Automatizacija regresijskog testiranja mobilne aplikacije

Svalina, Ivan

Undergraduate thesis / Završni rad

2023

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:200:839648

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

AUTOMATIZACIJA REGRESIJSKOG TESTIRANJA MOBILNE APLIKACIJE

Završni rad

Ivan Svalina

Osijek, 2023. godina

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 15.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Ivan Svalina
Studij, smjer:	Računalno inženjerstvo
Mat. br. Pristupnika, godina upisa:	R 4427, 11.10.2021.
OIB Pristupnika:	11630573104
Mentor:	doc. dr. sc. Ivan Vidović
Sumentor:	,
Sumentor iz tvrtke:	Vanja Pisačić
Naslov završnog rada:	Automatizacija regresijskog testiranja mobilne aplikacije
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	U sklopu ovog rada potrebno je istražiti postojeća rješenja za automatizaciju regresijskog testiranja mobilnih aplikacija na više platformi (Android i iOS). Nakon istraživanja, potrebno je predložiti vlastito okruženje za automatizaciju regresijskog testiranja mobilne aplikacije te ga usporediti s postojećim rješenjima. U praktičnom dijelu rada razvijeno okruženje je potrebno primijeniti na proizvolino odabranoj
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	15.09.2023.
Datum potvrde ocjene od strane Odbora:	24.09.2023.

Potvrda mentora o predaji konačne verzije rada:	Mentor elektronički potpisao predaju konačne verzije.
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 27.09.2023.

Ime i prezime studenta:	Ivan Svalina
Studij:	Računalno inženjerstvo
Mat. br. studenta, godina upisa:	R 4427, 11.10.2021.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Automatizacija regresijskog testiranja mobilne aplikacije**

izrađen pod vodstvom mentora doc. dr. sc. Ivan Vidović

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 2

2. PREGLED PODRUČJA 3

3. ODABIR APLIKACIJE I TESTNIH SLUČAJEVA 7

3.1. Aplikacija 7

3.2. Testni slučajevi 7

4. AUTOMATIZACIJA U POSTOJEĆEM ALATU 10

4.1. Postavljanje *Tricentis Testim* alata 10

4.2. Kreiranje automatiziranih testova 11

4.3. Kreiranje automatiziranog testa prijave postojećeg korisnika 13

5. VLASTITO RJEŠENJE ZA AUTOMATIZACIJU TESTIRANJA 18

5.1. Instalacija potrebnih alata 18

5.2. Postavljanje globalnih varijabli operacijskog sustava 20

5.3. Kreiranje i postavljanje projekta 21

5.4. Programsko pokretanje Appium poslužitelja i kreiranje baznih testova 22

5.5. Kreiranje *Page Object Modela* (*POM*) 24

5.6. Kreiranje testnih slučaja 25

5.7. Grupiranje testnih slučajeva u grupe i kreiranje testnog izvještaja 26

6. EVALUACIJA VLASTITOG RJEŠENJA 28

7. ZAKLJUČAK 30

LITERATURA 32

SAŽETAK 34

ABSTRACT 35

ŽIVOTOPIS 36

1. UVOD

Regresijsko testiranje postalo je jedno od najbitnijih načina testiranja programske podrške prilikom čijeg razvoja se koriste agilne metodologije. U agilnom razvoju timovi rade na malim inkrementalnim promjenama i poboljšanjima. Razvoj aplikacije se provodi u kratkim ciklusima koji se nazivaju sprintovi te se na kraju svakog sprinta isporučuje novi, poboljšani proizvod. Svrha regresijskog testiranja u agilnom razvoju je osiguravanje da implementirana poboljšanja i promjene nisu narušile stabilnost prijašnje verzije softvera. Regresijsko testiranje obuhvaća provjeru od prije postojećih ali i novo razvijenih funkcionalnosti. Manualno provođenje regresijskog testiranja je moguće na početku razvoja aplikacije kada aplikacija nema puno funkcionalnosti koje zahtijevaju provjeru. Kako aplikacija postaje slojevitija, sve više funkcionalnosti zahtijeva provjeru na kraju svakog sprinta te manualna izvedba regresijskog testiranja traje sve duže. Zbog toga se sve više timova oslanja na automatizirane regresijske testove. Automatizacija regresijskog testiranja je proces koji zahtijeva više truda i vremena pri početku postavljanja i pisanja testnih scenarija ali testerima kasnije omogućava lakše, brže i učinkovitije izvođenje regresijskog testiranja. Za manualno testiranje jedino je potrebno dovoljno dobro poznavati aplikaciju te elemente na koje mogu utjecati nove promjene, dok je za postavljanje i pisanje automatiziranih testova potrebno i imati dodatno tehničko znanje u programskim jezicima, alatima za pisanje testova te sami princip rada web ili mobilnih aplikacija.

Alati za automatizaciju testiranja mobilnih aplikacija se mogu svrstati u tri kategorije. U prvu kategoriju spadaju alati za čije je korištenje potrebno imati dobro poznavanje programskih jezika, strukture i načela razvoja aplikacije. To su alati poput *Selenium*-a i *Appium*-a, u kojima se pišu razne skripte za testiranje željenih funkcionalnosti. Za korištenje alata koji spadaju u drugu skupinu nije potrebna razina tehničkog znanja kao za prvu skupinu, jer je razina kodiranja u njima svedena na minimum ili potpuno izbačena. Treća kategorija je najnovija ali najmanje korištena zbog malog broja trenutno dostupnih alata. Alati iz ove skupine se nalaze u stadiju razvoja jer se u tim alatima želi iskoristiti brzo rastući i sve poznatiji koncept umjetne inteligencije.

U prvom poglavlju je opisan teorijski dio općeg i regresijskog testiranja. Objašnjena je važnost provođenja regresijskog testiranja u završnim fazama ciklusa razvoja softvera. Također su pokazana dva moguća pristupa regresijskom testiranju: manualno testiranje i automatizirano testiranje.

Nadalje, u drugom poglavlju je kratko objašnjen početak testiranja mobilnih aplikacija. Također su objašnjeni koncepti rada alata *Appium* i pogonskih programa koji se mogu koristiti u kombinaciji s alatom. U istom poglavlju je dan uvid u važnost *LowCode/NoCode* alata za razvoj softvera te su navedeni vodeći *LowCode/NoCode* alati na tržištu. Na kraju poglavlja je ukratko opisano rješenje i alat koje su autori, koji su profesori sveučilišta u Brazilu, kreirali zbog nedostataka komercijalno dostupnih alata.

U trećem poglavlju, opisana je aplikacija odabrana za automatizaciju testiranja. Osim toga odabrano je i opisano pet testnih slučajeva koji su kasnije u radu automatizirani u dostupnom alatu *Tricentis Testim* i vlastito kreiranom rješenju.

U četvrtom poglavlju je u cijelosti prikazan proces postavljanja alata *Tricentis Testim*. Opisan je opći tok kreiranja testova u alatu te je između ostaloga u potpunosti prikazano kreiranje jednog automatiziranog testnog slučaja.

Potom je u petom poglavlju približen zahtjevan proces razvoja vlastitog rješenja za automatizaciju testiranja. Prikazani su korišteni alati, važnost koncepta postavljanja globalnih varijabli operacijskog sustava i dan je uvid u izvorni kod razvijenog rješenja.

U zadnja dva poglavlja su uspoređeni proces postavljanja, tehnička zahtijevanja i rezultati korištenog alata *Tricentis Testim* i razvijenog rješenja. Također su navedena moguća daljnja proširenja vlastito kreiranog rješenja.

1.1. Zadatak završnog rada

U teorijskom dijelu rada potrebno je pronaći i proučiti već dostupna rješenja za automatizaciju testiranja mobilne aplikacije. Potom je u praktičnom dijelu potrebno odabrati i napisati nekoliko testnih scenarija koji se vrlo često, gotovo uvijek, provode i provjeravaju tijekom regresijskog testiranja odabrane mobilne aplikacije. Nadalje je potrebno strukturirati, postaviti i razviti vlastito okruženje koje omogućuje pisanje i izvođenje automatiziranih testnih slučajeva. Razvijeno okruženje treba imati mogućnost izvođenja testova na *iOS* i *Android* platformama uz minimalne izmjene u kodu.

2. PREGLED PODRUČJA

Kod automatizacije testiranja mobilnih aplikacija postoje razni pristupi, shodno tome i različita rješenja i proizvodi s različitim mogućnostima.

U ranim početcima korištenja mobilnih uređaja poput pametnih mobitela i tableta postojalo je mnogo različitih platformi i operacijskih sustava koje su ti uređaji koristili. Prema globalnim statistikama o korištenim operacijskim sustavima [1] se vidi da dva operacijska sustava, *Android* i *iOS*, ubrzo postaju najpopularniji mobilni operacijski sustavi. Većina aplikacija koje su se tada razvijale su bile nativne aplikacije, no na tržište su se polako počele probijati i hibridne aplikacije. Hibridne aplikacije kombiniraju nativne mogućnosti uređaja s mogućnostima web aplikacije, što omogućava korištenje aplikacije na više platformi. Takve aplikacije se razvijaju pomoću tehnologija poput *HTML5*, *JavaScript* i *CSS* te se na mobilnim uređajima pokreću u web preglednicima. Iako se mogu koristit na više platformi, hibridne aplikacije zaostaju za nativnim aplikacijama u pogledu performansi i responzivnosti. Zbog toga se pojavljuje potreba za automatiziranim testovima koji osiguravaju da aplikacije rade ispravno na mobilnim uređajima.

Iako je prvotno razvijen za testiranje web aplikacija na računalima, *Selenium* je bio jedan od prvih i najpopularnijih alata za automatizaciju testiranja mobilnih web aplikacija. U knjizi [2] autor objašnjava kako postaviti razvojno okruženje i izvoditi testove na *iOS* i *Android* platformama koristeći njihove pripadne pogonske programe. Pogonski programi su vrsta softvera koji omogućuju *Selenium*-u interakciju s web preglednikom. *iPhone* pogonski program i *Android* pogonski program komuniciraju pomoću *HTTPS/JSON* protokola s aplikacijama na *iOS* i *Android* uređajima kako bi izvodili testove za mobilne web aplikacije napisane u *Seleniumu*. Te aplikacije, *iWebDriver App* za *iOS* i *Android Server APK* za Android, se sastoje od implementacije *RemoteWeb* pogonskog programa i komponente za prikazivanje web preglednika. Daljnjim razvojem tehnologije, pametnih uređaja i mobilnih web aplikacija, *Selenium* počinje opadati u popularnosti te ga zamjenjuju alati posebno razvijeni za testiranje mobilnih web aplikacija.

Jedan od trenutno najpopularnijih alata za testiranje mobilnih aplikacija je *Appium* [3], *open-source* okvir koji koristi *WebDriver* protokole za interakciju s mobilnim uređajima. Koristeći različite *Web* pogonske programe mogu se automatizirati testovi na različitim platformama [4]. Primjerice, *Esspreso* ili *UIAutomator2* pogonski programi omogućuju automatiziranje testova na Android operacijskom sustavu dok se *XCUITest* koristi za automatiziranje na *iOS* platformi.

Appium XCUITest pogonski program je zadužen za rukovanje *WebDriver* naredbama i stvarno izvođenje tih naredbi na iOS uređaju. Stvarno izvođenje naredbi je ostvareno korištenjem *XCUITest* razvojnog okvira tvrtke *Apple*. *XCUITest* je posebno dizajnirani razvojni okvir isključivo namijenjen za testiranje korisničkog sučelja *iOS* aplikacija. Zasniva se na *XCTest* razvojnom okviru za testiranje bilo koje vrste koda i razina testiranja poput testiranja pojedinačnih komponenti koda i integracijskog testiranja. Prema [5] pri kreiranju testova za provjeru korisničkog sučelja koristi se referenca na *XCUIApplication* koja zapravo predstavlja zamjenu aplikacije koju se testira. Pokretanjem *XCUIApplication*-a otvara se željena aplikacija te se dobivaju reference na sve njene vidljive elemente. Reference vidljivih elemenata su zapravo reference na objekte potklasa klase *XCUIElement* koja predstavlja svaki *view* na zaslonu. Klasa *XCUIElement* pruža svojstva i metode pomoću kojih se može komunicirati s njenim objektima i djelovati na iste što omogućava simuliranje interakcije korisnika s aplikacijom.

Na sličan način kao i *XCUITest* pogonski program , *Appium Espresso* pogonski program koristi kombinaciju već postojećeg *Espresso* razvojnog okruženja i posebno napisanog koda za rukovođenje komunikacijom između *Espresso* okruženja i *Appium*-a. *Espresso* je razvijen od tvrtke *Google* 2013. godine specifično za testiranje korisničkih sučelja aplikacija na *Android* platformi. Autor u [6] pojašnjava opći tijek pisanja testova za provjeru korisničkog sučelja i njihovu konkretnu implementaciju u *Espresso* razvojnom okviru. Pisanje testova za provjeru korisničkog sučelja se također svodi na simuliranje interakcija krajnjeg korisnika s aplikacijom u tri koraka:

1. pronalazak svih vidljivih elemenata,
2. izvođenje radnji na pronađenim elementima,
3. provjera rezultata.

Za svaki prethodno navedeni korak Espresso razvojno okruženje pruža sučelja za izvođenje željene radnje. Pomoću objekata koji implementiraju sučelje *ViewMatcher* moguće je locirati neki element na zaslonu putem njegovog *ID*-a, oznake ili opisom sadržaja tog elementa. Nakon što je željeni element lociran na njemu se mogu izvoditi radnje pomoću *ViewAction* objekata. *ViewAction* objekti mogu izvoditi radnje poput pritiska na element, akcije *scrollanja*, dugog pritiska i mnoge druge. Zadnji korak je provjera rezultata koja se izvodi koristeći *ViewAssert* objekte.

Uz pogonske programe za *Android* i *iOS* platforme, *Appium* podržava brojne druge pogonske programe koji proširuju opseg platformi i uređaja za koje se mogu automatizirati testni scenariji. *Appium* omogućava programerima koji poznaju različite programske jezike lako korištenje jer omogućava pisanje testova u više jezika kao što su *Java, Python, JavaScript, Ruby, C#* i mnogi drugi.

Iako se *Appium* čini kao savršeni alat, zbog velike raznolikosti platformi i jezika koje podržava te mogućnosti koje pruža, za automatizaciju testova koristeći *Appium* potrebno je bolje poznavanje arhitekture i principa rada mobilnih aplikacija te postavljanje i pisanje testova nekada može biti dugotrajan i mukotrpan proces. Upravo iz tog razloga sve više su popularne *low-code* i *no-code*, skraćeno „*LC/NC*“, platforme i alati. Ovakve platforme omogućuju stvaranje i izvođenje testnih scenarija i testnih skripti pritom eliminirajući potrebu za većim tehničkim vještinama i kodiranjem. *LC/NC* alati koriste *point-and-click* i *drag-and-drop* funkcionalnosti kako bi olakšali svim korisnicima automatizaciju testiranja njihove aplikacije.

Neki od poznatijih *LC/NC* alata i platformi koji se danas koriste su *AccelQ*, *Tricentis Testim* i *Maestro*. Sva tri alata omogućuju kreiranje i izvođenje testova na sličan i jednostavan način. Za primjer je uzet alat *Tricentis Testim*. Kreiranje testova je vrlo jednostavan proces snimanja koraka i postupaka kojima se provjerava i verificira valjanost testirane funkcionalnosti. Također, ovi alati omogućuju izvedbu testova na stvarnim fizičkim uređajima ili virtualnim uređajima na emulatorima poput onoga u razvojnom okruženju *Android Studio*. Priprema izvedbe testova na virtualnim ili fizičkim uređajima je vrlo jednostavan proces instalacije *Mobile Agent* softvera na računalo te pokretanje emulatora ili spajanje fizičkog uređaja na računalo putem *USB* kabela. Ovakav jednostavan proces automatizacije testiranja je razlog zašto se očekuje da će *LC/NC* alati i platforme biti glavni, no ne i jedini, pristup automatizaciji, ali i općeg razvoja programske podrške [7].

Veliki nedostatak navedenih *LC/NC* alata za automatizaciju testiranja je nemogućnost pisanja automatiziranih testova za izvođenje na više platformi, poput *iOS* i *Android* platformi. Iako to ne izgleda kao veliki problem mora se razumjeti da broj testnih slučajeva, za kompleksne i slojevite aplikacije, može doseći i četveroznamenkaste iznose. U tom slučaju nemogućnost *cross-platform* izvođenja testova znači kako bi za dvije platforme testeri koji razvijaju i pišu testove morali napisati dupli skup testova, po jedan za svaku platformu. Upravo ovaj nedostatak je jedan od glavnih razloga zašto su, iako jednostavniji, *LC/NC* alati ipak drugi izbor naspram tradicionalnog razvoja automatiziranih testova na većim projektima. Potaknuti nedostacima

ostalih alata, autori u [8] predstavljaju novi pristup kreiranju skripti za testiranje mobilnih aplikacija na *iOS* i *Android* platformama. U radu su proučili šest poznatih i često korištenih pristupa i strategija individualnog lociranja elemenata korisničkog sučelja. U nastavku su predložili dvije strategije pronalaska elemenata koje koriste svih šest prethodno navedenih strategija u svrhu poboljšanja pouzdanosti i preciznosti testova. Naposljetku autori su kreirali svoje vlastito rješenje za kreiranje testnih skripti, imena *x-PATeSCO*, koje bi se mogle izvoditi na *iOS* i na *Android* platformama. Kreirano rješenje su testirali i usporedili s ostalim pristupima i strategijama te pokazali da je njihovo rješenje preciznije i više primjenjivo za testiranje na različitim platformama.

3. ODABIR APLIKACIJE I TESTNIH SLUČAJEVA

Odabir aplikacije za koju će se automatizirati regresijsko testiranje je važna stavka u ranim fazama procesa automatizacije. Postoji mnogo različitih vrsta aplikacija koje koriste različite tehnologije prilikom razvoja. Tehnologija kojom je aplikacija razvijena može utjecati na odabir pristupa i alata kojim će se testovi automatizirati. Na današnjem tržištu se koriste različite tehnologije razvoja aplikacije, od nativnih aplikacija koje se razvijaju za korištenje na samo jednoj platformi do hibridnih aplikacija koje se pokreću na uređajima putem web preglednika. Također postoje i aplikacije koje koriste nove tehnologije koje omogućavaju razvoj aplikacija za uporabu na više platformi. Proces automatizacije jednostavnijih aplikacija, aplikacija koje nemaju bazu podataka, ne koriste usluge trećih strana će biti puno jednostavniji naspram procesa automatizacije složenijih i slojevitih aplikacija. Također bitna stavka u odabiru aplikacije je profil korisnika aplikacije i sami broj korisnika aplikacije. Za aplikacije s većim brojem korisnika je potrebno osim automatizacije testnih slučajeva automatizirati takozvane stres testove kako bi se osiguralo pravilno ponašanje aplikacije pri velikom volumenu aktivnih korisnika.

3.1. Aplikacija

Odabrana aplikacija je *The Phoenix – A sober community [9]*, čiji je cilj pružanje podrške ljudima koji se bore s različitim vrstama ovisnosti putem događanja i druženja te raznih grupa gdje korisnici mogu podijeliti svoje hobije i strasti s ostalim članovima aplikacije. Aplikacija je odabrana jer je podržana na *iOS* i *Android* platformama, te razvijena pomoću nove tehnologije *Kotlin Multiplatform*, dalje u tekstu *KM*. *KM* je tehnologija koja omogućava programerima pisanje koda koji se može upotrijebiti na više platformi. Pomoću ove tehnologije programeri mogu koristit istu pozadinsku logiku, algoritme i pomoćne funkcije među različitim platformama dok i dalje dozvoljava korištenje specifičnih implementacija i funkcionalnosti svake platforme.

3.2. Testni slučajevi

Testni slučajevi su skup koraka ili akcija koji se trebaju izvesti pri testiranju, te očekivano ponašanje koje se treba dogoditi nakon izvođenja tih koraka. Očekivana ponašanja su najčešće navedena u dokumentu u obliku funkcionalnih i ne-funkcionalnih zahtjeva koje klijent postavi. Za potrebe ovog završnog rada izabrano je i opisano pet testnih slučajeva koji pokrivaju

najbitnije funkcionalnosti aplikacije te koji će biti automatizirani u praktičnom dijelu rada, prikazani u tablici 3.1.. Testni slučajevi koji će biti korišteni u ovome radu su:

- prijava postojećeg korisnika,
- odjava korisnika iz aplikacije,
- rezervacija mjesta za događaj,
- pridruživanje korisnika u grupu,
- kreiranje objave u grupi.

Tablica 3.1. Testni slučajevi za provjeru osnovnih funkcionalnosti aplikacije.

ID	**Opis**	**Koraci**	**Potrebni podaci**	**Očekivano ponašanje**
T-01	Prijava već postojećeg korisnika	1. Otvori aplikaciju 2. Pritisni gumb *Log In With Email* 3. Unesi e-mail 4. Unesi lozinku 5. Pritisni gumb *Continue*	Email: svaleqa+zr@gmail.com Lozinka: Test123!	Korisnik se uspješno prijavio
T-02	Odjava korisnika iz aplikacije	1. Otvori aplikaciju 2. Pritisni profilnu sliku 3. Pritisni gumb za postavke 4. Pozicioniraj se na dno stranice 5. Pritisni gumb *Log out*	-	Korisnik se uspješno odjavio iz aplikacije
T-03	Rezervacija mjesta za događaj	1. Otvori aplikaciju 2. Pritisni prvu karticu događaja u listi događaja 3. Pritisni gumb *Reserve Spot*	-	Korisnik je rezervirao mjesto i vidljiv je u listi polaznika
T-04	Pridruživanje korisnika u grupu	1. Otvori aplikaciju 2. Pritisni ikonu *Groups* 3. Pritisni prvu karticu grupe u listi grupa 4. Pritisni gumb *Join group*	-	Korisnik se pridružio grupi i vidljiv je u listi članova grupe

T-05	Kreiranje objave u grupi	1. Otvori aplikaciju 2. Pritisni ikonu za pretraživanje 3. U polje za unos unesi ime grupe „Testna Grupa“ 4. Pritisni na karticu grupe 5. U polje za unos *What's on your mind* unesi tekst objave 6. Pritisni gumb *Post*	-	Objava je kreirana i vidljiva u listi objava

4. AUTOMATIZACIJA U POSTOJEĆEM ALATU

Za usporedbu rezultata i mogućnosti okruženja odabrani testni slučajevi su automatizirani u dostupnom alatu *Tricentis Testim.* Ovo razvojno okruženje omogućuje vrlo brzu automatizaciju skupine testnih slučajeva kojoj mogu pridonijeti svi članovi tima, čak i oni bez tehničkog znanja. To je ostvareno lakim postavljanjem testnog okruženja, brzim i jednostavnim snimanjem testnih koraka i mogućnosti automatskog popravljanja testova koristeći umjetnu inteligenciju. *Tricentis Testim* je proizvod za čije korištenje je potrebno kupiti licencu no uz probni rok u trajanju petnaest dana, korisnici mogu proučiti alat i njegove mogućnosti te odabrati koja kombinacija mogućnosti im je zaista potrebna. Uz probni rok, *Tricentis Testim* nudi novim i probnim korisnicima mogućnost dogovaranja demonstracije alata gdje će korisnike upoznati sa svim koracima postavljanja, procesima i mogućnostima alata.

4.1. Postavljanje *Tricentis Testim* alata

Postavljanje alata *Tricentis Testim* je vrlo jednostavan i izravan proces. Nakon kupnje licence ili aktivacije probnog perioda korisnik će u e-mailu dobiti link na svoje okruženje. Za postavljanje okruženja korisnik mora pratiti skup koraka koji se nalaze na web stranici *Tricentis Testim* alata [10] ili na *Youtube* kanalu gdje je postavljanje okružja objašnjeno kroz više video uradaka. Prvi korak je preuzimanje i instalacija *Tricentis Mobile* agenta. *Tricentis Mobile* je softver koji rukuje konekcijama s mobilnim uređajima za izvođenje lokalnih testova na fizičkim ili virtualnim uređajima.

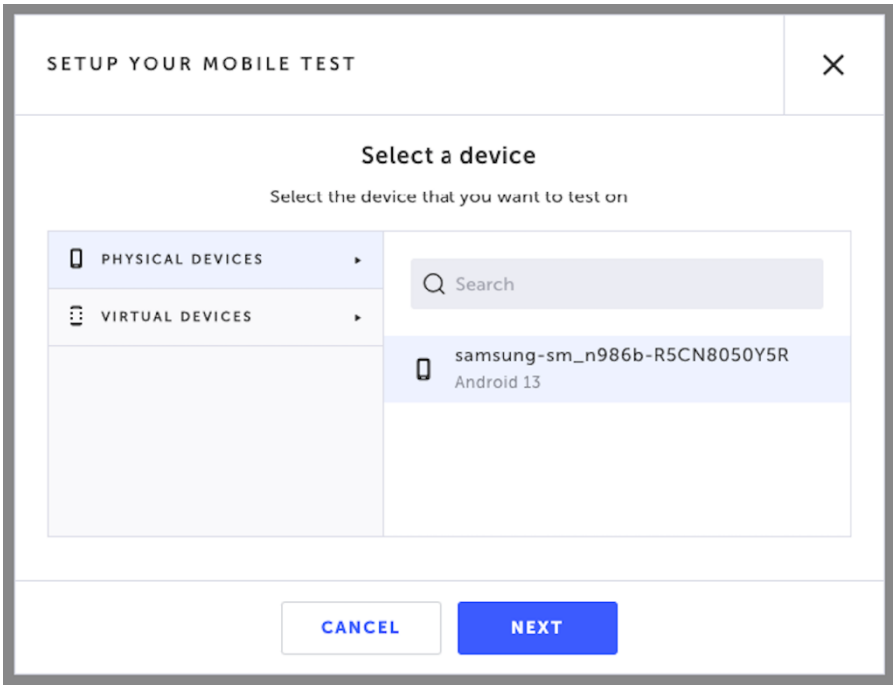
Nakon instalacije *Tricentis Mobile* agenta, prije početka snimanja i kreiranja samih testova, korisnik mora spojiti fizički ili virtualni uređaj. Fizički uređaj se spaja kabelom na računalo te na uređaju mora biti omogućena *USB Debbuging* opcija u postavkama uređaja kako bi računalo moglo komunicirati sa spojenim uređajem. Virtualni uređaj se pokreće kroz *Android Studio* emulator s preporučenim tehničkim specifikacijama. Nakon povezivanja uređaja *Tricentis Mobile* agent će prepoznati spojeni uređaj te će on biti prikazan u listi dostupnih uređaja.

Zadnji korak u postavljanju okruženja je učitavanje aplikacije za koju se kreiraju testovi. Aplikaciju se može učitati u okruženje na dva načina. Prvi način je kroz *Mobile apps* karticu u izborniku gdje se mora učitati *.sdk* datoteka za *Android* ili *.ipa* datoteka za iOS aplikacije. Drugi način učitavanja aplikacije je pri samom početku pisanja testa, *Tricentis Mobile* agent će prikazati izbornik svih instaliranih aplikacija na uređaju gdje se može odabrati željena

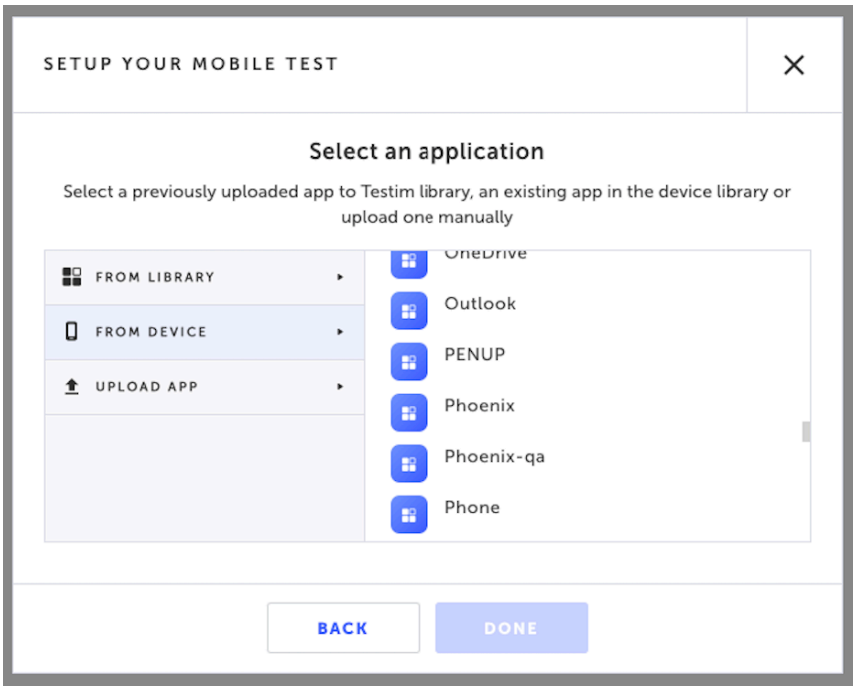
aplikacija. Nakon učitavanja aplikacije okruženje je postavljeno te se mogu početi pisati automatizirani testovi.

4.2. Kreiranje automatiziranih testova

Testovi se kreiraju tako da se snimaju koraci koje je potrebno izvesti u testnom slučaju. Za pokretanje stvaranja testnog slučaja treba stisnuti *New Test* nakon čega se otvara uređivač. U uređivaču se kreiraju testni slučajevi dodavajući testne korake u redu kako će se izvoditi. Prije dodavanja prvog koraka mora se odrediti na kojem spojenom uređaju te na kojoj aplikaciji će se izvoditi test (Slike *4.1*. i *4.2*.).



Slika 4.1. Izbornik za odabir uređaja na kojem će se izvršiti testni slučaj.



Slika 4.2. Izbornik za odabir aplikacije za koju se kreiraju testni slučajevi.

Nakon odabiranja aplikacije i uređaja, dodaje se prvi korak. Prvi korak je uvijek postavljanje aplikacije, to jest instalacija aplikacije ako se ona ne nalazi na uređaju, uspostavljanje komunikacije s uređajem te pokretanje aplikacije. Nakon uspostavljanja komunikacije s uređajem *Testim Mobile* alat započinje prikazivanje zaslona mobilnog uređaja na zaslonu računala. Na tom prikazu zaslona se upravlja aplikacijom za vrijeme kreiranja testa te se na njemu snimaju koraci testnog scenarija. Dodavanje koraka u testni scenarij se može izvesti stvarnom interakcijom s aplikacijom, poput pritiska na gumb, unosa teksta i akcija *scrollanja* ili dodavanjem prethodno definiranih koraka koji su dostupni u alatu. Neki od dostupnih prethodno definiranih osnovnih koraka su:

- validacija je li element vidljiv,
- validacije teksta unutar elementa,
- ekstrakcija vrijednosti iz nekog elementa,
- dodavanje teksta u polje za unos,
- akcija čekanja dok element ne postane vidljiv,
- generiranje privremene e-mail adrese.

Pomoću ovih koraka se mogu kreirati osnovni automatizirani testovi za provjeru korisničkog sučelja.

4.3. Kreiranje automatiziranog testa prijave postojećeg korisnika

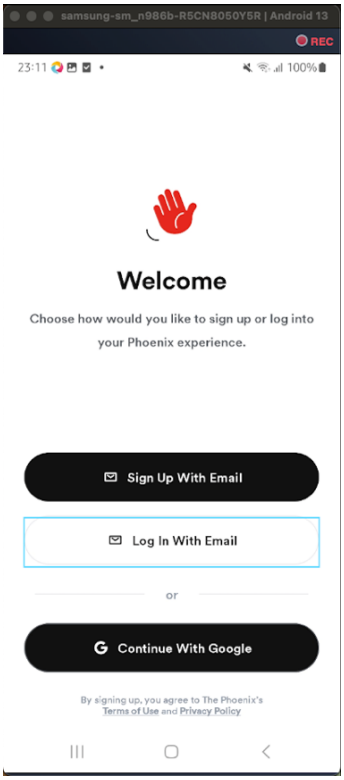
Prema tablici *3.1.* za provjeru testnog slučaja prijave postojećeg korisnika potrebno je izvesti sljedećih pet koraka:

1. otvori aplikaciju,
2. pritisni gumb *Log In With Email,*
3. unesi e-mail,
4. unesi lozinku,
5. pritisni gumb *Continue.*

Također potrebni su e-mail na koji je registriran korisnički račun i lozinka za prijavu:

- e-mail: svaleqa+zr@gmail.com,
- lozinka: Test123!

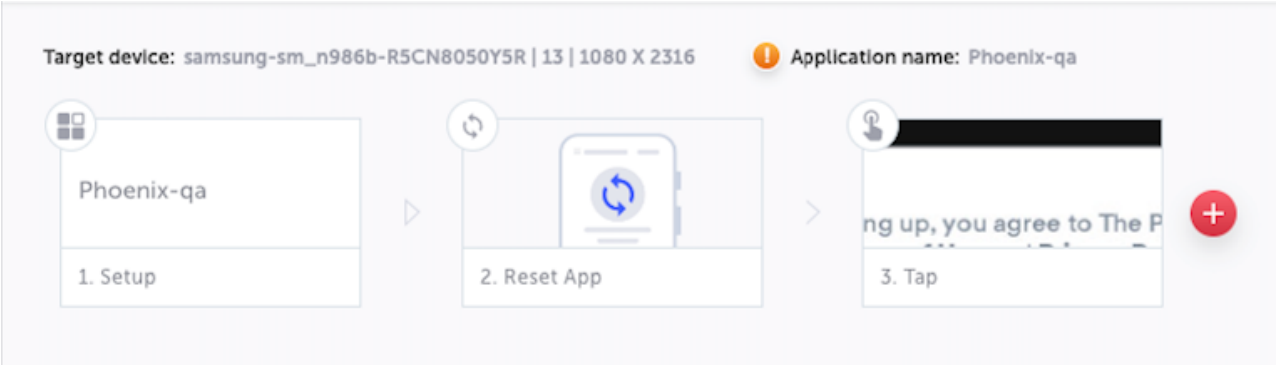
Nakon što su definirani koraci i potrebni podaci može se početi kreirati test. Prvi korak, kao što je navedeno prethodno u tekstu je postavljanje aplikacije. Nakon toga dobra je praksa staviti korak *Reset* koji potpuno resetira aplikaciju, briše *cache* memoriju te sve podatke aplikacije. Nakon dodavanja koraka *Reset* može se krenuti sa snimanjem koraka testnog slučaja. Prvi korak je pritisak na gumb *Log in with Email* (Slika 4.3.)



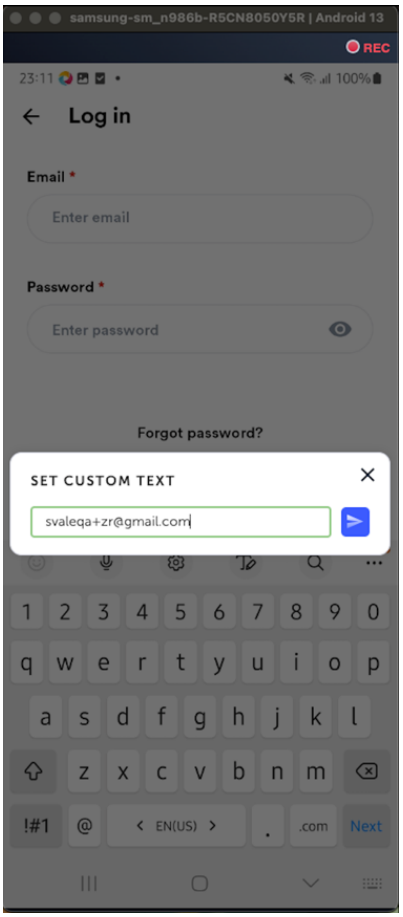
Slika 4.3. Pritisak na gumb *Log In With Email.*

Alat koristi posebne algoritme strojnog učenja kojim se analiziraju svi identifikatori u aplikaciji te ih svrstava u kategorije elemenata kao što su gumb, tekst, slika itd., te dodjeljuje moguće akcije svakoj kategoriji elemenata. Na prikazanom zaslonu elementi su istaknuti plavim obrubom kada se preko njih pređe pokazivačem miša. Nakon što je željena akcija snimljena ili dodana ona se pojavljuje u listi koraka (Slika 4.4.).

Sljedeći korak je unos e-maila i lozinke u polja za unos, tu pritiskom na polje za unos korisnik može unijeti tekst koristeći tipkovnicu računala (Slika 4.5.).

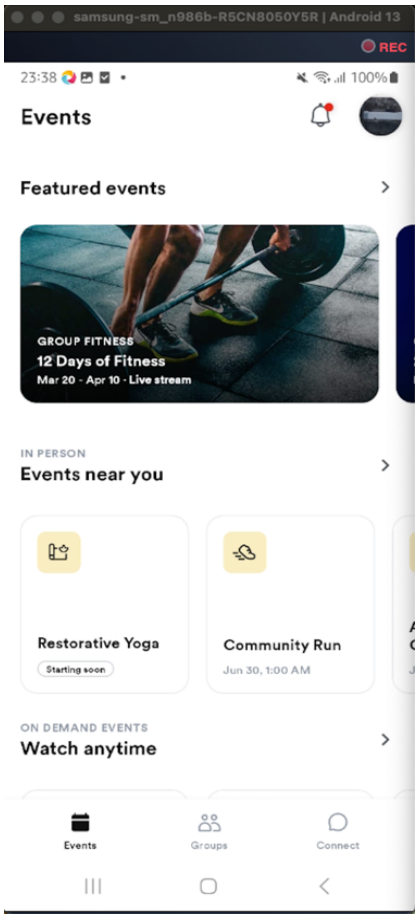


Slika 4.4. Lista koraka koji se izvode u testnom slučaju.



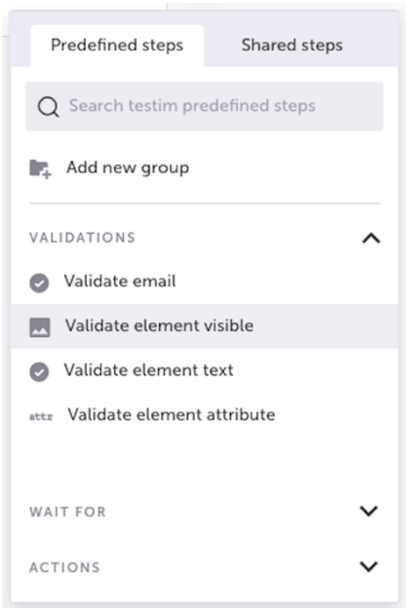
Slika 4.5. Korak unosa e-maila za prijavu u aplikaciju.

Nakon što su uneseni e-mail i lozinka potrebno je pritisnuti gumb *Continue* kojim će se izvršiti prijava u aplikaciju. Na kraju je potrebno potvrditi da je korisnik uspješno prijavljen u aplikaciju. To će se provjeriti validacijom da je naslov *Events* vidljiv na početnom zaslonu aplikacije. Taj naslov se nikada ne mijenja te je jedan od prvih elemenata vidljiv pri ulasku u aplikaciju tako da je to dobra točka za validaciju je li se korisnik uspješno prijavio u aplikaciju (Slika 4.6.).



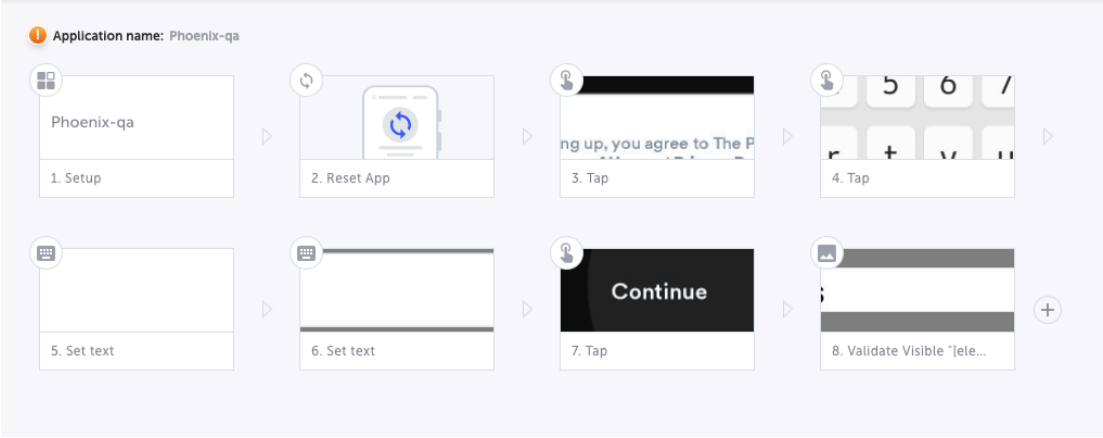
Slika 4.6. Početni zaslon aplikacije.

Validacija vidljivosti naslova *Events* provjerava se koristeći već dostupan korak *Validate element visible*. Predefinirani koraci se mogu koristiti tako da se klikom na dodavanje koraka otvori izbornik te odabere jedan od ponuđenih koraka (Slika 4.7.)



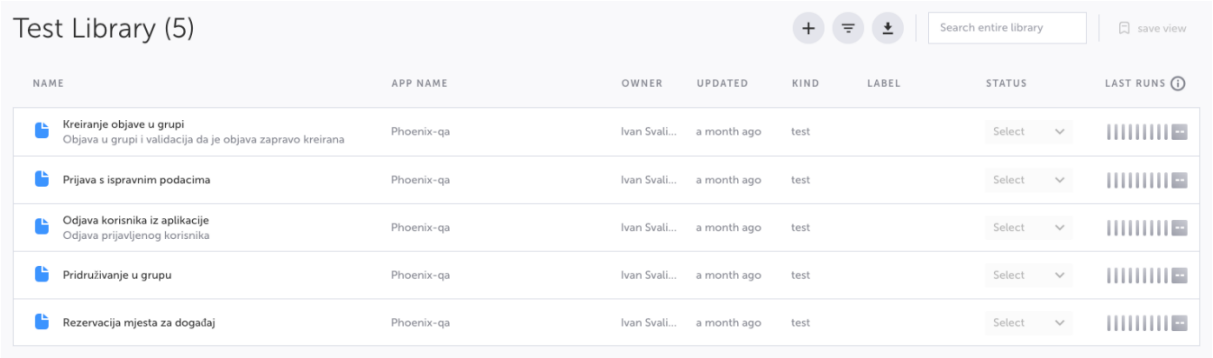
Slika 4.7. Izbornik dostupnih predefiniranih koraka.

Nakon što je dodan posljednji korak validacije naslova *Events* testni slučaj je u potpunosti automatiziran. Svi koraci su vidljivi u listi koraka, koja je prikazana na slici 4.8., te im se može mijenjati redoslijed, brzina izvođenja, dodavanje uvjeta te ostali razni parametri.



Slika 4.8. Lista svih potrebnih koraka za provjeru funkcionalnosti prijave postojećeg korisnika.

Jedan od bitnijih parametara koji se može mijenjati je ponašanje u slučaju da korak nije uspješno izveden. Može se definirati da se cijeli test označi kao neuspješan ili se izvođenje testa nastavi neovisno o rezultatu izvođenja koraka. Bitno je napomenuti da većina testova neće moći nastaviti s izvođenjem ako je jedan od koraka neuspješan. Nakon dodavanja svih koraka test se može spremiti i izvoditi kad je potrebno. Spremljeni test se nalazi u listi s ostalim spremljenim testnim slučajevima (Slika 4.9.).



Slika 4.9. Lista spremljenih testnih slučajeva.

Mobilni testovi u alatu se izvode tako da se otvori uređivač testa i pokrene test klikom na ikonu *play*. Klikom na ikonu *play*, alat počinje izvoditi korake testnog scenarija na uređaju onim redoslijedom kojim su dodani te ako se svi koraci izvedu uspješno testni slučaj se označava uspješno izvedenim.

5. VLASTITO RJEŠENJE ZA AUTOMATIZACIJU TESTIRANJA

Rješenje kreirano za potrebe ovog rada izvodi testove na stvarnim *Android* uređajima i virtualnim *iOS* uređajima. Moguće je postaviti konfiguraciju za izvođenje testova i na fizičkim uređajima koji koriste *iOS* operacijski sustav no zbog zatvorenosti *iOS* okoliša i drugih ograničenja odabrani su virtualni *iOS* uređaji. Rješenje je ostvareno koristeći alat *Appium* i pripadne pogonske programe za *Android* i *iOS* uređaje. Vlastito rješenje je pisano u programskom jeziku *Java* u *IntelliJ* integriranom razvojnom okruženju no iste mogućnosti i funkcionalnosti se mogu ostvariti u drugim jezicima i integriranim razvojnim okruženjima. U rješenju su također korištena različita proširenja koja se nalaze na *Maven* repozitoriju radi dodavanja dodatnih funkcionalnosti u rješenje poput automatskog stvaranja izvještaja o testiranju. Za svrhe praćenja i kontrolu verzija programa korišten je široko poznati *GitHub* alat koji omogućuje pohranu i verzioniranje programa na udaljenim repozitorijima. Kod kreiranog rješenja je u cijelosti dostupan na *GitHub* repozitoriju [11].

5.1. Instalacija potrebnih alata

Za razvoj vlastitog rješenja korišteni su sljedeći alati koje je potrebno instalirati na računalo prije početka razvoja rješenja:

- *Java Development Kit* (*JDK*),
- *Android Studio*,
- *Node,*
- *Appium* poslužitelj,
- *XCUITest Appium* pogonski program,
- *UIAutomator2 Appium* pogonski program,
- *Appium Inspector,*
- integrirano razvojno okruženje.

Java Development Kit, dalje *JDK*, je skup alata i biblioteka potrebnih za razvoj *Java* aplikacija. *JDK* uključuje potrebnu dokumentaciju, prevoditelj (eng. compiler) bez kojeg nije moguće pisati kod za *Java* aplikacije i ostale softvere. Instalacija *JDK*-a se vrši preuzimanjem *JDK* paketa dostupnog na web stranici [12] i pokretanjem preuzetog paketa te prateći instrukcije. *Node* je okruženje za pisanje i izvođenje *JavaScript* koda i aplikacija. U ovom slučaju je

potreban jer je *Appium* napisan u *JavaScript* jeziku te ga se u suprotnom ne bi moglo koristiti. Paketi za instalaciju *Node*-a su dostupni na linku [13]. *Android Studio* je integrirano razvojno okruženje izgrađeno posebno za razvoj aplikacija koje se koriste na *Android* operacijskom sustavu. Instalacija je potrebna jer se zajedno uz *Android Studio* instalira i *Android SDK*. Kao što je *JDK* biblioteka potrebna za razvoj *Java* aplikacija tako je *Android SDK* (eng. *Software Development Kit*) biblioteka potrebna za razvoj *Android* aplikacija. *Android SDK* je potreban za automatizaciju jer omogućava komunikaciju računala i mobilnog uređaja na kojem se izvršavaju testovi. *Android Studio* također ima mogućnost pokretanja emulatora za *Android* uređaje na kojima se mogu izvršavati testovi ako se ne može koristit fizički *Android* uređaj. Paket za instalaciju *Android Studia* je moguće pronaći na web stranici [14], te je sama instalacija vrlo jednostavna prateći instrukcije s web stranice. *Appium Inspector* je jedan od dostupnih alata za pronalaženje identifikatora za neki element u aplikaciji. Postupak instalacije *Appium Inspectora* je u potpunosti objašnjen na *GitHub* stranici alata kojoj je moguće pristupiti preko linka [15]. Nakon postavljanja globalnih varijabli, postupak opisan u poglavlju 5.2., potrebno je instalirati *Appium* poslužitelj i odgovarajuće *Appium* pogonske programe. *Appium* poslužitelj služi kao posrednik između napisanih testnih slučajeva i *Appium* pogonskih programa. Poslužitelj šalje naredbe, koje je tester napisao za neki testni slučaj, pogonskim programima koji ih potom izvode na uređaju. Instalacija *Appium* poslužitelja se vrši preko naredbene linije terminala tako da se upiše naredba vidljiva na slici 5.1.. Nakon izvršavanja naredbe terminal zatraži unos lozinke sustava za nastavak instalacije, ako je unesena lozinka ispravna sustav će

```
1:      sudo npm install -g appium
```

Slika 5.1. Naredba za instalaciju *Appium-a* kroz terminal.

započeti s instalacijom *Appiuma*.

Nakon uspješne instalacije *Appium* poslužitelja potrebno je instalirati i pripadne pogonske programe za *Android* i *iOS* platforme. Instalacija pogonskih programa se također izvodi putem terminala, upisivanjem i izvršavanjem naredbe vidljive na slici 5.2.. Za instalaciju drugog pogonskog programa potrebno je samo zamijeniti naziv pogonskog programa kojeg se želi instalirati, na primjer umjesto *„uiautomator2“* se upiše *„xcuitest“*. Uspješna instalacija pogonskih programa se može provjeriti upisivanjem naredbe sa slike 5.3. koja ispisuje listu svih uspješno instaliranih ali i svih dostupnih *Appium* pogonskih programa.

```
1:      appium driver install uiautomator2
```

Slika 5.2. Naredba za instalaciju *UIAutomator2 Appium* pogonskog programa kroz terminal.

```
ivansvalina — -zsh — 70×13
Last login: Tue Sep  5 14:12:41 on ttys000
ivansvalina@Ivans-MBP-3 ~ % appium driver list
✔ Listing available drivers
- uiautomator2@2.29.4 [installed (npm)]
- xcuitest@4.34.0 [installed (npm)]
- mac2 [not installed]
- espresso [not installed]
- safari [not installed]
- gecko [not installed]
- chromium [not installed]
ivansvalina@Ivans-MBP-3 ~ %
```

Slika 5.3. Lista instaliranih i dostupnih *Appium pogonskih programa*.

Naposljetku je potrebno instalirati integrirano razvojno okruženje unutar kojeg se piše sami kod i razvija vlastito rješenje. U ovom završnom radu je korišteno *IntelliJ IDEA Community Edition* integrirano razvojno okruženje koje je dostupno na web stranici [16]. Nije obvezno instalirati precizno prethodno spomenuto integrirano razvojno okruženje jer se isto rješenje može postići u drugim okruženjima poput *Eclipse* i *Visual Studio Code.*

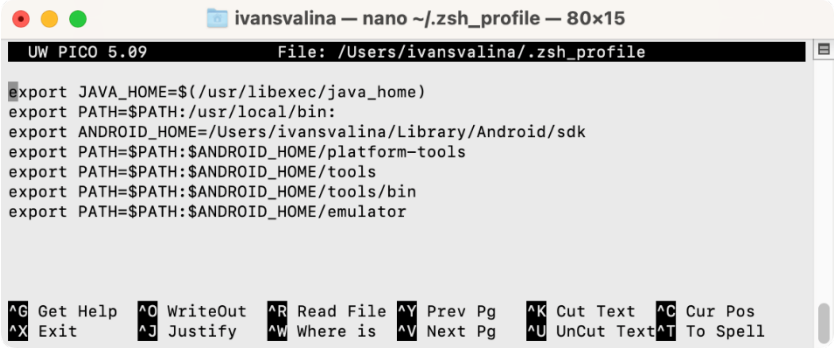
5.2. Postavljanje globalnih varijabli operacijskog sustava

Postavljanje globalnih varijabli je bitan postupak sa svrhom olakšavanja pronalaska izvršnih datoteka i olakšavanjem pokretanja potrebnih programa poput *Node-a* ili *Jave.* Globalne varijable se postavljaju kroz terminal tako da se definiraju cjelovite putanje do izvršne datoteke koju se želi pokrenuti i dodjeljivanja jedinstvenog imena toj putanji. Globalne varijable se zapisuju u *.zsh* ili *.bash* profile na računalu. Ti profili su zaduženi za konfiguraciju ljuske (eng. *shell*) računala koja se koristi za interakciju sa samim operativnim sustavom računala.

Prvo je potrebno otvoriti *.zsh* ili *.bash* profil u uređivaču po izboru izvršavanjem naredbe vidljive na slici 5.4.. Profil u koji će se varijable postaviti ovisi o profilu koji se koristi što se lako iščita iz naslova terminala. Potom je potrebno upisati sva imena varijabli i putanje do izvršnih datoteka koje će ta varijabla pokretati (Slika 5.5.). Imena varijabli su potpuno proizvoljna no putanje do izvršnih datoteka je najbolje kopirati izravno iz mape u kojoj se ta datoteka nalazi.

```
1:      nano ~/.zsh.profile
```

Slika 5.4. Naredba za otvaranje *.zsh* profila u uređivaču *Nano*.



Slika 5.5. Globalne varijable upisane u *.zsh* profil u *Nano* uređivaču.

Naposljetku je potrebno zatvoriti Nano uređivač i spremiti sve promjene te provjeriti jesu li se dodane varijable uspješno spremile u profil. Spremanje dodanih varijabli se vrši izvođenjem naredbe sa slike 5.6. dok se provjera vrši izvođenjem naredbe sa slike 5.7. koja će ispisati sve varijable u sustavu. Ako se novo unesene varijable ne nalaze u popisu svih globalnih varijabli potrebno je provjeriti *.zsh* profil i ispraviti moguće greške te ponovno spremiti promjene.

```
1:        source ~/.zsh.profile
```

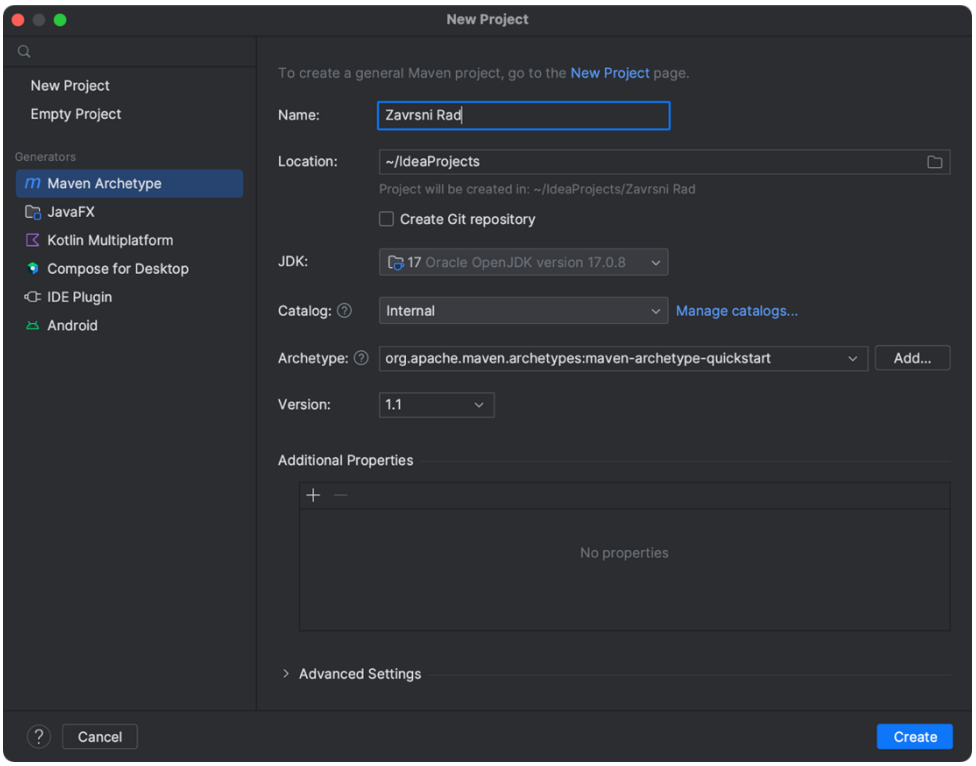
Slika 5.6. Naredba za trajno spremanje promjena napravljenih *u .zsh* profilu.

```
1:        printenv
```

Slika 5.7. Naredba za ispis svih globalnih varijabli sustava.

5.3. Kreiranje i postavljanje projekta

Nakon instalacije navedenih programa i postavljanja globalnih varijabli potrebno je kreirati projekt u *IntelliJ* razvojnom okruženju u kojem će biti kreirani testovi i potrebne funkcije. Kreiranje novog projekta se pokreće na način da se pokrene *IntelliJ* razvojno okruženje i na početnom zaslonu odabere opcija *New Project.* Potom će se otvoriti izbornik s opcijama za kreiranje novog projekta (Slika 5.8.). Potrebno je izabrati *Maven* opciju u izborniku *Generators* jer će se tada kreirati novi *Maven* projekt. Ime projekta je proizvoljno no najbolje je staviti ime koje dobro opisuje novokreirani projekt. Verziju *JDK*-a integrirano razvojno okruženje prepozna samo no u slučaju da verzija nije prepoznata treba odabrati onu verziju koja je preuzeta prilikom instalacije *JDK*-a. Nakon postavljanja svih opcija klikom na opciju *Create* se kreira prazni *Maven* projekt.



Slika 5.8. Izbornik s opcijama za kreiranje novog projekta u *IntelliJ* razvojnom okruženju.

5.4. Programsko pokretanje Appium poslužitelja i kreiranje baznih testova

Za komunikaciju s uređajem tijekom izvođenja testa prvo je potrebno pokrenuti *Appium* poslužitelj koji će uređaju slati instrukcije i naredbe. *Appium* poslužitelj je moguće pokrenuti iz terminala no to nije optimalno jer u tom slučaju tester prije izvršavanja svakog testa ručno treba pokrenuti *Appium* poslužitelj. Iz tog razloga je kreirana klasa *AppiumUtilities* koja sadrži metodu *startAppiumService* za programsko pokretanje poslužitelja (Slika 5.9.). Klasa *AppiumUtilites* također sadrži ostale metode koje koriste i *iOS* i *Android* testovi. Klasom *AppiumServiceBuilder* kreiramo novi *service* te mu se predaju potrebni podaci kao što su putanja do izvršne datoteke za pokretanje *Appiuma*, priključak na kojem *Appium* poslužitelj komunicira s uređajem i *IP* adresu lokalnog *Appium* poslužitelja.

```
public AppiumDriverLocalService startAppiumService(String ipAddress, int
port){
        service = new AppiumServiceBuilder()
                .withAppiumJS(new
File("/usr/local/lib/node_modules/appium/build/lib/main.js"))
                .withIPAddress(ipAddress)
                .usingPort(port)
                .build();

        service.start();
```

```
        return service;
    }
```

Slika 5.9. Metoda za kreiranje novog *Appium* poslužitelja.

Uz klasu *AppiumUtilities* potrebno je kreirati klase koje predstavljaju bazne testove za *iOS* i *Android,* koje nasljeđuju klasu *AppiumUtilities* te koriste njenu metodu *startAppiumService*. Klase *iOSBaseTest* i *AndroidBaseTest* pokreću kreirani *Appium* poslužitelj te mu pridružuju pripadajući *Appium* pogonski program (Slika 5.10.).

```
@BeforeClass(alwaysRun = true)
public void ConfigureAppium() throws IOException {
        Properties properties = new Properties();
        FileInputStream fileInputStream = new
FileInputStream("/Users/ivansvalina/IdeaProjects/ZavrsniRad/src/main/java/R
esources/AndroidData.properties");

        properties.load(fileInputStream);

        service = startAppiumService(
                properties.getProperty("ipAddress"),
                Integer.parseInt(properties.getProperty("port"))
        );

        UiAutomator2Options options = new UiAutomator2Options();
        options.setDeviceName(properties.getProperty("deviceName"))

.setPlatformVersion(properties.getProperty("platformVersion"))
                .setAutomationName("UIAutomator2")

.setApp("/Users/ivansvalina/IdeaProjects/ZavrsniRad/src/test/java/Resources
/Phoenix_Android.apk")
                .autoGrantPermissions();

        driver = new AndroidDriver(service.getUrl(), options);
        onboardingPage = new OnboardingPage(driver);
    }

@AfterClass(alwaysRun = true)
public void tearDown() {
        driver.quit();
        service.stop();
    }
}
```

Slika 5.10. Metode za pokretanje i zaustavljanje *Appium* poslužitelja i pripadajućeg *Appium pogonskog programa*.

Kreiranje pogonskih programa koji će se koristiti u testovima se vrši kreiranjem objekta klase *AndroidDriver* kojoj se predaju *URL* adresa pokrenutog *Appium* poslužitelja i prethodno definirane opcije. Uz *ConfigureAppium* metodu napisana je i metoda *tearDown* koja zaustavlja te gasi *Appium* poslužitelj i kreirani pogonski program. Klasa *iOSBaseTest* je u suštini ista kao i klasa *AndroidBaseTest* te su jedine razlike u tome da se kreira *IOS* pogonski program umjesto *Android* pogonskog programa. Metode u klasama baznih testova imaju anotacije *@BeforeClass*

i *@AfterClass* koje pripadaju *Java* programskom jeziku te ukazuju razvojnom okruženju da se ove metode izvode prije odnosno nakon svih drugih metoda u klasi.

5.5. Kreiranje *Page Object Modela* (*POM*)

Page Object Model, dalje POM, je strukturni obrazac i programski model koji odgovara jednom zaslonu u aplikaciji. *POM* je klasa koja se sastoji od identifikatora svih potrebnih elemenata na zaslonu aplikacije te metoda koje omogućavaju interakciju s tim elementima. Kreirane su dvije bazne *POM* klase, po jedna za *iOS* i jedna za *Android*. Bazne *POM* klase nemaju identifikatore elemenata nego samo propisuju i implementiraju metode koje ostale *POM* klase nasljeđuju i mogu koristiti (Slika 5.11.). Bazna *Android POM* klasa nasljeđuje klasu *AndroidActions* u kojoj su definirane sve metode koje su specifične za *Android* platformu ili se implementacija metode razlikuje od implementacije za *iOS* platformu. Baznu klasu nasljeđuju *POM* klase ostalih zaslona aplikacije, te su ovom strukturom sve metode koje se koriste više puta izdvojene u baznu klasu ili *AndroidActions* odnosno *iOSActions* klasu. Krajnje *POM* klase tada koriste metode *Tap, InputText i GetText* iz bazne *POM* klase za interakciju s elementima na specifičnom zaslonu kojeg ta *POM* klasa predstavlja.

```
public class AndroidBasePage extends AndroidActions {
    AndroidDriver driver;

    public AndroidBasePage(AndroidDriver driver) {
        super(driver);
        this.driver = driver;
        PageFactory.initElements(new AppiumFieldDecorator(driver), this);
    }

    public void Tap(WebElement element) {
        waitForElementToAppearByElement(element, driver);
        element.click();
    }

    public void InputText(WebElement element, String text){
        waitForElementToAppearByElement(element, driver);
        element.sendKeys(text);
    }

    public String GetText(WebElement element){
        waitForElementToAppearByElement(element, driver);
        return element.getText();
    }
}
```

Slika 5.11. Kod *Android* bazne *POM* klase.

POM klasa za zaslon *Events* u aplikaciji sadrži lokatore svih elemenata s istog zaslona koji su korišteni u testovima (Slika 5.12.). Postoje različite metode za lociranje elemenata, a neke od njih su korištenje atributa *accessibility*, *id* elementa i *Xpath* putanja do elementa. Uz varijable

koje predstavljaju elemente, *POM* za zaslon *Events* pruža metode za pritisak na profilnu sliku, pritisak na karticu događaja, navigaciju na sekciju za grupe i metodu za provjeru je li naslov zaslona prikazan (Slika 5.13.). Ove metode se koriste u različitim testovima za interakciju s elementima. Metode koje navigiraju na drugi zaslon aplikacije kao povratnu vrijednost imaju *POM* klasu zaslona koji će se otvoriti nakon interakcije s tim elementom.

```
@AndroidFindBy(id = "the.phoenix.android.qa:id/header")
private WebElement eventsTitle;
@AndroidFindBy(id = "the.phoenix.android.qa:id/profile_avatar")
private WebElement profilePicture;
@AndroidFindBy(id = "the.phoenix.android.qa:id/in_person_events_list")
private WebElement inPersonEventCarousel;
@AndroidFindBy(xpath = "//*[@resource-
id='the.phoenix.android.qa:id/in_person_events_list']//*[@class='androidx.c
ardview.widget.CardView']")
private List<WebElement> eventCards;
@AndroidFindBy(accessibility = "Groups")
private WebElement groupsIcon;
```

Slika 5.12. Elementi sa zaslona *Events* i njihovi pripadni lokatori.

```
public boolean isTitleDisplayed() {
    return eventsTitle.isDisplayed();
}

public ProfilePage tapProfilePicture() {
    Tap(profilePicture);
    return new ProfilePage(driver);
}

public EventDetailsPage tapEventCard() {
    waitForElementToAppearByElement(inPersonEventCarousel, driver);
    Tap(eventCards.get(0));
    return new EventDetailsPage(driver);
}

public GroupsPage NavigateToGroupsSection() {
    Tap(groupsIcon);
    return new GroupsPage(driver);
}
```

Slika 5.13. Metode za interakciju s elementima na *Events* zaslonu.

5.6. Kreiranje testnih slučaja

Nakon pronalaženja svih potrebnih elemenata i metoda za interakciju s tim elementima sve je spremno za pisanje koda za izvođenje testnih slučajeva. Pisanje koda za testni slučaj se svodi na kreiranje jedne klase s jednom metodom koja predstavlja jedan testni slučaj. U toj metodi se koraci testnog slučaja izvode pozivanjem metoda definiranih u *POM* klasama. Na slici 5.14. je prikazan kod za izvođenje testnog slučaja rezervacije mjesta za događaj. Redom se pozivaju metode koje predstavljaju interakciju korisnika s aplikacijom te se na kraju testnog slučaja provjerava uvjet kojim ćemo test označiti uspješnim ili neuspješnim. Prije izvođenja metode *ReserveSpotForRegularEvent* potrebno je izvesti metodu *LogIn* kojoj je dana *@BeforeMethod*

anotacija. Vlastito rješenje za automatizaciju je namjerno strukturirano na način da se metoda *LogIn* mora pozvati prije izvršavanja svakog testa. Ovim pristupom je ostvareno da jedan kreirani test reprezentira i služi kao jedan cijeli *end-to-end* test. Istim procesom pisanja *POM*-a i kreiranja klasa koje reprezentiraju testove automatizirani su ostali testovi iz tablice 3.1..

```
public class ReserveEventSpotTest extends AndroidBaseTest {
    @Test(groups = {"AndroidReg"})
    public void ReserveSpotForRegularEvent(){
        EventsPage eventsPage = new EventsPage(driver);
        EventDetailsPage eventDetailsPage = eventsPage.tapEventCard();

        int initialAttendees = eventDetailsPage.GetNumberOfAttendees();

        eventDetailsPage.TapReserveSpot();

        int currentAttendees = eventDetailsPage.GetNumberOfAttendees();

        Assert.assertEquals(initialAttendees + 1, currentAttendees);
    }
    @BeforeMethod(alwaysRun = true)
    public void LogIn(){
        LogInTests logInTests = new LogInTests();
        logInTests.onboardingPage = new OnboardingPage(driver);
        logInTests.LogIn();
    }
}
```

Slika 5.14. Klasa za testni slučaj rezervacije mjesta za događaj.

5.7. Grupiranje testnih slučajeva u grupe i kreiranje testnog izvještaja

Nakon što su testni slučajevi automatizirani moguće je grupirati više testnih slučajeva koji provjeravaju istu funkcionalnost u testni skup (eng. *test suite*) za provjeravanje samo te jedne funkcionalnosti. Tada se izvođenje svih testnih slučajeva može pokrenuti izvođenjem testnog skupa u kojem se oni nalaze umjesto pojedinačnog pokretanja svakog testa. Također je moguće kreirati jedan testni skup sa svim testnim slučajevima te tada taj testni skup služi kao skup za regresijsko testiranje. U vlastitom rješenju kreirana su dva regresijska testna skupa *AndroidMiniRegression* i *iOSMiniRegression*. Testni skupovi se kreiraju tako da se definira jedna *.xml* datoteka u kojoj navodimo koja grupa testova se izvodi i sve testne klase u kojima su definirani testovi. Također u klasama koje predstavljaju testove u svaku metodu koja pripada jednoj grupi treba dodati naziv te grupe (Slika 5.15.). U oba regresijska testna skupa su testni slučajevi navedeni točno onim redoslijedom kojeg bi korisnik izvodio koristeći aplikaciju prvi put. Time je osigurano da se testovi mogu izvoditi jedan nakon drugog bez potrebe za intervencijom testera između izvođenja testova.

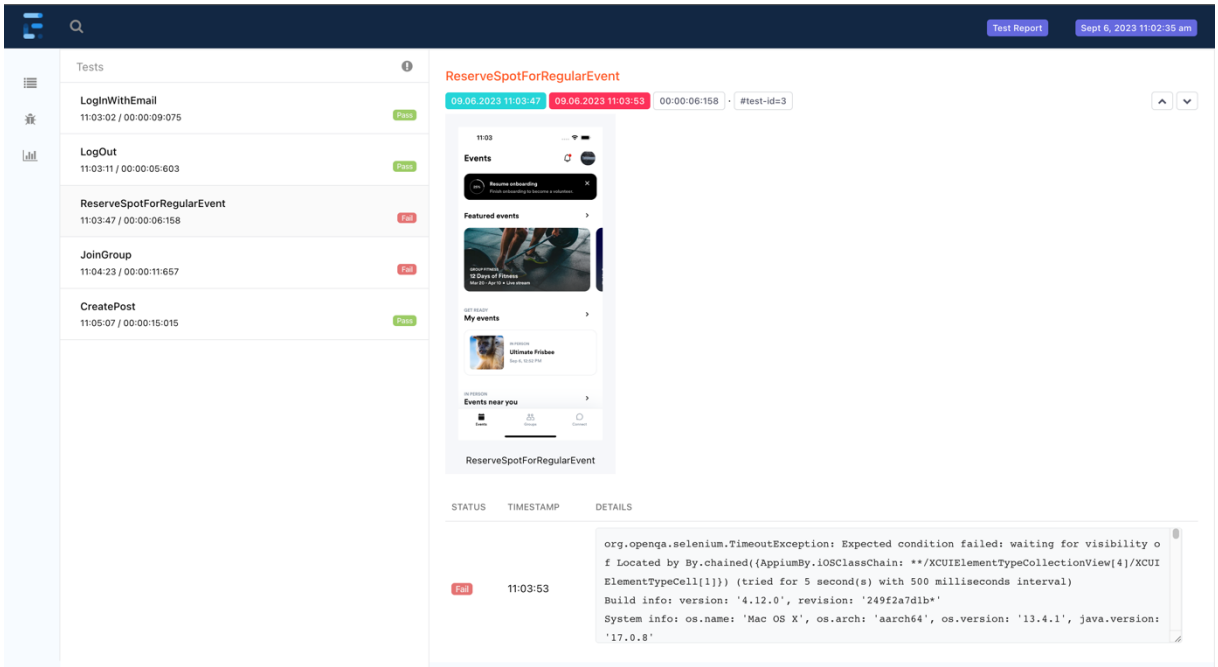
```
@Test(groups = {"iOSReg"})
public void ReserveSpotForRegularEvent(){
    EventsPage eventsPage = new EventsPage(driver);
```

```
    EventDetailsPage eventDetailsPage = eventsPage.tapEventCard();
    eventDetailsPage.TapReserveSpot();

    Assert.assertTrue(eventDetailsPage.IsAddToCalendarVisible());
}
```

Slika 5.15. Metoda *ReseveSpotForRegularEvent* koja pripada grupi *iOSReg*.

Uz definiranje testnih skupova, u vlastito rješenje je dodano proširenje s *Maven* repozitorija pomoću kojega se kreiraju testni izvještaji nakon izvođenja testnog skupa. Za kreiranje izvještaja dodane su dvije nove klase. Prva klasa, klasa *ExtentReporterNG,* je zadužena za kreaciju objekta koji stvara *HTML* datoteku izvještaja. Druga klasa, klasa *Listeners,* implementira metode sučelja *ITestListener* te se pomoću tih metoda dohvaćaju rezultati testova i prosljeđuju objektu klase *ExtentReporterNG.* U *.xml* datoteku testnog skupa je navedena klasa *Listeners* kako bi okruženje znalo koju klasu koristiti za dohvaćanje rezultata. Također je integrirano dodavanje slike zaslona i iznimke zbog koje je test prekinut u slučaju da test nije uspješno izveden. Sa svim rezultatima testova, slikama zaslona i iznimkama, po završetku izvođenja testnog skupa objekt klase *ExtentReporterNG* kreira *indeks.html* datoteku koja predstavlja finalni testni izvještaj. Kreirani izvještaj se otvara u pregledniku po izboru tako da se kopira putanja *indeks.html* datoteke i unese u pretraživač (Slika 5.16.).



Slika 5.16. Testni izvještaj za testni skup *iOSReg* s dva neuspješna testa.

6. EVALUACIJA VLASTITOG RJEŠENJA

U svakom obliku poslovanja utrošeno vrijeme na stvaranje proizvoda je proporcionalno zaradi koje to poduzeće ostvari, isto tako je i u sektoru proizvodnje softvera. Svakom projektnom timu je stoga u cilju što brže i efikasnije ostvariti zadani cilj. U kontekstu ovog rada cilj je automatizacija regresijskog testiranja softvera koji je u procesu razvoja. Kroz rad je lako uočljivo da je vrijeme potrebno za postavljanje i pisanje testova jedna od većih razlika između već postojećih rješenja, poput alata *Tricentis Testim*, i razvoja vlastitog rješenja. Postavljanje projekta i pisanje samih testova u dostupnim alatima, kao što je opisano u 4. poglavlju je vrlo jednostavan i brzi proces. Nasuprot tome postavljanje i pisanje testova u vlastitom rješenju je dugotrajniji proces koji zahtijeva dublje poznavanje različitih tehnologija i koncepata u razvoju softvera. Ova činjenica je jedan od razloga zbog kojeg se timovi i testeri mogu odlučiti za korištenje komercijalno dostupnih alata naspram razvoja vlastitih rješenja.

Oba alata, komercijalno dostupan *Tricentis Testim* i vlastito kreirano rješenje, uspješno izvode sve kreirane testove no postoje razlike u stabilnosti izvođenja testova. Zbog nedostupnosti podataka o načinu lociranja elemenata u alatu *Tricentis Testim* nije poznato na koji način i koji pristup lociranja elemenata alat koristi. Prednost vlastito kreiranog rješenja je ta što tester ima potpuni pristup i kontrolu nad lokatorima koji se koriste u testu. To je bitno zbog činjenice da se razvojem aplikacije lokatori, a shodno tome i putanje do nekih elemenata mogu promijeniti. Testeri zbog toga u vlastitim rješenjima koriste lokatore koji se ne mijenjaju nikad ili vrlo rijetko, poput *accessibility id* lokatora, te time osiguravaju veću stabilnost izvođenja testova.

U vremenima izvođenja testova se očituje stvarna razlika između postojećeg rješenja i vlastitog rješenja. Razlike između prosječnih vremena izvođenja testnih slučajeva su prikazane u tablici 6.1.. Iz tablice se može zaključiti kako vlastito rješenje ima općenito kraće vrijeme izvođenja testova iako razlika nije prevelika. Jedna od prednosti vlastitog rješenja naspram već postojećeg je u tome što se svi testovi počnu izvršavati jednim pokretanjem testne skupine dok to nije moguće u probnoj verziji već postojećeg alata.

Tablica 6.1. Vremena izvođenja testova u alatu *Tricentis Testim* i vlastito razvijenog rješenja.

Testni slučaj	**Vrijeme izvođenja u alatu *Tricentis Testim***	**Vrijeme izvođenja u vlastitom rješenju**
Prijava već postojećeg korisnika	13 sekundi	8 sekundi

Odjava korisnika iz aplikacije	11 sekundi	8 sekundi
Rezervacija mjesta za događaj	15 sekundi	13 sekundi
Pridruživanje korisnika u grupu	20 sekundi	14 sekundi
Kreiranje objave u grupi	26 sekundi	23 sekunde

Jedna od glavnih stavki zbog koje bi se projektni tim ili testeri odlučili za kreiranje vlastitog rješenja je proširivost vlastitog rješenja. Zbog činjenice da testeri sami razvijaju svoje rješenje mogu sami odlučiti koje funkcionalnosti su im potrebne. U slučaju da je potrebna nova funkcionalnost tester ju može implementirati jer ima pristup izvornom kodu i sam je zadužen za razvoj i održavanje rješenja. Jedan primjer funkcionalnosti koja se može dodati u vlastito rješenje je komunikacija s bazom podataka putem koje tester može temeljitije manipulirati podacima koji su vidljivi u aplikaciji. Komercijalno dostupno rješenje nije proširivo od strane testera koji ga koristi. Mogućnosti koje alat pruža su definitivne te u slučaju da je potrebna nova funkcionalnost nije moguće dodati ju kao u vlastito rješenje.

7. ZAKLJUČAK

U različitim modelima razvoja softvera testiranje se izvodi u različitim fazama razvojnog ciklusa. Primjerice u agilnom modelu razvoja cilj je testirati softver istovremeno dok se on razvija, dok se u modelu vodopada cjelokupno testiranje izvodi u zadnjoj fazi razvoja, nakon što su sve druge faze završile. Iako se može izvoditi u različitim fazama, cilj testiranja je osigurati što veću kvalitetu proizvoda koji će biti plasiran krajnjim korisnicima. Stoga je bitno da obavljeno testiranje bude što temeljitije i kvalitetnije. Regresijsko testiranje je jedan od načina i metoda provjere rada svih funkcionalnosti u softveru. Manualna izvedba regresijskog testiranja je proces koji zahtijeva puno vremena kojeg testeri često nemaju na raspolaganju. Na projektima koji razvijaju manje softvere manualna izvedba je moguća no na projektima većih, kompleksnijih i često slojevitijih softvera manualna izvedba potpunog i kvalitetnog regresijskog testiranja često nije moguća. Testeri se stoga okreću automatizaciji repetitivnih testnih slučajeva i onih kojima se provjeravaju glavne funkcionalnosti aplikacije.

Testeri u slučaju automatizacije imaju dvije opcije na raspolaganju, koristiti komercijalno dostupne alate za koje je potrebna kupovina licence ili razvijanje vlastitog rješenja. Oba pristupa imaju svoje prednosti i mane te su one u ovom radu prikazane kroz proces automatizacije odabranih testnih slučajeva.

Prednosti komercijalno dostupnog alata *Tricentis Testim* očitovale su se u lakoći postavljanja alata, kreiranju testnih slučajeva i minimalnoj potrebi tehničkog znanja. Uz alat je dostupan širok raspon dokumentacija, uputa i videozapisa s instrukcijama za korištenje alata te mogućnosti dogovora termina za demonstraciju alata. Najznačajnija mana svih komercijalno dostupnih alata je nemogućnost proširenja mogućnosti koja dovodi testere u poziciju gdje moraju birati između više alata koji nude različite mogućnosti.

U procesu kreiranja vlastitog rješenja postalo je očito da razvoj vlastitog rješenja zahtijeva puno više tehnološkog znanja i puno više vremena nego korištenje dostupnih alata. Iako inicijalno postavljanje projekta zahtijeva više vremena, kasnija modularnost i veća kontrola nad izvođenjem testova postaju prednosti koje se ne mogu zanemariti. Vlastito rješenje je razvijeno koristeći dobro poznate i široko korištene alate i koncepte poput *Appium-a*, *TestNG-a* koji imaju velike zajednice developera i testera koji su otvoreni i voljni pomoći u slučaju nailaska na zapreke i probleme.

Moguće je u budućnosti unaprijediti vlastito rješenje razvijeno u sklopu ovog završnog rada uvođenjem različitih mogućnosti poput implementacije komunikacije s bazom podataka. Komunikacija s bazom podataka je često način kojim se osigurava da se prije svakog testa izvedu metode koje će postaviti željene uvjete. To omogućava kreiranje više testnih slučajeva bez automatizacije tih novih slučaja nego samim mijenjanjem uvjeta prije izvođenja testa čime se jedan test može izvesti s različitim polaznim uvjetima.. Također to osigurava potrebne uvjete za uspješno izvođenje testova tako što testovi postaju neovisni o redoslijedu izvođenja. Još jedno od mogućih proširenja je integracija testova u *CI/CD* (eng. *Continuous Integration/Continuous Development*) cjevovode pomoću dostupnih proširenja i alata kao što su *Jenkins*, *Azure Pipelines* ili *GitHub Actions*.

LITERATURA

[1] „Mobile Operating System Market Share Worldwide“, *StatCounter Global Stats*. https://gs.statcounter.com/os-market-share/mobile/worldwide/ (pristupljeno 05. kolovoz 2023.).

[2] G. Unmesh, *Selenium Testing Tools Cookbook*. Birmingham Mumbai: Packt Publishing, 2012.

[3] „Appium Documentation - Appium Documentation“. https://appium.io/docs/en/2.0/ (pristupljeno 30. lipanj 2023.).

[4] „The Appium Ecosystem - Appium Documentation“. https://appium.io/docs/en/2.0/ecosystem/ (pristupljeno 30. lipanj 2023.).

[5] A. Tsadok, *Pro iOS Testing: XCTest Framework for UI and Unit Testing*, 1st ed. edition. New York, NY: Apress, 2020.

[6] D. Zelenchuk, *Android Espresso Revealed: Writing Automated UI Tests*. Apress, 2019.

[7] S. Shridhar, „Analysis of Low Code-No Code Development Platforms in comparison with Traditional Development Methodologies“, *IJRASET*, sv. 9, izd. 12, str. 508–513, pros. 2021, doi: 10.22214/ijraset.2021.39328.

[8] A. A. Menegassi i A. T. Endo, „Automated tests for cross-platform mobile apps in multiple configurations“, *IET Software*, sv. 14, izd. 1, str. 27–38, 2020, doi: 10.1049/iet-sen.2018.5445.

[9] „National Sober Active Community“, *The Phoenix*. https://thephoenix.org/ (pristupljeno 30. lipanj 2023.).

[10] „Configure Tricentis Mobile Agent“, *Testim*. https://help.testim.io/docs/configure-tricentis-mobile-agent (pristupljeno 30. lipanj 2023.).

[11] Svale09, „Svale09/appium-automation“. 29. kolovoz 2023. Pristupljeno: 06. rujan 2023. [Na internetu]. Dostupno na: https://github.com/Svale09/appium-automation

[12] „Download the Latest Java LTS Free“. https://www.oracle.com/java/technologies/downloads/ (pristupljeno 04. rujan 2023.).

[13] „Download“, *Node.js*. https://nodejs.org/en/download (pristupljeno 04. rujan 2023.).

[14] „Download Android Studio & App Tools“, *Android Developers*. https://developer.android.com/studio (pristupljeno 05. rujan 2023.).

[15] „Appium Inspector“. Appium, 06. rujan 2023. Pristupljeno: 06. rujan 2023. [Na internetu]. Dostupno na: https://github.com/appium/appium-inspector

[16] „Other Versions - IntelliJ IDEA Community Edition“, *JetBrains*. https://www.jetbrains.com/edu-products/download/other-IIE.html (pristupljeno 05. rujan 2023.).

SAŽETAK

Testiranje je kao faza razvojnog ciklusa softvera proces kojim se osigurava što veća kvaliteta softvera. Pri tome se koriste razne strategije i oblici testiranja. Jedno od oblika testiranja je regresijsko testiranje koje provjerava sve funkcionalnosti softvera. Regresijsko testiranje se može izvoditi manualno no zbog repetitivnosti testnih slučaja testeri često automatiziraju regresijsko testiranje. Za automatizaciju testeri mogu koristiti već postojeće alate ili kreirati svoje vlastito rješenje za automatizaciju. U radu su automatizirani testni slučajevi u već postojećem alatu *Tricentis Testim* te je razvijeno novo rješenje za automatizaciju. Vlastito rješenje je ostvareno koristeći alat *Appium* te je potom vlastito rješenje evaluirano u odnosu na alat Tricentis Testim.

Ključne riječi: Appium, automatizacija, mobilna aplikacija, regresija, testiranje.

ABSTRACT

Title: Mobile application regression testing automation

Testing, as a phase of the software development life cycle, is a process in which the highest quality of the software is ensured. In the testing phase various test strategies are being used. One of the used strategies is regression testing in which all functionalities of the software are tested. Regression testing can be conducted manually, but because of the repetitiveness, testers usually opt for automating the regression testing. Testers can use already available solutions and tools or develop their own frameworks. In this thesis a set of five test cases were automated in a commercially available tool Tricentis Testim and the newly developed framework. The framework was developed using *Appium* and was later evaluated in relation to the Tricentis Testim Tool.

Keywords: Appium, automation, mobile application, regression, testing.

ŽIVOTOPIS

Ivan Svalina rođen je 09.08.2000. u Osijeku. Osnovnu školu Dobriše Cesarića završava 2015. godine. Potom upisuje 1. Gimanziju u Osijeku te ju završava 2019. godine. Nakon završetka srednjoškolskog obrazovanja upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, smjer računarstvo. Godine 2022. počinje raditi kao tester u tvrtki Five.