Alati za automatizirano testiranje

Voćanec, Domagoj

Master's thesis / Diplomski 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:200:122403

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-30**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

ALATI ZA AUTOMATIZIRANO TESTIRANJE

Diplomski rad

Domagoj Voćanec

Osijek, 2023.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 20.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Domagoj Voćanec
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1178R, 13.10.2020.
OIB studenta:	50550744095
Mentor:	doc. dr. sc. Ivan Vidović
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Krešimir Romić
Član Povjerenstva 1:	doc. dr. sc. Ivan Vidović
Član Povjerenstva 2:	Josip Zidar, mag. ing. comp.
Naslov diplomskog rada:	Alati za automatizirano testiranje
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U ovom radu potrebno je istražiti dostupne alate za automatizirano testiranje. Nakon istraživanja potrebno je napraviti usporedbu najpopularnijih alata te na proizvoljno odabranom praktičnom primjeru prikazati mogućnosti jednog odabranog alata.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene od strane mentora:	20.09.2023.

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, 29.09.2023.

Ime i prezime studenta:	Domagoj Voćanec
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1178R, 13.10.2020.
Turnitin podudaranje [%]:	8

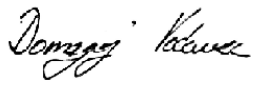
Ovom izjavom izjavljujem da je rad pod nazivom**: Alati za automatizirano testiranje**

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

2. AUTOMATIZIRANO TESTIRANJE 2

2.1. Prednosti automatiziranog testiranja 2

2.2. Alati za automatizirano testiranje 3

2.3. Tablica alata za automatizirano testiranje 8

3. SELENIUM 12

3.1. Selenium WebDriver 12

3.2. Praktični dio – automatsko testiranje web aplikacije 13

4. APACHE JMETER 21

4.1. Testiranje performansi 21

4.2. Praktični dio – automatsko testiranje opterećenja 22

4.2.1. View Results Tree 25

4.2.2. Graph Results 28

4.2.3. Aggregate Graph 30

5. APPIUM 33

5.1. Arhitektura Appiuma 33

5.1.1. *Appium* klijent 34

5.1.2. *Appium* poslužitelj 35

5.1.3. Krajnji uređaj 35

5.2. Praktični dio – automatsko testiranje mobilne aplikacije 37

6. ZAKLJUČAK 51

LITERATURA 52

SAŽETAK 54

ABSTRACT 55

1. UVOD

Zbog ubrzanog razvitka novih tehnologija, razvija se veliki broj softvera. Kako bi se razvio kvalitetan softver koji će zadovoljavati sve zahtjeve korisnika, potrebno je uložiti dosta vremena i resursa. Kako bi se isporučio kvalitetan proizvod koji će biti pouzdan i siguran, potrebno je provesti testiranje softvera. Često se događa da se korisnicima ponudi proizvod koji nije na razini kvalitete koja se očekuje. Zbog slabe testiranosti zna se potkrasti pokoja pogreška koja uzrokuje neočekivane radnje softvera. Kako bi se izbjegle takve pogreške, potrebna je što veća pokrivenost testovima. Naravno, nije moguće testirati cijeli softver, kako bi pokrivenost bila stopostotna, jer bi takvo testiranje potrošilo mnogo vremena i resursa. Stoga je testiranje jedan od najbitnijih procesa kod razvoja softvera.

Testiranje se može podijeliti na ručno testiranje i automatsko testiranje. Ručno testiranje vrlo je zahtjevan posao jer se testovi trebaju izvršavati nakon svake nadogradnje ili izmjene koda. Dakle, takav je način testiranja vrlo iscrpljujuć te troši prekomjerno vrijeme i resurse. Ponekad se testiranja provode jako rijetko i testovi su jednostavni, stoga je ručno testiranje u takvim slučajevima puno lakše i isplativije. S druge strane, kod velikih i zahtjevnih softvera potrebno je uvesti automatizaciju testova. Automatizirano testiranje smanjuje vrijeme testiranja, u odnosu na ručno testiranje, i povećava efikasnost testiranja. Međutim, potrebno je dobro promisliti koji alat odabrati jer neće svaki alat odgovarati svakoj aplikaciji koju se testira.

Danas postoji mnoštvo alata za automatizirano testiranje, a glavna je podjela na komercijalne alate, alate otvorenog koda (eng. o*pen source*), koji su besplatni, i prilagođene alate. Komercijalni su alati licencirani i plaćaju se, stoga je potrebno odabrati pravi alat za automatizaciju kako ne bi došlo do nepredviđenih i nepotrebnih troškova. Alati otvorenog koda besplatni su i dostupni svima, ali nerijetko imaju manje značajki i mogućnosti od komercijalnih. Ako ni jedan od ponuđenih komercijalnih i besplatnih alata ne odgovara potrebama testiranja, tada testni tim razvija prilagođeni alat.

U nastavku ovog rada objašnjeno je automatsko testiranje i njegove prednosti. Zatim će biti opisani neki od najčešće korištenih alata za automatsko testiranje te će se prikazati pojednostavljeni tablični prikaz tih istih alata i alata koji se koriste u praktičnom dijelu ovoga rada. Nakon toga, bit će objašnjeni alati *Selenium*, *Apache JMeter* i *Appium* te će biti objašnjeno i slikama prikazano njihovo korištenje.

2. AUTOMATIZIRANO TESTIRANJE

Automatizirano testiranje nastalo je uslijed povećane složenosti projekata, međuovisnosti i agilnosti. Današnji trendovi nalažu stalnu isporuku nadogradnji aplikacijskih rješenja i njihovu visoku razinu kvalitete i sigurnosti. Danas se sve više vremena troši na testiranje softvera, stoga su potrebni dobri automatizirani testovi kako bi se kontinuirano i konzistento isporučivali softveri. Predviđanja pokazuju kako će se korištenje automatiziranog testiranja povećati za 16 % do 2027. godine.

2.1. Prednosti automatiziranog testiranja

Prva je prednost automatiziranog testiranja ušteda vremena koja odgovara krilatici *vrijeme je novac*. Tri prednosti koje se primijete na samom početku korištenja alata za automatizirano testiranje jesu brzina, ponovljivost i učinkovitost. Pisanje skripti i osmišljavanje testova može potrajati, ali jednom kada su osmišljeni, mogu se stalno koristiti uz povremeno prilagođavanje prema potrebama. Tu je vidljivo da se vrijeme testiranja značajno skraćuje, ali je potrebno i manje vremena za razvoj softvera.

Ponekad je potrebno puno vremena kako bi se neki softver testirao, neki se testovi izvršavaju po nekoliko sati, pa čak i cijeli dan. U tom bi slučaju trebalo zaposliti više testera koji bi radili u smjenama i testirali bez prestanka. Upravo je zbog toga automatizirano testiranje efikasnije jer se može odvijati tijekom noći bez testera koji će sljedeći dan provjeriti dobivene rezultate.

Za razliku od manualnog testiranja, alati za automatizirano testiranje rade točnije i pouzdanije jer rade točno onako kako su isprogramirani, a ponavljajuće radnje uvijek su jednako izvršene. Moguće je postići visoku točnost i preciznost. Ako su skripte ispravno napisane, autor skripte mora osigurati i ispravnost same logike.

Automatiziranim testiranjem može se izvršiti veći broj testova. Moguće je izraditi skup testnih slučajeva i povećati pokrivenost testovima. Zbog toga automatizacija testiranja najbolje funkcionira za velike projekte jer omogućava povećanje kapaciteta radnog opterećenja bez potrebe za povećanjem radnog tima.

Automatizirano testiranje može se izvoditi brzo i paralelno te može pružiti brze povratne informacije. Sposobnost brze povratne informacije vrlo je poželjna u *DevOps* okruženjima gdje procesi traže brzu provjeru valjanosti učinjenih promjena koda kako bi provjerili da se nije što poremetilo. [1]

2.2. Alati za automatizirano testiranje

Kako bi automatizirano testiranje bilo uspješno, bitan je odabir pravog alata za automatizaciju. Uobičajene vrste testiranja zamjenjuju se sustavnim programima testiranja pomoću alata za automatizaciju. Automatizirano testiranje smatra se najučinkovitijim načinom za bolju pokretljivost, učinkovitost i djelotvornost softverske aplikacije. Odabir pravog alata nije lak zadatak jer ne odgovara svaki alat preduvjetima projekta, stoga se moraju ispitati specifičnosti projekta.

Pri odabiru odgovarajućeg alata kroz četiri koraka ispituje se sljedeće (Tablica 2.1.).

Tablica 2.1. Koraci ispitivanja za odabir odgovarajućeg alata. [2]

Razumijevanje zahtjeva projekta	• Automatizirano testiranje povećat će opseg i dubinu testova radi bolje kvalitete projekta. Prije samog početka testiranja, treba razumijeti zahtjeve projekta poput vrste i opsega projekta te sposobnosti programerskog tima.
Razmatranje postojećih alata kao mjerilo	• Tim razmatra alat i njegove specifikacije kao mjerilo za procjenu i određivanje najboljeg alata za određeni projekt. Bitno je odabrati odgovarajući alat koji odgovara zahtjevima projekta.
Određivanje ključnih kriterija	• Jednostavnost izrade i održavanje skripti • Jednostavnost izvršavanja testova • Podrška za desktop, mobilnu i web aplikaciju • Intuitivno izvješće testiranja • *Cros browser testing* • Tehnička podrška i pomoć • Cijena
Pugh Matrix tehnika za analizu	• *Pugh* matrica koristi se kako bi se odabrao najbolji alat za desktop, mobilno i web testiranje. Unutar matrice prikazane su prednosti i mane određenog alata.

Alate možemo podijeliti na sljedeći način [2].

- ***Open source*** – alati otvorenog koda koji su besplatni za korištenje i dostupni za svaku fazu testiranja.
- **Komercijalni alati** – alati koji služe u komercijalne svrhe, imaju više podrške i mogućnosti od *open source* alata.
- **Prilagođeni alati** – kada ne odgovara ni jedan *open source* ili komercijalni alat, tada se razmatra razvoj prilagođenog alata za automatizirano testiranje.

LambdaTest

LambdaTest alat je za automatsko testiranje koji se koristi za stolne i web programe. Može se provesti ručno i automatsko testiranje na više preglednika na kombinaciji stolnih i mobilnih preglednika koji preferiraju jezike poput *Pythona, Jave, Javascripta* itd. *LambdaTest* smanjuje vrijeme testiranja zahvaljujući paralelnom izvođenju testova te ima ugrađene alate za praćenje grešaka koji olakšavaju rješavanje problema. Neke su od ključnih značajki: testiranje stranica s lokalnim hostom i kompatibilnosti preglednika na mreži, pametno vizualno regresijsko testiranje kako bi se osigurao savršen izgled web stranice i ispitivanje kompatibilnosti s internetskim preglednikom. [3, 4]

TestComplete

TestComplete alat je za testiranje automatizacije stolnih, mobilnih i web aplikacija. Uz pomoć ovog alata mogu se graditi i pokretati funkcionalni testovi korisničkog sučelja, kao i provoditi testiranje ključnih riječi i podataka. Nudi se mogućnost snimanja i ponovne reprodukcije ili skriptiranja na više jezika kao što su *Python*, *C++*, *JavaScript* i *VBScript*. Također, postoji podrška za mnoštvo aplikacija poput *.Net*, *iOS* i *Android* aplikacije uz mogućnost regresijskog ispitivanja, ispitivanja putem preglednika i paralelnog ispitivanja. Nudi se i mogućnost poboljšanja testova za više okruženja kako bi se poboljšala kvaliteta softvera. [3, 4]

QMetry Automation Studio

QMetry Automation Studio (QAS) alat je za automatizaciju ispitivanja, upravljanje testovima i analizu kvalitete. Pokreće se pomoću *Eclipse IDE* te *Seleniumom* i *Appiumom* koji su otvorenog koda. Nudi jedinstveno rješenje za *omnichannel*, *multidevice* i *multilocale* podržavajući komponente weba, *native* i *mobile* weba. Moguća je automatizacija skaliranja smanjenjem alata

jer se mogu ponovno upotrijebiti alati za automatizaciju. Skripte su u *C++ Scriptu*, *Pythonu*, *JavaScriptu* i *VBScriptu*, što omogućuje naprednu kodiranu automatizaciju. [3, 4]

TestProject

TestProject platforma je za automatizaciju testiranja na webu i mobilnom uređaju. Omogućava stvaranje i izvršavanje testova na *Windows*, *Linux* i *MacOS* sustavima, a testove je moguće distribuirati lokalno i u oblak. Nudi se i mogućnost snimanja testova bez skripti, napredni SDK za skriptiranje, analiza nadzorne ploče, ugrađene integracije za *BrowserStack, Jenkins, Slack* i druge. [3]

Testsigma

Testsigma alat je za automatizaciju koji je označio početak novog doba automatizacije koje je prikladnije za današnje tržište *Agile* i *DevOps*. Temelji se na umjetnoj inteligenciji i koristi jednostavan engleski jezik za automatizaciju i udovoljava potrebama kontinuirane isporuke. Nudi mogućnost automatizacije web aplikacija, mobilnih aplikacija i API usluga te nudi sustav automatizacije ispitivanja s elementima potrebnim za kontinuirano testiranje. [3]

Worksoft

Worksoft nudi industrijsku platformu za automatizaciju *Agile-plus* i *DevOps* za složene poslovne programe. Nudi platformu za kontinuirano automatiziranje izgrađenu bez koda, udovoljavajući potrebama korisnika s velikim poduzećima, u kojima se moraju testirati ključni procesi u više aplikacija i sustava. *Worksoft Certify* smatra se *zlatnim standardom* za testiranje poslovnih aplikacija, nudi podršku za web aplikacije i aplikacije u oblaku s unaprijed izgrađenim, gotovim optimizacijama. *Worksoft* ima sposobnost testiranja složenih *end-to-end* procesa, mogućnosti automatskog otkrivanja i dokumentiranja, prepoznavanje objekata za *SAP Fiori* i brže izdavanje ažuriranih verzija. [3]

Qualibrate

Qualibrate je jednostavan, prilagodljiv i ima moć integracije s većinom CI/CD alata, a služi za automatizaciju testiranja SAP i web aplikacije, pri čemu se test slučajevi lako održavaju. Najosnovnije implementacije zahtijevaju dobro organizirane timove kako bi se nosili sa složenošću isporuke proizvoda, potreban je jedinstven pristup testiranju, dokumentiranju i

učenju. Kod *Qualibratea* postoji mogućnost snimanja poslovnog procesa, a to postaje temelj za dokumentaciju poslovnog procesa. [3]

***Basis* - svjedočanstvo**

Basis je samo dio platforme *DevOps* i automatizacije testiranja za SAP softver. Koristi RTA (robotsku test automatizaciju) tehnologiju koja podupire SAP regresijske mogućnosti testiranja. Koristeći RTA, tradicionalne skripte za regresijsko testiranje i upravljanje podacima više nisu potrebne, što znači da se mogu eliminirati troškovi, napor i složenost, koji su usko vezani uz učinkovito regresijsko ispitivanje. Neke su od prednosti to što se regresijski testovi mogu izvršavati brže i češće, ubrzana isporuka projekata i ažuriranja te izrada i ažuriranje testne biblioteke. [3]

Subject7

Subject7 platforma je utemeljena na oblaku koja podržava *end-to-end* automatizaciju testiranja. Platforma nudi testiranje s nizom naredbi koje pokrivaju web i web usluge (*Rest, SOAP*), mobilne uređaje, bazu podataka, radnu površinu, opterećenje, sigurnost i testiranje pristupačnosti. Korisnicima omogućuje jednostavno kreiranje i izvršavanje tijekova automatizacije uz minimalnu podršku i obuku, lako integrira druge tehnologije poput *Jire, GitHuba, Jenkinsa, REST-a* i većinu *DevOps* platformi. Podržava paralelno izvršavanje za aktivne i pasivne sigurnosne provjere i dostupan je za upotrebu u javnom oblaku, privatnom oblaku i hibridnom postavljanju. [3, 4]

Appsurify TestBrain

Appsurify TestBrain omogućuje programerima češće testiranje, lakše pronalaženje nedostataka i ubrzano vrijeme ciklusa. Može se uključiti u već postojeće okruženje za testiranje koje se nalazi u sustavu ili oblaku. Alat služi za testiranje strojnog učenja i štedi na vremenima završetka automatskog testiranja. Vraća rezultate testa nakon svakog urezivanja, stavlja karantenu na nestabilne testove radi brže objave bez narušavanja kvalitete. Dizajniran je da se olakša i skrati vrijeme testiranja, sačuva kvaliteta koda i ne naruši struktura. [3]

Micro Focus UFT

UFT ili objedinjeno funkcionalno ispitivanje, ranije poznato kao QTP (eng. *QuickTest Professional*), alat je za testiranje komercijalne automatizacije za funkcionalno testiranje web, desktop i mobilnih aplikacija. *UFT* omogućuje skriptiranje na *JavaScriptu*, *VBScriptu*, *Pythonu* i drugima. Funkcionalno testiranje postaje jednostavnije i prihvatljivije glede troškova. *UFT* je kompatibilan s više preglednika i platformi, također ima potencijal otkrivanja i prepoznavanja objekata temeljeno na slici, što olakšava ispravljanje istih, zatim višestruka rješenja testiranja i optimizirano distribuirano testiranje. [3, 4]

Ranorex

Ranorex je popularan izbor tvrtki za testiranje jer se koristi za testiranje desktop, web i mobilnih aplikacija. Ovaj alat koristi *Selenium WebDriver*, stoga ima mogućnosti prepoznavanja objekata u korisničkom sučelju, intuitivno sučelje i potpuni IDE (eng. *Integrated Development Environment*). Upravo ga takvo korisničko sučelje čini jednostavnim za korištenje za početnike, ali opet i moćnim alatom za stručnjake koji ga koriste za automatizaciju s potpunim IDE-om za *C#* ili *VB.Net* i otvorene API-je. Alat ima opciju generiranja skripti i reprodukciju, što olakšava testiranje automatizacije, zatim ima spremište objekata za dijeljenje te mogućnost testiranja paralelno i na više platformi. Također, prednosti su što ima prilagodljivo izvješće o ispitivanju, višekratne module koda za smanjenje održavanja ispitivanja, video izvještaj o izvršenju testa radi lakšeg pregleda i što se integrira s drugim alatima kao što su *Jira*, *Jenkins*, *Git*, *Travis CI*, *TestRail* i drugi. [3, 4]

Katalon Studio

Katalon studio sveobuhvatno je rješenje za automatizaciju testiranja web, desktop i mobilnih aplikacija i API-ja. Može se koristiti na *Windows*, *Linux* i *Mac* operacijskim sustavima, ali također podržava testiranje *Android* i *iOS* aplikacija, kao i web aplikacija na svim preglednicima i API usluga. *Katalon Studio* podržava vanjske *Java* knjižnice i koristi samo nekoliko jezika za pisanje poput *Groovyja*. Neke su od prednosti *Katalon Studija* generiranje testnih skripti, stvaranje testnih slučajeva, snimanje radnji, izvršavanje testova i izvještavanje o rezultatima. Postoji i mogućnost integriranja u CI/CD procese, kao i integriranja s mnogim drugim alatima poput *Jire*, *qTesta*, *Kobitona*, *Gita*, *Slacka* i drugih. [3, 4]

2.3. Tablica alata za automatizirano testiranje

U ovom poglavlju su prikazani alati koji su ranije navedeni i alati koji su korišteni kasnije u praktičnom dijelu. Tablica prikazuje pojednostavljeni prikaz alata za automatsko testiranje. Unutar tablice prikazani su logo pojedinog alata, godina nastanka, tip testne aplikacije, platforme koje podržavaju pojedini alat, programske jezike koji su podržani i vrstu alata prema tržištu (*open source*/komercijalni).

Tablica 2.2. Tablica alata za automatizirano testiranje – 1. dio.

Logo				
Naziv	LambdaTest	TestComplete	QMetry Automation Studio	TestProject
Godina	2017.	1999.	2008.	2015.
Tip testne aplikacije	Desktop i web programi i mobilni preglednik	Desktop Mobilne Web	Web Mobilne	Web Mobilni
Platforma	Windows Mac iOS Android	Desktop Web Mobilne	Windows Mac	Windows Linux Mac
Programski jezik	Python Java Javascript ...	Python Javascript VBScript ...	C++ Python Javascript VBScript	C# Java Python
Open source	✗	✗	✓	✓
Komercijalni	✓	✓	✗	✗

Tablica 2.3. Tablica alata za automatizirano testiranje – 2. dio.

Logo				
Naziv	Testsigma	Worksoft	Qualibrate	Basis
Godina	2019.	1998.	2013.	2017.
Tip testne aplikacije	Web Mobilne	Desktop Web Mobilne	Web SAP	SAP
Platforma	Windows Linux Mac Android iOS	Windows Mac	Windows Mac Linux	Windows Mac Linux Android iOS
Programski jezik	Python Java C# ...	Java Python C++ .Net ...	Java Python Javascript	Python Java
Open source	✓	✗	✗	✗
Komercijalni	✗	✓	✓	✓

Tablica 2.4. Tablica alata za automatizirano testiranje – 3. dio.

Logo				
Naziv	Subject7	Appsurify TestBrain	Micro Focus UFT	Ranorex
Godina	2011.	2017.	1976.	2007.
Tip testne aplikacije	Desktop Web Mobilne	Testiranje strojnog učenja	Desktop Web Mobilne	Desktop Web Mobilne
Platforma	Cloud	Cloud ili Sustav	Windows Mac	Windows Linux Mac
Programski jezik	Java	...	JavaScript VBScript C++ Python	C# VB.Net
Open source	✗	✗	✗	✗
Komercijalni	✓	✓	✓	✓

Tablica 2.5. Tablica alata za automatizirano testiranje – 4. dio.

Logo				
Naziv	Katalon Studio	Selenium	Apache JMeter	Appium
Godina	2016.	2004.	1998.	2011.
Tip testne aplikacije	Desktop Web Mobilne	Web	Web	Mobilne
Platforma	Windows Linux Mac Android iOS	Windows Linux Mac Android iOS	Windows Linux Mac	Windows Android iOS
Programski jezik	Groovy Java	Java PHP Python C#	Java Groovy BeanShell	Java JavaScript PHP Ruby Python C#
Open source	✓	✓	✓	✓
Komercijalni	✓	✗	✗	✗

3. SELENIUM

Selenium je prvi testni alat za automatizaciju koji koriste svi alati vezani uz testiranje web aplikacija i smatra se industrijskim standardom za testiranje automatizacije web aplikacija. Ovaj alat kompatibilan je s više programskih jezika i ostalim okvirima za automatsko testiranje te se može izvršiti na više preglednika i operativnih sustava. Pomoću ovog alata mogu se izraditi skripte za automatizaciju koje su skalabilne u različitim okruženjima i omogućuju regresijsko testiranje, istraživačko testiranje i brzu reprodukciju grešaka. *Selenium* nudi fleksibilnost testerima koji imaju iskustva u programiranju i skriptiranju koja nije moguća kod nekih alata. *Selenium* se sastoji od četiri komponente koje pokrivaju različite potrebe za testiranjem, a prema literaturi [5, 6] to su: *Selenium IDE (Integrated Development Environment), Selenium RC (Remote Control), Selenium Grid* i *Selenium WebDriver*.

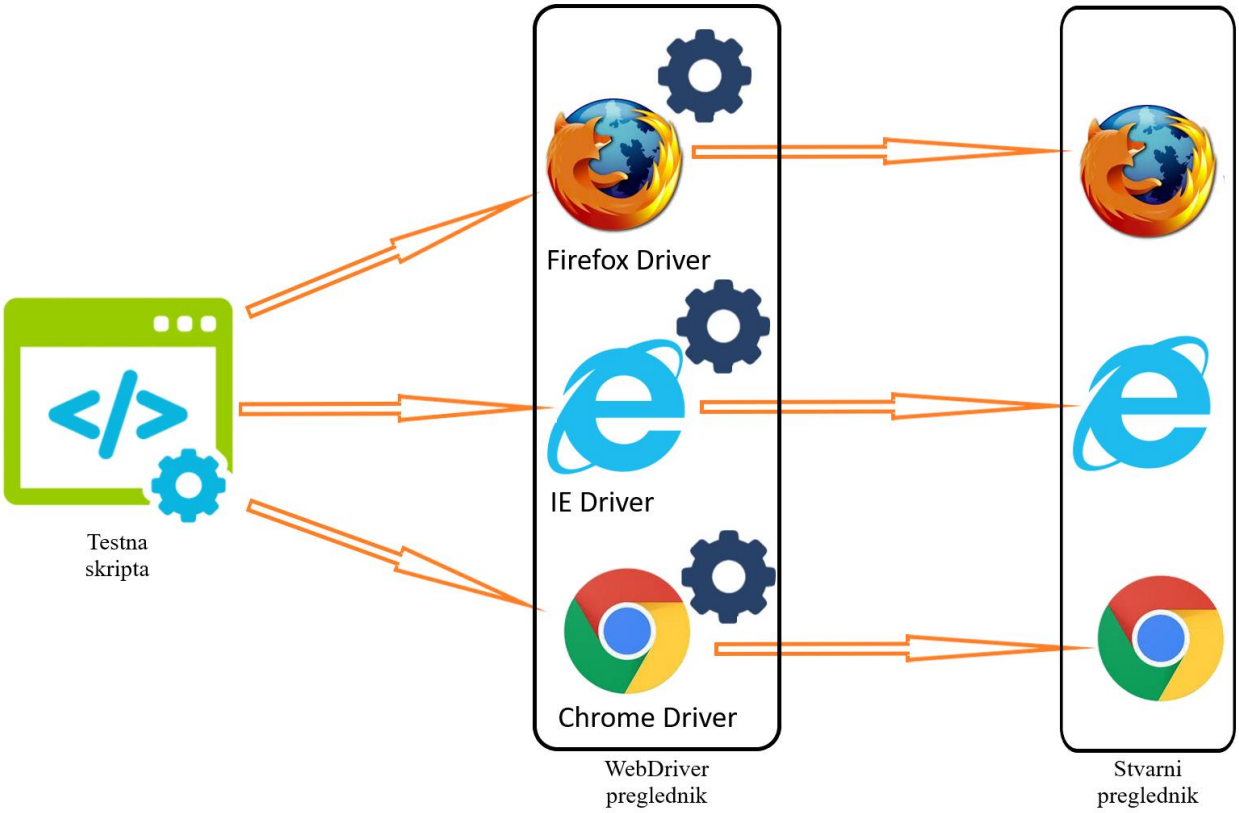
- ***Selenium IDE*** koji je najjednostavniji i najlakši za učenje i korištenje zbog čega se koristi kao prototip. Koristi se za brzo stvaranje skripti za reprodukciju *bugova* i istraživačko testiranje te za jednostavno snimanje i reproduciranje interakcije s preglednikom.
- ***Selenium RC*** dugo je bio vodeći okvir testiranja koji podržava programske jezike poput *PHP-a, Pythona, Jave, C#-a, Rubyja* i *Perla*.
- ***Selenium Grid*** koji se koristi zajedno sa *Selenium RC-om* za pokretanje paralelnih paketa testova istovremeno na različitim strojevima, što skraćuje vrijeme izvršavanja testova. Omogućuje testiranje na različitim verzijama preglednika i na više platformi.
- ***Selenium WebDriver*** upravlja komunikacijom s preglednikom uz stabilan i moderan pristup automatizaciji preglednika te podržava programske jezike kao i *Selenium RC* (*PHP, Python, Java, C#, Ruby, Perl*).

3.1. Selenium WebDriver

Kao što je ranije navedeno, *Selenium WebDriver* popularni je okvir za automatizaciju testiranja web aplikacija koji je dostupan za mnoge programske jezike te je moguće izvršiti testove na svim popularnim preglednicima. Testovi se izvršavaju prema principu navedenom u nastavku.

- Tester pokreće testnu skriptu koja daje naredbe *WebDriveru* što da radi na aplikaciji.
- *WebDriver* pokreće preglednik jednako kao da stvarni korisnik pristupa pregledniku, a izvan toga pokreće se interakcija s preglednikom.

- Po završetku generiraju se rezultati testiranja i spremaju se u određenu datoteku.



Slika 3.1. Arhitektura *Selenium WebDrivera*.

3.2. Praktični dio – automatsko testiranje web aplikacije

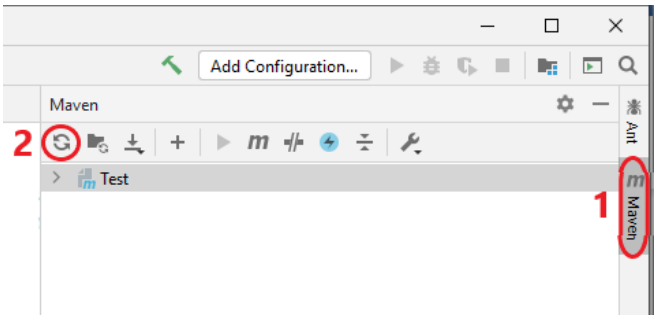
U ovom je poglavlju prikazano korištenje *Selenium WebDrivera* za automatsko testiranje web aplikacije. Alati potrebni za testiranje jesu: *development kit*, IDE i *web driver*. U ovom testiranju korištena je *Java JDK 14.0.2* kao *development kit*, *IntelliJ IDEA 2020.2.4* kao IDE i *Chrome driver*. Korisnik prvo preuzima *Chrome driver*, u skladu s verzijom preglednika, te ga sprema u željenu mapu. U ovom slučaju to je mapa *C:\Users\Desktop\Driver*. Zatim slijedi instalacija programa *Java JDK* i *IntelliJ IDEA*. Nakon toga, korisnik dodaje putanju one mape gdje je spremio *Chrome driver* (ili neki drugi *driver* za drugi preglednik) u sistemsku varijablu putanja okoliša. Nakon što se to izvrši, potrebno je ponovno pokrenuti računalo.

Nakon što se računalo ponovno pokrenulo, korisnik pokreće program *IntelliJ IDEA*. Zatim korisnik odabire karticu *File* u alatnoj traci, u padajućem izborniku odabire *New* i u kranjem padajućem izborniku odabire *Project* kako bi kreirao novi projekt.

U sljedećem koraku otvara se prozorčić u kojem korisnik odabire vrstu projekta, u ovom je slučaju to *Maven* projekt koji koristi *Selenium* i *TestNG* ovisnosti. Nakon odabira vrste projekta, korisnik lijevim klikom miša klikne gumb *Next*. [7]

Sljedeći prozor korisniku nudi unos naziva projekta i odabir lokacije za pohranu projekta. Ovisno o vrsti programa ili verziji koja se koristi, možda će biti potrebno unijeti *GroupId* i *ArtifactId*. U ovom slučaju oni su automatski popunjeni nakon što se unese naziv projekta. Nakon što je sve popunjeno, klikom na gumb *Finish* kreira se željeni projekt.

Nakon što se novi projekt učita i program se otvori, *Maven* ovisnosti automatski se uvoze u novonastali projekt. U slučaju da *Maven* ovisnosti nisu dodane u projekt, potrebno je u desnoj alatnoj traci odabrati opciju *Maven* i zatim ikonu za *Reload*, kako je prikazano na slici 3.2.. Ako se pojavi mala ikonica u desnom dijelu programa, klikom na nju može se pokrenuti *Reload* ili *Reimport*. Ikonica je prikazana dolje na desnoj slici 3.3..



Slika 3.2. *Maven* opcija i *Reload* ikona.

Slika 3.3. Reload ikona.

Također, nakon što je kreiran novi projekt, kreira se i *pom.xml* datoteka u koju se dodaju *Maven* ovisnosti poput biblioteka *selenium-java*, *testng*, *maven-surfire-report-plugin* i *maven-compiler-plugin* i njihove verzije koje su dostupne na poveznici https://mvnrepository.com/ .

U nastavku je prikazana *pom.xml* datoteka s dodanim ovisnostima koje su potrebne za testiranje (Slika 3.4.).

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.example</groupId>
    <artifactId>Testiranje</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
```

```
            <artifactId>selenium-java</artifactId>
            <version>3.12.0</version>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>6.14.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-report-plugin</artifactId>
            <version>2.22.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.22.1</version>
                <configuration>
                    <suiteXmlFiles>
                        <suiteXmlFile>testng.xml</suiteXmlFile>
                    </suiteXmlFiles>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Slika 3.4. Primjer koda *pom.xml* datoteke.

Nakon što su ovisnosti dodane u *pom.xml* datoteku, potrebno je kreirati testnu klasu. Testna se klasa kreira tako što se otvori mapa *src*, a zatim mapa *test* u kojoj se nalazi mapa *java*. Desnim klikom miša na mapu *java* otvara se izbornik gdje korisnik odabire *New* i u novom izborniku odabire *Java Class*. Nakon toga, korisnik unosi naziv nove *Java* klase i tipkom *Enter* potvrđuje odabir i naziv klase.

Nakon što je kreirana nova *Java* klasa i nazvana *TestClass*, potrebno je uvesti (*import*) *selenium* i *testng* ovisnosti. Zatim, kod globalnih varijabli deklarira se *Webdriver* varijabla i testna URL varijabla. U postavkama testa unutar anotacije *@BeforeMethod*, koja se izvršava prije *@Test*,

stvara se novi *ChromeDriver* i navigira prema testnoj URL varijabli, odnosno početnom prozoru na *Google Chrome* pregledniku. *@Test* anotacija važan je dio gdje se piše kod, odnosno poslovna logika. Određeni kod koji se nalazi unutar anotacije *@Test* treba automatizirati tako što se kod ubacuje u metodu testiranja. Metoda testiranja izvršava anotaciju *@Test* prosljeđivanjem atributa. Nakon testiranja, slijedi anotacija *@AfterMethod* koja se izvršava nakon *@Test* [8]. U ovom slučaju anotacija *@AfterMethod* služi za zatvaranje preglednika i završavanje sesije. U nastavku nalazi se kod koji je korišten u *Java* klasi nazvanoj *TestClass* (Slika 3.5.).

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;
public class TestClass {
    //-------------------Global Variables----------------------------------
    //Deklariranje Webdriver varijable
    public WebDriver driver;
    //Deklariranje testne URL varijable
    public String testURL = "http://www.google.com";
    //----------------------Test Setup-----------------------------------
    @BeforeMethod
    public void setupTest() {
        System.setProperty("webdriver.chrome.driver",
"C:\\Users\\Domagoj\\Desktop\\Driver\\chromedriver.exe");
    //Stvaranje novog ChromeDriver
        driver = new ChromeDriver();
        driver.navigate().to(testURL);
    }

    @Test
    public void testAliExpress() throws InterruptedException {

        Thread.sleep(2000); //2000 ms odgoda
        WebElement agreement = driver.findElement(By.id("L2AGLb"));
        agreement.click();  //click za klik kursorom
        Thread.sleep(2000);
        WebElement searchTextBox = driver.findElement(By.name("q"));
        searchTextBox.sendKeys("aliexpress");
        searchTextBox.submit();    //tipka enter
        Thread.sleep(3000);
        WebElement clickAliExpress =
driver.findElement(By.xpath("//*[@id=\"tads\"]/div/div/div/div/div[1]/a/div[1
]/span"));
        clickAliExpress.click();
        Thread.sleep(2000);
        WebElement acceptCookies =
driver.findElement(By.xpath("//*[@id=\"gdpr-new-
container\"]/div/div[2]/button[2]"));
        acceptCookies.click();
        Thread.sleep(2000);
        WebElement clickPhones = driver.findElement(By.xpath("//*[@id=\"home-
firstscreen\"]/div/div/div[2]/div/div[2]/dl[3]/dt/span/a"));
        clickPhones.click();
```

```
        Thread.sleep(2000);
        WebElement clickSearchBar =
driver.findElement(By.name("SearchText"));
        clickSearchBar.sendKeys("samsung");
        clickSearchBar.submit();
        Thread.sleep(2000);

        WebElement testLink =
driver.findElement(By.xpath("//*[@id=\"root\"]/div[1]/div/div[1]/div[1]/div[2
]/div[1]/div/ul/li[1]/ul/li[1]/a")
        );
        Assert.assertEquals(testLink.getText(), "Cellphones");
        System.out.print(testLink.getText());
    }
    @Test
    public void testLinkPrijava() throws InterruptedException {

        Thread.sleep(2000);
        WebElement agreement = driver.findElement(By.id("L2AGLb"));
        agreement.click();
        Thread.sleep(2000);
        WebElement searchTextBox = driver.findElement(By.name("q"));
        searchTextBox.sendKeys("ferit");
        searchTextBox.submit();
        Thread.sleep(2000);
        WebElement clickOnFerit =
driver.findElement(By.xpath("//*[@id=\"rso\"]/div[1]/div/div/div/div/div/div/
div/div[1]/div/a/h3"));
        clickOnFerit.click();
        Thread.sleep(2000);
        WebElement clickProjekti =
driver.findElement(By.xpath("//*[@id=\"videoTraka\"]/div/h3/a[3]"));
        clickProjekti.click();
        Thread.sleep(2000);
        WebElement studenti =
driver.findElement(By.xpath("//*[@id=\"izbornik\"]/div[1]/div/ul/li[3]/a"));
        studenti.click();
        Thread.sleep(2000);
        WebElement fakultet =
driver.findElement(By.xpath("//*[@id=\"izbornik\"]/div[1]/div/ul/li[5]/a"));
        fakultet.click();
        Thread.sleep(2000);

        WebElement testWeb =
driver.findElement(By.xpath("//*[@id=\"izbornik\"]/div[1]/div/ul/li[7]/a")
        );
        Assert.assertEquals(testWeb.getText(), "Prijava");
        System.out.print(testWeb.getText());
    }
    @Test
    public void testPortanova() throws InterruptedException {

        Thread.sleep(1000);
        WebElement agreement = driver.findElement(By.id("L2AGLb"));
        agreement.click();
        Thread.sleep(1000);
        WebElement searchTextBox = driver.findElement(By.name("q"));
        searchTextBox.sendKeys("portanova");
        searchTextBox.submit();
        Thread.sleep(1000);
        WebElement clickPorta =
```

```
driver.findElement(By.xpath("//*[@id=\"rso\"]/div[1]/div/div/div/div/div/div/
div/div[1]/div/a/h3"));
        clickPorta.click();
        Thread.sleep(1000);
        WebElement clickNavLink =
driver.findElement(By.xpath("/html/body/nav/div[2]/div[1]/ul/li[2]/a"));
        clickNavLink.click();
        Thread.sleep(2000);
        WebElement clickNavLink2 =
driver.findElement(By.xpath("/html/body/nav/div[2]/div[1]/ul/li[3]/a"));
        clickNavLink2.click();
        Thread.sleep(2000);


        WebElement testWeb = driver.findElement(By.xpath("//*[@id=\"js-
events\"]/a[1]/div/h2")
        );
        Assert.assertEquals(testWeb.getText(), "Nagradna igra \"Back to
School by Portanova!\"");
        System.out.print(testWeb.getText());
    }

    @AfterMethod
    public void teardownTest() {
        driver.quit();
    }
}
```

Slika 3.5. Primjer koda za *Selenium* testiranje web aplikacija.

Unutar *Selenium WebDrivera* osnovna je metoda *findElement()*, dok je osnovna klasa objekta *WebElement* [9]. Princip rada *Seleniuma* jest dohvaćanje elemenata s obzirom na neku specifičnost, kao što su to na primjer *id*, *name*, *xpath*, *linkText*, *tagName*, *partialLinkText*, *className* i *cssSelector* [10]. Metoda *findElement()* vraća objekt klase *WebElement* koji može pohraniti svaki HTML element. *Thread.sleep* se koristi za čekanje između dvije akcije i mjeri se u milisekundama. Testiranje se provodi tako što se prvo pokreće *webdriver* i preglednik koji su postavljeni u anotaciji *@BeforeMethod*. Zatim se izvodi kod unutar anotacije *@Test* i na kraju svakog testa poziva se anotacija *@AfterMethod* i preglednik se zatvara.

Prvi test nazvan testAliExpress testira internet trgovinu AliExpress. Testiraju se gumbi na navigacijskoj traci, tražilica i gumb nazvan *Cellphones*. Kao rezultat vraća se tekst *Cellphones*, koji se ispisuje na kraju testiranja. Drugi test nazvan testLinkPrijava testira web stranicu FERIT. Testiraju se kartice na navigacijskoj traci i na kraju se testira link *Prijava*. Rezultat testiranja je tekst *Prijava* koji se ispisuje na kraju testiranja. Treći test nazvan testPortanova testira web stranicu Portanova. Testiraju se kartice na navigacijskoj traci i na kraju se testira link od obavijesti *Nagradna igra \ Back to School by Portanova!*. Rezultat koji se ispisuje nakon ovog testiranja je tekst *Nagradna igra \ Back to School by Portanova!* .

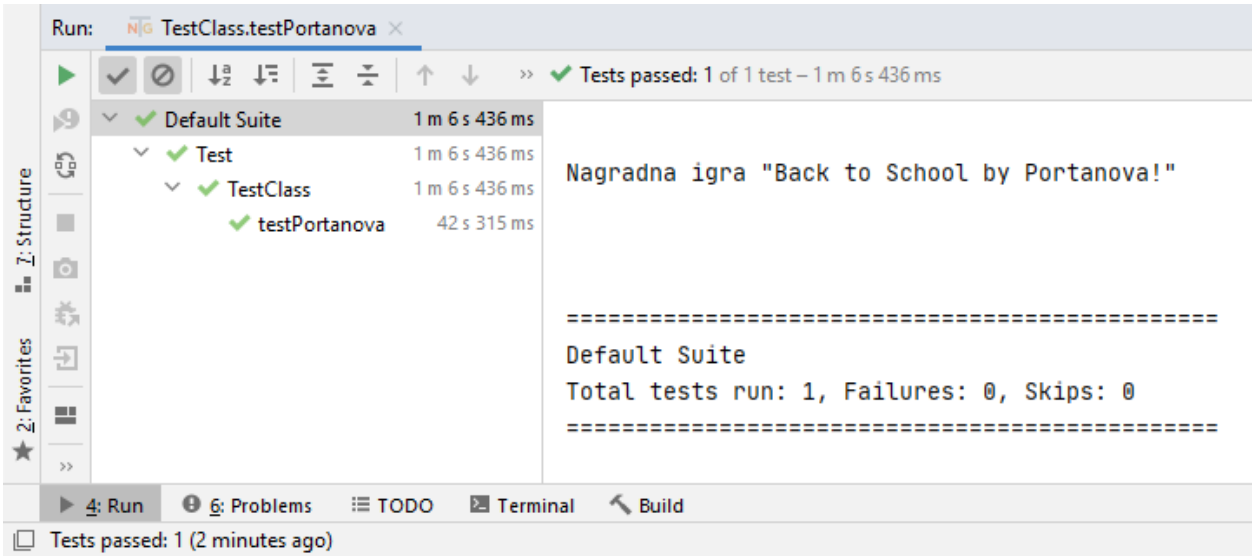
Nakon što je korisnik napisao kod u testnu klasu i otklonio sve pogreške, ako ih je bilo, potrebno je izgraditi projekt tako što korisnik u alatnoj traci odabire karticu *Build* i zatim u novom izborniku odabire *Build Project*.

Nakon što je projekt izgrađen, potrebno je desnom tipkom miša kliknuti na testnu klasu *TestClass* i u izborniku odabrati *Create 'TestClass'...* .Zatim se otvara prozorčić za konfiguraciju pokretanja testa gdje korisnik treba kliknuti gumb *OK*.

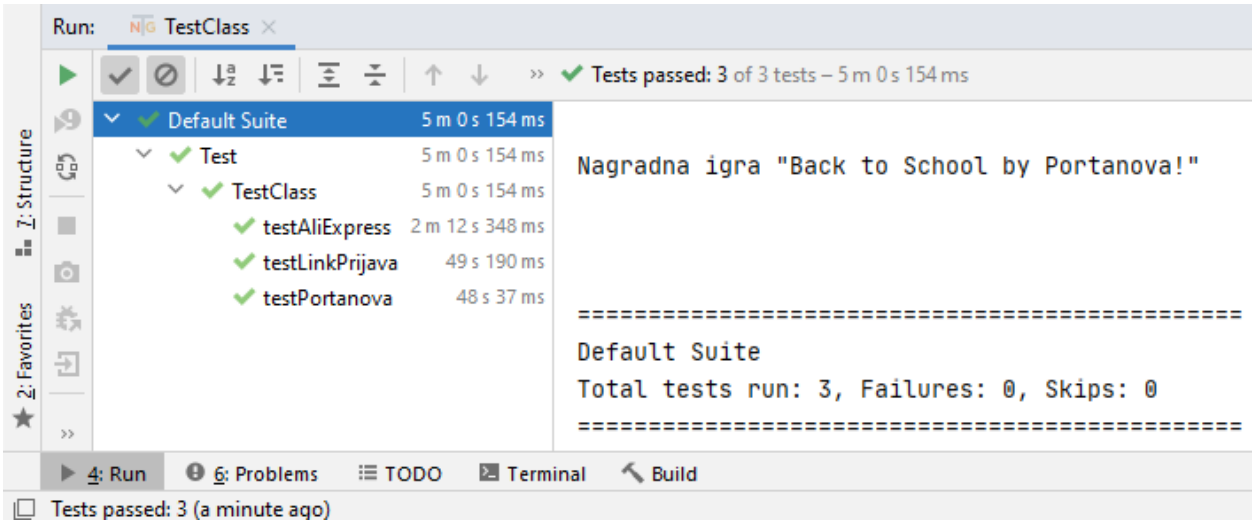
Kada je test kreiran, pojavljuje se ikona za pokretanje testa u gornjem desnom kutu programa. Odabirom kartice *Run* u alatnoj traci otvara se izbornik u kojem korisnik odabire *Run 'TestClass'* kako bi započelo testiranje. Testiranje se može pokrenuti i klikom na ikonu *Run* koja se nalazi u gornjem desnom kutu programa.

Nakon pokretanja testiranja, pokreće se *ChromeDriver* te se nakon toga otvara preglednik *Google Chrome* i započinje automatsko testiranje. U gornjem lijevom kutu preglednika pisat će poruka da je preglednik *Chrome* pod kontrolom softvera za automatsko testiranje.

Po završetku testiranja, u *Run* odjeljku prikazuju se rezultati testiranja. S lijeve strane odjeljka vidi se da su svi testovi prošli. Prva slika prikazuje treći testni slučaj koji je pojedinačno testiran (Slika 3.6.). Druga slika prikazuje sve testove koji su pokrenuti i koji su prošli (Slika 3.7.). Nakon svakog pozitivno završenog testa, ispisuje se tekst koji se dohvaća iz svojstava testnog elementa. Također, nakon svakog testa, preglednik se zatvara i ponovo pokreće za potrebe sljedećeg testa.



Slika 3.6. Pozitivan ishod jednog testiranja.



Slika 3.7. Pozitivan ishod cjelokupnog testiranja.

4. APACHE JMETER

Apache JMeter softver je za automatizaciju otvorenog koda zasnovan na *Javi* te namijenjen za testiranje opterećenja i performansi. Fokusira se na testiranje web aplikacija i usluga, ali može se koristiti i za jedinstveno testiranje i ograničeno funkcionalno testiranje. Vrlo je popularan izbor programera i testera jer ima korisničko sučelje koje je jednostavno za korištenje. *JMeter* ima arhitekturu koja je usredotočena na dodatke pomoću kojih nudi mnoštvo značajki izvan okvira. Alat je integriran s CI/CD alatima kao što je *Jenkins* te podržava mnoštvo aplikacija, protokola i poslužitelja poput FTP-a, TCP-a, weba, protokola pošte, *Java* objektata, baza podataka i drugih. Koristi se za simulaciju opterećenja na jednom poslužitelju, grupi poslužitelja, mreži ili objektu kako bi se testirala i analizirala ukupna izvedba pri različitim vrstama opterećenja.

Sastoji se od vrlo moćnog i opremljenog Test IDE-a koji nudi mogućnost snimanja plana testiranja, izgradnju i otklanjanje pogrešaka. Zatim ima mogućnost CLI načina ili načina naredbenog retka za učitavanje testa iz bilo kojeg operacijskog sustava koji je kompatibilan s *Javom* te mogućnost izvlačenja podataka iz popularnih formata poput *HTML*-a, *JSON*-a, *XML*-a i drugih tekstualnih formata. *JMeter* ima potpunu prenosivost i potpuni višenitni (eng. *multithreading*) okvir koji omogućuje istovremeno uzorkovanje različitih funkcija od strane zasebnih grupa niti i istovremeno uzorkovanje od strane više niti. Također, ima mogućnost ponovnog reproduciranja rezultata testiranja, predmemoriranja i izvanmrežne analize. [11]

JMeter ima vrlo proširivu jezgru, što znači da omogućuje neograničene mogućnosti testiranja, skriptirane uzorke te mogućnost priključenja vremenskih mjerača u svrhu odabiranja statistike opterećenja. Također omogućuje veliku proširivost koja je omogućena uz pomoć dodataka za analizu podataka i vizualizaciju, dinamički unos testa pomoću funkcija i manipulaciju podacima, integraciju kroz biblioteke otvorenog koda u svrhu kompatibilnosti s drugim alatima poput *Mavena*, *Gradlea* i *Jenkinsa*.

4.1. Testiranje performansi

Testiranje performansi jedna je od najkritičnijih faza lansiranja proizvoda. Potrebno je provjeriti i na kraju potvrditi učinkovitost nekog proizvoda. Testiranje performansi odnosi se na testiranje programa ili softvera pod određenim radnim opterećenjem i praćenje kako reagira. Provjerava se radi li aplikacija prema očekivanjima pod tim opterećenjem ili se ponaša neočekivano. Pri testiranju testira se pouzdanost, brzina, upotreba resursa i vrijeme odziva, a cilj je testiranja

pronaći i ukloniti potencijalne otpore softverskoj aplikaciji. Glavni je fokus pri testiranju usmjeren na sljedeća tri faktora.

- Brzina – odgovor aplikacije
- Skalabilnost – maksimalno opterećenje korisnika
- Stabilnost – provjera kod različitih opterećenja

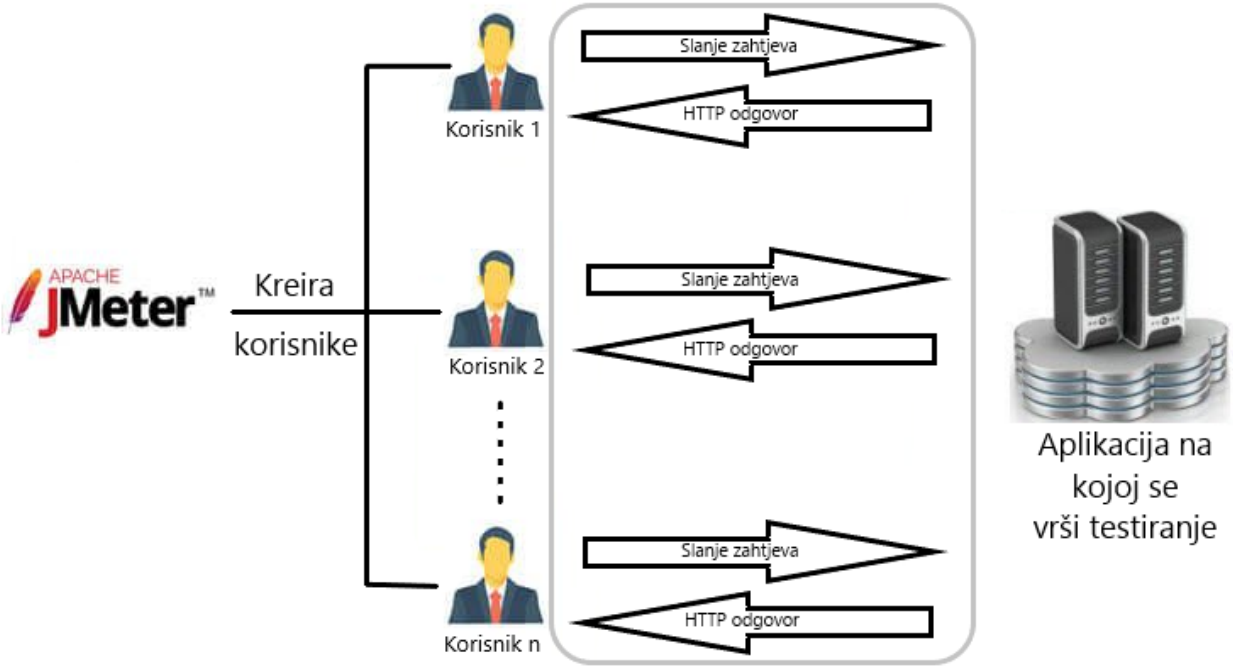
Postoji šest vrsta testiranja performansi navedenih u nastavku.

- **Test opterećenja** – Vrsta testa kojim se testira izvedba aplikacije pri normalnoj i vršnoj upotrebi. Učinkovitost se provjerava prema vremenu odziva i dosljednosti na različitim korisničkim opterećenjima.
- **Test količine** – Provjerava se izvedba sustava u vezi s količinom podataka. Kod izvođenja ovog testa u bazu podataka unosi se velika količina podataka.
- **Stres test** – Vrsta testa u kojem se pronalaze različiti načini kako bi se preopteretio sustav, odnosno testira se koliko opterećenje sustav može podnijeti i koliko opterećenje može izdržati u inkrementalnom pristupu.
- **Test kapaciteta** – Provjerava se sposobnost aplikacije da zadovolji poslovni kapacitet te se testiranje provodi pazeći na broj korisnika i izglede aplikacije kako bi to podnijela.
- **Test pouzdanosti** – Testiranje u kojem se provjerava hoće li se aplikacija vratiti u normalno stanje nakon nekog neočekivanog stanja te se procjenjuje vrijeme potrebno da se sustav vrati u normalno stanje.
- **Test skalabilnosti** – Utvrđivanje može li se aplikacija proširiti ako je to potrebno, a potreba za proširenjem dolazi zbog povećanja korisničkog opterećenja.

4.2. Praktični dio – automatsko testiranje opterećenja

U ovom poglavlju se izvršava test opterećenja. Testiranje će se izvršiti uz pomoć alata *Apache JMeter*. Test opterećenja izvodi se tako što se simulira višestruki pristup korisnika nekoj web usluzi istovremeno (Slika 4.1.). Ako aplikacija podržava određeni broj (npr. 1000) korisnika istodobno, aplikacija će raditi ispravno, a u suprotnom će slučaju aplikacija usporiti ili srušiti se. Stres test izvodi se tako što se testira kapacitet učitavanja na web poslužitelj. Kada se prekorači maksimalni kapacitet učitavanja, web poslužitelj počinje usporeno raditi i proizvoditi pogreške. Svrha takvog stres testa jest pronaći maksimalno opterećenje koje web poslužitelj može podnijeti.

Takvim se testiranjima pronalazi i utvrđuje maksimum rada sustava i postavljaju se ograničenja unutar kojih će aplikacija uredno raditi. Vrlo je bitno da brzina učitavanja i performanse sustava zadovoljavaju visoke zahtjeve korisnika.



Slika 4.1. *JMeter* simulira opterećenje velikog broja korisnika na web aplikaciju.

Kako bi se program *JMeter* pokrenuo, potrebno je pomoću *Naredbenog retka* otići u datoteku gdje se nalazi program i pokrenuti ga pomoću naredbe *JMeter*. Nakon što se pokrenio *Apache JMeter*, korisnik može promijeniti naziv testnog plana.

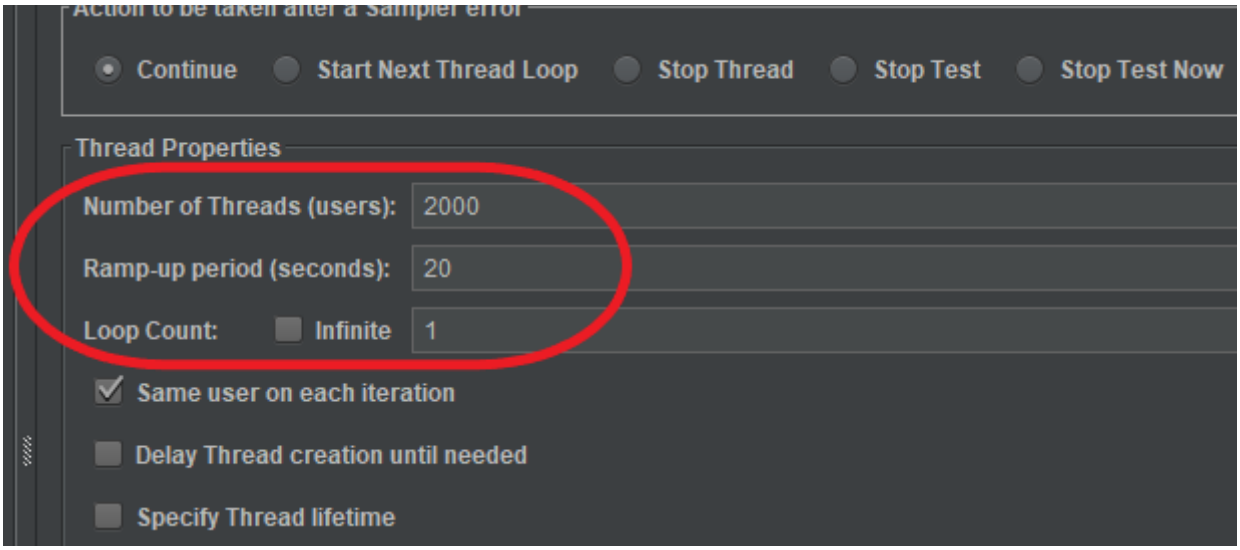
Zatim je potrebno desnom tipkom miša kliknuti na naziv projekta i u izborniku odabrati *Add*, potom u novom izborniku odabrati *Threads (Users)* i na krajnjem izborniku odabrati *Thread Group*. Nakon toga dodaje se nova grupa elemenata koja je vidljiva u stablu projekta.

Nakon što je dodana nova grupa elemenata, otvaraju se postavke *Thread Groupe*. Korisnik ima mogućnost promijeniti naziv grupe niti, a zatim slijedi odabir radnje koju treba poduzeti nakon što dođe do pogreške uzorkivača (eng. *sampler*). U ovom testiranju odabrana je radnja *Continue*, što znači da će se testiranje nastaviti nakon pogreške uzorkivača. Zatim je potrebno postaviti svojstva niti koja predstavljaju sljedeće (Tablica 4.2.).

Tablica 4.2. Postavke niti.

Number of Threads (users)	*Ramp-up period (seconds)*	*Loop Count*
2000 Broj korisnika koji se spaja na određenu web stranicu	20 Period pokretanja svih korisnika odnosno niti, računa se na sljedeći način: $\frac{20\ sekundi}{2000\ korisnika} =$ $0{,}01\ sekundi/korisnik$ Kašnjenje između svakog korisnika iznosi 0,01 sekunda, što znači da se šalje 100 zahtjeva korisnika u 1 sekundi.	1 Broj ponavljanja spajanja pojedinog korisnika. Postoji mogućnost odabira beskonačnog broja ponavljanja.

Pri kraju postavljanja niti, korisnik ima mogućnost odabira načina rada poput istog korisnika u svakoj iteraciji, odgode stvaranja niti dok ne bude potrebno i specificiranja životnog vijeka niti gdje se može odrediti trajanje i odgoditi pokretanje niti (Slika 4.3.). [12]



Slika 4.3. Postavke *Thread Groupe*.

Sljedeći element koji se dodaje jest *HTTP Request Default* koji služi za postavljanje web mjesta koje se testira. Dodaje se tako što se desnom tipkom miša klikne na testni plan *Load Test* pa se u

izborniku odabere *Add* koji otvara novi izbornik gdje se odabire *Config Element* te se otvara krajnji izbornik gdje se odabire *HTTP Request Defaults*. [12]

U ovom je testiranju *YouTube* web mjesto koje se testira te se pod *Server Name or IP* unosi web adresa *www.youtube.com*, a za protokol unosi se *http*. Taj se element treba nalaziti odmah ispod testnog plana (*Load Test*), odnosno drugi po redu u stablu projekta. Ako nije na drugom mjestu, potrebno je lijevom tipkom miša povući i spustiti odmah poslije testnog plana (eng. *Drag and drop*).

Zatim se unutar *Thread Group*e dodaje element *HTTP Request* tako što se klikne desnom tipkom miša na *Thread Group* i otvori izbornik te se odabere *Add* koji otvara novi izbornik gdje korisnik odabire *Sampler* i u krajnjem izborniku odabire *HTTP Request*. [12]

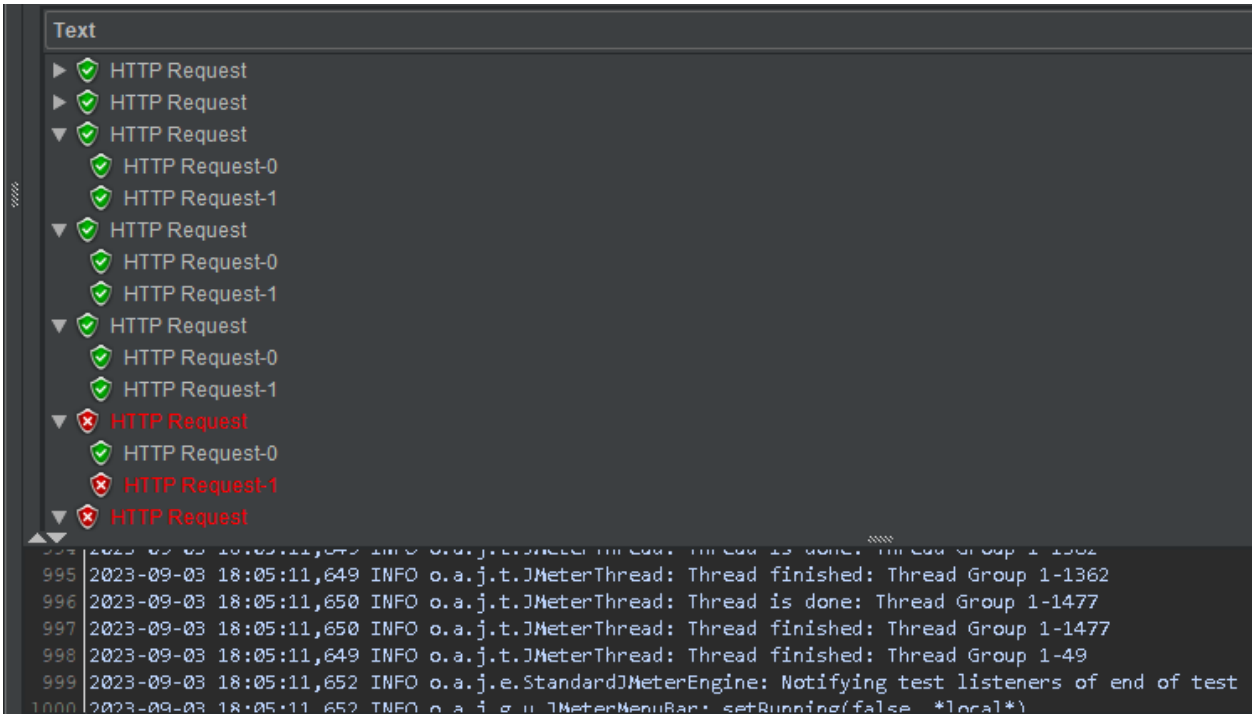
Kod postavljanja *HTTP Requesta* za *Method*, odabrana je metoda *GET*, a za *Path* odabran je *shorts*. *Path* označava krajnju točku na koju se želi poslati zahtjev, što znači da će u ovom slučaju *JMeter* kreirati URL zahtjev *http://www.youtube.com/shorts*. Ako je polje *Path* prazno, *JMeter* stvara URL zahtjev samo na *http://www.youtube.com*.

Nakon što je postavljeno web mjesto koje se testira i odabrana krajnja točka, što je opcionalno, potrebno je dodati slušatelje (eng. *Listeners*). Njih se dodaje tako da se klikne desnom tipkom miša na *Thread Group* gdje se otvara izbornik u kojemu se odabire *Add*. Zatim se otvara novi izbornik u kojemu se odabire *Listener* i u krajnjem se izborniku odabire *View Results Tree*. Pomoću ovog slušatelja, rezultati testiranja prate se redom kako su zahtjevi poslani te se rezultati mogu zapisati u datoteku po želji i moguće je naknadno čitati rezultate iz datoteke. Također, i drugi se elementi slušatelja (eng. *Listener*) dodaju na isti način, ovisno što korisnik želi koristiti kao *Listener*. U ovom testiranju kao slušatelj koristi se još *Aggregate Graph* i *Graph Results* koji grafički prikazuje rezultate testiranja. Radi usporedbe rezultata testiranja, testira se još opterećenje web lokacija *www.google.com* i *www.youtube.com*. Nakon što je testiranje web lokacije *www.youtube.com/shorts* završilo, web lokacije se dodaju u *HTTP Request Defaults* tako što se unutar *Server Name or IP* unese druga web lokacija. [13]

4.2.1. View Results Tree

View Results Tree slušatelj je koji provjerava zahtjev i odgovor te uspoređuje zahtjev i stvarni rezultat. Uzorci se prikazuju redoslijedom kojim su generirani iz pripadajuće skripte i pružaju se parametri i podatci za svaki uzorak. Dostupni su različiti formati poput teksta, *CSS Selector Testera*, *HTML*-a, *Regexp testera*, *Boundary Extractor Testera*, *JSON Path Testera*, *XPath*

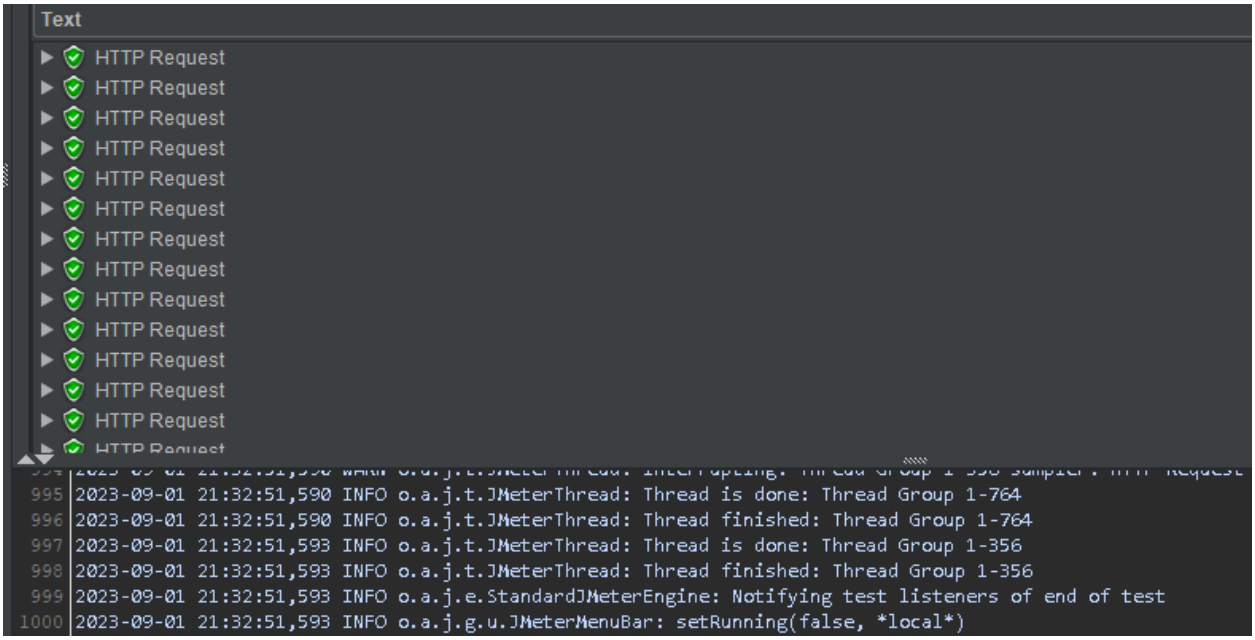
testera, JSON-a, HTML Source Formatteda, XML-a, XPath2 Testera, JSON JMESHPath Testera i *HTML*-a (resursi za preuzimanje). Koristi se za osnovno testiranje te troši veliki broj izvora, kao što su CPU i memorija. Prikaz slušatelja *View Results Tree* koji nakon testiranja pristupa web lokacijama *www.youtube.com/shorts*, *www.youtube.com* i *www.google.com* prikazan je na sljedećim slikama: Slika 4.4., Slika 4.5., Slika 4.6.. Rezultati ovise o nekoliko čimbenika, kao što su trenutačno opterećenje web stranice, brzina interneta i performanse računala. [14, 15]



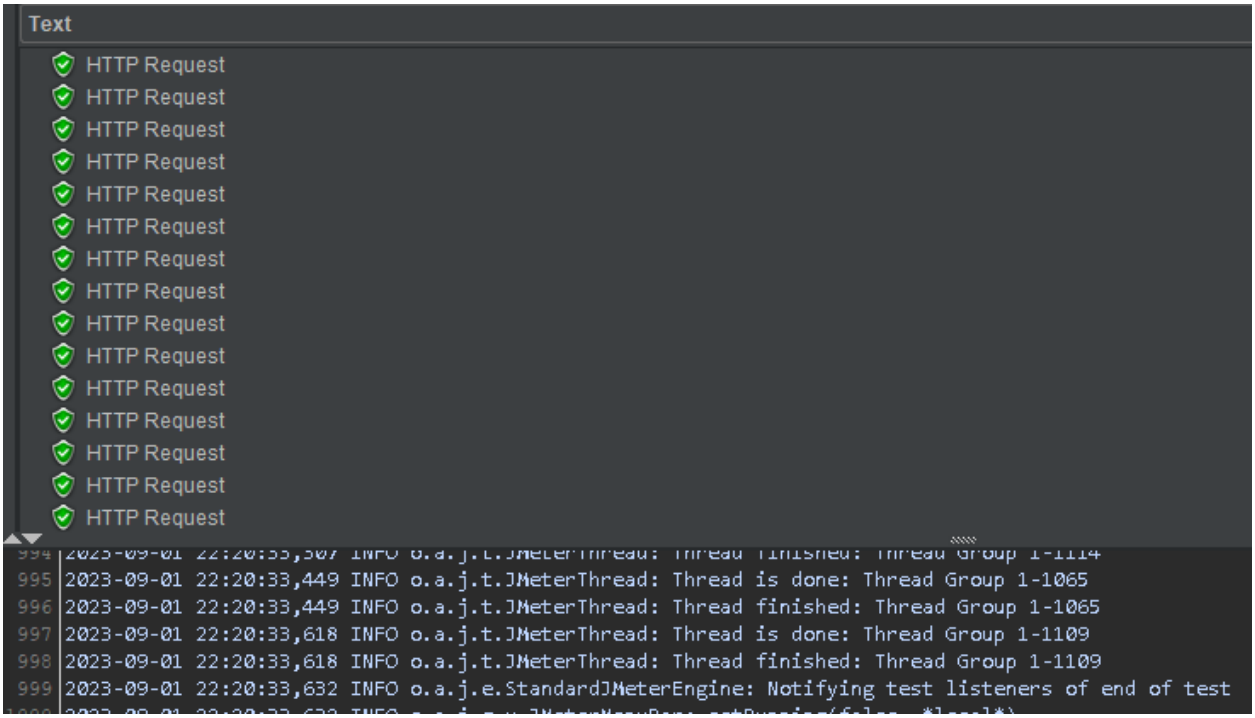
Slika 4.4. Rezultati testiranja opterećenja za web lokaciju *www.youtube.com/shorts* prikazani u slušatelju *View Results Tree*.

Kao što je vidljivo iz gornje slike, neki su zahtjevi uspješno poslani, ali ima i nekih zahtjeva koji nisu poslani zbog loše internetske veze ili opterećenja web stranice. Može doći do pogreške i zbog loših performansi računala.

Na sljedeće dvije slike prikazani su pristupi web stranicama *YouTube* i *Google*. Kao što je vidljivo iz priloženih slika, testiranje je prošlo uredno za većinu poslanih zahtjeva prema web stranici *YouTube*, dok su za *Google* web stranicu svi zahtjevi prošli uredno bez pogreške (Slika 4.5., Slika 4.6.).



Slika 4.5. Rezultati testiranja opterećenja za web lokaciju *www.youtube.com* prikazani u slušatelju *View Results Tree*.



Slika 4.6. Rezultati testiranja opterećenja za web lokaciju *www.google.com* prikazani u slušatelju *View Results Tree.*

4.2.2. Graph Results

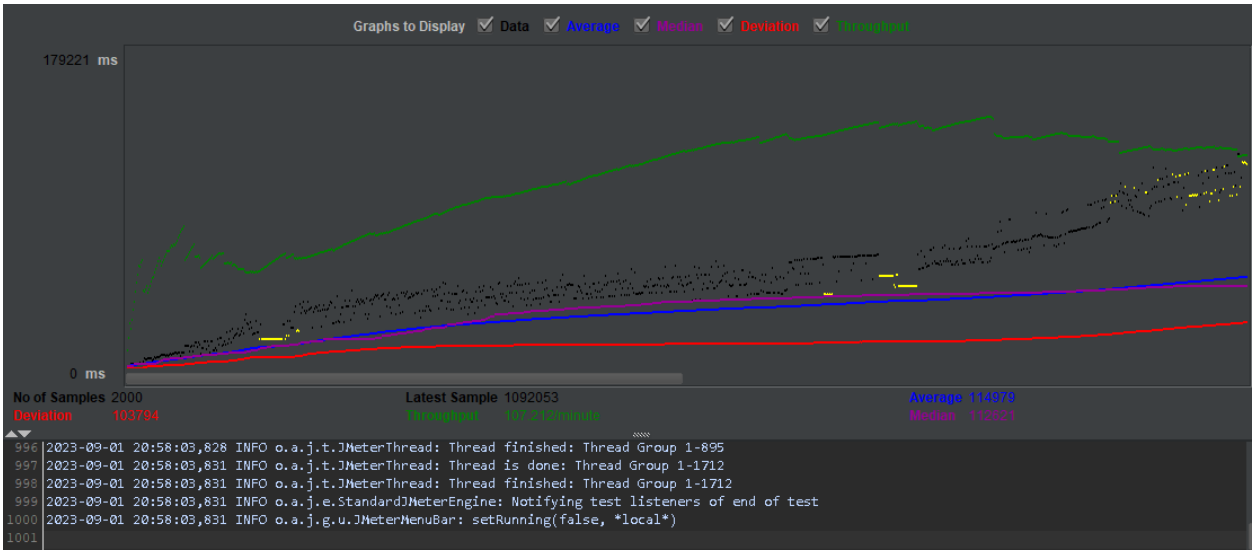
Graph Results iscrtava grafikon između vremena odgovora u milisekundama (y-os) i proteklog vremena (x-os). Linije grafikona prikazuju broj uzoraka, prosječno proteklo vrijeme, devijaciju, propusnost i srednju vrijednost. (Tablica 4.7.)

Tablica 4.7. Objašnjenje legende u grafikonu.

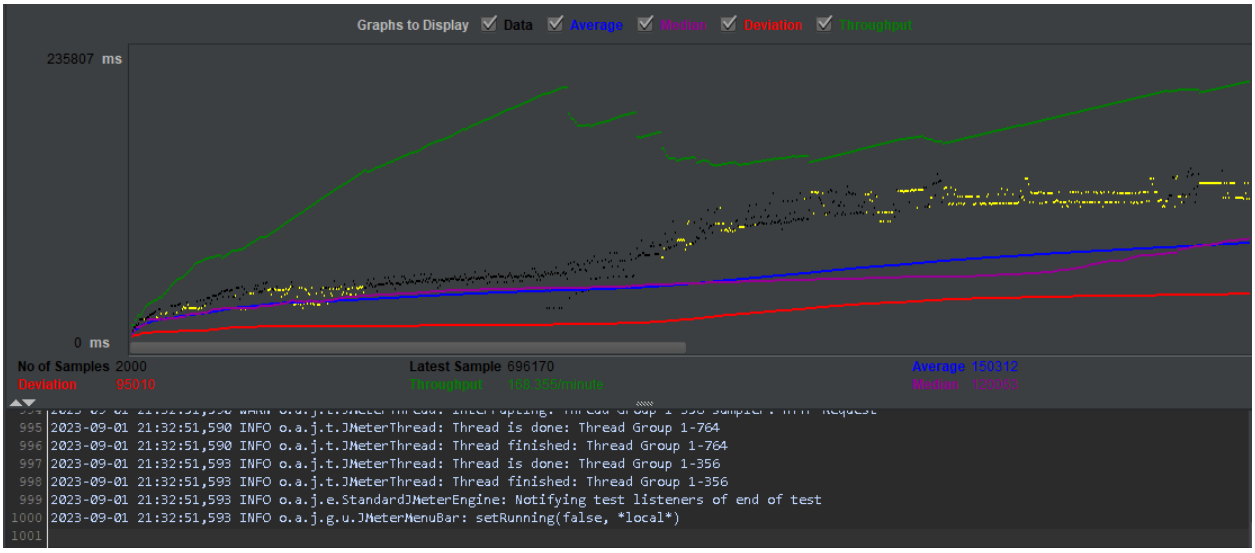
No of Samples	Broj uzoraka koji se obrađuje
Latest Sample	Posljednje proteklo vrijeme u milisekundama
Average	Prosječno proteklo vrijeme odgovora u milisekundama
Deviation	Odstupanje u vremenu odgovara u milisekundama
Throughput	Propusnost - broj uzoraka obrađenih u minuti
Median	Srednja vrijednost

Propusnost je najvažniji parametar koji predstavlja stvarni broj zahtjeva u minuti koje je poslužitelj obradio. Prednost je takvog izračuna što broj uzoraka u minuti predstavlja stvarnu propusnost poslužitelja. Povećavanjem broja niti ili smanjenjem kašnjenja može se otkriti maksimalna propusnost poslužitelja. Što je veća propusnost i što je manja devijacija, odnosno odstupanje od prosjeka, to je bolja izvedba poslužitelja. [14,15]

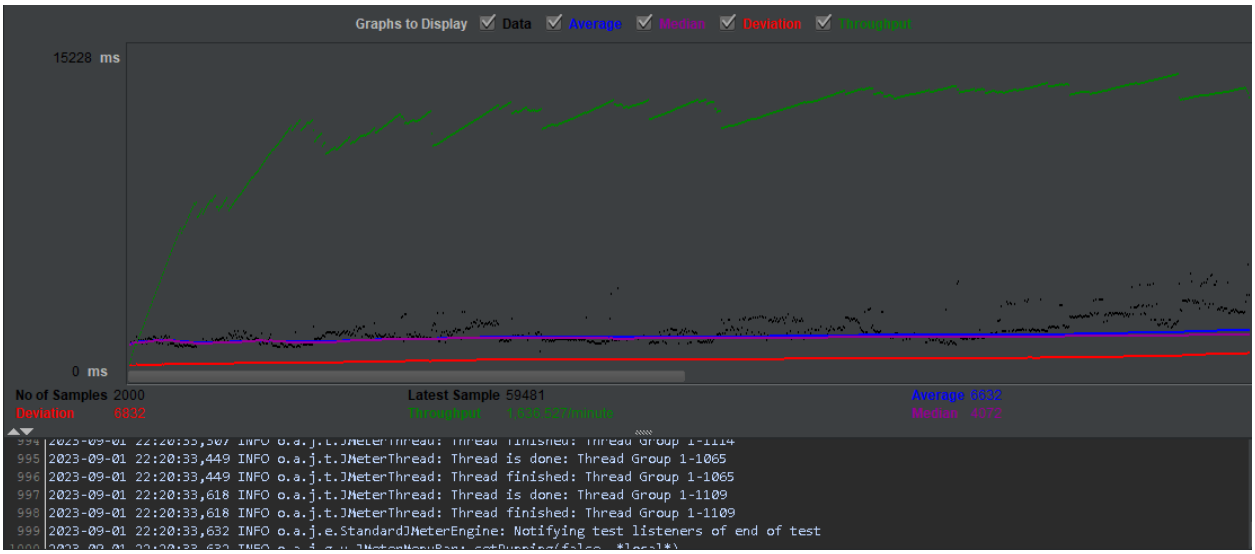
Naredne slike prikazuju slušatelja *Graph Results* za web lokacije *www.youtube.com/shorts*, *www.youtube.com* i *www.google.com* (Slika 4.8., Slika 4.9., Slika 4.10.).



Slika 4.8. Grafički prikaz rezultata testiranja za web lokaciju *www.youtube.com/shorts*.



Slika 4.9. Grafički prikaz rezultata testiranja za web lokaciju *www.youtube.com*.



Slika 4.10. Grafički prikaz rezultata testiranja za web lokaciju *www.google.com*.

Iz gore priloženih slika, vidljivo je kako je najveća propusnost kod *Google* poslužitelja, 1 636 527 zahtjeva u minuti, a devijacija ili odstupanje je najniže i iznosi 6 832. Zatim slijedi *YouTube* s propusnosti od 168 355 zahtjeva u minuti i devijacijom 95 010. Na kraju slijedi *YouTube/Shorts* s najnižom propusnosti od 107 212 zahtjeva u minuti i najvećom devijacijom 103 794. Prema ovim podacima, najbolju izvedbu ima *Google* poslužitelj.

4.2.3. Aggregate Graph

Aggregate Graph jedan je od važnijih slušatelja koji sadrži korisne metrike za cijeli test o svakom zahtjevu i kontroleru transakcije. Uključuje grafikon koji se može prilagoditi potrebama s mnogo različitih postavki. Rezultati prikazani u grafikonu mogu se izvesti kao PNG, a tablica s rezultatima može se izvesti kao CSV. Metrike koje se koriste jesu (Tablica 4.11.):

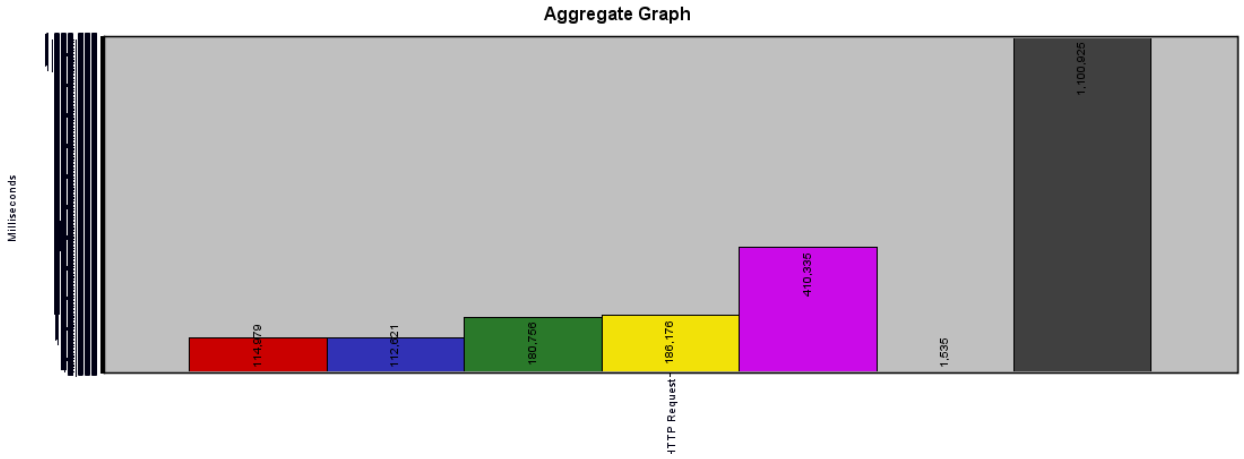
Tablica 4.11. Metrike korištene unutar slušatelja *Aggregate Graph*. [14, 15]

Label	Naziv zahtjeva
#Samples	Ukupan broj uzoraka
Average	Prosječno proteklo vrijeme u milisekundama
Median	Srednja vrijednost
90% Line	10% uzorkivača prekoračilo vrijeme za dolazak do poslužitelja
95% Line	5% uzorkivača prekoračilo vrijeme za dolazak do poslužitelja
99% Line	1% uzorkivača prekoračilo vrijeme za dolazak do poslužitelja
Min	Minimalno vrijeme potrebno uzorkivaču da stigne do poslužitelja
Max	Maksimalno vrijeme potrebno zahtjevu da stigne do poslužitelja
Error %	Broj uzorkivača pogreške/ukupni broj uzorkivača
Throughput	Propusnost – broj uzoraka po sekundi koje prima poslužitelj
Received KB/second	Broj kilobajta u sekundi koje prima klijent
Sent KB/second	Broj kilobajta u sekundi koji se šalju poslužitelju

Gore navedene metrike koje se koriste u slušatelju vrlo su važne i potrebne jer takva analiza podataka pomaže u poboljšanju performansi i izvedbi web aplikacije. Sljedeće slike prikazuju slušatelj *Aggregate Graph* za web lokacije *www.youtube.com/shorts*, *www.youtube.com* i *www.google.com* (Slika 4.12., Slika 4.13., Slika 4.14.).

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/...	Sent KB/sec
HTTP Request	2000	114979	112621	180756	186176	410335	1535	1100925	10.50%	1.8/sec	680.86	0.41
TOTAL	2000	114979	112621	180756	186176	410335	1535	1100925	10.50%	1.8/sec	680.86	0.41

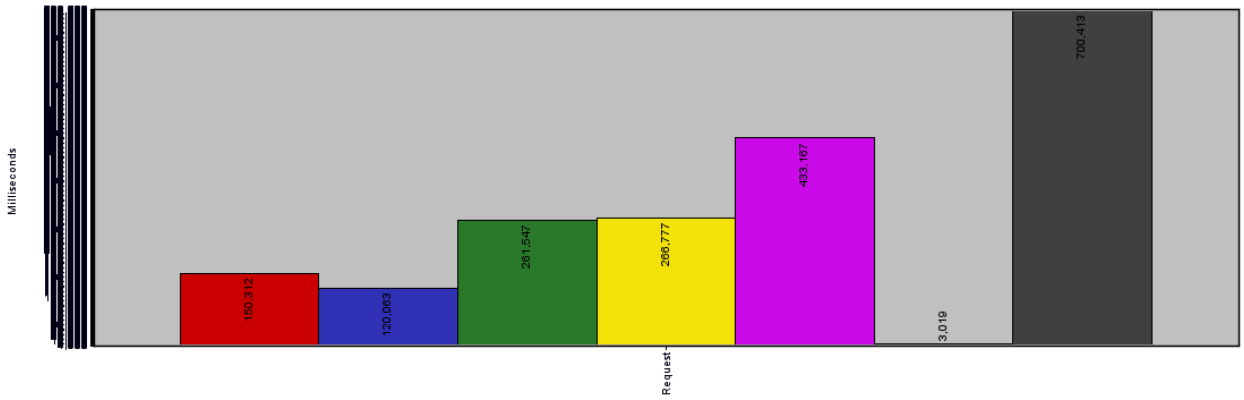
Settings Graph



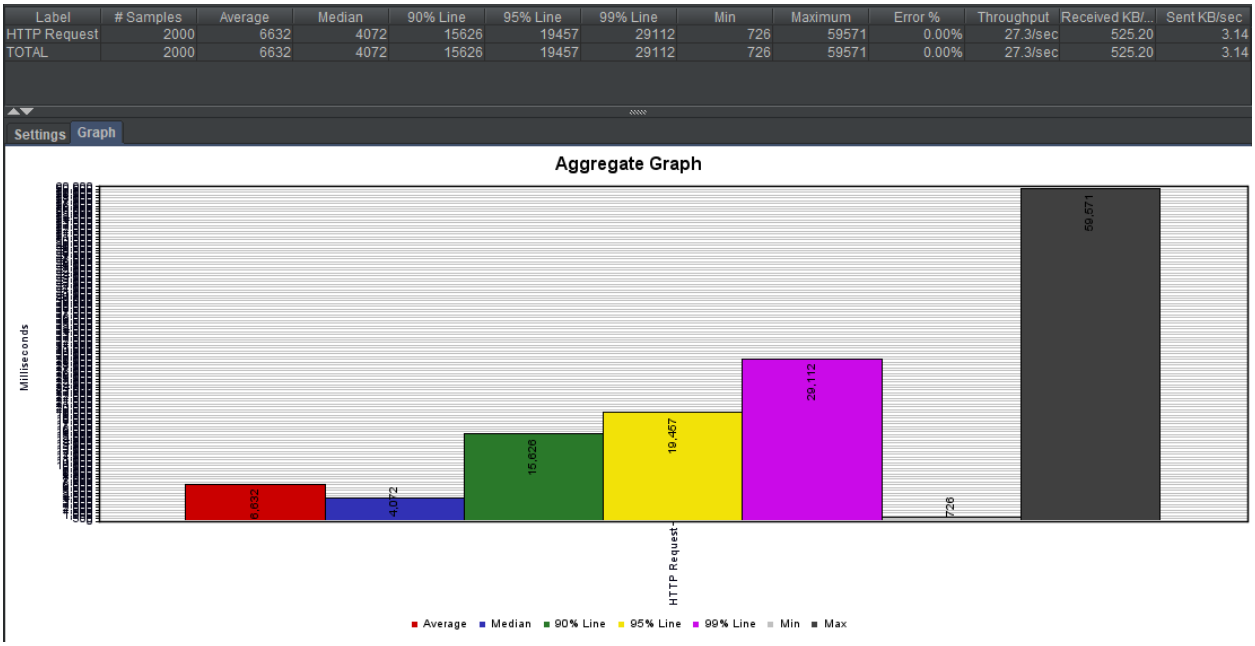
Slika 4.12. Grafički i tablični prikaz rezultata testiranja za web lokaciju *www.youtube.com/shorts*.

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/...	Sent KB/sec
HTTP Request	2000	150312	120063	261547	266777	433167	3019	700413	31.55%	2.8/sec	1469.79	0.69
TOTAL	2000	150312	120063	261547	266777	433167	3019	700413	31.55%	2.8/sec	1469.79	0.69

Settings Graph



Slika 4.13. Grafički i tablični prikaz rezultata testiranja za web lokaciju *www.youtube.com*.



Slika 4.14. Grafički i tablični prikaz rezultata testiranja za web lokaciju *www.google.com*.

Iz gore priloženih slika vidljivo je kako prema svim parametrima najbolju izvedbu ima *Google* poslužitelj, zatim *YouTube* i na kraju najlošiju izvedbu ima *YouTube/Shorts*.

5. APPIUM

Appium je platforma koja je otvorenog koda za automatizaciju testova namijenjena mobilnim aplikacijama. Pomoću *Appiuma* nastali su i mnogi drugi alati za automatizaciju zbog njegovog korisničkog sučelja i lakog povezivanja. Omogućava izvršavanje jednog testa na više platformi, jer je Appium platforma za više preglednika. Podržava razne programske jezike kao što su *PHP*, *Java*, *Perl*, *Python* i drugi. Jednostavan je za korištenje i koristi se za automatizaciju izvornih, hibridnih i mobilnih web aplikacija za *Android* i *iOS* (Tablica 5.1.).

Tablica 5.1. Vrste aplikacija. [16]

Izvorne aplikacije	• Aplikacije napisane uz pomoć *iOS-a, Androida* ili *Windowsa SDK* • Pristup nakon instalacije na uređaju • Primjeri su takve aplikacije *Skype, Viber, WhatsApp* • Mogućnost korištenja tek nakon instalacije, a ne može se otvoriti putem preglednika
Mobilne web aplikacije	• Pristup mobilnim web aplikacijama samo putem mobilnog preglednika • Primjer su *softwaretestinghelp.com, testgrid.io, swtestacademy.com ...* • Nema aplikaciju dostupnu za web mjesto
Hibridne aplikacije	• Ovakve aplikacije imaju omotač oko *webviewa* • *Webview* izvorna je kontrola koja omogućuje interakciju s web sadržajem • Pristup putem aplikacije instalirane na uređaju ili putem preglednika • Primjer su *Amazon, AliExpress, Facebook...*

5.1. Arhitektura Appiuma

Appium je HTTP poslužitelj napisan u *Node.js-u* koji rukuje *webdriver* sesijama. Poslužitelj prima HTTP zahtjeve od klijenta, a zatim se zahtjevi obrađuju na različite načine ovisno o vrsti platforme na kojoj se izvodi. *Appium* je *REST API* i prednost je ovog alata ta što je kod koji se koristi za interakciju napisan na mnogim jezicima. *Appium* se temelji na arhitekturi klijent – poslužitelj koja sadrži tri komponente, a to su *Appium* klijent, *Appium* poslužitelj i krajnji uređaj.[16]

5.1.1. *Appium* klijent

Appium klijent automatizirani je skriptirani kod. Kod može biti skriptiran u bilo kojem programskom jeziku, kao što je npr. *Java*, *PHP*, *Python*, *Perl*, itd. Skriptirani kod sadrži detalje konfiguracije mobilnog uređaja i aplikacije. Unutar anotacije *@BeforeClass* nalaze se specifikacije krajnjeg uređaja. Specifikacije se sastoje od naziva (npr. *Name)* i teksta (npr. *Nexus 5X*). Ostale specifikacije vidljive su na slici 5.2..

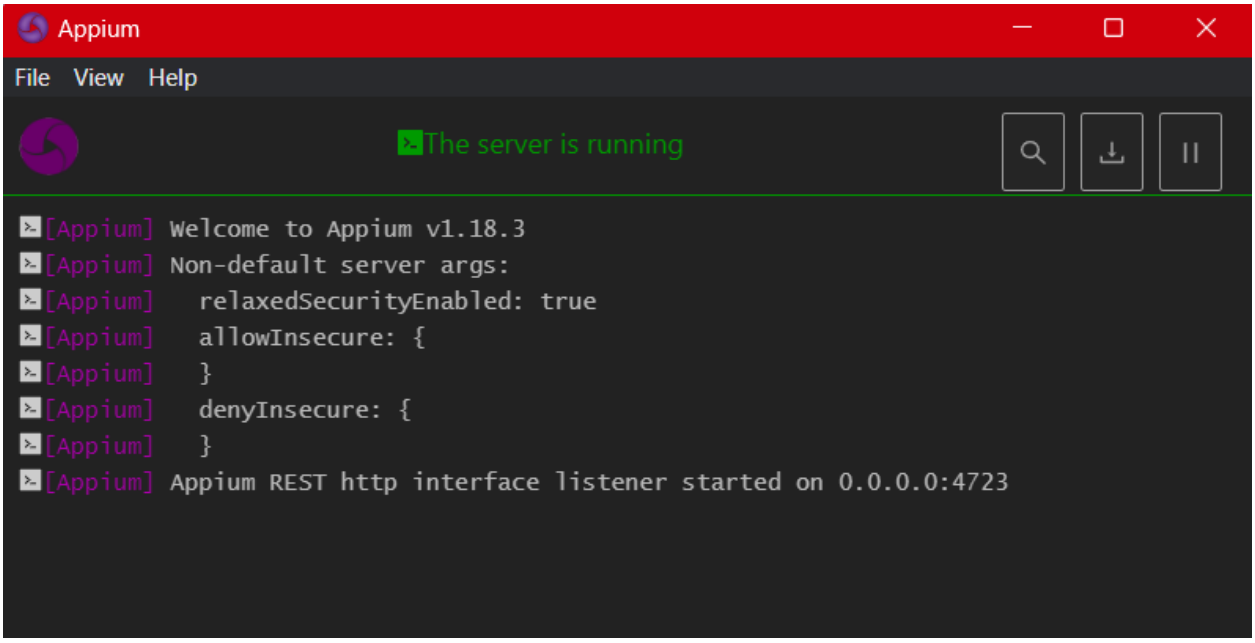
AndroidTestClass.java

```
1  import java.net.MalformedURLException;
2  import java.net.URL;
3  import java.util.concurrent.TimeUnit;
4  import io.appium.java_client.MobileElement;
5  import io.appium.java_client.android.AndroidDriver;
6  import org.openqa.selenium.By;
7  import org.openqa.selenium.remote.DesiredCapabilities;
8  import org.testng.annotations.BeforeClass;
9  import org.testng.annotations.Test;
10 public class AndroidTestClass{
11
12     AndroidDriver driver;
13     @BeforeClass ()
14     public void setUp() throws MalformedURLException{
15         //Set up desired capabilities and pass the Android app-activity and app-package to Appium
16         DesiredCapabilities capabilities = new DesiredCapabilities();
17         capabilities.setCapability( capabilityName: "App" , value: "C:\\Users\\Danijel\\ApkProjects\\ApiD\n" +
18                 "emos-debug\\ApiDemos-debug.apk" );
19         capabilities.setCapability( capabilityName: "Version" , value: "9.0" );
20         capabilities.setCapability( capabilityName: "Name" , value: "Nexus 5X" );
21         capabilities.setCapability( capabilityName: "platformName" , value: "Android" );
22         capabilities.setCapability( capabilityName: "control" , value: "emulator" );
23         driver = new AndroidDriver(new URL( spec: "http://127.0.0.1:4723/wd/hub" ), capabilities);
24         driver .manage().timeouts().implicitlyWait( l: 10 , TimeUnit. SECONDS );
25     }
26     @Test
```

Slika 5.2. *Appium* klijent – testna skripta.

5.1.2. *Appium* poslužitelj

Appium poslužitelj prima zahtjeve za povezivanje i naredbe poslane od strane klijenta u *JSON* formatu te se te naredbe izvršavaju na mobilnom uređaju. Poslužitelj stvara sesiju potrebnu za interakciju s krajnjim uređajima koja se pokreće prije pozivanja koda za automatizaciju (Slika 5.3.).



Slika 5.3. *Appium* poslužitelj.

5.1.3. Krajnji uređaj

Krajnji je uređaj mobilni uređaj ili emulator na kojem *Appium* poslužitelj izvršava skripte za automatizaciju pomoću naredbi klijenta. Klijent sadrži konfiguracije krajnjeg uređaja i skriptu s testnim slučajevima te šalje naredbe poslužitelju u *JSON* formatu. Skripta za automatizaciju pretvara se u *JSON* format pomoću ugrađenih *jar* datoteka u klijentu.

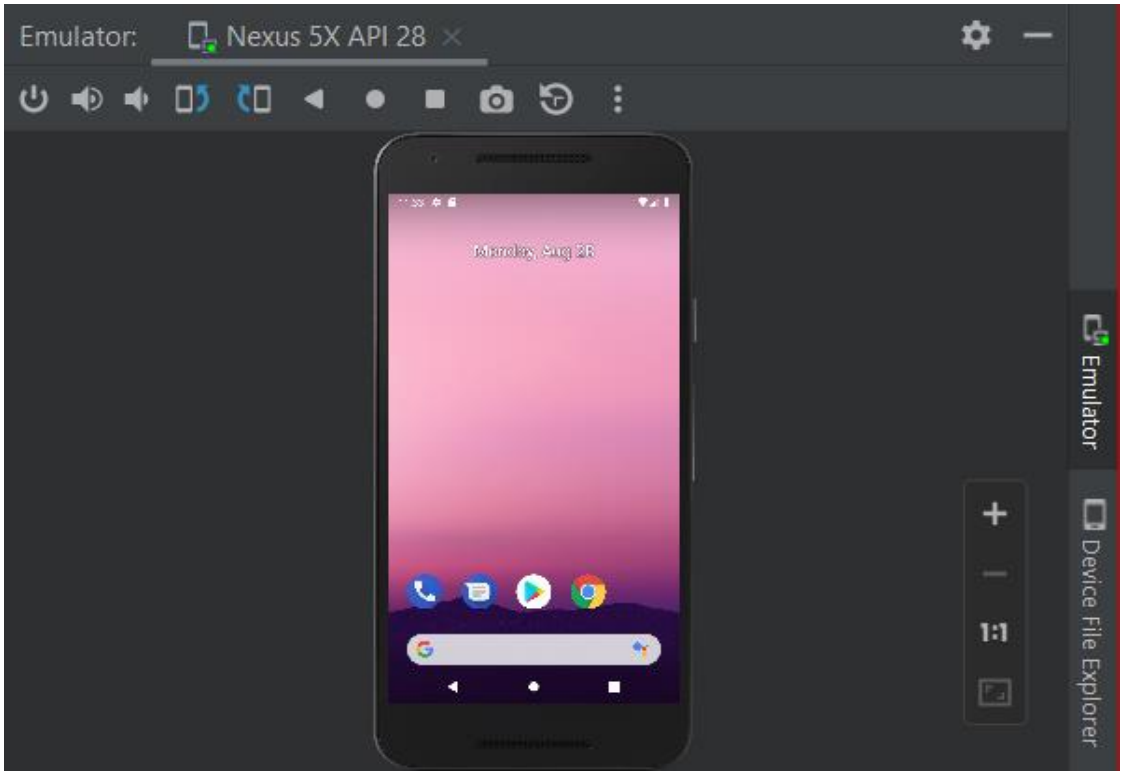
Poslužitelj prepoznaje naredbu i uspostavlja vezu s krajnjim uređajem. Nakon uspostavljanja veze, pokreće se izvođenje testnih slučajeva na krajnjem uređaju. Krajnji uređaj daje odgovor *Appiumu* u obliku HTTP-a. Sve informacije o izvršavanju testnih slučajeva zapisuju se u dnevnik svih radnji izvedenih na krajnjem uređaju (Slika 5.4.).

Način rada na *iOS-u*

Rad na *iOS* platformi omogućen je korištenjem *UIAutomation API*-ja za interakciju s elementima korisničkog sučelja aplikacije na *iOS* uređaju. *UIAutomation* je *JavaScript* biblioteka koja se koristi za automatizaciju. Kod izvršavanja testnih skripti, naredbe se preuzimaju u *JSON* formatu putem HTTP zahtjeva na poslužitelj. Zatim poslužitelj šalje naredbu instrumentima *UIAutomation* koji dalje traže datoteku *bootstrap.js.* Nakon toga, naredbe se izvršavaju u *bootstrap.js* datoteci unutar okruženja *iOS* instrumentima. Poslije izvršenja naredbe, klijent vraća poruku poslužitelju s detaljima dnevnika izvršene naredbe. [16]

Način rada na *Androidu*

Rad na *Androidu* omogućen je uz pomoć okvira *UIAutomator* za interakciju s elementima korisničkog sučelja aplikacije na *Android* uređaju. Taj su okvir razvili programeri za *Android* kako bi mogli testirati korisničko sučelje. Kod rada na *Androidu,* koristi se datoteka *bootstrap.jar* koja podržava sve verzije veće ili jednake *API*-ju 17, a za starije verzije koristi se okvir *Selendroid*. Kod izvršavanja testnih skripti, šalje se naredba u *JSON* formatu *UIAutomatoru/Selendroidu,* ovisno o *Android* verziji. Datoteka *bootstrap.jar* djeluje kao TCP poslužitelj i koristi se za slanje naredbe kako bi se izvršila akcija na *Android* uređaju pomoću *UIAutomatora/Selendroida*. [16]



Slika 5.4. Krajnji uređaj – emulator.

5.2. Praktični dio – automatsko testiranje mobilne aplikacije

U ovom poglavlju je prikazano testiranje mobilne aplikacije na platformi *Android*. Alati potrebni za izvođenje testiranja jesu *Android Studio, Java Development Kit (JDK), IDE (IntelliJ), TestNG, Selenium – Java, Node.js, Appium i APK App*. Verzija *Android Studija* korištena u ovom testiranju jest 2021.2.1, verzija *JDK-a* jest 14.0.2 i verzija *Appiuma* 1.18.3.

Prvo je potrebno postaviti okruženje u kojem će se provoditi testiranje. Okruženje se postavlja tako što se otvori *Upravljačka ploča* (eng. *Control panel*), zatim se odabere *Sustav i sigurnost* (eng. *System and security*) te se na lijevoj strani odabere stavka *Napredne postavke sustava* (eng. *Advanced system settings*). Kada se otvori prozor *Svojstva sustava* (eng. *System properties*), odabire se kartica *Napredno* (eng. *Advanced*) i lijevom se tipkom miša klikne na gumb koji se nalazi u donjem desnom uglu s nazivom *Varijable okruženja* (eng. *Environment variables*). U novom prozoru koji se naziva *Varijable okruženja* potrebno je dodati novu korisničku varijablu (eng. *User variables*). Ime varijable jest *ANDROID_HOME*, a vrijednost je varijable *C:\Users\"ime korisnika"\AppData\Local\Android\Sdk*. Zatim je potrebno dodati sistemske varijable (eng. *System variables*) pod naziv *PATH*, a vrijednosti tih varijabli jesu *C:\Users\"ime_korisnika"\AppData\Local\Android\Sdk\tools* i *C:\Users\"ime_korisnika"\AppData\Local\Android\Sdk\emulator*.

Zatim je potrebno pokrenuti *Daemon* pozadinski proces pomoću *Naredbenog retka* ili skraćeno, *cmd-a*. Nakon što se *cmd* pokrene, potrebno je otići u datoteku *platform-tools* sljedećom naredbom:

cd C:\Users\"ime_korisnika"\AppData\Local\Android\Sdk\platform-tools.

Zatim se pokreće *Daemon* proces naredbom *adb devices* i ispisuje se sljedeća poruka prikazana na slici 5.5..

```
C:\Users\Danijel\AppData\Local\Android\Sdk\platform-tools>adb devices
* daemon not running; starting now at tcp:5037
* daemon started successfully
List of devices attached
```

Slika 5.5. Pokretanje *Daemon* pozadinskog procesa.

Nakon što je Daemon uspješno pokrenut, treba pokrenuti *Android Studio*. Potrebno je u *Android Studiju* kreirati novi projekt. Projekt se kreira tako što se u početnom prozoru s lijeve strane

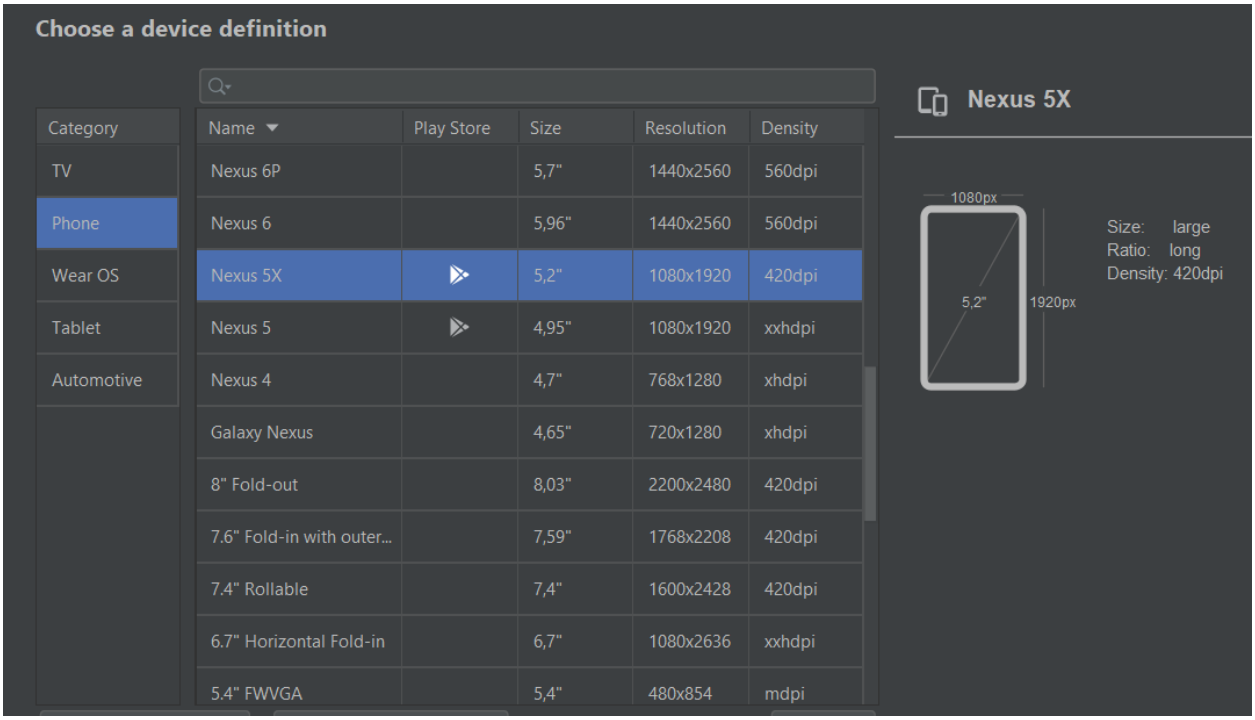
odabere stavka *Projects*, zatim se lijevim klikom miša klikne na gumb *New Project*. U novom prozoru s lijeve strane odabere se stavka *Phone and Tablet* pa stavka *No Activity* jer je potreban samo emulator na kojem će se izvršavati testiranje. Po završetku odabira aktivnosti, potrebno je kliknuti gumb *Next*.

U sljedećem prozoru potrebno je unijeti naziv projekta, odabrati lokaciju za pohranu projekta, odabrati programski jezik i postaviti minimalni SDK. Nakon što je sve uneseno i odabrano, klikne se na gumb *Finish*.

Nakon što se projekt kreira, u programu koji se otvori odabere se kartica *Tools* koja se nalazi u alatnoj traci, zatim se u padajućem izborniku odabere *Device Manager*.

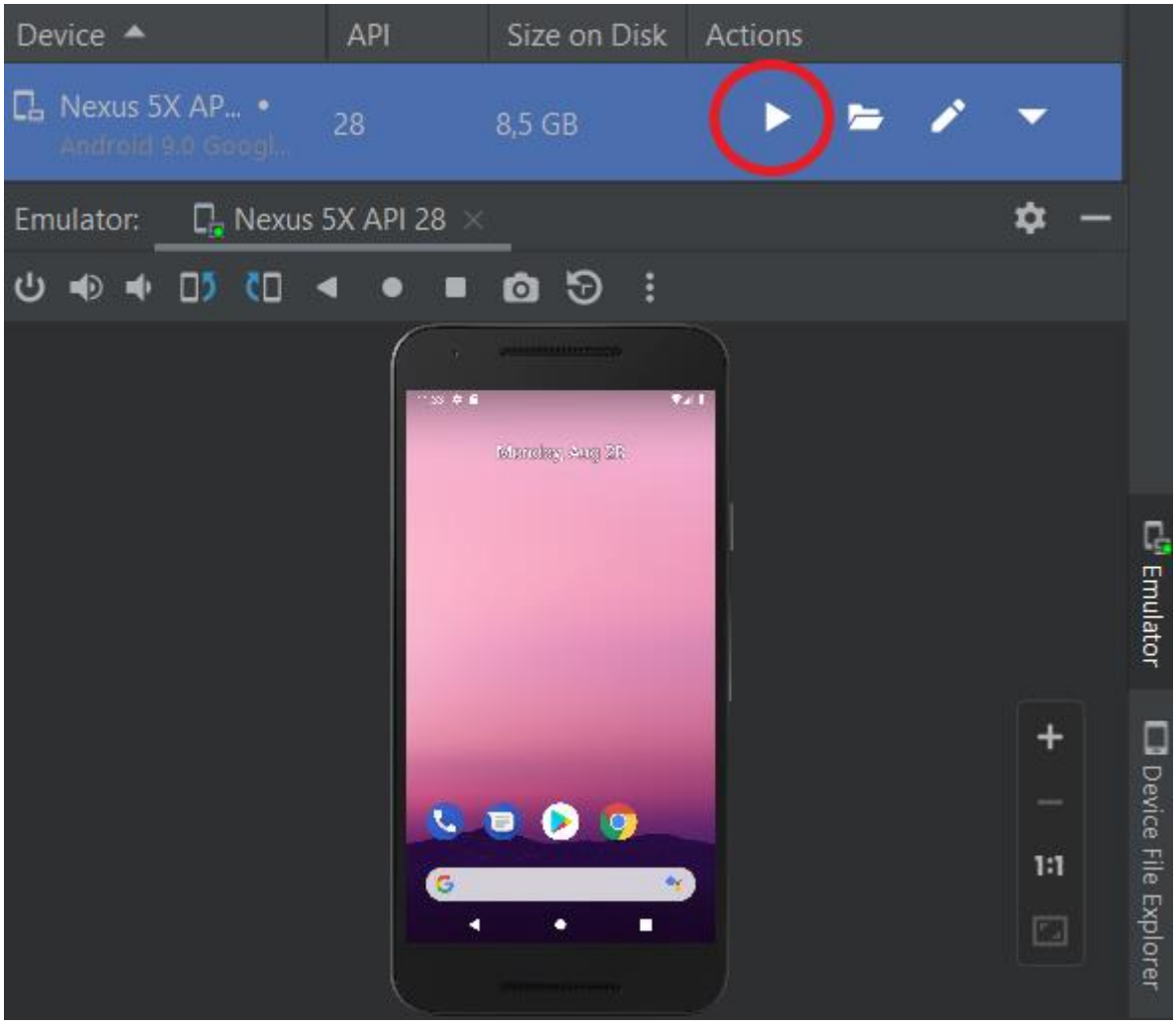
Testiranje se može provoditi na stvarnom fizičkom uređaju ili na virtualnom uređaju zvanom emulator. Ako se koristi stvarni uređaj, tada je potrebno odabrati karticu *Physical* i uređaj povezati s računalom pomoću USB kabla. Također, postoji mogućnost povezivanja putem *Wi-Fija* pomoću skeniranja QR koda ili koda za uparivanje. Kako bi se mogao koristiti stvarni uređaj, potrebno je u postavkama uređaja otići u *Opcije za razvojne programere* i označiti opciju *Uklanjanje pogrešaka*.

Ukoliko ne postoji niti jedan emulator, potrebno je kreirati novi. Klikom na gumb *Create device* otvara se prozor u kojem se odabire uređaj koji će se koristiti kao emulator. U ovom slučaju, to je mobilni uređaj naziva *Nexus 5X* koji podržava SDK API 28 i verzija je *Android 9.0 Pie* (Slika 5.6.). Poslije odabira uređaja, klikne se na gumb *Next* i u sljedećem prozoru pod karticom *Recommended* mora biti označena pripadajuća verzija uređaja. Nakon toga, klikne se na gumb *Next*. U zadnjem prozoru odabire se početni položaj mobitela (*Portrait* ili *Landscape*) i može se promijeniti naziv uređaja. Slika 5.6. prikazuje kreiranje emulatora, odnosno *Android Virtual Device* (*AVD*).



Slika 5.6. Kreiranje emulatora – odabir kategorije emulatora i odabir uređaja na kojem će se izvršiti testiranje.

Nakon završenog kreiranja emulatora, u *Device Manageru* na popisu uređaja pojavljuje se emulator koji je neposredno kreiran. Kako bi se emulator pokrenuo, potrebno je kliknuti na ikonu *Run* koja se nalazi desno u stupcu *Actions*. Nakon toga, pokreće se emulator i prikazuje se virtualni uređaj (Slika 5.7.).



Slika 5.7. Pokretanje emulatora u Android Studiu.

Emulator se može pokrenuti i putem *Naredbenog retka*. Prvo se odlazi u datoteku *emulator* sljedećim naredbama:

cd .. – (vraćanje unatrag)

cd emulator

ili naredbom:

cd C:\Users\"ime_korisnika"\AppData\Local\Android\Sdk\emulator.

Zatim se provjerava lista dostupnih uređaja naredbom *emulator -avd -list-avds* (Slika 5.8.).

```
C:\Users\Danijel\AppData\Local\Android\Sdk\emulator>emulator -avd -list-avds
Nexus_5X_API_28
```

Slika 5.8. Lista dostupnih uređaja.

Nakon toga, pokreće se emulator naredbom *emulator -avd „dostupni_uređaj“*, a u ovom je slučaju to *Nexus_5X_API_28* (Slika 5.9.).

```
C:\Users\Danijel\AppData\Local\Android\Sdk\emulator>emulator -avd Nexus_5X_A
PI_28
```

Slika 5.9. Pokretanje emulatora Nexus 5X.

Nakon što je emulator uspješno pokrenut, potrebno je vratiti se u datoteku *platform-tools*. To se postiže naredbom:

cd ..

cd platform-tools

ili naredbom:

cd C:\Users\“ime_korisnika“\AppData\Local\Android\Sdk\platform-tools.

Kako bi se vidio popis priključenih uređaja na listi u *Naredbenom retku*, potrebno je pokrenuti naredbu *adb devices* (Slika 5.10.).

```
C:\Users\Danijel\AppData\Local\Android\Sdk\platform-tools>adb devices
List of devices attached
emulator-5554   device
```

Slika 5.10. Popis priključenih uređaja.

Potom se pokreće testna aplikacija *APK App* na emulatoru naredbom:

adb -s emulator-5554 install „putanja_do_aplikacije“\ApiDemos-debug.apk.

U ovom slučaju, naredba je sljedeća:

adb -s emulator-5554 install C:\Users\“ime_korisnika“\ApkProjects\ApiDemos-debug\ApiDemos-debug.apk (Slika 5.11.).

```
C:\Users\Danijel\AppData\Local\Android\Sdk\platform-tools>adb -s emulator-55
54 install C:\Users\Danijel\ApkProjects\ApiDemos-debug\ApiDemos-debug.apk
Performing Streamed Install
Success
```

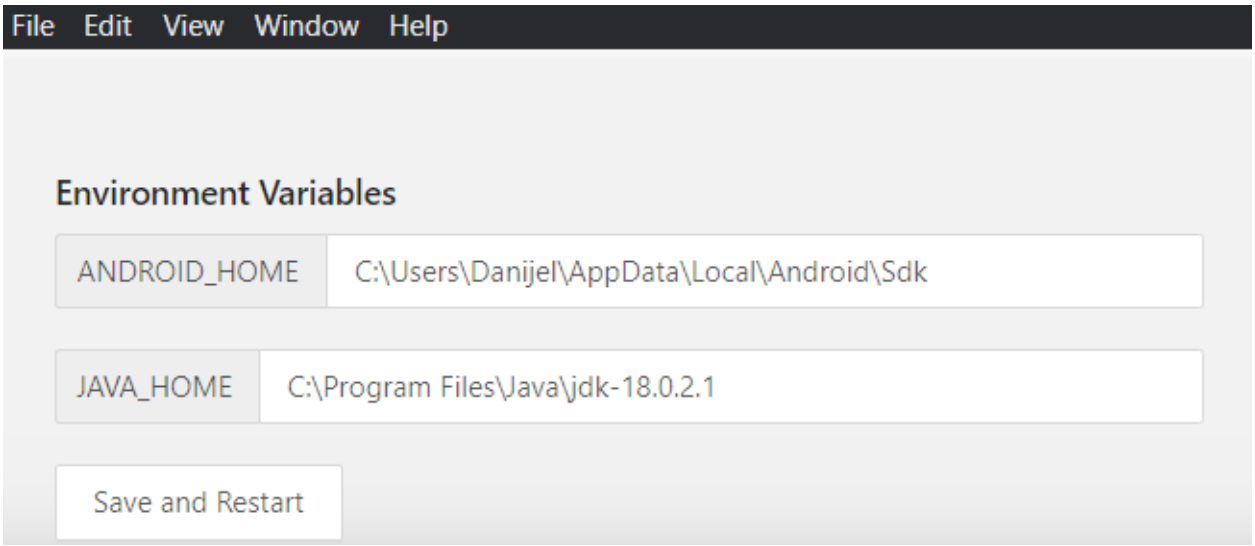
Slika 5.11. Pokretanje tetstne aplikacije.

Prije pokretanja *Appiuma*, potrebno je još pokrenuti i ADB poslužitelj koji će se upotrebljavati za slanje naredbi na krajnji uređaj. ADB poslužitelj pokreće se naredbom *adb start-server*. Ako je instalirana *Node.js* platforma, može se pokrenuti *Appium* poslužitelj (Slika 5.12.).

```
C:\Users\Danijel\AppData\Local\Android\Sdk\platform-tools>adb start-server
```

Slika 5.12. Pokretanje ADB poslužitelja.

Nakon što je *Appium* uspješno pokrenut, potrebno je u početnom prozoru otvoriti *Edit Configurations* i postaviti vrijednost varijabli za *ANDROID_HOME* i *JAVA_HOME*. Nakon što se unesu varijable, treba kliknuti na gumb *Save and Restart* (Slika 5.13.).

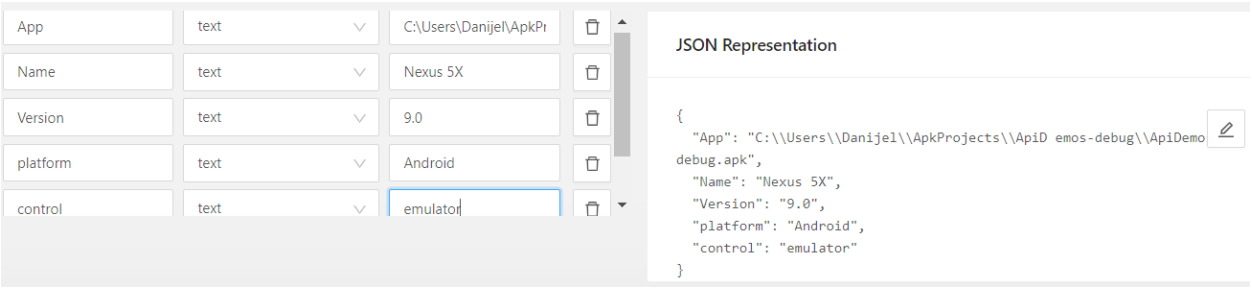


Slika 5.13. Namještanje konfiguracija.

Na početnom prozoru vidljiva je *host* i *port* opcija koje se automatski popunjavaju, ali se mogu promijeniti. Kada su konfiguracije postavljene i spremljene, poslužitelj se može pokrenuti klikom na *Start Server*.

Poslužitelj se pokreće na zadanoj *host* adresi i *portu* te se prikazuje izlazni dnevnik poslužitelja (Slika 5.3.).

Odabirom kartice *File* u alatnoj traci, otvara se padajući izbornik i odabire se *New Session Window*. Unutar tog novog prozora korisnik može istražiti svojstva elemenata mobilnog uređaja i unositi željene sposobnosti mobilnog uređaja poput imena, verzije, putanje testne aplikacije, vrste platforme i drugih specifikacija. Nakon dodavanja željenih sposobnosti, potrebno je kliknuti gumb *Start Session*. Primjer dodanih sposobnosti nalazi se na slici 5.14..



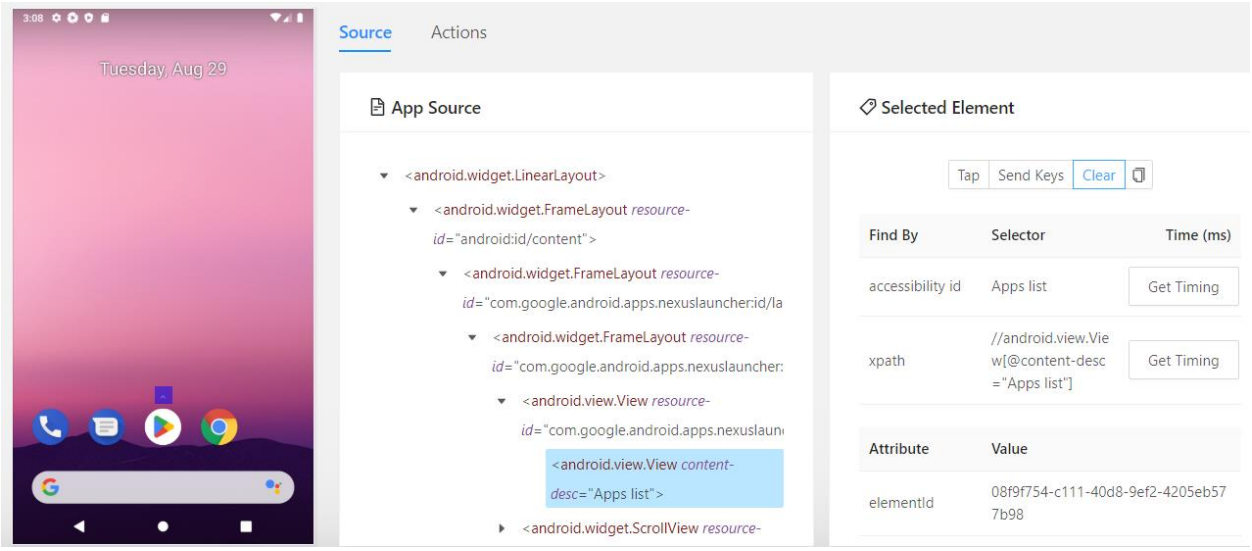
Slika 5.14. Dodavanje sposobnosti mobilnog uređaja koje su jednake sposobnostima u testnoj skripti.

Nakon što je sesija započeta, korisnik može pregledavati svosjtva elemenata na mobilnom uređaju ili emulatoru. Svaki od elemenata ima jedinstveno svojstvo. Pomoću tih svojstava može se upravljati elementima krajnjeg uređaja. Neka od najčešće korištenih svojstava jesu *name, xpath, class, resource-id* i *content-desc*. (Tablica 5.15.)

Tablica 5.15. Svojstva elemenata. [17]

Name	name	driver.findElement(By.name(„Name“));
Xpath	xpath	driver.findElement(By.xpath(„XPath“));
class	className	driver.findElement(By.className(„ClassName“));
resource-id	id	driver.findElement(By.id(„Id“));
content-desc	AccessibilityId	driver.findElementByAccessibilityId(„Content“);

Odabirom elementa i klikom na gumb *Tap*, može se kretati po aplikaciji, a klikom na gumb *Send Keys*, unosi se npr. tekst ili pojam u tražilicu odnosno izvršavaju se akcije na uređaju. Ta svojstva se koriste prilikom pisanja testne skripte kako bi se pomoću tih svojstava pronašli elementi aplikacije i olakšalo njihovo upravljanje (Slika 5.16.).



Slika 5.16. Pregled elemenata (npr. *android.view.View*) mobilne aplikacije i njihova svojstava (npr. *xpath*).

Nakon što je postavljeno razvojno okruženje, potrebno je napisati test koji će se izvršiti na mobilnom uređaju. Potrebno je u programu *IntelliJ* stvoriti novi *Maven* projekt kao što je objašnjeno ranije. Ponovno je korištena *Java Development Kit JDK 14.0.2*. Naziv je projekta *AndroidTest*.

Nakon što je kreiran projekt, potrebno je uvesti *Appium* i *TestNG* ovisnosti u *pom.xml* datoteku. Izgled *pom.xml* i dodane ovisnosti prikazane su na slici 5.17..

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.example</groupId>
    <artifactId>AndroidTest</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>io.appium</groupId>
            <artifactId>java-client</artifactId>
            <version>7.3.0</version>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>7.0.0</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
```

```
            <version>3.8.1</version>
            <type>maven-plugin</type>
        </dependency>
        <dependency>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-report-plugin</artifactId>
            <version>2.22.1</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.22.1</version>
                <configuration>
                    <suiteXmlFiles>
                        <suiteXmlFile>testng.xml</suiteXmlFile>
                    </suiteXmlFiles>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Slika 5.17. Primjer koda unutar pom.xml datoteke – dodavanje *Appium* i *TestNG* ovisnosti.

Nakon što su uvežene ovisnosti u novi projekt, potrebno je kreirati novu klasu. Kreiranje *Java* klase je opisano ranije u poglavlju *Selenium* [str. 15]. Naziv nove *Java* klase jest *AndroidTestClass*.

Kada je klasa dodana, potrebno je napisati testove. Na početku u anotaciji *@BeforeClass* potrebno je postaviti željene sposobnosti i proslijediti aktivnosti *Android* aplikacije i paket aplikacije u *Appium*. Sposobnosti (eng. *capabilities*) koje se postavljaju moraju biti iste one sposobnosti koje su se postavile u prozoru *New Session Window*. Zatim se postavlja vrijeme između svake akcije koju treba izvršiti. Nakon toga slijedi pisanje testova unutar anotacije *@Test.* Anotacija *@AfterClass* zatvara aplikaciju i završava sesiju.

U nastavku je prikazana testna skripta (Slika 5.18.).

```
import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.TimeUnit;
import io.appium.java_client.MobileElement;
```

```
import io.appium.java_client.android.AndroidDriver;
import org.openqa.selenium.By;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;
import org.testng.annotations.AfterClass;

public class AndroidTestClass{
    AndroidDriver driver;
    @BeforeClass ()
    public void setUp() throws MalformedURLException{
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability( "App" ,
"C:\\Users\\Danijel\\ApkProjects\\ApiD\n" +
                "emos-debug\\ApiDemos-debug.apk" );
        capabilities.setCapability( "Version" , "9.0" );
        capabilities.setCapability( "Name" , "Nexus 5X" );
        capabilities.setCapability( "platformName" , "Android" );
        capabilities.setCapability( "control" , "emulator" );
        driver = new AndroidDriver(new URL( "http://127.0.0.1:4723/wd/hub" ),
capabilities);
        driver .manage().timeouts().implicitlyWait( 10 , TimeUnit. SECONDS );
    }
    @Test
    public void testCall() throws Exception {
        MobileElement phone= (MobileElement)
                driver.findElement(By.xpath(
"//android.widget.TextView[@content-desc=\"Phone\"]" ));
        phone.click();
        MobileElement recentsCalls= (MobileElement)
                driver.findElement(By.xpath(
"/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.wi
dget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/andro
id.widget.RelativeLayout/android.widget.LinearLayout/android.widget.LinearLay
out[2]" ));
        recentsCalls.click();
        MobileElement contacts=(MobileElement)
                driver.findElement(By.xpath(
"/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.wi
dget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/andro
id.widget.RelativeLayout/android.widget.LinearLayout/android.widget.LinearLay
out[3]" ));
        contacts.click();
        MobileElement voicemail= (MobileElement)
                driver.findElement(By.xpath(
"/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.wi
dget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/andro
id.widget.RelativeLayout/android.widget.LinearLayout/android.widget.LinearLay
out[4]" ));
        voicemail.click();
        MobileElement keyPadButton= (MobileElement)
                driver.findElement(By.xpath(
"//android.widget.ImageButton[@content-desc=\"key pad\"]" ));
        keyPadButton.click();
        MobileElement number1= (MobileElement)
                driver.findElement(By.xpath(
"//android.widget.FrameLayout[@content-desc=\"1,\"]" ));
        number1.click();
        MobileElement number2= (MobileElement)
                driver.findElement(By.xpath(
"//android.widget.FrameLayout[@content-desc=\"2,ABC\"]" ));
```

```
        number2.click();

        driver.navigate().back();
        driver.navigate().back();
        driver.navigate().back();
    }
    @Test
    public void testMessages() throws Exception{
        MobileElement messages= (MobileElement)
                driver.findElement(By.xpath(
"//android.widget.TextView[@content-desc=\"Messages\"]" ));
        messages.click();
        MobileElement newConversationButton= (MobileElement)
                driver.findElement(By.id(
"com.google.android.apps.messaging:id/start_new_conversation_button" ));
        newConversationButton.click();
        MobileElement backButton= (MobileElement)
                driver.findElement(By.xpath(
"//android.widget.ImageButton[@content-desc=\"Navigate up\"]" ));
        backButton.click();
        MobileElement moreOptionsButton= (MobileElement)
                driver.findElement(By.xpath(
"//android.widget.ImageView[@content-desc=\"More options\"]" ));
        moreOptionsButton.click();
        MobileElement blockedContacts= (MobileElement)
                driver.findElement(By.xpath(
"/hierarchy/android.widget.FrameLayout/android.widget.FrameLayout/android.wid
get.ListView/android.widget.LinearLayout[2]/android.widget.LinearLayout" ));
        blockedContacts.click();

        driver.navigate().back();
        driver.navigate().back();
    }

    @Test
    public void testChrome() throws Exception{
        MobileElement chromeWeb= (MobileElement)
                driver.findElement(By.xpath(
"//android.widget.TextView[@content-desc=\"Chrome\"]" ));
        chromeWeb.click();
        /*MobileElement statisticsLabel= (MobileElement)
                driver.findElement(By.id(
"com.android.chrome:id/send_report_checkbox" ));
        statisticsLabel.click();
        MobileElement acceptContinue= (MobileElement)
                driver.findElement(By.id(
"com.android.chrome:id/terms_accept" ));
        acceptContinue.click();
        MobileElement noThanks= (MobileElement)
                driver.findElement(By.id(
"com.android.chrome:id/negative_button" ));
        noThanks.click();*/
        MobileElement searchingBar= (MobileElement)
                driver.findElement(By.xpath(
"//android.support.v7.widget.RecyclerView[@content-desc=\"New
tab\"]/android.widget.LinearLayout[1]/android.widget.LinearLayout/android.wid
get.EditText" ));
        searchingBar.click();
        MobileElement searching= (MobileElement)
                driver.findElement(By.xpath(
"/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.wi
```

```
dget.FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androi
d.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.FrameLayout/and
roid.widget.FrameLayout/android.widget.FrameLayout/android.widget.EditText"
));
        searching.click();
        searching.sendKeys("ferit");
        MobileElement selectFerit= (MobileElement)
                driver.findElement(By.xpath(
"/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.wi
dget.FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androi
d.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.ListView/androi
d.view.ViewGroup[1]/android.view.ViewGroup/android.widget.TextView" ));
        selectFerit.click();
        /*MobileElement acceptButton= (MobileElement)
                driver.findElement(By.xpath(
"/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.wi
dget.FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androi
d.view.ViewGroup/android.widget.FrameLayout[1]/android.widget.FrameLayout[2]/
android.webkit.WebView/android.view.View[11]/android.widget.Button" ));
        acceptButton.click();*/
        MobileElement linkFerit= (MobileElement)
                driver.findElement(By.xpath(
"/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.wi
dget.FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androi
d.view.ViewGroup/android.widget.FrameLayout[1]/android.widget.FrameLayout[2]/
android.webkit.WebView/android.view.View/android.view.View[2]/android.view.Vi
ew[2]" ));
        linkFerit.click();
        MobileElement navBar= (MobileElement)
                driver.findElement(By.xpath(
"/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.wi
dget.FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/androi
d.view.ViewGroup/android.widget.FrameLayout[1]/android.widget.FrameLayout[2]/
android.webkit.WebView/android.view.View[2]/android.view.View[2]" ));
        navBar.click();
        driver.navigate().back();
        driver.navigate().back();
        driver.navigate().back();
        driver.navigate().back();
    }
    @AfterClass
    public void teardown(){
        driver .quit();
    }
}
```

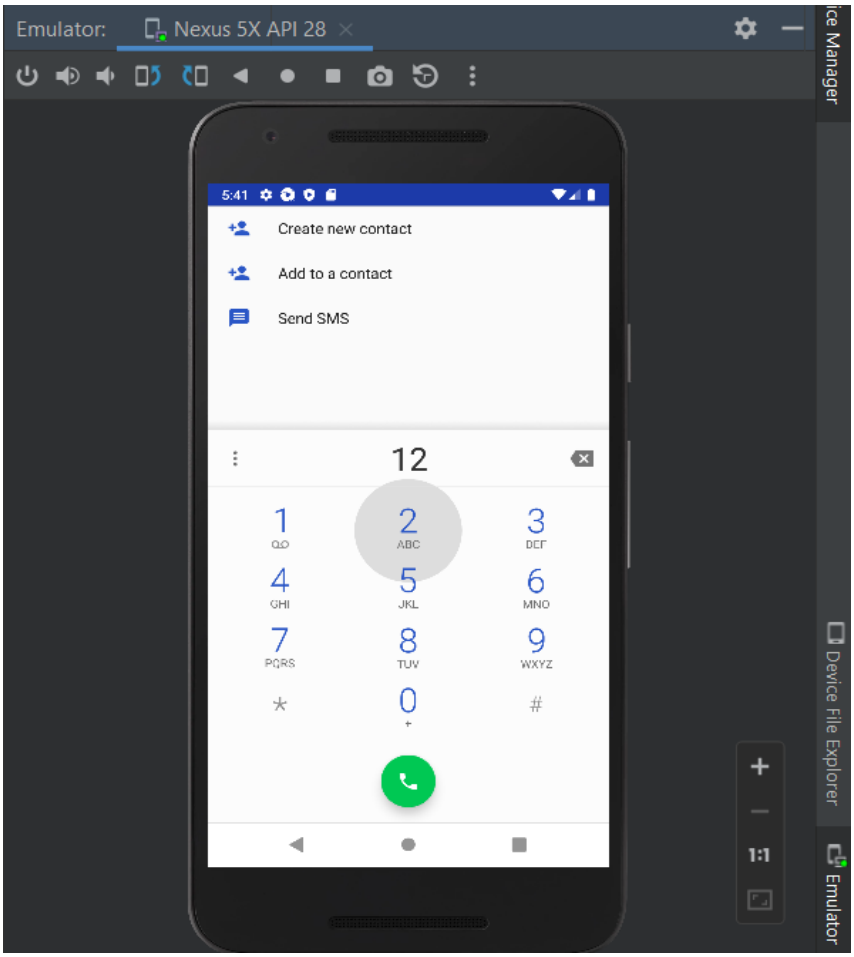
Slika 5.18. Testna skripta s tri testna slučaja.

Iz gore navedenog koda vidljivo da je osnovna metoda *findElement()*, dok je osnovna klasa objekta *MobileElement*. Metoda *findElement()* vraća objekt klase *MobileElement* koji može pohraniti svaki HTML element. Prvi test naziva *testCall* izvršava testiranje aplikacije *Telefon* (eng. *Phone*). Testiraju se gumbi *Nedavni pozivi*, *Kontakti* i *Glasovna pošta* te se pomoću tipkovnice unose brojevi. Drugi test se naziva *testMessages* i izvršava testiranje aplikacije *Poruke* (eng. *Messages*). Testiraju se gumbi *Novi razgovori, Više opcija* i gumb *Natrag* i testira

se kartica *Blokirani kontakti*. Treći test je naziva *testChrome* i testira se mobilna web aplikacija *Google Chrome*. Testiraju se tražilica, link *FERIT* i gumbi u navigacijskoj traci na web stranici *FERIT*. Dijelovi koda koji su unutar /**/ znakova komentara potrebni su samo pri prvom pristupu *Google Chromu*. Tada je potrebno odabrati opciju vezanu za upravljanje postavkama privatnosti, kolačićima i slanja povratnih informacija radi vođenja *Google* statistike.

Kada su testovi napisani, potrebno je prvo izgraditi projekt, a to se učini tako da se u alatnoj traci odabere kartica *Build* i u padajućem izborniku *Build Project*. Nakon toga, potrebno je desnom tipkom miša kliknuti na *Java* klasu *AndroidTestClass* i u padajućem izborniku odabrati *Create 'AndroidTestClass'* kako bi se kreirao test.

Nakon toga, potrebno je kliknuti na ikonu *Run* kako bi se testiranje pokrenulo. Pokretanje testiranja je moguće i odabirom kartice *Run*, a zatim u padajućem izborniku odabirom opcije *Run 'AndroidTestClass'*. Kada se testiranje pokrene, na emulatoru u *Android Studiju* može se vidjeti izvršavanje zadanih naredbi (Slika 5.19.).



Slika 5.19. Izvođenje testa na emulatoru.

Ako prilikom testiranja nije došlo do pogreške, u *Run* odjeljku s lijeve strane svi će testovi biti pozitvno označeni zelenim kvačicama. Također, bit će prikazano i njihovo vremensko trajanje u sekundama i milisekundama (Slika 5.20.). Testovi se mogu pokrenuti pojedinačno klikom na ikonicu *Run* koja se nalazi na početku svakog testa s lijeve strane.

Slika 5.20. Pozitivan ishod testiranja. Svi testni slučajevi su uspješno izvršeni.

6. ZAKLJUČAK

Kao što je prikazano u radu, automatizirano testiranje puno je brže i efikasnije od ručnog testiranja i uvelike pojednostavljuje postupak. Ručno je testiranje pogodno za testove koji se rijetko provode i nisu zahtjevni. Kod velikih softvera potrebni su alati za automatsko testiranje koji će pojednostaviti testiranje. Stoga, kao što je ranije objašnjeno, potrebno je odabrati ispravan alat za automatizaciju jer nije svaki alat pogodan za testiranje svake aplikacije. Izborom pogrešnog alata, dolazi do nepotrebnog gubitka resursa. U ovom su radu objašnjeni i prikazani alati otvorenog koda.

Selenium je alat za automatsko testiranje web aplikacija. U ovom radu je prikazano korištenje *Selenium WebDrivera*. Testiranje se izvršavalo na pregledniku *Google Chrome* uz pomoć *chromedrivera*. Prikazano je testiranje web aplikacije *AliExpress*, web stranice *FERIT* i web stranice *Portanova*. Testiranja su uspješno provedena i kao rezultat tih testiranja je tekst koji se ispisuje nakon svakog testa. Rezultat testiranja web aplikacije *AliExpress* je ispis teksta *Cellphones*. Zatim rezultat testiranja web stranice *FERIT* je ispis teksta *Prijava*. Rezultat zadnjeg testiranja web stranice *Portanova* je ispis teksta *Nagradna igra \ Back to School by Portanova!*.

Drugi alat čije korištenje je prikazano je *Apache JMeter*. Ovaj alat se koristi za automatsko testiranje opterećenja web lokacije. Testirane web lokacije su *YouTube/Shorts*, *YouTube* i *Google*. Pomoću slušatelja *View Results Tree*, *Graph Results* i *Aggregate Graph* analizirano je testiranje pojedine web lokacije. Prema svim parametrima najbolje performanse ima *Google*, a najlošije *YouTube/Shorts*.

Posljednji alat čije korištenje je prikazano u ovom radu je *Appium* koji se koristi za automatsko testiranje mobilnih aplikacija. Prikazana je arhitektura *Appiuma* koji se sastoji od klijenta, poslužitelja i krajnjeg uređaja. Testiranje se provelo uz pomoć emulatora koji je kreiran u *Android Studiju*, na kojem je izvršeno testiranje. Testna skripta sadrži tri testna slučaja gdje se testiraju mobilne aplikacije *Telefon* i *Poruke* i mobilna web aplikacija *Google Chrome*. Svi testovi su uspješno izvršeni.

Niti jedan alat nije stopostotno pouzdan, stoga je uvijek potreban ljudski faktor koji će odlučiti što je potrebno testirati, koji su prioriteti, kao i procjena korisnosti testova. Ponekad automatizirani testovi neće moći testirati neki dio, a tada nastupa ljudska kreativnost kod ručnog testiranja.

LITERATURA

1. Zaptest, Potpuni vodič za automatizaciju testiranja softvera, Zaptest, 2023., https://zaptest.com/hr/potpuni-vodic-za-automatizaciju-testiranja-softvera [21.08.2023.]
2. M.Grdić, Alati za automatizirano testiranje programskih proizvoda, Digital Repository Juraj Dobrila University of Pula, Pula, 2019., https://repozitorij.unipu.hr/islandora/object/unipu%3A4277/datastream/PDF/view, [20.08.2023.]
3. Myservername.com, Top 20 best automation testing tools 2021., myservername.com, 2021., https://hr.myservername.com/top-20-best-automation-testing-tools-2021 [23.08.2023.]
4. R.Wiltenburg, 20 najboljih alata za automatsko testiranje u 2020. godini, PC gadgets + blogs, 2020., https://pctown.co.nz/20-najboljih-alata-za-automatsko-testiranje-u-2020-godini/ [23.08.2023.]
5. K.Rungta, What is Selenium? Introduction to Selenium Automation Testing, Guru99, 2023., https://www.guru99.com/introduction-to-selenium.html [27.08.2023.]
6. BrowserStack, Selenium Testing: Detailed Giude, BrowserStack, 2023., https://www.browserstack.com/selenium [27.08.2023.]
7. O.Baskirt, Selenium Webdriver Tutorial with JAVA and TestNG, Software Test Academy, 2014., https://www.swtestacademy.com/selenium-webdriver-tutorial-java-testng/ [27.08.2023.]
8. G.Sekar, TestNG Annotations in Selenium Webdriver, Simplilearn.com, 2023., https://www-simplilearn-com.translate.goog/know-about-testng-annotations-in-selenium-webdriver-article [27.08.2023.]
9. Myservername.com, Top 25 Selenium Webdriver commands that you should know, myservername.com, 2020., https://hr.myservername.com/top-25-selenium-webdriver-commands-that-you-should-know [27.08.2023.]
10. Selenium, Locator strategies, Selenium, 2023., https://www.selenium.dev/documentation/webdriver/elements/locators/ [27.08.2023.]
11. Apache JMeter, Apache JMeter, The Apache Software Foundation, https://jmeter.apache.org/ [28.08.2023.]

12. Apache JMeter, Apache JMeter, The Apache Software Foundation, https://jmeter.apache.org/usermanual/component_reference.html [28.08.2023.]
13. K.Firsanov, JMeter Listeners – Part 1: Listeners with Basic Displays, BlazeMeter, 2020. https://www.blazemeter.com/blog/jmeter-listeners [29.08.2023.]
14. Software Testing Help, JMeter Listeners: Analyzing Results With Different Listeners, Software Testing Help, 2023., https://www.softwaretestinghelp.com/jmeter-listeners/ [29.09.2023.]
15. J.Loisel, JMeter Result Analysis: The Ultimate Guide, OctoPerf, 2017., https://octoperf.com/blog/2017/10/19/how-to-analyze-jmeter-results [31.08.2023.]
16. Software Testing Help, Introduction to Appium: What is Appium and its Architecture, Software Testing Help, 2023., https://www.softwaretestinghelp.com/what-is-appium/ [02.09.2023.]
17. Kobiton, Appium Element Locator Strategies, Kobiton, 2023., https://kobiton.com/blog/chapter-4-appium-locator-finding-strategies/ [02.09.2023.]

SAŽETAK

Cilj je ovog diplomskog rada bio predstaviti neke od popularnijih alata za automatizirano testiranje. Pomoću teksta, tablica i slika prikazano je korištenje tri različita alata za automatizirano testiranje. Prvi je alat *Selenium* gdje je korišten *Selenium WebDriver* za automatizirano testiranje web aplikacija. Drugi je alat *Apache JMeter* koji je korišten za testiranje opterećenja web usluga. Treći je alat *Appium* koji je korišten za automatizirano testiranje mobilnih aplikacija. Poslije svakog testiranja, prikazani su rezultati testiranja.

Ključne riječi: *Apache JMeter*, *Appium*, automatizirano testiranje, *Selenium*, testiranje

ABSTRACT

Automation testing tools

The aim of this thesis was to present some of the more popular tools for automated testing. Using text, tables and images, the use of three different tools for automated testing is demonstrated. The first tool is Selenium, where Selenium WebDriver was used for automated testing of web applications. Another tool is Apache JMeter which was used to test the load of the web service. The third tool is Appium, which was used for automated testing of mobile applications. After each test, the test results are presented.

Keywords: Apache Jmeter, Appium, automated testing, Selenium, testing