Selainpohjaisen järjestelmän automaatiotestaus Robot Framework -automaatiotyökalulla

Ilmari Ojanen

Haaga-Helia ammattikorkeakoulu
Liiketalouden ammattikorkeakoulututkinto
Tietojenkäsittelyn koulutusohjelma
Amk-opinnäytetyö
2024

Tekijä
Ilmari Ojanen

Tutkinto
Tradenomi – Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
Selainpohjaisen järjestelmän automaatiotestaus Robot Framework -automaatiotyökalulla

Sivu- ja liitesivumäärä
41 + 2

Automaatiotestauksesta on kehittynyt ohjelmistotuotannon vakiintunut työkalu, jolla voidaan saavuttaa monia hyötyjä kohdejärjestelmän kehityksessä. Automaatiolla voidaan lisätä testauksen kattavuutta ja testien suorittamisen nopeuttaa. Tietokoneen suorittamat automaatiotestit vapauttavat testauksen resursseja toistuvasta manuaalisesta suorituksesta muihin tehtäviin. Lisäksi automaatio mahdollistaa virheiden havaitsemisen ja niiden korjaamisen aikaisemmassa vaiheessa. Automaatiotestauksen toteuttamiseen on tarjolla paljon erilaisia kehyksiä ja toteutustapoja.

Opinnäytetyön tietoperustassa esitellään lyhyesti työssä käytetyt automaatiotyökalut ja kirjastot. Lisäksi tutustutaan erilaisiin automaatiotestien skriptaustekniikoihin ja niiden hyviin toimintatapoihin. Hyvien toimintatapojen osalta työssä nojataan vahvasti Robot Frameworkin alkuperäisen kehittäjän Peter Klärckin ohjeisiin ja Robot Frameworkin dokumentaatioon.

Työn toiminnallisessa osuudessa käynnistettiin automaatiotestaus toimeksiantajan selainpohjaiseen järjestelmään Robot Framework -automaatiokehystä hyödyntäen. Työn alussa sovittujen ominaisuuksien automaatiotestit laadittiin ensin avainsanakeskeisellä tyylillä käyttäen Selenium-kirjastoa, mutta projektin edetessä testit muutettiin käyttämään Browser- ja DataDriver-kirjastoja. Toimeksiantajalle toteutettiin myös erillinen Excel-pohjainen testien hallinnointikäyttöliittymä, jonka avulla järjestelmän ylläpitotiimi voi suorittaa testejä ilman automaatiokehyksen tuntemusta. Opinnäytetyön toimeksiantajaa, kohdejärjestelmää tai tuotoksia ei voida työssä esitellä tarkasti, mutta niitä kuvaillaan yleisellä tasolla. Lisäksi tuotoksia demonstroidaan julkisessa Gitlab-projektissa, joka löytyy osoitteesta https://gitlab.com/automaatiotesti/ont_robot_framework.

Työssä toteutetun automaatiotestauksen on koettu helpottavan ylläpitotiimin työtä ja se on tarjonnut hyvän pohjan, jolta kehitystä voidaan jatkaa. Automaation toteutuksessa käytetty Robot Framework on osoittautunut helposti lähestyttäväksi ja käyttäjäystävälliseksi työkaluksi, jonka käyttöä voidaan laajentaa muuhunkin kuin pelkkään testiautomaation.

Asiasanat
Automaatio, Testaus, Testit, Verkko-ohjelmointi, Käyttöliittymät

Sisällys

1 Johdanto

Automaatiotestaus yleistyy ja iso osa testaukseen ja laadunvarmistukseen liittyvistä työpaikoista listaa vaatimuksissa tai toiveissa automaatio-osaamisen. Hyötyjensä vuoksi automaatiotestaus on nopeasti kehittynyt testauksen kysytyimmäksi työkaluksi. Automaation tarkoitus on täydentää ja tukea manuaalista testausta. Tämä mahdollistaa resurssien siirtämisen työläästä käsin tehtävästä toisteisesta suorituksesta hyödyllisempään työhön.

Järjestelmät laajenevat ja monimutkaistuvat jokaisen uuden version myötä. Tarkistusluoteinen työ, jossa varmistetaan uusien ominaisuuksien toimivuus ja niiden yhteen sopiminen vanhojen kanssa, vie entistä enemmän aikaa testauksesta. Lisäksi vanhojen ominaisuuksien toiminnan varmistaminen vie joka versiossa enemmän aikaa, kun ominaisuuksia lisätään järjestelmään. Automaatiolla voidaan saada säästöjä ohjelmistokehityksen työmäärään vapauttamalla testaajat muihin tehtäviin käsin tehtävästä testien suorittamisesta.

Tämä opinnäytetyö on toimeksiannon pohjalta tehty toiminnallinen työ. Opinnäytetyössä kuvataan, kuinka hiljattain tuotantoon siirretyn verkkopohjaisen sovelluksen automaatiotestaus aloitettiin Robot Framework -testaustyökalulla. Toimeksiantajalla oli meneillään projekti, jossa useamman järjestelmän regressiotestausta lähdettiin automatisoimaan ja kohdejärjestelmä oli otettu projektiin mukaan. Käyttöliittymän automaatiotestauksen toivottiin helpottavan järjestelmän ylläpitotiimin työtä virheiden aikaisemmalla löytymisellä ja vähentämällä manuaalista työtä regressiotestauksessa.

Työn tuotoksia ei voi esitellä sellaisenaan salassapitosopimuksien vuoksi. Työtä tehdessä loin vastaavanlaisen testiympäristön omana harjoitteluna. Tässä Gitlab-repositoriossa on esiteltynä vastaavat komponentit kuin toimeksiantajalle tehtiin. ReadMe-tiedostosta ja Testien_hallinnointi.xlsm-tiedostosta löytyy ohjeet, joiden avulla tuotoksia voi ottaa itse käyttöön ja kokeilla Robot Framework -työkalua. Repositorio löytyy osoitteesta https://gitlab.com/automaatiotesti/ont_robot_framework.

1.1 Opinnäytetyön tavoitteet ja aiheen rajaus

Opinnäytetyöllä oli neljä päätavoitetta. Tärkeimpänä tavoitteena oli käynnistää kohdejärjestelmän automaatiotestaus testiympäristössä. Työssä toteutetaan testiskriptit valittuihin oleellisimpiin toiminteisiin. Automaatiossa käytettiin laajemmassa testausprojektissa valittuja työkaluja: Robot Framework -kehystä ja aluksi Selenium-kirjastoa, myöhemmin kirjastoiksi vaihdettiin Browser ja DataDriver. Työssä tehtiin myös suunnitelma testauksen jatkokehitykselle.

Testien materiaalille ja skripteille tarvittiin kehitysympäristö ylläpitoa varten. Tätä varten oli tavoite perustaa Gitlab-repositorio, johon materiaalit vietiin. Repositorioon haluttiin myös valmisteltavan CI/CD putki testien ajastamista varten. Prosessista tarvittiin dokumentaatio ja ylläpitotiimille selkeä ohjeistus testien käyttöönotosta.

Lisäksi toimeksiantajan puolelta tuli tavoite laatia testien suorittamista helpottava käyttöliittymä. Testejä tulee jatkossa ajamaan ylläpitotiimi, joka ei tunne kyseessä olevaa automaatiokehystä. Työkalusta toivottiin yksinkertaista tapaa suorittaa testejä ilman komentokehotteen käyttöä tai ohjelmointi sovellusten käyttöä.

Opinnäytetyön teoriaosuudessa käsitellään lyhyesti testausta yleisesti ja kuvataan automaatiotestausta sekä ”best practices” -tyyppisiä hyviä käytäntöjä testien luomisessa. Testaus ja automaatio ovat laajoja kokonaisuuksia ja erilaisia työkaluja on paljon. Tästä syystä työn aihe rajataan keskittymään käytettyihin automaatiotestauksen työkaluihin: Robot Frameworkiin ja sen kirjastoihin, sekä automaatioskriptauksen hyviin käytänteisiin. Ajatuksena oli tuottaa materiaalia, joka on hyödyllistä tietää automaatiotestauksen aloittamisessa.

1.2 Henkilökohtaiset tavoitteet

Automaatiotestaus oli itselleni uusi taito, johon olin tutustunut vain yhden opintojakson verran. Tavoitteena oli ottaa haltuun tärkeä taito nykypäivän IT-kehityksessä. Työn aikana sain käyttökokomusta yleisesti käytössä olevasta automaatiokehyksestä, sen työkaluista ja menetelmistä.

1.3 Keskeiset käsitteet

Automaatiotestaus Testien suorittamista tietokoneen tai muun työkalun avulla ilman ihmisen suoraa ohjeistusta (ISO/IEC/IEEE 29119-1:2022, 6).

Avainsana Avainsanapohjaisten automaatiokehysten perusosa. Eräänlainen kutsuttava funktio, joka viittaa suoritettaviin toimintoihin. (ISO/IEC/IEEE 29119-1:2022, 5.)

CI/CD-putki Continuous Integration / Continuous Deployment -putki, on ohjelmistokehityksen työkalu, jossa ohjelmiston uusia versioita voidaan luoda ajastetusti tai aina kun muutoksia lisätään. (Gitlab s.a.[a])

Verkkosivuelementin ID Elementin ID:tä käytetään määrittelemään tietty verkkosivun elementti. W3Schools s.a.).

Gitlab	Ohjelmistokehityksen versionhallinta-alusta, jota voidaan käyttää avuksi myös testauksessa (Gitlab s.a.[b]).
Gitlab runner	Sovellus, joka suorittaa ja ajaa Gitlabin CI/CD-töitä (Gitlab s.a.[b]).
Lambda-järjestelmä	Keksitty nimi toimeksiantajan selainpohjaiselle järjestelmälle, jonka automaatiotestausta lähdettiin kehittämään.
Manuaalitestaus	Ihmistestaajan suorittamaa järjestelmän testausta ja tiedon hankkimista (ISO/IEC/IEEE 29119-1:2022, 6).
Regressiotestaus	Testausta, jonka tarkoitus on varmistaa entisten muuttumattomien ominaisuuksien toimivuus, kun uusia ominaisuuksia lisätään järjestelmään (ISO/IEC/IEEE 29119-1:2022, 8).
Robot Framework	Automaatiokehys, jota voidaan käyttää testaukseen ja ohjelmistorobotiikkaan (Robot Framework Foundation s.a.[a]).
Selainpohjainen järjestelmä	Verkkosivun kautta toimiva ohjelmisto (ISO/IEC/IEEE 29119-1:2022., 45).
Selektorit/lokaattorit	Suomennettuna ”valitsimet”, ovat verkkosivuelementtien ominaisuuksia, joilla voidaan kohdistaa vaikutus tiettyyn elementtiin. (Mozilla s.a.) Robot Frameworkin avainsanoissa voidaan käyttää valitsimina muun muassa verkkosivuelementin ID:tä tai XPath.
Syntaksi	Säännöt, jotka määrittävät ohjelmointikielen tai kehyksen rakenteen (Cambridge University Press & Assessment s.a.).
VBA	Microsoftin Office tuotteita laajentava koodikieli Visual Basic For Applications (Microsoft 2021).

2 Testaus

Testaus itsessään on monitahoinen aihe, koska sillä saavutettavat tulokset ovat hyvin subjektiivisia. Laaja ja kattava testauskaan ei välttämättä löydä kaikkia virheitä eikä pelkkä testaus paranna tuotteen laatua. (Rytkönen & Kakkonen 2023, 70.) Testaus on toimintaa, joka luo uutta tietoa kohteesta. Tiedon avulla voidaan parantaa kohdetta, mutta testaus itsessään ei paranna asioita. Testaus ei ole sama kuin laadunvarmistus, sillä kattavinkaan testaus ei tuo 100 % varmuutta siitä, että järjestelmässä ei ole virheitä. (Niittyvirta 2019, 45–47, 53–54.) Testauksella kerättävällä tiedolla voidaan parantaa kohteen laatua. Laatu on kuitenkin subjektiivinen ja useammalla tavalla määriteltävissä oleva termi. Laadulla voidaan tarkoittaa esimerkiksi sitä, että vaatimukset täyttyvät, järjestelmää voi käyttää tehokkaasti, kaikki viat on korjattu tai sitä, että koodi on ymmärrettävää ja testaus on helppoa. (Rytkönen & Kakkonen 2023, 16–17, 27.)

2.1 Testauksen eri muodot ja tyypit

Testaus voidaan jakaa eri tyyppeihin monella tapaa riippuen siitä, mitä testataan, miten testataan ja missä tilassa kohdetta testataan. Yksi tapa on jakaa testaus dynaamiseen ja staattiseen testaukseen. Staattisessa testauksessa ei suoriteta koodia. Tällöin testaus voi olla esimerkiksi erilaista katselmointia koodiin tai dokumentaatioon tai käsittää erilaisia analysointeja. Dynaamisessa testauksessa kohdejärjestelmän koodia tai koodin osia suoritetaan testauksen aikana. Tämä testaus voidaan taas jakaa sen mukaan, millaista ominaisuutta testataan. Funktionaalisessa testauksessa tarkistetaan, että järjestelmä toimii niin kuin sen halutaan toimivan ja niin kuin se on koodattu. Ei-funktionaalisessa testauksessa tarkistetaan ohjelmiston muita osia kuten tietoturvaa, käytettävyyttä tai suorituskykyä. (Bajpai 2012, 8.)

Erilaisella testaamisella haetaan erilaisia tuloksia ja työ voidaan jaotella tämän tarkoituksen mukaan. Tarkistusluontoisella työllä voidaan valmiiden testitapausten avulla tuottaa tietoa siitä, että järjestelmä toimii kuten ennenkin. Käytännössä katsotaan, että virheitä ei ole ilmestynyt aiemmin toimiviin ominaisuuksiin. Saalistusluonteisella työllä taas pyritään löytämään uusia virheitä järjestelmästä. (Niittyviita 2019, 52–54.) Voidaan puhua myös tutkivasta testauksesta. Tutkivassa testauksessa minimoidaan tarkkojen testitapausten ja dokumentaation määrä ja keskitytään vikojen löytämiseen. Testejä ohjaa ohje, jossa kuvataan, mitä tutkitaan, millä keinoilla tutkitaan ja mahdollisesti mitä halutaan löytää. Tutkivaa testausta käytetään usein, kun etsitään virheitä käyttäjäpalautteen perusteella. Testaajan substanssiosaaminen, järjestelmätuntemus ja kokemus ovat isossa osassa tutkivassa testauksessa. Usein testissä löydetty pieni poikkeama johtaa uusiin testeihin, joilla yritetään avata tilannetta enemmän. Dokumentaation ja testitapausten puutetta voidaan kompensoida erilaisilla näytöntallennustyökaluilla, jotta testaaja voi palata takaisin ja tarkistaa, mitä oli tarkalleen tehty, jotta tilanne saatiin toistumaan. (Rytkönen & Kakkonen 2023, 165.)

Lisäksi testaaminen voidaan jakaa erityyppisiin alalajeihin. Järjestelmäkehityksen vaiheen ja tarkastelussa olevien ominaisuuksien mukaan käytetään eri testaustyyppejä. Yleisenä käytäntönä kannattaa panostaa useampaan erityyppiseen testaukseen jonkin verran kuin pelkästään yhteen. Näin voidaan parantaa testauksen kattavuutta. (Rytkönen & Kakkonen 2023, 133–163.)

Eri testityypit keskittyvät löytämään erilaisia asioita käyttäen erilaisia menetelmiä tai työkaluja. Alalajeja voidaan erotalla tarkastelemalla, mikä on testauksen kohde ja mihin testit perustuvat. Liitteessä 1 on kuvattu erilaisia testauksen tyyppejä ja mistä lähteestä testit johdetaan. Tyyppejä on muitakin ja näiden otsikoiden alle kuuluu erilaisia alalajeja.

2.2 Testauksen työkalut

Testauksessa käytettävän työkalun valinnassa tulee tarkastella useita erilaisia kriteereitä: 1) Mitä osiota järjestelmästä ollaan testaamassa ja millä teknologioilla kohdejärjestelmä on toteutettu? 2) Millä alustalla toimitaan? 3) Mitä työkaluja on aiemmin käytetty? 4) Mikä on testaajien osaamisen ja tietämyksen taso? 5) Onko jokin työkalu jo valmiiksi käytössä? 6) Onko valittu työkalu muualla käytössä ja tuettu? 7) Työkalun hinta. (Rytkönen & Kakkonen 2023, 241–243.)

Tässä työssä keskitytään projektin toteutuksen kannalta oleellisiin työkaluihin, automaatiotestaukseen ja siinä käytettäviin työkaluihin Robot Frameworkiin sekä tätä laajentaviin kirjastoihin.

2.3 Automaatiotestaus

Testaus on työläs ja sekä aikaa että rahaa vievä prosessi. Järjestelmät laajenevat ja monimutkaistuvat ja samalla testauksen vaatimukset kasvavat. (Milad ym. 2014, 194.) Testauksen automatisointi on yksi mahdollinen ratkaisu tähän ongelmaan.

Manuaalisessa testauksessa testaaja perinteisesti suunnittelee testit, käyttää järjestelmää ja yrittää löytää vikoja. Löydökset raportoidaan ja niiden merkitys arvioidaan. Automaatiotestauksessa laaditaan koneen tai muun työkalun avulla suoritettavia testiskriptejä, jotka tekevät samanlaisia toimintoja järjestelmässä kuin testaaja. (Rytkönen & Kakkonen 2023, 192.) Valmiita skriptejä pystytään suorittamaan paljon nopeammin ja useammin kuin ihmistestaaja pystyy suorittamaan. Kone ei kuitenkaan pysty samanlaiseen luovaan ajatteluun kuin ihminen. Tästä syystä automaatio on paremmin sopiva tarkistusluonteiseen ja toistuvaan testaukseen, kun taas ihmistestaajan työaika kannattaa käyttää saalistusluonteiseen luovuutta ja arviointia vaativaan manuaaliseen testaukseen. (Niittyviita 2019, 134-142.)

Automaatiotestaus ei ole sama asia kuin tekoäly. Skriptejä suorittava kone ei tiedä itse kohdejärjestelmästä mitään, eikä se pysty tekemään päätelmiä ”onnistunut” tai ”epäonnistunut” testin

pohjalta. Tekoäly on viime vuosina tehnyt voimakkaan tulon kaikille ohjelmistokehityksen osa-alueille. Testauksen piirissä uudella tekniikalla on haettu bugien saalistamisen mahdollistamista automaatiolla. Tämä on ollut tähän asti mahdollista vain ihmisen tekemänä. Olemassa olevien testitapausten ylläpito muuttuvissa käyttöliittymissä ja uusien testitapausten luonti ovat myös mahdollisia sovellutuksia tekoälylle. (Pham, Nguyen & Nguyen 2022.) Lisäksi tekoälyä voitaisiin käyttää erilaisten testitulosten analysoinnissa, testitapausten priorisoinnissa ja virheiden ennustamisessa (Islam ym. 2023, 528).

2.3.1 Automaation edut ja ongelmat

Mitä aiemmin virhe havaitaan ja korjataan, sitä halvempaa se on (Bajpai 2012, 8). Automaatiotestauksella voidaan suorittaa suuri määrä testejä jokaisen järjestelmäversion rakentamisen yhteydessä. Käytännössä aina kun koodiin tehdään muutoksia. Näin mahdolliset virheet voidaan havaita helpommin heti muutosten tekemisen yhteydessä ja diagnostiikka sekä korjaaminen voidaan aloittaa aikaisemmin. Aikaisessa vaiheessa suoritettavalla testauksella voidaan myös keksiä uusia huomioon otettavia seikkoja kehitykselle ja testaukselle. Näin voidaan välttää ja ennaltaehkäistä ongelmia, jotka muutoin ilmenisivät vasta myöhemmin kehityksessä. (Niittyviita 2019, 75-79.) Automaatiotestaus on tehokasta, säännönmukaista ja helposti toistettavaa (Rytkönen & Kakkonen 2023, 194).

Automaatiotestaukseen liittyy aina ongelmia ja riskejä. Automaation kehittäminen luo oman kehityshaaran varsinaisen ohjelman kehityksen rinnalle, sillä automaatio itse on ohjelmistokehitystä. Työ täytyy suunnitella kunnolla. Tämä vaatii aikaa, resursseja ja ylläpitoa. (Rytkönen & Kakkonen 2023, 252,) Automaatiolle voi olla myös epärealistisia odotuksia. Voidaan kuvitella, että automaatio löytää lisää uusia virheitä järjestelmästä. Säännöllisesti samalla tavalla ajettu testimassa voi myös antaa harhan, että järjestelmä on virheetön (Bajpai 2012, 8.) Automaatio nähdään usein manuaalisen testauksen korvaajana, mikä ei ole totta. Sillä voidaan vapauttaa testaajia tekemästä manuaalista tarkistusta, mutta kaikkea testaustoimintaa ei voida automatisoida. (Rytkönen & Kakkonen 2023, 257.) Koneella tehdyt tarkistusluonteiset testit tukevat muuta testausta ja antavat lisätyökaluja testaajille, mutta ne eivät poista vielä manuaalisen testauksen tarvetta.

2.3.2 Mitä kannattaa automatisoida?

Testit, jotka voidaan kuvata tarkasti ja jotka toistuvat usein samanlaisina, ovat hyvä kohde automaatiolle (Rytkönen & Kakkonen 2023, 164). Automaatioskriptissä täytyy pystyä tarkkaan kuvaamaan esimerkiksi mitä kone tekee, mitä elementtiä painetaan, mitä tekstiä odotetaan tai tarkistetaan. Tarkasti kuvattavan prosessin toimenpiteet ovat suoraviivaisia kääntää koneelle

annettaviksi komennoiksi. Kun skriptauksen työ suoritetaan kerran ominaisuudelle, jota toistetaan useaan kertaan tai jota täytyy testata toistuvasti, saadaan suurempaa hyötyä tehdylle työlle.

Tarkistusluontoiset tehtävät, kuten regressiotestaus, ovat erityisesti hyviä automatisoitavia (Milad ym. 2014, 194). Regressiotestaus on juuri edellä mainitun kaltaista usein samanlaisena toistuvaa testausta. Kattavaa tarkistusluonteista testausta kannattaa kohdistaa järjestelmän osiin, joihin liittyy raskas todistustaakka (Niittyviita 2019, 59–64). Automaatiotesteillä saadaan helposti dataa, jolla voidaan täyttää todistusvaatimus ja osoittaa, miten kyseinen järjestelmän osa on testattu. Testauksen metsästysluonteista virheiden etsimistehtävää ei vielä pystytä järkevästi automatisoimaan nykyisillä työkaluilla ja se on jätettävä manuaaliselle testaukselle.

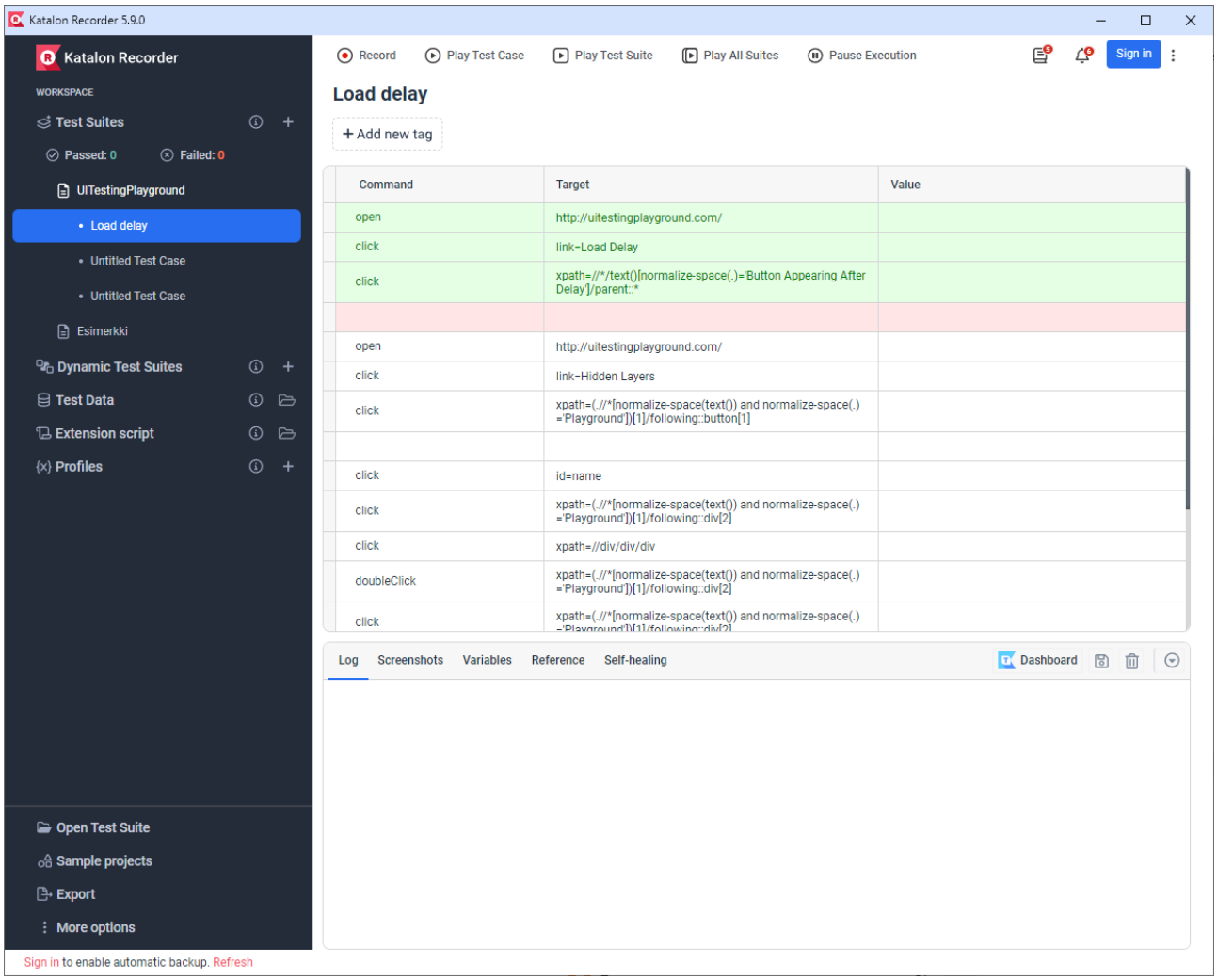
2.4 Automaatiotestauksen skriptauksen tekniikat

Automaatiotestiskriptejä voidaan laatia monella eri tapaa. Skriptit ovat pohjimmiltaan skriptauskielellä, esimerkiksi Pythonilla, laadittuja komentosarjoja, joilla suoritetaan testitapaus ja raportoidaan sen tulokset. Skripteihin voidaan luoda variaatiota muuttujilla, erilaisilla logiikkarakenteilla, toistorakenteilla ja testeistä voidaan laatia monikäyttöisempiä. (Milad ym. 2014, 194–195.) Tekniikoita ja niiden ominaisuuksia kuvataan usein erillisinä, mutta uudemmat tekniikat usein hyödyntävät vanhempien ominaisuuksia ja tekniikoita voidaan käyttää rinnakkain. Esimerkiksi data- ja avainsanakeskeistä skriptausta voidaan helposti yhdistää (Robot Framework Foundation s.a.[g]).

Työn kannalta oleellisimpia skriptauksen tekniikoita esitellään seuraavassa osiossa. Tekniikat rajattiin työssä käytettyihin tai kokeiltuihin tekniikoihin, joita voitiin helposti hyödyntää käytetyillä työkaluilla.

2.4.1 Lineaarinen skriptaus – Linear Scripting

Suoraviivaisin tapa on niin kutsuttu lineaarinen skriptaustekniikka. Tekniikka on nopea ja sen käyttö ei vaadi välttämättä ohjelmointiosaamista. Tässä tavassa testaustyökalu asetetaan nauhoittamaan testaajan toimet kohdejärjestelmässä. Työkalu luo nauhoitteen perusteella skriptin, joka voidaan tämän jälkeen suorittaa uudelleen koneellisesti. (Milad ym. 2014, 195–196.) Katalon Recorder on yksi tällainen työkalu. Se on Google Chromen selainlaajennus verkkosovelluksien testaamiseen. Kuvassa 1 on esitetty Katalon Recorder ja yksi mallitesti.



Kuva 1. Katalon Recorder ja mallitesti

Tällä tekniikalla tehtynä skriptejä kertyy suuri määrä, koska jokaiseen testiin käytännössä luodaan uudet rivit koodia. Skriptit koostuvat suuresta määrästä toimintoja ja kovakoodattua dataa. Tällöin testien ylläpito on hankalaa. Lisäksi testit voivat rikkoutua helposti, jos testin aikana tapahtuu jotain, mitä ei tapahtunut skriptiä tallennettaessa. Testien päivittäminen ja ylläpito vaatii käytännössä testien tallentamisen uudestaan. (Milad ym. 2014, 195.)

2.4.2 Rakenteinen skriptaus – Structured Scripting

Rakenteinen tai jaettu skriptaus on tekniikka, jossa testiskriptit jaetaan pienenpiin osiin ja testit rakennetaan useammasta pienemmästä skriptistä, joita suoritetaan peräkkäin. Yksi skripti suorittaa pienen toiminnon ja voi sisältää erilaista validointia. Näin skripteistä saadaan monikäyttöisempiä ja testejä voidaan rakentaa samoista skripteistä ilman, että koko testiä luodaan alusta asti uudestaan joka kerta. Tämä tekniikka vaatii käyttäjältä ohjelmointiosaamista ja skripteistä voi tulla monimutkaisia ja testidata on usein sidottu edelleen testiskripteihin. (Milad ym. 2014, 195–196.)

2.4.3 Datakeskeinen skriptaus – Data Driven Scripting

Data Driven -skriptaus tavoittelee testien ylläpidettävyyden lisäämistä siirtämällä testidatan skriptistä erilliseen tiedostoon. Tieto haetaan tästä erillisestä lähteestä testin suorittamisen aikana. (Milad ym. 2014, 196–197.) Esimerkiksi järjestelmän kirjautumissivun URL tai käyttäjätunnukset ja salasanat voidaan säilyttää erillisessä tiedostossa, jolloin niitä ei tarvitse päivittää erikseen jokaiseen skriptiin. Testidatan ylläpitäminen helpottuu, kun esimerkiksi muuttujan arvo päivitetään kerran erilliseen tiedostoon tilanteen muuttuessa. Testejä ei myöskään tarvitse tallentaa alusta uudelleen kuten lineaarisessa skriptauksessa. Datakeskeisiä testejä voidaan luoda nopeasti ja pienellä vaivalla saman skriptin ympärille testidataa muuttamalla. (Milad ym. 2014, 196–197.)

Esimerkiksi yhdellä “sisään kirjautumisen” -testiskriptillä voidaan luoda useita testitapauksia, joissa samaa skriptiä hyödynnetään, kun testataan erilaisia variaatiota. Taulukossa 1 kuvataan erilaisia variaatioita, joita yhdellä testitapauksella voidaan testata vaihtamalla muuttujia ja odotettua lopputulosta.

Taulukko 1. Data Driven -skriptauksen esimerkkitestitapauksia

Testitapaus	**Käyttäjätunnus**	**Salasana**	**Lopputulos**
Onnistunut sisäänkirjautuminen	KT1	SS1	Kirjautuminen onnistui
Onnistunut sisäänkirjautuminen	KT2	SS2	Kirjautuminen onnistui
Epäonnistunut kirjautuminen	KT1	NULL	Puuttuva tunnus tai salasana
Epäonnistunut kirjautuminen	KT1	SS2	Virheellinen tunnus tai salasana
Epäonnistunut kirjautuminen	NULL	NULL	Puuttuva tunnus tai salasana
Epäonnistunut kirjautuminen	NULL	SS1	Puuttuva tunnus tai salasana

Yhdellä laaditulla skriptillä saatiin nopeasti aikaan monta eri tarkistustestiä. Tekniikka vaatii usein ohjelmointiosaamista ja lisäksi testidatatiedostojen ylläpito vaatii työtä.

2.4.4 Avainsanakeskeinen skriptaus – Keyword Driven Scripting

Avainsanapohjainen skriptaus on usein luettavissa kuin manuaalisen testauksen "Lue-tee" -tyylinen suoritusohje. Skriptit koostuvat käyttäjälle selvästi luettavissa olevista avainsanoista, jotka sisältävät koodattuna tehtävät, joita ohjelma suorittaa. Näitä avainsanoja voidaan yhdistellä ja lisäksi voidaan luoda käyttäjän omia avainsanoja. Nykypäivän avainsanapohjaiset testauskehykset eivät välttämättä vaadi paljoa ohjelmointiosaamista käyttäjältään, koska valmiita

sisäänrakennettuja avainsanoja löytyy monista kehyksistä ja niiden ympärille on tehty erillisiä kirjastoja, joilla käytettävyyttä voi laajentaa. Käyttäjä voi laajentaa toimintaa rakentamalla lisätoiminnallisuuksia omiin kirjastoihin. (Milad ym. 2014, 198–200.)

Avainsanat voidaan jakaa korkean tai matalan tason avainsanoihin. Matalan tason avainsanat kuvaavat yksittäisiä toimintoja järjestelmässä esimerkiksi “Click" – paina elementtiä, “Get text” – poimi teksti, “Type text” – anna syöte. Korkean tason avainsanat ovat laajempia kokonaisuuksia ja kuvaavat järjestelmän toimintaa yleisemmällä tasolla. (Milad ym. 2014, 199.) Nämä avainsanat voivat koostua useammista matalan tason avainsanoista. Esimerkiksi “Kirjaudu järjestelmään” on avainsana, joka voisi pitää sisällään paljon erilaisia matalan tason avainsanoja: avaa selain, siirry sivustolle, paina elementtiä, anna syötteitä, tarkista tila tai teksti ja niin edelleen. Kuvassa 2 on esitelty yksi avainsanapohjainen testitapaus, jossa on kolme korkean tason avainsanaa ja sen alla on avattu yhden tällaisen avainsanan sisältö.

```
*** Test Cases ***
Run | Debug | Run in Interactive Console
Dynamic ID
    [Documentation]    Dynaaminen (muuttuva) ID
    [Tags]    regression
    Avaa Leikkikentta    ${viive}
    Avaa harjoitus    Dynamic ID
    Paina Nappia    ${dynaaminenNappi}

Load in Interactive Console
Avaa Leikkikentta
    [Arguments]    ${viive}
    new Browser    browser=chromium    headless=${HEADLESS}    slowMo=${viive}
    New Context    viewport=${VIEWPORT}
    New Page    http://uitestingplayground.com/
```

Kuva 2. Avainsanapohjainen testitapaus ja yksi avainsana

Koodausosaamisen tarve vähenee, kun käyttäjän ei tarvitse tietää kuin avainsanat ja miten niitä käytetään (Milad ym. 2014, 200). Olemassa olevia avainsanoja yhdistelemällä voidaan luoda uusia korkean tason avainsanoja ja valmiilla muissa testeissä käytetyillä avainsanoilla voidaan luoda uusia testejä helposti kuin palikoilla rakentamalla. Testiskriptien laadinnassa voidaan käyttää lineaarisen skriptauksen työkaluja. (Milad ym. 2014, 199.) Esimerkiksi edellä mainittua Katalon Recorderia

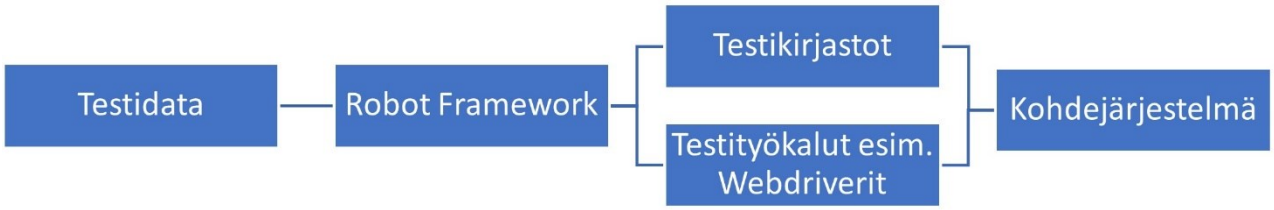
tai Selenium IDE -työkaluja voidaan käyttää apuna skriptien laadinnassa ja muokata skriptit monikäyttöisiksi avainsanakokonaisuuksiksi.

2.5 Robot Framework

Robot Framework on avoimen lähdekoodin avainsanakeskeinen automaatiokehys. Sitä voidaan soveltaa sekä testi- että prosessiautomaatioon. Yhteensä 55 yrityksen ja organisaation muodostama Robot Framework Foundation huolehtii, että kehyksen käyttö voi jatkua ja sitä kehitetään. Viimeisin Robot Frameworkin versio on julkaistu tammikuussa 2024. (Robot Framework Foundation s.a.[a])

Robot Framework on käyttäjälle yksinkertainen viitekehys käyttää. Se tarjoaa tavan laatia testitapauksia yhdenmukaisella taulukkomaisella tavalla. Kehys sisältää useita valmiita kirjastoja ja matalan tason avainsanoja. Käyttäjä voi näitä yhdistelemällä luoda uusia omia korkean tason avainsanoja. Näitä yhdistelemällä voidaan luoda testiskriptejä tai tehtäviä, joita kone suorittaa. (Robot Framework Foundation s.a.[b]). Työkalu on Python-pohjainen, mutta sitä voidaan laajentaa monella eri koodikielellä luomalla omia kirjastoja. Robot Framework voidaan asentaa käytettäväksi erilaisissa ympäristöissä.

Käytännössä Robot Framework ottaa käsittelyyn testidatan, joka sisältää testitapaukset, skriptit, avainsanat ja muuttujat, käsittelee ne komennoiksi ja suorittaa ne testikirjastojen ja muiden työkalujen avulla. Lopuksi Robot Framework luo tapahtuneesta raportit ja lokit selkeinä ja helppokäyttöisinä HTML tiedostoina. (Robot Framework Foundation s.a.[c]). Prosessi kokonaisuudessaan on kuvattu yksinkertaistetusti kuvassa 3. Liitteessä 2 on esimerkit virheitä sisältäneen testiajon raportista sekä onnistuneen testiajon raportista sekä lokista.



Kuva 3. Robot Frameworkin arkkitehtuuri

Avainsanojen dokumentointiin löytyy työkaluja Robot Frameworkiin sisään rakennettuna. Yksi työkaluista on esimerkiksi Libdoc-kirjasto, jolla saadaan luotua dataa avainsanoista. Robot Frameworkin kirjastojen avainsanadokumentaatio on laadittu tällä työkalulla HTML-muotoon. (Robot Framework Foundation s.a.[d])

2.6 Selenium-kirjasto

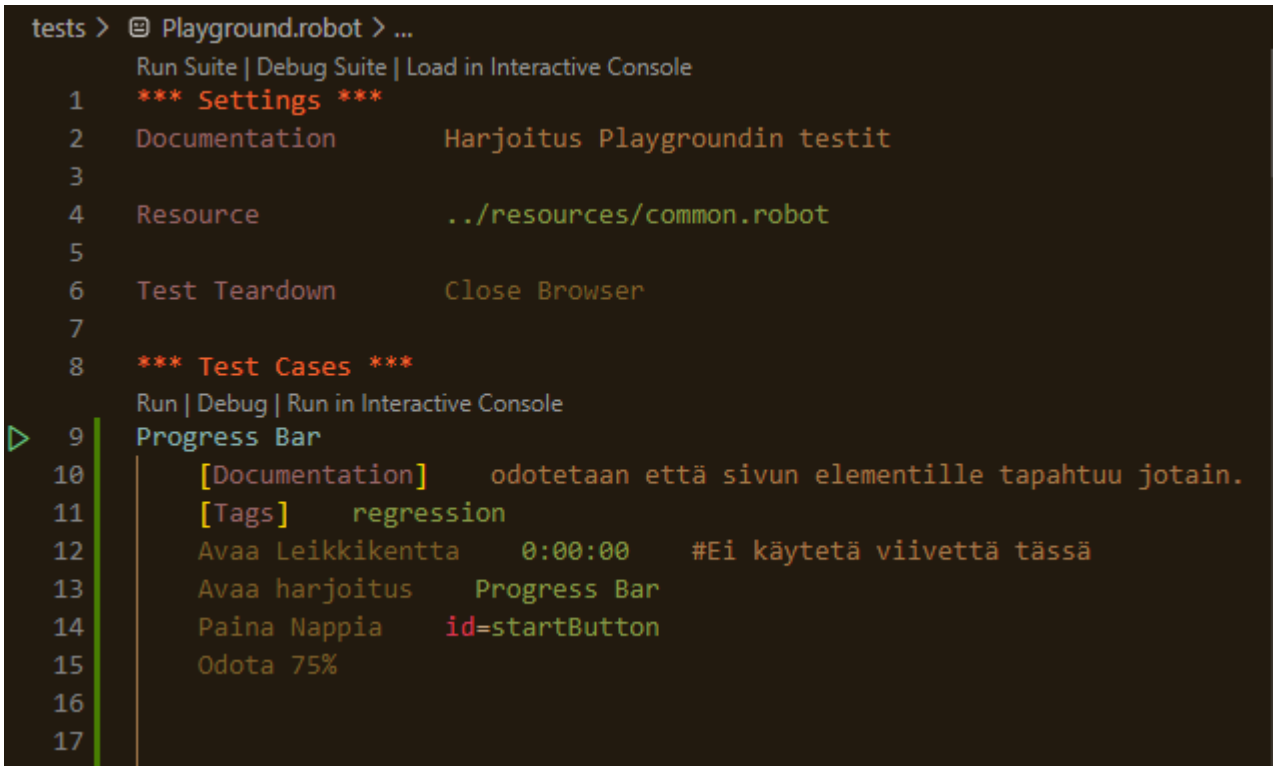
Selenium-kirjasto on Robot Frameworkin laajennuskirjasto verkkosovellusten testaamiseen. Sen käyttö vaatii Robot Frameworkin lisäksi selainajurin testeissä käytettävän selaimen mukaan (Robot Framework Foundation s.a.[e]). Kirjaston avulla käytetään Selenium-testauskehystä. Kirjaston viimeisin versio 6.2.0 sisältää 177 avainsanaa, joilla voidaan vaikuttaa selaimeen.

2.7 Browser-kirjasto

Erilaisia automaatiotestauksen kirjastoja ja viitekehyksiä on paljon. Toinen selaintestauksen automaatioon käytettävä kirjasto on Browser-kirjasto. Se käyttää pohjanaan Playwright-selainautomaatiokehystä. (Robot Framework Foundation s.a.[f]) Browser-kirjastolla Robot Frameworkin käyttäjä voi hyödyntää Playwrightin ominaisuuksia ilman sen syntaksin tuntemista.

Seleniumista poiketen, monet Browser-kirjaston elementteihin vaikuttavista avainsanoista sisältävät sisäänrakennetut odotustoiminnot. Tämä on Playwrightin tuoma ominaisuus. (Playwright s.a.) Esimerkiksi Browserin ”Click”-avainsana sisältää elementin odotuksen oletusarvolla 10 sekuntia. Käytännössä, jos klikattava elementti ei ilmesty näkyviin, ei ole stabiili tai käytettävissä kymmenen sekunnin aikana, testi epäonnistuu. Käyttäjän ei tarvitse tällöin itse välttämättä huolehtia erilaisista tarkistus- tai odotuskomennoista skripteissään. Tämä helpottaa ja nopeuttaa testien laatimista.

Kuvassa 4 on esitetty Browser-kirjastolla laadittu esimerkkitesti. Testi avaa harjoitussivuston, klikkaa linkkiä harjoitustehtävään ja käynnistää harjoituksen painikkeella. Harjoituksen tehtävänä on odottaa, että selaimessa tapahtuu tietty toiminto eli tässä tapauksessa latauspalkki saavuttaa arvon 75 %. Tämän jälkeen painetaan nappia ja pysäytetään harjoitus. Kuvassa näkyy korkean tason omat yhdistellyt avainsanat. Tiedoston Settings-osiossa ei näy kirjaston tietoja, vaan se on sisällytetty erilliseen common.robot-tiedostoon, jossa on myös omat avainsanat.

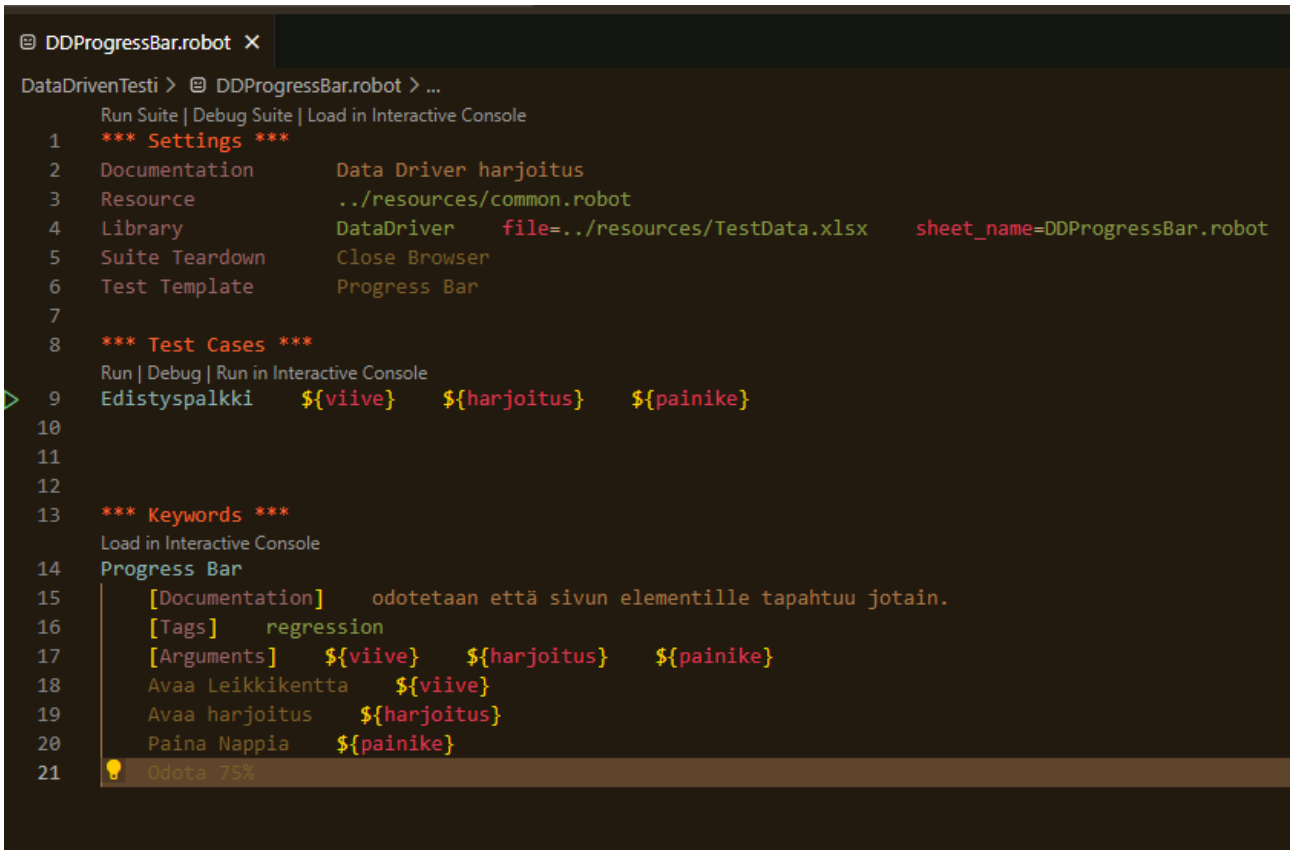


Kuva 4. Progress Bar esimerkkitesti Browser-kirjastolla

2.8 DataDriver-kirjasto

Joissakin tilanteissa voidaan haluta automatisoida monen eri syötteen testaus samassa toiminnossa, esimerkiksi sisäänkirjautuminen erilaisilla käyttäjätunnuksilla. Tällöin voidaan ottaa käyttöön DataDriver-kirjasto, jotta ei jouduta käynnistämään samaa testiä erilaisilla muuttujan arvoilla monta kertaa tai luomaan samaa testiä moneen kertaan eri arvoilla testitiedostoihin. Kirjaston avulla voidaan ajaa samaa testiä monta kertaa eri muuttujien arvoilla ja tarkistaa helposti sama toiminne eri arvoilla. Tätä testitapausten ja muuttujien dataa voidaan kirjaston avulla hallinnoida Excel.csv tai Excel.xlsx tiedostojen kautta. DataDriver-kirjasto asennetaan ja sitä käytetään kuten muitakin Robot Frameworkiä laajentavia kirjastoja, mutta se ei tuo yhtään omaa avainsanaa. Se aktivoituu ennen suoritettavaa testisarjaa, tarkistaa mitkä muuttujat testisarjaan on tulossa ja poimii vastaavat määrätystä Excel-tiedostosta Robot Frameworkin käyttöön. (Robot Framework Foundation s.a.[9])

Kuvassa 5 esitellään kuvan 4 testi, joka on muutettu käyttämään Data Driven -syntaksia ja DataDriver-kirjastoa. Data Driven -syntaksia on kuvattu tarkemmin kohdassa 2.8.5 Testitapausten kirjoitustyylit.



Kuva 5. Progress Bar esimerkkitesti DataDriver -kirjastolla ja Data Driven -syntaksilla

2.9 Skriptauksen hyvät toimintatavat

Testiautomaatio on ohjelmistokehittämistä samaan tapaan kuin itse järjestelmän koodin kirjoittaminen. Automaation kirjoittamiseen voidaan soveltaa samoja periaatteita kuin muunkin koodin kirjoittamiseen. Ylläpito on merkittävä kuluerä kaikessa ohjelmoinnissa, joten siksi testien ylläpidettävyyteen kannattaa panostaa. Tarve testien päivittämiseen voi tulla muutoksista vaatimuksiin. Esimerkiksi syötteen A ei pidäkään enää tuottaa tulostetta B. Muutokset järjestelmän toiminnassa voivat myös pakottaa testien päivittämiseen. Uusi elementti tai Pop-up-ikkuna käyttöliittymässä voi rikkoa olemassa olevia testejä. (Emery 2009, 1.)

Hyvillä ja johdonmukaisesti käytetyillä sovituilla toimintatavoilla voidaan parantaa automaatiokoodin ja skriptien ylläpidettävyyttä ja helpottaa sen kirjoittamista. Testien tulisi olla itsenäisiä ja uniikkeja. Niiden tulisi testata sellaista, mitä ei muualla testata. Testeillä täytyy saada uutta oikeaan aiheeseen liittyvää tietoa järjestelmästä. (Rytkönen & Kakkonen 2023, 259–260.)

2.9.1 Nimeäminen

Yleisesti voidaan sanoa, että nimeäminen pitäisi tehdä mahdollisimman kuvaavasti. Liian tarkkaan kuvaukseen ei kuitenkaan kannata mennä. Pelkästä nimestä pitäisi saada kuva, mitä testisarja

käsittelee, mihin testitapaus kohdistuu, mitä avainsana tekee tai mitä muuttuja sisältää. (Klärck 2016.) Parempi tapa on nimetä testitapaus esimerkiksi "onnistunut sisäänkirjautuminen" kuin "Syötetään validi käyttäjätunnus ja salasana ja klikataan kirjaudu-painiketta". Samat nimeämiskäytännöt koskevat sekä testisarjoja, testitapauksia että avainsanoja: Kerro selkeästi mitä tehdään, ei miten tehdään. Säännönmukaisuus nimeämisissä auttaa myös luettavuutta ja ymmärrettävyyttä. Kaikki nimeäminen kannattaa tehdä samalla logiikalla. Selkeällä nimeämisellä testi pitäisi voida suorittaa myös manuaalisestitestaamalla. (Klärck 2016.) Jos nimeäminen on toteutettu hyvin, ei erillistä dokumentaatiota välttämättä tarvita (Klärck 2014).

2.9.2 Dokumentointi

Robot Frameworkissa on mahdollista käyttää testisarjojen, testitapausten tai avainsanojen yhteydessä erilaisia elementtejä dokumentointiin. ”[Documentation]”-asetuksella voidaan antaa pidempää kuvausta. ”Tag”:it ovat yhden sanan tunnisteita, joilla voidaan helposti kuvata, mitä rakenne käsittelee (Emery 2009, 2–3). Dokumentointia voidaan lisätä testitiedostoihin selvyyden vuoksi ja kuvaamaan taustoja ja syitä testien takana, poikkeavia asioita tai erityisiä yksityiskohtia. Testin sisältö tulisi olla avainsanojen nimien pohjalta ymmärrettävissä. Ylläpidettävyyden kannalta liiallinen ja tarpeeton dokumentaatio on yksi lisätyö. Ylimääräistä kuvausta ole pakko tai kannattavaa antaa, jos dokumentaatio ei ole välttämätön testin selkeyden vuoksi. (Klärck 2016.)

Yksittäisille testitapauksille dokumentoinniksi voi riittää Tagien käyttö. Käyttäjän luomien omien avainsanojen dokumentointi ei ole välttämättä tarpeen, jos avainsana on yksinkertainen ja nimetty selkeästi. Argumentteja ja palautettavia arvoja voi olla tarpeen kuvata tarkemmin ymmärrettävyyden vuoksi. (Klärck 2016.) Koodia kommentoimalla voi testeistä tulla sekavia ja hankalia ylläpitää ja ymmärtää. Huonoa nimeämistä ei kannata paikata erillisellä dokumentaatiolla.

2.9.3 Testisarjan rakenne

Testisarjan on joukko testitapauksia ISO/IEC/IEEE 29119-1:2022, 15). Testisarjan sisällä olevien testitapausten tulisi loogisesti liittyä toisiinsa. Ei ole esimerkiksi järkevää testata tavaralogistiikan toiminteita kassaohjelmaa käsittelevässä testisarjassa. Yleissääntönä voidaan pitää 10 testiä yhtä testisarjaa kohden. Data Driven -syntaksia käytettäessä testitapauksia voi olla enemmän testisarjassa. (Klärck 2016.)

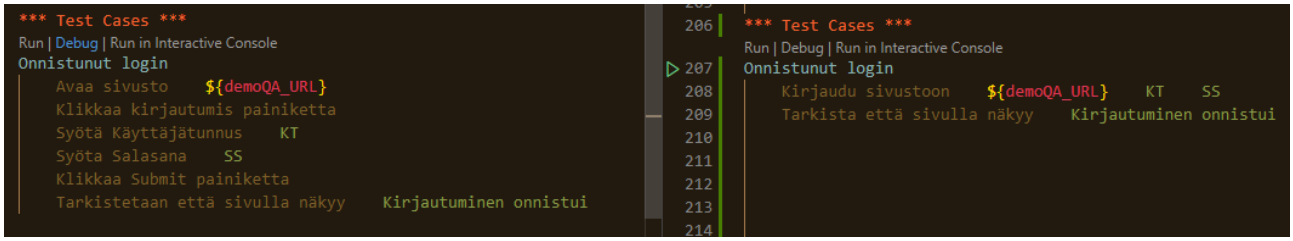
Testien välillä ei tulisi olla riippuvuuksia sarjan sisällä tai sarjojen välillä. Jos testi C riippuu testeistä A ja B, niin virheiden sattuessa ketjun alussa menevät loputkin testit rikki. (Klärck 2016.) Tämä voi aiheuttaa ylläpitotarvetta koko ketjuun. Lisäksi testien ajaminen voi hidastua, kun täytyy odottaa edellisten testien valmistumista.

2.9.4 Testitapauksen rakenne

Testitapaus on testin suorittamista ohjaava dokumentointi testin ennakkovaatimuksista, tehtävistä toimenpiteistä ja odotetusta lopputuloksesta. Testillä tarkoitetaan toimintaa, jossa järjestelmää tai sen osaa tarkastellaan ja arvioidaan. (ISO/IEC/IEEE 29119-1:2022, 10.) Testitapaukset kannattaa pitää yksinkertaisena ja helppona ymmärtää. Suoritettavasta testistä täytyy pystyä vaivatta sanomaan, mitä sillä tarkistetaan ja milloin testi onnistuu tai epäonnistuu. Kannattaa pyrkiä luomaan testitapaukset niin, että ne ovat toistettavissa helposti muissa ympäristöissä ja laitteissa. (Rytkönen & Kakkonen 2023, 259–260.) Yhden testitapauksen tulisi kuitenkin testata vain yhtä asiaa. Tämä asia voi olla pieni osa, kuten sisäänkirjautumisen kaltainen yksittäinen ominaisuus, tai laajempi end-to-end-tyyppinen kokonaisuus, jossa testataan jokin toiminne alusta loppuun (Klärck 2016.).

Testitapausten tulee olla selkeitä ja ymmärrettäviä. Liian tarkkoihin yksityiskohtiin ei kannata mennä testitapauksen kuvauksessa tai sen avainsanoissa. Avainsanoja ei kannata käyttää testitapauksen alla tarpeettoman suurta määrää. Testin luettavuuden kannalta suuremmat logiikat ja matalan tason avainsanat kannattaa sisällyttää korkean tason avainsanoihin. Erilaisia koodirakenteita, esimerkiksi toistorakenteita tai ehtolauseita, voi käyttää, mutta jos tarvitsee ohjelmoida monimutkaisempaa logiikkaa, kannattaa sille luoda oma kirjasto. (Klärck 2016.)

Kuvassa 6 on esimerkki onnistuneen sisäänkirjautumisen testistä kahdella tavalla muotoiltuna. Vasemmalla on testi, jossa on käytetty tarpeettoman tarkkoja avainsanoja isompi määrä kuin esimerkin toisessa testissä. Oikealla on sama testi laadittuna niin, että viisi matalamman tason yhdisteltyä avainsanaa on sisällytetty yhteen korkean tason avainsanaan. Testi on paljon helpommin luettavissa ja siitä näkee yhdellä silmäyksellä, mitä tehdään ja mikä on haluttu lopputulos.



Kuva 6. Esimerkit sisäänkirjautumisen “Onnistunut login” -testin muotoilusta

Testitapauksiin kannattaa sisällyttää aina asianmukaisia tarkistuksia, joilla varmistetaan testin olevan oikeassa vaiheessa tai tehdyn toiminnon onnistuminen (Klärck 2014). Tilanteessa, jossa testin viimeinen tapahtuma on painikeen klikkaaminen, voidaan saada virheellisesti onnistunut ”Passed”-tulos, jos viimeinen klikkaus on onnistunut, mutta järjestelmä kaatuukin toiminnon seurauksena. Tällainen tilanne voidaan välttää käyttämällä tarkistuksia. Robot Frameworkistä löytyviä ”Test setup” ja ”Test Teardown” -asetuksia kannattaa käyttää (Klärck 2014). Näillä voidaan

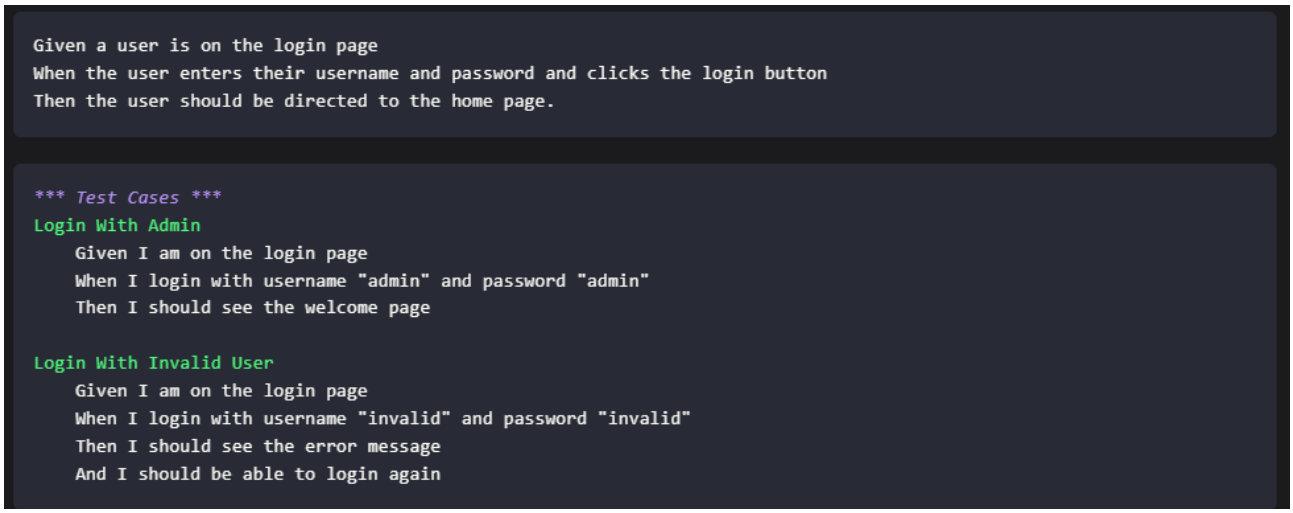
oletuksena alustaa testi valmiilla tai käyttäjän omalla avainsanalla ja testin lopuksi esimerkiksi sulkea avatut selaimet.

2.9.5 Testitapausten kirjoitustyylit

Robot Frameworkin testitapaukset koostuvat usein listasta avainsanoja, joka muistuttaa työnkulkua tai toimintaohjetta siitä, mitä kone tekee. Manuaalitestaaja voi periaatteessa suorittaa saman testin avainsanojen perusteella aivan kuten normaalin toimintaohjeen. Testitapauksia voidaan kirjoittaa monella eri tavalla. Kaksi yleisintä tapaa ovat avainsanakeskeinen- ja datakeskeinen tyyli. Näille löytyy erilaisia alalajeja. (Klärck 2016.) Avainsanakeskeinen tapa on hyvin yleispätevä ja sitä voidaan käyttää testiin kuin testiin. Datakeskeistä tyyliä kannattaa käyttää tilanteissa, joissa joudutaan testaamaan samaa asiaa monta kertaa samalla testillä, mutta eri arvoilla.

Avainsanakeskeinen tapa on koostaa testit useista avainsanoista. Niissä yleensä alustetaan lähtötilanne, tehdään toiminto ja sitten varmistetaan, että toiminto on onnistunut. (Robot Framework Foundation s.a.[h]) Testit ovat yleensä lyhyitä, maksimissaan kymmenen korkean tason avainsanan kokonaisuuksia. Testien tasolla ei ole suositeltavaa käyttää monimutkaista logiikkaa. Sen sijaan erilaiset logiikat kannattaa sijoittaa avainsanojen sisään tai kirjastoihin. (Klärck 2016.) Tämä helpottaa testien luettavuutta ja ymmärrettävyyttä. Testit voivat olla perinteisiä avainsanapohjaisia testejä: "tee asia 1", ”tee asia 2", "tarkista että X tapahtuu” (Klärck 2016).

Behaviour Driven, eli käyttäytymislähtöinen tyyli, painottaa järjestelmän toimintaa ja vaatimuksia. Nämä avataan helposti ymmärrettävässä ja testattavassa muodossa. Vaatimukset ja testit esitetään usein Gherkin-tyyliin muodossa "olosuhteessa X, kun tehdään Y, tapahtuu Z" - "Given, When, Then". (Farooq ym. 2023, 88008–88009.) Klassinen esimerkki vaatimuksesta on: "Kun kirjautumissivulla käyttäjä syöttää oikean käyttäjätunnuksen ja salasanan - kirjautuminen onnistuu". Kuvassa 7 on kuvattu esitystapaa vaatimuksille ja testitapauksille käyttäytymislähtöisellä tavalla.



```
Given a user is on the login page
When the user enters their username and password and clicks the login button
Then the user should be directed to the home page.
```

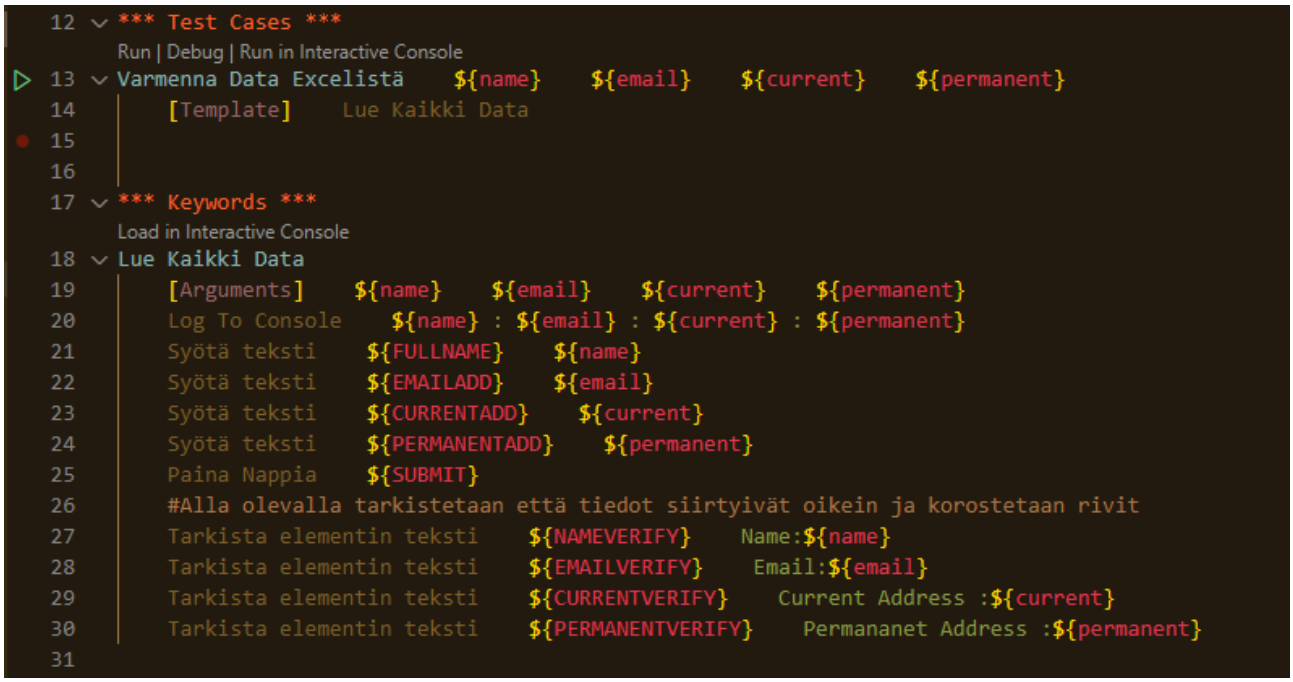
```
*** Test Cases ***
Login With Admin
    Given I am on the login page
    When I login with username "admin" and password "admin"
    Then I should see the welcome page

Login With Invalid User
    Given I am on the login page
    When I login with username "invalid" and password "invalid"
    Then I should see the error message
    And I should be able to login again
```

Kuva 7. Käyttäytymislähtöinen vaatimus ja testitapaukset (Robot Framework Foundation s.a.[i])

Käyttäytymislähtöinen lähestymistapa vähentää epäselvyyttä kehityksen aikana ja helpottaa käyttötapausten ja vaatimusten dokumentointia. Gherkin-tyyli on lyhyt, kuvaava ja suoraviivainen tapa kirjoittaa testejä. Sillä saadaan esitettyä selkeästi, mitä tapahtuu, kun tehdään toiminne tietyssä tilanteessa. (Klärck 2014.) Lopputulos on ymmärrettävä sekä käyttäjälle, koodaajalle että testaajalle ja väärinymmärrysten riski pienenee (Farooq ym. 2023, 15). Gherkin on yksi avainsanapohjaisten testitapausten muoto (Klärck 2016).

Datakeskeinen tyyli eli Data Driven -syntaksi. Joissain tilanteissa samaa testiä tarvitsee ajaa monta kertaa eri muuttujilla. Ei ole järkevää kopioida samaa testitapausta useaan paikkaan ja vaihtaa muuttujien arvoja. on hyvä tapa kirjoittaa testejä tällaisessa tilanteissa. (Klärck 2014.) Erilaiset variaatiot esimerkiksi sisäänkirjautumisessa voidaan testata tällä näppärästi ilman tarvetta kirjoittaa erillinen testi joka variaatiolle. Jokaiselle testitapaukselle luodaan yksi työnkulku-tyyppinen testi, joka toimii korkean tason avainsanana. Tämä avainsana määritellään testitapauksille testipohjaksi eli ”templateksi”. Eri muuttujat voidaan asetella ja ryhmittää testin luettavuuden vuoksi sarakkeisiin testitiedostossa. Kuvassa 8 on esitetty Data Driven -syntaksin testitapaus ”Varmenna data Excelistä” ja sen mallipohjana toimiva avainsana ”Lue Kaikki Data”.



Kuva 8. Testi Data Driven -syntaksilla ja DataDriver kirjastolla toteutettuna

Eri kirjoitustavat ovat

2.9.6 Muuttujat

Muuttujiin kannattaa säilöä testeissä usein toistuvat asiat. Monimutkaiset tai pitkät arvot kannattaa myös asettaa muuttujiin. Tällaisia voivat olla esimerkiksi elementtien valitsimet eli selektorit tai lokaattorit. (Klärck 2016.) Esimerkiksi XPath-arvot, joilla osoitetaan elementtejä tai URL:eja, kannattaa hallinnoida muuttujissa. Muuttujia voidaan asettaa testisarjoissa, testitapauksissa testien aikana tai globaalisti. Erillinen muuttujatiedosto voi helpottaa muuttujien hallinnointia ja ylläpidettävyyttä.

2.9.7 Odottaminen

”Sleep” on Robot Frameworkin valmis avainsana, jolla testi käsketään odottamaan määrätyn ajan (Robot Framework 2011). Avainsana on helppo käyttää, mutta se ei ole optimaalinen moneen tilanteeseen. Jos halutaan odottaa jonkin toiminteen toteutumista ennen seuraavaa toiminnetta ”Sleep” on asetettava usein varmuuden vuoksi tarpeettoman suureksi, jotta se varmasti odottaa riittävän pitkään. (Klärck 2016.) Tällöin testien suorittaminen venyy ja esimerkiksi tietoliikenneongelmien vuoksi testi saattaa epäonnistua toiminteen kestäessä tavallista pidempään. Parempi tapa on käyttää erilaisia valmiita ”Wait…”-avainsanoja. Näillä voidaan määrittää joustavasti, että testi odottaa tiettyä tapahtumaa ja etenee heti, kun ehto toteutuu. ”Wait”-tyyppisille odotuksille saa usein määritettyä tietyn takarajan, jottei testi jää jumiin odottamaan toimintoa, joka ei koskaan

valmistu. ”Sleep”-avainsanaa voi käyttää kuitenkin helpottamaan esimerkiksi testien debuggausta tai testiajojen seurantaa.

3 Työn toteutus

Toimeksiannon pohjalta lähdettiin toteuttamaan verkkopohjaisen järjestelmän automaatiotestausta. Tässä osiossa kuvataan opinnäytetyön taustat ja toteutuksen vaiheet.

3.1 Tausta ja lähtötilanne

Toimeksiantajalla oli meneillään juuri aloitettu regressiotestauksen automatisointiprojekti, johon otettiin mukaan seitsemän toimeksiantajan ylläpitämistä järjestelmistä. Kokonaisuudessaan tämän kattoprojektin tavoitteina oli:

1) Kehittää tehokas testiautomaatiokehys Robot Frameworkin avulla
2) Automatisoida testitapaukset valituille järjestelmille
3) Hallinta ja seurata testiskriptejä Gitlabissa
4) Luoda ylläpitosuunnitelma projektin jatkokehityksen tukemiseksi

Tarkoituksena oli siirtää korkean prioriteetin ja usein toistettavien manuaalisten regressiotestien suoritus automaatiolle. Työlästä manuaalista tarkistelua vähentämällä pyrittiin parantamaan testauksen tehokkuutta ja pienentämään inhimillisten virheiden mahdollisuutta. Vapautuneita resursseja oli tarkoitus ohjata järjestelmine ylläpitotiimeissä muille kriittisille tehtäville.

Työkalujen valinta oli tehty projektin alkuvaiheessa huomioiden jo käytössä olleet työkalut, testauksen kohdejärjestelmät sekä työkalujen hinnat. Robot Framework ja Selenium-kirjasto sopivat verkkoselainpohjaisten järjestelmien testaukseen ja Gitlab oli valmiiksi käytössä toimeksiantajalla. Molemmat työkalut ovat käyttäjälleen varsin kustannustehokkaita. Projektiin palkatulla automaatioinsinöörillä oli aiempaa kokemusta Robot Frameworkin käytöstä.

Projektiin oli otettu korkeakouluharjoittelijoita ja automaatioinsinööri, jotka yhdessä suorittivat automaatioskriptien luomisen järjestelmien ylläpitäjiltä saatujen testitapausten pohjalta. Sain toimeksiantajalta vastuulleni yhden projektissa mukana olleista järjestelmistä. Tietosuojasyistä kyseistä kohdejärjestelmää kutsutaan tässä työssä nimellä ”Lambda”. Lambda-järjestelmän automaatioon osallistui minun lisäkseni vain kattoprojektiin palkattu automaatioinsinööri.

Lambda on verkkoselainpohjainen järjestelmä, joka oli juuri siirtynyt tuotantoon ja ylläpitoon. Sen viimeiset käyttöönotot tehtiin vuoden 2023 viimeisellä kvartaalilla. Käyttäjiä tällä räätälöidyllä valmisohjelmistolla on noin 9000. Järjestelmän ylläpidosta ja kehityksen linjoista vastaa toimeksiantaja, mutta varsinainen kehitystyön suorittaa toimittaja. Toimeksiantajan vastuulla on Lambdan uusien versioiden ja muiden muutosten hyväksymistestaus ennen tuotantoon siirtämistä. Lisäksi

ylläpitotiimi tekee tutkivaa testausta käyttäjäpalautteen ja tikettien pohjalta, sekä yleistä laadunvalvontaa ja regressiotestausta valmiiden testitapausten pohjalta.

Lambdan testaus oli projektin alussa täysin manuaalista. Järjestelmä oli kuitenkin otollinen kohde automaatiotestauksen lisäämiselle, koska sen käyttöliittymän ei kohdistunut jatkuvia muutoksia. Lisäksi järjestelmän monet toiminnot ovat suoraviivaisia ja kuvattavissa automaatiotestauksen vaatimalla tarkkuudella. Koska järjestelmä on isolle osaa käyttäjistä kriittinen päivittäisessä työssä, sen toimintavarmuuden lisääminen jatkuvalla testaamisella oli tervetullut uudistus. Pienelle ylläpitotiimille koneellinen testausapu puolestaan loi helpotusta erityisesti versiopäivitysten aikaiseen kiireeseen.

3.2 Työn tavoite

Opinnäytetyö oli osa suurempaa kattoprojektia. Työn tarkoitus oli toteuttaa projektin tavoitteita kohdejärjestelmään, eli opinnäytetyössä päätavoitteena oli käynnistää automaatiotestaus toimeksiantajan ylläpitämään Lamba-järjestelmään. Käytännössä tarkoituksena oli tehdä automaatiotestit järjestelmän oleellisimpiin toimintoihin. Testit olivat funktionaalisia regressiotestejä, eli niillä testattiin järjestelmän toiminnallisia ominaisuuksia varmistaen, etteivät muutokset muualla ole rikkoneet kohdeominaisuuksia. Testauksen ensimmäiseen vaiheeseen valittavien toimintojen rajauksesta sovittiin järjestelmän palveluomistajan ja ylläpitotiimin kanssa. Automaatiotestien toivottiin vähentävän manuaalisen työn tarvetta uusia versioita testattaessa. Lisäksi automaatiosta haluttiin apua ympäristön ja palvelimien päivityksien onnistumisen varmistukseen. Testeille luotiin Gitlab-ympäristö testidatan hallinnointia ja ylläpitoa varten. Samalla laadittiin ohjeet testien käytöstä sekä niiden laatimisesta ja ylläpidosta.

Lambdan ylläpitotiimiltä tuli lisäksi toive saada testeille yksinkertainen käyttöliittymä, jonka avulla testien suorittaminen olisi helppoa. Ohjelmointikokemukseni oli vielä suhteellisen rajallista, joten emme päättäneet toteuttaa kokonaan alusta asti koodattavaa graafista käyttöliittymää. Muistin aiemmin nähneeni Exceliä käytettävän apuna Robot Frameworkin testeissä ja tästä ajatuksesta lähdettiin suunnittelemaan käyttöliittymää tiimille.

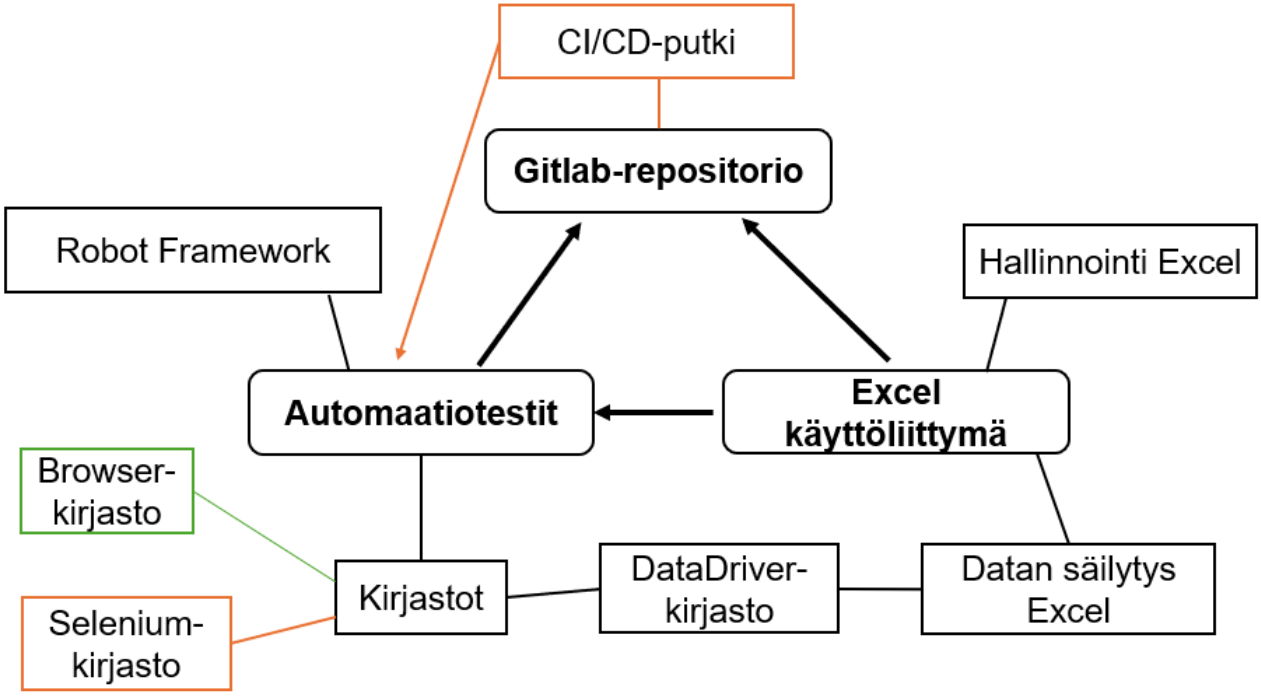
3.3 Suunnittelu

Suunnittelu aloitettiin automaatioinsinöörin ja Lambdan palveluomistajan kanssa pidetyllä palaverilla, jossa demonstroitiin testaustyökaluja. Sovimme suuntaviivat, miten projekti järjestelmän osalta toteutetaan. Työkaluiksi valittiin kattoprojektin tarjoamat työkalut: Robot Framework -automaatiokehykseksi ja Selenium selaimen käsittelykirjastoksi. Minun tehtäväkseni tuli alussa kirjata automatisoitavista toiminteista askelkohtaiset käyttötapauskuvaukset, joiden pohjalta lähdimme sitten luomaan automaatiokehittäjän kanssa Robot Framework -skriptejä.

Ylläpitotiimin kanssa laadimme listan Lambdan toiminteista, jotka tulisi ottaa automaation piiriin. Lista koostui järjestelmän savutestien tapauksista sekä normaalin käytön kannalta kriittisimmistä toiminteista, joiden toimivuus halutiin automatisoiduilla regressiotesteillä turvata. Savutestiksi kutsuttiin nopeaa testisarjaa, jolla tarkistettiin ympäristön palautuminen esimerkiksi palvelin käynnistysten jälkeen. Tässä testisarjassa tarkistettiin 10 kriittisen ominaisuuden toiminta. Järjestelmän hankinnan aikana oli laadittu lista oleellisista ominaisuuksista suorituskykytestejä varten. Tätä listaa pystyttiin hyödyntämään testien valinnassa. Toteutettavia testejä valittiin ensimmäiseen vaiheeseen yhteensä 11.

Toivottu käyttöliittymä päätettiin toteuttaa käyttäen kahta Excel-työkirjaa ja Robot Frameworkin DataDriver-kirjastoa. Ajatuksena oli, että toinen työkirja olisi varsinainen hallinnointiin tarkoitettu käyttöliittymä ja tämän tiedot linkattaisiin datatyökirjaan niin, että DataDriver-kirjastolla saadaan poimittua tiedot testeille. Testien suorittamisen käynnistykseen käytettäisiin VBA-skriptejä.

Gitlabin CI/CD-putkea pyrittiin hyödyntämään ja ajastetuksi ajoksi suunniteltiin yhden viikon välein suoritettavia ajoja. Järjestelmään ei tehty päivittäin muutoksia, vaan uudet versiot tulivat ympäristöihin toimittajan kanssa sovittuina ajankohtana. Säännöllisellä ajastetulla ajolla koettiin kuitenkin olevan hyötyä esimerkiksi ympäristöjen ja palvelimien tilan seurannassa. Kuvassa 9 esitetään graafisesti alkuperäinen suunnitelma mitä elementtejä työn toteutus tulisi sisältämään ja miten ne liittyvät toisiinsa. Punaisella viivalla on merkitty suunnitelman osat, jotka lopulta jäivät pois toteutuksesta ja vihreällä merkitty alkuperäiseen suunnitelman ulkopuolelta tulleet lisäykset. Lisäyksiä ja poistoja käsitellään työn etenemisen osiossa 3.4.



Kuva 9. Suunnitelma automaatiotesteille ja sen komponenteille

3.3.1 Riskiarvio työn alkaessa

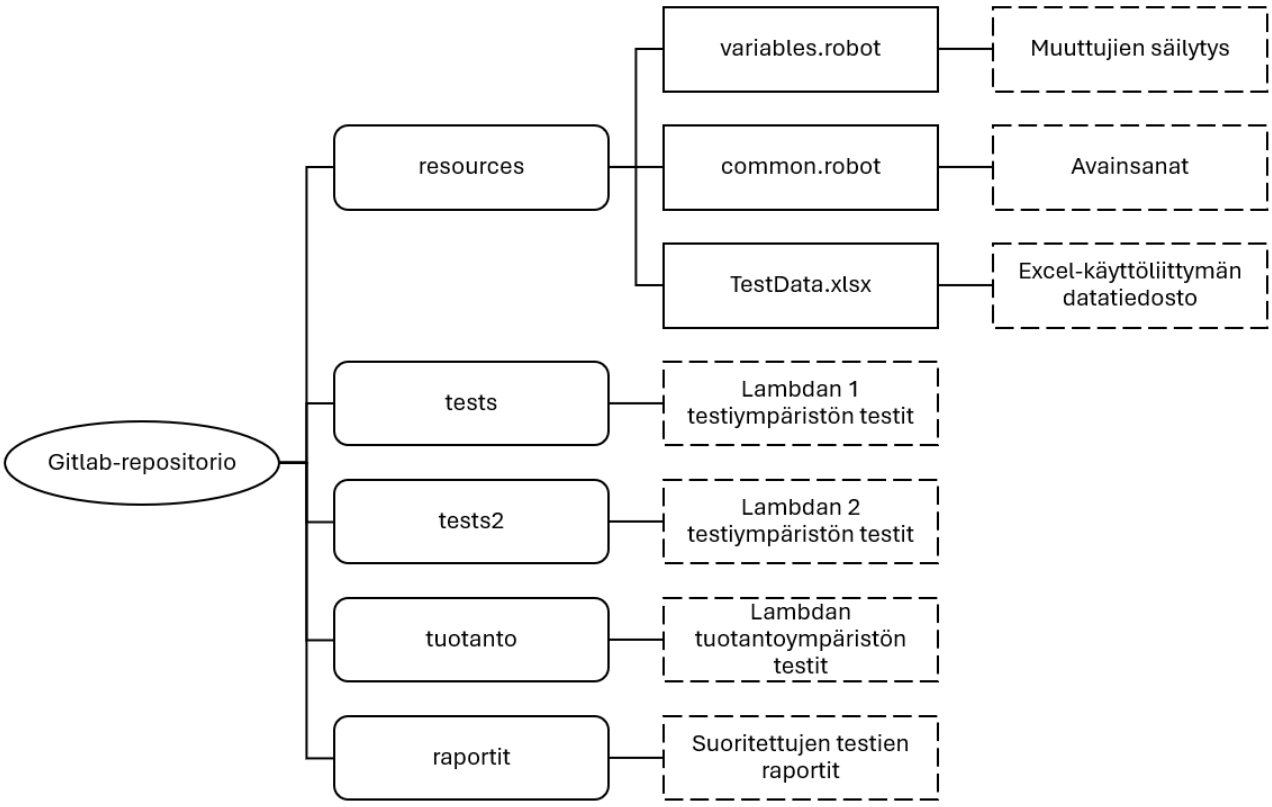
Oma aiempi kokemus automaatiosta rajoittui yhteen opintojaksoon ja demonäytökseen toisen järjestelmän testiympäristöstä. Suurimpana riskinä näin projektille oman osaamiseni automaation ja käytettävien työkalujen osalta. Riskin minimoimiseksi tein työn osiot ensin henkilökohtaisella työasemalla harjoitus mielessä julkisiin osoitteisiin. Näin sain tuntumaa tekemiseen ja välineisiin ennen oikeassa testiympäristössä työskentelemistä. Lisäksi projektissa mukana ollut automaatioinsinööri pienensi jäännösriskiä. Ongelmia kohdattaessa apua oli saatavilla muualtakin kuin Robot Frameworkin foorumeilta. Projektiin käytettävä aika sivutoimisella opiskelijalla oli myös riski. Päätoimisen työn ja perhevelvoitteiden lomassa riskiin varauduttiin väljähköllä aikataulutuksella.

3.4 Työn eteneminen

Työtä lähdettiin suorittamaan seuraavassa järjestyksessä: Gitlab-ympäristön perustus, ensimmäisen vaiheen testien käyttötapausten laadinta, testiskriptien laadinta, Excel-käyttöliittymä ja testien muokkaus DataDriver-kirjastoa käyttäviksi ja lopuksi ohjeistuksen sekä dokumentoinnin laadinta. Järjestys oli suuntaa antava ja osa tehtävistä limittyi päällekkäin. Esimerkiksi dokumentaatiota laadittiin koko projektin ajan.

3.4.1 Gitlab-ympäristö

Kattoprojektille oli luotu valmiiksi oma Gitlab-ympäristö. Tämän ympäristön alle automaatioinsinööri lisäsi oman Gitlab-projektin Lambda-järjestelmän käyttöön. Automaatiotestit ja muu testaukseen liittyvä materiaali vietiin tähän repositorioon. Gitlabiin luotiin kansiorakenne, jossa eri testiympäristön testeille oli omat hakemistot selvyyden vuoksi. Resursseille, kuten muuttujille ja käyttäjän omille avainsanoille, lisättiin oma resources-hakemisto ja raporteille oma kansio. Kansiorakennetta on esitelty kuvassa 10. Kansiot on kuvattu pyöristetyillä suorakulmiolla, tiedostot tavallisilla suorakulmioilla ja kansioiden -ja tiedostojen sisällön kuvaus on esitetty katkoviivaisilla suorakulmioilla.



Kuva 10. Gitlab-repositorion ja kansiorakenne ja sisältö

Lambdan osuutta oli tekemässä itseni lisäksi vain automaatioinsinööri, joten emme katsoneet tarpeelliseksi sopia erillistä käytäntöä tai prosessia muutosten viennistä repositorioon.

Dokumentaation, testien ja muun materiaalin säilyttäminen repositoriossa onnistui ilman ongelmia, mutta CI/CD-putki osoittautui haasteeksi. Lambda-järjestelmä vaatii suljetun sisäverkon käyttämistä, eikä testejä saatu ajettua suoraan Gitlabin kautta. Tilannetta lähdettiin ratkomaan erilliselle

sisäverkon palvelimelle perustettavalla Gitlab-runnerilla. Tämä työ jäi kesken ja ajastettuja ajoja ei saatu projektissa toteutettua. CI/CD-putken käyttöönotto jäi jatkokehitykselle.

3.4.2 Automaatiotestit

Testien perustana käytettiin Robot Frameworkin Selenium-kirjaston valmiita avainsanoja. Yksittäisiä avainsanoja käytettiin Robot Frameworkin asennuksen mukana tulleista standardikirjastoista Builtin, DateTime ja Dialogs. Testit jaoteltiin omiin kansioihinsa niiden suoritusympäristön mukaan. Testit käyttivät yleisien muuttujien keskitettynä säilytyspaikkana variables.robot -tiedostoa. Käyttäjien laatimat omat avainsanat sijoitettiin common.robot -tiedostoon. Kansiorakenne on sama, jota käytettiin Gitlab-ympäristössä ja sitä esiteltiin edellä kuvassa 10. Testit laadittiin aluksi avainsanakeskeisellä tyylillä mutta työn edetessä niitä muutettiin yhdistämään avainsana- ja datakeskeisiä skriptauksen tekniikoita. Testeistä pyrittiin tekemään monikäyttöisiä laatimalla uudelleen käytettäviä avainsanoja rakenteisen skriptauksen tapaan.

Testien laatimiseksi tutustuin Lambdan toimintaan ja automatisoitaviksi valittuihin ominaisuuksiin. Laadin valituista testeistä ”Lue-tee” -tyyppiset vaihe vaiheelta etenevät käyttötapaukset rakenteella:

”Avaa sivusto Lambdan URL:iin
Paina nappia ’Avaa toiminto X’
- Lambda avaa näytön X
Valitaan kenttään ’Y’ arvo ’Z’
Klikataan ’Hyväksy’-painiketta
- Saadaan ilmoitus onnistuneesta toiminnosta X”

Toimitin projektin automaatioinsinöörille yhteensä kuusi kappaletta tällaisia 15–30 kohdan aukikirjoitettuja käyttötapauksia, joiden pohjalta laadittiin ensimmäisten testien koeversiot. Loput käyttötapaukset säästin itselleni, jotta sain harjoiteltua skriptausta alusta alkaen.

Skriptien laatimisessa käytettiin hyviä käytänteitä, joita kuvattiin tietoperustassa. Nimeämiset ja testien logiikat pyrittiin pitämään järkevinä käytön ja ylläpidon kannalta. ”Test setup” ja ”Test teardown” ominaisuuksia hyödynnettiin ja vältettiin tarpeettomien ”Sleep”-odotuskomentojen käyttöä. Kävimme laadittuja skriptejä läpi automaatioinsinöörin kanssa ja sain häneltä mallia ja apuja omaan skriptaukseeni. Omalla kotikoneella tehtyjen kokeilujen jälkeen yleisimmät avainsanat olivat pintapuolisesti tuttuja.

Automaatiotestien luonnissa järjestelmän valitsimet aiheuttivat hieman ylimääräistä työtä. Osasta elementeistä puuttui elementtien ID:t kokonaan, jolloin ID:tä ei voitu käyttää valitsimena kyseisillä

elementeillä. Osassa elementeistä ID:t olivat olemassa, mutta ne olivat dynaamisia. Tämä tarkoittaa, että ID muuttui jokaisella sivuston latauksella. Jos ID:eitä olisi käytetty näiden elementtien valitsimina, testi skriptauksen aikana avoimena olleella selain istunnolla, mutta se rikkoutuisi heti kun testi ajettaisiin ja elementin dynaaminen ID muuttuu, kun sivu avataan uudestaan. Tämä aiheutti sen, että verkkosivun elementteihin ei aina pystytty käyttämään ID:t valitsimena. Monesti elementteihin vaikuttamisessa jouduttiin käyttämään valitsimina CSS:ää tai XPathia ja näiden poimiminen vei hieman enemmän aikaa. Tässä käytin apuna selaimen Devtools-työkalua, jolla saadaan avattua verkkosivun ”konepelti” ja voidaan tarkastella kaikkia sivun elementtejä. Kokeilin myös Katalon Recorderia lineaarisen skriptauksen toteuttamiseen. Skriptien laatiminen oli vaivatonta, kun järjestelmää käyttämällä pystyi saamaan aikaan toimivan testin. Näin luodut testit olivat kuitenkin vain yhteen tarkoitukseen sopivia ja ne rikkoutuivat helposti. Esimerkiksi elementtien muuttunut järjestys käyttöliittymän hakutoiminnon taulukossa olisi vaatinut skriptin uudelleen tallennusta tai suurta muutostyötä. Näiden kokemuksien vuoksi jätin tämän tekniikan sivuun ja keskityin avainsana- ja datakeskeisiin skriptauksiin.

Muutamassa testissä kohdattiin ongelmia tilanteissa, joissa selain yritti toiminnon yhteydessä avata Microsoft Word -ohjelmaa työasemalta, eikä Windowsin ”Alert”-tyyppistä huomautusta saatu ohitettua selaimen käsittelyyn tarkoitetulla kirjastolla. Ongelma kierrettiin sulkemalla selain välissä ja avaamalla se uudestaan haluttuun URLiin. Robot Frameworkin RPA.Framework-kirjastokokoelmasta löytyy Windowsin ja työaseman automaatioon tarkoitettuja kirjastoja, joilla ongelman voisi ratkaista. Työasema-automaation tekniikat ovat yksi kohde jatkokehitykselle.

Kun olin ehtinyt saada toimintakuntoon kolme omaa testiä, saimme ilmoituksen, että projektiin palkattu insinööri vaihtuu. Jäin edistämään Lambda-järjestelmän testejä omin voimin marraskuussa 2023, kunnes uusi insinööri saatiin palkattua. Otin tässä vaiheessa käsittelyyni myös lähteneen automaatioinsinöörin laatimat kuusi ensimmäistä testiä. Muutin niitä yleispätevämmin toimiviksi ja poistin valitsimien ja arvojen kovakoodauksia ja siirsin niitä variables-muuttujatiedostoon datalähtöisen skriptauksen periaatteiden mukaisesti.

Tässä vaiheessa valmiina oli yhdeksän toimivaa automaatioskriptiä yhteen Lambdan testiympäristöön ja siirryin laatimaan suunniteltuja testejä tuotantoympäristöä varten. Tuotantoympäristöön testejä ei voitu ajaa samassa laajuudessa, joten loimme neljä erillistä suppeampaa testiä tuotannon savutestiksi. Testiympäristöön luotuja avainsanoja yhdistelemällä ja valmiiksi selvitettyjä valitsimia hyödyntämällä testit saatiin valmiiksi tehokkaasti ja pienellä työmäärällä.

Tuotantoympäristöön kaavailtuihin testeihin lisättiin ylimääräisiä tarkistuksia, joilla pyrittiin estämään mahdolliset tahattomat muutokset tuotannossa. Yleensä tuotantoympäristöön ei automaatiotestejä suositella ajettavan, mutta Lambda-järjestelmälle toivottiin heidän tuotannossa tekemänsä

savutestin automatisointia. Tällä testisarjalla voitaisiin nopeasti tarkistaa järjestelmän avaintoiminteita ja yleistä ympäristön tilaa erilaisten päivitysten tai katkojen jälkeen. Testit laadittiin ensin Lambdan testiympäristöön ja testattiin siellä toimiviksi. Jatkoa varten jäätiin odottamaan hyväksyntää tuotannon testaamiseen ja testit pystyttään siirtämään tuotantoympäristöön muuttamalla yhden muuttujan arvo.

3.4.3 Migraatio Browser-kirjastoon

Uuden automaatioinsinöörin haun yhteydessä kattoprojektissa tehtiin päätös siirtyä Selenium-kirjastosta Browser-kirjastoon. Tähän mennessä tehdyt testit muokattiin käyttämään uutta kirjastoa. Toimeksiantajan sisäverkon suojaus aiheutti hieman ongelmia Browser-kirjaston asennuksessa. Asennus saatiin vietyä loppuun IT-tuen avustuksella.

Lambdan testien migrointi onnistui hyvin. Testisarjoihin ei tarvinnut tehdä montaakaan muutosta. Tarpeettomia avainsanoja poistetiin ja muutamaa muutettiin hieman. Suurin osa muutoksista kohdistui omiin avainsanoihin ja muuttujiin common.robot ja variables.robot tiedostoissa. Selektorit pysyivät hyvin samanlaisina eri kirjastoilla: XPathin käyttö on samanlainen molemmissa, elementtien ID:tä käytettäessä täytyi ”id:sivustoElementti” muuttaa ”id=sivustoElementti”. Browserista löytyi suhteellisen helposti Seleniumia vastaavat avainsanat tai sellaiset avainsanat, joilla sama lopputulos saatiin aikaan.

Skripteistä pystyttiin poistamaan erillisiä odotusavainsanoja ”Sleep” ja ”Wait for text”, koska Browser-kirjaston avainsanoissa on oletuksena sisäänrakennetut odotuskomennot. Lisäksi erilaisissa tekstejä käsittelevissä avainsanoissa on valmiit toiminnot arvojen validoinnille, eikä erillisiä rivejä tarvittu ylimääräisille tarkistuksille. Kuvissa 11 ja 12 on esitetty kuvissa 4 ja 5 olleen Progress bar -testin ”Odota 75 %” avainsana Selenium- ja Browser-kirjastoilla toteutettuina. Avainsanassa näkyy yksi ”Sleep” avainsanan käyttö. Tätä odotuskomentoa ei hyvien toimintatapojen mukaan suositella käytettävän. Kyseisessä tilanteessa sitä käytetään kuitenkin vain testin suorituksen seuraamisen helpottamiseksi.

```
197 Odota 75%
198     Set Selenium Speed    0.0s
199     FOR    ${i}    IN RANGE    100000
200         ${progress}=    SeleniumLibrary.get element attribute    xpath=//div[@id='progressBar']    aria-valuenow
201         IF    ${progress} >= 75
202             Paina Nappia    id:stopButton
203             Sleep    2s
204             BREAK
205         END
206     END
```

Kuva 11. Selenium-kirjastoa käyttävä yhdistelty avainsana

```
Load in Interactive Console
Odota 75%
    FOR    ${i}    IN RANGE    100000
        ${progress}=    Get Attribute    ${edistysPalkki}    aria-valuenow
        IF    ${progress} >= 75
            Paina Nappia    ${pysaytysNappi}
            Sleep    2s
            BREAK
        END
    END
```

Kuva 12. Yhdistelty avainsana Browser kirjastolla

Muutaman testin avainsanoissa jouduttiin tekemään migraatiossa suurempi remontti, kun haettiin sopivaa avainsanaa. Samalla muokattiin testejä yleispätevimmiksi ja parannettiin niiden toimintaa. Positiivinen havainto oli, että Browser-kirjasto mahdollisti paikoitellen ongelman ratkaisun valmiilla avainsanoilla, kun Seleniumilla oli vaatinut samoihin toimintoihin erillisillä lisälogiikoita.

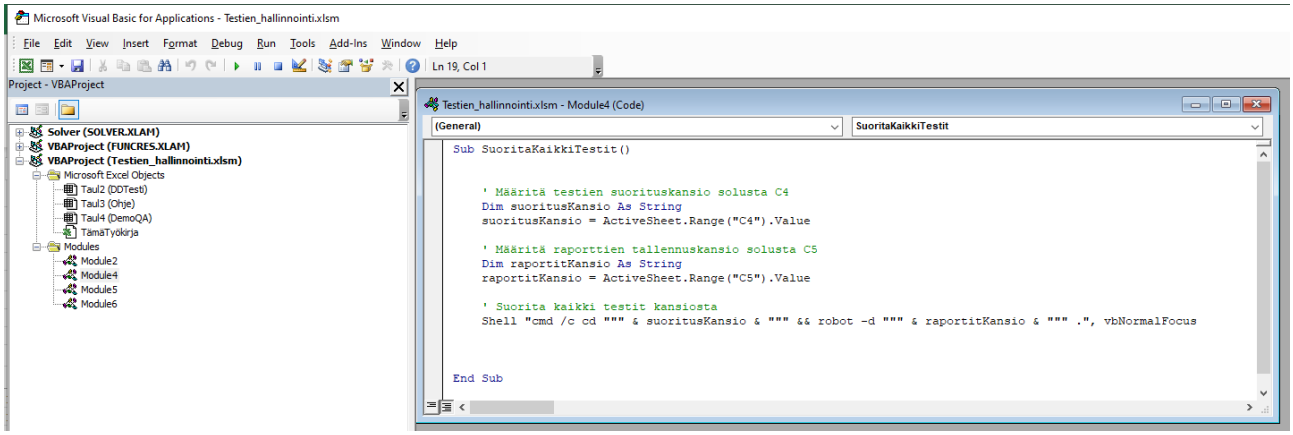
Samaan aikaan, kun Browser-migraatiota tehtiin, ryhdyttiin valmistelemaan DataDriver-kirjaston käyttöönottamista. Testien syntaksi muutettiin ”Data Driven”-tyyppiseksi ja laadimme hallinnointi-käyttöliittymän, johon testien tiedot vietiin. Lambdan kaikki laaditut testit muutettiin käyttämään Browser-kirjaston lisäksi DataDriver-kirjastoa ja ne vietiin osaksi luotua Excel-käyttöliittymää.

Uusi automaatioinsinööri saatiin projektiin vasta 2024 tammi-helmikuun vaihteessa. Lambdaan suunnitellut ensimmäisen vaiheen testit oli siihen mennessä saatu valmiiksi. Päätimme katsel-moida laaditut testit ja toteutuksen läpi uuden insinöörin kanssa. Sain rakentavaa palautetta, jonka perusteella muokkasin testejä. Tiivistin muutamaa testiä viemällä avainsanoja common.robot tie-doston puolelle ja korvaamalla ne korkean tason avainsanoilla, jotta testit olivat helpompia lukea. Samalla alitettiin ohjeellinen alle 10 avainsanaa testiä kohden raja. Yksi hakuja käsitellyt testita-paus eroteltiin kolmeksi erilliseksi testiksi. Vaikka alkuperäisessä testissä testattiin yhtä asiaa, eli hakutoimintoa, hakuja tehtiin kolmen eri arvon rajauksella ja käytännössä testattiin samaa asiaa kolmella eri tavalla.

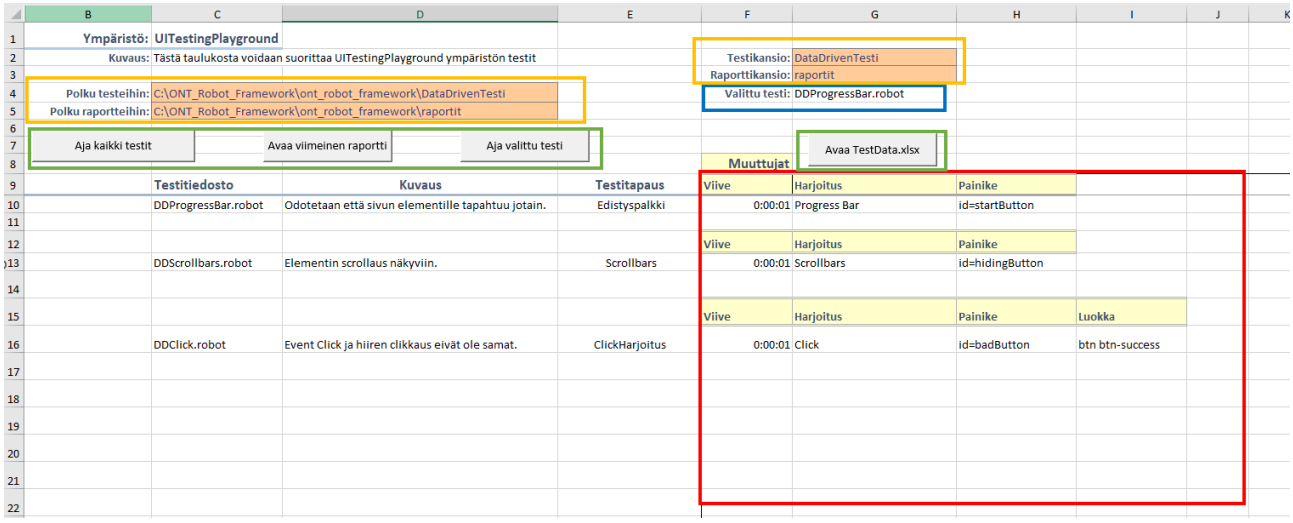
3.4.4 Excel-käyttöliittymä

Automaatiotestien vaivattomaan suorittamiseen suunniteltiin kokonaisuus, joka sisälsi kaksi Excel-tiedostoa ja DataDriver-kirjaston. Ajatuksena oli laatia helppokäyttöinen tapa ajaa testejä ja suorit-taa niitä eri parametreillä ilman Robot Frameworkin tuntemusta tai .robot-tiedostojen muokkaa-mista. Automaatioskriptit muutettiin käyttämään lähes kokonaan muuttujia arvojen syötössä tai verkkosivun elementteihin vaikutettaessa. Aluksi laadittiin prototyypin omaisesti Testien_hallin-nointi.xlsm tiedosto, johon tuotiin jokaiselle testattavalle Lambdan ympäristölle oma

laskentakaavio. Näille lisättiin testien tiedot sekä muuttujat, joita käyttäjien annettiin muokata. Excelin valmiita funktioita ja kehitystyökalujen hyödyntäen lisättiin radiopainikkeita ja soluja, joiden avulla pystyttiin valitsemaan yksittäinen testi ajettavaksi. Painikkeiden ja VBA-skriptien avulla lisättiin seuraavat toiminnallisuudet: kaikkien avoimen laskentataulukon testien suorittaminen, valitun testin suorittaminen ja viimeisen ajetun testin raportin avaaminen kansiosta. VBA-funktioiden laatimisessa hyödynnettiin ChatGPT 3.5 -kielimallin kyvykkyyksiä ja verkosta löytyneitä opetusmateriaaleja. Ennen VBA-skriptien käyttöä niiden asianmukaisuus tarkistettiin VBA:ta aiemmin käyttäneeltä kollegalta ja testasin niitä ensin omalla henkilökohtaisella työasemallani. Kuvassa 13 on esitetty Excelin VBA-ohjelmointi näkymässä yksi moduuli, jonka avulla voidaan suorittaa kaikki näkymän testit.



Kuva 13. Kaikkien testien suorittamisen käynnistävä VBA-koodi



Kuva 14. Hallinnointi Excelin esimerkkinäkymä

Exceliin automatisoitiin funktioilla tiedoston polun valitseminen. Näin saatiin aikaan toteutus, jonka pitäisi toimia riippumatta siitä, mihin tiedostot puretaan käyttäjän koneella. Käyttäjälle jätettiin

mahdollisuus muuttaa tiedostopolkuja, jos halutaan esimerkiksi tallentaa eri Lambdan ympäristöjen testien raportit omiin kansioihinsa. Kuvassa 14 esiteltiin yhtä käyttöliittymän näkymistä. Kuvaan on korostettu eri väreillä näkymän osiot, joita muokkaamalla käyttäjä voi vaikuttaa testeihin tai niiden suorittamiseen.

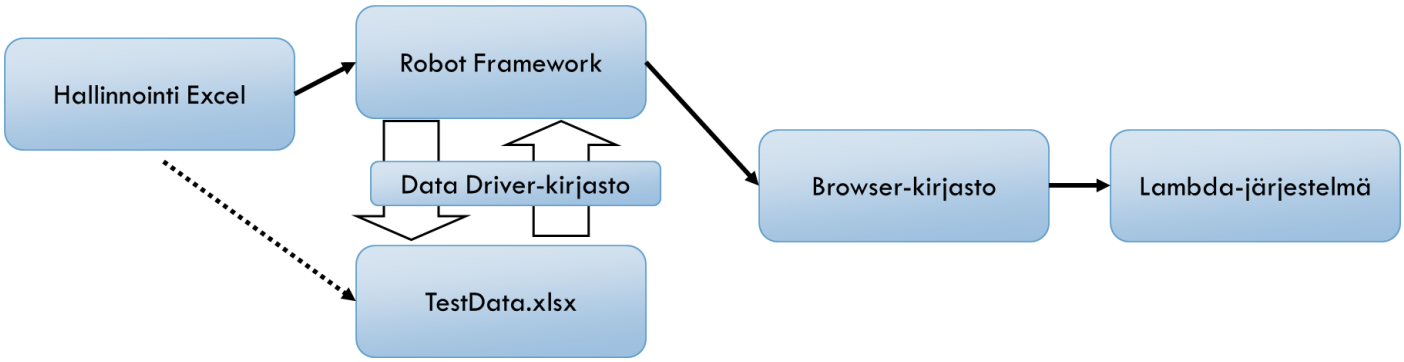
Kokonaisuuden toinen Excel-tiedosto oli TestData.xlsx tiedosto. Jokaiselle testille määritettiin oma laskentataulukko tähän tiedostoon. Testejä suoritettaessa DataDriver-kirjasto poimii skriptien käyttämien muuttujien arvot määrätystä välilehdestä. Testien_hallinnointi.xlsm-tiedoston muuttujien arvot kopioitiin linkkeinä TestData.xlsx tiedoston laskentataulukoille. Näin saatiin kokonaisuus, jossa muuttujien arvot voitiin vaihtaa hallinnointi Excelissä ja ainoastaan avata ja tallentaa TestData-tiedosto, jotta arvot saadaan päivitettyä. Kuvassa 15 on esitelty TestData-tiedoston yhden testin tiedot.

	A	B	C	D	E	F	G
1	*** Test Cases ***	${viive}	${harjoitus}	${painike}			
2	Scrollbars	0:00:01	Scrollbars	id=hidingButton			
3							
4							

DemoQA.robot | DDProgressBar.robot | **DDScrollbars.robot** | DDClick.robot

Kuva 15. TestData-tiedoston esimerkkinäkymä

Kokonaisuuden toimintaperiaate on kuvattu kuvassa 16. Hallinnointi Excel välittää testien muuttujien arvot TestData.xlsx tiedostoon. Hallinnointi Excelillä käynnistetään automaatiotestien suorittaminen VBA-skriptin avulla. Robot Framework poimii suoritettavan testin tiedot skriptiin määritellystä välilehdestä TestData.xlsx tiedostosta. Testi suoritettaan Robot Frameworkin muokkaaman testidatan avulla hyödyntäen Browser-kirjastoa ja sen sisällä olevia WebDrivereita, eli komponentteja, joilla voidaan vaikuttaa selaimeen.



Kuva 16. Testien_hallinnointi.xlsm käyttöliittymän toimintaperiaate

Kokeilimme asentaa työkalut ja ajaa testejä yhden Lambda-tiimin jäsenen työasemalla, kun saimme toimivan prototyypin valmiiksi. Kokeilussa huomattiin vielä puutteita testien ajettavuudessa ja yleiskäytettävyydessä, johtuen käyttäjien mahdollisuudesta tehdä roolinvaihtoja ja muokata järjestelmän omaa oletusnäkymää.

Prototyypin ja kokeilujen perusteella käyttöliittymää muutettiin helpommaksi ylläpitää poistamalla radiopainikkeet ja muuttamalla yksittäisen testin valinnan alasvetovalikoksi. Muuttujien arvojen vaihtamisen helpottamiseksi lisättiin VBA-skripti, jolla voitiin avata TestData-tiedosto hallinnointitiedostosta käsin, päivittää TestData-tiedosto ja sulkea se. Tiedostoja ei saatu päivittämään tietoja automaattisesti ilman tiedostojen avaamista, mutta tämä oli kuitenkin tässä tilanteessa käyttäjälle helpoin tapa päivittää muutokset. Automaatioskripteihin tehtiin parannuksia, jotta ne toimivat varmemmin erilaisilla käyttäjillä.

Työn loppupuolella havahduin ongelmaan käyttöliittymän suunnittelussa. Toteutuksesta puuttui helppo tapa päivittää käyttöliittymään uusia testitapauksia ilman, että hävitetään mahdolliset käyttäjän tekemät muutokset muuttujiin. Ideoin tähän erilaisia VBA-skriptejä, joille määritettäisiin viimeinen rivi, jolla testejä on edellisessä versiossa ja Testien_hallinnointi.xlsm-tiedostoon haettaisiin päivitetystä tiedostosta uudet testit määritetystä rivistä eteenpäin. Tätä ei muutamalla yrityksellä saatu toimimaan ja jatkoa varten testit suunniteltiin lähtökohtaisesti niin, että testit toimivat ilman muuttujien arvojen muuttamista. Ongelma kierrettiin laatimalla testit mahdollisimman yleispäteviksi, jotta käyttäjien ei alkuvaiheessa tarvitse vaihtaa muuttujien arvoja. Lisäksi asennustiedostoihin ja ohjeistukseen lisättiin erillinen ”Setup”-testi, jolla ajettiin yhteiset muutokset testiympäristöön. Näin saatiin varmistettua, että kaikki testit varmasti toimivat eri käyttäjillä. Setup-testi muodostettiin suurelta osin aiemmin yhdistellyistä omista avainsanoista. Testien_hallinnointi- ja TestData -tiedostot korvataan toistaiseksi suoraan uusilla tiedostoilla testejä lisättäessä.

Toteutuksen jatkokehitykseksi ideoitiin mahdollisuus luoda kokonaan oma käyttöliittymä, joka käyttäisi toista kirjastoa ja tietokantaa muuttujien säilyttämiseen.

4 Tuotoksien esittely

Lambda-järjestelmään laadittiin suunnitellut ensimmäisen vaiheen automaatiotestit. Testejä toteutettiin yhteensä 15. Kuuden testin ensimmäiset versiot olivat alkuperäisen automaatioinsinöörin laatimia. Minä laadin loput yhdeksän ja muokkasin edellä mainitut kuusi testiä käyttämään lopullisia kirjastoja, jotta ne vastasivat muita testejä. Omia yhdisteltyjä avainsanoja muodostettiin 110. Näistä alkuperäinen automaatioinsinööri laati noin 50.

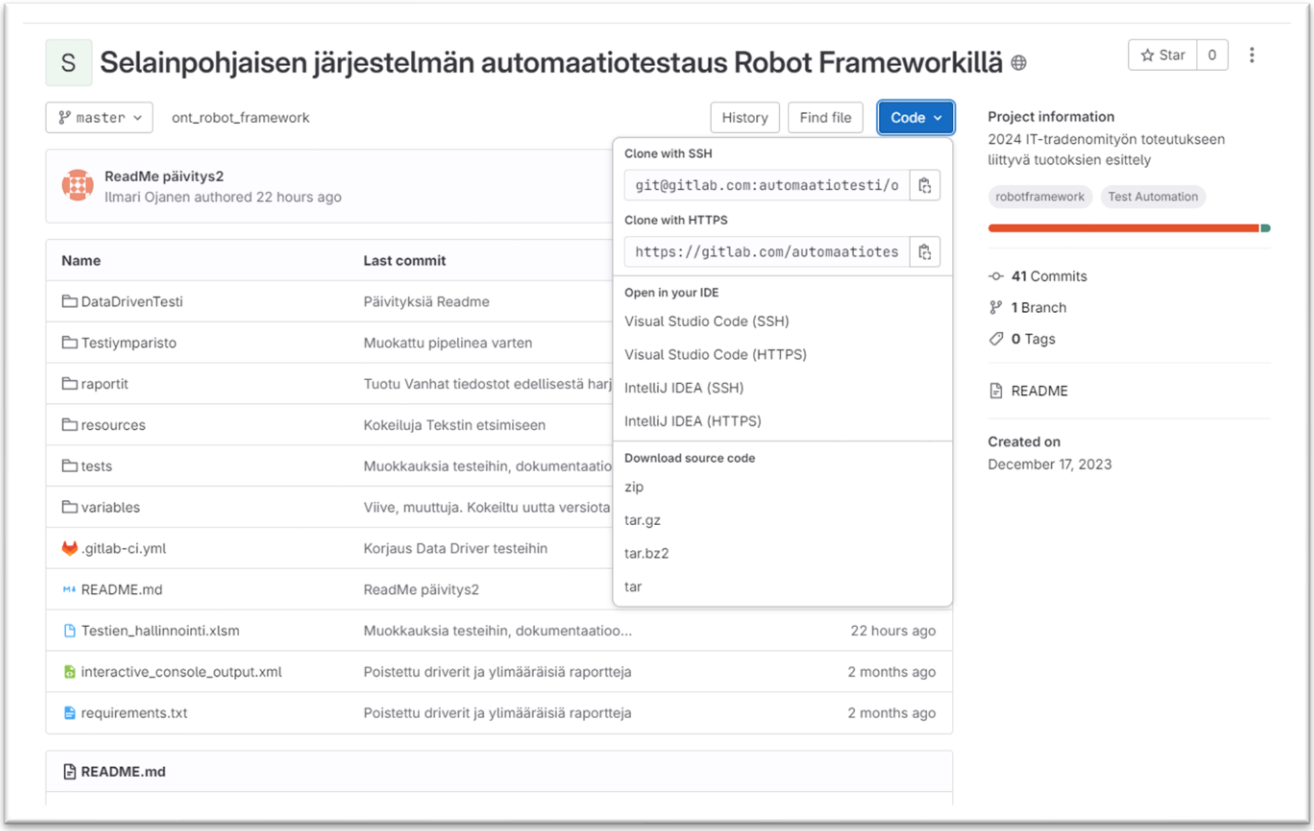
Tuotantoympäristön testejä ei vielä virallisesti lähdetty käyttämään, mutta testit ovat ajettavissa yhden muuttujan arvoa vaihtamalla. Ylläpitotiimille luotiin Excel-käyttöliittymä testaamisen avuksi. Testien suorittaminen onnistuu tämän avulla yhden napin painalluksella. Tiedostot ja materiaalit tallennettiin niitä varten perustettuun Gitlab-repositorioon. Materiaalien ja testien ylläpidosta laadittiin suunnitelma ja lisäksi tehtiin alustava ehdotus seuraavista automaatioon lisättävistä ominaisuuksista. Testien käyttöönotosta ja uusien testien laatimisesta sekä lisäämisestä Excel-käyttöliittymään laadittiin käyttöohjeet.

Lambdan ylläpitotiimi asensi kaikille onnistuneesti ohjeiden mukaan vaaditut ohjelmat ja testien suoritus onnistui kaikkien työasemilla. Automaatiotestit otettiin mukaan Lambdan uusien versioiden testaukseen. Lisäksi testejä ryhdyttiin ajamaan säännöllisesti osana jatkuvaa testausta. Excel-käyttöliittymä koettiin helppokäyttöiseksi ja tarpeet täyttäväksi.

Lambda-järjestelmää, siihen liittyviä materiaaleja tai ympäristöjä ei voida tässä opinnäytetyössä esitellä salassapitosopimusten vuoksi. Työn toteutuksen aikana harjoittelin tekniikoita ja tuotosten toteuttamista omalla tietokoneella julkisiin sivustoihin ja omassa Gitlabissa. Nämä esimerkkituotokset löytyvät julkisesta Gitlab-projektista, joka on nimeltään ”Automaatiotesti / Selainpohjaisen järjestelmän automaatiotestaus Robot Frameworkillä”. Projekti löytyy osoitteesta https://gitlab.com/automaatiotesti/ont_robot_framework.

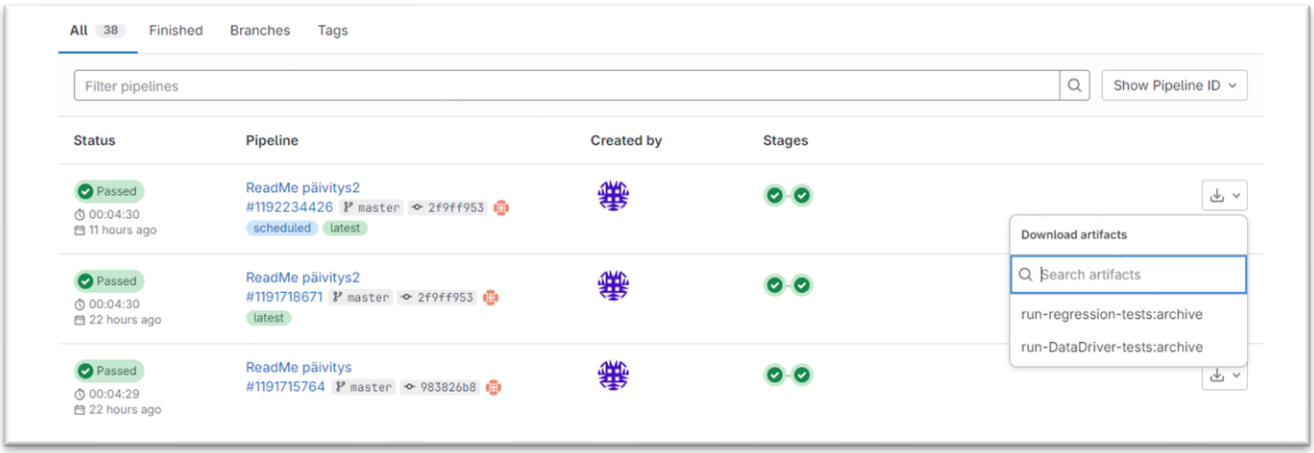
4.1.1 Gitlab-harjoitusympäristö

Luodussa esimerkkiympäristössä työ keskittyi kahteen osaan: Code-Repository-osioon, joka on käytännössä projektin etusivu, sekä Build-osioon. Etusivulla näkyvät kaikki projektin sisältämät tiedostot ja ReadMe-tiedosto, josta löytyy ohjeet testien käyttöönotosta. Tiedostot saa tästä näkymästä ladattua sinisen ”Code”-valinnan alta. Kuvassa 17 on esitelty repositorion etusivu ja latauslinkit.



Kuva 17. Gitlabin etusivu tiedostojen lataus valikko avattuna

Buildissa voi tarkastella toimivaa toteutusta CI/CD-putkesta. Testit on ”Pipeline schedules” osiossa määritetty ajettavaksi joka tiistai kello 12. Suoritettujen testien raportit voi ladata oikean reunan ”artifacts”-painikkeen alta. Tämä näkymä on esitetty kuvassa 18.



Kuva 18. Pipelines näkymä ja testiraporttien latauspainikkeet

CI/CD-putken määritykset, ajettavat testit ja raporttien säilyttäminen asetetaan .gitlab-ci.yml-tiedostossa, joka sijaitsee repository-osiossa. Määritystiedosto on esitelty kuvassa 19. Tässä voidaan määritellä Robot Framework skriptien suorituksen '—options'-muuttujat, joilla voidaan vaikuttaa testien ajoon. Tässä esimerkissä on vaihdettu testien ajoon muuttuja "headless" arvoon "true", jolloin testit ajetaan ilman näkyvää selainta ja viiveellä 0:00:00, eli testit suoritettaan ilman ylimääräistä viivettä. Gitlabin kautta ajettaessa ei tarvita näkyvää selainta eikä ylimääräistä viivettä helpottamaan ajon seurantaa. Testien ajoon on määritetty myös rajaus, jolla ohitetaan testejä. En saanut kahta esimerkkitesteistä toimimaan ja '--exlude virhe' optiolla jätettiin testien ajosta pois testit, jotka on merkitty tagilla "virhe".

Kuva 19. CI/CD-putken määritystiedosto

4.1.2 Robot Framework -testit

Gitlab-projektista löytyvien testien tarkoituksena oli mahdollistaa skriptauksen ja työkalujen käytön harjoittelu. Harjoitusympäristössä pystyin tutustumaan valmiiden kirjastojen avainsanoihin ja demonstroimaan niiden käyttöä. Testiskriptejä päivitettiin useampaan otteeseen työn edetessä ja tiedostojen historiasta voi tarkastella testien aiempia versioita ensin Selenium-kirjastoa, sitten Browser-kirjastoa ja lopulta DataDriver-kirjastoa käyttävinä versioina.

Testit ja niihin liittyvät tiedostot löytyvät neljästä kansiosta. Test-kansiossa on alkuperäiset UITestinPlayground.org sivustolle tehdyt automaatiotestit tiedostossa Playground.robot.

DataDrivenTesti-kansiossa on kolme Playground.robot testisarjasta poimittua testitapausta, jotka on muutettu käyttämään DataDriver-kirjastoa. Nämä ovat ajettavissa Testien_hallinnointi-Excelillä. Testiympäristö-kansiosta löytyvässä DemoQA.robot-testissä harjoiteltiin testausta toiseen sivustoon. Samalla kokeiltiin usean eri testitapauksen lisäämistä eri arvoilla samaan testisarjaan Data Driven development -mallilla. Resources-kansiosta löytyy common.robot-tiedosto, johon vietiin omat yhdistellyt avainsanat. Yleiset muuttujat, kuten testiympäristöjen URL:it ja monimutkaisemmat selektorit, lisättiin variables-kansion variables.robot-tiedostoon. Testitapauksia kertyi yhteensä 22 kappaletta. Playground.robot-testisarjan testeistä Dynamic Table ja Shadow DOM -testitapauksia ei saatu toimimaan. Oma koodaustaitoni ei riittänyt taulukoiden oikeaan käsittelyyn ja epäilen että Shadow DOM -testi olisi vaatinut työaseman automaatiota leikepöydän käsittelyn osalta.

Harjoitteluksi testatut UITestingPlayground ja DemoQA-sivustot ovat automaatioon ja testaukseen tarkoitettuja esimerkkisivuja. Erityisesti Playgroundin tehtävät oli laadittu keskittyen yleisimpiin ongelmiin, joihin testaaja voi selainautomaatiossa törmätä. Testeissä joutui käsittelemään erilaisia painikkeiden aktivoitumisia, elementtien muuttumisia kesken toimintojen, piilotettuja elementtejä sekä rakenteita, jotka estivät elementteihin vaikuttamisen suoraan. Monessa harjoitustestissä tuntuikin, että näiden kikkojen automatisointi oli vaikeampaa kuin oikean kohdejärjestelmän käsittely. Sivuston esimerkit on otettu oikeista järjestelmistä ja mielestäni aloittelevan testaajan kannattaa tutustua tällaisiin esimerkkisivustoihin ja oppia käytännön kautta erilaiset sudenkuopat.

Kokeilin omasta mielenkiinnostani käyttää tekoälyä harjoittelusivustojen muutamien skriptien laadinnassa ja virheiden etsimisessä. ChatGPT:ltä sai hyvin tekniikoiden perusteita ja se kuvasi mahdollisia virheen syitä kattavasti. Pariin kertaan tekoäly kuitenkin kompastui yksittäisten avainsanojen käyttöön. Luultavasti näissä oli ongelma kirjastojen viittausten kanssa, koska sama avainsana löytyi useammasta eri kirjastosta. Amerikkalainen Robocorp-yritys on lanseerannut sivuilleen oman generatiivisen tekoälychatbotin ReMarkin, joka painottaa Robot Frameworkiin liittyvää dokumentaatiota. Tällaiset työkalut ovat aloittelevalle testaajalle hyödyllisiä, jos käytettävissä ei ole toista silmäparia näkemässä virheitä, joille itse on sokea. Avoimien tekoälytyökalujen käytössä kannattaa kuitenkin muistaa aina tietosuoja ja miettiä, mitä materiaalia tekoälylle syötetään. Toimeksiantajan Lambda-järjestelmään liittyvää materiaalia tai tapauksia ei tekoälyn avulla yritetty ratkaista.

4.1.3 Testien hallinnointikäyttöliittymä

Testien_hallinnointi.xlsm -tiedosto on toimiva prototyyppi laaditusta Excel-käyttöliittymästä. Kaikkia harjoitusprojektin testejä ei tuotu tähän, vaan Excelin laadintaa ja toimintaa kokeiltiin neljällä eri testillä. Tiedostosta löytyy ”Ohje”-välilehti, jossa on kuvaus tiedoston toiminnasta ja testien käyttöönoton ohjeistus. Sama käyttöönotonohje on ReadMe-tiedostossa. Kokonaisuus on laadittu niin,

että tiedostot voi ladata ja purkaa koneelle mihin tahansa kansioon. Testien pitäisi toimia, kun asennustoimet on tehty ja makrot on sallittu Testien_hallinnointi.xlsm -työkirjalle.

5 Pohdinta

5.1 Työn arviointi

Projektin ensimmäiset palaverit olivat syksyllä 2023 ja sovitut osat saatiin valmiiksi helmikuussa 2024. Automaatioprojektiksi työstöaika oli verrattain pitkä suhteessa laadittujen testien määrään. Tulokset olisi saatu valmiiksi huomattavasti reippaammassa aikataulussa, jos työlle olisi ollut mahdollista antaa enemmän aikaa. Tuotokset ovat kuitenkin tarpeeseen nähden onnistuneet.

Päätavoite oli käynnistää Lambda-järjestelmän automaatiotestaus. Tässä onnistuttiin ja suunnitellut ensimmäisen vaiheen automaatiotestit saatiin laadittua. Testien toteutuksessa pyrittiin ottamaan huomioon niiden ylläpidettävyys noudattamalla hyviä skriptauksen periaatteita. Testeistä laadittiin uudelleenkäytettäviä käyttämällä määriteltäviä muuttujia, jotka vietiin kootusti omaan erilliseen tiedostoon tai Excel-käyttöliittymään. Testien, avainsanojen ja muuttujien nimeäminen pyrittiin pitämään yhdenmukaisena. Kunkin testin korkean tason avainsanojen määrä pidettiin alle kymmenessä, jotta testit olivat helpommin luettavia ja ymmärrettäviä. Laaditut testit otettiin mukaan järjestelmän jatkuvaan testaukseen ja testejä tullaan jatkokehittämään. Automaation suorittamilla testeillä saadaan helpotettua pienen tiimin työmäärää. Lambdan uudessa versiossa tai ympäristöissä ilmenevät ongelmat saadaan tutkittavaksi nopeammin, kun testit voidaan suorittaa automaatiolla nopeammin kuin manuaalisesti. Testien vaatimat ohjelmat ja komponentit saatiin asennettua koko Lambda-tiimille laaditun dokumentaation ja ohjeistuksen avulla.

Tavoite toimittaa ylläpitotiimille erillinen testien suorittamista helpottava käyttöliittymä toteutettiin ja tiimi oli sen toteutukseen tyytyväinen. Käyttöliittymä oli selkeä ja toiminnaltaan yksinkertainen. Toteutettu työkalu otettiin käyttöön Lambdan-ylläpitotiimille. Toteutus ei kuitenkaan ollut täydellinen. Testien datan päivittymistä ei saatu täysin automaattiseksi. Käyttäjien täytyy avata erillinen datan tallennustiedosto muuttujien päivittämiseksi. Lisäksi uusien testien lisääminen Excel-käyttöliittymään ilman käyttäjien omien muuttujien arvojen menettämistä osoittautui haasteelliseksi. Erillinen käyttöliittymä luo ylimääräisen ylläpidettävän rakenteen, mutta Excelin lisätyötä ei kuitenkaan koettu mahdottomaksi, koska Excel-pohja ja prosessi oli valmiiksi laadittu. Käyttöliittymä mahdollistaa testien käytön isommalle käyttäjämäärälle ilman korkeaa teknistä osaamista ja konseptia pystyisi tarvittaessa laajentamaan organisaation muihinkin järjestelmiin. Tällöin Excelin tilalle kannattaisi kuitenkin suunnitella graafinen käyttöliittymä. Excelin uudelleentyöstö mainittiin yhtenä jatkokehitysmahdollisuutena.

Gitlab-repositorion osalta tavoite saavutettiin lukuun ottamatta CI/CD-putken käyttöönottoa, joka jäi jatkokehitykselle. Projektin aineiston ja testien säilytystä ja ylläpitoa varten perustettu Gitlab-ympäristö on tarkoitukseen nähden toimiva ja se otettiin viralliseen käyttöön.

Kaiken kaikkiaan työtä voidaan pitää onnistuneena. Toimeksiantaja oli tyytyväinen toimitettuihin tuotoksiin ja ne otettiin kokonaisuudessaan käyttöön.

5.2 Johtopäätökset

Automaatiotestauksen käynnistäminen on suuri projekti ja se vaatii yritykseltä tahtoa ja resursseja. Automaatiotestauksen aloittaminen on järkevää projektoida ja suunnitella huolellisesti. Suunnittelussa kannattaa kiinnittää huomiota valittaviin tekniikoihin ja työkaluihin. Valinnat tulee tehdä testattavan kohteen mukaan ja tekijöiden tietotaidon perusteella, unohtamatta työkalujen kustannuksia. Virhe suunnittelussa aiheuttaa lisätöitä ja kustannuksia projektille, kun joudutaan tekemään korjausliikkeitä, kuten tässä projektissa kävi Selenium-Browser vaihdon kanssa.

Työssä käytetty Robot Framework ja sen kirjastot vaikuttivat käyttäjäystävällisiltä ja suhteellisen helppokäyttöisiltä vasta-alkajallekin. Ne sopivat sekä testauksen kohteeseen että -tavoitteeseen ja voikin sanoa, että työkalujen valinta oli onnistunut. Pieni kokemukseni ohjelmoinnista varmasti auttoi työkalun käyttöönotossa, sillä automaatiotestaus on itsessään ohjelmistokehitystä.

Koodin ylläpidon kustannukset voivat olla merkittävät. Artikkelissaan *Writing Maintainable Automated Acceptance Tests* Dale Emery (2009, 1) esittää, että ylläpidon kustannukset voivat olla jopa syy koko automaatiotestauksen lopettamiseen. Toteutuksen ylläpidettävyyteen ja testien uudelleenkäytettävyyteen kannattaakin kiinnittää huomiota jo suunnittelussa, koska niillä saadaan suurempaa arvoa tehdylle työlle. Automaation rakentamisessa kannattaa noudattaa hyviä skriptauksen käytäntöjä ja yhdenmukaisia menetelmiä. Näin saadaan helpommin ylläpidettävää koodia ja vähennetään ylläpitokustannuksia.

Kun testaus saadaan käynnistettyä ja luotua valmista testidataa pohjalle, on avainsanapohjaisen testauskehyksen jatkokehittäminen helpompaa. Projektin edetessä uusien testien rakentaminen kävi helpommaksi, kun pystyin hyödyntämään aiempia komponentteja seuraavien testien luomisessa.

5.3 Jatkokehittäminen

Työssä laadittuja testejä ja omia yhdisteltyä avainsanoja yhdistelemällä voidaan laatia lisää testejä. Lisäksi voidaan tehdä lisää omia avainsanoja uusille toiminteille. Myös aiemmissa testeissä valmiiksi poimitut selainelementtien valitsimet nopeuttavat testien jatkokehittämistä ja testauksen laajentamista. Testien osalta laadittiin jatkosuunnitelma siitä, miten testejä kehitetään ja mitä ominaisuuksia testaukseen otetaan jatkossa mukaan. Lisäksi lähdettiin tarkistamaan vanhoja Lambdan hyväksymistestejä ja poimimaan sieltä mahdollisia automatisoitavia testejä.

Työssä saatiin toteutettua toivottu Excel-käyttöliittymä testien ajamista varten, mutta tämän kehitystä voisi jatkaa parantamalla sen toimivuutta sekä ylläpidettävyyttä. Excelin ajatuksena oli, että eri käyttäjät voisivat helposti suorittaa testejä, tarvittaessa muuttaa muuttujien arvoja omiin käyttäjätileihinsä sopiviksi ja tehdä testejä erilaisilla arvoilla. Eri muuttujien arvot eri käyttäjillä aiheuttavat ongelman tiedostojen päivittämisessä. Taitoni eivät myöskään riittäneet testien lisäämiseen siten, ettei menetetty käyttäjien omia muuttujien arvoja. Exceliä pitäisi pystyä päivittämään muulla tavalla kuin luomalla uusi versio tai vaatimalla käyttäjiä kopioimaan tietoja tiedostosta toiseen. Ongelma kierrettiin työn aikana siten, että testit laadittiin niin yleispäteviksi, että käyttäjien ei välttämättä tarvitse muuttaa testien arvoja. Lisäksi laadittiin ylimääräinen ”Setup”-testi, jolla kaikkien käyttäjien tunnuksille luotiin samanlaiset tilanteet testejä varten.

Käyttöliittymän jatkokehityksenä voitaisiin Excelin tilalle pyrkiä kehittämään kokonaan oma graafinen käyttöliittymä. Testien tiedot ja muuttujat voitaisiin tallentaa omaan tietokantaan käyttämällä esimerkiksi SQL, MySQL, Microsoft Access tai muuta vastaavaa tietokantaa. Tällöin testien tietoja ja uusia testejä voitaisiin helpommin päivittää tietokantakyselyillä ilman, että käyttäjien omia arvoja muutetaan. TestData-tiedosto voitaisiin tässä mallissa hävittää ja korvata tietokannalla ja DataDriver-kirjasto voitaisiin korvata Database-kirjastolla.

Viime vuosina harppauksia ottanut tekoäly on ollut kiinnostuksen kohteena monella alalla. Myös testauksen piirissä on herätelty ajatuksia voisiko tekoälyn avulla automaatiotestausta siirtää tarkistusluonteisesta suorittavasta testauksesta bugien saalistustehtäviin. Toinen sovellutus tekoälylle voisi olla automaation kustannusten pienentäminen esimerkiksi käyttöliittymän muutostilanteissa, jolloin tarvitaan paljon resursseja skriptien laatimiseen ja testien ylläpitämiseen.

5.4 Opinnäyteprosessi

Prosessina työ oli ”kädet saveen ja syvään päähän” -tyyppinen. Työssä päästiin heti oikeaan tekemiseen kiinni. Toimeksiantajalle tehty toiminnallinen opinnäytetyö oli motivoiva suoritustapa, koska siinä päästiin toteuttamaan oikeaa työelämän tarpeeseen tulevaa konkreettista projektia.

Olin aiemmissa opinnoissani suorittanut yhden kandidaatin tasoisen lopputyön, joten tunsin, etten suuresti kaivannut ohjausta. Työ toteutettiinkin vain muutamalla alkuvaiheen yleisluonteisella ohjauskerralla. Koin kuitenkin, että seminaarissa saamani vertaisarviointi ja opinnäytetyön ohjaajan palaute olivat arvokkaita. Sain paljon hyviä korjausehdotuksia, joita otin huomioon työn viimeistä versiota laatiessani. Ajallisesti toteutus oli hivenen pitkä ja asetetuista aikataulu tavoitteista hiukan joustettiin, mutta lopputulos on mielestäni kaiken kaikkiaan onnistunut.

5.5 Oma oppiminen ja kehittyminen

Tietoni automaatiotestauksesta ja sen työkaluista olivat rajalliset johtuen suppeasta kokemuksesta työn alkaessa. Aiemmissa opintojaksoissa hankitut ohjelmoinnin perustaidot antoivat kuitenkin eväitä ohjelmointiin, ja Robot Frameworkin käyttö tuntui suhteellisen luontevalta. Ainoa asia, mitä jäin kaipaamaan, oli tarkempi kuvaus eri kirjastojen valmiiden avainsanojen käytöstä. Sekä Selenium- että Browser-kirjastojen avainsanojen dokumentaatioissa oli usein pari yksinkertaista esimerkkiä avainsanan käytöstä. Monessa tapauksessa nämä eivät antaneet mielestäni riittävän tarkkoja tietoja avainsanan toiminnasta ja oikeasta käytöstä, ja avainsanojen käyttöä täytyi muutaman kerran harjoitella erillisessä skriptissä ennen niiden käyttöä oikeassa ympäristössä. Robot Frameworkin perusteet ja käyttäminen konkretisoituivat työn aikana. Usean eri kirjaston käyttäminen antoi uutta näkemystä siitä, mitä on mahdollista tehdä erilaisilla valmiilla työkaluilla.

Työstä sain perustiedot ja -taidot oleellisen testauksen työkalun käyttöön. Oman jatkokehittymisen kannalta tunnistin muutaman laajempaa kokonaisuuden, joihin aion lähitulevaisuudessa panostaa. Robot Framework -osaaminen on keskiössä. Työkalut olivat sen verran helppokäyttöisiä, että en pysähtynyt tutkimaan pintaa syvemmältä, miten esimerkiksi Robot Frameworkin rajapinnat ja pohja logiikka toimii. Robocorpin tarjoamat Pythonin perusteita käsittelevät sertifiointikurssit ovat seuraavana listallani Robot Frameworkin ja automaation osaamisen kehittämiseksi.

Python osaamisen syventäminen ja omien kirjastojen laadinta avaavat lisää mahdollisuuksia Robot Frameworkin käyttämiseen. Aion myös tutustua laajemmin valmiisiin kirjastoihin. Robocorpin avoimen lähdekoodin RPA-Framework-kirjastokokoelma sisältää kirjastoja, joilla voidaan automatisoida työaseman toiminteita ja sovelluksia. Tekoälyn mahdollisuuksia automaatioskriptien luonnissa ja niiden ylläpidossa kannattaa myös selvittää.

Gitlabin tai muun vastaavan CI/CD-työkalun paremmalla tuntemuksella automaatiotestauksesta saadaan enemmän hyötyä, kun testien suorittamistakin voi automatisoida laajemmin. Lisäksi aion perehtyä yleiseen websovelluskehitykseen yleinen websovellus kehitys. Olen aiemmissa opinnoissani tutustunut verkkokehittämiseen vain kahden yksinkertaisen HTML-CSS-Javascript-React-tyyppisen verkkosivun verran. Verkkokehityksen sovellusten kehityksen perusteiden hallitseminen voisi antaa osaamista myös selainpohjaisten järjestelmien testaukseen.

Lähteet

Bajpai N. 2012. *A Keyword Driven Framework for Testing Web Applications*. International Journal of Advanced Computer Science and Applications, vol 3 no 3, s. 8–14.

Cambridge University Press & Assessment s.a. *Syntax* Luettavissa: https://dictionary.cambridge.org/dictionary/english/syntax. Luettu: 8.4.2024.

Emery D. H. 2009 *Writing Maintainable Automated Acceptance Tests*. Luettavissa: https://dhemery.com/pdf/writing_maintainable_automated_acceptance_tests.pdf. Luettu: 20.11.2023.

Farooq M.S., Omer U., Ramzan A., Rasheed M.A., Atal Z. 2023. *Behavior Driven Development: A Systematic Literature Review* in IEEE Access, vol. 11, s. 88008–88024.

Gitlab s.a.[a] *CI/CD pipelines.* Luettavissa: https://docs.gitlab.com/ee/ci/pipelines/ Luettu: 18.1.2024.

Gitlab s.a.[b]. *Get started with GitLab CI/CD.* Luettavissa: *https://docs.gitlab.com/ee/ci/* Luettu: 8.4.2024.

Islam M., Kahn S., Alam S., Hasan M. 2023. *Artificial Intelligence in Software Testing: A Systematic Review.* TENCON 2023 - 2023 IEEE Region 10 Conference (TENCON), s. 524–529.

Klärck P. 27.1.2016, *How to write good test cases using Robot Framework.* Päivitetty 5.4.2019. Luettavissa: https://github.com/robotframework/HowToWriteGoodTestCases/blob/master/HowToWriteGoodTestCases.rst. Luettu: 12.11.2023.

Klärck P. 1.9.2014. *Robot Framework Dos And Don'ts.* Luettavissa: https://www.slideshare.net/pekkaklarck/robot-framework-dos-and-donts. Luettu: 12.11.2023.

Microsoft 13.9.2021. *Office VBA Reference.* Luettavissa: https://learn.microsoft.com/en-us/office/vba/api/overview/. Luettu: 8.4.2024.

Milad H., Nahla E-H., Mostafa, S.A. 2014. *Review of Scripting Techniques Used in Automated Software Testing*. International Journal of Advanced Computer Science and Applications, vol 5, no 1, s. 194–202.

Mozilla s.a. *Attribute selectors.* Luettavissa: https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors/Attribute_selectors. Luettu: 12.11.2023.

Niittyviita A. 2019. *Storytools of testing.* Prove Expertise Ltd. Oulu.

Pham P., Nguyen.V-L, Nguyen T. H. 2022. *A Review of AI-augmented End-to-End Test Automation Tools*. International Journal of Advanced Computer Science and Applications, vol 5, no 1, 194–202.

Playwright s.a. *Auto-waiting.* Luettavissa: https://playwright.dev/python/docs/actionability. Luettu 15.1.2024.

Robot Framework Foundation s.a.[a]. *Community.* https://robotframework.org/. Luettu: 5.1.2024.

Robot Framework Foundation s.a.[b] *1.1.1 Why Robot Framework?* Luettavissa: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html Luettu: 1.10.2023.

Robot Framework Foundation s.a.[c]. *1.1.2 High-level architecture* Luettavissa: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#high-level-architecture Luettu: 1.10.2023.

Robot Framework Foundation s.a.[d]. *5.1 Library documentation tool (Libdoc).* Luettavissa: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html Luettu: 5.2.2024.

Robot Framework Foundation s.a.[e]. *SeleniumLibrary.* Luettavissa: https://robotframework.org/SeleniumLibrary/ Luettu: 1.10.2023.

Robot Framework Foundation s.a.[f] *robotframework-browser.* Luettavissa: https://github.com/MarketSquare/robotframework-browser#robotframework-browser Luettu 25.10.2023.

Robot Framework Foundation s.a.[g]. *DataDriver for Robot Framework.* Luettavissa: https://github.com/Snooz82/robotframework-datadriver. Luettu: 20.11.2023.

Robot Framework Foundation s.a.[h] *2.2.8 Different test case styles.* Luettavissa: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#different-test-case-styles. Luettu 15.10.2023.

Robot Framework Foundation s.a.[i] *BDD (Behavior Driven Development).* Luettavissa: https://docs.robotframework.org/docs/testcase_styles/bdd. Luettu: 5.1.2024.

Robot Framework Foundation. *BuiltIn.* 27.11.2011. Luettavissa: https://robotframework.org/robotframework/2.6.1/libraries/BuiltIn.html Luettu: 11.12.2023.

Rytkönen M. Kakkonen K. 2023. *ACT 2 LEAD - Ohjelmistotestauksen johtamisen käsikirja.* Ketterät Kirjat Oy. Tuusula.

ISO/IEC/IEEE 29119-1:2022(E) 2022. *International Standard - Software and systems engineering - -Software testing --Part 1:General concepts*. Institute of Electrical and Electronics Engineers, Inc. New York.

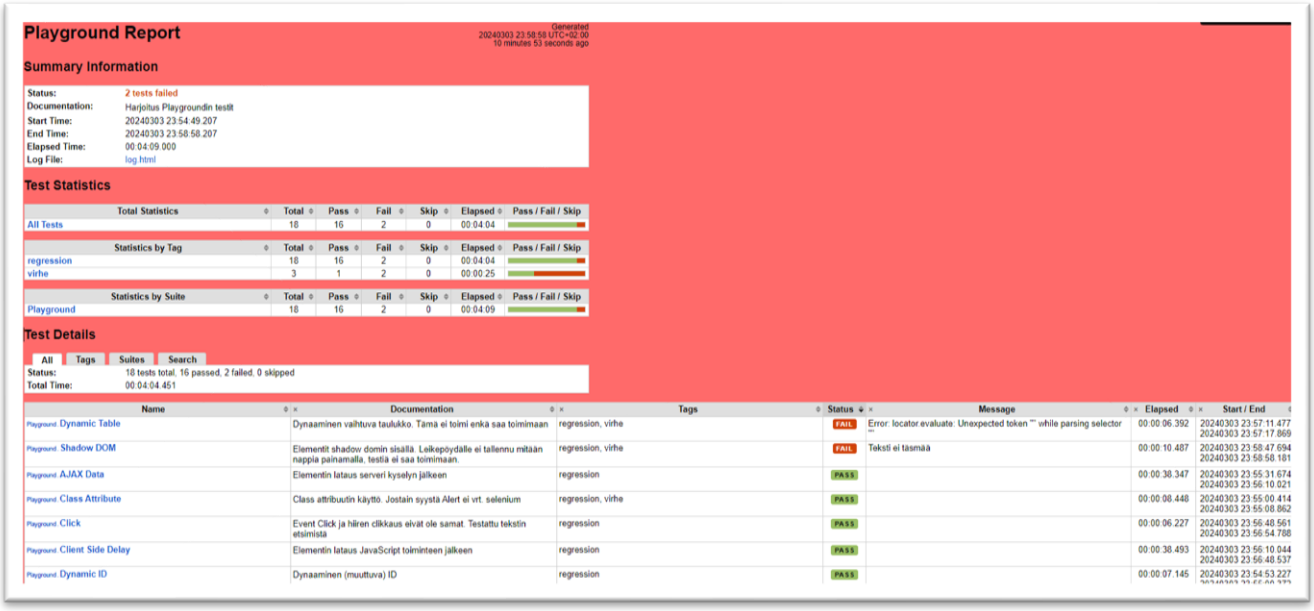
W3Schools s.a. *HTML id Attribute*. Luettavissa: https://www.w3schools.com/htmL/html_id.asp Luettu: 8.4.2024.

Liitteet

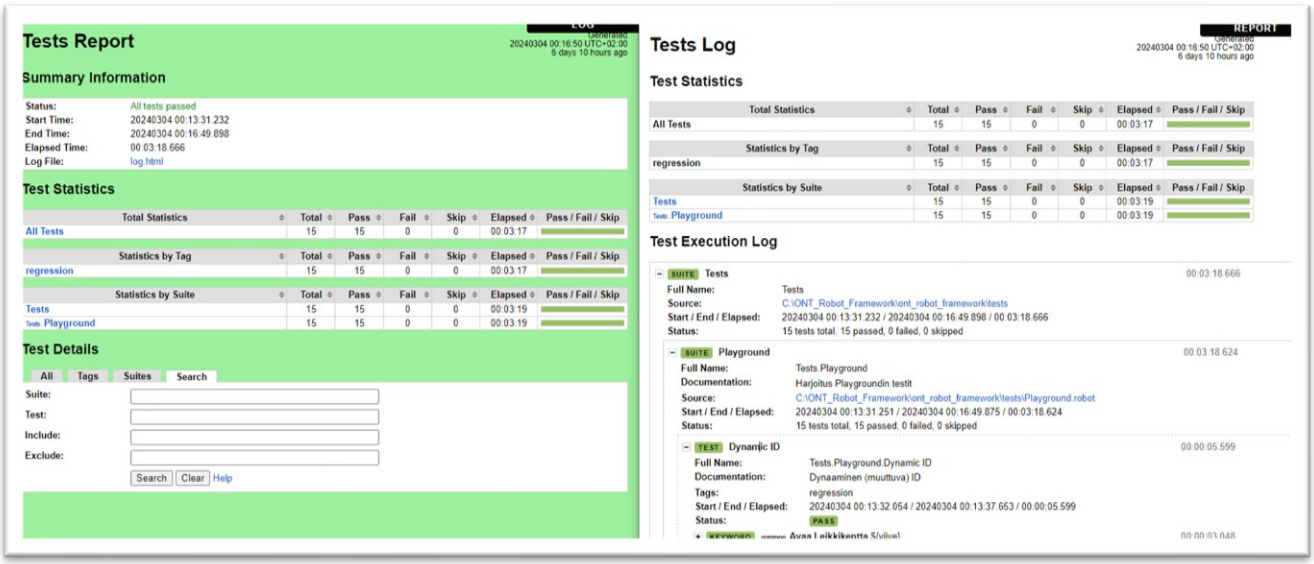
Liite 1. Testauksen tyypit (mukailen Rytkönen & Kakkonen 2023, 133–163)

Testaustyyppi	Kohde	Milloin suoritetaan	Testien johtaminen
Yksikkötestaus / Unit-testaus	Koodin funktiot ja metodit tai muut yksittäiset osat	Jatkuvasti osana ohjelmointia	Koodi
Integraatiotestaus	Yhdistettävien järjestelmänosien rajapinta	Järjestelmän osan luonnissa tai jos järjestelmän osia muutetaan	Rajapintamäärittelyt
Toiminnallinen testaus	Uusi ominaisuus ja sen toiminta	Kun lisätään uusi ominaisuus	Käyttäjätarinat ja määrittelyt
Saavutettavuustestaus	Käyttöliittymä, sen käytettävyys erilaisille käyttäjille	Järjestelmän valmistelussa, voidaan toistaa, kun tehdään muutoksia käyttöliittymään	Yleiset saavutettavuusohjeet ja standardit, esimerkiksi WCAG
Regressiotestaus	Järjestelmä kokonaisuudessaan tai valituin osin	Versioiden yhteydessä, aina kun koodiin tehdään muutoksia	Käyttäjätarinat, käyttötapaukset, käyttöohjeet ja kriittiset ominaisuudet
Hyväksymistestaus	Järjestelmä kokonaisuudessaan tai määritellyin osin	Ennen järjestelmän tai version käyttöönottoa	Määritykset, käyttäjätarinat
Rajapinta / API-testaus	Yhdistettävien järjestelmien rajapinnat	Uuden järjestelmien välisen rajapinnan luonnissa ja kun rajapintaa muutetaan	Rajapintamäärittely
Suorituskykytestaus	Järjestelmän kuorman kesto eri näkökulmista	Käyttöönoton yhteydessä, tarvittaessa ongelmien ilmetessä	Määrittelyt, normaali kuormituksesta suhteuttamalla
Tietoturvatestaus	Järjestelmän tietoturvan testaamista	Käyttöönoton yhteydessä, uusien ominaisuuksien lisäyksen yhteydessä. Sisältää myös järjestelmän ulkopuolista tarkastelua	Määrittelyt ja vaatimukset, yleiset hyvät käytännöt tai tunnetut riskit
Selain/laitetestaus - Yhteensopivuustestaus	Testataan että järjestelmä toimii erialustoilla	Käyttöönoton yhteydessä	Vaatimukset, kohdekäyttäjät ja heidän laitteensa

Liite 2. Robot Frameworkin raportit



Virheitä sisältäneen testisarjan raportti



Onnistuneen testisarjan raportti ja loki