Automatsko testiranje mobilnih uređaja s iOS i Android operacijskim sustavom

Vajak, Denis

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:658980

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK
Sveučilišni diplomski studij

AUTOMATSKO TESTIRANJE MOBILNIH UREĐAJA S IOS I ANDROID OPERACIJSKIM SUSTAVOM

Diplomski rad

Denis Vajak

Osijek, 2017.

SADRŽAJ

1. UVOD 1
2. PREGLED STANJA 2
2.1. Opis problema 2
2.2. Postojeća rješenja 2
3. PRIJEDLOG TESTNOG OKRUŽENJA 6
3.1. Razlike među mobilnim operacijskim sustavima 7
3.2. Postavljanje okruženja poslužitelja 7
3.3. Postavljanje okruženja klijenta 8
4. PRIMJERI I PISANJE FUNKCIONALNIH TESTOVA 12
4.1. Testiranje aplikacije: Deezer 12
4.2. Testiranje aplikacije: Dropbox 22
5. ZAKLJUČAK 29
LITERATURA 30
SAŽETAK 31
ABSTRACT 32
ŽIVOTOPIS 33

1. UVOD

U današnje vrijeme, razvoj mobilnih aplikacija je iznimno unosno tržište. Ponuda aplikacija je velika, ali je velika i potražnja. Time mnoge kompanije kreću u razvoj vlastitih aplikacija. Neke kompanije ulaze u igru s originalnim idejama, a neke rade na aplikacijama koje su slične onima koje već postoje, ali adaptirane za određenu upotrebu. Ovime nastaju velike količine programskog koda i velik broj raznih značajki. Sve ove značajke treba nekako testirati i provjeriti njihovu funkcionalnost. Ovo vrijedi i za mobilne aplikacije i za standardne računalne aplikacije.

Klasičan način testiranja aplikacija podrazumijeva testiranje aplikacije od strane osoba koje provjeravaju radi li aplikacija prema specifikacijama. Nažalost, ovaj postupak je iznimno spor i s obzirom na današnju dinamiku tržišta, često je nemoguće testirati sve značajke određene aplikacije na vrijeme prije puštanja u prodaju. Ručno testiranje je sporo, naporno i često neprecizno. Kako bi se riješio ovaj problem, raste popularnost za funkcionalnim automatskim testiranjem. Funkcionalno automatsko testiranje omogućuje oponašanje korisnika kako bi računalo testiralo sve funkcionalnosti određene aplikacije umjesto programera i dizajnera. Ovo značajno ubrzava postupak programiranja i smanjuje troškove zaposlenja ljudi koji bi ručno obavljali testove.

Danas postoji velik broj različitih alata za automatsko testiranje, neki su komercijalni, a neki su besplatni s otvorenim izvornim kodom. Ovaj diplomski rad se fokusira specifično na funkcionalno testiranje aplikacija. Analizirani su različiti alati za automatsko testiranje s različitih aspekata. Na temelju postavljenih zahtjeva odabran je najpogodniji alat na temelju kojega je izrađeno okruženje za automatsko testiranje Android i iOS operacijske sustave.

U radu je predstavljeno nekoliko testova kao demonstracija mogućnosti ovakvog okruženja. Testovi su napisani i za Android i za iOS operacijske sustave, koristeći iste aplikacije kao subjekte testiranja, kako bi se demonstriralo ključne razlike između ova dva operacijska sustava.

2. PREGLED STANJA

Automatsko testiranje je poznat problem i već postoje brojni alati koji omogućavaju automatsko testiranje. Od mobilnih aplikacija do web stranica, danas je moguće gotovo svaki dio programske podrške automatski testirati uz odgovarajuće alate.

2.1. Opis problema

U okviru radu potrebno je izraditi okruženje za funkcionalno automatsko testiranje Android i iOS mobilnih aplikacija. Zahtjevi koji se postavljaju za okruženje su: Okruženje mora koristiti što više alata otvorenog programskog koda, radi jednostavnosti i unifikacije Android i iOS operacijski sustavi se po mogućnosti trebaju izvršavati zajedno na istom računalu, testovi se moraju moći pozivati s udaljenog računala i na poslijetku cilj je testirati u okruženju crne kutije (engl. Black Box Testing) što znači da se aplikacija mora moći testirati bez njenog izvornog koda.

Za ovo je potrebno imati pristup i Android mobilnom uređaju i iOS mobilnom uređaju. S obzirom da iOS uređaji uglavnom ne funkcioniraju najbolje s Windows operacijskim sustavom, za ovo će biti potrebno Mac računalo s macOS operacijskim sustavom. To računalo mora biti glavno zaduženo za izvršavanje svih testova.

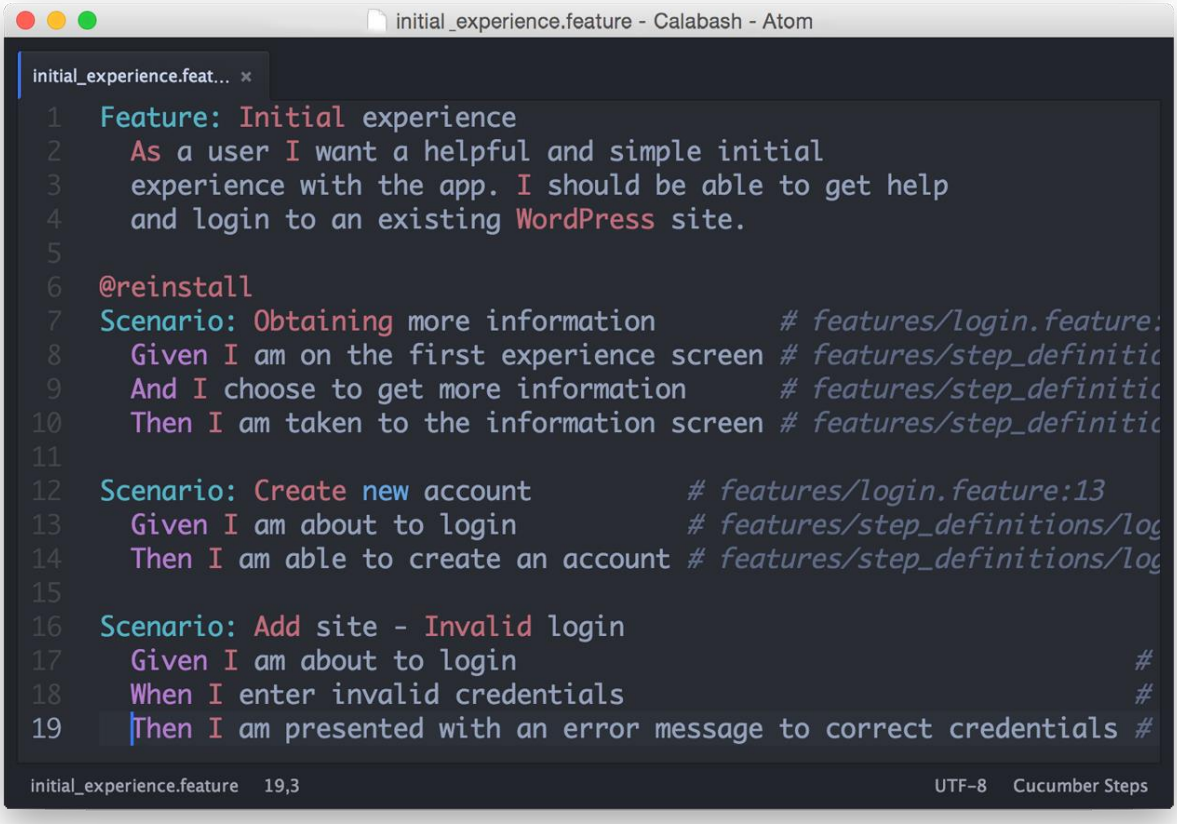
S obzuirom na postavljenje zahtjeve potrebno je odabrati najprikladniji alat koji već postoji, postaviti razvojno okruženje i raspored uređaja kao što je upravo opisano i pripremiti prilagođene biblioteke radi olakšanog korištenja alata za testiranje.

Naposljetku, treba provesti evaluaciju izrađenog okruženja tako što će biti napisano nekoliko pokaznih testova. Testovi moraju biti napisani i za Android i za iOS aplikacije kako bi se pokazala fleksibilnost okruženja.

2.2. Postojeća rješenja

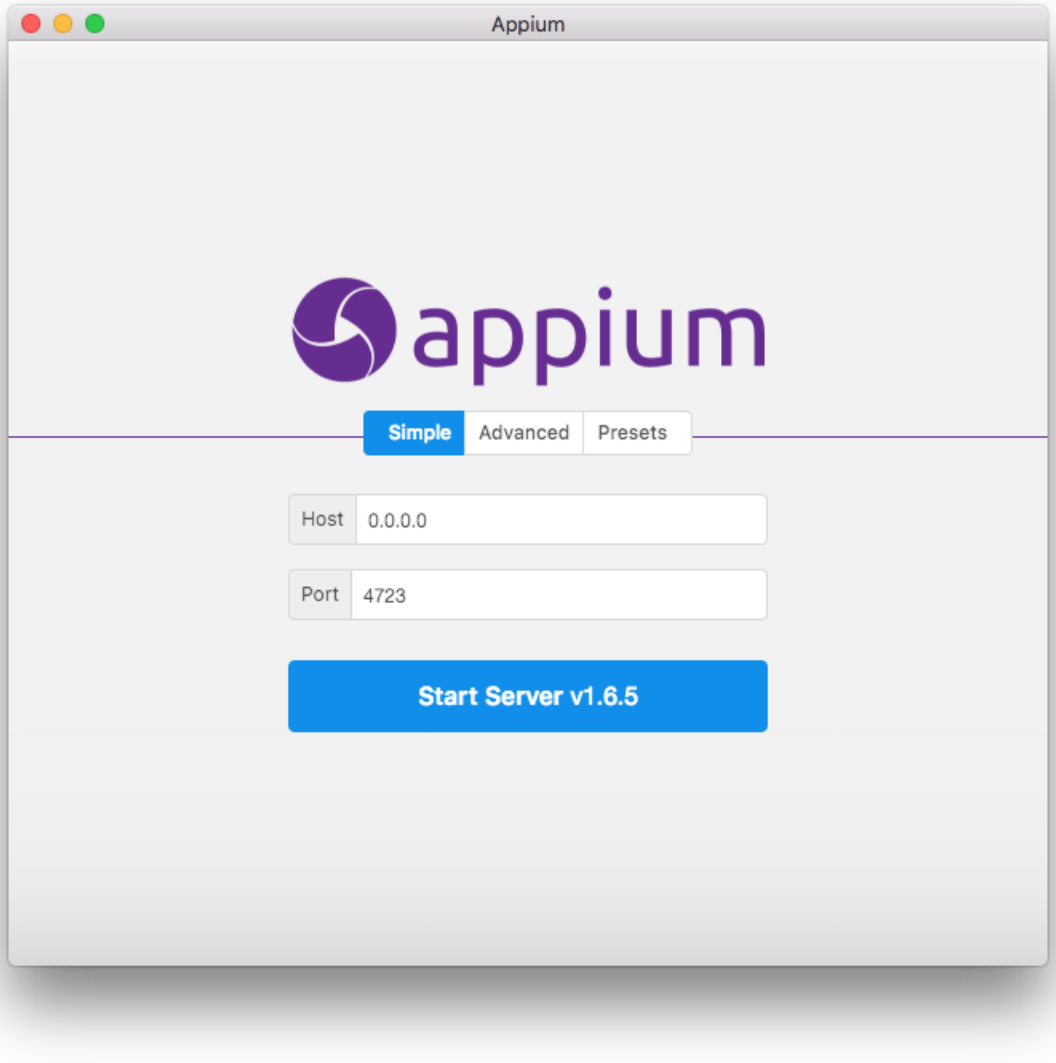
Kao što je ranije navedeno, automatsko testiranje je danas dosta popularna tema. Prema tome, postoji već velik broj alata koji ih omogućuju, pa je tako izbor mogućnosti dosta velik. Neki alati omogućuju testiranje samo Android aplikacija, neki samo iOS aplikacija, neki oba tipa. Jedno rješenje bi bilo pronaći dva alata i s njima postaviti okruženje, ali u idealnoj situaciji pronašao bi se alat koji može testirati oba u isto vrijeme.

Prvi takav alat je Calabash [1]. To je alat otvorenog izvornog koda koji može testirati i Android i iOS mobilne uređaje. Alat je baziran na Ruby programskom jeziku, a testne skripte se pišu u Cucumber jeziku. Cucumber je sličan prirodnom govoru pa je time lako razumljiv i netehničkom osoblju. Nažalost, ovakav programski jezik rezultira s puno ograničenja. Primjer Cucumber skipte je prikazan na slici 2.1. Kao što je vidljivo iz slike, klasično programiranje kao takvo ne postoji. Umjesto toga, testna skripta se piše prirodnim jezikom, što znači da su kompliciraniji algoritmi za testiranje teško izvedivi. Na svu sreću, testove je moguće pisati i s Ruby programskim jezikom, Cucumber nije obavezan. Pravi nedostatak alate je što je pri testiranju potrebno imati izvorni kod aplikacije i tijekom kompilacije ga je potrebno potpisati istim ključem koji koristi i Calabash tijekom instalacije vlastite telemetrije na mobilni uređaj.



Sl. 2.1. Primjer Cucumber skripte u Calabash-u.

Sljedeći alat je Appium [2]. Appium je alat baziran na Selenium-u. Selenium je otvorerno okruženjke za testiranje koje je dizajnirano primarno za testiranje internetskih preglednika. O Selenium-u je moguće više pročitati na [13]. Kako je Appium baziran na njemu, podržava prilično standardizirane pozive za testove. Appium je baziran na modelu poslužitelj-klijent, što znači da je sam Appium jednostavno poslužitelj koji osluškuje zahtjeve za izvršavanje testova i zatim ih proslijedi mobilnim uređajima. Zahvaljujući ovome, postoji velik broj različitih klijenata, za skoro pa i sve programske jezike, a u određenoj situaciji, moguće je i napraviti vlastiti klijent za slanje zahtjeva. Appium, isto kao i Calabash, podržava i Android i iOS mobilne uređaje, ali za testiranje nije potrebno imati izvorni kod. Među inačicama Appium-a, važno je za spomenuti Appium Desktop [3]. To je posebna inačica Appium-a koja dolazi s većinom potrebnih biblioteka kako bi se Appium poslužitelj mogao neometano izvršavati. Slika 2.2. prikazuje početni zaslon Appium Desktop-a, prije pokretanja poslužitelja.



Sl. 2.2. Početni zaslon Appium Desktop-a.

Ova dva alata su generalno najpopularniji. Osim njih postoji još veći broj različitih alata, ali u ovom radu neće biti svi pokriveni. Tako postoji Frank [4], alat za testiranje isključivo iOS mobilnih uređaja. Frank je sličan Calabash-u, u smislu da koristi Cucumber jezik za pisanje skripta. Činjenica da može testirati samo iOS aplikacije ga čini jako ograničenim i neprimamljivim. Isto tako postoji i Robotium [5], koji omogućuje testiranje Android mobilnih aplikacija koristeći Java-u kao programski jezik za testove. Slično kao i Frank, ovo ga čini neprimamljivim jer je ograničen na samo jedan mobilni operacijski sustav.

U tablici 2.1., ova četiri popularna alata su uspoređena prema mogućnostima koje su najvažnije za potrebe ovog okruženja. Iz tablice je vidljivo da najviše mogućnosti zadovoljava Appium, ne samo po osnovnim mogućnostima kao što su različiti operacijski sustavi, poslužitelj-klijent model, i tako dalje, nego i po broju podržanih jezika. S obzirom na njegov poslužitelj-klijent model i činjenice da koristi Selenium, u teoriji je moguće koristiti bilo koji programski jezik za pisanje testnih skripta, ako se za isti jezik razvije kompatibilni klijent. Prema tome, za potrebe izrade testnog okruženja, izabran je upravo Appium kao alat za funkcionalno automatsko testiranje.

Tab. 2.1. Usporedba alata za testiranje.

Mogućnost	**Calabash**	**Appium**	**Frank**	**Robotium**
Otvorenog izvornog koda	+	+	+	+
Android	+	+	-	+
iOS	+	+	+	-
Izvorni kod aplikacije nije potreban	-	+	-	+
Poslužitelj-klijent model	+	+	-	-
Programski jezici	Ruby Cucumber	Ruby Python Java JavaScript Objective C PHP C# itd…	Cucumber	Java

3. PRIJEDLOG TESTNOG OKRUŽENJA

Kao alat za testiranje je odabran Appium. Ova odluka je donešena na temelju činjenice da Appium ima veliku podršku u programerskoj zajednici, ima dobru i detaljno razrađenu dokumentaciju, a također je i popularan i poznat. Osim ovoga, Appium može testirati aplikacije i na iOS i na Android operacijskim sustavima što je upravo ono što će biti potrebno za ostvarenje zadatka. Appium također koristi poslužitelj-klijent model što omogućuje izbor velikog broja različitih klijenata.

Arhitektura testnog okruženja se sastoji od Mac računala na kojem su spojeni fizički Android i iOS mobilni uređaji. Na istoj mreži na kojoj se nalazi Mac računalo se nalazi još jedno računalo proizvoljnog operacijskog sustava, ali u ovom slučaju Windows. Na njemu se izvršava Appium klijent. Klijent se povezuje preko mreže s Mac računalom i šalje upute Appium poslužitelju za izvršavanje testa. Poslužitelj izvršava testove na fizičkim mobilnim uređajima i odgovara klijentu s rezultatima. Na slici 3.1. je prikazan dijagram ovog okruženja.



Sl. 3.1. Dijagram testnog okruženja.

Kao Appium poslužitelj odabran je Appium Desktop. Ovo je gotovo programsko rješenje koje u sebi sadrži sam Appium poslužitelj, klijent za proučavanje sučelja aplikacije i većinu potrebnih biblioteka.

Kao klijent odabran je Appium Python klijent [6]. Ovaj klijent je baziran na Python programskom jeziku. Python je iznimno jednostavan ali i moćan jezik, pa je savršen izbor za ovu situaciju. Pomoću njega je moguće pisati jednostavne skripte, kao u ovom slučaju, testove za mobilne aplikacije.

3.1. Razlike među mobilnim operacijskim sustavima

Android i iOS mobilni operacijski sustavi imaju velik broj razlika. Od dizajna do arhitekture operacijskog sustava. Zbog ovoga, situacija nije jednaka testira li se aplikacija na jednom ili drugom operacijskom sustavu.

Appium omogućuje testiranje Android aplikacija bez ikakvih problema i sve lagano funkcionira. Kod iOS-a je malo drugačija priča. Kako bi cijelo okruženje što bolje radilo, potrebno je nabaviti Apple Developer certifikat. Certifikat je moguće nabaviti na [7]. Bez navedenog certifikata gotovo je nemoguće efikasno testirati iOS aplikacije.

Druga značajna razlika je u samom Appium-u. Dok Appium omogućuje testiranje aplikacija na oba operacijska sustava, i dalje postoje razlike među naredbama jer one nisu unificirane. Ovo znači da je nemoguće pokrenuti i iOS i Android test koristeći istu skriptu, nego skripte trebaju biti prilagođene.

Pri pokretanju testa, Appium koristi podatkovni paket koji se naziva popis mogućnosti. U njemu je definirano mnoštvo različitih podataka koji su potrebni Appium poslužitelju kako bi znao kako započeti i naposljetku izvršiti test. Ovaj popis se znatno razlikuje između iOS i Android operacijskih sustava, ali uglavnom sadrži definiciju i verziju operacijskog sustava, ime uređaja za testiranje, identifikaciju paketa aplikacije koja će biti pokrenuta i još neke dodatne parametre.

3.2. Postavljanje okruženja poslužitelja

Operacijski sustav računala na kojem će se izvršavati Appium poslužitelj mora biti macOS. Razlog ovome je što je nemoguće testirati aplikacije za iOS uređaje na ijednom drugom operacijskom sustavu. Postupak postavljanja okruženja nije previše kompliciran, ali ima nekoliko koraka.

Prvi korak je preuzimanje i instaliranje Appium Desktop aplikacije. Appium Desktop je izdanje Appiuma koje u sebi sadrži Appium poslužitelj, alat za proučavanje korisničkog sučelja aplikacije i većinu potrebnih biblioteka za pokretanje cijelog Appium okruženja. Moguće ga je preuzeti i instalirati sa službene GitHub stranice projekta [3].

Nakon što je Appium Desktop instaliran, potrebno je instalirati još nekoliko dodatnih aplikacija potrebnih kako bi okruženje točno radilo. Prvi takav alat je Xcode o kojemu više piše na [8]. Xcode je okruženje za programiranje koje je razvio Apple, a potrebno je zbog prevođenja

programskog koda i alata koji omogućuju testiranje aplikacija na iOS uređajima. Xcode se može preuzeti i instalirati putem službenog AppStore-a za macOS.

Druga potrebna aplikacija je Android Studio kojeg je moguće preuzeti na [9]. Android Studio je programsko okruženje za programiranje i dizajn Android aplikacija. Dolazi s alatima za izvođenje testova na Android operacijskim sustavima pa je također neophodan za Appium testno okruženje ako je cilj testirati Android uređaje.

Osim ove dvije aplikacije, potrebno je još i instalirati dodatne aplikacije koje su Appium-u neophodne za funkcioniranje, ali ne dolaze s njim. Za ovo je prvo potrebno instalirati Homebrew o kojemu više piše na [10]. S obzirom da se macOS razlikuje od linuxa u nekim aspektima, ne dolazi sa standardnim rukovodiocem paketa. Instalacijom Homebrew-a, omogućuje se pristup standardnim linux paketima. Kako bi se Homebrew instalirao, potrebno je slijediti upute sa službene stranice. Kad je Homebrew instaliran, potrebno je instalirati dodatne aplikacije, a to su *libimobiledevice* i *ios-deploy*. Sada se obje mogu instatlirati koristeći *brew install* naredbu u komandnoj liniji.

S obzirom da će biti testirani fizički mobilni uređaji, potrebno je s poslužiteljem povezati i te iste uređaje. Za ovo će biti korišten jedan iOS mobilni uređaj i jedan Android mobilni uređaj, oba povezani s Mac računalom putem USB kabela. Ako su ovako povezani, Appium može na njima istovremeno izvoditi testove.

3.3. Postavljanje okruženja klijenta

Računalo s Windows operacijskim sustaovm korišteno je kao klijentsko računalo. Ovo nije toliko važno jer klijent može uspješno raditi na bilo kojem operacijskom sustavu jer se zasniva na Python programskom jeziku. Ovdje je potrebno instalirati dvije stvari: Python i Appium Python klijent.

Python je moguće instalirati sa službene stranice [11], pri čemu će biti korišten Python 2. Python 2 ima veću podršku nego Python 3 i Appium klijent s njim uspješno radi. Potrebno je preuzeti 64-bitnu instalaciju za Windows operacijski sustav. Appium Python klijent će biti korišten na klijentskoj strani jer je jednostavan i dobro dokumentiran na [6]. Kad je Python instaliran, instalacija Appium Python klijenta je jednostavna. Dovoljno je pokrenuti naredbu *pip install Appium-Python-Client* u komandnoj liniji.

Ovime je postavljeno cjelokupno okruženje za testiranje i na poslužiteljskoj i na klijentskoj strani. Ovakvo će okruženje raditi, ali ne najefikasnije. Kako bi se olakšalo pisanje

testova, razvijene su dvije Python biblioteke koje omogućuju olakšan pristup najčešće korištenim pozivima pri testiranju. Ove dvije biblioteke su nazvane *appium_ios.py* i *appium_android.py*.

Biblioteka za testiranje iOS aplikacija, *appium_ios.py*, za početak sadrži metode za pretragu elemenata. Postoje metode za pretragu po: oznaci, imenu, tipu, predikatu i XPath-u. Sve ove metode dolaze u dva oblika. Prvi tip vraća samo prvi element koji pronađe po danom kriteriju i usput provjerava postoji li taj element. Ako element ne postoji, odmah se uzrokuje pad testa. Drugi tip vraća polje svih pronađenih elemenata. Ovo polje može biti prazno ako nije pronađen nijedan element jer ove metode ne provjeravaju postojanje. Dok su pretrage po oznaci, imenu i tipu slične po načinu kako traže, pretrage po predikatu i XPath-u su drugačije. Pretraga po predikatu omogućuje korištenje logičkih operacija kako bi se pronašao element koji je na primjer nevidljiv i ima određenu kombinaciju znakova u imenu. Pretraga po XPath-u prima kao parametar takozvani XPath. XPath je unikatna putanja do određenog elementa i pomoću nje je moguće dohvatiti bilo koji element, čak i ako mu nedostaju svi drugi podaci. Ovaj tip pretrage se doduše ne savjetuje jer je jako lagano poremetiti test s najmanjom promjenom u korisničkom sučelju aplikacije i pretraga traje značajno duže nego kod ostalih tipova.

Osim pretrage, biblioteka nudi metode za: vertikalno pomicanje zaslona, pritiskanje na bilo koju točku na zaslonu po koordinatama, spremanje trenutne slike zaslona mobilnog uređaja i naposljetku metodu za pokretanje svih testova. Ovo naravno nisu sve funkcionalnosti Appium-a, ali su neke od najčešćih. Korištenjem ove biblioteke se značajno smanjuje programski kod testne skripte i čini ga se puno čitljivijim. U tablici 3.1. je prikazan popis svih dostupnih metoda u ovoj biblioteci.

Tab. 3.1. Popis metoda u iOS biblioteci.

Pretraga elementa/elemenata
Pretraga jednog elementa po oznaci
Pretraga više elemenata po oznaci
Pretraga jednog elementa po imenu
Pretraga više elemenata po imenu
Pretraga jednog elementa po tipu
Pretraga više elemenata po tipu
Pretraga jednog elementa po predikatu
Pretraga više elemenata po predikatu
Pretraga jednog elementa po XPath-u
Pretraga više elemenata po XPath-u

Dodatne metode
Vertikalno pomicanje zaslona
Pritiskanje na točku zaslona
Spremanje trenutne slike zaslona
Pokretanje svih testova

Biblioteka za testiranje Android aplikacija, *appium_android.py*, svojom je strukturom podosta slična iOS biblioteci, ali postoje i važne razlike. Tako na početku ima metode za pronalaženje elemenata, ali u ovom slučaju po: tekstu, opisu, identifikaciji i klasi. Isto kao i u drugoj biblioteci, ovdje postoje metode za pretragu jednog elementa i više njih. Pretrage po predikatu i po XPath-u ne postoje jer je to jednostavno ograničenje Android-a. Zbog ovoga nije lagano pronaći bilo koji element i ponekad je potrebno ručno pretraživati polja od više elemenata.

Osim pretrage, postoje i metode za pritiskanje fizičkih gumba Android uređaja. Tako postoje metode za sljedeće gumbe: gumb za povratak natrag, gumb za početni zaslon, gumb za meni i gumb za pregled aplikacija. Uz metode za pritisak gumba, u biblioteci su dodane i iste metode koje postoje i u iOS biblioteci, a to su: vertikalno pomicanje zaslona, pritiskanje na bilo koju točku na zaslonu po koordinatama, spremanje trenutne slike zaslona mobilnog uređaja i naposljetku metoda za pokretanje svih testova. U tablici 3.2. je prikazan popis svih dostupnih metoda u ovoj biblioteci.

Tab. 3.2. Popis metoda u Android biblioteci.

Pretraga elementa/elemenata
Pretraga jednog elementa po teksu
Pretraga više elemenata po tekstu
Pretraga jednog elementa po opisu
Pretraga više elemenata po opisu
Pretraga jednog elementa po identifikaciji
Pretraga više elemenata po identifikaciji
Pretraga jednog elementa po klasi
Pretraga više elemenata po klasi

Pritiskanje gumba
Gumb za povratak natrag
Gumb za početni zaslon
Gumb za meni
Gumb za pregled aplikacija

Dodatne metode
Vertikalno pomicanje zaslona
Pritiskanje na točku zaslona
Spremanje trenutne slike zaslona
Pokretanje svih testova

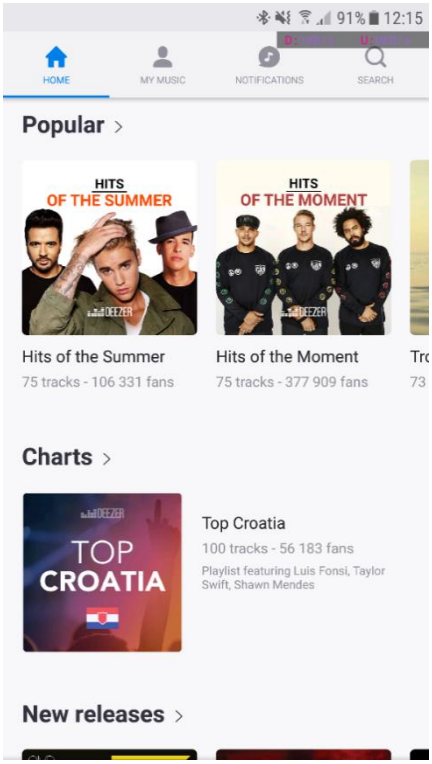
Kako bi se pokrenuli testovi, potrebno je na oba operacijska sustava saznati određene podatke aplikacije. Android traži identifikacijsku oznaku paketa i ime aktivnosti. Ovaj podatak se može dobiti tako da se pokrene aplikacija i brzo pošalje sljedeća naredba u komandnu liniju na računalu: *adb shell dumpsys window windows | grep –E 'mCurrentFocus|mFocusedApp'*. Kod iOS-a je malo drugačija situacija. Ovdje je potrebna samo identifikacijska oznaka paketa, ali ju je nemoguće dohvatiti kao na Android-u. Za ovu potrebu postoji internetska stranica „Bundle Id Finder“ kojoj je moguće pristupiti na [12].

4. PRIMJERI I PISANJE FUNKCIONALNIH TESTOVA

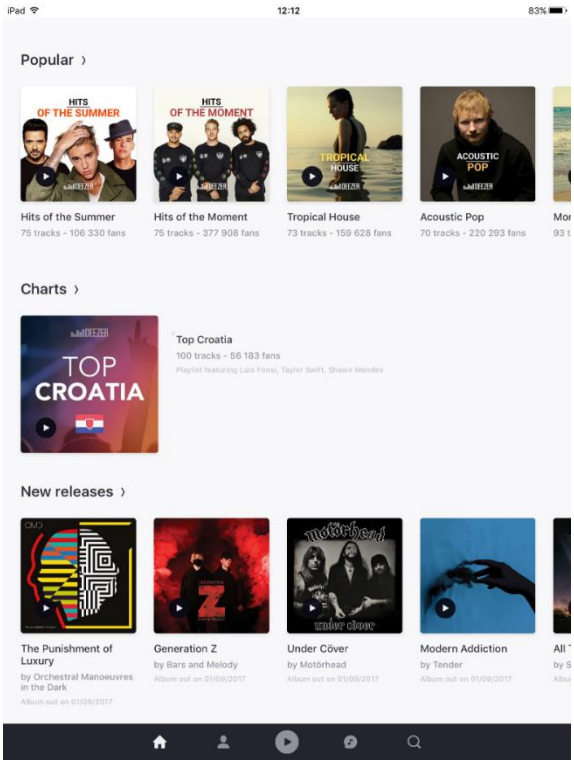
S postavljenim razvojnim okruženjem i pomoću napisanih biblioteke koje su opisane u prethodnom poglavlju, pisanje testova je relativno lagano. Radi istraživanja i usporedbe, testirale su se iste aplikacije i na Android i na iOS mobilnim uređajima. Za izvršavanje testova, Android uređaj koji je korišten je Samsung Galaxy S6 Edge, a iOS uređaj je Apple iPad Air 2.

4.1. Testiranje aplikacije: Deezer

Aplikacija koja je odabrana za početak je Deezer. Deezer je usluga i aplikacija koja omogućuje slušanje glazbe preko interneta bez trajnog preuzimanja sadržaja. Kao i sve druge aplikacije, moguće ju je preuzeti preko App Store-a na iOS-u i Google Play-a na Android-u. Glavno korisničko sučelje obje aplikacije nakon prijave je prikazano na slikama 4.1. i 4.2. za oba operacijska sustava. S obzirom da su korišteni uređaji za testiranje mobilni uređaj i tablet, i to još različitih operacijskih sustava, korisničko sučelje se značajno razlikuje. Zbog ovoga će testovi biti različiti u više pogleda.



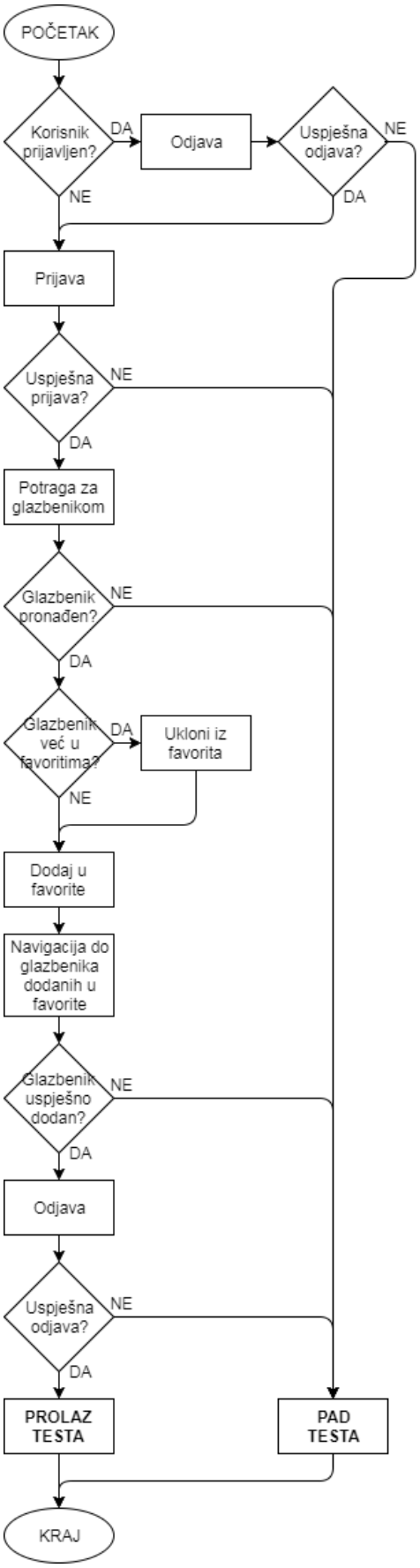
Sl. 4.1. Korisničko sučelje Deezer aplikacije na Android-u.



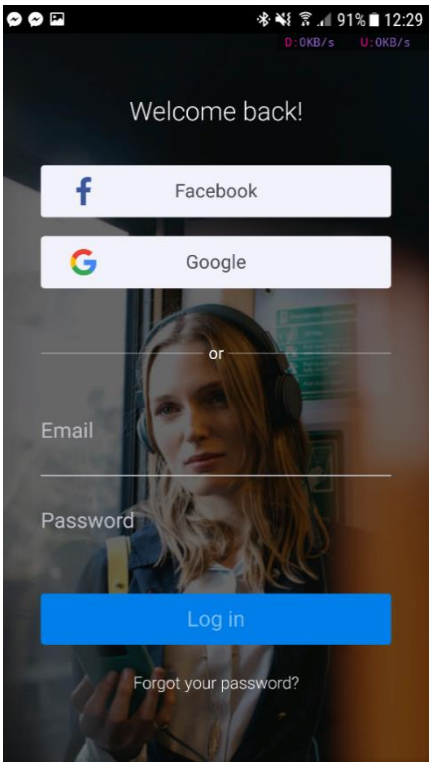
Sl. 4.2. Korisničko sučelje Deezer aplikacije na iOS-u.

Cilj ovoga testa će biti automatiziranje prijave u aplikaciju, navigacija do određenog glazbenika i dodavanje u favorite. Zatim, test mora provjeriti je li glazbenik uspješno dodan u favorite i naposljetku se odjaviti. Na slici 4.3. osmišljen je dijagram toka ovog testiranja aplikacije. Svaki blok se u stvarnosti sastoji od više manjih akcija kako bi test bio uspješno izveden, oslanjajući se na napisane biblioteke predstavljene u prethodnom poglavlju. Provjere uspješnosti testa su u dijagramu prikazane uglavnom samo simbolično kako bi dočarale da će se test srušiti tijekom svakog od blokova ako nešto nije kako bi trebalo biti.

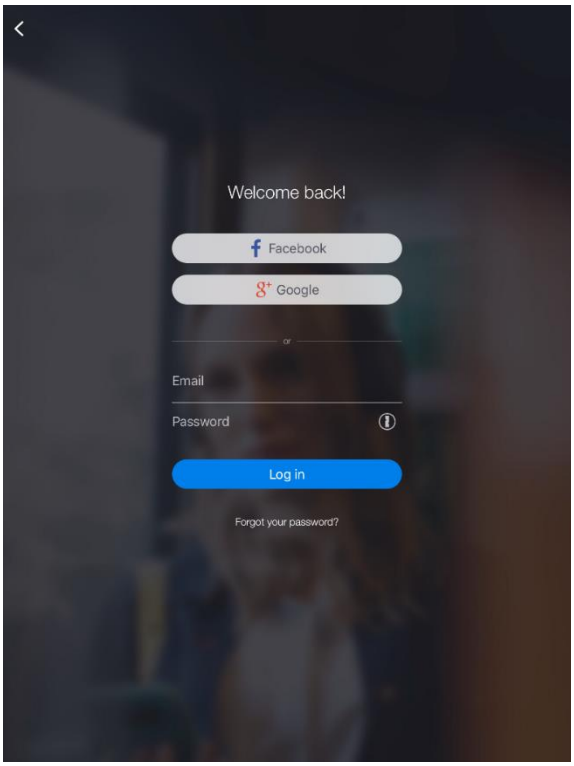
Na početku testa, potrebno je provjeriti je li korisnik prijavljen. Ovo se može provjeriti tako da se provjeri postojanje elementa gumba za prijavu što se jednostavno može ostvariti korištenjem ranije stvorenih biblioteka. Na slikama 4.4. i 4.5. je prikazan izgled zaslona za prijavu. Najkarakterističniji detalj je gumb za prijavu, pa je tako moguće koristiti pretragu za elementom koji ima tekst „Log in“ na Android-u, ili ime „Log in“ na iOS-u. U ovom slučaju se koristi pretraga za više elemenata kako test ne bi pao ako element ne postoji. Ako je vraćeno polje prazno, poziva se procedura za odjavu jer to znači da je korisnik već prijavljen.



Sl. 4.3. Dijagram toka Deezer testa.



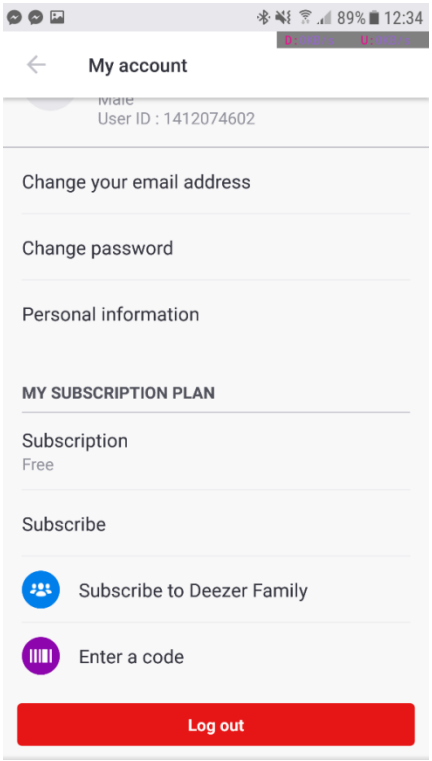
Sl. 4.4. Prijava u Deezer na Android-u.



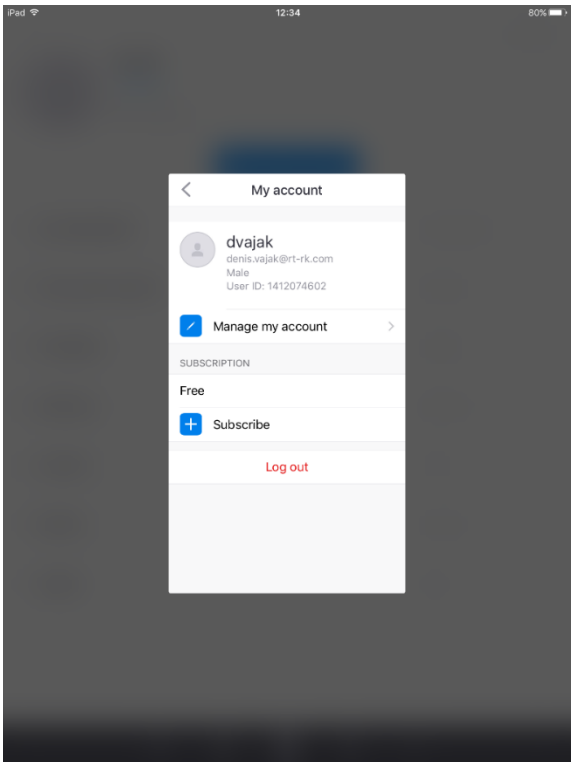
Sl. 4.5. Prijava u Deezer na iOS-u.

Koristeći veći broj poziva istih metoda, skripta navigira kroz korisničko sučelje dok ne dođe do gumba za odjavu. Na slikama 4.6 i 4.7. prikazan je izgled i lokacija tog gumba. Nalazi se u postavkama korisničkog računa. Način da se do njega dođe je različit između ove dvije verzije aplikacije, pa se time ovdje i testovi razlikuju. Kad je gumb dosegnut, test odjavljuje korisnika i, ako je sve uspješno prošlo, započinje proceduru prijave.

Procedura za prijavu se sastoji od traženja polja za unos adrese elektroničke pošte i zaporke. Na Android-u polje za elektroničku poštu je moguće pronaći koristeći pretragu po tekstu „Email“, dok se na iOS-u ovo može ostvariti pretragom po imenu „email“. Pretraga za poljem zaporke je nešto drugačija. Na Android-u se može pronaći po identifikaciji „deezer.android.app:id/password_input_edit“, dok se na iOS-u može pronaći po tipu „XCUIElementTypeSecureTextField“. Kad su polja pronađena, pozivaju se Appium-ove metode za unos teksta kako bi se upisale odgovarajuće vrijednosti. Naposljetku, pronalazi se gumb za prijavu i pritišće ga se.



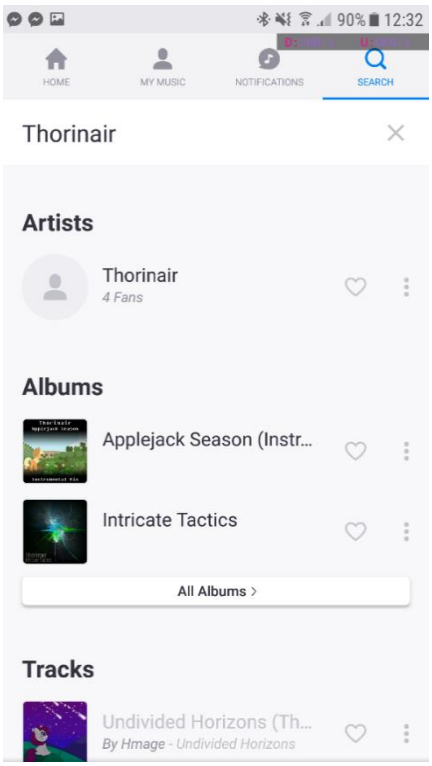
Sl. 4.6. Odjava iz Deezer aplikacije na Android-u.



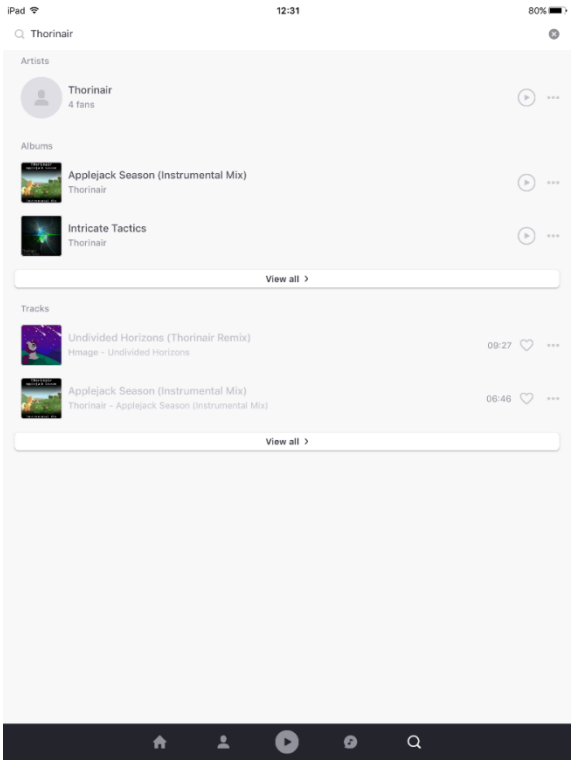
Sl. 4.7. Odjava iz Deezer aplikacije na iOS-u.

Nakon što je prijava uspjela, potrebno je dodati naredbu za čekanje. Ovo je važnan korak zato što ovisno o mreži na kojoj se uređaj nalazi, moguće je da će dohvaćanje Deezer početnog zaslona dulje trajati. Nakon što se zaslon učita, vrši se navigacija do pretrage. Ovo se značajno razlikuje između Android i iOS aplikacije jer je i korisničko sučelje značajno drugačije, kao što je vidljivo na slikama 4.1. i 4.2. Tako je na Android-u potrebno potrebno pronaći četvrti element s klasom „android.support.v7.app.ActionBar$Tab“ što odgovara čertvrtom gumbu na meniju na vrhu zaslona. Zatim je potrebno pronaći tekstualno polje za pretragu po tekstu „Search for music“ i unijeti ime glazbenika. Na iOS-u, navigacija je jednostavnija. Kako bi se došlo do stranice za pretragu, dovoljno je pronaći gumb po imenu „Search“. Tekstno polje se može pronaći po imenu „Search for an artist, track, playlist...“ i zatim se unosi ime glazbenika.

Glazbenik se smatra pronađenim ako se pojavljuje među rezultatima pretrage kao što je prikazano na slikama 4.8. i 4.9. S obzirom da pretraga nije trenutna, ovdje je opet potrebno dodati čekanje. Na Android-u se vrši provjera postojanja glazbenika tako da se provjerava postojanje elementa s opisom „Artists result“. Na iOS-u se jednostavno pretražuje element s imenom glazbenika.



Sl. 4.8. Pretraga za glazbenikom na Android-u.



Sl. 4.9. Pretraga za glazbenikom na iOS-u.

S obzirom da je cilj provjera uspješnosti dodavanja glazbenika u favorite, prvo je potrebno provjeriti je li glazbenik dodan u favorite i ako jest, ukloniti ga. Za ovo je prvo potrebno ući na stranicu glazbenika tako da se pritisnu ranije pronađeni elementi rezultata pretrage. Postupak dodavanja u favorite se razlikuje između Android i iOS aplikacije.

Na Android-u je postupak jednostavniji jer će glazbenik koji nije dodan u favorite imati element s opisom „Add to Favorites“ na svojoj stranici. Tako se provjerava postojanje ovog elementa i ako ne postoji, glazbenik je već dodan i potrebno ga je ukloniti iz favorita. Ovo se postiže pretragom za elementom koji ima opis „Remove from favorites“, pritiskom na njega i zatim pritiskom na gumb „OK“ u u dijalogu za potvrdu. Kad je glazbenik uklonjen, moguće ga je normalno dodati u favorite pretragom za elementom s opisom „Add to Favorites“ i njegovim pritiskom.

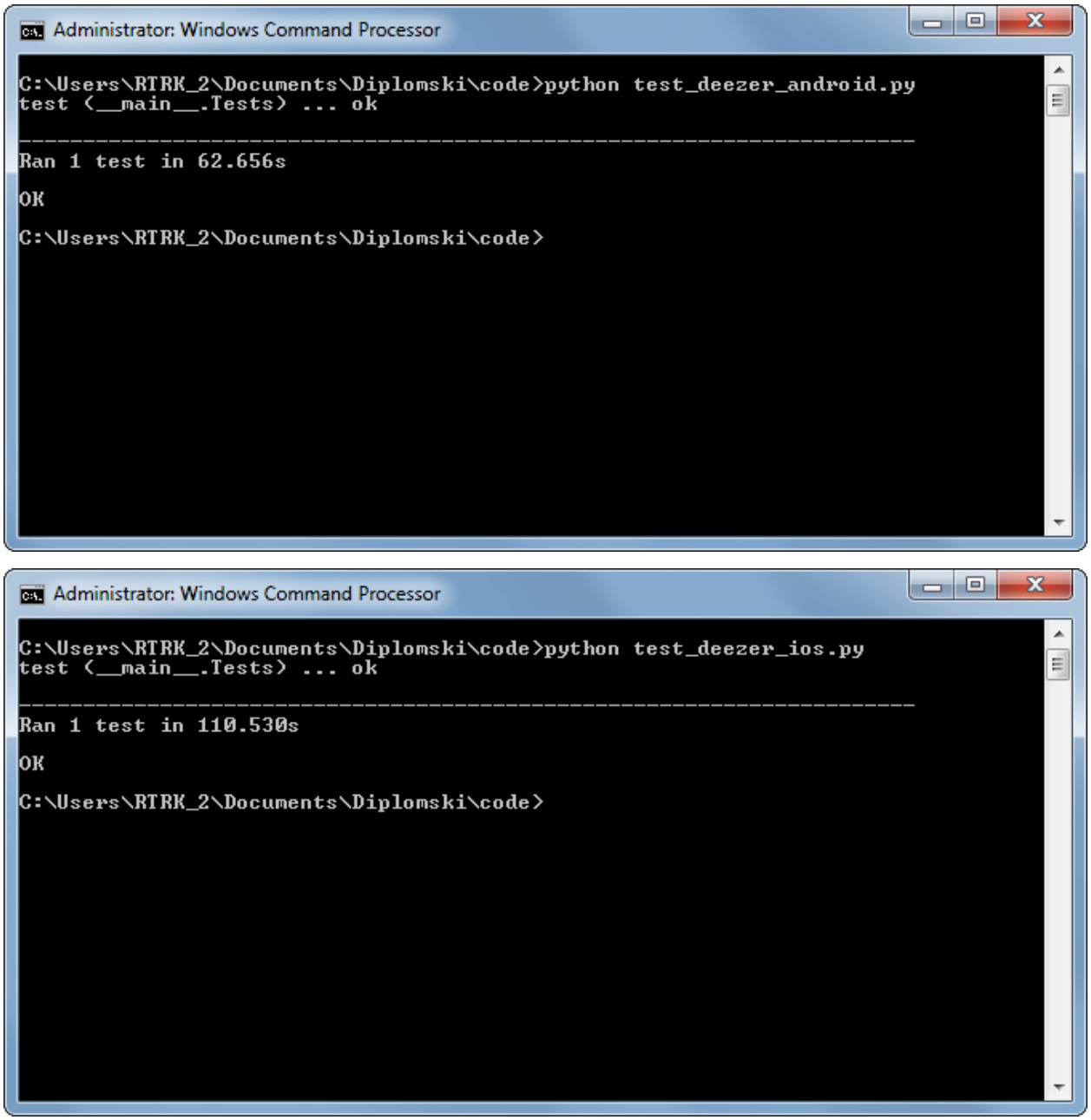
Kod iOS-a, ovi elementi nažalost nemaju opis i isti su bez obzira na stanje dodanosti glazbenika u favorite. Međutim, jedan od parametara se mijenja. Ako je glazbenik dodan u favorite, vrijednost se mijenja u 1, a u suprotnom ima nedefiniranu vrijednost. Kako bi se element pronašao, pretražuje se po svom XPath-u. Zatim se provjerava vrijednost. Ako je vrijednost 1, glazbenik se uklanja iz favorita tako da se na element vrši pritiskanje. Zatim se normalno opet pritišće na element kako bi se dodao u favorite.

Zadnji korak testa je navigirati do stranice favorita korisnika i provjeriti pojavljuje li se tamo glazbenik. Ova navigacija je slična navigaciji do pretrage, ali prije toga je na Android-u potrebno vratiti se unatrag skroz dok se ne pojavi meni na vrhu ekrana. Ovo se radi u petlji i u svakom koraku se pretražuje za poljem svih elemenata koji odgovaraju gumbima menija koristeći istu klasu kao što je spomenuta ranije. Ako je polje prazno, pritišće se Android gumb za povratak unatrag skroz dok se meni ne pojavi. Kad je meni vidljiv, pritišće se na drugi element, što odgovara drugom gumbu u meniju kako bi se došlo do korisnikove stranice. Na iOS-u je ovo jednostavnije. Pronalazi se element s imenom „My music“ i pritišće se na njega.

Ovdje je opet potrebno malo pričekati kako bi podaci imali vremena učitati se. Na Android-u će se traženi meni nalaziti prenisko da se na njega može samo pritisnuti, pa je ovdje potrebno koristiti metodu za vertikalno pomicanje zaslona iz biblioteke. Nakon toga je dovoljno pronaći i pritisnuti na element koji ima opis „Artists“ i zatim provjeriti postojanje elementa s opisom imena glazbenika koji je ranije dodan. Na iOS-u se pronalazi i pritišće na element s imenom „Artists“ i zatim se provjerava postojanje elementa s imenom glazbenika.

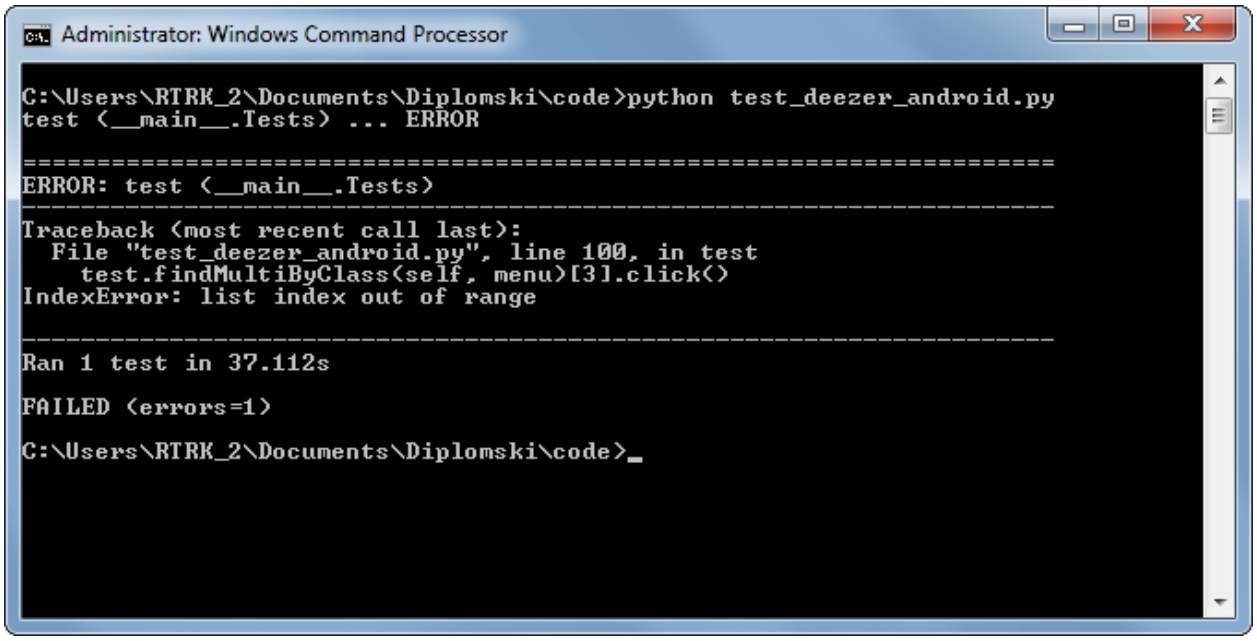
Na kraju, kako bi test bio što kompletniji, vrši se odjava korisnika. Postupak odjave u ovoj situaciji je u potpunosti identičan kao i ranije tijekom ovog testa.

Ovime je test efektivno završen. Ako je skroz sve korake uspješno prošao, smatra se prolazom, a ako se u bilo kojem koraku dogodilo nešto pogrešno, smatra se padom. Testovi se pokreću kroz komandnu liniju. Na slici 4.10 su vidljiva dva prozora komandne linije gdje su pokrenuti testovi i za Android i za iOS operacijske sustave i oba testa su uspješna.

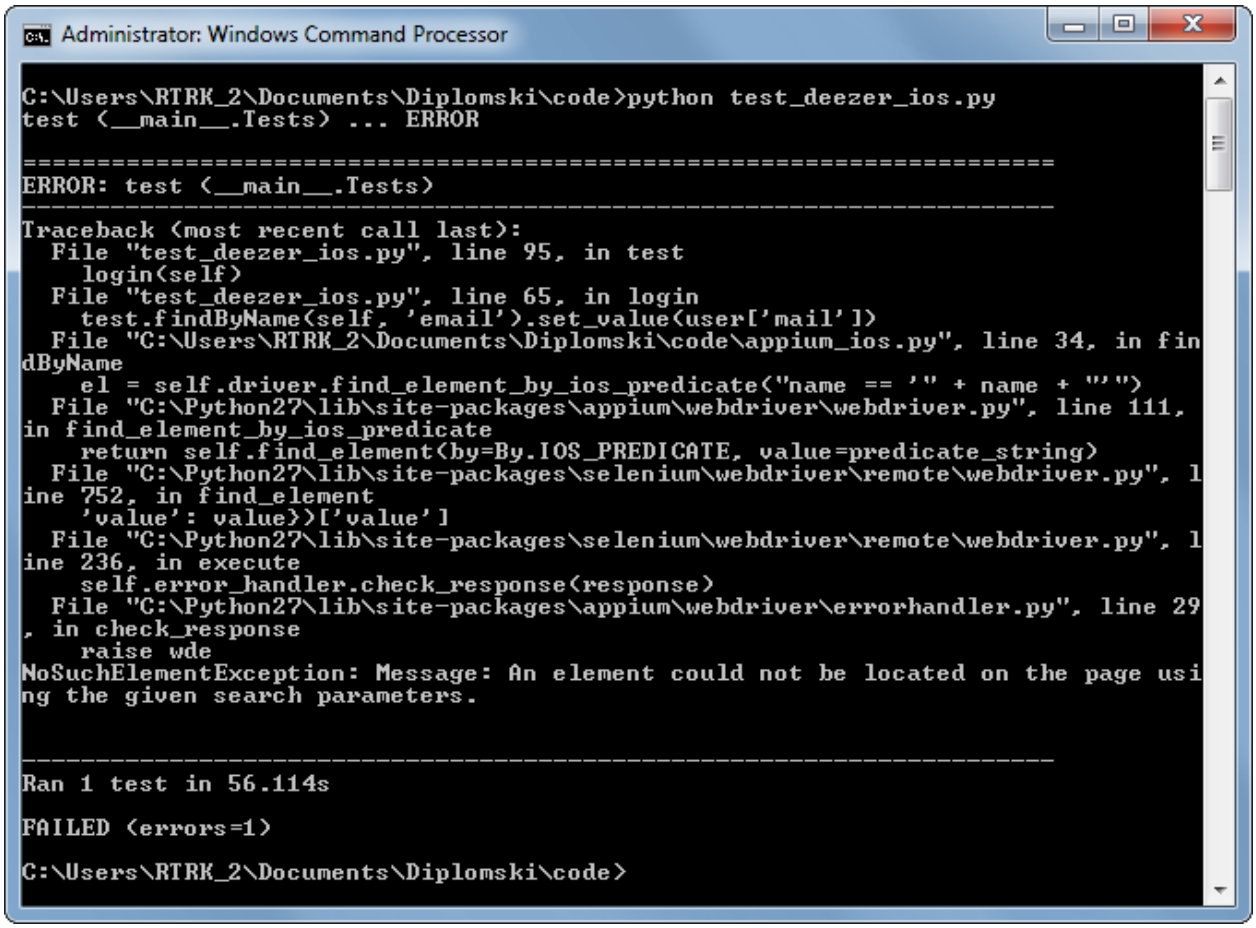


Sl. 4.10. Uspješni prolaz oba Deezer testa.

U realnoj situaciji, testovi neće uvijek biti uspješni. Uostalom, svrha im i je da se provjeri funkcionalnost aplikacije u raznim uvjetima. Tako su testovi ponovljeni, ali nakon što je na oba mobilna uređaja upaljen avionski način rada. Ovo onemogućuje pristup internetu pa će tako prijava u korisničke račune biti nemoguća. Na slikama 4.11. i 4.12. su prikazani rezultati ova dva testa. Zbog različitosti u korisničkom sučelju i dizajna samih skripti, postoje razlike između padova. Tako je pad na Android-u koji je prikazan na slici 4.11. prouzrokovan jer skripta pokušava pritisnuti na nepostojeći gumb u meniju. Na iOS-u, pad prikazan na slici 4.12. je nastao jer je skripta pokušala pronaći element po imenu koje ne postoji. Izvor oba ova pada je isti, a taj je da Deezer jednostavno ne može raditi bez pristupa internetu.



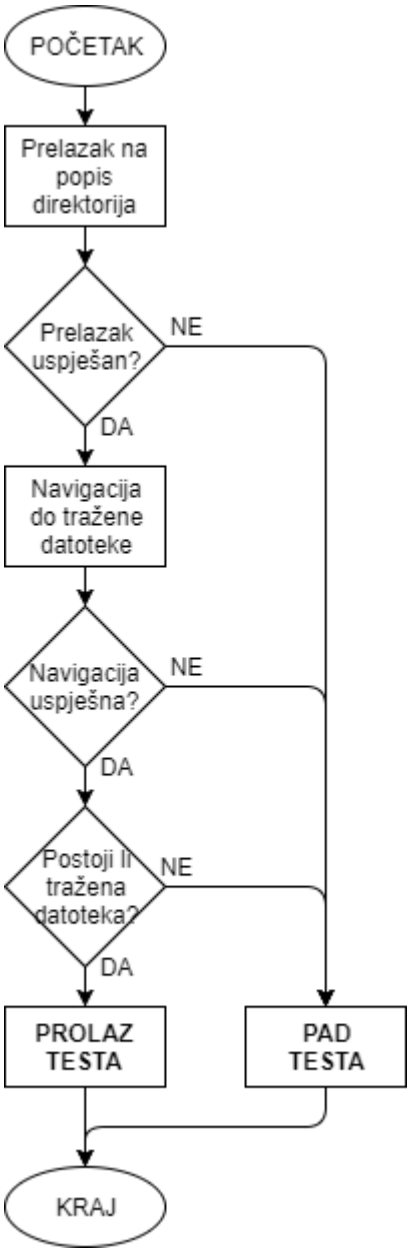
Sl. 4.11. Pad Deezer testa na Android-u.



Sl. 4.12. Pad Deezer testa na iOS-u.

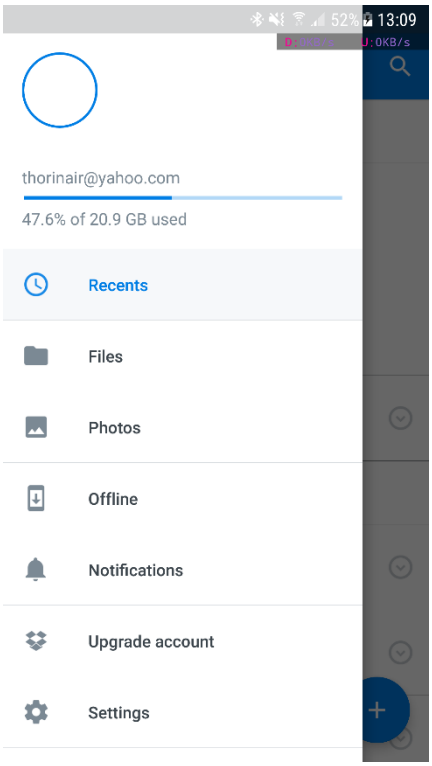
4.2. Testiranje aplikacije: Dropbox

Druga odabrana aplikacija je Dropbox. Dropbox je popularna usluga i aplikacija za pohranu datoteka u oblaku, dijeljenje datoteka s drugima i slično. Tako će cilj ovog testa biti navigirati kroz direktorije, pronaći traženu datoteku i provjeriti postoji li. Važno je napomenuti da je za ovaj test korišten unaprijed konfigurirani Dropbox, s unaprijed kreiranim datotekama i direktorijima. Na slici 4.13. osmišljen je dijagram toka ovog testa. Ovaj dijagram toka je značajno jednostavniji nego dijagram toka testa Deezer aplikacije i ima puno manje provjera. Isto kao i u prethodnom testu, provjere u dijagramu su simbolične i označavaju činjenicu da na bilo kojem koraku test može pasti.

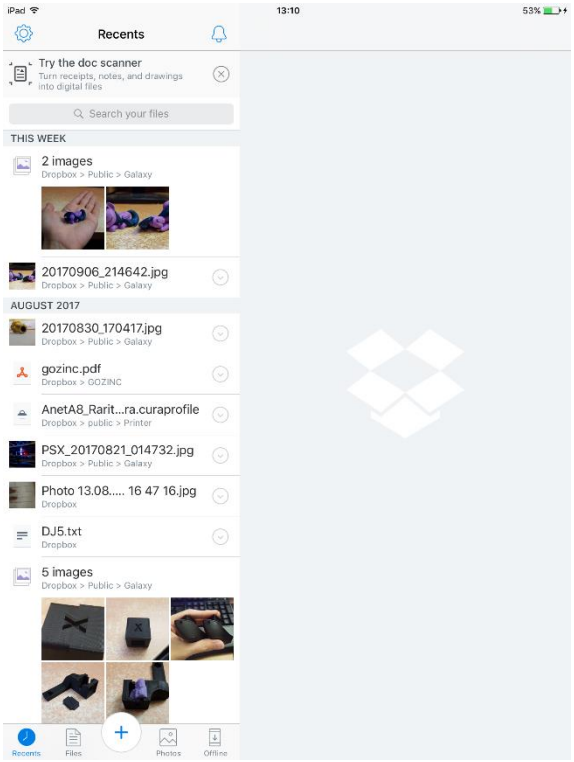


Sl. 4.13. Dijagram toka Dropbox testa.

Od samog početka, kao što je vidljivo na slikama 4.14. i 4.15., razlike između dvije verzije aplikacije su značajne. Ove slike prikazuju početni zaslon kakav je nakon što je aplikacija pokrenuta. Android verzija aplikacije zahtjeva dulje čekanje pri prvom učitavanju, pa je dodana naredba za čekanje. Kad je sučelje učitano, na Android aplikaciji se pronalazi i pritišće na element s tekstom „Files“. Na iOS aplikaciji, ovisno o situaciji, može biti više elementata koji se zovu „Files“, pa je tako potrebno dohvatiti polje elemenata i pritisnuti na zadnji element u polju kako bi se pritisnulo na točan gumb.



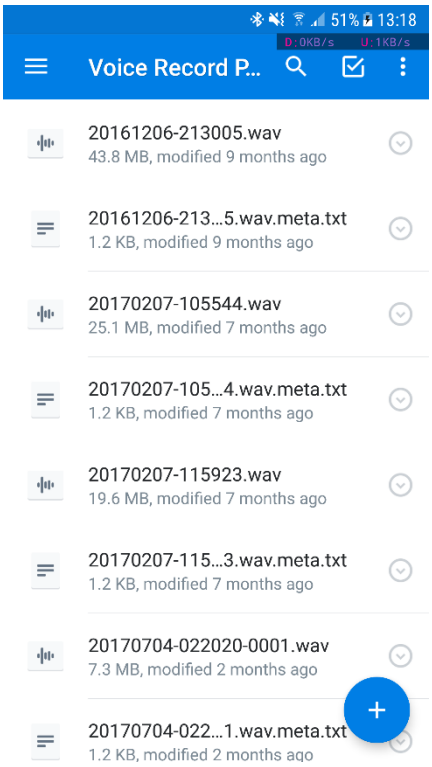
Sl. 4.14. Početni zaslon aplikacije Dropbox na Android-u.



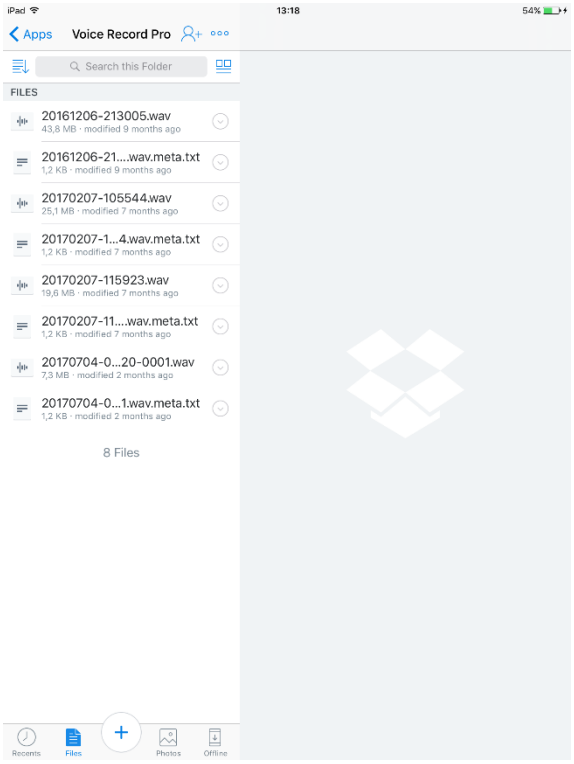
Sl. 4.15. Početni zaslon aplikacije Dropbox na iOS-u.

Ovdje je potrebno na Android aplikaciji opet pričekati. Android aplikacija se iz nekog razloga ponaša iznimno sporo pa je potrebno dodavati čekanja između svakog prelaska iz direktorij u direktorij. Kako bi se ostvarila navigacija kroz njih, elementi se pretražuju po tekstu i pritišću redoslijedom prvo „Apps“ i zatim „Voice Record Pro“. Na iOS-u je slična situacija gdje se elementi pretražuju po imenu. Ono što je kod iOS-a zanimljivo je što Dropbox aplikacija dodaje skrivenu vrijednost „. Folder“ i „. File“ ovisno je li riječ o direktoriju ili datoteci. Tako se na iOS-u traži i pritišće elemente s imenima „Apps. Folder“ i „Voice Record Pro. Folder“.

Nakon što je navigacija uspješna, korisničko sučelje će izgledati kako je prikazano na slikama 4.16. i 4.17. Ovdje se traži datoteka koja se zove „20161206-213005.wav“. Tako će se na Android-u tražiti element po tekstu „20161206-213005.wav“, a na iOS-u će se tražiti element po imenu „20161206-213005.wav. File“. Ako je element pronađen, test je uspješan.

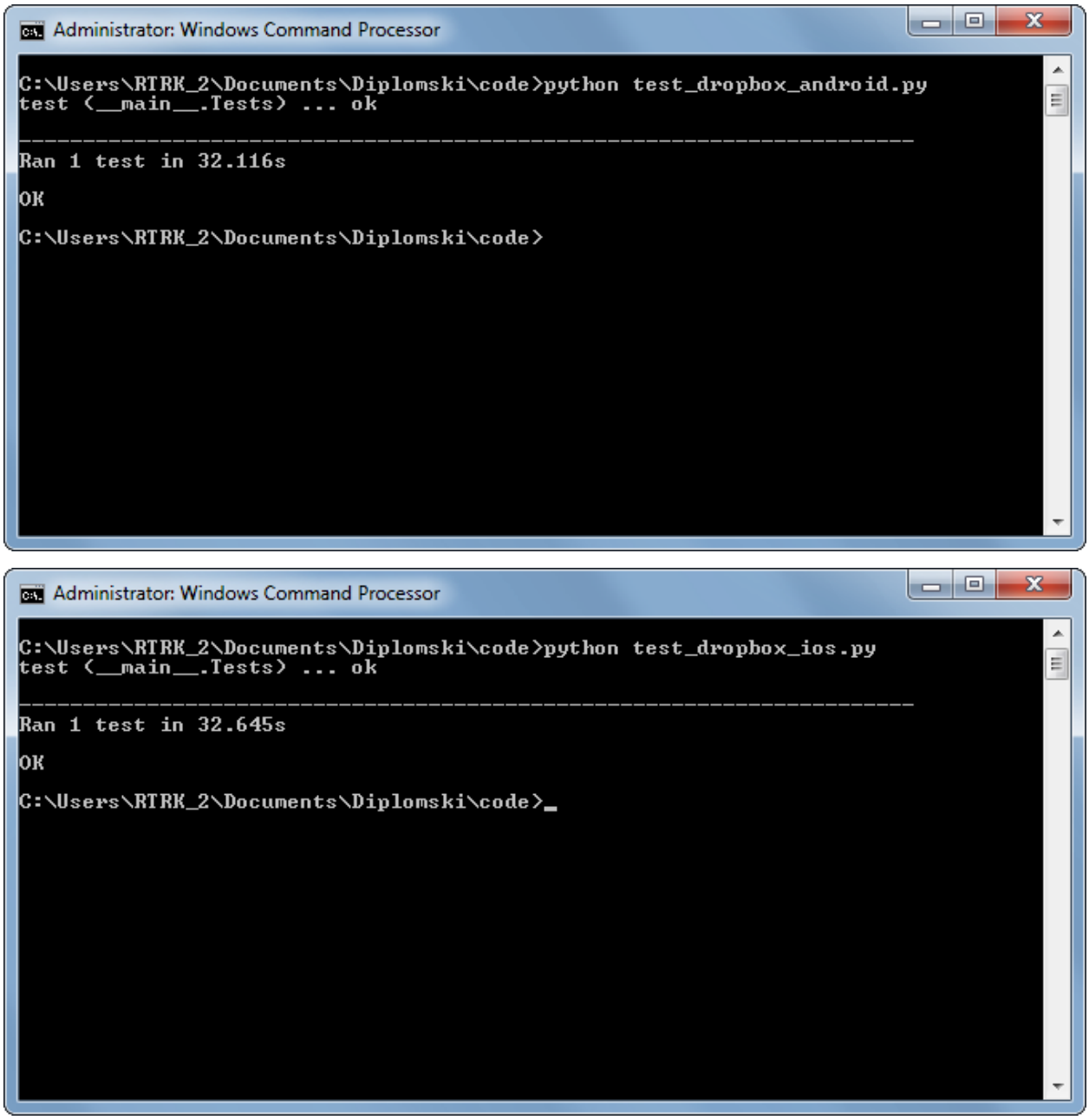


Sl. 4.16. Sadržaj direktorija u Dropbox-u na Android-u.

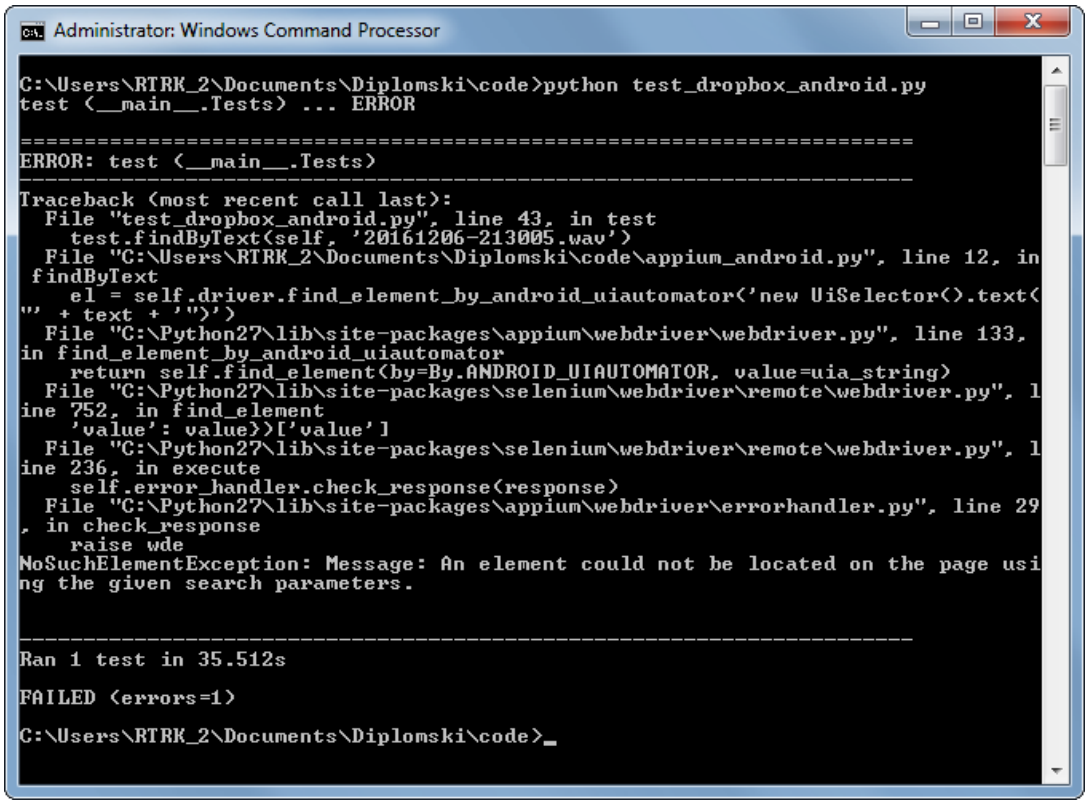


Sl. 4.17. Sadržaj direktorija u Dropbox-u na iOS-u.

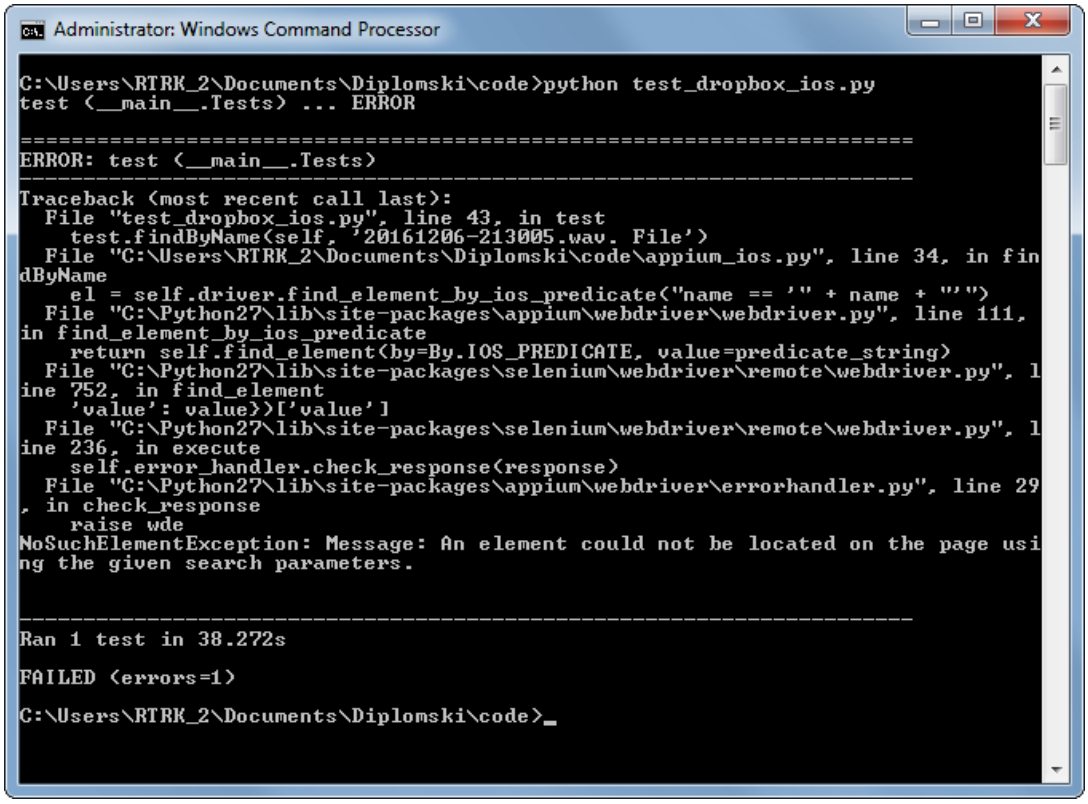
Na slici 4.18. je prikazan rezultat pokretanja ovih testova u komandnoj liniji. Kao što je vidljivo, oba testa uspješno prolaze. Slično kao što je urađeno s Deezer aplikacijom, potrebno je provjeriti i što se događa u neobičnoj situaciji Dropbox aplikacije. Tako je promijenjeno ime tražene datoteke i testovi su ponovno promijenjeni. Na slikama 4.19. i 4.20. je prikazan rezultat ova dva testa. Oba testa su pala jer je bilo nemoguće pronaći traženi element, a time i traženu datoteku.



Sl. 4.18. Uspješni prolaz oba Dropbox testa.



Sl. 4.19. Pad Dropbox testa na Android-u.



Sl. 4.20. Pad Dropbox testa na iOS-u.

5. ZAKLJUČAK

Kako se naš svijet razvija, tako se pojavljuju i sve kompleksnije mobilne aplikacije sa sve više mogućnosti. Testiranje svih mogućnosti se može raditi ručno, ali je ovo skup, dugotrajan i mukotrpan proces. Kako bi se ovi problemi riješili, raste potreba za automatskim testiranjem, to jest testovima koje izvodi računalo oponašajući interakciju stvarnih ljudi s aplikacijama.

Za razvoj testnog okruženja odabran je alat Appium jer je zadovoljio najviše kriterija potrebnih za ovaj rad. Testirani su Android i iOS mobilni uređaji budući da su to najčešće korišteni operacijski sustavi. Samo okruženje za testiranje se sastojalo od Mac računala sa spojenim Android i iOS mobilnim uređajima. Zajedno s Mac računalom je postojalo i Windows računalo spojeno u mrežu. Na oba računala je instalirano razvojno okruženje kako bi se testovi mogli neometano izvoditi. Windows računalo je slalo upute Mac računalu za izvođenje testova, a Mac računalo je testove izvodilo na fizičkim mobilnim uređajima. Naposljetku su napisani pokazni testovi za Deezer aplikaciju i Dropbox aplikaciju, kako bi se demonstrirala funkcionalnost ovako postavljenog okruženja za testiranje.

LITERATURA

[1] http://calaba.sh/ [Pristup ostvaren 30. 08. 2017.]

[2] http://appium.io/ [Pristup ostvaren 30. 08. 2017.]

[3] https://github.com/appium/appium-desktop [Pristup ostvaren 30. 08. 2017.]

[4] http://testingwithfrank.github.io/ [Pristup ostvaren 30. 08. 2017.]

[5] https://github.com/robotiumtech/robotium [Pristup ostvaren 30. 08. 2017.]

[6] https://github.com/appium/python-client [Pristup ostvaren 01. 09. 2017.]

[7] https://developer.apple.com/support/certificates/ [Pristup ostvaren 01. 09. 2017.]

[8] https://developer.apple.com/xcode/ [Pristup ostvaren 01. 09. 2017.]

[9] https://developer.android.com/studio/index.html [Pristup ostvaren 01. 09. 2017.]

[10] https://brew.sh/ [Pristup ostvaren 01. 09. 2017.]

[11] https://www.python.org/downloads/windows/ [Pristup ostvaren 01. 09. 2017.]

[12] http://offcornerdev.com/bundleid.html [Pristup ostvaren 08. 09. 2017.]

[13] http://www.seleniumhq.org/ [Pristup ostvaren 11. 09. 2017.]

SAŽETAK

U diplomskom radu opisuje se osmišljavanje, postavljanje i korištenje razvojnog okruženja koje omogućuje testiranje aplikacija za Android i iOS mobilne operacijske sustave. U uvodnim poglavljima je opisano trenutno stanje u svijetu automatskog testiranja i navedena su trenutna postojeća rješenja. Zatim je odabrano jedno od rješenja i opisan je postupak postavljanja okruženja. Na kraju je dano nekoliko pokaznih primjera testova kako bi se pokazala funkcionalnost okruženja za testiranje.

Okruženje je postavljeno koristeći Appium alat za testiranje i izvodi se na Mac računalu. Testovi se izvršavaju na fizičkim Android i iOS uređajima. Ovo računalo prima upute od Windows računala na kojemu se izvršava klijent. Klijent šalje zahtjeve Appium serveru za testovima, a koristi Python programski jezik. Okruženje je osmišljeno kako bi bilo što više robustno i fleksibilno za testiranje na oba mobilna operacijska sustava.

Ključne riječi: Appium, Python, macOS, iOS, Android, automatsko testiranje, mobilni uređaji, mobilne apliakcije

ABSTRACT

The thesis describes the designing process, setup and usage of a development environment which allows for testing of applications for the Android and iOS mobile operating systems. Introductory chapters describe the current state in the world of automated testing and lists currently available solutions. Then, one of the solutions is chosen and the procedure of setting up the environment is described. Finally, a few test examples are given, to show the functionality of the testing environment.

The environment is set up using the Appium testing tool and is run on a Mac computer. Tests are executed on physical Android and iOS devices. This computer receives instructions from a Windows computer where the client is executed. The client sends requests to the Appium server for tests and uses the Python programming langauge. The environment was designed to be as robust and flexible as possible for testing on both operating systems.

Key words: Appium, Python, macOS, iOS, Android, automated testing, mobile devices, mobile applications

ŽIVOTOPIS

Denis Vajak rođen je 20. siječnja 1992. godine u Osijeku, Hrvatska. Osnovnu školu završava 2007. godine u Osijeku, a iste godine upisuje III. Prirodoslovno-matematičku gimnaziju u Osijeku. Srednju školu završava 2011. godine i upisuje preddiplomski studij Računarstva na Elektrotehničkom fakultetu, Osijek i završava ga 2015. godine. Iste godine upisuje sveučilišni diplomski studij smjer Procesno računarstvo tijekom kojeg osvaja Rektorovu nagradu za rad „Pong igra za četiri igrača“. Tijekom diplomskog studija, bio je stipendist tvrtke Institut RT-RK Osijek d.o.o.

_____

Denis Vajak