Prethodno priopćenje
https://doi.org/10.31784/zvr.7.1.15
Datum primitka rada: 15. 1. 2019.
Datum prihvaćanja rada: 26. 2. 2019.

AUTOMATSKO TESTIRANJE *WEB*-APLIKACIJA UZ PODRŠKU *WEB*-DRIVERA GEB

Mario Petković
Mag. inf., Coadria d. o. o., Prolaz Marije Krucifikse Kozulić 1, 51 000 Rijeka, Hrvatska;
e-mail: mpetkovic.ri@gmail.com

Sanja Čandrlić
Dr. sc., docent, Sveučilište u Rijeci, Odjel za informatiku, Radmile Matejčić 2, 51 000 Rijeka, Hrvatska;
e-mail: sanjac@inf.uniri.hr

Martina Ašenbrener Katić
Dr. sc., poslijedoktorand, Sveučilište u Rijeci, Odjel za informatiku, Radmile Matejčić 2, 51 000 Rijeka, Hrvatska; *e-mail*: masenbrener@inf.uniri.hr

SAŽETAK

Svakodnevnim korištenjem raznih softvera ljudi se susreću s pogreškama nastalim u procesu njihova razvoja. One mogu biti trivijalne, ali mogu biti i kritične za korištenje određenih funkcionalnosti softvera. Pogreške prilikom razvoja su neizbježne, zato se u testiranje softvera ulažu velike količine novca i vremena. No, unatoč velikom trudu i ulaganju, nemoguće je pronaći apsolutno sve pogreške prije izlaska softvera u produkciju. Pri tome nam može pomoći automatsko testiranje. U radu je prikazan proces automatskog testiranja web*-aplikacija primjenom alata za automatsko testiranje:* web*-drivera Geb, programskog jezika Groovy i testnog frameworka Spock. Rezultati ovog istraživanja pokazali su da kombinacija navedenih alata predstavlja odgovarajuće i kompletno rješenje za provedbu automatskog testiranja* web*-aplikacija.*

Ključne riječi: *automatsko testiranje, Geb, Groovy, Spock,* web*-aplikacija.*

1. UVOD

Testiranje softvera je proces izvršavanja programa s namjerom pronalaska skrivenih pogrešaka u softveru (Vliet, 2008). Pritom je, uz tehničko znanje koje je nužno za testiranje, potrebno voditi računa i o tome kako točno testirati da bi testiranje bilo isplativo. Loše testiranje često se temelji na pokušaju da se tijekom testiranja pokaže da softver nema pogrešaka, odnosno da radi ono što bi trebao. Ova kriva pretpostavka može voditi k odabiru onih testnih podataka koji će upravo to dokazati, međutim, cilj je testiranja pronaći pogrešku i tako podići kvalitetu i pouzdanost softvera. Treba, dakle, krenuti od pretpostavke da pogreške u softveru postoje i koristiti testne podatke većeg

rizika s ciljem njihova otkrivanja (Myers et al., 2004). Dodatnu objektivnost osigurat će testiranje koje provodi osoba koja nije pisala programski kôd koji se testira ili neovisan tim za osiguranje kvalitete softvera (Perry, 2006).

U idealnim uvjetima testirala bi se svaka moguća permutacija softvera, no to uglavnom nije moguće. Čak i naizgled jednostavni programi mogu imati tisuće kombinacija ulaza i izlaza. Stvaranje i izvršavanje testnih slučajeva za svaku mogućnost vrlo je nepraktično, trajalo bi predugo i zahtijevalo previše ljudskih resursa da bi bilo isplativo (Kaner, Bach, Pettichord, 2002). Stoga se umjesto iscrpnih testova koriste procjene rizika i prioriteta za fokusiranje na bitne testove (Hambling, 2015).

U provedbi bržeg i jeftinijeg testiranja značajnu ulogu može imati automatsko testiranje. Automatsko testiranje je korištenje softvera za provedbu ili pomoć pri provedbi testnih aktivnosti kao što su dizajniranje, izvođenje i provjera rezultata testova (Van Veenendaal, Graham, 2012). Njime se može umanjiti trud potreban za neophodno testiranje, kao i povećati broj izvršenih testova u određenom vremenskom periodu. Testove koji se inače ručno/manualno izvršavaju satima, moguće je izvršiti u svega nekoliko minuta. Automatske testove moguće je ponavljati s istim testnim podacima i kad god je potrebno, a to se ne može uvijek garantirati kod manualnog testiranja. Osim toga, automatizacija uklanja potrebu za manualnom provjerom repetitivnih testnih slučajeva (Fewster, Graham, 1999) koji su ljudima često zamorni.

No, potrebno je definirati odgovarajuće okruženje za provedbu automatskog testiranja. Predmet ovog istraživanja je razvoj i provjera okruženja za automatsko testiranje *web*-aplikacija primjenom sljedećih alata: *web*-drivera Geb, programskog jezika Groovy i testnog frameworka Spock. Izgrađeno je testno okruženje, ukratko opisana metodologija provedbe testova i rezultati njihove primjene na dvije testne web-aplikacije i kod automatskog testiranja u oblaku.

2. AUTOMATSKO TESTIRANJE

Od velikog broja mogućih testova zbog ograničenog vremena može se izvesti samo njihov mali broj. Zato testovi trebaju biti odabrani tako da omogućuju pronalazak većine pogrešaka u sustavu. To znači da je odabir testnih slučajeva koji će se izvršiti od velike važnosti.

Automatsko testiranje razlikuje se od manualnog testiranja. Puno je skuplje automatizirati test nego ga jednom manualno izvršiti. Kako bi automatsko testiranje bilo korisno, testovi se moraju pomno odabrati i implementirati. Jednom implementiran automatski test puno je ekonomičnije rješenje od manualnog jer je cijena njegovog pokretanja niska. No njihov razvoj stoji više i zato je važno voditi računa o pravilnoj implementaciji i održavanju. Automatsko testiranje brže je i pouzdanije od manualnog jer se izvršava pomoću alata i skripti, što smanjuje mogućnost ljudske pogreške i ne zahtijeva ljudske resurse. Ipak, za provedbu automatskog testiranja potrebno je ulagati u softverske alate koji ga omogućavaju. Njegova je mana što ne jamči unaprjeđenje korisničkog iskustva, jer su ljudi isključeni iz interakcije s programom koji se testira (www.apicasystems.com).

Manualno testiranje najbolje je upotrijebiti kod (www.apicasystems.com): istraživačkog testiranja (engl. *Exploratory Testing*) kod kojeg je bitno znanje, iskustvo i intuicija; testova lakoće korištenja

(engl. *Usability Testing*) kojima se procjenjuje jednostavnost korištenja (engl. *user-friendlyness*) iz perspektive krajnjeg korisnika te *ad hoc* testiranja kod kojeg nema plana, već se testiranje provodi oslanjajući se na osobni uvid testera.

Automatsko testiranje preferira se (Fernandes, Di Fonzo, 2017) kod: regresijskog testiranja (jednom pripremljeni regresijski testovi mogu se izvoditi svaki put kad je to potrebno); testiranja osnovnih funkcionalnosti (engl. *Smoke Testing*) koje omogućuje kvalitetnu brzu procjenu verzije softvera na temelju koje se zatim donosi odluka o detaljnijem testiranju; statičnih i repetitivnih testova (mogućnost češćeg izvođenja većeg broja testova, što pridonosi pouzdanosti sustava); testiranja temeljenog na podacima (engl. *Data Driven Testing*) kod kojeg je potrebna validacija velike količine različitih ulaza i velikih setova podataka te testiranja opterećenja i brzine izvođenja (engl. *Load & Performance Testing*) uz simulaciju velikog broja korisnika. Testovi se mogu ponovo koristiti, što znači bržu isporuku proizvoda kupcu, a isti se testovi mogu dosljedno koristiti na različitom hardveru, operacijskim sustavima ili bazama podataka.

Automatsko testiranje ima svoja ograničenja. Ono ne zamjenjuje manualno testiranje već ga nadopunjuje. Iako ne pronalazi istu količinu pogrešaka kao manualno, može značajno povećati kvalitetu i produktivnost testiranja softvera (Fewster, Graham, 1999). Preduvjet automatskom testiranju su automatski testovi napisani kvalitetno i korektno implementirani u faze životnog ciklusa testiranja softvera (www.istqbexamcertification.com).

Postoje tri glavne vrste automatskih testova (Cohn, 2009): oni koji se provode na razini kôda (jedinice, engl. *Unit*) i pokreću zajedno s kompajliranjem produkcijskog kôda, API (engl. *Application Programming Interface*) koji provjeravaju poslovnu logiku i arhitekturu softvera i testiraju funkcionalnost, usklađenost podataka i sigurnosne značajke te testovi korisničkog sučelja (UI testovi) koji oponašaju korištenje softvera kako bi to radio krajnji korisnik, što uključuje i simulaciju korištenja ulaznih jedinica. Ovaj rad bavi se upravo potonjima, s naglaskom na funkcionalno testiranje preko korisničkog sučelja.

Automatsko testiranje *web*-aplikacija ili automatsko funkcionalno testiranje *web*-aplikacija putem grafičkog sučelja je upotreba testnih skripti za izvršavanje testova s ciljem provjere ispravnosti *web*-aplikacije (Zhan, 2014). Tijekom izvršavanja automatskih testova na *web*-aplikaciji ili stranici moguće je vidjeti akcije klikanja miša i unosa teksta s tipkovnice bez ljudske interakcije. Time se provjeravaju razni aspekti ispravnosti *web*-aplikacije ili stranice (Rasmusson, 2016). Testira se jesu li sve poveznice na *web*-stranici ispravne i vode li na pravo mjesto. Provjerava se ispravnost obrazaca prilikom korisničkog unosa podataka. Testiraju se poslovni scenariji kroz koje korisnik prolazi prilikom korištenja aplikacije, a mogu se testirati i kolačići (engl. *cookies*).

3. METODOLOGIJA ISTRAŽIVANJA

Istraživanje (Dustin, Garrett, Gauf, 2010) povezano s testiranjem softvera koje je provela organizacija Innovative Defense Technologies pokazuje da većina sudionika istraživanja smatra da je automatsko testiranje korisno, ali malo njih ga primjenjuje. Razlozi zbog kojih se automatski testovi ne uvode su nedostatak vremena, novaca i stručnosti u području. Nedostatak stručnosti

podrazumijeva činjenicu da većina besplatnih alata i *frameworka* zahtijeva programersko znanje koje većina testera softvera ne posjeduje.

S obzirom na navedeno, za uspješno provođenje automatskog testiranja *web*-aplikacija (Zhan, 2014):

- testne skripte moraju biti jednostavne za čitanje i održavanje
- testni *framework* i alati moraju biti jednostavni za učenje, ali i novčano pristupačni
- izvršavanje testova mora biti što brže.

Zbog toga je za automatsko testiranje *web*-aplikacije u okviru ovog istraživanja izgrađeno okruženje koje se temelji na alatima Groovy, Spock i GebWeb.

3. 1 Groovy

Groovy (Subramaniam, 2013) je jezik koji je nastao 2003. godine, a njegova verzija iz 2007. godine počela se koristiti u IT-tvrtkama u SAD-u i Europi i od tada njegova popularnost sve više raste. Groovy dolazi do izražaja u kombinaciji s odgovarajućim *frameworkom* ili alatima. Groovy je objektno orijentirani programski jezik namijenjen za Java platforme. Radi se o dinamičkom jeziku sa značajkama sličnima programskim jezicima kao što su Python, Ruby, Perl i Smalltalk.

Usporedba (Ledbrook, 2015; Gee, 2015) Groovyja s okruženjem Jave u kombinaciji s JUnit *frameworkom*, koji se za testiranje puno češće koristi, ističe da se Groovy brže uči, da je u njemu lakše pisati kôd i razumjeti ga te da je potrebno manje linija kôda u odnosu na Javu. Ipak, Java se može koristiti i unutar Groovyja. Groovy ima nedostatke vezane uz lošije performanse, odnosno brzinu izvođenja, a kako nije tako često korišten, ima manje financijskih sponzora i nešto manju IDE podršku. Zbog toga je za razvoj produkcijskog kôda bolji izbor Java, dok je, zbog svoje jednostavnosti i ekspresivnosti, za pisanje testova bolji izbor Groovy. Dakle, Groovy i Java nisu suparnici, već ih se može koristiti zajedno za različite primjene, a ako se tijekom razvoja ukaže prednost jednog ili drugog jezika, moguće je napraviti konverziju (Ledbrook, 2015).

Kôd Groovy može se koristiti kompajliranjem pomoću Java Virtual Machine ili kao skriptni jezik. U prvom načinu, Groovy kôd kompajlira se pomoću Java Virtual Machine, što znači da se velika većina Java kôda može koristiti unutar Groovyja. Standardne Java knjižnice dostupne su u programima Groovy (König, 2015). Drugi način korištenja omogućuje da se kôd napisan unutar Groovy datoteke može izvršiti pokretanjem skripte u konzoli.

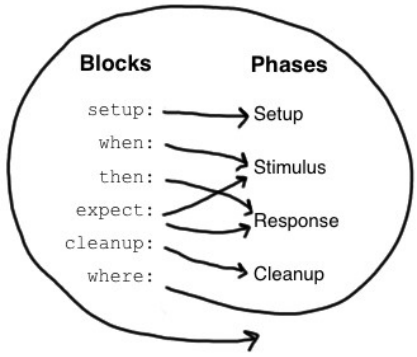
3. 2 Spock

Spock (Niederwieser, 2016) je testni i specifikacijski *framework* za Java i Groovy aplikacije. Zahvaljujući JUnit infrastrukturi koja pokreće testove, Spock je kompatibilan s većinom razvojnih alata. Pomoću Spocka moguće je pisati specifikacije odnosno testove koji opisuju očekivane značajke sustava u opsegu od jedne klase do cijele aplikacije.

Svojstvene metode (engl. *Feature Methods*) opisuju svojstva koja se očekuju unutar sustava koji se testira (Flecher, 2017). Sastoje se od četiri konceptualne faze: postavljanje, pružanje stimulansa sustavu, opis očekivanog ishoda, čišćenje, a prva i zadnja faza su opcionalne.

U Spocku se faze strukturiraju u blokove koji daju preglednu strukturu. Postoji šest vrsta blokova a to su: *setup (given), when, then, expect, cleanup* i *where* (slika 1). *Given, when* i *then* blokovi opisuju dani uvjet, stimulans ili akciju te očekivani ishod (Flecher, 2017).

Slika 1. Spock blokovi



Izvor: Niederwieser (2016)

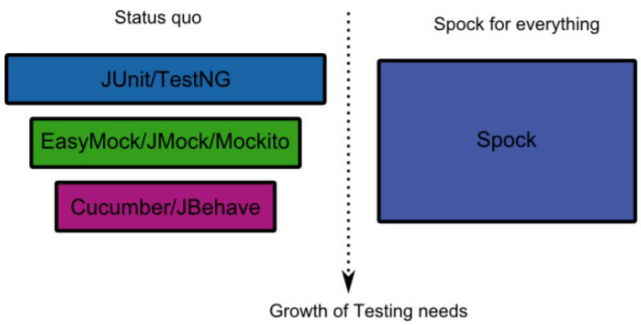
Za ovo istraživanje odabran je Spock, a ne JUnit jer:

- JUnit testovima nedostaje formalna semantika te je potrebno koristiti komentare u kôodu za bolju čitljivost. Korištenjem blokova Spock daje dobru strukturu i čitljivost testova, što je bitno kod velikih projekata (Kapelonis, 2016), a testne izvještaje razumiju i testeri, poslovni analitičari i menadžeri.
- U slučaju da test ne prođe, Spock pregledno prikazuje lokaciju pogreške, dok JUnit daje rezultat testa, ali ne i kontekst.
- Za razliku od Spocka, JUnit ne podržava *mocking* metode, već je za njihovo korištenje s JUnitom potreban odvojeni *framework*, na primjer Mockito. *Mocking* metode omogućuju oponašanje i izmjenu klasa s kojima testirani sustavi komuniciraju (www.baeldung.com).
- Spock se izvršava pomoću JUnita, pa je kompatibilan s postojećom Java infrastrukturom: moguće je dodati Spock testove u Java projekt, zadržati JUnit testove i pokretati ih zajedno sa Spock testovima, kao i koristiti alate kao što su Maven, IntelliJ, Sonar, Eclipse itd.
- Spock teži biti ultimativno rješenje za testiranje (slika 2), pokrivajući kompletni životni ciklus testiranja bez dodanih *frameworka* i biblioteka (Kapelonis, 2017).

3. 3 Geb

Geb je alat koji služi za automatizaciju interakcije između *web*-preglednika i *web*-sadržaja nastao s ciljem povećanja produktivnosti i smanjenja kompleksnosti automatizacije unutar *browsera*. Koristi iste biblioteke kao i Selenium *web*-driver, što znači da Geb može raditi u svim *browserima* koje podržava *web*-driver (Mishra, 2016). Automatizacija unutar *browsera* namijenjena je za *web*-testiranje, ali može služiti i za automatsko odrađivanje repetitivnih zadataka na *webu* (automatizirane skripte). Geb pruža podršku za funkcionalno *web*-testiranje zahvaljujući integraciji s popularnim testnim alatima kao što su Spock, JUnit i Cucumber-JVM, a njegovi autori preporučuju upravo korištenje Spocka. Za samu integraciju Spocka s Gebom dovoljno je uključiti Spockovu biblioteku i naslijediti baznu klasu za izvještaje (GebReportingSpec) (Daley, Erdmann, Pragt, 2015).

Slika 2. Spock kao nezavisni testni *framework*



Izvor: Kapelonis (2017)

3. 4 Predmet testiranja

Okruženje koje se temelji na navedenim alatima testirano je u tri dijela:

- automatsko testiranje postojeće *web*-stranice
- automatsko testiranje nove *web*-aplikacije
- automatsko testiranje u oblaku.

Rezultati provjere prikazani su u sljedećem poglavlju. Prikazana je metodologija provedbe provjere i pojedinačni koraci testiranja za navedene predmete testiranja. Prikazane su skripte i kôd korišten u testiranju.

4. REZULTATI ISTRAŽIVANJA

Prikazat će se rezultat provjere izgrađenog okruženja na postojećoj *web*-stranici, novoj *web*-aplikaciji te testiranje u oblaku.

4. 1 Testiranje postojeće *web*-stranice

Za primjer funkcionalnog automatskog testa *web*-stranice koristit će se *web*-stranica Odjela za informatiku Sveučilišta u Rijeci i dva testa:

1. Navigacija do stranice i provjera osnovnih podataka
    a. Postavljanje Google tražilice za početnu stranicu testa
    b. Provjera učitane stranice
    c. Upisivanje teksta: „odjel za informatiku rijeka“ u tražilicu
    d. Klik na prvi ponuđeni rezultat (*web*-stranica Odjela)
    e. Provjera učitane stranice
    f. Klik na „O odjelu“ link u glavnom izborniku *web*-stranice
    g. Provjera učitane stranice
    h. Provjera ispravnosti upisanih podataka (adresa, telefon, e-mail, OIB...)
2. Promjena jezika stranice
    a. Postavljanje *web*-stranice Odjela za informatiku za početnu stranicu
    b. Provjera učitane stranice
    c. Klik na ikonu za promjenu jezika stranice
    d. Provjera učitane stranice
    e. Provjera promjene URL-a
    f. Provjera prikazane poruke dobrodošlice (Welcome to the Department of Informatics official website!)

Slike 3 – 6 prikazuju kodove za prvi test, a slike 7 – 9 prikazuju kodove za drugi test.

Slika 3. GooglePage

```
package com.mycompany.component.library
import Geb.Page import static
com.mycompany.Config.CONFIG

class GooglePage extends Page {
static url = "http://${CONFIG.server}"
static at = { title == "Google" }
static atCheckWaiting = true

   static content = {
      searchBar(wait: true) { $('input', id: 'lst-ib') }
       prviRezultatPretrage(wait: true) { $('div', id: 'ires').$('a',
0)
}
   }
}
```

Izvor: izrada autora

Slika 4. OdjelZaInfOodjeluPage

```
package com.mycompany.component.library
import Geb.Page

class OdjelZaInfOodjeluPage extends Page {
   private static final ODJEL_ZA_INF_NASLOV = "Odjel za informatiku -
O odjelu"     private static final ADRESA = "Radmile Matejčić 2"
private static final TELEFON = "+ 385 51 584 700"    private static
final EMAIL = "ured@inf.uniri.hr"    private static final IBAN =
"HR0423600001400485134"    private static final OIB = "64218323816"

   static at = { title == ODJEL_ZA_INF_NASLOV }
static atCheckWaiting = true

   static content = {       adresaOdjela(wait:true) {
$('.item-page').$('div', text:
iContains(ADRESA)) }       telefonOdjela(wait:true) {
$('.item-page').$('div', text:
iContains(TELEFON)) }       emailOdjela(wait:true) {
$('.item-page').$('div', text:
iContains(EMAIL)) }       ibanOdjela(wait:true) { $('.item-
page').$('span', text:
iContains(IBAN)) }       oibOdjela(wait:true) { $('.item-
page').$('span', text:
iContains(OIB)) }
   }

  def provjeraOsnovnihPodataka() {
adresaOdjela.isDisplayed()
telefonOdjela.isDisplayed()
emailOdjela.isDisplayed()
ibanOdjela.isDisplayed()
oibOdjela.isDisplayed()
   }
}
```

Izvor: izrada autora

Slika 5. OdjelZaInfHomePage

```
package com.mycompany.component.library
import Geb.Page

class OdjelZaInfHomePage extends Page {    private static final
ODJEL_ZA_INF_NASLOV = "Odjel za informatiku - Naslovnica"
   private static final O_ODJELU_LINK = "O odjelu"

   static at = { title == ODJEL_ZA_INF_NASLOV }
static atCheckWaiting = true

   static content = {
      oOdjeluLink(wait: true) { $('a', text: iContains(O_ODJELU_LINK)) }
   }
}
```

Izvor: izrada autora

Slika 6. ProvjeraPodatakaOdjelZaInfTest

```
package com.mycompany.tests
import com.mycompany.component.library.GooglePage import
com.mycompany.component.library.OdjelZaInfHomePage import
com.mycompany.component.library.OdjelZaInfOodjeluPage
import Geb.Spock.GebReportingSpec import
org.openqa.Selenium.Keys import Spock.lang.Stepwise

@Stepwise
class ProvjeraPodatakaOdjelZaInfTest extends GebReportingSpec {
private static final String PRETRAZI_ODJEL_ZA_INF = "odjel za
informatiku rijeka"

   def "Provjera osnovnih podataka na web stranici Odjela za
informatiku"() {
      when: "Otvaranje Google.hr web stranice"
to GooglePage
      then: "Provjera otvorene stranice - Google.hr"
at GooglePage
      when: "Unos i pretraga trazenog pojma u trazilici"
searchBar.value(PRETRAZI_ODJEL_ZA_INF) << Keys.ENTER
      and: "Klik na prvi rezulat pretrage"
prviRezultatPretrage.click()
      then: "Provjera otvorene stranice - inf.uniri.hr/"
at OdjelZaInfHomePage

      when: "Klik na meni link 'O odjelu' "
oOdjeluLink.click()
      then: "Provjera osnovnih podataka"
at OdjelZaInfOodjeluPage

      //metoda provjerava ispravnost osnovnih podataka
provjeraOsnovnihPodataka()
   }
}
```

Izvor: izrada autora

Slika 7. OdjelZaInfHomePage

```
package com.mycompany.component.library
import Geb.Page
 import static
com.mycompany.Config.CONFIG

class OdjelZaInfHomePage extends Page {    private static final
ODJEL_ZA_INF_NASLOV = "Odjel za informatiku - Naslovnica"
   private static final O_ODJELU_LINK = "O odjelu"
    static url =
"http://${CONFIG.urlOdjelZaInf}"    static at =
{ title == ODJEL_ZA_INF_NASLOV }    static
atCheckWaiting = true

   static content = {
      oOdjeluLink(wait: true) { $('a', text: iContains(O_ODJELU_LINK)) }

      engleskiJezikZastava(wait: true) { $('.mod-languages').$('a',
href: '/en/') }
   }
}
```

Izvor: izrada autora

Slika 8. OdjelZaInfEngleskiHomePage

```
package com.mycompany.component.library
import Geb.Page
import static
com.mycompany.Config.CONFIG

class OdjelZaInfEngleskiHomePage extends Page {    private static
final ODJEL_ZA_INF_NASLOV = "Odjel za informatiku - Home page"
    static url =
"http://${CONFIG.urlOdjelZaInfEng}"    static at =
{ title == ODJEL_ZA_INF_NASLOV }    static
atCheckWaiting = true

   static content = {
porukaDobrodoslice1(wait:true) { $('span', id:
'result_box').$('span', class: 'hps', 0) }
porukaDobrodoslice2(wait:true) { $('span', id:
'result_box').$('span', class: 'hps', 1) }
}
}
```

Izvor: izrada autora

Slika 9. PromjenaJezikaTest

```
package com.mycompany.tests
import
com.mycompany.component.library.OdjelZaInfEngleskiHomePage
import com.mycompany.component.library.OdjelZaInfHomePage import
Geb.Spock.GebReportingSpec import Spock.lang.Stepwise

@Stepwise
class PromjenaJezikaTest extends GebReportingSpec {
   private static final String ENGLESKA_PORUKA_DOBRODOSLICE = "Welcome
to the Department of Informatics official website!"
    def "Promjena jezika na web stranici Odjela za informatiku"()
{     when: "Otvaranje naslovne stranice Odjela za informatiku"
to OdjelZaInfHomePage
      then: "Provjera otvorene stranice - inf.uniri.hr"
at OdjelZaInfHomePage

      when: "Klik na englesku zastavu za promjenu jezika"
engleskiJezikZastava.click()
      then: "Provjera otvorene stranice - inf.uniri.hr/en i
poruke dobrodošlice"       at OdjelZaInfEngleskiHomePage


      def porukaDobrodoslice = porukaDobrodoslice1.text() + " "
+ porukaDobrodoslice2.text()       porukaDobrodoslice ==
ENGLESKA_PORUKA_DOBRODOSLICE
   }
}
```

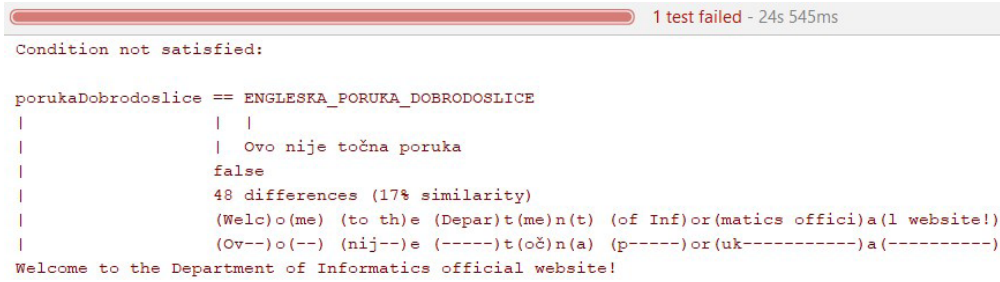
Izvor: izrada autora

Oba testa su prolazna (slika 10). Za dodatnu provjeru ispravnosti može se namjerno izazvati neuspjeli test, na primjer, izmjena očekivane poruke dobrodošlice na engleskom u oblik koji će izazvati pogrešku (slika 11).

Slika 10. Rezultat testova na stranicama Odjela za informatiku

Izvor: izrada autora

Slika 11. Neuspjeli test



Izvor: izrada autora

4. 2 Testiranje *web*-aplikacije u razvoju

Ovo poglavlje donosi automatske testove na *web*-aplikaciji u razvoju. Dane su osnovne informacije o strukturi aplikacije i tehnologijama korištenim za njen razvoj, opisana korištena testna metoda, koraci te sam kôd testa.

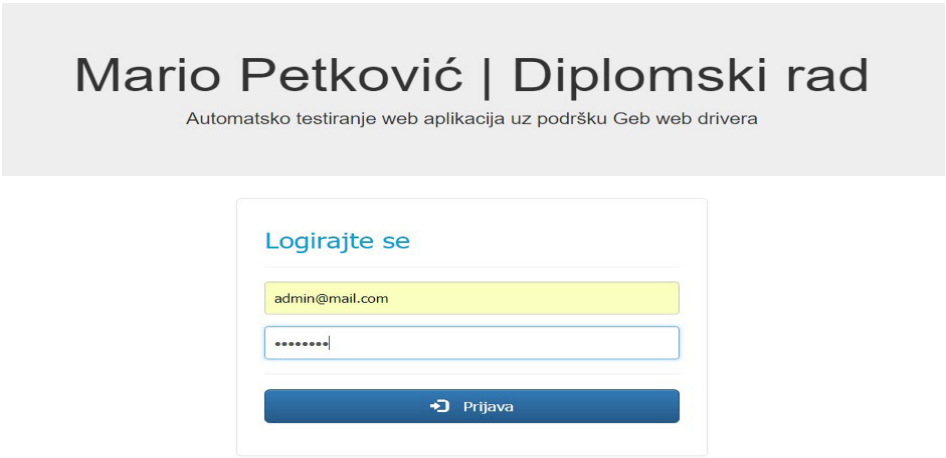
Za potrebe istraživanja kreirana je *web*-aplikacija koja ima funkcionalnosti prijave i validacije korisnika, navigacije, ispunjavanje i slanje formi. Za izradu aplikacije korišten je Javascript programski jezik za *front-end* i php za *back-end* te MySQL baza podataka. Osim toga, korištena je i Bootstrap

biblioteka za izgled korisničkog sučelja u kombinaciji s klasičnim HTML-om i CSS-om. Za asinkrone pozive sa serverom korištena je Ajax tehnika.

Web-aplikacija sastoji se od tri funkcionalne stranice:

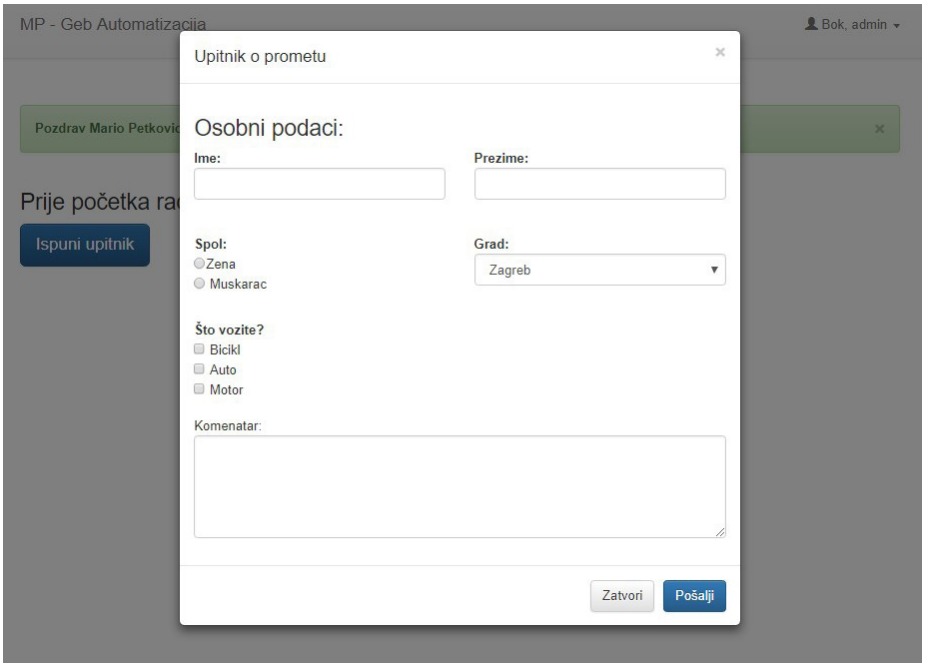
1. Stranica za prijavu korisnika (slika 12) – Jednostavna stranica za prijavu i validaciju postojećih korisnika. Ako korisnik unese krivi *e-mail* ili lozinku, prijava neće biti izvršena.
2. Početna stranica – Sastoji se od alatne trake s padajućim izbornikom, poruke dobrodošlice za prijavljenog korisnika i gumba za ispunjavanje upitnika (slika 13) koji otvara modalni prozor s formom.
3. Korisnički profil (slika 14) – Sadrži korisničke podatke: ime, prezime, lozinka itd.

Slika 12. Stranica za prijavu korisnika



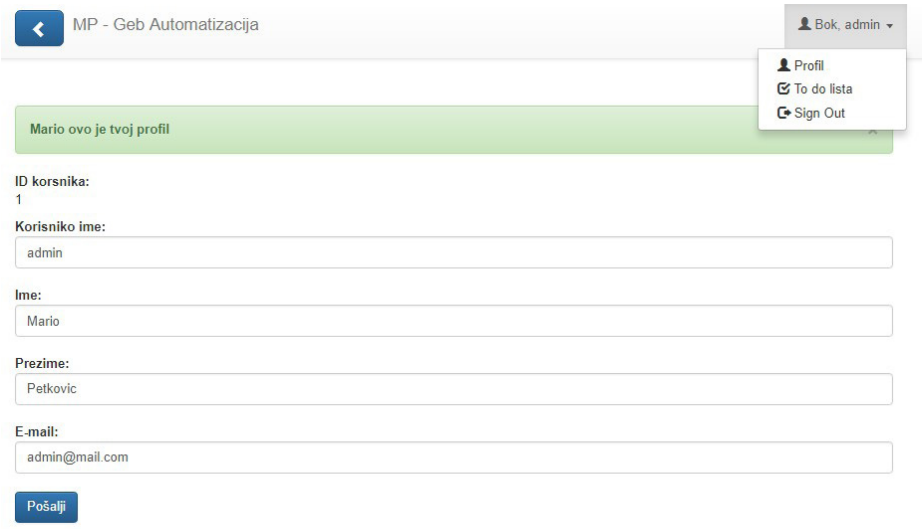
Izvor: izrada autora

Slika 13. Početna stranica



Izvor: izrada autora

Slika 14. Korisnički profil



Izvor: izrada autora

4. 3 Testni koraci

Na ovoj je *web*-aplikaciji primijenjena testna metoda koja pripada funkcionalnim UI testovima, a naziva se testiranje korisničkog puta (engl. *User journey test*) (Fowler, 2013) i predstavlja niz koraka koji čine mogući scenarij korisničke interakcije s aplikacijom (Szabo, 2017). Takav test pokriva moguću korisničku navigaciju kroz aplikaciju. Test ne mora provjeravati samo jednu stvar, već sve što se nađe na korisničkom putu.

Koraci:

1. Korisnik otvara *web*-aplikaciju
2. Unosi *e-mail* i neispravnu lozinku (prijava nije moguća)
3. Upisuje ispravnu lozinku te se prijavljuje u sustav
4. Preko padajućeg izbornika u alatnoj traci odlazi na svoj Profil
5. Izmjenjuje lozinku i sprema je
6. Klikom na gumb za povratak na alatnoj traci vraća se na početnu stranicu
7. Klikom na gumb za ispunjavanje upitnika otvara modalni prozor
8. Popunjava i šalje upitnik
9. Preko padajućeg izbornika u alatnoj traci odjavljuje se iz aplikacije
10. Ponavlja prijavu s novom lozinkom.

4. 3. 1 Izvođenje testa i rezultat

U ovom je primjeru korištena *strong typing* metoda pisanja kôda kojom se gubi na dinamičnosti, ali dobiva veća kontrola prilikom pisanja kôda, što je vrlo bitno kod velikih projekata. Primjer *strong typinga* dan je na slici 15.

Slika 15. Strong typing

```
//strong typing
WebAppLoginPage webAppLoginPage = browser.to(WebAppLoginPage)

webAppLoginPage.buttonPrijava.click()

//dinamičko pisanje
To WebAppLoginPage

buttonPrijava.click()
```

Izvor: izrada autora

User journey test na *web*-aplikaciji prikazan je na slici 16.

Slika 16. *User journey* test na *web*-aplikaciji

```
package com.mycompany.tests.webAplikacija
import
com.mycompany.component.library.webAplikacijaKomponente.WebAppHomePage
import
com.mycompany.component.library.webAplikacijaKomponente.WebAppProfilPage
import
com.mycompany.component.library.webAplikacijaKomponente.WebAppLoginPage
import Geb.Spock.GebReportingSpec
import Spock.lang.Stepwise
@Stepwise
class UserJourneyWebAppTest extends GebReportingSpec {
private static final EMAIL = "admin@mail.com"
private static final NEISPARVNA_LOZINKA = "pas"
private static final ISPRAVNA_LOZINKA = "password"
private static final NOVA_LOZINKA = "password123"

def "User journey test web apliakcije"() {
    given: "Stranica za prijavu trebala bi se prikazatii"
    WebAppLoginPage webAppLoginPage = browser.to(WebAppLoginPage)

    when: "Korisnik unosi e-mail i neispravnu lozinku"
    webAppLoginPage.inputEmail.value(EMAIL)
    webAppLoginPage.inputEmail.value() == EMAIL

    webAppLoginPage.inputPassword.value(NEISPARVNA_LOZINKA)
    webAppLoginPage.inputPassword.value() == NEISPARVNA_LOZINKA

    and: "Klik na gumb za prijavu"
    webAppLoginPage.buttonPrijava.click()

    then: "Prikaz poruke za neispravan emial/lozinku"
    webAppLoginPage.porukaPogreske.isDisplayed()

    when: "Korisnik unosi ispravnu lozinku"
    webAppLoginPage.inputPassword.value(ISPRAVNA_LOZINKA)
    webAppLoginPage.inputPassword.value() == ISPRAVNA_LOZINKA

    and: "Klik na gumb za prijavu"
    webAppLoginPage.buttonPrijava.click()

    then: "Početna stranica aplikacije i poruka dobrodoslice trebale bi
se prikazati"
    WebAppHomePage webAppHomePage = browser.at(WebAppHomePage)
    webAppHomePage.porukaUspjesnePrijave.isDisplayed()

    when: "Klik na padajući izbornik i zatim Profil"
    webAppHomePage.navigacijskaTraka.dropDownMeni.click()
```

Izvor: izrada autora

Slika 17 prikazuje rezultat testa provedenog na *web*-aplikaciji.

Slika 17. Rezultat testa provedenog na *web*-aplikaciji

Izvor: izrada autora

Test obuhvaća korake *user journeyja* i testira osnovne funkcionalnosti *web*-aplikacije.

Rezultat prolaznog testa (slika 17) izvršen je u 34.12 sekundi. Nakon testa uključena je i *cleanup()* metoda koja vraća lozinku na početnu vrijednost kako bi i u sljedećem pokretanju test mogao koristiti zadane podatke. Kôod prikazuje samo testnu klasu, a ne i Page klase budući da selekcija elemenata nije fokus ovog poglavlja, ali i zbog kompaktnosti rada.

4. 4 Pokretanje testova u oblaku (engl. *Cloud testing*)

Danas vrlo aktualno računalstvo u oblaku traži prilagodbu automatskog testiranja i na to okruženje. Iako se testiranje u oblaku može izvršavati i manualno pokretanjem aplikacija na virtualnim mašinama, najveća korist dobiva se automatskim testiranjem. Prednost pokretanja automatskih Selenium/Geb testova na nekoj platformi za testiranje u oblaku (na primjer, Sauce Labs, BrowserStack i tako dalje) je mogućnost izvođenja u različitim *web*-preglednicima, operacijskim sustavima i uređajima (Bruno, 2017), (Daley, Erdmann, Pragt, 2015).

S ciljem automatskog testiranja u oblaku, ranije pripremljeni testovi povezani su sa Sauce Labs platformom koja na svom GitHub repozitoriju (www.github.com) nudi velik broj različitih načina integracije testova sa svojom platformom, ovisno o testnom *frameworku*, programskom jeziku i *build* alatu. Struktura projekta vrlo je slična prikazanom projektu iz prijašnjeg poglavlja, ali postoje neke razlike koje omogućavaju povezivanje sa Sauce Labs platformom. Najvažnija klasa u novom projektu je BasePageGebSpec koja preuzima ulogu nasljeđivanja Spockove bazne klase za izvještaje (GebReportingSpec). Svaki test koji se uključuje u projekt i pokreće na Sauce Labs platformi mora naslijediti BasePageGebSpec klasu. To je zato što ona sadrži sve informacije vezane uz korisničke podatke za spajanje sa platformom, postavke drivera, ali i postavlja željene mogućnosti izvođenja testova, uključujući *web*-preglednik i operacijski sustav u kojem će se test izvoditi.

Slika 18. Dio BasePageGebSpec klase

```
//nasljedivanje class BasePageGebSpec extends
GebReportingSpec implements
SauceOnDemandSessionIdProvider {
public String username = System.getProperty('sauce.username')
public String accessKey = System.getProperty('sauce.key')
public String testDriver = System.getProperty('testDriver')
***

//setup metoda za po stavljanje desired capabilities testa
public void setup() throws Exception {
    if (!driverCreated || !isSpecStepwise()) {
        Map<String, String> capMap
        //testovi će se izvršiti u firefox browseru na Windows 10 OS-u
        String capabilityString = '{"browserName": "Firefox",
"platform": "Windows 10", "version": "42"}'

        if (capabilityString && testDriver == "sauce") {
            capMap = new JsonSlurper().parseText(capabilityString)
            DesiredCapabilities capabilities = new
DesiredCapabilities(capMap)
            String methodName = name.getMethodName()
String specName = this.class.getSimpleName()
if(isSpecStepwise()) {
                methodName = "All tests in " + specName
            }

              capabilities.setCapability("name", String.format("%s.%s",
 specName, methodName))
              capabilities.setCapability("newCommandTimeout", 180)
 driver = new RemoteWebDriver(
                      new URL("https://" + authentication.getUsername() +
 ":" + authentication.getAccessKey() +
                              "@ondemand.saucelabs.com:443/wd/hub"),
 capabilities)
               this.sessionId =

 (((RemoteWebDriver)
 driver).getSessionId()).toString()
        } else {
            FirefoxProfile profile = new FirefoxProfile()

 driver = new FirefoxDriver(profile)
         }
         driverCreated = true
    }
 }
```

Izvor: izrada autora

U navedenoj klasi moguće je uočiti i da implementira SauceOnDemandSessionIdProvider klasu koja provjerava status izvršenog testa i prikazuje ga na Sauce Labs korisničkom portalu. Nakon što se u konzolu spreme Sauce Labs korisnički podaci kao varijable radnog okruženja, testove je moguće paralelno pokrenuti pomoću *gradlew* naredbe.

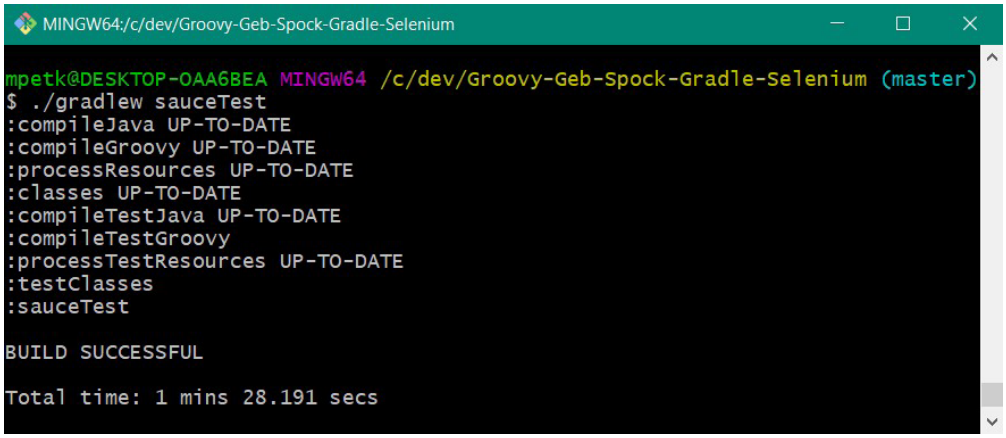
Na slikama 19 i 20 prikazano je pokretanje testova u konzoli i njihov rezultat.

Slika 19. Pokretanje testova u konzoli

```
//podaci za spajanje na Sauce Labs
$ export SAUCE_USERNAME=petkovicMario
$ export SAUCE_ACCESS_KEY=0971a578-b21d-4220-8512-d4da4e7852b7

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

//pokretanje testova

$ ./gradlew sauceTest
```

Izvor: izrada autora

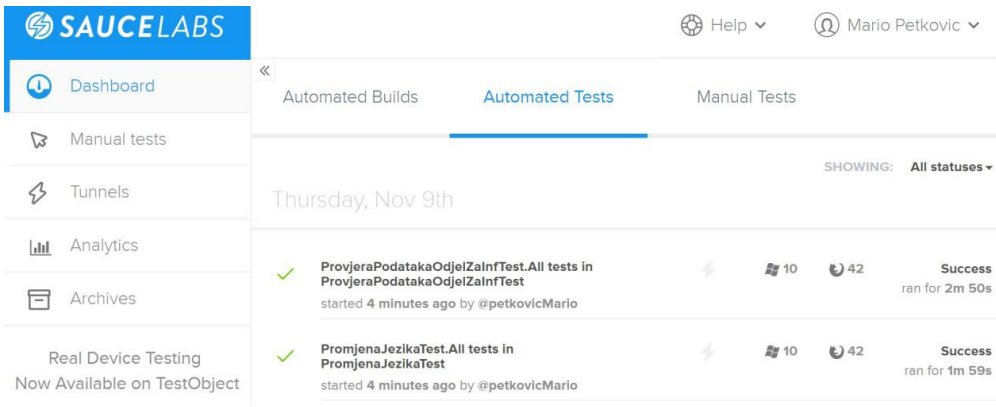
Slika 20. Pokretanje i rezultat testa



Izvor: izrada autora

Slika 21 prikazuje uspješno povezane i izvršene testove u oblaku. Ova vrsta platforme za testiranje u oblaku, osim samog rezultata testa, nudi i stvaranje izvještaja snimke zaslona i video izvršavanja testa koji može pomoći pri traženju pogreške u neprolaznim testovima. Važno je naglasiti da se automatski testovi mogu povezati sa Sauce Labs platformom preko *build* servera (na primjer Jenkins), što omogućava planirano automatsko pokretanje testova i nije potrebno ručno pokretanje kao u prikazanom primjeru. Ovakav način pokretanja vrlo je važan prilikom izvođenja većeg broja regresijskih testova (Gocheneur, 2016).

Slika 21. Rezultat pokrenutih testova na Sauce Labs platformi



Izvor: izrada autora

5. ZAKLJUČAK

U radu su predstavljeni rezultati istraživanja u okviru kojeg je izgrađeno okruženje za automatsko testiranje kombinacijom programskog jezika Groovy, *web*-drivera Geb i testnog *frameworka* Spock za pisanje i izvršavanje GUI funkcionalnih testova. Automatsko testiranje softvera značajno umanjuje trud potreban za testiranje i povećava broj izvršenih testova, što utječe na povjerenje u ispravnost softvera. Ipak, ne može se očekivati da će automatski testovi u potpunosti zamijeniti manualne. Ponekad je važan ljudski faktor pronalaska pogrešaka koji je jednostavno teško automatizirati. Iz tog razloga automatske testove ne treba gledati kao zamjenu, već kao nadopunu manualnom testiranju. Valja napomenuti da su preduvjet za uspješno automatsko testiranje dobro osmišljeni testovi, a tek onda se može govoriti o efikasnosti automatizacije. Jednom napisane automatske testove možda će s vremenom trebati doraditi i mijenjati.

Promatramo li testiranje *web*-aplikacija, unit i API automatski testovi imaju svoja ograničenja. *Web*-aplikacije postaju sve kompleksnije i sve raširenije na različitim platformama. Neke pogreške mogu se pojaviti na samo nekim verzijama *web*-preglednika ili na određenim operacijskim sustavima računala ili pametnih telefona. Za otkrivanje ovakvih pogrešaka korisni su automatski funkcionalni testovi koji testiraju aplikaciju kao krajnji korisnik. Tehnička strana, odnosno logika aplikacije, nebitna je testu dok god on dobiva očekivani rezultat. Drugim riječima, svi slojevi aplikacije, *back-end* i *front-end* testiraju se u isto vrijeme. No, efikasno pisanje i izvršavanje ove vrste testova predstavlja velik izazov. Testovi su vrlo skupi za pisanje, kompleksni za čitanje i vrlo teški za održavanje. U ovom radu prikazani su rezultati istraživanja integracije alata Geb, Groovy i Spock koja omogućava pisanje kvalitetnih, jednostavnih i održivih testova. U usporedbi s drugim sličnim alatima, preliminarni testovi koji su provedeni u okviru ove integracije, a dio ih je predstavljen u ovom radu, lako su čitljivi i uspješno se izvršavaju. U sljedećem koraku istraživanja ova će se kombinacija alata primijeniti na složenije primjere.

Dodatno je istražena i primjena automatskog testiranja *web*-aplikacija u oblaku. Bit će zanimljivo pratiti razvoj testiranja u oblaku kao i platforme koje pružaju tu uslugu. Naime, sve je veći broj proizvođača pametnih telefona i operacijskih sustava za njih koji će nastojati brže i kvalitetnije smanjiti rizik za pogreške putem testiranja. Alati prikazani u radu dobro se prilagođavaju u integraciji s testnim platformama u oblaku i može se očekivati da će se jednostavnost korištenja s vremenom sve više poboljšavati.

LITERATURA

„Automated Testing vs Manual Testing: Which Should You Use, and When?". https://www.apicasystems.com/blog/automated-testing-vs-manual-testing/, 7. 11. 2014. [17. 9. 2017.]

„Introduction to Testing with Spock and Groovy". http://www.baeldung.com/Groovy-Spock, [27. 12. 2018.]

„saucelabs-sample-test-frameworks". Internet: https://GitHub.com/saucelabs-sample-test-frameworks?utf8=%E2%9C%93&q=&type=&language=. [22. 12. 2018.]

„What is Software Testing Life Cycle (STLC)?". http://istqbexamcertification.com/what-is-software-testing-life-cycle-stlc/ [17. 9. 2017.]

Bruno, E. (2017) „Ten Reasons to Move to Cloud-based Testing". https://saucelabs.com/blog/ten-reasons-to-move-to-cloud-based-testing, 7. 2. 2017. [3. 11. 2017.]

Cohn, M. (2009) The Forgotten Layer of the Test Automation Pyramid, https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automationpyramid, 17. 12. 2009. [22. 9. 2017.]

Daley, L., Erdmann, M., Pragt, E. (2015) „The Book Of Geb". Internet: http://www.Gebish.org/manual/current/, 24. 6. 2015. [18. 10. 2017.]

Dustin, E., Garrett, T., Gauf, B. (2010) „Why automated software testing fails and pitfalls to avoid". https://www.edn.com/design/graphical-system-design/4199999/Why-automated-softwaretesting-fails-Part-I, 9. 6. 2010. [21. 11. 2017.]

Fernandes, J., Di Fonzo, A. (2017) „When to Automate Your Testing (and When Not To)". http://www.oracle.com/technetwork/topics/qa-testing/whatsnew/when-to-automate-testing-1-130330.pdf [19. 9. 2017.]

Fewster, M., Graham, D. (1999), Software test automation: effective use of test execution tools, Reading, MA: Addison-Wesley

Flecher, R. (2017) Spock: Up and Running: Writing Expressive Tests in Java and Groovy, Sebastopol: O'Reilly

Fowler, M. (2013) „User Journey Test". https://dzone.com/articles/user-journey-test, 8. 5. 2013. [2. 11. 2017.]

Gee, T. (2015) „Groovy Vs Java". http://trishagee.GitHub.io/presentation/groovy_vs_java/, 6. 4. 2015. [25. 10. 2017.]

Gocheneur, P. (2016) „Setting Up Sauce Labs with Jenkins". https://wiki.saucelabs.com/display/DOCS/Setting+Up+Sauce+Labs+with+Jenkins, 28. 6. 2016. [7. 11. 2017.]

Hambling, B. et at. (2015) Software Testing: An ISTQB-BCS Certified Tester Foundation guide, Swindon: British Computer Society

Kaner, C., Bach, J., Pettichord, B. (2002) Lessons Learned in Software Testing: A Context-Driven Approach, NY: WIley

Kapelonis, K. (2016) Java testing with Spock. Shelter Island, New York: Manning Publications

Kapelonis, K. (2017) „Spock testing framework versus JUnit". http://blog.codepipes.com/testing/spock-vs-junit.html#1-spock-enforces-a-clear-teststructure, 28. 4. 2017. [30. 10. 2017.]

König, D. et al. (2015) Groovy in action, Second edition. Shelter Island, NY: Manning

Ledbrook, P. (2015) „Groovy in light of Java 8". http://blog.cacoethes.co.uk/groovyandgrails/groovy-in-light-of-java-8, 16. 1. 2015. [27. 10. 2017.]

Mishra, D. D. (2016) „Learn Automation With Geb and Spock and get all benefit of Selenium WebDriver", http://www.abodeqa.com/2016/06/29/learn-automation-Geb-Spock/, 29. 6. 2016. [30. 10. 2017.]

Myers, G. J. et al. (2004) The art of software testing, Hoboken, NJ: Wiley & Sons

Niederwieser, P. (2016) „Spock Framework Reference Documentation". http://spockframework.org/spock/docs/1.1/all_in_one.html, 7. 11. 2016. [28. 10. 2017.]

Perry, W. E. (2006) Effective Methods for Software Testing, Indianapolis: Wiley

Rasmusson, J. (2016) The Way of the Web Tester: A Beginner's Guide to Automating Tests, Raleigh: Pragmatic Bookshelf

Subramaniam, V. (2013) Programming Groovy 2: Dynamic Productivity for the Java Developer, USA: The Pragmatic Programmers

Szabo, P. W. (2017) User experience mapping, Birmingham: Packt Publishing

Van Veenendaal, E., Graham, D. (2012) Foundations of Software Testing: Istqb Certification, 3. ed., Updated for ISTQB Foundation Syllabus 2011 and glossary 2.1. Andover: Cengage Learning

Van Vliet, H. (2008) Software engineering, Chichester: Wiley

Zhan, Z. (2014) Practical Web-Test Automation, Leanpub

Preliminary communication
https://doi.org/10.31784/zvr.7.1.15
Received: 15 January 2019
Accepted: 26 February 2019

AUTOMATIC TESTING OF WEB APPLICATIONS WITH THE SUPPORT OF GEB WEB DRIVER

Mario Petković
MA in Informatics, Coadria d.o.o., Prolaz Marije Krucifikse Kozulić 1, 51000 Rijeka, Croatia;
e-mail: mpetkovic.ri@gmail.com

Sanja Čandrlić
PhD, Assistant Professor, University of Rijeka, Department of Informatics, Radmile Matejčić 2,
51 000 Rijeka, Croatia; e-mail:sanjac@inf.uniri.hr

Martina Ašenbrener Katić
PhD, Postdoctoral Researcher, University of Rijeka, Department of Informatics, Radmile Matejčić 2,
51 000 Rijeka, Croatia; e-mail: masenbrener@inf.uniri.hr

ABSTRACT

Every day people encounter mistakes and bugs in software while they use it. Some of these bugs can be trivial, but some may be critical for some software functions. Since mistakes are inevitable, software testing requires a large amount of money, time and work. But despite efforts and investment, it is impossible to find all mistakes before software enters the market and production. Automatic testing can be very useful during that process. This paper presents methodology of automatic testing and its specific steps during testing of web applications, which is based on several chosen tools: Groovy programming language, Geb web driver and Spock test framework. The results have shown that this specific combination of tools presents an adequate and complete solution for automatic testing of web applications.

Key words: *automatic testing, Geb, Groovy, Spock, web application*