UI-automaatiotestaus Robot Frameworkilla ja Seleniumilla

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus
kevät, 2023

Sebastian Auer

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Tekijä Sebastian Auer Vuosi 2023

Työn nimi UI-automaatiotestaus Robot Frameworkilla ja Seleniumilla

Ohjaaja Esa Huiskonen

TIIVISTELMÄ

Tämän opinnäytetyön tarkoituksena on esitellä lukijalle testiautomaation hyödyt ja erot manuaalisesta testauksesta. Lisäksi opinnäytetyön käytännön osa keskittyy automaattiseen käyttöliittymätestaukseen.

Työssä esitellään kriittisimmät tiedot testiautomaatiosta tiiviissä ja helposti luettavassa muodossa. Työssä esitellään myös automaattisen testauksen ohjelmointia web-pohjaisille käyttöliittymille.

Tämä opinnäytetyö on toiminnallinen, jossa esitellään ensin testauksen ja testiautomaation teoria. Lisäksi esitellään testiautomaation eri tyypit, historia sekä testiautomaation viitekehykset. Käytännön osassa käytetyt sovelluskehykset olivat Selenium ja Robot Framework.

Toiminnallisessa osassa esitetty testiautomaation ohjelmointi tukee teoriaosassa esitettyä mainintaa siitä, että testiautomaation käyttöönotto projektin alkuvaiheessa tehostaa testausprosesseja. Lisäksi testiautomaatiota voidaan käyttää vähentämään inhimillisiä virheitä, joita voi tapahtua toistuvan manuaalisen testauksen aikana. Toiminnallisessa osassa pyrittiin esittelemään toimintoja ja testitapauksia, joita voitaisiin käyttää myös todellisen sovelluksen testaamiseen. Esimerkiksi lomakkeen täyttäminen ja kirjautumistietojen testaus voisi toimia järjestelmän regressiotestien kohteena.

Avainsanat Testiautomaatio, Testaus, Robot Framework, Selenium

Sivut 73 sivua ja liitteitä 14 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology | Abstract
Author Sebastian Auer | Year 2023
Subject UI Testing with Robot Framework and Selenium
Supervisor Esa Huiskonen

ABSTRACT

The purpose of this thesis is to introduce test automation's benefits and differences from manual testing to the reader. Additionally, the practical part of the thesis focuses on automated user interface testing.

The aim of this thesis is to present the most critical information about test automation in a concise and easy-to-read format. The thesis also shows how to program automated testing for web-based user interfaces.

This thesis is practical, where testing and test automation are first introduced. Test automation's history and different types are also explained. Additionally, test automation's frameworks are introduced. The frameworks used in the practical part were Selenium and Robot Framework.

The test automation programming presented in the functional part supports the notion presented in the theory part that introducing test automation in the early phase of a project makes testing processes more efficient. Additionally, test automation can be used to reduce human error that can happen during repeated manual testing. In the functional part, the aim was to present functions and test cases that could also be used in testing a real application. For example, filling out a form and testing login information could serve as the subject of regression testing performed in the system.

Keywords Test Automation, Testing, Robot Framework, Selenium
Pages 73 pages and appendices 14 pages

Sanasto

QA	Quality Assurance
ATDD	Acceptance Test Driven Development
TDD	Test-driven development
Manuaalitestaus	Ihmisen suorittama manuaalinen testaus
Testiautomaatio	Automaatiotyökaluilla suoritettava testaus
Hyväksymistestaus	Yleensä testauksen viimeinen vaihe, jossa kaikki liiketoimintojen prosessit testataan alusta loppuun
Käyttöliittymä	Osa sovelluksesta, jolla käyttäjä voi lähettää käskyjä sovellukselle, tunnetaan englanniksi nimellä UI (User Interface)
Käyttöliittymätestaus	Käyttöliittymän eri toimintojen testaaminen, tunnetaan myös nimellä UI-testaus
Regressiotestaus	Aikaisemmin testattujen toimintojen uudelleentestaus muutosten jälkeen
Testitapaus	Tapahtuma, jolla testataan sovelluksen toimintaa
Testisarja	Testitapausten ryhmittely tai kokoelma, jolla suoritetaan yksi suurempi kokonaisuus testejä
Adhoc-testaus	Satunnaisesti, ilman suunnittelua suoritettava testaus
Käytettävyystestaus	Testaus, jonka tavoitteena on tunnistaa mahdolliset ongelmat ohjelman käytettävyydessä
Tutkiva testaus	Testaus, jossa järjestelmää tutkitaan virheiden varalta ilman ennakkoon luotuja testitapauksia
Agile	Ketterä ohjelmistokehitys, eli Agile on kokoelma sovelluskehitysmetodeja, joissa toimitaan yhteistyössä toimittajan ja asiakkaan kanssa
Sovelluskehitys	Prosessi, jolla luodaan sovelluksia
Testisuunnitelma	Yleensä asiakirjaan koottu suunnitelma tavoitteista, resursseista, aikataulusta, määräajasta, työkaluista, virheiden hallinnasta, testiympäristöistä ja riskeistä
Testiraportti	Asiakirjaan koottu raportti testauksen tuloksista

Testianalyysi	Vaatimusten ja tavoitteiden kerääminen, jotta testauksen suorittamiselle saavutetaan tarvittavat olosuhteet
Havainto	Testauksen suorittamisen aikana tehty havainto, jonka ei tarvitse olla ohjelmistovirhe, vaan esimerkiksi pelkkä kirjoitusvirhe
Bugi	Ohjelman koodissa oleva virhe, joka estää sovelluksen toimimasta oikein
Virheraportti	Virheestä tai havainnosta laadittu raportti
Komponentti	Osa järjestelmää, joka koostuu useista eri komponenteista (komponentit koostuvat moduuleista)
Moduuli	Pieni osa ohjelmistoa, jolla suoritetaan toimintoja
Testiskripti	Riveittäin kirjoitettu kuvaus, joka sisältää tiedot tietyn järjestelmätapahtuman suorittamiseen ja validoimiseen
Robot Framework	Testiautomaatioon käytetty avainsanapohjainen kehys
Web-sovellus	Verkkoselaimella käytettävä sovellus
Selenium	Web-sovellusten automatisointitestaukseen käytetty sovelluskehys
Testikattavuus	Metriikka, jolla määritetään kuinka suuren osan sovelluksesta testit kattavat

Sisällys

1 Johdanto ... 1
2 Testaus ja testiautomaatio ... 2
2.1 Testiautomaation historia ... 3
2.2 Testiautomaation eri tyyppejä ... 3
2.2.1 Toiminnallinen testaus ... 4
2.2.2 Ei-toiminnallinen testaus ... 6
3 Testiautomaation viitekehykset ja työkalut ... 8
3.1 Viitekehysten eri tyypit ... 8
3.1.1 Datapohjainen viitekehys ... 8
3.1.2 Avainsanapohjainen viitekehys ... 9
3.1.3 Käyttäytymispohjainen viitekehys ... 9
3.1.4 Hybridipohjainen viitekehys ... 9
3.2 Testiautomaation sovelluskehykset ... 10
3.2.1 Selenium ... 10
3.2.2 Robot Framework ... 11
4 Testiautomaation erot manuaalitestaukseen ... 13
4.1 Manuaalitestauksen käyttökohteet ... 15
4.2 Automaatiotestauksen käyttökohteet ... 16
5 Testiautomaation kustannukset ... 19
5.1 Sijoitetun pääoman tuotto eli ROI ... 19
5.2 ROI:n laskeminen ... 19
6 Testiautomaation vastuut ja työrutiinit ... 22
7 Menetelmät ... 24
8 Robot Frameworkin ja Seleniumin käyttöönotto ... 25
8.1 Pythonin asennus ... 25
8.2 Pycharm IDE asennus ... 27
8.3 Selenium ja Robot Framework asennus ... 29
8.4 Lisämääritykset PyCharmissa ... 30
8.5 ChromeDriverin asennus ... 32
9 Testien ohjelmointi Robot Frameworkilla ja Seleniumilla ... 34
9.1 Kansion ja tiedoston luonti ... 34
9.2 Ensimmäisen testitapauksen kirjoittaminen ja ajaminen ... 36
9.3 Raportin tarkastelu ... 41

9.4 Ensimmäisen testitapauksen parantaminen 42
10 Muiden testitapausten ohjelmointi 45
10.1 Lomakkeen täyttäminen 45
10.2 Varoitusten käsittely ja välilehden vaihtaminen 49
10.3 Navigointi selaimessa ja kuvakaappaukset 51
10.4 Hiiritoiminnot ja näytön vieritys 52
10.5 Taulujen käsittely ja tietojen laskeminen 55
10.6 Testin alustaminen ja ryhmittely 58
10.7 Resurssitiedostot ja kirjautuminen eri yhdistelmillä 60
10.7.1 Resurssitiedoston määrittäminen 61
10.7.2 Testitiedoston määrittäminen 63
11 Johtopäätökset ja pohdinta 65
12 Yhteenveto 68
Lähteet 69

Kuvat, ohjelmakoodit ja taulukot

Kuva 1 ROI:n laskentakaava....21
Kuva 2 Järjestelmäominaisuudet....26
Kuva 3 Ympäristömuuttujat....26
Kuva 4 Ympäristömuuttujan muokkaus....27
Kuva 5 Pythonin version tarkistaminen....27
Kuva 6 PyCharmin ensinäkymä....28
Kuva 7 PyCharmin projektikansion luominen....28
Kuva 8 Pip list -komento....29
Kuva 9 SeleniumLibraryn asennuksen tarkastaminen....30
Kuva 10 PyCharmin projektin Python -asetukset....31
Kuva 11 Selenium-paketin asennus PyCharmissa....31
Kuva 12 Hyper RobotFramework Support -lisäosan asennus PyCharmiin....32
Kuva 13 ChromeDriverin ajotiedoston vienti Pythonin kansioon....33
Kuva 14 Kansion luominen PyCharmissa....34
Kuva 15 Tiedoston luominen PyCharmissa....35
Kuva 16 Terminaalin avaaminen PyCharmissa....35
Kuva 17 Elementtien tarkastelu Chromessa....37
Kuva 18 Käyttäjänimen paikannin....37
Kuva 19 Kirjautumispainikkeen paikantimen tarkastaminen....38
Kuva 20 Kirjautumispainikkeen XPathin kopioiminen....39
Kuva 21 Testin käynnistäminen PyCharmissa....40
Kuva 22 Testin läpi meneminen PyCharmissa....40
Kuva 23 Testiraportin tarkastelu....41
Kuva 24 Testiajon tietojen avaaminen....42
Kuva 25 Testin lokitietojen tarkastelu....42
Kuva 26 Epäonnistunut testi PyCharmissa....44
Kuva 27 Evästeiden hyväksyminen....46
Kuva 28 Nimikenttien elementin tarkastaminen....47
Kuva 29 Valintaruutujen elementtien tarkastaminen....47
Kuva 30 Pudotusvalikon elementin tarkastaminen....48
Kuva 31 Lomakkeen täyttäminen vaihe 1....48

Kuva 32 Listan elementin tarkastaminen 48
Kuva 33 Lomakkeen täyttäminen vaihe 2 49
Kuva 34 Painikkeen elementin paikannin....50
Kuva 35 Varoitusteksti....51
Kuva 36 Logon elementin paikannin 52
Kuva 37 Kontekstimenun elementin paikannin 53
Kuva 38 Tuplaklikattavan elementin paikannin 54
Kuva 39 Raahattavan elementin paikannin....55
Kuva 40 Taulukon elementin paikannin 55
Kuva 41 Taulun elementin paikannin 2 56
Kuva 42 Taulun tietojen käsittely vaihe 1....57
Kuva 43 Taulun tietojen käsittely vaihe 2....58
Kuva 44 Alustetun testin ajaminen 60
Kuva 45 Uuden kansion ja resurssitiedoston luominen 61

Ohjelmakoodi 1 Selaimen avaaminen ja ikkunan suurentaminen....36
Ohjelmakoodi 2 Tekstin syöttäminen....38
Ohjelmakoodi 3 Kirjautumispainikkeen painaminen ja selaimen sulkeminen 39
Ohjelmakoodi 4 Muuttujien ja omien avainsanojen lisääminen testiin 43
Ohjelmakoodi 5 Tekstikentän näkyvyyden varmistaminen 43
Ohjelmakoodi 6 Epäonnistuneen testin tekeminen....44
Ohjelmakoodi 7 Lomakkeen täyttäminen 45
Ohjelmakoodi 8 Varoitusten käsittely....49
Ohjelmakoodi 9 Navigointi selaimessa ja kuvakaappaukset....51
Ohjelmakoodi 10 Hiiritoiminnot ja näytön vieritys 52
Ohjelmakoodi 11 Taulujen käsittely ja tietojen laskeminen 55
Ohjelmakoodi 12 Testin alustaminen ja ryhmittely 58
Ohjelmakoodi 13 Resurssitiedosto kirjautumiseen....62
Ohjelmakoodi 14 Virheellisen kirjautumisen testaus 63
Ohjelmakoodi 15 Onnistuneen kirjautumisen testaus....63

Taulukko 1 Testiautomaation erot manuaalitestaukseen 14

Liitteet

Liite 1	Aineistonhallintasuunnitelma
Liite 2	Pythonin asennus
Liite 3	PyCharmin asennus
Liite 4	Seleniumin ja Robot Frameworkin asennus
Liite 5	ChromeDriverin asennus

1 Johdanto

Opinnäytetyön tarkoituksena on johdattaa lukija testiautomaation maailmaan. Testiautomaatio on tullut viime vuosina entistä enemmän käyttöön sovelluskehityksessä ja koen aiheen olevan minulle mainio mahdollisuus oppia lisää testiautomaatiosta ja sen hyödyistä. Olen tehnyt työurallani manuaalista testausta ja saanut hieman kokemusta testiautomaatiosta. Halusin kuitenkin sukeltaa syvemmälle testiautomaation oppimisessa. Teoriaosassa kerron siitä mitä käyttöliittymän testauksen automatisointi on, miten siitä hyödytään, miten se eroaa manuaalitestauksesta ja minkälaisia hyötyjä siitä saadaan rahallisesti sekä työtyytyväisyysmielessä. Työn kohdeyleisöksi on tarkoitettu testaajat, testauksesta kiinnostuneet ja IT-alan yritykset, jotka harkitsevat testiautomaation käyttämistä.

Toiminnallisessa osassa keskityn käyttöliittymän testiautomatisointiin eli UI-automaatiotestaukseen, koska koen sen olevan hyvä ja visuaalinen tapa demonstroida testiautomaation hyötyjä. UI-automaatiotestauksella tarkoitetaan käyttöliittymän eri toimintojen testaamista automaattisilla testeillä. Haluan antaa selkeän kuvan siitä, mitä kaikkea testien suorittaminen vaatii, vaikka testausta yrittäisi henkilö, joka ei ole kokenut testaaja. Kuvaan tarvittavien työkalujen käyttöönoton, jonka jälkeen ohjelmoin Robot Frameworkia ja Seleniumia hyödyntämällä testitapauksia, joilla saa selkeän kuvan siitä minkälaisia asioita voidaan automatisoida käyttöliittymän testausta varten. Testitapauksina toimii perustapauksia kuten lomakkeiden täyttäminen, valintanapit, pudotusvalikko, salasanatietojen testaaminen, ikkunoiden avaaminen ja suurentaminen. Lisäksi käsitellään varoituksia, navigointia, hiiren toimintoja sekä elementtien näkyvyyttä.

Tutkimuskysymykset:

- Miten testiautomaatio eroaa manuaalitestauksesta ja miten siitä hyödytään?
- Millaisia kustannuksia automaatiotestaus aiheuttaa verrattuna manuaalitestaukseen?
- Miten automatisoitu testaus vaikuttaa päivittäisiin työrutiineihin?
- Miten Robot Framework/Selenium otetaan käyttöön ja miten testejä kirjoitetaan?

2 Testaus ja testiautomaatio

Testaus on yksi sovelluskehityksen kriittisimpiä osa-alueita ja sillä varmistetaan, että ohjelmisto on virheetön ja vastaa sille laadittuja vaatimuksia. Testauksen tarkoituksena on löytää sovelluksesta virheitä sekä mahdollisia puutteita. Testauksella kokeillaan ja arvioidaan ohjelmiston eri komponentteja manuaalisilla tai automaattisilla keinoilla. Kunnolla testattu tuote takaa, että se on tarpeeksi turvallinen ja luotettava tuotantokäyttöön, jolloin myös asiakastyytyväisyyden ja -pysyvyyden saavuttaminen on helpompaa. (Hamilton, 2020a)

Yksi perinteisimmistä testaustavoista on manuaalinen testaus, jossa testejä suoritetaan ilman automatisointityökaluja, loppukäyttäjän näkökulmasta. Manuaalinen testaus on yksi varmimmista tavoista löytää virheitä sovellusta testattaessa, mutta ajan saatossa on esiintynyt tarve tehostaa testausprosesseja automatisoimalla testejä. Manuaalisessa testauksessa joudutaan suorittamaan paljon samankaltaisia testitapauksia ja tämä syö paljon aikaa, joka johtaa samalla testauksen kustannusten kasvamiseen. Laajemmissa sovelluksissa testitapausten määrä ja toistuvuus voi olla niin suuri, että inhimillisten virheiden riski kasvaa. (Hamilton, 2020b)

Testiautomaatiossa voidaan suorittaa yksinkertaisia sekä monimutkaisia testitapauksia käyttämällä testiautomaatioon tarkoitettuja työkaluja. Testien suorittamiseen ohjelmoidaan skriptejä, jotka voidaan toistaa milloin vain. Tämä sopii mainiosti tilanteeseen, jossa usein toistuvia tilanteita on pakko testata usein. Kun usein suoritettavia testejä saadaan automatisoitua, resursseja voidaan vapauttaa johonkin muuhun kehitysprojektin osa-alueeseen, kuten testauksen jatkosuunnitteluun. (*SmartBear*, n.d.)

Yksi suurimpia testiautomaation hyötyjä on siitä syntyvä raportointi. Raportit kertovat tarkalleen mitä on testattu ja testauksen tulokset. Raportit ovat myös paljon tarkemmalla tasolla, kuin mitä manuaalisesta testauksesta syntyy. Tämän lisäksi automatisoidun testauksen raportit ovat lähes heti saatavilla testien ajamisen jälkeen, kun taas manuaalisen testauksen raportteihin kuluu kauemmin aikaa, koska ne kirjoitetaan käsin. (Cummings-John, n.d.)

Testiautomaatiota käytetään usealla testauksen osa-alueella, kuten käyttöliittymä-, yksikkö-, integraatio- ja tietokantatestauksessa. Käyttöliittymän testaus on yksi kriittisimmistä testauksen osa-alueista ja sen toimivuus on todella tärkeää asiakastyytyväisyyden näkökulmasta. Jos sovelluksen käyttöliittymä toimii huonosti, tämä voi karkottaa sovelluksen käyttäjäkunnan nopeasti. (*SmartBear*, n.d.)

2.1 Testiautomaation historia

Testiautomaation alkuperä sijoittuu 1970-luvulle, jolloin tietokoneet eivät olleet vielä laajalti valtavirran käytössä. Internetiin pääsy oli vaikeampaa ja tämä hankaloitti mm. päivitysten levittämistä. Yksi suurimpia ongelmia oli sovellusten yhteensopivuus vain tietyn tyyppisten tietokoneiden kanssa. Monesti myös ohjelmien testit toimivat vain tietylle ohjelmistoversiolle, joka aiheutti sen, että testejä jouduttiin päivittämään ja levittämään ohjelmistoversion mukaan. (ZapTest, n.d.)

1970-luvulla todettiin lopulta, että testauksen suorittamiseen voitaisiin hyödyntää myös käytössä olevia ohjelmistoja. Jo tähän aikaan haluttiin puuttua manuaaliseen testaukseen ja korvata se automatisoinnilla. Kun tätä lähdettiin toteuttamaan käytännössä, ongelmaksi muodostui testien automatisointiin käytettävien sovellusten huono toimivuus. Testisarjat (test suite) hajosivat usein, kun ohjelmistoon, tietokantoihin tai kehitystyökaluihin tehtiin päivityksiä. Työkaluihin lisättiin toiminnallisuuksia, joilla pystyttiin vähentämään päivityksiä vaativia muutoksia, mutta tästä huolimatta automaattisten testien ylläpito vei enemmän aikaa kuin manuaalisen testauksen suorittaminen. (ZapTest, n.d.)

1990-luvulla graafisten käyttöliittymien ja henkilökohtaisten tietokoneiden yleistyminen kasvatti testiautomaation tarvetta ja testaukseen erikoistuneet tahot pitivät tärkeänä kehittää testiautomaatioon soveltuvia työkaluja. Nykypäivänä automatisointiin soveltuvia työkaluja on paljon ja niiden käyttö on helpottunut huomattavasti. (ZapTest, n.d.)

2.2 Testiautomaation eri tyyppejä

Testiautomaatiota varten voidaan hyödyntää useita eri testaustyyppejä. On kuitenkin tärkeää tuntea eri testaustyypit ennen kuin tekee päätöksiä siitä, mitä eri testaustyyppejä

ottaa käyttöön omassa organisaatiossa ja millaisia työkaluja tarvitaan. Kun organisaatio tietää tarkalleen minkä tyyppistä testausta käytetään, testiautomaatiosta saadaan enemmän irti. (*Adservio*, n.d.)

2.2.1 Toiminnallinen testaus

Testauksen voi jakaa kahteen eri päätyyppiin, toiminnallinen ja ei-toiminnallinen testaus. Tämä pätee myös testiautomaatioon. Toiminnalliset testit johdetaan sovelluksen vaatimuksista ja niissä keskitytään sovelluksen päätoimintoihin, käytettävyyteen ja saatavuuteen. (Prolifics, n.d.)

Käyttäjän ei tarvitse yleensä tuntea toiminnallisten testien suorittamiseen sovelluksen sisäistä rakennetta. Toiminnalliseen testaukseen kuuluu mm. yksikkö-, integraatio, hyväksymis-, käyttöliittymä-, ja regressiotestaus. (Prolifics, n.d.)

Yksikkötestaus voidaan suorittaa joko manuaalisesti tai automaatiolla (Lead, n.d.). Yksikkötestaus keskittyy sovelluksen yksittäisiin toiminnallisuuksiin tai komponentteihin ja useimmiten sen suorittaa sovelluskehittäjä. Yksikkötestauksen avulla saadaan tietoa myös siitä, voiko samanlaista koodia käyttää järjestelmän muissa osissa. (Prolifics, n.d.)

Yllä mainituista syistä yksikkötestaus auttaa kehitystiimiä ymmärtämään sovelluksen eri osapuolia. Yksikkötestaus on yksi ensimmäisistä testausvaiheista kehitysprojektissa, koska se auttaa löytämään virheitä sovelluksen kehityksen aikaisessa vaiheessa. (Prolifics, n.d.)

Integraatiotestausta käytetään, kun sovelluksessa on yhdistetty eri komponentteja. Järjestelmän komponentit rakennetaan usein useamman kehittäjän toimesta eri tavoilla ja integraatiotestauksen tarkoituksena on löytää virheitä komponenttien välisessä kommunikaatiossa. Integraatiotestaus suoritetaan yksikkötestauksen jälkeen ja sen avulla saadaan hyvä kuva siitä, onko eri komponenttien ja toimintojen välinen vuorovaikutus sujuvaa. Järjestelmä toimii hyvin kokonaisuutena, kun integraatiotestaus suoritetaan onnistuneesti. (Hamilton, 2020d)

Kehitysprojekteissa saattaa kuitenkin käydä myös niin, että uutta moduulia kehitetään ja asiakas esittää kehityksen aikana uusia vaatimuksia ja vaatimuksista syntyviä muutoksia ei

välttämättä yksikkötestata. Tässä tapauksessa integraatiotestaus tulee tarpeeseen. (Hamilton, 2020d)

Hyväksymistestauksen päätavoitteena on vakuuttua siitä, että sovellus täyttää sille asetetut hyväksymiskriteerit. Tähän tavoitteeseen päästään silloin, kun testien kattavuus on riittävä, järjestelmä on vakaa ja asiakaskäyttöön soveltuva. (TestProject, 2022)

Hyväksymistestaus ei voi onnistua, jos vaatimuksia ja hyväksymiskriteereitä ei ole kommunikoitu selkeästi. Kun hyväksymistestaus suoritetaan onnistuneesti, asiakas tai muu taho voi päättää hyväksytäänkö järjestelmä tuotantokäyttöön. (TestProject, 2022)

Automatisoitu hyväksymistestaus on käytännössä sitä, että laadunvarmistukseen määritetty ryhmä ajaa testiautomaatiotyökalulla sellaisia testejä, jotka ovat järjestelmän hyväksymiskriteereissä. Tämän avulla hyväksymistestaus voidaan suorittaa nopeasti, eikä testejä tarvitse suorittaa joka kerta manuaalisesti. (Kok, n.d.)

Käyttöliittymätestauksella todennetaan sovelluksen käyttöliittymän toimivuus, varsinkin loppukäyttäjän näkökulmasta. Käyttöliittymän testauksen automatisointi on yleistynyt nykypäivänä, mutta se on yleensä monimutkaisempaa kuin muut testiautomaation osa-alueet (Lead, n.d.). Käyttöliittymän avulla käyttäjä navigoi järjestelmää ja suorittaa eri toimintoja, kuten tietojen lisäämistä tai muokkaamista. (*Computer Hope*, n.d.)

Käyttöliittymän testaus on mahdollista suorittaa manuaalisesti, mutta manuaalitestaus voi pitkittyessään aiheuttaa inhimillisiä virheitä. Käyttöliittymän ominaisuuden testaus voidaan automatisoida yleensä silloin, kun sen kehitys on valmis. (*Devbridge*, n.d.)

Automatisoituja käyttöliittymätestejä voidaan suorittaa usealla eri selaimella ja alustalla samaan aikaan, joka korostaa aikatehokkuutta. Automatisoitu käyttöliittymätestaus on myös oiva tapa suorittaa regressiotestausta. Jos järjestelmän käyttöliittymään on tehty muutoksia ja aikaisemmin toiminut käyttöliittymätesti ei enää toimi, muutos on jollain tavalla rikkonut olemassa olevan toiminnallisuuden. (*Devbridge*, n.d.)

Toiminnallisen osuuden käyttöliittymätesteissä hyödynnetään XPathia elementtien polkujen tarkastelua varten. XPath (XML Path Language) on standardisoitu kieli, jota käytetään tiedon

hakemiseen XML-dokumenteista ja sen avulla saadaan myös selville verkkosivulla olevan elementin polku. XPath-lauseke koostuu askeleista, jotka erotetaan vinoviivalla. XPathista on kaksi merkintätapaa, lyhennetty ja täysi syntaksi. Täyden syntaksin heikkoutena pidetään sitä, jos sivulla olevan elementin polkuun tehdään muutoksia, sen käyttö epäonnistuu, jolloin se täytyy päivittää. Lyhennetyn syntaksin käyttö on suositeltavampaa, koska se ei käytä täyttä polkua juurielementistä. (Rungta, 2020b)

Regressiotestauksessa ajetaan toiminnallisia sekä ei-toiminnallisia testejä, kun sovellukseen on tehty muutoksia tai lisäyksiä. Päätarkoituksena regressiotestauksessa on vakuuttua siitä, että tehdyt muutokset eivät ole vaikuttaneet mihinkään sovelluksen osa-alueeseen. (Prolifics, n.d.)

Regressiotestaukseen valittavat tapaukset riippuvat organisaation tarpeista. Regressiotestauksesta hyödytään varsinkin silloin, kun sitä käytetään kriittisiin tai muutettuihin toiminnallisuuksiin sekä järjestelmän osiin, joissa esiintyy usein ongelmia. (Hamilton, 2020e)

Smoke-testaus suoritetaan silloin kun sovellukseen on tehty muutoksia ja sillä varmistutaan siitä, että muutokset eivät ole vaikuttaneet sovelluksen vakauteen ja eri toiminallisuuksien toimivuuteen. Smoke-testaus eroaa regressiotestauksesta siten, että smoke-testaus suoritetaan sovelluksen tuotantoversiolle tai versiolle, joka soveltuu tuotantokäyttöön. (Prolifics, n.d.)

Smoke-testaukseen määritetään pieni määrä kriittisiä testitapauksia ja jos yksikin näistä epäonnistuu, kehitystiimi joutuu korjaamaan tilanteen siten, ettei smoke-testaus epäonnistu seuraavalla kerralla. (Prolifics, n.d.)

2.2.2 Ei-toiminnallinen testaus

Ei-toiminnallisessa testauksessa keskitytään sovelluksen ei-toiminnallisiin osa-alueisiin, kuten suorituskykyyn, tietoturvaan ja skaalautuvuuteen. (Prolifics, n.d.).

Suorituskykytestauksella saadaan tietoa siitä, kuinka hyvin sovellus suoriutuu, kun sille asetetaan normaalia enemmän työkuormaa käsiteltäväksi. Yksi yleisiä ongelmia, joita

järjestelmässä voi esiintyä käyttäjien määrän kasvaessa on käyttöliittymän hidastuminen. Suorituskykytestauksen avulla päästään tällaisiin ongelmiin kiinni, jolloin ne voidaan myös korjata, etteivät ne esiinny tuotantokäytössä loppukäyttäjälle. (Lead & Boog, 2019)

Järjestelmän suorituskykyä mitataan semmoisella datalla, jota järjestelmässä esiintyy tuotantokäytössäkin. Testien tulokset ovat luotettavampia, kun testeissä käytettävä data on semmoista, jota syntyy myös loppukäyttäjän toimesta. (Lead & Boog, 2019)

Suorituskyvyn testaus on lopulta hyvin kriittistä järjestelmän käyttäjäkunnan kannalta ja jos sitä ei ole suoritettu kunnolla, järjestelmässä voi esiintyä suuremmalla todennäköisyydellä hitautta tai jopa kaatumisia. Tällaiset ongelmat voivat karkottaa käyttäjiä nopeasti. (Lead & Boog, 2019)

3 Testiautomaation viitekehykset ja työkalut

Ennen kuin aloitat testiautomaation, on hyvä tuntea testiautomaation erilaiset viitekehykset. Viitekehys koostuu säännöistä ja sovelluksista, joita käytetään testien rakentamiseen ja ajamiseen. Oikealla viitekehyksellä optimoidaan laadunvarmistustiimin testiautomaation tehokkuutta. (*SmartBear*, n.d.)

3.1 Viitekehysten eri tyypit

Viitekehysten tärkeimpänä tehtävänä on kasvattaa testiautomaation tehokkuutta. Viitekehyksiin liittyvät säännöt ja koodi on useimmiten standardisoitua ja tämä edesauttaa testien ajamista siten, ettei testien kattavuudessa ole poikkeuksia. Viitekehysten säännöstöjen vuoksi testitapaukset ovat samanlaisia, joka auttaa niiden uudelleenkäyttämisessä. Kun kirjoitat testitapausta, voit käyttää sitä johonkin samankaltaiseen testitapaukseen tulevaisuudessa. Kun testitapauksia käytetään uudelleen, niiden rakentamiseen käytetty aika vähenee, joka samalla säästää työstä syntyviä kustannuksia. (*BrowserStack*, n.d.)

Testiautomaation viitekehykset elävät siis tarkkojen sääntöjen keskellä, jolloin testien ajamiseen vaaditaan myös hyvin vähän manuaalista työtä. Jos testit epäonnistuvat, niiden uudelleenajamiseen vaaditaan yleensä vain pieniä muutoksia. (*BrowserStack*, n.d.)

3.1.1 Datapohjainen viitekehys

Datapohjainen viitekehys toimii parhaiten sellaisissa tilanteissa, joissa joudut testaamaan tiettyä tilannetta useaan kertaan, mutta erilaisella datalla. Testidata ei siis saa olla kovakoodattu testeihin tällaisissa tapauksissa. (*SmartBear*, n.d.)

Datapohjaisessa viitekehyksessä hyödynnetään ulkoisia tietolähteitä kuten Exceliä tai CSV-tiedostoja ja käyttäjä voi määrittää haluamansa parametrit testiskripteihin. Datapohjaisia testejä voidaan siis ajaa hyvin nopeasti, koska erilaisten tietolähteiden valmistelu ja muutosten tekeminen ei ole työlästä. (*SmartBear*, n.d.)

3.1.2 Avainsanapohjainen viitekehys

Avainsanapohjaista viitekehystä pidetään datapohjaisen viitekehyksen jatkeena. Avainsanojen tiedot poimitaan ulkoisista tietolähteistä ja testitapausten avainsanat esitetään taulukossa niin että ne ovat loogisessa järjestyksessä. (*SmartBear*, n.d.)

Avainsanojen avulla sovellus tietää mitä toimintoja testissä suoritetaan ja kun avainsanat ovat tallennettu taulukkoon, niitä on helppo ylläpitää ja niissä käytettävä kieli on selkeästi ymmärrettävää. Avainsanat laaditaan standardisoidulla tavalla, joten niitä voi käyttää myös muissa samankaltaisissa testeissä. (*SmartBear*, n.d.)

3.1.3 Käyttäytymispohjainen viitekehys

Käyttäytymispohjaisessa viitekehyksessä testitapaukset laaditaan helposti luettavassa ja ymmärrettävässä muodossa siten, että koko projektiryhmä ymmärtää ne. Tällä vältetään väärinymmärryksiä projektiryhmän välillä kehitysprojekteissa. Kun jargonia käytetään testitapauksissa vähän, teknisten ja vähemmän teknisten henkilöiden ymmärrys pysyy samalla tasolla. (*Software Testing Help*, 2023)

Käyttäytymispohjaisessa viitekehyksessä testitapaukset laaditaan ominaisuuskohtaisesti omiin Feature-tiedostoihin ja niissä keskitytään nimensä mukaisesti järjestelmän käyttäytymiseen. Testitapausten taustalle kirjoitetaan kuitenkin vielä itse koodi esimerkiksi Javalla tai Pythonilla. (*Software Testing Help*, 2023)

3.1.4 Hybridipohjainen viitekehys

Testiautomaation viitekehyksiä yhdistetään nykypäivänä. Hybridipohjaisella viitekehyksellä voidaan hyödyntää eri viitekehysten vahvuuksia ja samalla lieventää viitekehyskohtaisia heikkouksia. (*SmartBear*, n.d.)

Agile-malliset kehitysprojektit ovat yleisiä nykypäivänä ja oikeiden testausprosessien hyödyntäminen on kriittisen tärkeää projektin onnistumiselle. Sovellukset ovat aina erilaisia ja niihin tulisi hyödyntää parhaita mahdollisia testausprosesseja. Hybridikehyksellä

testausprosessit voidaan mukauttaa siten, että ne hyödyntävät projektia parhaalla mahdollisella tavalla. (*SmartBear*, n.d.)

3.2 Testiautomaation sovelluskehykset

Testiautomaation eri viitekehyksistä ja työkaluista muodostettua yhdistelmää kutsutaan sovelluskehykseksi, jolla voidaan muuttaa manuaaliset testit automaattisiksi. Käytännössä työkalut ovat ohjelmistoja, joilla testaus muutetaan sellaiseksi, ettei ihmisen tarvitse puuttua testien ajamiseen. Työkalujen välillä on kuitenkin eroja esimerkiksi testitapausten määritystavoissa ja mihin sovellustyyppeihin työkaluja käytetään. Myös lisensseissä on eroja ja jotkut työkalut ovat ilmaisia, maksullisia tai jotain siltä väliltä. Työkalut sopivat yleensä joko web-, mobiili- tai työpöytäsovelluksiin ja testitapaukset voidaan rakentaa joko ohjelmointi- tai skriptauskielellä. Jotkut työkalut eivät vaadi erityisiä ohjelmointitaitoja. (Testim, 2022).

3.2.1 Selenium

Seleniumin sarjaan kuuluu useampi työkalu, Selenium IDE, WebDriver ja Selenium Grid. Tämän työn toiminnallisessa osassa hyödynnettävä WebDriver on laajasti käytetty työkalu, jota käytetään Web-pohjaisten sovellusten testiautomaatioon. WebDriveria voidaan hyödyntää useilla eri selaimilla Python, Java, C# ja Ruby -ohjelmointikielillä. Sen voi myös ottaa käyttöön Windows, Mac, Solaris ja Linux -käyttöjärjestelmissä. (Rungta, 2020a)

WebDriverin avulla suoritettavat testit tapahtuvat samalla tavalla kuin loppukäyttäjä itse ohjaisi selainta, joko paikallisesti tai etänä. WebDriverin tarkoituksena on tarjota yksinkertainen ohjelmointiliittymä ja se soveltuu selaimen automatisointiin mainiosti. Käytännössä WebDriver on ajuri ja jokaiseen tuettuun selaimeen on oma WebDriver toteutuksensa. Ajurin avulla Selenium kommunikoi tietoa selaimen kanssa. (*Selenium*, n.d.)

Seleniumin käyttöönotto on hieman erilainen kuin muissa ohjelmointiin käytettävissä työkaluissa. Testien kirjoittamista varten joudut asentamaan itsellesi tiettyyn ohjelmointikieleen sitovan kirjaston, haluamasi internet-selaimen sekä selainkohtaisen ajurin. Kun olet asentanut selaimen, kirjaston ja ajurit, voit aloittaa testien ajamisen. Yksi

ensimmäisistä Selenium-pohjaisista testeistä voi olla esimerkiksi selaimen avaaminen ja tietylle sivulle siirtyminen. (*Selenium*, n.d.)

3.2.2 Robot Framework

Robot Framework on laajalti käytetty, avoimen lähdekoodin Python-pohjainen testiautomaation sovelluskehys. Robot Frameworkilla testit voidaan kirjoittaa lähes millä tahansa ohjelmointiympäristöllä ja testien avainsanoissa käytetty syntaksi eli muotoilusääntö on tarkoitettu helposti luettavaksi ja niiden ymmärtäminen ei vaadi laajaa ohjelmointiosaamista. Robot Frameworkissa on omia sisäänrakennettuja kirjastoja esim. merkkejä ja numeroita varten, mutta se tukee myös laajasti muita testiautomaatioon soveltuvia kirjastoja, kuten Seleniumia. Robot Framework soveltuu pääosin avainsana-, data- ja käyttäytymispohjaisten viitekehysten käyttöön. (*Tutorials Point*, n.d.)

Robot Frameworkissa on useita eri ominaisuuksia ja testitapaukset laaditaan yksinkertaisilla taulukkomuotoisilla avainsanoilla. Avainsanat ovat laajalti sisäänrakennettuja, mutta muita yhteensopivia, kuten Seleniumin avainsanoja voidaan hyödyntää. Avainsanoja voidaan myös soveltaa siten että niissä yhdistetään käyttäjän itse luomia avainsanoja sisäänrakennettujen avainsanojen kanssa. Avainsanoille voidaan välittää myös argumentteja, joka edesauttaa avainsanojen uudelleenkäyttämistä. (*Tutorials Point*, n.d.)

Resurssien avulla voit hyödyntää muita Robot Framework -tiedostoja. Käytännössä voit tuoda ulkoisen robottitiedoston, ja käyttää siinä olevia avainsanoja omissa testitapauksissasi. Resurssien käyttö on tästä syystä hyödyllistä, koska niiden avulla voit tarvittaessa hyödyntää muissa projekteissa tai testeissä käytettyjä avainsanoja. (*Tutorials Point*, n.d.)

Robot Framework sallii myös testitapauksien merkkaamisen (tagging). Merkkaamisen avulla voit ajaa isommasta erästä vain tiettyjä testejä tai poissulkea merkkaamasi testit. Merkkaaminen on siis järkevä tapa ajaa tietty ryhmä testitapauksia. (*Tutorials Point*, n.d.)

Robot Frameworkin sisäänrakennettu raportointi tarjoaa tiedot testiajojen tuloksista. Testiajosta syntyy raportti, josta nähdään mitä testejä on ajettu, mitkä testit ovat menneet onnistuneesti läpi ja mitkä eivät. Lisäksi raportista nähdään yksittäisten testien ja koko testiajon kesto. (*Tutorials Point*, n.d.)

Robot Framework ei tarjoa oletuksena kaikkia tarvittavia asioita verkkosivujen automaatiotesteihin. Tätä varten on olemassa SeleniumLibrary, jolla voidaan hyödyntää Seleniumin omia, verkkosivujen automaatiotestaukseen soveltuvia avainsanoja Robot Frameworkissa. (*BrowserStack*, n.d.)

Kun testejä ajetaan Robot Frameworkilla, SeleniumLibrary hyödyntää Selenium WebDriveria verkkoselaimen ohjaamiseen. SeleniumLibrarya voidaan hyödyntää verkkosivujen automaatiotesteissä useisiin erilaisiin toimintoihin, kuten verkkosivujen avaamiseen, linkkien klikkaamiseen, verkkosivujen elementtien tarkastamiseen, valintaruutujen valitsemiseen, ruutukaappausten ottamiseen tai tekstin täyttämiseen. (*Robot Framework*, n.d.)

4 Testiautomaation erot manuaalitestaukseen

Tässä luvussa käsitellään suurimpia eroja manuaalitestauksen ja testiautomaation välillä. Manuaali- ja automaatiotestauksessa on joitakin samankaltaisuuksia, mutta ne eroavat kuitenkin useilla eri tavoilla. Alaluvuissa käydään läpi myös millaisiin tilanteisiin manuaali- ja automaatiotestaus sopivat parhaiten. Manuaali- ja automaatiotestauksen suurimmat erot ovat esitetty myös Taulukko 1 Testiautomaation erot manuaalitestaukseen.

Testiautomaatio on kasvattanut suosiotaan viime vuosina. Testiautomaatiolla on parannettu testauksen tehokkuutta ja kattavuutta. Tämä on johtanut siihen, että myös uusien ominaisuuksien käyttöönotto sovelluksissa on nopeampaa. Manuaalitestauksella on kuitenkin vielä oma paikkansa eikä automaatiotestaus tule korvaamaan sitä, tai toisinpäin. (*BrowserStack*, n.d.)

Manuaalisessa testauksessa tutkitaan verkkosivun tai sovelluksen ominaisuuksia ja etsitään vikoja tai muita poikkeavuuksia. Sovellusta testataan siis samalla tavalla kuin loppukäyttäjä käyttäisi sitä tuotantokäytössä. Tämä mahdollistaa käyttäjäkokemuksen testaamisen verrattuna automaatiotestaukseen, joka ei ota huomiota käyttäjäkokemukseen. Manuaalisessa testauksessa noudatetaan yleensä kirjallista testisuunnitelmaa, johon on kirjattu erilaisia testitapauksia. Itse manuaalisen testin suorittamiseen ei liity viitekehyksiä vaan ennemminkin yleisiä sääntöjä ja prosesseja. Testitapauksista nähdään miten sovelluksen kuuluisi toimia ja jos manuaalitestauksessa esiintyy virheitä, nämä raportoidaan kehittäjille ohjelmointivirheenä eli bugina. Kehittäjä joutuu toistamaan testaajan raportoiman poikkeaman ja korjaamaan bugin. (*BrowserStack*, n.d.)

Testiautomaatiossa testitapausten suorittamiseen laaditaan koodia tai skriptejä ja testit ajetaan testiautomaatioon soveltuvalla työkalulla. Testit ovat aina ennalta laadittuja, viitekehysten säännöstöihin perustuvia ja niiden suorituksessa nähdään, toimiiko sovellus testitapauksiin määritetyllä tavalla. Vaikka automaatiotestauksella voidaan suorittaa testitapauksia automaattisesti, itse testiskriptien kirjoittaminen vaatii manuaalista työtä. (Hamilton, 2020c)

Jos sovellukseen tulee muutoksia, usein myös automaattisiin testeihin täytyy tehdä muutoksia ja manuaalisessa testauksessa muutokset voi testata saman tien esimerkiksi tutkivalla testauksella. Raportointi on yksi testiautomaation vahvuuksista ja useimmat testiautomaatiotyökalut generoivat raportit automaattisesti testien suorittamisen yhteydessä, kun taas manuaalisessa testauksessa raportit joudutaan kirjoittamaan käsin, joka on aikaa vievää. (Hamilton, 2020c)

Manuaalitestauksen suorittaminen itsessään on myös hitaampaa, kun taas testiautomaation suorittaminen on mahdollista sekunneissa tai minuuteissa, riippuen testitapausten määrästä ja kokoluokasta. Testiautomaatio vaatii usein alussa enemmän sijoitettavaa pääomaa, mutta yleensä tuotto on korkeampi kuin manuaalisella testauksella. Manuaalisessa testauksessa voi myös tapahtua inhimillisiä virheitä verrattuna automaatiotesteihin, jotka suoritetaan täysin kuten testiskripteihin on määritelty. Jos testejä täytyy suorittaa esimerkiksi usealle selaimelle samanaikaisesti, manuaalitestaajan täytyy tehdä testit erikseen ja automaatiotestaaja puolestaan voi ajaa testit samaan aikaan useammalle alustalle. Testiautomaatio soveltuu erinomaisesti usein toistettaviin testitapauksiin verrattuna manuaalitestaukseen, jossa niiden suorittaminen vie liikaa aikaa. (Hamilton, 2020c)

Sovelluksen suorituskyky on kriittinen osa-alue sovelluksen sujuvaa tuotantokäyttöä ja se on mahdollista pelkästään automaatiotestauksella. Verrattuna manuaalitestaukseen, automaatiotestaus sopii paremmin suorituskyky- ja regressiotestaukseen. Manuaalisen testauksen vahvuudet tulevat esille adhoc-, käytettävyys- ja tutkivassa testauksessa. (Hamilton, 2020c)

Taulukko 1 Testiautomaation erot manuaalitestaukseen

Mittari	Testiautomaatio	Manuaalitestaus
Yleinen määritelmä	Testit suoritetaan automaatiotyökaluilla	Testit suoritetaan ihmisen toimesta
Muutosten käsittely	Muutokset sovelluksessa vaativat usein testiskripteihin muutoksia	Muutokset eivät vaikuta testien suorittamiseen yhtä paljoa
Sijoittaminen	Vaatii sijoittamista testiautomaation työkaluihin	Manuaalitestauksen suorittaminen vaatii enemmän henkilötyövoimaa

Kirjallinen raportointi	Helpommin ja nopeammin saatavilla heti testien ajamisen jälkeen	Testaajan itse kirjoittama, ei ole heti saatavilla testauksen jälkeen
Käsittelyaika	Testit suoritetaan nopeammin automaatiotestauksella	Manuaalitestaus usein hitaampaa ja sitoo resursseja pidemmäksi aikaa
Tutkiva testaus	Ei mahdollista automaatiotestauksella	Mahdollista manuaalitestauksella
Sijoittaminen projektin alussa	Vaatii alussa enemmän aikaa, että vaadittavat työkalut saadaan käyttöön, mutta sijoitetun pääoman tuotto kuitenkin korkea	Pienempi kynnys aloittaa, mutta pääoman tuotto matalampi
Luotettavuus	Luotettavampi, koska työkalut suorittavat testauksen. Ei testauksesta aiheutuvaa uupumista.	Ei yhtä luotettavaa pitkäveteisissä tilanteissa, mahdollisuus inhimillisiin virheisiin
Rinnakkainen suorittaminen	Voidaan testata rinnakkain esim. eri sovelluksissa tai selaimissa	Mahdollista, mutta vaatii enemmän henkilötyövoimaa
Lähestymistapa	Soveltuu toistuviin testitapauksiin	Ei sovellu toistuviin testitapauksiin, parhaimmillaan kun testit täytyy suorittaa vain pari kertaa
Ihmisen havainnot	Ei huomioi ihmisen käyttäjäkokemusta	Ihmisen havainnointi käyttäjäkokemuksesta saatavilla
Suorituskykytestaus	Testataan yleensä automaatiotyökaluilla	Ei mahdollista manuaalitestauksella
Parhaat käyttökohteet	Soveltuu hyvin regressio- ja, suorituskykytestaukseen sekä usein toistettaviin testitapauksiin	Soveltuu parhaiten adhoc-, käytettävyys ja tutkivaan testaukseen
Viitekehykset	Käyttää erilaisia viitekehyksiä tehostamaan testausprosessia	Ei käytä viitekehyksiä, vaan yleisiä ohjeistuksia ja prosesseja testitapausten laatimiseen

4.1 Manuaalitestauksen käyttökohteet

Manuaalitestaus soveltuu parhaiten käyttöliittymän ja yleisen käyttäjäkokemuksen testaamiseen. Manuaalisesta testauksesta saa hyviä tuloksia, kun testaajalla on silmää yksityiskohtiin ja tuntee sovellukseen sisältyvät liiketoiminnan prosessit. Sovelluksen

testauksessa on tärkeää ottaa huomioon ominaisuuksien käytettävyys liiketoiminnan näkökulmasta ja tähän sopii manuaalisia testaustapoja kuten tutkiva-, käytettävyys- ja adhoc testaus. (*BrowserStack*, n.d.)

Manuaalista testausta on käytännössä järkevintä käyttää silloin, kun on sellaisia testitapauksia ja skenaarioita, jotka vaativat käyttäjän suoraa vuorovaikutusta sovelluksen kanssa. Manuaalisen testauksen suunnittelussa on tärkeää ottaa huomioon, miten itse testit suoritetaan ja niistä haluttavat tulokset. Kun testaus ja sen tulokset dokumentoidaan hyvin, myös ohjelmiston kehityksen prosesseja voidaan kehittää. (*Disbug*, n.d.)

Manuaalinen testaus on tietyissä tapauksissa myös joustavampaa. Jos testitapaukset ovat jo testaajien tiedossa, ne voidaan testata, jonka jälkeen tulokset ovat suullisesti raportoitavissa nopeammin. Automaattisessa testauksessa joudutaan käyttämään aikaa työkalujen valitsemiseen, käyttöönottoon, itse testiskriptien kirjoittamiseen sekä testien suorittamiseen. Testiautomaatio vaatii siis erityisen käyttöönottoprosessin verrattuna manuaalitestaukseen, jolloin investointi siihen ei välttämättä ole niin järkevää, jos kyseessä on lyhyempi projekti. Lyhyemmässä projektissa testiautomaatioon hinta ja sijoitettu pääoma voi olla liian korkea sen tarjoamaan arvoon nähden, jolloin manuaalinen testaus voi olla halvempi ja tehokkaampi vaihtoehto. (*SmartBear*, n.d.)

4.2 Automaatiotestauksen käyttökohteet

Kun sovelluksessa on sellaisia testitapauksia, joiden suorittamiseen menee kauan aikaa tai ne ovat usein toistettavia, testiautomaatio on hyvä tapa tehostaa testausprosessia. Vaikka itse testiskriptien kirjoittamiseen kuluu projektin alussa aikaa, itse testien suorittamiseen kuluu vähän aikaa. Jos sovellukseen tehdään muutoksia usein, manuaalinen testaus ei ole tarpeeksi tehokasta, koska testejä täytyy käydä läpi uudelleen ja uudelleen. (Schaffer, 2018)

Testiautomaatio auttaa myös saamaan toiminnallisuuksia nopeammin tuotantokäyttöön. Kun paljon aikaa vieviä ja usein toistettavia testejä automatisoidaan, testaustiimi voi käyttää ajan tehokkaammin muihin tehtäviin, kuten vaikeasti löydettävien bugien etsimiseen. Jos projektissa on aikataulupaineita julkaista uusia ominaisuuksia usein, testiautomaatio auttaa keventämään regressiotestauksen taakkaa, joka taas pienentää bugisen julkaisun riskiä. Kun

testiautomaatio on toteutettu onnistuneesti, päivityksiä voidaan julkaista useammin ja itse kehitysjaksoissa kestää vähemmän aikaa. (*Netguru*, n.d.)

Testien kattavuus on yksi suurimmista testiautomaation antamista eduista. Testien kattavuudella tarkoitetaan sitä, kuinka suuri osa testauksen alaisena olevasta sovelluksesta testataan nykyisillä testitapauksilla. Testiautomaatiolla testien sarjoja voidaan skaalata suuntaan tai toiseen ja suorittaa niitä samanaikaisesti eri laitteilla ja käyttöjärjestelmillä. Manuaalisella testauksella voidaan suorittaa laadukkaita testitapauksia, mutta niitä ei voi suorittaa yhtä aikaa ilman lisähenkilöitä. 100-prosenttisen testikattavuuden saaminen monimutkaiselle sovellukselle on jo muutenkin vaikeaa ja pelkällä manuaalisella testauksella se on lähes mahdotonta. Testiautomaation avulla uusien testien kirjoittamiseen jää enemmän aikaa, jonka seurauksena myös testien kattavuutta saadaan lisättyä helpommin. (*Netguru*, n.d.)

Ohjelmistovirheiden korjaamiseen voi kulua tietyissä tilanteissa enemmän aikaa, kuin on suotavaa. Jos sovelluksessa esiintyy liian usein virheitä, sen asiakkaat saattavat vaihtaa kilpailijan sovellukseen, joka puolestaan johtaa menetettyihin tuloihin. Testiautomaation avulla helpotetaan testauksen aloittamista projektin aikaisessa vaiheessa. Kun testauksen suorituksen tuloksista saadaan raportointi nopeasti, virheiden ja muiden ongelmien korjaaminen ohjelmistokehittäjien toimesta on tehokkaampaa. Kun virheet korjataan nopeasti, myös sovellus on laadukkaampi. Lisäksi automaatiotestauksessa voidaan löytää semmoisia virheitä, joita manuaalitestaajat eivät ole huomanneet. (*Netguru*, n.d.)

Suorituskykytestausta on lähes mahdotonta suorittaa manuaalisesti, joten tällaisten testien automatisointi on pakollista. Suorituskykytestauksessa voidaan simuloida suuria käyttäjä- tai datamääriä, jotka eivät ole mahdollisia testata manuaalisesti. Edistyneempien käyttöliittymätestien kuten eri selainten ja käyttöjärjestelmien välisen käyttäytymisen testaus on helpompaa automatisoida. (Schaffer, 2018)

Yllä mainitut testiautomaation hyödyt, kuten kehitys- ja testausprosessien tehostaminen ja ominaisuuksien nopeampi julkaiseminen auttavat laadukkaamman sovelluksen julkaisemisessa. Testiautomaatiolla sovelluksen eri osien testaamiseen tarvitaan vähemmän henkilötyövoimaa, joka säästää testauksesta syntyviä kustannuksia. Kehitysprojekteissa aika

ja raha ovat kriittisiä resursseja ja testiautomaatiolla testauksen kustannuksia voidaan laskea, ja lopputuloksena on silti laadukas tuote. Jos manuaalisia testaustehtäviä halutaan siirtää automaattisiksi, asianmukaiset testitapaukset täytyy analysoida, jotta myös sijoitetun pääoman tuotto voidaan määritellä. (*Netguru*, n.d.)

5 Testiautomaation kustannukset

Tässä pääluvussa selitetään, miten testiautomaation kustannukset ja siitä koostuvat tuotot voidaan laskea. Testiautomaatio on hyvä tapa tehostaa testausprosesseja, mutta siihen menee yleensä enemmän etukäteissijoituksia verrattuna manuaalitestaukseen. Testiautomaation käyttö tulee olla perusteltua ja siihen vaikuttaa myös kokonaiskustannukset ja saatava tuotto. Testiautomaatioon käyttöönotto ei ole organisaatiolle pieni toimenpide ja sille voidaan tietyissä tapauksissa vaatia jopa oma projekti. Testiautomaatio vaatii oman alueensa asiantuntijoita, uusien työkalujen käyttöönoton/kehittämisen ja mahdollisen lisäinfrastruktuurin. (*Kimputing*, 2021)

5.1 Sijoitetun pääoman tuotto eli ROI

Testiautomaatiosta saatava hyöty mitataan sijoitetun pääoman tuotolla eli ROI:llä (Return on Investment). Ennen kuin testiautomaatio otetaan käyttöön, on tärkeää, että sijoitetun pääoman tuotto lasketaan. Tähän tarvitaan testiautomaatioon menevät kulut ja ennustettavissa olevat tuotot. (*Kimputing*, 2021)

ROI:n antamat luvut auttavat organisaatiota päättämään, onko testiautomaation käyttöönotto taloudellisesti kannattavaa yritykselle. Laskennassa täytyy punnita testiautomaation hyötyjä ja riskejä sekä lyhyellä että pitkällä aikavälillä. ROI:n avulla voidaan laskea, kuinka monella prosenttiyksiköllä testiautomaatioon sijoittaminen maksaa itseään takaisin verrattuna manuaalitestaukseen. Tarkoituksena on nähdä, onko testiautomaatio oikeasti kannattava lisä tai vaihtoehto manuaaliselle testaukselle. Jos ROI antaa positiivisen tuloksen, testiautomaation käyttö on perusteltua ja päättävien tahojen on helpompi päättää itse investoinnista. (*Relevant Software*, 2021)

5.2 ROI:n laskeminen

Yksinkertainen tapa aloittaa sijoitetun pääoman tuoton (ROI) laskeminen on tuottaa ennuste siitä, kuinka paljon testaustehtävien muuttaminen automaattisiksi säästää aikaa. Esimerkkitilanteena koko järjestelmän manuaaliseen testaamiseen tarvitaan viisi henkilöä ja 100 tuntia. Kaksi testiautomaatioasiantuntijaa saisi kirjoitettua testit automaattisiksi

samassa ajassa, joka säästää rahaa. Tämän tyyppinen laskelma on kuitenkin yksinkertainen ja siinä ei oteta huomioon testien vaatimaa ylläpitoa. Testien määrä kasvaa kehitysprojekteissa uusien toimintojen myötä ja olemassa olevia testejä joudutaan ylläpitämään. Testit ovat ajan saatossa alttiita muutoksille, joten on tärkeää valita sopivat työkalut testien tehokkaaseen laatimiseen ja ylläpitämiseen. Tällöin testaajat voivat keskittyä enemmän testikattavuuden kasvattamiseen eikä testien ylimääräiseen ylläpitämiseen. (*Kimputing*, 2021)

ROI:n kunnolliseen laskemiseen tarvitset testiautomaatiosta saadut investointihyödyt ja -kustannukset. Investoinnin hyötyjen laskemiseen tarvitset käytännössä säästöt ja ne riskit, jotka vähentyvät, jos testiautomaatio otetaan käyttöön. Investoinnin hyötyihin sisältyy mm. säästetyt tunnit verrattuna manuaalitestaukseen, säästetyt kulut kuten pienemmän henkilötyövoiman käyttäminen ja vianmääritykseen käytetyt kulut. Investoinnin kustannuksiin kuuluu se aika, jota joudut käyttämään testiautomaation käyttöönottoon. Tähän kuuluu myös testiautomaation asiantuntijoiden hankkimisen hinta tai kolmannen osapuolen testauspalvelut, automaattisten testitapausten kirjoittamisen ja tarvittavien työkalujen käyttöönoton kesto. Lisäksi kustannuksiin kuuluu työnkulkujen ja viitekehysten ylläpitoon käytetyt kulut. (*Relevant Software*, 2021)

Kun investoinnin hyödyt ja kustannukset ovat laskettu, voit käyttää ROI:n laskemiseen kaavaa, jossa on kolme eri vaihetta. ROI:n laskemiseen käytetty kaava on esitetty myös Kuva 1 ROI:n laskentakaava. Ensin vähennetään investoinnin kustannukset investointihyödyistä ja tästä saatu summa jaetaan kustannuksilla. Jakolaskusta saatu summa luku kerrotaan lopuksi sadalla. Tällöin olet laskenut testiautomaatiosta odotetun tuoton prosentteina. On kuitenkin huomioitava, että ROI:n laskemisessa on asioita, joita ei näe paljaalla silmällä. Kun investoinnin hyötyjä ja kustannuksia lasketaan, siihen on laskettava myös työnkulkujen muutokset, joita testiautomaation käyttö aiheuttaa. Testiautomaation käyttöönotto ei myöskään tarkoita sitä, että voisit luopua manuaalisesta testauksesta kokonaan, koska tietyt skenaariot vaativat testiautomaatiosta huolimatta myös manuaalista testausta. (*Relevant Software*, 2021)

Muita huomioitavia asioita ROI:n laskennassa on esimerkiksi mitä testitapauksia automatisoidaan ja kuinka kauan niiden implementoiminen säästää aikaa. On harkittava myös regressiotestaukseen käytettyä aikaa. Lisäksi on hyvä tarkastella sellaisia bugeja, jotka

ovat tulleet testauksen jälkeen tuotantokäytössä vastaan. Kun estät tiettyjä bugeja pääsemästä tuotantoon, voit käyttää tästä aiheutuvia säästöjä ROI:n laskemiseen. Testiympäristöihin käytetyt kulut on myös hyvä huomioida ROI:n laskemisessa, koska automaation avulla testidatan keräämiseen ja testien suorittamiseen eri alustoilla kuluu vähemmän aikaa ja rahaa. Automaatio voi myös auttaa tilanteessa, jossa testaaja lähtee töistä, koska automaation avulla voidaan estää riippuvuus yksittäisiin testaajiin. Manuaalitestaajan lähdön tapauksessa voidaan menettää teknistä tietämystä ja uuden manuaalitestaajan perehdyttämisessä voi kulua kauemmin aikaa. (*Relevant Software*, 2021)

Yllä mainittu kaava on hyvä lähtökohta ROI:n laskemiseen, mutta on myös sellaisia seikkoja, jotka saattavat vaikuttaa laskutuloksen tarkkuuteen. Tiettyjä asioita, kuten bugit, jotka eivät ilmaannu, jos automaatio otetaan käyttöön, on vaikea arvioida etukäteen. Säästettyjen tuntien laskeminen voi vaikuttaa yksinkertaiselta, mutta testisarjojen ylläpitokustannuksia on vaikea arvioida etukäteen, koska niihin käytetty aika voi nousta projektin edetessä. Lisäksi testiautomaatioon soveltuvien asiantuntijoiden perehdyttäminen ja kouluttaminen ei ole välttämättä kiinteä kulu, vaan se voi muuttua yksilökohtaisesti. Testiautomaatiolla et voi myöskään saada varmuudella 100 %:n testikattavuutta, joka johtaa siihen, että virheitä voi lopulta tulla vastaan ja näiden korjaamiseen kuluu rahaa. On myös huomioitava, että automaatiotestaus ei sulje manuaalista testausta kokonaan pois, eli laadunvarmistustiimissä on oltava testaajat kaikkiin tarvittaviin testausrooleihin. (*Relevant Software*, 2021)

Kuva 1 ROI:n laskentakaava

$$ROI = \frac{\text{Hyödyt} - \text{Kustannukset}}{\text{Kustannukset}} x100$$

6 Testiautomaation vastuut ja työrutiinit

Testiautomaation asiantuntijalla on useita tehtäviä, jotka vaikuttavat päivittäisiin työrutiineihin ja ne ovat usein sellaisia, jotka eroavat manuaalisen testaajan tehtävistä. Tämän kappaleen päätarkoituksena on käydä läpi millaisia vastuita ja rutiineja testiautomaatio tuo työpäiviin ja mitä testiautomaation käyttöönottoon konkreettisesti tarvitaan.

Testiautomaation aloittamista varten joudutaan ottamaan muutamista perusasioista selvää. Yksi tärkeimmistä asioista on analysoida toistuvimmat ja eniten aikaa kuluttavat testitapaukset. Analysoinnin avulla saadaan selville, ovatko tapaukset niin toistuvia, että ne voitaisiin automatisoida. Kun tiedetään mitä automatisoidaan, täytyy selvittää, miten automatisoidaan. Jos haluat esimerkiksi automatisoida Help Deskin sulkemien tikettien raportoinnin omaan sähköpostiin, joudut selvittämään mikä menetelmä on tehokkain sen automatisointia varten. Kun tiedät, miten automatisoit testin, joudut etsimään oikean työkalun, jolla testit ajetaan. Oikealla työkalulla tehostat automatisointiin käytettäviä työnkulkuja ja voit myös integroitua muiden ulkoisten sovellusten kanssa. Kun testejä ajetaan määritetyllä työkalulla, on varmistettava, että testien ajaminen suoritetaan ongelmitta. Jos ongelmia ilmenee, ne korjataan ja testit ajetaan uudelleen. (*Zoho Blog*, 2022)

Automaatiotestaajan päävastuisiin kuuluu itse testien kirjoittaminen. Testeihin liittyvien skenaarioiden testaukseen saattaisi mennä manuaaliselta testaajalta useita tunteja tai jopa päiviä. Automaatio antaa tällöin manuaalitestaajille aikaa keskittyä muihin tärkeisiin testaustehtäviin, kuten tutkivaan testaukseen ja käyttäjäkokemustestaukseen. Tutkiva- ja käyttäjäkokemustestaus ei myöskään ole mahdollista suorittaa automaattisella testauksella, jolloin testausroolien tehtävät painottuvat ajankäytöllisesti oikein. (Commonwealth of Australia, 2022)

Yllä mainitut vastuut vaikuttavat testiautomaatioasiantuntijan päivittäisiin työrutiineihin. Työrutiinit riippuvat aina roolista ja projektista, mutta yleisimpiin päivittäisiin työtehtäviin kuuluu mm. testiskriptien kirjoittaminen, itse testien ajaminen, tulosten tarkistaminen ja mahdollisten vikojen analysointi. Automaatioasiantuntija tekee tiivistä yhteistyötä projektin muiden testaajien ja kehittäjien kanssa, jotta automaattisista testeistä saadaan

mahdollisimman tehokkaita. Testitulosten ja muutosten dokumentointi sekä raportointi on myös hyvin tärkeää ja yleensä testiautomaatioasiantuntija on näistä vastuussa. Testien mahdollisista vioista on hyvin kriittistä raportoida projektiryhmälle, jotta testauksen tilasta pysytään ajan tasalla. Päivittäisiin tehtäviin voi kuulua myös tapaamiset kuten daily-palaverit, jossa tiimiä päivitetään omien tehtävien etenemisestä ja mahdollisista ongelmista. (JanbaskTraining, 2022)

Muita kriittisiä vastuita on varsinkin testiautomaation käyttöönoton ensimmäisissä vaiheissa kuten käytettävien testitapausten tunnistaminen ja valitseminen. Testiautomaatiota varten laaditaan myös strategia, joka dokumentoidaan yksityiskohtaisesti. Testausympäristöt ja viitekehykset suunnitellaan ja konfiguroidaan tiimin kesken, jotta testien ajaminen onnistuu sujuvasti. Projektin edetessä testejä ajetaan sekä olemassa olevia testejä ylläpidetään ja parannetaan. Lisäksi uusia testitapauksia laaditaan ja mahdollisia virheellisiä testejä korjataan. Vuorovaikutus asiakkaiden kanssa on myös tärkeää, jotta mahdollisia ongelmatilanteita voidaan ratkaista ja testauksen tilanteesta pysytään ajan tasalla. (Intellipaat Blog, 2019)

Automaattisten testien määrittäminen vaatii usein myös yhteistyötä manuaalitestaajien kanssa, jotta testeille saadaan tarvittava data. Automaattisilla testeillä saadaan hoidettua yleisimmät rutiininomaiset perustapaukset, kun taas manuaalitestaajat voivat hoitaa monimutkaisempia skenaarioita. Jos automaattisissa testeissä on jotain vikaa, ne toistetaan omissa sisäisissä ympäristöissä, jonka jälkeen vaatimuksista ja koodista tarkistetaan mikä voisi olla ongelman aiheuttaja. Myös manuaaliset testaajat voivat suorittaa testejä uudelleen sen perusteella mitä löydöksiä automaatiotestaajat ovat tehneet ongelman aiheuttajasta. Kun juurisyy on löydetty, automaattisiin testeihin tehdään muutoksia niin että ne menevät läpi seuraavilla kerroilla. Vikojen analysoinnissa voi tulla ilmi, että ongelma on suurempi kuin on kuviteltu, jolloin se voi aiheuttaa sivuvaikutuksia muualle järjestelmään. Testiautomaation asiantuntijat voivat löytää keinoja tällaisten ongelmien ratkaisemiseen, jotta testauksesta saadaan tarvittava tulos. (Commonwealth of Australia, 2022)

7 Menetelmät

Tätä työtä on kirjoitettu noin viikon sprinteissä Scrum-projektinhallintamenetelmää hyödyntämällä. Ennen työn aloittamista jokainen pääluku on jaettu tehtäviksi Plannerissa omiin viikkokohtaisiin lohkoihin. Työn aikana on laadittu myös päiväkirjaa erilaisista havainnoista, kuten ongelmakohdista omassa työskentelyssä ja lisättävistä asioista. Päiväkirja on jaoteltu viikkokohtaisiin havaintoihin. Päälukuja kirjoittaessa olen pohtinut mitä päälukuun voisi vielä lisätä ja merkannut sen päiväkirjaan backlogiksi.

Kun pääluku on valmistumassa, kirjoitan siihen vielä lisäykset sen perusteella mitä olen merkannut päiväkirjaan. Jos päiväkirjaan on merkattu yleisiä asioita työskentely- tai kirjoitustavoista, olen ottanut ne erityisen huomion alle seuraavan luvun tekemisessä. Pääluvun valmistuessa olen lukenut sen vielä joitakin kertoja läpi ja korjannut mahdolliset kirjoitus- ja kielivirheet. Tämän jälkeen olen todennut luvun valmiiksi ja merkannut lukukohtaisen tehtävän Plannerissa valmiiksi. Ajanhallinta on sujunut melko hyvin, paria pidemmäksi venynyttä päälukua lukuun ottamatta, joiden osalta kirjoittaminen on saattanut venyä hieman viikkoa pidemmäksi.

8 Robot Frameworkin ja Seleniumin käyttöönotto

Tästä luvusta alkaa työn toiminnallinen osuus. Toiminnallisen osuuden on tarkoitus vahvistaa teoriaosassa esitettyjä asioita testiautomaation tehokkuudesta sekä antaa tarvittavat tiedot testiautomaation käyttöönottoa varten. Kun tarvittavat asennukset ja automatisoitujen käyttöliittymätestien kirjoittaminen ja suorittaminen esitetään, lukija voi hyödyntää tietoja esimerkiksi omien testiautomaatiotaitojen kehittämiseen.

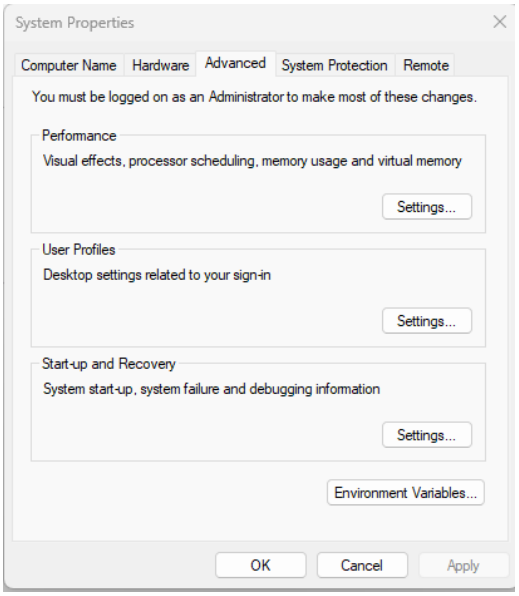
Luvussa kerrotaan tarvittavat asennukset, joilla pääsee alkuun UI-automaatiotestien suorittamisessa. Käyttöön asennetaan ohjelmointiympäristö Python, PyCharm, Robot Framework, Selenium ja ChromeDriver. Luvun tarkoitus on antaa selkeä ohjeistus kaikista asennettavista ohjelmista ja lisäosista, jotta itse testien kirjoittaminen ja suorittaminen voidaan aloittaa seuraavassa luvussa. Asennukseen vaadittavia konfigurointeja kuvataan tekstillä, kuvakaappauksilla ja komennoilla.

8.1 Pythonin asennus

Asennetaan ensin viimeisin versio Pythonista viralliselta verkkosivustolta https://www.python.org/downloads/. Viimeisin Pythonin LTS-versio työtä laadittaessa oli 3.11.3. LTS-versiolla tarkoitetaan sovelluksen versiota, jolla on pitkäaikainen tuki (long-term support). Pythonin eri versioiden elinkaaresta on tietoa Pythonin verkkosivustolla https://devguide.python.org/versions/. Kun asennus aloitetaan, kannattaa valita ”Add python.exe to PATH”, jolloin sitä ei tarvitse tehdä manuaalisesti. Otetaan talteen myös asennuskansio, koska sitä tarvitaan myöhemmin. Huom. yksityiskohtaisempi versio tästä ohjeesta on liitteessä 2.

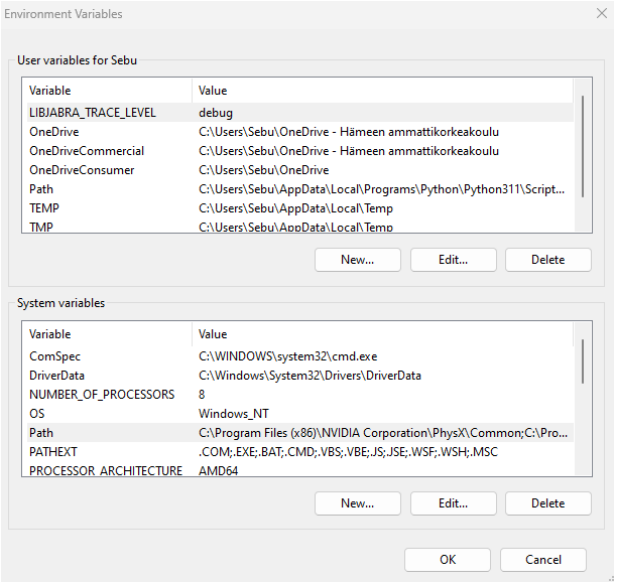
Pythonin asennuksen jälkeen lisätään Pythonin skriptikansio ympäristömuuttujaksi. Kopioidaan ensin Pythonin Scripts -kansion polku `C:\Users\User\AppData\Local\Programs\Python\Python311\Scripts`. Avataan tämän jälkeen resurssienhallinta, painetaan vaihtonäppäin pohjaan ja klikataan hiiren oikealla `This PC` vasemmalta ja valitaan luettelosta `Properties`. Kun klikataan `Advanced system settings`, `System Properties` -näkymä aukeaa, josta valitaan oikeasta alakulmasta `Environment Variables` (Kuva 2).

Kuva 2 Järjestelmäominaisuudet

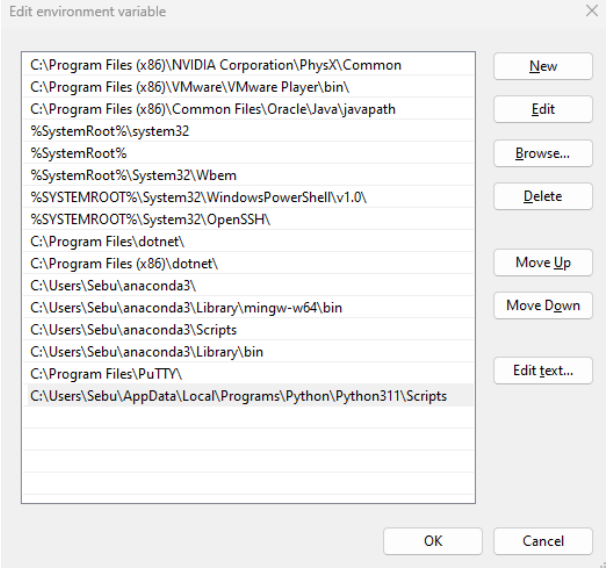


Kun ympäristömuuttujien ikkuna aukeaa, alemmasta `System Variables` -luettelosta valitaan `Path` ja klikataan `Edit`-painiketta (Kuva 3). Tämän jälkeen aukeaa `Edit Environment Variable` -ikkuna (Kuva 4), jossa klikataan `New` ja kopioidaan `Scripts` -kansion polku. Kaikki ikkunat voidaan lopuksi sulkea OK-painikkeella. Asennuksen voi tarkistaa lopuksi avaamalla komentorivin (command prompt) ja ajamalla komennon `python -version`. Myös pip -versio kannattaa tarkistaa `pip -version` komennolla (Kuva 5).

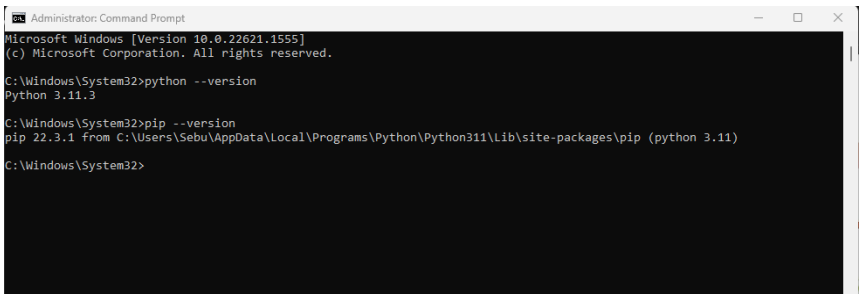
Kuva 3 Ympäristömuuttujat



Kuva 4 Ympäristömuuttujan muokkaus



Kuva 5 Pythonin version tarkistaminen



8.2 Pycharm IDE asennus

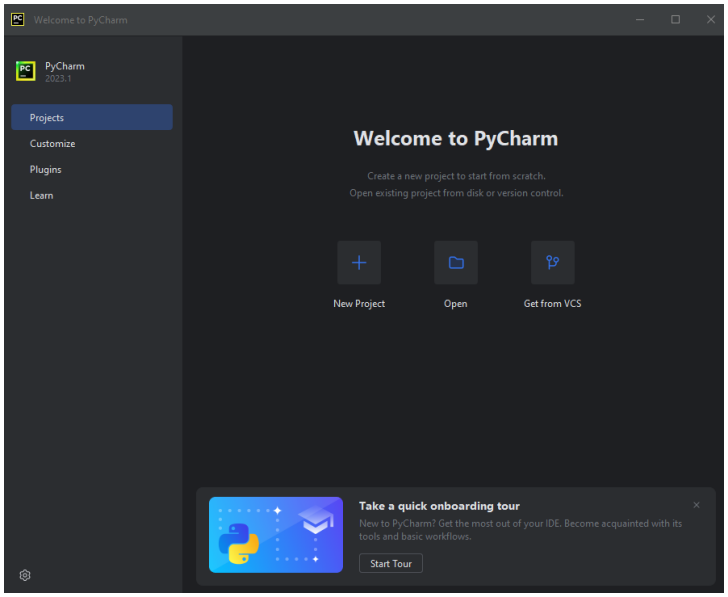
Ohjelmointiympäristöksi on valittu PyCharm. PyCharmin etuna on laaja tuki Pythonille ja Robot Frameworkin lisäosille. Se asennetaan osoitteesta https://www.jetbrains.com/pycharm/download/#section=windows. Valitaan Community-versio ja klikataan Download -painiketta. Huom. yksityiskohtaisempi versio tästä ohjeesta on liitteessä 3.

Kun asennus aloitetaan, valitaan kansio johon PyCharm asennetaan. Mikään valinnoista ei ole pakollinen `Installation Options` -näkymässä. Kannattaa valita ainakin työpöytäkuvakkeen luominen. `Choose Start Menu Folder` -osiossa voidaan painaa `Next`.

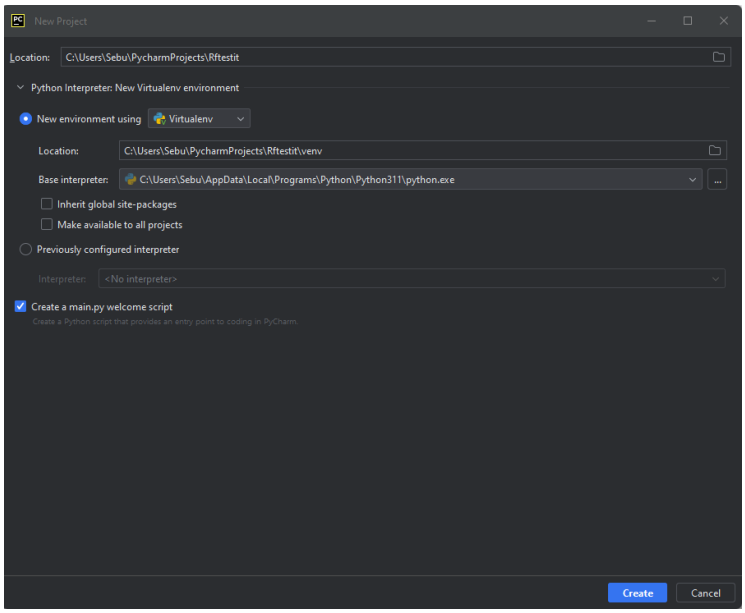
Asennus kestää hetken ja lopussa voidaan valita `Run PyCharm` ja klikata `Finish`. Kun PyCharm aukeaa, käyttäjäehdot täytyy hyväksyä.

PyCharm on nyt asennettu ja avattu. `New Project` -painikkeesta voidaan luoda uusi kansio myöhemmin kirjoitettavia testejä varten (Kuva 6). Kun uuden projektin näkymä aukeaa, kirjoitetaan kansion nimi tiedostopolkuun ja klikataan `Create`. Esimerkissä kansion nimi on `Rftestit` (Kuva 7). Tämän jälkeen projekti on luotu testejä varten.

Kuva 6 PyCharmin ensinäkymä



Kuva 7 PyCharmin projektikansion luominen



8.3 Selenium ja Robot Framework asennus

Seleniumin asentaminen on yksinkertaista. Avataan komentorivi ja ajetaan komento `pip install selenium`. Asennuksen lopussa näkyy listaus kaikista asennetuista paketeista. Huom. yksityiskohtaisempi versio tästä ohjeesta on liitteessä 4.

Jos edellisessä osuudessa käytetty komentorivi ei ole auki, se täytyy avata uudelleen. Kun komentorivi on auki, Robot Frameworkin asennus suoritetaan komennolla `pip install robotframework`. Asennuksen lopussa näkyy listaus kaikista asennetuista paketeista. Asennus voidaan tarkastaa komennolla `pip show robotframework`. Kaikki asennetut paketit voidaan tarkastaa `pip list` komennolla (Kuva 8). SeleniumLibrary puuttuu listauksesta, joten se asennetaan seuraavaksi.

Kuva 8 Pip list -komento

```
C:\Windows\System32>pip list
Package          Version
---------------- ---------
async-generator  1.10
attrs            23.1.0
certifi          2022.12.7
cffi             1.15.1
exceptiongroup   1.1.1
h11              0.14.0
idna             3.4
outcome          1.2.0
pip              22.3.1
pycparser        2.21
PySocks          1.7.1
robotframework   6.0.2
selenium         4.8.3
setuptools       65.5.0
sniffio          1.3.0
sortedcontainers 2.4.0
trio             0.22.0
trio-websocket   0.10.2
urllib3          1.26.15
wsproto          1.2.0
```

Jos edellisessä osuudessa käytetty komentorivi ei ole auki, se täytyy avata uudelleen. Suoritetaan SeleniumLibraryn asennus komennolla `pip install robotframework-seleniumlibrary`. Asennuksen lopussa näkyy listaus kaikista asennetuista paketeista. Vapaaehtoisesti uusin pip -versio voidaan asentaa komennolla `python.exe -m pip install –upgrade pip`. Komennolla `pip list` voidaan todeta, että SeleniumLibrary on asennettu (Kuva 9).

Kuva 9 SeleniumLibraryn asennuksen tarkastaminen

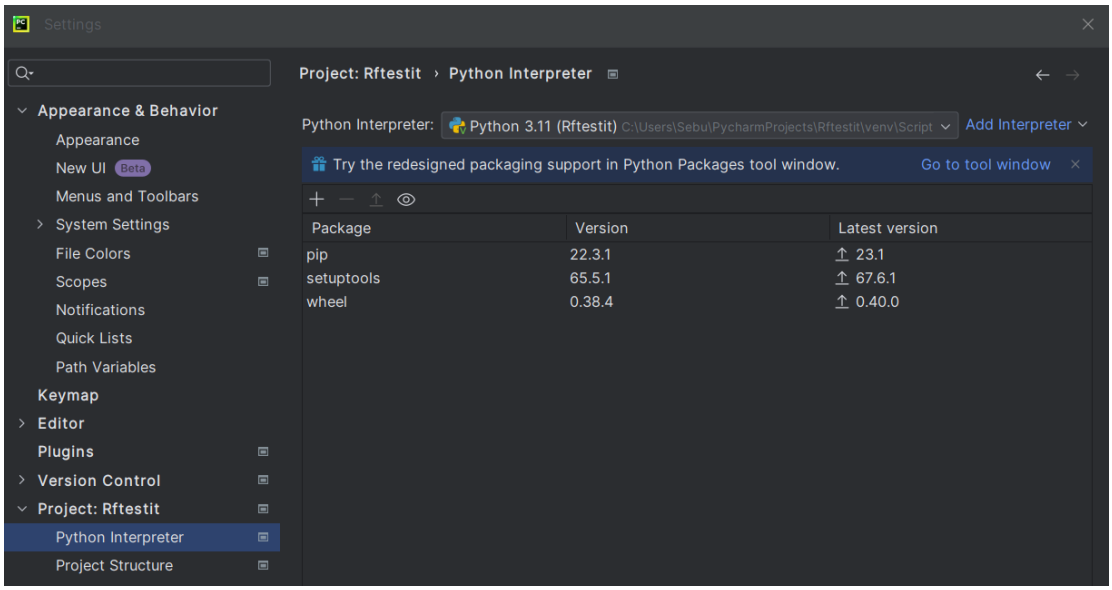
```
C:\Windows\System32>pip list
Package                       Version
----------------------------- ---------
async-generator               1.10
attrs                         23.1.0
certifi                       2022.12.7
cffi                          1.15.1
exceptiongroup                1.1.1
h11                           0.14.0
idna                          3.4
outcome                       1.2.0
pip                           22.3.1
pycparser                     2.21
PySocks                       1.7.1
robotframework                6.0.2
robotframework-pythonlibcore  4.1.2
robotframework-seleniumlibrary 6.0.0
selenium                      4.8.3
setuptools                    65.5.0
sniffio                       1.3.0
sortedcontainers              2.4.0
trio                          0.22.0
trio-websocket                0.10.2
urllib3                       1.26.15
wsproto                       1.2.0
```

8.4 Lisämääritykset PyCharmissa

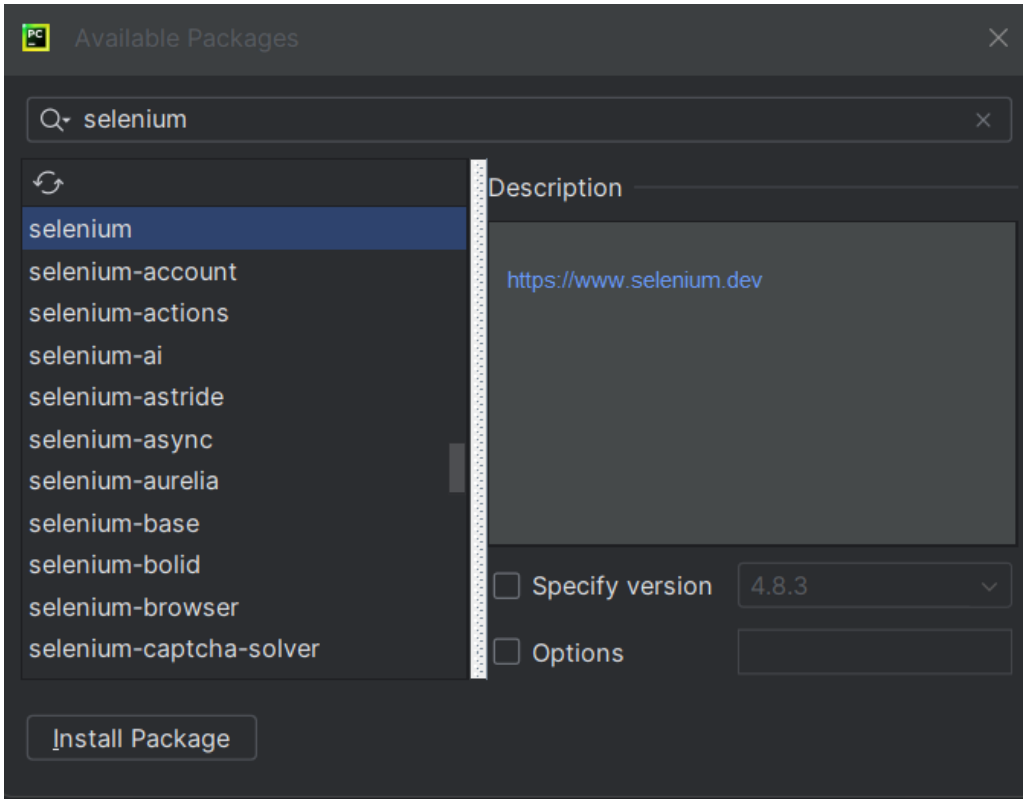
Seuraavaksi lisätään tieto Robot Frameworkin ja Seleniumin käyttämisestä projektiimme PyCharmissa. Tämä aloitetaan valitsemalla asetukset PyCharmin oikeassa yläkulmassa. Huom. projektikansion luomisvaiheessa syntynyt tiedosto main.py voidaan poistaa!

Asetuksissa valitaan luomamme projekti vasemmalta ja klikataan `Python Interpreter`. Jatketaan tämän jälkeen klikkaamalla Plus -painikkeesta (Kuva 10). Tämän jälkeen aukeaa `Available Packages` -ikkuna. Kirjoitetaan hakukenttään ”selenium”, valitaan se listauksesta ja klikataan lopuksi `Install package` (Kuva 11). Toistetaan sama Robot Frameworkin ja SeleniumLibraryn paketeilla. Lopuksi voidaan todeta asetuksissa olevasta `Python Interpreter` -listauksesta, että Robot Framework, Selenium ja SeleniumLibrary ovat asennettu onnistuneesti projektin käyttöön.

Kuva 10 PyCharmin projektin Python -asetukset

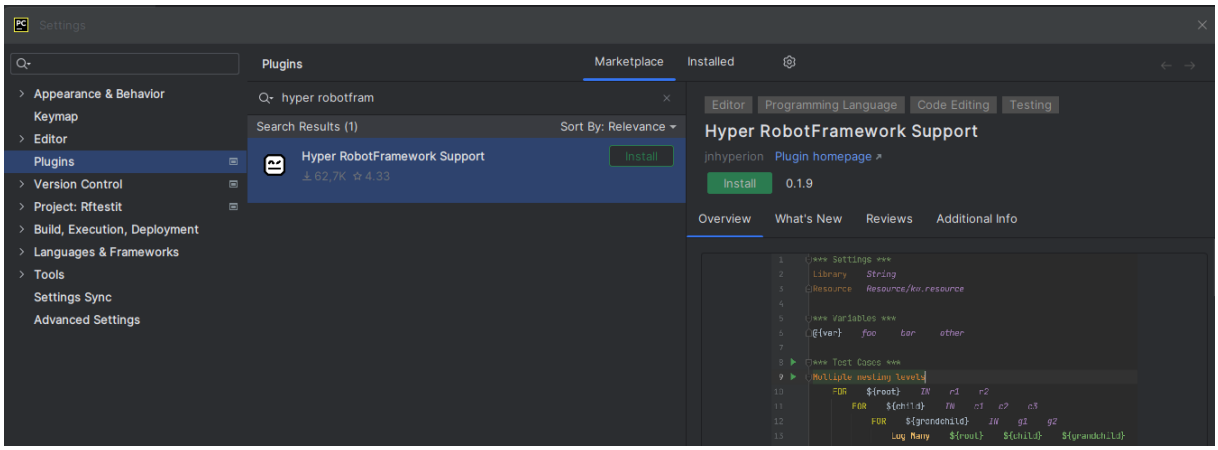


Kuva 11 Selenium-paketin asennus PyCharmissa



Asennetaan myös tuki Robot Frameworkin syntaksille, joka helpottaa testien kirjoittamista. Siirrytään PyCharmin asetuksiin ja valitaan `Plugins`, kirjoitetaan hakukenttään ”hyper robotframework support”, valitaan se ja klikataan `Install` -painikkeesta (Kuva 12). Asennuksen jälkeen kannattaa käynnistää PyCharm uudelleen.

Kuva 12 Hyper RobotFramework Support -lisäosan asennus PyCharmiin

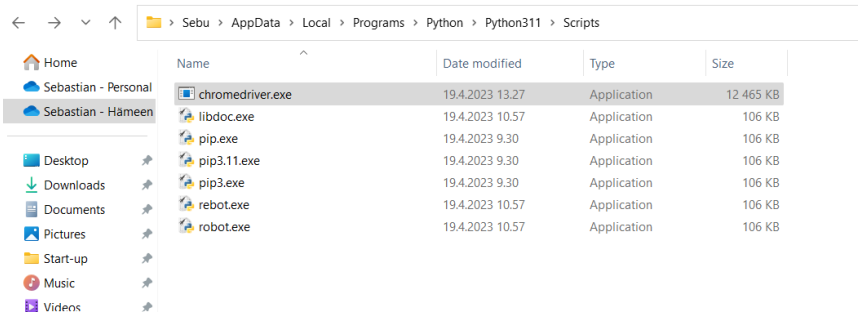


8.5 ChromeDriverin asennus

Viimeinen asia, joka asennetaan, on ChromeDriver. Se asennetaan osoitteesta https://chromedriver.chromium.org/downloads. ChromeDriverin versio valitaan selaimen version mukaan. Selaimen versio voidaan tarkistaa Chromen asetuksissa `Tietoja Chromesta` tai `About Chrome` -valikossa. Huom. yksityiskohtaisempi versio tästä ohjeesta on liitteessä 5.

Valitaan asennuspaketti `chromedriver_win32.zip`, kun käytetään Windows-käyttöjärjestelmää ja puretaan sen sisältö, jotta päästään käsiksi asennukseen tarvittavaan .exe-tiedostoon. Purettu kansio voidaan viedä esimerkiksi C-levyn juureen omaan kansioon. Navigoidaan resurssienhallinnassa purettuun kansioon ja kopioidaan chromedriver.exe Pythonin `Scripts` -kansioon (Kuva 13).

Kuva 13 ChromeDriverin ajotiedoston vienti Pythonin kansioon



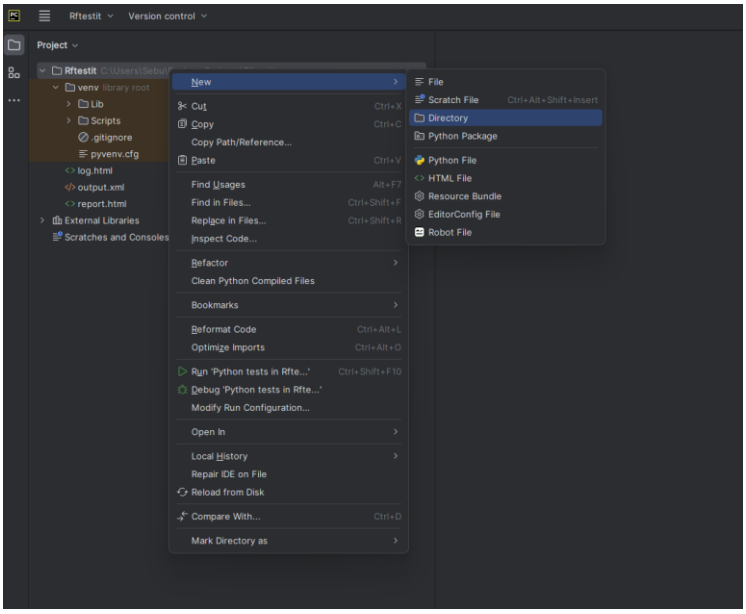
9 Testien ohjelmointi Robot Frameworkilla ja Seleniumilla

Tässä luvussa kuvataan erilaisia käyttöliittymän automaatiotestejä. Testeissä hyödynnetään Robot Frameworkiin sekä Seleniumiin tarkoitettuja esimerkkisivustoja. Edellisen luvun avulla kaikki tarvittavat asennukset on tehty, jotta itse testien kirjoittaminen ja suorittaminen päästään aloittamaan. Testien kirjoittaminen aloitetaan yksinkertaisista tapauksista, jotta testien kirjoittamiseen saa näppituntumaa. Testitapaukset ovat jaettu omiin alalukuihin. Perustapauksista saatu oppi käytetään myös laajempaan testitapaukseen. Kaikkiin testeihin vaadittavat koodit tuodaan työssä erikseen näytille.

9.1 Kansion ja tiedoston luonti

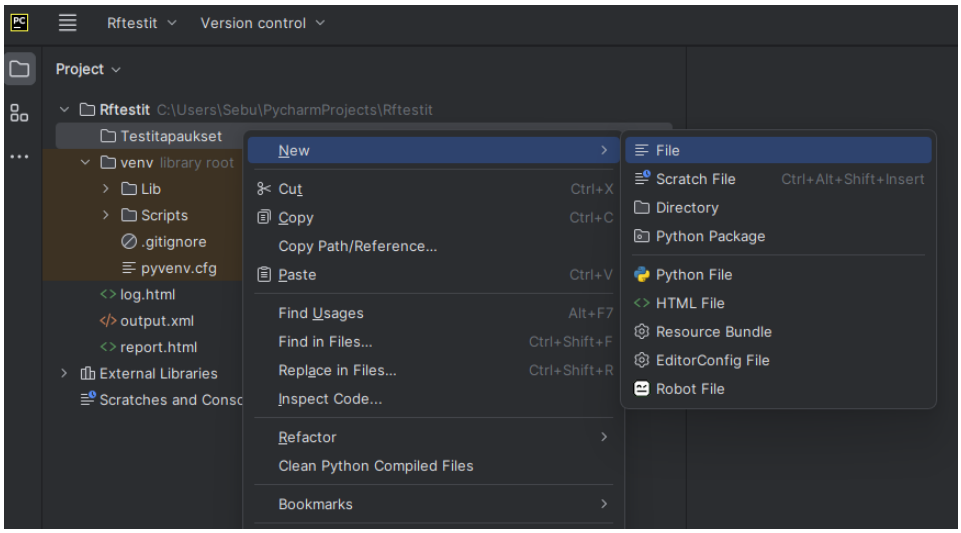
Aloitetaan testien kirjoittaminen avaamalla PyCharm ja luomalla uusi kansio projektille. Klikataan hiiren oikealla projektin nimestä, valitaan valikosta `New` ja `Directory`. Annetaan kansion nimeksi esimerkiksi ”testitapaukset” (Kuva 14).

Kuva 14 Kansion luominen PyCharmissa



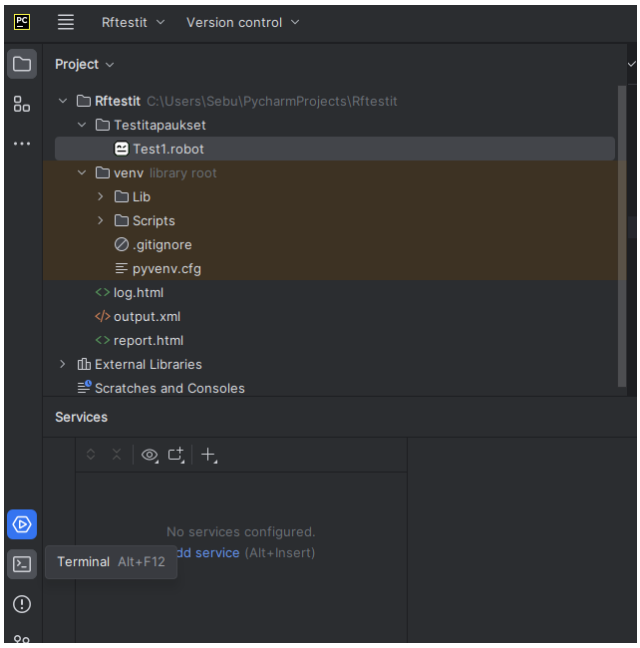
Luodaan ensimmäinen Robot-tiedosto klikkaamalla hiiren oikealla kansion nimestä, valitaan valikosta `New` ja `File`. Annetaan tiedoston nimeksi esimerkiksi `Test1.robot` (Kuva 15). Tämän jälkeen testien kirjoittaminen voidaan aloittaa.

Kuva 15 Tiedoston luominen PyCharmissa



Testien ajamiseen tarvitaan terminaalia, joka avataan vasemmasta alanurkasta `Terminal` -painikkeesta tai painamalla yhtä aikaa ALT ja F12 näppäimiä (Kuva 16). Terminaali toimii käytännössä samalla tavalla kuin komentorivi. Tarkistetaan tässä vaiheessa kansiorakenteen polku terminaalissa. Polku on oletuksena kansiorakenteen juuressa, joten kirjoitetaan komentoriville `cd Testitapaukset`. Näin päästään aikaisemmin luotuun testitapausten kansioon.

Kuva 16 Terminaalin avaaminen PyCharmissa



9.2 Ensimmäisen testitapauksen kirjoittaminen ja ajaminen

Ensimmäisenä kirjoitetaan asetus, jolla otetaan Selenium-kirjasto käyttöön testejä varten. Kirjoittaessa on huomioitava, että `Library` -sanan jälkeen joudutaan painamaan välilyöntiä kaksi kertaa tai sarkainta kerran. Sarkaimen tai välilyönnin painallukset täytyy muistaa jatkossa muidenkin avainsanojen kirjoittamisen jälkeen. Aloitetaan myös muuttujien lista (variables), mutta jätetään se toistaiseksi tyhjäksi, koska sitä tarvitaan myöhemmin.

Kirjoitetaan seuraavaksi ensimmäiset rivit testiin. Kun alkuun on kirjoitettu `*** Test Cases ***`, painetaan Enteriä rivin vaihtoa varten ja kirjoitetaan testitapauksen nimeksi esimerkiksi `OpenBrowser`. Painetaan Enteriä ja sarkainta, jonka jälkeen voidaan kirjoittaa avainsana selaimen avaamista varten. Koodia kirjoitettaessa on muistettava sarkaimen painallukset, eli kirjoitetaan ensin `open browser`, painetaan sarkainta, kirjoitetaan osoite eli https://robotsparebinindustries.com/, painetaan jälleen sarkainta ja kirjoitetaan selaimen nimi eli `chrome`. Kirjoitetaan lopuksi `Maximize Browser Window` -avainsana (Ohjelmakoodi 1). Hyper Robotframework Support -lisäosalla PyCharm ehdottaa automaattisesti eri avainsanoja. Voit navigoida eri ehdotuksia ylös ja alas nuolinäppäimillä ja painaa oikean vaihtoehdon kohdalla sarkainta, jolloin koodin kirjoittaminen on jouhevampaa. Voit myös tuoda ehdotukset esiin painamalla CTRL ja välilyönti näppäimiä, jos ne eivät ilmaannu.

Ohjelmakoodi 1 Selaimen avaaminen ja ikkunan suurentaminen

```
*** Settings ***
Library  SeleniumLibrary

*** Variables ***


*** Test Cases ***
OpenBrowser
    open browser    https://robotsparebinindustries.com/    chrome
    Maximize Browser Window
```

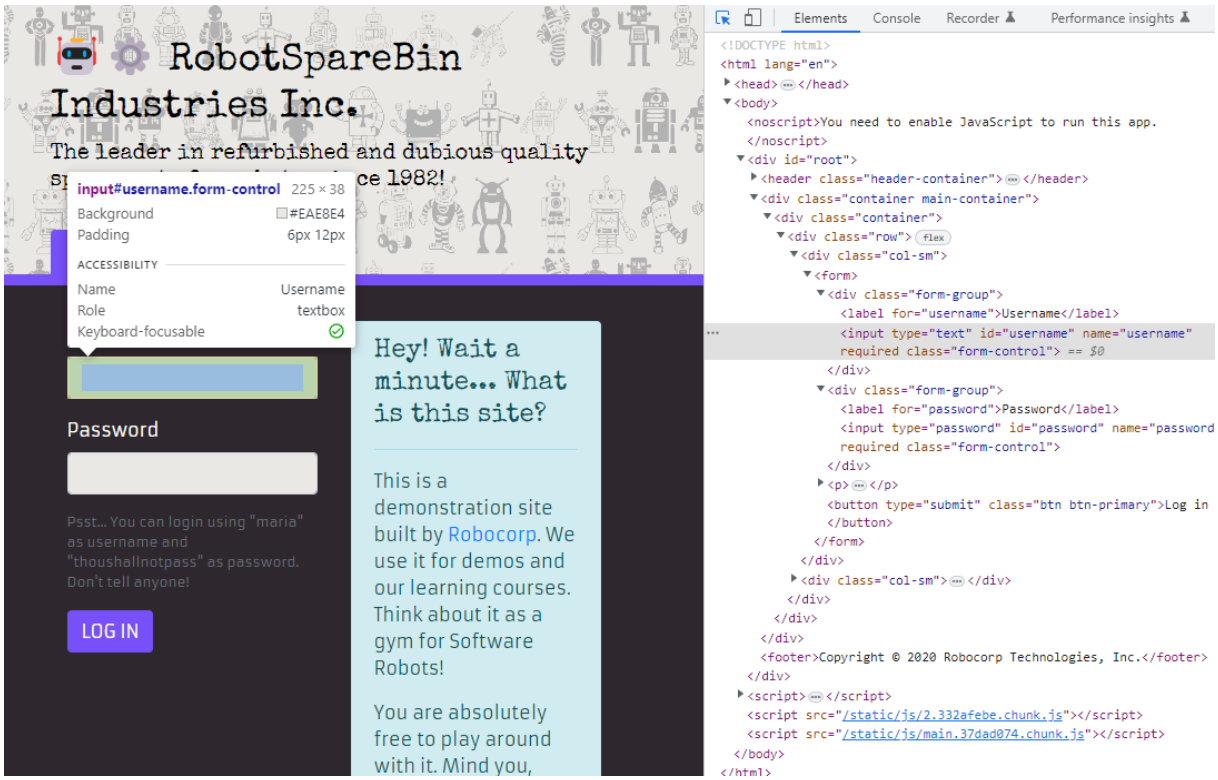
Testisivulla on mahdollisuus sisäänkirjautumiseen, jota käytetään ensimmäisessä esimerkissä. Kirjoitetaan siis ensin käyttäjätunnus ja salasana. Aloitetaan `Input Text` -avainsanalla ja sarkainta painamalla. Tämän jälkeen haetaan tekstikentän tunnus eli ID. Avataan testissä käytetty https://robotsparebinindustries.com/#/ selaimeen, painetaan F12-näppäintä ja klikataan kuvassa näkyvästä hiiren kuvakkeesta. Tällä voidaan näyttää tietoja

sivulla olevista elementeistä, kuten tekstikentistä ja painikkeista (Kuva 17). On kuitenkin huomioitava, että elementtien tarkasteluun on monia muitakin tapoja.

Kuva 17 Elementtien tarkastelu Chromessa

Kun hiiren kuvakkeesta painetaan, siirretään hiiri tekstikentän päälle ja klikataan sitä. Oikealla, valkoisella taustalla näkyvässä sivun elementtien listauksessa korostuu tekstikentän tiedot harmaalla. Harmaalla korostetusta kohdasta voidaan todeta, että tekstikentän tunnus on `username` (Kuva 18). Toistetaan sama myös salasanakentälle ja otetaan siitäkin tunnus talteen, joka on `password`.

Kuva 18 Käyttäjänimen paikannin



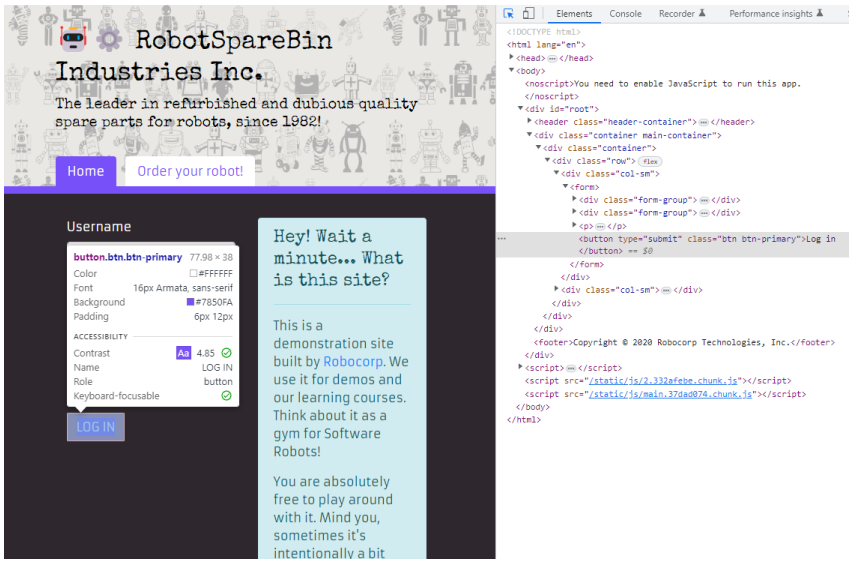
Kun tekstikenttien tunnukset on kopioitu talteen, lisätään ohjelmakoodiin kaksi `Input Text` -avainsanoilla alkavaa riviä ja kirjoitetaan niille erikseen tekstikentän ja salasanan tunnukset painamalla sarkainta `Input Text` -avainsanan jälkeen. Todetaan, että käyttäjätunnus ja salasana tarvitaan vielä (Ohjelmakoodi 2). Kun tekstikenttien tunnukset on lisätty, painetaan jälleen sarkainta, jonka jälkeen lisätään vielä käyttäjätunnus ja salasana.

Ohjelmakoodi 2 Tekstin syöttäminen

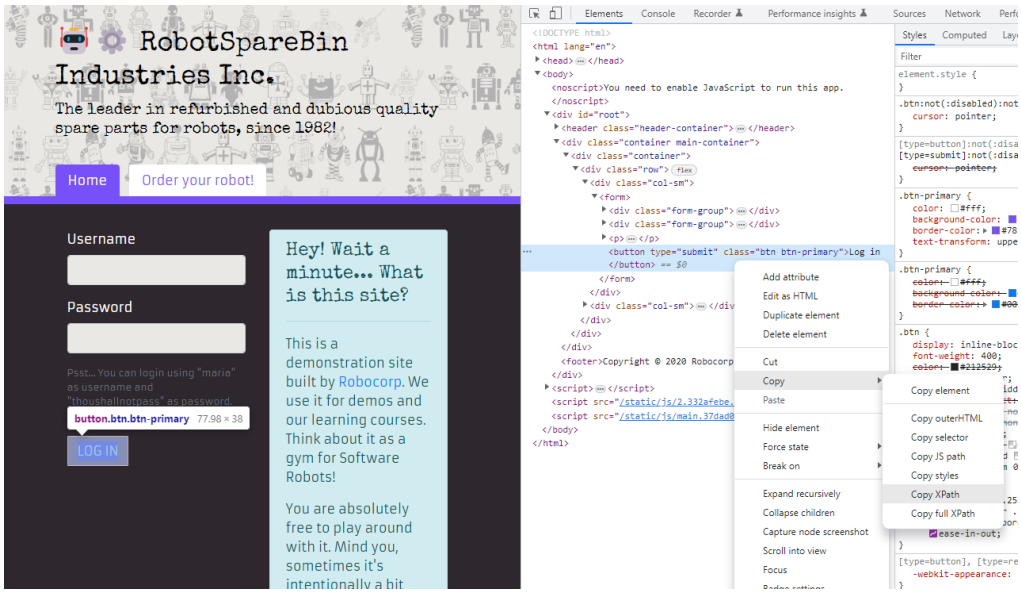
```
*** Settings ***
Library  SeleniumLibrary

*** Variables ***


*** Test Cases ***
OpenBrowser
    open browser    https://robotsparebinindustries.com/    chrome
    Maximize Browser Window
    Input Text    id:username    maria
    Input Text    id:password    thoushallnotpass
```

Tämän jälkeen testiin lisätään kirjautumispainikkeen klikkaaminen. Aktivoidaan elementtien tarkastaja uudelleen ja klikataan `Log In` -painiketta. Elementin voi tarkastaa tässä tapauksessa myös klikkaamalla kirjautumispainiketta hiiren oikealla painikkeella ja valitsemalla valikosta `Inspect` (Kuva 19). Tämän jälkeen klikataan hiiren oikealla painikkeella harmaalla korostettua kohtaa kirjautumispainikkeen tiedoista, valitaan `Copy` ja `Copy XPath` (Kuva 20).

Kuva 19 Kirjautumispainikkeen paikantimen tarkastaminen



Kuva 20 Kirjautumispainikkeen XPathin kopioiminen



XPathin avulla napin tiedot saadaan kopioitua helpolla tavalla. Testeissä käytetään lyhennettyjä XPath-polkuja. Aloitetaan uusi rivi ensin `Click Button` -avainsanalla, painetaan sarkainta, liitetään aikaisemmin kopioidun XPathin tiedot, painetaan Enteriä rivinvaihtoa varten ja kirjoitetaan lopuksi `Close Browser` -avainsana selaimen sulkemista varten. Loppuun voidaan lisätä myös pari muutaman sekunnin kestävää `Sleep` -toimintoa, jotta näemme testin suorittamisen selkeämmin (Ohjelmakoodi 3). `Sleep` -toiminnolla testin suoritus menee tauolle määritetyssä kohdassa, mutta sen aktiivinen käyttäminen ei ole järkevää ja sitä käytetään tässäkin testissä vain demonstrointimielessä. Järkevämpi vaihtoehto `Sleep` -toiminnolle on esimerkiksi `Wait until element is visible`, jolla tarkistetaan, että elementti on varmasti näkyvissä. Tällä voidaan varmistaa testin toimivuus helpommin.

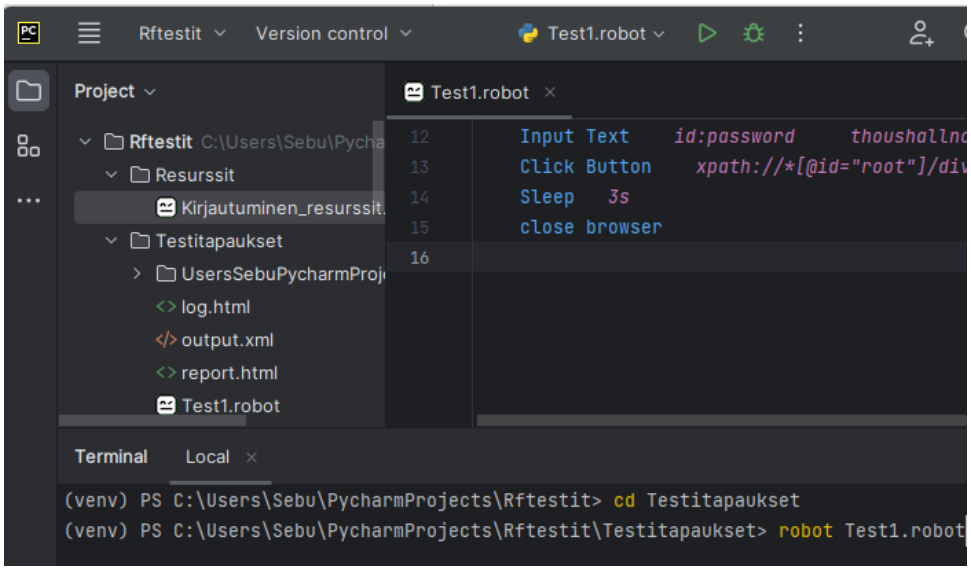
Ohjelmakoodi 3 Kirjautumispainikkeen painaminen ja selaimen sulkeminen

```
*** Settings ***
Library   SeleniumLibrary

*** Variables ***

*** Test Cases ***
OpenBrowser
    open browser    https://robotsparebinindustries.com/  chrome
    Maximize Browser Window
    Input Text    id:username   maria
    Sleep   3s
    Input Text    id:password     thoushallnotpass
    Click Button
xpath://*[@id="root"]/div/div/div/div[1]/form/button
```

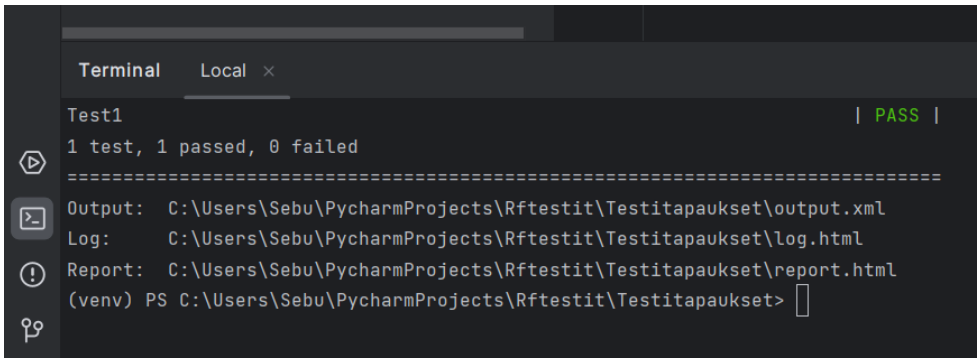
```
Sleep    3s
close browser
```

Tämän jälkeen testitapaus voidaan ajaa. Avataan PyCharmin vasemmasta alakulmasta terminaali, tarkistetaan että kansiorakenteessa ollaan `Testitapaukset` -kansiossa. Jos ei, niin ajetaan komento `cd testitapaukset`. Tämän jälkeen komentoriville kirjoitetaan `robot Test1.robot` ja painetaan Enteriä (Kuva 21). Testin ajamisen jälkeen tulokset nähdään terminaalissa ja se on mennyt läpi (Kuva 22). Seuraavassa alaluvussa tarkastellaan testistä syntynyttä raporttia.

Kuva 21 Testin käynnistäminen PyCharmissa



Kuva 22 Testin läpi meneminen PyCharmissa

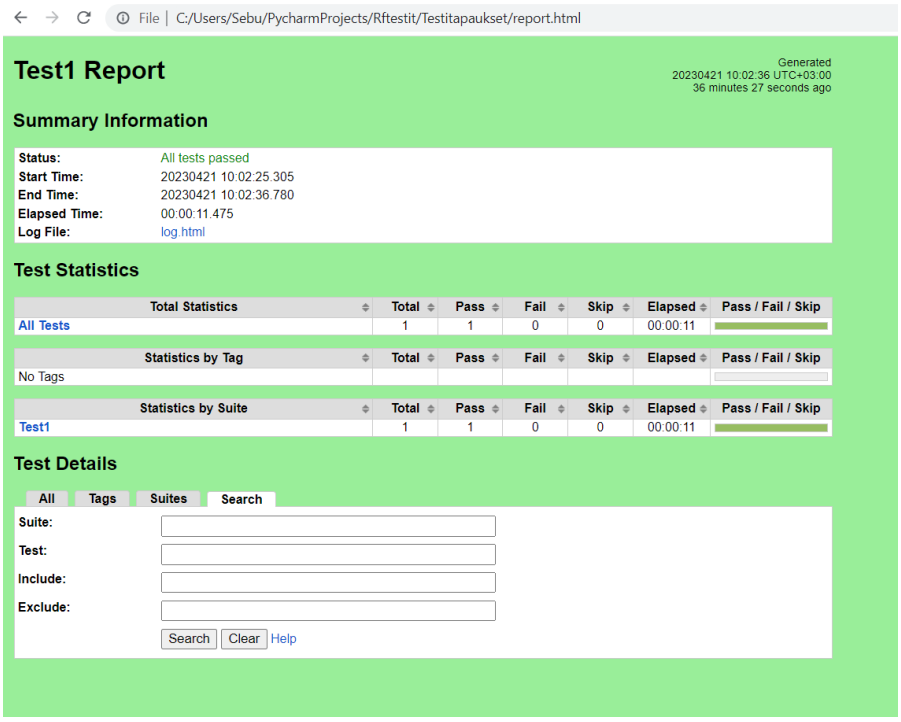


9.3 Raportin tarkastelu

Ensimmäisen testitapauksen suorittamisen jälkeen nähdään, että testistä on syntynyt kolme tiedostoa, output.xml, log.html ja report.html. Jos testin aikana on otettu kuvakaappauksia, eikä niille ole määritetty erillistä kansiota testissä, ne ilmaantuvat samaan kansioon kuin raporttitiedostot. Kopioidaan `report.html` -tiedoston polku tai navigoidaan resurssienhallinnan kautta kyseiseen sijaintiin (Kuva 22).

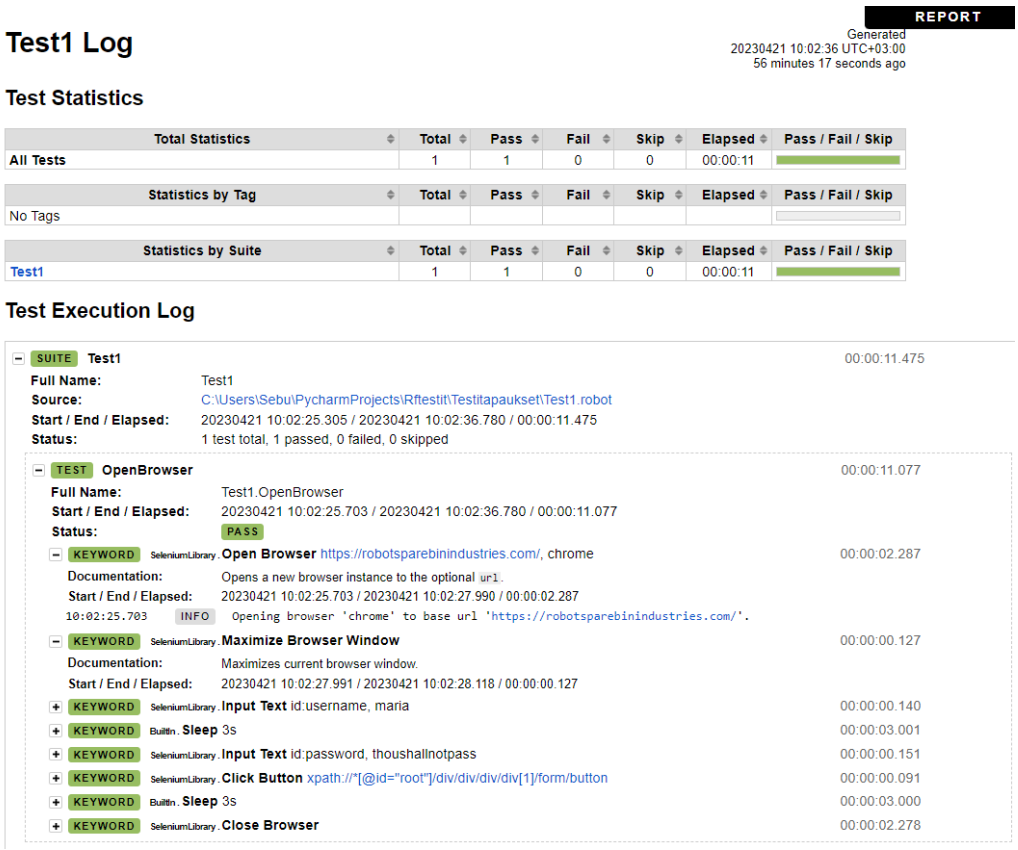
Raportin polku voidaan kopioida verkkoselaimen osoiteriville. Raportin yläosassa nähdään tiivistelmä testin suorituksesta. Raportista nähdään ovatko testit menneet läpi, aloitus- ja lopetusaika, kesto sekä linkki erilliseen lokitiedostoon. Keskiosassa suoritusten tiedot ovat jaettu erikseen koko testitiedoston, merkintöjen eli tagien ja testisarjojen välillä. Tässä tapauksessa on vain yksi testitapaus ja erillisiä merkintöjä ei ole määritetty. Raportin alaosassa tuloksia voidaan suodattaa. Klikataan lopuksi raportin alaosasta All-välilehti auki (Kuva 23). Testin nimestä voidaan klikata, jotta koko testin ajon lokitiedot saadaan näkyviin (Kuva 24). Lokista voidaan tarkastella jokaisen avainsanan suorituksen tietoja kuten avainsanan kuvaus ja suoritusaika (Kuva 25).

Kuva 23 Testiraportin tarkastelu



Kuva 24 Testiajon tietojen avaaminen

Kuva 25 Testin lokitietojen tarkastelu



9.4 Ensimmäisen testitapauksen parantaminen

Aikaisemmassa alaluvussa tehtyä testitapausta voidaan parannella hyödyntämällä muuttujia ja omia avainsanoja. Kirjautumiseen käytetyistä kohdista voidaan rakentaa oma avainsana `kirjaudusovellukseen` ja selaimen avaamiseen käytetty osoite ja selain voidaan siirtää muuttujiksi (Ohjelmakoodi 4).

Ohjelmakoodi 4 Muuttujien ja omien avainsanojen lisääminen testiin

```
*** Settings ***
Library  SeleniumLibrary

*** Variables ***
${browser}      chrome
${url}  https://robotsparebinindustries.com/

*** Test Cases ***
OpenBrowser
    open browser    ${url}  ${browser}
    Maximize Browser Window
    KirjauduSovellukseen
    close browser

*** Keywords ***
KirjauduSovellukseen
    ${"kayttaja_teksti"}    set variable  id:username
    Element Should Be Visible   ${"kayttaja_teksti"}
    Element Should Be Enabled   ${"kayttaja_teksti"}

    Input Text    ${"kayttaja_teksti"}   maria
    Sleep   3s
    Input Text    id:password     thoushallnotpass
    Click Button
xpath://*[@id="root"]/div/div/div/div[1]/form/button
    Sleep   5s
```

Ensin määritetään muuttujan nimeksi `${"kayttaja_teksti"}` ja asetetaan muuttuja `set variable` -avainsanalla. Lopuksi lisätään tekstikentän tunniste eli `id:username`. Muuttuja toimii käytännössä nyt tekstikentän tunnisteena, joten sitä voidaan hyödyntää myös elementtien näkyvyyden (element should be visible) ja aktivoitumisen (element should be enabled) tarkasteluun sekä tekstin syöttämiseen (Ohjelmakoodi 5).

Ohjelmakoodi 5 Tekstikentän näkyvyyden varmistaminen

```
*** Settings ***
Library  SeleniumLibrary

*** Variables ***
${browser}      chrome
${url}  https://robotsparebinindustries.com/

*** Test Cases ***
OpenBrowser
    open browser    ${url}  ${browser}
    Maximize Browser Window
    KirjauduSovellukseen
    close browser

*** Keywords ***
KirjauduSovellukseen
    ${"kayttaja_teksti"}    set variable  id:username
    Element Should Be Visible   ${"kayttaja_teksti"}
```

```
    Element Should Be Enabled   ${"kayttaja_teksti"}

    Input Text    ${"kayttaja_teksti"}   maria
    Sleep   3s
    Input Text    id:password     thoushallnotpass
    Click Button
xpath://*[@id="root"]/div/div/div/div[1]/form/button
    Sleep   5s
```

Testi voidaan muuttaa epäonnistuvaksi siten, ettei elementti saisi olla näkyvissä (Ohjelmakoodi 6). Testi päättyy virheeseen, koska elementti on näkyvissä, vaikka testiin määritettiin, ettei sen pitäisi olla (Kuva 26).

Ohjelmakoodi 6 Epäonnistuneen testin tekeminen

```
*** Settings ***
Library  SeleniumLibrary

*** Variables ***
${browser}      chrome
${url}  https://robotsparebinindustries.com/

*** Test Cases ***
OpenBrowser
    open browser    ${url}  ${browser}
    Maximize Browser Window
    KirjauduSovellukseen
    close browser

*** Keywords ***
KirjauduSovellukseen
    ${"kayttaja_teksti"}    set variable  id:username
    Element Should Not Be Visible   ${"kayttaja_teksti"}
    Element Should Be Enabled   ${"kayttaja_teksti"}

    Input Text    ${"kayttaja_teksti"}   maria
    Sleep   3s
    Input Text    id:password     thoushallnotpass
    Click Button
xpath://*[@id="root"]/div/div/div/div[1]/form/button
    Sleep   5s
```

Kuva 26 Epäonnistunut testi PyCharmissa

```
OpenBrowser                                                    | FAIL |
The element 'id:username' should not be visible, but it is.
------------------------------------------------------------------------
Test1                                                          | FAIL |
1 test, 0 passed, 1 failed
========================================================================
```

10 Muiden testitapausten ohjelmointi

Aikaisemmassa luvussa käytiin läpi yksinkertainen ensimmäinen testitapaus. Robot Frameworkilla ja Seleniumilla voidaan kuitenkin kirjoittaa testejä monia muita toimintoja varten. Tässä luvussa käydään erilaisia testitapauksia läpi ja lopuksi rakennetaan yksi isompi testi, jossa käytetään hyväksi resurssitiedostoa.

10.1 Lomakkeen täyttäminen

Tässä testitapauksessa täytetään lomake. Lomakkeeseen syötetään tekstiä, valitaan valintanappeja (radio button), arvoja listoista sekä valintaruuduista (checkbox). Koko testi on nähtävissä alla (Ohjelmakoodi 7).

Ohjelmakoodi 7 Lomakkeen täyttäminen

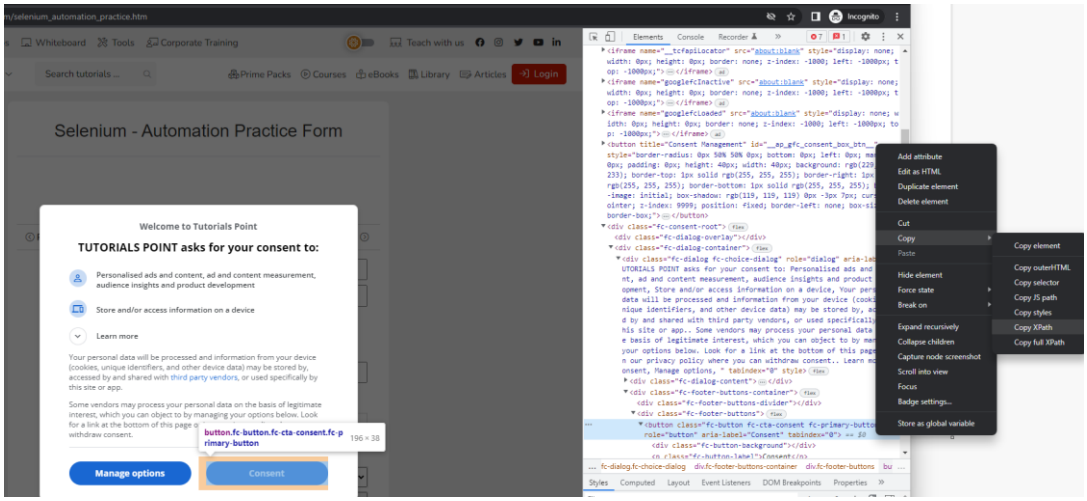
```
*** Settings ***
Library  SeleniumLibrary

*** Variables ***
${browser}      chrome
${url}
https://www.tutorialspoint.com/selenium/selenium_automation_practice.h
tm

*** Test Cases ***
Avaa selain ja tayta lomake
    open browser    ${url}  ${browser}
    Set Selenium Speed    0.2s
    Maximize Browser Window
    Tayta lomake
    Close Browser


*** Keywords ***
Tayta lomake
    Element Should Be Visible
xpath:/html/body/div[5]/div[2]/div[1]/div[2]/div[2]/button[1]
    Click Button
xpath:/html/body/div[5]/div[2]/div[1]/div[2]/div[2]/button[1]
    Click Link  xpath://*[@id="banner-accept"]

    Input Text    firstname     Seppo
    Input Text    lastname      Sepponen

    Select Radio Button     sex     Male
    Select Radio Button     exp     5
    Input Text
xpath://*[@id="mainContent"]/div[4]/div/form/table/tbody/tr[5]/td[2]/i
nput      25.4.2023
    Select Checkbox    Manual Tester
    Select Checkbox    Selenium Webdriver
```

```
    Select From List By Label        continents  Europe
    #Select From List By Index     continents  1
    Select From List By Label    selenium_commands  Browser Commands
    Unselect From List By Label     selenium_commands  Browser
Commands
    Select From List By Label    selenium_commands  Navigation
Commands
```

Yllä oleva testitapaus on kirjoitettu valmiiksi, mutta jokainen kohta on käytävä läpi. Aikaisemman testitapauksen muuttujia voidaan hyödyntää, mutta sivun osoite muutetaan. Testitapaus aloitetaan selaimen avaamisella ja suorittamiselle asetetaan nopeudeksi 0.2 sekuntia (set selenium speed). Tämän jälkeen selainikkuna suurennetaan, lomake täytetään eri tiedoilla ja lopuksi suljetaan selain. Seuraavassa vaiheessa kerrotaan lomakkeen täyttämiseen tarvittavat vaiheet.

Kun testissä käytetty sivu avataan, sen evästeet joudutaan hyväksymään. Elementti voidaan paikantaa samalla tavalla kuin aikaisemmassa testitapauksessa, eli painamalla F12-nappia ja aktivoimalla elementtien tarkastamiseen käytettävä toiminto (onnistuu myös CTRL+SHIFT+C näppäinyhdistelmällä). Kopioidaan napin XPath talteen (Kuva 27).

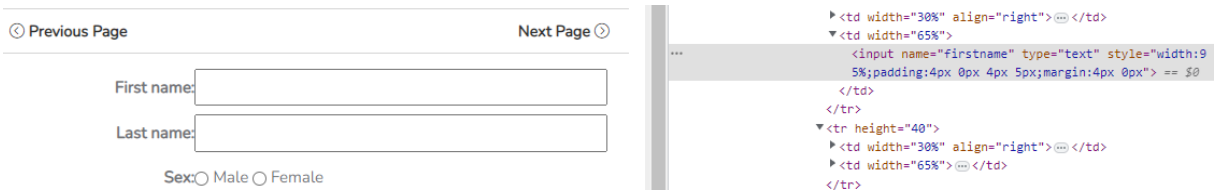
Kuva 27 Evästeiden hyväksyminen



Tarkistetaan kuitenkin, että evästeiden hyväksymiseen käytetty nappi on näkyvissä ja aloitetaan testi sillä (element should be visible), jonka jälkeen suoritetaan itse napin painaminen (click button). Tämän testin kirjoittamisen aikana havaittiin, että sivulle ilmaantui toinen valinta evästeiden hyväksymiselle linkkinä. Ensimmäisen evästeen hyväksymisen jälkeen lisätään rivi linkin klikkaamista varten, jossa hyödynnetään XPathia

`xpath://*[@id="banner-accept"]`. Tarkistetaan tämän jälkeen nimien täyttämiseen käytettävien kenttien nimet samalla tavalla kuin aikaisemmassa testissä. Todetaan, että kenttien nimet ovat `firstname` ja `lastname` (Kuva 28). Kirjoitetaan siis omat rivit etu- ja sukunimen täyttämiselle.

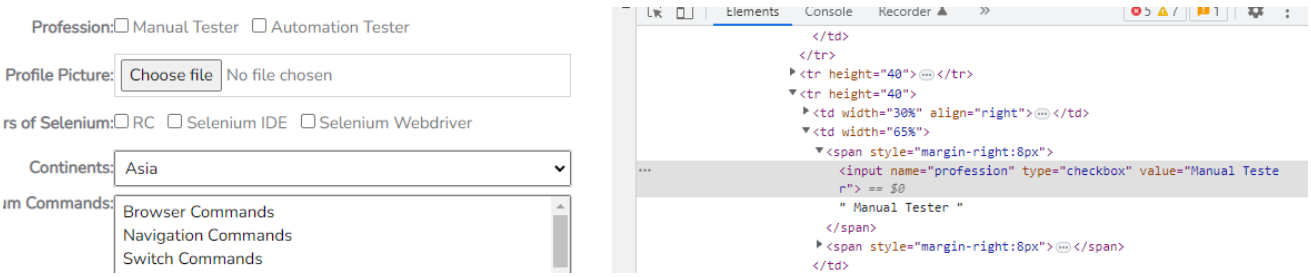
Kuva 28 Nimikenttien elementin tarkastaminen



Seuraavassa vaiheessa valitaan sukupuoli ja kokemusvuodet valintanapeista (select radio button), kirjoitetaan päivämäärä tekstikenttään ja valitaan ammatti sekä Seleniumin mieltymykset valintaruuduilla. Tarkistetaan ensin valintanappien nimet klikkaamalla niitä hiiren oikealla painikkeella ja valitsemalla `Inspect`. Sukupuolen valintaruudun nimi on `sex` ja kokemusvuosien `exp`. Nämä lisätään ohjelmakoodiin.

Täytetään valintanappien valitsemisen jälkeen päivämäärä tekstikenttään ja valitaan valintaruudut (select checkbox). Tarkistetaan myös valintaruutujen arvot (value). Valitaan ensimmäiseksi valintaruutu, jonka arvo on `Manual Tester`. Toisessa valintaruudussa valitsemme kolmannen vaihtoehdon, jonka arvo on `Selenium Webdriver` (Kuva 29). Täytetään valintaruutujen valitseminen myös ohjelmakoodiin.

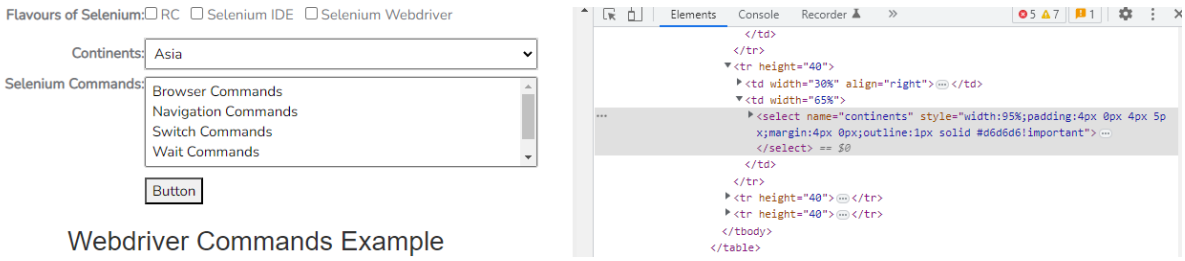
Kuva 29 Valintaruutujen elementtien tarkastaminen



Viimeisessä vaiheessa pudotusvalikosta valitaan maanosaksi Eurooppa ja valitaan listasta vaihtoehto. Ensin tarkistetaan pudotusvalikon nimi, joka on `continents` (Kuva 30). Listasta voidaan valita vaihtoehto nimen perusteella avainsanalla `Select From List by Label` tai

numeerisella arvolla `Select From List By Index` -avainsanalla (Kuva 31). Huom. koodia voidaan kommentoida ristikkomerkeillä (kommentoitua riviä koodissa ei suoriteta).

Kuva 30 Pudotusvalikon elementin tarkastaminen

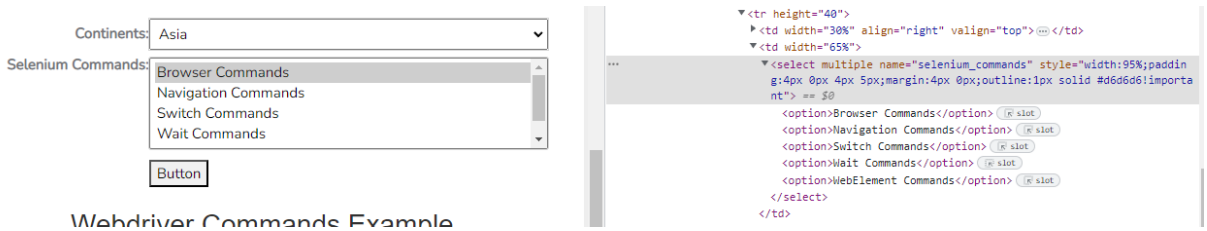


Kuva 31 Lomakkeen täyttäminen vaihe 1

```
*** Keywords ***
Tayta lomake
    Element Should Be Visible    xpath:/html/body/div[5]/div[2]/div[1]/div[2]/div[2]/button[1]
    Click Button    xpath:/html/body/div[5]/div[2]/div[1]/div[2]/div[2]/button[1]
    Click Link  xpath://*[@id="banner-accept"]

    Input Text    firstname    Seppo
    Input Text    lastname    Sepponen

    Select Radio Button    sex    Male
    Select Radio Button    exp    5
    Input Text    xpath://*[@id="mainContent"]/div[4]/div/form/table/tbody/tr[5]/td[2]/input    25.4.2023
    Select Checkbox    Manual Tester
    Select Checkbox    Selenium Webdriver

    Select From List By Label    continents  Europe
    #Select From List By Index    continents  1
```

Tarkistetaan tämän jälkeen listan nimi, joka on `selenium_commands` (Kuva 32). Listan alapuolella näkyy myös vaihtoehtojen nimet. Valitaan ensimmäinen vaihtoehto `Browser Commands`, mutta poistetaan valinta `Unselect From List By Label` -avainsanalla ja valitaan sen sijaan `Navigation Commands` (Kuva 33).

Kuva 32 Listan elementin tarkastaminen



Kuva 33 Lomakkeen täyttäminen vaihe 2

```
*** Keywords ***
Tayta lomake
    Element Should Be Visible   xpath:/html/body/div[5]/div[2]/div[1]/div[2]/div[2]/button[1]
    Click Button    xpath:/html/body/div[5]/div[2]/div[1]/div[2]/div[2]/button[1]
    Click Link  xpath://*[@id="banner-accept"]

    Input Text    firstname     Seppo
    Input Text    lastname      Sepponen

    Select Radio Button     sex     Male
    Select Radio Button     exp     5
    Input Text    xpath://*[@id="mainContent"]/div[4]/div/form/table/tbody/tr[5]/td[2]/input      25.4.2023
    Select Checkbox    Manual Tester
    Select Checkbox    Selenium Webdriver

    Select From List By Label        continents  Europe
    #Select From List By Index     continents  1
    Select From List By Label    selenium_commands  Browser Commands
    Unselect From List By Label     selenium_commands  Browser Commands
    Select From List By Label    selenium_commands  Navigation Commands
```

10.2 Varoitusten käsittely ja välilehden vaihtaminen

Tässä testitapauksessa käsitellään varoituksia, selainikkunoiden sulkemista sekä välilehtien käsittelyä (Ohjelmakoodi 8).

Ohjelmakoodi 8 Varoitusten käsittely

```
*** Settings ***
Library  SeleniumLibrary

*** Test Cases ***
Varoitusten käsittely
    Open Browser    https://chercher.tech/practice/practice-pop-ups-
selenium-webdriver  chrome
    Maximize Browser Window
    Click Element    xpath://*[@id="page-
top"]/div[1]/div/div/div/div/div[2]/input[1]
    Sleep   3s
    Handle Alert    accept
    #Alert Should Be Present     I am alert
    #Handle Alert    dismiss
    #Handle Alert   leave

    Open Browser    https://demoqa.com/browser-windows    chrome
    Maximize Browser Window
    Sleep   3s
    Click Element    xpath://*[@id="tabButton"]
    Sleep   3s
    Switch Window

    Switch Browser    1
    Sleep   3s
    ${title1}=  get title
```

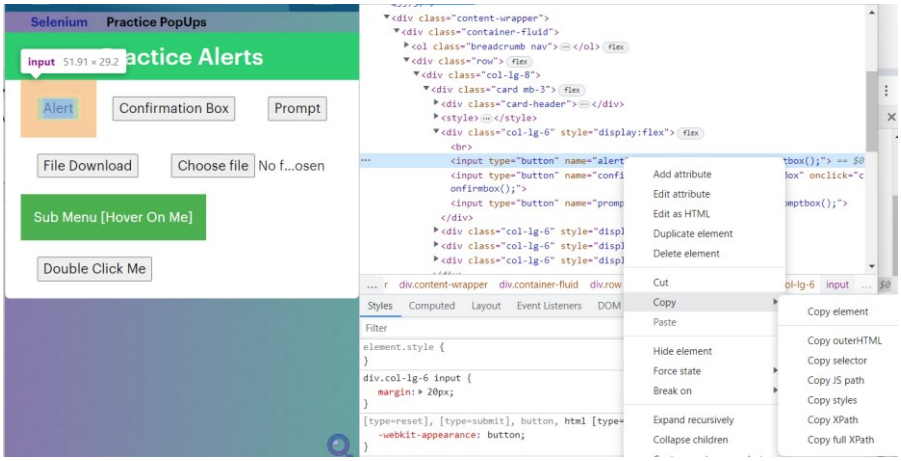
```
Log To Console    ${title1}
Close All Browsers
```

Yllä oleva testitapaus on kirjoitettu valmiiksi, mutta jokainen kohta käydään jälleen läpi. Testi aloitetaan selaimen avaamisella tiettyyn osoitteeseen, ikkunan suurentamisella ja klikkaamalla Alert-painiketta.

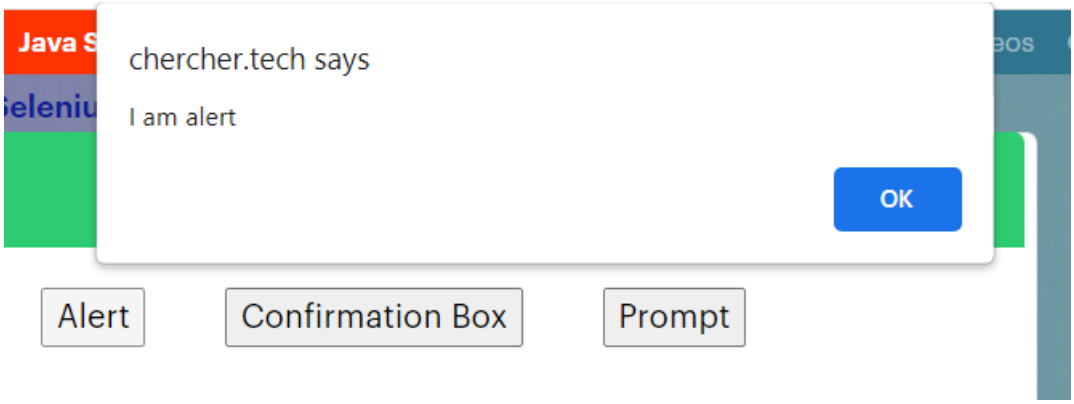
Ensin otetaan Alert-painikkeen XPath talteen tutulla tavalla (Kuva 34). Tämän jälkeen käsitellään varoitus `Handle Alert` -avainsanalla. Tässä tapauksessa varoitus hyväksytään `Accept` -avainsanalla. Varoitus voidaan myös hylätä (dismiss) tai jättää se auki (leave). Lisäksi varoituksen läsnäolo voidaan tarkastaa `Alert Should Be Present` -avainsanalla. Varoituksessa näkyy teksti `I am alert`, jota voidaan käyttää tarkastamiseen (Kuva 35). Seuraavassa vaiheessa avataan sivu uuteen ikkunaan ja klikataan linkkiä, joka avautuu uuteen välilehteen. Otetaan talteen klikattavan linkin XPath. Seuraavassa vaiheessa uuteen ikkunaan avataan sivu, jossa klikataan uuteen välilehteen avautuvaa linkkiä. Kopioidaan tämänkin linkin XPath.

Välilehden avaamisen jälkeen vaihdetaan välilehteä samassa ikkunassa `Switch Window` -avainsanalla. Vaihdetaan toinen selainikkuna aktiiviseksi `Switch Browser` -avainsanalla ja tarkistetaan sivun otsikko `Get Title` -avainsanalla. Otsikko otetaan siis tapauksessa ensimmäiseltä sivulta, jossa hyväksyimme varoituksen. Tallennetaan otsikko terminaaliin `Log to console` -avainsanalla ja suljetaan lopuksi kaikki selainikkunat `Close all browsers` -avainsanalla.

Kuva 34 Painikkeen elementin paikannin



Kuva 35 Varoitusteksti



10.3 Navigointi selaimessa ja kuvakaappaukset

Tässä yksinkertaisessa testitapauksessa navigoidaan selaimessa toiselle sivulle ja palataan aikaisemmalle sivulle (Ohjelmakoodi 9). Testin yhteydessä tallennetaan sivujen tiedot terminaaliin, otetaan kuvakaappaukset sivusta sekä yksittäisestä elementistä.

Ohjelmakoodi 9 Navigointi selaimessa ja kuvakaappaukset

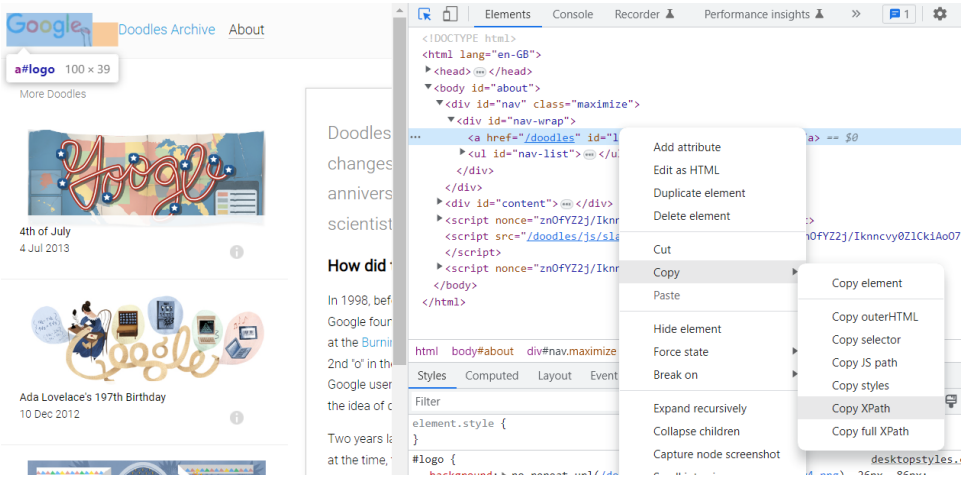
```
*** Settings ***
Library  SeleniumLibrary

*** Test Cases ***
Navigointi
    Open Browser    https://www.google.com/doodles/about    chrome
    Maximize Browser Window
    Set Selenium Speed    1
    ${loc}=     Get Location
    Log To Console    ${loc}


    Go To    https://duckduckgo.com/
    ${loc}=     Get Location
    Log To Console    ${loc}
    Capture Page Screenshot
C:/Users/Sebu/PycharmProjects/Rftestit/page.png

    Go Back
    ${loc}=     Get Location
    Log To Console    ${loc}
    Capture Element Screenshot  xpath://*[@id="logo"]
C:/Users/Sebu/PycharmProjects/Rftestit/logo.png
    Close Browser
```

Avataan ensin Googlen sivu https://www.google.com/doodles/about, suurennetaan selainikkuna, asetetaan viiveeksi yksi sekunti ja tallennetaan sivun osoite konsoliin `Get location` ja `Log to console` -avainsanoilla. Tämän jälkeen siirrytään uudelle sivulle ja

tallennetaan jälleen sivun osoite konsoliin. Otetaan myös koko sivusta ruutukaappaus haluttuun sijaintiin `Capture page screenshot` -avainsanalla.

Tämän jälkeen siirrytään aikaisemmalle sivulle `Go back` -avainsanalla, tallennetaan sivun osoite sekä otetaan ruutukaappaus sivulla olevasta Googlen kuvakkeesta `Capture element screenshot` -avainsanalla. Hyödynnämme kuvan tallentamisessa kuvakkeen XPathia (Kuva 36). Lopuksi suljemme selainikkunan.

Kuva 36 Logon elementin paikannin



10.4 Hiiritoiminnot ja näytön vieritys

Tässä testissä käsitellään hiiren erilaisia toimintoja selaimessa ja näytön vierittämistä (Ohjelmakoodi 10). Testissä avataan ensin selain ja klikataan kontekstivalikko auki sivulla. Tämän jälkeen siirrytään toiselle sivulle, tuplaklikataan linkkiä ja tarkistetaan tekstin perusteella, että tuplaklikkaus onnistui. Tämän jälkeen siirrytään uudelle sivulle, suoritetaan raahaus ja pudotus sekä tarkistetaan elementin tekstin perusteella, että pudotus onnistui. Lopuksi vieritetään näyttöä niin, että tietty elementti tulee näkyviin.

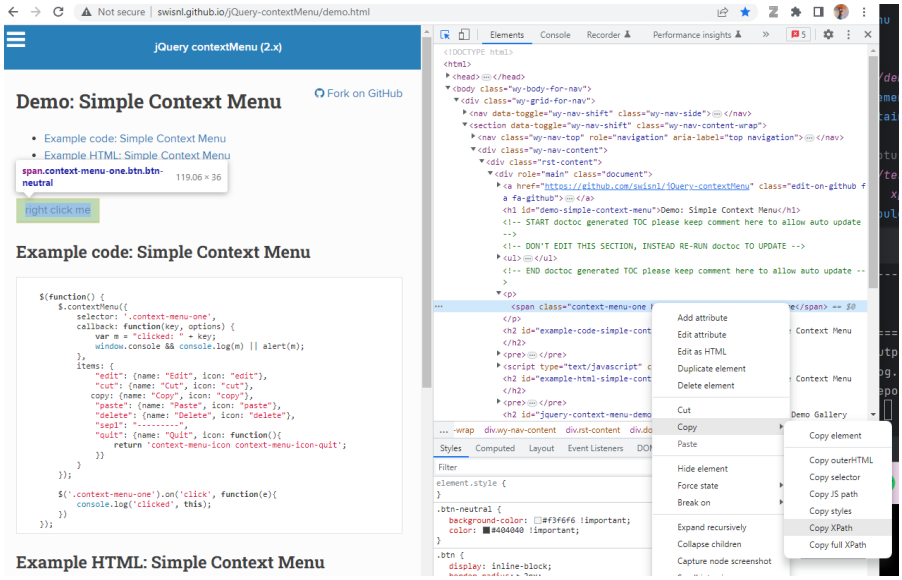
Ohjelmakoodi 10 Hiiritoiminnot ja näytön vieritys

```
*** Settings ***
Library   SeleniumLibrary

*** Test Cases ***
Hiiritoiminnot
    Set Selenium Speed     0.5s

    #Kontekstimenun avaaminen
```

```
    Open Browser    http://swisnl.github.io/jQuery-
contextMenu/demo.html    chrome
    Maximize Browser Window
    Open Context Menu
xpath:/html/body/div/section/div/div/div/p/span

    #Tuplaklikkaus
    Go To   https://demoqa.com/buttons
    Double Click Element    xpath://*[@id="doubleClickBtn"]
    Page Should Contain     You have done a double click

    #Raahaus ja pudotus
    Go To   https://testautomationpractice.blogspot.com/
    Drag And Drop    xpath://*[@id="draggable"]
xpath://*[@id="droppable"]
    Element Text Should Be  xpath://*[@id="droppable"]       Dropped!


    #Näytön vieritys
    #Execute Javascript  window.scrollTo(0,1250)
    #Execute Javascript
window.scrollTo(0,document.body.scrollHeight)          #sivun loppu
    #Execute Javascript    window.scrollTo(0,-
document.body.scrollHeight)          #sivun alku
    Scroll Element Into View
xpath://*[@id="HTML1"]/div[1]/table/tbody/tr[2]/td[1]


    Close Browser
```

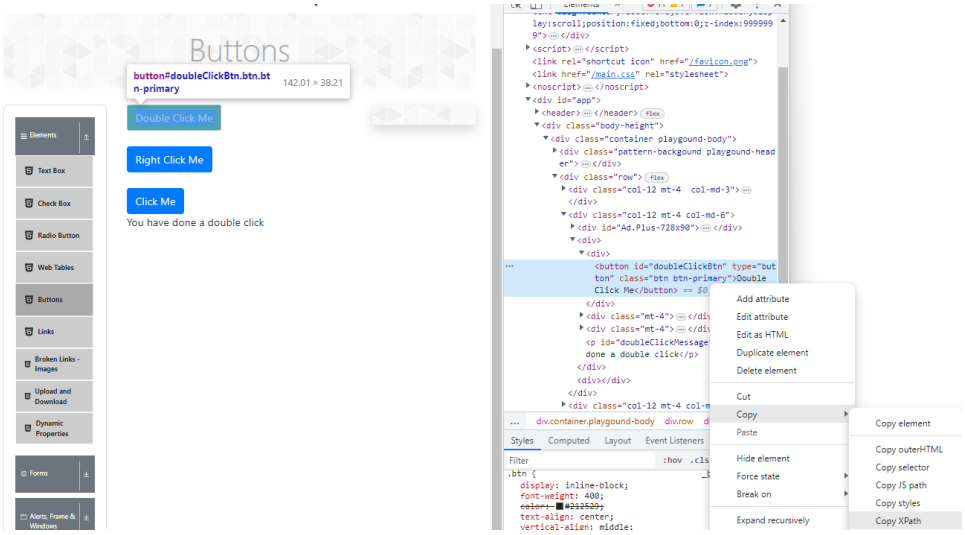
Suoritetaan ensin kontekstivalikon avaaminen (open context menu). Avataan selain osoitteeseen ja suurennetaan ikkuna. Tämän jälkeen paikannetaan ja kopioidaan kontekstimenun XPath (Kuva 37).

Kuva 37 Kontekstimenun elementin paikannin



Kontekstimenun avaamisen jälkeen siirrytään uudelle sivulle ja tuplaklikataan elementtiä `Double click element` -avainsanalla. Paikannetaan tuplaklikattavan elementin XPath ja kopioidaan se talteen (Kuva 38). Kun elementtiä tuplaklikataan, sivulle ilmestyy teksti, jota voidaan käyttää varmistuksena tuplaklikkauksen onnistumisesta. Lisätään varmistus tekstin näkymisestä tuplaklikkaamisen jälkeen.

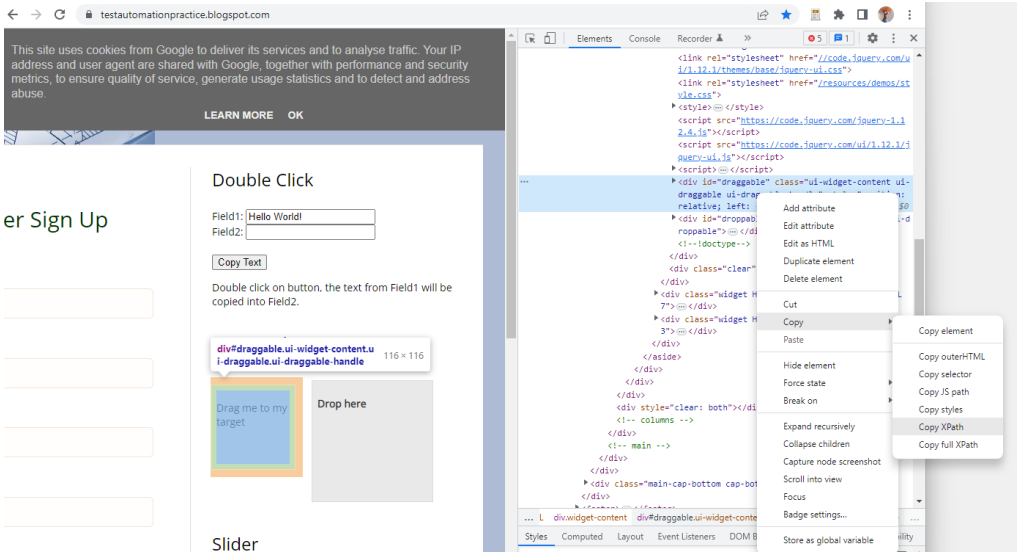
Kuva 38 Tuplaklikattavan elementin paikannin



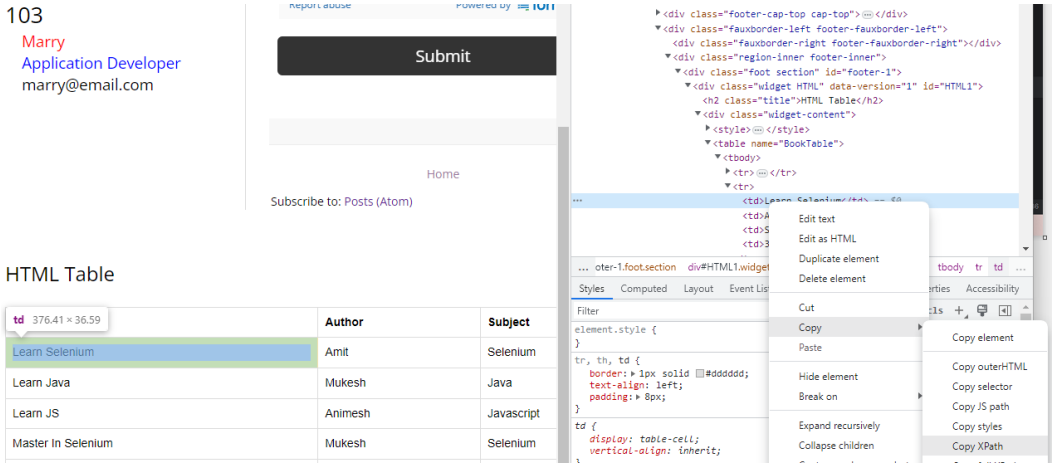
Tuplaklikkauksen ja sen tarkistamisen jälkeen siirrytään uudelle sivulle, jossa suoritetaan raahaus ja pudotus. Paikannetaan ensin raahattavan elementin XPath (Kuva 39). Tarkistetaan myös sen elementin XPath, johon raahattava laatikko pudotetaan. Laatikossa lukee `Drop here`. Pudottamisen jälkeen samaa XPathia voidaan käyttää sen tarkastamiseen, että elementissä lukee ”dropped!”.

Lopuksi käytetään samaa sivua näytön vieritykseen. Sivun alaosan taulussa näkyvä `Learn Selenium` -teksti halutaan näkyviin `Scroll element into view` -avainsanalla. Tähän voidaan hyödyntää taulun solun XPathia (Kuva 40). Esimerkkikoodissa on kolme muuta tapaa tehdä näytön vieritystä `Execute Javascript` -avainsanalla. Ensimmäisessä esimerkissä näyttöä vieritetään tietyn pikselimäärän verran alaspäin, toisessa siirrytään sivun yläosaan ja kolmannessa sivun loppuun.

Kuva 39 Raahattavan elementin paikannin



Kuva 40 Taulukon elementin paikannin



10.5 Taulujen käsittely ja tietojen laskeminen

Tässä yksinkertaisessa testissä käsitellään elementtien laskemista ja HTML-tauluja (Ohjelmakoodi 11). Testissä lasketaan rivien ja sarakkeiden määrä taulussa. Lisäksi haetaan ja tallennetaan tietyn solun teksti konsoliin. Lopuksi tarkistetaan, että tietyssä solussa ja rivissä lukee määritetty teksti.

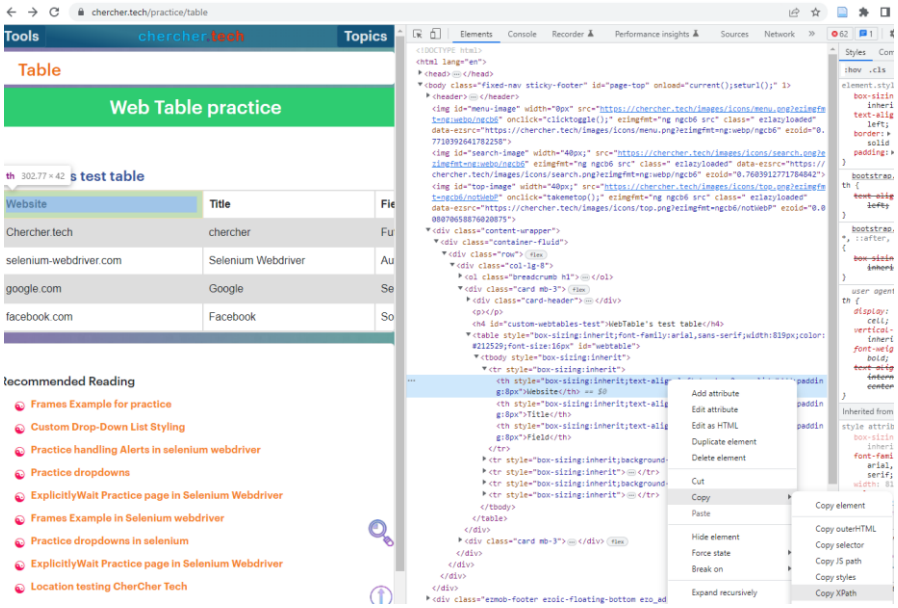
Ohjelmakoodi 11 Taulujen käsittely ja tietojen laskeminen

```
*** Settings ***
Library   SeleniumLibrary

*** Test Cases ***
```

```
Määrät ja HTML-pöydät
    Open Browser    https://chercher.tech/practice/table  chrome
    Maximize Browser Window


    #Rivien ja sarakkeiden määrä
    ${rivit}=   Get Element Count
xpath://*[@id="webtable"]/tbody/tr
    ${sarakkeet}=   Get Element Count
xpath://*[@id="webtable"]/tbody/tr/th
    Log To Console    ${rivit}
    Log To Console    ${sarakkeet}

    #Hae teksti
    ${teksti}=  Get Text
xpath://*[@id="webtable"]/tbody/tr[3]/td[2]
    Log To Console    ${teksti}

    #Solun ja rivin tekstin tarkastaminen
    Table Column Should Contain      xpath://*[@id="webtable"]/tbody
2   Title
    Table Row Should Contain          xpath://*[@id="webtable"]/tbody
5   facebook.com
```

Aloitetaan testi avaamalla selain osoitteeseen ja suurentamalla ikkuna. Tämän jälkeen kopioidaan ensimmäisen rivin ja sarakkeen XPath, josta karsitaan osa itse testiä varten (Kuva 41). Saamme XPathiksi `//*[@id="webtable"]/tbody/tr[1]/th[1]` ja karsimme sen niin, että pelkästään rivi on valittuna, eli `//*[@id="webtable"]/tbody/tr`.

Kuva 41 Taulun elementin paikannin 2



Itse testiin kirjoitetaan muuttujat rivien ja sarakkeiden määrän laskemista varten, jonka jälkeen ne tallennetaan konsoliin. Rivien määrä voidaan laskea XPathilla `//*[@id="webtable"]/tbody/tr`. Sarakkeiden laskemiseen voidaan hyödyntää XPathia `//*[@id="webtable"]/tbody/tr[1]/th[1]` , mutta karsitaan sitä niin että valitaan ensimmäisen rivin ensimmäisen sarakkeen sijasta pelkästään sarake, eli `//*[@id="webtable"]/tbody/tr/th` (Kuva 42). Haetaan rivien ja sarakkeiden laskemisen jälkeen tietyn solun teksti `Get text` -avainsanalla ja tallennetaan se konsoliin. Teksti voidaan paikantaa kopioimalla solun XPath talteen samalla tavalla kuin aikaisemmassa kohdassa. Tässä tapauksessa käytämme kolmannen rivin toista saraketta (Kuva 43).

Viimeisessä vaiheessa tarkistetaan teksti tietystä solusta ja rivistä. Aikaisemmin käytettyä XPathia voidaan karsia niin, että valitaan koko taulu, eli `//*[@id="webtable"]/tbody`. Tarkistetaan `Table column should contain` -avainsanalla, että toisessa solussa lukee ”Title”. Viidennellä rivillä lukeva Facebookin osoite tarkistetaan avainsanalla `Table row should contain`. Riville täytyy siis kirjoittaa itse toiminto, XPath, solun tai rivin numero ja teksti, jota tarkastellaan (Ohjelmakoodi 11:n viimeinen osuus).

Kuva 42 Taulun tietojen käsittely vaihe 1

```
*** Settings ***
Library  SeleniumLibrary

*** Test Cases ***
Määrät ja HTML-pöydät
    Open Browser    https://chercher.tech/practice/table  chrome
    Maximize Browser Window


    #Rivien ja sarakkeiden määrä
    ${rivit}=   Get Element Count    xpath://*[@id="webtable"]/tbody/tr
    ${sarakkeet}=   Get Element Count   xpath://*[@id="webtable"]/tbody/tr/th
    Log To Console    ${rivit}
    Log To Console    ${sarakkeet}
```

Kuva 43 Taulun tietojen käsittely vaihe 2

```
*** Settings ***
Library  SeleniumLibrary

*** Test Cases ***
Määrät ja HTML-pöydät
    Open Browser    https://chercher.tech/practice/table  chrome
    Maximize Browser Window


    #Rivien ja sarakkeiden määrä
    ${rivit}=   Get Element Count    xpath://*[@id="webtable"]/tbody/tr
    ${sarakkeet}=   Get Element Count   xpath://*[@id="webtable"]/tbody/tr/th
    Log To Console    ${rivit}
    Log To Console    ${sarakkeet}

    #Hae teksti
    ${teksti}=  Get Text    xpath://*[@id="webtable"]/tbody/tr[3]/td[2]
    Log To Console    ${teksti}
```

10.6 Testin alustaminen ja ryhmittely

Tässä testissä harjoitellaan testien alustamista ja ryhmittelyä (Ohjelmakoodi 12). Testit suoritetaan tallentamalla tietoja konsoliin, eli tässä testissä ei poikkeuksellisesti käytetä käyttöliittymän toimintoja. Testien alustaminen ja ryhmittely on hyödyllistä silloin, kun testejä on enemmän suoritettavana.

Ohjelmakoodi 12 Testin alustaminen ja ryhmittely

```
*** Settings ***

Suite Setup     Log To Console    Selaimen avaaminen
Suite Teardown    Log To Console    Selaimen sulkeminen

Test Setup    Log To Console    Kirjaudu sisään
Test Teardown    Log To Console    Kirjaudu ulos

*** Test Cases ***
Testi1 Kirjautuminen
    [Tags]    regressio
    Log To Console      Tämä on tuplaklikkauksen testi
    Log To Console    Tuplaklikkauksen testi päättyy
Testi2 Lomakkeen täyttäminen
    [Tags]    regressio
    Log To Console    Tämä on lomaketesti
    Log To Console    Lomaketesti päättyy
Testi3 Navigointi
    [Tags]    selain
    Log To Console    Tämä on navigointitesti
    Log To Console    Navigointitesti päättyy
```

Testitapaukset voidaan alustaa testisarjoilla (test suite). Sarjan alustuksella (suite setup) määritetään mikä toiminto suoritetaan alussa ennen yhdenkään testitapauksen suorittamista. Sarjan purkamisella (suite teardown) voidaan määrittää mikä toiminto suoritetaan kaikkien testitapausten suorittamisen jälkeen.

Vastaavasti yksittäisille testitapauksille voidaan määrittää omat alustukset (test setup). Testin alustuksella voidaan määrittää jokin toiminto, joka suoritetaan jokaisen testitapauksen alussa esimerkiksi sisäänkirjautuminen. Jokaisen testitapauksen jälkeen voidaan myös määrittää testin purkaminen (test teardown) esimerkiksi uloskirjautumisella.

Testeille voidaan lisätä merkintöjä (tag), joilla voidaan suorittaa tietyn merkinnän testitapauksia tai poissulkea tiettyjä testitapauksia. Alla olevissa kolmessa testitapauksessa ensimmäiset kaksi testitapausta ovat merkitty sanalla ”regressio”. Kolmas testi sen sijaan on merkitty sanalla ”selain”.

Alla olevassa testissä tehdään ensin testisarjan alustus selaimen avaamisella (tieto tallennetaan tekstinä konsoliin). Koko testisarja puretaan selaimen sulkemisella. Kaikki testitapaukset alustetaan sisäänkirjautumisella ja puretaan uloskirjautumisella. Itse testitapaukset ovat yksinkertaisia ja niissä ilmaistaan testin alkaminen ja päättyminen tekstinä.

Testisarja voidaan ajaa ensin normaalilla ajokomennolla, jonka jälkeen kaikki testitapaukset ajetaan merkinnöistä huolimatta. Kaikki tapahtuu ajossa kuten aiemmin määritettiin (Kuva 44). Ensin ilmoitetaan selaimen avaamisesta testisarjan alustusta varten, jonka jälkeen siirrytään ensimmäiseen testitapaukseen. Testitapaus alustetaan sisäänkirjautumisella, sitten itse testitapauksen tiedot tallennetaan konsoliin ja puretaan testitapaus uloskirjautumisella. Kaksi muuta testitapausta suoritetaan samalla tavalla, kunnes lopussa nähdään koko testisarjan päättyminen selaimen sulkemisella.

Kuva 44 Alustetun testin ajaminen

```
(venv) PS C:\Users\Sebu\PycharmProjects\Rftestit\testitapaukset> robot Test10tagit.robot
==============================================================================
Test10tagit
==============================================================================
Selaimen avaaminen
Testi1 Kirjautuminen                                                  Kirjaudu sisään
.Tämä on tuplaklikkauksen testi
.Tuplaklikkauksen testi päättyy
.Kirjaudu ulos
Testi1 Kirjautuminen                                                  | PASS |
------------------------------------------------------------------------------
Testi2 Lomakkeen täyttäminen                                          Kirjaudu sisään
.Tämä on lomaketesti
.Lomaketesti päättyy
.Kirjaudu ulos
Testi2 Lomakkeen täyttäminen                                          | PASS |
------------------------------------------------------------------------------
Testi3 Navigointi                                                     Kirjaudu sisään
.Tämä on navigointitesti
.Navigointitesti päättyy
.Kirjaudu ulos
Testi3 Navigointi                                                     | PASS |
------------------------------------------------------------------------------
Selaimen sulkeminen
Test10tagit                                                           | PASS |
3 tests, 3 passed, 0 failed
==============================================================================
Output:  C:\Users\Sebu\PycharmProjects\Rftestit\testitapaukset\output.xml
Log:     C:\Users\Sebu\PycharmProjects\Rftestit\testitapaukset\log.html
Report:  C:\Users\Sebu\PycharmProjects\Rftestit\testitapaukset\report.html
```

Tämän jälkeen testien ajamista voidaan kokeilla eri merkinnöillä. Aluksi voidaan suorittaa pelkästään ”selain” merkinnällä määritetyt testit komennolla `robot --include=selain Test10tagit.robot`. Vastaavasti ”selain” merkinnällä määritetyt testitapaukset voidaan poissulkea komennolla `robot --exclude=selain Test10tagit.robot`, jolloin muilla merkinnöillä määritetyt testitapaukset suoritetaan.

Testejä voidaan poissulkea myös ajokomennolla `robot -e regressio Test10tagit.robot`. Tässä tapauksessa poissuljetaan suorituksesta ”regressio” merkinnällä määritetyt testitapaukset. Samassa ajokomennossa voidaan myös määrittää, millä merkinnällä määritetyt testit poissuljetaan ja mitkä otetaan mukaan ajoon esimerkillä `robot -e selain -i regressio Test10tagit.robot`.

10.7 Resurssitiedostot ja kirjautuminen eri yhdistelmillä

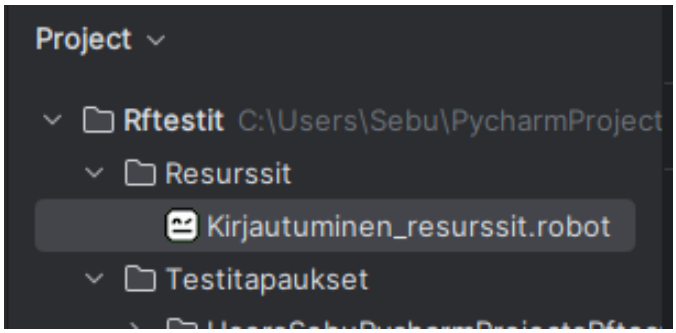
Viimeisessä testitapauksessa hyödynnetään resurssitiedostoa testien ajamiseen. Resurssitiedostoon määritetään kaikki toiminnot avainsanoiksi, joita hyödynnetään

erillisessä testitiedostossa. Testissä käytetään samaa sivua, kuin kohdassa 10.2. Testissä kokeillaan kirjautumisen epäonnistumista kolmella eri yhdistelmällä. Tässä vaiheessa käytännön osuutta ei enää käydä elementtien paikantamista erikseen läpi, vaan hyödynnetään aikaisemmista testeistä saatuja toimintoja ja dokumentaatiota.

10.7.1 Resurssitiedoston määrittäminen

Resurssitiedostolle luodaan oma kansio projektin juureen. Kansioon luodaan tiedosto, jonka nimi on `Kirjautuminen_resurssit.robot` (Kuva 45).

Kuva 45 Uuden kansion ja resurssitiedoston luominen



Alla olevaan resurssitiedostoon määritetään ensin kirjasto, eli SeleniumLibrary, jonka jälkeen määritetään muuttujat selaimelle ja käytettävälle osoittelle (Ohjelmakoodi 13). Avainsanoihin määritetään ensin oma avainsana `Avaa selain` selaimen avaamiselle ja ikkunan suurentamiselle. Tämän jälkeen määritetään avainsana `Avaa kirjautumissivu`, jossa avataan kirjautumissivu uudelleen. Tätä hyödynnetään siinä vaiheessa, kun kirjautuminen epäonnistuu ja halutaan varmentaa, että uusi yritys lähtee tyhjältä sivulta ja ilmoituksia ei ole näkyvissä. Määritetään oma avainsana `Sulje selain` selainikkunoiden sulkemista varten.

Tämän jälkeen lisätään avainsanat `Kirjoita kayttajanimi` ja `Kirjoita salasana` käyttäjänimen ja salasanan täyttämistä varten. Hyödynnetään tietojen täyttämistä varten argumentteja, jolloin itse testitapauksissa voidaan käyttää useita eri vaihtoehtoja kirjautumistietojen täyttämiseen. Kirjautumispainikkeen painamista varten käytettävässä avainsanassa `Klikkaa kirjautumispainiketta` voidaan hyödyntää samaa XPathia, kuin

kohdassa 10.2 tehdyssä testissä. Lisätään myös avainsana `Kirjaudu ulos` uloskirjautumista varten. Lopuksi tarkistetaan virheellisestä kirjautumisesta aiheutuva ilmoitus avainsanalla `Virheteksti pitaisi nakya`. Ilmoituksen XPath on otettu talteen kokeilemalla virheellistä kirjautumista manuaalisesti. Loppuun voidaan määrittää vapaaehtoinen avainsana `Sivu nayttaa tiedon kirjautumisen jalkeen`. Tällä avainsanalla voidaan tarkistaa, että kirjautumisen jälkeen näkyvä tekstikenttä on näkyvissä siinä tapauksessa, jos testeihin lisätään onnistunut kirjautuminen. Onnistunut kirjautuminen kokeillaan erillisellä testitiedostolla (Ohjelmakoodi 15).

Ohjelmakoodi 13 Resurssitiedosto kirjautumiseen

```
*** Settings ***
Library  SeleniumLibrary

*** Variables ***
${browser}      chrome
${url}  https://robotsparebinindustries.com/

*** Keywords ***
Avaa selain
    Set Selenium Speed    0.2s
    open browser    ${url}  ${browser}
    Maximize Browser Window
Avaa kirjautumissivu
    Go To   ${url}
Sulje selain
    Close All Browsers
Kirjoita kayttajanimi
    [Arguments]     ${kayttaja_teksti}
    Input Text    username   ${kayttaja_teksti}
Kirjoita salasana
    [Arguments]    ${salana_teksti}
    Input Text    password     ${salana_teksti}
Klikkaa kirjautumispainiketta
    Click Button
xpath://*[@id="root"]/div/div/div/div[1]/form/button
Kirjaudu ulos
    Click Button    logout
Virheteksti pitaisi nakya
    Page Should Contain Element
xpath://*[@id="root"]/div/div/div[1]
Sivu nayttaa tiedon kirjautumisen jalkeen
    Wait Until Element Is Visible   xpath://*[@id="firstname"]
```

10.7.2 Testitiedoston määrittäminen

Resurssitiedoston määrittämisen jälkeen testitapausten kansioon voidaan luoda uusi tiedosto, johon kirjoitetaan itse testitapaukset (Ohjelmakoodi 14). Testitapauksiin hyödynnetään avainsanoja, jotka kirjoitettiin resurssitiedostoon.

Ohjelmakoodi 14 Virheellisen kirjautumisen testaus

```
*** Settings ***
Library  SeleniumLibrary
Resource    ../Resurssit/Kirjautuminen_resurssit.robot
Suite Setup    Avaa Selain
Suite Teardown    Sulje Selain
Test Template     Virheellinen Kirjautuminen

*** Test Cases ***              kayttaja        salasana
Oikea kayttaja vaara salasana   maria           123
Vaara kayttaja oikea salasana   michael         thoushallnotpass
Vaara kayttaja vaara salasana   michael         123


*** Keywords ***
Virheellinen kirjautuminen
    [Arguments]    ${kayttaja_teksti}      ${salana_teksti}
    Kirjoita Kayttajanimi   ${kayttaja_teksti}
    Kirjoita Salasana   ${salana_teksti}
    Klikkaa Kirjautumispainiketta
    Virheteksti Pitaisi Nakya
    Avaa Kirjautumissivu
```

Ohjelmakoodi 15 Onnistuneen kirjautumisen testaus

```
*** Settings ***
Library  SeleniumLibrary
Resource    ../Resurssit/Kirjautuminen_resurssit.robot
Suite Setup    Avaa Selain
Suite Teardown    Sulje Selain
Test Template     Oikea Kirjautuminen

*** Test Cases ***              kayttaja        salasana
Oikea kayttaja oikea salasana   maria           thoushallnotpass

*** Keywords ***
Virheellinen kirjautuminen
    [Arguments]    ${kayttaja_teksti}      ${salana_teksti}
    Kirjoita Kayttajanimi   ${kayttaja_teksti}
    Kirjoita Salasana   ${salana_teksti}
    Klikkaa Kirjautumispainiketta
    Virheteksti Pitaisi Nakya
    Avaa Kirjautumissivu

Oikea kirjautuminen
    [Arguments]    ${kayttaja_teksti}      ${salana_teksti}
    Kirjoita Kayttajanimi   ${kayttaja_teksti}
    Kirjoita Salasana   ${salana_teksti}
```

```
Klikkaa Kirjautumispainiketta
Sivu nayttaa tiedon kirjautumisen jalkeen
```

Määritetään ensin kirjasto eli SeleniumLibary, jonka jälkeen ilmoitetaan resurssitiedoston polku. Määritetään koko testisarjan alustukseksi selaimen avaaminen. Testisarja päätetään vastaavasti selaimen sulkemisella. Molemmissa toiminnoissa hyödynnetään resurssitiedostoon kirjoitettuja toimintoja. Testipohjana (test template) käytetään tiedoston loppuosan avainsanoihin koottuja toimintoja testitapausten suorittamista varten. Testipohjilla muunnetaan tavallinen avainsanalähtöinen testitapaus tietopohjaiseksi testiksi. Näin testitapauksiin voidaan kirjoittaa käytettävät toiminnot vain kerran samalla hyödyntäen resurssitiedostoa. Eri kirjautumistiedoille hyödynnetään argumentteja.

Kun tiedoston `Keywords` -osuuteen on kirjoitettu käytettävät toiminnot, itse testitapaukset voidaan kirjoittaa `Test cases` -osuuteen. Testitapaukset kirjoitetaan kolmelle väärälle kirjautumistiedolle. Testitapausten nimien jälkeen itse kirjautumistiedot voidaan kirjoittaa suoraan, koska aikaisemmassa vaiheessa määritettiin argumenttien käyttö kirjautumistietojen täyttämistä varten.

11 Johtopäätökset ja pohdinta

Testiautomaation käyttöönotto eroaa manuaalisesta testauksesta huomattavasti ja toiminnallisessa osuudessa suoritetut toimenpiteet toimivat hyvänä vertauskuvana. Teoriaosuuteen tukeutuen käydään läpi myös sitä, missä vaiheessa projektin elinkaarta testiautomaatiota kannattaa käyttää.

Tarvittavien asennusten tekeminen voi vaikuttaa monimutkaiselta verrattuna manuaalitestaukseen, jonka voi aloittaa milloin vain tarvittavalla perehdytyksellä. Oppimiskäyrä työssä käytettyyn testiautomaatioon on lopulta loiva. Käytännössä on vain oltava tarkkana, että asennukset suoritetaan juuri kuten on ohjeistettu, jonka jälkeen testien kirjoittamisen aloittaminen on helpompaa.

Automaattisten testien kirjoittaminen vaatii erityistä tarkkuutta ja se eroaa huomattavasti manuaalisesta testauksesta. Manuaalisessa testauksessa olet koko ajan suorassa vuorovaikutuksessa sovelluksen kanssa, kun taas testiautomaatiossa ohjelmoit testejä koodina. Testit suoritetaan tarkalleen kuten ne on määritelty, joten sovelluksen käyttäjäkokemukseen ei ole suoraa näkyvyyttä. UI-automaatiotestien suorittamisessa kuitenkin nähdään suoraan, kun kaikki testeihin kirjoitetut toiminnot tapahtuvat. Tämä osoittautui hyväksi valinnaksi työlle, koska se antoi visuaalisen tavan demonstroida testiautomaation hyötyjä myös sellaiselle henkilölle, jolle testiautomaatio ei ole tuttua.

Työn kirjoittaminen on ollut hyvin opettavainen ja vastaa niitä odotuksia, joita asetin suunnitteluvaiheessa. Työlle oli helppo asettaa raamit jo aikaisessa vaiheessa, koska tutkimuskysymyksien perusteella pystyi laatimaan alustavan sisällysluettelon, joka myös helpotti lähteiden hakemista. Suunnitteluvaiheen aikana laadituista tutkimuskysymyksistä ei tarvinnut poiketa työn tekemisen aikana ja tämä edesauttoi työn jouhevaa edistymistä.

Teoriaosuuden tekemisen aikana esiintyi huolia siitä, onko lukuihin kirjoitettu tarpeeksi tietoja. Teoriaosuuden riittävyyden pystyi kuitenkin arvioimaan helposti pohtimalla, onko tietyssä luvussa oleva tieto tarpeeksi hyvä pohjustus toiminnallisessa osuudessa esitettäviin asioihin sekä peilaamalla kirjoitettua tietoa tutkimuskysymyksiin. Kattavan teoriaosuuden

laatiminen antoi itsevarmuutta työn toiminnalliseen osuuteen ja poisti epäilyksiä etenemisen vauhdista.

Ensimmäinen tutkimuskysymys oli automaattisen testauksen ja manuaalisen testauksen erot. Lisäksi halusin selittää parhaat tilanteet joko manuaalisen tai automaattisen testauksen käyttämiseen. Ensimmäinen tutkimuskysymys oli minulle tärkeä pitkän manuaalitestauksen taustan takia. Testiautomaatiota on otettu laajaan käyttöön viime vuosina ja koin, että lukijalle olisi tärkeä esittää miten testiautomaatio eroaa konkreettisesti perinteisemmästä manuaalisesta testauksesta. Erot voidaan selittää melko yksinkertaisesti, joten taulukon käyttäminen erojen selittämiseen selkeytti tutkimuskysymykseen vastaamista.

Toinen tutkimuskysymys käsitteli testiautomaation kustannuksia. Kustannukset ja tuotot ovat hyvin kriittisiä seikkoja sovelluskehityksen- ja testauksen maailmassa, jonka takia näin tarpeen selittää, miten testiautomaatioon sijoitetun pääoman tuotto voidaan laskea, jotta sitä voidaan käyttää testiautomaation päätöksentekoprosessissa. Tutkimuskysymykseen vastattaessa tärkeintä oli koota tiedot eri lähteistä semmoiseksi kokonaisuudeksi, että lukija sai selkeän kuvan siitä, miten kustannukset lasketaan ja mitkä asiat vaikuttavat laskentaan.

Kolmas tutkimuskysymys on, miten testiautomaatio vaikuttaa päivittäisiin työrutiineihin. Testiautomaatio eroaa monella tapaa manuaalisesta testauksesta ja halusin käsitellä testiautomaatioasiantuntijan erilaisia tehtäviä ja heidän varsinaisia päivittäisiä tehtäviänsä. Tutkimuskysymys oli tärkeä tälle työlle, koska jos sitä ei olisi ollut, lukija ei välttämättä saisi selkeää kuvaa siitä, millaista työ testiautomaation parissa oikeasti olisi. Erilaisia työtehtäviä kuvattaessa saattaa jäädä konkreettisuus pieneksi, jolloin henkilö saattaakin pettyä tietyssä roolissa ollessaan, koska on kuvitellut sen olevan jotain muuta.

Viimeinen tutkimuskysymys on, millaisia asennuksia tarvitaan käyttöliittymätestauksen automatisoinnin aloittamiseen ja miten automatisoituja käyttöliittymätestejä kirjoitetaan. Tämä tutkimuskysymys muodostaa käytännön osan perustan ja on yksi tärkeimmistä asioista koko työssä. Testiautomaation ohjelmointi tukee teoriaosuudessa esitettyä mainintaa siitä, että testiautomaation käyttöönotto projektin aikaisessa vaiheessa tehostaa testausprosesseja. Toiminnallisessa osuudessa pyrittiin esittämään sellaisia toimintoja ja testitapauksia, joita voitaisiin hyödyntää myös oikean sovelluksen testaamisessa. Esimerkiksi

lomakkeen täyttäminen ja kirjautumistietojen testaaminen eri yhdistelmillä voisi toimia muutosten yhteydessä tehtävien regressiotestien kohteena.

Toiminnallisen osuuden laatimisprosessi erosi teoriaosuuden kirjoittamisesta. Toiminnallisessa osuudessa esitetty testiautomaation käyttöönotto ja ohjelmointi olivat minulle uusia ja vaativat tarkkaa perehtymistä ennen kuin niitä lähti avaamaan. Laadin itselleni aluksi suunnitelman siitä, kuinka dokumentoin käyttöönotto- ja ohjelmointiprosessia, joka osoittautui hyväksi ratkaisuksi. Mietin etukäteen myös sen, millaisia toimintoja testitapauksissa esitellään. Kirjoitin ja talletin kuvakaappauksia työhön samalla kun tein asennuksia ja ohjelmointia. Tällöin pystyin varmistumaan tietojen oikeellisuudesta ja helposta seurattavuudesta.

Toiminnallisen osuuden valmistuessa oli helppo tehdä johtopäätös siitä, että tutkimuskysymyksiin on vastattu yksityiskohtaisesti. Työn kohderyhmänä on testaajat, jotka haluavat perehtyä testiautomaatioon sekä testauksesta kiinnostuneet kokemattomammat henkilöt. Koen, että työssä on tarpeeksi tietoa ja se tarjoaa kohderyhmälleen kattavan paketin tietoa ja lähtökohdan testiautomaation aloittamiselle. Työ toimi myös minulle erinomaisena tapana perehtyä toiseen testauksen osa-alueeseen eli testiautomaatioon. Työn sisältöön perehtymällä ja toiminnallisessa osuudessa esitettyjen asioiden toistaminen ja harjoittelu voi parhaassa tapauksessa edistää työllistymistä varsinkin testiautomaatioon liittyviin tehtäviin.

12 Yhteenveto

Tutkimuskysymyksiin vastaaminen onnistui hyvin. Suunnitteluprosessin avulla saadut alustavat luvut antoivat hyvän pohjan työskentelylle. Työ alkoi testiautomaation yleisellä esittelyllä ja historialla. Näiden lisäksi käytiin läpi testiautomaation eri tyyppejä sekä testiautomaation viitekehyksiä ja sovelluskehyksiä. Nämä antoivat hyvän pohjustuksen luvuille, jotka vastasivat tutkimuskysymyksiin.

Tutkimuskysymyksiin vastaaminen itsessään oli lopulta melko yksinkertaista, koska helposti luettavassa muodossa olevia lähteitä löytyi paljon. Helppo luettavuus on tärkeää myös tässä opinnäytetyössä, koska testiautomaatiosta on tarkoitus kertoa testauksen parissa työskentelevien lisäksi myös sellaisille, jotka eivät tunne testausta niin hyvin. Koin, että onnistuin tiedon keräämisessä niin, että tutkimuskysymyksiin pystyi vastata sopivan pituisina kokonaisuuksina. Pidin tärkeänä, että luvut eivät lähtisi paisumaan liian pitkiksi, vaan että tietoa olisi juuri riittävästi.

Tämä työ oli myös minulle suuri oppimismahdollisuus. Halusin opetella testiautomaatiosta enemmän, jotta pystyisin hyödyntämään opeteltuja taitoja työelämässä. Testauksen parissa työskennelleenä olen havainnut, että testiautomaation taidot ovat arvokkaat nykypäivänä, koska testiautomaatio on saanut vakiintuneen aseman yritysten testausprosesseissa.

Opin työtä tehdessä ottamaan käyttöön automatisoituun käyttöliittymätestaukseen käytetyn sovelluskehyksen. Tämä oli tärkeä osa työtä, koska testiautomaation suorittaminen on vain pieni osa kokonaisuutta. Sovelluskehysten käyttöönotto vaati kärsivällisyyttä ja kehitysprojekteissa sovelluskehysten tuleekin olla myös tarkoin suunniteltuja, jotta ne tehostavat yrityksen tai projektin testausprosesseja halutulla tavalla. Itse testien kirjoittaminen oli myös tärkeää oppia, koska tällä sai konkreettisen näkymän siihen mitä testiautomaation asiantuntijan päivittäiseen työhön sisältyy ja mitä oikeasti tapahtuu, kun testejä suoritetaan. Koen, että tämä työ tarjoaa tulevaa varten sopivan tietokokonaisuuden testiautomaation aloittamiselle kenelle tahansa aiheesta kiinnostuneelle. Koen myös, että IT-alan yritys voisi hyödyntää työtä harkitessaan testiautomaation käyttöä.

Lähteet

Agency, D. T. (2022, November 25). *Day in the life of an Automation Tester* (Australia) [Collection]. Commonwealth of Australia. https://www.dta.gov.au/blogs/day-life-automation-tester

Automated User Interface Testing · Devbridge. (n.d.). Automated User Interface Testing · Devbridge. Retrieved 28 March 2023, from https://www.devbridge.com/articles/automated-user-interface-testing/

Automating your day-to-day activities: Why it's helpful and how you can do it. (2022, February 1). Zoho Blog. https://www.zoho.com/blog/cliq/automating-your-day-to-day-activies-with-workflow-automation.html

Automation Testing Vs Manual Testing: Everything you need to know | Disbug Blog. (n.d.). Retrieved 1 March 2023, from https://disbug.io/en/blog/automation-testing-vs-manual-testing

Automation testing—Types, Frameworks, Steps & Benefits. (n.d.). Retrieved 26 March 2023, from https://adservio.fr/post/automation-testing-types-frameworks-steps-benefits

BDD (Behavior Driven Development) Framework: A Complete Tutorial. (2023, March 14). Software Testing Help. https://www.softwaretestinghelp.com/bdd-framework/

Browser navigation. (n.d.). Selenium. Retrieved 29 March 2023, from https://www.selenium.dev/documentation/webdriver/interactions/navigation/

Cummings-John, R. (n.d.). *What is automation testing?* Retrieved 23 March 2023, from https://www.globalapptesting.com/blog/what-is-automation-testing

Doing the Math for Test Automation ROI: How to Calculate it Right. (2021, September 16). Relevant Software. https://relevant.software/blog/test-automation-roi-how-to-calculate-it-right/

Hamilton, T. (2020a, January 2). *What is Software Testing? Definition*. https://www.guru99.com/software-testing-introduction-importance.html

Hamilton, T. (2020b, January 8). *What is Automation Testing? Test Tutorial*. https://www.guru99.com/automation-testing.html

Hamilton, T. (2020c, January 9). *Difference Between Manual and Automation Testing*. https://www.guru99.com/difference-automated-vs-manual-testing.html

Hamilton, T. (2020d, January 11). *Integration Testing: What is, Types with Example*. https://www.guru99.com/integration-testing.html

Hamilton, T. (2020e, January 14). *What is Regression Testing? Test Cases (Example)*. https://www.guru99.com/regression-testing.html

jake. (n.d.). Software Testing Automation—History, Benefits, Challenges & More! *Https://Www.Zaptest.Com/*. Retrieved 25 March 2023, from https://www.zaptest.com/a-complete-guide-to-software-testing-automation

JanbaskTraining. (2022, January 31). *Roles And Responsibilities of An Automation Tester*. JanbaskTraining. https://www.janbasktraining.com/blog/roles-and-responsibilities-of-automation-tester

Kok, T. (n.d.). *Manual vs. Automated: Which UAT Testing Method Is Best?* Retrieved 30 March 2023, from https://www.testmonitor.com/blog/manual-vs.-automated-which-uat-testing-method-is-best

Lead, T. Q. (n.d.). *What Are Automation Testing Tools? 9 Types & Examples*. The QA Lead. Retrieved 26 March 2023, from https://theqalead.com/tools/what-are-automation-testing-tools/

Lead, T. Q., & Boog, J. (2019, October 21). *5 Types of Performance Testing & Top Tools*. The QA Lead. https://theqalead.com/test-management/everything-you-need-to-know-about-performance-testing/

Manual Testing vs Automation Testing: Differences. (n.d.). BrowserStack. Retrieved 1 March 2023, from https://browserstack.wpengine.com/guide/manual-vs-automated-testing-differences/

Popular Test Automation Frameworks: How to Choose. (n.d.). BrowserStack. Retrieved 28 March 2023, from https://browserstack.com/guide/best-test-automation-frameworks/

Robot Framework and Selenium Automation: Tutorial. (n.d.). BrowserStack. Retrieved 30 March 2023, from https://browserstack.wpengine.com/guide/robot-framework-and-selenium-tutorial/

Robot Framework—Quick Guide. (n.d.). Retrieved 29 March 2023, from https://www.tutorialspoint.com/robot_framework/robot_framework_quick_guide.htm

Roles and Responsibilities of Automation Test Engineers in 2023. (2019, December 16). *Intellipaat Blog*. https://intellipaat.com/blog/roles-and-responsibilities-of-automation-test-engineers/

Rungta, K. (2020a, January 2). *What is Selenium? Introduction to Selenium Automation Testing*. https://www.guru99.com/introduction-to-selenium.html

Rungta, K. (2020b, January 14). *XPath in Selenium: How to Find & Write? (Text, Contains, AND)*. https://www.guru99.com/xpath-selenium.html

Schaffer, B. (2018, December 3). Automated vs. Manual User Testing—When to choose what? *Usersnap Blog*. https://usersnap.com/blog/automated-vs-manual-user-testing/

SeleniumLibrary. (n.d.). Retrieved 29 March 2023, from https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html

Test Automation Frameworks. (n.d.). Smartbear.Com. Retrieved 28 March 2023, from https://smartbear.com/learn/automated-testing/test-automation-frameworks/

Test Automation—Expense or Investment? (2021, February 18). Kimputing. https://kimputing.com/blog/test-automation-expense-or-investment/

Testim. (2022, June 9). *Test Automation Tool: Definition and 5 Best Ones*. AI-Driven E2E Automation with Code-like Flexibility for Your Most Resilient Tests. https://www.testim.io/blog/what-is-a-test-automation-tool/

TestProject. (2022, April 14). *The Importance of Automated User Acceptance Testing*. TestProject. https://blog.testproject.io/2022/04/14/the-importance-of-automated-user-acceptance-testing/

Types Of Automated Testing Explained. (n.d.). *Prolifics*. Retrieved 26 March 2023, from https://prolifics.com/types-of-automated-testing/

WebDriver. (n.d.). Selenium. Retrieved 29 March 2023, from https://www.selenium.dev/documentation/webdriver/

What is an Interface? (n.d.). Retrieved 28 March 2023, from https://www.computerhope.com/jargon/i/interfac.htm

What is Automated Testing? (n.d.). Smartbear.Com. Retrieved 23 March 2023, from https://smartbear.com/learn/automated-testing/what-is-automated-testing/

When to use manual testing vs. Automated testing. (n.d.). Smartbear.Com. Retrieved 3 April 2023, from https://smartbear.com/test-management/manual-vs-automated-testing/

Why and When to Use Automation in Testing? (n.d.). Retrieved 4 April 2023, from https://www.netguru.com/blog/automation-in-testing-importance

Liite 1: Aineistonhallintasuunnitelma

Lähestyn työtä kehitysprojektina. Pyrin tekemään työtä n. viikon sprinteissä. Tiedon kerääminen tukeutuu vahvasti saatavilla oleviin lähteisiin. Kuvakaappauksia kerääntyy paljon toiminnallista osuutta varten, kun esittelen testiautomaatioon liittyvää ohjelmointia. Käytän työn seurantaan mm. Planneria ja kerään uutta tietoa päiväkirjaan. Säilytän toiminnallisesta osasta saamiani tietoja vähintään kahtena kappaleena pilvessä sekä paikallisesti omalla tietokoneella.

Pidän kehitysprojektin aikana päiväkirjaa, johon kerään uusia havaintoja projektista esimerkiksi siitä, jos jokin tieto täytyy muuttaa ja perustelen sen päiväkirjaan. Tämä tieto analysoidaan opinnäytetyötä varten. Säilytän päiväkirjaa tietokoneen C-asemalla, ja teen siitä säännöllisesti varmuuskopioita OneDriveen. Päiväkirjaa ja muuta aineistoa kuten kuvakaappauksia säilytän C-asemalla ainakin vuoden verran opinnäytetyön valmistumisesta. Sama pätee myös testiautomaation ohjelmoinnista syntyneeseen koodiin.

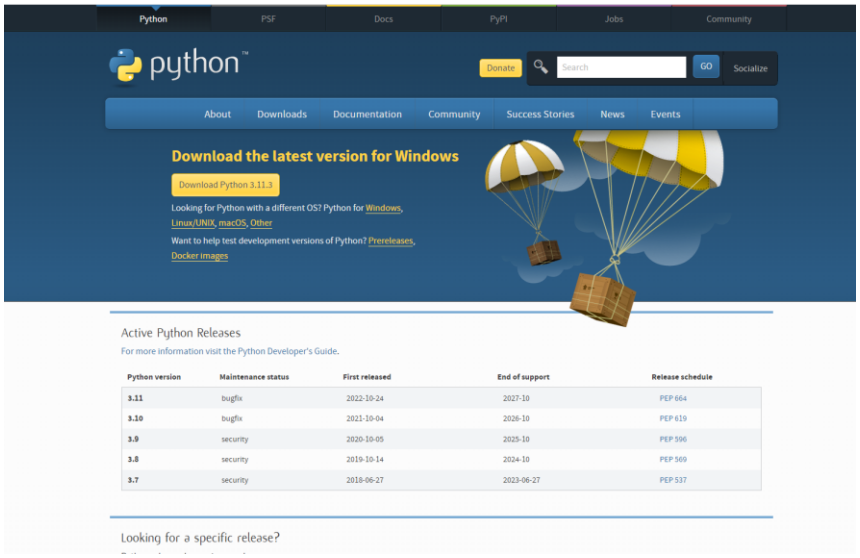
Opinnäytetyöaineiston jatkokäyttö työn valmistumisen jälkeen:

1. En halua hyödyntää tai antaa tutkimusaineistoa jatkokäyttöön

Tutkimusaineistoa ei jatkokäytetä. Opinnäytetyön tekijä säilyttää aineiston tietoturvallisesti vuoden ajan opinnäytetyön hyväksymispäivästä, jotta opinnäytetyön tulokset voidaan tarvittaessa varmistaa ja hävittää tämän jälkeen aineiston tietoturvallisesti.

Liite 2: Pythonin asennus

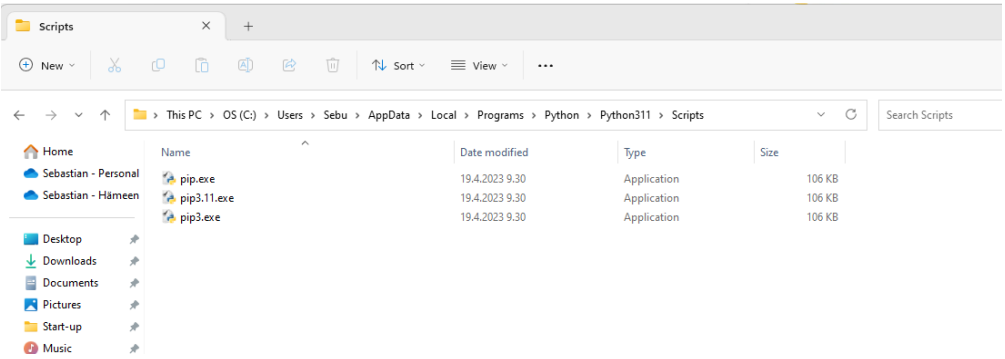
Asennetaan ensin viimeisin versio Pythonista viralliselta https://www.python.org/downloads/. Viimeisin Pythonin LTS-versio työtä laadittaessa oli 3.11.3. LTS-versiolla tarkoitetaan sovelluksen versiota, jolla on pitkäaikainen tuki (long-term support). Pythonin eri versioiden elinkaaresta on tietoa Pythonin verkkosivustolla https://devguide.python.org/versions/.

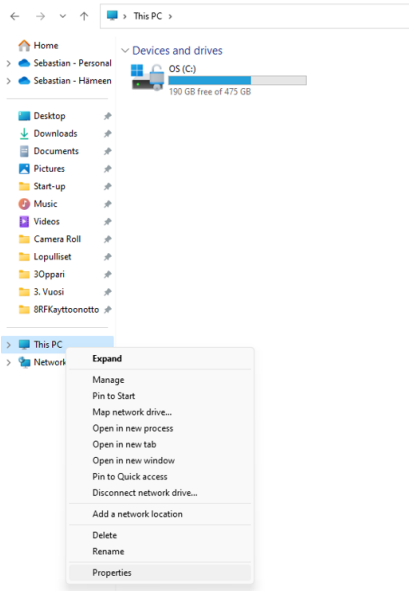


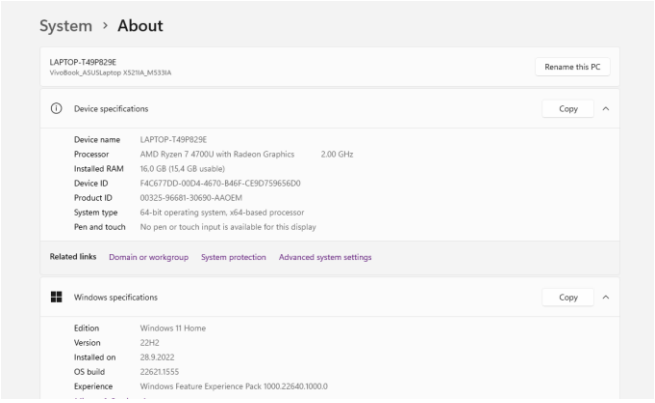
Kun asennus aloitetaan, kannattaa valita ”Add python.exe to PATH”, jolloin sitä ei tarvitse tehdä manuaalisesti. Otetaan talteen myös asennuskansio, koska sitä tarvitaan myöhemmin.

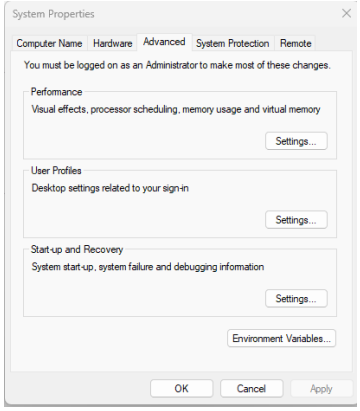
Pythonin asennuksen jälkeen lisätään Pythonin skriptikansio ympäristömuuttujaksi. Kopioidaan ensin Pythonin Scripts -kansion polku `C:\Users\User\AppData\Local\Programs\Python\Python311\Scripts`. Avataan tämän jälkeen resurssienhallinta, painetaan vaihtonäppäin pohjaan ja klikataan hiiren oikealla `This PC`

vasemmalta ja valitaan luettelosta `Properties`. Kun klikataan `Advanced system settings`, aukeaa `System Properties` -näkymä, josta valitaan oikeasta alakulmasta `Environment Variables`.

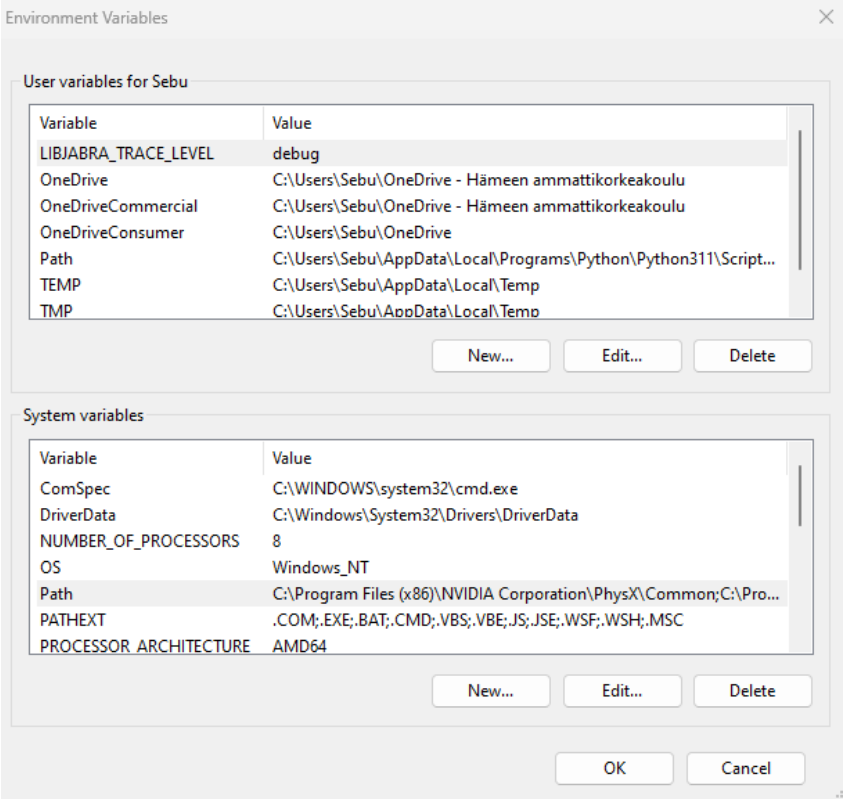


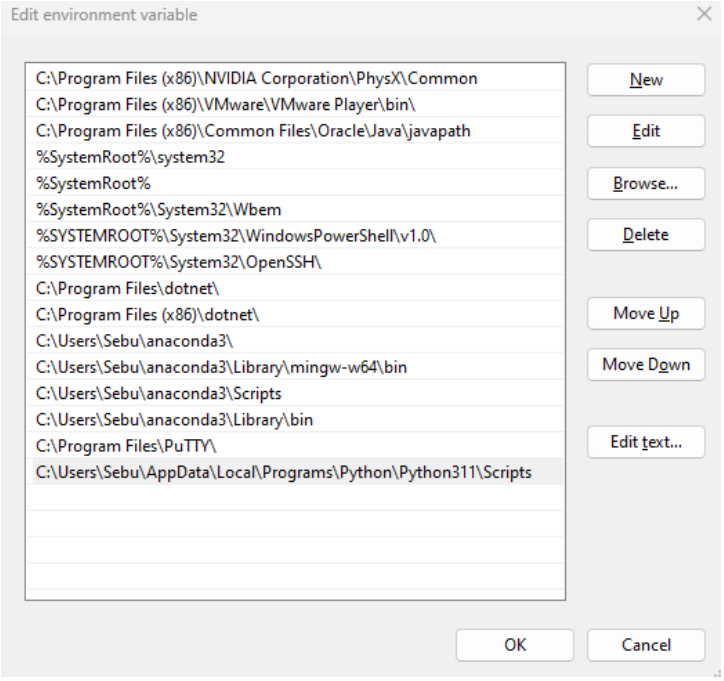


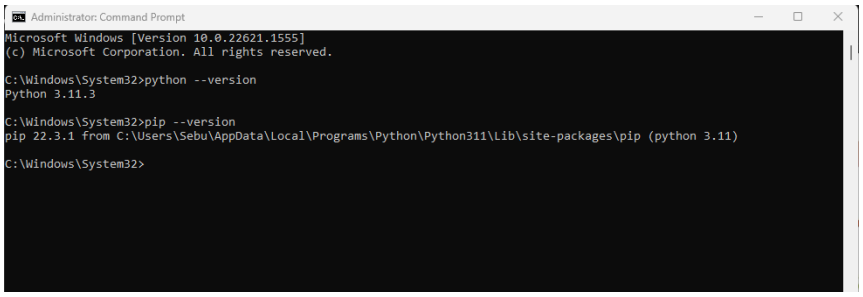




Kun ympäristömuuttujien ikkuna aukeaa, alemmasta `System Variables` -luettelosta valitaan `Path` ja klikataan `Edit`-painiketta. Tämän jälkeen `Edit Environment Variable` -ikkuna aukeaa, jossa klikataan `New` ja kopioidaan `Scripts` -kansion polku. Kaikki ikkunat voidaan lopuksi sulkea OK-painikkeella. Asennuksen voi tarkistaa lopuksi avaamalla komentorivin (command prompt) ja ajamalla komennon `python -version`. Myös pip -versio kannattaa tarkistaa `pip -version` komennolla.



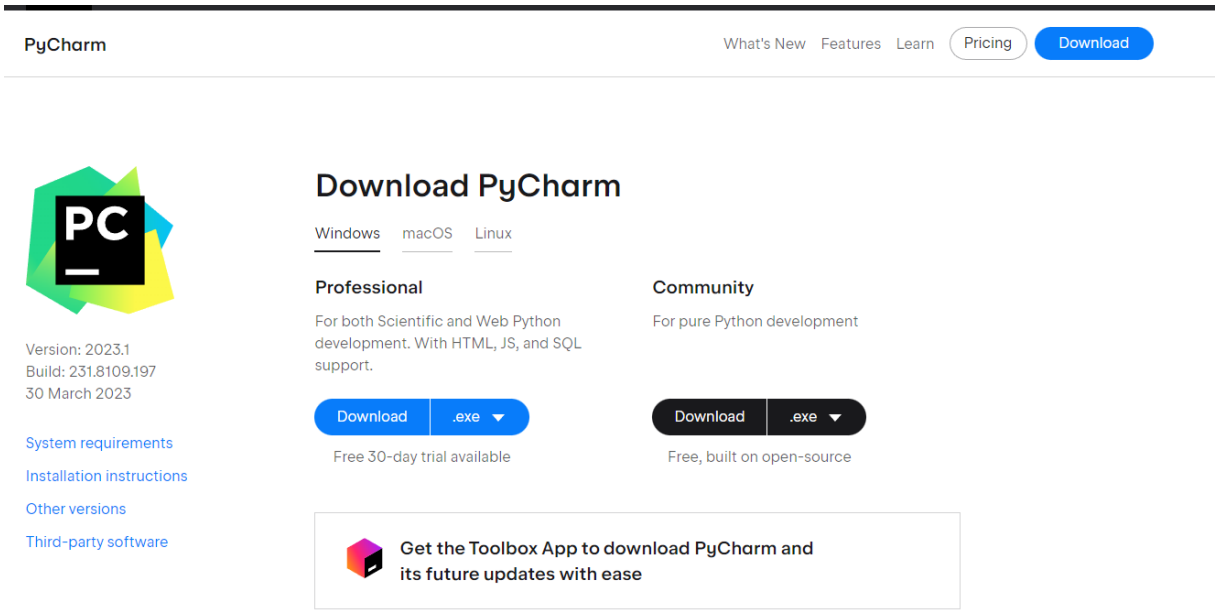

Edit environment variable
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common
C:\Program Files (x86)\VMware\VMware Player\bin\
C:\Program Files (x86)\Common Files\Oracle\Java\javapath
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\dotnet\
C:\Program Files (x86)\dotnet\
C:\Users\Sebu\anaconda3\
C:\Users\Sebu\anaconda3\Library\mingw-w64\bin
C:\Users\Sebu\anaconda3\Scripts
C:\Users\Sebu\anaconda3\Library\bin
C:\Program Files\PuTTY\
C:\Users\Sebu\AppData\Local\Programs\Python\Python311\Scripts
New
Edit
Browse...
Delete
Move Up
Move Down
Edit text...
OK
Cancel


Administrator: Command Prompt
Microsoft Windows [Version 10.0.22621.1555]
(c) Microsoft Corporation. All rights reserved.
C:\Windows\System32>python --version
Python 3.11.3
C:\Windows\System32>pip --version
pip 22.3.1 from C:\Users\Sebu\AppData\Local\Programs\Python\Python311\Lib\site-packages\pip (python 3.11)
C:\Windows\System32>

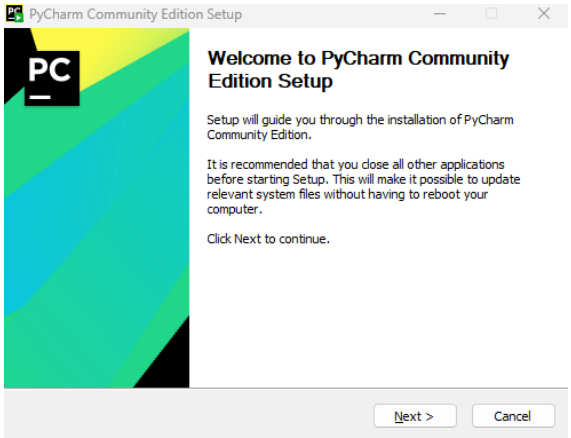
Liite 3: PyCharmin asennus

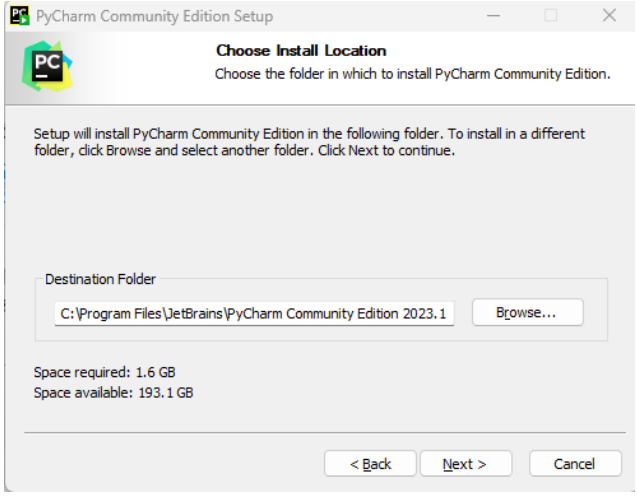
Ohjelmointiympäristöksi on valittu PyCharm. PyCharmin etuna on laaja tuki Pythonille ja Robot Frameworkin lisäosille. Se asennetaan osoitteesta https://www.jetbrains.com/pycharm/download/#section=windows.

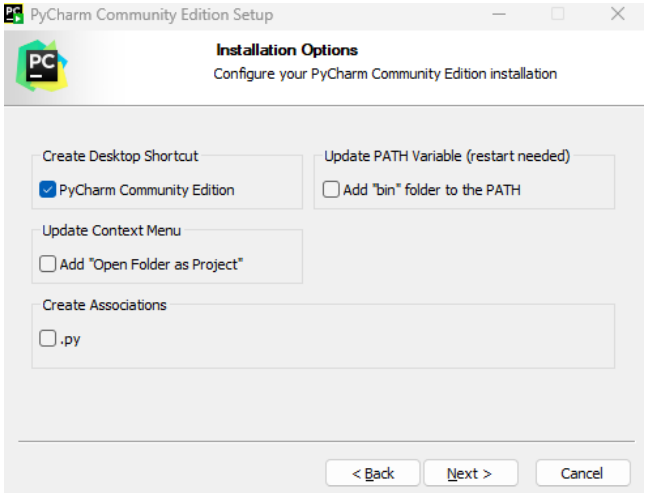
Valitaan Community-versio ja klikataan Download -painiketta.

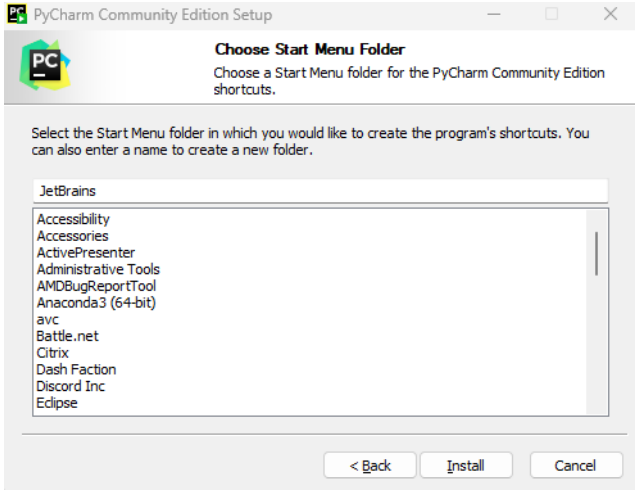


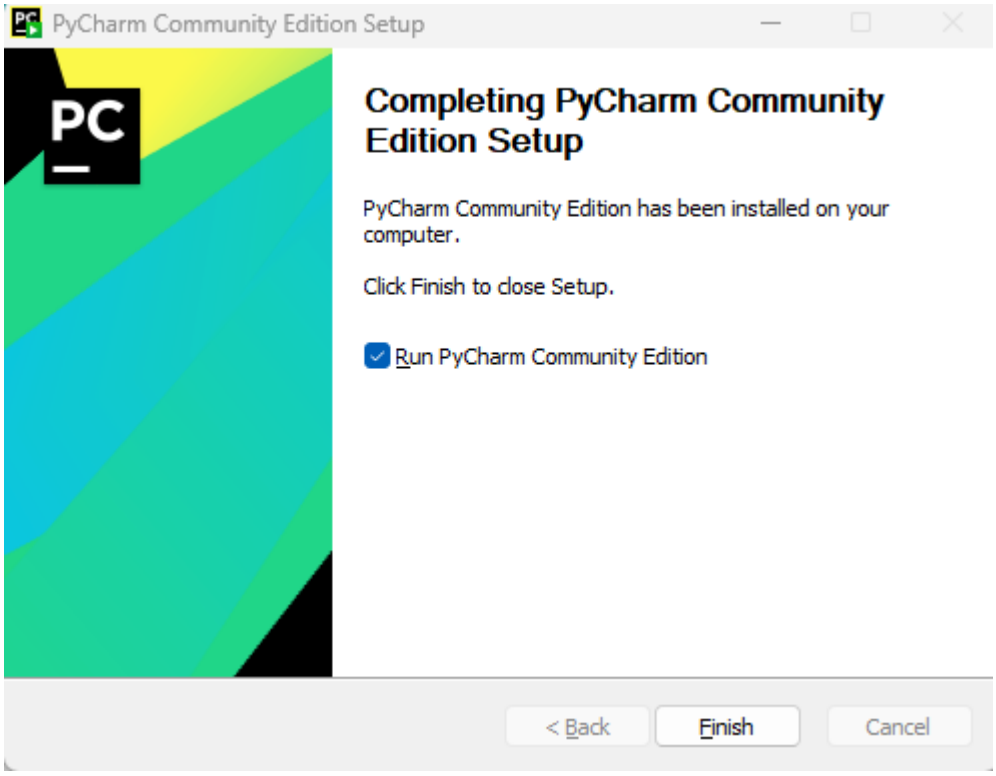
Kun asennus aloitetaan, valitaan kansio johon PyCharm asennetaan. Mikään valinnoista ei ole pakollinen `Installation Options` -näkymässä. Kannattaa valita ainakin työpöytäkuvakkeen luominen. `Choose Start Menu Folder` -osiossa voidaan painaa `Next`. Asennus kestää hetken ja lopussa voidaan valita `Run PyCharm` ja klikata `Finish`. Kun PyCharm aukeaa, käyttäjäehdot täytyy hyväksyä.

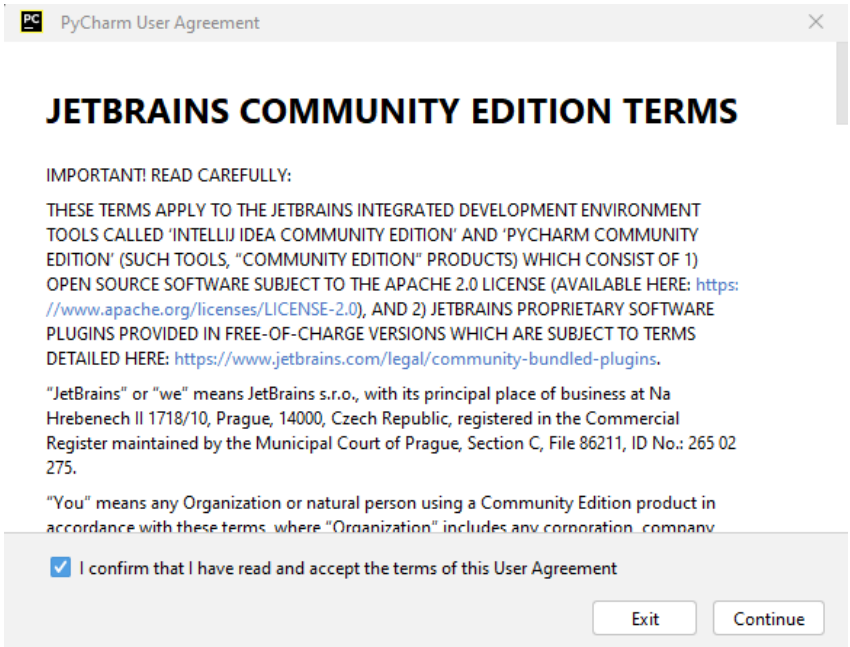



PyCharm Community Edition Setup
Choose Install Location
Choose the folder in which to install PyCharm Community Edition.
Setup will install PyCharm Community Edition in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:\Program Files\JetBrains\PyCharm Community Edition 2023.1
Browse...
Space required: 1.6 GB
Space available: 193.1 GB
< Back
Next >
Cancel

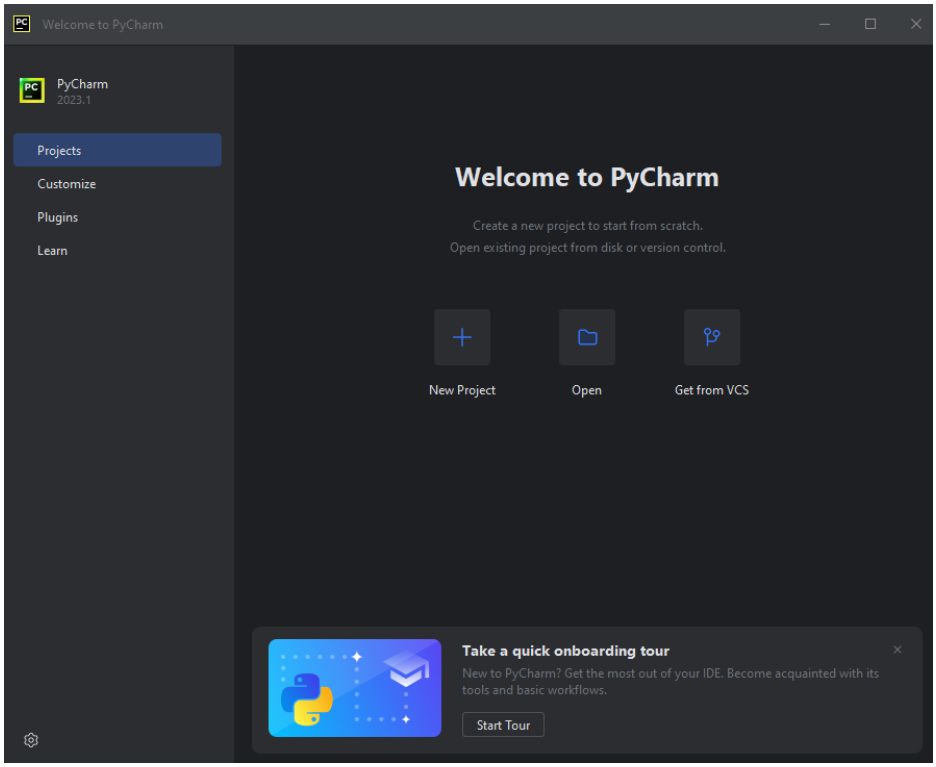

PyCharm Community Edition Setup
Installation Options
Configure your PyCharm Community Edition installation
Create Desktop Shortcut
PyCharm Community Edition
Update PATH Variable (restart needed)
Add "bin" folder to the PATH
Update Context Menu
Add "Open Folder as Project"
Create Associations
.py
< Back
Next >
Cancel

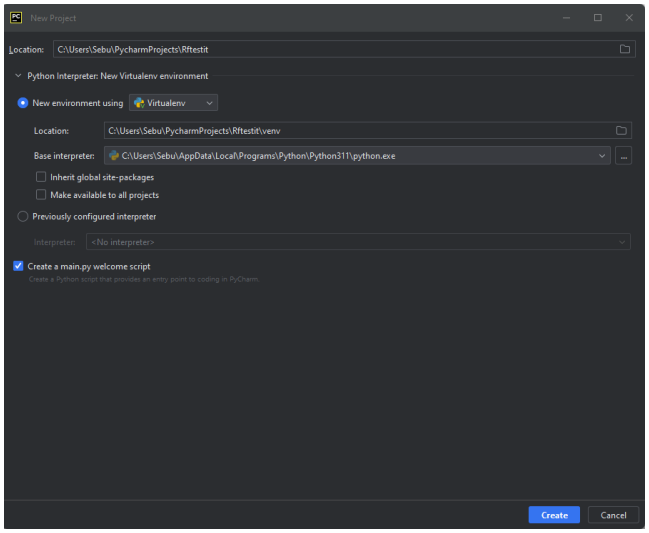

PyCharm Community Edition Setup
Choose Start Menu Folder
Choose a Start Menu folder for the PyCharm Community Edition shortcuts.
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.
JetBrains
Accessibility
Accessories
ActivePresenter
Administrative Tools
AMDBugReportTool
Anaconda3 (64-bit)
avc
Battle.net
Citrix
Dash Faction
Discord Inc
Eclipse
< Back
Install
Cancel

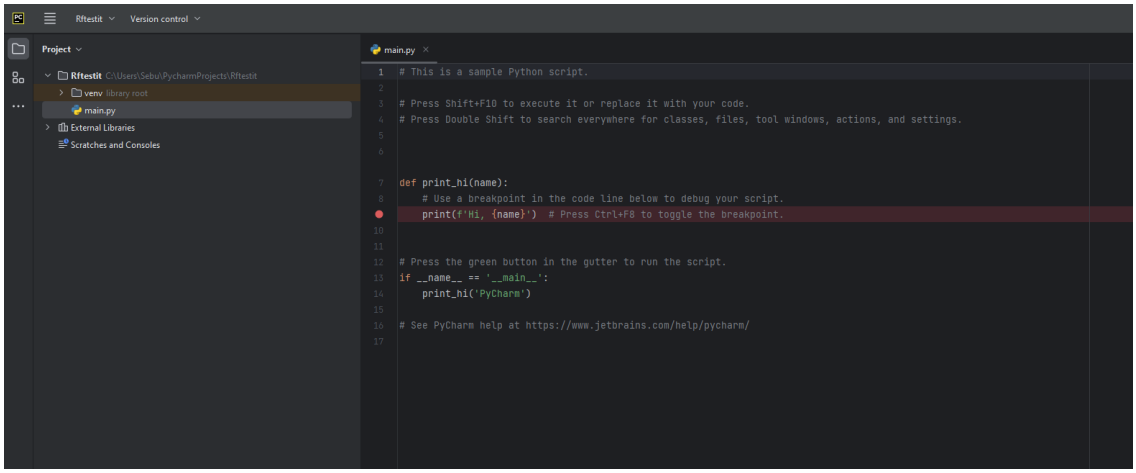




PyCharm on nyt asennettu ja avattu. `New Project` -painikkeesta voidaan luoda uusi kansio myöhemmin kirjoitettavia testejä varten. Kun uuden projektin näkymä aukeaa, kirjoitetaan kansion nimi tiedostopolkuun ja klikataan `Create`. Kuvan esimerkissä kansion nimi on `Rftestit`. Tämän jälkeen projekti on luotu testejä varten.


Welcome to PyCharm
PyCharm
2023.1
Projects
Customize
Plugins
Learn
Welcome to PyCharm
Create a new project to start from scratch.
Open existing project from disk or version control.
New Project
Open
Get from VCS
Take a quick onboarding tour
New to PyCharm? Get the most out of your IDE. Become acquainted with its tools and basic workflows.
Start Tour


New Project
Location:
C:\Users\Sebu\PycharmProjects\Rftestit
Python Interpreter: New Virtualenv environment
New environment using
Virtualenv
Location:
C:\Users\Sebu\PycharmProjects\Rftestit\venv
Base interpreter:
C:\Users\Sebu\AppData\Local\Programs\Python\Python311\python.exe
Inherit global site-packages
Make available to all projects
Previously configured interpreter
Interpreter:
<No interpreter>
Create a main.py welcome script
Create a Python script that provides an entry point to coding in PyCharm.
Create
Cancel


Rftestit
Version control
Project
Rftestit C:\Users\Sebu\PycharmProjects\Rftestit
venv library root
main.py
External Libraries
Scratches and Consoles
main.py
# This is a sample Python script.
# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.
def print_hi(name):
# Use a breakpoint in the code line below to debug your script.
print(f'Hi, {name}') # Press Ctrl+F8 to toggle the breakpoint.
# Press the green button in the gutter to run the script.
if __name__ == '__main__':
print_hi('PyCharm')
# See PyCharm help at https://www.jetbrains.com/help/pycharm/

Liite 4: Seleniumin ja Robot Frameworkin asennus

Seleniumin asentaminen on yksinkertaista. Avataan komentorivi ja ajetaan komento `pip install selenium`. Asennuksen lopussa näkyy listaus kaikista asennetuista paketeista.

```
C:\Windows\System32>pip --version
pip 22.3.1 from C:\Users\Sebu\AppData\Local\Programs\Python\Python311\Lib\site-packages\pip (python 3.11)

C:\Windows\System32>pip install selenium
Collecting selenium
  Downloading selenium-4.8.3-py3-none-any.whl (6.5 MB)
     ---------------------------------------- 6.5/6.5 MB 9.9 MB/s eta 0:00:00
Collecting urllib3[socks]~=1.26
  Downloading urllib3-1.26.15-py2.py3-none-any.whl (140 kB)
     ---------------------------------------- 140.9/140.9 kB 8.2 MB/s eta 0:00:00
Collecting trio~=0.17
  Downloading trio-0.22.0-py3-none-any.whl (384 kB)
     ---------------------------------------- 384.9/384.9 kB 12.1 MB/s eta 0:00:00
Collecting trio-websocket~=0.9
  Downloading trio_websocket-0.10.2-py3-none-any.whl (17 kB)
Collecting certifi>=2021.10.8
  Downloading certifi-2022.12.7-py3-none-any.whl (155 kB)
```

```
Administrator: Command Prompt
Collecting sniffio
  Downloading sniffio-1.3.0-py3-none-any.whl (10 kB)
Collecting cffi>=1.14
  Downloading cffi-1.15.1-cp311-cp311-win_amd64.whl (179 kB)
     ---------------------------------------- 179.0/179.0 kB 11.3 MB/s eta 0:00:00
Collecting exceptiongroup
  Downloading exceptiongroup-1.1.1-py3-none-any.whl (14 kB)
Collecting wsproto>=0.14
  Downloading wsproto-1.2.0-py3-none-any.whl (24 kB)
Collecting PySocks!=1.5.7,<2.0,>=1.5.6
  Downloading PySocks-1.7.1-py3-none-any.whl (16 kB)
Collecting pycparser
  Downloading pycparser-2.21-py2.py3-none-any.whl (118 kB)
     ---------------------------------------- 118.7/118.7 kB 6.8 MB/s eta 0:00:00
Collecting h11<1,>=0.9.0
  Downloading h11-0.14.0-py3-none-any.whl (58 kB)
     ---------------------------------------- 58.3/58.3 kB 3.2 MB/s eta 0:00:00
Installing collected packages: sortedcontainers, urllib3, sniffio, PySocks, pycparser, idna, h11, excepti
ongroup, certifi, attrs, async-generator, wsproto, outcome, cffi, trio, trio-websocket, selenium
Successfully installed PySocks-1.7.1 async-generator-1.10 attrs-23.1.0 certifi-2022.12.7 cffi-1.15.1 exce
ptiongroup-1.1.1 h11-0.14.0 idna-3.4 outcome-1.2.0 pycparser-2.21 selenium-4.8.3 sniffio-1.3.0 sortedcont
ainers-2.4.0 trio-0.22.0 trio-websocket-0.10.2 urllib3-1.26.15 wsproto-1.2.0

[notice] A new release of pip available: 22.3.1 -> 23.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Windows\System32>
```

Jos edellisessä osuudessa käytetty komentorivi ei ole auki, se täytyy avata uudelleen. Kun komentorivi on auki, aloitetaan Robot Frameworkin asennus komennolla `pip install robotframework`. Asennuksen lopussa näkyy listaus kaikista asennetuista paketeista. Asennus voidaan tarkastaa komennolla `pip show robotframework`. Kaikki asennetut paketit voidaan tarkastaa `pip list` komennolla. SeleniumLibrary puuttuu listauksesta, joten se asennetaan seuraavaksi.

```
Administrator: Command Prompt
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Windows\System32>pip install robotframework
```

```
C:\Windows\System32>pip install robotframework
Collecting robotframework
  Downloading robotframework-6.0.2-py3-none-any.whl (658 kB)
     ---------------------------------------- 658.7/658.7 kB 3.8 MB/s eta 0:00:00
Installing collected packages: robotframework
Successfully installed robotframework-6.0.2

[notice] A new release of pip available: 22.3.1 -> 23.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Windows\System32>
```

```
C:\Windows\System32>pip show robotframework
Name: robotframework
Version: 6.0.2
Summary: Generic automation framework for acceptance testing and robotic process automation (RPA)
Home-page: https://robotframework.org
Author: Pekka Klärck
Author-email: peke@eliga.fi
License: Apache License 2.0
Location: C:\Users\Sebu\AppData\Local\Programs\Python\Python311\Lib\site-packages
Requires:
Required-by:

C:\Windows\System32>
```

```
C:\Windows\System32>pip list
Package          Version
---------------- ---------
async-generator  1.10
attrs            23.1.0
certifi          2022.12.7
cffi             1.15.1
exceptiongroup   1.1.1
h11              0.14.0
idna             3.4
outcome          1.2.0
pip              22.3.1
pycparser        2.21
PySocks          1.7.1
robotframework   6.0.2
selenium         4.8.3
setuptools       65.5.0
sniffio          1.3.0
sortedcontainers 2.4.0
trio             0.22.0
trio-websocket   0.10.2
urllib3          1.26.15
wsproto          1.2.0
```

Jos edellisessä osuudessa käytetty komentorivi ei ole auki, se täytyy avata uudelleen. Aloitetaan SeleniumLibraryn asennus kommennolla `pip install robotframework-seleniumlibrary`. Asennuksen lopussa näkyy listaus kaikista asennetuista paketeista. Vapaaehtoisesti uusin pip -versio voidaan ajaa komennolla `python.exe -m pip install -upgrade pip`. Komennolla `pip list` voidaan todeta, että SeleniumLibrary on asennettu.

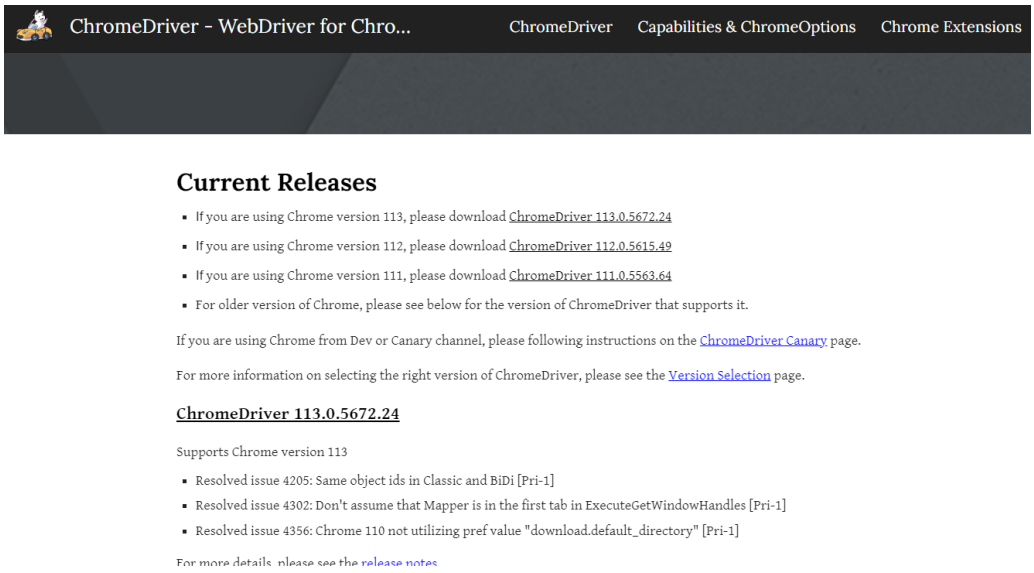
```
C:\Windows\System32>pip install robotframework-seleniumlibrary
```

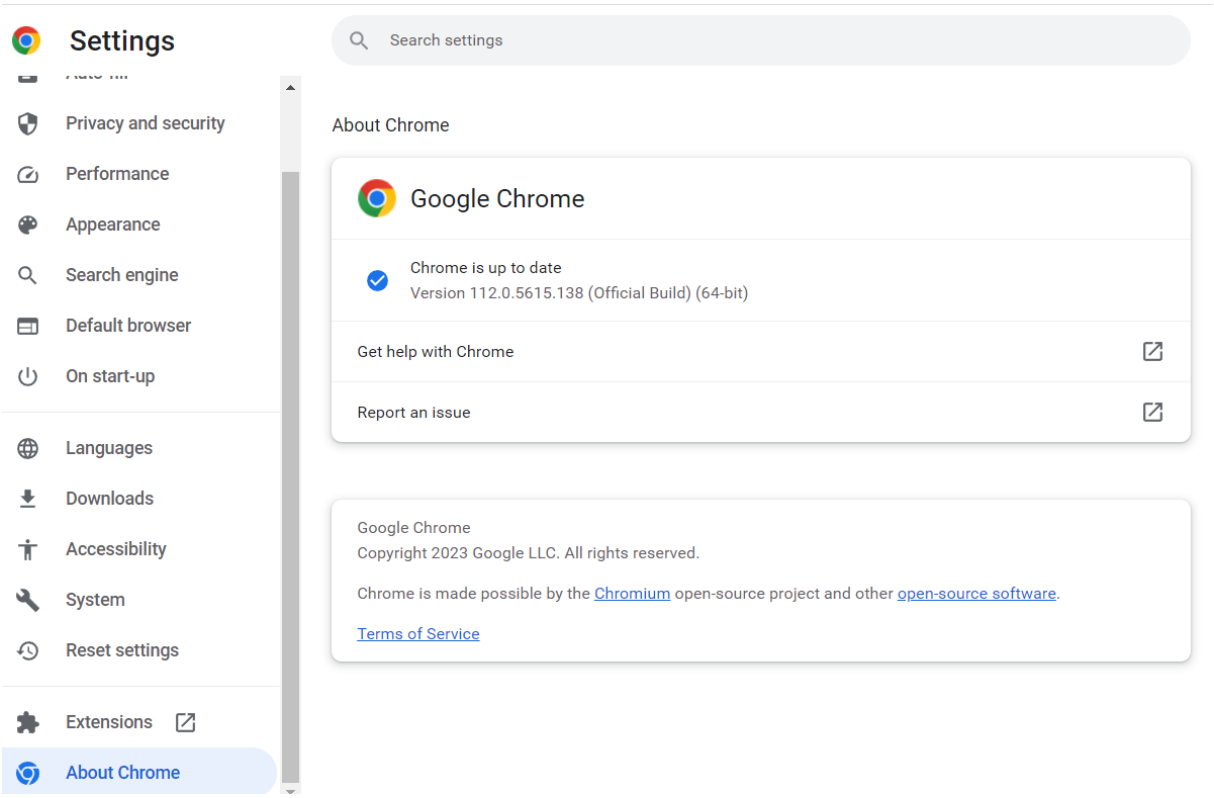
```
Requirement already satisfied: pycparser in c:\users\sebu\appdata\local\programs\python\python311\lib\
e-packages (from cffi>=1.14->trio~=0.17->selenium>=4.0.0->robotframework-seleniumlibrary) (2.21)
Requirement already satisfied: h11<1,>=0.9.0 in c:\users\sebu\appdata\local\programs\python\python311\
\site-packages (from wsproto>=0.14->trio-websocket~=0.9->selenium>=4.0.0->robotframework-seleniumlibra
 (0.14.0)
Installing collected packages: robotframework-pythonlibcore, robotframework-seleniumlibrary
Successfully installed robotframework-pythonlibcore-4.1.2 robotframework-seleniumlibrary-6.0.0

[notice] A new release of pip available: 22.3.1 -> 23.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Windows\System32>
```

```
C:\Windows\System32>pip list
Package                       Version
----------------------------- ---------
async-generator               1.10
attrs                         23.1.0
certifi                       2022.12.7
cffi                          1.15.1
exceptiongroup                1.1.1
h11                           0.14.0
idna                          3.4
outcome                       1.2.0
pip                           22.3.1
pycparser                     2.21
PySocks                       1.7.1
robotframework                6.0.2
robotframework-pythonlibcore  4.1.2
robotframework-seleniumlibrary 6.0.0
selenium                      4.8.3
setuptools                    65.5.0
sniffio                       1.3.0
sortedcontainers              2.4.0
trio                          0.22.0
trio-websocket                0.10.2
urllib3                       1.26.15
wsproto                       1.2.0
```

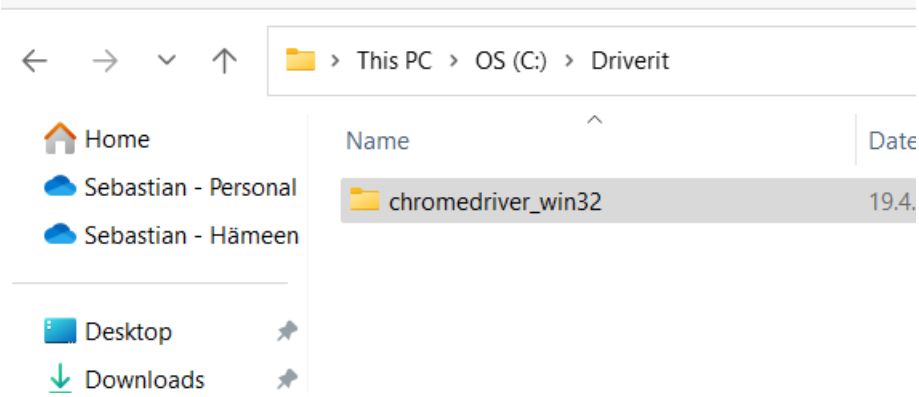
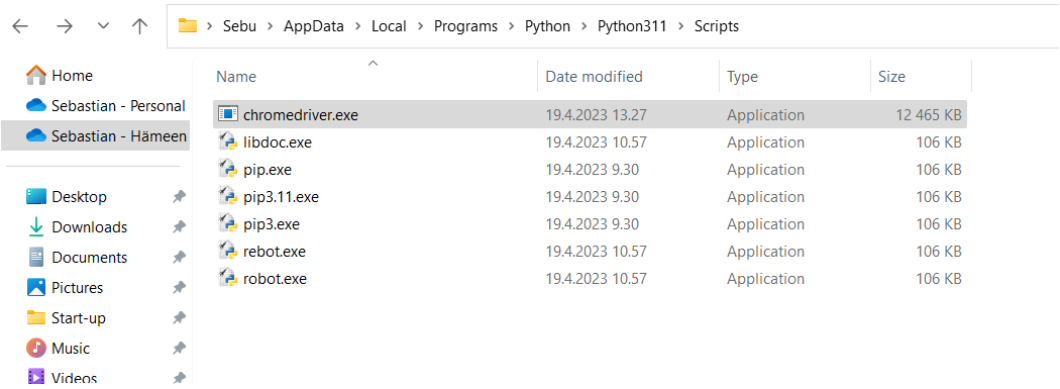
Liite 5: ChromeDriverin asennus

Viimeinen asia, joka asennetaan, on ChromeDriver. Se asennetaan osoitteesta https://chromedriver.chromium.org/downloads. ChromeDriverin versio valitaan selaimen version mukaan. Selaimen versio voidaan tarkistaa Chromen asetuksissa `Tietoja Chromesta` tai About `Chrome` -valikossa.





Valitaan `chromedriver_win32.zip`, kun käytetään Windows-käyttöjärjestelmää ja puretaan sen sisältö, jotta päästään käsiksi asennukseen tarvittavaan .exe-tiedostoon. Purettu kansio voidaan viedä esimerkiksi C-levyn juureen omaan kansioon. Navigoidaan resurssienhallinnassa purettuun kansioon ja kopioidaan chromedriver.exe Pythonin `Scripts` -kansioon.

Index of /112.0.5615.49/

Name	Last modified	Size	ETag
Parent Directory		-	
chromedriver_linux64.zip	2023-04-05 10:27:32	6.75MB	9f887638c5e8bc5313d027a802b092cc
chromedriver_mac64.zip	2023-04-05 10:27:35	8.79MB	a9f0c8afeaecc961d3bdf087f33f71c9
chromedriver_mac_arm64.zip	2023-04-05 10:27:39	8.04MB	19231ea55d8f84baf1aff6e38c6a22e9
chromedriver_win32.zip	2023-04-05 10:27:42	6.80MB	3aa0e2c3dca02a93422152248411c964
notes.txt	2023-04-05 10:27:48	0.00MB	07a5cc857188e777937ca2dcc0017fb6