Automatsko testiranje i primjena alata za vizualnu usporedbu mrežnog sjedišta

Boras, Ana

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Humanities and Social Sciences / Sveučilište Josipa Jurja Strossmayera u Osijeku, Filozofski fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:142:673133

Sveučilište J.J. Strossmayera u Osijeku

Filozofski fakultet

Dvopredmetni diplomski studij informatologije i informacijskih tehnologija

Ana Boras

Automatsko testiranje i primjena alata za vizualnu usporedbu mrežnog sjedišta

Diplomski rad

Mentor: izv. prof. dr. sc. Boris Bosančić

Sumentor: Maja Klajić, mag. inform.

Osijek, 2020.

Sveučilište J.J. Strossmayera u Osijeku
Filozofski fakultet Osijek
Odsjek/samostalna katedra
Dvopredmetni diplomski studij informatologije i informacijskih tehnologija

---

Ana Boras

Automatsko testiranje i primjena alata za vizualnu usporedbu mrežnog sjedišta

Diplomski rad

Društvene znanosti, Informacijske i komunikacijske znanosti, Informacijsko i programsko inženjerstvo

Mentor: izv. prof. dr. sc. Boris Bosančić

Sumentor: Maja Klajić, mag. inform.

Osijek, 2020.

Prilog: Izjava o akademskoj čestitosti i o suglasnosti za javno objavljivanje

Obveza je studenta da donju Izjavu vlastoručno potpiše i umetne kao treću stranicu završnog odnosno diplomskog rada.

IZJAVA

Izjavljujem s punom materijalnom i moralnom odgovornošću da sam ovaj rad samostalno napravio te da u njemu nema kopiranih ili prepisanih dijelova teksta tuđih radova, a da nisu označeni kao citati s napisanim izvorom odakle su preneseni.
Svojim vlastoručnim potpisom potvrđujem da sam suglasan da Filozofski fakultet Osijek trajno pohrani i javno objavi ovaj moj rad u internetskoj bazi završnih i diplomskih radova knjižnice Filozofskog fakulteta Osijek, knjižnice Sveučilišta Josipa Jurja Strossmayera u Osijeku i Nacionalne i sveučilišne knjižnice u Zagrebu.

U Osijeku, datum 01.09.2020.

Ana Boras, 0122220417
ime i prezime studenta, JMBAG

Sažetak

Svrha ovog rada je pobliže prikazati proces automatskog testiranja. Shodno tome, u radu je dan osvrt na različite alate i tehnologije koji mogu unaprijediti i poboljšati proces automatskog testiranja. Rad se sastoji od dva dijela – teorijskog i praktičnog. U teorijskom dijelu rada objašnjeno je što je to testiranje programske podrške i programskog rješenja te je opisan životni ciklus navedenog procesa. Nadalje, navedeni su nedostaci programske podrške koji mogu utjecati na samo testiranje, kao i terminologija koja je direktno povezana sa samim testiranjem. Naglasak je na tzv. *bugovima* te je objašnjeno što je to *bug* i kako se određuje ozbiljnost i prioritet *buga*. Osim toga, u teorijskom dijelu rada također je opisana uloga testera u procesu testiranja programskog rješenja. S obzirom da, osim automatskog, postoji i ručno testiranje programskog rješenja, u radu je prikazana usporedba ovih dviju vrsta testiranja. Na samome kraju teorijskoga dijela navedene su različite razine i metode testiranja. U praktičnom dijelu u središtu pažnje je Five mrežno sjedište. Prije svega, opisano je što je sve potrebno napraviti kako bi se postavilo radno okruženje i instalirali alati potrebni za provedbu automatskog testiranja. Nakon toga, prikazano je ukupno dvadeset i dva uspješno provedena testa s odgovarajućim opisima. Na kraju praktičnog dijela opisani su problemi i izazovi na koje se naišlo prilikom testiranja Five mrežnog sjedišta. Kako bi se bolje prikazao postupak provedbe automatskog testiranja, praktični dio sadrži veliki broj slika koje jasno prikazuju strukturu testnih slučajeva, snimke zaslona mrežnog sjedišta i slično.

Ključne riječi: automatsko testiranje, vizualna usporedba, WebdriverIO, Selenium osiguravatelj kvalitete

Sadržaj

1. Uvod

U ovom radu govori se važnosti testiranja programske podrške i programskog rješenja. Postoji veliki broj prednosti testiranja programske podrške, a neke od njih su: isplativost (ako se programska podrška testira na vrijeme pomaže u dugoročnoj uštedi novca), sigurnost (programska podrška je pouzdana i sigurna), kvaliteta (osnovni zahtjev svakog klijenta) i zadovoljstvo klijenta.[1] Postupak testiranja osim programske podrške uključuje i testiranje programa za specifičnu namjenu te programskih rješenja poput dinamičkih mrežnih sjedišta. Programsko rješenje se može testirati ručno i automatski i svaka od ovih opcija ima svoje prednosti i nedostatke. Ovaj rad sastoji se od teorijskog i praktičnog dijela. Cilj ovog rada je ukazati na to koliko automatsko testiranje može povećati učinkovitost testera i kvalitetu njegova rada te predstaviti nekoliko alata za automatsko testiranje. U teorijskom dijelu rada opisano je što je to testiranje, dan osvrt na životni ciklus testiranja programskog rješenja, te pobrojani njegovi nedostaci. Prilikom testiranja otkrivaju se greške i mane programskog rješenja. Postoje različite vrste grešaka, ali i statusi u kojima se te greške mogu nalaziti. Upravo iz tih razloga, u ovom radu govori se i o životnom ciklusu grešaka, tj. njihovom prioritetu i ozbiljnosti. U nastavku rada objašnjeno je tko je tester i koji je njegov zadatak u procesu testiranja. Također, tester treba imati razvijene tehničke i ne-tehničke vještine. Nadalje, u radu se navode i objašnjavaju razine i metode testiranja. Četiri najčešće korištene razine testiranja su testiranje jedinica, integracijsko testiranje, testiranje sustava i testiranje prihvatljivosti, a najpoznatije metode testiranja su metode crne, bijele i sive kutije. Nakon teorijskog dijela rada, slijedi praktični dio na kojem je stavljen naglasak u ovom radu. U praktičnom dijelu automatski je testirano Five mrežno sjedište. Cilj je kreirati testove koji će osigurati kontinuiranu kvalitetu mrežnog sjedišta i prevenirati nastanak grešaka u produkciji. Također, pokazat će se kako se testiranje svakog mrežnog sjedišta može automatizirati uz poznavanje određenih alata. Također, ovaj rad može poslužiti kao osnova za učenje automatskog testiranja jer su u praktičnom dijelu opisani sljedeći korisni alati: WebdriverIO, Percy, TestRail, Allure i BrowserStack. Osim toga, opisani su i koraci koje je potrebno napraviti kako bi se isti i instalirali. Nakon toga, predstavljeni su svi testovi koji su kreirani za potrebe testiranja Five mrežnog sjedišta. Sveukupno, kreirana su dvadeset dva testa koji su u radu detaljno opisani te uz svaki priložena i odgovarajuća Percy i TestRail slika. Također, testovi su takvi da na njima može netko drugi nastaviti raditi zbog toga što su podložni promjenama te se mogu bez

[1] Usp. Guru99. URL: https://www.guru99.com/software-testing-introduction-importance.html#5 (2020-08-08)

problema nadopunjavati. Na samom kraju rada navedeni su određeni izazovi i problemi koji su nastali prilikom testiranja te zaključak.

2. Općenito o testiranju programske podrške i programskog rješenja

Testiranje programske podrške je pojam koji se može objasniti na više načina kako bi on bio razumljiv osobama koje se prvi put susreću s njim. Prije svega, potrebno je objasniti što je to programska podrška. Programska podrška podrazumijeva određeni skup programa i podataka koji su neophodni za rad na računalu, a to su svi neopipljivi dijelovi računalnog sustava.[2] Programska podrška kao takva treba proći kroz fazu razvoja, testiranja i ispravljanja grešaka. Postupak testiranja osim programske podrške uključuje i testiranje programskih rješenja. Prilikom testiranja programskog rješenja razmatraju se različita pitanja od samog izvođenja programskog rješenja, njegove stabilnosti i načina na koji se postupa s greškama. Kako bi se odgovorilo na pitanje kako programsko rješenje postupa s greškama potrebno je postaviti testni scenarij u određenim kontroliranim uvjetima te pratiti i procijeniti rezultate. Programsko rješenje testira se zbog toga da bi se uvidjelo zadovoljava li korisnička očekivanja i korisničke zahtjeve ili ne.[3] Osim toga, testiranje programskog rješenja obavezno je zato što svatko radi greške koje u većini slučajeva budu nenamjerne, ali mogu biti od velike važnosti za ispravan i konzistentan rad. Prilikom testiranja programskog rješenja potrebno je ukazati na nedostatke i pogreške koje su nastale u različitim fazama razvoja. Programeri mogu pogriješiti prilikom implementacije programa, a neki od razloga za to mogu biti nedostatak iskustva programera, nepoznavanje programskog jezika, nedovoljno iskustva u domeni, zbog složene logike ili jednostavno zbog ljudske pogreške. Ponekad ni sami zahtjevi za razvoj programa nisu dovoljno jasni i zbog toga se pokušava napraviti što više od onoga što je ponuđeno i dostupno.[4] Kada je riječ o kvaliteti programskog rješenja, ono se može definirati kao stupanj izvrsnosti, a program visoke kvalitete odgovara korisničkim zahtjevima. Kvalitetan i ispravan program štedi veliku količinu vremena i novca jer program koji ima manje oštećenja štedi vrijeme koje se trebalo iskoristiti na fazu testiranja i održavanja. Veća pouzdanost doprinosi neizmjernom povećanju zadovoljstva korisnika. Osiguravanje kvalitete programa predstavlja veliki dio ukupnog broja troškova, stoga će upravo ti troškovi biti manji ukoliko se gradi kvalitetno i ispravno programsko rješenje.[5] Postoji veliki broj slučajeva u kojima su se dogodile greške i koji ukazuju na važnost testiranja programa, a neki od njih su: China Airlines Airbus A300 srušio se zbog

[2] Usp. Programska podrška. // Hrvatska enciklopedija. URL: https://www.enciklopedija.hr/natuknica.aspx?ID=50557 (2020-07-01)
[3] Usp. C, Padmini. Beginners guide to software testing. [s.l.], [s.n.], str. 6. URL: https://www.softwaretestingclass.com/wp-content/uploads/2016/06/Beginner-Guide-To-Software-Testing.pdf (2020-07-01)
[4] Usp. Try QA. URL: http://tryqa.com/why-is-testing-necessary/ (2020-07-01)
[5] Usp. C, Padmini. Nav. dj., str. 6. (2020-07-01)

greške u programu 1994. godine; Nissan automobili povukli su s tržišta više od milijun automobila zbog programske greške vezane uz zračne jastuke, a zabilježene su i dvije prometne nesreće povezane s tom greškom itd. Neke od prednosti testiranja programa su: isplativost (jedna od najvažnijih prednosti; pomaže dugoročnoj štednji novca), sigurnost (najosjetljivija prednost testiranja jer korisnici i klijenti traže pouzdan proizvod), te kvaliteta i zadovoljstvo korisnika, što je zapravo i glavni cilj.[6]

2.1. Životni ciklus testiranja programskog rješenja

Pravilno testiranje osigurava otkrivanje grešaka i ostalih problema u ranom životnom ciklusu proizvoda ili aplikacije. Životni ciklus testiranja programskog rješenja sastoji se od nekoliko ključnih faza: planiranje, analiza, dizajn, faza testiranja jedinica, testiranje, završno testiranje i implementacija te naknadna implementacija. Svaka faza u životnom ciklusu programskog rješenja uključuje određene odgovarajuće aktivnosti. Prilikom planiranja dolazi do definiranja ciljeva, ispitivanja kvalitete, identificiranja problema i sl., a najbitnije je napraviti raspored testiranja i u njega uključiti što je više moguće vrsta testiranja. Analiza uključuje aktivnosti koje razvijaju funkcionalnu validaciju na temelju zahtjeva poslovanja (pisanje testnih slučajeva), razvijaju se oblici testnih slučajeva (procjena vremena i određivanje prioriteta), određuju se vremenski rokovi, identificiraju se testni slučajevi koji će se automatizirati (ako je moguće primijeniti automatizaciju), definiraju se područja za izvođenje različitih vrsta testova, definiraju se postupci za održavanje (sigurnosna kopija, validacija) te se pregledava dokumentacija. U fazi dizajna potrebno je revidirati testni plan te nastaviti pisati testne slučajeve i dodati nove koji se baziraju na promjenama, razviti kriterije za procjenu rizika i dovršiti testni plan (procijeniti resurse potrebne za testiranje jedinica). U fazi testiranja jedinica potrebno je ispuniti sve što je zadano u prethodnim fazama i započeti stres test i testiranje performansi kao i automatizaciju. U fazi testiranja potrebno je voditi se testnim slučajevima, prijavljivati greške i dodavati nove testne slučajeve ukoliko je to potrebno. Završno testiranje i implementacija odnosi se na izvođenje svih testnih slučajeva ručno i automatski na *frontendu* i *backendu* uz izvođenje svih vrsta testiranja i ažuriranje procjena za testne slučajeve i testne planove. U fazi naknadne implementacije može doći do sastanka na kojem se pregledava cijeli

[6] Usp. Guru 99. Nav. dj. URL: https://www.guru99.com/software-testing-introduction-importance.html (2020-07-01)

projekt, strategije koje su korištene te se pokušavaju identificirati problemi koji su se našli na putu kako bi se broj istih smanjio u budućnosti.[7]

2.2. Nedostaci programa

Postavlja se pitanje zašto programi ponekad ne rade ispravno, a odgovora na to pitanje može biti više; jedan od njih je taj da program stvara čovjek koji griješi. Ako netko pogriješi prilikom izrade programa, to vodi direktno do problema – program se koristi pogrešno i ne ponaša se u skladu s prethodnim očekivanjima. Takve greške dovode do nedostataka u samom programu i nazivaju se *bugovi*.[8] Ako se program ne ponaša onako kako je to predviđeno, onda u njemu postoje *bugovi*. Svrha testiranja je pronalaženje grešaka i ostalih mana, nedostataka i kvarova u radu programa, a greške koje su vezane uz program nalaze se u samom kôdu koji stvara programer i nalazit će se tamo dok god se one ne isprave.

Postoji veliki broj grešaka i zbog toga su one klasificirane i podijeljene u određene skupine. Razlikujemo greške vezane uz sintaksu, greške u postupku obrade, greške u preciznosti, greške zbog prekoračenja, greške zbog performansi, greške u dokumentaciji, greške u vremenskoj koordinaciji i greške zbog nepoštivanja standarda. Greške koje su vezane uz sintaksu povezane su s korištenjem programskog jezika. Ovakve greške najčešće se otkriju prilikom kompajliranja programa, a događaju se ako npr. programer zaboravi staviti zarez na kraju naredbe, ako se koristi drugačiji tip podatka, ako petlje nisu dobro strukturirane i sl. Također, u nekim slučajevima sintaksna greška ne može biti otkrivena kompajliranjem i to može utjecati na funkcionalnost što na kraju može dovesti do puno većih i ozbiljnijih posljedica. Greške u postupku obrade nastaju kada ulazni podaci nisu ispravno implementirani i najčešće se povezuju s nepravilnim grananjem u programu, parametrima u petlji koji mogu biti pogrešno zadani i sl. Ovakva vrsta greški može se otkloniti testiranjem programa te je potrebno pratiti izvršavanje programa i usporediti rezultate s onima koji su bili očekivani. Greške u preciznosti odnose se na formule koje ne nude precizan rezultat na kraju izračuna ili ako je formula dobra, a podaci nisu odgovarajućeg tipa, opet može doći do smanjenja preciznosti u dobivenim vrijednostima. Greške prekoračenja najčešće se javljaju zbog prekoračenih ograničenja koja su unaprijed definirana. Greške zbog performansi nastaju kada

[7] Usp. C, Padmini. Nav. dj., str. 13. (2020-07-01)
[8] Usp. Graham, Dorothy…[et. al.]. Foundations of software testing: ISTQB Certification. Australia…[et.al]: Cengage Learning Emea, 2008., str. 6. URL: http://www.freepmstudy.com/PDFFiles/ISTQB%20Foundations%20of%20software%20testing-Dorothy%20Graham.pdf (2020-07-03)

program ne postiže ono što je unaprijed predviđeno projektnim zadacima. Kako bi se izbjeglo da program radi sporo, potrebno je iste testirati u ekstremnim uvjetima. Kada se program testira u ekstremnim uvjetima, bilo bi dobro provjeriti i što bi s dogodilo kada bi se on maksimalno opteretio jer ako je neki program namijenjen za 12 korisnika, potrebno ga je testirati kada je svih 12 korisnika aktivno, ali i provjeriti što bi se dogodilo kada bi broj korisnika porastao. Greške u dokumentaciji javljaju se ako kôd i odgovarajuća dokumentacija nisu usklađeni. Kako bi se te greške razriješile potrebno je ispraviti dijelove dokumentacije tj. uskladiti dokumentaciju i kôd. Greške u vremenskoj koordinaciji odnose se na programe koji se izvršavaju u realnom vremenu, na one koji se trebaju izvršavati istovremeno i na one koji se izvršavaju u određenom, unaprijed definiranom redoslijedu. Greške koje se u tom slučaju događaju teško je ponoviti zbog toga što je teško ponovno uspostaviti istu okolinu zbog prevelikog broja parametara i upravo zato teško je provesti detaljniju analizu vezanu uz ove greške. Greške zbog nepoštovanja standarda nastaju kada se ne poštuje ono što je unaprijed definirano u proceduri izrade programa. Programer je taj koji se mora pridržavati svih dogovorenih procedura, a ako ih se ne pridržava, može doći do pojave grešaka koje neće smetati njemu ili narušavati kôd, ali će stvarati probleme dizajnerima i testerima koji u tom trenutku neće razumjeti samu logiku rada programa.[9]

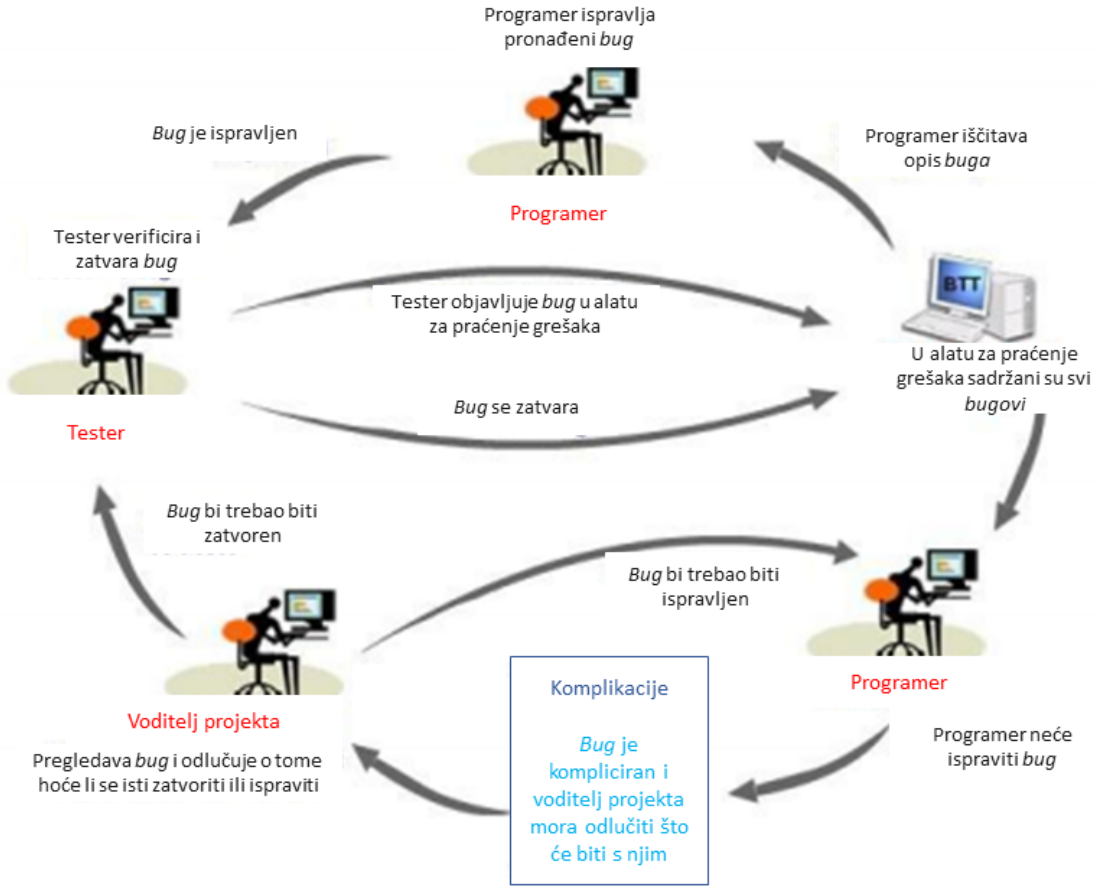
2.3. Životni ciklus *bugova*

Životni ciklus *bugova* započinje nenamjernom greškom u programu i završava kada programer kojem je dodijeljen *bug* isti i ispravi. Različite tvrtke koriste različite sustave za prijavljivanje *bugova* i ostalih zadataka vezanih uz određeni produkt i zbog toga je teško pronaći nekakvo jednoznačno rješenje koje će vrijediti za sve. U nastavku će se dati općeniti pregled životnog ciklusa *buga* i svih statusa kroz koje on može prolaziti.

Kada se pronađe greška tj. *bug* u radu aplikacije/programa/produkta, potrebno ga je prijaviti i dodijeliti programeru koji ga treba ispraviti. Nakon što se *bug* ispravi, isti je potrebno ponovno provjeriti i testirati, ali i ostale funkcionalnosti povezane s njim, kako bi se utvrdilo da se nisu stvorili problemi na drugim mjestima. Razlikuje se nekoliko različitih statusa životnog ciklusa *buga*: status Otvoren (*bug* je otvoren kada tester identificira područje koje je problematično), status Prihvaćen (*bug* je dodijeljen programeru koji će ga ispraviti), status Nije prihvaćen/Neće se popravljati (ako programer smatra da je ovo *bug* niskog prioriteta ili uopće

[9] Usp. Tomašević, Violeta. Razvoj aplikativnog softvera. Beograd: Univerzitet Singidunum, 2017., str. 129-132. URL: https://singipedia.singidunum.ac.rs/preuzmi/40784-razvoj-aplikativnog-softvera/3171 (2020-07-04)

nije *bug,* stavlja ga u ovo stanje i onda voditelj projekta odlučuje što će biti s njim), status Na čekanju (*bug* koji nije moguće ispraviti odmah), status Ispravljen (programer je ispravio *bug*), status Zatvoren (kada je *bug* ispravljen, i kada se tester uvjerio u njegovu ispravnost) i status Ponovno otvoren (ispravljene *bugove* testeri mogu ponovno otvoriti u slučaju da su se zbog tog *buga* stvorili problemi na nekom drugom mjestu).[10]



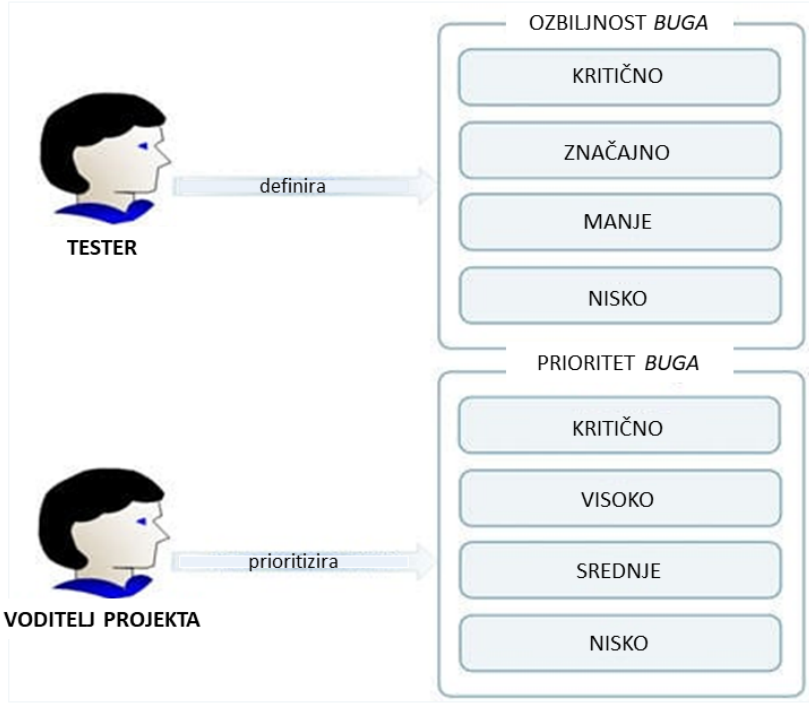
Slika 1. Životni ciklus *buga*.[11]

Na Slici 1. jasno je prikazan i opisan životni ciklus *buga* i kako se isti može pratiti uz korištenje različitih sustava za praćenje. Nadalje, na slikovit način prikazan je životni ciklus *bugova* i statusi u kojima se mogu naći, a koji su opisani u tekstu uz svaku sliku faze ispravljanja *buga*..

[10] Usp. C, Padmini. Nav. dj., str. 16. (2020-07-04)
[11] Usp. Isto, str. 17.

2.3.1. Ozbiljnost i prioritet *bugova*

Kada tester prijavljuje *bug*, uz sve podatke koje unosi, unosi i podatke o klasifikaciji samog *buga* što pomaže u učinkovitom procesu praćenja i održavanja greški u programu. Dva glavna parametra koja čine osnovu za učinkovito praćenje *bugova* su prioritet (engl. *priority*) i ozbiljnost (engl. *severity*) *buga.* Oba koncepta su vrlo slična što često dovodi do zabune prilikom klasifikacije. Prioritet se prema definiciji koristi u usporedbi dviju stvari ili stanja kada se jednom mora dati veći značaj nego drugom. Prioritet označava pozornost, prednost i važnost rješavanja zadatka. Stoga bi se prioritet mogao definirati kao pokazatelj hitnosti koja određuje što se prvo treba popraviti. Ozbiljnost je parametar koji označava koliko je kritična greška i koliko ta greška utječe na funkcionalnost cijelog sustava. Ozbiljnost definira tester, a prioritete definiraju svi poslovni sudionici uz voditelja projekta.[12]



Slika 2. Ozbiljnost i prioritet *buga*.[13]

[12] Usp. Software Testing Help. URL: https://www.softwaretestinghelp.com/how-to-set-defect-priority-and-severity-with-defect-triage-process/ (2020-07-05)
[13] Usp. Isto.

Na Slici 2. prikazane su razine koje se mogu odabrati prilikom određivanja prioriteta i ozbiljnosti. Prioritet je povezan s rasporedom zadataka, a ozbiljnost je povezana sa standardima. Prioritet se može klasificirati na nekoliko sljedećih načina:

- **Prioritet 1) Kritično** (engl. *immediate/critical*) – greška se mora riješiti što prije (po mogućnosti u roku 24 sata), a to se najčešće događa u slučajevima kada je blokirana cijela funkcionalnost i kada je program ili njegova značajka trenutno u neupotrebljivom stanju.
- **Prioritet 2) Visoko** (engl. *high*) – nedostatak koji ima ovaj prioritet sljedeći je kandidat koji se mora popraviti nakon što su kritični nedostaci ispravljeni.
- **Prioritet 3) Srednje** (engl. *medium*) – nedostatak koji ima ovaj prioritet potrebno je otkloniti nakon što se isprave svi *bugovi* s kritičnim i visokim prioritetom i u ovu skupinu spadaju svi manji nedostaci.
- **Prioritet 4) Nisko** (engl. *low*) – greška s niskim prioritetom ukazuje na to da definitivno postoji problem, ali ga nije potrebno riješiti odmah i u ovu skupinu spadaju greške vezane uz sintaksu, dizajn ili implementaciju sitnijih značajki koje bi poboljšale korisničko iskustvo.[14]

Osim prioriteta, klasificirati se može i ozbiljnost *buga*:

- **Ozbiljnost 1) Kritično** (engl. *critical*) – greška koja u potpunosti blokira testiranje programa kao npr. aplikacija koja se neprestano ruši ili bilo koje druge kritične greške koje dovode do toga da se aplikacija ne može koristiti.
  Primjer u praksi: prilikom prijave u Gmail unosi se korisničko ime i lozinka i umjesto da prijava bude uspješna, sustav se ruši i ova greška čini aplikaciju neupotrebljivom.
- **Ozbiljnost 2) Značajno** (engl. *major*) – svaka značajka koja ne ispunjava zahtjeve i ponaša se drugačije od očekivanog, svrstava se u ovu skupinu.
  Primjer u praksi: kada se sastavlja e-pošta u Gmail-u, nužno je omogućiti dodavanje više primatelja u CC odjeljak, a ako se to ne može učiniti, to upravo predstavlja značajan nedostatak jer jedna od glavnih funkcionalnosti sustava ne radi ispravno.
- **Ozbiljnost 3) Manji** (engl. *minor*) – svaka implementirana značajka koja ne ispunjava zahtjeve i ponaša se drugačije od očekivanog, ali je njezin utjecaj do neke mjere

[14] Usp. Isto.

zanemariv ili nema veliki utjecaj na aplikaciju može se svrstati u ovu skupinu, a do ovakvih grešaka dolazi kada se ne ispunjavaju određeni kriteriji.

Primjer u praksi: na Gmail-u postoji opcija pod nazivom „Uvjeti i odredbe“ te se u njoj nalazi više poveznica koje vode na druga mrežna mjesta i ako jedna od tih poveznica ne radi dobro, to predstavlja grešku manje ozbiljnosti zbog toga što ne dolazi do gubitka podataka i nema kvara u radu sustava nego je u pitanju doživljaj korisničkog iskustva.

- **Ozbiljnost 4) Nisko** (engl. *low*) **–** bilo koji nedostaci vezani uz pravopisne greške mogu se svrstati u ovu skupinu i ove greške kada se pojave gotovo i da nemaju utjecaj na funkcionalnost sustava, ali se svakako trebaju ispraviti.

  Primjer u praksi: na Gmail stranici postoje razne poveznice na određene licence koje mogu sadržavati pravopisne greške ili određene neusklađenosti. Ove greške spadaju u greške niske ozbiljnosti koje nemaju utjecaj na ponašanje sustava, gubitak podataka, korisničko iskustvo itd.[15]

2.4. Tko je tester i što on radi?

Kao što sama riječ kaže, tester je osoba koja testira određeni program/produkt/aplikaciju i bilo što drugo za što je u tom trenutku zadužena. Kao tester, osoba treba kreirati testne slučajeve, skripte i generirati podatke. Također, testeri ubrzavaju proces razvoja programskog rješenja prepoznavanjem grešaka u ranoj fazi, smanjuju rizik organizacije, povećavaju kvalitetu programa, osiguravaju uspješno lansiranje proizvoda uz maksimalne uštede novca, vremena i ugleda tvrtke te promiču kontinuirano poboljšanje.[16] Vještine koje tester treba imati mogu se podijeliti u dvije skupine, a to su tehničke i ne-tehničke vještine (engl. *non-technical/soft skills*). Ne-tehničke vještine ključne su za testera, kao i za ostale sudionike u razvoju programa, a neke od njih su: analitičke vještine, komunikacijske vještine, organizacijske vještine, motivacija i sl. Analitičke vještine mogu pomoći u podjeli programa na manje jedinice kako bi se steklo bolje razumijevanje programa te zbog lakšeg pisanja testnih slučajeva. Svaki tester treba imati razvijene vještine usmene i pismene komunikacije, a testni slučajevi koje je napisao tester trebaju biti jasni i razumljivi. Testiranje ponekad može biti zahtjevan posao, posebice prilikom distribucije kôda, kada je na testeru veliko opterećenje s kojim se svaki tester mora naučiti nositi. Lista tehničkih vještina je dugačka, a neke od njih su: osnovno znanje o bazama podataka, iskustvo u korištenju alata za praćenje grešaka, znanje vezano uz automatsko

[15] Usp. Isto.
[16] Usp. C, Padmini. Nav. dj., str. 7. (2020-07-07)

testiranje i sl.[17] Postavlja se pitanje koje bi osobine trebao posjedovati dobar tester, a u literaturi navodi se sljedeće: tester treba poznavati najnoviju tehnologiju, treba biti perfekcionist, ali i realist, treba biti uvjerljiv i imati određenu taktiku. Nadalje, tester treba biti istraživački nastrojen i biti spreman upuštati se u nepoznate situacije, treba biti uporan, organiziran, objektivan, točan itd.[18]

U literaturi se spominju određene smjernice za nove testere koje bi im trebale olakšati rad i prilagodbu na nove situacije u kojima se mogu pronaći, a u nastavku će se navesti i opisati neke od tih smjernica. Prva navedena smjernica govori o tome da testiranje ne pokazuje da *bugovi* ne postoje. Bitna stavka testiranja je spriječiti nedostatke. Tester može pronaći i prijaviti *bugove*, ali ni u jednom trenutku ne može jamčiti da u određenom programu nema više grešaka. Također, nijedan program/aplikacija/produkt nije 100% „*bug free*“. Druga smjernica se nadovezuje na prvu, a govori o tome kako se program ne može testirati u potpunosti, a ista stvar je i s najjednostavnijim programom jer dolazi do čestih promjena u specifikacijama. Tester ne može jamčiti kvalitetu jer se svaka stavka ne može testirati i on nije odgovoran za kvalitetu. Nadalje, jedna od stvari na koju tester treba obratiti posebnu pažnju je prijava detaljno i točno opisane greške kako bi programeru ili trećoj osobi bilo jasno na što se ona odnosi. Sljedeća smjernica govori o ciljanom okruženju i krajnjim korisnicima koji su glavni čimbenici koji se trebaju sagledati i uzeti u obzir. U tom slučaju treba se razmotriti je li korištenje aplikacije namijenjeno za jednu osobu ili više njih i prema tome postupati dalje. Tester se prilikom testiranja treba postaviti u poziciju laika koji ne zna ništa o aplikaciji i na taj način provesti testiranje jer će upravo tako pronaći najviše grešaka i nelogičnosti u samoj aplikaciji. Također, tester treba graditi i svoju vjerodostojnost u provedbi testiranja koja je poput kvalitete koja uključuje pouzdanost, znanje, dosljednost, povjerenje, stav i pažnju prema detaljima. Vjerodostojnost se gradi s vremenom, a lakše će se steći ako tester prepozna svoje snage i slabosti, ako je u stanju prilagoditi se i ako je spreman priznati greške koje je učinio. Prilikom testiranja ništa ne treba pretpostavljati, nego sve treba provjeriti. Nadalje, važno je zapamtiti da svi pronađeni *bugovi* neće uvijek biti ispravljeni. Odluka koji će se ispraviti, a koji neće temelji se na riziku. Neki od razloga zašto neki *bug* neće biti ispravljen je taj što tester nema dovoljno vremena na raspolaganju kao i činjenica da on korisnika ne ometa u normalnom korištenju aplikacije; u međuvremenu, razvile su se nove, drugačije značajke programskog rješenja, pa popravljanje tog *buga* može biti i vrlo rizično. Pretposljednja smjernica govori o

[17] Usp. Guru 99. URL: https://www.guru99.com/software-testing-career-complete-guide.html (2020-07-07)
[18] Usp. C, Padmini. Nav. dj., str. 8. (2020-07-07)

tome kako je dobro proučavati konkurentske proizvode kako bi se stvorio uvid u različitosti. Dobro je upoznati se s funkcionalnošću konkurentskog proizvoda i njegovim ponašanjem jer to može olakšati pisanje testnih slučajeva i može pomoći u shvaćanju prednosti i nedostataka proizvoda na kojem se radi. Ujedno može doprinijeti i razvoju novih, poboljšanih značajki za testirani proizvod. Posljednja, ali ne i manje važna smjernica govori o tome kako je potrebno slijediti standarde i procese.[19] Tester prilikom testiranja nije pristran zbog toga što ne testira svoj vlastiti proizvod, ali unaprijed mora steći određena znanja o programskom rješenju koji testira kako bi mogao uspješno kreirati testne slučajeve.[20]

2.5. Vrste testiranja

Razlikujemo dvije glavne vrste testiranja, a to su ručno i automatsko testiranje. Ručno testiranje je proces koji uključuje korake koji osiguravaju da se mrežna stranica testira korak po korak. Kao što sama riječ kaže, mrežnu stranicu (ili bilo koji drugi program/produkt/aplikacija) tester ručno testira. Testeri koji provode testiranje ručno osiguravaju da mrežna stranica radi pravilno pod različitim uvjetima. Ručno testiranje postupak je pronalaska greški, nedostataka ili problema na mrežnom mjestu bez korištenja alata za testiranje. Prilikom testiranja provjerava se dizajn, korisničko sučelje, funkcionalnost i performanse mrežne stranice prilikom klika na različite elemente na mrežnom mjestu.[21] Premda je, ova vrsta testiranja jedna je od najprimitivnijih vrsta testiranja, svaka nova aplikacija najprije mora biti ručno testirana prije procesa automatizacije.[22] Testeri provjeravaju stvarno ponašanje programa prema njegovom očekivanom ponašanju koje je unaprijed zadano, a svako odstupanje prijavljuje se kao greška. Ručno testiranje važan je dio svake strategije testiranja jer pomaže testerima da steknu bolji uvid iz ugla krajnjeg korisnika. Budući da ručno testiranje provodi čovjek bez intervencije okvira automatskog testiranja, on prosuđuje programsko rješenje iz najvažnije značajke: korisničkog iskustva (engl. *user experience – UX*). Postoje razni savjeti kako uspješno ručno testirati programsko rješenje, a neki od njih su: analizirati projektnu dokumentaciju, kreirati jasan testni plan, napisati testne slučajeve koji uključuju sve zahtjeve definirane u projektnoj

[19] Usp. Isto, str. 9-10.
[20] Usp. Spillner, Andreas; Linz, Tilo; Schaefer, Hans. Software Testing Foundations: A study guide for the certified tester exam. Santa Barbara: Rocky Nook Inc., 2014., str. 32. URL: http://prof.mau.ac.ir/images/Uploaded_files/Software%20Testing%20Foundations%20A%20Study%20Guide%20for%20the%20Certified%20Tester%20Exam%5B5309302%5D.PDF (2020-07-07)
[21] Usp. Shah, Jaymine. Beginner's manual software testing guide, 2019. URL: https://medium.com/techcompose/beginners-manual-software-testing-guide-8096bcd99a14 (2020-07-08)
[22] Usp. Guru 99. URL: https://www.guru99.com/manual-testing.html (2020-07-08)

dokumentaciji, poslati nadređenoj osobi testne slučajeve na pregled, testirati program uz pomoć testnih slučajeva te pronaći greške i na samom kraju prijaviti te iste greške.[23]

Automatsko testiranje provodi se uz pomoć alata za automatizaciju. Prilikom ove vrste testiranja ponekada je potrebno više vremena provesti na održavanju skripte u kojoj se nalaze napisani testovi, nego na testiranju temeljenom na istraživanju (engl. *exploratory testing*) po programu. Automatsko testiranje pogodno je za velike projekte, zatim za projekte koji zahtijevaju testiranje istih područja svaki put iznova i za one koji su već prošli kroz početni postupak ručnog testiranja.[24] Troškovi automatizacije su u početku nešto veći, ali kada su skripte spremne, one se mogu izvoditi nebrojeno puno puta s istom točnošću i prilično brzo, što štedi sate i sate ručnog testiranja što ima za posljedicu značajno snižavanje troškova testiranja. Postoji veliki broj situacija kada se više isplati automatsko testiranje nego ručno, a neke od tih situacija su:

- usporedba dviju slika, 'piksel po piksel'
- usporedba dviju proračunskih tablica koje sadrže tisuće stupaca i redaka
- testiranje aplikacije koju koristi +/- 100 000 korisnika (tzv. testiranje opterećenja)
- testiranje aplikacije u različitim mrežnim preglednicima i na različitim operacijskim sustavima paralelno.[25]

Izgradnja uspješne strategije automatskog testiranja je teška i pristup se razlikuje ovisno o timu. Neki timovi mogu imati više testera zaduženih za ručno testiranje nego automatsko, dok neki timovi ovise o programerima i njihovom vremenu koje mogu utrošiti na testiranje. Uz pomoć automatskog testiranja, vrijeme testiranja se drastično skraćuje, ali najviše se vremena troši na pisanje tj. kodiranje testova te njihovo ponavljanje i poboljšanje zbog čestog prilagođavanja.[26] Automatsko testiranje programa važno je zbog toga što je jako teško ručno testirati višejezične stranice i jer automatizacija ne zahtjeva ljudsku intervenciju (potrebno je samo pokrenuti testove). Nadalje, automatizacija povećava brzinu izvođenja testa, a testovi povećavaju pokrivenost programa. Jedna od bitnih funkcionalnosti automatskog testiranja je održavanje koje je potrebno zbog poboljšanja učinkovitosti skripti.[27]

---

[23] Usp. Unadkat, Jash. Manual testing for beginners, 2020. URL: https://www.browserstack.com/guide/manual-testing-tutorial (2020-07-08)
[24] Usp. Smartbear. URL: https://smartbear.com/learn/automated-testing/what-is-automated-testing/ (2020-07-09)
[25] Usp. Software testing help. URL: https://www.softwaretestinghelp.com/automation-testing-tutorial-1/ (2020-07-09)
[26] Usp. SmartBear. Nav. dj. (2020-07-09)
[27] Usp. Guru 99. URL: https://www.guru99.com/automation-testing.html (2020-07-09)

2.5.1. Usporedba ručnog i automatskog testiranja

Uz pomoć ručnog testiranja čovjek izvršava testne slučajeve. To znači da je on unaprijed pripremio i sastavio testne i rubne slučajeve kojima želi provjeriti funkcionalnost. Prilikom ručnog testiranja ne koriste se alati za automatizaciju. Pomoću automatskog testiranja sve se, kao što to samo ime kaže, automatizira. Koriste se alati i skripte u kojima se nalaze podaci koji se trebaju procesuirati. I ručno i automatsko testiranje imaju svoje prednosti i nedostatke. Jedna od prednosti ručnog testiranja je da tester ima potpunu kontrolu nad izvršavanjem svake akcije. A kako ručno testiranje ne zahtjeva nikakve alate za testiranje, troši se i manje novca. Prilikom ručnog testiranja često se može naići na odstupanja od dizajna. Ta odstupanja mogu otkriti testeri prilikom promatranja dizajna, a to se s alatima za automatizaciju ne može (ispravno) otkriti. Jedan od nedostataka je taj što je za ručno testiranje potrebno više vremena. Proces je spor i sklon ljudskim greškama. Također, veliki broj testova teško je simulirati ručnim testiranjem, posebice ako je riječ o stres testiranju prilikom kojeg se testira veliki broj zahtjeva u kratkom vremenskom periodu te je nemoguće da tester koji ručno testira program uspješno izvrši ovaj test. Što se tiče automatskog testiranja, ono je pouzdano i uvijek daje isti rezultat koji se može ponovno reproducirati. Budući da je svaki korak automatiziran, ne podliježe ljudskim greškama kao ručno testiranje. Testna automatizacija omogućuje pokretanje svih testova puno brže te je moguće simulirati složene scenarije. Nadalje, uz pomoć automatskog testiranja do programera puno brže stižu povratne informacije. Nedostaci automatskog testiranja se ogledaju u tome da ono ne može koristiti za otkrivanje odstupanja od dizajna jer je za to ipak predviđeno ljudsko oko. Također, alati za automatizaciju prilikom testiranja u nekim situacijama neće pokriti sve scenarije, a nije ni moguće svaki slučaj automatizirati. U automatizaciju ne vrijedi ni ulagati kada su projekti manjeg opsega zbog čestih promjena značajki sustava koji se testira.[28] Ručno testiranje najbolje je koristiti za testiranje upotrebljivosti, ad-hoc testiranje i testiranje istraživanjem. Testiranje upotrebljivosti usredotočeno je na mjerenje korisničke upotrebe aplikacije. Provodi se kada se pokušava testirati koliko je program jednostavan za upotrebu, učinkovit i prikladan za krajnje korisnike. Ad-hoc testiranje koristi slobodan pristup i izvodi se bez ikakvog formalnog plana ispitivanja i testnih slučajeva. Testeri prilikom ad-hoc testiranja pokušavaju testirati program uz pomoć različitih kombinacija na temelju iskustva i nagađanja o greškama. Također, ovo testiranje pomaže u prepoznavanju skrivenih greški koje su se mogle propustiti u svim prethodnim

[28] Usp. Testim. URL: https://www.testim.io/blog/test-automation-vs-manual-testing/ (2020-07-13)

pokušajima testiranja. Testiranje istraživanjem fokusira se na prikupljanje informacija i znanja o aplikaciji. Ova vrsta testiranja najkorisnija je kada je dokumentacija s tehničkim podacima loše napisana i kada se ne zna puno o specifikacijama i značajkama. Automatsko testiranje dobro je za regresijsko testiranje, ispitivanje performansi i ispitivanje opterećenja. Regresijsko testiranje provodi se kada se kôd često mijenja i najvažnija je aktivnost u održavanju programa i osiguranju kvalitete, pouzdanosti i valjanosti programa nakon izmjene. Ispitivanje performansi provodi se radi utvrđivanja brzine i stabilnosti aplikacije. Pomaže u prepoznavanju prirode i ponašanja u okviru očekivanog rada. Ispitivanje opterećenja vrsta je nefunkcionalnog testiranja programa koja se vrši kako bi se razumjelo ponašanje aplikacije pod određenim očekivanim opterećenjem. Ako se kôd promijeni nekoliko puta, također je bolje provesti test opterećenja.[29]

2.6. Razine testiranja

Četiri najčešće korištene razine testiranja (svaka sa svojim ciljevima) su: testiranje jedinica (engl. *Unit testing*) (služi za traženje grešaka i provjeru funkcionalnosti), integracijsko testiranje (engl. *Integration testing*) (služi za testiranje različitih dijelova programa), testiranje sustava (engl. *System testing*) (služi za testiranje sustava prema unaprijed utvrđenim zahtjevima) i testiranje prihvatljivosti (engl. *Acceptance testing*) (služi za provjeru valjanosti u skladu s potrebama korisnika).[30]

Prilikom testiranja jedinica traže se nedostaci i provjerava se funkcioniranje programa. Ova razina testiranja može se izvesti neovisno o ostatku programa. Također, može uključivati testiranje funkcionalnosti i specifičnosti koje nisu karakteristične.[31] Tijekom ovog testiranja potrebno je usredotočiti se na određene jedinice ili dijelove programa kako bi se utvrdilo je li svaka u potpunosti funkcionalna. Glavni cilj ovog nastojanja je odrediti funkcionira li aplikacija onako kako je zamišljeno. U ovoj se fazi jedinica može odnositi na funkciju, pojedinačni program ili čak postupak, a za obavljanje posla obično se koristi metoda bijele kutije (engl. *white box testing*). Jedna od najvećih prednosti ove faze testiranja je ta što se ona može izvoditi svaki puta kada se promijeni dio kôda, omogućujući rješavanje problema što je brže moguće. Uobičajeno je da testiranje jedinica provodi programer prije isporuke programa testerima. Integracijsko testiranje pruža mogućnost kombiniranja svih jedinica unutar

[29]Usp. Scientech Easy. URL: https://www.scientecheasy.com/2019/09/difference-manual-vs-automation-testing.html/ (2020-07-13)
[30] Usp. Graham, Dorothy…[et. al.]. Nav. dj., str. 40. (2020-07-20)
[31] Usp. Isto, str. 44.

programa. Ova razina testiranja osmišljena je za pronalaženje grešaka između različitih funkcionalnosti. Uz pomoć toga određuje se učinkovitost rada jedinica u grupi. Na funkcionalnost programa utječu jedinice koje nisu pravilno integrirane, bez obzira koliko su učinkovite u radu. Za pokretanje ove razine testiranje mogu se koristiti razne metode, ali specifična metoda koja će se koristiti uvelike ovisi o načinu definiranja jedinica.[32] Što je veći opseg integracije, to je teže izolirati greške na određenom sučelju, što može dovesti do povećanog rizika i do različitih pristupa integracijskom testiranju. Jedna krajnost je da su svi dijelovi sustava integrirani istovremeno, nakon čega se testiranje provodi na razini programskog rješenja u cjelini. To se naziva integracijsko „*big-bang"* testiranje. Prednost ovog testiranja ogleda se u činjenici da je postupak testiranja završen prije početka integracijskog testiranja te nije potrebno simulirati nedovršene dijelove. Glavni nedostatak ovog testiranja je taj što je u većini slučajeva teško pronaći uzrok neuspjeha. Druga krajnost je da su svi programi integrirani jedan po jedan, a testiranje se provodi nakon svakog koraka. Između ove dvije krajnosti postoji niz drugih varijanti. U svakoj fazi integracije testeri se koncentriraju isključivo na samu integraciju. Npr., ako integriraju dio A s dijelom B, oni trebaju testirati komunikaciju između ta dva dijela, a ne njihovu funkcionalnost.[33] Testiranje sustava prva je razina u kojoj se program testira u cjelini. Cilj ove razine testiranja je procijeniti je li sustav ispunio sve unaprijed utvrđene zahtjeve i vidjeti ispunjava li standarde kvalitete. Testiranje sustava provode testeri koji nisu bili dio razvoja programa. Testiranje sustava vrlo je važno jer potvrđuje ispunjava li program tehničke, funkcionalne i poslovne zahtjeve koje je postavio klijent.[34] Nadalje, testiranje sustava najčešće je završni test u razvoju i potrebno je omogućiti sustav koji zadovoljava tražene specifikacije. Svrha testiranja sustava je pronaći što je moguće više grešaka.[35]

Završna razina testiranja je testiranje prihvatljivosti koja se provodi kako bi se utvrdilo mogu li krajnji korisnici koristiti sustav. Promjene se ponekad mogu pogrešno protumačiti na način koji ne zadovoljava predviđene potrebe korisnika. Tijekom ove završne razine korisnik će testirati program kako bi otkrio zadovoljava li isti potrebe poslovanja. Program se nakon uspješnog završetka ovog procesa započinje proizvoditi.[36] U okviru testa prihvatljivosti razlikuju se dvije vrste ispitivanja koje se obično pripremaju i izvode odvojeno. Korisnički test

[32] Usp. Pearson, LaTonya. The four levels of software testing, 2015. URL: https://www.seguetech.com/the-four-levels-of-software-testing/ (2020-07-20)
[33] Usp. Graham, Dorothy…[et. al.]. Nav. dj., str. 45-46. (2020-07-20)
[34] Usp. Pearson, LaTonya. Nav. dj. (2020-07-20)
[35] Usp. Graham, Dorothy…[et. al.]. Nav. dj., str. 46. (2020-07-20)
[36] Usp. Pearson, LaTonya. Nav. dj. (2020-07-20)

prihvatljivosti usredotočen je uglavnom na funkcionalnost i na taj način provjerava se prikladnost sustava za korištenje, dok operativni test prihvatljivosti potvrđuje ispunjava li sustav uvjete za rad. Ako je program razvijen za masovno tržište, često ga je nemoguće testirati i takav program većinom prolazi kroz dvije faze testiranja prihvatljivosti, a to su alfa i beta testiranje.[37] Alfa testiranje provodi se zbog identificiranja svih mogućih problema i grešaka prije omogućavanja dostupnosti programa. Fokus ovog testiranja je simuliranje stvarnih korisnika pomoću tehnika crne i bijele kutije (engl. *black and white box testing*). Cilj je izvršiti zadatke koje bi tipični korisnik mogao obavljati, a ispitivači su većinom zaposlenici tvrtke. Najjednostavnije rečeno, ovakva vrsta ispitivanja naziva se alfa samo zato što se provodi rano, potkraj razvoja programa i prije faze beta testiranja. Beta testiranje obavljaju stvarni korisnici u realnom okruženju i može se smatrati oblikom vanjskog testiranja prihvatljivosti. Pristup beti verziji programa omogućava se ograničenom broju krajnjih korisnika kako bi se dobile povratne informacije o njegovoj kvaliteti. Beta testiranjem smanjuje se rizik od neuspjeha i omogućuje se povećana kvaliteta. To je posljednji test prije omogućavanja dostupnosti proizvoda korisnicima, a glavna prednost beta testiranja su izravne povratne informacije korisnika.[38]

2.7. Metode testiranja

Testiranje programa ključna je aktivnost u procesu razvoja programa. [39]

Testiranje metodom crne kutije odnosi se na tehniku dobivanja testnih slučajeva. Također, ova metoda provjerava funkcionalnost programa i zbog toga se naziva još i funkcionalno testiranje. Testiranje metodom crne kutije osigurava pokazatelje koji govore jesu li svi funkcionalni zahtjevi koje je korisnik naveo zadovoljeni. Isto tako, provjerava se postoji li neka funkcija u programu koja se nepravilno implementirala, kao i nedostaje li nešto i postoji li kakva greška. Uz pomoć testiranja metodom crne kutije može se otkriti postoji li problem s bazom podataka i pohranjivanjem podataka. Testiranje metodom crne kutije obično se obavlja u kasnijoj fazi testiranja, tj. kada je GUI (engl. *graphical user interface*) programa spreman. Prilikom testiranja metodom crne kutije testeri ne trebaju posjedovati znanje programiranja jer se testira funkcionalnost programa.[40]

---

[37] Usp. Graham, Dorothy…[et. al.]. Nav. dj., str. 48. (2020-07-20)
[38] Usp. Guru 99. URL: https://www.guru99.com/alpha-beta-testing-demystified.html (2020-07-20)
[39] Usp. Binary Terms: URL: https://binaryterms.com/difference-between-black-box-testing-and-white-box-testing.html (2020-07-21)
[40] Usp. Isto.

Testiranje metodom bijele kutije naziva se još i strukturno ispitivanje. Testiranje metodom bijele kutije vrši se na proceduralnoj strukturi programa, što uključuje testiranje svih logičkih puteva u programu. Također, provjerava se rade li sve komponente programa ispravno. Ako se testiranje metodom bijele kutije izvodi s velikom pažnjom, rezultat je 100% ispravan program. Uz pomoć metode bijele kutije, testiraju se sve petlje u programu i interne strukture podataka kako bi se osigurala valjanost pohranjenih podataka. Prilikom testiranja metodom bijele kutije, testeri trebaju posjedovati znanje programiranja jer ovo testiranje uključuje strukturno testiranje. U većini slučajeva, ovo testiranje provode programeri jer su upućeni u implementaciju svih komponenti programa. Također, ispitivanje metodom bijele kutije može se provesti u ranoj fazi testiranja tj. prije nego što GUI programa bude spreman. Testiranje metodom bijele kutije vrlo je dugotrajan i iscrpan proces jer uključuje temeljito testiranje programa i detaljnu analizu proceduralnog dizajna programa.[41]

Testiranje metodom sive kutije tehnika je testiranja programa s djelomičnim poznavanjem unutarnjeg rada aplikacije. Svrha ove metode testiranja je pretraživanje grešaka zbog nepravilne strukture kôda ili nepravilnog funkcioniranja programa. Testiranje metodom sive kutije provodi se iz nekoliko razloga: smanjuje se dugotrajni proces ispitivanja, programer ima dovoljno vremena popraviti greške, testiranje se vrši s korisničkog stajališta, a ne s dizajnerskog itd.[42]

[41] Usp. Isto.
[42] Usp. Guru 99. URL: https://www.guru99.com/grey-box-testing.html (2020-07-21)

3. Automatsko testiranje i primjena alata za vizualnu usporedbu mrežnog sjedišta tvrtke Five

3.1. Uvod u automatsko testiranje

Praktični dio ovog rada sastoji se od automatskih testova i primjene alata za vizualnu usporedbu mrežnog sjedišta tvrtke Five.[43] Osim vizualnog alata, koristio se i alat za upravljanje testnim slučajevima, alat koji je namijenjen za stvaranje izvješća testova i platforma za testiranje „u oblaku“ za mrežne i mobilne uređaje. Visual Studio Code je moćni uređivač izvornog kôda u kojem su se pisali testovi. Razvio ga je Microsoft, a dostupan je za Windows, macOS i Linux operacijske sustave.[44] Tvrtka Five nedavno je dobila sasvim novo mrežno sjedište te se ovim radom htjelo osigurati pravovremeno otkrivanje potencijalnih nedosljednosti i *bugova* s obzirom na dinamiku nadolazećih promjena spomenutog mrežnog sjedišta. U skladu s tim, okvirno, u radu su se postavile sljedeće hipoteze: korišteni alati u testiranju pridonose uspješnoj provedbi automatskih testova i kreirani testovi ispunjavaju kriterij stabilnosti njihove postojanosti u vremenu.

Nadalje, u ovom radu je pokazan i način kreiranja automatskih testova. Također se nastojalo pokazati i da se testiranje velikog broja mrežnih stranica može automatizirati i pri tome olakšati sam postupak testiranja. Osim toga, ovaj rad omogućuje i učenje postupka automatskog testiranja te je kreirane testove moguće prilagođavati i nadopunjavati. Korišteni alati pružaju veliki broj mogućnosti, a o svakome od njih bit će više riječi u nastavku rada. Osim toga, nakon što se alati opišu, prikazat će se što je sve potrebno instalirati za uspješno provođenje testova, odnosno primjenu alata te krajnji rezultat.

3.2. Korišteni alati

3.2.1. WebdriverIO

Jedan od korištenih alata, a ujedno i najvažniji alat za testiranje je WebdriverIO. Odabran je jer omogućuje integraciju s velikim brojem drugih alata. Ovaj se alat uz Visual Studio Code rabio za stvaranje testova. To je alat u otvorenom pristupu koji služi za automatsko testiranje i omogućuje pisanje jednostavnih testova u JavaScript programskom jeziku. Za izradu i pokretanje testova korišten je Node.js. Uz pomoć WebdriverIO testni kôd izgleda jednostavno,

[43] Usp. Five. URL: https://five.agency/ (2020-08-03)
[44] Usp. Visual Studio Code. URL: https://code.visualstudio.com/docs (2020-08-03)

sažeto i čitljivo. Također, rad s elementima na stranici je izrazito lagan. WebdriverIO dokumentacija je vrlo detaljno i jasno napisana i svatko koga zanima automatizacija može ju brzo savladati koristeći upute. Nadalje, WebdriverIO omogućuje pojednostavljivanje elemenata mrežnog sjedišta.[45] Elementi se dohvaćaju uz pomoć selektora na kojem se poziva metoda. Postoje različite vrste selektora, a oni koji su korišteni u ovom radu su ID, klasa, xPath, naziv HTML elementa i element s određenim tekstom. Najpovoljnije bi bilo da mrežno mjesto sadrži što više ID atributa HTML elementa, ali to ovdje nije bio slučaj i zbog toga se naišlo i na nekoliko izazova koji su na kraju uspješno riješeni.[46] Kako bi testovi bili pregledniji i jasniji korišten je *Page Object pattern*. *Page Object pattern* pruža sve moguće značajke od nasljeđivanja između *Page Objects*, učitavanja elemenata do enkapsulacije.[47] Također, on je postao popularan u automatskom testiranju zbog održavanja testova i smanjenja dupliciranja kôda. Prilikom primjene *Page Object pattern*, testovi koriste metode i oni se ne trebaju mijenjati, mijenja se samo kôd unutar *Page Object-a*. Vrlo je bitno odvojiti testove i *Page Objects* u zasebne datoteke. Zahvaljujući *Page Object patternu* svaka promjena u korisničkom sučelju može se lako implementirati, ažurirati i održavati.[48]

3.2.2. Percy

Percy je sveobuhvatna platforma koja služi za vizualnu usporedbu tj. vizualno testiranje mrežnog sadržaja. Percy pomaže prilikom automatizacije i testeru olakšava pronalaženje *bugova* vizualne prirode koji se javljaju na razini korisničkog sučelja. Percy upravlja sa svime, od prikazivanja i snimanja zaslona do obavještavanja testera o vizualnim promjenama. Bez obzira na to kako se provode vizualni testovi, Percy obrađuje i otkriva vizualne promjene. Također, on uspoređuje snimke zaslona s prethodno generiranim snimkama kako bi provjerio jesu li se promijenile veličine piksela na slikama. Snimke zaslona za pojedine komponente i stranice su grupirane i kao takve se stavljaju na pregled kao jedan build.[49]

3.2.3. TestRail

TestRail je alat za mrežno upravljanje testnim slučajevima. Koriste ga testeri, programeri i voditelji timova, a pruža pomoć u upravljanju, praćenju i organizaciji testiranog programa.

[45] Usp. Webdriver IO. URL: http://v4.webdriver.io/ (2020-08-03)
[46] Usp. Webdriver IO. URL: https://webdriver.io/docs/selectors.html (2020-08-03)
[47] Usp. Webdriver IO. URL: https://webdriver.io/docs/pageobjects.html (2020-08-03)
[48] Usp. Medium. URL: https://medium.com/tech-tajawal/page-object-model-pom-design-pattern-f9588630800b (2020-08-03)
[49] Usp. Percy. URL: https://docs.percy.io/docs/getting-started (2020-08-03)

TestRail omogućuje timovima pregled testnih slučajeva, njihovu organizaciju i izvršavanje, a sve to uz pomoć jednostavnog mrežnog sučelja. Također, on omogućuje stvaranje, upravljanje i organiziranje testnih slučajeva i paketa u okviru optimiziranog korisničkog sučelja i strukture aplikacija. TestRail pomaže koordinirati testove i povećava produktivnost i odgovornost testera. Cilj je osigurati da svaki član tima zna svoje zadatke u svakom trenutku i da voditelj tima može dodijeliti nove zadatke testerima, ovisno o njihovom opterećenju. Za donošenje važnih projektnih odluka ključno je imati pristup detaljnim informacijama o napretku testiranja i rezultatima ispitivanja. Također, bitna značajka TestRaila je ta što olakšava testiranje zbog integracije s alatima za praćenje grešaka (npr. Jira).[50]

3.2.4. Allure

Allure je *framework* koji je fleksibilan i koji pruža izvještaj o onome što je testirano u obliku mrežnog izvješća i omogućava svima koji sudjeluju u procesu razvoja programa da izvuku veliki broj korisnih informacija iz svakodnevnog izvođenja testova. Allure izvješća skraćuju životni ciklus grešaka i on pruža jasnu sliku o tome koje su značajke pokrivene, gdje se greške nalaze, kako izgleda vremenska crta izvješća itd. Prednost Allure-a je modularnost i proširivost te prilagodljivost ostalim alatima.[51]

3.2.5. BrowserStack

BrowserStack je platforma za testiranje mrežnih sjedišta i mobilnih aplikacija. Omogućava testiranje mrežnog sjedišta u više različitih preglednika ili mobilnih aplikacija na više vrsta mobilnih uređaja bez ikakvih virtualnih strojeva, uređaja i emulatora tj. on je zapravo tzv. *„device-lab“* u oblaku. BrowserStack je opcija kada se želi izbjeći složenost prebacivanja između operacijskih sustava i preglednika. Glavne značajke BrowserStack-a su testiranje putem više preglednika i na različitim operacijskim sustavima, testiranje hibridnih aplikacija, automatizacija mrežnog sjedišta i sl.[52]

3.3. Instalacija alata

Prije samog početka automatskog testiranja Five mrežnog sjedišta, bilo je potrebno postaviti programsku okolinu za uspješno provođenje testova. Testovi su kreirani i pokretani na

[50] Usp. TestRail. URL: https://www.gurock.com/testrail/docs/user-guide/getting-started/walkthrough (2020-08-03)
[51] Usp. Allure Framework. URL: https://docs.qameta.io/allure/ (2020-08-03)
[52] Usp. Software Testing Help. URL: https://www.softwaretestinghelp.com/browserstack-tutorial/ (2020-08-03)

Windows operacijskom sustavu. U nastavku će biti opisani koraci koje je bilo potrebno poduzeti kako bi se osigurali uvjeti za pisanje testova te koraci i uvjeti za instalaciju svih korištenih alata.

Preduvjeti koji trebaju biti ispunjeni za testiranje su posjedovanje jednog od mrežnih preglednika kao npr. Chrome, Node.js platforme[53] i alata u kojem će se pisati testovi (Visual Studio Code[54]). Nakon što su svi preduvjeti zadovoljeni, potrebno je pripremiti direktorij u koji će se pohraniti cijeli projekt. Direktorij se može napraviti uz pomoć Command Prompt-a (CMD) tako da se izvede naredba *mkdir* koja služi za kreiranje direktorija i naredba cd koja služi za pozicioniranje korisnika u novi, kreirani direktorij:

```
mkdir five-web
cd five-web
```

Nakon toga, potrebno je izvesti naredbu za pokretanje Node.js u projektu:

```
npm init -y
```

lovo -y označava da će svi uvjeti biti prihvaćeni i kreirat će se standardi npm projekt, ali -y se može i izostaviti ako korisnik želi sâm navesti pojedinosti o projektu.

Nakon toga, izvršava se naredba za instalaciju WebdriverIO i naredba za selenium-standalone koji se nalazi u *package.json* datoteci:

```
npm i webdriverio --save-dev
npm i selenium-standalone --save-dev
```

Zatim je potrebno instalirati WebdriverIO CLI (engl. *command-line interface*):

```
npm i @wdio/cli --save-dev
```

Nakon što su osnovni koraci napravljeni, potrebno je još odraditi konfiguraciju WebdriverIO-a uz pomoć node_modules:

```
node_modules\.bin\wdio config
```

Na zaslonu se pojavljuje pomoć za WebdriverIO konfiguraciju i postavljaju se pitanja koja unaprijed imaju određene vrijednosti kao odgovor. Preporuča se ostaviti sve onako kako je

[53] Node.js se može preuzeti ovdje: https://nodejs.org/en/download/ (2020-08-04)
[54] Visual Studio Code se može preuzeti ovdje: https://code.visualstudio.com/download (2020-08-04)

unaprijed zadano, a jedina stvar koja je u ovom slučaju promijenjena je osnovni url na url od Five mrežnog sjedišta.

Nakon toga se može kreirati prvi test koji će pokazati je li glavno okruženje ispravno instalirano. Prije nego li je prvi test napisan, unutar five-web direktorija kreirana su tri nova: prvi direktorij je test, a unutar testa su smještena još dva direktorija pod nazivom *specs* i *pageobjects*.

Test koji je provjeravao je li okruženje ispravno instalirano pohranjen je unutar test direktorija i imao je sljedeću sintaksu:

```
const assert = require('assert');

describe('Five web', () => {
    it('Should have the right title', () => {
        browser.url(' https://five.agency/);
    });
});
```

Nakon što je test napisan, potrebno je urediti *package.json* datoteku:

```
 "scripts": {

    "test": "wdio wdio.conf.js"
  },
```

Kada je *package.json* datoteka uređena, potrebno je otvoriti terminal unutar Visual Studio Code-a (ili Command prompt i pozicionirati se u root direktorij projekta) i izvršiti naredbu za pokretanje testova:

```
npm run test
```

Ako je test ispravan, stranica će se automatski pokrenuti i odraditi ono što je zadano u testu. Općenita struktura test datoteke je sljedeća:

```
describe("Test title", function() {

    beforeEach(function() {
      // optional
      // Your code goes here
    });
```

```
    it("Test case title", function() {
     // Your codes goes here
    });

    }
   );
```

Describe je funkcija u kojoj se upisuje naziv testa i unutar nje se nalaze ostali dijelovi test datoteke, a to su *beforeEach* i *afterEach* koji su neobavezni dio testa i *it* koji je obavezni dio. Unutar *beforeEach* funkcije mogu se nalaziti radnje koje se žele izvršiti prije pokretanja svakog testa (prije svakog *it* dijela) – primjerice učitavanje početne stranice. Unutar *afterEach* funkcije mogu se nalaziti radnje koje se žele izvršiti nakon pokretanja svakog testa (nakon svakog *it* dijela) – primjerice odjava ili uklanjanje kolačića ili predmemorije. Najbitniji dio je *it* dio koji označava samu test funkciju, a može ih biti jedna ili više. U tom dijelu dolazi do simulacije određenih akcija poput klikanja, *'scrollanja'*, upisivanja teksta u određena polja i sl. Test funkcija na kraju javlja je li test prošao.

Sintaksa kôda je sljedeća:

a) prikaz klika na gumb koji ima ID „button“:

```
$(selektor).akcija(); => $("#button").click();
```

b) otvaranje željenog URL-a:

```
browser.akcija(); => browser.url("https://five.agency/")
```

Nakon što je WedriverIO uspješno instaliran i prvi test uspješno dovršen, potrebno je integrirati ostale alate s WebdriverIO-om. Prvi integrirani alat je Percy. Kako bi se Percy ispravno instalirao, potrebno je pozicionirati se na five-web direktorij, otvoriti terminal i pokrenuti sljedeću naredbu:

```
npm install -save-dev @percy/webdriverio
```

Nakon instalacije Percy-a, u *wdio.conf.js* datoteku, na kraju, potrebno je dodati sljedeću funkciju koja se brine da je Percy uključen u testove na globalnoj razini:

```
before: function (capabilities, specs) {
    // Import percySnapshot function
    const { percySnapshot } = require('@percy/webdriverio');
    // Make percySnapshot available as a global variable in all wdio tests
    global.percySnapshot = percySnapshot;
  }
```

Nakon što je instalacija završena, potrebno je kreirati račun na Percy[55] stranici, a nakon računa potrebno je kreirati novu organizaciju i projekt. Nakon što se kreira projekt, u postavkama je potrebno preuzeti Percy token koji je potreban za komunikaciju između WebdriverIO-a i Percy-a te token pokrenuti lokalno kao naredbu i dodati novu skriptu u *package.json* datoteku:

```
set PERCY_TOKEN=<your token here>
```

```
"percy-
test": "set PERCY_TOKEN={yourTokenHere} && set PERCY_BRANCH=local && percy exe
c -- wdio wdio.conf.js"
```

Nakon toga, potrebno je u datoteku u kojoj se nalaze testovi, ispod svakog testa dodati *snapshot* funkciju:

```
browser.call(
    async () => await percySnapshot(browser, "unique test case name", { widths
: [1280] })
  );
```

Percy test pokreće se koristeći naredbu:

```
npm run percy-test
```

Nakon što se naredba izvrši, potrebno je pričekati build na Percy stranici u kojem će se nalaziti snimke zaslona. Nadalje, Percy kao alat nudi mogućnost integracije s drugim alatima te je u ovom slučaju napravljena integracija sa Slack-om[56]. Prednost ove integracije je ta što svaki put kada je pokrenut Percy test, na Slack bi došla obavijest kada je test završen, koji je to build po redu i koliko je snimki zaslona snimljeno. Prilikom klika na poveznicu na Slack-u, otvorio bi se trenutni build i sve što on sadrži. Osim toga, prednost je što je cijeli tim pravovremeno informiran da su testovi prošli, da sadrže vizualna odstupanja ili ne i sl.

Nakon Percy-a, potrebno je integrirati TestRail. U Visual Studio Code-u potrebno se pozicionirati na direktorij five-web, otvoriti terminal i pokrenuti sljedeću naredbu:

[55] Račun je moguće kreirati ovdje: https://percy.io/ (2020-08-04)
[56] Slack se može preuzeti ovdje: https://slack.com/intl/en-hr/ (2020-08-04)

```
npm install wdio-wdiov5testrail-reporter --save-dev
```

Nakon toga, potrebno je napraviti račun na TestRail[57] stranici, a u *wdio.conf.js* potrebno je dodati sljedeće informacije koje se dobiju prilikom registracije, a to su domena, korisničko ime, lozinka, ProjectId i suiteId, a runName je proizvoljan naziv automatskih testova.

```
let WdioTestRailReporter = require('wdio-5-testrail-reporter');
    reporters: [[WdioTestRailReporter, {
    domain: "yourdomain.testrail.net",
        username: "your username",
    password: "your password",
    projectId: 1,
    suiteId: 1,
    runName: "My test run"
  }]]
```

Na kraju je u *package.json* potrebno dodati sljedeći tekst:

```
"test-testrail": "wdio wdio-testrail.conf.js"
```

TestRail pokreće se naredbom:

```
npm run test-testrail
```

Sljedeći alat koji je potrebno integrirati je Allure. Kako bi se Allure uspješno instalirao, u terminalu je potrebno pokrenuti sljedeću naredbu:

```
npm install @wdio/allure-reporter --save-dev
```

U *package.json* datoteku potrebno je dodati sljedeću funkciju pomoću koje će se pokretati Allure izvještaj:

```
"generate-report": "allure generate allure-results --clean && allure open"
```

[57] Račun je moguće kreirati ovdje: https://www.gurock.com/testrail/?utm_source=adwords&utm_medium=cpc&utm_campaign=europe_en_brand&utm_content=testrail&creative=239490653021&keyword=testrail&matchtype=e&network=g&device=c&gclid=CjwKCAjwsan5BRAOEiwALzomXwM_xzuS_nfhh4K_-W2qNQLxfXPRxBxKv0_ymM0YGppX-rsKwumL4hoCvRcQAvD_BwE (2020-08-04)

Zatim, u svaku test datoteku potrebno je *importati* Allure, kao i dodati Allure funkciju ispod svakog testa:

```
import allureReporter from '@wdio/allure-reporter'

allureReporter.addFeature('your test name')
```

Na samom kraju potrebno je napraviti konfiguraciju Allure-a, a sljedeći kôd potrebno je ubaciti u *wdio.conf.js* datoteku:

```
exports.config = {
    // ...
    reporters: ['spec',['allure', {
        outputDir: 'allure-results',
    }]],
    // ...
}
```

Kako bi dobili Allure izvještaj, potrebno je pokrenuti sljedeću naredbu u terminalu:

```
npm run generate-report
```

Zadnji alat koji je integriran s WebdriverIO je BrowserStack. BrowserStack se instalira tako što se u terminalu pokrene naredba:

```
npm install @wdio/browserstack-service --save-dev
```

Osim toga, potrebno je napraviti i BrowserStack konfiguraciju i u *wdio.conf.js* datoteku ubaciti sljedeći kôd (username i key se izgeneriraju prilikom kreiranja računa na BrowserStack-u)[58]:

```
export.config = {
    // ...
    user: process.env.BROWSERSTACK_USERNAME,
    key: process.env.BROWSERSTACK_ACCESS_KEY,
    services: [
        ['browserstack', {
            browserstackLocal: true
        }]
    ],
```

[58] Račun je moguće kreirati ovdje: https://www.browserstack.com/accounts/settings (2020-08-14)

```
    // ...
};
```

Zatim je u *package.json* potrebno upisati sljedeće:

```
"test-browserstack": "wdio wdio-browserstack.conf.js"
```

BrowserStack pokreće se naredbom:

```
npm run test-browserstack
```

3.4. Automatski testovi

Nakon što je okruženje uspješno postavljeno, alati instalirani i integrirani, potrebno je napisati testove i dobiti rezultate istih. Automatski testovi koji su rađeni na Five mrežnom sjedištu podijeljeni su u tri datoteke. U spomenutom direktoriju *specs* nalaze se datoteke pod sljedećim nazivima *home.spec.js, products.spec.js i socialnetwork.spec.js*, a u *pageobjects* direktoriju nalaze se datoteke koje odgovaraju datotekama u direktoriju *specs*, a to su *home.page.js, products.page.js, socialnetwork.page.js i page.js*. U *specs* datotekama su testovi, a u *pageobjects* datotekama su selektori. Testovi i selektori odvojeni su kako ne bi došlo do promjene selektora na mrežnom sjedištu; potrebno je samo izmijeniti selektore koji se nalaze u datotekama unutar *pageobjects* direktorija i to se automatski primjenjuje na sve testove. Testovi su na ovaj način podijeljeni kako bi se u datoteci *home.spec.js* nalazili svi testovi vezani uz naslovnu stranicu i ono što se nalazi u njoj. U datoteci *products.spec.js* nalaze se testovi vezani uz uspješne Five-ove projekte, a u datoteci *socialnetwork.spec.js* nalazi se test koji vodi na društvenu mrežu.

Prvo će se opisati testovi koji se nalaze u *home.spec.js* datoteci i odgovarajući *Page* objekti. Na samom početku, u svakoj test datoteci potrebno je napraviti *import* Allure-a, *page* objekta i *assert* paketa:

```
import assert from "assert";
import allureReporter from '@wdio/allure-reporter';
import Home from "../pageobjects/home.page"
```

Describe dio prve testne datoteke izgleda ovako:

```
describe("Five web", function()  {
```

```
    beforeEach(function(){
        Home.open();
        Home.homeTitle.waitForDisplayed();
    });
```

Prvi test je najjednostavniji, a pomoću njega otvara se naslovna stranica tvrtke Five.

```
 it("C1 Should open homepage", function()  {

        allureReporter.addFeature('Should open homepage');

        Home.scrollFooter.scrollIntoView();

        browser.call(
            async () => await percySnapshot(browser, "Five Homepage", { widths
: [1280] })
          );
          Home.homeTitle.waitForDisplayed();

    });
```

Svaki test je unaprijed dodan u TestRail, a TestRail i napisani testovi povezani su uz pomoć ID atributa HTML elementa koji se nalazi prije samog imena testa (u ovom slučaju C1). *AllureReporter* funkcija spaja testove s Allure paketom koji generira HTML rezultate, a *scrollFooter* služi tomu da se prilikom ulaska na stranicu dospije do kraja stranice. Uz pomoć *browser.call* funkcije dobije se snimka zaslona, a *waitForDisplayed* potvrđuje da se učitao očekivani element,.

Cilj drugog testa je otvoriti „About“ poveznicu Five mrežnog sjedišta. U drugom testu potrebno je dodati *allureReporter*, kliknuti na *About* poveznicu, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C2 Should open About section", function()  {
        allureReporter.addFeature('Should open About section');

        Home.aboutSection.click();
        Home.homeTitle.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();

        browser.call(
            async () => await percySnapshot(browser, "Five 'About' Page", { wi
dths: [1280] })
          );

    });
```

Cilj trećeg testa je otvoriti „Services“ poveznicu Five mrežnog sjedišta. U trećem testu potrebno je dodati *allureReporter*, kliknuti na *Services* poveznicu, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C3 Should open Services section", function()  {
        allureReporter.addFeature('Should open Services section');

        Home.servicesSection.click();
        Home.homeTitle.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();

        browser.call(
            async () => await percySnapshot(browser, "Five 'Services' Page", {
 widths: [1280] })
          );

    });
```

Cilj četvrtog testa je otvoriti „Work“ poveznicu Five mrežnog sjedišta. U četvrtom testu potrebno je dodati *allureReporter*, kliknuti na *Work* poveznicu, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C4 Should open Work section", function()  {
        allureReporter.addFeature('Should open Work section');

        Home.workSection.click();
        Home.homeTitle.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();

        browser.call(
             async () => await percySnapshot(browser, "Five 'Our Work' Page",
{ widths: [1280] })
            );
     });
```

Cilj petog testa je otvoriti „Careers“ poveznicu Five mrežnog sjedišta. U petom testu potrebno je dodati *allureReporter*, kliknuti na *Careers* poveznicu, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C5 Should open Careers section", function()  {
        allureReporter.addFeature('Should open Careers section');

        Home.careersSection.click();
        Home.homeTitle.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();
```

```
        browser.call(
             async () => await percySnapshot(browser, "Five 'Careers' Page", {
 widths: [1280] })
            );
    });
```

Cilj šestog testa je otvoriti „Contact“ poveznicu Five mrežnog sjedišta. U šestom testu potrebno je dodati *allureReporter*, kliknuti na *Contact* poveznicu, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
 it("C6 Should open Contact section", function()  {
        allureReporter.addFeature('Should open Contact section');

        Home.contactSection.click();
        Home.homeTitle.waitForDisplayed();

        browser.call(
             async () => await percySnapshot(browser, "Five 'Contact' Page", {
 widths: [1280] })
            );
    });
```

Cilj sedmog testa je otvoriti „Blog“ poveznicu Five mrežnog sjedišta. U sedmom testu potrebno je dodati *allureReporter*, kliknuti na *Blog* poveznicu, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
 it("C7 Should open Blog section", function()  {
        allureReporter.addFeature('Should open Blog section');

        Home.blogSection.click();
        Home.homeTitle.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();
        browser.pause(3000);

        browser.call(
             async () => await percySnapshot(browser, "Five 'Blog' Page", { wi
dths: [1280] })
            );
    });
```

Cilj osmog testa je otvoriti „Blog“ poveznicu i „Android“ poveznicu unutar njega na Five mrežnom sjedištu. U osmom testu potrebno je dodati *allureReporter*, kliknuti na *Blog* poveznicu, kliknuti na Android poveznicu unutar *Bloga*, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C8 Should open Blog Android section", function()  {
        allureReporter.addFeature('Should open Blog Android sectio');

        Home.blogSection.click();
        Home.androidSection.click();
        Home.articleName.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();

        browser.call(
             async () => await percySnapshot(browser, "Five 'Blog-
Android' Page", { widths: [1280] })
            );
    });
```

Cilj devetog testa je otvoriti „Blog“ poveznicu i „Business“ poveznicu unutar njega na Five mrežnom sjedištu. U devetom testu potrebno je dodati *allureReporter*, kliknuti na *Blog* poveznicu, kliknuti na *Business* poveznicu unutar *Bloga*, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C9 Should open Blog Business section", function()  {
        allureReporter.addFeature('Should open Blog Business section');

        Home.blogSection.click();
        Home.businessSection.click();
        Home.articleName.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();

        browser.call(
             async () => await percySnapshot(browser, "Five 'Blog-
Business' Page", { widths: [1280] })
            );
    });
```

Cilj desetog testa je otvoriti „Blog“ poveznicu i „Design“ poveznicu unutar njega na Five mrežnom sjedištu. U desetom testu potrebno je dodati *allureReporter*, kliknuti na *Blog* poveznicu, kliknuti na *Design* poveznicu unutar *Bloga*, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C10 Should open Blog Design section", function()  {
        allureReporter.addFeature('Should open Blog Design section');

        Home.blogSection.click();
        Home.designSection.click();
        Home.articleName.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();
```

```
        browser.call(
             async () => await percySnapshot(browser, "Five 'Blog-
Design' Page", { widths: [1280] })
            );
    });
```

Cilj jedanaestog testa je otvoriti „Blog“ poveznicu i „Development“ poveznicu unutar njega na Five mrežnom sjedištu. U jedanaestom testu potrebno je dodati *allureReporter*, kliknuti na *Blog* poveznicu, kliknuti na *Development* poveznicu unutar *Bloga*, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
 it("C11 Should open Blog Development section", function()  {
        allureReporter.addFeature('Should open Blog Development section');

        Home.blogSection.click();
        Home.developmentSection.click();
        Home.articleName.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();

        browser.call(
             async () => await percySnapshot(browser, "Five 'Blog-
Deveopment' Page", { widths: [1280] })
            );
    });
```

Cilj dvanaestog testa je otvoriti „Blog“ poveznicu, zatim „Development“ poveznicu i članak unutar „Developmenta“ na Five mrežnom sjedištu. U dvanaestom testu potrebno je dodati *allureReporter*, kliknuti na *Blog* poveznicu, kliknuti na *Development* poveznicu unutar *Bloga* te kliknuti na članak unutar *Development* poveznice, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
 it("C12 Should open Blog Development article", function()  {
        allureReporter.addFeature('Should open Blog Development article');

        Home.blogSection.click();
        Home.developmentSection.click();
        Home.articleName.click();
        Home.scrollFooter.scrollIntoView();
        browser.pause(2000);

        browser.call(
             async () => await percySnapshot(browser, "Five 'Blog-
Development Article' Page", { widths: [1280] })
```

```
            );
    });
```

Cilj trinaestog testa je otvoriti „Blog“ poveznicu i „iOS“ poveznicu unutar njega na Five mrežnom sjedištu. U trinaestom testu potrebno je dodati *allureReporter*, kliknuti na *Blog* poveznicu, kliknuti na iOS poveznicu unutar *Bloga*, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
 it("C13 Should open Blog IOS section", function()  {
        allureReporter.addFeature('Should open Blog IOS section');

        Home.blogSection.click();
        Home.iosSection.click();
        Home.articleName.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();
        browser.pause(2000);

        browser.call(
             async () => await percySnapshot(browser, "Five 'Blog-
IOS' Page", { widths: [1280] })
            );
    });
```

Cilj četrnaestog testa je otvoriti „Blog“ poveznicu i „Java“ poveznicu unutar njega na Five mrežnom sjedištu. U četrnaestom testu potrebno je dodati *allureReporter*, kliknuti na *Blog* poveznicu, kliknuti na Java poveznicu unutar *Bloga*, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C14 Should open Blog Java section", function()  {
        allureReporter.addFeature('Should open Blog Java section');

        Home.blogSection.click();
        Home.javaSection.click();
        Home.articleName.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();

        browser.call(
             async () => await percySnapshot(browser, "Five 'Blog-
Java' Page", { widths: [1280] })
            );
    });
```

Cilj petnaestog testa je otvoriti „Legal“ poveznicu Five mrežnog sjedišta. U osmom testu potrebno je dodati *allureReporter*, kliknuti na *Legal* poveznicu, dospjeti do kraja stranice da se učitaju svi elementi i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C15 Should open Legal section", function()  {
        allureReporter.addFeature('Should open Legal section');

        Home.legalSection.click();
        Home.homeTitle.waitForDisplayed();
        Home.scrollFooter.scrollIntoView();

        browser.call(
             async () => await percySnapshot(browser, "Five 'Legal' Page", { w
idths: [1280] })
            );
    });
```

Nakon što je svih petnaest testova iz *home.spec.js* datoteke prikazano, prikazat će se i kako izgleda struktura *home.page.js* i unaprijed zadanog *page.js*. *Page* objekti koji se nalaze u *home.page.js* datoteci uvelike su pomogli u pisanju testova, zbog toga što su testovi zbog njih pregledniji, te je olakšano snalaženje testera u kôdu.

Struktura *page.js* datoteke izgleda ovako:

```
export default class Page {
    open(path) {
      browser.url(path);
    }
  }
```

Struktura *home.page.js* datoteke izgleda ovako:

```
import Page from "./page";

class Home extends Page {
  open() {
    super.open("/");
  }
get homeTitle(){
    return $("h1");
}
get scrollFooter(){
    return $("footer");
}
get aboutSection(){
    return $("span=About");
}
```

```
get servicesSection(){
    return $("span=Services");
}
get workSection(){
    return $("span=Work");
}
get careersSection(){
    return $("span=Careers");
}
get contactSection(){
    return $("span=Contact");
}
get blogSection(){
    return $("span=Blog");
}
get legalSection(){
    return $("span=Legal");
}
get androidSection(){
    return $(".cat-item.cat-item-20");
}
get articleName(){
    return $("/html/body/div[1]/div[1]/div/div/div[2]/div[1]/div/div[2]/div[1]
/div/h2/a");

}
get businessSection(){
    return $(".cat-item.cat-item-12");
}
get designSection(){
    return $(".cat-item.cat-item-36");
}
get developmentSection(){
    return $(".cat-item.cat-item-13");
}
get iosSection(){
    return $(".cat-item.cat-item-24");
}
get javaSection(){
    return $(".cat-item.cat-item-37");
}}
export default new Home();
```

Kao što je vidljivo u kôdu iznad, u sklopu *Page* objekata korišteni su različiti selektori. Na nekim mjestima su korištene klase (npr. $(".cat-item.cat-item-12");), na nekim mjestima HTML elementi (npr. $("h1");), na nekim mjestima elementi s tekstom (npr.

$("span=About");), dok na nekim mjestima xPath (npr. $("/html/body/div[1]/div[1]/div/div/div[2]/div[1]/div/div[2]/div[1]/div/h2/a");) i sl.

Nakon što su opisani testovi koji se nalaze u *home.spec.js* datoteci, opisat će se testovi koji se nalaze u *products.spec.js* datoteci. Kao i u prvoj datoteci i ovdje su iste stvari na samom početku importirane u datoteku:

```
import assert from "assert";
import allureReporter from '@wdio/allure-reporter';
import Products from "../pageobjects/products.page"
```

Describe dio druge datoteke izgleda slično kao i u prvoj datoteci:

```
describe("Five web", function()  {

    beforeEach(function(){
        Products.open();
        Products.homeTitle.waitForDisplayed();
    });
```

U ovoj datoteci nalaze se testovi od kojih svaki otvara Five mrežno sjedište na kojem se nalaze opisi njihovih uspješnih projekata. Na mrežnom sjedištu, trenutno je predstavljeno njih šest.

Cilj šesnaestog testa je otvoriti „Work“ poveznicu i „Marriott Careers“ projekt na Five mrežnom sjedištu. U šesnaestom testu potrebno je dodati *allureReporter*, kliknuti na *Work* poveznicu, kliknuti na *Marriott Careers* projekt i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C16 Should open Work-Marriott Careers case study", function()  {
    allureReporter.addFeature('Should open Work-Marriott Careers case study');

    Products.workSection.click();
    Products.marriottCareers.click();
    Products.productTitle.waitForDisplayed();

    browser.call(
         async () => await percySnapshot(browser, "Five 'Mariott Careers' Page
", { widths: [1280] })
        );
 });
```

Cilj sedamnaestog testa je otvoriti „Work“ poveznicu i „Rosetta Stone“ projekt na Five mrežnom sjedištu. U sedamnaestom testu potrebno je dodati *allureReporter*, kliknuti na *Work* poveznicu, kliknuti na *Rosetta Stone* projekt i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C17 Should open Work-Rosetta Stone case study", function()  {

    allureReporter.addFeature('Should open Work-Rosetta Stone case study');

    Products.workSection.click();
    Products.rosettaStone.click();
    Products.productTitle.waitForDisplayed();

    browser.call(
         async () => await percySnapshot(browser, "Five 'Roseta Stone' Page",
{ widths: [1280] })
        );
 });
```

Cilj osamnaestog testa je otvoriti „Work“ poveznicu i „Moonlite“ projekt na Five mrežnom sjedištu. U osamnaestom testu potrebno je dodati *allureReporter*, kliknuti na *Work* poveznicu, kliknuti na *Moonlite* projekt i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C18 Should open Work-Moonlite case study", function()  {
    allureReporter.addFeature('Should open Work-Moonlite case study');

    Products.workSection.click();
    Products.moonlite.click();
    Products.productTitle.waitForDisplayed();

    browser.call(
         async () => await percySnapshot(browser, "Five 'Moonlite' Page", { wi
dths: [1280] })
        );
 });
```

Cilj devetnaestog testa je otvoriti „Work“ poveznicu i „Napster“ projekt na Five mrežnom sjedištu. U devetnaestom testu potrebno je dodati *allureReporter*, kliknuti na *Work* poveznicu, kliknuti na *Napster* projekt i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C19 Should open Work-Napster case study", function()  {
    allureReporter.addFeature('Should open Work-Napster case study');

    Products.workSection.click();
    Products.napster.click();
    Products.productTitle.waitForDisplayed();

    browser.call(
        async () => await percySnapshot(browser, "Five 'Napster' Page", { widt
hs: [1280] })
        );
```

```
});
```

Cilj dvadesetog testa je otvoriti „Work“ poveznicu i „Nanit“ projekt na Five mrežnom sjedištu. U dvadesetom testu potrebno je dodati *allureReporter*, kliknuti na *Work* poveznicu, kliknuti na *Nanit* projekt i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C20 Should open Work-Nanit case study", function()  {
    allureReporter.addFeature('Should open Work-Nanit case study');

    Products.workSection.click();
    Products.nanit.click();
    Products.productTitle.waitForDisplayed();

    browser.call(
         async () => await percySnapshot(browser, "Five 'Nanit' Page", { width
s: [1280] })
        );
 });
```

Cilj dvadeset prvog testa je otvoriti „Work“ poveznicu i „Game of Thrones“ projekt na Five mrežnom sjedištu. U dvadeset prvom testu potrebno je dodati *allureReporter*, kliknuti na *Work* poveznicu, kliknuti na *Game of Thrones* projekt unutar njega i na kraju uz pomoć *browser.call-a* snimiti zaslon.

```
it("C21 Should open Work-Game of Thrones case study", function()  {
    allureReporter.addFeature('Should open Work-Game of Thrones case study');

    Products.workSection.click();
    Products.gameOfThrones.click();
    Products.productTitle.waitForDisplayed();

    browser.call(
         async () => await percySnapshot(browser, "Five 'Game of Thrones' Page
", { widths: [1280] })
        );
 });
```

Nakon što je svih šest testova iz *products.spec.js* direktorija prikazano, struktura *products.page.js* direktorija.

Struktura *products.page.js* direktorija izgleda ovako:

```
import Page from "./page";
```

```
class Products extends Page {
  open() {
    super.open("/");
  }
get homeTitle(){
    return $("h1");
}
get productTitle(){
    return $("h3");
}
get workSection(){
    return $("span=Work");
}
get marriottCareers(){
    return $("//*[@id='case-studies-showcase-
block_5e43f702c3fda']/div/div/div/div[1]/div/a[2]");
}
get rosettaStone(){
    return $("//*[@id='case-studies-showcase-
block_5e43f702c3fda']/div/div/div/div[2]/div/a[2]");
}
get moonlite(){
    return $("//*[@id='case-studies-showcase-
block_5e43f702c3fda']/div/div/div/div[5]/div/a[2]");
}
get napster(){
    return $("//*[@id='case-studies-showcase-
block_5e43f702c3fda']/div/div/div/div[6]/div/a[2]");
}
get nanit(){
    return $("//*[@id='case-studies-showcase-
block_5e43f702c3fda']/div/div/div/div[9]/div/a[2]");
}
get gameOfThrones(){
    return $("//*[@id='case-studies-showcase-
block_5e43f702c3fda']/div/div/div/div[10]/div/a[2]");
}}
export default new Products();
```

Kao što je vidljivo u prikazanom kôdu iznad, od selektora u *products.page.js* direktoriju korišteni su HTML elementi i u nešto većoj mjeri xPath-ovi.

Nakon što su opisani i testovi koji se nalaze u product.spec.js datoteci, opisat će se testovi koji se nalaze u *socialnetwork.spec.js* datoteci. Kao i u prve dvije datoteke i ovdje su određene stvari importirane u datoteku:

```
import chai from "chai";
import allureReporter from "@wdio/allure-reporter";
import Social from "../pageobjects/socialnetwork.page";
```

Describe dio posljednje, treće datoteke izgleda slično kao i kod prve dvije:

```
describe("Five web", function () {
  beforeEach(function () {
    Social.open();
  });
```

U ovoj datoteci nalazi se test koji otvara društvenu mrežu Facebook Five mrežnog sjedišta. Cilj posljednjeg, dvadeset drugog testa je otvoriti Five mrežno sjedište, dospjeti do dna stranice i s desne strane kliknuti na ikonu od društvene mreže Facebook te istu otvoriti u novom prozoru.

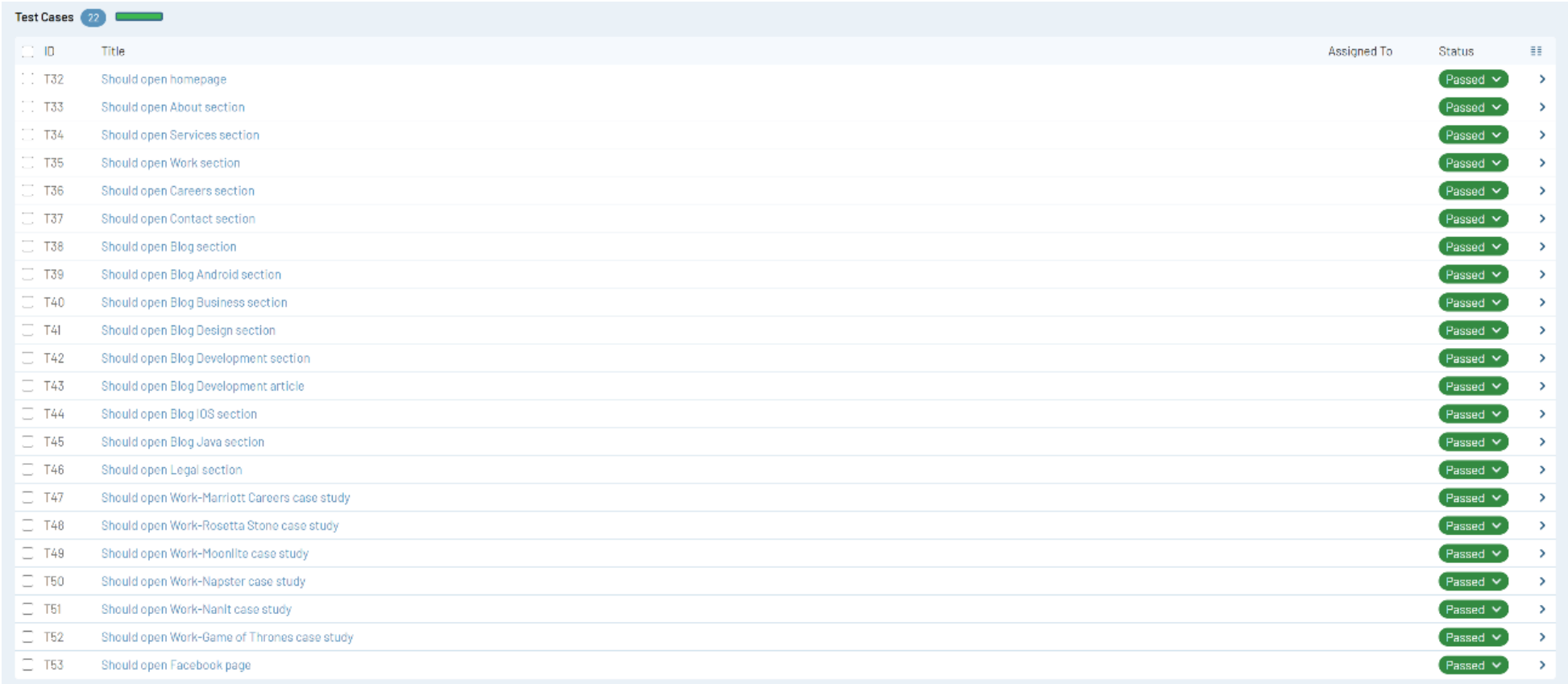
```
 it("C22 Should open Facebook page", function () {
    allureReporter.addFeature("Should open Facebook page");

    Social.facebook.click();
    browser.switchWindow("FIVE - Home | Facebook");

    browser.call(
      async () =>
        await percySnapshot(browser, "Five Facebook Page", { widths: [1280] })
    );
  });
```

Nakon što je prikazan posljednji, dvadeset drugi test iz *socialnetwork.spec.js* datoteke, struktura *socialnetwork.page.js* datoteke izgleda ovako:

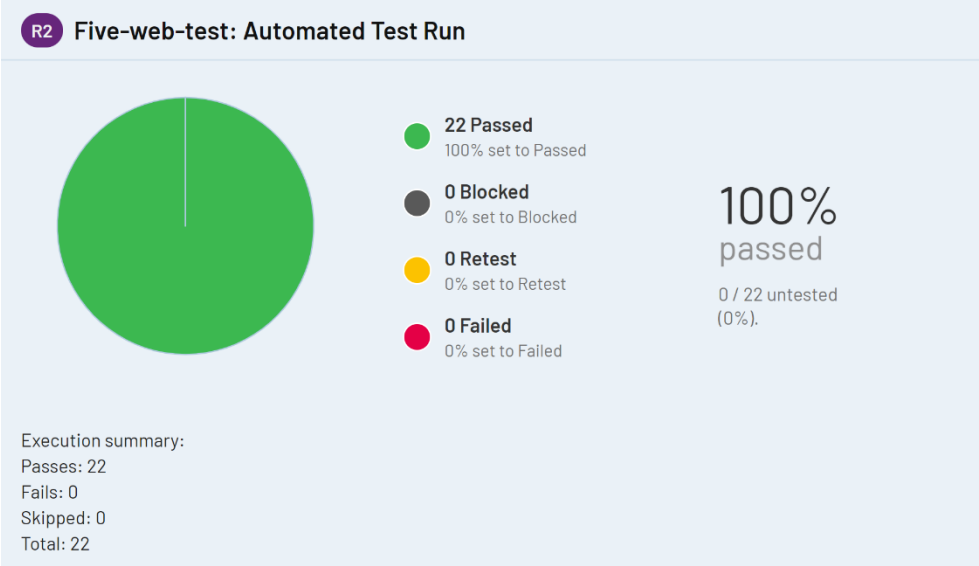
```
import Page from "./page";

class Social extends Page {
  open() {
    super.open("/");
}
get homeTitle(){
    return $("h1");
}
get facebook(){
  return $(".icon.icon-facebook");
}}
```

```
export default new Social();
```

TestRail nudi mogućnost prikaza svih testnih slučajeva na jednom mjestu i Slika 3. prikazuje kako izgledaju svi testovi na jednom mjestu, a Slika 4. prikazuje kako izgleda dijagram koji pokazuje broj uspješno provedenih testova, broj neuspješno provedenih testova, koliko je testova bilo blokirano i koje testove treba ponoviti. Svaka od varijabli naznačena je različitom bojom što dodatno olakšava uvid u stanje pojedinog testa. Također, uz dijagram se nalazi i postotak koji prikazuje rezultate prolaza testnih slučajeva.

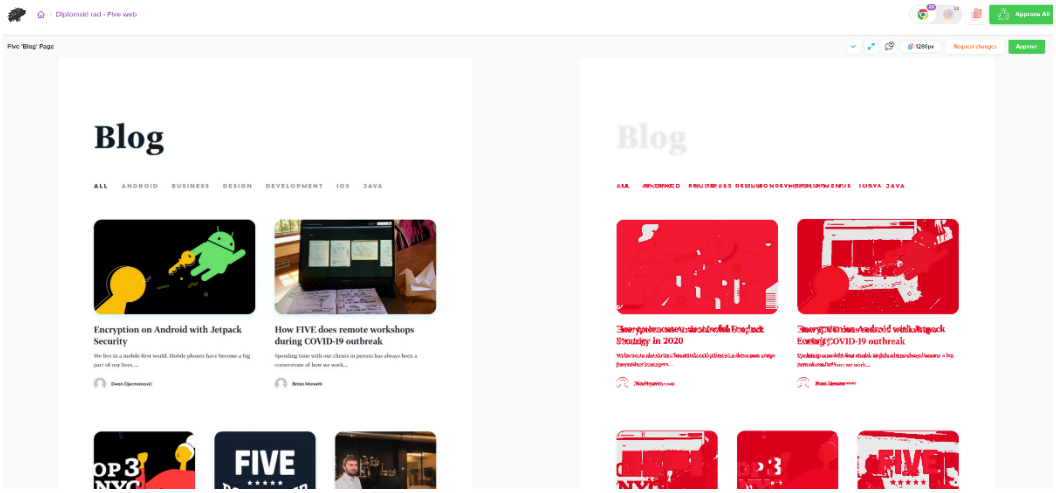


Slika 3. Prikaz svih testnih slučajeva na jednom mjestu.

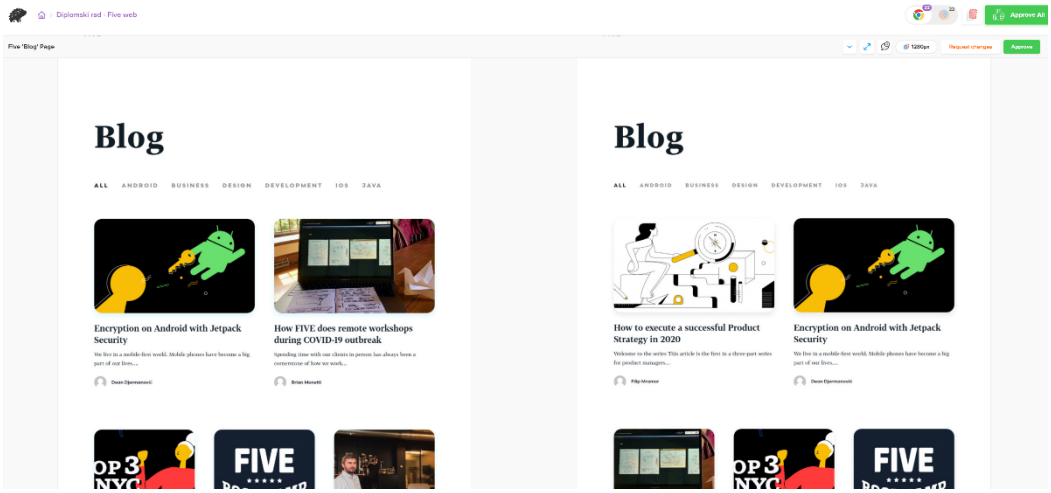


Slika 4. Prikaz dijagrama testnih slučajeva.

Što se tiče Percy-a, kao što je već rečeno, nakon što se ovaj test pokrene više puta i kreiraju nove snimke zaslona, te se nove snimke uspoređuju s postojećima (Slika 5.). Na lijevoj strani je *„baseline“* slika odnosno posljednja verificirana snimka zaslona, a na desnoj strani je nova snimka zaslona prekrivena vizualnom razlikom – izmijenjeni pikseli označeni su crvenom bojom. U ovom slučaju došlo je do objave novog članka na Five blogu. Kada se klikne na desnu snimku zaslona, crvena boja nestane te je na Slici 6. vidljivo kako je dodan novi članak čime je promijenjen raspored objava na stranici na što Percy upozorava. Ako su nastale očekivane promjene, snimka zaslona se prihvaća i ona postaje novi *„baseline“* s kojim će se nadolazeće snimke zaslona uspoređivati. Ako promjena nije očekivana, nego se radi o *bugu*, takva snimka zaslona se ne prihvaća te se prijavljuje *bug*. Ovaj alat je vrlo koristan jer vrši 'piksel po piksel' usporedbu i otkriva najsitnija odstupanja poput, primjerice, promjena u razmaku između elemenata, a što golim okom nije vidljivo. Upravo zbog toga je idealan da upotpuni klasični test *framework* koji samo vrši akcije nad elementima ne znajući je li došlo do stilskih promjena među njima.

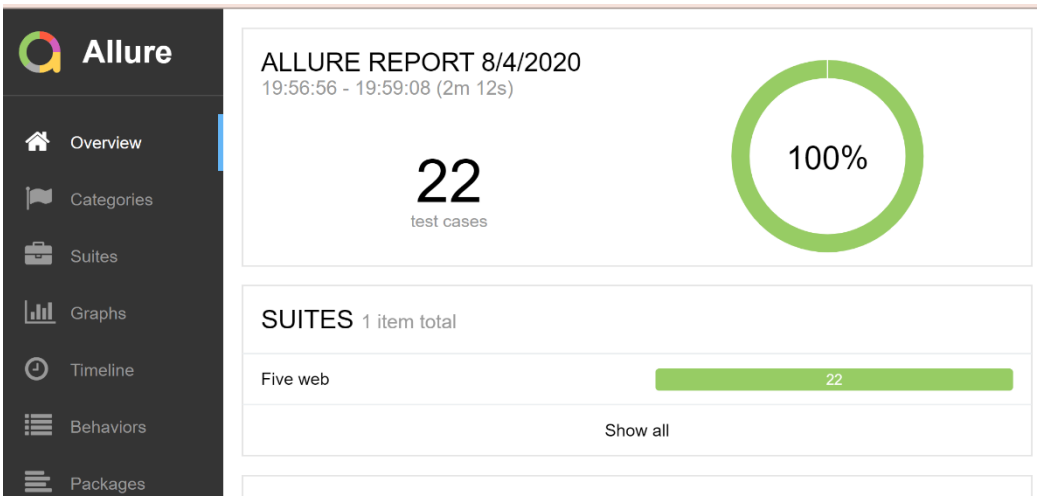


Slika 5. Vizualna usporedba dviju snimki zaslona.

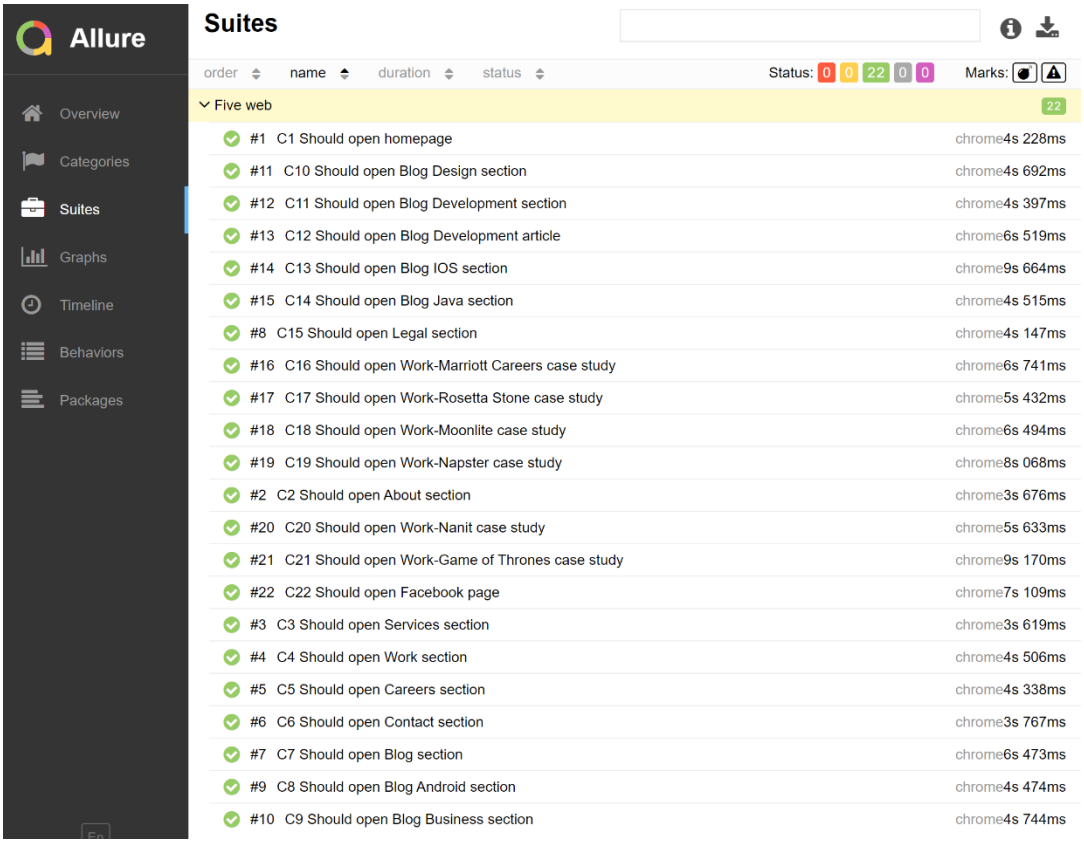


Slika 6. Vizualna usporedba dviju snimki zaslona bez crveno označenih piksela.

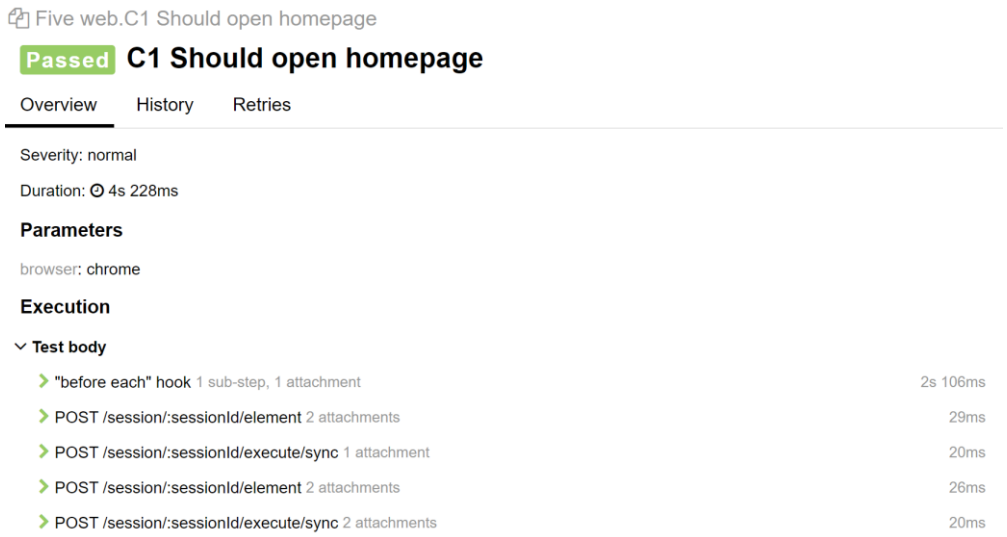
Što se tiče Allure-a, na Slici 7. prikazana je nadzorna ploča na kojoj je prikazana statistika o svim testovima koji su pokrenuti na terminalu, a na Slici 8. nalazi se standardni strukturni prikaz izvršenih testova koji su podijeljeni u određene skupine, dok se na Slici 9. nalazi izvještaj jednog od testova. Allure je idealan jer generira rezultate u HTML-u i ti rezultati se također mogu isporučiti na Slack ili bilo koji drugi komunikacijski kanal nakon izvršenih testova. Osim što jasno prikazuju jesu li testovi uspješno izvršeni ili ne, Allure i točno prikazuje sve korake i akcije te ispisuje *„error trace“* u slučaju da test nije uspješno proveden, a to uvelike smanjuje vrijeme koje programeri troše ne bi li pronašli uzrok greške.



Slika 7. Prikaz Allure nadzorne ploče.

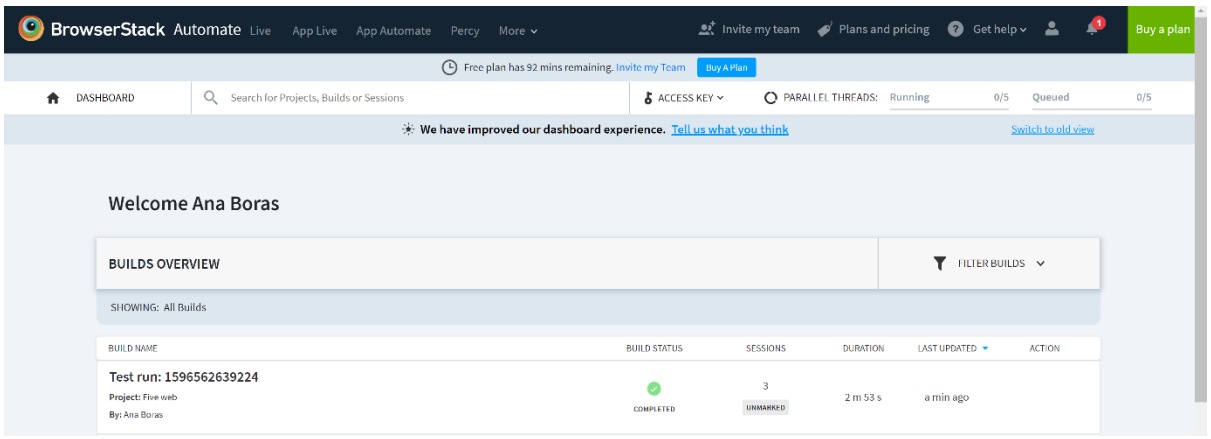


Slika 8. Strukturirani prikaz izvršenih testova u Allure-u.

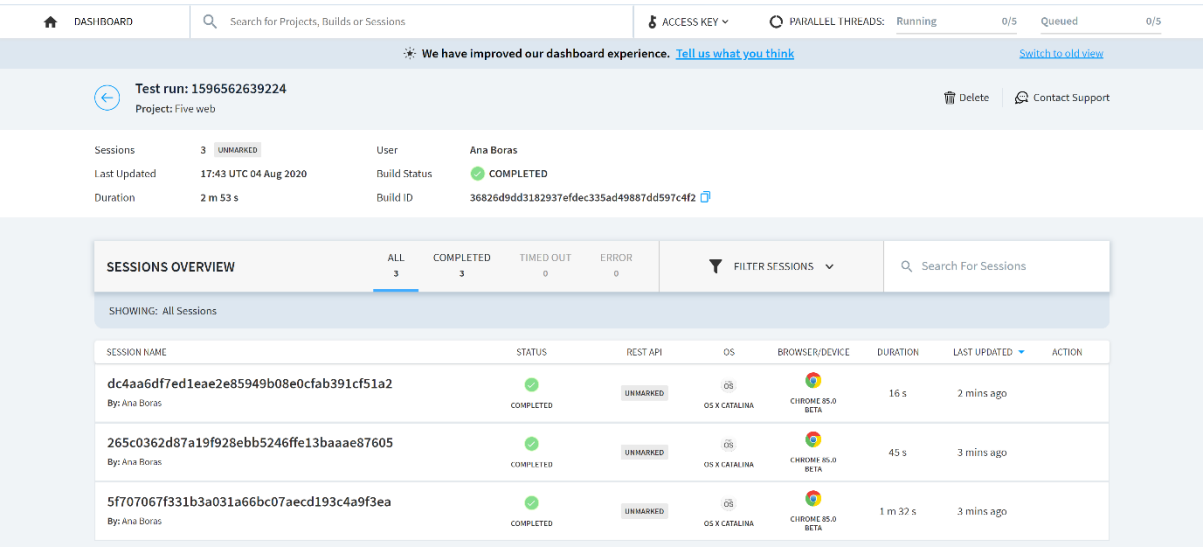


Slika 9. Prikaz izvršenog testa u Allure-u.

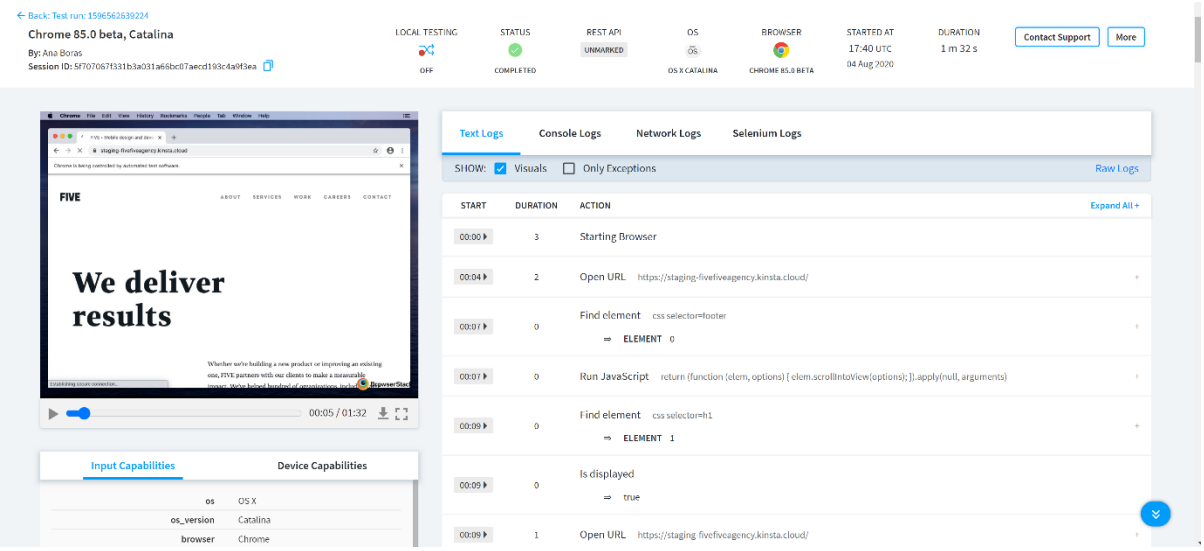
Na sljedećim slikama prikazano je kako izgleda mrežno sjedište BrowserStack-a. Na Slici 10. prikazan je pregled builda, njegovo stanje, od koliko se sesija sastoji, koliko je vremena bilo potrebno da se build završi i sl. Na Slici 11. prikazano je što se sve nalazi unutar jednog builda – prikazane su sve tri izvršene sesije u kojima se nalaze testovi te za svaku sesiju vrijeme trajanja, status, preglednik na kojem su izvedene i sl., dok se na Slici 12. nalazi prikaz jedne od sesija unutar koje se nalazi video koji prikazuje svaki korak koji je poduzet na Five mrežnom sjedištu. Također, desno od videa prikazan je svaki korak koji je izvršen i vrijeme u kojem je izvršen. BrowserStack je bitan jer predstavlja *„device-lab"* u oblaku i nudi mogućnost konfiguracije različitih preglednika, njegovih verzija te operacijskih sustava: Windows, Mac, mobile, Chrome, Opera, Safari, Firefox. Također, iz BrowserStack-a postoji mogućnost prijave *bugova* u Jira-i i mogu se priložiti bitne stvari poput videa, opisa okuženja i sl.



Slika 10. Nadzorna ploča BrowserStack-a i pregled builda.



Slika 11. Prikaz triju sesija na BrowserStack-u.



Slika 12. Prikaz jedne sesije na BrowserStack-u.

Na samom kraju, bitno je spomenuti i Slack. Nakon svakog izvršenog testa generira se izvještaj i šalje se na Slack koji unutar tvrtke Five predstavlja zajednički kolaboracijski alat. Prilikom integracije moguće je odabrati osobu na Slack-u ili Slack kanal na koji će pristizati izvještaji. Na ovaj su način svi korisnici pravovremeno informirani o potencijalnim greškama ili o tome da je sve spremno za poduzimanje daljnjih koraka u otklanjanju otkrivenih grešaka.

3.5. Analiza rezultata testiranja Five mrežnog sjedišta

Prilikom testiranja Five mrežnog sjedišta, kreirana su dvadeset i dva automatska testa. Svi testovi su uspješno provedeni, a nedosljednosti se pojavljuju kod prvog testa koje proizlaze iz dinamičkog svojstva naslovne stranice Five mrežnog sjedišta na kojoj se slike i animacije neprestano izmjenjuju. Zbog toga je bilo jako teško snimiti ispravnu snimku zaslona. Vrijedno je istaknuti da za vrijeme testiranja nijedan *bug* nije pronađen. Svi alati korišteni prilikom testiranja pokazali su se uspješnima u provedbi testova. Na osnovi provedenog testiranja može se zaključiti i kako su testovi ispunili kriterij stabilnosti, te da stoga zadovoljavaju potrebe automatskog testiranja. Na ovaj način su obje okvirne hipoteze, postavljene u radu, i potvrđene. S obzirom na pojedine izazove na koje se naišlo prilikom automatskog testiranja, osobito kada je riječ o provođenju vizualne usporedbe kod prvog testa, ostaje prostora za unaprjeđenje istog. Kako su automatski testovi provedeni na operacijskom sustavu Windows 10, u budućnosti bi bilo dobro ponoviti testne slučajeve i na drugim operacijskim sustavima kako bi se usporedilo testiranje na različitim operacijskim sustavima i eventualno otkrile sličnosti i razlike testiranja.

3.6. Izazovi prilikom testiranja

Prilikom testiranja naišlo se na nekoliko izazova i problema. Jedan od većih izazova bio je taj što su se prilikom izrađivanja mrežnog sjedišta najviše koristile klase, a ne ID atributi HTML elementa preko kojih je najlakše i najbolje pristupiti nekom elementu na mrežnoj stranici. Problem s klasama, xPathovima, HTML elementima i pojedinim drugim selektorima je u tome što se oni lako mogu promijeniti na mrežnom sjedištu i onda ih je potrebno i nužno promijeniti unutar *pageobjects* direktorija. Također, xPath može biti problematičan jer promjene u strukturi stranice mogu utjecati na njega i test može biti neuspješan, iako se u određenom trenutku uspješno provodio. Još jedan od većih izazova vezan je uz naslovnu stranicu Five mrežnog sjedišta. Prilikom pokretanja Percy-a za prvi test, snimka zaslona se nije mogla dobiti u zadovoljavajućem obliku, jer je naslovna stranica dinamična odnosno sklona izmjenama sadržaja, pa se pri svakom pokretanju testa dobivala različita snimka zaslona, što je i vidljivo na Slici 14. Neki od ostalih izazova bili su problemi s Chrome driverom odgovornim za pokretanje testova u Chrome pregledniku. U određenom trenutku nedostajala je podrška za novije verzije Chrome preglednika te su se testovi mogli pokretati isključivo na verziji 81. U

međuvremenu su ti izazovi riješeni i sada se testovi mogu pokretati na bilo kojoj verziji Chrome-a.

4. Zaključak

Testiranje programa neizostavni je dio njegovog razvojnog ciklusa, a prilikom testiranja potrebno je ukazati na nedostatke i pogreške koje su nastale u različitim fazama razvoja. Programer, dizajner, tester i bilo koja druga osoba u životnom ciklusu programa ima bitnu ulogu u njegovom razvoju, ali tester kao tester je vrlo bitan, jer je upravo njegov zadatak da pronađe greške koje su nastale prilikom razvoja programa ili programskog rješenja. Također, postoji veliki broj slučajeva u kojima je došlo do tehničkih i financijskih neprilika upravo zbog toga što program nije bio prikladno testiran ili uopće nije bio testiran. Upravo ti slučajevi prikazuju važnost njegova testiranja. Postoje dvije vrste testiranja, a to su ručno i automatsko i, kako je u radu pokazano, svako od njih ima svoje prednosti i nedostatke.

U praktičnom dijelu rada prikazano je automatsko testiranje Five mrežnog sjedišta i vizualna usporedba istog. Automatski testovi pisali su se u Visual Studio Code uređivaču kôda u JavaScrpit programskom jeziku. Osim toga, korišteni su i WebdriverIO kao glavni alat za testiranje, Node.js platforma, Chrome internet preglednik, Percy za vizualnu usporedbu, TestRail, Allure i BrowserStack. Testovi su bili podijeljeni u tri *spec.js* datoteke, a svaka *spec.js* datoteka imala je odgovarajuću *page.js* datoteku u kojoj su se nalazili *page* objekti. Sveukupno, napisano je dvadeset dva testa. Na Five mrežnom sjedištu testirane su opcije poput '*scrollanja*' stranicom, klikanja na određene elemente, snimanje zaslona i sl. Također, u radu su predstavljeni svi koraci za instalaciju svih alata koji su se koristili. Napisani testovi se uz instalaciju alata i postavljanje odgovarajućeg okruženja mogu nadopunjavati i proširivati.

Ovim testiranjem ustanovljeno je kako je mrežno sjedište tvrtke Five izrazito stručno kreirano, jedini nedostatak po pitanju automatskog testiranja je bio taj što se prilikom izrade mrežnog sjedišta nisu u većoj mjeri koristili ID atributi HTML elementa. Osim toga, mrežno sjedište ne sadrži *bugove* koji mogu utjecati na korisničko iskustvo korištenja istog. Također, ovaj rad pokazuje kako se na jednostavan način uz poznavanje osnova programiranja može testirati bilo koje mrežno sjedište te unaprijediti djelovanje istog. Automatski testovi uvelike olakšavaju proces testiranja programskih rješenja jer nije potrebno ručno provjeravati njihovu ispravnost. Također, testovi se mogu integrirati u postojeći tijek rada programskog rješenja i greške se mogu pravovremeno otkriti, a posebice one koje golim okom nisu vidljive – vizualnom usporedbom 'piksel po piksel'. Takav pristup omogućava postizanje višeg stupnja sigurnosti, bolju kvalitetu i smanjuje se mogućnost da krajnji korisnici pronađu *bugove*.

Literatura

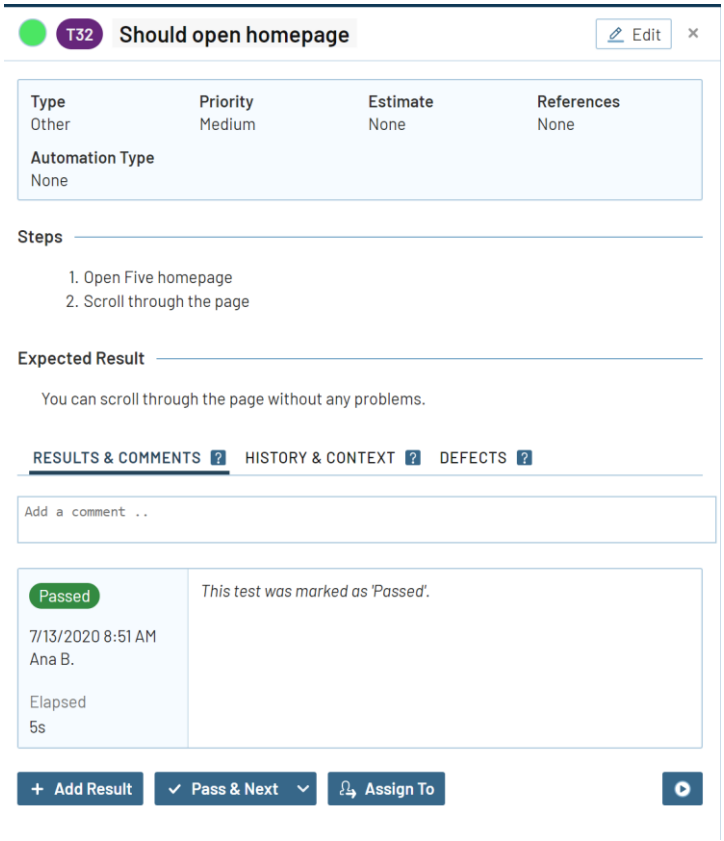
1. Allure Framework. URL: https://docs.qameta.io/allure/ (2020-08-03)
2. Binary Terms: URL: https://binaryterms.com/difference-between-black-box-testing-and-white-box-testing.html (2020-07-21)
3. C, Padmini. Beginners guide to software testing. [s.l.], [s.n.]. URL: https://www.softwaretestingclass.com/wp-content/uploads/2016/06/Beginner-Guide-To-Software-Testing.pdf (2020-07-01)
4. Codefirst. URL: https://www.codefirst.co.uk/blog/difference-black-white-box-testing/ (2020-07-21)
5. Graham, Dorothy…[et. al.]. Foundations of Software Testing. Australia…[et.al]: Cengage Learning Emea, 2008. URL: https://www.capeco.org/store-imagenes/files/gg.pdf (2020-07-03)
6. Guru 99. URL: https://www.guru99.com/alpha-beta-testing-demystified.html (2020-07-20)
7. Guru 99. URL: https://www.guru99.com/automation-testing.html (2020-07-09)
8. Guru 99. URL: https://www.guru99.com/grey-box-testing.html (2020-07-21)
9. Guru 99. URL: https://www.guru99.com/manual-testing.html (2020-07-08)
10. Guru 99. URL: https://www.guru99.com/software-testing.html (2020-07-01)
11. Guru 99. URL: https://www.guru99.com/software-testing-career-complete-guide.html (2020-07-07)
12. Medium. URL: https://medium.com/tech-tajawal/page-object-model-pom-design-pattern-f9588630800b (2020-08-03)
13. Pearson, LaTonya. The four levels of software testing, 2015. URL: https://www.seguetech.com/the-four-levels-of-software-testing/ (2020-07-20)
14. Percy. URL: https://docs.percy.io/docs/getting-started (2020-08-03)
15. Programska podrška. // Hrvatska enciklopedija. URL: http://www.enciklopedija.hr/natuknica.aspx?ID=50557 (2020-07-01)
16. Scientech Easy. URL: https://www.scientecheasy.com/2019/09/difference-manual-vs-automation-testing.html/ (2020-07-13)

17. Shah, Jaymine. Beginner's manual software testing guide, 2019. URL: https://medium.com/techcompose/beginners-manual-software-testing-guide-8096bcd99a14 (2020-07-08)
18. Smartbear. URL: https://smartbear.com/learn/automated-testing/what-is-automated-testing/ (2020-07-09)
19. Software testing help. URL: https://www.softwaretestinghelp.com/automation-testing-tutorial-1/ (2020-07-09)
20. Software Testing Help. URL: https://www.softwaretestinghelp.com/browserstack-tutorial/ (2020-08-03)
21. Software Testing Help. URL: https://www.softwaretestinghelp.com/how-to-set-defect-priority-and-severity-with-defect-triage-process/ (2020-07-05)
22. Spillner, Andreas; Linz, Tilo; Schaefer, Hans. Software Testing Foundations: A study guide for the certified tester exam. Santa Barbara: Rocky Nook Inc., 2014. URL: http://prof.mau.ac.ir/images/Uploaded_files/Software%20Testing%20Foundations%20A%20Study%20Guide%20for%20the%20Certified%20Tester%20Exam%5B5309302%5D.PDF (2020-07-07)
23. Testim. URL: https://www.testim.io/blog/test-automation-vs-manual-testing/ (2020-07-13)
24. TestRail. URL: https://www.gurock.com/testrail/docs/user-guide/getting-started/walkthrough (2020-08-03)
25. Tomašević, Violeta. Razvoj aplikativnog softvera. Beograd: Univerzitet Singidunum, 2017., str. 129-132. URL: https://singipedia.singidunum.ac.rs/preuzmi/40784-razvoj-aplikativnog-softvera/3171 (2020-07-04)
26. Try QA. URL: http://tryqa.com/why-is-testing-necessary/ (2020-07-01)
27. Unadkat, Jash. Manual testing for beginners, 2020. URL: https://www.browserstack.com/guide/manual-testing-tutorial (2020-07-08)
28. Visual Studio Code. URL: https://code.visualstudio.com/docs (2020-08-03)
29. Webdriver IO. URL: http://v4.webdriver.io/ (2020-08-03)
30. Webdriver IO. URL: https://webdriver.io/docs/selectors.html (2020-08-03)
31. Webdriver IO. URL: https://webdriver.io/docs/pageobjects.html (2020-08-03)

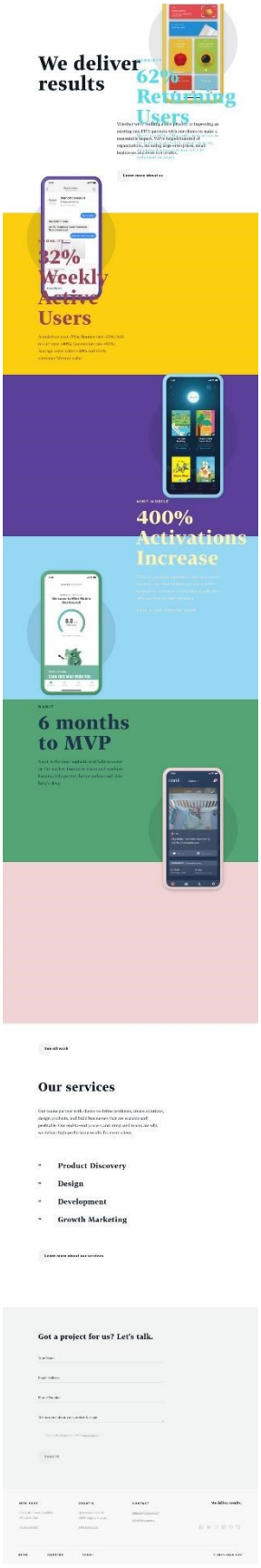
Prilozi

Slike testova i slike odgovarajućih snimki zaslona

Slika 13. prikazuje kako izgleda prvi test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 14. prikazuje kako izgleda Percy snimka zaslona prvog testa. Kao što je i vidljivo, naslovna stranica sadrži elemente koji se prikazuju kao različite animacije te ova snimka zaslona ujedno predstavlja i jedan od većih izazova ovog automatskog testiranja.

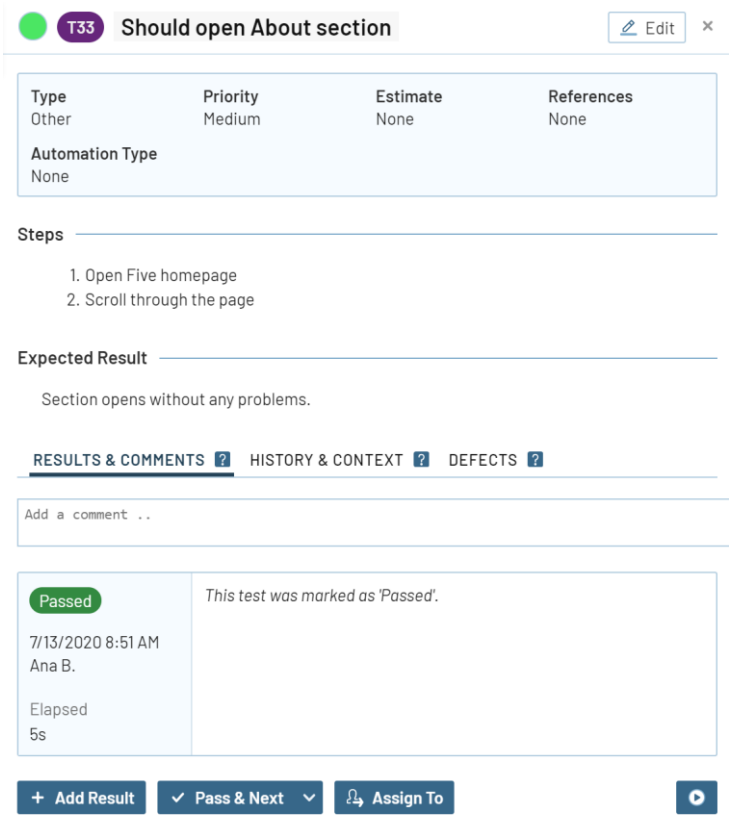


Slika 13. Prikaz prvog testa u TestRail-u.

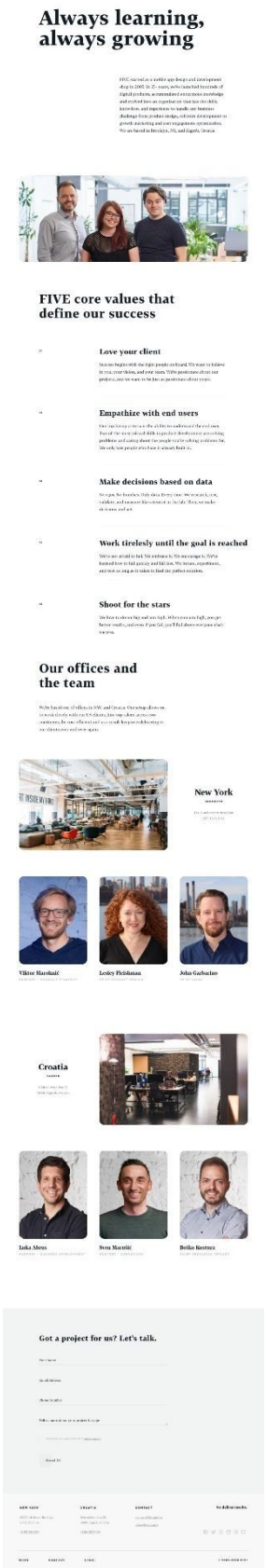


Slika 14. Percy snimka zaslona prvog testa.

Slika 15. prikazuje kako izgleda drugi test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 16. prikazuje kako izgleda Percy snimka zaslona drugog testa.

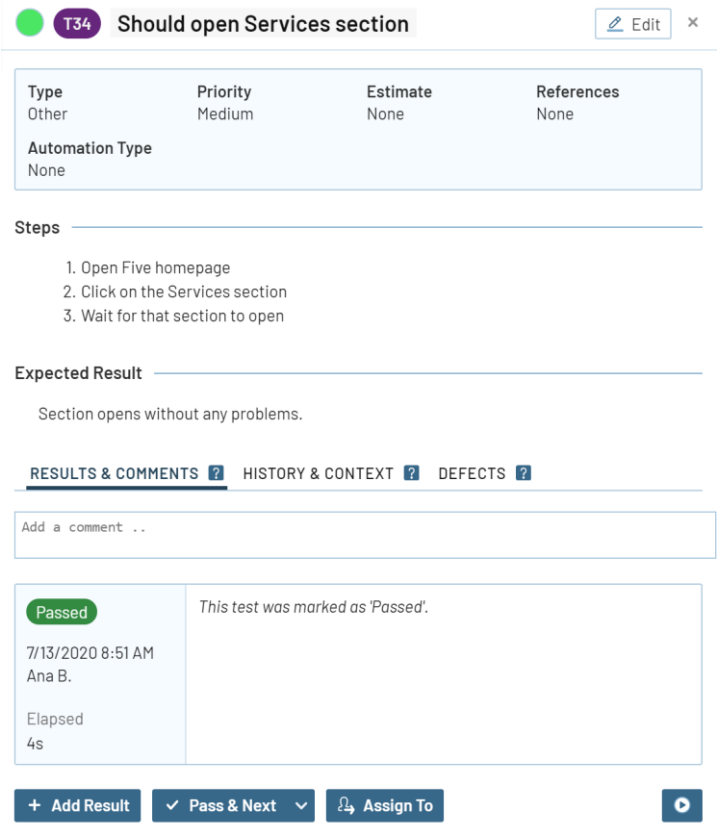


Slika 15. Prikaz drugog testa u TestRail-u.

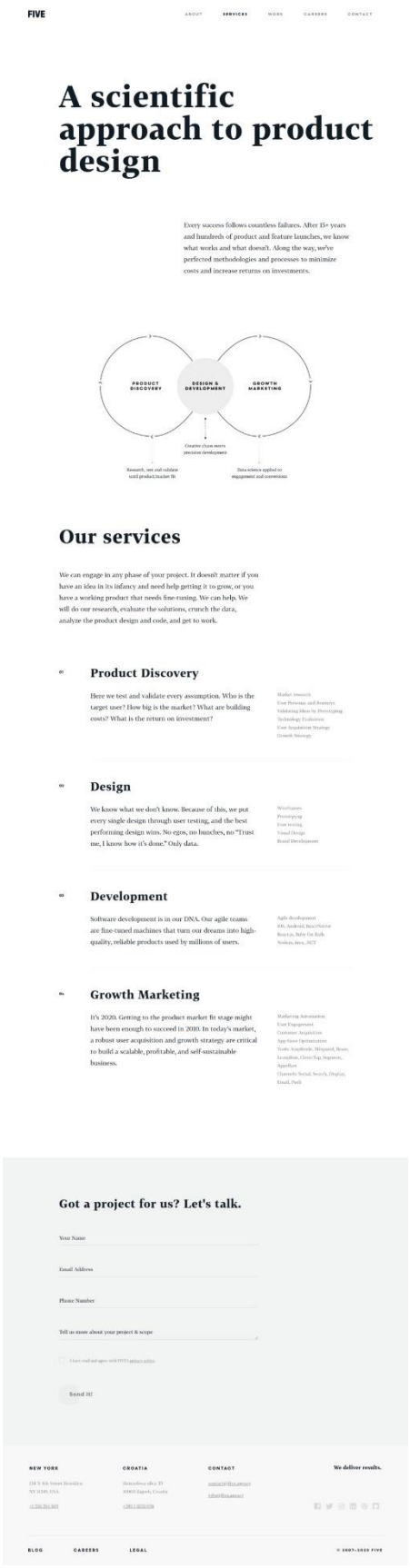


Slika 16. Percy snimka zaslona drugog testa.

Slika 17. prikazuje kako izgleda treći test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 18. prikazuje kako izgleda Percy snimka zaslona trećeg testa.

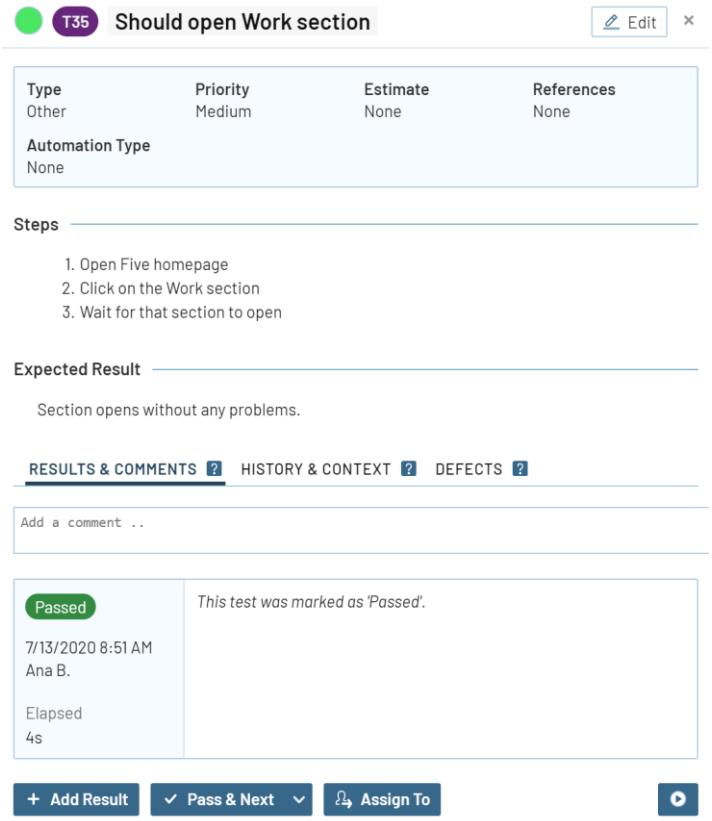


Slika 17. Prikaz trećeg testa u TestRail-u.



Slika 18. Percy snimka zaslona trećeg testa.

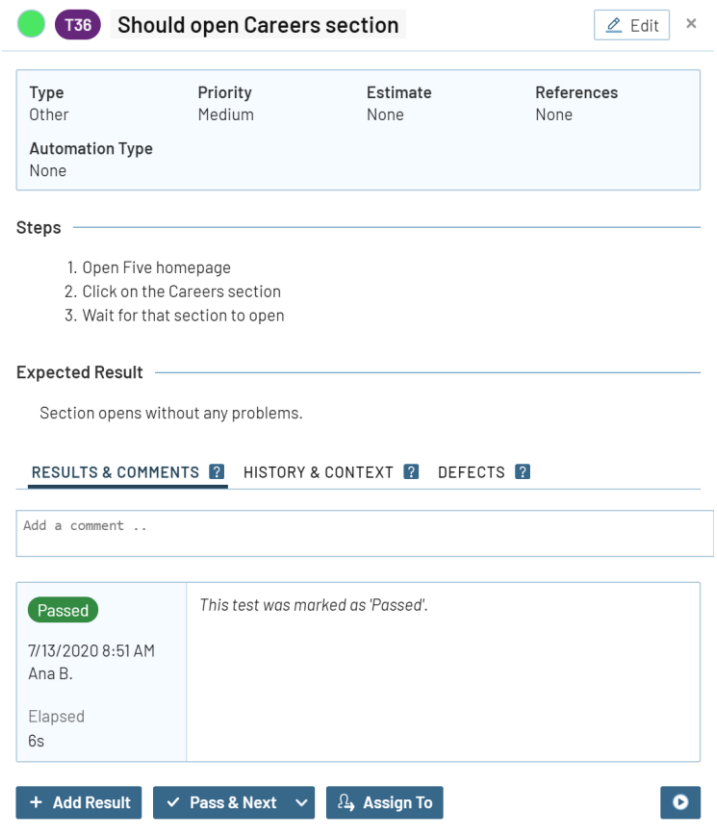
Slika 19. prikazuje kako izgleda četvrti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 20. prikazuje kako izgleda Percy snimka zaslona četvrtog testa.



Slika 19. Prikaz četvrtog testa u TestRail-u.

Slika 20. Percy snimka zaslona četvrtog testa.

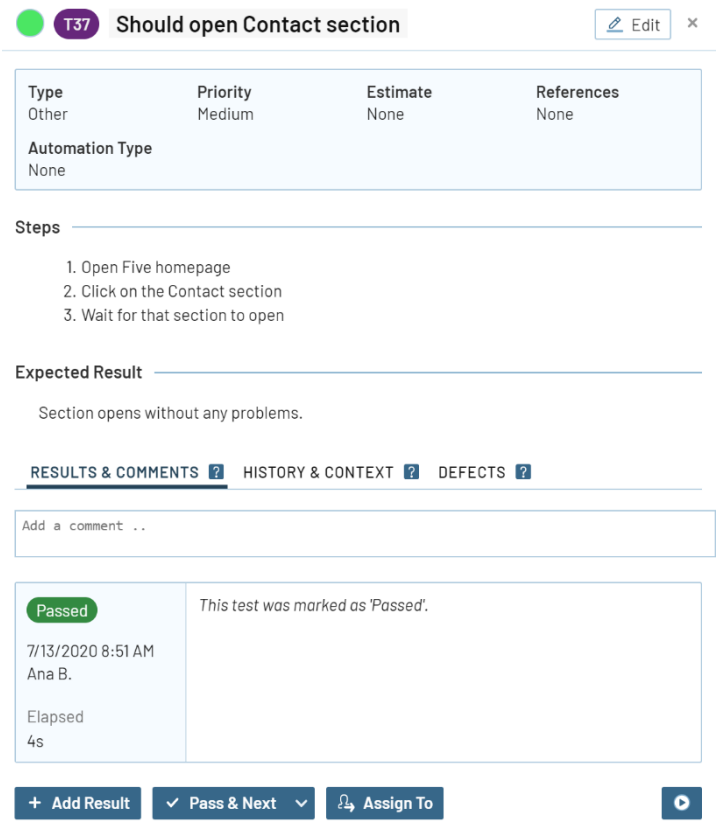
Slika 21. prikazuje kako izgleda peti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 22. prikazuje kako izgleda Percy snimka zaslona petog testa.



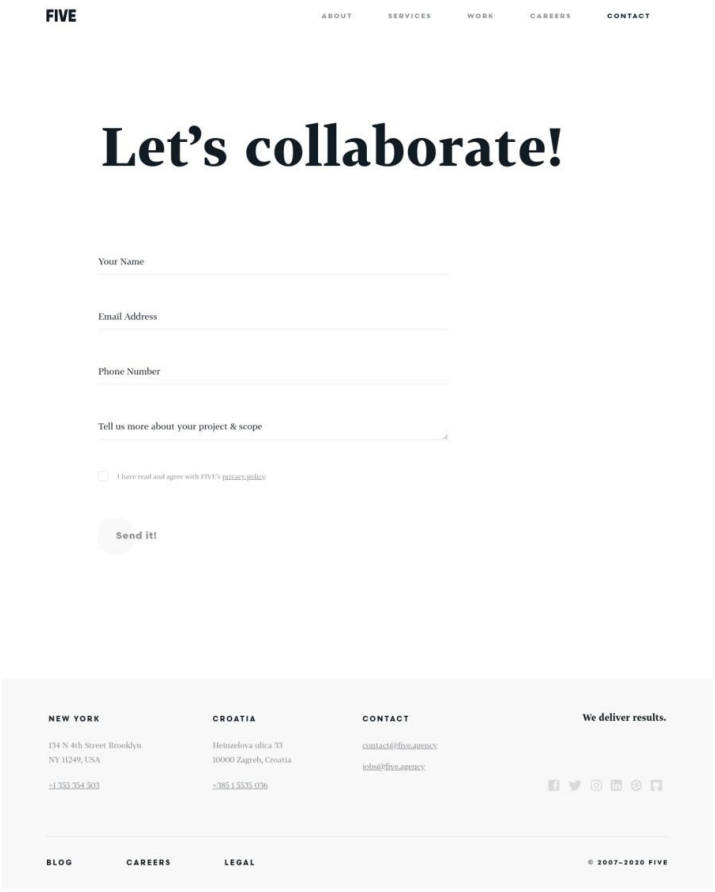
Slika 21. Prikaz petog testa u TestRail-u.

Slika 22. Percy snimka zaslona petog testa.

Slika 23. prikazuje kako izgleda šesti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 24. prikazuje kako izgleda Percy snimka zaslona šestog testa.

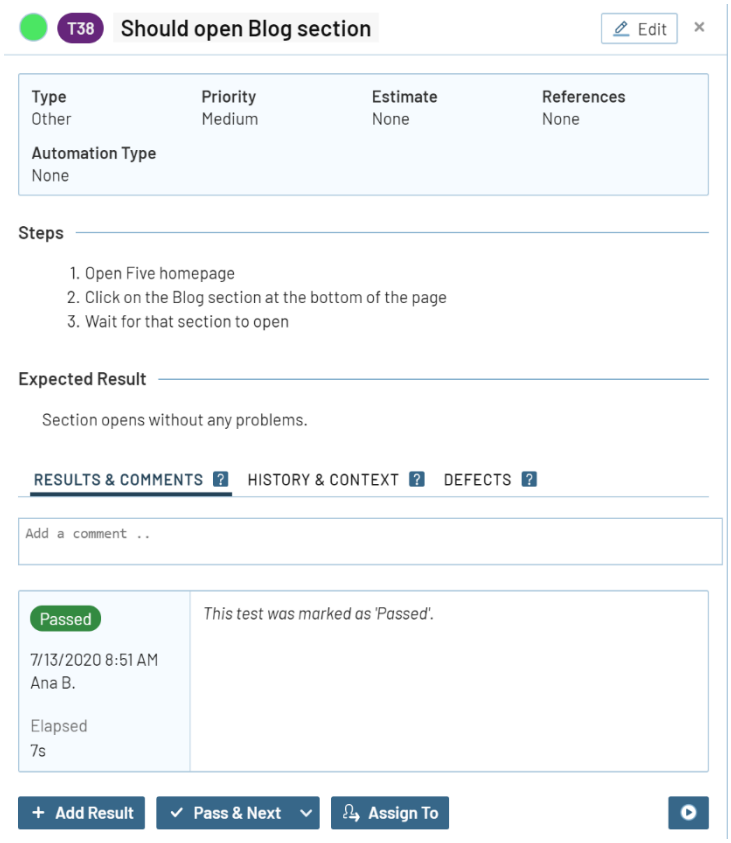


Slika 23. Prikaz šestog testa u TestRail-u.

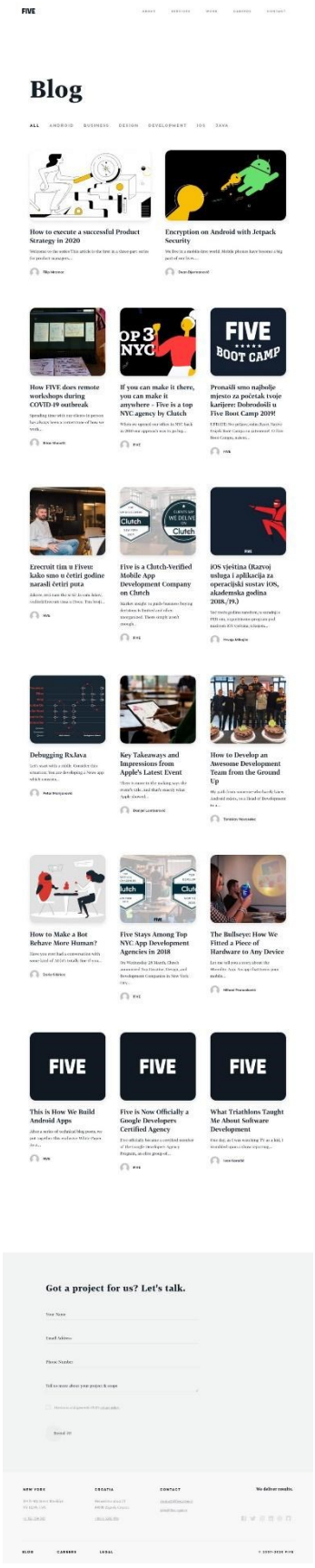


Slika 24. Percy snimka zaslona šestog testa.

Slika 25. prikazuje kako izgleda sedmi test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 26. prikazuje kako izgleda Percy snimka zaslona sedmog testa.

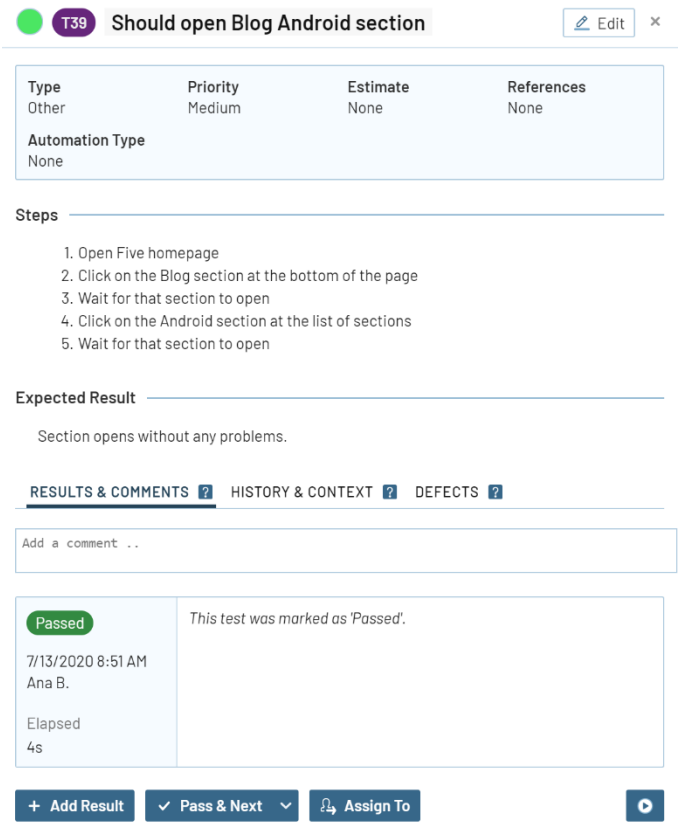


Slika 25. Prikaz sedmog testa u TestRail-u.

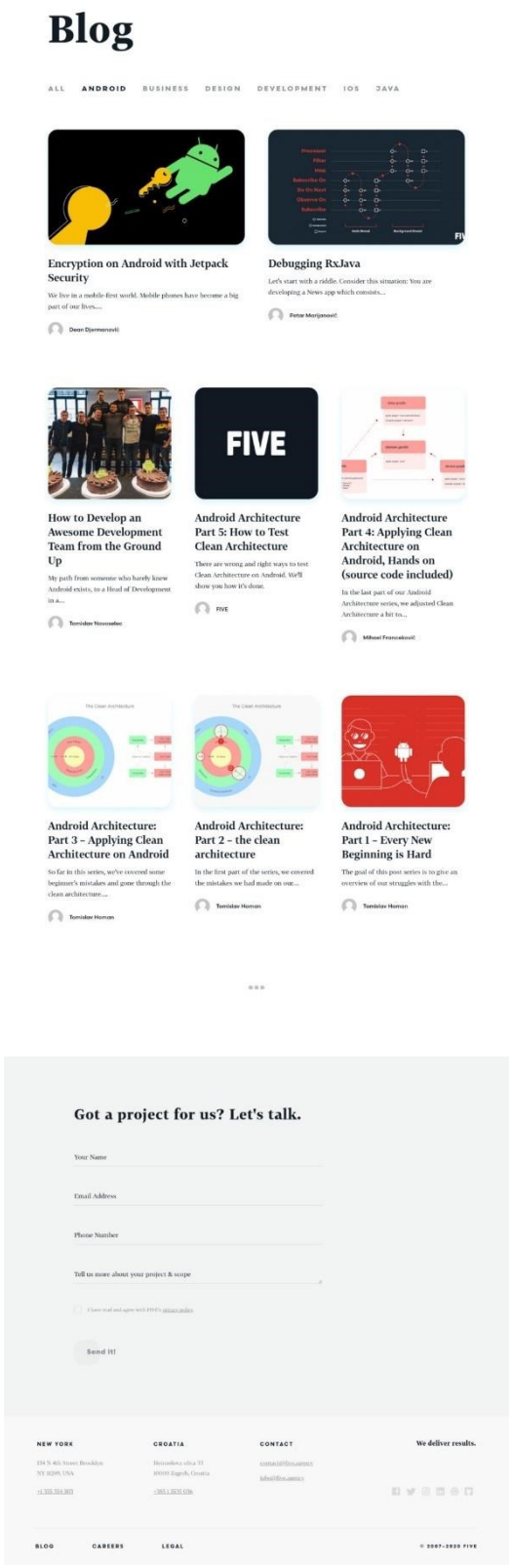


Slika 26. Percy snimka zaslona sedmog testa.

Slika 27. prikazuje kako izgleda osmi test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 28. prikazuje kako izgleda Percy snimka zaslona osmog testa.

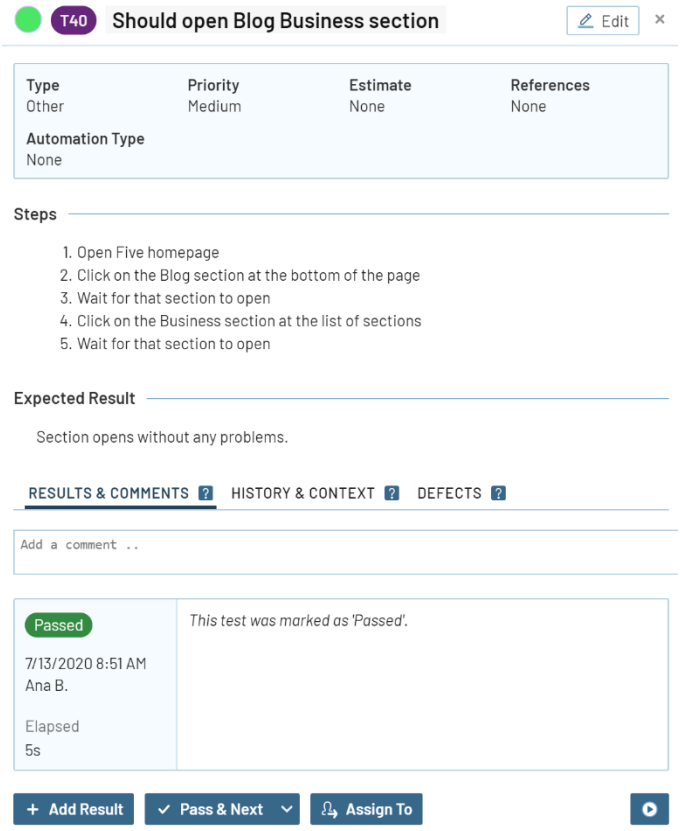


Slika 27. Prikaz osmog testa u TestRail-u.

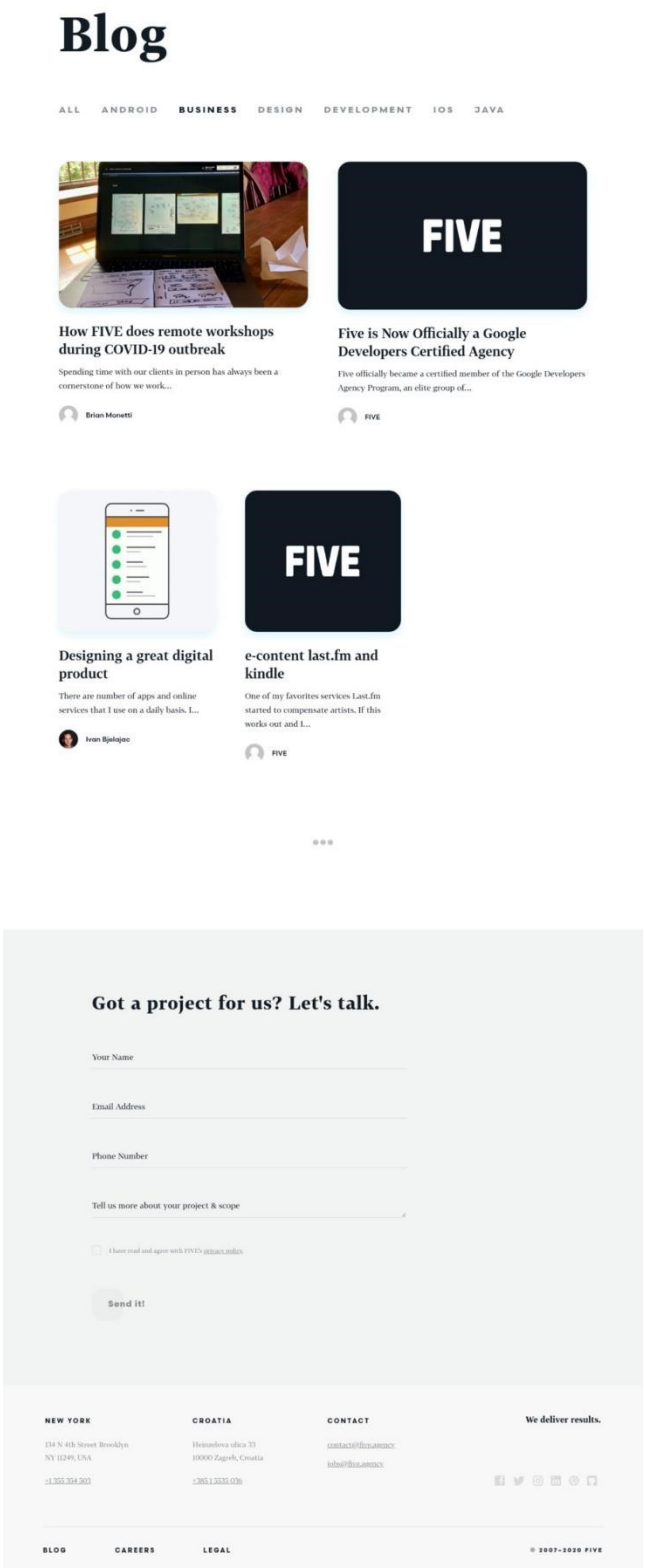


Slika 28. Percy snimka zaslona osmog testa.

Slika 29. prikazuje kako izgleda deveti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 30. prikazuje kako izgleda Percy snimka zaslona devetog testa.

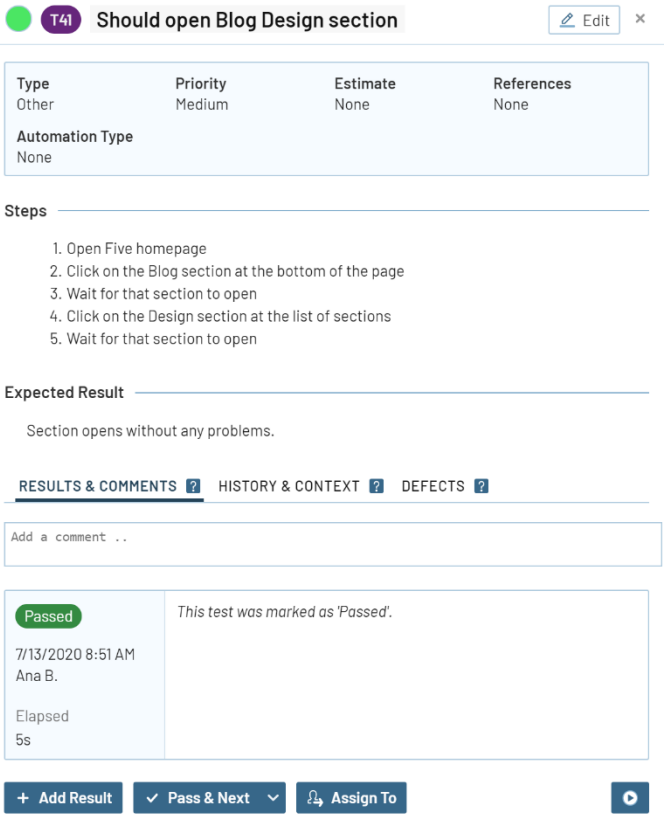


Slika 29. Prikaz devetog testa u TestRail-u.

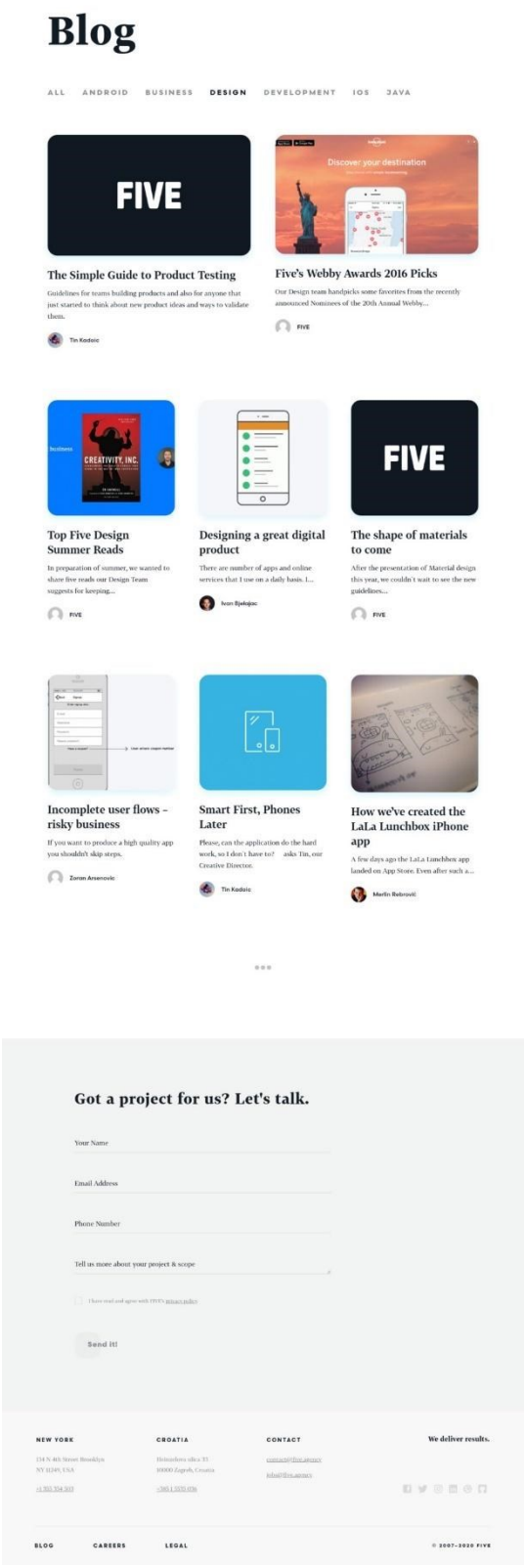


Slika 30. Percy snimka zaslona devetog testa.

Slika 31. prikazuje kako izgleda deseti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 32. prikazuje kako izgleda Percy snimka zaslona desetog testa.

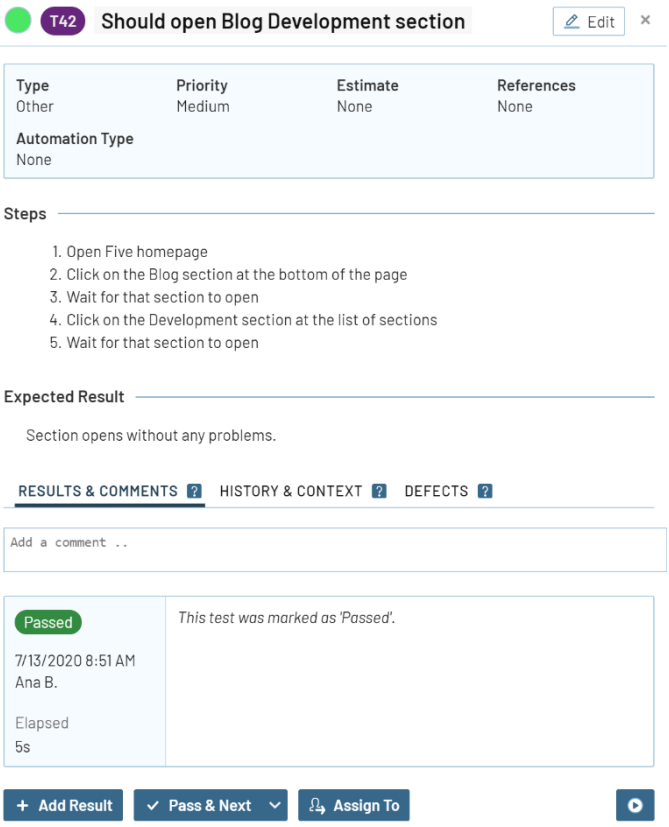


Slika 31. Prikaz desetog testa u TestRail-u.

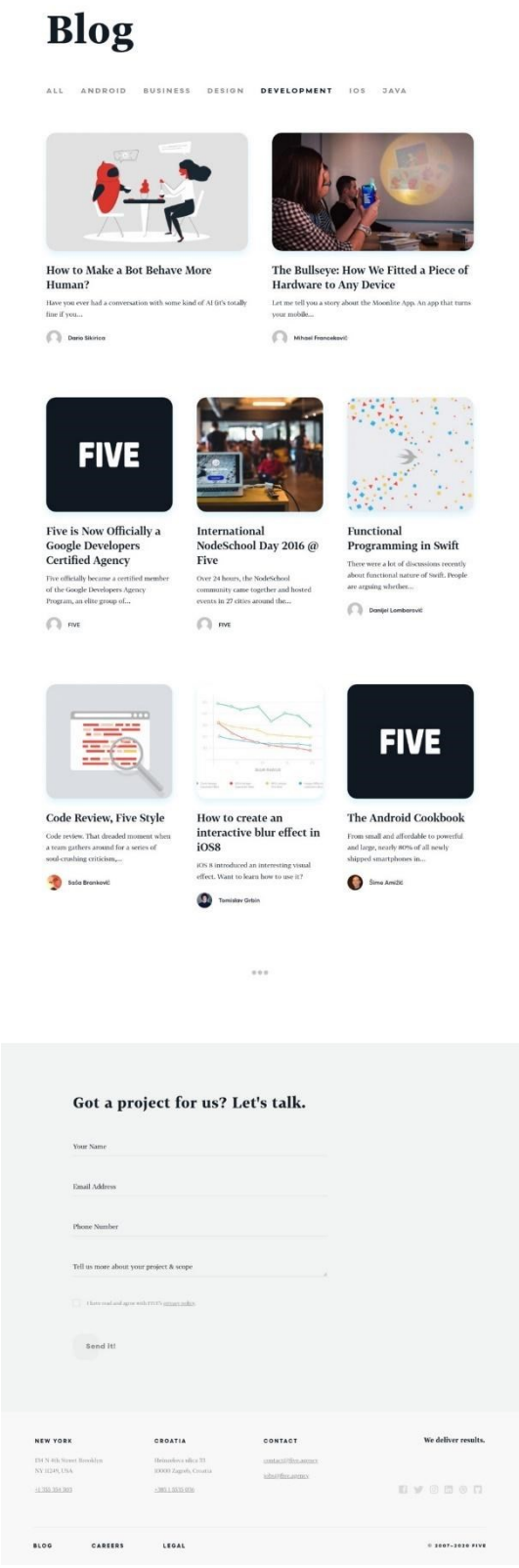


Slika 32. Percy snimka zaslona desetog testa.

Slika 33. prikazuje kako izgleda jedanaesti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 34. prikazuje kako izgleda Percy snimka zaslona jedanaestog testa.

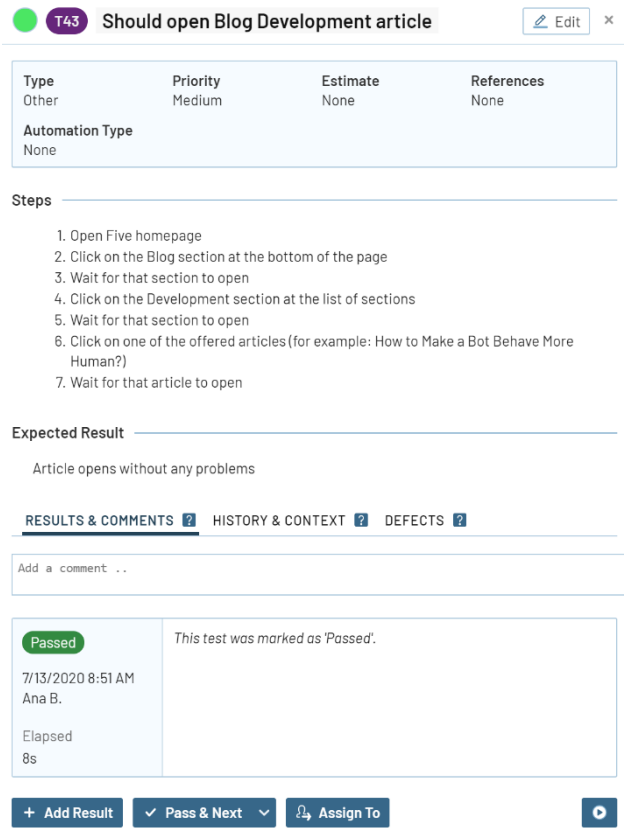


Slika 33. Prikaz jedanaestog testa u TestRail-u.



Slika 34. Percy snimka zaslona jedanaestog testa.

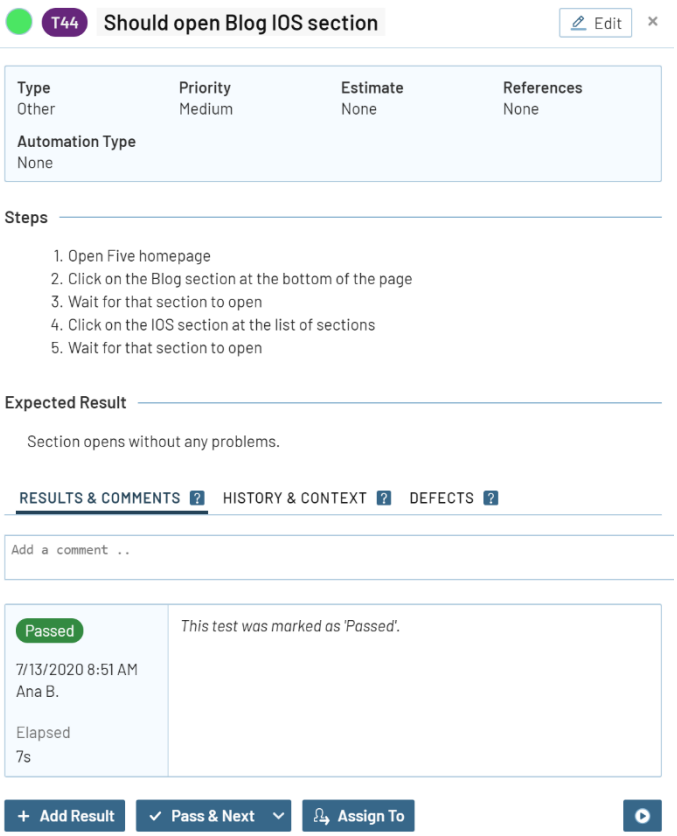
Slika 35. prikazuje kako izgleda dvanaesti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 36. prikazuje kako izgleda Percy snimka zaslona dvanaestog testa.



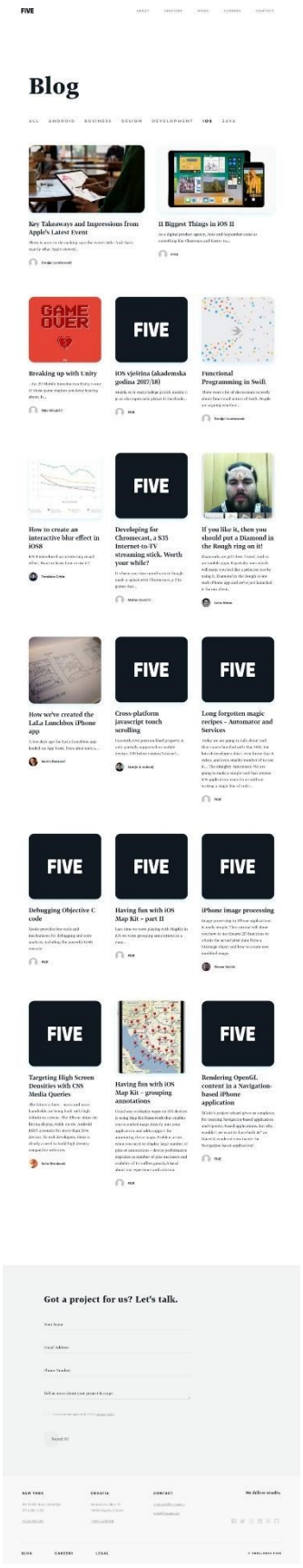
Slika 35. Prikaz dvanaestog testa u TestRail-u.

Slika 36. Percy snimka zaslona dvanaestog testa.

Slika 37. prikazuje kako izgleda trinaesti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 38. prikazuje kako izgleda Percy snimka zaslona trinaestog testa.

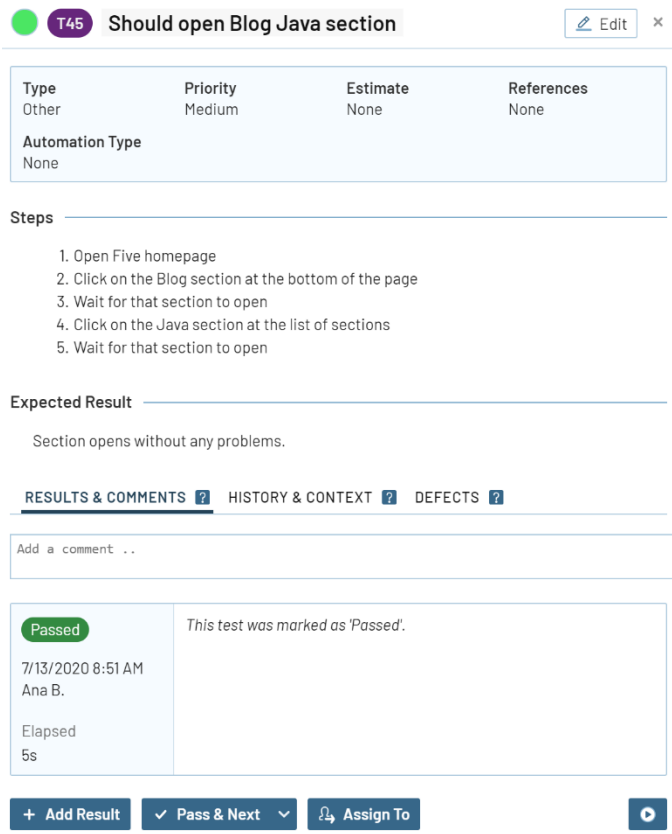


Slika 37. Prikaz trinaestog testa u TestRail-u.

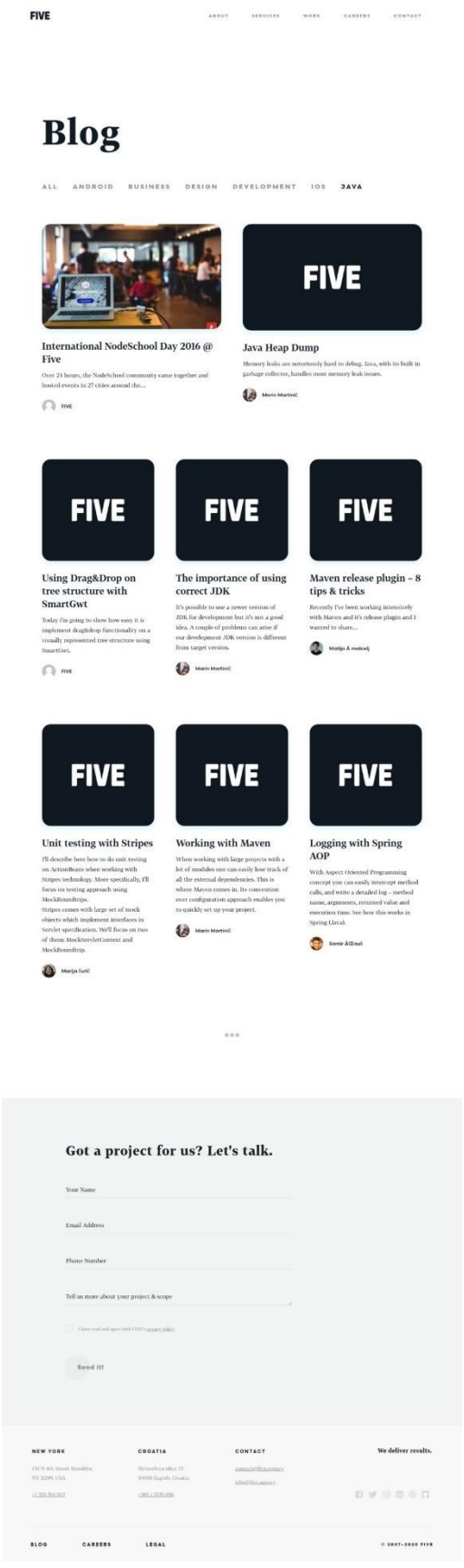


Slika 38. Percy snimka zaslona trinaestog testa.

Slika 39. prikazuje kako izgleda četrnaesti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 40. prikazuje kako izgleda Percy snimka zaslona četrnaestog testa.

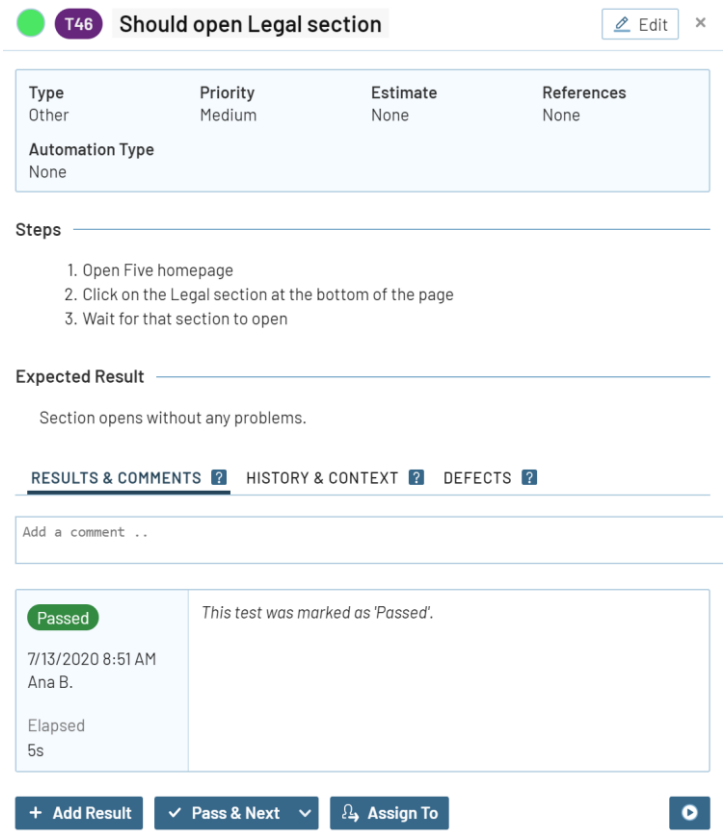


Slika 39. Prikaz četrnaestog testa u TestRail-u.



Slika 40. Percy snimka zaslona četrnaestog testa.

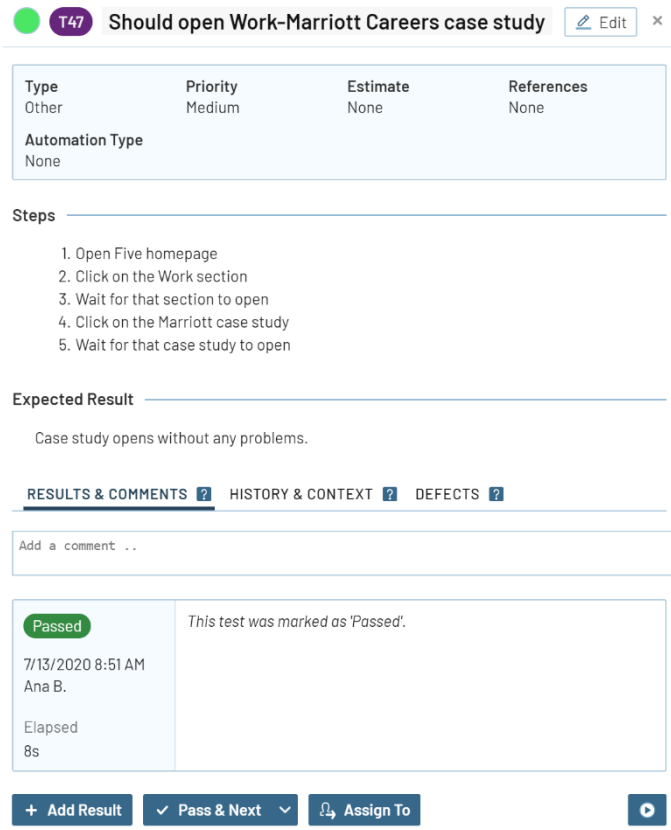
Slika 41. prikazuje kako izgleda petnaesti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 42. prikazuje kako izgleda Percy snimka zaslona petnaestog testa.



Slika 41. Prikaz petnaestog testa u TestRail-u.

Slika 42. Percy snimka zaslona petnaestog testa.

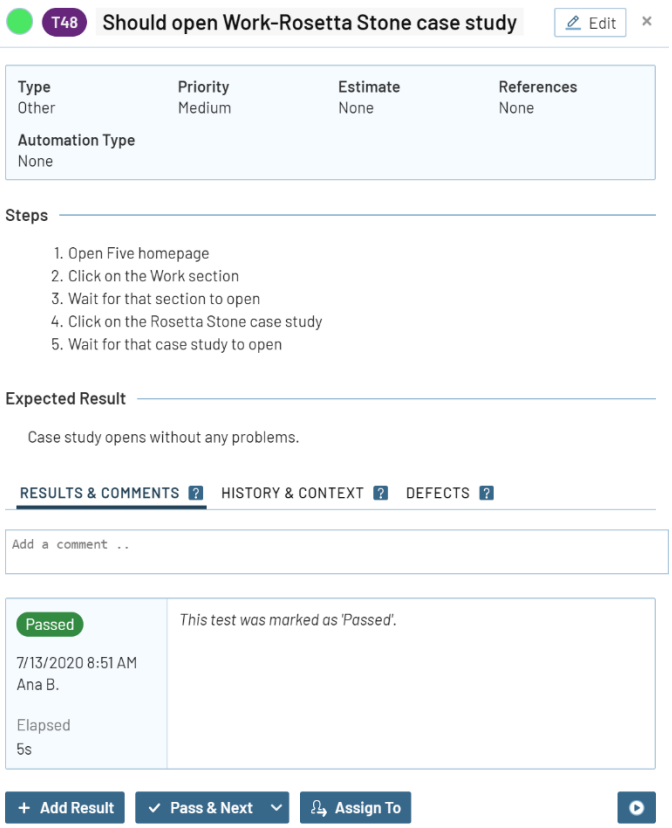
Slika 43. prikazuje kako izgleda šesnaesti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 44. prikazuje kako izgleda Percy snimka zaslona šesnaestog testa.



Slika 43. Prikaz šesnaestog testa u TestRail-u.

Slika 44. Percy snimka zaslona šesnaestog testa.

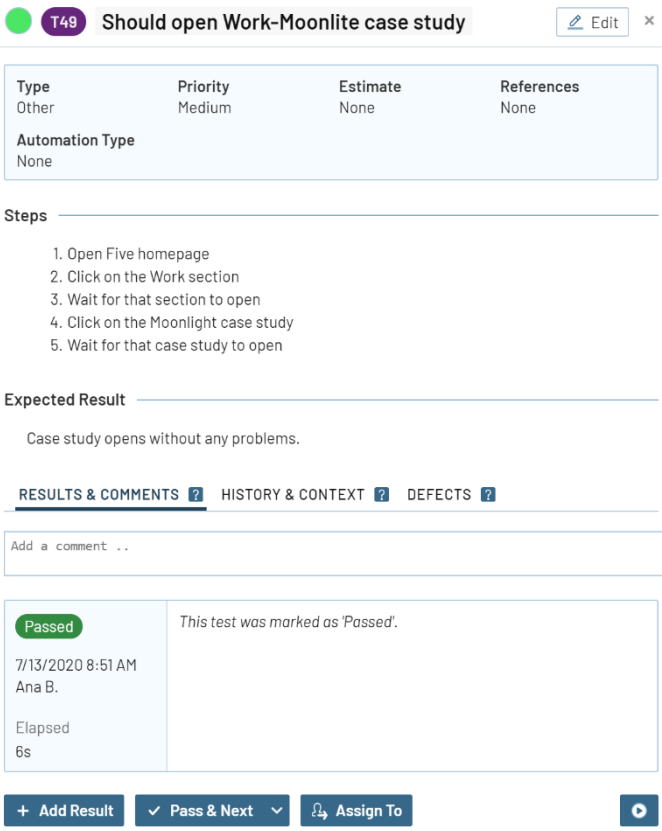
Slika 45. prikazuje kako izgleda sedamnaesti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 46. prikazuje kako izgleda Percy snimka zaslona sedamnaestog testa.



Slika 45. Prikaz sedamnaestog testa u TestRail-u.

Slika 46. Percy snimka zaslona sedamnaestog testa.

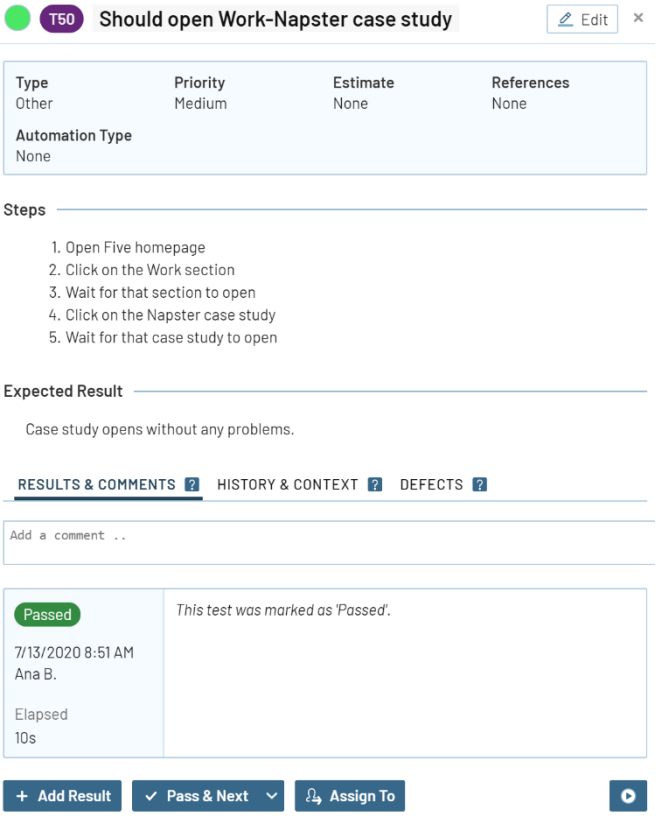
Slika 47. prikazuje kako izgleda osamnaesti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 48. prikazuje kako izgleda Percy snimka zaslona osamnaestog testa.



Slika 47. Prikaz osamnaestog testa u TestRail-u.

Slika 48. Percy snimka zaslona osamnaestog testa.

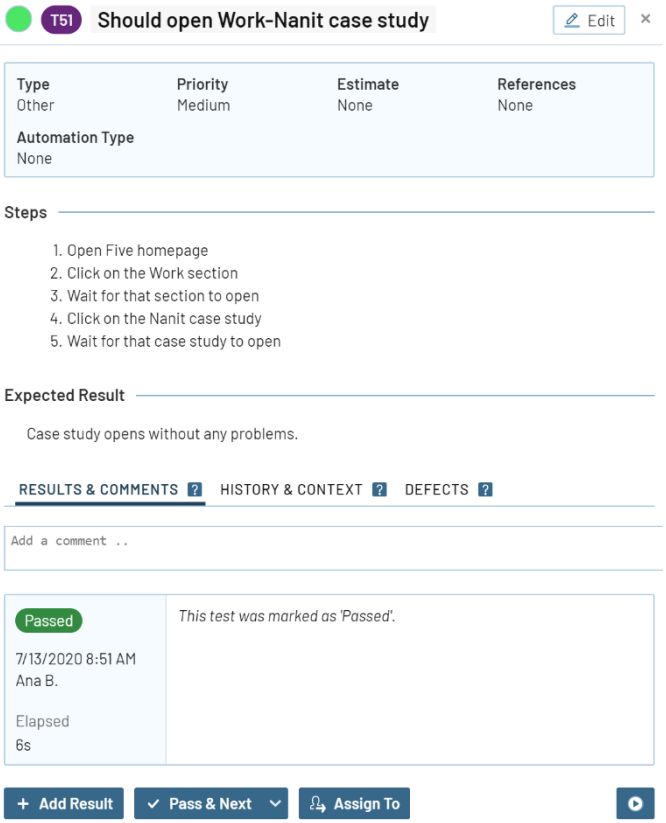
Slika 49. prikazuje kako izgleda devetnaesti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 50. prikazuje kako izgleda Percy snimka zaslona devetnaestog testa.



Slika 49. Prikaz devetnaestog testa u TestRail-u.

Slika 50. Percy snimka zaslona devetnaestog testa.

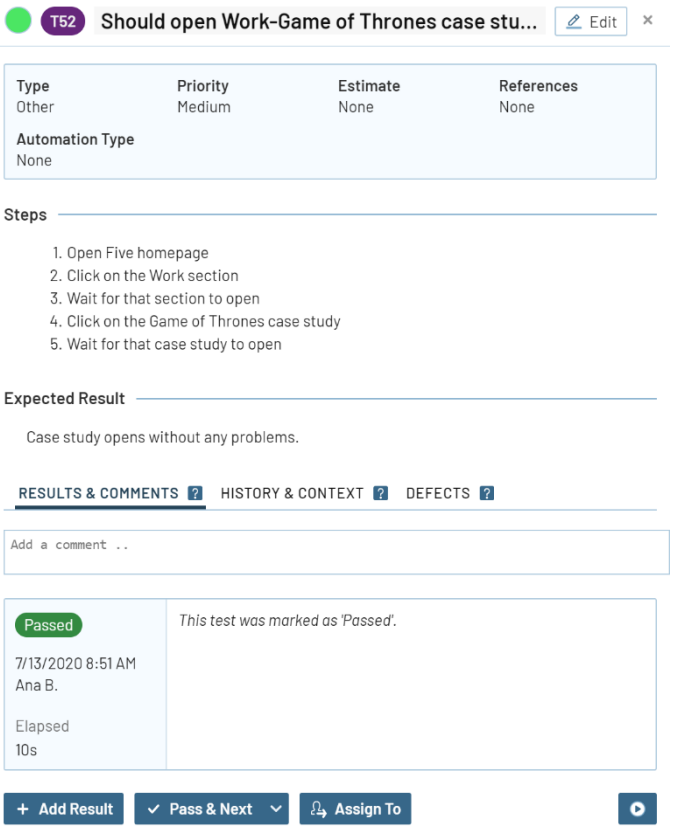
Slika 51. prikazuje kako izgleda dvadeseti test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 52. prikazuje kako izgleda Percy snimka zaslona dvadesetog testa.



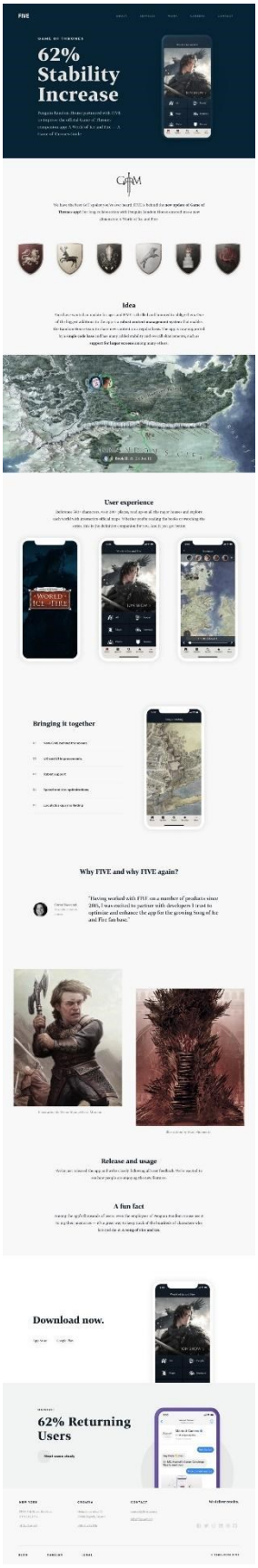
Slika 51. Prikaz dvadesetog testa u TestRail-u.

Slika 52. Percy snimka zaslona dvadesetog testa.

Slika 53. prikazuje kako izgleda dvadeset prvi test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 54. prikazuje kako izgleda Percy snimka zaslona dvadeset prvog testa.

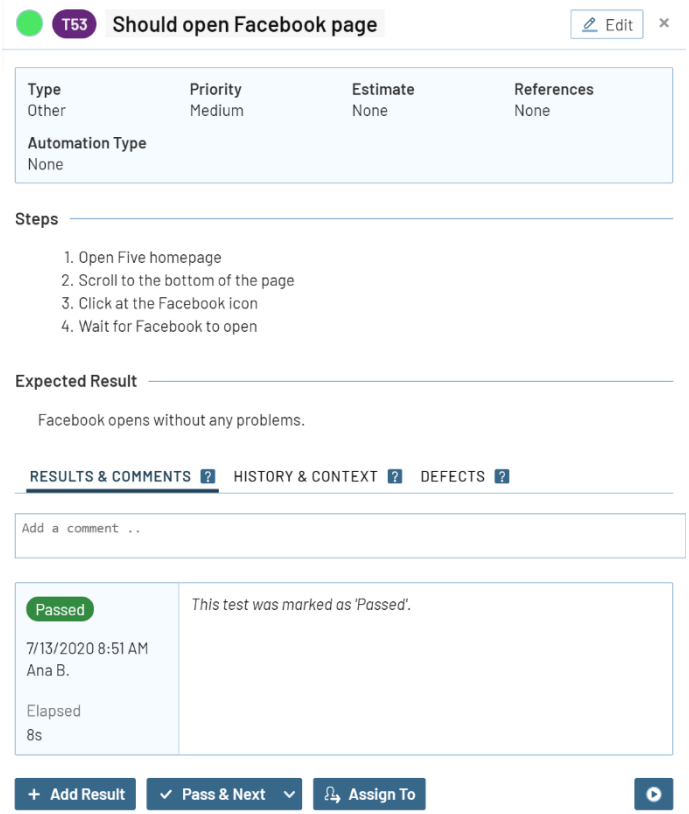


Slika 53. Prikaz dvadeset prvog testa u TestRail-u.

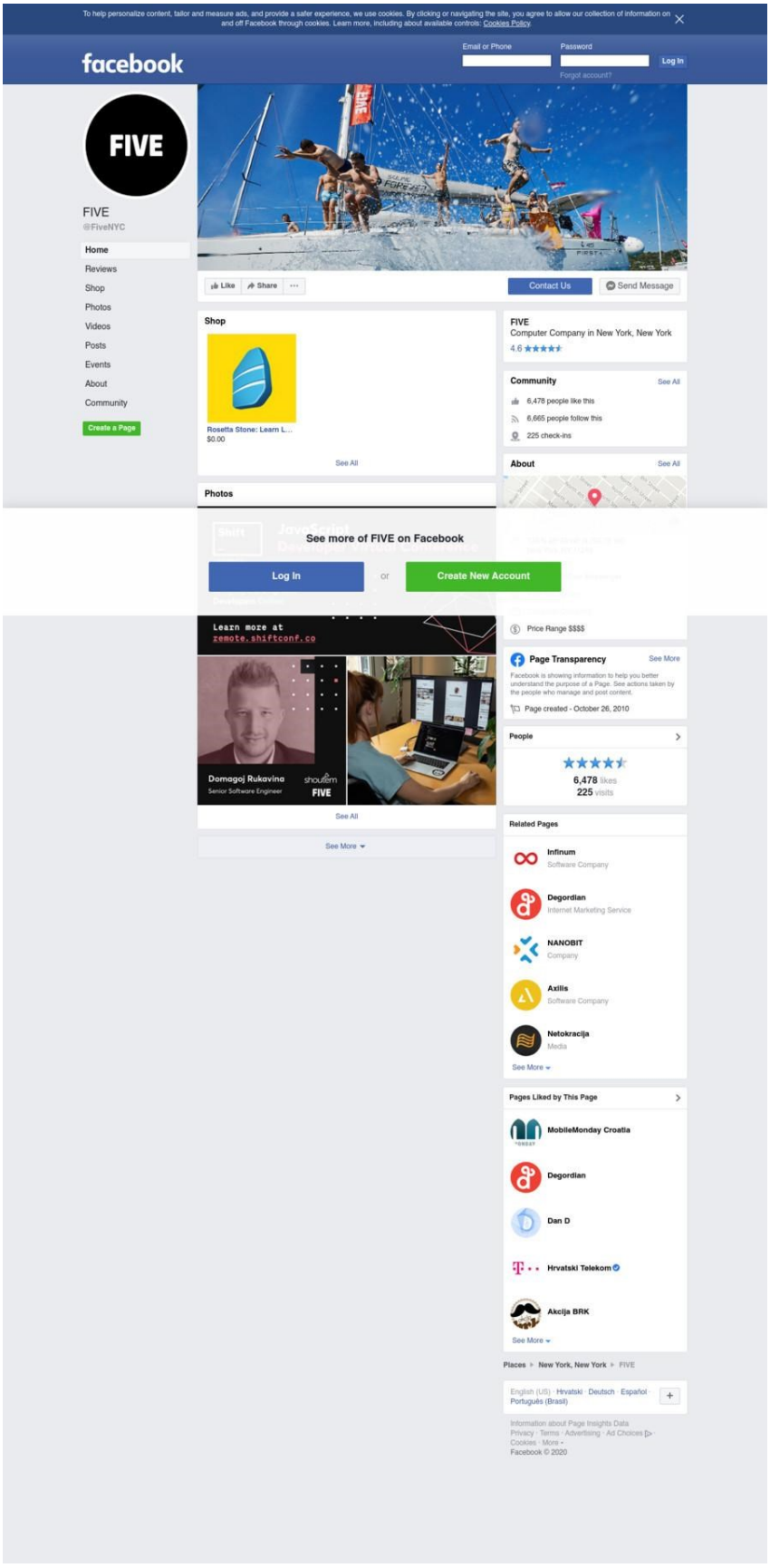


Slika 54. Percy snimka zaslona dvadeset prvog testa.

Slika 55. prikazuje kako izgleda dvadeset drugi test koji je uspješno izvršen nakon što je TestRail naredba pokrenuta u terminalu, a Slika 56. prikazuje kako izgleda Percy snimka zaslona dvadeset drugog testa.



Slika 55. Prikaz dvadeset drugog testa u TestRail-u.



Slika 56. Percy snimka zaslona dvadeset drugog testa.