jamk.fi

Hyvinvointianalyysi-järjestelmän testauksen automatisoinnin suunnittelu

Firstbeat Technologies Oy

Sami Heikkinen

Opinnäytetyö
Heinäkuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), hyvinvointiteknologian tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Heikkinen, Sami	Opinnäytetyö, AMK	Heinäkuu 2017
	Sivumäärä 46	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Hyvinvointianalyysi-järjestelmän testauksen automatisoinnin suunnittelu

Tutkinto-ohjelma
Hyvinvointiteknologia

Työn ohjaaja(t)
Ström Markku, Siistonen Matti

Toimeksiantaja(t)
Firstbeat Technologies Oy

Tiivistelmä

Firstbeat Technologies Oy on jyväskyläläinen ohjelmistojen kehitykseen sekä hyvinvointiin keskittyvä yritys. Yksi yrityksen tärkeimmistä tuotteista on hyvinvointia mittaava Firstbeat Hyvinvointianalyysi. Kyseisen mittauksen toiminnasta vastaa Hyvinvointianalyysi-järjestelmä, joka koostuu neljästä asiakasohjelmasta. Ohjelmia kehitetään jatkuvasti yhä laajemmiksi toiminnallisuuksiltaan, ja tämä vaatii jatkuvaa ohjelmistotestausta. Hyvinvointianalyysi-järjestelmän testaus on täysin manuaalisen testauksen varassa, mikä alati kasvavan ohjelmiston myötä hankaloituu jatkuvasti.

Tutkimuksessa suunniteltiin kuinka testauksen automatisointia tulisi lähteä toteuttamaan käytännössä. Automatisoinnin tueksi kerättiin laaja katsaus Hyvinvointianalyysi-järjestelmän asiakasohjelmien toiminnasta sekä niiden testauksessa huomioitavista tekijöistä. Tutkimuksessa valittiin automatisoinnin käytännöntoteutuksessa hyödynnettävä automatisointityökalu sekä automatisoinnin vaatima fyysinen laitteisto. Suunnitelmissa pohdittiin myös mille ohjelmistojen osille rasitustestauksen automatisointia tulisi toteuttaa.

Tutkimuksen tuloksena rakentui automatisointiohjelmiston kolmitasoinen rakenne, jota voidaan hyödyntää automatisoinnin toteutuksessa. Automatisointiin valittiin työkaluksi Robot Framework ja sille luotu Selenium2Library -verkkokirjasto. Tutkimuksessa tuodaan myös esille esimerkkitapauksia eri asiakasohjelmiin toteutettavista automaatiotesteistä sekä kuinka rasitustestausta tulisi toteuttaa ohjelmistojen tuotantopalvelimelle. Tutkimuksen pohjalta automatisoinnin vaatimat fyysiset laitteistot ovat kaksikymmentä Bodyguard 2 -mittalaitetta, joista puolet sisältävät dataa ja puolet ovat tyhjiä. Lisäksi testaus vaatii Windows ja Mac kannettavat tietokoneet, sekä USB-hubeja joiden avulla Bodyguard 2 -mittalaitteet ovat yhdistettävissä tietokoneisiin.

Avainsanat (asiasanat)

Hyvinvointianalyysi-järjestelmä, asiakasohjelma, testauksen automatisointi, page-objekti, avainsana

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Heikkinen, Sami	Bachelor's thesis	July 2017
		Language of publication: Finnish
	Number of pages: 46	Permission for web publication: x

Title of publication

Designing Firstbeat Lifestyle Assessment system's testing automation

Degree programme

Degree programme in Wellness Technology

Supervisor(s)

Ström Markku, Siistonen Matti

Assigned by

Firstbeat Technologies Oy

Abstract

Firstbeat Technologies Oy is a software and wellness technology company from Jyväskylä. One of their biggest consumer products is the Firstbeat Lifestyle Assessment which measures the human wellbeing. Firstbeat Lifestyle Assessment requires the system to work which consists four clients. Firstbeat develop all four clients all the time and it requires continuous software testing. Firstbeat Lifestyle Assessment system's testing is based entirely on manual testing which gets more and more difficult when the system is advanced.

The study included designing how to start implementing the test automation. At the beginning there was needed overview how all the clients work and what is needed to take into consideration on testing. For the automation implement in the study was selected the automation tool and the hardware which was required. Design part also included speculation about where should the stress testing focus.

The result of this study was the three-level structure of the automation software which can be used when the automation will be implemented to the system. Robot Framework and Selenium2Library was selected to be the tool for the automation implement. In the study there was examples for different test scenarios in the automation system. It was also decided that stress tests should be focused to the production server. According to the study the automation required twenty Bodyguard 2 -measuring device, half of which contains data and half of which doesn't. There was also needed two laptops which one is Windows and one is Mac, and USB-hubs which can be used to connect Bodyguard 2 -measuring devices to laptops.

Keywords/tags (subjects)

Firstbeat Lifestyle Assessment system, client, test automation, page-object, keyword

Miscellaneous

Sisältö

Kuviot

1 Johdanto

Jo vuonna 2004 ohjelmistotuotteiden myynti saavutti Yhdysvalloissa 180 miljardia dollaria vuodessa ja raporttien mukaan lopputuotteiden sisältämät bugit eli viat maksavat Yhdysvaltojen taloudelle 59,5 miljardia dollaria vuodessa. Tämän myötä ohjelmistotestausta ja testaustyökaluja on kehitetty jatkuvasti eteenpäin, jotta yritysten taloudelliset menetykset olisivat mahdollisimman pienet, sekä asiakastyytyväisyys pysyisi mahdollisimman korkealla. (Li, K. & Wu, M. 2004. 2-3)

Ohjelmistotestauksen automatisoinnin kehittyminen on nykypäivää. Automatisoinnissa hyödynnettävät työkalut ovat jatkuvasti kasvava bisnes, sillä ohjelmistojen markkinat ovat alati kasvamassa. Yritykset panostavat yhä enemmän ja enemmän ohjelmistojen laatuun, sillä tilastojen kuten edellä mainitun raportin mukaan ohjelmistoviat ovat todella suuri kuluerä yritykselle ja sitä kautta koko maan taloudelle. Ohjelmistojen jatkuva kehitys vaatii jatkuvaa testausta, joka ilman testauksen automatisointia joudutaan suorittamaan manuaalisesti.

Manuaalisen testauksen etuna on suorittajan eli ihmisen oma näkemys ja luovuus ohjelmiston toimintojen ongelmien löytämisessä. Ohjelmistot kuitenkin sisältävät paljon toimintoja, joissa ei yksinkertaisesti ole useita eri ongelmia aiheuttavia poikkeuskäsittelyitä. Yksittäisenä toimintoja tällaiset ovat helppo testata manuaalisesti, mutta ohjelmistojen kasvaessa manuaalisesta testauksesta aiheutuvat kustannukset yksinkertaisten testien suorittamisessa kasvavat huomattavasti ja näiden testien suorittamiseen kuluva aika on pois monimutkaisempien ongelmien löytämisestä. Yksinkertaisten testien toistuva manuaalinen testaus myös kuormittaa ihmistä, sillä itseään toistava työ ei ole palkitsevaa eikä kehittävää.

Firstbeat Technologies Oy on suurelta osin ohjelmistoja kehittävä yritys. Jatkuva kasvu Suomessa ja kansainvälisesti vaatii myös jatkuvaa kehitystä yrityksen tuotteissa. Ohjelmistokehityksessä tämä tarkoittaa ohjelmistojen eli tuotteiden luotettavuutta sekä uusien toimintojen ja ominaisuuksien jatkuvaa kehitystä. Luotettavuus olemassa oleville ja uusille tuotteille varmistetaan ohjelmistotestauksen kautta. Testausprosessia pyritäänkin jatkuvasti parantamaan kehittämällä testausprosessia sekä testauksen automatisointia ohjelmistoihin.

Tässä opinnäytetyössä suunnittelin Firstbeat Technologies Oy:n Hyvinvointianalyysi-järjestelmälle testauksen automatisoinnin pohjan. Automatisoinnin runkoa varten suunnittelin mitä testauksen automatisoinnin työkaluja hyödynnetään, kuinka automatisointiohjelmisto rakentuu, millaisia testitapauksia olisi syytä toteuttaa sekä mitä fyysisiä laitteistoja käytännössä tarvitaan. Opinnäytetyöni tavoitteena oli tehdä selkeä selvitys, jonka pohjalta automatisointi olisi helppo lähteä käytännössä toteuttamaan.

2 Firstbeat Technologies Oy

2.1 Yritys

Firstbeat on vuonna 2002 Jyväskylässä perustettu yritys, joka keskittyy sykereaktioiden ja sykevaihtelun mittaukseen ilman laboratorio -laitteistoa tarkoitetun teknologian kehittämiseen. Yrityksen teknologian taustalla on osaamista fysiologian, matemaattisen mallintamisen sekä käyttäytymistieteiden aloilta. Idea Firstbeatin perustamiseen on lähtöisin Kilpa- ja huippu-urheilun tutkimuskeskuksessa (KIHU) 1990-luvulla tehdyistä urheilijoiden ylikunnon tutkimuksista. Jyväskylän yliopistossa käynnistyi 2000-luvun alussa tutkimus, jonka tarkoituksena oli selvittää mahdollisuuksia hyödyntää urheilijoiden mittaamiseen kehitettyä teknologiaa työntekijöiden stressin ja palautumisen mittaamiseen. Tutkimuksissa todettiin, että tavallisten ihmisten kehon toimintaa ja stressiä on mahdollista mitata epäsuorasti sykevaihteluun perustuvilla menetelmillä. Firstbeatin kehittämien sykereaktioiden ja sykevaihtelun mittaukset pohjautuvat suuriin mittaustietokantoihin, jotka on kehitetty laboratorio ja kenttämittausten avulla. Teknologian tavoitteena on mallintaa kehon toimintaa mahdollisimman tarkasti fysiologian näkökulmasta. (Firstbeat Tarinamme N.d.)

Firstbeatin toiminta jakautuu kolmeen osa-alueeseen: hyvinvointi, huippu-urheilu sekä kuluttajatuotteet. Vuonna 2004 Suunto julkaisi ensimmäisen sykemittarin, joka sisälsi Firstbeatin kehittämää teknologiaa, joka mittasi mm. tarkan hapenkulutuksen, EPOC:n (Harjoituksen jälkeinen lepotason ylittävä hapenkulutuksen määrä) sekä harjoitusvaikutuksen. Nykyään on saatavilla jo yli 70 eri kuluttajatuotetta, joissa hyödynnetään Firstbeatin teknologiaa. Hyvinvointipuolella ensimmäinen versio Hyvinvointianalyysistä valmistui vuonna 2004. Hyvinvointianalyysiä kehitetään tänäkin päivänä

jatkuvasti eteenpäin erityisesti asiakkaiden kanssa yhteistyössä yhä enemmän palvelutuotteeksi eikä yksistään mittaus- ja tutkimusvälineeksi. Huippu-urheilun puolella ensimmäinen Firstbeatin oma tuote Firstbeat SPORTS julkaistiin 2006, jonka kehitystä tehdään myös jatkuvasti. Tuotetta käytetään maailmanlaajuisesti eri lajien ja harjoittelukeskuksien toimesta. (Firstbeat Tarinamme N.d.)

Firstbeat on saanut useita palkintoja ja mainintoja kuten vuonna 2004 Tasavallan presidentin INNOSUOMI-palkinnon sykeanalyysiteknologian kehittämisestä. Yritys valittiin myös vuonna 2004 vuoden teknologiahautomoyritykseksi. Vuonna 2005 Venture Cup valitsi Firstbeatin parhaiten menestyneeksi yliopistopohjaiseksi startup-yritykseksi. Firstbeat mainittiin myös Deloitte:n toimesta, sillä heidän selvityksessään he listasivat Firstbeatin yhdeksi nopeimmin kasvavista suomalaisista teknologiayrityksistä vuosina 2007-2009. (Firstbeat Tarinamme N.d.)

2.2 Hyvinvointianalyysi

2.2.1 Käsitteenä

Hyvinvointianalyysi (HVA) (ks. liite 1) on työkalu, jonka avulla seurataan päivittäisiä stressi- ja palautumisjaksoja. Analyysin raportista saadaan selville, esimerkiksi vuorokauden aikana tapahtuvaa palautumista, sekä yön aikana tapahtuvan palautumisen tehon. Analyysissä seurataan myös liikunnan tehoa ja määrää sekä analysoidaan liikunnan terveyshyötyjen suhdetta mahdollisiin riskeihin. Hyvinvointianalyysin teknologia mahdollistaa mittaamisen ilman laboratorio-olosuhteita töissä, urheillessa sekä vapaa-ajalla. Mittaaminen tapahtuu Bodyguard 2 -laitteella, joka kiinnitetään kahdella anturilla rintaan. (Firstbeat Tekniset tiedot N.d.)

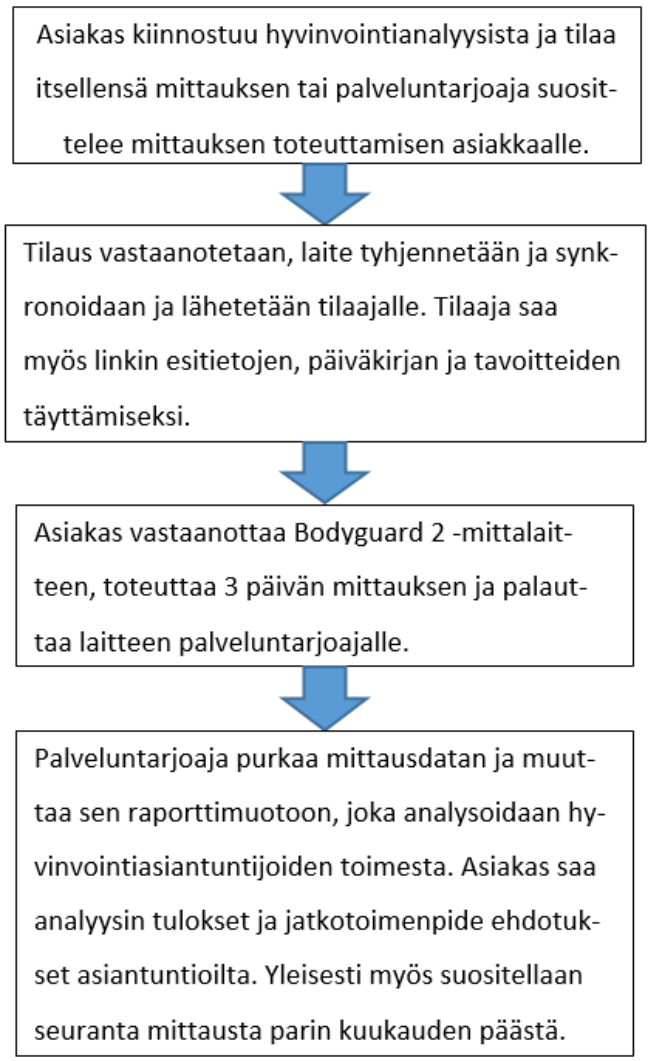
Kuvio 1. Bodyguard 2 -mittalaite (Firstbeat Tekniset tiedot N.d.)

Hyvinvointianalyysissä mitataan sykevälivaihtelua (heart rate variability = HRV). Käytännössä sykevälivaihtelu tarkoittaa R-R intervallien eli peräkkäisten sydämenlyöntien välisen ajan vaihtelun mittaamista. Sykevälivaihtelua mittaamalla saadaan tietoa palautumisesta ja rentoutumisesta, jolloin sykevälivaihtelun määrä kasvaa. Samalla mitataan myös kuormittumista, jota aiheutuu esimerkiksi stressistä ja liikunnasta, jolloin sykevälivaihtelun määrä vähenee. Mitattujen sykevälivaihtelu arvojen avulla Firstbeatin kehittämä tietokoneohjelma laskee ihmisen fysiologiset arvot, jotka Hyvinvointianalyysi muuttaa asiakkaille ymmärrettävään muotoon. (Stress and Recovery Analysis Method Based on 24-hour Heart Rate Variability 2014.)

2.2.2 Käytännön toiminta

Hyvinvointianalyysi on palvelu, jota pääasiallisesti myydään ulkoisille palveluntarjoajille. Firstbeat itse myös järjestää mittauksia ja palautteen antoja. Palveluntarjoajat myyvät palvelua eteenpäin heidän asiakkailleen. Hyvinvointianalyysin käyttöönotossa Bodyguard 2 -mittauslaite valmistellaan asiakkaalle tyhjentämällä se aikaisemmasta mittausdatasta, sekä synkronoimalla laitteen aikavyöhyke. Laitetta valmistellessa asiakas saa itsellensä linkin, jossa täytetään esitietoja, tavoitteita ja päiväkirjaa mittauksen aikana. Mittalaite ja sensorit lähetetään asiakkaalle, jonka jälkeen laitteen saavuttua asiakas tekee yleensä kolme päivää kestävän mittauksen. Mittauksen jälkeen laite palautetaan palveluntarjoajalle, ja palveluntarjoaja purkaa Bodyguard 2 -

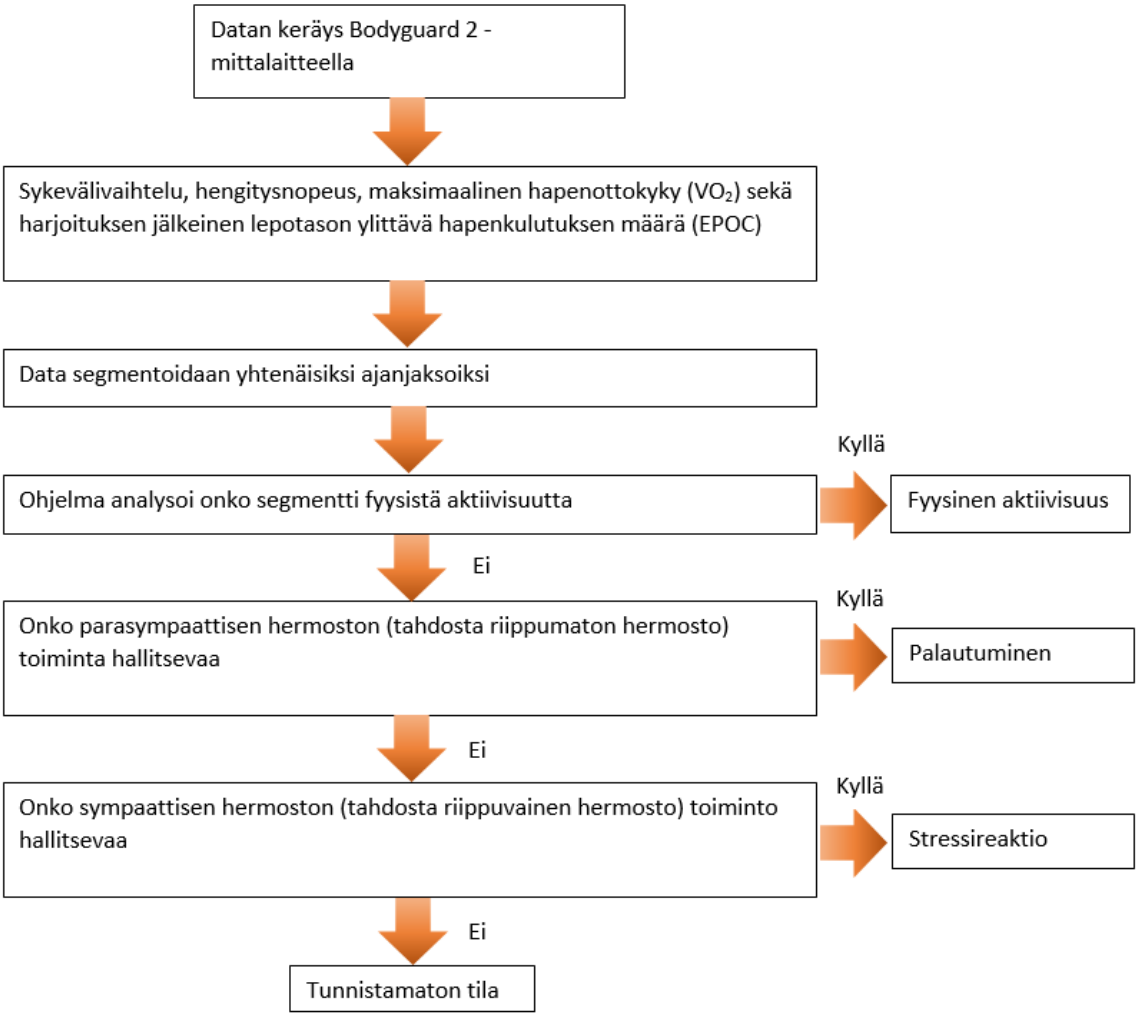
mittalaitteiden datan ja luo raportit. Raportit analysoidaan hyvinvointiasiantuntijoiden toimesta, jotka antavat myös palautetta ja jatkotoimenpide ehdotuksia analyysin tulosten mukaisesti. Asiakkaille suositellaan usein seurantamittausta, joka suoritetaan esimerkiksi 6 kuukauden päästä ensimmäisestä mittauksesta. Seurantamittauksen tarkoituksena on vertailla, kuinka tulokset ovat kehittyneet tai muuttuneet ensimmäisestä mittauksesta.



Kuvio 2. Hyvinvointianalyysin käyttöönotto asiakkaalle

Hyvinvointianalyysin mittausdata kulkee analyysiprosessin läpi missä datan analysointiin yhdistetään henkilön määrittelemät esitiedot: ikä, sukupuoli, pituus, paino sekä fyysisen aktiivisuuden taso. Näiden esitietojen pohjalta ohjelma arvioi henkilön

hengitysnopeuden, maksimisykkeen sekä maksimaalisen hapenottokyvyn. Nämä tiedot tarkentuvat mittausdatan avulla, jolloin arvot päivittyvät automaattisesti henkilön todellisiin yksilöllisiin arvoihin. Bodyguard 2 -mittalaitteessa on sisäisesti virhedatan suodatus, mikä mahdollistaa sen, ettei arvoja lasketa mahdollisista hetkellisistä mittausvirheistä. (Stress and Recovery Analysis Method Based on 24-hour Heart Rate Variability 2014.)



Kuvio 3. Mittausdatan analyysiprosessi (Stress and Recovery Analysis Method Based on 24-hour Heart Rate Variability 2014.)

Mittauksen analysoinnin avulla voidaan nähdä, jos mitattavalla henkilöllä tapahtuu sydämen lisälyöntejä tai muuta poikkeavuutta esimerkiksi sykkeessä tai suurissa

stressireaktioissa. Jos mittauksen analysoinnissa päädytään toteamaan tunnistamaton tila, voidaan mitattavalle asiakkaalle suositella menemistä lääkäriin jatkotutkimuksia varten. Lääkäriasemat voivat hyödyntää Hyvinvointianalyysin dataa, mutta sen pohjalta ei koskaan tehdä diagnoosia sairaudesta/sairauksista vaan tehdään viralliset lääketieteelliset tutkimukset. (Stress and Recovery Analysis Method Based on 24-hour Heart Rate Variability 2014.)

2.3 Hyvinvointianalyysin järjestelmä

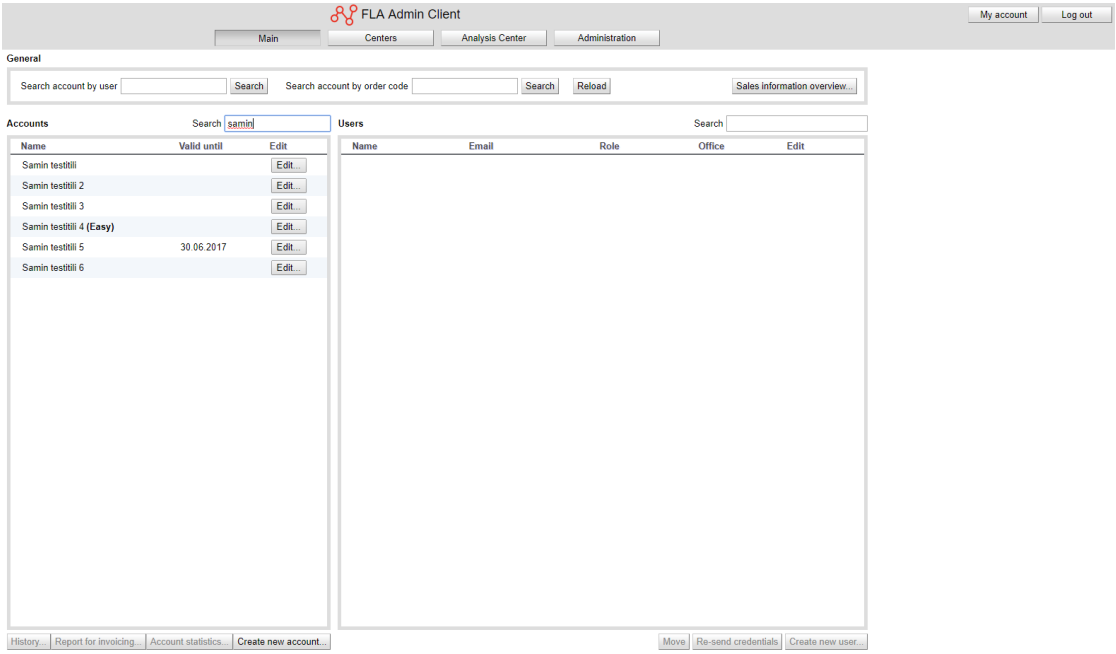
Hyvinvointianalyysi-järjestelmä koostuu neljästä asiakasohjelmasta, jotka ovat ohjelmoitu Java-ohjelmointikielellä ja käännetty Google Web Toolkitin (GWT) avulla JavaScript-ohjelmointikielelle. Järjestelmä koostuu Admin Client, Firstbeat Lifestyle Assessment (FLA), Logistics Center sekä Analysis Center asiakasohjelmista. Nämä neljä asiakasohjelmaa keskustelevat keskenään monien eri toimintojen avulla, sekä ovat yhteydessä Firstbeatin ylläpitämään palvelimeen. Järjestelmän pohjimmainen tarkoitus on olla käyttäjäkeskeinen sekä toiminnoiltaan monipuolinen ja tarpeellinen Hyvinvointianalyysien luontiin, raportointiin sekä tilien ja tilauksien hallintaan.

2.3.1 Admin Client

Admin Client on järjestelmän ylläpitoon tarkoitettu asiakasohjelma, joka on vain Firstbeatin sisäinen työkalu. Admin Client:ssä luodaan kaikkien järjestelmän verkkosivujen käyttäjätilit. Main-välilehdessä luotavat tilit (Accounts) ovat Firstbeat Lifestyle Assessment järjestelmän asiakastilejä. Asiakastileille luodaan erikseen käyttäjätilejä, joilla on 3 mahdollista roolia: Customer Admin, Pro tai End-user. Asiakastilien asetuksista voidaan määritellä asiakkaan tiedot, sallitut palvelut kuten Logistics Center:n ja Analysis Center:n käyttö, sallittujen asiantuntijoiden (Customer Admin ja Pro) määrä, Assessment- ja Center-krediittien määrä, joita vaaditaan Hyvinvointianalyysi-raporttien luontiin sekä tilauskoodeja, joilla voidaan tehdä Hyvinvointianalyysien tilaaminen. (Firstbeat Admin Client N.d.)

Customer Adminilla on eniten oikeuksia ja suurin määrä muokattavia ominaisuuksia FLA:ssa, kuten palvelun mukauttamista, jossa voidaan määritellä Hyvinvointianalyysiraporttiin palveluntarjoajan logo, raportin kansilehti sekä palvelun nimi. Customer

Admin pystyy myös seuraamaan asiakastilin muiden Customer Admineiden sekä asiantuntijoiden toimintoja tapahtumalokista sekä tilastoja raportin luonneista. Asiantuntijalla ei ole mahdollisuuksia palvelun mukauttamiseen ja tilastojen seurantaan, mutta muuten Customer Admin sekä Pro ovat oikeutettuja luomaan analyysejä ja raportteja sekä tilaamaan lisää krediittejä tilille. Loppukäyttäjät ovat asiakkaita joita voidaan luoda Admin Client:ssä sekä FLA:ssa analyysin luonnin yhteydessä. Loppukäyttäjät tallentuvat myös Admin Clientin asiakastilin alaisuuteen. (Firstbeat Admin Client N.d.)

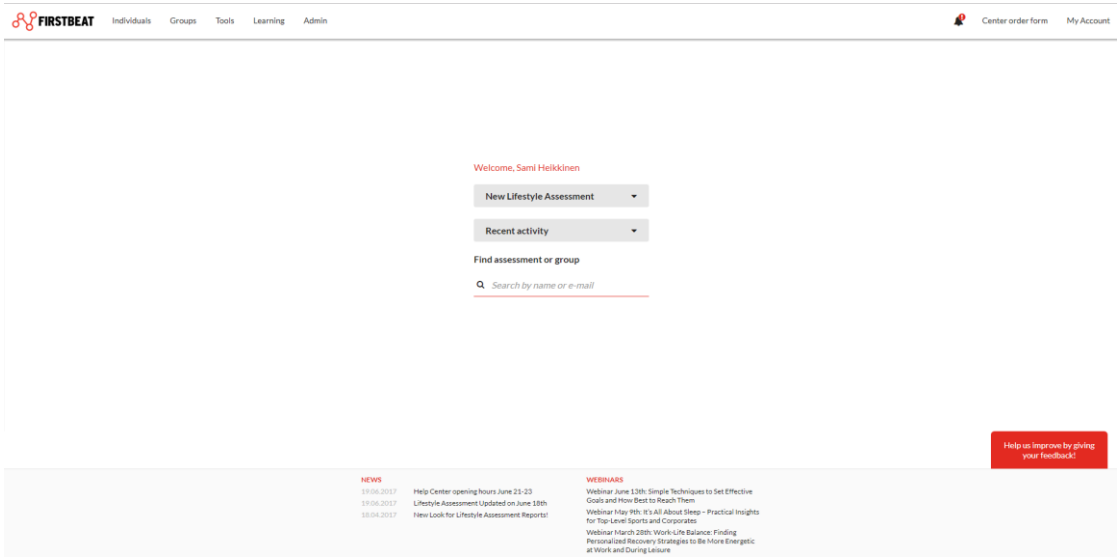


Kuvio 4. Admin Client etusivu (Firstbeat Admin Client N.d.)

Centers-välilehdellä luodaan ja hallitaan Logistics Center tilejä sekä tileille luotavia käyttäjiä. Logistics Center -käyttäjille on mahdollista asettaa rooliksi Admin tai Logistician. Adminilla on oikeus luoda uusia Logistician-käyttäjiä tai poistaa näitä, mutta muuten Adminilla ja Logistician-käyttäjällä on Logistics Center:ssä samat oikeudet tilausten hallintaan. Analysis Center-välilehdellä luodaan ja hallitaan Analysis Center-tilejä ja Administration-välilehdellä Admin Client:n tilejä. (Firstbeat Admin Client N.d.)

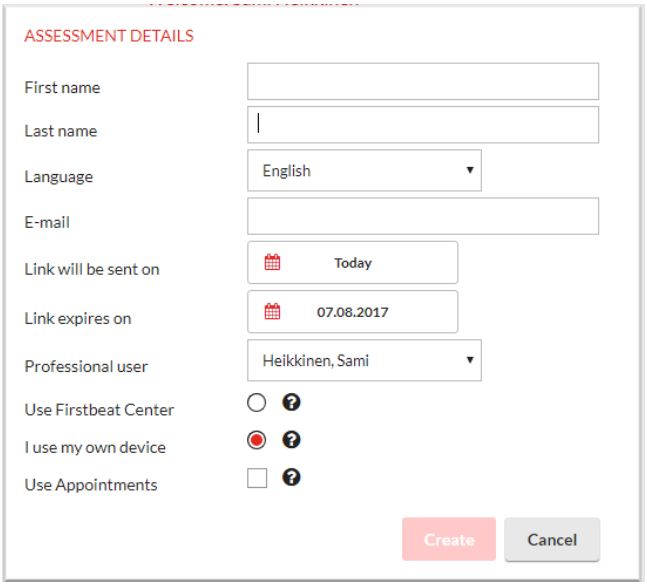
2.3.2 Firstbeat Lifestyle Assessment (FLA)

Firstbeat Lifestyle Assessment on yksilö- ja ryhmäanalyysien luontia varten suunniteltu asiakasohjelma. FLA:n tarkoitus on olla Firstbeatin hyvinvointiasiantuntijoiden sekä ulkoisten palveluntarjoajien käytössä, jotka ovat ostaneet sen itselleen. Yksilö- ja ryhmäanalyysejä luodessa voidaan valita luoko uudelle asiakkaalle vai jo olemassa olevalle. Uuden asiakkaan luonti luo loppukäyttäjän Admin Client:n, jolloin se tulee myös FLA:n listauksissa näkyviin. Olemassa olevalle asiakkaalle luodessa uusi analyysi näkyy FLA:ssa kyseisen asiakkaan alaisuudessa. (Firstbeat Lifestyle Assessment N.d.)



Kuvio 5. FLA etusivu (Firstbeat Lifestyle Assessment N.d.)

Yksilö- ja ryhmäanalyysien luonnissa voidaan määritellä kyseisen analyysin asetukset Admin Client:ssä määriteltyjen oikeuksien mukaisesti. Jos Admin Client:ssä on sallittu Logistics Centerin ja Analysis Centerin käyttö, analyysiä luodessa voidaan valita käytettäväksi Logistics ja Analysis Centeriä tai omia mittalaitteita, jos palveluntarjoajat ovat sellaiset itselleen ostaneet. Myös ajanvarauksen eli asiakkaalle järjestettävän loppupalautteen ja parannusehdotusten käyttö ja ajanvarauksen osoite voidaan määritellä heti analyysin luonnin yhteydessä. Asetukset ovat muutettavissa, missä vaiheessa analyysia tahansa. Analyysin luonnin jälkeen laite tai laitteet riippuen onko yksilö- vai ryhmäanalyysi, valmistellaan FLA:ssa tai Logistics Center:ssä. (Firstbeat Lifestyle Assessment N.d.)



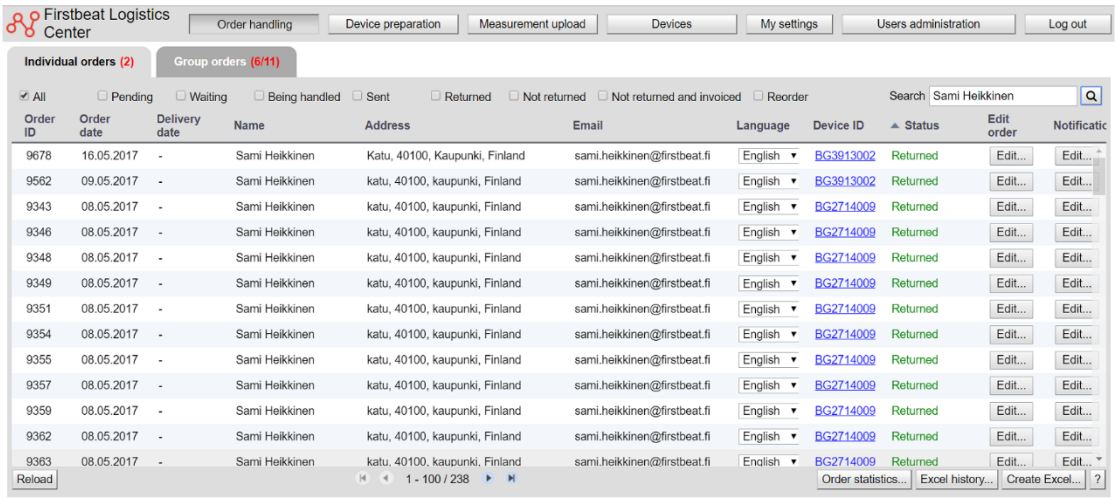
Kuvio 6. Analyysin luonnin aloitus (Firstbeat Lifestyle Assessment N.d.)

FLA sisältää erinäisiä työkaluja, jotka helpottavat krediittien seurantaa ja tilaamista, analyysien tapahtumien kuten valmistumisten ja Analysis Center ilmoitusten seurantaa, ryhmäraporttien luontia sekä Bodyguard 2 -mittalaitteiden valmistelua ja virran sekä laiteversion seurantaa. Työkalut yksinkertaistavat tärkeimpien ominaisuuksien hallintaa ja käyttöä, jottei käyttäjän tarvitsisi etsiä useammista paikoista edellä mainittuja ominaisuuksia. FLA sisältää myös pääsyn Firstbeatin järjestämiin kursseihin, jotka sisältävät opetusmateriaalia Firstbeatin asiakasohjelmien käyttöön sekä raporttien analysointiin ja jatkotoimenpide-ehdotuksien antamiseen sekä ymmärtämiseen. Kurssit ovat tilattavia ominaisuuksia, jotka aktivoidaan Admin Client:ssä. (Firstbeat Lifestyle Assessment N.d.)

2.3.3 Logistics Center

Logistics Center on asiakasohjelma, joka on vain logistiikkatyöntekijöiden käytössä. Sen avulla käsitellään Hyvinvointianalyyseihin liittyvät Bodyguard 2 -mittalaitteiden tilaukset eli laitteet valmistellaan, lähetetään, vastaanotetaan sekä data puretaan. Logistics Center:n avulla voidaan myös seurata ja muokata asiakastilien laitetilausten tilaa ja tietoja sekä lisätä huomioita. Jos asiakastilille annetaan oikeudet Logistics Center:n käyttöön Admin Client:n kautta, niin Center oikeudet määrätään yhdelle ha-

lutulle Logistics Center-tilille. Määritellyn Logistics Center-tilin käyttäjät näkevät asiakastilin kaikki Center-tilaukset ja hallitsee niitä. Tilan (status) avulla nähdään, onko tilaus odottamassa laitetta, laite lähetetty tai laite palautettu tilassa. Jos laite on lähetetty asiakkaalle, mutta sitä ei ole palautettu 37 päivän kuluessa lähetyksestä, Logistics Center:in tilaukseen tulee näkymään ilmoitus, että laitetta ei ole palautettu 37 päivään ja tilaajaa on laskutettu. Tämä aika huomioidaan kuitenkin tapauskohtaisesti, jos laite lähetetään esimerkiksi ulkomaille, jonne toimitusaika edestakaisin on normaalia pidempi. (Firstbeat Logistics Center N.d.)



Kuvio 7. Logistics Center tilausten hallinta sivu (Firstbeat Logistics Center N.d.)

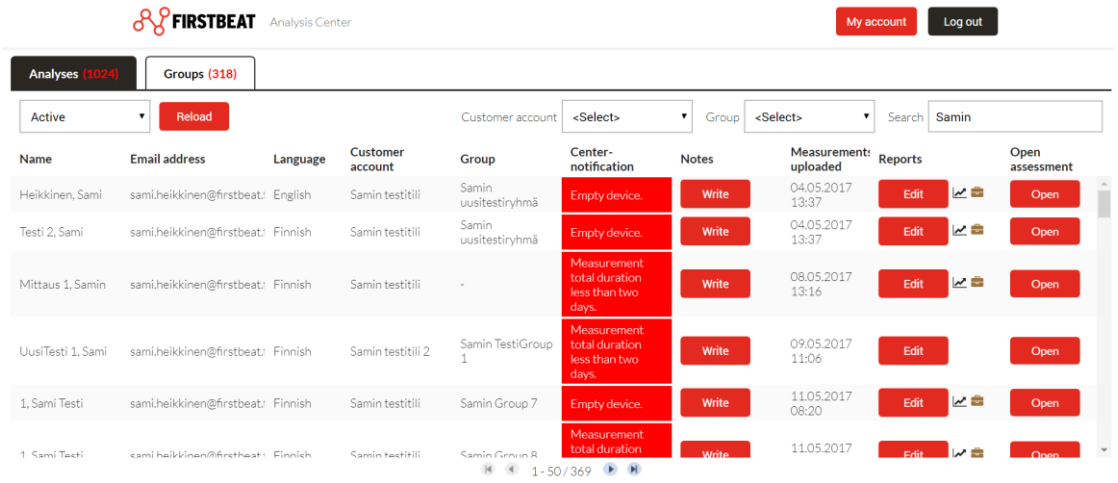
Laitteiden valmistelussa nähdään Bodyguard 2 -mittalaitteen yksilöllinen ID, laiteversio sekä virran määrä. Laitteelle määritellään myös aikavyöhyke, minkä avulla päiväkirjamerkinnät ovat luettavissa muualla kuin paikan päällä mittauspaikalla oikein. Purkaessa mittauksia nähdään myös Bodyguard 2 -mittalaitteen yksilöllinen ID, tilaukselle automaattisesti generoitu ID sekä mittausdatan ajallinen kesto laitteessa. Logistics Center:ssä on myös mahdollista seurata Bodyguard 2 -mittalaitteiden tilaa, joita on käytetty kyseisessä Centerissä eli onko laite varastossa, jossain tilauksessa tai onko laite määrätty poistettavaksi vian takia. (Firstbeat Logistics Center N.d.)

2.3.4 Analysis Center

Analysis Center on asiakasohjelma, joka on vain Firstbeatin sisäisessä käytössä oleva työkalu. Sen avulla on nähtävissä kaikkien asiakastilien, joille on määritelty Logistics

Center ja Analysis Center oikeudet, Center tilaukset. Analysis Center:n käyttötarkoitus on luoda raportteja Center yksilö- ja ryhmäanalyyseihin valmiiksi palveluntarjoajille. Analysis Centerissä on omat välilehdet yksilö- ja ryhmäanalyysien hallintaan, sekä siellä on nähtävissä aktiiviset sekä jo suljetut analyysit. Yksilöpuolella jokaisen analyysin kohdalla on nähtävissä analyysin asiakkaan nimi, sähköpostiosoite, kieli, mille asiakastilille tilaus kuuluu, ryhmä jos analyysi on osa ryhmätilausta, Center-ilmoitus eli onko laite tyhjä tai onko siinä esimerkiksi 3 päivän mittaus sekä milloin mittaukset on purettu. Yksilöpuolella voidaan lisätä huomioita analyyseihin, jotka ovat joko näkyvissä ainoastaan Analysis Center:ssä tai ne voidaan lähettää myös näkyviin FLA:han sekä kyseisen analyysin asiantuntijan sähköpostiin. Analyysi voidaan myös avata, jolloin aukeaa sama näkymä kuin FLA:n analyysin hallinnassa. (Firstbeat Analysis Center N.d.)

Ryhmäanalyysien välilehdellä on nähtävissä ryhmäanalyysin asiakastili, ryhmän nimi, tehtyjen tilauksien määrä, palautuneiden mittalaitteiden määrä, kuinka monta raporttia on valmiina sekä raporttien valmistumisten viimeinen palautuspäivä. Ryhmäpuolella kartoitus voidaan myös avata samaan näkymään kuin, mitä se aukeaa FLA:ssa. (Firstbeat Analysis Center N.d.)



Kuvio 8. Analysis Center yksilöanalyysien välilehti (Firstbeat Analysis Center N.d.)

3 Ohjelmistotestaus

Ohjelmistotestaus on prosessi, jonka tarkoituksena on varmistaa ohjelmiston toiminnan vastaavuus siltä vaadittaviin ominaisuuksiin. Yksinkertaistettuna ohjelmistotestauksessa testataan tuotteen toimivuutta verraten siihen, miten ohjelmistotuotteen halutaan toimivan. Testauksen tavoitteena on löytää ohjelmistosta ne kohdat, missä tuote ei toimi suunnittelussa määritellyllä tavalla. Ohjelmistotestauksen toteutus on hyvin vaihtelevaa riippuen ohjelmistotalojen asiakaskunnista ja toimintatavoista. Testaaja voi suorittaa testauksen manuaalisesti, itse joutua kirjoittamaan koodia tai erimerkiksi suunnittelemaan ja toteuttamaan haastatteluja kohdeyleisölle. Myös testauksen kohde vaihtelee esimerkiksi tuotteen käytettävyyden testauksesta graafisten elementtien toiminnan testaukseen. (Kasurinen, J. P. 2013, 10-11)

3.1 Testausprosessi

Testausprosessi voidaan toteuttaa monin eri tavoin riippuen projektista, sekä ohjelmistotalon tavoista. Yleisiä testausprosessin toteutustapoja ovat: vesiputous-, v-, RUP- sekä Scrum-malli. Jokaisen toteutustavan perusperiaatteena on vaatimusten määrittely, testauksen suunnittelu, toteutuksen suunnittelu, itse testausprosessi sekä testauksen jatkuva ylläpitäminen laadun takaamiseksi. Toteutustavan valinta suoritetaan projektin alkuvaiheessa ja tätä prosessia noudatetaan koko projektin ajan, eikä tapaa muuteta kesken projektin, sillä tämä vaatisi huomattavaa resurssien lisäkäyttöä. Ohjelmistotestausvaihe kestää ohjelmistokehityksessä niin pitkään, kunnes tuote täyttää asiakastarpeet, eikä sisällä merkittäviä virheitä. Kun ohjelmisto saadaan tuotantoon, testaus siirtyy ylläpito vaiheeseen. (Kasurinen, J. P. 2013, 12-28)

3.2 Staattinen testaus

Staattisen testaamisen tarkoitus on testata järjestelmää jo ohjelman kehitysvaiheessa, jolloin järjestelmä ei vielä varsinaisesti edes toteuta mitään toimintoa. Kyseisessä testauksessa testataan järjestelmää, käyttäen esimerkiksi koodianalysaattoreita tai arvioiden koodia ja arkkitehtuurisuunnittelua. Tämän tarkoituksena on löytää järjestelmästä huomattavat ongelmat jo alkutoteutuksen aikana, sillä staattisen testauksen vaiheessa huomatut ongelmat, ovat huomattavasti halvempia korjata.

Tällaisia ongelmia ovat esimerkiksi syntaksivirheet sekä järjestelmän logiikka virheet. Syntaksi virheet tarkoittavat kirjoitusvirheitä järjestelmän koodissa. Logiikka virhe on esimerkiksi sellainen, missä ohjelmalle kooditasolla annetaan syöte, jonka jälkeen seurataan, toimiiko järjestelmä sisäisesti oikein sekä palauttaako se oikean tuloksen syötteeseen nähden. (Kasurinen, J. P. 2013, 64-67)

3.3 Dynaaminen testaus

Dynaamisessa testauksessa järjestelmää käytetään konkreettisesti. Testauksen tarkoitus on antaa järjestelmälle erilaisia syötteitä kuten käyttäjätunnusten syöttöä, napin painamista tai järjestelmän sulkemista ja varmistaa, että järjestelmän vasteet ovat oikeita syötteeseen nähden. Dynaamiseen testaukseen lasketaan suurin osa testausmenetelmistä kuten yksikkö-, järjestelmä- sekä regressiotestaus. Dynaaminen testaus aloitetaan heti kun järjestelmää voidaan edes osittain käyttää. Dynaamisella testauksella löydetyt viat ovat huomattavasti kalliimpia korjata, kuin staattisella testauksella löydetyt viat, mutta useita vikoja ei löydy ennen järjestelmän varsinaista koekäyttöä. (Kasurinen, J. P. 2013, 64-66)

3.4 Yksikkötestaus

Yksikkötestauksessa testataan järjestelmän yksittäisen metodin, funktion tai olion toimintaa järjestelmässä. Jokaisen yksittäisen toiminnon testaus tulisi tapahtua mahdollisimman monipuolisilla syötteillä, jotta olisi mahdollista poissulkea poikkeustiloista aiheutuvat viat. Yksikkötestaus on hyvä testauksen automatisoinnin kohde, sillä näiden ajaminen automaattisesti on nopeaa, sekä tarpeellista suorittaa usein, jotta nähdään aiheuttaako koodin muutokset vikoja aikaisemmin olemassa olleisiin toimintoihin. (Sommerville, I. 2016, 232-235)

Yksikkötestaus käytännössä tulisi tapahtua järjestelmän kehittäjän tai ohjelmoijan toimesta. Tätä testausta varten rakennetaan yleisesti funktioiden tai olioiden kutsuja, jotka ovat syötteitä, mihin järjestelmän täytyy antaa oikea vaste tai tehdä oikeanlainen toiminto. Tämän etu on sen helppo seurattavuus, jolloin kehittäjä tai ohjelmoija näkee helposti, onko testattavassa komponentissa vikaa. Yksikkötestauksen ongelma

on yleisesti siinä, ettei järjestelmän yksittäinen komponentti toteuta mitään itsenäisesti, vaan se on vuorovaikutuksessa useiden eri komponenttien kanssa. (Kasurinen, J. P. 2013, 50-53)

3.5 Integrointitestaus

Integrointitestauksessa järjestelmää testataan siten, että yksikkötestatut komponentit sovitetaan pikkuhiljaa kokonaisuudeksi ja jokaisen komponentin liittämisen jälkeen testataan yhdistettyjen komponenttien toimivuus. Vaikka komponentit toimivat yksikkötestauksessa, niin tämän testausvaiheen tavoitteena on löytää viat, jotka syntyvät komponentteja yhdisteltäessä. Integrointitestauksessa voidaan joutua luomaan komponentteja, jotka eivät ole toiminnollisuuksiltaan valmiita, mutta ovat välttämättömiä esimerkiksi järjestelmän käynnistämistä varten. Tällaisia komponentteja sanotaan tyngiksi. Integrointitestauksen suurin kuluerä syntyy yleisesti tynkien luonnista ja ylläpidosta, sillä ne voivat jäädä vain sijaiskomponenteiksi, joita ei ikinä hyödynnetä lopullisessa järjestelmässä. (Kasurinen, J. P. 2013, 54-55)

3.6 Järjestelmätestaus

Järjestelmätestauksessa testataan koko järjestelmää, sen jälkeen, kun järjestelmän komponentit ovat kulkenut yksikkö- ja integrointitestauksen läpi. Järjestelmä ei tässä vaiheessa enää voi sisältää sijaiskomponentteja, jotka eivät tule lopulliseen tuotteeseen. Järjestelmätestaus toteutetaan testausta varten luodussa ympäristössä. Järjestelmätestausvaiheessa tapahtuu usein vielä muutoksia, sillä lopullinen järjestelmän toiminta ja puutteet nähdään parhaiten. Tästä johtuen järjestelmätestaus vaiheessa tapahtuu myös integrointitestausta. (Kasurinen, J. P. 2013, 56-57)

Järjestelmätestauksen yksi tärkeimmistä tehtävistä on testata komponenttien toimintaa kokonaisuutena. Usein komponenttien ohjelmointia tekee usea eri ohjelmoija tai ohjelmoijatiimi, jolloin voi syntyä eroavaisuuksia, vaikka suunnitelmat ja kommunikaatio näiden yksilöiden ja tiimien välillä olisikin ollut kunnossa. Järjestelmässä komponenttien on sovittava yhteen, vastattava oikein annettuihin syötteisiin sekä keskusteltava keskenään. (Sommerville, I. 2016, 240-242)

Järjestelmätestausta varten usein suunnitellaan tapauksia ja skenaarioita, joita mietitään järjestelmän mahdollisimman monipuolisen testauksen sekä asiakkaan/järjestelmän käyttäjän näkökulmasta. Esimerkiksi jos luodaan käyttäjätunnukset sivustolle, on tarkistettava, että tunnukset tallentuvat sivuston palvelimelle, jotta ne toimivat sivustolle kirjautuessa. Jatkotestaus tähän on esimerkiksi käyttäjätunnuksen salasanan vaihto, jolloin tarkistetaan, että vaihdettu salasana tallentuu palvelimelle oikein ja toimii sivustolla. Järjestelmätestauksen automatisointi on huomattavasti vaikeampaa verraten yksikkötestauksen automatisointiin, sillä yksikkötestauksen automatisoinnissa tiedetään komponenttien antamat vasteet, kun taas järjestelmän antamat vasteet voivat olla niin suuria tai vaikea ennustaa kaikkia mahdollisia vaihtoehtoja. (Sommerville, I. 2016, 240-242)

Hyväksymistestaus on lähes sama kuin järjestelmätestaus, mutta hyväksymistestauksessa testaus yleensä siirretään sen lopulliseen ympäristöön, eikä siihen tehdä enää suuria muutoksia. Hyväksymistestauksessa painotus on enemmän sen toiminnan kokonaisuuden testauksessa, eikä järjestelmätestauksessa tapahtuvaa yksittäisten komponenttien vikoja etsitä, eikä integrointitestauksen tarvetta juurikaan ole. Tämän testauksen tarkoitus on osoittaa, että järjestelmä täyttää vaatimukset ja on valmis loppuasiakkaan käyttöön. (Kasurinen, J. P. 2013, 57-58)

3.7 Regressiotestaus

Regressiotestauksessa testataan järjestelmän olemassa olevia toimintoja, järjestelmään tehtyjen muutosten jälkeen. Tämän testauksen tarkoitus ei ole löytää uusien toimintojen vikoja, vaan nimenomaan varmistamaan jo olemassa olevien osien toiminta. Usein jo pienetkin muutokset yksittäisessä järjestelmän komponentissa voi aiheuttaa ongelmia useissa, jopa erittäin kaukana olevissa komponenteissa ja tätä kautta koko järjestelmässä. Regressiotestauksen tulisi olla erittäin suuressa roolissa, jokaisessa ohjelmistoprojektissa, jotta testauksessa ei jäisi huomaamatta jo aikaisemmin testattuihin komponentteihin syntyneitä uusia tai jo aikaisemmin korjattuja mutta uusiutuneita vikoja. Tehokkaan testauksen kannalta regressiotestaus prosessi on ehdoton automatisoida. (Ammann P. & Offutt J. 2008, 215-217)

3.8 Kuormitus- ja suorituskykytestaus

Kuormitustestauksen tarkoitus on testata järjestelmää sille suunnitellun toimintakapasiteetin äärirajoilla. Tällaisella testauksella nähdään, onko järjestelmässä pullonkauloja eli esimerkiksi, kuinka palvelimien välinen yhteys kestää suuria kuormia, kuinka nopeasti järjestelmä suorittaa halutut toiminnot suurilla määrillä dataa tai mitä tapahtuu, kun järjestelmän muisti on kuormitettu äärirajoille. Kuormitustestauksella voidaan myös ylirasittaa järjestelmää tarkoituksellisesti, eli rasitus testata sitä. Tällöin voidaan nähdä, kuinka järjestelmä toimii tällaisissa tilanteissa, eli tapahtuuko järjestelmän lukkiutumista tai muistivuotoja, jotka voivat altistaa järjestelmän palvelunestohyökkäyksille. Kuormitus- ja stressitestaus on yleisesti automatisoitava prosessi, sillä manuaalisesti testattuna se on työlästä ja vaikea toteuttaa suurissa määrin. (Kasurinen, J. P. 2013, 71-72)

Suorituskykytestauksessa testataan järjestelmää sille suunniteltujen toimintarajojen sisällä. Tällä testataan, että järjestelmä toimii halutulla tavalla ja nopeudella odotetulla käyttäjä- ja datanprosessointi määrällä. Tässä testauksessa järjestelmää ei ole tarkoitus ylikuormittaa. (Kasurinen, J. P. 2013, 72)

4 Testauksen automatisointi

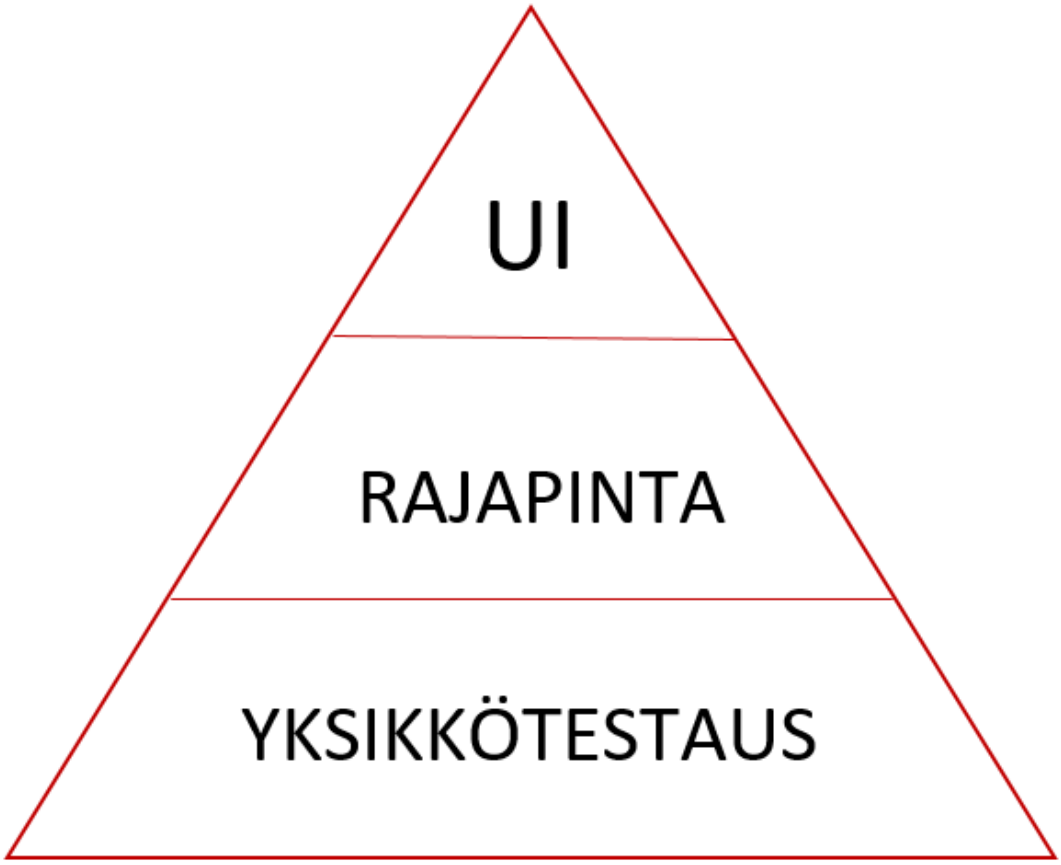
Tässä luvussa käsitellään testauksen automatisoinnin hyötyjä, haittoja sekä tarpeellisuutta. Tarpeellisuuden näkökulmassa pohditaan sen vaikutusta manuaalitestauksen tehostamiseksi. Hyötyjen ja haittojen näkökulmasta pohditaan sen taloudellisiin ominaisuuksiin. Testauksen automatisoinnissa yleisesti testataan joko moduulien rajapintoja suoraan koodista tai käyttöliittymän testausta niille suunnitelluilla työkaluilla. Lisäksi käydään läpi työkaluja jotka suorittavat käyttöliittymätestausta, joiden pohjalle testauksen automatisointi toteutetaan käytännössä tässä kyseisessä työssä. (Kasurinen, J. P. 2013, 76-77)

4.1 Testauksen automatisoinnin tavoite

Testauksen automatisointi ei korvaa manuaalista testausta, vaan sen tavoite on olla toiminto joka täydentää sitä. Testausautomaation tarkoituksena on vähentää yksinkertaisten ja useasti toistettavien testitapausten suorittamista manuaalitestaajien toimesta. Automaatio testit voidaan toteuttaa esimerkiksi yön aikana, jolloin seuraavana aamuna testien tulokset voidaan käydä läpi ja tehdä tarvittavat korjaukset. (Kasurinen, J. P. 2013, 76-77)

Automatisointi on ehdottoman tärkeää erityisesti sellaisissa tapauksissa, missä ohjelmistosta käännetään usein uusi buildi eli testattava ohjelmisto, johon on lisätty ominaisuuksia ja vanhojen vikojen korjauksia. Uusi buildi vaatii jo ohjelmalle aikaisemmin suoritettujen perustoimintojen testauksen uudelleen, eli regressiotestauksen. Tässä tapauksessa regressiotestauksen automatisointi olisi järkevää, erityisesti jos buildeja tulee useita kertoja, sillä perustoimintojen manuaalinen testaaminen vie työaikaa uusien ominaisuuksien ja monimutkaisten testitapausten suorittamiselta. (Kasurinen, J. P. 2013, 76-77)

4.2 Automatisoinnin tasot



Kuvio 9. Testausautomaation tasot (Lent J. 2013.)

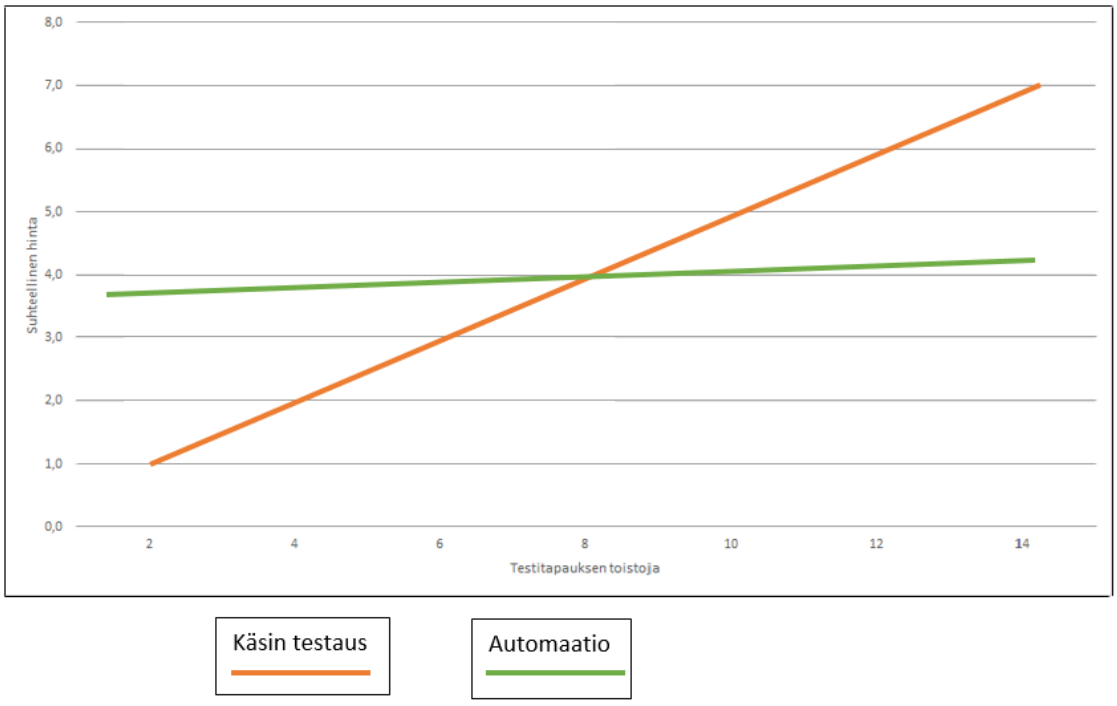
Testauksen automatisointi voidaan jakaa pyramidin tasoihin kuten kuviossa 9 on esitetty. Yksikkötestauksen tulisi olla pohjana kaikelle automatisoinnille, joten siksi se esitetään pyramidin pohjana suurimalla pinta-alalla. Tällaiset automatisoidut yksikkötestit tulisi kirjoittaa suoraan koodiin, jolloin vian paikantaminen olisi ohjelmoijalle itselleen helppoa, sillä manuaalisesti löydettynä kyseisen vian aiheuttama koodi voi sisältyä esimerkiksi 1000 riviin koodia, mutta koodiin sisälle kirjoitettu yksikkötesti kykenisi kohdistamaan vian jopa muutamalle riville. Yksikkötestit voidaan yleensä kirjoittaa samalla kielellä kuin itse ohjelmistoa kirjoitetaan, mikä helpottaa ohjelmoijien työtä. Rajapintatestauksessa automatisoidaan syötteen tai syötteiden lähettäminen rajapinnalle, ja vasteiden tarkastaminen, jotta ne ovat halutun kaltaisia. Tämän kaltainen automatisoitu testaus rajapinnalle on huomattavasti nopeampaa kuin samojen testien suorittaminen käyttöliittymässä. (Cohn M. 2009.)

UI:n eli käyttöliittymän automatisointi on pyramidin huippuna, sillä sen testaus tulisi suhteessa yksikkö- ja rajapintatestauksen automatisointiin nähden vähäistä. UI:n tes-

tausta varten kirjoitetaan koodia, joka suorittaa sille määritellyt testitapaukset ja seuraa toteutuvatko testit vai tapahtuuko testin aikana poikkeavuutta ohjelman toiminnassa. Näiden testien kirjoittaminen on aikaa vievää, kallista ja tällainen koodi on herkkä muutoksille, mikä tarkoittaa, että se vaatii ylläpitoa ja muutoksia. On kuitenkin myös paljon testitapauksia, jotka ovat viisainta suorittaa UI tasolla. (Cohn M. 2009.)

4.3 Automatisoinnin hyödyt

Testauksen automatisoinnilla ei tavoitella manuaalitestauksen poistamista, vaan yksinkertaisten ja useasti toistettavien regressiotestitapausten vähentämistä manuaalisesta työstä. Tutkimusten mukaan testauksen automatisointia olisi järkevä suunnitella, jos regressiotestejä suoritetaan vähintään 4-20 kertaa ohjelmistolle. Esimerkiksi jos ohjelmistolle suoritetaan täydellinen regressiotestaus aina ohjelmistoon kohdistuvan uuden projektin alussa, on järkevä suunnitella ja toteuttaa automaatiotestit kattavissa määrin. Automatisoinnilla on myös hyvä toteuttaa järjestelmälle rasitustestausta, millä varmistetaan palvelimen toiminta ohjelmiston käytön kasvaessa. (Kasurinen, J. P. 2013, 76-78)



Kuvio 10. Kustannuskäyrä manuaalinen testaus vs automatisoitu testaus (Kasurinen, J. P. 2013, 78)

Kuviossa 10. on esitetty manuaalisen testauksen kustannukset suhteessa automatisoituun testaukseen, kun katsotaan testitapauksen toistojen määrää. Lähtökohtaisesti manuaalinen testaus on edullisempaa, kun toistomäärät ovat hyvin vähäisiä, mutta tutkimuksien mukaan testauksen automatisointi olisi taloudellisesti järkevää, kun regressiotestitapauksia suoritetaan vähintään 8 kertaa. Manuaalisen testauksen kustannukset nousevat merkittävästi jokaisella regressiotestitapaus kerralla, mutta automatisoinnin kustannukset eivät nouse lähtötilanteesta enää merkittävästi riippumatta testauksen toistojen määrästä. (Kasurinen, J. P. 2013, 78-79)

Automatisoidun testauksen odotusarvona on ajan sekä resurssien säästäminen, standardien noudattaminen sekä tuotteiden laadun huomattava parantaminen. Automatisoinnissa on huomioitava, mitä järjestelmän toimintoja on automatisoitava, mutta taloudelliselta kannalta, mitä enemmän olemassa olevia järjestelmän toimintoja automatisoidaan laadukkaasti, sitä enemmän säästää testauksen työvoimaa järjestelmä- ja hyväksymistestaukseen. Laadukas automatisointi ohjelmiston arkkitehtuuriin nähden vähentää automatisoitujen testien ylläpitoa ja korjausta, sekä parantaa testien luotettavuutta siihen nähden, että vika on oikeasti järjestelmässä eikä testausautomaation koodissa. (Li, K. & Wu, M. 2004, 2-6)

4.4 Automatisoinnissa ilmenevät yleiset ongelmat

Testauksen automatisoinnissa ilmenee usein myös ongelmia, joita ei huomioida monipuolisesti. Automatisointia suunniteltaessa ja toteutettaessa usein nähdään siitä yritykselle aiheutuvat kulut, mutta ei ymmärretä tai osata laskea sen aiheuttamia säästöjä testauksen työmäärässä sekä testauksen laadun paranemisessa. Tähän liittyen tutkimusten mukaan, myös yrityksen taloudellinen panostus eli testauksen automatisointiin määrätty budjetti ei ole riittävää. Yleisesti budjetissa ei myöskään osata huomioida muita ylimääräisiä automatisoinnista aiheutuvia kustannuksia kuten ylläpitoa automatisoinnille. (Kasurinen, J. P. 2013, 78-79)

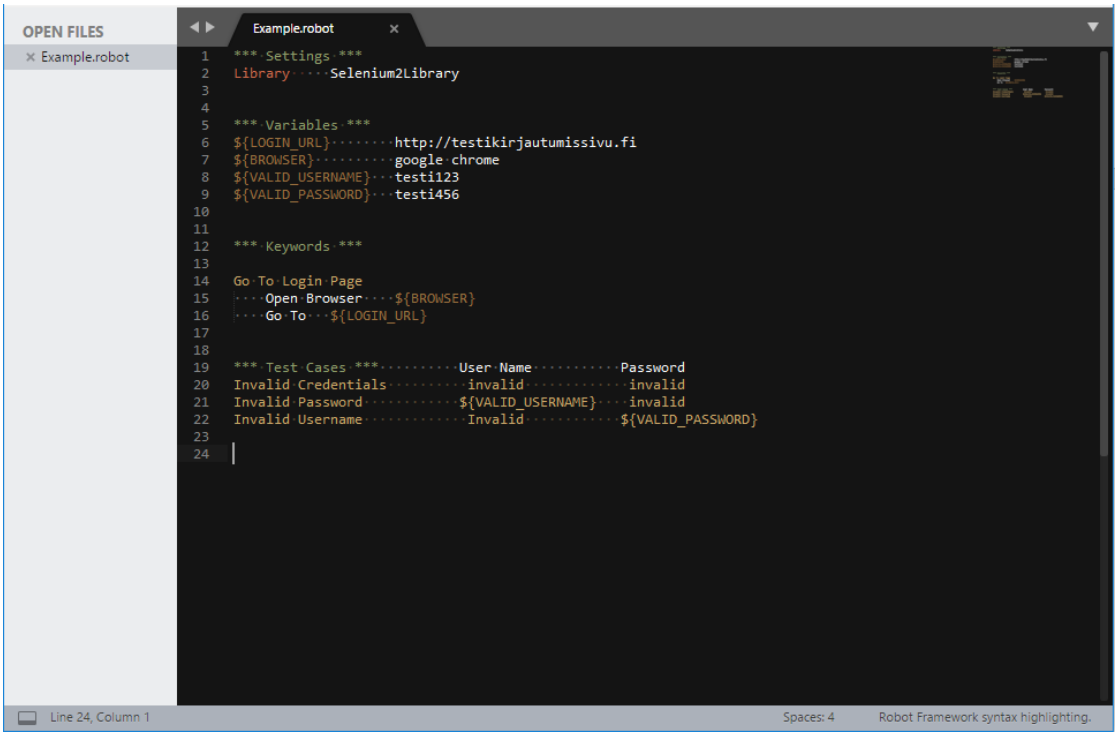
Automatisoinnissa törmätään usein myös muihinkin ongelmiin, kuten sille annettavan huomion vähäisyyteen. Projekteissa voidaan suunnitella automatisoinnin hyödyntämistä, mutta toteutusvaiheessa sille ei priorisoida työvoimaa tarpeeksi, eikä pa-

nostus ole riittävän kattavaa hyödyllisen automatisoinnin toteuttamiseen. Muita ongelmia syntyy, jos automatisointi kohdistetaan vääriin toimintoihin, sillä automatisoinnin ymmärrys ei ole riittävä. Automatisointiin panostuksen ja ymmärryksen vähäisyys taas johtaa siihen, että projektissa käytetään aikaa ja resursseja laaduttoman automatisoinnin toteutukseen ja ylläpitoon, yritetään laajentaa automatisoitua testausta liikaa ja liian nopeasti sekä automatisointiin käytetään vääriä työkaluja eikä toteuttajilla ole tarpeeksi ammattitaitoa. On myös mahdollista, että automatisoinnin parissa työskentelevät eivät ole saaneet tarpeeksi opetusta kyseisten työkalujen käyttämiseen ja automatisoinnin laadukkaaseen toteutukseen. (Kasurinen, J. P. 2013, 78-79)

4.5 Robot Framework

Robot Framework on avoimella lähdekoodilla oleva testauksen automatisointiin suunniteltu työkalu, mikä on toteutettu Python-ohjelmointikielellä. Robot Framework toimii myös Jythonilla sekä IronPython.NET ohjelmistoilla. Toiminnaltaan Robot Framework on helppokäyttöinen taulukkomainen testaustietojen syntaksi. (Robot Framework. N.d.)

Robot Framework on suunniteltu helppokäyttöiseksi, sillä sen toiminta perustuu avainsanoihin, jotka koostuvat erilaisista testaukseen soveltuvista toiminnoista yhdeksi toimintakokonaisuudeksi. Robot Frameworkin toimintaa voidaan myös laajentaa käyttämällä erilaisia valmiiksi luotuja testikirjastoja, jotka sisältävät lisää valmiita avainsanoja ja joista voidaan luoda korkeamman tason avainsanoja testausta varten. Robot Frameworkin käyttöönotto vaatii Python ohjelmointikielen asennuksen tietokoneelle, tekstieditorin joka tukee Robot Frameworkin käyttöä kuten esimerkiksi RIDE tai Sublime Text 3 sekä itse Robot Frameworkin asentamisen tekstieditoriin. (Robot Framework. N.d.)



Kuvio 11. Sublime Text 3 editorilla kirjoitettu Robot Framework esimerkki (Robot Framework. N.d.)

Kuviossa 11 on esitetty yksinkertainen esimerkki Robot Frameworkilla kirjoitetusta testitapauksen rungosta. *** Settings *** kohdan alle annetaan esimerkiksi kirjasto, jota testissä hyödynnetään. Kirjaston alle voidaan myös lisätä esimerkiksi Test Setup ja Test Teardown, jotka ovat jokaisen testin alussa ja lopussa suoritettavia toimintoja. Nämä toiminnot esimerkiksi avaavat selaimen testin alussa ja sulkevat lopussa. *** Variables *** kohdan alle kirjoitetaan testissä käytettyjä muuttujia, jotka yksinkertaistavat testien luettavuutta. Esimerkiksi ${LOGIN_URL} on helpommin luettava koodin sisältä kuin se, että koodiin syötetään kokonainen verkkosivun osoite. Muuttujien käyttö myös helpottaa koodin korjaamista, sillä jos ${LOGIN_URL} on käytössä useassa eri paikassa ja osoite muuttuu, on helpompaa muuttaa se *** Variables *** kohtaan, jolloin se muuttuu kaikkialle, eikä tarvitse jokaisen kirjoitetun testin kohdalle käydä muuttamassa kyseistä osoitetta. Saman *** Variables *** kohdan alaisuuteen kirjoitetaan myös testeissä tarvittavat elementit, kuten tekstikentät tai pudotusvalikot. Elementit tarvitsevat myös jonkun identifioivan merkkijonon, joka voidaan esimerkiksi kirjoittaa suoraan lähdekoodiin. (Robot Framework. N.d.)

Automatisointia kirjoitettaessa luodaan *** Keywords ***, minkä alaisuuteen kirjoitetaan uusia suurempia avainsanoja, jotka koostuvat yksittäisistä käytetyn kirjaston avainsanoista. Kuviossa 11 esitetty Go To Login Page koostuu avainsanoista Open Browser sekä Go To. Open Browserin jälkeen on annettu muuttuja, joka hakee *** Variables *** kohdasta tiedon, että mikä selain avataan. Go To avainsanan jälkeen on annettu muuttuja ${LOGIN_URL}, joka myös hakee muuttujan arvon eli mille sivulle testi vie. *** Test Cases *** alle kirjoitetaan suunnitellut testitapaukset, kuten Kuviossa 11 on kirjautumisen testaaminen väärillä tunnuksilla kolmella eri tavalla. (Robot Framework. N.d.)

Robot Frameworkilla tehdyt testit antavat palautteen, joka mahdollistaa testin läpimenon seuraamisen (ks. liite 2.). Palaute ”Pass and Fail Test Log” mahdollistaa sen, että testejä ei tarvitse seurata paikan päällä, jotta nähdään missä ohjelmiston vikatilanne tapahtuu vai johtuuko syntynyt virhe muusta tekijästä, kuten virheellisestä automaatio koodista. Palautteen hyödyntäminen myös helpottaa testaajan ja ohjelmoijan välistä kommunikaatiota, jos testitulokset ohjautuvat molemmille automaattisesti heti testin päätyttyä. (Pass And Fail Test Log. N.d.)

4.5.1 Selenium2Library

Selenium2Library on verkkopohjaisen sovelluksen käyttöliittymän testausta varten luotu kirjasto, joka koostuu testauksen automatisointia varten luoduista avainsanoista. Selenium2Library käyttää Selenium 2 web ajureita verkkopohjaisen sovelluksen ohjaamiseen. Valmiiksi luotuja avainsanoja Selenium2Libraryssä on 167, joista automatisointia kirjoitettaessa luodaan kokonaisuuksia. Jokaisen avainsanan perässä on myös kerrottu, mitä avainsana tekee. (Selenium2Library. N.d.)

Go To	URL	Navigates the active browser instance to the provider URL

Kuvio 12. Selenium2Library avainsanan esimerkki (Selenium2Library. N.d.)

Kuviossa 12 on esimerkki Selenium2Libraryssä olevasta valmiiksi luodusta avainsanasta. Kyseistä avainsanaa on hyödynnetty myös kuviossa 11, missä nähdään, että Go To on toiminto joka vie haluttuun osoitteeseen, jonka arvo annetaan url kohtaan. Go To avainsanan lopussa on myös kerrottu yksinkertaisesti sen toiminta. (Selenium2Library. N.d.)

4.6 Muita automatisoinnin työkaluja

Testauksen automatisointia varten on kehitetty useita kymmeniä työkaluja. Tässä kappaleessa kerron muutamista suosituimmista automatisoinnin työkaluista, joiden toimintatavat ovat hieman poikkeavia toisistaan. Osa työkaluista on ilmaisia, mutta joidenkin käyttö vaatii maksullisen lisenssin ostamista.

4.6.1 Selenium

Selenium on vuonna 2004 kehitetty selainpohjaisten sovellusten automatisointiin käytettävä työkalu. Yksi suurimmista ominaisuuksista Seleniumilla pidetään sen tukea suorittaa testejä useilla selain alustoilla. Selenium itsessään koostuu useista työkaluista, kuten Selenium 2 eli Selenium WebDriver, Selenium IDE sekä Selenium-Grid. Selenium IDE on Mozilla Firefox -verkkoselaimeen asennettava laajennus, joka sisäl-

tää nauhoitustoiminnon. Nauhoitustoiminto seuraa käyttäjän toimintoja, eli manuaalista testausta, selaimella ja kirjoittaa näistä uudelleen käytettävää koodia. Nauhoituksen kirjoittama koodi eli suoritetut testit voidaan ajaa myöhemmin. Selenium IDE:ä ei ole kuitenkaan suunniteltu laajan testauksen automaation toteuttamiseen, sillä nauhoituksen kirjoittama koodi ei laadultaan vastaa manuaalisesti kirjoitettua koodia. Työkalu onkin enemmän testauksen automatisoinnin prototyyppien suunniteluun suunnattu. (SeleniumHQ. 27.9.2017.)

Selenium WebDriver työkalun tarkoitus on luoda kutsuja selainpohjaiselle sovellukselle. Kutsut kirjoitetaan valitulla ohjelmointikielellä. WebDriver eroaa esimerkiksi Selenium IDE:stä, sillä testit kirjoitetaan, eikä työkalu sisällä nauhoitustoimintoa. WebDriver on suunniteltu toimimaan erityisesti dynaamisilla verkkosivuilla eli verkkosivuilla, joissa tapahtuu muutoksia siten, että sivu ei varsinaisesti päivitä itseään. Selenium WebDriver voidaan asentaa ja sitä voidaan hyödyntää useilla eri kielillä kuten: Java:lla, C#:lla, Pythonilla sekä JavaScriptillä. (SeleniumHQ. 27.9.2017.)

4.6.2 Squish

Squish on maksullinen GUI (Graphical user interface) eli graafisen käyttöliittymän regressio- ja järjestelmätestauksen automatisointiin tarkoitettu työkalu. Squish on suunniteltu toimimaan työpöytä-, mobiili-, web- sekä sulautetuissa järjestelmissä. Työkalu sisältää testauksen nauhoitusominaisuuden, sekä testi skriptien kirjoittamisen ohjelmointikielillä, kuten Pythonilla, JavaScriptillä, Rubyllä, Perlillä sekä Tcl:llä. Muita työkalun sisältämiä ominaisuuksia ovat esimerkiksi: Tehokas testien kehittämisympäristö, Data-Driven testing eli testeissä voidaan hyödyntää datan käyttöä useista eri lähteistä kuten TSV, CSV sekä TXT tiedostoista ja Object Map & Object Identification tools jonka avulla nauhoitettu testaus voidaan muuttaa testiskriptiksi ohjelman tukemille kielille. (Froglogic Home. N.d.)

remove_duplicates.py

```
import codecs
import sys
import time


def main( argv ):
    if len( argv ) != 1:
        print_usage()
        return
    lines = read_file_lines_utf8( argv[0] )
    stripped_lines = {}
    res = []
    for l in lines:
        sl = l.replace( "\r", "" )
        sl = sl.replace( "\n", "" )
        if sl in stripped_lines:
            print 'Removed line: "%s"' % sl
            continue
        stripped_lines[sl] = ""
        res.append( l )
    if len( lines ) == len( res ):
        print "No duplicate lines found."
    else:
        write_file_lines_utf8( argv[0], res )


def print_usage():
    print
    print "USAGE:"
    print
    print "  %s utf8_encoded_text_file_to_remove_duplicate_lines_from" % sys.argv[0]

def read_file_lines_utf8( file_name ):
    res = []
    f = codecs.open( file_name, "r", "utf8" )
    lines = f.readlines()
    f.close()
    for l in lines:
        # Strip the BOM from the beginning of the Unicode string, if it exists
        l = l.lstrip( unicode( codecs.BOM_UTF8, "utf8" ) )
        res.append( l )
    return res

def write_file_lines_utf8( file_name, content_lines ):
    f = codecs.open( file_name, "w", "utf8" )
    f.writelines( content_lines )
    f.close()

if __name__ == "__main__":
    start_time = time.time()

    argv = sys.argv[1:]
    if len( argv ) == 0 or len( argv ) > 1:
        print_usage()
        exit( -1 )

    main( argv )

    t = time.time() - start_time
    print "Done after %.2f seconds." % t
```

Kuvio 11. Esimerkki Squish skriptistä kirjoitettu Python kielellä. Skripti tarkistaa onko koodissa duplikoituja rivejä (Froglogic Home. N.d.).

4.6.3 TestComplete

TestComplete on SmartBear:in kehittämä maksullinen GUI testauksen automatisoinnin työkalu. Työkalu on suunniteltu tukemaan useita eri ohjelmointikieliä sekä sisältämään testien nauhoitusominaisuuden. TestComplete mainostaa nauhoitusominaisuuden laatua, sillä että työkalua voidaan hyödyntää ilman ohjelmistokielten osaamista. SmartBear:lla on TestCompletessa hyödynnettävä ympäristönhallinta työkalu,

minkä avulla testejä voidaan ajaa eri käyttöjärjestelmillä sekä selaimilla. Ympäristönhallinta työkalu toimii pilvipalveluna, mikä mahdollistaa sen hyvän ylläpidon. TestComplete antaa testien ajon jälkeen tiedot testeistä, jotka menivät läpi tai epäonnistuivat. Palautteesta voidaan myös katsoa epäonnistunut testi alusta epäonnistumishetkeen asti videona. (SmartBear. N.d.)

5 Tutkimuksen tarve ja tavoitteet

Firstbeat haluaa viedä jatkuvasti eteenpäin ohjelmistokehitys prosessia, minkä yhtenä suuntana on testauksen automatisointi. Yrityksen ohjelmistotuotteet kasvavat jatkuvasti ja ne koostuvat suuresta määrästä erilaisia komponentteja, joiden testaus manuaalisesti rajallisella työvoimalla on hankalaa. Tämä vaatii testauksen automatisoinnin integroimista testaukseen, jotta ohjelmistokehitysprosessissa viat havaittaisiin mahdollisimman aikaisin, mikä säästäisi aikaa sekä aiheuttaisi vähemmän kustannuksia ja estäisi sen, että työ ei lähtisi missään vaiheessa väärille raiteille. Automatisoinnin yksi suurimmista hyödyistä onkin sen mahdollisuus antaa nopeasti sekä automaattisesti palautetta löydetyistä ongelmista suoraan ohjelmistokehittäjille. Firstbeatin tuotteiden kysyntä on jatkuvasti kasvavaa, mikä vaatii myös tuotteiden uusien ominaisuuksien yhä nopeampaa kehittämistä ja julkaisua. Jokainen suuri projekti vaatii regressiotestauksen toteuttamisen, mikä tällä hetkellä tapahtuu manuaalisesti, jolloin suuri osa ajasta kuluu yksinkertaisten ja rutiininomaisten testien toteuttamiseen, eikä uusien ominaisuuksien ja epätodennäköisempien poikkeustilanteiden etsimiseen. Testauksen automatisoinnilla olisi tavoitteena myös toteuttaa testejä, jotka simuloisivat mahdollisimman tarkasti todellisia käyttötilanteita. (Pyöriä, P. 2017.)

Firstbeat on kehittänyt testauksen automatisointia Firstbeat SPORTS järjestelmälle, sekä kehittää sitä uusille ohjelmistotuotteille jatkuvasti. Hyvinvointianalyysi-järjestelmän laajuus ja automatisoinnin ammattilaisten vähäisyys yrityksessä on kuitenkin vielä ollut esteenä Hyvinvointianalyysi-järjestelmän automatisoinnin toteutukselle. Työni tarkoitus olisi helpottaa kyseisen järjestelmän automatisointia siten, että työni pohjalta olisi mahdollisimman helppo vähemmänkin järjestelmää käyttäneen aloittaa testauksen automatisoinnin toteuttaminen, sillä työni suunnitelma antaisi selkeän kuvan järjestelmän toiminnasta ja automatisoinnin toteuttamisen vaatimuksista.

Firstbeatin ohjelmistokehityksessä pyritään menemään jatkuvasti eteenpäin, mutta esimerkiksi järjestelmien rasitustestaus on vaikeasti toteutettavissa pienellä työvoimalla. Rasitustestaus on hyvä tapa seurata palvelimen kapasiteettia tulevaisuutta varten, mutta laadukas rasitustestaus on mahdoton toteuttaa manuaalisesti. Työssäni pyrin löytämään järjestelmän toimintoja, mihin olisi hyvä toteuttaa rasitustestausta, jotta nähtäisiin, kuinka palvelin reagoi mahdollisimman todellisessa käyttötilanteessa.

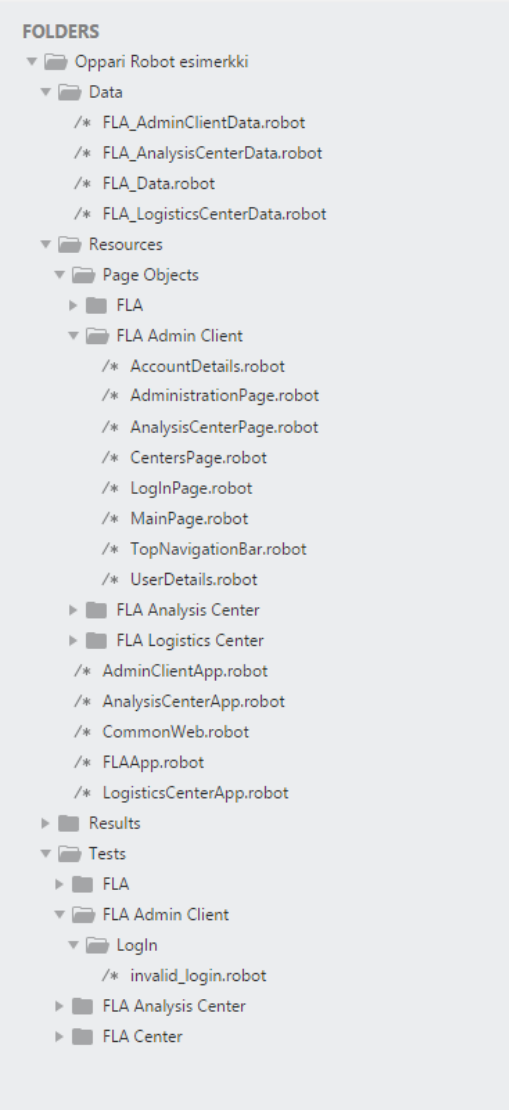
Opinnäytetyöni tavoitteena on siis tehdä selkeä suunnitelma, kuinka Hyvinvointianalyysin järjestelmän automatisointia olisi hyvä lähteä toteuttamaan. Suunnitelma antaisi mahdollisimman selkeän kuvan järjestelmän toiminnasta yksittäisinä asiakasohjelmina, sekä Hyvinvointianalyysi-järjestelmänä. Osana työtä suunnittelen ja toteutan automatisointiohjelmiston rakenteen sekä pohdin myös, millainen fyysinen laitteisto vaaditaan testien toteuttamiselle.

6 Tutkimuksen toteutus

Valitsin työni toteutusta varten automatisoinnin työkaluksi Robot Frameworkin sekä tekstieditoriksi Sublime Text 3 ohjelman. Käytin lisäksi Robot Frameworkissa Selenium2Libaryä avainsanojen luonnissa. Automatisointiohjelman valintaan vaikutti paljon se, että Firstbeat on hyödyntänyt kyseistä työkalua jo SPORTS puolella, joten yrityksessä on hieman aikaisempaa kokemusta sen käytöstä. Sublime Text 3 tekstieditori on myös ilmainen, yksinkertainen ja selkeä työkalu, johon Robot Frameworkin sai asennettua. Robot Framework on myös ilmainen työkalu, toisin kuin moni muu, joten tämä oli myös yksi suuri tekijä valintaani. Toteutusosiossa käyn läpi suunnitelman kehittymistä automatisoinnin ohjelmistorakenteesta asiakasohjelmien ja koko järjestelmän läpikäymiseen sekä automatisoidussa testauksessa tarvittavaan fyysiseen laitteistoon.

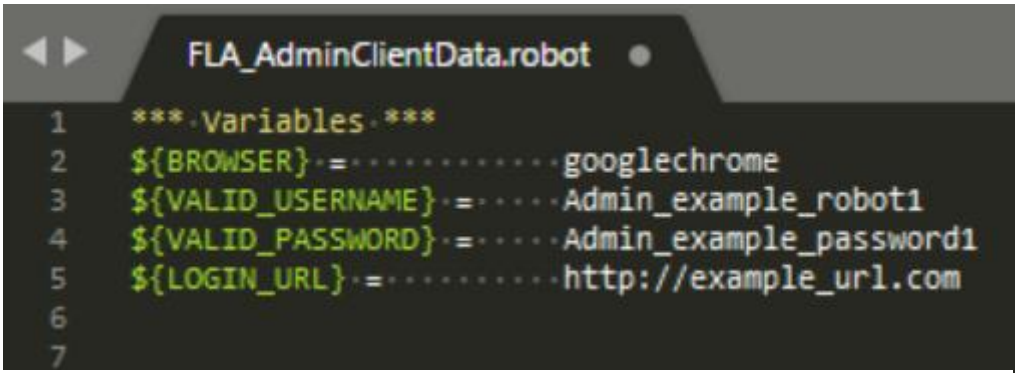
6.1 Automatisoinnin ohjelmistorakenne

Aloitin suunnittelussa käymään läpi Hyvinvointianalyysi-järjestelmän yksittäisiä asiakasohjelmia, joista rakentui runko automatisoinnin ohjelmistorakenteelle. Rakenne koostui kolmesta tasosta: Page-objekti -tiedostoista, App -tiedostoista sekä itse testeistä. Loin myös erikseen datakansion, jonka tarkoitus on sisältää esimerkiksi käyttäjätunnuksia sekä testaussivustojen verkko-osoitteet. Kolmen tason hyödyntäminen selkeyttää yksittäisten avainsanojen ymmärrettävyyttä sekä koodin läpikäyntiä ja korjausta, sillä mitä rivi- sekä merkkimäärältään pienempinä muuttujat, avainsanat sekä testit kirjoitetaan, sen tarkempi Robot Frameworkin luoma testauksen raportti on. Jokainen tiedosto on nimetty päätteellä .robot, sillä Robot Framework vaatii tämän toimiakseen.



Kuvio 13. Automatisointiohjelmiston rakenne

Data -kansiossa jokainen asiakasohjelma on eritelty omaksi tiedostoksi. Jokaisessa tiedostossa on kyseisellä ohjelmalla käytetyt käyttäjänimet, salasanat, verkko-osoite sekä muu kiinteä data kuten käytettävä selain, joita voidaan tarvittaessa helposti muokata, sillä ne ovat selkeästi yhdessä ja samassa paikassa. Data -kansiota voidaan tarvittaessa laajentaa entisestään esimerkiksi tiedostolla, joka sisältää dataa usean asiakasohjelman yhtäaikaiseen testaukseen.

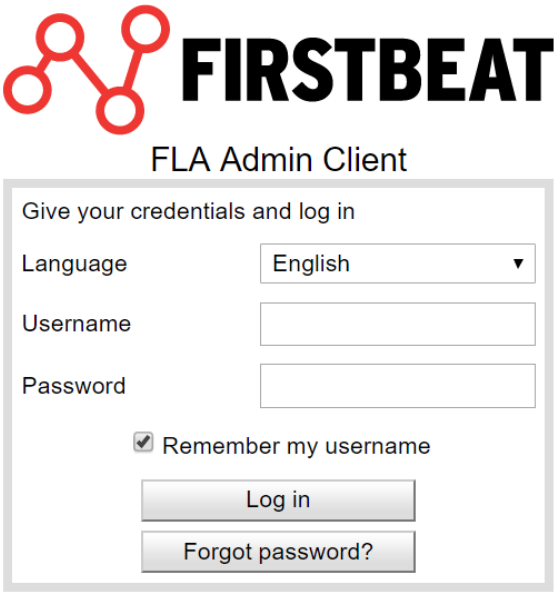


Kuvio 14. Admin Client data esimerkki

6.1.1 Page-objekti tiedostot

Automatisointiohjelmiston rakenteeseen loin Resources -kansion, joka koostuu Page-objekti sekä App -tiedostoista. Rakenteen ensimmäiselle tasolle eli Page-objekti -tiedostoihin keräsin testeissä tarvittavat elementit, jotka vaativat jokainen oman identifioivan merkkijonon ohjelmiston lähdekoodiin. Nimesin elementit mahdollisimman elementtiä kuvaavalla tavalla, mikä helpottaa koodin kirjoittamista, sillä elementin nimestä heti selviää mitä kyseisellä merkkijonolla tarkoitetaan. Tällä hetkellä Hyvinvointianalyysi-järjestelmän lähdekoodissa ei ole elementtien omia identifioivia merkkijonoja, joten nämä olisi hyvä lisätä, kun testauksen automaatiota lähdetään käytännössä toteuttamaan. Lähdekoodista voidaan kopioida myös esimerkiksi XPath merkkijono, joka toimii elementin identifioivana merkkijonona, mutta tämä saattaa aiheuttaa ongelmia, sillä lähdekoodin muutokset voivat muuttaa myös tätä XPath merkkijonoa, jolloin testi ei enää toimi vanhalla merkkijonolla.

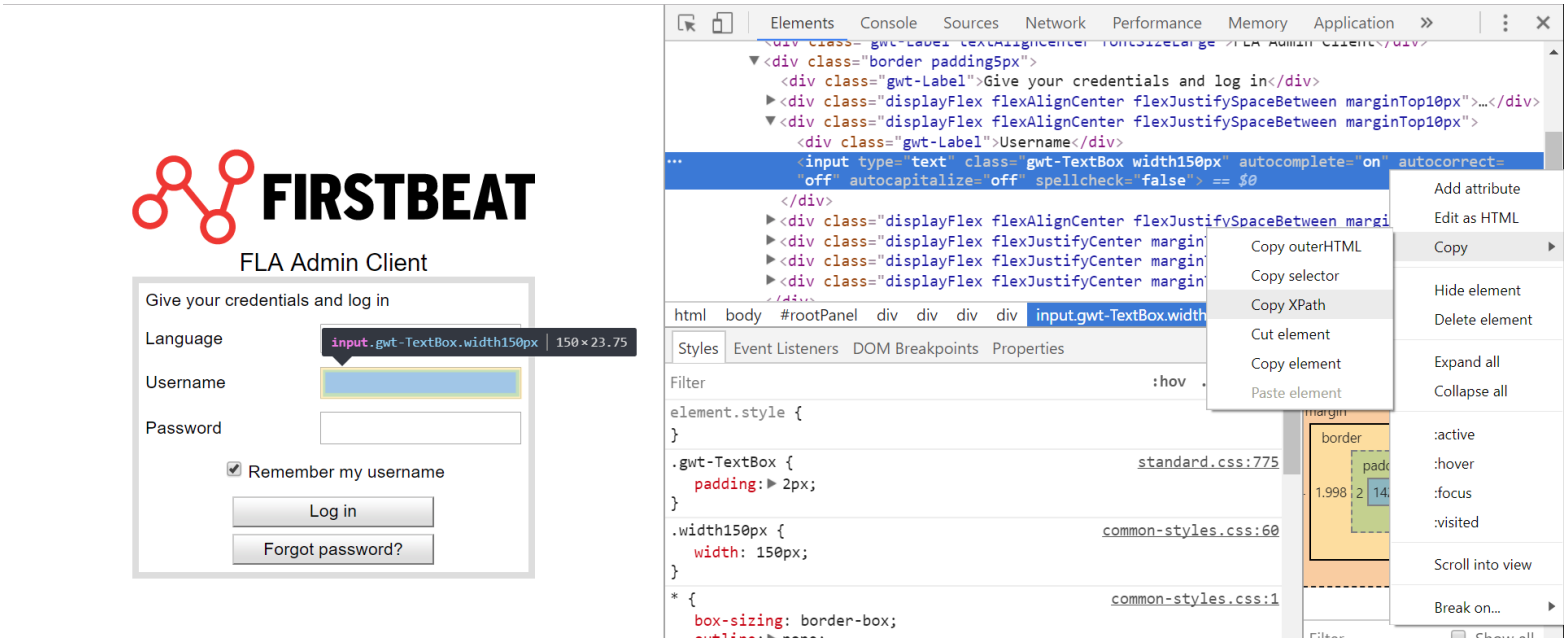
```
4
5   *** Variables ***
6   ${ADMINLOGIN_LANGUAGE_DROPBOX}    id=
7   ${ADMINLOGIN_USERNAME_FIELD}      xpath=//*[@id="rootPanel"]/div/div[2]/div/div[2]/input
8   ${ADMINLOGIN_PASSWORD_FIELD}      id=
9   ${ADMINLOGIN_REMEMBER_CHECKBOX}   id=
10  ${ADMINLOGIN_LOGIN_BUTTON}        id=
11  ${ADMINLOGIN_FORGOT_PASSWORD_BUTTON}   id=
12  ${ADMINLOGIN_FORGOT_PASSWORD_FIELD}    id=
13  ${ADMINLOGIN_FORGOT_PASSWORD_OK_BUTTON}   id=
14  ${ADMINLOGIN_INVALID_CREDENTIALS}   id=
```



Kuvio 15. Admin Client kirjautumissivun page-objektit ja kirjautumissivu

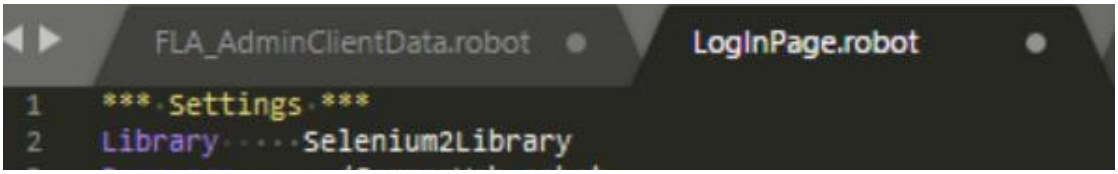
Kuviossa 15. nähdään Admin Client:n Page-objektit sekä itse kirjautumissivu. Nimesin elementit mahdollisimman kuvaavasti kuten ${ADMINLOGIN_USERNAME_FIELD}, joka tarkoittaa Admin Client:n käyttäjätunnus-tekstikenttää. Elementin perään tulisi lisätä id eli identifioiva merkkijono, kunhan sellaiset lisätään lähdekoodiin. Identifioivien merkkijonojen lisäys Java-lähdekoodiin voi tapahtua testauksen automatisoijan toimesta, jolloin jokaisen elementin identifioivaa merkkijonoa ei tarvitse lisätä, vaan ainoastaan testeissä tarvittavat identifioijat. Toinen vaihtoehto olisi että lähdekoodia kirjoittava ohjelmoija itse lisää sovitut identifioivat merkkijonot koodiin. Esimerkkinä käytin myös käyttäjätunnuksen tekstikentän identifioimiseen XPath merkkijonoa.

Identifioivien merkkijonojen löytämiseen hyödynsin Google Chromen Developer -työkalua, joka on rakennettu Chrome selaimeen. Kuviossa 16. on Javalla kirjoitetun lähdekoodin pätkä, joka on muutettu GWT:llä html koodiksi (<input type=”text”…). Developer -työkalulla kopioin halutun elementin XPath:n tai jos Java lähdekoodiin on lisätty oma identifioiva merkkijono niin sitten sellaisen. Itse lisättävä identifioiva merkkijono käyttäjätunnus-tekstikentälle voisi olla esimerkiksi id=loginpage-usernamefield.



Kuvio 16. Google Chrome Developer -työkalun käyttö

Page-objekti -tiedostot sisältävät elementtien lisäksi yksinkertaisia eli ”tyhmiä” avainsanoja, jotka tekevät yksittäisinä vain tietyn pienen toiminnon. Näiden avainsanojen luonnissa hyödynsin Selenium2Libraryä. Yksinkertaisilla avainsanoilla luodaan suurempia kokonaisuuksia, jolloin kaikki käytetyt avainsanat pysyvät kohtuullisissa rivimäärissä, eivätkä veny pitkiksi, jolloin niitä on vaikeampi tulkita ja korjata. Suunnittelussa loin yksinkertaisia avainsanoja esimerkeiksi Admin Client:lle. Jokaiseen Page-objekti -tiedostoon tuodaan *** Settings *** avulla Selenium2Library, sekä esimerkiksi automatisointiohjelmistossa luodut muut tiedostot. Kuviossa 17. on esitetty esimerkkinä, kuinka LogInPage.robot tiedostoon toin Selenium2Libraryn, sekä Common-Web.robot ja FLA_AdminClientData.robot tiedostoihin tallennetut avainsanat ja datat käyttöön. Omien tiedostojen tuonnissa käytetään ../../ merkkejä, minkä avulla Robot Framework ymmärtää miltä automatisointiohjelmiston puun tasolta tiedosto haetaan eli yksi ../ menee rakennepuussa yhden kansiotason ylöspäin.



Kuvio 17. Kirjaston ja resurssien tuonti page-objekti -tiedostoon.

Kuviossa 18. on esitetty yksinkertaisia avainsanoja Admin Client:n kirjautumissivulle tarkoitettujen testien luontia varten. Esimerkiksi ”Open Log In Page” avainsanana menee sille määrättyyn verkko-osoitteeseen ${LOGIN_URL}. Tämän tiedon se hakee FLA_AdminClientData.robot tiedostosta, mihin haluttu verkko-osoite on tallennettu. ”Open Log In Page” ei yksittäisenä avainsanana ole kuitenkaan toimiva testi, sillä testaus vaatii erikseen esimerkiksi selaimen avaamisen. Jokaisen asiakasohjelman Page-objekti -tiedostoihin luodaan siinä ohjelmassa tarvittavat yksinkertaiset avainsanat.

```
*** Keywords ***
Open Login Page
    [Arguments]  ${login_url}=${AC_LOGIN_URL}
    Go To  ${login_url}


Verify Login Page Is Open
    Wait Until Page Contains  FLA Admin Client
    Wait Until Page Contains Element  ${ADMINLOGIN_LOGIN_BUTTON}

Input Username
    [Arguments]  ${username}=${VALID_USERNAME}
    Wait Until Page Contains Element  ${ADMINLOGIN_USERNAME_FIELD}
    Input Into Text Field  ${ADMINLOGIN_USERNAME_FIELD}  ${username}

Input Password
    [Arguments]  ${password}=${VALID_PASSWORD}
    Wait Until Page Contains Element  ${ADMINLOGIN_PASSWORD_FIELD}
    Input Into Text Field  ${ADMINLOGIN_PASSWORD_FIELD}  ${password}

Submit Credentials
    Click Element  ${ADMINLOGIN_LOGIN_BUTTON}

Sign In
    [Arguments]  ${username}=${AC_VALID_USERNAME}  ${password}=${AC_VALID_PASSWORD}
    Input Username  ${username}
    Input Password  ${password}
    Submit Credentials

Login Failed
    [Arguments]  ${msg}
    Wait Until Page Contains Element  ${ADMINLOGIN_INVALID_CREDENTIALS}
    Element Text Should Be  ${ADMINLOGIN_INVALID_CREDENTIALS}  ${msg}
    Click Element  ${ADMINLOGIN_INVALID_CREDENTIALS_OK_BUTTON}
    Sleep  1s
```

Kuvio 18. Admin Client:n yksinkertaisia avainsanoja

6.1.2 App -tiedostot

Automatisointiohjelman rakenteen toiseen tasoon loin jokaisesta erillisestä asiakasohjelmasta App.robot tiedostot. Näiden tiedostojen on tarkoitus koota Page-objekti -tiedostojen yksinkertaisia avainsanoja ja luoda näistä korkeamman tason avainsanoja, joista automatisoidut testit voidaan luoda. App -tiedostoihin on myös hyvä lisätä kommentteja, jotka jakavat avainsanoja niiden toiminnallisuuksien mukaan. Tämä helpottaa suurien avainsana määrien selaamista. Ohjelman Page-objekti -tiedostoon voidaan luoda hieman yksinkertaista korkeampi avainsana kuten sisäänkirjautuminen ”Sign In”, joka koostuu avainsanoista ”Input Username”, ”Input Password” sekä ”Submit Credentials”, joilla avainsana syöttää käyttäjätunnuksen, salasanan sekä painaa kirjaudu sisään painiketta. App -tiedostoon luodaan tästä sitten korkeampi avainsana, joka käyttää ”Sign In” avainsanaa ja toteuttaa lisäksi jonkun testissä tarvittavan toiminnallisuuden. Tarvittavan lisätoiminnallisuuden testaaminen vaatii automaation ajamista. Kuviossa 19. esimerkkinä kirjautumiselle luotu avainsana AdminClientApp.robot tiedostoon, minkä lisätoiminnallisuutena lisäsin Sleep 2s.

```
18  Admin Log In
19      [Arguments]     ${username}=${AC_VALID_USERNAME}     ${password}=${AC_VALID_PASSWORD}
20      LogInPage.Sign In   ${username}  ${password}
21      Sleep 2s
22
```

Kuvio 19. App -tiedoston korkeampi avainsana

App -tiedostoihin on myös hyvä kirjoittaa testeissä tarvittavat Test Setup sekä Test Teardown ja Suite Setup sekä Suite Teardown tapaukset joita testeissä tullaan käyttämään. Esimerkkinä Test Setup, joka luodaan sisäänkirjautumista varten voisi sisältää tietyn selaimen avaamisen, selaimen ikkunan koon maksimoimisen ja siirtymisen asiakasohjelman kirjautumissivulle. Tätä hyödynnettäisiin testeissä siten, että itse testaus alkaa vasta sen jälkeen, kun Test Setup on ajettu. Toisena esimerkkinä Test Teardown voisi kirjautua asiakasohjelmasta ulos sekä sulkea selaimen. Testien alussa ja lopussa suoritettavia toimintoja on hyvä luoda aina tarpeen tullen, sillä jokaista testiä ei ole syytä aloittaa ohjelmaan sisäänkirjautumisella. Nämäkin avainsanat on syytä nimetä sen toimintoja mahdollisimman hyvin kuvaavalla nimellä kuten End Test Case And Close Browser.

6.1.3 Tests -kansio

Tests -kansio on automatisointiohjelman kolmas taso, johon luodaan itse automatisoitavat testitapaukset. Loin Tests -kansioon, jokaiselle asiakasohjelmalle oman kansion, jonka sisään tulisi luoda jokaiselle yksittäisen asiakasohjelman näkymälle oma kansio, jonka sisään lopulta luodaan testejä sisältävät tiedostot. Kuviossa 20. on esimerkkinä Admin Client:n sisäänkirjautumista testaava tapaus, jonka tarkoituksena on tarkistaa, että asiakasohjelma antaa oikean virheviestin, jos käyttäjä syöttää eri tavoin väärät tunnukset.

```
*** Settings ***
Resource        ../../../Data/FLA_Data.robot
Resource        ../../../Resources/AdminClientApp.robot
Resource        ../../../Resources/CommonWeb.robot
Suite Setup     AdminClientApp.Begin Web Test
Test Teardown   AdminClientApp.End Test Case
Suite Teardown  AdminClientApp.End Test Case And Close Browser
Test Template   Login With Invalid Credentials Should Fail

*** Test Cases ***
Empty Username      ${EMPTY}            ${AC_VALID_PASSWORD}    Please give username and password and try again.
Empty Password      ${AC_VALID_USERNAME}    ${EMPTY}            Please give username and password and try again.
Both Empty          ${EMPTY}            ${EMPTY}                Please give username and password and try again.
Invalid Username    Invalid             ${AC_VALID_PASSWORD}    Login failed. Please check your username and password.
Invalid Password    ${AC_VALID_USERNAME}    Invalid             Login failed. Please check your username and password.
Both Invalid        Invalid             Invalid                 Login failed. Please check your username and password.
Expired Account     ${AC_EXPIRED_USER}      ${AC_VALID_PASSWORD}    Your account has been closed. Please contact the suppor

*** Keywords
Login With Invalid Credentials Should Fail
    [Arguments]     ${username}     ${password}     ${msg}
    AdminClientApp.Go To Login Page
    AdminClientApp.Admin Login  ${username}  ${password}
    AdminClientApp.Login Should Fail  ${msg}
```

Kuvio 20. Admin Client:n sisäänkirjautumista testaava tapaus

Sisäänkirjautumisen esimerkkitapauksessa testi pyrkii kirjautumaan sisään Admin Client:in erilaisilla variaatioilla kuten tyhjällä käyttäjätunnuskentällä, tyhjällä salasanakentällä sekä tunnuksilla joita ei ole olemassa. Jokaisen kirjautumisyrityksen jälkeen testi tarkistaa antaako ohjelma halutun virheviestin ruudulle. Käytännössä testi ajaa avainsanan ”Login With Invalid Credentials Should Fail”, ja [Arguments] tietoihin ${username}, ${password} sekä ${msg} se saa datan *** Test Cases *** alta olevasta datasta, jotka taas hakevat datat muista tiedostoista.

Samalle tasolle Test -kansion kanssa automatisointiohjelmiston rakenteessa, loin Results -kansion. Tänne tallennetaan testien tulokset, kun testejä ajetaan. Tämän avulla testitulokset ovat helposti löydettävissä. Automatisointiohjelmiston rakennepuun selkeys ja tiedostojen sekä avainsanojen selkeä jaottelu, helpottaa ohjelmiston ylläpitoa, tulosten läpikäymistä sekä niiden jakamista sitä tarvitseville osapuolille.

6.2 Automatisointi

Tässä kappaleessa käyn läpi testien automatisoinnin suunnittelua yleisesti Hyvinvointianalyysi-järjestelmälle, sekä yksittäisille asiakasohjelmille. Lisäksi suunnittelen mihin Hyvinvointianalyysi-järjestelmässä olisi syytä toteuttaa rasitustestausta. Pohdin myös automatisoinnin mahdollisuutta useampien asiakasohjelmien yhtenäiseen testaamiseen käymällä läpi, kuinka asiakasohjelmat keskustelevat keskenään ja kuinka suurissa määrin tällainen testaus olisi toteutettavissa. Käyn lopuksi läpi millaisia fyysisiä laitteistoja automaatiotestien ajaminen vaatisi.

6.2.1 Testien automatisoinnin suunnittelu

Testauksen automatisointi tulisi aloittaa käymällä läpi asiakasohjelmien regressiotestitapaukset, jotka halutaan automatisoida. Automatisoinnilla ei poisteta järjestelmästä manuaalitestauksen tarvetta ja jokainen testitapaus pitää suorittaa manuaalisesti ennen kuin se automatisoidaan. Hyvänä esimerkkinä regressiotestitapauksen automatisoinnista on kuviossa 20. esitetty kirjautuminen useilla eri tavoilla, joissa kaikissa kirjautumisen tulisi epäonnistua. Tällaisten yksinkertaisten testien automatisoinnissa Robot Frameworkilla päästään luotettavalle tasolle, joten manuaaliselle testaukselle ei ole tarvetta tässä tapauksessa. Tällöin kirjautumissivun manuaalinen testaus voidaan jättää automaation varaan ja keskittyä huomattavasti monimutkaisempien vikojen etsimiseen.

Automatisoitavien testien läpikäymisen jälkeen tulisi kerätä testeissä tarvittavat elementit Page-objekti tiedostoihin. Elementtien keräyksen yhteydessä olisi hyvä lisätä elementtien identifioivat merkkijonot lähdekoodiin, jolloin objektin nimeämisen yhteydessä identifioiva merkkijono olisi myös helppo lisätä Robot Framework koodiin. Identifioivien merkkijonojen lisäämisessä täytyy huomioida se, ettei muuta varsinaista lähdekoodia mitenkään, sillä merkkijonojen tulisi päätyä jokaiselle käytetylle palvelimelle, jotta automatisointia voisi hyödyntää jokaisella palvelimella muuttamalla vain testien verkko-osoitetta.

Elementtien ja identifioivien merkkijonojen avulla kirjoitetaan yksinkertaiset sekä korkeamman tason avainsanat, joita testauksessa nähdään tarpeellisena. Testita-

pausten toteutus olisi hyvä aloittaa tekemällä jokaiselle asiakasohjelmalle perusta-pausten ”best-case scenario”, eli tilanteet joissa asiakasohjelmaa käytetään juuri niin kuin sitä pitäisi, eikä tapahtuisi mitään poikkeuskäsittelyjä. Yleisesti automatisoinnissa toteutetaan kaikki testit tällä tavalla, mutta mielestäni olisi syytä huomioida poikkeuskäsittelyt, joista aiheutuu ongelmia ja joita asiakkaat tekevät tilastollisesti usein.

Admin Client:n ”best-case scenario” automatisointi tapauksia olisi esimerkiksi sisäänkirjautuminen onnistuneesti sekä yrittäminen väärillä tunnuksilla, siirtyminen sivun näkymien (Main page, Center page, Administration page ja Analysis Center page) välillä, eri tilien ja käyttäjien luonti pienimmällä mahdollisella tili/käyttäjä tietomäärällä, tilien ja käyttäjien tietojen editointi sekä uloskirjautuminen. Mielestäni Admin Client:n Excel tiedostojen luonnin toiminta voisi automatisoida mutta datan tarkistamista ei ole syytä automatisoida Excel tiedostoista vaan testaus olisi mahdollista toteuttaa Robot Frameworkilla siten, että testi loisi ja lähettäisi lokimerkintöjä palvelimen rajapinnalle ja tarkistaisi onko vaste eli Exceliin tulostuva data oikeaa. Admin Client:n rasitustestausta voisi kohdistaa suureen tilien sekä käyttäjien luontiin tuotantopalvelimella.

FLA -asiakasohjelmiston automatisointi tapauksia voisi esimerkiksi olla myös sisäänkirjautuminen onnistuneesti sekä yrittäminen väärillä tunnuksilla, yksilö- ja ryhmäanalyysien koko prosessin läpikäynti, eri työkalujen toiminnan varmistaminen, käyttäjätilin asetusten muokkaaminen sekä uloskirjautuminen. Kyseisessä asiakasohjelmassa on myös useita poikkeustapauksia, joita automatisoinnissa olisi mahdollisuus huomioida. Poikkeustapausten huomiointi automaatiossa ei ole tärkeää, ennen kuin ”best-case scenario” testit on luotu ja saatu stabiilisti toimimaan. Automatisoinnilla voidaan myös helpottaa manuaalista testausta siten, että automatisoidaan esimerkiksi suurien ryhmien (yli 10 henkilöä) luonti, jonka jälkeen tällaiset ryhmät voitaisiin ottaa käyttöön manuaalitestauksessa. FLA:lle olisi myös hyvä toteuttaa rasitustestausta esimerkiksi mittausten analysoinnissa sekä raportinluonnissa. Tässä tulee huomioida se, että testauspalvelinten resurssienkäyttö raportinluonnissa on huomattavasti vähäisempää kuin tuotantopalvelimella, joten rasitustestaus tulisi kohdistaa

tuotantoon. Automatisoinnin käytännön toteutuksen suurin työmäärä tulee kohdistumaan FLA asiakasohjelmaan, sillä kyseisessä ohjelmassa on suurin määrä toimintoja sekä sille on luotu suurin määrä testitapauksia.

Logistics Center -asiakasohjelman perusautomatisoinnissa tulisi testata sisäänkirjautuminen eri tavoilla, käyttäjän asetusten muokkaaminen, uusien käyttäjien luominen sekä uloskirjautuminen. Muutoin kyseisen asiakasohjelman testauksen automatisoinnissa tulee huomioida se, että testaus vaatii FLA:ssa luotua dataa toimiakseen. Testaus olisi mahdollista toteuttaa siten, että testit ajetaan sen jälkeen, kun aikaisemmat testit eka luovat FLA:ssa Logistics Center dataa. Toinen vaihtoehto on se, että data luodaan manuaalisesti FLA:ssa Logistics Center:in, jonka jälkeen testit ajetaan käyttäen tätä valmiiksi luotua dataa. Tällainen data vaaditaan siihen, että voidaan tarkistaa eri tilauslistojen sisällön data sekä voidaan testata laitteiden valmistelu ja purkaminen. Manuaalitestauksen helpottamiseksi automatisointia voisi hyödyntää tässäkin tapauksessa FLA:n puolella, missä luotaisiin useita Logistics Center:in tulevia tilauksia, jonka jälkeen manuaalitestaus voisi keskittyä siinä vaiheessa yksinomaan Logistics Center:n asiakasohjelmaan. Logistics Center:n rasitustestauksessa voisi luoda FLA:n kautta suuria määriä tilauksia, jotka menevät Logistics Center:in käsittelyyn, jolloin sinne tulisi suuri määrä dataa listoihin.

Analysis Center -asiakasohjelmassa kuten muissakin perusautomatisoinnissa tulisi testata sisäänkirjautumista onnistuneesti sekä eri tavoilla, jotka testaavat epäonnistuvaa sisäänkirjautumista, tilin asetusten muokkaamista, yksilö- ja ryhmäpuolen analyysien listauksen datan tarkistamista, listojen lajittelua eri tavoin ja yksittäisten analyysien hakemista eri tavoilla. Analysis Center vaatii Logistics Center:n tavoin valmiiksi luotua dataa, jotta sen testaaminen on mahdollista. Tässäkin on mahdollista ajaa Analysis Center testit muiden testien jälkeen, jolloin FLA:n testit ovat luoneet dataa käytettäväksi Analysis Center:in. Vaihtoehtoisesti data voidaan luoda etukäteen manuaalisesti, joiden avulla testit toteutetaan. Kyseisen asiakasohjelman rasitustestauksessa pitäisi luoda FLA:n kautta todella suuri määrä analyysejä, jotka valmisteltaisiin ja palautettaisiin Logistics Center:ssä, jonka jälkeen ne tulevat Analysis Center:n listaukseen.

Jokaisen yksittäisen asiakasohjelman ”best-case scenario” testitapausten automatisoinnin jälkeen, automatisointiohjelmisto vaatii ylläpitoa testien jatkuvan ajamisen ja

tarvittavien korjausten avulla. Testien tulosten suora jakaminen asiakasohjelmien parissa työskentelevien kanssa olisi tehokas tapa hyödyntää automatisointia. Ohjelmiston jatkokehityksessä voisi kehittää useamman asiakasohjelman testaamista yhdessä, siten että esimerkiksi Admin Client:ssä luodaan tili ja käyttäjä, jolla kirjaudutaan FLA:in. Myös tilauksista lähtevien sähköpostien käsittely olisi syytä ottaa automatisointiin mukaan. Sähköpostien käsittelyä varten automatisointiohjelman rakenteeseen voisi luoda Emails -kansion, johon kerätään eri sähköpostien käsittelyssä tarvittavat elementit ja avainsanat. Testit tulisi pitää kuitenkin kohtuullisissa rajoissa, sillä suurien kokonaisuuksien testaaminen Hyvinvointianalyysi-järjestelmässä sisältää todella paljon erilaisia mahdollisia poikkeustilanteita asiakasohjelmien käytössä, jotka ovat parempi jättää manuaalitestauksen varaan.

6.2.2 Vaadittava laitteisto

Testauksen automaatio vaatii ohjelmistototeutuksen lisäksi fyysistä laitteistoa, minkä avulla testit ajetaan. Testit olisi hyvä toteuttaa vähintään kahdella eri kannettavalla tietokoneella, joista toinen olisi Windows kone ja toinen Mac. Tämä johtuen siitä, että esimerkiksi mikään verkkoselain, kuten Google Chrome ja Mozilla Firefox, ei ole täysin identtinen näiden kahden käyttöjärjestelmän välillä. Olisi myös hyvä toteuttaa testien ajo eri selaimilla, erityisesti niillä millä käyttäjät tilastojen mukaan käyttävät Hyvinvointianalyysi-järjestelmää. Automatisoitujen testien määrän kasvaessa testien ajamiseen kuluva aika myös kasvaa, jonka vuoksi voidaan joutua varaamaan vielä useampi kannettava tietokone testausta varten. Erityisesti silloin, jos testit halutaan suorittaa esimerkiksi neljällä eri selaimella molemmissa Windows ja Mac tietokoneissa saman yön aikana.

Tietokoneiden lisäksi Hyvinvointianalyysi-järjestelmän testauksessa tarvitaan Bodyguard 2 -mittalaitteita, joista osa on tyhjiä ja osa sisältää dataa. Olemassa olevien manuaalitestien perusteella parhaimman mahdollisen testauksen verraten todelliseen käyttöön saataisiin, jos yhdessä testikoneessa olisi kiinni 20 Bodyguard -mittalaitetta, joista 10 sisältäisi 3 päivän mittaukset ja 10 olisi tyhjiä. Tämä sen takia, että esimerkiksi testauksessa tarvitaan ryhmä, jossa vähintään 10 henkilöä on suorittanut mielipidekyselyn siitä, kuinka hyvä heidän mielestään Hyvinvointianalyysi oli. Tämä vaatii sen, että kyseisessä ryhmässä on vähintään 10 henkilöä ja näille kaikille on

luotu raportit, jotka vaativat mittausdatan ja henkilökohtaisten tietojen täyttämisen. Testin voisi suorittaa myös yhdellä tyhjällä ja yhdellä dataa sisältävällä Bodyguard2 -mittalaitteella, mutta tämä ei vastaisi todellista käyttötilannetta, sillä todellisuudessa asiakkaiden mittadata ei ole ikinä täysin samanlaista. Toki datan sisältö, ei vaikuta kyseiseen testiin suoranaisesti, mutta näen henkilökohtaisesti parempana toteuttaa kaikki testaus mahdollisimman paljon todellisuutta vastaavana. Rasitustestauksessa kuitenkin, vaikka loisi 100 asiakkaan ryhmän, on pakko käyttää samaa dataa useasti, sillä tietokoneeseen ei ole mitenkään mahdollista saada yhdistettyä toimivasti niin montaa Bodyguard 2 -mittalaitetta USB-porttien vähäisyyden takia.

Mittalaitteiden yhdistämisessä tietokoneeseen vaaditaan lisälaite, sillä esimerkiksi kannettavassa tietokoneessa ei ole 20:tä USB-porttia. Tätä varten Firstbeatilla on tällä hetkellä käytössä kuviossa 21. esitetty D-link DUB-H7 USB-hubi, joka yhdistyy tietokoneen yhteen USB-porttiin. Näihin saa yhdistettyä yhtäaikaisesti 7 Bodyguard 2 -mittalaitetta ja sen avulla laitteiden käyttäminen ja virran lataaminen ovat mahdollista. Kannettavissa tietokoneissa, joissa on Windows käyttöjärjestelmä, on yleensä 3 USB-porttia, joten 20:n Bodyguard 2 -mittalaitteen yhdistämiseen tarvitaan 3 D-link laitetta. Näin ollen 3 USB-porttia on riittävä määrä.

Kuvio 21. D-link DUB-H7 USB-hubi (Verkkokauppa.com D-link. N.d.)

Macin kannettavissa tietokoneissa on yleensä vain yksi USB-portti, joten mittalaitteiden yhdistämistä varten tulisi pystyä jatkamaan yksi USB-portti kahdeksikymmeneksi portiksi. Testasin toimisiko kuviossa 21. esitettyjen D-link USB-hubien käyttö niin, että niitä kytkettäisiin toisiinsa useita, siten että yksi D-link USB-hubi olisi tietokoneessa kiinni ja siihen olisi yhdistetty 7 D-link USB-hubia, jolloin USB porttien määrä olisi ollut 7*7=49. Tämä ei kuitenkaan ollut toimiva ratkaisu, sillä tietokone ei tunnistanut tämän yhdistelmän kautta tarpeeksi suurta määrää Bodyguard 2 -mittalaitteita. Yksi mahdollinen ratkaisu kuitenkin olisi kuviossa 22. esitettävä Manhattan MondoHub USB-hubi laite, johon on mahdollista yhdistää 28 USB-laitetta. Tämän laitteen avulla 20 mittalaitteen yhdistäminen onnistuisi tietokoneeseen yhdellä USB-portilla.

Kuvio 22. Manhattan MondoHub USB-hubi (Thinkgeek. N.d.)

7 Tulokset ja jatkotoimenpiteet

Työni tuloksena sain toteutettua kolmitasoisen automatisointiohjelmisto rakenteen, jonka pohjalle testien kirjoittaminen on mahdollista. Automatisointityökaluksi päädyin valitsemaan Robot Frameworkin, sillä kyseinen työkalu on ilmainen ja hyvin laajalti käytetty ohjelmistotestauksessa. Yrityksessä on myös toteutettu Robot Framework automatisointia SPORTS puolella, joten oli luonnollista valita työkalu, jonka osaamista yrityksen sisällä on. Kokosin automatisointiohjelmistoon eri asiakasohjelmien sivujen elementtejä, joita testeissä voidaan hyödyntää. Kirjoitin myös yksinkertaisia avainsanoja sekä yksittäisiä esimerkkitapauksia testien automatisoinnista.

Kerroin työssäni yksittäisten asiakasohjelmien automatisoinnista ”best-case scenario” testeillä, joista automatisointi pääosin koostuu. Totesin kuitenkin myös sen, että testeissä olisi mahdollisesti syytä huomioida asiakkaiden tekemät poikkeuskäsittelyt, joista aiheutuu ongelmia ja joita he tekevät tilastollisesti usein. Pohdin työssäni myös rasitustestauksen kohdistamista tuotannossa käytettävälle palvelimelle, eri asiakasohjelmissa.

Suunnittelin työni lopussa myös automatisoinnin vaatimia fyysisiä laitteistoja. Fyysisten laitteiden tarpeessa otin huomioon sen, että optimaalinen tilanne automatisoiduissa testeissä olisi se, että testit vastaisivat mahdollisimman hyvin todellisia käyttötilanteita. Testit olisi mahdollista toteuttaa, jopa kahdella Bodyguard 2 -mittalaitteella, joista toinen sisältäisi dataa ja toinen olisi tyhjä. Tämä ei kuitenkaan ole hyvä ratkaisu, sillä esimerkiksi Logistics Center:n laitteiden valmistelu estää saman laitteen valmistelemisen usealle asiakkaalle yhtä aikaa. Tämä aiheuttaa sen, että tyhjä laite valmistellaan asiakkaalle, jonka jälkeen toisella välilehdellä kyseiselle asiakkaalle vaihdetaan dataa sisältävä laite ja kolmannella välilehdellä data puretaan. Jos testaus kohdistuu Logistics Center:n ryhmätilaukseen, tulisi turhia välilehden vaihtoja ja Robot Framework koodin toistoa todella paljon.

Jatkokehityksessä testauksen automatisointia tulisi laajentaa siten, että Hyvinvointianalyysi-järjestelmästä testataan myös loppukäyttäjän käyttöliittymää, analyysien tilauskoodeja ja tilauslomakkeita sekä mahdollisesti useampaa järjestelmää yhdessä käytettynä. Automatisointiohjelmiston jakamista ja hyödyntämistä varten tulisi käyttää esimerkiksi Git versionhallintaohjelmistoa, jolla ohjelmiston jakaminen ja yhtäaikainen kehittäminen olisi mahdollista. Automatisoinnissa pitäisi myös huomioida esimerkiksi mahdollisuus hyödyntää Jenkins-työkalua, jonka avulla testejä voidaan ajastaa toteutumaan haluttuun aikaan, kuten esimerkiksi yöllä.

8 Pohdinta

Työni tarkoitus oli suunnitella Firstbeat Technologies Oy:n Hyvinvointianalyysi-järjestelmälle testauksen automatisoinnin toteutus. Nykyisin testaus kyseiselle järjestelmälle on täysin manuaalisesti toteutettavaa, mutta asiakasohjelmistojen jatkuva kehitys vaatii automatisoinnin kehittämistä manuaalisen testauksen rinnalle. Jokainen uusi projekti vaatii regressiotestausta koko järjestelmälle sekä panostusta uusien ominaisuuksien testaamiseen. Asiakasohjelmista koostuvan järjestelmän regressiotestaus manuaalisesti on kuitenkin lähes kokonaisen projektin kestävä työ, jolloin keskittyminen uusien ominaisuuksien testaukseen jää liian vähäiseksi. Firstbeat halusi opinnäytetyön kyseisestä aiheesta, jotta Hyvinvointianalyysi-järjestelmän automatisointi saisi ensimmäisen askeleen kohti toteutumista.

Tutkimuksen perusteella kolmitasoisen testauksen automatisointiohjelmiston hyödyntäminen olisi hyvä ratkaisu Hyvinvointianalyysi-järjestelmälle ja sen yksittäisille asiakasohjelmille. Automatisointityökaluna luonnollinen valinta on Robot Framework ja sille rakennettu Selenium2Library, sillä ne ovat yleisesti tunnettuja ja hyödynnettyjä, mikä osoittaa niiden olevan laadukkaita. Robot Framework on myös ilmainen, mikä säästää resursseja siinä, ettei tarvitse ostaa lisenssiä työkalua varten. Kyseistä työkalua on jo hyödynnetty yrityksessä, mikä helpottaa automatisoinnin käytännöntoteutuksen toteuttamista sekä automatisointiohjelmiston ylläpitoa toteutuksen jälkeen.

Mielestäni suurin hyöty tutkimuksessani on sen antama yleiskuva koko Hyvinvointianalyysi-järjestelmän automatisoinnille ja siinä huomioitaville asioille. Työstäni saa laajan yleiskuvan järjestelmän yksittäisten asiakasohjelmien toiminnoista, automatisoinnin rakenteesta, huomioitavista testeistä sekä tarvittavasta fyysisestä laitteistosta. Suunnittelemaani automatisointiohjelmiston rakennetta voidaan hyödyntää suoraan automatisoinnin toteutusvaiheessa.

Tutkimukseni ja suunnittelutyöni täytti mielestäni työssä määritellyt tavoitteet ja työtäni voidaan hyödyntää automatisoinnin toteutuksessa. Työ toimi myös ensimmäisenä askeleena kohti automatisoinnin konkreettista toteutumista yrityksessä.

Lähteet

Ammann P. & Offutt J. 2008. Introduction to software testing. USA: Cambridge University Press. Viitattu 27.8.2017

Cohn M. 2009. Artikkeli: The forgotten layer of the Test Automation Pyramid. Viitattu 17.9.2017. https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid

Firstbeat Admin Client. N.d. Admin Clientin projekti ympäristö. Viitattu 21.7.2017. https://project.firstbeat.fi/admin-client/#

Firstbeat Analysis Center. N.d. Analysis Centerin projekti ympäristö. Viitattu 21.7.2017. https://project.firstbeat.fi/nexus/#

Firstbeat Lifestyle Assessment. N.d. Lifestyle Assessmentin projekti ympäristö. Viitattu 21.7.2017. https://project.firstbeat.fi/fla/#

Firstbeat Logistics Center. N.d. Logistics Centerin projekti ympäristö. Viitattu 21.7.2017. https://project.firstbeat.fi/logistics-center/

Firstbeat Tarinamme. N.d. Firstbeat Technologies Oy:n internet-sivut. Viitattu 14.7.2017. https://www.firstbeat.com/fi/yritys/tarina/

Firstbeat Tekniset tiedot. N.d. Firstbeat Technologies Oy:n internet-sivut Viitattu 14.7.2017. https://www.firstbeat.com/fi/tyo-ja-hyvinvointi/hyvinvoinnin-ammattilaiset/tekniset-tiedot/

Froglogic Home. N.d. Squish: Reliable GUI Test Automation that works. Viitattu 8.10.2017. https://www.froglogic.com/

Kasurinen, J. P. 2013. Ohjelmistotestauksen käsikirja. 1. p. Jyväskylä: Docendo. Viitattu 5.7.2017

Lent J. 2013. FAQ: Automated software testing basics. Viitattu 17.9.2017. http://searchsoftwarequality.techtarget.com/feature/FAQ-Automated-software-testing-basics

Li, K. & Wu, M. 2004. Effective software test automation: Developing an automated software testing tool. USA: Sybex. Viitattu 17.9.2017

Pass And Fail Test Log. N.d. Robot Framework. Viitattu 24.9.2017 http://robotframework.org/robotframework/latest/images/visible_log_level.html

Pyöriä, P. 2017. Chief engineer. Firstbeat Technologies Oy. Haastattelu 25.9.2017

Robot Framework. N.d. Generic test automation framework for acceptance testing and ATDD. Viitattu 23.9.2017. http://robotframework.org/#introduction

Selenium2Library. N.d. Robot Framework. Viitattu 24.9.2017. http://robotframework.org/Selenium2Library/Selenium2Library.html

SeleniumHQ. 27.9.2017. Documentation. Viitattu 30.9.2017. http://www.seleniumhq.org/docs/01_introducing_selenium.jsp

SmartBear. N.d. TestComplete: Easiest To Use Automated UI Tetsing Tool. Viitattu 8.10.2017. https://smartbear.com/product/testcomplete/overview/

Sommerville, I. 2016. Software Engineering (10th Edition). Pearson Education Limited. Viitattu 26.8.2017

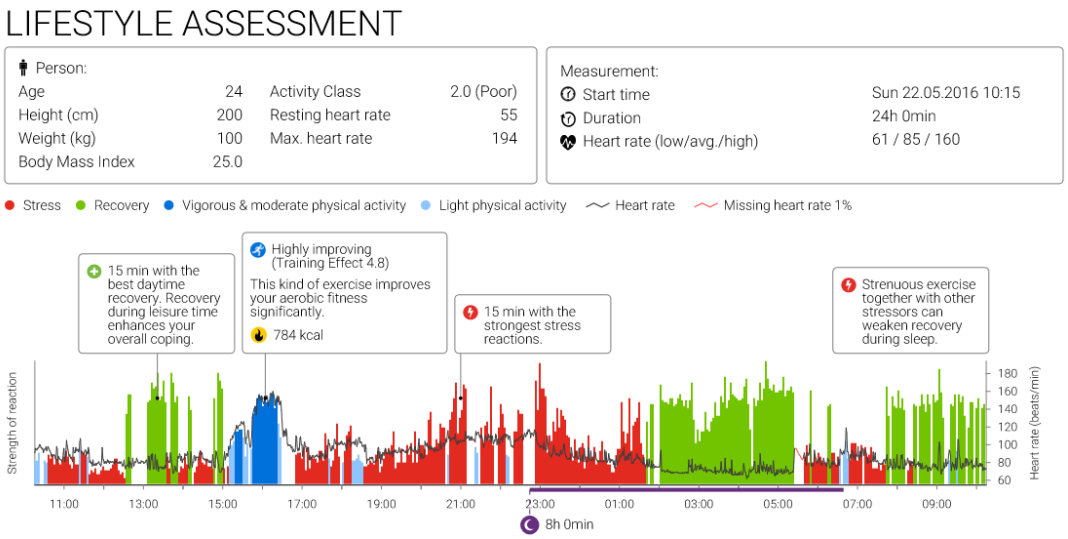
Stress and recovery analysis method based on 24-hour heart rate variability. 2014. White paper: Firstbeat Technologies Oy. Viitattu 14.7.2017. https://www.firstbeat.com/app/uploads/2015/10/Stress-and-recovery_white-paper_20145.pdf

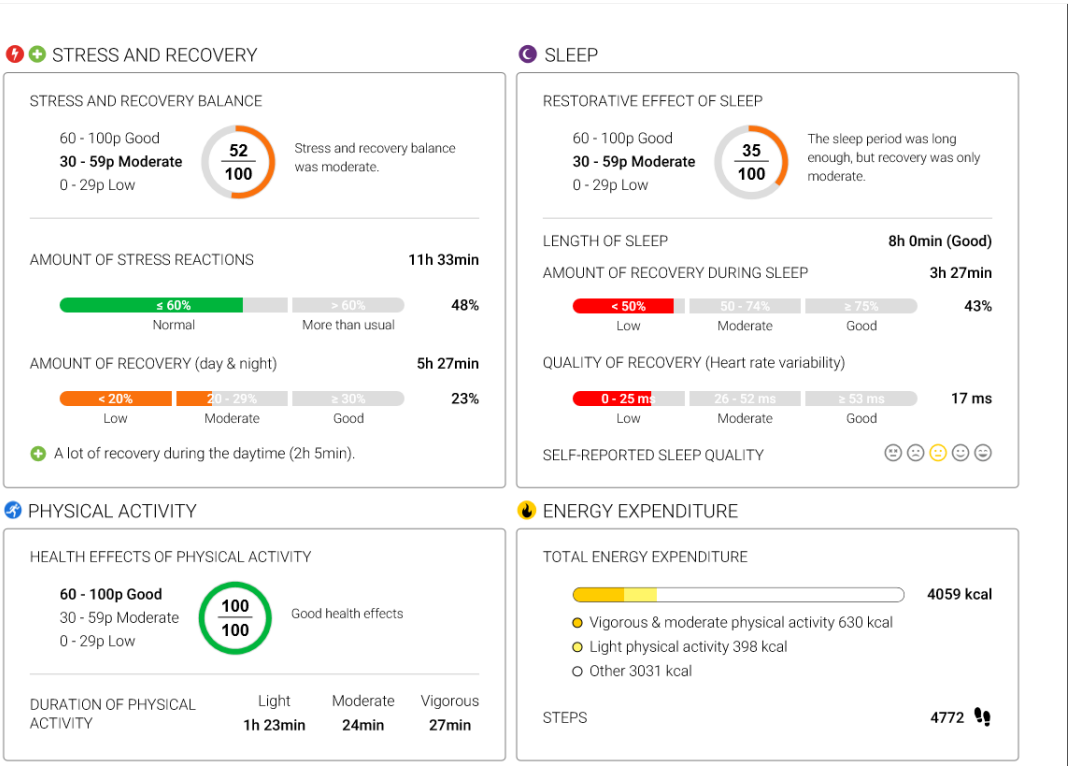
Thinkgeek. N.d. MondoHub 28 Port USB 2.0/3.0 Hub. Viitattu 20.10.2017. http://www.thinkgeek.com/product/eecf/#tabs

Verkkokauppa.com D-link. N.d. D-link DUB-H7 USB-hubi. Viitattu 20.10.2017. https://www.verkkokauppa.com/fi/product/19670/kvfk/D-link-DUB-H7-USB-hubi

Liitteet

Liite 1. Hyvinvointianalyysin raportin data (Firstbeat Lifestyle Assessment. N.d.)





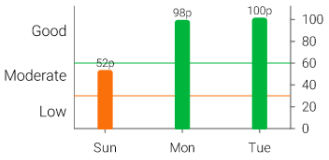
LIFESTYLE ASSESSMENT SCORE

The score is based on your combined stress and recovery, sleep and physical activity result. By improving these areas, you can promote your well-being and improve your Lifestyle Assessment score.

85 - 100p Excellent
60 - 84p Good
30 - 59p Moderate
15 - 29p Low
0 - 14p Very low

The average score of all Lifestyle Assessment participants is 55p.

STRESS AND RECOVERY BALANCE

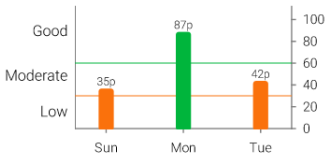


Your result

GOOD

In your age group average is 59p.

RESTORATIVE EFFECT OF SLEEP

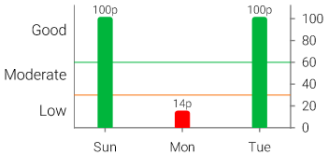


Your result

MODERATE

In your age group average is 64p.

HEALTH EFFECTS OF PHYSICAL ACTIVITY



Your result

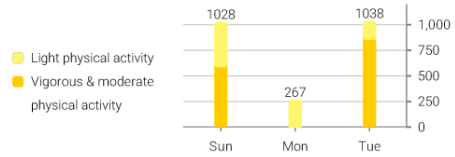
GOOD

The measurement included 2 very hard workouts. Remember to balance with rest and light exercise.

Great! The health effects produced by physical activity were good on 2 days.

ENERGY EXPENDITURE

ACTIVE CALORIES (kcal)



	Sun	Mon	Tue
TOTAL	4058	2791	3082
STEPS	4772	3182	11647

Liite 2. Robot Framework testin tulokset (Pass And Fail Test Log. N.d.)

REPORT
Log level: TRACE

Pass And Fail Test Log

Generated
20120522 11:01:55 GMT +03:00
5 years 125 days ago

Test Statistics

Total Statistics	Total	Pass	Fail	Graph
Critical Tests	2	1	1	
All Tests	2	1	1	

Statistics by Tag	Total	Pass	Fail	Graph
fail	1	0	1	
force	2	1	1	
pass	1	1	0	

Statistics by Suite	Total	Pass	Fail	Graph
Pass And Fail	2	1	1	

Test Execution Log

TEST SUITE: **Pass And Fail** Expand All

Full Name: Pass And Fail
Documentation: Some tests here
Source: /private/tmp/pass_and_fail.html
Start / End / Elapsed: 20120522 11:01:55.797 / 20120522 11:01:55.848 / 00:00:00.051
Status: 2 critical test, 1 passed, 1 failed
2 test total, 1 passed, 1 failed

SETUP: **My Keyword Suite Setup** Expand All

TEST CASE: **Pass** Expand All

TEST CASE: **Fail** Expand All

Full Name: Pass And Fail.Fail
Documentation: FAIL Expected failure
Tags: fail, force
Start / End / Elapsed: 20120522 11:01:55.840 / 20120522 11:01:55.847 / 00:00:00.007
Status: FAIL (critical)
Message: Expected failure

KEYWORD: **My Keyword Fail** Expand All

KEYWORD: **BuiltIn.Fail msg=Expected failure** Expand All

Documentation: Fails the test immediately with the given (optional) message.
Start / End / Elapsed: 20120522 11:01:55.845 / 20120522 11:01:55.847 / 00:00:00.002

```
11:01:55.845   TRACE   Arguments: [ msg=u'Expected failure' ]
11:01:55.846   FAIL    Expected failure
11:01:55.846   DEBUG   Traceback (most recent call last):
                         File "/Users/jmalinen/Documents/workspace/robot/src/robot/libraries/BuiltIn.py", line 305, in Fail
                           raise AssertionError(msg) if msg else AssertionError()
```