jamk.fi

Laitehallinta-sovelluksen käyttöliittymän ja API-rajapinnan testausautomaatio

Timo Heinonen

Opinnäytetyö
Toukokuu 2020
Tekniikan ja liikenteen ala
Insinööri (AMK), Tieto- ja viestintätekniikka
Ohjelmistotekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Heinonen, Timo	Opinnäytetyö, AMK	Toukokuu 2020
	Sivumäärä 45	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Laitehallinta-sovelluksen käyttöliittymän ja API-rajapinnan testausautomaatio

Tutkinto-ohjelma
Insinööri (AMK), tieto-ja viestintätekniikka

Työn ohjaaja(t)
Marko Rintamäki, Esa Salmikangas

Toimeksiantaja(t)
Nodeon Finland Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli toteuttaa toimeksiantajalle Laitehallinta-sovelluksen käyttöliittymän sekä ohjelmointirajapinnan testausautomaatioratkaisu. Työn tavoitteena oli varmistaa web-sovelluksen kriittisten ominaisuuksien toiminta sekä sulauttaa testausautomaatioratkaisu osaksi ohjelmistokehitysprosessia.

Käyttöliittymä- sekä rajapintatestaus toteutettiin Robot Framework-ohjelmistokehyksellä ja sen eri kirjastoilla. Testitapaukset käsittelivät erilaisten laitteiden, kalustoiden sekä järjestelmien hallinnan testausta. Käyttöliittymätestauksessa käytettiin SeleniumLibrary-avainsanakirjastoa. Rajapintatestaus suoritettiin RESTInstance-avainsanakirjastoa hyödyntäen.

Automaatiotestit sulautettiin osaksi Microsoft Azure Pipelines automatiosointipalvelua ja testauksen suoritti Microsoftin ylläpitämä virtuaalikone. Työssä määritettiin automatisointiputken toiminnan märittelevä YAML-tiedosto.

Työn tuloksena saatiin toteutettua toimiva testausautomaatioratkaisu. Työstä jäi jatkokehitettävää liittyen testitapausten suorittamisen luotettavuuteen sekä virtuaalikoneen suorittamien testitapausten lokitiedostojen turvalliseen säilytykseen.

Avainsanat (asiasanat)
Automaatiotestaus, Testausautomaatio, Robot Framework, Selenium

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Heinonen, Timo	Bachelor's thesis	May 2020
		Language of publication: Finnish
	Number of pages: 45	Permission for web publication: x

Title of publication

Test automation solution for UI and programming interface of device management application

Degree programme

Information and Communication Technology

Supervisor(s)

Rintamäki, Marko; Salmikangas, Esa

Assigned by

Nodeon Finland Oy

Abstract

The aim of this thesis was to implement a test automation solution for the client's device management application. The testing is targeted towards the application's user interface and the programming interface. The objective was to ensure the operation of the critical functionalities of the application and to integrate the test automation solution into the software development process.

The test automation was implemented by using Robot Framework and many of its keyword libraries. The test cases test the management of various devices and sites. SeleniumLibrary was used in the testing of the user interface. Testing of the programming interface was implemented using RESTInstance keyword library.

Automated tests were integrated with Microsoft Azure Pipelines automation service where the tests were run by Microsoft-hosted virtual machine. The YAML file defining the operation of the automated pipeline was configured in this thesis.

This thesis resulted as the implementation of a functional test automation solution. The implementation left room for further development in terms of the reliability of test case execution and the secure storage of test case log files generated by the virtual machine.

Keywords/tags (subjects)

Test automation, Automation Testing, Robot Framework, Selenium

Miscellaneous (Confidential information)

Sisältö

Kuviot

Sanasto

API

Application Programming Interface eli ohjelmointirajapinta toimii eri ohjelmien välisen keskustelun rajapintana.

DevOps

Toimintamalli, joka pyrkii automatisoimaan ohjelmiston kehityksen, testauksen sekä ylläpidon.

FTP

File Transfer Protocol on tiedonsiirtomenetelmä kahden tietokoneen välillä hyödyntäen *TCP*-protokollaa.

JSON

Javascript Object Notation on avoimen standardin tiedostomuoto tiedonvälitykseen.

JUnit

Java-ohjelmistokoodin yksikkötestaukseen tarkoitettu ohjelmistokehys. On osa *xUnit*-ohjelmistokehysryhmää.

Testiskripti

Komentosarja tai lyhyt ohjelma, jolla kuvataan testitoimenpiteet.

XPath

XML Path Language on kyselykieli, jota käytetään *XML*-dokumenttien osien osoittamiseen.

xUnit

Kollektiivinen nimi useille yksikkötestaukseen tarkoitetuille ohjelmistokehyksille, joiden rakenne ja toiminnallisuus polveutuvat *SmallTalk*-ohjelmistokielen *SUnit*-ohjelmistokehyksestä

1 Johdanto

Tämän opinnäytetyön toimeksiantajana toimi vuonna 2013 perustettu Nodeon Finland Oy, joka tuottaa erityisesti teknologiaratkaisuiden suunnittelua ja toteutusta kriittisen infrastruktuurin ympäristöissä. Asiantuntijuutta löytyy automaatio- ja sähkösuunnittelusta, tietoliikenneratkaisuista sekä ohjelmistosuunnittelusta (Lyhyesti n.d). Nodeonin toimipiste sijaitsee Jyväskylässä ja se työllistää noin kolmekymmentä henkilöä.

Opinnäytetyön tarkoituksena oli toimittaa testausautomaatioratkaisu Nodeonin sisäiseen käyttöön kehitetylle Laitehallinta-sovellukselle. Työn tavoitteena on mahdollistaa regressiotestaus sovelluksen React-käyttöliittymälle sekä REST API-rajapinnalle. Testitapausten tulisi kattaa sovelluksen kriittisimmät ominaisuudet, joihin kuuluu erilaisten järjestelmien, kalustoiden sekä laitteiden hallinta sovelluksen käyttöliittymän kautta. Testausautomaatioratkaisu tulisi myös integroida osaksi *Azure DevOps* ympäristöä. Käyttöliittymän ja rajapinnan testauksen toteuttavaa automaatiotestausteknologiaa ei ole tilaajan puolesta ennalta määritetty, mutta tilaaja suosii avoimen lähdekoodin teknologiaratkaisuja.

2 Ohjelmistokehitys ja testausautomaatio

2.1 Yleistä

Ohjelmiston testaus on tärkeä osa ohjelmistokehitystä. Se varmistaa sen, että tuotettu ohjelmistokoodi on eheää ja toimii tarkoitetulla tavalla. Yksikään ohjelmoija ei tuota täydellistä ohjelmistokoodia ja on altis virheille. Testaus tuo lisäkuluja ohjelmiston kehitysvaiheessa, mutta säästää resursseja julkaisun jälkeen mahdollisissa ohjelmistokoodin korjauksissa. Rikkinäisen tai puutteellisen ohjelmiston toimittaminen asiakkaalle voi vahingoittaa yrityksen julkista imagoa ja tulevaisuuden liiketoimintaa. (Bastanzhieva 2019.)

Ohjelmiston tuotejulkaisuprosessissa on usein paljon toistuvia, ihmisen suorittamia, työvaiheita. Näiden töiden automatisointi säästää aikaa ja resursseja. DevOps-toimintamalli pyrkii automatisoimaan sekä virtaviivaistamaan tuotteiden julkaisua ja tuotannon ylläpitoa. Tämä ohjelmistokehittäjien ja palvelinylläpidon välisen yhteistyön automatisoinnin tarve on suurempi isommissa yrityksissä, koska henkilöstön kasvaessa kehittäjien ja ylläpidon välinen kommunikaatio vaikeutuu. (Klemetti 2013.)

Kanerin (2000, 3.) mukaan testausautomaatio, jossa ihminen joutuu manuaalisesti suorittamaan automatisoidut testitapaukset, on vain tietokoneavustettua testausta. Tämä voidaan korjata ottamalla *DevOp*s-tomintamalli käyttöön, jolla automatisoidaan testitapausten suoritus osana jatkuvaa integraatiota ja kehitystä.

Jatkuvalla integraatiolla (engl. Continious integration) tarkoitetaan prosessia, joka automatisoi ohjelmakoodin käännöksen ja testauksen aina kun ohjelmistokehittäjä julkaisee muutoksen versionhallintaan. Automaatiotestaus osana jatkuvaa integraatiota varmistaa, että ohjelmiston laatu pysyy samana koodimuutosten välillä. (Guckenheimer 2017.)

DevOps-toimintamallin käyttöönotto testausautomaation näkökulmasta tuo joitakin haasteita. Testattavasta sovelluksesta täytyy luoda erillinen testiympäristö, jonka koodipohjaa vasten testit suoritetaan. Testien tulokset ja testien ajosta luodut virhelokit tulisi jotenkin saada testiympäristöstä ihmisen luettavaksi. Tämän työn DevOps-toimintamallin toteuttavaksi työkaluksi työn tilaaja määritteli Microsoftin *Azure DevOps* ympäristön, jossa jatkuvan integroinnin ja jatkuvan kehityksen suorittaa *Azure Pipelines* automatisointipalvelu.

2.2 Erilaiset testausmetodit

Testausmetodit voidaan karkeasti jakaa toiminnalliseen ja ei-toiminnalliseen testaukseen. Toiminnallinen testaus tarkoittaa ohjelmiston testaamista määriteltyjä vaatimuksia vasten eli testataan, että ohjelmisto ja sen osat toimivat tarkoitetulla tavalla loppukäyttäjän näkökulmasta. Ei-toiminnallinen testaus pitää sisällään

loppukäyttäjälle ei näkyvän toiminnallisuuden testauksen, kuten esimerkiksi suorituskyky. (Aebersold n.d.)

Regressiotestaus tarkoittaa ohjelmiston testausta jokaisen ohjelmistokoodin muutoksen jälkeen. Tämän tarkoitus on varmistaa, etteivät uudet ohjelmistokoodin muutokset riko ohjelmiston toiminnallisuutta. Regressiotestauksen tavoite on havaita ohjelmiston viat mahdollisimman varhaisessa ohjelmistoprojektin vaiheessa. Taloudellisesti viat ovat paljon halvempia korjata ohjelmiston kehitysvaiheessa kuin ylläpitovaiheessa. Regressiotestauksen haasteena on ylläpitää ja luoda testejä ohjelmistoprojektin kasvaessa. (Daityari 2019.)

Regressiotestauksen suurin ongelma on testien muuttumattomuus. Kun samat testin ajetaan aika toisensa jälkeen, jossain vaiheessa testit eivät enää löydä uusia ohjelmointivirheitä, vaikkakin virheitä olisi olemassa. Ratkaisuna on suorittaa uusia testejä regressiotestauksen yhteydessä. (Kaner 2000, 8.)

Hyväksyntätestaus on toiminnallisen testauksen viimeinen vaihe. Siinä määritetään täyttääkö ohjelmisto sille asetetut vaatimukset. Ketterässä ohjelmistokehityksessä on tärkeää saada asiakkaan palaute jokaisesta ohjelmistoversiosta, jotta ohjelmiston kehitys voidaan viedä asiakkaan haluamaan suuntaan mahdollisimman varhaisessa vaiheessa. (Aebersold n.d.)

2.3 Automatisoidun ja manuaalitestauksen erot

Termeillä testausautomaatio (engl. Test automation) ja automaatiotestaus (engl. Automated testing) on eroja. Automaatiotestaus on määritettyjen testitapausten automatisointia, jotka muuten suoritettaisiin manuaalisesti. Testausautomaatio taas käsittää testien automatisoidun hallinnan, seurannan sekä ajon. (McMeekin 2017.)

Manuaalisessa ohjelmistotestauksessa ihminen suorittaa testitapaukset ilman työkaluja tai ohjelmistoskriptejä. Web-sovelluksen käyttöliittymätestauksessa testaaja menee itse sovellukseen ja suorittaa testiaskeleet hiirtä ja näppäimistöä käyttäen. Testaajan tulee itse tarkkailla, käyttäytyykö sovellus tarkoituksen

mukaisesti. Automaatiotestauksessa nämä kaikki työt ovat nimensä mukaisesti automatisoitu. (Mulders 2019.)

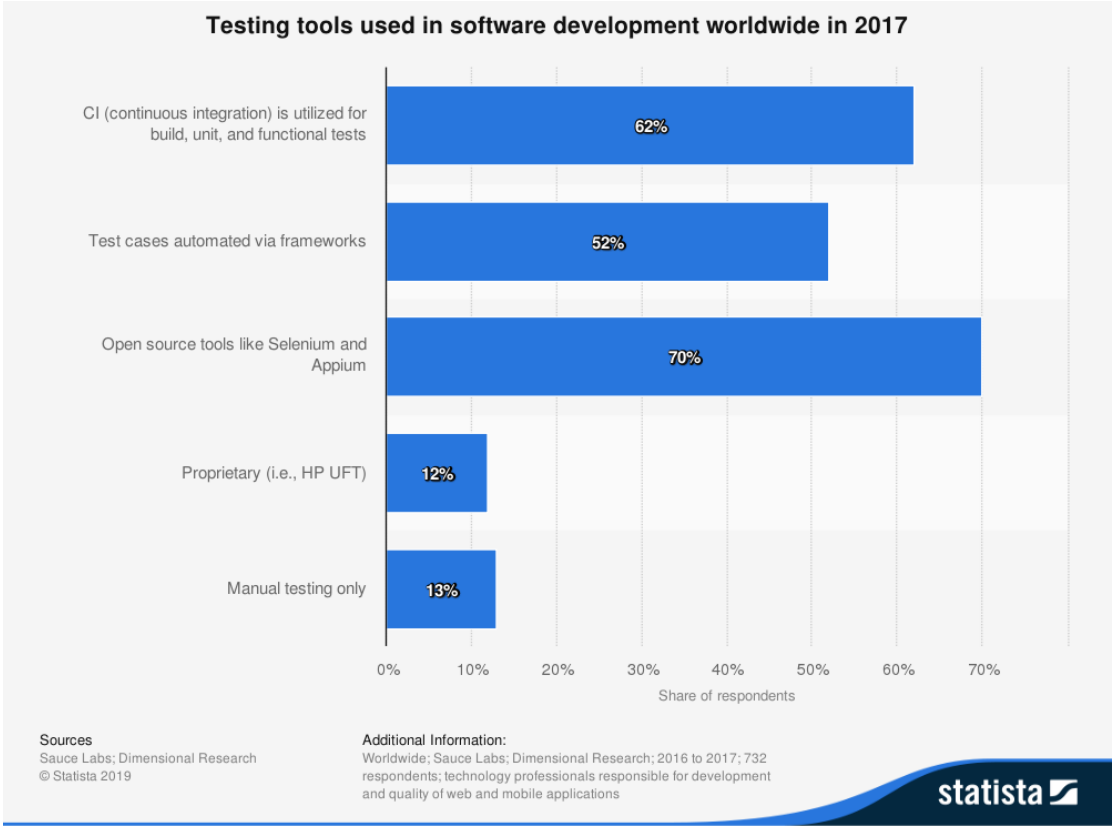
Visuaaliset ohjelmistovirheet ovat helpommin havaittavissa manuaalisella testauksella. On mahdotonta kirjoittaa testiskriptiä, johon on määritelty kaikki käyttöliittymän elementit ja niiden oikeat sijainnit sovelluksessa. Jos edellä mainitun skriptin onnistuu kirjoittamaan, se on erittäin työlästä ylläpitää. Pienikin muutos ohjelmistokoodissa vaatii testitapauksen muokkausta. Pieniä muutoksia ohjelmassa on nopeinta tarkistaa manuaalisesti, kuin ajaa testiskripti. Joissakin tapauksissa, esimerkiksi regressiotestauksessa, manuaalinen testaus vie enemmän aikaa ja resursseja kuin automaatiotestaus. Usein kuvitellaan, että manuaalinen testaus olisi halvempaa. Kulut tulevat ohjelmistojen lisenssien sijasta työntekijöiden palkoista. On olemassa myös avoimen lähdekoodin teknologiavaihtoehtoja automaatiotestaukseen. (Mulders 2019.)

2.4 Automaatiotestaustyökalut

2.4.1 Yleistä

Automaatiotestaustyökalulla tarkoitetaan ohjelmistokehystä tai sovellusta, joka auttaa testiskriptien toteutuksessa ja ajaa niitä automaattisesti. Todellisen testausautomaation saavuttamiseksi on työkalu pystyttävä integroimaan osaksi jatkuvaa integraatiota. Työssä käytettävän testaustyökalun yksi valintakriteereistä on siis mahdollisuus integroida työkalu osaksi *Azure DevOps* ympäristössä jatkuvan integraation toteuttavaa *Azure Pipelines* automatisointipalvelua.

Avoimen lähdekoodin testityökalujen osuus ohjelmistokehityksessä oli 70%:n verran vuonna 2017. Maksullisten työkalujen osuus oli 12%:n verran. Osana testausta jatkuvaa integraatiota hyödynsi 62%:n verran. Nämä tiedot selviävät kuviosta 1.



Kuvio 1. Ohjelmistokehityksessä käytetyt testaustyökalut vuonna 2017 (Liu 2018)

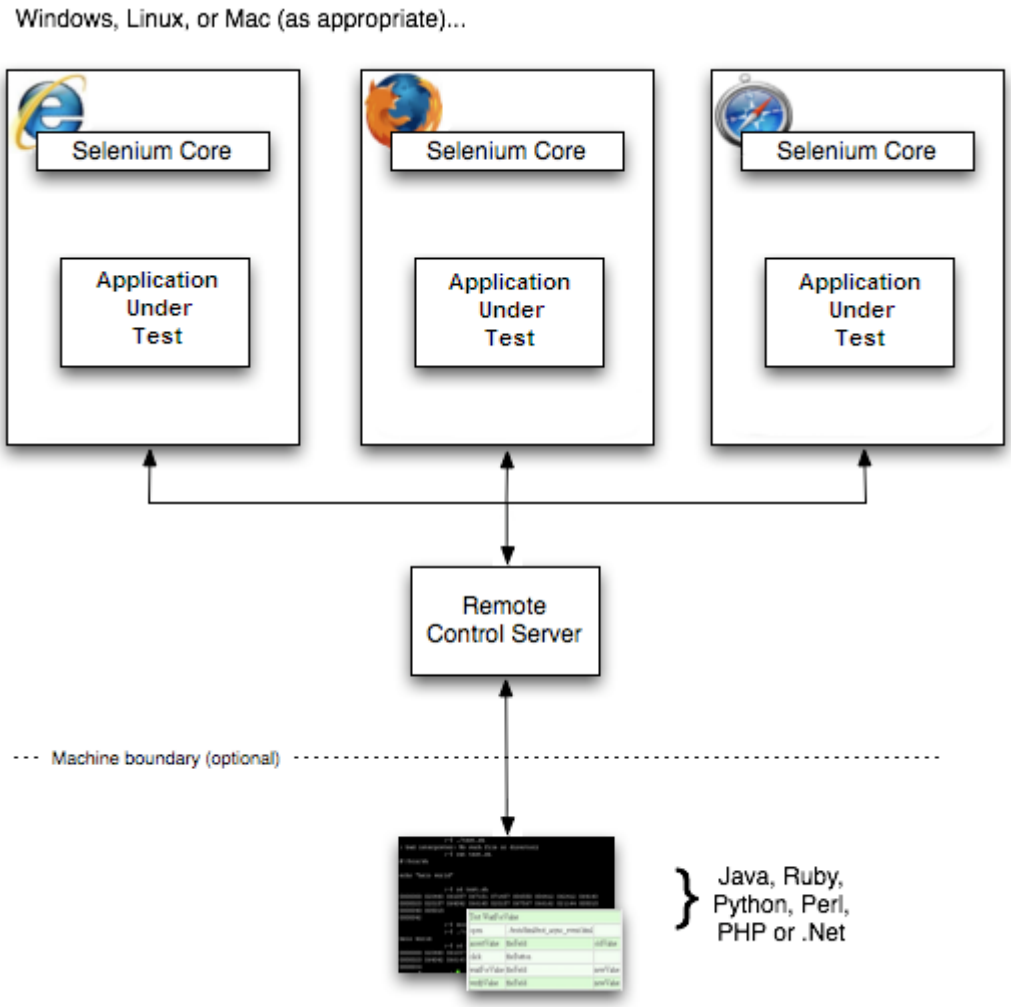
2.4.2 Selenium

Selenium on suosituin vaihtoehto web-käyttöliittymätestaukseen ja toimii pohjana monelle eri testityökalulle ja -kehykselle. Selenium testaa sovellusta simuloimalla loppukäyttäjän tekemiä toimintoja. Selenium koostuu monesta eri työkalusta, mutta sen ydin on Selenium WebDriver.

Selenium IDE (Integrated Development Environment) työkalulla on mahdollista nauhoittaa ja toistaa automaatiotestiskriptejä. Nauhoitetut testit voi ladata työkalusta usealla eri ohjelmistokielellä. Työkalun voi ladata laajennuksena *Google Chrome* tai *Mozilla Firefox* verkkoselaimiin. Selenium IDE oli ennen nimeltään *Selenium recorder.* (Sadiq 2020.)

Selenium RC (Remote Control) on testien suorittamiseen tarkoitettu testausohjelmistokehys, joka oli ensimmäisessä Seleniumin versiossa Seleniumin

ydin. Selenium RC koostuu palvelimesta ja asiakaspään kirjastoista. Palvelin hallinnoi verkkoselaimia, välittää Selenium komentoja testiohjelmalta verkkoselaimeen sekä toimii HTTP-välipalvelimena. Asiakaspään kirjastot toimivat rajapintana Selenium RC palvelimen ja ohjelmointikielen välillä. Kehyksen arkkitehtuuri todettiin olevan liian monimutkainen ja ominaisuuksiltaan rajallinen verrattuna Selenium WebDriveriin, johon Selenium RC yhdistyi Seleniumin toisessa versiossa. Selenium RC:tä ei enää tueta. Kuviossa 2 on kuvattu yksinkertainen rakenne Selenium RC:n arkkitehtuurista. (Selenium 1 (Selenium RC) 2020.)

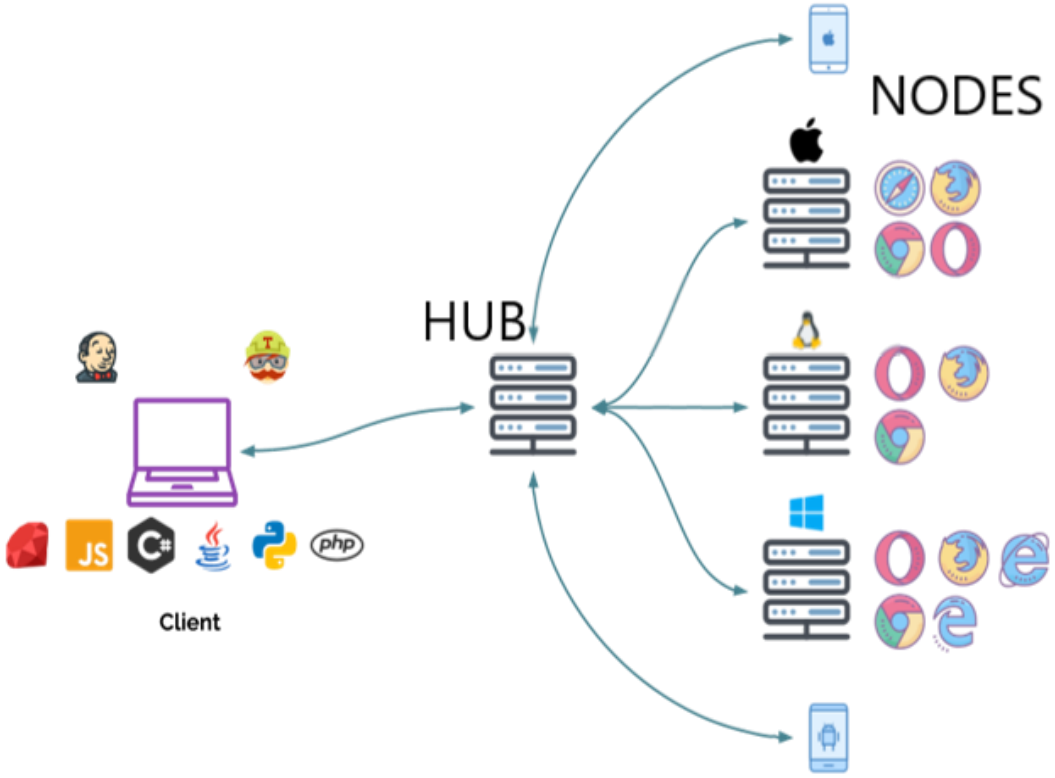


Kuvio 2. Selenium RC yksinkertaistettu arkkitehtuuri (Selenium 1 (Selenium RC) 2020.)

Selenium WebDriverin käyttötarkoitus on sama kuin Selenium RC:llä, mutta on helppokäyttöisempi ja arkkitehtuuri on yksinkertaisempi. Arkkitehtuurillisesti isoin

muutos on palvelimen puuttuminen täysin. WebDriver tarvitsee ajamiseen vain testiskriptin, jossa on Selenium komennot, sekä verkkoselaimen. WebDriver kommunikoi verkkoselaimelle ajurin avulla. WebDriver antaa komentoja ajurille, joka välittää ne selaimelle ja palauttaa selaimessa tapahtuvat tiedot takaisin WebDriverille. Ajurit ovat selainkohtaisia eivätkä toimi ristiin, esimerkiksi Google Chrome selaimen kanssa kommunikointiin tarvitaan ChromeDriver ja taas Mozilla Firefoxin kanssa GeckoDriver. Webdriver tukee myös verkkoselaimia, joissa ei ole graafista käyttöliittymää toisin kuin Selenium RC. (Understanding the Components 2020.)

Useita testejä on mahdollista ajaa samanaikaisesti Selenium Grid välityspalvelimen avulla. Selenium Grid rakentuu yhdestä keskuksesta eli hubista ja yhdestä tai useammasta nodesta. Kuviossa 3 on kuvattu Selenium Gridin rakenne. Kuviossa esiintyvä Client-kone on jokin laite, joka käynnistää testien suorittamisen.



Kuvio 3. Selenium Gridin rakenne (Components of a Grid 2020)

Nodet voidaan konfiguroida suorittamaan testit eri verkkoselaimilla ja niiden eri versioilla. Siten saadaan todettua sovelluksen toimivuus eri ympäristöissä. Selenium Gridin avulla voidaan myös nopeuttaa testien suoritusaikaa jakamalla eri testit suoritettavaksi eri nodeille. Tämä vaatii sen, että testit on suunniteltu niin etteivät ne ole riippuvaisia toisistaan ja voidaan suorittaa missä järjestyksessä tahansa. (Components of a Grid 2020)

2.4.3 Katalon Studio

Katalon Studio on Katalon LLC:n kehittämä testausautomaatiosovellus, joka on kehitetty Seleniumin ja Appiumin pohjalta. Katalon soveltuu erityisesti web-käyttöliittymäautomaatiotestiskriptien luomiseen ja uudelleen ajamiseen ilman, että käyttäjän tarvitsee osata ohjelmoida. Katalonin luomia testiskriptejä pystyy kuitenkin muokkaamaan Apache Groovy-ohjelmointikielellä. (The Good and the Bad of Katalon Studio Automation Testing Tool 2019.)

Testitapausten nauhoitus on työkalun kiinnostavin ominaisuus. Käyttäjä pystyy manuaalisesti itse suorittamaan testin, jonka aikana Katalon Studio tallentaa käyttäjän klikkaamien elementtien XPath-lausekkeet. Lausekkeiden avulla pystytään paikantamaan elementit verkkosivulta. Tämä on nopea tapa tehdä testiskriptille pohja, mutta käyttäjän on silti itse määritettävä testien hyväksyntäkriteerit. Huono puoli tässä ominaisuudessa on se, että edellä mainitut XPath-lausekkeet ovat absoluuttisia. Tällöin aina kun web-sovelluksen elementtien hierarkia muuttuu edes hieman, on pakko muokata testejä. Parempi tapa olisi käyttää relatiivisia XPath-lausekkeita tai jotain uniikkia elementin attribuuttia esimerkiksi elementin Id.

Työkalu on ilmainen, mutta Katalonin tarjoama tekninen tuki on maksumuurin takana. Pienen käyttäjäyhteisön vuoksi ongelmanratkaisu eri foorumeiden kautta on vaikeaa. Lisäksi Apache Groovy tarjoaa rajalliset mahdollisuudet testiskriptien muokkaukseen. Edellä mainittujen syiden vuoksi Katalon Studiota ei valittu tämän työn toteutukseen.

2.4.4 Robot Framework

Robot Framework on alun perin Nokian kehittämä avoimen lähdekoodin testaus-ohjelmistokehys, joka soveltuu hyväksymistestaukseen ja erilaisten prosessien automatisointiin. Robot Framework ei ole käyttöliittymä- tai ohjelmistoriippuvainen. Testit koostuvat selkokielisistä avainsanoista, jonka ansiosta käyttäjä voi keskittyä testien logiikkaan. Robot Frameworkin sisäänrakennettujen (Built-In) avainsanojen lisäksi voi asentaa kolmannen osapuolen tarjoamia avainsanakirjastoja. Myös itse rakennettuja Python- tai Java-pohjaisia avainsanakirjastoja voi tuoda Robot Frameworkiin. (Introduction n.d.)

Robot Frameworkin suurin etu on käyttöönoton helppous. Käyttäjä ei tarvitse aikaisempaa ohjelmointikokemusta testien toteutukseen ja ajoon selkokielisten avainsanojen ansiosta (Molinero 2019). Selenium WebDriver ajurin tuomat edut käyttöliittymätestaukseen voidaan tuoda Robot Frameworkiin SeleniumLibrary-avainsanakirjastolla.

Robot Framework valikoitui työn toteutukseen. Hyvä dokumentaatio sekä tuen saamisen helppous internetin kautta olivat merkitsevässä roolissa valinnan tekemisessä. Seuraavassa luvussa käsitellään tarkemmin Robot Frameworkin teknistä rakennetta ja työssä käytettäviä avainsanakirjastoja.

2.5 Robot Frameworkin tekninen rakenne

2.5.1 Yleistä

Robot Framework käyttää testausdatan hallitsemiseen *.robot*-päätteisiä tiedostoja. Testausdata käsittää tässä tapauksessa testitapauksia ja niiden sisältämää logiikkaa. Data voidaan jakaa kuuteen eri osioon, mutta kaikkia niitä ei tarvitse käyttää. Suositeltava syntaksi eri osioiden määrittämiseen on kolme asteriskia osion nimen molemmin puolin. Tärkeimmät osiot ovat *Settings, Variables, Test Cases* ja *Keywords.* Resurssi- ja muuttujatiedostot sekä avainsanakirjastot tuodaan Robot Frameworkin käytettäväksi *Settings*-osiossa. Samassa osiossa määritellään metadata testitapauksille ja -joukoille. (Robot Framework User Guide n.d.)

Testausdatan toteuttamiseen voidaan käyttää melkeinpä mitä tahansa ohjelmistoympäristöä. RIDE on ohjelmistoympäristö, joka on erikoistunut Robot Framework testausdatan toteuttamiseen ja testien ajamiseen. RIDE selkeyttää Robot Frameworkin testitapausten taulukkomaisten syntaksin toteuttamista. Tämän työn kehitysvaiheessa kokeiltiin RIDE-ohjelmistoeditorin versioita 1.7.4 ja 1.7.4a1, jotka molemmat olivat liian epävakaita kehitykseen.

Visual *Studio Code* ohjelmistoympäristöön voi ladata *Robot Framework Intellisense* laajennuksen, joka tuo muun muassa *.robot* ja *.resource* tiedostotyyppeihin syntaksikorostuksen. Testit suoritetaan ajamalla komentorivillä *robot*-komento, jonka parametriksi asetetaan ajettavan testaustiedoston nimi.

2.5.2 Testijoukkojen ja -tapausten syntaksi

Testitapaukset Robot Frameworkissä luodaan testitapaustiedostoihin, jotka voidaan järjestää hakemistoihin. Nämä tiedostot ja hakemistot luovat hierarkkisen testijoukkorakenteen. Testitapaustiedosto luo automaattisesti sisältämistään testitapauksista testitapausjoukon. Testitapausjoukolla (engl. *Test Suite*) voi olla loputon määrä testitapauksia, mutta kuitenkin on suositeltavaa, että niiden lukumäärä olisi alle kymmenen. (Robot Framework User Guide n.d.)

Ylemmän tason testijoukkoja voi luoda järjestämällä testitapaustiedostot hakemistoihin. Ylemmän tason testijoukolla ei voi olla suoraan testitapauksia, vaan ne koostuvat useista testijoukoista. Asetuksia ylemmän tason testijoukolle voidaan määrittää luomalla hakemistoon *__init__.robot*-niminen alustustiedosto. Alustustiedoston rakenne ja syntaksi on samanlainen kuin testitapaustiedostolla, mutta siinä on muutamia rajoituksia. Testitapauksia ei voi määrittää, eikä kaikkia asetuksia tueta. (Robot Framework User Guide n.d.)

Testitapausten syntaksi on taulumainen, jonka sarakkeet erotellaan vähintään kahdella välilyönnillä tai yhdellä tabulaattorilla. Ensimmäisessä sarakkeessa on aina testitapauksen nimi. Testitapaus jatkuu sinne asti, kunnes vastaan tulee joko tyhjä- tai toinen rivi, jolla on ensimmäisessä sarakkeessa jotain. Toisessa sarakkeessa on

yleensä pelkästään avainsanoja, poikkeuksena on, kun avainsanasta palautetaan muuttujaan arvo. Silloin toisessa sarakkeessa on muuttuja ja kolmannessa on avainsana, josta palautusarvo saadaan. Kuviossa 4 esitetään testitapausten taulukkomainen rakenne. Jokaiselle testitapaukselle voi myös asettaa testitapauskohtaisia asetuksia. Asetukset määritellään toisessa sarakkeessa hakasulkujen sisällä ja niissä voidaan määrittää esimerkiksi testitapauksen alustus ja purku. (Robot Framework User Guide n.d.)

```
*** Test Cases ***
Valid Login
    Open Login Page
    Input Username    demo
    Input Password    mode
    Submit Credentials
    Welcome Page Should Be Open

Setting Variables
    Do Something    first argument    second argument
    ${value} =    Get Some Value
    Should Be Equal    ${value}    Expected value
```

Kuvio 4. Testitapausten syntaksi (Robot Framework User Guide n.d.)

Testintapauksen alustukseen (engl. Test Setup) voi määrittää avainsanan, joka ajetaan ennen testitapausta. Jokaiselle yhteen testitiedostoon kuuluvalle testitapaukselle voi asettaa yhteisen alustuksen asettamalla *Settings*-osioon *Test Setup*-asetus. Testitapauksen purkua (engl. Test Teardown) voidaan hallinnoida samalla tavalla kuin testin alustusta. Purku suoritetaan testitapauksen ajon jälkeen, ottamatta kantaa onnistuiko testi vai ei. Purku määritellään testitapaukselle hakasulkujen sisällä toisessa sarakkeessa muiden avainsanojen jälkeen. Testitapauksen alustuksen ja purun voi olla määrittelemättä antamalla argumentiksi *NONE* tai ei mitään. Koko testijoukolle voidaan asettaa alustus tai purku *Suite Setup*- tai *Suite Teardown*-asetuksilla, jotka ajetaan vain kerran ennen tai jälkeen testijoukon suorituksen. (Robot Framework User Guide n.d.)

2.5.3 Muuttujat

Variables-osiossa nimensä mukaisesti määritellään testausdatan käyttämät muuttujat, jotka voidaan jakaa skalaarimuuttujiin, listoihin, assosiaatiotauluihin sekä

ympäristömuuttujiin. Muuttujat koostuvat tyypintunnisteesta ja aaltosuluista, joiden sisälle muuttujan nimi määritetään. Erilaiset muuttujien tyypintunnisteet ovat esitetty kuviossa 5. Robot Frameworkissä muuttujien kirjainkoolla ei ole merkitystä eikä välilyöntejä oteta huomioon. Kun muuttujaa käytetään skalaarimuuttujana, muuttujan arvoa käytetään sellaisenaan. Jos skalaarimuuttujan ympärillä ei ole avainsanaa tai tekstiä, se käyttäytyy merkkijonomuuttujan tavoin. Skalaarimuuttuja voi olla myös olio. (Robot Framework User Guide n.d.)

```
    ${SCALAR}       Skalaarimuuttuja
    @{LIST}         Lista
    &{DICTIONARY}   Assosiaatiotaulu
    %{ENVIRONMENT}  Ympäristömuuttuja
```

Kuvio 5. Erilaiset Robot Framework muuttujat

Muuttujan arvon ollessa listamainen voidaan muuttujaa käyttää listamuuttujana. Jos avainsanalle syötetään listamuuttuja, avainsanan ensimmäinen argumentti on listan ensimmäinen arvo ja avainsanan toinen arvo on listan toinen arvo ja niin edelleen. Avainsanalle voi listamuuttujan lisäksi syöttää useampia argumentteja kuvion 6 mukaisesti. Robot Framework version 2.9 jälkeen voi listamuuttujaa käyttää skalaarimuuttujana ja toisinpäin. (Robot Framework User Guide n.d.)

```
*** Test Cases ***
Example
    Keyword     @{LIST}     more    args
    Keyword     ${SCALAR}     @{LIST}     constant
    Keyword     @{LIST}     @{ANOTHER}     @{ONE MORE}
```

Kuvio 6. Listamuuttujan käyttö avainsanan argumenttina (Robot Framework User Guide n.d.)

Python assosiaatiotaulun kaltaista muuttujaa voidaan käyttää Robot Framework assosiaatiotauluna, joka on rakenteeltaan lista nimettyjä argumentteja. Assosiaatiotaulua voidaan käyttää samalla tavalla muun datan seassa kuten listamuuttujaa kuviossa 6. Asetuksien argumenttina assosiaatiotaulua voidaan käyttää ainoastaan kirjastojen tuonnissa sekä testijoukon alustamisessa ja purkamisessa. (Robot Framework User Guide n.d.)

Robot Frameworkin ympäristömuuttujat voivat olla ainoastaan merkkijonomuuttujia ja niitä voidaan luoda *Set Environment Variable*-avainsanalla ja poistaa *Delete Environment Variable*-avainsanalla. Edellä mainitut avainsanat ovat osana *OperatingSystem*-avainsanakirjastoa. Ympäristömuuttujat ovat näkyviä kaikille testitapauksille ja niitä voidaan muokata ajon aikana. Tehdyt muutokset ympäristömuuttujaan katoavat testin ajon jälkeen. Myös tavallisia muuttujia voidaan asettaa globaalisti näkyväksi ajon aikana *Set Global Variable*-avainsanalla, joka osana Robot Frameworkin sisäänrakennettua *BuiltIn*-avainsanakirjastoa. Kirjastosta löytyy myös avainsanat *Set Test Variable* ja *Set Suite Variable,* jotka asettavat muuttujan näkyvyyden koko testitapaukselle tai -joukolle. (Robot Framework User Guide n.d.)

2.5.4 Avainsanat

Käyttäjä voi luoda omia ylemmän tason avainsanoja, jotka koostuvat muista avainsanoista. Näistä avainsanoista käytetään termiä *User keyword* erottuakseen kirjastoiden omista alatason avainsanoista. Ylemmän tason avainsanat määritellään *Keywords*-osion alle. Avainsanat määritellään samalla tavalla kuin testitapaukset, ensimmäisessä sarakkeessa nimi ja toisessa avainsanat. Argumentteja voi avainsanalle määrittää avainsanan nimen jälkeen kuvion 7 mukaisesti. (Robot Framework User Guide n.d.)

```
*** Keywords ***
Open Login Page
    Open Browser    http://host/login.html
    Title Should Be    Login Page

Title Should Start With
    [Arguments]    ${expected}
    ${title} =    Get Title
    Should Start With    ${title}    ${expected}
```

Kuvio 7. Avainsanan luonti (Robot Framework User Guide n.d.)

2.5.5 SeleniumLibrary

SeleniumLibrary on kirjasto Robot Frameworkiin, joka tuo Selenium WebDriverin rajapinnasta saatavat Selenium komennot Robot Frameworkin ymmärtämiksi

avainsanoiksi. Kirjaston avulla Robot Frameworkiä voidaan hyödyntää web-käyttöliittymien testaukseen (SeleniumLibrary 2020).

SeleniumLibrary avainsanat tarvitsevat *locator*-nimisen argumentin paikantaakseen elementin verkkosivulta. Argumentti voi olla jokin elementin attribuutti, CSS valitsin tai XPath-lauseke. Joillekin avainsanoille voi antaa uniikkeja argumentteja, esimerkiksi *Click Link* avainsanalle voi antaa *locator* argumentin viittaamaan *href* attribuuttiin oletusattribuuttien lisäksi (SeleniumLibrary 2020).

Avatakseen selaimen SeleniumLibrary tarvitsee Selenium Webdriver ajurin. Selain käynnistetään joko avainsanalla *Open Browser* tai *Create Webdriver. Open Browser* avainsanalle tulee argumenttina verkko-osoite ja verkkoselaimen nimi. Kuviossa 8 on annettu verkko-osoite muuttujana. Jos halutaan käyttää Open Browser avainsanaa, polku WebDriverin ajuriin tulee olla PATH ympäristömuuttujassa.

```
Open Browser    ${BASEURL}  Chrome
```

Kuvio 8. Open Browser avainsanan syntaksi

Jos ei ole mahdollista asettaa ajurin polkua ympäristömuuttujaan, voidaan käyttää avainsanaa *Create Webdriver.* Kuviossa 9 annetaan *executable_path* argumentille polku WebDriverin ajuriin muuttujana.

```
Create Webdriver    Chrome   executable_path=${WEBDRIVER}
Go To       ${BASEURL}
```

Kuvio 9. Create Webdriver avainsanan syntaksi

2.5.6 RESTinstance

RESTinstance on avainsanakirjasto *REST*-rajapintojen testaukseen, joista saadaan dataa *JSON*-muodossa. Testattavan rajapinnan osoite tulee antaa argumenttina kirjastoa tuodessa Robot Frameworkiin. Kuviossa 10 on kuvattuna esimerkki.

```
*** Settings ***
Library           REST  https://esimerkkiUrl.net/api
```

Kuvio 10. RESTinstance kirjaston tuonti Robot Frameworkiin

Kirjaston tuonnin jälkeen voidaan käyttää REST-protokollan mukaisia kutsuja avainsanoina, esimerkiksi *GET* ja *POST*. Argumenttina avainsanoille tulee antaa rajapinnan päätepiste, jota halutaan testata. Avainsana palauttaa *response*-olion, josta saadaan HTTP-vastauksen statuskoodi sekä HTTP-viesti. Kuviossa 11 on kuvattu statuskoodin tarkistus *Number*-avainsanalla.

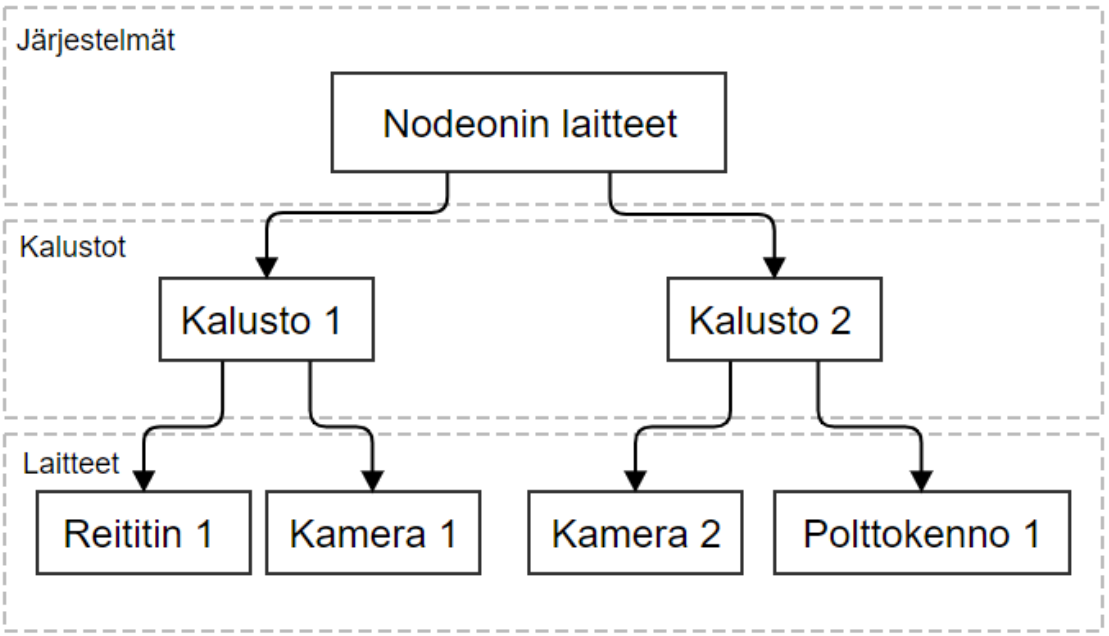
```
GET         /esimerkkipaate
Number      response status     200
```

Kuvio 11. REST-kutsun lähetys ja vastauksen tarkistus

3 Testausautomaatioratkaisun suunnittelu ja toteutus

3.1 Testikohde

Tämän työn automaatiotestauksen kohteena oli Nodeonin sisäiseen käyttöön kehitetty Laitehallinta-sovellus, joka pitää kirjaa Nodeonin eri laitteista ja niiden kokoonpanoista. Laitteet, esimerkiksi reitittimet ja polttokennot, voivat kuulua yhteen kokoonpanoon, kalustoon. Nämä kalustot voidaan taas liittää johonkin järjestelmään. Kuviossa 12 on kuvattu esimerkkitilanne.



Kuvio 12. Laitehallinta-sovelluksen arkkitehtuurin rakenne

Tavoitteena on testata näiden kolmen eri tason toimintaa. Järjestelmiä, kalustoja ja laitteita tulee pystyä luomaan, poistamaan sekä liittämään toisiinsa käyttöliittymästä.

3.2 Testitapausten suunnittelu

Testitapauksien suunnittelussa isoin haaste oli saada testit olemaan riippumattomia toisistaan. Laitteiden irrottamista ja liittämistä kalustoon ei voi testata, jos ei ole olemassa kalustoa. Tähän ongelmaan oli kaksi vaihtoehtoa, joko luodaan uusi kalusto aina ennen kuin testataan laitetta tai luodaan yksi kalusto testien alussa, jota vasten jokaista laitetyyppiä testataan. Sama ongelma pätee myös kaluston testauksessa. Kalusto tarvitsee järjestelmän testausta varten. Uuden kaluston tai järjestelmän luominen aina ennen jokaista testitapausta lisäisi suoritusaikaa huomattavasti.

Ratkaisuna päädyttiin suorittamaan testit tietyssä järjestyksessä. Ensimmäisenä testattaisiin uuden järjestelmän ja kaluston lisäys, jonka jälkeen kaikki yhdeksän laitetyyppiä testataan niitä vasten. Viimeiseksi testataan kaluston irrottaminen järjestelmästä sekä kaluston ja järjestelmän poisto. Se ettei testejä voida suorittaa missä järjestyksessä tahansa on huono käytäntö. Testitapauksia on kuitenkin sen verran vähän ja testejä tulee ajamaan vain yksi virtuaalikone kerrallaan. Tämä kuitenkin rajoittaa tulevaisuudessa skaalautuvuutta, jos testitapauksia tulee

enemmän. Liitteessä 1 on kuvattu testitapaukset ja niiden suoritusjärjestys. Laitetyyppikohtaiset testit tulee myös tehdä tietyssä järjestyksessä, minkä vuoksi jokaisen laitetyypin testit päätettiin koota yhdeksi isoksi testitapaukseksi.

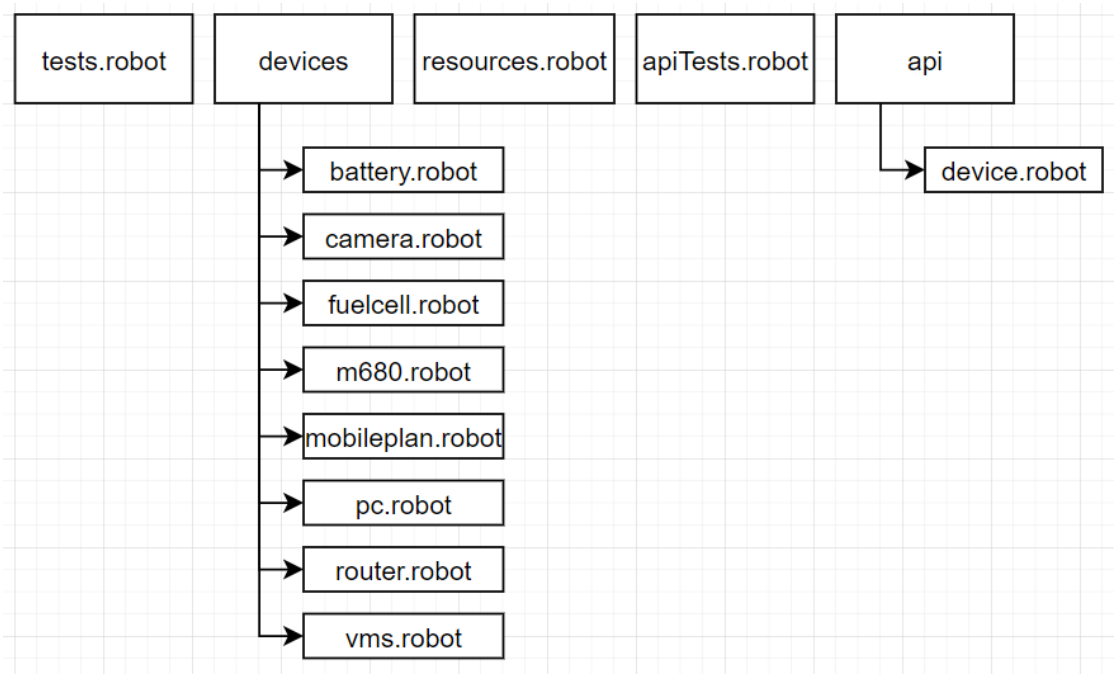
3.3 Ohjelmistojen asennus

Ensimmäinen vaihe on asentaa Python. Useimmissa UNIX-käyttöjärjestelmissä se on esiasennettuna. Windows käyttöjärjestelmälle se pitää asentaa erikseen. Python asennuksen mukana tulee asennustyökalu Pip. Robot Frameworkin asennus tapahtuu ajamalla komentorivillä *pip install robotframework*, jonka jälkeen asennetaan SeleniumLibrary ajamalla *pip install --upgrade robotframework-seleniumlibrary*. SeleniumLibraryn ensimmäisellä asennuskerralla voi *--upgrade* option jättää pois.

Työhön valittiin Seleniumin WebDriver-ajuriksi ChromeDriver. Moduulin voi ladata verkko-osoitteesta *chromedriver.chromium.org/downloads*. On tärkeää ladata sellainen versio ChromeDriverista, joka on yhteensopiva järjestelmään asennetun Google Chrome verkkoselaimen kanssa. Ladattu tiedosto tulee siirtää sellaiseen ohjelmistopolkuun, joka on helposti löydettävissä. Tiedostoa ei tarvitse ajaa.

3.4 Testaustiedostojen hakemistorakenne

Käyttöliittymän testaukseen liittyvistä testitapauksista koottiin vain yksi testijoukko, joka määritellään *tests.robot*-tiedostossa. Testitapaukset koostuvat ylemmän tason avainsanoista, jotka määritellään erilisissä tiedostoissa. Laitteiden testauksessa käytettävät avainsanat on eroteltu laitetyypittäin eri tiedostoihin *devices*-hakemiston alle. *Resources.robot*-tiedosto sisältää järjestelmä- ja kalustotason testaukseen tarvittavat avainsanat sekä testijoukon alustukseen tarvittavat testauslogiikat. Edellä mainitut tiedostot tuodaan *tests.robot*-tiedostoon *Resource-asetuksella*, joka määritellään *Settings*-osiossa. Rajapinnan testaukseen luotiin toinen testijoukko, joka on määritelty *apiTests.robot*-tiedostossa. Kaikki sen tarvitsemat avainsanat on määritelty *api*-hakemiston alla. Testaustiedostojen hakemistorakenne on kuvattu kuviossa 13.



Kuvio 13. Testaustiedostojen hakemistorakenne

3.5 Käyttöliittymän testaus

3.5.1 Testijoukon alustus

Ensimmäisenä Laitehallinta-sovellukseen tultaessa käyttäjä päätyy kirjautumissivulle. Tämä kirjautuminen olisi tarkoitus tehdä vain kerran testien ajon aikana, joten tämä tulisi suorittaa testijoukon alustuksessa. Testijoukon alustuksen argumentiksi annettiin korkeamman tason avainsana *Login,* jossa sisäänkirjautumisen lisäksi alustetaan *ChromeDriver*-ajuri.

Kirjautumissivulla syötetään käyttäjätunnus ja salasana niille tarkoitetuille kentille. Salasanan syöttämiseen *Input Password*-avainsanaa käytetään *Input Text*-avainsanan sijasta, koska *Input Password*-avainsana ei kirjoita lokiin minkä arvon se on salasanakenttään syöttänyt. Robot Frameworkin käyttämät sisäänkirjautumistiedot on asetettu testaustiedoston *Variables*-osiossa muuttujiin, mikä voi olla tietoturvariski. Testaustiedostoa säilytetään pilvessä, johon on vain Nodeonin

työntekijöillä pääsy ja testaustunnuksilla pääsee kirjautumaan vain testausympäristöön, jossa ei säilytetä reaalimaailman laitteiden tietoja.

Testit tullaan ajamaan Microsoftin ylläpitämillä virtuaalikoneilla, *Azure Pipelines agenteilla,* joihin on valmiiksi asennettu *WebDriver*-ajuri. Ajurin polku on kaikissa agenteissa sama, joten polku asetetaan muuttujaan kuvion 14 mukaisesti. Kuviossa on myös kommentoitu paikallista kehitystä varten oleva *Open Browser*-avainsana sekä verkkoselaimen resoluution asettaminen samaksi kuin agentilla. Täten saadaan kiinni kaikki käyttöliittymän resoluutiosta johtuvat graafiset virheet testien kehityksen aikana.

```
*** Variables ***
${WEBDRIVER}      C:\\SeleniumWebDrivers\\ChromeDriver\\chromedriver.exe

*** Keywords ***
Login

    # Azuren koneella toimiva
    Create Webdriver    Chrome    executable_path=${WEBDRIVER}
    Set Selenium Speed    0.1s
    Set Selenium Timeout    10s
    Go To       ${BASEURL}
    Maximize Browser Window
    # Lokaalisti toimiva, käyttäen Azuren VM:n näytön resoluutiota
    # Open Browser    ${BASEURL}  Chrome
    # Set Window Size  1024  613
    Wait Until Page Contains Element    class:login-button    10
    Click Button    tag:button
    Wait Until Element Is Visible    id:username    10
    Input Text    id:username    ${USER}
    Input Password    name:password    ${PASSWORD}
    Click Button    name:action
    Wait Until Page Contains Element    class:body
    Title Should Be    Laitehallinta
```

Kuvio 14. Testijoukon alustus

Työn toteutuksen aikana huomattiin että, Seleniumin normaalilla suoritusnopeudella kaikki käyttöliittymän elementit eivät ehtineet ladata tarpeeksi nopeasti. Tällöin ajuri yritti klikata elementtiä ennen kuin se oli valmis rekisteröimään klikkaustapahtuman. Ratkaisuna oli hidastaa Seleniumin suoritusnopeutta avainsanalla *Set Selenium Speed*

sekunnin kymmenykseen. Työn toteutuksen aikana ongelmaksi muodostui se, että testit toimivat paikallisesti mutta silloin tällöin epäonnistuivat agentilla. Syynä oli se, että paikallisesti käytettiin *Open Browser*-avainsanaa ja agentilla *Create Webdriver*-avainsanaa. *Create Webdriver*-avainsana nollaa kaikki Seleniumin asetukset. *Open Browser*-avainsana ei tätä nollausta tee. Ongelman ratkaisemiseksi siirrettiin Seleniumin asetuksia määrittävät avainsanat *Create WebDriver*-avainsanan jälkeen.

3.5.2 Testitapausten toteutus

Tässä luvussa käydään tarkemmin liitteessä 1 määritettyjen testitapausten toteutus. Testeissä järjestelmistä käytetään termiä *Site,* kalustoista *Device* ja laitteista *SubDevice*. Kaikki testit olettavat olevansa testattavan sovelluksen pääsivulla testien alussa. Sovelluksen sisäänkirjautumisen jälkeen ensimmäisenä testataan pääsivun kaikkien navigaatiopainikkeiden toimivuus. *Click Element*-avainsanalle annetaan argumentiksi *XPath*-lauseke, jonka avulla ajuri löytää oikean painikkeen. Painikkeen klikkauksen jälkeen katsotaan, onko verkkoselaimen osoiterivissä oikea verkko-osoite. Osoitteen ollessa väärin testi epäonnistuu. Testin suorituksen jälkeen tehdään testin purku, jolle on annettu argumentiksi *Go To Main*-avainsana. Testin toteutus on kuvattu kuviossa 15.

```
Check Navigation Buttons
    Click Element    //div[contains(text(), 'Kartta')]
    Location Should Be    ${BASEURL}/
    Click Element    //div[contains(text(), 'Järjestelmät')]
    Location Should Be    ${BASEURL}/sites
    Click Element    //div[contains(text(), 'Kalustot')]
    Location Should Be    ${BASEURL}/devices
    Click Element    //div[contains(text(), 'Tiedostot')]
    Location Should Be    ${BASEURL}/files
    Open SubDevice    Reitittimet
    Location Should Be    ${BASEURL}/routers
    Open SubDevice    VMS-Opastetaulut
    Location Should Be    ${BASEURL}/vms
    Open SubDevice    Teollisuus PC:t
    Location Should Be    ${BASEURL}/pcs
    Open SubDevice    Puhelinliittymät
    Location Should Be    ${BASEURL}/mobileplans
    Open SubDevice    Silmukkamittauslaitteet
    Location Should Be    ${BASEURL}/m680
    Open SubDevice    Polttokennot
    Location Should Be    ${BASEURL}/fuelcells
    Open SubDevice    Kamerat
    Location Should Be    ${BASEURL}/cameras
    Open SubDevice    Akut
    Location Should Be    ${BASEURL}/batteries
    [Teardown]    Go To Main
```

Kuvio 15. Navigaatiopainikkeiden testaus

Laitesivujen testaukseen luotiin uusi ylemmän tason avainsana *Open SubDevice*, jolle annetaan argumenttina painettavan elementin *Id*-attribuutti kuvion 16 mukaisesti. Laitesivut ovat web-käyttöliittymässä sijoitettu erillisen valikon alle, joka avataan klikkaamalla painiketta, jonka Id on *Laitteet.* Sen jälkeen *Mouse Over*-avainsanalla ajuri liikuttaa hiiren halutun elementin päälle ja klikkaa siitä. Muuten ajuri suorittaa askeleet liian nopeasti ja valikko katoaa ennenaikaisesti. Kuviossa 16 määritetyn *Open SubDevice*- avainsanan tarkoituksena on vähentää logiikan toistoa testaustiedostossa.

```
Open SubDevice
    [Arguments]    ${SUBDEVICE}
    Click Element    id:Laitteet
    Wait Until Element Is Visible    id:${SUBDEVICE}
    Mouse Over    id:${SUBDEVICE}
    Click Element    id:${SUBDEVICE}
    Wait Until Element Is Visible    class:subdevice-page
```

Kuvio 16. *Open SubDevice*-avainsanan määrittely

Järjestelmän ja kaluston lisäys sekä hallinta testataan seuraavaksi. Järjestelmän lisäyksen testitapaus on kuvattu kuviossa 17. Testitapausten määrittely *tests.robot*-tiedostossa päätettiin pitää mahdollisimman lyhyinä ja siirtää testauslogiikka erillisiin tiedostoihin. *Add Site*-avainsana on kuvattu kuviossa 18 ja se on määritelty *resources.robot*-tiedostossa. Ensimmäisenä ajuri menee järjestelmän hallintasivulle ja painaa järjestelmän lisäyspainiketta. Ajuri kirjoittaa järjestelmän nimeksi avainsanalle asetetun muuttujan *SITE01* ja omistajaksi *Robot Framework InputText*-avainsanalla, jonka jälkeen ajuri klikkaa tallennuspainiketta. Ajuri odottaa järjestelmäsivulla kymmenen sekunnin ajan uuden järjestelmän ilmestymistä sivulle. Jos sivulle ei ole ilmestynyt *div*-elementtiä, jonka sisällä on *SITE01*-muuttujan arvo tekstinä, oletetaan ettei järjestelmän lisäys onnistunut ja testi epäonnistuu.

```
*** Variables ***
${SITE01}         TestSite

Add Site
    Add Site    ${SITE01}
    [Teardown]    Go To Main
```

Kuvio 17. *Add Site*-testitapaus

```
Add Site
    [Arguments]    ${SITE}
    Click Element    id:Järjestelmät
    Click Element    //span[contains(text(), 'Lisää uusi')]
    Wait Until Element Is Visible    id:AddSiteDiv
    Input Text    name:siteName    ${SITE}
    Input Text    name:owner    RobotFramework
    Click Element    id:submitSite
    Wait Until Element Is Visible  //span[contains(text(), 'Ok')]/..
    Click Element    //span[contains(text(), 'Ok')]/..
    Wait Until Element Is Visible    //div[contains(text(), '${SITE}')]
    Wait Until Element Is Visible    //div[contains(text(), 'RobotFrame-
work')]
    Location Should Be    ${BASEURL}/sites
```

Kuvio 18. *Add Site*-avainsana

Uuden kaluston testaus suoritetaan samanlaisella logiikalla kuin järjestelmän lisäys. Ajuri siirtyy kalustosivulle, lisää uuden kaluston ja tarkastelee lopputulosta. Lisäyksen jälkeen testataan kaluston liittäminen järjestelmään. Jos uuden kaluston tai järjestelmän lisäys aiemmissa testitapauksissa epäonnistui, tämä testi tulee myös epäonnistumaan. Kuviossa 19 on kuvattu ylemmän tason avainsana, joka yhdistää valmiiksi olevan kaluston järjestelmään. Testi onnistuu, jos oikea kalusto ilmestyy järjestelmäsivulle.

```
Attach Device To Site
    [Arguments]     ${SITE}     ${DEVICE}
    Click Element    //div[contains(text(), 'Järjestelmät')]
    Scroll Blueprint Table
    Wait Until Element Is Visible    id:${SITE}MoreButton
    Click Button    id:${SITE}MoreButton
    Wait Until Element Is Visible    id:AppBarMenuButton
    Click Button    id:AppBarMenuButton
    Click Element    id:addDevice
    Wait Until Page Contains Element    id:${DEVICE}
    Click Element    //input[contains(@id, '${DEVICE}')]/..
    Click Element    id:attachDevice
    Wait Until Element Is Not Visible    id:attachDevice
    Wait Until Page Contains Element    id:${DEVICE}
```

Kuvio 19. Kaluston lisäys järjestelmään

Testitapauksen olisi voinut toteuttaa niin, että testitapauksessa itsessään luodaan kyseinen kalusto ja järjestelmä, jonka jälkeen ne liitetään toisiinsa. Tämä tekisi testitapauksesta täysin riippumattoman muista testeistä, mutta testi epäonnistuisi joka tapauksessa, jos uuden kaluston tai järjestelmän lisäys ei onnistuisi. Tämän testitapauksen riippumattomuudesta olisi hyötyä vain, jos testit ajettaisiin useammalla kuin yhdellä koneella samanaikaisesti.

Jokaisen laitetyypin hallinnan testauslogiikka on samanlainen, pieniä eroja lukuun ottamatta. Nämä testitapaukset löytyvät liitteestä 1 numeroilla 5 – 13. Tässä työssä käydään läpi vain yhden laitetyypin, reitittimen, testaus.

Reitittimen hallinnan testitapaus on jaettu neljään osaan: reitittimen lisäys, sen liittäminen ja irrottaminen kalustosta sekä reitittimen poisto. *Test Router*-testitapauksen sisältämät ylemmän tason avainsanat on määritetty *devices*-hakemiston alla *router.robot*-tiedostossa. Nämä ovat kuvattu kuviossa 20. Reitittimen poisto testataan testin purkuna, koska jos jokin edellä olevista testiaskeleista epäonnistuu, halutaan yrittää reitittimen poistoa testausympäristöstä. Ympäristön tietokantaa ei alusteta uudestaan testikertojen välissä, mikä aiheuttaa ongelmia, jos edelliseltä testikerralta on jäänyt testilaitteita kantaan. Tulevaisuudessa kehityskohteena on poistaa reititin suoraan ohjelmistorajapinnasta, jos poisto ei onnistu käyttöliittymän kautta.

```
Test Router
    Add New Routers
    Sub To Device    ${ROUTER01}    ${DEVICE01}
    Remove Router From Device
    [Teardown]  Delete Routers
```

Kuvio 20. *Test Router*-avainsana

Reitittimen lisäyksen testaava *Add New Routers*-avainsana on jaettu kahteen osaan, reitittimen lisäykseen ja tietojen tarkistukseen. Näiden toimenpiteiden jälkeen palataan takaisin etusivulle. Avainsana on kuvattu kuviossa 21.

```
*** Variables ***
${ROUTER01}       TestRouter01

Add New Routers
    Add Router    ${ROUTER01}
    Check Router Values    ${ROUTER01}
    [Teardown]    Go To Main
```

Kuvio 21. *Add New Routers*-avainsana

Jokaisen laitetyypin lisäyssivulla on eri määrä tekstikenttiä eri nimillä, minkä vuoksi jokaisen laitetyypin lisäyksestä tehtiin oma avainsanansa. Kuviossa 22 kuvatun *Add Router*-avainsanan toimintaperiaate on erittäin suoraviivainen. Ajuri navigoin reitittimien hallintasivulle ja klikkaa uuden reitittimen lisäyspainiketta. Uuden reitittimen lisäyssivu on lomakemainen ja sivulla olevat tietojen lisäykseen tarkoitetut tekstikentät voidaan erottaa toisistaan elementin *name*-attribuutilla. Lisäyssivun juurielementin *Id* on *routerDiv*, jonka latausta ajuri odottaa avainsanalla *Wait Until Page Contains Element.* Kun lisäyssivu on ladannut, voi ajuri jatkaa testitapausta eteenpäin. Ajuri syöttää eri tekstikentille argumentiksi saadun *${ROUTER}*-muuttujan sekä tekstikentän nimen. Tämä helpottaa syötettyjen arvojen tarkistusta myöhemmin. Tietojen syötön jälkeen ajuri klikkaa tallennuspainiketta, jonka ajuri paikantaa sivulta *XPath*-lausekkeella. Lauseke valitsee sellaisen elementin, jonka lapsielementti on *Tallenna*-tekstin sisältävä *span*-elementti. Tämä tuo joustavuutta testeihin, koska testi ei ole riippuvainen elementin attribuuteista. Tallennuksen jälkeen ajuri odottaa uuden *${ROUTER}*-muuttujan arvon nimisen reitittimen ilmestymistä reitittimien hallintasivulle.

```
Add Router
    [Arguments]    ${ROUTER}
    Open SubDevice    Reitittimet
    Click Element    id:addNewSub
    Wait Until Page Contains Element    id:routerDiv
    Input Text    name:name    ${ROUTER}
    Input Text    name:model    ${ROUTER}model
    Input Text    name:type    ${ROUTER}type
    Input Text    name:serialNumber    ${ROUTER}serialNumber
    Input Text    name:mac    ${ROUTER}mac
    Input Text    name:imei    ${ROUTER}imei
    Input Text    name:manufacturer    ${ROUTER}manufacturer
    Input Text    name:consumption    ${ROUTER}consumption
    Click Element    //span[contains(text(), 'Tallenna')]/..
    Wait Until Element Is Not Visible    id:routerDiv
    Wait Until Page Contains Element  //div[con
tains(text(), '${ROUTER}')]
```

Kuvio 22. *Add Router*-avainsana

Seuraavaksi ajuri tarkistaa, että reitittimelle syötetyt tiedot tallentuivat oikein. Tiedot tarkistetaan *Check Router Values*-avainsanalla. Reitittimet ovat hallintasivulla asetettu taulukkoon, jossa yhdellä rivillä on kuvattu yksi reititin. Jokaisella reitittimellä on oma muokkauspainikkeensa. Ohjelmistokoodissa on muokkauspainikkeelle asetettu *Id*-attribuutiksi reitittimen nimi, jolla ajuri osaa paikantaa oikean reitittimen muokkauspainikkeen. Ajuri klikkaa painikkeesta ja tarkastaa, että jokaisella tekstikentällä tiedot ovat oikein. Tämän jälkeen ajuri sulkee muokkausikkunan *Peruuta*-painiketta klikkaamalla.

Reitittimen hallinnan testauksessa seuraava testitapaus on reitittimen liittäminen kalustoon, joka tapahtuu kuviossa 23 kuvatulla avainsanalla *Sub To Device,* joka on määritetty *resources.robot*-tiedostossa. Avainsanalle annetaan ensimmäisenä argumenttina laitteen nimi ja toisena kalusto. Testitapauksen alustuksessa ajuri navigoi sovelluksen pääsivulle. Sen jälkeen ajuri menee kalustosivulle ja avaa testattavan kaluston hallintasivun. Sivulta ajuri navigoi painikkeen, josta laite voidaan liittää kalustoon. Painikkeen klikkaus tuo käyttöliittymään listan kaikista laitteista, joille ei ole asetettu kalustoa. Testattavan laitteen valintapainikelle on ohjelmistokoodissa annettu *Id*-attribuuttina merkkijono *${SUB}Select,* jonka ensimmäinen osa on avainsanalle annettu ensimmäinen argumentti. Merkkijonon

toinen osa auttaa ajuria paikantamaan elementin muista elementeistä, joiden *Id*-attribuuttina käytetään laitteen nimeä. Oikean laitteen valinnan jälkeen ajuri klikkaa tallennuspainiketta ja odottaa laitteen ilmestymistä kaluston hallintasivulle. Testi onnistuu laitteen ilmestyessä kaluston hallintasivulle kymmenen sekunnin sisällä.

```
Sub To Device
    [Arguments]    ${SUB}    ${DEVICE}
    [Setup]     Go To Main
    Click Element    //div[contains(text(), 'Kalustot')]
    Location Should Be    ${BASEURL}/devices
    Wait Until Keyword Succeeds    10s    2s    Wait Until Page Contains Ele-
ment    id:${DEVICE}EditButton
    Click Element    //button[@id = '${DEVICE}EditButton']
    Wait Until Element Is Visible    id:deviceMenu
    Click Element    id:deviceMenu
    Mouse Over        //a[@id = 'addSubDeviceToDevice']
    Click Element    //a[@id = 'addSubDeviceToDevice']
    Wait Until Keyword Succeeds    10s    2s    Wait Until Element Is Visi-
ble    //input[@id ='${SUB}Select']/..
    Scroll Element Into View    //input[@id = '${SUB}Select']/..
    Click Element     //input[@id = '${SUB}Select']/../..
    Click Element    //span[text() = 'Lisää valitut']/..
    Wait Until Element Is Visible    //div[contains(text(), '${SUB}')]
    [Teardown]    Go To Main
```

Kuvio 23. Reitittimen liittäminen kalustoon

Kun laite on liitettynä kalustoon, voidaan testata sen irrottamista. Ajuri navigoi takaisin kaluston hallintasivulle. Sivulla on listattu taulukkomaisesti kaikki kalustoon liitetyt laitteet ja jokaisella laitteella on oma irrotuspainikkeensa. Painike-elementti paikannetaan *Id*-attribuutilla *${SUB}Remove*. Kuviossa 24 on annettu avainsanalle argumentiksi laitteen ja kaluston lisäksi laitteen tyyppi, joka annetaan argumentiksi *Scroll SubTable*-avainsanalle. Kaluston hallintasivulla laitteet on jaoteltu taulukkoihin laitetyyppien mukaan. Nämä taulukot piilottavat irrotuspainikkeen käytettäessä käyttöliittymää pienellä resoluutiolla. Painikkeen saa takaisin näkyviin vierittämällä taulua vaakasuunnassa. *SeleniumLibrary*-avainsanakirjaston *Scorll Element Into View*-avainsana ei toimi tässä tapauksessa. Ratkaisuna luotiin ylemmän luokan avainsana, joka suorittaa *Execute Javascript*-avainsanan avulla *JavaScript*-ohjelmistokoodia, jolla

voidaan vierittää taulukkoa. Ajuri klikkaa irrotuspainiketta ja odottaa laitteen katoamista kaluston hallintasivulta.

```
Remove Sub From Device
    [Arguments]    ${SUB}    ${DEVICE}  ${TYPE}
    Click Element    //div[contains(text(), 'Kalustot')]
    Location Should Be    ${BASEURL}/devices
    Wait Until Page Contains Element    id:${DEVICE}EditButton
    Wait Until Keyword Succeeds  10s  2s  Click Element  //button[con-
tains(@id, '${DEVICE}EditButton')]
    Wait Until Element Is Visible     //div[contains(@id, 'subdevice-wrapper')]
    Wait Until Element Is Visible    //div[contains(@class,'device-subDevices')]     20s
    Scroll SubTable     ${TYPE}
    Wait Until Page Contains Element    //button[contains(@id, '${SUB}Remove')]
    Scroll Element Into View    //button[contains(@id, '${SUB}Remove')]
    Click Element    //button[contains(@id, '${SUB}Remove')]
    Click Element    //div[contains(@class, 'deleteAlert')]//child::span[text() = 'Irrota']/..
    Wait Until Element Is Not Visible    //button[contains(@id, '${SUB}Remove')]
```

Kuvio 24. Reitittimen irrottaminen kalustosta

Test Router-testitapauksen purkuna suoritetaan reitittimen poisto. Kaikkien laitetyyppien poiston testauksessa käytetään yhtä *Delete SubDevice*-avainsanaa, jolle annetaan argumenteiksi poistettavan laitteen nimi sekä tyyppi. Reitittimen poiston testaus on erittäin suoraviivainen toimenpide. Ajuri navigoi reitittimien hallintasivulle ja paikantaa halutun reitittimen poistopainikkeen. Painikkeen klikkauksen jälkeen ajuri odottaa laitteen poistumista sivulta.

Laitetyyppien hallinnan testauksen jälkeen testataan kaluston irrotus järjestelmästä, sekä kaluston ja järjestelmän poisto. Näiden testien toimintaperiaate on samanlainen kuin aiemmin käytyjen testien. Testijoukon purkuna käytetään ylemmän tason avainsanaa *Log and Close,* jonka tarkoituksena on kirjoittaa testauslokiin testien aikana syntyneet verkkoselaimen virheviestit sekä sulkea selain. Selaimen virheviestien näyttämiseksi luotiin oma avainsanakirjasto *Python*-ohjelmointikielellä, mutta haluttua toiminnallisuutta ei tähän työhön saatu toteutettua. Kirjasto tuodaan Robot Frameworkiin samalla tavalla kuin muutkin kirjastot *Settings*-osion alla. *Library*-asetuksen argumentiksi annetaan *Python*-tiedoston polku. Tuodun tiedoston

funktioita voidaan käyttää avainsanoina ja niille voidaan antaa argumentteja ja palautusarvoja.

3.6 API-rajapinnan testaus

Tässä luvussa käytetään API-rajapinnasta lyhennettyä termiä rajapinta.

Työn toteutusta tehtäessä huomattiin käyttöliittymätestien testaavan myös rajapinnan toimintaa. Jokainen käyttöliittymätestitapaus testasi, miten rajapinta käyttäytyy, kun käyttöliittymästä luodaan ja poistetaan erilaisia laitteita. Jos rajapinta ei olisi toiminut halutulla tavalla, ei yksikään käyttöliittymätesti olisi onnistunut. Tämän vuoksi päätettiin, ettei tehdä erillistä rajapintatestausratkaisua. Päätettiin kuitenkin luoda esimerkkitestitapaus, joka testaa rajapinnan toimintaa ilman käyttöliittymää.

REST-kutsujen autentikointi oli rajapintatestauksen isoin haaste. Käyttäjän kirjautuessa Laitehallinta-sovellukseen, REST-kustujen autentikointiin tarvittava avain luodaan verkkoselaimen istunnon muistiin (engl. *Session storage*). Tämä muuttuja vaihtuu joka kirjautumiskerran jälkeen. Käyttöliittymästä ei siis päästä täysin eroon rajapintatestauksessa, sillä ennen testausta ajurin täytyy kirjautua käyttöliittymään. Kirjautuminen suoritetaan testijoukon alustuksessa samalla tavalla kuin käyttöliittymätestauksessa. Kirjautumisen jälkeen ajuri suorittaa *JavaScript*-ohjelmistokoodilauseen, jonka avulla istunnon muistista sijoitetaan käyttäjän autentikointiolio Robot Frameworkin oliomuuttujaan *${TOKENOBJ}*. Tästä oliosta parsitaan autentikointiavain *${TOKEN}*-muuttujaan, joka asetetaan näkyväksi koko testijoukolle avainsanalla *Set Suite Variable*. *SessionStorage.getItem*-funktiolle tulee antaa argumentiksi olion nimi, joka halutaan hakea istunnon muistista. Kuviossa 25 edellä mainittu argumentti on kuvattu tyhjällä muuttujalla.

```
GET Token
    ${TOKENOBJ}=  Execute Javascript  return sessionStorage.getItem(${})
    ${TOKEN}=     Execute Javascript  return ${TOKENOBJ}.access_token
    Set Suite Variable   ${TOKEN}
```

Kuvio 25. Autentikointiavaimen hakeminen muuttujaan

Rajapinnan testauksen esimerkkitestitapauksena päätettiin testata kaluston hallintaan liittyviä rajapinnan päätepisteitä. Testitapaus koostuu muutamasta ylemmän tason avainsanasta, jotka on kuvattu kuviossa 26. Kuviossa 27 on kuvattu avainsana *GET all devices*, joka hakee rajapinnasta kaikki kalustot sekä tarkistaa että *Http*-vastauskoodi oli 200 eli haku onnistui. Tämän jälkeen avainsana asettaa kaikkien kalustoiden *Id*-attribuutit erilliseen tiedostoon seuraavia testejä varten.

```
Test Device API
    GET all devices
    GET device by id
    POST & DELETE device
```

Kuvio 26. Test Device API-testitapaus

RESTInstance-avainsanakirjastosta löytyvällä *GET*-avainsanalla voidaan suorittaa *Http*-protokollan mukainen *GET*-pyyntö. Avainsanalle annetaan ensimmäisenä argumenttina rajapinnan päätepiste, jonka jälkeen voidaan antaa vaihtoehtoisia asetuksia. Kuviossa 27 on peitetty ensimmäinen argumentti. Toisessa argumentissa asetetaan autentikointiavain *GET*-pyynnön otsikkotietoihin. *GET*-pyyntö palauttaa *JSON*-olion, jonka perusteella *RESTInstance*-kirjasto luo *JSON*-mallin (engl. *JSON Schema*). Mallin avulla käsitellään pyynnön palauttamaa *JSON*-oliota Robot Frameworkin testaustiedostoissa. Pyynnön jälkeen tarkistetaan *Http*-vastauskoodi viittaamalla edellä mainittuun *JSON*-malliin argumentilla *response status*. *Number*-avainsana vertaa, että numerot *response status* ja 200 ovat yhtä suuret.

```
***Variables ***
${FILEDIR}  ${CURDIR}/deviceIds.json

*** Keywords ***

GET all devices
    GET [redacted]  headers={"Authorization": "Bearer ${TOKEN}"}
    Number      response status     200
    Output      $..id       file_path=${FILEDIR}
```

Kuvio 27. Kaikkien kalustojen haku

Yksittäisen kaluston haun testaamisessa tarvitaan yhden olemassa olevan kaluston *Id*-attribuutti. Tämän vuoksi edellinen testi tallensi kaikkien kalustoiden *Id*-attribuutit

erilliseen tiedostoon, josta yksi haetaan satunnaisesti testausta varten. Kuviossa 28 on kuvattu *Get random id*-avainsana, jolle annetaan argumenttina polku tiedostoon, jossa kalustoiden *Id*-attribuutit ovat tallennettu. *JSON-olio* toimii Robot Frameworkissä merkkijonomuuttujan tavoin. Merkkijono parsitaan listaksi pilkkumerkin mukaan. Jokainen listan elementti parsitaan for-silmukan sisällä ja asetetaan uuteen listaan, josta valitaan satunnaisesti yksi arvo. Tämä arvo on avainsanan palautusarvo. Tätä palautusarvoa käytetään *GET*-pyynnön ensimmäisessä argumentissa. Yksittäisen kaluston haun testauksen toimintaperiaate on muuten samanlainen kuin kaikkien kalustoiden testaus.

```
Get random id
    [Arguments]     ${FILEPATH}
    ${fileIds}=     Get File    ${FILEPATH}
    ${unparsed}    Split String    ${fileIds}    ,
    ${idList}=      Create List
    : FOR   ${ELEMENT}  IN  @{unparsed}
    \    ${parsedValue}=   Strip String    ${ELEMENT}    characters=[]\n
    \    ${parsedValue}=   Strip String    ${parsedValue}
    \    Append To List    ${idList}    ${parsedValue}
    ${listLength}=  Get Length      ${idList}
    ${index}=       Execute Javascript  return Math.floor(Math.random() * ${listLength} - 1)
    ${deviceId}     Get From List   ${idList}     ${index}
    ${deviceId}=    Convert To Integer  ${deviceId}
    [Return]    ${deviceId}
```

Kuvio 28. Satunnaisen numeromuuttujan haku JSON-tiedostosta

Uuden kaluston lisäyksen testauksessa käytetään *POST*-avainsanaa. Avainsana tarvitsee toisena argumenttina *POST*-pyynnön sisällön. Kuviossa 29 pyynnön sisältö haetaan erillisestä tiedostosta ja asetetaan *${BODY}*-muuttujaan. Ajuri tarkistaa pyynnön lähetyksen jälkeen *HTTP*-vastauskoodin. Rajapinta palauttaa juuri luodun kaluston *Id*-attribuutin *HTTP*-vastauksen sisältönä. RESTInstance-kirjastolla ei voida suoraan asettaa vastauksen sisältöä muuttujaan, vaan tämä *Id*-attribuutti asetetaan ensin erilliseen tiedostoon, josta se haetaan myöhemmin. Ensin haetaan *GET*-pyynnöllä juuri lisätty kalusto ja tarkistetaan pyynnön onnistuminen. Sen jälkeen poistetaan kalusto *Delete*-pyynnöllä, jonka jälkeen yritetään uudestaan hakea kalusto *GET*-pyynnöllä. Pyynnön *HTTP*-vastauskoodi tulisi olla 204 eli *HTTP*-vastauksen sisältö on tyhjä.

```
POST & DELETE device
    ${BODY}=     Get File    ${CURDIR}/device.json
    POST     OUTTUJA      ${BODY}     headers={"Authorization": "Bearer ${TOKEN}"}
    Number  response status     200
    Output   response body  file_path=${CURDIR}/id.json
    ${NEWID}=   Get File    ${CURDIR}/id.json
    GET      AHAAAA/${NEWID}   headers={"Authorization": "Bearer ${TOKEN}"}
    Number      response status     200
    Delete  /ahhaaaahhaa/${NEWID}     headers={"Authorization": "Bearer ${TOKEN}"}
    Number  response status     200
    GET      /Device/${NEWID}    headers={"Authorization": "Bearer ${TOKEN}"}
    Number  response status     204
```

Kuvio 29. Kaluston lisäys ja poisto

3.7 Testien integrointi pilveen

Laitehallinta-sovelluksen versionhallinta on Microsoftin Azure DevOps-ympäristössä. Testien integraation tarkoitus on suorittaa regressiotesti aina kun ohjelman lähdekoodiin tulee muutos, jolloin tiedetään, ettei muutos rikkonut toiminnallisuuksia. Ohjelmistokoodin juurihakemistoon tulee luoda *YAML*-tiedosto nimeltä *azure-pipelines.yml*, jossa määritetään *Azure Pipelines* automatisointiputkien toiminta. Tässä työssä automatisointiputkeen määritettiin kaksi työtä, uuden testausympäristön rakentaminen ja testien ajo testausympäristössä.

Työt suorittavat automatisointiputkessa Microsoftin ylläpitämä virtuaalikone eli *agentti*. Uuden ohjelmistokoodin lisäys versionhallintaan laukaisee automatisointiputken suorituksen, jolloin Microsoftin agenttijoukosta valitaan yksi agentti työn suorittamista varten. Agenttijoukko, josta agentti valitaan, määritetään *YAML*-tiedostossa. Tähän työhön valittiin *windows-latest*-agenttijoukko, jossa agenteilla on viimeisin *Windows*-käyttöjärjestelmä sekä *Seleniumin WebDriver*-ajurit esiasennettuna.

Jos uuden testausympäristön rakennus onnistui, suoritetaan käyttöliittymätestit ympäristössä. Kuviossa 30 on kuvattu osa *YAML*-tiedostosta, jossa määritellään käyttöliittymän testaustyö. Ensimmäisenä asennetaan Robot Framework sekä SeleniumLibrary, jonka jälkeen aloitetaan testien suoritus.

```
- stage: Test
  dependsOn: Build
  condition: succeeded('Build')
  jobs:
    - job: Test
      pool:
        vmImage: 'windows-latest'
      strategy:
        matrix:
          Python37:
            python.version: '3.7'
      steps:
      - task: UsePythonVersion@0
        inputs:
          versionSpec: '$(python.version)'
        displayName: 'Use Python $(python.version)'
      - script: pip install robotframework robotframework-seleniumlibrary
      - powershell: robot -x outputxunit.xml robot/tests.robot
      - task: PublishTestResults@2
        inputs:
          testResultsFormat: 'JUnit'
          testResultsFiles: 'outputxunit.xml'
        condition: succeededOrFailed()
        displayName: 'Publish Test Results outputxunit.xml'
```

Kuvio 30. Testien suoritus automaatioputkessa

Testituloksista voidaan luoda selkolukuinen kuvio Azure DevOps-ympäristöön. Kuviossa 30 on lisätty testien suorituskomentoon *-x outputxunit.xml*, joka luo testien lokeista kopion xUnit-muodossa. *PublishTestResults@2* niminen tehtävä julkaisee testien tulokset yhteenvetosivulle. Vaikka testitulokset ovat xUnit muodossa, tulee testitulokset asettaa *JUnit*-muotoiseksi kohdassa *testResultsFormat.* Yhteenvetosivulta näkee vain testien suorittamiseen kulunut aika sekä onnistuneiden ja epäonnistuneiden testien lukumäärä. Tarkempaa tietoa testien epäonnistumisesta ei yhteenvetosivulla kerrota.

Haasteena oli saada Robot Frameworkin luoma lokitiedosto *log.html* testaajan luettavaksi testien suorituksen jälkeen. Agenttiin ei pääse käsiksi automatisointiputken töiden suorituksen jälkeen, joten lokitiedostot on siirrettävä muualle suorituksen aikana. Väliaikaiseksi ratkaisuksi päädyttiin siirtämään lokitiedostot *FTP*-yhteydellä erilliselle verkkosivulle, josta niitä pääsee

tarkastelemaan. Parempi vaihtoehto olisi lähettää lokitiedostot sähköpostilla testaajalle, mutta tätä ei saatu tähän työhön toteutettua.

4 Pohdinta

Työn tavoitteena oli toteuttaa testausautomaatioratkaisu Laitehallinta-sovelluksen käyttöliittymän sekä API-rajapinnan regressiotestaukseen. Testausautomaatioratkaisu tulisi myös integroida osaksi *Azure DevOps* ympäristöä.

Työlle asetettuihin tavoitteisiin päästiin. Saatiin toteutettua testausautomaatioratkaisu, joka suorittaa sovelluksen kriittisten ominaisuuksien regressiotestauksen käyttöliittymästä. Testitapaukset onnistuttiin suorittamaan *Azure DevOps* ympäristössä osana *Azure Pipelines* automatisointipalvelua, joten voidaan todeta testausratkaisun olevan todellisesti automatisoitu. Käyttöliittymän testausautomaatioratkaisun toteutuksen aikana havaittiin käyttöliittymätestien testaavan myös sovelluksen ohjelmointirajapintaa, joten erillistä testausratkaisua ohjelmointirajapinnalle ei toteutettu. Työssä toteutettiin kuitenkin pari esimerkkitestitapausta havainnollistamaan, miten ohjelmointirajapintaa voisi testata ilman käyttöliittymää.

Testien luotettavuudessa on vielä kehittämistä. Joissakin tilanteissa testien epäonnistuessa testausympäristöön jää testauslaitteita tietokantaan. Tämä aiheuttaa seuraavien testien virheellisiä epäonnistumisia. Ratkaisuna olisi joko tyhjentää testausympäristön tietokanta ennen jokaista testauskertaa tai kehittää luotettava tapa siivota ylimääräiset testauslaitteet tietokannasta testien jälkeen. Työn testitapaukset ovat myös liian riippuvaisia toisistaan, joka luo mahdollisesti ongelmia testitapausten lukumäärän kasvaessa suureksi. Tällöin testien suoritusaika kasvaa ja ratkaisuna olisi käyttää useaa virtuaalikonetta testien suoritukseen, mikä ei ole mahdollista tällä hetkellä.

Lähteet

Aebersold, K. N.d. Software Testing Methodologies. Smartbear sivusto. Viitattu 23.4.2020. https://smartbear.com/learn/automated-testing/software-testing-methodologies/

Bastanzhieva, D. 2019. Software testing: why is it so important? Viitattu 22.4.2020. https://medium.com/swlh/software-testing-why-is-it-so-important-175d37cd2b90

Guckenheimer, S. 2017. What is Continuos Integration?. Viitattu 11.3.2020. https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-integration

Components of a Grid, 2020. Seleniumin dokumentaatio. Viitattu 1.4.2020. https://www.selenium.dev/documentation/en/grid/components_of_a_grid/

Daityari, S. 2019. Regression Testing: A Detailed Guide. Viitattu 10.5.2020. https://www.browserstack.com/guide/regression-testing

The Good and the Bad of Katalon Studio Automation Testing Tool. 2019. Altexsoft sivusto. Viitattu 28.3.2020. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-katalon-studio-automation-testing-tool/

Introduction. N.d. Robotframework sivusto. Viitattu 23.2.2020. https://robotframework.org

Kaner, C. 2000. Architectures of Test Automation. Viitattu 10.3.2020. http://www.kaner.com/pdfs/testarch.pdf

Klemetti, M. 2013. Mitä on devops?. Blogikirjoitus Eficode-verkkosivustolla. Viitattu 10.3.2020. https://www.eficode.com/blogi/blogi/mita-on-devops

Liu, S. 2018. Testing tools used in software development worldwide in 2017. Viitattu 30.3.2020. Haettu verrko-osoitteesta https://www.statista.com/statistics/673467/worldwide-software-development-survey-testing-tools/

Lyhyesti. N.d. Nodeon Finland Oy yrityksen kotisivut. Viitattu 10.5.2020. https://www.nodeon.com/yritys/lyhyesti

McMeekin, K. 2017. Test Automation vs. Automated Testing. DevOps Zone sivusto. Viitattu 19.4.2020. https://dzone.com/articles/test-automation-vs-automated-testing-the-differenc

Molinero, A. 2019. Robot Framework and test automation. Telematiikka insinöörin blogikirjoitus. Viitattu 10.5.2020. https://www.teldat.com/blog/en/robot-framework-open-source-test-automation/

Mulders, M. 2019. Test automation vs manual testing: Picking the right balance. Viitattu 21.3.2020 https://www.testim.io/blog/test-automation-vs-manual-testing/

Robot Framework User Guide. N.d. Robot Frameworkin virallinen dokumentaatio. Viitattu 26.4.2020. https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

Sadiq, S. 2020. What is Selenium? Viitattu 31.3.2020. https://hackr.io/blog/what-is-selenium

Selenium 1 (Selenium RC). 2020. Selenium dokumentaatio. Viitattu 31.3.2020. https://www.selenium.dev/documentation/en/legacy_docs/selenium_r/

SeleniumLibrary. N.d. SeleniumLibrary dokumentaatio. Viitattu 12.4.2020. https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html

Understanding the components. 2020. Selenium WebDriver dokumentaatio. Viitattu 2.4.2020. https://www.selenium.dev/documentation/en/webdriver/understanding_the_components/

Liitteet

Liite 1. Testitapausten suunnitelma

ID	Testiskenaario	Testidata	Testi askeleet	Odotettu lopputulos
1	Testaa navigointipalkin linkkien toimivuus		1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Järjestelmät" 3. Sivulla tulisi näkyä Järjestelmät-sivu 4. Toista kohdat 2. ja 3. jokaiselle navigointipalkin linkille.	Jokainen navigointipalkin linkki vie käyttäjän tarkoitetulle sivulle
2	Lisää Järjestelmä	SITE = TestSite	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Järjestelmät" 3. Paina sivulla "Lisää järjestelmä" 4. Syötä "Omistaja" tekstikenttään "RobotFramework" 5. Syötä järjestelmän nimeksi SITE 6. Valitse järjestelmälle jokin tyyppi 7. Tallenna uusi järjestelmä	Järjestelmät sivulla tulisi olla uusi järjestelmä nimeltä SITE
3	Lisää kalusto	DEVICE = RFTestDevice01	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Kalustot" 3. Paina sivulla "Lisää kalusto" 4. Syötä kaluston nimeksi DEVICE 5. Valitse jokin kaluston tyyppi 6. Tallenna kalusto	Kalustot sivulla tulisi olla uusi kalusto nimeltä DEVICE
4	Liitä kalusto järjestelmään	SITE = TestSite DEVICE = RFTestDevice01	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Järjestelmät" 3. Valitse testattava järjestelmä 4. Paina "Liitä kalusto" painiketta 5. Valitse testattava kalusto ja paina tallenna	Järjestelmän hallintasivulla tulisi olla uusi kalusto nimeltä DEVICE
5	Testaa reitittimen hallinta	DEVICE = RFTestDevice01 ROUTER = TestRouter01	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Reitittimet" 3. Paina "Lisää uusi" 4. Lisää jokaiseen tekstikenttään ROUTER + kentän nimi 5. Tallenna reititin 6. Tarkista että sivulla on uusi reititin nimellä ROUTER 7. Muokkaa ROUTER reititintä 8. Tarkista että syötetyt tiedot ovat oikeilla kentillä 9. Mene kalustot sivulle 10. Paina "Näytä lisää" DEVICE kaluston kohdalta 11. Liitä reititin ROUTER kalustoon 12. Tarkista että reititin ROUTER ilmestyy kaluston DEVICE sivulle 13. Irroita ROUTER reititin kalustosta 14. Tarkista että reititin ROUTER ei ole enään kaluston DEVICE sivulla 15. Mene "Reitittimet" sivulle 16. Poista reititin ROUTER	Reititin ROUTER tulisi poistua järjestelmästä. Jokaisen tarkistus kohdan on mentävä lävitse.

6	Testaa kameran hallinta	DEVICE = RFTestDevice01 CAMERA = TestCamera01	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Kamerat" 3. Paina "Lisää uusi" 4. Lisää jokaiseen tekstikenttään CAMERA + kentän nimi 5. Tallenna kamera 6. Tarkista että sivulla on uusi kamera nimellä CAMERA 7. Muokkaa CAMERA nimistä kameraa 8. Tarkista että syötetyt tiedot ovat oikeilla kentillä 9. Mene kalustot sivulle 10. Paina "Näytä lisää" DEVICE kaluston kohdalta 11. Liitä CAMERA niminen kamera kalustoon 12. Tarkista että CAMERA niminen kamera ilmestyy kaluston DEVICE sivulle 13. Irroita CAMERA niminen kamera kalustosta 14. Tarkista että kamera CAMERA ei ole enään kaluston DEVICE sivulla 15. Mene "Kamerat" sivulle 16. Poista kamera CAMERA	Laite CAMERA tulisi poistua järjestelmästä. Jokaisen tarkistus kohdan on mentävä lävitse.
7	Testaa polttokennon hallinta	DEVICE = RFTestDevice01 FUELCELL = TestFuelcell01	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Polttokennot" 3. Paina "Lisää uusi" 4. Lisää jokaiseen tekstikenttään FUELCELL + kentän nimi 5. Tallenna polttokenno 6. Tarkista että sivulla on uusi polttokenno nimellä FUELCELL 7. Muokkaa polttokennoa FUELCELL 8. Tarkista että syötetyt tiedot ovat oikeilla kentillä 9. Mene kalustot sivulle 10. Paina "Näytä lisää" DEVICE kaluston kohdalta 11. Liitä polttokenno FUELCELL kalustoon 12. Tarkista että polttokenno FUELCELL ilmestyy kaluston DEVICE sivulle 13. Irroita FUELCELL polttokenno kalustosta 14. Tarkista että polttokenno FUELCELL ei ole enään kaluston DEVICE sivulla 15. Mene "Polttokennot" sivulle 16. Poista polttokenno FUELCELL	Laite FUELCELL tulisi poistua järjestelmästä. Jokaisen tarkistus kohdan on mentävä lävitse.

8	Testaa liittymän hallinta	DEVICE = RFTestDevice01 MOBILEPLAN = TestMobileplan01	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Liittymät" 3. Paina "Lisää uusi" 4. Lisää jokaiseen tekstikenttään MOBILEPLAN + kentän nimi 5. Tallenna liittymä 6. Tarkista että sivulla on uusi liittymä nimeltään MOBILEPLAN 7. Muokkaa MOBILEPLAN liittymää 8. Tarkista että syötetyt tiedot ovat oikeilla kentillä 9. Mene kalustot sivulle 10. Paina "Näytä lisää" DEVICE kaluston kohdalta 11. Liitä liittymä MOBILEPLAN kalustoon 12. Tarkista että liittymä MOBILEPLAN ilmestyy kaluston DEVICE sivulle 13. Irroita MOBILEPLAN liittymä kalustosta 14. Tarkista että liittymä MOBILEPLAN ei ole enään kaluston DEVICE sivulla 15. Mene "Liittymät" sivulle 16. Poista liittymä MOBILEPLAN	Laite MOBILEPLAN tulisi poistua järjestelmästä. Jokaisen tarkistus kohdan on mentävä lävitse.
9	Testaa VMS-opastetaulujen hallinta	DEVICE = RFTestDevice01 VMS = TestVMS01	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "VMS-opastetaulut" 3. Paina "Lisää uusi" 4. Lisää jokaiseen tekstikenttään VMS + kentän nimi 5. Tallenna VMS 6. Tarkista että sivulla on uusi VMS nimeltä VMS 7. Muokkaa VMS opastetaulua 8. Tarkista että syötetyt tiedot ovat oikeilla kentillä 9. Mene kalustot sivulle 10. Paina "Näytä lisää" DEVICE kaluston kohdalta 11. Liitä VMS opastetaulu kalustoon 12. Tarkista että VMS opastetaulu ilmestyy kaluston DEVICE sivulle 13. Irroita VMS opastetaulu kalustosta 14. Tarkista että VMS opastetaulu ei ole enään kaluston DEVICE sivulla 15. Mene "VMS-opastetaulut" sivulle 16. Poista VMS opastetaulu	Laite VMS tulisi poistua järjestelmästä. Jokaisen tarkistus kohdan on mentävä lävitse.

10	Testaa silmukkamittauslaitteen hallinta	DEVICE = RFTestDevice01 M680 = TestM68001	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Silmukkamittauslaitteet" 3. Paina "Lisää uusi" 4. Lisää jokaiseen tekstikenttään M680 + kentän nimi 5. Tallenna laite 6. Tarkista että sivulla on uusi laite nimellä M680 7. Muokkaa M680 laitetta 8. Tarkista että syötetyt tiedot ovat oikeilla kentillä 9. Mene kalustot sivulle 10. Paina "Näytä lisää" DEVICE kaluston kohdalta 11. Liitä laite M680 kalustoon 12. Tarkista että laite M680 ilmestyy kaluston DEVICE sivulle 13. Irroita M680 laite kalustosta 14. Tarkista että laite M680 ei ole enään kaluston DEVICE sivulla 15. Mene "Silmukkamittauslaitteet" sivulle 16. Poista laite M680	Laite M680 tulisi poistua järjestelmästä. Jokaisen tarkistuksen on mentävä lävitse.
11	Testaa akun hallinta	DEVICE = RFTestDevice01 BATTERY= TestBattery01	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Akut" 3. Paina "Lisää uusi" 4. Lisää jokaiseen tekstikenttään BATTERY + kentän nimi 5. Tallenna laite 6. Tarkista että sivulla on uusi laite nimellä BATTERY 7. Muokkaa BATTERY laitetta 8. Tarkista että syötetyt tiedot ovat oikeilla kentillä 9. Mene kalustot sivulle 10. Paina "Näytä lisää" DEVICE kaluston kohdalta 11. Liitä laite BATTERY kalustoon 12. Tarkista että laite BATTERY ilmestyy kaluston DEVICE sivulle 13. Irroita BATTERY laite kalustosta 14. Tarkista että laite BATTERY ei ole enään kaluston DEVICE sivulla 15. Mene "Akut" sivulle 16. Poista laite BATTERY	Laite BATTERY tulisi poistua järjestelmästä. Jokaisen tarkistus kohdan on mentävä lävitse.

12	Testaa teollisuus-PC:n hallinta	DEVICE = RFTestDevice01 PC = TestPC01	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Teollisuus-PC:t" 3. Paina "Lisää uusi" 4. Lisää jokaiseen tekstikenttään PC + kentän nimi 5. Tallenna laite 6. Tarkista että sivulla on uusi laite nimellä PC 7. Muokkaa PC laitetta 8. Tarkista että syötetyt tiedot ovat oikeilla kentillä 9. Mene kalustot sivulle 10. Paina "Näytä lisää" DEVICE kaluston kohdalta 11. Liitä laite PC kalustoon 12. Tarkista että laite PC ilmestyy kaluston DEVICE sivulle 13. Irroita PC laite kalustosta 14. Tarkista että laite PC ei ole enään kaluston DEVICE sivulla 15. Mene "Teollisuus-PC:t" sivulle 16. Poista laite PC	Laite PC tulisi poistua järjestelmästä. Jokaisen tarkistus kohdan on mentävä lävitse.
13	Testaa tutkan hallinta	DEVICE = RFTestDevice01 RADAR = TestRadar01	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Tutkat" 3. Paina "Lisää uusi" 4. Lisää jokaiseen tekstikenttään RADAR + kentän nimi 5. Tallenna laite 6. Tarkista että sivulla on uusi laite nimellä RADAR 7. Muokkaa RADAR laitetta 8. Tarkista että syötetyt tiedot ovat oikeilla kentillä 9. Mene kalustot sivulle 10. Paina "Näytä lisää" DEVICE kaluston kohdalta 11. Liitä laite RADAR kalustoon 12. Tarkista että laite RADAR ilmestyy kaluston DEVICE sivulle 13. Irroita RADAR laite kalustosta 14. Tarkista että laite RADAR ei ole enään kaluston DEVICE sivulla 15. Mene "Tutkat" sivulle 16. Poista laite RADAR	Laite RADAR tulisi poistua järjestelmästä. Jokaisen tarkistus kohdan on mentävä lävitse.
14	Irrota kalusto	DEVICE = RFTestDevice01 SITE = TestSIte	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Järjestelmät" 3. Valitse järjestelmä SITE 4. Irrota kalusto DEVICE järjestelmästä	Kaluston DEVICE tulisi poistua järjestelmän SITE sivulta
15	Poista kalusto	DEVICE = RFTestDevice01	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Kalustot" 3. Valitse kalusto DEVICE 4. Valitse kaluston valikosta kohta "Poista kalusto"	Kaluston DEVICE tulisi poistua "Kalustot" sivulta
16	Poista järjestelmä	SITE = TestSite	1. Mene pääsivulle 2. Mene navigoinnissa kohtaan "Järjestelmät" 3. Valitse järjestelmä SITE 4. Valitse järjestelmän valikosta kohta "Poista järjestelmä"	Järjestelmän SITE tulisi kadota "Järjestelmät" sivulta