Manualno i automatizirano testiranje korisničkih sučelja web aplikacija

Majstorović, Luka

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Majstorović

MANUALNO I AUTOMATIZIRANO TESTIRANJE KORISNIČKIH SUČELJA WEB APLIKACIJA

DIPLOMSKI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Majstorović

Matični broj: 0016133143

Studij: Informacijsko i programsko inženjerstvo

MANUALNO I AUTOMATIZIRANO TESTIRANJE KORISNIČKIH SUČELJA WEB APLIKACIJA

DIPLOMSKI RAD

Mentorica:

Prof. dr. sc. Valentina Kirinić

Varaždin, rujan 2023.

Luka Majstorović

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Na početku teorijskog dijela rada opisani su osnovni elementi web aplikacije. Zatim je opisana osnovna uloga i svrha testiranja u razvoju softvera. Nakon toga, opisano je manualno testiranje, te sve njegove vrste (npr. funkcionalno, regresijsko, pristupačnosti i sl.). U ovom dijelu je također opisan način izrade testnih scenarija i izvođenja testiranja, izrada testnih izvještaja i izvještaja o greškama (*engl. bug report*).

Nakon opisa manualnog testiranja, predstavljeno je automatsko testiranje te njegove vrste (funkcionalno, testiranje krajnjeg korisnika (*engl. end-to-end*), jedinično testiranje (*engl. unit)* itd.). Uz to, načini pisanja automatiziranih testova te nekoliko alata za pisanje koji ne zahtijevaju pisanje koda (*engl. No-Code*) i nekoliko programskih jezika za pisanje testova.

U praktičnom dijelu iskorišteni su testni scenariji opisani u teorijskom dijelu rada te napisani testovi za testiranje web aplikacije. Za pisanje testova korišten je programski jezik JavaScript te biblioteka za pisanje testova *Cypress*. Prije samog opisa koda i strukture testova, opisan je korišteni programski jezik i njegove osnovne funkcije. Zatim je opisan programski kod te izrade izvještaja i čitanje istih. Za sam kraj praktičnog dijela, daje se usporedba manualnog i automatskog testiranja te načina na koji se odabiru i prioritiziraju.

Ključne riječi: testiranje, *JavaScript*, *Cypress*, automatizacija, manualno, softver, web, kvaliteta

Sadržaj

1. Uvod

U svijetu gdje je razvoj softvera svakodnevna pojava i gdje je softver dio svakodnevnog života, potreba za održavanjem i osiguravanjem kvalitete i ispravnog funkcioniranja softvera je sve veća.

Kako je razvoj softvera napredovao, tako su napredovale i tehnike održavanja i osiguravanja kvalitete istog. Mnogobrojne tehnike testiranja na tržištu mogu se svrstati u dvije glavne skupine, manualne i automatske načine testiranja softvera.

Ovaj rad obuhvaća temu manualnog i automatskog testiranja softvera. Na samom početku objašnjeni su koncepti web stranice te je čitatelj upoznat s bitnim informacijama i pojmovima za lakše shvaćanje daljnjeg rada. Zatim su kroz opise i primjere čitatelju približeni pojmovi automatskog i manualnog testiranja, njihov način izvođenja, vrste te struktura. Prikazani su primjeri manualnih testnih scenarija, izvođenja manualnog testiranja te pravila pisanja manualnih testova. Isto tako, pokriveno je područje pravila pisanja automatizacije te automatiziranih testova.

Teorija je obuhvaćena praktičnim primjerom koji na primjeru web aplikacije prikazuje primjenu koncepata manualnog i automatiziranog testiranja. Prikazano je pisanje i izvođenje manualnih testova na konkretnom primjeru. Također, prikazan je programski kod i pravilno strukturiranje automatiziranih testova te izvođenje istih.

Na kraju praktičnog dijela prikazani su testni izvještaji izvođenja testiranja te dani zaključci koje se iz istih mogu iščitati.

Na samom kraju rada, poglavlje o usporedbi manualnog i automatskog testiranja čitatelju daje više informacija o manama i prednostima vrsti testiranja te o samoj prioritizaciji i logici kod odabira testiranja u projektu.

Razlog odabira teme je njena aktualnost i sve veća važnost u razvoju programskih proizvoda. Gotovo svaki razvojni tim današnjice sadrži odjel osiguranja kvalitete koji obuhvaća širok spektar poslova.

2. Metode i tehnike rada

Za izradu ovog rada korištene su detaljne istraživačke metode i tehnike koje su pokrijepljene iskustvom i znanjem dobivenim iz prakse vezane za temu rada.

Obzirom da se radi o temi koja je vezana za razvoj softvera, kao izvori većinom su korištene aktualne službene web stranice alata i tehnologija koje su korištene. Razlog tome je njihova aktualnost te provjerenost podataka.

U praktičnom radu korišten je alat *Cypress* koji je baziran na programskom jeziku *Javascript*. Korišten je za pisanje automatskih testova.

Za prikazivanje i usporedbu podataka korišteni su izvještaji, kako generirani od strane drugog softvera, tako kreirani od strane autora.

3. Web aplikacija

Kako bi tema testiranja web aplikacije bila jasnija i kako bi posao bio dobro obavljen, potrebno je poznavati osnovne informacije o web aplikacijama. Ovo poglavlje daje bolji uvid u web aplikacije, njihovu strukturu, primjere web aplikacija te razliku između web aplikacije i web stranice.

3.1. Način rada web aplikacije

Web aplikacije su najčešće izrađene u programskim jezicima poput JavaScript i skriptom jeziku HTML, obzirom na to da se navedeni jezici oslanjaju na preglednik kako bi prikazao (*engl. render*) stranicu. Aplikacije najčešće dolaze u dvije vrste, statičke (*engl. static*) i dinamičke (*engl. dynamic*). (StackPath, bez dat.)

U narednom tekstu opisan je primjer načina rada web aplikacije:

1. Korisnik kreira zahtjev prema poslužitelju (*engl. server)* preko interneta
2. Web poslužitelj (*engl. web server)* prosljeđuje zahtjev na aplikacijski poslužitelj (*engl. application server)*
3. Aplikacijski poslužitelj (*engl. application server*) obavlja logičke zadatke poput: dohvaćanja podataka iz baze, procesuiranje podataka i slično.
4. Aplikacijski poslužitelj (*engl. application server*) vraća odgovor i procesuirane podatke

   Web poslužitelj (*engl. web server)* po potrebi obrađuje podatke i prikazuje ih korisniku

3.2. Primjer web aplikacija

Web aplikacije imaju široku primjenu i korištene su svakodnevno u različitim dijelovima naših života. Njihova primjena nema granice. Koriste se u sportu, zabavi, medicini, znanosti, turizmu i u još mnogo drugih aspekata naših života.

Mogu imati različite uloge, poput npr.: Obrada fotografije, procesuiranje teksta, konverzija datoteka, slanje poruka, gledanje videa itd.

Poznatiji primjeri aplikacija su: Google Karte, Google Dokumenti, Google Tablice, Facebook, Instagram, WhatsApp Web i mnoge druge. (StackPath, bez dat.)

3.3. Razlika između web aplikacije i web stranice

Danas često miješani pojmovi su Web aplikacija i Web stranica. Iako im je sama srž ista, funkcionalno se razlikuju.

Obje su Web mjesta koje žive na određenom poslužitelju, na određenoj domeni, no do razlike dolazi u njihovoj funkciji.

Web stranica je primarno statičke prirode, što znači da nema gotovo nikakve obrade podataka i primanja ulaznih podataka korisnika, nego se uglavnom radi o prikazivanju statičkog, rijetko mijenjanog sadržaja. Sama izrada i prezentiranje su jednostavni, te je opseg posla i potreba za radnom snagom kod izrade jednostavna. Najčešće se izradom ovakvog tipa stranice bave Web dizajneri (*engl. Web designer*). Primjeri ovakvih stranica su stranice s dnevnim novinama i vijestima, prikaz prognoze, sportske novosti i slično. (GeeksForGeeks, 2022)

Web aplikacija je primarno izrađena za interakciju s korisnicima. To znači da se radi o konstantnoj razmjeni podataka između korisnika i aplikacije. Za samu izradu je najčešće potreban opsežniji tim koji se sastoji od stručnjaka za izradu korisničkog dijela (*engl. frontend*) te serverskog, odnosno aplikacijskog dijela (*engl. backend*). Primjeri ovakvih aplikacija su Facebook, Instagram, Njuškalo i slično. (GeeksForGeeks, 2022)

4. Manualno testiranje

Manualno testiranje je proces kojim se “ručno”, odnosno manualno odrađuju koraci koje će potencijalni korisnici raditi u svrhu potvrđivanja ispravnog rada. Ova generička definicija se može primijeniti na sve oblike i vrste testiranja, ne vezano za softver.

Kod manualnog testiranja u informatici, primarna svrha je otkrivanje i prepoznavanje grešaka *(engl. bug)*, kako bi se isti ispravili te kako ne bi narušili iskustvo krajnjih korisnika.

Manualno testiranja je potencijalno zahtjevan i dugotrajan proces, ovisno o kompleksnosti i opsegu softvera koji se testira. Ispravno planiranje, priprema te dokumentacija su ključ ispravnog i kvalitetno obavljenog testiranja.

Danas se testiranje u informatici najčešće spominje u obliku uloge testera, koja kao primarni zadatak ima pisanje i izvršavanje testnih scenarija, te uloge osiguranja kvalitete *(engl. Quality assurance)* koja uz samo testiranje ima i opširniji zadatak vezan za kontrolu kvalitete izrade softvera te samog procesa izrade.

Ovo poglavlje opisuje vrste manualnog testiranja, pokriva proces kreiranja testnih scenarija, izvođenja istih scenarija te generiranja testnih izvještaja koji pomažu u praćenju napretka i stanja programskog proizvoda.

4.1. Vrste manualnog testiranja

U aktualne vrste manualnog testiranja spadaju: Testiranje metodom Crne kutije *(engl. Black Box)*, Testiranje metodom Bijele kutije *(engl. White Box)*, Jedinično testiranje *(engl. Unit)*, Sistemsko Testiranje *(engl. System),* Integracijsko testiranje *(engl. Integration)* te Testiranje prihvatljivosti *(engl. Acceptance).* (guru99, 2023b)

U daljnjem tekstu detaljnije su opisane sve vrste gore navedenih testiranja uz navedene svrhe i ciljeve.

4.1.1. Testiranje metodom Crne kutije

Testiranje metodom crne kutije predstavlja proces u kojem osoba koja testira ne poznaje pozadinu i način na koji softver funkcionira. Odnosno, tester izvodi određene korake te dobiva krajnji rezultat, bez da ima ikakve informacije koja je logika iza tog rezultata. Tako se ostvaruje posvećenost i fokus na krajnji rezultat, odnosno na ono što krajnji korisnici vide. (guru99, 2023e)

4.1.2. Testiranje metodom Bijele kutije

Testiranje metodom bijele kutije testeru daje pristup kodu od kojeg je programski proizvod načinjen. Za ovu vrstu testiranja potrebno je veće poznavanje programskog koda i načina strukturiranja istog. Iz tog razloga, ova vrsta testiranja je često obavljena od strane razvojnog tima, a ne tima testera ili osiguranja kvalitete. Vrijednost koju ova vrsta testiranja donosi je to što se uz samu funkcionalnost, testiraju kvaliteta, održivost i sigurnost u samom kodu. (guru99, 2023h)

4.1.3. Jedinično testiranje

Ova vrsta testiranja se obavlja tijekom samog razvoja aplikacije. Svrha je da se testira i osigura kvaliteta svakog pojedinačnog dijela koda, odnosno metoda. Ideja je da svaka posebna metoda ima svoj test ili testove u posebnim slučajevima. Svrha tih testova je da osiguraju da metoda daje očekivani izlaz, odnosno "odgovor". Ovime se osigurava kvaliteta koda na najmanjoj razini i ove testove najčešće pišu developeri. (guru99, 2023d)

4.1.4. Sistemsko testiranje

Sistemsko testiranje se najčešće izvodi na samom kraju razvojnog procesa, odnosno nakon implementacije programskog proizvoda. Razlog tomu je što je cilj osigurati da proizvod funkcionira u svojoj okolini, odnosno u sistemu u kojem je implementiran. Kako je programski proizvod samo dio kompletnog sustava u koji spadaju i komponente, drugi softver, sistemski softver itd., potrebno je dodatno potvrditi uspješno funkcioniranje softvera u tom okruženju. (guru99, 2023f)

4.1.5. Integracijsko testiranje

S obzirom na to da je programski proizvod rijetko načinjen od jednog modula, potrebno je potvrditi međusobnu integraciju modula i funkcionalnosti. Za to služi integracijsko testiranje. Kada se na već postojeću funkcionalnost ili na grupu funkcionalnosti dodaje nova funkcionalnost, potrebno je potvrditi da je integracija nove funkcionalnosti uspješna. (guru99, 2023a)

4.1.6. Testiranje prihvatljivosti

Testiranje prihvatljivosti, odnosno korisničke prihvatljivosti *(engl. User Acceptance Testing)*, obavljaju krajnji korisnici proizvoda. Njihov je zadatak koristiti proizvod u stvarnom okruženju sa stvarnim primjerima i slučajevima korištenja. Tako se potvrđuje da je proizvod dosegao kvalitetu i funkciju koja se od njega očekuje.

Bitno je napomenuti da se od testiranja prihvatljivosti ne očekuje da korisnici pronalaze greške u radu aplikacije, nego da provjere i pronađu potencijalne promjene koje žele implementirati. (guru99, 2023g)

4.2. Izrada testnih scenarija

Kao što je dokumentacija srž razvoja svakog programskog proizvoda, tako su testni scenariji srž, odnosno dokumentacija svakog testnog procesa.

Dokument testnih scenarija je dokument koji predstavlja popis svih testnih slučajeva koji trebaju biti obavljeni kako bi se potvrdio programski proizvod.

Da bi testni scenariji bili ispravno napravljeni i da bi uistinu služili svoju svrhu, potrebno ih je kvalitetno napisati. (Paul C. Jorgensen, 2014)

Za pisanje testnih scenariji, potrebno je odabrati alat u kojem će testni scenariji biti napisani. Taj alat može biti u potpunosti jednostavan poput Microsoft Word-a, *Google Sheet*-a te *Jira*-e. Uz to, postoje i kompleksniji alati koji su specijalizirani za upravljanje testnim scenarijima, poput: *TestRail*-a, *Testpad*-a, *PractiTest*-a i drugih. Jednostavni alati su besplatni i jednostavni za postaviti i krenuti sa samim pisanjem. Njihova mana je to što je održavanje, odnosno uređivanje, dodavanje i brisanje testova otežano, obzirom da to nije njihova osnovna namjera. Dok je samo pisanje i održavanje testova uvelike olakšano, specijalizirani alati se najčešće plaćaju i nisu pogodni za manje projekte.

Ne postoji definirana struktura testnih scenarija, ali postoji velika količina elemenata koji se najčešće koriste u definiciji strukture testnih scenarija, navedeni u tekstu ispod:

Identifikator scenarija - koristi se kako bi se jednostavno i efikasno pronašao testni scenarij. Isto tako, najčešće se koristi kako bi se scenarij povezao s drugim scenarijem ili scenarijima u automatizaciji.

Naziv scenarija - cilj naziva scenarija je da ukratko opiše svrhu scenarija kako bi se brzo i jednostavno mogao identificirati. Naziv scenarija mora biti definitivan i ne davati prostora za tumačenje na više načina.

Testni koraci - Testni koraci su popis koraka koje tester mora napraviti kako bi uspješno izvršio testni slučaj.

Očekivani rezultat - Ovaj stupac predstavlja koji je očekivani rezultat nakon što su testni koraci izvršeni. Očekivani rezultat odlučuje o tome je li test bio uspješan ili neuspješan.

Rezultat testa - Rezultat testa je jednostavan stupac koji prikazuje je li izvođenje testa bilo uspješno ili ne.

Ovo su osnovni elementi koje bi svaki popis testnih scenarija trebao sadržavati. Uz to, mogu se koristiti dodatni elementi koji odgovaraju i pomažu pri organizaciji projekta. (Paul C. Jorgensen, 2014)

Na slici 1, vidljiv je primjer tri testna scenarija uz dodatan stupac koji opisuje grešku u slučaju pada testa. Isto tako, testovi su napisani u Google tablicama koje su često korišten alat za kreiranje testne dokumentacije.

Identifikator Scenarija	Naziv Scenarija	Preduvjeti	Testni koraci	Očekivani rezultat	Rezultat testa	Opis greške
TC-1	Dodavanje novog računa	Korisnik je uspješno prijavljen i ima pravo kreiranja novog računa	1. Pritisni gumb za kreiranje novog računa 2. Popuni sva obavezna polja 3. Popuni sva ne obavezna polja 4. Pritisni gumb za spremanje računa	Poruka o uspješnosti kreiranja računa je prikazana te je račun vidljiv u popisu računa.	Prošao	-
TC-2	Brisanje računa	Korisnik je uspješno prijavljen i ima pravo brisanja računa. Postoji prethodno kreiran račun.	1. Odaberi račun 2. Pritisni gumb za brisanje računa 3. Potvrdi radnju klikom na "Da"	Poruka o uspješnosti brisanja računa je prikazana te račun nije vidljiv u popisu računa.	Prošao	-
TC-3	Uređivanje računa	Korisnik je uspješno prijavljen i ima pravo uređivanja računa. Postoji prethodno kreiran račun.	1. Odaberi račun 2. Pritisni gumb za uređivanje računa 3. Promijeni polje "Šifra računa" 4. Pritisni gumb za spremanje računa	Poruka o uspješnosti uređivanja računa je prikazana i račun s promjenjenom šifrom je vidljiv u popisu računa.	Pao	Šifra računa nije promjenjena nakon spremanja uređivanja.

Slika 1: Primjer popisa testnih scenarija (Izvor: Autor)

4.3. Izvođenje testnih scenarija

Izvođenje je glavni akcijski korak u testnom procesu. U ovom koraku se izvode prethodno definirani testni scenariji kako bi se potvrdio softver. Kod samog izvođenja scenarija, nerijetko se dogodi da se pronađu testni scenariji koji nisu evidentirani, te je bitno popisati i dokumentirati te iste scenarije.

Uz samo evidentiranje propuštenih scenarija, bitno je pisati i evidentirati sva neočekivana ponašanja softvera. To se najčešće obavlja u obliku kreiranja novih kartica *(engl. Ticket)* koje opisuju nastali problem ili neočekivano ponašanje.

Testni scenariji mogu se izvoditi na više načina, a najčešće se koriste u Validacijskom i Regresijskom testiranju. Validacijsko testiranje je fokusirano na potvrđivanje novo kreirane funkcionalnosti i potrebno je uspješno izvesti sve scenarije kako bi se potvrdio softver. Uz validacijsko testiranje, scenariji se koriste i u regresijskom testiranju koje obuhvaća cijelu aplikaciju te se koriste testni scenariji, ili uži skup testnih scenarija kako bi se nanovo potvrdio softver.

4.4. Testni izvještaj

Testni izvještaj je dokument koji sadrži sažetak svih testnih aktivnosti i krajnjih rezultata testiranja proizvoda. Testni izvještaj je dokument koji govori koliko dobro je testiranje obavljeno i koliko dobro je sam proizvod kreiran. Testni izvještaj je odličan izvor informacija za voditelje projekta i voditelje razvojnog tima kako bi mogli napraviti odluke vezane za daljnji razvoj proizvoda. (guru99, 2023c)

Struktura testnog izvještaja, isto kao i kod testnih scenarija, ovisi o projektu i potrebama projekta. Iako nije striktna struktura, jedna od najčešćih struktura i elemenata koje testni izvještaj treba sadržati je sljedeća:

1. Informacije o projektu
2. Cilj testiranja
3. Sažetak testnog procesa
4. Greške

4.4.1. Informacije o projektu

Informacije o projektu se sastoje od imena projekta te opisa samog projekta. Cilj je kako bi čitatelju dalje okvirne informacije o projektu te kontekst u kojem je testiranje izvršeno. Primjer prikaza informacija o projektu vidljiv je na slici 2.

Informacije o projektu			
Naziv projekta:	Aplikacija za izdavanje računa		
Naziv proizvoda:	Aplikacija za izdavanje računa		
Opis proizvoda:	Aplikacija za svrhu ima opciju kreiranja, brisanja i uređivanja računa.		
Vrijeme početka:	18.08.2023	**Vrijeme završetka:**	20.08.2023

Slika 2: Primjer informacija o projektu (Izvor: Autor)

4.4.2. Cilj testiranja

Cilj testiranja se sastoji od informacija koje definiraju vrstu testiranja te svrhu testiranja. Taj cilj govori što se pokušava postići izvršenim testiranjem. Na slici 3 prikazan je primjer popisa ciljeva testiranja.

Vrsta testiranja	Cilj testiranja
Ad hoc testiranje	Prilikom samog razvijanja proizvoda, povremeno je potrebno istestirati funkcionalnosti kako bi se istražio proizvod i potencijalno pronašli defekti i ne slaganja u procesima kreiranja i brisanja računa.
Validacijsko testiranje	Nakon završenog kreiranog *Story*-a, potrebno je testirati cijeli *Story* te validirati sve kriterije prihvatljivosti
Integracijsko testiranje	Nakon završengo *Story*-a, potrebno je testirati cijeli *Story* u kontekstu ostalih funkcionalnosti i potvrditi kriterije integracije
Regresijsko testiranje	Nakon završenog *Story-a*, potrebno je testirati cijeli proizvod i prethodno potvrđene funkcionalnosti te potvrditi ispravno funkcioniranje.

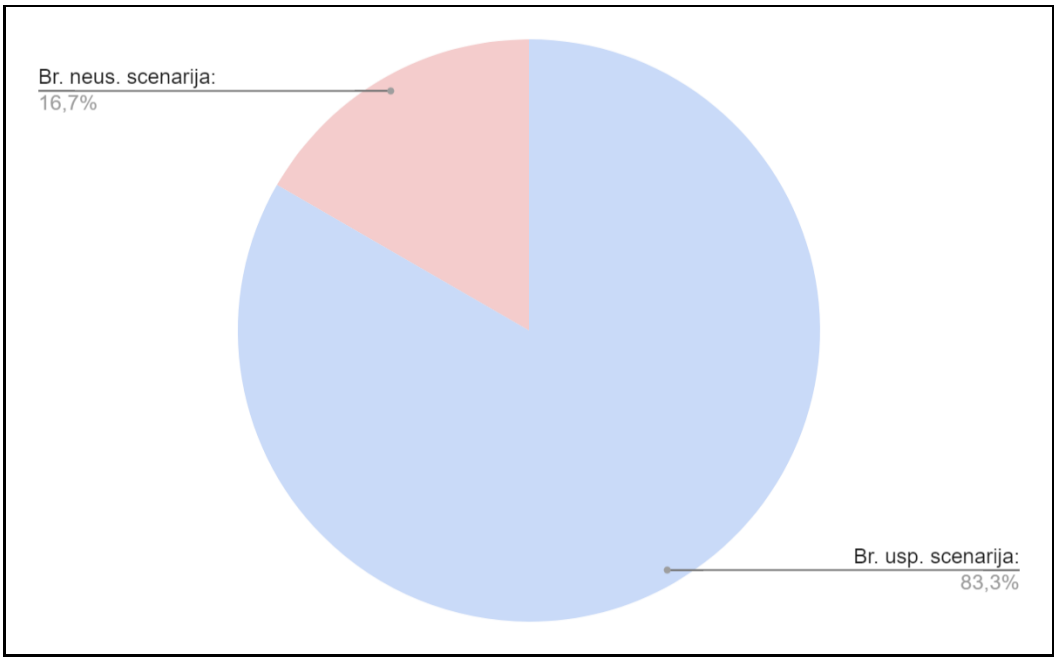
Slika 3: Primjer ciljeva testiranja (Izvor: Autor)

4.4.3. Sažetak testnog procesa

Sažetak testnog procesa za svrhu ima definirati detaljne informacije o testnom procesu poput: broj izvršenih scenarija, broj uspješnih scenarija, broj neuspješnih scenarija, postotak uspjeha, postotak neuspjeha, komentar i slično. Sažetak testnog procesa poželjno je prikazati u jednom od sljedećih oblika, a potencijalno u oba. Na slici 4 se nalazi prikaz u obliku tablice koji je potpomognut grafičkim prikazom koji se vidi na slici 5.

Sažetak testnog procesa	
Broj izvršenih scenarija:	24
Broj uspješnih scenarija:	20
Broj neuspjelih scenarija:	4
Postotak prolaznosti:	83%
Komentar:	Iako je postotak prolaznosti zadovoljen (80%), testiranje je neuspješno zbog utjecaja grešaka na proizvod

Slika 4: Primjer sažetka testnog procesa (Izvor: Autor)



Slika 5: Primjer ishoda testiranja u grafičkom obliku (Izvor: Autor)

4.4.4. Greške

Greške daju informacije o svim greškama, razlogu nastajanja, njihovoj kritičnosti te svim drugim informacijama koje mi smatramo da su bitne za naš projekt.

Broj greške je jedinstveni identifikator svake greške. Tako možemo pratiti grešku kroz cijeli proces njenog prijavljivanja, popravljanja te ponovne validacije.

Status daje informaciju u kojem stanju je prijavljena greška. Možemo vidjeti da je greška 1 zatvorena, odnosno popravljena i validirana, dok je greška 1 otvorena, što nam daje do znanja da još nije ispravljena

Prijavio je stupac koji daje informacije o osobi koja je prijavila problem. Tako, možemo kontaktirati osobu u slučaju da postoje dodatna pitanja te kako bi se dodatno potvrdio problem nakon ispravljanja. Ovaj stupac je izrazito koristan kod većih timova testera.

Verzija testiranja daje informacije o verziji softvera koju smo testirali. Kao prethodni stupac, izrazito je koristan kod softvera koji ima veći broj razvojnih verzija u isto vrijeme. Tako programeri mogu na ispravnoj verziji provjeriti i ispraviti problem.

Identifikator scenarija je poveznica na testni scenarij koji je izvršen, te nakon kojeg je problem uočen.

Koraci su jedan od najbitnijih elemenata izvještaja o greškama. Daju informaciju programerima te svima zainteresiranima koje korake je korisnik, odnosno tester napravio netom prije pogreške. Dobro napisani koraci za ponavljanje problema mogu napraviti razliku

između jednostavnog i brzog ispravljanja problema i napornog, kompliciranog procesa koji je gubitak vremena.

Očekivani rezultat je ono što je korisnik, odnosno tester očekivao da će se dogoditi nakon samog izvođenja koraka. Jednostavnije, očekivano ponašanje aplikacije.

Rezultat je rezultat koji se dogodio nakon izvođenja koraka. Ovo je najčešće opis problema i ne slaganja s originalnim zahtjevom.

Prioritet predstavlja koliko je hitno da se problem ispravi. Ova informacija varira ovisno o projektu i trenutnom stanju projekta. Isto tako, ovisi, ali nije striktno vezana za utjecaj samog problema na rad aplikacije. Primjer je greška 1 na slici 5. Kod ove greške funkcionalnost u pravilu radi, ali se korisniku ne prikaže poruka o uspješnosti. U ovoj situaciji, korištenje aplikacije nije blokirano te je prioritet svrstan kao nizak. Kod greške 2, situacija je kompleksnija. U ovom slučaju, prioritet je visok zato što greška u potpunosti blokira korištenje aplikacije.

Utjecaj je usko povezani pojam uz prioritet. Njihov odnos je prikazan na grešci 2 koja za prioritet ima postavljen visok, ali je utjecaj srednji. U ovom slučaju, razlog tome je činjenica da se uređivanje računa ne koristi često, te je iz tog razloga utjecaj srednji. Iako funkcionalnost nije korištena često i njen utjecaj nije visok, prioritet je i dalje postavljen za visok, obzirom da funkcionalnost ne radi.

5. Automatsko testiranje

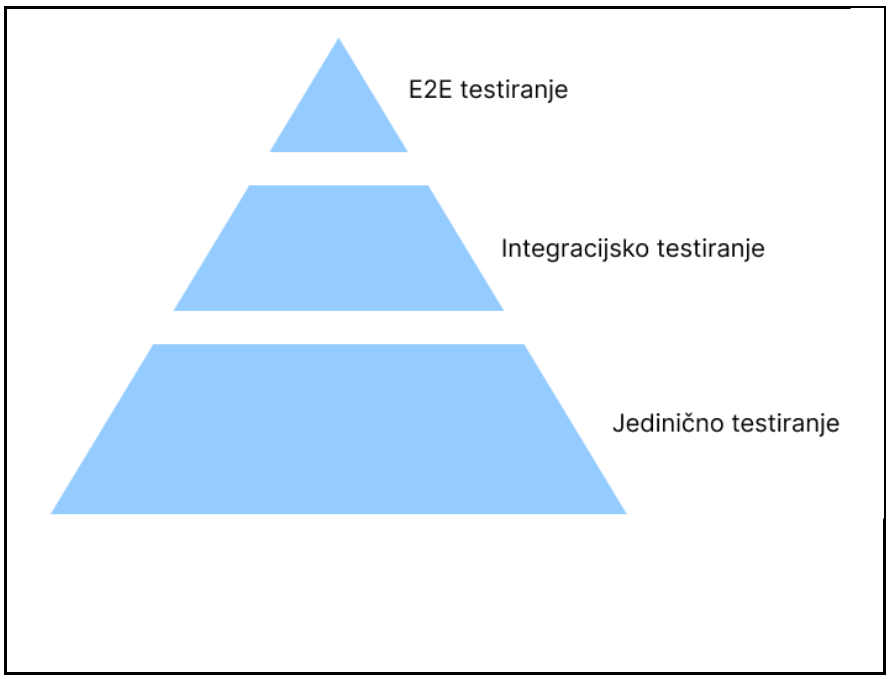
Automatsko testiranje je proces u kojem se testiranje softvera obavlja drugim softverom, odnosno automatski po potrebi. Ovo omogućuje uštedu vremena, ali i osiguravanje višeg stupnja kvalitete, obzirom da se faktor ljudske greške oduzima iz jednadžbe jednom kad su testovi kreirani i potvrđeni. (Anton Hristov, 2022)

Danas često postavljeno pitanje je “isplati li se automatizirati testove”, a odgovor je daleko od jednostavnog. Da bi se na ovo pitanje moglo precizno odgovoriti, u priču je potrebno uvesti puno faktora, od same vrste projekta, do trajanja razvoja, do načina i metodologija razvoja i tako dalje. Za ovaj dio posvećeno je poglavlje u praktičnom dijelu.

Kroz ovo poglavlje, opisane su vrste automatskog testiranja te njihov odnos. Uz to, prikazane su upute i prakse pisanja automatskih testova. Na samom kraju, dani su primjeri alata za kreiranje automatiziranih testova.

5.1. Vrste automatskog testiranja

Vrste automatskog testiranja i njihov odnos je nabolje predstaviti piramidom automatskog testiranja na slici 6.



Slika 6: Prikaz piramide automatskog testiranja (Izvor: Autor prema (Oren Rubin, 2017))

Vrste automatskog testiranja su E2E testiranje, Integracijsko testiranje te jedinično testiranje. Njihov odnos prikazuje piramida sa slike 6. Što je vrsta testiranja niže na piramidi, to je jednostavnija za pisanje, obuhvaća manji dio softvera ali i donosi “najmanje” korisnosti kod automatskog testiranja.

Povijesno, sve vrste testiranja s piramide su se obavljale ručno, a to je bio skup, mukotrpan i proces pun potencijalnih grešaka. (Anton Hristov, 2022)

5.1.1. Jedinično testiranje

Jedinično testiranje je djelomično pokriveno u poglavlju koje opisuje manualno testiranje. Razlog tome je što spada u obije kategorije. Dok se u manualnom dijelu odnosi na pripremu i planiranje samog jediničnog testiranja, automatski dio se odnosi na samo pisanje programskog koda te pokretanje i održavanje testova.

Na samoj piramidi automatskih testova, jedinični testovi su posljednji, odnosno najniže. To znači da je za njihovo pisanje potrebno manje vremena od ostalih, ali i njihov doprinos je mali.

Jedan jedinični test za zadaću ima potvrditi ispravno funkcioniranje jednog djela programskog koda, odnosno jedne metode. Deterministične je prirode i za jedinstveni ulaz podataka očekuje uvijek identičan rezultat.

Iako jednostavni za pisanje, njihova vrijednost je isto tako “mala”, odnosno niža od ostalih u piramidi. Automatizirani test jedne metode u cijelom programskom proizvodu ne donosi puno vrijednosti i osiguranja u pogledu cijelog proizvoda.

Danas, gotovo svi jedinični testovi su automatizirani zato što je pisanje jediničnih testova svrstano u najbolje prakse pisanja programskog koda, te čak postoji i metodologija razvoja koja se temelji na pisanju jediničnih testova, a to je razvoj vođen testovima *(engl. Test driven development).* (Anton Hristov, 2022)

5.1.2. Integracijsko testiranje

Integracijsko testiranje ima za zadaću potvrditi međusobno funkcioniranje nekoliko dijelova programskog proizvoda.

Za integracijske testove, potrebno je spomenuti API *(engl. Application Programming Interface)* te njegovu ulogu u cijelom programskom proizvodu. API se koristi kako bi podatke s poslužitelja *(engl. Server)*, poslao na korisnički dio aplikacije. Pisanje automatskih testova za API programe spada u integracijsko testiranje. Razlog tomu je što se potvrde podatci iz većih

dijelova koda, odnosno za jednu informaciju koju potvrđujemo, zaduženo je više dijelova programskog koda.

Na piramidi automatskih testova, integracijski testovi se nalaze u sredini, što znači da je za njihovo pisanje potrebno više vremena, ali i da je isplativost veća, te se jednim integracijskim testom pokrije veći dio programskog koda.

5.1.3. E2E testiranje

E2E *(engl. End to End)* testiranje je automatizacija simulacija procesa krajnjih korisnika. Za zadaću ima rekreirati potencijalne akcije koje korisnik može raditi na programskom proizvodu.

Dok se ostale vrste testova na piramidi pišu tako da se fokusiraju na pozadinski dio koda *(engl. backend)*, u daljnjem tekstu backend, E2E testiranje se fokusira na prednji dio koda *(engl. frontend)*, u daljnjem tekstu frontend. To znači da se ne provjerava funkcioniranje aplikacijskog i serverskog dijela aplikacije, nego korisničkog.

To se postiže tako da se pišu testovi koji simuliraju korisnikove akcije na samoj aplikaciji. Npr. popunjavanje polja, klikovi na gumbe, odabir opcija, pomicanje elemenata i slično.

Na piramidi automatskih testova su prvi, odnosno za pisanje jednog E2E testa je potrebno puno vremena, ali je njegov utjecaj, odnosno dodana vrijednost velika. Jednim E2E testom, pokrije se nekoliko integracijskih testova i nekoliko desetaka, ako ne i stotina jediničnih testova.

Iz razloga navedenih gore, u testnoj i QA ulozi, ova vrsta testiranja je najzastupljenija i tema je praktičnog dijela ovog rada.

5.2. Pisanje automatskih testova

Način pisanja automatskih testova je jedan od najbitnijih faktora kada se postavlja pitanje “Je li automatizacija testova isplativa”. Ovisno o načinu na koji će testovi biti napisani, te hoće li oni biti održivi, o tome će ovisiti isplativost cijelog testnog projekta.

Kako bi se ispravno pisali automatski testovi, postoji 10 pravila na koje treba obratiti pozornost. U nastavku teksta, svako od pravila je pokriveno primjerom iz prakse.

5.2.1. Prioritizacija

Kao što je već spomenuto, većina aplikacija se sastoji od stotina, tisuća ili čak desetaka tisuća scenarija. Pokrivanje svakog od ovih scenarija automatskim testom je skup proces i rijetko isplativ, pogotovo u slučajevima kada su testovi vizualne prirode i softverom je teško pokriti funkcionalnost. (Oren Rubin, 2017)

Svaka aplikacija ima set funkcionalnosti koje se koriste češće i one koje se koriste rjeđe. Uz to, set funkcionalnosti na kojima se češće radi i one na kojima se rjeđe radi. Za prioritizaciju automatizacije cilj je uzeti funkcionalnosti koje se koriste češće i na kojima se radi češće. To će omogućiti da se ne troši vrijeme na ručno testiranje nakon svake promjene koje programeri naprave.

Za primjer uzmimo funkcionalnost prijave. Ova funkcionalnost je sastavni i glavni dio svake aplikacije. Ako aplikacija ima prijavu, gotovo uvijek je prijava nužan korak za korisnika. Iz tog razloga, scenarije vezane za prijavu je vrlo isplativo automatizirati. Za usporedbu uzmimo funkcionalnost pretplate na novosti. Ova funkcionalnost je svakako bitna za poslovanje, ali nikako za funkcioniranje aplikacije. Iz tog razloga, funkcionalnosti poput ove mogu imati niži prioritet.

5.2.2. Smanji, Recikliraj, Ponovno iskoristi

Ovo pravilo govori da je bitno da su sve komponente razlomljene u cjeline koje imaju jedinstvenu zadaću. To znači da svaki test ili skupina testova mora izvršavati jedinstvenu zadaću i pokrivati jedinstvenu funkcionalnost aplikacije. (Oren Rubin, 2017)

Kao i u svakom programskom razvoju, imenovanje elemenata koda je izrazito bitan faktor. Dobre prakse imenovanja omogućit će da su testovi samo objašnjivi, jednostavni za razumijevanje i laki za održavanje.

Još jedno pravilo iz općenitih pravila razvoja je ponovno korištenje komponenti i dijelova koda. Kada god postoji potreba za kopiranjem dijelova koda, to znači da se taj komad koda može odvojiti i spremiti kao jedinstvena, ponovno iskoristiva cjelina. Tako ćemo spriječiti gomilanje jedno te istog koda i olakšati ponovno korištenje. (Oren Rubin, 2017)

Na kodu ispod prikazan je loš primjer onoga što je napisano u odlomku iznad. Ova funkcija kada se pokrene, ispisuje tekst: Luka Majstorović 1998

Imenovanje varijabli nam ne daje nikakve informacije o samoj svrsi varijabli, što već na ovom primjeru, gdje se radi o 9 linija koda, stvara probleme pri shvaćanju svrhe ove funkcije.

Uz to, način strukturiranja funkcije otežava ponovno korištenje. Ukoliko se podatci žele ispisati podatke drugim redoslijedom, potrebno je raditi identičnu funkciju, koja će u liniji 8 imati drugačiji redoslijed ispisa riječi. Time je otežano održavanje i kreirano je puno redundantnog koda.

```
ispis()
function ispis()
{
    var a = 'Luka'
    var b = 'Majstorovic'
    var c = '1998'

    console.log(a + ' ' + b + ' ' + c)
}
```

Na kodu ispod je primjer dobre prakse imenovanja varijabli i metoda te ponovnog iskorištavanja koda.

U ovom primjeru, imena varijabli imaju konkretno značenje, te je iz njih lako prepoznati što podatak predstavlja. Ime, prezime te godina rođenja su korištene varijable u aplikaciji. Za ispis samih podataka modificirana je funkcija iz primjera iz prethodnog koda. Ova funkcija ima konkretnije ime te ispisuje riječi po redu: prva riječ, druga riječ, treća riječ. Na ovaj način se mogu lako ispisati podatci o korisniku drugim redoslijedom, što je vidljivo u linijama koda 5 - 8. Linije ispisuju:

- Linija 5: Luka Majstorović 1998
- Linija 6: Majstorović Luka 1998
- Linija 7: 1998 Majstorović Luka
- Linija 8: 1998 Luka Majstorović

Ako je potrebno promijeniti način rada funkcije, npr. da se umjesto razmaka ispisuje `'-'`, potrebno je samo urediti liniju 12 te promijeniti razmake u `'-'`.

Kako bi se ispisale sve moguće permutacije riječi, korištenjem ove logike, na prvom primjeru koda je potrebno 60 linija koda, dok je na drugom potrebno 13 linija koda. Iako same

linije koda ne označavaju dobar ili loš kod, ovdje su odličan pokazatelj optimiziranosti koda, odnosno testa.

```
var ime = 'Luka'
var prezime = 'Majstorovic'
var godina_rodenja = '1998'

ispis_podataka_o_korisniku(ime, prezime, godina_rodenja)
ispis_podataka_o_korisniku(prezime, ime, godina_rodenja)
ispis_podataka_o_korisniku(godina_rodenja, prezime, ime)
ispis_podataka_o_korisniku(godina_rodenja, ime, prezime)

function ispis_podataka_o_korisniku(prva_rijec, druga_rijec, treca_rijec)
{
    console.log(prva_rijec + ' ' + druga_rijec + ' ' + treca_rijec)
}
```

5.2.3. Kreiranje strukturiranih testova s jedinstvenom zadaćom

Test koji ima jedinstvenu zadaću potvrditi samo jednu stvar u jednom trenu. Razlog kreiranja ovakvih testova je lakša održivost te brzina izvođenja. Ako se pokuša testirati nekoliko stavki u jednom trenu, mogućnost je da prva od njih ne radi ispravno te test “pukne” i proces završi. Ovime se ne dobije nikakva informacija o drugim komponentama koje potencijalno rade. (Oren Rubin, 2017)

Svaki test bi se trebao sastojati od četiri glavna dijela koja su opisana u nadolazećim pod poglavljima.

5.2.3.1. Postavljanje (*engl. Setup)*

Svrha ovog dijela testa je da aplikaciju dovede do željenog stanja, odnosno stanja neposredno prije pokretanja testa. Ovaj korak je neophodan kako bi se kreirali pouzdani i održivi testovi te spriječilo ponavljanje mnogih koraka testova kako bi aplikacija bila u određenom stanju. (Oren Rubin, 2017)

Kao primjer pripreme testa, navedeni su sljedeći koraci:

- Otvori aplikaciju
- Navigiraj se na stranicu za prijavu

Ovi koraci su stavljeni u dio postavljanja zato što se u određenom testu ne testira njihova funkcionalnost, te je ona provjerena u nekom od prethodnih testova. Oni služe kako bi postavili aplikaciju u željeno stanje.

5.2.3.2. Akcije (*engl. Actions*)

Akcije su korisnikovo ponašanje koje je potrebno simulirati. Ovdje se stavljaju koraci i funkcije koje će simulirati korisnikove akcije nad stranicom. (Oren Rubin, 2017)

Kao primjer akcija testa, navedeni su sljedeći koraci:

- Generiraj nasumično ime od 5 slova
- Generiraj nasumičnu lozinku od 5 slova
- Unesi ime u odgovarajuće polje
- Unesi lozinku u odgovarajuće polje
- Pritisni gumb za prijavu

Ovime je simuliran korisnikov proces prijave u sustav. Razlog odabira nasumičnog imena te lozinke objašnjen je u nadolazećem poglavlju validacije.

5.2.3.3. Validacija (*engl. Validation)*

Validacije su koraci koji provjeravaju je li ponašanje i stanje aplikacije ispravno. Same validacije ne moraju nužno provjeravati stanje korisničke strane aplikacije, nego mogu provjeriti npr. stanje baze podataka. (Oren Rubin, 2017)

Kao primjer validacija testa, navedeni su sljedeći koraci:

- Prikazana je poruka da korisnik s navedenim imenom ne postoji
- Ponuđena je opcija za kreiranje računa
- Ponuđena je opcija za oporavak računa
- Potvrdi da u bazi ne postoji korisnik s navedenim imenom

Kao što je prikazano u primjerima validacije iznad, 4. korak provjerava stanje u bazi podataka. Tako se potvrđuje da korisnik uistinu ne postoji te da greška nije u testu nego je generirano nasumično ime nepostojano.

5.2.3.4. Destrukcije (*engl. Tear down)*

Destrukcija se koristi kako bi aplikaciju vratila u stanje spremno za sljedeći test. Dok se ostali koraci uglavnom fokusiraju na korisnički dio aplikacije, ovaj korak se fokusira na to da bazu podataka pripremi za sljedeći test. (Oren Rubin, 2017)

Razlog korištenja ovog koraka je utjecanje prethodnog testa na naredni test, što se ne slaže s konceptom da svaki test ima isti izlaz na isti ulaz. Primjer problema je registracija. Ako postoji test koji registrira email luka.majstorovic@foi.hr i uspješno je proveden, taj isti test će pasti ako se opet pokrene zato što će taj email već postojati.

5.2.4. Početno stanje svakog testa mora biti konzistentno

S obzirom na to da automatski testovi uvijek ponavljaju predefinirani skup koraka, oni uvijek moraju kretati od istog inicijalnog stanja. Najčešći problemi kod održavanja automatskih testova je osiguravanje integriteta početnog stanja. Zato što automatski testovi ovise o početnom stanju, ako ono nije konzistentno, niti rezultati testa neće biti. Inicijalno stanje je najčešće dobiveno iz prethodnih korisnikovih akcija. (Oren Rubin, 2017)

Primjeri problema:

- Dodavanje stavke u listu kada lista već koristi stavku (iz prethodnog pokretanja testova)
- Ponašanje aplikacije koje je različito za nove korisnike i postojeće korisnike
- Automatsko preusmjeravanje ako je korisnik prijavljen

Potencijalna rješenja:

- Kreiranje novog korisnika kod svakog pokretanja testova
- Korištenje zasebnog razvojnog okruženja za pokretanje testova
- Prije svakog pokretanja testova, postaviti stanje baze na jednako
- Eliminacija međusobne ovisnosti testova. Npr. Test 1 ne bi trebao utjecati na Test 2

5.2.5. Složeni testovi trebaju biti sastavljeni od jednostavnih testova

Optimizacija redoslijeda pokretanja testova i same veličine testova često je spominjana u ovih 10 pravila. Pravilo broj 5 govori kako svi kompleksniji testovi trebaju biti sastavljeni od više jednostavnih testova. (Oren Rubin, 2017)

To znači da se jednostavni testovi koriste kao gradivni elementi za kompliciranije testove i da se jednostavni testovi trebaju pokretati i potvrditi prije kompleksnijih.

Na kodu ispod prikazan je test koji se sastoji od 8 manjih testova. Funkcije na linijama 1-8 mogu se koristiti kao zasebni mali testovi koji potvrđuju svaku od navedenih komponenti.

```
prijavi_korisnika(email, lozinka)
potvrdi_kolacice()
otvori_popis_artikala()
odaberi_artikl_krušku()
odaberi_artikl_jabuku()
nastavi_na_kupnju()
unesi_podatke_za_kupnju(kartice.ispravna_kartica)
potvrdi_kupnju()
```

Npr. `prijavi_korisnika()` je funkcija koja može funkcionirati sama za sebe i ispravno je kreirati test koji će potvrditi isključivo tu funkcionalnost.

Na ovaj način, ako se promjeni logika prijave korisnika, ta logika će se morati urediti samo na jednom mjestu (u metodi `prijavi_korisnika()`), a ne svugdje gdje je prijava napisana.

5.2.6. Validacija na ključnim točkama

Validacije testova se uglavnom koriste na kraju testa kako bi dobili informaciju je li test prošao ili ne. Ovo pravilo govori kako je dobra praksa dodati ih i na bitnim točkama u radu samih testova. Tako, ako je test kompleksnije prirode ili postoje razna uvjetovanja, možemo spriječiti izvršavanje testova do kraja ako su već na nekom od prethodnih koraka bili neispravni. Ovime se optimizira brzina rada i daje detaljnija povratna informacija. (Oren Rubin, 2017)

5.2.7. Bez korištenja spavanja (*engl. Sleep)* za bolje performanse

Korištenje funkcije za spavanje je velika zamka u pisanju automatskih testova. Iako ima pozitivne strane i rješava mnoge probleme, njeno korištenje uvelike usporava rad testova. Funkcija za spavanje forsira izvođenje koda da stane, bez obzira na stanje u kojem taj kod je. To može biti korisno kada nakon određene akcije čekamo da se stranica osvježi ili da se određeni element učita. Problem s čekanjem je što ne znamo koliko će to učitavanje trajati, pa moramo koristiti gornju granicu uvijek, a i to nam ne garantira da se kod sporije veze element neće učitati. (Oren Rubin, 2017)

```
describe('Primjer testa s cekanjem i bez', () => {
    it('S cekanjem', () => {
      cy.visit('https://foi.hr');
      cy.wait(3000);
      cy.contains('Fakultet organizacije i
informatike').should('be.visible');
    });

    it('Bez cekanja', () => {
      cy.visit('https://foi.hr');
      cy.contains('Fakultet organizacije i
      informatike').should('be.visible').timeout(3000);
      });
  });
```

Većina alata ima implementirana pametnija rješenja za čekanje određenih elemenata da se pojave na stranici. Na kodu iznad prikazan je primjer programskog jezika JavaScript. Na linijama 2-6 se nalazi test koji koristi funkciju za čekanje, u ovom slučaju `cy.wait()`, dok se na linijama 8-10 nalazi test koji koristi praktičnije rješenje. Kao što je već spomenuto, funkcija `cy.wait(3000)` će zaustaviti izvođenje koda na 3000ms, odnosno 3 sekunde, neovisno o stanju u kojem je program. To znači da i ako se element učita za jednu sekundu, aplikacija će čekati 3 sekunde.

Dok rješenje ispod toga koristi `timeout(3000)` funkciju koja se automatski izvodi tijekom izvođenja `should('be.visible')` funkcije. Odnosno, aplikacija konstantno provjerava je li se tekst 'Fakultet organizacije i informatike') učitao na stranici, te u trenu kada je, korak prolazi dalje. Ako se element ne učita u 3 sekunde, korak će se smatrati neispravnim.

5.2.8. Minimalno dvije razine apstrakcije

Za pisanje automatizacije koriste se razvojni okviri (*engl. framework*), te oni imaju svoju implementaciju logike i potrebnih akcija vezanih za preglednik kako bi simulirali akcije korisnika.

Ovo pravilo govori kako je potrebno odvojiti samu logiku programskog jezika od logike razvojnog okvira. Jedan od glavnih razloga je jednostavnost održavanja te jednostavnost shvaćanja i modificiranja logike samim programerima, ako dođe do grešaka u testu. (Oren Rubin, 2017)

Na kodu iznad je prikazano odvajanje unosa podataka korisnika te izračuna dobi korisnika. Izračun dobi koristi isključivo funkcije *JavaScript*-a, dok unos podataka koristi funkcije *Cypress*-a, koji je razvojni okvir za *JavaScript*.

```
unesi_podatke(dob)
{
    cy.get('#ime').sendKeys('Luka')
    cy.get('#prezime').sendKeys('Majstorovic')
    cy.get('#godina_rodenja').sendKeys('1998')
    cy.get("#dob").sendKeys(dob)
}

izracunaj_dob(god_rod)
{
    const trenutniDatum = new Date();
    const trenutnaGodina = currentDate.getFullYear();
    return trenutnaGodina - god_rod
}
```

5.2.9. Smanjiti količinu uvjetovanja

Stanje testa može ovisiti o mnogo faktora i ponekad je potrebno provjeriti trenutno stanje kako bi se moglo nastaviti s testom u ispravnom smjeru. Velika količina uvjetovanja stvara probleme kod održivosti testova i potencijalnih promjena ponašanja aplikacija. U tim slučajevima se cijele uvjetne strukture moraju mijenjati. Da bi se izbjeglo korištenje uvjetnih struktura, potrebno je: (Oren Rubin, 2017)

- Pokretati testove od predefiniranih stanja
- Ugasiti potencijalne nasumične skočne prozore
- Ako postoji A/B testiranje, isključiti ga

5.2.10. Pisanje neovisnih i izoliranih testova

Kao što je spominjano, svi testovi moraju biti neovisni. Ovo pravilo govori o kreiranju testova tako da su izolirani te da se mogu paralelno izvoditi. Ovo omogućava veliku uštedu vremena te smanjenje troškova resursa kod ponavljajućih testiranja. (Oren Rubin, 2017)

5.3. Alati za izradu automatskih testova

Trenutno na tržištu postoji velika količina alata za automatizaciju. Riječ “alat”, u ovom kontekstu obuhvaća cijele aplikacije koje su specificirane za generiranje automatskih testova i razvojne okvire, koji pomažu u pisanju automatskih testova.

Ove dvije kategorije popularno se zovu i nisko kodne (*engl. low code*) te bez koda *(engl. no code*). U daljnjem tekstu opisane su dvije kategorije te potkrijepljene s nekoliko primjera takvih alata. (Lucia Cavero-Baptista, bez dat.)

5.3.1. Alati bez pisanja koda

Alati bez pisanja koda omogućavaju korisnicima da kreiraju testove bez da napišu i jednu liniju koda. To rade tako da snimaju korisnikove akcije na stranici te spremaju akcije u funkcije, odnosno generiraju kod za korisnika. U posljednje vrijeme sve više se koristi umjetna inteligencija kako bi se poboljšale performanse ovakvih platformi. Neke od primjera alata bez pisanja koda su: (Lucia Cavero-Baptista, bez dat.)

- Testim.io
- Leapwork
- Katalon Studio

5.3.2. Alati s niskom razinom koda

Alati s niskom razinom koda su zapravo alati koji koriste isključivo pisanje koda kako bi realizirali automatizaciju, ali su naziv dobili od činjenice da su oni zapravo razvojni okviri, koji u sebi sadrže blokove koda od kojih se zatim rade testovi. Neki od primjera alata bez pisanja koda su: (Lucia Cavero-Baptista, bez dat.)

- Cypress
- Selenium
- Laravel Dusk

Najveći fokus ovog rada je na alatu *Cypress* te je njegovo funkcioniranje detaljno opisano u praktičnom radu.

6. Praktični rad

Praktični dio ovog rada se sastoji od dva dijela, a to su izrada i automatizacija testiranja te analiza i usporedba istih.

S ciljem da se prikaže važnost obije vrste testiranja, detaljno će biti pokriveni i opisani postupci kreiranja istih praćenjem teorije i opisanih praksi u teorijskom dijelu ovog rada.

Prvotni cilj je opisati korištenu aplikaciju. Zatim opisati korištene tehnologije i započeti s pripremanjem testnih scenarija. Nakon toga odrađeno je manualno testiranje aplikacije nakon kojeg slijedi priprema i automatizacija testiranja. Na samom kraju, rezultati su uspoređeni i dobiven je krajnji rezultat i odgovor na početno pitanje ovog rada.

6.1. Uvod u praktični dio

Cilj praktičnog dijela ovog rada je prikazati potpun proces kreiranja i raspisivanja testnih scenarija, izvođenja tih istih scenarija a zatim kreiranja automatskih testova, pokretanja i tumačenja izvještaja.

Nakon kreiranih i izvršavanja automatskih testova, cilj je usporediti ulogu i doprinos manualnog i automatskog testiranja te njihove prednosti i mane.

U uvodu u sam praktični dio pokrivena je odabrana aplikacija te metodika i pristup cijelom procesu.

6.1.1. Odabrana aplikacija

Odabrana je aplikacija *Swag Labs*, proizvođača *Saucedemo*. Odabrana aplikacija omogućuje da se u četrdesetak testnih scenarija demonstrira proces planiranja, kreiranja te automatizacije testiranja na određenom programskom proizvodu.

Bitno je napomenuti da kod odabira aplikacije za automatizaciju treba biti izrazito oprezan. Razlog tomu je što gotovo svaka web aplikacija ima uvijete i pravila korištenja, među kojima su i pravila koja zabranjuju bilo kakvo neovlašteno korištenje robota i preuzimanja podataka sa stranice. U tu kategoriju spadaju i automatski testovi. Automatski testovi korisničkih sučelja rade jednako kao i roboti za preuzimanje podataka te su iz tog razloga zabranjeni za korištenje. Stranica *Saucedemo* je eksplicitno izrađena u svrhu demonstriranja i vježbanja automatizacije te je dozvoljeno koristiti bilo kakvu vrstu automatizacije nad njom.

U praksi, stvari funkcioniraju tako da inženjeri razvijaju automatske testove na aplikacijama svoje tvrtke ili su dobili izričitu dozvolu da razvijaju testove za određenu aplikaciju.

Uz to, aplikacije na kojima se izvršavaju automatski testovi se najčešće pokreću na zasebnim okolinama. Tako testovi ne utječu na ponašanje produkcijske okoline i mogu se izvršavati kada god je potrebno.

Zahvaljujući opcijama i različitim korisnicima koje pruža stranica, testirat će se s 3 vrste korisnika:

- Ispravan korisnik - Očekivano uspješno izvođenje svih testova
- Korisnik s problemima u radu - Očekivane greške i padanje određenih testova
- Korisnik koji ima probleme s performansama - Očekivano uspješno izvođenje svih testova uz razliku vremena izvođenja

Saucedemo je stranica koja je svojom glavnom funkcionalnosti online web shop. Funkcionalnosti koje sadrži su:

- Prijava u sustav
- Navigacija
- Prikaz liste te prikaz pojedinih proizvoda
- Proces kupovine proizvoda

6.1.2. Pristup i način rada

Stranica *Saucelabs* je po svojoj funkciji web shop te je u tu svrhu potrebno prilagoditi pristup testiranju. Činjenica da se radi o web shop-u govori da je veliki fokus stavljen na funkcionalnosti kupovine i svih popratnih radnji.

Kako bi automatizacija imala svoju vrijednost i isplativost, treba joj se pristupiti sistematično i organizirano. U tu svrhu, kreirani su testni scenariji s detaljnim opisom i odrađeno je detaljno manualno testiranje kako bi se stvorile metrike poput: količine testnih scenarija, vrijeme potrebno za ručno testiranje, količina grešaka, ključni korisnički scenariji i slično.

Nakon toga slijedi priprema i prioritizacija testova o kojoj je više napisano u jednom od nadolazećih poglavlja.

6.2. Javascript i Cypress

Javascript i *Cypress* korišteni su u svrhu izrade praktičnog dijela. Prvenstveno je odabran *Cypress*, kao moderno rješenje i glavni suparnik *Selenium* rješenju za automatizaciju web aplikacija. *Javascript* je odabran jezik pisanje iz razloga što *Cypress* kao razvojni okvir podržava isključivo njega. U daljnjem tekstu detaljnije su opisana svojstva i načini funkcioniranja ova dva alata.

6.2.1. Javascript

Javascript je po svojoj definiciji interpretni, odnosno jezik koji se kompajlira u vremenu izvođenja. Najpoznatiji je kao skriptni jezik za web stranice, ali koristi se i u druge svrhe.

Mnoge okoline van preglednika ga također koriste te je implementiran u razne jezike i razvojne okvire. Najpoznatiji od njih je Node.js koji se koristi za izradu serverskih aplikacija.

Uz njega poznati su i Apache *CouchDB* te *Adobe ACrobat*. Javascript je dinamički jezik koji podržava objektno orjentirano, imeprativno te funkcijsko programiranje.

Javascript nema nikakve veze s programskim jezikom Java. Radi se o dva potpuno različita programska jezika koja se razlikuju po načinu funkcioniranja, načinu pisanja te po samoj sintaksi pisanja koda. (MDM contributors & Mozilla, 2023)

Kod pisan u *Javascript*-u je lagan i jednostavan za pisanje. To znači da samo izvođenje koda je brzo i ne zahtjeva puno resursa. To je još jedan razlog zbog kojeg je *Javascript* odličan izbor za pisanje automatskih testova. (MDM contributors & Mozilla, 2023)

6.2.2. Cypress

Cypress je alat otvorenog koda (*engl. Open source*) koji se fokusira na testiranje krajnjih korisničkih akcija (*engl. End to End*) nad web aplikacijama. Kreiran je s ciljem da olakša developerima pisanje i pokretanje automatiziranih testova za njihove web aplikacije. (Cypress.io, 2023b)

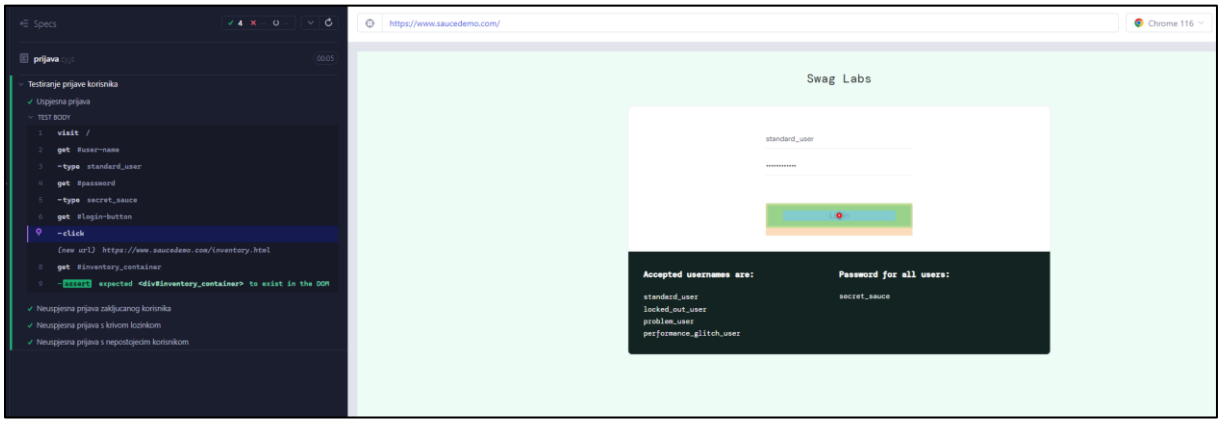
Osigurava da aplikacije rade ispravno i efikasno na različitim preglednicima i u različitim okruženjima.

Iako relativno nov alat, sadrži mnoge funkcionalnosti koje stariji i „iskusniji“ alati ne sadrže, a neke od tih funkcionalnosti uz primjere su opisane u nadolazećem tekstu. (Cypress.io, 2023a)

6.2.2.1. Testiranje u stvarnim preglednicima

Za razliku od mnogih aktualnih alata za testiranje, *Cypress* kreira stvarnu instancu preglednika te testiranje izvršava u toj instanci. Mnogi alati koriste bezglavi *(engl. Headless)* način rada, što znači da se korisniku ne prikazuje simulacija akcija nego krajnji rezultat. U tim vrstama testiranja, DOM *(engl. Document Object Model)* je generiran i akcije su obavljene bez da se to korisniku prikaže. *Cypress* odlazi korak dalje te programerima omogućuje da u stvarnom vremenu prate stanje svojih testova. (Cypress.io, 2023a)

Na slici 7 je vidljivo sučelje koje pruža *Cypress.* S lijeve strane su prikazane sve akcije i pozivi koje aplikacija radi, a s desne strane simulacija tih akcija i procesa u realnom vremenu. Ovime se omogućuje da u slučaju greške u testu, developer može reagirati i pogledati s lijeve strane gdje je došlo do greške. Skraćuje se vrijeme rješavanja problema i povećava sama efikasnost testnog procesa.

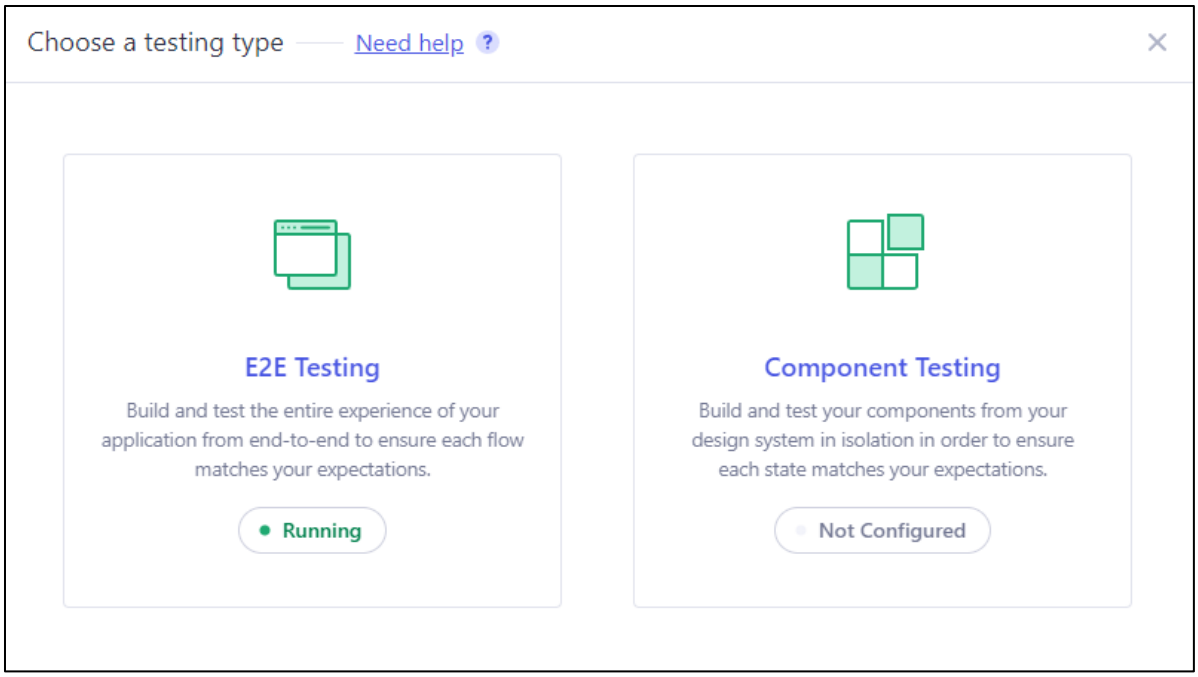


Slika 7: Korisničko sučelje i praćenje izvođenja testova (Izvor: Autor prema (Cypress.io, 2023b))

6.2.2.2. E2E i Komponentno testiranje

Cypress omogućuje svojim korisnicima jednostavan i održiv način kreiranja i pisanja E2E testova te je njegov glavni fokus testiranje aplikacije iz korisničke perspektive te verifikacija da aplikacija u potpunosti radi kao cjelina.

Komponentno testiranje nije pokrivenu u praktičnom dijelu ovog rada iz razloga što se fokusira na integracijsku razinu te jediničnu razinu testiranja. Ovim testiranjem se testira funkcioniranje komponenti (npr. Forma, gumb itd.) kao zasebnih cjelina, radije nego kao dijela kompletnog proizvoda. Bitno je napomenuti, iako *Cypress* podržava komponentno testiranje, prvotno je kreiran i zamišljen kao alat za E2E testiranje. Slika 8 prikazuje odabir vrste testiranja u sučelju. (Cypress.io, 2023a)



Slika 8: Opcija odabira vrste testiranja u *Cypress-u* (Izvor: Autor prema (Cypress.io, 2023b))

6.2.2.3. Automatsko čekanje elemenata

Glavni problem aktualnih alata za automatizaciju korisničkih sučelja je brzina odgovora elemenata na stranici. U određenim slučajevima elementima treba više vremena nego je to standardno da se generiraju na stranici te to može dovesti do problema kod izvođenja testova.

Uzmimo za primjer test u primjeru ispod:

```
it('Test s akcijama', () => {
    cy.get('#login').click()
    cy.get('#pretraga').type('proizvod')
});
```

U prikazanom isječku programskog koda iznad, izvode se dvije uzastopne akcije na stranici. Prva akcija prijavljuje korisnika u sustav, a druga upisuje tekst u polje pretrage na sljedećoj stranici. U ovom slučaju, koristeći većinu drugih alata za testiranje, ovaj test ne bi prošao te bi pao nakon prijave korisnika. Razlog tomu je brzina generiranja, odnosno prikazivanja sljedeće stranice. Za riješit ovaj problem postoje dvije mogućnosti:

1. *Korištenjem* `sleep()` funkcije
2. Korištenjem `wait_for_element()` funkcije

Rješenje broj 1 je loša praksa, te se kosi s pravilima izrade automatskih testova pa ćemo ga automatski odbaciti. Rješenje broj 2 je ispravnije rješenje, ali bi se u svakom od ovih slučajeva trebala pisati navedena naredba. Uz to, naredba mora biti precizno nadodana na element koji

čekamo. Ovime se otvara prostor za grešku i potencijalne nekonzistentnosti u samim testovima.

Cypress ovom problemu pristupa na način da svako učitavanje stranice ili elementa je automatski povezano s naredbom `wait_for_element()` te time olakšavao posao i onemogućuje neželjeno pucanje testova. U svojoj osnovnoj postavki, `wait_for_element()` čeka 60 sekundi za element da se pojavi na stranici. U slučajevima gdje želimo testirati performanse stranice, ova količina vremena je prevelika te ju je potrebno smanjiti. To se može napraviti korištenjem funkcije `timeout()` te dodjeljivanja vrijednosti maksimalnog vremena čekanja u milisekundama. Primjer ovoga je prikazan u isječku koda ispod:

```
 it('Test s akcijama', () => {
   cy.get('#login').click()
   cy.get('#pretraga', {timeout:10000}).type('proizvod')
});
```

6.2.2.4. Podrška kontinuirane integracije

Kako bi i sama automatizacija bila optimiziranija te uistinu automatizirana, potrebno je omogućiti automatsko izvođenje automatiziranih testova. To se odnosi na činjenicu da ne zahtijevaju ljudski aspekt kako bi bili pokrenuti.

U tu svrhu, koristi se koncept kontinuirane integracije (*engl. Continous integration)* te koncept kontinuiranog dostavljanja koda (*engl. Continous deployment)*, odnosno kratica i termin korišten dalje u tekstu, CI/CD.

CI/CD je dio *devops-a* te omogućuje programerima da imaju automatizirane procese vezane za razvoj programskog proizvoda. Alati koji se često koriste u tu svrhu su: *Docker, Gitlab, Github, Kubernetes, Jenkins* itd. Ovi alati imaju za svrhu automatizaciju procesa poput predavanja koda *(engl. Code push),* spajanja koda (*engl. Code merge)*, verzioniranje koda (*engl. Versioning)* itd.

Najčešće se automatizirani testovi postave na način da se sami pokrenu kada se dogodi određena radnja nad kodom, npr. *Code merge*. U toj situaciji spojene su dvije verzije koda i potrebno je testirati aplikaciju. Iz tog razloga pokrenu se testovi koji se odrade u pozadini i vrate izvještaj svog pokretanja. Više o izvještajima u nadolazećim poglavljima.

6.2.2.5. Ostalo

Ovo su samo neke od funkcionalnosti koje *Cypress* pruža. Ostatak funkcionalnosti je direktno pokriven u praktičnom radu te će tamo biti i objašnjen.

6.3. Pisanje i izvođenje manualnog testiranja

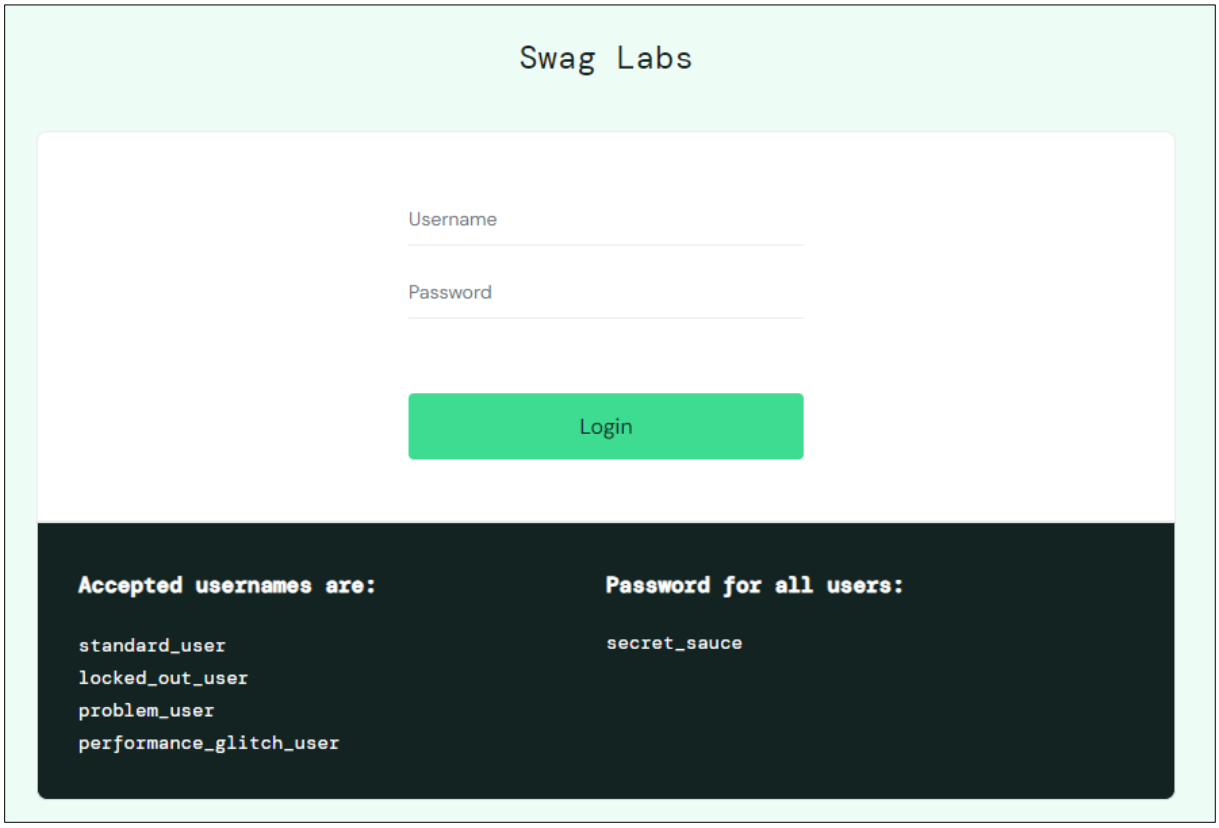
Kako bi se obavila priprema za automatsko testiranje, potrebno je odraditi detaljno i planirano manualno testiranje. Ovo poglavlje prikazuje ideju, proces te izvršavanje kreiranja i obavljanja manualnog testiranja.

6.3.1. Pisanje testnih scenarija

Za pisanje testnih scenarija korištena je struktura objašnjena u teorijskom dijelu ovog rada. Nastavak ovog poglavlja prikazati će svaki od dijelova stranice te njegove popratne scenarije.

6.3.1.1. Prijava u sustav

Prijava u sustav se sastoji od 2 polja za unos, korisničko ime te lozinka. *Swag Labs* nudi opciju nekoliko vrsta korisnika s različitim rezultatima prijave, te je potrebno kreirati testne scenarije za svakog od tih korisnika. Prijava stranice prikazana je na slici 9.



Slika 9: Stranica prijave u aplikaciju (Izvor:(Saucedemo, 2023))

Za potrebe testiranja ove stranice kreirano je četiri testna scenarija, numeriranih oznakama TC-01 – TC-04. Scenariji su vidljivi na slici 10.

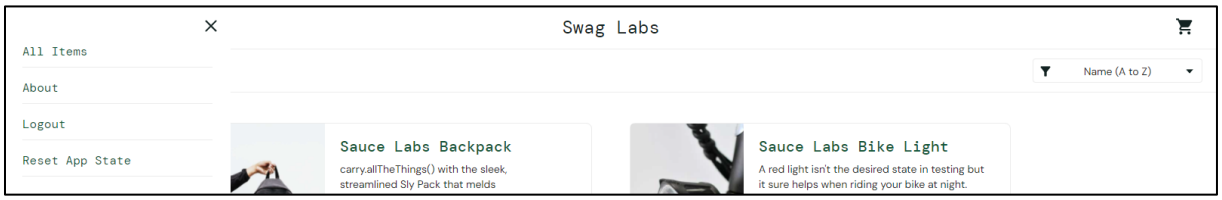
Identifikator Scenarija	Naziv Scenarija	Preduvjeti	Testni koraci	Očekivani rezultat	Rezultat testa	Opis greške	Automatizacija
			Prijava				
TC-01	Prijava s ispravnim podatcima je moguća	Kreiran korisnički račun	1. Unesi ispravno korisničko ime 2. Unesi ispravnu korisničku lozinku 3. Pritisni "Login" gumb	Korisnik je uspješno prijavljen i prikazan je popis proizvoda	Prošao	-	Da
TC-02	Zaključani korisnik se ne može prijaviti	Kreiran korisnički račun koji je zaključan	1. Unesi korisničko ime zaključanog korisnika 2. Unesi ispravnu korisničku lozinku 3. Pritisni "Login" gumb	Korisnik nije prijavljen i prikazana je poruka koja informira o statusu korisnika (zaključan)	Prošao	-	Da
TC-03	Prijava s neispravnom lozinkom nije moguća	Kreiran korisnički račun	1. Unesi ispravno korisničko ime 2. Unesi neispravnu korisničku lozinku 3. Pritisni "Login" gumb	Korisnik nije prijavljen i prikazana je poruka koja informira o statusu korisnika (neispravna kombinacija imena i lozinke)	Prošao	-	Da
TC-04	Prijava s ne postojećim korisničkim imenom	Ne postoji kreiran račun s unesenim imenom	1. Unesi neispravno korisničko ime 2. Unesi ispravnu korisničku lozinku 3. Pritisni "Login" gumb	Korisnik nije prijavljen i prikazana je poruka koja informira o statusu korisnika (neispravna kombinacija imena i lozinke)	Prošao	-	Da

Slika 10: Testni scenariji Prijave (Izvor: Autor)

6.3.1.2. Navigacija

Navigacija se sastoji od dva dijela. Prvi dio je klasična navigacijska traka koja u sebi sastoji dodatan *Hamburger* meni te ikonu koja vodi na košaricu korisnika. Slika 11 i slika 12 prikazuju strukturu navigacije.

Slika 11: Osnovna navigacijska traka (Izvor: (Saucedemo, 2023))



Slika 12: Proširen *Hamburger* izbornik (Izvor: (Saucedemo, 2023))

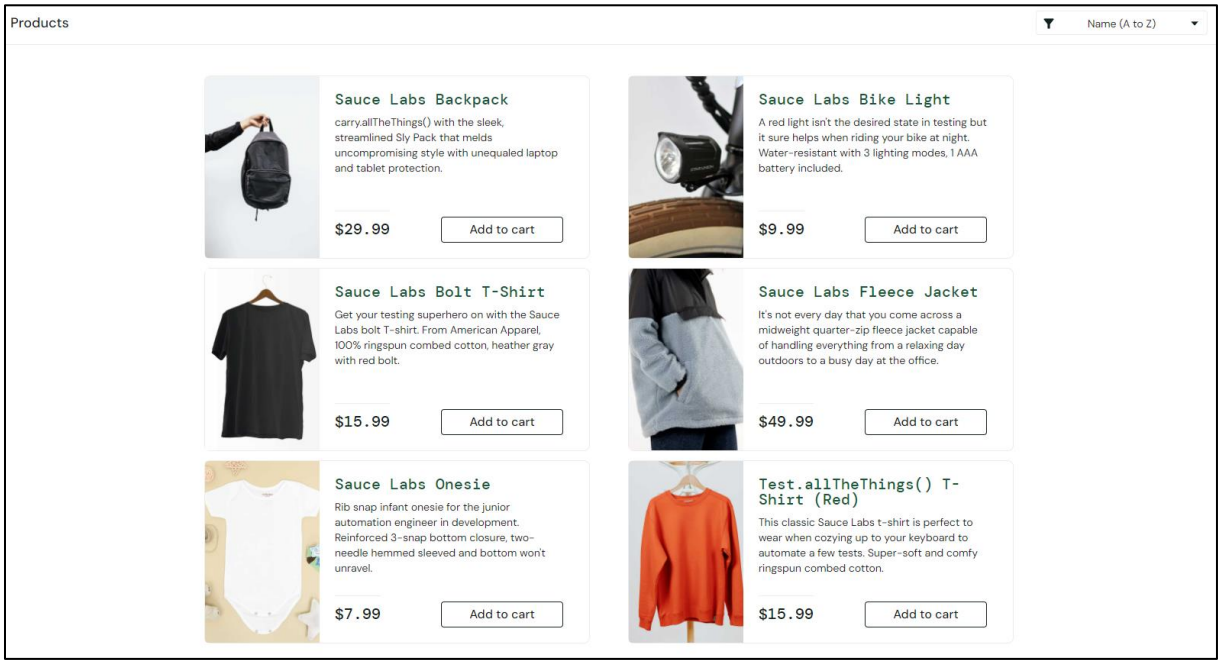
U testovima TC-05 – TC-09 su scenariji koji pokrivaju navigacijske funkcionalnosti. Vidljivi su na slici 13.

Identifikator Scenarija	Naziv Scenarija	Preduvjeti	Testni koraci	Očekivani rezultat	Rezultat testa	Opis greške	Automatizacija
			Navigacija				
TC-05	Navigacija se otvara klikom na hamburger ikonu	Prijavljeni korisnik	1. Pritisni na ikonu "Hamburger" u gornjem lijevom kutu	Otvoren je izbornik s lijeve strane	Prošao	-	Da
TC-06	Navigacija se zatvara klikom na 'x' ikonu	Prijavljeni korisnik	1. Pritisni na ikonu "Hamburger" u gornjem lijevom kutu 2. Pritisni na ikonu "x" u izborniku	Izbornik je zatvoren	Prošao	-	Da
TC-07	Pritisak na "About" otvara ispravnu stranicu	Prijavljeni korisnik	1. Pritisni na ikonu "Hamburger" u gornjem lijevom kutu 2. Pritisni na ikonu "About" u izborniku	Otvorena je stranica saucelabs.com	Prošao	-	Da
TC-08	Pritisak na "Logout" odjavljuje korisnika	Prijavljeni korisnik	1. Pritisni na ikonu "Hamburger" u gornjem lijevom kutu 2. Pritisni na ikonu "Logout" u izborniku	Korisnik je odjavljen i odveden na stranicu prijave	Prošao	-	Da
TC-09	Pritisak na ikonu košarice otvara košaricu korisnika	Prijavljeni korisnik	1. Pritisni na ikonu "Košarice" u gornjem desnom kutu	Korisniku je prikazana stranica košarice	Prošao	-	Da

Slika 13: Testni scenariji navigacije (Izvor: Autor)

6.3.1.3. Stranica svih proizvoda

Stranica svih proizvoda je cjelovita stranica koja se prikazuje korisniku nakon prijave. Sastoji se od filtera koji omogućuje različite vrste sortiranja te prikaza mreže *(engl. Grid)* svih proizvoda. Svaki od proizvoda se sastoji od svog imena, teksta, cijene te gumba za dodavanje u košaricu. Slika 14 prikazuje stranicu svih proizvoda.



Slika 14: Prikaz stranice svih proizvoda (Izvor: (Saucedemo, 2023))

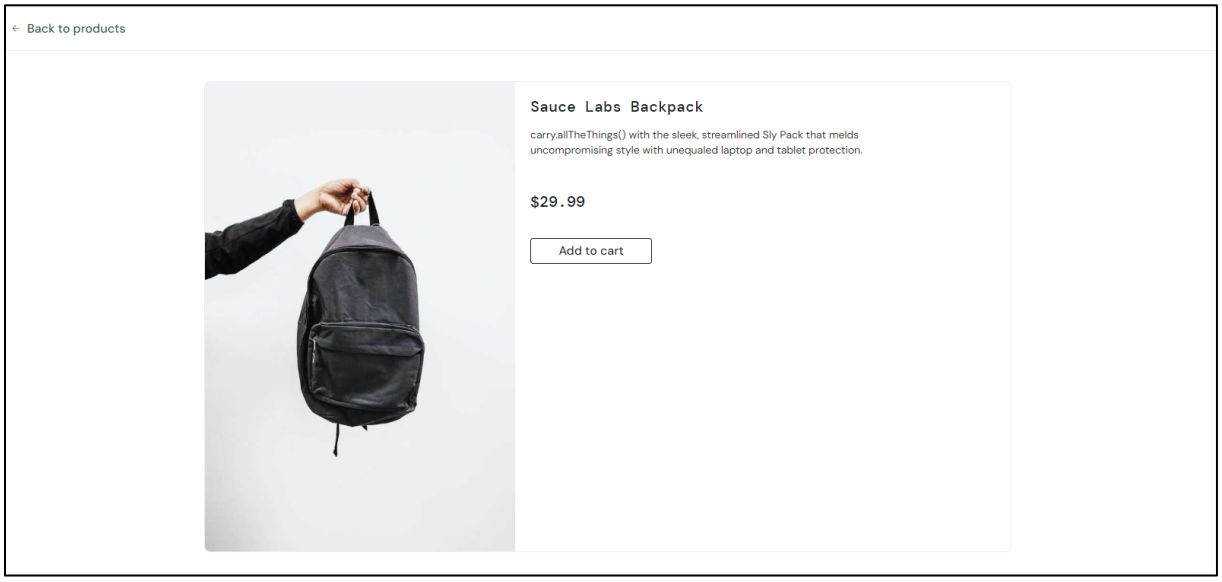
Za testiranje stranice svih proizvoda, kreirano je 7 scenarija koji su prikazani na slici 15. Scenariji TC-13 – TC-19 pokrivaju scenarije koje korisnik može izvršiti na ovoj stranici.

Identifikator Scenarija	Naziv Scenarija	Preduvjeti	Testni koraci	Očekivani rezultat	Rezultat testa	Opis greške	Automatizacija
TC-13	Abecedno sortiranje uzlazno radi ispravno	Prijavljeni korisnik	1. Pritisni na izbornik filtera 2. Odaberi opciju "Name (A to Z)"	Proizvodi su poredani po abecedi (Ime. Silazno)	Prošao	-	Da
TC-14	Abecedno sortiranje silazno radi ispravno	Prijavljeni korisnik	1. Pritisni na izbornik filtera 2. Odaberi opciju "Name (Z to A)"	Proizvodi su poredani po abecedi (Ime. Uzlazno)	Prošao	-	Da
TC-15	Cijenovno sortiranje uzlazno radi ispravno	Prijavljeni korisnik	1. Pritisni na izbornik filtera 2. Odaberi opciju "Price (low to high)"	Proizvodi su poredani po cijeni (Od niže prema višoj)	Prošao	-	Da
TC-16	Cijenovno sortiranje silazno radi ispravno	Prijavljeni korisnik	1. Pritisni na izbornik filtera 2. Odaberi opciju "Price (high to low)"	Proizvodi su poredani po cijeni (Od više prema nižoj)	Prošao	-	Da
TC-17	Pritisak na određeni proizvod otvara detaljan prikaz proizvoda	Prijavljeni korisnik	1. Pritisni na određeni proizvod	Otvorena je stranica s detaljnim prikazom proizvoda	Prošao	-	Da
TC-18	Pritisak na "Add to cart" dodaje proizvod u košaricu	Prijavljeni korisnik	1. Pritisni na "Add to cart" na proizvodu	Proizvod je dodan u košaricu. Ikona košarice ima naznaku dodanog proizvoda	Prošao	-	Da
TC-19	Pritisak na "Remove" briše proizvod iz košarice	Prijavljeni korisnik	1. Pritisni na "Remove" na proizvodu	Proizvod je obrisan iz košarice. Naznaka se maknula s ikone košarice	Prošao	-	Da

Slika 15: Testni scenariji stranice svih proizvoda (Izvor: Autor)

6.3.1.4. Stranica pojedinog proizvoda

Stranica pojedinog proizvoda prikazuje detaljne informacije o odabranom proizvodu. Na nju korisnik dolazi klikom na ime bilo kojeg proizvoda. Sastoji se od imena proizvoda, opisa proizvoda, cijene te gumba za dodavanje u košaricu. Isto tako, na njoj se nalazi gumb koji korisnika vodi nazad na pregled svih proizvoda. Na slici 16 prikazan je opisani ekran.



Slika 16: Prikaz stranice odabranog proizvoda (Izvor: (Saucedemo, 2023))

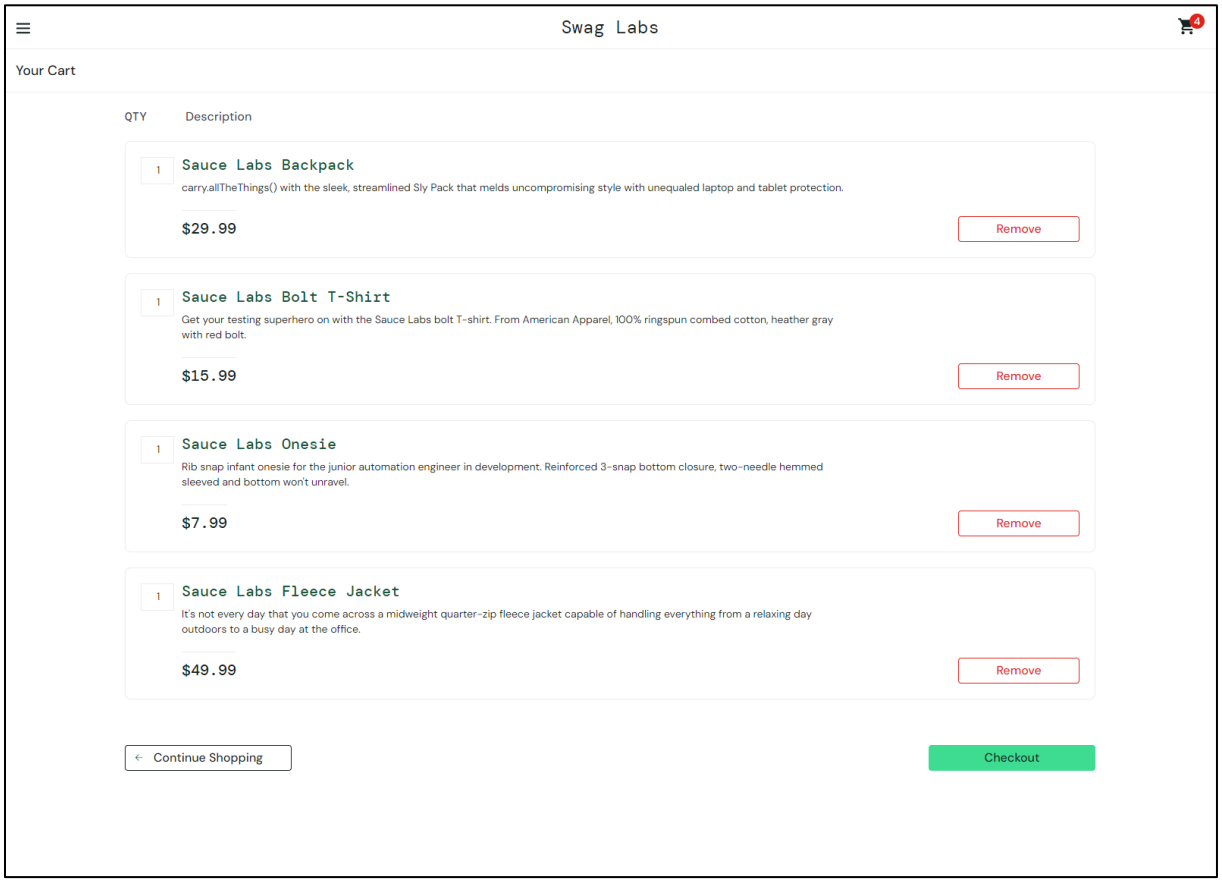
Za testiranje ove stranice napisana su tri testna scenarija. Scenariji TC-20 – TC-22 prikazani su na slici 17.

Identifikator Scenarija	Naziv Scenarija	Preduvjeti	Testni koraci	Očekivani rezultat	Rezultat testa	Opis greške	Automatizacija
Stranica pojedinog proizvoda							
TC-20	Pritisak na "Back to products" vodi na ispravnu stranicu	Prijavljeni korisnik	1. Pritisni na proizvod 2. Pritisni "Back to products"	Otvorena je stranica prikaza svih proizvoda	Prošao	-	Da
TC-21	Pritisak na "Add to cart" dodaje proizvod u košaricu	Prijavljeni korisnik	1. Pritisni na proizvod 2. Pritisni "Add to cart"	Proizvod je dodan u košaricu. Ikona košarice ima naznaku dodanog proizvoda	Prošao	-	Da
TC-22	Pritisak na "Remove" briše proizvod iz košarice	Prijavljeni korisnik	1. Pritisni na proizvod 2. Pritisni "Add to cart" 3. Pritisni na "Remove"	Proizvod je obrisan iz košarice. Naznaka se maknula s ikone košarice	Prošao	-	Da

Slika 17: Testni scenariji stranice pojedinog proizvoda (Izvor: Autor)

6.3.1.5. Košarica

Košarica sadrži proizvode koje je korisnik dodao u nju. Do nje korisnik dolazi klikom na ikonu košarice u gornjem desnom kutu. Isto tako, ikona košarice promijeni izgled te dobije indikator ukoliko se u košarici nalaze proizvodi. Na slici 18 prikazan je izgled ekrana na kojem se nalazi prikaz košarice.



Slika 18: Prikaz košarice (Izvor: (Saucedemo, 2023))

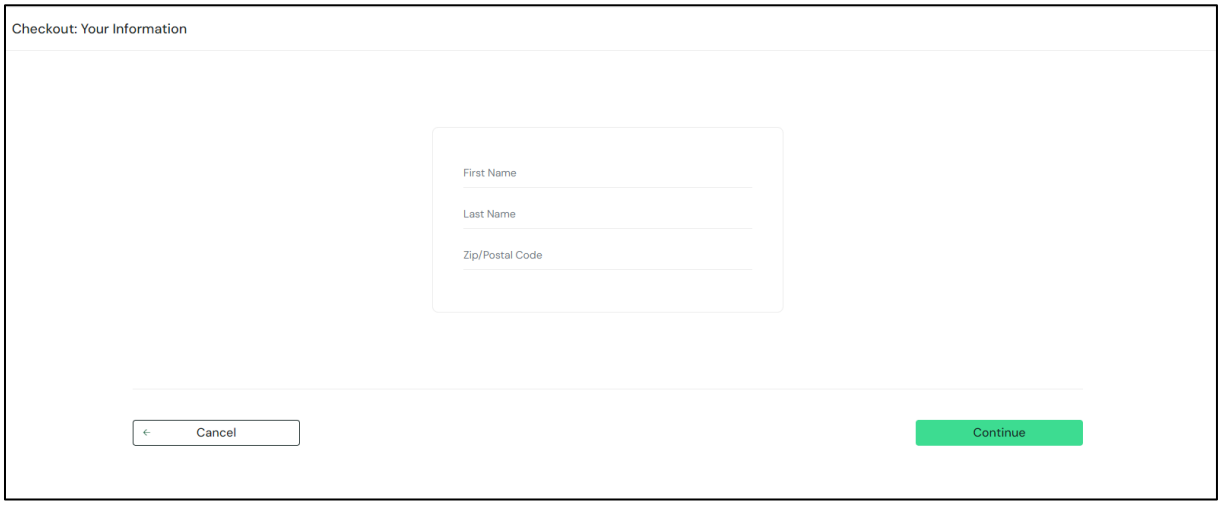
Sa stranice košarice korisnik može napraviti nekoliko akcija. Može uklanjati proizvode iz svoje košarice ili završiti kupovinu pritiskom na gumb *Checkout.* Isto tako, može se vratiti na prethodni ekran, odnosno prikaz svih proizvoda klikom na *Continue Shopping*. Ova funkcionalnost je pokrivena scenarijima TC-23 – TC-26 koji su prikazani na slici 19.

Identifikator Scenarija	Naziv Scenarija	Preduvjeti	Testni koraci	Očekivani rezultat	Rezultat testa	Opis greške	Automatizacija
Košarica							
TC-23	Pritisak na "Continue Shopping" vodi na ispravnu stranicu	Prijavljeni korisnik	1. Pritisni na ikonu košarice 2. Pritisni na "Continue Shopping"	Otvorena je stranica prikaza svih proizvoda	Prošao	-	Da
TC-24	Pritisak na "Checkout" vodi na ispravnu stranicu	Prijavljeni korisnik	1. Pritisni na ikonu košarice 2. Pritisni na "Checkout"	Otvorena je stranica za unos detaljnih informacija naplate	Prošao	-	Da
TC-25	Pritisak na "Remove" briše proizvod iz košarice	Prijavljeni korisnik, dodan proizvod u košarici	1. Pritisni "Remove"	Proizvod je obrisan i nije više vidjiv u košarici	Prošao	-	Da
TC-26	Pritisak na ime proizvoda otvara detaljan prikaz o proizvodu	Prijavljeni korisnik, dodan proizvod u košarici	1. Pritisni Ime proizvoda	Otvorena je stranica s detaljnim prikazom proizvoda	Prošao	-	Da

Slika 19: Testni scenariji stranice košarice (Izvor: Autor)

6.3.1.6. Kupnja

Na stranicu kupnje korisnik dolazi pritiskom *Checkout* gumba na stranici košarice. Stranica kupnje se sastoji od tri polja za unos teksta u koje korisnik treba unijeti svoje ime, prezime te poštanski broj. Nakon toga, može pritisnuti *Continue* za nastavak procesa kupovine ili *Cancel* kako bi prekinuo postupak. Na slici 20 prikazana je stranica kupovine.



Slika 20: Prikaz stranice kupnje (Izvor: (Saucedemo, 2023))

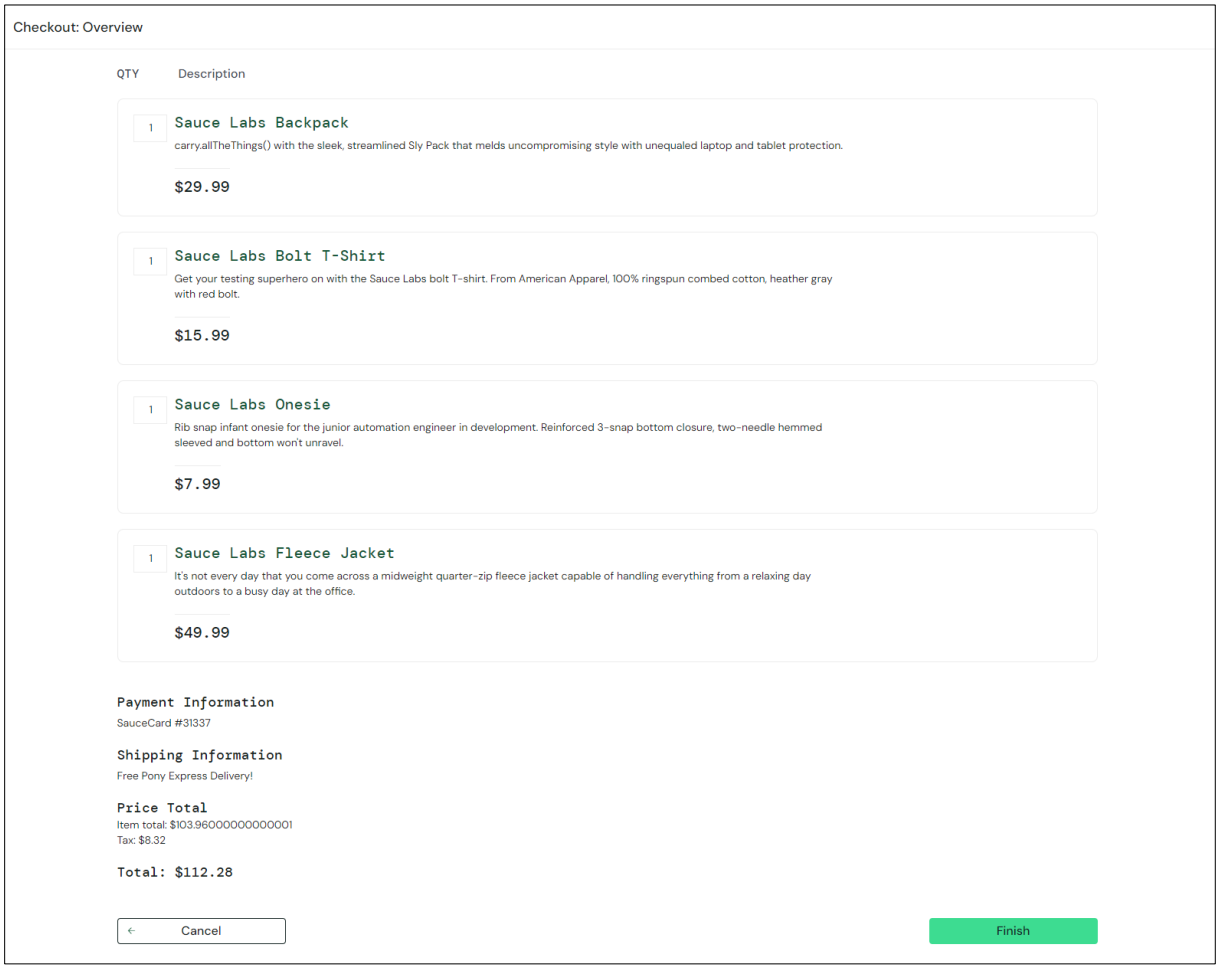
U svrhu testiranja ove stranice kreirano je pet testnih scenarija, TC-27 – TC-31 koji su prikazani na slici 21.

Identifikator Scenarija	Naziv Scenarija	Preduvjeti	Testni koraci	Očekivani rezultat	Rezultat testa	Opis greške	Automatizacija
			Kupnja				
TC-27	Ime je obavezno polje	Prijavljeni korisnik	1. Unesi prezime 2. Unesi poštanski broj 3. Pritisni "Continue"	Greška je prikazana zbog nedostatka imena. Sljedeća stranica nije otvorena.	Prošao	-	Da
TC-28	Prezime je obavezno polje	Prijavljeni korisnik	1. Unesi ime 2. Unesi poštanski broj 3. Pritisni "Continue"	Greška je prikazana zbog nedostatka prezimena. Sljedeća stranica nije otvorena.	Prošao	-	Da
TC-29	Poštanski broj je obavezno polje	Prijavljeni korisnik	1. Unesi ime 2. Unesi prezime 3. Pritisni "Continue"	Greška je prikazana zbog nedostatka poštanskog broja. Sljedeća stranica nije otvorena.	Prošao	-	Da
TC-30	Pritisak na "Cancel" vodi na ispravnu stranicu	Prijavljeni korisnik	1. Ispuni sve podatke ispravno 2. Pritisni "Cancel"	Otvorena je stranica prikaza svih proizvoda	Prošao	-	Da
TC-31	Pritisak na "Continue" vodi na ispravnu stranicu	Prijavljeni korisnik	1. Ispuni sve podatke ispravno 2. Pritisni "Continue"	Ovorena je stranica za potvrdu kupnje	Prošao	-	Da

Slika 21: Testni scenariji stranice kupnje (Izvor: Autor)

6.3.1.7. Potvrda kupnje

Stranica potvrde kupnje je posljednji korak korisnikove kupnje proizvoda. Na njoj korisnik vidi prikaz svih proizvoda iz svoje košarice koje kupuje te informacije vezane za plaćanje. Pritiskom na *Finish* korisnik završava proces kupnje, a pritiskom na *Cancel* se vraća na prikaz svih proizvoda. Na slici 22 prikazana je stranica potvrde kupnje.



Slika 22: Prikaz stranice potvrde kupnje (Izvor: (Saucedemo, 2023))

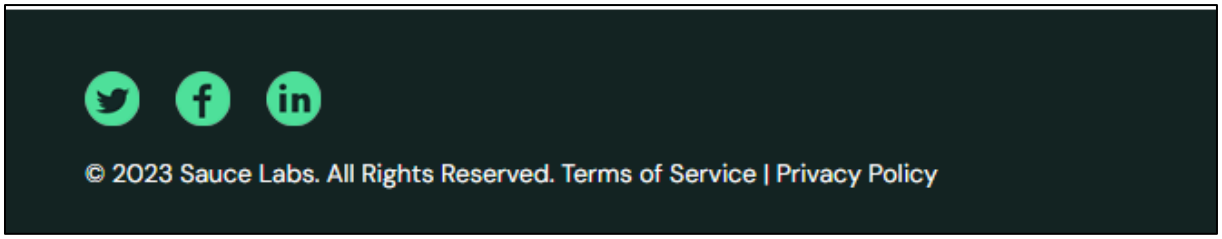
Za testiranje ove stranice, kreirano je četiri scenarija. Scenariji TC-32 – TC-35 prikazani su na slici 23.

Identifikator Scenarija	Naziv Scenarija	Preduvjeti	Testni koraci	Očekivani rezultat	Rezultat testa	Opis greške	Automatizacija
Potvrda kupnje							
TC-32	Pritisak na "Cancel" vodi na ispravnu stranicu	Prijavljeni korisnik	1. Pritisni na "Cancel"	Otvorena je stranica za Kupnju	Prošao	-	Da
TC-33	Pritisak na "Finish" potvrđuje kupnju	Prijavljeni korisnik	1. Pritisni na "Finish"	Otvorena je stranica s potvrđenom kupnjom	Prošao	-	Da
TC-34	Pritisak na "Back Home" vodi na ispravnu stranicu	Prijavljeni korisnik	1. Pritisni na "Back Home"	Otvorena je stranica s prikazom svih proizvoda	Prošao	-	Da
TC-35	Pritisak na "Finish" briše sve artikle iz košarice	Prijavljeni korisnik	1. Pritisni na "Finish"	Nakon uspješne kupnje, svi proizvodi su obrisani iz košarice	Prošao	-	Da

Slika 23: Testni scenariji stranice potvrde kupnje (Izvor: Autor)

6.3.1.8. Podnožje

Podnožje stranice se sastoji od tri jednostavne akcije, odnosno poveznica na društvene mreže Facebook, Twitter te LinkedIn. Na slici 24 prikazano je podnožje stranice.



Slika 24: Prikaz podnožja stranice (Izvor: (Saucedemo, 2023))

Na slici 25 prikazani su testni scenariji koji su kreirani u svrhu testiranja podnožja.

Identifikator Scenarija	Naziv Scenarija	Preduvjeti	Testni koraci	Očekivani rezultat	Rezultat testa	Opis greške	Automatizacija
Podnožje							
TC-10	Pritisak na "Twitter" ikonu otvara ispravnu stranicu	Prijavljeni korisnik	1. Pritisni na ikonu "Twittera" u donjem lijevom kutu stranice	Korisnik je odveden na twitter.com/saucelabs u novom prozoru	Prošao	-	Da
TC-11	Pritisak na "Facebook" ikonu otvara ispravnu stranicu	Prijavljeni korisnik	1. Pritisni na ikonu "Facebook" u donjem lijevom kutu stranice	Korisnik je odveden na facebook.com/saucelabs u novom prozoru	Prošao	-	Da
TC-12	Pritisak na "LinkedIn" ikonu otvara ispravnu stranicu	Prijavljeni korisnik	1. Pritisni na ikonu "LinkedIn" u donjem lijevom kutu stranice	Korisnik je odveden na https://www.linkedin.com/company/sauce-labs/ u novom prozoru	Prošao	-	Da

Slika 25: Testni scenariji stranice podnožja (Izvor: Autor)

6.3.2. Izvođenje testnih scenarija

Proces izvođenja testnih scenarija odrađen je na način da su se izvršili svi testni scenariji (njih 35) za svakog od 3 vrste korisnika. Ovo poglavlje prikazati će rezultate izvršavanja testnih scenarija za sve 3 vrste korisnika, prikazujući vrijeme pojedinog izvršavanja, prosječno vrijeme izvršavanja te informacije o rezultatima izvršavanja.

Izvještaji koji su prikazani su kreirani po uzoru na prikazani primjer izvještaja u teoretskom dijelu, te će se poslužiti tabličnim i grafičkim prikazom grešaka.

6.3.2.1. Izvođenje testnih scenarija standardnog korisnika

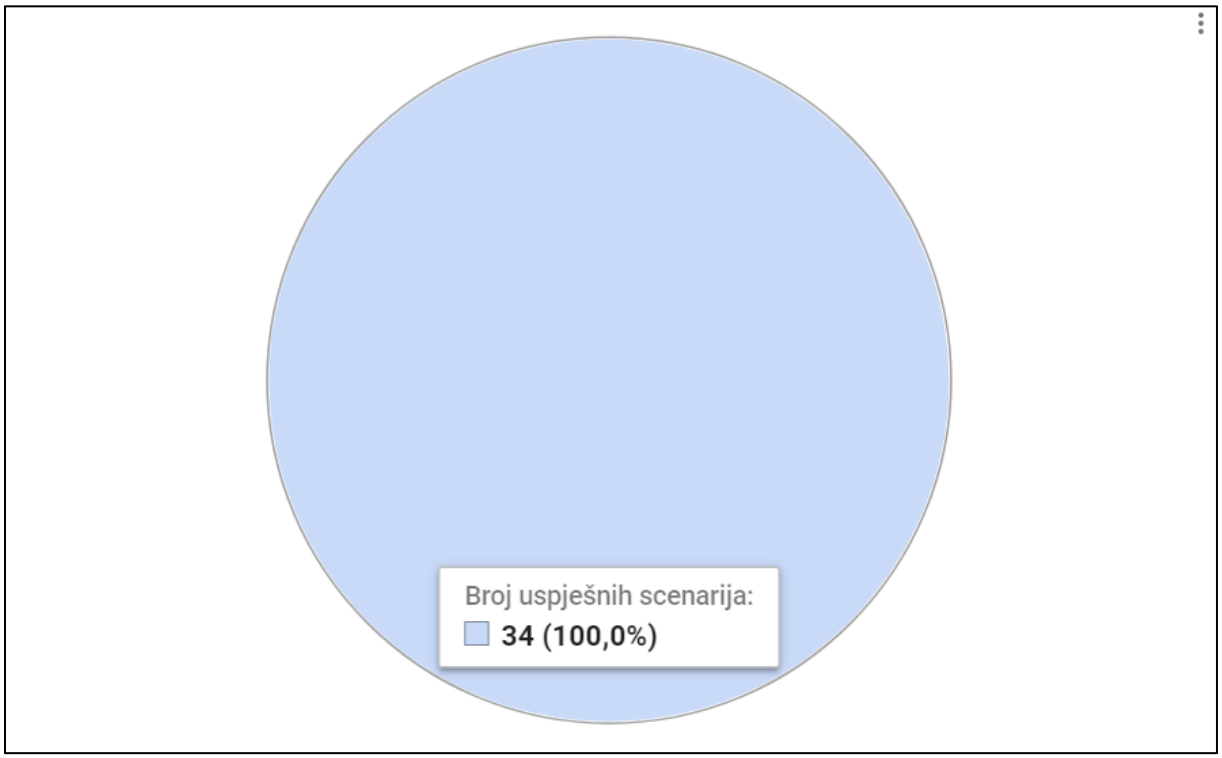
Na slici 26 prikazan je izvještaj izvođenja testiranja aplikacije. Informacije projekta sadrže osnovne informacije o projektu te datum izvođenja testiranja.

Informacije o projektu			
Naziv projekta:	*SauceLabs web shop*		
Naziv proizvoda:	*Swag Labs*		
Opis proizvoda:	Web Shop aplikacija za kupovinu robe. Testiranje sa standardnim korisnikom.		
Vrijeme početka:	25.08.2023	**Vrijeme završetka:**	25.08.2023
Sažetak testnog procesa			
Izvršavanje 1			
Broj izvršenih scenarija:	34		
Broj uspješnih scenarija:	34		
Broj neuspjelih scenarija:	0		
Postotak prolaznosti:	100,00%		
Trajanje izvršavanja (min):	21		
Komentar:	Svi testni scenariji su prošli kao što je očekivano. Prolaznost je 100% te je proizvod spreman za korisničko testiranje		
Izvršavanje 2			
Broj izvršenih scenarija:	34		
Broj uspješnih scenarija:	34		
Broj neuspjelih scenarija:	0		
Postotak prolaznosti:	100,00%		
Trajanje izvršavanja (min):	19		
Komentar:	Svi testni scenariji su prošli kao što je očekivano. Prolaznost je 100% te je proizvod spreman za korisničko testiranje		
Izvršavanje 3			
Broj izvršenih scenarija:	34		
Broj uspješnih scenarija:	34		
Broj neuspjelih scenarija:	0		
Postotak prolaznosti:	100,00%		
Trajanje izvršavanja (min):	14		
Komentar:	Svi testni scenariji su prošli kao što je očekivano. Prolaznost je 100% te je proizvod spreman za korisničko testiranje		
Prosječno vrijeme izvršavanja:	18		

Slika 26: Testni izvještaj standardnog korisnika (Izvor: Autor)

Testovi su izvršavani 3 puta kako bi se dobila informacija o vremenskoj učinkovitosti. Prvo izvođenje testiranja je trajalo 21 minutu, zatim 19 minuta pa 14 minuta. Razlog tomu je upoznavanje proizvoda te poznavanje korisničkih slučajeva, što je direktno utjecalo na brzinu izvođenja scenarija.

Na slici 27 prikazan je tortni grafikon koji prikazuje postotak uspješnosti provedenih testova. Obzirom da se u ovom slučaju radi o 100% uspješnosti, grafikon izgleda jednolično.



Slika 27: Grafikon izvještaja standardnog korisnika (Izvor: Autor)

Zaključak provođenja testiranja standardnog korisnika je taj da aplikacija u potpunosti funkcionira te da je spremna za korisničko testiranje, koje će nadalje potvrditi ispravnost aplikacije.

6.3.2.2. Izvođenje testnih scenarija korisničkog profila s poteškoćama s performansama

Na slici 28 prikazan je izvještaj korisničkog profila s poteškoćama s performansama.

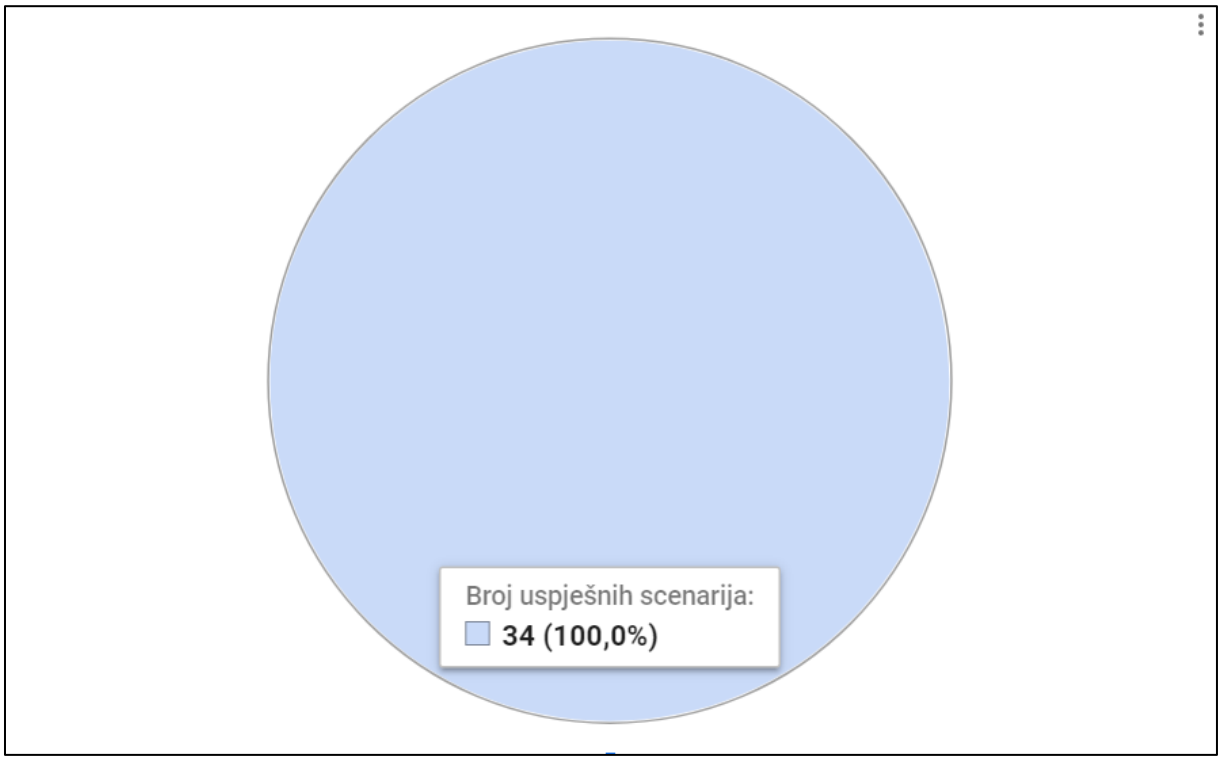
Informacije o projektu			
Naziv projekta:	*SauceLabs web shop*		
Naziv proizvoda:	*Swag Labs*		
Opis proizvoda:	Web Shop aplikacija za kupovinu robe. Testiranje sa standardnim korisnikom.		
Vrijeme početka:	25.08.2023	**Vrijeme završetka:**	25.08.2023
Sažetak testnog procesa			
Izvršavanje 1			
Broj izvršenih scenarija:	34		
Broj uspješnih scenarija:	34		
Broj neuspjelih scenarija:	0		
Postotak prolaznosti:	100,00%		
Trajanje izvršavanja (min):	25		
Komentar:	Svi testni scenariji su uspješno izvršeni. Bitno je napomenuti da su vremena učitavanja dulja od očekivanog. Molim istražiti.		
Izvršavanje 2			
Broj izvršenih scenarija:	34		
Broj uspješnih scenarija:	34		
Broj neuspjelih scenarija:	0		
Postotak prolaznosti:	100,00%		
Trajanje izvršavanja (min):	27		
Komentar:	Svi testni scenariji su uspješno izvršeni. Bitno je napomenuti da su vremena učitavanja dulja od očekivanog. Molim istražiti.		
Izvršavanje 3			
Broj izvršenih scenarija:	34		
Broj uspješnih scenarija:	34		
Broj neuspjelih scenarija:	0		
Postotak prolaznosti:	100,00%		
Trajanje izvršavanja (min):	26		
Komentar:	Svi testni scenariji su uspješno izvršeni. Bitno je napomenuti da su vremena učitavanja dulja od očekivanog. Molim istražiti.		
Prosječno vrijeme izvršavanja:	26		

Slika 28: Testni izvještaj korisničkog profila s poteškoćama s performansama (Izvor: Autor)

Kao što je prikazano u izvještaju te grafičkom prikazu na slici 29, svi scenariji su uspješno odrađeni te je prolaznost testiranja 100%.

Iako su svi testni scenariji uspješno izvršeni, bitno je obratiti pozornost na komentar izvođenja scenarija. U prethodnom primjeru, scenariji su odobreni i proizvod je bio spreman za korisničko testiranje. U ovom slučaju, iako su svi testni scenariji uspješno provedeni, postoje određene greške u performansama te je potrebno dodatno provjeriti o čemu se radi.

Vrijeme izvršavanja scenarija isto tako potkrepljuje gore navedeno. Vidljivo je da je vrijeme izvođenja u prosjeku 40% dulje od prethodnog slučaja, što upućuje na dulja vremena učitavanja stranice, izvođenja akcija i slično.



Slika 29: Grafikon izvještaj korisničkog profila s poteškoćama s performansama (Izvor: Autor)

6.3.2.3. Izvođenje testnih scenarija profila korisnika s greškom u radu

Na slici 30 prikazan je izvještaj testiranja profila korisnika s greškom u radu. U ovom slučaju dolazi do prvih padanja testnih scenarija i do prijavljenih grešaka.

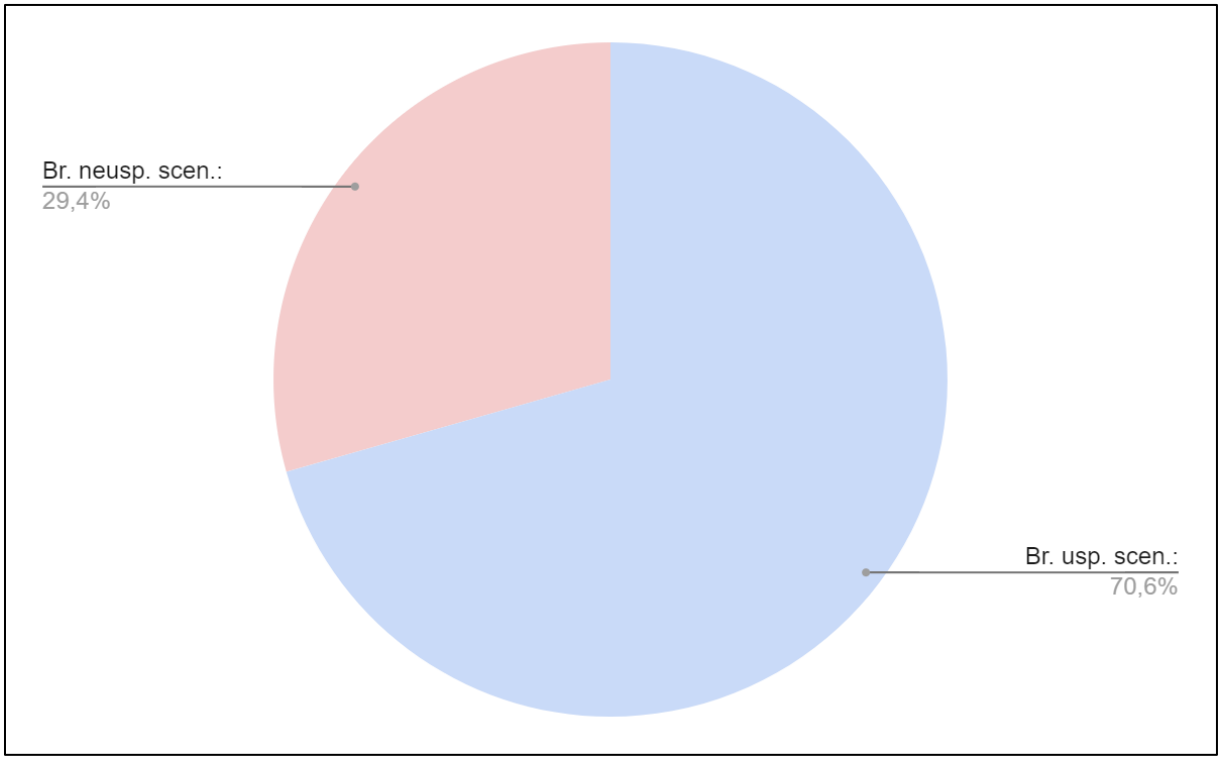
Informacije o projektu			
Naziv projekta:	*SauceLabs web shop*		
Naziv proizvoda:	*Swag Labs*		
Opis proizvoda:	Web Shop aplikacija za kupovinu robe. Testiranje sa standardnim korisnikom.		
Vrijeme početka:	25.08.2023	**Vrijeme završetka:**	25.08.2023
Sažetak testnog procesa			
Izvršavanje 1			
Broj izvršenih scenarija:	34		
Broj uspješnih scenarija:	24		
Broj neuspjelih scenarija:	10		
Postotak prolaznosti:	70,59%		
Trajanje izvršavanja (min):	32		
Komentar:	10 Testnih scenarija ne prolazi. Svi su ključni za funkcioniranje aplikacije te je potrebno pogledati izvještaj grešaka i ispraviti ih.		
Izvršavanje 2			
Broj izvršenih scenarija:	34		
Broj uspješnih scenarija:	24		
Broj neuspjelih scenarija:	10		
Postotak prolaznosti:	70,59%		
Trajanje izvršavanja (min):	31		
Komentar:	10 Testnih scenarija ne prolazi. Svi su ključni za funkcioniranje aplikacije te je potrebno pogledati izvještaj grešaka i ispraviti ih.		
Izvršavanje 3			
Broj izvršenih scenarija:	34		
Broj uspješnih scenarija:	24		
Broj neuspjelih scenarija:	10		
Postotak prolaznosti:	70,59%		
Trajanje izvršavanja (min):	27		
Komentar:	10 Testnih scenarija ne prolazi. Svi su ključni za funkcioniranje aplikacije te je potrebno pogledati izvještaj grešaka i ispraviti ih.		
Prosječno vrijeme izvršavanja:	30		

Slika 30: Testni izvještaj profila korisnika s greškom u radu (Izvor: Autor)

Od izvršenih 34 testna scenarija, 10 ih je izvršeno s greškom, odnosno očekivani rezultat testnog scenarija nije zadovoljen.

U slučajevima kada testni scenariji javljaju grešku, potrebno je kreirati testni izvještaj koji prikazuje popis grešaka i detaljno ih opisuje. U svrhu ovog rada, i kako se ne bi ponavljale informacije, testni izvještaj prikazan je u dijelu rada koji opisuje automatsko testiranje ovog proizvoda.

Na slici 31 prikazan je tortni dijagram koji sada jasno prikazuje postotak prolaznosti, odnosno ne prolaznosti izvršenih testnih scenarija za profila korisnika s greškom u radu.



Slika 31: Grafikon izvještaj profila korisnika s greškom u radu (Izvor: Autor)

6.4. Programski kod i pokretanje testova

Ovo poglavlje za svrhu ima prikazati način pisanja i strukturiranja testova. Testovi će biti strukturirani i pisani po 10 pravila opisanih u teorijskom dijelu.

Kao što su manualni testovi bili prikazani po stranicama i komponentama koje su automatizirali, tako će i automatski testovi biti podijeljeni. Prije samog opisa testova, prikazana je općenita struktura *Cypress* projekta te komponente od kojih se on sastoji.

6.4.1. Struktura projekta

Struktura projekta prikazana je u isječku ispod:

```
cypress
        e2e
                kosarica.cy.js
                kupnja.cy.js
                navigacija.cy.js
                podnozje.cy.js
                potvrda_kupnje.cy.js
                prijava.cy.js
                stranica_pojedinog_proizvoda.cy.js
                stranica_svih_proizvoda.cy.js
        fixtures
                korisnici.json
                podatci_za_kupnju.json
                podatci_za_prijavu.json
        support
                commands.js
                e2e.js
                utils.js
        screenshots
```

Struktura *Cypress* projekta se sastoji od glavne datoteke *Cypress* te od četiri pomoćne datoteke: *e2e, fixtures, support* te *screenshots.* Datoteka *e2e* sadrži dokumente koji predstavljaju testne skupove, odnosno dokumente s testovima koji će se pri pokretanju pokretati. Iz tog razloga, ova datoteka je izostavljena u ovom djelu i opisana u samom opisu programskog koda. Ostale datoteke opisane su u nastavku teksta.

6.4.1.1. Datoteka *fixtures*

Datoteka *fixtures* predstavlja sve dodatne i pomoćne datoteke koje se koriste u samom testiranju. U navedenom primjeru, nalaze se tri datoteke:

```
korisnici.json
podatci_za_kupnju.json
podatci_za_prijavu.json
```

Datoteka *korisnici.json* u sebi sadrži popis korisnika, odnosno testnih scenarija za korisnike koji se koriste kod prijave u sustav. Ova datoteka se direktno koristi u testnim scenarijima vezanim za prijavu u sustav te je njen sadržaj i struktura detaljnije opisan u poglavlju koje opisuje testne scenarije za prijavu.

Datoteka *podatci_za_kupnju.json* ima identičnu svrhu kao i datoteka *korisnici.json*. Sadrži podatke o testnim scenarijima vezanim za kupnju proizvoda.

Datoteka *podatci_za_prijavu.json* koristi se kako bi se lako mijenjao korisnik s kojim se želimo prijaviti. Sadržaj datoteke prikazan je ispod:

```
{
    "standardni_korisnik" : "standard_user",
    "korisnik_s_problemom" : "problem_user",
    "korisnik_s_greskom_u_radu" : "performance_glitch_user",
    "lozinka" : "secret_sauce"
}
```

Definirane su varijable za svaku vrstu korisnika te za njihovu lozinku. U slučaju promjene informacija za prijavu, promjeni se desna strana npr. „`standard_user`“ u drugu željenu vrijednost i ta vrijednost aplicirana je na sve varijable „`standardni_korisnik`“. Primjena ove datoteke u radu koda prikazana je u nadolazećem tekstu vezanom za pisanje koda.

U datoteku *fixtures* mogu se staviti svi potrebni dokumenti, za npr. Prebacivanje (*engl. Upload)* dokumenta, slike s kojima korisnik može raditi, audio zapisi, video zapisi i slično.

6.4.1.2. Datoteka *support*

Datoteka *support* služi kako bi se u nju pohranili svi dokumenti koji u sebi sadrže dodatan kod koji se koristi u svrhu izvođenja testova. U ovom projektu postoje tri datoteke, a to su *commands.js, e2e.js* te *utils.js.*

U nastavku su svaka od navedenih datoteka opisane te je prikazana njihova svrha u kontekstu ovog projekta.

Datoteka *commands.js* u sebi sadrži naredbe koje se koriste kroz sve testne dokumente. Njena najčešća svrha je kako bi se pohranile naredbe koje se koriste za dovođenje testova u određeno stanje.

U kodu ispod prikazana je naredba `dohvati_sve_artikle()` koja se koristi kako bi se dohvatio popis svih artikala sa stranice. Ova naredba se koristi kod testiranja sortiranja na stranici. Njena svrha je da dohvati sve artikle sa stranice a zatim te iste artikle pohrani u mapu na način da pohrani kombinacije imena i cijene. Nakon toga, u testnom dokumentu se provjerava redoslijed artikala ovisno o uvjetu sortiranja te se verificira ispravno funkcioniranje sortiranja.

```
Cypress.Commands.add("dohvati_sve_artikle", () => {
    cy.get('.inventory_item').then(artikli => {
      const mapaArtikala = new Map();

      artikli.each((index, artikl) => {
        const naziv =
Cypress.$(artikl).find('.inventory_item_name').text();
        const cijena =
Cypress.$(artikl).find('.inventory_item_price').text();

        mapaArtikala.set(naziv, cijena);
      });

      return mapaArtikala;
    });
  });
```

Pomoću naredbe `cy.get('.inventory_item')` dohvaćaju se svi elementi koji u sebi imaju klasu `'inventory_item'`. Pomoću `'.'` dohvaćaju se elementi iz DOM-a po atributu klase. Nakon što su dohvaćeni svi artikli, prolazi se kroz njih `.each()` naredbom te se dohvaćaju njihovi atributi imena i cijene koji se zatim spremaju u mapu koja se sastoji od njihovih kombinacija. Na samom kraju, mapa svih artikala se vraća kao odgovor metode.

U sljedećim naredbama se pokrivaju funkcije koje se koriste na svim stranicama. Te funkcije su najčešće one koje se nalaze u navigacijskoj traci te su dostupne na svakoj stranici. Uz funkcije iz navigacijske trake, također se koriste funkcije koje nam služe za rad s artiklima, obzirom da te iste funkcije postoje na više stranica aplikacije.

```
Cypress.Commands.add("pritisni_kosaricu", () =>{
  cy.get('#shopping_cart_container').click()
})
```

Funkcija `pritisni_kosaricu()` prikazana u kodu iznad se koristi kako bi se jednostavno navigiralo na korisnikovu košaricu. Iako naredba sadrži samo jednu liniju koda te je teoretski moguće koristiti tu jednu liniju kada god je potrebno, korištenjem aliasa `cy.pritisni_kosaricu()` daje čitatelju koda puno više informacija o samoj radnji od linije koja glasi `cy.get('#shopping_cart_container').click()`.

Naredne dvije naredbe se koriste kako bi se prozivod dodao u košaricu te maknuo iz košarice. Nakon svake naredbe, provjerava se stanje oznake na ikoni košarice. Nakon što se proizvod doda u košaricu, očekuje se da će se pojaviti oznaka artikla na ikoni košarice, što se provjerava naredbom: `cy.get('.shopping_cart_badge').should('exist')` koja govori da mora postojati element koji sadrži klasu .shopping_cart_badge. Jednako funkcionira i provjera koji potvrđuje da indikator na košarici ne postoji nakon što se proizvod makne iz košarice: `cy.get('.shopping_cart_badge').should('not.exist')` koja pomoću `'not.exist'` naredbe provjerava da element ne postoji u sastavu stranice (DOM-u).

```
Cypress.Commands.add("dodaj_proizvod_u_kosaricu", () =>{
  cy.contains('Add to cart').click()
  cy.get('.shopping_cart_badge').should('exist')
})

Cypress.Commands.add("ukloni_proizvod_iz_kosarice", () =>{
  cy.contains('Remove').click()
  cy.get('.shopping_cart_badge').should('not.exist')
})
```

Korištenjem *commands.js* dokumenta, poštuje se pravilo Smanji, Recikliraj, Ponovno iskoristi koje govori kako se ponovno iskoristivi dijelovi koda trebaju odvajati na logičke cjeline.

Drugi dokument u datoteci *support* je *utils.js*. *Utils.js*, za razliku od *support.js* i *e2e.js* nije datoteka koja se kreira kod postavljanja projekta. Ta datoteka je kreirana kako bi se pohranile naredbe koje će se koristiti u svrhu generiranja testova. Radi jednostavnijeg objašnjavanja i objašnjavanja u kontekstu testnih dokumenata u kojima se koriste, ove naredbe objašnjene su u kasnijim poglavljima.

6.4.2. Programski kod

U narednim pod poglavljima opisani su dijelovi koda, odnosno testovi. Razvrstani su po stranicama aplikacije koje testiraju. Nakon prikaza programskih kodova, objašnjena je njihova funkcija te naredbe koje se koriste.

Isto tako, fokus je stavljen na pravila pisanja automatizacije koja su opisana u teorijskom dijelu rada.

Zbog količine koda, za svaku od stranica prikazane su specifičnosti te ono što nije prikazano na niti jednoj drugoj stranici.

6.4.2.1. Testiranje prijave

Od svih testnih dokumenata, prijava sadrži najkompleksniji sistem pisanja testova. Način je kompleksan iz razloga što su komponente za testiranje prijave podijeljene u tri različita dokumenta i datoteka, a oni su:

- `e2e/prijava.cy.js`
- `fixtures/korisnici.json`
- `support/utils.js`

Struktura pisanja ovih testova razlikuje se od ostalih po tome što je zamišljena kao održivi sustav kojeg bilo tko može nadograđivati, pa čak i osobe koje nemaju programerskog znanja ili iskustva. Sustav funkcionira tako da su testni scenariji definirani u *korisnici.json* dokumentu kao lista objekata.

```
{
    "korisnici":[
    {
        "test" : "Uspjesna prijava",
        "ime" : "standard_user",
        "lozinka" : "secret_sauce",
        "uspjeh" : true,
        "greska" : null
    },
    {
        "test" : "Neuspjesna prijava zakljucanog korisnika",
        "ime" : "locked_out_user",
        "lozinka" : "secret_sauce",
        "uspjeh" : false,
        "greska" : "Epic sadface: Sorry, this user has been locked out."
    }]}
```

Na isječku iz dokumenta *korisnici.json* iznad, prikazana je struktura *.json* dokumenta te atributi svakog od objekta.

- `test` – Atribut koji definira ime testnog scenarija
- `ime` – Atribut koji definira korisničko ime koje će se unijeti u polje
- `lozinka` – Atribut koji definira korisničku lozinku koja će se unijeti u polje
- `uspjeh` – Varijabla koja poprima jednu od dvije vrijednosti, `true` ili `false` te definira očekuje li se uspješna prijava ili ne
- `greska` – Varijabla koja definira poruku u slučaju neuspješne prijave

Zatim slijedi ispunjavanje forme za prijavu koristeći gore navedene atribute. U tu svrhu je kreirana metoda `prijava()` koja se nalazi u datoteci *utils.js*. Njen zadatak je da dohvati elemente sa stranice pomoću `cy.get()` metode te zatim u njih upiše vrijednosti iz proslijeđenih vrijednosti pomoću `cy.type()` metode. Nakon ispunjavanja polja, potrebno je dohvaćeni gumb pritisnuti pomoću `cy.click()` metode.

```
export const prijava = (kor_ime, lozinka, postavke = {}) => {
    cy.visit('/');
    cy.get('#user-name').type(kor_ime);
    cy.get('#password').type(lozinka);
    cy.get('#login-button').click();

    if (!postavke.uspjeh) {
      cy.contains(postavke.greska).should('exist');
    } else {
      cy.get('#inventory_container').should('exist');
    }
  };
```

Nakon unosa podataka i pritiska na gumb, potrebno je provjeriti stanje aplikacije. Ukoliko se očekuje neuspjeh testa, odnosno `uspjeh` varijabla je postavljena na `false`, potrebno je provjeriti nalazi li se očekivani tekst iz atributa `greska`. U suprotnom, očekuje se da je prijava uspješna te da se učitala stranica nakon uspješne prijave, odnosno provjeravamo postoji li objekt s identifikatorom `inventory_container`, što predstavlja popis svih artikala.

Nakon što je omogućena logika ispunjavanja polja te izvođenja testova, potrebno je kreirati testove iz njih. To se obavlja u dokumentu *prijava.cy.js.* Za početak, potrebno je uključiti i učitati sve potrebne datoteke za rad, odnosno povezati datoteke svih testnih slučajeva i datoteku u kojoj se nalazi metoda `prijava()`, što je vidljivu u naredne dvije linije koda.

```
import podatci from '../fixtures/korisnici.json';
import { prijava } from '../support/utils';
```

U prvoj liniji koda, uključena je datoteka svih korisnika, odnosno testnih scenarija i spremljena pod ime varijable podatci. U drugoj liniji, uključena je metoda `prijava()` iz *utils.js* dokumenta. Ovaj proces se ponavlja i u drugim testnim dokumentima s različitim datotekama podataka i različitim metodama te više neće biti spominjan.

Nakon uključivanja potrebnih datoteka, potrebno je kreirati grupu scenarija pomoću metode `describe()` koja nam omogućuje da spremimo više testnih scenarija pod jednu grupu. Još jedna njena funkcija je omogućavanje dovođenja svih testnih scenarija unutar grupe u isto stanje, što će biti objašnjeno u jednom od nadolazećih primjera koji tu funkcionalnost koriste.

Zatim se prolazi kroz svaki objekt iz liste podatci te se kreira novi testni scenarij pomoću metode `it()`. Kao atribut imena u metodu `it()` prosljeđujemo ime testa, odnosno atribut test koji smo definirali u `korisnici.json`. Nakon toga pozivamo uključenu metodu `prijava()` za svakog od korisnika te joj prosljeđujemo podatke iz objekta korisnika.

```
describe('Testiranje prijave korisnika', () => {
  podatci.korisnici.forEach((korisnik) => {
    it(`${korisnik.test}`, () => {
      prijava(korisnik.ime, korisnik.lozinka, {
        uspjeh: korisnik.uspjeh,
        greska: korisnik.greska,
      });
    });
  });
});
```

Na ovaj način kreirali su se održivi testni scenariji koji se mogu uređivati od strane testera bez znanja programiranja. Isto tako, razdvajanjem testnih metoda i logike u više dokumenata postignute su više razine apstrakcije, kao i poštivano pravilo razdvajanja većih testnih scenarija na manje. Uz to, kreirani test ima isključivo jednu zadaću, a to je provjeriti ispravnost prijave. Ovo je primjer kako samo pisanjem jednog testa, mogu se pokriti 7 od 10 pravila za pisanje automatizacije.

Opisivanje strukture testova, odnosno korištenja metoda `describe()` i `it()` nije opisivano u daljnjem radi izbjegavanja redundancije u tekstu.

6.4.2.2. Testiranje navigacije

Testiranje navigacije se sastoji od pet jednostavnijih testova koji testiraju funkcionalnosti padajućeg izbornika i navigacijske trake.

Za početak, unutar grupe testova, prikazano je ispravno korištenje metode `beforeEach()`. Ta metoda prije izvođenja svakog testa obavi sve akcije i pozove sve metode koje se u njoj nalaze. Na taj način uvijek dovodimo aplikaciju u jednako stanje prije svakog izvođenja testnih scenarija.

U ovom konkretnom slučaju, pozivamo metodu `prijava()` kako bi korisnika prijavili u aplikaciju svaki put prije testiranja navigacije, obzirom da je navigacija vidljiva isključivo prijavljenim korisnicima. U linijama koda ispod prikazan je način pisanja `beforeEach()` metode.

```
  beforeEach(() => {
    prijava(korisnik.standardni_korisnik, korisnik.lozinka, {uspjeh:true})
  });
```

Testni dokument razdvojen je na dva dijela, na same testove te na pomoćne metode koje inkapsuliraju akcije radi lakšeg razumijevanja. Naime, iako intuitivne, metode koje su pružene od strane *Cypress-a* u kombinaciji s imenima elemenata u web aplikaciji mogu učiniti shvaćanje samih testova težim.

```
function otvori_izbornik()
{
  pritisni_hamburger()
  cy.get('.bm-menu-wrap').should('have.attr', 'aria-hidden', 'false');
}

function pritisni_hamburger()
{
  cy.get('#react-burger-menu-btn').click()
}

function pritisni_x()
{
  cy.get('#react-burger-cross-btn').click()
}
```

U kodu iznad prikazan je način funkcioniranja prethodno objašnjenog. Kreirane su pomoćne metode koje će se koristiti u samim testovima kao testni koraci. U funkciji `pritisni_hamburger()` spremljena je linija koda koja dohvaća element padajućeg izbornika

te odrađuje korisnikov klik na njega. Na ovaj način, spremili smo nejasnu liniju koda poput `cy.get('#react-burger-menu-btn').click()` u intuitivnu metodu poput: `pritisni_hamburger()`.

Ovim pristupom postiže se lakša čitljivost testnog koda. U ovom testnom dokumentu, koristi se provjera postojanja određenog atributa u određenom elementu. To je vidljivo u funkciji `otvori_izbornik()`. Linija koda `cy.get('.bm-menu-wrap').should('have.attr', 'aria-hidden', 'false');` provjerava `nalazi li se u elementu koji sadrži klasu .bm-menu.wrap` atribut (`have.attr`) imena `aria-hidden` i provjerava je li njegova vrijednost `false`.

Kako ova funkcija sadrži validacijski korak, odnosno provjerava je li se nakon klika na ikonu padajući izbornik pojavio, ispunjava pravilo validacije na ključnim točkama. Isto tako, korištenjem `beforeEach()` metode, smanjena je potreba uvjetovanja i provjeravanja stanja aplikacije. U prva dva testna dokumenta, pokrivena su sva pravila automatskog pisanja testova.

Na isječcima koda ispod, prikazana je sama testna datoteka koja provjerava navigaciju. Prikazani su neki od testova kako bi se demonstriralo korištenje gore opisanih metoda.

```
describe('Testiranje navigacije', () => {
  beforeEach(() => {
    prijava(korisnik.standardni_korisnik, korisnik.lozinka, {uspjeh:true})
  });

  it('Navigacija je zatvorena klikom na x', () => {
    otvori_izbornik()
    pritisni_x()
    cy.get('.bm-menu-wrap').should('have.attr', 'aria-hidden', 'true');
  });

  it('About opcija sadrži poveznicu na saucelabs.com', () => {
    otvori_izbornik()
    cy.get('#about_sidebar_link').should('have.attr', 'href',
'https://saucelabs.com/')
  });
});
```

6.4.2.3. Testiranje svih proizvoda

Testiranje stranice svih proizvoda donosi još jedan izazov u testiranju korisničkih sučelja te će fokus ovog poglavlja biti na tom izazovu, obzirom da su ostali elementi pokriveni u prethodne dvije vrste testiranja.

Stranica svih proizvoda sadrži sortiranje po četiri različita uvjeta. Kako bi se obavila akcija sortiranja, potrebno je dohvatiti sve proizvode sa stranice te dohvatiti njihove atribute, koji će se zatim uspoređivati s ostalim atributima iz liste kako bi se provjerila ispravnost sortiranja.

Za početak, u svakom od testova potrebno je odabrati željeni filter. To se obavlja na način da se dohvati padajući izbornik pomoću metode `get()`, a zatim pomoću metode `select()` odabere vrijednost koju želimo.

Zatim, potrebno je dohvatiti sve artikle sa stranice. Za to je kreirana komanda `dohvati_sve_artikle` koja je opisana u poglavlju koje govori o strukturi koda. Pomoću ove komande, dohvatit će se svi artikli te pohraniti u listu modeliranu kao mapa koja će sadržavati kombinacije imena i cijene svih artikala na stranici.

Obzirom na način funkcioniranja *JavaScript*-a, a samim time i *Cypress*-a, nije moguće interno pohraniti podatke dohvaćene sa stranice, iz razloga što će se nakon promjene stanja stranice ti elementi maknuti. Iz tog razloga, potrebno je ulančati funkcije dohvaćanja proizvoda i provjere ispravnosti sortiranja pomoću metode `then()`. Spajanjem ovdje opisanih radnji, dobije se testna metoda nalik na komad koda ispod.

```
it('Filter "Name A-Z" sortira proizvode po abecedi, uzlazno', () => {
  cy.get('.product_sort_container').select('Name (A to Z)');
  cy.dohvati_sve_artikle().then(artikli => {
    provjeri_poredak_a_z(artikli)
  });
```

Metoda `provjeri_poredak_a_z()` kreirana je u istom dokumentu iz razloga što se ne koristi u drugim dokumentima te nema potrebe izdvajati ju. Njena zadaća je da prolazi kroz sve artikle te uspoređuje cijenu, odnosno abecedni poredak s onim artiklom prije. Ukoliko dođe do ne slaganja, ovisno o uvjetu filtera, izbaciti će grešku i pomoću `cy.fail()` metode učiniti test neuspjelim. Na taj način provjerena je ispravnost rada filtera.

Ponovno, korištenjem ove strukture i razdvajanja koda na smislene dijelove s jedinstvenom zadaćom poštivana su gotovo sva pravila pisanja automatiziranih testova koja su primjenjiva na ovom slučaju. U kodu ispod prikazana je metoda `provjeri_poredak_a_z()`.

```
function provjeri_poredak_a_z(artikli) {
  let prethodni = null;


  artikli.forEach((cijena, naziv) => {
    if (prethodni !== null && naziv < prethodni) {
      cy.fail(`Artikli nisu abecedno poredani: ${naziv} dolazi poslje
${prethodni}`);
    }
    prethodni = naziv;
  });
   }
```

6.4.2.4. Testiranje stranice pojedinog proizvoda

Stranica pojedinog proizvoda ne sadrži nove funkcionalnosti *Cypress*-a, ali prikazuje korištenje naredbi koje je definirao korisnik. U ovom testnom dokumentu, koriste se metode `dodaj_proizvod_u_kosaricu()` te `ukloni_proizvod_iz_kosarice()` koje su opisane u poglavlju strukture projekta.

Na ovaj način, u četiri linije moguće je kreirati testni scenarij koji u sebi sadrži dvije provjere, odnosno nakon dodavanja proizvoda u košaricu provjerava postoji li proizvod u košarici, odnosno je li se indikator košarice pojavio, te nakon brisanja proizvoda iz košarice, provjerava je li se taj isti indikator maknu.

Ovdje je vidljiva primjena pravila provjera na ključnim točkama, kao i pravila razdvajanja i ponovnog korištenja koda.

6.4.3. Pokretanje testova

Opcije pokretanja testova u *Cypress*-u su mnogobrojne. Ovo poglavlje opisati će osnovne opcije pokretanja testova pomoću grafičkog sučelja te pomoću naredbenog retka.

6.4.3.1. Pokretanje testova naredbenim retkom

Osnovna naredba kojom se pokreću testovi *Cypress*-a je: `npx cypress run`. Ova naredba će pokrenuti testove te postepeno prikazivati njihove rezultate kako se izvršavaju. Na kraju će dati detaljan izvještaj. Ova naredba se najčešće koristi u CI/CD okruženjima te kada nema potrebe za nikakvim dodatnim opcijama kod samog izvršavanja.

Na osnovnu naredbu moguće je dodavati argumente. Jedan od često korištenih argumenata je `--browser` argument koji definira u kojem pregledniku će se izvršavati testovi. Naredba korištenjem ovog argumenta izgleda: `npx cypress run --browser chrome`.

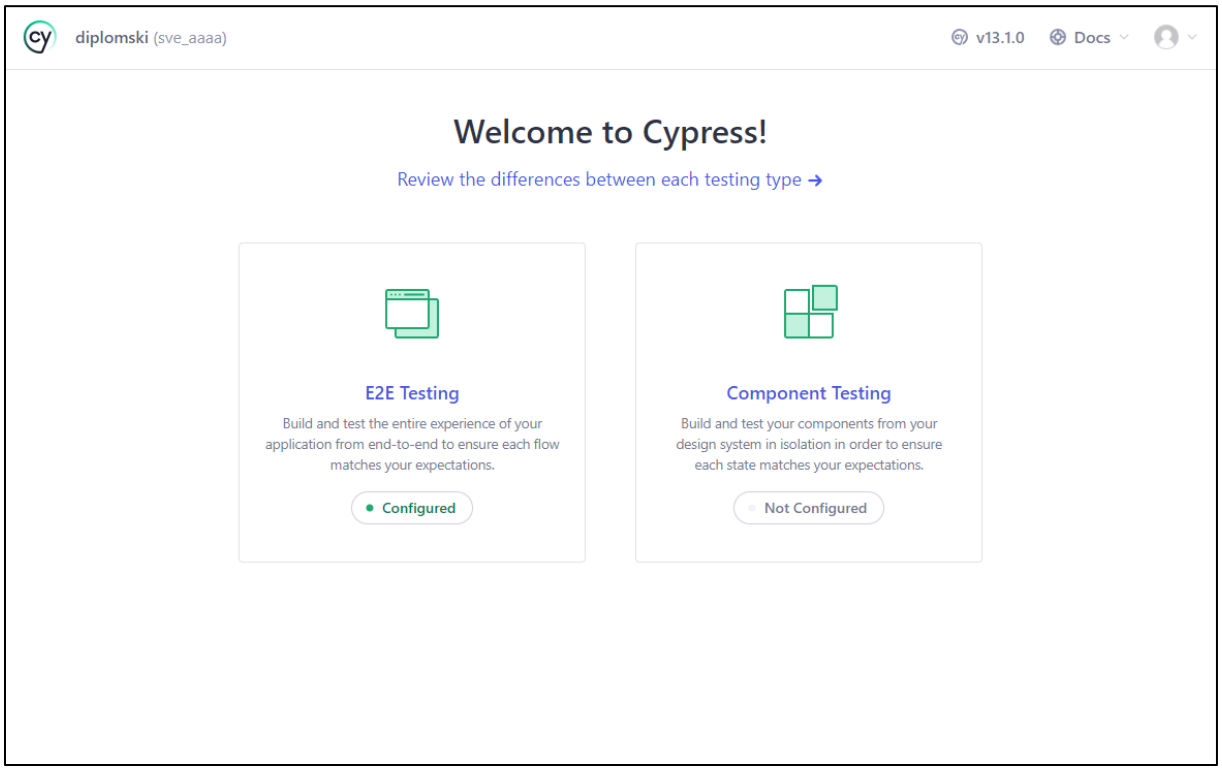
Osnovna naredba izvodi sve testne dokumente, a kako bi se izvršili testovi iz određenog testnog dokumenta, potrebno je koristiti `--spec` argument, te naredba izgleda: `npx cypress run --spec "cypress/e2e/prijava.cy.js"`.

Kako bi se vidio prikaz i simulacija korisnikovih akcija u pregledniku, moguće je koristiti naredbu `--headed`. Ova naredba pokrenuti će preglednik te sve testove izvršiti u pregledniku. Ova opcija se koristi kako bi se lakše pronašle i identificirale greške kod samog izvođenja. Uz naredbu `--headed`, često se koristi i naredba `--no-exit`, koja će *Cypress*-u reći da nakon izvršavanja testova ne zatvara preglednik kako bi se mogli pregledat potencijalni neispravni testovi. Naredba izgleda: `npx cypress run --spec "cypress/e2e/prijava.cy.js" --headed --no-exit`.

6.4.3.2. Pokretanje testova u korisničkom sučelju

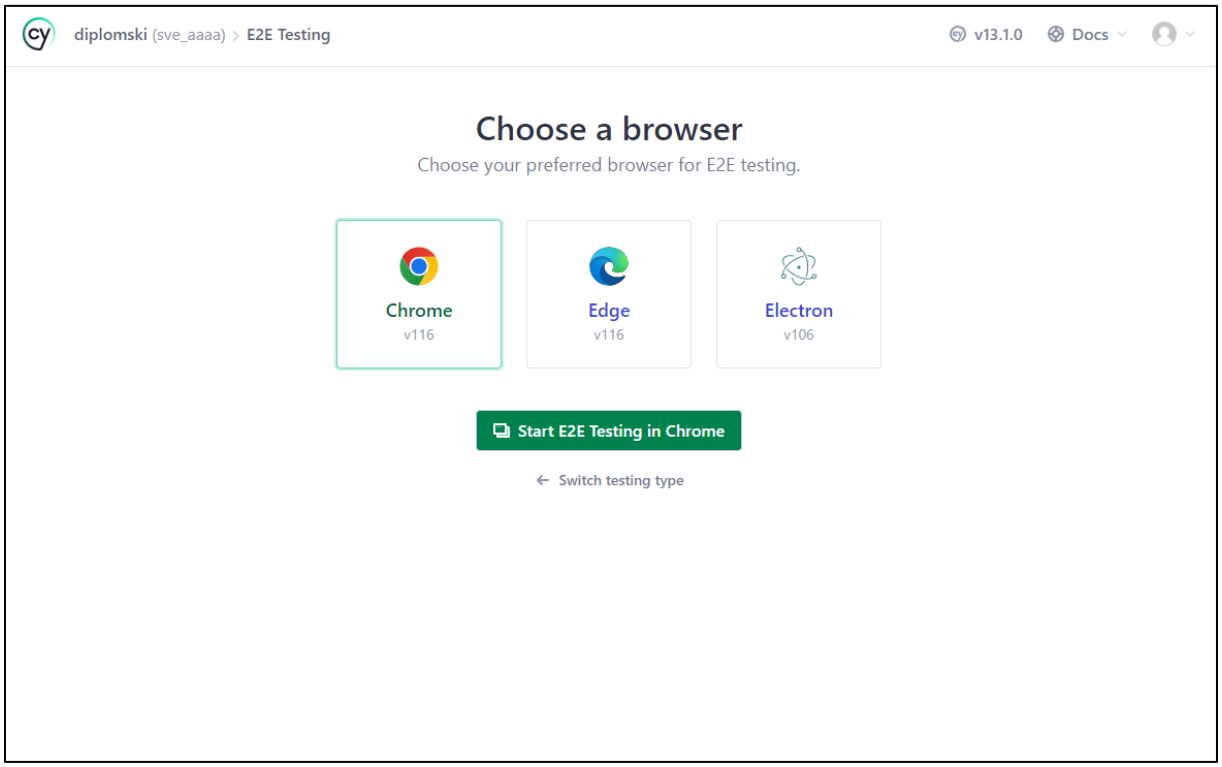
Češće korišteno tokom samog razvoja, pokretanje testova u korisničkom sučelju je izrazito korisna opcija *Cypress*-a. Ovo poglavlje opisuje korake podešavanja i pokretanja testova u korisničkom sučelju.

Za početak, potrebno je pokrenuti korisničko sučelje naredbom `npx cypress open`. Nakon toga korisniku se otvori prozor prikazan na slici 32. Korisnik treba odabrati opciju, u ovom slučaju *E2E Testing.*



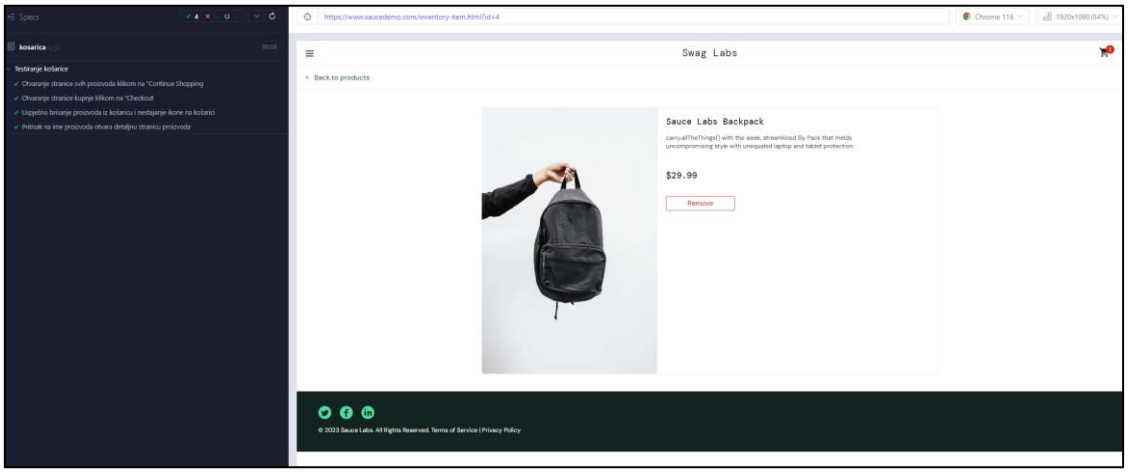
Slika 32: Odabir vrste testiranja (Izvor: (Cypress.io, 2023b))

Nakon odabira vrste testiranja, potrebno je odabrati preglednik. U ovom slučaju, odabran je preglednik Chrome, ali moguće je odabrati bilo koji preglednik koji je instaliran na uređaju na kojem se testovi pokreću. Prikazano na slici 33.



Slika 33: Odabir preglednika (Izvor: (Cypress.io, 2023b))

Nakon odabira preglednika, otvara se popis svih testnih dokumenata. Odabirom na bilo koji od njih pokrenuti će se prikaz testiranja i simulacija korisnikovih akcija na desnoj strani ekrana. Ova metoda uvelike je korištena u izradi testova za ovu aplikaciju. Slika 34 prikazuje ekran pokrenutih testova.



Slika 34: Prikaz pokrenutih testova (Izvor: Autor prema: (Cypress.io, 2023b; Saucedemo, 2023))

Velika prednost ovog pogleda je mogućnost proširenja bilo kojeg od testova te pregledavanje detaljnih koraka i akcija koje su napravljene od strane softvera. Prikaz na slici 35.

Slika 35: Detaljan prikaz testa (Izvor: Autor & (Cypress.io, 2023b))

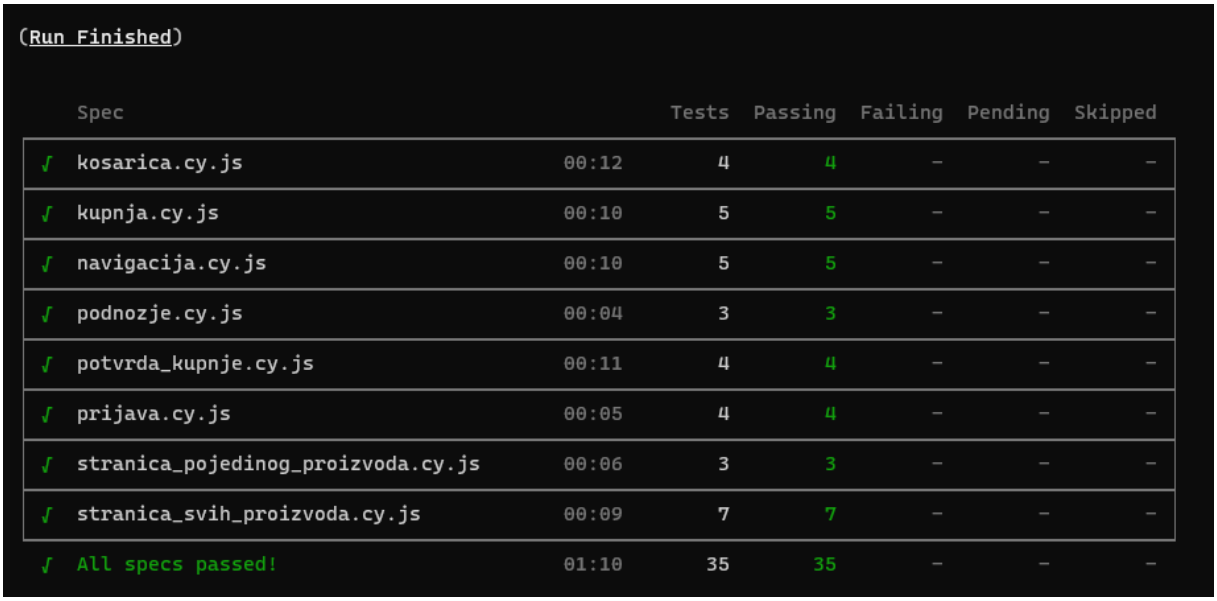
6.5. Izvještaji izvršenih testova

U svrhu izrade kvalitetne usporedbe s manualnim testiranjem, ovo poglavlje će prezentirati pokretanje testova za svaku od vrsti korisnika tri puta te usporediti rezultate vezane za brzinu izvedbe, točnost rezultata te količine korisnih informacija.

6.5.1. Izvođenje testnih scenarija standardnog korisnika

Na slikama ispod (36, 37, 38) prikazani su izvještaji pokretanja testova za standardnog korisnika. Na njima vidimo da su svi testovi uspješno izvršeni, te vrijeme izvođenja pojedinog testnog dokumenta i svih dokumenata zajedno. Prosječno izvođenje sva tri pokretanja je 1 minuta i 17 sekundi.

Bitno je napomenuti da na izvođenje utječe i brzina konekcije računala s kojeg se izvođenje obavlja. Sve poteškoće s vezom utječu na brzinu učitavanja elemenata na stranici, a samim time i na brzinu izvođenja.


(Run Finished)

	Spec		Tests	Passing	Failing	Pending	Skipped
✓	kosarica.cy.js	00:12	4	4	-	-	-
✓	kupnja.cy.js	00:10	5	5	-	-	-
✓	navigacija.cy.js	00:10	5	5	-	-	-
✓	podnozje.cy.js	00:04	3	3	-	-	-
✓	potvrda_kupnje.cy.js	00:11	4	4	-	-	-
✓	prijava.cy.js	00:05	4	4	-	-	-
✓	stranica_pojedinog_proizvoda.cy.js	00:06	3	3	-	-	-
✓	stranica_svih_proizvoda.cy.js	00:09	7	7	-	-	-
✓	All specs passed!	01:10	35	35	-	-	-

Slika 36: Izvještaj 1 standardnog korisnika (Izvor: Autor)

(Run Finished)

	Spec		Tests	Passing	Failing	Pending	Skipped
√	kosarica.cy.js	00:22	4	4	-	-	-
√	kupnja.cy.js	00:13	5	5	-	-	-
√	navigacija.cy.js	00:11	5	5	-	-	-
√	podnozje.cy.js	00:06	3	3	-	-	-
√	potvrda_kupnje.cy.js	00:11	4	4	-	-	-
√	prijava.cy.js	00:06	4	4	-	-	-
√	stranica_pojedinog_proizvoda.cy.js	00:05	3	3	-	-	-
√	stranica_svih_proizvoda.cy.js	00:11	7	7	-	-	-
√	All specs passed!	01:28	35	35	-	-	-

Slika 37: Izvještaj 2 standardnog korisnika (Izvor: Autor)

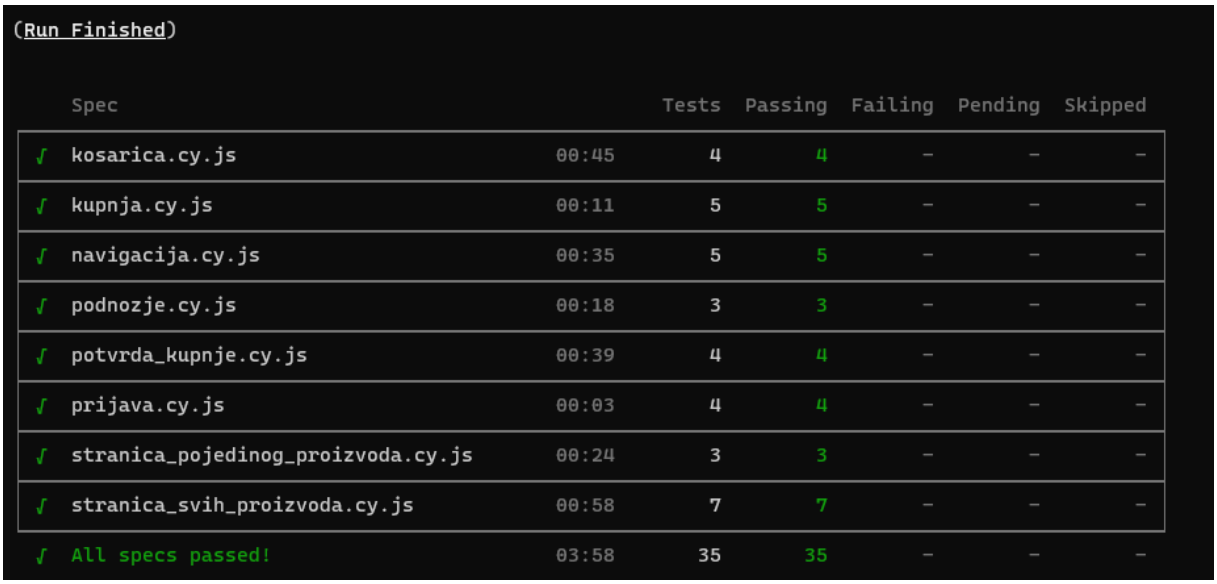
(Run Finished)

	Spec		Tests	Passing	Failing	Pending	Skipped
√	kosarica.cy.js	00:12	4	4	-	-	-
√	kupnja.cy.js	00:13	5	5	-	-	-
√	navigacija.cy.js	00:09	5	5	-	-	-
√	podnozje.cy.js	00:04	3	3	-	-	-
√	potvrda_kupnje.cy.js	00:11	4	4	-	-	-
√	prijava.cy.js	00:05	4	4	-	-	-
√	stranica_pojedinog_proizvoda.cy.js	00:05	3	3	-	-	-
√	stranica_svih_proizvoda.cy.js	00:12	7	7	-	-	-
√	All specs passed!	01:14	35	35	-	-	-

Slika 38: Izvještaj 3 standardnog korisnika (Izvor: Autor)

6.5.2. Izvođenje testnih scenarija korisničkog profila s poteškoćama s performansama

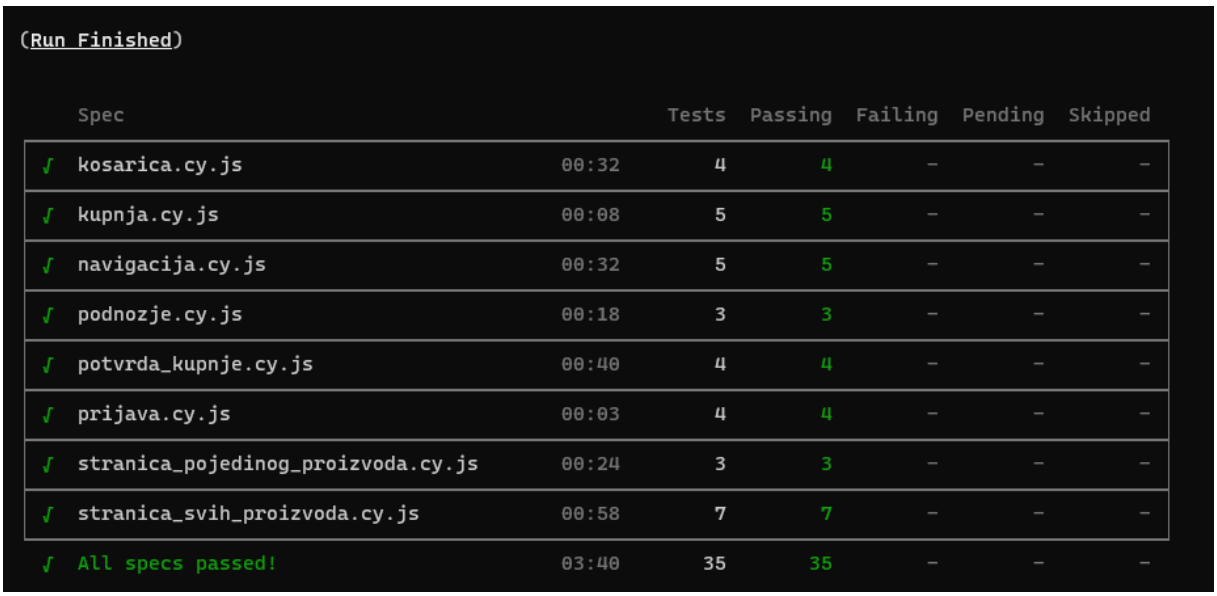
Na prvi pogled na slike 39, 40 te 41, svi testovi izgledaju uspješno izvedeni, ali usporedimo li vrijeme izvođenja s prethodnom vrstom korisnika, ovdje se radi o tri puta duljem prosječnom vremenu izvođenja. Prosječno vrijeme izvođenja ovih testova iznosi 3 minute i 45 sekundi.



(Run Finished)

Spec		Tests	Passing	Failing	Pending	Skipped
✓ kosarica.cy.js	00:45	4	4	-	-	-
✓ kupnja.cy.js	00:11	5	5	-	-	-
✓ navigacija.cy.js	00:35	5	5	-	-	-
✓ podnozje.cy.js	00:18	3	3	-	-	-
✓ potvrda_kupnje.cy.js	00:39	4	4	-	-	-
✓ prijava.cy.js	00:03	4	4	-	-	-
✓ stranica_pojedinog_proizvoda.cy.js	00:24	3	3	-	-	-
✓ stranica_svih_proizvoda.cy.js	00:58	7	7	-	-	-
✓ All specs passed!	03:58	35	35	-	-	-

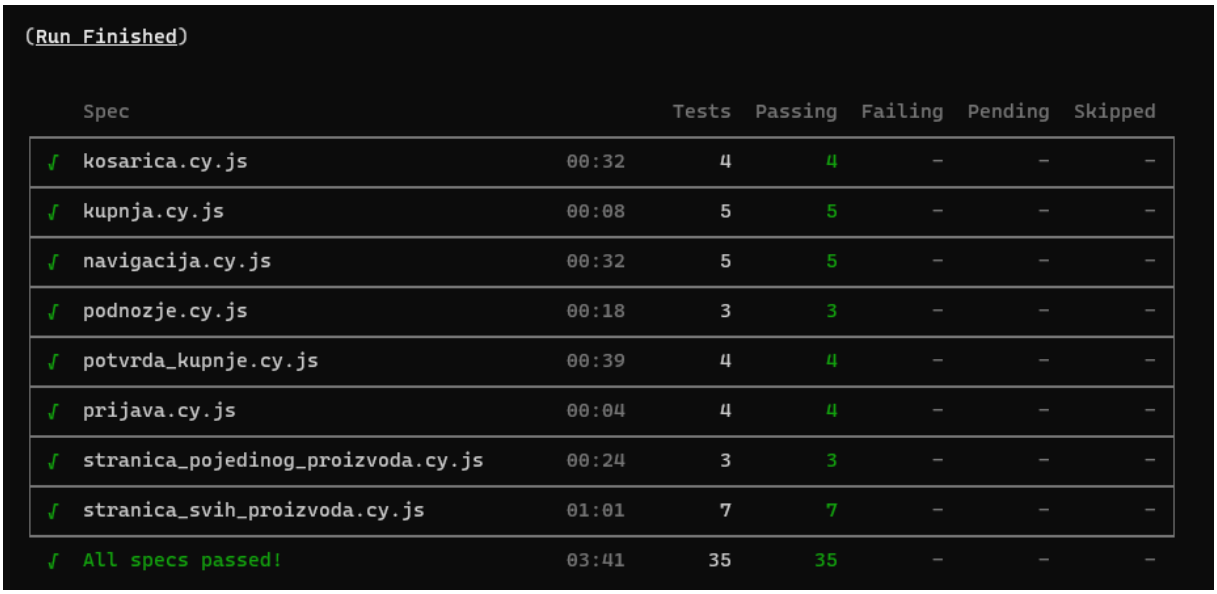
Slika 39: Izvještaj 1 korisničkog profila s poteškoćama s performansama (Izvor: Autor)



(Run Finished)

Spec		Tests	Passing	Failing	Pending	Skipped
✓ kosarica.cy.js	00:32	4	4	-	-	-
✓ kupnja.cy.js	00:08	5	5	-	-	-
✓ navigacija.cy.js	00:32	5	5	-	-	-
✓ podnozje.cy.js	00:18	3	3	-	-	-
✓ potvrda_kupnje.cy.js	00:40	4	4	-	-	-
✓ prijava.cy.js	00:03	4	4	-	-	-
✓ stranica_pojedinog_proizvoda.cy.js	00:24	3	3	-	-	-
✓ stranica_svih_proizvoda.cy.js	00:58	7	7	-	-	-
✓ All specs passed!	03:40	35	35	-	-	-

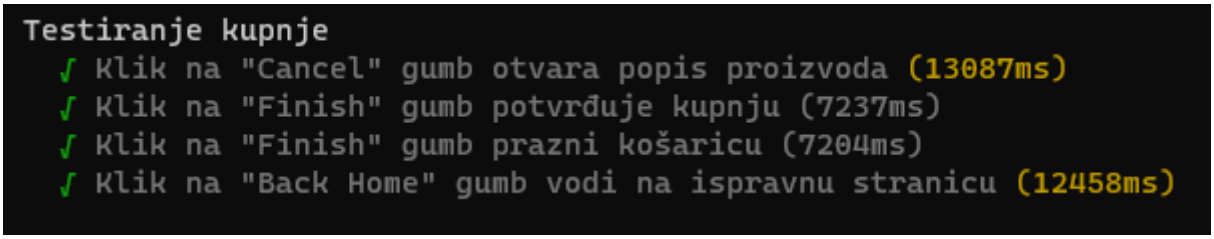
Slika 40: Izvještaj 2 korisničkog profila s poteškoćama s performansama (Izvor: Autor)



Slika 41: Izvještaj 3 korisničkog profila s poteškoćama s performansama (Izvor: Autor)

Ova informacija na prvu govori da postoje dvije opcije. Ili se radi o problemima s konekcijom testne aplikacije ili su greške u performansi. Za ovakve slučajeve potrebna je intervencija i ručno testiranje određenih dijelova aplikacije kako bi se utvrdio pravi izvor problema.

Cypress-ov prikaz izvođenja testova pomaže u ovom slučaju na način da testove koji se izvode sporo (dulje od 10s) označava žutom bojom, što naznačuje da postoji problem u brzini izvođenja testa. Vidljivo na slici ispod (slika 42)



Slika 42: Prikaz popisa testnih scenarija koji su sporiji (Izvor: Autor)

6.5.3. Izvođenje testnih scenarija profila korisnika s greškom u radu

Kao i kod manualnog testiranja, korisnik s greškama u radu je prikazao testove koji nisu uspješno izvršeni. Radi se o 8 testova koji nisu uspjeli (označeni crvenom bojom), te o 3 testa koji su preskočeni (označeni plavom bojom). Izvještaji su prikazani na slikama 43, 44 i 45

(Run Finished)

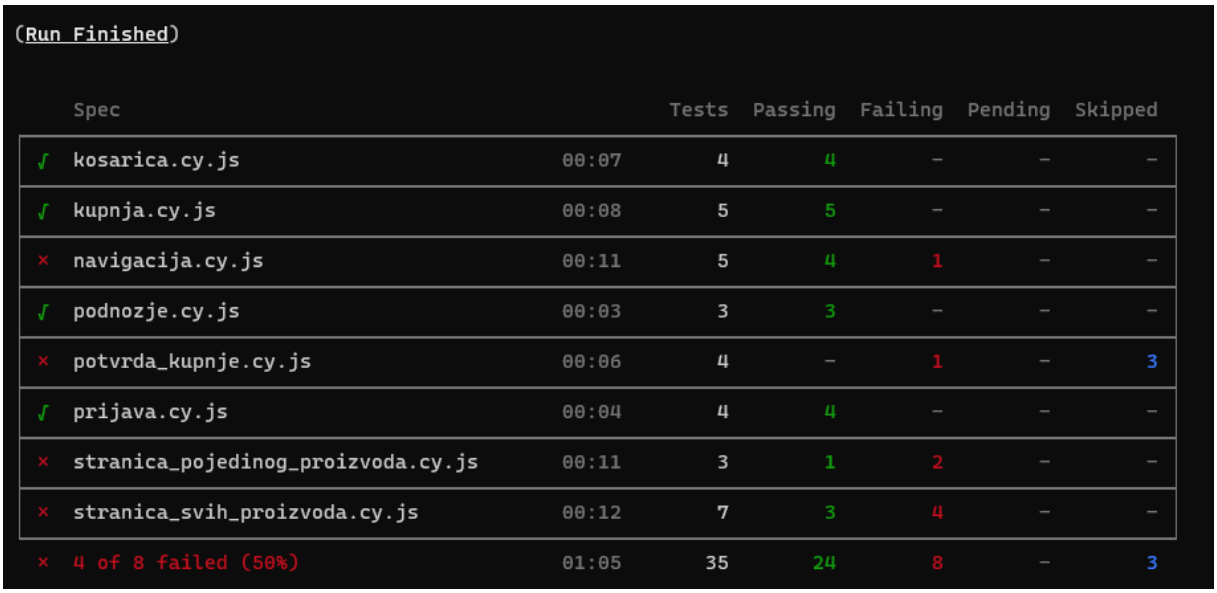
	Spec		Tests	Passing	Failing	Pending	Skipped
√	kosarica.cy.js	00:07	4	4	-	-	-
√	kupnja.cy.js	00:08	5	5	-	-	-
×	navigacija.cy.js	00:11	5	4	1	-	-
√	podnozje.cy.js	00:03	3	3	-	-	-
×	potvrda_kupnje.cy.js	00:06	4	-	1	-	3
√	prijava.cy.js	00:04	4	4	-	-	-
×	stranica_pojedinog_proizvoda.cy.js	00:12	3	1	2	-	-
×	stranica_svih_proizvoda.cy.js	00:13	7	3	4	-	-
×	4 of 8 failed (50%)	01:07	35	24	8	-	3

Slika 43: Izvještaj 1 profila korisnika s greškom u radu (Izvor: Autor)

(Run Finished)

	Spec		Tests	Passing	Failing	Pending	Skipped
√	kosarica.cy.js	00:07	4	4	-	-	-
√	kupnja.cy.js	00:08	5	5	-	-	-
×	navigacija.cy.js	00:10	5	4	1	-	-
√	podnozje.cy.js	00:03	3	3	-	-	-
×	potvrda_kupnje.cy.js	00:06	4	-	1	-	3
√	prijava.cy.js	00:04	4	4	-	-	-
×	stranica_pojedinog_proizvoda.cy.js	00:12	3	1	2	-	-
×	stranica_svih_proizvoda.cy.js	00:13	7	3	4	-	-
×	4 of 8 failed (50%)	01:06	35	24	8	-	3

Slika 44: Izvještaj 2 profila korisnika s greškom u radu (Izvor: Autor)



(Run Finished)

	Spec		Tests	Passing	Failing	Pending	Skipped
✓	kosarica.cy.js	00:07	4	4	-	-	-
✓	kupnja.cy.js	00:08	5	5	-	-	-
×	navigacija.cy.js	00:11	5	4	1	-	-
✓	podnozje.cy.js	00:03	3	3	-	-	-
×	potvrda_kupnje.cy.js	00:06	4	-	1	-	3
✓	prijava.cy.js	00:04	4	4	-	-	-
×	stranica_pojedinog_proizvoda.cy.js	00:11	3	1	2	-	-
×	stranica_svih_proizvoda.cy.js	00:12	7	3	4	-	-
×	4 of 8 failed (50%)	01:05	35	24	8	-	3

Slika 45: Izvještaj 3 profila korisnika s greškom u radu (Izvor: Autor)

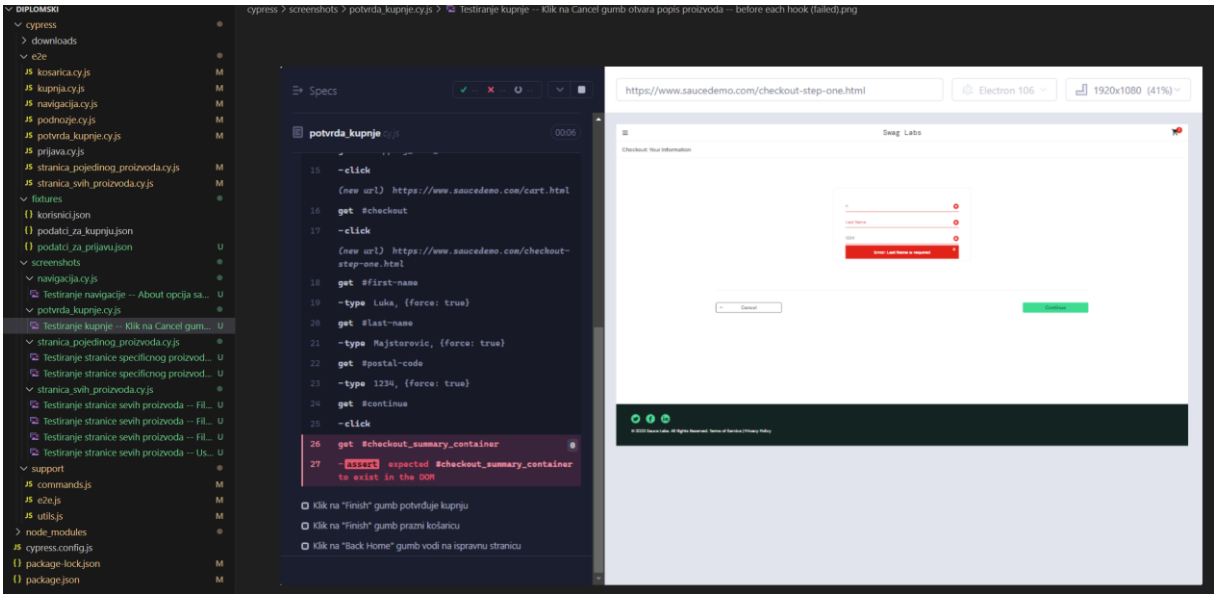
Kao i kod manualnog testiranja, korisnik s greškama u radu je prikazao testove koji nisu uspješno izvršeni. Radi se o 8 testova koji nisu uspjeli (označeni crvenom bojom), te o 3 testa koji su preskočeni (označeni plavom bojom).

Preskočeni testovi su nov pojam te do njih dolazi kada se u `beforeEach()` djelu koda dogodi greška. Razlog tomu je što ako samo postavljanje testa u određeno stanje ne radi, nema smisla izvršavati sam test. Iz tog razloga, *Cypress* automatski poništi izvršavanje drugih testova koji su u grupaciji. *Cypress* također ispiše detaljne informacije o ovom problemu, kao što je prikazano na slici 46.

```
Because this error occurred during a `before each` hook we are skipping the remaining tests in the current suite: `Testiranje kupnje`
    at kupnja (webpack:///./cypress/support/utils.js:35:44)
    at Context.eval (webpack:///./cypress/e2e/potvrda_kupnje.cy.js:10:11)
```

Slika 46: Prikaz greške izvođenja testova (Izvor: Autor)

U primjeru prikazanom ispod, nakon kupovine proizvoda te unosa podataka za kupovinu, pritisak na gumb *Continue* korisnika odjavi umjesto da ga odvede na sljedeći ekran. To je vidljivo u slici 47. Uz to, vidljiv je način prikazivanja neuspjelih testova i njihovih slika ekrana.



Slika 47: Prikaz greške nakon izvođenja testiranja (Izvor: Autor)

7. Usporedba automatskog i manualnog testiranja

Nakon obavljenog planiranja, kreiranja i izvršavanja manualnog testiranja te planiranja, pisanja te izvođenja automatiziranih testova, potrebno je obije vrste testiranja staviti u kontekst te ih usporediti.

Naredna dva poglavlja za svrhu imaju odrediti prednosti i mane obije vrste testiranja te objasniti logiku iza njihove prioritizacije.

7.1. Prednosti i mane

Kao i kod svakog odabira, odnosno prioritizacije, potrebno je obratiti pozornost na prednosti i mane izbora.

U nastavku ovog poglavlja pokrivene su prednosti i mane manualnog i automatskog testiranja.

7.1.1. Prednosti manualnog testiranja

Jedna od najvećih prednosti manualnog testiranja je prilagodljivost testera na promjene. Kada dođe do drastične promjene korisničkih zahtjeva, vrijeme potrebno testeru da se prilagodi i osposobi za testiranje promjena je izrazito kraće od vremena potrebnog da se te iste promjene prilagode za automatizaciju.

Mogućnost testiranja nefunkcionalnih zahtjeva je još jedna velika prednost manualnog testiranja. Izgleda aplikacije, korisničko iskustvo, brzina učitavanja te svi ostali aspekti koje automatizirani testovi ne mogu pokriti, manualno testiranje obavlja odlično i jedna je njegove najveće prednosti.

Testiranje u ranim fazama najčešće je nemoguće automatskim testovima iz razloga što oni zahtijevaju kompletnu strukturu projekta te komunikaciju između dijelova aplikacije kako bi bili kvalitetno izvedeni. Manualno testiranje nema problema s time. Kada god je proizveden funkcionalan dio aplikacije, testeri mogu testirati taj dio bez potrebe za čekanjem ostatka aplikacije.

Ljudski faktor, definitivno najveća prednost manualnog testiranja i ono što će se pokazati kao činjenica koja ga čini nezaobilaznim. Ljudski faktor te procjene koje tester može napraviti tokom testiranja ne mogu biti zamijenjeni automatiziranim testovima.

7.1.2. Mane manualnog testiranja

Količina posla koja je potrebna, pogotovo na većim projektima, jedna je od najvećih mana manualnog testiranja. Količina sati koja će biti utrošena kako bi se iznova provjerile određene funkcionalnosti je izrazito velika te je mana manualnog testiranja.

Ljudski faktor, isto kao i prednost, predstavlja i problem gdje se uvođenjem ljudskog faktora dovodi prostor greškama. Ovaj faktor se uvelike izražava kod aplikacija s velikom količinom testnih scenarija, malom količinom osoblja te potrebitim čestim ponovnim testiranjima.

7.1.3. Prednosti automatiziranog testiranja

Efikasnost i preciznost najveće su prednosti automatskog testiranja. Automatski testovi su konzistentni, te jednom kada su ispravno napisani i pokrivaju potrebne slučajeve korištenja, nepogrješiv su izvor istine o ispravnosti rada koda.

Kod dugih regresijskih testiranja, manualni posao postaje pre velik te se otvara prostor za greške i propuste. Jednom postavljen projekt s potrebnim testovima za regresiju, tih problema nema. Testovi se pokrenu i tumači se testni izvještaj. Ovo je još jedna velika prednost automatskog testiranja.

Ukoliko su testovi ispravno pisani i praćena su pravila pisanja automatizacije, paralelizacija je velika prednost. Dok jedna osoba može izvršavati jedan testni scenarij u jednom trenutku, automatizacija može izvršavati nekoliko. Ovime se postiže još veća ušteda vremena.

7.1.4. Mane automatiziranog testiranja

Inicijalno postavljanje definitivno je jedna od najvećih mana automatskog testiranja. Iako efikasni, inicijalno postavljanje mora biti odrađeno savršeno te uz minimalno propusta kako bi testovi bili efikasni. Uz to, inicijalno postavljanje testova zahtjeva veliku količinu vremena te ljudskih resursa.

Održavanje automatskih testova potencijalno je veliki problem ukoliko se radi o loše napisanim testovima. Na ovaj način se izrazito isplativi automatski testovi čine izrazito neisplativim, kada god je riječ o promjeni.

Ukoliko projekt ili aplikacija koja je održavana često prolazi kroz promjene korisničkog sučelja, automatsko testiranje loš je izbor. Svaka promjena korisničkog sučelja zahtijevat će velike promjene u samim testovima.

7.2. Prioritizacija

Prioritizacija testiranja jedno je od glavnih pitanja u toku samog planiranja projekta razvoja programskog proizvoda.

Bez obzira na prednosti i mane, manualno testiranje je gotovo uvijek odabrano kao inicijalan pristup testiranju. Razlog tome je laka prilagodljivost na promjene te mogućnost testiranja u ranim fazama. Naravno, ovo pravilo ima iznimke, te ovisno o prioritetima razvojnog tima i samog projekta prioritet testiranja se mijenja.

U odmaklim fazama razvoja te u fazi održavanja, dolazi do pitanja uvođenja automatizacije. Iako zvuči kao logičan korak za fazu održavanja, prije same odluke potrebno je postaviti nekoliko pitanja poput:

1. Koliko dugo će se proizvod održavati?
2. Koliko su česte promjene na proizvodu?
3. Koliko su česte promjene korisničkog sučelja?

Odgovori na ova tri pitanja iznad će dati puno bolje informacije i čišću sliku je li potrebno prebaciti fokus na automatizaciju testiranja. Ukoliko se radi o proizvodu koje će se održavati kratko vrijeme ili možda uopće neće biti u održavanju, automatizacija nema svrhu niti isplativost. Ako se u suprotnom radi o proizvodu koji je u održavanju dulje period, automatizacija je i više nego dobro došla.

Količina promjena uvelike igra ulogu u prioritizaciji automatizacije. Ukoliko će proizvod u svojoj fazi održavana biti minimalno mijenjan, automatizacija možda i nije isplativa te je bolje zadržati se isključivo na manualnom testiranju.

Kao što je spomenuto u manama automatizacije, česte promjene korisničkog sučelja mogu biti fatalne za automatizaciju, te je isplativije zadržati se na manualnom testiranju u takvim slučajevima.

Pitanje prioritizacije testiranja uključuje mnogo faktora koji obuhvaćaju sve članove razvojnog tima i njihovu ekspertizu te sam projekt i prioritete projekta. Bitno je napomenuti da ispravno odabrana prioritizacija testova čini razliku između lako održivog proizvoda koji je konstantno na najvišoj razini kvalitete te proizvoda koji ima problema s održavanjem kvalitete.

8. Zaključak

U ovom radu pokrivena je tema manualnog i automatskog testiranja korisničkih sučelja. Na samom početku rada, objašnjeni su glavni pojmovi i dane informacije vezane za web aplikacije u svrhu shvaćanja ostatka rada.

Kroz teoretski dio prikazan je način pisanja te izvođenja manualnog testiranja. Čitatelju su prikazani glavni koncepti i načini, te primjeri pisanja i izvođenja manualnog testiranja. Zatim je u detalje opisan koncept automatiziranog testiranja, njegova pravila i glavne vodilje u pisanju. Kroz primjere isječaka koda prikazane su dobre prakse i objašnjeni koncepti.

Praktični dio prikazao je proces manualnog i automatiziranog testiranja sučelja web aplikacije. Provedeno je detaljno planiranje i kreiranje manualnog testiranja te izvođenje istog. Zatim su kreirani automatizirani testovi po uzoru na testne scenarije manualnog testiranja. Kreirani su i predstavljeni izvještaji, statistike te zaključci izvođenja testiranja.

Na samom kraju, postavljeno je pitanje usporedbe i prioritizacije manualnog i automatiziranog testiranja. Kao zaključak, donesena je činjenica da svaka od vrsta testiranja ima svoje prednosti i mane, te ovisno o projektu, prioritetima te načinu funkcioniranja tima treba uložiti veliki trud kako bi se ispravno prioritizirale vrste testiranja.

Popis literature

Anton Hristov. (2022). *Automated testing*. https://www.atlassian.com/devops/devops-tools/test-automation

Cypress.io. (2023a). *Why Cypress?* https://docs.cypress.io/guides/overview/why-cypress

Cypress.io. (2023b). *Cypress.io*. https://www.cypress.io/

GeeksForGeeks. (2022, siječanj 27). *Difference Between Web application and Website*. https://www.geeksforgeeks.org/difference-between-web-application-and-website/

guru99. (2023a). *Integration testing*. https://www.guru99.com/integration-testing.html

guru99. (2023b). *Manual Testing Tutorial*. https://www.guru99.com/manual-testing.html

guru99. (2023c). *Test Summary Reports*. https://www.guru99.com/how-test-reports-predict-the-success-of-your-testing-project.html

guru99. (2023d). *Unit testing*. https://www.guru99.com/unit-testing-guide.html

guru99. (2023e). *What is BLACK Box Testing*. https://www.guru99.com/black-box-testing.html

guru99. (2023f). *What is system testing?* https://www.guru99.com/system-testing.html

guru99. (2023g). *What is User Acceptance Testing?* https://www.guru99.com/user-acceptance-testing.html

guru99. (2023h). *White Box Testing*. https://www.guru99.com/white-box-testing.html

Lucia Cavero-Baptista. (bez dat.). *Low-code vs No-code Automation*. Preuzeto 03. rujan 2023., od https://www.leapwork.com/blog/low-code-vs-no-code-automation

MDM contributors, & Mozilla. (2023, lipanj 8). *JavaScript*.

Oren Rubin. (2017, rujan 19). *DevOps*. https://devops.com/10-rules-for-writing-automated-tests/

Paul C. Jorgensen. (2014). *Software Testing* (4. izd.). CRC Press.

Saucedemo. (2023). *Swag Labs*. https://www.saucedemo.com/

StackPath. (bez dat.). *What is a Web Application?* Preuzeto 03. rujan 2023., od https://www.stackpath.com/edge-academy/what-is-a-web-application/

Popis slika