OULUN AMMATTIKORKEAKOULU

Miika Koppelo

AUTOMAATIOTESTAUKSEN KÄYTTÖÖNOTTO VERKKOSIVUN KÄYTTÖLIITTYMÄLLE

AUTOMAATIOTESTAUKSEN KÄYTTÖÖNOTTO VERKKOSIVUN KÄYTTÖLIITTYMÄLLE

Miika Koppelo
Opinnäytetyö
Syksy 2023
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä(t): Miika Koppelo
Opinnäytetyön nimi: Automaatiotestauksen käyttöönotto verkkosivun käyttöliittymälle
Työn ohjaaja(t): Meija Lohiniva
Työn valmistumislukukausi ja -vuosi: Syksy 2023 Sivumäärä: 39

Opinnäytetyön tavoitteena oli dokumentoida testiautomaation suunnittelu ja toteutuksen käyttöönotto verkkosivun käyttöliittymää varten. Tavoitteena oli tarkastella testaustyökalun valintaa ja testiautomaation suunnittelun ja toteutuksen käyttöönottoa esimerkkejä hyödyntäen. Tarkoituksena oli siis tarjota verkkosivun käyttöliittymän testiautomaatioon ohjeet, jonka avulla lukija voisi päästä alkuun tämänkaltaisessa projektissa ja hyödyntää tästä työstä saatua tietoa.

Opinnäytetyön teoriaosuus käsittelee yleisesti ohjelmistotestausta, testauksen prosessimallia ja automaatiotestausta. Teoriaosuudessa tiedot on koottu aiheeseen liittyvistä verkkosivustoista ja kirjallisuudesta. Testiautomaation suunnitteluosuudessa käydään läpi käyttöliittymän automaatiotestaukseen liittyvää teoriaa, valitaan sopiva testaustyökalu ja suunnitellaan esimerkkitestit tätä työtä varten. Suunnitteluosuudessa on käytetty apuna verkkolähteitä ja testaustyökalujen omia dokumentteja. Testauksen toteutusvaiheessa näytetään testaustyökalun asennus, esitellään toteutetut automaatiotestit kuvien ja esimerkkien kautta ja esitellään testien lopputulos ja raportointi. Toteutuksen apuna on ollut testaustyökalun ja selainkirjaston omat dokumentit.

Lopputulokseksi työstä muodostui ohjeet testaustyökalun valintaan, automaatiotestauksen suunnitteluun ja toteutukseen verkkosivun käyttöliittymän automaatiotestauksen alkuun pääsemiseksi. Lisäksi työhön on koottu teoriatietoa ohjelmistotestauksesta, automaatiotestauksesta ja verkkosivujen käyttöliittymistä.

Asiasanat: ohjelmistotestaus, automaatiotestaus, käyttöliittymätestaus

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author(s): Miika Koppelo
Title of thesis: Getting started with website user interface test automation
Supervisor(s): Meija Lohiniva
Term and year when the thesis was submitted: Autumn 2023
Number of pages: 39

The goal of this thesis was to create guidelines on getting started with the design and implementation of test automation for a website user interface. The goal was to compare the selection of testing tools and introduce test automation design and implementation using examples. The purpose was to provide instructions which would allow the reader to get started on this kind of project and utilize the information from this thesis.

The result of this thesis were instructions for choosing a testing tool and getting started on automation test design and implementation for the website's user interface. In addition, this thesis provides theoretical information about software testing, automation testing and website user interfaces.

Keywords: software testing, test automation, user interface testing

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön tavoitteena oli suunnitella ja toteuttaa automaatiotestauksen käyttöönotto verkkosivun käyttöliittymälle. Testauksen kohteena oli oamk.fi-verkkosivusto ja työssä keskitytään tämän verkkosivun käyttöliittymän toiminnallisuuksien ja elementtien testaamiseen. Opinnäytetyö antaa lukijalle vastauksen seuraaviin tutkimuskysymyksiin:

1. Millä perusteella testaustyökalu voidaan valita verkkosivun käyttöliittymälle?
2. Miten testiautomaatio suunnitellaan verkkosivun käyttöliittymälle?
3. Miten testaustyökalua käytetään?
4. Miten verkkosivun käyttöliittymän automaatiotestaus tapahtuu valitulla testaustyökalulla?

Tämän opinnäytetyön on tarkoitus olla ohjeen kaltainen teos ja opinnäytetyön luettuaan lukijalla pitäisi olla käsitys siitä, miten verkkosivun käyttöliittymän automaatiotestauksessa pääsee alkuun. Automaatiotestauksen työkalun valinnasta, suunnittelusta ja toteutuksista on valittu useita esimerkkejä, mikä auttaa saamaan paremman kuvan siitä, minkälaista tämänkaltaisen projektin aloittaminen voisi olla.

Opinnäytetyön rakenne muodostuu alkuosan johdanto- ja teorialuvuista sekä testauksen kohteeksi valitun sivuston esittelystä. Johdannon jälkeen luvussa 2 kerron aiheeseen liittyvästä teoriasta, jossa avaan lukijalle ohjelmistotestauksen ja automaatiotestauksen käsitteitä ja tarpeellisuutta sekä kerron ohjelmistotestauksessa yleisesti käytetystä V-malli-prosessimallista. Luvussa kolme kerron käyttöliittymän testiautomaation suunnittelusta. Tämä luku alkaa teoriasta, jossa kerrotaan yleisesti verkkosivujen käyttöliittymistä ja niiden automaatiotestauksesta. Seuraavaksi luvussa kerrotaan testaustyökalun valinnasta, miten työkalu voidaan valita ja mikä työkalu valittiin tätä työtä varten. Valitusta työkalusta on myös teoriaosuus, jossa käsitellään työkalun arkkitehtuuria ja syntaksia ja testien raportointia. Lopuksi tässä luvussa kerron testien suunnittelusta ja testitapausten valinnasta tätä työtä varten. Neljännessä luvussa kerron suunnittelun pohjalta tekemieni testien toteutuksesta. Tämä luku sisältää ohjeet testaustyökalun asennukseen, dokumentaatiota testien toteutuksesta ja raportoinnista sekä pohdintaa testien luotettavuudesta. Viimeinen eli viides luku on opinnäytetyöni päätäntöluku.

2 OHJELMISTOTESTAUS

Ohjelmistotestauksen käsitteellä tarkoitetaan suunnitelmallista virheiden etsintää ohjelmaa tai jotakin sen osaa suorittamalla (Haikala & Mikkonen 2011, 205). Ohjelmistotestaus voidaan määritellä myös tavaksi verifioida ja validoida kehitettävän ohjelmiston toiminta, jotta ohjelmistossa olisi mahdollisimman vähän vikoja. Näin ohjelmisto saadaan vastaamaan vaadittuja teknisiä vaatimuksia ja suunnitelmia sekä kykenemään suoriutumaan normaalisti myös poikkeustilanteissa. Näiden lisäksi ohjelmistotestauksen tavoitteena on myös parantaa ohjelmiston tehokkuutta, virheettömyyttä ja käytettävyyttä. (Gupta 2023.)

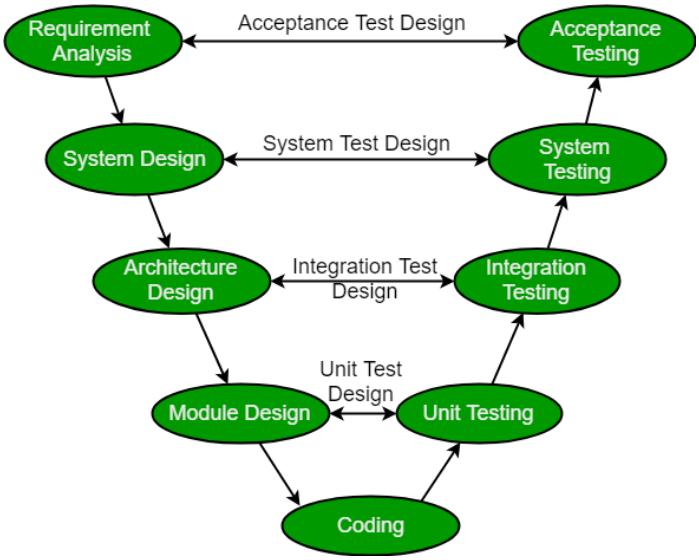
Mikään ohjelma ei kuitenkaan ole täysin virheetön. Ohjelmistotestauksella voidaan testata vain pieni osa kaikista mahdollisista ohjelmiston toiminnoista. Tämän vuoksi ohjelmistotestaus usein onkin satunnaista kokeilemista erilaisilla syötteillä. Tästä huolimatta testaukseen kannattaa kuitenkin panostaa, mutta edellä mainituista syistä täytyy kuitenkin pitää mielessä, että hyvätkään testitulokset eivät takaa täysin luotettavaa ohjelmistoa. (Haikala & Mikkonen 2011, 205.)

2.1 Ohjelmistotestauksen tarpeellisuus

Laajemmin tarkasteltuna ohjelmistotestausta tarvitaan havaitsemaan bugeja eli ohjelmistovirheitä suoritettavasta ohjelmasta ja testaamaan, vastaako ohjelma asiakkaan vaatimuksia. Tämä auttaa ohjelmiston kehittäjiä korjaamaan vikoja ja tuottamaan parempilaatuisia tuotteita. Ohjelmistokehityksessä on useita vaiheita, missä voi tapahtua inhimillisiä virheitä ja tämä voi johtaa siihen, että kehitettävä ohjelma ei vastaa vaatimuksia. Näistä testaukseen liittyvinä esimerkkeinä ovat koodaukseen liittyvät virheet tai osaamisen puute. Huolellisella testauksella pyritään välttämään huonon koodin päätymistä tuotantoon. Ohjelmistotestauksen tärkeys korostuu myös siinä, että mikäli huolellista testausta ei tehtäisi, voisivat ohjelmistovirheet aiheuttaa huomattavasti lisätöitä, kalliita häiriöitä ja pahimmassa tapauksessa ihmisvahinkoja. Ohjelmistokehityksen historiassa onkin tapahtunut useita ohjelmistovirheitä, jotka ovat aiheuttaneet edellä mainittujen kaltaisia ihmishenkien menetyksiä tai miljoonien eurojen rahallisia tappioita. Esimerkiksi vuonna 1962 tuhoutui 18 miljoonan dollarin arvoinen The Mariner -avaruusalus, jonka yksinkertainen ohjelmistovirhe johti aluksen ohjautumiseen väärään suuntaan ja lopulta tuhoutumiseen. Huolellisella testauksella tämäkin olisi voitu mahdollisesti välttää. (Try QA n.d.)

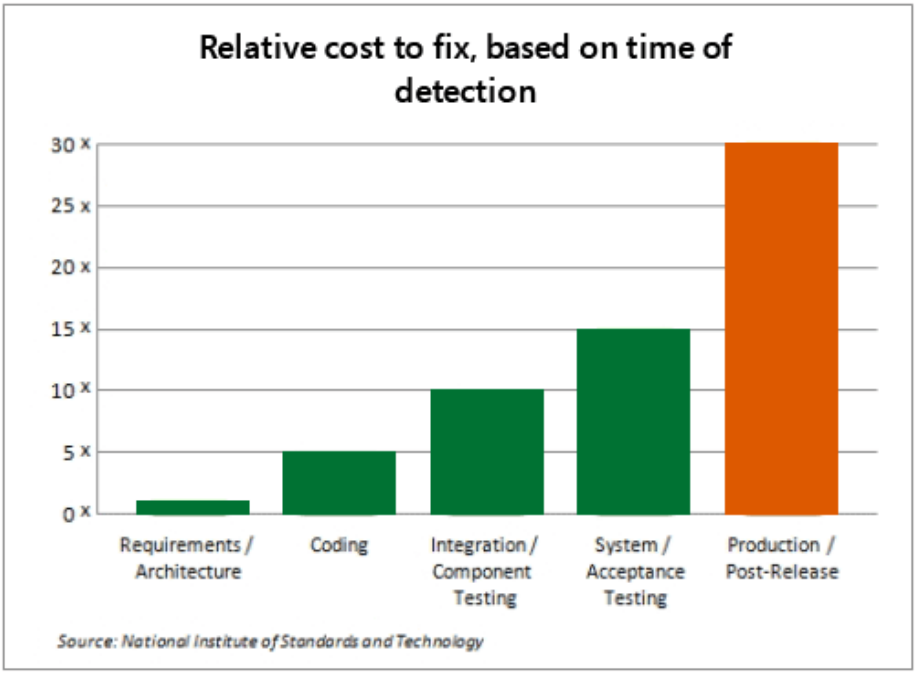
2.2 Ohjelmistotestauksen V-malli ja tasot

V-malli on ohjelmistokehityksessä käytettävä prosessimalli, jonka avulla havainnollistetaan kehitettävän ohjelman ja testauksen suhdetta. Mallin mukaan jokaisella ohjelman suunnittelutasolla on myös oma vastaava testaustasonsa. Testauksen tuloksia todetaan oikeiksi vertailemalla niitä vastaavan tason dokumentteihin (kuva 1). (Haikala & Mikkonen 2011, 206.)



KUVA 1. V-malli (Dharmendra 2023).

V-malli on kehitetty vesiputousmallin pohjalta, jossa ohjelmistokehitys ja testaaminen tapahtuvat peräkkäisessä järjestyksessä. Testaus tapahtuu vesiputousmallissa vasta ohjelmiston implementaation jälkeen, ja tämän vuoksi myöhemmin on havaittu ongelmaksi se, että oleelliset yksityiskohdat saattavat helposti jäädä huomaamatta arkkitehtuuria ja vaatimuksia suunnitellessa. Tämä saattaa aiheuttaa virheellisesti tuotettuja sovelluksia, joita on kallista korjata. V-malli onkin tätä ongelmaa varten luotu prosessi, jonka avulla mahdollisiin ongelmiin pystytään tarttumaan jo varhain. Kuten kuvasta 2 nähdään, mitä aikaisemmin virhe havaitaan, sitä halvempaa sen korjaaminen on. (Hamilton 2023a.)



KUVA 2. Ohjelmistovirheiden korjaamisen suhteellinen hinta kehitysvaiheeseen nähden (Hamilton 2023a).

V-mallin mukaiset testaustasot ovat yksikkötestaus, integrointitestaus, järjestelmätestaus ja hyväksymistestaus. Yksikkötestauksessa testataan ohjelman luokkia tai moduuleita. Nämä testit suunnitellaan arkkitehtuurin ja yksityiskohtaisten suunnitelmien pohjalta ja näitä verrataan toteutetun luokan toimintaan. Integrointitestauksessa testataan luokkien ja moduulien yhdistelmiä, jotka muodostavat osajärjestelmiä. Integrointitestauksen tarkoituksena on testata, miten järjestelmän eri osat toimivat keskenään. (Haikala & Mikkonen 2011, 206−208.) Opinnäytetyöni aihe liittyy käyttöliittymien testaukseen ja tämä testausvaihe toteutetaan järjestelmä- tai hyväksymistestauksen tasolla. Seuraavissa luvuissa kerron näistä tasoista tarkemmin.

2.2.1 Järjestelmätestaus

Järjestelmätestauksessa testataan koko kehitettävää järjestelmää. Tyypiltään järjestelmätestausta voidaan kutsua mustalaatikkotestaukseksi eli testaajalla ei ole tietoa ohjelman tai koodin toteutuksesta, vaan testaukset toteutetaan sen pohjalta, mikä on järjestelmän odotettu käyttäytyminen tai lopputulos. Tässä testausvaiheessa testauksen suorittajien tulisi olla järjestelmän kehityksestä riippumattomia henkilöitä. Järjestelmän kehityksessä mukana olleet henkilöt keskittyvät herkästi vain järjestelmän toimiviin osiin testauksessaan, mistä syystä tämä vaihe suoritetaan usein erillisen

testaustiimin johdosta tai järjestelmän tilanneiden asiakkaiden puolelta. (Haikala & Mikkonen 2011, 208−209.)

Järjestelmätestauksessa on useita testauksen kohteita, joista esimerkkeinä toimivat muun muassa ominaisuuksien testaus, käyttöliittymän testaus, kuormitustestaus ja tietoturvatestaus. Nämä ovat kuitenkin vain pieni osa kaikista järjestelmätestauksessa testattavista kohteista. (SoftwareTestingHelp 2023.)

2.2.2 Hyväksymistestaus

Hyväksymistestaus on viimeinen V-mallissa käytettävä testauksen taso. Hyväksymistestaus ja järjestelmätestaus ovat tyypiltään melko lähellä toisiaan, mutta selkeä erottava tekijä näiden kahden välillä on se, että hyväksymistestauksen tekee lähes aina asiakas, joka kyseisen järjestelmän on tilannut. Asiakas vertaa tuotetta niihin vaatimuksiin, minkä pohjalta tuote oli tilattu ja mitä tuotteen on haluttu tekevän. Tätä vaihetta kutsutaan usein myös loppukäyttäjätestaukseksi, koska tässä vaiheessa käytetään aitoa tuotantoon menevää dataa. Hyväksymistestauksen tuloksena asiakas tai muu valtuutettu taho päättää, hyväksytäänkö järjestelmä vai ei. Tästä syystä tässä vaiheessa epäonnistunut testi tarkoittaa myös epäonnistunutta tuotetta. (SoftwareTestingHelp 2023.)

2.3 Automaatiotestaus

Testausta voidaan suorittaa kahdella eri tavalla: manuaalitestauksella ja automaatiotestauksella. Näiden kahden erona on se, että manuaalitestauksen tekee ihminen käsin, kun taas automaatiotestaus tapahtuu skriptien avulla ja testit suorittaa tietokone automaatiotyökaluja hyödyntäen. Testiautomaatiosovellukset kykenevät myös syöttämään testidataa testattavaan järjestelmään ja vertaamaan niitä odotettuihin tuloksiin. Tämän vertailun pohjalta testisovellus tuottaa testiraportteja, joista voidaan nähdä testien tulokset. Jatkuvan ohjelmistokehityksen aikana samoja testitapauksia täytyy suorittaa lukuisia kertoja uudelleen, jotta varmistutaan siitä, että tehtyjen muutosten jälkeen järjestelmä toimii vielä halutulla tavalla. Monia testitapauksia pystytään automatisoimaan, koska näissä automatisoiduissa testeissä testattavan datan ei pitäisi muuttua. (Hamilton 2023b.) Taulukossa 1 on lyhyesti selitettynä, minkälaisia testejä pystytään automatisoimaan.

TAULUKKO 1. Automaatiotestauksen tyyppejä (Alexander 2023).

Automaatiotestauksen tyypit	Tyypin kuvaus
Regressiotestaus	Tämän tyyppiset automaatiotestit pyrkivät varmistamaan, että uusi muutos koodiin ei riko sovellusta.
Yksikkötestaus	Testaa tiettyä osaa koodista. Onnistunut testi antaa signaalin koodin onnistuneesta käännöksestä ja suorituksesta.
Savutestaus	Testaa sovelluksen toimivuuden kokonaisuutena uuden koontiversion (build) tekemisen jälkeen.
Integraatiotestaus	Vahvistaa, että kommunikaatio toimii ohjelmiston luokkien ja moduulien välillä.
Käyttöliittymätestaus	Nämä automaatiotestit kohdistuvat komentoriviin tai graafiseen käyttöliittymään. Esimerkiksi tekstin syöttämistä tai nappien painalluksia.
Turvallisuustestaus	Sovelluksen tietoturvaan liittyviä automaatiotestejä. Esimerkiksi sovelluksen käyttäjävaltuutukset, hyökkäysvektorit ja tunnettujen koodivirheiden testaus.
Suorituskykytestaus	Testaa sovelluksen suorituskykyä. Esimerkiksi laskentateho, verkon ruuhkautuminen ja laajennettavuus. Testien tarkoituksena on puskea sovellus äärirajoille.
Hyväksymistestaus	Varmistaa, että sovellus vastaa sille asetettuja vaatimuksia. Esimerkiksi vaatimuksissa voi olla asetettuna, että sovelluksessa on käyttäjän sisäänkirjautuminen ja automaatiotesti testaa tämän ominaisuuden toimivuuden kokonaisuutena.

Taulukossa näkyvät automaatiotestauksen tyypit eivät ole toisiaan poissulkevia, mikä tarkoittaa tässä tapauksessa sitä, että eri testauksen tyyppejä voidaan yhdistää testitapauksissa. Esimerkiksi regressiotestaukseen saattaa kuulua käyttöliittymätestauksessa käytettäviä elementtejä.

Automaatiotestauksessa käytetään erillistä ohjelmaa tai kehystä, joka suoritetaan kehitettävän järjestelmän rinnalla. Testauskehykset tarjoavat käyttäjilleen valmiita kirjastoja parantamaan testien rakennetta, vähentämään koodia, helpottamaan testien ylläpitoa ja parantamaan uudelleenkäytettävyyttä. Testauskehyksiä on kehitetty useita ja jokaisella niistä on omat vahvuutensa (Hamilton 2023b.)

2.4 Automaatiotestauksen hyödyt ja haitat

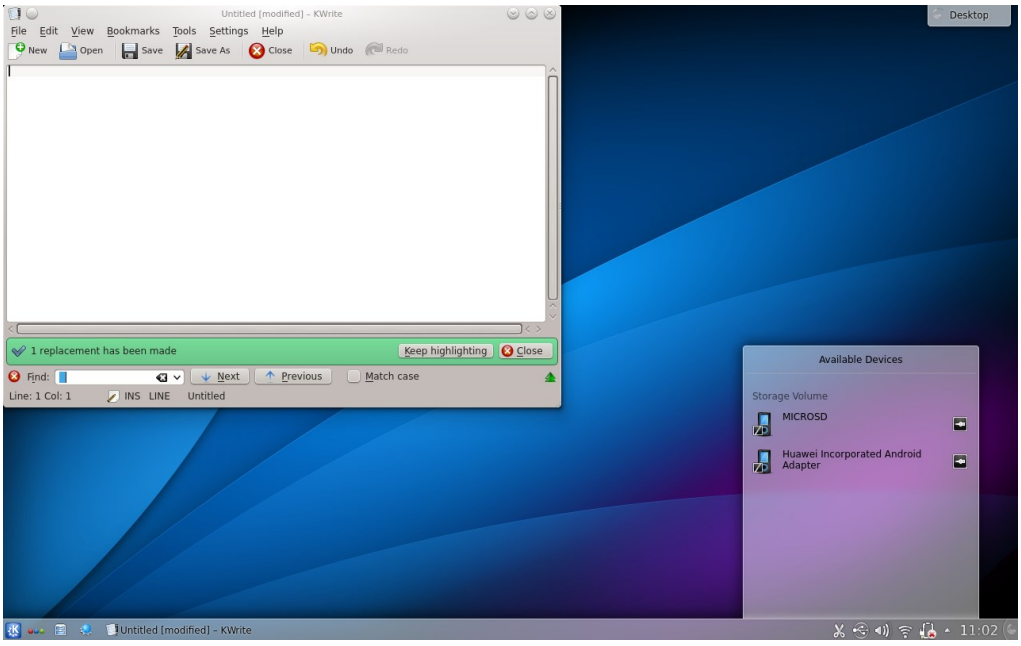
Automatisoitujen testien tarkoituksena on vähentää manuaalisen testaamisen tarvetta, tällä säästetään huomattavasti resursseja. Testaus on nopeampaa, koska testauskehys kykenee suorittamaan useamman testitapauksen yhtä aikaa ja kehys pystyy myös antamaan syötteet välittömästi. Kun testin suorittajana on automaatiotyökalu, säästetään myös testaajien työmäärää. Näiden lisäksi automatisointi lisää testaamisen luotettavuutta, koska testit suoritetaan aina samalla tavalla ja tämä on hyödyllistä etenkin järjestelmän kannalta kriittisissä testitapauksissa. Testien automatisoiminen mahdollistaa myös laajemman testauskattavuuden, koska kaikkia testejä ei tarvitse suorittaa käsin. (Hayes 2004, 8−13.)

Automaatiotestauksen heikkous on kuitenkin se, että sillä ei pystytä testaamaan mitään, missä on muuttuvaa dataa. Tämän vuoksi automatisoitujen testien suunnittelussa tulee olla tarkka. Väärin tehty automaattinen testi saattaa myös ohjata järjestelmän kehittäjää tekemään vääränlaisia muutoksia omaan koodiinsa. Lisäksi käyttöliittymän testauksessa automatisoidut testit eivät pysty havaitsemaan esimerkiksi, minkälainen käyttöliittymän muutos on hyvä käytettävyyden kannalta. (Hayes 2004, 15−17.)

3 KÄYTTÖLIITTYMÄN AUTOMAATIOTESTAUKSEN SUUNNITTELU

3.1 Verkkosivun käyttöliittymä

Lähes kaikilla verkkosivuilla on graafinen käyttöliittymä, joka on kuviin, teksteihin ja muihin visuaalisiin elementteihin perustuva tapa olla vuorovaikutuksessa elektronisten laitteiden kanssa. Graafinen käyttöliittymä kehitettiin 1970-luvulla vastaamaan tekstipohjaisten komentoliittymien heikon käytettävyyden haasteisiin tavallisilla käyttäjillä. Graafisista käyttöliittymistä on nykypäivänä tullut käyttäjäkeskeisen suunnittelun standardi ohjelmistojen kehityksessä. Ne mahdollistavat käyttäjilleen keinoja operoida tietokoneita ja muita laitteita intuitiivisesti suoraan hyödyntämällä graafisia kuvakkeita, kuten esimerkiksi painikkeita, valikkoja, vierityspalkkeja ja ikkunoita. Näihin kuvakkeisiin käyttäjä pääsee käsiksi muun muassa tietokoneen hiirellä, näppäimistöllä tai kosketusnäytöllä. Graafiset käyttöliittymät ovat nykypäivänä sisällytettynä lähes kaikissa interaktiivisissa digitaalisissa sovelluksissa, kuten esimerkiksi pankkiautomaateissa, itsepalvelukassoilla, videopeleissä, älypuhelimissa ja käyttöjärjestelmissä. (HEAVY.AI 2022.)



KUVA 3. Nykyaikainen käyttöliittymä (Wikipedia 2023).

Käyttöliittymien hyöty tulee siitä, että niiden avulla voidaan huomattavasti parantaa elektronisten laitteiden helppokäyttöisyyttä tavallisilla käyttäjillä. Graafisissa käyttöliittymissä käyttäjien ei

välttämättä tarvitse osata syöttää sovellukselle komentoja itse, vaan käyttöliittymä tarjoaa käyttäjälleen visuaalisia vihjeitä, miten sovellusta voi käyttää. (HEAVY.AI 2022.)

3.2 Verkkosivun käyttöliittymän testaus

Verkkosivun käyttöliittymän testauksessa testauksen kohteena ovat kaikki käyttöliittymästä löytyvät ominaisuudet ja visuaaliset elementit. Verkkosivun käyttöliittymän testaamisen tarkoituksena on varmistaa, vastaavatko sovelluksen toiminnallisuudet, kuten esimerkiksi näkymät, valikot, napit ja ikonit, asetettuja vaatimuksia. Verkkosivun käyttöliittymän testaus keskittyy näiden elementtien toiminnallisuuden lisäksi myös käyttöliittymän suunnittelun rakenteeseen. Tavallinen käyttäjä näkee vain käyttöliittymän eikä hänellä ole tietoa järjestelmän toiminnallisuudesta. Hyvät käyttöliittymät ovat helposti ymmärrettäviä ja käytettäviä. Jos käyttäjä huomaa, että käyttöliittymää on vaikea käyttää ja monimutkainen ymmärtää, käyttäjä ei välttämättä enää halua käyttää kyseistä sovellusta uudelleen. Tästä syystä hyvään käyttöliittymän testaamiseen kuuluu myös varmistaa, miten suunnittelussa on onnistuttu. (Hamilton 2023c.)

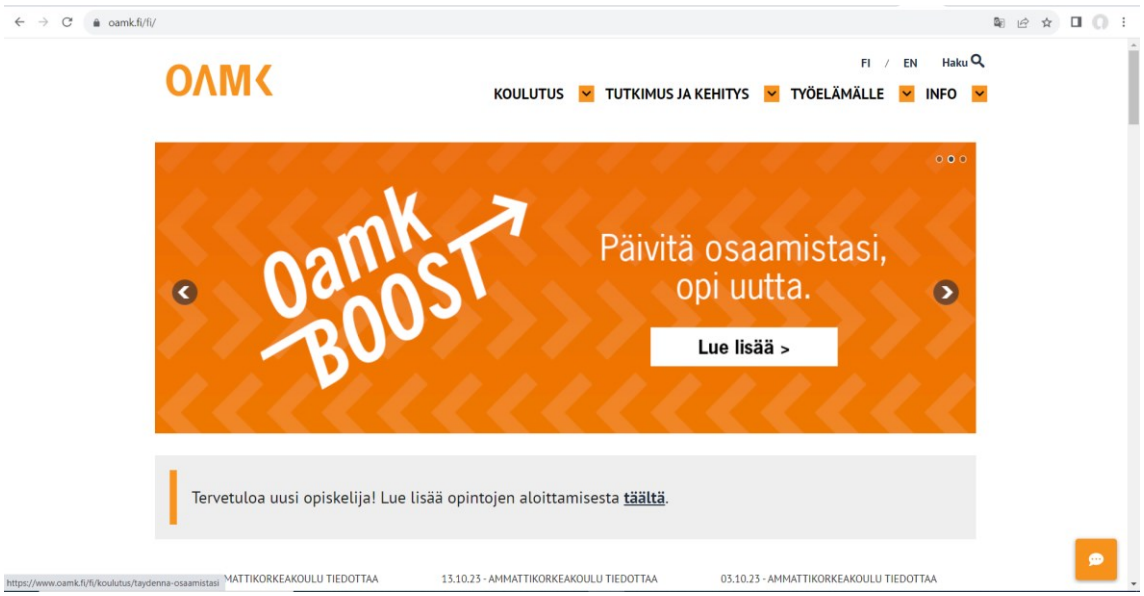
On olemassa lukuisia erilaisia tapoja testata käyttöliittymää. Esimerkiksi verkkosivujen käyttöliittymän testaamisessa on otettava huomioon, että selainikkunan koko voi muuttua erittäin pienestä resoluutiosta hyvinkin suureen resoluutioon. Käyttöliittymien elementtien tulee asettua oikein ruutuun sen mukaan, mitä kokoa käytetään. Käyttöliittymästä löytyvien painikkeiden tulisi suorittaa niille asetettu toiminto oikein. Esimerkiksi jostain kuvakkeesta painamalla pitäisi siirtyä toiselle sivulle onnistuneesti. Vastaavasti myös virhetilanteissa virheviestien tulisi tulla ruutuun näkyviin oikealla tavalla. Sivun tekstin ja kuvien tulisi olla selkeitä ja oikealla tavalla asetettuja. Värien tulisi olla miellyttäviä eivätkä ne saisi sekoittua muihin väreihin. Lisäksi myös syöteikkunoiden pitäisi toimia oikein, kun esimerkiksi niihin syöttää oikeanlaista dataa, vääränlaista dataa ja erikoismerkkejä. (Hamilton 2023c.)

Näiden esimerkkitapausten pohjalta pystyy jo suhteellisen helposti erottelemaan, mitä asioita käyttöliittymien testaamisessa pystytään automatisoimaan ja mitä ei. Testiautomaatiossa käytettävä sovellus ei kykene erottamaan, mikä on hyvää käyttöliittymän suunnittelua ja mikä ei. Testiautomaatiosovellukset kykenevät kuitenkin kahdella eri tavalla testaamaan graafista käyttöliittymää. Ensimmäinen tapa on se, että automaatiotestisovellukselle annetaan käsky ”klikata” käyttöliittymässä olevia elementtejä tietyssä järjestyksessä ja sovellus raportoi, tapahtuiko testin suoritus

onnistuneesti. Tämän tavan hyötynä on helppokäyttöisyys eli testejä on helppo luoda. Huonona puolena on taas se, että testejä joutuu herkästi tekemään uudelleen, koska tämän kaltaiset testit taipuvat huonosti dynaamisiin käyttöliittymän muutoksiin. Toinen tapa on luoda testiskriptejä, joissa tunnistetaan graafisen käyttöliittymän elementit tietynlaisilla paikantimilla ja tämän jälkeen niihin suoritetaan haluttuja komentoja. Tämän tavan hyöty on se, että skriptit pystyvät dynaamisesti paikantamaan elementit, vaikka niiden paikka tai koko olisi muuttunut. Huono puoli taas se, että skriptien tekeminen vie enemmän aikaa ja vaatii enemmän osaamista. (Hamilton 2023c.)

3.3 Testattava käyttöliittymä

Opinnäytetyölläni ei ole tilaajaa, joten päätin valita opinnäytetyöni testauksen kohteeksi oamk.fi-verkkosivuston, joka on Oulun ammattikorkeakoululle luotu oma sivusto (kuva 4). Sivustolta löytyy monenlaista OAMK:n toimintaan, kuten esimerkiksi koulutukseen, tutkimukseen, työpaikkoihin ja muuhun ajankohtaiseen liittyviä asioita.



Kuva 4. oamk.fi-etusivu

Testaaminen keskittyy oamk.fi-sivuston etusivulta löytyviin käyttöliittymän ominaisuuksiin. Oamkin etusivulta löytyy paljon visuaalisia elementtejä, joita pystytään testaamaan. Kuvassa näkyy esimerkiksi OAMK-logo, jota klikkaamalla päästään OAMK-verkkosivuston etusivulle. Tätäkin elementtiä voitaisiin testata esimerkiksi siten, että mitä tapahtuu, kun klikkaa logoa jollain toisella sivulla kuin

aloitussivulla tai kun ”klikkaa” logoa aloitussivulla, ja onko logo ylipäätään edes klikattavissa. Tämä on vain yksi esimerkki, mitä sivustolta voitaisiin testata.

3.4 Käytettävät työkalut

Verkkosivujen automaatiotestauksessa on oleellista päästä käsiksi testattavaan ympäristöön ja valita verkkosivujen käyttöliittymän testaamiseen sopiva työkalu. Aivan minimivaatimukset ovat siis verkkoselain, testauskehys ja ohjelmointiympäristö. Valittu kehys ratkaisee, millä ohjelmointikielellä testien koodaus tullaan tekemään. Ohjelmointiympäristökin riippuu pitkälti siitä, mitä testauskehystä käytetään.

3.4.1 Testaustyökalun valinta

Testaustyökaluja tutkiessani huomasin, että verkkosivujen käyttöliittymien testaamiseen sopivia vaihtoehtoja on monia. Työkalut eroavat toisistaan muun muassa siinä, että osa niistä tarjoaa käyttäjälleen oman ladattavan sovelluksen ja osassa näistä sovelluksista ei välttämättä tarvita lainkaan koodaamista. Jotkin työkalut ovat erikoistuneet tiettyihin asioihin, kuten esimerkiksi puhelimien sovelluksiin tai tiettyihin ohjelmointikieliin. Yrityksissä työkalun valinta täytyy tehdä ensisijaisesti sen mukaisesti, mikä soveltuu parhaiten yrityksen käyttötarkoituksiin ja mitä ohjelmointikieltä suositaan. Tässä opinnäytetyössä valinnan ei tarvinnut olla niin tarkka, mutta halusin testityökalun vastaavan seuraaviin vaatimuksiin:

- Työkalun tulee olla helposti käyttöönotettava.
- Työkalun tulee olla ilmainen eli avointa lähdekoodia.
- Testien tekemisen ja koodisyntaksin ymmärtäminen tulisi olla mahdollisimman helppoa ja yksinkertaista.

Näiden vaatimusten perusteella valitsin kolme ehdokasta. Ensimmäinen näistä on Selenium, joka on verkkoselaimiin erikoistunut automaatiotyökalu. Selenium tarjoaa käyttäjälleen WebDriver -kirjaston, joka tukee useita ohjelmointikieliä. Seleniumilla on myös muita työkaluja kuten Selenium IDE, jossa käyttäjä voi muun muassa tallentaa testejä record and replay -työkalulla. Tämä mahdollistaa helposti ja nopeasti testien luomisen, kun järjestelmä nauhoittaa käyttäjän syötteet ja toistaa ne perässä testissään. Selenium ei tarjoa testeille tulosraportointia, vaan tämä vaatii erillisen työkalun. (Ashiq 2022.)

Seuraavana on Robot Framework, ilmainen testiautomaatiossa käytettävä työkalu. Robot Framework on yksi suosituimmista testaustyökaluista, koska se on avointa lähdekoodia ja helposti laajennettavissa. Robot Frameworkin yksi ominaisuus on avainsanapohjainen (Keyword-driven) syntaksi, joka tekee testien tekemisestä helppoa ja myös koodista helpommin ymmärrettävää. Robot Framework taipuu lähes kaikenlaiseen testaamiseen laajan tukensa ansioista. Verkkosivujen testaamisessa Robot Framework voi hyödyntää Seleniumin kirjastoa. Robot Framework tukee ainoastaan Python-ohjelmointikieltä. (Ashiq 2022.)

Valitsin verrattavaksi vielä kolmannen vaihtoehdon Cypressin. Cypress on moderneille verkkosivuille suunniteltu testiautomaatiotyökalu. Cypress on erikoistunut JavaScript -pohjaisten kehysten kanssa käytettäväksi yhtä aikaa. Näitä ovat esimerkiksi React, Angular ja Vue. Cypressissä käytettävä ohjelmointikieli on JavaScript. Cypress tukee ainoastaan Chromium-pohjaisia selaimia, kuten Chromea ja Edgeä. (Ashiq 2022.)

Tein vaatimuksieni pohjalta taulukon 2, josta näkee vertailun kohteena olevien testaustyökalujen sopivuuden näihin vaatimuksiini. Kuvaan on merkitty ruksilla kohdat, jotka eivät vastanneet vaatimuksiin ja valintamerkillä kohdat, jotka vastasivat vaatimuksiin.

TAULUKKO 2. Testaustyökalujen vertailu vaatimuksiin nähden.

Testaustyökalu	**Helppo käyttöönotto**	**Ilmainen**	**Testien tekemisen helppous**
Selenium	✗	✓	✓
Robot Framework	✓	✓	✓
Cypress	✓	✓	✗

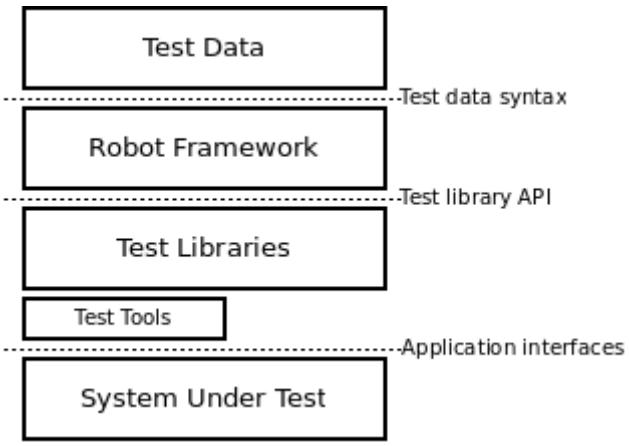
Lukemani perusteella Seleniumin IDE:n käyttöönotto ei vastannut vaatimuksiani, koska sen käyttöönotto vaati perehtymistä erilliseen käyttöliittymään. Myöskään Cypress ei vastannut näihin vaatimuksiin, koska koodisyntaksi vaikutti hieman monimutkaiselta ja olisi vaatinut enemmän asiaan perehtymistä. Päädyin siis lopulta valitsemaan työkalukseni Robot Frameworkin, koska se oli

minulle entuudestaan tuttu ja halusin toteuttaa testit Python kielellä. Lisäksi sen helposti ymmärrettävä ja opittava syntaksi vaikutti ratkaisevasti valintaani. Selenium kuitenkin osittain on mukana tässä työssä, koska Robot Framework voi hyödyntää sen kirjastoja vuorovaikutukseen verkkosivun elementtien kanssa.

3.4.2 Robot Framework

Pekka Klärckin (2006) diplomityötä ”Data-Driven and Keyword-Driven Test Automation Frameworks” varten kehitetty avainsanakäyttöinen testiautomaatiokehyksen prototyyppi osoittautui menestyksekkääksi ja Klärck jatkoikin myöhemmin kehyksen kehittämistä asiakasprojektissa. Vuonna 2008 kehys avattiin avoimelle lähdekoodille ja tästä syntyi Robot Framework. (Klärck 2009.) Robot Frameworkistä on tämän jälkeen tullut yksi maailman suosituimmista ohjelmistotestauskehyksistä ja sille on muodostunut laaja yhteisö (Robot Framework Foundation).

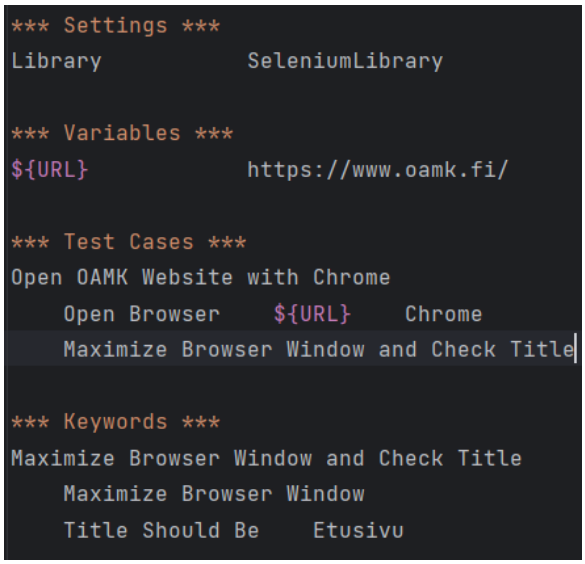
Kuvasta 5 näkyy että Robot Frameworkissä on modulaarinen arkkitehtuuri. Robot Framework prosessoi käynnistyessään testaajan kirjoittaman testidatan, suorittaa testitapaukset ja luo lokit ja raportit. Kehyksen ydin ei ole tietoinen testattavasta järjestelmästä, vaan kommunikointi tapahtuu testauskirjastojen kautta. Kirjastot voivat joko käyttää testattavan sovelluksen rajapintaa suoraan tai hyödyntää alemman tason työkaluja ajureina. (Robot Framework Foundation.)



KUVA 5. Robot Frameworkin arkkitehtuuri. (Robot Framework Foundation.)

Robot Framework käyttää taulukkotyylistä syntaksia testidatassaan. Tämä tarkoittaa sitä, että avainsanat pystytään kirjoittamaan taulukkomaisesti allekkain ja argumenttien välillä on vähintään neljä välilyöntiä, mikä tekee koodin lukemisesta helpompaa ja rakenteesta selkeämpää. Robot

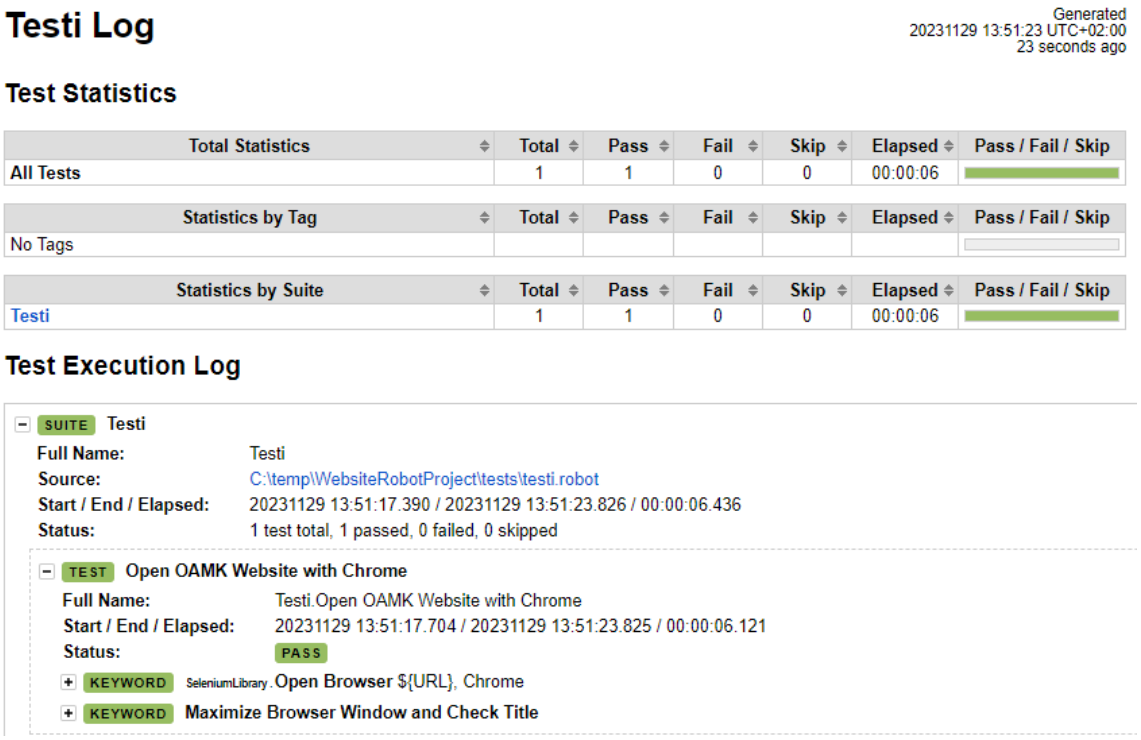
Frameworkin testidatan syntaksia tarkastellessa (kuva 6) nähdään, minkälainen rakenne testidatalla on. Ensimmäisenä ovat lohkot, jotka voidaan tunnistaa kolmesta tähtimäisestä asteriskimerkistä lohkon nimen molemmilla puolilla. Testidata jaetaan omiin lohkoihinsa ja jokaisella niistä on oma tarkoituksensa. Kuvan 6 esimerkissä ovat Settings-, Variables-, Test Cases- ja Keywords -lohkot. Settings-lohkossa määritetään käytettävät kirjastot, resurssitiedostot tai muuttujatiedostot. Variables-lohkossa määritetään taas muuttujat, joita voidaan käyttää muualla testidatassa. Test Cases -lohkoon kirjoitetaan varsinaiset testitapaukset, jotka luodaan avainsanoja hyödyntämällä. Keywords-lohkossa ovat käyttäjän itse luomat avainsanat ja ne ovat ikään kuin funktioita, joita pystytään hyödyntämään muualla testidatassa. (Robot Framework Foundation.) Kuvassa 6 on esimerkkitesti, josta myös näkee millaista testidataa testeissä käytetään.



KUVA 6. Esimerkki Robot Frameworkin testidatasta.

Esimerkkitestissä alustetaan Robot Framework käyttämään Selenium-kirjastoa ja muuttujiin on asetettu URL-osoite. Tämän jälkeen suoritetaan **Open OAMK Website with Chrome** -testitapaus, jossa käytetään kahta avainsanaa **Open Browser** ja itse luotu **Maximize Browser Windows and Check Title**. Avainsana **Open Browser** vaatii itselleen argumenteiksi testattavan verkkosivuston osoitteen, joka on määritetty muuttujissa, ja avattavan verkkoselaimen. Keywords-lohkoon on määritetty oma avainsana, joka avaa selaimen koko ruudun kokoiseksi ja tarkistaa, onko verkkosivun otsikko ”Etusivu”. Kokonaisuudessaan tämä testi siis avaa oamk.fi-verkkosivun Chrome-selaimella, suurentaa sen ikkunan koko ruudun kokoiseksi ja tarkistaa, vastaako sivuston otsikko sanaa ”Etusivu”. Testitapaus joko onnistuu tai epäonnistuu tämän viimeisen tarkistuksen myötä.

Kuten jo aiemmin on mainittu, Robot Framework tarjoaa myös oletuksena käyttäjälleen lokeja ja raportteja testien tuloksista. Kuvasta 7 näkyy aiemman esimerkkitestin tuottaman lokitiedoston sisältö.



KUVA 7. Esimerkki Robot Frameworkin tuottaman lokitiedoston sisällöstä.

Kun lukuisia testejä suoritetaan yhtä aikaa, lokit ja raportit tarjoavat käyttäjälleen helposti luettavat tulokset testien onnistumisesta. Kaikkien testien tuloksia voidaan tarkastella erikseen ja esimerkiksi epäonnistuneissa testeissä nähdään, missä vaiheessa testi epäonnistui ja mistä syystä. Lokien ja raporttien erona on se, että lokit ovat tarkoitettu enemmän testaajille itselleen, koska niistä näkee tarkemmat tiedot testeistä. Raportit ovat tarkoitettu myös muiden nähtäviksi, joilla ei ole tarkkaa tietoa testien toteutuksesta.

3.4.3 SeleniumLibrary testikirjasto

SeleniumLibrary on verkkosivujen testaamiseen luotu kirjasto, jota voidaan hyödyntää eri testauskehyksissä, kuten Robot Frameworkissä. Kirjasto tarjoaa testaajalle käytettäväksi valmiita avainsanoja, joita käytetään verkkosivujen automaatiotestauksessa tai automatisoinnissa.

SeleniumLibrary käyttää Selenium WebDriver-moduuleja hallinnoimaan verkkoselaimia. SeleniumLibraryn yksi tärkeimmistä ominaisuuksista ovat paikantimet (locator). Verkkosivuilta löytyviin elementteihin pääsee käsiksi näiden paikantimien avulla. Tietyt avainsanat vaativat argumenteikseen paikantimen, jossa määritellään elementin sijainti.

3.5 Testien suunnittelu

Tämän luvun lähtökohtana on hyvä tietää, että testauksen kohteeksi valittu sivusto ei ollut minulle entuudestaan tuttu testauksen kohde. En ole myöskään saanut aihetta opinnäytetyölle OAMK:lta tai ollut heillä töissä. Minulla ei siis ollut mitään valmiiksi laadittuja testisuunnitelmia tai dokumentteja, mistä selviäisi mitä kuuluu testata ja mitkä ovat eri toiminnallisuuksien halutut lopputulokset.

Testausta suunnitellessani pyrin etsimään verkosta käyttöliittymien testitapauksia ja näiden pohjalta valitsin tähän työhön sopivat testit. Useita aiheeseen liittyviä sivustoja selatessani huomasin, että ei ole mitään yhtä ja oikeaa tapaa suunnitella automaatiotestausta sovelluksille. Sen sijaan verkkosivun käyttöliittymien ominaisuuksia voidaan testata eri automaatiotestauksen tyyppien näkökulmasta, jotka esiteltiin aiemmin luvussa 2.3. Esimerkiksi jokin verkkosivun ominaisuus voidaan testata hyväksymistestauksen näkökulmasta eli testataan sen toiminnallisuutta ja samaa ominaisuutta voidaan testata myös graafisen käyttöliittymätestauksen näkökulmasta, mikä tarkoittaa esimerkiksi ominaisuudessa olevien painikkeiden oikeanlaisen käyttäytymisen varmistamista.

Oamk.fi sivuston etusivulla on paljon testattavia elementtejä ja alkuun täytyikin miettiä, mitä testejä voitaisiin automatisoida. Etusivulta löytyy paljon sisältöä, joka muuttuu jatkuvasti, kuten esimerkiksi ammattikorkeakoulun tiedotukset, blogit, haastattelut, artikkelit ja tapahtumat. Näiden sisältöjen testaamista on hankala automatisoida, jos esimerkiksi halutaan testata linkkien toimivuus. Sivustolta löytyy myös paljon pysyviä elementtejä, kuten sivuston yläosasta hakutoiminto, pudotusvalikot, OAMK-logo ja sivuston alareunasta yhteystietoja, linkkejä OAMK-sometileihin, uutiskirjeen tilaaminen ja lisätietoja. Koska tämän työn aiheena on verkkosivun käyttöliittymän automaatiotestauksen käyttöönotto, en tule luomaan automaatiotestausta koko sivuston sisällölle vaan halusin ottaa kolme esimerkkiominaisuutta testauksen alkuun pääsemiseksi. Päädyin lopulta siis valitsemaan testauksen kohteeksi hakutoiminnon, pudotusvalikot ja chatbotin.

3.6 Testitapausten valinta

Testaukseen valituilla toiminnoilla on jokaisella omat eri käyttötarkoituksensa ja tämän myötä jokaiselle pitää räätälöidä myös omat testitapauksensa näiden tarkoitusten pohjalta. Lähtökohtana testitapausten valinnalle oli edellisessä luvussa mainitut automaatiotestauksen tyypit. Tämän lisäksi Guru 99 -verkkosivuston artikkelista ”GUI Testing – UI Test Cases” löytyi esimerkkejä verkkosivun käyttöliittymään sopivista testitapauksista ja poimin tältä sivulta omiin testeihini muun muassa toimintojen perustoiminnallisuuksien testaamisen, elementtien näkyvyyden eri resoluutiolla ja tekstin syöttämisen testaamisen (Hamilton 2023c). Halusin ottaa tähän työhön jokaiselle ominaisuudelle neljä esimerkki testitapausta. Koska testien suunnittelussa ei noudateta mitään tarkkaa kaavaa, valitsin testitapaukset oman harkinnan mukaan siitä, mistä näiden ominaisuuksien testaaminen voisi loogisesti alkaa. Laadin alle taulukot jokaisesta toiminnallisuudesta ja niille suunnatuista testitapauksista.

Testattavan verkkosivun oikeassa yläreunassa on hakutoiminto, jolle aion luoda ensimmäiset testitapaukset. Hakutoiminto toimii siten, että ensin klikataan Haku-elementtiä. Tämän jälkeen aukeaa pieni ikkuna, mihin voi syöttää hakusanan ja painaa suurennuslasin kuvaketta haun suorittamiseksi. Taulukko 3 näyttää tälle toiminnolle suunnitellut automaatiotestit.

TAULUKKO 3. Hakutoiminnon testitapaukset

Testitapaus	Testitapauksen kuvaus
Hakutoiminnon perustoiminnallisuus	Testataan hakutoimintoa sanalla, joka tuottaa vähintään yhden osuman.
Haku tyhjällä hakusanalla	Testataan, miten hakutoiminto käsittelee tyhjäksi jätetyn hakukentän.
Haku erikoismerkeillä ja symboleilla	Testataan, miten hakutoiminto käsittelee erikoismerkkejä ja symboleja.
Normaali haku ilman tuloksia	Testataan hakutoimintoa normaalilla hakusanalla, joka ei tuota yhtään tulosta.

oamk.fi etusivulta löytyy neljä pudotusvalikkoa: Koulutus, Tutkimus ja Kehitys, Työelämälle ja Info. Valikot avautuvat esiin, kun hiiren kohdistimen laittaa pienen oranssin nuolen päälle. Teen tässä

työssä automaatio testit yhteen pudotusvalikkoon ”Koulutus”. Taulukko 4 näyttää pudotusvalikon testitapaukset.

TAULUKKO 4. Pudotusvalikon testitapaukset

Testitapaus	Testitapauksen kuvaus
Pudotusvalikko oletuksena piilossa	Pudotusvalikko ei saa olla verkkosivulle mennessä jo valmiiksi auki.
Pudotusvalikon aukaisu	Pudotusvalikon pitäisi aueta, kun kohdistin on nuolen päällä.
Pudotusvalikon linkkien toimivuus	Testataan muutaman linkin toimivuus. Klikkaamalla linkkiä pitäisi päästä oikealle sivulle.
Pudotusvalikon sopeutuminen pieneen ruutuun	Valikkojen pitäisi piiloutua, kun selaimen resoluutio menee tarpeeksi pieneksi.

Verkkosivun oikeaan alareunaan ilmestyy puhekuplan kuva, jota painamalla aukeaa chatbot-ikkuna. Chatbot antaa valmiita hakusanoja, jonka avulla käyttäjä voi etsiä tietoa, mitä tuli sivustolta hakemaan. Taulukko 5 näyttää mitä tästä toiminnallisuudesta testataan.

TAULUKKO 5. Chatbotin testitapaukset

Testitapaus	Testitapauksen kuvaus
Chatbotin saatavuus	Chatbotin kuvake pitäisi ilmestyä verkkosivulle.
Chatbotin avaaminen	Chatbot pitäisi saada avattua normaalisti.
Chatbotin hakutesti	Haetaan jotain tietoa chatbotin avulla.
Chatbotin jätä kysymys -ominaisuuden ei pitäisi hyväksyä tyhjää kysymystä	Testataan hyväksyykö chatbot tyhjän kysymyksen.

Kaikkien ominaisuuksien testitapauksista löytyy perustoiminnallisuuteen liittyvää funktionaalista testausta ja tämän lisäksi vaihtelevasti erilaisten graafisen käyttöliittymän elementtien testaamista. Koska kaikki testitapaukset on automatisoitu, se tarkoittaa, että ne ovat myös tyypiltään regressiotestausta.

4 TESTAUKSEN TOTEUTUS

4.1 Testaustyökalun asennus

Robot Framework on toteutettu Pythonilla, joten asennuksen esivaatimuksena on asentaa Python tai PyPy testauksen suorittavalle laitteelle. Testien ajaminen tapahtui Windows-käyttöjärjestelmän päällä, joten Python ladattiin heidän omilta verkkosivuiltaan python.org. Uusin Robot Framework vaatii vähintään Python-version 3.6. (Robot Framework Foundation.) Tässä työssä käytin versiota 3.10.5, joka oli jo valmiiksi asennettuna tietokoneelleni. Kuvassa 8 näkyy tietokoneelleni tehty testaustyökalun asennus ja Pythonin ja Robot Frameworkin versioiden tarkistus komentoriviä käyttäen.

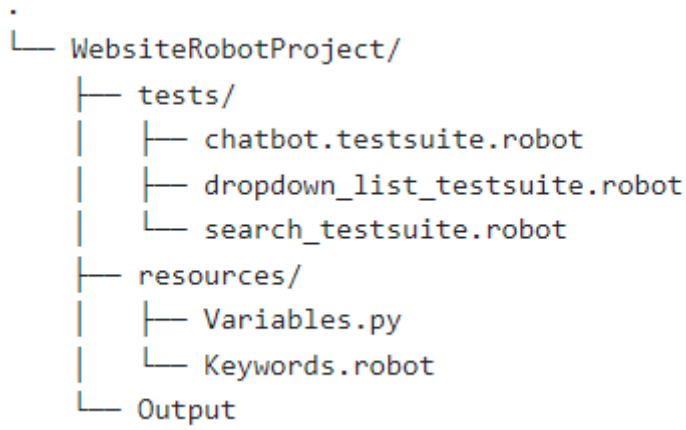
```
c:\>pip install robotframework
Collecting robotframework
  Using cached robotframework-6.1.1-py3-none-any.whl (699 kB)
Installing collected packages: robotframework
Successfully installed robotframework-6.1.1

c:\>python --version
Python 3.10.5

c:\>robot --version
Robot Framework 6.1.1 (Python 3.10.5 on win32)
```

KUVA 8. Robot Frameworkin asennus ja version tarkistus.

Asensin Robot Frameworkin Windowsin komentorivin kautta käyttämällä **pip**-komentoa. Pip on Pythonilla luotu paketinhallintajärjestelmä, joka on yhteydessä verkossa olevaan Python Package Index -säilytyspaikkaan, josta käyttäjä voi halutessaan asentaa, poistaa tai hallinnoida asennuspaketteja (Philipp).

4.2 Testien kirjoittaminen

Testien kirjoittamisessa lähdin liikkeelle kansiorakenteen luomisesta projektia varten (kuva 9). Testitiedostot (test suite) sijoitettiin omaan kansioonsa ja globaalit muuttujat ja avainsanat omaansa. Lisäksi luotiin vielä output-kansio, mihin testien tulosraportit ja lokit menevät suoraan testien suorituksen jälkeen.



KUVA 9. Projektin kansiorakenne kuvattuna ASCII-tyylillä.

Kuvasta näkee, että globaalit muuttujat ja avainsanat löytyvät resources-kansion **Variables.py** ja **Keywords.robot**-tiedostoista. Globaalit muuttujat ovat projektissa yleisesti käytettyjä muuttujia, mitä voidaan hyödyntää kaikissa testitiedostoissa. Kuvassa 10 on **Variables.py**-tiedoston sisältö, josta löytyy testattavan verkkosivun osoite ja testeissä käytettävä selain. Testitiedostot olen sijoittanut tests-kansioon, josta löytyy jokaiselle suunnitelman mukaiselle ominaisuudelle omat testitiedostonsa.

```
URL = "https://www.oamk.fi"
BROWSER = "Chrome"
```

*KUVA 10. **Variables.py**-tiedostosta löytyvät muuttujat*

Yleisesti käytettyjä avainsanoja tehtiin **Keywords.robot**-tiedostoon yksi kappale **Open the website and maximize window**. Tämän avainsanan tarkoituksena on tehdä alustus jokaiselle testitapaukselle avaamalla ensin Chrome-selain ja suurentaa ikkuna koko ruudun kokoiseksi. Huomasin lisäksi testejä ajaessani, että komentoriville tuli ylimääräisiä virheilmoituksia ja tämän vuoksi **Open Browser** -avainsanan perästä löytyy myös asetuksia näiden virheilmoitusten piilottamiseksi. Kuvassa 11 **Keywords.robot**-tiedoston sisältö.

```
*** Keywords ***

Open the website and maximize window
    Open Browser    ${URL}    ${BROWSER}    options=add_experimental_option("excludeSwitches", ["enable-logging"])
    Maximize Browser Window
```

*KUVA 11. **Keywords.robot**-tiedoston sisältämä avainsana.*

Jokaisen testitiedoston alussa on Settings-lohko, missä alustetaan SeleniumLibrary kirjasto, **Keywords.robot**-resurssi ja **Variables.py**-tiedosto käyttöön. Kuvasta 12 näkee, miten tämä alustus on toteutettu.

```
*** Settings ***
Library     SeleniumLibrary
Variables     C:/temp/WebsiteRobotProject/resources/Variables.py
Resource    C:/temp/WebsiteRobotProject/resources/Keywords.robot
```

KUVA 12. Testitiedostojen asetukset.

Ensimmäinen testitiedosto **search_testsuite.robot** sisältää testitapaukset oamk.fi sivun oikeasta yläkulmasta löytyvälle hakutoiminnolle. Hakutoiminto toimii siten, että ensin klikataan haku painiketta, tämän jälkeen kirjoitetaan hakusana kenttään ja haun suorittamiseksi klikataan suurennuslasin kuvaketta. Haun lopuksi aukeaa erillisen sivu hakutuloksille. Testejä kirjoittaessa, jokainen vaihe tuli ottaa huomioon, jotta testiautomaatio saatiin toimimaan onnistuneesti. Kuva 13 näyttää **search_testsuite.robot** sisällön.

```
*** Settings ***
Library     SeleniumLibrary
Variables     C:/temp/WebsiteRobotProject/resources/Variables.py
Resource     C:/temp/WebsiteRobotProject/resources/Keywords.robot

*** Variables ***
${HAKU_BUTTON}       xpath://a[@href='javascript://' and contains(@onclick, "$('#elahakualue').fadeToggle()")]
${HAE_BUTTON}        xpath://button[@title='Hae' and @value='HAE']
${INPUT_FIELD}       xpath://input[@id='hakusana' and @class='searchtext autosuggest ui-autocomplete-input']
${SEARCH_RESULT}     xpath://*[@id="ela_hakutulokset"]/div[1]

*** Test Cases ***
Test search with atleast one hit
    Open the website and maximize window
    Click Element     ${HAKU_BUTTON}
    Input Text     ${INPUT_FIELD}     opinnäytetyö
    Click Element     ${HAE_BUTTON}
    Element Should Contain     ${SEARCH_RESULT}     tulosta
    Element Should Not Contain     ${SEARCH_RESULT}     0 tulosta
    Close Browser

Test search with zero hits
Test search with special characters and symbols
Test with empty input field
```

KUVA 13. ***search_testsuite.robot****-tiedoston testidata.*

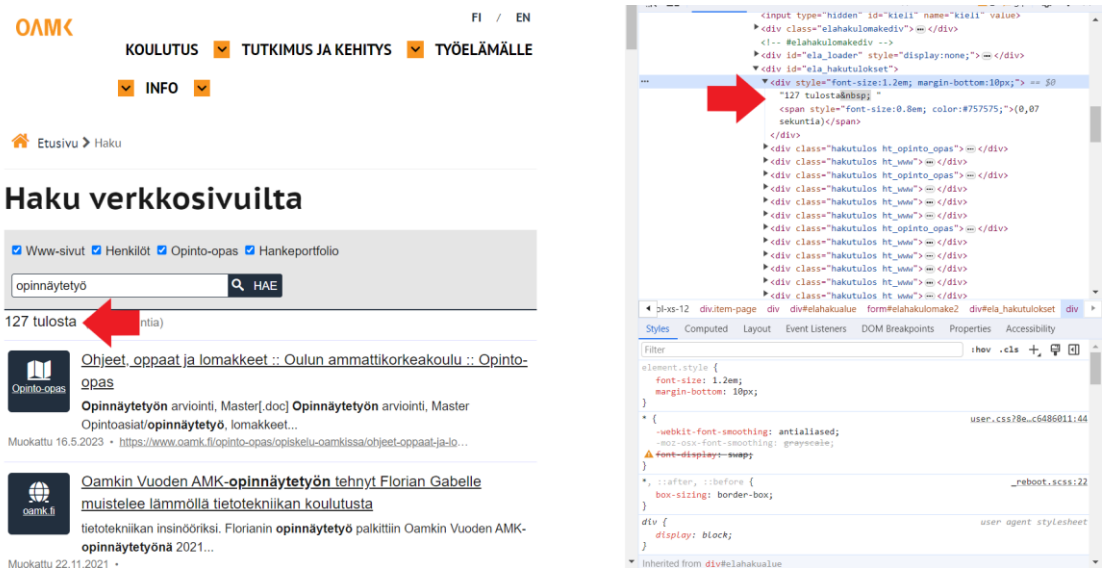
Kuvasta löytyvät asetuksien lisäksi testitiedostossa käytetyt muuttujat ja testitapaukset. Testitiedoston muuttujat (Variables) viittaavat kaikki tiettyihin oamk.fi sivustolta löytyviin elementteihin. Esimerkiksi **HAKU_BUTTON**-muuttuja viittaa etusivulta löytyvään ”haku”-painikkeeseen. Nämä viittaukset on tehty xpath-paikantimilla ja **HAKU_BUTTON**-muuttujan tapauksessa paikannin etsii sivuston lähdekoodista elementin, josta löytyy attribuutti nimeltä **href=javascript://** ja joka sisältää myös **onclick**-funktion **$(’#elahakualue’).fadetoggle**(). Paikantimissa on tärkeää yksilöidä elementit siten, että niiden paikannus kohdistuu vain yhteen elementtiin. Tämä siksi, että sivuston lähdekoodista saattaa löytyä useampi elementti samalla nimellä.

Testitapauksia tehtiin neljä kappaletta ja kuvassa 13 näkyy esimerkkinä ensimmäinen testitapaus **Test search with atleast one hit**. Muut testitapaukset ovat myös tiedostossa, mutta kuvassa pienennettynä. Ensimmäinen testitapaus on tehty suunnitelman pohjalta, jonka vaatimuksena on kokeilla haku ominaisuutta hakusanalla, joka tuottaa useamman osuman. Lopputuloksena hakukoneen pitäisi löytää ainakin yksi osuma tälle hakusanalle. Testitapauksen kulku menee seuraavasti:

1. Avaa verkkosivu oamk.fi Chrome-selaimella ja suurenna selainikkuna koko ruudun kokoiseksi.
2. Klikkaa haku painiketta.
3. Syötä haun tekstikenttään hakusana ”Opinnäytetyö”.
4. Klikkaa suurennuslasin kuvaketta, joka avaa uuden ”hakutulos”-sivun.

5. Hakutuloksen ilmoittamasta elementistä pitäisi löytyä ”tulosta” sana.
6. Hakutuloksen ilmoittamasta elementistä ei saisi löytyä ”0 tulosta” sana.
7. Sulje selain testin päätyttyä.

Hakutulokset näyttävältä sivulta löytyy elementti, joka ilmoittaa hakutulosten määrän. Esimerkiksi ”Opinnäytetyö”-hakusana asettaa elementin näyttämään ”127 tulosta”. Testien tarkistus hakee hakutuloksen tästä elementistä ja tekee sen pohjalta vertailun haluttuun tulokseen. Haku ominaisuus toimii myös siten, että jos hakukenttään ei syötä mitään ja suorittaa haun, tämä hakutulokset näyttävä elementti ei näy ollenkaan, joten ensimmäiseen tarkistukseen riittää ainoastaan ”tulosta” sanan löytyminen. Kuvasta 14 löytyy vasemmalta elementti, joka näyttää hakutulosten määrän. Kun tätä elementtiä klikkaa hiiren oikeanpuoleisella painikkeella ja valitsee **inspect**, saa selaimeen auki sivun, joka näyttää suoraan elementin sijainnin lähdekoodissa (kuvassa oikealla).



KUVA 14. Hakutulosten näyttävä elementti ja sen sijainti lähdekoodissa.

Muissa testitapauksissa automaattisten testien kulku etenee samalla tavalla, mutta ainoastaan hakusana muuttuu. **Test search with zero hits** ja **Test search with special characters and symbols** -testitapauksen loppuvertailu haluaa, että hakutulokset ilmoittavasta elementistä löytyy teksti ”0 tulosta”. Kun hakukonetta käytetään erikoismerkeillä tai symboleilla, hakukoneen edelleen oletetaan toimivan normaalisti eli hakusivun pitäisi vain näyttää 0 tulosta. Viimeisessä testissä **Test with empty input field** oletetaan, että tätä hakutulosten näyttävää elementtiä ei ole lainkaan näkyvissä.

Toinen testitiedosto **dropdown_list_testuite.robot** sisältää testitapaukset oamk.fi-etusivulta löytyvälle Koulutus-pudotusvalikolle. Testitapaukset kohdistuvat oranssiin nuoleen, mistä pudotusvalikko avautuu, ja valikon sisältä löytyvään linkkiin. Kuvassa 15 on tämän testitiedoston sisältö pois lukien **Settings**-lohko, mikä on kaikissa testitiedostoissa sama.

```
*** Variables ***
${DROPDOWN_LIST}      xpath=//*[@id="dj-megamenu90"]/li[2]/div
${DROPDOWN_ARROW}     xpath=//*[@id="dj-megamenu90"]/li[2]
${TEST_LINK}          xpath=//*[@id="dj-megamenu90"]/li[2]/div/div/div[1]/ul/li[1]


*** Test Cases ***
Dropdown list initial state should be hidden
    Open the website and maximize window
    Element Should Not Be Visible    ${DROPDOWN_LIST}
    Close Browser

Dropdown list should open when hovering over
    Open the website and maximize window
    Mouse Over    ${DROPDOWN_ARROW}
    Element Should Be Visible    ${DROPDOWN_LIST}
    Close Browser

Test dropdown menu links
    Open the website and maximize window
    Mouse Over    ${DROPDOWN_ARROW}
    Click Element    ${TEST_LINK}
    Title Should Be    Tutkintoon johtava koulutus
    Close Browser

Dropdown menu should be hidden when browser window is small
    Open Browser    ${URL}    ${BROWSER}    options=add_experimental_option("excludeSwitches", ["enable-logging"])
    Set Window Size    600    600
    Element Should Not Be Visible    ${DROPDOWN_ARROW}
    Close Browser
```

*Kuva 15. **dropdown_list_testsuite.robot**-tiedoston sisältö*

Pudotusvalikon testitiedostosta löytyy myös muuttujat paikantimille. Tässä tapauksessa paikantimet oli määritetty oranssille nuolelle, avatulle pudotusvalikolle ja linkille, josta pääsee ”Tutkintoon johtava koulutus” -sivulle.

Ensimmäisessä testitapauksessa **Dropdown list initial state should be hidden** tarkistetaan, että pudotusvalikko ei ole valmiiksi avattuna, kun mennään oamk.fi verkkosivulle. Tämä tarkistus tapahtuu suoraan verkkosivulle mentäessä avainsanalla **Element Should Not Be Visible**, jonka perään on laitettu avatun pudotusvalikon elementin paikannin. Tämä avainsana tarkistaa siis, että tämä elementti ei ole nähtävissä. Seuraava testitapaus tarkistaa, että pudotusvalikko aukeaa, kun hiiren kursorin laittaa oranssin nuolen päälle. **Mouse Over** avainsanalla pystytään testitapauksessa laittamaan kursorin jonkin elementin päälle. Kolmannessa testissä **Test dropdown menu links** testataan linkkiä, josta pääsee ”Tutkintoon johtava koulutus” sivulle. Lopun tarkistus

tarkastaa, että sivun otsikko on ”Tutkintoon johtava koulutus”. Viimeinen testi on siitä hieman poikkeava, että siinä ei tehdä samanlaista alustusta, kuin kaikissa muissa testeissä vaan sen sijaan, että näyttö laitettaisiin koko ruudun kokoiseksi, se asetetaankin 600 x 600 resoluutioon. Tämä testi siis testaa, että kun näyttö pienennetään tarpeeksi pieneksi, pudotusvalikot menevät piiloon. Lopun tarkistus tarkastaa, että oranssi nuoli ei ole näkyvissä.

Kolmas testitiedosto **chatbot_testsuite.robot** sisältää neljä testitapausta oamk.fi sivulta löytyvälle chatbotille. Tämä chatbot avautuu omana ikkunanaan sivun oikeaan alareunaan. Kuvassa 16 näkyy osa tämän tiedoston sisällöstä.

```
*** Variables ***
${CHATBOT_BUTTON}          xpath://button[contains(@class, 'ld-launch-btn')]
${ACCEPT_COOKIES}          xpath://button[contains(@onclick, "document.cookie='oamkcookiet=kaikki")]
${CHATBOT_OPENED}          xpath://div[@class='chat' and @id='root']
${CHATBOT_BUTTON_FRAME}    xpath://iframe[@title='Leadoo chatbotin käynnistin']
${CHATBOT_WINDOW_FRAME}    xpath://iframe[@class='ld-chat-bot ld-chat-window' and @title='Leadoon chatbotti-ikkuna']

*** Test Cases ***
Chatbot icon should be visible on the website
Chatbot should open
Test chatbot for finding information
    Open the website and maximize window
    Set Selenium Implicit Wait    5
    Click Element    ${ACCEPT_COOKIES}
    Select Frame    ${CHATBOT_BUTTON_FRAME}
    Click Element    ${CHATBOT_BUTTON}
    Unselect Frame
    Select Frame    ${CHATBOT_WINDOW_FRAME}
    Find opinto-opas
    Switch to new window and check title
    Close Browser
```

*Kuva 16. **Chatbot_testsuite.robot**-tiedoston osittainen sisältö.*

Chatbotin automaatiotestauksessa oman haasteensa toi se, että tässä ominaisuudessa on hyödynnetty iframeja. Nämä framet ovat siis omia ”dokumenttejaan”, jotka aukeavat pääsivun HTML-dokumentin päälle omana ikkunanaan. Testitapauksissa pitää aina määrittää, mihin frameen komennot kohdistetaan ja aina framea vaihtaessa pitää muistaa ajaa avainsana **Unselect Frame**, jotta päästään vaihtamaan framea. Lisäksi chatbotilla kestää jonkin aikaa vastata kysymyksiin, jonka vuoksi testeissä pitää joissain kohdissa käyttää **Sleep**-avainsanaa, joka pysäyttää Robot Frameworkin suoritukset käyttäjän määrittämäksi ajaksi.

Kuvan **Test chatbot for finding information** on otettu tähän esimerkkinä. Tämä testitapaus suoritetaan seuraavasti:

1. Avaa verkkosivu oamk.fi ja suurenna ikkuna koko ruudun kokoiseksi.
2. Aseta Selenium odottamaan maksimissaan 5 sekuntia elementtien ilmestymistä.
3. Klikkaa elementtiä, joka hyväksyy sivun evästeet (tämä evästepalkki estää chatbotin näkymisen).
4. Valitse frame, jossa on chatbotin aukaisunappi.
5. Klikkaa chatbotin aukaisunappia.
6. Poistu framesta.
7. Vaihda frameksi avattu chat-ikkuna.
8. Suorita avainsana **Find opinto-opas**.
9. Suorita avainsana **Switch to new window and check title**. Tässä tarkistetaan, että uuden avautuvan verkkosivun otsikko vastaa testissä määritettyä otsikkoa.
10. Sulje selain.

Tässä suorituksessa oli kaksi omaa avainsanaa **Find opinto-opas** ja **Switch to new window and check title**. Nämä avainsanat luotiin siksi, että saataisiin testitapaus lyhyempään ja selkeämpään muotoon. Kuva 17 näyttää näiden avainsanojen sisällön.

```
*** Keywords ***
Find opinto-opas
    Click Element    xpath://button[contains(@class, 'ld-button') and contains(@class, 'shadow') and .//strong[contains(text(), 'Opiskelijalle')]]
    sleep    4
    Click Element    xpath://button[@class='ld-button shadow' and div[contains(text(), 'Opinto-opas')]]
    sleep    4
    Click Element    xpath://button[@class='ld-button shadow' and div[contains(text(), 'Siirry opinto-oppaaseen')]]
    sleep    4

Switch to new window and check title
    ${window_handles} =    Get Window Handles
    Switch Window    ${window_handles}[1]
    Title Should Be    Oulun ammattikorkeakoulu :: Opinto-opas
```

*Kuva 17. **Test chatbot for finding information** -testitapauksessa käytetyt avainsanat.*

Ensimmäinen **Find opinto-opas** -avainsana käy keskustelua chatbotin kanssa klikkailemalla vastauksia chatbotin kysymyksiin. Tässä tapauksessa vastausten perusteella halutaan päästä opinto-oppaasta kertovalle sivulle. Toinen avainsana **Switch to new window and check title** tarvittiin siksi, koska opinto-opassivu aukeaa automaattisesti toiseen ikkunaan linkistä. Tämä avainsana vaihtaa aktiivisen ikkunan toiseen ikkunaan ja tarkistaa, vastaako tämän uuden sivun otsikko ”Oulun ammattikorkeakoulu :: Opinto-opas” tekstiä.

Tämän esimerkki testitapauksen lisäksi oli kaksi testitapausta, jotka liittyivät chatbotin perustoimivuuteen. Ensimmäisessä testitapauksessa tarkistettiin, että tuleeko chatbot-elementti näkyviin

etusivulle, ja toisessa testitapauksessa testattiin, saadaanko chatbot auki. Viimeisessä testitapauksessa testattiin chatbotin ”Jätä kysymys” ominaisuutta, jossa testitapaus yrittää jättää tyhjän kysymyksen ja tarkistaa, että verkkosivu antaa virheilmoituksen tyhjästä kentästä.

4.3 Testien ajaminen

Ajoin tässä työssä kaikki testitiedostot Windows-komentorivin kautta. Komentoriviin syötetään esimerkiksi komento **Robot search_testsuite.robot**. Komennon jälkeen Robot Framework suorittaa kaikki tiedoston sisältämät testitapaukset. Testien suorittava komento tulee ajaa siitä kansiosta, missä tämä testitiedosto sijaitsee tai määrittää tarkempi sijainti, mistä tiedosto löytyy. Kuvassa 18 on esimerkki testin suorituksesta.

```
c:\temp\WebsiteRobotProject\tests>Robot -d C:/temp/WebsiteRobotProject/output search_testsuite.robot
==============================================================================
Search Testsuite
==============================================================================
Test search with atleast one hit                                      | PASS |
------------------------------------------------------------------------------
Test search with zero hits                                            | PASS |
------------------------------------------------------------------------------
Test search with special characters and symbols                       | PASS |
------------------------------------------------------------------------------
Test with empty input field                                           | PASS |
------------------------------------------------------------------------------
Search Testsuite                                                      | PASS |
4 tests, 4 passed, 0 failed
==============================================================================
Output:  C:\temp\WebsiteRobotProject\output\output.xml
Log:     C:\temp\WebsiteRobotProject\output\log.html
Report:  C:\temp\WebsiteRobotProject\output\report.html
```

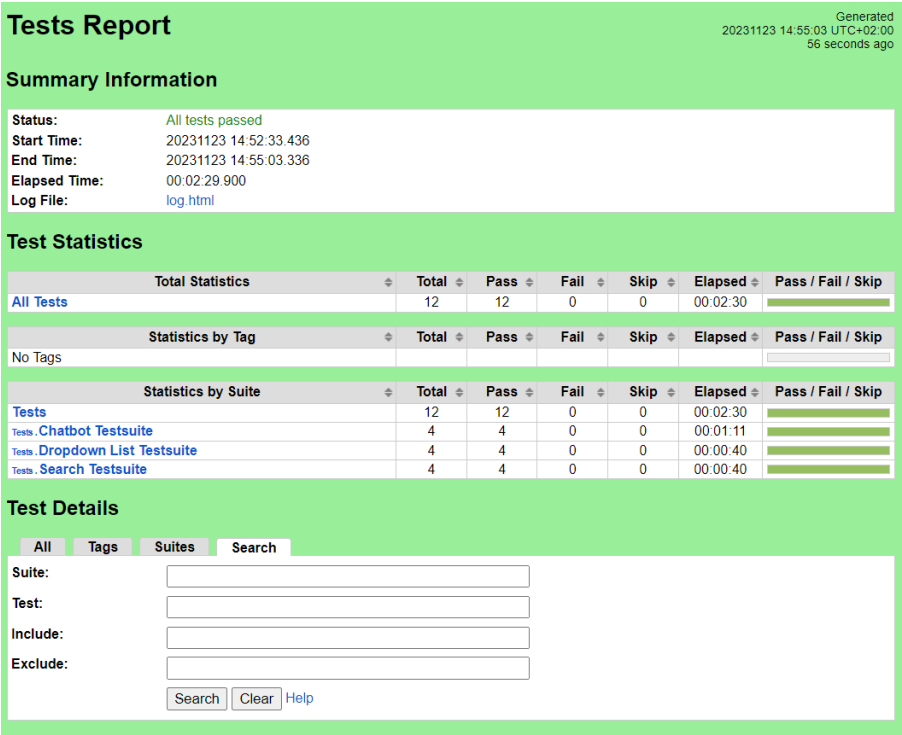
KUVA 18. Testin ajaminen komentorivin kautta.

Lisäsin ajokomentojen mukaan **-d C:/temp/WebsiteRobotProject/output** -komennon. Tämä komento määrittää kansion, mihin testituloksien raportit ja lokit tallennetaan testin suorituksen jälkeen. Kuvasta näkee myös, millä tavalla komentorivi ilmoittaa testien suorituksen onnistumisesta. Jos testi suoriutuu onnistuneesti, lopputulokseksi tulee ilmoitus **PASS**, ja jos taas testi epäonnistuu tai tulee muuta ongelmaa, testi antaa tulokseksi **FAIL**.

4.4 Testien raportointi

Testien suorituksen jälkeen testien tuloksia voi tarkastella Robot Frameworkin generoimasta raportista tai lokista. Raportti tallentuu **Report.html**-tiedostoon ja sen saa avattua selaimella. Raportti

antaa tiivistetyn kuvan testien suoritusten tuloksista. Kuvassa 19 on raportti kaikkien testitiedostojen suorituksesta.



Tests Report

Generated
20231123 14:55:03 UTC+02:00
56 seconds ago

Summary Information

Status:	All tests passed
Start Time:	20231123 14:52:33.436
End Time:	20231123 14:55:03.336
Elapsed Time:	00:02:29.900
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	12	12	0	0	00:02:30	

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

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Tests	12	12	0	0	00:02:30	
Tests . Chatbot Testsuite	4	4	0	0	00:01:11	
Tests . Dropdown List Testsuite	4	4	0	0	00:00:40	
Tests . Search Testsuite	4	4	0	0	00:00:40	

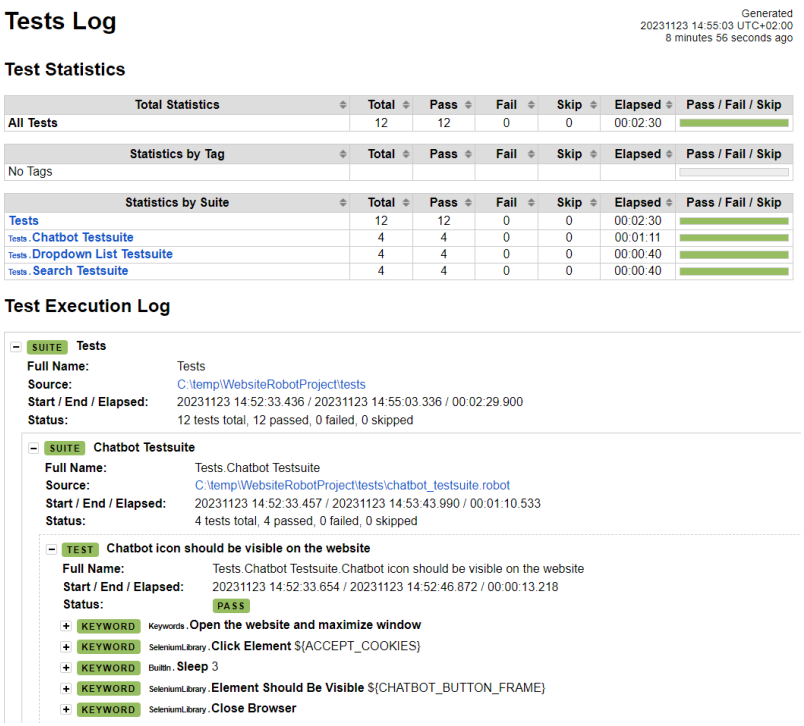
Test Details

All | Tags | Suites | Search

Suite:
Test:
Include:
Exclude:
Search Clear Help

*KUVA 19. **Report.html**-tiedosto avattuna.*

Raportin lisäksi Robot Framework generoi **log.html**-tiedoston, josta voi tarkastella lokeja. Tämä tiedosto antaa tarkemman kuvauksen testauksen suorituksesta. Siitä näkee jokaisen testitapauksen vaihe vaiheelta. Kuvasta 20 näkee, että kaikki kolme testitiedostoa ovat suoriutuneet testeistä onnistuneesti. Lisäksi kuvassa on avattuna **chatbot.testsuite**-testitiedoston **Chatbot icon should be visible on the website** -testitapaus, josta näkee jokaisen avainsanan onnistumisen. Vihreä väri esittää onnistumista, ja jos jokin avainsana epäonnistuu, näkee ohjelmiston kehittäjä tai testaaja suoraan, mikä vaihe epäonnistui.



Tests Log

Generated
20231123 14:55:03 UTC+02:00
8 minutes 56 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	12	12	0	0	00:02:30	

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

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Tests	12	12	0	0	00:02:30	
Tests.Chatbot Testsuite	4	4	0	0	00:01:11	
Tests.Dropdown List Testsuite	4	4	0	0	00:00:40	
Tests.Search Testsuite	4	4	0	0	00:00:40	

Test Execution Log

SUITE Tests
Full Name: Tests
Source: C:\temp\WebsiteRobotProject\tests
Start / End / Elapsed: 20231123 14:52:33.436 / 20231123 14:55:03.336 / 00:02:29.900
Status: 12 tests total, 12 passed, 0 failed, 0 skipped

SUITE Chatbot Testsuite
Full Name: Tests.Chatbot Testsuite
Source: C:\temp\WebsiteRobotProject\tests\chatbot_testsuite.robot
Start / End / Elapsed: 20231123 14:52:33.457 / 20231123 14:53:43.990 / 00:01:10.533
Status: 4 tests total, 4 passed, 0 failed, 0 skipped

TEST Chatbot icon should be visible on the website
Full Name: Tests.Chatbot Testsuite.Chatbot icon should be visible on the website
Start / End / Elapsed: 20231123 14:52:33.654 / 20231123 14:52:46.872 / 00:00:13.218
Status: PASS

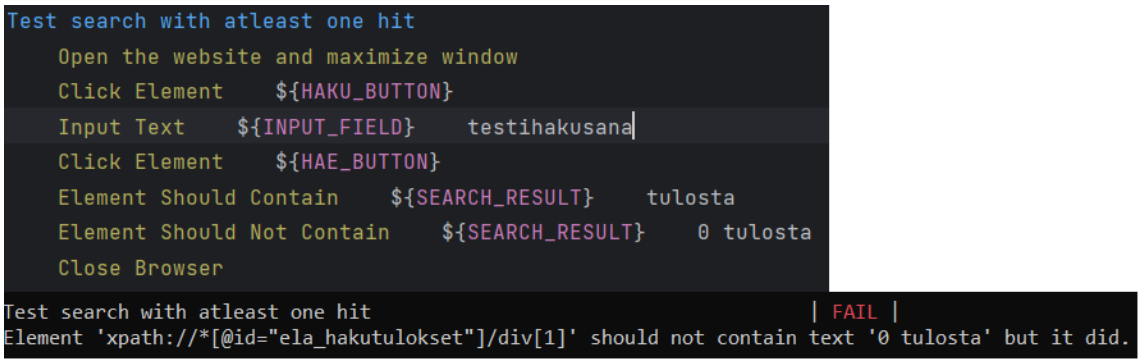
KEYWORD Keywords . Open the website and maximize window
KEYWORD SeleniumLibrary . Click Element ${ACCEPT_COOKIES}
KEYWORD BuiltIn . Sleep 3
KEYWORD SeleniumLibrary . Element Should Be Visible ${CHATBOT_BUTTON_FRAME}
KEYWORD SeleniumLibrary . Close Browser

KUVA 20. Lokitiedosto avattuna.

Lokien tarkastelusta oli minulle hyötyä tässä työssä, koska niiden avulla pystyin näkemään, missä vaiheessa testitiedostot epäonnistuivat, ja lokitiedosto näytti myös testien epäonnistumisesta muodostuneet virheilmoitukset. Nämä auttoivat minua paljon testitiedostojen korjaamisessa.

4.5 Lopputulos ja testien luotettavuus

Työn lopputuloksena oli se, että kaikki testit saatiin suoritettua onnistuneesti. Tässä tapauksessa se tarkoittaa sitä, että testit voitiin suorittaa loppuun ja ne antoivat hyväksytyn tuloksen. Automaatiotestejä kirjoittaessa heräsi kuitenkin ajatus siitä, ovatko kirjoittamani testit luotettavia. Tämä siis tarkoittaa sitä, että vaikka testi antaisikin lopputulokseksi **PASS** eli testi onnistuu, voi olla, että testien lopputarkistus ei välttämättä tarkista oikeaa elementtiä tai sitten tarkistus olisi tehty muuten väärällä tavalla. Päätin kokeilla testien luotettavuutta ja tätä pystyy testaamaan esimerkiksi yrittämällä saada testit epäonnistumaan. Esimerkkinä jo aiemmin esitelty **Test Search with atleast one hit** -testitapaus, jonka pitäisi epäonnistua, jos hakusana tuottaa 0 tulosta tai hakusana on tyhjä. Kuvassa 21 testitapaus muokattuna ja sen epäonnistuminen.



KUVA 21. Testitapauksen muokkaus ja testin epäonnistuminen.

Kuten kuvasta näkee, testi epäonnistui, kun hakusanaa muutettiin sellaiseksi, joka ei antanut yhtään tulosta. Tämän jälkeen Robot Framework ilmoittaa käyttäjälle, että se löysi elementin, jossa lukee ”0 tulosta” vaikka näin ei saisi tapahtua. Tällä tavoin kokeilin muitakin tekemiäni testejä ja pyrin varmistamaan mahdollisimman hyvän luotettavuuden.

5 LOPUKSI

Opinnäytetyöni tavoitteena oli suunnitella ja toteuttaa automaatiotestauksen käyttöönotto verkkosivun käyttöliittymälle. Työn tuloksena tuli ohjeen kaltainen teos, josta lukija saa käsityksen, miten verkkosivun käyttöliittymän automaatiotestauksen suunnittelussa ja toteutuksessa pääsee alkuun.

Tämä työ antoi vastauksen tutkimuskysymyksiini, joista ensimmäisenä oli automaatiotestaustyökalun valintaperusteet. Perustelin eri testaustyökalujen valintaan vaikuttavia tekijöitä ja kolmen vaihtoehdon ominaisuuksia niistä löytyvien lähteiden perusteella. Jälkikäteen testaustyökaluksi valittu Robot Framework osoittautui erittäin hyväksi valinnaksi, koska testien tekeminen sillä sujui suhteellisen vaivattomasti ja tällä työkalulla kaikkien suunnittelemieni testitapausten suorittaminen oli mahdollista. Toisaalta eri testaustyökaluja löytyy valtava määrä ja olisin voinut perehtyä tarkemmin muihinkin vaihtoehtoihin ja jopa päätyä valitsemaan jonkin muun työkalun.

Verkkosivun käyttöliittymän automaatiotestauksen suunnittelu oli sinänsä haasteellista, koska sille ei löytynyt yhtä ja oikeaa toteutustapaa eikä järjestystä, mitä tulisi testata ensin. Osasin kuitenkin suunnitella esimerkkitestit tätä työtä varten hyödyntäen Tech Targetin artikkelia ”Automated software testing” (Alexander 2023), josta löytyi automaatiotestaukseen liittyviä testaustyyppejä, ja lisäksi hyödynsin Guru99 verkkosivun artikkelista ”GUI Testin – UI Test Cases” löytyviä esimerkkitestitapauksia. Jälkikäteen ajateltuna esimerkkejä ei olisi tarvinnut olla niin montaa ja olisin voinut keskittää kaikki testitapaukset yhteen verkkosivun ominaisuuteen ja testata sitä vielä useamman automaatiotestaustyypin näkökulmasta.

Valitun testaustyökalun käyttöä avasin Robot Frameworkin teoriaosuudessa luvussa 3.4.2 ja Testauksen toteutus -luvussa. Avasin teorian ja käytännön avulla mielestäni hyvin tämän työkalun asennusta, syntaksia, seleniumLibraryn käyttöä, raportointia ja avainsanoja.

Testiautomaation tekeminen verkkosivun käyttöliittymälle Robot Frameworkilla oli suhteellisen vaivatonta ja sain dokumentoitua tekemäni testit tähän työhön kertomalla esimerkkitapauksen jokaisesta testitiedostosta. Tekemieni testien onnistumisen takana oli tätä työtä varten tehty teoria Robot Frameworkistä ja aiempi kokemus työkalun käytöstä. SeleniumLibrary -kirjaston avainsanoihin tarvittavien paikantimien löytyminen aiheutti minulle kuitenkin eniten päänvaivaa, koska tämä vaati minulta asiaan tarkemmin perehtymistä ja verkkosivun lähdekoodin ymmärtämistä.

Tätä opinnäytetyötä voisi kehittää vielä kertomalla automaatiotestauksen käytöstä ohjelmistokehityksen tukena. Oman käsitykseni mukaan yksittäiset kehittäjät voivat ajaa testiautomaatiota manuaalisesti omalta koneeltaan, kun taas monissa yrityksissä hyödynnetään jatkuvan kehityksen työkaluja testiautomaation suorittamisessa.

LÄHTEET

Alexander, Gillis. 2023. Automated testing. WWW-dokumentti. Hakupäivä 29.11.2023. https://www.techtarget.com/searchsoftwarequality/definition/automated-software-testing.

Ashiq, Fahad 2022. 15 Best Automated UI Testing Tools In 2023. Blogi. Hakupäivä 13.11.2023. https://www.lambdatest.com/blog/top-ui-automated-testing-tools/.

Dharmendra, Kumar 2023. Software Engineering | SDLC V-Model. WWW-dokumentti. Hakupäivä 6.10.2023. https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/.

Wikipedia 2023. Kuva. Graafinen käyttöliittymä. Hakupäivä 5.10.2023. https://fi.wikipedia.org/wiki/Graafinen_k%C3%A4ytt%C3%B6liittym%C3%A4.

Heavy.AI 2022. Graphical user interface. WWW-dokumentti. Hakupäivä 20.10.2023. https://www.heavy.ai/technical-glossary/graphical-user-interface.

Gupta, Aditya 2023. What is Software Testing? WWW-dokumentti. Hakupäivä 3.10.2023. https://www.geeksforgeeks.org/software-testing-basics/.

Haikala, Ilkka & Mikkonen, Tommi 2011. Ohjelmistotuotannon käytännöt. Helsinki: Talentum Media Oy.

Hamilton, Thomas 2023c. GUI Testing – UI Test Cases (Examples). WWW-dokumentti. Hakupäivä 15.11.2023. https://www.guru99.com/gui-testing.html.

Hamilton, Thomas 2023a. V-Model in Software Testing. WWW-dokumentti. Hakupäivä 15.10.2023. https://www.guru99.com/v-model-software-testing.html.

Hamilton, Thomas 2023b. What is Automation Testing? WWW-dokumentti. Haettu 5.10.2023 https://www.guru99.com/automation-testing.html

Hayes, Linda 2004. The Automated Testing Handbook. Texas: Software Testing Institute.

Klärck, Pekka 2009. Experience. WWW-dokumentti. Hakupäivä 10.11.2023. http://eliga.fi/experience.html.

Philipp, Acsany. Using Python's pip to Manage Your Projects' Dependencies. WWW-dokumentti. Hakupäivä 18.11.2023. https://realpython.com/what-is-pip/.

Robot Framework Foundation. Community. WWW-dokumentti. Hakupäivä 17.10.2023. https://robotframework.org/.

Robot Framework Foundation. Robot Framework User Guide. WWW-dokumentti. Hakupäivä 3.11.2023. https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html.

Try QA 2023. What is Software Testing? WWW-dokumentti. Hakupäivä 12.10.2023. https://tryqa.com/what-is-software-testing/.

Software Testing Help 2023. What Is System Testing – A Ultimate Beginner's Guide. WWW-dokumentti. Hakupäivä 4.10.2023. https://www.softwaretestinghelp.com/system-testing/.