jamk.fi

Verkkokaupan automaatiotestaus

Tupaq Castro

Opinnäytetyö
Toukokuu 2016
Tekniikan ja liikenteen ala
Insinööri (AMK), Ohjelmistotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Castro, Tupaq	Opinnäytetyö, AMK	17.04.2016
	Sivumäärä 33	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Verkkokaupan automaatiotestaus

Tutkinto-ohjelma
Ohjelmistotekniikka

Työn ohjaaja(t)
Esa Salmikangas

Toimeksiantaja(t)
Solteq oyj

Tiivistelmä

Opinnäytetyön kohteena olleessa verkkokauppa-projektissa on pitkään tehty pelkästään manuaalista testausta. Opinnäytetyön tarkoituksena oli parantaa kehitysprosessia automatisoimalla regressiotestaus, jotta turhaa aikaa ei kuluisi regressiotestauksen manuaaliseen suorittamiseen. Valmiita automatisoituja testitapauksia oli sitten tarkoitus suorittaa osana kehitysprosessia eli testejä suoritettaisiin jatkuvan integraation palvelimella, aina kun verkkokauppaan tehtäisiin muutoksia.

Työn alussa pohdittiin mitkä ovat verkkokaupan päätoiminnallisuudet ja näiden päätoiminnallisuuksien selvittyä päätettiin, että vain näiden toiminnallisuuksien testaaminen tultaisiin automatisoimaan tämän työn puitteissa. Muiden vähemmän tärkeiden ominaisuuksien automatisointi olisi sitten osa jatkokehitystä.

Automatisointi toteutettiin käyttämällä Cucumber, sekä Capybara -nimisiä automatisointi -työkaluja. Työn tuloksena saatiin kattava määrä automatisoituja testitapauksia ja niitä voitiin suorittaa jatkuvan integraation palvelimella. Verkkokaupan ulkoasu on kuitenkin muuttunut siitä, kun testitapauksien automatisointia lähdettiin toteuttamaan, joten suurin osa automaattisista testeistä ei tällä hetkellä toimi.

Johtopäätöksenä voidaankin sanoa, että suurimmassa osassa tavoitteista onnistuttiin, mutta testeistä ei ole vielä hyötyä kehitysprosessille. Jatkokehitysideana olisikin, että olemassa olevat testitapaukset korjattaisiin, jotta testitapaukset voitaisiin suorittaa aina kun verkkokauppaan tulee muutoksia.

Avainsanat (asiasanat)

testaus, automaatiotestaus, verkkokauppa, Cucumber, Jenkins, Capybara

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Castro, Tupaq	Bachelor's thesis	17.04.2016
		Language of publication: Finnish
	Number of pages 33	Permission for web publication: x

Title of publication
Test automation of an online store

Degree programme
Software Engineering

Supervisor(s)
Salmikangas, Esa

Assigned by
Solteq oyj

Abstract

The purpose of the thesis was to improve the developing process of an online store by automating the regression testing. The regression testing has been done manually and it's a very time consuming process. With test automation this time can be saved and used to something else. One of the goals was to run these automated tests as a part of the development process in the continuous integration server every time there is a change in the code.

The first step for test automation was to figure out what the main functionalities of an online store are. Once the functionalities had been figured out, the next task was to create a smoke test set from these main functionalities and automate the testing of the smoke test set. The automation tools used were Cucumber and Capybara.

The thesis results are a smoke test set that can be run on the continuous integration server; however, currently most of the tests fail as there was a layout update for the online store. Thus, the result is actually more like a proof of concept than actual working automated smoke test set that can be run as a part of the actual development process. Nevertheless, as the core for test automation is ready, it is just matter of time until the tests are up and running.

In this thesis testing in general is discussed with the focus on the actual work done. First, the main functionalities picked for the smoke test set are examined and what the tools used for the test automation were and how they were installed. Then there is a chapter on the actual test automation work and what was needed to get the tests to work in the continuous integration server. The last part of the thesis states the results.

Keywords/tags (subjects)
testing, test automation, Cucumber, Jenkins, Capybara

Miscellaneous

Sisältö

Kuviot

1 Työn lähtökohdat

1.1 Toimeksiantaja

Opinnäytetyön toimeksiantaja oli entinen Descom Oy, joka yrityskaupan johdosta on nykyään osa Solteq Oyj:tä. Yrityskaupan myötä Solteqista tuli digitaalisen kaupankäynnin palvelutalo, joka työllistää n. 500 asiantuntijaa ja jonka vuosittainen liikevaihto on noin 68 miljoonaa euroa (proforma 2014). Solteqilla on toimintaa kolmessa eri maassa, ja toimituksia Eurooppaan, Pohjois-Amerikkaan, Aasiaan ja Australiaan. (Solteq lyhyesti 2016.)

1.2 Opinnäytetyön tavoitteet

Opinnäytetyön tarkoituksena oli automatisoida verkkokaupan perustoiminnallisuuksien regressiotestaus. Regressiotestauksen automatisoinnin tarkoituksena oli laskea regressiotestaukseen meneviä kuluja, sekä säästää aikaa ja resursseja. Automatisoinnin avulla myös henkilöistä johtuvien testausvirheiden määrän pitäisi laskea, sillä manuaalinen regressiotestaus on paljon toistoa sisältävä, sekä aikaa vievä prosessi ja manuaalista testausta tekevä henkilö voi tympääntyessään oikaista prosessista. Myös testien hyväksyntäkriteerit tulisivat automaation myötä selkeästi dokumentoiduksi.

Testauksen automatisointia suunniteltiin ja toteutettiin yhdessä verkkokauppa projektin testausasiantuntijan kanssa. Työn alussa vertailtiin erilaisia työkaluja, joiden avulla voidaan toteuttaa web-sovelluksen testauksen automatisointi. Vertailua tehtiin Robot Frameworkin ja Cucumber -nimisten työkalujen välillä ja lopulta päädyttiin käyttämään Cucumberia, sekä siihen liittyviä muita työkaluja.

Kun käytettävät työkalut oli saatu päätettyä, lähdettiin suunnittelemaan mikä olisi regressiotestauksen laajuus eli minkä kaikkien toiminnallisuuksien testaaminen pitäisi automatisoida. Automatisoinnin laajuuden selvittyä, opinnäytetyössä käsitellään testien toteuttamista, sekä niiden suorittamista osana kehitysprosessia jatkuvan integraation palvelimella. Lopuksi käsitellään työssä saatuja tuloksia, sekä pohditaan, että miten työssä onnistuttiin ja mitä opittiin.

2 Ohjelmistotestaus

2.1 Yleistä

Yleisesti puhekielen termillä testaus tarkoitetaan lähes mitä tahansa kokeilemista ja kokeiluja. Ohjelmistojen testauksen puolella testaus taas määritellään perinteisesti suunnitelmalliseksi virheiden etsimiseksi ajamalla ohjelmaa tai jotain tiettyä osaa siitä. Ohjelman tekijän ollessa testaajana tavoitteena on pikemminkin osoittaa ohjelman toimivuus kuin virheiden löytäminen. (Haikala & Märijärvi 2006, 284.)

Ohjelmistotestauksen avulla voidaan myös informoida asiakasta testattavan tuotteen tai palvelun laadusta (Kaner 2006, 11).

Kun testin tulos on oikein, on testi onnistunut. Testin onnistuminen edellyttää, että on olemassa spesifikaatio, jonka perusteella testitapaukset voidaan suunnitella ja jonka avulla voidaan ennalta päätellä hyväksyttävät lopputulokset. (Haikala & Märijärvi 2006, 285.) Toisin sanoen testauksella voidaan myös osoittaa, että ohjelmisto täyttää kaupalliset ja tekniset vaatimukset.

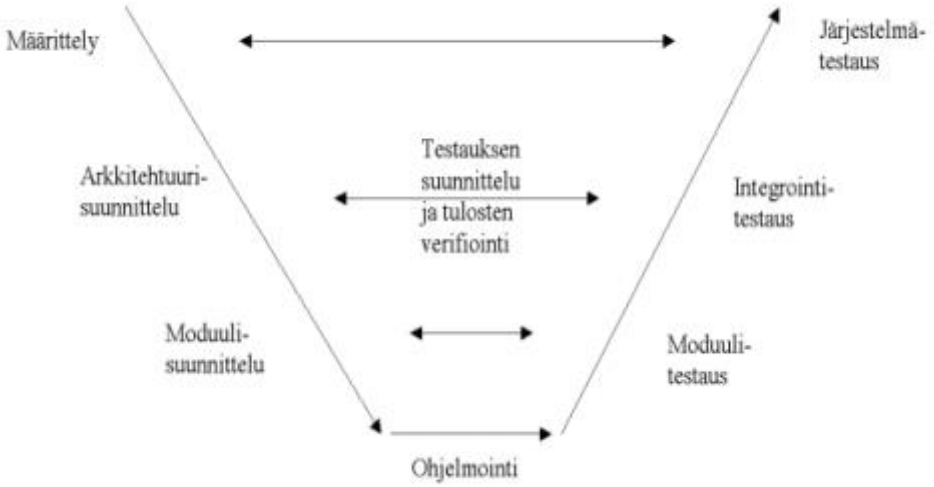
Ohjelmiston testaaminen voidaan suorittaa missä tahansa kehitysvaiheessa riippuen valitusta testaustavasta. Suurin osa testauksesta kuitenkin suoritetaan, kun kaikki vaatimukset on määritelty ja ohjelmointivaihe on suoritettu.

Testauksen avulla ei voida varmistaa ohjelman virheettömyyttä edes yksinkertaisimmissa tapauksissa, mutta sillä voidaan kuitenkin löytää ja osoittaa ohjelmasta löytyvät virheet. Käytännössä testauksen avulla voidaan kattaa vain häviävän pieni murto-osa kaikista mahdollisista ohjelman tilanteista. Tämä ei kuitenkaan tarkoita sitä, ettei-kö testaukseen kannattaisi panostaa, vaan hyvistä testituloksista huolimatta ohjelman toimivuuteen ei kannata liiaksi luottaa. Tämän tarkastelun perusteella korostuu sekä ohjelmoinnin merkitys, että ohjelman huolellinen suunnittelu (Haikala & Marijärvi 2006, 286-287).

2.2 Testaustasot

2.2.1 Yleistä

Haikalan ja Märijärven mukaan (2006, 288) testauksen eri tasoja ns. V-mallin mukaan ovat moduulitestaus, integrointitestaus sekä järjestelmätestaus (ks. kuvio 1). Järjestelmätestausta voi seurata myös erillinen kenttätestaus ja/tai hyväksymistestaus.



Kuvio 1. V-malli *(alkup. kuvio ks. Haikala & Märijärvi 2006, 289)*

Haikala ja Märijärvi (2006, 290) kertovat, että sitä kalliimmaksi virheen korjaaminen tulee, mitä korkeammalla tasolla V-mallissa ollaan. Virheen korjaus voi samalla aiheuttaa myös uusia virheitä ohjelmaan. Esimerkiksi kun järjestelmätestauksessa huomattu virhe korjataan, voi korjaaminen aiheuttaa muutoksia useisiin moduuleihin. Tästä johtuen muut moduulit pitää testata, siltä varalta, ettei jokin muutostarve jäisi huomaamatta. Myös järjestelmätestaus täytyy suorittaa lopuksi uudelleen. Edellä kuvattua uudelleentestausta (testaustasosta riippumatta) kutsutaan regressiotestaukseksi. V-mallin mukaiset testaustasot esitetään luvuissa 2.2.2 - 2.2.4.

2.2.2 Moduulitestaus

Moduuli- eli yksikkötestaus tarkoittaa yksittäisen moduulin testausta. Se koostuu yleensä 100-1000 ohjelmarivistä. Moduuli- ja arkkitehtuurisuunnittelun tuloksia verrataan yksittäisen moduulin toimintaan, yleisesti tekniseen määrittelydokumenttiin. Yksikkötestauksessa moduulin toteuttaja suorittaa yleensä testauksen. (Haikala & Märijärvi 2006, 289.)

Moduulien toimivuutta testattaessa voidaan joutua luomaan testipetejä. Näihin testipeteihin voi sisältyä ohjelman ympäristöä simuloivia osia, normaalisti tynkämoduuleja ja testiajureita. Testiajurit mahdollistavat moduulin suorittamien palvelujen kutsumisen ja tulosten tarkastelun. Tynkämoduulit taas korvaavat testattavan moduulin vaatimat muut moduulit, jos näitä ei ole vielä olemassa. (Mts. 289.)

2.2.3 Integrointitestaus

Integrointitestauksessa päämääränä on yhdistellä moduuleita tai moduuliryhmiä (osajärjestelmiä). Painopisteenä on moduulien välisten rajapintojen toimivuuden tutkiminen. Testauksesta saatuja tuloksia verrataan yleisesti tekniseen määrittelyyn. Integrointi- ja moduulitestausta suoritetaan usein samanaikaisesti ja sen vuoksi näitä kahta onkin usein tarpeetonta tarkastella erikseen. Toisaalta testauksen kattavuuden kannalta modaalitestauksessa on helpompi saavuttaa kunnollinen testikattavuus, sillä testattavan kokonaisuuden kasvaessa on vaikea läpikäydä kaikkea koodia. Yleensä integrointi etenee kokoavasti (bottom up) alimman tason moduuleista ylöspäin. Päinvastaista etenemissuuntaa kutsutaan jäsentäväksi eli osittavaksi (top down) integroinniksi. (Haikala & Märijärvi 2006, 290.)

2.2.4 Järjestelmätestaus

Nimensä mukaan järjestelmätestausvaiheessa testauksen kohteena on koko järjestelmä. Testauksesta saatuja tuloksia verrataan määrittelydokumentaatioon (ohjelmiston toiminnalliseen määrittelyyn) ja asiakasdokumentaatioon. Järjestelmätestauksessa testaajien pitää olla kehitystyöstä mahdollisimman riippumattomia henkilöitä. (Haikala & Märijärvi 2006, 290.)

Järjestelmätestauksessa testataan lisäksi järjestelmän ei-toiminnallisia ominaisuuksia kuten luotettavuustestit, kuormittavuustestit, käytettävyystestit, asennustestit ja niin edelleen. Edellä mainittuja ei-toiminnallisia ominaisuus-testejä voidaan FiSTB eli Finnish Software Testing Board (2012) mukaan määritellä seuraavasti:

Luotettavuustestien avulla määritetään kuinka hyvin ohjelmistotuote suorittaa siltä vaaditut toiminnot määritetyissä olosuhteissa määritetyssä ajassa tai määritetyille toiminnoille.

Kuormittavuustestin avulla mitataan komponentin tai järjestelmän käyttäytymistä kasvavan kuormituksen alla.

Käytettävyystestit avulla määritellään missä määrin ohjelma on ymmärrettävä, helppo oppia ja käyttää sekä houkutteleva käyttäjälle, kun sitä käytetään tietyissä olosuhteissa.

Asennustestien avulla selvitetään, voiko ohjelmistotuotteen asentaa määriteltyyn ympäristöön.

Järjestelmätestaukseen voi lisäksi liittyä hyväksymistestaus sekä kenttätestaus (Haikala & Märijärvi 2006, 290). Hyväksymistestaus ja kenttätestaus käsitellään tarkemmin luvuissa 2.3.1 ja 2.3.2.

2.3 Muut testausmenetelmät

2.3.1 Hyväksymistestaus

Ohjelmistotestauksessa ISTQB eli International Software Testing Qualifications Board (2010, 10) määrittelee hyväksymistestauksen (Acceptance testing) seuraavanlaisesti: Testaus, joka ottaa huomioon käyttäjän tarpeet, vaatimukset ja liiketoiminnan prosessit, joiden kautta määritellään, täyttääkö järjestelmä hyväksymisvaatimukset sekä mahdollistaa käyttäjän, asiakkaan tai muun auktoriteetin joko hyväksymään tai hylkäämään järjestelmän.

ISTQB:n (2016a) mukaan eri hyväksymistestien tyyppejä ovat:

Käyttäjähyväksymistesti (user acceptance test) keskittyy pääsääntöisesti toiminnallisuuteen, täten se varmistaa järjestelmän käyttövalmiuden bisneskäyttäjän näkökulmasta. Testit suorittavat käyttäjät ja sovellusmanagerit.

Operatiivinen hyväksymistesti (Operational acceptance test) tunnetaan myös nimellä tuotannon hyväksymistesti, joka varmistaa, että järjestelmä täyttää operoinnin vaatimukset. Suurimmassa osassa orgnisaatioita järjestelmän ylläpitäjät suorittavat operatiiviset hyväksymistestit ennen järjestelmän julkaisua. Testit voivat sisältää varmuuskopioinnin/palautuksen testaamisen, vikatilasta palautuminen (disaster recovery), ylläpitötehtävät sekä ajoittaisen turvallisuus-haavoittuvuksien tarkastamisen.

Sopimushyväksymistestaus (Contract acceptance testing): Sopimusta tehtäessä hyväksymiskriteerit pitää määritellä virallisesti ja testit suoritetaan näitä hyväksymiskriteerejä vasten.

Säännöstenmukaisuuden hyväksymistestaus (Regulation acceptance testing): Suoritetaan voimassaolevia sääntöjä vasten, kuten valtion asetukset, lakiasetukset tai turvallisuusmääräykset.

2.3.2 Kenttätestaus

FiSTB (2012) määrittelee kenttätestauksen eli betatestauksen seuraavanlaisesti: Potentiaalisten tai jo olemassa olevien käyttäjien ja/tai asiakkaiden muualla kuin kehitysympäristössä suorittama toiminnallinen testaus. Testauksella varmistetaan, että komponentti tai järjestelmä täyttää käyttäjien/asiakkaiden tarpeet ja toimii liiketoimintaprosessien mukaan. Kenttätestausta käytetään usein osana valmisohjelmistojen ulkoista hyväksymistestausta tuottamaan palautetta markkinoilta.

2.3.3 Regressiotestaus

Regressiotestauksen tarkoituksena on varmistaa, ettei muutos ohjelmistoon tai ympäristöön ole aiheuttanut tahattomasti haitallisia sivuvaikutuksia ja että järjestelmä toimii edelleen vaatimusten ja määritysten mukaisesti. Regressiotestit ovat pääsääntöisesti automatisoituja, koska virheen korjaamisessa samaa testiä joudutaan suorittamaan uudelleen monta kertaa ja sen manuaalinen suorittaminen on pitkäveteistä.

Regressiotestit suoritetaan aina kun ohjelma muuttuu, johtui se sitten virheiden korjautumisesta tai muuttuneista/uusista toiminnallisuuksista. (ISTQB 2016b.)

2.3.4 Savutestaus

Savutestaus (Smoke testing) on ohjelmiston alustava testaus, jolla voidaan paljastaa mahdolliset virheet, jotka estävät esimerkiksi mahdollisen julkaisun. Testaaja valitsee joukon testitapauksia kaikista testeistä, jotka varmistavat komponentin tai järjestelmän tärkeimmät toiminnallisuudet. Nämä testitapaukset suorittamalla varmistetaan ohjelman kriittisempien toiminnallisuuksien toimivuus. (ISTQB 2010.)

Savutestauksen päämääränä on määritellä, onko sovellus niin pahasti rikki, että jatkotestaaminen on turhaa. Kanerin, Bachin ja Pettichordin (2002, 95) mukaan savutestit suurin piirtein kattavat tuotteen toiminnallisuudet rajoitetussa ajassa. Jos avaintoiminnallisuudet eivät toimi tai jos kriittiset virheet ovat korjaamatta, tiimin ei kuuluisi tuhlata enempää aikaa asentamiseen tai testaamiseen.

Savutestaus, joka suoritetaan tiettyä käännöstä vasten, tunnetaan myös nimellä käännöksen varmennustestaus (build verfication test). Testaaja voi tämän avulla päättää, kannattaako käännös hyväksyä jatkotestausta varten. Parhaiden käytäntöjen mukaan päivittäinen käännös - ja savutestaus on suositeltavaa.

2.4 Automaatiotestaus

Automaatiotestauksella tarkoitetaan manuaalisen testien suorittamista automaattisesti jollakin automaatiotestaus-ohjelmalla. Tämä edellyttää, että testitapaukset, sekä testitapauksien odotetut tulokset ovat tarkasti määriteltyjä vaatimusmäärittelyjä ja suunnitteludokumentteja vasten. (Zambelich N-d,4.)

Kaikkea testaamista ei kannata automatisoida ja automatisoidut testit eivät korvaa ihmisen tekemää testaamista. Esimerkiksi jos testi suoritetaan vain kerran tai testi on todella kompleksinen, niin silloin ei ole järkeä tuhlata aikaa automatisoidun testin tekemiseen, verifioimiseen ja dokumentoimiseen, joka vie 3-10 kertaa enemmän aikaa verrattuna testitapauksen manuaaliseen suorittamiseen (Kaner 2002, 3).

Automaatiotestauksen tarkoituksena on tehdä asioita joita manuaalisella testauksella ei ole mahdollista tai järkevää toteuttaa. Esimerkiksi paljon toistoa sisältävä, sekä aikaa vievä regressiotestaus kannattaa suorittaa automaatiotestauksen avulla. Toinen hyvä automatisoinnin kohde on kuormitustestaus, jossa automatisoinnin avulla voidaan järjestelmään simuloida kymmeniä, satoja tai jopa tuhansia yhtäaikaisia käyttäjiä.

3 Testauksen automatisointi

3.1 Lähtötilanne

Verkkokauppaa kehitetään neljän viikon sykleissä ja siitä toimitetaan uusi versio noin kerran kuussa. Uusia versiota voidaan julkaista myös tiheämmin, jos on tarve korjata kriittisiä virheitä, joita on löydetty tuotantoympäristöstä. Ennen kuin uusi neljän viikon kehityssykli alkaa, asiakas toimittaa vaatimukset uusista toiminnallisuuksista, jotka halutaan saada valmiiksi seuraavaan julkaistavaan versioon.

Kun uusi toiminnallisuus valmistuu, niin sen käy ensin hyväksymistestaamassa testausasiantuntija, jonka jälkeen toiminnallisuus menee asiakkaalle testattavaksi. Uudet toiminnallisuudet pitää aina testata ennen kuin ne voidaan hyväksyä seuraavaan julkaistavaan versioon. Jos toiminnallisuutta ei hyväksytä niin se palaa takaisin kehitykseen.

Kun neljän viikon kehitys loppuu, alkaa regressiotestausvaihe, jonka aikana testataan, etteivät uudet toiminnallisuudet rikkoneet jo aikaisemmin toteutettuja toiminnallisuuksia. Tämä regressiovaihe kestää viikon, ja siihen osallistuu koko kehitystiimi, osa etsien virheitä asiakkaan kanssa ja osa korjaten löytyneitä virheitä. Koska tämä regressiotestausvaihe on aikaa ja resursseja vievää, nousi tarve parantaa tätä regressiotestausprosessia. Testausprosessia lähdettiin parantamaan automatisoimalla regressiotestaus eli sen sijaan että kehitystiimi ja asiakas tuhlaisivat aikaa kaikkien olemassa olevien toiminnallisuuksien testaamiseen, sen tekisi robotti.

3.2 Automatisoinnin suunnittelu

3.2.1 Automatisoinnin laajuus

Aluksi lähdettiin pohtimaan että, mitkä ovat verkkokaupan minimivaatimukset, jotta voidaan todeta, että verkkokauppa toimii. Pohdinnan tuloksena saatiin seuraavanlaiset vaatimukset:

Tilaaminen

Tämä on todennäköisesti verkkokaupan kriittisin toiminnallisuus. Verkkokaupasta pitää pystyä ostamaan tuotteita ja jos tämä toiminnallisuus on rikki niin kauppaa ei synny ja asiakkaalle tulee tappiota menetetyistä tilauksista.

Haku

Verkkokaupasta on todella vaikeaa löytää haluamaansa tuotetta, jos haku on rikki ja tuotteita joutuu etsimään pää-navigoinnin avulla. Jos loppukäyttäjä ei löydä tuotteita helposti, hän todennäköisesti menee kilpailijan kauppaan ja tämä merkitsee menetettyjä tilauksia asiakkaalle.

Kategoriat ja Tuotteet

Verkkokaupassa on tärkeää, että tuotteet näkyvät kategoria-sivuilla ja että tuotteita voidaan lisätä ostoskoriin. Jos tuotteita ei näy tai niitä ei voida lisätä ostokoriin, niin tarkoittaa se taas menetettyjä kauppoja asiakkaalle.

Rekisteröityminen ja sisäänkirjautuminen

Yksi verkkokaupan perusominaisuuksista on se, että sinne voidaan rekisteröityä ja kirjautua sisään.

Nämä edellä mainitut vaatimukset ovat niin sanottuja verkkokaupan perustoiminnallisuuksia ja näiden toiminnallisuuksien on aina toimittava, jotta voidaan minimissään todeta, että verkkokauppa toimii jollain tasolla ja mikään uusi toiminnallisuus ei saa rikkoa mitään näistä toiminnallisuuksista. Kun perustoiminnallisuudet oli kartoitettu, niin sovimme toimeksiantajan kanssa, että tämän opinnäytetyön tarkoituksena olisi automatisoida testaaminen perustoiminnallisuuksille eli toisin sanoen kehittää au-

tomatisoidut testit savutestausta varten. Muiden toiminnallisuuksien testauksen automatisointi olisi sitten osa jatko-kehitystä.

3.2.2 Automatisoinnin työkalut

Kun opinnäytetyössä tehtävän automatisoinnin laajuus oli selvillä, aloitettiin pohtimaan mitä työkaluja käytettäisiin testauksen automatisoinnin tekemiseen. Verkkokauppa on web-sovellus ja web-sovellusten testaamiseen löytyy paljon eri työkaluja.

Opinnäytteessä päädyttiin käyttämään Cucumber-nimistä automaatiotestaustyökalua. Cucumberin ideana on, että testitapaukset kirjoitetaan selkokielisenä tekstinä, jotka sitten muutetaan suoritettaviksi testeiksi. Koska testitapaukset ovat selkokielistä tekstiä, niin niitä on helppo esittää suoraan asiakkaalle.

Cucumber ei itsestään ole web-selaimen automaatiotyökalu, mutta se toimii hyvin web-selainten automaatioon tarkoitettujen työkalujen kanssa. Opinnäytetyössä on käytetty Capybara-nimistä web-selain automaatiotyökalua Cucumberin kanssa. Näiden työkalujen tarkempia ominaisuuksia on avattu seuraavissa alakappaleissa.

Cucumber

Cucumber on kirjoitettu Ruby-ohjelmointikielellä ja se suorittaa Gherkin kielellä kirjoitettuja testitapauksia. Gherkin on jäsennettyä selkokielistä tekstiä, joka on suunniteltu helposti opittavaksi, mutta sen avulla bisnes-säännöt voidaan kuvata selkeästi ja ytimekkäästi. Gherkin kielessä jokainen tekstiä sisältävä rivi alkaa Gherkin avainsanalla, jonka jälkeen teksti voi olla mitä vain (ks. kuvio 2). (Cucumberin viiteasiakirja N.d.)

```
Feature: Refund item

  Scenario: Jeff returns a faulty microwave
    Given Jeff has bought a microwave for $100
    And he has a receipt
    When he returns the microwave
    Then Jeff should be refunded $100
```

Kuvio 2. Gherkin kielellä kirjoitettu esimerkkitesti (alkup. kuvio ks. mt.)

Gherkin avainsanat voidaan kuvata lyhyesti seuraavanlaisesti:

Feature on yleiskuvaus ohjelman toiminnallisuudesta. Sen avulla voidaan ryhmittää toisiinsa liittyvät testitapaukset (Scenario) yhteen (Mt).

Scenario on yksittäinen testitapaus ja se sisältää listan askeleista (Step), joiden avulla testitapaus määritellään ja dokumentoidaan. Askelten avulla määritetään testitapauksen lähtötilanne (**Given**), tapahtumat (**When**) ja odotetut lopputulokset (**Then**). Näiden lisäksi voidaan käyttää **And** ja **But** -askelta, kun skenaariossa halutaan käyttää monta peräkkäistä Given, When tai Then -askelta (ks. liite 1 ja 2).

Jos sama Given -askel toistuu saman featuren sisällä kaikissa skenaariossa, sen voi siirtää omaan **Background** -avainsanan alle ennen ensimmäistä skenaariota (ks. kuvio 3).

```
Background:
  Given a $100 microwave was sold on 2015-11-03
  And today is 2015-11-18
```

Kuvio 3. Esimerkki Background -avainsanan käytöstä (alkup. kuvio ks. mt.)

Cucumber ei suoraan ymmärrä Gherkin kielellä kirjoitettua testitapausta, vaan jokaisen testitapauksen askeleella pitää olla olemassa askelmääritelmä (Step definition). Askelmääritelmät ovat pieniä koodinpätkiä, joiden avulla testiin liittyvät askeleet oikeasti suoritetaan. Eli kun Cucumber suorittaa Gherkin kielellä kirjoitettua testitapausta, se etsii sitä vastaavat askelmääritelmät ja suorittaa niiden sisältämät koodit. (Mt.)

Askelmääritelmät voidaan toteuttaa monella eri ohjelmointikielellä. Opinnäytetyössä Cucumberin askelmäärittelyt toteutettiin käyttämällä Ryby-ohjelmointikieltä, sekä Capybaran ohjelmointirajapintaa.

Capybara

Capybara on web-selaimen automatisointiin tarkoitettu ohjelmointirajapinta ja se on kirjoitettu Ruby-ohjelmointikielellä. Capybara avulla voidaan helposti simuloida käyt-

täjän vuorovaikutusta testattavan sovelluksen kanssa. (Test your app with Capybara N.d.)

Sen avulla voidaan esimerkiksi avata eri web-sivuja, klikata eri elementtejä sivulla, täyttää lomakkeita ja parsia HTML-dokumentteja. Capybara tarjoaa yhden yhteisen ohjelmointirajapinnan web-sovellusten testauksen automatisointiin ja tukee montaa eri web-ajuria. Web-ajurit ovat ohjelmistorajapintoja, jotka käskyttävät selainta. Tässä opinnäytetyössä käytettiin kahta erilaista web-ajuria Capybaran kanssa:

Selenium 2 (Selenium WebDriver) on työkalu web-sovellusten testaamisen automatisointia varten ja sen avulla verifioidaan, että testit toimivat odotustenmukaisesti (Selenium WebDriver N.d).

Selenium 2 -ajuri tukee eri selaimia ja tässä opinnäytetyössä käytettiin Seleniumia Firefoxin ja Chromen kanssa. Kun Capybara suorittaa testiä selenium 2 -ajuria käyttämällä, niin Selenium avaa selaimen ja käskyttää sitä. Tämä oli hyödyllistä silloin kun testejä kehitetiin, sillä selainikkunasta näki suoraan mitä testiaskeleet tekivät web-sovelluksessa.

Poltergeist on ajuri Capybaralle. Poltergeistin avulla testit suoritetaan käyttämällä PhantomJS:n tarjoamaa Headless webkit -selainta. (Poltergeist - A PhantomJS driver for Capybara. N.d.)

Headless -selain on web-selain jolla ei ole graafista käyttöliittymää, mutta se kuitenkin pystyy renderöimään ja ymmärtämään HTML-, CSS- ja JavaScript koodia (Girdwood 2009).

Eli testejä pystytään suorittamaan palvelinympäristöissä, joissa on käytössä vain komentorivi. Tämä on hyödyllistä, sillä testejä tultaisiin suorittamaan Jenkins -jatkuvan integraation palvelimella.

3.3 Työkalujen asentaminen

3.3.1 Työkalujen asentaminen Windows-ympäristöön

Testejä kehitettiin Windows-työympäristössä ja tätä varten työkalut piti asentaa Windows-ympäristöön. Ympäristö oli 64-bittinen, joten asennuspaketeissa suosittiin 64-bittisiä versiota.

Testejä kehitettiin Ruby-ohjelmointikielellä ja myös Cucumber tarvitsi toimiakseen Rubyn, joten aivan ensimmäiseksi asennettiin Ruby sekä Rubyn development kit (devkit). Tämä tapahtui lataamalla Windowsille tarkoitetut asennuspaketit Ruby:n lataus sivuilta http://rubyinstaller.org/downloads. Rubysta asennettiin 2.0 versio. Ruby devkitin asennuspaketti purki sisällön ennalta määrättyyn sijaintiin. Tähän sijaintiin piti navigoida käyttäen Windowsin komentorivi -työkalua. Devkit alustettiin syöttämällä komento ”*ruby dk.rb init”* komentoriville. Komento generoi *config.yml* -tiedoston. Tähän tiedostoon piti lisätä aiemmin asennetun Rubyn sijainti: ”-C:\Installed\Ruby200-x64”. Lopuksi ajettiin asennus komento ”*ruby dk.rb install*”.

Kun Ruby ja Ruby devkit oli asennettu onnistuneesti, voitiin muut työkalut Jenkisiä lukuun ottamatta asentaa Ruby gems -työkalun avulla, syöttämällä seuraavat komento komentoriville: ”*gem install cucumber, capybara, selenium-webdriver, poltergeist, rake”.* Poltergeista varten, piti myös PhantomJS asentaa. Tämä onnistui lataamalla zip-paketti osoitteesta http://phantomjs.org/download.html. Zip-paketin sisältö purettiin kansioon: ”*C:\phantomjs-2.0.0-windows\bin\phantomjs.exe”.*

Jotta testejä olisi selkeämpää lukea, komentoriviä varten asennettiin Ansicon -työkalu, joka muuttaa konsolin output-tekstin värin testien onnistuessa vihreäksi ja epäonnistuessa punaiseksi. Työkalu asennettiin lataamalla zip-paketti osoitteesta https://github.com/adoxa/ansicon/downloads. Komentorivillä navigoitiin puretun zip-paketin sijaintiin ja suoritettiin seuraava komento: *”ansicon.exe -i”.* Lopuksi komentorivi piti käynnistää uudelleen. Jenkins asennettiin lataamalla ajettava asennuspaketti osoitteesta https://jenkins.io/index.html.

3.3.2 Työkalujen asentaminen Linux-ympäristöön

Työkalut piti asentaa myös Linux-ympäristöön, sillä tuotanto ympäristössä jatkuvan integraation palvelin Jenkins on asennettu Linuxin centos -käyttöjärjestelmän päälle. Asennus haluttiin tehdä hallitusti, joten ensin testattiin asennuskomennot virtuaalikoneessa, johon oli asennettu centos -käyttöjärjestelmä. Centosilla ohjelmia asennettiin yum install -komennolla.

Ruby asennettiin komennolla: *”sudo yum install ruby”.* Ruby tarvitsi myös seuraavat työkalut toimiakseen: *“sudo yum install gcc g++ make automake autoconf curl-devel openssl-devel zlib-devel httpd-devel apr-devel apr-util-devel sqlite-devel ruby-rdoc ruby-devel”.* Ruby gems-työkalu asennettiin komennolla: *”sudo yum install rubygems”.*

Kun Ruby ja Ryby gems -työkalu oli asennettu, muut työkalut voitiin asentaa syöttämällä seuraava komento komentoriville: *”gem install cucumber, capybara, selenium-webdriver, poltergeist, rake”.* Jenkins asennettiin seuraavilla komennoilla: *”sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo”, “sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key”, ”sudo yum install jenkins”.*

PhantomJS asennettiin samalla tavalla kuin Windowsissa eli lataamalla purettavan paketin osoitteesta http://phantomjs.org/download.html. Paketin sisältö piti purkaa kansioon: */usr/bin/phantomjs.sh*

3.4 Testien kehittäminen

Testejä kehitettiin Windows-työympäristössä ja editorina käytettiin Sublime textiä (https://www.sublimetext.com/). Sublime textiä varten asennettiin myös liitännäinen (https://github.com/drewda/cucumber-sublime-bundle), jotta se tunnistaisi Gherkin syntaksin ja korostaisi sitä oikein (ks. kuvio 4). Opinnäytetyön kohdeprojektissa oli käytössä Mercurial -versionhallintaohjelma ja testejä varten luotiin oma sub-repository versiohallintaan. Testejä kehitettiin yhdessä verkkokaupan testausasiantuntijan kanssa, joten versiohallinta oli hyvä olla käytössä. Näin pystyttiin seuraamaan helposti mitä muutoksia testeihin tehtiin.

```
Feature: User account
  As a user I can login, logout, create account and change my contact information and password.

  Background:
    Given I am on homepage

  @global
  Scenario: Login with correct credentials
    Given I use user "Tester" credentials
    When I log in
    Then I should see my contact information

  @global
  Scenario: Login with wrong credentials
    Given I use user "Tester_wrong_password" credentials
    When I log in
    Then I should see an error message

  @global
  Scenario: Logout
    Given I am a logged in user
    When I logout
    Then I should see login page

```

Kuvio 4. Käyttäjään liittyvät testit

Cucumber -testien kehittämisprosessin kuvaamista varten esimerkkinä käytetään osaa käyttäjiin liittyvistä testitapauksista (ks. kuvio 4). Testien kehittäminen alkoi valitsemalla testattava toiminnallisuus ja siihen liittyvien testitapauksien suunnittelusta. Kun testitapaukset oli suunniteltu ja kirjoitettu feature-tiedostoon, niin piti niitä vastaavat itse testejä suorittavat askelmääritelmät toteuttaa. Askelmääritelmät toteutettiin käyttämällä Ruby-ohjelmointikieltä, sekä Capybaran tarjoamaa valmista ohjelmointirajapintaa, joka sisältää kattavan määrän web-sovelluksen testaamista varten käytettäviä ominaisuuksia.

Cucumber käyttää regex -syntaksia linkittääkseen testiaskeleen, sitä vastaavaan askelmääritelmään. Eli kun Cucumber suorittaa skenaariota, jossa sisäänkirjautuminen onnistuu (ks. kuvio 5), niin se oikeasti suorittaa skenaarion liittyviä askelmääritelmiä (ks. kuvio 6).

```
  @global
  Scenario: Login with correct credentials
    Given I use user "Tester" credentials
    When I log in
    Then I should see my contact information
```

Kuvio 5. Skenaario: onnistunut sisään kirjautuminen

Sisäänkirjautumistestitapauksen (ks. kuvio 5) testi askeleet voidaan avata seuraavanlaisesti: Testin lähtötilanteessa ollaan verkkokaupan etusivulla, tämä on määritetty background-avainsanalla (ks. Kuvio 4). Tämä tapahtuu siis ennen kaikkia testitapauksia, jotka ovat samassa feature-tiedostossa.

Skenaarion alussa Cucumber lähtee suorittamaan ’Given I use user ”Tester” credentials’ -askelta, joka lähettää tekstin ”Tester” askelmääritelmälle. Askelmääritelmissä on regexin avulla tämä teksti parsittu user-muuttujaan (ks. Kuvio 6).

```
69  Given(/^I use user "(.*?)" credentials$/) do |username|
70    @user=@users[username]
71  end
72
73  Given(/^I log in$/) do
74    login(@user.email, @user.password)
75  end
76
77  Given(/^I should see my contact information$/) do
78    page.assert_selector(:xpath,'.//p[@class="myaccount_desc"]')
79
80  end
```

Kuvio 6. Askelmääritelmät (Step definitions)

Testin seuraavassa vaiheessa Cucumber suorittaa ’When I log in’ -askelta, joka askelmääritelmissä kutsuu *login* -funktiota (ks. Kuvio 6). *Login* -funktio kutsuu puolestaan *open_shopping_context_page* -funktiota, joka klikkaa etusivulta löytyvää kirjaudu nappia. Napin klikkaaminen avaa pop-up ikkunan selaimessa, johon Capybara täyttää käyttäjän sähköpostin ja salasanan, sekä klikkaa submit nappia (ks. Kuvio 7).

```
36
37  def login(username, password)
38    open_shopping_context_page()
39    fill_in('logonId', :with => username)
40    fill_in('logonPassword', :with => password)
41      page.find(:xpath,'.//button[@type="submit"]').click
42  end
43
```

Kuvio 7. Login -funktio

Testin viimeisessä vaiheessa varmistetaan, että sisäänkirjautuminen onnistui. Sisäänkirjautumisen onnistuessa, käyttäjä ohjataan hänen omat tiedot -sivulle. Viimeisessä

askelmääritelmässä tarkistetaan, että sivulta löytyy HTML elementti , jonka CSS luokkana on ”myaccount_desc” eli varmistetaan, että päädyttiin oikealle sivulle (ks. kuvio 6).

Testien kehittämisvaiheessa Capybaran kanssa käytettiin Selenium 2 ajuria, koska selenium avasi selaimen oikeasti ja selainikkunasta nähtiin mitä testitapauksessa tapahtuu. Esimerkiksi jos sisäänkirjautumistesti olisi epäonnistunut väärän salasanan takia, selain olisi mennyt sisäänkirjautumissivulle virheviestin kera.

3.4.1 Cucumber-komento

Testejä ajettiin komentoriviltä kutsumalla Cucumber-komentoa ja antamalla sille tarvittavat parametrit. Jos esimerkiksi haluttiin suorittaa edellisessä kappaleessa mainittua käyttäjiin liittyviä testitapauksia (ks. kuvio 4), niin sitä kutsuttiin seuraavalla komennolla:

cucumber features\user_account.feature --tags @fi,@global --tags ~@ignore --format pretty -f json -o Reports/result.json COUNTRY=FI driver=selenium

Komennon parametrit on erotettu seuraavanlaisesti:

features\user_account.feature määrittää ajettavan feature -tiedoston.

--tags -parametri kertoo, mitkä skenaariot suoritetaan, tässä tapauksessa suoritetaan skenaariot, joilla on *@fi* tai *@global* tagi. Testauksen automatisoinnin kohteena olevalla verkkokaupalla on omat sivustot viidelle eri kielellä eli tässä tapauksessa @global -tagilla tarkoitetaan sitä, että testi voidaan ajaa jokaisella eri kielisellä sivustolla. Skenaariot, jotka ovat sivustokohtaisia merkitään sivuston kieli -tagilla, esimerkiksi vain suomenkieliselle sivustolle tarkoitetut skenaariot on merkattu @fi -tagilla.

--tags ~@ignore kertoo, että kaikki skenaariot, jossa on @ignore tagi jätetään suorittamatta.

--format pretty -f json -o Reports/result.json kertoo komennolle missä muodossa tulokset tulostetaan, mitä muotoa käytetään, sekä minne tulokset tallennetaan. Eli tässä tapauksessa käytetään *pretty*-formaattia ja tulostetaan tulokset *json* muodossa *Reports/result.json* tiedostoon.

Country -parametrin avulla kerrotaan Cucumberille, että ajetaan testejä Suomen sivustoa vasten.

Driver -parametri kertoo Capybaralle sen, mitä ajuria käytetään.

Kun aiemmin mainittu Cucumber -komento ajetaan komentorivillä, niin Cucumber tulostaa Gherkin kielellä kirjoitetut askeleet konsoliin. Askeleen onnistuessa väri on vihreä ja epäonnistuessa punainen. Askeleen epäonnistuessa myös mahdollinen virheviesti tulostetaan (ks. kuvio 8).

Askeleen mennessä virheeseen suoritettava komento keskeytettiin manuaalisesti ja lähdettiin korjaamaan testissä tapahtunutta virhettä. Konsoliin tulostuvasta virheviestistä näki yleensä suoraan, että mikä askeleessa on vikana. Esimerkiksi kuviossa 8 esiintyvä virheviesti viittaisi siihen, että askelmäärittelyssä (ks. kuvio 6) käytettävä instanssitaulukko @users ei olisi alustettu ennen kuin sitä kutsutaan.

```
C:\IBM\WCDE_ENT70\workspace\WebTests\TestAutomation>cucumber features\user_accou
nt.feature --tags ~@ignore --format pretty -f json -o Reports/result.json COUNTR
Y=FI driver=selenium
Feature: User account
  As a user I can login, logout, create account and change my contact informatio
n and password.

  Background:              # features/user_account.feature:4
    Given I am on homepage # features/step_definitions/test_steps.rb:1

  @global
  Scenario: Login with correct credentials   # features/user_account.feature:8
    Given I use user "Tester" credentials    # features/step_definitions/user_st
eps.rb:72
      undefined method `[]' for nil:NilClass (NoMethodError)
      ./features/step_definitions/user_steps.rb:73:in `/^I use user "(.*?)" cred
entials$/'
      features/user_account.feature:9:in `Given I use user "Tester" credentials'
```

Kuvio 8. Cucumber komennon suorittaminen

3.4.2 Rake

Rake on Make tyylinen ohjelma, joka on implementoitu Rubylla. Raken avulla voidaan luoda rake-tiedostoja, jonne voidaan määrittää komentoja tehtäviksi. Rake tiedostot kirjoitetaan normaalilla Rubyn syntaksilla. (RAKE – Ruby Make N.d.)

Tässä opinnäytetyössä Rakea käytettiin muuttamaan pitkät Cucumber-komennot helposti kutsuttaviksi tehtäviksi (ks. kuvio 9).

```
require 'rubygems'
require 'cucumber'
require 'cucumber/rake/task'

task :default => [:features]

Cucumber::Rake::Task.new(:features) do |t|
  t.cucumber_opts = "features --expand --format pretty -f json -o Reports/results.json driver=selenium HEADLESS=true"
end

Cucumber::Rake::Task.new(:Poltergeist_linux) do |t|
  t.cucumber_opts = "features --expand --format pretty -f json -o Reports/results.json driver=poltergeist_linux COUNTRY=FI"
end
```

Kuvio 9. Rake-tiedosto

Rakea käytettiin menemällä rake-tiedoston sijaintipaikkaan komentorivillä ja kutsumalla rake-komentoa ja antamalla suoritettavan tehtävän nimi:

rake Poltergeist_linux

Komento suorittaa siis oikeasti pitkän Cucumber-komennon, joka on määritelty rake -tiedostoon (ks. kuvio 9).

3.5 Jenkins

Jenkins on avoimen lähdekoodin automaatiopalvelin, jonka avulla voidaan kääntää, testata ja julkaista ohjelmistoja (Meet Jenkins 2016).

Opinnäytetyön kohteena olevassa verkkokauppaprojektissa Jenkinssiä käytetään uusien versioiden kääntämiseen ja julkaisemiseen. Kun käännöstä lähdetään rakentamaan, Jenkins hakee uuden koodipohjan versionhallinnasta. Käännöksiä julkaistaan testi- ja tuotantoympäristöön.

Tämän opinnäytetyön yhtenä tarkoituksena oli saada Jenkins suorittamaan automatisoituja testejä osana julkaisuprosessia. Eli kun uusi versio verkkokaupasta julkaistaan, Jenkins suorittaisi Cucumber-testeistä luodun savutestaus -setin heti uutta versiota vasten. Tämän avulla huomattaisiin suoraan, jos uusi koodipohja on rikkonut jotain perustoiminnallisuuksista. Myöhemmin kun testien määrä kasvaisi, tarkoituksena olisi määrittää Jenkins suorittamaan kaikki testit kerran päivässä.

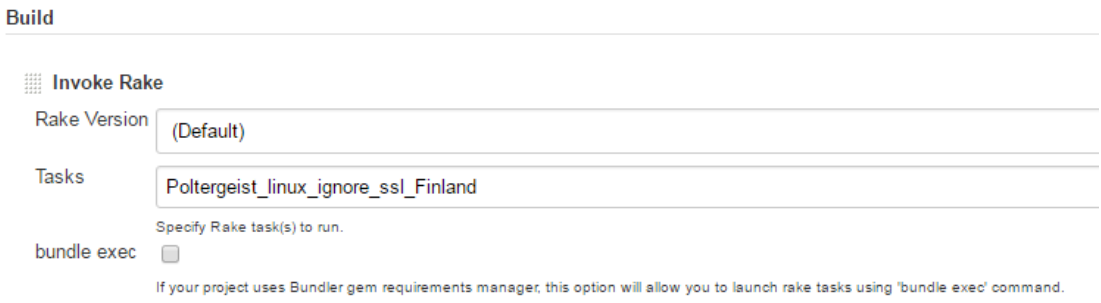
Jokaiselle verkkokaupan maakohtaiselle sivustolle luotiin Jenkinsiin oma käännöstyönsä. Jenkins haki myös testejä varten testien uusimmat versiot versionhallinnasta aina kun testejä aloitettiin suorittamaan.

Jenkinsiä on helppo laajentaa erilaisten liitännäisten avulla. Opinnäytetyötä varten Jenkinsiin asennettiin 3 liitännäistä.

Rake liitännäinen

Cucumber-komentoja oli helppo suorittaa Raken avulla. Jenkinsiin löytyykin liitännäinen Rakelle: https://wiki.jenkins-ci.org/display/JENKINS/Rake+plugin

Rake liitännäisen avulla voitiin määrittää minkä tehtävän mikäkin käännös työ suorittaa. Esimerkiksi Suomen sivustoa vasten suoritettavia testejä varten luotiin rake-tiedostoon tehtävä nimellä *Poltergeist_linux_ignore_ssl_Finland*. Tehtävä sisälsi Cucumber komennon, joka tagien avulla määrittää, että vain @fi ja @global tagilla merkityt skenaariot suoritetaan. Rake liitännäisen avulla määritettiin, että Suomen Jenkins työ ajaa *Poltergeist_linux_ignore_ssl_Finland* -nimisen tehtävän (ks. kuvio 10).



Kuvio 10. Rake liitännäisen konfiguraatio

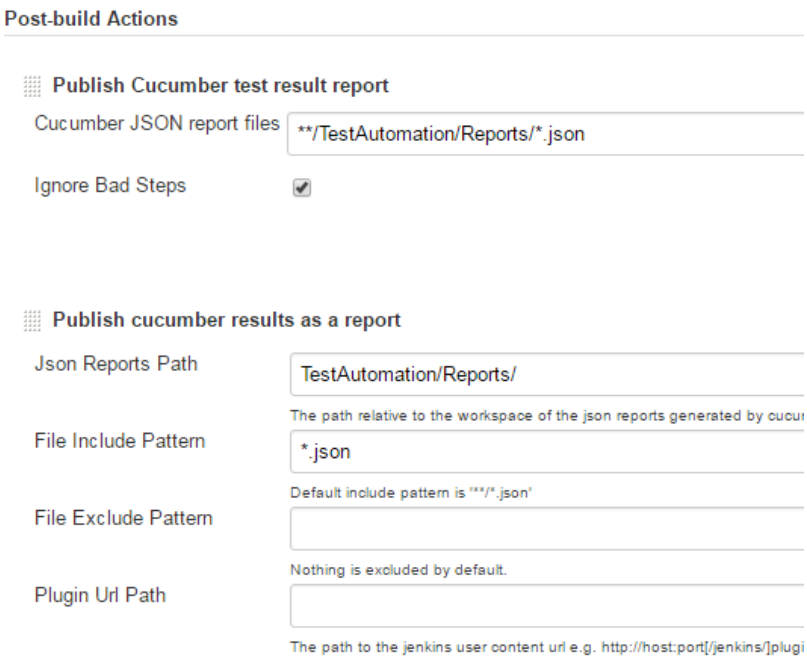
Raportti liitännäiset

Testien tulosten tarkastelua varten Jenkinsiin asennetiin kaksi raporttiliitännäistä:

https://wiki.jenkins-ci.org/display/JENKINS/Cucumber+Reports+Plugin

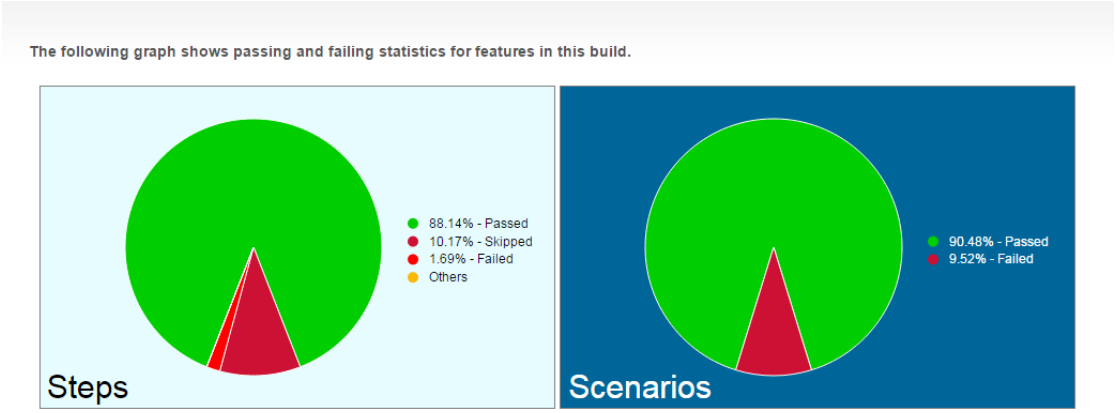
https://wiki.jenkins-ci.org/display/JENKINS/Cucumber+Test+Result+Plugin

Kummatkin liitännäiset osasivat parsia Cucumberin luoman json tiedoston ja tiedoston sijainti piti konfiguroida molemmille liitännäiselle (ks. kuvio 11).



Kuvio 11. Raportti liitännäisten konfigurointi

Cucumber Reports -liitännäinen loi tuloksista omat HTML-sivut, jossa se näytti piirakkakaaviossa onnistuneet askeleet ja skenaariot (ks. kuvio 12).



Kuvio 12. Onnistunet askeleet ja skenaariot

Feature Statistics

	Scenarios			Steps							
Feature	Total	Passed	Failed	Total	Passed	Failed	Skipped	Pending	Undefined	Missing	Duration	Status
Favorite and recent stores	5	5	0	28	28	0	0	0	0	0	2m 26s 309ms	passed
Products	3	3	0	14	14	0	0	0	0	0	29s 067ms	passed
Purchase process	2	0	2	22	8	2	12	0	0	0	1m 20s 887ms	failed
Search	2	2	0	14	14	0	0	0	0	0	14s 057ms	passed
Store selection	1	1	0	4	4	0	0	0	0	0	10s 806ms	passed
User account	8	8	0	36	36	0	0	0	0	0	47s 779ms	passed
6	21	19	2	118	104	2	12	0	0	0	5m 28s 906ms	Totals

Kuvio 13. Feature statistiikka

Lisäksi liitännäinen avaa kaikkien suoritettujen feature -tiedoston statistiikkaa taulukkomuotoon, josta nähdään tarkemmin, mitkä featureista ovat onnistuneet ja missä on vielä puutteita (ks. kuvio 13). Taulukosta voi klikata tietyn featuren statistiikan omaksi taulukokseen, josta nähdään tiedot askel kohtaisesti (ks. kuvio 14).

Cucumber Reports

Jenkins Previous build Last build Tags Feature Steps

Result for *User account* in build: 248

	Scenarios			Steps							
Feature	Total	Passed	Failed	Total	Passed	Failed	Skipped	Pending	Undefined	Missing	Duration	Status
User account	8	8	0	36	36	0	0	0	0	0	47s 779ms	passed

Feature: User account

As a user I can login, logout, create account and change my contact information and password.

Background:
Given I am on homepage 02s 089ms

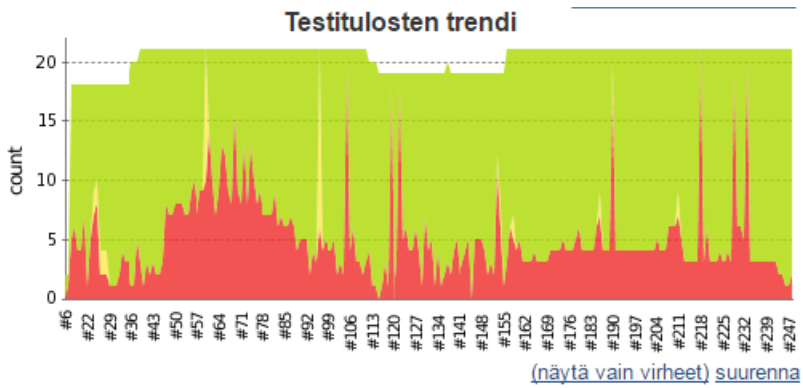
@global
Scenario: Login with correct credentials
Given I use user "Tester" credentials 000ms
When I log in 06s 188ms
Then I should see my contact information 137ms

@global
Scenario: Login with wrong credentials
Given I use user "Tester_wrong_password" credentials 000ms
When I log in 03s 576ms
Then I should see an error message 003ms

@global
Scenario: Logout
Given I am a logged in user 06s 466ms
When I logout 05s 583ms
Then I should see login page 004ms

Kuvio 14. User account featuren statistiikka

Cucumber Test Result liitännäinen osaa myös tulostaa testien statistiikkaa, mutta liitännäistä käytettiin lähinnä testitulosten trendin seuraamiseen (ks. kuvio 15).



Kuvio 15. Testitulosten trendi

3.6 Testauksen tulokset

Testejä voitiin siis ajaa Jenkinsillä, mutta niin kuin kuviosta 13 näkyy, eivät kaikki testit vielä mene läpi ja skenaariota on vielä aika vähän. Esimerkiksi ostoprosessiin liittyvät testit ovat vielä aika kesken, sillä siihen on tehty vasta kaksi skenaariota, jotka kummatkin ovat epäonnistuneet. Testitapauksia saatiin kuitenkin tehtyä jokaiselle suunnitetuille päätoiminnallisuuksille vaikkakin vain muutama per toiminallisuus. Yhteensä testitapauksia on vasta 22 ja jotta regressiotestauksen manuaalista tekemistä voitaisiin vähentää, niin pitäisi testitapauksia luoda vielä lisää.

4 Pohdinta

Opinnäytetyön tavoitteina oli selvittää mitkä ovat verkkokaupan päätoiminnallisuuksia, näiden toiminnallisuuksien testaamisen automatisointi, sekä näiden automatisoitujen testitapauksien suorittamista osana kehitysprosessia. Tuloksena saatiin automatisoituja testejä, joita voidaan ajaa jatkuvan integraation palvelimella. Suurin osa testitapauksista on vielä kehitysvaiheessa tai eivät toimi verkkokaupan uudistuneen ulkoasun takia. Tilanteen näkee hyvin testitulosten trendistä (ks. kuvio 15). Tästä johtuen regressiotestausta tehdään vielä manuaalisesti.

Tuloksena saatiin siis enemmänkin ”proof of concept” kuin valmis kattava testisetti, jota voitaisiin suorittaa osana kehitysprosessia. Osa syystä oli siinä, että opinnäytetyötä tehtiin normaalin työn ohessa, eikä sille allokoitu tarpeeksi aikaa. Tilanne on nyt kuitenkin hyvä, sillä Gherikin syntaksiset testiaskeleet ovat valmiina ja vain askelmääritelmät pitää korjata. Asiakaskin on priorisoinut testauksen automaatiota suuremmaksi ja tarkoituksena olisi saada suunniteltu savutestaus-setti automatisoiduksi kevään 2016 aikana.

Kunhan savutestaus-setti eli päätoiminnallisuuksien testaaminen on saatu kokonaan automatisoitua, niin testauksen automatisointia tullaan varmasti jatkamaan automatisoimalla muut vähemmän kriittiset toiminnallisuudet.

Opinnäytetyötä tehdessä opin paljon testaamisesta ja testauksen automaatiosta. Yksi yllättävä asia, minkä opin on se miten paljon aikaa yksinkertaisen testitapauksenkin automatisointi vie. Opinnäytetyössä käytetyt työkalut olivat Jenkinsiä lukuun ottamatta uusia tuttavuuksia, ja näiden työkalujen oppiminen on varmasti hyödyksi tulevissa työtehtävissä.

Lähteet

Cucumberin viiteasiakirja. N.d. Viiteasiakirja Cucumberin eri ominaisuuksista. Viitattu 16.04.2016. https://cucumber.io/docs/reference

FiSTB. 2012. ISTQB:n testaussanasto v. 2.3 Suomi – Englanti. Viitattu 17.04.2016. http://www.fistb.fi/sites/fistb.ttlry.mearra.com/files/istqb_sanasto_2015-04-30%202.3%20FI-ENG.pdf

Girdwood, A. 2009. What is a headless browser? Viitattu 16.04.2016 http://blog.arhg.net/2009/10/what-is-headless-browser.html

Given When Then. 2014. Cucumber wiki. Viitattu 17.04.2016. https://github.com/cucumber/cucumber/wiki/Given-When-Then#and-but

Haikala, I. & Märijärvi, J. 2006. Ohjelmistotuotanto. Helsinki: Talentum.

ISTQB. 2010. Standard glossary of terms used in Software Testing. Toim. E. Van Veenendaal. Viitattu 19.03.2016. http://www.estb.org.eg/Documents/ISTQB_Glossary_Terms.pdf

ISTQB. 2016a. ISTQB Certification exam study material. Viitattu 19.03.2016. http://istqbexamcertification.com/what-is-acceptance-testing/

ISTQB. 2016b. ISTQB Certification exam study material. Viitattu 19.03.2016. http://istqbexamcertification.com/what-is-regression-testing-in-software/

Kaner, C., Bach, J. & Pettichord, B. 2002. Lessons Learned in Software Testing. Wiley Computer Publishing.

Kaner, C. 2002. Avoiding Shelfware: A Managers' View of Automated GUI Testing. Viitattu 23.04.2016. http://www.kaner.com/pdfs/shelfwar.pdf

Kaner, C. 2006. Exploratory Testing. Viitattu 28.02.2016. http://www.kaner.com/pdfs/ETatQAI.pdf

Meet Jenkins. 2016. Viitattu 17.04.2016. https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

Pan, J. 1999. Software Testing. Viitattu 28.02.2016. https://users.ece.cmu.edu/~koopman/des_s99/sw_testing/

Poltergeist - A PhantomJS driver for Capybara. N.d. Github sivu Poltergeist ohjelmalle. Viitattu 16.04.2016. https://github.com/teampoltergeist/poltergeist

RAKE – Ruby Make. N.d. Raken Github sivu. Viitattu 17.04.2016. https://github.com/ruby/rake

Selenium WebDriver. N.d. Viitattu 16.04.2016. http://docs.seleniumhq.org/docs/03_webdriver.jsp

Solteq lyhyesti. N.d. Artikkeli Solteq Oyj:n sivuilla. Viitattu 14.4.2016. https://www.solteq.com/fi/sijoittajat/solteq-sijoituskohteena/solteq-lyhyesti/

Test your app with Capybara. N.d. Capybaran sivut. Viitattu 16.04.2016. http://jnicklas.github.io/capybara/

Zambelich, K. N.d. Totally Data-Driven Automated Testing. A White Paper. Viitattu 23.04.2016. 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

Liitteet

Liite 1. Monta Given-, When-, ja Then-sanaa skenaariossa (alkup. Kuvio ks. Given When Then, 2014).

```
Scenario: Multiple Givens

    Given one thing

    Given another thing

    Given yet another thing

    When I open my eyes

    Then I see something

    Then I don't see something else
```

Liite 2. And- ja But-sanojen käyttö skenaariossa (alkup. Kuvio ks. Given When Then, 2014).

```
Scenario: Multiple Givens

    Given one thing

      And another thing

      And yet another thing

    When I open my eyes

    Then I see something

      But I don't see something else
```