QRec – automaatiotyökalun kehitys

Mika Lindh

2019 Laurea

Laurea-ammattikorkeakoulu

QRec – automaatiotyökalun kehitys

Mika Lindh
Tietojenkäsittelyn tradenomi
Opinnäytetyö
Helmikuu 2019

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittelyn koulutusohjelma
Tradenomi (AMK)

Lindh, Mika

QRec – automaatiotyökalun kehitys

Vuosi 2019 Sivumäärä 55

Tässä toiminnallisessa opinnäytetyössä kuvataan projekti, jossa tehtävänä oli suunnitella ja kehittää Qentinel Finland Oy:lle sovellus, jonka avulla tekniseltä osaamiseltaan rajallisesti asiaa ymmärtävä käyttäjä pystyy luomaan automaatioskriptiä web-sovelluksille. Sovelluksen ideana on tallentaa käyttäjän suorittamat painallukset ja syötteet käyttöliittymärajapinnasta. Tallenteeseen perustuen generoidaan suoritusvalmis skripti, joka voidaan ajaa yrityksen PACE -automaatioperheeseen kuuluvalla Console -sovelluksella.

Työn teoreettinen viitekehys sisältää työn suorittamisen kannalta keskeisimmät käsitteet ja käytetyt menetelmät. Teoriaosuudessa taustoitetaan myös yrityksen käyttämää Pacewords -skriptaustekniikkaa ja skriptaustekniikan merkitystä automaatioprojektien kannattavuuden ja ylläpidon kannalta.

Työn tietoperustana käytettiin alan ammattikirjallisuutta sekä kokeneiden asiantuntijoiden henkilöhaastatteluja. Työn toiminnallisen osuuden pohjana oli tekijän oma aiempi osaaminen, sekä toteutuksessa käytettyjen menetelmien tekniset dokumentaatiot.

Työn tuotoksena syntyi toimiva sovellus, jonka avulla automatisointi-, ja ohjelmointitaidoton käyttäjä voi nauhoittaa käyttötapauksia, jotka sovellus kääntää ohjelmistorobotin ymmärtämäksi skriptiksi.

Asiasanat: Testiautomaatio, Chrome-laajennus, JavaScript, skriptaustekniikka

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bachelor's Thesis

Lindh, Mika

QRec – automation tool development process

2019 Pages 55

The objective of this bachelor's thesis was a development project where the task was to design and develop an automation tool that can record user actions from a web application's user interface and then generate an automation script based on those recordings. The purpose of the project was that any technical or non-technical user can automate any use case using the tool. The developed tool was a part of Qentinel Finland's Pace product, which is a fully scalable test automation solution for companies.

The theoretical part of the thesis contains the most important concepts and methods which were used during the development process. The thesis also described the Pacewords scripting technique that is used by the company, and the importance of the scripting technique in terms of reusability and maintenance costs. All information was based on interviews with experienced professionals and on written and electronic sources.
The functional part of the thesis was based on the author's prior JavaScript knowledge and technical documentations of the used technologies and libraries.

The outcome of the project was a functional Chrome Extension–based application that can be used by any technical-, or non-technical user which was also defined in the beginning of the development project. The tool is today used daily by consultants within the company.

Keywords: Test automation, Chrome Extension, JavaScript, scripting technique

Sisällys

1 Johdanto

Tässä toiminnallisessa opinnäytetyössä kehitettiin Qentinel Finland Oy:lle sovellus, joka generoi suoritusvalmista testiautomaatioskriptiä tallentamalla käyttäjänsä suorittamia käyttötapauksia web-sovelluksen käyttöliittymärajapinnasta. Generoitu testiskripti noudattaa yrityksen Pace -automaatiotuotteen käyttämää Pacewords -skriptaustekniikkaa ja sitä voidaan ajaa sellaisenaan Pace-konsolissa. Pace-konsoli on ohjelma, joka sisältää Qentinel Pacen QAutomation -kirjastot, joista tämän työn yhteydessä käsitellään QWebiä.

Qentinel Pace automaatiotuote on Qentinel Finlandin kokonaisratkaisu, joka sisältää yhtenä osanaan QAutomation kirjastokokoelman. QWeb on websovellusten testaamiseen ja automatisoimiseen kehitetty automaatiokirjasto. Tämän opinnäytetyön aiheena olevan QRec sovellus on kehitetty palvelemaan QWebin konsultti- ja asiakaskäyttäjiä.

Opinnäytetyö tehtiin pääosin syksyn 2018 aikana. Työssä kuvataan kehitetty sovellus, kehitysprosessin kulku, sekä kerrotaan työhön valitusta teknisestä viitekehyksestä, joka on Chrome -selainlaajennus ja sen tarjoamat rajapinnat. Sovelluksen arkkitehtuurin ja komponenttien lisäksi työssä kuvaillaan joitain prosessin aikana esiin tulleita teknisiä haasteita, sekä ratkaisumalleja, joilla näitä haasteita on selvitetty.

Työn ulkopuolelle on rajattu varsinaisen ohjelmointityön, ohjelmointikielien tai koodin dokumentointi. Toteutuksessa käytettyjä, ja toiminnallisen osuuden kuvauksessa näkyviä JavaScript/jQuery -tekniikoita käsitellään yleisellä tasolla tapauksissa, joissa se on tämän työn dokumentoinnin osalta oleellista. Ulkopuolelle rajautuivat myös isommassa kuvassa työn keskeisenä viitekehyksenä olevat testiautomaatio ja RPA.

Työn teoreettinen viitekehys käsittelee Keyword-perusteista skriptaustekniikkaa ja sen tuomia etuja verrattuna ohjelmointityyliseen lähestymistapaan. Teoriaosuudessa taustoitetaan myös yrityksen käyttämää Pacewords-skriptaustekniikkaa, sekä skriptaustekniikan merkitystä automaatioprojektien ylläpidon kannalta yleisesti ottaen.

2 Toimeksiantaja ja työn tausta

Työn toimeksiantaja on Qentinel Finland Oy, jossa opinnäytetyön tekijä työskentelee testiautomaatiokonsulttina. Qentinel Finland on v. 2002 perustettu laadunvarmistukseen erikoistunut yritys, jolla on toimintaa Suomen lisäksi Saksassa ja USA:ssa. Opinnäytetyön aiheena oleva toimeksianto on osa yhtiön testiautomaatiotuotteen normaalia tuotekehitystyötä. (Qentinel Finland Oy 2018.)

Opinnäytetyöntekijällä oli jonkin verran aiempaa osaamista JavaScriptistä, joka ohjelmointityön osalta nähtiin web-sovellusten natiivikielenä järkevimmäksi lähestymistavaksi ratkaisun toteuttamiseen, ja vaikutti siten osaltaan myös tekijän valikoitumiseen työn suorittajaksi.

2.1 Aihe ja tavoite

Kehitystyön suunnitteluvaiheessa kartoitettiin sovelluksen toteuttamiseen sopiva menetelmä, jonka jälkeen tavoite oli valittua menetelmää käyttäen toteuttaa sovellus, jolla kyetään tuottamaan yksinkertaisia automatisoituja käyttötapauksia helposti ja nopeasti. Ratkaisun tulisi olla niin helppokäyttöinen, että sen omaksuminen ei vaadi ymmärrystä ohjelmoinnista, tai itse automaatiotuotteen toiminnasta.

Pace-testiautomaatiotuotteen yksi keskeisistä ajatuksista on tehdä automatisoinnista ja testiskriptien ylläpitotyöstä niin helppoa, että käyttötapauksia pystyy kohtuullisen pienellä opastuksella automatisoimaan kuka tahansa käyttäjä, joka hallitsee perustasoiset tietokoneen käyttötaidot ja osaa käyttää automatisoitavaa järjestelmää. Automatisoinnin tekninen tuominen käyttäjien itsensä ulottuville mahdollistaa sen, että usein rajallisia ja kalliita alan asiantuntijaresursseja voidaan vapauttaa aikaa syövien perusrutiinien suorittamisen sijaan haastavampien ongelmien ratkaisuun.

Työn aiheena olevan sovelluksen nähtiin täydentävän erinomaisesti yrityksen jo olemassa olevaa kokonaispakettia, ja tukevan osaltaan näkemystä helposta ja tehokkaasta automatisointiprosessista. Sovellus madaltaa teknisesti vähemmän vihkiytymättömän henkilön kynnystä kokeilla automatisoimista, ja auttaa tätä omaksumaan yrityksen käyttämän skriptaustekniikan perusteet. Sovellus nopeuttaa ja tehostaa myös alan ammattilaisten työskentelyä.

Markkinoilla on valmiiksi olemassa sekä kaupallisia-, että avoimen lähdekoodin -ratkaisuja perustuen esimerkiksi Robot Frameworkin ja Seleniumin käyttämiin skriptaustekniikoihin, mutta näiden lähestymistapa ei sellaisenaan soveltunut Qentinel Paceen. Tutkimalla valmiita ratkaisuja todettiin myös, että jonkin olemassa olevan tuotteen muokkaaminen Pacen käyttämälle Pacewords-tekniikalle sopivaksi, olisi joka tapauksessa niin mittava työ, että kokonaan oman ratkaisun kehittäminen koettiin järkevimmäksi ratkaisuksi.

2.2 Minimivaatimukset

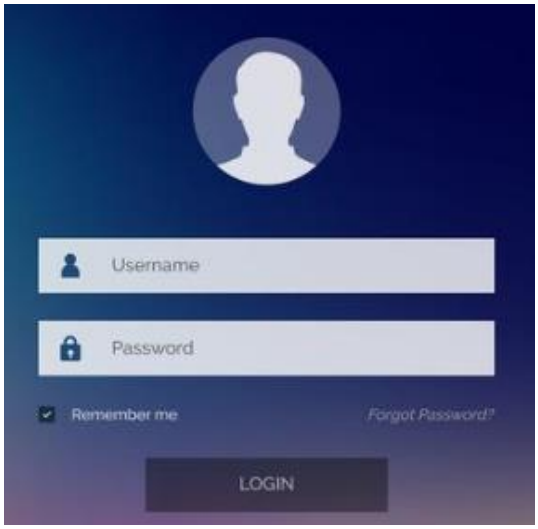
Vaatimusmäärittelyssä lähdettiin siitä, että toteutuksen tulee olla helposti asennettavissa ja käytettävissä. Lähtökohtaisesti koko sovelluksen käyttötapahtuman tulee olla kaikilta osin niin helppo, että kuka tahansa tietokoneen käyttötaidot omaava pystyy itsenäisesti sekä asentamaan ohjelman, että aloittamaan sen käytön. Toiminnallisiin minimivaatimuksiin otettiin työn aloitusvaiheessa muutamien avaintapahtumien tunnistaminen ja generoiminen skriptiksi, sekä yhden robottitiedoston luominen.

Käytännössä vaatimusmäärittely piti sisällään sen, että aloitusvaiheessa ohjelman piti osata tunnistaa interaktiiviset elementit yksinkertaiselta klikkaa ja kirjoita tyyppiseltä sivustorakenteelta, poimia kyseisiin elementteihin kohdistuvat käyttötapahtumat ja keräämänsä informaation perusteella luoda testiskripti, jolla voidaan toistaa kyseinen käyttötapaus Pace-robotilla. Taulukossa 1 esitetään käyttötapaus, jonka tapahtumavirta sovelluksen tulisi tunnistaa ja generoida tunnistuksensa perusteella automaatioskripti.

1	Käyttäjä navigoi webohjelman kirjautumissivulle.
2	Käyttäjä avaa QRecin, kirjoittaa sen otsikko -kenttään käyttötapauksen nimeksi Login Test, jonka jälkeen hän käynnistää QRecin tallennuksen painamalla siihen tarkoitukseen tehtyä nappulaa.
3	Käyttäjä kirjoittaa Username kenttään käyttäjätunnuksen demo.
4	Käyttäjä kirjoittaa Password kenttään salasanaksi mode.
5	Käyttäjä painaa nappia, jossa lukee Login.
6	Käyttäjä haluaa vahvistaa, että kirjautumisen jälkeen ruudussa näkyy teksti Welcome

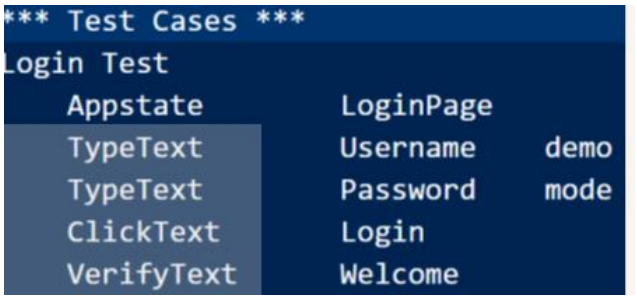
Taulukko 1: Käyttötapaus: Kirjautuminen

Käyttötapausesimerkki simuloi käyttäjän kirjautumista sovelluksen kirjautumisikkunaan. Ikkuna sisältää tekstikentät käyttäjänimelle ja salasanalle sekä nappulan, jossa lukee Login (kuvio 1).



Kuvio 1: Tyypillinen kirjautumisikkuna (Vecteezy 2018)

QRec tuottaa käyttötapauksesta automaatioskriptin, jonka perusteella robotti osa toistaa käyttötapauksen. Jokaisesta käyttäjän nauhoitettavaan sovellukseen kohdistamasta toimenpiteestä generoidaan yksi rivi skriptiä, jotka yhdessä muodostavat kokonaisen käyttötapauksen (kuvio 2).



Kuvio 2: Minimivaatimukset täyttävä testiskripti

Teknisenä vaatimuksena TypeText, ClickText, VerifyText ovat hyvin itsensä selittäviä toimintoja, ja myös sovelluksen kehitystyön osalta vähemmän taustalogiikkaa vaativia. Kun käyttäjä klikkaa jotakin, niin poimitaan teksti, tai tarvittaessa joku muu attribuutti, jolla voidaan osoittaa robotille oikea elementti. Vastaavasti, kun käyttäjä kirjoittaa jotain, niin ohjelman tulee em. asioiden lisäksi poimia käyttäjän ohjelmalle kirjoittama syöte (esimerkkisyötteet demo ja mode).

VerifyText-pacewordia varten sovellukseen tarvitaan erillinen mekanismi, joka mahdollistaa käyttäjälle helpon ja intuitiivisen tavan osoittaa haluttu teksti QRecille. Appstate on aloitustila, esim. tyypillisesti joku url-osoite, johon ohjelma halutaan ohjata ennen varsinaisen käyttötapauksen suoritusta. Appstaten alustukseen liittyvät toimet eivät saa näkyä itse testiskriptissä, joten kehitettävään sovellukseen tarvitaan logiikka, jonka perusteella se nimeää nauhoituksen alkaessa Appstatelle annettavan parametrin (kuvio 2:ssa LoginPage). Sovellus sijoittaa robottilogiikkaa sisältävään tiedostoon navigointitoimet, jotka automaatio suorittaa aina, kun Appstate sille kyseisen parametrin lähettää.

2.3 Rajaus

Seuraavissa otsikoissa esitellään skriptaustekniikka, johon tehdyn sovelluksen generoima automaatioskripti pohjautuu. Lisäksi esitellään projektiin liittyvää suunnitteluprosessia, sen toteutustavaksi valittua teknistä viitekehitystä, sekä yleisiä teknisiä toimintaperiaatteita. Tämän opinnäytetyön ulkopuolelle on rajattu varsinaisen ohjelmointityön, ohjelmointikielien tai koodin dokumentointi.

Toteutuksessa käytettyjä ja toiminnallisen osuuden kuvauksessa näkyviä JavaScript/jQuery -tekniikoita käsitellään yleisellä tasolla tapauksissa, joissa se on tämän työn dokumentoinnin osalta oleellista. Teoreettisen viitekehyksen ulkopuolelle rajautuvat isommassa kuvassa työn keskeisenä viitekehyksenä olevat testiautomaatio ja RPA. Rajaukset perustuvat opinnäytetyön kokoon, joka kasvaisi koko alue mukaan lukemalla liian isoksi. Ohjelmointityötä ja kooditasoisia yksityiskohtia ei esitetä, koska työ on osa kaupallista kokonaisuutta ja se sisältää materiaalia, jota ei voi julkaista. Kaikki työssä näkyvät esimerkit ovat yleisluontoisia, eivätkä sisällä mitään sellaista logiikkaa, joka vaarantaisi tilaajan tuotesuojaa. Joitakin työssä käytettäviä

termejä (kuten Keyword Driven Approach, Background Script, Content Script etc.) ei ole lähdetty suomentamaan, koska näiden menetelmien tekniseksi ilmaisutavaksi on vakiintunut englanti.

2.4 Käsitteet

Testiautomaatio: Testiautomaatiolla tarkoitetaan ohjelmistojen testaamista, joka toteutetaan ohjelmallisesti. Testaaminen on keskeinen osa ohjelmistokehityshankkeita. Automatisaation avulla pyritään karsimaan siihen kuluvia ihmistyötunteja, joka tehostaa ja nopeuttaa kehitysprosessia. Automaation avulla päästään myös entistä korkeampaan testauskattavuuteen, joka näkyy lopputuotteessa parempana laatuna.

RPA: RPA on lyhenne sanoista Robotic Process Automation. RPA on teknologia, jonka avulla automatisoidaan sellaisia työtehtäviä ja prosesseja, joiden suorittaminen on loogisesti mahdollista ilman ihmistä.

Scrum: Ketterässä ohjelmistokehityksessä yleisesti käytettävä projektinhallintamenetelmä, jossa projekti on vaiheistettu toisiaan seuraaviksi kehitysjaksoiksi, eli sprinteiksi.

Skripti (script): Tämän työn yhteydessä skripteillä tarkoitetaan lyhyitä, automaatio-ohjelmistolle annettavia komentoja, joiden avulla tätä käskytetään suorittamaan sille annettua tehtävää.

DOM: Lyhenne sanoista Document Object Model. DOM on ympäristöriippumaton ohjelmointirajapinta, joka käsittelee HTML, XML ja XHTML-dokumentteja strukturoituna puuna, jonka jokainen elementti on oma itsenäinen komponentti osana dokumentti.

HTML: Hypertext Markup Language on internet-selainten lukema kuvauskieli, jolla selaimelle kuvataan hypertekstin rakenne, ja minkälaisena selaimen tulee tämä esittää.

CSS, Bootstrap: CSS on kieli, jolla määritellään HTML-dokumentin tyyli. Bootstrap on CSS-kirjasto, joka tarjoaa ohjelmoijalle valmiita tyyliteltyjä malleja käytettäväksi.

JavaScript, jQuery: JavaScript on pääasiassa Web ympäristössä käytettävä ohjelmointikieli, jota on käytetty tämän opinnäytetyön sovelluksen tekemiseen. jQuery on ominaisuuksiltaan rikas JavaScript-kirjasto, joka tarjoaa apuvälineitä, joiden avulla JavaScriptiä voidaan kirjoittaa joissain tapauksissa helppolukuisemmin ja nopeammin.

JSON: JavaScript Object Notation on avoimen standardin tiedostomuoto tiedonvälitykseen. JSON on objekti, joten ohjelmoija voi sijoittaa sinne strukturoidusti lähes minkälaista dataa hyvänsä ja lukea sen myöhemmin helposti pois sieltä.

2.5 Tutkimusongelmat

1.) Ratkaisun tulee perusominaisuuksiltaan olla riippumaton web-sovelluksesta, jonka käyttöä sillä taltioidaan. Se ei saa estää käsiteltävän sovelluksen normaalia toimintaa.

2.) Sovelluksen toiminnan tulee olla session ajan täysin kontrolloitavissa, joka tarkoittaa, että se osaa reagoida esim. dynaamisesti muuttuvaan DOM-sisältöön siten, että jokainen ruudulla näkyvä elementti on ohjelman seurannassa heti sinne ilmestymisensä jälkeen.

3.) Sovelluksen tulee osata poimia HTML -standardielementeistä oikeanlaiset identifoijat generoitavaan Pacewords -skriptiin. Kehitystyötä tehdessä on otettava huomioon, että varsinkin nykyaikaisilla JS Frameworkeilla tuotetut sivut poikkeavat DOM -rakenteensa osalta hyvinkin paljon perus HTML -sivuista ja myös toisistaan. Onko mahdollista löytää mahdollisimman geneerinen ratkaisumalli, joka toimii sivulla kuin sivulla.

4.) Koska web-maailmassa ns. best practices käytännöt elävät jatkuvasti ja ovat usein sidoksissa käytettyyn frameworkiin, niin ratkaisun tulee olla kohtalaisen pienillä muutoksilla räätälöitävissä vaikkapa hyvinkin yksilölliseen käyttötarpeeseen.

3 Skriptaustekniikka – keyword perustainen lähestymistapa

QRecin generoima ja QWebin käyttämä Pacewords perustuu Keyword Driven Approach (Avainsana pohjainen) -nimiseen skriptaustekniikkaan. Tekniikan ajatuksena on se, että varsinainen testiskripti sisältää vain ja ainoastaan testi-/käyttötapauksen, eikä se ota kantaa siihen, miten se suoritetaan. Menetelmään perustuen voidaan kirjoittaa hyvin samanlaisia itsensä dokumentoivia testitapauksia kuin manuaalitestauksessa. Tekniikan keskeisenä etuna nähdäänkin, että se mahdollistaa itsensä selittävien ja helposti luettavien skriptien kirjoittamisen, joka helpottaa merkittävästi niiden dokumentoimista ja ylläpitoa. (Fewster & Graham 1999, 89-90.)

Lähestymistapa vaatii erillisen kontrolliskriptin, joka osaa tulkata käytetyt keywordit, sekä jonkinlaisen ohjelmointilogiikan näiden taustalle. Koska menetelmä vaatii useamman erillisen palasen sovittamista toisiinsa, niin se saattaa aluksi kuulostaa monimutkaisemmalta, kun yksi tietylle tapahtumalle koodattu skripti, joka sisältää kaiken tarpeellisen. Todellisuudessa tilanne on kuitenkin päinvastainen. Mitä suurempi määrä käyttötapauksia automatisoidaan, sen suurempi hyöty lähestymistavalla saavutetaan. Kun kontrolliskripti ja taustalogiikka on kerran kirjoitettu, niin itse käyttötapausten kirjoittaminen on nopeaa, eikä niiden kirjoittaminen vaadi tekijältään tietämystä taustaskriptien toiminnasta. (Fewster & Graham 1999, 89-91.)

Keywordit ovat monessa tapauksessa uudelleenkäytettäviä, joka helpottaa projektien ylläpitotyötä merkittävästi. Tekniikka mahdollistaa selkeästi suuremman automaatioasteen saavuttamisen nopeammalla aikataululla ja automaation ylläpitämisen aiempaa pienemmillä kustannuksilla. (Fewster & Graham 1999, 89-91.)

Keyword-perusteista automaatiota voikin paremmin kuvailla siten, että testiskriptit eivät ole varsinaisesti testiskriptejä, vaan toimintaohjeita/testidokumentaatioita, jotka voivat kaikki käyttää mahdollisesti vain yhtä ja samaa skriptiä, joka osaa lukea ja suorittaa ne. Lähestymistapa mahdollistaa tuhansien testitapausten suorittamisen vain parin sadan skriptin avulla. (Fewster & Graham 1999, 89-91.)

Kuviossa 3 esitettävästä perinteisestä ohjelmointityylisestä skriptistä on todennettavissa, että skripti ei ole kovinkaan hyvin itseään selittävä. Skriptin kirjoittaminen vaatii ymmärrystä ohjelmoinnista.

```
driver = webdriver.Chrome()
driver.get(loc.postcode_url)
assert "Posti" in driver.title
elem = driver.find_element_by_name("streetname")
elem.clear()
elem.send_keys("Bertel Jungin aukio")
elem = driver.find_element_by_name("postcodeorcommune")
elem.clear()
elem.send_keys("Espoo")
elem.send_keys(Keys.RETURN)
assert "02600" in driver.page_source
driver.close()
```

Kuvio 3: Yhden käyttötapauksen suorittava ohjelmointityylinen skripti (Heimola 2018)

Saman testitapauksen voisi kirjoittaa keyword-pohjaista menetelmää käyttäen huomattavasti helpommin tulkattavalla tavalla. Taulukon 2 esimerkissä vahvistettuna olevat käskyt ovat käyttäjille tarjotut keywordit, joiden avulla tämä voi ohjailla suorituksen kulkua (minne tehdään ja mitä tehdään). Jokainen keyword kuvaa yhtä yksinkertaista toiminnallisuutta, joka voidaan suorittaa miten monta kertaa hyvänsä. Käyttötapauksen kirjoittajan ei tarvitse tietää miten automaatio toiminnallisuuden suorittaa.

1	**Avaa selain ja mene osoitteeseen** www.osoitetiedot.com	
2	**Kirjoita kenttään** streetname	Bertel Jungin aukio
3	**Kirjoita kenttään** postcodeorcommune	Espoo\n
4	**Vahvista sivulta teksti** 02600	

Taulukko 2: Käyttötapaus toteutettuna Keyword -skriptaustekniikalla

Keyword perusteinen malli mahdollistaa testi-/käyttötapausten kirjoittamisen teknisesti rajoittuneille testaajille tai liiketoiminnan edustajille. Ohjelmointityötä tarvitaan edelleen, mutta se tehdään käyttötapausten sijaan kontrolliskriptiin ja taustarakenteisiin, jotka ohjaavat miten keywordit toimivat. Tämä tekee automaatioprojektin osaamisresursoinnista selkeää, ja projekteista itsessään entistä tehokkaampia ja ketterämpiä (Crispin & Gregory 2009, 182-183).

Edellä taulukossa 2 kuvatun esimerkin ’Kirjoita kenttään’ -avainsana tekee aina täsmälleen saman asian. Ainoastaan sijainti ja syöte vaihtuvat. Ohjelmointilogiikkaa tarvitsevassa taustaskriptissä määritellään miten mikäkin asia tehdään, eikä käyttötapauksen kirjoittajan tarvitsee tietää siitä mitään. Kuviossa 4 esitettävä taustaskripti sisältää taulukossa 2 esitettyjen keywordien suorittamiseen tarvittavan logiikan. Taustaskripti on uudelleenkäytettävä ja samaa logiikkaa käyttäviä käyttötapauksia voidaan kirjoittaa kuinka monta hyvänsä ilman, että taustascriptiin tarvitsee koskea.

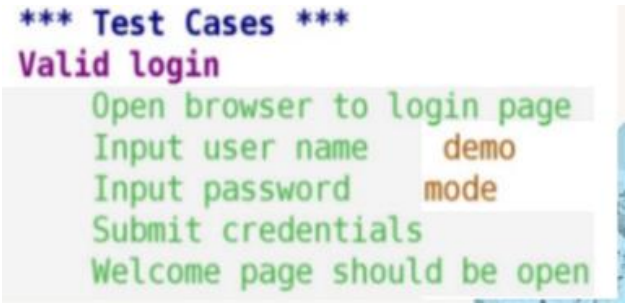
```
Kirjoita kenttään (haluttu kenttä, haluttu syöte)
    driver = webdriver.Chrome()
    elem = driver.find_element_by_name(haluttu kenttä)
    elem.send.keys(haluttu syöte)

Avaa selain ja mene osoitteeseen(osoite)
    driver = webdriver.Chrome()
    driver.get(osoite)

Vahvista sivulta teksti(haluttu teksti)
    driver = webdriver.Chrome()
    assert haluttu teksti in driver.page_source
```

Kuvio 4: Keyword-tekniikalla toteutetun tapauksen taustaskripti

3.1 Robot Framework

Nykyaikaista automaatiotyökaluista esim. Suomessa kehitetty, ja Qentinel Pace -automaatiotuotteen pohjallakin käytettävä, Robot Framework perustuu keyword-pohjaiseen skriptaustekniikkaan. Robot Framework tarjoaa käyttäjälleen viitekehyksen ja oman kirjastokokoelmansa, jonka päälle käyttäjä voi rakentaa testinsä (Robot Framework 2018).

Robot Framework on geneerinen, pääasiassa Python ja Java -ohjelmointikieliä tukeva testiautomaatio- ja nykyisellään myös RPA sovelluskehys. Se soveltuu lähes kaikkeen testaamiseen, ja on järjestelmä-, sekä ohjelmistoriippumaton, jonka vuoksi se vaatii jonkin rajapinnan itsensä ja käsiteltävän ohjelman väliin. Robot Framework tarjoaa hyvät raportointiominaisuudet ja se tukee laajasti erilaisia ulkoisia rajapintakirjastoja. Robot Framework on helposti käyttöönotettava avoimen lähdekoodin alusta, jonka ansiosta sen kehitys ja rajapintakirjastojen lisääntyminen on todella nopeaa. Alustan sen päälle voidaan kirjoittaa myös omia kirjastoja. (Robot Framework, 2018.)



Kuvio 5: Tyypillinen Robot Frameworkilla kirjoitettu käyttötapaus

Robot Framework ja keyword-pohjainen skriptaustekniikka itsessään, vie automaatioratkaisuita uudelleenkäytettävämpään ja ylläpidettävään suuntaan, mutta on tärkeää huomioida, että kyseessä on vasta viitekehys. Hyvin usein automaatioprojekteissa käy edelleen niin, että keywordeista tulee hyvin ympäristöspesifejä (uudelleenkäyttö kyseenalaista) ja niiden määrä kasvaa ylläpidon kannalta hallitsemattoman suureksi. Tilaajayrityksen toimesta on tutkittu ja arvioitu, että perus Robot Framework syntaksiin perustuvalla mallilla saavutetaan vain n. 20-30 % uudelleenkäytettävyys. (Heimola 2018.)

3.2 Pacewords

Pacewords on Qentinel Finlandin Pace-automaatiotuotetta varten kehittämä skriptaustekniikka, joka perustuu keyword-pohjaiseen malliin. Pacewordit ovat ohjelmoitu alusta- ja ympäristöriippumattomaan kirjastokokoelmaan, joka pyörii Robot Frameworkin päällä. Tuotteen avulla rajataan spesifi, tekstintunnistukseen ja rajattuun keywordien määrään perustuva viitekehys, joka tekee automaatioskriptin kirjoittamisesta helposti omaksuttavaa, ja jonka avulla on päästy merkittävästi aiempaa korkeampaan keywordien uudelleenkäytettävyysasteeseen. (Heimola 2018.)

Yrityksen tutkimuksen mukaan käyttöliittymän kautta tapahtuvissa automaatioprojekteissa pienellä, lukumääräisesti vain noin 20:llä - projekti- ja alustariippumattomalla - uudelleenkäytettävällä pacewordilla ylletään 80 % käyttötapauskattavuuteen. Prosenttiluku perustuu

tuotteen kehityksen aikana tutkittuun yli 700000 yksittäisen stepin otokseen. Nopeus ja kustannustehokkuus ovat menetelmän avulla parantuneet merkittävästi. Pacewordeja on toki lukumääräisesti enemmän kuin mainitut 20 kappaletta, mutta tutkimuksen mukaan 20:llä yleisimmällä pystytään kattamaan lähes kaikki yleisimmät käyttötapaukset. (Heimola 2018.)

Pacewordit ovat ohjelmoitu Pythonilla erilliseen kiinteästi Pace-automaatiotuotteeseen kuuluvaan ja tällä hetkellä Robot Frameworkin päällä pyörivään kirjastoon. Skriptaustekniikka ja pacewordit ovat on yhdenmukaisia riippumatta siitä mikä on automatisoitava kohde ja mitä tekniikkaa käytetään esimerkiksi käyttöliittymärajapinnassa olevien elementtien ja tekstien tunnistamiseen. (Heimola 2018.)

Kuviossa 6 esitetään opinnäytetyöntekijän kirjoittama toimiva skripti, jolla on automatisoitu käyttötapaus, jonka kuluessa toimintoja suoritetaan sekä web-sovelluksen, työpöytäintegraation, että MS Excelin käyttöliittymärajapinnoissa. Skriptin kirjoittajan näkökulmasta useampi erilainen ympäristö ei näy automatisoidessa paljoakaan. Esimerkkiskriptiä lukemalla voimme tunnistaa vaiheen, jossa käsitellään Exceliä, mutta emme voi sanoa skriptin perusteella missä vaiheessa käyttötapausta käsittelemme web-sovellusta ja missä vaiheessa kyseessä on työpöytäintegraatio. Kunkin pacewordin taustalla oleva logiikka valikoituu käsiteltävän ympäristön mukaan (esim. tässä tapauksessa perus web-valitsimien, konenäön, ja office- kirjastojen välillä), eikä automaatioskriptiä kirjoittavan henkilön tarvitse siitä välittää.

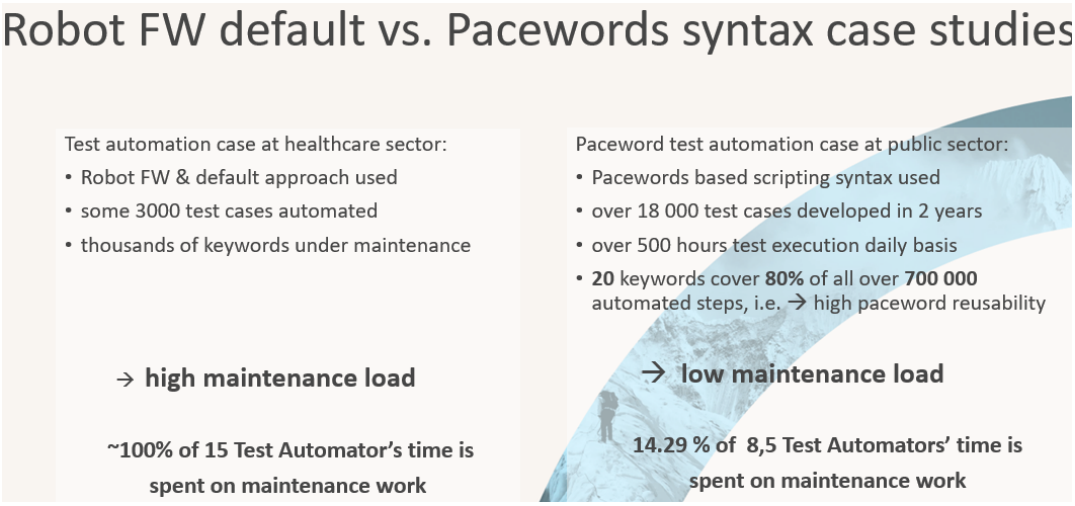
```
Webbi, desktop, Excel
    [tags]          Web+Excel
    Appstate        Kirjaus
    ClickItem       Navigointi
    ClickItem       Kirjaukset
    ClickItem       Tehtävät
    ClickText       Luo tapahtuma
    VerifyText      Erän nimi
    TypeText        Tapahtuman nimi     Testi
    TypeText        kirja               Robottitestaus
    TypeText        henkilö             QRobot
    ClickText       Lähetä
    UseExcel        testi.xlsm
    ClickText       kylla
    TypeText        tunnus              demo
    TypeText        salasana            effect
    ClickIcon       Login
    VerifyCell      D6                  1000
    CellCopyPaste   D5                  E13
    TypeCell        I13                 QRobot_Test
    TypeCell        N13                 1
    TypeCell        P13                 10.000
    TypeCell        Q13                 QRobot_Test
```

Kuvio 6: Pacewords -esimerkkiskripti

Pacewordit, kuten esimerkiksi TypeText, ClickText, ClickItem ja VerifyText ovat kiinteitä ja tekevät aina saman asian. Ne ovat itsensä selittäviä ja pienen lukumääränsä vuoksi helppoja omaksua ja muistaa. Käyttäjältä vaaditaan ainoastaan viittaus paikkaan, johon toiminto koh-

distetaan sekä annettavat syötteet. Menetelmä perustuu ohjelman käyttöliittymän visuaalisten tekstien, erilaisten tooltippien, soluviittausten yms. mahdollisimman intuitiivisesti rekisteröitävien asioiden tunnistamiseen. (Heimola 2018.)

Uudelleenkäytettävyydellä ja helposti omaksuttavalla tekniikalla voidaan tehostaa merkittävästi automaatioprojekteja. Tehostamisen ansiosta saavutetaan selvästi suurempi käyttötapauskattavuus ja ylläpitoresursseja tarvitaan entistä vähemmän.



Kuvio 7: Uudelleenkäytettävyyden merkitys (Heimola 2018)

4 Tutkimusmenetelmät

Ennen kehitystyön aloittamista tehtiin tutkimus, jonka avulla kartoitettiin työkalun toteuttamiseen liittyvät keskeisimmät tutkimusongelmat, sekä tekninen viitekehys, jonka avulla havaitut ongelmat voitaisiin ratkaista. Tutkimuksen tulokset ovat tukena sekä projektin suunnittelu-, että sen tekovaiheessa. Tämän kehitysprojektin pääasiallisina tutkimusmenetelminä käytettiin toimintatutkimusta, empiiristä tutkimusta sekä hiljaisen tiedon tutkimusta.

4.1 Toimintatutkimus

Toimintatutkimus on itseohjautuva toiminnan kehittämisen menetelmä, jossa lähestytään tutkittavaa kohdetta aktiivisen toiminnan näkökulmasta. Toimintatutkimuksessa projektin tekijäryhmä aloittaa ja toteuttaa tuotekehitystyön itsenäisesti. Tekijäryhmän jäsenillä on yhteinen ennalta asetettu tavoite, johon he haluavat löytää toimintatutkimuksen avulla ratkaisun. Toimintatutkimuksessa ainoastaan työn lopputulos on alisteinen esimerkiksi yritysjohdon tai muun toimintaan liittyvän tahon hyväksynnälle. (Zuber-Skerritt 2002, 125-126.)

Toimintatutkimus soveltuu erittäin hyvin ketterillä menetelmillä toteutettavaan ohjelmistokehitysprojektiin. Toimintatutkimuksen kaksi keskeistä tavoitetta: toiminnan kehittäminen ja

ongelmalliseksi koettuun toimintatilanteeseen vaikuttaminen (Suojanen 2019). Toimintatutkimukseen kuuluu aina taustateorioiden kerääminen ja niihin perehtyminen, sekä niistä saatujen kokemusten analysoiminen ja raportoiminen (Suojanen 2019). Iteratiivisesti etenevään ohjelmistokehitysprojektiin mallia voi soveltaa suoraan. Kehittäjäryhmä oppii työn kuluessa koko ajan lisää käsiteltävästä aiheesta. He raportoivat ja analysoivat prosessin aikana kohtaamiaan ongelmia ja menetelmiä, sekä muokkaavat kerääntyneen tiedon perusteella toimintatapojaan paremmin tavoitetta palvelevaan suuntaan.

4.2 Empiirinen tutkimus

Empiirinen tutkimus on ongelmanratkaisun menetelmä, jossa pyritään selvittämään tutkimuskohteen lainalaisuuksia kokeilemalla siihen liittyviä teorioita käytännössä. Näistä kokeista tarkkailemalla kerätty tutkimusaineisto on järjestelmällisesti hankittua reaalitietoa, joiden avulla voidaan vahvistaa osoitettujen teorioiden paikkansapitävyys. (MacKenzie 2012, 129.)

Ohjelmistokehityksessä päätökset käytettävien menetelmien ja tekniikoiden suhteen tulee perustua faktoihin eikä teoriaan tai mielipiteisiin (Sampaio & Sampaio 2019, 1.). Tämä tarkoittaa sitä, että empiiristä ja todisteisiin perustuvaa tutkimusta tarvitaan ohjelmistokehitysprojekteissa. Käytäntöön perustuvalla empiirisellä tutkimuksella pyritään hankkimaan tietoa jolla saadaan ratkaistua joku olemassa oleva käytännön ongelma, tai sen osa, joka parantaa mahdollisuuksia saavuttaa lopullinen tavoite. Käytäntöön perustuvan empirian ei tarvitse välttämättä perustua mihinkään olemassa olevaan valmiiseen teoriaan. (Sampaio & Sampaio 2019, 1.)

Tämän työn viitekehyksessä empiirinen tutkimus oli keskeinen tutkimusmenetelmä kehitystyön aikana käytettävää toteutustekniikkaa valittaessa. Suunnitelluista menetelmistä kerättiin teoriaa, ja teorioista pohjaavan empiirisen tutkimuksen avulla voitiin jo ennen varsinaisen kehitystyön aloitusta toteamaan riittävällä varmuudella, että valittava menetelmä soveltuu tarkoitukseen ja on oikea.

4.3 Hiljaisen tiedon tutkimus

Hiljaisella tiedolla tarkoitetaan yleensä ei-sanallista kokemuspohjaista tietoa, jota on kerääntynyt vuosien varrella esimerkiksi yrityksen työntekijöille toiminnallisen kokemuksen kautta. Tämä tieto/osaaminen on vaikeasti tunnistettavaa, koska yksilö ei usein edes itse tiedosta tietävänsä aiheesta jotakin sellaista, joka ei olisi yleisesti tiedossa. Hiljainen tieto on tiedon lajina erilainen kuin eksplisiittinen tieteellinen tieto. Sen symbolisia ja käsitteellisiä yhteyksiä ei ole määritelty, joten niitä ei ole käsitteellisessä mielessä olemassakaan ennen kuin ne on sellaiseksi määritelty. (Pohjalainen 2012, 1-10.)

Hiljainen tieto on tärkeässä asemassa ohjelmistokehitysprojekteissa. Kehitystyö perustuu ongelmien ratkaisemiseen ja jatkuvaan oppimiseen. Alan yritysten on erittäin tärkeää huolehtia

siitä, että projekteissa lisääntynyt osaamista ja tietämystä jaetaan yrityksen henkilöstön välillä. (Dreyer & Wynn 2016, 154-155)

4.4 Kerätyn tiedon validiteetti ja reliabiliteetti

Validiteetilla ilmaistaan kuinka hyvin ja kattavasti tutkimuksessa käytetty menetelmä mittaa sillä tutkittavaa asiaa. Validiteettia voidaan arvioida sen perusteella, kuinka täsmällisesti ja argumentoidusti sen avulla pystytään avaamaan tutkimusongelma. Reliabiliteetilla tarkoitetaan tutkimuksen luotettavuutta. Kvantitatiivisessa tutkimuksessa sillä arvioidaan tutkittavien mittareiden johdonmukaisuutta, jolla tarkoitetaan sitä, että mittari mittaa aina samaa asiaa, eli on konsistentti (KvantiMOTV, 2008).

Tämänkaltaisessa kehitysprojektissa validiteettia voidaan arvioida esimerkiksi tutkimuskysymystemme valossa ja kuinka täsmällisesti ne kuvasivat tutkimuksen avulla ratkaistua ongelmaa. Tutkimuksen reliabiliteettia voidaan arvioida tarkemmalla tasolla. Onko tutkimuksella pystytty perustellusti osoittamaan, että tutkimuksen perusteella valittu menetelmä oli työhön soveltuva ja tutkituista menetelmistä sopivin.

5 Menetelmän kartoitus empiirisen tutkimuksen avulla

Vaatimusmäärittelyssä keskeiseksi asiaksi nostettu ehdoton helppokäyttöisyys rajasi käytännössä pois hiiren ja näppäimistön liikkeitä seuraavan ja kuvantunnistukseen perustuvan sovelluksen. Vaihtoehtoa haluttiin silti tutkia vähintäänkin konseptimielessä ja se onnistuttiin demoamaan teknisesti toteutettavissa olevaksi ratkaisuksi (kokeiltiin ainoastaan Windows -ympäristössä). Menetelmän etuna olisi ollut skaalautuvuus työpöytäympäristössä käytettäville sovelluksille. Toisaalta sen käyttö myös vaatisi käyttäjältä selvästi enemmän syventymistä aiheeseen, sekä pakottaisi tämän lataamaan koneelleen ohjelmia, joita ei välttämättä halua tai edes osaa asentaa.

Kuvantunnistukseen perustuvan ratkaisun heikkoutena nähtiin myös teknisesti hankalampi toteutettavuus, sekä suppeammat toiminnalliset mahdollisuudet (verrattuna JavaScript -toteutukseen), koska sovelluksen koodilla ei olisi ollut pääsyä käsiteltävän websovelluksen sisäpuolelle. Idea koettiin konseptimielessä kiinnostavaksi, ja se nähtiin aiheena, jota on syytä opiskella ja ideoida lisää, mutta tähän opinnäytetyöhön liittyen se ei ollut em. syistä potentiaalinen vaihtoehto. Kuvantunnistukseen perustuvaa ratkaisua demottiin Python -ohjelmointikielellä mm. pyHook, pyautogui, win32, ja open cv -kirjastoja käyttäen.

Käytettävissä olevilla tutkimustiedoilla vaihtoehdot rajautuivat melko nopeasti toteutukseen, jossa työkalu injektoi oman JavaScript -koodinsa nauhoitettavaan käyttöliittymärajapintaan. Valittu menetelmä mahdollistaa suoran pääsyn käsiteltävän sovelluksen DOM:iin. DOM:ia manipuloimalla pystytään tarkasti ja luotettavasti poimimaan käyttäjän käyttöliittymään kohdistamat toimenpiteet. Opinnäytetyöntekijä ei onnistunut yrityksistään huolimatta löytämään

yhtään toimivaa ratkaisua, jossa vastaavanlainen ohjelma olisi onnistuneesti toteutettu jollain muulla kuin JavaScriptiin perustuvalla menetelmällä.

5.1 JavaScript injektio - Selenium WebDriver

Ensimmäisenä tutkittavana JavaScriptiin pohjaavana vaihtoehtona oli Selenium WebDriver, jonka avulla voidaan käskyttää web-selainta tekemään automaattisesti kaikki se minkä käyttäjäkin pystyy sillä tekemään. Selenium on websovellusten käyttöliittymän kautta toteutettavissa automaatio- ja RPA projekteissa, jos ei standardi, niin selvä ykkönen.

Selenium tukee suurinta osalle markkinoilla olevista selaimista ja on ohjelmoitavissa ainakin Pythonilla, Javalla, C#:lla, Rubylla, PHP:lla, Perlillä ja JavaScriptillä. Lisäksi se on integroitavissa moniin automaatio-frameworkkeihin kuten esimerkiksi Robot Framework, WebDriver js, TestNG. (SeleniumHQ 2018.)

Selenium tarjoaa mahdollisuuden ajaa ohjelman ulkopuolelta tulevaa JavaScriptiä selainikkunassa, joka sillä on hallussaan, joten injektointi sinänsä on teknisesti mahdollista. Lisäksi Selenium olisi mahdollistanut kehitettävän työkalun implementoinnin suoraan osaksi sitä käyttäviä automaatiokirjastoja ja myös helposti toteutettavan taustalogiikan ohjelmoinnin. Generoitu testiskripti olisi ollut nauhoituksen jälkeen ajettavissa suoraan työkalusta ilman erillisen rajapintatyökalun ohjelmointia.

Seleniumin käyttöön liittyen jäi ratkaisematta, olisiko sen avulla saanut hallittua tallennusta ja webohjelmalta tarvittavaa datavirtaa koko session ajan, ja kuinka helposti se olisi ollut tehtävissä. Lisäksi haasteena olisi ollut miten suoraan sivustoon injektoitu ja kaikkine tapahtumankäsittelijöineen suhteellisen kompleksinen JavaScript -olisi saatu eristettyä siten, että sen vaikutukset ohjelman itsensä toimintoihin olisivat minimalistiset.

Edellä mainittuihin kysymyksiin ei tutkimuksen aikana löytynyt selviä vastauksia, eikä Seleniumiin perustuvaa ratkaisua nähty tarpeelliseksi lähteä demoamaan. Tähän päätökseen vaikutti osin sekin, että edes Seleniumin oma – tämän opinnäytetyön aiheen kanssa vastaavanlainen toteutus - ei ole toteutettu heidän oman driverinsa avulla.

5.2 JavaScript injektio - Selainlaajennukset

Seleniumin lisäksi tutkinnassa olivat selainlaajennukset, johon käytännössä kaikki tämänkaltaiset sovellukset pohjautuvat. Selainlaajennukset tarjoavat rajapinnat, joiden avulla voidaan kuunnella ja käsitellä sekä selaimeen itseensä kohdistuvia tapahtumia, että siinä pyörivää sivustoa/ohjelmaa. (MDN web docs 2018; Chrome Developers 2018.)

Laajennuksia käytetään tyypillisesti lisäämään toiminnallisuuksia jollekin tietylle web-sivulle. Niiden avulla voidaan vaikkapa näyttää käyttäjälle ilmoituksia hänen sähköposteistaan tai tilatusta rss -syötteistään. Voidaan poistaa mainoksia web-sivulta, tai vaikka tehdä jotain pieniä käyttäjäystävällisiä lay-out muutoksia tämän niin halutessa. (Rouse 2018.)

Vaikka laajennuksen tyypillinen käyttötarkoitus saattaakin luoda toisenlaisen mielikuvan, niin kyseessä on varsin voimakas työkalu, jonka avulla kehittäjä saa kontrollin lähes kaikkeen selaimessa tapahtuvaan. Opinnäytetyön aiheena olevaan sovellukseen liittyen laajennus tarjoaa helpon tavan injektoida sovelluksen oma ohjelmakoodi mihin tahansa selaimessa pyörivään web-ohjelmistoon. Injektoitumisen lisäksi rajapinta mahdollistaa selaintapahtumien kuuntelemisen. Näin sen avulla on mahdollista kontrolloida ohjelman suorituksen aikaista tapahtumavirtaa paitsi itse sovelluksen kautta, niin myös selaimessa tapahtuvien muutosten kautta.

Erittäin tärkeänä laajennusta puoltavana tekijänä nähtiin sekin, että laajennuksella web-ohjelmistoon injektoitu Content Script toimii eristetyssä hiekkalaatikossa ohjelman ulkopuolella, eivätkä siihen tehdyt muutokset muuta sivuston sisäistä JavaScriptiä (Chrome Developers 2018). Työn aiheena olevassa ohjelmassa koko toiminta perustuu hyvin pitkälti erilaisten kuuntelijoiden lisäämiseen, muokkaamiseen ja toiminnan ohjaamiseen, sekä näitä kuuntelijoita laukaisevien tapahtumien ja elementtien tunnistamiseen. Tietämättä kohdesovelluksen sisäistä rakennetta, olisi kokonaisuutta hyvin vaikeaa, ellei mahdotonta, hallita eristämättömässä ympäristössä vaikuttamatta samalla sovelluksen toimintaan.

Heikkoutena laajennuksiin liittyen voi jossain tapauksissa pitää laajennusta itseään. Laajennus, kuten myös JavaScript, asettaa tarkat raamit sille mitä voidaan tehdä. Se myös pakottaa kehittäjän tekemään asiat tavalla, jolla laajennus haluaa ne tehtävän, eikä tavalla, jolla kehittäjä haluaisi asian tehdä.

Laajennus asettaa rajoitteita mm. käyttöliittymään ja sitä kautta käytettävyyteen liittyen, ohjelmointitekniikoihin yms. Ennen selainlaajennuksessa toimivan ohjelman suunnittelua on hyvä käydä tarkasti kehyksen sisällön turvallisuuteen liittyvät määräykset ja dokumentaatio läpi. Tällöin jo suunnitteluvaiheessa pystytään huomioimaan asiat, jotka määrittelevät miten ohjelman tapahtumavirta voidaan hallita käytössä olevassa viitekehyksessä. (Chrome Developers 2018.)

Toisena isona heikkoutena on selainriippuvuus. Mikäli tuki halutaan useammalle selaimelle, niin kehittäjä joutuu tekemään oman version sovelluksesta kullekin erikseen.

5.3 Selainlaajennuksen todistaminen validiksi menetelmäksi

JavaScript -ohjelmointikielellä toteutettava selainlaajennus oli tutkimuksen perusteella joistain heikkouksistaan huolimatta selvästi järkevin toteutustapa. Se osoittautui myös ainoaksi

vaihtoehdoksi, joka tiedettiin tutkimuksen perusteella toteuttamiskelpoiseksi ilman pitkää listaa lisäselvitystä vaativista asioista.

Menetelmällä toteutettu demo oli vahvistus tutkimuspäätelmien oikeellisuudelle. Päämääränä oli tallentaa muutama erilainen käyttötapaus ja luoda niistä Pacewords -tekniikan mukainen testiskripti. Demon avulla voitiin viimeistään vakuuttua siitä, että valittu menetelmä on oikea. Tutkimuksen kannalta demo nähtiin tärkeänä myös siksi, että sen avulla päästäisiin kiinni mahdollisiin uusiin tutkimuskysymyksiin, joita ei oltu otettu huomioon tutkimuksen alkuvaiheessa, mutta jotka pitäisi joka tapauksessa selvittää ja ottaa huomioon viimeistään sovelluksen teknisessä suunnitteluvaiheessa.

Demo syntyi noin viikossa. Sen aikana tuli laajennuksen toimintaan liittyen useampikin, lähinnä tapahtumavirran hallintaan liittyvä asia ilmi, joita ei pelkästään teorian ja käytössä olleiden dokumentaatioden perusteella osattu ottaa huomioon, joten se oli työvaiheena erittäin hyödyllinen. Kuviossa 8 esitellään demo-ohjelman käyttöliittymä ja ensimmäinen generoitu testiskripti.

Kuvio 8: Demo-ohjelman käyttöliittymä ja generoitu testiskripti

Tutkimuksen jälkeen päätettäväksi jäi, minkä selainlaajennuksen ympärille ensimmäistä versiota ohjelmasta aletaan tekemään. Loppusuoralla olivat mukana Firefox ja Chrome, joista jälkimmäinen tuli valituksi lähinnä sen vuoksi, että suurempi osa testiautomaatioinsinööreistä katsoi sen kyselytutkimuksen perusteella mielekkäämmäksi työskentely-ympäristöksi.

6 Kehitystyön toteutus

Tässä osiossa kuvaillaan opinnäytetyön aiheena olevan sovellusta ja sen kehitysprosessia yksityiskohtaisemmin. Aluksi avataan hieman projektin toteutustapaa, jonka jälkeen käydään läpi selainlaajennus, sen arkkitehtuuri, ja keskeisimpien komponenttien toiminta.

6.1 Aikataulu ja projektin kulku

Demo ja iso osa sovelluksen suunnittelusta toteutettiin kesällä 2018. Kehitystyön opinnäytetyötä koskeva osuus tapahtui kokonaisuudessaan syksyn 2018 aikana. Opinnäytetyöntekijä

työskenteli eräänlaisena yhden miehen Scrum tiiminä, käyden työn tilaajan kanssa säännöllistä dialogia projektin etenemisen suhteen. Tarkkaa aikataulua tai kiinteän pituisia sprinttejä ei tekijän muista toimeksiannoista johtuen ollut. Sprintit tehtiin pääasiassa noiden toimeksiantojen välillä ja viikonloppuisin.

Projekti eteni demovaiheen jälkeen siten, että tekijä valmisteli siihen pohjautuen suunnitelman ja ensimmäinen backlogin. Ensimmäinen backlog ei sisältänyt yhtään uutta ominaisuutta vrt. demovaihe, vaan sen aikana oli tarkoitus hioa ohjelman perusta kuntoon. Pääasiassa tämä sisälsi sellaisten mekanismien opiskelemista ja rakentamista, joiden avulla voitiin hallita QRecin suorituksenaikaista tapahtumavirtaa siten, että käsiteltävä websovellus pysyy jatkuvasti monitoroituna, vaikka sen dynaaminen sisältö muuttuu suorituksen kuluessa.

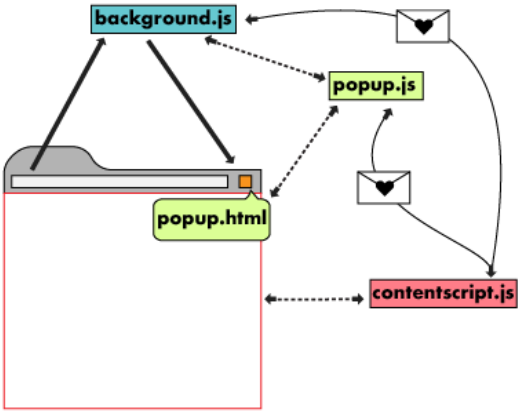
Ensimmäisen backlog osoittautui selvästi suuritöisimmäksi projektin aikana. Sen valmistuttua sovelluksen katsottiin reagoivan sekä selaimen kertomiin, että sovelluksen DOM -rakenteessa tapahtuviin muutoksiin niin hyvin, että varsinaisten toiminnallisuuksien implementointi voitiin aloittaa. Kuhunkin sprinttiin otettiin implementoitavaksi 1-2 ominaisuutta kerrallaan ja kunkin implementointivaiheen jälkeen oli n. viikon pituinen testausvaihe. Vaiheen aikana voitiin todentaa, ettei mitään ole mennyt rikki, ja toisaalta myös se, että lisätty ominaisuus toimii oletetusti. Havaitut virheet korjattiin pois ennen seuravan kierron aloittamista.

Edellä kuvatulla kierrolla projekti eteni marraskuun puoleenväliin asti. Tuolloin työkalun katsottiin olevan ominaisuuksiensa ja vakautensa osalta sellaisella tasolla, että ensimmäinen julkaisuvalmis beta meni ”tuotantotestiin” isommalle joukolle ihmisiä.

6.2 Chrome -selainlaajennuksen arkkitehtuuri

Sovellus toteutettiin Chrome -selainlaajennuksena. Selainlaajennus sisältää monta eri tehtäviä varten tarvittavaa ja toisiinsa kytkeytyvää komponenttia, jotka tarjoavat sovelluskehittäjälle käytettäväksi ison joukon rajapintoja ja metodeita. Selainlaajennus muodostuu manifest.json tiedostosta, Background Scriptistä, Content Scriptistä ja UI elementistä (popup.html + sen JavaScript popup.js), jotka ovat laajennuksen viisi pääkomponenttia. Laajennuksen komponentit kehitetään webteknologioita käyttäen (HTML, CSS, JavaScript, JSON) ja niillä on oma yksilöllinen käyttötarkoituksensa.

Kuviossa 9 esitellään Chrome-laajennuksen arkkitehtuuri. Pääkomponenteista popup.html on normaali, joskin maksimikokoon 600px*800px rajattu, html -sivu, joka toimii sovelluksen käyttöliittymänä. popup.js on kyseisen html-sivun toiminnallisuuksia ohjaava JavaScript-tiedosto. contentscript.js on JavaScript -tiedosto, joka injektoidaan selainikkunassa pyörivään ohjelmaan. background.js on JavaScript -tiedosto, jolla on pääsy selaimeen ja esimerkiksi chrome.runtime API:iin. Script komponentit voivat kommunikoida keskenään Message Passing -tekniikan avulla. (Chrome Developers 2018.)



Kuvio 9: Chrome selainlaajennuksen arkkitehtuuri ja komponenttien välinen viestiminen (Chrome Developers 2018)

6.2.1 Laajennuksen käyttöoikeuksien määrittely – manifest.json

Ennen kun laajennusta otetaan käyttöön, sille pitää määritellä tarvittavat oikeudet. Oikeuksien määrittely tapahtuu manifest.json -tiedostossa, jonka tarvitsee olla mukana jokaisessa laajennukseksi tarkoitetussa ohjelmassa. Tiedostossa määritellään sovelluksen manifest-versio, ohjelman nimi ja versio, käyttöoikeudet, background scriptit, content scriptit, sekä se mille url-osoitteille oikeudet annetaan ja missä vaiheessa sivun latausta sivu injektoidaan. (Kuvio 10)

```
{
    "manifest_version": 2,
    "name": "MyAPP",
    "version": "1",
    "permissions": [
      "tabs",
      "activeTab",
      "contextMenus",
      "downloads",
      "webNavigation",
      "notifications",
      "storage",
      "unlimitedStorage",
      "debugger",
      "downloads",
      "<all_urls>"
    ],

    "background": {
      "scripts": ["js/jquery-3.3.1.min.js","js/background.js", "js/popup.js"],
      "persistent": false
    },
    "content_scripts": [
      {
        "matches": [
          "<all_urls>"
        ],
        "js": ["js/jquery-3.3.1.min.js", "js/content.js", "js/finder.js"],
        "all_frames": true,
        "run_at": "document_idle"
      }
    ],
    "browser_action": {
      "default_icon": {
        "16":"img/logo.png"
    },
      "default_popup": "MyApp.html",
      "default_title": "Some App"
    }

}
```

Kuvio 10: Laajennuksen oikeudet (manifest.json)

6.2.2 Laajennuksen pääkomponentit - Background Scripts

Background script on työn kohteena olevan sovelluksen suorituksenaikaisen kontrolloinnin osalta keskeinen komponentti, koska se tarjoaa chrome.tabs API:n kautta sovellukselle välttämätöntä tietoa selaimelta. Tämänkaltaista tietoa on esimerkiksi sivun päivittyminen tai aktiivisen selainikkunan vaihtuminen ohjelman suorituksen aikana. Sovelluksen toiminnan ja sen lopettamisen pitää olla käskytettävissä sen omasta popup-käyttöliittymästä, joten kuuntelijat asettavaa content.scriptin funktiota ei voida suorittaa sokkona aina kun selain päivittyy, tai uusi sivu avataan. Näin toimiessa sovellus olisi aktiivinen aina sen jälkeen, kun se on ensimmäisen kerran käynnistetty. Sovelluksen tilan hallitsemiseen tarvitaan erillinen mekanismi.

Background Script asetetaan kuuntelemaan selaimessa tapahtuvia muutoksia, jotka vaikuttavat käsiteltävän ohjelman tilaan ja viestii niistä Content Scriptille. Content Script seuraa viestiliikennettä, ja osaa siihen sekä omaan sen hetkiseen tilaansa perustuen tehdä kulloisessakin tilanteessa päätöksen tarvitseeko kuuntelijoita lisätä aktiivisen ikkunan elementeille, vai

onko sovellus tilassa, jossa sen ei oletetakaan tekevän mitään ilman käyttäjän erikseen antamaa käskyä.

Kuviossa 11 nähdään background-scriptiin koodattu funktio. Funktio käyttää tabs.apia ja lähettää viestin muiden komponenttien poimittavaksi aina kun selainikkuna päivittyy.

```
chrome.tabs.onUpdated.addListener( function (tabId, changeInfo, tab) {
    if (changeInfo.status === 'complete') {
        chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
            var activeTab = tabs[0];
            /**
             * Funktio lähettää viestin kuuntelijalle joka kerta kun selainikkuna
             * päivittyy
             */
            chrome.tabs.sendMessage(activeTab.id, {data: "tab_is_updated"});
        });
    }
});
```

Kuvio 11: Funktio, joka reagoi selaimen päivittyessä

Kuviossa 12 näkyy chrome.tabs API:n tarjoamia metodeita, joiden avulla voidaan kuunnella käytännössä jokaista selainkehyksen tapahtumaa. API tarjoaa rajapinnan myös kautta selaintapahtumien ohjaamiseen.

Methods

- get – `chrome.tabs.get(integer tabId, function callback)`
- getCurrent – `chrome.tabs.getCurrent(function callback)`
- connect – `runtime.Port chrome.tabs.connect(integer tabId, object connectInfo)`
- sendRequest – `chrome.tabs.sendRequest(integer tabId, any request, function responseCallback)`
- sendMessage – `chrome.tabs.sendMessage(integer tabId, any message, object options, function responseCallback)`
- getSelected – `chrome.tabs.getSelected(integer windowId, function callback)`
- getAllInWindow – `chrome.tabs.getAllInWindow(integer windowId, function callback)`
- create – `chrome.tabs.create(object createProperties, function callback)`
- duplicate – `chrome.tabs.duplicate(integer tabId, function callback)`
- query – `chrome.tabs.query(object queryInfo, function callback)`
- highlight – `chrome.tabs.highlight(object highlightInfo, function callback)`
- update – `chrome.tabs.update(integer tabId, object updateProperties, function callback)`
- move – `chrome.tabs.move(integer or array of integer tabIds, object moveProperties, function callback)`
- reload – `chrome.tabs.reload(integer tabId, object reloadProperties, function callback)`
- remove – `chrome.tabs.remove(integer or array of integer tabIds, function callback)`

Kuvio 12: chrome.tabs API:n metodeita (Chrome Developers 2018)

Perustoiminnallisuuden kannalta toinen tärkeä selainlaajennuksen tarjoama API QReciin liittyen on chrome.runtime, jota tarvitaan esimerkiksi viestien välitykseen ohjelman eri komponenttien välillä. Chrome.runtime tarjoaa tabs API:sta poiketen joitain metodeita ja eventtejä myös content scriptien käyttöön. Kuviossa 13 esitelty esimerkkifunktio reagoi Content scriptin

ja käyttöliittymän lähettämiin viesteihin päivittämällä tai tyhjentämällä käyttäjälle notifikaatioita näyttävän ikonin. Notifikaatio-ikoni sijaitsee selainkehyksessä, joten sen toimintaa hallitaan background scriptin kautta.

```
chrome.runtime.onMessage.addListener(function(msg, sender, response){
    if(msg.data === "new_step") {
        chrome.browserAction.setBadgeBackgroundColor({color: [255, 69, 0, 255]});
        chrome.browserAction.setBadgeText({text: msg.counter});
    } else if(msg.data === "reset_counter"){
        chrome.browserAction.setBadgeText({text:""});
    }
});
```

Kuvio 13: Runtime API:a käyttävä funktio

Laajennuksia suunniteltaessa on huomioitavaa, että background.js on tarkoitettu vain ja ainoastaan itse selaimen kuunteluun ja manipuloimiseen, eikä sillä ole pääsyä selaimessa pyörivään sivuun / ohjelmistoon. Mikäli selaimessa pyörivään ohjelmistoon halutaan pääsy, niin siihen tarvitaan Content Scriptiä. (MDN web docs 2018.)

6.2.3 Laajennuksen pääkomponentit - Content Scripts

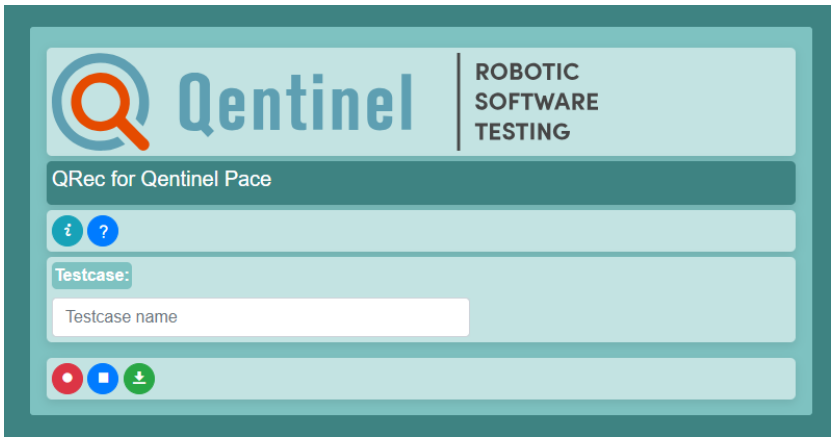
Content Scriptit ovat JavaScript tiedostoja, jotka injektoituvat selainikkunassa pyörivään sivustoon / web-sovellukseen. Content Script voi tehdä käytännössä kaiken sen minkä JavaScriptiä käyttäen pystyy normaalistikin tekemään käyttäen standardia DOM API:a. On kuitenkin huomioitavaa, että Content Scriptillä ei voi poistaa tai muokata sivuston omien scriptien toiminnallisuuksia, eikä sillä ole pääsyä sivuston sisäänrakennettujen scriptien muuttujiin. Content Script ei edes näe sivuston omaa JavaScriptiä, eikä voi käyttää esimerkiksi siellä olevia kirjastoja. Halutessaan käyttää kirjastoa Content Scriptissä, kehittäjän pitää injektoida se tiedoston mukana. Tämä tapahtuu määrittelemällä käytetyt kirjastot manifest.json -tiedostossa. Aiemman kuvion 8 manifest.jsonissa on tällä tavalla otettu jQuery -kirjasto käyttöön.

Toisin kun aiemmin käsitellyillä Background Scripteillä, Content Scripteillä ei ole pääsyä kuin muutamiin Chrome API:n tarjoamiin palveluihin, joista keskeisimpinä chrome.runtime API. API tarjoaa onMessage eventin ja sendMessage metodin, joiden avulla Content Script voi kommunikoida viestein muiden komponenttien kanssa. (MDN web docs 2018; Chrome Developers 2018.)

6.2.4 Laajennuksen pääkomponentit - UI Popup ja popup.js

UI popup tarjoaa keinon rakentaa laajennukselle käyttöliittymä. Se on normaali, maksimikooltaan rajoitettu (leveys 600px * pituus 800px) HTML-sivu, joka määritellään manifest.json -tiedostossa. Käyttöliittymän rakentaminen tapahtuu samoin kuin normaalin HTML -sivun ja se tukee kaikkia JavaScript kirjastoja ja CSS Frameworkeja.

Opinnäytetyön tekijä käytti tässä työssä käyttöliittymän tekemiseen HMTL-Bootstrap yhdistelmää. Kuviossa 14 esitellään sovelluksen käyttöliittymän layout.



Kuvio 14: Sovelluksen HTML + Bootstrap käyttöliittymä

Popup toimii itsenäisenä komponenttinaan, jolla voi olla normaalin html -sivuston tapaan mitä tahansa omia JavaScript toiminnallisuuksiaan. JavaScriptiä tarvitaan myös viestimiseen Background Scriptien ja Content Scriptien kanssa. Kuvio 15 esittelee funktion, joka suoritetaan, kun UI:n ’Pause’-nappulaa painetaan. Funktio antaa informaatiota käyttäjälle käyttöliittymän kautta, sekä kertoo muulle ohjelmalle suorituksen olevan ’wait’-tilassa.

```
$('#pause_btn').click(function (event) {
    /**
     * Pass message to content script that session is on hold,
     * Enable input field so it's ready for new session
     */
    var text = 'pause';
    $('#title_input').prop('disabled', false);
    $('#title_input').attr("placeholder","Write header for next test before continue");
    $('#alert').html('Session is on hold. You can continue later with another test. Press stop to finish s
    $('#pause_btn').css('display','none');
    $('#start_btn').css('display','inline-block');
    chrome.storage.local.set({sessionStatus:"wait"});
    chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
        chrome.tabs.sendMessage(tabs[0].id, {data: text}, function(response) {
            console.log('Session on hold');
        });
    });
});
```

Kuvio 15: Pause-napin suorittama funktio QRecin käyttöliittymässä

6.2.5 Pääkomponenttien välinen kommunikointi - Message Passing

Chrome laajennuksen komponenteilla on erilaiset toiminnallisuudet käytettävissään, eikä niillä ole mahdollisuutta päästä käsiksi toistensa muuttujiin tai funktioihin. Käytönaikaiseen tapahtumien hallintaan tarvitaan jokin yhteinen rajapinta, joka Chromen selainlaajennuksessa on toteutettuna runtime.api:n viestien avulla.

Content Scriptin injektoituminen web-sovellukseen on QRecin tapauksessa automatisoitu manifest.json-tiedostossa. Sovellus ei tee normaalitilassaan mitään, koska tähän JavaScript ohjelmakoodiin ei ole ohjelmoitu toiminnallisuutta, joka voisi herättää sovelluksen ilman käyttäjän antamaa käskyä. Käskyn saatuaan se luo omat toiminnallisuutensa nauhoitettavaan sivuun, jonka jälkeen Background Script pitää sovelluksen toiminnassa, kunnes käyttäjä haluaa lopettaa. Kaikki käskyjen välittäminen ja komponenttien välinen kommunikointi tapahtuu viestein. Viestit lähetetään JSON -objekteina, joten ne voivat käytännössä sisältää lähes mitä tahansa informaatiota, joka halutaan välittää komponentilta toiselle.

6.3 Muut keskeiset tekniikat ja rajapinnat

Edellisissä kappaleissa esitellyt selainlaajennuksen peruskomponentit muodostavat pohjan tässä opinnäytetyössä esitellylle sovellukselle. Kokonaisarkkitehtuuri sisältää myös muita webteknologian tekniikoita, joista sovelluksen toiminnan kannalta keskeisimmät ovat kuvattu seuraavissa aliotsikoissa.

6.3.1 Chrome Storage

Chromen tarjoama Storage API muistuttaa keskeisiltä ominaisuuksiltaan HTML5 Web Storagen tarjoamia local- ja session storage -palveluita. Palvelut mahdollistavat datan säilömisen selaimeen. Tyypillisesti tämä tarkoittaa esim. muuttujien arvoja, joiden avulla ohjelma saadaan palautettua vaikkapa selainikkunan päivittymisen jälkeen täsmälleen samaan tilaan missä se oli ennen päivittymistä.

JavaScript toimii vain selaimessa, joten se ei voi tallentaa tietoa käyttäjän tietokoneelle. Ilman serveripäätä pyörivään JavaScript -ohjelmaan tarvitaan aina jonkinlainen selaimen käytettävissä oleva tietovarasto, mikäli ohjelman suorituksen aikana on odotettavissa tilanteita, joissa sivu ladataan uudelleen, tai käyttäjä navigoi toiselle sivulle. (MDN web docs 2018.)

HTML5:n local- ja session storagessa on rajoitteita, joista kriittisimmät tähän opinnäyteyöhön liittyen olivat sen domain-, ja protokollariippuvuus. Tämän vuoksi ne eivät sopineet opinnäytetyön sovelluksen käytettäväksi. (w3schools.com 2018.)

Chromen tarjoamassa Storagessa domain- ja protokollariippuvuutta ei ole. Koska serveripään ratkaisua ei ainakaan tässä vaiheessa projektia nähty tarpeelliseksi toteuttaa, niin ajon aikainen tiedon liikuttaminen hoidetaan Chrome Storagen kautta. Chrome Storage API on avoin

kaikille selainlaajennuksen komponenteille, joten se on viestien ohella toinen yhteinen rajapinta, jonka avulla sovelluksen eri osat voivat saavat informaatiota toisistaan (Google Developers 2018).

Ajon aikaisessa Storagen käytössä on huomioitavaa, että sen get -metodin arvon palauttaa aina Callback -funktio. Callback -funktio on asynkronoitu, joka tarkoittaa sitä, että sitä ei välttämättä suoriteta heti. Koko ohjelmaa ei pysäytetä odottelun ajaksi, vaan muu suoritus jatkuu normaalisti. Ohjelman rakenteen suunnittelun kannalta tämä tarkoittaa sitä, että luotettavuuden varmistamiseksi Callback funktioiden luonne on huomioitava. Synkronoituja rakenteita - joiden oikeanlainen toiminta on välittömässä riippuvaisuussuhteessa asynkronoidun get funktion palauttamaan arvoon – ei tule käyttää (Kantor 2018; MDN web docs 2018).

6.3.2 Mutation Observer

MutationObserver Web API tarjoaa mahdollisuuden tarkkailla parametrein rajattavia DOM -puun sisäisiä muutoksia sovelluksen suorituksen aikana. MutationObserverin observe() -metodin avulla QRecin injektoima content script saa tiedon käsiteltävän web-sovelluksen sisäisistä muutoksista, ja osaa reagoida niihin tarpeen mukaan (W3C 2015; MDN web docs 2018).

Kuviossa 16 esitetään Mutation Observer. Se tarjoaa metodit tarkkailuun, nauhoitukseen sekä näiden toimintojen katkaisemiseen. Tarkkailtavat mutaatiot asetetaan parametrein.

```
[Constructor(MutationCallback callback)]
interface MutationObserver {
  void observe(Node target, MutationObserverInit options);
  void disconnect();
  sequence<MutationRecord> takeRecords();
};

callback MutationCallback = void (sequence<MutationRecord> mutations, MutationObserver observer);

dictionary MutationObserverInit {
  boolean childList = false;
  boolean attributes;
  boolean characterData;
  boolean subtree = false;
  boolean attributeOldValue;
  boolean characterDataOldValue;
  sequence<DOMString> attributeFilter;
};
```

Kuvio 16: Mutation Observerin hyväksymät parametrit ja niiden oletusarvot (W3C 2015)

QRec sovelluksen pitää pystyä reagoimaan kohdesovelluksen dynaamisesti muuttuvaan ympäristöön. Tämä tarkoittaa, että sovelluksen tulee osata lisätä kuuntelijat jokaiselle sivulla olevalle elementille riippumatta siitä ovatko elementit siellä jo sivun latautuessa, vai ilmestyvätkö myöhemmin esimerkiksi jonkun taustalla pyörivän AJAX -prosessin, tai käyttäjän laukaiseman tapahtuman jälkeen. Mutation Observer on sovelluksen toiminnan kannalta keskeisessä asemassa, jotta käsiteltävän ja dynaamisesti muuttuvan UI:n jokainen elementti pysyy kuunneltuna. Background Script kertoo muutoksista, jotka kohdistuvat selainikkunaan asti. Mutation Observer tekee saman sovelluksen DOM:in ja CSS:n osalta, joka mahdollistaa sen, että kaikki dynaamiset muutokset ovat kehittäjän kontrolloitavissa.

7 Osien niputtaminen ohjelmaksi

Luvussa 5 on kuvailtu komponentit, jotka toimivat tämän opinnäytetyön aiheena olevan QRec -sovelluksen runkona ja muodostavat sen ytimen. Ytimen päälle on implementoitu toiminnallisuudet, jotka toteuttavat vaatimusmäärittelyssä asetettuja tehtäviä. Tarkoituksena ei ole kuvata kooditasoisin esimerkein miten yksittäiset toiminnallisuudet ovat toteutettu, tai logiikkaa, jonka perusteella vaikkapa DOM -elementtien kuunteleminen ja niistä haluttujen attribuuttien poimiminen tässä yhteydessä toimii, vaan enemmänkin kuvailla tapahtumavirran hallintaa yleisellä tasolla.

7.1 Valmiin ohjelman toiminta

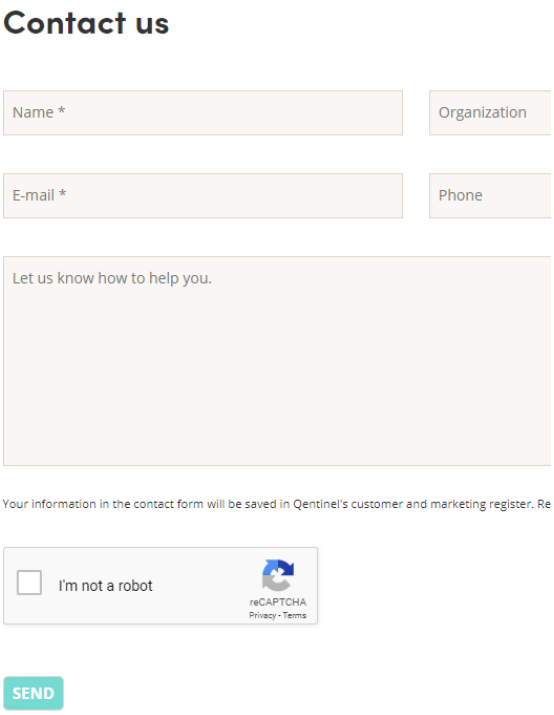
Ohjelman nauhoittaa käyttäjän suorittamia käyttötapauksia ja poimii kunkin tapauksen aikana kaikki elementteihin kohdistuvat toimenpiteet. Jokaisesta poimitusta toimenpiteestä generoidaan rivi Pacewords -tekniikalla toteutettua automaatioskriptiä. Nämä yksittäiset rivit muodostavat ketjuna kyseisen käyttötapauksen robottikielisen version, eli suoritusvalmiin testiautomaatioskriptin.

Oikeanlaisen skriptirivin luodakseen sovellus osaa poimia kullekin tapahtumalle oikean Pacewordin (esim. ClickText, Dropdown, TypeText, VerifyText etc.). Tämä tapahtuu tapahtuman laukaisseen elementin tyypin, ja käyttäjän siihen suorittaman toimenpiteen perusteella. Pacewordin luomisen lisäksi sovellus etsii elementille oikean teksti-, tai attribuuttitunnisteen, jonka perusteella Pace -robotti osaa tunnistaa elementin oikein, kun skripti suoritetaan. Tekstiä vastaan ottavien elementtien sisältä tallennetaan lisäksi käyttäjän elementtiin syöttämä teksti.

Mikäli oikeanlaista elementtitunnistetta ei ole saatavilla, niin elementistä luodaan Robot Framework -muuttuja, johon sijoitetaan arvoksi tämän xpath-tunniste. Helpommin luettavan skriptisyntaksin vuoksi testiskriptiin sijoitetaan ainoastaan muuttujan nimi, ja robottilogiikan sisältävään tiedostoon luodaan toinen rivi, jossa on itse sijoitusoperaatio.

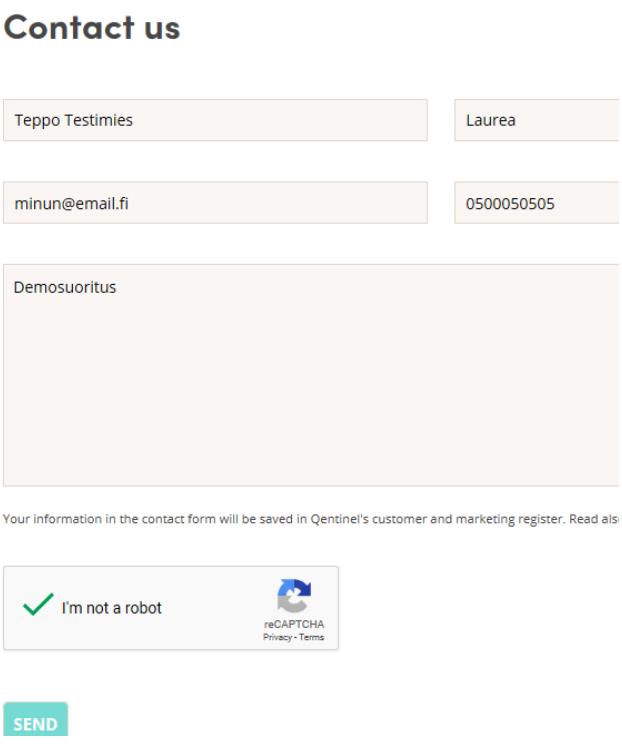
Seuraavissa kuvioissa esitetään täytettävä HTML-palautelomake, sekä sen täyttämisestä QRecillä generoitu automaatioskripti. Sovellus luo kaksi .robot tiedostoa, joista ensimmäisessä on käyttötapaus, ja jälkimmäisessä on kaikki kyseiseen tapaukseen liittyvä robottilogiikka. Käyttötapauksia voidaan nauhoittaa monta peräkkäin ilman, että niistä tarvitsee luoda uusia tiedostoja. Käyttäjän lopettaessa hän nimeää projektin haluamallaan tavalla ja tallentaa tiedot kovalevylleen. Kovalevylle tallennetut skriptit ovat välittömästi suoritettavissa.

Kuviossa 17 lomake ennen kun siihen on suoritettu toimintoja. Tässä vaiheessa käyttäjä asettaa QRec sovelluksen päälle.



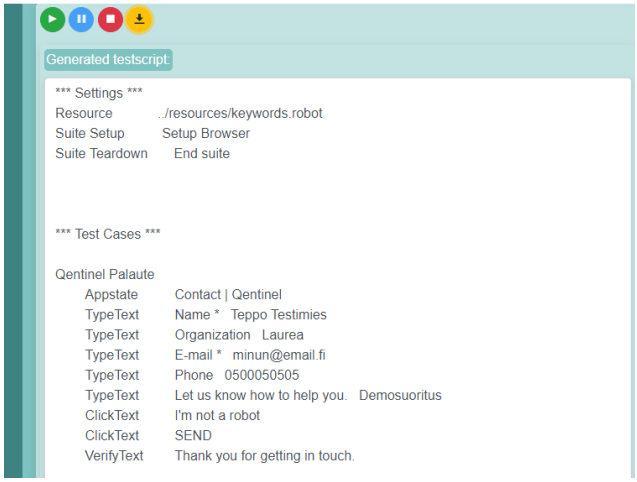
Kuvio 17: Täytettävä palautelomake (Qentinel 2018)

Kuvio 18 esittää lomaketta täytettynä. Käyttäjä on kirjoittanut tietonsa, sekä klikannut i'm not a robot-tekstin sisältävää laatikkoa ja SEND-tekstin sisältävää tekstiä.



Kuvio 18: Täytetty lomake (Qentinel 2018)

Kuviossa 19 esitetään lomakkeen täyttämisestä generoitu automaatioskripti. Appstate sisältää käyttäjän navigoinnin lomakkeelle. Tämän jälkeen lomakkeen täyttö, jossa TypeText kuvaa kirjoittamista, sille annettava ensimmäinen parametri (esim. Name*) kenttää, johon kirjoitetaan, ja toinen parametri tekstiä, joka kirjoitetaan (esim. Teppo Testimies). ClickText kuvaa sille parametrina annetun tekstin klikkaamista ja VerifyText vahvistaa, että ruudussa lukee haluttu teksti.

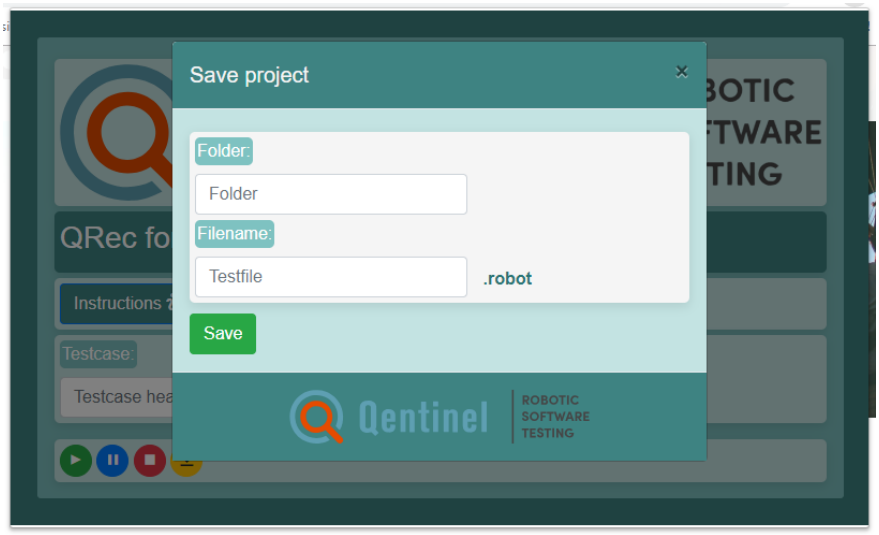


Kuvio 19: Käyttötapaus

Kuviossa 20 on QRecin edellä kuvatulle automaatioskriptille generoima robottilogiikan sisältävä tiedosto. Tiedosto sisältää koko automaatiotiedoston alustamiseen ja lopettamiseen sisältävät toimet sekä yksittäisen käyttötapauksen alustamiseen liittyvät toimet.

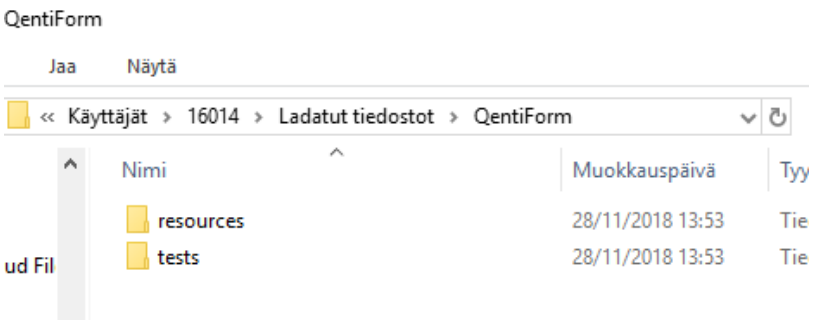
Kuvio 20: QRec -sovelluksen generoima keywords.robot.

Valmiit robottitiedostot voidaan tallentaa levylle painamalla käyttöliittymän download -nappia. Nappi avaa dialogin, jossa käyttäjä antaa haluamansa nimen sekä projektille, että käyttötapaukset sisältävälle tiedostolle (kuvio 21).



Kuvio 21: Käyttöliittymä download -napin painamisen jälkeen

Tallennus tapahtuu siten, että tekstikentissä olevista skriptit muunnetaan JavaScriptiä käyttäen .robot päätteisiksi tiedostoiksi, jonka jälkeen sovellus muodostaa projektille kuviossa 22 esitettävän kansiohierarkian. Pääkansio, joka on käyttäjän yllä nimeämä ”QentiForm”, jonka alla tests-kansio, joka sisältää automaatioskriptin, sekä resources-kansio, joka sisältää robottilogiikkaa, muuttujia yms. sisältävän tiedoston.



Kuvio 22: QRecin tallentama kansio kovalevylle tallennettuna

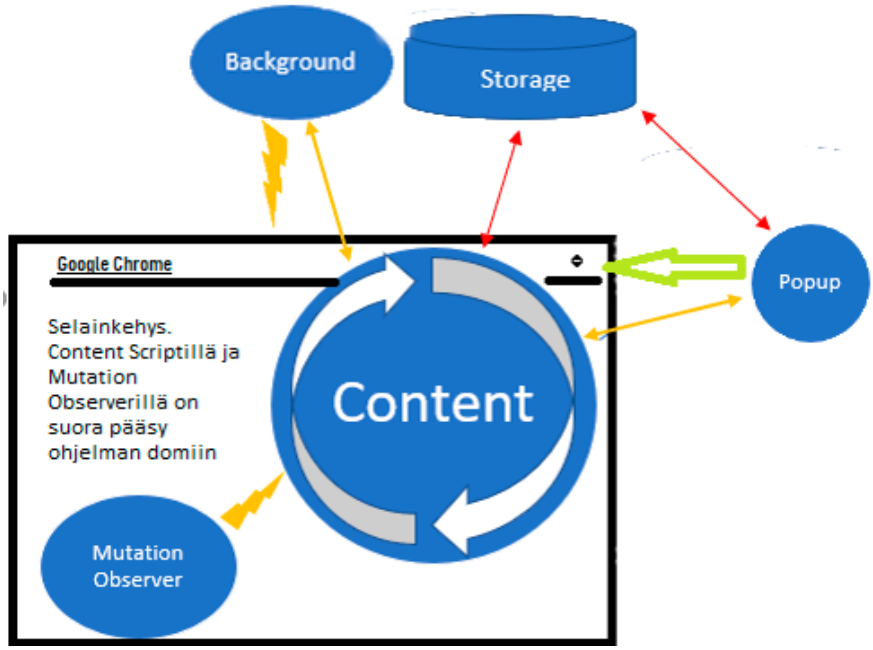
Ohjelma toimii tällä hetkellä ainoastaan asiakaspäässä. JavaScriptillä ei ole sellaisenaan pääsyä käyttäjän tiedostojärjestelmään, joten kansiorakenteen-, ja tekstitiedostojen luomiseen sekä niiden tallentamiseen käytetään HTML Downloads API:a. API tarjoaa rajapinnan, jonka avulla JavaScriptillä voidaan tallentaa tiedostoja käyttäjän selaimelleen määrittelemään lataus -kansioon, joka tyypillisesti on Downloads/Ladatut tiedostot -niminen.

7.2 Toimintaperiaate

Luvussa 5 kuvatut komponentit muodostavat sovelluksen arkkitehtuurin. Kokonaisuudessa sen eri tehtäviä suorittavat osaset ovat liitetty yhteen tässä kappaleessa kuvatulla tavalla.

Content Script on sovelluksen sydän. Se toimii kuten mikä tahansa käsiteltävän web-ohjelman oma JavaScript -tiedosto. Se on injektoitu käsiteltävään sovellukseen ja sisältää kaiken sellaisen QRecin tarvitseman toiminnallisuuden, jota tarvitaan tapahtumien tallentamiseen ja skriptirivien generoimiseen. Tiedosto sisältää funktiot, joiden avulla elementeille asetetaan tapahtuman kuuntelijat sekä funktiot, jotka ohjaavat ohjelman toiminnallisuuksia. Content Script sisältää myös Mutation Observerin. Kaikki komponenttien välinen viestiliikenne kulkee Content Scriptin kautta. Koska serveripäätä ei ole, niin Storage tarvitaan säilömään tieto session tilasta, sekä jo generoidusta skriptiriveistä.

Kuvio 23:ssa esitetyt keltaiset nuolet kuvaavat komponenttien välistä kahdensuuntaista viestiliikennettä. Punaiset nuolet kuvaavat datan liikkumista Content Scriptiltä Storagen kautta käyttöliittymälle (valmiit skriptirivit), sekä myös toisin päin (session tila). Salamat kuvaavat Background Scriptin ja Mutation Observerin lähettämiä impulsseja käsiteltävän ohjelman DOM:issa tai selaimessa tapahtuneista muutoksista, joiden perusteella Content Script päivittää itseään.



Kuvio 23: Ohjelman toiminta suorituksen ollessa käynnissä

Content Scriptin voisi nähdä auton moottorina, joka herää virta-avaimesta (käyttöliittymä), ja pysyy sen jälkeen käynnissä niin kauan kun sille syötetään polttoainetta (Background Script) ja virtaa (Mutation Observer), tai se sammutetaan virta-avaimesta (käyttöliittymä). Yksinkertaistettuna:

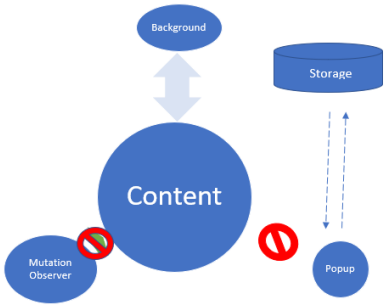
1.) Lähtötilanteessa Popup antaa Content Scriptille impulssin käynnistyä, ja tallentaa Storageen tilansa.

2.) Content Scriptin saadessa viestin se luo kuuntelijat sivun elementeille, ja alkaa itse kuuntelemaan Mutation Observerin, Background Scriptin ja Storagen tilaa.

3.) Kun kuuntelijat ovat luotu, niin Content on luupissa, jossa se ajaa ja päivittää itsensä joka kerta, kun saa Mutation Observerilta tai Backgroundilta impulssin DOM-sisällössä tapahtuneista muutoksista tai selaintapahtumista.

4.) Käyttäjän tehdessä jotain kuunnellulle elementille, Content Script suorittaa kyseiseen tapahtuman käsittelyyn liittyvän funktion ja tallentaa sen seurauksena generoimansa skriptidatan JSON -objektina Storagessa olevaan taulukkoon, josta käyttöliittymä hakee sen käyttäjälle esitettäväksi. Jokaisesta tapahtumasta lähtee viesti myös Background Scriptille, joka antaa käyttäjälle notifikaation onnistuneesti generoidusta tapahtumasta.

5.) Kun sovellus halutaan lopettaa, niin käyttöliittymä päivittää muuttuneen tilansa Storageen sekä lähettää tapahtumasta viestin Content Scriptille. Content Script poistaa luomansa kuuntelijat ja sammuttaa Mutation Observerin. Background Scriptin ja Content Scriptin välinen yhteys on olemassa ohjelman suorituksen ulkopuolellakin, mutta Background Script ei pysty käynnistämään sovellusta ilman käyttöliittymältä tulevaa käskyä.



Kuvio 24: Lepotilassa oleva sovellus

7.2.1 Ohjelman toiminta running -tilassa

Käyttöliittymän ’Start’-napin painallus lähettää Content Scriptille viestin. JSON-objektina lähetettävän viestin mukana käyttöliittymä välittää Content Scriptille sen nauhoituksen aloitustapahtumaan tarvitsemat parametrit, joita ovat mm. sivun url-osoite ja otsikko, sekä käyttötapauksen otsikko (kuvio 25).

```
$('#start_btn').click (function (ev) {
    /**
     * Create event listener for start button. Set session status to wait.
     * get header text from input field and page title + url from tabs query.
     * Then pass message to content script including that data.
     * Disable input field
     */
    chrome.storage.local.set({sessionStatus:"wait"});
    var tcHeader = $('#title_input').val();
    if (tcHeader === "") {
        $('#alert').html('Testcase name is missing.');
    } else {
        var text = 'start';
        $('#title_input').prop('disabled', true);
        $('#pause_btn').css('display','inline-block');
        $('#start_btn').css('display','none');
        $('#alert').html('Recording session is on');
        chrome.tabs.query({active: true, currentWindow: true}, function (tabs) {
            var url = tabs[0].url;
            var title = tabs[0].title;
            chrome.tabs.sendMessage(tabs[0].id, {
                data: text,
                header: tcHeader,
                url: 'www.laurea.fi',
                title: title
            }, function (response) {
                console.log('success');
            })
        });
    }
});
```

Kuvio 25: Funktio, joka ajetaan käyttäjän aloittaessa suorituksen

Viestin saatuaan Content Script tarkistaa sen sisällön. Mikäli sisältö on ’start’, niin kutsutaan initSession -funktiota ja välitetään sille sen tarvitsemat parametrit (kuvio 26).

```
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
    /**
     * Picks messages from background and popup.
     * start, stop and pause are messages from UI
     * tab -related messages are from background script
     * reguest = JSON object
     * sender = Active tab
     * sendResponse = callback
     */
    var data = request.data || {};
    chrome.storage.local.get('sessionStatus', function (result) {
        if (data === 'start' && result.sessionStatus === "wait"){
            initSession(request.header, request.url, request.title, data);
        }else if(data === 'stop'){
```

Kuvio 26: Content Script käynnistää initSession -funktion

Kuvion 27 funktio alustaa käyttötapauksen luomalla saamiensa parametrien perusteella JSON -objektit, jotka se tallentaa Storageen. Tämän jälkeen se kutsuu setSessionStatus-funktiota, joka asettaa session ”running” -tilaan. Lopuksi kutsutaan createListeners-funktiota, joka kategorisoi oman - Pace automaatiotyökaluun – pohjaavan logiikkansa perusteella DOM:in elementit erilaisiin ryhmiin, ja luo näille oikeanlaiset kuuntelijat.

```
function initSession(header, url, title, msg) {
    /**
     * Initializing new recording session. Adding test case header and navigation information to storage
     * and changing session status to running mode so that event listeners can be activate.
     * Needs case header, page's url, title and 'start' -msg(for notifying user) as parameters
    */
    setSessionStatus('running');
    /**
     * Joukko alustustoimenpiteitä..
     */
        createListeners(msg);

}
```

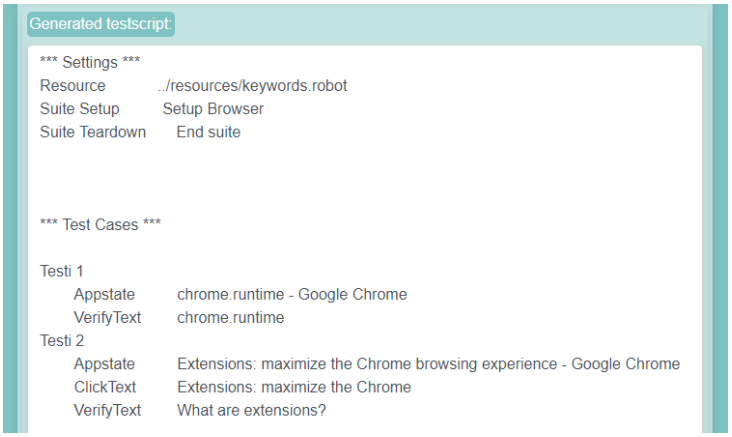
Kuvio 27: initSession-funktion keskeisimmät tehtävät

Edellä kuvattujen toimenpiteiden jälkeen ohjelman on ”running”-tilassa, jonka jälkeen se suorittaa aiemmin kuviossa 23 kuvattua ohjelmaluuppia, kunnes session tilaa muutetaan käyttöliittymästä. Ohjelman toiminnallisuudet, kuten mitä kuuntelijoita createListeners-sisältää, tai mitä mitäkin kuuntelijalla tehdään, rakentuvat tämän perustuksen päälle.

7.2.2 Sovelluksen ”wait” – ja ”stop” -tilat.

Edellisessä kappaleessa esitetyn Start-nappulan käynnistämän ”running” -tilan lisäksi käyttöliittymästä kontrolloitavia tiloja ovat ”wait” ja ”stop”. Pause-nappula asettaa ohjelman tilaan ”wait” ja poistaa kuuntelijat, mutta ei tyhjennä suorituksen aikaista muistia eikä lopeta mitään muuta toimintaa.

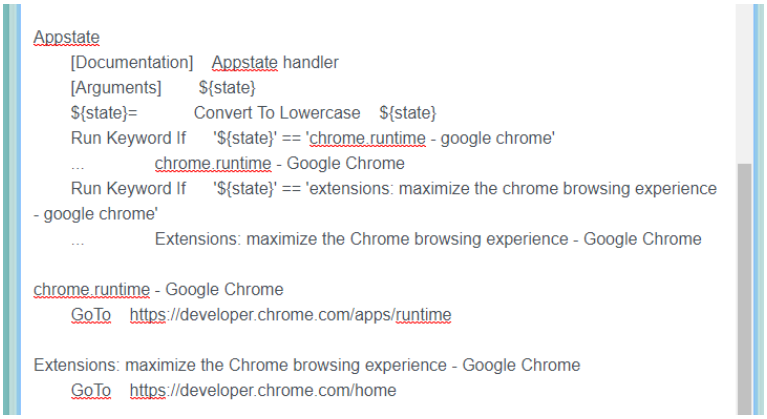
Tilaa tarvitaan tilanteissa, joissa käyttäjä haluaa generoida samaan tiedostoon useampia skriptejä. Kuuntelijat eivät voi aktivoitua wait-tilassa, joten käyttäjä voi rauhassa navigoida seuraavan testin aloitustilaan ilman, että nämä navigointitoimet tallentuisivat mihinkään. Kun alustustoimet ovat valmiina, käyttäjä jatkaa sovelluksen suoritusta palauttamalla ohjelman ”running” -tilaan käyttöliittymän ’Start -napilla. Kuviossa 28 nähdään Testi 1 ja Testi 2 nimiset käyttötapaukset, joissa on eri Appstate.



Kuvio 28: Eri tilasta alkavat käyttötapaukset

Kuviosta 29 on todennettavissa käyttötapausten alkavan eri url-osoitteista, vaikka kuvion 28 Testi 1 -käyttötapaus ei sisällä mitään navigointitapahtumia. Appstate -Pacewordin avulla hallitaan testien alustamiseen liittyvät navigointitapahtumat. Käyttäjä on navigoinut toiseen osoitteeseen nauhoittamaan testiä 2 sovelluksen ollessa wait-tilassa.

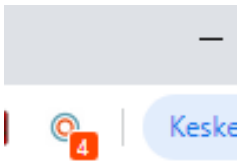
Appstate chrome.runtime kutsuu keywords.robotista keywordia chrome runtime – Google Chrome, joka ohjaa selaimen eri url- osoitteeseen, kuin Testi 2:n Appstate Extensions – maximize *. Appstatea hallitaan erillisellä handlerilla, jonka syntaksi perustuu Robot Frameworkiin.



Kuvio 29: Handler ja Appstaten käyttämät keywordit

Kolmas käyttöliittymästä hallittava tila on Stop, joka nimensä mukaisesti pysäyttää ohjelman suorituksen. Se poistaa kuuntelijat, sammuttaa Mutation Observerin, sekä tyhjentää kaikki Storagessa olevat lopetettuun sessioon liittyvät tiedot.

Testiskriptien kirjoittamista html -käyttöliittymään ei tarvitse lopetustoimien ohessa tehdä erikseen. Toiminnallisuus on toteutettu siten, että joka kerta kun käyttöliittymä avataan, niin sen skripteille varattuihin <textarea>-tyyppisiin elementteihin kirjoitetaan sen hetkinen nauhoituksen tilanne. Tapa mahdollistaa sen, että käyttäjä voi yhdellä klikkauksella tarkastella generoitua skriptiä missä tahansa suoritusvaiheessa. Käyttöliittymän avaava ikoni esitellään kuviossa 30.



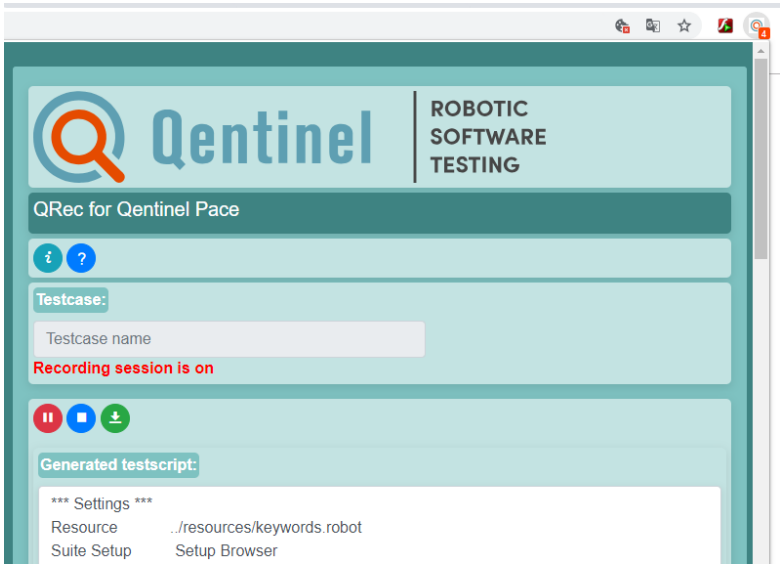
Kuvio 30: Ikoni. Numero kertoo tapauksen aikana generoidut skriptirivit

Kuvion 31 funktio hakee Storagen sisällön käyttöliittymän avautuessa. Tämän jälkeen se kutsuu parseToScreen funktiota, joka kirjoittaa parametreina saamansa sisällön ruudulle.

```
$(document).ready(function(){
    /**
     * When UI is opened, get values from storage.
     * Populate text area if testSteps variable contains data
     */
    $('#input_forms').append('<span id="alert" style="color:red;"></span>');
    chrome.storage.local.get(['tcInfo', 'testSteps', 'variables','sessionStatus'], function (result) {
        if (result.testSteps) {
            if(result.testSteps.length > 0) {
                parseToScreen(result.testSteps, result.tcInfo, result.variables);
```

Kuvio 31: Käyttöliittymän document ready -funktio

Edellä kuvattujen toimintojen ansiosta käyttäjä voi koska tahansa katsoa mitä sovellus on generoinut. Liittymässä lukee myös suorituksen sen hetkinen tila, joka esimerkkikuviossa ’Recording session is on’ (kuvio 32).



Kuvio 32: Suorituksen aikana avattu käyttöliittymä

7.3 Kuuntelijoiden lisääminen ja käyttötapahtumien poimiminen

Sovelluksen varsinaiset ominaisuudet rakentuvat edellisissä kappaleissa esitetyn ohjelmamoottorin päälle. Työ kokonaisuudessaan sisältää yli tuhat riviä JavaScript-koodia, ja on osa kaupallista automaatiotuotekokonaisuutta, joten tämä kohta aiheesta käsitellään yksinkertaisen teknisen esimerkin kautta.

Käyttäjän suorittamien tapahtumien poimiminen on sinänsä triviaali toimenpide. Käsiteltävään ohjelmaan injektoidun Content Scriptin avulla vodaan luoda mille tahansa elementille haluamamme kuuntelijat. Tämän jälkeen määrittelemme toimenpiteet, jotka käyttäjän laukaisema kuuntelija toteuttaa (kuvio 33).

```
$.each(dropdowns,function(e,elem){
    /**
     * In case of <select> elements we are listening onchange event then
     * get locator and text/value from selected option
     */
    if(!elem.attributes.dd) {
        elem.setAttribute("dd", true);
        var locator = "";
        var inputText = "";
        $(elem).on('change', function (event) {
            if (event.target === event.currentTarget){
                if (locator = getDropdownElem(event.target)) {
                    try { //in case there is no text..
                        inputText = this.selectedOptions[0].textContent.trim();
                    } catch (err) {
                        inputText = this.selectedOptions[0]
```

Kuvio 33: Funktio kuuntelee onchange -tapahtumaa <select> HTML-elementeistä

Kaikki käyttäjän toimenpiteiden tallentamiseen ja automaatioskriptin luomiseen liittyvät ominaisuudet toimivat saman logiikan mukaan. Kun sovelluksen edellisten otsikkojen alla kuvailtu perusrunko oli kasassa, niin uusien ominaisuuksien lisääminen tapahtui aina samalla kaavalla:

1.) Valittiin elementti-, ja tapahtumatyyppi kuunneltavaksi.

2.) Lisättiin elementille kuuntelijat.

3.) Määritellään mitä haluamme tehdä (ja miten se tehdään) kun kuuntelemamme tapatuma laukeaa.

4.) Muunnetaan tapahtumasta keräämämme data testiskriptiksi.

5.) Tallennetaan tapahtuma Storageen testiskriptin sisältävänä JSON -objektina.

7.4 Haasteet – Ohjelmamoottorin suorituskyky

Content Scriptin keskeisin funktio luo tapahtumakuuntelijat. Funktiota kutsutaan aina kun selaimen tila, tai joku käsiteltävän sivun osa muuttuu. Funktio saatetaan yhden session aikana suorittaa kymmeniä, tai jopa satoja tai tuhansia kertoja, jos web-sovelluksessa on paljon dynaamisesti muuttuvaa sisältöä. Funktiota kutsutaan esimerkiksi tilanteessa, jossa sivulle on tullut yksi uusi elementti lisää, vaikka sen tuhat aiempaa elementtiä ovat jo kuuntelun alaisena.

Kun jokaisella iteraatiolla käytiin kaikki elementit läpi, niin sekä kohdesovellukseen että työkaluun liittyviltä suorituskykyongelmilta ei voinut varsinkaan raskaampia sovelluksia käsiteltäessä välttyä. Ongelma saatiin hallintaan manipuloimalla käsiteltävää dom-elementtiä. QRec lisää kullekin kuuntelemalleen elementille oman yksilöivän attribuuttinsa, jonka avulla sovellus tietää elementin olevan jo kuunneltuna eikä sille tarvitse enää tehdä mitään (kuvio 34).

```
var dropdowns = document.querySelectorAll('select:not([dd="true"])')
```

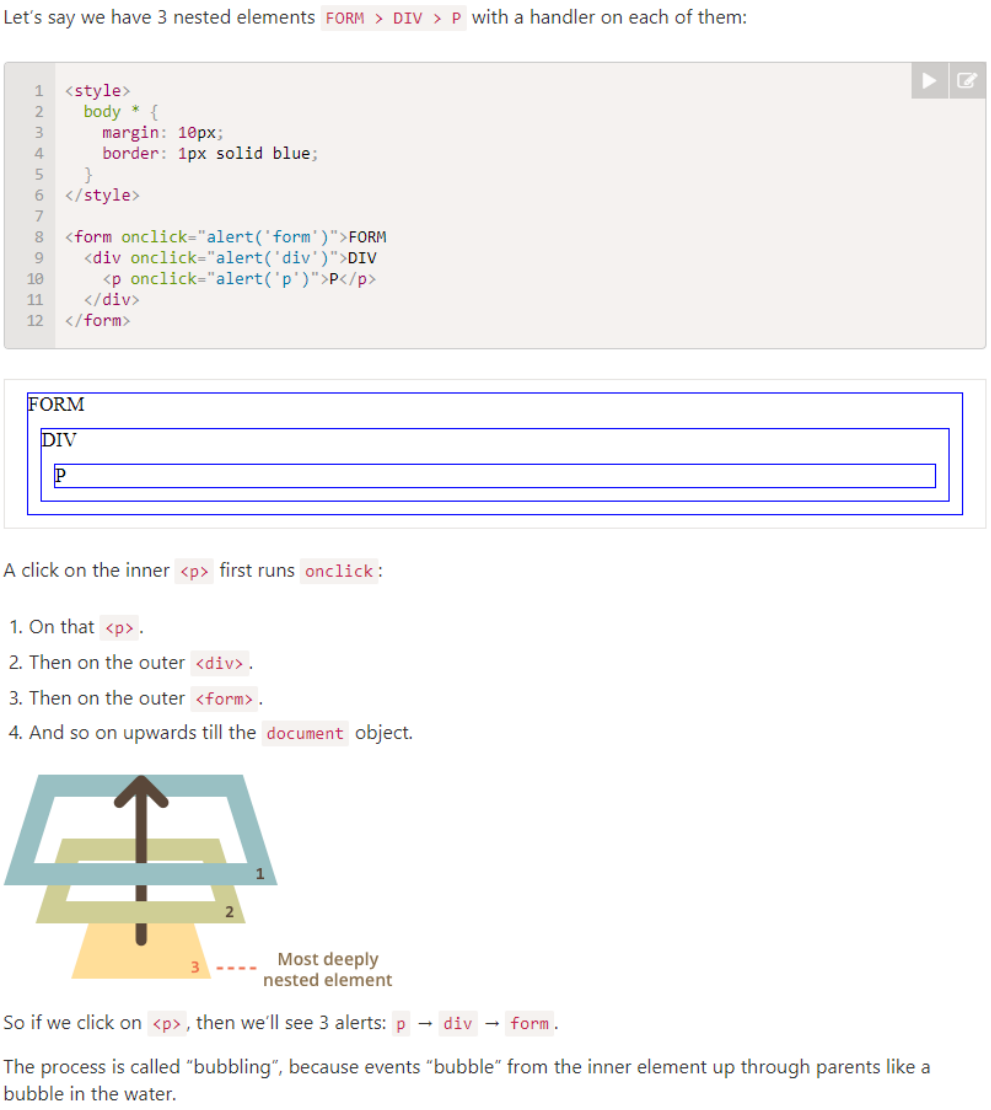
Kuvio 34: Lauseke valitsee dokumentista kaikki <select> -elementit, joilla ei ole dd -attribuuttia

Mutation Observer itsessään havaittiin potentiaaliseksi suorituskykyongelmia aiheuttavaksi komponentiksi. Muutosten seuranta haluttiin kattavaksi, mutta raskailla dynaamisilla sivustoilla Observerin läpi kulkee valtavasti mutaatioita. Jos mutaatiot etsivässä luupissa on löydettyihin uusiin elementteihin liittyen funktiokutsuja yms. sivua kuormittavaa logiikkaa mukana, niin joillain sivustoilla tämä aiheutti pahimmillaan koko sovelluksen jumiutumisen. Ongelma ratkaistiin siten, että kaikki mutaatiot käytiin edelleen läpi, mutta sieltä etsittiin ainoastaan osumia ilman elementtien yksilöintiä tai mitään muutakaan toiminnallisuutta. Observerista tehtiin laskuri, joka palauttaessaan jotain muuta kuin nollan aktivoi tapahtumankuun-

telijat lisäävän funktion. Kun tapahtumankuuntelijat lisäävässä funktiossa on aiemmin kuvi-ossa 34 esitetyn kaltaisia ehtoja, joiden avulla toimintoja suoritetaan ainoastaan tarpeeseen, niin Observerissa ei enää tarvinnut ottaa kantaa siihen onko sen löytämä muutos validi vai ei.

7.5 Haasteet – Event bubbling

Suurin haaste toiminnallisuuksien lisäämisen ja oikeiden elementtitunnisteiden poimimisen suhteen oli Event bubbling. Event bubbling on JavaScriptin ominaisuus, jolla tarkoitetaan sitä, että käyttäjän kohdistaessa toiminnon jonkin toisen elementin sisällä olevaan lapsielement-tiin, myös sen ympäröimien parent-elementtien kuuntelijat laukeavat. (Kuvio 35)



Kuvio 35: Tagin <p> sisällä olevan lapsielementin klikkaaminen laukaisee myös sen vanhem-mat (Kantor 2018)

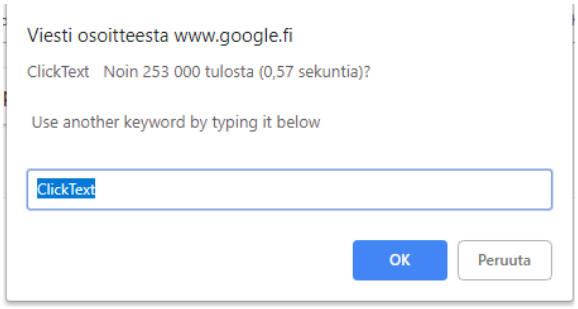
Suurimmat haasteet kohdistuivat oikean eventin ja elementtitunnisteen poimimiseen rakenteista, joissa oli paljon sisäkkäisiä elementtejä. Moderneilla JavaScript -tekniikoilla koodatuissa websovelluksissa saattaa elementtejä, joiden sisällä on kymmenen muuta elementtiä. Näistä vaikkapa viidessä voi olla jokin QRec-sovelluksen lisäämä ’click’-tapahtuman kuuntelija. Kun pinon sisin laukaisee tapahtuman, niin nuo kaikki laukeavat. Tämä aiheutti paljon ongelmia oikean elementtitunnisteen löytämisen suhteen.

Normaalisti websovelluksen sisään ohjelmoidun JavaScript -koodin kanssa bubbling -tilanteita voidaan välttää esimerkiksi event.stopPropagation, ja event.stopImmediatePropagation funktioiden avulla. Tämän avulla identifiointiin ongelmat olisi saanut QRecin osaltakin ratkaistua. Funktioiden käyttö esti kuitenkin monissa tapauksissa myös käsiteltävän ohjelman omien tapahtumien laukeamisen, joten käyttö oli erikoistapauksia – jolloin sivulla ei odotetakaan tapahtuvan mitään – lukuun ottamatta poissuljettu vaihtoehto.

Pace-automaatioratkaisussa elementtien tunnistaminen perustuu aina ensisijaisesti tekstin- ja erilaisten, usein tooltip-teksteinä, käyttäjälle esitettävien ikoniattribuuttien tunnistamiseen. Oikeiden identifioijien löytämiseen jouduttiin ohjelmoimaan varsin monimutkaisia DOM:in läpikäymiseen tarkoitettuja funktioita, jotka etsivät tietyssä järjestyksessä erilaisia asioita sekä kohde-elementistä, että sen sukulaisista, kunnes päättelee löytäneensä etsimänsä. Osoitintyyppi (event.target tai event.currentTarget/this) ja etsintäjärjestys riippuu aina kunkin elementin tyypistä (teksti, kuva, nappi, input, ikoni jne.), sekä siitä pitääkö elementti sisällään muita elementtejä ja jos pitää, niin minkä tyyppisiä nämä elementit ovat.

Ratkaisumalleilla päästiin lopulta varsin hyvään lopputulokseen vaikuttamatta kohdesovelluksen toimintaan. QRec pystyy valitsemaan virheettömästi Pacewordin kaikkiin yleisimpiin elementtityyppeihin, sekä poimimaan tarkasti elementtien oikeat visuaaliset tunnisteet sovellukselle ohjelmoidun logiikan mukaan. Käytettävä logiikka on sama mitä varsinainen automaatiotuotekin käyttää. Elementeistä, joiden tunnusta ei pystytä yksilöimään, tai joilla ei ole yksilöivää visuaalista tunnistetta, luodaan Robot Framework muuttuja, jonka käyttäjä saa nimetä itse.

Tunnistamattomien elementtityyppien varalta jokaiseen sivulla olevaan elementtiin on lisätty tuplaklikkausta kuunteleva funktio, joka palauttaa käyttäjälle elementin visuaalisen tunnisteen samalla logiikalla, kun muidenkin elementtien tapauksessa. Näissä tapauksissa käyttäjä voi itse määritellä haluamansa tapahtumaa kuvaavan Pacewordin sovelluksen avaamassa dialogissa (kuvio 36).



Kuvio 36: Käyttäjä on tuplaklikannut tunnistamatonta elementtiä

8 Yhteenveto ja saavutetut ominaisuudet

Kehitystyön tuloksena syntyi toimiva työkalu, joka täytti sille asetetut vaatimukset. Ominaisuuksia kertyi lopulta aiottua enemmän. Ohjelman ensimmäinen versio laitettiin jakoon 17.12. ja sillä pystytään tallentamaan ja kääntämään skriptiksi kaikki webelementteihin kohdistuvat käyttötapahtumat.

Käyttötapahtumien tallentamisen lisäksi sovellus pystyy generoimaan tarpeellisen Robot Framework logiikan. Logiikan avulla robotti osaa ajaa kaikki käyttäjän nauhoittamat käyttötapaukset aloittaen ne oikeasta aloituspisteestä, ilman että testiskriptin sisältävässä .robot tiedostossa näkyy yhtään navigointitapahtumaa. Perustoiminnallisuuksiin kuuluu myös käyttäjän omavalintaisesti nimeämän, xpathia osoittimena käyttävän muuttujan luominen elementeille, joille sovellus ei löydä robotille sopivaa tunnistetta.

QRec sisältää mekanismin, jonka avulla se osaa tunnistaa elementtitunnisteiden mahdolliset duplikaatit. Näissä tapauksissa käyttäjää pyydetään osoittamaan robotille oikea elementti valitsemallaan ankkurointitekstillä tai indeksillä.

Vaatimusmäärittelyyn kirjattu helppokäyttöisyys saatiin toteutettua. Latauslinkin saanut käyttäjä voi ladata ja asentaa sovelluksen suoraan Chromen webstoresta. Tekstien- ja arvojen verifioimista lukuun ottamatta käyttötapausta tallentavan käyttäjän ei tarvitse tehdä mitään normaalista poikkeavia eleitä käyttötapausta suorittaessaan. Poikkeuksena tapaukset, joissa ohjelma pyytää käyttäjältä nimen luomalleen muuttujalle, tai vapaavalintaisen ankkuroivan tunnisteen elementille, jolla on duplikaatti.

Sovellus on siinä mielessä älykäs, että se ei tallenna jokaista klikkausta tai fokuksen muutosta, vaan ainoastaan tilanteet, joissa oikeasti tapahtuu jotakin. Näin ollen sen generoima automaatioskripti on syntaksiltaan siistiä, eikä sisällä ylimääräisiä toimintoja. Käyttäjälle annetaan palautetta suorituksen aikana pienellä notifikaatio-ikonilla (Nähtävissä kuviossa 30). Ominaisuus koettiin käyttäjän kannalta mukavaksi, koska sen avulla voidaan nähdä heti taltioiko QRec tehdyn toimenpiteen.

Valmiit skriptit saa tallennettua levylle, josta ne ovat välittömästi Pace-konsolin ja QWebin suoritettavissa. Sovellus luo automaattisesti oikeanlaisen kansiohierarkian. JavaScriptin rajoitteista johtuen tallennus on ohjelman ensimmäisessä versiossa mahdollista ainoastaan käyttäjän selaimeensa määrittelemään lataukset -kansioon, josta ne voidaan välittömästi tämän jälkeen suorittaa.

Kehitetyn sovelluksen ominaisuuspaletti on runsas. Sen avulla voidaan nauhoittaa standardeja ja hyvää ohjelmointitapaa seuraavilta web-sovelluksilta lähes mikä tahansa käyttötapaus ja generoida se suoritusvalmiiksi skriptiksi. Hankalimmissakin tapauksissa sitä käyttävä asiantuntija pystyy rakentamaan sovelluksen avulla valmiit rungot samalla kun tutustuu automatisoitavan sovelluksen toimintoihin. Ajansäästö erityisesti projektien alkuvaiheessa on merkittävä vs. tilanteeseen, jossa jokainen tiedostoissa lukeva rivi on siihen itse kirjoitettu.

Ihmisen asiantuntijuutta tämänkaltaisella tuotteella ei voi korvata. Työn tehokkuuden lisäämisen lisäksi toisena teknisesti merkittävänä tekijänä työkalulle voikin nähdä asiantuntijuuden lisäämisen. Aiheeseen vielä vihkiytymättömän automatisoijan on sovelluksen avulla hyvin helppoa omaksua Paceword -skriptaustekniikka itse luomiensa käyttötapausten kautta, sekä lisätä ymmärrystä taustalla olevan peruslogiikan suhteen liittyen esimerkiksi omien Appstate keywordien luomiseen, ja kuinka niiden toimintaa voidaan hallita kirjastojen alla toimivan Robot Framework -viitekehyksen tarjoamien ominaisuuksien avulla.

Tutkimusvaiheen tiedonhankinta suoritettiin pääosin keräämällä kirjallista aineistoa erilaisista teknisistä viitekehyksistä, sekä jo olemassa olevista ratkaisuista ja teorioista, joiden avulla ongelma voisi olla ratkaistavissa. Hiljaista tietoa kerättiin sekä keskustelemalla, että yrityksen viestintäkanavista, joissa oli paljon materiaalia erilaisista aiheeseen liittyvistä haasteista ja henkilöiden niihin kehittämistä ratkaisumalleista.

Kerättävän tiedon validiteettia ja reliabiliteettia ei pyritty tutkimuksessa raamittamaan tiukasti. Suunniteltuja menetelmiä ei yksilöity etukäteen, eikä käytettäviä kysymyksiä rajattu alkuvaiheessa koskemaan jotain tiettyä menetelmää tai ongelmaa. Validiteetti ja reliabiliteetti määriteltiin tunnistamalla kerätystä aineistosta teoriassa tähän viitekehykseen sopivat, geneeriseksi muunnettavissa olevat ratkaisumallit sekä testaamalla näiden mallien toimivuutta käytännössä.

Menettelyllä haluttiin varmistaa, että emme rajaa etukäteen pois vaihtoehtoja, jotka voivat osoittautua toteuttamiskelposeksi joko tässä sovelluskehitysprojektissa, tai jossain tulevaisuuden hankkeessa. Kerättyä tietoa ja ideoita analysoitiin projektin aloitusvaiheessa useissa palavereissa. Näiden perusteella voitiin muodostaa kokonaiskuva keskeisimmistä tutkimuskysymyksistä, ja menetelmistä, joiden katsottiin voivan vastata kysymyksiimme valituilla tutkimusmenetelmillä.

Kehitystyössä saavutettiin odotusten mukainen tulos. Ennen kehitystyötä tehty tutkimus osoittautui kredibiliteetiltään täsmälliseksi, eikä kehitystyön aikana noussut esiin asioita, jotka olisivat kyseenalaistaneet tehdyn tutkimuksen. Kehitystyöhön valitut ratkaisut osoittautuivat projektin edetessä jopa paremmin työhön soveltuviksi, kuin pelkän tutkimuksen perusteella voitiin päätellä. Tutkimus osoitti myös sen, että muilla tutkituilla menetelmillä sovelluksen toteutus ei olisi ollut mahdollinen annetussa aikaikkunassa ja siihen varatuilla resursseilla. Tutkimuksen validiteettia arvioidessa on syytä huomioida valitun menetelmän suppea viitekehys (selainriippuvuus, vaatimusmäärittely, aikataulu). Validiteetti oli pätevä omassa viitekehyksessään. Kehitystyö ja sen lopputuloksen voidaan katsoa osoittavan, että tutkimuksesta tehdyt johtopäätökset olivat kokonaisuudessaan oikeita ja tutkimuksen kredibiliteetti ja validiteetti vahvistettu pätevaksi. Työn tilaajalta ja konsulttikäyttäjiltä saatu palaute on ollut hyvää.

9 Mahdollinen jatkokehitys

Jatkokehitystä ajatellen seuraava mahdollinen askel on serveripään rakentaminen, joka avaa paljon uusia mahdollisuuksia sovelluksen kehittämisen suhteen. Näitä olisivat esimerkiksi generoitujen skriptien ajaminen suoraan sovelluksesta, jolloin käyttäjä voisi hallita paitsi testien luomisen, niin myös niiden ajamisen samasta käyttöliittymästä. Serveripään mukaan ottaminen mahdollistaisi myös erilaiset editoritoiminnallisuudet, joiden toteuttaminen pelkän Chrome -laajennuksen avulla ei ole järkevää, eikä kaikilta osin mahdollistakaan.

Pääsy serverille ja käyttäjän tiedostojärjestelmään mahdollistaisi myös tiedostojen uudelleenkäytön. Uusia skriptejä voitaisiin generoida jo olemassa oleviin tiedostoihin, sekä liikutella suoraan versionhallintajärjestelmään ja kehitysputkeen ilman, että skriptiä nauhoittavan ja/tai kirjoittavan käyttäjän tarvitsee tietää esim. git, Jenkins, AWS/Azure putkesta tai sen käytöstä yhtään mitään. Tässä vaiheessa sovellus ei enää olisikaan erillinen skriptaamista helpottava ja nopeuttava työkalu, vaan kiinteä osa kokonaisratkaisua, jossa automaatioskriptin kehitystyötä viedään lähemmäksi pistettä, jossa sen tehokas kirjoittaminen voisi olla yhtä triviaalia kuin vaikkapa jonkun Microsoft OneNoten käyttö, eikä sitä kirjoittavan asiantuntijan tarvitse hallita kuin yksi helppokäyttöinen työpöytäsovellus sen tarjoamine toimintoineen.

Työn esivalmistelujen ja tiedonhankinnan aikana esiin nousi myös mahdollisuus kuvantunnistukseen ja käyttäjän näppäimistön ja hiiren tallentamiseen perustuvasta ratkaisusta. Tätä ei sinällään nähty järkeväksi web-maailmaan, jossa asiat saadaan tehtyä luotettavammin ja nopeammin jo olemassa olevilla tekniikoita hyväksikäyttäen. Tulevaisuutta ajatellen konsepti on kuitenkin erittäin mielenkiintoinen.

Erilaisia jo olemassa olevia menetelmiä hyödyntämällä ja yhdistelemällä voisi hyvinkin olla mahdollista kehittää täysin skaalautuva ja ympäristöriippumaton hybridisovellus, jossa

voimme nauhoittaa käyttötapauksia ja generoida automaatioskriptiä mistä tahansa sovellusympäristöstä tai niiden yhdistelmästä. Tämä on paitsi testiautomaatiota, niin erityisesti voimakkaassa kasvussa olevaa RPA:ta ajatellen mahdollisuus, joka on tulevaisuutta ajatellen ehdottomasti yksi jatkotutkimuksen arvoinen aihe.

10 Oman oppimisen arviointi

Valitsin työn opinnäytetyöni aiheeksi, koska se tuntui kokonaisuudessaan erittäin mielenkiintoiselta projektilta. Projektin aikana joutuisin todella haastamaan itseäni ja omaksumaan paljon sellaisia asioita ja tekniikoita, joihin en opiskelujen aikana tai työelämässä ollut vielä törmännyt.

Projekti oli opinnäytetyön aiheeksi todella iso. Aliarvioin kokemattomuuttani rajusti työmäärän, joka olisi vaadittu alkuperäisen, tähän opinnäytetyöhön suunnittelemani viitekehyksen dokumentoimiseen, joten jouduin miettimään rajaukseni uudelleen. Tämä uudelleensuunnittelu aiheuttikin aikataulullisen n. 1-2 kuukauden viivästyksen alustavaan suunnitelmaani, jossa olisin valmistunut opinnoistani jouluksi 2018.

Chrome-laajennus ei aiheena ollut ollenkaan tuttu ennen mahdollisten teknisten menetelmien kartoitusta, joten se oli jo sinällään haaste projektiin lähdettäessä. Tietoa ja kokemuksia aihepiiristä löytyi yleisesti ottaen kuitenkin melko helposti. Lisäksi Googlella on erittäin kattava tekninen dokumentaatio, joten sikälikään asia ei muodostunut missään vaiheessa ongelmaksi. Työssä käytetyt web-tekniikat, kuten HTML, CSS, JavaScript, jQuery - ja niihin liittyen erityisesti DOM-manipulaatio - olivat opiskelujen ja työelämän kautta kohtalaisen hyvin hallussa entuudestaan, joten ohjelmointityö itsessään ei varsinkaan projektin alussa ollut hidasteena, vaikka selainlaajennus pakottikin tekemään asiat hieman eri tavalla kuin olin aiemmin tottunut.

Opin työn aikana valtavasti paitsi laajennuksen toiminnasta, niin myös JavaScript -ohjelmoinnista. Erityisesti sen joistakin erityispiirteistä, jotka eivät sellaisenaan olleet aiemmissa projekteissani tulleet vastaan, mutta joiden huomioonottaminen ja omaksuminen oli tässä työssä kokonaisuuden kannalta kaiken perustana. Kun normaalisti vaikkapa web-sovellusta ohjelmoidessa voi itse kertoa mitä ohjelman haluaa tekevän ja kuinka se sen tekee, niin tämän työn puitteissa piti pikemminkin etsiä keinot, jolla saada sovellus pysymään pystyssä muuttuvissa olosuhteissa ja ympäristöissä, jotta se olisi reagointivalmiina kaikkiin mahdollisiin tilanteisiin vaikuttamatta samalla mitenkään käsiteltävän web-sovelluksen toimintaan.

Ongelmanratkaisun kannalta keskeisenä asiana oli selainlaajennuksen asettamat rajoitteet ja säännöt, jotka tekivät joidenkin asioiden toteutuksesta ohjelmointimielessä huomattavasti oletettua haastavampaa. Haasteena olivat myös sovelluskehitystyössä tarvitut lukuisat erilai-

set apit, kirjastot, ja tekniikat, joista en aiemmin ollut kuullut yhtään mitään. Henkilökohtaiselta kantilta työn voisikin kiteyttää niin, että lopputuotteessa on huomattavan paljon vähemmän sellaista mitä osasin ennalta, kun sellaista, jonka jouduin opettelemaan projektin aikana saadakseni kokonaisuuden toimimaan, vaikka projektin aloitusvaiheessa kuvittelin asian olevan päinvastoin.

Työllä saatiin vastattua tarpeeseen. Vaikka tuotteistamista ajatellen jatkokehitystä kannattaa tehdä vielä paljon, niin työ tavoitti sille asetetut tavoitteet sekä toimeksiantajan, että tämän opiskeluihini liittyvän viitekehyksen osalta.

Lähteet

Painetut

Crispin, L & Gregory, J. 2009. Agile Testing. Boston: Pearson Education, Inc.

Fewster, M & Graham, D. 1999. Software Test Automation. ACM Press.

MacKenzie, S. 2012. Human-Computer Interaction: An Empirical Research Perspective. Elsevier Science & Technology.

Pohjalainen, M. 2012. Hiljaisen tiedon käsite ja hiljaisen tiedon tutkimus. Tampere: Informaatiotutkimuksen yhdistys.

Sampaio, A & Sampaio, I-B. 2015. Theory and Basic Research in Software Engineering. Polytechnic Institute of Porto/Superior Institute of Engineering.

Wynn, M & Dreyer, H. 2016. International Journal on Advances in Software 2016 vol.9 nr.3&4. IARIA.

Zuber-Skerritt, O. 2002. Action Learning, Action Research and Process Management. Emerald Publishing Limited.

Sähköiset

Chrome Developers. 2018. Chrome APIs. Viitattu 22.11.2018. https://developer.chrome.com/extensions/api_index

Chrome Developers. 2018. chrome.runtime. Viitattu 28.11.2018. https://developer.chrome.com/extensions/runtime

Chrome Developers. 2018. chrome.storage. Viitattu 28.11.2018. https://developer.chrome.com/extensions/storage

Chrome Developers. 2018. chrome.tabs. Viitattu 28.11.2018. https://developer.chrome.com/extensions/tabs

Chrome Developers. 2018. Content Scripts. Viitattu 1.12.2018. https://developer.chrome.com/extensions/content_scripts

Chrome Developers. 2018. Content Security Policy (CSP). Viitattu 26.11.2018. https://developer.chrome.com/extensions/contentSecurityPolicy

Chrome Developers. 2018. Develop Extensions. Viitattu 26.11.2018. https://developer.chrome.com/extensions/devguide

Chrome Developers. 2018. Overview. Viitattu 28.11.2018. https://developer.chrome.com/extensions/overview

Kantor, I. 2018. Bubbling and capturing. Viitattu 27.11.2018. https://JavaScript.info/bubbling-and-capturing

Kantor, I. 2018. Introduction: callbacks. Viitattu 4.12.2018. https://JavaScript.info/callbacks

Yhteiskuntatieteellinen tietoarkisto(KvantiMOTV). 2018. Viitattu 28.2.2019. https://www.fsd.uta.fi/menetelmaopetus/mittaaminen/luotettavuus.html

MDN web docs. 2018. background. Viitattu 29.11.2018. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/background

MDN web docs. 2018. Callback function. Viitattu 4.12.2018. https://developer.mozilla.org/en-US/docs/Glossary/Callback_function

MDN web docs. 2018. Client-side storage. Viitattu 8.12.2018. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Client-side_storage

MDN web docs. 2018. MutationObserver. Viitattu 1.12.2018. https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver

MDN web docs. 2018. runtime.onMessage. Viitattu 28.11.2018. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/runtime/onMessage

MDN web docs. 2018. What are extensions? Viitattu 22.11.2018. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/What_are_WebExtensions

Qentinel. 2018. Tietoa meistä. Viitattu 15.9.2018. https://qentinel.com/fi/tietoa-meista/

Qentinel. 2018. Tietoa meistä. Viitattu 15.9.2018. https://qentinel.com/fi/ota-yhteytta/

Robot Framework. 2019. Introduction. Viitattu 15.1.2019. https://robotframework.org/

Rouse, M. 2018. browser extension. Viitattu 25.11.2018. https://whatis.techtarget.com/definition/browser-extension

SeleniumHQ. 2018. Platforms Supported by Selenium. Viitattu 22.11.2018. https://www.seleniumhq.org/about/platforms.jsp#programming-languages

Suojanen, U. 2019. Toimintatutkimus ammatillisen kehittymisen välineenä. Viitattu 26.2.2019. https://metodix.fi/2014/05/19/suojanen-toimintatutkimus/

W3C. 2018. Mutation observers. Viitattu 1.12.2018. https://www.w3.org/TR/dom/#mutation-observers

W3Schools. 2018. HTML5 Web Storage. Viitattu 2.12.2018. https://www.w3schools.com/HTML/html5_webstorage.asp

Vecteezy. 2019. Modern login form interface design with username and password. Viitattu 4.1.2019. https://www.vecteezy.com/vector-art/185753-modern-login-form-interface-design-with-username-and-password

Julkaisemattomat

Heimola, A. 2018. Henkilöhaastattelu 4.11.2018. Qentinel Oy. Espoo.

Heimola, A. 2018. Sisäiset esitysmateriaalit. Qentinel Oy. Espoo. Luettu 8.11.2018.

Kuviot

Taulukot