HAMMASHOIDON INTERNET-AJANVARAUKSEN AUTOMAATIOTESTAUKSEN SUUNNITTELU JA TOTEUTUS ROBOT FRAMEWORKILLA JA SELENIUMILLA

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus, Hämeenlinnan korkeakoulukeskus
syksy, 2020

Mirka Pihlaja

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Hämeenlinnan korkeakoulukeskus

Tekijä	Mirka Pihlaja	Vuosi 2020
Työn nimi	Hammashoidon internet-ajanvarauksen automaatiotestauksen suunnittelu ja toteutus Robot Frameworkilla ja Seleniumilla	
Ohjaajat	Mirlinda Kosova-Alija	

TIIVISTELMÄ

Testausprosessien sujuvuus on erittäin tärkeä osa tuotekehitysprosessissa. Manuaalinen testaus vie aikaa ja tuo kustannuksia. Testien automatisointi taas vapauttaa aikaa muuhun työhön ja tuotekehitykseen. Testit, joita toistetaan useampaan kertaan, ovat monesti automatisoitavissa. Tämän opinnäytetyön tavoitteena oli luoda toimeksiantajalle lähtötaso internet-ajanvarauksen-webkäyttöliittymän testien automatisointiin Robot Frameworkin ja Seleniumin avulla. Toimeksiantajan nimeä ei mainita tässä opinnäytetyössä, vaan toimeksiantajaa kutsutaan nimellä Yritys X. Aihe valikoitui omasta mielenkiinnosta testiautomaatioon ja Robot Frameworkiin.

Opinnäytetyön teoriaosassa käsiteltiin muun muassa testausprosesseja, jossa suurimmassa roolissa oli automaatiotestaus sekä sen työkalut Robot Framework ja Selenium. Näitä työkaluja myös käytettiin testitapauksien automatisoinnissa. Opinnäytetyön aineisto hankittiin pääosin eri internet-lähteistä, artikkeleista ja kirjallisuudesta.

Opinnäytetyö on toiminnallinen ja prosessin aikana luotiin automatisoidut testitapaukset manuaalitestien perusteella. Käytännönosuudessa käydään läpi eteen tulleita ongelmia ja annetaan esimerkki automatisoidusta testitapauksesta. Yritys X saa käyttöönsä kaikki automatisoidut testit ja niiden avulla yritys voi lähteä kehittämään internet-ajanvarauksen testausautomaatiota eteenpäin.

Avainsanat Ohjelmistotestaus, Robot Framework, Selenium, Testausautomaatio

Sivut 31 sivua, ei liitteitä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology **Abstract**

Hämeenlinna University Centre

Author Mirka Pihlaja Year 2020

Subject Implementation of automation testing of dental internet appointment booking with Robot Framework and Selenium

Supervisors Mirlinda Kosova-Alija

ABSTRACT

The smoothness of testing processes is a very important part of the product development process. Manual testing takes time and brings cost. Test automation, on the other hand, saves time for other work and product development. The tests that are repeated multiple times can be often be automated. The aim of this thesis was to create a starting level for the client to automate Internet appointment-web user interface tests using Robot Framework and Selenium. The name of the client is not mentioned in this thesis, so the client is called Company X. The topic was selected from my own interest in test automation and Robot Framework.

In the theoretical part of the thesis was examined e.g. testing processes and automation testing. The biggest role in automation testing was played by the tools Robot Framework and Selenium. These tools were also used to automate the test cases. The material of the thesis was obtained mainly from various internet sources, articles, and literature.

The thesis is functional and during the process automated test cases were created based on the existing manual tests. In the practical part, the problems encountered are reviewed and an example of an automated test cases is given. Company X will have access to all automated tests and will allow the company to move forward with the development of internet appointment testing automation.

Keywords Software testing, Robot Framework, Selenium, Testing automation.

Pages 31 pages, no appendix

Sisällys

Kuvat, ohjelmakoodit ja taulukot

1 Johdanto

Nopeasti kehittyvässä maailmassa tuotekehityksen prosessien optimointi on ensiarvoisen tärkeää. Tässä testauksen sujuvuus on keskeisessä roolissa. Testien automatisointi vapauttaa aikaa muuhun työhön ja tuotekehitykseen. Säästöt tulevat pitkälti luotettavasti ja tarkasti tehdyistä testeistä, jos verrataan manuaalitestaamiseen, joissa inhimillisten virheiden määrä voi nostattaa kuluja ja testauskertoja. Jotta optimaalinen testaustulos saavutettaisiin, voidaan kuitenkin käyttää manuaalitestausta automaatiotestauksen ohella testeissä, joissa automaatiotestien tekeminen on hankalaa.

Tässä opinnäytetyön käytännön osassa asiakkaalle suunnitellaan ja toteutetaan lähtötaso automatisoituihin testitapauksiin, joka koskee selainpohjaista internet-ajanvarausta. Testeissä hyödynnetään olemassa olevia testitapauksia ja työkaluna testien tekemiseen käytetään Robot Frameworkia ja Selenium-kirjastoa. Lisäksi opinnäytetyössä pohditaan ongelmia, jotka ovat ilmenneet tätä opinnäytetyötä tehdessä ja etsitään mahdollinen ratkaisu ongelmien selvittämiseksi.

Teoriaosassa käsitellään muutamia testaukseen liittyviä prosesseja, testaustasoja ja testausmenetelmiä, joita voidaan tehdä ennen ohjelmiston julkaisua. Tarkastellaan automaatiotestauksen työkaluja Robot Frameworkia ja Seleniumia, joilla automaatiotestit suoritetaan käytännön osuudessa. Lisäksi testausautomaation alla käsitellään Gherkin Styleä, joka on erityinen tapa kirjoittaa testitapauksia. Versiohallintaa koskevassa kappaleessa käsitellään versiohallinnan toimintaa. Viimeisessä teoriakappaleessa käsitellään potilastietojärjestelmän käsitettä ja otetaan katsaus hammashoidon potilastietojärjestelmän ominaisuuksiin ja tietosuojakäytäntöön.

Opinnäytetyön keskeiset tutkimuskysymykset ovat:

- Mitä on regressiotestaus ja millaista testaamista voidaan tehdä ennen ohjelman julkistamista?
- Mitä on testausautomaatio?
- Miten Robot Framework ja Selenium toimivat?

2 Toimeksiantaja ja testattava tuote

Opinnäytetyön toimeksiantajana on Suomessa toimiva ohjelmistotalo, joka tuottaa yksityisen puolen hammashoidon potilastietojärjestelmää. Yrityksessä työskentelee noin 40 henkilöä Uudenmaan ja Pirkanmaan alueella. Toimeksiantajan nimeä ei mainita tässä opinnäytetyössä, vaan kutsumme toimeksiantajaa nimellä Yritys X.

Testattava webkäyttöliittymä on hammashoidon tarpeisiin tuotettu ajanvarausjärjestelmä, jolla voidaan varata hammashoidon aikoja. Ajanvarauksen webkäyttöliittymää käyttävät hammashoidon palveluntuottajien asiakkaat. Hammashoidon palveluntuottajat ovat puolestaan Yritys X:n asiakkaita ja käyttävät Yritys X:n tuottamaa potilastietojärjestelmää. Ajanvarauksessa valitaan ja täytetään seuraavat kohdat: toimipiste, tuotettava palvelu, hoitava henkilö ja aika. Näiden lisäksi syötetään varattavan omat tiedot ja mahdolliset lisätiedot. Varauksen jälkeen asiakas saa vahvistuksen varatusta ajasta varaussivulle sekä puhelimeen.

Yritys X:n päätuote on hammashoitoon tarkoitettu potilastietojärjestelmä, joka on pilvipohjainen SaaS-palvelu. Kevään 2020 aikana on yrityksessä lähdetty suunnittelemaan tämän osalta automaatiotestausta, ja työkaluksi on valikoitunut Robot Framework. Opinnäytetyössä käytetään myös testiautomaation työkaluna Robot Frameworkia ja tämän lisänä Selenium-kirjastoa, joka soveltuu web-pohjaiseen testaamiseen.

3 Testaus ja testausautomaatio

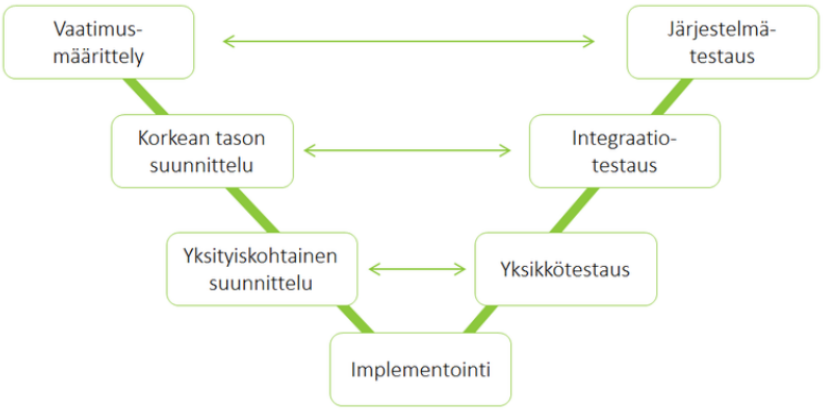
Ohjelmistotestauksen tarkoituksena on varmistaa, että ohjelmiston koodi on toimivaa, samalla löytää sieltä mahdolliset virheet ja ongelmakohdat. Koodin tuottaminen tapahtuu suurelta osin vielä ihmisen toimesta, joten inhimillisiä virheitä tulee jatkossakin. Lisäksi testaaminen parantaa ohjelmiston laatua ja samalla saadaan tarkistetuksi, ovatko ominaisuudet sellaisia kuin on suunniteltu niiden olevan ja toimivan. (Gunawan, 2019; Salminen 2020; Kasurinen, 2013, s. 10)

Testausautomaation potentiaali on se, että sitä voidaan käyttää erilaisissa ohjelmistoprojekteissa. Joskus kuitenkin manuaalistestaus on oikea lähestymistapa testaamiseen, kun halutaan testata sellaisia asioita, jotka voivat kehitysprosessissa muuttua paljon, tai jos ominaisuudet ovat liian mutkikkaita ja joiden automaation tekemiseen menee enemmän aikaa, kuin siitä olisi hyötyä. Vaikka testausautomaatio on työkalu, joka helpottaa ja nopeuttaa työtä, on monia asioita silti hyvä testata manuaalisesti. (Vala Group, n.d.)

3.1 Testauksen tasot

Testauksessa sovelletaan ohjelmistokehitykseen kuuluvaa V-mallia (Kuva 1). Se määrittää testaamisen päävaiheet kehitysvaiheessa. Malli on nelitasoinen ja siihen kuuluvat yksikkötestaus, integraatiotestaus, validaatiotestaus ja järjestelmätestaus. Testausta voidaan tehdään eri tasoilla. (Smart education, n.d.; Kasurinen, 2013, s. 40)

Kuva 1 Testauksen V-malli (Smart education n.d.)



V-malli

Yksikkötestaus on yleisin ja tavallisin testausmuoto, kun halutaan testata yksittäistä pientä ohjelmiston osan toimintaa. Hyödyt testaustavasta nähdään heti kehitysprosessin aikana. Kun ohjelmoija tekee koodiin muutoksen, voi hän samalla ajaa yksikkötestin, jolloin nähdään heti mahdolliset muutoksen tuomat virheet. Tämän myötä muutoksien korjaaminen onnistuu nopeasti ohjelmoijalta. (Smart education, n.d.; Kasurinen, 2013, s. 40)

Integraatiotestauksen avulla testataan järjestelmän yhtenäistä kokonaisuutta, kuinka sovitetut komponentit toimivat yhdessä. Testaustavalla yritetään löytää virheet, joita ei löydetty vielä yksikkötesteissä. Komponentit voivat ohjelmiston toteuttamisessa muuttua ja kasvaa uusien ominaisuuksien myötä, jolloin on hyvä suorittaa regressiotestauksia. Regressiotestauksen avulla voidaan ajaa läpi aikaisemmin suoritettuja testejä, regressiotestaus on uudelleen testaamista. Tämän avulla saadaan tietoon missä komponentin muutoksessa voi olla mahdollinen virhe tai toimimattomuus. Regressiotestejä pystytään ajamaan manuaalisesti tai automaattisesti. Integraatiotestauksen yksi tarkastelumalleista on savutestaus, joka tarkoittaa testattavuutta yksinkertaisilla testaustoimenpiteillä. (Smart education, n.d.)

Validaatiotestaus tehdään ennen järjestelmätestausta. Tällöin varmistetaan, että kaikki ohjelmiston vaatimukset täyttyvät, jotta järjestelmätestaus voidaan tehdä. Alun vaatimusmäärittely sanelee pitkälti minkälaiset testitapaukset tehdään validaatiotesteihin ja suuntaa näyttävät myös suunnitteluvaiheeseen tehdyt järjestelmän toimintojen kuvaukset.

Käytännössä testitapaukset luodaan siten, että ovat hyvin lähellä oikeita käyttötilanteita. Myös validaatiotestaus voidaan ajaa automaation avulla. (Smart education, n.d.)

Järjestelmätestaus katsoo ohjelman kokonaisuutta ja tarkistelee, ovatko käyttötarkoitukset ja ohjelmalle asetetut vaatimukset kunnossa. Testaus tehdään aidossa ympäristössä huomioiden kaikki vaikuttavat sidosryhmät, kuten laitteet, käyttäjät ja mahdolliset tietokannat. Järjestelmätestauksen avulla on tarkoitus parannella ja viimeistellä ohjelman toimintaa. Suurin osa testeistä tehdäänkin aikaisemmilla testauksen tasoilla. Järjestelmätestaus voidaan jakaa useampaan osa-alueeseen, koska se on laajin kokonaisuus testaamisen tasoista. Osa-alueissa voidaan testata esimerkiksi suorituskykyä, käytettävyyttä, rasitusta tai asennusta. (Smart education, n.d.)

3.2 Testausmenetelmät ennen julkaisua

Alfa- ja Betatestaus ovat testaustapoja sellaisille ohjelmistoille, jotka ovat tehty myytäväksi. Alfa-testaus on testauksen ensimmäinen vaihe integrointi- ja järjestelmätestauksen jälkeen. Alfatestauksen tarkoituksena on testata ohjelmiston toimivuutta kokonaisvaltaisesti siten, että seuraava testausvaihe eli Betatestaus voidaan aloittaa. Alfatestauksessa ohjelmisto kokonaisuudessaan toimii, mutta kaikki toiminnot eivät vielä ole hiottuja valmiiksi lopputuotteeksi. (Kasurinen, 2013, s. 55–56)

Betatestaus voidaan toteuttaa yhdessä asiakkaan kanssa ja testikäyttäjät voivat tulla asiakasyritykseltä. Koska ohjelmisto ei ole vielä julkaistu käyttöön, siihen voidaan tehdä vielä mahdollisia korjauksia ja muutoksia. Betatestausta voidaan pitää myös tuotteen hyväksymistestauksena. (Kasurinen, 2013, s. 56)

Savutestauksen tarkoituksena on testata uuden ohjelmistoversion toimivuutta yksinkertaisilla testeillä. Näitä ovat esimerkiksi Web-sivuston internet-ajanvarauksen linkkien ja painikkeiden toimivuus, ja tietojen syöttäminen ajanvarausjärjestelmään. Savutestaus ei ole perusteellinen testaustapa, mutta se kattaa suurimman osan ohjelmiston päätoiminnoista ja mahdolliset virheet havaitaan jo aikaisessa vaiheessa. Jos testit läpäistään, voidaan jatkaa tarkempiin testeihin. Jos testit eivät mene läpi, on ilmoitettava

korjauksista tulevaa testausta varten. Toteutus pystytään tekemään manuaalisena tai automaationa. (Chauhan 2014; Kasurinen, 2013 s. 55)

3.3 Regressiotestaus

Regressiotestauksen ideana on varmistaa, että ohjelmisto toimii minkä tahansa ohjelmaan tehdyn muutoksen jälkeen oikein. Regressiotestaus ei ole erillinen testauksen muoto, vaan yleisesti tarkoittaa testaamisen ja testien toistamista uudelleen. Kehitysversioiden välillä voidaan toteuttaa regressiotestausta ja testata uusien ja aikaisempien versioiden muutokset ja toiminnallisuudet. Tärkeimpänä on näyttää toteen, että muutoksien jälkeen uudet sekä vanhat toiminnot eivät ole menneet rikki. Versionhallinnassa tehtyjen kehityshaarojen yhdistämisen jälkeen on hyvä varmistaa regressiotestien avulla myös yhdistetyn version toimivuus. Testausautomaation avulla regressiotestit pystytään ajamaan helposti projektin eri vaiheissa ja useaan otteeseen. (Kasurinen, 2013, s. 54)

3.4 Testausautomaatio

Testausautomaatio on testausmuoto, jota varten käytetään automatisoitua työkalua, jolla saadaan testattua useampaa testiä toistuvasti ja nopeammin. Automaatiotestien ideana on helpottaa ja nopeuttaa testausprosessia, tällöin testaajille jää aikaa myös toisiin tehtäviin. Automatisoidut testit voidaan ajaa esimerkiksi yöllä, jolloin seuraavana aamuna tehdään mahdolliset tarkastukset ja korjaukset testien tuloksista. (Kasurinen 2013, s.60)

On kuitenkin huomioitava, että testausautomaatio on vain työkalu, jolla nopeutetaan testien ajoprosessia. Se ei ota kantaa testien kattavuuteen tai suunnittelemiseen. Testausautomaation suunnittelussa olisi hyvä ottaa huomioon omalle testattavalle ohjelmalle testausautomaation-viitekehys, jossa mukana on erilaisia automaatioon liittyviä työkaluja. Testausautomaatio on tehokas työkalu oikein käytettynä, mutta jos sitä käytetään väärin, tai ei ole tarpeeksi osaamista, voidaan saada myös negatiivisia vaikutuksia. (Vala Group, n.d.)

Koska ohjelmistoja julkistetaan ja kehitetään nopealla aikataululla, on yhä tärkeämpää huomioida myös DevOps- ja ketterät menetelmät yrityksen kehitysprosesseissa. Jotta

ketteryyttä ja joustavuutta saadaan aikaan, on hyvä siirtyä jatkuvaan testaukseen. Testauksen tulee olla tiiviisti yhteydessä eri yksiköihin, kuten esimerkiksi kehitys- ja liiketoimintayksikköön. Automaatio antaa mahdollisuuden testata nopeasti ja tarpeeksi aikaisin. Sidosryhmät saavat tietoa mahdollisista riskeistä, joka auttaa liiketoiminnan päätöksenteossa. Menestyvän testausautomaation saavuttaminen ei ole helppoa, testausautomaatioaste on yrityksissä keskimäärin noin 16 %, jolloin suurin osa yrityksen testausprosesseista on vielä manuaalista. (Banerjee, 2019)

DevOps muodostuu kehittäjien (**developers**) ja palvelinylläpidon (**operations**) yhteistoiminnasta. Yrityksien kompastuskivenä on, että aikaa ja resursseja tuhlataan tuotejulkaisun yhteydessä sellaisiin toistuviin työvaiheisiin, jotka tehdään käsin. Suurin osa toistuvista työvaiheista olisi automatisoitavissa. DevOps pitää sisällään automatisoinnin, joka liittyy tuotteiden julkaisuun ja tuotannon ylläpitoon. Käytännössä turhat työvaiheet karsitaan ja toistuvat prosessit automatisoidaan. (Klemetti, 2013)

Testausautomaation käyttäminen ei poista kokonaan tarvetta manuaalitestaamiselle. Automatisointi on tarkoitettu muun muassa vähentämään manuaalisen regressiotestauksen määrää. Vaikka testausautomaation tekemisessä on lisävaivaa, sen ylläpito ja käyttäminen on helpompaa ja halvempaa toistuvaisessa käytössä verrattuna manuaalitestaukseen. Automatisointia tulisi harkita, jos testiä toistetaan useampaan kertaan. Savutestaus, integraatiotestaus ja yksikkötestaus ovat sopivia testaustapoja automaatiotestaukseen. Kyseisiä testaustapoja voidaan suorittaa aina uudelleen, kun tehdään regressiotestausta. Testausautomaation ideana ei ole löytää uusia virheitä, vaan varmistaa että komponentit toimivat vielä versiomuutoksienkin jälkeen. (Kasurinen, 2013, s.62)

Testausautomaation hyötynä on ajan säästö, nopeus ja toistettavuus. Testitapauksia on helppo ylläpitää ja samalla ne ovat uudelleenkäytettäviä. Manuaalitestauksesta voi tulla tylsää ja samalla virheet voivat lisääntyä. Testausautomaation avulla taas kustannukset pienevät ja testauksen tarkkuus sekä luotettavuus lisääntyvät. Testausautomaatio parantaa testien kattavuutta ja vapauttaa resursseja muihin testaustehtäviin. (Nabil, 2017; Vala Group, n.d.)

Testausautomaatioon on hyvä valita lisäksi sellaisia testejä, joita on vaikea suorittaa manuaalisesti tai jotka vievät erityisesti aikaa. Lisäksi mukaan on hyvä ottaa sellaiset testitapaukset, joissa on korkea riski suorittaa testi manuaalisesti väärin. Testausautomaatioon ei sovellu uudet testitapaukset, tai tapauskohtaisesti toteutettavat testit. Tähän samaan kategoriaan kuuluvat myös testitapaukset, joiden vaatimukset muuttuvat usein. (Nabil, 2017)

Testiautomaation keskeisin kysymys on, mitä tullaan automatisoimaan ja miten. Vaikka käytettävissä olisi suositellut työkalut, ei niillä tekisi mitään, ellei osaisi niitä käyttää. Asiantuntemusta tarvitaan testiympäristöjen luomiseen ja tehokkaaseen automaatiokehyksen asentamiseen. Automaatiokehyksen luomiseen tarvitaan lisäksi aikaa ja kustannuksia. Haasteena on, miten löytää oikeat henkilöt ylläpitämään ja perustamaan testausautomaatiota. Jatkuva testaus on tärkeää, mutta samalla myös erittäin haastavaa. Testaus on tietynlainen viimeinen etappi ohjelmistokehityksen laadun varmistamisessa. (Banerjee, 2019)

4 Testausautomaation työkalut

Kehittynyttä testausautomaatioteknologiaa on monenlaista ja maailmalla on kehitetty erilaisia sovelluksia tähän tarkoitukseen. Osa näistä perustuu avoimeen lähdekoodiin, ja testiautomaatiosovelluksia löytyy sekä ilmaisena, että maksullisena versiona. Yleensä ilmaisversioissa suuri merkitys on omalla osaamisella ja harjoittelulla. Useat yritykset voivat myös tarjota erilaisia testiautomaatiopilotteja. Piloteissa käydään läpi mahdollisia käyttöönottoprosesseja ja koulutuksia. (Qentinel, 2020; Eficode. n.d.)

4.1 Robot Framework ja Selenium

Robot Framework on automaatiotestauksen työkalu, joka pohjautuu avoimeen lähdekoodiin. Sitä voidaan käyttää testausautomaatioon sekä RPA:han, eli robotiikan prosessiautomaatioon. Robot Frameworkille on perustettu oma säätiö, joka sponsoroi Robot Frameworkin infran kehitystä ja mahdollistaa, että sitä pystytään käyttämään ilmaiseksi myös jatkossa. Robot Frameworkin kehitys on lähtenyt liikkeelle Pekka Klärckin diplomityöstä, jonka jälkeen sitä on kehitetty eteenpäin Nokia Networksilla. Käyttöön se otettiin vuonna 2008. (Bisht, 2013, s. 26; Robot Framework, n.d.)

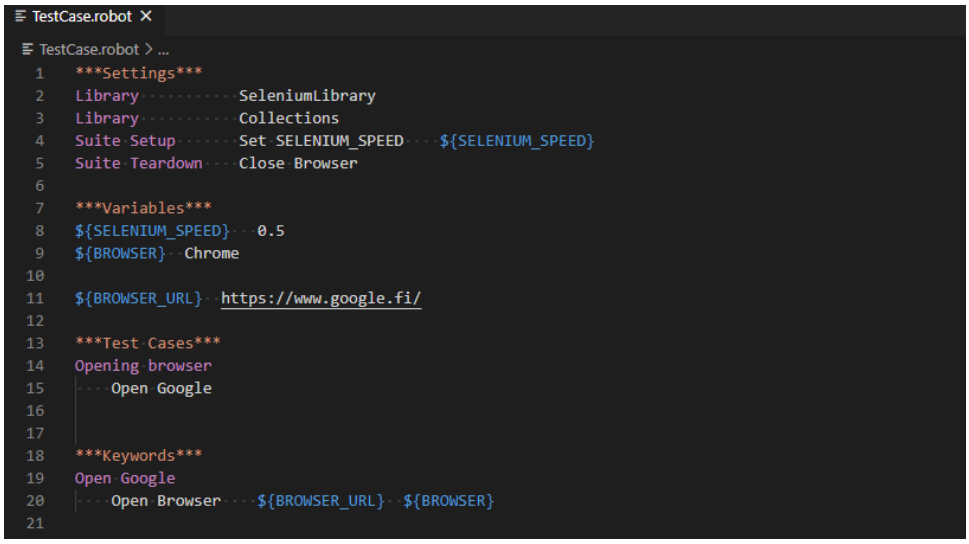
Avointa lähdekoodia voi kuka tahansa muokata ja jakaa. Malli on julkistettu kaikkien nähtäville. Avoimen lähdekoodin- konsepti on lähtenyt ohjelmistokehityksestä , mutta samalla ulottautunut myös tämän alueen ulkopuolelle, kuten tieteeseen ja mediaan. Projektit, jotka toteutetaan avoimen lähdekoodin- konseptilla, saavat pitkäaikaisia etuja myös tulevaisuudessa opetuskäytössä, prototyyppien kehityksessä, ja ideoiden jakamisessa. (Robocorp, n.d.)

Testitapaukset, jotka kirjoitetaan Robot Frameworkilla perustuvat erilaisiin ”Key Wordeihin”, joita voi käyttää monissa eri testitapauksissa. Vahvuutena on, että testitapaukset kirjoitetaan sellaiseen muotoon, että niitä on helppo käyttää uudelleen ja yhdistää testikirjastoihin. Testikirjastoja käytettäessä koko kehitysorganisaatio pystyy käyttämään samaa työkalua ja huomioimaan yhdenmukaisuuden testitapauksia tehdessä. Robot Frameworkin ydinteknologia perustuu Pythoniin ja testitapaukset tehdään luonnollisella kielellä tiedostoihin. (Eficode, n.d.)

Robot Frameworkia käytetään Web-automaatiotestaukseen yhdessä erillisen Selenium-kirjaston avulla. Selenium-kirjasto on tarkoitettu selainpohjaiseen Web-testaukseen. Se määrittelee joukon **Keywordeja,** eli avainsanoja, jotka soveltuvat erityisesti web-sivujen testien rakentamiseen. Pelkästään avainsanat eivät riitä, vaan niiden on oltava vuorovaikutuksessa testattavan verkkosivun elementtien kanssa ja tietylle elementille on löydettävä oma locator. Locator kertoo elementin koodipohjaisen sijainnin, tämä voi esimerkiksi olla ID tai XPath-elementti. (Salminen, 2020; Robot Framework SeleniumLibrary, 2020)

Testitapauksien tekemiseen löytyy erilaisia työkaluja, yksi tärkeimmistä on tekstieditori, jolla kirjoitetaan testitapaukset tiedostoon. Tunnetuin erillinen Robot Framework editori on RIDE, joka on tarkoitettu vain Robot Framework testitiedostojen muokkaamiseen. Muita tunnettuja editoreita on PyCharm, Eclipse ja Visual Studio Code. Tekstieditoriin saa lisäosilla erilaisia työkaluja, jotka helpottavat testitapausten muokkausta ja rakentamista. Moni näistä työkaluista on kehitetty erillisenä ja osa löytyy valmiiksi sisällettynä editoriin. (Robot Framework, n.d.; Test Guild, 2016)

Kuva 2 Testitapauksen rakenne

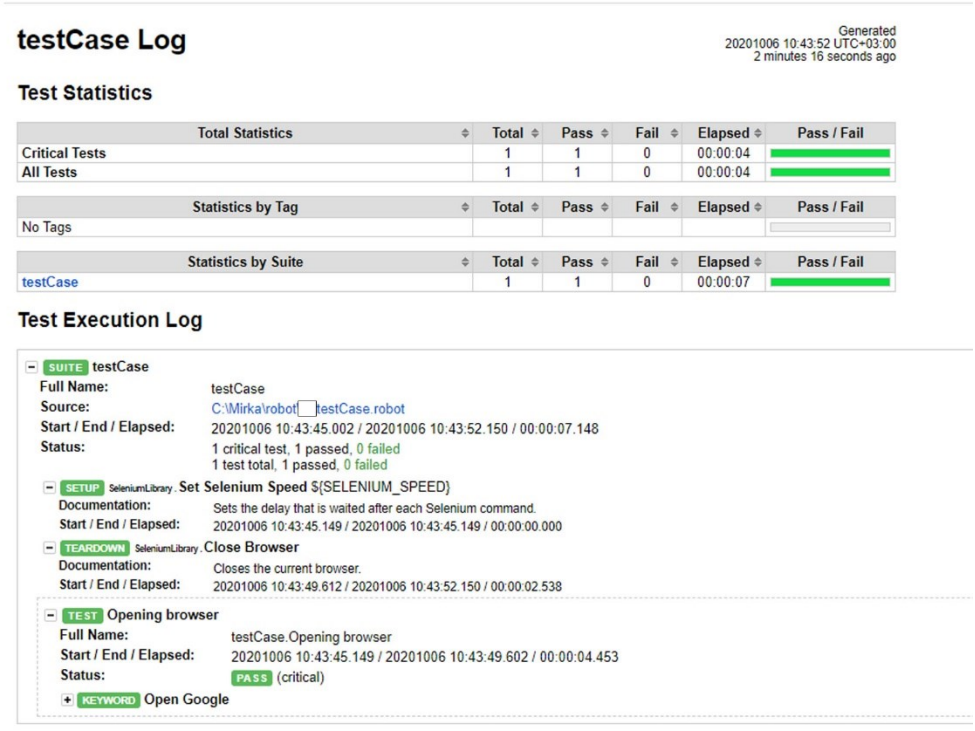


Tekstieditorissa testitapauksen rakenne voidaan jakaa neljään osaan (Kuva 2). **Settings**-osassa käydään läpi asetukset, joihin voidaan lisätä dokumentaatio ja kirjastot. Täällä voidaan lisäksi määrittää toimenpiteitä, jotka tehdään ennen ja jälkeen varsinaisen testiajon. Myös nämä toimenpiteet hyödyntävät keywordeja. **Variables**-osiossa luetellaan mahdolliset

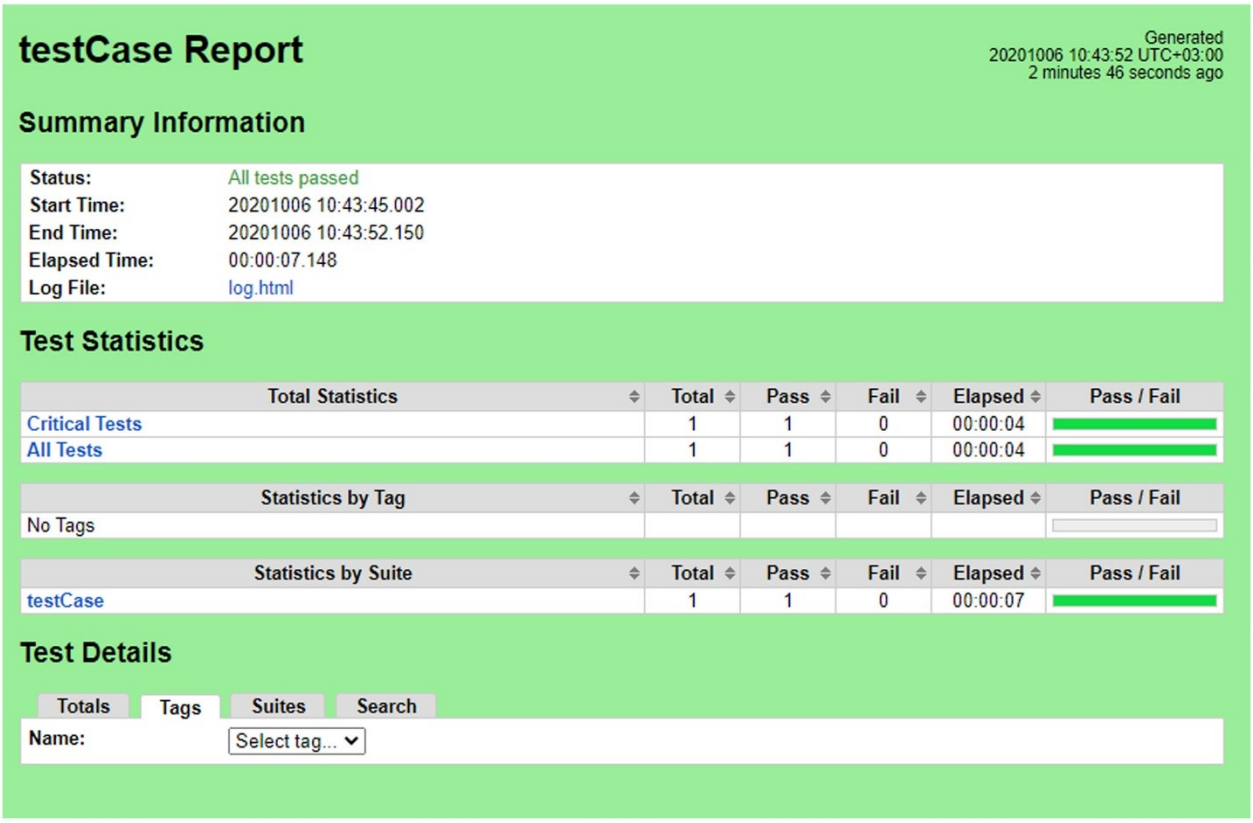
muuttujat, joita voidaan käyttää missä vain kohdassa testidataa. **Test Cases** -osiossa luodaan testitapaukset, joissa käytetään avainsanoja, eli Keywordeja. Avainsanat luodaan testitapauksen **Keywords**-osassa, varsinainen toiminta tapahtuu avainsanojen takana. Avainsanoina voidaan käyttää myös olemassa olevien kirjastojen avainsanoja, tai luoda näistä olemassa olevista uusia. (Robot Framework User Guide, 2018)

Jokaisen ajetun testitapauksen jälkeen ohjelma luo testistä raportit, josta nähdään miten testitapaus on edennyt ja missä kohtaa testiajossa mahdollinen virhe on sattunut (Kuva 3 ja 4). Lokista löytyy yksityiskohtaiset tilastot testin jokaisesta vaiheesta ja avainsanoista. Vikatilanteessa voidaan tarkastella helposti raportin kautta, missä ongelma on tapahtunut. Log-tiedosto ottaa lisäksi sivulta kuvakaappauksen, jolloin tarkastelu ongelmatilanteissa on yksityiskohtaisempaa (Kuva 5). Raportit löytyvät tiedostosta **log.html, report.html ja output.xml** – nimellä. Report.html pitää sisällään kokonaisraportin, log.html antaa ajon aikaisen lokin ja output.xml kokonaistulokset. (Robot Framework, n.d.; Robot Framework User Guide, 2018)

Kuva 3 Esimerkki hyväksytystä log-tiedostosta.



Kuva 4 Esimerkki hyväksytystä report-tiedostosta



testCase Report

Generated
20201006 10:43:52 UTC+03:00
2 minutes 46 seconds ago

Summary Information

Status:	All tests passed
Start Time:	20201006 10:43:45.002
End Time:	20201006 10:43:52.150
Elapsed Time:	00:00:07.148
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:04	
All Tests	1	1	0	00:00:04	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

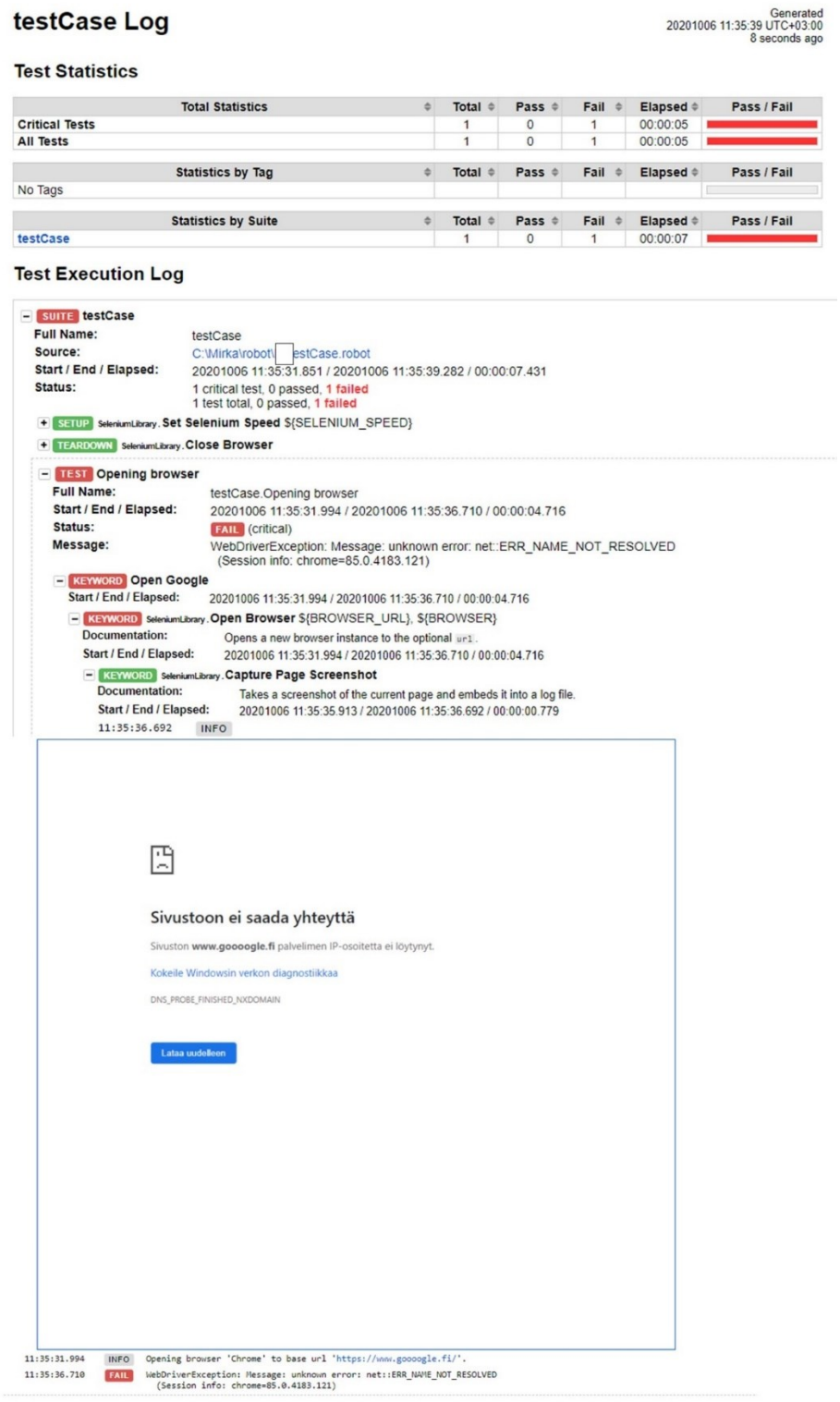
Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
testCase	1	1	0	00:00:07	

Test Details

Totals | Tags | Suites | Search

Name: Select tag...

Kuva 5 Esimerkki epäonnistuneen testiajon log-tiedostosta.



testCase Log

Generated
20201006 11:35:39 UTC+03:00
8 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	0	1	00:00:05	
All Tests	1	0	1	00:00:05	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
testCase	1	0	1	00:00:07	

Test Execution Log

SUITE testCase
Full Name: testCase
Source: C:\Mirka\robot\ estCase.robot
Start / End / Elapsed: 20201006 11:35:31.851 / 20201006 11:35:39.282 / 00:00:07.431
Status: 1 critical test, 0 passed, 1 failed
1 test total, 0 passed, 1 failed

SETUP SeleniumLibrary . Set Selenium Speed ${SELENIUM_SPEED}
TEARDOWN SeleniumLibrary . Close Browser

TEST Opening browser
Full Name: testCase.Opening browser
Start / End / Elapsed: 20201006 11:35:31.994 / 20201006 11:35:36.710 / 00:00:04.716
Status: FAIL (critical)
Message: WebDriverException: Message: unknown error: net::ERR_NAME_NOT_RESOLVED
(Session info: chrome=85.0.4183.121)

KEYWORD Open Google
Start / End / Elapsed: 20201006 11:35:31.994 / 20201006 11:35:36.710 / 00:00:04.716

KEYWORD SeleniumLibrary . Open Browser ${BROWSER_URL}, ${BROWSER}
Documentation: Opens a new browser instance to the optional url .
Start / End / Elapsed: 20201006 11:35:31.994 / 20201006 11:35:36.710 / 00:00:04.716

KEYWORD SeleniumLibrary . Capture Page Screenshot
Documentation: Takes a screenshot of the current page and embeds it into a log file.
Start / End / Elapsed: 20201006 11:35:35.913 / 20201006 11:35:36.692 / 00:00:00.779
11:35:36.692 INFO

Sivustoon ei saada yhteyttä
Sivuston www.goooogle.fi palvelimen IP-osoitetta ei löytynyt.
Kokeile Windowsin verkon diagnostiikkaa
DNS_PROBE_FINISHED_NXDOMAIN
Lataa uudelleen

11:35:31.994 INFO Opening browser 'Chrome' to base url 'https://www.goooogle.fi/'.
11:35:36.710 FAIL WebDriverException: Message: unknown error: net::ERR_NAME_NOT_RESOLVED
(Session info: chrome=85.0.4183.121)

4.2 Gherkin

Robot Framework tukee myös Gherkin-tyylillä tehtyjä testaustapauksia. Gherkin on tyyli, jonka avulla voi kuvata liiketoiminnan käyttäytymistä ilman, että tarvitsee mennä toteutuksen yksityiskohtiin. Gherkin käyttää muutamia tärkeitä avainsanoja, jotka antavat rakenteen ja merkityksen suoritettavalle määritykselle. Nämä avainsanat ovat käännetty englanniksi ja jokainen testitapauksen kohta alkaa omalla termillään **Given, When, Then, And,** tai **But.** (Salminen, 2020; Cucumber, 2019)

Testin määritykset rakennetaan seuraavasti: **Given** on tietty tilanne, jolla on esivaatimukset, **When**, kun jotain ilmenee ja **Then** on seuraava toiminta mitä tulee tämän jälkeen tehdä (Kuva 6). Gherkin-kielellä määritellyt testitapaukset ovat selkeitä myös henkilöille, joilla ei ole ohjelmointitaustaa. Testien uudelleen käyttö on myös helpompaa muiden tiimin jäsenien keskuudessa. (Jansen, B. 2019.)

Kuva 6 Esimerkki Gherkin-tyylillä kirjoitetusta testitapauksesta.

```
***Test Cases***
Syntax
#Tämä on esimerkki Gherkin syntaxista: Given, When Then
    Given Testin esivaatimukset
    When Kun jotain tapahtuu
    And Kun jotain muuta tapahtuu
    And Suoritetaan uudelleen joku tapahtuma
    Then Testin tulos on saavutettu
    And Myös jotakin muuta tapahtuu vielä

Esimerkkiskenaario
#Asiakas varaa ajan internet-ajanvarauksen kautta
    Given Asiakas avaa internet-ajanvarauksen
    When Asiakas valitsee toimenpiteen, vastaanoton, ja toimenpiteen suorittavan henkilön
    And Asiakas syöttää tunnistetiedot
    And Asiakas siirtyy vahvistukseen
    Then Asiakas tarkistaa tiedot ja hyväksyy varauksen
    And Henkilö saa ilmoituksen varauksen hyväksymisestä
```

4.3 Versionhallinta

Versionhallinta on tiedostojen tallentamista eri kehitysvaiheissa omaan versiohallintajärjestelmään siten, että järjestelmästä nähdään eri versioiden sisällöstä versioiden väliset muutokset sekä erot datassa. On hyvä muistaa, että versionhallinta ei koske pelkästään ohjelmointia, vaan mistä tahansa projektista voidaan luoda eri versioita ja tallentaa niitä omaan versionhallintajärjestelmään, jolloin ne voivat koskettaa kaikkia

henkilöitä, jotka jollakin tapaa liittyvät projektiin. Ohjelmistoa voidaan kehittää yhtä aikaa useamman henkilön toimesta, jolloin ohjelmistosta tehdään omia sivuhaaroja. Näitä sivuhaaroja kutsutaan **brancheiksi**. Varsinainen ohjelma etenee päähaarassa, jota kutsutaan **masteriksi**. Kun sivuhaaroja yhdistetään masteriin, tapahtuu **merge.** (Laurikkala, 2019).

Versionhallinnan ytimessä on palvelin, joka sisällyttää kootusti tallennetut versiot tietovarastoon, jota kutsutaan **repositoryksi**. Täällä päivitetään versiohistoria. Git on yksi versionhallintajärjestelmä, joka tekee yllä mainitut asiat ja siirtää tiedot omalle verkkopalvelulle. Verkkopalveluja voi olla esimerkiksi GitHub ja Azure. (Laurikkala, 2019)

5 Potilastietojärjestelmä

Potilastietojärjestelmä on terveydenhuollon ammattilaisten päivittäiseen työhön käyttämä järjestelmä, johon voi liittyä paljonkin potilashoitoon liittyvää toiminnollisuutta. Muun muassa ajanvarauksien hallinnointia, potilastietojen kirjaamista ja aikaisempien tietojen hakemista. Järjestelmän avulla tehostetaan ja nopeutetaan potilaan hoitoa. Sen avulla saadaan kerättyä yhteen perusterveydenhuollon, sosiaalitoimen sekä erikoissairaanhoidon kirjaamat tiedot. Yhtenäinen järjestelmä antaa ajantasaisen tilannekuvan potilaan terveydestä. Tietoa voidaan jakaa niin maakunnallisella tasolla, kuin yksikkötasollakin. Asiakas- ja potilastietojärjestelmän tietojen saatavuudessa on huomioitava ammattilaisten käyttövaltuudet tietosuojalainsäädännön sallimissa rajoissa. (Keski-Suomen sairaanhoitopiiri, 2020.)

Tietojärjestelmille ja niiden valmistajille, sekä sosiaali- ja terveydenhuollon palvelun antajille yleiset vaatimukset määrittelee sosiaali- ja terveydenhuollon asiakastietojen sähköinen käsittely (159/2007)-laki. Valvira valvoo tietojärjestelmien vaatimusten toteutumista. Valvira ylläpitää rekisteriä tietojärjestelmistä, jotka ovat vaatimusten mukaisia. Tietojärjestelmät lajitellaan A ja B luokkaan ominaisuuksiensa ja käyttötarkoitustensa perusteella. Luokkaan A kuuluvat Kanta-palvelut, joita ylläpitää Kansaneläkelaitos sekä tietojärjestelmät, jotka liitetään suoraan tai erillisen välityspalvelun kautta Kanta-palveluihin. Luokkaan B kuuluvat kaikki muut tietojärjestelmät. Tietojärjestelmärekisteri päivittyy kerran kuussa ja löytyy Excel-muotoisena Valviran-internetsivuilta. (Valvira, n.d.)

Kanta-palvelut ovat terveyden- ja sosiaalihuollon sähköinen tietokanta. Palvelua käyttää mm. julkisen ja yksityisen terveydenhuollon palveluntuottajat, apteekit ja kansalaiset. Potilastiedot ovat ajan tasalla ja kätevästi käytettävissä hoitotilanteessa. Kanta-palvelu on palvelukokonaisuus, joka sisältää Omakannan, josta henkilö näkee muun muassa omat terveystiedot ja reseptit. Kanta-palveluista löytyy myös sähköinen reseptipalvelu, lääketietokanta ja potilastiedon arkisto. Kanta-palvelut on otettu käyttöön porrastetusti vuodesta 2010 lähtien ja palvelua laajennetaan ja kehitetään edelleen. (Kanta, n.d.)

Tietokanta on tiedon tallentamiseen tarkoitettu paikka. Se on kokoelma kohdennettuun aihealueeseen kuuluvia säilytettäviä tietoja. Tietokannan käyttöönotto voidaan tehdä

esimerkiksi yritykselle tai yhdistykselle, jotta saadaan säilytettyä ja haettua tietoa. Tietokannan etuina on keskitetty tietojen hallinta. Myös tietojen päällekkäisyys pienenee ja tietoa saadaan hallittua paremmin. Järjestelmää tehdessä huomioidaan lisäksi tietojen eri käyttötarpeet. (Salminen, 2020; University of Helsinki n.d.)

5.1 Hammashoidon potilastietojärjestelmä

Hammashoidon potilaskäynteihin kuuluvat suun perustarkastukset ja tähän liittyvät toimenpiteet, jotka suorittaa hammaslääkäri tai muu alalle koulutettu ammattihenkilö. Toimenpiteen aikana tehdään suun tutkimus, ja tutkimuksen aikaiset löydökset kirjataan potilastietojärjestelmään. Hammashoidon potilastietojärjestelmästä löytyy visuaalinen mallinnus hammaskartasta, johon syötetään oikean hampaan kohdalle tietty koodi tehdystä havainnosta sekä mahdolliset toimenpiteet ja jatkosuunnitelmat. Järjestelmästä löytyy tarvittavat potilastiedot ja suunnitelmat myös tulevia käyntejä varten. (Aalto, Virkkunen, Turunen, Saarela & Räty, 2020, s.11)

Hammashoidon potilastietojärjestelmistä löytyy laaja skaala erilaisia ominaisuuksia ja lisätoimintoja. Potilaskirjauksien lisäksi järjestelmistä voi löytyä mm. ominaisuuksia ajanvaraukseen, laskutukseen, kirjanpitoon ja raportointiin liittyen. Järjestelmässä voi olla erilaisten liittymien integrointimahdollisuus esimerkiksi röntgenlaitteisiin, maksupäätteisiin sekä hammashoidon muihin työkaluihin. Yksityisen puolen potilastietojärjestelmistä voi löytyä myös sähköinen ominaisuus Kelan suorakorvauksen hakemiseen, sekä liittymä Kanta-palveluun, jonne viedään sähköiset reseptit ja potilastiedot. (Receptum, n.d.; Opus Dental, n.d.)

Kelan suorakorvausta käyttävät palveluntuottajat, jotka ovat tehneet erillisen sopimuksen suorakorvausmenettelystä Kelan kanssa. Kelalta saa osakorvauksen yksityisen hammaslääkärin toimenpidekustannuksista. Korvausta ei saa julkisen hammashoidon kustannuksista tai toimenpiteistä, jotka ovat kosmeettisia. Suorakorvaus vähennetään hoitopaikassa suoraan maksun yhteydessä Kela-korttia näyttämällä. Henkilö joutuu hakemaan korvausta manuaalisesti, jos sähköistä ominaisuutta ei ole palveluntuottajalla käytössä. (Kansaneläkelaitos, 2016)

Vuodesta 2017 lähtien sähköinen resepti on tullut pakolliseksi terveydenhuollon järjestelmiin. Sähköinen resepti löytyy integroituna useimmista hammashoidon potilastietojärjestelmistä, mutta reseptien kirjoittamiseen voidaan käyttää myös erillistä järjestelmää. Hammaslääkäri laatii ja allekirjoittaa sähköisen lääkemääräyksen, joka tallennetaan Kanta-palveluun. Kanta-palvelun avulla esimerkiksi lääkkeiden hankinta voi tapahtua mistä tahansa apteekista. (Suomen Hammaslääkäriliitto, 2016)

Vuoden 2017 alussa on ollut myös virallinen määräaika potilastiedon arkistoon siirtymisestä. Kanta-palvelun arkiston avulla saadaan potilastiedot säilytettyä turvallisesti ja pitkäaikaisesti, sekä jaettua tarvittaessa eri terveydenhuollon toimijoiden kesken. Tämän lisäksi potilas pääsee näkemään omat tietonsa Oma Kanta-palvelun avulla. Hammaslääkärivastaanotolla arkistoa käytetään hammashoidon potilastietojärjestelmän kautta, jolloin ammattihenkilö saa tarvittavat historiatiedot potilaasta, vaikka potilas ei olisi aiemmin käynytkään vastaanotolla. Yhteneväinen potilastietojen kirjaamistapa myös parantaa potilasasiakirjojen tasoa. (Tilander, 2016; Tilander, 2017)

5.2 Tietosuojalaki

Henkilötietojen käsittelyä koskeva tietosuojalaki 5.12.2018/1050 on astunut voimaan 1.1.2019. Tietosuojaperiaatteita tulee noudattaa aina henkilötietoja käsitellessä. Henkilötiedoiksi luetaan tiedot, joista voidaan tunnistaa luonnollinen henkilö. Henkilötietoja ovat esimerkiksi nimi, kotiosoite, sähköpostiosoite, puhelinnumero, ja potilastiedot. Henkilötietoja löytyy esimerkiksi tietokannoista ja sähköisistä tiedostoista. (Tietosuojavaltuutetun toimisto, n.d.)

Rekisterinpitäjän pitää kyetä todistamaan, että tietosuojaperiaatteita noudatetaan henkilötietojen käsittelyssä. Osoitusvelvollisuuden periaatteena on, että mahdollisessa tietoturvaloukkaustilanteessa rekisterinpitäjä on tehnyt tarvittavat ennakoivat toimenpiteet ja tunnistanut mahdollisia riskejä henkilötietojen suojaamiseksi. Tietosuoja.fi-sivulta löytyy luettelo tietosuojaperiaatteiden mukaisista henkilötiedoista ja niiden käsittelystä. Tähän kuuluu, että tietoja tulee esimerkiksi käsitellä lainmukaisesti mutta kuitenkin läpinäkyvästi. Kerättävällä tiedolla on oltava nimenomainen laillinen tarkoitus, ja sitä on käsiteltävä

tarkoitusta varten. Tarkoitukseen on kerättävä vain tarpeellinen määrä henkilötietoja, ja tiedot on päivitettävä ja oikaistava, jos virheellisiä tietoja löytyy. Tiedot on käsiteltävä luottamuksellisesti. (Tietosuojavaltuutetun toimisto, n.d.)

Henkilötietojen käsittelyn on oltava läpinäkyvää. Rekisteröidyn on saatava tietää ymmärrettävästi mitä ja mihin tarkoituksiin tietoja kerätään ja millä tavoin niitä ollaan käsittelemässä. Lisäksi on kerrottava, minkälaiset oikeudet rekisteröidyllä on. Rekisteröidyllä on mm. oikeus saada tietoa henkilötietojensa käsittelystä, rajoittaa käsittelyä, ja oikaista ja poistaa tietoja. Kaikkia oikeuksia ei kuitenkaan voi käyttää kaikissa tilanteissa, tilanteeseen vaikuttaa millä perusteella henkilötietoja on käsitelty.(Tietosuojavaltuutetun toimisto, n.d.)

6 Testausautomaatio toimeksiantajalle

Kevään 2020 aikana Yritys X on lähtenyt kehittämään testausautomaatiota SaaS-pohjaiseen Hammashoidon potilastietojärjestelmään. Testausautomaatio-kehityksen ulkopuolelle on jäänyt web-käyttöliittymä, internet-ajanvaraus, jolle lähdettiin opinnäytetyön avulla toteuttamaan ratkaisua selainpohjaiseen testausautomaatioprosessiin. Koska selainpohjaista testausautomaatiota Yritys X:llä ei aiemmin ole ollut, on testausautomaatiota lähdetty työstämään tyhjältä pöydältä. Alussa testejä on lähdetty tekemään koulussa opituilla tavoilla. Loppua kohden työtä on muutettu ja paranneltu sitä mukaa, kun olen omassa työssäni oppinut automaatiotestausprosessiin kuuluvista asioista ja käytännöistä. Opinnäytetyön tarkoituksena on helpottaa Yritys X:n prosesseja ja auttaa selainpohjaisen web-sivuston testaustoimenpiteitä automaatiotestien avulla.

Yritys X saa käyttöönsä opinnäytetyöprosessin aikana tehdyt automatisoidut testitapaukset. Nämä automatisoidut testit luovat lähtötason, josta Yritys X voi lähteä muokkaamaan ja ylläpitämään prosessia eteenpäin itselle määrittelemällään ja sopivalla tavalla. Tässä opinnäytetyön käytännön osassa annetaan esimerkki automatisoidusta testitapauksesta, jossa testataan, toimiiko ajanvarauksessa oleva Tietosuoja-linkki ja vastaako sen osoite asetettua.

Opinnäytetyön käytännön osuus lähti liikkeelle kesäkuun 2020 alussa yhteisellä palaverilla, jossa mukana oli Yritys X:n yhteyshenkilöitä, jotka mm. työskentelevät internet-ajanvaraukseen web sivuston ja ohjelman testaamisen ja automaatiotestauksen parissa. Palaverin jälkeen sain tiedoksi mallit manuaalitesteistä, internet-ajanvaraus-linkin, missä testaaminen tapahtuu, sekä ohjeet ja pääsyn testikantaan. Tämän jälkeen varsinaisten automatisoitujen testien rakentaminen alkoi.

6.1 Testaamisen lähtökohdat

Internet-ajanvarauksen toimivuutta testataan manuaalisesti aina päivityksien ja versiomuutosten jälkeen. Testaaminen on välttämätöntä, koska internet-ajanvaraus on sidoksissa varsinaiseen ohjelmistoon, potilastietojärjestelmään. Regressiotestauksen ideana lähtökohtaisesti on tarkistaa, että ajanvaraukseen syötetyt tiedot siirtyvät eteenpäin

potilastietojärjestelmään ja siellä olevaan kalenteriin. Lisäksi yleisien painikkeiden ja linkkien toimivuus, kielivalinnat, sekä virheilmoitukset kuuluvat normaaleihin testausprosesseihin. Manuaalitestauksen aikana joudutaan myös lisäämään uusia elementtejä testitietokantaan, potilastietojärjestelmään. Nämä elementit voivat olla esimerkiksi uusia toimipaikkoja ja palveluita, jolloin ne saadaan ajanvarauksen valintaan mukaan.

Testausympäristöjä Yritys X:ltä löytyy kolme kappaletta, jolloin testaamista tulee myös tehdä useammassa testausympäristössä. Testausympäristöissä voi esimerkiksi olla käytössä ohjelman eri versioita ja jokaisella ympäristöllä on oma datansa, jota päivitetään testaamiseen tarkoitetussa tietokannassa. Ohjelman eri versioissa voi olla esimerkiksi eroavaisuuksia käyttöliittymässä näkyvien elementtien välillä. Tietokannoissa eroavaisuudet tulevat muun muassa toimipaikoista, palveluista ja hoidon suorittavista työntekijöistä. Automaatiotestejä voidaan soveltaa useampaan testausympäristöön. Testit tulee kuitenkin muokata vastaamaan tietokannan dataa testausympäristön mukaisesti.

Yritys X:llä on ollut Robot Framework aiemmin käytössä, jotta testit toimivat tulee lisätyökaluksi asentaa **SeleniumLibrary**, sekä **Seleunium Webdriver**. Asennus voidaan tehdä komentokehoitteella tai tekstieditorin, Visual Studio Code, Terminalissa. SeleniumLibaryn asennus tapahtuu yksittäisellä komennolla (Komento 1). Webdriver asennus tapahtuu kaksiportaisesti, ensimmäisessä osassa asennetaan webdrivermanager (Komento 2). Toisessa osassa määritellään selain, jota käytetään testeissä. Opinnäytetyöhön tehdyt testit ovat suoritettu Chrome-selaimella, jolloin tämä määritellään komentoon (Komento 3). Webdriverin asennuskansion polku kopioidaan komentoriviltä ja lisätään muuttujaksi Path-kansioon.

Komento 1 SeleniumLibraryn asennus

```
pip install -upgrade.robotframework-seleniumlibrary
```

Komento 2 Webdrivermanagerin asennus

```
pip install webdrivermanager
```

Komento 3 Webdrivemanagerin selaimen määritys

```
webdrivermanager chrome
```

Testien kirjoittamiseen valikoitui alussa tekstieditoriksi RIDE-ohjelma. Prosessin loppuvaiheessa tekstieditori vaihtui kuitenkin Visual Studio Codeen. Vaihdos tuotti paljon muutoksia esimerkiksi yleiseen näkymään ja testitapauksien kirjoitusmuotoon, mutta samalla se selkeytti yksittäisen testin näkymää ja toimintatapaa. Tekstieditorin vaihto antoi mahdollisuuden käyttää Visual Studio Codessa olevia lisäosia, jotka avustavat testien kirjoittamisessa. Lisäosia ovat esimerkiksi Robot Framework Debugger ja Robot Framework Intellisense. Automatisoidut testitapaukset tallennetaan **.robot** -muotoiseen tiedostoon, jolloin Yritys X voi halutessaan valita minkä tahansa itselleen sopivan tekstieditorin automaatiotestien muokkaamiseen.

Visual Studio Code myös mahdollistaa projektien viennit versiohallintaan. Ohjelmassa on sisään integroitu Git-versionhallintajärjestelmä, mutta lisäosien avulla saadaan käyttöön myös muiden palveluntarjoajien vaihtoehdot. Automatisoidut testit on hyvä viedä versiohallintaan talteen. Useampi projektihenkilö pystyy työskentelemään saman projektin kanssa ja versiohallintaan jää muistijälki ja kommentit testeihin tehdyistä muutoksista. Yritys X voisikin hyödyntää Visual Studio Coden mahdollisuuksia omassa versiohallinnan talletuksessaan, tai käyttää omaa ratkaisuaan.

6.2 Esimerkki automatisoidusta testitapauksesta

Regressiotestauksen aikana voidaan tarkistaa esimerkiksi painikkeiden ja linkkien toimivuutta. Ajanvarausjärjestelmässä tulee olla näkyvä linkki tai painike yrityksen tietosuojakäytännöstä. Testitapauksessa käsitellään tämän painikkeen toimivuutta ja varmistetaan, että kyseinen internetsivun osoite, jota painikkeella painetaan antaa saman arvon kuin mitä on asetettu painikkeen alle ohjelman asetuksissa. Jos osoite ei täsmää asetettuun arvoon, testi ei mene läpi.

Manuaalitestauksessa tietosuojalinkin toimivuutta testattaisiin manuaalisesti seuraavin toimenpitein: Avaa selain ja syötä osoitteeksi internet-ajanvarauksen osoite. Paina internet-ajanvaraus -sivun Tietosuojalinkki-painiketta. Viereiselle välilehdelle tulisi avautua asetuksissa määritelty sivusto, tarkista osoitteiden yhteneväisyys. Sulje selaimet.

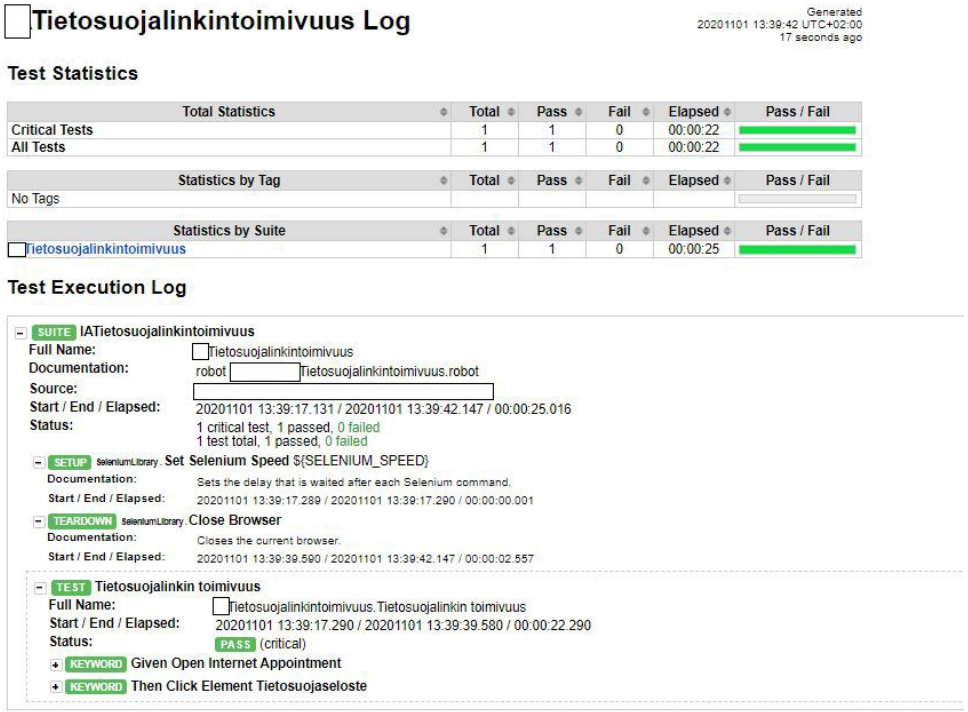
Kun lähdetään rakentamaan automatisoitua testitapausta, otetaan huomioon alla lueteltavia asioita. Testitapauksen **Settings**-osion alle voidaan lisätä mahdollista dokumentaatiota, sekä käytettävät kirjastot. Tässä testitapauksessa dokumentaation alle on lisätty robot-käsky, jolla aloitetaan testin ajaminen Terminalissa. Kun testejä ajetaan yksitellen, voidaan komentokäsky kopioida helposti testitapauksen Settings-osiosta käyttöön. Osion alta löytyy myös **Resource**-kansiot, joiden avulla voidaan yhdistää muiden kansioiden sisältöä ajettavan testin mukaan. Esimerkkitestissä käytetään erillisiä **Variables**- ja **Keyword**-kansioita, jotka sisällytetään mukaan tähän esimerkkitestiin. Erillisestä Variables-kansioista löytyy esimerkiksi yleiset muuttujat kuten **${BROWSER_URL}** ja **${BROWSER}**. Muuttuja ${BROWSER_URL}:n alle on määritelty internet-ajanvarauksen osoite ja ${BROWSER} määrittää selaimen arvoksi Chromen. **Suite Setup** asettaa testiympäristön määritykset ennen kuin testi lähtee ajoon ja **Suite Teardown** sulkee selaimen testin ajon jälkeen (Kuva 7).

Kuva 7 Esimerkki Tietosuojalinkin toimivuustestistä.

```
*** Settings ***
Documentation
...                 robot [redacted]Tietosuojalinkintoimivuus.robot


Library             OperatingSystem
Library             Collections
Library             SeleniumLibrary
Library             DateTime

Resource            ./Variables.robot
Resource            ./Keywords.robot

Suite Setup     Set Selenium Speed      ${SELENIUM_SPEED}
Suite Teardown      Close Browser

***Variables***
${tietosuojalinkki}=    [redacted]


***Test Cases***
Tietosuojalinkin toimivuus
    Given Open Internet Appointment
    Then Click Element Tietosuojaseloste

***Keywords***
Open Internet Appointment
    Open Browser    ${BROWSER_URL}  ${BROWSER}
    Location Should Be    ${BROWSER_URL}

Click Element Tietosuojaseloste
    Wait until page contains element    xpath://a[contains(text(),'Tietosuojaseloste')]
    Click element    xpath://a[contains(text(),'Tietosuojaseloste')]
    ${handle}    Get Window Titles
    Select Window    title=@{handle}[1]
    Location Should Be    ${tietosuojalinkki}
```

Variables-kohdassa on testikohtaiset muuttujat, esimerkkitapauksessa ${tietosuojalinkki} määrittää halutun osoitteen. Muuttujia käytetään **Keyword**-osiossa, jossa suoritetaan tapahtuma halutulle muuttujalle. Muuttujat voivat olla toisessa kansiossa tai oman testitapauksen sisällä. **Test Cases** -osiossa tapahtuu varsinaiset testin polkukomennot Gherkin syntaxin ja Keywordien muodossa. Luotu Keyword, **Open Internet Appointment** avaa selaimen omilla määrityksillään. Tämän jälkeen **Click Element Tietosuojaseloste** etsii sivulta elementin, joka sisältää tekstin ”Tietosuojaseloste”. Tämän jälkeen painaa elementtiä, jolloin uusi välilehti avautuu ja ohjautuu asetetulle sivulle. Testi tarkastaa molempien välilehtien otsikot ja valitsee aktiiviseksi juuri avatun välilehden. Testi tarkastaa, että lokaatio on sama kuin annettu muuttujan arvo, jossa on määritelty tietosuojalle asetettu sivuston osoite ja menee hyväksytysti läpi (Kuva 8). Testiin voidaan kokeilla vaihtaa muuttujan arvoksi toinen osoite, jolloin testin ei pitäisi mennä läpi ja antaa virheilmoitus esimerkiksi muodossa **Location should have been ’https://osoite.fi’ but was ’ https://osoite1.fi’**

Kuva 8 Tietosuojalinkin toimivuustestin hyväksytty log.-tiedosto



Tietosuojalinkintoimivuus Log

Generated
20201101 13:39:42 UTC+02:00
17 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:22	
All Tests	1	1	0	00:00:22	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Tietosuojalinkintoimivuus	1	1	0	00:00:25	

Test Execution Log

SUITE IATietosuojalinkintoimivuus
Full Name: Tietosuojalinkintoimivuus
Documentation: robot Tietosuojalinkintoimivuus.robot
Source:
Start / End / Elapsed: 20201101 13:39:17.131 / 20201101 13:39:42.147 / 00:00:25.016
Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

SETUP SeleniumLibrary . Set Selenium Speed ${SELENIUM_SPEED}
Documentation: Sets the delay that is waited after each Selenium command.
Start / End / Elapsed: 20201101 13:39:17.289 / 20201101 13:39:17.290 / 00:00:00.001

TEARDOWN SeleniumLibrary . Close Browser
Documentation: Closes the current browser.
Start / End / Elapsed: 20201101 13:39:39.590 / 20201101 13:39:42.147 / 00:00:02.557

TEST Tietosuojalinkin toimivuus
Full Name: Tietosuojalinkintoimivuus.Tietosuojalinkin toimivuus
Start / End / Elapsed: 20201101 13:39:17.290 / 20201101 13:39:39.580 / 00:00:22.290
Status: PASS (critical)

KEYWORD Given Open Internet Appointment
KEYWORD Then Click Element Tietosuojaseloste

6.3 Ongelmat ja niiden ratkaisut

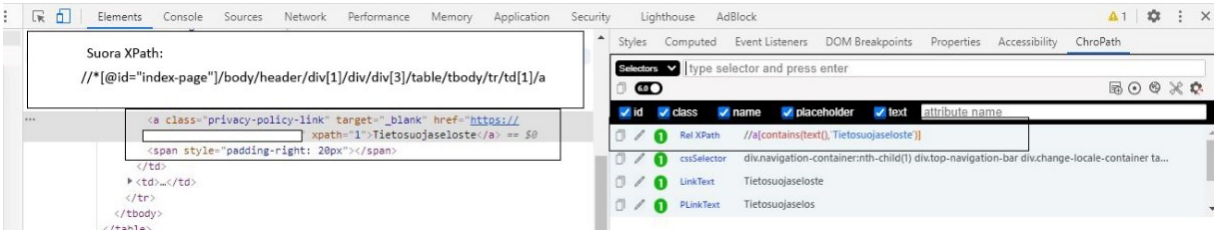
Testit ovat riippuvaisia manuaalisesti luoduista esiehdoista, jotka syötetään potilastietojärjestelmän testitietokantaan. Testit ovat kuitenkin yritetty luoda siten, että hakevat tietystä polusta haluttua elementtiä nimen mukaan. Muun muassa toimipisteistä ja palveluista on muodostettu muuttujat ja muuttujan arvoa vaihtamalla saadaan syötettyä testiin tarvittavat vaatimukset. Ajanvarauksen vapaat ajat syötetään etukäteen tietokantaan, ja tämän myötä ajat voivat vaihdella paljon. Kun testataan ajanvarauksen tekemistä, yhtenä ratkaisuna tähän on testiin rakennettu Keyword, joka ottaa listasta ensimmäisen vapaan ajan. Jos testejä ajetaan testikannasta löytyvien vanhojen tietojen mukaan, ilman että käytäisiin luomassa uusia aikoja työntekijöille tiettyihin toimipisteisiin, voivat testit epäonnistua väärien tietojen perusteella.

Yhtenä ratkaisuna olisi esimerkiksi tietokannan automaattinen generointi, jolloin lähtötilanne olisi sama jokaisella kertaa jokaisessa testausympäristössä, kun lähdetään ajamaan testejä. Luotaisiin automaattisesti testiin liittyvät esiehdot ja testin jälkeen tietokanta siivottaisiin alkuperäiseen muotoonsa. Generointi auttaisi pitämään automatisoidut testitapaukset puhtaana, esimerkiksi muuttujien arvoja ei tarvitsisi erikseen vaihtaa. Lisäksi manuaalisen alkukäsittelyn voisi jättää pois ja koko prosessin saisi automatisoitua. Yritys X:n tulisi pohtia mitä tähän vaadittaisiin ja selvittää onko tietokannan generointi mahdollista tulevaisuudessa.

Haasteeksi muodostui myös **XPath**-locatorien muokkaus. XPath-locatorit haetaan XML-polun avulla. Locatorina olisi paras käyttää olemassa olevaa elementin ID:tä tai luokkaa. ID:tä ei aina ole kuitenkaan annettu, joten Locator pitää muodostaa muilla tavoilla. Koulussa opetetulla tavalla, käytin automaatiotestien tekemiseen suoraa XPath-locatoria. Suora XPath-locator on kuitenkin paljon haavoittuvaisempi, jos ohjelman koodiin tehdään erillisiä muutoksia. Tällöin testit eivät välttämättä enää toimikaan, jos muutos koskee myös olemassa olevaa polkua. Vähemmän haavoittuvaisempi tapa on lyhentää suoraa XPathia Relative XPath -muotoon. Käytännössä elementin alkuun lisätään kaksi vinoviivaa, joka kertoo, että elementin etsiminen voi alkaa mistä tahansa kohtaa polusta. Lisäksi etsitään mahdollisia attribuutteja ja arvoja mitä polku voisi sisältää. Locatorin lyhentäminen on kuitenkin hidasta ja vaivalloista, varsinkin jos aikaisempaa kokemusta tämän tyyppisestä

etsimisestä ei ole. Sain omalta työpaikalta ehdotuksen käyttää Chromelle sopivaa lisälaajennusta, ChroPathia**,** joka lyhentää suoran XPath-elementin lyhyempään muotoon (Kuva 9).

Kuva 9 Esimerkki XPath-elementin lyhentämisestä ChroPathilla.



6.4 Tulevaisuuden näkymät

Automatisoiduista testeistä voidaan lähteä jatkamaan prosessia eteenpäin yrityksen tulevaisuuden visioiden mukaan. Opinnäytetyössä tehtyjen automaatiotestien osa-alueita voidaan ottaa mukaan myös muihin testauksen osa-alueisiin, kuten esimerkiksi savutestaukseen. Lisäksi Testausautomaatiota voitaisiin lähteä jatkokehittämään DevOps-mallin mukaan ketterämmäksi, jolloin Robot Frameworkilla suoritettavien testien ajo myös automatisoitaisiin. Ajot voisivat tapahtua yöllä ja aamulla käytäisiin läpi mahdolliset virheraportit ja näiden korjaamiset. Tämä vaatisi kuitenkin kaikkien testien automatisointia alusta lähtien ja pitäisi ratkaista millä tavalla alun manuaalivaiheet automatisoitaisiin.

7 Yhteenveto

Opinnäytetyön tavoitteena oli automatisoida Yritys X:n manuaalitestit, jotka koskevat hammashoidon internet-ajanvarauksen web-käyttöliittymää. Työssä myös pohdittiin mahdollisia ongelmia, jotka koskivat XPath-locatoria, manuaalista alkukäsittelyä ja tietokannan generointia. Työn aikana toteutettiin automatisoidut testitapaukset, jotka Yritys X saa käyttöönsä tiedostona. Prosessi kuitenkin jatkuu vielä opinnäytetyön kirjoittamisen jälkeen testitapauksien viimeistelyllä. Viimeistely johtuu pitkälti siitä, että olen loppuvaiheessa saanut enemmän tietotaitoa Robot Framework -työskentelystä ja testitapauksien kirjoittamisesta. Lisäksi toimeksiantajan kanssa tullaan pitämään palaveri työn tuloksista ja katsotaan läpi mahdolliset muutokset.

Opinnäytetyön tutkimuskysymykset muotoutuivat prosessin edetessä ja työn avulla löydettiin vastaus kysymyksiin. Teoriaosassa käsiteltiin automaatiotestausta ja testausmenetelmiä, joita tehdään ennen julkaisua. Sekä regressiotestausta, joka käsittää testitapauksien uudelleen testaamisen. Laajin ja tärkein tutkimuskysymys kosketti testausautomaation työkaluja Robot Frameworkia ja Seleniumia. Näiden toimintaa käsiteltiin teoriaosassa sekä käytännön osan testiesimerkissä.

Lähdin työstämään ulkopuoliselle toimeksiantajalle opinnäytetyötä ja automatisoituja testitapauksia koulusta saaduilla opeilla. Kun alkusyksystä vaihdoin omalla työpaikalla työnkuvaa automaatiotestauksen puolelle, sai opinnäytetyö uuden suunnan. Opinnäytetyön aihe selkeytyi konkreettisesti ja aikaisemmin tehtyjä asioita joutui muokkaamaan sitä mukaa kun oppi uutta oman työkokemuksen kautta. Opin paljon opinnäytetyöprosessin aikana, ja tätä suurimmaksi osaksi edesauttoi työnkuvan muutos. Ilman muutosta uskoisin, että opinnäytetyö olisi jäänyt vajavaiseksi. Opinnäytetyö auttoi saamaan hyvän alun työelämässä ja testausautomaatiotyöskentelyyn sai myös yritysnäkökulmaa.

Lähteet

Aalto, A., Virkkunen, H., Turunen, S., Saarela, H.-L. & Räty, T. (2020). *Suun terveydenhuollon potilaskertomusmerkintöjen toiminnalliset määrittelyt*. Terveyden ja hyvinvoinnin laitos 4/2020. Haettu 25.11.2020 osoitteesta https://www.julkari.fi/bitstream/handle/10024/139695/Suun%20terveydenhuollon%20toiminnalliset%20m%c3%a4%c3%a4rittelyt_2020_FINAL.pdf?sequence=1&isAllowed=y

Banerjee, K. (2018). Test Automation – A Recipe for Continuous Delivery Success. Haettu 18.10.2020 osoitteesta https://dzone.com/articles/test-automation-a-recipe-for-continuous-delivery-s

Bisht, S. (2013). Robot Framework Test Automation. E-Kirja. UK: Packt Publishing

Chauhan, V. (2014). Smoke Testing. Haettu 17.9.2020 osoitteesta https://www.researchgate.net/profile/Vinod_Chauhan3/publication/284305832_Smoke_Testing/links/5651514e08aeafc2aab77fe6/Smoke-Testing.pdf

Cucumber. (2019). Gherkin Reference. Haettu 6.10.2020 osoitteesta https://cucumber.io/docs/gherkin/reference/

Gunawan, A. (2019). KeywordRecorder for Robot Framework. Haettu 21.8.2020 osoitteesta http://jirae.petra.ac.id/index.php/jirae/article/view/19370/18855

Suomen Hammaslääkäriliitto. (2016). Sähköinen resepti tulee koko terveydenhuoltoon 2017. Haettu 26.11.2020 osoitteesta https://www.hammaslaakariliitto.fi/fi/ajankohtaista/ajassa/sahkoinen-resepti-tulee-koko-terveydenhuoltoon-2017#.X7_MYc0zaUl

Jansen, B. (2019). Automated Testing of models of Cyber-Physical Systems. Haettu 16.9.2020 osoitteesta http://essay.utwente.nl/80034/1/Jansen_MA_EEMCS.pdf

Kanta. (n.d.). Mitä Kanta-palvelut ovat? Haettu 26.11.2020 osoitteesta https://www.kanta.fi/mita-kanta-palvelut-ovat

Kasurinen, J. (2013). Ohjelmistotestauksen käsikirja. E-Kirja. Jyväskylä: Docendo

Kansaneläkelaitos. (2016). Hammashoidon korvaukset. Haettu 25.11.2020 osoitteesta https://www.kela.fi/hammashoito

Keski-Suomen sairaanhoitopiiri. (2020). Asiakas- ja potilastietojärjestelmä. Haettu 18.10.2020 osoitteesta https://www.ksshp.fi/fi-FI/Sairaanhoitopiiri/Uusi_sairaala_projekti/ICTratkaisut/Asiakas_ja_potilastietojarjestelma

Keski-Suomen sairaanhoitopiiri. (2020). Kysymyksiä ja vastauksia uuteen asiakas- ja potilastietojärjestelmään liittyen. Haettu 18.10.2020 osoitteesta https://www.ksshp.fi/fi-FI/Sairaanhoitopiiri/Uusi_sairaala_projekti/ICTratkaisut/Asiakas_ja_potilastietojarjestelma/Kysymyksia_ja_vastauksia_uuteen_asiakas_(59684)

Klemetti, M. (2013). Mitä on devops. Haettu 18.10.2020 osoitteesta https://www.eficode.com/blogi/blogi/mita-on-devops

Laurikkala, J. (2019). Versionhallinta ja Git. Olio-Ohjelmoinnin perusteet II-kurssimateriaali. Tampereen yliopisto. Haettu 24.9.2020 osoitteesta https://www.sis.uta.fi/~oope/oope2/kevat-2019/luennot/luento03/oope2_2019_versionhallinta_ja_git.pdf

Nabil, M. (2017). The Automation Testing and Agile Haettu 18.10.2020 osoitteesta https://medium.com/@Moatazeldebsy/the-automation-testing-and-agile-7a8a8c983ed0

Opus Dental. (n.d.). About us. Haettu 25.11.2020 osoitteesta https://www.opusdental.com/about-us1/

Qentinel. (2010). Mitä on testausautomaatio, ja miten teet siitä liiketoimintasi vauhdittajan. Haettu 6.10.2020 osoitteesta https://value.qentinel.com/fi/mita-on-testausautomaatio#teknologia

Receptum. (n.d.). Helmi Potilastietojärjestelmä. Haettu 25.11.2020 osoiteesta https://www.receptum.fi/helmi/#hyodyt

Robocorp. (n.d.). Haettu 18.10.2020 osoitteesta https://hub.robocorp.com/knowledge-base/articles/what-is-robot-framework-about/

Robot Framework. (n.d.). Haettu 16.9.2020 osoitteesta https://robotframework.org/

Robot Framework, SeleniumLibrary. (2020). Haettu 6.10.2020 osoitteesta https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html

Robot Framework User Guide (2018). Robot Frameworks Foundation. Haettu 6.10.2020 Haettu osoitteesta http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

Salminen, L. (2020). Testausprosessit-moduulin verkkoaineisto. 01 Luento 1, Moodle. Hämeen ammattikorkeakoulu. Haettu 14.9.2020 osoitteesta https://learn.hamk.fi

Salminen, L. (2020). Testausprosessit-moduulin verkkoaineisto. 05 Robot Framework, Moodle. Hämeen ammattikorkeakoulu. Haettu 15.9.2020 osoitteesta https://learn.hamk.fi

Salminen, L. (2020). Testausprosessit-moduulin verkkoaineisto. 07 Systeemi hyväksyntätestaus bdd, Moodle. Hämeen ammattikorkeakoulu. Haettu 15.9.2020 osoitteesta https://learn.hamk.fi

Salminen, L. (2020). Testausprosessit-moduulin verkkoaineisto. 09 Automatisoitu Testaus Ja Selenium Web-driver, Moodle. Hämeen ammattikorkeakoulu. Haettu 6.10.2020 osoitteesta https://learn.hamk.fi

Salminen, L. (2020). Tietokannat-moduulin verkkoaineisto. 01 Johdatus tietokantoihin, Moodle. Hämeen ammattikorkeakoulu. Haettu 18.10.2020 osoitteesta https://learn.hamk.fi

Smart education. (n.d.). Testauksen tasot. Haettu 6.10.2020 osoitteesta http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus/testauksen-tasot

Test Guild. (2016). Robot Framework Creator Pekka Klärck (Podcast) Haettu 6.10.2020 osoitteesta https://testguild.com/episode-91-the-robot-framework/

Tietosuojavaltuutetun toimisto. (n.d.). Mikä on henkilötieto. Haettu 18.10.2020 osoitteesta https://tietosuoja.fi/mika-on-henkilotieto

Tietosuojavaltuutetun toimisto. (n.d.). Rekisteröidyn oikeudet. Haettu 20.10.2020 osoitteesta https://tietosuoja.fi/rekisteroidyn-oikeudet

Tietosuojavaltuutetun toimisto. (n.d.). Tietosuoja turvaa oikeutesi henkilötietoja käsiteltäessä. Haettu 18.10.2020 osoitteesta https://tietosuoja.fi/tietosuoja

Tilander, A. (18.11.2016). Sähköinen resepti ja potilastiedon arkisto käyttöön 2017. *Suomen Hammaslääkärilehti.* https://www.hammaslaakarilehti.fi/fi/uutinen/sahkoinen-resepti-ja-potilastiedon-arkisto-kayttoon-2017

Tilander, A. (21.8.2017). Potilastiedon arkisto käyttöön suun terveydenhuollossa. *Suomen Hammaslääkärilehti.* https://www.hammaslaakarilehti.fi/fi/uutinen/potilastiedon-arkisto-kayttoon-suun-terveydenhuollossa

University of Helsinki, Department of Computer Science. (n.d.). Tietokantojen perusteet. Haettu 24.9.2020 osoitteesta https://tietokantojen-perusteet.github.io/

ValaGroup. (n.d). Testiautomaatio-opas. Haettu 20.9.2020 osoitteesta https://www.valagroup.com/wp-content/uploads/pdf/Testiautomaatio-opas-2020.pdf?utm_term=testiautomaatio&utm_campaign=Testiautomaatio-opas&utm_source=adwords&utm_medium=ppc&hsa_acc=9088882292&hsa_cam=10061361721&hsa_grp=100900344453&hsa_ad=435280672285&hsa_src=g&hsa_tgt=kwd-364989503405&hsa_kw=testiautomaatio&hsa_mt=b&hsa_net=adwords&hsa_ver=3&gclid=EAIaIQobChMI5ZrhyPb36wIVDdOyCh1E9wG3EAAYAyAAEgKTO_D_BwE

Valvira. (n.d). Sosiaali- ja terveydenhuollon tietojärjestelmät. Haettu 18.10.2020 osoitteesta https://www.valvira.fi/terveydenhuolto/sosiaali-ja-terveydenhuollon-tietojarjestelmat