSONSEARCH}    {"a": 10, "b": ["0", "50", "${EMPTY}", ""]}
${JSONLOGIN}    {"a": 12, "b": ["ont_testikayttaja@xreach.fi", "Salasana123#", "true"]}
${JSONCREATETICKET}    {"a":2,"c":[{"h":"${EMPTY}","i":"Description","a":"8b3e2d952d1247fd864ec931e09313aa"}],"b":["8b3e2d952d1247fd864ec931e09313aa"]}
${JSONLOGOUT}    {"a": 20}
${VERSION}    3.6.8


*** Test Cases ***
Search for tickets
    [Tags]    search
    Login
    FOR    ${i}    IN RANGE    0    5
        Search
    END
    [Teardown]    Logout

Create a new ticket
    [Tags]    ticket
    Login
    FOR    ${i}    IN RANGE    0    5
        Create ticket
    END
    [Teardown]    Logout
```

Kuva 8. Robot Frameworkilla toteutettu automaatiotestin asetukset ja testitapaukset.

Tiedoston alussa on Robot-skriptin asetuksia. Tämän testin asetuksissa ei ole kerrottu muuta, kuin että missä *.resource*-tiedosto sijaitsee. Jos käytössä olisi ulkoisia kirjastoja, voisi ne tuoda mukaan tässä kohdassa skriptiä. Tällä kertaa ulkoiset kirjastot on tuotu mukaan *.resource*-tiedostossa.

Seuraavaksi alustetaan muutamia muuttujia, joita testissä käytetään useammin. Muuttujien alustaminen alkuvaiheessa helpottaa koodin kirjoittamista sekä lukemista myöhemmässä vaiheessa. Tässä tapauksessa muuttujien data on tekstijonoja, joita XReach järjestelmä lähettää palvelimelle POST-pyyntöinä. Muuttujat muunnetaan myöhemmin JSON-muotoon.

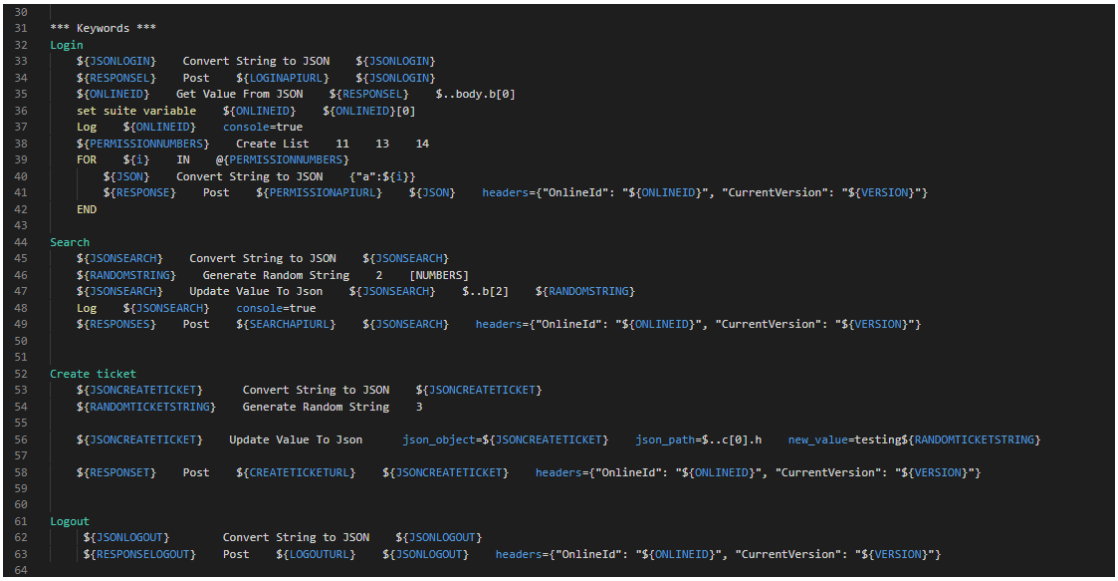
Asetusten ja muuttujien jälkeen tulee ensimmäinen testitapaus. Testitapauksen nimi on ”Search for tickets”. Koska samassa tiedostossa voi olla useita testitapauksia eikä jokaisella ajolla haluta välttämättä ajaa kaikkia

testitapauksia, on ”tags” hyödyllinen apuväline. Tagsin avulla voidaan ajaa vain yksi tietty testitapaus, tai vaikka kaksi tiettyä. Ilman tagseja robotti ajaa koko tiedoston, eli kaikki tiedostossa olevat testitapaukset. Testitapauksesta löytyy avainsana ”Login”. Login on minun luoma avainsana eli keyword, jonka sisältö näkyy myöhemmin esiteltävässä ”Keywords” osiossa. Loginin jälkeen seuraa for-silmukka. Silmukka ajetaan, ja avainsana ”Search” ajetaan tässä tapauksessa 5 kertaa, koska jälkimmäinen ”In Range” arvo on 5. Silmukka pyörii niin monta kertaa kuin ”${i}” on suurempi kuin 0 mutta yhtä suuri tai pienempi kuin 5. Silmukasta päästään pois sisäänrakennetulla, valmiilla avainsanalla ”END”. Seuraava on *[teardown] Logout*. Tässä tapauksessa ”teardown” on Robottiin sisäänrakennettu metodi, mikä tarkoittaa sitä, että avainsana ”Logout” ajetaan aina ennen testin lopettamista, vaikka testi epäonnistuu tai onnistuu. Esimerkiksi jos testi epäonnistuu for-silmukan epäonnistumisen vuoksi (hakuja ei saada suoritettua), kirjautuu Robotti ulos järjestelmästä ennen ajon varsinaista lopettamista. Avainsana ”Logout” on selitetty myöhemmässä vaiheessa, jossa käydään läpi avainsanat.

En selosta tarkemmin testitapauksen ”Create a new ticket” tapahtumia, koska ne ovat oikeastaan täysin samat kuin edellä mainitun testitapauksen tapahtumat. Ainoa eroavaisuus on for-silmukan sisällä ajettava avainsana. Toisessa testitapauksessa haetaan tikettejä ja toisessa luodaan tikettejä.

Kaikkiaan testi koostuu siis kahdesta testitapauksesta. Testitapauksia voidaan ajaa joko rinnakkain tai erikseen. Erillisiä testejä voidaan ajaa komentoriviltä antamalla käynnistyskomennossa testitapauksen nimi. Mikäli testitapauksen nimeä ei erikseen syötetä, ajaa robotti kaikki testitapaukset.

Seuraavassa kuvassa (Kuva 9) on saman skriptitiedoston avainsanat: ”Login”, ”Search”, ”Create ticket” ja ”Logout”.



```
*** Keywords ***
Login
    ${JSONLOGIN}    Convert String to JSON    ${JSONLOGIN}
    ${RESPONSEL}    Post    ${LOGINAPIURL}    ${JSONLOGIN}
    ${ONLINEID}    Get Value From JSON    ${RESPONSEL}    $..body.b[0]
    set suite variable    ${ONLINEID}    ${ONLINEID}[0]
    Log    ${ONLINEID}    console=true
    ${PERMISSIONNUMBERS}    Create List    11    13    14
    FOR    ${i}    IN    @{PERMISSIONNUMBERS}
        ${JSON}    Convert String to JSON    {"a":${i}}
        ${RESPONSE}    Post    ${PERMISSIONAPIURL}    ${JSON}    headers={"OnlineId": "${ONLINEID}", "CurrentVersion": "${VERSION}"}
    END

Search
    ${JSONSEARCH}    Convert String to JSON    ${JSONSEARCH}
    ${RANDOMSTRING}    Generate Random String    2    [NUMBERS]
    ${JSONSEARCH}    Update Value To Json    ${JSONSEARCH}    $..b[2]    ${RANDOMSTRING}
    Log    ${JSONSEARCH}    console=true
    ${RESPONSES}    Post    ${SEARCHAPIURL}    ${JSONSEARCH}    headers={"OnlineId": "${ONLINEID}", "CurrentVersion": "${VERSION}"}


Create ticket
    ${JSONCREATETICKET}    Convert String to JSON    ${JSONCREATETICKET}
    ${RANDOMTICKETSTRING}    Generate Random String    3

    ${JSONCREATETICKET}    Update Value To Json    json_object=${JSONCREATETICKET}    json_path=$..c[0].h    new_value=testing${RANDOMTICKETSTRING}

    ${RESPONSET}    Post    ${CREATETICKETURL}    ${JSONCREATETICKET}    headers={"OnlineId": "${ONLINEID}", "CurrentVersion": "${VERSION}"}


Logout
    ${JSONLOGOUT}    Convert String to JSON    ${JSONLOGOUT}
    ${RESPONSELOGOUT}    Post    ${LOGOUTURL}    ${JSONLOGOUT}    headers={"OnlineId": "${ONLINEID}", "CurrentVersion": "${VERSION}"}
```

Kuva 9. Robot Framework-skriptin avainsanat.

Ensimmäisenä avainsanoissa on ”Login”. Alussa muunnetaan aikaisemmin mainittu tekstijono JSON-muotoon. JSON-dataa on helpompi käsitellä kuin tekstijonoja. Seuraavaksi lähetetään äsken muunnettu JSON ”${JSONLOGIN}” palvelimelle osoitteeseen ”${LOGINAPIURL}” ja sieltä saatu vastaus tallennetaan muuttujaan ”${RESPONSEL}”.

Seuraavaksi käsitellään äsken saatua vastausta eli ”${REPONSEL}” muuttujaa. Muuttuja on JSON-dataa, joten sieltä voidaan poimia tietoja, kuten tässä esimerkissä. JSON-rakenteen b[0]-ominaisuudessa on aikaisemmin mainittu *OnlineId* eli *SessionId*. Tämä on erittäin tärkeää onnistuneen sisäänkirjautumisen kannalta. Ilman tätä ei käytännössä ole käyttöoikeuksia tehdä järjestelmässä mitään toimintoja. *OnlineId* on nyt tallennettu muuttujaan ”${ONLINEID}”.

Seuraavaksi luodaan lista, joka sisältää käyttöoikeuksien tunnuksia. Nämä täytyy lähettää palvelimelle POST-pyyntönä. Tätä seuraavaa silmukkaa ajetaan kolme kertaa koska listassa on kolme objektia, 11, 13 ja 14. Palvelimelle lähetään siis erikseen {”a”:11}, {”a”:13} ja {”a”:14}.

Viimeiseksi lähetetään kerätyt datat ja otetaan vastaus talteen. Mikäli tehty POST-pyyntö ei onnistu, ei sisäänkirjautuminen ole onnistunut.

Avainsana ”Search” on huomattavasti yksinkertaisempi kuin ”Login”. Alussa esitelty muuttuja ”${JSONSEARCH}” muunnetaan tekstimuodosta JSON-muotoon. Seuraavaksi luodaan satunnaisesti generoitu tekstijono joka on tässä tapauksessa kaksi numeroa. Tämä siksi, jotta jokaisella haulla käytetään hieman erilaista hakusanaa. Lisäksi päivitetään luotu satunnainen tekstijono mukaan JSON-objektiin, joka lopuksi lähetetään palvelimelle. Seuraavaksi tiedot lähetetään ja vastaus tallennetaan muuttujaan ”${RESPONSES}”. JSON-datan lisäksi palvelimelle tulee myös lähettää otsikkotiedot *OnlineId* ja *CurrentVersion*, jotta käyttäjällä on oikeus tehdä hakuja.

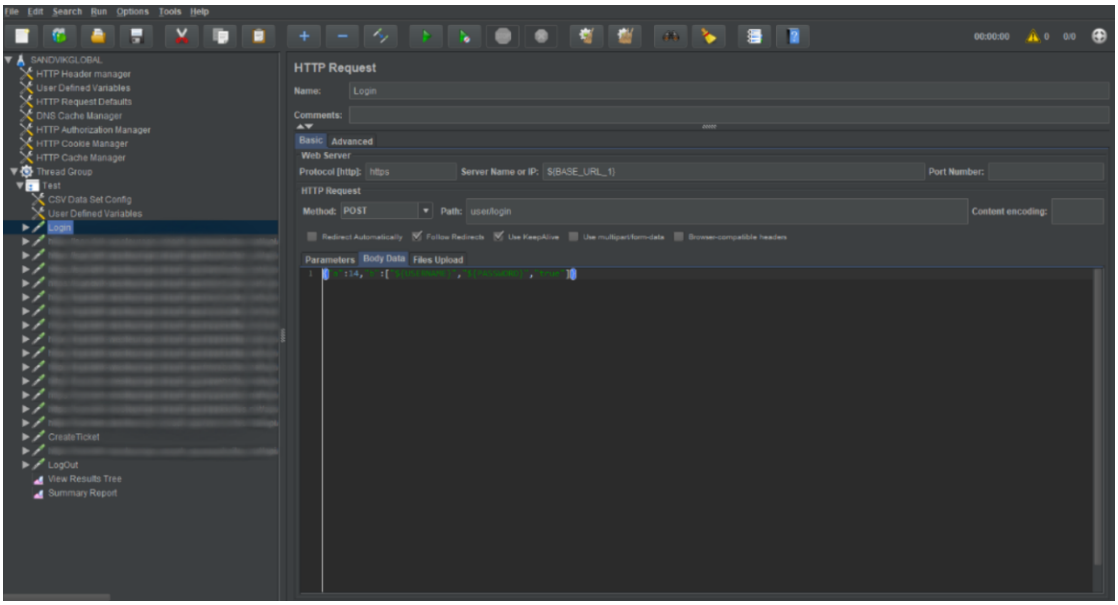
Avainsana ”Create ticket” on hyvin samanlainen kuin ”Search”. Ensin muutetaan taas alussa alustettu tekstijono JSON-muotoon. Seuraavalla rivillä luodaan satunnainen tekstijono, joka sisältää kolme kirjainta tai numeroa. Satunnaisesti generoitu tekstijono lisätään myöhemmin lähetettävään JSON-objektin ominaisuuteen c[0].h. Seuraavaksi tiedot lähetetään palvelimelle ja vastaus tallennetaan muuttujaan ”${RESPONSET}”. JSON-objektin lisäksi palvelimelle tulee myös lähettää otsikkotiedot *OnlineId* sekä *CurrentVersion*, jotta käyttäjällä on oikeus luoda tikettejä.

”Logout” on kaikista avainsanoista yksinkertaisin. Ensin tekstijono muunnetaan JSON-muotoon, jonka jälkeen tämä lähetetään palvelimelle otsikkotietojen kanssa. Taas kerran vastaus tallennetaan, jotta sitä voidaan tarvittaessa tarkastella.

4.2 Apache JMeter suorituskykytestin esittely

Seuraavaksi esittelen Apache JMeterillä tehdyn suorituskykytestin. Testi käyttää jokaiselle simuloidulle käyttäjälle eri kirjautumistietoja CSV-tiedostosta jossa on 1000 eri käyttäjän kirjautumistiedot. Testi kirjautuu sisään, luo järjestelmään tiketin ja kirjautuu sen jälkeen ulos. Testi on hyvin yksinkertainen, mutta riittävä järjestelmän kuormituksen testaamiseksi. Yksi testikierros sisältää kaiken kaikkiaan 18 POST-pyyntöä palvelimelle.

Seuraavassa kuvassa (Kuva 10) on XReach sovelluksen suorituskykytestin sisäänkirjautuminen. Kuvassa esitetään miten POST-pyyntö lähetetään osoitteeseen https://xreach.app/user/login.



Kuva 10. XReach-sovellukseen kirjautuminen suorituskykytestissä.

Lähetetty data on JSON-muodossa ja se sisältää tietoja, jotka järjestelmä vaatii käyttäjältä sisäänkirjautumiseen, esimerkiksi käyttäjätunnus ja salasana. Tässä testissä käyttäjätunnus ja salasana ovat muuttujia, joten ensimmäinen simuloitu käyttäjä käyttää siis aiemmin mainitun CSV-tiedoston ensimmäisen rivin tietoja. Vastauksena käyttäjä saa joko hyväksytyn tai hylätyn vastauksen, riippuen siitä ovatko kirjautumistiedot hyväksytty. Mikäli käyttäjätunnus tai salasana on väärä, on vastaus palvelimelta ”Wrong username or password”. Mikäli kirjautuminen hyväksytään, saa käyttäjä vastauksessa *OnlineId*:n, joka vastaa *SessionId*:tä. *OnlineId* on sen käyttäjän ja sen session id, joka mahdollistaa sovelluksen käyttämisen. Ilman *OnlineId*:tä järjestelmä ei anna käyttäjälle oikeuksia sen käyttöön. JMeterissä on otettu käyttöön tätä varten ”JSON Extractor” työkalu, joka purkaa kirjautumisen yhteydessä saadun vastauksen JSON-datan ja ottaa sieltä talteen kyseisen käyttäjän *OnlineId*:n, jota testi ja käyttäjä sitten käyttää aina uloskirjautumiseen asti. *OnlineId* on voimassa niin kauan, kunnes sessio loppuu tai lopetetaan. Päällekkäinen kirjautuminen tuhoaa käyttäjän vanhan *OnlineId*:n ja luo tilalle uuden. Tällöin ensin kirjautunut käyttäjä ei enää voi käyttää järjestelmää ja käyttäjä kirjataan automaattisesti ulos.

Suorituskykytestin tarkoituksena oli saada selville minkälaisia käyttäjämääriä järjestelmä pystyy käsittelemään kerralla. ”Request unit” on pyyntöjen käsittelijä. Järjestelmällä oli testin aikana käytössä tietty määrä pyyntöjen käsittelijöitä per sekunti. Jos pyyntöjä tulee sekunnissa yli tietyn määrän, joka järjestelmällä oli käytössä, palauttaa järjestelmä virheviestin, jossa kerrotaan käyttäjälle että pyyntöä ei voitu käsitellä, koska pyyntöjen käsittelijät ovat lopussa. Kun näitä virheitä alkaa tulemaan, on selvää, että järjestelmän raja on löydetty.

Löydettyäni sovelluksen kuormituksen rajat lisäsimme saatavilla olevien pyyntöjen käsittelijöiden määrää. Nyt käytössä on systeemi, joka ostaa lisää käsittelijöitä tarpeen vaatiessa eli niiden loppuessa.

5 PÄÄTÄNTÖ

Opinnäytetyön tavoitteena oli tutustuttaa lukija ohjelmistotestaukseen ja lopussa toteuttaa XReach projektiin suorituskyky- ja automaatiotestit. Opinnäytetyössä päästiin tavoitteisiin.

Olen itse tyytyväinen opinnäytetyöhön. Mielestäni onnistuin hyvin avaamaan lukijalle muutamia ohjelmistotestauksen perusmenetelmistä konkreettisten esimerkkien kautta. Työssä näitä menetelmiä käsitellään aluksi yleisellä tasolla ja lopuksi näytetään konkreettinen esimerkki testitapauksista. Testit, joita opinnäytetyössä esiteltiin, eivät kuitenkaan ole täydellisiä. Iso osa Robot Frameworkin testeistä on käyttöliittymätestejä. Lisäisin esimerkkitesteihin myös käyttöliittymätestin.

Toimeksiantaja on myös tyytyväinen tulokseen. Opinnäytetyössä toteutetuista testeistä oli toimeksiantajan projektille konkreettista hyötyä. Suurin hyöty oli suorituskykytestistä, jonka suorittamisen seurauksena projektiin saatiin ratkaisu, joka edistää sen kuormituksen sietokykyä. Nyt järjestelmä pystyy käsittelemään isomman määrän käyttäjiä samanaikaisesti.

Testit ovat nyt XReach tiimissä oikeassa käytössä ja toivottavasti niitä tullaan myös tulevaisuudessa hyödyntämään tarpeen vaatiessa. Testejä on helppo muokata eri ympäristöihin sopivaksi, joita projektissa on paljon. Jokaiselle asiakkaalle on oma ympäristö, jota tulee testata.

Jälkikäteen testejä tarkastellessa ne näyttävät hyvinkin yksinkertaisilta ja nopeilta tehdä. Testien suunnitteluun ja järjestelmän toiminnallisuuteen tutustumiseen meni kuitenkin yllättävän paljon aikaa. Olen kuitenkin tyytyväinen, että olen käyttänyt sen ajan opinnäytetyön tekemiseen.

LÄHTEET

Apache JMeter, 2020. WWW-dokumentti. Saatavissa: https://jmeter.apache.org/ [viitattu 24.9.2020]

Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo Oy.

Kotilainen, S. 2018. Diplomityöntekijä keksi ohjelmiston, josta tuli suomalaisten suosikki. Verkkolehti. Päivitetty 16.5.2018. Saatavissa: https://www.tivi.fi/uutiset/diplomityontekija-keksi-ohjelmiston-josta-tuli-suomalaisten-suosikki/ccb0d491-794f-3cd5-b17e-53ff1c932c64 [viitattu 23.9.2020]

Robot Framework, s.a. Foundation. WWW-dokumentti. Saatavissa: https://robotframework.org/foundation/ [viitattu 23.9.2020]

Robot Framework, s.a. Introduction. WWW-dokumentti. Saatavissa: https://robotframework.org/#introduction [viitattu 23.9.2020]

Softability, s.a. XReach. WWW-dokumentti. Saatavissa https://softability.fi/xreach/ [viitattu 08.10.2020]

Tassey, G. 2002. The Economic Impacts of Inadequate Infrastructure for Software Testing. PDF-dokumentti. Saatavissa: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf [viitattu 2.9.2020].

Testdevlab, 2018. 4 Reasons why software testing is important. WWW-dokumentti. Saatavissa: https://www.testdevlab.com/blog/2018/07/importance-of-software-testing/ [viitattu 2.9.2020]

Thinkingportfolio, 2016. Projektien vesiputousmalli ja sen viisi heikkoutta. WWW-dokumentti. Saatavissa: https://thinkingportfolio.com/projektien-vesiputousmalli-ja-sen-viisi-heikkoutta/ [viitattu 2.9.2020]

Vuori, M, 2010. Suorituskykytestaus/kuormitustestaus. PDF-dokumenti. Saatavissa: https://www.mattivuori.net/julkaisuluettelo/liitteet/suorituskykytestaus.pdf [viitattu 23.9.2020]

KUVALUETTELO