Christoffer Tverin Mellavuo

Robot Framework automaattitestauksen työkaluna

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

21.05.2022

Tiivistelmä

Tekijä:	Christoffer Tverin Mellavuo
Otsikko:	Robot Framework automaattitestauksen työkaluna
Sivumäärä:	32 sivua
Aika:	21.05.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Yliopettaja Auvo Häkkinen Projektipäällikkö Lassi Heikkilä

Automaattitestauksen rooli kasvaa samalla, kun sovelluksien koko kasvaa. Sovelluksien kasvaessa tarvittavien testien määrä kasvaa, jotta sovelluksen luotettavuus säilyy. Ilman automaattitestausta tämä tarkoittaisi lisää ihmisten käsin tehtäviä testejä, jotta voidaan varmistua, että sovellus toimii oikein. Uusien testien lisäksi myös kaikkien aikaisempien testien on mentävä hyväksytysti läpi.

Automaattitestauksen tarkoitus on luoda testejä, joita on helppo toistaa aina samalla tavalla. Opinnäytetyössä esitellään Protractor- ja Robot Framework -työkalut, joiden avulla voi tehdä automaattitestausta. Opinnäytetyön painopiste on Robot Frameworkissä, koska sillä halutaan korvata olemassa olevia Protractor-testejä. Opinnäytetyössä käytetään Angular-sovellusta, johon on toteutettu Protractor-testejä.

Testaukseen liittyy myös erilaisia testaustekniikoita. Testaustekniikoilla on kaikilla oma vahvuutensa ja käyttötarkoituksensa. Opinnäytetyössä käydään läpi muutama tekniikka ja sovelletaan niitä testitapauksien yhteydessä.

Avainsanat: Automaattitestaus, Robot Framework, Protractor

Abstract

Author:	Christoffer Tverin Mellavuo
Title:	Robot Framework as Tool for Automated Testing
Number of Pages:	32 pages
Date:	21 May 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Software Engineering
Supervisors:	Auvo Häkkinen, Principal Lecturer
	Lassi Heikkilä, Project Manager

The role of automated testing increases as the software grows. When the software grows the amount of tests needed to keep the realibility of the software is increased. Without automated tests this would mean that people would need to do more tests by hand for the reliablity to stay at the same level. Not only would people need to do more tests, but the previous tests would need to be made exactly as before for them to give correct results.

Automated testing aims to create tests that are easy to repeat exaclty as before. The thesis introduces the Protractor and Robot Framework-tools enabling creating automated tests. The thesis focuses on the Robot Framework tool as it will be used to replace Protractor tests.

Testing can be done in different ways and a few of them are covered here. In the study they were also put to practice as to test cases.

Keywords: Automated testing, Robot Framework, Protractor

Sisällys

Lyhenteet

E2E *End-to-End Testing*. Testaustekniikka, jonka tavoite on luoda testi, joka simuloisi käyttäjän tekemiä asioita. Testien tarkoitus on käydä alusta loppuun kaikki vaiheet, joita käyttäjä tekisi.

HTML *Hypertext Markup Language.* HTML on kuvauskieli. Kielellä voi tehdä tekstistä rakenteellisen. Rakenteellinen tarkoittaa, että voi esimerkiksi jakaa teksti otsikkoon ja leipätekstiin. HTML:ää käytetään erityisesti verkkosivujen kirjoittamiseen.

IDE *Integrated Development Environment*. Ohjelmistoympäristö sisältää koodieditorin, kääntäjän sekä debuggerin.

NPM *Package Manager for Node JavaScript Platform*. Asennusohjelma, jolla voi asentaa ja hallita JavaScript-paketteja.

PIP *Package Installer for Python*. Asennusohjelma, jolla voi asentaa ja hallita Python-paketteja.

UI *User Interface.* Käyttöliittymä on kohta, jonka kautta käyttäjä käyttää palvelua, appia tai verkkosivua.

XPATH *XML Path Language.* XML-kuvauskielellä luotu dokumentti tai sivu ja valitsemalla siitä osa koodista, jolla voi viitata tiettyyn kohtaan. Viittaus voi olla yhteen tai moneen kohtaan. Toimii myös HTML-kuvauskielessä.

1 Johdanto

Tässä opinnäytetyössä kerrotaan automaattitestauksesta ja uudistetaan käytössä olevat testit Angular-sovelluksessa. Angular-sovellus on toteutettu TypeScript-kielellä.

Opinnäytetyössä käytetään sovellusta, jossa automaattitestaus on toteutettu Protractor-työkalulla. Opinnäytetyön tarkoitus on vaihtaa nykyistä automaatiotestausta toiseen. Projektissa haluttiin ottaa käyttöön Robot Framework, jolla korvataan Protractor-automaattitestausta. Tavoite oli saada vähintään sama testikattavuus, kuin mitä Protractorilla oli saavutettu, mutta mahdollisuuksien mukaan laajentaa kattavuutta. Työ toteutettiin osana isompaa tiimiä, joka vastasi testauksesta.

Luvussa kaksi käydään läpi yleisellä tasolla, mitä automaattitestauksella tarkoitetaan ja eri vaihtoehtoja siihen. Kolmannessa luvussa kerrotaan, miltä lähtötilanne näytti ennen testauksen päivittämistä ja tarvittavat alkuvaiheet. Luvussa neljä käydään läpi, kuinka testaus toteutettiin tässä työssä. Viidennessä luvussa käydään läpi haasteet sekä opit ja oivallukset työstä.

2 Automaattitestaus

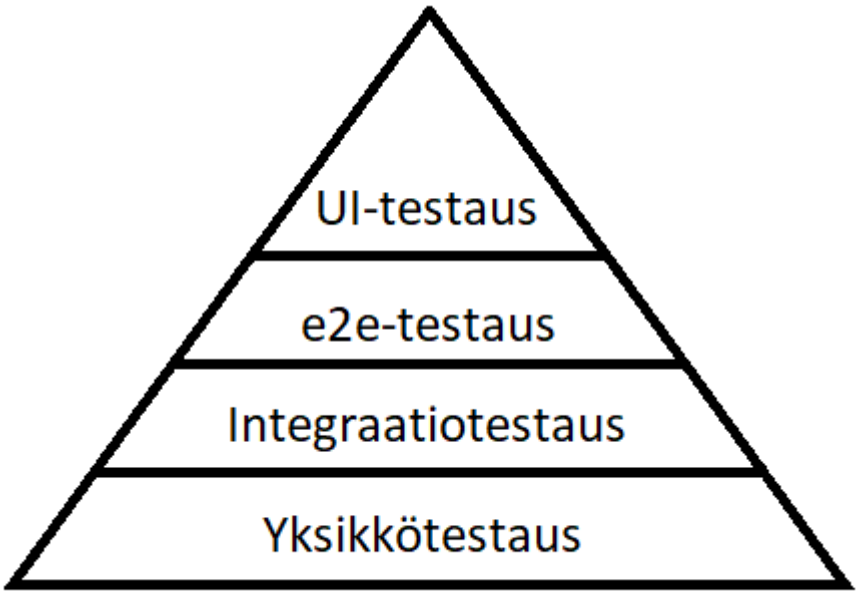
Automaattitestauksella tarkoitetaan tietokoneohjelman testausta, kun käytetään testityökalua. Automaattitestaus tapahtuu ohjelmallisesti ilman testaajan apua. Testityökalut ovat valmiiksi luotuja testiajureita, jotka on suunniteltu testausta varten. [1.] Automaattitestit käynnistyvät itsenäisesti. Testejä voidaan ajaa vuorokauden ympäri [2].

Vaihtoehtona automaattitestaukselle on manuaalitestaus. Manuaalitestaus on ohjelman testausta, jonka henkilö suorittaa käsin käyttämättä erillistä työkalua. Manuaalitestauksessa on riski ihmisen tekemään virheeseen. [3.]

Regressiotestaus, joka tarkoittaa kaikkien olemassaolevien testien toistamista, voi manuaalitestauksessa viedä paljon aikaa. Regressiotestaus tarkoittaa uudelleentestausta. Näin voidaan varmistua siitä, että sovellus toimii niin kuin ennenkin, vaikka lähdekoodi muuttuu. Koska automaattitestauksella testi suoritetaan aina samalla tavalla, kunnes testien koodia muutetaan, on testien toistaminen helppoa. [4.]

2.1 Lähestymistapoja testaukseen

Testauksessa käytetään erilaisia menetelmiä, joilla on eri tavoite ja jotka riippuvat sovelluksen rakenteesta ja sovelluksen saatavilla olevasta tiedosta. Tässä luvussa käydään läpi muutama testausmenetelmä.



Kuva 1. Testauspyramidi.

Yksikkötestauksessa luodaan testejä, jotka testaavat mahdollisimman pienen osa-alueen sovelluksesta kerralla. Yksikkötesteissä eristetään testattava koodi muusta koodista, jolloin voi varmistua, että se toimii oikein. Yksikkötestaus on testauspyramidissa alimmaisena, koska se testaa koodia mahdollisimman tar-

kasti ilman ulkoisia tekijöitä. Yksikkötestauksen määrä on yleensä muita testausmenetelmiä enemmän, koska yksikkötestauksella pyritään testaamaan mahdollisimman kattavasti lähdekoodia ja jokainen yksikkötesti testaa vain pienen osan siitä.

Yksikkötestaus voidaan luokitella White Box -testaukseksi, koska se perustuu lähdekoodin testaukseen. [5.]

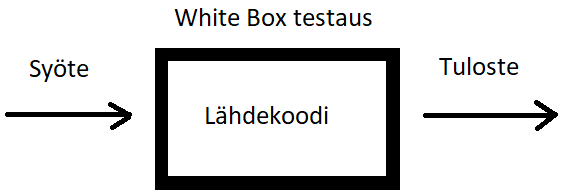
Automaatiopyramidissa seuraavana testausmenetelmänä yksikkötestauksen jälkeen on integraatiotestaus. Integraatiotestauksessa testataan, toimiiko sovelluksen lähdekoodiosat yhdessä halutulla tavalla. Testeissä testataan myös, että rajapinnat ja tietokanta toimivat oikein, jos sellaisia on käytössä. [6.]

E2e-testauksessa luodaan testikokonaisuuksia, joilla simuloidaan, miten loppukäyttäjä käyttäisi sovellusta. E2e-testauksessa huomioidaan myös rajapinnat ja tietokanta, joita loppukäyttäjät käyttävät. [7.]

UI-testauksessa eli käyttöliittymän testauksessa huomioidaan neljä asiaa: käytettävyys, saavutettavuus, johdonmukaisuus ja yhteensopivuus. UI-testauksessa huomioidaan ainoastaan sovelluksen toimivuutta käyttäjien näkökulmasta. Testeillä pyritään vastaamaan kahteen kysymykseen: Onko kaikki termit ja toiminnallisuus helppo ymmärtää sovelluksessa? Voiko sovellusta käyttää ilman, että se lakkaa toimimasta tai ilmeneekö virheitä? [8.] Opinnäytetyössä keskitytään e2e-testaukseen ja UI-testaukseen. Nämä molemmat toteutetaan Black Box -testauksella.

2.1.1 White Box -testaus

White Box -testaus on tekniikka, joka voidaan jakaa kahteen osaan. Ensimmäinen vaihe on, että testien tekijän on ymmärrettävä sovelluksen lähdekoodia. Seuraava vaihe on kirjoittaa testejä, jotka testaavat lähdekoodia. [9.]

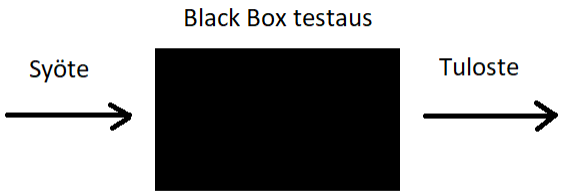


Kuva 2. White Box -testaus.

White Box -testauksessa lähdekoodi on saatavilla. Testit tarkistavat, että syöte ja tuloste vastaavat oletusta huomioiden myös lähdekoodissa olevat eri vaiheet. [9.]

2.1.2 Black Box -testaus

Black Box -testaus on tekniikka, jossa testataan sovellusta ilman, että tiedetään täsmällisesti, miten lähdekoodi toimii. Vain ulospäin näkyvä toiminnallisuus on tiedossa. Testit tehdään niin, että syötetään tietyt arvot ja oletetaan joku arvo tulokseksi. [10.]



Kuva 3. Black Box -testaus.

Tulosta, jonka sovellus palauttaa, verrataan oletettuun arvoon ja sen perusteella testi joko onnistuu tai epäonnistuu [10].

2.2 Protractor

Protractor on e2e-testityökalu, jolla voi testata Angular-sovelluksia sekä AngularJS-sovelluksia [11]. Angular-sovellukset kirjoitetaan TypeScript-kielellä, kun

taas AngularJS-sovellukset kirjoitetaan JavaScript-kielellä. TypeScript on JavaScript, jonka päälle on lisätty uusia ominaisuuksia. TypeScript muunnetaan JavaScriptiksi koodin kääntämisvaiheessa, koska web-selain tulkitsee vain JavaScriptiä. TypeScriptissä toimivat samat kirjastot kuin JavaScriptissä [12]. TypeScript on vahvasti tyypitetty ja JavaScript löyhästi tyypitetty [13]. Koska TypeScript on vahvasti tyypitetty, ilmenevät tyypitysvirheet ohjelmistoympäristössä (IDE). Saman virheen tunnistaminen JavaScriptissä vaatisi kääntämisen. Protractoria voi käyttää käyttäytymislähtöiseen kehityksen testaukseen. Käyttäytymislähtöinen testaus keskittyy tekemään testejä, jotka tarkistavat, että järjestelmä toimii halutulla tavalla. [14; 15.]

Protractor ajaa testejä selaimessa. Protractor on helppo asentaa Node package managerin kautta (NPM). Protractor-testit kirjoitetaan JavaScript- tai TypeScript-kielellä. Protractor toimii ainoastaan JavaScript- tai TypeScript-pohjautuvien sovellusten kanssa. [11.]

Kuvassa 4 on esimerkki Protractorille JavaScriptillä kirjoitetusta koodista. Funktio rajataan aaltosuluilla, jonka sisällä on lisätty haluttu toiminnallisuus. Simuloidessa hiirellä tehty valinta toteutetaan kertomalla ensin mikä sivun elementti halutaan valita ja sen jälkeen kutsutaan click()-funktiota. Funktio vastaa hiiren painikkeen napsautusta käyttöliittymässä. Kun halutaan verrata esimerkiksi saatua arvoa oletettuun, kutsutaan funktiota toEqual() ja sulkujen sisälle sijoitetaan yksi arvoista ja ennen toEqual()-muuttujaa, joka sisältää toisen arvon. [11.]

```
describe('angularjs homepage todo list', function() {
  it('should add a todo', function() {
    browser.get('https://angularjs.org');

    element(by.model('todoList.todoText')).sendKeys('write first protractor test');
    element(by.css('[value="add"]')).click();

    var todoList = element.all(by.repeater('todo in todoList.todos'));
    expect(todoList.count()).toEqual(3);
    expect(todoList.get(2).getText()).toEqual('write first protractor test');

    // You wrote your first test, cross it off the list
    todoList.get(2).element(by.css('input')).click();
    var completedAmount = element.all(by.css('.done-true'));
    expect(completedAmount.count()).toEqual(2);
  });
});
```

Kuva 4. Protractor-koodiesimerkki.

Expect()-funktiolla määritellään, mitä odotetaan toiminnon tulokseksi. Se vertaa suorituksesta saatua tulosta testin tulokseksi haluttuun arvoon.

2.3 Robot Framework

Robot Framework on työkalu, jolla voi automatisoida testejä. Robot Framework soveltuu hyväksymistestaukseen [16]. Hyväksymistestaus eroaa Protractorin käyttäytymislähtöisestä testauksesta sillä, että testeillä pyritään tarkistamaan, että sovellus toimii oikein tilanteissa, jotka ovat samanlaisia kuin käyttäjien käyttötavat. Käyttäytymislähtöinen testaus keskittyy enemmän tarkistamaan järjestelmän eri vaiheet eikä käyttäjän eri vaiheet. [15.]

Robot Framework on ilmainen, koska se julkaistaan avoimella lähdekoodilla. Robot Frameworkin ominaisuuksia voi laajentaa erilaisilla Python- ja Java-kirjastopakkauksilla. Robot Frameworkin -syntaksi muistuttaa Python-kieltä. [17.]

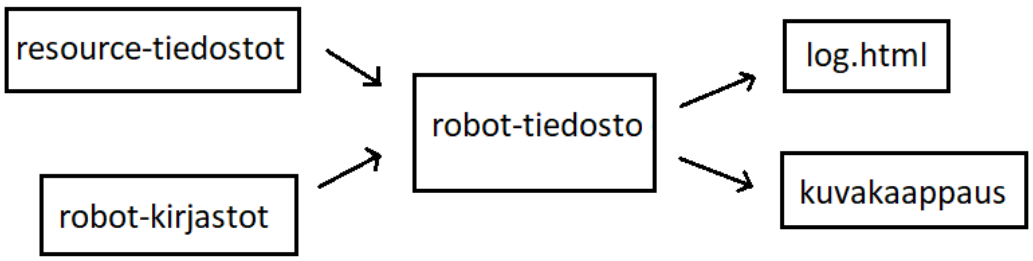
Robot Framework -syntaksi on helppo lukea, koska se käyttää avainsanoja, jotka kuvaavat, mitä on tarkoitus tehdä. Esimerkki selkeästä syntaksista on sisäänkirjautuminen, joka tehdään Set Login Name -syntaksilla, jonka jatkona on

haluttu kirjautumistunnus. [17.] Avainsanat on sama asia kuin muissa ohjelmointikielissä metodit ja funktiot.

Python, johon Robot Frameworkn pohjautuu, ja Protractorin käyttämä JavaScript eroavat kirjoitusasultaan. Python käyttää sisennystä, jolla koodia rajataan eri osiin, kun taas JavaScript käyttää aaltosulkuja.

Robot Frameworkissä testit kirjoitetaan tekstitiedostoihin, joiden tiedostopääte on robot. Tekstitiedostoihin, joiden tiedostopääte on resource, voi kirjoittaa avainsanoja ja parametreja, joita voi kutsua robot-tiedostoista. Samaa Resource-tiedostoa voi hyödyntää monesta eri robot-tiedostosta eikä näin ollen tarvitse kuin kerran kirjoittaa Resource-tiedoston sisältö. Kun testit on ajettu, syntyy html-tiedostoja, joissa on luettavissa, miten testit onnistuivat. Testeihin voi lisätä avainsanoja, joilla otetaan selaimesta kuvakaappaus ja tallennetaan se. Kun testit ovat ajettu valmiiksi, myös tallennetut kuvakaappaukset ovat mukana html-tiedostossa. [17.]

Robot-tiedoston sisältämiä Test Case -testit eivät ole mahdollista kutsua muissa Test Case -testeissä. Robot-tiedostossa sisältämät avainsanat on mahdollista vain kutsua sen tiedoston sisällä.



Kuva 5. Robot Framework -tiedosto.

Kuvan 6 Robot Framework -esimerkin alussa on Settings-osio, jossa Resource keywords.resource -rivi ottaa käyttöön sen nimisen tekstitiedoston sisältämät määrittelyt [18].

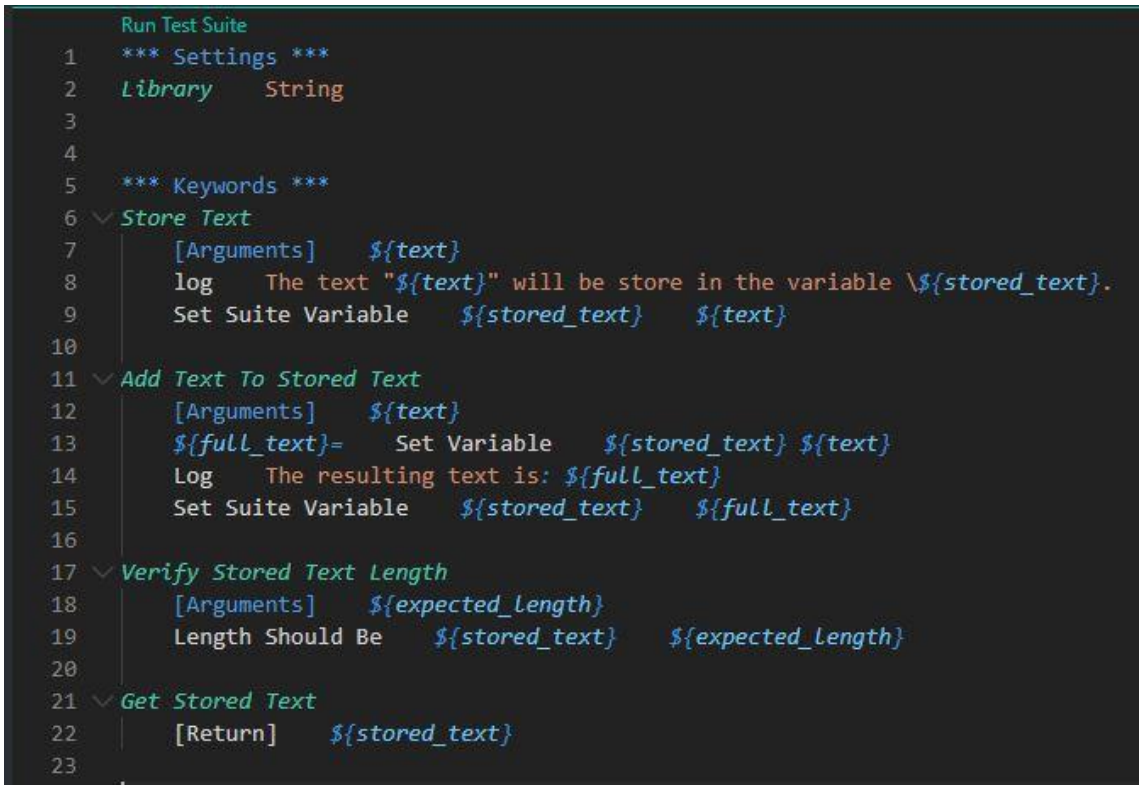
```
Run Test Suite
*** Settings ***
Documentation     This .robot file is a suite
...
...               Keywords are imported from the resource file
Resource          keywords.resource

*** Test Cases ***
Run Test
Simple Test Case
    [Documentation]    Shows some assertion keywords
    Should Be Title Case    Robot Framework
    Should Be Title Case    roboT frameWork
    Should Be Equal    Text123    Text123
    Should Be True    5 + 5 == 10

Run Test
Test with Keywords
    Store Text    Hail Our Robot
    Add Text To Stored Text     Overlords!
    Verify Stored Text Length    25
    ${current_text}=    Get Stored Text
    Should Be Equal    ${current_text}    Hail Our Robot Overlords!
```

Kuva 6. Robot Framework -testiesimerkki.

Keywords.resource-tekstitiedoston sisältö näkyy kuvassa 7 [18].



Kuva 7. Tekstitiedosto, joka sisältää avainsanoja.

Testit (kuva 6) on jaettu niin, että haluttu toiminnallisuus on sisennetty ilman aaltosulkuja testin nimen alapuolelle. Kaikki rivit, jotka on sisennetty Simple Test Case -rivin jälkeen, suoritetaan yhdessä ja muodostavat yhden testin. Kaikki Test with Keywords -rivin alla olevat sisennetyt rivit suoritetaan yhdessä ja muodostavat toisen testin.

Robot Frameworkissä erilliset elementit erotetaan toisistaan kahdella tai useammalla välilyönnillä. Simple Test Case -testissä on ensin [Documentation] Shows some assertion keywords, joka ainoastaan kuvaa testiä. Should Be Title Case -avainsana tarkistaa, onko Robot Framework -lauseen molemmissa sanoissa ainakin yksi iso kirjain. Robot Framework -otsikon molemmissa sanoissa tulee siis olla iso kirjain. Seuraavaksi tarkistetaan Should Be Title Case -avainsanalla, että hyväksytäänkö myös roboT frameWork. Koska roboT-sanan viimeinen kirjain on iso ja frameWork-sanan kuudes kirjain on iso, niin tämä testi suoritetaan myös onnistuneesti. Testitulokset ovat luettavissa kuvasta 8.

Avainsana Should Be Title Case ei ole Robot Frameworkin oletuksen tarjoama avainsana, vaan se tulee String-kirjastosta. String-kirjasto on otettu käyttöön kuvan 5 resource-tiedostossa. Should Be Equal -avainsanalla voi verrata kahta objektia. Esimerkissä verrataan kaksi samaa lausetta Text123, joka on tosi. Robot Frameworkissä voi myös laskea ja tarkistaa, onko laskutoimitus oikein. [17.]

Kuvan 6 Test with Keywords -testin toiminnallisuus alkaa rivillä Store Text Hail Our Robot. Store Text -teksti ei ole oletusavainsana. Kuvassa 7 luodaan omia avainsanoja, joista yksi on Store Text. Se ottaa vastaan yhden parametrin ja tallentaa sen ${stored_text}-muuttujaan. Set Suite Variable -avainsana mahdollistaa sen, että muuttujaa voi käyttää myös muissa testeissä. Add Text To Stored Text -avainsana lisää Overloards!-tekstin ${stored_text}-muuttujaan. Verify Stored Text Length 25 tarkistaa, onko tallennetussa lauseessa 25 merkkiä. Get Stored Text palauttaa ${stored_text}-muuttujaan tallennetun arvon. Viimeiseksi tarkistetaan, että ${current_text} sisältää tekstin Hail Our Robot Overlords!

Robot Frameworkissä koodi jaetaan osiin ja jokaisella osiolla on oma tietty tarkoitus. Settings-osiossa on kerrottu, mitä kirjastoja käytetään, mitä liitetiedostoja käytetään ja mitä tehdään ennen ja jälkeen testien. [18.]

Variables-osioon tallennetaan arvot, jotka eivät muutu vaan pysyvät aina samoina. Samoja arvoja voi hyödyntää kaikissa testeissä ja samalla vahvistua siitä, että testit käyttävät samaa arvoa.

Test Cases -osioon kirjoitetaan testit. Nämä suoritetaan, kun ajetaan testitiedostoja.

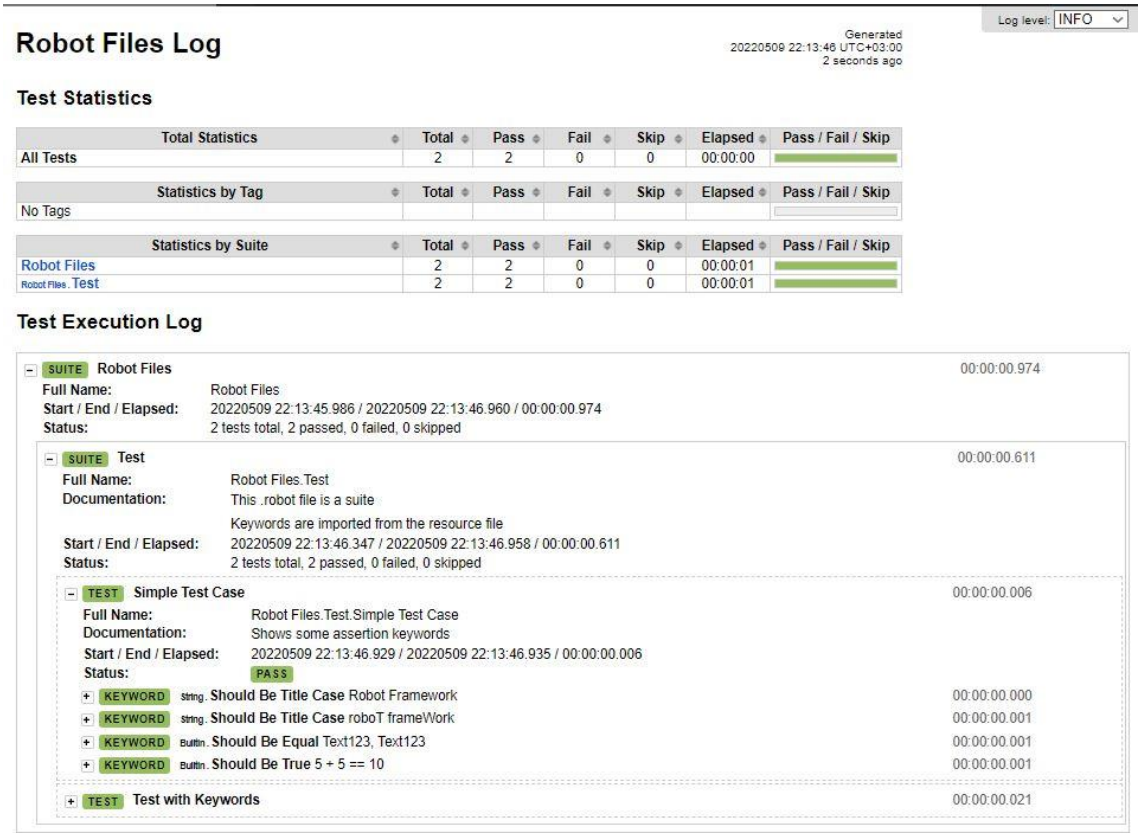
Tasks-osio toimii samalla tavalla kuin testitapaukset. Sinne kirjoitetaan halutut testit. Robot Framework tarjoaa kaksi sanavaihtoehtoa testiosion tekemiseen joko Tasks tai Test Cases. Samassa testitiedostossa voi käyttää vain joko Test Cases -osiota tai Tasks-osiota.

Comments-osioon voi kirjoittaa lisätekstiä, joka helpottaa ihmistä ymmärtämään, mitä testissä halutaan tehdä. Kommentit ovat vapaamuotoista tekstiä, eivätkä ne vaikuta testien toiminnallisuuteen.

Taulukko 1. Robot Frameowork rakenne.

Robot Frameworkin osiot	**Suomennos**
Settings	Asetukset
Variables	Muuttujat
Test Cases	Testi tapaukset
Tasks	Tehtävät
Keywords	Avainsanat
Comments	Kommentit

Robot Framework luo aina testiajojen jälkeen testitulokset html-tiedostoon. Aikaisemmin esitelty koodiesimerkki kuvissa 5 ja 6 tuottaa kuvassa 9 näkyvän log.html-tiedoston.



Robot Files Log

Log level: INFO

Generated
20220509 22:13:46 UTC+03:00
2 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	2	2	0	0	00:00:00	

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

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Robot Files	2	2	0	0	00:00:01	
Robot Files. Test	2	2	0	0	00:00:01	

Test Execution Log

SUITE Robot Files 00:00:00.974
Full Name: Robot Files
Start / End / Elapsed: 20220509 22:13:45.986 / 20220509 22:13:46.960 / 00:00:00.974
Status: 2 tests total, 2 passed, 0 failed, 0 skipped

SUITE Test 00:00:00.611
Full Name: Robot Files.Test
Documentation: This .robot file is a suite
Keywords are imported from the resource file
Start / End / Elapsed: 20220509 22:13:46.347 / 20220509 22:13:46.958 / 00:00:00.611
Status: 2 tests total, 2 passed, 0 failed, 0 skipped

TEST Simple Test Case 00:00:00.006
Full Name: Robot Files.Test.Simple Test Case
Documentation: Shows some assertion keywords
Start / End / Elapsed: 20220509 22:13:46.929 / 20220509 22:13:46.935 / 00:00:00.006
Status: PASS

KEYWORD String. Should Be Title Case Robot Framework 00:00:00.000
KEYWORD String. Should Be Title Case roboT frameWork 00:00:00.001
KEYWORD BuiltIn. Should Be Equal Text123, Text123 00:00:00.001
KEYWORD BuiltIn. Should Be True 5 + 5 == 10 00:00:00.001

TEST Test with Keywords 00:00:00.021

Kuva 8. Robot Frameworkin testitulokset log.html-tiedostona.

Testituloksesta nähdään, että testejä oli yhteensä kaksi kappaletta ja molemmat on ajettu onnistuneesti. Simple Test Case -testi on laajennettu ja nähdään, että testi muodostui neljästä pienemmästä testistä Should Be Title Case, Should Be Title Case, Should Be Equal ja Should Be True. Log.html-sivustolta nähdään myös, miten kauan kukin testi on kestänyt. [17.]

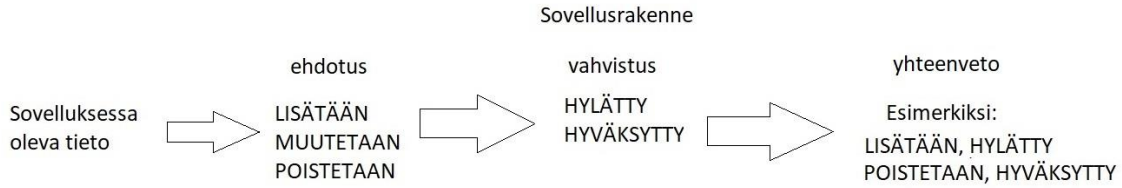
2.4 Muita testaustyökaluja

Cypress on työkalu, jolla voi testata JavaScript-sovelluksia. Alussa Cypressä oli vain tuki Chrome-selaimelle, mutta vuonna 2020 julkaistussa Cypress 4.0 on tuki myös Edge- ja Firefox-selaimille [19]. Cypress ei tue Internet Explorer- tai Safari-selainta. Cypress eroaa Protractorista ja Robot Frameworkista, koska se ei käytä WebDriver-ajuria vaan Cypressin omaa ajuripakkausta. Cypress-testit kirjoitetaan joko JavaScript-kielellä tai TypeScript-kielellä. [20.] Cypress oli

myös projektissa harkinnan alla ennen kuin päädyttiin käyttämään Robot Frameworkiä.

3 Projektin alkutilanne

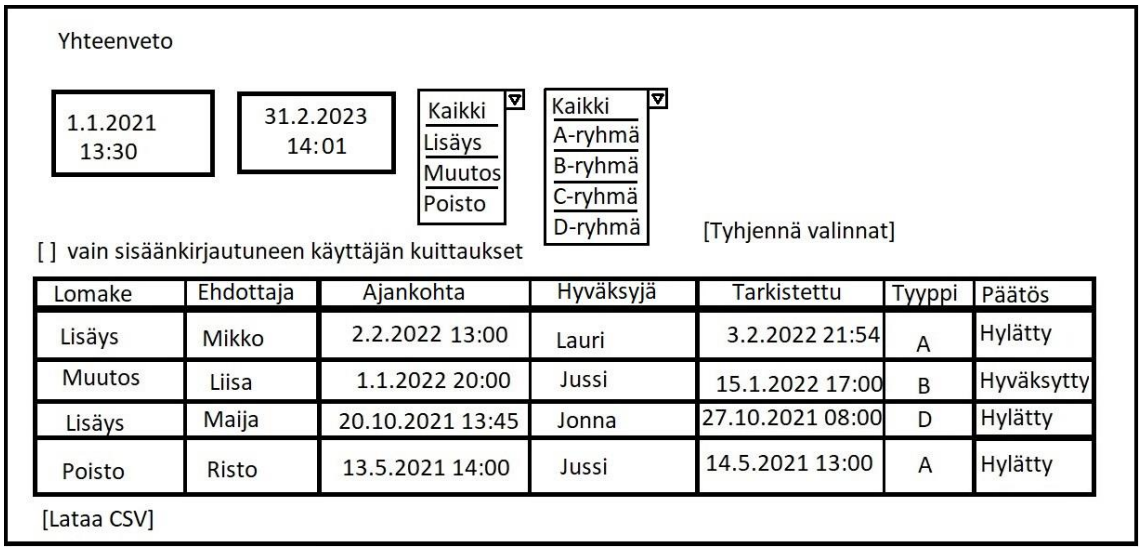
Projektissa käytetty sovellus toimi niin, että käyttäjä pystyi katsomaan saatavilla olevaa tietoa. Käyttäjä pystyi ehdottamaan tietoa, jonka haluaisi lisätä kantaan. Käyttäjä pystyi ehdottamaan, että olemassa olevalle datalle tehdään muutoksia. Käyttäjä pystyi myös ehdottamaan, että jokin tieto poistetaan käytöstä.



Kuva 9. Sovelluksen eri vaiheet.

Kun ehdotus oli lähetetty, ylläpitäjät tarkistivat sen. Ylläpitäjät joko hyväksyivät tai hylkäsivät ehdotukset. Ylläpitäjillä oli myös mahdollisuus muokata ehdotusta, jos oli tarvetta. Ylläpitäjien vahvistettua ehdotuksen, ja kun ehdotus oli luokiteltu hyväksytyksi tai hylätyksi, siitä muodostui myös yhteenveto, ja se lisättiin yhteenvetosivulle. Tässä opinnäytetyössä päivitettiin yhteenvetosivulla olevat automaatiotestit.

Sovelluksen yhteenvetosivu muodostui kahdesta kalenterikomponentista, joilla pystyi valitsemaan vuoden, kuukauden, päivän sekä tunnin ja minuutit. Kuvassa 11 on hahmotelma sivusta. Vasemmalla oleva kalenteri kertoi, mistä lähtien haetaan tietoa ja oikeanpuoleinen kalenteri rajasi, mihin asti tietoa haetaan. Sivulta löytyi myös kaksi pudotusvalikkoa, joiden oletusarvo oli Kaikki-arvo. Ensimmäinen pudotusvalikko rajasi, mitkä lomakkeet näytetään. Toinen pudotusvalikko rajasi, minkä tyyppiset lomakkeet näytetään. Sivulla oli myös valintaruutu, jonka avulla pystyi rajamaan näytettävää tietoa vain nykyisen käyttäjän tarkistettujen ehdotusten mukaan, tai sitten kaikki ehdotukset, jotka olivat tarkistettu annetulla aikavälillä.



Kuva 10. Yhteenvetosivun hahmotelma.

Taulukossa olevan tiedon voi ladata myös csv-tiedostona. Sivulla löytyi Tyhjennä valinnat -painike, jolla pystyi tyhjentämään kokonaan kalenterissa valitut päivämäärät ja kellonajat sekä palauttamaan pudotusvalikoiden oletusarvot. Oletuksena taulukko oli piilotettuna, ja vasta kun aloitus- ja lopetuspäivämäärä ja aika oli asetettu, haettiin tieto. Heti kun tarvittavat kentät oli täytetty, käynnistyi tietokantahaku, joka palautti kaikki hakuehdoilla löydetyt ehdotukset haun valmistuttua. Samaan aikaan, kuin haku lähetettiin sovelluksen Backend-osiolle ja siitä eteenpäin tietokantaan sovelluksessa, taulukko tuli käyttöliittymässä esiin ja vain otsikot näytettiin. Taulukkoon päivitettiin hakutulokset, kun ne olivat valmistuneet. Kun tuloksia ei löytynyt, esitettiin taulukko, jossa vain otsikot näkyivät.

3.1 Tarvittavat järjestelmäasetukset

Käytössä oli Chrome-selain, joten Protractoria varten oli asennettava ChromeDriver-ajurit, jotka mahdollistivat testien ajon selaimessa. Sovellus oli tehty TypeScriptillä ja käytti npm-pakkaushallinta, joten sovelluksen Frontend-osio oli helppo pystyttää käyttäen Node JavaScript -kirjastoja. Backend-osiossa oli käytössä Apache Maven -pakkaushallinta. Tietokanta oli SQL-pohjainen. Koko sovellus oli mahdollista asentaa paikallisesti. Paikallisella sovelluksella muutokset

ja testit oli mahdollista aluksi testata, ennen kuin ne etenivät muiden käytettäviksi. Protractor asentui myös npm-pakkaushallinnan kautta. Protractorin asennusvaiheessa oli myös asennettava ChromeDriver-ajuri, jonka tuli vastata selaimen versiota. Koska asennuksessa käytettiin npm-pakkaustenhallintaa, asennettiin myös ChromeDriver sen hakemistopolkuun. Projektissa oli käytössä JetBrains IDEA ohjelmointiympäristö (IDE). Se sisältää koodieditorin, kääntäjän sekä debuggerin. Käytetyssä IDEssä oli tuki Protractorin lähdekoodille, joka on JavaScript ja TypeScript, sekä Robot Frameworkin Python-tyyliselle lähdekoodille.

Opinnäytetyössä käytetyt testityökalut pohjautuvat kaikki samaan WebDriver-ajuripakkaukseen. WebDriver on avoimeen lähdekoodiin perustuva ajuripakkaus, jolla pystyy hallitsemaan selainta ja ajamaan testejä selaimessa [21]. WebDriverilla voi hallita selaimen sivustoja toimimalla manuaalitestaajan tilalla. WebDriver voi avata selaimen ja antaa sille testattavan sovelluksen URL-osoitteen. Se voi tarkistaa sivuilta löytyvää tekstiä sekä napsauttamaan eri sivun kohtia samalla tavalla kuin käyttäjä tekisi hiirellä. WebDriver pystyy myös syöttämään tietoa syötekenttiin.

WebDriveristä pitää valita selainkohtaisesti oikea versio. Tämä tarkoittaa, että Firefox, Chrome, Edge ja Safarille on jokaiselle oma pakkaus. Esimerkiksi Chromelle tehty pakkaus löytyy nimellä ChromeDriver ja Firefoxille nimeltä GeckoDriver. Pakkausta valittaessa pitää myös valita sama versio kuin selaimessa. Eri selainversioissa voi olla eroavaisuuksia siinä, miltä selain näyttää ja miten se toimii.

3.2 Testikattavuus alkuvaiheessa

Aikaisemmin tehdyt testit, jotka haluttiin korvata, oli kirjoitettu TypeScriptillä Protractorille. Olemassa olevat testit testasivat alusta loppuun muutamalla eri tavalla mahdolliset tapaukset, joita ylläpitäjät voisivat tehdä yhteenvetosivulla. Kaikki testit alkoivat kirjautumalla ylläpitäjänä sivulle. Tämän jälkeen luotiin testi-

dataa, joka tallennettiin tauluun. Sama testidata oli käytettävissä jokaisessa testissä. Testit suodattivat ja järjestelivät taulukon arvot, jonka jälkeen testitapaukset tarkistivat, että taulukon arvot olivat halutulla tavalla. Oli myös luotu testi, joka latasi csv-tiedoston ja tarkisti sen jälkeen tiedoston sisältö. Testien lopussa kaikki testauksessa käytetty data poistettiin.

4 Automaattitestauksen toteutus Robot Frameworkilla

Kun Protractor-testit olivat tulleet tutuksi, oli aika suunnitella Robot Frameworkille vastaavat testit. Testaaminen aloitettiin UI-testauksella ja saada kaikki sivun elementit testattua ja sen jälkeen siirtyä e2e-testaukseen.

4.1 Tarvittavat järjestelmäasetukset

Ensimmäisessä vaiheessa asennettiin koneelle Python 3.9 ja Robot Framework sekä kaikki sen vaatimat kirjastot. Pythonin mukana tulee PIP-asennusohjelma, jolla voi helposti asentaa Robot Frameworkin. PIP myös tunnistaa, mitkä muut kirjastot tarvitaan. Oletuksena PIP asentaa uusimman version, mutta tarvittaessa se voi asentaa myös vanhemman version Robot Frameworkistä. [18.] Asennusvaiheessa oli myös asennettava ChromeDriver-ajurit Pythonin asennuskansioon, josta Robot Framework käyttää sitä. Projektin aikana, kun selain päivittyi, oli myös manuaalisesti korvattava ChromeDriver-ajurit uudemmalla.

Robot Frameworkin oletuskirjaston lisäksi käytettiin QWeb-kirjastoa. QWeb valittiin, koska se oli otettu käyttöön muualla sovelluksessa ja voitiin hyödyntää aikaisemmin tehtyjä avainsanoja sekä lisätä uusia avainsanoja laajasti käytettäväksi. QWeb oli myös saatavilla PIP-asennusohjelman kautta. QWeb asentui komennolla pip install QWeb. Kirjastojen käyttöönotto vaati tiedoston alussa Settings-osiossa rivin Library QWeb (esimerkkikoodi 1). [22.]

```
*** Settings ***
Library    QWeb

*** Test Cases ***
Name of test
    Keywords
```

Esimerkkikoodi 1. QWeb-kirjaston käyttö

SeleniumLibrary-kirjasto on toinen suosittu Robot Framework -kirjasto, jolla voi testata selaimessa toimivaa sovellusta. SeleniumLibrary-kirjaston ja QWeb-kirjaston avainsanoja ei voi käyttää yhdessä saman selainsession aikana, koska ne ovat erillisiä kirjastoja ja vain yksi kirjastoista on luettavissa selainsession aikana. [23.]

4.2 Verkkosivun UI-testaus

Ensimmäinen vaihe oli luoda metodi, jolla voi avata sivun selaimessa. Kun sivu oli auennut selaimeen, tarvittiin metodi, jolla kirjauduttiin sovelluksen sisään. Seuraavaksi luotiin metodi, joka navigoi sivustolla halutulle sivulle, johon testit oli tarkoitus toteuttaa. Nämä metodit suoritettiin aina aloittaessa, joten niitä oli järkevä toteuttaa erillisinä avainsanoina.

Ensimmäinen testikokonaisuus tarkisti, että sovelluksen sivun osoite toimi ja sivulta löytyi haluttu otsikko. Testi kutsui ensin metodia, joka avasi sovelluksen selaimeen ja sen jälkeen metodia, joka kirjautui sovellukseen. Onnistuneen kirjautumisen jälkeen valittiin testiä varten haluttu sivu. Koska jokaisessa testissä oli avattava yhteenvetosivu, niin Resource-tekstitiedostoon lisättiin yhteenvetosivun avaamista varten tehdyn avainsanan. Esimerkkikoodissa 2 Open Yhteenvetosivu -avainsanalla voidaan napsauttaa ensin Menu-teksti ja sen jälkeen napsauttaa Yhteenveto-teksti.

```
*** Settings ***
Documentation    Keywords for navigating

*** Keywords ***
Open Yhteenvetosivu
    Click Text    Menu
    Click Text    Yhteenveto
```

Esimerkkikoodi 2. Keywords.resource-tiedosto, joka sisältää Open Yhteenvetosivu -avainsanan.

Kun sivu oli latautunut, siirryttiin viimeiseen testivaiheeseen, joka tarkisti, että oikea otsikko löytyi sivulta. Tämän jälkeen selain suljettiin. Tällä testillä pystyi tarkistamaan nopeasti, että sivu oli olemassa ja ainakin otsikon perusteella oikea sivu oli auennut. Esimerkkikoodi 3 kertoo, miten tarkistus tehtiin. Robot Frameworkissä voi etsiä, löytyykö haluttu teksti sivulta. Esimerkkikoodissa numero 3, Verify Text -avainsana etsi sivulta tekstin Yhteenveto [24].

```
*** Settings ***
Library    QWeb

*** Test Case ***
Test That Text is Found
     Open Browser    http://localhost:4000     chrome
     Open Yhteenvetosivu
     Verify Text    Yhteenveto
```

Esimerkkikoodi 3. Tekstin tarkistaminen.

Toinen testi oli jatkoa ensimmäiselle testille. Testillä käytiin läpi, että kaikki sivulla esitetystä tekstistä vastasi oletusta. Testissä huomioitiin myös pudotusvalikoiden sisältämä teksti. Pudotusvalikot tunnistettin xpath-polulla.

Verify Text -avainsana löytyy QWeb-kirjastossa. Myös Open Browser on QWeb-kirjaston avainsana, jolla Chrome-selain avataan osoitteeseen http://localhost:4000 esimerkkikoodissa 3. [24.]

Testejä on helppo tehdä sivulta löytyvän tekstin mukaan, jos teksti on saatavilla sivulla. Kun sivun rakenne monimutkaistuu, on hyvä opetella Robot Frameworkin tukema xpath. Xpath on syntaksi, jolla voi paikantaa sivun elementtejä. HTML-kuvauskieli pohjautuu XML-kuvauskieleen ja siksi myös HTML-sivuilla

voi paikantaa elementtejä xpath-polulla [25]. Esimerkkikoodi 4 esittää xpath-polkujen käyttöä.

```
*** Settings ***
Library    QWeb

*** Test Case ***
Test That Elements Can be Clicked On
     Open Browser    http://localhost:4000     chrome
     Open Yhteenvetosivu
     Click Element    xpath\=//button[@class=”middle”] 1
     ClickElement    //*[@id=”small-button”]
```

Esimerkkikoodi 4. Xpath-polku kahdelle sivulla olevalle elementille.

ClickElement-avainsana kuuluu QWeb-kirjaston tarjoamiin avainsanoihin. ClickElement-avainsanan voi kirjoittaa joko kaikki yhteen ClickElement tai välilyönnillä Click Element. ClickElementille annetaan myös, mistä napsautettava elementti löytyy. Ensimmäisessä polkuesimerkissä xpath\=//button[@class=”middle”] xpath on kirjoitettu ensin. Seuraavaksi tarkennetaan, että kyse on button-elementti ja viimeiseksi class nimeltä middle. Viimeiseksi oleva numero 1 tarkoittaa, että napsautetaan ensimmäistä painiketta, jolla on middle-luokka määriteltynä. [24.]

Seuraava ClickElement toimii samalla tavalla, vaikka xpath-alkuliite on jätetty pois. Tähti-merkillä haetaan kaikki elementit, jotka sopivat xpath-polkuun. Id:llä small-button rajataan polku vain sen id:n mukaan. [24.]

Toisessa testissä käytettiin QWebistä löytyvää Verify Element Text -avainsanaa, jolla saatiin pudotusvalikot testattua. Avainsanalle annettiin elementin polku ja teksti, jotka oletettiin löytyvän. Lause toistettiin kaikille arvoille. [24.]

```
*** Settings ***
Library    QWeb

*** Test Case ***
Test That Element Has Right Value
    Open Browser    http://localhost:4000     chrome
    Open Yhteenvetosivu
    Verify Element Text //select[@id\=”dropdown-menu”] Kaikki
    Verify Element Text //select[@id\=”dropdown-menu”] Muutos
    Verify Element Text //select[@id\=”dropdown-menu”] Lisäys
    Verify Element Text //select[@id\=”dropdown-menu”] Poisto
```

Esimerkkikoodi 5. Elementissä olevan arvon tarkistaminen.

Kolmannessa testissä tarkistettiin, että sivun kaksi kalenteria toimivat. Näillä pystyi valitsemaan alkupäivämäärän ja loppupäivämäärän, jotka suodattivat esittävät tiedot sen mukaan. Tässä käytettiin hyväksi Robot Frameworkissä oletuksena olevaa Get Time -avainsanaa, jolla saatiin aika valittua. Aloitusvuodeksi valittiin 2021 ja lopetusvuodeksi 2023. Koska käytettiin Get Time -avainsanaa, arvot päivittyvät sovelluksessa myös, kun kalenterivuosi vaihtuu. [18.]

Neljäs testi tarkisti, että sivulta löytyi valintaruutu, jonka pystyi aktivoimaan ja deaktivoimaan. Avuksi käytettiin QWebin Click Checkbox -avainsanaa [24].

Viides testi tarkisti, että pudotusvalikoiden arvot pystyi napsauttamaan yksitellen, ja testi tarkisti sen jälkeen, että valinta oli oikein. Testissä tarkistettiin ensin kaikki ehdotusvalinnat sisältävä pudotusvalikko ja sen jälkeen kaikki eri tyyppien sisältävä pudotusvalikko.

Viisi ensimmäistä testiä oli UI-testejä. Testeillä tarkistettiin, että kaikki tarvittavat elementit ja opasteet löytyivät sivulta. Testeillä tarkistetiin myös, että elementit, joita käyttäjät pystyivät napsauttamaan, toimivat oikein eivätkä aiheuttaneet virhetilanteita.

Kun kaikki UI-testit oli suoritettu onnistuneesti, oli mahdollista jatkaa e2e-testauksella. E2e-testauksessa oli mahdollista hyödyntää aikaisemmat testit, joilla kutsuttiin sivun eri elementit. UI-testeissä käytettiin ainoastaan Frontend-ympäristöä.

4.3 Verkkosivun e2e-testaus

Sovellukselle oli jo valmiina testejä, jotka loivat testidataa, muokkasi testidataa sekä poisti testidatan. Aikaisemmin tehtyjen joukossa oli myös testejä, jotka tarkistivat ehdotusta ja joko hyväksyivät tai hylkäsivät ehdotuksen. Näiden testien pohjalta oli mahdollista luoda testidataa, jonka avulla oli mahdollista tarkistaa yhteenvetosivun toimivuutta.

Testidata luotiin aina samoilla arvoilla, joten se oli helppo tunnistaa muun tietokantasisällön joukosta. Testidata luotiin täyttämällä ehdotuslomakkeita sovelluksessa samalla tavalla kuin käyttäjä. Käyttäessä sovellusta testidatan luomiseen huomattiin, että testidata ei luomisen jälkeen löytynyt yhteenvetosivulla.

Virheselvityksen jälkeen selvisi, että virhe oli kalenterissa. Ehdotuksen luontihetkellä, ja kun ehdotus oli hyväksytty tai hylätty, tallentui senhetkinen kellonaika. Testissä tarkistettiin yhteenvetosivulla oleva tieto heti, kun ehdotus oli valmis ja käytettiin kalenterin oletusaika, joka oli senhetkinen kellonaika. Yhteenvetosivulla kalenterin rajaus ei ottanut mukaan nykyistä kellonaikaa, vaan yläraja oli minuutti aikaisemmin. Kun kalenteri siirrettiin tulevaisuuteen, niin myös juuri tehty ehdotus ilmestyi yhteenvetosivulle. Virheestä tehtiin korjauspyyntö. Ennen kuin korjausta kalenterille oli tehty, oli kalenteriin asetettava tulevaisuuteen kellonaika, jotta haku palauttaisi myös juuri tehtyjä tietokantalisäyksiä ja mahdollisti oikeellisuus tarkistaminen. Sovelluksessa oli mahdollista valita suodatus, joka näytti käytössä olevan käyttäjän lisäämät tiedot. Kun testidata hyväksyttiin samalla käyttäjällä, jolla tarkistettiin sitä, pystyttiin hyödyntämään suodatinominaisuutta ja vähentämään haettavaa datamäärää sekä näyttämään pelkästään kirjautuneen käyttäjän luomaa dataa. Lisättyä tietoa pystyi suodattamaan eri tyypin avulla. Valitun tyypin jälkeen tarkistettiin, että suodatus toimi ja pelkästään rajausehtojen mukaista dataa oli näkyvillä. Aikaisemmin testidata oli luotu lähdekoodissa ja aikaväli, jonka välillä tieto haettiin, oli huomattavasti laajempi kuin testidatan aikaleima. Tämä tarkoitti, että kalenterissa ollut ongelma ei vaikuttanut aikaisempiin testeihin.

Testidata luotiin kolmilla eri arvoilla, joilla tarkistettiin, että jokainen arvo on suodatettavissa. Data on mahdollista luokitella hyväksytty- tai hylätty-arvolla. Tämän takia luotiin testidata, joka sisälsi sekä hyväksyttyjä ja hylättyjä arvoja ja tarkistettiin, että tieto näytettiin oikein.

Taulukko 2. E2e-testausvaiheet. (X= näkyy listalla, 0 = ei näy listalla)

Ehdotus	**Rajaus: Kaikki**	**Rajaus: Lisäys**	**Rajaus: Muutos**	**Rajaus: Poisto**
Lisäysehdotus	X	X	0	0
Muutosehdotus	X	0	X	0
Poistoehdotus	X	0	0	X

E2e-testeillä oli lähtötavoite tarkistaa, että kaikki kolme ehdotustyyppiä on yhteenvetosivulla näkyvillä ja pudotusvalikosta pystyi valitsemaan eri ehdotustyyppejä ja tämä suodattaisi pois muut ehdotukset.

Ensimmäisessä e2e-testissä kirjauduttiin käyttäjänä sovellukseen ja luotiin lisäysehdotus ja kirjauduttiin ulos. Sen jälkeen kirjauduttiin ylläpitäjänä sisään, jonka jälkeen ehdotus hylättiin ja siirryttiin tarkistamaan yhteenvetosivulla näytettävää tietoa.

Kuvassa 11 on esitetty kalenterielementti, josta valitaan vuosi, kuukausi, päivä sekä kellonaika.

Kuva 11. Kalenterielementti.

Kalenterin tunnit ja minuutit olivat myös muokattavissa nuoli ylös- ja nuoli alas -painikkeilla. Tämä oli helpoiten tehdä For-silmukassa, joka esitetään koodiesimerkissä 6.

```
*** Settings ***
Library    QWeb

*** Test Case ***
Change Calendar Time
    Click Element         //input[@id=”calendarTimeFrom”]
    ${times}=    Convert To Integer    3
    FOR  ${i}  IN RANGE  ${times}
        Click Element     //button[@label\\=”Vähennä minuutti”]
    END
    Click Text            Aseta

    Click Element         //input[@id=”calendarTimeTo”]
    FOR  ${i}  IN RANGE  ${times}
        Click Element     //button[@label\\=”Lisää minuutti”]
    END
    Click Text            Aseta
```

Esimerkkikoodi 6. For-silmukan käyttö Robot Frameworkissä.

Koodiesimerkin 6 alussa napsautettiin elementtiä calendarTimeFrom-id:llä jolla saatiin kalenteri esiin. ${times}-muuttujaan tallennettiin numero kolme. For-silmukassa tehtiin kolme kierosta ja jokaisella kierroksella napsautettiin Vähennä minuutti -painiketta. Kun kolmas kierros oli valmis, siirryttiin napsauttamaan Aseta-tekstiä. Silmukka oli vähentänyt kolme minuuttia calendarTimeFrom-kalenterista. Seuraavaksi testissä napsautettiin calendarTimeTo-elementtiä, joka avasi kalenteria. For-silmukassa tehtiin kolme kierrosta, jonka jokaisen kierroksen aikana napsautettiin Lisää minuutti -painiketta. Kun silmukka on suoritettu loppuun, napsautettiin Aseta-tekstiä, joka tallentaa kalenterimuutokset. Silmukka oli lisännyt kolme minuuttia calendarTimeTo-kalenteriin.

Ensimmäisessä e2e-testissä asetettiin kalenterin aloitusaika 3 minuuttia taaksepäin ja lopetusaika saman verran eteenpäin ja odotettiin, että tieto ladattiin näkyviin. Näin saatiin näkyviin vain viimeisimpiä ehdotuksia. Sen jälkeen valittiin, että vain käyttäjän omat tarkistetut ehdotukset näytetään sivulla ja odotettiin, että lista päivittyi. Taulukon latauduttua tarkistettiin, että taulukosta löytyi yksi lisäysehdotus ja sen tyyppi oli oikein ja se oli luokitettu hylätyksi. Seuraavaksi valittiin pudotusvalikosta lisäysehdotus ja tarkistettiin, että myös tämän rajauksen jälkeen taulukosta löytyi yksi lisäysehdotus ja sen tyyppi oli oikein sekä että se oli hylätty. Ensimmäisen e2e-testin yhteydessä oli tarkoitus kokeilla myös QWe-

bin tarjoamaa Use Table -avainsanaa. QWeb-dokumentaatiossa tämä on heidän esimerkkinsä käsitellä taulukoita. Valitettavasti jouduttiin kuitenkin toteamaan, että Use Table -avainsanaa ei soveltunut testaamaan yhteenvetosivulla näytettävää taulukkoa. Ongelma ilmeni taulukon html-rakenteessa, joka ei noudattanut yleistä taulukkorakennetta. Testeissä tarkistetaan, että lisäysehdotuksen sisältämä tieto löytyy sivulta. Mutta testi ei huomioi, onko tieto syötetty oikean sarakkeeseen. Use Table -avainsana olisi mahdollistanut solukohtaisen tarkistuksen. [24.]

Toisessa e2e-testissä kirjauduttiin ensin käyttäjänä sovellukseen ja luotiin lisäysehdotus ja kirjauduttiin ulos ja sen jälkeen kirjauduttiin ylläpitäjänä sisään ja hyväksyttiin ehdotus. Hyväksymisen jälkeen kirjauduttiin ulos ja peruskäyttäjänä taas sisälle ja tehtiin muutosehdotus. Muutosehdotuksen lähetyksen jälkeen käyttäjä kirjautui ulos. Tämän jälkeen kirjauduttiin sisään ylläpitäjänä, ja hyväksyttiin tämä ehdotus. Sen jälkeen siirryttiin yhteenvetosivulle. Yhteenvetosivulla asetettiin kalenteriin kello 5 minuuttia taaksepäin, jotta äsken luodut ehdotukset tulisivat näkyviin. Loppuaika siirrettiin myös muutama minuutti eteenpäin, jotta kalenterissa oleva bugi ei vaikuttaisi hakuun.

Kolmantena e2e-testinä luotiin testidata, joka sisälsi lisäysehdotuksen, muutosehdotuksen sekä poistoehdotuksen. Ensin kirjauduttiin sovellukseen peruskäyttäjänä ja täytettiin lisäysehdotus, jonka jälkeen kirjauduttiin ulos ja uudelleen sisään ylläpitäjänä ja hyväksyttiin ehdotus. Nämä samat vaiheet toistettiin myös muutosehdotukselle ja poistoehdotukselle. Kun kaikki kolme ehdotusta oli luotu ja hyväksytty, avattiin yhteenvetosivu. Kalenterista valittiin 5 minuuttia aikaisempi hetki ja loppuaika muutama minuutti eteenpäin. Ennen kuin taulukoiden arvoja tarkistettiin, aktivoitiin myös valintaruutu, että vain nykyisen käyttäjän tarkistetut ehdotukset näkyisivät. Ensimmäiseksi tarkistettiin, että kaikki kolme tyyppiä olivat näkyvillä ja ne oli hyväksytty. Kun kaikki arvot oli tarkistettu, oli

vuorossa valita yksi tyyppi kerrallaan pudotusvalikosta ja tarkistaa, että vain kyseinen tyyppi oli silloin näkyvillä.

Neljäntenä testattiin latausominaisuutta, jolla taulukon arvot pystyttiin lataamaan csv-tiedostona. Testi käynnisti latauksen ja sen jälkeen tarkisti, että lisävalinnat tulivat esille, jolla pystyi perumaan latauksen.

Viidennes testi tarkisti, että peruskäyttäjällä ei ole pääsyä yhteenvetosivulle eikä navigointipalkissa ole kyseistä sivua tarjolla.

Kun testit olivat saatu valmiiksi, lisättiin avainsana, joka tallensi taulukon (kuva 10) jokaisen solun arvo listaan. Tämä tehtiin ensin käyttämällä Get Element Count -avainsanaa ja xpath-polku sarakeotsikoiden sisältävälle elementille. Tämä palautti luvun sarake määrästä, joka tallennettiin ${columns}-muuttujaan. Tämän jälkeen For-silmukassa tallennettiin jokaisen sarakkeen nimi listaan. Lista tallennettiin toisen listan sisään, jotta saataisiin 2-uloitteinen lista taulukon sisältämästä tiedosta. Seuraavaksi tarkistettiin taulukon tietoa sisältävien rivien määrä antamalla Get Element Count -avainsanalle xpath-polku taulukon dataa sisältävälle elementille. Tämä palautti luku rivimäärästä, joka tallennettiin ${rows}-muuttujaan. Kuvassa 6 rivejä oli neljä, joten tallennettu luku oli myös neljä. Seuraavaksi käytiin koko taulukko läpi ja tallennettiin jokainen solun arvo listaan. Tämä toteutettiin kahden For-silmukan avulla, joka ulompi kasvatti rivinumeroa ja sisempi kasvatti sarake numeroa. Aina kun sisempi silmukka oli loppunut, tallennettiin listalle saadut arvot aikaisempaan listaan, joka sisälsi jo otsikkonimet. Avainsanalla saatiin 2-uloitteinen taulukko, jonka nimi oli ${list}. Listaan tallennetut arvot olivat mahdollista hakea antamalla kaksi lukua, joista ensimmäinen oli rivinumero ja toinen oli sarakenumero. Esimerkiksi listan ensimmäisen rivin ja ensimmäisen sarakkeen arvo oli saatavilla ${list}[0][0]-syntaksilla ja kuvassa 10 se oli Lomake-teksti.

Tällä avainsanalla pystyimme toteuttamaan saman ominaisuuden kuin Use Table -avainsana olisi meille tarjonnut. Testien tekeminen tämän avainsanan avulla jäi toteutumisvaiheeseen.

Kaikkiin Robot Framework-testeihin oli mahdollista määrittää Test Teardown -avainsanaa, joka kertoi, mitä tehdään jokaisen testin lopussa. Kun se yhdistettiin Run Keywork If Test Failed -avainsanan kanssa, joka kertoo, mitä tehdään, kun testi epäonnistuu ja lisättiin vielä Log Screenshot -avainsanaa Qweb-kirjastosta, saatiin aina kuvakaappaus siitä, miltä tilanne näytti, kun testi epäonnistui.

4.4 Testikattavuuden laajentaminen

Uusien testien avulla pystyttiin varmistamaan, että sivun teksti pysyi ennallaan. Uusilla testeillä pystyttiin tarkistamaan, että sovelluksen käyttöliittymä toimi. Testeillä testattiin alusta loppuun kaikki käyttäjien tekemät vaiheet ennen kuin ehdotus näkyi yhteenvetosivulla ja miltä se lopulta näytti yhteenvetosivulla. Valitettavasti tarkistukset eivät olleet solukohtaisia, vaan ne tarkistivat, että koko taulukon sisältämä tieto oli oikein. Tämä ominaisuus oli aikaisemmin käytössä ja vaatii jatkokehitystä toimiakseen myös jatkossa.

5 Lopputulos

Robot Framework -työkalu otettiin tämän opinnäytetyön aikana onnistuneesti käyttöön. Uudella testikokonaisuudella on helppo testata yhteenvetosivun toimivuutta.

5.1 Nykyinen tilanne

Aikaisemmin oli käytössä muutama kattava testi, jotka oli toteutettu Proctractor-työkalulla. Nyt testit on pilkottu kahteen ryhmään. UI-testit keskittyvät vain Frontend-osion toimivuuteen käyttäjän näkökulmasta. Toiseen ryhmään kuuluvat e2e-testit, jotka tarkistavat, että käyttäjän tekemät kaikki vaiheet alusta loppuun toimivat ja testeissä käytetään Frontend- ja Backend-osioita sekä tietokantaa. UI- sekä e2e-testit on toteutettu Robot Frameworkin avulla.

UI-testit sisältävät vähemmän koodia per testi ja toteuttavat vain jonkun pienemmän testitapauksen. E2e-testit ovat isompia kokonaisuuksia, joissa tarkistetaan

sovelluksen toimivuus alusta loppuun. Näin voi suorittaa joko kaikki testit tai valita UI- ja e2e-testit erikseen suoritettavaksi.

Protractor-testejä löytyy vielä muualla sovelluksessa. Myös nämä on tarkoitus korvata Robot Frameworkillä tulevaisuudessa. Niiden parissa jatketaan myös tämän opinnäytetyön jälkeen.

5.2 Mitä voisi tehdä eri tavalla

On muistettava, että Robot Framework tarjoaa laajan valikoiman lisäkirjastoja ja verkossa olevat ohjeet viitaa johonkin tiettyyn kirjastoon. Kirjasto pitää tuoda mukaan sovellukseen, jos ei sitä aikaisemmin ole käytetty tai etsiä vastaava ratkaisu nykyisistä käytössä olevista kirjastoista. Tähän ongelmaan törmättiin muutaman kerran.

Suosittelemme Robot Frameworkin tarjoamaa kuvakaappausominaisuutta. Tämä oli ominaisuus, joka löydettiin vasta projektin aloitettua. Jatkossa tätäkin ominaisuutta tullaan käyttämään. Kuvakaappauksen avulla testeistä saadaan tallennettua tietty tilanne, joka muuten voisi jäädä huomioimatta testaajalta testien toteutusvaiheessa. Kun Test Teardown -avainsanaan, joka määrittää mitä suoritetaan jokaisen testin lopussa, yhdistetään Run Keyword If Test Failed -avainsana ja Log Screenshot -avainsana, sadaan virhetilanteesta kuvakappaus, joka helpottaa virhetilanteen selvittämisen.

Vaikka Robot Frameworkin mukana tulee myös Sleep-avainsana, sitä kannattaa käyttää harkiten, koska se voi hidastaa sovelluksen testausta huomattavasti. Sleep-avainsanan käyttö on vaihtoehto, kun sivulla on joku elementti, jonka lataaminen kestää. Testit, jotka suoritetaan ja kohdistuvat sivuelementtiin, joka ei vielä ole latautunut valmiiksi, johtaa siihen, että testi epäonnistuu. Ne muutamat kohdat, jotka on toteutettu Sleep-avainsanan avulla, voisi tulevaisuudessa toteuttaa eri tavalla.

Monivaiheisissa testeissä lisättiin eri vaiheille konsoliin tulostettava teksti, kun testi oli saavuttanut tietyn kohdan koodista onnistuneesti. Konsolitulostus helpotti virheen paikantamista silloin, kun virheviesti ei ollut selkeä.

6 Yhteenveto

Opinnäytetyön tavoitteena oli korvata sovelluksessa olevia testejä, jotka oli toteutettu Protractor-työkalulla. Uudet testit luotiin Robot Framework -työkalulla. Robot Framework vaati alkuvaiheessa paljon opiskelua, koska kirjastot olivat kirjoittajalla kokonaan uusia. Myös uusien avainsanojen tekeminen vei aikaa. Myöhemmässä vaiheessa oli mahdollista hyödyntää aikaisemmin tehdyt avainsanat ja testien teko nopeutui. Koska projektin sovellus koostuu monesta eri lomakkeesta ja sivusta, niin testejä, jotka tarvitaan testatessa koko sovellusta, on yli sata kappaletta. Suuren testien määrän takia Robot Framework oli avuksi, koska pystyttiin käyttämään uudestaan luotuja avainsanoja. Pienimmissä sovelluksissa, joissa testejä on vain muutama, Robot Frameworkin hyöty voi olla vähäinen. Robot Frameworkin käyttäminen vaati syntaksin oppimisen. Pienimmissä sovelluksissa voi olla järkevämpää käyttää testityökalua, joka käyttää samaa ohjelmointikieltä kuin itse sovellus. Tarve avainsanojen uudelleenkäytettävyydellä ei ole iso, jos testejä on vain muutama.

Opinnäytetyön tavoitteena oli korvata Protractor-työkalu Robot Frameworkillä. Tämä onnistui melkein kokonaan. Testi, joka ei vielä opinnäytetyön aikana ollut toteutettu oli csv-tiedoston tarkistaminen. Myös taulukon eri solujen sisältämä tekstin tarkistus jäi kesken mutta tarvittava avainsana sen toteuttamiseen oli valmis. Muilta osin testit olivat onnistuneesti saatu toteutettua Robot Frameworkin avulla. Testeissä huomioitiin myös käyttäjän ja ylläpitäjän eri roolit, jotka ei ollut aikaisemmin huomioitu. Uudet testit, jotka tarkistivat yhteenvetosivulla, että kaikki teksti löytyi sivulta, varmisti sivun käytettävyyttä.

Lähteet

1 WHAT ARE AUTOMATION TESTING TOOLS? 9 TYPES & EXAMPLES. 2021. The QA Lead Team. Verkkoaineisto. <https://theqa-lead.com/topics/what-are-automation-testing-tools/> Luettu 6.5.2022.

2 Eran Kinsbruner. 2021. Manual Testing vs. Automated Testing. Verkkoaineissto. <https://www.perfecto.io/blog/automated-testing-vs-manual-testing-vs-continuous-testing> Luettu 6.5.2022.

3 Thomas Hamilton. 2022. Manual Testing Tutorial: What is, Concepts, Types & Tool. Verkkoainesto <https://www.guru99.com/manual-testing.html> Luettu 6.5.2022.

4 Thomas Hamilton. 2022. What is Regression Testing? Verkkoaineisto <Definition, Test Cases (Example). <https://www.guru99.com/regression-testing.html> Luettu 6.5.2022.

5 Javatpoint. 2021. Unit Testing. Verkkoaineisto <https://www.javatpoint.com/unit-testing> Luettu 6.5.2022.

6 Javatpoint. 2021. Integration testing <https://www.javatpoint.com/integration-testing> Luettu 6.5.2022.

7 Katalon. 2022. What is End-to-End (E2E) Testing? All You Need to Know. Verkkoaineisto. <https://katalon.com/resources-center/blog/end-to-end-e2e-testing> Luettu 6.5.2022.

8 William Hruska. 2019. Why UI testing is important? Verkkoaineisto. <https://hruskawilliam.medium.com/why-ui-testing-is-important-80ae086924e6> Luettu 6.5.2022.

9 Thomas Hamilton. 2022. What is WHITE Box Testing? Techniques, Example & Types. Verkkoaineisto. <https://www.guru99.com/white-box-testing.html> Luettu 6.5.2022.

10 Thomas Hamilton. 2022. What is BLACK Box Testing? Techniques, Example & Types. Verkkoaineisto. <https://www.guru99.com/black-box-testing.html> Luettu 6.5.2022.

11 Protractor. 2022. Protractor – end-to-end testing for AngularJS. Verkkoaineisto. <http://www.protractortest.org/#/> Luettu 6.5.2022.

12 Chinmayee Deshpande. 2021. TypeScript vs. JavaScript: Which One is Better? Verkkoaineisto. <https://www.simplilearn.com/tutorials/typescript-tutorial/typescript-vs-javascript> Luettu 6.5.2022.

13 *Jani Tarvainen. 2016. Mikä on TypeScript?* Verkkoaineisto. <https://symfony.fi/artikkeli/mika-on-typescript> Luettu 6.5.2022.

14 Protractor. 2022. Protractor - end-to-end testing for AngularJS. Verkkoaineisto. <http://www.protractortest.org/#/frameworks> Luettu 6.5.2022.

15 Jash Unadkat. 2021. BDD vs TDD vs ATDD : Key Differences. Verkkoaineisto. <https://www.browserstack.com/guide/tdd-vs-bdd-vs-atdd> Luettu 6.5.2022.

16 Sumit Bisht. 2013. Robot Framework Test Automation, E-kirja. Packt Publishing.

17 Robot Framework. 2022. Robot Framework ry. Verkkoaineisto. <https://robotframework.org> Luettu 6.5.2022.

18 Robot Framework Foundation. 2022. Robot Framework User Guide Version 5.0.0. Verkkoaineisto <https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#introduction> Luettu 6.5.2022.

19 Amir Rustamzadeh. 2020. Introducing Firefox and Edge Support in Cypress 4.0. Verkkoaineisto <https://www.cypress.io/blog/2020/02/06/introducing-firefox-and-edge-support-in-cypress-4-0/> Luettu 6.5.2022.

20 Cypress.io. 2022. Why Cypress? Verkkoaineisto. <https://docs.cypress.io/guides/overview/why-cypress#What-you-ll-learn> Luettu 6.5.2022.

21 Krishna Rungta. 2022. What is Selenium WebDriver? Difference with RC. Verkkoaineisto. <https://www.guru99.com/introduction-webdriver-comparison-selenium-rc.html> Luettu 6.5.2022.

22 QWeb. 2022. Keyword driven automation for the web. Verkkoaineisto. <https://github.com/qentinelqi/qweb> Luettu 6.5.2022.

23 Qentinel. 2021. qweb_workshop: QWeb workshop material. Verkkoaineisto. <https://github.com/qentinelqi/qweb_workshop/blob/main/01/browser.adoc> Luettu 6.5.2022.

24 QWeb. 2022. QWeb. Verkkoaineisto. <https://qentinelqi.github.io/qweb/QWeb.html> Luettu 6.5.2022.

25 No Data No Business. 2022. Find an xPath with limited HTML knowledge. Verkkoaineisto. <https://nodatanobusiness.com/resources/find-an-xpath-with-little-html-knowledge/> Luettu 6.5.2022.