Tampereen ammattikorkeakoulu

Testiautomaatio ja vertailu testiautomaatiotyökaluista

Niko Kallio

Teea Kyrölä

OPINNÄYTETYÖ
Tammikuu 2024

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

KALLIO, NIKO & KYRÖLÄ, TEEA:
Testiautomaatio ja vertailu testiautomaatiotyökaluista

Opinnäytetyö 57 sivua, joista liitteitä 2 sivua
Toukokuu 2024

Testiautomaation käyttö ohjelmistokehityksessä kasvaa jatkuvasti. Sen tarkoituksena on automatisoida ja nopeuttaa yksitoikkoisten, toistuvien testausprosessien suorittamista.

Opinnäytetyön tarkoituksena oli tutustua testi- ja web-automaatioon sekä esitellä neljä tunnettua testiautomaatiotyökalua ja vertailla niiden ominaisuuksia, dokumentaatiota sekä niiden käyttöönottoa ja helppokäyttöisyyttä web-ympäristössä. Vertailuun valittiin seuraavat neljä testiautomaatiotyökalua: Katalon, Selenium IDE, Cypress ja Robot Framework. Työn tarkoituksena on tarjota tietoa kaikille testiautomaatiosta kiinnostuneille ja helpottaa erityisesti web-automaation tarpeessa olevia sopivan työkalun valinnassa.

Työn teoriaosuudessa tutustuttiin aluksi ohjelmistotestauksen tärkeyteen ja eri testaustyyppeihin sekä pohjustettiin, mitä testi- ja web-automaatiolla tarkoitetaan. Tämän jälkeen esiteltiin vertailuun valittujen testiautomaatiotyökalujen asennus, käyttöönotto ja ominaisuudet.

Vertailuosuudessa analysoitiin työkalujen ominaisuuksia, saatavilla olevaa viralista dokumentaatiota sekä asennuksen ja käytön helppoutta. Käyttöönoton ja helppokäyttöisyyden arviointia varten laadittiin testisuunnitelma, jota noudattaen samat testit suoritettiin jokaista valittua työkalua käyttäen. Testit suoritettiin hyödyntäen valmista Automation Exercise-sivustoa, joka on luotu automaation harjoittelua varten.

Jokaisella tähän vertailuun valitulla työkalulla on varmasti paikkansa testiautomaatiossa käyttäjästä ja käyttökohteesta riippuen, joten vertailun tarkoituksena ei ollut niinkään valita näistä parasta työkalua, vaan esitellä eri vaihtoehtoja, joiden joukosta jokainen voi valita omaan käyttöönsä sopivan vaihtoehdon.

Asiasanat: testiautomaatio, ohjelmistotestaus, vertailu, web-automaatio

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

KALLIO, NIKO & KYRÖLÄ, TEEA:
Test Automation and Comparison of Test Automation Tools

Bachelor's thesis 57 pages, appendices 2 pages
May 2024

The aim of this thesis was to familiarise the reader with the basics of test and web automation and to introduce four popular test automation tools comparing their features, implementation, and user-friendliness in the web environment. The goal of this work was to provide information and assist those in need of web automation to find a suitable tool for themselves.

The theoretical section of the thesis initially explained the importance of software testing and different testing types, followed by explanations of test and web automation. Subsequently, the implementation and features of the test automation tools were explained.

In the comparison section, the features of the tools, the available official documentation, and the ease of installation and usage were analysed. To assess the ease of installation and usage, a test plan was created. By following this plan, the same tests were executed with each tool.

Each tool selected for this comparison undoubtedly has its place in test automation, depending on the user and the use case. Therefore, the purpose of the comparison was not so much to determine the best tool among them as to introduce different options from which everyone could choose the most suitable tool for their needs.

Key words: test automation, software testing, comparison, web automation

SISÄLLYS

LYHENTEET JA TERMIT

UAT	user acceptance testing
IDE	integrated development environment
CI/CD	continuous integration and continuous delivery/deployment
API	application programming interface
ALM	application lifecycle management
AI	artificial intelligence
DevOps	development and operations
BDD	behavior-driven development

1 JOHDANTO

Ohjelmistotestaus on yksi ohjelmistokehityksen keskeisimmistä vaiheista. Sen tarkoituksena on varmistaa, että ohjelmisto toimii vaatimuksien mukaisesti. Testaus parantaa ohjelmiston luotettavuutta, ja vaatimuksien mukaan toimiva ohjelmisto lisää asiakastyytyväisyyttä, joka taas on yksi jokaisen yrityksen liiketoiminnan tärkeimmistä kulmakivistä. Testausprosessin tehostamiseksi ja kustannuksien vähentämiseksi yritykset ottavat nykyään yhä useammin manuaalisen ohjelmistotestauksen rinnalle käyttöön testiautomaation.

Tämän työn tarkoituksena on johdattaa lukija ohjelmistotestauksen ja eri ohjelmistotestaustyyppien sekä testi- ja webautomaation perusteisiin, ja suorittaa vertailu neljän tunnetun testiautomaatiotyökalun välillä. Opinnäytetyön teoriaosuudessa kerrotaan aluksi siitä, mitä ohjelmistotestaus ylipäätään on, miksi sitä tehdään ja mitä eri ohjelmistotestauksen tyyppejä on olemassa ja mihin niitä käytetään, jonka jälkeen kerrotaan testi- ja web-automaatiosta, niiden hyödyistä ja merkityksestä ohjelmistotestaamisessa nykyään. Seuraavaksi esitellään vertailuun valitut testityökalut ja niiden asennus, jonka jälkeen siirrytään varsinaiseen vertailuosioon.

Vertailuosio sisältää työkalujen välisen vertailun muun muassa niiden ominaisuuksien, saatavilla olevan virallisen dokumentaation sekä niiden asennuksen ja käyttöönoton helppouden perusteella. Joidenkin vertailun tuloksien esittämisessä on havainnollistamisen helpottamiseksi hyödynnetty taulukoita. Vertailun tavoitteena on auttaa lukijaa hänelle itselleen sopivan testiautomaatiotyökalun valitsemisessa esittelemällä siihen valittujen työkalujen eroja ja ominaisuuksia sen sijaan, että työkalujen joukosta valittaisiin paras.

Vertailuosuutta varten laadittiin testisuunnitelma (liite 1), jota noudattamalla jokaisella työkalulla suoritettiin samat testit. Testisuunnitelmaan on sisällytetty yleisimmät tehtävät, joita käyttäjä web-sivustoilla vieraillessaan suorittaa. Tällaisia tehtäviä ovat muun muassa lomakkeiden täyttäminen, painikkeiden klikkailu ja navigointi sivulta toiselle. Testeissä käyttöjärjestelmänä toimi Windows 10, ja testit

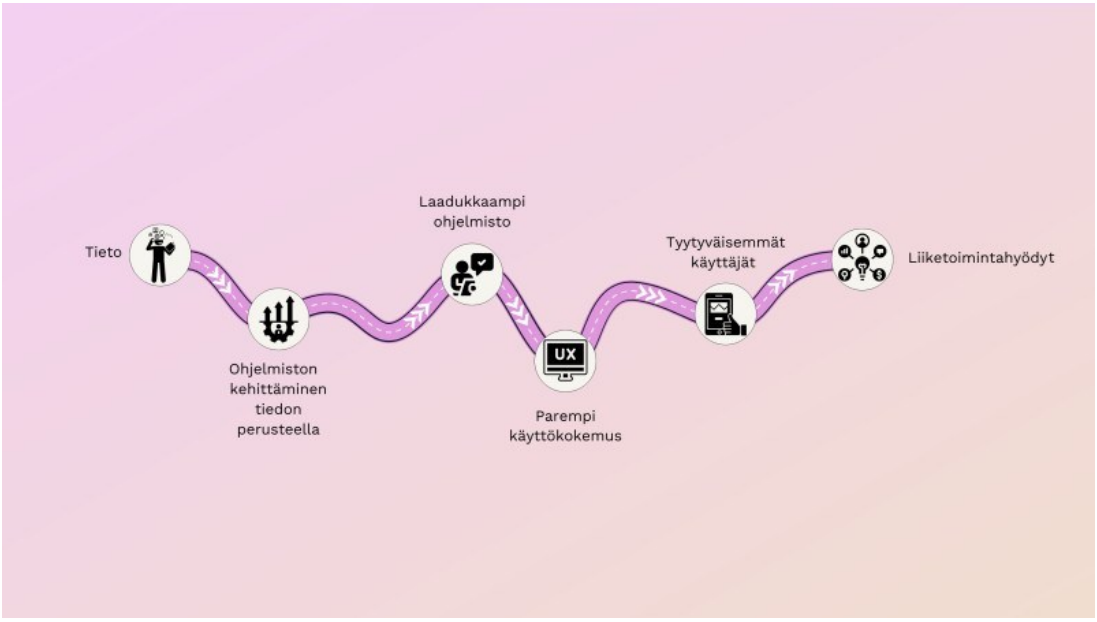
suoritettiin web-ympäristössä valmista Automation exercise -sivustoa hyödyntäen. Automation exercise -sivusto on tarkoitettu web-automaation kokeilua ja harjoittelua varten. Vertailuosion jälkeinen kappale sisältää yleistä pohdintaa pääasiassa vertailun tuloksista, testiautomaatiosta sekä siitä, miten opinnäytetyötä voisi kehittää.

Tämän työn tarkoitus on toimia tietopakettina kaikille testi- tai web-automaatiosta kiinnostuneille ja auttaa testiautomaation käyttöönottoa harkitsevia sopivan testiautomaatiotyökalun löytämisessä.

2 OHJELMISTOTESTAUS

Jose (2021) määrittelee ohjelmistotestauksen siten, että se on ohjelmistokehitykseen kuuluva vaihe, jolla pyritään varmistamaan, että ohjelmistokokonaisuus tai jokin sen osa vastaa odotuksia ja vaatimuksia. Ohjelmistotestauksen tavoitteena on siis löytää mahdollisia ongelmakohtia ja virheitä ja pyrkiä varmuuteen siitä, että ohjelmisto toimii vaatimuksien mukaisesti. Ohjelmistotestauksessa voidaan analysoida ja antaa palautetta myös ominaisuuksista, mitä vaatimuksiin ei suoranaisesti ole välttämättä kirjattu, kuten ohjelmiston helppokäyttöisyydestä tai esteettömyydestä.

Ohjelmistotestaus on tärkeä osa ohjelmistokehitysprosessia, sillä se antaa kehittäjille arvokasta palautetta ohjelmiston toimivuudesta ja sitä kautta lisää kehittäjien tehokkuutta sekä luottamusta omaan tuotteeseen. Ohjelmistotestaajilta saatu palaute antaa kehittäjille mahdollisuuden tarvittaessa korjata ja parannella ohjelmistoa. Mahdollisimman helppokäyttöinen, laadukas ja vaatimuksien mukainen ohjelmisto parantaa asiakastyytyväisyyttä, joka taas on liiketoiminnan kannalta hyvin tärkeää. Ohjelmistotestauksesta saatuihin hyötyihin sisältyy siis myös liiketoiminnallisia hyötyjä, kuten myös kuvasta 1 voidaan todeta.

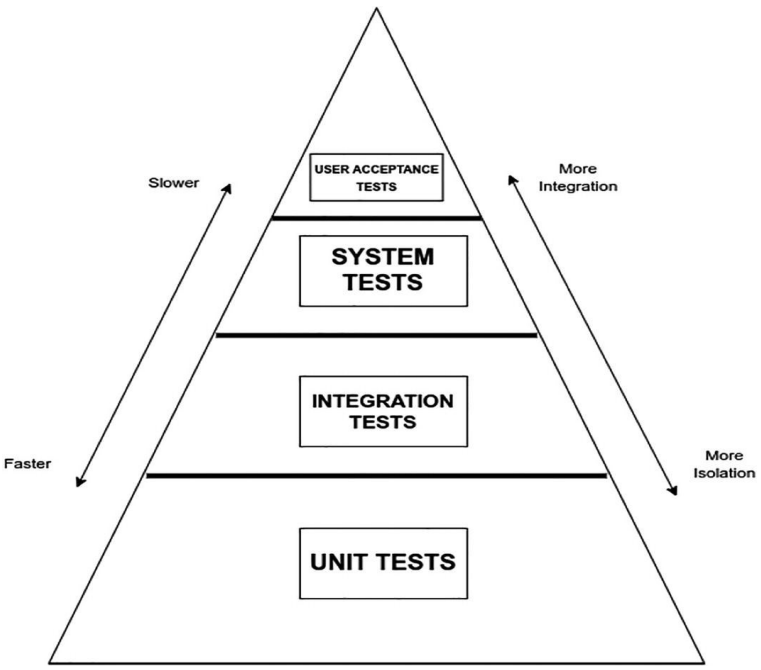


KUVA 1. Ohjelmistotestauksen hyötyjen yhteenveto. Lähde: VALA, 2022.

Ohjelmistotestausprosessissa on usein käytössä erilaisia testaustyyppejä. Eri testaustyyppien käytöllä pyritään ohjelmiston mahdollisimman kattavaan testaamiseen testaamalla ohjelmiston eri osa-alueita kehityksen eri vaiheissa ja eri tasoilla, joiden hierarkiaa usein havainnollistetaan pyramidin muodossa (kuva 2). Kattava testaus edesauttaa virheiden löytymistä mahdollisimman aikaisessa vaiheessa ja parantaa siten ohjelmiston laatua ja vähentää kehityskustannuksia.

Usein ohjelmistotestausprosessi jaetaan vähintään neljään eri tyyppiin: yksikkö-, integraatio-, järjestelmä- ja hyväksymistestaus, joita voidaan kutsua myös yhteisesti funktionaalisiksi testaustyypeiksi (Krysik, 2023). Nämä testaustyypit myös ovat usein automatisoitavissa. Muita usein automatisoitavia testaustyyppejä ovat muun muassa regressio-, savu- (eng. smoke testing) ja end-to-end–testaus.

Kuva 2 sisältää yhden tavan esittää funktionaalisten testaustyyppien tasot. Kuvasta voidaan havaita yksikkötestauksen olevan hierarkiapyramidin pohjimmainen taso. Tämä tarkoittaa, että se on testauksen aikaisin ja laajin vaihe sekä lähimpänä kehittäjärajapintaa, kun taas hyväksymistestaus korkeimpana tasona on testaamisen viimeisin vaihe ja lähimpänä asiakasrajapintaa. Mitä alemmaksi pyramidissa mennään, sitä nopeammin ja eristetymmin testit suoritetaan (Leloudas, 2023).



KUVA 2. Testaustyyppien eri tasot. Lähde: Leloudas, 2023.

2.1 Yksikkötestaus

Yksikkötestaus on ohjelmistotestauksen kaikkein alin taso. Krysikin (2023) määritelmän mukaan yksikkötestauksesta puhutaan silloin, kun ohjelmiston koodista eristetään ja testataan sen pienimpiä osia, eli yksiköitä (eng. unit). Tällaisia osia ovat esimerkiksi erilaiset funktiot, metodit, komponentit ja moduulit, joita ohjelmisto käyttää (Pittet, n.d.). Yksikkötestauksen tavoitteena on varmistaa, että jokainen yksikkö toimii vaatimuksien mukaan.

Yleensä yksikkötestaus suoritetaan kehittäjien toimesta ohjelmiston varsinaisen koodin kirjoittamisen lomassa. Tällöin testien suorittaja tietää koodin rakenteen ja voi hyödyntää sitä testien kirjoittamisessa (*white box* -testaus). (VALA, 2022.)

2.2 Integraatiotestaus

Integraatiotestaus on yleensä esitetty ohjelmistotestauksen toiseksi alimpana tasona. Leloudas (2023) määrittelee, että integraatiotestauksella tarkoitetaan yksikköjen ja komponenttien rajapintojen vuorovaikutuksen testaamista. Integraatiotestauksen tarkoituksena on siis varmistaa, että yksiköt toimivat keskenään ongelmitta.

Mahdollisten ongelmien paikantamisen helpottamiseksi integraatiotestaus tehdään tyypillisesti yksikkötestien jälkeen. Kun yksiköiden toiminta on varmistettu, on todennäköisempää, että ongelma on nimenomaan yksiköiden välisessä vuorovaikutuksen toteutuksessa, kuin itse yksiköiden toiminnassa.

2.3 Järjestelmätestaus

Järjestelmätestaus on usein esitetty testauksen toiseksi ylimpänä tasona. Järjestelmätestauksessa nimensä mukaisesti testataan koko integroitua järjestelmää.

Järjestelmätestauksen tarkoitus on varmistaa, että järjestelmäkokonaisuus toimii määriteltyjen vaatimuksien mukaan. Järjestelmätestaus suoritetaan useimmiten yksikkö- ja integraatiotestauksen jälkeen. (Desikan & Ramesh, 2007.)

2.4 Hyväksymistestaus

Hyväksymistestaus, eli UAT-testaus, on testausprosessin viimeinen taso ennen järjestelmän virallista käyttöönottoa. VALAn (2023) määritelmän mukaan hyväksymistestauksessa on kyse prosessista, jossa loppukäyttäjät testaavat ohjelmiston varmistaakseen sen toimivan suunnitellusti ja vaatimukset täyttävällä tavalla ennen ohjelmiston siirtymistä tuotantokäyttöön.

2.5 Regressiotestaus

Regressiotestaus sijoitetaan usein pyramidissa järjestelmä- ja hyväksymistestauksen väliin. Regressiotestauksen tarkoitus on varmistaa, että ohjelmistoon tehdyt muutokset eivät riko ominaisuuksia, jotka on jo aiemmin todettu toimiviksi (VALA, 2023).

2.6 Savutestaus

Yleensä savutestaus suoritetaan yksikkötestien jälkeen, ennen integraatiotestausta. Savutestaus on nopea ja hyvin yleistasoinen testausvaihe, jolla pyritään varmistamaan uuden tai päivitetyn ohjelmistoversion perustoimivuus, kuten ohjelman käynnistyminen ilman virheitä ja sen tärkeimpien ominaisuuksien toimivuus. Sen tarkoitus on ehkäistä ajan ja resurssien tuhlausta vialliseen ohjelmistoversioon. (Biswas, 2023.)

2.7 End-to-end–testaus

End-to-end–testauksessa tarkoituksena on nimensä mukaisesti testata ohjelmisto päästä päähän, eli varmistaa, että koko ohjelmistokokonaisuus toimii vaatimuksien mukaisesti. End-to-end–testauksessa simuloidaan todellisia käyttötilanteita ja prosesseja käyttäjän näkökulmasta, joka tekee siitä tärkeän osan testausprosessia.

End-to-end–testauksen tarkoituksena on pyrkiä löytämään ohjelmiston toiminnasta mahdollisia virheitä, joita on voinut syntyä yhdistäessä ohjelmiston eri osat integroiduksi kokonaisuudeksi. (Katalon, n.d.)

End-to-end–testaus tehdään yleensä yksikkö-, integraatio- ja järjestelmätestien jälkeen. Sitä ei kuitenkaan tule sekoittaa hyväksymistestaukseen, sillä end-to-end–testaus suoritetaan käyttäjien sijaan kehitystiimiin kuuluvien ohjelmistotestaajien tai muiden testausasiantuntijoiden toimesta.

3 TESTIAUTOMAATIO

Testiautomaatiosta puhutaan, kun testaukseen otetaan avuksi ohjelmistoja, jotka suorittavat ohjelmiston käyttäjätoimintoja automaattisesti sen sijaan, että ohjelmistotestaaja tekisi manuaalisesti tarvittavat toimet testin suorittamiseksi (Jose 2021). Testiautomaation tavoitteena on helpottaa ja tehostaa testausprosessia sekä vähentää testauksesta aiheutuvia kustannuksia, jonka vuoksi testiautomaation käyttö yrityksissä yleistyy jatkuvasti (Leloudas, 2023). Testiautomaatio mahdollistaa oikein käytettynä ohjelmistojen laajemman ja tarkemman testauksen, kuin mitä pelkällä manuaalisella testaamisella saataisiin aikaan.

Jotta testiautomaatiosta saataisiin mahdollisimman suuri hyöty, on ennen sen käyttöönottoa tärkeää selvittää, onko se ylipäätään kannattavaa. Meszaros (2007) painottaa teoksessaan, että ennen testiautomaation käyttöönottoa tulee varmistaa, ettei se lisää ohjelmiston kehityskustannuksia. Tämä tarkoittaa sitä, että vaikka testiautomaation käyttöönotosta koituukin kuluja, tulisi siitä pidemmällä aikavälillä saatu taloudellinen hyöty kuitenkin ylittää käyttöönottokustannukset. Testiautomaatiota kannattaa hyödyntää ohjelmiston toiminnan todentamisessa erityisesti silloin, kun testataan yksitoikkoisia, toistuvia testitapauksia, joissa manuaalinen testaus olisi aikaa vievää tai muuten hankala suorittaa perusteellisesti. Tällaisten prosessien testaaminen voi ihmisen suorittamana johtaa helposti inhimillisiin virheisiin tai puutteelliseen testaukseen.

Testiautomaatio on tärkeässä roolissa erityisesti ketterässä (eng. agile) ohjelmistokehityksessä, jossa ohjelmistoprojektin kehitys tehdään lyhyissä sykleissä, eli sprinteissä. Tällöin uusia versioita tulee testattavaksi lyhyin väliajoin, ja testaamisen tulisi olla tehokasta ja nopeaa. Tämä voi pelkällä manuaalisella testaamisella tuottaa ohjelmistokehitykseen paineita ja hidastaa kehitysprosessia. Automaattisen testaamisen avulla jokainen uusi versio voidaan testata perusteellisesti nopeallakin aikataululla.

Testiautomaatiotyökalua valitessa tulee kiinnittää huomiota omat tarpeet huomioiden muun muassa työkalun skaalautuvuuteen eri ympäristöissä (esim. eri lait-

teet tai verkkoselaimet), helppokäyttöisyyteen, käyttöönotto- ja käyttökustannuksiin sekä siihen, onko saatavilla tarvittaessa tukea tai ohjeita työkalun käyttöön (esim. dokumentaatio). Lisäksi ennen valintaa tulee varmistaa testityökalun yhteensopivuus mahdollisten muiden kehityksessä käytettävien työkalujen kanssa. (Jose, 2021.)

Testiautomaatiotyökalut tarjoavat monia erilaisia toimintoja testiskriptien luomiseen ja niiden suorittamiseen. Yleisimpiä menetelmiä testien luomiseen ovat skriptaus (eng. scripting), avainsanapohjainen testaus (eng. keyword driven testing) sekä tallennus- ja toistotoiminto (eng. record and replay). Suurin osa työkaluista mahdollistaa eri toimintojen yhdistämisen, jolloin puhutaan hybriditestityökaluista (eng. hybrid tools). Eri toimintoja sovelletaan testiautomaatiossa eri tilanteissa muun muassa testitapauksen monimutkaisuudesta sekä käyttäjän osaamisesta riippuen.

3.1 Skriptaus

Testiautomaatiossa puhutaan skriptaamisesta silloin, kun testitapauksen suorittamista varten luodaan tiedosto, johon kirjoitetaan manuaalisesti koodina vaiheet, joita testityökalu kyseistä tiedostoa ajaessaan seuraa suorittaakseen testin (Jain, 2024). Tällaista tiedostoa kutsutaan testiskriptiksi.

Testien suorittaminen skriptaamalla vaatii käyttäjältä kohtalaista osaamista jostakin ohjelmointikielestä, mitä käytettävä testityökalu tukee. Tämä voi käyttäjän osaamisesta riippuen johtaa tiettyjen vaihtoehtojen karsimiseen sopivaa testityökalua valitessa.

Skriptaus tarjoaa enemmän joustavuutta ja muokattavuutta testeihin kuin avainsanapohjainen testaus tai tallennus- ja toistotoiminto. Muokattavuuden vuoksi skriptaus on usein testien ylläpidon kannalta tehokkaampi ratkaisu, sillä ohjelmistoon tehdyt muutokset on helppo lisätä tai muokata olemassa oleviin testitapauksiin. Käsin koodatut testiskriptit mahdollistavat myös hyvin monimutkaisten, skaalautuvien tai tarkkojen testien suorittamisen, joka etenkään tallennus- ja toistotoimintoa käyttäen ei välttämättä ole aina kannattavaa tai edes mahdollista (Leloudas, 2023).

3.2 Tallennus ja toisto

Osa testiautomaatiotyökaluista sisältää toiminnon, jolla voidaan tallentaa web- tai mobiilisovelluksissa manuaalisesti suoritettuja toimintoja. Tämä toiminto tallentaa esimerkiksi käyttäjän tekemät painallukset ja tekstikenttien täyttämiset, ja luo näistä automaattisesti testiskriptin (Leloudas, 2023). Tallennustoiminnolla luotua skriptiä voidaan tarpeen mukaan toistaa myöhemmin automaattisesti, kun halutaan varmistua siitä, että tallenteessa suoritettu toiminto toimii yhä esimerkiksi koodiin tehtyjen muutoksien jälkeen.

Tallennus- ja toistotoiminto eroaa muista yleisistä testiautomaatiotoiminnoista siten, että sen käyttöön tarvitaan joko hyvin vähän tai ei lainkaan koodaustaitoa tai kokemusta testiautomaation käytöstä (Test Evolve, n.d.). Tämän vuoksi se sopii hyvin erityisesti aloitteleville testiautomaation käyttäjille. Tallennus- ja toistotoiminto sopii käytettäväksi erityisesti yksinkertaisten, muuttumattomien ominaisuuksien tai toimintojen testaamiseen.

Mikäli testattavaan sovellukseen tehdään usein muutoksia, ei välttämättä ole kannattavaa käyttää tallennusta ja toistoa ainakaan ainoana testaustapana, sillä erityisesti sovelluksen rakenteeseen ja käyttöliittymään tehdyt muutokset aiheuttavat helposti tallennus- ja toistotoiminnolla luotujen testiskriptien vanhentumisen. Vanhentuneet skriptit ovat epäluotettavia, joten ne täytyy päivittää, joka voi käytetystä työkalusta ja tehtyjen muutoksien merkittävyydestä riippuen olla hankalaa. Joissakin tapauksissa tallennus- ja toistotoiminnolla luodut skriptit joudutaan luomaan kokonaan uudelleen. Tämä on aikaa vievää, joten merkittävästi ja usein muuttuvan sovelluksen testaamisessa voi tallennus- ja toistotoiminnon käytöstä olla pidemmällä aikavälillä kustannuksien ja ajankäytön kannalta enemmän haittaa kuin hyötyä.

3.3 Avainsanapohjainen testaus

Avainsanapohjainen testaus on yleinen skriptauspohjainen testiautomaatiomenetelmä, jossa ohjelman tai sovelluksen testien suunnittelu, toteutus ja suorittaminen tehdään pääasiassa avainsanoja käyttäen. Suurin osa avainsanapohjaista

menetelmää käyttävistä testaustyökaluista sisältää valmiin kokoelman ennalta määriteltyjä avainsanoja yleisimpien toimintojen, kuten selaimen avaamisen, tekstikentän täyttämisen tai napin klikkaamisen, suorittamiseksi (Khan, n.d.), mutta useimmilla työkaluilla voidaan tarvittaessa myös luoda ja määritellä itse uusia avainsanoja omien tarpeiden mukaan. Avainsanat kuvaavat testitapauksissa suoritettavia toimintoja, kuten *avaa_selain* tai *täytä_lomake*.

Avainsanapohjaisen testaamisen tehokkuus perustuu avainsanojen uudelleenkäytettävyyteen, joka mahdollistaa perusteellisen ja kattavan testaamisen. Lisäksi avainsanapohjaisia testiskriptejä on helppo muokata ja ylläpitää. (Khan, n.d.)

Mikäli testitapauksen vaatimat avainsanat on valmiiksi määritelty, sopii avainsanapohjainen testausmenetelmä hyvin myös aloittelevalle tai vähäisen ohjelmointiosaamisen omaavalle testiautomaation käyttäjälle. Mikäli testitapaus taas vaatii uusien avainsanojen luomisen ja määrittelyn, on käyttäjällä hyvä olla kohtalaista ohjelmointiosaamista.

3.4 Hybriditestityökalut

Testiautomaatiossa puhutaan hybriditestityökalusta silloin, kun samalla testityökalulla voidaan hyödyntää useampaa eri testausmenetelmää. Usein tämä käytännössä tarkoittaa sitä, että tallennus- ja toistotoiminnon rinnalle otetaan käyttöön jokin skriptauspohjainen testausmenetelmä. (Leloudas, 2023.)

Hybridityökaluissa yhdistyvät tallennus- ja toistotoiminnon helppokäyttöisyys ja skriptauspohjaisten menetelmien muokattavuus ja joustavuus. Nämä kaksi yhdessä mahdollistavat laajempien ja kattavampien testien suorittamisen, kuin mitä niillä yksinään saisi aikaan.

Hybridityökalut ovat suosittuja myös niiden käyttäjäystävällisyyden vuoksi. Monet hybridityökalut tarjoavat käyttäjilleen alustan (IDE), jolla voidaan helposti luoda testitallenteita ja muokata skriptejä.

Kuten aiemmin mainituissa skriptauspohjaisissa menetelmissä, myös hybridityökalun käyttäjällä on hyvä olla vähintään keskinkertaista ohjelmointiosaamista. Tällöin hybridityökalusta saadaan testaamisessa irti mahdollisimman suuri hyöty.

4 WEB-AUTOMAATIOTESTAUS

Angappan (n.d.) määritelmän mukaan web-automaatiolla viitataan skriptien ja työkalujen käyttämiseen erilaisten tehtävien suorittamiseksi automaattisesti verkossa. Tällaisia tehtäviä voivat olla esimerkiksi verkkosivulla navigointi, sisällön tarkistaminen, painikkeiden tai linkkien klikkailu, tiedostojen lataaminen, lomakkeiden täyttäminen ja muut yleiset prosessit, joita käyttäjä usein sivustolla vierailessaan suorittaa. Tätä ominaisuutta hyödyntäen voidaan web-automaatiota käyttää verkkosivujen tai -sovellusten testaamisessa, jolloin puhutaan web-automaatiotestauksesta.

Web-automaatiotestaus on testiautomaation muoto, joka eroaa muusta testiautomaatiosta siten, että se keskittyy pääasiassa käyttäjätoimintojen simuloimiseen verkkoympäristössä. Web-automaatiotestit integroidaan usein osaksi CI/CD-prosessia, yleensä siten, että testit suoritetaan aina, kun koodiin tehdyt muutokset yhdistetään pääkehityshaaraan (eng. main). Tällöin kehittäjät saavat jatkuvaa ja nopeaa palautetta tehtyjen muutoksien toimivuudesta, jonka ansiosta mahdolliset viat saadaan korjattua mahdollisimman varhaisessa vaiheessa.

Testaamisen lisäksi web-automaatiota voidaan hyödyntää esimerkiksi verkkokauppojen tilauksien käsittelyssä, automaattisessa markkinoinnissa (mainosten optimointi, sähköpostimarkkinointi) ja evästeiden hallinnassa.

Käyttökohteesta riippumatta web-automaatio jakaa testiautomaation kanssa pääpiirteittäin samat hyödyt ja tavoitteet: ajan säästäminen, tehokkuuden lisääminen, tuotteen tai palvelun laadun parantaminen sekä kustannuksien vähentäminen.

5 KATALON

Katalon on vuonna 2016 KMS Technologyn julkaisema testiautomaatiotyökalu, joka on tarkoitettu APIen sekä työpöytä-, web- ja mobiilisovellusten (iOS ja Android) testauksen automatisointiin. Katalonia voidaan käyttää kattamaan useimmat testausvaiheet, kuten testien alustamisen, luomisen, ajamisen, testitulosten analysoinnin, raporttien luomisen ja testien ylläpidon. (Katalon, n.d.)

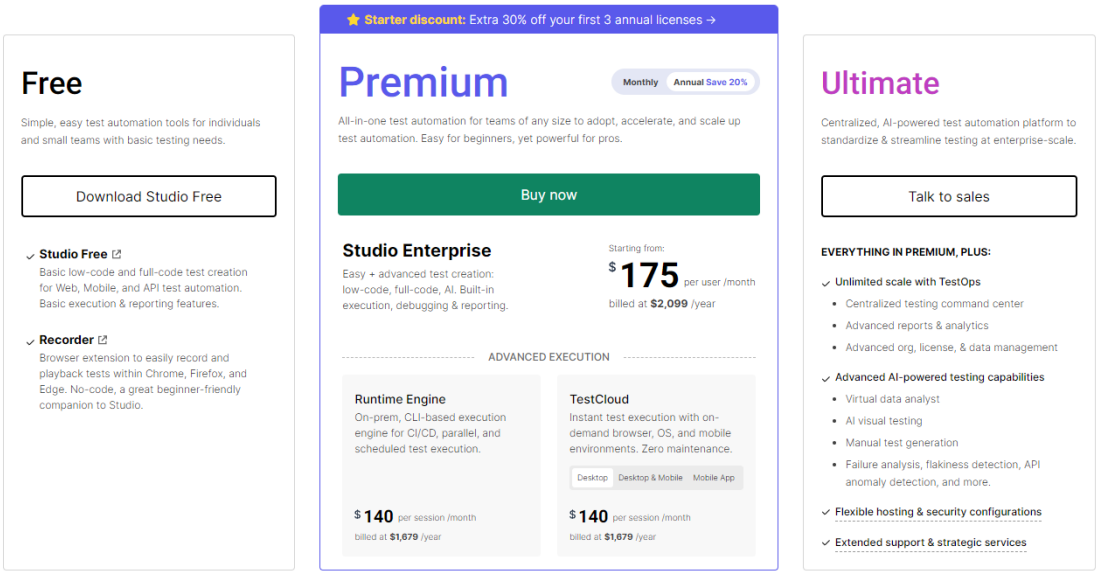
Katalonin suosio perustuu sen skaalautuvuuteen, helppokäyttöisyyteen ja aloittelijaystävällisyyteen. Katalon tarjoaa työkalun opetteluun muun muassa GitHub-repositorioita, Udemy-verkkokursseja ja YouTube-opetusvideoita (AltexSoft, 2021).

Katalon voidaan myös integroida osaksi lukuisia eri pilvipalveluita (mm. AWS Device Farm, BrowserStack ja Docker image), ALM-työkaluja (mm. Git, Jira, Rally ja Slack), CI/CD-putkia (mm. AWS CodeBuild, Azure DevOps, BitBucket ja Jenkins), ohjelmistokehyksiä (eng. framework) (mm. Cypress, Playwright, Pytest ja Mocha) sekä muita testityökaluja, kuten Selenium, Postman ja JUnit (Katalon Docs, n.d.).

Mahdollisesti Katalonin suurin heikkous moniin muihin suosittuihin testiautomaatiotyökaluihin verrattuna on, että se tukee tällä hetkellä ainoastaan Groovy- ja Java-ohjelmointikieliä. Toisaalta Katalon painottaa työkalunsa olevan *vähäkoodinen* (eng. low-code), sillä se tarjoaa useita eri työkaluja, joilla testejä voidaan luoda ja ylläpitää ilman ohjelmointiosaamista (Katalon Docs, n.d.). Katalon mahdollistaa esimerkiksi yksinkertaisten testien luomisen tallennus- ja toistotoimintoa käyttäen.

Katalonilla on tarjolla kolme eri tilausvaihtoehtoa: *Free*, *Premium* ja *Ultimate* (kuva 3). Tilausmuodosta riippuen Katalon tarjoaa käyttäjälle testaamiseen seuraavanlaisia palveluja ja työkaluja:

- *Katalon Studio* -IDE, joka mahdollistaa testien luomisen skriptauspohjaisesti, tallennus- ja toistotoimintoa käyttäen sekä tilausmuodosta riippuen AI:n avulla
- Katalon Studion *Runtime Engine* -lisäosa, joka mahdollistaa testien ajoittamisen ja suorittamisen komentorivitilassa
- *TestCloud*-pilvipalvelu, joka mahdollistaa testien suorittamisen pilvessä
- *TestOps*-sovellus, joka on tarkoitettu testien ja DevOpsin yhteensovittamiseen. TestOps mahdollistaa edistyneiden raporttien ja analysoinnin tekemisen sekä lisenssien hallinnan ja datanhallinnan.



KUVA 3. Katalonin tilausmuodot, niiden sisältämät ominaisuudet ja hinnoittelu.

Free on ilmainen tilausmuoto, jota Katalon markkinoi sopivaksi perustasoisen testaustarpeen omaaville henkilöille tai pienille tiimeille. Free-tilausmuoto tarjoaa käyttäjälle Katalon Studion ilmaisversion, joka sisältää tallennus- ja toistotoiminnon sekä mahdollistaa perustasoisten skriptien kirjoittamisen manuaalisesti APIen sekä web- ja mobiilisovelluksien testaamista varten. Ilmaisversio sisältää perustason ominaisuudet testien luomiseen ja suorittamiseen sekä tuloksien raportointiin. (Katalon, n.d.)

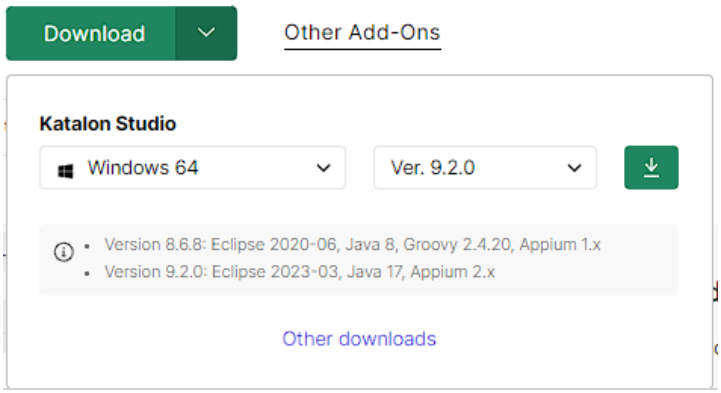
Premium-tilausmuotoa Katalon markkinoi sopivaksi kaikenkokoisille tiimeille, niiden testitarpeiden laajuudesta ja testiautomaation osaamistasosta riippumatta. Premium-tilausmuoto sisältää Katalon Studion maksullisen, yrityskäyttöön tarkoitetun version. Katalon Studion yritysversio sisältää ilmaisversion ominaisuuksien

lisäksi myös edistyneempiä työkaluja ja ominaisuuksia, jotka helpottavat testien luomista ja suorittamista. Yritysversio mahdollistaa esimerkiksi epäonnistuneiden testien automaattisen uudelleenajon sekä testiskriptien luomisen AI:n avulla. Premium-tilaukseen voi myös sisällyttää Katalonin Runtime engine -lisäosan sekä TestCloud -sovelluksen. (Katalon, n.d.)

Katalon markkinoi Ultimate-tilausmuotoa sopivaksi erityisesti suurille organisaatioille, jotka ovat muuttamassa testauskäytäntöjään tai tehostamassa automatisoitua testaamistaan koko yrityksen laajuisesti. *Ultimate* pitää sisällään kaikkien Premium-tilausmuodon ominaisuuksien lisäksi TestOps-sovelluksen sekä *Premiumia* edistyneemmän AI-teknologian käytön muun muassa datan tai vikojen analysointia varten. (Katalon, n.d.)

5.1 Asennus ja käyttöönotto

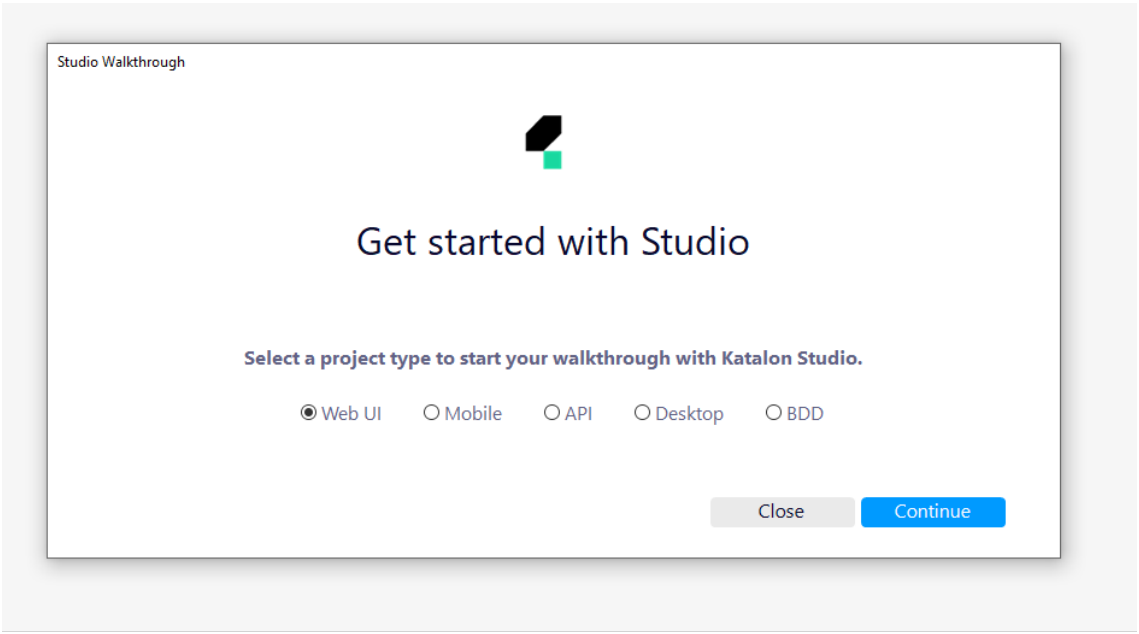
Katalonin asennusprosessi on hyvin yksinkertainen, eikä se vaadi ohjelmointiosaamista tai komentorivityöskentelyä. Katalonin käyttöönottamiseksi tulee käyttäjän ensin asentaa Katalon Studion asennusohjelma, joka hoituu Katalonin viralliselta verkkosivustolta. *Download*-painike avaa asennusvalikon, jossa valitaan haluttu Katalon-versio sekä käyttöjärjestelmä, jolle Katalon asennetaan (kuva 4).



KUVA 4. Katalon Studion asennusvalikko.

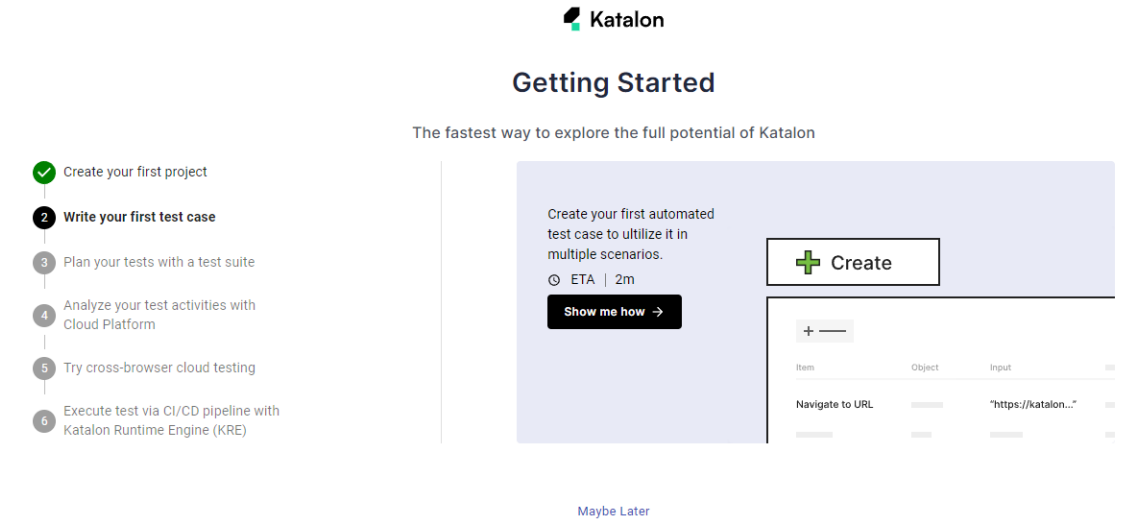
Asennuksen valmistuttua ajetaan asennettu *KatalonSetup.exe*-asennusohjelma, joka asentaa ja alustaa Katalon Studion automaattisesti laitteelle. Asennuksen valmistuttua Katalon Studio on valmis käytettäväksi.

Kun Katalon Studio avataan ensimmäistä kertaa, tarjoaa Katalon alkuun pääsemiseksi IDEn läpikäynnin projektityypin mukaan (web-, työpöytä-, mobiili-, API- tai BDD-projekti) (kuva 5).



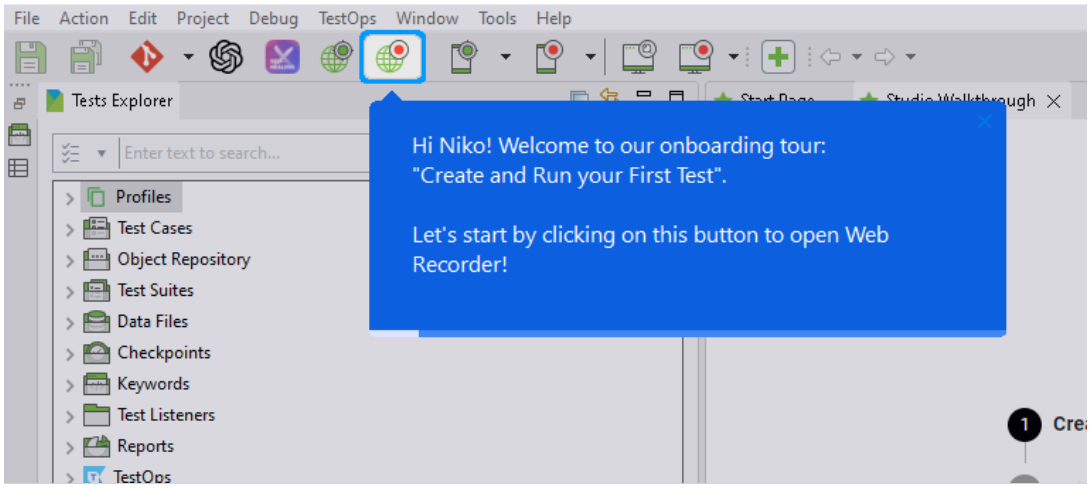
KUVA 5. Katalonin *Get started with Studio* -ikkuna läpikäynnin aloitukseen.

Läpikäynnissä käyttäjä suorittaa ohjeistettuja osa-alueita, joiden tarkoituksena on esitellä Katalonin ominaisuuksien käyttöä valitun projektityypin mukaan. Kuvasta 6 voidaan nähdä, miltä läpikäynti näyttää esimerkiksi *Web UI*-projektityypillä.



KUVA 6. Läpikäynnin aloitus *Web UI*-projektityypillä.

Läpikäynnin ohjeistus tapahtuu pääasiassa ponnahdusikkunoiden avulla (kuva 7). Nämä ponnahdusikkunat esittelevät käyttäjälle Katalonin eri ominaisuuksia ja työkaluja sekä ohjeistavat niiden käyttöä. Ne sisältävät myös muuta hyödyllistä tietoa, kuten erilaisia ohjelmistotestaukseen ja testiautomaatioon liittyviä termejä. Katalonin tarjoama läpikäynti on hyvin yksinkertainen ja helposti seurattava myös aloittelevalle testiautomaation käyttäjälle.



KUVA 7. Esimerkki Katalonin läpikäynnin ohjeistuksesta.

5.2 Testien tekeminen Katalonilla

Katalonin helppokäyttöisyyttä analysoitiin suorittamalla opinnäytetyötä varten luodun testisuunnitelman (liite 1) mukaiset testit. Testiskriptien luomiseen käytettiin pääasiassa Katalonin tallennus- ja toistotoimintoa sekä avainsanapohjaista testausta (kuva 9).

Katalon mahdollistaa tallennus- ja toistotoiminnon avulla automaattisesti luotujen skriptien (kuva 8) muokkaamisen manuaalisesti jälkikäteen, jolloin skriptiin voidaan lisätä testausvaiheita, mitä pelkällä tallennuksella ei voi tehdä, kuten testin automaattinen uudelleen ajaminen, mikäli sen suorittaminen epäonnistuu. Tämän ominaisuuden ansiosta skriptistä voidaan myös tarvittaessa poistaa vaiheita, kuten mahdollisia tallenteeseen tallentuneita virhepainalluksia.

```
import static com.kms.katalon.core.checkpoint.CheckpointFactory.findCheckpoint

WebUI.openBrowser('')

WebUI.navigateToUrl('https://automationexercise.com/')

WebUI.verifyElementPresent(findTestObject('Page_Automation Exercise/h2_Features Items (1)'), 0)

WebUI.verifyElementPresent(findTestObject('Page_Automation Exercise/h2_recommended items'), 0)

int maxRetries = 3 // Maximum number of retries

int retries = 0 // Initialize retry counter

while (retries < maxRetries) {
    try {
        WebUI.click(findTestObject('Object Repository/Page_Automation Exercise/a_View Product (2)'))

        WebUI.verifyTextPresent('Condition: New', false)

        WebUI.verifyTextPresent('Brand: Madame', false)

        WebUI.setText(findTestObject('Object Repository/Page_Automation Exercise - Product Details/input_quantity'), '4')

        WebUI.click(findTestObject('Object Repository/Page_Automation Exercise - Product Details/button_Add to cart'))

        WebUI.click(findTestObject('Object Repository/Page_Automation Exercise - Product Details/u_View Cart'))

        WebUI.verifyElementPresent(findTestObject('Page_Automation Exercise - Checkout/button_4'), 0)

        break
    }
    catch (Exception e) {
        WebUI.comment("Exception occurred: $e.getMessage()")

        retries++

        if (retries < maxRetries) {
            WebUI.comment("Retrying test (attempt $retries of $maxRetries)")
        } else {
            WebUI.comment('Maximum number of retries reached. Test failed.')
        }
    }
    // Retry the test if random pop up causes error
    finally {
        WebUI.closeBrowser()
    }
}
```

KUVA 8. Esimerkki Katalonin tallennus- ja toistotoimintoa hyödyntäen luodusta testiskriptistä.

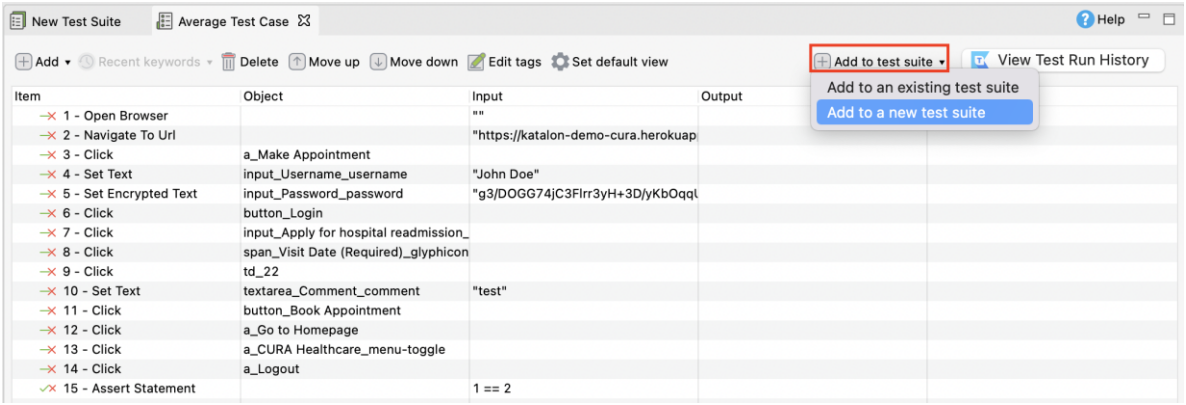
Testeissä hyödynnettiin myös Katalonin *Spy Web*-työkalua, joka mahdollistaa testiobjektien älykkään sieppauksen web-ympäristössä. Spy Webillä siepatut objektit voidaan tallentaa, jolloin niitä voidaan hyödyntää myöhemmissä testeissä. Tätä ominaisuutta hyödynnettiin esimerkiksi kuvan 9 testitapauksen kohdassa 11, *Verify Element Present*, jossa testiobjekti *button_4* haettiin verkkosivulta *Spy Web*-työkalua hyödyntäen. Spy Webin käyttö testiobjektien poimimisessa nopeuttaa testien tekemistä, sillä se eliminoi tarpeen etsiä manuaalisesti esimerkiksi objektin nimen, ID:n tai muun ominaisuuden, jolla kyseinen objekti voidaan paikantaa.

Item	Object	Input
1 - Open Browser		""
2 - Navigate To Url		"https://automationexercise.com/"
3 - Verify Element Present	h2_Features Items (1)	0
4 - Verify Element Present	h2_recommended items	0
5 - Click	a_View Product (2)	
6 - Verify Text Present		"Condition: New"; false
7 - Verify Text Present		"Brand: Madame"; false
8 - Set Text	input_quantity	"4"
9 - Click	button_Add to cart	
10 - Click	u_View Cart	
11 - Verify Element Present	button_4	0
12 - Close Browser		

KUVA 9. Esimerkki Katalonilla tehdystä avainsanapohjaisesta testistä.

Katalonin tarjoaman läpikäynnin sekä helppolukuisen dokumentaation avulla edellä mainittuja testausmenetelmiä ja -työkaluja hyödyntämällä testisuunnitelman mukaiset testit saatiin luotua ja suoritettua melko vaivattomasti. Näihin perehtymällä testien luominen ja suorittaminen onnistuu Katalonilla jopa ilman aiempaa ohjelmointiosaamista.

Testien suorittamisen tehostamiseksi Katalonilla voidaan tarvittaessa myös helposti yhdistää erilliset testitapaukset yhtenäiseksi, pakettina suoritettavaksi testisarjaksi (eng. test suite) (kuva 10).



KUVA 10. Testisarjan luominen Katalonilla. Lähde: Katalon Docs, n.d.

CYPRESS

Cypress on Brian Mannin kehittämä ja vuonna 2018 julkaisema avoimen lähdekoodin (eng. open-source) front-end–testiautomaatiotyökalu modernien web-sovelluksien automaattiseen testaamiseen (Cypress, n.d.). Cypress mahdollistaa muun muassa testien alustamisen ja kirjoittamisen sekä niiden automaattisen suorittamisen osana CI/CD-prosessia.

Cypressilla on useita vahvuuksia, joita ovat muun muassa sen helppokäyttöisyys sekä sen kattava dokumentaatio ja aktiivinen yhteisö, joista saa tarvittaessa tukea työkalun käyttöön. Lisäksi testien suorittaminen Cypressilla on nopeampaa moniin muihin testiautomaatiotyökaluihin verrattuna. Cypress on myös erittäin kattava työkalu, sillä se mahdollistaa kaikkien tyypillisimmin automatisoitavien testityyppien testien, kuten end-to-end–, komponentti-, integraatio- ja yksikkötestien, luomisen ja suorittamisen web-ympäristössä (Cypress Docs, n.d.).

Cypressin suuri suosio perustuu edellä mainittujen vahvuuksien lisäksi sen ainutlaatuisiin ominaisuuksiin, joiden ansiosta se erottuu muiden testiautomaatiotyökalujen joukosta. Yksi näistä on sen mahdollistama reaaliaikainen, visuaalinen vianmääritys selaimessa. Kehittäjät voivat siis visuaalisesti käydä suoritetun testin läpi komento kerrallaan, joka auttaa hahmottamaan, missä kohtaa testi epäonnistuu. Tämän lisäksi Cypress sisältää älykkäitä automaattisia odotuksia, joiden tarkoitus on eliminoida testiskriptiin manuaalisesti lisättävien odotuksien tarve. Odotuksien tarkoitus on varmistaa, että dynaamiset elementit ja toiminnot valmistuvat ennen seuraavan testikomennon suorittamista. (Di Sciullo, 2024.)

Cypress poikkeaa useimmista testiautomaatiotyökaluista siten, että sitä ajetaan useimmiten testattavan sovelluksen kanssa samaan aikaan, samassa ympäristössä, joka lisää testien suoritusnopeutta. Tämän ansiosta Cypressilla on myös pääsy kaikkiin ajettavan sovelluksen objekteihin, jolloin niitä voidaan hyödyntää testiskripteissä ilman ylimääräisiä säätöjä (Cypress Docs, n.d.). Tämä mahdollistaa hyvin tarkkojen ja mukautettujen testien tekemisen, mikä parantaa testien kattavuutta. Koska Cypress asennetaan tietokoneelle paikallisesti, se voi hyödyntää myös käyttöjärjestelmää testipalautetta antaessa. Cypress voi muun muassa

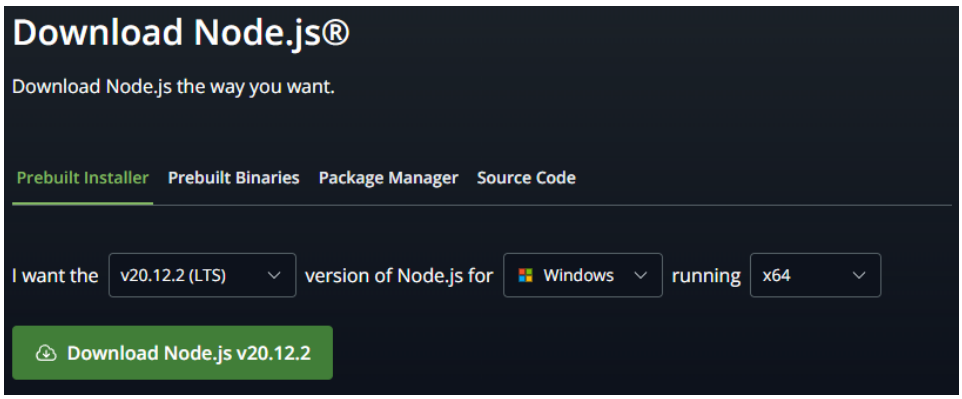
ottaa kuvakaappauksia ja tallentaa videoita testien suorittamisesta, jolloin voidaan helposti tarkistaa, mitä testin suorittamisessa on tapahtunut (Khan, 2021).

Katalonin tavoin myös Cypressin heikkouksiin lukeutuu sen vähäinen ohjelmointikielituki; Cypress nimittäin tukee ainoastaan JavaScriptiä testien kirjoittamisessa. Lisäksi Cypress ei omaa tallennus- ja toisto-ominaisuutta testiskriptien luomiseen, joten vähintään perustason ohjelmointiosaaminen JavaScriptillä on välttämätön. Cypress ei myöskään tue mobiilisovelluksien testaamista, eikä sellaisien testiskenaarioiden testaamista, jotka sisältävät useamman ikkunan, välilehden tai selaimen samanaikaista testaamista, joka myös osaltaan rajoittaa Cypressin testausmahdollisuuksia (Khan, 2021).

Kaikki Cypressin tärkeimmät ominaisuudet ovat saatavilla ilmaiseksi, mutta se tarjoaa käyttäjilleen myös maksullisen *Cypress Cloud*-version, jonka tarkoitus on lisätä Cypressin suorituskykyä sekä tehostaa ja laajentaa sen olemassa olevia ominaisuuksia. Se tarjoaa muun muassa paranneltuja testituloksien analysointiominaisuuksia, helpottaa testien jakamista tiimin kesken ja parantaa Cypressin skaalautuvuutta, jolloin testejä voidaan suorittaa samanaikaisesti useilla eri selaimilla tai laitteilla. *Cypress Cloud* sopii erityisesti laajoja testisarjoja testaaville erikokoisille tiimeille, jotka kaipaavat testien suorittamiseen ja analysointiin tehostusta.

5.3 Asennus ja käyttöönotto

Cypressin käyttöönottamiseksi tulee ensin asentaa Node.js, mikäli laitteella ei sitä vielä ole. Node.js saadaan asennettua sen viralliselta verkkosivustolta. Verkkosivun *Download*-välilehdeltä voidaan valita haluttu Node.js-versio sekä käyttöjärjestelmä, jolle se asennetaan (kuva 11). Asennuspaketin latauduttua ajetaan asennettu tiedosto ja seurataan asennusikkunan ohjeita.



KUVA 11. Node.js asennusvalikko.

Kun Node.js on asennettu, siirrytään itse Cypressin asennukseen. Cypressin asennusta varten avataan haluttu komentokehote. Aloitetaan asennus siirtymällä projektin kansioon syöttämällä seuraava komento:

```
> cd .\projektin-juurikansio
```

Seuraavaksi luodaan uusi kansio, jonka sisälle Cypress asennetaan. Tämä kansio tulee sisältämään myös Cypress-testiskriptit. Kansio luodaan komennolla

```
> mkdir testikansion-nimi
```

Tämän jälkeen siirrytään juuri luotuun kansioon suorittamalla seuraava komento:

```
> cd testikansion-nimi
```

Seuraavaksi luodaan ja alustetaan tyhjä JavaScript-projekti ajamalla komento

```
> npm init -y
```

Tämä komento luo oletusarvoisen *package.json*-tiedoston, jota käytetään projektin määrittelyyn. Mikäli tiedostolle halutaan itse syöttää halutut asetukset, voidaan argumentti *-y* jättää pois, jolloin interaktiivisia kehotteita ei ohiteta oletusarvoilla.

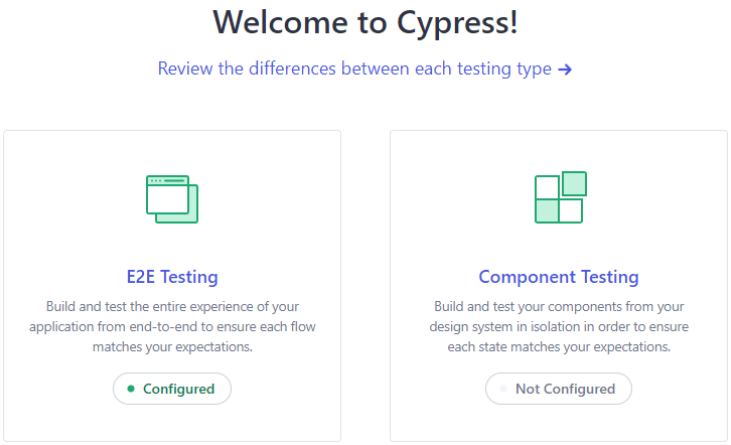
Kun package.json-tiedosto on luotu ja alustettu, asennetaan Cypress seuraavaa komentoa käyttäen:

```
> npm install cypress --save-dev
```

Asennuksen valmistuttua voidaan Cypress avata selaimessa suorittamalla komento

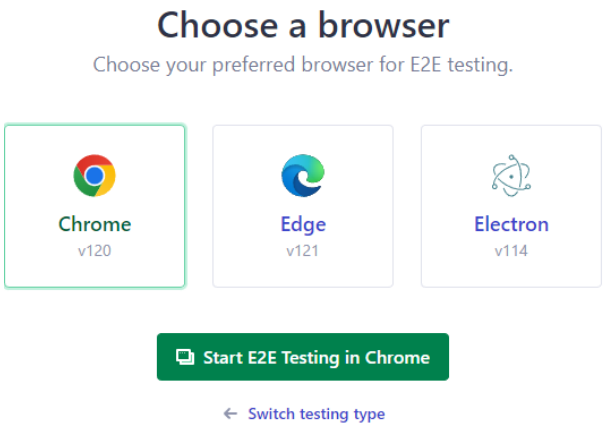
```
> npx cypress open
```

Tämä komento käynnistää Cypressin sekä avaa sen käyttöliittymän, josta voidaan valita haluttu testaustyyppi (kuva 12): *E2E Testing*, eli end-to-end–testaus, tai *Component Testing*, eli komponenttitestaus. Opinnäytetyön tapauksessa valittiin vaihtoehdoista end-to-end–testaus, jolla voidaan testata sovelluksen tai sivuston toimivuutta kokonaisuudessaan, joten nämä asennusohjeet eivät käsittele komponenttitestauksen käyttöönoton vaiheita. Mikäli komponenttitestaus kuitenkin vastaa käyttäjän tarpeita paremmin, tarjoaa käyttöliittymä selkeät ja yksinkertaiset ohjeet myös Cypressin valjastamiseen komponenttitestejä varten.



KUVA 12. Cypressin käyttöliittymän aloitusnäkymä.

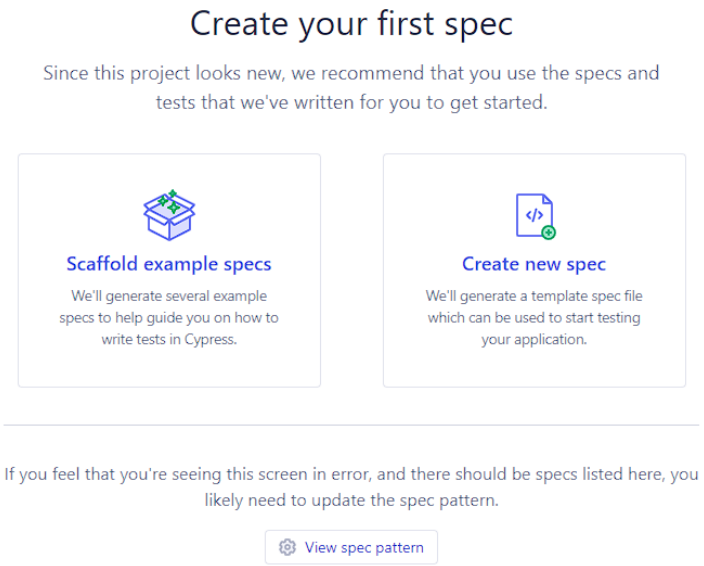
Cypress luo automaattisesti konfiguraatiotiedostot riippuen valitusta testaustyypistä, ja esittelee ne selityksineen käyttäjälle. Tämän jälkeen päästään valitsemaan testeissä käytettävä selain (kuva 13).



KUVA 13. Selaimen valinta Cypress testejä varten.

Cypress avaa valitussa selaimessa testausympäristön, jossa testejä voidaan luoda ja suorittaa sekä tarkastella niiden tuloksia. Ensimmäinen Cypress-testi voidaan luoda valitsemalla aloitusnäkymän *Create new spec*-vaihtoehto (kuva 14).

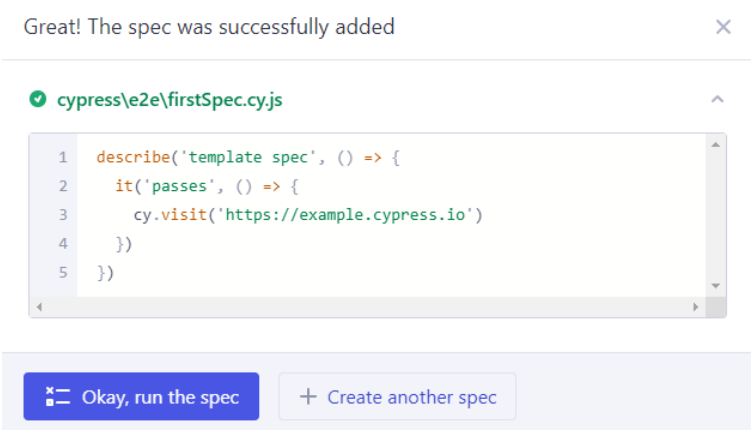
Cypress tarjoaa myös mahdollisuuden luoda esimerkkitestitiedostoja, joista käyttäjä voi tarvittaessa ottaa mallia testien kirjoittamiseen Cypressilla. Tällöin kuvan 14 näkymästä valitaan vaihtoehto *Scaffold example specs*.



KUVA 14. Vaihtoehdot uuden testitiedoston luomiseen.

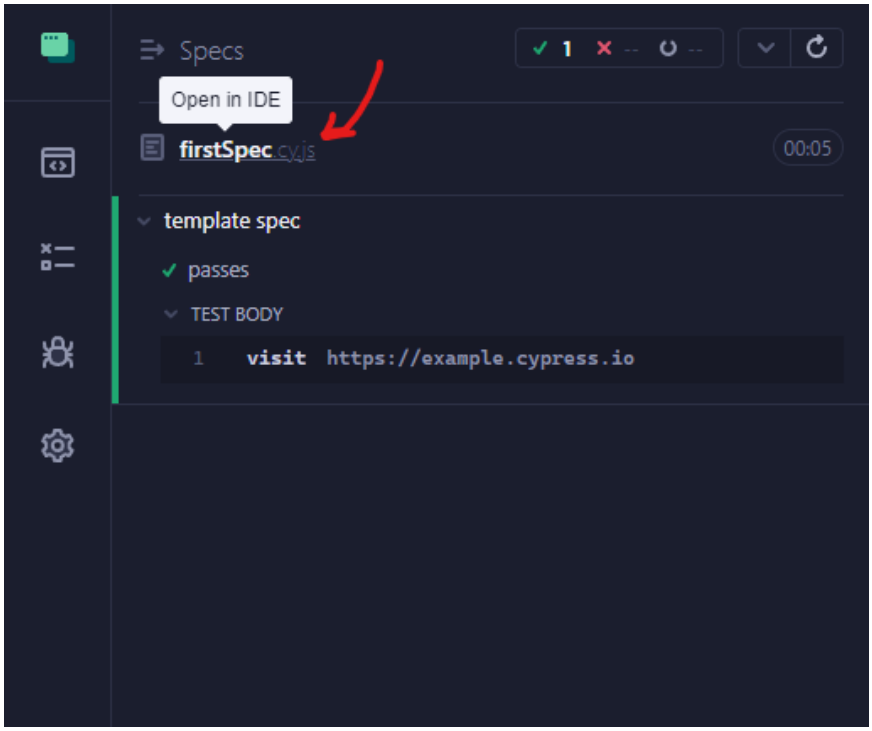
Create new spec-valinta avaa ikkunan testitiedoston polun määrittämiseksi. Testi nimetään polun loppuosan mukaan. Kun haluttu polku on määritetty, klikataan ikkunan *Create spec*-nappia. Tämän jälkeen Cypress ilmoittaa testin luomisen

valmistumisesta kuvan 15 mukaisella ikkunalla, joka voidaan sulkea *X*-painikkeella.



KUVA 15. Uuden testitiedoston luomisen valmistumisilmoitus.

Testitiedoston muokkaamista varten käyttäjällä tulee olla asennettuna jokin tekstieditori, kuten Visual Studio Code tai Notepad++. Testitiedosto voidaan avata tekstieditorissa joko manuaalisesti kansiosta, jossa testitiedosto sijaitsee, tai automaattisesti klikkaamalla testitiedoston nimeä Cypressin testinsuoritusnäkymässä (kuva 16).



KUVA 16. Cypressin testinsuoritusnäkymä.

5.4 Testien tekeminen Cypressilla

Kuten Katalonilla, myös Cypressilla suoritettiin helppokäyttöisyyden analysoimiseksi liitteen 1 testisuunnitelman mukaiset testit. Cypress tarjoaa testaamiseen monipuolisia komentoja, joita hyödyntämällä testisuunnitelman testiskriptit kirjoitettiin (kuva 17). Vaikka Cypressin komennot saattavat muistuttaa avainsanoja, ei Cypress kuitenkaan ole avainsanapohjainen työkalu. Cypress ei myöskään sisällä tallennus- ja toisto-ominaisuutta, jonka puuttuminen saattaa tehdä esimerkiksi verkkosivustolla navigointia sisältävien testien luomisen hitaammaksi.

Vaikka Cypressin testiskriptit luodaan kokonaan skriptaamalla, ei se kuitenkaan vaadi sen edistyneempää ohjelmointiosaamista kuin avainsanapohjainen testaaminenkaan, sillä Cypressin komennot ovat hyvin yksinkertaisia ja helppoja ymmärtää, ja niiden käyttöön löytyy kattavasti ohjeita Cypressin virallisesta dokumentaatiosta.

KUVA 17. Esimerkki Cypress-testiskriptistä.

Cypressin tarjoama reaaliaikainen testinsuoritusnäkymä (kuva 16) osoittautui testejä tehdessä hyödylliseksi työkaluksi, mikäli testin suoritus epäonnistui odottamattomasti. Sen avulla oli kohtuullisen helppoa löytää epäonnistumisen syy, oli se sitten testattavaan sivustoon liittyvä ongelma tai virhe testiskriptissä, sillä näkymä osoitti sen testivaiheen, jossa ongelma ilmeni.

Mikäli Cypress työkaluna ei ole ennestään tuttu, voi sen käyttö tuntua aluksi hitaalta, ja komentojen kirjoittaminen saattaa vaatia totuttelua. Komentoihin tutustumisen ja dokumentaatioon perehtymisen jälkeen Cypress-testien kirjoittaminen osoittautui kuitenkin helpommaksi ja nopeammaksi kuin monien muiden skriptauspohjaisten työkalujen käyttö.

6 ROBOT FRAMEWORK

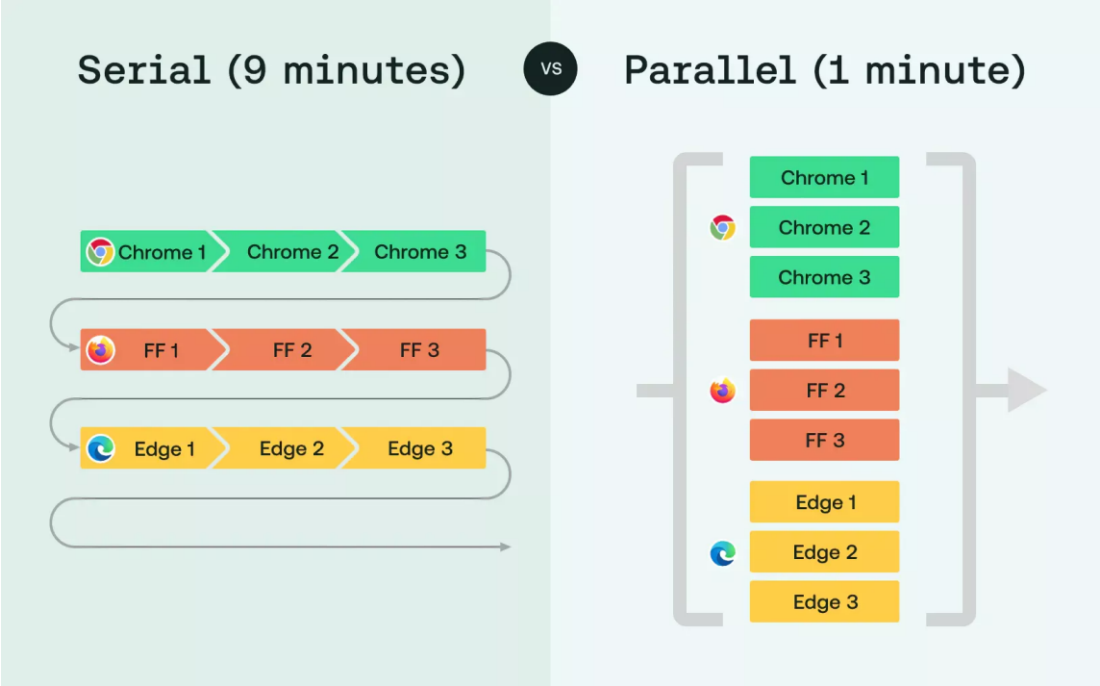
Robot Framework on Pekka Klärckin kehittämä ja ensimmäisen kerran vuonna 2005 julkaisema avainsanapohjainen automaatiokehys testaamisen ja robottiprosessien automatisointiin. Saavuttamansa suuren suosion vuoksi Robot Frameworkista kehitettiin myöhemmin uusi, avointa lähdekoodia käyttävä versio, joka julkaistiin vuonna 2008. (Eficode, 2016.)

Yksi Robot Frameworkin suurimmista ja erottuvimmista vahvuuksista on sen laajennettavuus, joka tekee siitä myös hyvin joustavan ja räätälöitävän työkalun. Käyttäjällä on käytössään valtava määrä eri kirjastoja, joilla Robot Frameworkia voidaan laajentaa ja muokata käyttäjän omien tarpeiden mukaiseksi. Robot Framework kannustaa yhteisöään luomaan myös omia kirjastoja, joita he halutessaan voivat jakaa myös muiden käyttäjien käyttöön. Tämä tekee Robot Frameworkista hyvin monipuolisen ja jatkuvasti kehittyvän työkalun, joka sopii lukuisiin eri testaustarpeisiin. (Johnson, 2023.)

Kirjastojen lisäksi Robot Framework mahdollistaa integroinnin muiden työkalujen ja järjestelmien kanssa sen ominaisuuksien ja toimintojen laajentamiseksi. Työkalun ydinominaisuuksia laajentaessa integraatioilla tai kirjastoilla on kuitenkin hyvä muistaa, että mahdollisissa ongelmatilanteissa tukea yhteisöstä tai dokumentaatiosta saattaa olla hankalampi löytää.

Robot Framework pohjautuu pääasiassa avainsanapohjaiseen testaamiseen, mutta se tukee myös skriptausta testien kirjoittamisessa. Se tarjoaa käyttäjälle listan valmiita avainsanoja, joita testiskriptien kirjoittamisessa voidaan hyödyntää. Valmiiden avainsanojen lisäksi Robot Frameworkilla voidaan tarvittaessa luoda ja alustaa myös omia avainsanoja. Avainsanapohjaiset testitapaukset ovat helppolukuisia ja ymmärrettäviä, jonka vuoksi niiden sujuvaan kirjoittamiseen useimmiten riittää perustason ohjelmointiosaaminen. Toisaalta ohjelmointiosaamisen puute ei välttämättä ole este testien kirjoittamiseen ja uusien avainsanojen luomiseen Robot Frameworkilla, sillä näihin löytyy paljon tukea ja ohjeita Robot Frameworkin virallisesta dokumentaatiosta sekä sen aktiivisesta käyttäjäyhteisöstä. (Johnson, 2023.)

Robot Framework mahdollistaa testien suorittamisen samanaikaisesti eri laitteilla, ympäristöillä ja selaimilla, joka nopeuttaa ja tehostaa testien suorittamista huomattavasti, kuitenkaan tinkimättä testaamisen laadusta. Tätä tapaa testata kutsutaan nimellä *rinnakkainen testaus* (eng. parallel testing) (kuva 18).



KUVA 18. Testaaminen sarjassa vs. rinnakkain. Lähde: Testsigma, n.d.

Edellä mainittujen vahvuuksien lisäksi Robot Framework tarjoaa testituloksien analysoinnin helpottamiseksi selkeät ja informatiiviset virheviestit testin epäonnistuessa, joka helpottaa ja nopeuttaa mahdollisten vikojen paikannusta ja korjaamista.

Robot Frameworkin haasteisiin luetaan usein sen jyrkkä oppimiskäyrä. Erityisesti sellaisille käyttäjille, jotka eivät omaa kokemusta Python-ohjelmointikielestä tai testiautomaatiosta, voi Robot Framework tuntua alkuun hankalalta käyttää. Toisin kuin esimerkiksi Katalon tai Selenium IDE, se ei tarjoa valmista listaa käytettävissä olevista avainsanoista, vaan ne täytyy etsiä itse esimerkiksi Robot Frameworkin virallisesta dokumentaatiosta.

Muita Robot Frameworkin heikkouksia ovat muun muassa sen suorituskykyongelmat laajoja testisarjoja suorittaessa sekä sen vahva pohjautuminen avainsanojen käyttöön, joka saattaa hankaloittaa monimutkaisempien testien luomista.

6.1 Asennus ja käyttöönotto

Koska Robot Framework on Python-pohjainen, tulee sen käyttämiseksi laitteella olla asennettuna Python. Haluttu Python-versio halutulle käyttöjärjestelmälle saadaan helposti asennettua sen virallisen verkkosivuston *Downloads*-välilehdeltä.

Asennuspaketin latauksen valmistuttua ajetaan ladattu tiedosto, joka avaa Pythonin asennusohjelman. Pythonia asennettaessa suositellaan lisäämään Python polkuun (eng. path), jotta sitä työkaluineen voidaan käyttää komentorivillä riippumatta siitä, missä kansiossa käyttäjä sijaitsee komentoja suorittaessa. Tämä onnistuu valitsemalla *Add python.exe to PATH* -valintaruutu asennusohjelman aloitusnäkymässä (kuva 19).

KUVA 19. Pythonin asennusohjelma.

Kun Pythonin asennus on valmis, avataan komentokehote ja varmistetaan, että Python on asennettu onnistuneesti. Mikäli kaikki on kunnossa, tulisi komentokehotteen palauttaa käyttäjälle Pythonin versio (esim. *Python 3.12.3)* seuraavan komennon syötettyä:

```
> python --version
```

Robot Frameworkin asennuksessa hyödynnetään *pipiä*, Pythonin yleisintä pakетinhallintatyökalua. Ennen Robot Frameworkin asennusta tulee siis varmistaa, että *pip* on käytettävissä syöttämällä seuraava komento:

```
> pip --version
```

Mikäli *pip* on käytettävissä, palauttaa komentokehote vastauksena käytössä olevan *pip*-version ja sen sijainnin laitteella. Mikäli *pip* taas ei syystä tai toisesta ole asentunut Pythonin mukana, löytyy ohjeet sen asentamiseen Pythonin virallisesta dokumentaatiosta.

Kun edellä mainitut asiat ovat kunnossa, voidaan siirtyä itse Robot Frameworkin asennukseen. Robot Frameworkin uusin versio saadaan asennettua seuraavalla komennolla:

```
> pip install robotframework
```

Kuten Cypress, myös Robot Framework vaatii testiskriptien kirjoittamiseksi jonkin ulkoisen tekstieditorin, kuten Visual Studio Coden.

Tarvittaessa Robot Frameworkille voidaan helposti asentaa erilaisia kirjastoja esimerkiksi mobiili- tai verkkosovelluksien testaamiseksi. Esimerkiksi seuraavalla komennolla Robot Frameworkille saadaan asennettua *SeleniumLibrary*-kirjasto, joka laajentaa Robot Frameworkin selaintukea sekä tarjoaa monipuolisia ja kattavia toimintoja verkkosivustojen testaamiseen:

```
> pip install robotframework-seleniumlibrary
```

6.2 Testien tekeminen Robot Frameworkilla

Kuten aiemmilla työkaluilla, myös Robot Frameworkilla suoritettiin testisuunnitelman (liite 1) mukaiset testit. Testiskriptien kirjoittamisessa hyödynnettiin sekä Robot Frameworkin valmiita että kirjoittajien itsetekemiä avainsanoja (kuva 20). Robot Frameworkin omien avainsanojen käyttö vaati pientä perehtymistä dokumentaatioon, sillä se ei tarjoa käyttäjälle käytettyyn IDEen (esim. Visual Studio Code)

valmista listaa olemassa olevista avainsanoista, kuten esimerkiksi Katalon tai Selenium IDE tarjoavat.

```
*** Settings ***
Documentation     Simple example of testing a login page using Robot Framework
Library           SeleniumLibrary


*** Variables ***
${BROWSER}        Chrome
${URL}            https://automationexercise.com/



*** Keywords ***
Handle Popup If Visible
    ${consent_button_visible}    Run Keyword And Return Status    Wait Until Element Is Visible    xpath://button[@aria-label='Consent']    1s
    Run Keyword If    '${consent_button_visible}' == 'True'    Click Element    xpath://button[@aria-label='Consent']



*** Test Cases ***


Testitapaus 1: Rekisteröi käyttäjä
```

KUVA 20. Esimerkki Robot Framework -testitiedoston asetuksista, muuttujista ja omista avainsanoista.

Dokumentaation avulla testien kirjoittaminen osoittautui lopulta kohtuullisen helpoksi Robot Frameworkin avainsanojen ja syntaksin selkeyden ansiosta. Kuva 21 esittää esimerkin Robot Frameworkin avainsanojen käytöstä.

```
156  Testitapaus 7: Varmista tuotteen määrä ostoskorissa
157      #1. Käynnistä selain 2. Siirry osoitteeseen 'http://automationexercise.com'
158      Open Browser    ${URL}    ${BROWSER}
159      Handle Popup If Visible
160
161      #3. Varmista, että etusivu näkyy onnistuneesti
162      Element Text Should Be    xpath: /html/body/section[2]/div/div/div[2]/div[1]/h2    FEATURES ITEMS
163
164      #4. Klikkaa 'View Product' minkä tahansa tuotteen kohdalla etusivulla
165      Click Element   xpath: /html/body/section[2]/div/div/div[2]/div/div[2]/div/div[2]/ul/li/a
166      Handle Popup If Visible
167
168      #5. Varmista, että tuotteen yksityiskohdat avautuvat
169      Element Should Be Visible   xpath: /html/body/section/div/div/div[2]/div[2]/div[2]/div
170
171      #5. Varmista, että tuotteen yksityiskohdat avautuvat
172      Element Text Should Be  xpath: /html/body/section/div/div/div[2]/div[2]/div[2]/div/p[4]    Brand: Polo
173
174      #6. Aseta määräksi 4
175      input Text    xpath: /html/body/section/div/div/div[2]/div[2]/div[2]/div/span/input[1]    4    True
176
177      #7. Klikkaa 'Add to cart' -painiketta
178      Click Element   xpath: /html/body/section/div/div/div[2]/div[2]/div[2]/div/span/button
179      Sleep   2s
180
181      #8. Klikkaa 'View Cart' -painiketta
182      Click Element   xpath: /html/body/section/div/div/div[2]/div[1]/div/div/div[2]/p[2]/a
183
184      #9. Varmista, että tuote näkyy ostoskorisivulla tarkalleen oikealla määrällä
185      Element Text Should Be  xpath: /html/body/section/div/div[2]/table/tbody/tr/td[4]/button    4
186      Close Browser
187
```

KUVA 21. Esimerkki Robot Framework -testiskriptistä.

Vaikka avainsanat ovatkin lopulta helppoja käyttää, tulee pieni miinus Robot Frameworkin helppokäyttöisyyteen kuitenkin tallennus- ja toistotoiminnon puuttumisesta.

7 SELENIUM IDE

Selenium IDE on avoimen lähdekoodin testiautomaatiolaajennus Firefox-, Chrome- ja Edge-selaimille, joka mahdollistaa testien alustamisen, luomisen, suorittamisen sekä testituloksien analysoinnin (Fortner, 2023).

Selenium IDE oli alun perin Shinya Kasatanin kehittämä vuonna 2006 julkaistu testiautomaatiotyökalu Firefox-selaimelle. Alkuperäisellä Selenium IDEllä oli useita rajoituksia, kuten sen vähäinen selaintuki, hankaluus luoda monimutkaisia testejä ja luotujen testien epäluotettavuus (Fortner, 2023), joiden vuoksi työkalun suosio vuosien varrella väheni. Alkuperäinen Selenium IDE lakkasi lopulta kokonaan toimimasta vuonna 2017 julkaistun *55.0* Firefox-version myötä (Battat, 2021). Vuonna 2018 Selenium IDEstä julkaistiin nykyinen, parempi versio.

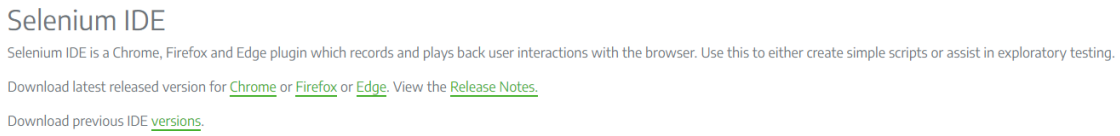
Selenium IDEn suurimmat vahvuudet ovat sen helppokäyttöisyys ja aloittelijaystävällisyys. Selenium IDE mahdollistaa testien luomisen Katalonin tavoin sekä tallennus- ja toistotoimintoa kuin avainsanapohjaistakin menetelmää käyttäen. Selenium IDE tarjoaa avainsanapohjaiseen testaamiseen käyttöliittymässään valmiin listan olemassa olevista avainsanoista.

Selenium IDEn kanssa testien luominen on helppoa ja nopeaa, mutta testatessa laajempia ohjelmistoja, jotka vaativat monimutkaisempia testejä, saattavat Selenium IDEn tarjoamat ominaisuudet jäädä vajaiksi. Tällaisten testien luomisessa esimerkiksi Seleniumin tarjoan WebDriver-työkalu saattaa olla parempi vaihtoehto. Tarvittaessa Selenium IDEllä luotu testi voidaan myös viedä (eng. export) haluttuun tekstieditoriin, jolloin se käännetään samalla halutulle ohjelmointikielelle ja sitä päästään muokkaamaan WebDriver-työkalulla skriptauspohjaisesti.

7.1 Asennus ja käyttöönotto

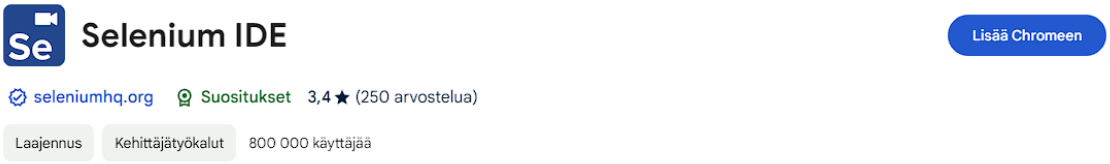
Koska Selenium IDE on selainlaajennus, on sen asennus ja käyttöönotto hyvin nopea ja suoraviivainen prosessi. Linkit Selenium IDEn lisäämiseen eri selaimiin

löytyvät Seleniumin virallisen verkkosivuston *Download*-välilehdeltä. Etsitään välilehden listasta Selenium IDE, jonka jälkeen klikataan sen selaimen linkkiä, johon laajennus lisätään (kuva 22).



KUVA 22. Linkit Selenium IDEn käyttöönottoon eri selaimille.

Linkki ohjaa käyttäjän sivulle, josta Selenium IDE päästään lataamaan kyseiselle selaimelle (kuva 23).



KUVA 23. Selenium IDEn lisäys Chrome-selaimeen.

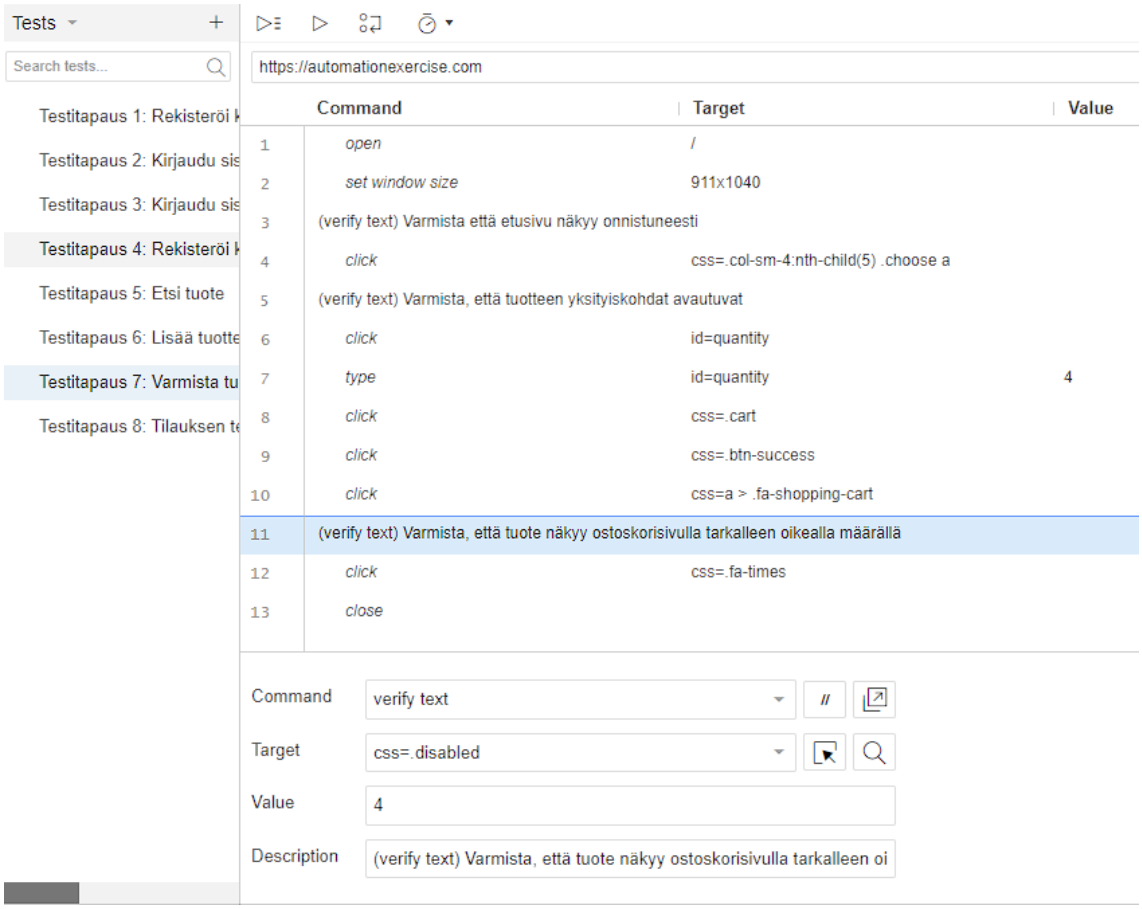
Kun Selenium IDE on onnistuneesti lisätty haluttuun selaimeen, on se heti valmis käytettäväksi.

7.2 Testien tekeminen Selenium IDEllä

Kuten aiemmin esitellyillä työkaluilla, myös Selenium IDEllä suoritettiin testisuunnitelman (liite 1) mukaiset testit. Testien luomisessa hyödynnettiin sekä tallennus- ja toistotoimintoa että avainsanapohjaista testaamista (kuva 24). Selenium IDEllä testit oli helppo luoda, ja sen ominaisuudet riittivät mainiosti yksinkertaisten web-testien tekemiseen. Testien luomisessa hyödynnettiin Selenium IDEn tarjoamaa *Select target in page* -ominaisuutta, jolla nimensä mukaisesti pystytään poimimaan testiobjekti sivulta.

Selenium IDEn käyttö testaamisessa ei välttämättä vaadi lainkaan ohjelmointiosaamista, mutta siitä saattaa kuitenkin olla hyötyä avainsanapohjaisten testien tekemisessä.

Cypressin tavoin Selenium IDE hyödyntää testien suorittamisessa automaattisia odotuksia, joka pyrkii vähentämään turhia virheitä odottamalla sivun latautumisen valmistumista.



KUVA 24. Esimerkki Selenium IDEllä luodusta testistä.

Lopuksi tehtiin myös kokeilu testiskriptin viemisestä JavaScript Mocha -koodiksi, jolloin sitä voitiin muokata tarkemmin käyttämällä Selenium WebDriveria (kuva 25).

```javascript
// Generated by Selenium IDE
const { Builder, By, Key, until } = require('selenium-webdriver')
const assert = require('assert')

describe('Testitapaus 7: Varmista tuotteen määrä ostoskorissa', function() {
  this.timeout(30000)
  let driver
  let vars
  beforeEach(async function() {
    driver = await new Builder().forBrowser('chrome').build()
    vars = {}
  })
  afterEach(async function() {
    await driver.quit();
  })
  it('Testitapaus 7: Varmista tuotteen määrä ostoskorissa', async function() {
    // Test name: Testitapaus 7: Varmista tuotteen määrä ostoskorissa
    // Step # | name | target | value
    // 1 | open | / |
    await driver.get("https://automationexercise.com/")
    // 2 | setWindowSize | 911x1040 |
    await driver.manage().window().setRect({ width: 911, height: 1040 })
    // 3 | verifyText | css=.features_items > .title | FEATURES ITEMS
    // (verify text) Varmista että etusivu näkyy onnistuneesti
    assert(await driver.findElement(By.css(".features_items > .title")).getText() == "FEATURES ITEMS")
    // 4 | click | css=.col-sm-4:nth-child(5) .choose a |
    await driver.findElement(By.css(".col-sm-4:nth-child(5) .choose a")).click()
    // 5 | verifyElementPresent | css=.product-information |
    // (verify text) Varmista, että tuotteen yksityiskohdat avautuvat
    {
      const elements = await driver.findElements(By.css(".product-information"))
      assert(elements.length)
    }
    // 6 | click | id=quantity |
    await driver.findElement(By.id("quantity")).click()
    // 7 | type | id=quantity | 4
    await driver.findElement(By.id("quantity")).sendKeys("4")
    // 8 | click | css=.cart |
    await driver.findElement(By.css(".cart")).click()
    // 9 | click | css=.btn-success |
    await driver.findElement(By.css(".btn-success")).click()
    // 10 | click | css=a > .fa-shopping-cart |
    await driver.findElement(By.css("a > .fa-shopping-cart")).click()
    // 11 | verifyText | css=.disabled | 4
    // (verify text) Varmista, että tuote näkyy ostoskorisivulla tarkalleen oikealla määrällä
    assert(await driver.findElement(By.css(".disabled")).getText() == "4")
    // 12 | click | css=.fa-times |
    await driver.findElement(By.css(".fa-times")).click()
    // 13 | close |  |
    await driver.close()
  })
})

```

KUVA 25. Selenium IDE -testitapaus muutettuna JavaScript-kielelle.

8 VERTAILU

Katalonin, Cypressin, Robot Frameworkin ja Selenium IDEn välillä suoritettiin vertailu, joka sisältää pohdintaa muun muassa testityökalujen dokumentaatioiden laajuudesta ja helppolukuisuudesta, hinnoittelusta, asennuksen ja käyttöönoton helppoudesta, ominaisuuksista sekä työkalujen tarjoaman selain- ja ohjelmointikielituen laajuudesta. Vertailun tuloksien havainnollistamisessa hyödynnettiin taulukoita, jotka toimivat samalla yhteenvetona vertailun tuloksista.

8.1 Dokumentaatio

Virallinen dokumentaatio on yksi testityökalujen tärkeimmistä käyttöohjeiden ja -tuen lähteistä. On tärkeää, että dokumentaatio on selkeää ja ymmärrettävää myös käyttäjille, joille työkalu ei ole ennestään tuttu.

Vertailuun valituista työkaluista selkeimmät ja kattavimmat dokumentaatiot löytyivät Katalonilta ja Cypressilta. Näiden dokumentaatioissa on käytetty runsaasti kuvakaappauksia, joiden lisäksi Cypress on hyödyntänyt dokumentaatiossaan myös videoita. Kummankin työkalun dokumentaatio sisältää selkeän hakemiston sekä hakuominaisuuden, jonka avulla dokumentaatiosta on helppo löytää etsimänsä. Tehtyjä hakuja voidaan myös lisätä suosikeiksi, jolloin ne tallentuvat käyttäjälle myöhempää käyttöä varten.

Myös Robot Frameworkin dokumentaatio on kattava, ja Katalonin ja Cypressin tavoin se sisältää selkeän hakemiston. Robot Frameworkin dokumentaatio sisältää edellä mainittujen työkalujen dokumentaatioihin verrattuna hyvin paljon tekstiä ilman kuvakaappauksia, joka saattaa tehdä siitä melko raskasta luettavaa. Robot Frameworkin dokumentaatio sisältää paljon koodiesimerkkejä ja testipohjia, jotka helpottavat testien luomista.

Selenium IDEn tarjoama dokumentaatio jää määrällisesti sisällöltään selkeästi suppeammaksi muiden työkalujen dokumentaatioon verrattuna. Saatavilla oleva dokumentaatio on kuitenkin helppolukuista ja selkeää, jonka lisäksi se sisältää muun muassa havainnollistavia kuvakaappauksia.

Lisäksi Robot Frameworkin ja Selenium IDEn dokumentaatiot eivät sisällä erillistä hakutoimintoa, joka tekee näiden dokumentaatioiden selaamisesta jonkin verran hitaampaa.

8.2 Hinnoittelu

Kaikilla vertailuun valituilla työkaluilla on rajaton aika käyttää ilmaista versiota, mutta Katalonilla ja Cypressillä osa ominaisuuksista vaatii maksullisen version, jota kumpikin työkalu markkinoi pääasiassa suuremmille tiimeille, laajempiin ohjelmistoprojekteihin.

TAULUKKO 1. Hinnoittelu.

Testityökalu	Ilmainen	Maksullinen
KATALON	X	X
CYPRESS	X	X
ROBOT FRAMEWORK	X	
SELENIUM IDE	X	

8.3 Ominaisuudet ja helppokäyttöisyys

Sekä Katalon että Cypress tarjoavat valmiit esimerkkisovellukset, joiden avulla niiden käyttöä voidaan harjoitella, jonka lisäksi Katalon Studio sisältää myös kattavia läpikäyntejä sen eri ominaisuuksien käytöstä. Myös Robot Framework tarjoaa dokumentaatiossaan linkkejä muutamiin demoihin, mutta nämä eivät kuitenkaan ole yhtä kattavia kuin edellä mainittujen työkalujen tarjoamat läpikäynnit ja esimerkit. Selenium IDEllä edellä mainittuja mahdollisuuksia käytön harjoitteluun ei ole, joten sen käyttö tulee opetella itsenäisesti esimerkiksi dokumentaation pohjalta.

Taulukko 2 sisältää tiivistettynä työkalujen tukemat menetelmät testien luomiseen. Osa testityökaluista, kuten Katalon, tukee useita eri menetelmiä testien luomiseen, jonka ansiosta se sopii monenlaisten testien tekemiseen niin aloittele-

valle kuin kokeneemmallekin testaajalle, kun taas esimerkiksi Cypress tukee ainoastaan skriptausmenetelmää, jonka vuoksi sen käyttö vaatii jonkin verran ohjelmointiosaamista.

Selenium IDE taas tukee ainoastaan tallennus- ja toistotoiminnon sekä avainsanapohjaisen testauksen käyttöä testien luomisessa, jolloin monimutkaisempien testien luominen saattaa olla haastavaa. Selenium IDEllä luotuja testiskriptejä voi kuitenkin tarvittaessa viedä muokattavaksi Seleniumin WebDriver-työkalulla, jolloin myös monimutkaisempien testien luominen on mahdollista.

Robot Framework mahdollistaa skriptausta ja avainsanapohjaista menetelmää tukevana työkaluna monimutkaistenkin testien luomisen. Samalla tämä kuitenkin tarkoittaa sitä, että sen käyttö vaatii jonkin verran ohjelmointiosaamista.

TAULUKKO 2. Tuetut menetelmät testien luomiseksi.

Testityökalu	Tallennus ja toistotyökalut	Skriptaus	Avainsanapohjainen testaus
KATALON	X	X	X
CYPRESS		X	
ROBOT FRAMEWORK		X	X
SELENIUM IDE	X	Mahdollista Selenium WebDriverillä	X

Katalonilla ja Selenium IDEllä voidaan helposti poimia verkkosivustolta haluttuja testiobjekteja, joka voi nopeuttaa testien tekemistä. Lisäksi kumpikin työkalu tarjoaa avainsanapohjaiseen testaamiseen käyttöliittymissään pudotusvalikot, jotka sisältävät valmiin listan olemassa olevista avainsanoista. Näiden ominaisuuksien ansiosta ne saattavat tuntua Cypressia ja Robot Frameworkia helpommilta käyttää.

8.4 Asennus ja käyttöönotto

Käyttöönoton ja asennuksen vaikeuteen (taulukko 3) vaikuttaa esimerkiksi komentorivityöskentelyn tarve sekä dokumentaation asennusohjeiden selkeys. Katalonin asennus hoituu helposti lataamalla ja ajamalla sen asennuspaketti, jonka jälkeen asennus suoriutuu ilman tarvetta käyttäjän toimille.

Cypressin asennuksessa vaaditaan hieman komentorivityöskentelyä, jonka lisäksi tulee asentaa Node.js. Robot Frameworkin asennuksessa on Cypressin kanssa yhteiset haasteet, sillä myös Robot Frameworkin asennus vaatii sekä komentorivityöskentelyä että ulkoisien ohjelmien (Python) asentamisen. Selenium IDEn asennus taas onnistuu helposti valitsemalla sen lisäosa halutun selaimen lisäosavalikoimasta.

TAULUKKO 3. Asennuksen vaikeustaso.

Testityökalu	Asennuksen vaikeustaso
KATALON	Helppo (ei komentorivityöskentelyä tai ulkopuolisten sovellusten asentamista)
CYPRESS	Keskivaikea (perustason komentorivityöskentelyä ja ulkopuolisten sovellusten asentamista)
ROBOT FRAMEWORK	Keskivaikea (perustason komentorivityöskentelyä ja ulkopuolisten sovellusten asentamista)
SELENIUM IDE	Helppo (ei komentorivityöskentelyä tai ulkopuolisten sovellusten asentamista)

8.5 Tuetut ohjelmointikielet ja selaimet

Taulukossa 4 on esitettynä kunkin työkalun tukemat ohjelmointikielet. Katalonin ja Cypressin kielituki on melko suppeaa, sillä Cypress tukee testiskriptien kirjoittamisessa ainoastaan JavaScriptiä ja Katalon ainoastaan Javaa ja Groovyä.

Robot Framework omaa vertailun työkaluista laajimman ohjelmointikielituen testiskriptien kirjoittamiseen. Se mahdollistaa testien kirjoittamisen lukuisilla eri ohjelmointikielillä, muun muassa Pythonilla, Javalla, JavaScriptillä ja C#:lla.

Taulukon sisällöstä tulee huomioida, että Selenium IDEn kohdalla ohjelmointikielituella tarkoitetaan tässä tapauksessa niitä kieliä, joille sillä luodut testiskriptit voidaan kääntää viedessä ne muokattavaksi Selenium WebDriverillä. Selenium IDE ei itsessään mahdollista skriptien kirjoittamista koodina, joten se ei varsinaisesti tue itse mitään ohjelmointikieliä testiskriptien luomiseen.

TAULUKKO 4. Tuetut ohjelmointikielet.

Testityökalu	Tuetut ohjelmointikielet
KATALON	Groovy, Java
CYPRESS	JavaScript
ROBOT FRAMEWORK	mm. Python, Java, C#, JavaScript, Ruby
SELENIUM IDE	mm. Java, Python, Ruby, C#

Taulukossa 5 on esitettynä selaimet, joilla työkalut mahdollistavat web-testien suorittamisen. Katalon tarjoaa vertailluista työkaluista laajimman valikoiman tuettuja selaimia, mutta myös Selenium IDE ja Cypress tukevat yleisimpiä verkkoselaimia (Firefox, Edge ja Chrome).

Robot Frameworkillä ei varsinaisesti tue mitään selaimia natiivisti, vaan se vaatii selaintuen saamiseksi erillisen kirjaston. Yleisimmin tähän tarkoitukseen käytetty kirjasto on SeleniumLibrary, jonka avulla Robot Framework saa tuen esimerkiksi Chrome-, Firefox-, Edge- ja Safari-selaimille.

TAULUKKO 5. Tuetut selaimet.

Testityökalu	Tuetut selaimet
KATALON	Internet Explorer, Chrome, Edge, Firefox, Opera, Safari
CYPRESS	Chrome, Edge, Firefox
ROBOT FRAMEWORK	Selaintuki vaatii erillisen kirjaston (esim. SeleniumLibrary)
SELENIUM IDE	Chrome, Edge, Firefox

9 POHDINTA

Opinnäytetyön päätavoitteena oli tutustua testiautomaatioon sekä vertailla neljää eri web-automaatiotestauksessa yleisesti käytettävää testiautomaatiotyökalua: Katalonia, Cypressia, Robot Frameworkia ja Selenium IDEä. Tutkimuksessa selvitettiin vertailtavien työkalujen ominaisuuksia, dokumentaatiota ja käytettävyyttä web-ympäristössä, jonka lisäksi esiteltiin vaihe vaiheelta jokaisen työkalun asennusprosessi.

Testiautomaation tutkimuksessa kävi ilmi, että maailman kiihtyvän teknologisoitumisen vuoksi testiautomaation suosio on jatkuvassa kasvussa kaikenkokoisissa ohjelmistoprojekteissa, sillä ohjelmistojen laajentuessa myös ohjelmistotestaamisen tarpeet kasvavat. Testiautomaatio tarjoaa monia eri työkaluja muun muassa testien alustamisen, luomisen ja suorittamisen nopeuttamiseen ja tehostamiseen, ja se voi oikein käytettynä vähentää testauskustannuksia hyvinkin merkittävästi.

Työkalujen vertailua varten laadittiin testisuunnitelma (liite 1), jonka mukaiset testit jokaisella työkalulla suoritettiin. Testaamisessa hyödynnettiin valmista Automation Exercise -sivustoa. Testisuunnitelman testitapaukset pidettiin perustasoisina ja yksinkertaisina, sillä niiden tarkoituksena oli ainoastaan auttaa perehtymään työkaluihin paremmin ja saamaan selkeämpi käsitys niiden käyttöönotosta, perusominaisuuksista ja helppokäyttöisyydestä. Tämän lähestymistavan ansiosta saatiin omakohtaista kokemusta työkalujen asennuksesta, käyttöönotosta, testien alustamisesta ja testitapausten luomisesta sekä testien ajamisesta ja niiden tuloksien analysoimisesta onnistuneissa ja epäonnistuneissa testitapauksissa, mikä helpotti työkalujen ominaisuuksien tutkimista ja vertailua.

Työkalujen vertailun tuloksena todettiin, että jokaisella työkalulla on omat vahvuutensa ja heikkoutensa, ja näin ollen niillä kaikilla on paikkansa web-automaatiotestauksessa riippuen käyttäjän osaamistasosta ja käyttökohteesta. Esimerkiksi helppokäyttöisyyden ja testitarpeen puolesta Selenium IDE sopii erityisesti aloittelevalle web-testiautomaation käyttäjälle, jolla ei ole vankkaa ohjelmointiosaamista tai tarvetta monimutkaisien testien suorittamiseen, kun taas Robot

Framework ja Cypress ovat hyviä vaihtoehtoja kokeneemmalle testaajalle, joka omaa kohtalaista ohjelmointiosaamista ja tarvitsee tehokkaan työkalun, joka mahdollistaa myös monimutkaisempien testien kirjoittamisen ja suorittamisen. Katalon puolestaan sopii ominaisuuksiensa puolesta sekä aloittelevalle että kokeneemmalle web-automaatiotestaajalle, sillä se sopii kaikenkokoisten projektien testaamiseen ja mahdollistaa testien luomisen monin eri tavoin, joista osa ei vaadi lainkaan ohjelmointiosaamista.

Työkalujen helppokäyttöisyyttä kokeillessa huomattiin, että testien luominen ja suorittaminen oli työkalu työkalulta helpompaa ja nopeampaa, vaikka uusi työkalu olisikin osaamistasoltaan aiempia vaativampi käyttää. Tästä voisi päätellä, että kaikenlaisesta aiemmasta testiautomaatiokokemuksesta on hyötyä uuden työkalun opettelussa, vaikka työkalut keskenään sisältäisivätkin eri ominaisuuksia ja menetelmiä.

Työkalujen asennusta ja käyttöönottoa tarkasteltaessa tultiin siihen tulokseen, että se oli helpointa Katalonilla ja Selenium IDEllä. Katalonin asennus tapahtui yksinkertaisesti ajamalla sen asennusohjelma, jonka jälkeen käyttöönotto onnistui sen interaktiivisten kehotteiden avulla. Selenium IDE on selainlaajennus, joka teki sen asentamisesta erityisen helppoa ja nopeaa. Lisäksi Selenium IDEn käyttö on hyvin intuitiivista sen ominaisuuksien ja käyttöliittymän yksinkertaisuuden ansiosta.

Työkalujen asennus oli haastavinta Cypressilla ja Robot Frameworkilla, sillä niiden asennus vaatii käyttäjältä ulkoisten ohjelmien asentamista sekä jonkin verran komentorivityöskentelyä, joka tarkoittaa, että käyttäjän tulee tietää tai osata etsiä asennuksessa käytettävät komennot. Myös käyttöönotto näillä kahdella oli haastavampaa, sillä ne eivät tarjoa käyttöönoton yhteydessä ohjeita tai valmista listaa olemassa olevista komennoista tai avainsanoista. Mikäli käyttäjä ei perehdy näihin etukäteen, saattaa testien kirjoittaminen aluksi tuntua hitaalta ja hankalalta.

Testityökalujen vertailua voisi laajentaa muun muassa sisällyttämällä siihen useampia testityökaluja sekä vertailemalla ja kokeilemalla työkalujen eri ominaisuuksia laajemmin, esimerkiksi testaamalla niiden suorituskykyä (esim. testien suori-

tusnopeus) ja testien luotettavuutta tarkemmin sekä testaamalla työkalujen skaalautuvuutta, esimerkiksi eri selaimilla ja laitteilla. Lisäksi testisuunnitelmaan voisi sisällyttää myös monimutkaisempia, vaativampia testejä, jonka lisäksi voisi tutkia tarkemmin työkalujen raportointi- ja analysointimahdollisuuksia.

Vertailtavien työkalujen testauskohteena toimi valmis verkkosivu, ja vaikka testit saatiinkin onnistuneesti suoritettua jokaista työkalua käyttäen, ei tämän vuoksi esimerkiksi Cypress päässyt testeissä täyteen loistoonsa, sillä se on suunniteltu nimenomaan käytettäväksi oman sovelluksen kehityksen rinnalla. Reilumman vertailutuloksen saamiseksi Cypressin voisi vaihtaa johonkin toiseen, tarkoituksenmukaisempaan työkaluun, tai vastaavasti testaamisen kohteena voisi toimia jokin itsekehitetty web-sovellus.

LÄHTEET

Jose, B. 2021. Test Automation: A manager's guide. O'Reilly Online Learning. Viitattu 3.3.2024.

VALA. 2022. Mitä on ohjelmistotestaus ja mitä hyötyä siitä on? Verkkosivu. Viitattu 3.3.2024. https://www.valagroup.com/fi/blogi/mita-on-ohjelmistotestaus-ja-mita-hyotya-siita-on/

Krysik, A. 2023. 21 Types of Software Testing Every Engineer Should Be Using for Better Results. Verkkosivu. Viitattu 3.3.2024. https://stratoflow.com/types-of-software-testing/

Leloudas, P. 2023. Introduction to Software Testing: A Practical Guide to Testing, Design, Automation, and Execution. O'Reilly Online Learning. Viitattu 18.3.2024.

Pittet, S. n.d. The different types of software testing. Atlassian. Verkkosivu. Viitattu 10.3.2024. https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

Desikan, S. & Ramesh, G. 2007. Software Testing: Principles and Practices. O'Reilly Online Learning. Viitattu 12.3.2024.

VALA. 2023. UAT testaus, eli hyväksymistestaus: Mitä se tarkoittaa ja miksi se on tärkeää? Verkkosivu. Viitattu 12.3.2024. https://www.valagroup.com/fi/blogi/hyvaksymistestaus/

VALA. 2023. Mitä on regressiotestaus? Verkkosivu. Viitattu 12.3.2024. https://www.valagroup.com/fi/blogi/mita-on-regressiotestaus/

Biswas, S. 2023. What Is Smoke Testing: Examples & Best Practices. Lambdatest. Verkkosivu. Viitattu 19.3.2024. https://www.lambdatest.com/learning-hub/smoke-testing

Katalon. n.d. What is End-to-End Testing? Definition, Tools, Best Practices. Katalon. Verkkosivu. Viitattu 30.3.2024. https://katalon.com/resources-center/blog/end-to-end-e2e-testing

Meszaros, G. 2007. xUnit Test Patterns: Refactoring Test Code. O'Reilly Online Learning. Viitattu 10.3.2024.

Jain, R. 2024. How to Create & Run Automated Test Scripts | A Quick Guide. Testsigma. Verkkosivu. Viitattu 3.4.2024. https://testsigma.com/blog/automated-test-scripts/

Test Evolve. n.d. Record and Replay Testing vs Scripting: Navigating the Pros and Cons for Your Testing Needs. Test Evolve. Verkkosivu. Viitattu 7.4.2024. https://www.testevolve.com/blog/record-and-replay-testing-vs-scripting

Khan, S. n.d. Keyword Driven Testing Tutorial: A Comprehensive Guide with Examples and Best Practices. Lambdatest. Verkkosivu. Viitattu 8.4.2024. https://www.lambdatest.com/learning-hub/keyword-driven-testing

Angappa, A. n.d. Web Automation Guide: Examples & Best Practices. Lambdatest. Verkkosivu. Viitattu 10.3.2024. https://www.lambdatest.com/learning-hub/web-automation

Katalon. n.d. KMS Technology Releases Katalon Studio, Free Intelligent Test Automation Toolset. Katalon. Verkkosivu. Viitattu 15.4.2024. https://katalon.com/resources-center/blog/kms-technology-releases-katalon-studio-free-intelligent-test-automation-toolset

AltexSoft. 2021. The Good and the Bad of Katalon Studio Automation Testing Tool. AltexSoft. Verkkosivu. Viitattu 16.4.2024. https://www.altexsoft.com/blog/the-good-and-the-bad-of-katalon-studio-automation-testing-tool/

Katalon Docs. n.d. Supported Technologies for Katalon Studio. Katalon Docs. Verkkosivu. Viitattu 16.4.2024. https://docs.katalon.com/katalon-studio/supported-technologies-for-katalon-studio

Katalon. n.d. Katalon Pricing. Katalon. Verkkosivu. Viitattu 16.4.2024. https://katalon.com/pricing

Katalon Docs. n.d. Manage test suites in Katalon Studio. Katalon Docs. Verkkosivu. Viitattu 18.4.2023. https://docs.katalon.com/katalon-studio/manage-test-artifacts/manage-test-suites-in-katalon-studio

Cypress. n.d. Testing is the Key to Continuous Innovation: The Story of Cypress.io. Cypress. Verkkosivu. Viitattu 16.4.2024. https://www.cypress.io/about-us/our-story

Cypress Docs. 2022. Why Cypress? Cypress Docs. Verkkosivu. Viitattu 16.4.2024. https://docs.cypress.io/guides/overview/why-cypress

Di Sciullo, V. 2024. Cypress.io pros and cons. Medium. Verkkosivu. Viitattu 17.4.2024. https://medium.com/@disciullovincenzo/cypress-io-pros-and-cons-3945ef58a8d6

Cypress Docs. 2022. Key Differences. Cypress Docs. Verkkosivu. Viitattu 16.4.2024. https://docs.cypress.io/guides/overview/key-differences

Khan, S. 2021. Advantages and Disadvantages of Cypress (End to End Testing Tool) before choosing it as your Testing Automation Tool. Medium. Verkkosivu. Viitattu 18.4.2024. https://skakarh.medium.com/advantages-and-disadvantages-of-cypress-end-to-end-testing-tool-before-choosing-it-as-your-347b6436dec8

Eficode. 2016. Robot Framework: Past, Present and Future. Eficode. Verkkosivu. Viitattu 21.4.2024. https://www.eficode.com/blog/en/blog/robot-framework

Johnson, E. 2023. Pros and Cons of Robot Framework – [2024] General Overview. Test Automation Tools. Verkkosivu. Viitattu 21.4.2024. https://testautomationtools.dev/pros-and-cons-of-robot-framework-general-overview/

Testsigma. n.d. The Comprehensive Guide to Parallel Testing. Testsigma. Verkkosivu. Viitattu 17.4.2024. https://testsigma.com/parallel-test-runs

Fortner, T. 2023. Selenium IDE Tutorial: Overview, Features, & Benefits. SauceLabs. Verkkosivu. Viitattu 17.4.2024. https://saucelabs.com/resources/blog/selenium-ide-tutorial-overview-features-&-benefits

Battat, M. 2021. 16 Reasons Why to Use Selenium IDE in 2021 (and 1 Why Not). Applitools. Verkkosivu. Viitattu 17.4.2024. https://applitools.com/blog/why-selenium-ide-2019/

LIITTEET

Liite 1. Testisuunnitelma

Testitapaus 1: Rekisteröi käyttäjä

1. Käynnistä selain
2. Siirry osoitteeseen 'http://automationexercise.com'
3. Varmista, että etusivu näkyy onnistuneesti
4. Klikkaa 'Signup / Login' -painiketta
5. Varmista, että 'New User Signup!' näkyy
6. Syötä nimi ja sähköpostiosoite
7. Klikkaa 'Signup' -painiketta
8. Varmista, että 'ENTER ACCOUNT INFORMATION' näkyy
9. Täytä tiedot: Title, Name, Email, Password, Date of birth
10. Valitse valintaruutu 'Sign up for our newsletter!'
11. Valitse valintaruutu 'Receive special offers from our partners!'
12. Täytä tiedot: First name, Last name, Company, Address, Address2, Country, State, City, Zipcode, Mobile Number
13. Klikkaa 'Create Account button'-painiketta
14. Varmista, että 'ACCOUNT CREATED!' näkyy
15. Klikkaa 'Continue' -painiketta
16. Varmista, että 'Logged in as username' näkyy
17. Klikkaa 'Delete Account'-painiketta
18. Varmista, että 'ACCOUNT DELETED!' näkyy ja klikkaa 'Continue' -painiketta

Testitapaus 2: Kirjaudu sisään käyttäjällä oikeaa sähköpostia ja salasanaa käyttäen

1. Käynnistä selain
2. Siirry osoitteeseen 'http://automationexercise.com'
3. Varmista, että etusivu näkyy onnistuneesti
4. Klikkaa 'Signup / Login' -painiketta
5. Varmista, että 'Login to your account' näkyy
6. Syötä oikea sähköpostiosoite ja salasana
7. Klikkaa 'login' -painiketta
8. Varmista, että 'Logged in as username' näkyy
9. Klikkaa 'Logout' -painiketta
10. Varmista, että 'Login to your account' näkyy

Testitapaus 3: Kirjaudu sisään käyttäjällä väärää sähköpostia ja salasanaa käyttäen

1. Käynnistä selain
2. Siirry osoitteeseen 'http://automationexercise.com'
3. Varmista, että etusivu näkyy onnistuneesti
4. Klikkaa 'Signup / Login' -painiketta
5. Varmista, että 'Login to your account' näkyy
6. Syötä väärä sähköpostiosoite ja salasana
7. Klikkaa 'login' -painiketta
8. Varmista, että virhe 'Your email or password is incorrect!' näkyy

Testitapaus 4: Rekisteröi käyttäjä varatulla sähköpostilla

1. Käynnistä selain
2. Siirry osoitteeseen 'http://automationexercise.com'
3. Varmista, että etusivu näkyy onnistuneesti
4. Klikkaa 'Signup / Login' -painiketta
5. Varmista, että 'New User Signup!' näkyy
6. Syötä nimi ja rekisteröidyn käyttäjän sähköpostiosoite
7. Klikkaa 'Signup' -painiketta
8. Varmista, että virhe 'Email Address already exist!' näkyy

(jatkuu)

Testitapaus 5: Etsi tuote

1. Käynnistä selain
2. Siirry osoitteeseen 'http://automationexercise.com'
3. Varmista, että etusivu näkyy onnistuneesti
4. Klikkaa 'Products' -painiketta
5. Varmista että käyttäjä ohjataan ALL PRODUCTS -sivulle onnistuneesti.
6. Syötä tuotenimi hakukenttään ja klikkaa search-painiketta
7. Varmista että 'SEARCHED PRODUCTS' näkyy
8. Varmista, että kaikki hakuun liittyvät tuotteet ovat näkyvissä

Testitapaus 6: Lisää tuotteita ostoskoriin

1. Käynnistä selain
2. Siirry osoitteeseen 'http://automationexercise.com'
3. Varmista, että etusivu näkyy onnistuneesti
4. Klikkaa 'Products' -painiketta
5. Vie hiiri ensimmäisen tuotteen päälle ja klikkaa 'Add to cart'
6. Klikkaa 'Continue Shopping' -painiketta
7. Vie hiiri toisen tuotteen päälle ja klikkaa 'Add to cart'
8. Klikkaa 'View Cart' -painiketta
9. Varmista, että molemmat tuotteet on lisätty ostoskoriin
10. Varmista niiden hinnat, määrät ja kokonaishinta

Testitapaus 7: Varmista tuotteen määrä ostoskorissa

1. Käynnistä selain
2. Siirry osoitteeseen 'http://automationexercise.com'
3. Varmista, että etusivu näkyy onnistuneesti
4. Klikkaa 'View Product' minkä tahansa tuotteen kohdalla etusivulla
5. Varmista, että tuotteen yksityiskohdat avautuvat
6. Aseta määräksi 4
7. Klikkaa 'Add to cart' -painiketta
8. Klikkaa 'View Cart' -painiketta
9. Varmista, että tuote näkyy ostoskorisivulla tarkalleen oikealla määrällä

Testitapaus 8: Tilauksen tekeminen: Kirjaudu sisään ennen kassalle siirtymistä

1. Käynnistä selain
2. Siirry osoitteeseen 'http://automationexercise.com'
3. Varmista, että etusivu näkyy onnistuneesti
4. Klikkaa 'Signup / Login' -painiketta
5. Täytä sähköposti, salasana ja klikkaa ' Login ' -painiketta
6. Varmista, että 'Logged in as username' näkyy ylhäällä
7. Lisää tuotteita ostoskoriin
8. Klikkaa 'Cart ' -painiketta
9. Varmista, että ostoskorisivu näytetään
10. Klikkaa 'Proceed To Checkout' -painiketta
11. Varmista osoitetiedot ja tarkista tilauksesi
12. Syötä kuvaus kommenttikenttään ja klikkaa 'Place Order'-painiketta
13. Syötä maksutiedot: Name on Card, Card Number, CVC, Expiration date
14. Klikkaa 'Pay and Confirm Order' -painiketta
15. Varmista, että ' Congratulations! Your order has been confirmed!' näkyy
16. Klikkaa 'Logout' -painiketta
17. Varmista, että 'Login to your account' näkyy