SAVONIA
ammattikorkeakoulu

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

VERTAILU WEB-SIVUJEN TESTAUS-TYÖKALUISTA: TEHOKKUUS, KÄY-TETTÄVYYS JA KUSTANNUKSET

TEKIJÄ Juho Pennanen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Juho Pennanen	
Työn nimi Vertailu web-sivujen testaustyökaluista: tehokkuus, käytettävyys ja kustannukset	
Päiväys 8.5.2023	Sivumäärä/Liitteet 34
Toimeksiantaja/Yhteistyökumppani(t) Ecomond Oy	

Tiivistelmä (Huom. kirjoita teksti alla näkyvään harmaaseen kenttään; huomioi tämä myös kopioitaessa)

Opinnäytetyön aiheena oli etsiä web-sivujen automaatiotestaustyökaluja ja vertailla niitä tehokkuuden, käytettävyyden ja kustannusten perusteella. Työn toimeksiantaja oli Ecomond Oy, joka on kuopiolainen ohjelmistofirma. Opinnäytetyön tavoitteena oli löytää monien automaatiotestaustyökalujen joukosta parhaat niiden ominaisuuksien perusteella. Syvempänä tarkoituksena oli löytää paras mahdollinen työkalu yrityksen tuotteiden jatkokehitystä varten.

Opinnäytetyö toteutettiin kahdessa osassa, vertailun tekemisellä ja raportin kirjoittamisella. Työssä edettiin erilaisten karsintakierrosten kautta ja päästiin lopulta tilanteeseen, jossa tutkittavia työkaluja oli jäljellä enää sopiva määrä syvempää tarkastelua varten. Kullakin automaatiotestaustyökalulla suoritettiin monivaiheinen testisarja ja arvioimme niiden käyttökokemusta, työkalun oppimisen pituutta ja testisarjan suoritus aikaa.

Opinnäytetyössä päädyttiin lopulta testaamaan seitsemää eri testaustyökalua. Työssä käytettiin monia eri toteutustapoja ja koodikieliä. Lopputuloksena saatiin yleiskatsaus näistä syvemmin tarkastelluista työkaluista ja niiden ominaisuuksista. Työssä onnistuttiin projektisuunnitelman mukaisesti. Käytännön osuus saatiin toteutettua sitä varten suunnitellussa ajassa.

Opinnäytetyössä saavutettiin sitä varten asetetut tavoitteet. Työn etenemisprosessi toteutui hieman suunnitellusta poiketen. Opinnäytetyössä lähteinä käytettiin monia erilaisia toteutuksia.

Avainsanat

Automaatiotestaus, web-sivu, Katalon studio, Lambdatest, Selenium, Ghost Inspector, BrowserStack, Robot Framework, Telerik Test Studio

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport	
Degree Programme Degree Programme in Information Technology	
Author(s) Juho Pennanen	
Title of Thesis Comparison of Testing Tools for Websites: Efficiency, Usability, and Costs	
Date 8.5.2023	Pages/Appendices 34
Client Organisation /Partners Ecomond Oy	

Abstract (NOTE: write/insert all your text in the grey box below, also if you use copy + paste)

The aim of this thesis was to search automation testing tools for websites/webapps and compare them based on efficiency, usability and costs. The client of this thesis was Ecomond Oy, which is a software company in Kuopio. The goal was to find the top group among many automation testing tools based on their features. The deeper purpose was to find the best possible tool for the further development of the company's products.

The thesis was done in two phases, first testing the tools and comparing them, and then writing a report. The work made progress with various qualifying rounds and finally reached a situation where there were only a suitable number of tools left for a thorough examination. With each tool a multi-step test series was executed, and the tools were evaluated using the following criteria: user experience, length of learning time and the time it took for the tool to execute the test series.

Finally, there were seven tools that were thoroughly tested. Many different implementation methods and code languages were used during this testing period. As a result, an overview of these tools and their features was formed. In regards of the project plan, the work was successful. The practical part was completed in the planned time.

The thesis achieved the goals set for it. The progress of the work progressed a little differently than planned. Many different implementations were used as sources in the thesis.

Keywords
Test Automation, Website, Katalon studio, Lambdatest, Selenium, Ghost Inspector, BrowserStack, Robot Framework, Telerik Test Studio

SISÄLTÖ

1 JOHDANTO 6
1.1 Tausta ja tavoite 6
1.2 Sanastoa 6
2 TEORIA 7
2.1 Ohjelmistotestaus 7
2.1.1 Yksikkötestaus 7
2.1.2 Integraatiotestaus 7
2.1.3 Järjestelmätestaus 8
2.1.4 Hyväksymistestaus 8
2.2 Automaatiotestaus 8
2.3 Tehokkuus 8
2.4 Käytettävyys 9
2.5 Kustannukset 9
3 TUTKIMUKSEN TOTEUTTAMINEN 11
3.1 Lähtökohta 11
3.2 Työkalujen karsiminen 11
3.3 Työkalujen läpikäynti 13
3.4 Arviointi 13
4 KATALON STUDIO 14
4.1 Esittely 14
4.2 Tehokkuus 14
4.3 Käytettävyys 15
4.4 Kustannukset 17
5 LAMBDATEST 18
5.1 Esittely 18
5.2 Tehokkuus 18
5.3 Käytettävyys 19
5.4 Kustannukset 19
6 SELENIUM WEBDRIVER 20
6.1 Esittely 20
6.2 Tehokkuus 20

6.3 Käytettävyys ... 21
6.4 Kustannukset ... 21
7 GHOST INSPECTOR ... 22
7.1 Esittely ... 22
7.2 Tehokkuus ... 22
7.3 Käytettävyys ... 22
7.4 Kustannukset ... 23
8 BROWSERSTACK ... 24
8.1 Esittely ... 24
8.2 Tehokkuus ... 24
8.3 Käytettävyys ... 24
8.4 Kustannukset ... 24
9 ROBOT FRAMEWORK ... 26
9.1 Esittely ... 26
9.2 Tehokkuus ... 26
9.3 Käytettävyys ... 26
9.4 Kustannukset ... 26
10 TELERIK TEST STUDIO ... 27
10.1 Esittely ... 27
10.2 Tehokkuus ... 27
10.3 Käytettävyys ... 27
10.4 Kustannukset ... 28
11 YHTEENVETO ... 29
11.1 Tulokset ... 29
11.2 Oman työn arviointi ... 30
LÄHTEET ... 32

1 JOHDANTO

1.1 Tausta ja tavoite

Ohjelmiston testaus on aina ollut tärkeä vaihe ohjelmistotuotannossa. Testausta on ollut niin kauan kuin koodiakin on kirjoitettu. Tämä pätee myös web-sivujen käyttöliittymän testaukseen. Automaation käyttöönotolla voidaan vähentää testaustyön työläyttä. Vaikka web-sivujen automaatiotestaus ei ole uusi käytäntö, sen käyttö on ollut hankalaa ja työlästä. Aika lailla aina kun koodi muuttuu, testienkin on muututtava. Nyt kun elämme tekoälyn aikaa, myös automaatiotestaustyökalut ovat kehittyneet käyttämään niitä. Tekoälyavusteiset sovellukset tarjoavat nykyään tuotteitaan Self healing -ominaisuudella, joka korjaa omaa toteutustaan kesken prosessin.

Tämän opinnäytetyön tarkoituksena on löytää ja vertailla websivujen automaatiotestaustyökaluja. Vertailussa otetaan huomioon tehokkuus, käytettävyys ja kustannukset. Tavoitteena on tutkimisen ja testauksen kautta laatia yleiskatsaus kustakin listalla olevasta sovelluksesta, testata niiden tehokkuutta ja käytettävyyttä ja selvittää niiden kustannuksia.

Opinnäytetyön toimeksiantajana toimii kuopiolainen tehtävienhallintaan ja optimointiin erikoistunut ohjelmistotoimittaja Ecomond Oy. Yritys on perustettu 2002. Heidän päätuotteensa ovat TCS-järjestelmä ja TCS-optimointijärjestelmä.

1.2 Sanastoa

IFrame - Sisäinen kehys (iframe) on HTML- elementti, joka lataa toisen HTML-sivun asiakirjaan. Se käytännössä asettaa toisen verkkosivun pääsivulle. Niitä käytetään yleisesti mainoksissa, upotetuissa videoissa, verkkoanalytiikassa ja interaktiivisessa sisällössä.

IDE - Integrated Development Environment. Tarkoittaa ohjelmisto ympäristöä, jolla ohjelmoija suunnittelee ja toteuttaa ohjelmistoja.

IntelliSense - Integrated Development Environment. Tarkoittaa ohjelmisto ympäristöä, jolla ohjelmoija suunnittelee ja toteuttaa ohjelmistoja.

Plugin - Plugin tarkoittaa Suomeksi liitännäistä. Käytännössä sillä tarkoitetaan johonkin verkko-ohjelmistoon suunniteltua lisäosaa, lisätyökalua, jonka avulla ohjelmistoon saadaan uusia ominaisuuksia ja toiminnallisuuksia.

CI/CD – CI tulee sanoista Continuous Integration ja CD tulee sanoista Continuous Delivery. Nämä tarkoittavat jatkuvaa integraatiota ja jatkuvaa toimitusta.

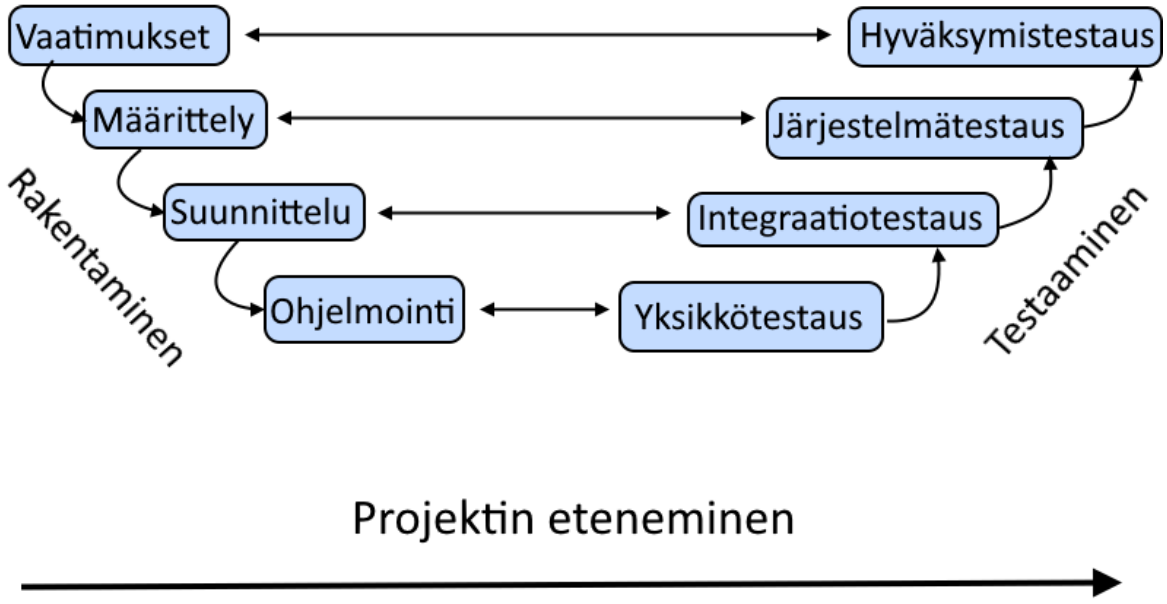
Tutoriaali - Opetusohjelma opetuksessa on tiedon siirron menetelmä ja sitä voidaan käyttää osana oppimisprosessia.

Framework – Ohjelmoinnin viitekehys, joka tarjoaa valmiita komponentteja tai ratkaisuja, jotka on räätälöity nopeuttamaan kehitystä.

2 TEORIA

2.1 Ohjelmistotestaus

Ohjelmistotestaus on käsitteenä laaja, joten se yleensä jaetaan eri osa-alueisiin ja tasoihin. Yleensä tämä jaottelu sisältää seuraavat aihealueet: Yksikkötestaus, Integraatiotestaus, järjestelmätestaus ja hyväksymistestaus. (Valagroup, 2022)



Kuva 1 Testauksen v-malli. Muokattu (Kasurinen, 2013)

2.1.1 Yksikkötestaus

Alhaisimpana testauksen tasona, jossa varmistetaan koodin pienimmätkin erilliset osat, on yksikkötestaus. Tämä testauksen muoto kuitenkin edellyttää sitä, että koodi on kirjoitettu tietyllä tavalla, pienissä osissa, jotta kunkin yksikön voi sitten testata omassa testissään. (Valagroup, 2022)

Yksiköllä tarkoitetaan esimerkiksi yhtä toimintoa, funktiota tai olio-ohjelmoinnissa yhtä luokkaa. (Wikipedia, 2023)

2.1.2 Integraatiotestaus

Yksikkötestauksen seuraava vaihe on Integraatiotestaus, sen ydin on se, miten järjestelmän eriosat keskustelevat keskenään. (Valagroup, 2022)

Integraatiotestausta on esimerkiksi se, että kun yksikkötestit on saatu tehtyä, yhdistellään niistä kokonaisuuksia / moduuleita ja testataan näiden eri moduulien yhteistä toimivuutta. (Software Testing Help, 2023)

2.1.3 Järjestelmätestaus

Kun integraatiotestaukset on saatu valmiiksi, voidaan alkaa miettiä koko järjestelmän mittaavia testejä. Tarkoituksena on yhdistää kaikki integroidut osat ja testata ohjelmaa kehitysympäristön kaltaisessa ympäristössä. (Valagroup, 2022)

2.1.4 Hyväksymistestaus

Tämä testaus tapahtuu projektin loppuvaiheilla, jolloin loppuasiakas itse testaa ohjelmistoa, joka on jo lähes valmis. Testauksessa käydään läpi vastaako se oikeita käyttötapaus vaatimuksia, ja se on yleensä pelkästään toiminnallista testausta. (Valagroup, 2022)

2.2 Automaatiotestaus

Automaatiotestaus tai toisinpäin testausautomaatio on ohjelmistotestauksen menetelmä, jossa erilaisten teknologisten ratkaisujen kautta suoritetaan testit automaattisesti. Tämä menetelmä ei korvaa täysin muita testausmenetelmiä, vaan toimii manuaalisen testauksen rinnalla. (ite wiki, ei pvm) Nämä testit, joita suoritetaan automaattisesti ovat jotain edellä mainituista menetelmistä, yksikkö-, integraatio-, järjestelmä- tai hyväksymistestejä. Olemassa on myös muita testaus tyyppejä ja niiden alalajeja. Tässä dokumentaatiossa keskitymme automaation testaustasoon, jossa testaamme websivujen käyttöä. Tämä testaustaso sisältyy funktionaaliseen testaukseen, jossa sovellusta käyttämällä saadaan selville, toimiiko se toivotulla tavalla.

Automatisoitu testaus on prosessi, jonka avulla varmennetaan ohjelman asianmukaisuus ja vaatimusten täyttyminen ennen tuotteen tuotantoon julkaisemista. Apuna käytetään testaustyökaluja, joilla ajetaan testien komentosarjoja. (Gillis, 2019)

2.3 Tehokkuus

Fysiikan tunneilta tunnemme jo sanat työ ja teho. Teho (P) on määritelty olevaksi työ (W) jaettuna siihen käytetyllä ajalla (t), $P = \frac{W}{t}$. Tehon yksikkönä on watti. (Hautala & Peltonen, 2014)

Kuten fysiikan maailmassa, niin ohjelmistojen testauksessakin voimme käyttää samaa logiikkaa ”jossain määrin” määritelläksemme miten tehokas jokin sovellus on. Kuitenkaan tätä tehon kaavaa emme pysty soveltamaan tuleviin testeihimme. Joten ohjelmiston tehokkuuden mittaamiseen käytämme jotain toista menetelmää.

Tulemme mittaamaan tehokkuutta tämän opinnäytetyön merkeissä seuraavin menetelmin:

1. Järjestämme jonkin vakio testisarjan, jonka jokainen työkalu tulee suorittamaan.
2. Mittaamme käyttämämme ajan tämän työkalun opetteluun ja testien tekemiseen.
3. Mittaamme työkalun suoritus ajan, joka testisarjan suorittamiseen menee.
4. Listaamme ylös mittaamamme ajat.

Tämä testisarja tulee pitämään sisällään seuraavat vaiheet:

1. Pääsivun avaus
2. Kirjautuminen palveluun
3. Varmenna kirjautuminen

4. Luo tehtävälista
    a. Syötetään testidata tehtävälistalle
5. Varmennetaan että tehtävälista muodostui käyttöliittymästä
6. Luo tehtävä
    a. Syötetään testidata tehtävälistalle
7. Varmenna että tehtävä muodostui käyttöliittymästä
8. Muokkaa tehtävää
9. Varmenna että tehtävä muokkaantui käyttöliittymästä
10. Siirrä tehtävä luodulle tehtävälistalle
11. Varmenna että tehtävä siirtyi
12. Kuittaa tehtävä valmiiksi
    a. Avaa tehtävä kuitattavaksi ja syötä kuittaus tiedot
13. Varmenna että tehtävä kuittaantui
14. Poista tehtävä
15. Poista lista
16. Kirjaudu ulos

Opinnäytetyössä tullaan esittämään lopulta kaavio, jossa näkyy jokaisen työkalun pisteytykset.

2.4 Käytettävyys

Käytettävyydestä on hyvä puhua myös terminä tässä dokumentaatiossa, sillä se on yksi kriteeri, jolla lopulta pisteytetään työkalut. Hangasmaa (Hangasmaa, 2010) kertoo opinnäytetyössään viidestä käytettävyyden ominaisuudesta, joita ovat, miellytettävyys, helppous, tyydyttävyys, johdonmukaisuus ja muistettavuus. Näistä osaa tullaan ottamaan huomioon kussakin työkalun testaus vaiheessa.

Hangasmaa myös lisää käytettävyydelle rinnakkaisominaisuuksia, joita voidaan pitää osana käytettävyyttä, näitä ovat palvelevuus, houkuttelevuus, helppokäyttöisyys, esteettömyys, käyttäjäkokemus ja käyttökokemus. (Hangasmaa, 2010)

Käytettävyyttä voimme mitata kussakin työkalun käsittelyvaiheessa seuraavasti:

1. Onko työkalua helppo käyttää?
2. Onko työkalu houkutteleva?
3. Millaisen työn kautta työkalu saadaan osaksi jatkuvaa integraatiota?
4. Ylläpidetäänkö sovellusta säännöllisesti?

2.5 Kustannukset

Yleensä yrityksessä kustannukset jaetaan muuttuviin ja kiinteisiin kustannuksiin. Sen, kumpaanko kustannukset kuuluvat, määrää toiminta-aste. Tämä toiminta aste on määritelmän mukaan tuotannon määrä aikayksikköä kohden. Muuttuvat kustannukset ovat nimensä mukaan muuttuvia, mutta tarkemman selityksen mukaan kustannukset ovat muuttuvia, jos ne kasvavat tai vähenevät toiminta-asteen muuttuessa. Ja täten kiinteät kustannukset ovat niitä, joissa kustannusten määrä ei muutu, eli pysyy vakiona. (Tenhunen, 2013)

Nämä kustannukset voidaan myös lajitella välittömiin ja välillisiin kustannuksiin.

Tässä opinnäytetyössä tulemme keskittymään seuraaviin seikkoihin kustannuksien kohdalla:

1. Hinta kuukaudessa.
2. Mitä peruspaketti pitää sisällään.

3 TUTKIMUKSEN TOTEUTTAMINEN

3.1 Lähtökohta

Työssä lähdettiin liikkeelle tutustumalla automaatiotestaukseen ja sen keskeisiin käsitteisiin. Pyrittiin löytämään vastauksia kysymykselle, ”mitä automaatiotestaus on”. Automaatiotestaus on ohjelmistotestauksen osajoukko, joka pitää sisällään yksikkö-, integraatio-, järjestelmä- ja hyväksymistestauksen. Ohjelmistotestauksessa testit suoritetaan manuaalisesti tai automaation avulla. Tässä opinnäytetyössä keskityttiin web-sivujen testaukseen ja sen automatisointiin eri sovellusten avulla.

3.2 Työkalujen karsiminen

Tavoitteena on löytää mahdollisimman monta työkalua, joilla voitaisiin toteuttaa automaatiotestejä. Työssä lähdettiin etsimään tietoa internetistä käyttämällä erilaisia hakusanoja, kuten ”alternatives for Testproject.io”. Vastaukseksi saatiin monista hakusanojen variaatioista yleensä Cypress ja Selenium. Työkaluja löytyi vain kourallinen sillä menetelmällä, joten työssä otettiin avuksi uusi tekoäly tuote ChatGPT ja kysyttiin siltä, mitä eri testiautomaatio työkaluja mahdollisesti on. Tekoäly tuotti vastaukseksi ulos kymmeniä vastauksia, jotka nyt lisättiin jo olemassa olevaan listaan.

Työkaluvaihtoehtoja oli tässä vaiheessa kuitenkin liian paljon läpikäytäväksi, joten karsintoja tuli toteuttaa. Taulukossa (Taulukko 1) nähdään kaikki työkalut, jota listalta alussa löytyi, ja kunkin karsintakierroksen kohdalla X merkintä, jos kyseinen työkalu poistettiin listalta.

Ensimmäisen karsintakierroksien aiheena oli poistaa listalta kaikki työkalut, jotka olivat selvästi vain mobiilitestausta varten. Listalta poistui tämän kautta vain neljä työkalua, joten oli laajennettava karsintakriteerejä. Toisen karsinnan kriteerinä oli ”Onko työkalu tarkoitettu automaatiotestaukseen ja päivitetäänkö työkalua säännöllisesti/onko tämä projekti hylätty?”, sillä tekoälyn vastaukset eivät välttämättä pitäneet paikkaansa. Kävi ilmi, että moni listatuista työkaluista ei ollutkaan pääasiassa juuri tätä automaatiotestausta varten. Tämän avulla listalta poistui kahdeksan työkalua. Kolmannella karsinta kierroksella kriteerinä oli ”Onko kyseessä pelkästään apukirjasto toiselle työkalulle? ”. Monet työkaluista olivat Node.js pohjaisia opensource-kirjastoja, joita pystyi käyttää itsenäisesti, mutta jotkut listalla olevista työkaluista käyttävät näitä kirjastoja, joten tällaiset duplikaatit pystyttiin poistamaan listalta. Tämän avulla listalta saatiin karsittua vielä kahdeksantoista työkalua.

Työkalu	Karsinta kierros 1	Karsinta kierros 2	Karsinta kierros 3
Selenium WebDriver			
Katalon Studio			
Cypress			
Robot Framework			
Testim.io			
Virtuoso			
TestComplete			
Bugbug.io			X

Rapise			
Browser stack			
JUnit		X	X
TestCafe			X
Puppeteer			X
Lambdatest			
Intruder		X	X
TestRigor			X
Testpad		X	X
QAWolf			X
TestNG			X
Appium	X	X	X
Protractor	X	X	X
WebdriverIO			X
Mocha			X
Chai		X	X
Nightwatch.js			X
JBehave			X
Cucumber			X
SpecFlow			X
Sauce Labs	X	X	X
Microsoft Test Manager(MTM)	X	X	X
Gauge			X
TensorFlow		X	X
PyTest			X
NUnit		X	X
Karma			X
CodeceptJS			X
Ghost Inspector			
TestProject.io		X	X
TestRail			X
Zephyr		X	X

Ranorex			X
Telerik Test Studio			

Taulukko 1. Karsinnat

Karsintojen jälkeen jäljelle jäi vielä 12 työkalua. Nyt rajaukset tuli tehdä eri metodein eli tutustumalla työkaluihin ja poistamalla semmoiset, jotka eivät osoittaneet luottamusta tai hinta oli muihin nähden korkea. Jäljelle jäi tämän jälkeen viimeiset 7 läpikäytävää työkalua, Katalon Studio, Selenium WebDriver, Lambdatest, Ghost Inspector, BrowserStack, Telerik Test studio ja Robot Framework.

3.3 Työkalujen läpikäynti

Kunkin työkalun kohdalla lähdettiin liikkeelle siitä, että tutustuttiin sen opetusmateriaaliin ja alettiin tutkimaan mitä tämän työkalun käyttäminen vaatii. Opetusmateriaalina toimi kunkin sovelluksen oma dokumentaatio ja videotutoriaalit muista lähteistä. Tehokkuuteen liittyvää työkalun läpikäynnin aikaa alettiin pitää heti. Käyttökokemuksesta alettiin pitämään päiväkirjan tyylistä kirjausta kustakin työkalusta. Työkalun pienen opettelun jälkeen alettiin tekemään testisarjan testejä, jotta nähtiin miten helposti ja intuitiivisesti testien luominen sujui. Testien läpipääsyn jälkeen lopetettiin ajanotto, ajastettiin testin suoritusaika ja alettiin kirjoittamaan siihen liittyvää dokumentaatiota.

3.4 Arviointi

Työkalujen läpikäynnistä tehtiin Excel-taulukko, joka sisälsi pisteytyksiä kunkin läpikäytävän vertailun aiheista. Kukin läpikäytävä aihe sisälsi aliotsikoita, joiden avulla saimme paremman pisteytyksen. Tehokkuuteen liittyviä aliotsikoita olivat opettelun pituus ja testien suoritus aika. Käytettävyyden kohdalla näitä olivat helppokäyttöisyys, houkuttelevuus, työnmäärä jotta CI/CD toimii ja ylläpidetäänkö sovellusta enää. Kustannusten kohdalla näitä olivat hinta kuukaudessa, ja lisäpiste jos tuotteella on jokin ylivertainen lisäominaisuus. Työkaluista koottiin pisteytystaulukko tämän dokumentaation yhteenveto-osuuteen.

4 KATALON STUDIO

4.1 Esittely

Katalon Studio on mobiilisovellusten ja verkkoautomaatioon suunnattu työkalusarja, joka sisältää monia ominaisuuksia haastamaan UI-automaatiotestauksen yleiset ongelmat, esimerkiksi odotusajan ja iFrame-ponnahdusikkunat. (Yerukala, ei pvm)

Katalon Studiosta saa kaiken irti käyttämällä sitä yhdessä Katalon Platformin kanssa. Katalon Platform on laadunhallinta-alusta, jota käytetään tuotteen testauksessa alusta loppuun. Näihin vaiheisiin kuuluu testien suunnittelu, luominen, järjestäminen, suorittaminen ja raporttien analysointi. (Katalon, ei pvm)

Katalon Platform tarjoaa monta eri työkalua, eri vaiheisiin ja eri tarkoituksiin. Nämä työkalut ovat Katalon TestOps, Katalon Test Cloud, Katalon Runtime Engine ja Katalon Studio. Tässä dokumentaation osiossa käsittelemme Katalon Studiota.

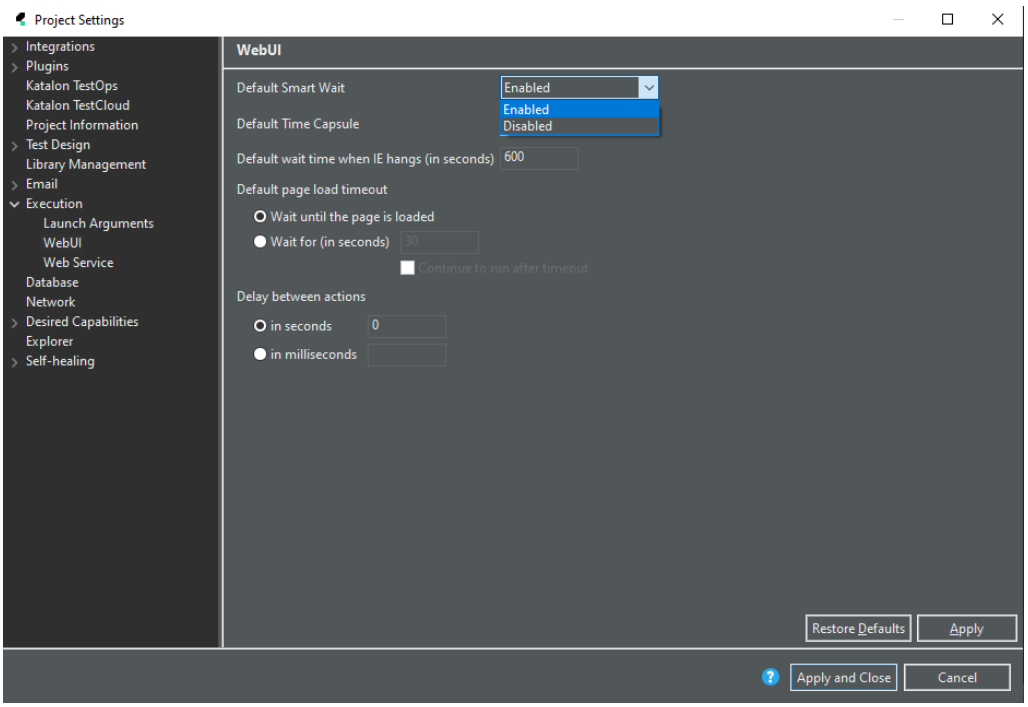
Katalon Studio on testiautomaatiotyökalu, joka on rakennettu Seleniumin päälle. Testauksen aloittamisesta on tehty helppoa sillä ohjelmointitaitoa ei välttämättä tarvita ollenkaan. (Katalon, ei pvm)

4.2 Tehokkuus

Katalon Studion ensimmäinen havaittu ominaisuus oli hitaus. Sovelluksen IntelliSense otti aikansa, kunnes tuotti järkeviä koodin täydennysvaihtoehtoja. Sovellus suoritti testit käyttäen aikaa odotellessa elementtejä sivulla. Syynä tähän oli työkalun SmartWait-ominaisuus.

Yleensä testejä suorittaessa testien vaiheet etsivät verkkosivulta elementtejä. Nämä elementit jostain syystä saattoivat olla piilossa tai jonkun toisen elementin takana. Testin vaiheen keskellä, kun sivut päivittyivät tai sulkivat jotain ponnahdusikkunaa, ei kaikki välttämättä ollut valmista juuri siinä hetkessä. Testi etsi piilossa tai olemassa olematonta elementtiä ja ei löytänyt sitä, minkä jälkeen ohjelma heitti virheilmoituksen. Virheen jälkeen testin suoritus loppui, koska kaikki ei mennyt niin kuin piti. Puhdasta Seleniumia käyttävät yleensä ratkaisevat tämän ongelman käyttämällä ”implicit wait” ja ”explicit wait” funktioita. Nämä funktiot laittavat ohjelman koodin tavallaan uneen ja heräävät vasta kun tietty ehto on saavutettu. Vaikka Katalon Studio on rakennettu Seleniumin päälle, ei se tarjoa mahdollisuutta luoda omia odotusehtoja helposti.

Katalon Studio tuotti tilalle Smart Wait-ominaisuuden toteutukseensa. Testit nyt automaattisesti odottivat, että sivut tai sivun sisällöt latautuivat kokonaan, ja vasta sen jälkeen suoritti testien seuraavat vaiheet. Tämän ominaisuuden pystyi ottaamaan käyttöön Katalon Studiossa projektin asetuksista kohdasta Execution -> WebUI. (Kuva 2)



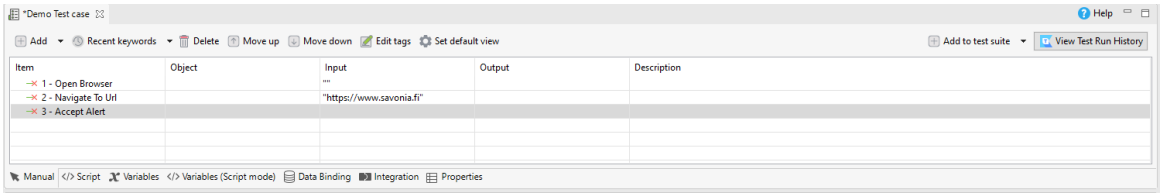
Kuva 2. Project Settings Katalon studiossa (Katalon Studio, 2023)

Katalon Studion läpikäyminen ja opettelu toteutui kolmen työpäivän sisällä. Työkalusta oli paljon tietoa tarjolla, joten oppiminen oli tehokasta. Oppimiseen pystyi käyttämään itse Katalonin tuottamaa dokumentaatiota sovelluksesta tai katsoa videotutoriaaleja. Katalon tarjosi myös maksua vastaan oppaan, joka opettaa sovelluksen käyttöä. Tätä palvelua ei käytetty opinnäytetyössä.

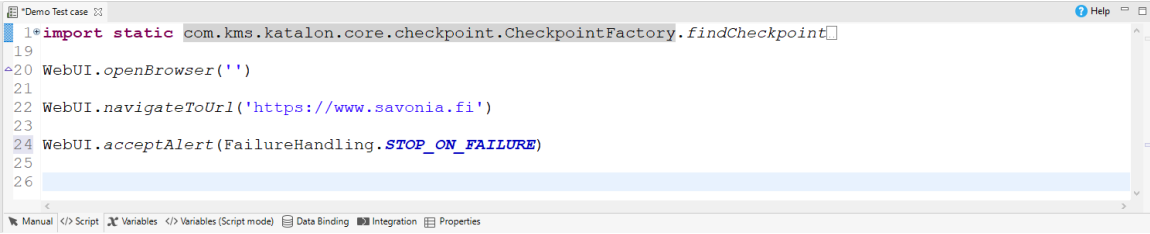
Testien tekeminen oli kohtuullisen tehokasta, koska apuna pystyi käyttämään Web Recorderia. Koodin kirjoittaminen oli Java-kielen tuntevalle helppoa koska Katalon studio käyttää Groovy-ohjelmointikieltä, joka on kehitetty Javasta.

4.3 Käytettävyys

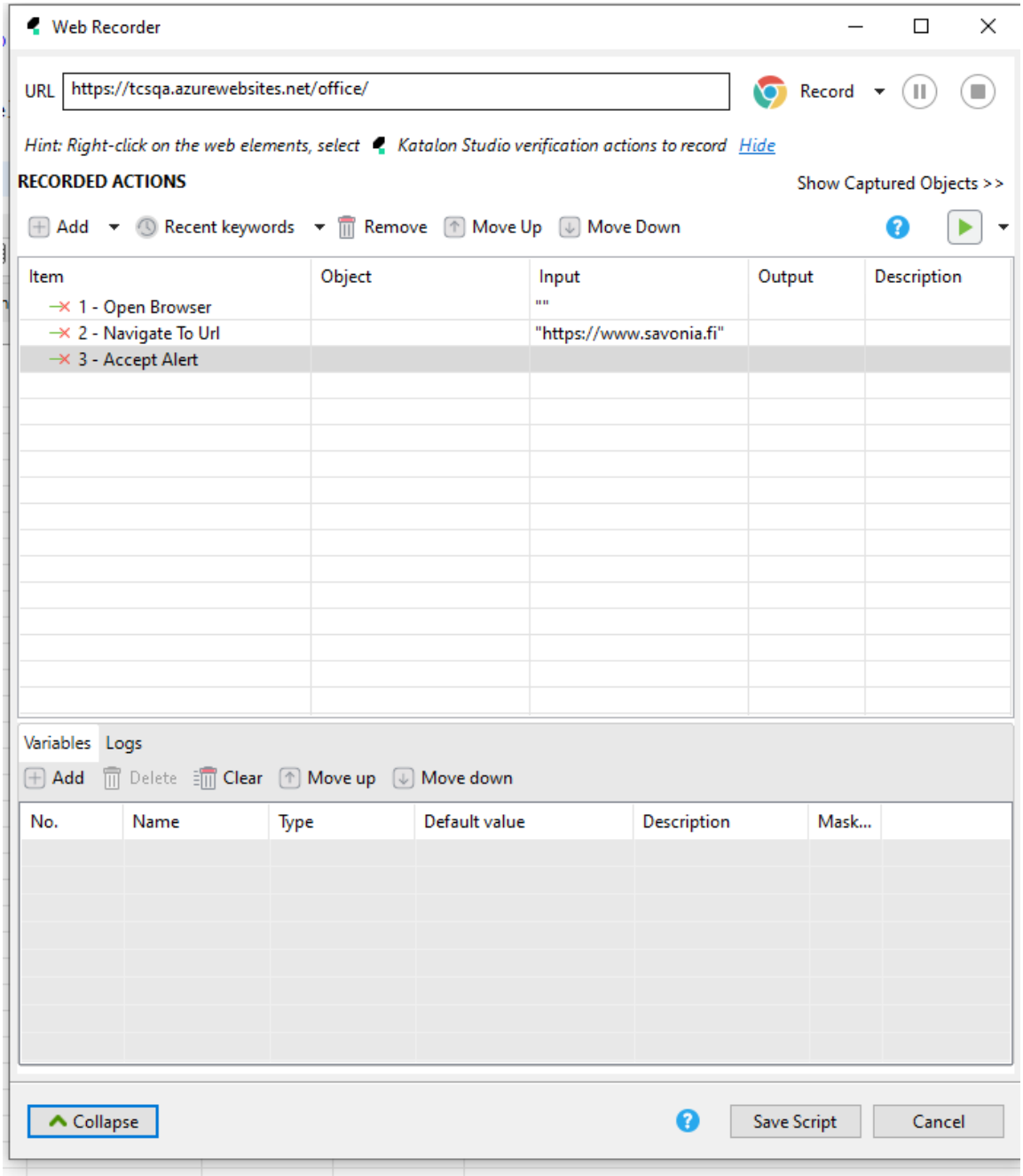
Sovellus oli käytettävyydeltään hyvä. Testien tekeminen oli itsessään helppoa, sillä käytössä oli kolme eri vaihtoehtoa, testien nauhoittaminen (Kuva 5) eli Web Recorder, manuaalinen muokkaaminen (Kuva 3) ja skriptin muokkaus (Kuva 4). Jokainen näistä vaihtoehdoista oli koko ajan käytettävissä, sillä eri muodot olivat liitoksissa toisiinsa. Jos muokkasi testiä jossain näistä, se päivittyi kahteen muuhun.



Kuva 3. Manual Mode Katalon studiossa (Katalon Studio 2023)

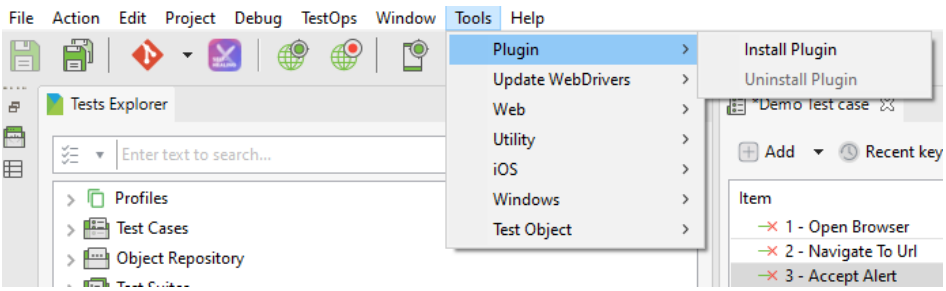


Kuva 4. Script mode Katalon studiossa (Katalon Studio 2023)



Kuva 5. Record mode Katalon studiossa (Katalon Studio 2023)

Ohjelman asentaminen on erittäin helppoa. Katalonin kotisivuilta pystyi lataamaan sovelluksen pakattuna tiedostona, jonka jälkeen se tuli vain purkaa haluttuun paikkaan ja avata sovellus. Kaikki toimi ilman mitään asennusvelhoa. Paketin mukana tuli kaikki mitä sovellus tarvitsi, ja jos jotain lisäosia siihen toivoi, pystyi ne hankkimaan Katalon storesta. Sieltä pystyi lataamaan plugin lisäosia Katalon studioon. Pluginien asennus oli erittäin helppoa, sillä sovelluksen valikkoriviltä pystyi valitsemaan Tools -> Plugin -> Install Plugin ja valita haluamansa plugin (ks. Kuva 6).



Kuva 6. Plugin menubar Katalon studiossa (Katalon Studio 2023)

CI/CD Integraatio esimerkiksi Jenkinsiin tai Devops pipelineen toimi ainoastaan maksullisen lisäosan, Katalon Runtime Enginen kautta. Tässä opinnäytetyössä ei keskitytty CI/CD integraation toteuttamiseen.

4.4 Kustannukset

Työkalusta oli tarjolla ilmaisversio, jonka avulla saatiin tutustuttua itse työkaluun ja siihen liittyviin ominaisuuksiin. Ilmaisversion mukana saatiin otettua 30 päivän pituisen koeaika jakson Premium versiosta. Ilmaisversio rajoitti Testitulosten määrän kahteen tuhanteen kuussa. Alustan käyttäjiä oli saatavilla vain 5 kappaletta. Tietoja säilytettiin 6 kk. Teknistä tukea sai vain yhteisö foorumeilta tai dokumentaatiosta itsekatsomalla.

Premium versio maksoi $300 kuukaudessa, joka laskutettaisiin vuosittain. Tämän version myötä testitulosten määrä kuukaudessa alkoi 3500:sta. Alustan käyttäjiä oli nyt saatavilla loputtomasti. Tietoja säilytettiin 12 kk. Teknistä tukea sai tikettijärjestelmän kautta, joka toimii 24/5 periaatteella.

Premium version lisäksi pystyi ostaa halutessaan erityökaluja käyttöönsä vuodeksi kerrallaan. Tarjolla oli seuraavia työkaluja: Katalon Studio Enterprice $1999, Katalon TestCloud $1849, ja Katalon Runtime Engine $1599. Kukin työkalu toi mukanaan omat hyvät ominaisuutensa, mutta tärkeimpänä luultavasti oli Katalon Runtime Engine, joka mahdollisti CI/CD-integroinnin.

Tarjolla oli myös Ultimate-versio, jonka hinnoittelusta pystyi keskustelemaan itse yrityksen kanssa. Tämän version myötä ei tullut edellisessä kohdassa luetellut maksulliset työkalut mukana. Ne piti tässäkin vaiheessa ostaa erikseen. Tässä versiossa aika lailla kaikki oli loputonta, käyttäjä määrä, projektien määrä, tietojen säilytys ja jopa teknisessä tuessa sinulle määrätty perehdytyspäällikkö.

5 LAMBDATEST

5.1 Esittely

LambdaTest on omien sivujensa mukaan ”CrossBrowser Testing Cloud” eli suomeksi monen eri selaimen tukeva testauspilvipalvelu. Palvelulla saa suoritettua sekä tutkivaa että automatisoitua testausta yli 3000 eri selaimella, oikeilla laitteilla ja käyttöjärjestelmillä. (LambdaTest, 2023)

Kyseessä on skaalautuva testausalusta, johon on sulautettu joitakin upeita ominaisuuksia joidenka avulla se erottuu muiden kilpailevien työkalujen joukosta. Näitä ominaisuuksia on muun muassa online-selaimen yhteensopivuuden testaus, testaus uusimmilla selaimilla, vasteen testaus kaikenkokoisilla näytöillä, jatkuva yhteistyö testauksen aikana, kuvien ottaminen kokoruudusta, visuaalinen testaus ja sisäänrakennettu ongelmien seuranta. (The New Stack, 2019)

Vaikka LambdaTest on pilvipalvelu, ei sen sivuilla itsessään kirjoiteta testejä. Testit tuotetaan itse koodin muodossa, ja ne testit ajetaan LambdaTestin tarjoamien palvelujen kautta. LambdaTest on kasannut yhteen suosituimmat frameworkit testejä varten ja on tehnyt niiden käytöstä helppoa. Käytössä on tunnetuimmat, Selenium, Cypress ja Appium-frameworkit. Mukana ovat myös Playwright, Puppeteer, Espresso, XCUI ja HyperExecute. (LambdaTest, 2022)

Lambdatest-työkalun kohdalla käytettiin Seleniumia ja sen testit ajettiin Java-koodikieltä käyttäen. Jotta testien suoritus ryhmittyisi hyvin, apuna käytettiin TestNG lisäosaa Eclipse IDE ympäristöön. Täten voidaan helposti annotoida/kommentoida testien vaiheet @Test kommenteilla.

5.2 Tehokkuus

Testien tekemiseen ei mennyt kauan aikaa, koska testit olivat jo valmiina Selenium WebDriver osion kautta. Testit tuli vain kopioida, järjestellä oikein ja muokata ne yhteensopiviksi TestNG formaatin kanssa. Pisin aika opettelussa meni itse pilvipalvelun ymmärtämiseen.

Testeissä käytettiin Selenium pohjaista testausta, joten käytössä oli kaikki työkalut, joilla Seleniumkin pyörii. Seleniumia käytettiin Java koodikielellä. LambdaTestin dokumentaatio sivujen kautta saatiin selville mitä eri työkaluja Selenium Java tukee, ja vaihtoehtoina ovat, Selenide, Gauge, TestNG, Geb, JUnit, Cucumber ja Serenity BDD. Testit suoritettiin TestNG toteutuksella.

Testausympäristön pystyttäminen tapahtui seuraavin vaihein:

1. Lataa ja asenna Eclipse.
2. Asenna Eclipseen plugin TestNG.
3. Luo Maven Projekti.
4. Lisää pom.xml tiedostoon kohta dependencies jonka sisään lisätään Maven Central Repositorystä Selenium Java ja TestNG.

Testit suoritettiin etänä pilvessä käyttäen Seleniumin tukemaa Remote Webdriveriä, joten pientä viivettä testeihin tuli verrattuna siihen, että testit olisi suoritettu omalla paikallisella työasemalla.

5.3 Käytettävyys

LambdaTest sivustoa oli kohtuullisen helppo käyttää. Kirjauduttua sisään palveluun, sivusto tarjosi helposti tulkittavan valikon, josta löysi tarpeelliset asiat. Käyttämisen aloittaminen oli helppoa, sillä sivustolla itsellään on tutoriaalitehtäviä, joilla voi päästä alkuun testien tekemisessä.

Sivusto oli houkuttelevan ja luotettavan näköinen. Dokumentaatiota ja opasvideoita työkalun käytöstä oli runsaasti. Integraatio ominaisuudet olivat helposti käytössä ja ne tulivat peruspaketin mukana hintaan $ 79 kuussa, joka laskutettiin vuosittain.

LambdaTest itsessään ei sisältänyt tekoälyavusteista self-healing ominaisuutta. Jos käyttöliittymä muuttuu suuresti, kaikki siihen liittyvät testit tuli uudelleen kirjoittaa. Tähän ongelmaan apuun olisi tullut integraation kautta algoQA. Aiheen rajauksen vuoksi jätettiin tämä käymättä läpi.

Sovellusta ylläpidetään kohtuullisen säännöllisesti, sillä päivityksiä tulee LambdaTest muutosloki sivujen mukaan useita kuukaudessa. Muutosloki sivu on järjestelty helposti luettavaan muotoon eri päivitys tyyppien avulla. (LambdaTest, ei pvm)

5.4 Kustannukset

LambdaTest sisälsi ilmaisversion manuaalista testausta varten, mutta siinä testien määrää rajattiin 60 minuuttiin kuukaudessa. Joka kuukausi käyttäjä sai uuden 60 minuuttia käyttöönsä. Tämä manuaalitestausmuoto ei ollut se mitä haetaan.

Automaatiotestaus maksaa $ 79 kuukautta kohden, joka maksetaan vuosittain. Tähän pakettiin sisältyy muun muassa loputtomasti automaatiotestaamista selaimilla, tuen kaikkiin Selenium versioihin ja tuen päälle 35 automaatio frameworkkiin. Tämä automaatiotestauksen tilaus pitää sisällään myös integraatio mahdollisuudet, loputtomasti geolokaatio testausta, tuen paikallisille tai yksityisesti hostatuille kotisivuille ja kolmannen osapuolen sovelluksien integraation. Kaiken tämän päälle saa vielä 24/7 tuen. Jokainen yksi Parallel test tuo mukanaan viisi mahdollista käyttäjää, ja hinta nousee sen $79 verran.

Alustasta on myös tarjolla kalliimpia vaihtoehtoja, joissa saa mukaan mobiilitestauksen ja oikeilla laitteilla testauksen. Mobiilitestauksen kanssa hinta nousee $ 99 kuukaudessa. Jos lisäksi haluaa oikeilla laitteilla testauksen, hinta kohoaa $ 128 kuukaudessa.

Hinnoittelut on jaettu järkevästi tarpeen mukaan. Skaalautuvuutta auttaa se, ettei tarvitse tilata semmoista mille ei ole käyttöä. Koska Lambdatest on pilvipalvelu, se mahdollistaa kustannustehokkaan testauksen. Testit suoritetaan vain silloin, kun niitä tarvitaan, eikä kehittäjien tarvitse ylläpitää omia testausympäristöjä.

6 SELENIUM WEBDRIVER

6.1 Esittely

Selenium on vapaan lähdekoodin automaatioframework. Sen myötä se on myös ilmainen. Selenium testi skriptejä pystyy kirjoittamaan monella eri koodikielellä, joita ovat mm. Java, C# ja Python. Seleniumin on kehittänyt Jason Huggins vuonna 2004. Tämä työkalu oli alun perin toteutettu tarpeesta automatisoida selaimen toimintaa. Ensimmäisen version nimi oli JavaScriptTestRunner. (Rungta, 2023)

Selenium on kehittynyt ajan myötä ja laajentanut toimintaansa, ja nykyään tarjolla on 3 eri versiota eritarkoituksiin. Selenium IDE, Selenium WebDriver ja Selenium Grid. Kullakin työkalun toteutuksella on oma tarkoituksensa.

Ensimmäisenä on Selenium IDE, joka on selaimen lisäosa, joka saadaan lisättyä Chrome tai Firefox selaimeen. IDE:n avulla saat tallennettua testiskriptin record & playback muodossa. Tämä on upea työkalu siltä osin, että Selenium skriptin syntaksin oppii helposti tämän kautta.

Toisena on Selenium WebDriver, jota tullaan käymään tässä dokumentaatiossa läpi. Selenium WebDriver ajaa selainta paikallisesti omalla koneella, tai jossain etätyöpöydällä käyttäen Selenium Serveriä. Skriptit koodataan itse käyttäen Seleniumin tukevaa koodikieltä.

Kolmantena on Selenium Grid, joka mahdollistaa testitapauksien ajamisen eri koneilla ja eri alustoilla. Kun WebDriverillä testien ajamiset on tehty ja olisi tarpeen tuoda mukaan systeemitestausta, Selenium Grid on vastaus. (Selenium, 2022)

Valinnaksi osui Seleniumin WebDriver, koska opinnäytetyössä on tarkoitus käydä monenlaisia eri työkaluja läpi, jotka käyttävät tätä pohjanaan. Joten Seleniumin syntaksi on hyvä tuntea entuudestaan, jotta muiden työkalujen käyttö olisi sujuvampaa. Selenium WebDriveriä voi käyttää monella koodikielellä, joita ovat Java, Python, C#, Ruby, JavaScript ja kotlin mobiilitestausta varten. Näistä päädyttiin Java koodikieleen.

6.2 Tehokkuus

Kehitysympäristön pystyttäminen oli helppoa. Seleniumin dokumentaation sivulla oli oivat ohjeet sitä varten. Jos sieltä ei aivan kaikki tieto tullutkaan, löytyi helposti seurattavia videotutoriaaleja. Kehitysympäristön pystyttäminen toteutui seuraavin askelin:

1. Lataa ja asenna Eclipse
2. Lataa Seleniumin sivulta Kirjastot
3. Lataa ChromeDriver sen omilta sivuilta.
4. Lisätään Selenium kirjastot External Libraryna Eclipse java projektiin.
5. Lisätään Chromedriver System.propertyn avulla.

WebDriverin opettelu toteutui vaivatta. IntelliSense antoi helposti apuja ja Seleniumin kirjastoista löytyi dokumentaatio, joten käyttö oli helppoa. Itse testien tekeminen ilmeni työlääksi ilman Selenium IDE:tä. Elementtien etsiminen ja niiden paikantimien löytäminen tuli tehdä manuaalisesti Xpath

ominaisuutta käyttäen. XPathin avulla otettiin kiinni verkkosivun elementeistä, käyttäen niiden attribuutteja haku parametreina.

Selenium WebDriver on itsessään todella tehokas ja nopea. Tämä tehokkuus tuottaa hieman tuskaa testejä suorittaessa. Seleniumin testit haluaisivat suorittaa koodin seuraavia rivejä jo, mutta sivu ei välttämättä ole latautunut. Tämän myötä testejä piti vähän jarrutella käyttäen Wait-ominaisuuksia. Näitä Seleniumilla on esimerkkinä kolmenlaisia, Implicit Wait, Explicit Wait ja Fluent Wait. Implicit Wait on odotuksen tyyppi, joka konfiguroidaan vain yhden kerran ja se pätee kaikille elementeille. Se määrää Seleniumia odottamaan tietyn määrän aikaa ennen kuin heittää virheilmoituksen, jos elementti ei ole löytynyt. Explicit Wait on odotuksen muoto, jossa määritellään, mitä se odottaa, kunnes skripti jatkuu. Esimerkiksi odotetaan niin kauan, että jokin elementti ilmestyy ruudulle. Fluent Waits on odotuksen muoto, jossa käytetään aikakatkaisinta ja väliä, jolla tarkistus tehdään.

Testit suoriutuivat erittäin nopeasti, sillä odotukset sai tehtyä juuri sellaisiksi, kun on tarve. Pahimpana vastuksena oli enää itse sivun nopeus tehdä asioita ja päivittää käyttöliittymää.

6.3 Käytettävyys

Käytettävyys on ohjelmoijalle erittäin hyvä, sillä kaikki toimii loogisesti. Kirjoitetaan koodia, joka toteutuu juuri niin kuin on tarkoitettu. Huonona puolena on se, että itsessään Selenium ei omaa tekoälyavusteista self-healing ominaisuutta. Jos testattava käyttöliittymä päivittyy, on testitkin päivitettävä samalla.

Työkalu ei välttämättä ole houkuttelevin vaihtoehto, mutta se oli hyvä oppia, sillä se toimi pohjana niin monelle muulle sovellukselle. Houkuttelevuutta lisäsi se, että se oli täysin vapaan lähdekoodin tuote, joten jatkokehitystä voi tehdä oman mielen mukaan.

Integraatio mahdollisuudet olivat itsemääriteltävissä. Apunaan pystyi käyttämään muiden palveluntarjoajien maksullisia palveluita, joilla Selenium saatiin helposti kiinni jatkuvaan integraatioon, tai sitten pystyi itse tekemään Selenium WebDriver Java ohjelmasta .jar paketin ja ajamaan sitä Jenkinsin putken kautta. Integraatio mahdollisuudet olivat hieman työläämmät kuin valmiissa työkaluissa.

Selenium on GitHubissa ylläpidetty sovellus, jota päivitetään aika säännöllisesti. Kyseessä on monien isojenkin yritysten suosima ja luotettu sovellus.

6.4 Kustannukset

Selenium WebDriver on täysin ilmainen. Ainoa mikä maksaa, on itse laitteet millä käytät sitä, ja lisäominaisuudet mihin sen haluat liittää muilta palvelun tarjoajilta.

7 GHOST INSPECTOR

7.1 Esittely

Ghost Inspector on helppokäyttöinen automaatiotestaustyökalu verkkosivuille ja verkkosovelluksille. Työkalu luotiin päämäärällä tarjota kattava ja helposti lähestyttävä palvelu vuonna 2014. (Klemm, ei pvm)

Ghost Inspector tarjoaa kahdenlaista tekniikkaa testaamiseen, aloittelijoille kooditonta ja kokeneelle testaajalle koodin kanssa end-to-end testausta. Kooditonta versiota käytetään recorderin kanssa, jonka Ghost Inspector on luonut selaimen laajennuksena. Tuettuja selaimia tällä hetkellä on Chrome ja Firefox (Ghost Inspector, ei pvm). Joidenkin arvioiden mukaan Ghost Inspector on paras valinta uusille automaatiotestauksen maailmaan tuleville (Capterra, ei pvm). Ghost Inspector on selaimella toimiva pilvipalvelu, jonka avulla luodaan/suoritetaan testejä ja monitoroidaan niitä (Stojanovski, 2017). Työkalulla on myös osittainen tuki Seleniumin ominaisuuksiin. Ghost Inspectorilla voi importata ja exportata Selenium IDE:n .side tiedosto formaattia (Ghost Inspector, 2021).

7.2 Tehokkuus

Työkaluun tutustuminen ja sen kykyjen ymmärtäminen veivät aikaa koska oppaita ei ollut paljon tarjolla. Työkalulla oli tarjolla oma dokumentaatio ja muutama video YouTubessa, mutta ei paljon muuta. Työkalun oppiminen ei ollut kovin vaikeaa, mutta hieman työlästä. Ghost Inspectorin yksinkertaisuus auttoi oppimista.

Työkalu suoritti testit pilvessä, joten suorituskyky riippui siitä. Testien raportti oli nähtävillä suorituksen jälkeen. Sen kattavuus on hyvä. Jokaisen vaiheen suoritus oli kirjattu ja tarjolla oli myös videoformaatti, jolla näkyy testien suoritus.

7.3 Käytettävyys

Työkalu oli käytettävyydeltään hieman kehno. Testit luotiin helpoiten käyttämällä Ghost Inspectorin omaa Web Recorder Chrome laajennusta. Laajennukseen kirjauduttiin sisään ja sillä pystyi nyt luoda testejä suoraan omalle tilille. Työkalulla oli vaikea tehdä suuria testejä tai testisarjoja, sillä testejä ei voinut kunnolla eritellä pienempiin osiin. Vaikka erittely olisi ollut mahdollista, jokaisen testin alussa tuli olla sivulle kirjautuminen. Tämä osoittautui ongelmalliseksi testeissä, joissa edellisen testin tietoja tarvittiin seuraavassa.

Työkalu ei muistanut edellistä testiä tai sitä mihin se jäi, vaan aloittaa tavallaan aivan uuden instanssin. Testit voivat kuitenkin käyttää jonkinlaista muistiominaisuutta, sillä tarjolla on testikohtaiset muuttujat, tai jopa Test Suite kohtaiset muuttujat. Nämä osoittautuivat hyödyllisiksi työkalua käyttäessä. Esimerkiksi, jos haluttiin luoda joku asia muuttujalla ja käyttää samaa muuttujaa myöhemmin tunnistamaan jonkun elementin olemassaolon, tapahtuma sarja olisi seuraavaa:

1. Luodaan testit ilman muuttujia, katsotaan vain silmä määräisesti missä lukee oikeita asioita. todennetaan niitä.
2. Luodaan pilvi ympäristöön globaali muuttuja mitä kaikki voi käyttää.

3. Muokataan testien niitä kohtia missä tarvitaan muuttujia, ja syötetään niihin kohti oikeat muuttujan nimet.

Työkalu tarjosi testien peräkkäisen suorittamisen, mutta toteutuksessa tehtiin kaikki testin vaiheet yhteen ainoaan testiin, sillä joka vaiheessa uudelleen kirjautuminen olisi tuottanut lisää turhia testivaiheita, mikä olisi väärentänyt tehokkuus tulosta jossain määrin.

Työkaluun oli tarjolla integraatio mahdollisuudet, joihin löytyi ohjeet työkalun dokumentaatiosta.

7.4 Kustannukset

Työkalusta oli tarjolla ilmainen testijakso, jonka pituus oli 14 vuorokautta. Alhaisin taso ”Small” oli tarjolla hintaan $109 / kuukausi. Siinä tuli mukana 10,000 testin suoritus kertaa kuukautta kohti, viisi tiimin jäsentä ja kuusi kuukautta tulosten säilytystä. Normaali ”Medium” taso maksoi $225 / kuukausi. Testien suorituskertoja 30,000 kpl. Tiimin jäseniä medium tasolla sai 15. Tulosten säilytys oli tässä tasossa kuusi kuukautta. Suuri ”Large” taso maksoi $449 /kuukausi. Testien suoritus kertoja oli 100,000 kappaletta. Tiimin jäseniä suuressa tasossa oli 40. Tulosten säilytystä tässäkin tasossa oli kuusi kuukautta. Tarjolla oli myös ”Enterprise” taso, jossa kaikki ominaisuudet sai sovittua erikseen.

8 BROWSERSTACK

8.1 Esittely

BrowserStack on pilvessä toimiva testaustyökalu web-sivuja ja mobiilisovelluksia varten. (Software Testing Help, 2023)

Työkalulla on käytössä tutuimmat frameworkit, Selenium, Cypress, PlayWright, Puppeteer ja JS Testing API. Myös mobiiliapplikaatio testaukseen on monta eri alustaa: Appium, Espresso, XCUITest, EarlGrey ja Flutter. (BrowserStack, ei pvm)

Tämän työkalun kohdalla käytettiin Cypress toteutusta. Cypress on käyttöliittymän testaustyökalu, samoin kuten Selenium, mutta ne eivät ole sama asia. Ne ovat pohjimmiltaan erilaisia sovelluksia. Cypressiä ei vaivaa samat ongelmat kuten Seleniumia, koska Cypress pyörii itse selaimessa. (Cypress, ei pvm)

BrowserStackilla on tarjolla montaa eri palvelua kuten Live, Automate, Percy, App Live, App Automate ja App Percy. Tässä dokumnentaatiossa tutkimme Automate sovelluksen ominaisuuksia.

8.2 Tehokkuus

Cypress itsessään oli todella nopea käyttöinen ja tehokas tuote. Tällä työkalulla automaatiotestaus oli todella helppoa. Koska koodikielenä oli JavaScript, se puhui hyvin sivuston kanssa. Cypressin asennus ei ollut vaikeaa ja testien luominen oli sujuvaa. Sovellus ei sisältänyt Record & playback toiminnallisuutta, mutta kun testejä suoritettiin itse sovelluksessa, voitiin käyttää helposti elementtien haku toiminnallisuutta ja saada suora koodi elementtiin.

BrowserStack on tehokkuudeltaan hyvä. Se suorittaa testit siinä missä muutkin. Huonona puolena löytyy se että, vaikka Cypress onkin nopea paikallisesti, kestää testien suoritus pidempään pilvessä.

8.3 Käytettävyys

BrowserStack tarjoaa automatisointi puolella ainoastaan raportti näkymän. Itse pilvestä ei saanut ajettua testejä. Testien ajot täytyi toteuttaa toista kautta, olipa sen käynnistys omasta projektista tai sitten putkessa jonkun toisen sovelluksen laukaisemana.

Testit luotiin paikallisesti itse omaan projektiin, ja siellä liitettiin omat BrowserStack tunnukset konfiguraatio tiedostoissa. Testit suuntasivat BrowserStackin pilvipalveluun ja suoritti testit siellä. Raporttinäkymässä tuli esille mitkä selaimet ovat saaneet suoritettua testit tai epäonnistuneet niissä.

Raportissa näkyi tarkasti koodin kohdat mitkä menivät pieleen ja niiden syyt. Testien suorituksesta tallentui video, missä Cypressin kohdalla näkyy myös se Cypress sovellus ja sen eri testi kohdat.

8.4 Kustannukset

Työkalusta oli tarjolla automaatiota varten kolmea eri versiota: Desktop, Desktop & Mobile ja Enterprice. Desktop version sai käyttöönsä hintaan $129 / kuukausi. Sen mukana tuli 1 parallel test, loputtomasti automaatiotestaajia, loputtomasti testaus minuutteja, päälle 3000 selainta ja jonkunlaiset integraatiomahdollisuudet.

Desktop & Mobile version sai hintaan $199 / kuukausi. Mukana tuli Desktop version ominaisuudet ja lisäksi seuraavat asiat: päälle 20 000 aitoa iOS ja Android laite yksikköä, Intregraatio kaikkien Appium versioiden kanssa, ”Localhost, Staging and private website testing”, omat työkalut nopeaan virheenkorjaukseen mobiililaitteissa, 19 palvelin keskusta 13:sta maalimanlaajuisessa paikassa, testaus oikean maailman skenaarioista, geolokaatiotestaus päälle sadasta eri maasta ja Google Analytics sisäinen liikenne pois.

Enterprise version hinnat sai neuvoteltua myyntihenkilöiden kanssa. Sen mukana tuli kaikki mitä aiemmissakin versioissa on käyty läpi, ja päälle vielä paranneltu käytettävyys, yksittäinen kirjautuminen, IP valkolistaus, paranneltu paikallinen testaus, käyttö analytiikat ja prioriteetti apu.

9 ROBOT FRAMEWORK

9.1 Esittely

Robot Framework on suomalainen avoimen lähdekoodin automaatiokehys. Sen avulla voidaan suorittaa automaatiotestausta ja robottien prosessien automaatiota. Robot Framework on hyvin laajennettava tuote, sillä se integroituu käytännössä kaiken kanssa. Kehyksen käyttö on ilmaista. Syntaksi Robot Frameworkissa on helppo, koska se käyttää ihmisluettavia avainsanoja. Alusta on helposti laajennettavissa erilaisilla kirjastoilla kuten python, Java tai monilla muilla koodikielillä. (Robot Framework, ei pvm)

Robot Framework tarvitsee toimiakseen Pythonin asennuksen.

9.2 Tehokkuus

Testeissä käytettiin Robot Framework alustan apuna myös Browser Librarya. Se itsessään käytti kielenään PlayWright koodikieltä. Testien koodeissa siis käytettiin kahta eri kieltä, Pythonia ja PlayWright:ia.

Tämä kyseinen työkaluyhdistelmä suoritti tehtävät paikallisesti omalla tietokoneella, mikä toi itsessään nopeutta testien suorittamiseen. Tämä työkalu suoriutui nopeudeltaan paremmin kuin Selenium WebDriver paikallisesti ajettuna.

Koodin oppimiseen meni oma aikansa, mutta sen jälkeen kirjoitus oli todellakin helppoa. Robot Framework itsessään käyttää ihmisluettavaa koodia, mikä vaikutti asiaan.

9.3 Käytettävyys

Työkalun käyttöönotto oli hieman hankalaa, sillä ei kerrottu kunnolla, miten aletaan käyttämään kyseistä koodia. Työkalu vaati vain pythonin asennuksen, ja sitä käyttämällä robot frameworkin ja browser libraryn asennuksen ”pip install” toiminnolla. Itse testit suoriutuvat yhdestä ainoasta tiedostosta. Täytyi luoda vaan <tiedoston nimi>.robot tiedosto ja suorittaa se komennolla ”robot <tiedoston nimi>.robot”.

Kielen oppiminen vei hieman aikaa, mutta työkalulla itsellään ja browser libraryllä on hyvät ohjeet, joiden avulla oppiminen suoriutui. Robot Frameworkin rakenne oli hieman erilainen mitä aiemmissa työkaluissa oli tullut vastaan, joten uudelleen orientoituminen tähän koodikieleen vei hieman aikaa.

Kun opit oli otettu talteen, testien kirjoittaminen sujui todella nopeasti. Koko testisarjan kirjoittaminen sujui muita toteutuksia nopeammin.

9.4 Kustannukset

Robot Framework oli ilmainen työkalu, ja niin oli myös Browser Library. Kustannuksia ei syntynyt sen osalta. Samoin kuten Seleniumin kohdalla, ainoat kulut mitä tämän työkalun kohdalla tulee, on oma laitteisto yms.

10 TELERIK TEST STUDIO

10.1 Esittely

Telerik Test Studio on sivujensa mukaan testiautomaatioalusta web-, työpöytä- ja responsiivisille web sovelluksille. Tarjolla on funktionaalisen UI:n testauksen, lataus/performanssi ja RESTful API testauksen. Tuote on suunniteltu sekä vähemmän kokemusta omaaville, että taitavampia automaatioinsinöörejä varten. Tämä työkalu sijoittuu visuaalisten tallennusvälineiden kategoriaan, mikä antaa käyttäjille mahdollisuuden luoda testejä record & playback toiminnon avulla. (Telerik, ei pvm)

Test Studio sisältää yksinomaisen tuen applikaatioille, jotka ovat rakennettu Telerik ja Kendo UI komponentteja käyttäen. (Telerik, 2023)

Test Studio käyttää omia puitteita sovelluksessaan, Telerik Testing Frameworkkia, eikä sovellus perustu ollenkaan Seleniumin päälle tai muiden avoimen lähdekoodin kehyksiin. Kehys käyttää C# ja Visual Basic koodikieliä. (Telerik, ei pvm)

10.2 Tehokkuus

Telerik Test Studion testejä ajaessa huomattiin, että tekstien syöttäminen automaattisesti tapahtui todella hitaasti, vain yksi kirjain kerrallaan. Tämä toiminnallisuus varmaan toi turvaa tekstin syötössä, mutta aiheutti todella paljon viivettä. Tähän yritettiin löytää korjaus tapaa nopeuttamaan syöttämistä, mutta etsinnässä epäonnistuttiin.

Testien suorituksessa oli vaihtoehto toteuttaa testit etäympäristössä, mutta siihen tarvittiin erillinen lisäosa nimeltä Runtime Add-on. Paikallisten testien suorittaminen tapahtui omassa valitsemassa verkkoselaimessa kuten Chrome tai Firefox. Testien suoritus kuitenkin tahtoi sulkea kaikki muut auki olevat selaimet, jotta suoritus voi alkaa.

10.3 Käytettävyys

Testien luominen oli helppoa tiettyyn pisteeseen asti. Käytössä oli Web Recorder, joka tallensi testien vaiheet, mutta sillä oli rajansa. Näiden testien muuttaminen ”Data Driven” testeiksi oli hieman työlästä. Sen toteuttaminen saatiin aikaan luomalla erilaisia ulkoisia datalähteitä, ja liittämällä ne osaksi testejä.

Recorderilla elementtien luominen oli näppärää, mutta vastaan tuli ongelma, kun elementtejä piti löytää. Recorderilla pystyi tekemään tarkastuksia, onko jokin teksti olemassa, mutta kun se oli valittu recorderin kautta, se tarrautui johonkin luotuun elementtiin, missä paikantimet eivät täsmänneet enää seuraavalla kertaa. Syynä tietysti se, että testidatan muuttujat muuttuvat, kuten jotkin identifioivat aikaleimat. Teksti on kyllä selaimessa näkyvissä, mutta elementin paikantimet osoittivat edelliseen listalla löytyvään alkioon. Tämän ongelman ratkaiseminen oli haastava toteuttaa, sillä sivulta dynaaminen hakeminen ei onnistunut helposti. Dokumentaatiota tästä ei paljon ollut, mutta pitkän etsinnän jälkeen vastaus löytyi. Toteutustapoja oli kaksi: joko luoda jokin datalähde, jota käytettiin ja liitettiin elementteihin erilaisin muuttujin, tai toteuttaa elementin etsintä koodissa. Työssä

päädyttiin käyttämään jälkimmäistä vaihtoehtoa, koska datalähteiden kautta toteuttaminen oli todella työlästä.

Työkalun IDE oli helppo ymmärtää ja käyttää. Koodikielenä toimi C# tai Visual Basic. Testien onnistuessa IDE tarjosi raportin paikallisesti, missä ei näkynyt paljoa. Jos testit epäonnistuvat, joissakin tilanteissa näytettiin kuva, missä kohtaa testi meni pieleen. IDE näytti testejä rakentaessa virhe ilmoituksen, jos koodissa oli jotain vikaa. Työkalulla oli myös ominaisuus, jossa voitiin nähdä testien jokaisen vaiheen suoritus ja elementtien etsintä. Tämä oli todella hyödyllinen etsiessä syytä, miksi testi ei löydä oikeaa elementtiä.

Työkalun dokumentaatio oli aika laaja, mutta suurin osa keskittyi vain toiminnallisuuksien käyttöön ilman koodia. Koodistakin oli joitakin dokumentin osia, mutta kunnon oppaita ei. Joistakin dokumentin artikkeleista ei tahtonut olla apua, koska informaatio oli niin suppea.

Työkalulla oli mahdollisuus CI/CD integraatioon mikä mahdollisti esimerkiksi Azure Devopsiin tai Jenkinsiin yhdistämisen. Tämän opinnäytetyön aikana ei keritty testaamaan integraatio ominaisuuksia.

10.4 Kustannukset

Telerik Test Studiosta löytyi kahta eri versiota ja ostettava lisäosa. Ensimmäinen, halvempi vaihtoehto oli Test Studio Web & Desktop, mikä maksoi $ 2,499. Sen mukana tuli yleisimmät ominaisuudet, luo ja aja testejä. Pakettiin sisältyi myös Standalone IDE, Test Scheduler, CI/CD-integraatio ja Remote Execution mahdollisuus. Jotta etäsuoritus toimisi, RunTime Add-on tuli ostaa erikseen, joka maksoi $ 349.

Test Studio Ultimate versiossa tuli lisäksi mukana API testaus, web pohjainen dashboard jolla näki testien raportit ja yksi Runtime edition lisenssi. Tämä tuli maksamaan tonnin enemmän eli yhteensä $3,499. Kaikissa tuotteissa oli ikuinen lisenssi.

11 YHTEENVETO

11.1 Tulokset

Tässä opinnäytetyössä käytiin läpi 7 eri työkalua automaatiotestaukseen liittyen. Jokaiseen työkaluun tutustuttiin käytettävyyden, tehokkuuden ja kustannusten kautta. Käytettävyyttä testattiin suorittamalla jokaisella työkalulla testisarja ja kirjattiin ylös, mikä oli hyvää ja mikä huonoa. Tehokkuutta mitattiin ottamalla aikaa testisarjan suorituksesta ja siitä, miten kauan kesti luoda testit kyseisellä työkalulla. Kustannukset etsittiin kunkin työkalun nettisivuilta ja tutkailtiin mitä kaikkea perus- ja premium-tilauksella saa.

Tulokset on kasattu pisteineen seuraavaan taulukkoon kokonaispiste järjestyksessä.

Työkalu	Käytettävyys	Tehokkuus	Kustannukset	Pisteet yhteensä
Ghost Inspector	7	4	2	**13**
Telerik Test Studio	11	7	4	**22**
Katalon Studio	17	4	2	**23**
Robot Framework	12	7	5	**24**
BrowserStack	17	7	2	**26**
Selenium WebDriver	16	7	5	**28**
Lambdatest	18	7	5	**30**

Taulukko 2. Tulokset

Kukin pisteiden otsikko sisältää eri kriteerit, joiden avulla pisteisiin päästiin. Käytettävyydessä mitattiin mm. helppokäyttöisyyttä, houkuttelevuutta, työnmäärää jotta jatkuva kehitys toimii ja ylläpidetäänkö sovellusta enää. Tehokkuutta mitattiin opettelun pituudella ja testien suoritusnopeudella. Kustannuksia mitattiin yksinkertaisesti työkalun perushinnalla ja satunnaisen lisäpisteen sai, jos työkalusta löytyi jotain muihin nähden ylivoimaista.

Kärkikolmikoksi osoittautui Lambdatest, Selenium WebDriver ja BrowserStack. Kunkin työkalun käytettävyys oli korkealla, osittain siksi että työkalut olivat vain pilvipalveluita (pois lukien Selenium WebDriver). Pilvipalvelut mahdollistivat testien tekemisen haluamallaan koodaustyylillä, joita on useita. Pilvipalvelut tukivat näitä tutuimpia automaatio työkaluja kuten Selenium ja Cypress. Näiden kahden pilvipalvelun jatkuvan kehityksen mahdollisuudet olivat suuret ja dokumentaation mukaan erittäin helpot käyttää. Selenium WebDriver oli yksinomaan vain itse kehitettävä testien suorittaja. Se ei itsessään ollut mikään palvelu, vaan itsenäinen rakenne, jota muotoiltiin haluamallaan tyylillä. Selenium oli käytettävyydeltään todella hyvä, hinnaltaan ilmainen ja tehokkuudeltaan hyvä.

Kummassakin kärkikolmikon pilvipalvelussa oli monipuolinen tuki eri toteutus kehyksille. BrowserStackissa päädyttiin käyttämään Cypressiä mikä oli JavaScript pohjainen toteutus. Lambdatest pilvipalvelussa käytettiin Selenium WebDriveriä joka ajoi TestNG testit. Selenium WebDriver testit

ajettiin käyttäen puhdasta Java kieltä ilman TestNG mahdollisuuksia. Tämä koodikielien valinta mahdollisuus oli osasyy sille, että ne ovat nyt kärkikolmikossa. Muut työkalut pakottivat käyttämään jotain tiettyä toteutustyyliä. Kuten Katalon Studion osalta oli pakko käyttää juuri sitä sovellusta ja sen koodikieltä. Ghost Inspectoria käyttäessä oli pakko käyttää niiden tuottamaa selaimen lisäosaa testien tallentamiseen.

Se, mikä, nosti erityisesti nämä kaksi pilvipalvelua (BrowserStack ja Lambdatest) muiden yläpuolelle, oli se, että ne ottivat videon testisarjan suorituksesta. Tämän videon pystyi lataamaan työkalujen raporttinäkymästä ja sitä pystyy käyttämään hyödykseen muissa tarkoituksissa. Työkalut olivat myös kohtuullisen halpoja muihin nähden. Molemmat saivat hinnaltaan samat pisteet, mutta LambdaTest ansaitsi raporttinäkymällään lisäpisteitä.

Opinnäytetyössä onnistuttiin löytämään monipuolinen valikoima työkaluja. Alkujaan mahdollisten työkalujen lista oli 42 alkiota pitkä, lopulta erilaisten karsintakierrosten kautta päädyttiin perehtymään seitsemään eri työkaluun ja toteuttamaan testit kullakin. Saimme kokoon näistä pisteytysten kautta kärkikolmikon. Tämän myötä opinnäytetyön tavoitteeseen päästiin.

Jatkokehityksen suhteen tämän työn avulla päästiin siihen pisteeseen, että saatiin hyvä tietämys kustakin sovelluksesta, ja tieto siitä millä työkalulla mahdollisesti olisi hyvä toteuttaa jatkossa suurempi määrä testejä web-sivuille.

11.2 Oman työn arviointi

Työssä saavutettiin sitä varten asetetut tavoitteet. Näitä tavoitteita olivat yleiskäsitys automaation käytöstä ohjelmisto kehityksessä, laajempi ymmärrys käsiteltävistä työkaluista ja niistä parhaimman löytäminen. Opinnäytetyössä pohdittiin työkalujen tehokkuutta, käytettävyyttä ja kustannuksia. Työkaluja vertailtiin niiden avulla ja saatiin lopputulokseksi pisteytetty taulukko.

Työn etenemisprosessi toteutui hieman suunnitellusta poiketen. Opinnäytetyön käytännönosuus ja kirjallinen raportointi tuli tapahtua erikseen, mutta työtä tehdessä kävi helpommaksi tuottaa raporttia päiväkirjan lailla. Kustakin työkalusta kirjattiin päiväkirjaa ja muistiinpanoja yhtä aikaa testauksen ohella.

Opinnäytetyössä lähteinä käytettiin monia erilaisia toteutuksia. Mukana oli asia tekstiä, artikkeleita, työkalujen osalta kunkin omaa dokumentaatiota, yleisesti ottaen luotettujen arviointisivujen tekstejä ja toisinaan niiden sivujen kommentteja. Tässä dokumentissa esitettyjä tuloksia tulisi ottaa vastaan kritiikki mielessä. Tehokkuus ja kustannukset ovat fakta peräisiä aiheita, mutta käytettävyys on oman käyttökokemuksen perusteella toteutettu ja se voidaan ottaa joissain tapauksissa jopa mielipidekirjoituksena.

Tutkimus osoitti, että suuresta työkalujen kirjosta löytyi tiettyjen kriteerien avulla kärki joukkio. Työkaluja oli monenlaisia, koodattavia ohjelmia ja selainpohjaisia toteutuksia. Kussakin työkalussa löytyi omat hyvät ja huonot puolensa. Koska opinnäytetyö ja tutkimus olivat rajattu ainoastaan web-sivujen testaamiseen tietokoneella, jäi pois esimerkiksi mobiilitestaus. Tämän rajauksen myötä tästä työstä ei tullut liian laaja, vaan saatiin tuotettua keskitetty tutkimus.

Opinnäytetyön tekeminen oli minulle hyvin opettavainen tilanne. Opin suorittamaan suurempi mittaista tutkimusta määräajassa ja asettamaan asioita tärkeysjärjestykseen. Olen tyytyväinen tutkimuksessa suoritettuun testaus vaiheeseen, sillä opin perusteita uusista koodikielistä. Koin haasteita työn kirjoitus osuudessa ja sen läpiviennissä, sillä itselleni asettamani askeleet olivat suuria. Tulevaisuudessa tulen asettamaan askeleet hieman pienemmiksi, jotta ne olisi helpompi saavuttaa.

LÄHTEET

BrowserStack. (ei pvm). *Documentation*. Haettu 1. 3. 2023 osoitteesta BrowserStack: https://www.browserstack.com/docs/

Capterra. (ei pvm). *Ghost Inspector Reviews*. Haettu 20. 2. 2023 osoitteesta Capterra: https://www.capterra.com/p/156853/Ghost-Inspector/reviews/

Cypress. (ei pvm). *Why Cypress*. Haettu 13. 3. 2023 osoitteesta Cypress: https://docs.cypress.io/guides/overview/why-cypress

Ghost Inspector. (26. 6. 2021). *How to use Selenium IDE with Ghost Inspector*. Haettu 20. 2. 2023 osoitteesta Ghost Inspector: https://ghostinspector.com/blog/selenium-ide/

Ghost Inspector. (ei pvm). *Resources*. Haettu 20. 2. 2023 osoitteesta Ghost Inspector: https://ghostinspector.com/docs/

Gillis, A. S. (9. 2019). *Automated Testing*. Noudettu osoitteesta TechTarget: https://www.techtarget.com/searchsoftwarequality/definition/automated-software-testing

Hangasmaa, H. (2010). Opinnäytetyö. *Käytettävyyden merkitys ja käytettävyystestaus*. Noudettu osoitteesta https://urn.fi/URN:NBN:fi:amk-2010062112421

Hautala, M.;& Peltonen, H. (19. 8. 2014). *Dynamiikka* (11. painos p.). Saarijäervi: Lahden Teho-Opetus Oy. Haettu 13. 4. 2023

ite wiki. (ei pvm). *Mitä eroa on ohjelmistorobotiikalla ja testausautomaatiolla?* Haettu 11. 4. 2023 osoitteesta Ite wiki: https://www.itewiki.fi/p/mita-eroa-on-ohjelmistorobotiikalla-ja-testausautomaatiolla

Katalon. (ei pvm). *A Deep Dive into Selenium, Its Alternative Solution for 2022 and Beyond*. Haettu 13. 2. 2023 osoitteesta Katalon: https://katalon.com/resources-center/blog/selenium-alternative-solution

Katalon. (ei pvm). *About Katalon Platform*. Haettu 13. 2. 2023 osoitteesta Katalon: https://docs.katalon.com/docs

Klemm, J. (ei pvm). *What Is Ghost Inspector? Overview & Tour Of Features*. Haettu 20. 2. 2023 osoitteesta QA Lead: https://theqalead.com/test-management/what-is-ghost-inspector-overview-tour-of-features/

LambdaTest. (18. 11. 2022). *LambdaTest Support and Knowledge Base*. Haettu 17. 2. 2023 osoitteesta LambdaTest: https://www.lambdatest.com/support/docs/

LambdaTest. (2023). *Cross Browser*. Haettu 17. 2. 2023 osoitteesta LambdaTest: https://www.lambdatest.com/

LambdaTest. (ei pvm). *Changelog*. Haettu 17. 2. 2023 osoitteesta LambdaTest: https://changelog.lambdatest.com/

Robot Framework. (ei pvm). *Introduction*. Haettu 15. 3. 2023 osoitteesta Robot Framework: https://robotframework.org/

Rungta, K. (11. 2. 2023). *What is Selenium? Introduction to Selenium Automation Testing*. Haettu 17. 2. 2023 osoitteesta guru99: https://www.guru99.com/introduction-to-selenium.html

Selenium. (29. 8. 2022). *Documentation*. Haettu 17. 2. 2023 osoitteesta Selenium: https://www.selenium.dev/documentation/

Software Testing Help. (13. 2. 2023). *Browserstack Tutorial: App And Browser Testing Platform [GUIDE]*. Haettu 1. 3. 2023 osoitteesta Software Testing Help: https://www.softwaretestinghelp.com/browserstack-tutorial/

Software Testing Help. (19. 1. 2023). *What Is Integration Testing (Tutorial With Integration Testing Example)*. Haettu 14. 2. 2023 osoitteesta Software Testing Help: https://www.softwaretestinghelp.com/what-is-integration-testing/

Stojanovski, A. (27. 10. 2017). *Ghost Inspector – Cloud-Based solution for Web Automated Testing (Part I)*. Haettu 20. 2. 2023 osoitteesta IW Connect: https://iwconnect.com/ghost-inspector-cloud-based-solution-web-automated-testing-part/

Telerik. (22. 2. 2023). *About Telerik Test Studio Ultimate*. Haettu 21. 3. 2023 osoitteesta Component Source: https://www.componentsource.com/product/telerik-test-studio/about

Telerik. (ei pvm). *Frequently Asked Questions*. Haettu 21. 3. 2023 osoitteesta Telerik: https://www.telerik.com/teststudio#what-is-telerik-test-studio

Telerik. (ei pvm). *Welcome to Test Studio*. Haettu 21. 3. 2023 osoitteesta Telerik Docs: https://docs.telerik.com/teststudio/welcome

Tenhunen, M.-L. (12. 3. 2013). *Johdon laskentatoimen peruskäsitteet, menetelmät ja tekniikat – osa 2*. Haettu 15. 2. 2023 osoitteesta https://tilisanomat.fi/koulut/johdon-laskentatoimen-koulu-koulut/johdon-laskentatoimen-peruskasitteet-menetelmat-ja-tekniikat

The New Stack. (1. 3. 2019). *How LambdaTest Automates Cross-Browser Testing from the Cloud*. Haettu 17. 2. 2023 osoitteesta The New Stack: https://thenewstack.io/how-lambdatest-automates-cross-browser-testing-from-the-cloud/

Valagroup. (10. 11. 2022). *Mitä on ohjelmistotestaus ja mitä hyötyä siitä on?* Haettu 14. 2. 2023 osoitteesta Valagroup: https://www.valagroup.com/fi/blogi/mita-on-ohjelmistotestaus-ja-mita-hyotya-siita-on/?utm_term=ohjelmistotestaus&utm_campaign=Traffic&utm_source=adwords&utm_medium=ppc&hsa_acc=9088882292&hsa_cam=812356547&hsa_grp=143456284203&hsa_ad=636870501771&hsa_src=g

Wikipedia. (4. 2. 2023). *Yksikkötestaaminen*. Haettu 14. 2. 2023 osoitteesta Wikipedia: https://fi.wikipedia.org/wiki/Yksikk%C3%B6testaaminen

Yerukala, M. (ei pvm). *What is Katalon Studio - Complete Tutorial Guide*. Haettu 13. 2. 2023 osoitteesta MindMajix: https://mindmajix.com/katalon-studio-tutorial

Kasurinen, J. P. (2013). Ohjelmistotestauksen käsikirja (1. p.). Docendo.

Kuva 1 Testauksen v-malli. Muokattu (Kasurinen, 2013) 7

Kuva 2. Project Settings Katalon studiossa (Katalon Studio, 2023) 15

Kuva 3. Manual Mode Katalon studiossa (Katalon Studio 2023) 15

Kuva 4. Script mode Katalon studiossa (Katalon Studio 2023) 16

Kuva 5. Record mode Katalon studiossa (Katalon Studio 2023) 16

Kuva 6. Plugin menubar Katalon studiossa (Katalon Studio 2023) 17