Usporedba ručnog i automatiziranog testiranja na primjeru web aplikacije "Swag Labs" koristeći alat "Playwright"

Milin, Ana

SVEUČILIŠTE U SPLITU

EKONOMSKI FAKULTET

DIPLOMSKI RAD

USPOREDBA RUČNOG I AUTOMATIZIRANOG TESTIRANJA NA PRIMJERU WEB APLIKACIJE „SWAG LABS“ KORISTEĆI ALAT „PLAYWRIGHT"

Mentor:

Prof.dr.sc. Maja Ćukušić

Student:

Ana Milin

Split, lipanj 2023.

SADRŽAJ:

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, ANA MILIN ,

(ime i prezime)

izjavljujem i svojim potpisom potvrđujem da je navedeni rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja na objavljenu literaturu, što pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio navedenog rada nije napisan na nedozvoljeni način te da nijedan dio rada ne krši autorska prava. Izjavljujem, također, da nijedan dio rada nije korišten za bilo koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Split, 30.06.2023 godine

Vlastoručni potpis : Ana Milin

1. UVOD

1.1. Definicija problema

Programski proizvodi su postali sastavni dio svakodnevnog života, pa se samim time povećala i potreba za osiguranjem kvalitete istih. Glavni cilj svakog programskog proizvoda jest zadovoljavanje zahtjeva krajnjeg korisnika. Kako bi kompanije ostale konkurentne na tržištu, i stekle lojalne korisnike, proces proizvodnje programskog proizvoda mora biti jeftin i izvediv u relativno kratkom roku, a sam softver treba biti koristan, pouzdan i siguran. Kako bi se prethodno navedeno ostvarilo, testiranje softvera je potrebno uzeti kao ključan dio razvoja uspješnog programskog proizvoda. Cilj testiranja softvera je identificirati pogreške u softverskim aplikacijama, čime se smanjuje trošak i vrijeme izrade programskog proizvoda. Testiranje softvera se definira kao proces vrednovanja programskog proizvoda kako bi se identificirale pogreške, nedostaci ili drugi problemi koji mogu utjecati na funkcionalnost, pouzdanost, performanse i sigurnost programskog proizvoda (Pan, 1999).

U samim začecima testiranja, tijekom ranih 1950-ih godina, fokus testiranja je bio isključivo na otklanjanju pogrešaka. Nije postojao koncept testiranja kao danas, a sam proces testiranja su izvodili programeri. Tijekom pisanja koda, programeri bi uočavali, analizirali i ispravljali probleme. Nakon prve faze, već 1957. godine testiranje se počelo provoditi kao zasebna aktivnost, a vrlo brzo je uočena razlika između otklanjanja pogrešaka i testiranja. Glavni cilj testiranja je bilo osigurati da funkcionalnosti programskih proizvoda budu zadovoljene. Ova faza je trajala sve do 1970-ih kada se testiranje počelo smatrati procesom razbijanja koda u svrhu pronalaženja grešaka u softveru. Inzistiralo se na odvajanju aktivnosti ispravljanja pogrešaka i verifikacije softvera, ali još uvijek nije postojao pristup prevenciji pogrešaka. Slijedeće razdoblje, 1980-ih godina, je bilo orijentirano na evaluaciju i mjerenje kvalitete softvera. Takav pristup je znatno poboljšao pouzdanost i kvalitetu softvera. Početkom 1990-ih godina pojavio se pristup usmjeren na prevenciju pogrešaka i dokazivanje da softver zadovoljava određene specifikacije. U ovoj fazi je bilo ključno identificiranje tehnika testiranja, kao što je tehnika istraživačkog testiranja. Početkom 2000-ih godina pojavljuju se novi koncepti testiranja i alati koji olakšavaju cjelokupan proces testiranja. Pojavom alata za automatizirano testiranje kao što su Selenium i Playwright, događa se velika revolucija u testiranju (Hamilton, 2023).

Testiranje softvera je poprilično skup i zahtijevan proces te na njega najčešće odlazi polovica projektnog budžeta. U samim počecima testiranja, proces osiguravanja kvalitete programskog proizvoda nije bio prepoznat kao ključan proces tijekom izrade softvera te se testiranje ostavljalo za sam kraj projekta. Takav pristup je bio vrlo skup i neučinkovit, a rokovi isporuke softvera se nisu mogli poštovati. Evolucijom razvoja programskih proizvoda, prepoznata je važnost uključenosti testiranja

softvera od samog početka razvoja istog. Razvoj softvera se sastoji od brojnih faza, a ako se greške pronađu odmah u početku, trošak otklanjanja je puno manji nego na samom završetku projekta. Testiranje je neophodno za izradu stabilnog softvera, bez pogrešaka, no i testerima ponekad mogu promaknuti potencijalni nedostaci.

Postoje tri metodologije testiranja: testiranje crne kutije, testiranje bijele kutije i testiranje sive kutije. Testiranje crne kutije se koristi za provjeru funkcionalnosti sustava temeljenih na zahtjevima klijenata. Ovom metodom se ne ulazi u unutarnju strukturu softvera, ne pregledava se programski kod, već se fokusira na funkcionalno testiranje (Nidhra & Dondeti, 2012). To bi značilo da se testiranje provodi s gledišta krajnjeg korisnika. Cilj je utvrditi pogreške u dizajnu korisničkog sučelja, pogreške u ponašanju nekih funkcionalnosti te utvrditi koje funkcionalnosti nedostaju, a navedene su u specifikaciji. Testiranje bijele kutije se koristi za validaciju programskog koda, a najčešće ga izvode programeri. Ovom metodom se provjerava svaka linija koda, kako bi se na vrijeme ispravile pogreške i otkrile potencijalne poteškoće prilikom korištenja softvera (Nidhra & Dondeti, 2012). Pregledom koda utvrđuje se očekivano ponašanje softvera tijekom unosa određenih testnih podataka. Najčešće se metodologija crne kutije i metodologija bijele kutije koriste skupa, čiji je rezultat cjelovit uvid u stanje programskog proizvoda. Upravo se ta kombinacija naziva metodologijom testiranja sive kutije. Ovakva vrsta testiranja se najčešće koristi kada tester ima ograničeno znanje o unutarnjoj strukturi programa. Dakle, tester ima pristup bazi podataka, ograničeni pristup dijelu programskog koda i pristup dizajnu sustava (Murray, 2021). Testiranje sive kutije se koristi kada nije moguće u potpunosti testirati sustav korištenjem samo metodologije testiranja crne kutije.

S obzirom na način izvršavanja testnih slučajeva postoje dvije metode testiranja softvera, ručno i automatizirano testiranje. Ručno testiranje se smatra najstarijom i najrigoroznijom metodom testiranja softvera (Pan, 1999). Ovu vrstu testiranja izvodi tester, koji detaljno ispituje sve dijelove aplikacije prema unaprijed izrađenim testnim slučajevima. Po izvršenju svakog testnog slučaja, tester uspoređuje dobivene rezultate s očekivanim ponašanjem aplikacije te na taj način pronalazi pogreške. Ručno testiranje je vrlo zahtjevna aktivnost koja od testera zahtijeva kreativnost, inovativnost, strpljivost i pomalo perfekcionizam. Iako je ručno testiranje vrlo efektivna metoda za pronalaženje pogrešaka u softveru ona sa sobom nosi i određene nedostatke. Ručno testiranje zahtijeva jako puno vremena za pripremu i izvođenje testnih slučajeva. Budući da sve testne slučajeve izvode testeri potrebna su velika novčana ulaganja u ljudske resurse, a prisutan je i faktor ljudske pogreške. Također je potrebno naglasiti i nemogućnost programiranja kompliciranijih testova („tutorialspoint“, bez dat.). Smatra se da se većina novih pogrešaka pronalazi upravo ručnim testiranjem, no kako bi se smanjili troškovi ljudskih resursa i otklonila mogućnost ljudske pogreške, sve više se koristi automatizirano testiranje. Automatizirano testiranje podrazumijeva izradu testnih skripti koristeći programske jezike

kao što su Python, JavaScript ili TCL, kako bi testne slučajeve mogla izvršavati računala, bez potrebe za intervencijom testera (Albarka & Zhanfang, 2019). Automatizirano testiranje se u potpunosti oslanja na prethodno napisane testne skripte čijim se automatskim izvođenjem dobiveni rezultati uspoređuju s očekivanim ponašanjem aplikacije. Za izradu testnih skripti testeri koriste automatizacijske alate kao što su: Cypress, Loadrunner, TestProject, Playwright… Glavni cilj automatiziranog testiranja jest smanjiti broj testnih slučajeva koji se moraju izvoditi ručno, a da se pritom manualno testiranje ne izbaci u potpunosti, čime se vrijeme testiranja softvera znatno smanjuje (Sharma, 2014). Većina IT poduzeća teži automatiziranom testiranju programskih proizvoda, zbog manjeg troška ljudskih resursa, budući se testovi izvode kroz automatizacijski alat. Ovakva vrsta testiranja je pogodna za izvođenje regresije, jer se isti test može ponavljati neograničen broj puta, a testne skripte se mogu izvršavati na različitim verzijama softvera (Albarka & Zhanfang, 2019). Rezultati testiranja se smatraju jako pouzdanima jer svaki put kada se test izvodi on prati identične testne korake, što znači da nema mjesta pogrešci (Sharma, 2014). Također, vrlo je bitno spomenuti mogućnost izvođenja testnih slučajeva za svaku funkcionalnost u softveru.

I ručno i automatizirano testiranje imaju svoje prednosti i ograničenja, stoga se kao najčešći oblik testiranja bira upravo kombinacija ručnog i automatiziranog testiranja. Ručno testiranje se ne smatra toliko pouzdanim zbog mogućnosti ljudske pogreške, ali treba naglasiti da i alati za automatizaciju također mogu napraviti grešku (Naik & Tripathy, 2008). U tom slučaju je potrebno ponoviti testiranje, ali ručnom metodom, što uzrokuje dodatne troškove.

U ovom radu fokus istraživanja je upravo na usporedbi ručnog i automatiziranog testiranja na primjeru web aplikacije „Swag Labs“ koristeći alat za automatizirano testiranje, Playwright.

1.2. Predmet istraživanja

Iz definiranog problema istraživanja proizlazi predmet istraživanja ovog rada koji se temelji na usporedbi ručnog i automatiziranog testiranja na primjeru web aplikacije „Swag Labs“, dostupne na “https://www.saucedemo.com/“ . To je ogledna web aplikacija, koju je izradila tvrtka „Sauce Labs“ u svrhu učenja ručnog i automatiziranog testiranja. Aplikacija je koncipirana kao fiktivna online trgovina gdje korisnik može dodati stavke u košaricu, izraditi narudžbu, filtrirati proizvode, micati dodane proizvode iz košarice, mijenjati količinu proizvoda… Budući da je „Swag Labs“ ogledna aplikacija, svi selektori koji su potrebni za automatizirano testiranje, kao što su ID i data-test atributi su uredno postavljeni te lako pregledni. Na taj način su testne skripte lako čitljive i razumljive ne samo testeru, već i svima ostalima koji ih čitaju, što je vrlo bitno za znanstveno istraživanje.

Web aplikacija je kompjuterski softver koji se pokreće u Internet pregledniku, radi na temelju klijent-server strukture, a programirana je u HTML-u ili Java Script-u („QA Team“, 2022). U svrhu testiranja, izradit će se testni slučajevi za ručno testiranje i testne skripte za automatizirano testiranje. Pri izradi automatiziranih skripti koristit će se programski jezik Typescript. Dobiveni rezultati testova uspoređivat će se s očekivanim ponašanjem aplikacije kako bi se utvrdilo je li test prošao ili ne, a na kraju će se usporediti rezultati ručnog i automatiziranog testiranja.

Za izradu i pokretanje automatiziranih testnih skripti koristit će se alat Playwright. Taj alat je odabran zato što omogućuje end-to-end (od kraja do kraja) automatizirano testiranje web aplikacija pomoću istog API-ja , na više preglednika kao što su Firefox, Chrome, Safari („Microsoft“, 2023). Na takav način, Playwright zaobilazi ograničenja preglednika kao što su brzina i fleksibilnost, te vrlo brzo provjerava funkcionalnosti odabrane web aplikacije. End-to-end testiranje je metodologija testiranja softvera gdje se provjerava cijela aplikacija, od početka do kraja, simulirajući stvarni korisnički scenarij. „Playwright“ alat je zbog svojih funkcionalnosti trenutno među najkorištenijim alatima za automatizirano testiranje softvera.

O samoj važnosti i potrebi testiranja, svjedoče i brojni primjeri softverskih pogrešaka koje su u povijesti rezultirale ozbiljnim problemima. Slijedeći primjeri softverskih pogrešaka objavljeni su u digitalnom časopisu NewScientist (Adee, 2013).

Ariane 5 – dana 4. lipnja 1996. godine, probno je lansirana raketa Ariane 5, no zbog greške u upravljačkom softveru, samouništenje rakete je pokrenuto trideset sedam sekundi nakon polijetanja.

Knight Capital – Zbog računalnog kvara investicijska tvrtka je u pola sata izgubila pola milijarde dolara. Softverska pogreška je omogućila računalima da kupuju i prodaju dione bez ljudskog nadzora što je uzrokovalo pad cijene dionica kompanije za oko 75% u dva dana.

Terapija zračenjem – 1980-ih godina softverska pogreška u Therac-25 stroju za terapiju zračenjem uzrokovala je smrt pet pacijenata, nakon što su primili ogromnu dozu zračenja. Uzrok tog događaja su bile linije koda koje su se pokušavale sve izvršiti u isto vrijeme.

American Airlines – Nakon pokušaja kombiniranja sustava napisanih u različitim programskim jezicima, dogodila se programska pogreška zbog čega se cijela flota American Airlinesa-a naglo i neočekivano prizemljila.

Navedeni događaji svojom ozbiljnošću situacije kao posljedicom softverske pogreške još jednom potvrđuju da je testiranje vrlo važno za razvitak kvalitetnog softvera, ali i njegovo daljnje održavanje.

1.3. Istraživačka pitanja

Nakon definiranih problema i predmeta istraživanja prikazana su istraživačka pitanja na koja će se odgovoriti na temelju rezultata testiranja web aplikacije „Swag Labs“ odnosno eksperimentalnog istraživanja.

Istraživačka pitanja glase:

IP1: Automatizirano testiranje je brže i efikasnije od ručnog testiranja.

Spoznajom ključnih činjenica o automatiziranom i ručnog testiranju izvodi se prvo istraživačko pitanje da je automatizirano testiranje brže i efikasnije od ručnog testiranja. Dok je za manualno testiranje aplikacije potrebno više testera koji ručno prolaze kroz sve korake testnog slučaja svaki put kada se isti treba izvršiti, za automatizirano testiranje je dovoljan minimalan broj testera kojima je zadatak pokrenuti testove i pričekati rezultate, upravo zato što je svaki testni korak automatiziran („tutorialspoint“, bez dat.).Može se reći da automatizirano testiranje omogućuje brže povratne informacije, što povećava produktivnost projektnog tima („tutorialspoint“, bez dat.). Automatizacija se ne isplati na malim projektima kratkog vremenskog opsega, jer je ipak potrebno vremena za implementaciju alata za automatizirano testiranje i izradu testnih skripti. Za takve projekte se koristi isključivo manualno testiranje. Kod većih projekata je automatizacija vrlo korisna jer se osim uštede vremena testeri jednostavno mogu izmjenjivati na projektu. To je upravo zbog toga što se jednom napisani testovi sami izvršavaju pa nije potrebno dugotrajno upoznavanje novih testera sa starijim testnim slučajevima.

IP 2: Ručno testiranje je korisnije nego automatizirano prilikom testiranja korisničkog iskustva.

Automatizirano testiranje ne uključuje ljudsko razmatranje zbog čega nikada ne može u potpunosti jamčiti pozitivno korisničko iskustvo. S druge strane, ručno testiranje softvera omogućuje ljudsko promatranje što je vrlo korisno za prilagodbu sustava korisniku (Bezsmolna, 2019). Prije automatiziranog testiranja vrlo je bitno softver prvo testirati manualno kako bi se otklonila mogućnost prividnog prolaska testa koji zapravo nije u skladu s korisničkim zahtjevima te prema tome ne bi trebao prolaziti. Na primjer, kod automatizacije je moguće postaviti test koji ispituje je li određeni element vidljiv. Moguće je da takav test prođe, ali da taj element osim što je vidljiv ne bude u skladu s korisničkim zahtjevom kao što je na primjer razlika u obliku ili boji elementa što je uočeno prilikom ručnog testiranja.

IP 3: Kombinacija ručnog i automatiziranog testiranja najefektivniji je način osiguravanja kvalitete programskog proizvoda.

Analizom prednosti i nedostataka ručnog i automatiziranog testiranja softvera postavlja se istraživačko pitanje da je kombinacija ručnog i automatiziranog testiranja najefektivniji način osiguranja kvalitete programskog proizvoda. Glavne odlike ručnog testiranja su te što je pogodno za istraživačko testiranje i za testiranje malih promjena u softveru, no isto tako se smatra manje pouzdanim od automatiziranog testiranja jer postoji faktor ljudske pogreške (Bezsmolna, 2019). Automatizirano testiranje u puno kraćem vremenu može pronaći puno više pogrešaka nego tester, pouzdanije je, može se odvijati konstantno, ali je problem u tome što je teško dobiti uvid u vizualne elemente aplikacije (Sabev & Grigorova, 2015). Pomoću automatizacije je moguće otkriti ako je određeni element minimalno pomaknut van okvira u kojem treba biti, kao što je na primjer pomak od jednog piksela koje ljudsko oko ne može primijetiti.

1.4. Ciljevi istraživanja

Glavni cilj istraživanja ovog diplomskog rada je analizirati učinkovitost ručnog i automatiziranog testiranja na primjeru web aplikacije „Swag Labs“.

Sporedni ciljevi istraživanja su:

- Analizirati i definirati pojmove ručnog i automatiziranog testiranja, vrste metodologija u testiranju i tehnike testiranja
- Prezentirati automatizirane skripte kroz alat Playwright
- Prezentirati shemu testnih slučajeva za manualno testiranje
- Uvidjeti važnost testiranja softvera

1.5. Metodologija istraživanja

U empirijskom dijelu diplomskog rada provodit će se eksperimentalna metoda i metoda studije slučaja.

Eksperimentalna metoda se temelji na promatranju pojave koja se ispituje pod točno određenim uvjetima, a ona se svaki put uz ponavljanje tih istih uvjeta može ponovno izazvati (Zelenika , 2000). Može se reći da je ova metoda jedna od najosnovnijih i najvažnijih metoda teorijskog i praktičnog dijela znanstvenog rada. Osnovni činitelji eksperimentalne metode su: eksperimentator koji praktično izvodi eksperiment, predmet eksperimenta što predstavlja pojave i procese bilo koje vrste, a u ovom slučaju je to ručno i automatizirano testiranje na web aplikaciji, zatim sredstva i oruđe eksperimenta pomoću kojih se obavlja eksperiment, u ovom slučaju to bi bili testni slučajevi za ručno i automatizirano testiranje i naravno alat Playwright, zatim sam eksperimentalni postupak što je u ovom slučaju testiranje web aplikacije, nakon toga slijedi provjeravanje određenih istraživačkih pitanja, daje se odgovor na postavljena istraživačka pitanja u radu na temelju rezultata testiranja, te se na kraju prema svim podacima određuje stvarni rezultat eksperimenta i izvodi objašnjenje (Zelenika, 2000). Eksperimentalna metoda se često upotrebljava u kombinaciji s drugim metodama kao što će se u ovome radu upotrebljavati u kombinaciji s metodom studije slučaja.

Metoda studije slučaja je kvalitativna metoda istraživanja koja se koristi za prikupljanje velikog broja podataka koji su rezultat proučavanja određenog slučaja. Studija slučaja uključuje opisivanje jednog ili nekoliko specifičnih slučajeva određenih karakteristika, koji se detaljno analiziraju kako bi se iz njih izvukli relevantni zaključci (Toth & Metzinger, 2020). Ova metoda je vrlo korisna kada se određene pojave želi dublje istražiti. Kako bi se osigurala pouzdanost prikupljenih podataka te izbjegle moguće pogreške važno je pratiti određena pravila prilikom analize rezultata (Toth & Metzinger, 2020). U ovom slučaju promatrat će se rezultati testnih slučajeva ručnog i automatiziranog testiranja na primjeru web aplikacije „Swag Labs“.

U teorijskom dijelu rada koristit će se slijedeće metode znanstveno-istraživačkog rada (Zelenika, 2000):

1. Induktivna metoda – je metoda induktivnog načina zaključivanja u kojem se na temelju analize pojedinačnih činjenica dolazi do zaključka o općem sudu, od zapažanja pojedinačnih slučajeva dolazi se do općih zaključaka. U ovom radu polazit će se od općih spoznaja o testiranju programskih proizvoda, ručnom i automatiziranom testiranju kako bi se došlo do spoznaje kolika je učinkovitost ručnog i automatiziranog testiranja prilikom testiranja web aplikacije.

2. Deduktivna metoda – je metoda u kojoj se iz općih stavova izvode pojedinačni zaključci.

3. Metoda analize – je metoda raščlanjivanja složenih tvorevina na jednostavnije elemente i izučavanje svakog posebno.

4. Metoda sinteze – je metoda sastavljanja jednostavnih, izdvojenih elemenata.

5. Metoda kompilacije – je postupak preuzimanja tuđih rezultata znanstveno–istraživačkog rada, odnosno tuđih opažanja, stavova, zaključaka i spoznaja uz citiranje i navođenje preuzetih dijelova.

6. Metoda deskripcije – je postupak očitavanja i navođenja činjenica.

7. Metoda dokazivanja – je jedna od najvažnijih znanstvenih metoda, a temelji se na izvođenju istinitosti pojedinih stavova na temelju znanstvenih činjenica ili ranije utvrđenih istinitih stavova. Svrha postupka je utvrđivanje točnosti spoznaje.

1.6. Doprinos diplomskog rada

Znanstveni doprinos ovog diplomskog rada se ogleda u analizi rezultata ručnog i automatiziranog testiranja web aplikacije „Swag Labs“ te procjeni učinkovitosti navedenih pristupa usporedbom dobivenih rezultata. Provođenjem ručnog i automatiziranog testiranja na istoj web aplikaciji istaknut će se prednosti i ograničenja obje tehnike testiranja. Na takav način istaknut će se važnost kombinacije ručnog i automatiziranog testiranja kako bi se poboljšala ukupna kvaliteta i pouzdanost programskih proizvoda te maksimizirale prednosti svake od metoda. To može biti korisno tvrtkama za razvoj softvera koje se suočavaju s izazovima kombinacije tehnika testiranja softvera kako bi ostali unutar vremenskih i financijskih ograničenja.

U radu će se dati uvid i u izazove i ograničenja alata za automatizirano testiranje, a posebno alata Playwright koji će se konkretno koristiti u ovom diplomskom radu. Ovakav pristup procjene učinkovitosti alata Playwright može koristiti tvrtkama za razvoj učinkovitijih alata za automatizirano testiranje.

Može se reći da ovaj diplomski rad može značajno pridonijeti području softverskog inženjerstva analizirajući razumijevanje najboljih praksi testiranja softvera i alata za automatizirano testiranje.

1.7. Struktura diplomskog rada

Diplomski rad će se sastojati od ukupno trinaest poglavlja.

U prvom poglavlju će se dati pregled cijelog diplomskog rada. Prvo će se pojasniti problem i predmet istraživanja, zatim temeljna istraživačka pitanja i ciljevi istraživanja, prikazat će se metodologije istraživanja koje će se koristiti u radu te će se ukazati na to da je testiranje najvažnija faza tijekom razvoja programskog proizvoda.

U drugom poglavlju će se prikazati pregled literature.

U trećem poglavlju će se pojasniti terminologija vezana uz testiranje softvera kao što je testni slučaj, softverska pogreška, izvješće o pogreškama. Nakon toga će se navesti glavni ciljevi testiranja softverskog proizvoda te načela testiranja čime će se ukazati na važnost testiranja programskih proizvoda.

U četvrtom poglavlju će se dati pregled razvoja testiranja programskih proizvoda kroz povijest. Evolucija testiranja će se prikazati po najvažnijim erama i pristupima testiranja koji su obilježili te godine.

U petom poglavlju će se pojasniti životni ciklus razvoja programskih proizvoda. Svaka faza razvoja će biti detaljno prikazana, a osim toga dat će se uvid i u nekoliko vrsta modela razvoja programskog proizvoda.

U šestom poglavlju, životni ciklus testiranja programskih proizvoda će se prikazati kroz šest faza.

U sedmom poglavlju će se dati definicije, prednosti i nedostaci ručnog i automatiziranog testiranja. Na kraju poglavlja napravit će se usporedba ove dvije vrste testiranja.

U osmom poglavlju će se detaljno objasniti tri glavne metodologije testiranja programskih proizvoda, metodologija sive kutije, metodologija bijele kutije i metodologija crne kutije. Također će se predstavit deset razina testiranja programskih proizvoda. Svaku od razina će se detaljno objasniti te naglasiti koje razine se koriste u empirijskom dijelu diplomskog rada.

U devetom poglavlju će se dati pregled najpopularnijih alata za ručno i automatizirano testiranje, a poseban naglasak će se staviti na alat za automatizaciju „Playwright“ koji će biti detaljno objašnjen.

U desetom poglavlju će se ukazati na različite vrste aplikacija, web aplikacije, nativne i hibridne aplikacije, s posebnim naglaskom na web aplikacijama jer se upravo na toj vrsti aplikacije provodi istraživanje u ovom diplomskom radu.

U jedanaestom poglavlju će biti predstavljeno istraživanje, odnosno testni slučajevi za ručno i automatizirano testiranja web aplikacije „Swag Labs“. Nakon toga će se prikazati rezultati svakog od testiranja i na kraju će se dobiveni rezultati međusobno usporediti. Na temelju dobivenih rezultata dati će se odgovor na istraživačka pitanja.

U dvanaestom poglavlju prikazat će se sva moguća ograničenja istraživanja.

Posljednji dio rada se sastoji od zaključka, popisa slika, popisa grafova, literature te sažetka na hrvatskom i engleskom jeziku.

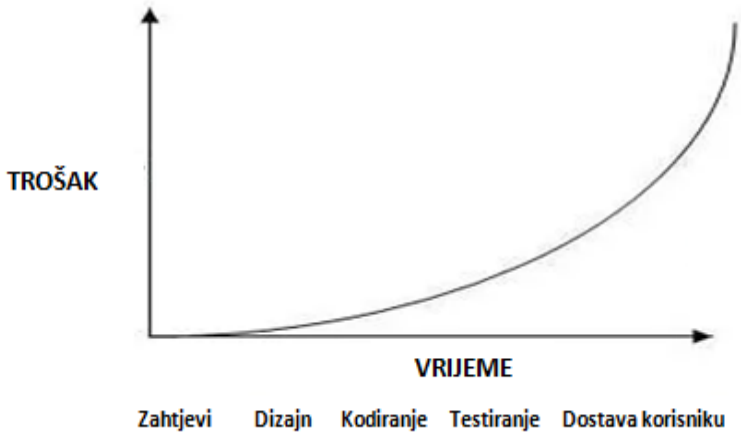
2. PREGLED LITERATURE

Pan (1999) u svom istraživanju „Software Testing“ daje slijedeću definiciju testiranja softvera „Testiranje softvera se definira kao proces vrednovanja programskog proizvoda kako bi se identificirale pogreške, nedostaci ili drugi problemi koji mogu utjecati na funkcionalnost, pouzdanost, performanse i sigurnost programskog proizvoda“. U to vrijeme se testiranje promatralo kao funkcija otklanjanja pogrešaka u softveru, a ne kao nešto što je nužno za osiguranje kvalitetnog programskog proizvoda koji zadovoljava korisničke zahtjeve klijenta. Magner (2005) u svojoj knjizi „Softversko Inženjerstvo“ ističe upravo tu važnost usklađenosti softvera s korisničkim zahtjevima u smislu osiguranja kvalitete softvera te predstavlja agilne i tradicionalne metodologije razvoja softvera. On daje nešto moderniju definiciju softvera od Pan-a (1999) u kojoj ističe da testiranje osim otkrivanju pogrešaka, služi i verifikaciji i validaciji samog softvera kako bi se potvrdilo da je programski proizvod u skladu s poslovnim, tehničkim i korisničkim zahtjevima. S pojavom modernijih načina razvoja softvera, pojavljuju se i nove tehnike testiranja softvera čiji dublji pregled daje Myers (2004) u svojoj knjizi „The Art of Software Testing“. On pojašnjava različite metode i pristupe testiranju te izvodi načela testiranja softvera kojima se potrebno voditi kako bi se testiranje uspješno obavilo. Osim toga, naglašava da je dobar testni slučaj onaj koji ima veliku mogućnost pronalaženja još uvijek neotkrivenih pogrešaka, a to se postiže samo tako da se u proces testiranja softvera kreće s mišlju da je softver pun pogrešaka. Slična razmišljanja iznose i Naik Tripathy (2008) u knjizi „Software Testing and Quality Assurance, Theory and Practice“ gdje daju detaljan pregled ručnog i automatiziranog testiranja te različitih razina i tehnika testiranja. Navedena knjiga objašnjava sve temeljne koncepte testiranja počevši od definicije pogreške i testnog slučaja pa sve do rubnih tehnika testiranja. Osim navedenih autora potrebno je spomenuti i autore Spillner i Schaefer (2014) koji u knjizi „Software Testing Foundations“ prikazuju slična razmišljanja kao Naik i Tripathy, ali s većim naglaskom na testnu dokumentaciju i primjere iste. Knott (2015) u svojoj knjizi „Hands On Mobile App Testing“ objašnjava koncepte testiranja u okviru mobilnih aplikacija koje dijeli na web, nativne i hibridne. Pojašnjava načine testiranja mobilnih aplikacija te navodu primjere korisnih alata za automatizirano testiranje. Za razliku od prijašnjih autora, on ističe i važnost određenih vještina koje tester mora imati kako bi uspješno izvodio sve postojeće tehnike testiranja softvera. S navedenim autorom je zaokružen pregled najvažnijih izvora koji će se koristiti u diplomskom radu.

3. VAŽNOST TESTIRANJA PROGRAMSKOG PROIZVODA

Testiranje programskog proizvoda je ključan korak u životnom ciklusu razvoja programskog proizvoda. Kako bi softver bio kvalitetno izrađen prema svim specifikacijskim kriterijima, potrebno je kontinuirano praćenje, analiziranje i ispravljanje pogrešaka tijekom cijelog procesa razvoja softvera. Važnost testiranja se ogleda i u zahtjevima koje klijenti imaju od samog softvera, a to je prvenstveno pouzdanost i sigurnost, mogućnost mijenjanja softvera u skladu s potrebama korisnika, efikasnost i upotrebljivost, što znači da softver mora imati zadovoljavajuće korisničko sučelje i performanse. (Magner, 2005).

Osim zadovoljstva korisnika i pozitivnih povratnih informacija o proizvodu, pravovremenim testiranjem otkrivaju se pogreške koje se u ranim fazama razvoja softvera mogu riješiti efikasnije i jednostavnije nego u završnoj fazi razvoja softvera. Ako se testiranje ostavi za sam kraj, onda će trošak otklanjanja pogrešaka biti puno veći, a samim uklanjanjem pogreške u tako kasnoj fazi može rezultirati narušavanjem stabilnosti ostatka koda (Johnson, 2016). Samim time se produžuje vrijeme isporuke, potrebno je ponovno testiranje cijele aplikacije, a trošak projekta se znatno povećava. Dakle, sve greške koje se pronađu u softveru predstavljaju trošak, jer je svaku od tih pogrešaka potrebno ispraviti, bez obzira na to radi li se o pogrešci u dizajnu ili o nemogućnosti prijave u aplikaciju. Slika 3.1. prikazuje trošak popravljanja pogrešaka po fazama razvoja softvera. Može se uočiti kako je trošak popravljanja pogrešaka u završnoj fazi razvoja najveći, dok je u početnoj fazi razvoj najmanji.



Slika 3.1. Trošak popravljanja pogrešaka u softveru
Izvor: https://www.testing.guru/what-is-the-cost-of-defects-in-software-testing/

Testiranje softvera je neophodno za stabilno funkcioniranje aplikacije jer se greške tijekom razvoja softvera mogu potkrasti bilo kojem članu razvojnog tima. Na primjer dizajneri mogu pogriješiti prilikom

dizajniranja sučelja aplikacije, a takvu greško je poželjno testiranjem otkriti i otkloniti odmah u početku, jer će kasnije programeri nastaviti raditi po krivom dizajnu, a trošak uklanjanja pogrešaka će se samo povećavati. Najčešći razlog zbog kojeg nastaju pogreške u softveru su pogreške programera prilikom implementacije softvera (Everett & Mcleod, 2007). Uzroci tome mogu biti manjak iskustva programera, nedovoljno poznavanje određenog programskog jezika, faktor ljudske pogreške ili nedovoljno jasno definirani zahtjevi od strane klijenta.

Važnost testiranja softvera se često zanemaruje jer je to dugotrajan i skup proces. Sam proces testiranja se zbog toga često prepušta programerima, što i nije najbolja opcija jer se testiranje može obaviti površno. Odnosno, programeri će najčešće testirati samo ispravnost koda, a zanemariti testiranje dizajna aplikacije i korisničkog iskustva čime ostaje puno skrivenih pogrešaka koje će klijent vrlo brzo uočiti.

Budući da je testiranje poprilično skup proces, u pojedinim industrijama je dopušteno na tržište dostaviti proizvod koji u sebi ipak sadrži određene pogreške. To je dopušteno u slučaju da neke pogreške nije isplativo popravljati, a ne utječu na stabilnost i pouzdanost aplikacije. S druge strane, postoje industrije u kojima je vrlo bitno da softver ne sadrži niti jednu pogrešku jer o tome mogu ovisiti ljudski životi ili može doći do velikih novčanih gubitaka. Na primjer, 2021. godine se zbog softverske pogreške u sustavu zaštite podataka T-Mobile-a dogodio hakerski napad čija je posljedica bilo curenje osobnih podataka pedeset milijuna korisnika. Iako se direktor T-Mobile-a ispričao zbog navedenog događaja i napomenuo kako će pojačati sustav zaštite podataka, velik broj korisnika je prešao na drugog tele-operatora, a neki korisnici su čak podigli tužbu protiv T-Mobile-a (Stojmanovska, 2021). Ova softverska pogreška je uzrokovala milijunske gubitke s čijim će se posljedicama operater morati nositi još godinama. Još jedan primjer značajne softverske pogreške u 2021. godini odnosi se na Tesla automobile. Nakon što je tvrtka počela primati povratne informacije od kupaca da „Full-Self Driving“ softver lažno prepoznaje prijetnje sudara što uzrokuje aktiviranje naglog zaustavljanja vozila, otkrili su komunikacijsku pogrešku u softveru. Ta pogreška je uzrokovala lažno upozorenje na sudar, što je automatski aktiviralo sustav za naglo zaustavljanje automobila, čime se u opasnost dovode svi putnici. Tesla je odmah objavio izvješće o sigurnosnom opozivu 12 000 automobila što je uzrokovalo milijunske gubitke. Nakon toga, tvrtka je izdala rješenje za rješavanje pogreške i ažuriranje softvera (Stojmanovska, 2021).

Navedeni primjeri još jednom potvrđuju važnost testiranja softvera od samog početka razvoja softvera, jer ako na tržištu završi nesiguran i loš proizvod posljedice mogu biti fatalne. Potrebno je naglasiti da testiranje omogućuje povratne informacije o kvaliteti softvera te bolje razumijevanje softvera, što u konačnici rezultira kvalitetnim i pouzdanim proizvodom.

3.1. Ciljevi testiranja programskog proizvoda

Testiranje softvera se definira kao pokusno pokretanje softvera ili samo njegovih dijelova, na umjetno izrađenim podacima, uz pomno analiziranje rezultata (Magner, 2005).Glavni cilj testiranja je otkrivanje pogrešaka i slabosti softvera u svrhu postizanja što veće kvalitete istog. Osim otkrivanja pogrešaka, testiranje služi verifikaciji i validaciji samog softvera kako bi se potvrdilo da je programski proizvod u skladu s poslovnim, tehničkim i korisničkim zahtjevima. Prilikom testiranja je vrlo bitno pokušati dokazati i reproducirati što više pogrešaka, jer će se na taj način softver najbolje test. S druge strane, ako se softver testira s mišlju da isti nema niti jedne pogreške, tada se tester neće niti potruditi prouzrokovati i pronaći pogreške u radu softvera, a testiranje neće biti kvalitetno odrađeno. Kako bi se testiranje obavilo što kvalitetnije, potrebno je težiti nekoliko glavnih ciljeva kao što su (Qaudri & Farooq, 2010):

1. Verifikacija i Validacija – testiranje softvera nije samo pronalaženje pogrešaka, već je to kontrola kvalitete programskog proizvoda u smislu da zadovoljava sve kriterije tehničke specifikacije i potrebe korisnika. Testiranje pruža povratne informacije o trenutnom statusu softvera u usporedbi sa zahtjevima klijentima čime se utvrđuje ispunjava li softver određenje zahtjeve koji su potrebni za puštanje na tržište. Dakle, verifikacijom se provjerava radi li softver prema specifikacijskim zahtjevima, a validacijom se testira je li softver zadovoljava potrebe korisnika.

2. Prioritetno testiranje – iscrpno testiranje čitavog softvera je jednostavno nemoguće, stoga je testiranje potrebno provoditi efikasno, efektivno i unutar vremenskih i novčanih ograničenja. Stoga je prilikom testiranja potrebno odrediti prioritete i razumno raspodijeliti resurse. Glavne funkcionalnosti softvera koje su potrebne za očekivani rad istog potrebno je testirati više puta, s različitim vrijednostima podataka, dok je sve ostale funkcionalnosti potrebno testirati barem jednom.

3. Uravnoteženost – prilikom testiranja potrebno je podjednako obratiti pažnju na specifikacijske zahtjeve, tehnička ograničenja softvera i očekivanja korisnika. Rezultati testiranja moraju biti ponovljivi, neovisno o tome tko testira softver. Tester tijekom testiranja mora biti dosljedan i nepristran te uzeti u obzir sve zahtjeve korisnika kako bi softver ispunio sve potrebne kriterije.

4. Dokumentiranje – kako bi se proces testiranja što više olakšao potrebno je dokumentirati sve testne slučajeve i njihove rezultate, bili oni uspješni ili ne. Testni plan sadrži sve testne slučajeve koji su raspisani vrlo detaljno i jasno kako bi svatko tko čita testnu dokumentaciju mogao utvrditi o čemu se radi. Na takav način se izbjegava suvišno testiranje funkcionalnosti, a pogreške se mogu vrlo lako prevenirati.

5. Deterministički pristup – otkrivanje pogrešaka i problema u softveru ne smije se provoditi nasumično, već ciljano, određenim redoslijedom. Na takav način niti jedan dio softver neće ostati izostavljen iz procesa testiranja te se omogućuje bolja procjena troškova uklanjanja pogrešaka.

3.2. Načela testiranja programskog proizvoda

Myers (2004) ističe nekoliko načela kojima se potrebno voditi prilikom testiranja softvera, a to su:

1. Prilikom pisanja testnog slučaja nužno je odmah definirati očekivani output odnosno rezultat.

Navedeno načelo uzrok je jedne od najčešćih pogrešaka prilikom testiranja softvera upravo zato što se često zanemaruje. Dakle, ako rezultat testnog slučaja nije unaprijed precizno definiran najčešće će se pogrešan rezultat protumačiti kao točan, zbog podsvjesne želje testera da prikaže točan rezultat. Kako bi se navedeno izbjeglo potrebno je unaprijed definirati očekivane rezultate softvera. Zbog toga testni slučaj mora biti napisan u dva dijela gdje se u prvom navode ulazni podaci za testiranje softvera, a u drugom dijelu se navodi precizan opis očekivanih izlaznih podataka za određeni ulazni skup podataka.

2. Programer ne bi trebao testirati program koji je on sam izradio.

Kod testiranja softvera od strane programera koji ga je izradio često se javlja problem da programer ne želi pronaći greške u vlastitom kodu. Nakon što je programer konstruktivno kodirao softver, jako je teško promijeniti fokus gledišta na taj isti softver u smislu da ga promatra destruktivno. Smatra se da većina programera ne može učinkovito testirati svoj kod zbog podsvjesne želje da izbjegnu pogreške iz straha od nadređenog ili klijenta. Kao drugi problem navodi se mogućnost programerovog pogrešnog razumijevanja specifikacije, što otvara mogućnost prenošenja istog tog nesporazuma u testove softvera. Učinkovitije je kada softver testira netko drugi, no programeri su neophodni prilikom ocjenjivanja specifikacije softvera i samog koda.

3. Programerski tim ne bi trebao testirati vlastite softvere.

Argumenti su slični onima iz prethodnog načela, no ovdje je glavna razlika to što se programerski tim ocjenjuje prema sposobnosti dostave pouzdanog programa, do određenog datuma uz procijenjene troškove. Upravo zato je programerskom timu teško biti objektivan prilikom testiranja softvera, jer ako se u softveru pronađe puno pogrešaka to se može smatrati smanjenjem vjerojatnosti ispunjavanja zadanih ciljeva i rokova uz povećanje troška projekta. Dakle, iako programerski tim može testirati svoje softvere s određenim stupnjem uspjeha, smatra se da je uspješnije i ekonomičnije testiranje softvera od strane objektivnih i neovisnih suradnika.

4. U svakom krugu testiranja softvera potrebno je analizirati rezultate svakog testa.

Iako je navedeno načelo vrlo jasno, ono se često zanemaruje pa se neke vidljive pogreške nerijetko izostave iz daljnjih testiranja. Dakle, pogreške koje se pronađu u kasnijim testovima bile su izostavljene u rezultatima ranijih testova.

5. Testni slučajevi moraju biti napisani za podatke koji nisu valjani ili su neočekivani, kao i za podatke koji su valjani i očekivani.

Prilikom testiranja softvera postoji tendencija ka koncentriranju isključivo na očekivane rezultate softvera, zanemarujući neočekivane rezultate i one koji nisu validni. Smatra se da se najviše pogrešaka u softveru otkrije upravo kada se softver koristi ne neočekivani način. Upravo zato je vrlo bitno sastaviti testne slučajeve i za podatke koji nisu očekivani ili nemaju validne ulazne podatke.

6. Testiranje softvera kako bi se utvrdilo da softver ne radi ono što je očekivano je tek pola posla, drugi dio posla se odnosi na testiranje softvera na način da se provjerava radi li softver ono što nije očekivano.

Navedeno načelo se nastavlja na prethodno načelo. Dakle, softveri se moraju testirati na način da se testira očekivano ponašanje i na način da se testira neželjeno ponašanje. Samo na takav način će se dobiti stabilan i pouzdan softver.

7. Potrebno je izbjegavanje brisanja dijelova testne dokumentacije, osim ako nije sigurno da se taj dio softvera neće realizirati.

Testni slučajevi su vrlo vrijedna dokumentacija jer prate razvoj softvera od samog početka, pa tako i sve pogreške koje su se ikad pojavile u softveru. Kad god se softver mora ponovno testirati zbog novih funkcionalnosti, već postojeći testovi se ponovno izvršavaju, a novi testovi se nadopisuju. Spremanje testnih slučajeva i njihovo ponovno pokretanje nakon promjena na nekim drugim funkcionalnostima naziva se regresijsko testiranje.

8. Testni slučajevi se ne bi smjeli pisati s pretpostavkom da softver ne sadrži nikakve pogreške.

Ako se kreće s pretpostavkom da je testiranje proces pokretanja softvera kako bi se prikazalo da isti radi ispravno, pogreške će ostati skrivene te će u kasnijim fazama razvoja softvera predstavljati problem. Dakle, potrebno je testirati s namjerom pronalaženja pogrešaka. Čak i nakon iscrpno provedenog testiranja uvijek će postojati još neke pogreške koje nisu pronađene.

9. Vjerojatnost pronalaska dodatnih pogrešaka u određenom dijelu softvera, proporcionalna je broju već pronađenih pogrešaka u tom dijelu softvera.

Ovaj fenomen može se objasniti na način da se pogreške pojavljuju u klasterima, odnosno da se više pojavljuju u određenim dijelovima softvera. Takva vrsta povratne informacije je testeru vrlo korisna jer mu omogućuje da testiranje koncentrira na dijelove softvera koja ima veću sklonost ka pogreškama. Navedeno načelo prikazano je na slici 3.2.1.

Slika 3.2.1. Proporcionalnost već pronađenih pogrešaka u softveru i onih koje još nisu pronađene
Izvor: Mayers, G. (2004.): The Art of Software Testing, Word Association, Inc, New Jersey

10. Testiranje je iznimno kreativan i intelektualno zahtijevan zadatak.

Smatra se da je za uspješno testiranje softvera osim tehničkog znanja potrebna i doza kreativnosti jer softver nikad nije moguće dovoljno testirati da jamči nepostojanje pogrešaka.

4. EVOLUCIJA TESTIRANJA PROGRAMSKIH PROIZVODA KROZ POVIJEST

Izraz „bug“ se prvi put spominje već 1870-ih godina od strane američkog izumitelja Thomasa Alve Edisona. Prema Yaleovoj knjizi citata, Edison koristi izraz „bug“ u pismu Theodoru Puskasu, kako bi objasnio određenu pogrešku u sustavu. U to vrijeme se izraz „bug“ svakodnevno koristio tijekom rasprava o pogreškama u sustavu (Meerts, bez dat.).

1947. godine, znanstvenica sa sveučilišta Harvard Grace Murray je radeći na računalu Mark II, otkrila da se moljac zaglavio između prekidača računala zbog čega se kontakt nije mogao uspostaviti. Taj događaj je zapisala u svom radnom dnevniku te opisala moljca kao „bug“ koji je uzrokovao pogrešku u radu računala. Otklanjanje moljca iz računala nazvala je „debugiranje“, odnosno uklanjanje pogreške. U to vrijeme su računala zauzimala jako puno prostora pa su i testovi bili usmjereni na hardver (Hernandez, 2019).

U razdoblju od 1947. do 1956. godine, poznatom kao „Razdoblje otklanjanja pogrešaka“, pojam „debugiranje“ (eng. debugging) se povezivao sa poduzimanjem korektivnih mjera kako bi program mogao nastaviti raditi. U ovoj fazi nije bilo razlike između testiranja i otklanjanja pogrešaka, a fokus je bio isključivo na ispravljanju pogrešaka. Koncept testiranja ili testera je bio potpuna nepoznanica, a programeri su sami ispravljali pogreške u kodu (Hernandez, 2019).

Alan Turing 1949. godine piše rad u kojem daje odgovor na pitanje kako provjeriti je li rutina testiranja softvera uistinu ispravna. To je bio njegov prvi rad o provođenju testiranja na nekom softveru. Nakon toga, 1950. godine, Turing piše svoj drugi rad pod nazivom “Computing Machinery and Intelligence” gdje predstavlja Turingov test. Turingov test služio je za dokazivanje inteligencije strojeva. Test se sastojao od tri uloge: čovjek, stroj i ispitivač. Ispitivač unosi pitanja u stroj te na temelju odgovora koje dobiva mora zaključiti je li taj odgovor dao stroj ili čovjek. Cilj stroja je da natjera ispitivača da pogrešno zaključi tko je zapravo dao odgovor, stroj ili čovjek, a cilj čovjeka je pokušati pomoći ispitivaču da zaključi u kojem slučaju je stroj dao odgovor. Na kraju ispitivanja ispitivač mora donijeti odluku tko je stroj, a tko čovjek. Ako je stroj toliko uvjerljivo odgovarao na pitanja, da ga ispitivač nije uspio prepoznati, to znači da je stroj prošao Turingov test i pokazao inteligenciju. Ovaj test je imao veliki utjecaj na polje razvoja umjetne inteligencije (Graham & David, 2021).

1957. godine Charles L. Baker u svojem osvrtu na knjigu „Computer Programming“ autora Dana McCrackena ističe jasnu razliku između testiranja programa i otklanjanja pogrešaka. Baker objašnjava potrebu za razvojem testova jer se jedino tako može osigurati da softver zadovoljava unaprijed definirane specifikacije zahtjeve i funkcionalnost programa. S vremenom su se počele razvijati sve složenije aplikacije, a troškovi rješavanja nedostataka u takvim aplikacijama utjecali su na rizik za

profitabilnost projekta. Stoga je razvoj testova postajao sve važniji. Po prvi put se kvaliteta programskog proizvoda počela povezivati sa rezultatima tijekom testiranja, stoga je poseban fokus stavljen na povećanje broja testnih slučajeva i kvalitete testiranja od samog početka razvoja softvera (Meerts, bez dat.).

Razdoblje od 1957. godine do 1978. godine naziva se „Doba orijentirano na demonstracije“. U ovom razdoblju glavni cilj je bio dokazati da programski proizvod radi onako kako je očekivano prema specifikaciji i zahtjevima klijenta. Budući da je u ovom razdoblju povućena jasna razlika između otklanjanja pogrešaka i testiranja, testiranje se provodilo kao zasebna aktivnost. Još jedan zaključak koji proizlazi iz ove ere je da se s povećanjem testiranja programskog proizvoda povećava vjerojatnost pronalaska greške, odnosno opada funkcija softvera. U ovom razdoblju se nije prakticiralo negativno testiranje (Meerts, bez dat.).

1979. godine Glenford J. Myers predstavlja knjigu „The Art of Software Testing“ što je bila prva knjiga koja je govorila isključivo o testiranju softvera. On tada iznosi definiciju testiranja softvera kojom je radikalno promijenio način otkrivanja pogrešaka u softveru, a ona glasi: „Testiranje softvera je proces pokretanja programa s namjerom pronalaženje grešaka“. Myers je smatrao da ako se softver nastoji prikazati kao besprijekoran, da će tester odabrati ulazne podatke za koje je mala vjerojatnost da će uzrokovati pogreške u programu. Myers je bio prvi koji je uspio odvojiti proces uklanjanja pogrešaka od testiranja (Meerts, bez dat.). Tijekom ove faze pozornost je bila usmjerena samo na testiranje kvarova, bez pristupa prevencije kvarova. Smatralo se da je uspješan testni slučaj samo onaj koji uspije pronaći još jednu pogrešku u softveru. Testeri bi testirali softver na način da ga pokvare, odnosno unište, no takav pristup nije zaživio jer se niti jedan softver na taj način nikad ne bi objavio. Ova era je trajala od 1979. godine do 1982. godine, a naziva se: „Era usmjerena na destrukciju programskog proizvoda“ (Hernandez, 2019).

Razdoblje od 1983. do 1987. godine naziva se „Era orijentirana na evaluaciju“, a glavni cilj je bio mjerenje kvalitete softvera. Uočeno je da testiranje poboljšava pouzdanost softvera, stoga je uvedeno da testeri trebaju testirati programski proizvod sve dok se broj otkrivenih pogrešaka ne smanji na odgovarajuću razinu. Testiranje je prepoznato kao neophodna faza tijekom razvoja softvera, a njegova učinkovitost se posebno ističe pojavom prvih alata za automatizirano testiranje. Vrlo je bitno istaknuti da se u ovom razdoblju, 1984. godine organizira prva međunarodna konferencija o testiranju programskih proizvoda ikad (Meerts, bez dat.).

Razdoblje od 1988. do 2000. godine naziva se „Era usmjerena na prevenciju“, a tijekom tog razdoblja pojavljuje se novi pristup s testovima koji su usmjereni na dokazivanje da softver zadovoljava sve prethodno dogovorene specifikacije, pronalaženje pogrešaka i prevenciju nedostataka. Kod se dijelio

na testirajući i netestirajući, a testirajući kod je naravno imao manje pogrešaka od netestirajućeg koda koji je bio vrlo nestabilan. Godine 1991. Hetzel daje definiciju testiranja u kojoj napominje da je testiranje u osnovi planiranje, izgradnja, održavanje i izvođenje testova i testnih okruženja. U ovoj fazi fokus je stavljen na identificiranje tehnika testiranja. Tada se pojavljuje i pojam istraživačko testiranje, a prvi put ga spominje Cem Kaner u svojoj knjizi „Testiranje računalnih programa“. Može se reći da je to prvi put da se prikazuju stvarni problemi u testiranju, čime se postavlja novi standard testiranja (Meerts, bez dat.).

Početkom 2000-ih godina razvili su se novi načini testiranja softvera, kao što su razvoj vođen testiranjem i razvoj vođen ponašanjem. U Edinburghu je 2002. godine osnovan Međunarodni odbor za kvalifikacije za testiranje programskih proizvoda (ISTQB), a tada je objavljena i prva verzija najpopularnijeg alata za praćenje grešaka „Jira“. Može se reći da je 2004. godina bila velika revolucija za testiranje zbog pojave alata za automatizirano testiranje kao na primjer „Selenium“. Te godine se pojavljuje i API testiranje putem alata kao što je „SOAP UI“ što označava još jednu prekretnicu u povijesti testiranja (Hernandez, 2019).

Trenutno razdoblje je fokusirano na istraživanja o testiranju softvera pomoću umjetne inteligencije i na testiranje softvera na više preglednika istovremeno.

5. RAZVOJ PROGRAMSKIH PROIZVODA

5.1. Metodologije razvoja programskih proizvoda

Prilikom započinjanja projekta čija je svrha razvoj programskog proizvoda jako je bitno koristiti odgovarajuću metodologiju razvoja softvera. Korištenjem adekvatne metodologije za razvoj softvera povećava se stopa uspješnosti projekta, osigurava se da programski proizvod bude isporučen na vrijeme i unutar određenog budžeta te da zadovoljava sve korisničke zahtjeve. Metodologija razvoja softvera se definira kao skup pravila i smjernica koje se koriste u procesu istraživanja, planiranja, projektiranja, razvoja, testiranja i održavanja programskog proizvoda (Despa, 2014). Postoji na stotine metoda razvoja softvera koje se dijele na klasične i agilne metode razvoja softvera. U ovom radu će se od klasičnih metoda objasniti metoda Vodopada i metoda Unificirani Proces, a od agilnih metoda će se objasniti Ekstremno Programiranje i Scrum.

Klasične metode su se pojavile 1970-ih godina, a koriste se još i danas. Korištenjem klasičnih metoda razvoj softvera se pomno planira kroz sve korake procesa te se određuje točan početak i kraj procesa razvoja softvera. Klasične metode zahtijevaju opsežno dokumentiranje svih aktivnosti i zahtjeva projekta kao i svih izrađenih dijelova programskog proizvoda. Koristi se prediktivni pristup, precizno se planiraju sve aktivnosti razvoja softvera za duži vremenski period, jer se klasične metode najčešće koriste za projekte gdje su korisnički zahtjevi vrlo jasni i detaljni, a plan razvoja se može relativno jednostavno izraditi. Klasične metode imaju za cilj definirati proces koji će uredno funkcionirati za sve članove tima kao što su programeri, menadžeri, dizajneri i testeri (Magner, 2005).

Tijekom 1960-ih godina jedina metoda razvoja softvera koja se koristila bila je „kodiraj i popravi“, a implementirana je od strane programera. Budući da se navedena metoda nije pokazala toliko uspješnom, 1970. godine Winston Royce predlaže metodologiju Vodopada. Vodopad je prva metodologija koja je opće priznata kao metoda razvoja softvera. Ona naglašava iscrpno planiranje svake aktivnosti i opsežno dokumentiranje svakog koraka procesa razvoja softvera (Magner, 2005). Metodologija Vodopada je linearni, sekvencijalni proces gdje svaka slijedeća faza može započeti tek kad je prethodna faza završena. Dakle, svaka faza ima za rezultat određeni dio softvera. Kao što je već rečeno, metoda Vodopada se temelji na rigoroznom planiranju svake faze razvoja, pa je samim time vrlo predvidljiva i ne ostavlja mjesta za naknadne izmjene dokumentacije. Tek na samom kraju razvoja programskog proizvoda, nakon što je proizvod u potpunosti testiran i razvijen, klijent daje povratnu informaciju o tome koliko proizvod u osnovi zadovoljava njegove zahtjeve.

Metodologija Vodopada sadrži šest faza razvoja softvera čiji rezultati moraju biti ostvareni prije nego započne slijedeća faza (Hamilton, 2023). Prva faza se odnosi na istraživanje i prikupljanje korisničkih zahtjeva te softverskih zahtjeva kako bi na temelju istih bilo moguće planirati cijeli proces razvoja softvera te izraditi svu potrebnu dokumentaciju. Slijedeća je dizajn faza tijekom koje je potrebno detaljno osmisliti dizajn softvera što uključuje bazu podataka, cjelokupnu arhitekturu programskog proizvoda i izgled korisničkog sučelja. Nakon faze dizajna slijedi faza implementacije programskog proizvoda. U toj fazi programeri kodiraju softver prema korisničkim i softverskim zahtjevima. Nakon što je faza implementacije u potpunosti završena, slijedi faza testiranja softvera. U ovoj fazi softver se iscrpno testira kako bi se utvrdilo zadovoljava li specifikacijske zahtjeve i radi li softver ispravno. Nakon što su pronađene greške ispravljene i softver zadovoljava korisničke zahtjeve slijedi faza dostave proizvoda korisnicima. Tijekom korištenja, klijent i ostali korisnici daju povratne informacije o softveru, a programeri prijavljene nedostatke rješavanju u fazi održavanja, što je ujedno i posljednja faza razvoja softvera putem metodologije Vodopada (Despa, 2014). Faze metodologije Vodopada su prikazane na slici 5.1.1.

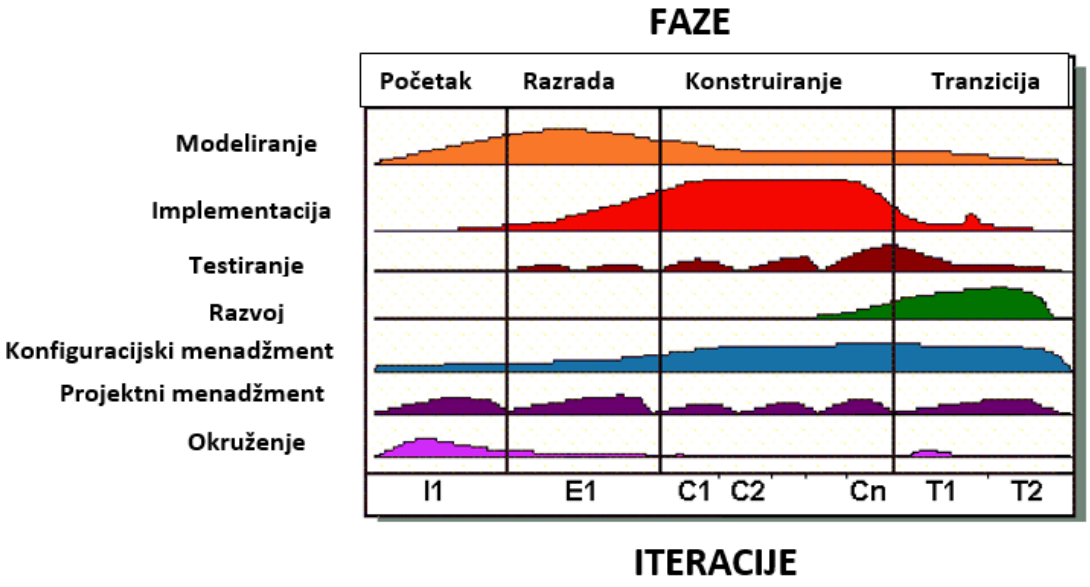
Slika 5.1.1. Faze metodologije Vodopada

Izvor: Vlastita izrada

Osim metodologije Vodopada pojasnit će se još jedna klasična metodologija poda nazivom Unificirani Proces (UP). Navedena metodologija nastala je pred kraj dvadesetog stoljeća kao rezultat suradnje Ivara Jacobsona, Gradya Boochea i Jamesa Rumbaugha. Naziv Unificirani Proces proizlazi iz toga da ova metoda objedinjuje nekoliko metoda koje su spomenuti autori već proučavali. Na primjer, UP se koristi u kompletu s grafičkim jezikom UML u okviru istog projekta s namjerom da nadopunjuju jedna drugu. Metoda UP definira na koji način će se provoditi proces razvoja softvera, a UML omogućava dokumentiranje razvoja softvera. UML programski kod prikazuje u obliku dijagrama, kako bi se manje tehnički kompetentnim pojedincima omogućilo razumijevanje problema čime se doprinosi boljem uspjehu projekta. Naglašava se važnost kontinuirane provjere kvalitete softvera kroz planiranje

kontrole kvalitete i procjene kvalitete softvera. UP se temelji i na principu slučajeva upotrebe i scenarija koji su se pokazali odlični u vidu ispunjavanja funkcionalnih zahtjeva i održavanja sustava u okviru očekivanog ponašanja. Unificirani Proces se izvodi na iterativni i inkrementalni način, a koristi se za razvoj objektno orijentiranih sustava (Avad, 2005). Navedene značajke su ključne smjernice kojih se treba pridržavati tijekom životnog ciklusa razvoja projekta.

Unificirani proces se dijeli na četiri faze, a to su: početak, razrada, izgradnja i prijelaz. U fazi početka se izrađuje poslovni slučaj te se procjenjuje izvedivost projekta i opseg dizajna softvera. U fazi razrade izrađuje se osnovna arhitektura softvera, dogovara plan izgradnje istog te se provodi analiza rizika. Do kraja faze konstrukcije trebalo bi biti dostupno beta izdanje softvera spremno za preliminarno testiranje od strane klijenta. Tijekom faze prijelaza sustav se predstavlja korisnicima i još jednom klijentu, gdje se projektni tim i klijent trebaju složiti da su dogovoreni ciljevi iz početne faze zadovoljeni. Navedene faze prikazane su na slici 5.1.2. Potrebno je naglasiti da UP unaprijed definira uloge projektnog tima što cijeli projekt čini manje fleksibilnim i podložnim promjenama.



Slika 5.1.2. Životni ciklus Unificiranog Procesa

Izvor: Ambler, W,. S, (2005): Enterprise Unified Process

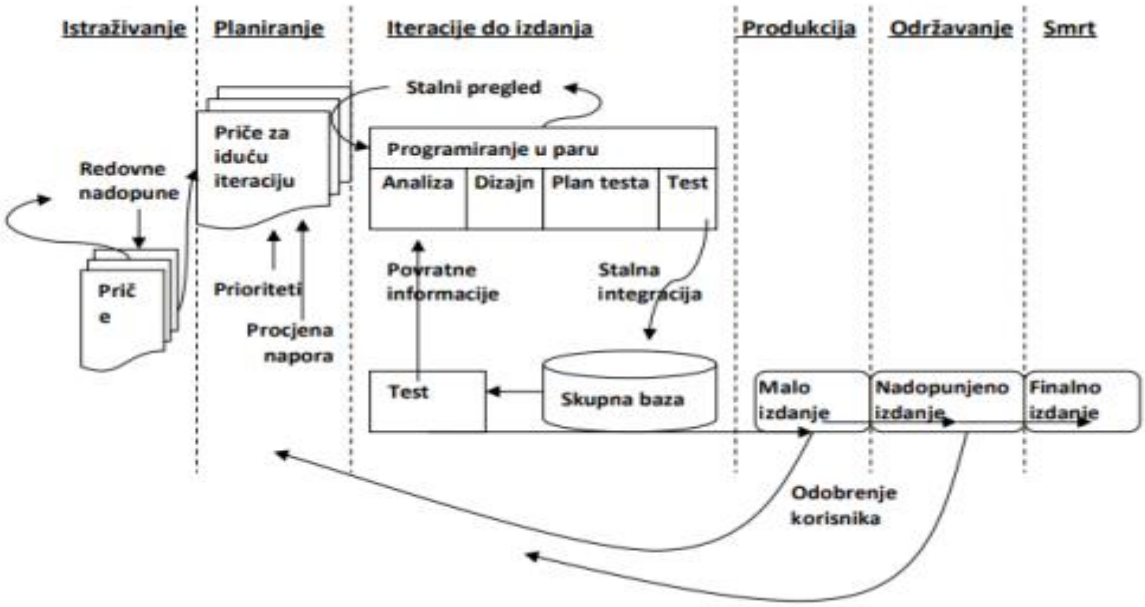
Agilne metodologije se pojavljuju u 21. stoljeću, a glavni cilj im je bio ubrzati proces razvoja softvera i smanjiti opseg projektne dokumentacije. Ono što agilne metode čini inovativnim nisu samo novi pristupi i metode koje koriste, već njihovo prepoznavanje ljudi kao primarnih pokretača projekta. Ističu važnost malih projektnih timova sa snažnim fokusom na učinkovit razvoja softvera. Dakle, agilne metodologije su za razliku od klasičnih orijentirane na ljude, članove projektnog tima i krajnje korisnike

kao najvažnije čimbenike uspješnog razvoja softverskog proizvoda (Magner, 2005). Agilne metodologije, kao što im i sam naziv govori, prihvaćaju promjene u svim fazama projekta, a na promjene zahtjeva gledaju pozitivno jer na takav način bolje konkuriraju na samom tržištu. Naglašavaju da promjene nije moguće zaustaviti, stoga se kao izazov postavlja pronalazak ispravnog odgovora na promjene uz minimalan trošak. Ističe se važnost usklađenosti programskog proizvoda sa stvarnim rezultatima, a ne sa detaljnom dokumentacijom. Smatra se da je sam program, u izvornom obliku, najbolja dokumentacija za daljnji razvoj softvera. Osim svođenja dokumentacije na minimum i planiranje postaje kratkoročno, odnosno aktivnosti se planiraju tek za tjedan ili dva unaprijed. Na takav način projekti se ne planiraju sukladno s opsežnim planom, već s poslovnom vrijednošću. Svaki razvojni ciklus (eng. Sprint) dodaje određenu poslovnu vrijednost proizvodu. Budući da se razvoj softvera promatra kroz kontinuirani niz razvojnih ciklusa čiji broj nije moguće točno predvidjeti svaka iteracija se usklađuje s trenutnim zahtjevima koji se konstantno mijenjaju. Iz tog razloga ne postoji konačna specifikacije cijelog projekta. Agilne metode razvijaju softver kao nelinearni postupak što bi značilo da se više procesa događa odjednom, te nije potrebno čekati da se jedna faza završi da bi druga mogla početi. Naglasak se stavlja na decentralizirani pristup, što bi značilo da odluke donosi cijeli tim, a ne samo menadžer, čime se potiče bolja međusobna suradnja i organizacija. Kako bi svi članovi tima i klijent uvijek imali sve potrebne informacije o projektu, organiziraju se interni dnevni sastanci među članovima tima te tjedni sastanci s klijentom. Na taj način se potiče bolja komunikacija i suradnja. U nastavku će se pojasniti metode Ekstremno Programiranje (XP) i Scrum.

Ekstremno Programiranje (XP) je agilna metodologija nastala 2000-te godine, a razvio ju je Kent Beck (Magner, 2005). Pojam ekstremno proizlazi iz toga što se programiranje dovodi do ekstremnih razmjera. Proces razvoja softvera se sastoji od mnoštva kontinuiranih, vrlo brzih iteracija koje ne traju više od dva tjedna. Rezultat svake iteracije je nova, ažurirana verzija softvera. Nova verzija softvera može sadržavati neke nove funkcionalnosti ili samo popravke nekih pogrešaka već postojećih funkcionalnosti. Korisnički zahtjevi se definiraju u obliku korisničkih priča (eng. user stories) gdje kupac opisuje kako želi da softver radi. Korisničke priče se prioritiziraju te se dijele po iteracijama, onako kako ih je moguće implementirati. Tu vrlo važnu ulogu imaju klijenti koji su stalno dostupni članovima tima donoseći korisničke priče, sadržaj pojedine verzije softvera i sudjeluju u testiranju softvera. Na ovakav način nije moguće kašnjenje isporuke programskih proizvoda, jer se jednostavno ono što nije implementirano ostavlja za buduće iteracije, a ono što je izrađeno se isporučuje. Naglasak je na jednostavnom dizajnu što znači da dizajnira najjednostavnije moguće rješenje, a nepotrebna složenost i dodatni kod odmah se uklanjaju. Na to se naslanja koncept restrukturiranja sustava uklanjanjem duplog koda, pojednostavljenjem i dodavanjem fleksibilnosti, ali bez mijenjanja funkcionalnosti softvera. Tu se ističe još jedno načelo na koje se oslanja Ekstremno Programiranje, a to je da se svaki

novi dio koda potrebno što prije integrirati s već postojećim, ali tek nakon što se ta nova funkcionalnost testira. Prilikom integracije novih funkcionalnosti potrebno je ponovno testirati cijeli softver kako bi promjene u konačnici bile sigurno integrirane. Jedna od inovacija koju je XP razvio jest programiranje u paru (Despa, 2014). To bi značilo da sav programski kod pišu dva programera zajedno, na jednom računalu. Na takav način lakše dolaze do rješenja i međusobno si ispravljaju eventualne pogreške. Parovi se tijekom projekta izmjenjuju čime se dolazi do koncepta kolektivnog vlasništva. To znači da niti jedna osoba ne posjeduje niti jedan segment koda niti je odgovorna za njega. Svatko može promijeniti bilo koji dio koda, u bilo kojem trenutku.

Životni ciklus Ekstremnog programiranja se sastoji od šest faza: istraživanje, planiranje, razvoj softvera po iteracijama, proizvodnja, održavanje i gašenje projekta (Magner, 2005). U fazi istraživanja klijent smišlja korisničke priče za koje želi da budu uključene u performanse softvera. To vodi do faze planiranja gdje se te korisničke priče prioritiziraju te se kreće u razvoj softvera po iteracijama. U prvoj iteraciji se stvara arhitektura cijelog sustava, a zatim se cijeli sustav postupno nadograđuje. Nakon toga se sustav dodatno testira kako bi se mogao predstaviti klijentu i pustiti u fazu proizvodnje. Korisničke priče koje nisu implementirane, odgađaju se za kasniju implementaciju u ažuriranim izdanjima izrađenima tijekom faze održavanja. U konačnici, kada klijent više nema novih korisničkih priča i sve potrebne funkcionalnosti su implementirane dolazi se u fazu gašenja projekta. Navedene faze prikazane su na slici 5.1.3.



Slika 5.1.3. Životni ciklus Ekstremnog programiranja

Izvor: M. A. Avad (2005) : A Comparison between Agile and Traditional Software Development Methodologies

Scrum metodologija je iterativni, inkrementalni pristup koji se koristi za razvoj bilo kojeg proizvoda (Magner, 2005). Ova metodologija se koncentrira na način funkcioniranja projektnog tima u svrhu fleksibilne izrade softvera u konstantno promjenjivom okruženju. Scrum ne nudi nikakve posebne metode razvoja softvera, već zahtijeva određene menadžerske metode i alate koje se koriste u različitim faza razvoja programskog proizvoda. Na takav način se nastoji izbjeći nepredvidivost i prevelika kompleksnost procesa. Navedena metodologija se temelji na čestoj isporuci vrijednosti korisnicima, međusobnoj suradnji članova projektnog tima, fleksibilnosti procesa i transparentnosti. Cilj metodologije je brza reakcija na promjene zahtjeva korisnika te isporuka kvalitetnog softvera koji je prilagođen svim zahtjevima tržišta.

Postoji nekoliko ključnih stavki na kojima počiva cijela Scrum metodologija, a to su: specifikacija proizvoda (eng. product backlog) , sprint, sprint sastanci, dokumentacija svakog sprinta (eng. sprint backlog), dnevni Scrum (Kanan & Verma, 2014).

Specifikacija proizvoda sadrži prioritetnu listu svih funkcionalnosti proizvoda te sve promjene koje klijent želi implementirati. Član projektnog tima koji je zadužen za ažuriranje specifikacije naziva se product owner.

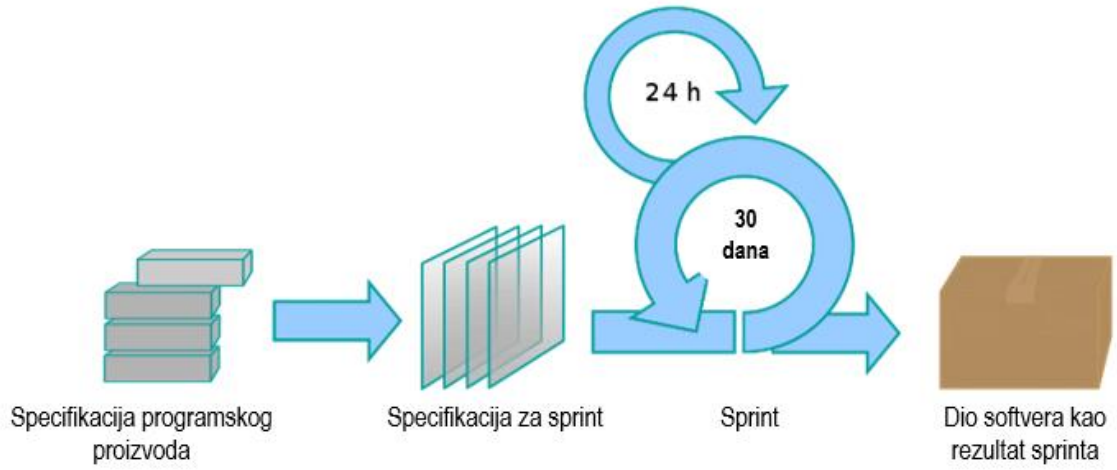
Sprintevi su vremenski ciklusi u trajanju od dva do četiri tjedna, a rezultat svakog sprinta je nova funkcionalnost ili ispravak određenih pogrešaka u softveru. Svaki sprint mora rezultirati dijelom softvera koji se može isporučiti.

Sprint sastanci služe sastavljanju plana za slijedeći sprint, a tome prisustvuju klijenti, menadžment, product owner i projektni tim. Tu se odlučuje o ciljevima i funkcionalnostima koje slijedeći sprint treba isporučiti.

Dokumentacija svakog sprinta sadrži popis značajki koje su dodijeljene određenom sprintu. Kada su sve te funkcionalnosti isporučene slijedi novi sprint.

Dnevni Scrum je dnevni sastanak koji traje približno petnaest minuta, a organizira se zbog praćenja napretka projektnog tima i rješavanja bilo kakvih nedoumica ili prepreka s kojima se tim suočava.

Vrlo je bitno naglasiti da u Scrum pristupu nema tipičnog menadžerskog upravljanja, već se potiče samoorganizacija projektnog tima koji sam odlučuje što će slijedeće učiniti, a menadžment uklanja određene zapreke i probleme. Takav pristup značajno poboljšava produktivnost tima i uspješnost razvoja programskog proizvoda. Slijedeća slika prikazuje pojednostavljeni prikaz Scrum metode.



Slika 5.1.4. Prikaz Scrum metodologije

Izvor: https://www.datascience-pm.com/scrum/

Klasične metodologije su prikladne za dugotrajne projekte gdje su zahtjevi vrlo detaljni i nisu skloni promjenama. S druge strane agilne metodologije pružaju veliku fleksibilnost i prilagodljivost promjenama zbog čega su prikladne za projekte sklone promjenama. Dok se klasične metodologije oslanjaju na opsežnu dokumentaciju, agilne metodologije su je svele na minimum, dokumentirajući isključivo rezultate performansi samog softvera. Agilne metodologije su prikladne za projekte gdje su softverske specifikacije nejasne i sklone promjenama. Uspoređujući navedene pristupe razvoju softvera, može se zaključiti da svaka metodologija ima svoje prednosti i mane u odnosu na vrstu projekta za čiji će se razvoj koristiti.

5.2. Faze razvoja programskog proizvoda

Kako bi se određeni programski proizvod uspješno razvio od same ideje pa sve do konačnog proizvoda, on mora proći kroz određene faze razvoja. Tako je na samom početku potrebno prikupiti i analizirati zahtjeve na temelju čega se planira daljnji razvoj softvera. Nakon toga se dizajnira korisničko sučelje i struktura cijelog softvera. Korisničke i softverske zahtjeve je zatim potrebno implementirati i testirati te promijeniti i popraviti funkcionalnosti koje nisu prilagođene specifikacijskim zahtjevima. Nakon što je softver testiran i siguran za korištenje isporučuje se na tržište te se dalje bavi održavanjem razvijenog programskog rješenja. Sve navedeno se može sažeti pod nazivom „životni ciklus razvoja programskog proizvoda“. Svrha životnog ciklusa razvoja programskog proizvoda jest isporuka vrlo kvalitetnog softvera koji je u skladu s korisničkim zahtjevima. Kaner, Falk i Nguyen (1994), dijele životni ciklus

softverskog proizvoda u pet faza: planiranje i analiza zahtjeva, dizajn, implementacija, testiranje, isporuka proizvoda na tržište i održavanje. U nastavku će se detaljno objasniti svaka od navedenih faza.

Planiranje i analiza zahtjeva je prva faza životnog ciklusa razvoja programskog proizvoda, a smatra se temeljnom i najvažnijom fazom. U ovoj fazi trebaju biti uključeni iskusniji članovi tima kako bi klijentu pomogli u definiranju vizije i funkcije samog proizvoda. Temeljem korisničkih zahtjeva počinje se sastavljati projektna dokumentacija gdje se opisuju funkcionalnosti i okvirni dizajn programskog proizvoda. Kako bi se riješile određene nedoumice u vezi izvedivosti nekih funkcionalnosti potrebno je od samog početka u projekt uključiti tim za osiguravanje kvalitete proizvoda. Nakon što su dogovoreni svi korisnički zahtjevi na red dolazi planiranje budžeta i vremena potrebnog za izradu softvera. Tu se razmišlja o tome jesu li funkcionalnosti proizvoda dovoljno smislene i povezane te je li to smjer u kojem se proizvod treba razvijati. Također je potrebno razmišljati o tome jesu li svi zahtjevi kompatibilni s obzirom na budžet, vrijeme i broj ljudi u timu, odnosno definirati opseg projekta. Ako se odmah od samog početka počne razmišljati i planirati u smjeru ovakvog osiguravanja kvalitete moguće je uštediti dosta novca i vremena („Software Testing Help“, 2023).

Druga faza razvoja programskog rješenja se odnosi na dizajniranje. Dizajneri na temelju analize korisničkih zahtjeva dizajniraju arhitekturu cijelog sustava, bazu podataka, izgled korisničkog sučelja. Dizajniranje se dijeli na vanjsko i unutarnje. Vanjsko dizajniranje se odnosi na dio softvera koji je vidljiv korisnicima, a to je korisničko sučelje. Unutarnje dizajniranje se odnosi na organiziranje arhitekture sustava, baze podataka i različitih modula. Vrlo je bitno da pri tome sve funkcionalnosti budu logično posložene te da međusobno mogu funkcionirati bez problema. S druge strane je potrebno paziti na to da korisničko sučelje bude vrlo intuitivno te da ne zbunjuje korisnike. Zbog toga je važno da cijeli dizajn pregleda tim za osiguranje kvalitete te predloži određene ispravke i komentare (Raugnath, et al., 2010). Na takav način će se osigurati dizajn kvalitetnog i pouzdanog proizvoda. Kao rezultat faze dizajniranja proizlazi dokument s detaljnim planom implementacije softvera.

Implementacija je treća faza životnog ciklusa razvoja programskog rješenja. Tijekom ove faze programeri na temelju dokumenta s detaljno opisanim unutarnjim i vanjskim dizajnom aplikacije počinju kodirati funkcionalnosti softvera. Bitno je da programeri jasno razumiju što se točno misli pod određenom funkcionalnosti, stoga im dizajneri određene zahtjeve pojašnjavaju u pseudokodu. Tom kombinacijom programskog i govornog jezika dizajneri znatno olakšavaju razumijevanje dizajna od strane programera („tutorialspoint, bez dat.). Tijekom kodiranja programeri sami testiraju dijelove koda kako bi bili sigurni da određene funkcionalnosti rade kako je očekivano.

Nakon faze implementacije slijedi faza testiranja programskog proizvoda kako bi se osiguralo da softver radi onako kako je očekivano. Nakon što su programeri testirali sam kod daljnje testiranje korisničkog

sučelja provode testeri. Prilikom testiranja testeri se oslanjaju na softversku specifikaciju kako bi bili sigurni da je softver izrađen prema korisničkim zahtjevima. Testiranje se provodi sve dok softver ne bude u skladu sa očekivanim standardom.

Posljednja faza životnog ciklusa razvoja programskog proizvoda je isporuka proizvoda na tržište i održavanje. Nakon isporuke proizvoda na tržište programeri brinu o održavanju proizvoda na način da popravljaju prijavljene pogreške i ažuriraju bazu podataka. S druge strane, testeri svako ažuriranje aplikacije ponovno testiraju provodeći regresijsko testiranje prema izrađenim testnim slučajevima.

6. ŽIVOTNI CIKLUS TESTIRANJA PROGRAMSKIH PROIZVODA

Testiranje softvera je ključni dio osiguravanja kvalitetnog softvera spremnog za isporuku na tržište. Životni ciklus testiranja programskog proizvoda se sastoji od različitih faza i aktivnosti kako bi se što bolje osiguralo očekivano ponašanje softvera prema korisničkim zahtjevima. Proces testiranja se dijeli na šest glavnih faza: analiza zahtjeva, kreiranje plana i strategije testiranja, dizajniranje testnih slučajeva, postavljanje testne okoline, izvođenje testova i završna faza testiranja (Deshpande, 2018).

Analiza zahtjeva je prva faza životnog ciklusa testiranja programskog proizvoda. Temelji se na analiziranju softverske specifikacije kako bi se razumjele sve funkcionalnosti koje je potrebno testirati. Tijekom ove faze tim za osiguravanje kvalitete pregledava sve softverske zahtjeve te utvrđuje mogu li se navedeni zahtjevi testirati ili ne. Ako određene zahtjeve nije moguće testirati, tim to mora iskomunicirati s klijentom i voditeljem projekta kako bi se uvele potrebne promjene.

Nakon što su utvrđene sve funkcionalnosti koje je potrebno testirati kreće se na slijedeću fazu, a to je kreiranje plana i strategije testiranja. Potrebno je utvrditi okvirno vrijeme potrebno za testiranje projekta te iznos troškova testiranja, a to određuje i planira test menadžer. Na temelju toga se planira opseg resursa potrebnih za testiranje projekta i alati koji će se koristiti prilikom testiranja.

Tijekom faze dizajniranja testnih slučajeva za svaku funkcionalnost se izrađuju detaljni testni slučajevi s definiranim očekivanim rezultatom, testnim koracima i testnim podacima. Nakon što su testni slučajevi i skripte u potpunosti izrađeni, još jednom se pregledavaju i ako je potrebno ispravljaju. Bitno je naglasiti da kada se testira softver kojem se konstantno dodaju nove funkcionalnosti da će se i testni slučajevi morati povremeno ažurirati.

Kako bi se testiranje softvera moglo neometano izvršavati vrlo je bitno osiguravanje postavljanja testne okoline. Na taj način testeri mogu testirati aplikaciju bez da se isprepliću s programerskom okolinom. Ako se pak dogodi da klijent nije osigurao testnu okolinu, to otežava posao testerima jer se usred testiranja aplikacije može dogoditi promjena koda u pozadini aplikacije što će uzrokovati kratkotrajni nestanak svih podataka iz aplikacije i tester će morati ponoviti testove.

Nakon pripreme testnih slučajeva i testne okoline na red dolazi izvođenje testova. Upotrebom testnih podataka i opisanih koraka u testnom planu, izvode se testni slučajevi kako bi se testiralo ispunjavaju li očekivani rezultat. Ako postoji odstupanje od očekivanog i stvarnog rezultata tada taj test nije prošao i potrebno je prijaviti pogrešku. Pogreške se prijavljuju ili putem određenog programa ili putem izvješća o pogreškama. Nakon što su prijavljene pogreške uklonjene ponovno se izvodi isti taj testni slučaj. Osim što se izvode testni slučajevi za očekivano ponašanje aplikacije, potrebno je izvoditi i testne slučajeve za neočekivana stanja aplikacije jer se na taj način lakše pronalaze pogreške (Hamilton, 2023).

U završnoj fazi testiranja izrađuje se izvješće o testiranju gdje se provjerava jesu li sve pogreške popravljene i zadovoljava li programski proizvod korisničke i softverske zahtjeve. Okuplja se cijeli projektni tim i zajedno komentira je li softver spreman za isporuku na tržište. Ako se softver smatra da softver zadovoljava sve kriterije daje se odobrenje za isporuku na tržište. U suprotnom se softver vraća na daljnju doradu.

7. VRSTE TESTIRANJA PROGRAMSKIH PROIZVODA

7.1. Ručno testiranje programskog proizvoda

Ručno testiranje se smatra najstarijom i najrigoroznijom metodom testiranja softvera upravo zato što tester ručno priprema testne slučajeve, a kasnije ih izvršava kako bi identificirao pogreške u softveru (Sharma, 2014). Na temelju testnog plana tester detaljno izvodi svaki testni slučaj te dobivene rezultate uspoređuje s očekivanim rezultatima. Bitno je napomenuti da se za izvođenje testnih slučajeva ne koriste nikakvi alati, stoga za uspješno ručno testiranje nije potrebno poznavanje niti jednog alata za testiranje. Ručno testiranje je ključno za pronalaženje kritičnih i rubnih pogrešaka u programskom proizvodu, stoga se svi novi softveri se prvo moraju testirati ručno, a tek nakon toga se može prijeći na automatizirano testiranje.

Slijedeći moderne trendove u softverskoj industriji, većina tvrtki preferira automatizirano testiranje, no ručno testiranje softvera je uistinu neophodno kako bi se osigurao softver koji zadovoljava sve kriterije korisničkih zahtjeva. U nastavku će se pojasniti nekoliko razloga zbog čega je ručno testiranje ključno za osiguravanje kvalitetnog softvera. Iako vlada mišljenje da je automatizacija nepogrešiva i u potpunosti točna, to nije istina. Kao i svaki drugi softver pa tako i alati za automatizirano testiranje mogu praviti pogreške tijekom testiranja i davati lažne rezultate testova. Upravo zato je prije automatizacije potrebno ručno testirati softver kako bi se znao očekivani rezultat određenog testa. Osim toga, testne skripte mogu sadržavati brojne greške u kodu koje mogu uzrokovati preskakanje nekih testova ili davanje pozitivnih rezultata za pogrešne funkcionalnosti. U tom slučaju je potrebno provesti ručno testiranje. Neki softveri mogu sadržavati vrlo složene funkcionalnosti čiji scenarij nije moguće automatizirati. U takvim situacijama tester se ponovno oslanja na ručno testiranje i procjenjuje određenu funkcionalnost. Ručno testiranje je ključno prilikom korištenja istraživačke tehnike testiranja jer navedena tehnika ne omogućava izradu testnih slučajeva unaprijed, a koristi se u vrlo složenim scenarijima testiranja. Dakle, ručnim testiranjem se izrađuju vrlo složeni testni slučajevi. Osim navedenog, ručno testiranje se koristi i kao metrika za procjenu prosječnog vremena potrebnog da projektni tim isporuči programski proizvod. S ručnim testiranjem se može započeti gotovo odmah dok je za automatizirano testiranje potrebna izrada automatiziranih skripti i postavka alata što znatno produljuje početno vrijeme testiranja. Upravo zato se za kratke projekte bira samo ručno testiranje. Testovi se mogu jednostavno prilagođavati novim funkcionalnostima softvera, a trošak se svodi samo na ljudske resurse. Važnost ručnog testiranja se posebno ističe prilikom testiranja korisničkog iskustva jer se takve vrste testova jednostavno ne mogu automatizirati. Korisničko iskustvo

je direktno povezano s korisnicima, stoga ga se može testirati jedino ručnom metodom testiranja (Sharma, 2014).

Iako ručno testiranje ima brojne prednosti, postoje i određeni nedostaci. Ručno testiranje može biti dugotrajno i vrlo zamorno zato što se svi testni slučajevi svaki put izvršavaju ručno od strane testera. Samim time se nameće potreba za velikim ulaganjem u ljudske resurse. Ručno testiranje se može smatrati i manje pouzdanim jer se zbog faktora ljudske pogreške testovi ne izvodi uvijek s jednakom razinom preciznosti. Kao još jedan nedostatak navodi se to što neke kompliciranije testove nije moguće kodirati u automatizirani test. Dakle, dugotrajno i iscrpno ručno testiranje može postati dosadno i zbog toga podložno pogreškama.

7.2. Automatizirano testiranje programskog proizvoda

Automatizirano testiranje je vrsta testiranja softvera za čije je izvođenje potrebno znanje o alatima za automatizirano testiranje softvera te znanje nekog programskog jezika kao što je TypeScript, JavaScript, C#, ili Python. Na samom početku procesa testiranja tester sam izrađuje testne skripte što zahtijeva više vremena i novčanih resursa za uspostavljanje alata za automatizaciju. No, kada se testni slučajevi uspješno automatiziraju, trošak testiranja drastično pada kao i vrijeme potrebno za izvršavanje svakog testa. Međutim, ako se automatizacija koristi za testiranje softvera koji se konstanto nadograđuje s novim funkcionalnostima, ažuriranje samih testnih skripti postaje skupo i mukotrpno. Softver mora biti dovoljno stabilan da bi automatizacija imala smisla. Zato se automatizacija koristi na dugotrajnim, stabilnim projektima koji nemaju nekih većih promjena. Tijekom vremena se stvara bogatu repozitorij testnih slučajeva što omogućuje bolju pokrivenost aplikacije testnim slučajevima i olakšava izvršenje određenog seta testnih slučajeva. Tijekom svakog pokretanja testnih skripti one se izvršavaju na jednak način, kao i svaki prijašnji put, što kod ručnog testiranja nije slučaj. Kod automatiziranog testiranja dobivaju se brže i jasnije povratne informacije o tome o kakvoj je pogrešci riječ. To dovodi do bolje organiziranog i produktivnijeg pristupa testiranju, a pogreške se brže uklanjaju. To u konačnici skraćuje vrijeme koje je potrebno da proizvod izađe na tržište. Također, automatizacija povećava učinkovitost testiranja jer se testovi mogu izvoditi i noću, bez prisutnosti testera („Katalon“, 2023).

Dugoročno gledano automatizacija softvera je vrlo isplativa jer se smanjuje trošak održavanja softvera zbog vrlo kratkog vremena potrebnog za izvođenje regresije koja je potrebna nakon svakog ažuriranja sustava te zbog potrebe za manjim brojem testera. Osim za regresijsko testiranje, automatiziranu pristup je odličan i za testiranje otpornosti softvera na stres. Ovakva vrsta testiranja zahtijeva opterećenje sustava tijekom duljeg vremenskog perioda što je vrlo teško realizirati ručnim testiranjem

(„Software Testing Help“, 2023). Iako je automatizirano testiranje sve popularnije, treba biti svjestan da je potrebno odvojiti određeno vrijeme i resurse za izradu testnih skripti, a kasnije je tim testovima potrebno organizirani upravljati. Osim navedenih prednosti kao što su isplativost, repetitivnost, višekratna upotreba testnih skripti na različitim verzijama softvera, pouzdanost i sveobuhvatnost automatizirano testiranje ima i određene nedostatke. Automatizacija nikada ne može pokriti sve testne slučajeve, a posebno ne one koji testiraju upotrebljivost i korisničko iskustvo. Vrlo je teško automatizirati sve testne slučajeve, a obično polovina testnih slučajeva na razini cijelog sustava ne može biti automatizirana. Dakle, cilj automatizacije testiranja nije samo smanjenje troška ljudskih resursa već poboljšanje produktivnosti, kvalitete i učinkovitosti izvođenja testova, a čak i ako su svi testovi automatizirani opet će postojati potreba za ručnim testiranjem. Automatizaciju nije lako samo odjednom uvesti, već organizacija mora zadovoljiti određene uvjete za uspješnu implementaciju automatizacije. Dakle, testni slučajevi moraju biti vrlo detaljno definirani kako bi se mogli automatizirati, alat za testiranje je implementiran u postojeću infrastrukturu, testeri već posjeduju znanje o automatiziranom testiranju. Ispunjavanjem navedenih preduvjeta organizacija osigurava efikasnu automatizaciju testiranja softvera („Katalon“, 2023).

7.3. Usporedba ručnog i automatiziranog testiranja programskog proizvoda

Kao što je već navedeno, i automatizirano i ručno testiranje imaju svojih prednosti i nedostataka, a svaki pristup je za određenu razinu testiranja bolji od drugog. Tako se može reći da je automatizirano testiranje najefektivnije kada se radi o izvršenju regresijskog testiranja, odnosno ispitivanju kvalitete aplikacije nakon popravka određenih pogrešaka ili izlaska nove verzije aplikacije, te za testiranje opterećenja softvera (Bezsmlna, 2019). Ručno testiranje je pogodno za testiranje složenih testnih slučajeva i testiranje prilagođenosti aplikacije korisniku. Ovaj dio testiranja nije moguće obaviti automatizacijom jer takvo testiranje uključuje vizualno promatranje aplikacije i pronalaženje grešaka u smislu korisničkog iskustva. Ručno testiranje je dosta fleksibilnije od automatiziranog, jer nije potrebno ponovno pisati nove testne skripte i pokretati ih, pa je ova vrsta testiranja prikladna u slučaju neplaniranih promjena u aplikaciji, odnosno ad-hoc testiranja (Bezsmlna, 2019). Automatizirano testiranje je pogodnije za dugotrajne projekte, jer je potrebno puno vremena za sastavljanje kvalitetnih testnih skripti, a ponekad upravo to može produžiti rok isporuke programskog proizvoda. Bitno je naglasiti da jednom kada se testne skripte kvalitetno izrade, testeri će uštedjeti puno vremena, a organizacija će smanjiti troškove testiranja. Automatiziranim testiranjem se dobivaju brže povratne informacije čime se smanjuje vrijeme potrebno za isporuku kvalitetnog proizvoda, no ne dobivaju se

nikakve povratne informacije o korisničkom iskustvu gdje je potrebno uvesti ručno testiranje (Hamilton, 2023).

Potrebno je naglasiti da nikad nije moguće automatizirati sve testove, pa se u praksi uvijek kombiniraju navedeni pristupi testiranju. Na takav način se mogu iskoristiti sve prednosti oba pristupa ovisno o vrsti projekta, korisničkim zahtjevima i dostupnim resursima.

8. METODOLOGIJE I RAZINE TESTIRANJA PROGRAMSKIH PROIZVODA

8.1. Metodologije testiranja

Kako bi se izvođenje testnih slučajeva izvršavalo na najefektivniji način potrebno je slijediti određene metodologije testiranja. Metodologije predstavljaju smjernice koje se koriste za uspješno testiranje softvera, a dijele se na: metodologiju crne kutije, metodologiju bijele kutije i metodologiju sive kutije („tutorialspoint“, bez dat.).

U nastavku će se pojasniti sve tri navedene metode.

8.1.1. Metodologija testiranja crne kutije

Prema Naik i Tripathy (2008) metodologija testiranja crne kutije se naziva još i funkcionalno testiranje jer u potpunosti zanemaruje unutarnju strukturu softvera te se fokusira isključivo na testiranje funkcionalnosti softvera prema korisničkim zahtjevima. Tester nema pristup izvornom kodu softvera i njegovoj strukturi pa se programski proizvod promatra kao crna kutija. Tester unosi testne podatke na temelju specifikacijskih zahtjeva i promatranjem utvrđuje jesu li rezultati testiranja očekivani. Glavni cilj testiranja crne kutije jest uvidjeti zadovoljava li softver korisničke zahtjeve. Prilikom korištenja navedene metode otkrivaju se pogreške vezane uz korisničko iskustvo, dizajn korisničkog sučelja ili propusti u smislu nepostojanja zahtijevanih funkcionalnosti.

Neke od prednosti testiranja softvera metodom crne kutije su te da pristup kodu nije potreban, pa nema preduvjeta za znanje programskog jezika što znači da ovakvo testiranje može provoditi velik broj umjereno kvalificiranih testera. Osim toga, ova vrsta testiranja je vrlo učinkovita za testiranje velikih segmenata koda te se jasno odvaja uloga programera i testera što znatno olakšava testiranje. Testovi se izvode s točke gledišta korisnika, a samo testiranje oduzima manje vremena od drugih metoda testiranja. Budući da je posao testera donekle blokiran nedostatkom informacija o unutarnjoj strukturi softvera, pristup testiranju crne kutije uključuje i tehniku eksperimentiranja, odnosno pristup pokušaja i pogreške (Naik & Tripathy, 2008).

Osim brojnih prednosti, postoje i određeni nedostaci metode testiranja crne kutije. Testne slučajeve za ovakvu vrstu testiranja nije moguće izraditi bez detaljno opisanih funkcionalnosti. Budući da se ne ulazi u unutarnju strukturu softvera samo određeni dio ulaznih podataka može biti ispravno testiran, a to može uzrokovati neučinkovito testiranje. Dakle, ograničava se testna pokrivenost softvera jer se izvršava samo određeni broj testnih slučajeva. Tester pokušava pokriti što je više moguće testnih

slučajeva, ali ne može ciljati točna područja sklona pogreškama jer ne ulazi u unutrašnjost softvera. Kako je već spomenuto, tester i programer su neovisni jedno o drugome što je u nekim slučajevima dobro, ali manjak suradnje može rezultirati ponavljanjem testova koji su već izvršeni od strane programera.

8.1.2. Metodologija testiranja bijele kutije

Metoda testiranja bijele kutije se naziva i tehnika strukturalnog testiranja softvera jer se testira unutarnja struktura kao što je kod i programska logika softvera ili samo određena komponenta. Budući da je tijekom testiranja programeru vidljiv cjelokupni kod ova metoda se naziva još i testiranje u otvorenoj kutiji, testiranje u prozirnoj kutiji ili testiranje temeljeno na kodu. Metoda testiranja bijele kutije je vrlo efektivna u pronalasku pogrešaka u kodu i popravka istih. Može se reći da se ovakva vrsta testiranja smatra i sigurnosnim testiranjem softvera jer se kod provjerava u svrhu validacije kompatibilnosti s dogovorenim dizajnom softvera te kako bi se otkrilo potencijalne slabe karike sustava. Navedena metoda se primjenjuje samo na manje jedinice nekog većeg sustava kako bi programer mogao temeljito testirati i razumjeti određenu softversku jedinicu te ispraviti sve propuste (Hamilton, 2023).

Budući da programer razumije izvorni kod softvera, olakšan je proces pronalaska ulaznih podataka za učinkovito testiranje softvera što znači da nema eksperimentiranja s ulaznim vrijednostima. Ovakva vrsta testiranja omogućuje optimizaciju programskog koda i uklanjanje nepotrebnih linija koda, a to može rezultirati pronalaskom skrivenih pogrešaka. Tijekom izrade testova programer se oslanja na strukturu softvera kako bi što bolje pokrio sve dijelove koda što rezultira učinkovitijim testiranjem (Nidhra & Dondeti, 2012).

S druge strane, potrebno je naglasiti da je testiranje bijele kutije nešto skuplja metoda jer je potreban vješti programer koji poznaje strukturu softvera. Bez obzira na to koliko kod programer bio vješt u kodiranju i testiranju nemoguće je testirati svaki dio koda, uvijek će ostati neke skrivene greške koje kasnije mogu uzrokovati probleme u radu softvera. Osim što ova vrsta testiranja zahtijeva iskusnog programera, potrebni su još i specijalizirani alati za analiziranje kod ili uklanjanje pogrešaka.

8.1.3. Metodologija testiranja sive kutije

Metodologija testiranja sive kutije predstavlja kombinaciju metodologija testiranja bijele i crne kutije. Metodologija testiranja sive kutije se koristi za testiranje kompleksnijih softvera na osnovu specifikacije, ali i znanja o unutarnjoj strukturi programskog proizvoda. Testiranje sive kutije objedinjuje sve prednosti metode testiranja crne i bijele kutije pokušavajući u potpunosti izuzeti njihove nedostatke. Naziv siva kutija je nastao iz toga što tester samo donekle vidi unutarnju strukturu softvera pa se metaforički naziva polu-prozirna kutija. Cilj navedenog testiranja je pronalaženje nedostataka u softverima na temelju softverske specifikacije i na temelju proučavanja unutarnje strukture softvera (Murray, 2021).

Ključna prednost metode testiranja sive kutije je ta da se testiranje izvodi i s gledišta programera i s gledišta krajnjeg korisnika što poboljšava učinkovitost testiranja. Korištenjem ove metode poboljšava se konačna kvaliteta softvera i osigurava se da zadovoljava i korisničke i softverske zahtjeve. Također, djelomično razumijevanje internih sustava softvera omogućuje kvalitetniju izradu testnih slučajeva.

Osim brojnih prednosti postoje i neki nedostaci navedene metode. Iako tester donekle ima pristup unutarnjoj strukturi softvera, to i dalje nije dovoljno za postizanjem dostatne pokrivenosti koda. To u konačnici može uzrokovati skrivene pogreške koje nije bilo moguće identificirati zbog nedovoljne upoznatosti s kodom softvera. U nekim situacijama može doći do ponavljanja testova, ako je programer već prije proveo određene testove.

Kako bi se još jednom istaknule glavne razlike između testiranja crne, bijele i sive kutije na slici 8.1.3. su jasno prikazane sve tri metode i razlika u vidljivosti unutarnje strukture sustava.

Slika 8.1.3. Prikaz ključnih razlika metoda testiranja crne, bijele i sive kutije
Izvor: https://aqua-cloud.io/what-is-white-box-testing/

8.2. Razine testiranja

8.2.1. Regresijsko testiranje

Svrha regresijskog testiranja (eng. Regression testing) je osigurati da promjene koje su uvedene u softver, kao na primjer nova funkcionalnost, nisu utjecale na ostale dijelove softvera. Regresijsko testiranje je potpuno ili djelomično izvođenje već ranije kreiranih testnih slučajeva kako bi se osiguralo da softver i dalje zadovoljava specifikacijske zahtjeve (Wong et al.,1997). Budući da je ovakva vrsta testiranja dugotrajna i skupa te čini najveći napor tijekom testiranja, poželjno je prioritizirati testne slučajeve kako bi se postigla bolja efikasnost. Bitno je odabrati one testove za koje je najveća vjerojatnost da će otkriti nove pogreške. To su najčešće testni slučajevi koji provjeravaju temeljne funkcionalnosti proizvoda, testni slučajevi za funkcionalnosti koje su prošle više promjena u posljednje vrijeme ili rubni testni slučajevi. Osim potrebe za regresijskim testiranjem nakon uvođenja nove funkcionalnosti, navedeno testiranje je potrebno izvoditi i nakon što je popravljena neka pogreška ili nakon integracije softvera s vanjskim sustavom.

Bitno je utvrditi razliku između ponovnog testiranja i regresijskog testiranja. Dakle, ponovno testiranje znači ponovno testiranje funkcionalnosti kako bi se osiguralo da je pogreška popravljena. Regresijsko testiranje znači testiranje softvera nakon promjene koda kako bi se osiguralo da novi kod nije utjecao na ostale dijelove koda. Ponovno testiranje ne provjerava sve prethodne verzije već samo provjerava rade li funkcionalnosti u sadašnjoj verziji, dok se kod regresijskog testiranja provjerava rade li sve značajke i na prethodnim verzijama softvera. Ponovno testiranje se koristi za testiranje neuspjelih testova dok se regresija koristi za sve testove koji su prošli. Regresija je testiranje niskog prioriteta jer se radi o ukupnom testiranju svih mogućih nedostataka dok je ponovno testiranje visokog prioriteta jer se fokusira na određeni problem. Regresijsko testiranje je moguće i poželjno automatizirati zbog velikog broja testova, dok se ponovno testiranje ne može automatizirati jer provjerava samo specifične nedostatke (Naik & Tripathy, 2008).

Prednosti regresijskog testiranja su te da olakšava učinkovit razvoj softvera i poboljšava kvalitetu istog. Ovakvim testiranjem postiže se usklađenost s korisničkim zahtjevima te se održava stabilnost softvera. Postoje i određeni nedostaci, a oni se odnose na to da je regresiju potrebno provesti svaki put kada se napravi i najmanja promjena te ako se provodi ručno može biti vrlo iscrpno i dugotrajno jer zahtijeva često ponavljanje (Hamilton, 2023).

8.2.2. Testiranje sistema

Testiranjem sistema (eng. System testing) se potvrđuje pravilno i potpuno integriran softverski proizvod. To je kritična faza u procesu razvoja softvera jer se procjenjuje zadovoljava li cijeli sustav u potpunosti korisničke zahtjeve, a ne samo mali dio softvera koji je integriran u veći sustav. Ovakvo testiranje spada pod metodologiju testiranja crne kutije jer se prvenstveno testira korisničko iskustvo aplikacije, a znanje o kodu sustava nije potrebno. Testira način na koji međusobno djeluju komponentne integriranog softvera sa sustavom u cjelini, a to se naziva testiranjem od kraja do kraja (eng. end to end testing). Temeljito se testiraju svi ulazni podaci kako bi se provjerilo jesu li dobiveni očekivani rezultati (Syed, et al., 2016).

Postoji nekoliko vrsta testiranja sistema, a to su: testiranje oporavka (eng. recovery testing), sigurnosno testiranje (eng. security testing), testiranje korisničkog sučelja i testiranje kompatibilnosti (Hamilton, 2023). Testiranje oporavka je vrsta nefunkcionalnog testiranja čija je glavna svrha isprovocirati kvar u sustavu kako bi se moglo procijeniti koliko je vremena potrebno da se sustav efikasno oporavi. Sigurnosno testiranje osigurava da nema velikih propusta i slabosti u softveru koji bi mogli uzrokovati oštećenja sustava. Tester napada sustav s posebnim softverom za provaljivanje u softver kako bi osigurao da obrambeni sustav radi bez pogreške. Testiranje korisničkog sučelja se izvodi s gledišta korisnika te se procjenjuje zadovoljava li korisničko sučelje sve zahtjeve u smislu svih zahtijevanih gumbova, ikona, menu-a. I posljednje je testiranje kompatibilnosti gdje se sustav testira na način da se pokreće na različitim operativnim sustavima.

Prednosti navedenog testiranja se ogledaju u tome što se sustav testira u cijelosti, testiranje se provodi na okolini vrlo sličnoj produkcijskoj te se na takav način najbolje može procijeniti zadovoljava li sustav specifikacijske zahtjeve. Osim prednosti mogu se navesti i nedostaci kao što je nepoznavanje unutarnje strukture softvera, oduzima dosta vremena i nije poznato na koji način je sustav integriran.

8.2.3. Istraživačko testiranje

Istraživačko testiranje (eng. Exploratory testing) se prema Bach (2003) definira kao istovremeno učenje, dizajniranje testnih slučajeva i izvođenje istih. Dakle, testni slučajevi se ne kreiraju unaprijed već testeri softver provjeravaju u hodu. Ovakvo testiranje se najčešće koristi u agilnim metodama i fokus se stavlja na otkrivanje, istraživanje i učenje. Istraživačko testiranje je simultani proces dizajniranja testnih slučajeva i izvođenja testova u isto vrijeme pa ovakvo testiranje nije moguće automatizirati. Potrebno je istaknuti da istraživačko testiranje nije nasumično testiranje već ad-hoc

pristup sa svrhom istraživanja sustava i pronalaženja pogrešaka. Kod ovakvog testiranja vrlo se lako izgubiti pa misija testiranja treba biti vrlo jasna te je poželjno voditi bilješke o tome što je potrebno testirati i kakva je kvaliteta programskog proizvoda.

Istraživačko testiranje je korisno kada softverska specifikacija nije dostupna ili je samo djelomično popunjena. Ovakvim pristupom se mogu otkriti pogreške koje bi neke druge tehnike testiranja možda zanemarile te pomaže proširiti opus testnih slučajeva što u konačnici rezultira efektivnijim testiranjem. Također se potiče kreativnost i generiranje novih ideja od strane testera što povećava njegovu produktivnost tijekom testiranja. Budući da se ovo testiranje ne temelji na softverskoj dokumentaciji već isključivo na razmišljanju i vještinama ispitivača to se može prepoznati kao potencijalni nedostatak ove metode.

8.2.4. Testiranje upotrebljivosti

Testiranje upotrebljivosti (eng. Usability testing) se definira kao metoda testiranja kojom se ispituje koliko je softverska aplikacija prilagođena i shvatljiva korisniku za korištenje. Zbog toga je ova razina testiranja poznata i pod nazivom testiranje korisničkog iskustva. Aplikacija se daje na korištenje malom skupu korisnika koji koriste aplikaciju u svrhu otkivanja pogrešaka i nedostataka u okviru upotrebljivosti. Osim jednostavnosti korištenja i sposobnosti aplikacije da ispuni korisničke zahtjeve važan je i izgled sučelja. Korisnici obično one programske proizvode koji imaju dobro dizajnirano korisničko sučelje procjenjuju kao kvalitetne. Dakle, cilj testiranja je zadovoljiti korisnika u kontekstu jednostavne navigacije kroz aplikaciju, ujednačenost formata svih zaslona unutar aplikacije, mogućnost pretraživanja aplikacije, točnost podataka i svih linkova koji vode na neke web stranice (Hamilton, 2023).

Testiranje upotrebljivosti pomaže poboljšati zadovoljstvo krajnjeg korisnika te učiniti sustav učinkovitim i jednostavnim za korištenje. Ovakvim testiranjem se prikupljaju povratne informacije od ciljane skupine korisnika što pomaže u otkivanju problema upotrebljivosti. Kao glavni nedostatak se navodi trošak testiranja jer je zapošljavanje brojnih ispitivača vrlo skupo.

8.2.5. Testiranje prihvatljivosti

Testiranje prihvatljivosti (eng. User acceptance testing) je vrsta testiranja koja se izvodi od strane klijenta kako bi se potvrdilo da softver radi prema korisničkim zahtjevima prije nego se proizvod pusti na tržište. Ova vrsta testiranja se izvodi u završnoj fazi testiranja, nakon što je već obavljeno

funkcionalno, integracijsko i sistemsko testiranje. Tijekom testiranja prihvatljivosti fokus je na potvrdi da su svi korisnički zahtjevi integrirani i da rade onako kako je očekivano. Navedeno testiranje se provodi na zasebnoj okolini s testnim podacima kao na produkcijskoj okolini. Potreba za korisničkim testiranjem prihvatljivosti se javlja zbog toga što programer možda nije točno razumio korisničke zahtjeve već je izradio neke funkcionalnosti prema vlastitom razumijevanju dokumentacije. S druge strane, zahtjevi klijenta se tijekom vremena mogu promijeniti, a da to klijent nije jasno iskomunicirao s programerima. Na kraju je najbitnije to da je klijent zadovoljava softverom.

8.2.6. Testiranje jedinica

Testiranje jedinica (eng. Unit testing) se temelji na testiranju najmanjih pojedinačnih komponentni softvera kao što su funkcije, metode, procedure i moduli. Svrha ove razine testiranje je potvrditi da svaka jedinica softverskog koda radi onako kako je očekivano. Jedinično testiranje se provodi još tijekom faze kodiranja, a provode ga programeri. Ova razina testiranja spada pod metodologiju testiranja bijele kutije jer se testira samo unutarnja struktura sustava, a jediničnim testovima se izoliraju dijelovi koda i provjerava se njihova funkcionalnost (Hamilton, 2023).

Prednost ovakve vrste testiranja je ta što se izvodi od strane programera još tijekom faze kodiranja pa minimalizira troškove testiranja. Na ovakav način se postiže najbolja struktura koda za određeni softver, a budući da se pogreške mogu uočiti još prilikom pisanja koda nije ih teško otkloniti. Osim navedenog potrebno je istaknuti da tijekom testiranja pojedinačnih jedinica softvera nije potrebno čekati dostupnost ostalih jedinica softvera.

Kao nedostatak navodi se to što nije moguće uočiti sve pogreške u softveru jer se jedinice testiraju odvojeno, a samim time što je svaku jedinicu potrebno zasebno testirati troši se jako puno vremena. Osim toga, nije jednostavno napisati testne slučajeve koji će pokriti sve jedinice sustava na pravi način (Hamilton, 2023).

8.2.7. Integracijsko testiranje

Integracijsko testiranje (eng. Integration testing) se definira kao testiranje gdje su svi softverski moduli logički integrirani i testirani kao grupa. Takvo testiranje je vrlo bitno jer se obično softver sastoji od više modula koje su izveli različiti programeri. Svrha ovakvog testiranja jest otkriti nedostatke u interakciji između integriranih softverskih modula (Syed et al., 2016). Može se reći da se integracijsko testiranje nalazi negdje između testiranja sistema i testiranja jedinica (eng. unit testing).

Postoji tri tehnike integracijskog testiranja, a to su: odozgor prema dole (eng. top down integration), odozdo prema gore (eng. bottom up integration) i vrsta integracijskog testiranja kao veliki prasak (eng. bing bang integration). Pristup integracije odozgor prema dole znači započeti integracijsko testiranje od softvera kao jedinstvenog modula pa sve do manjih modula. Integracijsko testiranje odozdo prema gore znači započeti testiranje od najmanjih modula pa sve do testiranja softvera kao jedinstvenog modula. Integracijsko testiranje kao veliki prasak znači testiranje softvera kombiniranjem svih modula i testiranjem pogrešaka u cjelini (Hamilton, 2023).

Prednosti su te što se testiranje bazira na već prethodno testiranim modulima i svaki modul se odvojeno testira. Nedostaci su ti što je teško otkloniti pogreške, korisnik nema pristup aplikaciji sve do posljednje faze razvoja.

8.2.8. Testiranje performansi

Testiranje performansi (eng. Performance testing) je proces testiranja softvera koji testira performanse softvera kao što je vrijeme učitavanja, brzina, stabilnost, skalabilnost i pouzdanost pod određenim radnim opterećenjem (Hooda & Chhillar, 2015). Testiranjem performansi utvrdit će se zadovoljava li softver određene zahtjeve stabilnosti, skalabilnosti i brzine pod određenim radnim opterećenjem kako na tržište ne bi izašao proizvod s lošim performansama. Svrha ovog testiranja je identificirati i ukloniti uska grla u performansama aplikacije. Uska grla su prepreke u sustavu koje otežavaju rad ostatka sustava, a najčešće ih uzrokuje neispravan dio koda. Kao rješenje za popravak uskih grla nameće se ponovno pokretanje loše pokrenutih procesa ili dodavanje dodatnog hardvera. Neki od primjera uskih grla su: zagušenost procesora, puna memorija, ograničenje operativnog sustava (Hamilton, 2023). Glavni problem s izvedbom aplikacije se najviše odnosi na brzinu, jer aplikacija koja radi sporo gubi pažnju korisnika. Dakle, problemi kao što je dugo vrijeme učitavanja, loše vrijeme odgovora i slaba skalabilnost rezultiraju nestabilnim proizvodom loše kvalitete. Vrijeme učitavanja aplikacije, odnosno vrijeme potrebno da se ona pokrene mora se svesti na minimum, po mogućnosti na manje od nekoliko sekundi. Vrijeme odgovora aplikacije je vrijeme koje je potrebno da aplikacija na temelju unesenih podataka dostavi odgovarajući odgovor. Bez obzira na to koliko se aplikacija brzo učitava ili odgovara na unesene podatke, ako softver ima slabu skalabilnost što znači da aplikacija ne može podržati veći broj korisnika tada aplikacija jednostavno ne može konkurirati na tržištu.

Postoji nekoliko vrsta testiranja performansi kao što je testiranje opterećenja, testiranje otpornosti na stres, testiranje izdržljivosti, spike testiranje, testiranje volumena i testiranje skalabilnosti. Testiranje opterećenja testira rad softvera pod određenim korisničkim opterećenjem kako bi se identificirala uska

grla. Testiranje otpornosti na stres se izvodi na način da se softver stavlja pod ekstremno radno opterećenje kako bi se identificirala prijelomna točka rada softvera. Testiranje izdržljivosti se provodi da bi se procijenilo može li aplikacija izdržati određeno opterećenje tijekom dužeg vremenskog perioda. Spike testiranje se izvodi na način da se aplikaciju iznenadno opterećuje u većim količinama, a opterećenja izvode korisnici. Tijekom testiranja volumena popunjava se baza podataka aplikacije, te se prati njezino ponašanje pod različitim veličinama baze podataka. Testiranje skalabilnosti pomaže u planiranju dodavanja opterećenja sustavu (Hamilton, 2023).

8.2.9. Unakrsno testiranje

Unakrsno testiranje (eng. Cross Browser Testing) je vrsta funkcionalnog testiranja kojom se provjerava radi li aplikacija prema očekivanjima na različitim preglednicima. Ovakva vrsta testiranja je potrebna jer korisnik može otvoriti aplikaciju u bilo kojem pregledniku pa treba omogućiti jednaku izvedbu aplikacije na svim preglednicima. Problemi s različitim preglednicima mogu biti na primjer nejednaka veličina fonta na svim preglednicima, implementacija JavaScripta može biti drugačija, poravnanje stranice ili kriva orijentacija slika. Dakle, potrebno je provjeriti da se svi elementi ispravno prikazuju, a to može biti vidljivost teksta, slika ili nekih gumbova. Ovakvim testiranjem se osigurava dosljednost i pozitivno korisničko iskustvo („Software Testing Help“, 2023).

9. ALATI ZA TESTIRANJE PROGRAMSKIH PROIZVODA

Alati za testiranje softvera se definiraju kao proizvodi koji podržavaju različite testne aktivnosti kao što je planiranje testiranja, prikupljanje zahtjeva, izvođenje testova, bilježenje pogrešaka i analiza testnih slučajeva. Na tržištu postoji jako puno alata za ručno i automatizirano testiranje programskih proizvoda, a u nastavku će se pojasniti neki od njih.

9.1. Alati za ručno testiranje programskih proizvoda

Alati za ručno testiranje pružaju mogućnost jednostavnije izrade i planiranja testova te praćenja pogrešaka. Neki od trenutno najkorištenijih na tržištu su: TestRail, Zephyr Enterprise, Testpad, Jira, Jmeter („InterviewBit“, 2023).

TestRail je alat koji se koristi za skalabilno i prilagodljivo upravljanje testnim slučajevima. Pruža mogućnost uvida u sva izvješća o aktivnostima na projektu te dokumentiranje testnih slučajeva putem videa. Navedeni alat povećava učinkovitost tima zbog jednostavne komunikacije putem informativnih nadzornih ploča.

Zephyr Enterprise je alat koji pruža mogućnosti izrade testnog plana, definiranja korisničkih zahtjeva i generiranje izvješća o testiranju. Ovaj alat pomaže tvrtkama svojom fleksibilnošću i brzinom isporuke izvještaja o pogreškama.

Testpad je pristupačan alat za ručno testiranje koji umjesto upravljanja testnim slučajevima jednim po jednim koristi planove testiranja na temelju softverske specifikacije. Također, vrlo je jednostavan za korištenje pa ga mogu koristiti i oni koji nisu softverski testeri.

Jira je alat koji se koristi za praćenje pogrešaka tijekom razvoja softvera. Budući da se ovim alatom može pratiti razvoj svih projekata nije ograničen samo na softverski razvoj, a kompatibilan je s agilnim projektima.

Jmeter je alat namijenjen statičkom i dinamičkom testiranju performansi web aplikacija. Jednostavan je za korištenje, a korisnicima olakšava testiranje performansi aplikacije. Testiranje performansi se može izvoditi na različitim preglednicima što ga čini vrlo fleksibilnim alatom.

9.2. Alati za automatizirano testiranje programskih proizvoda

Alati za automatizirano testiranje su softveru dizajnirani za testiranje programskih proizvoda na temelju automatiziranih skripti u svrhu bržeg slanja proizvoda na tržište, dugoročne uštede što se tiče troška testiranja te da bi se poboljšala kvaliteta proizvoda. Neki od alata za automatizirano testiranje koji će se pojasniti u nastavku su: Selenium, Appium, Katalon, Cypress, Postman i naravno Playwright („Katalon“, bez dat.).

Selenium je jedan od najkorištenijih alata za automatizirano testiranje web aplikacija. Neke od značajki ovog alata su te da podržava više programskih jezika za pisanje automatiziranih skripti, omogućava unakrsno testiranje na više preglednika. Preglednici koje podržava su: Chrome, Firefox, IE, Microsoft Edge, Opera, Safari. Programski jezici koje podržava su: Java, C#, Python, JavaScript, Ruby, PHP.

Appium je alat za automatizirano testiranje mobilnih aplikacija, a podržava Android i iOS sustave. Omogućava izvršavanje testova na simulatorima, emulatorima i stvarnim uređajima, a podržava programske jezike kao što su Java, C#, Python, JavaScript, Ruby, PHP, Perl.

Katalon je alat za automatizirano testiranje web aplikacija, API-a i mobilnih aplikacija. Ovaj omogućava integraciju s drugim alatima kao što je Jira ili Git. Odlikuje ga intuitivno korisničko sučelje za otklanjanje pogrešaka i izvješća o testiranju i otklanjanju pogrešaka.

Cypress je alat za automatizaciju usmjeren na testiranje web aplikacija. Podržava programski jezik JavaScript, a popularnost je stekao zbog jednostavnog i intuitivnog korisničkog sučelja. Tijekom izvršavanja testova moguće je pratiti svaki korak testa i pratiti izvršavaju li se svi zahtjevi kako je očekivano.

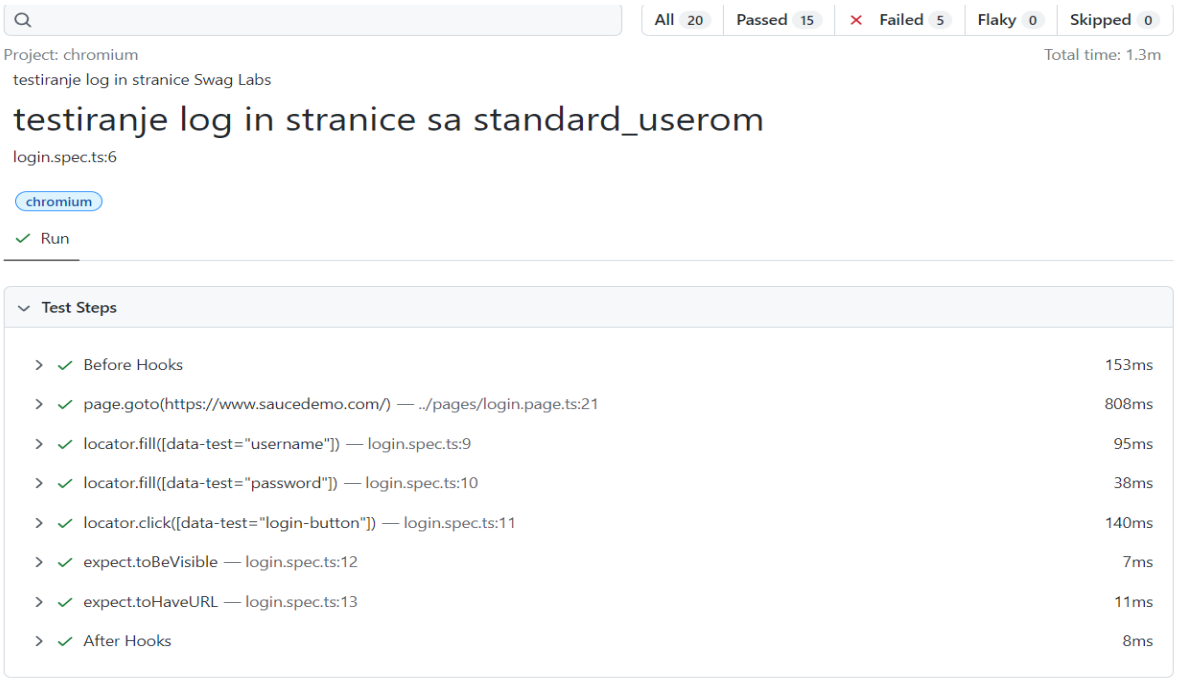
Postman je jedan od najčešće korištenih alata za automatizirano testiranje API-a. API (eng. Application Programming Inetrface), odnosno aplikacijsko programsko sučelje jest mehanizam koji omogućuje da softverske komponente komuniciraju jedna s drugom putem određenog skupa definicija i protokola. Korisnicima omogućuje pisanje različitih vrsta testova kao što su regresijski testovi. Sučelje je jednostavno za korištenje i stvaranje kolekcija testova.

9.2.1. Playwright

Playwright je relativno novi alat za automatizirano testiranje otvorenog koda, a razvijen je od strane tvrtke Microsoft. Na tržište je izašao 2020. godine, a od tada do danas je broj aktivnih korisnika dosegao brojku od 23 200 korisnika. Od prve verzije alata pa sve do danas izdano je ukupno 94 verzije što

pokazuje koliko je Microsoft aktivan u popravljanju pogrešaka i dodavanju novih značajki („Microsoft, 2023).

U vrlo kratkom vremenu Playwright je postao vrlo popularan jer omogućuje pisanje testova u različitim jezicima kao što su TypeScript, JavaScript, Python i C# te testiranje na web preglednicima kao na primjer Chrome, Firefox, Safari i Edge, što potvrđuje koliko je to fleksibilan alat. Ovaj alat podržava i testiranje na više platformi kao što su Windowsi, Linux i mac-OS. Playwright podržava funkcionalno testiranje, testiranje od kraja do kraja softvera (eng. End to end) i API testiranje te različite vrste testnih slučajeva. Navedeni alat već u sebi sadrži komponentne za izvještavanje kao što su JSON, Line, Dot ili HTML, a kreira precizna izvješća o provedenim testovima („GH, Community Contributor“, 2023). Slijedeća slika prikazuje primjer automatski generiranog izvješća u HTML-u samo za jednu funkcionalnost iako su pokrenuti svi testovi.



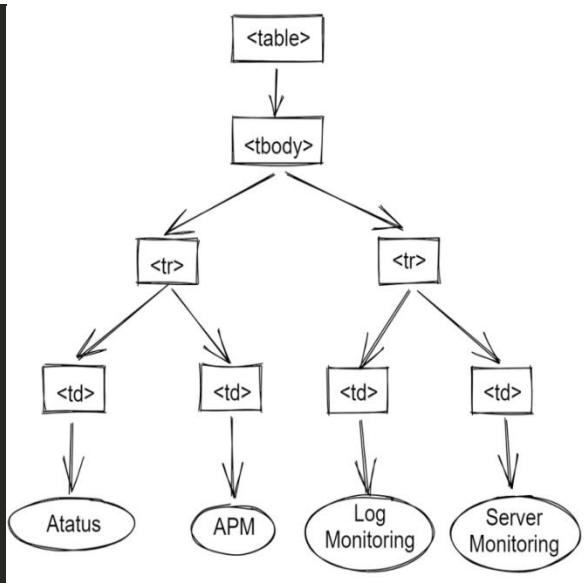
Slika 9.2.1. Izvješće o provjeri log in funkcionalnosti

Izvor: automatski generirano izvješće u alatu Playwright

Za potrebe razumijevanja automatiziranog testiranja u nastavku će se objasniti pojmovi: CSS (Cascading Style Sheets), HTML (Hypertext Markup Language), selektori, data test atributi, DOM (Document Object Model), POM (Page Object Model). Playwright omogućava automatizirano testiranje pomoću CSS i HTML selektora što je vrlo bitno jer se automatizirano testiranje ne može izvoditi bez selektora, odnosno hvatanja za elemente web stranice. CSS je kratica za „Cascading Style Sheets“, a opisuje izgled HTML dokumenta, odnosno web stranice („Ghost Inspector, 2023). HTML je standardni programski jezik za izradu web stranica te opisuje strukturu web stranice koja se sastoji od

niza elemenata. U istraživanju će se koristiti podvrsta CSS selektora pod nazivom Data test atributi kako bi se locirali elementi na web stranici. Data test atributi su vrsta selektora koji se dodjeljuju HTML elementima kako bi se tijekom automatiziranog testiranja određeni elementi web stranice jednostavnije locirali. Slijedeći izraz prikazuje primjer data test atributa *„[data-test="login-button"]“.* Kao što se može vidjeti u navedenom primjeru data test atributi imaju jedinstveni „test-id“ identifikator što olakšava lociranje elemenata web stranice. Razlog zašto je data test atribute bolje koristiti neko klasne selektore je taj što se oni postavljaju na način da je jedna vrijednost pripisana isključivo jednom elementu, dok je kod klasnih selektora slučaj da je jedan selektor moguće postaviti na više elemenata. To kod automatizacije predstavlja problem jer testovi trebaju biti vrlo precizni pa samim time selektori trebaju odgovarati samo jednom elementu na stranici. Kako bi svi ovi objašnjeni pojmovi imali smisla potrebno je objasniti još DOM i POM. DOM (Document Object Model) objektni model dokumenta opisuje strukturu web stranice, odnosno predstavlja konceptualni model web stranice za programsko sučelje koje se može implementirati (https://www.atatus.com/blog/a-beginners-guide-to-dom/#What-is-DOM?). Definira se još i kao API za programiranje HTML dokumenata jer definira logičku strukturu dokumenta i način na koji se pristupa dokumentima. Glavni cilj DOM-a je osigurati programsko sučelje koje se može koristiti u širokom rasponu aplikacija i okruženja. Struktura objektnog modela vrlo je slična strukturi dokumenata koje predstavlja, podsjećajući na stablo, a to prikazuje slijedeća slika.

```
<TABLE>
    <ROWS>
        <TR>
            <TD>Atatus</TD>
            <TD>APM</TD>
        </TR>
        <TR>
            <TD>LeadMine</TD>
            <TD>Lead Generator</TD>
        </TR>
    </ROWS>
</TABLE>
```



Slika 9.2.2. Prikaz strukture objektnog modela dokumenta

Izvor: https://www.atatus.com/blog/a-beginners-guide-to-dom/#What-is-DOM?

Potrebno je objasniti i još što je to POM (Page Object Model) odnosno objektni model stranice. Objektni modeli stranice su pristup strukturiranja testnog paketa kako bi se optimiziralo i olakšalo testiranje. Objekt stranice je dio web aplikacije koji se testira, odnosno dokument unutar kojeg se spremaju vrijednosti selektora u varijable kojima se dodjeljuju lako čitljiva imena kako bi kod bio pregledan („Microsoft, 2023). Također odgovara DRY principu (eng. Do not Repeat Yourself) što znači da ako se prilikom pisanja koda neka vrijednost koristi više puta, da se ta vrijednost spremi unutar jedne varijable te da se svaki put poziva na tu varijablu. Time se osigurava da ako postoji potreba za mijenjanjem selektora da se to obavi unutar POM dokumenta te će se ta promjena automatski vidjeti na svim mjestima gdje je ta varijabla pozvana. POM dokument može biti na primjer za prijavu u aplikaciju, stranicu za naplatu ili web shop. Na slijedećoj slici prikazan je POM za prijavu u aplikaciju „Swag Labs“ s korisnikom „standard_user“.

```
import { Locator, Page } from "@playwright/test";
export class LogIn {

    readonly page: Page;
    readonly korisnickoIme: Locator;
    readonly lozinka: Locator;
    readonly logInGumb: Locator;
    readonly porukaGreske: Locator;

    constructor(page: Page) {
     this.page = page;
     this.korisnickoIme = page.locator('[data-test="username"]');
     this.lozinka = page.locator('[data-test="password"]');
     this.logInGumb = page.locator('[data-test="login-button"]');
     this.porukaGreske = page.locator('[data-test="error"]');

    }

    async goto() {
        await this.page.goto("https://www.saucedemo.com/");
   }
}
```

Slika 9.2.3. POM za prijavu u aplikaciju „Swag Labs“

Izvor: Vlastita izrada u alatu Playwright

Nakon svega navedenog može se uvidjeti koliko je Playwright fleksibilan, razumljiv i brz alat koji testeru ostavlja mogućnost različitih vrsta automatiziranog testiranja web aplikacija.

10. VRSTE APLIKACIJA

Prema Knott (2015) aplikacija se definira kao softver koji korisnicima omogućuje razmjenu informacija i izvršavanja određenih zadataka. Postoje tri vrste aplikacija: web, izvorne i hibridne aplikacije, a razlikuju se po metodi razvoja i unutarnjoj strukturi sustava. Web aplikacije nije potrebno instalirati na uređaj, već se njima pristupa putem internetskog preglednika. Izvorne aplikacije su izrađene samo za određenu vrste platforme kao što je Android ili iOS. Hibridne aplikacije kombiniraju najbolje elemente izvornih i web aplikacija.

U početku razvoja računala, većina aplikacija je koristila arhitekturu klijent – poslužitelj. To znači da je poslužitelj lokalno obrađivao i pohranjivao podatke, a korisnici su na svoja računala morali instalirati klijentske aplikacije koje su komunicirale s poslužiteljem. Razvojem interneta poslužitelj i klijent se mogu nalaziti bilo gdje što je aplikacijama dalo veliku fleksibilnost. Da industrija razvijanja aplikacija i dalje raste potvrđuju statistike iz 2020. godine kada je ukupno preuzeto 218 bilijuna aplikacija, a za 2023. godinu se predviđa brojka od 258 bilijuna preuzimanja aplikacija na cijelome svijetu.

10.1. Web aplikacije

Web aplikacije su u osnovi web stranice kojima se pristupa putem Internet preglednika. Takve aplikacije su optimizirane za mobilne web preglednike i u potpunosti su neovisne o mobilnoj platformi. To znači da se pokreću unutar web preglednika uređaja putem URL-a, pa nije potrebno preuzeti i instalirati web aplikaciju. Web aplikacije se razvijaju putem web tehnologija kao što su HTML i JavaScript. HTML omogućuje integraciju interaktivnih elemenata na web stranicu kao što je na primjer video.

Dobra strana web aplikacija je ta što ne zahtijeva odobrenje od strane trgovine aplikacijama već se jednostavno i brzo može ažurirati na noviju verziju aplikacije. Neovisne su o mobilnim platformama i lako im se pristupa putem web preglednika bez prethodno potrebne instalacije. Web aplikacije su jeftinije i brže za razvijanje od hibridnih i nativnih aplikacija. S druge strane iako su jeftinije i brže za razvijanje, web aplikacije imaju ograničen pristup značajkama uređaja, kao što su kamera ili mikrofon. Kako bi neometano radile cijelo vrijeme moraju biti povezane s Internetom, a preuzimanje nekih datoteka ili videa može potrajati. Također, još jedan nedostatak je taj što korisnici ne mogu pronaći aplikaciju u trgovini s aplikacijama što može uzrokovati manjak korisnika. Osim toga, web aplikacije mogu imati veći trošak održavanja zbog različitih preglednika na kojima se zasebno moraju održavati („Yeeply“, 2021).

10.2. Izvorne aplikacije

Izvorne aplikacije se razvijaju s posebnim programskim jezikom za specifičnu mobilnu platformu kao što je na primjer Android ili iOS. Tako se aplikacije za Android razvijaju u programskom jeziku Java, a aplikacije za iOS se razvijaju u programskom jeziku Swift. Zbog ovakvog načina razvoja izvorne aplikacije imaju pristup svim značajkama operativnog sustava na kojem su instalirane (Knott, 2015).

Može se reći da izvorne aplikacije imaju najbolju izvedbu budući da su izrađene i optimizirane samo za jednu vrstu platforme, pa time aplikacija postiže visoku razinu performansi. Izvorne aplikacije nude bolje korisničko sučelje jer dobivaju sučelje od operativnog sustava svog uređaja kako bi izgledale i funkcionirale kao dio uređaja, a osim toga rade i s ugrađenim značajkama uređaja kao što su GPS, mikrofon ili kamera. Može se reći da je tijekom razvijanja izvornih aplikacija manja mogućnost pogreške jer se koncentrira na razvoj za samo jednu platformu. Izvorne aplikacije se smatraju i sigurnijima u smislu zaštite korisničkih podataka, a u većini slučajeva ovakva vrsta aplikacija ne mora imati kontinuiranu internetsku vezu („Yeeply“, 2023). Pored svih prednosti izvornih aplikacija nameću se i nedostaci u smislu troška razvoja, jer se aplikacija mora odvojeno razvijati za svaki operativni trošak, a samim time se povećavaju i troškovi održavanja same aplikacije.

10.3. Hibridne aplikacije

Hibridne aplikacije kombiniraju najbolje elemente izvornih i web aplikacija. Koriste različite web tehnologije kao što su HTML ili JavaScript, a jednom kada se razvije web dio aplikacije programeri kreću na razvoj izvornog dijela aplikacije. Kada je aplikacija spremna za korištenje, ona se instalira kao izvorna aplikacija, ali unutar aplikacije se nalazi web aplikacija (Knott, 2015).

Za razliku od izvornih aplikacija hibridne aplikacije nije potrebno razvijati zasebno za Android i iOS već se izrađuje jedna aplikacija koja može raditi na oba operativna sustava. Na taj način se mogu uštedjeti vrijeme i novac. Hibridne aplikacije mogu pristupati značajkama uređaja, a ažuriranja same aplikacije su jednostavna za izvođenje. S druge strane hibridne aplikacije zahtijevaju razvoj aplikacija visoke kvalitete kako bi ponudile što bolje korisničko iskustvo svim korisnicima. Hibridne aplikacije nikad ne mogu ponuditi potpuno korisničko iskustvo svim korisnicima jer će uvijek postojati neke sitne razlike u korisničkom iskustvu u usporedbi s izvornim aplikacijama.

11. ISTRAŽIVANJE

Cilj ovog istraživanja je na primjeru web aplikacije „Swag Labs“ predstaviti ručno i automatizirano testiranje te prikazati sve prednosti i nedostatke navedenih testiranja. Aplikacija je izrađena od strane tvrtke „Sauce Labs“ u svrhu razvijanja testerskih vještina u domeni ručnog i automatiziranog testiranja, a dostupna je na “https://www.saucedemo.com/“. „Swag Labs“ predstavlja fiktivnu online trgovinu gdje korisnik može dodati stavke u košaricu, izraditi narudžbu, filtrirati proizvode, micati dodane proizvode iz košarice, prijaviti se u aplikaciju, izvršiti naplatu…

Istraživanjem će se spoznati efikasnost automatiziranog i ručnog testiranja u okviru osiguravanja da su sve najvažnije funkcionalnosti aplikacije pravilno testirane. Prilikom izvođenja testnih slučajeva uvidjet će se prednosti i nedostaci korištenih pristupa, a posebno prilikom rubnih testnih slučajeva kao što je testiranje korisničkog iskustva. Glavne metrike na temelju kojih će se donositi zaključci o efikasnosti i brzini jednog i drugog pristupa jest vrijeme koje je potrebno za izvođenje testnih slučajeva i vrijeme koje je uloženo u izradu istih tih testnih slučajeva. Dakle, uspoređivat će se vrijeme potrebno za izradu testnih skripti kod automatiziranog i ručnog testiranja te vrijeme izvođenja testnih slučajeva za obje metode. Istraživanjem će se prikazati razlika između izrade testnih skripti za automatizirano i ručno testiranje te će se ustanoviti koliko je u osnovi jednostavno ili komplicirano izraditi testne skripte za jednu i drugu metodu testiranja.

U skladu s istraživanjem i njegovom svrhom, na temelju dobivenih rezultata prihvatit će se ili opovrgnuti slijedeća istraživačka pitanja:

IP 1: Automatizirano testiranje je brže i efikasnije od ručnog testiranja.

IP 2: Ručno testiranje je korisnije nego automatizirano prilikom testiranja korisničkog iskustva.

IP 3: Kombinacija ručnog i automatiziranog testiranja najefektivniji je način osiguravanja kvalitete programskog proizvoda.

Kako bi se istražile prednosti i nedostaci automatiziranog i ručnog testiranja te da bi se dao odgovor na postavljena istraživačka pitanja za svaki pristup koristit će se metoda studije slučaja i eksperimentalna metoda. Metoda studije slučaja će se koristiti jer se temelji na opisivanju jednog ili nekoliko specifičnih slučajeva određenih karakteristika, koji se detaljno analiziraju kako bi se na temelju dobivenih zaključaka istraživačka pitanja prihvatila ili odbila. U ovom slučaju detaljno će se analizirati rezultati testnih slučajeva za pojedine funkcionalnosti aplikacije „Swag Labs“ prilikom korištenja ručnog i automatiziranog testiranja. U kombinaciji s navedenom metodom će se koristiti i eksperimentalna metoda gdje predmet eksperimenta predstavlja testiranje aplikacije „Swag Labs“ ručnim i

automatiziranim testiranjem, a sredstva pomoću kojih će se obavljati eksperiment su testni slučajevi za ručno i automatizirano testiranje te alat Playwright.

Za bolje razumijevanje istraživanja u nastavku će se dati definicija testnog slučaja. Testni slučaj se definira kao detaljan skup smjernica ili radnji koje se provode u svrhu testiranja određene funkcionalnosti softvera. Testni slučaj mora sadržavati detaljne korake koje je potrebno slijediti da bi se testni slučaj pravilno izvršio. Osim toga mora sadržavati i informacije o testnim podacima, opis cijelog scenarija, očekivani rezultat te preduvjete potrebne za izvođenje testiranja.

Za potrebe ručnog testiranja testni slučajevi će se izraditi u programu Excel u obliku tablice koja će sadržavati sve navedene elemente. S druge strane, kod automatiziranog testiranja testni slučajevi će izgledati nešto drugačije, iako će i dalje sadržavati sve ključne elemente testnog slučaja. Dakle, automatizirane testne skripte će biti napisane u kodu koristeći programski jezik Typescript. Typescript se definira kao podset JavaScript programskog jezika zbog čega se Typescript prije izvršavanja u pregledniku prvo prevodi u JavaScript kod, a to se izvršava pomoću npm-a. Npm (Node Package Manager) se definira kao sustav koji upravlja JavaScript paketima. Uz pomoć npm-a će se skinuti paket koji je sastavljen od JavaScript koda koji daje definiciju kako koristiti Playwright. Playwright je alat za automatizirano testiranje izrađen od strane tvrtke Microsoft, a već je pojašnjen u poglavlju osam.

11.1. Ručno testiranje web aplikacije „Swag Labs“

Ručno testiranje aplikacije „Swag Labs“ izvodit će se pomoću dvadeset dva testna slučaja. Testni slučajevi sadržavat će slijedeće komponente: modul (dio aplikacije koji se testira), naslov testnog slučaja, opis testnog slučaja, preduvjet za izvođenje testnog slučaja, koraci potrebni za izvođenje testa, testni podaci, očekivani rezultat, prioritet, odnosno važnost prolaska testa kako bi aplikacija radila onako kako je očekivano, na kraju rezultat testiranja za korisnika „standard_user“, rezultat testiranja za korisnika „problem_user“ i testne bilješke.

Testovi će biti podijeljeni na slijedećih šest modula: „prijava“ (eng. log in), „odjava“ (eng. log out), „košarica“, „narudžba“, „online trgovina“ (eng. web shop), te na kraju modul „menu. Svi testovi će se testirati s dva korisnika „standard_user“ i „ problem_user“. U nastavku će se prikazati svi testni slučajevi kojima će se ručnom metodom testirati aplikacija „Swag Labs“ čime će se dati odgovor na postavljena istraživačka pitanja. Svi testovi su prikazani i u prilogu na kraju rada.

Slika 11.1.1 prikazuje sve testne slučajeve za modul „prijava“ (eng. log in). Prikazanim testnim slučajevima se ispituje mogućnost prijave u aplikaciju „Swag Labs“ s korisničkim imenima

„standard_user“, „problem_user“ i „locked_out_user“ te lozinkom „secret_sauce“. Osim toga testirat će se i mogućnost prijave s korisničkim imenom i lozinkom koji nisu prihvaćeni od strane aplikacije „Swag Labs“. Preduvjet za izvršavanje testnih slučajeva je odlazak na „https://www.saucedemo.com/“ te nakon toga izvršavanje testnih koraka. Očekivani rezultat testova sa prihvaćenim korisničkim imenima je da se korisnik uspješno prijavio u aplikaciju, a prioritet prolaska testova je visok, jer bez prijave u aplikaciju nije moguće koristiti niti jednu njezinu funkcionalnost. Očekivani rezultati testova za korisnike „standard_user“, „problem_user“ i „locked_out_user“ su prolaz, a očekivani rezultat za korisničko ime koje nije prihvaćeno od strane „Swag Labs“ je da test ne prođe. Kao što se može vidjeti na slici 11.1.1 testovi za korisnička imena „problem_user“ i „standard_user“ su prošli kako je i očekivano, a test za „locked_out_user“ nije prošao, što nije bilo očekivano. Test s korisničkim imenom koje nije prihvaćeno „Ana123!! te s lozinkom „Ana123!“ je prošao, u smislu da je rezultat u skladu s očekivanim rezultatom, odnosno korisnik se nije uspio ulogirati i pojavila se poruka o pogrešci.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_user)	Rezultat (problem_user)	Bilješke
Prijava (Log in)	Prijava sa imenom "standard_user" i lozinkom "secret_sauce"	Provjerite je li se moguće prijaviti s imenom "standard_user" i lozinkom "secret_sauce"	Idite na: https://www.saucedemo.com/	1.	Unesite ime "standard_user" u tekstni okvir "username"	Ime: standard_user	Korisnik je prijavljen - https://www.saucedemo.com/inventory.html	Visok	Prolaz		
				2.	Unesite lozinku "secret_sauce" u tekstni okvir "password"	Lozinka: secret_sauce					
				3.	Kliknite na "Log in" gumb						
Prijava (Log in)	Prijava sa imenom "locked_out_user" i lozinkom "secret_sauce"	Provjerite je li se moguće prijaviti s imenom "locked_out_user" i lozinkom "secret_sauce"	Idite na: https://www.saucedemo.com/	1.	Unesite ime "locked_out_user" u tekstni okvir "username"	Ime: locked_out_user	Korisnik je prijavljen - https://www.saucedemo.com/inventory.html	Visok			Test nije prošao za korisnika locked_out_user
				2.	Unesite lozinku "secret_sauce" u tekstni okvir "password"	Lozinka: secret_sauce					
				3.	Kliknite na "Log in" gumb						
Prijava (Log in)	Prijava sa imenom "problem_user" i lozinkom "secret_sauce"	Provjerite je li se moguće prijaviti s imenom "problem_user" i lozinkom "secret_sauce"	Idite na: https://www.saucedemo.com/	1.	Unesite ime "problem_user" u tekstni okvir "username"	Ime: problem_user	Korisnik je prijavljen - https://www.saucedemo.com/inventory.html	Visok		Prolaz	
				2.	Unesite lozinku "secret_sauce" u tekstni okvir "password"	Lozinka: secret_sauce					
				3.	Kliknite na "Log in" gumb						
Prijava (Log in)	Prijava s podacima koji nisu prihvaćeni kao korisničko ime i lozinka za https://www.saucedemo.com/	Provjerite da se nije moguće prijaviti s korisničkim imenima i lozinkama koji nisu prihvaćeni na https://www.saucedemo.com/"	Idite na: https://www.saucedemo.com/	1.	U tekstualni okvir "username" upišite "Ana123!"	Ime: Ana123!	Korisnik nije prijavljen i pojavljuje se poruka o pogrešci "Korisničko ime i lozinka ne odgovaraju nijednom korisniku u ovoj usluzi"	Visok			Test je prošao, pojavila se poruka o pogrešci.
				2.	U tekstualni okvir "password" upišite "Ana123!"	Lozinka: Ana123!					
				3.	Kliknite na "Log in" gumb						

Slika 11.1.1. Prijava u aplikaciju „Swag Labs“

Izvor: Vlastita izrada

Slika 11.1.2. prikazuje testni slučaj koji se odnosi na modul „odjava“ (eng. log out), a testirati će se s korisničkim imenima „standard_user“ i „problem_user“ koji su uspješno prošli testove za prijavu u aplikaciju. Preduvjet za izvršavanje navedenog testa je odlazak na „https://www.saucedemo.com/“ te prijava u aplikaciju s korisničkim imenom za koji se provodi test odjave s korisničkog računa. Test je za oba korisnička računa uspješno prošao, onako kako je i očekivano.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_user)	Rezultat (problem user)	Bilješke
Odjava (Log out)	Odjava s korisničkog računa	Provjerite je li se moguće odjaviti s korisničkog računa	Idite na: https://www.saucedemo.com/	1.	Kliknite na izbornik u gornjem lijevom kutu	Korisnik1	Korisnik je odjavljen - https://www.saucedemo.com/	**Visok**	**Prolaz**	**Prolaz**	
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na "Log out" gumb	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Slika 11.1.2. Odjava iz aplikacije „Swag Labs“

Izvor: Vlastita izrada

Slijedeći set testnih slučajeva se odnosi na modul „košarica“. Testni slučajevi navedenog modula su visokog prioriteta jer se radi o aplikaciji koja predstavlja online trgovinu, a ako se proizvodi ne mogu staviti u košaricu, aplikacija nema smisla. Preduvjeti za izvršavanje svih šest testova su odlazak na „https://www.saucedemo.com/“ i prijava s korisničkim računom, a ostali preduvjeti će se mijenjati po potrebi testa.

Prvi od šest testova koja će se izvršiti, odnosi se na testiranje mogućnosti stavljanja svih proizvoda u košaricu. Za korisnika „standard_user“ test je prošao i ostvaren je očekivani rezultat da su svi proizvodi uspješno dodani u košaricu. Za korisnika „problem_user“ očekivani rezultat nije postignut jer nije moguće dodati sve proizvode u košaricu, stoga test nije prošao.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_user)	Rezultat (problem user)	Bilješke
Košarica	Dodavanje svih proizvoda u košaricu s korisnicima "standard_user" i "problem_user"	Provjerite je li moguće dodavanje svih proizvoda u košaricu s korisnicima "standard_user" i "problem_user"	Idite na: https://www.saucedemo.com/	1.	Kliknite na gumb "Add to cart" za svaki proizvod	Korisnik1	Svi proizvodi su uspješno dodani u košaricu	**Visok**	**Prolaz**	**Test nije prošao**	Neki proizvodi kod korisnika "problem_user" se ne mogu dodati u košaricu
			Ulogirajte se s korisnikom: standard_user	2.	Klikom na gum "Košarica" u gornjem desnom kutu provjerite na	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Slika 11.1.3. Dodavanje svih proizvoda u košaricu

Izvor: Vlastita izrada

Drugi test za modul „košarica“ se odnosi na testiranje uklanjanja svih proizvoda iz košarice s korisnicima „standard_user“ i „problem_user“. Preduvjet za izvršavanje testa, osim odlaska na web aplikaciju i prijave na korisnički račun, je i dodavanje što više proizvoda u košaricu kako bi se kasnije mogla testirat funkcionalnost micanja proizvoda iz košarice putem gumba „makni“ (eng. remove) koji se nalazi u košarici. Kod korisnika „standard_user“ svi proizvodi su uspješno uklonjeni iz košarice pa je test prošao, dok se kod korisnika „problem_user“ određeni proizvodi nisu mogli ukloniti iz košarice pa test nije prošao. Navedeni test je opisan na slici 11.1.4.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_ user)	Rezultat (problem user)	Bilješke
Košarica	Uklanjanje svih proizvoda iz košarice s korisnicima "standard_user" i "problem_user"	Provjerite je li moguće uklanjanje svih proizvoda iz košarice s korisnicima "standard_user" i "problem_user" klikom na gumb "Remove" unutar košarice	Idite na: https://www.saucedemo.com/	1.	Kliknite na košaricu						
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "Remove" kako biste uklonili proizvode iz košarice	Korisnik1	Svi proizvodi su uspješno uklonjeni iz košarice	**Visok**	**Prolaz**	**Test nije prošao**	Kod korisnika problem_user neki proizvodi se ne mogu ukloniti iz košarice klikom na gumb "Remove"
			Ulogirajte se s korisnikom: problem_user			Ime: standard_user					
			Dodajte sve proizvode u košaricu			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Slika 11.1.4. Testiranje mogućnosti uklanjanja proizvoda iz košarice putem gumba „makni“ unutar košarice
Izvor: Vlastita izrada

Posljednji test za modul „košarica“ se odnosi na mogućnost uklanjanja svih proizvoda iz košarice s korisnicima „standard_user“ i „problem_user“ klikom na gumb „makni“ (eng. remove) koji se nalazi na naslovnoj stranici online trgovine. Preduvjeti za izvršavanje testa su jednaki kao u prethodnom testu. Kod korisnika „standard_user“ svi proizvodi su uspješno uklonjeni iz košarice, dok kod korisnika „problem_user“ nisu svi proizvodi mogli biti uklonjeni iz košarice klikom na gumb „makni“. Navedeni test prikazan je na slici 11.1.5.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_user)	Rezultat (problem user)	Bilješke
Košarica	Uklanjanje svih proizvoda iz košarice s korisnicima "standard_user" i "problem_user"	Provjerite je li moguće uklanjanje svih proizvoda iz košarice s korisnicima "standard_user" i "problem_user" klikom na gumb "Remove" na naslovnoj stranici web shop-a	Idite na: https://www.saucedemo.com/	1.	Kliknite na gumb "Remove", na naslovnoj stranici web shop-a, za svaki proizvod koji je dodan u košaricu	Korisnik1	Svi proizvodi su uspješno uklonjeni iz košarice	**Visok**	**Prolaz**	**Test nije prošao**	Kod korisnika "problem_user" proizvodi nisu uklonjeni iz košarice klikom na gumb "Remove" na naslovnoj stranici web shop-a.
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
			Dodajte sve proizvode u košaricu			Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Slika 11.1.5. Testiranje mogućnosti uklanjanja proizvoda iz košarice putem gumba „makni“ na naslovnoj stranici online trgovine

Izvor: Vlastita izrada

Modul „narudžba“ je sljedeći na redu za testiranje, a sadrži pet testnih slučajeva. Preduvjet za izvršavanje slijedećih testova je odlazak na „https://www.saucedemo.com/“ i prijava s korisnikom „standard_user“ ili „problem_user“.

Prvi od deset testova se odnosi na testiranje nemogućnosti dovršetka prazne narudžbe. To znači da je očekivani rezultat testa taj da je nemoguće dovršiti praznu narudžbu. Kod korisnika „standard_user“ test nije prošao jer je praznu narudžbu moguće završiti. S druge strane, kod korisnika „problem_user“ test također nije prošao, ali iz drugog razloga. Dakle, kod spomenutog korisnika nije moguće u cijelosti popuniti informacijski obrazac o kupcu, nije moguć unos vrijednosti u polje „prezime“, a bez toga se ne može nastaviti na dovršetak narudžbe. Test je prikazan na slici 11.1.6.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_user)	Rezultat (problem user)	Bilješke
Narudžba	Prazna narudžba	Provjerite je li nemoguće dovršiti praznu narudžbu.	Idite na: https://www.saucedemo.com/	1.	Kliknite na praznu košaricu	Korisnik1	Trebalo bi biti nemoguće dovršiti praznu naridžbu.	**Visok**	**Test nije prošao**	**Test nije prošao**	Kod korisnika standard_user moguće je dovršiti praznu narudžbu. Kod korisnika problem_user nije moguće popuniti polje "prezime".
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "checkout"	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user	3.	Unesite informacije o kupcu (checkout information)	Lozinka: secret_sauce					
				4.	Kliknite na gumb "Continue"	Checkout information: First name: Ana Last name: Milin Zip/postal code: 21000					
				5.	Kliknite na gumb "Finish"						

Slika 11.1.6. Testiranje nemogućnosti dovršetka prazne narudžbe

Izvor: Vlastita izrada

Drugi test u okviru modula košarica ispituje hoće li podaci o kupnji ostati spremljeni ako kupac popuni informacijski obrazac i ode, sa „https://www.saucedemo.com/checkout-step-two.html“ na „https://www.saucedemo.com/checkout-step-one.html“. Očekivani rezultat je da podaci ostanu spremljeni, međutim kod oba korisnika, „standard_user“ i „problem_user“ podaci nisu ostali spremljeni, stoga test nije prošao.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_user)	Rezultat (problem user)	Bilješke
Narudžba	Spremljeni podaci o kupcu i naplati pri povratku od - https://www.saucedemo.com/checkout-step-two.html do - https://www.saucedemo.com/checkout-step-one.html"	Provjerite hoće li podaci o kupnji biti spremljeni kada se kupac vrati s - https://www.saucedemo.com/checkout-step-two.html na - https://www.saucedemo.com/checkout-step-one.html"	Idite na: https://www.saucedemo.com/	1.	Kliknite na košaricu	Korisnik1	Podaci o kupcu i naplati bi trebali ostati spremljeni pri povratku s https://www.saucedemo.com/checkout-step-two.html	**Srednji**	**Test nije prošao**	**Test nije prošao**	Podaci o kupcu i naplati ne ostanu spremljeni pri povratku s https://www.saucedemo.com/checkout-step-two.html.
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "checkout"	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user	3.	Unesite informacije o kupcu (checkout information)	Lozinka: secret_sauce					
				4.	Kliknite na lijevu strelicu kako biste došli natrag na - https://www.saucedemo.com/	Checkout information: First name: Ana Last name: Milin Zip/postal code: 21000					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Slika 11.1.7. Testiranje mogućnosti spremanja podataka o kupcu

Izvor: Vlastita izrada

Treći test je prikazan na slici 11.1.8., a odnosi se na testiranje informacijskog obrasca u smislu prepoznavanja podataka koji nisu validni za određeno polje. Očekivani rezultat testa je da obrazac prepozna podatke koji nisu validni te da se pojavi poruka o pogrešci, a da kupac ne može nastaviti na dovršetak narudžbe. Kod korisnika „standard_user“ test nije prošao jer obrazac nije prepoznao neispravne podatke, poruka o pogrešci se nije pojavila te je korisnik mogao nastaviti na završetak narudžbe. Kod korisnika „problem_user“ test nije prošao jer nije moguće popuniti polje „prezime“ u informacijskom obrascu.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_ user)	Rezultat (problem user)	Bilješke
Narudžba	Informacijski obrazac prepoznaje iste podatke u svakom polju kao one koji nisu validni.	Provjerite je li informacijski obrazac prepoznaje iste podatke u svakom polju kao one koji nisu validni.	Idite na: https://www.saucedemo.com/	1.	Kliknite na košaricu	Checkout information: First name: Ana!23 Last name: Ana!23 Zip/postal code: Ana!23	Pojavljuje se poruka o pogrešci "Podaci o naplati nisu valjani" i kupac ne može nastaviti na sljedeći korak	**Visok**	**Test nije prošao**	**Test nije prošao**	Poruka o pogrešci "Podaci nisu valjani" se nije pojavila i kupac može nastaviti na slijedeći korak. Kod korisnika problem_user nije moguće popuniti polje "prezime".
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "checkout"	Korisnik1					
				3.	Unesite informacije o kupcu/naplati, isti podatak u svako polje	Ime: standard_user					
				4.	Kliknite na gumb "continue"	Lozinka: secret_sauce					

Slika 11.1.8. Testiranje informacijskog obrasca s neispravnim podacima

Izvor: Vlastita izrada

Četvrti testni slučaj ispituje mogućnost popunjavanja informacijskog obrasca i dovršetak narudžbe. Očekivani rezultat je uspješno popunjavanje obrasca i dovršetak narudžbe, što i je slučaj kod korisnika „standard_user“. Kod korisnika „problem_user“ nije moguće popuniti polje prezime u informacijskom obrascu, stoga test nije prošao. Opisani test je prikazan na slici 11.1.9.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_ user)	Rezultat (problem user)	Bilješke
Narudžba	Ispunjavanje informacijskog obrasca i dovršetak narudžbe	Provjerite je li moguće uspješno ispunjavanje informacijskog obrasca i dovršetak na rudžbe.	Idite na: https://www.saucedemo.com/	1.	Kliknite na košaricu	Checkout information: First name: Ana Last name: Milin Zip/postal code: 21000	Informacijski obrazac je uspješno ispunjen i kupac može dovršiti narudžbu.	**Visok**	**Prolaz**	**Test nije prošao**	Kod korisnika problem_user nije moguće ispuniti polje prezime (Last Name) u informacijskom obrazcu, pa nije moguće niti dovršiti narudžbu.
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "checkout"	Korisnik1					
			Ulogirajte se s korisnikom: problem_user	3.	Unesite informacije o kupcu	Ime: standard_user					
			Dodajte proizvode u košaricu	4.	Kliknite na gumb "continue"	Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Slika 11.1.9. Testiranje mogućnosti popunjavanja informacijskog obrasca i dovršetka narudžbe

Izvor: Vlastita izrada

Posljednji test za modul „narudžba“ odnosi se na provjeru točnosti ukupne sume narudžbe na računu. Kako bi se test mogao izvršiti, osim prijave u aplikaciju potrebno je prvo dodati nekoliko proizvoda u košaricu. Nakon toga slijedi popunjavanje podataka o kupcu i provjera točnosti sume na računu. Za korisnika „standard_user“ test je prošao, ali za korisnika „problem_user“ test nije prošao zbog nemogućnosti popunjavanja polja „prezime“ u informacijskom, obrascu.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_ user)	Rezultat (problem user)	Bilješke
Narudžba	Ukupna suma narudžbe na računu je točna.	Provjerite je li ukupna suma narudžbe na računu točna.	Idite na: https://www.saucedemo.com/	1.	Kliknite na košaricu	Checkout information: First name: Ana Last name: Milin Zip/postal code: 21000	Ukupna suma narudžbe na računu je točna.	**Visok**	**Prolaz**	**Test nije prošao**	
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "checkout"	Korisnik1					
			Ulogirajte se s korisnikom: problem_user	3.	Unesite informacije o kupcu	Ime: standard_user					
			Dodajte proizvode u košaricu	4.	Kliknite na gumb "continue"	Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

11.1.10. Provjera točnosti ukupne sume narudžbe na računu

Izvor: Vlastita izrada

Sljedeći set testova se odnosi na modul „online trgovina“ (eng. web shop), a sadrži šest scenarija koji se testiraju s dva korisnika što je ukupno dvanaest testnih slučajeva. Preduvjeti za sljedeće testove su odlazak na „https://www.saucedemo.com/“te prijava s korisničkim računom „standard_user“ ili „problem_user“.

Prvi test za modul „online trgovina“ ispituje relevantnost slike proizvoda i opisa proizvoda, odnosno je li slika prikazuje ono što stoji u opisu proizvoda. Prioritet ovog testa je visok, jer ako slika proizvoda nije ispravna kupci neće kupovati proizvode. Dakle, očekivani rezultat je da slika svakog proizvoda odgovara njegovom opisu. Za korisnika „standard_user“ ovaj test je prošao, dok za korisnika „problem_user“ test nije prošao jer se umjesto slika proizvoda pojavljuju slike psa.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_ user)	Rezultat (problem user)	Bilješke
Online trgovina (Web shop)	Sve slike proizvoda točno prikazuju opisani proizvod	Provjerite je li sve slike proizvoda točno prikazuju opisani proizvod	Idite na: https://www.saucedemo.com/	1.	Pogledajte sliku svakog proizvoda	Korisnik1	Slika svakog proizvoda odgvora opisu tog proizvoda.	**Visok**	**Prolaz**	**Test nije prošao**	Kod korisnika problem_user umjesto slika proizvoda, vidljive su samo slike psa.
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user	.		Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Slika 11.1.11. Testiranje relevantnosti slike i opisa proizvoda

Izvor: Vlastita izrada

Na slici 11.1.12 prikazani su ostali testovi za online trgovinu. Kroz testove vidljivosti ikone košarice, vidljivosti filtera, cijene, naziva, opisa i slike proizvoda, naziva „Swag Labs“ i ikone menu, rađena je vizualna regresija. Za „standard_usera“ svih pet testova prolazi dok za „problem_usera“ nije prošao test vidljivosti slike proizvoda jer ne odgovara naslovu, opisu i cijeni proizvoda.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_user)	Rezultat (problem_user)	Bilješke
Online trgovina (Web shop)	Testiranje vidljivosti ikone košarica	Provjerite je li ikona košarice vidljiva na stranici web shop-a	Idite na: https://www.saucedemo.com/	1.	Pogledajte nalazi li se ikona košarice u gornjem desnom kutu	Korisnik1	Ikona košarice je vidljiva.	**Visok**	**Prolaz**	**Prolaz**	
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Online trgovina (Web shop)	Testiranje vidljivosti naziva "Swag Labs"	Provjerite je li naziv"Swag Labs" vidljiv na stranici web shop-a	Idite na: https://www.saucedemo.com/	1.	Pogledajte nalazi li se naslov "Swag Labs" gore, na sredini web shop-a.	Korisnik1	Naslov "Swag Labs" je vidljiv.	**Visok**	**Prolaz**	**Prolaz**	
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Online trgovina (Web shop)	Testiranje vidljivosti ikone "Menu"	Provjerite je li ikona "Menu" vidljiva na stranici web shop-a	Idite na: https://www.saucedemo.com/	1.	Pogledajte nalazi li se ikona "Menu" u gornjem lijevom kutu.	Korisnik1	Ikona "Menu" je vidljiva.	**Visok**	**Prolaz**	**Prolaz**	
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Online trgovina (Web shop)	Testiranje vidljivosti slike proizvoda, cijene proizvoda, naziva i opisa proizvoda te gumba "Add to cart" na jednoj kartici proizvoda.	Provjerite jesu li svi elementi kartice jednog proizvoda vidljivi, slika proizvoda, cijena proizvoda, naziv i opisa proizvoda te gumb "Add to cart"	Idite na: https://www.saucedemo.com/	1.	Pogledajte je li vidljiva slika proizvoda na jednoj kartici.	Korisnik1	Na jednoj kartici su vidljivi slika proizvoda, cijena proizvoda, naziv proizvoda, opisa proizvoda i gumb "Add to cart".	**Visok**	**Prolaz**	**Test nije prošao**	Kod problem_user su umjesto slika proizvoda vidljive slike psa.
			Ulogirajte se s korisnikom: standard_user	2.	Pogledajte je li na istoj kartici vidljiv naziv proizvoda.	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user	3.	Pogledajte je li na istoj kartici vidljiv opis proizvoda.	Lozinka: secret_sauce					
				4.	Pogledajte je li na istoj kartici vidljiva cijena proizvoda.	Korisnik2					
				5.	Pogledajte je li na istoj kartici vidljiv "Add to cart" gumb.	Ime: problem_user Lozinka: secret_sauce					
Online trgovina (Web shop)	Testiranje vidljivosti filtera	Provjerite je li filter vidljiv u gornjem desnom kutu	Idite na: https://www.saucedemo.com/	1.	Pogledajte nalazi li se filter u gornjem desnom kutu.	Korisnik1	Filter je vidljiv i nalazi se u gornjem desnom kutu.	**Srednji**	**Prolaz**	**Prolaz**	
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Slika 11.1.12. Testiranje vidljivosti ikone košarica, menu, filter, cijena, opis, naziv proizvoda, naziv „Swag Labs“

Izvor: Vlastita izrada

Posljednji set testova se odnosi na modul „menu“ koji sadrži ukupno šest testova, odnosno tri scenarija od kojih se svaki izvršava za korisnike „standard_user“ i „problem_user“.

Prvi test ispituje ispravnost gumba „Svi proizvodi“ (eng. All items), odnosno hoće li se nakon klika na gumb izlistati svi proizvodi u online trgovini. Prioritet testa je visok, jer ako se ne mogu prikazati svi proizvodi, onda ih kupac ne može niti kupiti. Test je uspješno prošao za oba korisnika, što se može vidjeti i na slici 11.1.13.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_user)	Rezultat (problem user)	Bilješke
Menu	Nakon što se klikne na gumb "All Items" unutar menua, svi proizvodi će biti izlistani.	Provjerite je li se izlistaju svi proizvodi nakon što se klikne na gumb "All Items" unutar menua.	Idite na: https://www.saucedemo.com/	1.	Kliknite na ikonu menua u gornjem lijevom kutu	Korisnik1	Nakon klika na gumb "All Items" unutar menua, svi proizvodi će biti izlistani.	Visok	Prolaz	Prolaz	(Ako se u obzir uzima screenshot slika proizvoda test će pasti.)
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "All Items"	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Slika 11.1.13. Provjera ispravnosti gumba „Svi proizvodi“ unutar menu-a

Izvor: Vlastita izrada

Sljedeći test se odnosi na ispitivanje je li gumb „O nama“ (eng. About) vodi na stranicu „https://saucelabs.com/“. Dakle, očekivani rezultat je da je nakon klika na gumb korisnik uspješno preusmjeren na spomenutu stranicu. Prioritet ovog testa je nizak jer ova funkcionalnost nije nužna za dovršetak kupovine što je i svrha aplikacije za online trgovinu. Za korisnika „standard_user“ test je uspješno prošao, dok za korisnika „problem_user“ test nije prošao, jer se nakon klika na gumb „O nama“ korisnik preusmjerio na „https://saucelabs.com/error/404“.

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_user)	Rezultat (problem user)	Bilješke
Menu	Gumb "About", unutar menua vodi na stranicu https://saucelabs.com/	Provjerite je li gumb "About" unutar menua vodi na stranicu https://saucelabs.com/	Idite na: https://www.saucedemo.com/	1.	Kliknite na ikonu menua u gornjem lijevom kutu	Korisnik1	Nakon klika na gumb "About" korisnik je uspješno preusmjeren na https://saucelabs.com/	Nizak	Prolaz	Test nije prošao	Kod korisnika problem_user gumb "About" vodi na https://saucelabs.com/error/404
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "About"	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Slika 11.1.14. Testiranje gumba „O nama“ unutar menu-a

Izvor: Vlastita izrada

Posljednji test se odnosi na testiranje gumba „Resetiranje stanja aplikacije“ (eng. Reset App State) unutar menu-a. Prioritet ovog testa je visok jer kupac mora biti u mogućnosti sve vratiti na početno

stanje kako bi lakše mogao kupovati proizvode. Kod korisnika „standard_user“ i „problem_user“ klikom na gumb stanje aplikacije stanje aplikacije tek djelomično resetiralo, stoga test nije prošao.

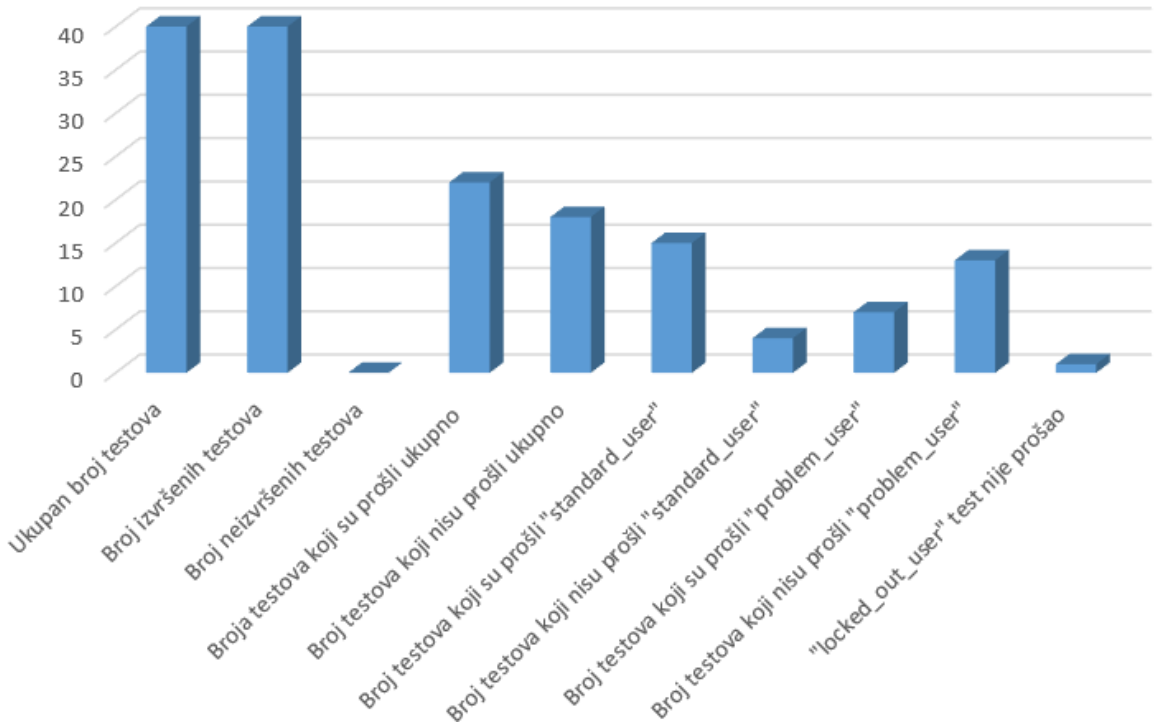
Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_ user)	Rezultat (problem user)	Bilješke
Menu	Klikom na gumb "Reset App State" unutar menua, sve se postavke računa resetiraju.	Provjerite je li se klikom na gumb "Reset App State" unutar menua, sve postavke računa resetiraju.	Idite na: https://www.saucedemo.com/	1.	Kliknite na ikonu menua u gornjem lijevom kutu	Korisnik1	Nakon klika na gumb "Reset State App" sve postavke korisničkog računa su resetirane.	**Visok**	**Test nije prošao**	**Test nije prošao**	Nakon klika na gumb "Reset App State" samo dio postavki je resetiran. Proizvodi su maknuti iz košarice, ali "Remove" gumb još uvijek je vidljiv, a automatski bi se trebao promijeniti u "Add to Cart" gumb.
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "Reset App State"	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Slika 11.1.15. Testiranje gumba „Resetiranje stanja aplikacije“ unutar menu-a

Izvor: Vlastita izrada

11.1.1 Rezultati ručnog testiranja

Ukupan broj testnih slučajeva za ručno testiranje je četrdeset, a od toga ih je uspješno prošlo dvadeset dva. Ostalih osamnaest testnih slučajeva nije prošlo, a većinom se odnose na korisnika „problem_user“. U nastavku je prikazan graf koji prikazuje rezultate ručnog testiranja.



Graf 11.1.1. Rezultati ručnog testiranja „Swag Labs“

Izvor: Vlastita izrada

Većina testova koji nisu prošli odnosi se na korisnika „problem_user“, a to je trinaest, dok kod „standard_usera“ nisu prošla samo četiri testa. Za korisnika „standard_user“ je prošlo petnaest testova dok je za „problem_usera“ prošlo svega sedam testova.

Vrijeme koje je bilo potrebno za izradu svih testnih slučajeva iznosi tri radna dana, odnosno dvadeset četiri sata. Vrijeme potrebno za izvršavanje svih testova iznosi jedan radni dan, odnosno osam sati.

11.2. Automatizirano testiranje web aplikacije „Swag Labs“ u alatu „Playwright“

Automatizirane skripte su podijeljene u dva dijela, jedan dio se odnosi na korisnika „standard_user“, a drugi na korisnika „problem_user“. U svakoj skripti se nalazi po dvadeset testova što je ukupno četrdeset testova, a sadrže sve elemente kao i testovi za ručno testiranje, samo su zapisani na drugačiji način. Svaki testni dokument sadrži testove za jedan modul te svaki im svoj POM (page object model) file u kojem se nalaze svi lokatori potrebni za pojedini test. U svaki testni dokument je potrebno instancirati odgovarajući POM dokument kako bi se mogli koristiti potrebni lokatori za testove. U skoro svakom testnom dokumentu prije ostalih testova nalazi se „before each hook“, a u prijevodu je to preduvjet za izvršavanje svakog slijedećeg testa kao što je na primjer prijava u aplikaciju. Prvo će se prikazati svi testovi za svaki modul, a kasnije će se prikazati rezultati testova.

Prvi testni dokument sadrži testove za prijavu u aplikaciju s korisnicima „standard_user“, „problem_user“, „locked_out_user“ i korisnikom koji nije prihvaćen od strane aplikacije. Testovi su prikazani na slici 11.2.2., a na slici 11.2.1. je prikazan POM dokument sa svim lokatorima potrebnim za izvršenje testova.

U POM dokumentu se definira klasa kao što je na primjer „class LogIn“ unutar koje se navode lokatori, koji se koriste za pronalaženje HTML elemenata stranice s data test atributima. Za prijavu su potrebni lokatori za polje ime, prezime, gumb za prijavu, lokator za očekivanu poruku greške u određenom slučaju. Na kraju dokumenta nalazi se naredba „ goto“ što omogućuje navigaciju na stranicu kao što je „https://www.saucedemo.com/“. Nakon što je klasa „LogIn“ popunjena sa svim potrebnim lokatorima kreće se na izradu testova počevši od pozivanja klase u testni dokument. Testovi se nalaze na slikama 11.2.2. i 11.2.3. te se na samom početku može se vidjeti naredba za umetanje klase. Također, klasu je potrebno instancirati na početku svakog testa. Osim što je potrebno unijeti lokatore, potrebno ih je i povezati s akcijama kao što je „ .click (); „ ili „fill in ();“ kako bi bilo moguće kliknuti ili ispuniti određeni element stranice. I na kraju, kako bi se moglo znati je li test uistinu prošao, potrebno je uvesti i očekivana stanja (eng. assertions) kao na primjer „expect(page.locator('#shopping_cart_container a')).toBeVisible();“ što znači da je nakon izvršenih koraka potrebno provjeriti je li određeni element

vidljiv ili ne te se može uvesti i očekivano stanje kao slijedeće „await expect(page).toHaveURL(/.*inventory.html/);“ gdje se očekuje da je nakon prijave u aplikaciju korisnik upravo na toj stranici. Testovi za modul „log in“ i POM dokument prikazani su na slijedećim slikama.

```
import { Locator, Page } from "@playwright/test";
export class LogIn {

    readonly page: Page;
    readonly korisnickoIme: Locator;
    readonly lozinka: Locator;
    readonly logInGumb: Locator;
    readonly porukaGreske: Locator;

    constructor(page: Page) {
     this.page = page;
     this.korisnickoIme = page.locator('[data-test="username"]');
     this.lozinka = page.locator('[data-test="password"]');
     this.logInGumb = page.locator('[data-test="login-button"]');
     this.porukaGreske = page.locator('[data-test="error"]');

    }

    async goto() {
        await this.page.goto("https://www.saucedemo.com/");
    }
}
```

Slika 11.2.1. POM dokument za klasu „LogIn“

Izvor: Vlastita izrada u alatu „Playwright“

```
import { test, expect } from '@playwright/test';
import { LogIn } from '../pages/login.page';

test.describe('testiranje log in stranice Swag Labs', () => {

  test('testiranje log in stranice sa standard_userom', async ({ page }) => {
    const login = new LogIn(page);
    await login.goto();
    await login.korisnickoIme.fill("standard_user");
    await login.lozinka.fill("secret_sauce");
    await login.logInGumb.click();
    await expect(page.locator('#shopping_cart_container a')).toBeVisible();
    await expect(page).toHaveURL(/.*inventory.html/)
  });

  test('testiranje log in stranice sa problem_userom', async ({ page }) => {
    const login = new LogIn(page);
    await login.goto();
    await login.korisnickoIme.fill("problem_user");
    await login.lozinka.fill("secret_sauce");
    await login.logInGumb.click();
    await expect(page.locator('#shopping_cart_container a')).toBeVisible();
    await expect(page).toHaveURL(/.*inventory.html/);
  });

  test('testiranje log in stranice sa locked_out_userom', async ({ page }) => {
    const login = new LogIn(page);
    await login.goto();
    await login.korisnickoIme.fill("locked_out_user");
    await login.lozinka.fill("secret_sauce");
    await login.logInGumb.click();
    await expect(page.locator('data-test="error"')).toBeVisible();
  });
```

Slika 11.2.2. Testovi za prijavu u aplikaciju „Swag Labs“

Izvor: Vlastita izrada u alatu „Playwright“

```
  test('testiranje log in stranice sa podacima koji nisu prihvaćeni kao korisničko ime i lozinka', async ({ page }) => {
    const login = new LogIn(page);
    await login.goto();
    await login.korisnickoIme.fill("Ana123!");
    await login.lozinka.fill("Ana123!");
    await login.logInGumb.click();
    await expect(page.locator('#shopping_cart_container a')).not.toBeVisible();
    await expect(page).not.toHaveURL(/.*inventory.html/);
  });
});
```

Slika 11.2.3. Test za prijavu u aplikaciju „Swag Labs“

Izvor: Vlastita izrada u alatu „Playwright“

Sljedeća slika prikazuje POM dokument za klasu „odjava“ (eng. Log Out). Nju čine lokatori za menu, za gumb „odjava“ i za gumb „prijava“ (eng. Log In).

```
import { Locator, Page } from "@playwright/test";
export class Logout {

    readonly page: Page;
    readonly menuGumb: Locator;
    readonly logOutGumb: Locator;
    readonly logInGumb: Locator;

    constructor(page: Page) {
     this.page = page;
     this.menuGumb = page.getByRole('button', { name: 'Open Menu' });
     this.logOutGumb = page.getByRole('link', { name: 'Logout' });
     this.logInGumb = page.locator('[data-test="login-button"]');

    }

    async goto() {
        await this.page.goto("https://www.saucedemo.com/");
   }
}
```

Slika 11.2.4. POM dokument za klasu „odjava“

Izvor: Vlastita izrada u alatu „Playwright“

Svi navedeni lokatori su potrebni za izvođenje testa prikazanog na slici 11.2.5. Dakle, na početku je potrebno ponovno uvesti klasu „prijava“ (eng. Log In) zbog toga što se ona koristi za izvršavanje preduvjeta (eng. before each hook) prijave u aplikaciju sa korisnikom „standard_user“. Nakon toga se uvodi klasa „odjava“ (eng. Log Out). Kako bi se znalo da je test sigurno prošao, očekivano stanje je da se nakon odjave korisnik nalazi na „https://www.saucedemo.com/“i da je gumb za prijavu vidljiv.

```
import { test, expect } from '@playwright/test';
import { LogIn } from '../pages/login.page';
import { Logout } from '../pages/logout.page';


test.describe('testiranje log out stranice s korisnikom standard_user', () => {

    test.beforeEach(async ({page}) => {
        const login= new LogIn(page);
        await login.goto();
        await login.korisnickoIme.fill("standard_user");
        await login.lozinka.fill("secret_sauce");
        await login.logInGumb.click();
    });

    test('testiranje log out stranice sa standard_userom', async ({ page }) => {
        const logout = new Logout(page);
        await logout.menuGumb.click();
        await logout.logOutGumb.click();
        await expect(page).toHaveURL('https://www.saucedemo.com/');
        await expect(logout.logInGumb).toBeVisible();
    });
});
```

Slika 11.2.5. Testovi za odjavu iz aplikacije „Swag Labs“

Izvor: Vlastita izrada u alatu „Playwright“

Slike 11.2.6. i 11.2.7 prikazuju POM dokument za klasu „Košarica“, koji sadrže sve lokatore potrebne za uspješno dodavanje ili micanje proizvoda iz košarice kao i lokator za nastavak kupovine.

```
import { Locator, Page } from "@playwright/test";
export class Kosarica {

    readonly page: Page;
    readonly dodajUkosaricuGumbRuksak: Locator;
    readonly dodajUKosaricuGumbSvjetloZaBiciklu: Locator;
    readonly dodajUkosaricuGumbBoltMajica: Locator;
    readonly dodajUkosaricuGumbVunenaJaketa: Locator;
    readonly dodajUkosaricuGumbBodi: Locator;
    readonly dodajUkosaricuGumbCrvenaMajica: Locator;

    readonly ikonaKosariceGumb: Locator;

    readonly ukloniGumbKosaricaRuksak: Locator;
    readonly ukloniGumbKosaricaSvjetloZaBiciklu: Locator;
    readonly ukloniGumbKosaricaBoltMajica: Locator;
    readonly ukloniGumbKosaricaVunenaJaketa: Locator;
    readonly ukloniGumbKosaricaCrvenaMajica: Locator;
    readonly ukloniGumbKosaricaBodi: Locator;

    readonly ukloniGumbWebTrgovinaaRuksak: Locator;
    readonly ukloniGumbWebTrgovinaSvjetloZaBiciklu: Locator;
    readonly ukloniGumbWebTrgovinaBoltMajica: Locator;
    readonly ukloniGumbWebTrgovinaVunenaJaketa: Locator;
    readonly ukloniGumbWebTrgovinaCrvenaMajica: Locator;
    readonly ukloniGumbWebTrgovinaBodi: Locator;

    readonly nastavakKupovineGumb: Locator;
```

Slika 11.2.6. Prvi dio POM dokumenta za klasu „Košarica“

Izvor: Vlastita izrada u alatu „Playwright“

```
    constructor(page: Page) {
        this.page = page;
        this.dodajUkosaricuGumbRuksak = page.locator('[data-test="add-to-cart-sauce-labs-backpack"]');
        this.dodajUKosaricuGumbSvjetloZaBiciklu = page.locator('[data-test="add-to-cart-sauce-labs-bike-light"]');
        this.dodajUkosaricuGumbBoltMajica = page.locator('[data-test="add-to-cart-sauce-labs-bolt-t-shirt"]');
        this.dodajUkosaricuGumbVunenaJaketa = page.locator('[data-test="add-to-cart-sauce-labs-fleece-jacket"]');
        this.dodajUkosaricuGumbBodi = page.locator('[data-test="add-to-cart-sauce-labs-onesie"]');
        this.dodajUkosaricuGumbCrvenaMajica = page.locator('[data-test="add-to-cart-test\\.allthethings\\(\\)-t-shirt-\\(red\\)"]');

        this.ikonaKosariceGumb = page.locator('#shopping_cart_container a');

        this.ukloniGumbKosaricaRuksak = page.locator('[data-test="remove-sauce-labs-backpack"]');
        this.ukloniGumbKosaricaSvjetloZaBiciklu = page.locator('[data-test="remove-sauce-labs-bike-light"]');
        this.ukloniGumbKosaricaBoltMajica = page.locator('[data-test="remove-sauce-labs-bolt-t-shirt"]');
        this.ukloniGumbKosaricaVunenaJaketa = page.locator('[data-test="remove-sauce-labs-fleece-jacket"]');
        this.ukloniGumbKosaricaCrvenaMajica = page.locator('[data-test="remove-test\\.allthethings\\(\\)-t-shirt-\\(red\\)"]');
        this.ukloniGumbKosaricaBodi = page.locator('[data-test="remove-sauce-labs-onesie"]');

        this.ukloniGumbWebTrgovinaaRuksak = page.locator('[data-test="remove-sauce-labs-backpack"]');
        this.ukloniGumbWebTrgovinaSvjetloZaBiciklu = page.locator('[data-test="remove-sauce-labs-bike-light"]');
        this.ukloniGumbWebTrgovinaBoltMajica = page.locator('[data-test="remove-sauce-labs-bolt-t-shirt"]');
        this.ukloniGumbWebTrgovinaVunenaJaketa = page.locator('[data-test="remove-sauce-labs-fleece-jacket"]');
        this.ukloniGumbWebTrgovinaCrvenaMajica = page.locator('[data-test="remove-test\\.allthethings\\(\\)-t-shirt-\\(red\\)"]');
        this.ukloniGumbWebTrgovinaBodi = page.locator('[data-test="remove-sauce-labs-onesie"]');

        this.nastavakKupovineGumb = page.locator('[data-test="continue-shopping"]');

    }

    async goto() {
        await this.page.goto("https://www.saucedemo.com/");
    }
}
```

Slika 11.2.7. Drugi dio POM dokumenta za klasu „Košarica“

Izvor: Vlastita izrada u alatu „Playwright“

Kao što se može vidjeti na slici 11.2.7. za testiranje košarice potrebni su lokatori kao što je gumb „dodaj“ za svaki proizvod posebno, gumb „makni“ unutar košarice ili na stranici online trgovine također za svaki proizvod posebno, gumb za nastavak kupovine i naravno lokator za ikonu košarice. Na slici 11.2.8. prikazan je preduvjet za izvedbu testova za košaricu.

```
import { test, expect } from '@playwright/test';
import { Kosarica } from '../pages/kosarica.page';
import { LogIn } from '../pages/login.page';

test.describe('testiranje košarice s korisnikom standard_user', () => {

    test.beforeEach(async ({page}) => {
        const login= new LogIn(page);
        await login.goto();
        await login.korisnickoIme.fill("standard_user");
        await login.lozinka.fill("secret_sauce");
        await login.logInGumb.click();
    });
```

Slika 11.2.8. Preduvjet za izvođenje testova za klasu „Košarica“

Izvor: Vlastita izrada u alatu „Playwright“

Na početku dokumenta uvode se klase „Log In“ i „Košarica“ te se kao preduvjet navodi prijava u aplikaciju. Prvim testom, koji je prikazan na slici 11.2.9. se provjerava mogućnost uklanjanja svih proizvoda klikom na gumb „ukloni“ unutar košarice. Osim toga u navedenom testu se ujedno provjerava i mogućnost dodavanja proizvoda u košaricu, jer ako proizvodi nisu dodani u košaricu onda se ne mogu niti ukloniti. Kao očekivana stanja navedeno

je to da ako su proizvodi dodani da bi gumb „ukloni“ unutar košarice trebao biti vidljiv, a nakon što se proizvodi uklone gumb „ukloni“ unutar košarice više ne bi trebao biti vidljiv.

```
test('Provjera mogućnosti micanja svih proizvoda iz košarice klikom na gumb Remove (ukloni) unutar košarice', async ({ page }) => {
    const kosarica= new Kosarica(page);
    await kosarica.dodajUkosaricuGumbRuksak.click();
    await kosarica.dodajUKosaricuGumbSvjetloZaBiciklu.click();
    await kosarica.dodajUkosaricuGumbBoltMajica.click();
    await kosarica.dodajUkosaricuGumbVunenaJaketa.click();
    await kosarica.dodajUkosaricuGumbBodi.click();
    await kosarica.dodajUkosaricuGumbCrvenaMajica.click();
    await kosarica.ikonaKosariceGumb.click();
    await expect(kosarica.ukloniGumbKosaricaRuksak).toBeVisible();
    await expect(kosarica.ukloniGumbKosaricaSvjetloZaBiciklu).toBeVisible();
    await expect(kosarica.ukloniGumbKosaricaBoltMajica).toBeVisible();
    await expect(kosarica.ukloniGumbKosaricaVunenaJaketa).toBeVisible();
    await expect(kosarica.ukloniGumbKosaricaBodi).toBeVisible();
    await expect(kosarica.ukloniGumbKosaricaCrvenaMajica).toBeVisible();
    await kosarica.ukloniGumbKosaricaRuksak.click();
    await kosarica.ukloniGumbKosaricaSvjetloZaBiciklu.click();
    await kosarica.ukloniGumbKosaricaBoltMajica.click();
    await kosarica.ukloniGumbKosaricaVunenaJaketa.click();
    await kosarica.ukloniGumbKosaricaBodi.click();
    await kosarica.ukloniGumbKosaricaCrvenaMajica.click();
    await expect(kosarica.ukloniGumbKosaricaRuksak).not.toBeVisible();
    await expect(kosarica.ukloniGumbKosaricaSvjetloZaBiciklu).not.toBeVisible();
    await expect(kosarica.ukloniGumbKosaricaBoltMajica).not.toBeVisible();
    await expect(kosarica.ukloniGumbKosaricaVunenaJaketa).not.toBeVisible();
    await expect(kosarica.ukloniGumbKosaricaBodi).not.toBeVisible();
    await expect(kosarica.ukloniGumbKosaricaCrvenaMajica).not.toBeVisible();
});
```

Slika 11.2.9. Drugi dio dokumenta s testovima za klasu „Košarica“

Izvor: Vlastita izrada u alatu „Playwright“

Slika 11.2.10. prikazuje test za provjeru uklanjanja proizvoda iz košarica putem gumba „ukloni“ na naslovnoj stranici online trgovine. Očekivano stanje je da nakon što se proizvodi uklone iz košarice da gumb „ukloni“ više nije vidljiv na stranici online trgovine.

```
    test('Provjera mogućnosti micanja svih proizvoda iz košarice klikom na gumb Remove (ukloni) na web trgovini', async ({page}) => {
        const kosarica= new Kosarica(page);
        await kosarica.dodajUkosaricuGumbRuksak.click();
        await kosarica.dodajUKosaricuGumbSvjetloZaBiciklu.click();
        await kosarica.dodajUkosaricuGumbBoltMajica.click();
        await kosarica.dodajUkosaricuGumbVunenaJaketa.click();
        await kosarica.dodajUkosaricuGumbBodi.click();
        await kosarica.dodajUkosaricuGumbCrvenaMajica.click();
        await kosarica.ikonaKosariceGumb.click();
        await expect(kosarica.ukloniGumbKosaricaRuksak).toBeVisible();
        await expect(kosarica.ukloniGumbKosaricaSvjetloZaBiciklu).toBeVisible();
        await expect(kosarica.ukloniGumbKosaricaBoltMajica).toBeVisible();
        await expect(kosarica.ukloniGumbKosaricaVunenaJaketa).toBeVisible();
        await expect(kosarica.ukloniGumbKosaricaBodi).toBeVisible();
        await expect(kosarica.ukloniGumbKosaricaCrvenaMajica).toBeVisible();
        await kosarica.nastavakKupovineGumb.click();
        await kosarica.ukloniGumbWebTrgovinaaRuksak.click();
        await kosarica.ukloniGumbWebTrgovinaSvjetloZaBiciklu.click();
        await kosarica.ukloniGumbWebTrgovinaBoltMajica.click();
        await kosarica.ukloniGumbWebTrgovinaVunenaJaketa.click();
        await kosarica.ukloniGumbWebTrgovinaCrvenaMajica.click();
        await kosarica.ukloniGumbWebTrgovinaBodi.click();
        await kosarica.ikonaKosariceGumb.click();
        await expect(kosarica.ukloniGumbKosaricaRuksak).not.toBeVisible();
        await expect(kosarica.ukloniGumbKosaricaSvjetloZaBiciklu).not.toBeVisible();
        await expect(kosarica.ukloniGumbKosaricaBoltMajica).not.toBeVisible();
        await expect(kosarica.ukloniGumbKosaricaVunenaJaketa).not.toBeVisible();
        await expect(kosarica.ukloniGumbKosaricaBodi).not.toBeVisible();
        await expect(kosarica.ukloniGumbKosaricaCrvenaMajica).not.toBeVisible();
    });
});
```

Slika 11.2.10. Treći dio dokumenta s testovima za klasu „Košarica“

Izvor: Vlastita izrada u alatu „Playwright“

Sljedeća slika prikazuje POM dokument za klasu „Narudžba“. Taj dokument sadrži lokatore potrebne za dovršetak narudžbe, a to su: ikona košarice, gumb za odjavu, polja u informacijskom obrascu, gumb za završetak narudžbe i gumb za poništavanje.

```
import { Locator, Page } from "@playwright/test";
export class Narudzba {

    readonly page: Page;
    readonly ikonaKoSariceGumb: Locator;
    readonly checkoutGumb: Locator;
    readonly poljeIme: Locator;
    readonly poljePrezime: Locator;
    readonly poljePostanskiBroj: Locator;
    readonly gumbNastavi: Locator;
    readonly gumbZavrsi: Locator;
    readonly ponistiGumb: Locator;

    constructor(page: Page) {
        this.page = page;
        this.ikonaKoSariceGumb = page.locator('#shopping_cart_container a');
        this.checkoutGumb = page.locator('[data-test="checkout"]');
        this.poljeIme = page.locator('[data-test="firstName"]');
        this.poljePrezime = page.locator('[data-test="lastName"]');
        this.poljePostanskiBroj = page.locator('[data-test="postalCode"]');
        this.gumbNastavi = page.locator('[data-test="continue"]');
        this.gumbZavrsi = page.locator('[data-test="finish"]');
        this.ponistiGumb = page.locator('[data-test="cancel"]');


    }

    async goto() {
        await this.page.goto("https://www.saucedemo.com/");
    }
}
```

Slika 11.2.11. POM dokument za klasu „Narudžba“

Izvor: Vlastita izrada u alatu „Playwright“

Za izvođenje testova za modul „narudžba“ potrebno je osim klase „Narudžba“ uvesti još i klasu „Log In“ i klasu „Košarica“. Dokument s testovima sadrži ukupno pet testova u kojima se testira nemogućnost dovršetka prazne narudžbe, popunjavanje informacijskog obrasca s podacima o kupcu koji nisu validni, popunjavanje informacijskog obrasca s podacima koji su validni, provjera točnosti ukupne sume narudžbe sa šest proizvoda te provjera hoće li podaci o kupcu ostati spremljeni nakon povratka s pregleda konačne narudžbe na informacijski obrazac. Kao očekivana stanja navodi se da je za nemogućnost dovršetka prazne narudžbe očekivano da neće biti vidljiva poruka o uspješno izvršenoj narudžbi, za testiranje nemogućnosti popunjavanja informacijskog obrasca navodi se da je očekivana poruka o pogrešci, za uspješno popunjavanje informacijskog obrasca i uspješan dovršetak narudžbe navodi se da je očekivano da se korisnik nalazi na (/.*checkout-complete.html/) te da je gumb „povratak na proizvode“ vidljiv. Za provjeru točnosti ukupne sume šest proizvoda kao očekivani iznos na računu navodi se „ Total: $140.34" te se za test o spremljenim korisničkim podacima navodi da je

nakon povratka na informacijski obrazac očekivano da su polja još uvijek popunjena s određenim vrijednostima. Ovi opisani testovi nalaze se na slikama 11.2.12., 11.2.13., 11.2.14., 11.2.15.

```
import { test, expect } from '@playwright/test';
import { LogIn } from '../pages/login.page';
import { Narudzba } from '../pages/narudzba.page';
import { Kosarica } from '../pages/kosarica.page';

test.describe('testiranje izrade narudžbe i računa s korisnikom standard_user', () => {

    test.beforeEach(async ({page}) => {
        const login = new LogIn(page);
        await login.goto();
        await login.korisnickoIme.fill("standard_user");
        await login.lozinka.fill("secret_sauce");
        await login.logInGumb.click();
    });

    test('testiranje nemogućnosti dovršetka narudžbe s nula proizvoda', async ({ page }) => {
        const narudzba= new Narudzba(page);
        await narudzba.ikonaKoSariceGumb.click();
        await narudzba.checkoutGumb.click();
        await narudzba.poljeIme.fill("Ana");
        await narudzba.poljePrezime.fill("Milin");
        await narudzba.poljePostanskiBroj.fill("21000");
        await narudzba.gumbNastavi.click();
        await narudzba.gumbZavrsi.click();
        await expect(page.locator('#checkout_complete_container')).not.toBeVisible();
    });

    test('testiranje informacijskog obrasca s podacima o kupcu koji nisu validni i dovršetak narudžbe', async ({ page }) => {
        const narudzba= new Narudzba(page);
        await narudzba.ikonaKoSariceGumb.click();
        await narudzba.checkoutGumb.click();
        await narudzba.poljeIme.fill("2");
        await narudzba.poljePrezime.fill("2");
        await narudzba.poljePostanskiBroj.fill("2");
        await narudzba.gumbNastavi.click();
        await expect(page.locator('.error-message-container')).toHaveText("Error: Customer information is not valid");
    });
```

Slika 11.2.12. Prvi dio dokumenta s testovima za klasu „Narudžba“

Izvor: Vlastita izrada u alatu „Playwright“

```
    test('ispunjavanje informacijskog obrasca validnim vrijednostima i dovršetak narudžbe', async ({ page }) => {
        const kosarica= new Kosarica(page);
        const narudzba= new Narudzba(page);
        await kosarica.dodajUkosaricuGumbRuksak.click();
        await kosarica.dodajUKosaricuGumbSvjetloZaBiciklu.click();
        await kosarica.dodajUkosaricuGumbBoltMajica.click();
        await kosarica.dodajUkosaricuGumbVunenaJaketa.click();
        await kosarica.dodajUkosaricuGumbBodi.click();
        await kosarica.dodajUkosaricuGumbCrvenaMajica.click();
        await kosarica.ikonaKosariceGumb.click();
        await narudzba.checkoutGumb.click();
        await narudzba.poljeIme.fill("Ana");
        await narudzba.poljePrezime.fill("Milin");
        await narudzba.poljePostanskiBroj.fill("21000");
        await narudzba.gumbNastavi.click();
        await narudzba.gumbZavrsi.click();
        await expect(page).toHaveURL(/.*checkout-complete.html/);
        await expect(page.locator('[data-test="back-to-products"]')).toBeVisible();
    });
```

Slika 11.2.13. Drugi dio dokumenta s testovima za klasu „Narudžba“

Izvor: Vlastita izrada u alatu „Playwright“

```
    test('provjera točnosti ukupne sume računa sa šest proizvoda', async ({ page }) => {
        const kosarica= new Kosarica(page);
        const narudzba= new Narudzba(page);
        await kosarica.dodajUkosaricuGumbRuksak.click();
        await kosarica.dodajUKosaricuGumbSvjetloZaBiciklu.click();
        await kosarica.dodajUkosaricuGumbBoltMajica.click();
        await kosarica.dodajUkosaricuGumbVunenaJaketa.click();
        await kosarica.dodajUkosaricuGumbBodi.click();
        await kosarica.dodajUkosaricuGumbCrvenaMajica.click();
        await kosarica.ikonaKosariceGumb.click();
        await narudzba.checkoutGumb.click();
        await narudzba.poljeIme.fill("Ana");
        await narudzba.poljePrezime.fill("Milin");
        await narudzba.poljePostanskiBroj.fill("21000");
        await narudzba.gumbNastavi.click();
        await expect(page.getByText('Total: $140.34')).toHaveText("Total: $140.34");
    });
```

Slika 11.2.14. Treći dio dokumenta s testovima za klasu „Narudžba“

Izvor: Vlastita izrada u alatu „Playwright“

```
    test('provjera hoće li podaci o kupcu ostati spremljeni pri povratku s pregleda konačne narudžbe na informacijski obrazac', async ({ page
        const kosarica= new Kosarica(page);
        const narudzba= new Narudzba(page);
        await kosarica.dodajUkosaricuGumbRuksak.click();
        await kosarica.dodajUKosaricuGumbSvjetloZaBiciklu.click();
        await kosarica.dodajUkosaricuGumbBoltMajica.click();
        await kosarica.dodajUkosaricuGumbVunenaJaketa.click();
        await kosarica.dodajUkosaricuGumbBodi.click();
        await kosarica.dodajUkosaricuGumbCrvenaMajica.click();
        await kosarica.ikonaKosariceGumb.click();
        await narudzba.checkoutGumb.click();
        await narudzba.poljeIme.fill("Ana");
        await narudzba.poljePrezime.fill("Milin");
        await narudzba.poljePostanskiBroj.fill("21000");
        await narudzba.gumbNastavi.click();
        await narudzba.ponistiGumb.click();
        await expect(page).toHaveURL(/.*checkout-step-one.html/);
        await expect(page.locator('.checkout_info')).toHaveText("Ana Milin 21000");
    });
});
```

Slika 11.2.15. Četvrti dio dokumenta s testovima za klasu „Narudžba“

Izvor: Vlastita izrada u alatu „Playwright“

Slika 11.2.16. prikazuje POM dokument za klasu „Web Shop“ odnosno online trgovina. On sadrži lokatore kao što su: menu gumb, ikona košarice, slika, naziv, opis, cijena proizvoda, naslov online trgovine, filter, dodaj u košaricu gumb kako bi se testovi koji su prikazani na slikama 11.2.17. i 11.2.18. mogli izvršiti. Osim toga potrebno je uvesti i klasu „Log In“ kako bi se izvršio preduvjet prijave u aplikaciju. Ukupno će se izvršiti pet testova koji ispituju redom: testiranje vidljivosti ikone košarica, testiranje vidljivosti ikone menu, testiranje vidljivosti naslova „Swag Labs“, testiranje vidljivosti filtera, testiranje vidljivosti cijene, opisa, naziva, slike proizvoda i gumba „Dodaj u košaricu“ (eng. add to cart).

```
import { Locator, Page } from "@playwright/test";
export class WebShop {

    readonly page: Page;
    readonly ikonaKosariceGumb: Locator;
    readonly menuGumb: Locator;
    readonly naslovStranice: Locator;
    readonly kartica: Locator;
    readonly slika: Locator;
    readonly naziv: Locator;
    readonly opis: Locator;
    readonly cijena: Locator;
    readonly dodajUkosaricuGumb: Locator;
    readonly filterGumb: Locator;

    constructor(page: Page) {
        this.page = page;
        this.ikonaKosariceGumb = page.locator('#shopping_cart_container a');
        this.menuGumb = page.getByRole('button', { name: 'Open Menu' });
        this.naslovStranice = page.getByText('Swag Labs');
        this.kartica = page.locator('.inventory_item').first();
        this.slika = page.locator('#item_4_img_link');
        this.naziv = page.locator('#item_4_title_link');
        this.opis = page.getByText('carry.allTheThings() with the sleek, streamlined Sly Pack that melds uncompromis');
        this.cijena = page.getByText('$29.99');
        this.dodajUkosaricuGumb = page.locator('[data-test="add-to-cart-sauce-labs-backpack"]');
        this.filterGumb = page.locator('[data-test="product_sort_container"]');

    }

    async goto() {
        await this.page.goto("https://www.saucedemo.com/");
    }
}
```

Slika 11.2.16. POM dokument za klasu „Web Shop“

Izvor: Vlastita izrada u alatu „Playwright“

```
import { test, expect } from '@playwright/test';
import { LogIn } from '../pages/login.page';
import { WebShop } from '../pages/webshop.page';


test.describe('testiranje web shopa s korisnikom standard_user', () => {
    test.beforeEach(async ({page}) => {
        const login = new LogIn(page);
        await login.goto();
        await login.korisnickoIme.fill("standard_user");
        await login.lozinka.fill("secret_sauce");
        await login.logInGumb.click();
    });

    test('testiranje vidljivosti ikone Košarica', async ({ page }) => {
        const webshop = new WebShop(page);
        await expect(webshop.ikonaKosariceGumb).toBeVisible();
    });

    test('testiranje vidljivosti ikone Menu', async ({ page }) => {
        const webshop = new WebShop(page);
        await expect(webshop.menuGumb).toBeVisible();
    });

    test('testiranje vidljivosti teksta Swag Labs', async ({ page }) => {
        const webshop = new WebShop(page);
        await expect(webshop.naslovStranice).toBeVisible();
    });
```

Slika 11.2.17. Prvi dio dokumenta s testovima za klasu „Web Shop“

Izvor: Vlastita izrada u alatu „Playwright“

```
    test('testiranje vidljivosti slike proizvoda, naziva proizvoda, opisa proizvoda, cijene i gumba "Add to cart" ', async ({ page }) => {
        const webshop = new WebShop(page);
        await expect(webshop.kartica).toBeVisible();
        await expect(webshop.slika).toBeVisible();
        await expect(webshop.cijena).toBeVisible();
        await expect(webshop.naziv).toBeVisible();
        await expect(webshop.opis).toBeVisible();
        await expect(webshop.dodajUkosaricuGumb).toBeVisible();
    });

    test('testiranje vidljivosti filtera ', async ({ page }) => {
        const webshop = new WebShop(page);
        await webshop.filterGumb.click();
        await expect(webshop.filterGumb).toBeVisible();
    });
});
```

Slika 11.2.18. Drugi dio dokumenta s testovima za klasu „Web Shop“

Izvor: Vlastita izrada u alatu „Playwright“

Posljednji modul koji će se testirati je „Menu“, a slika 11.2.19. pokazuje njegov POM dokument. Kako bi se testovi uspješno izvršili potrebni su lokatori kao što je gumb „Menu“, gumb „Svi proizvodi“, gumb „O nama“, gumb „Resetiraj“, gumb „Zatvori menu“. Osim navedenih lokatora potrebni su još neki koji će se uvesti uz pomoć klasa „Košarica“ i „Log In“.

```
import { Locator, Page } from "@playwright/test";
export class Menu {

    readonly page: Page;
    readonly menuGumb: Locator;
    readonly sviProizvodiGumb: Locator;
    readonly oNamaGumb: Locator;
    readonly resetirajStanjeAplikacijeGumb: Locator;
    readonly zatvoriMenuGumb: Locator;

    constructor(page: Page) {
     this.page = page;
     this.menuGumb = page.getByRole('button', { name: 'Open Menu' });
     this.sviProizvodiGumb = page.getByRole('link', { name: 'All Items' });
     this.oNamaGumb = page.getByRole('link', { name: 'About' });
     this.resetirajStanjeAplikacijeGumb = page.getByRole('link', { name: 'Reset App State' });
     this.zatvoriMenuGumb = page.getByRole('button', { name: 'Close Menu' });

    }

    async goto() {
        await this.page.goto("https://www.saucedemo.com/");
    }
}
```

Slika 10.2.19. POM dokument za klasu „Menu“

Izvor: Vlastita izrada u alatu „Playwright“

Dokument sa testovima sadrži ukupno tri testa koja ispituju gumb „Svi proizvodi“ (eng. All items), gumb „O nama“ (eng. About), te gumb „Resetiraj stanje aplikacije“ (eng. Reset App State). Navedeni testovi se nalaze na slikama 11.2.20. i 11.2.21.

```
import { test, expect } from '@playwright/test';
import { LogIn } from '../pages/login.page';
import { Menu } from '../pages/menu.page';
import { Kosarica } from '../pages/kosarica.page';

test.describe('testiranje menua stranice s korisnikom standard_user', () => {
    test.beforeEach(async ({page}) => {
        const login = new LogIn(page);
        await login.goto();
        await login.korisnickoIme.fill("standard_user");
        await login.lozinka.fill("secret_sauce");
        await login.logInGumb.click();
    });

    test('testiranje gumba All Items unutar menua', async ({ page }) => {
        const menu = new Menu(page);
        await menu.menuGumb.click();
        await menu.sviProizvodiGumb.click();
        await expect(page.locator('.inventory_list')).toBeVisible()
    });

    test('testiranje gumba About unutar menua', async ({ page }) => {
        const menu = new Menu(page);
        await menu.menuGumb.click();
        await menu.oNamaGumb.click()
        await expect(page).not.toHaveURL("https://saucelabs.com/error/404");
    });
```

Slika 11.2.20. Prvi dio dokumenta s testovima za klasu „Menu“

Izvor: Vlastita izrada u alatu „Playwright“

```
    test('testiranje gumba Reset App State unutar menua', async ({ page }) => {
        const kosarica = new Kosarica(page);
        const menu = new Menu(page);
        await kosarica.dodajUkosaricuGumbRuksak.click();
        await kosarica.dodajUKosaricuGumbSvjetloZaBiciklu.click();
        await menu.menuGumb.click();
        await menu.resetirajStanjeAplikacijeGumb.click();
        await menu.zatvoriMenuGumb.click();
        await expect(kosarica.dodajUkosaricuGumbRuksak).toBeVisible();
        await expect(kosarica.dodajUKosaricuGumbSvjetloZaBiciklu).toBeVisible();
    });
});
```

Slika 11.2.21. Drugi dio dokumenta s testovima za klasu „Menu“

Izvor: Vlastita izrada u alatu „Playwright“

Svi prikazani testovi su za korisnika „standard_user“, ali za korisnika „problem_user“ su jednaki, osim što je preduvjet drugačiji, a on izgleda kao na slijedećoj slici.

```
test.beforeEach(async ({page}) => {
    const login = new LogIn(page);
    await login.goto();
    await login.korisnickoIme.fill("problem_user");
    await login.lozinka.fill("secret_sauce");
    await login.logInGumb.click();
});
```

Slika 11.2.22. Prikaz preduvjeta za testove korisnika „problem_user“

Izvor: Vlastita izrada u alatu „Playwright“

Nakon što se svi prikazani testovi izvrše, slijedi analiza rezultata automatiziranog testiranja.

11.2.1. Rezultati automatiziranog testiranja

Izvršit će se ukupno četrdeset testova, podijeljenih u dvije testne skripte, za svakog od korisnika „problem_user“ i „standard_user“. Testovi će se pokrenuti u terminalu naredbom prikazanoj na sljedećoj slici.

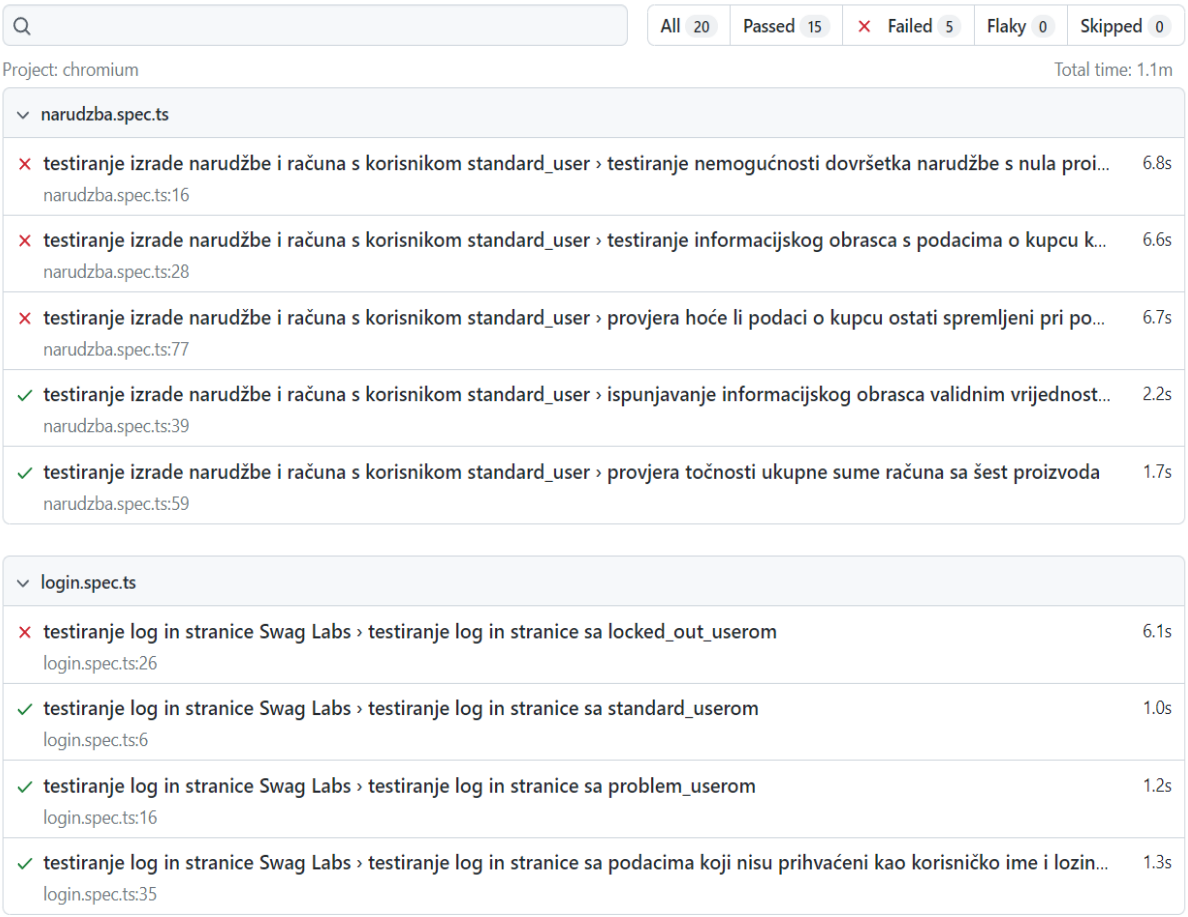
```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Users\anami\Desktop\Swag-Labs1 (standard_user)> npm run playwright
```

Slika 11.2.1.1. Prikaz naredbe za pokretanje automatiziranih testova

Izvor: Vlastita izrada u alatu „Playwright“

Nakon što su testovi pokrenuti potrebno je pričekati izvješće testiranja. Potrebno je naglasiti da će se aplikacija testirati isključivo u „Chrome“ pregledniku. U nastavku slijedi ispis testova za oba korisnika. Na slikama 11.2.1.2. i 11.2.1.3. se nalazi ispis rezultata testova za „standard_usera“, a na slikama 11.2.1.4. i 11.2.1.5 se nalazi ispis rezultata testova za korisnika „problem_user“.



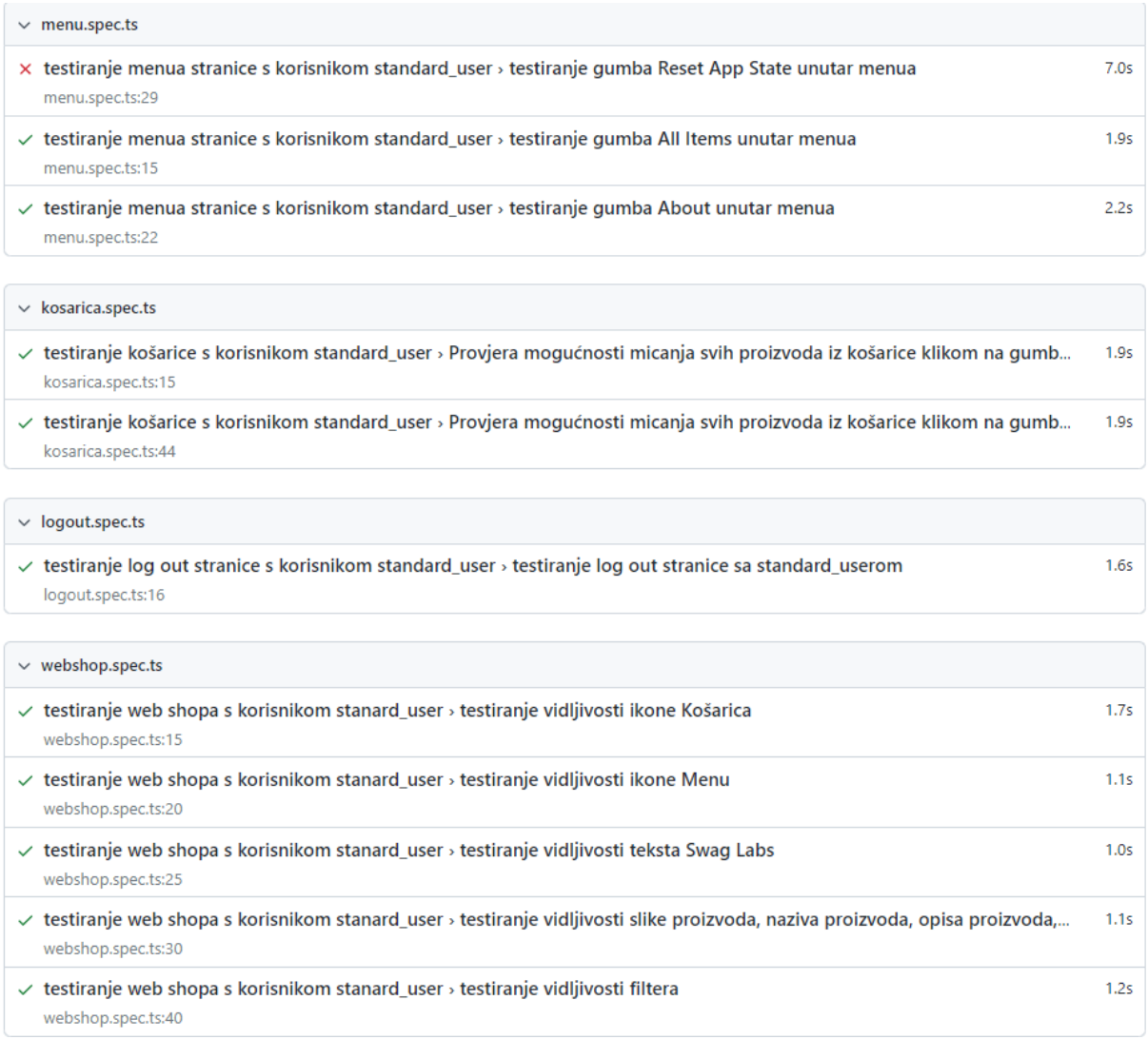
Slika 11.2.1.2. Prvi dio izvješća o rezultatima testova za korisnika „standard_user“

Izvor: Automatski generirano izvješće

Kao što se može vidjeti na prikazu izvješća, za korisnika „standard_user“ je izvršeno ukupno dvadeset testova, od čega je uspješno prošlo pet, a očekivani rezultat nije zadovoljilo pet testova. Ukupno vrijeme koje je bilo potrebno za izvođenje testova iznosi 1.1 minutu.

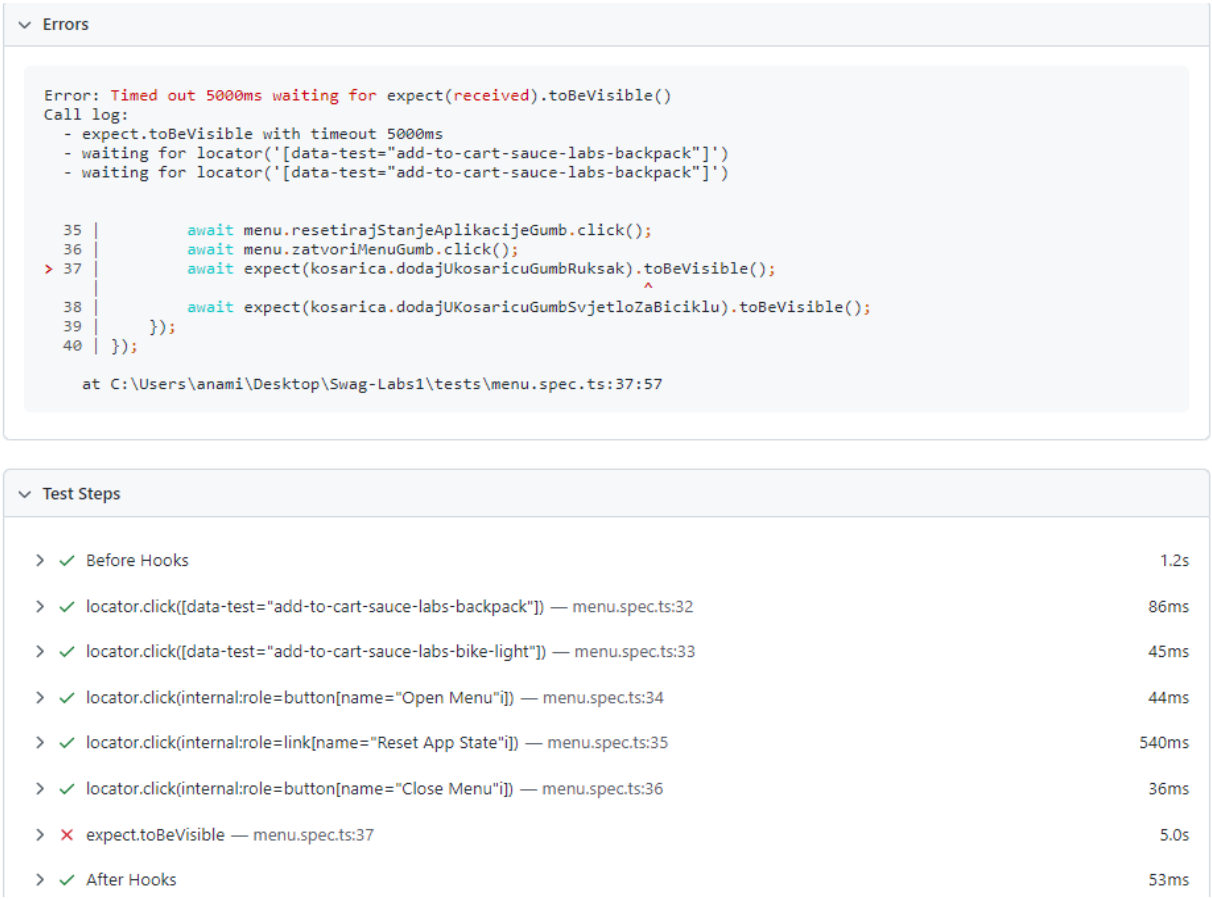
Moduli koji imaju potpunu prolaznost testova su: „Online trgovina“, „Odjava“, „Košarica“, a ostali imaju djelomičnu prolaznost. Tako u modulu „Menu“ nije prošao test za gumb „Resetiraj stanje aplikacije“, za modul „Prijava“ nije prošao test za „locked_out_user“ te za modul „Narudžba“ nisu prošla tri testa. To su testovi gdje se ispituje je li moguće dovršiti narudžbu s nula proizvoda, prepoznaje li informacijski obrazac podatke o kupcu koji nisu validni, te hoće li podaci o kupcu ostati spremljeni nakon povratka s prethodne stranice.

Prednost ovakvih izvješća je to što se za svaki test koji nije prošao odmah ispisuje obrazloženje problema kao što je prikazano na slici 11.2.1.4.



Slika 11.2.1.3. Drugi dio izvješća o rezultatima testova za korisnika „standard_user“

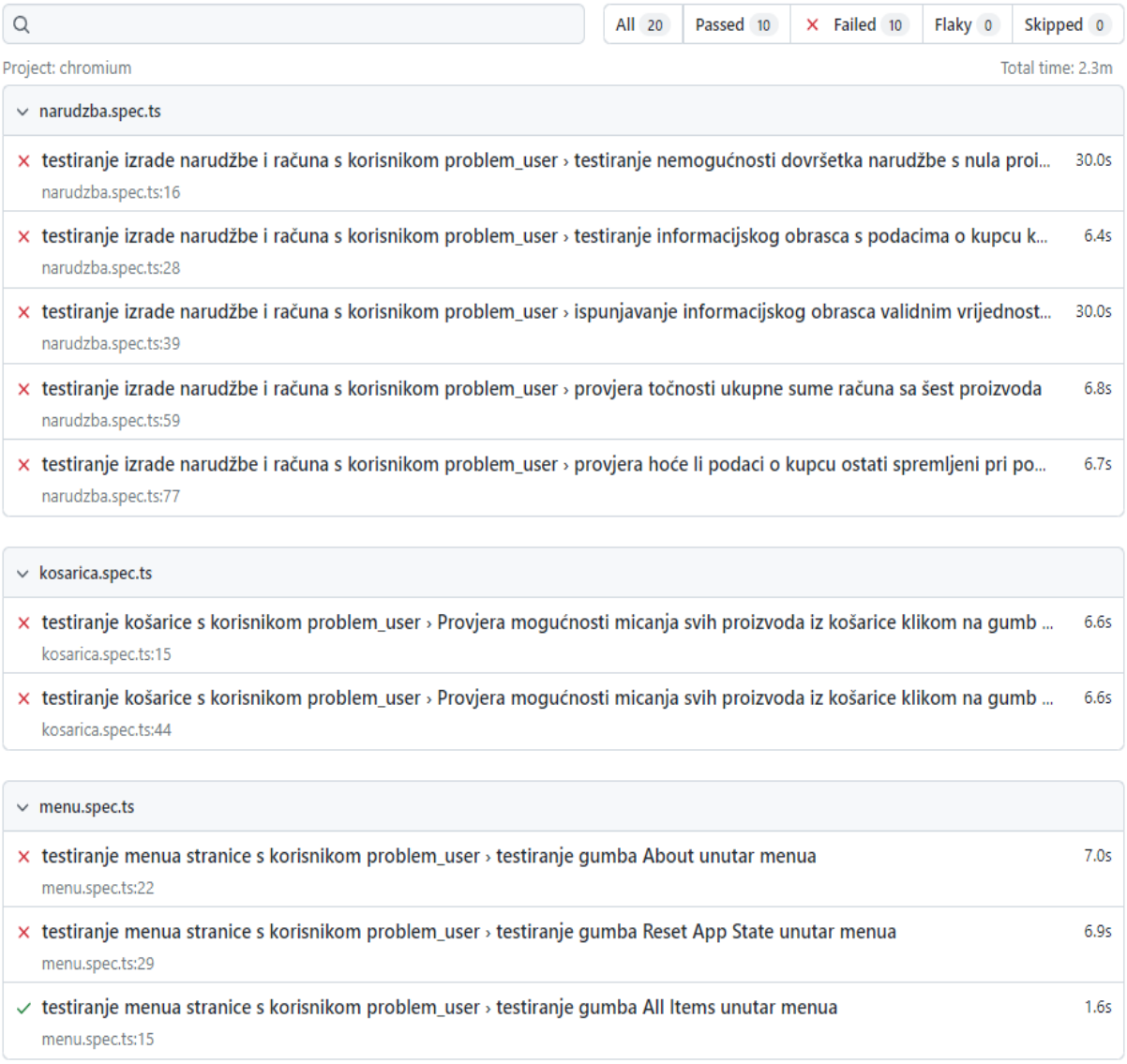
Izvor: Automatski generirano izvješće



Slika 11.2.1.4. Izvješće o pogrešci za testiranje gumba „Resetiraj stanje aplikacije“

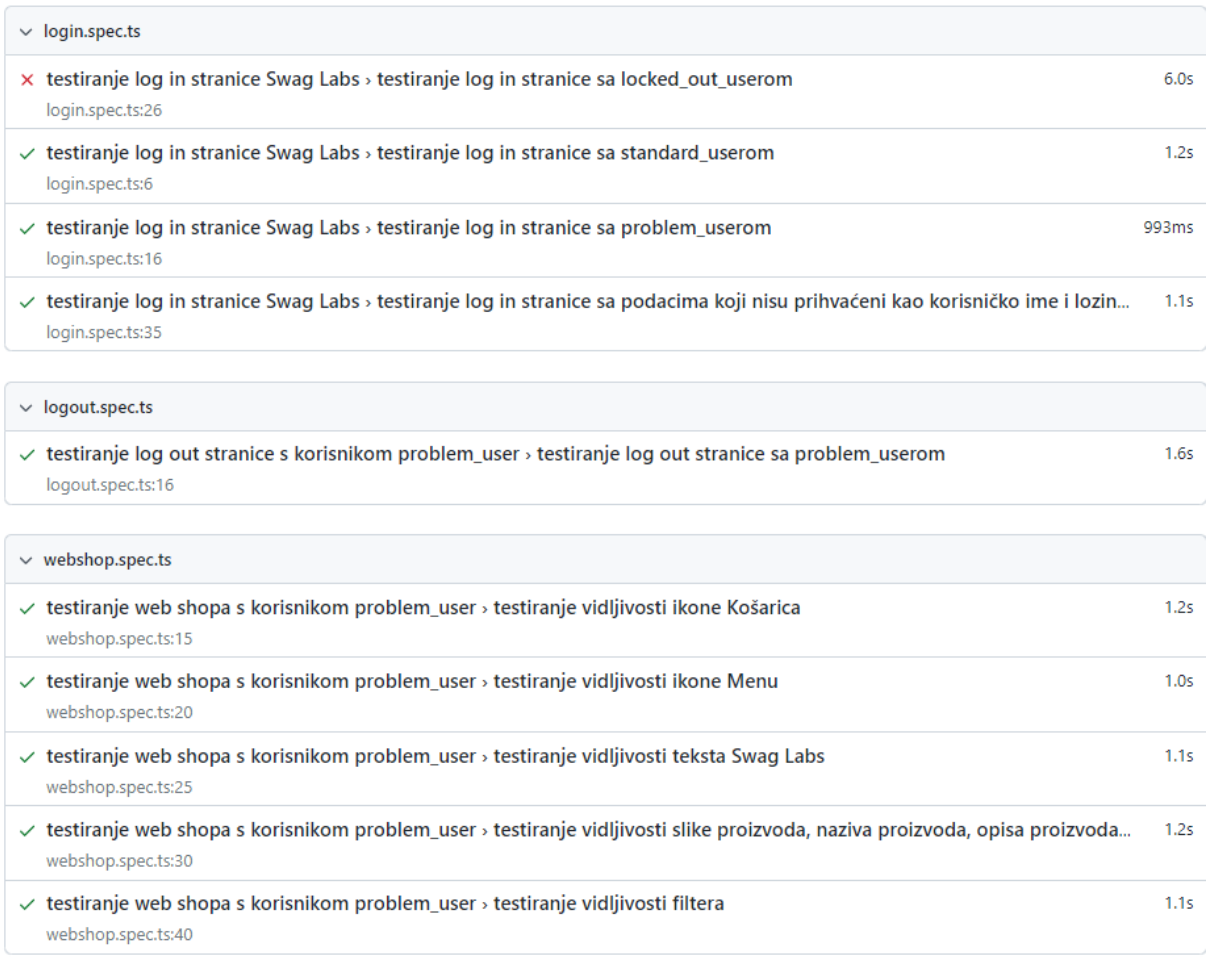
Izvor: Automatski generirano izvješće

U nastavku slijedi prikaz izvješća rezultata testova za korisnika „problem_user“. Izvršeno je ukupno dvadeset testova, a nije ih prošlo deset, što je i očekivani rezultat od korisnika „problem_user“. Za izvršenje testova bilo je potrebno 2.3 minute. Na slikama 11.2.1.5. i 11.2.1.6. je prikazano izvješće za rezultate testova. Modul koji imaju potpunu prolaznost testova su „Odjava“ i „Online trgovina“, a ostali moduli imaju djelomičnu prolaznost. Testovi koji nisu prošli su: testiranje log in stranice sa locked_out_userom, testiranje gumba „Resetiraj stanje aplikacije“, testiranje mogućnosti micanja proizvoda iz košarice klikom na gumb „Ukloni“ unutar košarice, testiranje mogućnosti micanja proizvoda iz košarice klikom na gumb „Ukloni“ na stranici online trgovine, testiranje hoće li podaci o kupcu ostati spremljeni pri povratku s prethodne stranice, testiranje točnosti ukupne sume proizvoda na računu, testiranje ispunjavanja informacijskom obrasca s podacima koji su validni i onima koji nisu te na kraju testiranje nemogućnosti dovršetka narudžbe s nula proizvoda.



Slika 11.2.1.5. Prvi dio izvješća o rezultatima testova za korisnika „problem_user“

Izvor: Automatski generirano izvješće



Slika 11.2.1.6. Drugi dio izvješća o rezultatima testova za korisnika „problem_user“

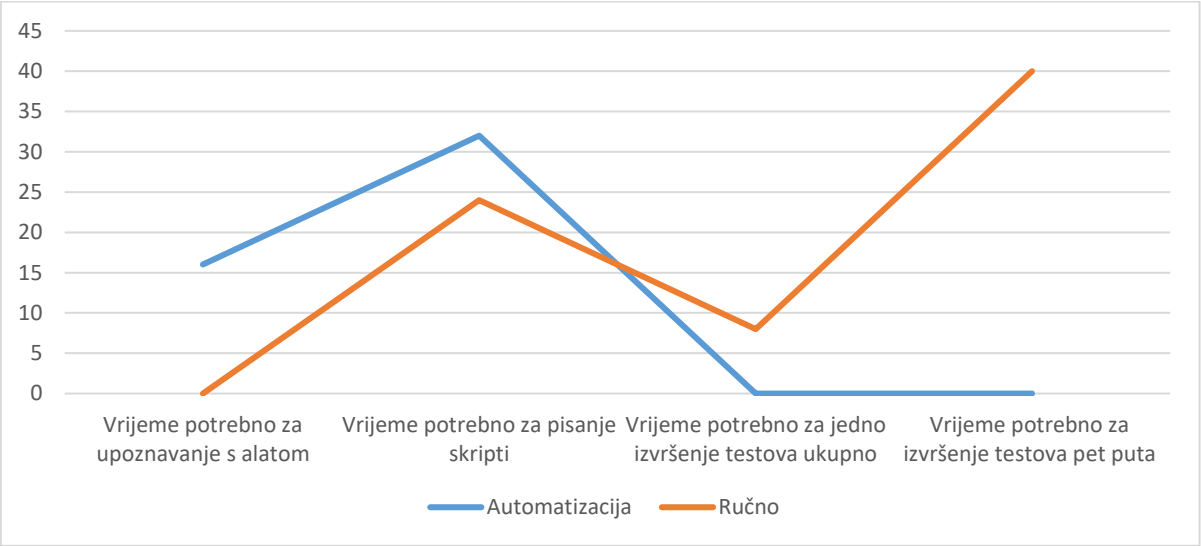
Izvor: Automatski generirano izvješće

11.3. Rasprava

Cilj ovog istraživanja je bilo analizirati učinkovitost automatiziranog i ručnog testiranja na primjeru web aplikacije „Swag Labs“. Za svaki pristup testiranju provedeno je po četrdeset testova, a kao glavne metrike istraživanja uzet će se vrijeme potrebno za izradu testova i vrijeme potrebno za izvršavanje testova. Na temelju navedenih metrika dat će se odgovori na istraživačka pitanja.

U prvom istraživačkom pitanju se navodi da je automatizirano testiranje brže i efikasnije od ručnog testiranja jer se testovi izvršavaju sami te se omogućavaju brže povratne informacije nego što je to slučaj kod ručnog testiranja. Prema rezultatima istraživanja, vrijeme koje je bilo potrebno za upoznavanje s alatom „Playwright“ i njegovu instalaciju iznosi dva radna dana, što je ukupno šesnaest sati, a vrijeme potrebno za pisanje automatiziranih skripti iznosi četiri radna dana, odnosno trideset dva sata. Nakon što su testne skripte bile u potpunosti spremne, bilo ih je potrebno samo pokrenuti, a za „standard_usera“ vrijeme izvršenja iznosi 1.1. minutu, dok za „problem_usera“ vrijeme izvršenja

iznosi 2.3. minute. S druge strane, vrijeme potrebno za izradu testnih slučajeva za ručno testiranje iznosi tri radna dana, odnosno dvadeset i četiri sata, dok vrijeme potrebno za izvršenje testova iznosi osam radnih sati. Navedeni rezultati prikazani su na slijedećem grafu.



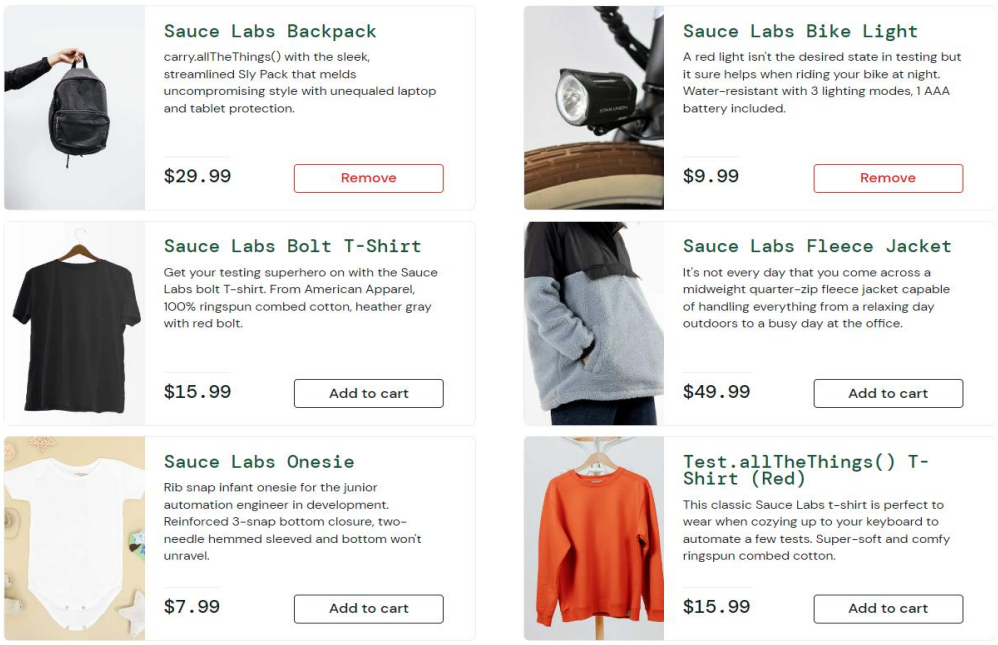
Graf 11.3.1 Prikaz vremenskog troška ručnog i automatiziranog testiranja

Izvor: Vlastita izrada

Na prikazanom grafu može se uočiti da na samom početku istraživanja automatizirano testiranje oduzima više vremena od ručnog testiranja zbog potrebe za upoznavanjem alata i instaliranja istoga, dok za ručno testiranje to nije potrebno. Vrijeme potrebno za izradu automatiziranih testnih skripti također zahtijeva više vremena zbog korištenja programskog jezika „Typescript“, dok je za izradu ručnih testova potrebno nešto manje vremena. Nakon što su testni slučajevi izrađeni, za izvršenje automatiziranih testnih slučajeva je potrebno svega 3.4 minute dok je za izvršenje ručnih testova potreban jedan radni dan. Što je testove potrebno više puta izvršiti to se trošak vremena za ručno testiranje povećava dok je kod automatiziranog testiranja trošak vremena postojan. Dakle, na temelju navedenih rezultata prvo istraživačko pitanje da je automatizirano testiranje brže i efikasnije od ručnog se prihvaća, jer je dugoročni trošak vremena za automatizirano testiranje vrlo malen, dok se za ručno testiranje samo povećava. Isto tako, potrebno je naglasiti da se automatizirano testiranje više isplati za dugoročne projekte nego za kratke projekte gdje je poželjnije manualno testiranje.

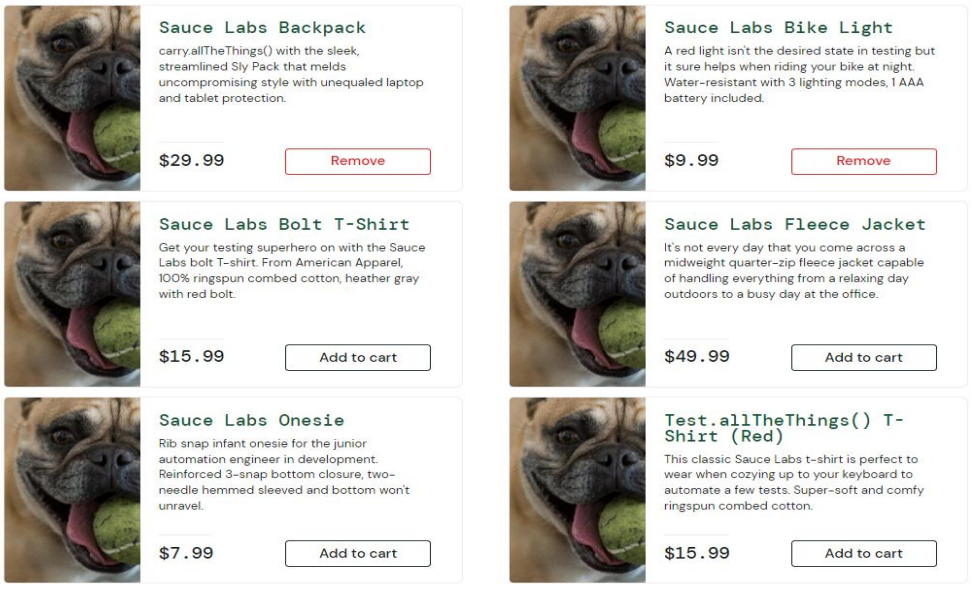
Drugo istraživačko pitanje navodi da je ručno testiranje korisnije nego automatizirano prilikom testiranja korisničkog iskustva jer automatizirano testiranje ne uključuje ljudsko razmatranje. Također, prije svakog automatiziranog testiranja je potrebno prvo obaviti ručno testiranje kako bi se izbjegla mogućnost lažnih rezultata. Dakle, prilikom automatizacije je moguće ispitati je li određen element vidljiv, ali nema uvida u to je li element u skladu sa korisničkim zahtjevima. Prethodno navedeno se može vidjeti na primjeru testiranja modula „Online trgovina“. Dakle, test koji je proveden za ručno

glasi: „Testiranje vidljivosti slike proizvoda, cijene proizvoda, naziva i opisa proizvoda te gumba "Dodaj u košaricu“ na jednoj kartici proizvoda.“. Rezultat testa za korisnika „standard_user“ je bio prolaz, dok za korisnika „problem_user“ test nije prošao, jer iako su vidljive sve navedene stavke, slika proizvoda se ne poklapa s korisničkim zahtjevima, odnosno prikazane su slike psa umjesto slike proizvoda. Razlika je prikazana na slijedećim slikama.



Slika 11.3.1. Prikaz naslovne stranice online trgovine „Swag Labs“ za korisnika „standard_user“

Izvor: https://www.saucedemo.com/



Slika 11.3.2. Prikaz naslovne stranice online trgovine „Swag Labs“ za korisnika „problem_user“

Izvor: https://www.saucedemo.com/

Iako je razlika između izgleda prethodne dvije slike evidentna, automatiziranim testom za korisnika „problem_user“ dobio se pozitivan rezultat testa. To je zato što je za očekivano stanje postavljeno da stavke slika proizvoda, cijena proizvoda, naziv i opis proizvoda te gumb "Dodaj u košaricu“ trebaju biti vidljivi kao što je prikazano na slijedećoj slici.

```
test('testiranje vidljivosti slike proizvoda, naziva proizvoda, opisa proizvoda, cijene i gumba "Add to cart" ', async ({ page }) => {
    const webshop = new WebShop(page);
    await expect(webshop.kartica).toBeVisible();
    await expect(webshop.slika).toBeVisible();
    await expect(webshop.cijena).toBeVisible();
    await expect(webshop.naziv).toBeVisible();
    await expect(webshop.opis).toBeVisible();
    await expect(webshop.dodajUkosaricuGumb).toBeVisible();
});
```

Slika 11.3.3. Prikaz automatiziranog testa za korisnika „problem_user“

Izvor: Vlastita izrada u alatu „Playwright“

Navedene stavke i jesu vidljive na stranici online trgovine, ali automatizacija nije mogla prepoznati da slike proizvoda nisu ispravne, zbog čega je test prošao, a loše korisničko iskustvo nije otkriveno.

Zbog navedenih rezultata testiranja, drugo istraživačko pitanje da je ručno testiranje korisnije nego automatizirano prilikom testiranja korisničkog iskustva se prihvaća.

Treće istraživačko pitanje glasi da je kombinacija ručnog i automatiziranog testiranja najefektivniji način osiguravanja kvalitete programskog proizvoda. Na temelju dobivenih rezultata koji su prezentirani na grafu 11.3.1. može se zaključiti da je na početku projekta efektivnije prvo istraživački testirati aplikaciju te uvidjeti koje to korisničke zahtjeve aplikacija mora zadovoljiti kako bi korisničko iskustvo bilo intuitivno, a zatim odrediti testne slučajeve koji će se automatizirati. Time će se na duži period uštedjeti puno više vremena nego da se testiranje svaki put provodi ručno. Također, svaku aplikaciju je potrebno prvo testirati ručno, upravo zato da se ne bi dogodila pogreška kao što je prikazano u rezultatima za istraživačko pitanje broj dva te na slikama 11.3.1. i 11.3.2. Dakle, da se aplikacija nije testirala prvo ručnom metodom, pogreška na slikama proizvoda se ne bi otkrila te bi korisničko iskustvo bilo vrlo loše. Zaključak za treće istraživačko pitanje je taj da je kombinacija ručnog i manualnog testiranja uistinu najefektivnija metoda testiranja te da se istraživačko pitanje prihvaća.

12. OGRANIČENJA ISTRAŽIVANJA

Kada je riječ o ograničenjima provedenog istraživanja potrebno je naglasiti resursno ograničenje, odnosno to da je testiranje provedeno na jednoj aplikaciji koja ima određeni broj funkcionalnosti. To znači da su testni podaci ograničeni te je samim time i broj testnih scenarija ograničen. Kako bi se dobili općenitiji rezultati potrebno je provesti testiranje na više aplikacija s velikim brojem različitih funkcionalnosti. Osim toga, alat Playwright ima i određena ograničenja, kao što je nemogućnost testiranja korisničkog iskustva. Budući da je istraživanje ograničeno značajkama samog alata, nije bilo moguće usporediti manualno i automatizirano testiranje na primjeru testiranja korisničkog iskustva. Dakle, provedeno je ukupno četrdeset testova za ručno testiranje te četrdeset testova za automatizirano testiranje na temelju čijih se rezultata donio odgovor na sva tri istraživačka pitanja. Takvim uzorkom se može ograničiti opseg rezultata.

Bez obzira na navedena ograničenja, riječ je o značajnom istraživanju, koje može doprinijeti poboljšanju provedbe ispitivanja kvalitete softvera u brojnim IT tvrtkama.

13. ZAKLJUČAK

Evolucijom pristupa i tehnika testiranju softvera postalo je jasno da je testiranje softvera neophodan korak u procesu razvoja visoko kvalitetnog programskog proizvoda koji zadovoljava korisničke zahtjeve. Iako je testiranje softvera dugotrajan i skup proces jedini je način osiguravanja kvalitetnog proizvoda čime tvrtke postižu konkurentnu prednost na tržištu. Testiranjem se poboljšava korisničko iskustvo, pronalaze se propusti i pogreške te se smanjuje rizik od pojave novih nedostataka. Kako bi se testiranje uspješno provelo potrebno se držati određenih metodologija kao što su metodologija testiranja crne, bijele i sive kutije, određenih razina testiranja te vrsta testiranja kao što je ručno i automatizirano testiranje. Ručno testiranje softvera se koristi za kraće projekte te omogućuje detaljno istraživačko testiranje softvera poput obuhvaćanja rubnih testnih slučajeva. Vrlo je korisno u prvim fazama razvoja softvera kada se često uvode nove funkcionalnosti i poboljšavaju one već postojeće. Osim navedenog, ručnim testiranjem se osigurava intuitivno korisničko sučelje koje omogućuje kvalitetnu interakciju s korisnikom što u konačnici rezultira pozitivnim korisničkim iskustvom. U ovom diplomskom radu je na temelju rezultata istraživanja zaključeno da je ručno testiranje učinkovitije od automatiziranog testiranja prilikom testiranja korisničkog iskustva. To je tako zato što automatizirano testiranje može testirati poziciju, vidljivost ili klikabilnost nekog elementa, ali ne može odrediti je li taj element zadovoljava standarde očekivanog korisničkog iskustva. S druge strane, automatizirano testiranje se najviše koristi za regresijsko testiranje te za dugotrajne projekte gdje nema previše promjena u funkcionalnostima softvera. Ono pruža brže povratne informacije o pogreškama te omogućuje prevenciju mogućih nedostataka u softveru. U početku je trošak automatiziranog testiranja nešto veći zbog implementacije i upoznavanja s alatom za automatizirano testiranje, ali kasnije je trošak testiranja vrlo nizak jer se testovi izvršavaju sami i to velikom brzinom. Navedeno je potvrđeno i rezultatima testiranja u ovom diplomskom radu gdje se pokazalo da kako testiranje sve više odmiče da se vremenski trošak ručnog testiranja povećava, a za automatizirano testiranje on ostaje postojan. Time je potvrđeno istraživačko pitanje da je automatizirano testiranje efikasnije od ručnog testiranja. Iako se u posljednje vrijeme sve više koristi automatizirano testiranje, nije moguće izraditi kvalitetne testne skripte ako prije toga nije provedeno ručno testiranje softvera. Automatizirano testiranje može davati lažno pozitivne rezultate, a ako softver prije toga nije testiran ručnim putem, pogreška se neće otkriti sve dok se ne dostavi klijentu koji će shvatiti da proizvod ne zadovoljava korisničke zahtjeve. Kako bi se osigurao visoko kvalitetan proizvod nužno je kombinirati ručno i automatizirano testiranje, što je potvrđeno i u ovom istraživanju prilikom testiranja aplikacije „Swag Labs“. Može se zaključiti da je testiranje softvera vrlo važan dio procesa razvoja softvera, koji treba biti uključen u proces razvoja od samog početka, a nužno je testiranje kombinacijom ručnog i automatiziranog testiranja.

LITERATURA

KNJIGE

[1] Everett, G. D., & Mcleod, R. (2007): *Software Testing Testing Across the Entire Software Development Life Cycle*. New Jersey: Wiley, ISBN 978-0-471-79371-7

[2] Fewster, M. , Graham, D., (1999) : *Software Test Automation : Effective use of test execution tool.* New York: ACM Press, ISBN 0-201-33140-3

[3] Kaner, C., Falk, J., Nguyen, H., Q., (1994) : Testing Computer Software, New Jersey: Wiley, ISBN 978-0471358466

[4] Knott, D., (2015) : Hands – On Mobile App Testing. Crawfordsville: Addison-Wesley Professional, ISBN: 978-0-13-419171-3

[5] Magner, R. (2005) : Softversko Inženjerstvo. Zagreb: Element, ISBN 978-953-197-600-8

[6] Mili, A., Tchier F., (2015) : *Software Testing: Concepts and Operation*. New Jersey: Wiley, ISBN 978-1-118-66287-8

[7] Myers, G. J., Sandler C. , Badgett T., (2011) : *The Art of Software Testing, 3rd Edition*. New Jersey: Wiley, ISBN 978-1-118-03196-4

[8] Naik, K., Tripathy P. , 2008 : *Software Testing And Quality Assurance, Theory and Practice.* New Jersey: Wiley, ISBN 978-1-118-21163-2

[9] Patton, R., (2009) : *Software Testing, Second Edition.* Carmel, Indiana: Sams, ISBN 978-0-672-32798-8

[10] Spillner, A., Linz T. , Schaefer H., (2014) : *Software Testing Foundations, 4th Edition.* Santa Barbara: Rocky Nook, ISBN 978-1-937538-42-2

[11] Toth, M., Čendo Metzinger T. : Me*todologija istraživačkog rada za stručne studije*. Velika Gorica: Veleučilište Velika Gorica, ISBN 978-953-7716-90-5

[12] Zelenika, R., (2000): *Metodologija i tehnologija izrade znanstvenog i stručnog djela*. Rijeka: Ekonomski fakultet u Rijeci, ISBN 953-614-810-2

ČLANCI

[13] Adee, S., (2013), : *Bad bugs: The worst disasters caused by software fails.* NewScientist, URL: https://www.newscientist.com/gallery/software-bugs/ (pristupljeno u svibnju 2023.)

[14] Albarka Umar, M., Zhanfang, C., (2019) : *A Study of Automated Software Testing: Automation Tools and Frameworks*. International Journal of Computer Science Engineering (IJCSE), DOI 10.5281/zenodo.3924795

[15] Ammann, P., Offutt, J., (2008) : *Introduction to software testing*. Cambridge University Press.

[16] Bach, J., (2003): Exploratory Testing Explained v.1.3 4/16/03

[17] Bevanda, V., Sustavi, G. S., Potpori, Z. U., (2009) : *Sustavi Znanja u Potpori Upravljanju Kvalitetom Softverskog Proizvoda. Pula, Informatologia*, *42*, 284–292.

[18] Deshpande, M., V., (2018): Interdependency and Segregation associates between software Development and software testing life cycles

[19] Despa, M. L. (2014) : Comparative study on software development methodologies Database Systems Journal, vol. V, no. 3/2014

[20] Itkonen, J., Mäntylä, M. v., & Lassenius, C. (2009). How do testers do it? An exploratory study on manual testing practices. *3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM, DOI 10.1109/ESEM.2009.5314240

[21] Kannan, V., Jhajharia, S., Verma, S., (2014). Agile vs waterfall: A Comparative Analysis. International Journal of Science, Engineering and Technology Research (IJSETR), 3(10), 2680-2686.

[22] Nidhra, S., Dondeti, J. (2012) : *Black box and White box testing techniques // International Journal of Embedded Systems and Applications (IJESA) Vol.2, 2012*

[23] Pan, J., (1999) : *Software Testing*

[24] Ragunath, PK, Velmourougan, S. (2010) : Evolving A New Model (SDLC Model-2010) For Software Development Life Cycle
(SDLC).// IJCSNS International Journal of Computer Science and Network Security, Vol.10, Br.1, 2010.

[25] Riaz, S. S. A. : *International Journal of Engineering Science and Technology Vol. 1(3*), *2009, 119-128 :Studyng the Feasibility and Importnance of Software Testing: An Analysis*

[26] Sabev P., Grigorova K., 2015: *Manual to Automated Testing: An Effort-Based Approach for Determining the Priority of Software Test Automation*

[27] Sawant, A., A., Bari, H., P., Chawan, P., M., (2012) : Software Testing Techniques and Strategies, ISSN 2248-9622

[28] Sharma, R., M., (2014) : *Quantitative Analysis of Automation and Manual Testin,* ISSN 2277-3754

[29] Smys, S., Shakya S. , (2020): *Reliable Automated Software Testing Through Hybrid Optimization Algorith,* DOI https://doi.org/10.36548/jucct.2020.3.002

[30] Syed, R., J., Syed, T., U., S., Zia, U., J.,, Yasin., S, Fazlullah, K., (2016): An Innovative Approach to

[31] Investigate Various Software Testing Techniques and Strategies, ISSN 2394-4099

[32] Wong, E., W., Horgan, J., R.,London, S., Agrawal, H., (1997) : A Study of Effective Regression Testing in Practice

INTERNET IZVORI

[33] Apica Product Team, (2022): *Manual vs. Automated Software Testing*
URL: https://www.apica.io/difference-between-automated-manual-testing/#:~:text=In%20manual%20testing%20(as%20the,of%20any%20successful%20software%20project (pristupljeno uožujku 2023)

[34] Atatus, (2021): A Beginner's Guide to DOM (Document Object Model)
URL: https://www.atatus.com/blog/a-beginners-guide-to-dom/#What-is-DOM? (pristupljeno u svibnju 2023)

[35] Hernandez A., D., (2019): *History of software testing*. URL: https://davidmoremad.medium.com/history-of-software-testing-cfa461c4ae0a (pristupljeno u veljači 2023)

[36] Bezsmlna, V. , (2019) : *7 Benefits of Implementing Automated Testing*. URL: https://smartbear.com/blog/7-benefits-of-implementing-automated-testing/ (pristupljeno u svibnju 2023)

[37] GH, Community Contributor, (2023): Playwright Automation Framework: Tutorial URL: https://www.browserstack.com/guide/playwright-tutorial(pristupljeno u svibnju 2023)

[38] Ghost Inspector, (2023): CSS Selector Cheat Sheet: Strategies for Automated Browser Testing URL:https://ghostinspector.com/blog/css-selector-strategies-automated-browser-testing/ (pristupljeno u svibnju2023)

[39] Hamilton, T., (2023) : *Automation Testing Vs. Manual Testing: What's the Difference?* URL:https://www.guru99.com/difference-automated-vs-manual-testing.html (pristupljeno u veljači 2023.)

[40] Hamilton, T., (2023) : *How to Write Test Cases: Sample Template with Examples* URL:https://www.guru99.com/test-case.html (pristupljeno u veljači 2023.)

[41] Hamilton, T., (2023) : *Unit Testing Tutorial – What is, Types & Test Example* URL:https://www.guru99.com/unit-testing-guide.html (pristupljeno u ožujku 2023)
[42] Hamilton, T., (2023) : White Box Testing – What is, Techniques, Example & Types URL:https://www.guru99.com/white-box-testing.html (pristupljeno u veljači 2023)

[43] Hamilton, T., (2023): STLC (Software Testing Life Cycle) Phases, Entry, Exit Criteria URL: https://www.guru99.com/software-testing-life-cycle.html (pristupljeno u ožujku 2023)

[44] Hamilton, T., (2023): Difference Between Manual and Automation Testing URL:https://www.guru99.com/difference-automated-vs-manual-testing.html (pristupljeno u ožujku 2023)

[45] Hamilton, T., (2023): What is Regression Testing? Test Cases Example URL:https://www.guru99.com/regression-testing.html (pristupljeno u travnju 2023)

[46] Hamilton, T., (2023): What is System Testing? Types with Example URL:https://www.guru99.com/system-testing.html (pristupljeno u travnju2023)

[47] Hamilton, T., (2023): Integration Testing: What is, Types with Example URL:https://www.guru99.com/integration-testing.html (pristupljeno u ožujku 2023)

[48] HDONWEB, : *Nativna aplikacija ili web aplikacija?* URL:https://www.hdonweb.com/mobiteli/nativna-aplikacija-mobilna-web-stranica (pristupljeno u svibnju 2023.)

[49] Hernandez, A. D., (2019) : History of software testing URL: https://davidmoremad.medium.com/history-of-software-testing-cfa461c4ae0a (pristupljeno u ožujku 2023)

[50] IBM: *What is Software Testing?* URL: https://www.ibm.com/topics/software-testing (pristupljeno u veljači 2023.)
[51] Infinum, (2023): CSS, URL: https://infinum.com/handbook/qa/automation/locators/css (pristupljeno u svibnju 2023)

[52] Intland Software (2021) : Why is QA Testing So Important? URL: https://intland.com/blog/agile/test-management/why-qa-testing-is-so-important/ (pristupljeno u veljači 2023)

[53] Interviewvbit, (2023): Top Automation Testing Tools To Know URL: https://www.interviewbit.com/blog/manual-testing-tools/ (pristupljeno u travnju 2023)

[54] Johnson, P., (2021): *Gray Box Testing Guide*
URL: https://www.mend.io/resources/blog/gray-box-testing/ (pristupljeno u veljači 2023.)
[55] Katalon, (2023) : *Automation Testing 101: What, Why and How*
URL: https://katalon.com/resources-center/blog/what-is-automation-testing (pristupljeno u travnju 2023.)
[56] Katalon, (2023): What Is Automation Testing? Ultimate Guide & Best Practices
URL: https://katalon.com/resources-center/blog/what-is-automation-testing (pristupljeno u travnju 2023)

[57] Lowenthal, J., (2020) : *How to Get Started Testing: Best Test Cases to Automate*
URL: https://smartbear.com/blog/how-to-get-started-testing-best-test-cases-to-auto/ (pristupljeno u svibnju 2023.)
[58] Meerts, J., Graham, D., (2019): *The History of Software Testing*
URL: https://www.testingreferences.com/testinghistory.php (pristupljeno u veljači 2023.)
[59] Microsoft, (2023) : *Playwright* , URL: https://playwright.dev/ (pristupljeno u lipnju 2023)
[60] Microsoft, (2023): Page Object Models , URL: https://playwright.dev/docs/pom (pristupljeno u travnju 2023)
[61] Murray, A. (2021): Gray Box Testing Guide ,URL: https://www.mend.io/resources/blog/gray-box-testing/ (pristupljeno u travnju 2023)

[62] Oppy G., Dowe D., (2021) : The Turing Test URL: https://plato.stanford.edu/entries/turing-test/ (pristupljeno u ožujku 2023)

[63] PFLB : *The Role of Manual Software Testing in Software Development*.,
URL: https://pflb.us/blog/manual-testing-in-software-development/ (pristupljeno u veljači 2023.)
[64] Rajkumar, (2021) : *What is Software Testing Life Cycle (STLC) & STLC Phases*
URL: https://www.softwaretestingmaterial.com/stlc-software-testing-life-cycle/ (pristupljeno u veljači 2023.)

[65] Sanyal, R., (2021): URL: https://www.lambdatest.com/blog/web-vs-hybrid-vs-native-apps/ (pristupljeno u svibnju 2023)

[66] Seganfredo, V., (2019): An Analysis of Selectors on Test Automation,
URL: https://medium.com/docler-engineering/an-analysis-of-selectors-in-test-automation-f3cc40c34426 (pristupljeno u lipnju 2023)

[67] Singh, R., (2021) : *Manual Testing.* , URL: https://www.toolsqa.com/software-testing/manual-testing/ (pristupljeno u veljači 2023.)

[68] Software Testing Help (2023) : *SDLC (Software Development Life Cycle) Phases, Process, Models* URL: https://www.softwaretestinghelp.com/software-development-life-cycle-sdlc/#5_Deployment (pristupljeno u travnju 2023)

[69] Software Testing Help, (2023): What Is Automation Testing (Ultimate Guide To Start Test Automation) URL: https://www.softwaretestinghelp.com/automation-testing-tutorial-1/ (pristupljeno u travnju 2023)

[70] Software Testing Help, (2023): What Is Cross Browser Testing And How To Perform It: A Complete Guide URL: https://www.softwaretestinghelp.com/how-is-cross-browser-testing-performed/(pristupljeno u svibnju 2023)

[71] Tanna, H., (2017) : *Top 10 Benefits of Test Automation*
URL: https://dzone.com/articles/top-10-benefits-of-test-automation (pristupljeno u svibnju 2023.)
[72] Testim, (2020): *Test Automation Benefits: 12 Reasons to Automate in 2020*
URL: https://www.testim.io/blog/test-automation-benefits/ (pristupljeno u veljači 2023.)
[73] Testim, (2020): *Test Automation vs Manual Testing: Picking the Right Balance*.
URL: https://www.testim.io/blog/test-automation-vs-manual-testing/ (pristupljeno u svibnju 2023.)
[74] Tutorialspoint : *Software Testing – Types of Testing*
URL: https://www.tutorialspoint.com/software_testing/software_testing_types.htm (pristupljeno u svibnju 2023.)
[75] Tutorialspoint : *Software Testing Methods.*
URL: https://www.tutorialspoint.com/software_testing/software_testing_methods.htm (pristupljeno u svibnju 2023.)

[76] Tutorialspoint : *Software Development Life Cycle*
URL:https://www.tutorialspoint.com/sdlc/sdlc_overview.htm# (pristupljeno u svibnju 2023)

[77] Tutorialspoint : Software Testing Methods
URL: https://www.tutorialspoint.com/software_testing/software_testing_methods.htm (pristupljeno u svibnju 2023)

[78] Ullah, S. , (2019) : *A brief history of software testing*. , URL: https://salsa.digital/insights/a-brief-history-of-software-testing (pristupljeno u svibnju 2023.)
[79] *Vodič za razvoj web i mobilnih aplikacija* , URL: https://zale.hr/vodic-za-razvoj-web-i-mobilnih-aplikacija/ (pristupljeno u svibnju 2023.)
[80] QA Team., *QA Process: How the QA Team Tests Your Project.*
URL: https://kruschecompany.com/quality-assurance-in-projects/ (pristupljeno u svibnju2023.)

POPIS SLIKA

POPIS GRAFIKONA

SAŽETAK

U ovom diplomskom radu analizira se učinkovitost i efikasnost ručnog i automatiziranog testiranja na primjeru web aplikacije „Swag Labs“ korištenjem alata za automatizirano testiranje „Playwright“. Daje se prikaz testnih slučajeva za ručno testiranje i testnih skripti za automatizirano testiranje. Temeljem rezultata istraživanja jasno se ističe razlika između navedenih vrsta testiranja te se daje zaključak o najboljim praksama za osiguravanje kvalitete proizvoda. Time se ističe važnost testiranja softvera kao primarnog koraka za osiguravanje kvalitete programskog proizvoda. Osim toga pojašnjavaju se različite metodologije i tehnike testiranja, kao što je testiranje crne, bijele i sive kutije te unakrsno testiranje, jedinično testiranje i regresijsko testiranje. Kako bi se važnost testiranja softvera u potpunosti razumjela prikazana je evolucija testiranja softverskih proizvoda kroz povijest. Time se ukazuje na razliku u testiranju softvera nekad i danas te se zaključuje da osiguravanje kvalitete programskog proizvoda ovisi o uključivanju testiranja od najranijih faza razvoja softvera.

Ključne riječi: ručno testiranje, automatizirano testiranje, kvaliteta softvera, Playwright, Swag Labs

SUMMARY

This thesis analyzes the effectiveness and efficiency of manual and automated testing on the example of the web application "Swag Labs" using the automated testing tool "Playwright". Test cases for manual testing and test scripts for automated testing are presented. Based on the results of the research, the difference between the mentioned types of testing is clearly highlighted, and a conclusion is given on the best practices for ensuring product quality. This highlights the importance of software testing as a primary step for ensuring the quality of a software product. In addition, various testing methodologies and techniques are explained, such as black, white, and gray box testing, as well as cross-testing, unit testing, and regression testing. In order to fully understand the importance of software testing, the evolution of software product testing throughout history is presented. This indicates the difference in software testing then and now, and it is concluded that ensuring the quality of the software product depends on the inclusion of testing from the earliest stages of software development.

Keywords: manual testing, automated testing, software quality, Playwright, Swag Labs

PRILOG

Modul	Naslov	Opis	Preduvjet	#	Koraci	Testni podaci	Očekivani rezultat	Prioritet	Rezultat (standard_user)	Rezultat (problem_user)	Bilješke
Prijava (Log in)	Prijava sa imenom "standard_user" i lozinkom "secret_sauce"	Provjerite je li se moguće prijaviti s imenom "standard_user" i lozinkom "secret_sauce"	Idite na: https://www.saucedemo.com/	1.	Unesite ime "standard_user" u tekstni okvir "username"	Ime: standard_user	Korisnik je prijavljen - https://www.saucedemo.com/inventory.html	**Visok**	**Prolaz**		
				2.	Unesite lozinku "secret_sauce" u tekstni okvir "password"	Lozinka: secret_sauce					
				3.	Kliknite na "Log in" gumb						
Prijava (Log in)	Prijava sa imenom "locked_out_user" i lozinkom "secret_sauce"	Provjerite je li se moguće prijaviti s imenom "locked_out_user" i lozinkom "secret_sauce"	Idite na: https://www.saucedemo.com/	1.	Unesite ime "locked_out_user" u tekstni okvir "username"	Ime: locked_out_user	Korisnik je prijavljen - https://www.saucedemo.com/inventory.html	**Visok**			**Test nije prošao za korisnika locked_out_user**
				2.	Unesite lozinku "secret_sauce" u tekstni okvir "password"	Lozinka: secret_sauce					
				3.	Kliknite na "Log in" gumb						
Prijava (Log in)	Prijava sa imenom "problem_user" i lozinkom "secret_sauce"	Provjerite je li se moguće prijaviti s imenom "problem_user" i lozinkom "secret_sauce"	Idite na: https://www.saucedemo.com/	1.	Unesite ime "problem_user" u tekstni okvir "username"	Ime: problem_user	Korisnik je prijavljen - https://www.saucedemo.com/inventory.html	**Visok**		**Prolaz**	
				2.	Unesite lozinku "secret_sauce" u tekstni okvir "password"	Lozinka: secret_sauce					
				3.	Kliknite na "Log in" gumb						

Prijava (Log in)	Prijava s podacima koji nisu prihvaćeni kao korisničko ime i lozinka za https://www.saucedemo.com/	Provjerite da se nije moguće prijaviti s korisničkim imenima i lozinkama koji nisu prihvaćeni na https://www.saucedemo.com/"	Idite na: https://www.saucedemo.com/	1.	U tekstualni okvir "username" upišite "Ana123!"	Ime: Ana123!	Korisnik nije prijavljen i pojavljuje se poruka o pogrešci "Korisničko ime i lozinka ne odgovaraju nijednom korisniku u ovoj usluzi"	**Visok**			**Test je prošao, pojavila se poruka o pogrešci.**
				2.	U tekstualni okvir "password" upišite "Ana123!"	Lozinka: Ana123!					
				3.	Kliknite na "Log in" gumb						
Odjava (Log out)	Odjava s korisničkog računa	Provjerite je li se moguće odjaviti s korisničkog računa	Idite na: https://www.saucedemo.com/	1.	Kliknite na izbornik u gornjem lijevom kutu	Korisnik1	Korisnik je odjavljen - https://www.saucedemo.com/	**Visok**	**Prolaz**	**Prolaz**	
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na "Log out" gumb	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Košarica	Dodavanje svih proizvoda u košaricu s korisnicima "standard_user" i "problem_user"	Provjerite je li moguće dodavanje svih proizvoda u košaricu s korisnicima "standard_user" i "problem_user"	Idite na: https://www.saucedemo.com/	1.	Kliknite na gumb "Add to cart" za svaki proizvod	Korisnik1	Svi proizvodi su uspješno dodani u košaricu	**Visok**	**Prolaz**	**Test nije prošao**	Neki proizvodi kod korisnika "problem_user" se ne mogu dodati u košaricu
			Ulogirajte se s korisnikom: standard_user	2.	Klikom na gum "Košarica" u gornjem desnom kutu provjerite na	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Košarica	Uklanjanje svih proizvoda iz košarice s korisnicima "standard_user" i "problem_user"	Provjerite je li moguće uklanjanje svih proizvoda iz košarice s korisnicima "standard_user" i "problem_user" klikom na gumb "Remove" unutar košarice	Idite na: https://www.saucedemo.com/	1.	Kliknite na košaricu						
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "Remove" kako biste uklonili proizvode iz košarice	Korisnik1	Svi proizvodi su uspješno uklonjeni iz košarice	**Visok**	**Prolaz**	**Test nije prošao**	Kod korisnika problem_user neki proizvodi se ne mogu ukloniti iz košarice klikom na gumb "Remove"
			Ulogirajte se s korisnikom: problem_user			Ime: standard_user					
			Dodajte sve proizvode u košaricu			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Košarica	Uklanjanje svih proizvoda iz košarice s korisnicima "standard_user" i "problem_user"	Provjerite je li moguće uklanjanje svih proizvoda iz košarice s korisnicima "standard_user" i "problem_user" klikom na gumb "Remove" na naslovnoj stranici web shop-a	Idite na: https://www.saucedemo.com/	1.	Kliknite na gumb "Remove", na naslovnoj stranici web shop-a, za svaki proizvod koji je dodan u košaricu	Korisnik1	Svi proizvodi su uspješno uklonjeni iz košarice	**Visok**	**Prolaz**	**Test nije prošao**	Kod korisnika "problem_user" proizvodi nisu uklonjeni iz košarice klikom na gumb "Remove" na naslovnoj stranici web shop-a.
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
			Dodajte sve proizvode u košaricu			Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Narudžba	Prazna narudžba	Provjerite je li nemoguće dovršiti praznu narudžbu.	Idite na: https://www.saucedemo.com/	1.	Kliknite na praznu košaricu	Korisnik1	Trebalo bi biti nemoguće dovršiti praznu naridžbu.	**Visok**	**Test nije prošao**	**Test nije prošao**	Kod korisnika standard_user moguće je dovršiti praznu narudžbu. Kod korisnika problem_user nije moguće popuniti polje "prezime".
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "checkout"	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user	3.	Unesite informacije o kupcu (checkout information)	Lozinka: secret_sauce					
				4.	Kliknite na gumb "Continue"	Checkout information: First name: Ana Last name: Milin Zip/postal code: 21000					
				5.	Kliknite na gumb "Finish"						
Narudžba	Spremljeni podaci o kupcu i naplati pri povratku od - https://www.saucedemo.com/checkout-step-two.html do - https://www.saucedemo.com/checkout-step-one.html"	Provjerite hoće li podaci o kupnji biti spremljeni kada se kupac vrati s - https://www.saucedemo.com/checkout-step-two.html na - https://www.saucedemo.com/checkout-step-one.html"	Idite na: https://www.saucedemo.com/	1.	Kliknite na košaricu	Korisnik1	Podaci o kupcu i naplati bi trebali ostati spremljeni pri povratku s https://www.saucedemo.com/checkout-step-two.html	**Srednji**	**Test nije prošao**	**Test nije prošao**	Podaci o kupcu i naplati ne ostanu spremljeni pri povratku s https://www.saucedemo.com/checkout-step-two.html.
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "checkout"	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user	3.	Unesite informacije o kupcu (checkout information)	Lozinka: secret_sauce					

				4.	Kliknite na lijevu strelicu kako biste došli natrag na - https://www.saucedemo.com/	Checkout information: First name: Ana Last name: Milin Zip/postal code: 21000					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Narudžba	Informacijski obrazac prepoznaje iste podatke u svakom polju kao one koji nisu validni.	Provjerite je li informacijski obrazac prepoznaje iste podatke u svakom polju kao one koji nisu validni.	Idite na: https://www.saucedemo.com/	1.	Kliknite na košaricu	Checkout information: First name: Ana!23 Last name: Ana!23 Zip/postal code: Ana!23	Pojavljuje se poruka o pogrešci "Podaci o naplati nisu valjani" i kupac ne može nastaviti na sljedeći korak	**Visok**	**Test nije prošao**	**Test nije prošao**	Poruka o pogrešci "Podaci nisu valjani" se nije pojavila i kupac može nastaviti na slijedeći korak. Kod korisnika problem_user nije moguće popuniti polje "prezime".
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "checkout"	Korisnik1					
				3.	Unesite informacije o kupcu/naplati, isti podatak u svako polje	Ime: standard_user					
				4.	Kliknite na gumb "continue"	Lozinka: secret_sauce					
Narudžba	Ispunjavanje informacijskog obrasca i dovršetak narudžbe	Provjerite je li moguće uspješno ispunjavanje informacijskog obrasca i dovršetak na rudžbe.	Idite na: https://www.saucedemo.com/	1.	Kliknite na košaricu	Checkout information: First name: Ana Last name: Milin Zip/postal code: 21000	Informacijski obrazac je uspješno ispunjen i kupac može dovršiti narudžbu.	**Visok**	**Prolaz**	**Test nije prošao**	Kod korisnika problem_user nije moguće ispuniti polje prezime (Last Name) u informacijskom obrazcu, pa nije moguće niti dovršiti narudžbu.
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "checkout"	Korisnik1					

			Ulogirajte se s korisnikom: problem_user	3.	Unesite informacije o kupcu	Ime: standard_user					
			Dodajte proizvode u košaricu	4.	Kliknite na gumb "continue"	Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Narudžba	Ukupna suma narudžbe na računu je točna.	Provjerite je li ukupna suma narudžbe na računu točna.	Idite na: https://www.saucedemo.com/	1.	Kliknite na košaricu	Checkout information: First name: Ana Last name: Milin Zip/postal code: 21000	Ukupna suma narudžbe na računu je točna.	**Visok**	**Prolaz**	**Test nije prošao**	
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "checkout"	Korisnik1					
			Ulogirajte se s korisnikom: problem_user	3.	Unesite informacije o kupcu	Ime: standard_user					
			Dodajte proizvode u košaricu	4.	Kliknite na gumb "continue"	Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Online trgovina (Web shop)	Sve slike proizvoda točno prikazuju opisani proizvod	Provjerite je li sve slike proizvoda točno prikazuju opisani proizvod	Idite na: https://www.saucedemo.com/	1.	Pogledajte sliku svakog proizvoda	Korisnik1	Slika svakog proizvoda odgvora opisu tog proizvoda.	**Visok**	**Prolaz**	**Test nije prošao**	Kod korisnika problem_user umjesto slika proizvoda, vidljive su samo slike psa.
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user	.		Lozinka: secret_sauce					
						Korisnik2					

						Ime: problem_user					
						Lozinka: secret_sauce					
Online trgovina (Web shop)	Testiranje vidljivosti ikone košarica	Provjerite je li ikona košarice vidljiva na stranici web shop-a	Idite na: https://www.saucedemo.com/	1.	Pogledajte nalazi li se ikona košarice u gornjem desnom kutu	Korisnik1	Ikona košarice je vidljiva.	**Visok**	**Prolaz**	**Prolaz**	
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Online trgovina (Web shop)	Testiranje vidljivosti naziva "Swag Labs"	Provjerite je li naziv"Swag Labs" vidljiv na stranici web shop-a	Idite na: https://www.saucedemo.com/	1.	Pogledajte nalazi li se naslov "Swag Labs" gore, na sredini web shop-a.	Korisnik1	Naslov "Swag Labs" je vidljiv.	**Visok**	**Prolaz**	**Prolaz**	
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Online trgovina (Web shop)	Testiranje vidljivosti ikone "Menu"	Provjerite je li ikona "Menu" vidljiva na stranici web shop-a	Idite na: https://www.saucedemo.com/	1.	Pogledajte nalazi li se ikona "Menu" u gornjem lijevom kutu.	Korisnik1	Ikona "Menu" je vidljiva.	**Visok**	**Prolaz**	**Prolaz**	
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					

			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Online trgovina (Web shop)	Testiranje vidljivosti slike proizvoda, cijene proizvoda, naziva i opisa proizvoda te gumba "Add to cart" na jednoj kartici proizvoda.	Provjerite jesu li svi elementi kartice jednog proizvoda vidljivi, slika proizvoda, cijena proizvoda, naziv i opisa proizvoda te gumb "Add to cart"	Idite na: https://www.saucedemo.com/	1.	Pogledajte je li vidljiva slika proizvoda na jednoj kartici.	Korisnik1	Na jednoj kartici su vidljivi slika proizvoda, cijena proizvoda, naziv proizvoda, opisa proizvoda i gumb "Add to cart".	**Visok**	**Prolaz**	**Test nije prošao**	Kod problem_user su umjesto slika proizvoda vidljive slike psa.
			Ulogirajte se s korisnikom: standard_user	2.	Pogledajte je li na istoj kartici vidljiv naziv proizvoda.	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user	3.	Pogledajte je li na istoj kartici vidljiv opis proizvoda.	Lozinka: secret_sauce					
				4.	Pogledajte je li na istoj kartici vidljiva cijena proizvoda.	Korisnik2					
				5.	Pogledajte je li na istoj kartici vidljiv "Add to cart" gumb.	Ime: problem_user Lozinka: secret_sauce					
Online trgovina (Web shop)	Testiranje vidljivosti filtera	Provjerite je li filter vidljiv u gornjem desnom kutu	Idite na: https://www.saucedemo.com/	1.	Pogledajte nalazi li se filter u gornjem desnom kutu.	Korisnik1	Filter je vidljiv i nalazi se u gornjem desnom kutu.	**Srednji**	**Prolaz**	**Prolaz**	
			Ulogirajte se s korisnikom: standard_user			Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					

Menu	Nakon što se klikne na gumb "All Items" unutar menua, svi proizvodi će biti izlistani.	Provjerite je li se izlistaju svi proizvodi nakon što se klikne na gumb "All Items" unutar menua.	Idite na: https://www.saucedemo.com/	1.	Kliknite na ikonu menua u gornjem lijevom kutu	Korisnik1	Nakon klika na gumb "All Items" unutar menua, svi proizvodi će biti izlistani.	**Visok**	**Prolaz**	**Prolaz**	(Ako se u obzir uzima screenshot slika proizvoda test će pasti.)
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "All Items"	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Menu	Gumb "About", unutar menua vodi na stranicu https://saucelabs.com/	Provjerite je li gumb "About" unutar menua vodi na stranicu https://saucelabs.com/	Idite na: https://www.saucedemo.com/	1.	Kliknite na ikonu menua u gornjem lijevom kutu	Korisnik1	Nakon klika na gumb "About" korisnik je uspješno preusmjeren na https://saucelabs.com/	**Nizak**	**Prolaz**	**Test nije prošao**	Kod korisnika problem_user gumb "About" vodi na https://saucelabs.com/error/404
			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "About"	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					
Menu	Klikom na gumb "Reset App State" unutar menua, sve se postavke računa resetiraju.	Provjerite je li se klikom na gumb "Reset App State" unutar menua, sve postavke računa resetiraju.	Idite na: https://www.saucedemo.com/	1.	Kliknite na ikonu menua u gornjem lijevom kutu	Korisnik1	Nakon klika na gumb "Reset State App" sve postavke korisničkog računa su resetirane.	**Visok**	**Test nije prošao**	**Test nije prošao**	Nakon klika na gumb "Reset App State" samo dio postavki je resetiran. Proizvodi su manknuti iz košarice, ali "Remove" gumb još uvijek je vidljiv, a automatski bi se trebao promijeniti u "Add to Cart" gumb.

			Ulogirajte se s korisnikom: standard_user	2.	Kliknite na gumb "Reset App State"	Ime: standard_user					
			Ulogirajte se s korisnikom: problem_user			Lozinka: secret_sauce					
						Korisnik2					
						Ime: problem_user					
						Lozinka: secret_sauce					