Joona Ripatti

E2E-AUTOMAATIOTESTAUS KYSELYIDEN TÄYTTÄMISESSÄ

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittelyn koulutus

2023

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Tradenomi (AMK)
Tekijä/Tekijät	Joona Ripatti
Työn nimi	E2E-automaatiotestaus kyselyiden täyttämisessä
Toimeksiantaja	Xamk / Active Life Lab / Hyviö
Vuosi	2023
Sivut	43 sivua
Työn ohjaaja(t)	Marjo Puikkonen

TIIVISTELMÄ

Opinnäytetyö käsitteli ja vertaili päästä päähän -testauksen (engl. end-to-end-testing eli E2E-testaus) automatisoinnintyökaluja sekä kuinka E2E-testejä toteutettiin tähän työhön parhaaksi nähdyllä työkalulla. Tavoitteena oli antaa toimeksiantajalle tietoa tämänhetkisistä E2E-testauksen työkaluista sekä toteuttaa testiskriptit heidän verkkosovelluksessaan käytettävien kyselyiden testaamisen nopeuttamiseksi. Toimeksiantajana toimi Kaakkois-Suomen ammattikorkeakoulun Active Life Lab -tutkimus ja kehitysyksikön Hyviö-sovelluksen tiimi. Active Life Lab on mikkeliläinen tutkimus- ja kehitysyksikkö, joka tarjoaa palveluntarjoajille käyttöön Hyviö-sovelluksen, jonka avulla pystytään keräämään mittaustietoja erilaisten kyselyiden ja mittareiden avulla.

Työn teoriaosuudessa käsiteltiin pääpiirteittäin ohjelmistotestausta ja sen tärkeyttä sekä sen eri osa-alueita. Tämän jälkeen perehdyttiin myös testauksen automatisointiin ja keskityttiin erityisesti E2E-testaukseen sekä sen automatisointiin. Teoriaosuudessa perehdyttiin myös syvällisemmin E2E-testauksen suosittuihin työkaluihin ja vertailtiin niitä keskenään. Vertailun tarkoituksena oli tuoda esille työkalujen olennaisimmat erot ja helpottaa työhön käytettävän työkalun valitsemista. Työssä vertailtuja työkaluja olivat Cypress, Selenium, Playwright ja Katalon.

Opinnäytetyön kehittämistyöosuus toteutettiin käyttämällä Playwrightia. Kehittämistyöosuus alkoi työkalun asentamisella ja käyttöönotolla. Käyttöönoton jälkeen suunniteltiin halutut testitapaukset ja aloitettiin testien luominen. Samalla perehdyttiin siihen, kuinka työkalua käytettiin testienluomista varten. Testiskriptien valmiiksi saannin jälkeen käytiin läpi, miten testejä pystyttiin suorittamaan ja mitä kaikkia eri vaihtoehtoja testien suorittamiseen liittyy. Kehittämistyöosuuden lopuksi käytiin vielä läpi testien tuloksia ja pohdittiin, mitä hyötyjä testeistä oli.

Tuloksena saatiin toimivat testiskriptit, jotka nopeuttivat kyselyiden testaamista merkittävästi ja hyvää taustatietoa suosituimmista E2E-testauksen automatisoinnintyökaluista. Työn lopussa käytiin vielä läpi omia ajatuksia, kuinka työ onnistui ja mahdollisia jatkokehitysideoita.

Asiasanat: ohjelmistotestaus, automaatiotestaus, E2E-testaus, Playwirght

South-Eastern Finland
University of Applied Sciences

Degree title	Bachelor of Business Administration
Author (authors)	Joona Ripatti
Thesis title	E2E automation testing in the completion of questionnaires
Commissioned by	Xamk / Active Life Lab / Hyviö
Time	2023
Pages	43 pages
Supervisor	Marjo Puikkonen

ABSTRACT

This thesis examined and compared end-to-end testing (E2E testing) automation tools and how to implement E2E testing with the tool that was best suited for this work. The objective was to provide thesis commissioner with information on current E2E testing tools, and to implement test scripts to speed up the testing of questionnaires used in their web application. The commissioner was Active Life Lab research and development team of the Hyviö application at the South-Eastern Finland University of Applied Sciences. Active Life Lab is a research and development unit based in Mikkeli, Finland, that provides service providers with the Hyviö application which enables them to collect measurement data using various questionnaires and metrics.

The theoretical part of the thesis outlined software testing and its importance, as well as its different aspects. This was followed by an introduction to the automation of testing, with a particular focus on E2E testing and its automation. The theoretical part also went into more detail on the popular tools for E2E testing and compared them with each other. The purpose of the comparison was to highlight the main differences between the tools and to ease the selection of the tool to be used for the work. The tools compared were Cypress, Selenium, Playwright and Katalon.

The development part of the thesis was done using Playwright. The development part started with the installation and implementation of the testing tool. After the implementation, the desired test cases were planned and the development of tests was started. During this process, it was explained how to use the tool to create tests. Once the test scripts were ready, it was then explained how the tests could be run and what all different options there were for running the tests. At the end of the development phase, the results of the tests were reviewed and the benefits of the tests were assessed.

The result was working test scripts that significantly shortened the time needed to test the surveys and good background information on the most popular E2E testing automation tools. At the end of the thesis, thoughts on the success of the work and possible ideas for further development were discussed.

Keywords: software testing, automation testing, E2E testing, Playwirght

SISÄLLYS

1 JOHDANTO

Ohjelmistotestaus on hyvin tärkeä osa jokaisen sovelluksen, web-sivun tai minkä tahansa muun ohjelman valmistusprosessia. Hyvällä ja laadukkaalla testaamisella vältytään monilta ongelmilta sekä säästytään isoilta kustannuksilta. Testaukseen tehtyjä työvälineitä on monia, mutta kaikki työvälineet eivät sovellu mihin tahansa testaustapaukseen. On hyvä tietää ja perehtyä työvälineisiin, joita aikoo käyttää omaan projektiinsa. On myös hyvä vertailla mahdollisia työvälineitä, jotta löytää juuri sen oikean testaustyövälineen. Tässä työssä tullaan perehtymään automaatiotestaukseen ja erityisesti päästä-päähän-testauksen (engl. end-to-end-testing eli E2E) automatisointiin sekä sen työkaluihin.

Opinnäytetyön tavoitteena on nopeuttaa toimeksiantajan verkkopohjaisen sovelluksen kyselyiden testaamisprosessia. Ongelmaa aloitetaan ratkaisemaan perehtymällä käytetyimpiin ja suosituimpiin E2E-testiautomaatiotyökaluihin. Perehtymisen jälkeen vertaillaan työkaluja keskenään, jotta pystytään valitsemaan paras testaustyökalu toimeksiantajan kyselyiden testaamiseen. Parhaaksi todettuun työkaluun perehdytään tarkemmin ja toteutetaan automatisoitujatestejä sen avulla. Luodut testiskriptit nopeuttavat kyselyiden testaamista huomattavasti ja antavat hyvän pohjan, jos toimeksiantaja päättää laajentaa testaamista sovelluksen muihinkin osiin.

Opinnäytetyön toimeksiantajana toimii Kaakkois-Suomen ammattikorkeakoulun Active Life Lab -tutkimus ja kehitysyksikön Hyviö-sovelluksen tiimi. Active Life Lab on mikkeliläinen tutkimus- ja kehitysyksikkö, joka auttaa yrityksiä kehittämään hyvinvointipalveluita, jotka tuottavat mitattavia hyvinvointivaikutuksia. Active Life Lab tarjoaa palveluntarjoajille käyttöön Hyviö-sovelluksen, jonka avulla palveluntarjoaja voi kerätä järjestelmällisesti tietoa palvelujensa vaikuttavuudesta. Hyviö toimii selainpohjaisena sovelluksena, jossa se kerää mittaustietoja erilaisten kyselyiden ja mittareiden avulla.

Tämän työn teoriaosuudessa käydään läpi ohjelmistotestausta ja etenkin E2E-testauksen automatisointia sekä sen työkaluja. Opinnäytetyön olennaisimmat

käsitteet ovat ohjelmistotestaus, automaatiotestaus ja E2E-testaus. Teoriaosuudessa vertaillaan tämänhetkisiä E2E-testauksen automatisoinnin suosituimpia työkaluja ja valitaan yksi työkalu, jolla toteutetaan opinnäytetyön kehittämistyöosuus. Teoriaosuuden tarkoituksena on antaa toimeksiantajalle hyvä kuva tämänhetkisistä E2E-testauksen automatisoinnin työkaluista.

Opinnäytetyön kehittämistyönosuudessa toteutetaan teoriaosuudessa valitulla työkalulla toimeksiantajalle toimivat testiskriptit Hyviö-sovelluksen kyselyiden testaamista varten. Tämän osion alussa käydään läpi, kuinka työkalu otetaan käyttöön ja muut mahdolliset huomiot asennukseen liittyen. Tämän jälkeen käydään läpi tarvittavat vaiheet testiskriptien luomiseen ja muokkaamiseen, jotta toimeksiantaja pystyy itsekin tehdä ja muokata testejä. Osion lopussa käydään myös läpi testien tuloksia ja pohditaan, minkälaisia hyötyjä testeistä sai sekä mitä hyötyjä testien automatisoinnista oli.

2 OHJELMISTOTESTAUS

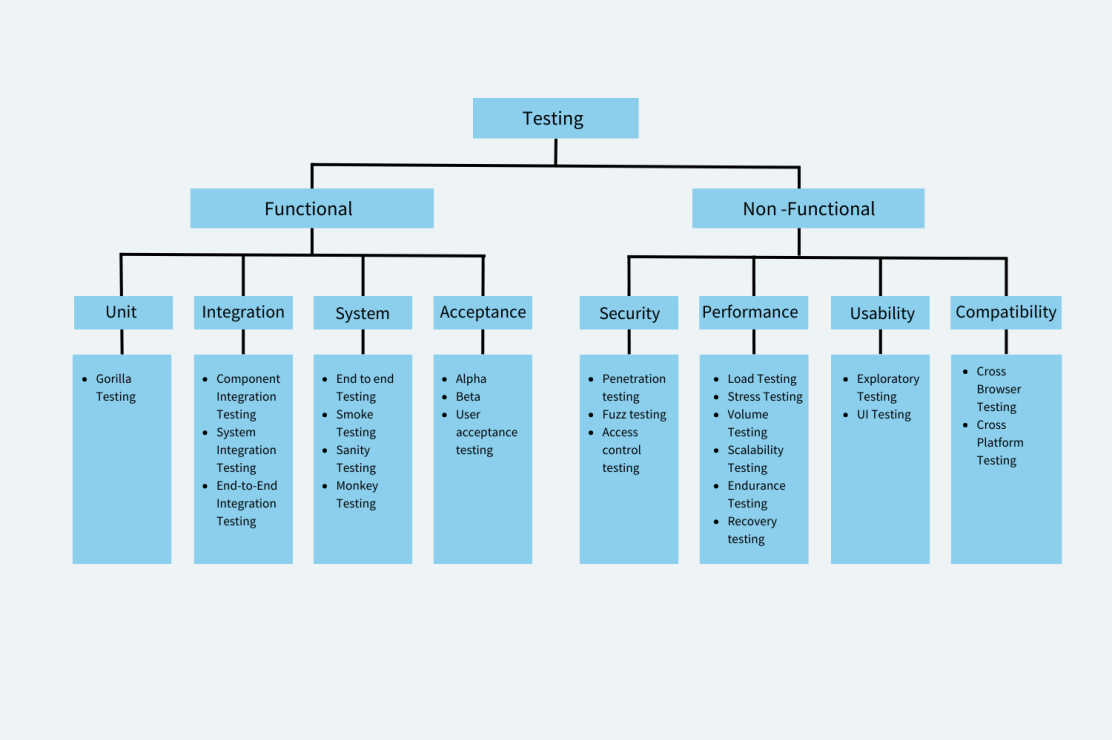
Ohjelmistotestaus on laaja käsite itsessään ja se sisältää monia eri alakäsitteitä. Ohjelmistotestauksella tarkoitetaan työtä, jota tehdään, jotta varmistutaan, että ohjelmistotuotteesta tulee toivotun kaltainen ja kaikki valmiit ominaisuudet toimivat niin kuin pitääkin (Kasurinen 2017, 8). Testausta voidaankin ajatella ohjelmistokehityksen laadun valvontana. Testit varmistavat, että järjestelmä tekee sen, mitä sen pitääkin tehdä, ja toisaalta varmistaa sen, että ohjelmisto ei tee mitään sellaista, mitä sen ei kuuluisikaan tehdä. (Tasanto 2018.)

Joskus testaus voidaan nähdä resursseja vievänä ja siksi siihen ei panosteta. Tosiasia on kuitenkin se, että hyvällä testaamisella säästää paljon aikaa, resursseja ja rahaa. Vertailusta on käynyt ilmi, että tuotteiden testauksella ja kannattavuudella on selkeä yhteys toisiinsa. Yritykset, jotka testaavat tuotteitansa huolellisesti, saavat parempaa katetta verrattuna yrityksiin, jotka eivät testaa tuotteitansa yhtä huolellisesti. Ero korostuu entisestään, kun otetaan huomioon huonojen tuotteiden tuoma huono maine ja maineesta johtuva asiakaskato. Testaamiseen kannattaakin panostaa jo suunnitteluvaiheessa, sillä virheiden korjaaminen suunnitteluvaiheessa maksaa vain 1–2 prosenttia siitä, mitä se maksaisi julkaisun jälkeen tehtynä. (Kasurinen 2017, 9.)

Ohjelmistotestaus on siis hyvin olennainen osa ohjelmistotuotantoa. Täydellinen testaaminen ei kuitenkaan ole yksinkertaisesti mahdollista. Täydellisessä testaamisessa otettaisiin huomioon kaikki mahdolliset vaihtoehdot, kaikille mahdollisille tapauksille ja tehtäisiin näille kaikille vaihtoehdoille sekä tapauksille omat testinsä. Jo yksinkertaisessakin sovelluksessa tarvittavien testien määrä nousisi useisiin miljooniin testeihin, kun pitäisi testata kaikki mahdollinen ja testien suorittamiseen tarvittava aika olisi reilusti yli ihmisen elinajan. (Kasurinen 2017, 14.) Tämän takia on tärkeä tehdä hyvä suunnitelma ja määritellä mahdollisimman tarkasti, minkälaista ohjelmistoa ollaan toteuttamassa, ketkä ovat pääasiakasryhmät ja millä laitteella ohjelmistoa käytetään (Tasanto 2018). Ilman näitä määrittelyjä oikean testaustavan ja testityökalun valitseminen vaikeutuu. Testaustyyppejä on useita erilaisia ja jokaisella testaustyypillä on oma käyttötarkoituksensa. Eri testaustyypeille löytyy myös omat työkalut.

2.1 Ohjelmistotestauksen eri osa-alueet

Ohjelmistotestauksen voi jakaa kahteen isompaan osa-alueeseen: toiminnallinen testaus ja ei-toiminnallinen testaus (Doshi 2023). Molemmat näistä pitävät sisällään useita pienempiä ja yksityiskohtaisempia osa-alueita. Näiden pienempien osa-alueiden sisälle on sijoitettu eri testausmenetelmiä riippuen siitä, minkälaisia ja minkä tyyppisiä nämä testit ovat. Yksittäisiä testejä pystyisi vielä jakamaan sen mukaan, voiko tai kannattaako testejä automatisoida vai pitääkö testit suorittaa manuaalisesti. Alla on havainnollistettu, kuinka ohjelmistotestaus jakautuu toiminnallisiin ja ei-toiminnallisiin testeihin (kuva 1).



Kuva 1. Eri testaustyyppien luokittelu (Doshi 2023)

Toiminnallisilla testeillä (engl. functional testing) tarkoitetaan testejä, joiden avulla varmistutaan, että ohjelmisto toimii niin kuin käyttäjä olettaa sen toimivan. Toiminnalliset testit testaavat siis ohjelmiston käytettävyyttä, erityisvirhetilanteita ja muita tärkeitä toiminnallisuuksia ohjelmistossa. (TestingXperts 2022.) Toiminnallisia testejä on monia erilaisia. Seuraavaksi muutamia esimerkkejä niistä. Yksikkötesteissä (engl. unit testing) testataan ohjelmiston yksittäistä komponenttia erillään ohjelmistokokonaisuudesta, jotta varmistutaan sen toimivuudesta. Eräs tyyli tehdä yksikkötestejä on gorillatestaus, jossa keskitytään yksittäiseen komponenttiin perusteellisesti ja varmistutaan, että se kestää suuria kuormituksia sekä toimii optimaalisesti äärriolosuhteissakin. Integraatiotestauksessa testataan, että ohjelmiston eri komponentit toimivat oikein yhdessä ja, että ne ovat integroitu oikein ohjelmistoon. Integraatiotestejä on erilaisia, kuten järjestelmän integrointitestaus, jossa keskitytään testaamaan ohjelmiston eri osajärjestelmien välistä vuorovaikutusta. (Doshi 2023.) Toiminnallisia testejä on vielä paljon enemmän, mutta olennaisin testaustyyppi tämän opinnäytetyön kannalta on päästä päähän -testaus (engl. end-to-end-testing eli E2E-testing).

Ei-toiminnalliset testit (engl. non-functional testing) puolestaan keskittyvät siihen, kuinka hyvin ohjelmisto suorittaa toimintonsa, eikä niinkään se, mitä se

tekee (Doshi 2023). Erilaisia ei-toiminnallisia testejä on myös monia. Seuraavaksi muutamia esimerkkejä niistä. Tietoturvatestaus, jossa testataan ohjelmiston tietoturvallisuutta. Eräs tapa suorittaa tietoturvatestausta on tunkeutumistestaus, jossa simuloidaan hakkerointiyritystä käyttämällä hyväksi ohjelmiston mahdollisia tietoturva-aukkoja. Suorituskykytestauksessa nimensä mukaan testataan ohjelmiston suorituskykyä. Suorituskykytestejä on monia erilaisia. Kuormitustestit auttavat löytämään ohjelmiston maksimaalisen rajan ja varmistavat sen toimivuuden maksimirajassa. Stressitestaus auttaa sen sijaan löytämään pisteen, jossa ohjelmisto ei pysty enää käsittelemään tietoja ja toimintoja. Kestävyystestaus tuo esille, pystyykö ohjelmisto pyörimään normaaleissa olosuhteissa pitkiäkin aikoja. (Doshi 2023.) Ei-toiminnallisien testien lista jatkuisi vielä pidemmällekin, mutta en keskity niihin tässä opinnäytetyössä. Käyn kuitenkin vielä käyttöliittymätestauksen hieman tarkemmin läpi, sillä haluan tuoda esille käyttöliittymätestauksen ja E2E-testauksen eron.

2.2 Käyttöliittymätestaus

Graafinen käyttöliittymätestaus, eli GUI-testaus, on ohjelmistotestauksen osa-alue, jossa keskitytään testaamaan graafisen käyttöliittymän eri osia ja niiden toimivuutta. Käyttöliittymiä on useampia erilaisia, mutta ne pystytään kuitenkin jakamaan kahteen pääryhmään: komentorivikäyttöliittymät ja graafiset käyttöliittymät. Komentorivikäyttöliittymässä käyttäjä kirjoittaa terminaaliin komentoja, joiden avulla hän pystyy käyttämään sovellusta. Graafisessa käyttöliittymässä käyttäjä puolestaan voi painella eri painikkeita, kirjoittaa tekstiä tekstikenttiin ja niin edespäin. GUI-testaus ja UI-testaus ovat hyvin samankaltaisen kuuloisia termejä, mutta niillä on kuminkin selkeä ero. Termi UI pätee kuitenkin kaikkiin käyttöliittymiin, joten voidaan ajatella, että GUI on yksi käyttöliittymätestauksen osa-alue. (Testim 2020.) Käyttöliittymätestaus on käytettävyystestausta ja kuuluu ei-toiminnalliseen testaukseen.

UI-testauksessa suoritetaan monia erilaisia testejä käyttöliittymän eri osiin ja osien toimivuuteen. Kaikkia UI-testauksen osa-alueita ei pystytä automatisoimaan, vaan osa niistä on tehtävä manuaalisesti. Käyttökokemusta ei esimerkiksi pysty testaamaan automaatiolla mitenkään, vaan sitä on aina testattava manuaalisesti. Joitakin osioita pystyy sen sijaan automatisoimaan kuten nap-

pien painallukset, tekstikenttien täytöt ja muut kohteet mitä kone pystyy painamaan sekä täyttämään. Automatisoituja testejä on paljon nopeampi suorittaa, eikä niissä ilmene inhimillisiä virheitä, joita ihmiset saattavat tehdä, kun he joutuvat testaamaan samaa asiaa useamman kerran. Automatisoidut testit usein myös raportoivat tulokset automaattisesti verrattuna manuaaliseen testaukseen, jossa raportti pitää itse kirjoittaa, tallentaa ja lähettää oikeille henkilöille. (Bose 2023a.)

2.3 E2E-testaus

E2E-testaus tai end-to-end-testing (suom. päästä päähän -testaus) tarkoituksena on testata sovelluksen kulkua alusta loppuun. E2E-testaus on järjestelmätestausta ja kuuluu toiminnalliseen testaukseen. Tässä testaustavassa simuloidaan todellista käyttäjäskenaariota ja varmistutaan sovelluksen toimivuudesta. Testit suoritetaan oikeassa käyttöympäristössä, jotta testit olisivat mahdollisimman todenmukaisia. (Software Testing Help 2023.) E2E-testaus on yksi tärkeimmistä testaustavoista, koska sen avulla pystytään varmistumaan sovelluksen toimivuus oikeassa käyttöympäristössä. Tämä testaustyyli voi olla aikaa ja resursseja vievää, riippuen testattavasta kokonaisuudesta, mutta nämä testit paljastavat yleensä hyvin toimiiko sovellus oikeasti niin kuin oli ajateltu. E2E-testaus paljastaa siis, toimivatko kaikki komponentit yhdessä, ja E2E-testauksessa tulee myös testattua käyttöliittymän toimivuutta (Testim 2022b.)

E2E-testejä suoritetaan usein manuaalisesti, sillä niiden automatisointi voi olla työlästä, ja pienetkin muutokset voivat rikkoa testit. E2E-testejä pystytään kuitenkin automatisoimaan ja hyvin tehtynä ne säästävät paljon aikaa sekä niitä on helppo ylläpitää. Ilman automaatiotestausta laadukkaita ohjelmistoja ei voi julkaista nopeasti (Testim 2022b). E2E-testejä on kahta erilaista, horisontaalinen ja vertikaalinen. Horisontaalinen E2E-testaus on kaikista käytetyin ja tunnetuin testausmuoto. Tässä testausmuodossa pyritään varmistumaan, pystyykö käyttäjä navigoimaan sovelluksessa, toimiiko se odotetusti ja löytyykö sovelluksesta odottamattomia virheitä tai poikkeuksia. Vertikaalisessa E2E-testauksessa sen sijaan tarkoituksena on testata järjestelmän tai tuotteen jokainen osa alusta loppuun laadun varmistumiseksi. Vertikaalista testausta käytetään yleensä monimutkaisiin tietotekniikkajärjestelmiin ja sen kriittisten

komponenttien testaamiseen. Vertikaalisissa testeissä ei yleensä ilmene minkäänlaisia käyttöliittymiä tai käyttäjiä. (Gillis 2023.)

2.4 Automaatiotestaus

Ohjelmistotestejä pystytään suorittamaan manuaalisesti ja automatisoidusti. Manuaalisessa testaamisessa testaaja käy itse ohjelmiston eri osat ja toiminnot läpi sekä kirjaa havainnot ylös. Manuaalinen testaaminen voi viedä paljon aikaa ja resursseja, etenkin jos samoja testejä joudutaan tekemään useita kertoja. (Smartbear s.a.) Automatisoitu testi on etukäteen suunniteltu ja mietitty testi, joka suoritetaan koneellisesti askelittain (Tasanto 2018). Automatisoidut testit eivät vaadi vahtimista, vaan testit voi laittaa pyörimään milloin tahansa ja kun testit ovat valmiita ohjelma raportoi automaattisesti menikö testit läpi vai ei sekä raportoi mahdolliset virhe kohdat. Automatisoidut testit vapauttavat siis työntekijän muihin hommiin siksi ajaksi, kun testit suoritetaan. Automatisoituja testejä pystytään myös suorittamaan useita samaan aikaan, mikä nopeuttaa entisestään testaustyötä. (Smartbear s.a.)

Kaikkea ei ole järkevä automatisoida, eikä kaikkea edes pysty automatisoimaan. Käyttökokemusta ei esimerkiksi pysty testaamaan automatisoiduilla testeillä. Tämän takia automaatiotestausta tulisi aina täydentää manuaalisella testaamisella. Automaatiotestejä suunniteltaessa on otettava huomioon myös projektin koko, testauksen työmäärä ja siihen käytettävät resurssit. (Tasanto 2018). Automatisoitujen testien suunnittelu, tekeminen ja ylläpito voi olla resursseja vievää sekä kallista. Tästä huolimatta automatisoitujen testien ylläpito ja käyttäminen on helpompaa sekä halvempaa tietyn pisteen jälkeen, verrattuna manuaaliseen testaamiseen. Automatisoituja testejä kannattaa tehdä kohteille, jotka voidaan toistaa useaan kertaan muuttumattomina tai pienillä muutoksilla. (Kasurinen 2017, 50–51.)

Automaatiotestauksen prosessi voi ja varmasti, vaihtelee eri projekteissa riippuen niiden koosta sekä käytössä olevista resursseista. Tässä on eräs malli, kuinka automaatiotestejä luodaan. Aivan ensimmäiseksi pitää valita sopivin testaustyökalu projektiin. Tässä vaiheessa pitäisi olla jo tiedossa mitä ollaan testaamassa, jotta osataan valita oikea työkalu testien tekoa varten. Seuraavaksi määritellään automaatiotestauksen laajuus. Tässä vaiheessa otetaan

huomioon käytössä olevat resurssit, mitkä asiat ovat tärkeimpiä testauskohteita ja mitä kaikkea automatisoidaan. Tämän jälkeen on suunnitteluvaihe, jossa määritellään testit mitä halutaan suorittaa ja mitä odotetaan näiden testien tuloksien olevan. Kun suunnitelmat ovat tehty siirrytään suorittamaan testejä. Testejä suoritetaan ja tuloksia analysoidaan tässä vaiheessa. Viimeinen vaihe on testiskriptien ylläpito. Tässä vaiheessa päivitetään skriptejä tarpeen mukaan, jotta ne pysyvät toiminnallisina, vaikka koodipohjaan tai testaustyökaluun tulisikin muutoksia. (Cummings-John 2020.)

3 OHJELMISTOTESTAUKSEN TYÖKALUT

Ohjelmistotestausta voi suorittaa monella eri tapaa, joten työkaluja testien suorittamiseen löytyy myös monia erilaisia. Manuaalisissa testeissä ei suoriteta testejä erillisillä työkaluilla vaan testit suoritetaan itse, kun taas automatisoiduissa testeissä käytetään työkaluja testien ajamiseen (What is Manual Testing? A Comprehensive Guide (With Examples). Testien automatisoinnilla tarkoitetaan prosessia, jossa testit suoritetaan mahdollisemman vähäisellä ihmisen vuorovaikutuksella nopeuden ja tehokkuuden lisäämiseksi. Automaatiotesteillä viitataankin siis testien ajamisen automaatioon, itse testiskriptin joutuu silti itse manuaalisesti tekemään. Minkälainen skripti on, riippuu työkalusta ja testistä. Testit joudutaan ehkä itse koodaamaan tai mahdollisesti pystytään käyttämään toimintojennauhoittamis-ominaisuutta, riippuen työkalusta. Lyhyesti sanottuna, automaatiotestaustyökalut ovat erilaisia ohjelmistoja, joiden avulla ihmiset määrittelevät testaustehtäviä, jotka sen jälkeen suoritetaan mahdollisemman vähäisellä ihmisvuorovaikutuksella. (Testim 2022a.)

Automatisoitujatestejä on monenlaisia. On olemassa perinteisimpiä tyylejä, joissa testit joudutaan koodaamaan itse, on myös olemassa koodittomia testaus tyylejä ja näiden kahden yhdistelmiä. Eräät työkalut esimerkiksi käyttävät avainsanoja, joiden avulla testejä on helpompi rakentaa, vaikka ei tietäisi koodaamisesta mitään. Testaustyökaluja löytyy työpöytäsovelluksille, nettisivuille ja mobiililaitteille. Työkalut voivat olla kaupallisia tai avoimeen lähdekoodiin perustuvia. Osat työkalut ovat täysin ilmaisia, osassa saattaa saada lisä ominaisuuksia maksua vastaan ja jotkin työkalut ovat puolestaan täysin maksullisia. (Testim 2022a.)

Automaatiotestauksentyökaluille on myös olemassa muutama yleinen viitekehys (engl. framework). Nämä ovat lineaarinen viitekehys, modulaarinen viitekehys ja kirjastoarkkitehtuuriin perustuva viitekehys (engl. library architecture framework). Lineaarinen viitekehys on kaikista yksinkertaisin viitekehys ja siinä testaajat koodaavat sekä suorittavat testiskriptit jokaiselle yksittäiselle testitapaukselle. Yksinkertaisuutensa vuoksi tämä viitekehys sopii parhaiten pienille tiimeille ja aloittelijoille. Modulaarisessa viitekehyksessä jokainen testitapaus järjestetään pienempiin osiin ja näitä osia kutsutaan moduuleiksi sekä sen lisäksi nämä moduulit ovat toisistaan riippumattomia. Tämän jälkeen minkä tahansa skenaarion moduulit käsitellään yhdenmukaisesti pääskriptin avulla. Tämä säästää testaajien aikaa sekä optimoi heidän työnkulkuansa. (Types of Automation Testing: A Beginner’s Guide.)

Modulaarisen viitekehyksen käyttö vaatii kuitenkin hyvää ennakkosuunnittelua ja testiautomaation tuntemusta, jotta siitä saa kaiken hyödyn irti. Kirjastoarkkitehtuurikehys perustuu modulaariseen kehykseen, mutta tämä kehys ei pilko testitapauksia osiin, vaan kehys ryhmittelee samankaltaiset tehtävät testiskriptissä funktioihin ja tallentaa ne kirjastoon. Nämä funktiot koostuvat tehtävistä, joilla on yhteiset tavoitteet, joten testiskripti voi käyttää niitä aina, kun toimintoja tarvitaan. Tämä mahdollistaa paremman uudelleenkäytettävyyden ja joustavuuden testauksessa. Tämä kehys vaatii kuitenkin enemmän aikaa testien koodaamiseen ja hyvää tuntemusta testien automatisoinnista. (Types of Automation Testing: A Beginner’s Guide.)

3.1 E2E-testauksen automatisointi

E2E-testejä pystytään automatisoimaan erilaisilla työkaluilla. Testejä pystytään suorittamaan myös manuaalisesti, mutta manuaalisissa testeissä on aina riskinä inhimilliset virheet ja niiden suorittamiseen voi mennä pitkä aika, jos testattava kokonaisuus on suuri. Automatisoiduissa testeissä ei esiinny inhimillisiä virheitä testien suorituksen aikana. Testit ovat myös toistettavissa paljon helpommin ja testien tulokset ovat aina yhtä tarkkoja jokaisen suoritus kerran jälkeen. Automatisoituja testejä voi myös suorittaa paljon nopeammin kuin manuaalisia testejä. (RTIH 2023.)

E2E-testauksen työkaluja löytyy eri ohjelmointikielille, eri osaamistasoille, työkalut voivat toimia erillisinä ohjelmina tai suoraan selaimen sisällä. Osa työkaluista vaatii erillisien lisäosien asentamisen ja osassa taas tulee kaikki tarpeellinen valmiiksi asennettuna. Asennustapoja työkalujen asentamiseen on myös erilaisia, riippuen siitä, mitä ohjelmointikieltä projekti käyttää. On siis tärkeä selvittää, mikä työkalu soveltuu parhaiten projektiin, jossa sitä tullaan käyttämään.

3.2 Cypress

Cypress julkaistiin lokakuussa 2018. Kyseessä on siis vielä melko uusi testaustyökalu, mutta siitä huolimatta Cypress on kerännyt paljon huomiota ja suosiota. Sillä on yli 5 000 000 viikoittaista latausta (Cypress s.a.). Cypress on JavaScript-pohjainen testauskehys, joka perustuu Mochaan ja Chaihin sekä käyttää myös Node.js:ää. (Kinsbruner 2021.) Mocha on eräs toinen testauskehys JavaScriptille ja Chai on assertion-kirjasto, jota käytetään pääsääntöisesti Mochan rinnalla (Felice 2023). Cypressin avulla voi suorittaa E2E-testejä ja komponenttitestejä. Komponenttitestaus on uusi ominaisuus, joka lisättiin hiljattain Cypressiin ja siinä käyttäjät voivat testata komponentteja suoraan selaimessa samalla kun käyttävät kyseisiä komponentteja. Tämän ominaisuuden ansiosta Cypress on mahdollinen vaihtoehto työkaluihin kuten Jest ja Storybook. (Perfecto 2022.) Cypress on jatkuvasti kehittyvä työkalu ja uusia ominaisuuksia sekä päivityksiä lisätään säännöllisesti.

Cypress on ilmainen ja avoimeen lähdekoodiin perustava testaustyökalu, jota voi käyttää kaikilla suurimmilla käyttöjärjestelmillä, Windos, macOS ja Linux. Cypress tukee rinnakkaistestausta, joka mahdollistaa useiden testien ajamisen samanaikaisesti sekä siinä on hyvät testiraportit, jotka työkalu muodostaa testien suorituksen jälkeen automaattisesti. (Cypress s.a.)

Cypress Cloudista, jonne testitulokset tallentuvat ja josta pystyy tarkastelemaan eri näköisiä tuloksia sekä mittareita, on olemassa erilaisia tilaus vaihtoehtoja. Tästäkin löytyy ilmaisversio, mutta maksua vastaan saa kuitenkin nostettua esimerkiksi: tallennus määrää ajettujen testien tuloksista, organisaation käyttäjämäärää, tietoja säilytetään pidempään maksua vastaa ja monia muita hyötyjä, jos yritys on isompi. (Cypress s.a.)

Cypress toimii suoraan selaimessa sen DOM-manipulaatiotekniikan avulla. Tämän ansiosta testien suorittaminen on nopeata ja se tuo myös muita hyviä ominaisuuksia mukanaan. Työkalua käytettäessä ei tarvitse käyttää erillisiä odotuksia, että elementit, DOM tai muut asiat kerkeävät latautumaan, sillä tarvittavat odotukset tapahtuvat automaattisesti. Cypressissä on myös paljon muita hyödyllisiä ominaisuuksia, kuten ajassa matkustaminen, jossa pystyy tarkastelemaan jokaista kohtaa, jonka testi suoritti. Kuvankaappaukset epäonnistuneista testeistä, video koko testin suorittamisesta, debuggaus suoraan selaimen kehittäjäntyökaluista, samoja ominaisuuksia kuin yksikkötestauksessa, verkkoliikenteen hallintaa ja muita ominaisuuksia. Ennen kaikkea Cypressin testitulokset ovat toistettavia ja nopeita. Cypress ei käytä WebDriveria eikä perustu Seleniumiin. (Kinsbruner 2021.)

Myös huonoja puolia löytyy jonkin verran. Testejä ei pysty suorittamaan useammalla selaimella samanaikaisesti, eikä testejä pysty suorittamaan useilla eri välilehdillä samanaikaisesti. Useita testejä pystyy kuitenkin suorittamaan samanaikaisesti. Cypress tukee vain yhtä ohjelmointi kieltä ja se on JavaScript. Työkalun tuettuihin selaimiin kuuluvat vain Chrome-, Firefox- ja Edge-selaimet. Safaria Cypress ei tue vielä virallisesti, eikä myöskään iFramejen testaamista. Cypressillä ei myöskään pysty testaamaan natiiveja mobiilisovelluksia, mutta se tukee eri näyttökokojen testaamista verkkosivuilla, joten mobiili näkymän emulointi on mahdollista. (Unadkat 2023.) Cypress on vielä melko uusi, joten sille ei ole kerennyt vielä kasvamaan yhtä isoa yhteisöä kuin joillekin vanhemmille testaustyökaluille, joten joskus tiedon löytäminen voi olla hankalampaa verrattuna muihin työkaluihin.

3.3 Selenium

Selenium julkaistiin jo vuonna 2004, mutta on edelleen suuressa suosiossa ja laajalti käytetty verkkosivujen automaatiotestauksessa. Selenium WebDriverilla on 1 800 000 miljoonaa viikoittaista latausta. Selenium on avoimeen lähdekoodiin perustava ilmainen testaustyökalu ja kirjastokokonaisuus. Moni suosii Seleniumia, sillä se tukee montaa eri ohjelmointikieltä kuten: JavaScript, Pyhton, C#, PHP ja Java. Työkalu on suosittu myös, koska sillä pystyy testaa-

maan kaikkia suurimpia ja käytetyimpiä selaimia, Chrome, Firefox, Edge, Safari sekä IE, WebDriverin avulla. Selenium on myös helposti laajennettavissa ja laajennuksien avulla pystyy esimerkiksi suorittamaan mobiilitestaamista. Työkalun pystyy myös integroimaan muihin suosittuihin työkaluihin ja testauskehyksiin. (BrowserStack s.a.)

Selenium koostuu neljästä eri komponentista, Selenium IDE, Selenium RC, Selenium WebDriver ja Selenium Grid. Selenium IDE on Chrome- ja Firefox-lisäosa, jonka avulla pystyy nauhoittamaan käyttäjän toimintoja selaimessa ja nauhoituksen jälkeen toistamaan kyseiset toiminnot automaattisina testeinä. Tämän jälkeen ohjelma luo testeistä testiskriptit eri ohjelmointikielille uudelleen käyttöä varten. Selenium RC rakennettiin automatisoimaan verkkosovelluksien testausta, simuloimalla käyttäjän toimintoja eri selaimilla ja alustoilla. Tällä komponentilla oli kuitenkin paljon heikkouksia esim. se oli melko hidas suorittamaan testejä, sillä se käytti JavaScript-pohjaista välityspalvelinmekanismia, mikä aiheutti epävakautta ja muita rajoituksia. Ongelmana oli myös se, että jokainen eri selain vaati oman ajurin, jotta ohjelma toimi oikein. Tämä aiheutti paljon ylläpitoa ja yhteensopivuus ongelmia. Nykyään Selenium RC:tä ei käytetä juurikaan, sillä Selenium WebDriver on korvannut sen käyttötarkoituksen. (BrowserStack s.a.)

SeleniumWebDriver on web-kehys, jonka avulla voi suorittaa selaintenvälisiä testejä. WebDriveria käytetään web-pohjaisten sovellusten testauksen automatisoinnissa, jotta varmistutaan niiden toimivan odotetusti. Selenium Grid on älykäs välityspalvelin, jonka avulla testejä pystyy suorittamaan rinnakkain useilla koneilla. Välityspalvelin reitittää komentoja etäkäytettäville selain instansseille, joissa yksi palvelin toimii keskuksena. Tämä keskitin reitittää JSON-muodossa olevat testikomennot useisiin rekisteröityihin Grid-solmuihin. Tämä mahdollistaa testien suorittamisen useilla selaimilla samanaikaisesti, testejä pystyy suorittamaan myös eri selain versioille, eri laitteille, eri käyttöjärjestelmille (Windows, macOS, Linux) ja useammilla välilehdillä. (BrowserStack s.a.)

Selenium voi olla hieman vaikea ottaa käyttöön, sillä se ei ole valmis paketti vaan käyttäjän pitää ladata tarvittavat lisäosat riippuen siitä, mihin käyttötarkoitukseen se tulee. Seleniumilla on suuri yhteisö, joten apua ja tietoa löytyy

melko helposti, mutta sen käyttöönotto voi silti olla hankalaa, etenkin jos on aloittelija ohjelmistotestauksessa. Itse testien suorittaminen työkalulla on hieman hidasta, koska se käyttää WebDriveria. Yksi lisä haitta minkä WebDriver tuo tullessaan on eri selain- ja WebDriver-versiot. Nämä eivät aina saata täsmätä ja silloin testit eivät välttämättä toimi oikein. Seleniumssa ei ole myöskään automaattisia odotuksia, vaan tarvittavat odotukset joudutaan itse määrittelemään. Työkalu ei myöskään tee automaattisesti testiraportteja vaan se vaatii lisäosan, joka luo raportit. (BrowserStack s.a.) Selenium on näistä ongelmista huolimatta silti yksi suosituimmista automaatiotestaustyökaluista. Se mahdollistaa monen erityyppisen testauksen ja testejä voi suorittaa samanaikaisesti monia. Jos päätyy käyttämään Seleniumia ja käyttää aikaa sen käyttöönotossa sekä opettelussa, saa siitä hyvän työkalun, jolla pystyy suorittamaan erilaisia testejä laajasti ja kattavasti.

3.4 Playwright

Playwright on melko uusi ilmainen avoimen lähdekoodin työkalu. Se julkaistiin tammikuussa 2020. Playwrightin on kehittänyt Microsoft ja sen tekijöihin kuuluu samoja henkilöitä, ketkä kehittivät Puppeteer nimisen testaustyökalun. Työkalut ovat samankaltaisia, mutta Playwright täydentää ominaisuuksia mitä Puppeteerista jäi uupumaan. Playwright on noussut todella nopeasti suosituksi automaatiotestaustyökaluksi. Sillä on melkein 2 000 000 viikoittaista latausta. Työkalu suorittaa testit oletuksena headless-testeinä ja testeihin on sisään rakennettu tarvittavat odotukset, joten niitä ei tarvitse itse määritellä. (Chaudhary 2023.) Headless-testissä ohjelma ajetaan taustalla, eikä ruudulle aukea mitään käyttöliittymää (Ganguly 2023). Tämä nopeuttaa testien suorittamista.

Playwright on suosittu sen kattavien ominaisuuksien ansiosta. Se tukee montaa eri ohjelmointikieltä kuten, JavaScript/TypeScript, Java, C# ja Pyhton. Playwright toimii myös kaikilla isoimmilla käyttöjärjestelmillä, Windows, macOS ja Linux. Sen asentaminen on helppoa ja nopeata sekä sillä pystyy testaamaan kaikkia käytetyimpiä selaimia, Chrome, Firefox, Edge ja Safari. Työkalu tukee myös iFramejen testaamista. Playwrightilla pystyy suorittamaan erilaisia testejä kuten, toiminnallisia-, E2E- ja API-testejä. Testejä pystyy ajamaan montaa samanaikaisesti ja testejä pystyy myös suorittamaan useilla eri

selaimilla samaan aikaan. Eräs iso etu työkalussa on sen ominaisuus, jonka avulla pystyy suorittamaan testejä useammilla välilehdillä ja ikkunoilla. Tämä on erittäin hyödyllinen ominaisuus, sillä joskus on tilanteita, jossa täytyy avata uusi ikkunan ja sen jälkeen palata takaisin edelliseen ikkunaan. Testien suorittaminen työkalulla on myös todella nopeaa. Playwrightissa on sisään rakennettu raportointi ominaisuus, jonka avulla pystyy tarkastelemaan testien tuloksia. Työkaluun pystyy myös tekemään omia kustomoituja raportteja sekä se tukee myös erilaisia debuggaus-vaihtoehtoja, jotka helpottavat koodaajaa. (GH 2023.)

Jokaisessa työkalussa on aina myös huonoja puolia. Playwrightilla ei pysty testaamaan natiiveja mobiilisovelluksia, mutta sillä pystyy emuloimaan mobiili näkymää verkkosivuilla. Eräs yksi vähäisempi huono puoli Playwrightissa on se, että sillä ei pysty testaamaan IE11 (Internet Explorer 11). Tosin IE on jo pääsääntöisesti koitettu korvata Edge-selaimella, mutta IE on silti käytössä joillakin edelleen. Vaikka Playwright on uusi työkalu ja tukee modernien nettisivujen testaamista hyvin, ei uutuus ole aina hyvästä. Asiat voivat muuttua paljon, kun sovellus kehittyy eteenpäin ja joitakin olennaisia ominaisuuksia saattaa vielä puuttua. Uudella työkalulla ei yleensä myöskään ole kerennyt kasvaa suurta yhteisöä vielä, joka voi vaikeuttaa ongelmatilanteiden ratkaisemista. Microsoft kuuntelee kylläkin palautetta ja päivittää työkalua säännöllisesti. (GH 2023.)

3.5 Katalon

Katalon julkaistiin tammikuussa 2015 ja se on Selenium sekä Appium pohjainen automaatiotestaustyökalu. Katalon käyttää Groovy (Java) ohjelmointikieltä, eikä se tue muita ohjelmointikieliä. Toisin kuin muut työkalut mitä olen esitellyt työssäni, Katalon ei ole avoimen lähdekoodin työkalu. Sillä on kuitenkin avoimeen lähdekoodin perustuva viitekehys Katalium. Tämä viitekehys on tosin tarkoitettu vain Seleneium- ja TestNG-pohjaisiin projekteihin. Katalon Store on myös avoimeen lähdekoodin perustuva, mutta se on tarkoitettu pelkästään lisäosien lataamiseen Katalon Studiota varten. (AltexSoft 2021.)

Katalon Studio on varsinainen työkalu mitä käytetään testien ajamiseen ja luomiseen. Katalonilta löytyy myös monia muita eri ohjelmistoja, kuten Katalon

TestOps, Katalon TestCloud, Katalon Runtime Engine, Katalon Recorder ja Ai Visual Testing. Katalonin työkalut ovat pääsääntöisesti ilmaisia, mutta maksua vastaan saa paljon lisäominaisuuksia. Maksullisella versiolla saa oikeuden esimerkiksi kaikkiin lisäosiin, käyttäjä määrää ei rajoiteta, kuukausittain saa enemmän testituloksia tehtyä, saa paremmat raportit ja analyysit testeistä sekä muita hyötyjä. (AltexSoft 2021.)

Katalon Runtime Engine on lisäosa, joka mahdollistaa testien aikatauluttamisen ja testien suorittamisen automaattisesti konsolitilassa sekä komentorivitilassa. Tämä lisäosa on maksumuurin takana. Katalon TestOps on web-pohjainen sovellus, joka on suunniteltu testien ja DevOpsin organisointiin. Työkalu luo reaaliaikaisia testiraportteja ja käyttäjät pystyvät seuraamaan niiden edistystä sekä laatua. Käyttäjät voivat myös priorisoida joitakin tärkeimpiä testejä, jos näin haluavat. Katalon Recorder on web-laajennus, jonka avulla pystyy tallentamaan käyttäjän toimintoja ja toistamaan ne sitten skripti muodossa. Katalon TestCloud avulla pystyy testaamaan eri käyttöjärjestelmiä ja selaimia pilvessä. Ai Visual Testing hyödyntää tekoälyä testauksen helpottamiseksi. Se auttaa esimerkiksi, visuaalisissa regressio testeissä, auttaa huomioimaan merkittäviä muutoksia käyttöliittymässä ja auttaa testien luonnissa avainsanojen avulla. (AltexSoft 2021.)

Katalonilla on paljon hyviä puolia. Se tukee kaikkia isoimpia käyttöjärjestelmiä, eli Windows, macOS ja Linux sekä kaikkia isoimpia selaimia: Chrome, Firefox, Edge, Safari ja IE. Katalonilla pystyy myös testaamaan iFrameja. Testejä pystyy suorittamaan monia erilaisia kuten, API-testaus, web-testaus, mobiilitestaus ja työpöytäsovellusten-testaus. Katalonin käyttöönotto on melko vaivatonta, sillä testejä pystyy luomaan käyttöliittymän avulla ja avainsanapohjaisesti. Testejä pystyy myös nauhoittamaan, joka alentaa käyttökynnystä edelleen, mutta testejä pystyy myös koodaamaan, jos haluaa. Katalon tekee ajetuista testeistä myös automaattisesti raportit. Työkalulla pystyy suorittamaan useita testejä samanaikaisesti, testejä pystyy suorittamaan myös useammalla selaimella ja välilehdellä samanaikaisesti. Katalonilla on myös paljon videoita ja hyvät dokumentaatiot, joiden avulla oppii käyttämään työkalua hyvin. (AltexSoft 2021.) Työkalu tukee myös automaattisia odotuksia testien suorittamisessa (Katalon Smart Wait: A New Way to Handle Web Loading Issues).

Yksi selkeä heikkous työkalussa on se, että Katalon tukee vain Groovy-ohjelmointikieltä. Jos käyttäjät eivät osaa sitä etukäteen ei heillä ole mahdollisuutta käyttää muita ohjelmointikieliä, jos he haluavat koodata testejä. Katalonin yhteisö ei ole yhtä iso verrattuna muihin vanhempiin työkaluihin, joten se rajoittaa tiedon etsimistä joissakin ongelma tilanteissa. Avoimen lähdekoodin puuttuminen rajoittaa kehittäjien määrää Katalonin yhteisössä. Kehittäjät eivät pysty tekemään omia versioita tai omia korjauksia Kataloniin, mikä voi olla heille syy miksi he eivät käytä sitä. Vaikka Katalon on valmis paketti, on sillä myös omat huonot puolensa. Tämä tarkoittaa, että jos haluaa käyttää, Katalonia on käyttäjä riippuvainen sen ekosysteemistä ja ei kunnolla pysty käyttämään muita työkaluja tai viitekehyksiä sen kanssa. (Sharma 2023.)

3.6 Työkalujen vertailu

Alla on taulukko, jossa esitellään tärkeimpiä ominaisuuksia mitä työkaluissa esiintyi. Taulukon (taulukko 1) tarkoituksena on tuoda työkalujen erot tiiviisti ja selkeästi esille. Taulukko helpotti myös omaa päätöksentekoa työkalun valinnassa.

Taulukko 1. Työkalujen vertailu

OMINAISUUS	**CYPRESS**	**SELENIUM**	**PLAYWRIGHT**	**KATALON**
JULKAISU VUOSI	2018	2004	2020	2015
ILMAINEN	Kyllä, mutta löytyy maksullisia ominaisuuksia	Kyllä	Kyllä	Kyllä, mutta löytyy maksullisia ominaisuuksia
OHJELMOINTIKEILI	JavaScript	JavaScript, Python, Ruby, Java, Kotlin, C#	JavaScript, Pyhton, Java, C#	Groovy (Java)
AVOIMEN LÄHDEKOODIN	Kyllä	Kyllä	Kyllä	Ei
TUETUT KÄYTTÖJÄRJESTELMÄT	Windows, Linux, macOS	Windows, Linux, macOS	Windows, Linux, macOS	Windows, Linux, macOS

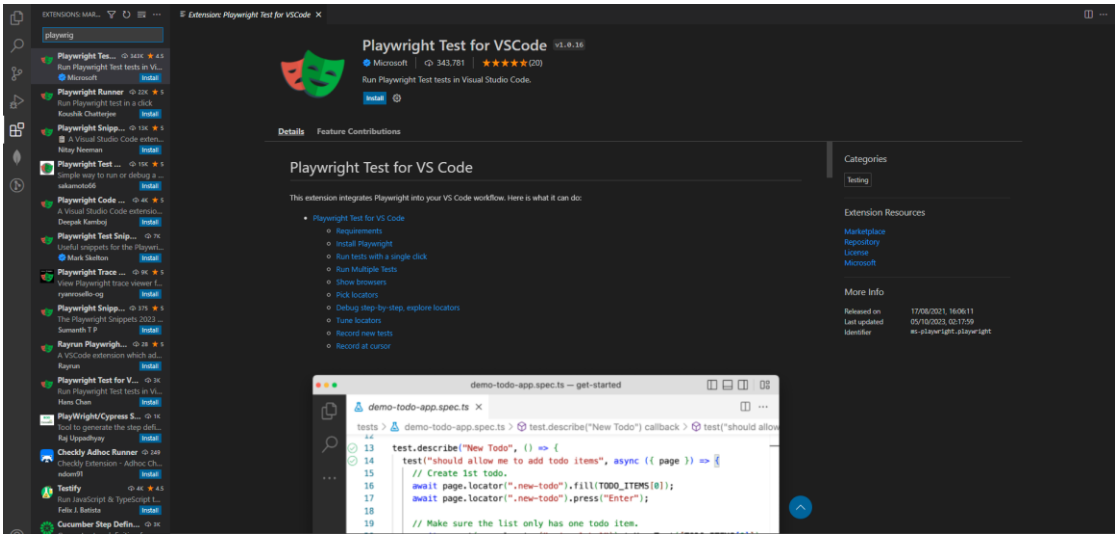
TUETUT SELAIMET	Chrome, Firefox, Edge, Safari (kokeellinen)	Chrome, Firefox, Edge, Safari, IE	Chrome, Firefox, Edge, Safari	Chrome, Firefox, Edge, Safari, IE
MOBIILI TESTAUS	Ei tue natiivi sovelluksia, pystyy emuloimaan selainnäkymää	Tukee natiivisovelluksien testaamista lisäosien avulla, pystyy emuloimaan selainnäkymää	Ei tue natiivi sovelluksia, pystyy emuloimaan selainnäkymää	Tukee natiivisovelluksien testaamista, pystyy emuloimaan selainnäkymää
RINNAKKAISTESTAUS	Mahdollista	Mahdollista	Mahdollista	Mahdollista
USEAN VÄLILEHDEN TESTAUS	Ei tuettu	Mahdollista	Mahdollista	Mahdollista
USEAN SELAIMEN TESTAUS	Ei tuettu	Mahdollista	Mahdollista	Mahdollista
AUTOMAATTINEN ODOTUS	Kyllä	Pystyy määrittämään erilaisia odotuksia testeihin	Kyllä	Kyllä
TESTIRAPPOTTI	Kyllä	Vaati lisäosan raportteja varten	Kyllä	Kyllä

Kaikki taulukossa (taulukko 1) olevat työkalut ovat varsin hyviä vaihtoehtoja E2E-testauksen automatisointiin. Selenium on todella käytetty ja siinä on paljon ominaisuuksia, mutta sen vanhuuden myös huomaa, kun vertaa uudempiin työkaluihin. Testien suorittaminen ei ole yhtä nopeaa ja se vaatii paljon lisäosia, jotta se toimii yhtä hyvin kuin muut työkalut. Katalonissa on myös paljon ominaisuuksia, mutta moni näistä ominaisuuksista on maksumuurin takana ja päätimme toimeksiantajan kanssa pysyä ilmaisissa työkaluissa. Cypress ja Playwright ovat molemmat erittäin hyviä työkaluja E2E-testaukseen, mutta Playwright on täysin ilmainen ja siinä on enemmän hyödyllisiä ominaisuuksia verrattuna Cyprekseen. Kysyin myös toimeksiantajan mielipidettä Cypressistä ja Playwrightista ja tulimme tulokseen, että Playwright sopii paremmin heidän käyttötarkoitukseensa. Käytin Playwrightia testiskriptien luomiseen.

4 TYÖKALUN KÄYTTÖÖNOTTO

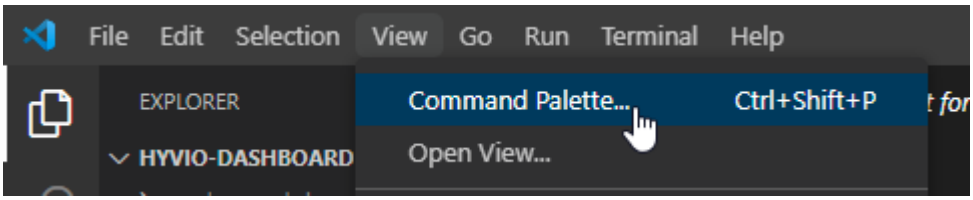
Playwrightin asennus projektiin onnistuu suosituilla paketinhallintajärjestelmillä kuten npm, yarn ja pnpm. Käyttämällä koodieditoria kuten esimerkiksi Visual Studio Code, asentaminen on todella helppoa. Haluttu projekti avataan Visual Studio Codessa (lyhennys VS Code) ja VS Codessa avataan uusi terminaalin, johon syötetään asennuskomento. Npm:ssä asennuskomento on seuraavanlainen: npm init playwright@latest. Tämä käynnistää asennuksen. Asennuksen alussa voi valita haluaako käyttää TypeScriptiä vai JavaScriptiä, oletuksena on TypeScript. Tämän jälkeen käyttäjältä kysytään kansion nimeä, johon testit tulevat menemään, oletuksena kansion nimi on tests. Jos projektissa on samanniminen kansio jo olemassa, nimetään uusi kansio nimellä e2e, ellei toisin määritellä. Asennuksen yhteydessä voi valita haluaako lisätä GitHub Actionsin, jonka avulla testit suoritetaan CI:ssä. Tämä tarkoittaa siis sitä, että testit ajetaan aina automaattisesti, kun tietty toiminto suoritetaan GitHubissa. Viimeiseksi käyttäjältä kysytään, asennetaanko kaikki selaimet mitä Playwright pystyy testaamaan, oletuksena kaikki selaimet ovat valittuina. Tämän jälkeen Playwright asentuu ja on käytettävissä projektissa. Playwrightin asetuksia pystyy muokkaamaan asennuksen jälkeen playwright.config-tiedostossa. (Playwright s.a.)

VS Codeen on olemassa myös laajennus, Playwright Test for VSCode, jonka avulla Playwrightin pystyy asentamaan. Laajennuksen asentamisen jälkeen saa käyttöönsä muutamia käteviä lisäominaisuuksia. Käytin itse tätä tapaa Playwrightin asentamiseen. Laajennuksen löytää, kun menee VS Coden laajennukset-osioon ja kirjoittaa hakukenttään ”Playwright”. Tämän jälkeen hakukentän alle tulee lista laajennuksista, joista tulee valita Playwright Test for VSCode, jonka on julkaissut Microsoft (kuva 2).



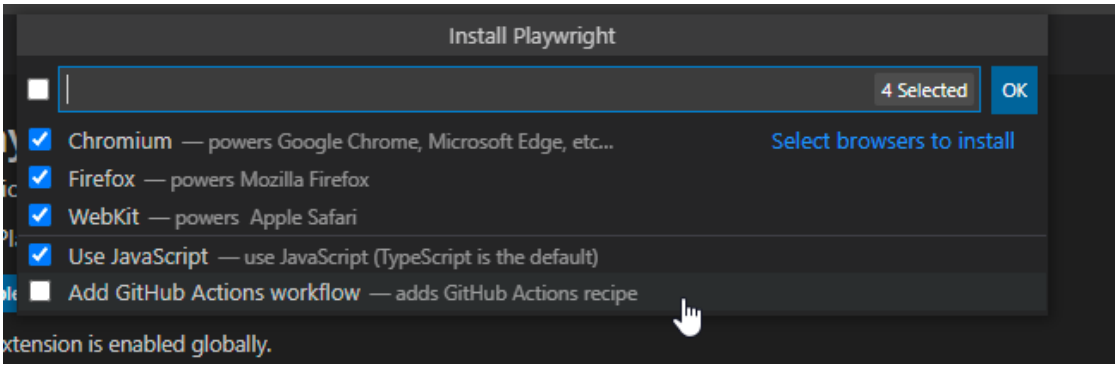
Kuva 2. Playwright Test for VSCode-laajennus

Asennus tapahtuu Install-napin avulla. Napin painamisen jälkeen asennuksen saa vietyä loppuun, kun avataan command palette VS Coden yläpalkista (kuva 3) ja kirjoitetaan auenneeseen tekstikenttään: Test: Install Playwright. Tämän jälkeen käyttäjältä kysytään samat vaihtoehdot mitä kysyttäisiin, jos työkalun asentaisi paketinhallintajärjestelmän avulla.



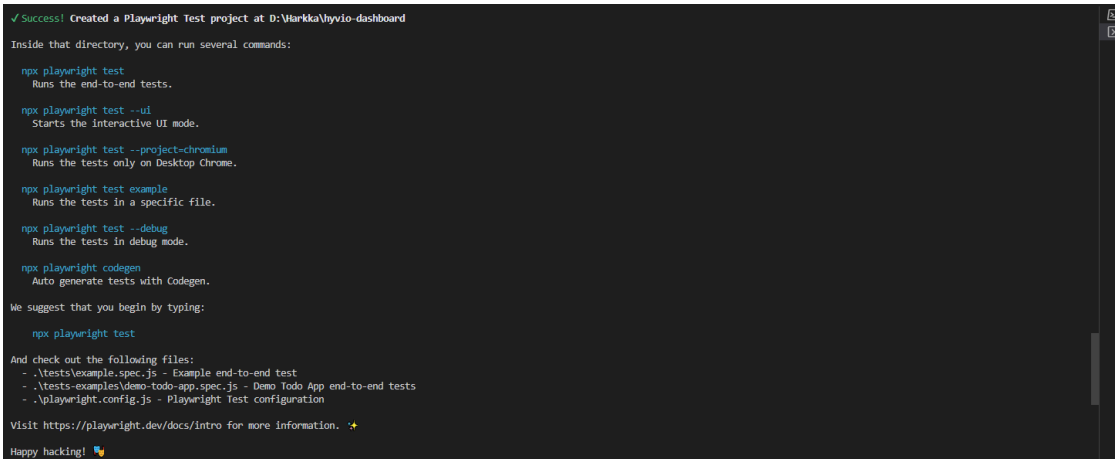
Kuva 3. Command palette.

Oletuksena ovat samat vaihtoehdot mitkä olivat myös toisessa asennustavassa (kuva 4). Itse muutin hieman niitä, sillä projekti, johon asensin Playwrightin käytti JavaScriptiä ja en halunnut ottaa GitHub Actionsia käyttöön. Kun olin tehnyt valinnat mitä halusin asentaa painoin OK-nappia ja asennus käynnistyi.



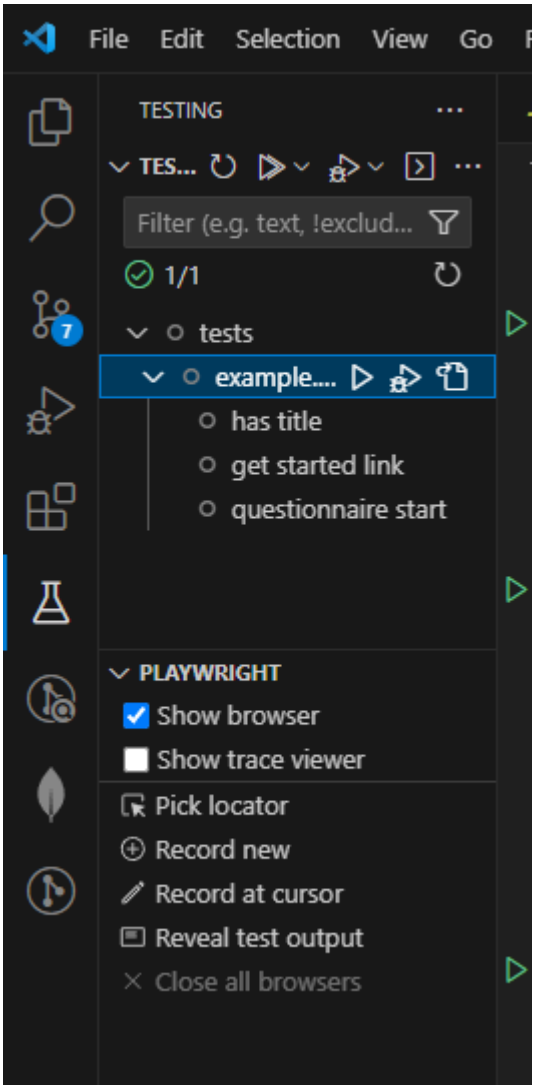
Kuva 4. Install Playwright

Asennus aukaisee uuden terminaalin, josta voi seurata asennuksen edistymistä. Asennuksen aikana Playwright asentaa kaiken tarpeellisen automaattisesti ja kun asennus on mennyt läpi (kuva 5) tulee näkyviin varmistusviesti asennuksen onnistumisesta sekä muutama hyödyllinen komento, jotka auttavat testien ajamisessa.



Kuva 5. Asennus onnistui

Huomasin asennuksen jälkeen pienen ongelman laajennuksen kanssa. Jostain syystä laajennus ei tunnistanut testejä, joten en pystynyt käyttämään laajennuksen testiajuria testien suorittamiseen. Terminaaliin kirjoitetut komennot kuitenkin toimivat, joten Playwrightin asennus oli onnistunut. Selviteltyäni asiaa päädyin päivittämään VS Coden, sillä minulla oli huomattavasti vanhempi versio siitä käytössä. VS Coden päivittämisen jälkeen ajoin Playwrightin asennuskomennon uudestaan ja laajennus alkoi toimimaan normaalisti (kuva 6).



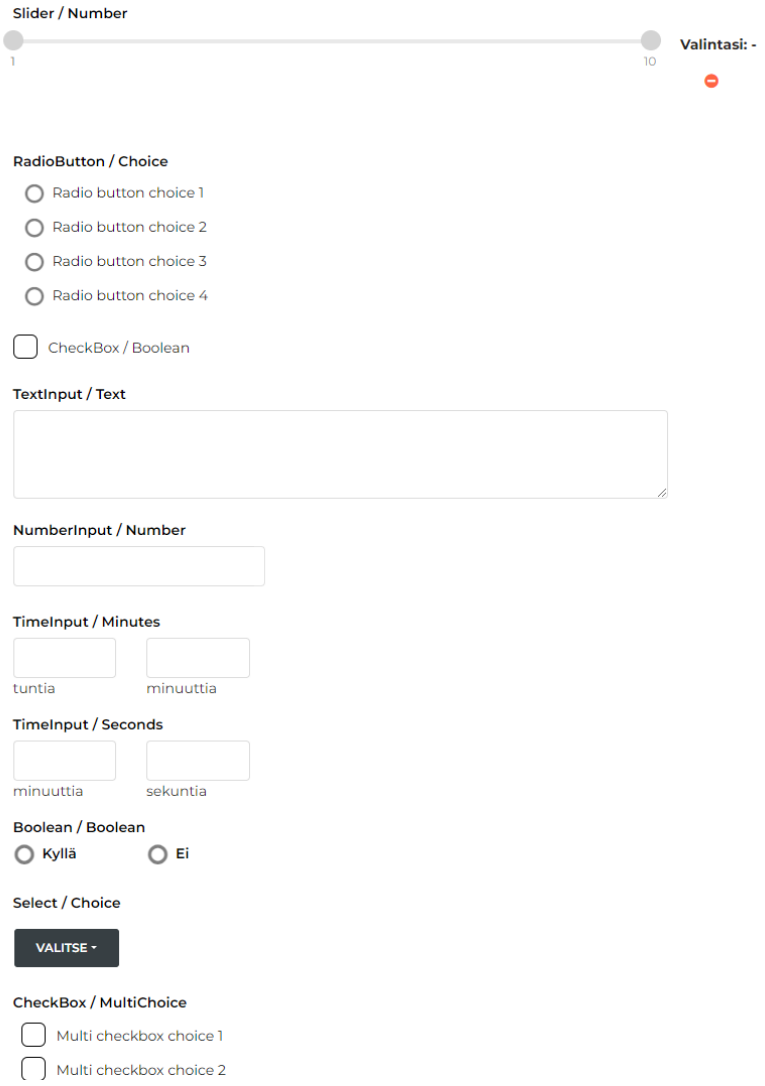
Kuva 6. Laajennus toimii

Varmistuin laajennuksen toimivuudesta ajamalla testit, jotka tulivat Playwrightin asennuksen yhteydessä (kuva 6). Loin myös yhden oman yksinkertaisen testin ja ajoin sen laajennuksen avulla myös. Kaikki toimi normaalisti, joten pystyin aloittamaan varsinaisten testien luomisen seuraavaksi.

4.1 Testien suunnittelu

Aloitin testien suunnittelun perehtymällä ensiksi testattavaan kyselyyn. Toimeksiantaja oli luonut testikyselyn, jossa oli kaikki eri vastaustyypit mitä heidän kyselyissään pystyi olemaan (kuva 7). Kyselystä löytyi erinäköisiä liukusäätimiä, teksti- ja numerokenttiä, monivalintakysymyksiä, vaihtoehtokysymyksiä ja kyselystä löytyi myös ei-pakollisia sekä pakollisia kysymyksiä. Kyselyssä oli myös napit sivunalareunassa, joista pystyi menemään alkuun, eteenpäin, taaksepäin, lähettämään ja tulostamaan kyselyn riippuen miten pitkällä

kyselyssä oltiin. Kyselyn täyttämisen ja lähettämisen jälkeen ilmestyi vielä tulossivu, josta näki omat vastauksensa, mahdolliset tulokset sekä palautteet mitä kyselyyn liittyi.



Kuva 7. Kyselyn vastaustyyppejä

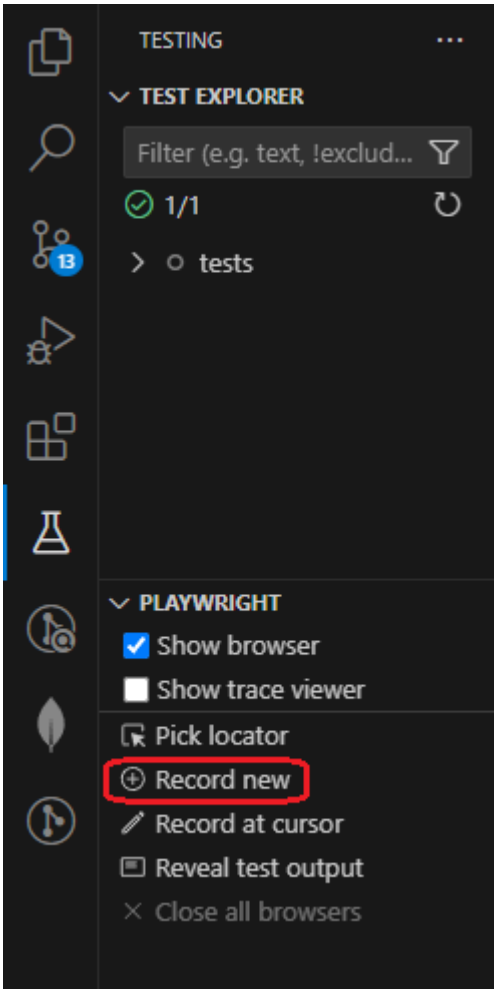
Testejä pystyisi tekemään monia, jos jokaista kohtaa haluaisi testata yksitellen, mutta E2E-testauksessa keskitytään koko kyselyn täyttöprosessiin. Samalla kun kyselyn täyttää kokonaan tulee myös todennettua, että suurin osa kyselyn kohdista toimii oikein. Toki on hyvä tehdä joitakin yksityiskohtaisempiakin testejä, kuten vastata vain pakollisiin kohtiin ja jättää vastaamatta pakollisiin kohtiin. On myös hyvä testata monivalintakohdissa eri vaihtoehtoja, vaihtelee vastauksia ja tyhjentää vastauksia, jos se on mahdollista. Tulisi myös testata, että numerokentät eivät hyväksy tekstiä tai muita virheellisiä syötteitä. Kyselyssä pystyi myös palaamaan aloitussivulla, jonka jälkeen kyselyyn jo vastatut kohdat tyhjenivät, kun aloitti kyselyn vastaamisen uudestaan.

Tätäkin ominaisuutta oli hyvä testata. Kyselyn kieltä pystyi vaihtamaan suomen ja englannin välillä, joten tein myös testin, joka varmisti kielen vaihdon onnistuneen. Testejä oli myös hyvä kokeilla useammalla eri selaimella ja mobiilinäkymässä, jotta varmistuttiin kyselyn toimivuudesta eri laitteilla sekä selaimilla.

Toimeksiantaja toivoi myös, että loisin testin, joka testaisi tulossivun tuloksia. Eli testi tarkastaisi kyselyssä käytettävän pistelaskurin antavan oikean tuloksen ja että käyttäjän vastaamat vastaukset täsmäisivät tulossivulla oleviin vastauksiin. Toimeksiantaja toivoi myös minun tekevän toisen testin, joka epäonnistuu eräästä bugista, joka kyselyssä oli. Tämä bugi liittyi ominaisuuteen, jonka avulla vastaaja pystyi ohittamaan kysymyksen, jos hän valitsi niin. Jos käyttäjä vastasi kysymykseen, mutta päätti jälkikäteen ohittaa kysymyksen eikä poistanut vastaustaan vastauskentästä, näkyi syötetty vastaus tarkistussivulla edelleen.

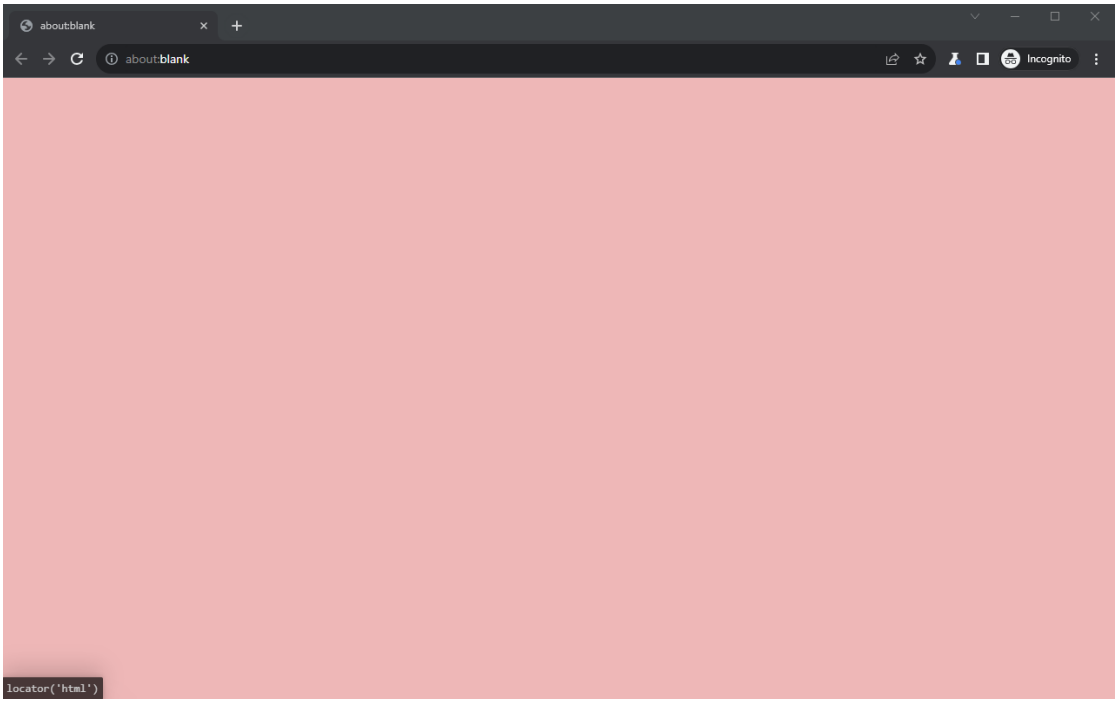
4.2 Testien luominen

Testien luominen on todella yksinkertaista etenkin, jos käyttää Playwrightin VSCode-laajennusta. Laajennuksen avulla pystyy nauhoittamaan omat toiminnot, mitä selaimessa tekee ja laajennus luo samaan aikaan tarvittavat koodit testiskriptiin, jotta testin pystyy toistamaan. Kun nauhoitusnappia (kuva 8) painaa, aukeaa uusi selainikkuna, jota laajennus nauhoittaa. Testejä pystyy myös tekemään itse kokonaan alustaloppuun pelkästään koodieditorissa, mutta laajennuksen avulla saa nopeasti luotua hyvän pohjan testiskripteille.



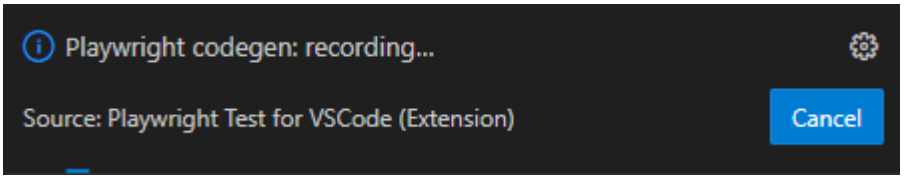
Kuva 8. Testin nauhoitus laajennuksella

Auenneeseen selainikkunaan (kuva 9) syötetään osoiteriville haluttu sivusto, jota halutaan testata. Tämän jälkeen voi alkaa suorittamaan haluttuja toimintoja kyseisellä sivustolla ja laajennus tallentaa tehdyt toiminnot sekä tekee niistä toimivat komennot testiskriptiin. Laajennuksen nauhoitusominaisuus ei ole täydellinen, sillä joidenkin elementtien kanssa voi tulla ongelmia, kun yrittää olla vuorovaikutuksessa niiden kanssa. Eräs tallainen elementti oli liukusäädin (ks. kuva 7 sivulla 26), sen vetäminen askelittain ei onnistunut nauhoituksen aikana. Liukusäätimestä pystyi kuitenkin valitsemaan ensimmäisen-, keskimmäisen- ja viimeisenkohdan. Huomasin myös, että pitää olla tarkkana mihin painaa nauhoituksen aikana, sillä joskus laajennus saattoi jättää kohdennuksen edelliseen kohtaan tai kohdentaa väärään kohtaan. Nämä ongelmat olivat korjattavissa jälkikäteen testiskriptissä, jonka nauhoitus oli luonut.



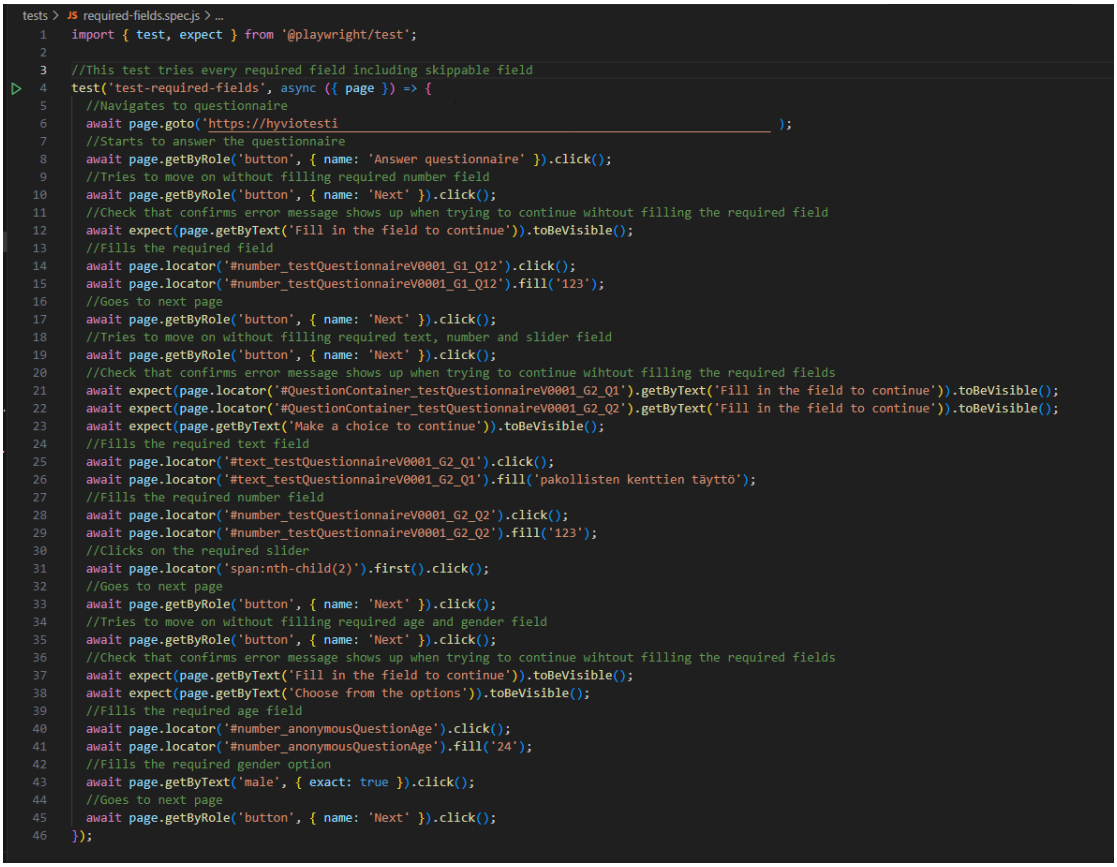
Kuva 9. Auennut selainikkuna

Nauhoituksen pystyy lopettamaan VSCoden ilmoituslaatikon peruuta-napista painamalla (kuva 10). Tämän jälkeen muodostunutta koodia pystyy muokkaamaan, jos sille on tarvetta. Riippuen kuinka monimutkainen ja laaja testitapaus on, se ei välttämättä tarvitse mitään muutoksia ja testiä pystyy käyttämään sellaisenaan.



Kuva 10. Nauhoituksen lopetus

Tässä esimerkki (kuva 11) eräästä onnistuvasta testistä, jonka suoritin kyselyyn. Testi varmistaa pakollisten kenttien toimivuuden. Aivan ensimmäiseksi laajennus tuo kaksi ominaisuutta Playwrightista, test ja expect. Test-ominaisuus määrittää Playwrightille, että tästä alkaa uusi testi. Test vaatii sisälleen testin nimen ja funktion. Testin nimen voi määritellä itse ja siitä kannattaa tehdä mahdollisimman kuvaava. Funktio saa parametrin page, jonka avulla Playwright pystyy olemaan vuorovaikutuksessa nettiselaimen kanssa eri näköisten toimintojen avulla. Expectiä testi ei käytä nauhoituksen aikana, mutta jälkikäteen voi lisätä erinäköisiä varmistuksia testiin sen avulla. Voidaan varmistua esim. siitä, että jokin otsikko sivustolla varmasti sisältää oikean tekstin.

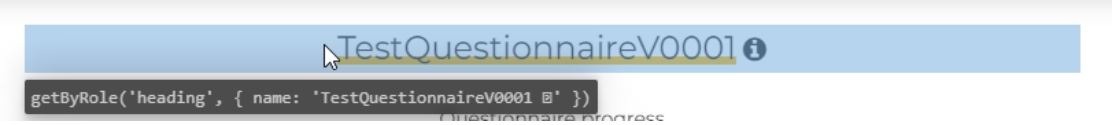


```
tests > JS required-fields.spec.js > ...
import { test, expect } from '@playwright/test';

//This test tries every required field including skippable field
test('test-required-fields', async ({ page }) => {
  //Navigates to questionnaire
  await page.goto('https://hyviotesti                                                    ');
  //Starts to answer the questionnaire
  await page.getByRole('button', { name: 'Answer questionnaire' }).click();
  //Tries to move on without filling required number field
  await page.getByRole('button', { name: 'Next' }).click();
  //Check that confirms error message shows up when trying to continue wihtout filling the required field
  await expect(page.getByText('Fill in the field to continue')).toBeVisible();
  //Fills the required field
  await page.locator('#number_testQuestionnaireV0001_G1_Q12').click();
  await page.locator('#number_testQuestionnaireV0001_G1_Q12').fill('123');
  //Goes to next page
  await page.getByRole('button', { name: 'Next' }).click();
  //Tries to move on without filling required text, number and slider field
  await page.getByRole('button', { name: 'Next' }).click();
  //Check that confirms error message shows up when trying to continue wihtout filling the required fields
  await expect(page.locator('#QuestionContainer_testQuestionnaireV0001_G2_Q1').getByText('Fill in the field to continue')).toBeVisible();
  await expect(page.locator('#QuestionContainer_testQuestionnaireV0001_G2_Q2').getByText('Fill in the field to continue')).toBeVisible();
  await expect(page.getByText('Make a choice to continue')).toBeVisible();
  //Fills the required text field
  await page.locator('#text_testQuestionnaireV0001_G2_Q1').click();
  await page.locator('#text_testQuestionnaireV0001_G2_Q1').fill('pakollisten kenttien täyttö');
  //Fills the required number field
  await page.locator('#number_testQuestionnaireV0001_G2_Q2').click();
  await page.locator('#number_testQuestionnaireV0001_G2_Q2').fill('123');
  //Clicks on the required slider
  await page.locator('span:nth-child(2)').first().click();
  //Goes to next page
  await page.getByRole('button', { name: 'Next' }).click();
  //Tries to move on without filling required age and gender field
  await page.getByRole('button', { name: 'Next' }).click();
  //Check that confirms error message shows up when trying to continue wihtout filling the required fields
  await expect(page.getByText('Fill in the field to continue')).toBeVisible();
  await expect(page.getByText('Choose from the options')).toBeVisible();
  //Fills the required age field
  await page.locator('#number_anonymousQuestionAge').click();
  await page.locator('#number_anonymousQuestionAge').fill('24');
  //Fills the required gender option
  await page.getByText('male', { exact: true }).click();
  //Goes to next page
  await page.getByRole('button', { name: 'Next' }).click();
});
```

Kuva 11. Nauhoitettu testiskripti

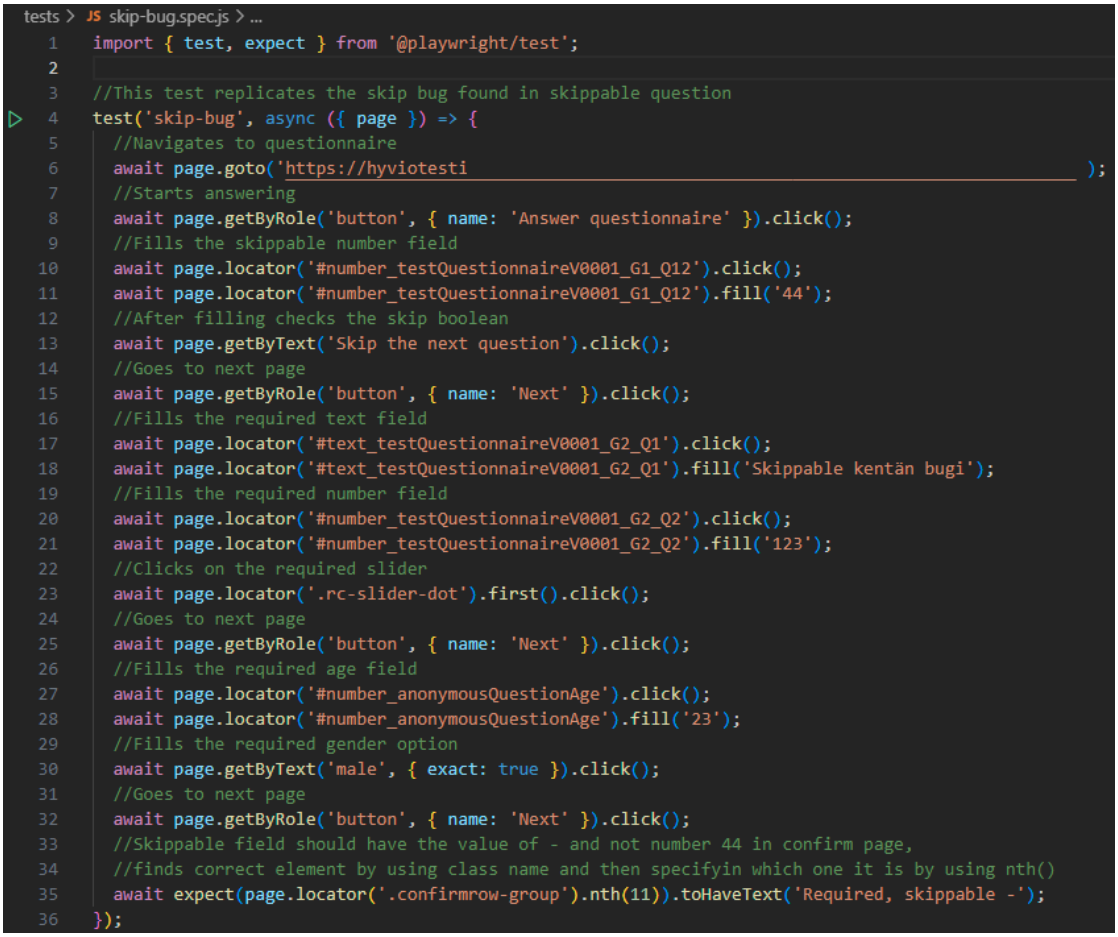
Testi alkaa menemällä annettuun osoitteeseen goto-funktion avulla. Kaikki testin toiminnot ovat asynkronisia. Tämä tarkoittaa, että testi odottaa automaattisesti elementtien latautuvan kokonaan ja testi odottaa myös, että aikaisemmat toiminnot on suoritettu, ennen kuin se jatkaa eteenpäin. Laajennus löytää oikeat elementit nauhoituksen aikana automaattisesti eri näköisten paikantimien (engl. locator) avulla (kuva 12). Laajennus käyttää useampaa eri paikanninta eri elementtien löytämiseen. Riippuen elementistä laajennus käyttää joko getByRole-, getByText- tai locator-funktiota. Paikantimien tarkoituksena on osoittaa testille mistä juuri kyseinen elementti löytyy, joten siksi laajennus käytti eri paikantimia eri elementtien löytämiseen. GetByRole-funktiota käytettään erilaisten klikattavien elementtien kanssa kuten, napit, valintaruudut ja linkit. GetByText-funktio etsii sivulta täsmäävän tekstipätkän jonkin elementin sisältä. Locator-funktiota puolestaan käytettään, jos elementin haluaa löytää esimerkiksi sen luokan, id:n tai elementin perusteella. Locator-funktiossa on useita eri vaihtoehtoja, minkä avulla elementin voi löytää.



Kuva 12. Esimerkki paikantimesta

Testissä painettiin useampaa eri elementtiä sekä täytettiin erinäköisiä teksti- ja numerokenttiä. Painallukset voidaan nähdä click-funktioina ja tekstin syötöt fill-funktioina. Lisäsin expectin avulla tarkastuksia, että virheviesti tuli näkyviin, jos käyttäjä yritti edetä testissä eteenpäin ilman, että hän täytti pakolliset kentät. Expect tarvitsi toimiakseen paikantimen ja funktion, joka suoritti tarkastuksen. Tässä testissä käytin toBeVisible-funktiota, joka tarkasti onko elementti näkyvissä vai ei. Se oli looginen tarkastus tehdä, sillä oletuksena virheviestit eivät ole näkyvissä, joten tämän tarkastuksen avulla voitiin varmistua virheviestien toimivuudesta. Testin kulkua pystyi selkeyttämään, jos lisäsi itse kommentteja komentojen yläpuolelle. Näin testistä tuli selkeämpi ja helposti ymmärrettävämpi.

Toimeksiantajan pyynnöstä tein myös testin, jonka tarkoituksena oli epäonnistua. Tämän testin (kuva 13) tarkoituksena oli testata kyselyssä olevaa virhettä. Virhe ilmeni ohitettavassa kysymyksessä, jos käyttäjä vastasi kysymykseen, mutta päätti ohittaa sen myöhemmin ilman, että poisti vastaustaan vastauskentästä. Vastaustentarkastussivulla ei pitäisi olla mitään tulosta kyseisen kysymyksen kohdalla, mutta jostain syystä syötetty vastaus näkyi silti tarkastussivulla.



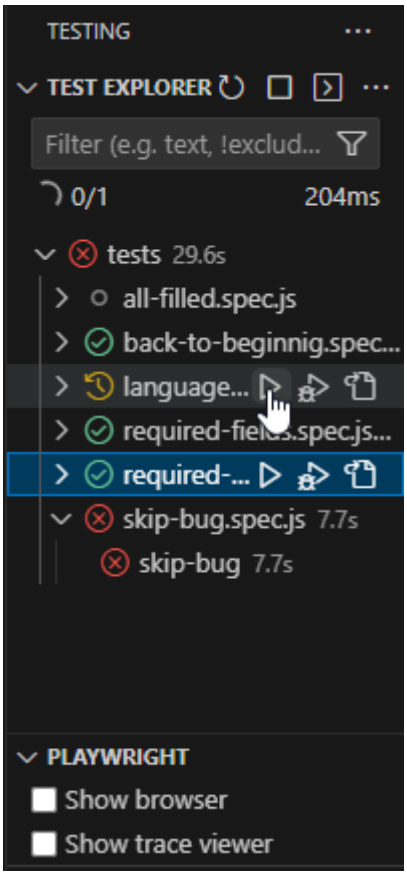
```
tests > JS skip-bug.spec.js > ...
1   import { test, expect } from '@playwright/test';
2
3   //This test replicates the skip bug found in skippable question
4   test('skip-bug', async ({ page }) => {
5     //Navigates to questionnaire
6     await page.goto('https://hyviotesti                                                    );
7     //Starts answering
8     await page.getByRole('button', { name: 'Answer questionnaire' }).click();
9     //Fills the skippable number field
10    await page.locator('#number_testQuestionnaireV0001_G1_Q12').click();
11    await page.locator('#number_testQuestionnaireV0001_G1_Q12').fill('44');
12    //After filling checks the skip boolean
13    await page.getByText('Skip the next question').click();
14    //Goes to next page
15    await page.getByRole('button', { name: 'Next' }).click();
16    //Fills the required text field
17    await page.locator('#text_testQuestionnaireV0001_G2_Q1').click();
18    await page.locator('#text_testQuestionnaireV0001_G2_Q1').fill('Skippable kentän bugi');
19    //Fills the required number field
20    await page.locator('#number_testQuestionnaireV0001_G2_Q2').click();
21    await page.locator('#number_testQuestionnaireV0001_G2_Q2').fill('123');
22    //Clicks on the required slider
23    await page.locator('.rc-slider-dot').first().click();
24    //Goes to next page
25    await page.getByRole('button', { name: 'Next' }).click();
26    //Fills the required age field
27    await page.locator('#number_anonymousQuestionAge').click();
28    await page.locator('#number_anonymousQuestionAge').fill('23');
29    //Fills the required gender option
30    await page.getByText('male', { exact: true }).click();
31    //Goes to next page
32    await page.getByRole('button', { name: 'Next' }).click();
33    //Skippable field should have the value of - and not number 44 in confirm page,
34    //finds correct element by using class name and then specifyin which one it is by using nth()
35    await expect(page.locator('.confirmrow-group').nth(11)).toHaveText('Required, skippable -');
36  });
```

Kuva 13. Epäonnistuva testiskripti

Skripti alkaa samalla tavalla kuin aikaisempikin testiskripti (ks. kuva 11 sivulla 30). Aluksi tuodaan tarvittavat toiminnot Playwrightista ja sen jälkeen määritellään uusi testi. Testi navigoi oikeaan osoitteeseen goto-funktiolla ja aloittaa elementtien painelun sekä täyttämisen. Kommenttien avulla (vihreä teksti) pystyy helpommin seuraamaan missä vaiheessa testi suorittaa mitäkin. Alussa testi täyttää ohitettavan kentän, mutta heti täytön jälkeen päättää ohittaa kysymyksen. Testi etenee kyselyssä eteenpäin ja täyttää pakolliset kentät, kunnes testi on edennyt tarkastussivulle, jossa expectin avulla tarkastetaan ohitettavan kentän tulos. Expect löytää oikean elementin sivulta sen luokan perusteella käyttäen locator-funktiota ja nth-funktiota. Nth-funktion avulla testi tietää monesko elementti tai paikannin on kyseessä, jos on monta samannimistä elementtiä tai paikanninta sivulla. Lopuksi expectille määritetään tarkastusfunktio toHaveText. Tähän funktioon syötetään teksti, joka oletetaan saavan tulokseksi. Tämä testi epäonnistuu virheen takia, sillä saatu tulos ei vastaa oletettua tulostekstiä, joka syötettiin tarkastusfunktioon.

4.3 Testien suorittaminen

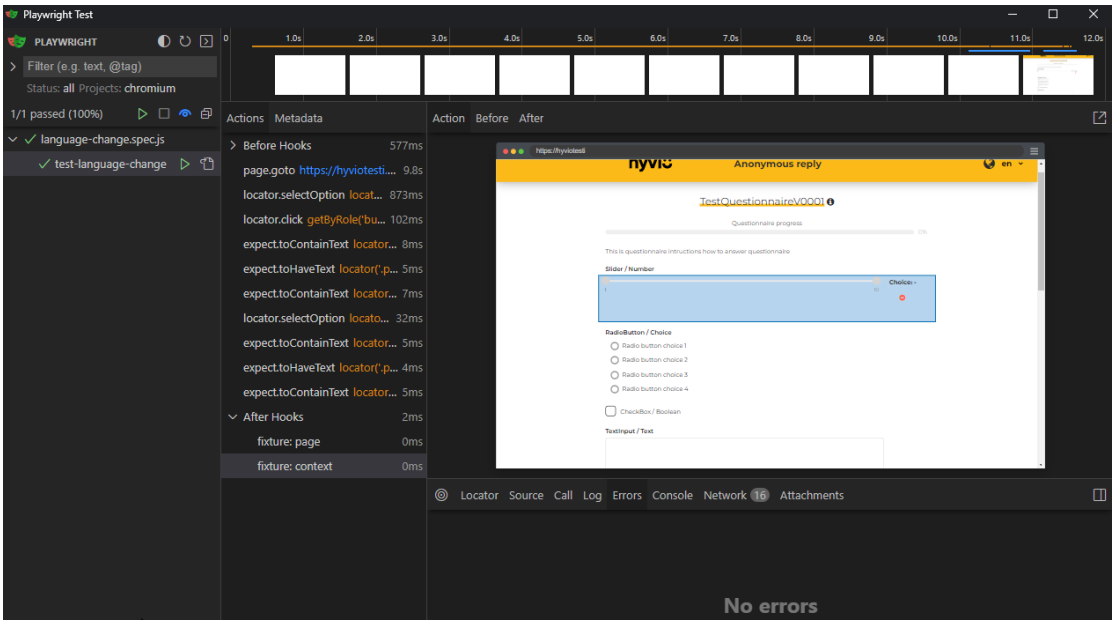
Testien ajamiseen löytyy useita eri vaihtoehtoja. Kaikista yksinkertaisin tapa ajaa testejä on käyttämällä Playwright Test for VSCode-laajennusta (kuva 14). Laajennuksen avulla ei tarvitse kirjoittaa mitään komentoja terminaaliin, vaan pelkkä napinpainallus riittää testien ajamiseen. Testejä pystyy ajamaan yksitellen painalla toistonappia halutun testitiedoston kohdalla. Kaikki testit voidaan ajaa samaan aikaan painalla laajennuksen yläreunassa olevaa toistonappia tai testikansion vieressä olevaa toistonappia. Läpi menneet testit näkyvät laajennuksessa vihreänä, epäonnistuneet testit punaisella ja keltaisella värillä merkitään testejä, jotka odottavat suorittamisen aloittamista. Testin suorittamista voi seurata selaimessa, jos valitsee vaihtoehdon show browser laajennuksesta. Oletuksena testit suoritetaan headless-tilassa, jolloin selainta ei aukea.



Kuva 14. Testien ajaminen laajennuksella

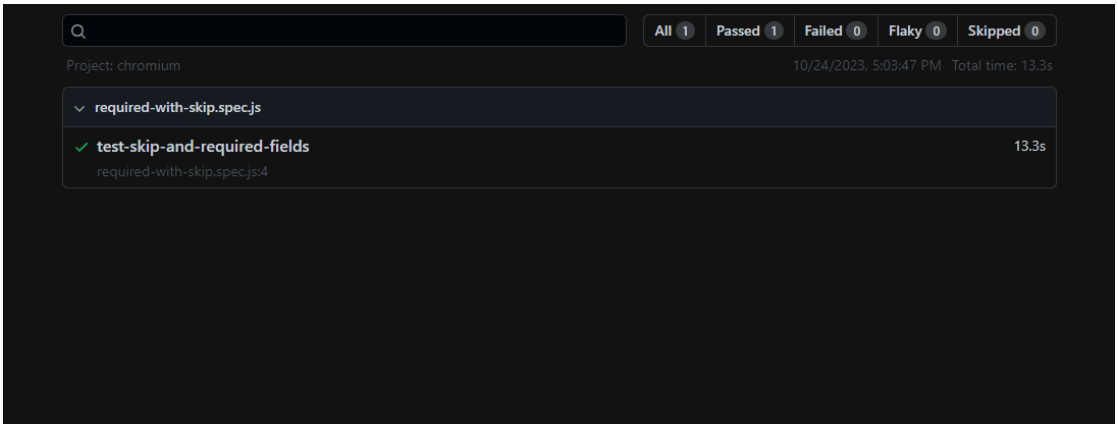
Playwrightissa on myös monia eri komentorivin käskyjä testien ajamiseen. Käskyillä pystyy suorittamaan samoja asioita mitä laajennuksellakin pystyy,

mutta käskyjen avulla voi suorittaa myös enemmän asioita verrattuna laajennukseen. Kaksi isointa asiaa mitä laajennuksella ei pysty suorittamaan on, testien ajaminen UI-tilassa ja testiraporttien avaaminen. UI-tilassa (kuva 15) pystyy käymää testin jokaisen vaiheen läpi ja näkemään, mitä tapahtuu ennen kutakin vaihetta, mitä tapahtui vaiheen aikana sekä vaiheen jälkeen. UI-tila on erittäin kätevä, sillä se helpottaa virheiden löytämistä ja korjaamista sekä tarjoaa muitakin ominaisuuksia kuten paikantimien valitsimen. UI-tilan saa auki käskyllä npx playwright test –ui.



Kuva 15. UI-tila

Testien ajamisen jälkeen testiraportin saa auki käskyllä npx playwright show-report. Tämä aukaisee uuden välilehden selaimeen, jossa näkyy kaikki ajetut testit (kuva 16). Jokaisen testin pystyy avaamaan ja silloin raportti näyttää lisätietoja testin ajosta. Raportti kertoo esim. kuinka kauan testin eri vaiheet kestivät ja testin kokonaissuoritusajan sekä millä selaimella tai laitteella testi suoritettiin. Jos ajettuja testejä on paljon, pystyy raporttinäkymässä hakemaan tai rajoittamaan tuloksia myös.



Kuva 16. Testiraportti näkymä

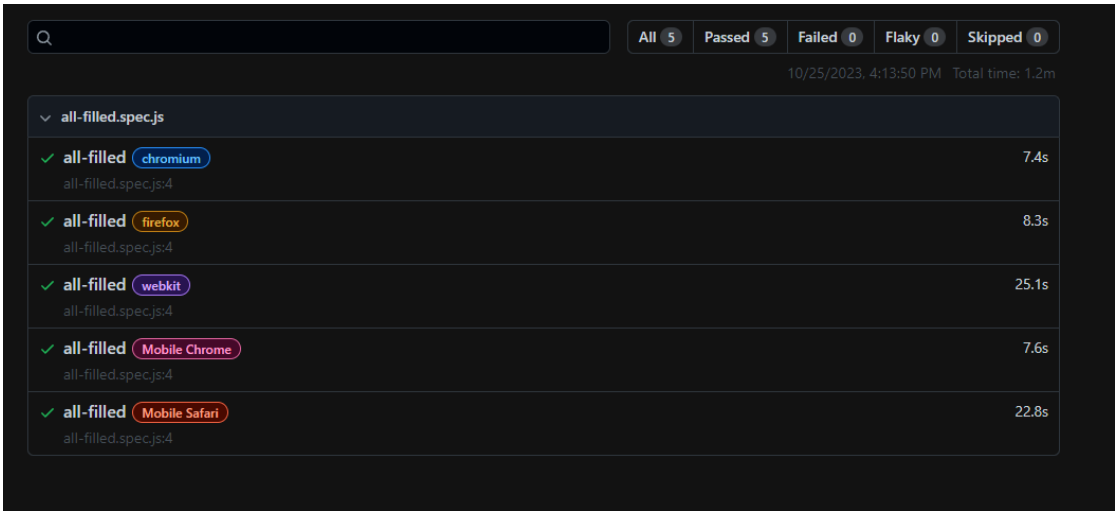
Muitakin käskyjä mitä laajennuksella ei pysty suorittamaan, mutta komentorivissä pystyy, on jonkin verran. Komentorivin avulla pystyy suorittamaan esim. testejä useasta eri kansiosta ja testejä, jotka sisältävät määritetyn sanan. Komentorivin avulla voi valita tietyt selaimet ja laitteet millä testit suoritetaan sekä komentorivin avulla voi myös suorittaa testejä debuggaus-tilassa. Debuggaus-tilassa pystyy käymään testin askelittain läpi ja löytämään virhekohdat testissä.

4.4 Testauksen tuloksia

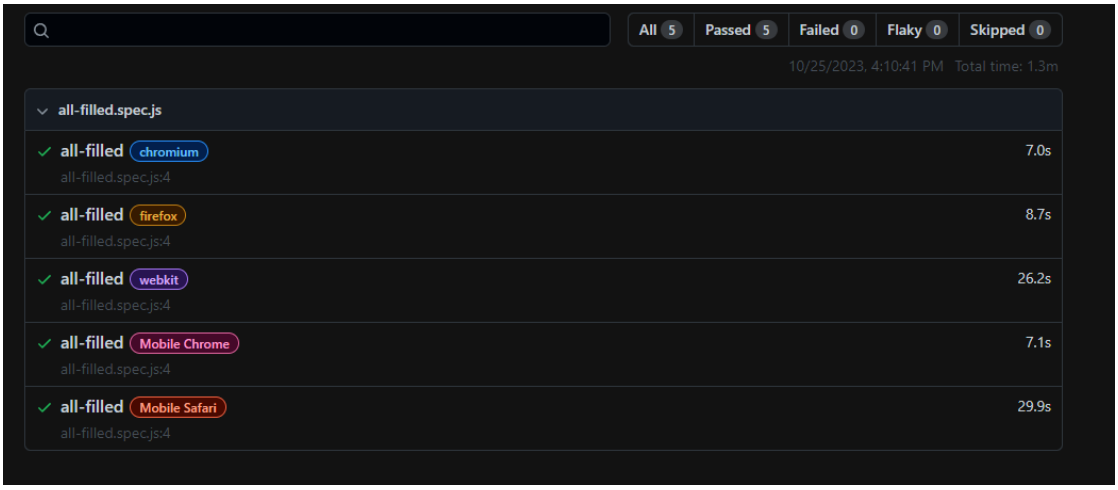
Tehtyäni kaikki testitapaukset, ajoin ne vielä kertaalleen läpi kaikilla selaimilla ja kahdella eri mobiilinäkymällä. Halusin varmistua, että kaikki testit varmasti onnistuivat ilman virheitä. Suoritin kaikki testit myös kahdessa eri tilassa, headed- ja headless-tilassa. Headed-tilassa huomasin ongelman Safari-selaimen mobiili näkymää testatessa. Jostain syystä se ei onnistunut suorittamaan testiä loppuun, kun ajoi kaikki testit samaan aikaan. Testin ajaminen headless-tilassa ei tuottanut ongelmia, testi onnistui myös, jos sen ajoi yksittäin headed-tilassa. En tiedä miksi testi ei onnistunut, kun ajoi kaikki testit samaan aikaan, mutta muulloin onnistui. Koitin ajaa kaikki testit useampaan kertaan headed-tilassa ja sain eri kerroilla eri virheen eri kohdissa. Päädyin lopputulokseen, että virheet mahdollisesti johtuivat Playwrightin webkit-selaimen emuloinnista ja sen hitaudesta. Kaikki muut testit onnistuivat niin kuin niiden kuuluikin.

Halusin nähdä, oliko headless- ja headed-tilalla suuria eroja suoritus nopeuksissa. Testasin nopeuseroja kaikista suurimmalla yksittäisellä testitapauksella.

Kuvista 17 ja 18 nähdään, että headless- ja headed-tilojen suoritusnopeuksissa oli melkein kahdeksan sekunnin ero. Testi suoritettiin kaikilla selaimilla ja kahdella eri mobiilinäkymällä. Vaikka testejä ei suoritettu montaa, pystyi silti huomaamaan headless-tilan hyödyn testien ajossa. Jos testitapauksia olisi enemmän ja ne olisivat laajempia, huomaisi headless-tilan nopeuden vielä paremmin. Testejä kannattaakin siis suorittaa headless-tilassa, ellei jostain syystä halua nähdä testin kulkua selaimessa. Suosittelisin tässä tapauksessa käyttämään Playwrightin UI-tilaa. Tässä tilassa pystyy tarkastelemaan testin kulkua paljon tarkemmin verrattuna headed-tilaan.



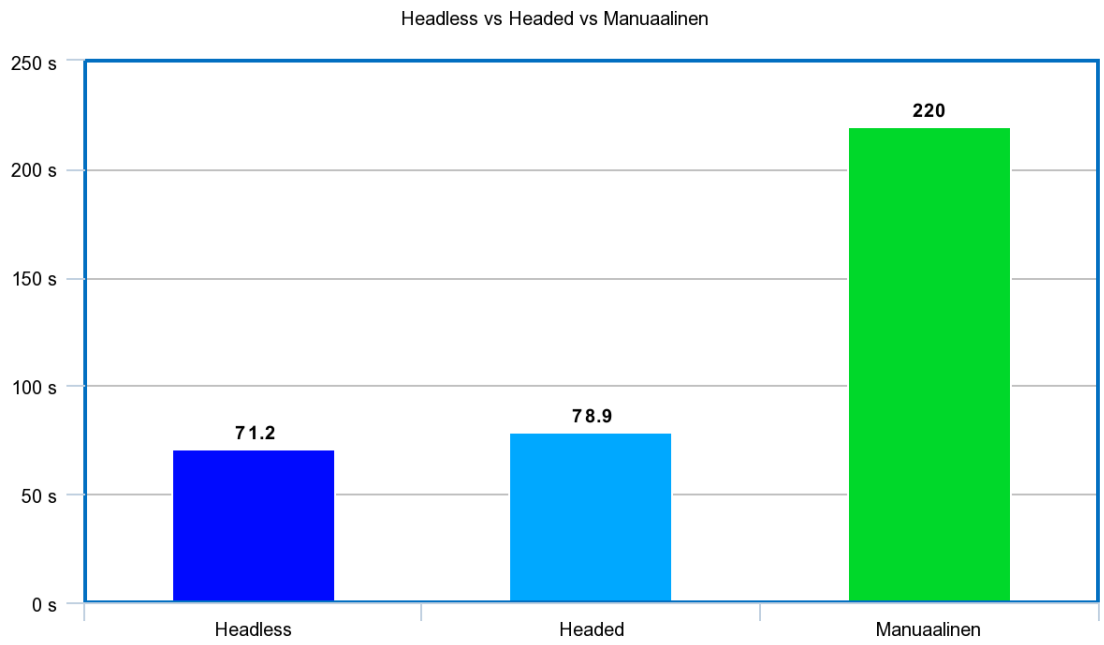
Kuva 17. Headless-tila



Kuva 18. Headed-tila

Suoritin samankaltaisen testitapauksen kuin kuvissa 17 ja 18, myös manuaalisesti yhdellä selaimella. Ajastin paljonko minulla kului aikaa avata selain, navigoida kyselyyn ja täyttää kysely. Sain ajaksi noin 40 sekuntia. Tässä pitää ottaa huomioon, että kysely oli jo ennestään tuttu minulle. Vastasin jokaiseen

kohtaan mahdollisimman nopeasti, joten en täyttänyt kyselyä tismalleen samalla tavalla kuin testiskripti. Loin pylväsdiagrammin (kuva 19) kuvastamaan eri testaustyylien nopeuseroja.



Kuva 19. Suoritusnopeuksien vertailua.

Kuvassa 19 nähdään, että laitoin manuaalisen-pylvään arvoksi 220 sekuntia. En suorittanut kyselyjä neljästi uudestaan eri selaimilla ja mobiili näkymillä, vaan kerroin ajastetun aikani viidellä sekä lisäsin siihen hieman lisäaikaa kompensoimaan eri selainten aukaisemista ja mobiilinäkymien emuloimista. Aika ei ole siis tarkka, mutta antaa silti hyvän kuvan siitä, kuinka paljon nopeampaa testaus on suorittaa automatisoidusti. Mitä enemmän ja mitä laajempia testitapauksia on, sitä suurempi hyöty automatisoiduista testeistä myös on.

5 PÄÄTÄNTÖ

Ohjelmistotestaus on hyvin olennainen osa mitä tahansa ohjelmiston elinkaarenvaihetta. Olipa ohjelmisto vasta kehitysvaiheessa tai jo ylläpitovaiheessa, löytyy jokaiselle vaiheelle ja kohdalle jokin testaustyyppi, jolla testata ohjelmiston toimivuutta. Ohjelmistotestauksen automatisoinnilla pystytään vapauttamaan työntekijä tekemään muita töitä siksi aikaa, kun testi suoritetaan. Testien automatisointi poistaa myös inhimillisten virheiden mahdollisuuden, etenkin jos testattavaa on paljon ja toistojen määrä on suuri.

E2E-testauksessa varmistutaan, että ohjelmisto tai sen jokin osa toimii alusta loppuun oikein, niin kuin oletettiin. Usein tämänkaltaisia testejä suoritetaan manuaalisesti, sillä testitapaukset voivat olla monimutkaisia ja helposti muuttuvia. Testejä pystyy kuitenkin automatisoimaan ja hyvin tehtyinä ne säästävät paljon aikaa sekä vaivaa, sillä E2E-testien suorittaminen manuaalisesti on aikaa vievää työtä. Luotuja testiskriptejä täytyy ylläpitää, jotta testit toimivat jatkossakin, kun ohjelmistoa jatkokehitetään tai päivitetään.

Työssä tutkittiin E2E-testauksen automatisoinnintyökaluja ja vertailtiin niitä keskenään. Vertailun tavoitteena oli antaa toimeksiantajalle tietoa käytetyimmistä ja suosituimmista työkaluista tällä hetkellä sekä helpottaa työhön käytettävän työkalun valitsemista. Vertailu säästi toimeksiantajan aikaa, kun heidän ei tarvinnut tehdä selvitystyötä alusta loppuun itse ja sain itsekin samalla tietoa tämänhetkisistä työkaluista. Keskustelin toimeksiantajan kanssa työssä esitellyistä työkaluista ja päädyimme lopulta käyttämään Playwrightia, sillä se sopi parhaiten työhön määriteltyyn testaustarkoitukseen.

Kehittämistyöosuudessa luotiin varsinaiset testiskriptit käyttäen Playwrightia. Skriptien tarkoituksena oli nopeuttaa testaamisprosessia toimeksiantajan Hyviö-sovelluksessa käytettävien kyselyiden täyttämisessä. Ohjelmistotestaus ei ollut kovin tuttua minulle aiemmin. Olin käynyt koulussa yhden kurssin, joka käsitteli ohjelmistotestausta, mutta sen enempää kokemusta minulla ei ollut aiheesta. Aihe oli kuitenkin jäänyt kiinnostamaan minua ja olin erittäin iloinen, että pääsin tutustumaan siihen enemmän opinnäytetyön parissa. Playwrightin käyttö oli melko helppoa loppupeleissä, sillä JavaScript oli minulle jo tuttu ohjelmointikieli opintojeni ansiosta ja Playwrightin syntaksi oli hyvin yksinkertaista ymmärtää.

Teoriaosuudessa opittujen tietojen avulla oli helppo suunnitella järkeviä- ja realistisia testejä. Onnistuin mielestäni testien luonnissa hyvin, sillä testit testaavat kyselyn eri ominaisuudet ja kohdat laajasti sekä nopeutti testaamisprosessia huomattavasti manuaaliseen testaamiseen verrattuna. Testejä pystyisi jatkokehittämään testaamaan tarkemmin eri kohtia esim. kokeilemalla vielä enemmän erilaisia kirjaimia, numeroita, erikoismerkkejä ja erikielen kirjaimia. Joitakin paikantimia pystyisi mahdollisesti tarkentamaan paremmaksi, mutta toisaalta tämän kaltaisia muutoksia joutuu kuitenkin tekemään, kun kyselyä

jatkokehitetään tai päivitetään. Eräs jatkokehitysidea olisi myös tehdä testejä muihinkin Hyviö-sovelluksen osiin esim. voisi testata Hyviöön kirjautumista, jonka jälkeen voisi navigoida johonkin testiin ja vastata siihen. Tässä tulisi testattua laajemmin eri toiminnallisuuksia ja todennettua sovelluksen toimivan kokonaisuutena.

Kaiken kaikkiaan työ meni mielestäni hyvin ja pysyin hyvin aikataulussakin. Aihe oli erittäin mielenkiintoinen ja se helpotti työhön paneutumista erittäin paljon. Aiheen vierasperäisyys ei oikeastaan haitannut tekemistä, toki muutamat kohdat testien luonnissa aiheutti hieman mietintää ja ongelman ratkaisua, mutta se on luonnollista. Yksi asia, joka jäi hieman mietityttämään, oli virhe, joka esiintyi, kun ajoi kaikki testit kerrallaan headed-tilassa. Jostain syystä Safari-selaimen mobiilinäkymässä suoritettu testi epäonnistui tällöin. En löytänyt syyhyn mitään järkevää selitystä, enkä myöskään saanut korjattua virhettä, vaikka yritin muutella testiä. Voin vain olettaa, että se on jonkinlainen bugi Playwrightin emulaattorissa tai testiajurissa. Muutoin olen tyytyväinen testeihin ja teoriaosuuteen. Koen, että opin työnaikana paljon uutta ja hyödyllistä tietoa sekä uusia taitoja.

LÄHTEET

AltexSoft. 2021. The Good and the Bad of Katalon Studio Automation Testing Tool. Blogi. Päivitetty 15.06.2021. Saatavissa: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-katalon-studio-automation-testing-tool/ [viitattu 27.09.2023].

Bose, S. 18.6.2023a. UI Testing: A Detailed Guide. WWW-dokumentti Saatavissa: https://www.browserstack.com/guide/ui-testing-guide [viitattu 16.09.2023].

BrowserStack s.a. Selenium Testing. WWW-dokumentti. Saatavissa: https://www.browserstack.com/selenium [viitattu 25.09.2023].

Chaudhary, A 2023. A Brief & Quick Tutorial On Playwright Testing Tool – Devstringx. Blogi. Saatavissa: https://www.devstringx.com/playwright-tool [viitattu 26.09.2023].

Cummings-John, R. 2020. What is automation testing? Blogi. Saatavissa: https://www.globalapptesting.com/blog/what-is-automation-testing [viitattu 17.09.2023].

Cypress s.a. Loved by OSS, trusted by Enterprise. WWW-dokumentti. Saatavissa: https://www.cypress.io/ [viitattu 24.09.2023].

Doshi, K. 22.3.2023. Types of Testing: Different Types of Software Testing in Detail. WWW-dokumentti. Saatavissa: https://www.browserstack.com/guide/types-of-testing [viitattu 16.09.2023].

Felice, S. 2023 Unit testing for NodeJS using Mocha and Chai. WWW-dokumentti. Saatavisa: https://www.browserstack.com/guide/unit-testing-for-nodejs-using-mocha-and-chai [viitattu 24.09.2023].

Ganguly, S. 27.08.2023. What is Headless Browser and Headless Browser Testing? WWW-dokumentti. Saatavissa: https://www.browserstack.com/guide/what-is-headless-browser-testing [viitattu 26.09.2023].

GH. 2023. Playwright Automation Framework: Tutorial WWW-dokumentti. Saatavissa: https://www.browserstack.com/guide/playwright-tutorial [viitauttu 26.09.2023].

Katalon Smart Wait: A New Way to Handle Web Loading Issues s.a. Katalon. Blogi. Saatavissa: https://katalon.com/resources-center/blog/handle-selenium-wait [viitattu 04.10.2023].

Kasurinen, J. P. 2017. Ohjelmistotestauksen käsikirja. Docendo Oy. E-kirja. Saatavissa: https://payhip.com/b/VgAJ [viitattu 15.09.2023].

Kinsbruner, E. 2021 What is Cypress Testing Automation? What It Is and How to Get Started. Blogi. Saatavissa: https://www.perfecto.io/blog/cypress-testing [viitattu 24.09.2023].

Perfecto. 27.06.2022. What's New in Cypress 10? Everything You Need to Know. Blogi. Saatavissa: https://www.perfecto.io/blog/cypress-10 [viitattu 24.09.2023].

Playwright s.a. Installation. WWW-dokumentti. Saatavissa: https://playwright.dev/docs/intro [viitattu 11.10.2023].

RTIH. 6.1.2023. End-to-end testing process explained: manual vs. automated testing. WWW-dokumentti. Saatavissa: https://retailtechinnovation-hub.com/home/2023/1/5/end-to-end-testing-process-explained-manual-vs-automated-testing [viitattu 20.09.2023].

Gillis, A. 9.3.2023. end-to-end testing. WWW-dokumentti. Päivitetty 3.2023. Saatavissa: https://www.techtarget.com/searchsoftwarequality/definition/End-to-end-testing [viitattu 06.09.2023].

Sharma, Y. 21.09.2023 Katalon Studio for API Testing: Learn All About this Automation Testing Tool. WWW-dokumentti. Saatavissa: https://www.careervira.com/advice/learn-advice/katalon-studio-for-api-testing-learn-all-about-this-automation-testing-tool [viitattu 27.09.2023].

Smartbear s.a. What is Automated Testing. WWW-dokumentti. Saatavissa: https://smartbear.com/learn/automated-testing/what-is-automated-testing/ [viitattu 17.09.2023].

Tasanto, T. 4.1.2018. Ohjelmistotestaus parantaa laatua. Blogi. Saatavissa: https://www.atrsoft.com/teknologiat/ohjelmistotestaus-parantaa-laatua [viitattu 15.09.2023].

Testim. 17.12.2020. UI Testing: A Beginner's Guide With Checklist and Examples. Blogi. Saatavissa: https://www.testim.io/blog/user-interface-testing/ [viitattu 16.09.2023].

Testim. 24.6.2022b. What Is End to End Testing? A Helpful Introductory Guide Blogi. Saatavissa: https://www.testim.io/blog/end-to-end-testing-guide/ [viitattu 16.09.2023].

Testim. 09.6.2022a. Test Automation Tool: Definition and 5 Best Ones. Blogi. Saatavissa: https://www.testim.io/blog/what-is-a-test-automation-tool/ [viitattu 19.09.2023].

TestingXperts. 27.8.2020. Functional Testing – An Informative Guide for Beginners Blogi. Päivitetty 22.7.2022 Saatavissa: https://www.testing-xperts.com/blog/functional-testing [viitattu 16.09.2023].

Types of Automation Testing: A Beginner's Guide. Katalon s.a. Blogi. Saatavissa: https://katalon.com/resources-center/blog/automation-testing-types [viitattu 19.09.2023].

Unadkat, J. 14.02.2023. Cypress vs Selenium: Key Differences. WWW-dokumentti. Saatavissa: https://www.browserstack.com/guide/cypress-vs-selenium [viitattu 24.09.2023].

What is Manual Testing? A Comprehensive Guide (With Examples) s.a. Katalon. Blogi. Saatavissa: https://katalon.com/resources-center/blog/manual-testing [viitattu 19.09.2023].

KUVALUETTELO