jamk

Cypress
Muster-katsastusjärjestelmän testausautomaatiossa

Jere Liikka

Opinnäytetyö, AMK
12/2021
Tietojenkäsittely ja tietoliikenne
Tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Liikka, Jere

Cypress Muster-katsastusjärjestelmän testausautomaatiossa

Jyväskylä: Jyväskylän ammattikorkeakoulu. **Joulukuu 2021**, 30 sivua

Tekniikan ala. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: Kyllä

Tiivistelmä

Opinnäytetyön tarkoituksena oli tutkia, voisiko Cypress korvata Pinja Muster Oy:n Muster-katsastusjärjestelmässä käytettävän Seleniumin web-käyttöliittymän testausautomaatiotyökaluna. Tärkeimpinä vertailukohteina olivat ylläpidettävyys, luotettavuus ja suorituskyky. Tavoitteena siis oli vertailla Cypressia ja Seleniumia edellä mainittujen vertailukohteiden perusteella. Opinnäytetyön alussa käydään läpi ohjelmiston testaamisesta yleisesti sekä kerrotaan Cypressin ja Seleniumin taustojen lisäksi myös muista testausatuomaatiotyökaluista.

Tutkimus toteutettiin luomalla kehitysympäristö, jossa testejä pystyi ajamaan molemmilla työkaluilla. Kehitysympäristöön otettiin käyttöön paikallinen versio Muster-katsastusjärjestelmästä, jolloin testien pyörittäminen onnistui suoraan kehityskoneelta. Testien läpäisyprosentit sekä suoritusaika kirjattiin ylös ja työkaluja verrattiin näiden tulosten perusteella. Tutkimus tehtiin kvantitatiivisena tutkimuksena.

Tutkimus osoitti, että kokonaisuutena Cypressilla tuotetut testit ovat parempia kuin Seleniumilla tuotetut testit. Cypressilla luotujen testien suoritusaika oli huomattavasti parempi kuin Seleniumin, muut tulokset (ylläpidettävyys ja luotettavuus) olivat toisiaan vastaavat molemmilla työkaluilla.

Avainsanat (asiasanat)

Testausautomaatio, Cypress, Selenium

Muut tiedot (salassa pidettävät liitteet)

Liite 1, koodinäytteet 1, 2, 3, 4, 5, kuvio 6 sekä Musterin front-endin käynnistyskomento ovat salassa pidettäviä, ja ne on poistettu julkisesta työstä.

jamk

Description

Liikka, Jere

Using Cypress in test automation for Muster application

Jyväskylä: JAMK University of Applied Sciences, December 2021, 30 pages

Engineering and technology. Degree Programme in Information and Communications Technology. Bachelor's thesis.

Permission for web publication: Yes

Language of publication: Finnish

Abstract

The purpose of the study was to find out if Cypress could replace Selenium as Pinja Muster Oy's Muster vehicle inspection system's test automation tool. The key points to be compared were sustainability, dependability, and performance. The main objective was to compare Cypress and Selenium with the given key points. The thesis begins with a theoretical part that explains test automation and testing a software in general. In addition, the theoretical part also discusses test automation tools in general and specifically Cypress and Selenium.

The thesis was carried out as research-based development. The research section was conducted by creating an environment, where testing could be performed with both test automation tools. The environment was fitted with a local version of Muster so that the Muster and the tests could be run inside the development environment. Both passing rates and execution times for Cypress and Selenium were written down and then compared with each other.

The results showed that tests conducted with Cypress were in general more desirable than those made with Selenium. The execution times of Cypress tests were remarkably better than those with Selenium; however, the other results - sustainability and dependability - were in correspondence with each other.

Keywords/tags (subjects)

Test automation, Cypress, Selenium

Miscellaneous (Confidential information)

Attachment 1, code snippets 1, 2, 3, 4, 5, figure 6 and the command to start Muster's front-end are confidential and have been removed from the public thesis.

Sisältö

Kuviot

Koodinäytteet

Taulukot

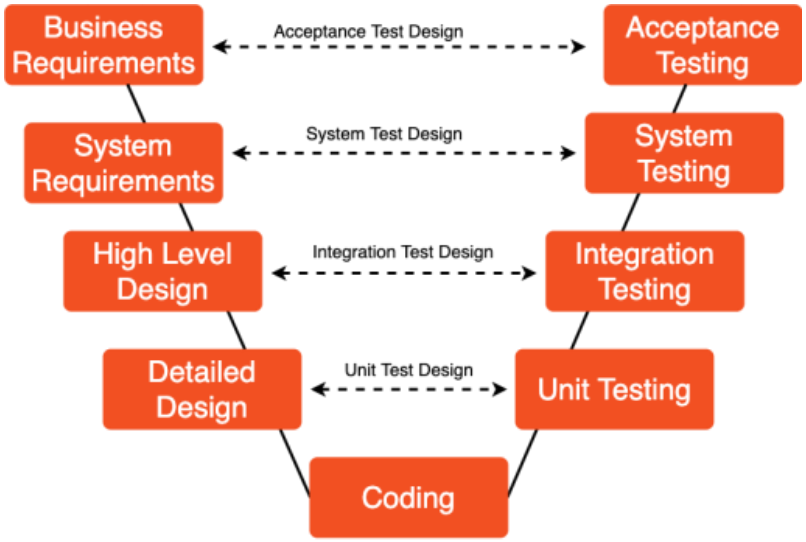
1 Johdanto

1.1 Ohjelmiston testaaminen

Ohjelmiston testaaminen on tärkeässä osassa ohjelmistokehitystä. Ohjelmiston testaaminen mahdollistaa ohjelmiston toimivuuden sekä sen avulla pystytään havaitsemaan mahdolliset viat. Vikojen esiintymistä ohjelmiston kehitysvaiheessa on lähes mahdoton estää ja siksi niiden etsiminen testaamisen avulla on tärkeää.

Ohjelmistoa testattaessa tärkeässä roolissa on testaamisen järkevä suunnittelu. Eräs suosittu testaamisen malli on V-malli (eng. V-model). Tässä mallissa ohjelmistoa testataan jo sen ensimmäisien ominaisuuksien valmistuttua, toisin kuin vesiputousmallissa (eng. waterfall model). Ohjelmistotestauksen vesiputousmallissa testaaminen tapahtuu vasta vaatimusmäärittelyn, suunnittelun ja itse ohjelmiston tuottamisen jälkeen. Vesiputousmallissa ongelmana on se, että jo määrittely- ja suunnitteluvaiheessa voi ilmetä virheitä, joita ei sillä hetkellä huomata ilman testaamista. Huonoimmassa tapauksessa tämä voi johtaa siihen, että tuotettu toteutus ei vastaa alkuperäistä tarvetta, ja projekti joudutaan aloittamaan uudelleen. Mitä aikaisemmin ongelmat huomataan, sitä kustannustehokkaampaa niihin on puuttua. Jotta tällaisilta tilanteilta vältyttäisiin, ohjelmistotestauksen V-malli tuo testausvaiheen jokaiseen ohjelmiston tuotannon vaiheeseen. (Hamilton 2021b.)

Kuviossa 1 on esitetty ohjelmistotestauksen V-malli. Kuviossa vasemmalla puolella on ohjelmiston tuotannon eri vaiheet ja oikealla puolella testauksen eri vaiheet.



Kuvio 1: Ohjelmistotestauksen V-malli (SDLV - V Model)

Ohjelmistoa on mahdollista testata kahdella tavalla; ensimmäisenä vaihtoehtona on testata ohjelmistoa käsin ja toisena vaihtoehtona on käyttää testaamiseen ennalta määriteltyä testausautomaatio-ohjelmistoa. Monessa tapauksessa myös käsin tehtävät testit, erityisesti käyttöliittymän käytettävyyteen liittyvät testit, ovat tärkeässä roolissa, mutta monessa tilanteessa käsin tehtävä ohjelmiston testaus on todella aikaa vievää eikä se ole tarpeellista. (What Is Test Automation? A Simple, Clear Introduction 2019.)

Asiakastyytyväisyys on suuressa roolissa koko ohjelmiston kehityksen ajan ohjelmistontuottajan näkökulmasta. Jos järjestelmä ei toimi, asiakkaat eivät ole tyytyväisiä. Tästä syystä mahdollisten vikojen etsiminen ennen ohjelmiston julkaisua on äärimmäisen tärkeää. Sama pätee myös kaikkiin päivityksiin, joita ohjelmistoon sen elinkaaren aikana tulee. Se, että ohjelmisto toimii luotettavasti jatkuvalla tahdilla, on hyvää mainosta myös mahdollisia tulevia asiakkaita ajatellen. Jos ohjelmisto taas ei toimi luotettavasti, asiakkaat saattavat kääntyä jonkin vastaavan palveluntuottajan puoleen.

1.2 Toimeksiantaja

Tämän opinnäytetyön toimeksiantajana toimi Pinja Muster Oy, joka on osa Pinja konsernia. Pinja on Suomalainen 1990-luvulla perustettu yritys, joka työllistää tällä hetkellä noin 400 työntekijää.

Pinja on keskittynyt digitalisaation sekä teollisuuden uudistukseen, pääasiallisina toimialoina hyvinvointi- ja terveysteknologia, meriteollisuus, energiateollisuus, logistiikka, puunjalostusteollisuus ja asiantuntija- ja palvelusyritysten digitalisaatio. Pinja palvelee teollisuus- sekä yritysasiakkaita Suomessa sekä yli 30 maassa. Liikevaihtoa Pinjalla oli vuonna 2020 50 miljoonaa. (Pinja 2021.)

1.3 Tutkimusmenetelmä

Tämä opinnäytetyö on toteutettu tutkimuksellisena kehittämistyönä. Tutkimuksellisen kehittämistyön tarkoituksena on ratkaista ongelmia hyödyntämällä systemaattisesti tuotettua tietopohjaa käytännössä. Pääasiallisena tavoitteena tutkimuksellisessa kehittämistyössä on luoda uusia ideoita, käytäntöjä, tuotteita tai palveluita. Työn toteutusmenetelmän ansiosta työskentely on järjestelmällistä sekä analyyttistä, mikä saa tarkastelemaan tutkittavaa kohdetta myös kriittisesti. Kehittämistyössä tutkimuksellisuus on tärkeässä roolissa, sillä näin myös tulokset ovat perusteltavissa paremmin. (Kehittämistyön menetelmät tukena opinnäytetyössä 2018; Suvanto 2014, 13.)

2 Työn tavoitteet

2.1 Tehtävä ja tavoitteet

Musterissa on tällä hetkellä käytössä Selenium-pohjainen automaatiotestausjärjestelmä, jonka avulla järjestelmää testataan mahdollisten vikojen varalta esimerkiksi ennen päivitysten julkaisua. Kaikki päivitykset julkaistaan ensin testiympäristöön, jossa pystytään testaamaan järjestelmää mahdollisten vikojen varalta. Tässä kohtaa automaatiotestaus tulee mukaan. Koko järjestelmän toimivuuden testaaminen käsin veisi huomattavasti aikaa, joten on kannattavampaa antaa valmiiksi suunnitellun järjestelmän käydä ennalta määritetyt testit läpi huomattavasti lyhyemmässä ajassa. Tällä hetkellä käytössä olevassa testausautomaatiojärjestelmässä on kuitenkin havaittu olevan ongelmia testien luotettavuuteen liittyen. Testien tulokset ovat epäsäännöllisiä ja testit saattavat antaa vääriä tuloksia. Tämän opinnäytetyön tarkoituksena olikin tutkia mahdollisuuksia vaihtoehtoisen testausautomaatioteknologian käytölle Muster-katsastusjärjestelmässä.

Muster-katsastusjärjestelmä on järjestelmä, jota sadat katsastusasemat ympäri Suomen käyttävät päivittäin katsastusaikavarausten vastaanottamiseen, autojen katsastamiseen, asiakkuuksien hal-

lintaan, raportointiin sekä rahaliikenteen hallintaan (Muster by Pinja, n.d.). Opinnäytetyön tekohetkellä Muster-katsastusjärjestelmän automaatiotestaukseen käytettiin Selenium-pohjaista testausautomaatioteknologiaa ja tilanne oli se, että haluttiin tutkia mahdollisia vaihtoehtoja.

Muster on SaaS-palvelu, eli Software as a Service -palvelu. SaaS-palvelu on palvelu, jossa asiakkaille myydään jo olemassa olevaan palveluun käyttöoikeutta korvausta vastaan. Hyvä esimerkki SaaS-palvelusta on suoratoistopalvelu Netflix (Chai & Casey 2021). SaaS-palvelut toimivat pilvessä ja ovat näin loppukäyttäjän saatavilla helposti. Muster toimii Microsoftin Azuressa.

2.2 Tutkimuskysymykset

Ennen tämän tutkimuksen aloittamista käytiin työn toimeksiantajan kanssa palaveri tutkimuksen sisällöstä sekä sen tavoitteista. Pääasiallisiksi kysymyksiksi nousivat:

- Kuinka helposti ylläpidettäviä Cypressilla tuotetut testit ovat?
- Ovatko Cypressilla luodut testit luotettavia?
- Ovatko Cypressilla luodut testit tarpeeksi nopeita?

Ylläpidettävyys: Kuinka helposti ylläpidettäviä Cypressilla tuotetut testit ovat?

Jotta testausautomaatiojärjestelmän käyttö ja ylläpito olisi kannattavaa, on testien oltava hyvin järjesteltävissä sekä suunniteltavissa. Hyvä esimerkki tästä on se, että itse testitapaukset ovat määriteltynä omassa sijainnissaan ja testikomentojen määrittelyt omassa tiedostossa. Näin jos testattavan järjestelmän jokin ominaisuus muuttuu ja testit eivät enää päde, voidaan helposti käydä muuttamassa testikomentojen määrittelyjä erillisestä tiedostosta koskematta itse testitapauksiin.

Luotettavuus: Ovatko Cypressilla luodut testit luotettavia?

Tällä hetkellä ongelmana Musterilla on se, että käytössä olevat testit antavat vääriä tuloksia, eikä niihin ole aina voinut luottaa. Eli esimerkiksi on tapauksia, joissa testiraportilla on nähtävissä, että osa testeistä ei mennyt läpi, vaikka todellisuudessa järjestelmä toimii testatuilta osa-alueilta. Tämä

luo ongelman, koska ohjelmistonkehittäjien täytyy käydä läpi jokainen epäonnistunut testitapaus ja se kuluttaa aikaa sekä resursseja. Joissain tapauksissa voi jopa olla, että kaikkia testitapauksia ei ehditä käymään läpi ennen tuotantoon julkaisua ja on olemassa riski, että julkaistussa tuotteessa on vikoja.

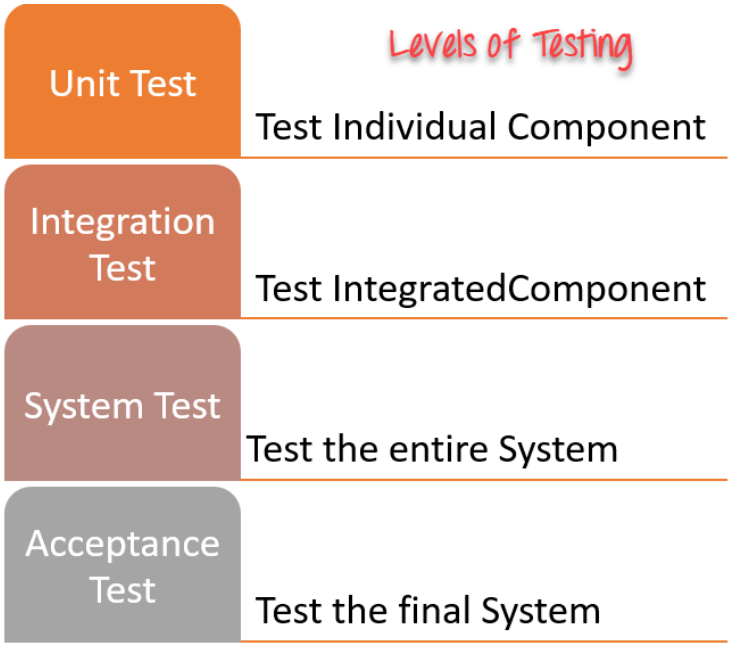
Suorituskyky: Ovatko Cypressilla luodut testit tarpeeksi nopeita?

Musterissa tällä hetkellä käytössä olevien testien suorittamiseen on käytössä testausautomaatiotyökalulle dedikoitu palvelin, joka aktivoidaan aina, kun testejä halutaan suorittaa. Jokainen minuutti maksaa Musterille, joten aika, jonka testausautomaatiotyökalun palvelin on käynnissä, halutaan pitää minimissä. Tällä hetkellä yhtenä ongelmana on se, että testitapauksien suorittaminen vie aika-ajoin huomattavasti enemmän aikaa kuin mitä testeiltä odottaisi. Tämä liittyy osin testien luotettavuuteen, mutta haluttiin pohdittavan omana kysymyksenään.

3 Testausautomaatio

3.1 Ohjelmistotestaus

Ohjelmiston testaaminen voidaan jakaa neljään eri kategoriaan; Yksikkötestaukseen (eng. unit testing), integraatiotestaukseen (eng. integration testing), järjestelmätestaukseen (eng. system testing) ja hyväksyntätestaukseen (eng. acceptance testing). Yksikkötestauksessa keskitytään mahdollisimman pieniin kokonaisuuksiin. Siinä testataan sovelluksen toiminnallisuuksia niin komponentti kerrallaan. Integraatiotestauksessa testataan, liikkuuko data sovelluksen osista toisiin. Järjestelmätestauksessa testataan koko järjestelmää toiminnassa. Siinä keskitytään sekä toiminnallisiin, että ei-toiminnallisiin ominaisuuksiin, kuten latausaikoihin, suorituskykyyn, luotettavuuteen ja tietoturvaan. Hyväksyntätestauksessa selvitetään, täyttääkö järjestelmä ennalta sovitut määritykset ja toiminnallisuudet. (Hamilton 2021a.) Tässä opinnäytetyössä keskitytään hyväksyntätesteihin. Kuviossa 2 on esitetty testauksen eri tasot.



Kuvio 2. Levels of Testing in Software Testing (Hamilton 2021a).

3.2 Testausautomaatio

Ohjelmiston **testausautomaatiolla** tarkoitetaan ohjelmiston testaamisen automatisointia. Automatisoitujen testien hyöty perinteisen, manuaalisen, testaamisen yli saadaan varsinkin ajan säästämisellä sekä testaamisen toteuttamisen helppoudella. Testeistä saadaan myös tarkempia ja niiden toteuttaminen on huomattavasti kustannustehokkaampaa.

Ohjelmistotestaaminen voidaan jakaa karkeasti kahteen eri lähestymistapaan. Näitä ovat lasilaatikkotestaus (eng. white box testing) sekä black box testaus (eng. black box testing). Ensin mainitussa ideana on se, että testejä suoritettaessa tiedetään, miten testattava ohjelmisto toimii ja miten se on rakennettu. Jälkimmäisenä mainitussa ideana taas on se, että testattavasta ohjelmistosta ei ole sen käyttöliittymän käyttöä syvempää tietämystä. Näiden lisäksi on olemassa kahden edellä mainitun välimuoto, grey box testaus (eng. grey box testing). Grey box testauksessa ideana on, että testejä suoritettaessa käyttöliittymän lisäksi tietoa löytyy myös ohjelmiston toiminnasta ja rakenteesta. Tämä opinnäytetyö lähestyy testattavaa Muster-katsastusjärjestelmää lasilaatikkotestauksen näkökulmasta. (Test Automation: The Ultimate Guide n.d.)

Jotta automaattitestien teko on kannattavaa, testattavan järjestelmän on oltava sellaisessa tilassa, jossa sen käyttöliittymä ei koe jatkuvia muutoksia. Jos näin ei olisi, tämä loisi tilanteen, jossa automaattitestejä joutuu korjaamaan toistuvasti vastaamaan alati uudistuvaa käyttöliittymää. Testien tulee myös olla sellaisia, jotka palauttavat järjestelmän samaan tilaan, jossa se oli ennen testien aloittamista.

3.3 Testausautomaatiotyökalut

Nykypäivänä on tarjolla useita erilaisia työkaluja web-sovellusten testausautomaatioon, kuten Selenium, Playwright, Puppeteer, RobotFramework ja Cypress. Sopivan työkalun valinta riippuu siitä, millaisia ominaisuuksia testaukseen käytettävältä työkalulta tarvitsee sekä siitä, millainen testattava ohjelmisto on. Yksi suosituimmista työkaluista testauksen automatisointiin on Selenium (Why is Selenium Still a Leader in the Software Test Automation Market in 2020 and Beyond? 2020).

Seleniumin tarina alkoi vuonna 2004, kun tuolloin ThoughtWorks -nimisessä yrityksessä työskennellyt Jason Huggings kehitti ohjelman nimeltään JavascriptTestRunner. Tämän ohjelman tarkoituksena oli ThoughtWorksin sisäisen ohjelmiston testaaminen. Huggins esitteli ohjelmaa useille kollegoilleen ja vastaanotto oli todella positiivista. Pian alettiin jo keskustelemaan ohjelman julkaisemisesta avoimeen lähdekoodiin ja lopulta tämä myös toteutui. (Selenium History n.d.) Nykyään Seleniumin avulla on mahdollista testata web-ohjelmistoja tunnetuimmilla selaimilla (kuten Chromella, Safarilla ja Firefoxilla) ja se tukee monia koodikieliä (kuten Java, C# ja Python). (Why is Selenium Still a Leader in the Software Test Automation Market in 2020 and Beyond? 2020; Veikkola 2020, luku 4.)

Yksi uusimmista testausautomaatiotyökaluista on Microsoftin vuonna 2020 julkaisema **Playwright**. Vaikka Playwright ei käytä Seleniumiin liitettyjä työkaluja suoraan, voidaan sen kuitenkin ikään kuin katsoa polveutuvan Seleniumista. Playwright nimittäin sai alkunsa, kun osa Googlen ylläpitämän avoimen lähdekoodin automaatiotyökalun, Puppeteerin, kehittäjistä siirtyi työskentelemään Microsoftille vuonna 2019. Puppeteer puolestaan hyödynsi toiminnassaan Seleniumin WebDriverin osaa, Chrome DevTools -protokollaa. Playwrightia kehitetään hyvin aktiivisesti, sillä se on saanut pelkästään vuonna 2020 yli 2500 päivitystä. (Veikkola 2020, luku 4.)

3.4 Cypress

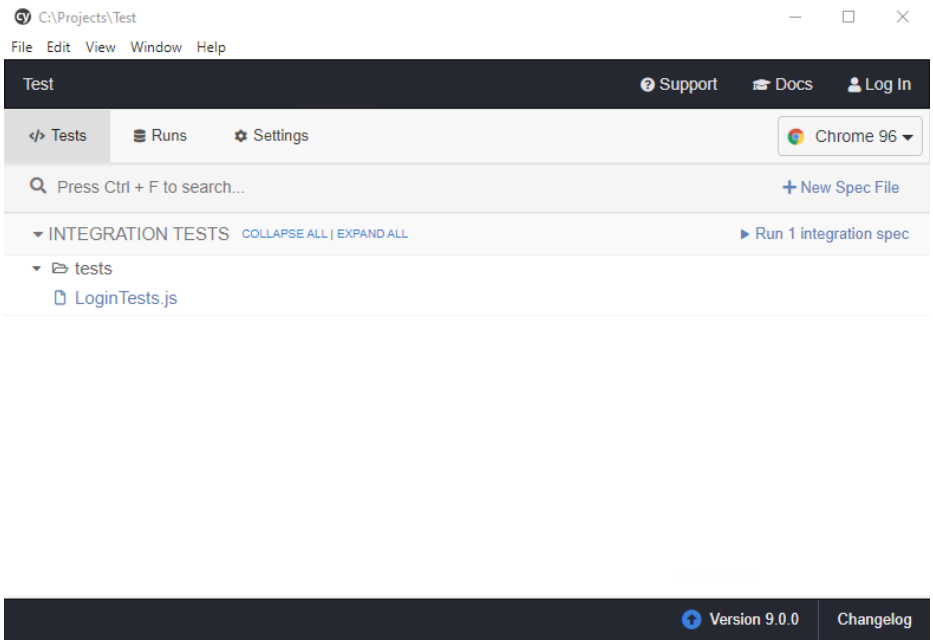
Cypress on web-ohjelmistojen käyttöliittymien testaukseen luotu työkalu, jonka tarkoituksena on helpottaa kehittäjien sekä testaajien elämää. Cypressin testit ovat käyttöliittymäpohjaisia end-to-end testejä. Se on kehitetty Node.js:n pohjalle ja tukee siis testien kirjoittamisessa JavaScriptiä. Suurin osa testien rakenteesta kirjoitetaan kuitenkin helposti ymmärrettävissä olevilla Cypressin omilla komennoilla. (Khetarpal 2021.)

Cypressia verrataan usein Seleniumiin, mutta molemmat eroavat rakenteeltaan merkittävästi (Taky 2021). Toisin kuin monet testausautomaatiotyökalut, Cypress toimii suoraan selaimessa, graafisen käyttöliittymän kautta. Oikean käyttäjän käyttökokemusta simuloidaan tarkemmin kuin monissa muissa tapauksissa, sillä Cypress lähettää komennot suoraan selaimen rajapinnalle. (Khetarpal 2021.)

Cypressin käyttöliittymä on helppokäyttöinen, ja sitä kautta käyttäjä pääsee selaamaan kaikkia testejä. Käyttöliittymän kautta voi myös suorittaa haluamiaan testejä. Käyttöliittymän lisäksi testejä voi suorittaa myös esimerkiksi PowerShell-ikkunan kautta Cypress-projektin juuressa seuraavalla komennolla:

```
./node_modules/.bin/cypress run
```

Käyttöliittymän kautta pääsee myös tarkastelemaan jo suoritettujen testien raportteja. Kuviossa 3 on Cypressin käyttöliittymä.



Kuvio 3. Cypressin käyttöliittymä.

4 Tutkimuksen toteutus

4.1 Tutkimuksessa käytetyt työkalut ja kehitysympäristö

Musterin kehittäjät käyttävät ohjelmiston kehitykseen virtuaali-kehitysympäristöä. Kehitysympäristöä pidetään yllä Microsoftin Hyper-V:llä. Virtuaalisen kehitysympäristön ehdoton hyöty ohjelmistontuotannossa on siinä, että jokaiselle projektille pystyy keskittämään oman kehitysympäristön, optimoiden koko ympäristön projektin tarpeiden mukaiseksi.

Tätä opinnäytetyötä varten pystytettiin erillinen virtuaali-kehitysympäristö, jotta Musterin versio pysyisi samana koko opinnäytetyöprojektin ajan. Virtuaalikoneelle asennettiin tarvittavat työkalut kehitykseen, versionhallintaan, sekä Musteria ja Cypressia varten. Itse kehitystyötä varten virtuaalikoneelle asennettiin Visual studio ja versionhallintaan käytettiin Gittiä. Cypressia varten virtuaalikoneelle asennettiin Node.js. Musteria varten edellä mainittujen lisäksi virtuaalikoneelle asennettiin Redis, Visual Studio Code sekä SQL Server.

4.2 Tehtävien testien määritys

Ennen tässä tutkimuksessa käytettyjen testien suunnittelua ja toteuttamista käytiin toimeksiantajan kanssa palaveri siitä, minkä tyyppisiä testejä haluttaisiin verrata toisiinsa. Päädyttiin ratkaisuun, jossa kaikki testit keskitetään sisäänkirjautumiseen sekä käyttäjien hallintaan. Alkuperäinen suunnitelma oli se, että tehtäisiin testejä mahdollisimman monelta osa-alueelta, mutta katsottiin järkevämmäksi tuottaa yhden sarjan testit kokonaisuudessaan, niin kuin ne Musterissa ovat. Näin pystytään testaamaan Musterin perusominaisuuksia mahdollisimman tehokkaasti sekä pystytään vertaamaan molemmilla työkaluilla tehdyt testit kattavasti.

Jotta eri testausautomaatioteknologioita voitaisiin testata mahdollisimman tarkasti, tuli Cypressilla luodut testit luoda samalla logiikalla, kuin millä Musterissa käytössä olevat Selenium-testit ovat luotu. Tässä työssä on siis tuotettujen testien pohjana käytetty Musterin hyväksyntätestejä, tarkemmin sanottuna Musterin LoginTests-testejä (suom. kirjautumistestit).

4.3 Kehitysympäristön pystyttäminen

Tutkimuksen toteutus alkoi tutustumalla Musterissa jo käytössä oleviin testeihin. Ensimmäiset testit päätettiin kirjoittaa ja suorittaa Musterin testiympäristöä vasten, joten erillistä kehitysympäristöä ei aluksi vaadittu. Ainoat asiat, mitä tietokoneelle siis tuli asentaa, oli Node.js sekä sen avulla Cypressin omat työkalut.

Työn edetessä Musterin saamat päivitykset kuitenkin loivat ongelman. Musterin käyttöliittymä sai säännöllisiä päivityksiä, jolloin jo kirjoitettuja testejä piti korjata. Ongelman välttämiseksi luotiin virtuaalikone ja virtuaalikoneeseen luotiin kehitysympäristö, jossa Musteria pystyi ajamaan paikallisesti ja pitämään versio samana koko työn ajan. Näin myös Musterin alkuperäiset testit pysyivät muuttumattomina, koska testit löytyvät samasta kloonatusta repositoriosta, kuin Musterin lähdekoodi. Koodinäytteessä 1 Musterin kehitysympäristön pystyttämistä varten luotu PowerShell-skripti.

Koodinäyte 1 on salassa pidettävä.

Koodinäyte 1. Musterin kehitysympäristöä varten asennettavat työkalut

Cypress ei vaadi toimiakseen Node.js-ajoympäristöä, mutta se helpottaa Cypressin asentamista sekä käyttöä huomattavasti. Cypress voidaan asentaa lataamalla asennuspaketti suoraan Cypressin sivuilta, mutta paljon helpompi ratkaisu on asentaminen npm-paketinhallintajärjestelmän avulla. Npm on Node.js-ajoympäristön oletuspaketinhallintajärjestelmä. Kun npm on asennettu, Cypress asennetaan haluttuun tiedostosijaintiin PowerShell-ikkunassa seuraavalla komennolla:

```
npm install cypress --save-dev
```

Edellä mainittu komento asentaa tarvittavat tiedostot kansioon, jonka sisässä PowerShell-ikkuna on komennon ajohetkellä. Tämä komento ei kuitenkaan vielä luo tarvittavaa Cypressin kansiorakennetta, jota käytetään muun muassa testitapausten sekä testikomentojen kirjoittamiseen. Tämä kansiorakenne luodaan Cypressin ensimmäisellä käynnistyskerralla. Cypress käynnistetään PowerShell-ikkunalla samassa sijainnissa, kuin mihinin Cypress asennettiin seuraavalla komennolla:

```
./node_modules/.bin/cypress open
```

Ensimmäisen käynnistyskerran yhteydessä Cypress luo esimerkkejä testikomennoista helpottaakseen käyttäjän alkuun pääsemistä. Kuviossa 4 kuvattu Cypressin automaattisesti luoma kansiorakenne.

```
/cypress
  /fixtures
    - example.json

  /integration
    /examples
      /1-getting-started
        - todo.spec.js
      /2-advanced-examples
        - actions.spec.js
        - aliasing.spec.js
        - assertions.spec.js
        - connectors.spec.js
        - cookies.spec.js
        - cypress_api.spec.js
        - files.spec.js
        - local_storage.spec.js
        - location.spec.js
        - misc.spec.js
        - navigation.spec.js
        - network_requests.spec.js
        - …

  /plugins
    - index.js

  /support
    - commands.js
    - index.js

```

Kuvio 4. Cypressin kansiorakenne oletuksena (mukaillen Writing and Organizing Tests n.d.)

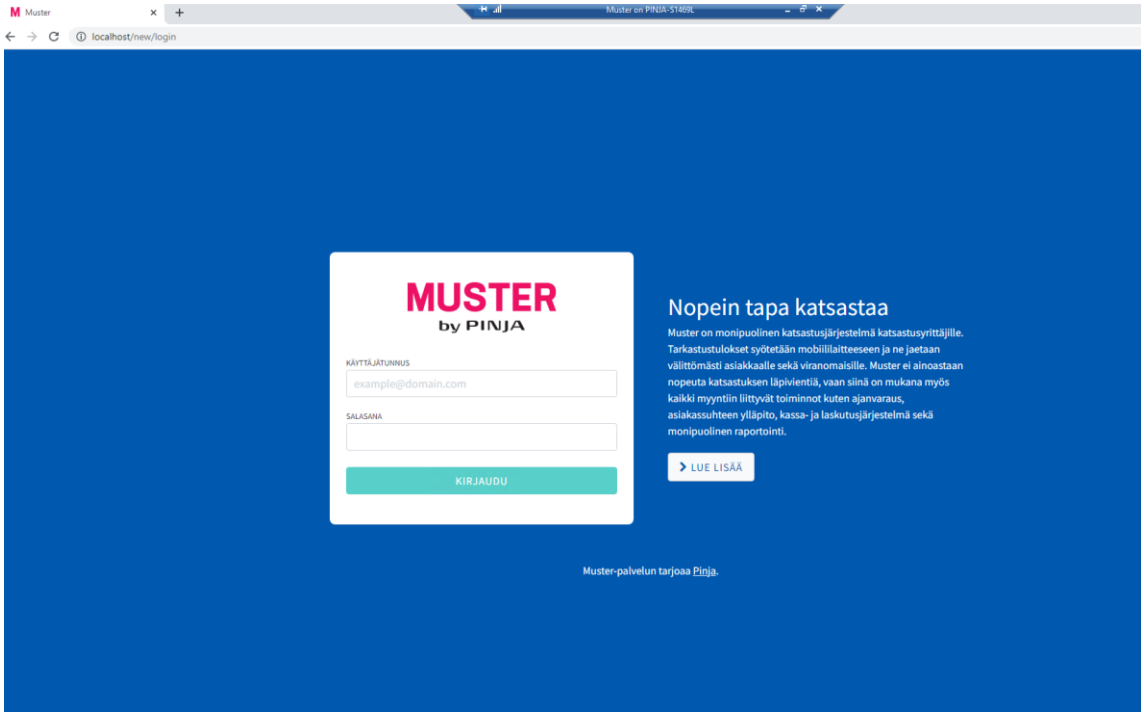
Kun Cypress on asennettu, seuraava toimenpide oli pystyttää Muster paikallisesti virtuaaliseen kehitysympäristöön. Tätä varten kehityskoneelle tuli kloonata kaksi Musterin repositoriota: front-end repositorio sekä back-end repositorio. Front-end repositorio sisältää kaiken tarpeellisen Musterin käyttöliittymää varten ja back-end repositorio nimensä mukaisesti sisältää kaiken tarpeellisen taustalla tapahtuvien prosessien suorittamiseen. Back-end repositorio sisältää myös testidatan tietokantaa varten.

Musterin back-end saatiin pyörimään paikallisesti kääntämällä Musterin back-end repositoriosta löytyvä Visual Studio solution. Front-end käynnistettiin avaamalla Musterin front-end repositorion juuri Visual Studio Codella ja ajamalla seuraava komento sen terminaalissa:

Musterin front-endin käynnistykomento on salassa pidettävä.

Tämän jälkeen tuli vielä pystyttää tietokanta. Tietokanta pyöri paikallisesti SQL Serverin avulla, joka ladattiin ja asennettiin Musterin työkalujen asennuksen yhteydessä. Tietokanta rakennettiin

Musterin back-endistä löytyneen datan avulla ja tämän jälkeen Musterin käyttöliittymä saatiin auki paikallisesti. Kuviossa 4 Musterin etusivu.



Kuvio 5. Musterin etusivu.

4.4 Testien kirjoitus

Testien kirjoittaminen aloitettiin ottamalla Musterin kirjautumistestit auki. Musterin hyväksyntätestit ovat toteutettu siten, että kirjautumiseen liittyvät testitapaukset ovat määritelty Tests-kansiossa LoginTests.cs-tiedostossa ja testitapauksissa käytetyt testikomennot ovat määritelty Pages-kansiossa eri tiedostoissa, riippuen mihin osa-alueeseen testit liittyvät. Kuviossa 6 on kuvattu Musterin testausautomaation kansiorakenne.

Kuvio 6 on salassa pidettävä.

Kuvio 6. Musterin testausautomaation kansiorakenne.

Cypressilla luotuja testejä lähdettiin toteuttamaan samalla periaatteella, kuin miten Musterin testit ovat toteutettu. Pages- ja Tests-kansioiden sijaan Cypress käyttää oletuksena support- ja integ-

ration-kansioita. Testien toteutuksen aikana tavoitteena oli myös testien optimointi, sillä tavoitteena oli selvittää, voisiko Cypress olla mahdollinen Selenium-testausautomaation korvaaja Musterille.

Itse testitapaukset sisältävä tiedosto, LoginTests.js sisältää lähdeviitteet tarvittaviin määrittelytiedostoihin, tarvittavat muuttujat testejä varten sekä testit. Testien osuus alkaa koko testipatteristoa määrittelevällä kuvauksella ja jatkuu testitapauksilla, joille on myös asetettu kuvaus. Koodinäytteessä 2 näyte LoginTest.js-tiedoston sisällöstä. Näyte sisältää kaksi ensimmäistä testitapausta.

Koodinäyte 2 on salassa pidettävä.

Koodinäyte 2. Näyte LoginTest.js-tiedoston sisällöstä.

Testien kirjoituksen aikana tavoitteena oli pitää sekä itse testitapaukset että testikomentojen määrittelytiedostot mahdollisimman hyvin jäsenneltynä sekä helposti luettavina. Pelkästään kirjautumiseen liittyvien testikomentojen määrittely jakautui siis useampaan lähdetiedostoon. Tämä tukee sekä helpottaa mahdollista tämän työn jälkeistä projektin jatkamista toimeksiantajan puolella. Koodinäytteessä 3 näyte testikomentojen määrittelytiedostosta, LoginResource.js:tä. Koko tiedosto on nähtävissä liitteissä (liite 1).

Koodinäyte 3 on salassa pidettävä.

Koodinäyte 3. Näyte testikomentojen määrittelytiedostosta.

Testikomentojen määrittäminen sekä määritettyjen komentojen käyttäminen on tehty käyttäjälle helpoksi. Seuraavilla syntakseilla määritetään komennot ja ne otetaan testitapauksen sisällä käyttöön:

```
Cypress.Commands.add("KomennonNimi", (muuttuja1, muuttuja2) => {Komennon määrittely})
cy.KomennonNimi(muuttuja1, muuttuja2)
```

5 Tulokset

Tämän työn tarkoituksena oli tutkia, voisiko Cypress korvata Musterin tällä hetkellä käyttämän testausautomaatiotyökalun, Seleniumin. Valitut vertailukohdat olivat testien ylläpidettävyys, luotettavuus sekä suorituskyky. Luotettavuuteen ja suorituskykyyn parhaaksi mittariksi katsottiin toimeksiantajan kanssa toistuva testien ajaminen. Tähän vertailuun päätettiin sopivaksi testien ajokertojen määräksi 200 ajokertaa. Koska testitapauksia on 18 kappaletta ajokertaa kohden, lopulliseksi testien määräksi tulee 3600 testiä vertailukohdetta kohden.

Sekä Seleniumista että Cypressista löytyy hyvät raportointityökalut. Ongelmana oli ainoastaan se, että paikallisesti testiympäristössä testattaessa testien tallennus tapahtui oletuksena ainoastaan yksi ajokerta kerrallaan. Ongelman välttämiseksi luotiin skripti, joka kävi hakemassa raporttitiedostosta tarvittavan datan ja talletti sen uuteen tiedostoon, johon kerättiin data jokaiselta ajokerralta. Musterin testien raportointi tapahtui xml-muodossa ja Cypressin json-muodossa. Talletettavaksi dataksi testiraporteilta päätettiin testin nimi tunnistusta varten, sekä testin ajoon kulunut aika ja tieto, onko testi mennyt läpi vai ei. Taulukossa 1 pääpointteja tutkimuksen tuloksista.

	Selenium	Cypress
Testien määrä	3600 kpl	3600 kpl
Testeihin kulunut kokonaisaika	12 h 21min 4 s	6 h 1 min 46 s
Keskimääräinen yhteen testiin kulunut aika	12.4 s	6.0 s
Onnistuneet testit	3599 kpl	3599 kpl
Epäonnistuneet testit	1 kpl	1 kpl
Testien Läpäisyprosentti	99,97 %	99,97 %

Taulukko 1. Pääpointteja tutkimuksen tuloksista.

Ylläpidettävyys

Tässä työssä tärkeimmäksi ylläpidettävyyden mittariksi päätettiin se, että minkälaisia eroja verrattavien testausautomaatioteknologioiden testien kirjoittamisessa, päivittämisessä ja korjaamisessa on. Musterin testitapaukset ovat kirjoitettu metodien sisään, kun taas Cypressissa testit on mahdollista kuvata huomattavasti selvemmin luettavasti. Koodinäytteissä 4 ja 5 testitapaus numero 16 Cypressilla ja Seleniumilla.

Koodinäyte 4 on salassa pidettävä.

Koodinäyte 4. Testitapaus numero 16 Cypressilla.

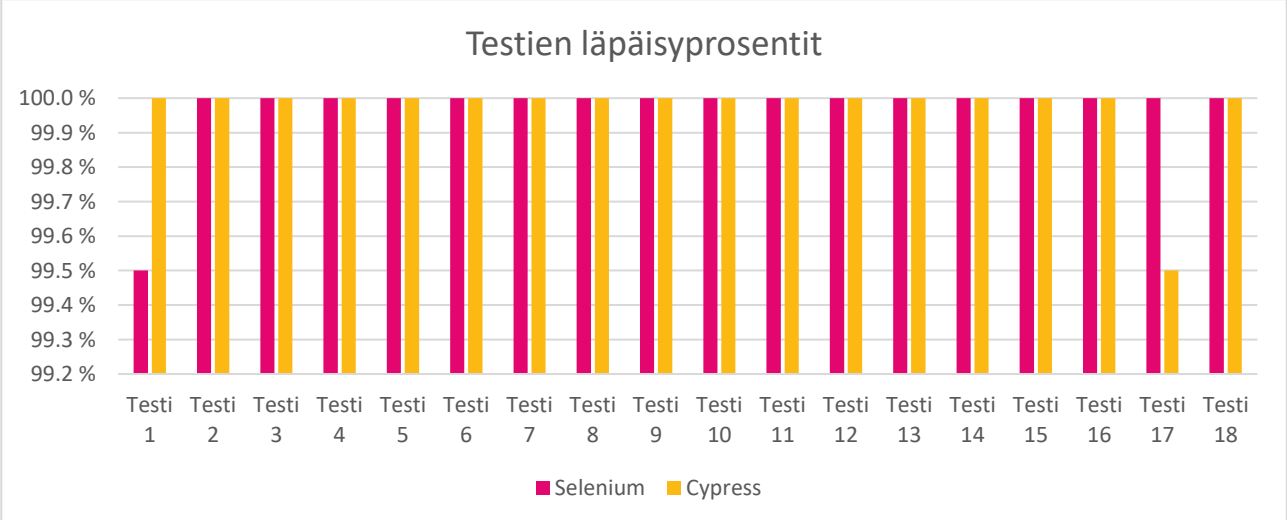
Koodinäyte 5 on salassa pidettävä.

Koodinäyte 5. Testitapaus numero 16 Seleniumilla.

Itse testien ja testikomentojen muodostaminen sekä Seleniumilla että Cypressilla on perusperiaatteeltaan melko toisiaan vastaavaa. Molemmissa pystyy määrittelemään haluamiaan testikomentoja haluamassaan sijainnissa ja käyttää niitä testitapauksissa haluamallaan tavalla.

Luotettavuus

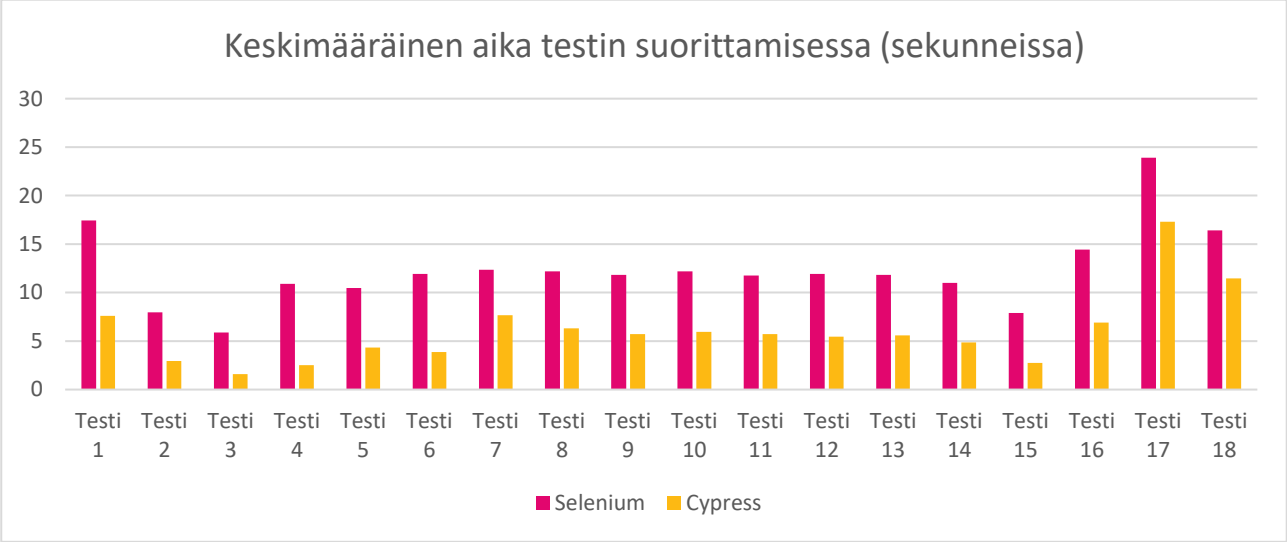
Tutkimuksen tulosten perusteella molemmat työkalut ovat yhtä luotettavia. 99.97 % testeistä meni läpi molemmilla työkaluilla, joka tarkoittaa sitä, että kaikista suoritetuista testeistä ainoastaan yksi testi epäonnistui. Testitulokset osoittavat, että molemmat testausautomaatiotyökalut ovat yhtä luotettavia. Taulukossa 2 testitapausten läpäisyprosentit molemmilla työkaluilla.



Taulukko 2. Testietapausten läpäisyprosentit molemmilla työkaluilla.

Suorituskyky

Suurimmaksi eroksi Seleniumin Ja Cypressin välillä osoittautui testien suoritusaika. Kuten testitulokset osoittavat, Cypressilla toteutetut testit olivat huomattavasti nopeampia, kuin Seleniumilla toteutetut testit. Cypress kävi testit läpi 48.82 %:ssa Seleniumin kokonaisajasta. Testien suoritusaikaan Seleniumilla vaikutti oletettavasti se, että jo käytössä oleviin testeihin on jouduttu tekemään paljon kompromisseja testien läpi menemisen varmistamiseksi. Testien askelten välille on jouduttu lisäämään merkittävästi viivettä, jotta Selenium on pystynyt toimimaan luotettavammin. Taulukossa 3 keskimääräinen testin suoritusaika testeittäin molemmilla työkaluilla.



Taulukko 3. Keskimääräinen testin suoritusaika testeittäin molemmilla työkaluilla.

Mitä web-käyttöliittymän testausautomaatioon tulee, suoritetun tutkimuksen tuloksien perusteella Cypressilla tuotetut testit ovat Seleniumilla tuotettuja testejä parempi valinta. Merkittävimpänä etuna Seleniumiin verrattuna, Cypressin nopeus testien suorittamisessa on yli kaksi kertaa parempi, vaikka testien onnistunut suoriutuminen on molemmissa samalla tasolla. Tämä tarkoittaa myös sitä, että Cypressia käyttämällä on mahdollista vaikuttaa edistävästi testien onnistumisprosenttiin ja pysyä silti Seleniumin suoritusaikojen puitteissa.

6 Pohdinta

Tämän opinnäytetyön tavoitteena oli tutkia, voisiko Cypress korvata Seleniumin Musterin testausautomaatiotyökaluna. Toimeksiantajan kanssa sovittuna toimeksiantona oli se, että tuotan itse Musterissa jo käytössä olevia kirjautumistestejä mukaillen kirjautumistestipatteriston Cypressilla, jonka jälkeen ajan määrätyn määrän testejä molemmilla työkaluilla. Toimeksiantajan kanssa sovitut tutkimuskysymykset käsittelivät molempien testausautomaatiotyökalujen ylläpidettävyyttä, suorituskykyä sekä luotettavuutta. Näiden tietojen pohjalta tätä opinnäytetyötä oli kivutonta lähteä toteuttamaan.

Tuloksista on pääteltävissä, että Cypress voisi olla Musterin käytössä nykyistä parempi työkalu käyttöliittymän testausautomaatioon. Huomioon otetuissa osa-alueissa (ylläpidettävyys, luotettavuus, suorituskyky) Cypress suoriutui kokonaistulosta tarkastellessa merkittävästi paremmin. Ylläpidettävyyden ja luotettavuuden suhteen tulokset vastasivat molemmilla työkaluilla toisiaan, mutta Cypressin nopeus oli yli kaksi kertaa parempi, kuin mitä Seleniumin.

Tässä opinnäytetyössä onnistunutta oli ainakin se, että minun mielestäni asetetut tavoitteet täytettiin. Toimeksiantona oli tutkia, voisiko Cypress toimia Musterin testausautomaatiotyökaluna ja tähän kysymykseen tämä opinnäytetyö vastasi. Parannettavaa olisi voinut olla ainakin testien kattavuuden suhteen, mutta jo tämän opinnäytetyön alussa toimeksiantajan kanssa sovittiin pysyvämme vain tietyssä osa-alueessa, kirjautumistesteissä. Täysin mahdollista olisi voinut myös olla se, että esimerkiksi testien määrän olisi pitänyt samana, mutta testejä olisi jakanut kirjautumistestien lisäksi esimerkiksi katsastustapahtumien ja myyntitapahtumien kesken.

Rajoittavana tekijänä tässä opinnäytetyössä voisi pitää sitä, että vaikka testien kirjoittamisen mallina käytettiin jo Musterissa käytössä olevia testejä, niin testien kirjoittaja on kuitenkin eri henkilö, kuin mitä Musterin alkuperäisten testien tekijä. Testien logiikassa voi olla pieniä eroja, jotka voivat vaikuttaa tuloksiin. Tämä heijastuu omalta osaltaan myös tulosten luotettavuuteen sekä käyttökelpoisuuteen. En kuitenkaan itse usko, että vaikutus on kovinkaan merkittävä. Testien ajokertojen määrää olisi myös tietenkin voinut olla suurempi, mutta tämän opinnäytetyön tuottamat tulokset kuitenkin antavat hyvää osviittaa molempien testausautomaatiotyökalujen eroista.

Tämän työn jatkokehityksen suhteen voisi ajatella mahdollisuutta, jossa tässä työssä käytettyyn vertailuun ottaisi mukaan useampia eri teknologioita. Nykyään on olemassa paljon erilaisia työkaluja testauksen automatisointiin ja niiden vertailusta saa suurta hyötyä itselle sopivaa työkalua valittaessa.

Lähteet

Casey, K & Chai, W. 2021. Software as a Service (SaaS). Viitattu 10.9.2021. https://searchcloudcomputing.techtarget.com/definition/Software-as-a-Service.

Hamilton, T. 2021a. Levels of Testing in Software Testing. Viitattu 10.10.2021. https://www.guru99.com/levels-of-testing.html.

Hamilton, T. 2021b. V-Model in Software Testing. Viitattu 30.10.2021. https://www.guru99.com/v-model-software-testing.html.

Kehittämistyön menetelmät tukena opinnäytetyössä. 2018. Viitattu 30.11.2021. https://essee-pankki.proakatemia.fi/kehittamistyon-menetelmat-tukena-opinnaytetyossa/.

Khetarpal, A. 2021. What is Cypress: Introduction and Architecture. Viitattu 31.10.2021. https://www.toolsqa.com/cypress/what-is-cypress/.

Muster by Pinja. N.d. Viitattu 19.9.2021. https://pinja.com/palvelut/kaupan-ja-palveluala/muster.

Pinja, 2021, Viitattu 20.9.2021. https://pinja.com/pinja.

SDLC – V Model. N.d. Viitattu 30.10.2021. https://notepub.io/notes/software-engineering/software-development-life-cycle/sdlc-v-model/.

Selenium History. N.d. Viitattu 16.11.2021. https://www.selenium.dev/history/.

Suvanto, M. 2014. Uusia malleja työelämän kehittämiseen. Satakunnan ammattikorkeakoulu. Viitattu 30.11.2021. https://urn.fi/URN:ISBN:978-951-633-152-5.

Taky, M, T. 2021: Automated Testing with Cypress. Opinnäytetyö, AMK. Technology and Communication. Viitattu 30.10.2021. https://urn.fi/URN:NBN:fi:amk-202105026582.

Test Automation: The Ultimate Guide. N.d. Viitattu 25.10.2021. https://www.leapwork.com/test-automation.

Veikkola, A. 2021. Selenium ja Playwright - testiautomaatiotyökalujen vertailu web-automaatiokäytössä. Opinnäytetyö, AMK. Tieto- ja viestintätekniikan tutkinto-ohjelma. Viitattu 17.11.2021. https://urn.fi/URN:NBN:fi:amk-2021102618946.

What Is Test Automation? A Simple, Clear Introduction. 2019. Viitattu 29.9.2021. https://www.testim.io/blog/what-is-test-automation/.

Why is Selenium Still a Leader in the Software Test Automation Market in 2020 and Beyond? 2020. Viitattu 16.11.2021. https://www.softwaretestingclass.com/why-is-selenium-still-a-leader-in-software-test-automation-market/.

Writing and Organizing Tests. N.d. Viitattu 17.11.2021. https://docs.cypress.io/guides/core-concepts/writing-and-organizing-tests.

Liitteet

Liite 1. LoginRecource.js-tiedoston sisältö

Liite 1 on salassa pidettävä.