Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Domen Gaber

Primerjava orodij za avtomatizacijo testiranja uporabniških vmesnikov

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Razvoj sodobnih spletnih in mobilnih aplikacij zahteva učinkovito testiranje. V diplomskem delu predstavite tri sodobna orodja za testiranje spletnih in mobilnih aplikacij Selenium, Appium in Eggplant, ki omogočajo učinkovito avtomatizacijo testiranja uporabniških vmesnikov aplikacij. Vsa orodja podrobno predstavite ter prikažite njihovo uporabo pri izdelavi avtomatiziranih testov za izbrani testni primer. Nalogo zaključite s primerjavo orodij po hitrosti in zanesljivosti.

Zahvaljujem se mentorju, viš. pred. dr. Igorju Rožancu, za mentorstvo, nasvete in odlično komunikacijo.
Posebno zahvalo namenjam domačim, ki so mi omogočili študij in mi stali ob strani v zahtevnih ter manj zahtevnih trenutkih.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
DOM	Document Object Model	objektni model dokumenta
HTML	Hypertext Markup Language	označevalni jezik za oblikovanje večpredstavnostnih dokumentov
CSS	Cascading Style Sheets	prekrivni slogi
OCR	Optical Character Recognition	optično prepoznavanje znakov
TIG	Text-Image Generator	generatorji besedilo-slika
AJAX	Asynchronous JavaScript + XML	asinhroni JavaScript in XML
SDLC	Software Development Life Cycle	življenski cikel razvoja programske opreme

Povzetek

Naslov: Primerjava orodij za avtomatizacijo testiranja uporabniških vmesnikov

Avtor: Domen Gaber

V diplomskem delu je predstavljeno delovanje in uporaba modernih orodij za avtomatizacijo testiranja uporabniških vmesnikov. Namen dela je primerjati izbrana orodja za avtomatizacijo testiranja in olajšati odločitev pri izboru orodja za avtomatizacijo testiranja.

Glavni cilj avtomatizacije je zmanjšanje stroškov in hitrejše izvajanje testiranja. Na trgu je danes na voljo več orodij za avtomatizacijo testiranja, ki so si med seboj različna po zahtevnosti uporabe, načinu delovanja in podpori različnih okolij za izvajanje testov. V okviru diplomskega dela si izberemo tri orodja za avtomatizacijo testiranja, jih opišemo, uporabimo in analiziramo njihovo delovanje. Izbrana orodja so Selenium, Appium in Eggplant.

Uporabo orodij prikažemo z izdelavo testnih skript na podlagi podanega testnega primera. Z izvajanjem izdelanih testov orodja med seboj primerjamo po hitrosti izvajanja in zanesljivosti delovanja. Pridobljene rezulate nato opišemo in analiziramo. Iz rezultatov ugotovimo večje razlike med hitrostjo in zanesljivostjo primerjanih orodij, navedemo vzroke in podamo predloge za izboljšavo primerjave.

Ključne besede: avtomatizacija testiranja, Selenium, Appium, Eggplant.

Abstract

Title: Comparison of Automated Graphical User Interface Testing Tools

Author: Domen Gaber

The thesis presents the analysis of modern tools for automated testing of various web based user interfaces. The purpose of the work is to compare specific test automation solutions and point out the most suitable test automation tool amongst them.

One of the main goals of test automation is to gain faster execution when compared to manual testing and overall cost deduction. There are multiple test automation solutions available on the market, which differ in complexity of use, type of operation and environment support.

This thesis concentrates on three tools intended for test automation: Selenium, Appium and Eggplant. It describes their main characteristics, evaluates their overall performance and analyses their effectiveness.

Every test automation solution is presented through a set of automated test scripts executed in a specific environment under test. Automated test scripts are compared in speed and reliability of execution first. Then, a more detailed analysis follows.

The analysis results are significantly different in both speed and reliability for different test automation solutions and in thesis we point out some of the reasons behind them. Finally, suggestions on how to improve the overall comparison process are made.

Keywords: test automation, Selenium, Appium, Eggplant.

Poglavje 1

Uvod

Testiranje programske opreme je eden izmed ključnih faz razvoja programske opreme. Proces testiranja je del postopka zagotavljanja kakovosti, ki jo dosežemo z zaznavanjem in odpravljanjem napak v programski opremi. Zaradi hitrejšega razvoja in uporabe novih metodologij se poleg ročnega testiranja pojavlja vse večja potreba po avtomatizaciji testiranja [1].

Avtomatizacija testiranja izboljšuje nekatere slabosti, ki so prisotne pri ročnem testiranju programske opreme. Glavni značilnosti avtomatizacije testiranja sta povečana hitrost in nižja cena izvajanja testov. Cena in hitrost izvajanja testov sta pomembni predvsem pri razvoju programske opreme, ki je deležna pogostih popravkov ali nadgradenj [1]. Vse več programske opreme se namreč razvija z uporabo novih pristopov, ki določajo iterativen in inkrementalen razvoj. Primer takega pristopa je agilni razvoj programske opreme [2], ki s pogostimi iteracijami omogoča boljšo prilagodljivost na spremembe v zahtevah naročnika. Pred izdajo popravkov in nadgradenj želimo z izvajanjem testiranja zagotoviti, da spremembe ne vplivajo na delovanje obstoječih funkcionalnosti.

Hitrost izvajanja orodij za avtomatizacijo je višja od ročnega testiranja, kar nam omogoča, da v istem času izvedemo več testov, kot bi jih izvedli z ročnim testiranjem. Izvajanje avtomatiziranih testov poteka nenadzorovano, s čimer zmanjšamo stroške izvajanja. Zaradi manjšega časa in cenejšega

izvajanja lahko z avtomatizacijo povečamo pokritost testov, ki določa število testiranih funkcionalnosti v postopku testiranja [1].

Za avtomatizacijo testiranja imamo na voljo več orodij, ki se razlikujejo po delovanju in funkcionalnostih, ki jih ponujajo. Pred izbiro orodja za avtomatizacijo je potrebno poznati prednosti, slabosti in omejitve različnih orodij. Nepremišljena odločitev pri izbiri orodja lahko pomeni, da bomo v določenem trenutku ugotovili, da je orodje neprimerno za našo uporabo. Orodje bomo morali zamenjati, kar pomeni izgubo časa in sredstev, ki smo jih vložili v orodje za avtomatizacijo in izdelavo testnih skript [1].

Cilj diplomskega dela je olajšati odločitev pri izboru orodja in prihraniti čas pri uporabi izbranega orodja za avtomatizacijo testiranja. Cilj bomo dosegli z opisom in primerjavo izbranih orodij. Za primerjavo bomo v izbranih orodjih izdelali teste, s pomočjo katerih bomo primerjali hitrost izvajanja in zanesljivost delovanja izbranih orodij.

Diplomsko delo sestavljajo naslednja poglavja:

- **Uvod**, kjer se bomo seznanili s problemom in ciljem, ki ga želimo z izdelavo diplomskega dela rešiti.
- **Pregled področja**, kjer bomo predstavili testiranje programske opreme. Izpostavili bomo motivacijo za avtomatizacijo testiranja ter omenili njene prednosti in slabosti.
- **Orodja za avtomatizacijo testiranja**, kjer bomo izbrali in podrobno opisali izbrana orodja za avtomatizacijo.
- **Primerjava orodij**, kjer bomo za predstavljen testni primer v izbranih orodjih izdelali teste. Orodja bomo nato s pomočjo izdelanih testov primerjali po hitrosti in zanesljivosti izvajanja testov.
- **Sklepne ugotovitve**, kjer bomo strnili rezultate diplomskega dela, podali smiseln zaključek in predloge za nadgradnjo dela.

Poglavje 2

Pregled področja

V pregledu področja bomo opisali namen in postopek testiranja programske opreme in še zlasti razložili postopek izvajanja funkcionalnih testov. Opisali bomo agilni in testno voden razvoj programske opreme ter poiskali povezavo med agilno metodologijo in avtomatizacijo testiranja. Kasneje se bomo osredotočili na avtomatizacijo testiranja, kjer bomo razložili kdaj nastane potreba po avtomatizaciji in izpostavili prednosti ter slabosti avtomatizacije.

2.1 Testiranje programske opreme

Testiranje programske opreme je ključen del življenjskega cikla razvoja programske opreme (angl. Systems Development Life Cycle) [3]. Z izvajanjem testiranja želimo povečati kakovost in zanesljivost programa [4].

Pogosta napaka pri razumevanju testiranja je, da je testiranje postopek, kjer želimo pokazati, da v programu ni napak in s tem potrditi pravilno delovanje programa. Ker kljub testiranju ne moremo zagotoviti, da v programu ni napak, testiranje raje začnemo s predpostavko, da program vsebuje napake, ki jih bomo z izvajanjem testiranja poizkušali poiskati. Tako lahko testiranje definiramo kot: „*Testiranje programske opreme je izvajanje programa z namenom iskanja napak*“ [4].

V popolnem svetu bi s testiranjem želeli doseči popolno pokritost (angl.

coverage) programa, kar pa pogosto ni izvedljivo, saj ima lahko že na videz preprost program veliko število vhodnih in izhodnih kombinacij. Zato je za uspešno testiranje pomembno, da ima izvajalec testov primeren odnos ali vizijo, ki je v določenih primerih celo bolj pomembna kot sam proces tesiranja. Zaželjeno je, da je izvajalec testov neodvisna oseba, ki ni sodelovala pri razvoju programa [4].

Ena od možnih delitev testiranja je delitev na več nivojev [4]:

- **testiranje enot (angl. unit testing)**, kjer testiramo posamezne dele programske kode.
- **testiranje integracije (angl. integration testing)**, kjer testiramo medsebojno delovanje enot in komponent.
- **funkcionalno testiranje (angl. functional testing)**, kjer testiramo funkcionalnosti izdelka na podlagi zunanje specifikacije (angl. external specification) funkcionalnih zahtev.
- **testiranje sistema (angl. system testing)**, kjer testiramo nefunkcionalne zahteve sisteme.
- **sprejemno testiranje (angl. acceptance testing)**, kjer naročnik ali uporabniki preverijo, če izdelek ustreza njihovim pričakovanjem.
- **testiranje namestitve (angl. installation testing)**, kjer preverimo, če nameščen programski izdelek še vedno deluje ustrezno.

2.2 Funkcionalno testiranje

Funkcionalno testiranje (angl. Functional testing) je proces v postopku testiranja, kjer poizkušamo poiskati neskladnosti med programom in zunanjo specifikacijo (angl. external specification) funkcionalnih zahtev. Zunanja specifikacija natančno opisuje obnašanje programa z vidika uporabnika. Za izvedbo funkcionalnega testa je potrebno pregledati specifikacijo, na podlagi katere določimo množico testnih primerov [4].

Funkcionalno testiranje se ponavadi izvaja z uporabo pristopa črne škatle (angl. black-box), ki je eden izmed treh pristopov k testiranju [5]:

- **Princip bele škatle (angl. white box)** zahteva, da za testiranje poznamo celotno izvorno kodo in notranje delovanje programa, saj testiranje izvajamo na podlagi logike in strukture izvorne kode.
- **Princip črne škatle (angl. black box)** je tehnika testiranja, kjer testiranje ne izvajamo na podlagi notranje zgradbe in delovanja programa, ampak poznamo namen temeljnih vidikov sistema, za katere pripravimo vhode in preverjamo pravilnost izhodov.
- **Princip sive škatle (angl. gray box)** združuje tehniki bele in črne škatle. Testiranje izvajamo z omejenim obsegom znanja notranjega delovanja (bela škatla) in poznavanjem temeljnih vidikov sistema (črna škatla).

2.3 Agilni razvoj programske opreme

Agilna metodologija (angl. Agile Methodology) je metoda razvoja programske opreme, ki temelji na iterativnem in inkrementalnem razvoju. Agilna metodologija spodbuja k prilagodljivem načrtovanju in nadgrajevanju razvoja [1]. Vrednote agilnega razvoja so [2]:

- **Posamezniki in interakcije pred procesi in orodji**: agilni razvoj spodbuja samostojno organiziranje, motivirane posameznike in interakcijo med sodelavci.
- **Delujoča programska oprema pred vseobsežno dokumentacijo**: delujoča in razumljiva programska oprema je bolj uporabna kot obsežna dokumentacija.
- **Sodelovanje s stranko pred pogodbenimi pogajanji**: sodelovanje s stranko med razvojem programskega izdelka je pomembno za dodajanje zahtev, ki niso bile določene ob začetku projekta.

- **Odziv na spremembe pred togim sledenjem načrtom**: hitri odzivi na spremembe z namenom zmanjšanja časa potrebnega za dostavo spremembe stranki.

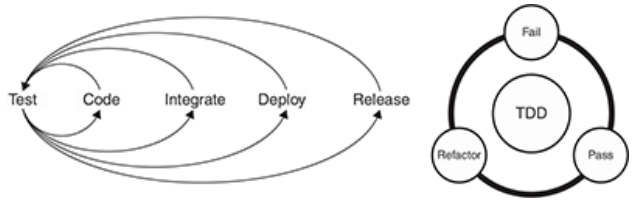
Agilni razvoj programske opreme je sestavljen iz množice metodologij in praks, ki strmijo k hitremu razvoju programske opreme. Najpogostejše metodologije uporabljene v agilnem razvoju so [1]:

- Scrum [6]
- Ekstremno programiranje (angl. Extreme Programming) [7]
- Testno voden razvoj (angl. Test Driven Development) [8]
- Funkcionalno voden razvoj (angl. Feature Driven Development) [9]

Izmed naštetih metodologij se bomo osredotočili na testno voden razvoj, ki predstavlja agilno različico izvedbe testiranja.

2.3.1 Testno voden razvoj (TDD)

Testno voden razvoj programske opreme je agilna praksa, ki zagovarja izdelavo testov pred začetkom kodiranja. Izdelani testi določajo specifikacijo za izdelavo programske kode. Lastnost testno vodenega razvoja so zelo kratki cikli razvoja, kjer vsakemu ciklu sledi postopek testiranja. Na sliki 2.1 je prikazano izvajanje testov po vsaki fazi razvoja.



Slika 2.1: Testno voden razvoj (TDD)[1].

V kolikor med testiranjem posameznega cikla ugotovimo napake, se razvita koda zavrne in pošlje v popravke. Zaradi kratkih razvojnih ciklov se

testno voden razvoj opira na avtomatizirane teste, ki omogočajo hitro izvajanje testov [1].

2.4 Testna avtomatizacija

Testna avtomatizacija je postopek razvoja in zaganjanja testnih skript. V življenjskem ciklu razvoja programske opreme [3] želimo zmanjšati čas za dostavo programskega izdelka, hkrati pa vzdrževati visoko kakovost programskega izdelka. Testna avtomatizacija zato lahko igra pomembno vlogo v SDLC, saj pospeši proces testiranja, hkrati pa s povečanjem pokritosti s testi zagotovimo, da ob nadgradnjah ali popravljanju napak ne pokvarimo prej delujočih funkcionalnosti programskega izdelka. Izdelane testne skripte se izvajajo nenadzarovano s pomočjo primerjave dejanskih rezultatov s pričakovanimi rezultati [1].

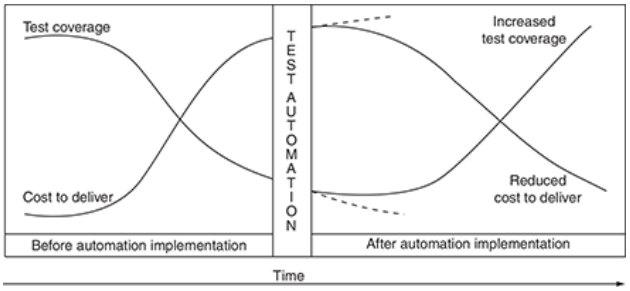
2.4.1 Cilji avtomatizacije

Z avtomatizacijo testiranja želimo doseči [1]:

- **Boljše testiranje**: povečanje pokritosti s testi in povečanje zanesljivosti testiranja z odpravo možnosti človeških napak, ki se lahko pojavijo pri ročnem testiranju.
- **Pospešitev testiranja**: hitrost izvajanja testov v orodjih za avtomatizacijo je višja od ročnega izvajanja.
- **Cenejše testiranje**: testna avtomatizacija pomaga pri obsežnejšem testiranju aplikacije, ki bi ga drugače ročno izvajali testerji, s čimer zmanjšamo stroške izvajanja testiranja.

V začetnih fazah projekta je zaradi manjšega obsega testiranih funkcionalnosti pokritost s testi visoka, stroški za ročno izvajanje testov pa nizki. Potreba po avtomatizaciji testiranja se pokaže z razširjanjem in povečevanjem zahtevnosti aplikacije. Sčasoma se zaradi pomanjkanja časa osredotočimo

na testiranje novih funkcionalnosti in počasi zmanjšujemo obseg testiranja obstoječih funkcionalnosti, s čimer se zmanjšuje pokritost s testi [1]. Z avtomatizacijo testiranja lahko dosežemo boljšo pokritost testov, saj izdelane teste lahko izvajamo brez dodatnih stroškov. Problem zmanjševanja pokritosti pred in po avtomatizaciji testiranja je prikazan na sliki 2.2.



Slika 2.2: Pokritost s testi pred in po avtomatizaciji [1].

2.4.2 Prednosti in slabosti avtomatizacije

Glavne prednosti avtomatizacije testiranja so [1]:

- **Zanesljivost**: ob vsakem izvajanju testov se izvedejo iste operacije, s čimer izključimo možnost človeške napake.
- **Ponovna uporaba**: izdelane teste lahko uporabimo za testiranje aplikacije v različnih okoljih.
- **Hitrost**: orodja za avtomatizacijo izvajajo teste hitreje kot človek.
- **Izčrpnost**: testi lahko pokrivajo vse funkcionalnosti aplikacije.
- **Zmanjšanje stroškov**: testi se lahko izvajajo brez prisotnosti človeka, s čimer zmanjšamo človeški trud in stroške.

Testna avtomatizacija ne prinaša samo prednosti, ampak se je potrebno zavedati tudi slabosti avtomatizacije. Glavne slabosti avtomatizacije so [1]:

- **Začetni stroški**: izdelava avtomatiziranih testov zahteva veliko začetno investicijo v orodje za avtomatizacijo in razvoj ogrodja.
- **Stroški vzdrževanja**: izdelane skripte moramo zaradi sprememb v aplikaciji redno posodabljati.
- **Pomanjkanje testiranja uporabnosti**: orodja za avtomatizacijo ne moremo uporabiti za ocenjevanje in analizo uporabnosti uporabniškega vmesnika aplikacije.
- **Ni takojšnje povrnitve vložka**: stroški avtomatizacije se povrnejo šele po določenem številu testnih izvajanj.

2.4.3 Smiselnost avtomatizacije

Zmotno je misliti, da je smiselno avtomatizirati vse testne scenarije. Za vsak testni scenarij se je potrebno odločiti med avtomatizacijo in ročnim izvajanjem testov. Omenili bomo nekaj kriterijev, s pomočjo katerih lahko sprejememo odločitev o avtomatizaciji testnega scenarija [1]:

- **Kritičnost za posel**: testni scenarij testira ključne funkcionalnosti aplikacije. Nedelovanje testiranih funkcionalnosti bi povzročilo veliko finančno izgubo za organizacijo.
- **Ponovljivost**: testni scenarij se pogosto izvaja.
- **Ponovna uporaba**: testni scenarij je potrebno izvesti z veliko množico različnih vhodnih podatkov.
- **Robustnost**: testni scenarij je nagnjen k človeškim napakam.
- **Čas izvajanja**: testni scenarij za izvajanje vzame veliko časa.

Testni scenariji, ki so primerni za avtomatizacijo ustrezajo enemu ali več kriterijem. Zgoraj naštete kriterije lahko uporabimo za določitev razporeda

avtomatizacije testnih scenarijev, kjer najprej avtomatiziramo testne scenarije, ki ustrezajo več kriterijem. S tem se najprej osredotočimo na avtomatizacijo testnih scenarijev, kjer je avtomatizacija najbolj smiselna [1].

Poglavje 3

Orodja za avtomatizacijo testiranja uporabniških vmesnikov

Za avtomatizacijo testiranja uporabniških vmesnikov imamo na voljo več različnih orodij. Ker je orodij veliko, so na spodnjem seznamu našteta le nekatera, ki se pogosto uporabljajo za testiranje uporabniških vmesnikov [10]:

- Selenium [11]
- Appium [12]
- Eggplant [13]
- Unified Functional Testing (UFT) [14]
- Watir [15]
- Rational Functional Tester [16]
- TestComplete [17]
- Tricentis Tosca [18]

- Ranorex Studio [19]
- Robot Framework [20]

Vseh naštetih orodij je preveč za neposredno primerjavo, zato se bomo osredotočili na tri, ki so med pogosteje uporabljenimi:

- Selenium [11]
- Appium [12]
- Eggplant [13]

Osredotočili smo se na orodja, ki podpirajo avtomatizacijo različnih platform in omogočajo avtomatizacijo aplikacij brez poseganja v programsko kodo. Pri izbiri orodij smo bili pozorni tudi na pristop, ki ga orodja uporabljajo za avtomatizacijo testiranja.

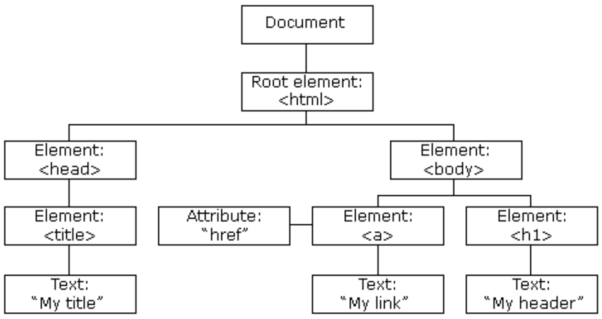
3.1 Selenium

Selenium [11] je odprtokodno orodje namenjeno avtomatizaciji testiranja spletnih brskalnikov na namiznih računalnikih. Razvoj orodja Selenium se je začel leta 2004, ko je Jason Huggins izdelal Javascript knjižnico za avtomatizacijo spletnih brskalnikov. Izvorna koda orodja je dostopna na platformi GitHub [21], kjer lahko spremljamo in sodelujemo pri razvoju orodja.

Delovanje orodja temelji na pregledovanju objektnega modela dokumenta (slika 3.1) in interakciji s posameznimi HTML elementi v objektnem modelu dokumenta. Orodje s spletnim brskalnikom komunicira preko programskega vmesnika WebDriver [22], ki zagotavlja enoten programski vmesnik preko katerega spletni brskalniki sprejmajo ukaze in jih nato s pomočjo lastne podpore za avtomatizacijo tudi izvedejo [23].

3.1.1 Orodja

Orodje Selenium je sestavljeno iz več programskih orodij, ki nam omogočajo različne pristope pri avtomatizaciji testiranja:



Slika 3.1: Primer objektnega modela dokumenta (DOM) [24].

- Selenium 1 ali Selenium RC [25]
- Selenium 2 ali Selenium WebDriver [26]
- Selenium IDE [27]
- Selenium-Grid [28]

V diplomski nalogi se bomo osredotočili na orodje Selenium WebDriver, ki je zaradi podpore za WebDriver trenutno najprimernejše za izdelavo zahtevnejših avtomatiziranih testov.

3.1.1.1 Selenium 1 (Selenium RC)

Začetki razvoja orodja Selenium segajo v leto 2004, ko je Jason Huggins za potrebe testiranja interne aplikacije v podjetju ThoughtWorks razvil Javascript knjižnico, ki je omogočala izdelavo in zagon avtomatiziranih testov v različnih spletnih brskalnikih. Ko se je spletna stran v spletnem brskalniku naložila v celoti, je knjižnica poskrbela za vstavljanje Javascript funkcij v brskalnik. Vstavljene funkcije so se uporabljale za upravljanje spletnega brskalnika [29].

Knjižnica je postala osnova za orodja Selenium IDE [26] in Selenium RC (Selenium Remote Control) [25]. Selenium RC je postalo prvo orodje, ki je omogočalo upravljanje spletnih brskalnikov z uporabo programskega jezika

izbranega po lastni izbiri [29]. Razvoj orodja Selenium RC je bil opuščen z izdajo orodja Selenium 2.

3.1.1.2 Selenium 2 (Selenium WebDriver)

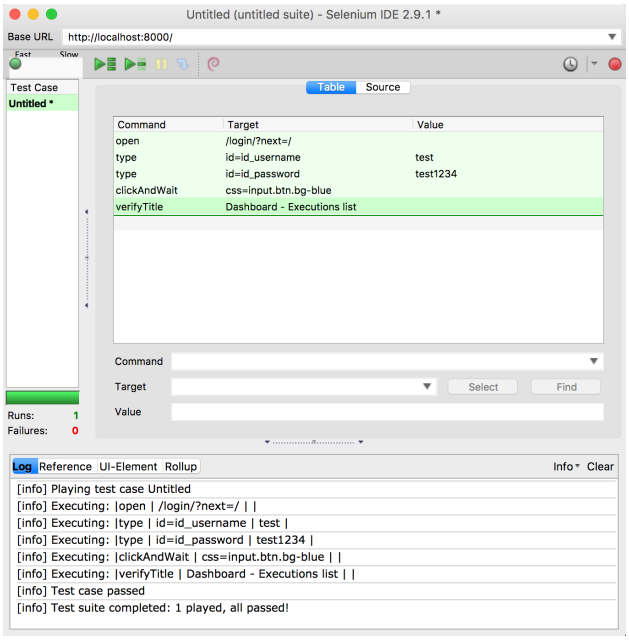
Brskalniki so sčasoma dodajali varnostne omejitve pri izvajanju Javascripta, zato so bile določene funkcionalnosti orodja Selenium RC onemogočene. Zaradi omenjenih omejitev je leta 2006 Google inženir Simon Stewart začel delo na projektu imenovanem WebDriver [22]. Namen projekta WebDriver je bil izdelava testnega orodja, ki omogoča neposredno komuniciranje s spletnim brskalnikom. Upravljanje spletnih brskalnikov poteka s pomočjo gonilnikov (angl. drivers), ki podane ukaze izvedejo neposredno v brskalniku [29].

Selenium in WebDriver projekta sta se združila leta 2008, ko je staro osnovo orodja Selenium RC zamenjal WebDriver, ki je odpravil pomanjkljivosti Javascript knjižnice. Tako je nastalo orodje Selenium 2 [26] imenovano tudi Selenium WebDriver. Selenium 2 ponuja infrastrukturo za WebDriver v različnih brskalnikih in na različnih operacijskih sistemih, medtem ko slednji zagotavlja komunikacijo in upravljanje s spletnim brskalnikom preko programskega vmesnika [29].

3.1.1.3 Selenium IDE

Selenium IDE [27] je vtičnik za spletni brskalnik Firefox, ki omogoča izdelavo avtomatiziranih testov s pomočjo grafičnega uporabniškega vmesnika (slika 3.2). Vtičnik lahko prenesemo iz Firefox spletne trgovine [30].

Selenium IDE je primeren za izdelavo preprostih skript in raziskovanje funkcionalnosti orodja Selenium. Uporabniški vmesnik nam omogoča preprosto izdelavo testne skripte s pomočjo snemanja oziroma zajemanja akcij na spletni strani. Zajete akcije se pretvorijo v ukaze, ki jih lahko po potrebi ročno spreminjamo ali dodajamo (slika 3.3). Izdelane skripte v uporabniškem vmesniku tudi izvajamo in razhroščujemo [23].

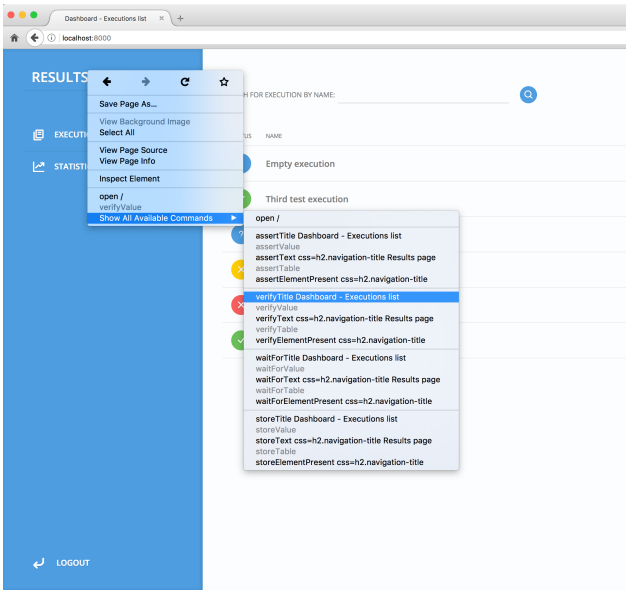


Slika 3.2: Uporabniški vmesnik Selenium IDE.

3.1.1.4 Selenium Grid

Selenium Grid [28] nam omogoča vzporedno izvajanje testov v različnih spletnih brskalnikih na različnih testnih napravah. Vzporedno izvajanje je smiselno, kadar je v naših testnih zahtevah določeno izvajanje testov v več različnih brskalnikih ali operacijskih sistemih, saj z vzporednostjo zmanjšamo skupen čas izvajanja testov [23].

Za primer vzemimo nabor 200 testnih primerov, ki se v posameznem brskalniku povprečno izvajajo 20 minut in jih je potrebno izvesti v treh različnih brskalnikih. Skupen čas zaporednega izvajanja celotnega nabora testov v vseh brskalnikih znaša 60 minut, saj se izvajanje testov v naslednjem brskalniku začne šele, ko se zaključi izvajanje testov v prejšnjem brskalniku. Pri vzporednem izvajanju se testi hkrati izvajajo v vseh brskalnikih, zato je skupen čas izvajanja 20 minut, oziroma enak času izvajanja testov v brskalniku, ki je za izvajanje potreboval največ časa. V našem primeru bi skupen čas izvajanja zmanjšali za ≈ 40 minut, kar predstavlja ≈ 200 % pospešitev.



Slika 3.3: Izbira ukaza na izbranem elementu v orodju Selenium IDE.

Selenium Grid naprave povezuje v omrežje, ki je sestavljeno iz enega zvezdišča (angl. hub) in vsaj enega vozlišča (angl. node). Vozlišče predstavlja testno napravo, kjer bo potekalo izvajanje testov. Vsako vozlišče mora ob prijavi na zvezdišče sporočiti brskalnik in operacijski sistem, ki ju bomo uporabljali na testni napravi. Ob izvajanju testov zvezdišče za vsak test na podlagi prejete informacije o zahtevanem brskalniku in operacijskem sistemu izbere prosto vozlišče, ki ustreza testnim zahtevam. Izbrano vozlišče od zvezdišča prejme ukaze, ki jih izvede v izbranem brskalniku [23].

Za delovanje zvezdišča in vozlišč je na vsako napravo potrebno namestiti *Selenium Server*, ki služi kot strežnik za komuniciranje znotraj omrežja. Ob zagonu strežnika *Selenium Server* napravi določimo vlogo, ki jo bo naprava imela v omrežju.

3.1.2 Programski jeziki

Ena izmed glavnih prednosti orodja Selenium je široka izbira programskih jezikov, ki jih lahko uporabimo za izdelavo testov. Različni programski jeziki

so podprti s pomočjo knjižnic, ki implementirajo Selenium API za komunikacijo z WebDriverjem v obliki metod in funkcij v posameznem programskem jeziku. Orodje s pomočjo knjižnic podpira 10 različnih programskih jezikov:

- C#
- Java
- Python
- Ruby
- JavaScript
- Haskell
- Objective-C
- Perl
- PHP

3.1.3 Podprti operacijski sistemi

Teste izdelane v orodju Selenium lahko izvajamo na operacijskih sistemih Microsoft Windows, Apple MacOS in Linux. Avtomatizacija nekaterih spletnih brskalnikov je možna samo na določenih operacijskih sistemih, saj jih ni možno namestiti na preostale operacijske sisteme (Safari na MacOS-u in Internet Explorer na Windows-ih).

3.1.4 Podprti brskalniki

Orodje Selenium podpira izvajanje testov v vseh najpogosteje uporabljenih spletnih brskalnikih. Za avtomatizacijo brskalnika je na testno napravo potrebno namestiti pripadajočo implementacijo WebDriverja, ki skrbi za sprejemanje ukazov preko programskega vmesnika (API) in izvajanje podanih ukazov znotraj brskalnika. Zaradi standardiziranega programskega vmesnika,

ki ga določa WebDriver, lahko izdelane teste izvajamo v vseh podprtih brskalnikih brez potrebe po spreminjanju obstoječe testne skripte. Podprti brskalniki:

- Chrome (s pomočjo ChromeDriverja)
- Firefox (s pomočjo FirefoxDriverja)
- Safari (s pomočjo SafariDriverja)
- Internet Explorer (s pomočjo InternetExplorerDriverja)
- Opera (s pomočjo OperaDriverja)

3.2 Appium

Orodje Appium [12] je odprtokodno orodje, ki je namenjeno avtomatizaciji mobilnih naprav z operacijskimi sistemi Android in iOS. Začetki razvoja orodja Appium segajo v leto 2011, ko je Dan Cuellar podprl avtomatizacijo naprav iOS. Svojo rešitev je leta 2012 predstavil na konferenci Selenium. Z razvojem orodja je nato nadaljevalo podjetje Sauce Labs [31].

Orodje za svoje delovanje uporablja Appium strežnik, ki uporablja enak pristop kot Selenium WebDriver. Arhitektura delovanja je zasnovana na principu odjemalec/strežnik. Appium strežnik je HTTP strežnik napisan v Node.js [32], ki implementira REST programski vmesnik za sprejem zahtevkov [33]. Odjemalec generira zahtevke s pomočjo knjižnic, ki ukaze pretvorijo v zahtevek in zahtevek pošljejo na strežnik. Strežnik prejete zahtevke obdela in izvede na različne načine, ki so odvisni od platforme, na kateri teče strežnik [34].

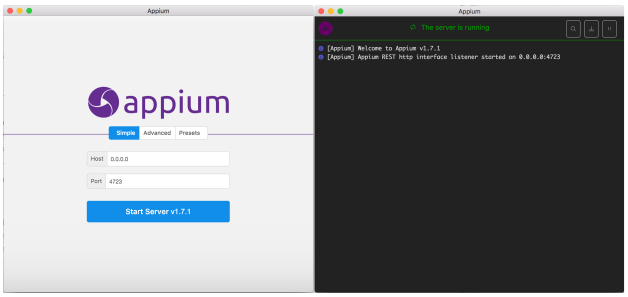
3.2.1 Orodja

Orodje Appium je sestavljeno iz več orodij, ki skupaj nudijo podporo za izdelavo in zagon avtomatiziranih testov na mobilnih testnih napravah.

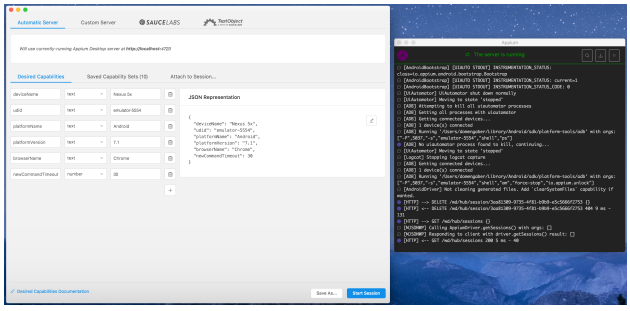
3.2.1.1 Appium Desktop

Appium Desktop [35] je aplikacija za namizne računalnike, ki omogoča enostavno zaganjanje Appium strežnika preko grafičnega uporabniškega vmesnika (slika 3.4). V uporabniškem vmesniku določimo IP naslov in vrata, na katerih želimo, da bo strežnik poslušal za zahtevke[34].

S pomočjo orodja Appium Desktop lahko preko uporabniškega vmesnika vzpostavimo povezavo s testno napravo in željenimi nastavitvami (slika 3.5).



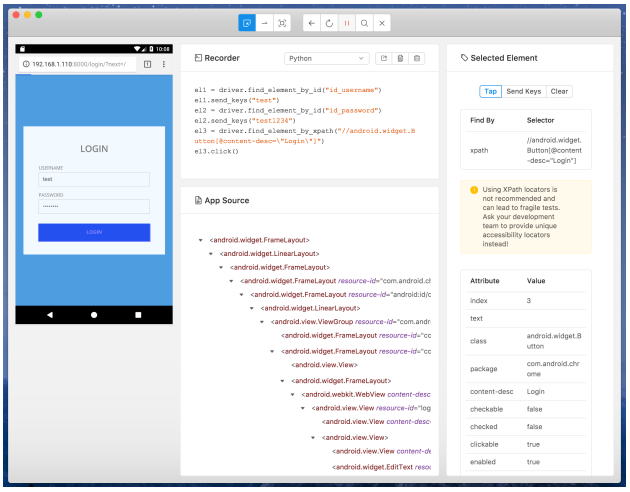
Slika 3.4: Grafična aplikacija za zagon Appium strežnika.



Slika 3.5: Vzpostavitev povezave s testno napravo s pomočjo orodja Appium Desktop.

Orodje Appium Desktop za pomoč pri izdelavi testov ponuja Appium Inspector, ki omogoča pregledovanje zgradbe uporabniškega vmesnika. S pomočjo pregledovalnika zgradbe grafičnega vmesnika aplikacij izberemo posamezne elemente grafičnega vmesnika in jim določimo akcijo, ki naj se na

izbranem elementu izvede. Orodje iz izbranih elementov in zaporedja akcij zgradi programsko kodo v izbranem programskem jeziku.



Slika 3.6: Pregledovalnik uporabniškega vmesnika.

3.2.1.2 Gonilnik XCUITest

Orodje Appium za avtomatizacijo uporabniškega vmesnika na napravah z operacijskim sistemom iOS uporablja knjižnico XCUITest [36]. Orodje dostopa do knjižnice XCUITest preko WebDriverAgent strežnika. WebDriverAgent je implementacija WebDriver strežnika, ki kot domorodna aplikacija (angl. native application) teče na testni napravi in z uporabo XCUITest knjižnice omogoča interakcijo s testno napravo. Za povezavo z WebDriverAgent strežnikom je v orodju Appium na voljo XCUITest gonilnik [37], ki skrbi za preusmerjanje ukazov na WebDriverAgent strežnik in dodaja dodatne funkcionalnosti, kot je recimo upravljanje s simulatorjem [38].

3.2.1.3 Gonilnik UiAutomator2

Interakcija z mobilnimi napravami, ki imajo nameščen operacijski sistem Android, poteka s pomočjo gonilnika UiAutomator [39]. Gonilnik za avtomatizacijo testnih naprav uporablja tehnologijo Ui Automator [40], ki jo je razvilo

podjetje Google. UI Automator ponuja programski vmesnik za gradnjo testov, ki izvajajo interakcijo s testno napravo [33].

3.2.2 Programski jeziki

Izdelava in zagon testov v orodju Appium je mogoča v vseh programskih jezikih, ki implementirajo odjemalca za povezavo z Appium strežnikom. Odjemalec v orodju Appium je razširitev Selenium odjemalca, ki dodaja nove funkcionalnosti namenjene mobilnim napravam [41]. Orodje Appium preko knjižnic implementira odjemalca v programskih jezikih:

- Ruby
- Python
- Java
- JavaScript
- Objective C
- PHP
- C#

3.2.3 Podprti operacijski sistemi

Teste izdelane v orodju Appium lahko izvajamo na testnih napravah z operacijskima sistemoma Android in iOS. Za izvajanje testov potrebujemo namizni računalnik, ki ima nameščenega enega izmed operacijskih sistemov Microsoft Windows, Apple MacOS ali Linux. Potrebno je omeniti, da avtomatizacija testnih naprav z operacijskim sistemom iOS zahteva orodje XCode, ki je na voljo samo na operacijskih sistemih MacOS.

3.2.4 Podprti brskalniki

Orodje Appium omogoča avtomatizacijo privzetih brskalnikov na operacijskih sistemih iOS in Android. To pomeni, da na operacijskem sistemu iOS lahko avtomatiziramo brskalnik Safari, brskalnik Google Chrome pa na operacijskem sistemu Android.

3.3 Eggplant Functional

Eggplant Functional je orodje za avtomatizacijo testiranja uporabniških vmesnikov, ki ga je izdelalo podjetje Redstone Software. Leta 2008 je podjetje TestPlant [42] prevzelo Redstone in nadaljevalo z razvojem orodja [43]. Orodje se uporablja za celoten proces testiranja, saj omogoča tako izdelavo in zagon testov kot tudi analizo ter shranjevanje rezultatov.

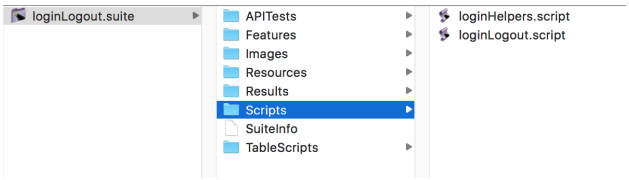
Posebnost orodja Eggplant Functional je njegovo delovanje oziroma pristop pri avtomatizaciji testiranja. Orodje se osredotoča na vidik uporabnika, saj testiranje temelji na stvareh, ki jih vidi uporabnik [44]. Ideja orodja je, da se za svoje delovanje ne zanaša in navezuje na programsko kodo. Prepoznavanje elementov uporabniškega vmesnika na zaslonu poteka s pomočjo iskanja in prepoznavanja vnaprej posnete referenčne slike ali besedila na zaslonu naprave. Zaradi neodvisnosti od programske kode lahko avtomatiziramo vse aplikacije, ki ponujajo uporabniški vmesnik.

Orodje Eggplant Functional s testno napravo komunicira preko povezave z VNC strežnikom z uporabo TCP/IP[45] protokola. Povezava VNC omogoča oddaljeno deljenje zaslona in pošiljanje ukazov iz tipkovnice in miške iz enega računalnika na drugega [46].

Vizualno prepoznavanje elementov ni edini način pristopa, saj je bila z različico `v18.0`[47] v orodje Eggplant Functional dodana podpora za WebDriver, ki omogoča enak pristop k avtomatizaciji kot pri orodju Selenium. Slabost je, da tako izdelane teste v orodju Eggplant lahko izvajamo samo na namiznih računalnikih, saj mobilne naprave trenutno še niso podprte. Na uporabo WebDriverja znotraj orodja Eggplant se zaradi podobnega delovanja

z orodjem Selenium ne bomo osredotočili.

Nabor testov (angl. suite), ki jih izdelamo v orodju Eggplant, je na pomnilniški napravi shranjen kot imenik s končnico `.suite`. Imenik vsebuje več podimenikov, ki hranijo testne skripte in referenčne slike. Primer imenika je prikazan na sliki 3.7.



Slika 3.7: Imeniška struktura nabora testov v orodju Eggplant na pomnilniški napravi.

3.3.1 Orodja

Za izdelavo in zaganjanje testov v orodju Eggplant Functional je na voljo več orodij, ki razširjajo funkcionalnosti.

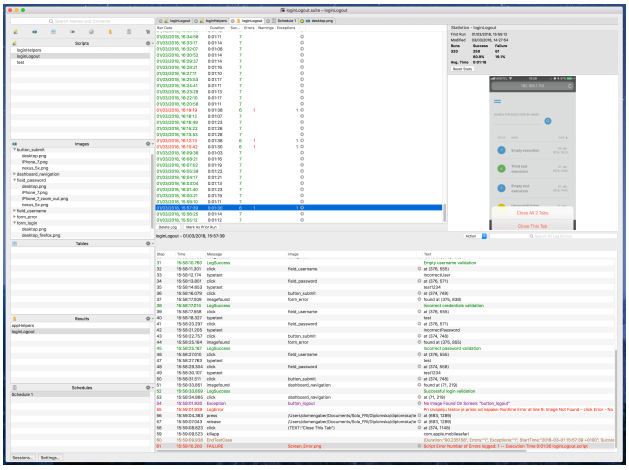
3.3.1.1 Eggplant Functional

Orodje Eggplant Functional ponuja razvojno okolje, ki omogoča izdelavo in zagon testov s pomočjo uporabniškega vmesnika. Za izdelavo testov v orodju Eggplant Functional uporabljamo dva okna uporabniškega vmesnika `Suite Window` in `Viewer Window`.

Suite Window

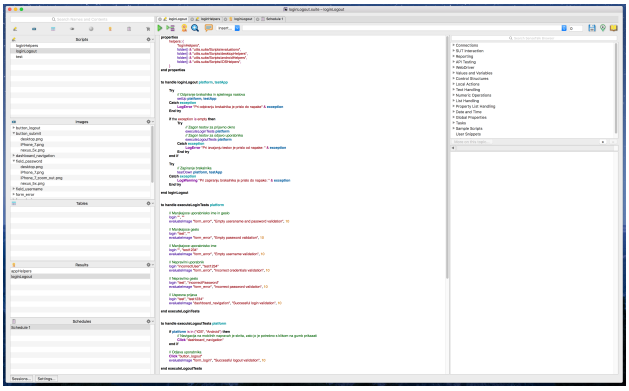
Okno `Suite Window` je glavno okno uporabniškega vmesnika orodja Eggplant Functional. V njem lahko izdelujemo, razhroščujemo (angl. debugging) in urejamo testne skripte ali slike [48]. Okno Suite Window omogoča zaganjanje in načrtovanje izvajanja testov ter pregledovanje rezultatov (slika 3.8).

Za urejanje in izdelavo testnih skript v jeziku *Sensetalk* [49] je na voljo urejevalnik kode (slika 3.9). Urejevalnik pri izdelavi testne skripte pomaga



Slika 3.8: Pregledovanje rezultatov v oknu Suite window.

s samodejnim dokončanjem ukazov in za boljšo berljivost ponuja barvanje sintakse programskega jezika *SenseTalk*.



Slika 3.9: Urejevalnik kode v oknu Suite window.

Okno `Suite Window` ponuja tudi urejevalnik slik, kjer lahko urejamo zajete referenčne slike elementov na testni napravi [50]. Vsaka referenčna slika zajeta v orodju Eggplant ima pripadajočo datoteko s končnico `.imageinfo`, ki hrani vse lastnosti slike, ki jih nastavimo v urejevalniku slik. Lastnosti referenčne slike, ki jih lahko spremenimo v urejevalniku (slika 3.10):

- način iskanja oziroma primerjave s sliko na zaslonu, kjer je na voljo 5

različnih iskalnih načinov:

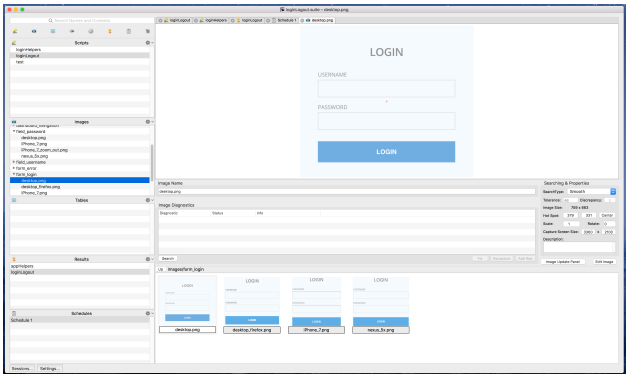
  - `precise` način zahteva, da sta referenčna slika in slika na zaslonu identični.
  - `tolerant` način zazna in dopušča manjše razlike v barvi posameznih točk, ki lahko nastanejo zaradi prosojnosti ali sprememb ozadja.
  - `smooth` način zazna in dopušča manjše spremembe pri izrisu besedila na zaslonu, ki so posledica mehčanja pisave (angl. text anti-aliasing).
  - `pulsing` način dopušča razliko v barvi točk, ki so posledica animacij na iskanem elementu.
  - `smooth and pulsing` način združuje načina Smooth in Pulsing.

- dopustnost razlik med referenčno sliko in sliko na zaslonu, ki je predstavljena s toleranco in določa, za največ koliko se lahko razlikuje barva primerjanih točk.
- spreminjanje pozicije točke za interakcijo (angl. hotspot).
- spreminjanje velikosti in zasuka slike.
- urejanje slike, kjer lahko iz slike izbrišemo posamezne točke, sliko obrežemo ali pa določimo barvo, ki jo želimo spregledati ali iskati. Urejevalnik slike je prikazan na sliki 3.11.

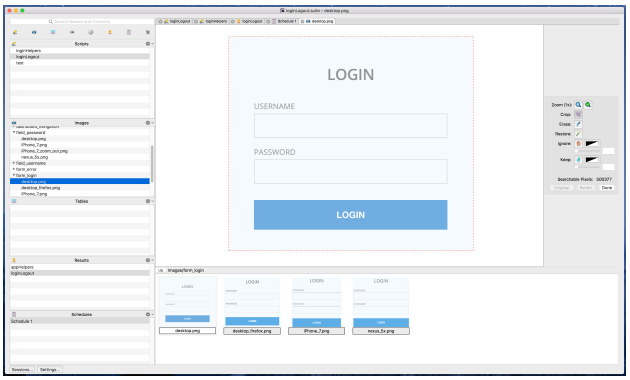
Viewer Window

Okno `Viewer Window` se odpre ob vzpostavitvi povezave s testno napravo in prikazuje zaslon povezane testne naprave. `Viewer Window` deluje v dveh načinih:

- *način v živo (angl. live mode)*: omogoča interakcijo s testno napravo. Interakcijo izvajamo s kliki ali tipkovnico.

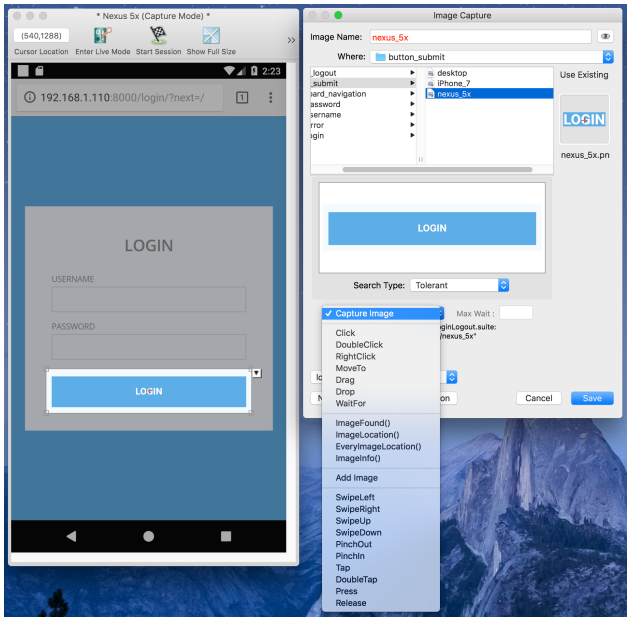


Slika 3.10: Urejevalnik lastnosti slike v oknu Suite window.



Slika 3.11: Urejevalnik slik v oknu Suite Window.

- *način zajema (angl. capture mode)*: omogoča zajem zaslonskih slik testne naprave. Za zajem slike označimo prostor na zaslonu, ki ga želimo zajeti, in kliknemo na gumb `Capture Image`. Odpre se nam dialog, kjer določimo lokacijo, kamor želimo zajeto sliko shraniti. V dialogu lahko izberemo tudi ukaz, s katerim želimo uporabiti posneto sliko. Izbran ukaz se bo samodejno vstavil v testno skripto. Z zaporedjem zajemanja slik in izbiro ukazov lahko izdelamo testno skripto brez pisanja programske kode. Zajem slike v oknu `Viewer Window` je prikazan na sliki 3.12.



Slika 3.12: Zajem referenčne slike elementa v oknu `Viewer Window`.

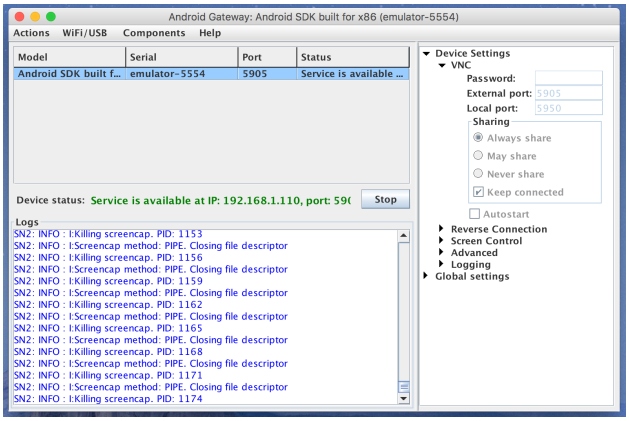
3.3.1.2 Android Gateway in iOS gateway

Android Gateway [51] je orodje, ki omogoča zagon VNC strežnika na mobilnih napravah z operacijskim sistemom Android. Testno napravo preko USB priključka povežemo z računalnikom, kjer je nameščen in zagnan Android Gateway. Povezana naprava se prikaže v tabeli naprav (slika 3.13). Napravi lahko poljubno spreminjamo nekatere nastavitve [52]. Ob kliku na gumb `Start` se na napravi zažene VNC strežnik in s tem je naprava pripravljena za avtomatizacijo.

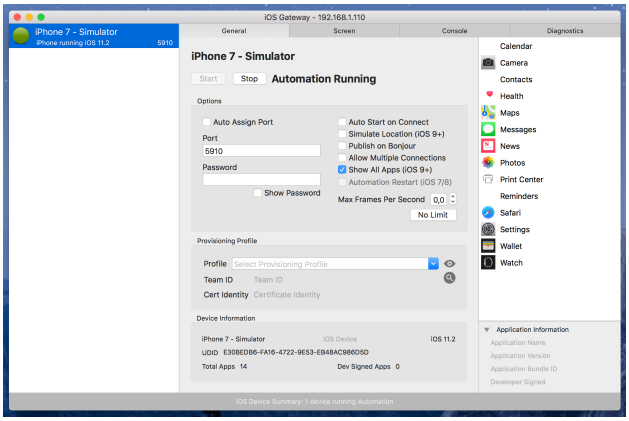
Orodje iOS Gateway [53] ponuja enako funkcionalnost kot orodje Android Gateway le da je namenjeno povezavi z napravami z operacijskim sistemom iOS. Uporabniški vmesnik orodja iOS Gateway je prikazan na sliki 3.14.

3.3.1.3 Eggplant AI

Eggplant AI [54] je spletna aplikacija, ki s pomočjo algoritmov za učenje izdela testne primere. Testni primeri se izdelajo iz preprostega modela (slika



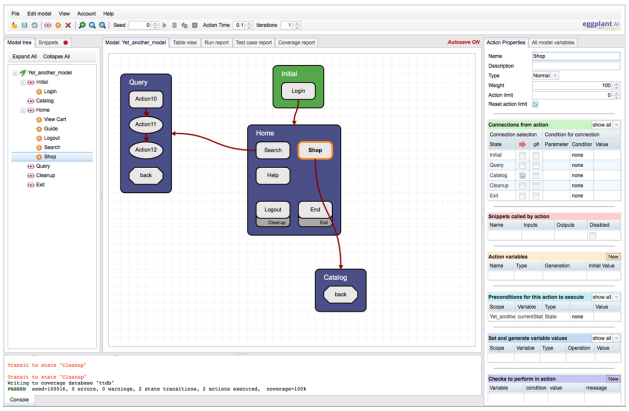
Slika 3.13: Povezava testne naprave s pomočjo orodja Android Gateway.



Slika 3.14: Povezava testne naprave s pomočjo orodja iOS Gateway.

3.15), ki predstavlja stanja, strani aplikacije in akcije, ki jih uporabnik v določenem stanju lahko izvede in uporabnika pripeljejo iz enega stanja v drugo.

Eggplant AI se ob generiranju testnih primerov sprehodi čez vse možne poti v podanem modelu, ki predstavljajo tudi možne poti uporabnika v testirani aplikaciji. Odkrite poti uporabnika se nato uporabijo za izdelavo testne skripte v orodju Eggplant Functional [54]. Pretvarjanje akcij in stanj v testno skripto poteka s pomočjo delčkov kode (angl. snippets) [55], ki vsebujejo ukaze, ki se ob določenem stanju ali akciji izvedejo.



Slika 3.15: Model stanj v uporabniškem vmesniku Eggplant AI [54].

3.3.2 Programski jeziki

Za pisanje testnih skript v okolju Eggplant se uporablja programski jezik SenseTalk [49]. SenseTalk je objektno orientiran, visokonivojski in skriptni programski jezik, ki je bil razvit z namenom poenostavljenega pisanja in izdelave avtomatskih testov uporabnikom, ki nimajo veliko izkušenj s programiranjem. Sintaksa jezika je podobna angleškim stavkom, zaradi česar je lahko berljiv in razumljiv hkrati pa tudi enostaven za pisanje [49].

V izseku 3.1 je prikazan primer zaporedja ukazov v programskem jeziku SenseTalk.

```
# Nastavi vrednost spremenljivki stevilo na 10
set stevilo to 10
# Povecaj vrednost spremenljivke stevilo za 1
add 1 to stevilo
# Izpisi vrednost spremenljivke stevilo (11)
put stevilo
```

Izsek 3.1: Primer skripte v programskem jeziku SenseTalk

3.3.3 Podprti operacijski sistemi

Teste izdelane v orodju Eggplant Functional lahko izvajamo na testnih napravah z operacijskim sistemom Microsoft Windows, Apple MacOS, Linux, Android, iOS in Windows Phone oziroma na vsaki napravi z VNC povezavo.

3.3.4 Podprti brskalniki

Z orodjem Eggplant Functional lahko avtomatiziramo vse brskalnike in aplikacije, saj je delovanje orodja neodvisno od aplikacije, ki jo testiramo.

Poglavje 4

Primerjava orodij

Izbrana orodja bomo primerjali po treh kriterijih:

- **Izdelava testov**: V izbranih orodjih bomo prikazali postopek izdelave testov na podlagi testnega primera ter omenili nekaj prednosti in omejitev posameznih orodij.
- **Preformančna primerjava**: Pri performančni primerjavi bomo izbrana orodja primerjali po hitrosti izvajanja izdelanih testov. Pozorni bomo predvsem na odstopanja v času izvajanja, za katere bomo kasneje poizkušali poiskati vzroke.
- **Zanesljivost izvajanja**: Primerjali bomo zanesljivost delovanja posameznih orodij ob zagonu izdelanih testov. Pri zanesljivosti bomo pozorni predvsem na napake, ki so se zgodile med izvajanjem in so posledica nepravilnega delovanja orodja.

4.1 Izdelava testov na podlagi testnega primera

S pomočjo podanega testnega scenarija bomo prikazali izdelavo testov in nekaj omejitev izbranih orodij. Za izdelavo testov v orodjih Selenium in

Appium bomo uporabili programski jezik Python, medtem ko bodo testi v orodju Eggplant Functional izdelani s pomočjo skriptnega jezika SenseTalk [49].

4.1.1 Opis testirane aplikacije

Za primerjavo testnih orodij smo razvili spletno aplikacijo, ki jo bomo uporabili za izdelavo in izvajanje avtomatiziranih testov v orodjih Selenium, Appium in Eggplant Functional. Spletna aplikacija je namenjena pregledovanju in shranjevanju izvajanj testov in je sestavljena iz treh podstrani:

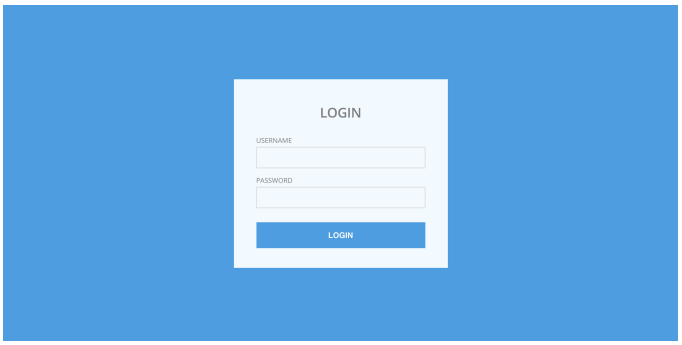
1. Stran s prijavnim oknom (slika 4.1)
2. Seznam izvajanj testov (slika 4.2)
3. Podrobnosti izbranega testnega izvajanja (slika 4.3)

Izdelana spletna aplikacija je preprosta, a vsebuje ključne gradnike (tekstovna polja, vnosna polja, gumbe, navigacijo in povezave), ki se uporabljajo za izdelavo grafičnih vmesnikov spletnih aplikacij in jih bomo uporabili pri izdelavi testov.

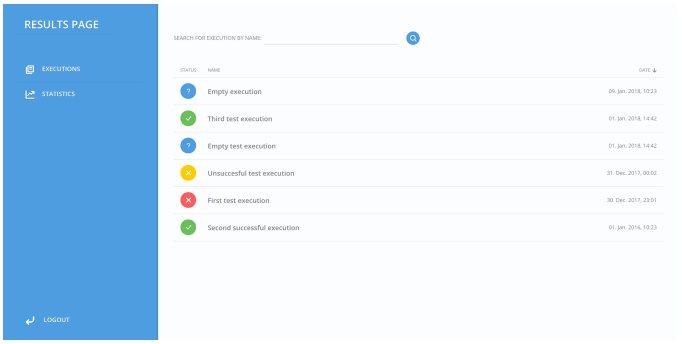
4.1.2 Opis testnega primera

Izdelali bomo avtomatizirane teste za testiranje prijavnega okna. Prijavno okno služi prijavi v spletno aplikacijo z uporabniškim imenom in geslom. Z uspešno prijavo se uporabniku omogoči dostop do podstrani za pregledovanje seznama testnih izvajanj. V primeru neuspešne prijave se uporabnika o napaki obvesti preko sporočila v prijavnem oknu.

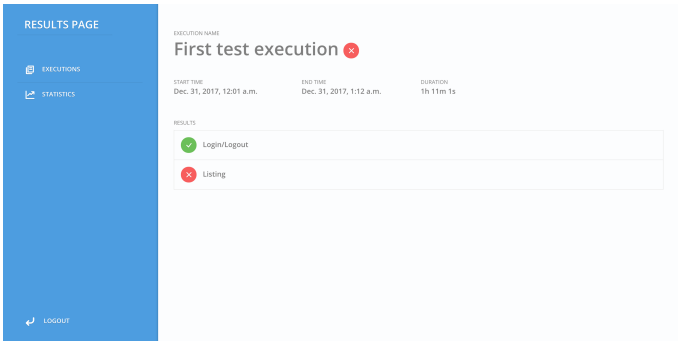
Testi za prijavno okno bodo v vnosna polja za `uporabniško ime` in `geslo` vnašali različne vrednosti in preverjali ali je bil uporabniku dostop na podlagi podanih vrednosti odobren ali zavrnjen. Vrednosti za vnosni polji uporabniško ime in geslo ter pričakovan rezultat, ki jih bomo uporabili za testiranje prijavnega okna, so zapisani v tabeli 4.1.



Slika 4.1: Podstran s prijavnim oknom.



Slika 4.2: Podstran s seznamom testnih izvajanj.



Slika 4.3: Podstran s podrobnostimi izbranega testnega izvajanja.

Uporabniško ime	Geslo	Rezultat prijave
(prazno)	(prazno)	Neuspešno
test	(prazno)	Neuspešno
(prazno)	test1234	Neuspešno
incorrectUser	test1234	Neuspešno
test	incorrectPassword	Neuspešno
test	test1234	Uspešno

Tabela 4.1: Pari vrednosti uporabniško ime in geslo ter pričakovan rezultat prijave.

Orodja Selenium in Appium delujeta na principu preiskovanja DOM drevesa, zato moramo za izdelavo testov poznati HTML strukturo strani, ki jo želimo avtomatizirati. HTML struktura prijavnega okna, ki bo predmet testiranja, je prikazana v izseku 4.1.

```
<form id="login-form" method="post" class="gray">
    <h2 class="text-center gray">Login</h2>

    <div class="login-form">
        <div class="input-group">
            <label for="id_username" class="small-title">Username</label>
            <input id="id_username" name="username" type="text">
        </div>
        <div class="input-group">
            <label for="id_password" class="small-title">Password</label>
            <input id="id_password" name="password" type="password">
        </div>
    </div>

    <input type="submit" value="Login" class="btn bg-blue">
    <input type="hidden" name="next" value="/">
</form>
```

Izsek 4.1: HTML struktura prijavnega okna.

4.1.3 Izdelava testov za podan testni primer

Za podani testni primer bomo v izbranih orodjih izdelali testne skripte, ki jih bomo kasneje uporabili za primerjavo zanesljivosti in hitrosti delovanja izbranih orodij.

Pred izdelavo testov bomo spoznali in opisali osnovne funkcionalnosti izbranih orodij s pomočjo katerih bomo nato izdelali teste. Osredotočili se bomo na funkcionalnosti, ki so najpogosteje uporabljene in zadoščajo za izdelavo preprostih testov:

- vzpostavitev povezave s testno napravo.
- odpiranje aplikacij in spletnega naslova v brskalniku.
- iskanje elementov uporabniškega vmesnika.
- interakcija z elementi uporabniškega vmesnika.
- branje vrednosti elementa uporabniškega vmesnika.

4.1.3.1 Vzpostavitev povezave s testno napravo

Prvi korak za izvajanje in razvoj testov je vzpostavitev povezave za komuniciranje in upravljanje s testno napravo.

Selenium

Za vsak brskalnik, ki bo predmet testiranja, moramo iz spleta prenesti pripadajoči WebDriver gonilnik in ga namestiti na testno napravo. Povezavo s spletnim brskalnikom vzpostavimo z inicializacijo WebDriver objekta (izsek 4.2), ki zažene nameščen WebDriver gonilnik za izbrani brskalnik. Ob inicializaciji objekta se na testni napravi odpre izbrani brskalnik, ki je pripravljen za izvajanje prejetih ukazov.

```
from selenium import webdriver

# Vzpostavitev povezave z Google Chrome brskalnikom
driver = webdriver.Chrome()
# Vzpostavitev povezave s Firefox brskalnikom
driver = webdriver.Firefox()
# Vzpostavitev povezave z Safari brskalnikom
driver = webdriver.Safari()
```

Izsek 4.2: Vzpostavitev povezave z brskalniki v orodju Selenium

Za vzpostavitev povezave z oddaljeno testno napravo je na testno napravo potrebno namestiti `Selenium Server`, ki ga zaženemo z ukazom `java -jar selenium-server-standalone-x.x.x.jar`. Privzeto bo strežnik dostopen na vratih `4444`, ki jih moramo skupaj z IP naslovom podati ob inicializaciji WebDriver objekta, ki bo vse ukaze pošiljal na podani naslov in vrata.

```
from selenium import webdriver
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities

# Vzpostavitev povezave z Selenium streznikom
# na naslovu 192.168.0.1 in vratih 4444
driver = webdriver.Remote(
    command_executor='http://192.168.0.1:4444/wd/hub',
    desired_capabilities=DesiredCapabilities.CHROME
)
```

Izsek 4.3: Vzpostavitev povezave z brskalnikom na oddaljeni napravi

Ob zaključku izvajanja je potrebno povezavo s spletnim brskalnikom prekiniti s klicem metode `quit()` na instanci WebDriverja (izsek 4.4).

```
# Prekinitev povezave z brskalnikom
driver.quit()
```

Izsek 4.4: Prekinitev povezave z brskalnikom

Appium

Vzpostavitev povezave s testno napravo v orodju Appium poteka podobno kot povezava z oddaljeno napravo v orodju Selenium.

Za vsako testno napravo je potrebno zagnati Appium strežnik, ki skrbi za komunikacijo s testno napravo. Appium strežnik namestimo z uporabo upravljalca Javascript paketov NPM in zaženemo z ukazom `appium -p vrata`, ki mu s pomočjo parametra podamo vrata, na katerih bo strežnik poslušal. Namestitev in zagon Appium strežnika je mogoča tudi s prenosom aplikacije za namizne računalnike, ki nam omogoča zagon strežnika preko grafičnega vmesnika.

Ob inicializaciji WebDriver objekta je potrebno podati željene nastavitve testne naprave. Nastavitve so množica parov ključ-vrednost, ter morajo vsebovati vsaj naslednje ključe [56]:

- `platformName`: ime platforme (Android, iOS, itd.).
- `platformVersion`: različica operacijskega sistema.
- `deviceName`: ime testne naprave.
- `app` ali `browser`: ime aplikacije ali brskalnika, v katerem se bodo izvajali testi (npr. Chrome, Safari, itd.).
- `automationName`: ime gonilnika, ki ga želimo uporabiti za avtomatizacijo.

Med možnimi željenimi nastavitvami obstajajo tudi ključi, ki so specifični za določeno platformo. Tako moramo za zagon testov na fizičnih napravah s platformo iOS ob vzpostavitvi podati ID naprave, ID ekipe, v katero smo vključeni na Apple portalu za razvijalce, in ime certifikata za podpisovanje iOS aplikacij. V kolikor želimo na platformi iOS izvajati teste v brskalniku Safari, moramo za testno napravo zagnati tudi `iOS WebKit Debug Proxy` [57], ki omogoča pošiljanje ukazov v brskalnik Safari.

```
from appium import webdriver

# Primer nastavitev za vzpostavitev povezave z fizicno testno napravo iOS
desired_caps = {
    'deviceName': 'iPhone 6',
    'platformName': 'iOS',
    'platformVersion': '11.2',
    'browserName': 'Safari',
    'automationName': 'XCUITest',
    'udid': '9d6fe50db64aa84ea2c0f7e891bac08482abd292',
    'xcodeOrgId': 'TA6EFMT5CX', # ID podjetja
    'xcodeSigningId': 'iPhone Developer',
    'startIWDP': True
}

driver = webdriver.Remote('http://0.0.0.0:4723/wd/hub', desired_caps)
```

Izsek 4.5: Primer vzpostavitve povezave z napravo iOS v orodju Appium

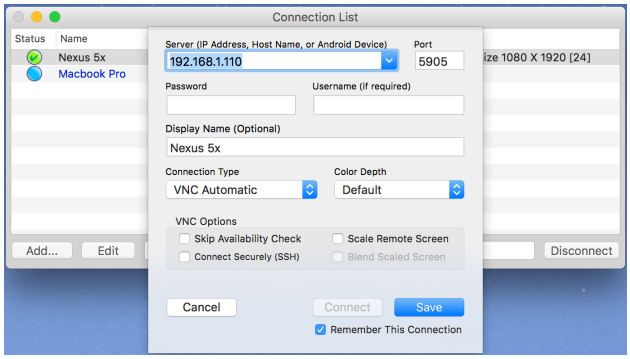
Eggplant Functional

Povezavo s testno napravo v orodju Eggplant Functional vzpostavimo s pomočjo okna `Connection list` (slika 4.4). Komunikacija med orodjem Eggplant Functional in testno napravo poteka preko povezave VNC. Za vzpostavitev VNC povezave z namiznim računalnikom je nanj potrebno namestiti VNC strežnik, medtem ko namestitev in zagon strežnika VNC za Android in iOS naprave poteka s pomočjo Android Gateway (poglavje 3.3.1.2) ali iOS Gateway (poglavje 3.3.1.2) orodij.

V oknu `Connection list` moramo za vsako napravo podati:

- IP naslov strežnika VNC
- vrata, na katerih testna naprava posluša za vpostavitev VNC povezave
- prikazno ime testne naprave

Povezava se vzpostavi ob dvojnem kliku na shranjeno testno napravo v seznamu povezav. Povezavo lahko vzpostavimo tudi v času izvajanja testne skripte s pomočjo ukaza `Connect` in prikaznega imena testne naprave.



Slika 4.4: Okno za vzpostavljanje povezave z napravo znotraj orodja Eggplant Functional.

```
Connect "Nexus 5x"
```

Izsek 4.6: Vzpostavitev povezave s testno napravo v času izvajanja skripte

4.1.3.2 Odpiranje aplikacij in spletnega naslova v brskalniku

Prvi del testov je običajno odpiranje željene testne aplikacije ali v primeru testiranja spletne aplikacije odpiranje spletnega brskalnika in željenega spletnega naslova.

Selenium in Appium

Orodja Selenium in Appium testno aplikacijo odpreta ob inicializaciji WebDriverja. V kolikor naši testi zahtevajo večkratno odpiranje in zapiranje aplikacije, lahko v orodju Appium uporabimo metodi `driver.launch_app()` za odpiranje in `driver.close_app()` za zapiranje aplikacije.

Za odpiranje spletnega naslova v zagnanem brskalniku je v obeh orodjih na voljo metoda `driver.get(url)`, ki mu kot argument podamo spletni naslov. Metoda počaka, da se spletna stran v celoti naloži in nato nadaljuje z izvajanjem testne skripte.

Eggplant Functional

Odpiranje aplikacij na mobilnih napravah in namiznih računalnikih se v orodju Eggplant functional precej razlikuje. Za odpiranje aplikacij na mobilnih napravah je na voljo ukaz `LaunchApp "applicationName"` [58], ki mu kot argument podamo ime aplikacije nameščene na testni napravi. Ukaz za odpiranje aplikacij na namiznih računalnikih ne obstaja, zato ga moramo implementirati sami.

Izdelali smo funkcijo za odpiranje aplikacij na operacijskem sistemu Apple MacOS, kjer smo uporabili privzeti iskalnik *Spotlight Search* [59], preko katerega lahko poiščemo in zaženemo vse aplikacije, ki so nameščene na napravi (izsek 4.7). Slabost takšnega pristopa pri odpiranju aplikacij je, da je delovanje omejeno na specifičen operacijski sistem in moramo za delovanje na drugih operacijskih sistemih implementirati drugačen postopek.

```
on OpenApp appToLaunch
    // Bliznjica za zagon Spotlight Search
    TypeText CommandKey, space
    // V Spotlight Search vnesemo ime aplikacije
    TypeText appToLaunch
    // In jo z pritiskom tipke Enter odpremo
    TypeText enterKey
end OpenApp

on CloseApp appToClose
    // Aplikacijo postavimo v ospredje
    OpenApp appToClose
    // Zapremo vsa okna podane aplikacije
    TypeText CommandKey, "q"
end CloseApp
```

Izsek 4.7: Funkciji za odpiranje in zapiranje aplikacij na namiznih računalnikih z operacijskim sistemom MacOS

Podoben problem se pojavi pri odpiranju spletnega naslova. Ker se orodje Eggplant Functional ne zaveda odprte aplikacije, moramo funkcionalnost za odpiranje spletnega naslova na namiznih računalnikih in mobilnih napravah implementirati sami. V izsekih 4.8 in 4.9 je prikazano odpiranje spletnega naslova na namiznih računalnikih z operacijskim sistemom MacOS in mobilnih

napravah Android.

```
on OpenBrowser browserName, url
    OpenApp browserName
    wait 2

    // Odpri nov zavihek brez belezenja zgodovine
    if browserName is "Firefox" then
        TypeText shiftKey, CommandKey, "P"
    else
        TypeText shiftKey, CommandKey, "N"
    end if

    // Izberi URL vrstico in vpisi URL naslov
    TypeText CommandKey, "l"
    TypeText url
    TypeText enterKey
end OpenBrowser
```

Izsek 4.8: Funkcija za odpiranje brskalnika in spletnega naslova v orodju Eggplant na namiznih računalnikih z operacijskim sistemom MacOS.

```
to handle OpenChromeBrowser url
    set suiteImageFolder to folder() & "utils.suite/Images/"
    set appName to "com.android.chrome"

    // Odpri aplikacijo Google Chrome
    LaunchApp appName

    // Poisci ikono za meni in odpri nov zavihek
    Click ("ImageName": suiteImageFolder & "chrome/menu_icon")
    Click("Text": "New incognito tab")

    // Poisci in izberi polje za vnos spletne naslova URL
    Click("Text": "Search or type URL")
    # Vnesi spletni naslov
    TypeText url
    TypeText returnKey
end OpenChromeBrowser
```

Izsek 4.9: Funkcija za odpiranje brskalnika in spletnega naslova v orodju Eggplant na mobilnih napravah Android.

4.1.3.3 Iskanje elementov

Iskanje elementov v aplikaciji ali na spletni stran je najbolj pogosto uporabljena funkcionalnost testnih orodij, ki jo testne skripte uporabljajo. Uporabljajo se za preverjanje prisotnosti iskanih elementov v aplikaciji, hkrati pa je v večini primerov osnova za vso interakcijo z elementi, saj moramo vsak element pred interakcijo najprej poiskati.

Selenium in Appium

Iskanje elementov v orodjih Selenium in Appium poteka s klicanjem metod na WebDriver objektu. Elemente izbiramo s pomočjo izbirnikov (angl. selectors) v DOM drevesu spletne aplikacije. Izbirniki se navezujejo na zgradbo HTML dokumenta in lastnosti posameznega elementa. Element lahko izberemo z različnimi pristopi, vendar moramo biti pri izbiri izbirnikov previdni, saj želimo uporabiti izbirnik, ki najbolj enolično opisuje naš iskani element in za katerega obstaja najmanjša verjetnost, da se bo ob razvoju programskega izdelka spremenil.

Nekaj najpogostejših izbirnikov bomo prikazali s pomočjo HTML strukture prijavnega okna testne aplikacije (izsek 4.1). Elemente lahko izbiramo glede na [60]:

- **ID elementa**: predstavlja enoličen identifikator elementa in se lahko pojavi samo enkrat v strukturi spletne strani, zato je tudi najbolj priporočen in učinkovit pristop za izbiro elementa. Primer:

  Element: `<input id="id_username" name="username" type="text">`

  Izbirnik: `driver.find_element_by_id("id_username")`

- **Ime razreda elementa (angl. Class Name)**: omogoča izbiro vseh elementov z enakim razredom. Primer:

  Element: `<input type="submit" value="Login" class="btn bg-blue">`

Izbirnik: `driver.find_elements_by_class_name("btn")`

- **Ime značke (angl. Tag Name)**: omogoča izbiro elementa z željeno HTML značko. Primer:

  Element: `<form id="login-form" method="post" class="gray"></form>`

  Izbirnik: `driver.find_element_by_tag_name("form")`

- **Ime (angl. Name)**: omogoča izbiro elementa preko HTML atributa `name`. Primer:

  Element: `<input id="id_password" name="password" type="password">`

  Izbirnik: `driver.find_element_by_name("password")`

- **CSS**: omogoča izbiro elementa s pomočjo CSS izbirnikov, ki omogočajo izbiro po več lastnostih hkrati. Primer:

  Struktura: `<div class="login-form"><div class="input-group"><input id="id_username" name="username" type="text"></div></div>`

  Izbirnik: `driver.find_element_by_css_selector("div.login-form .input-group input[name="username"]")`

- **XPath**: namenjen je preiskovanju elementov in atributov v XML dokumentu.

  Primer:

  `driver.find_elements_by_xpath("/html/body/div/form/h2")`

Orodje Appium za iskanje elementov v mobilnih aplikacijah dodaja izbirnike, ki so specifični za mobilne naprave in platforme. Tako imamo dodatno možnost izbire po:

- `accessibility id`: predstavlja enoličen identifikator elementa v uporabniškem vmesniku aplikacije.

Primer: `driver.find_elements_by_accessibility_id("id_primer")`

- `ios uiautomation`: omogoča iskanje elementa z uporabo UIAutomation knjižnice [61].

  Primer:
  ```
  driver.find_element_by_ios_predicate(
      "tableViews()[1].cells().firstWithPredicate(
          "label == 'Olivia'"
      )"
  )
  ```

- `android uiautomator`: omogoča iskanje elementa s pomočjo UiAutomator [62].

  Primer:
  ```
  driver.find_element_by_android_uiautomator(
      "new UiSelector().className("android.widget.TextView")
      .instance(0)"
  )
  ```

Pri izdelavi testov se moramo zavedati, da ni nujno, da bo celotna vsebina strani naložena hitro ali sočasno. Problem se pojavi predvsem pri testiranju aplikacij, ki uporabljajo asinhrono (angl. asynchronous) nalaganje ali spreminjanje vsebine s pomočjo AJAX [63] zahtev. Zgodi se lahko, da iskani element v času iskanja ne bo obstajal v DOM drevesu, kar bo povzročilo napako. Omenjenim napakam se lahko izognemo s čakanjem na prisotnost elementa na strani. V orodjih Selenium in Appium sta na voljo dva načina čakanja [64]:

- izrecno čakanje (angl. Explicit Waits) omogoča določitev čakalnega časa za določeno iskanje. Uporabljamo ga kadar potrebujemo različne čakalne čase za različna iskanja. Na primeru 4.10 je prikazano izrecno čakanje na element z IDjem `login-form` s časovno omejitvijo 10 sekund. Časovna omejitev določa, kdaj bo orodje prenehalo z iskanjem elementa in sprožilo izjemo.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Chrome()
driver.get('http://192.168.1.110:8000/login')

element = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "login-form"))
)
```

Izsek 4.10: Primer izrecnega čakanja na element

- brezpogojno čakanje (angl. Implicit Waits) omogoča določitev čakalnega časa za vsa iskanja znotraj seje. Za razliko od izrecnega čakanja, kjer se čakalni čas določi ob samem iskanju elementa, se brezpogojni čas določi na objektu `WebDriver`.

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.implicitly_wait(10)

driver.get('http://192.168.1.110:8000/login')
element = driver.find_element_by_id("login-form")
```

Izsek 4.11: Primer brezpogojnega čakanja na element

Eggplant Functional

Prvi in najbolj pogost način iskanja elementov v orodju Eggplant Functional je iskanje vnaprej zajetih slik. Ob izdelavi testov moramo za vsak element, ki ga želimo uporabiti v testni skripti, posneti referenčno sliko, ki predstavlja pravilen videz iskanega elementa. Referenčne slike zajamemo s pomočjo uporabniškega vmesnika, ki nam omogoča obrezovanje, zajemanje in shranjevanje zaslonske slike testne naprave. Za iskanje zajetih referenčnih slik je na voljo več funkcij:

- `ImageFound(10, "ImeSlike")`: kot argumenta sprejme časovno omejitev iskanja in ime slike. Funkcija vrne rezultat `True`, če je slika `ImeSlike` prisotna in `False`, v kolikor slika ni prisotna na zaslonu.
- `ImageLocation("ImeSlike")`: vrne koordinati `x` in `y`, ki predstavljata pozicijo podane slike.
- `ImageRectangle("ImeSlike")`: vrne para koordinat `x` in `y`, ki predstavljata poziciji levega zgornjega in desnega spodnjega kota slike.
- `WaitFor(10, "PrvaSlika", "DrugaSlika")`: znotraj časovne omejitve čaka na pojavitev katerekoli od podanih slik na zaslonu.
- `WaitForAll(10, "PrvaSlika", "DrugaSlika")`: za razliko od funkcije `WaitFor` funkcije čaka na pojavitev vseh podanih slik.

Drugi način iskanja elementov v orodju Eggplant Functional je iskanje besedila na zaslonu testne naprave. Uporablja se, ko za iskanje ne moremo uporabiti slik [65]:

- vrednost besedila poznamo šele ob izvajanju skripte,
- pisava, velikost ali barva besedila se spreminja,
- iz zaslona testne naprave želimo prebrati vrednost besedila.

Za iskanje besedila se privzeto uporablja OCR pogon, ki iz slike zaslona testne naprave prepozna posamezne črke, jih sestavi v nize in tako prebere vsa besedila med katerimi nato poišče iskani niz. V kolikor želimo lahko za iskanje besedila uporabimo tudi tako imenovane TIG generatorje [66], ki nam iz podanega besedila in podanih podatkov (pisava, velikost, barva, ...) izdelajo sliko z besedilom, ki jo nato uporabijo za iskanje besedila na napravi.

```
ImageFound(Text: "Login", SearchRectangle: (200,150,400,450), WaitFor: 10)
```

Izsek 4.12: Iskanje besedila „Login“ znotraj podanega pravokotnika.

Iskalna funkcija za hitrejše in bolj zanesljivo delovanje podpira dodatne parametre:

- `SearchRectangle`: omeji iskanje besedila na podan prostor na zaslonu testne naprave. Iskalni prostor je predstavljen kot pravokotnik, ki je določen s koordinatami levega zgornjega in spodnjega desnega kota pravokotnika, kjer za vsak kot podamo x in y koordinato. Namesto parov koordinat lahko podamo sliki, ki ju bo orodje uporabilo za določitev iskalnega pravokotnika.

- `Contrast`: omogoča izboljšanje kontrasta med besedilom in ozadjem. Parameter lahko zavzame vrednosti `On` ali `Off`. V kolikor poznamo barvo ozadja, jo preko dodatnega parametra `ContrastColor` podamo iskalni funkciji. Točke (angl. Pixels) na sliki, ki bodo ustrezale podani barvi, se bodo pred prepoznavo obarvale z belo barvo, preostale pike pa se bodo obarvale s črno barvo.

- `ValidCharacters`: omeji iskanje OCR pogona le na podano množico znakov. Uporablja se za preprečevanje napačnega prepoznavanja podobnih znakov (npr. črka O in številka 0).

- `IgnoreSpaces`: OCR pogon se ne bo zmenil za presledke med posameznimi znaki. Uporablja se kadar OCR pogon zaradi razmika med posameznimi znaki bodisi ne zazna obstoječih ali pa zazna neobstoječe presledke med besedami.

- `Language`: omeji iskanje besedila na slovar besed podanega jezika.

4.1.3.4 Interakcija z elementi

Selenium in Appium

V orodjih Selenium in Appium je element pred interakcijo potrebno poiskati z uporabo zgoraj opisanih metod (podpoglavje 4.1.3.3). Iskalna funkcija nam vrne objekt `WebElement` [67], ki predstavlja izbrani element v naši testni

aplikaciji. Objekt `WebElement` za interakcijo z različnimi elementi implementira več metod:

- `click()`: izvede klik na izbrani element. Primer:

  ```
  driver.find_element_by_class_name('logout-button').click()
  ```

- `send_keys(vrednost)`: vnese podano vrednost v izbrano vnosno polje. Podana vrednost je lahko niz ali pot do datoteke, ki jo želimo podati v vnosno polje. Primer:

  ```
  driver.find_element_by_id('id_username').send_keys('Janez')
  ```

- `clear()`: pobriše obstoječo vrednost v izbranem vnosem polju. Primer:

  ```
  driver.find_element_by_id('id_username').clear()
  ```

- `submit()`: pošlje obrazec (angl. form) z vrednostmi elementov znotraj izbranega obrazca. Funkcionalnost te metode je enaka kliku na gumb za pošiljanje obrazca. Primer:

  ```
  driver.find_element_by_id('login-form').submit()
  ```

Selenium preko razreda `ActionChains` [68] ponuja tudi naprednejše interakcije, kot so povleci in spusti (angl. drag and drop), klikni in drži (angl. click and hold), pritisni (angl. key down), spusti (angl. key up), itd.

Interakcija z napravami z zaslonom na dotik poteka preko dotika na zaslon, zato se razlikuje od namiznih računalnikov, pri katerih praviloma napravo upravljamo z miško in tipkovnico. V orodju Appium je interakcija z napravami z zasloni na dotik podprta preko razreda `TouchAction` [69], ki omogoča veriženje akcij. Razred `TouchAction` dodaja akcije:

- `tap`: izvede pritisk na zaslon. Primer:

  ```
  TouchAction(driver).tap(element).perform()
  ```

- `press`: izvede pritisk na zaslon.

- `long_press`: izvede daljši pritisk na zaslon.

- `release`: spusti pritisk zaslona.
- `move_to`: izvede premik na izbrano točko ali koordinate.

Sodobne naprave z zaslonom na dotik podpirajo večdotičnost (angl. multitouch), ki omogoča sočasni pritisk na več točk. Za podporo večdotičnosti orodje Appium implementira razred `MultiAction` [70], s pomočjo katerega lahko sočasno izvedemo več akcij `TouchAction` (izsek 4.13).

```
from appium.webdriver.common.touch_action import TouchAction
from appium.webdriver.common.multi_action import MultiAction

action_1 = TouchAction().press(50, 200).move_to(150, 200).release()
action_2 = TouchAction().press(300, 200).move_to(200, 200).release()

# Socasno zvede action_1 in action_2
MultiAction(self.driver).add(action_1, a_action_2).perform()
```

Izsek 4.13: Primer večdotične interakcije v orodju Appium.

Eggplant Functional

Orodje Eggplant podpira izvajanje interakcij z namiznimi računalniki in napravami z zaslonom na dotik. Interakcija s testno napravo se izvede s pomočjo koordinat, ki predstavljajo točko in pozicijo interakcije na zaslonu testne naprave. Pozicija interakcije se določi z iskanjem pozicije podane slike ali besedila na zaslonu. Funkcijam za interakcijo lahko namesto slike ali besedila podamo par koordinat x in y, ki predstavlja točko na zaslonu. Akcije podprte v orodju Eggplant so [71, 72, 73]:

- `Click` in `DoubleClick`: izvede enojni ali dvojni klik na podano pozicijo. Primer:

  ```
  Click(Image:"login.png")
  ```

- `MoveTo`: izvede premik miške na podano pozicijo. Primer:

  ```
  MoveTo(Text:"Login")
  ```

- `DragAndDrop`: izvede pritisk levega gumba miške na prvi poziciji, premakne miško na preostale pozicije in spusti miško na zadnji podani poziciji. Primer:

  `DragAndDrop (100, 200), (400, 200), (200, 200)`

- `Tap`: izvede pritisk podane pozicije na zaslonu mobilne naprave. Primer:

  `Tap((100, 200))`

- `PressBackButton`: izvede pritisk na gumb za vračanje na predhodno aktivnost mobilne naprave.

- `PressHomeButton`: izvede pritisk na gumb za vrnitev na domači zaslon mobilne naprave.

- `SwipeDown, SwipeLeft, SwipeRight` in `SwipeUp`: izvedejo poteg v izbrano smer.

- `KeyDown` in `KeyUp`: pritisne ali spusti podane tipke.

- `TypeText`: vpiše podan niz v aktivni gradnik na testni napravi.

4.1.3.5 Branje vrednosti elementa

Selenium in Appium

Vrednost elementa lahko v orodjih Selenium in Appium pridobimo na dva načina:

- `element.text`: vrne niz znakov znotraj začetne in zaključne značke izbranega elementa. Primer:

  Element: `<h2 class="text-center gray">Login</h2>`

  Branje vrednosti elementa:

  ```
  # Vrednost spremenljivke title je enaka nizu "Login"
  title = driver.find_element_by_tag_name("h2").text
  ```

- `element.get_attribute("value")`: vrne vrednost v vnosnem polju elementa, ki jo uporabnik lahko spreminja preko uporabniškega vmesnika.

  Element:
  ```
  <input id="id_username" name="username" type="text"
  value="Janez">
  ```

  Branje vrednosti elementa:
  ```
  element = driver.find_element_by_id("id_username")
  # Vrednost spremenljivke name je enaka nizu "Janez"
  name = element.get_attribute("value")
  ```

Eggplant Functional

Orodje Eggplant omogoča branje vrednosti nizov iz zaslona s pomočjo funkcije `ReadText` [74]. Funkcija `ReadText` prebere in vrne niz znotraj podanega pravokotnika, ki ga lahko določimo:

- s parom koordinat x in y. Primer:

  ```
  ReadText((225,240), (360,410))
  ```

- s parom slik, kjer prva slika določa levi zgornji kot in desna slika določa desni spodnji kot iskalnega pravokotnika. Primer:

  ```
  ReadText(("upperLeftImage", "bottomRightImage"))
  ```

4.1.3.6 Izdelava testov

Selenium in Appium

Pri izdelavi testov v orodjih Selenium in Appium smo želeli izdelati teste, ki jih bomo lahko brez spreminjanja poganjali tako na mobilnih napravah kot tudi namiznih računalnikih.

Ob izdelavi testov za podan testni primer smo ugotovili, da je v našem primeru edina razlika med orodjema Selenium in Appium vzpostavitev povezave s testno napravo oziroma inicializacija `WebDriver` objekta. Za inicializacijo

`WebDriver` objekta smo ustvarili razred `DriverSingleton` (priloga A.1), ki na podlagi vrednosti podanega parametra `platform` inicializira Selenium ali Appium `WebDriver` objekt.

Testi so izdelani s pomočjo ogrodja `unittest` [75], ki omogoča nalaganje in zaganjanje testov ter sporočanje rezultatov. Orodje `unittest` pred začetkom izvajanja testa kliče metodo `setUp`, kjer določimo korake za katere želimo, da se izvedejo pred izvajanjem testa. V našem primeru smo metodo `setUp` uporabili za odpiranje spletnega naslova v brskalniku.

V izseku 4.14 je prikazan primer testa prijavnega okna. Test preverja ali prijavno okno uporabnika prijavi in preusmeri na nadzorno ploščo spletne aplikacije, v kolikor mu podamo veljavno uporabniško ime in geslo. Prvi korak testa preveri ali je uporabnik prijavljen v spletno aplikacijo. V kolikor je uporabnik že prijavljen, potem uporabnika pred nadaljevanjem odjavimo. V naslednjem koraku se s podanim uporabniškim imenom in geslom poizkusimo prijaviti v spletno aplikacijo. Za testiranje prijavnega okna smo izdelali funkcijo (izsek 4.15), ki ji kot argument podamo uporabniško ime, geslo in WebDriver objekt. Funkcija poišče elemente prijavnega okna, podane vrednosti vnese v vnosna polja in pošlje obrazec s klikom na gumb za pošiljanje. Uspešnost prijave preverimo z branjem naslova spletne strani in primerjavo s pričakovanim naslovom.

Vsi testi izdelani s pomočjo orodij Selenium in Appium so priloženi v prilogi A.2.

```
def test_login_form_success(self):
    self.logout_if_logged_in()
    login(CONFIG.LOGIN_CREDENTIALS_USERNAME, CONFIG.LOGIN_CREDENTIALS_PASSWORD,
          self.driver)
    WebDriverWait(self.driver, 15).until(
        EC.visibility_of_element_located((By.CLASS_NAME, "main-navigation"))
    )
    self.assertTrue('Dashboard' in self.driver.title)
```

Izsek 4.14: Primer testa prijavnega okna v orodjih Selenium in Appium.

```
def login(username, password, driver):
    username_field = driver.find_element_by_id('id_username')
    password_field = driver.find_element_by_id('id_password')
    submit_button = driver.find_element_by_css_selector('input[type="submit"]')

    if username is not None and username != '':
        username_field.clear()
        username_field.send_keys(username)
    if password is not None and password != '':
        password_field.clear()
        password_field.send_keys(password)

    submit_button.click()
```

Izsek 4.15: Funkcija za vnos vrednosti v orodjih Selenium in Appium.

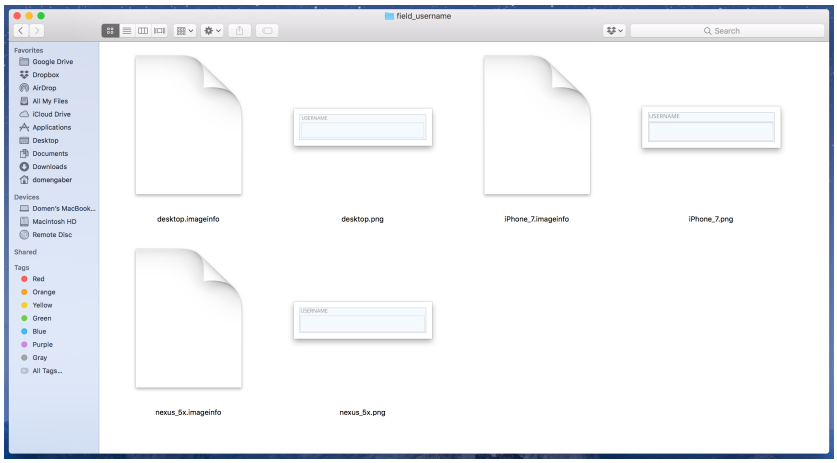
Eggplant Functional

V orodju Eggplant lahko vse izdelane teste izvajamo tako na mobilnih napravah kot tudi na namiznih računalnikih, saj za izvajanje na drugi testni napravi z ukazom `Connect "naprava"` preprosto povežemo drugo testno napravo. Za povezavo s testno napravo in odpiranje naslova smo izdelali funkcijo `setUp` (izsek 4.16).

```
to handle setUp platform, testApp
    set url to "192.168.1.110:8000"
    // Odpremo URL naslov v brskalniku
    if platform is "Android" then
        Connect "Nexus 5x"
        OpenChromeBrowser url
    else if platform is "iOS" then
        Connect "iPhone 7 - Simulator"
        // Connect "iPhone 7"
        OpenSafariBrowser url
    else if platform is "Desktop" then
        Connect "Macbook Pro"
        OpenBrowser testApp, url
    end if
end setUp
```

Izsek 4.16: Funkcija za povezavo s testno napravo v orodju Eggplant.

Pri izdelavi testov v orodju Eggplant je potrebno za vsako testno napravo posneti pripadajoče referenčne slike. Za vsak iskan element ustvarimo mapo, ki vsebuje referenčne slike elementa na različnih testnih napravah. Primer mape z referenčnimi slikami za vnosno polje je prikazan na sliki 4.5.



Slika 4.5: Mapa z referenčnimi slikami za vnosno polje.

Ob iskanju elementa namesto poti do referenčne slike podamo pot do mape z referenčnimi slikami elementa na različnih testnih napravah. Orodje Eggplant pri iskanju elementa uporabi vse referenčne slike elementa znotraj podane mape. Omenjen pristop nam omogoča, da ob dodajanju nove naprave posnamemo le referenčne slike in jih shranimo v željene mape, medtem ko lahko testne skripte ostanejo nespremenjene.

V izseku 4.17 je prikazan test prijavnega okna z veljavnim uporabniškim imenom in geslom. Podobno kot v orodjih Selenium in Appium smo tudi v orodju Eggplant izdelali funkcijo za vnos vrednosti v prijavno okno (izsek 4.18). Uspešnost prijave preverimo z iskanjem slike navigacijskega menija, ki je prisoten na nadzorni plošči.

```
// Uspesna prijava
login "test", "test1234"
evaluateImage "dashboard_navigation", "Successful login validation", 10
```

Izsek 4.17: Primer testa prijavnega okna v orodju Eggplant.

```
to handle login username, password

    // Vpis uporabniskega imena v vnosno polje
    Click(ImageName: "field_username", WaitFor: 15)
    TypeText username

    // Vpis gesla v vnosno polje
    Click "field_password"
    TypeText password

    // Klik na gumb za prijavo
    Click "button_submit"

end login
```

Izsek 4.18: Funkcija za vnos vrednosti v prijavno okno v orodju Eggplant.

Vsi testi izdelani v orodju Eggplant so priloženi v prilogi A.3.

4.2 Performančna primerjava

Pri performančni primerjavi bomo primerjali čas izvajanja testov v orodjih Selenium, Appium in Eggplant. Za primerjavo po času izvajanja testov smo se odločili, ker hitrost izvajanja avtomatskih testov spada med glavne prednosti avtomatizacije. Zaradi pogostega zaganjanja testov si želimo, da so testi izvedeni kar se da hitro.

Za primerjavo hitrosti izvajanja testov smo uporabili teste, ki smo jih izdelali v poglavju 4.1.3.6. Pri primerjavi se je potrebno zavedati, da so pridobljeni rezultati povprečnih časov izvajanja odvisni od testne aplikacije in testnega primera. Z izbiro drugačnega testnega primera ali testne aplikacije bi izdelali drugačno testno skripto, ki bi lahko vsebovala drugačno število iskanj, drugačne interakcije s testno napravo ali drugačen postopek odpiranja testne aplikacije. Naštete spremembe bi vplivale na povprečen čas izvajanja testov, zato bi lahko pridobili drugačne rezultate.

Izdelane teste smo v orodjih zaporedoma ponovili 50 krat in pri tem beležili čas, ki je potekel med začetkom in koncem izvajanja posameznega testa. Pri izbiri števila ponovitev je potrebno izbrati dovolj veliko število ne-

odvisnih ponovitev, s čimer zmanjšamo možnost vpliva naključnih dejavnikov na rezultate in povečamo natančnost rezultatov. Zaradi časovnih omejitev smo se odločili za 50 ponovitev, kar predstavlja kompromis med časom izvajanja in velikim številom ponovitev. Zato se zavedamo tveganja, da bi ob izbiri drugačnega števila ponovitev lahko pridobili nekoliko drugačne rezultate.

Za izvajanje testov smo uporabili 4 različne naprave z različnimi operacijskimi sistemi:

- **Namizni računalnik z operacijskim sistemom MacOS** in brskalnikom Google Chrome smo uporabili za zagon testov v orodjih Selenium in Eggplant.
- **Mobilno napravo z operacijskim sistemom iOS** in brskalnikom Safari smo uporabili za zagon testov v orodjih Appium in Eggplant.
- **Simulator operacijskega sistema iOS** z brskalnikom Safari smo uporabili za zagon testov v orodjih Appium in Eggplant.
- **Simulator operacijskega sistema Android** z brskalnikom Google Chrome smo uporabili za zagon testov v orodjih Appium in Eggplant.

Za izbrane naprave smo se odločili, ker smo želeli preveriti, če je hitrost izvajanja odvisna od operacijskega sistema testne naprave. Pri mobilnih napravah smo se odločili za izvajanje na simulatorjih operacijskih sistemov iOS in Android ter na pravi mobilni napravi z iOS. Z uporabo prave testne naprave iOS smo želeli preveriti, če obstaja razlika med izvajanjem na simulatorju, ki uporablja strojno opremo gostiteljevega računalnika, in izvajanjem na pravi mobilni napravi iOS. Vsa orodja ne podpirajo izvajanja testov v vseh okoljih, zato bomo orodja med seboj primerjali po posameznih okoljih. To pomeni, da ne moremo neposredno primerjati orodij Selenium in Appium, saj omogočata izvajanje testov na različnih okoljih.

Iz pridobljenih rezultatov smo izračunali povprečne čase izvajanja testov v posameznih orodjih na testnih napravah. Povprečni časi izvajanja testov so prikazani v tabeli 4.2.

Testna naprava	Selenium	Appium	Eggplant
Namizni računalnik	8s	/	44s
iOS naprava	/	29s	1m 17s
iOS simulator	/	27s	1m 12s
Android simulator	/	23s	1m 45s

Tabela 4.2: Povprečni časi izvajanja izdelanih testov v orodjih Selenium, Appium in Eggplant.

Iz rezultatov v tabeli 4.2 lahko ugotovimo, da je izvajanje testov v orodjih Selenium in Appium hitrejše od izvajanja testov v orodju Eggplant. Izvajanje testov v orodju Selenium je za ≈ 450 % hitrejše kot izvajanje testov v orodju Eggplant. Nekoliko manjša razlika je med orodjema Appium in Eggplant. Orodje Appium je pri avtomatizaciji testnih naprav z operacijskim sistemom iOS za ≈ 166 % hitrejše od orodja Eggplant. Pri avtomatizaciji testnih naprav z operacijskim sistemom Android je razlika v času izvajanja nekoliko večja, saj je orodje Appium za ≈ 350 % hitrejše od orodja Eggplant. Iz rezultatov lahko tudi razberemo, da je izvajanje na simulirani iOS napravi v orodjih Appium in Eggplant za ≈ 7 % hitrejše od izvajanja na pravi napravi.

Zaradi velikih razlik v povprečnih časih izvajanja med posameznimi orodji bomo podrobneje analizirali izvajanje testov. Najprej se bomo osredotočili na čas, ki ga orodje porabi za iskanje elementa. Za analizo smo v orodjih izdelali preprosto skripto, ki odpre spletni naslov prijavnega okna in počaka, da se stran v celoti naloži. Ko je spletna stran v celoti naložena, začnemo z iskanjem vnosnega polja. Iskanje smo ponovili 20 krat in za vsako iskanje shranili čas, ki je pretekel od začetka do konca iskanja. Iz pridobljenih časov smo izračunali povprečni čas iskanja elementa v posameznem orodju. Povprečni iskalni časi so prikazani v tabeli 4.3 .

Iz zbranih rezultatov lahko sklepamo, da je iskanje v orodjih Selenium in Appium hitrejše od iskanja elementov v orodju Eggplant. Iskanje elementa v brskalniku na namiznih računalnikih je v orodju Selenium za ≈ 2200 %

hitrejše od iskanja v orodju Eggplant. Podobno ugotovimo ob primerjavi rezultatov med orodjema Appium in Eggplant. Pri primerjavi iskanja na mobilnih napravah lahko ugotovimo, da je na napravah z operacijskim sistemom iOS iskanje v orodju Appium za ≈ 2860 % hitrejše od iskanja v orodju Eggplant in za ≈ 15560 % hitrejše na napravah z operacijskim sistemom Android. Sklepamo lahko, da je preiskovanje DOM drevesa, ki ga uporabljata orodja Selenium in Appium, hitrejše in bolj učinkovito od iskanja slike na zaslonu, ki se uporablja v orodju Eggplant.

Testna naprava	Selenium	Appium	Eggplant
Namizni računalnik	0,02s	/	0,46s
iOS naprava	/	0,05s	1,47s
iOS simulator	/	0,05s	1,49s
Android simulator	/	0,03s	4,70s

Tabela 4.3: Povprečni iskalni čas vnosnega polja v orodjih Selenium, Appium in Eggplant.

Ogledali si bomo tudi povprečne čase za vzpostavitev povezave testne naprave in odpiranja ter zapiranja brskalnika. Rezultati so prikazani v tabeli 4.4. Iz rezultatov lahko vidimo, da so povprečni časi v orodjih Selenium, Appium in Eggplant zelo različni. Za orodje Selenium je čas pričakovano nizek, saj je odpiranje spletnega naslova v brskalniku podprto znotraj samih gonilnikov. Prav tako je hitro vzpostavljanje povezave s testno napravo, saj je za vzpostavitev povezave potrebno ustvariti le novo sejo. Pri orodju Appium se povprečni čas poveča zaradi namestitve in zagona gonilnika na testni napravi. Najvišji povprečni čas za vzpostavitev povezave in odpiranje spletnega naslova ima orodje Eggplant, kar je posledica pomanjkanja podpore za odpiranje aplikacij in spletnega naslova v brskalniku. Zaradi tega se mora za odpiranje aplikacije in željenega spletnega naslova izvesti skripta, ki s pomočjo navigiranja skozi uporabniški vmesnik odpre spletni naslov (opisano v razdelku 4.1.3.2).

Testna naprava	Selenium	Appium	Eggplant
Namizni računalnik	1,0s	/	7,2s
iOS naprava	/	9,6s	18,7s
iOS simulator	/	5,8s	17,2s
Android simulator	/	9,9s	15,2s

Tabela 4.4: Povprečni čas vzpostavitve povezave in odpiranje ter zapiranje brskalnika v orodjih Selenium, Appium in Eggplant.

4.3 Zanesljivost delovanja

Ena izmed glavnih lastnosti, ki jih mora imeti orodje za avtomatizacijo testiranja, je zanesljivost delovanja. Zanesljivost orodja pomeni, da so napake, ki se zgodijo v času izvajanja posledica napake v testni aplikaciji ali testni skripti in niso posledica naključnih napak, ki se pojavijo zaradi nepravilnega delovanja orodja. V primeru prevelike nezanesljivosti obstaja tveganje, da nekatere nepake, ki so posledica nepravilnega delovanja testne aplikacije, spregledamo zaradi česar lahko sčasoma izgubimo zaupanje v izdelane teste.

Pri ocenjevanju zanesljivosti delovanja se bomo osredotočili na število napak, ki so se zgodile ob izvajanju testov in niso posledica napake v testni aplikaciji, ampak so posledica nepravilnega delovanja orodja ali testne skripte. Rezultate, ki so prikazani v tabeli 4.5, smo pridobili ob zagonu testov za primerjavo hitrosti izvajanja, kjer smo beležili tudi uspešnost izvajanj. Pri primerjavi zanesljivosti dopuščamo možnost drugačnih rezultatov, v kolikor bi uporabili drugačen testni primer ali spremenili število izvajanj.

V tabeli 4.5 so zapisani odstotki uspešnosti 50-ih izvajanj izdelanih testov. Orodja Selenium in Appium sta dosegla 100% in 99,3% uspešnost. V orodju Appium je pri izvajanju testov na simulatorju operacijskega sistema Android prišlo do napake pri vzpostavljanju povezave, kar je povzročilo, da se testi niso začeli izvajati. Problem se je samostojno odpravil ob ponovnem vzpostavljanju povezave.

Testna naprava	Selenium	Appium	Eggplant
Namizni računalnik	100%	/	100%
iOS naprava	/	100%	92%
iOS simulator	/	100%	96%
Android simulator	/	98%	88%
Povprečna uspešnost	100%	99,3%	94%

Tabela 4.5: Uspešnost testnih izvajanj v orodjih Selenium, Appium in Eggplant izražena v odstotkih.

Orodje Eggplant je doseglo povprečno zanesljivost 94%. Opazimo lahko, da se problemi z zanesljivostjo pojavijo na mobilnih napravah z operacijskimi sistemi Android in iOS. Glavni razlog za nezanesljivost pri simulatorju z operacijskim sistemom Android je bila nepravilna izvedba interakcije s testno napravo. Ob pregledu dnevnika smo ugotovili, da je bila v 4 primerih interakcija zabeležena v dnevnik, kljub temu pa se ni izvedla na testni napravi. Za preostala 2 primera napak smo ugotovili, da se je zgodil nepravilen izračun pozicije interakcije, ki je posledica animacij operacijskega sistema ob odpiranju aplikacij. Zaradi animacij je orodje Eggplant zaznalo element v času, ko se je aplikacija odpirala, zaradi česar je bila zaznana pozicija elementa nižje na zaslonu kot ob odprti aplikaciji. Pri simulatorju in pravi napravi z operacijskim sistemom iOS smo opazili enake probleme kot pri simulatorju z operacijskim sistemom Android. Prava naprava je imela probleme z nepravilno izvedbo interakcije, medtem ko je imel simulator težavo z animacijami in z napačnim izračunom pozicije interakcije.

Pri primerjavi zanesljivosti moramo opozoriti, da rezultati niso povsem zanesljivi, saj so bile napake naključne in niso vedno ponovljive. Naključne napake niso nujno posledica nepravilnega delovanja orodja, ampak so lahko tudi posledica nepravilnega delovanja testne naprave v času izvajanja testov. Zanesljivost rezultatov bi lahko izboljšali z večjim številom ponovitev izvajanj v daljšem časovnem obdobju, s čimer bi zmanjšali vpliv omenjenih naključnih

napak.

4.4 Povzetek primerjave

Orodja Selenium, Appium in Eggplant smo primerjali po postopku izdelave testnih skript, hitrosti izvajanja in zanesljivosti delovanja. Ugotovili smo, da ima vsako orodje svoje prednosti in slabosti. Prednosti in slabosti primerjanih orodij, ki smo jih ugotovili med primerjavo, so povzete v tabeli 4.6.

Iz primerjave lahko ugotovimo, kdaj je določeno orodje primerno za uporabo. Orodje Selenium je primernejše za uporabo, kadar želimo avtomatizirati spletne brskalnike na namiznih računalnikih in zahtevamo visoko hitrost izvajanja. Podobno velja za orodje Appium, ki je primerno za avtomatizacijo spletnih brskalnikov in aplikacij na mobilnih napravah. Zaradi hitrega in zanesljivega delovanja sta orodji primerni za večje projekte, kjer bo testiranje obsežnejše. Orodje Eggplant je zaradi preproste izdelave testnih skript primerno, kadar testne skripte izdelujejo osebe brez znanja programiranja. Ker iskanje elementov temelji na prepoznavanju slik, je orodje Eggplant primerno tudi za testiranje aplikacij, kjer zahtevamo preverjanje grafičnega izgleda testirane aplikacije.

Orodje	Prednosti	Slabosti
Selenium	• visoka hitrost izvajanja, • visoka zanesljivost delovanja, • ponovna uporaba testnih skript v drugačnih okoljih in orodju Appium, • podpora za odpiranje brskalnika in spletnega naslova, • odprtokodni projekt.	• ne podpira avtomatizacije mobilnih naprav, • za namestitev in izdelavo testov je potrebno znanje programiranja.
Appium	• visoka zanesljivost delovanja, • ponovna uporaba testnih skript v drugačnih okoljih in orodju Selenium, • podpora za odpiranje aplikacij in spletnega naslova, • odprtokodni projekt.	• nekoliko nižja hitrost izvajanja od orodja Selenium, • ne podpira avtomatizacije namiznih računalnikov, • za namestitev in izdelavo testov je potrebno znanje programiranja.
Eggplant	• enostavna izdelava testov, • za izdelavo testov ne potrebujemo znanja programiranja, • ponovna uporaba testnih skript v različnih okoljih, • omogoča avtomatizacijo namiznih računalnikov in mobilnih naprav, • omogoča preverjanje izgleda aplikacije.	• nižja hitrost izvajanja, • nižja zanesljivost delovanja na mobilnih napravah, • za vsako testno napravo moramo posneti pripadajoče referenčne slike.

Tabela 4.6: Povzetek primerjave orodij Selenium, Appium in Eggplant.

Poglavje 5

Sklepne ugotovitve

V diplomski nalogi smo se spoznali z avtomatizacijo testiranja programske opreme. Avtomatizacijo testiranja smo umestili v postopek testiranja in agilnega razvoja programske opreme. Izpostavili smo potrebo po avtomatizaciji testiranja in cilje, ki jih z avtomatizacijo želimo doseči. Opozorili smo na prednosti in slabosti avtomatizacije ter podali osnovne kriterije s pomočjo katerih lahko ugotovimo primernost testa za avtomatiziranje.

Tri izbrana orodja za avtomatizacijo testiranja (Selenium, Appium in Eggplant Functional) smo podrobneje preučili in predstavili njihovo delovanje. Za vsako orodje smo navedli funkcionalnosti, ki jih ponujajo, in našteli podprte programske jezike, v katerih lahko izdelujemo testne skripte. Našteli smo brskalnike in operacijske sisteme, kjer lahko teste izdelane v izbranih orodjih izvajamo.

Izdelali smo spletno aplikacijo, ki je bila predmet testiranja pri primerjavi izbranih orodij. Izdelana spletna aplikacija omogoča preverjanje osnovnih funkcionalnosti, ki jih potrebujemo za avtomatizacijo (vzpostavitev povezave z testno napravo, izbira elementa, preverjanje vrednosti, izvajanje akcij, vnos v vnosno polje). Določili smo kriterije, po katerih smo primerjali izbrana orodja na praktičnem primeru. Predstavili smo testni primer, določili vhodne vrednosti in pričakovane rezultate. Podrobno smo opisali in primerjali izdelavo testnih skript za podani testni primer. Izdelane testne skripte smo

uporabili za primerjavo hitrosti in zanesljivosti delovanja izbranih orodij. Pri primerjavi hitrosti izvajanja in zanesljivosti delovanja dopuščamo možnost drugačnih rezultatov, v kolikor bi za primerjavo uporabili obsežnejši testni primer ali povečali število izvajanj.

S pridobljenimi rezultati smo prispevali k razumljivosti delovanja izbranih orodij za avtomatizacijo testiranja. Ugotovili smo, da ima vsako orodje svoje prednosti in slabosti.

Orodje Selenium pri svojem delovanju uporablja DOM zgradbo uporabniškega vmesnika. S testno napravo komunicira preko WebDriver programskega vmesnika in gonilnikov, ki prejete ukaze izvajajo v brskalniku. Orodje podpira avtomatizacijo spletnih aplikacij v vseh najpogosteje uporabljenih brskalnikih na namiznih računalnikih. Prednost orodja Selenium pri izdelavi testov je, da omogoča izdelavo testov v več različnih programskih jezikih. Pri primerjavi hitrosti in zanesljivosti se je izkazal za najhitrejše in najbolj zanesljivo orodje. Slabost orodja je, da ne podpira avtomatizacije mobilnih naprav.

Orodje Appium z implementacijo podpore za avtomatizacijo mobilnih naprav rešuje slabost orodja Selenium. Orodje omogoča avtomatizacijo spletnih aplikacij v privzetih brskalnikih in avtomatizacijo domorodnih aplikacij. Teste izdelane v orodju Appium lahko zaganjamo na mobilnih napravah iOS in Android. Delovanje orodja je podobno delovanju orodja Selenium. Ugotovili smo, da skripte izdelane za orodje Selenium, lahko uporabimo tudi v orodju Appium. Orodja lahko brez večjih težav združimo, s čimer omogočimo izvajanje testov na namiznih računalnikih in mobilnih napravah. Pri primerjavi hitrosti se je izkazal za nekoliko počasnejše orodje v primerjavi z orodjem Selenium. Počasnejše izvajanje je predvsem posledica počasnejše vzpostavitve povezave s testno napravo. Zato dopuščamo možnost, da bi bili lahko rezultati na večjem testnem primeru drugačni, saj bi se zmanjšal vpliv časa za vzpostavitev povezave na rezultate. Neposredna primerjava z orodjem Selenium ni najprimernejša, saj ju nismo zaganjali na enakih testnih napravah. Iz primerjave z orodjem Eggplant, kjer smo teste zaganjali na enakih

napravah, pa lahko sklepamo, da je orodje Appium zanesljivejše in občutno hitrejše.

Orodje Eggplant za svoje delovanje uporablja prepoznavanje slik na zaslonu naprave. Med izdelavo testov zajamemo referenčne slike iskalnih elementov, ki jih v testni skripti uporabimo za iskanje elementov in interakcijo z napravo. Komunikacija s testno napravo poteka z uporabo protokola VNC, ki omogoča deljenje zaslona in upravljanje z napravo. Glavni prednosti orodja sta preprosta postavitev okolja in izdelava testnih skript s pomočjo grafičnega vmesnika, zaradi česar ne potrebujemo programerskega znanja. Kot glavno slabost pristopa prepoznavanja slik lahko izpostavimo dejstvo, da je za vsako testno napravo potrebno zajeti pripadajoče referenčne slike. Ob spremembah videza aplikacije je potrebno za vse testne naprave ročno posneti nove referenčne slike, iz česar lahko sklepamo, da so stroški vzdrževanja testnih skript veliki. Druga slabost, ki smo jo opazili, je počasnejše izvajanje testov in slabša zanesljivost. Iz rezultatov smo ugotovili, da je pristop prepoznavanja slik na zaslonu počasnejši od pristopa preiskovanja DOM strukture. Opazili smo tudi slabšo zanesljivost delovanja zlasti pri izvajanju testov na mobilnih napravah, ki je posledica nepravilnega delovanja orodja. Vseeno pa dopuščamo možnost drugačnih rezultatov, če bi za primerjavo izbrali drugačen testni primer.

Zaključimo lahko z ugotovitvijo, da sta orodji Selenium in Appium primernejši za večje projekte, kjer sta pomembni hitrost in zanesljivost izvajanja testov. Orodje Eggplant je zaradi enostavnosti in pristopa izdelave testnih skript primernejše, kadar bo teste izdelovala oseba brez znanja programiranja ali je v testnih zahtevah določeno preverjanje videza grafičnih elementov aplikacije.

Kljub doseženim ciljem diplomske naloge predlagamo nekaj izboljšav, ki bi pripomogle k boljši primerjavi orodij:

- V primerjavo bi vključili več orodij in več različnih testnih aplikacij.
- Orodja bi primerjali po hitrosti in zanesljivosti izvajanja ob vzporednem izvajanju testov na več testnih napravah.

- Za bolj realne scenarije bi bilo potrebno izdelati večje število testov na zahtevnejši aplikaciji in teste izvajati v daljšem časovnem obdobju. V obdobju testiranja bi dodajali spremembe, ki bi vplivale na delovanje aplikacije, s čimer bi lahko ocenili stroške vzdrževanja testnih skript in ugotovili učinkovitost orodij pri zaznavanju napak.

- S primerjavo učnih krivulj, bi primerjali zahtevnost učenja in izdelave testnih skript v posameznih orodjih. Rezultate bi pridobili z merjenjem časa, ki ga oseba brez predznanja potrebuje za učenje orodij in izdelavo testnih skript.

Priloge

A Izvorna koda

A.1 Razred DriverSingleton

```
from .device_platforms import DevicePlatform
from selenium import webdriver as selenium_webdriver
from appium import webdriver as appium_webdriver


class DriverSingleton:
    """
    Razred, ki poskrbi za inicializacijo WebDriverja
    in zagotavlja globalen dostop do ustvarjene instance

    """
    __driver = None

    @staticmethod
    def setup_driver(desired_capabilities: dict, platform: str, host: str, port: str):
        if DevicePlatform.is_mobile(platform):
            DriverSingleton.__driver = appium_webdriver.Remote(
                command_executor=DriverSingleton.get_command_executor(host, port),
                desired_capabilities=desired_capabilities
            )
        else:
            DriverSingleton.__driver = selenium_webdriver.Remote(
                command_executor=DriverSingleton.get_command_executor(host, port),
                desired_capabilities=desired_capabilities
            )
        DriverSingleton.__driver.implicitly_wait(15)
        return DriverSingleton.__driver

    @staticmethod
    def get_driver():
        return DriverSingleton.__driver

    @staticmethod
    def get_command_executor(host: str, port: str) -> str:
        return 'http://{}:{}/wd/hub'.format(host, port)
```

Izsek 1: Razred za inicializacijo in dostop do Selenium ali Appium WebDriverja.

A.2 Izdelani testi v orodjih Selenium in Appium

```
import unittest
import tests.selenium.config as CONFIG

from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait

from tests.selenium.helpers import login
from tests.selenium.device_platforms import DevicePlatform
from .driver_singleton import DriverSingleton


class TestLoginLogout(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        cls.driver = DriverSingleton.get_driver()

    def setUp(self):
        self.driver.get(CONFIG.BASE_URL)

    def _invalid_login(self, username, password):
        login(username, password, self.driver)
        error_message = self.driver.find_element_by_class_name('form-error')
        self.assertEqual(error_message.text, 'Your username or password is incorrect.
     Please try again.')

    def test_login_form_empty_fields(self):
        self.logout_if_logged_in()
        self._invalid_login('', '')

    def test_login_form_empty_password(self):
        self.logout_if_logged_in()
        self._invalid_login(CONFIG.LOGIN_CREDENTIALS_USERNAME, '')

    def test_login_form_empty_username(self):
        self.logout_if_logged_in()
        self._invalid_login('', CONFIG.LOGIN_CREDENTIALS_PASSWORD)

    def test_login_form_incorrect_username(self):
        self.logout_if_logged_in()
        self._invalid_login('incorrectUser', CONFIG.LOGIN_CREDENTIALS_PASSWORD)

    def test_login_form_incorrect_password(self):
```

```
        self.logout_if_logged_in()
        self._invalid_login(CONFIG.LOGIN_CREDENTIALS_USERNAME, 'incorrectPassword')

    def test_login_form_success(self):
        self.logout_if_logged_in()
        login(CONFIG.LOGIN_CREDENTIALS_USERNAME, CONFIG.LOGIN_CREDENTIALS_PASSWORD,
     self.driver)
        WebDriverWait(self.driver, 15).until(
            EC.visibility_of_element_located((By.CLASS_NAME, "main-navigation"))
        )
        self.assertTrue('Dashboard' in self.driver.title)

    def test_logout(self):
        WebDriverWait(self.driver, 15).until(
            EC.visibility_of_element_located((By.CLASS_NAME, "main-navigation"))
        )
        self.assertTrue('Dashboard' in self.driver.title)
        self.logout()

        WebDriverWait(self.driver, 15).until(
            EC.visibility_of_element_located((By.ID, "login-form"))
        )
        self.assertEqual(self.driver.title, 'Login')

        self.driver.get(CONFIG.BASE_URL)
        self.assertEqual(self.driver.title, 'Login')

    def logout(self):
        if DevicePlatform.is_mobile(self.driver.capabilities.get('platformName')):
            self.driver.find_element_by_id('mobile-navigation-button').click()

        self.driver.find_element_by_class_name('logout-button').click()
        WebDriverWait(self.driver, 15).until(
            EC.visibility_of_element_located((By.ID, "login-form"))
        )

    def logout_if_logged_in(self):
        if 'Dashboard' in self.driver.title:
            self.logout()
```

Izsek 2: Testi za testiranje prijavnega okna v orodjih Selenium in Appium.

A.3 Izdelani testi v orodju Eggplant

```
properties
    helpers: {
        "loginHelpers",
        folder() & "utils.suite/Scripts/evaluations",
        folder() & "utils.suite/Scripts/desktopHelpers",
        folder() & "utils.suite/Scripts/androidHelpers",
        folder() & "utils.suite/Scripts/iOSHelpers",
    }
end properties


to handle loginLogout platform, testApp

    Try
        // Odpiranje brskalnika in spletnega naslova
        setUp platform, testApp
    Catch exception
        LogError "Pri odpiranju brskalnika je prislo do napake" & exception
    End try

    if the exception is empty then
        Try
            // Zagon testov za prijavno okno
            executeLoginTests platform
            // Zagon testov za odjavo uporabnika
            executeLogoutTests platform
        Catch exception
            LogError "Pri izvajanju testov je prislo od napake: " & exception
        End try
    end if

    Try
        // Zapiranje brskalnika
        tearDown platform, testApp
    Catch exception
        LogWarning "Pri zapiranju brskalnika je prislo do napake: " & exception
    End try

end loginLogout


to handle executeLoginTests platform

    // Manjkajoce uporabnisko ime in geslo
```

```
    login "", ""
    evaluateImage "form_error", "Empty useraname and password validation", 10

    // Manjkajoce geslo
    login "test", ""
    evaluateImage "form_error", "Empty password validation", 10

    // Manjkajoce uporabnisko ime
    login "", "test1234"
    evaluateImage "form_error", "Empty username validation", 10

    // Nepravilni uporabnik
    login "incorrectUser", "test1234"
    evaluateImage "form_error", "Incorrect credentials validation", 10

    // Nepravilno geslo
    login "test", "incorrectPassword"
    evaluateImage "form_error", "Incorrect password validation", 10

    // Uspesna prijava
    login "test", "test1234"
    evaluateImage "dashboard_navigation", "Successful login validation", 10

end executeLoginTests


to handle executeLogoutTests platform

    if platform is in ("iOS", "Android") then
        // Navigacija na mobilnih napravah je skrita, zato jo je potrebno s klikom na gumb
     prikazati
        Click "dashboard_navigation"
    end if

    // Odjava uporabnika
    Click "button_logout"
    evaluateImage "form_login", "Successful logout validation", 10

end executeLogoutTests


to handle setUp platform, testApp

    set url to "192.168.1.110:8000"
    // Odpremo URL naslov v brskalniku
    if platform is "Android" then
        Connect "Nexus 5x"
```

```
        OpenChromeBrowser url
    else if platform is "iOS" then
        Connect "iPhone 7 - Simulator"
        // Connect "iPhone 7"
        OpenSafariBrowser url
    else if platform is "Desktop" then
        Connect "Macbook Pro"
        OpenBrowser testApp, url
    end if

end setUp


to handle tearDown platform, testApp

    // Zapremo aktivno okno brskalnika
    if platform is "Android" then
        CloseChromeBrowser
    else if platform is "iOS" then
        CloseSafariBrowser
    else if platform is "Desktop" then
        CloseBrowserWindow testApp
    end if

end tearDown
```

Izsek 3: Testi za testiranje prijavnega okna v orodju Eggplant.

Literatura

[1] Rajeev Gupta. *Agile Automation and Unified Functional Testing*. Pearson Education India, 2016.

[2] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. *Manifesto for agile software development. http://agilemanifesto.org*, 2001.

[3] *Systems development life cycle*. Dosegljivo: `https://en.wikipedia.org/wiki/Systems_development_life_cycle`. [Dostopano: 1. 3. 2018].

[4] Glenford J. Myers Tom Badgett, Corey Sandler. *The Art of Software Testing, 3rd Edition*. John Wiley & Sons, 2011.

[5] Mohd Ehmer Khan, Farmeena Khan, et al. *A comparative study of white box, black box and grey box testing techniques. Int. J. Adv. Comput. Sci. Appl*, 3(6), 2012.

[6] Kenneth S Rubin. *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley, 2012.

[7] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley professional, 2000.

[8] Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[9] Steve R Palmer and Mac Felsing. *A practical guide to feature-driven development*. Pearson Education, 2001.

[10] *Comparison of GUI testing tools*. Dosegljivo: `https://en.wikipedia.org/wiki/Comparison_of_GUI_testing_tools`. [Dostopano: 10. 3. 2018].

[11] *Selenium*. Dosegljivo: `https://www.seleniumhq.org`. [Dostopano: 10. 3. 2018].

[12] *Appium*. Dosegljivo: `http://appium.io/`. [Dostopano: 10. 3. 2018].

[13] *Eggplant*. Dosegljivo: `https://www.testplant.com/digital-automation-intelligence/`. [Dostopano: 10. 3. 2018].

[14] *Unified Functional Testing (UFT)*. Dosegljivo: `https://software.microfocus.com/fr-ca/products/unified-functional-automated-testing/overview`. [Dostopano: 10. 3. 2018].

[15] *Watir*. Dosegljivo: `http://watir.com`. [Dostopano: 10. 3. 2018].

[16] *Rational Functional Tester*. Dosegljivo: `https://www.ibm.com/us-en/marketplace/rational-functional-tester`. [Dostopano: 10. 3. 2018].

[17] *TestComplete*. Dosegljivo: `https://smartbear.com/product/testcomplete`. [Dostopano: 10. 3. 2018].

[18] *Tricentis*. Dosegljivo: `https://www.tricentis.com/software-testing-tools`. [Dostopano: 10. 3. 2018].

[19] *Ranorex*. Dosegljivo: `https://www.ranorex.com`. [Dostopano: 10. 3. 2018].

[20] *Robot Framework*. Dosegljivo: `http://robotframework.org`. [Dostopano: 10. 3. 2018].

[21] *SeleniumHQ/selenium: A browser automation framework and ecosystem.* Dosegljivo: `https://github.com/SeleniumHQ/selenium`. [Dostopano: 14. 2. 2018].

[22] *WebDriver*. Dosegljivo: `https://www.w3.org/TR/webdriver/`. [Dostopano: 11. 3. 2018].

[23] David Burns. *Selenium 2 Testing Tools Beginner's Guide*. Packt Publishing, 2012.

[24] *Javascript HTML DOM*. Dosegljivo: `https://www.w3schools.com/js/js_htmldom.asp`. [Dostopano: 14. 2. 2018].

[25] *Selenium Remote Control*. Dosegljivo: `https://www.seleniumhq.org/projects/remote-control/`. [Dostopano: 14. 2. 2018].

[26] *Selenium WebDriver*. Dosegljivo: `https://www.seleniumhq.org/projects/webdriver/`. [Dostopano: 14. 2. 2018].

[27] *Selenium IDE*. Dosegljivo: `https://www.seleniumhq.org/projects/ide/`. [Dostopano: 14. 2. 2018].

[28] *Selenium Grid*. Dosegljivo: `https://www.seleniumhq.org/docs/07_selenium_grid.jsp`. [Dostopano: 14. 2. 2018].

[29] *Brief History of The Selenium Project*. Dosegljivo: `https://docs.seleniumhq.org/docs/01_introducing_selenium.jsp`. [Dostopano: 14. 2. 2018].

[30] *Selenium IDE – Add-ons for Firefox*. Dosegljivo: `https://addons.mozilla.org/en-US/firefox/addon/selenium-ide/`. [Dostopano: 16. 2. 2018].

[31] *Sauce Labs*. Dosegljivo: `https://saucelabs.com`. [Dostopano: 10. 3. 2018].

[32] *Node.js*. Dosegljivo: `https://nodejs.org/en/`. [Dostopano: 16. 2. 2018].

[33] Gaurang Shah, Prayag Shah, and Rishikesh Muchhala. *Software testing automation using appium. International Journal of Current Engineering and Technology*, 4(5):3528–3531, 2014.

[34] Hans Manoj. *Appium Essentials*. Packt Publishing, 2015.

[35] *Appium Desktop*. Dosegljivo: `https://github.com/appium/appium-desktop`. [Dostopano: 16. 2. 2018].

[36] *User Interface Testing*. Dosegljivo: `https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/09-ui_testing.html`. [Dostopano: 17. 2. 2018].

[37] *appium/appium-xcuitest-driver: Appium iOS driver, backed by Apple XCUITest*. Dosegljivo: `https://github.com/appium/appium-xcuitest-driver`. [Dostopano: 17. 2. 2018].

[38] *XCUITest (iOS) - Appium*. Dosegljivo: `https://appium.io/docs/en/drivers/ios-xcuitest/`. [Dostopano: 17. 2. 2018].

[39] *appium/appium-uiautomator2-driver*. Dosegljivo: `https://github.com/appium/appium-uiautomator2-driver`. [Dostopano: 17. 2. 2018].

[40] *UI Automator — Android Developers*. Dosegljivo: `https://developer.android.com/training/testing/ui-automator.html`. [Dostopano: 17. 2. 2018].

[41] *The Appium Clients - Appium*. Dosegljivo: `https://appium.io/docs/en/about-appium/appium-clients/`. [Dostopano: 18. 2. 2018].

[42] *Home - TestPlant*. Dosegljivo: `https://www.testplant.com/`.

[43] *Eggplant Functional - Wikipedia*. Dosegljivo: https://en.wikipedia.org/wiki/Eggplant_Functional. [Dostopano: 19. 2. 2018].

[44] *Eggplant Functional - TestPlant*. Dosegljivo: https://www.testplant.com/products/eggplant-functional/. [Dostopano: 19. 2. 2018].

[45] *TRANSMISSION CONTROL PROTOCOL*. Dosegljivo: https://tools.ietf.org/html/rfc793. [Dostopano: 18. 2. 2018].

[46] *Virtual Network Computing*. Dosegljivo: https://en.wikipedia.org/wiki/Virtual_Network_Computing. [Dostopano: 18. 2. 2018].

[47] *Testplant Creates World's First Fusion Test Automation Engine - TestPlant*. Dosegljivo: https://www.testplant.com/2018/01/02/testplant-creates-worlds-first-fusion-test-automation-engine/. [Dostopano: 20. 2. 2018].

[48] *The Suite Window*. Dosegljivo: http://docs.testplant.com/ePF/gettingstarted/epf-suite-window.htm. [Dostopano: 19. 2. 2018].

[49] *About SenseTalk*. Dosegljivo: http://docs.testplant.com/ePF/SenseTalk/stk-about-sensetalk.htm. [Dostopano: 23. 2. 2018].

[50] *The Image Viewer*. Dosegljivo: http://docs.testplant.com/ePF/using/epf-image-viewer.htm. [Dostopano: 19. 2. 2018].

[51] *Android Gateway*. Dosegljivo: http://docs.testplant.com/ePF/using/epf-getting-started-android-gateway.htm. [Dostopano: 11. 3. 2018].

[52] *Android Gateway Settings*. Dosegljivo: http://docs.testplant.com/ePF/using/epf-android-gateway-settings.htm. [Dostopano: 11. 3. 2018].

[53] *iOS Gateway*. Dosegljivo: http://docs.testplant.com/ePF/using/epf-getting-started-ios-gateway.htm. [Dostopano: 11. 3. 2018].

[54] *Getting Started with Eggplant AI.* Dosegljivo: http://docs.testplant.com/EAI/eai-getting-started-eggplant-ai.htm. [Dostopano: 23. 2. 2018].

[55] *Linking Eggplant AI Models to Snippets.* Dosegljivo: http://docs.testplant.com/EAI/eai-snippets.htm. [Dostopano: 23. 3. 2018].

[56] *Getting Started - Appium, Session Initialization.* Dosegljivo: http://appium.io/docs/en/about-appium/getting-started/#session-initialization. [Dostopano: 10. 2. 2018].

[57] *google/ios-webkit-debug-proxy: A DevTools proxy (Chrome Remote Debugging Protocol) for iOS devices (Safari Remote Web Inspector).* Dosegljivo: https://github.com/google/ios-webkit-debug-proxy. [Dostopano: 16. 2. 2018].

[58] *Mobile Control and Touch Events.* Dosegljivo: http://docs.testplant.com/ePF/SenseTalk/stk-mobile-control-touch-events.htm#launchapp. [Dostopano: 12. 2. 2018].

[59] *Use Spotlight on your Mac.* Dosegljivo: https://support.apple.com/en-us/HT204014. [Dostopano: 10. 2. 2018].

[60] *Selenium-WebDriver API Commands and Operations.* Dosegljivo: http://www.seleniumhq.org/docs/03_webdriver.jsp. [Dostopano: 10. 2. 2018].

[61] *iOS Predicate Guide - Appium.* Dosegljivo: http://appium.io/docs/en/writing-running-appium/ios/ios-predicate/index.html. [Dostopano: 28. 2. 2018].

[62] *UiSelector — Android Developers.* Dosegljivo: https://developer.android.com/reference/android/support/test/uiautomator/UiSelector.html. [Dostopano: 28. 2. 2018].

[63] *Ajax - Web developer guides — MDN*. Dosegljivo: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX. [Dostopano: 28. 2. 2018].

[64] *5. Waits — Selenium Python Bindings 2 documentation*. Dosegljivo: http://selenium-python.readthedocs.io/waits.html. [Dostopano: 26. 2. 2018].

[65] *Working with Optical Character Recognition (OCR)*. Dosegljivo: http://docs.testplant.com/ePF/using/epf-working-with-optical-character-recognition-ocr.htm. [Dostopano: 16. 2. 2018].

[66] *Text-Image Generators*. Dosegljivo: http://docs.testplant.com/ePF/using/epf-working-with-text-image-generators-tigs.htm. [Dostopano: 16. 2. 2018].

[67] *selenium.webdriver.remote.webelement — Selenium 3.9 documentation*. Dosegljivo: https://seleniumhq.github.io/selenium/docs/api/py/webdriver_remote/selenium.webdriver.remote.webelement.html. [Dostopano: 26. 2. 2018].

[68] *selenium.webdriver.common.action_chains — Selenium 3.9 documentation*. Dosegljivo: https://seleniumhq.github.io/selenium/docs/api/py/webdriver/selenium.webdriver.common.action_chains.html. [Dostopano: 26. 2. 2018].

[69] *Touch Actions - Appium*. Dosegljivo: http://appium.io/docs/en/writing-running-appium/touch-actions/. [Dostopano: 25. 2. 2018].

[70] *Multi Touch Perform - Appium*. Dosegljivo: http://appium.io/docs/en/commands/interactions/touch/multi-touch-perform/. [Dostopano: 26. 2. 2018].

[71] *Mouse Events and Control*. Dosegljivo: http://docs.testplant.com/ePF/SenseTalk/stk-mouse-events-control.htm. [Dostopano: 12. 2. 2018].

[72] *Keyboard and Clipboard Events*. Dosegljivo: http://docs.testplant.com/ePF/SenseTalk/stk-keyboard-clipboard-events.htm. [Dostopano: 12. 2. 2018].

[73] *Mobile Control and Touch Events*. Dosegljivo: http://docs.testplant.com/ePF/SenseTalk/stk-mobile-control-touch-events.htm. [Dostopano: 12. 2. 2018].

[74] *Text-Reading Functions*. Dosegljivo: http://docs.testplant.com/ePF/SenseTalk/stk-ocr-text-reading-functions.htm?Highlight=readtext. [Dostopano: 1. 3. 2018].

[75] *Unit testing framework*. Dosegljivo: https://docs.python.org/3/library/unittest.html. [Dostopano: 16. 2. 2018].