Automatisoitujen käyttöliittymän regressiotestien optimointi

Niko Mäkelä

2020 Laurea

Laurea-ammattikorkeakoulu

Automatisoitujen käyttöliittymän regressio-testien optimointi

Niko Mäkelä
Tietojenkäsittely
Opinnäytetyö
Helmikuu, 2020

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittely
Tradenomi (AMK)

Niko Mäkelä

Automatisoitujen käyttöliittymän regressiotestien optimointi

Vuosi 2020 Sivumäärä 32

Tämä toiminnallinen opinnäytetyö on toteutettu LAURA Rekrytointi Oy:lle. Opinnäytetyön tarkoituksena on kehittää käytössä olevia automaatiotestejä, joita käytetään Rekrytointi.com -rekrytointisivuston käyttöliittymän regressiotestauksessa. Automaatiotestauksessa työkaluna käytetään käyttäytymislähtöisen kehityksen Behat-kehystä.

Automaatiotestien puutteita ja ongelmakohtia kartoitettiin haastattelemalla Rekrytointi.com -kehitystiimin jäseniä. Haastattelun perusteella opinnäytetyön tavoitteiksi valittiin testien suoritusajan puolittaminen, testitulosten taltioinnin kehittäminen ja väärien virheilmoitusten karsiminen. Automaatiotestien kehityksessä hyödynnettiin testien iteratiivista läpikäyntiä, lähteistä löytyneitä automaatiotestauksen hyviä käytäntöjä sekä Behatin omaa dokumentaatiota.

Opinnäytetyön tuotoksena syntyi paranneltu versio käytössä olleista automaatiotesteistä. Testien suoritusaikaa saatiin lyhennettyä 15 minuutista alle kahdeksaan ja puoleen minuuttiin. Automaatiotesteihin lisättiin raportointiominaisuus, jonka avulla testitulosten statistiikka saatiin helposti tarkasteltavaan muotoon. Ominaisuuden avulla testeistä saadaan talteen aikaisempaa laadukkaampaa tietoa. Lisäksi testeissä esiintyneitä vääriä virheilmoituksia saatiin vähennettyä.

Automaatiotestien kehittäminen jatkuu vielä tulevaisuudessa. Suurimpana kehittämiskohteena on saada testit toimimaan rinnakkain, jotta testaaminen olisi nopeampaa ja tehokkaampaa. Tällä hetkellä automaatiotestit kattavat vain suomenkielisen käyttöliittymän avaintoiminnot. Jatkossa on tarkoitus laajentaa testien käyttöä myös englanninkieliselle käyttöliittymälle.

Asiasanat: Automaatiotestaus, Behat, Ohjelmistotestaus

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bachelor's Degree

Niko Mäkelä

Optimization of Automated User Interface Regression Tests

Year 2020 Pages 32

This functional thesis is commissioned by LAURA Rekrytointi Oy. The purpose of this thesis is to further develop automated tests which are used in Rekrytointi.com recruitment site's user interface regression testing. Behat Framework for Behavior-Driven Development is used as a tool in automation testing.

The shortcomings and problems of the automated tests were identified by interviewing members of the Rekrytointi.com development team. Based on the interview, the goals of the thesis were to halve the duration of the tests, to improve the recording of the test results and to reduce false positive errors. The development of automated tests utilized iterative review of tests, good practices in test automation based in sources, and Behat's documentation.

The thesis produced an improved version of the automated tests. The duration of tests was reduced from 15 minutes to less than eight and half minutes. A reporting feature was added to the automated tests to provide better and easier viewing possibility of test results. The feature provides better quality test data than before. In addition, false positive errors in the tests were reduced.

The development of automated tests will continue in the future. The focus is to enable the tests run in parallel to make testing faster and more efficient. Currently, the automated tests cover only the key functions of the Finnish language interface. In the future, it is planned to extend the use of tests to the English language interface.

Keywords: Automated Testing, Behat, Software Testing

Sisällys

1 Johdanto

Tämä toiminnallinen opinnäytetyö on toteutettu LAURA Rekrytointi Oy:lle. Yritys on digitaalisen rekrytoinnin asiantuntija, joka on ollut alalla jo yli 21 vuotta. LAURA Rekrytointi Oy toimittaa kolmea palvelua, jotka ovat Rekrytointi.com -rekrytointisivusto, LAURA™-rekrytointijärjestelmä sekä Rekrytointi.com Osaajapankki. Palveluiden tavoitteena on auttaa yrityksiä onnistumaan paremmin rekrytoinneissa sekä auttaa osaajia työurallaan eteenpäin.

LAURA Rekrytointi Oy:llä on otettu keväällä 2019 käyttöön automaatiotestit, joita käytetään Rekrytointi.com -rekrytointisivuston käyttöliittymän regressiotestauksessa. Automaatiotestaustyökaluna käytetään käyttäytymislähtöisen kehityksen Behat-kehystä. Automaatiotestien tavoitteena on helpottaa kehitystöiden ja päivitysten tuotantoon vientiä. Automaatiotesteillä vähennetään koodimuutosten aiheuttamia virheitä sekä nopeutetaan ja tehostetaan testausta, ja siten parannetaan työn laatua.

Opinnäytetyön aiheeksi valikoitui näiden automaatiotestien kehittäminen. Olen itse ollut mukana AMK-harjoitteluni ja sen jälkeisen työsuhteeni aikana testien tekemisessä sekä niiden käyttöönotossa. Testien käyttöönoton jälkeen testeissä nähtiin vielä parantamisen varaa. Yhdessä kehitystiimin sekä esimieheni kanssa käytyjen keskustelujen perusteella automaatiotestien jatkokehittäminen nähtiin tarpeelliseksi, ja aihe valittiin kehittämistyön kohteeksi.

Opinnäytetyön tarkoituksena on selvittää, kuinka käytössä olevia automaatiotestejä voidaan kehittää ja optimoida. Automaatiotestien puutteita ja ongelmakohtia kartoitettiin haastattelemalla Rekrytointi.com -kehitystiimin jäseniä. Haastattelun pohjalta kehittämistyön konkreettisiksi tavoitteiksi valikoituivat testien suoritusajan puolittaminen, testitulosten taltioinnin kehittäminen sekä väärien virheilmoitusten (engl. False Positives) karsiminen.

Automaatiotestien toimintakykyä parannettiin valittujen kehityskohtien osalta. Kehittämistyössä hyödynnettiin testien iteratiivista läpikäyntiä, lähteistä löytyneitä automaatiotestauksen hyviä käytäntöjä sekä Behatin omaa dokumentaatiota. Myös kehittäjiltä saadut palautteet ja kommentit testeistä otettiin huomioon kehittämistyön aikana. Kehittämistyön tuotoksena syntyi automaatiotestien uusittu versio, joka on tällä hetkellä päivittäisessä käytössä LAURA Rekrytointi Oy:llä.

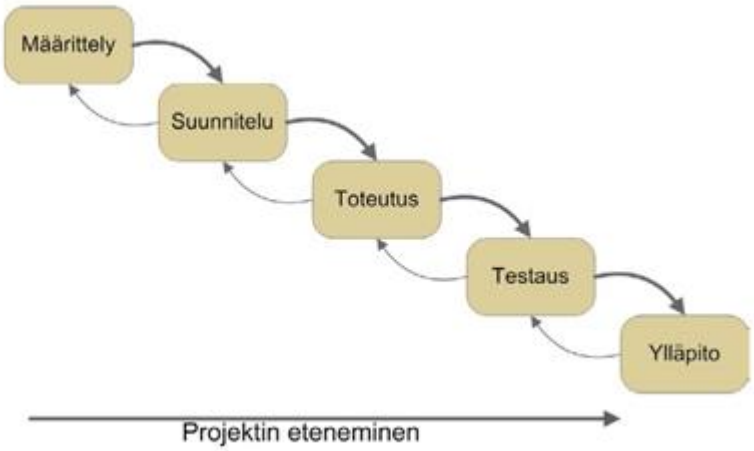
2 Ohjelmistotestaus

Tässä luvussa käsitellään ohjelmistotestausta yleisellä tasolla sekä perehdytään erilaisiin tapoihin, kuinka sitä voidaan toteuttaa. Aihetta havainnollistetaan ohjelmistotuotannon prosessimallien sekä testaustasojen avulla. Luvussa käsitellään myös ohjelmiston laadun varmistamisen kannalta tärkeää testausmenetelmää, regressiotestausta.

2.1 Ohjelmistotestaus yleisesti

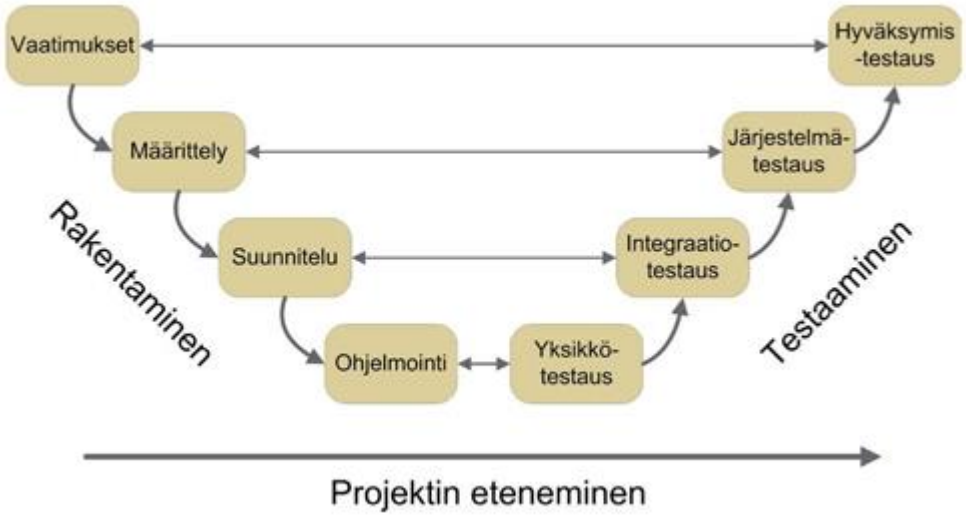
Ohjelmistotestauksella tarkoitetaan työtä, jolla varmistetaan, että kehitettävä ohjelmistotuote on sellainen kuin on toivottu, ja että sen ominaisuudet toimivat niin kuin on suunniteltu. Käytännössä ohjelmistotestauksella varmistetaan, että kehitetty tuote on toteutettu oikein. Testaustyön tarkoituksena on myös huomata sellaiset ohjelman kohdat, jotka poikkeavat suunnitellusta toteutuksesta. (Kasurinen 2013, 11.)

Kasurisen (2013, 14-15) mukaan testaus nähdään usein vain yhtenä työvaiheena ohjelmistotuotannossa. Tämä käsitys on peräisin perinteisestä ohjelmistotuotannon vesiputousmallista, jonka vaiheet ovat määrittely, suunnittelu, toteutus, testaus ja ylläpito (Kuvio 1). Vesiputousmallissa ideana on siirtyä aina vaiheesta seuraavaan, kunnes kehitettävä tuote on valmis. Malli on kuitenkin hieman vanhanaikainen ja sen ongelmiksi koetaan muun muassa liian myöhäinen testaaminen.



Kuvio 1: Vesiputousmalli (Kasurinen 2013, 14)

V-malli (Kuvio 2) on myös yksi ohjelmistotuotannon prosessimalleista. V-malli on Kasurisen (2013, 15) mukaan vesiputousmallia parempi vaihtoehto ohjelmistotuotantoon. V-mallissa tuotteen rakentamisosio on hyvin samankaltainen kuin vesiputousmallissa. Ero näkyy kuitenkin selvästi testauksen toiminnoissa. V-mallissa testaus ei ole oma yksittäinen työvaiheensa, vaan jokainen rakentamisen vaihe sisältää oman testausosionsa.



Kuvio 2: V-malli (Kasurinen 2013 ,15)

V-malli ei yleensä ole kuitenkaan sellaisenaan käytössä organisaatioissa, mutta siinä näkyvät testaamisen tasot esiintyvät usein alalla testaamisesta puhuttaessa. Yleensä testaaminen onkin jaettu mallin mukaisesti neljään tasoon. (Jyväskylän yliopisto n.d.) Nämä tasot ovat yksikkötestaus (engl. Unit Testing), integraatiotestaus (engl. Integration Testing), järjestelmätestaus (engl. System Testing) sekä hyväksymistestaus (engl. Acceptance Testing) (Burnstein 2003, 133).

2.2 Testauksen tasot

Edellä mainituista testauksen tasoista kolme kuuluu ohjelmiston kehitysvaiheen teknisen testauksen osioon: yksikkötestaus, integraatiotestaus ja järjestelmätestaus. Nämä testauksen tasot ovat eri laajuisia ja kaikilla niistä on oma tarkoituksensa. Yhteistä näillä kehitysvaiheen testaustasoilla on kuitenkin se, että testaaminen suoritetaan vielä kehitysympäristössä. Hyväksymistestauksessa taas testaaminen suoritetaan ohjelmiston todellisessa käyttöympäristössä ja testaamiseen osallistuu usein myös ohjelmiston loppukäyttäjät. (Kasurinen 2013, 41; Homes 2012, 62.)

Yksikkötestauksen tasolla testataan yksittäisen olion, moduulin tai funktion toimintaa itse toteutuksen yhteydessä. Yksikkötestauksen hoitaa yleensä kehittäjä itse. Tarkoituksena on varmistaa, että uuden komponentin toiminnalliset ominaisuudet toimivat oikein sekä varmistaa, että toteutuksessa ei ole koodivirheitä, jotka estäisivät ohjelman virheettömän kääntymisen. Yksikkötestaus on testausmuotona käytössä yleisesti kaikissa ohjelmistoalan organisaatioissa. (Kasurinen 2013, 41-42.) Hyvin toteutetulla yksikkötestauksella voidaan säästää mahdollisissa kuluissa, koska virheiden korjaaminen tulee kalliimmaksi, mitä pidemmälle testausprosessissa edetään (Software Testing Levels 2019).

Yksikkötestauksen jälkeen siirrytään integraatiotestaustasolle. Integraatiotestauksen tarkoituksena on testata kehitetyn komponentin toimivuutta osana jo valmiiksi testattua järjestelmää. Integraatiotestauksen päätarkoitus on varmistaa, että ohjelmiston eri osat toimivat yhdessä (Kasurinen 2013, 44.)

Järjestelmätestauksen tarkoituksena taas on testata kokonaista järjestelmää. Järjestelmätestausvaiheeseen siirrytään, kun kehitetyt komponentit on testattu toimiviksi ja integroitu osaksi järjestelmää integraatiotestausvaiheessa. Tavoitteena on varmistaa, että järjestelmä ei sisällä virheitä, ja että se on asetettujen vaatimusten mukainen sekä valmis toimitettavaksi käyttäjille. (Homes 2012, 61.) Kasurisen (2013, 46) mukaan järjestelmätestaukseen on erilaisia tapoja. Testausta voidaan toteuttaa esimerkiksi lasilaatikko- tai mustalaatikko-mallilla, käyttäjätesteillä tai millä tahansa testausmenetelmällä. Järjestelmätestausta tehdään edelleen testiympäristössä, ja siinä etsitään virheitä vielä myös yksittäisistä komponenteista.

Viimeisellä tasolla eli hyväksymistestauksen tasolla testataan, että järjestelmä on laadukas, ja että se on asetettujen vaatimusten mukainen. Käytännössä tässä vaiheessa tarkastetaan ja varmistetaan virallisesti, että järjestelmä toimii. Hyväksymistestausta ei normaalisti tehdä enää testausympäristössä, vaan testaus tapahtuu järjestelmän todellisessa käyttöympäristössä. (Kasurinen 2013, 46.) Normaalisti hyväksymistestauksen suorittaa järjestelmän loppukäyttäjät, mutta mukana voi olla myös testaajia (Homes 2012, 62).

2.3 Ketterät menetelmät

Ketterällä menetelmällä (engl. Agile Method) tarkoitetaan ohjelmistotuotannon menetelmää, jonka tarkoituksena on keventää perinteisen ohjelmistokehityksen malleja vähentämällä byrokratiaa ja hallintoa. Ketterien menetelmien tavoitteena on saada työstä helpommin hallittavaa ja mahdollistaa parempi reagoiminen muutoksiin. Ketterien menetelmien tarkoituksena ei ole varautua kaikkiin mahdollisiin ongelmatilanteisiin, vaan mahdollistaa nopea reagoiminen niihin. Ketterille menetelmille on tyypillistä ohjelmiston toimintojen, yksilöidenvälisen kommunikoinnin ja asiakasyhteistyön korostaminen. (Kasurinen 2013, 26.)

Yksi ketterän ohjelmistokehityksen menetelmistä on Scrum, joka on malliltaan melko helposti hallittava ja yksinkertainen. Se soveltuu parhaiten projekteihin, joissa työskentely tapahtuu tiiviissä yhteistyössä tiimin kesken, niin että kommunikointi ja tiedon välitys on helppoa ja vapaata. Scrumissa on tyypillistä aloittaa päivä lyhyellä Scrum-palaverilla, jossa käsitellään tulevan päivän tai viikon työtehtäviä. (Kasurinen 2013, 26.)

Scrum-mallissa ohjelmiston toteutus jaetaan iteraatioihin eli sprintteihin, jotka sisältävät jonkin osakokonaisuuden kehitettävästä ohjelmistosta. Jokaisen sprintin alussa kehitystiimi valitsee yhdessä tuote-backlogista joukon korkeimman prioriteetin kehitystehtäviä sprinttiin. Tuote-backlog on lista, joka sisältää tuotteen kaikki kehitystehtävät. (ISTQB 2014, 12.) Yksi

sprintti kestää yleensä muutaman viikon. Kun jokainen sprintti on suoritettu ja ohjelmisto vastaa vaatimusmäärittelyjä, on ohjelmisto valmis. (Kasurinen 2013, 26.) Kasurinen (2013, 26) kuitenkin huomauttaa, että Scrum on pelkästään malli, joka tarjoaa yritykselle ehdotuksen, kuinka toimia. Jokainen organisaatio kuitenkin luo itselleen oman, heidän tarpeeseensa sopivan version mallista.

Toinen ketterän kehityksen malli on Kanban, joka osiltaan muistuttaa Scrum-mallia. Kanban-malli keskittyy työnkulun optimointiin ja visualisoimiseen. Kanban-mallissa hyödynnetään Kanban-taulua työtehtävien hallinnassa ja työnkulun kuvaamisessa. Kanban-taulu koostuu sarakkeista, jotka kuvaavat työtehtävien tilaa. Sarakkeita voisivat olla esimerkiksi työn alla, testauksessa ja valmis. Sitä mukaan, kun tehtävät etenevät, ne liikkuvat aina sarakkeesta seuraavaan. Toisin kuin Scrum-mallissa, Kanbanissa sprintit eivät ole pakollisia. Kanban-malli myös mahdollistaa toiminnallisuuksien julkaisemisen yksitellen, kokonaisen versiojulkaisun sijaan. (ISTQB 2014, 12-13.) LAURA Rekrytointi Oy soveltaa yhdessä Scrum-mallia sekä Kanban-mallia Rekrytointi.com sivuston kehittämisessä.

Ketterässä tiimissä jokainen jäsen vastaa osaltaan testaustyöstä ja tuotteen laadusta. Ketterät menetelmät tuovat mukanaan kuitenkin riskejä testaukseen liittyen. Tällaisia riskejä ovat esimerkiksi testaajien ajattelutavan kärsiminen johtuen läheisestä työskentelystä kehittäjien kanssa. Testaajat saattavat myös helpommin katsoa läpi sormien tiimin tehotonta toimintaa. Testaajilla voi myös olla vaikeuksia pysyä mukana jatkuvien muutosten kanssa. (ISTQB 2014, 26.)

ISTQB:n (2014, 20-23) mukaan ketterissä menetelmissä oletetaan, että muutoksia voi tapahtua missä tahansa vaiheessa projektia. Muutokset olemassa oleviin toimintoihin lisää testaamisen tarvetta ja erityisesti regressiotestauksen tarve korostuu. On tärkeää varmistaa, että kehitystiimi pystyy hallitsemaan muutosten mukana tulevat riskit. Yksi tapa testauksen helpottamiseen on automaatiotestaus. Ongelmien välttämiseksi automatisoitujen regressiotestien pitäminen ajan tasalla on tärkeää.

2.4 Regressiotestaus

Regressiotestauksella tarkoitetaan järjestelmän jo aiemmin testattujen osien uudelleen testaamista (Homes 2012, 18). Regressiotestaus on yleistermi, jota voidaan käyttää, kun toimivaan järjestelmään tehdyn muutoksen jälkeen halutaan varmentaa, että järjestelmä toimii edelleen niin kuin pitää. Regressiotestauksessa tarkoituksena on varmistaa, että aiemmin korjatut ongelmat eivät uusiudu järjestelmään tehtyjen uusien muutosten jälkeen, eikä uusia ongelmia ilmene. (Kasurinen 2013, 55.) Muutosten aiheuttamia odottamattomia sivuvaikutuksia kutsutaan regressioiksi (ISTQB 2018, 41).

Kasurisen (2013, 55) mukaan järjestelmän virheet liittyvät usein uusiin komponentteihin tai niitä käyttäviin, aiemmin toteutettuihin järjestelmän toimintoihin. Kasurisen toteamaa tukee myös ISTQB (2018, 41), jonka mukaan koodimuutoksella voi olla odottamattomia vaikutuksia järjestelmän muihin toimintoihin. Tämän takia varsinkin ketterissä ohjelmistokehityksen malleissa regressiotestaus on erityisen tärkeää. Ketterien menetelmien iteratiivisen ja asteittaisen kehityksen elinkaaren takia järjestelmään tulee toistuvia muutoksia. Myös järjestelmän kasvaessa regressiotestauksen merkitys korostuu (ISTQB 2018, 29).

Kasurisen (2013, 55) ja ISTQB:n (2018, 41) mukaan regressiotestaus soveltuu hyvin testausautomaation käyttökohteeksi. Regressiotestausta voidaan suorittaa missä tahansa testaustasolla, ja sen automatisointi kannattaa aloittaa hyvissä ajoin (ISTQB 2018, 41). Varsinkin ketterissä menetelmissä regressioiden välttämiseksi ja tuotteen laadun varmistamiseksi, on tärkeää hyödyntää automaatiotestausta kaikilla testauksen tasoilla. (ISTQB:n 2014, 24)

LAURA Rekrytointi Oy:llä hyödynnetään käyttöliittymän automatisoituja regressiotestejä osana testausprosessia. Automaatiotestit suoritetaan kehittäjän toimesta, kun ohjelmiston koodipohjaan on tehty muutoksia. Automaatiotestit suoritetaan varhaisessa vaiheessa, jo ennen yksittäisen toiminnon siirtymistä testaajalle. Näin mahdolliset virheet saadaan huomattua hyvissä ajoin ja onnistutaan säästämään testaustyötä tekevän henkilön, kuin myös kehittäjien työaikaa.

3 Automaatiotestaus

Tässä luvussa käsitellään automaatiotestausta. Aluksi avataan automaatiotestaus -käsitettä, jonka jälkeen kerrotaan siihen liittyvistä hyvistä ja huonoista puolista. Lopuksi kerrotaan vielä automaatiotestaukseen liittyvistä yleisistä hyvistä käytännöistä.

3.1 Automaatiotestaus yleisesti

Automaatiotestaus on yksi testaustoiminnan muodoista. Tässä testausmuodossa rakennetaan automaattityökaluja ohjelmiston testaamista varten. (Kasurinen 2013, 61.) Käytännössä tarkoituksena on automatisoida nykyisiä manuaalisen testauksen vaiheita (Zambelich n.d., 4). Automaatiotestauksella automatisoidaan usein toistuvia tai manuaalisesti vaikeasti testattavia tehtäviä (Software Testing Help 2019). Kasurisen (2013, 61) mukaan automaatiotestauksen ydinajatus on vapauttaa testaajia muihin työtehtäviin. Kasurinen kuitenkin painottaa, että automaatiotestauksen ei ole tarkoitus korvata manuaalista testaamista, vaan täydentää sitä.

Tehokkaan automaatiotestauksen toteutumiseen vaaditaan vähintään tarkat testitapaukset sekä erillinen testausympäristö. Testitapauksien tulee sisältää odotetut tulokset, jotka on ennalta määritelty. Testausympäristössä tulee olla myös oma tietokantansa, minkä voi palauttaa ennalleen, jotta testejä voidaan ajaa aina kun ohjelmistoon tulee uusia ominaisuuksia. (Zambelich n.d., 4.)

Yksi yleinen käyttötarkoitus automaatiotesteille on käyttöliittymän testaus. Tällaisessa tapauksessa automaatiotestausohjelmalle annetaan toimenpidesarjoja, jotka ohjelma ajaa läpi, ja tarkastaa, toteutuuko ennalta määrätyt toimenpiteet. Tällaisia toimenpidesarjoja voi olla esimerkiksi hakukoneen käyttö, tietyn näkymän aukeaminen tai tiettyjen tietojen tulostuminen. (Kasurinen 2013, 62.)

3.2 Automaatiotestauksen hyödyt

Testauksen automatisoinnista voi saada monenlaisia hyötyjä oikein toteutettuna. Yksi merkittävä hyöty on manuaalisen testaamisen vähentyminen (ISTQB 2018, 81). Automaatiotestit mahdollistavat toistuvien testien suorittamisen vaivattomasti, ja aina samalla tavalla. Tällä pystytään varmistamaan muun muassa se, että testauksesta ei unohdu jotain olennaista vaihetta ja testit on aina toteutettu halutulla tavalla. (Rouse 2018.) Samalla tämä mahdollistaa testaajien vapauttamisen muihin työtehtäviin (Kasurinen 2013, 61).

Automaatiotyökalu pystyy suorittamaan testaamisen myös huomattavasti ihmistä nopeammin ja luotettavammin. Toisin kuin automaatiotyökalu, ihminen on altis virheille. Manuaalista testaamista suorittava henkilö voi erehtyä ja tehdä virheitä ohjelmaa tarkastaessa. Yksi automaatiotestauksen hyödyistä onkin tällaisten virheiden välttäminen. (Rouse 2018.)

Automaatiotestaustyökalut mahdollistavat myös testausinformaation helpomman ja paremman tarkastelun. Työkalujen ansiosta voidaan seurata muun muassa testien etenemistä, ohjelman suorituskykyä, erilaisia tilastoja sekä virheiden määrää. Tämä mahdollistaa testien objektiivisemman arvioinnin. (ISTQB 2018, 81.)

3.3 Automaatiotestauksen ongelmat

Testauksen automatisointiin liittyy myös riskejä ja mahdollisia haittoja. Yhtenä riskitekijänä on, että automaatiotyökalulta odotetaan yksinkertaisesti liikoja. Automaatiotestien rakentaminen ja ylläpito voi olla hyvin kallista ja aikaa vievää, joka saattaa usein yrityksissä unohtua. (ISTQB 2018, 81.) Yhden automaatiotestin rakentaminen vie usein kolmesta kymmeneen kertaa enemmän aikaa manuaalisen testin tekemiseen verrattuna (Hoffman 2003, 48). Tämä tarkoittaa sitä, että kehittäjiltä vaaditaan huomattava määrä ylimääräistä aikaa automaatiotestien kehittämiselle ja ylläpidolle, mikä hidastaa muiden tehtävien etenemistä.

Väärät virheilmoitukset (engl. False Positives) ovat myös yksi ongelma automaatiotestauksessa. Väärillä virheilmoituksilla tarkoitetaan työkalun palauttamia virheilmoituksia tilanteista, joissa testattavassa toiminnossa ei todellisuudessa ole virheitä (ISTQB 2018, 16). Tällaiset virheilmoitukset ovat yleisiä varsinkin käyttöliittymätestauksessa, jossa pienetkin muutokset käyttöliittymän näkymässä voivat aiheuttaa virheilmoituksen. Väärät virheilmoitukset ovat kalliita, koska niiden selvittäminen ja korjaaminen vievät ylimääräistä aikaa testauksen suorittajalta. (Hoffman 2003, 20.)

Vuosina 2009-2010 suomalaisille ohjelmistoalan yrityksille tehdyn tutkimuksen mukaan, suurin ongelma automaatiotestauksessa on testitapausten puutteellinen dokumentaatio. Puutteellisesta dokumentaatiosta johtuen, ei monissa tilanteissa tiedetä, kuinka testitapauksen tulisi toimia. Tämän takia testitapaus pitää läpikäydä manuaalisesti lähdekoodin avulla. (Kasurinen 2013, 124.)

Toinen tutkimuksessa noussut automaatiotestauksen ongelma oli ihmistarkastajan tarve. Testausautomaatio on yleensä itse rakennettu, joten testien toimivuudelle tarvitaan tarkastaja. Tämä oli yksi syy, miksi yritykset kokivat automaatiotestauksen ylläpidon hankalaksi ja kalliiksi. Useissa tapauksissa automaatiotestauksen käyttöönotto jäi toteuttamatta puuttuvien resurssien takia. (Kasurinen 2013, 124.)

3.4 Automaatiotestauksen onnistumiseen vaadittavat tekijät

Ensimmäinen vaihe automaatiotestauksen käyttöönotossa on testaustyökalun valinta. Automaatiotestaukseen tarkoitettuja työkaluja löytyy useita, ja niillä on erilaisia käyttötarkoituksia, joten valinnassa tulee ottaa huomioon testaamisen vaatimukset yrityksessä (Zambelich n.d., 5). Yhden työkalun käytön osaamisvaatimukset voivat olla myös hyvin erilaisia kuin toisen. Tämän takia on tärkeää varmistaa, että yrityksestä löytyy tarvittavaa osaamista työkalun käytölle. Myös budjetti on hyvä ottaa huomioon. (Software Testing Help 2019.)

Automaatiotestaustyökalun valinnan jälkeen tulisi miettiä kuka tekee testit. Tärkeimmät taidot, jotka testien tekijän tulisi hallita, ovat testaus sekä ohjelmointi. Testien tekijälle tulisi olla testauksen prosessit ja niihin liittyvät vaatimukset tuttuja, jotta testeistä saadaan halutunlaisia. Testien tekijällä pitäisi olla ymmärrystä myös ohjelmoinnista, jotta testeistä on mahdollista tehdä toimivia ja helposti ylläpidettäviä. (Pettichord 2001.)

Automaatiotestejä tehdessä tulisi tarkkaan miettiä, mitä testejä automatisoidaan. Testien automatisointi kannattaa aloittaa testeistä, jotka vievät paljon aikaa manuaalisesti testattaessa. Automaatiotestien laatijalla tulee olla myös hyvä ymmärrys testien normaaleista menettelytavoista, jotta testeistä saadaan mahdollisimman optimaalisia. Näin voidaan välttyä tilanteilta, joissa automaatiotestit joudutaan tekemään uudestaan, koska on keksitty parempi testaustapa kyseiselle ohjelmiston osalle. (Pettichord 2001.)

Automaatiotestien ylläpito on usein aikaa vievää. Tämän takia on tärkeää automatisoida vain sellaisia testejä, jotka ovat olennaisia ohjelmiston testauksen kannalta (Pettichord 2001). Yritykselle ei ole kannattavaa, että kehittäjät käyttävät aikaansa sellaisten testien kehittämiseen tai ylläpitämiseen, jotka testaavat epäolennaisia ohjelmiston toimintoja.

Ylläpidettävyyden kannalta on myös tärkeää, että testit kestävät ohjelmiston käyttöliittymän muutokset. Automaatiotestaustyökalut käyttävät erilaisia tapoja eri elementtien tunnistamiseen, esimerkiksi hyödyntäen elementin sijaintia. Tämä tarkoittaa sitä, että jos elementin

paikkaa muutetaan, antaa työkalu siitä virheilmoituksen, ja testi ei mene läpi. Tämä voidaan välttää antamalla elementeille uniikit nimet, joita käytetään testeissä elementtien tunnistamiseen. Näin testeistä saadaan vakaampia ja minimoidaan niiden kaatuminen. (Automated Testing Best Practices and Tips 2019.)

Kasurisen (2013, 62) mukaan yksikkötestausvaiheen testit ovat loistava kohde testauksen automatisoinnille. Testaaminen on hyvä aloittaa ajoissa, jotta virheet voidaan löytää mahdollisimman aikaisin. Virheiden korjaaminen kehitystyön alkuvaiheissa tulee merkittävästi halvemmaksi, kuin aivan loppumetreillä. (Automated Testing Best Practices and Tips 2019.)

4 Behat -automaatiotestaustyökalu

Tässä luvussa käsitellään Behat -automaatiotestaustyökalua, jonka avulla LAURA Rekrytointi Oy:llä suoritetaan automatisoituja käyttöliittymän regressiotestejä. Luvussa kerrotaan, kuinka työkalu toimii ja miten sillä voidaan suorittaa automaatiotestausta.

4.1 Behatin määrittelyä

Behat on PHP-ohjelmointikielelle rakennettu avoimenlähdekoodin käyttäytymislähtöisen kehityksen kehys (engl. Behavior-Driven Development Framework). Käyttäytymislähtöisessä kehityksessä korostetaan sidosryhmien välistä kommunikointia ohjelmiston kehittämisen aikana, mitä Behat tukee yksinkertaisten ja helposti ymmärrettävien esimerkkien avulla. (Behat Documentation n.d.)

Behat on tarkoitettu ohjelmistojen toimittamisen tukemiseen. Se on testausautomaation työkalu, joka tukee ohjelmistokehitystä jatkuvan viestinnän sekä testausautomaation avulla. Behatin tavoitteena on helpottaa ja rikastuttaa ohjelmiston vaatimuskeskusteluja, jonka avulla saadaan rakennettua oikeanlainen ohjelmisto. (Behat n.d.)

4.2 Behatin toiminta

Behat testaa ohjelmiston käyttäytymistä testitapauksien avulla, jotka ovat kirjoitettu Gherkin kielellä. Gherkin kielen syntaksi on toteutettu selkokielisellä tekstillä, joka mahdollistaa kielen helpon oppimisen ja bisnessääntöjen selkeän kuvaamisen. Gherkin käyttää sisennyksiä rakenteen jäsentelyyn, ja normaalisti jokainen rivi alkaa kielen omalla erityisellä avainsanalla. (Behat Documentation n.d.)

Behatissa testit ovat jaettu erillisiin feature -tiedostoihin. Jokainen tällainen tiedosto sisältää yhden testattavan toiminnallisuuden (Feature). Toiminnallisuuden testaaminen aloitetaan tiedoston alussa olevalla avainsanalla Feature. Tämän avainsanan jälkeen kirjoitetaan lyhyt kuvaus testattavasta toiminnallisuudesta.

Tiedostoon määritetty toiminnallisuus sisältää aina yhden tai useamman skenaarion, joissa testataan toiminnallisuuden jotain tiettyä tapahtumaa (Kuvio 3). Skenaariot aloitetaan aina avainsanalla Scenario, jonka perään voidaan kirjoittaa skenaariolle otsikko tai lyhyt kuvaus. Skenaariot koostuvat vaiheista (Steps), jotka alkavat yleensä avainsanalla Given, When, Then, But tai And. Behat käsittelee jokaista avainsanaa samalla tavalla, mutta on suositeltavaa, että käyttäjä ei näin tekisi. Eri avainsanojen käytöllä pystytään mahdollistamaan testien helppo ymmärrettävyys ja luettavuus. (Behat Documentation n.d.)

```
Feature: Login Feature
  Login to the application

  @smoke
  Scenario: Test that user can login.
    # Open login form
    Given I visit "/"
    When I click the "#mega-menu-item-33026 a" element
    Then I should see "Kirjaudu sisään"
    And I should not see "Työkalut"
    And I should not see "Kirjaudu ulos"

    # Add incorrect credentials
    Then I fill in "username" with "incorrect.username@example.com"
    And I fill in "password" with "incorrect-password"
    When I press "Kirjaudu"
    Then I should see "VIRHE: Virheellinen käyttäjätunnus."

    # Add correct credintials
    Then I click the "#mega-menu-item-33026 a" element
    And I fill in "username" with "correct.username@example.com"
    And I fill in "password" with "correct-password"
    When I press "Kirjaudu"

    # Now we should be logged in.
    Then I should see "Työkalut"
    And I should see "Kirjaudu ulos"
```

Kuvio 3: Esimerkkitesti kirjautumistoiminnolle

Vaikka Behat käsittelee avainsanoja samalla tavalla, on niille kuitenkin määritetty käyttötarkoitukset (Taulukko 1).

Avainsana	**Käyttötarkoitus**
Given	Järjestelmän asettaminen haluttuun tilaan, ennen käyttäjän toimintaa.
When	Kuvaa käyttäjän tekemän avaintoiminnan.
Then	Tarkastelee käyttäjän toiminnasta seuranneita, järjestelmän tulostamia tuloksia. Tarkoituksena on tarkastella vain sellaisia asioita, jotka näkyisivät myös todelliselle käyttäjälle.
And ja But	Käytetään tilanteessa, jossa muuten toistuisi useampi When, Given tai Then -vaihe. Tämän tarkoituksena on helpottaa skenaarion luettavuutta.
Background	Feature -tiedostoon määritettävä osio, joka suoritetaan jokaisen kyseisen tiedoston skenaarion alussa.

Taulukko 1: Gherkin-kielen avainsanat (Behat Documentation n.d.)

Toiminnallisuuksien testaamista varten tarvitaan funktioita, joiden avulla skenaarion vaiheiden toimintalogiikka määritetään ja toiminta toteutetaan. Jokaisen yksittäisen vaiheen taustalla toimii siis funktio, joka suorittaa kyseisen vaiheen toiminnot. Funktiot toteutetaan niin kutsutuissa kontekstiluokissa, joista käytetään nimitystä POPO eli Plain Old PHP Object. (Behat Documentation n.d.)

```php
/**
 * Authenticates a user.
 *
 * @Given /^I am logged in as "([^"]*)" with the password "([^"]*)"$/
 */
public function iAmLoggedInAsWithThePassword( $username, $password ) {
    $this->visitPath( '/' );
    // Open login form.
    $page = $this->getSession()->getPage();
    $link = $page->find( 'css', '#mega-menu-item-33026 > a' );
    $link->click();

    // Fill in the username.
    $usernameField = $page->findField( 'username' );
    $usernameField->setValue( $username );

    // Fill in the password
    $passwordField = $page->findField( 'password' );
    $passwordField->setValue( $password );

    // Log in
    $page->findButton( 'login' )->click();
    $link = $this->getSession()->getPage()->findLink( 'Työkalut' );
    if ( empty( $link ) ) {
        throw new Exception(
            'Login failed at ' . $this->getSession()
            ->getCurrentUrl()
        );
    }
}
```

Kuvio 4: Esimerkki funktiosta, joka suorittaa kirjautumisvaiheen: Given I am logged in as ”example-username” with the password ”example-password”.

Behatia käytetään komentorivin avulla. Komentorivi on tekstikäyttöliittymä tietokoneelle. Se on ohjelma, jolle voidaan antaa komentoja, jotka ohjelma lähettää eteenpäin tietokoneen käyttöjärjestelmälle. (Codecademy 2020.) Automaatiotestien suorittamista varten siirrytään komentorivin avulla projektikansioon, joka sisältää Behat-tiedostot. Tämän jälkeen testit käynnistetään esimerkiksi komennolla: ”php bin/behat”, joka suorittaa kaikki Feature-tiedostoihin määritetyt testit. Testien suoritukseen on mahdollista asettaa erilaisia filttereitä, joiden avulla pystytään määrittämään, mitä testejä suoritetaan. (Behat Documentation n.d.)

4.3 Web-sovellusten testaaminen Behatilla

Behat tarvitsee verkkoselaimessa käytettävän sovelluksen testaamiseen työkalun, jonka avulla pystytään olemaan vuorovaikutuksessa testattavan sovelluksen kanssa. Tämän toiminnan mahdollistavat selainemulaattorit, joiden tarkoituksena on simuloida verkkoselaimen toimintaa. Verkkoselain taas on ikkuna, jonka avulla käyttäjä käyttää web-sovellusta. (Behat Documentation n.d.)

Selainemulaattoreita on monia, mutta ne toimivat eritavoin ja niillä on erilaiset ohjelmointirajapinnat. Selainemulaattorit voidaan kuitenkin jakaa kahteen ryhmään. Emulaattoreihin,

jotka toimivat ilman käyttöliittymää (engl. Headless Browser Emulators) sekä selainohjaimiin (engl. Browser Controllers), jotka ohjaavat oikeaa selainta. (Mink documentation 2015.)

Headless Browser -emulaattorilla tarkoitetaan selainemulaattoria, jota voidaan ajaa serverillä konsolin avulla, ilman graafista käyttöliittymää. Nämä selainemulaattorit lähettävät oikeita HTTP-kutsuja serverille, jonka jälkeen ne jäsentävät serverin palauttaman vastauksen. Headless Browser -emulaattoreiden hyviä puolia ovat niiden nopeus, yksinkertaisuus sekä kyky toimia ilman oikeaa selainta. Nämä selainemulaattorit eivät kuitenkaan tue JavaScriptiä tai Ajaxia, mikä vaikeuttaa käyttöliittymän kattavaa testausta. (Mink documentation 2015.)

Selainohjaimien tarkoituksena taas on ohjata oikeaa selainta ja simuloida käyttäjän toimia selaimessa. Selainohjainten suurin hyöty on JavaScriptin ja Ajaxin käytön mahdollistaminen. Selainohjaimet ovat kuitenkin Headless Browser -emulaattoreihin verrattuna hitaampia. Ne vaativat myös toimiakseen oikean selaimen sekä usein lisäkonfiguraatiota. (Mink documentation 2015.)

Molemmissa emulaattorityypeissä on siis hyviä sekä huonoja puolia. Testejä ei kannata koostaa käyttämällä pelkästään selainohjaimia, koska testeistä tulisi erittäin hitaita. Ainoastaan Headless Browser -emulaattoreiden hyödyntäminen ei myöskään ole mahdollista, koska silloin ei pystytä testaamaan Ajax-toimintoja. Tämän takia testauksessa kannattaakin hyödyntää kumpaakin emulaattorityyppiä. (Behat Documentation n.d.)

Behat ei yksin pysty hyödyntämään molempia edellä mainittuja selainemulaattoreita, koska niillä on erilaiset ohjelmointirajapinnat ja ne toimivat eritavoin. Tähän ongelmaan on ratkaisuna Mink, joka on selainemulaattorin abstraktiotaso (engl. Browser Emulator Abstraction Layer). Mink piilottaa emulaattoreiden erot yhtenäisen ohjelmointirajapinnan taakse ja mahdollistaa näin useamman emulaattorin käytön testauksessa. Mink tulee integroida osaksi Behatia, jonka jälkeen sitä voidaan hyödyntää Behatin automaatiotesteissä. (Behat Documentation n.d.)

5 Automaatiotestien optimointi

Tässä luvussa esitellään tarkemmin tämän toiminnallisen opinnäytetyön käytännön toteutusta. Aluksi tarkastellaan automaatiotestien käytön taustoja, jonka jälkeen esitellään optimoinnin kohteet ja kerrotaan, miten niitä kehitettiin.

5.1 Testien tausta

LAURA Rekrytointi Oy käyttää automaatiotestaustyökaluna Behatia. Behatia käytetään yrityksen Rekrytointi.com -rekrytointisivuston käyttöliittymän regressiotestauksessa. Automaatiotestaustyökalu on otettu osaksi sivuston päivittäistä kehitystyötä vuoden 2019 alussa.

Rekrytointi.com -sivusto on tarkoitettu kohtaamispaikaksi työnhakijoille ja rekrytoijille. Sivusto mahdollistaa rekrytoijille avoimien työpaikkojen ja erilaisten koulutusten julkaisemisen. Vastaavasti hakijat voivat etsiä sivustolta itselleen uusia uramahdollisuuksia. Sivustolta löytyy myös erilaisia artikkeleita työnhakuun ja rekrytointiin liittyen, niin työnhakijoille kuin myös rekrytoijille. Automaatiotestejä ei kuitenkaan käytetä tällä hetkellä artikkelipuolen testaamiseen. Automaatiotestit eivät myöskään kata vielä kaksikielisen sivuston englanninkielistä puolta.

Automaatiotestejä käytetään helpottamaan koodimuutosten aiheuttamien virheiden löytämistä. Lisäksi ne nopeuttavat testaamista ja parantavat työnlaatua. Automaatiotestien käyttöönotolla on alun perin haluttu helpottaa isojen lisäosapäivityksien testaamista, koska päivitysten muutoksilla on usein vaikutusta koko sivustoon. Koko sivuston testaaminen manuaalisesti vie paljon aikaa, ja kaikkien virheiden löytäminen on haastavaa. Automaatiotestejä hyödynnetään kuitenkin myös uusien toiminnallisuuksien ja virhekorjauksien testaamisessa.

Automaatiotestit suoritetaan silloin, kun issuen tekninen toteutus on kehittäjän toimesta tehty. Issuella tarkoitetaan yksittäistä työtehtävää, joka voi olla esimerkiksi uusi toiminnallisuus, virheen korjaus tai sivuston lisäosan päivitys. Lähes aina, kun sivuston koodipohjaan tehdään muutoksia, automaatiotestit ajetaan läpi. Automaatiotestit suoritetaan aina sen kehittäjän toimesta, joka on tehnyt kyseisen issuen toteutuksen.

Automaatiotestit suoritetaan tällä hetkellä kehittäjän lokaalissa kehitysympäristössä ja niiden käynnistys tapahtuu manuaalisesti komentoriviltä. Testien suorituksen jälkeen kehittäjä tarkistaa testien tulokset. Jos jokin suoritetun testijoukon testeistä ei ole mennyt läpi, kehittäjä selvittää, tarvitaanko koodiin korjauksia vai täytyykö testiä päivittää. Mikäli testeissä ei näy virheilmoituksia, siirretään issue erilliseen testiympäristöön manuaalista testausta varten.

Testattava testijoukko sisältää tällä hetkellä kahdenlaisia skenaarioita. Skenaarioita, jotka suoritetaan Goutte-testeinä, ja skenaarioita, jotka suoritetaan Selenium-testeinä. Goutte-testit suoritetaan Goutte Headless Browserissa GoutteDriver -ajurin avulla. Goutte kuuluu siis Headless Browser -emulaattoreihin, joka mahdollistaa testien nopean suorittamisen, mutta se ei tue JavaScriptiä tai Ajax-toimintoja (Mink documentation 2015). Selenium-testit taas suoritetaan Selenium2 -selainohjaimen avulla käyttäen Selenium2Driver -ajuria. Se mahdollistaa tuen JavaScriptille ja Ajaxille, mutta testit ovat huomattavasti Goutte-testejä hitaampia (Mink documentation 2015).

Automaatiotestit sisälsivät ennen optimointityön aloittamista 20 toimintoa. Nämä toiminnot sisälsivät taas yhteensä 35 skenaariota. Näistä yksittäisistä testiskenaarioista muodostuu regressiotestaus kokonaisuus, jonka testaus suoritetaan Behat -työkalun avulla. Näistä 35 skenaariosta 28 olivat Selenium-testejä ja seitsemän Goutte-testejä. Testijoukon kokonaissuoritusaika oli noin 15 minuuttia.

5.2 Kehityskohtien valinta

Automaatiotestien kehityskohtien löytämiseksi haastateltiin kahta Rekrytointi.com -sivuston kehitystiimin kehittäjää. Haastattelua päätettiin hyödyntää kehityskohtien löytämisessä, sillä se mahdollistaa automaatiotesteihin liittyvien kokemuksien ja havaintojen jakamisen helposti. Haastattelussa kartoitettiin automaatiotestauksen nykytilannetta ja kehityskohtia, joihin opinnäytetyössä tulisi keskittyä, sekä tulevaisuuden tavoitteita.

Haastattelussa tuli ilmi erilaisia potentiaalisia kehityskohtia, joista valittiin kiireellisimmät ja tärkeimmät. Automaatiotestien kehityskohdiksi ja kehittämistyön konkreettisiksi tavoitteiksi valikoituivat testien suoritusnopeuden puolittaminen, testitulosten taltioinnin kehittämien sekä väärien virheilmoituksien karsiminen.

Automaatiotestit olivat olleet tässä vaiheessa kuitenkin vasta vähän aikaa käytössä, joten oli vaikea sanoa, tuliko haastattelussa esille kaikki tärkeimmät kehityskohdat. Testien aktiivinen käyttö kuitenkin vahvisti valittujen kehityskohtien tärkeyden.

5.3 Testien suoritusnopeus

Ensimmäiseksi kehityskohdaksi valikoitui automaatiotestien suoritusnopeus. Koko automaatiotestijoukon suorittaminen kesti ennen optimointia noin 15 minuuttia. Testit ajetaan kehittäjän toimesta keskimäärin kerran päivässä tai kerran kahdessa päivässä, riippuen työstettävän issuen laajuudesta ja työmäärästä. Testit eivät aina mene ensimmäisellä suorituskerralla läpi ilman virheilmoituksia, joka tarkoittaa usein sitä, että testit tai kehittäjän koodi tarvitsee korjauksia. Korjausten jälkeen on hyvä varmistaa, että testit ja toiminnallisuudet toimivat kuten pitää, jonka takia testejä voidaan joutua suorittamaan useampaan kertaan. Rekrytointi.com -sivustoa myös kehitetään jatkuvasti, minkä takia testattavien toimintojen määrä tulee lisääntymään. Näiden takia on tärkeää, että automaatiotestien suoritusaika saadaan minimoitua.

Kuten aiemmin mainittiin, testattava testijoukko sisälsi 35 skenaariota, joista suurin osa oli Selenium-testejä. Testien toimintaa lähdettiin tarkastelemaan yksi skenaario kerrallaan. Testien toiminnan tarkastelussa kävi ilmi, että moni testattava toiminnallisuus, jota testattiin Selenium-testinä, ei tarvinnut tukea Ajaxille ja JavaScriptille. Tämä tarkoittaa sitä, että kyseisten testien suorituksen kestoa on mahdollista lyhentää, jos testit pystytään muuttamaan Goutte-testeiksi ja suorittamaan nopeamman GoutteDriverin avulla.

Testiskenaarioiden muuttaminen Selenium-testeistä Goutte-testeiksi on usein suhteellisen helppoa. Aluksi testaustyökalulle täytyy kertoa, että skenaario suoritetaan Selenium2Driverin sijasta GoutteDriverilla. Tämä onnistuu poistamalla skenaarion alusta javascript-tagin, joka

kertoo Behatille, että testissä pitää käyttää Selenium2Driveria (Kuvio 5). Tämän jälkeen skenaario vaatii usein kuitenkin myös muokkaamista, koska skenaariossa saattaa olla vaiheita, jotka suoritetaan JavaScriptin avulla.

```
@javascript
Scenario: 00001 Example scenario with javascript tag
    Given I visit "/"
    Then I should see "Some content"
    And I should see "Some other content"
```

Kuvio 5: Selenium2Driverin testit merkataan javascript -tagin avulla.

Toinen tapa, jolla testien suoritusaikaa lyhennettiin, oli usein toistuvien vaiheiden optimointi. Usein toistuvilla vaiheilla tarkoitetaan sellaisia vaiheita, jotka esiintyvät monissa eri skenaarioissa. Hyvä esimerkki tällaisesta oli käyttäjän sisäänkirjautuminen. Sisäänkirjautuminen tapahtuu useassa skenaariossa, koska monen sivuston toiminnon testaaminen vaatii kirjautuneen käyttäjän. Sisäänkirjautumisvaiheen suorittava funktio sisälsi vaiheita, jotka pystyttiin suorittamaan ainoastaan Selenium2Driverilla ajetuissa testeissä. Tämä tarkoittaa sitä, että jokainen sisäänkirjautumista tarvitseva skenaario jouduttiin suorittamaan Selenium2Driverilla, vaikka testattava toiminnallisuus olisi muuten voitu suorittaa GoutteDriverilla. Funktiota muokattiin niin, että sitä pystyy käyttämään molempien ajureiden testeissä.

Useissa skenaarioissa esiintyi myös vaiheita, joissa odotettiin jonkin elementin ilmestymistä käyttöliittymään. Behat suorittaa usein testien vaiheita nopeammin kuin testattava ohjelmisto pystyy toimimaan (Behat Documentation n.d.). Tähän ongelmaan käytetään funktioita, joiden avulla suoritettaviin vaiheisiin pystytään lisäämään odotusaikaa, ennen kuin vaihe varsinaisesti yritetään suorittaa. Käytännössä siis voidaan määrittää esimerkiksi, että nappulan klikkauksen jälkeen odotetaan viisi sekuntia, jonka jälkeen tarkastetaan, näkyykö käyttöliittymässä testiin määritetty elementti. Testeissä hyödynnettiin funktiota, johon määritettiin kiinteä odotusaika, jonka jälkeen testin vaihetta yritettiin suorittaa. Tämä ei ole paras mahdollinen tapa, koska jos elementti latautuukin odotusajaksi määritettyä aikaa nopeammin, ei testin suoritusta kuitenkaan jatketa, ennen kuin koko odotusaika on kulunut. Ongelmaa varten testeihin otettiin käyttöön funktio, joka yrittää suorittaa vaihetta yhden sekunnin välein, niin kauan kunnes se onnistuu, tai määritetty odotusaika umpeutuu. Näin vältytään ylimääräiseltä odottelulta, ja odotusajaksi voidaan varmuudeksi määrittää pidempi aika, jotta vaihe menee läpi myös silloin kun järjestelmä toimii normaalia hitaammin.

Testien suoritusnopeutta parannettiin myös karsimalla samoja toimintoja testaavia osioita eri skenaarioista. Tällaiset tapaukset olivat tilanteita, joissa jonkin toiminnon testaamisen ohella testattiin myös toisen toiminnon toimivuus. Yksi esimerkkitapaus oli hakuvahtitoiminnon tes-

taaminen. Hakuvahtitoiminnon avulla käyttäjä pystyy tilaamaan omaan sähköpostiinsa ilmoituksia hänelle sopivista työ- ja koulutuspaikoista. Hakuvahtitoiminto esiintyy työ- ja koulutuspaikkalistaussivulla sekä yksittäisissä ilmoituksissa. Hakuvahtitoiminnolle oli tehty oma testinsä, jossa testattiin sekä yksittäisessä ilmoituksessa esiintyvä hakuvahtipainike että listaussivulla esiintyvä painike. Tämän lisäksi kuitenkin myös työ- ja koulutuspaikkojen listaussivulle tehdyssä testissä testattiin jo kertaalleen testattua hakuvahtitoiminnon toimivuutta.

LAURA Rekrytointi Oy:llä suoritetaan issuen toteutuksen jälkeen koko regressiotestijoukko, jotta voidaan varmistua, että tehdyt muutokset eivät ole aiheuttaneet ongelmia sivuston toimintoihin muualla. Samojen toiminnallisuuksien testaaminen useammassa kuin yhdessä skenaariossa ei ole kannattavaa, koska kaikki automaatiotestit suoritetaan yhdellä kertaa. Toistuvat samat testit ainoastaan lisäävät testien suoritusaikaa.

Testien suoritusnopeuden parantamiseksi karsittiin myös ylimääräisiä vaiheita yksittäisistä skenaarioista. Testijoukossa oli kaksi testiä, jotka oli rakennettu hyvin monivaiheisiksi. Molemmissa testeissä testattiin tekstisyötekenttien avulla käyttäjän syöttämien arvojen tallentumista. Testeissä syötettiin aina yhteen tekstisyötekenttään kerrallaan jokin arvo, jonka jälkeen suoritettiin tallennus. Tallennuksen jälkeen tarkastettiin, onko tieto tallentunut järjestelmään. Tämä suoritettiin jokaiselle tekstisyötekentälle erikseen. Näin tehtynä testin suorittaminen on hidasta. Testejä muutettiin niin, että syötekenttiin annetaan samalla kertaa kaikki arvot, joiden tallentuminen tarkastetaan myös yhdellä kertaa.

5.4 Testitulosten taltiointi

Kehitystiimin haastattelussa ilmeni puutteita testitulosten taltioinnissa, minkä takia se valikoitui toiseksi kehityskohteeksi. Etenkin testitulosten tarkastelumahdollisuuksissa nähtiin parantamisen varaa. Ongelmana oli se, että testitulokset eivät jääneet talteen. Myös testitulosten raportointitapa koettiin epäkäytännöllisenä, ja raportti suppeana.

Käytössä olevat automaatiotestit suoritetaan komentorivin avulla. Testijoukon suorituksen aikana komentoriville tulostuu testien nimet, kuvaukset, skenaariot ja vaiheet, sitä mukaan, kun niitä käydään läpi. Näin testien etenemistä on mahdollista seurata suorituksen aikana ja tarkastella tuloksia suorituksen jälkeen.

Testijoukon suorituksen jälkeen komentoriville tulostuu raportti, josta ilmenee skenaarioiden ja vaiheiden määrä sekä testijoukon suoritusaika. Raportista näkee myös, kuinka monta skenaariota ja vaihetta on epäonnistunut sekä epäonnistuneiden testien nimet ja niissä virheen aiheuttaneiden vaiheiden rivinumero koodissa (Kuvio 6). Mikäli testisuorituksessa epäonnistunut skenaario on suoritettu Selenium2Driverin avulla, tallennetaan skenaarion epäonnistuneesta vaiheesta näyttökuva. Näyttökuvassa näkyy kuva selaimen ikkunasta sillä hetkellä, kun virhe on tapahtunut testissä. Näyttökuvat tallentuivat erilliseen Screenshots -kansioon.

```
--- Failed scenarios:

    features/00100_job_creation.feature:8
    features/00660_order_history.feature:7

36 scenarios (34 passed, 2 failed)
1102 steps (1064 passed, 2 failed, 36 skipped)
21m47.82s (11.19Mb)
```

Kuvio 6: Komentoriville tulostuva raportti.

Kehitystiimissä toivottiin, että jokaisesta yksittäisestä testistä olisi saatavilla tarkempaa dataa. Oleellisimmiksi tiedoiksi valittiin testin suoristusaika sekä testin vaiheiden määrä. Toiveena oli myös raportoinnin muuttaminen niin, että testijoukosta jäisi statistiikkaa talteen, esimerkiksi erilliseen html-tiedostoon. Tämä mahdollistaisi pitkäaikaisemman testien toiminnan ja suorituskyvyn seurannan sekä jatkokehittämisen.

Testitulosten raportin selkeyttä ja sen tarkastelumahdollisuuksia haluttiin myös parantaa. Testitulokset tulostuivat ainoastaan komentoriville, jonka jälkeen joutui selaamaan yli tuhat riviä pitkää tulostetta ja etsimään suorituksessa epäonnistuneen testivaiheen testijoukosta. Vaikka epäonnistuneet vaiheet näkyivätkin punaisella värillä korostettuna, kehitystiimissä koettiin tulosten tarkastelumahdollisuus epäkäytännölliseksi (Kuvio 7).

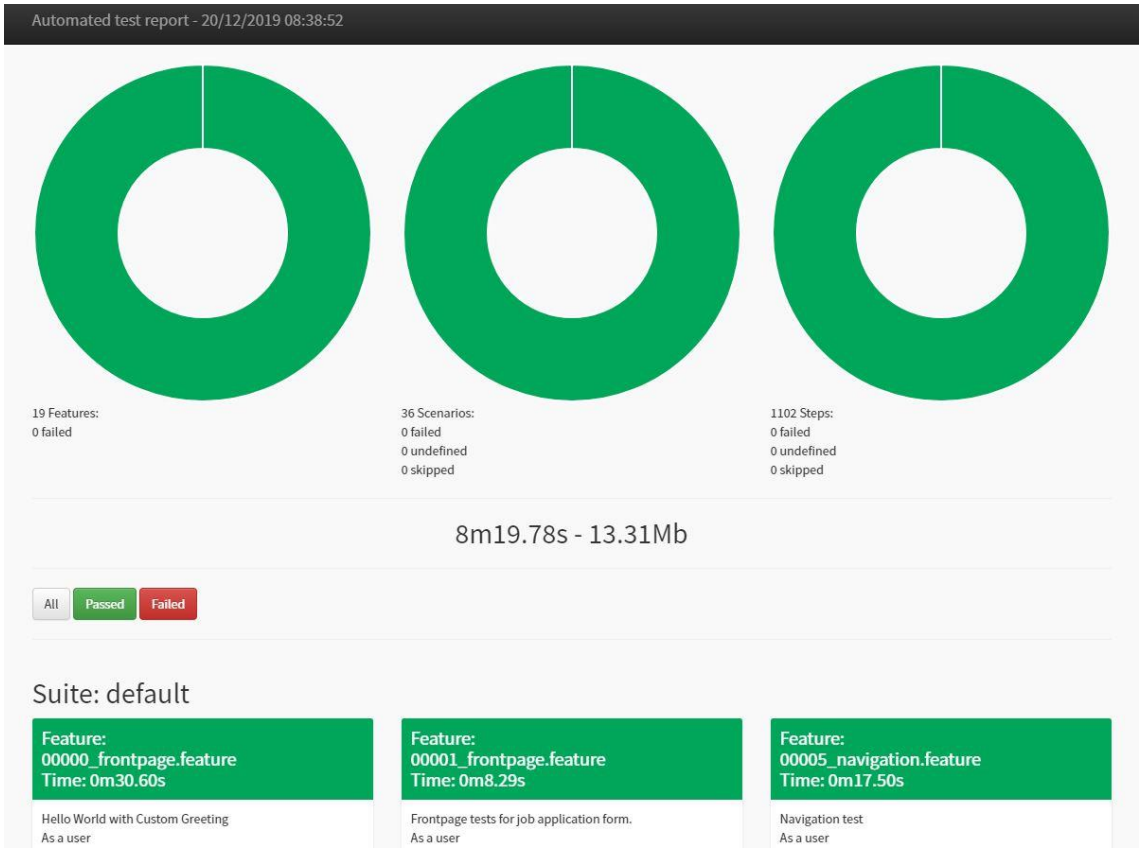
```
@smoke
Scenario: 00000 Test that footer copyright is correct
  Given I visit "/"
  Then I should see a ".site-info" element
  And I should see "LAURA Rekrytointi Oy" within ".site-info"

@smoke
Scenario: 00000 Test that navigation has correct elements
  Given I visit "/"
  Then I should see a "#mega-menu-primary" element
  And I should see "Työpaikat" within "#mega-menu-primary"
  And I should see "Koulutukset" within "#mega-menu-primary"
  And I should see "Uravinkit" within "#mega-menu-primary"
  And I should see "Ilmoittajalle" within "#mega-menu-primary"
  And I should see "Jätä ilmoitus" within "#mega-menu-primary"
  And I should see "Testi" within "#mega-menu-primary"
    Did not find text in selector:#mega-menu-primary (Exception)
  |
  □ Screenshots are not supported by Behat\Mink\Driver\GoutteDriver (Behat\Mink\Exception\UnsupportedDriverActionException)
  |
  └ @AfterStep

@javascript
Scenario: 00000 Areas of expertise widget test

  Given I visit "/"

  Then I should see "OSAAMISALUEET"
```

Kuvio 7: Epäonnistunut vaihe näkyy tulosteessa punaisena.

Testitulosten taltiointimahdollisuuksien parantamiseksi asennettiin ja konfiguroitiin Behatin lisäosa BehatHtmlFormatter. Lisäosa mahdollistaa testitulosten keräämisen talteen, ja ennen kaikkea tulosten selkeän ja kattavan esitysmallin. BehatHtmlFormatter -lisäosa tulostaa erillisen html-tiedoston, jossa testiajokerran tulokset ovat kuvattu graafien ja selkeiden elementtien avulla.

Lisäosan luomaa raporttia sanotaan Twig-raportiksi. Raportti on toteutettu Twig-mallimoottorin (engl. Twig Template Engine) avulla. Twig on PHP-ohjelmointikielelle luotu mallimoottori, joka mahdollistaa mallien kääntämisen yksinkertaiseksi ja optimoiduksi PHP-koodiksi. (Twig n.d.).

Twig-raportista on helposti nähtävissä testijoukon tulokset. Raportti on interaktiivinen raportti, josta pystyy erikseen tarkastelemaan jokaisen testin skenaarioita ja niiden vaiheita. Raportista pystyy myös suoraan avaamaan epäonnistuneiden skenaarioiden ottamat näyttökuvat kyseisten skenaarioiden kohdalta.

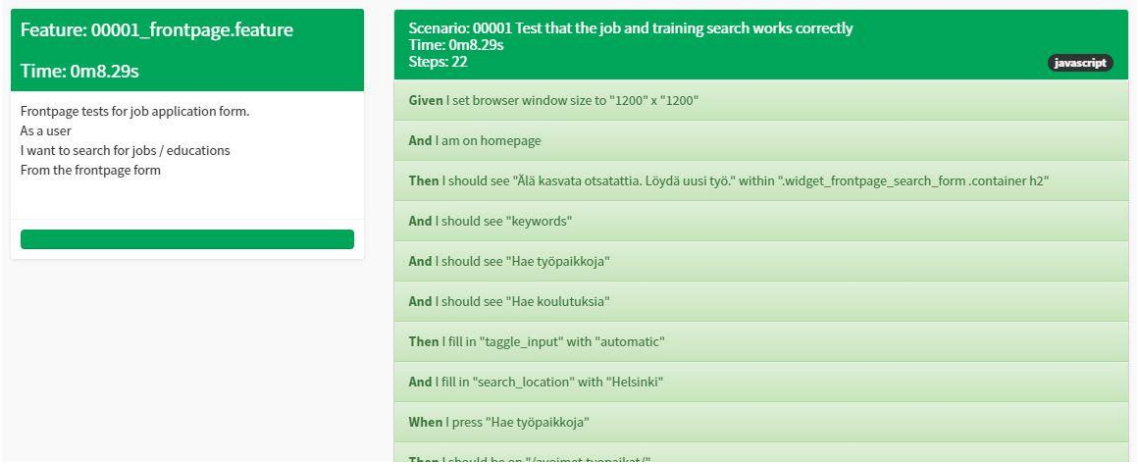


Kuvio 8: Twig-raportti

BehatHtmlFormatter -lisäosan konfiguroinnin jälkeen Twig-raportti oli jo huomattava kehitysaskel testitulosten taltioinnissa edelliseen verrattuna, mutta siihen haluttiin kuitenkin tehdä vielä lisäyksiä. Lisäosan koodipohjaan tehtiin omia muokkauksia, jotta raportista saatiin toivotunlainen.

Raporttiin lisättiin testijoukon suoritusaika, joka siitä vielä puuttui. Jokaiselle toiminnallisuudelle ja skenaariolle lisättiin myös erilliset suoritusajat, ja skenaarioille laskettiin niiden vaiheiden määrät (Kuvio 9). Nämä muutokset mahdollistavat yksittäisten testien tarkemman ja

helpomman tarkastelun, jonka ansiosta on helpompi tehdä päätelmiä niiden toimintakyvystä sekä kehitystarpeista.



Kuvio 9: Yksittäisen testin tarkastelu Twig-raportissa.

Twig-raportit asetettiin tallentumaan erilliseen raporttikansioon. Raportin nimeksi asetetaan tallennushetken päivämäärä ja kellonaika sekä kehityshaaran nimi, jossa automaatiotestit on suoritettu. Näin testituloksiin on helpompi palata myöhemmin ja tiedetään, minkä issuen yhteydessä testit on suoritettu.

5.5 Väärät virheilmoitukset

Kolmanneksi kehityskohteeksi valittiin ongelmallisiksi koetut väärät virheilmoitukset. Näillä tarkoitetaan työkalun palauttamia virheilmoituksia tilanteista, joissa testattavassa toiminnossa ei todellisuudessa ole virheitä (ISTQB 2018, 16). Väärät virheilmoitukset ovat yleisiä varsinkin käyttöliittymää testattaessa (Hoffman 2003, 20). Testattava käyttöliittymä on muuttuva, joka tekee siitä alttiin tällaisille ongelmille.

Yleisimmät tilanteet, joissa ilmeni vääriä virheilmoituksia, olivat tapaukset, joissa Behat yrittää suorittaa testiskenaarion vaiheita liian nopeasti. Skenaariossa voidaan esimerkiksi ladata jokin tietty näkymä, jonka jälkeen yritetään välittömästi paikantaa skenaarion seuraavaan vaiheeseen määritetty elementti, kuten nappula. Nappula ei kuitenkaan välttämättä ole kerennyt vielä latautumaan sivulle, jonka takia testityökalu ei löydä nappulaa ja antaa siitä virheilmoituksen. Tällaisten virheilmoitusten vähentämiseksi skenaarioihin lisättiin vaiheita, jotka yrittävät paikantaa tiettyä elementtiä, kuten nappulaa, niin pitkään kunnes elementti ilmestyy tai kun ennalta määritetty odotusaika on täyttynyt.

Vääriä virheilmoituksia esiintyi myös tilanteissa, joissa sivuston elementtiä yritetään paikantaa käyttöliittymässä näkyvän tekstin perusteella. Tämä ei ole paras mahdollinen tapa toimia,

koska pienikin käyttöliittymän tekstimuutos aiheuttaa virheilmoituksen. Sen sijaan elementtien paikantamiseen olisi hyvä käyttää esimerkiksi elementeille määritettyjä uniikkeja nimiä tai ID-kenttiä (Automated Testing Best Practices and Tips 2019). Tämä tulisi kuitenkin ottaa huomioon jo uusia toiminnallisuuksia kehitettäessä, jotta toiminnallisuuden testauksessa voidaan hyödyntää kyseisiä kenttiä. Vastaavanlaisia elementin kohdistamiseen liittyviä virheilmoituksia ilmeni myös tilanteissa, joissa elementtiä on liikutettu paikasta A paikkaan B, jolloin testityökalu ei osaa etsiä elementtiä oikeasta paikasta, ja antaa virheilmoituksen.

Tällaisten virheilmoitusten vähentämiseksi elementtien kohdistuksia muutettiin niiltä osin, kun oli mahdollista. Kaikissa tapauksissa tämän tekeminen ei kuitenkaan onnistunut, koska kaikille testattaville elementeille ei ollut määritettynä uniikkia ID:tä tai luokkaa. Nämä tulisi itse lisätä koodiin, mutta opinnäytetyön aikataulussa pysymisen kannalta ei ollut järkevää lähteä tätä tekemään.

Moni käytössä oleva testi suorittaa myös järjestelmään sisäänkirjautumisen, koska usea sivuston toiminto vaatii kirjautuneen käyttäjän. Joissain tilanteissa sisäänkirjautuminen kuitenkin epäonnistui, vaikka kirjautuminen sivustolla toimi normaalisti. Nämä virheilmoitukset näyttivät olevan aivan sattumanvaraisia, koska ne ilmenivät usein eri testiskenaarioiden kohdalla. Kirjautuminen tapahtui WordPress-ohjelmiston vakiokirjautumisnäkymässä, joka ei ole oikeille käyttäjille näkyvissä. WordPress on julkaisujärjestelmä, jota käytetään Rekrytointi.com -sivuston alustana. Tämän ongelman ratkaisemiksi kirjautumisvaiheen suorittavaa funktiota, muokattiin niin, että kirjautuminen suoritetaan Rekrytointi.com -sivuston normaalissa kirjautumisnäkymässä. Tämä on näkymä, jota todelliset palvelun käyttäjät eli työnantajat sekä työnhakijat käyttävät kirjautuessaan palveluun. Muutoksella onnistuttiin karsimaan sisäänkirjautumisessa ilmenneet ongelmat ja myös testien kirjautumisvaihe vastaa nyt paremmin todellista käyttötilannetta.

Eräs väärien virheilmoitusten aiheuttaja oli testien riippuvuus toisista testeistä. Esimerkkinä tästä voidaan käyttää työpaikkailmoituksen jättämiseen käytettyä testiä sekä sähköpostin toimivuutta testaavaa testiä. Tiivistetysti kerrottuna, ensimmäisessä testissä kirjaudutaan työnantajakäyttäjänä palveluun, tehdään uusi työpaikkailmoitus ja julkaistaan se sivustolle. Työpaikkailmoituksen julkaisusta järjestelmä lähettää käyttäjälle automaattisesti vahvistusviestin sähköpostiin. Tätä vahvistusviestiä ei kuitenkaan tarkasteta samassa testissä, vaan vasta myöhemmin suoritettavassa sähköpostin toimivuutta testaavassa testissä. Tämä tekee sähköpostitestin onnistumisesta täysin riippuvaisen toisen testin onnistumisesta. Jos työpaikkailmoituksen luominen epäonnistuu ensimmäisessä testissä, epäonnistuu myös sähköpostitesti, koska työpaikanluonnin vahvistusviestiä ei ole lähetetty.

Testit ovat tarkoitettu toimivan eristetysti, toisistaan riippumattomina. Näin pystytään mahdollistamaan myös yksittäisten testien suorittaminen koko testijoukon läpiajamisen sijaan.

Tällaisten testien välisten riippuvuuksien löytämistä ja niiden poistamista varten käytiin kaikki testiskenaariot läpi, suorittaen ne yksitellen. Toisistaan riippuvaiset testit muokattiin toimimaan itsenäisesti tai yhdistettiin yhdeksi testiksi.

6 Kehittämistyön tulokset

Kehittämistyön tavoitteena oli käytössä olevien automatisoitujen käyttöliittymän regressiotestien suoritusajan puolittaminen, testitulosten taltioinnin kehittäminen ja väärien virheilmoitusten karsiminen. Testitulosten taltioinnin ja väärien virheilmoitusten osalta tavoitteet voidaan todeta onnistuneiksi. Suoritusajan nopeuttamisessa ei aivan päästy tavoitteeseen. Alkuperäistä suoritusaikaa onnistuttiin lyhentämään 15 minuutista alle kahdeksaan ja puoleen minuuttiin.

Testien suoritusajan parantamiseksi tehtiin paljon pieniä toimenpiteitä, joiden avulla suoritusaikaa onnistuttiin lyhentämään merkittävästi, vaikka tavoitetta ei saavutettukaan. Suoritusajan parantamisessa otettiin huomiin myös testien laatu. Testien nopeuttamiseksi pyrittiin tekemään vain sellaisia muutoksia, jotka eivät vaikuta testien laatuun ja luotettavuuteen negatiivisesti.

Testitulosten taltiointiominaisuuksia saatiin parannettua huomattavasti. Aiemmin testitulosraportti tulostettiin ainoastaan komentoriville, eikä tuloksista jäänyt minkäänlaista raporttia talteen. Yksittäisistä testeistä saatiin vain tieto siitä, onko testi mennyt läpi vai ei. Opinnäytetyön tuotoksena testitulokset jäävät nyt talteen html-tiedoston muodossa. Testitulosten tarkastelu on huomattavasti aiempaa helpompaa, ja testituloksista saadaan nyt myös testiskenaario kohtaisia tietoja.

Väärien virheilmoitusten osalta onnistuttiin myös hyvin. Aiemmin testijoukon suorituksessa ilmeni useasti tapauksia, joissa jokin testi on antaa virheilmoituksen, vaikka testattavassa toiminnallisuudessa ei ole vikaa. Tällaiset opinnäytetyön aikana esiintyneet virheilmoitukset onnistuttiin karsimaan pois. Tämä ei kuitenkaan tarkoita, että vääriä virheilmoituksia ei enää tulevaisuudessa ilmenisi. Testattava käyttöliittymä on muuttuva ja käyttöliittymän automaatiotestit ovat yleisestikin alttiita tällaisille virheilmoituksille.

7 Johtopäätökset ja jatkokehitysideat

Rekrytointi.com -kehitystiimissä opinnäytetyön tuotos koettiin hyödylliseksi. Varsinkin testitulosten uutta raportointityökalua kehuttiin erittäin käteväksi. Automaatiotestit ovat nyt myös nopeampia, ja testeissä esiintyneitä virheilmoituksia saatiin karsittua. Automaatiotestien kehittämisellä testauksesta saatiin nopeampaa ja vaivattomampaa, mikä näkyy eniten kehittäjien työajan säästönä.

Kehittämistyön aikana myös oma osaamiseni kehittyi. Ennen kehittämistyön aloittamista osaamiseni automaatiotesteihin liittyen oli melko vähäistä. Kokemukseni koostui lähinnä yksittäisten testiskenaarioiden kirjoittamisesta. Kehittämistyön toteuttaminen syvensi ymmärrystäni automaatiotestien toiminnasta ja niiden käyttömahdollisuuksista. Ennen kaikkea Behat -työkalun käyttötaitoni kehittyi merkittävästi.

Automaatiotestien kehittämistä tullaan jatkamaan tulevaisuudessa. Seuraava vaihe jatkokehittämisessä on testiskenaarioiden rinnakkain suorittamisen mahdollistaminen. Tämä tarkoittaa sitä, että useampaa testiskenaarioita olisi mahdollista suorittaa samanaikaisesti, mikä tehostaisi ja nopeuttaisi huomattavasti testausprosessia. Testien rinnakkain suorittamisen mahdollistaminen vaatii kehitystiimin 32-bittisten kehitysympäristöjen päivittämisen 64-bittisiksi ympäristöiksi.

Toinen jatkokehitystoimi on automaatiotestauksen laajentaminen Rekrytointi.com -sivuston englanninkieliselle käyttöliittymälle. Tämän avulla testauksesta saadaan kattavampaa ja luotettavampaa. Tätä ennen tulee kuitenkin mahdollistaa testien rinnakkain suorittaminen, jotta testien suoritusaika ei kasva nykyisestä.

Lähteet

Painetut

Burnstein, I. 2003. Practical software testing. Department of Computer Science.

Homes, B. 2012. Fundamentals of software testing. Iso-Britannia: London Wiley.

Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. Docendo

Sähköiset

Behat. N.d. A php framework for autotesting your business expectations. Viitattu 12.7.2019. https://behat.org/en/latest/index.html

Behat. N.d. Behat Documentation. Viitattu 12.7.2019. https://behat.org/en/latest/guides.html

Codecademy. 2020. List of Command Line Commands. Viitattu 7.1.2020. https://www.codecademy.com/articles/command-line-commands

Hoffman, D. 2003. A Course on Software Test Automation Design. Viitattu 31.5.2019. http://www.testingeducation.org/course_notes/hoffman_doug/test_automation/auto8.pdf

International Software Test Institute™. 2019. Software Testing Levels. Viitattu 14.5.2019. https://www.test-institute.org/Software_Testing_Levels.php

International Software Testing Qualifications Board (ISTQB). 2018. Certified Tester Foundation Level Syllabus. Viitattu 31.5.2019. https://www.istqb.org/downloads/send/51-ctfl2018/208-ctfl-2018-syllabus.html

International Software Testing Qualifications Board (ISTQB). 2014. Foundation Level Extension Syllabus Agile Tester. Viitattu 8.2.2020. https://www.istqb.org/downloads/send/5-foundation-level-agile-tester/41-agile-tester-extension-syllabus.html

Jyväskylän yliopisto. N.d. Testauksen tasot. Viitattu 14.5.2019. http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus/testauksen-tasot

Mink documentation. 2015. Mink at a Glance. Viitattu 15.1.2020. http://mink.behat.org/en/latest/at-a-glance.html

Pettichord, B. 2001. Succes with Test Automation. Viitattu 12.7.2019. http://ceng557.cankaya.edu.tr/uploads/files/Success%20with%20Test%20Automation.doc

Rouse, M. 2018. Test Automation. Viitattu 06.06.2019. https://searchsoftwarequality.techtarget.com/definition/automated-software-testing

SmartBear. 2019. Automated Testing Best Practices and Tips. Viitattu 12.7.2019. https://smartbear.com/learn/automated-testing/best-practices-for-automation/

Software Testing Help. 2019. What is Automation Testing (Ultimate Guide to Start Test Automation). Viitattu 11.5.2019. https://www.softwaretestinghelp.com/automation-testing-tutorial-1/

Software Testing Help. 2019. How to Choose the Best Automation Testing Tool (A Complete Guide). Viitattu 06.06.2019. https://www.softwaretestinghelp.com/automation-testing-tutorial-4/

Twig - The flexible, fast, and secure template engine for PHP. N.d. Viitattu 1.2.2020. https://twig.symfony.com/

Zambelich, K. N.d. Totally Data-Driven Automated Testing – White Paper. Viitattu 11.5.2019. http://scholar.google.com/scholar_url?url=http://staff.cs.psu.ac.th/Apirada/344-454/White_Paper-Automate%2520testing.doc&hl=fi&sa=X&scisig=AAGBfm3_2-7TrHA4dTiCkwiA5PKeb147-w&nossl=1&oi=scholarr

Kuviot

Taulukot