Tampereen yliopisto

Mikko Pirhonen

WEB-SOVELLUSTEN PÄÄSTÄ PÄÄHÄN -TESTAUS

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastaja: Mikael Filppula
Toukokuu 2021

TIIVISTELMÄ

Mikko Pirhonen: Web-sovellusten päästä päähän -testaus
Kandidaatintyö
Tampereen yliopisto
Tietotekniikan kandidaatin tutkinto-ohjelma
Toukokuu 2021

Web-sovelluksen toimintaympäristö on hyvin laaja. Web-sovellusten toimintaan vaikuttavat lukuisat web-standardit ja teknologiat, jotka kehittyvät koko ajan. Web-sovellukset ovat usein myös hajautettuja, eli sovelluslogiikka suoritetaan usealla laitteella. Tässä laajassa ja muuttuvassa ympäristössä web-sovelluksille erityisen tärkeää on kokonaisuuden toimivuus. Kokonaisuuden toimivuutta voidaan varmistaa automatisoidulla päästä päähän -testauksella. Päästä päähän -testauksessa web-sovellusta testataan ohjaamalla selainta käyttämään sovellusta samalla tavalla kuin loppukäyttäjä ja varmistamalla sovelluksen oikeellinen toiminta. Sillä pyritään siis varmistamaan sovelluksen oikea toiminta varsinaisessa toimintaympäristössään, kokonaisena ohjelmana.

Päästä päähän -testaukseen löytyy useita työkaluja, joilla automatisoidaan selaimen toimintaa ja varmistetaan web-sovelluksen toimivan halutulla tavalla. Testauskehysten perusteellisin ero on siinä, miten ne ohjaavat selaimen toimintaa. Tapa, jolla testauskehys ohjaa selaimia, usein määrittää myös testauskehyksen muita ominaisuuksia.

Tämän tutkielman tarkoituksena on selvittää päästä päähän -testaukseen tarkoitettujen testauskehysten vahvuudet ja heikkoudet, jotta testauskehyksen valinta uuteen web-sovellukseen helpottuisi. Verrattaviksi testauskehyksiksi valittiin neljä toteutkseltaan tai testien luontitavaltaan eroavaa testauskehystä: Selenium IDE, Selenium WebDriver, Cypress sekä Playwright. Testauskehyksiä verrattiin esimerkiksi luotettavuuden, joustavuuden ja nopeuden kannalta. Vertailun avuksi kehitettiin testauskehyksillä lyhyet esimerkkitestit web-sovellukselle, jonka jälkeen kehitysprosessia arvioitiin kehysten välillä.

Vertailun tuloksena kullekkin testauskehykselle löydettiin vahvuuksia ja heikkouksia. Vanhempien testauskehysten eduksi havaittiin joustavuus, sillä niiden ohjelmointikieli- ja selaintuki oli huomattavasti parempi kuin uudemmilla työkaluilla. Uudempien testauskehysten etu vanhoihin verrattuna oli parempi luotettavuus ja paremmat työkalut testien vianetsintään. Päästä päähän -testauskehyksen valinnassa tuleekin usein valita parhaan joustavuuden ja parhaan käytettävyyden väliltä.

Avainsanat: testaus, päästä päähän -testaus, web-sovelluksen testaus, web-sovellus, end-to-end testing

SISÄLLYSLUETTELO

1. JOHDANTO

Ohjelmistokehitys kehittyy jatkuvasti nopeammaksi. Reagointikyky muuttuviin vaatimuksiin ja ohjelmistotarpeisiin on avainasemassa ohjelmistohankkeiden menestyksessä. Lisäksi sovellusten tulee olla nopeasti ja aina saatavilla, ja uusien ominaisuuksien julkaisemisen tulee tapahtua usein ja sulavasti. Tämä vaativa kehitysympäristö vaatii ohjelmien kattavaa ja toimivaa testausta tuotteen toimivuuden takaamiseksi.

Koska web-sovellusten kehityksessä tavallisesti käytetään paljon ulkopuolisia komponentteja ja kirjastoja, on erityisen tärkeää varmistaa ohjelmiston toimivuus kokonaisuutena. Ohjelmiston eri osien tulisi siis kommunikoida keskenään oikein ohjelman käyttötavoitteen saavuttamiseksi.

Web-sovellusten toimintaympäristön laajuuden takia myös testauksen kompleksisuus kasvaa. Koska koko sovelluksen toiminta riippuu niin monesta tekijästä, web-ympäristössä hyvinkin kattavatkaan yksikkötestit eivät usein riitä takaamaan ohjelmiston oikeellisuutta. Päästä päähän -testauksella (engl. end-to-end testing) tarkoitetaan koko järjestelmän testausta sellaisena, kuin loppukäyttäjä sitä tulee käyttämään. Tällöin varmistetaan kaikkien ohjelman osien yhteentoimivuus. Päästä päähän -testaus tuo testaukseen kuitenkin myös lisähaasteita. Web-sovelluksien toimintaympäristö on todella laaja, joka lisää vaatimuksia testauskehyksille. Testauskehysten tulee esimerkiksi pystyä hallitsemaan asynkronisia tapahtumia ja simuloimaan useiden eri selainten toimintaa.

Tämän tutkielman tarkoituksena on vertailla päästä päähän -testaukseen tarkoitettuja testauskehyksiä. Eri testauskehyksiä pyritään tarkastelemaan monesta näkökulmasta, jotta testauskehysten vahvuudet ja heikkoudet olisivat hyvin tunnistettavissa. Tällöin testauskehyksen valinnassa voitaisiin painottaa kyseisen käyttötapauksen tarpeita.

Ensin tutkielmassa kartoitetaan web-sovellusten toimintaympäristöä avaamalla selainten ja web-ohjelmien keskeisiä käsitteitä. Seuraavaksi kerrotaan testauksesta ja sen motivaatiosta, ja siitä, mitä erityispiirteitä sisältyy juuri web-sovellusten testaamiseen. Kolmanneksi tutustutaan neljään eri testauskehykseen, ja kerrotaan kehysten toimintaperiaatteista ja ominaisuuksista. Neljänneksi vertaillaan testauskehyksiä keskenään edeltävän luvun havaintojen pohjalta. Lopuksi selvitetään tutkimuksen tulokset, ja kerrotaan eri testauskehysten vahvuudet ja heikkoudet.

2. TESTAUS

Ohjelmistojen automaattinen testaus tarkoittaa sellaisten testausskriptien suorittamista, jotka varmistavat ohjelmiston eri osien toimivan halutulla tavalla. Automaattinen testaus manuaalisen testauksen sijaan parantaa tuotteen laatua, nopeuttaa ohjelmistoprojektin aikataulua sekä pienentää tarvittavaa kehitystyötä [1]. Rafi *et al.* [2] kertovat tutkimuksessaan automaattisen testauksen hyödyiksi uudelleenkäytettävyyden, toistettavuuden ja testien suoritusnopeuden. Heikkouksiksi mainitaan alun korkea työpanos, testien ylläpito ja testaustyökalujen heikkous.

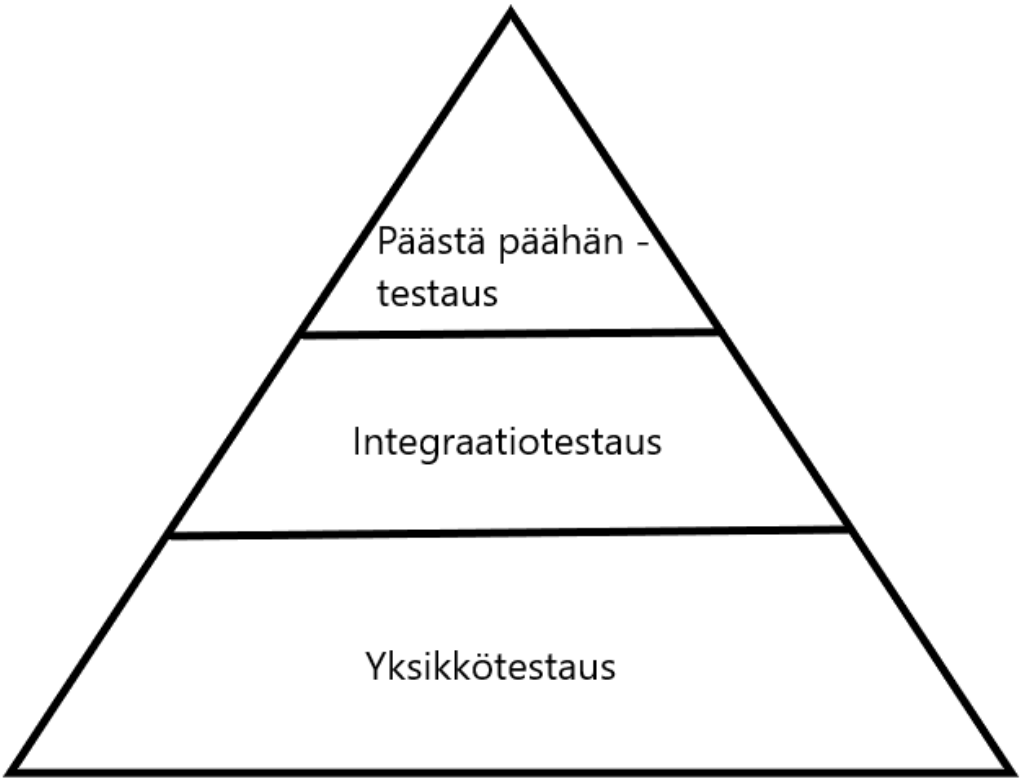
2.1 Ohjelmistojen testauksen kategoriat

Ohjelmistojen testaus voidaan jaotella karkeasti kolmeen kategoriaan, jotka ovat yksikkötestaus, integraatiotestaus ja päästä päähän -testaus. Kattavan testauksen saavuttamiseksi tarvitaan kaikkia testausmuotoja. Testauksen määrään eri tasoilla voidaan käyttää *testauspyramidia*, joka on esitetty kuvassa 2.1. Testauspyramidin esitti ensimmäisen kerran Mike Cohn [3]. Testauspyramidissa Cohn ohjeistaa testaamaan vähemmän, mitä korkeamalla abstraktiotasolla testataan. Seuraavaksi tarkennetaan eri testauskategorioita.

Yksikkötestaus tarkoittaa pienten yksiköiden, kuten funktioiden ja moduulien sisäistä testaamista. Tavoitteena on varmistaa oikeellinen lopputulos joillakin testisyötteillä. Yksikkötestit toimivat ohjelmiston pienimmällä abstraktiotasolla, joten testipyramidin mukaan niitä tulisi olla enemmän kuin muita testejä. Koska testattavilla yksiköillä on tavallisesti pieni vastuualue, ovat yksikkötestit usein yksinkertaisimpia ja nopeimpia kehittää.

Integraatiotestauksen tavoitteena on varmistaa isomman ohjelmakokonaisuuden toiminta. Siten testattavana on monen eri moduulin ja funktion keskinäinen toiminta. Integraatiotestauksen kohteena voisi olla esimerkiksi kirjautumisen onnistuminen palvelimella oikeilla käyttäjätunnuksilla. Koska integraatiotestattava kokonaisuus on suurempi kuin yksikkötesteissä, tulee myös testien teosta työläämpää [3].

Päästä päähän -testaamisen tavoitteena on varmistaa ohjelman toiminta varsinaisessa toimintaympäristössään eli kokonaisena ohjelmana. Ohjelma on siis tavallisesti päästä päähän -testauksessa samanlaisessa tilassa, kuin loppukäyttäjä tulee sen näkemään tuotantokäytössä. Tällöin varmistuu kokonaisuuden toiminta. Päästä päähän -testauksen on-



Kuva 2.1. *Testauspyramidi [4]*

gelma on testien hauraus ja suuri tarvittavan kehitystyöhön määrä [3].

Kaikissa testauksen kategorioissa testitapausten kulkua voidaan mallintaa *Arrange-Act-Assert* suunnittelumallilla, jonka esitteli William C. Wake [5]. Suunnittelumallissa Arrange-vaiheessa järjestetään testauksen kohde tilaan, jossa testaus on mahdollista. Act-vaiheessa suoritetaan testauksen kohteelle jokin toiminto. Lopuksi Assert-vaiheessa varmistetaan testauksen kohteen tilan olevan haluttu. Toisin sanoen varmistetaan, että Act-vaiheen toiminto suoritettiin halutulla tavalla.

2.2 Päästä päähän -testaus web-sovelluksissa

Web-sovellusten päästä päähän -testaukseen on monenlaisia työkaluja. Testauskehykset eroavat toisistaan esimerkiksi tuetuissa selaimissa ja testien luontiin käytetyissä ohjelmointikielissä. Yksi tapa kategorisoida testauskehykset on jakaa kehykset HTML-elementtien paikannusstrategian ja testien luomistavan mukaan [6].

Sivun HTML-elementtien paikannukseen yleisimmät keinot ovat seuraavat: absoluuttiset pikseliarvot, web-sovelluksen sisäisten DOM-elementtien käyttö, ja visuaaliset tunnistustyökalut. Testien luomiseen voidaan taas käyttää C&R (Capture & Replay) -lähestymista-

paa, jossa testaustyökalu toistaa alunperin ihmisen tekemät askeleet. Toinen lähetymistapa testien luomiseen on testien ohjelmointi. [6] Tällöin testit kirjoitetaan ohjelmakoodina testausskriptiksi, joka suorittaa tarvittavat askeleet testin suorittamiseksi.

Päästä päähän -testaus on erityisen tärkeää web-sovelluksissa, koska sovelluksen suoritusympäristö on hyvin laaja ja sovelluksen eri osien yhteensopivuus on kriittistä. Web-sovellusten testaamisessa tulee myös tyypillisesti ottaa huomioon vaihtuvat suoritusalustat eli selaimet. Eri selaimet tukevat eri ominaisuuksia web-standardeista, ja ominaisuuksien totetutus voi poiketa selainten välillä. Jos web-sovelluksen halutaan toimivan käytetyimmissä selaimissa, tulisi myös testejä suorittaa näillä selaimilla.

3. YMPÄRISTÖ

Web-sovellusten ympäristö koostuu monesta web-standardista ja teknologiasta, jotka kehittyvät jatkuvasti. Web-sovelluksille näistä tärkeimmät ovat:

- HTML (Hypertext Markup Language)
- CSS (Cascading Style Sheets)
- HTTP (Hypertext Transfer Protocol)
- JavaScript

Tässä luvussa selvitetään tarkemmin näitä web-sovelluksen toimintaympäristön tärkeimpiä teknologioita.

3.1 HTML

HTML on merkkauskieli, jolla ilmaistaan verkkosivun sisältö ja rakenne [7]. HTML-dokumentti on puurakenteinen ja koostuu HTML-elementeistä, jotka on merkitty kulmasulkein. HTML-elementeille voidaan lisäksi antaa attribuutteja, jotka kertovat miten elementin tulisi toimia [8]. HTML on deklaratiivinen kieli, eli se kuvaa sitä, *mitä* internet sivulla tulisi näyttää, mutta ei kerro *miten*.

Esimerkkinä on seuraava HTML-dokumentti

```
<body>
     <h1 id="sivun-otsikko">Otsikko</h1>
</body>,
```

jossa elementti `body` sisältää otsikkoelementin `h1`. Otsikkoelementille on lisäksi annettu attribuuttina yksilöivä tunniste `sivun-otsikko`, ja otsikkoelementin tekstisisältö `Otsikko` on elementin käyttäjälle näkyvä osa.

HTML:n käyttäjä eli selain parsii HTML-dokumentin, ja rakentaa muistiin DOM-puun (Document Object Model). DOM-puu on täysin HTML-dokumenttia vastaava objekti selaimen muistissa, jota voidaan manipuloida API:n (Application Programming Interface) avulla [8].

HTML:ään liittyy vahvasti CSS, joka kertoo miten DOM-puu tulisi esittää käyttäjälle [9]. CSS:llä voidaan hallita esimerkiksi elementtien värejä ja kokoja. CSS-tiedostossa tyyli-

informaatio annetaan selektoreiden ja ominaisuuksien avulla. Selektori kertoo, mihin elementteihin annetut ominaisuudet vaikuttavat. Erilaisia selektoreita ovat esimerkiksi elementtityypit, elementtien uniikit tunnistimet (id), sekä elementtien luokat (class).

3.2 HTTP

HTTP on geneerinen ja tilaton tiedonsiirtoprotokolla [10]. Sitä käytetään pääasiassa selainten ja palvelimien väliseen tiedonsiirtoon. HTTP-tiedonsiirto toimii client–server-mallilla, eli asiakas (client) pyytää palvelimelta (server) jotain, ja jää odottamaan vastausta.

Web-sovelluksissa HTTP on hyvin keskeisessä asemassa. Ensiksi, web-sovelluksen sisältävä HTML-dokumentti välitetään selaimelle HTTP-pyynnön seurauksena. Toiseksi, hyvin monet web-sovellukset tarvitsevat ulkopuolista ajonaikaista dataa. Yksi tapa tämän datan hakuun on JavaScriptillä toteutetut HTTP-pyynnöt.

3.3 JavaScript

JavaScript on web-sovellusten pääasiallinen ohjelmointikieli. Web-sovelluksille on tarjolla suuri määrä API-rajapintoja (Web API), joita käytetään juuri JavaScriptin kautta [11]. Yksi yleisimmistä rajapinnoista on DOM-rajapinta, joka mahdollistaa DOM-puun manipuloimisen. Dom-rajapinnalla voi manipuloida koko web-sovelluksen sisältöä, joten se mahdollistaa hyvin dynaamisten verkkosivujen suunnittelun. Kun verkkosivun dynaamisuus kasvaa ja sen tarkoitus ei ole vain näyttää tekstiä ja kuvia, voidaan alkaa puhua web-sovelluksesta verkkosivun sijaan.

Myös monet päästä päähän -testaukseen tarkoitetut testauskehykset mahdollistavat testien kirjoittamisen JavaScriptillä. Näissä tapauksissa JavaScript suoritetaan selaimen ulkopuolella, joten Web API:t eivät ole käytettävissä. Yleinen vaihtoehto on Node.js, joka suorittaa JavaScriptiä Google Chrome -selaimen JavaScript-moottorilla. [12]

3.4 Selaimet

Selain on ohjelma, joka yhdistää kaikki edellä mainitut standardit ja muodostaa DOM-puusta ja tyyliinformaatiosta käyttäjälle web-sovelluksen käyttöliittymän. Siksi selaimen vastuulla on hyvin suuri osa web-sovelluksen toiminnasta.

Selaimia on olemassa monia, ja usein selaimista on omat versiot eri käyttöjärjestelmille. Koska eri selaimia kehittävät eri organisaatiot, ja web-standardit kehittyvät koko ajan, tukevat eri selaimet eri osia web-standardeista. Tämä voi aiheuttaa eroja web-sovellusten toiminnassa eri selaimilla. Pahimmassa tapauksessa web-sovellus toimii vain yhdellä selaimella tietyssä käyttöjärjestelmässä.

3.5 Web-sovellusten tyypillinen rakenne

Tyypillinen web-sovellus on totetutettu client–server-mallilla [13]. Tälläisessä sovelluksessa suoraan selaimella näkyvän web-sovelluksen lisäksi toiminnallisuutta on myös jollain käyttäjän ulkopuolisella palvelimella. Perinteisesti data haetaan palvelimelta HTTP-pyynnöillä, mutta uusiakin tapoja on kehitetty, kuten Facebookin kehittämä GraphQL [14].

Koska useat web-sovellukset ovat hajautettu client–server-mallilla, tapahtuu sovelluksen suorittama logiikka monella eri alustalla. Tästä syystä web-sovelluksissa juuri kokonaisuuden testaaminen on tärkeää. On myös pystyttävä varmistamaan sovelluksen toiminta usealla eri selaimella.

4. TESTAUSKEHYKSET

Päästä päähän -testaukseen tarkoitetut testauskehykset ovat tavallisesti paketinhallintajärjestelmän avulla tallennettavia kirjastoja tai erilisiä ohjelmia, jotka mahdollistavat selaimen ohjauksen. On olemassa myös erilaisia testauskehyksiä, kuten selainlisäosana toteutettu Selenium IDE [15]. Testauskehykset eroavat toisistaan toteutusperiaatteiltaan sekä ominaisuuksiltaan.

4.1 Selainten ohjaus

Päästä päähän -testauskehyksien täytyy ohjata selaimen toimintaa, jotta web-sovellusten toimintaa voidaan testata. Ensimmäiset keinot selaimen ohjaamiseen perustuivat internetsivuun injektoituun JavaScript-koodiin. Esimerkiksi tällä hetkellä suosituimman testauskehyksen Selenium WebDriverin edeltäjän (Selenium RC server) toiminta perustui tähän periaatteeseen. [16] Selenium-projekti siirtyi tästä tavasta käyttämään WebDriver-standardia, mutta viime vuosina tämä tapa on taas yleistynyt. Tällä hetkellä suosittuja selaimessa suoritettavaa JavaScriptiä hyödyntäviä kehyksiä ovat vuonna 2017 julkaistu Cypress, sekä vuonna 2019 julkaistu TestCafe.

Hyvin suosittu tapa ohjata selaimia on WebDriver-standardi, joka tarjoaa vakiintuneen rajapinnan selainten ohjaamiseen ulkoisesti [17]. WebDriver-pohjaiseen selaimen ohjaukseen tarvitaan selainajuri, joka suorittaa natiivit selaintoiminnot standardin mukaisesti. Webdriver-standardi on tunnetun W3C (World Wide Web Consortium) järjestön ylläpitämänä standardina hyvin vakiintunut. Tästä syystä selainajuritoteutukset löytyvät todella monelle selaimelle, kuten vanhemmalle Internet Explorerille, jota uudemmat työkalut eivät tue. Suosittuja WebDriver-pohjaisia testauskehyksiä ovat Selenium WebDriver sekä OpenJs Foundation:in kehittämä Webdriver IO.

Uudempi tapa selainten ohjaukseen on selainten DevTools-protokollat. Devtools-työkalut ovat nimensä mukaisesti tarkoitettu sovelluskehittäjille web-sovellusten tarkasteluun ja vianetsintään, joten ne mahdollistavat selainten ohjauksen hyvin kattavasti. DevTools-protokollat eroavat siis WebDriver-standardista tarjoamalla laajemman kontrollin selaimen toimintaan. Esimerkiksi web-sovellusten tekemien ja vastaanottamien HTTP-pyyntöjen tarkastelu ja muokkaus ei sisälly WebDriver-standardiin, mutta on mahdollista DevTools-protokollilla. DevTools-protokollat eivät myöskään tarvitse testiajurin ja selaimen välille

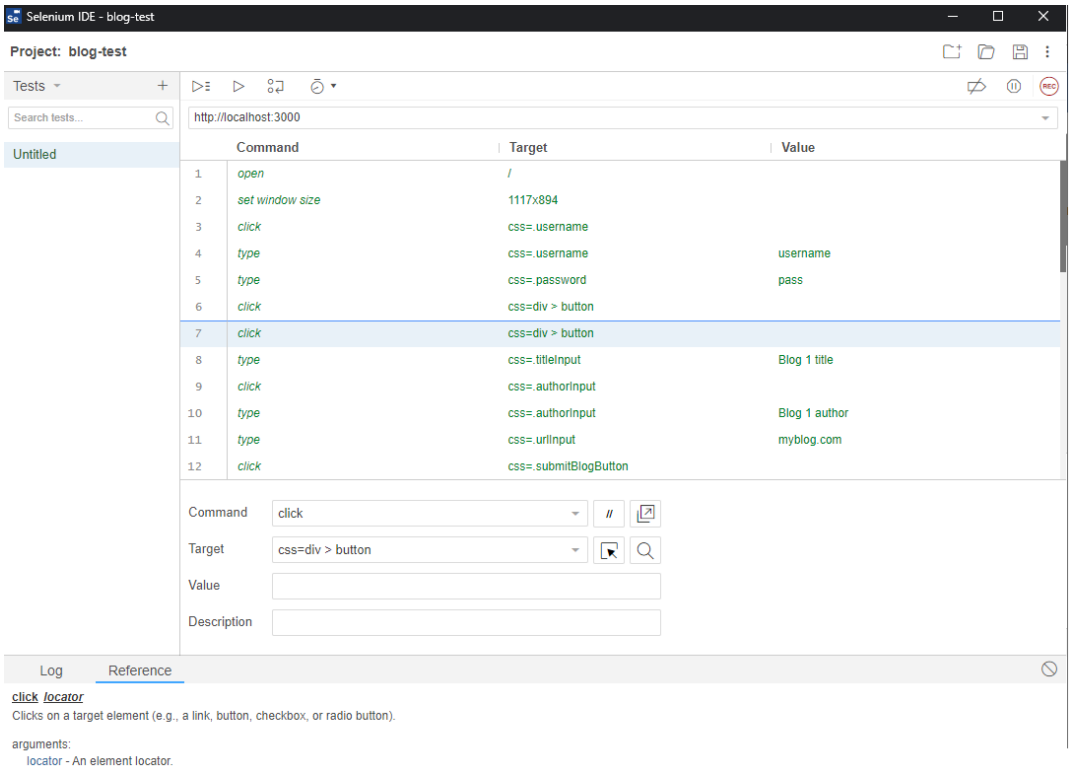
Kuva 4.1. *Esimerkkisovelluksen kirjautuneen käyttäjän näkymä*

selainajuria, koska DevTools-protokollan käskyt annetaan suoraan selaimelle. Välikappaleen puuttumisesta johtuen DevTools-pohjaiset työkalut ovat yleensä testien suorituksessa nopeampia kuin WebDriver-työkalut. Käytetyimpiä DevTools-pohjaisia testauskehyksiä ovat Googlen kehittämä Google Chrome -selaimen automatisointiin ja testaukseen tarkoitettu Puppeteer, sekä alunperin Puppeteerista lähtöisin oleva Microsoftin Playwright, joka on laajentanut tukeaan myös muihin selaimiin.

4.2 Vertailumetodi

Vertailtavina kehyksinä ovat Selenium WebDriver, Selenium IDE, Cypress sekä Playwright. Testauskehyksiä verrataan kirjoittamalla esimerkkisovellukselle yksittäinen päästä päähän -testi, ja vertaamalla prosessia kehysten välillä.

Esimerkkisovellus on yksinkertainen blogisovellus, johon käyttäjän tulee kirjautua käyttäjänimellä ja salasanalla. Sisäänkirjautumisen jälkeen käyttäjälle on näkyvissä kirjautumistiedot ja lista sovellukseen tallennetuista blogeista. Kuvassa 4.1 on esitetty näkymä kirjautuneesta käyttäjästä esimerkkisovelluksessa. Esimerkkitestissä testataan sovellukseen lisätyn blogin tykkäystoiminnallisuutta. Ennen kuin blogin tykkäys on mahdollista, tulee käyttäjän lisätä blogi sovellukseen. Tykkäysten määrän tulisi kasvaa yhdellä tykkäysnapin painalluksesta.



Kuva 4.2. *Selenium IDE*

4.3 Selenium IDE

Selenium IDE on osa Selenium-ohjelmakokonaisuutta, johon kuuluu myös Selenium WebDriver. Se eroaa muista verrattavista testauskehyksistä testien tuottamisessa. Selenium IDE:llä testejä luodessa ei ohjelmakoodia tarvitse välttämättä tuottaa ollenkaan. Testien luonti tapahtuu selainlisäosalla, joka tallentaa käyttäjän toiminnan askeleina. Testit ajettaessa testiajuri toistaa nämä askeleet. Selenium IDE on siis C&R tyylinen testauskehys. Askeleiden toistamisen jälkeen voidaan todentaa jokin ehto, kuten jonkin HTML-elementin olemassaolo, jolla varmistutaan ohjelman toimineen oikein.

Kuvassa 4.2 on Selenium IDE:n käyttöliittymä ja osa testisovellusta varten tehdystä testistä. Käyttöliittymästä voidaan luoda testausaskeleita manuallisesti, tai muuttaa olemassaolevien askelten ominaisuuksia, kuten HTML-elementtien paikannuslogiikkaa.

Testin tallentaminen Selenium IDE:llä oli hyvin yksinkertaista. Ensimmäinen testi oli hyvin nopea tuottaa, koska ohjelmointirajapintaan tutustumiseen ei tarvinnut käyttää aikaa. Käyttöliittymä oli myös yksinkertainen ja helppokäyttöinen. Käyttöliittymästä testin suorittaminen oli kuitenkin hyvin epäluotettavaa. Testien lopputulos vaihteli eri ajokerroilla, aiheuttaen hyvin epävarman vaikutelman työkalun käytöstä.

Selenium IDE:llä testien muokkaaminen ensimmäisen tallennuskerran jälkeen oli vaikeaa ja hidasta. Tämä voi olla yhteinen piirre C&R työkaluille, sillä myös Leotta *et al.* [6] toteavat C&R testien muokkauksen ohjelmoituja testejä huomattavasti hitaammaksi. Tämä

ongelma voi korjaantua työkalun pitkäaikaisemmalla käytöllä.

4.4 Selenium WebDriver

Selenium WebDriver on pitkään ollut käytetyin päästä päähän -testauskehys web-sovelluksille [16]. Se on julkaistu ensimmäisen kerran vuonna 2008. Selenium Webdriverin toimintaperiaate perustuu Webdriver-standardiin. Webdriver ohjaa web-sovelluksia kommunikoimalla erillisen ohjelman, selainajurin (engl. driver) kanssa. Ajurit ovat selainkohtaisia, ja selainten kehittäjät ylläpitävät myös ajureita. Erillisten testausajureiden asennus pitkittää hieman testausympäristön pystyttämistä.

Selenium Webdriver on vain rajapinta WebDriver-standardin mukaiseen selaimen ohjaukseen, eli sen vastuualueella ei ole testaus eikä tulosten tarkastelu. Tästä syystä varsinaiseen päästä päähän -testaukseen tarvitaan WebDriverin lisäksi jokin testauskirjasto, joka suorittaa testiajot ja varmistaa testien tuloksien oikeellisuuden. [18] Tämä tekee WebDriverista hyvin monipuolisen, ja voi mahdollistaa saman testauskirjaston käytön monella testauksen tasolla. Toisaalta testausympäristön rakentaminen voi pitkittyä, kun testauskirjasto tulee valita ja asentaa.

Kuvassa 4.3 on esitetty esimerkkitesti Selenium WebDriverin JavaScript rajapinnalla ja Mocha [19] testauskirjastolla. Testissä näkyy elementtien paikannus DOM-puusta CSS- tai XPath-selektoreilla ja elementeille suoritettavat toiminnot kuten metodi `.click()`. Mocha testauskirjastoa käytetään testien organisoimiseen ja rivin 41 tuloksen varmistukseen. Ohjelmassa rivillä 19 tehty HTTP-pyyntö on tehty käyttäen axios kirjastoa [20] toiminnon yksinkertaistamiseksi. Selenium on kehittänyt WebDriver-rajapinnat Python, Java, C-Sharp, JavaScript, Ruby, PHP ja Perl ohjelmointikielille. Ohjelmointirajapinnat myös toimivat ohjelmointikielien tavallisilla mekanismeilla, joten testauskirjaston voi valita vapaasti.

Kuvan 4.3 testistä voi myös hyvin tunnistaa Arrange-Act-Assert suunnittelumallin vaiheet. Testin alku kuuluu Arrange-vaiheeseen, kun ympäristö saatetaan sellaiseen tilaan, jossa blogin tykkääminen on mahdollista. Blogin tykkääminen riveillä 36–37 taas kuuluu Act-vaiheeseen. Lopuksi riveillä 40–42 tehdään varmistus lopputuloksesta eli suunnittelumallin Assert-vaihe. Päästä päähän -testeissä on tyypillisesti monimutkainen Arrange-vaihe, sillä moni sovelluksen alijärjestelmä tulee saada haluttuun tilaan.

Selenium WebDriverin ohjelmointirajapinta on selkeä, ja koska testeihin voi kirjoittaa mielivaltaista JavaScript koodia, on testausskriptien tuottamisessa Selenium IDE:en verrattuna enemmän vapautta. Testauskirjaston ominaisuudet myös tukevat WebDriverin komentoja hyvin. Alkuvalmistelujen jälkeen testien teko oli hyvin sulavaa.

```
const { Builder, By, until } = require('selenium-webdriver');
const axios = require('axios');
const assert = require('assert');

describe('With Selenium, blogs', () => {
    let driver;

    beforeEach(async () => {
        driver = await new Builder().forBrowser('chrome').build();
        // Yritä löytää elementtejä 1 sekunti
        await driver.manage().setTimeouts({ implicit: 1000 });
        await driver.get('http://localhost:3000/');
    })

    afterEach(async () => {
        await driver.close();
        await driver.quit();
        // Tyhjennä tietokanta
        axios.post('http://localhost:3000/api/test/reset')
    })

    it('can be liked', async () => {
        // Sisäänkirjautuminen
        await driver.findElement(By.css('.username')).sendKeys('username');
        await driver.findElement(By.css('.password')).sendKeys('pass');
        await driver.findElement(By.xpath('//button[@type=\"submit\"]')).click();

        // Blogin tallennus
        await driver.findElement(By.css('#toggle-button')).click();
        await driver.findElement(By.css('.titleInput')).sendKeys('Blog 1');
        await driver.findElement(By.css('.authorInput')).sendKeys('Blog 1 author');
        await driver.findElement(By.css('.urlInput')).sendKeys('myblog.com');
        await driver.findElement(By.css('.submitBlogButton')).click();

        // Blogin tykkääminen
        await driver.findElement(By.css('.expandButton')).click();
        await driver.findElement(By.css('.likeButton')).click();

        // varmista että blogilla tykkäys
        await driver.wait(until.elementTextContains(driver.findElement(By.css('.likeCount')), '1'), 500);
        const likeCount = await driver.findElement(By.css('.likeCount')).getText();
        assert.strictEqual(likeCount, '1');
    });
});
```

Kuva 4.3. *Blogien tykkäämisen testaus Selenium WebDriverilla.*

4.5 Cypress

Cypress on Cypress.io:n kehittämä vuonna 2017 julkaistu testauskehys. Cypressin lähestymistapa selaimen ohjaukseen eroaa Selenium WebDriverista merkittävästi. Cypressissä kaikki testausskriptin komennot suoritetaan samassa prosessissa kuin web-sovellus. [21] Tämä lähestymistapa mahdollistaa testausskriptin ajamisen huomattavasti lähempänä itse sovellusta. Tämä tarkoittaa esimerkiksi sitä, että HTML-elementtiä tarkastellessa Cypressillä on täysi hallinta elementistä DOM-rajapinnan kautta.

Kuvassa 4.4 on esitetty testausskripti blogin tykkäyksen testaamiseen Cypressin avulla. Cypressin ohjelmointirajapinta on hyvin yksinkertainen käyttää lyhyen tutustumisen jälkeen. Vertaamalla kuvan 4.3 WebDriver-toteutukseen, voi Cypressissä testin ilmaista paljon yksinkertaisemmin. Cypressissa on myös oletuksena DOM-elementtien odottaminen elementin käsittelyn yhteydessä. Tästä johtuen esimerkkitestissä ei WebDriver-toteutuksen tavoin määritellä erikseen odotusaikaa elementtien etsimiseen. Cypressissä myös arvojen oikeellisuuden tarkastelussa yritetään varmistusta automaattisesti uu-

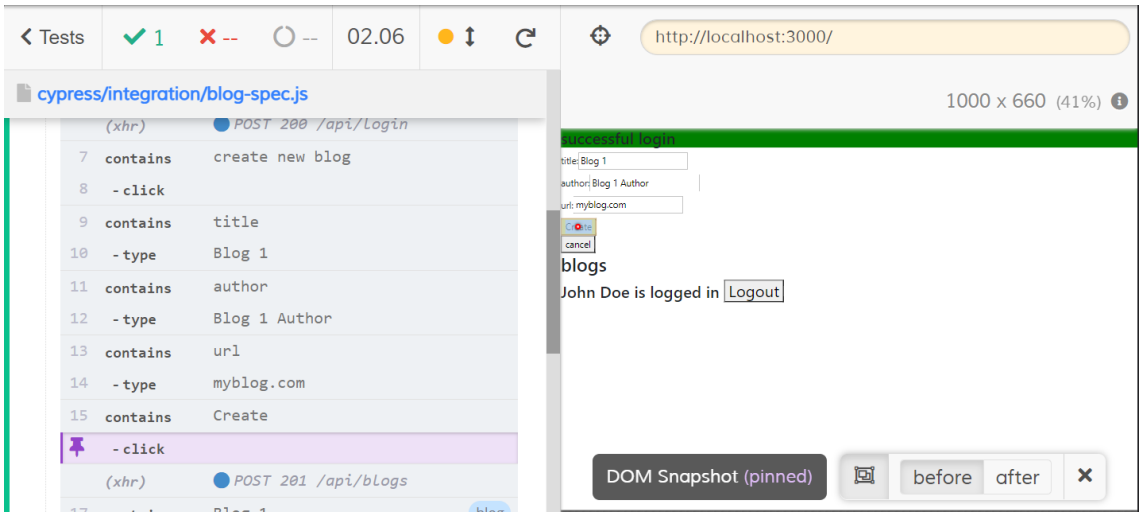
```
1  describe('With Cypress, blogs', () => {
2
3      beforeEach(() => {
4          cy.visit('http://localhost:3000');
5      });
6
7      afterEach(() => {
8          cy.request('POST', 'http://localhost:3000/api/test/reset');
9      });
10
11     it('can be liked', () => {
12         // Sisäänkirjautuminen
13         cy.contains('Username').type('username');
14         cy.contains('Password').type('pass');
15         cy.contains('login').click();
16
17         // Blogin tallennus
18         cy.contains('create new blog').click();
19         cy.contains('title').type('Blog 1');
20         cy.contains('author').type('Blog 1 Author');
21         cy.contains('url').type('myblog.com');
22         cy.contains('Create').click();
23
24         // Blogin tykkääminen
25         cy.contains('Blog 1').as('blog')
26         cy.get('@blog').contains('show').click();
27         cy.get('@blog').contains('like').click();
28
29         // varmista että blogilla on tykkäys
30         cy.get('@blog').get('.likeCount').should((likeCount) => {
31             expect(likeCount.text()).to.eq('1');
32         })
33     });
34 });
```

Kuva 4.4. *Blogin tykkäämisen testaaminen Cypressillä*

delleen tietyn ajan, joten Cypressissä voi rivillä 31 suoraan testata tykkäyksen arvoa 1. WebDriverissa testi suoritettaisiin liian nopeasti, joten samanlainen varmistus antaisi virheen, sillä arvo olisi edelleen 0 tietoliikenneviiveestä johtuen.

Cypressin `cy.contains()`-syntaksi mahdollistaa elementtien paikallistamisen ulkoisen tekstin avulla. Etu tässä lähestymistavassa on käyttäjälähtöisyys; käyttäjä tulee navigoimaan sovellusta samojen tekstien avulla. Etsimällä elementtejä ulkoisen tekstin perusteella testit eivät ole riippuvaisia sovelluksen sisäisestä toteutuksesta kuten DOM-puun rakenteesta. Tämä tekee testeistä vakaampia sovelluksen sisäisten muutosten suhteen.

Yksi Cypressin tärkeistä ominaisuuksista on testien debuggaus eli virheiden etsiminen testiskriptistä. Komentorivikomennolla `cypress open`, aukeaa Cypress Test Runner, josta voidaan ajaa Cypress-testejä. Kun jonkun testin ajaa, Cypress avaa halutun selaimen



Kuva 4.5. *Cypressin Test Status Menu*

ja Cypressin *Test Status Menu* käyttöliitymän. Tässä valikossa näkyy Cypressin suorittamat komennot testin edetessä, sekä lisätietoa selaimen muusta toiminnasta, kuten selaimen suorittamista HTTP-pyynnöistä. Selainnäkymä on esitetty kuvassa 4.5. Kuvasta nähdään vasemmalla Cypressin suorittamat komennot ja selaimen HTTP-pyynnöt, kuten POST-pyyntö blogin tietokantaan lisäämisessä. Selainnäkymästä voi myös valita jonkin Cypressin suorittaman komennon, jolloin selain näyttää sovelluksen tilan hetkellä ennen kuin komento suoritettiin ja kyseisen komennon kohde-elementin korostettuna. Kuvassa on valittuna tykkäysnapin painamista vastaava komento, jolloin Cypress näyttää punaisella pisteellä myös klikkauspisteen. Tämä näkymä helpottaa testien muokkausta huomattavasti.

Cypress ei tarvitse ulkopuolisia riippuvuuksia, joten se on asennuksen jälkeen valmis suorittamaan testejä. Asennukseen kuuluu testauskirjasto (Mocha) ja Chromium-pohjainen selain. Cypressin käyttöönotto on tästä syystä nopeaa.

Koska Cypress ei käytä WebDriver-standardin mukaista kommunikaatiota, on sen selaintuki rajallinen. Tällä hetkellä tuettuja selaimia ovat vain Google Chrome, Microsoft Edge sekä Mozilla Firefox. Olennaisena puuttena on Apple-laitteiden WebKit-pohjaiset selaimet, kuten Safari. Koska testausskriptiä ajetaan selaimessa, voidaan Cypress-testejä kirjoittaa vain JavaScriptillä. Näillä osa-alueilla Selenium Webdriver on Cypressiä huomattavasti monipuolisempi.

4.6 Playwright

Playwright on Microsoftin kehittämä selainten automaatio- ja testaustyökalu. Playwrightin virallinen 1.0 versio julkaistiin vasta marraskuussa 2020, joten se on muita työkaluja huomattavasti uudempi. Toisin kuin Cypressissä, Playwrightissa on tuki kaikille käytetyimmille selaimille: Google Chrome, Mozilla Firefox sekä Safari.

Playwright on Selenium WebDriverin tavoin ensisijaisesti automaatiotyökalu. Tästä syystä se tarvitsee varsinaiseen päästä päähän -testaukseen erillisen testauskirjaston. Toisin kuin WebDriver, Playwrightin käyttöön ei tarvitse asentaa erillisiä selainajureita. Tämä johtuu siitä, että Playwright käyttää WebDriver-standardin sijasta DevTools-protokollia, jotka sisältyvät selaimiin. Koska kaikki selaimet eivät tue DevTools-protokollia tarpeellisella tasolla, on Playwrightin kehittäjien täytynyt luoda selaimista muokatut versiot, joista tarvittava toiminnallisuus löytyy [22]. Nämä muokatut selaimet sisältyvät Playwrightin asennukseen.

Kuvassa 4.6 on Playwrightilla ja Mochalla totetutettu testi blogin tykkäämiseen. Playwrightin ohjelmointirajapinta on hyvin samanlainen kuin Selenium WebDriverilla. Eroja WebDriveriin on elementtien automaattinen odotus ja mahdollisuus hyvin monenlaiseen elementtien paikannusstrategiaan, kuten tekstisisältöön perustuva `text="hakusana"`-syntaksi. Koska Playwright käyttää DevTools-protokollaa, on sillä myös pääsy selaimen HTTP-pyyntöihin. Esimerkiksi kuvan testissä rivillä 42 tapahtuva blogin tykkäysten haku osaa odottaa HTTP-pyynnön valmistumista ennen komennon suoritusta. Vastaavalla ohjelmakoodilla Selenium WebDriver antaa tykkäysten määräksi 0, koska tykkäysten määrä ei ole ehtinyt päivittymään.

Playwrightissa on Cypressin tavoin työkalu helpottamaan vianetsintää testausskriptistä. Playwrightissa työkalun nimi on Playwright Inspector, ja se voidaan asettaa avautumaan testausskriptiä suorittaessa. Tällä työkalulla voidaan testausskriptiä käydä läpi rivi kerrallaan, jolloin käyttöliittymässä näkyy kohteena oleva elementti ja klikkauksen kohde.

```
const { chromium } = require('playwright');
const axios = require('axios');
const assert = require('assert');

describe('With Playwright, blogs', () => {
    let browser;
    let context;
    let page;

    beforeEach(async () => {
        browser = await chromium.launch({ headless: false, slowMo: 800 });
        context = await browser.newContext();
        page = await context.newPage();
        await page.goto('http://localhost:3000');
    })

    afterEach(async () => {
        await context.close();
        await browser.close();
        // Tyhjennä tietokanta
        axios.post('http://localhost:3000/api/test/reset')
    })

    it('can be liked', async () => {
        // Sisäänkirjautuminen
        await page.fill('text=Username >> input', 'username');
        await page.fill('text=Password >> input', 'pass');
        await page.click('text=login');

        // Blogin tallennus
        await page.click('text="create new blog"');
        await page.fill('text=title >> input', 'Blog 1');
        await page.fill('text=author: >> input', 'Blog 1 Author');
        await page.fill('text=url >> input', 'myblog.com');
        await page.click('text=Create');

        // Blogin tykkääminen
        await page.click('text=show');
        await page.click('text=like');

        // Varmista että blogilla on tykkäys
        const likes = await page.textContent('.likeCount');
        assert.strictEqual(likes, '1');
    });
});
```

Kuva 4.6. *Blogin tykkäämisen testaaminen Playwrightilla*

5. TESTAUSKEHYSTEN VERTAILU

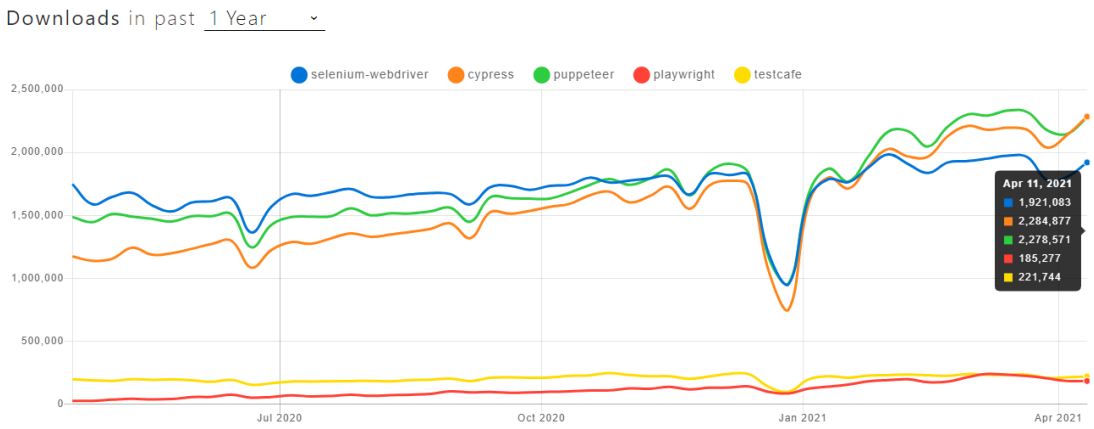
Testauskehyksiä voidaan vertailla monesta näkökulmasta. Kehysten vertailussa on tarkoituksena löytää testauskehysten vahvuudet ja heikkoudet, jolloin testauskehyksen valinta voi pohjautua projektin ja testaajien tarpeisiin. Monesti testauskehyksen kriteerit vaihtelevat suuresti projektien välillä.

5.1 Kehysten suosio

Selenium-projektin työkalut (Selenium WebDriver, Selenium IDE) ovat olleet pitkään hyvin vakiintuneessa asemassa suosituimpina päästä päähän -testauksen työkaluina. Cerioli *et al.* [23] toteavat työilmoitustutkimuksessa Seleniumin olevan myös arvokkain testauskehys. Seleniumin WebDriverin ja Selenium IDE:n lisäksi myös moni muu selaimen automaatioon ja testaukseen tarkoitettu työkalu on rakennettu alun perin Selenium-tiimin ehdottaman WebDriver-standardin päälle [16]. Standardin mukaiset selaimen etäohjaamista suorittavat ajurit ovat hyvin tärkeä osa monia testaustyökaluja. Esimerkiksi suositut WebDriverIO ja Katalon Studio käyttävät sisäisesti WebDriver-standardia.

Koska WebDriveriin perustuvat työkalut jakavat saman pohjan, on niissä yhteisiä ongelmia. Yleisimpiä ongelmia WebDriver-pohjaisissa testauskehyksissä ovat ylläpidettävyys, luotettavuus sekä testien kirjoittamiseen tarvittava kehitysaika [16]. Viime vuosina WebDriver-standardiin pohjautuvien testauskehysten rinnalle on noussut myös ulkopuolisia työkaluja kuten edellä mainitut Cypress, TestCafe, Puppeteer sekä Playwright. Tarve uudenlaisille työkaluille on syntynyt jatkuvasti kehittyvistä web-sovellusten dynaamisuudesta ja kompleksisuudesta, johon WebDriverin toiminnalisuus ei enää riitä.

Testauskehysten suosiota kuvaa hyvin suositun paketinhallintajärjestelmän Npm:n [24] lataustilastot, jonka avulla kaikki mainitut työkalut voidaan asentaa. Kuvaaja lataustilastoista on esitetty kuvassa 5.1. Kuvaajasta näkee viimeisen vuoden aikana Cypressin sekä Puppeteerin ohittaneen Selenium WebDriverin latauksissa. Kuvaaja ei kuitenkaan kerro koko tarinaa, sillä WebDriver-standardia käyttää sisäisesti moni testauskehys. Myös muiden ohjelmointikielien Selenium Webdriver toteutukset eivät ole osa tätä lukua. Koska Cypress ja Puppeteer tukevat vain JavaScriptiä, suuri osa käyttäjistä on osa kuvaajaa. Kuvan perusteella WebDriver on siis luultavasti edelleen käytetyin testauskehys, mutta uudempien testauskehysten suosio kasvaa jatkuvasti.



Kuva 5.1. *Testaustyökalujen lataustilastot [25]*

5.2 Testien luonti

Verrattavissa työkaluissa voi testit muodostaa ohjelmakoodista tai tallentamalla testausaskeleet. Selenium IDE on puhtaasti C&R-työkalu, mutta myös Playwright ja Cypress tarjoavat vaihtoehdot luoda testejä C&R-tyyliin. Molemmissa työkaluissa nauhoitus muodostaa toimivan testausskriptin. Leotta *et al.* [6] toteavat rikkinäisten testaustulosten korjausnopeutta tarkastellessa testausskriptien olevan C&R-menetelmiä järkevämpi ratkaisu jo kahden julkaisusyklin jälkeen. Tämän luvun perusteella jokaisen varteenotettavan projektin tulisi käyttää testausskriptejä C&R-menetelmien sijasta.

C&R-menetelmien arvo löytyy testauksen alkuvaiheista. Ensimmäinen testi on hyvin nopea toteuttaa C&R-työkaluilla, sillä se ei tarvitse tutustumista työkalun ohjelmointirajapintaan [6]. Tuotettu testausskripti antaa hyvän pohjan jatkokehitykselle ja samalla helpottaa ohjelmointirajapinnan lähestymistä. Cypress ja Playwright tuottavat tallennustökaluilla suoraan käsinkirjoitettua vastaavan testausskriptin, ja Selenium IDE tarjoaa mahdollisuuden testin vientiin Selenium WebDriver -testausskriptiksi. Kaikkia WebDriver-yhteensopivia kieliä tosin ei tueta.

5.3 Luotettavuus

Luotettavuus on hyvin tärkeää päästä päähän -testauksessa. Luotettava testauskehys saavuttaa samalla testikonfiguraatiolla saman lopputuloksen, kun testi suoritetaan useamman kerran. Epäluotettava testauskehys voi johtaa tilanteeseen, jossa testien tulokset ovat epäluotettavia. Tässä tapauksessa esimerkiksi automaattisesti suoritettu testi pitää suorittaa uudestaan testitapauksen epäonnistuessa, joka aiheuttaa ylimääräistä työtä.

Selenium WebDriver ja Selenium IDE jakavat samoja ongelmia luotettavuuden suhteen. Koska WebDriver-komentojen täytyy kulkea ensin selainajurin kautta, voivat web-sovelluksen elementtien paikat muuttua toimintojen välissä. Esimerkiksi Googlen kehittämä

Chrome-selainta ohjaavan ChromeDriverin dokumentaatio kertoo tästä ilmiöstä [26]. Tämä ongelma esiintyy erityisesti silloin, kun web-sovelluksen sisältö on dynaamista.

Cypressin ja Playwrightin luotettavuudesta ei löydy tutkimusta, mutta molempien lähestymistavat selainautomaatioon osin poistavat WebDriver-työkalujen epäluotettavuutta. Cypress toimii samassa prosessissa kuin web-sovellus, joten sen pitäisi olla aina tietoinen elementtien sijainnista. Myös Playwrightin suora yhteys selaimeen DevTools-protokollalla poistaa testiajurin ja selaimen välistä yhden askeleen, joten toiminnan pitäisi olla varmempaa. Sekä Cypress ja Playwright automaattisesti odottavat sovelluksen elementtejä, joka myös auttaa testien luotettavuudessa.

5.4 Suoritusaika

Nopeus voi olla hyvin kriittinen osa-alue päästä päähän -testauskehyksen valinnassa. Testien riittävä suoritusnopeus on tärkeää tilanteissa, joissa testejä on hyvin suuri määrä tai testit tulee suorittaa usein. Myös testejä kehittäessä nopea suoritusaika nopeuttaa testien tuottamista.

Eri testauskehysten testien suoritusaika vaihtelee hieman. Testauskehyksien välillä voi olla joitakin eroja samoissa toiminnoissa, esimerkiksi eräässä testissä Cypress oli 23% hitaampi kuin Puppeteer [27]. Suurimmat hyödyt testien suoritusajassa tulevat kuitenkin testien rinnakkaistamisella. Osana Selenium-projektia on Selenium Grid [28]. Se mahdollistaa Selenium IDE:llä ja Selenium WebDriverilla tehtyjen testien ajamisen usealla eri järjestelmällä rinnakkain. Cypress ja Playwright tukevat testien rinnakkaistamista ilman lisäkomponentteja.

5.5 Kehitysaika

Testien kehityksessä ei ole merkittävää eroa testauskehysten välillä, mutta Cypressin ja Playwrightin syntaksi on yksinkertaisempaa kuin Selenium WebDriverissa, joka helpottaa kehitystyötä. Selenium WebDriverista puuttuu myös oma vianetsintätyökalu, joka löytyy Cypressistä ja Playwrightista. Vianetsintätyökalu helpottaa testien luontia näyttämällä virheiden syyt testien virhetilanteissa. Vianetsintätyökaluista Cypressin työkalussa on enemmän ominaisuuksia, kuten web-sovelluksen tilan tarkastelu testin jokaisessa vaiheessa.

Dokumentaatio on myös tärkeä osa kehitystä. Hyvä dokumentaatio nopeuttaa testien kehitystä ja auttaa noudattamaan oikeita periaatteita. Kaikkien työkalujen dokumentaatio on hyväksyttävällä tasolla, mutta Cypressin dokumentaatio on työkaluista kattavin. Toisaalta, WebDriver on ollut hyvin monta vuotta suosituin testaustyökalu, joten sen käytöstä löytyy eniten ulkopuolisia resursseja [16].

5.6 Joustavuus

Testauskehyksen joustavuudella voidaan tarkoittaa joko kehitysympäristön joustavuutta tai testattavan ympäristön joustavuutta. Kehitysymäristön joustavuudella voidaan helpottaa kehitystyötä käyttämällä esimerkiksi kehittäjille tuttua ohjelmointikieltä ja testauskehystä. Testattavassa ympäristössä voidaan tukea monia eri selaimia, jolloin varmistutaan sovelluksen toiminta myös esimerkiksi vanhemmalla selaimella.

Kehitysympäristössä testauskehys voi tukea eri ohjelmointikieliä testausskriptin luomiseen. Tarkastelluista kehyksistä Selenium WebDriver ja Playwright tukevat useita eri ohjelmointikieliä, mutta Cypress tukee vain JavaScriptillä kirjoitettuja testejä. Koska Cypress ajaa testejä selaimessa, ei se todennäköisesti tule tukemaan muita kieliä JavaScriptin lisäksi. Myös testauskirjastojen suhteen WebDriver ja Playwright tukevat lähes mitä vain testauskirjastoa, sillä molemmat ovat ominaisuuksiltaan pääasiassa automaatiotyökaluja. Cypress-testeissä oletetaan käytettävän Cypressin asennuksen kuuluvia testauskirjastoja.

Testattavaan ympäristöön joustavuutta antaa tuki usealle selaimelle. Selenium WebDriverissa on tuki kaikille käytetyimmille selaimille, myös vanhemmalle Internet Explorerille. Playwright taas tukee käytetyimpiä moderneja selaimia: Mozilla Firefox, Chromium-selaimet ja Safari. Cypress tukee vain Mozilla Firefoxia ja Chromium-selaimia, eli se tukee testatuista työkaluista pienintä määrää selaimia. Cypressin kehittäjät suunnittelevat myös Safarin tukemista jatkossa [21].

6. YHTEENVETO

Web-sovelluksien laajassa ympäristössä on erityisen tärkeää varmistaa ohjelmiston osien toimivuus yhdessä. Tähän varmistamiseen voi käyttää päästä päähän -testausta, jossa ohjelmistoa testataan samassa ympäristössä kuin loppukäyttäjä. Päästä päähän -testaukseen on kehitetty useita työkaluja, joita tässä tutkielmassa vertailtiin eri näkökulmista.

Tutkielman haasteena on suppea työkalujen vertailu. Esimerkkitestit olivat yksinkertaisia, joten testauskehyksistä saatu kuva jäi suppeaksi. Tarkemmalla työkaluihin tutustumisella olisi vertailusta saanut kattavamman ja luotettavemman. Nykyinen vertailu painottuu ensivaikutelmaan testauskehykistä, eikä testauskehysten kehityneempiä ominaisuuksia juurikaan käytetty. Uudemmista työkaluista, kuten Playwrightista ja Cypressistä, oli myös hyvin vähän tietoa niiden omien dokumentaatioiden lisäksi, mikä voi antaa vääristyneen kuvan testauskehysten vahvuuksista ja heikkouksista.

Näiden haasteiden perusteella jatkotutkimukselle on aihetta. Jatkotutkimuksen voisi kohdistaa testauskehysten tarkempaan tarkasteluun esimerkiksi tekemällä suuremmalle websovellukselle kattavat päästä päähän -testit. Vertailuun voisi myös lisätä lisää testauskehyksiä kattavemman kuvan saavuttamiseksi. Myös uudempien testauskehysten suorituskykyä tulisi tutkia empiirisesti, jolloin suorituskyvyn syvempi vertailu olisi mahdollista.

Jokaiselle työkalulle löytyy käyttötarkoitus. Capture & Replay työkalut kuten Selenium IDE ovat hyvä tapa aloittaa päästä päähän -testaus. Niillä ensimmäinen testi saadaan tuotettua hyvin nopeasti, koska testin luonti ei vaadi ohjelmakoodin kirjoittamista. C&R työkaluiden suurin heikkous on testien heikko muokattavuus. Vaikean muokattavuuden vuoksi testit suositellaan muunnettavan esimerkiksi Selenium IDE:n tapauksessa Selenium WebDriver -skriptiksi. Tällöin päästä päähän -testauksessa päästään nopeasti alkuun, mutta vältetään heikon muokattavuuden aiheuttamat ongelmat.

Selenium WebDriverin ja muiden WebDriver-standardiin pohjautuvien testauskehyksien etuina on joustavuus. Testejä voi luoda hyvin monilla eri ohjelmointikielillä ja testauskirjastoilla. Ohjelmointikielien lisäksi WebDriver-testauskehykset tukevat suurinta määrää selaimia. WebDriverin heikkouksia on testien satunnainen epäluotettavuus, sekä monimutkainen testausympäristö. WebDriver-testausympäristöön pitää lisätä selainajurit ja testauskirjasto, ennen kuin yhtäkään testiä voi suorittaa.

Cypressin etuna on testien luomisen helppous. Sen asennus sisältää kaikki tarvittavat työkalut testaukseen. Cypressissä on myös kattavat työkalut testien vianetsintään ja tarkasteluun, joka tekee testien muokkaamisesta vaivattomampaa. Testien epäluotettavuus on pyritty ratkaisemaan kokeilemalla komentojen suoritusta automaattisesti uudestaan tietyn ajan ennen virheen synnyttämistä. Cypressin heikkouksia on vain yksi tuettu ohjelmointikieli ja rajallinen määrä tuettuja selaimia.

Playwright tukee WebDriverin tavoin useita ohjelmointikieliä ja useimpia käytettyjä selaimia, mutta siinä on pyritty ratkaisemaan testien epäluotettavuus yrittämällä komentojen suoritusta uudestaan kuten Cypressissä. Playwright tarjoaa myös laajemman kontrollin selaimesta käyttämällä DevTools-protokollaa WebDriver-standardin sijasta. Playwright on kuitenkin hyvin uusi työkalu (julkaistu 2020), joten sen ohjelmointirajapinnat voivat muuttua rajusti. Uudemman työkalun käyttöön löytyy myös vähemmän resursseja kuin pitkään käytössä olleeseen Selenium WebDriveriin.

Lopulta testauskehyksen valinta riippuu projektin tarpeista. Kehyksen valinnassa täytyy usein valita parhaan joustavuuden ja parhaan käytettävyyden välillä. Playwright vaikuttaa kuitenkin olevan lähes WebDriverin tasolla joustavuudessa, mutta parantaa käytettävyydessä esimerkiksi automaattisella odotuksella ja yksinkertaisemmalla syntaksilla. Playwright voi siis tulevaisuudessa olla lähes joka tilanteessa parempi ratkaisu kuin Selenium WebDriver.

LÄHTEET

[1] Dustin, E. *Automated Software Testing: Introduction, Management, and Performance*. Toim. editor. Addison Wesley Professional, 1999.

[2] Rafi, D., Moses, K., Petersen, K. ja Mäntylä, M. Proceedings of the 7th International Workshop on automation of software test. *Benefits and limitations of automated software testing: systematic literature review and practitioner survey*. 2012.

[3] Cohn, M. *Succeeding with Agile: Software Development Using Scrum*. Toim. editor. Addison-Wesley Professional, 2009.

[4] *The Practical Test Pyramid*. 26. helmikuuta 2018. URL: https://martinfowler.com/articles/practical-test-pyramid.html (viitattu 21. 03. 2021).

[5] Wake, W. C. *Extreme Programming Explored*. Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN: 0201733978.

[6] Leotta, M., Clerissi, D., Ricca, F. ja Tonella, P. Approaches and tools for automated end-to-end web testing. *Advances in Computers*. Vol. 101. Elsevier, 2016, s. 193–237.

[7] *HTML basics*. 19. helmikuuta 2021. URL: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics (viitattu 19. 03. 2021).

[8] *HTML. The Living Standard*. 18. maaliskuuta 2021. URL: https://html.spec.whatwg.org/ (viitattu 19. 03. 2021).

[9] *CSS. Cascading Style Sheets*. 18. maaliskuuta 2021. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (viitattu 19. 03. 2021).

[10] *Hypertext Transfer Protocol – HTTP/1.1*. 1. kesäkuuta 2014. URL: https://tools.ietf.org/html/rfc2616 (viitattu 20. 03. 2021).

[11] *Web APIs*. 19. helmikuuta 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API (viitattu 19. 03. 2021).

[12] *Introduction to Node.js*. 20. maaliskuuta 2021. URL: https://nodejs.dev (viitattu 20. 03. 2021).

[13] Reese, G. *Database programming with JDBC and Java*. O'Reilly, 2000.

[14] *GraphQL*. 15. huhtikuuta 2021. URL: https://graphql.org/ (viitattu 15. 04. 2021).

[15] *Selenium*. 15. huhtikuuta 2021. URL: https://www.selenium.dev/ (viitattu 15. 04. 2021).

[16] Garcia, B., Gallego, M., Gortazar, F. ja Munoz-Organero, M. A survey of the selenium ecosystem. *Electronics* 9.7 (2020), s. 1067.

[17] *WebDriver*. 24. elokuuta 2020. URL: https://www.w3.org/TR/webdriver/ (viitattu 15. 04. 2021).

[18] *Understanding the components*. 15. huhtikuuta 2021. URL: https://www.selenium.dev/documentation/en/webdriver/understanding_the_components/ (viitattu 15. 04. 2021).

[19] *Mocha*. 15. huhtikuuta 2021. URL: https://mochajs.org/ (viitattu 15. 04. 2021).

[20] *Axios*. 15. huhtikuuta 2021. URL: https://github.com/axios/axios (viitattu 15. 04. 2021).

[21] *Cypress*. 15. huhtikuuta 2021. URL: https://www.cypress.io/ (viitattu 15. 04. 2021).

[22] *Playwright*. 15. huhtikuuta 2021. URL: https://github.com/microsoft/playwright (viitattu 15. 04. 2021).

[23] Cerioli, M., Leotta, M. ja Ricca, F. What 5 million job advertisements tell us about testing: a preliminary empirical investigation. *Proceedings of the 35th Annual ACM Symposium on applied computing*. SAC '20. ACM, 2020, s. 1586–1594. ISBN: 9781450368667.

[24] *Npm*. 15. huhtikuuta 2021. URL: https://www.npmjs.com/ (viitattu 15. 04. 2021).

[25] *npm trends*. 19. huhtikuuta 2021. URL: https://www.npmtrends.com/ (viitattu 19. 04. 2021).

[26] *Clicking issues*. 16. huhtikuuta 2021. URL: https://sites.google.com/a/chromium.org/chromedriver/help/clicking-issues (viitattu 16. 04. 2021).

[27] *Cypress vs Selenium vs Playwright vs Puppeteer speed comparison*. 27. tammikuuta 2021. URL: https://blog.checklyhq.com/cypress-vs-selenium-vs-playwright-vs-puppeteer-speed-comparison/ (viitattu 16. 04. 2021).

[28] *Selenium Grid*. 24. huhtikuuta 2021. URL: https://www.selenium.dev/documentation/en/grid/ (viitattu 25. 04. 2021).