Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutusohjelma

Yrityksen tietojärjestelmät

2015

Satu Suominen

SELENIUM WEBDRIVERIN SOVELTUVUUS KUVAKAAPPAUSTEN OTTAMISEEN WEB-SOVELLUKSESTA

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittelyn koulutusohjelma | Yrityksen tietojärjestelmät

2015 | 39

Tuomo Helo

Satu Suominen

SELENIUM WEBDRIVERIN SOVELTUVUUS KUVAKAAPPAUSTEN OTTAMISEEN WEB-SOVELLUKSESTA

Opinnäytetyössä käsitellään projektia, jossa otetaan kuvakaappauksia web-sovelluksen käyttöliittymästä. Tavoitteena oli automatisoida kuvakaappausten ottaminen, jotta säästetään aikaa ja manuaalista työtä. Opinnäytetyön toimeksiantajana toimii Wallac Oy ja sovellus, josta kuvia otetaan, on nimeltään LifeCycle. LifeCycle on kansainväliseen jakeluun menevä raskaudenaikainen riskilaskentaohjelmisto, joka oli syksyllä 2015 käännetty kahdeksalle eri kielelle.

Kuvakaappausten ottaminen toteutettiin käyttämällä apuna Selenium WebDriveria, joka on avoimen lähdekoodin testaustyökalu web-sovelluksille. Projekti toteutettiin Microsoft Visual Studio ympäristössä ohjelmointikielen ollessa C#.

Tutkimus tehtiin CASE-studyna, ja työn tuloksena saatiin automatisoitua kuvien ottaminen testimetodi suorittamalla ilman manuaalista työtä. Projekti onnistui tavoitteiden mukaisesti, kuvat saatiin automatisoidusti ohjeeseen ja jatkossa kuvia voi ottaa ilman manuaalista työtä. Seleniumin käyttö projektissa ajoi tarkoituksensa.

ASIASANAT:

Selenium, web-sovellus, sovellusohjelmointi, C#, kuvakaappaus

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology | Information Systems

2015 | 39

Tuomo Helo

Satu Suominen

TAKING SCREENSHOTS WITH SELENIUM WEBDRIVER OF A WEB APPLICATION

The subject of this bachelor's thesis is a project whose aim was to facilitate taking screenshots from web application's user interface. The target of this project was to automate taking screenshots to save time and manual labor. The client of this thesis was Wallac Oy and the application whose the pictures are taken is called LifeCycle. LifeCycle is a prenatal screening software which had eight different language versions at the time of writing this thesis and it will be distributed globally.

Screenshots were implemented by using Selenium WebDriver. Selenium is an open-source automating and testing tool for web applications. This thesis project is conducted in a Microsoft Visual Studio environment and the programming language was C#.

This thesis work was carried out as a case study and the result was that no manual labor is needed to take the screenshots because screenshots can now be taken by running the test method developed in this thesis. The goal of this thesis project was met. The screenshots were taken automatically so that they could be incorporated in the application user guide and in the future pictures can be taken without manual labor.

KEYWORDS:

Selenium, web application, programming, C#, screenshot

SISÄLTÖ

KUVAT

KOODIESIMERKIT

KÄYTETYT LYHENTEET JA SANASTO

API	Application Programming Interface eli ohjelmointirajapinta on rajapinta, jonka avulla päästään käsiksi ohjelmiston sisäiseen toiminnallisuuteen. (What are APIs? 2015.)
Boolean	Totuusarvo, joka voi saada joko arvon tosi tai epätosi.
CSS	CSS (Cascading Style Sheetis) on tyylikieli, jossa määritellään miten HTML elementit esitetään. (W3schools 2015b.)
HTML	HTML on standardoitu merkintäkieli web-dokumenttien esittämiseen. (W3schools 2015a.)
JavaScript	JavaScript on komentosarjakieli, jonka avulla kehitetään dynaamisia web-sovelluksia.
SDK	Software Development Kit. Kokoelma kehitystyökaluja joilla voi kehittää sovelluksia tietylle laitteelle tai käyttöjärjestelmälle. (TechTerms 2015.)
Syntaksi	Lauseoppi, joka määrittelee ohjelmointikielen lailliset ilmaukset.
URL	Uniform Resource Locator. Verkkosivuston osoite jolla navigoidaan oikeaan osoitteeseen.

1 JOHDANTO

Opinnäytetyössä perehdytään projektiin, jossa automatisoidaan kuvakaappausten ottaminen Selenium WebDriverilla. Opinnäytetyön toimeksiantajana toimii Wallac Oy ja projekti, jonka toteutuksessa olen mukana, on nimeltään LifeCycle. LifeCycle on kansainväliseen jakeluun kehitetty raskaudenaikainen riskilaskentaohjelmisto, joka tulee toimimaan internetselaimessa. Ohjelmiston version 5.0 kehitys oli jo pitkällä liittyessäni projektiin mukaan ja oma osuuteni projektissa liittyy ohjelmiston ohjetiedostoon.

Opinnäytetyön tarkoituksena on perehtyä siihen, miten Selenium WebDriver soveltuu kuvakaappausten ottamiseen. Projektin tarkoituksena on ottaa LifeCyclen käyttöliittymästä kuvakaappauksia käyttämällä Selenium WebDriveria. Selenium valittiin kuvien ottamiseen sen takia, että ohjelmistosta on monta eri kieliversiota ja kuvien yksittäinen kaappaaminen manuaalisesti ohjelmistosta olisi ollut todella työlästä. Seleniumilla kuvien ottaminen ohjelmoidaan omaksi testiluokakseen, jolloin kuvat saa otettua automatisoidusti testiohjelma suorittamalla. Lisäksi ohjelmiston päivityksen yhteydessä myös kuvat saa helposti päivitettyä ajamalla testiluokka läpi.

Selenium on ohjelmistokehys internetsovellusten testaamisen ja automatisointiin. Seleniumilla on kaksi pääkomponenttia, joita tarkastelen tarkemmin tässä opinnäytetyössä: Selenium IDE ja Selenium WebDriver. IDE:n avulla voi nauhoittaa ja toistaa testejä Firefoxilla. Selenium WebDriver taas toimii siten, että testit ensin kirjoitetaan Seleniumin omalla ohjelmointikielellä Selenesellä, ja sen jälkeen nämä komennot ajetaan selaimessa. WebDriver toimii ilman erillistä serveriä eli se käynnistää suoraan selaimen ja ajaa kirjoitetut komennot selaimessa.

LifeCycle on toteutettu JavaScriptilla sekä Microsoft Visual Studiolla ja C#:lla. Selenium on laajasti käytössä ohjelmistossa myös testaustarkoituksessa. Näiden lisäksi olennainen työkalu projektissani oli internetselain ja sen kehitystyö-

kalut. Tässä projektissa ovat olleet käytössä sekä Google Chrome että Mozilla Firefox.

Alun teoriassa perehdytään lyhyesti siihen, mitä ylipäätänsä tarkoitetaan web-sovelluksella ja sen jälkeen luvuissa 3 ja 4 perehdytään Seleniumiin sekä mobiili- että työpöytäkäytössä. Seleniumin mobiilikäyttöä tarkastellaan tässä opinnäytetyössä vain teorianäkökulmasta, sillä seuraavan LifeCyclen version on tarkoitus toimia myös tabletilla, mutta nykyisessä projektissa ei ole vielä tukea mobiilitestaamiselle, joten olen jättänyt sen tarkastelun tässä opinnäytetyössä ainoastaan teorian tasolle. Luvussa 5 perehdytään itse toteutukseen ja viimeisessä luvussa on kokonaisuuden yhteenveto.

2 YLEISTÄ WEB-SOVELLUKSESTA

World Wide Web on hyvin suuri osa miljoonien ihmisten päivittäistä elämää ympäri maailmaa. World Wide Webin kehityksen myötä nykypäivän internetsivut ovat yhä dynaamisempia ja interaktiivisempia. Webin läsnäolo kaikkialla ja myös meidän riippuvuutemme siitä tekee välttämättömäksi sen, että internetsovellukset ovat laadukkaita, turvallisia ja toimivia. (Li ym. 2014.)

Web-sovelluksen määritelmä

Web-sovellukseksi voidaan määritellä mikä tahansa sovellus, joka käyttää internetselainta asiakkaana. Web-sovellus toimii asiakas-palvelin-periaatteella. Asiakas viittaa siihen ohjelmaan, jota käytetään sovelluksen suorittamiseen. (Nations 2015.) Palvelin taas viittaa tietokantaan johon asiakas ottaa yhteyden, ja joka lähettää asiakkaan pyytämät tiedot takaisin asiakkaan koneelle. Palvelin voi fyysisesti sijaita missä tahansa, ja yhteys muodostetaan vain tiedonsiirron ajaksi. (Li ym. 2014.) Web-sovellukset siis ovat ohjelmia, jotka sallivat käyttäjiensä lähettää ja vastaanottaa tietoja tietokannasta käyttämällä internetselainta. Web-dokumentit generoidaan standardimuodossa (HTML tai XHTML) jolloin sovellukset tukevat kaikkia selaimia. JavaScript taas on asiakaspuolen ohjelmointikieli, joka mahdollistaa dynaamiset elementit sivulla. (Acutenix 2015.)

Web-sovellus muodostuu siis asiakaspuolen ja palvelinpuolen skriptien yhdistelmästä. Asiakaspuolen skriptien tarkoitus on esittää sisältö ymmärrettävässä muodossa käyttäjälle ja palvelinpuolen skriptit taas pitävät huolta tiedon vastaanottamisesta ja käsittelystä. Asiakkaalle näkyvä osa sivustosta on niin kutsuttu front-end eli käyttöliittymäpuoli, kun taas palvelinpuolella sijaitsevaa dataa kutsutan back-endiksi. (Nations 2015.)

Kahdenlaisia web-sovelluksia

Web-sovelluksia on kahdenlaisia, staattisia ja dynaamisia. Staattisessa sivussa sisältö on aina staattista ja se näkyy jokaiselle käyttäjälle samalla tavalla. Staattisessa sivussa sivuston sisältö muuttuu ainoastaan, mikäli sivuston kehittäjä itse tekee siihen muutoksia. Dynaamisella sivustolla taas sivun sisältö muuttuu käyttäjän syötteiden mukaan. Dynaamisella sivulla käyttäjä voi pyytää dataa tietokannasta sekä myös syöttää sinne itse dataa. Staattisen sivun luomiseen riittää pelkkä HTML-koodi, kun taas dynaamiset sivut vaativat toimiakseen erityisiä ohjelmistoja serveripuolella. Esimerkkejä dynaamisista ohjelmointikielistä ovat PHP, Python ja ASP.NET. (How does the internet work? 2015.)

Tämän projektin web-sovellus on monimutkainen dynaaminen sovellus, jonka avulla käyttäjä voi mm. syöttää potilastietoja, tehdä potilashakuja ja muodostaa raportteja. Potilastietojen, äidin verinäytetuloksien ja ultraäänitutkimustulosten perusteella ohjelmisto suorittaa riskilaskentaa sikiölle.

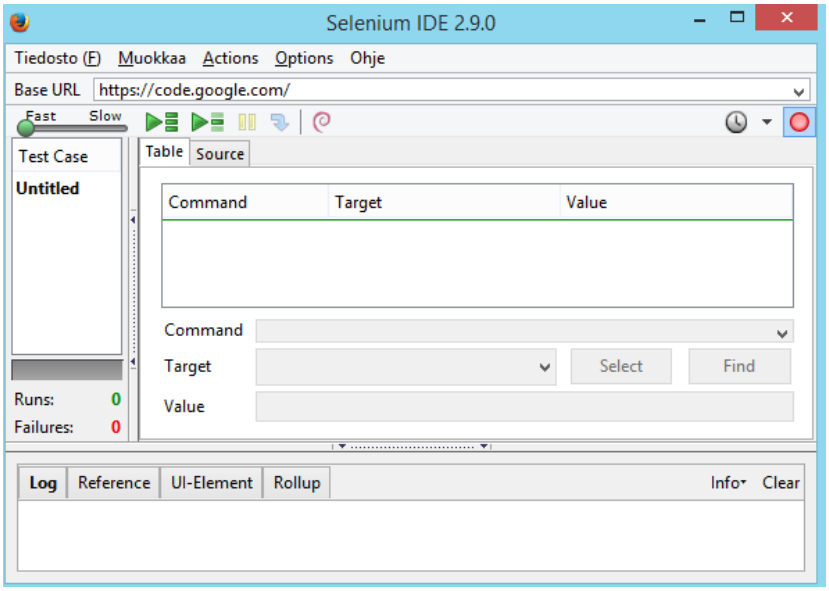
3 SELENIUM

Seleniumin kehitti ohjelmistosuunnittelija Jason Huggins vuonna 2004. Selenium on ohjelmoitu Javalla ja se on avoimen lähdekoodin alustariippumaton ohjelmisto. Selenium koostuu Selenium RC:stä, Selenium Webdriverista sekä Selenium IDE:stä. Selenium RC on Seleniumin ensimmäinen versio, Selenium 1.0. Selenium RC oli pitkään päätuote, mutta sen rajoitusten takia kehitettiin Selenium 2.0, jota myöhemmin alettiin kutsua Selenium Webdriveriksi. Selenium Webdriver on nopeampi kuin RC, koska se kommunikoi suoraan selaimen kanssa, eikä erillisen serverin käynnistäminen ole enää pakollista. Lisäksi Webdriver tukee monipuolisesti eri selaimia sekä myös Ajax-sovelluksia. Selenium IDE on Firefoxin lisäosa, jolla pystyy nauhoittamaan ja editoimaan tapahtumia selaimesta. (Gojare ym. 2015.)

Seleniumin Webdriverin oletusselain on Mozilla Firefox, mutta se tukee myös useita muita selaimia. Lisäksi Selenium tukee useita eri ohjelmointikielellä, kuten Javaa, C#:ia, Pyhonia, Rubya sekä JavaScriptia. (SeleniumHQ 2015a.)

3.1 Selenium IDE

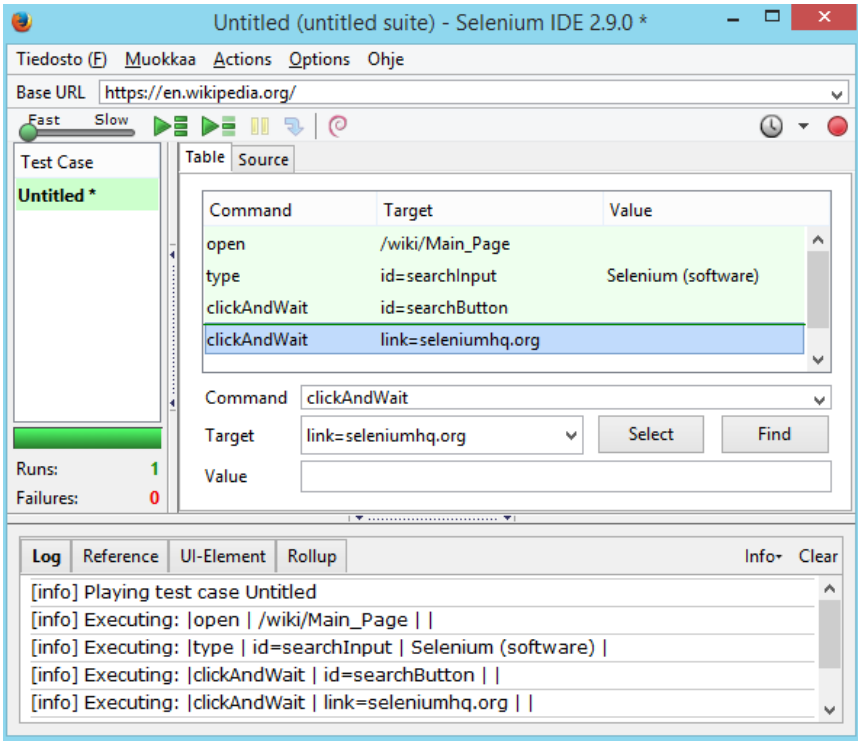
Selenium IDE (Integrated Development Environment) on Firefoxin avoimen lähdekoodin lisäosa, jonka tarkoituksena on tallentaa vuorovaikutus selaimen kanssa. Selenium IDE yhdistettiin osaksi Selenium-projektia vuonna 2006. Selenium IDE nauhoittaa kaikki käyttäjän valinnat ja syötteet selaimen kanssa testitapahtumikseen, joita käyttäjä voi tallentaa ja editoida. (SeleniumHQ 2015b.) Kuvassa 1 on Selenium IDE ennen testitapahtuman aloitusta.



Kuva 1. Selenium IDE.

Base URL ylhäällä kertoo sen sivuston URL:n, mistä lähdetään liikkeelle. Sen alapuolelta navigaatiopalkista voi suorittaa ja pysäyttää nauhoitetun testin sekä aloittaa testin nauhoituksen. Vasemmassa reunassa näkyvät testien nimet ja keskelle Table-välilehteen tulevat näkyviin kaikki nauhoitetut käskyt, niiden kohteet ja arvot. Command-valintaruudukko näyttää kaikki mahdolliset komennot, joita testien tallentamiseen voi käyttää. Value-tekstiruudussa voi muuttaa komennon arvoa. Alapuolella olevasta Log-välilehdestä näkee testien lokin sekä mahdolliset virheilmoitukset. (SeleniumHQ 2015b.)

Tyypillisin tapa luoda testitapauksia on nauhoittaminen, jolloin Selenium IDE tallentaa automaattisesti kaikki käyttäjän syötteet selaimessa. Kuvassa 2 on esimerkki yksinkertaisesta testiajosta, jossa lähdetään liikkeelle Wikipedian aloitussivulta, syötetään hakukenttään ”Selenium (software)”, painetaan hakupainiketta ja vielä siirrytään Seleniumin virallisille sivuille. Testiajon voi tallentamisen jälkeen vielä toistaa, jolloin selain suorittaa kaikki testiajon komennot. Mikäli testiajo ei mene läpi, pystyy Log-välilehdeltä tarkistamaan komennon, josta virhe löytyy. (SeleniumHQ 2015b.)



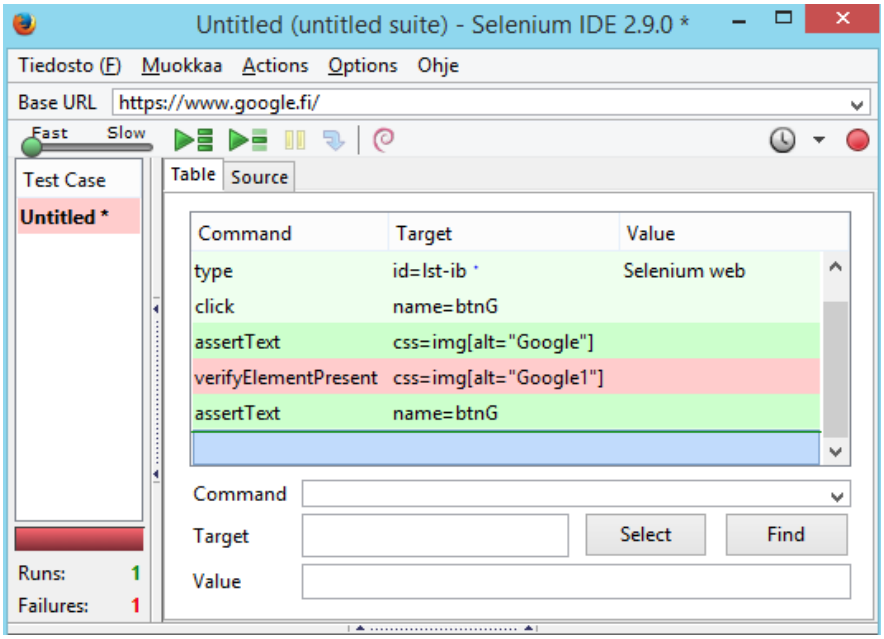
Kuva 2 Yksinkertainen testiajo Selenium IDE:llä.

Seleniumissa käytettävän komentokielen nimi on Selenese. Seleniumin komennot voi jakaa kolmeen eri pääryhmään: *Actions*, joiden voisi sanoa olevan toimintoja, *Accessors*, jotka tarkistavat ohjelman tilan ja *Assertions*, eli eräänlaiset varmistajat, jotka pitävät huolta, että ohjelman tila noudattaa odotettua kulkua.

Actions-komennot yksinkertaisesti vaikuttavat suoraan ohjelman tilaan. Tyypillisiä Actions-komentoja Seleniumilla ovat open, click, clickAndWait, type ja select. Open-komento avaa testin aloitussivun ja odottaa, että sivu on täysin ladattu ennen kuin testi etenee. Click- ja clickAndWait -komennot avaavat linkin. ClickAndWait-komento odottaa lisäksi sivun latautumista ennen testin etenemistä. Type-komento kirjoittaa annetun arvon ja select-komento valitsee vaihtoehdon alasvetovalikosta.

Accessors-komennot tarkistavat ohjelman tilan ja tallentavat halutut arvot muuttujiin. Esimerkkejä Accessor-komennoista ovat storeTitle tai storeElementPresent. Osa Accessors-komennoista tallentaa muistiin varsinaisen arvon ja osa taas boolean-arvon true tai false.

Assertion-kommennot toimivat kuin Accessors-komennot, mutta lisäksi ne varmistavat, että ohjelman tila noudattaa odotettua kulkua. Assertion-komentoja on kolmenlaisia: assert, verify ja waitFor. Esimerkiksi assertText, verifyText ja waitFotText. Jos assert-komento epäonnistuu, koko testiajo keskeytyy. Mikäli verify komento epäonnistuu, testi jatkaa suorittamistaan, mutta kirjaa virheen lokiin (Kuva 3). Kuvassa 3 näkyy, että testiajo on suoritettu loppuun, vaikka verifyElementPresent-komento on epäonnistunut. waitFor-komento taas odottaa että ehto on tosi, ennen kuin testiajo jatkaa suoritustaan. (SeleniumHQ 2015b.)



Kuva 3 Testiajo suoritettuna loppuun asti, vaikka verify-komento on epäonnistunut.

3.2 Selenium Webdriver

WebDriverin tarkoituksena on simuloida täsmällisesti tapaa, jolla käyttäjä on vuorovaikutuksessa selaimen kanssa. Selenium WebDriver tukee useita ohjelmointikieliä. Testiskriptit kirjoitetaan aina lähes samaa syntaksia käyttäen (Selenese) kaikissa ohjelmointikielissä. Tämän opinnäytetyön kaikki koodiesimerkit on luotu C#:ia käyttäen.

3.2.1 Webdriverin historia

Ensimmäinen versio Seleniumista julkaistiin Apachen lisenssin alla vuonna 2004. Tämä versio tunnetaan nimellä Selenium Core ja se oli lähinnä vain kokoelma erinäisiä JavaScript-tiedostoja joilla pystyi automatisoimaan selainta. Vaikka Selenium Coren julkaisun aikana Internet Explorer oli vielä ylivoimaisesti käytetyin selain, tuki Selenium Core monia muitakin selaimia, sillä kaikki muutkin selaimet jolla sitä testattiin, tukivat JavaScriptiä. Selenium Coren ongelma oli, että selain ei antanut JavaScriptin kutsua muita sivuja kuin samaa, jossa testattavan sovelluksen palvelin sijaitsi. Jo Selenium Core kehitettiin kolmisarakkeisella kaavalla joka pohjautui FITiin (Fremework for Integrated Tests). Jokainen taulukon rivi on jaettuna kolmeen sarakkeeseen. Ensimmäinen sarake kertoo suoritettavan komennon nimen, toinen sarake kertoo elementin tunnisteen ja kolmas sarake sisältää valinnaisen arvon (Kuva 4). (Stewart 2010.)

```
type       name=q       Selenium WebDriver
```

Kuva 4 Kolmisarakkeinen komentorakenne.

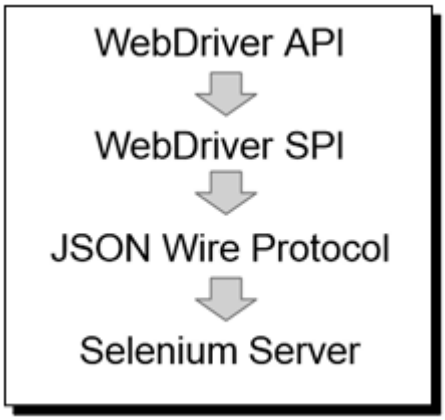
Tämä rakenne on hyvin nähtävillä Selenium IDE:ssä, rakenne vastaa command, target ja value -kenttiä, joista jokainen käydään läpi ennen ohjelman siirtymistä seuraavalle riville. Selenium Coren jälkeen kehitettiin Selenium RC (Remote Control), johon lisättiin http-välityspalvelin. Tämä ratkaisi Coressa ilmenneen puutteen, ja välityspalvelimen avulla päästiin eroon palvelinriippuvuudesta. Selenium RC tuki edelleen kolmiportaista taulukkopohjaista syntaksia, jonka pohjalle Core kehitettiin ja siitä tuli yleisesti tunnettu Selenese, Selenium komennoissa käytettävä ohjelmointikieli. (Stewart 2010.)

HTML5:n tullessa yleiseksi standardikieleksi ja mobiililaitteiden yleistyessä Selenium RC ei pystynyt enää täyttämään kaikkia sille asetettuja vaatimuksia. Näitä vaatimuksia täyttämään kehitettiin Selenium Webdriver. Se kommunikoi suoraan selaimen kanssa eikä vaadi erillistä palvelinta toimiakseen. Tämä tekee Webdriverin käytöstä huomattavasti nopeampaa kuin RC:n. RC toimi samalla

tavalla jokaisessa selaimessa eli se ajoi samat JavaScipt-funktiot jokaisessa selaimessa ja käytti näitä funktiota ajaakseen suoritettavissa olevan ohjelman läpi. WebDriver taas käyttää jokaisen selaimen sisäänrakennettua tukea ohjelman suorituksessa. WebDriver ja RC olivat alun perin kaksi erillistä projektia, mutta vuonna 2009 ne yhdistettiin ja tunnetaan nykyään nimellä Selenium 2.0 tai Selenium WebDriver. (Stewart 2010.)

3.2.2 Arkkitehtuuri

Selenium WebDriver kommunikoi suoraan selaimen kanssa ilman erillisiä JavaScript-kirjastoja. Toisin sanoen se kontrolloi selainta selaimen ulkopuolelta. WebDriver käyttää kunkin selaimen kohdalla sopivinta tapaa ajaa testit. Esimerkiksi Mozilla Firefox käyttää JavaScriptiä ja Internet Explorer C++:aa. Tämä lähestymistapa takaa, että selainta voidaan kontrolloida parhaalla mahdollisella tavalla, mutta tarkoittaa myös sitä, että uudella selaimella ei automaattisesti ole tukea. Selenium WebDriverin arkkitehtuuri koostuu neljästä eri osasta (Kuva 5).



Kuva 5 WebDriverin arkkitehtuuri

WebDriver API on se osa järjestelmää, jonka kanssa käyttäjä on koko ajan vuorovaikutuksessa. WebDriver API:lla kirjoitetaan testikoodit halutulla ohjelmointi-

kielellä. Kuvassa kuusi on yksinkertainen koodiesimerkki, joka käyttää WebDriver- ja WebElement.olioita. Tutustumme näihin tarkemmin luvussa 3.2.3.

```
driver.findElement(By.name("q"))
```

Kuva 6. Koodiesimerkki elementin paikallistamisesta.

Tämän jälkeen komennot siirretään WebDriver SPI:lle (Stateless Programming Interface). SPI selvittää mikä elementti on ja sen jälkeen kutsuu asiaankuuluvaa komentoa. Kuvan 6 koodiesimerkki SPI vaiheessa on kuvattu kuvassa 7.

```
findElement(using="name", value="q")
```

Kuva 7. Kuvan 6 koodiesimerkki SPI vaiheessa.

Tämän jälkeen kutsutaan JSON Wire -protokollaa (JOSNWP), joka on WebDriverin kehittäjien luoma siirtomekanismi. Se on siirtomekanismi serverin ja asiakkaan, tässä tapauksessa ohjelmoijan välillä. JSONWP siis välittää elementit koodiin. (Burns 2012.)

3.2.3 Elementtien paikallistaminen

Ensimmäinen asia testien kirjoittamisessa on paikallistaa web-elementit sivulta. Elementtien paikallistamiseen Seleniumi API:n luokkakirjasto tarjoaa useita eri keinoja. Paikallistajat sijaitsevat IWebelement-luokassa. Myöhemmin tässä luvussa esittelen yleisimmät paikallistajat. Elementin paikallistamiseen käytetään aina metodia FindElement, jolloin sivuilta paikallistetaan yksi tietty elementti. Elementtejä voi paikallistaa myös käyttämällä metodia FindElements, jolloin metodi palauttaa listan elementeistä annetuilla arvoilla. Koodiesimerkissä 1 on nähtävissä eri tapoja, jolla elementtejä paikallistetaan sivulta. Keinoja on useita lisää, mutta esimerkissä on esitettynä yleisimmät tavat. (SeleniumHQ 2015c.)

```
//paikallista elementti ID:n perusteella
IWebElement element1 = driver.FindElement(By.Id("searchInput"));

//paikallista elementti luokan nimen perusteella
IWebElement element2 = driver.FindElement(By.ClassName("searchButton"));

//paikallista lista elementtejä luokan nimen perusteella
IList<IWebElement> list1 = driver.FindElements(By.ClassName("searchButton"));

//paikallista elementti tagin nimen perusteella
IWebElement element3 = driver.FindElement(By.TagName("iframe"));

//paikallista lista elementtejä tagin nimen perusteella
IList <IWebElement> list2 = driver.FindElements(By.TagName("iframe"));

//paikallista elementti nimen perusteella
IWebElement element4 = driver.FindElement(By.Name("search"));

//paikallista elementti linkin tekstin perusteella
IWebElement element5 = driver.FindElement(By.LinkText("Google"));

//paikallista elementti XPATHIN avulla
IWebElement element6 = driver.FindElement(By.XPath("//input"));

//paikallista elementti CSS:n avulla
IWebElement element7 = driver.FindElement(By.CssSelector("input.gsfi"));
```

Koodiesimerkki 1. Yleisimmät tavat paikallistaa HTML-elementti.

Paikallistajat voi jakaa kahteen osaan; sellaisiin, jotka ovat riippuvaisia sivun rakenteesta löytääkseen elementin, ja sellaisiin, joissa paikallistajat etsivät elementin tietyn attribuutin perusteella, jolloin sivun rakenteella ei ole merkitystä elementin paikallistamisen kannalta. Koodiesimerkissä 1 esitetyistä tavoista Xpath ja CSS ovat riippuvaisia sivun rakenteesta, ja muut ovat attribuuttiperusteisia paikallistajia. Mikäli annetuilla arvoilla on useita vastaavuuksia, attribuuttiperusteiset paikallistajat palauttavat ensimmäisen vastaavan elementin. (Newcircle 2013.)

Attribuuttiperusteiset paikallistajat

Elementin paikallistaminen ID:n, luokan nimen ja name-attribuutin perusteella toimivat hyvin samoilla tavoin. Elementin etsiminen ID:llä on nopein tapa löytää elementti, joten mikäli elementiltä ID löytyy, kannattaa sitä käyttää paikallistami-

seen. Lisäksi aikaisemmin suosittiin tätä tapaa sillä ID oli yleensä aina yksilöllinen jokaisella elementillä. ID:n avulla paikallistamista ei kuitenkaan voi käyttää, mikäli elementin ID generoidaan automaattisesti. (Newcircle 2013) Mikäli elementillä ei ole yksilöllistä ID:tä, voi sen paikallistaa luokan nimen avulla, mutta käytännössä tämä ei aina ole toimiva tapa, sillä samaa luokan nimeä voi olla käytetty useassa eri elementissä. Elementin paikallistaminen name-atrribuutin avulla toimii kuten ID, ja sitä käytetään lähinnä input-kenttien paikallistamiseen, sillä name-attribuutit eivät ole enää muutoin tuettuja HTML5:ssä. (Selenium Python Bindigs 2014.)

Mikäli elementtiä ei pysty paikallistamaan attribuuttien perusteella, voi myös vaihtoehtoisesti käyttää paikallistamista linkin tekstin tai tagin nimen perusteella. Linkin teksti paikallistaa tekstin joka on näkyvä teksti sivulla. Elementin pystyy paikallistamaan myös osittaisen linkin tekstin perusteella, jolloin käytetään syntaksia PartialLinkText. Tagin nimen perusteella pystyy paikallistamaan mitä tahansa HTML-tageja, kuten koodiesimerkin 1 iframe, mutta yleensä näitä on useita sivulla, joten tämä ei ole kovin käytetty tapa. (Selenium Python Bindigs 2014.)

Rakennepohjaiset paikallistajat

Paikallistaminen CSS:n ja Xpathin avulla toimii eri tavoin kuin edellä kuvatut tavat. XPathilla paikallistetaan elementtejä XML-dokumentista polkurakenteen avulla. Jokaisella selaimella on omanlainen tuki XPathille tai välttämättä tukea ei ole ollenkaan. Tämä tarkoittaa, että paikallistamiseen käytettävä syntaksi vaihtelee eri selaimilla. XPathia käytetään yleensä, jos elementille ei löydy yksilöivää ID- tai luokka -attribuuttia. (Seleniumhq.org 2015c) XPathilla pystyy hakemaan elementin absoluuttisen sijainnin perusteella, tai vaihtoehtoisesti viittaamaan lähellä olevaan ID- tai class -attribuuttiin, jolloin kohde-elementtiin voi viitata sen perusteella onko se child- vai parent-element paikallistetulle attribuutille. (Selenium Python Bindigs 2014.)

Elementtejä voi paikallistaa myös CSS kyselyjen avulla. Myös CSS:n avulla paikallistaminen on selainriippuvaista. Kuten XPathia, kaikki selaimet eivät myöskään välttämättä tue CSS -kyselyjä. CSS-kysely toimii hyvin samalla tavalla kuin XPath, ja on jopa XPathia nopeampi tapa paikallistamiseen ja toimii erityisesti Internet Explorerin kanssa nopeammin. (Newcircle 2013.)

3.2.4 Ohjelman rakenne

Kirjoitettaessa testiohjelmia Selenium WebDriverilla ohjelmat rakennetaan dynaamisesti siten, että kaikille toiminnoille rakennetaan omat metodit. Yksinkertainen ohjelma rakennetaan siten, että yhdessä metodissa käynnistetään selain ja navigoidaan oikealle sivulle, toisessa suljetaan selain ja kolmannessa tehdään itse testi. Kaikki eri toiminnot rakennetaan omiksi metodeikseen. Mikäli samaa elementtiä käytetään monessa eri suoritusvaiheessa, kannattaa elementin paikallistaminenkin kirjoittaa omaksi metodikseen. Koodiesimerkissä 2 on nähtävillä yksinkertaisen testin rakenne C#:lla kirjoitettuna.

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;

namespace SeleniumTest
{
    [TestClass]
    public class SimpleTestStructure
    {
        IWebDriver driver;

        //varsinainen testimetodi, jossa ohjelman toiminnallisuus suoritetaan
        [TestMethod]
        public void Search()
        {
            IWebElement searchInput = driver.FindElement(By.Name("q"));
            searchInput.SendKeys("Selenium");
            searchInput.SendKeys(Keys.Enter);
        }

        //alustetaan testiohjelma ja navigoidaan oikealle sivulle
        [TestInitialize]
        public void Setup()
        {
            driver = new FirefoxDriver();
            driver.Navigate().GoToUrl("https://www.google.com/");
        }
```

```
        //lopetetaan testi
        [TestCleanup]
        public void TestCleanUp()
        {
            driver.Quit();
        }
    }
}
```

Koodiesimerkki 2. Yksinkertainen testiohjelma C#:lla.

Koodiesimerkissä on kolme metodia Search(), jossa tehdään itse testi, Setup(), jossa määritellään mitä selainta käytetään, käynnistetään selain ja navigoidaan haluttuun osoitteeseen, sekä TestCleanUp(), joka sulkee selaimen ja lopettaa testin. Lisäksi testin alussa tuodaan Selenium-paketit projektiin mukaan using-syntaksia käyttäen. Esimerkissä käytetään Firefoxia selaimena, joka määritetään syntaksilla IWebdriver driver = new FirefoxDriver(). Mikäli käytetään selaimena Google Chromea, kutusutaan ChromeDriver:ia ja Internet Exploreria InternerExplorerDriver. Halutulle sivulle navigoidaan driver.Navigate()-funktion avulla. Selain suljetaan ja ohjelman suoritus lopetetaan driver.Quit()-funktiolla.

4 SELENIUM MOBIILIKÄYTÖSSÄ

Seleniumilla on tuki kahdelle suurimmalle mobiilialustalle; Andoidille ja iOS:lle. Mobiilitestaus on kuitenkin lähivuosina yleistynyt niin paljon, että Seleniumin rinnalle on kehitetty Seleniumiin pohjautuvia erillisiä avoimen lähdekoodin testausympäristöjä. Näistä yleisimmät ovat Appium ja Selendroid sekä iOS driver. Mobiilitestit voidaan ajaa joko oikeassa laitteessa tai erillisen simulaattorin avulla. Oikeassa laitteessa ajettavissa testeissä laitteet yhdistetään koneeseen USB-kaapelin avulla. Appiumialla voi testata sekä Android- että iOS-alustoilla pyöriviä sovelluksia. Selendroid taas on suunniteltu pelkästään Android-alustalla toimivien laitteiden testaamiseen ja iOS driver iOS-alustoilla toimivien laitteiden testaamiseen. (Appium 2015; Selendroid 2015a.)

4.1 Appium

Appiumin toiminta-ajatuksena on, että testit kirjoitetaan kerran ja sen jälkeen niitä pystyy testaamaan eri laitteilla ja alustoilla ilman erillistä muokkaamista. Lisäksi Appium tukee samoja ohjelmointikieliä kuin Selenium WebDriver ja se on alustariippumaton, joten samat testit voi ajaa eri selaimilla. Appiumilla voi myös testata mobiililaitteen natiiveja sovelluksia. Appiumilla pystyy testaamaan Androidin versiosta 4.2 ylöspäin, tätä vanhemmissa Androidin versioissa täytyy käyttää Selendroidia. (Appium.io 2015.) Seleniumin toimii Appiumin pohjana, ja lisäksi Appium vaatii toimiakseen Appium-serverin, jonka avulla se pyörittää ohjelmia oikeissa laitteissa tai simulaattoreissa. Appium vaatii Android-alustojen testaamiseen Android SDK:n, ja iOS-alustojen testaamiseen iOS SDK:n. iOS-alustoilla toimivia laitteita ei pysty testaamaan Windows-ympäristössä, vaan Windows-koneelle täytyy ladata Mac virtualbox, mikäli haluaa tehdä iOS-testejä Windows-ympristössä. iOS-ympäristössä kuitenkin pystyy testaamaan Android-laitteita. (Tino 2014.)

Appium toimii siten, että testit kirjoitetaan ensin halutulla ohjelmointikielellä ja käynnistetään kehitysympäristöstä, sen jälkeen Appium-serveri ottaa yhteyden

kytkettyyn laitteeseen ja suorittaa testit. Testejä ei siis voi suoraan suorittaa laitteessa ilman erillistä Appium-serveriä joka välittää testit laitteeseen. Koodiesimerkissä 3 on kuvattu koodiesimerkin 2 ohjelma suoritettuna Android-laitteella.

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Appium;
using OpenQA.Selenium.Appium.Interfaces;
using OpenQA.Selenium.Appium.MultiTouch;
using OpenQA.Selenium.Interactions;
using OpenQA.Selenium.Remote;
using OpenQA.Selenium.Appium.Android;

namespace AppiumTest
{

[TestClass]
public class TestAppium
{
    public IWebDriver driver;

    //varsinainen testimetodi, jossa toiminnallisuus suoritetaan
    [TestMethod]
    public void Search()
    {
        IWebElement searchInput = driver.FindElement(By.Id("q"));
        searchInput.SendKeys("Selenium");
        searchInput.SendKeys(Keys.Enter);
    }

    //alustetaan laite ja asetetaan Appium serveriksi
    //ja navigoidaan haluttuun osoitteeseen
    [TestInitialize]
    public void SetUp()
    {
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.SetCapability("device", "Android");
        capabilities.SetCapability("deviceName", "H60-L04");
        capabilities.SetCapability("browserName", "chrome");
        capabilities.SetCapability("platformName", "Android");
        capabilities.SetCapability("platformVersion", "4.4.2");
        capabilities.SetCapability("deviceName", "Huawei honor 6");

        driver = new RemoteWebDriver(new Uri("http://127.0.0.1:4723/wd/hub"),
       capabilities, TimeSpan.FromSeconds(180));
        driver.Navigate().GoToUrl("http://www.google.com");
    }

    //lopetetaan testi
    [TestCleanup]
    public void End()
    {
        driver.Quit();
    }
}
```

Koodiesimerkki 3. Yksinkertainen koodiesimerkki mobiililaitteella suoritetusta testistä Appiumilla toteutettuna

Kuten esimerkistä huomataan, Appiumille täytyy määrittää laite jota käytetään. Tähän käytetään DesiredCapabilites-luokkaa. Luokka sisältää tärkeimmät parametrit testiympäristön alustamiselle. Lisäksi alussa tuodaan Appium-paketit mukaan projektiin ja driveriksi asetetaan RemoteWebDriver, joka siis viittaa Appiumin serveriin, eikä suoraan selainta kuten pelkkää Seleniumia käytettäessä tehdään. Itse ohjelman suoritus alustuksen jälkeen on kuitenkin täysin samanlainen, joten varsinaista koodia ei muuten tarvitse muuttaa. (Tino 2014.)

4.2 Selendroid ja iOS driver

Selendroid on kehitetty pelkästään Android-alustoilla toimivien laitteiden testaamiseen. Siinä on tuki Androidin versioon 4.4 asti, eli se ei tue enää Androidin uusinta Lollipop-käyttöjärjestelmää. Myös Selendroid vaatii toimiakseen Android SDK:n, sekä myös Java SDK:n. Selendroid tarjoaa virallisilla sivuilla koodiesimerkkejä testaamiseen Javalla, Pythonilla ja Rubylla, mutta testaaminen onnistuu myös C#:lla. Testaamiskaava Selendroidilla on hyvin samankaltainen kuin Appiumilla. Selendroi ladataan koneelle .jar-tiedostona, mutta mitään varsinaista sovellusta ei asenneta, vaan asetusten konfiguroimiseen ja testien tulosteiden tarkasteluun käytetään komentoriviä. RemoteWebdriveriksi asetetaan luonnollisesti Selendroid-server, joka välittää komennot laitteeseen. Koodiesimerkissä 4 on aiemmin nähty koodi Selendroidilla ja Android-alustalla suoritettuna. (Selendroid 2015b.)

```
using System;
using System.Threading;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Interactions;
using OpenQA.Selenium.Remote;

namespace SeleniumTest
{
    [TestClass]
    public class SimpleTestStructureSelendroid
    {
        public IWebDriver driver;

        //testimetodi, jossa varsinainen toiminnallisuus suoritetaan
        [TestMethod]
        public void Search()
        {
            IWebElement searchInput = driver.FindElement(By.Name("q"));
            searchInput.SendKeys("Selenium");
            searchInput.SendKeys(Keys.Enter);
        }

        //alustetaan testimetodi asettemalla Selendtroid-server ja
        //navigoidaan haluttuun osoitteeseen
        [TestInitialize]
        public void Setup()
        {
            DesiredCapabilities capabilities =
            OpenQA.Selenium.Remote.DesiredCapabilities.Android();
            capabilities.SetCapability
            ("app", "io.selendroid.androiddriver:0.16.0");
            capabilities.SetCapability
            ("mainActivity", "io.selendroid.androiddriver.WebViewActivity");

            driver = new RemoteWebDriver(new Uri("http://localhost:4444/wd/hub"),
            capabilities, TimeSpan.FromSeconds(5000));
            driver.Navigate().GoToUrl("https://www.google.com/");
        }

        //lopetetaan testi
        [TestCleanup]
        public void TestCleanUp()
        {
           driver.Quit();
        }
    }
}
```

Koodiesimerkki 4. Yksinkertainen Selendroid-ohjelma

Esimerkissä ei ole juurikaan eroja verrattuna testien suorittamiseen Appiumilla. Koen kuitenkin Appiumin käyttämisen kätevämmäksi verrattuna Selendroidiin

sen laajemman Android tuen takia, ja lisäksi Appium tukee sekä Android- että iOS-alustoilla toimivia laitteita. Windows-laitteiden tukea kummallakaan testausympäristöllä ei ole.

iOS-alustoilla toimivan iOS driverin toimintaperiaate on samanlainen kuin Selendroidin. Se vaatii toimiakseen iOS SDK:n ja ladataan Selendroidin tavoi .jar-tiedostona koneelle, ja myös sitä käytetään komentorivin kautta. iOS driver ei kuitenkaan tue testaamista oikealla laitteella, vaan testaamisessa on käytettävä simulaattoria. Appiumin ja Selendroin tavoin koodin alustamiseen käytetään DesiredCapabilities-luokkaa. (Ios-driver 2015.)

5 TOTEUTUS

Opinnäytetyön tarkoituksena oli toteuttaa automatisoitujen kuvakaappausten otto Selenium WebDriverilla. Kuvakaappaukset kopioidaan sen jälkeen LifeCycle-ohjelmiston ohjetiedostoon. Ideana oli, että kuvakaappausten ottaminen automatisoidaan, jolloin vältytään suurelta määrältä manuaalista työtä, sillä kuvakaappauksia tulee monia satoja huomioiden ohjelmiston kaikki eri kieliversiot. Eri kieliversioita on tällä hetkellä suomi mukaan luettuna yhteensä kahdeksan. Lisäksi tämä takaa sen, että help-tiedostoissa on aina päivitetyt versiot kuvista. Selenium on ollut laajasti käytössä LifeCycle-projektissa myös testaustarkoituksessa, joten ohjelmistolla oli jo valmiina tuki Seleniumin käytölle.

5.1 Tekninen toteutus

Koko projekti on toteutettu Microsoft Visual Studio -kehitysympäristössä. Tein oman projektini omaan luokkaansa, mutta projektissa oli paljon valmiita luokkia ja metodeita joita pystyin käyttämään hyödyksi omassa projektissani. Omassa luokassani paikallistin elementit joista kuvat otetaan, ja kutsuin eri luokassa olevaa metodia varsinaiseen kuvakaappausten ottamiseen.

5.1.1 Kuvakaappausten ottaminen

Seleniumissa on valmiina tuki kuvakaappausten ottamiseen sekä tallentamiseen. Kuvakaappaus toteutetaan GetScreenshot()-metodilla joka sijaitsee ITakeScreenshot-rajapinnassa.

```
var screenshotDriver = m_driver as ITakesScreenshot;
Screenshot screenshot = screenshotDriver.GetScreenshot();
```

Koodiesimerkki 5. Kuvakaappauksen ottaminen Seleniumilla.

Kuvakaappausten ottaminen omassa projektissani tapahtui luokassa WebTest-Helpers, jossa oli oma metodi Screenshot(), jolla kuvakaappaukset otettiin. Koodiesimerkissä 5 näkyy syntaksi jolla kuvakaappausten ottaminen on projektissani toteutettu. Itse kuvakaappausten lisäksi metodissa määriteltiin myös se, että mihin kansioon kuvakaappaukset talletetaan ja missä tiedostomuodossa, sekä lisäksi se, että otetaanko kuva koko sivulta vai vain tietystä elementistä. Help-tiedostoon tulevista kuvista lähes kaikki kuvat otettiin tietystä elementistä. Koodiesimerkissä 6 on kuva koko Screenshot()-metodista.

```
public void Screenshot(string fileNameWithoutExtension, bool languageDir = false,
IWebElement element = null, bool helpFile = false, int x = -1, int y = -1, int w = -1, int h
= -1)
        {
            try
            {
                Thread.Sleep(500); // Time to draw screen
                var screenshotDriver = m_driver as ITakesScreenshot;
                Screenshot screenshot = screenshotDriver.GetScreenshot();
                var bmpScreen = new Bitmap(new MemoryStream(screenshot.AsByteArray));
                UTLogger.Log("Screenshot taken on page: " + GetPageUrl(false));
                // replace invalid characters with '_' for file name
                fileNameWithoutExtension =
Path.GetInvalidFileNameChars().Aggregate(fileNameWithoutExtension, (current, c) =>
current.Replace(c.ToString(), "_"));
                string file = GetFullFileName(fileNameWithoutExtension + ".png");

                if (languageDir)
                {
                    string testRoot = GetTestDirForLng();
                    string lng = GetLocalizationId();

                    if (element != null || helpFile == true)
                    {
                        Directory.CreateDirectory(Path.Combine(testRoot, lng + "_help"));
                        file = Path.Combine(testRoot, lng + "_help",
fileNameWithoutExtension + ".png");
                    }

                    if (element == null && helpFile == false)
                    {
                        Directory.CreateDirectory(Path.Combine(testRoot, lng));
                        file = Path.Combine(testRoot, lng, fileNameWithoutExtension +
".png");

                    }
                    Assert.IsFalse(File.Exists(file), "File already exists: " + file);
                }
```

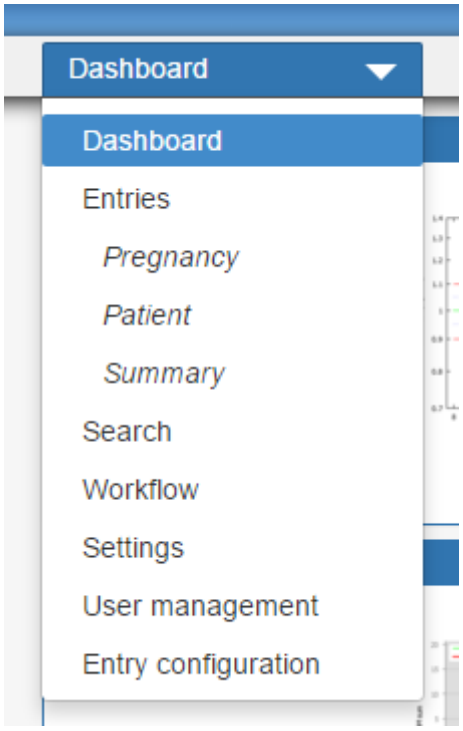
```
                if (element == null && x == -1)
                {
                    bmpScreen.Save(file, ImageFormat.Png);
                    UTLogger.Log("Screenshot saved in: " + file);
                }

                if (element != null && x == -1)
                {
                    var cropArea = new Rectangle(element.Location, element.Size);
                    Bitmap bm = bmpScreen.Clone(cropArea, bmpScreen.PixelFormat);
                    bm.Save(file, ImageFormat.Png);
                }

                if (element == null && x != -1)
                {
                    var cropArea = new Rectangle(x, y, w, h);
                    Bitmap bm = bmpScreen.Clone(cropArea, bmpScreen.PixelFormat);
                    bm.Save(file, ImageFormat.Png);
                }

            }
            catch (Exception ex)
            {
                UTLogger.Log("Error in screenshot; file: " + fileNameWithoutExtension + ",
ex: " + ex.Message);
            }
        }
```

Koodiesimerkki 6. Screenshot()-metodi

Metodissa olevilla parametreilla määritetään kuvakaappauksen tiedostonimi sekä se, että tallennetaanko kuva kielitiedostoihin, otetaanko kuva pelkästä elementistä ja se, meneekö kuva help-kuville tarkoitettuun tiedostoon vai ei. Lisäksi määritetään kuvakaappausten mahdolliset koordinaatit, mikäli kuvaa ei oteta tietystä elementistä vaan koko sivulta. Riippuen mitä parametreja metodille on annettu, if-lauseilla toteutetaan oikeat toimenpiteet. Varsinaisille elementtikohtaisille kuville ei tarvitse määrittää sijaintia, sillä Seleniumin WebElement-luokassa on valmiina location- ja size-metodit joilla pystyy määrittämään elementin koordinaatit sekä koon. Elementtikohtaiset kuvat toteutetaan siten, että ensin otetaan kuva koko sivulta, ja sen jälkeen kopioidaan kuva ja annetaan sille kooksi elementin koko ja sijainniksi elementin sijainti ja sen jälkeen tallennetaan tämä kopio.

5.1.2 Testiluokan rakenne

Varsinainen luokka jossa käyttöliittymästä otettavien kuvien sijainti määritellään, on rakennettu siten, että jokaisella LifeCycle-ohjelmistossa olevalle sivulle tehtiin oma metodi kuvakaappausten ottamiseen. LifeCycle-ohjelmisto koostuu aloitussivun lisäksi kuudesta eri sivusta sekä lisäksi Entry-sivun kolmesta eri alasivusta jotka ovat nähtävissä ohjelmiston navigaatiopalkista kuvassa 8.



Kuva 8. Ohjelmiston eri sivut navigaatiopalkissa.

Varsinainen testimetodi, jossa eri sivuille tehtyjä omia metodeita kutsutaan, on nimeltään HelpScreenshots(), joka on esitettynä koodiesimerkissä 6. Tämä metodi on rakennettu siten, että ensin taulukoidaan kaikki eri kieliversiot ja käydään ne ohjelmassa yksi kerrallaan läpi foreach-lausetta käyttäen. Sen jälkeen kutsutaan ensimmäisenä sisäänkirjautumismetodia ja käydään tämän jälkeen kaikki ohjelman sivut läpi omissa metodeissaan ja lopuksi kirjaudutaan ulos. Lopussa myös määritellään, että mitä selainta käytetään. HelpScreenshots()-metodi on toteutettu Firefoxia käyttäen.

```
        [TestMethod]
        [TestCategory(LCTestCategories.GUITest)]
        public void HelpScreenshots()
        {
            SeleniumTestHelper.RunTest((helper) =>
            {
                //add patient with calculated risks
                UTHelpers.DeleteAllPatientData();
                string patId = UTHelpers.UniqueId("Localization");
                UTHelpers.AddPatientWithCalculationData(patId, lastName: "Customer",
firstName: "Test", calculate: true, calcProtocol: UTHelpers.CONFIGURATION_LC40,
testDirectory: TestContext.DeploymentDirectory);

       foreach (string lng in new[] { "en-US", "zh-CN", "es-ES", "fr-FR", "ru-RU", "zh-CN",
       "fi-FI" })
                {
                    m_CurrentLang = lng;
                     UTLogger.Log(Environment.NewLine + "*** Starting help page picture
                     capturing for language: " + m_CurrentLang);

                    MasterPage mp = helper.GetPageObject<MasterPage>();

                    LoginHelp(helper);

                    DashboardViewHelpPictures(helper);

                    EntryViewHelpPictures(helper, patId: patId);
                    EntryViewHelpPictures(helper, NavViews.Entry_Patient);
                    EntryViewHelpPictures(helper, NavViews.Entry_Summary);

                    SearchViewHelpPictures(helper);

                    WorkflowHelpPictures(helper);

                    SettingsViewHelpPictures(helper);

                    UserManagementViewHelpPictures(helper);

                    EntryConfigurationViewHelpPictures(helper);

                    mp.ClickLogout();

                    UTLogger.Log("    Picture capturing ended for language: " +
                    m_CurrentLang + Environment.NewLine);
                }
            },
            TestContext, new SeleniumDriverType[] { SeleniumDriverType.Firefox});
        }
```

Koodiesimerkki 7. HelpScreenshot()-metodi

Eri metodien sisällä määritetään, että mistä elementistä kussakin metodissa otetaan kuvia. Koodiesimerkissä 8 on nähtävissä toteutus Entry Configuration -sivulta. Metodin alussa navigoidaan ensin oikealle sivulle ja otetaan ensin yksi yleiskuva näkymästä ja sen jälkeen listataan taulukkoon kaikki elementit joista kuvia halutaan ottaa. Taulukossa määritellään ensin elementin sijainti, nimetään kuva ja määritellään mahdolliset edellä olevat elementit joita tulee klikata, ennen kuin elementti josta varsinainen kuva halutaan tulee näkyviin. Elementti josta kuva otetaan, on paikallistettu XPAthin tai ID:n avulla. Ensisijaisesti käyte-

tään ID:tä, mutta mikäli sitä ei ole, eikä myöskään yksilöivää luokan nimeä, käytetään XPathia. Lopuksi kutsutaan metodia TakePicture(), joka suorittaa parametrien edellyttämät toimenpiteet ennen kuin varsinaista TakeScreenshot()-metodia kutsutaan.

```
/// <summary>
/// This is for taking element pictures to Helpfile of EntryConfigurationview
/// </summary>
/// <param name="helper">WebTestHelper</param>
private void EntryConfigurationViewHelpPictures(WebTestHelper helper)

{
  MasterPage mp = helper.GetPageObject<MasterPage>();

  mp.NavigatetoView(NavViews.EntryConfig, true);
  EntryConfig config = helper.GetPageObject<EntryConfig>();

  string page = "EntryConfig";

//one general picture of the view
helper.Screenshot(page, languageDir: true, helpFile: true, x: 20, y: 40, w: 1200, h: 350);

List<TakeHelpPicture> list = new List<TakeHelpPicture> { };
list.Add(new TakeHelpPicture(picture: By.Id("toolbarButtons"),
pictureName:"ToolbarButtons"));
list.Add(new TakeHelpPicture(picture: By.Id("entryDefBar"), pictureName: "NavBar"));
list.Add(new TakeHelpPicture(picture:
By.XPath("//div[@id='mainGroupTableToolbar']/parent::div/parent::div"),
pictureName:   "EntrySections"));
list.Add(new TakeHelpPicture(picture:
By.XPath("//div[@id='subGroupTableToolbar']/parent::div/parent::div"),
pictureName: "FieldGroups"));
list.Add(new TakeHelpPicture(picture:
By.XPath("//div[@id='fieldTableToolbar']/parent::div/parent::div"), pictureName: "Fields"));
list.Add(new TakeHelpPicture(click: By.Id("btnToolbarGroups"), picture:
By.XPath("//div[@id='userGroupTableContainer']/parent::div/parent::div/parent::div/parent::d
iv/parent::div"), pictureName: "UserGroups", bootBoxdialog: true, loadingImage: true));
list.Add(new TakeHelpPicture(click: By.Id("btnToolbarGroupDefaults"), picture:
By.XPath("//div[@id='userGroupDefaultsTableContainer']/parent::div/parent::div/parent::div/p
arent::div/parent::div"), pictureName: "DefaulUserGroups",
bootBoxdialog: true, loadingImage: true));


    for (int j = 0; j < list.Count; j++)
       {
           TakePicture(config, helper, page, list[j]);
       }
}
```

Koodiesimerkki 8. EntryConfigurationViewHelpPictures()-metodi

5.2 Seleniumin soveltuvuus kuvakaappausten ottamiseen

Selemium ajoi projektissa tarkoituksensa hyvin ja erityisesti elementtikohtaisten kuvien ottaminen onnistui Seleniumilla hyvin. Joissain tilanteissa oli kuitenkin selkeämpää, että kuvassa on elementtiä hieman laajempi alue näkyvillä. Tällai-

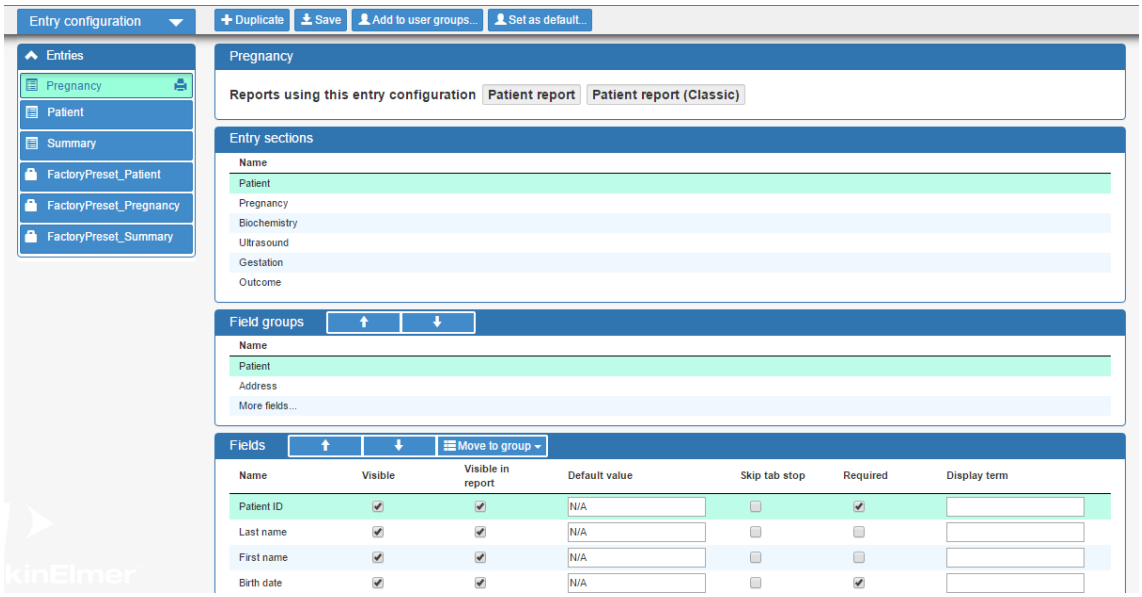
sia tilanteita ovat esimerkiksi alasvetovalikot. Elementtikohtaisen kuvan ottaminen rajoittaa kuvan tiettyyn elementtiin, jolloin taustan mukaan ottaminen ei ole mahdollista. Mikäli kuvaan halusi taustaa mukaan, tai kuva otettiin useammasta elementistä tai lähes koko sivusta kuten ohjelmiston jokaisen sivun yleiskuva, piti kuvalle määrittää erikseen x- ja y-koordinaatit sekä myös leveys ja korkeus. Tässä menetelmässä on myös otettava huomioon, että eri näytöillä on erilainen resoluutio, jolloin kuva ei tule jokaisesta näytöstä täysin samasta kohdasta. Näiden seikkojen takia lähes kaikki mahdolliset kuvat otettiin elementtikohtaisesti. Ainoastaan sellaisissa tapauksissa, jossa kuvaan oli olennaista saada taustaa mukaan tai kuva oli isosta alueesta, otin kuvan erikseen määriteltyjen koordinaattien perusteella.

Oma vaiheensa projektissa oli myös eri selainten testaaminen ja sopivimman valinta kuvakaappausten ottamiseen. Kaikki selaimet käyttäytyvät Seleniumin kanssa hieman eri tavoin, vaikka koodissa annetut käskyt ovatkin pääsääntöisesti täysin samat. Chrome toimii siten, että se ottaa kuvaan mukaan vain näkyvän osan sivusta, vaikka sivu olisikin vieritettävä. Firefox ja Internet Explorer (IE) taas ottavat kuvankaappauksiin mukaan koko sivun alueen, myös sen mikä ei ole näkyvissä. IE kuitenkin toimii siten, että näkyvän osan ulkopuolelle jäävä alue on täysin mustaa. Tämän takia IE suljettiin ensimmäisenä vaihtoehdoista pois. Chrome taas toimii hyvin sellaisissa tilanteissa, jossa halutaan ottaa kuva sellaisen sivun näkyvältä osalta, jossa on paljon alaspäin rullattavaa. Firefox taas ei ole tällaisessa tilanteessä kätevä, sillä se ottaa ainoastaan yhden kuvan, joka siis kasvaa niin korkeaksi kuin sivun todellinen korkeus on. Ohjeeseen tulevista kuvista suurin osa on kuitenkin toteutettu elementtikohtaisesti, joten tässä projektissa oli kätevintä käyttää Firefoxia, sillä sen avulla saatiin kuvat otettua mistä tahansa kohdasta sivua, vaikka elementti ei olisikaan sijainnut sivun näkyvässä osassa.

Projektin aikana tuli myös eteen sivun rakenteesta johtuvia kuvakaappausten ottamista hankaloittavia tekijöitä. Läheskään kaikille elementeille, joista kuvia otin, ei oltu ollenkaan määritelty yksilöllistä ID:tä. Tällöin niiden paikallistaminen saattaa olla huomattavasti hankalampaa, sillä ensin täytyy paikallistaa lähin sel-

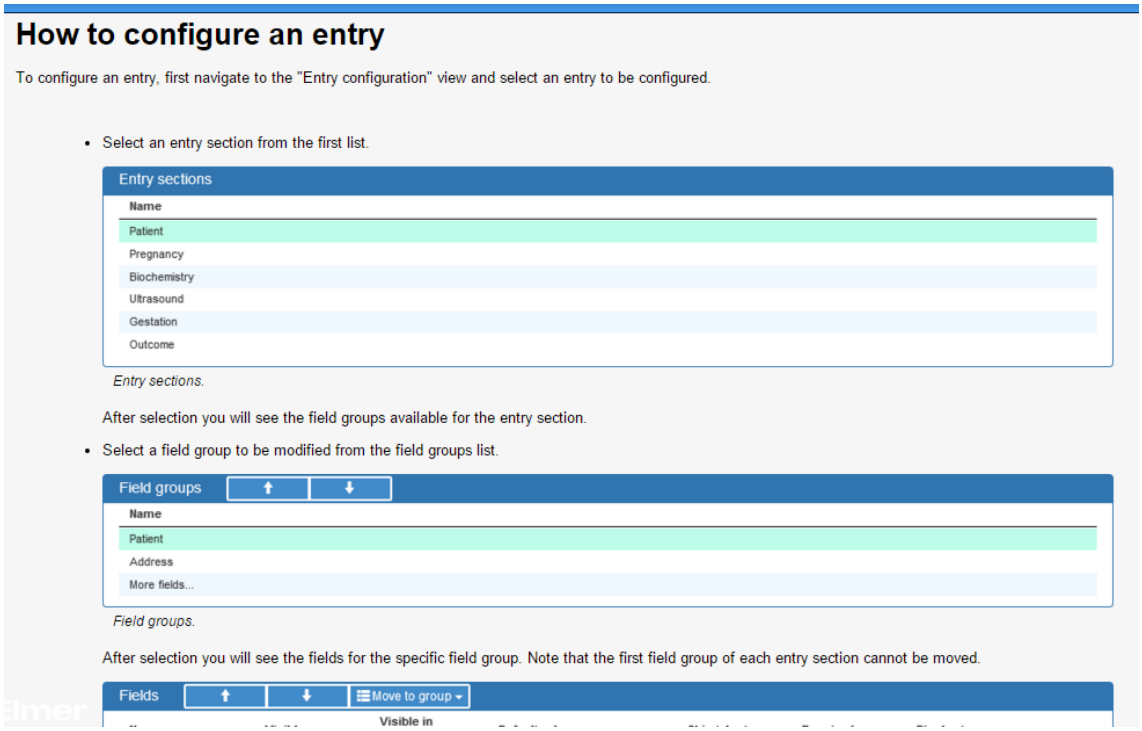
lainen elementti, jonka pystyy yksilöimään. Sen jälkeen halutun elementin voi paikallistaa käyttäen parent-element tai sibling-element -menetelmää, tai vaihtoehtoisesti jälkikäteen lisätä elementille ID. Olisi siis tärkeää, että sivun HTML-rakenneta koodatessa olisi jo pääpiirteittäin selvillä, että mistä elementeistä on tarkoitus ottaa kuvia.

Selenium joka tapauksessa toimi kuvakaappausten ottamisessa halutulla tavalla ja oli onnistunut valinta tähän tarkoitukseen. Kuvassa 9 esimerkkinä on näkymä ohjelmiston Entry Configuration -sivusta.



Kuva 9. Entry Configuration sivun yleisnäkymä

Sivu koostuu yläpalkista, sivupalkista sekä itse sisällöstä. Ohjeeseen päätyi tästä näkymästä kuvia kuvan 10 mukaisesti.



Kuva 10. Ote Entry Configuration sivun ohjeesta

6 YHTEENVETO

Selenium WebDriverin käyttö automatisoituihin kuvakaappauksiin tuotti paljon etuja, mutta siinä oli myös muutamat heikkoutensa, jotka piti ottaa huomioon. Suurin syy Seleniumin valitsemiseen tähän projektiin oli, että se säästää pidemmän päälle hyvin paljon aikaa. Syksyllä 2015 LifeCyclesta oli oma versio kahdeksalle eri kielelle, ja jatkossa kieliversioita on mahdollisesti tulossa lisää. Mikäli kuvakaappaukset otettaisiin ohjelmiston help-tiedostoon manuaalisesti, menisi työhön todella paljon aikaa, koska kuvat tulee luonnollisesti ottaa jokaiselta kieleltä erikseen. Yhteensä kuvia ohjeeseen tuli noin 90 jokaisesta kielestä, eli yhteensä yli 700 kuvaa. Lisäksi mikäli sivuston käyttöliittymään tulee muutoksia, pitää myös kuva päivittää ohjeeseen aina jokaiselle kielelle erikseen. Toinen suuri etu mikä automatisoidusti tehdyillä kuvakaappauksilla saavutetaan, on se että help-tiedosto pysyy aina ajan tasalla. Joka kerta kun ohjelmistosta tehdään päivitys, päivittyy myös kuvat ohjeeseen.

Projekti onnistui kokonaisuudessaan hyvin ja opinnäytetyön kirjoittamisen aikana ohjelmisto oli edennyt testausvaiheeseen. Ohjeeseen oli tarkoitus tulla mahdollisimman monipuolisesti kuvia helpottamaan käyttäjää ja tässä tavoitteessa onnistuttiin. Seleniumilla pystyi ottamaan kuvia monipuolisesta ja eri tilanteista ohjeeseen. Lisäksi Seleniumin valintaa tukee myös se, että välillä oli tarvetta tehdä alustavia toimenpiteitä käyttöliittymässä ennen kuvan ottamista. Koska Selenium on työkalu, jolla web-sovelluksia automatisoidaan, oli koodiin helposti mahdollista ohjelmoida tällaiset tilanteet. Esimerkkinä tästä on mm. tilanne jossa halutaan ottaa ohjeeseen kuva potilashaun muokkaamisesta. Ennen haun muokkaamista täytyy käyttöliittymässä ensin kopioida oletushaku uudeksi hauksi uudella nimellä, sillä oletushakuja ei pysty muokkaamaan. Vasta tämän jälkeen oli mahdollista päästä muokkaaman hakua ja ottaa kuva tilanteesta.

Elementtikohtaisten kuvien ottaminen onnistui Seleniumilla kuten ennalta oli tarkoitettu. Kuitenkin muutaman kerran tuli eteen tilanteita, joissa kuvia ei ollut mahdollista ottaa halutusta elementistä. Tämä ongelma tuli vastaan sellaisten alasvetovalikoiden kohdalla, jotka olivat toteutettu select-tagia käyttäen. Näistä

elementeistä ei pystynyt ottamaan kuvaa Seleniumilla, sillä jostain syystä ennen kuvan ottamista kohdistus siirtyi pois valikosta ja valikko ehti sulkeutua ennen kuvakaappausmetodin kutsua. Tämä ei kuitenkaan muodostunut isoksi ongelmaksi ohjeen toteuttamisen kannalta.

Kokonaisuutena voi sanoa, että Selenium ajoi tarkoituksensa projektissa, ja mikä tärkeintä jatkossa kuvien päivittäminen ohjeeseen ei vaadi juuri lainkaan työtä vaan käyttöliittymän päivittyessä myös kuvat päivitetään pelkästään testimetodi suorittamalla. Lisäksi kuvakaappaukset ovat oletuksena riittävän pienen kokoisia, mutta kuitenkin tarpeeksi tarkkoja, joten niitä ei ole tarvetta käsitellä ennen kopioimista ohjeeseen.

LÄHTEET

Acutenix 2015. Web Applications: What are they? What of them? Viitattu 24.11.2015. http://www.acunetix.com/websitesecurity/web-applications/

Appium 2015. Introduction. Viitattu 25.8.2015. http://appium.io/introduction.html?lang=fi

Burns, D. 2012. Selenium 2 Testing Tools: Beginners Guide. Birmingham: Packt Publishing Ltd.

Gojare, R.; Rahul, J & Gaigaware, D. 2015. Analysis and Design of Selenium WebDriver Automation Testing Framework. Procedia Computer Science 50/2015, 341-346.

Ios-driver 2015. Ios-driver. Viitattu 27.8.2015. https://ios-driver.github.io/ios-driver/?page=home

Li, Y.; Das, P. & Dowe, D. 2014. Two dacades of Web application testing – A survey of recent advances. Information Systems 43/2014, 20-54.

Nations, D. 2015. Web Applications: What is a Web Application? Viitattu 24.11.2015. http://webtrends.about.com/od/webapplications/a/web_application.htm

Newcircle 2013. Selenium Tutorial: Locators. Viitattu 15.8.2015. https://newcircle.com/bookshelf/selenium_tutorial/locators

Selendroid 2015a. Selendroid. Viitattu 27.8.2015. http://selendroid.io/

Selendroid 2015b. Getting started with Selendroid. Viitattu 27.8.2015. http://selendroid.io/

Selenium Python Bindigs 2014. Docs: Locating Elements. Viitattu 15.8.2015. http://selenium-python.readthedocs.org/en/latest/locating-elements.html#

SeleniumHQ 2015a. Downloads. Viitattu 15.8.2015. http://www.seleniumhq.org/download/

SeleniumHQ 2015b. Documentation: Selenium-IDE. Viitattu 15.8.2015. http://www.seleniumhq.org/docs/02_selenium_ide.jsp

SeleniumHQ 2015c. Documentation: Selenium WebDriver. Viitattu 15.8.2015. http://www.seleniumhq.org/docs/03_webdriver.jsp

Stewart, S. 2010. Selenium WebDriver. The Architecture of Open Source Applications. Viitattu 1.8.2015. http://www.aosabook.org/en/selenium.html

TechTerms 2015. SDK. Viitattu 4.12.2015. http://techterms.com/definition/sdk

Tino, A. 2014. How to set up a basic working Appium test environment. Viitattu 25.8.2015.http://blogs.technet.com/b/antino/archive/2014/09/22/how-to-set-up-a-basic-working-appium-test-environment.aspx

How does the internet work 2015. W3C. Viitattu 24.11.2015. http://www.w3.org/wiki/How_does_the_Internet_work

What are APIs? 2015. W3C. Viitattu 26.11.2015. http://www.w3.org/2008/webapps/

W3schools 2015a. HTML Introduction. Viitattu 26.11.2015.
http://www.w3schools.com/html/html_intro.asp

W3schools 2015b. CSS Introduction. Viitattu26.11.2015.
http://www.w3schools.com/css/css_intro.asp