Automatizirano testiranje mrežnih mjesta koristeći radnu okolinu Playwright

Žilić, Marija

Master's thesis / Diplomski rad

2024

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:142:723984

Sveučilište J.J. Strossmayera u Osijeku
Filozofski fakultet Osijek
Dvopredmetni diplomski studij Informatologije i Informacijske tehnologije

Marija Žilić

Automatizirano testiranje mrežnih mjesta koristeći radnu okolinu Playwright

Diplomski rad

Mentor: izv. prof. dr. sc. Tomislav Jakopec

Osijek, 2024.

Sveučilište J.J. Strossmayera u Osijeku
Filozofski fakultet Osijek
Odsjek za informacijske znanosti
Dvopredmetni diplomski studij Informatologije i Informacijske tehnologije

Marija Žilić

Automatizirano testiranje mrežnih mjesta koristeći radnu okolinu Playwright

Diplomski rad

Društvene znanosti, Informacijske i komunikacijske znanosti, Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tomislav Jakopec

Osijek, 2024.

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

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

IZJAVA

Izjavljujem s punom materijalnom i moralnom odgovornošću da sam ovaj rad samostalno napisao/napisala te da u njemu nema kopiranih ili prepisanih dijelova teksta tuđih radova, a da nisu označeni kao citati s navođenjem izvora odakle su preneseni.

Svojim vlastoručnim potpisom potvrđujem da sam suglasan/suglasna da Filozofski fakultet u Osijeku trajno pohrani i javno objavi ovaj moj rad u internetskoj bazi završnih i diplomskih radova knjižnice Filozofskog fakulteta u Osijeku, knjižnice Sveučilišta Josipa Jurja Strossmayera u Osijeku i Nacionalne i sveučilišne knjižnice u Zagrebu.

U Osijeku 22.08.2024

Marija Žilić, 0122225728

Ime i prezime studenta, JMBAG

ZAHVALA

Iskreno zahvaljujem mentoru izv. prof. dr. sc. Tomislavu Jakopecu na trudu, strpljenju i neizmjernoj pomoći tijekom pisanja ovog rada.

Jedno veliko hvala komentorici Mariji Holjevac, mag. inf. et mag inf. et comm. na iznimnoj pomoći i još većoj podršci. Njena prisutnost, pomoć i savjeti su bili vjetar u leđa tijekom cijelog procesa pisanja rada.

Hvala i mojoj obitelji i prijateljima na pitanju “Kad će diplomski?” - ovaj rad je i za vas.

Sažetak

Ovaj rad bavi se automatskim testiranjem mrežnog mjesta u Playwright radnom okviru. Svrha ovog rada je prikazati kako automatsko testiranje pomoću Playwright-a može poboljšati proces testiranja mrežnih aplikacija, s fokusom na testiranje cijelog toka aplikacije, od početka do kraja (eng. end-to-end testing). U teorijskom dijelu rada prikazane su osnove testiranja programskih rješenja, pregled ključnih pojmova u testiranju programskih rješenja (test plan, testni slučaj i sl.), vrste, razine i metode testiranja te procesi testiranja. Također, u radu je opisano i kako napisati izvješće o pogrešci radi cjelokupnog pregleda procesa testiranja. Nadalje, rad nastoji objasniti što je automatsko testiranje te alate koji se koriste za automatsko testiranje. Također, nastoji ukazati na važnost automatskog testiranja, ali i prednosti i potencijalne rizike uvođenja automatizacije te povezanost automatskog testiranja sa regresijskim testiranjem. Osim toga, u radu je opisan Playwright radni okvir koji podržava automatsko testiranje različitih mrežnih preglednika te pruža podršku za različite programske jezike. Playwright ima ugrađene razne mogućnosti kao što su izvješća o testiranju, paralelno testiranje više preglednika i podršku za otklanjanje pogrešaka. U istraživačkom dijelu, odabrana je Demo Web Shop mrežna aplikacija koja imitira funkcionalnosti online trgovine te ju se može koristiti za testiranje raznih funkcionalnosti kao što je prijava i registracija korisnika, pregled proizvoda i dodavanje stavki u košaricu te obavljanje kupovine bez stvarnih transakcija. Istraživački dio rada obuhvaća 35 testnih slučajeva implementiranih na odabranoj mrežnoj aplikaciji, uz prikaz ključnih Playwright-ovih značajki potrebnih za implementaciju testnih slučajeva. Također, u radu je opisano kako se testni slučajevi pokreću te prikaz izvješća o rezultatima testiranja.

Ključne riječi: Playwright, Automatsko testiranje, Mrežna aplikacija, Demo Web Shop

Sadržaj

1. Uvod

Ubrzani razvoj IT industrije doveo je do potrebe za pronalaskom bržeg i efikasnijeg načina testiranja programskih rješenja. Sve više firmi počinje ulagati u automatizaciju testnih slučajeva kako bi mogli držati korak s konkurencijom. Automatsko testiranje je vrsta testiranja programskih rješenja koja se izvodi korištenjem posebnih alata za izvođenje testnih slučajeva.[1] Uzastopni razvojni ciklusi zahtijevaju ponavljanje istih testnih slučajeva kako bi se osigurala bolja kvaliteta proizvoda, stoga testerima, kao i ostalim dionicima, alati za automatizaciju omogućavaju jednostavno i brzo ponavljanje testnih cjelina koje mogu reproducirati po potrebi.[2]

Teorijski dio rada obuhvaća pregled osnova testiranja programskih rješenja, procese testiranja, vrste, razine i metode te razmatranje automatskog testiranja u Playwright radnom okviru. Playwright je radni okvir otvorenog koda koji služi za automatsko testiranje mrežnih aplikacija, s fokusom na testiranje cijelog toka aplikacije od početka do kraja (eng. end-to-end testing). Playwright pozdržava automatsko testiranje različitih mrežnih preglednika kao što su Chromium, Webkit i Firefox te pruža podršku za više programskih jezika (Java, C, Python, Javascript / Typescript). Osim toga, pruža i razne mogućnosti poput ugrađenih izvješća o testiranju, paralelno testiranje više preglednika, podršku za otklanjanje pogrešaka (eng. Debugging Tools Support) itd.[3] U istraživačkom dijelu rada, automatsko testiranje izvršeno je na Demo Web Shop mrežnom mjestu. Demo Web Shop je mrežna aplikacija koja imitira funkcionalnosti online trgovine te ju testeri i programeri mogu koristiti kao testno okruženje za razvoj i testiranje raznih programskih rješenja. Aplikacija dakle uključuje razne funkcionalnosti koje jedna online trgovina obuhvaća, poput pregleda proizvoda, dodavanja proizvoda u košaricu, proces kupovine i sl., ali bez ostvarivanja stvarnih financijskih troškova. Istraživački dio rada sastoji se od 35 testnih slučajeva, uz prikaz svih koraka provedenih tijekom automatskog testiranja te opis ključnih značajki korištenja Playwright okvira.

[1] Hamilton, Thomas. Automation Testing. URL: https://www.guru99.com/automation-testing.html (2024-06-09)
[2] Isto.
[3] BrowserStack; Playwright Automation Framework: Tutorial. 2023. URL:https://www.browserstack.com/guide/playwright-tutorial (2024-06-06)

2. Osnove testiranja sustava

Sustav koji je nepouzdan i neupotrebljiv rezultira gubitkom vremena, novca i može uništiti ugled tvrtke.[4] Veliku ulogu u osiguravanju kvalitete sustava i smanjenju rizika ima tester koji otkriva greške sustava tijekom rane, a ponekad i kasne faze razvoja proizvoda. Testiranje se često shvaća kao izvršavanje testnih slučajeva i provjera njihovih rezultata. Međutim, testiranje obuhvaća puno više aktivnosti koje su različito organizirane u skladu sa životnim ciklusom proizvoda.[5] Iako su aktivnosti testera najčešće regulirane između klijenta i proizvođača, mogu se temeljiti i na internim smjernicama tvrtke. Uz osnovne aktivnosti, one obično uključuju i planiranje testova, analizu te dizajn i implementaciju testnih slučajeva. Dodatne aktivnosti mogu uključivati i pisanje izvješća o rezultatima testiranja te analizu rizika.[6]

Testiranjem se provjerava ispunjava li sustav svoje zahtjeve, korisničke priče i specifikacije zahtjeva, kao i osigurava li proizvod želje i očekivanja korisnika u stvarnom okruženju. Zadatak testera je provjeriti i je li moguće sustav koristiti kako je namijenjen i otkriti ispunjava li svoju svrhu.[7] Važno je istaknuti da ne postoji sustav bez pogrešaka te da je malo vjerojatno da će se to promijeniti za bilo koji sustav koji prelazi određeni stupanj složenosti.[8] Mnoge pogreške nastaju uslijed neuspješnog identificiranja i testiranja iznimaka. Stoga je ponekad neizbježno da sustav funkcionira unatoč prisutnosti pogrešaka u određenim kombinacijama ulaznih podataka. Bez obzira na to koliko puta testovi pokažu da ne postoje pogreške u sustavu, tester ne može biti u potpunosti siguran da dodatni testovi neće otkriti ranije neotkrivene pogreške te zbog toga testiranjem nije moguće dokazati potpunu odsutnost pogrešaka.[9]

Osnovno polazište za testiranje i prepoznavanje pogrešaka je definirati koje je očekivano ponašanje sustava. No da bi tester znao koje je očekivano ponašanje, mora poznavati zahtjeve sustava koji testira te druge dodatne informacije poput zahtjeva specifikacija, korisničkih priča i sl.[10] Te informacije čine osnovu za testiranje prema kojoj se provode testovi i odlučuje je li

[4] Spillner, Andreas; Linz, Tito. Software Testing Foundations; A Study Guide for the Certified Tester Exam - Foundation Level & ISTQB® Compliant. 5th, revised and updated Edition. Heidelberg, Germany: dpunkt.verlag, 2021. Str. 7.
[5] Isto, str. 8.
[6] Isto.
[7] Isto.
[8] Isto, str. 9.
[9] Isto.
[10] Isto.

određena funkcionalnost (ne)ispravna. Ukoliko tester otkrije grešku, ona se tretira kao neuspjeh unaprijed definiranog zahtjeva, odnosno odstupanje od očekivanog ponašanja.[11]

Pogreške se događaju iz različitih razloga, a neki od najučestalijih uzroka su: ljudska pogreška, vremenski pritisak, složenost zadatka, nesporazum i šum u komunikaciji između sudionika u razvoju projekta, složenost tehnologije koja se koristi ili korištenje nove tehnologije koja je nepoznata sudionicima projekta, te nedostatak iskustva ili obučavanja sudionika projekta.[12] U konačnici, otkrivanje pogrešaka i proučavanje njenih uzroka može poučiti izbjegavanju istih ili sličnih pogrešaka u budućnosti. Znanje koje se stječe na ovaj način pomaže u optimizaciji procesa i smanjivanju/sprječavanju dodatnih pogrešaka.[13]

Nadalje, proces pronalaženja i ispravljanja pogrešaka u sustavu naziva se debugiranje i odgovornost je programera. Potrebno je naglasiti da pojmovi debugiranje i testiranje često stvaraju konfuziju. Naime, debugiranje podrazumijeva otkrivanje pogrešaka u kodu, a testiranje otkriva učinke koje je ta pogreška uzrokovala.[14] Pod uvjetom da ispravljanje pogreške ne uzrokuje dodatne, nove pogreške na drugom mjestu, ono poboljšava kvalitetu proizvoda. Testeri su odgovorni provjeriti je li greška uspješno uklonjena, te se takvo testiranje naziva testovima potvrde (eng. *confirmation testing*).[15] U stvarnim situacijama, ispravljanje jedne pogreške vrlo često može uzrokovati stvaranje druge pogreške na drugačijem mjestu, a takve se pogreške mogu otkriti tek testiranjem novih scenarija. Drugim riječima, nije dovoljno samo testirati je li pogreška ispravljena, nego je potrebno i ponoviti prethodne testove i napisati nove testove kako bi bili sigurni da je greška ispravljena, ali i da ista nije uzrokovala nove pogreške.[16]

2.1. Pregled pojmova u testiranju sustava

Dvije su osnovne vrste testova: dinamički i statički testovi. Dinamički testovi su vrsta testova koja se koristi u svrhu testiranja dinamičkog ponašanja programskog koda. Drugačije rečeno, dinamički testovi način su izvođenja testova s dinamičkim varijablama i varijablama koje nisu

[11] Spillner, Andreas; Linz, Tito. Nav. dj., str. 9.
[12] Isto, str. 10.
[13] Isto, str. 11..
[14] Isto, str. 12.
[15] Isto.
[16] Isto.

konstantne, a testovi se mogu izvoditi tek ukoliko je je kod izvršen.[17] Osim dinamičkih testova (npr. izvođenje na računalu), statički testovi (testiranje dokumentacije o specifikaciji zahtjeva, korisničkih priča (eng. *user stories*) i izvornog koda također imaju ulogu u prepoznavanju grešaka u ranoj fazi razvoja i ispravljanju istih kako bi se osigurao bolji razvojni proces.[18]

Statički i dinamički testovi osmišljeni su za postizanje nekih od sljedećih ciljeva:

- Kvalitativna procjena proizvoda u skladu sa zahtjevima specifikacije, korisničkim pričama, dizajnom i kodom,
- Dokazivanje da su svi specifični zahtjevi implementirani te da testni objekt funkcionira u skladu sa očekivanjima korisnika i dionika,
- Pružanje informacija koje dionicima omogućuju čvrstu procjenu kvalitete te stvaranje povjerenja u pruženu kvalitetu,
- Smanjenje razine rizika uočavanjem i ispravljanjem pogrešaka,
- Analiza proizvoda i njegove dokumentacije kako bi se izbjegle neželjene pogreške te dokumentirale i otklonile poznate,
- Dokazivanje da je predmet testiranja u skladu sa ugovornim, zakonskim i regulatornim zahtjevima i standardima.[19]

Za pravilno izvođenje dinamičkih testova, potrebno je poznavati sljedeće:

Baza testiranja (eng. *test basis*)

Baza (osnova) za testiranje sastoji se od sve potrebne dokumentacije koja omogućava procjenu postoji li pogreška u sustavu ili ne. Ta dokumentacijska baza (koja obuhvaća specifikaciju programskih zahtjeva, korisničke priče i sl.) definira očekivano ponašanje objekta testiranja. Osim toga, zdrav razum i stručno znanje također se mogu smatrati dijelom testne baze i koristiti za donošenje odluka.[20] Specifikacija programskih zahtjeva podrazumijeva detaljan opis svrhe, značajki, funkcionalnosti i drugih elemenata programske aplikacije. Ona je vodič i temeljna

[17] Hamilton, Thomas. What is Dynamic Testing? Types, Techniques & Example. URL: https://www.guru99.com/dynamic-testing.html (2024-06-09)
[18] Spillner, Andreas; Linz, Tito. Nav. dj., str 8.
[19] Isto, str. 13.
[20] Isto, str. 14.

referenca koju razvojni tim treba slijediti tijekom razvoja sustava.[21] Korisničke priče (eng. User stories) su neformalan i općeniti opis značajki sustava iz perspektive krajnjeg korisnika. Svrha korisničkih priča je opisati kako će značajke sustava pružiti vrijednost korisniku. U korisničkim pričama koristi se netehnički jezik te one pomažu timu u razumijevanju konteksta što se gradi, zašto i koju vrijednost će to stvoriti kod korisnika.[22]

Testni slučajevi i izvođenje testova (eng. *test cases and test runs*)

Testni slučaj odnosi se na radnje koje su potrebne za provjeru određene značajke ili funkcionalnosti tijekom testiranja programskih rješenja. Testni slučaj detaljno opisuje korake, očekivane rezultate i preduvjete koje su potrebne za testiranje funkcionalnosti.[23] Uz pomoć osnova za testiranje, definiraju se testni slučajevi, a izvođenje testova se izvršava tijekom dodavanja odgovarajućih testnih podataka testnom objektu te izvršavanjem testnih slučajeva na računalu. Rezultati testnih slučajeva ukazuju postoji li u sustavu pogreška odnosno odstupanje između očekivanog i stvarnog ponašanja.[24]

Uvjeti testiranja (eng. *test conditions*)

Uvjeti testiranja podrazumijevaju specifikaciju koju tester mora slijediti tijekom testiranja.[25] Oni predstavljaju određena ograničenja i izvode se iz testne baze kako bi se postigli specifični ciljevi.[26] Testni uvjet može biti bilo što što tester želi provjeriti i može ga se provjeriti jednim ili više testnih slučajeva.

Pojedinačni testovi ne mogu se koristiti za testiranje cijele baze, nego se fokusiraju na određeni aspekt sustava. Drugim riječima, potrebno je kreirati pojedinačne, specifične testove koji su izvučeni iz baze testova kako bi se postigli specifični ciljevi testiranja.[27]

---

[21] IEEE; Software Requirements Specifications: Building a Blueprint for Success: Navigating the Complexities of Software Requirements. URL: https://www.computer.org/resources/software-requirements-specifications (2024-08-12)
[22] Rehkopf, Max. User stories with examples and a template. URL: https://www.atlassian.com/agile/project-management/user-stories (2024-08-12)
[23] Bose, Shreya. How to write Test Cases in Software Testing? (with Format & Example), 2024. https://www.browserstack.com/guide/how-to-write-test-cases (2024-08-12)
[24] Spillner, Andreas; Linz, Tito. Nav. dj., str. 14-15.
[25] Hamilton, Thomas. Test Condition vs Test Scenario in Software Testing, 2024. URL: https://www.guru99.com/test-scenario-vs-test-condition.html (2024-08-13)
[26] Spillner, Andreas; Linz, Tito. Nav. dj., str. 15.
[27] Isto.

Testni predmet (eng. *test item*)

Podrazumijeva se da se ispitni objekt ne može testirati kao cjelina. Potrebno je identificirati zasebne testne predmete ili komponente sustava koji se testiraju kako bi se osiguralo da zadovoljavaju određene kriterije kvalitete.[28]

Testne cjeline i raspored izvršavanja testova (eng. *test suites and test execution schedules*)

Testni slučajevi se ponekad kombiniraju u testne cjeline (eng. test suites) koji se izvode u testnom ciklusu radi lakšeg upravljanja i izvršavanja. Takvi skupovi obično obuhvaćaju niz testnih slučajeva koji se koriste za validaciju određenog aspekta sustava i mogu biti organizirani po određenim kriterijima poput funkcionalnosti, faze razvoja, modula i sl. Vrijeme testiranja ciklusa se definira u rasporedu izvođenja testova, a obično se odnosi na ukupno trajanje potrebno za planiranje, izvršavanje, analizu i retestiranje svih testnih aktivnosti u jednom ciklusu testiranja.[29]

Testne skripte (eng. *test scripts*)

Teste cjeline automatiziraju su pomoću testnih skripti koje obuhvaćaju slijed testova te sve radnje koje su potrebne da bi se ispunili preduvjeti za testiranje. Te posebno napisane skripte ključne su u automatizaciji testiranja, omogućavajući testnim timovima da brže i efikasnije izvrše veći broj testnih slučajeva.[30]

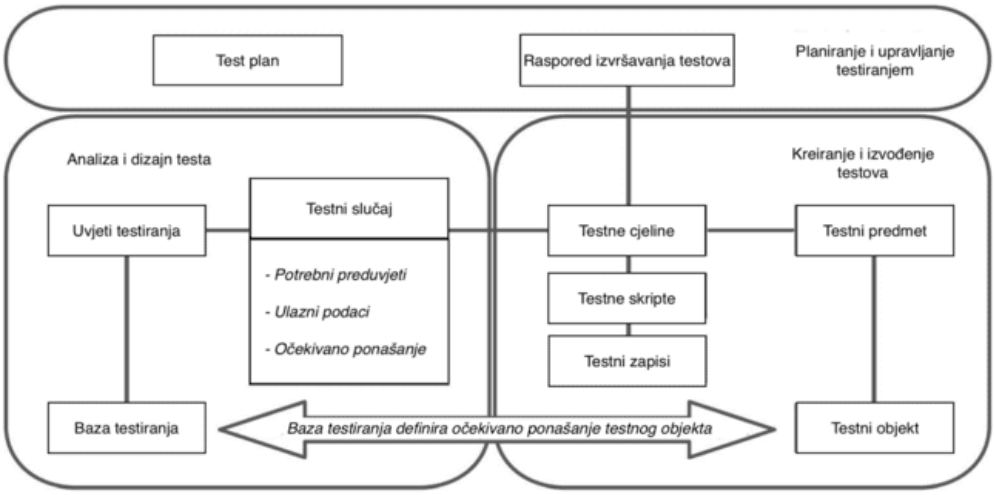
Testni zapisi (eng. *test logs*)

Procesi planiranja i provedbe testova, kao i njihovi rezultati, zapisuju se u izvješću, odnosno testnom zapisu.[31] Takav zapis pruža vrijedne informacije vezane uz proces testiranja, konfiguraciju okruženja, probleme i sl., a neki drugi član tima ih kasnije može koristiti za razumijevanje i analizu svih aspekata testiranja.[32]

[28] Spillner, Andreas; Linz, Tito. Nav. dj., str. 15.
[29] Isto.
[30] Isto.
[31] Spillner, Andreas; Linz, Tito. Nav. dj., str. 15.
[32] Dwivedi, Sonal. What is a Test Log?, 2023. URL: https://www.browserstack.com/guide/what-is-test-log (2024-08-14)

Test plan (eng. *test plan*)

Za objekt testiranja potrebno je kreirati plan testiranja koji definira sve potrebno za izvođenje testova. Test plan obuhvaća i izbor predmeta testiranja, tehnike, definiciju ciljeva testiranja te sva izvješća i koordinaciju aktivnosti koje su povezane s testom.[33]



Slika 1. Odnosi između testnih pojmova.[34]

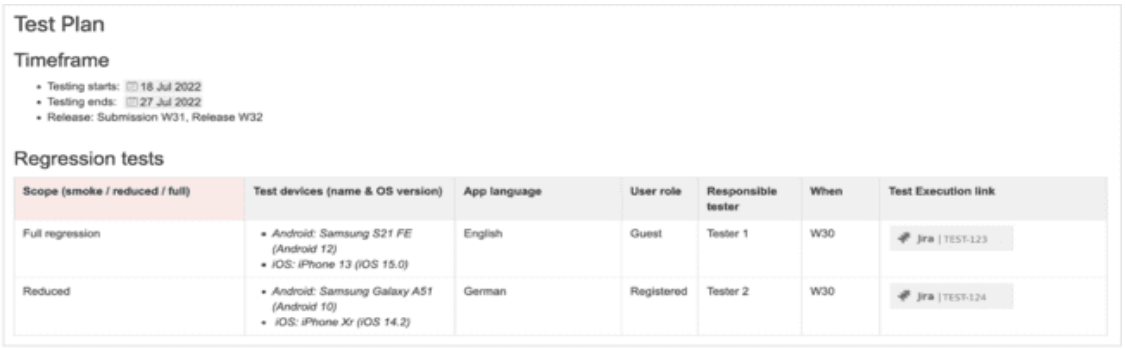
Na slici 1. prikazani su odnosi između testnih pojmova: baza testiranja je osnova za definiranje uvjeta testiranja koji postaju osnova za dizajniranje testnih slučajeva. Testni slučajevi kombiniraju se u testne cjeline, a testne cjeline automatiziraju se pomoću testnih skripti te se procesi planiranja i rezultati testiranja zapisuju u testnim zapisima. Raspored izvršavanja testova definira vrijeme testiranja testnih cjelina. Testne cjeline se razdvajaju na manje komponente (testne predmete) koje testiraju jedan testni objekt dok test plan definira sve potrebno za izvođenje testova.[35]

Testna dokumentacija

U konačnici, test dokumentacija obuhvaća prethodno spomenuti test plan, dokumentaciju o testnim slučajevima, izvještaj o testiranju te dokumentaciju o strategiji koja će se koristiti tijekom testiranja. Pisanje test plana razlikuje se ovisno o projektu/sustavu koji se testira i može ovisiti o internim smjernicama tvrtke. Važne stvari koje test plan (Slika 2.) treba obuhvaćati

[33] Spillner, Andreas; Linz, Tito. Nav. dj., str. 15.
[34] Isto, str. 16.
[35] Isto, str. 14.-15.

jesu: vremenski okvir (početak i kraj testiranja, važni datumi, primjerice datum izdavanja (eng. release date) i sl., opseg testiranja, specifične informacije o sustavu, uređaji za testiranje, ime osobe koja testira (ukoliko više testera radi na projektu), datum izvođenja testova, poveznicu na izvođenje testova (ukoliko se koristi Jira (Testmo), TestRail i sl.[36]).[37]



Test Plan

Timeframe

- Testing starts: 18 Jul 2022
- Testing ends: 27 Jul 2022
- Release: Submission W31, Release W32

Regression tests

Scope (smoke / reduced / full)	Test devices (name & OS version)	App language	User role	Responsible tester	When	Test Execution link
Full regression	• Android: Samsung S21 FE (Android 12) • iOS: iPhone 13 (iOS 15.0)	English	Guest	Tester 1	W30	Jira | TEST-123
Reduced	• Android: Samsung Galaxy A51 (Android 10) • iOS: iPhone Xr (iOS 14.2)	German	Registered	Tester 2	W30	Jira | TEST-124

Slika 2. Primjer test plana.[38]

Izrada dokumentacije o testnim slučajevima podrazumijeva definiranje ulaznih vrijednosti, preduvjete testiranja, očekivano ponašanje sustava i rezultate testiranja. Tijekom pisanja dokumentacije o testnim slučajevima, dobro je imati i zapisane testne scenarije, koji su može se reći, testni slučajevi visoke razine. Scenariji ne zahtijevaju detaljno opisivanje, ali su vrlo korisni testeru kao podsjetnik što testirati.[39] Testni slučajevi odgovaraju na pitanje kako testirati. Minimalno informacija koje bi testni slučaj (Slika 3.) trebao sadržavati je: naziv, koraci i očekivani rezultat, a osim toga može sadržavati i referencu na zahtjev/korisničku priču i preduvjete testiranja.[40]

[36] Alati za upravljanje testnim slučajevima.
[37] Quality Assurance Handbook; Test Documentation, 2023. URL: https://infinum.com/handbook/qa/basics/test-documentation (2024-03-02)
[38] Isto.
[39] Quality Assurance Handbook; Writing test cases, 2022. URL: https://infinum.com/handbook/qa/basics/writing-test-cases (2024-03-02)
[40] Isto.

```
Title:
Verify that photos can be exported with visible tags

Pre-conditions:
The user is logged into his account.
The user has uploaded photos on his account.

Steps:

1. Go to the Photos tab
2. Click on the "View photo" button
3. Click on the "Edit tags" button
4. Add tags
5. Click on the "Save tags" button
6. Click on the vertical ellipsis (⋮) in the upper right corner
7. Click on "Download"
8. Click on the "Export with VISIBLE tags" button


Expected result:
Verify that all the tags are visible on the exported photo.

Reference:
(insert link)
```

Slika 3. Primjer testnog slučaja.[41]

Na Slici 3. prikazan je primjer testnog slučaja provjere izvoza slike s vidljivim oznakama. Kako bi se ovaj testni slučaj izveo, potrebni je ispuniti preduvjete: korisnik je prijavljen na vlastiti korisnički račun te ima već učitane fotografije na svom računu. Na Slici 3. su također opisani koraci koji se trebaju dogoditi da bi test bio izvršen: otići na na karticu (eng. tab) *Fotografije*, kliknuti na *View photo* gumb, zatim kliknuti na *Edit tags* gumb, dodati oznake i kliknuti na *Save tags* gumb, potom kliknuti na vertikalnu elipsu u gornjem desnom kutu ekrana i kliknuo na *Download* te na kraju kliknuo na *Export with visible tags* gumb. Očekivani rezultat ovog testnog slučaja je da su korisniku nakon izvoza, sve oznake koje su dodane fotografiji vidljive na preuzetoj (izvezenoj) verziji te fotografije.

Izvještaj testiranja obuhvaća sažetak ciljeva, aktivnosti i rezultata testiranja. On se koristi kako bi se dionicima projekta omogućilo lakše razumijevanje i uvid u kvalitetu proizvoda. Osnovno izvješće uključuje izvršni pregled (sažetak ključnih saznanja), cilj testa (informacije o vrsti i svrsi testa), sažetak testa (prikaz uspješnih, neuspješnih i blokiranih testova) i pogreške (opisane s naznakom prioriteta i statusa).[42]
Dokumentacija o strategiji testiranja obuhvaća sve opisano u ovom odjeljku i više. Drugim riječima, strategija testiranja obuhvaća sve dokumente koji objašnjavaju kako će proces

[41] Quality Assurance Handbook; Writing test cases, 2022. URL: https://infinum.com/handbook/qa/basics/writing-test-cases (2024-03-02)
[42] Quality Assurance Handbook; Test Documentation, 2023. URL: https://infinum.com/handbook/qa/basics/test-documentation (2024-03-02)

testiranja izgledati na konkretnom projektu.[43] Neke od informacija koje ovaj dokument opisuje su: opseg i pregled izvršenja testova, pristup testiranju (odgovornosti pojedinog člana tima, vrste testiranja, alate za automatizaciju i sl.), informacije o testnim okolinama, alate koji će se koristiti na projektu, plan izdavanja (eng. release plan) i životni ciklus pogreške (dokument koji objašnjava tijek prijavljene pogreške).[44]

2.2. Vrste, razine i metode testiranja

U domeni testiranja postoji više vrsta testiranja, ali dvije najčešće i najosnovnije su ručno i automatsko testiranje. Ručno testiranje podrazumijeva ručno testiranje funkcionalnosti aplikacije bez upotrebe alata za automatizaciju. Tester koji provodi ručno testiranje izvodi radnje, promatra što se događa i uspoređuje stvarne rezultate s očekivanim rezultatom.[45] Automatsko testiranje podrazumijeva izvođenje testnih slučajeva putem alata za automatsko testiranje.[46] Vrlo često razvojni ciklusi zahtijevaju ponavljanje određenih skupova testova, a alati za automatizaciju omogućuju da se ti skupovi testova ponovno reproduciraju prema potrebi.

Razinama testiranja uobičajeno se smatraju sljedeće četiri razine:

1. Testiranje jedinica (eng. Unit testing),
2. Integracijsko testiranje (eng. Integration testing),
3. Testiranje sustava (eng. System testing),
4. Testiranje prihvatljivosti (eng. Acceptance testing).[47]

Testiranje jedinica je razina koja se koristi za testiranje pojedinačnih jedinica i komponenti sustava. Svrha ovog testiranja je potvrditi da svaka jedinica ili komponenta radi ispravno i

[43] Quality Assurance Handbook; Test Documentation, 2023. URL: https://infinum.com/handbook/qa/basics/test-documentation (2024-03-02)
[44] Isto.
[45] Hamilton, Thomas. Manual Testing Tutorial, 2024. URL: https://www.guru99.com/manual-testing.html (2024-06-06)
[46] Hamilton, Thomas. Automation Testing, 2024. URL: https://www.guru99.com/automation-testing.html (2024-06-06)
[47] Hamilton, Thomas. Levels of Testing in Software Testing, 2024. URL: https://www.guru99.com/levels-of-testing.html (2024-06-06)

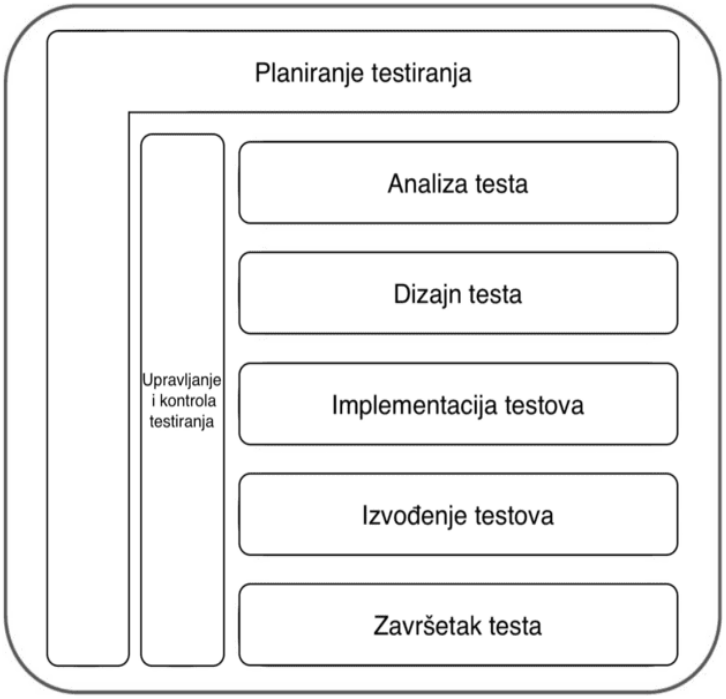
prema očekivanjima. Provodi se tijekom faze razvijanja proizvoda na način da se izolira dio koda za određenu jedinicu i testira se se ponašanje iste.[48] Integracijsko testiranje je razina koja slijedi nakon testiranja jedinica. Uz pretpostavku da je testiranje jedinica izvršeno u prethodnoj fazi (i da su se pogreške ispravile ukoliko ih je bilo), integracijsko testiranje zatim provodi integraciju jedinica u logički povezanim većim skupovima jedinica i testira ih se kao grupu, odnosno provjerava kako funkcioniraju komponente zajedno.[49] Kada je integracijsko testiranje završeno, testiranje prelazi u višu razinu; testiranja sustava. Tijekom ove razine se testira sustav kao cjelina i provjerava ispunjava li isti specifikacije.[50] Prethodne razine stavljaju fokus na testiranje tehničkih specifikacija, dok testiranje sustava omogućava testeru proučavanje ponašanja sustava iz perspektive krajnjeg korisnika.[51] Za posljednju razinu, testiranja prihvatljivosti, direktno su uključeni krajnji korisnici/klijenti kako bi potvrdili i prihvatili ponašanje programa prije premještanja aplikacije u produkcijsko okruženje.[52]

Dvije najvažnije metode testiranja su testiranje crne (eng. black-box testing) i bijele kutije (eng. white-box testing). Testiranje crne kutije je tehnika kod koje tester ne mora poznavati interni rad ili strukturu koda, odnosno fokus je isključivo na vanjskom ponašanju programa. Ključne karakteristike ove metode obuhvaćaju neovisno testiranje, dizajniranje testova na temelju zahtjeva i specifikacija sustava, funkcionalno testiranje i testiranje bez znanja o internom kodu.[53] Nasuprot tome, metoda testiranja bijele kutije je tehnika koja se fokusira na internu logiku i strukturu koda. Ona dakle omogućuje pristup izvornom kodu te pregled i provjeru unutarnjeg rada programa. Za korištenje ove tehnike potrebno je dobro poznavanje interne strukture, a ona omogućuje identifikaciju problema koji su često nevidljivi drugim metodama testiranja.[54]

2.3. Proces testiranja (eng. testing process)

[48] Hamilton ,Thomas. What is Unit Testing?, 2024. URL: https://www.guru99.com/unit-testing-guide.html (2024-06-06)
[49] Spillner, Andreas; Linz, Tito. Nav. dj., str. 66.
[50] Spillner, Andreas; Linz, Tito. Nav. dj., str. 74.
[51]Spillner, Andreas; Linz, Tito. Nav. dj., str. 74.
[52] Hamilton, Thomas. What is User Acceptance Testing (UAT)? Examples, 2024. URL: https://www.guru99.com/user-acceptance-testing.html (2024-06-06)
[53] Das, Sourojit. What is Black Box Testing: Types, Tools & Examples, 2023. URL: https://www.browserstack.com/guide/black-box-testing (2024-06-09)
[54] Akhtar, Hamid. What is White Box Testing? (Example, Types, & Techniques), 2023. URL: https://www.browserstack.com/guide/white-box-testing (2024-06-09)

Proces testiranja programskih rješenja je strukturiran pristup koji osigurava da sustav ispunjava postavljene standarde kvaliteta, funkcionalnosti i performansi. Uključuje niz faza koje se kontinuirano prate kako bi se identificirale i ispravile greške, osigurala pouzdanost i funkcionalnost sustava. Slika 4. prikazuje procese testiranja te se u nastavku rada nalazi pregled tipičnih aktivnosti u procesu testiranja.



Slika 4. Proces testiranja.[55]

Planiranje testiranja (eng. test planning)

Svako testiranje započinje osmišljavanjem i izradom plana testiranja. Test plan potrebno je redovito pregledavati i ažurirati te ga prilagođavati promjenjivim situacijama tijekom cijele faze razvoja proizvoda.[56] U test planu se definiraju objekti testiranja, karakteristike kvalitete (podrazumijeva raspisivanje očekivanog ponašanja određenih testnih objekata), ciljevi testiranja te aktivnosti koje je potrebno odraditi za provjeru funkcionalnosti sustava.[57] Planiranje testiranja također obuhvaća i definiranja kriterije pokrivenosti testiranja (eng.

[55] Spillner, Andreas; Linz, Tito. Nav. dj., str. 27.
[56] Isto, str. 29.
[57] Isto, str. 29-30.

coverage criteria) kojim se određuje točka u kojoj je izvršeno dovoljno testiranja prema planiranom obuhvatu.[58] Ono podrazumijeva da za svaku razinu testiranja, tester može procijeniti jesu li se testovi proveli dostatno i jesu li postignuti ciljevi testiranja. S obzirom da se u test planu definiraju testovi koji se trebaju provesti na određenoj razini testiranja, često ima smisla izraditi poseban test plan za specifičnu razinu testiranja, ali tester također može i spojiti sve u jedan "glavni" test plan. U test planu mogu se definirati i rokovi potrebni za izvršavanje ključnih aktivnosti testiranja.[59]

Upravljanje i kontrola testiranja (eng. test monitoring and control)

Upravljanje i kontrola testiranja obuhvaćaju stalno promatranje napretka aktivnosti testiranja i usklađenosti tih aktivnosti s planiranim aktivnostima, prijavu potencijalnih odstupanja te provođenje aktivnosti potrebnih za postizanje ciljeva u promijenjenim okolnostima.[60] Ova aktivnost temelji se na tzv. kriterijima završetka (eng. *exit criteria*) za svaki zadatak, pri čemu se procjenjuje se jesu li ispunjeni određeni uvjeti za završetak testiranja pojedinog objekta.[61] Ukoliko ti uvjeti nisu ispunjeni, potrebno je osmisliti i izvršiti dodatne testove. U slučaju da to nije izvedivo, potrebno je razjasniti situaciju i procijeniti rizike koji iz toga mogu proizaći.[62] Od testera se očekuju redoviti izvještaji o napretku testiranja i odstupanja od inicijalnog test plana. Također, izvještaji trebaju uključivati sve informacije koje su potrebne dionicima, od informacija o prerano prekinutim testovima do očekivanog vremena završetka testiranja, rezultata testiranja i sl.[63]

Analiza testa (eng. test analysis)

Analiza uključuje određivanje što se točno treba testirati. Proučava se spomenuta osnova za testiranje (eng. test basis) prema kojoj se analizira je li dokumentacija dovoljno detaljna i mogu li se uopće funkcionalnosti testirati prema definiranim uvjetima testiranja.[64] Analiza se također provodi u svrhu provjeravanja uvjeta testiranja (eng. test conditions) prema određenih

[58] Spillner, Andreas; Linz, Tito. Nav. dj., str. 30.
[59] Isto.
[60] Isto.
[61] Isto, str. 30-31.
[62] Isto. str. 31.
[63] Isto, str. 31.
[64] Isto.

definiranim kriterijima završetka (eng. test coverage). Analiza osnova za testiranje uključuje razmatranje zahtjeva koji definiraju planirano funkcionalno i nefunkcionalno ponašanje sustava i komponenti, a to obuhvaća tehničke zahtjeve, funkcionalne zahtjeve, korisničke priče, slučajeve korištenja (eng. use cases) i sličnu dokumentaciju.[65] Ukoliko zahtjevi ne specificiraju ponašanje sustava dovoljno precizno, tada se testni slučajevi ne mogu jednostavno izvesti. Nadalje, u ovoj fazi provodi se i analiza dokumentacije i pogrešaka u dokumentaciji, što uključuje dokumente o arhitekturi sustava, specifikacije dizajna, specifikacije sučelja i sl. te provjeravanje ima li praznina i propusta u dokumentaciji.[66] Tijekom analize pogrešaka u dokumentaciji važno je provjeriti sadrži li dokumentacija kakve nedosljednosti, proturječja i nepreciznosti te na takve nedostatke ukazati. Potrebno je analizirati i testni objekt, pojedinačne komponente ili sam sustav, kako bi znali što testirati.

Ova aktivnost uključuje i analizu rizika, koja podrazumijeva proučavanje izvješća o analizi rizika vezanih uz funkcionalne, nefunkcionalne te strukturne aspekte sustava i njihovih komponenti[67]. Potencijalne neispravnosti mogu uzrokovati ozbiljne rizike, stoga pažnja na detalje treba biti sastavni dio procesa testiranja proizvoda.

Dizajn testa (eng. test design)

Dizajnom testa određuje se kako će se testirati i koji testni slučajevi će se koristiti. U ovoj fazi najveću ulogu imaju uvjeti testiranja koji se koriste za stvaranje testnih slučajeva. Razlikuju se dvije razine testnih slučajeva: apstraktna i konkretna (eng. abstract and concrete test cases). Apstraktnom razinom određuju se nespecifični testni slučajevi i ista ne uključuje specifične ulazne podatke i očekivano ponašanje nego se opisuje pomoću logičkih operatora.[68] Za razliku od apstraktne razine, konkretni test slučajevi obuhvaćaju specifične ulazne podatke i definirane očekivane vrijednosti.[69] Ove dvije razine najjednostavnije je objasniti na sljedećem primjeru; autosalon daje mogućnost popusta na cijenu vozila. Za cijene vozila ispod €15,000 ne daje se popust, za cijene vozila između €15,000 i €20,000 moguć je popust od 5%. Za cijene vozila od €20,000 do €25,000 moguće je dobiti popust od 7%. Za cijene vozila iznad €25,000, klijent će dobiti popust od 8.5%. Ovaj tekst daje mogućnost testeru da uspostavi veze između cijena i popusta na sljedeći način:

[65] Spillner, Andreas; Linz, Tito. Nav. dj. str. 32.
[66] Isto, str. 32-33.
[67] Isto, str. 32.
[68] Spillner, Andreas; Linz, Tito. Nav. dj., str. 34.
[69] Isto.

Cijena $<$ 15,000	Popust = 0%
15.000 $\leq$ Cijena $\leq$ 20,000	Popust = 5%
20.000 $<$ Cijena $<$ 25,000	Popust = 7%
Cijena $\geq$ 25,000	Popust = 8.5%

Na temelju formula moguće je definirati apstraktni test slučaj (Slika 5.) gdje prvi redak predstavlja ulazne vrijednosti cijene u eurima, a drugi očekivani rezultat (koliki je popust u postocima):

Apstraktni testni slučaj	**1**	**2**	**3**	**4**
Ulazna vrijednost x (cijena u €)	x < 15000	15000 ≤ x ≤ 20000	20000 < x < 25000	x ≥ 25000
Očekivani rezultat (popust u %)	0	5	7	8.5

Slika 5. Primjer apstraktnog test slučaja.[70]

No, da bi se izvršili ovi testovi, potrebno ih je pretvoriti u konkretne slučajeve (Slika 6.) na način da im se doda specifična vrijednost i provjeri konkretan rezultat cijene popusta:

Konkretni testni slučaj	**1**	**2**	**3**	**4**
Ulazna vrijednost x (cijena u €)	14500	16500	24750	31800
Očekivani rezultat (popust u €)	0	825	1732.50	2703

Slika 6. Primjer konkretnog test slučaja.[71]

Važno je napomenuti da opis ovih razina služe samo za jednostavniji prikaz razlika između apstraktnih i konkretnih test slučajeva, te da slučajevi nisu pokriveni detaljno. Na primjer, u ovoj ilustraciji nema testnog slučaja koji pokriva pogrešan unos (negativna cijena) i slične

[70] Spillner, Andreas; Linz, Tito. Nav. dj., str. 35.
[71] Isto.

slučajeve.[72] Nadalje, u dizajnu testa definiraju se i prioriteti testnih slučajeva, a prvo se izvršavaju slučajevi visokog prioriteta. Da bi se mogli izvršiti test slučajevi, testeru je potrebno osigurati testno okruženje i alate potrebne za testiranje.[73]

Implementacija testova (eng. test implementation)

Implementacija testova završni je korak prije početka izvođenja testova. U ovoj fazi tester odrađuje završne pripreme svih aktivnosti kako bi se testni slučajevi mogli izvoditi u sljedećoj fazi. Ovo podrazumijeva pripremu testnih materijala (eng. testware) kao i provjeru testnog okruženja i testne infrastrukture.[74] U ovoj fazi, definira se procedura testiranja, odnosno testni slučajevi se trebaju staviti u logičan slijed. Da bi testni proces bio učinkovit i da bi se zadržala logička struktura, potrebno je organizirati testne slučajeve u testne cjeline.[75] Tester mora imati u vidu da slijed testova mora planirati unaprijed jer pojedini testni slučajevi uvjetuju izvršavanje drugih testnih slučajeva.

Izvođenje testova (eng. test execution)

Izvođenje testova ima najviše smisla započeti na način da se prvo provjeri jesu li sve testne komponente dostupne za testiranje i mogu li se na njima početi izvoditi testovi. Bilo da se testovi izvode ručno ili automatski, vrlo je važno tijekom izvođenja testova voditi bilješke o rezultatima provedenih testova. Smatra se da je izvođenje testova bez zapisivanja svih rezultata (prolaz, neuspjeh, blokiran) bezvrijedno.[76] Svaki zapis tijekom izvođenja testova mora biti precizan i razumljiv. Bilješke također služe za provjeru je li cijela strategija testiranja izvršena prema planu. Kroz bilješke treba biti prikazano koje su funkcionalnosti testirane, kada, tko ih je testirao, koliko detaljno i kakvi su rezultati testa.[77] Ukoliko je bilo koji testni slučaj preskočen iz određenog razloga, potrebno ga je zabilježiti. Jako je važno zapisivati bilješke u vezi testnih slučajeva, ne samo kako bi testovi bili razumljivi i osobama koje nisu direktno uključene u proces testiranja, nego i kako bi se ti testovi kasnije mogli lakše ponoviti.[78] Pogotovo u

[72] Spillner, Andreas; Linz, Tito. Nav. dj., str. 35.
[73] Isto, str. 36.
[74] Isto, str. 36-37.
[75] Isto, str. 37.
[76] Spillner, Andreas; Linz, Tito. Nav. dj., str. 38.
[77] Isto.
[78] Isto.

situacijama kada tester otkrije pogrešku (eng. bug) na način da prepozna odstupanje očekivanog ponašanja od stvarnog ponašanja sustava. Pogrešku bi bilo najbolje prvo dokumentirati, a zatim početi tražiti uzroke. Moguće je da će biti potrebno specificirati i izvršiti dodatne test slučajeve. Također, ukoliko tester otkrije pogrešku, potrebno je istu i prijaviti. Nakon što programer ispravi grešku u kodu, tester bi trebao ponoviti testni slučaj da provjeri je li pogreška ispravljena, no također i kreirati nove testne slučajeve, da bi provjerio da ispravljanje koda nije izazvalo pogrešku na nekom drugom mjestu.[79] U ovoj fazi se prati i slijed testova, da bi se moglo prepoznati je li svaka komponenta testiranja bila obuhvaćena u cijelosti. Zadnji korak je provjeriti jesu li ciljevi testiranja ispunjeni. Tester bi trebao provjeriti koji su zahtjevi prošli planirane i uspješno izvedene testove, a koji zahtjevi nisu verificirani zbog neuspješnih testova ili pogrešaka koje su uočene tijekom testiranja. Na temelju toga, može se procijeniti jesu li ispunjeni spomenuti kriteriji završetka i je li test uspješno završen.[80]

Završetak testa (eng. Test Completion)

Završetak testiranja predstavlju finalnu aktivnost u cijelom procesu testiranja. Ono podrazumijeva uspoređivanje svih podataka koji su prikupljeni prethodno dovršenim aktivnostima testiranja u svrhu evaluacije iskustva testiranja i pregledavanje testnih materijala.[81] Jednostavnije rečeno, ono može uključivati procjenu kako je testiranje prošlo, analizu rezultata, kao i organizaciju i pregled testnog materijala kako bi se osigurala dosljednost i pravilno vođenje evidencije. Točan trenutak u kojem je test završen često se razlikuje prema vrsti projekta/modela na kojem se vrše testiranja, a neki od trenutaka kada se može zaključiti da je testiranje završeno jesu: kad se aplikacija pokrene na produkcijskoj okolini (namijenjenoj stvarnim korisnicima), završetkom ili prekidom testnog projekta, završetkom agilnog ciklusa projekta, kompletiranjem testnih aktivnosti za određeno izdanje (eng. release) aplikacije i sl.[82] Otvorene i neriješene pogreške ostaju otvorene te se rješavaju u sljedećim iteracijama ili izdanjima. Sažeto izvješće obuhvaća sve aktivnosti i rezultate testiranja te se daje na uvid dionicima projekta.[83] Tijekom životnog vijeka sustava, velika je vjerojatnost da će doći do novih zahtjeva korisnika i javiti potreba za održavanjem sustava, te da će se u tom razdoblju pojaviti i pogreške koje nisu otkrivene tijekom testiranja. Zato je jako važno da testni materijali

[79] Spillner, Andreas; Linz, Tito. Nav. dj., str. 38.
[80] Isto, str. 39.
[81] Isto, str. 40.
[82] Spillner, Andreas; Linz, Tito. Nav. dj., str. 40.
[83] Isto.

(eng. testware) koji su se inicijalno koristili mogu ponovno iskoristiti i prilagoditi ga novim zahtjevima i promjenama, umjesto da ga se piše od nule.[84]

2.4. Pisanje izvještaja o pogrešci

Izvješće o pogrešci važan je dio procesa testiranja i ispravljanja pogrešaka. Prijavljene greške omogućuju praćenje nedostataka i druga odstupanja tijekom testiranja sustava. Cilj ove dokumentacije je pružiti timu stručnjaka informacije o razini grešaka na koje se naišlo tijekom procesa testiranja.[85] Tijekom pisanja izvješća važno je imati na umu da bi on trebao biti jasan, precizan i detaljan kako bi ubrzao proces ispravljanja pogreške. Ne postoji točan predložak kako bi on trebao izgledati, ali najčešće korištena i potrebna polja za opisivanje bugova su:

- Bug ID / Naslov (eng. Bug id/ Title),
- Ozbiljnost i prioritet (eng. Severity and Priority),
- Opis (eng. Description),
- Okolina (eng. Environment),
- Koraci za reprodukciju (eng. Steps to reproduce),
- Očekivani rezultat (eng. Expected result),
- Stvarni rezultat (eng. Actual result),
- Prilozi; snimke zaslona, videozapisi (eng. Attachments; screenshots, videos).[86]

Kvalitetno napisano izvješće omogućuje veću šansu za brže i efektivnije ispravljanje pogreške. Ako tester ne napiše jasno izvješće, programer bi mogao odbiti ispraviti bug s razlogom da ga ne može reproducirati. Zato je potrebno tijekom pisanja izvješća biti konkretan i precizan. Bug se mora moći reproducirati, ako se on ne može ponoviti, nikada se neće popraviti. Iz tog razloga potrebno je jasno opisati korake za reprodukciju. Također, dobro izvješće mora biti jasno i

[84] Spillner, Andreas; Linz, Tito. Nav. dj., str. 40.
[85] Walker, Alyssa. How to Write A Bug Report with Examples, 2024. URL: https://www.guru99.com/how-to-write-a-bug-report.html (2024-04-05)
[86] Quality Assurance Handbook; Writing bug reports, 2023. URL: https://infinum.com/handbook/qa/basics/writing-bug-reports (2024-06-06)

sažeto bez izostavljanja ključnih informacija. Svaki nedostatak jasnoće može dovesti do nesporazuma i usporiti razvojni proces.[87]

3. Automatsko testiranje

Automatsko testiranje predstavlja ključni element modernog razvoja programskih rješenja u raznim tvrtkama kako bi ostali konkurentni na tržištu koje se stalno i brzo razvija.[88] Prelazak s tradicionalnog modela vodopada (eng. Waterfall) na agilni pristup razvoju programskih rješenja omogućio je veću fleksibilnost i brzu prilagodbu zahtjevima tržišta.[89] Agilna metodologija koristi scrum okvir za kontinuirano poboljšanje proizvoda kroz mala i česta izdanja.[90] Testni ciklusi su počeli zahtijevati automatizaciju kako bi se održao korak s brzinom razvoja programskih rješenja i sveobuhvatno provjerio sustav za sve moguće pogreške.[91] Automatizacija testiranja sustava zahtjeva znatna ulaganja novca i vremena, stoga odluka o investiranju u automatizaciju nije jednostavna. Napor koji je uložen u kreiranje automatskih testova mora se oduzeti od računice koliko je uštađeno automatizacijom te analiza troškova i koristi obično daje negativan rezultat nakon jednog korištenja. Tek nakon što se izvrše više automatiziranih regresijskih testova, moći će se vidjeti rezultati uštede vremena i novca.[92] Potencijalni rizici tijekom uvođenja automatizacije su što je previše lako podcijeniti vrijeme koje je potrebno i troškove za automatizaciju, napor koji je uložen u kreiranje i održavanje automatskih testova bude veći od očekivanog, korištenje automatiziranih testova u situacijama kad je ručno testiranje jednostavnije, ekonomičnije i sl.[93] No ipak, prednosti automatskog testiranja je povećanje ukupne učinkovitosti procesa testiranja, ponavljanje testnih cjelina, smanjivanje napora testiranja te jednostavnije prikupljanje i procjenu podataka (napredak

[87] Software Testing Help; How To Write A Good Bug Report? Tips and Tricks. URL: https://www.softwaretestinghelp.com/how-to-write-good-bug-report/ (2024-06-06)
[88] BrowserStack; Benefits of Automation Testing. 2023. URL: https://www.browserstack.com/guide/benefits-of-automation-testing (2024-08-10)
[89] Shadow Software News. How to Transition from Waterfall to Agile: A Step-by-Step Guide, 2024. URL: https://shadowsoftware.com/how-to-transition-from-waterfall-to-agile-a-step-by-step-guide/ (2024-08-22)
[90] Drumond, Claire. What is scrum and how to get started: Scrum Guide - What it is, how it works, and how to start. URL: https://atlassian.com/agile/scrum (2024-08-22)
[91] BrowserStack; Benefits of Automation Testing. 2023. URL: https://www.browserstack.com/guide/benefits-of-automation-testing (2024-08-10).
[92] Spillner, Andreas; Linz, Tito. Nav. dj., str. 269.
[93] Isto, str. 270

testova, pokrivenost, podaci o neuspjehu itd.).[94] Cilj automatizacije nije potpuno eliminirati ručno testiranje, nego smanjiti ručno izvođenje testnih slučajeva.[95] Kao tester, potrebno je konstantno izoštravati vlastiti misaoni proces, postavljati pitanja, biti znatiželjni, stvoriti naviku preispitivanja svega što se testira.

Automatsko testiranje najčešće se povezuje s regresijskim testiranjem (eng. regression testing). Regresijsko testiranje se također definira kao vrsta testiranja sustava, a svrha ovog testiranja je provjeriti da nedavne promjene izvršene na sustavu nisu utjecale na postojeće funkcionalnosti sustava.[96] Drugim riječima, može se reći da je regresijsko testiranje ponovno izvođenje postojećih testnih slučajeva kako bi potvrdili da postojeće funkcionalnosti rade ispravno.[97] Automatizacija regresijskih testova je poželjna unutar DevOps procesa CI/CD budući da je ručno izvođenje postojećih testnih slučajeva dugotrajan proces. Osobito kada aplikacija obuhvaća više izdanja, automatizirano testiranje ima nekoliko prednosti nad ručnim testiranjem: testne skripte mogu se višekratno koristiti, smanjuje se mogućnost pogreške, brže je od procesa ručnog testiranja, skripte je moguće izvršiti istovremeno te nije potrebno povećanje novih resursa za automatizaciju.[98]

Automatizacija testova zahtjeva odluku o alatu/radnom okviru koji će se koristiti za poboljšanje učinkovitosti testiranja. Neki od alata za automatsko testiranje koji su dostupni na tržištu su Playwright, Selenium, Cypress i Puppeteer.[99]

Playwright je sve popularniji radni okvir za automatsko testiranje koji omogućava testiranje cijelog toka aplikacije, provjeru funkcionalnosti na raznim preglednicima i platformama te osigurava da su svi dijelovi sustava pravilno integrirani i da ispravno funkcioniraju zajedno.[100] Predstavljen je 2020. godine od strane Microsoftovih suradnika te podržava automatsko testiranje više različitih mrežnih preglednika (Chrome, Firefox, Safari…) i operativnih sustava

[94] Spillner, Andreas; Linz, Tito. Nav. dj., str. 269.-270.
[95] Isto.
[96] Hamilton, Thomas.What is Regression Testing?, 2024. URL: https://www.guru99.com/regression-testing.html (2024-08-20)
[97] Kitakabee. Automated Regression Testing: A Detailed Guide, 2023. URL: https://www.browserstack.com/guide/automated-regression-testing (2024-08-20)
[98] Hamilton, Thomas.What is Regression Testing?, 2024. URL: https://www.guru99.com/regression-testing.html (2024-08-20)
[99] Ulili, Stanley. Playwright vs Puppeteer vs Cypress vs Selenium (E2E testing), 2024. URL: https://betterstack.com/community/comparisons/playwright-cypress-puppeteer-selenium-comparison/#4-debugging-tools (2024-08-20)
[100] Playwright. URL: https://playwright.dev/ (2024-06-10)

(Windows, macOS, Linux, mobilne platforme kao što su Android i iOS…) na jednom API (eng. Application Programming Interface); jedinstvenom programskom sučelju koje omogućuje interakciju s različitim preglednicima i platformama.[101] Playwright API je također i višejezičan, odnosno podržava korištenje više programskih jezika: JavaScript, TypeScript, Python, .NET i Java.[102]

Playwright testovi dizajnirani su za jednostavno korištenje, pisanje i razumijevanje testova. Omogućuju imitaciju korisničkih radnji na mrežnoj stranici (klikanje gumbova, ispunjavanje formi i sl.). Nakon što test izvrši radnje, provjerava da je stanje aplikacije prema očekivanom ponašanju. Drugačije rečeno, Playwright je dizajniran da može oponašati korisničke radnje na aplikaciji te prema tome utvrditi je li rezultat tih radnji u skladu s očekivanim ponašanjem aplikacije.[103]

Konkurencija Playwright-u je Selenium, jedan od najstarijih alata za automatsko testiranje na tržištu, predstavljen 2004. godine. Selenium također podržava korištenje više programskih jezika (Java, Python, C#, Ruby, Kotlin i JavaScript) te više mrežnih preglednika (Chrome, Firefox, Safari i Opera).[104] Nadalje, Cypress je također jedan od vodećih alata za automatsko testiranje, dizajniran za pisanje, izvođenje i otklanjanje pogrešaka. Omogućuje pisanje testova u JavaScript programskom jeziku te podržava mrežne preglednike kao što su Chrome, Firefox i Microsoft Edge. Puppeteer također poput Cypressa omogućava pisanje testova u JavaScript programskom jeziku. Puppeteer je izvorno fokusiran na testiranje Chrome preglednika (uz Microsoft Edge) te posjeduje eksperimentalnu podršku za Firefox, ali nema podršku za testiranje vrlo popularnog Safari preglednika.[105]

Svaki od spomenutih alata za automatsko testiranje ima svoje specifične prednosti i nedostatke te odluka o odabiru pravog alata uglavnom ovisi o specifičnim potrebama projekta. Primjerice, ukoliko je brzina izvršavanja testova ključna u donošenju odluke, tada bi Playwright imao prednost nad ostalim alatima jer brže izvršava testove od ostalih alata za automatsko

[101] Playwright. URL: https://playwright.dev/ (2024-06-10)
[102] Isto.
[103] Playwright; Writing tests. URL:https://playwright.dev/docs/writing-tests (2024-06-09)
[104] Ulili, Stanley. Playwright vs Puppeteer vs Cypress vs Selenium (E2E testing), 2024. URL: https://betterstack.com/community/comparisons/playwright-cypress-puppeteer-selenium-comparison/#4-debugging-tools (2024-08-20)
[105] Ulili, Stanley. Playwright vs Puppeteer vs Cypress vs Selenium (E2E testing), 2024. URL: https://betterstack.com/community/comparisons/playwright-cypress-puppeteer-selenium-comparison/#4-debugging-tools (2024-08-20)

testiranje.[106] Kao što je spomenuto, Puppeteer ne podržava testiranje na Safari pregledniku, te time potencijalno ograničava testiranje kompatibilnosti s više preglednika. Selenium primjerice ne podržava automatske ponovne pokušaje (eng. retries) izvršavanja testova, koje Playwright i Cypress imaju ugrađene, a važni su radi upravljanja nestabilnim (eng. flaky) testovima. No važno je naglasiti da svi spomenuti alati imaju podršku za CI/CD integraciju.[107] CI/CD predstavlja kontinuiranu integraciju i kontinuiranu isporuku/implementaciju (eng. continuous integration and continuous delivery/deployment), koja ima za cilj održavanje kontinuiranog ciklusa razvoja i ažuriranja sustava.[108]

4. Istraživački rad - Automatsko testiranje Demo Web Shop aplikacije

Istraživački dio rada prikazuje primjenu automatskog testiranja u razvoju mrežnih aplikacija, s naglaskom na tzv. testiranje od početka do kraja (eng. end-to-end testing). End-to-end testiranje ili tzv. testiranje od početka do kraja, obuhvaća testiranje tijeka rada cjelokupne aplikacije, na način na koji bi korisnik to činio kada bi se služio aplikacijom.[109]

Svrha ovog istraživanja je pokazati učinkovitost i prednosti automatiziranog testiranja mrežnih mjesta koristeći Playwright radnu okolinu. Cilj je demonstrirati kako Playwright može pomoći u ubrzavanju procesa testiranja, poboljšanju kvalitete sustava uz smanjenje ljudske pogreške te optimizaciji vremena potrebnog za izvođenje testnih slučajeva.

Demo Web Shop mrežna je aplikacija vrlo pogodna za učenje i izvođenje automatiziranih testnih slučajeva. Na aplikaciji su imitirane osnovne funkcionalnosti online trgovine koje omogućuju testerima jednostavno i stvarno iskustvo testiranja funkcionalnosti bez rizika od financijske štete.

---

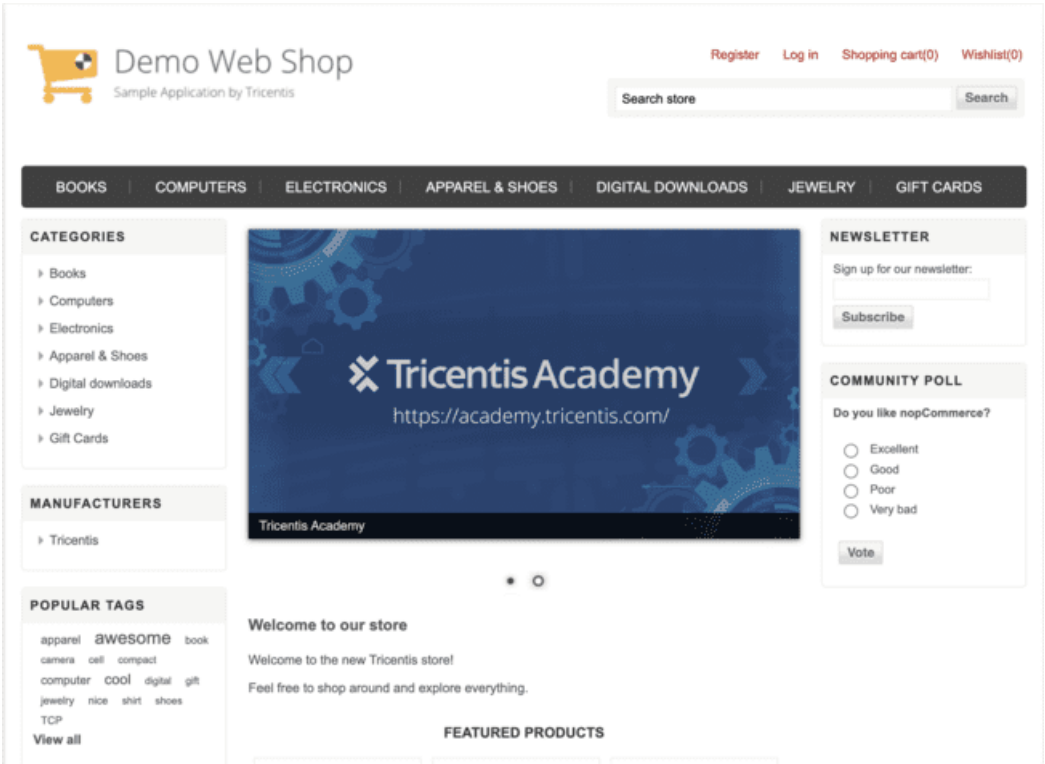
[106] Ulili, Stanley. Playwright vs Puppeteer vs Cypress vs Selenium (E2E testing), 2024. URL: https://betterstack.com/community/comparisons/playwright-cypress-puppeteer-selenium-comparison/#4-debugging-tools (2024-08-20)

[107] Isto.

[108] Red Hat. What is CI/CD?, 2023. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd (2024-08-21)

[109] Verma, Antra. End To End Testing: Tools, Types, & Best Practices, 2024. URL: https://www.browserstack.com/guide/end-to-end-testing (2024-06-09)

Radni okvir u kojem su se izvršavali testovi je Playwright, a u radu će se u nastavku prikazati cjelokupan proces automatskog testiranja Demo Web Shop aplikacije. Istraživački rad obuhvaća analizu i provedbu 35 različitih testova izvršenih na Demo Web Shop aplikaciji s namjerom pokrivanja ključnih funkcionalnosti i provjere cjelokupnog korisničkog iskustva testiranog mrežnog mjesta. Za uređivanje i izvršavanje koda koristio se Visual Studio Code; jednostavni uređivač izvornog koda dostupan za Windows, macOs i Linux te ugrađenom podrškom za JavaScript, TypeScript i Node.js.[110] Testovi su pisani u TypeScript programskom jeziku koji se temelji na JavaScript-u te strogo brine o tipovima podataka kako bi podržao čvršću integraciju s uređivačem koda.[111] Svi testovi dostupni su na Git Hub repozitoriju na sljedećoj poveznici: https://github.com/mzilic98/diplomski-playwright.



Slika 7. Početna stranica Demo Web Shop aplikacije.[112]

Na Slici 7. prikazana je početna stranica mrežne aplikacije Demo Web Shop na kojoj se izvršavaju automatski testovi. Testni slučajevi obuhvaćaju funkcionalnosti registracije, prijave korisnika, proizvoda, kategorija i košarice.

[110] Visual Studio Code. URL: https://code.visualstudio.com/docs (2024-06-09)
[111] TypeScript. URL: https://www.typescriptlang.org/ (2024-08-20)
[112] Demo Web Shop. URL: https://demowebshop.tricentis.com/ (2024-07-28)

4.1. Konfiguracija testnog okruženja

Konfiguracijska datoteka prikazana na Slici 8. koristi funkciju *defineConfig* iz *@playwright/test* paketa u svrhu definiranja različitih opcija za testiranje. Za potrebe rada u konfiguracijskoj datoteci definirano je:

- *testDir: ' ./tests'* - direktorij u kojem će se nalaziti svi testovi, u ovom slučaju svi testovi će se nalaziti u direktoriju *tests,*
- *outputDir: 'test-results'* - direktorij u koji će se spremati svi testovi,
- *timeout: 30 * 1000* - vremenski limit od 30 sekundi za svaki test, ukoliko test nije izvršen u ovom vremenskom roku bit će neuspješan,
- *expect: { timeout: 5000,}* - vremenski limit od 5 sekundi za očekivanja, ukoliko očekivanja (primjerice provjera vidljivosti elementa) ne budu ispunjena u tom vremenskom roku, test će biti neuspješan,
- use: { baseURL: 'https://demowebshop.tricentis.com/', } - osnovni URL za korištenje u radnjama poput *'await page.goto('/')'*.

Nadalje, u ovoj datoteci definiraju se preglednici na kojima će automatski testovi biti izvršeni. Konfigurirani su projekti za pokretanje testova za "Desktop Chrome", "Desktop Firefox" te "Desktop Safari" preglednike.

```
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  testDir: './tests',
  outputDir: 'test-results',
  timeout: 30 * 1000,
  expect: {
    timeout: 5000,
  },
  use: {
    baseURL: 'https://demowebshop.tricentis.com/',
  },

  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },

    {
      name: 'firefox',
      use: { ...devices['Desktop Firefox'] },
    },

    {
      name: 'webkit',
      use: { ...devices['Desktop Safari'] },
    }

  ],

});
```

Slika 8. Konfiguracija testnog okruženja.[113]

4.2. Struktura i organizacija testova

Za automatsko testiranje Demo Web Shop u Playwright radnom okviru, korišten je pristup tzv. objektnog modela stranice (eng. Page object model) s ciljem boljeg strukturiranja i organizacije testova. Pojednostavljeno, Demo Web Shop ima različite stranice; stranicu za registraciju, stranicu za kupovinu, stranicu s proizvodima i sl. Ukoliko stranice predstavimo u različite objektne modele stranica pojednostavljeno je održavanje koda iz razloga što selektore i lokatore (tzv. hvatače elemenata stranice) kreiramo na jednom mjestu te je taj kod ponovno upotrebljiv i izbjegava se redundantnost.[114] U radu je zbog korištenja ovog pristupa kreirana mapa *pages* u kojoj se nalazi pet objekata stranice koji će se testirati:

- *register.page.ts*,
- *login.page.ts*,
- *categories.page.ts*,
- *shopping_cart.page.ts*,
- *product.page.ts*.

[113] https://github.com/mzilic98/diplomski-playwright/blob/main/playwright.config.ts
[114] Playwright; Page object models. URL: https://playwright.dev/docs/pom (2024-06-09)

Svaki od ovih stranica sadrže klase koje obuhvaćaju lokatore za elemente na stranici i metode za interakciju s istima.

S druge strane, mapa *tests* obuhvaća sve testne slučajeve koji koriste klase definirane u *pages* datotekama za izvođenje testova. Mapa *tests* također se dijeli na više strukturiranih datoteka radi organizacije i preglednosti, te se u svakoj datoteci (može se reći testnoj skripti) nalaze specifični testni slučajevi koji koriste *pages* metode za interakciju s aplikacijom i provjeru očekivanih rezultata. U radu su za izvođenje testova kreirane sljedeće testne datoteke:

- *register.spec.ts*,
- *login.spec.ts*,
- *categories.spec.ts*,
- *shopping_cart.spec.ts*,
- *product.spec.ts*.

Spomenute datoteke služe za testiranje procesa registracije i prijave korisnika, ključne funkcionalnosti produkata i kategorija te upravljanja stavkama u košarici s ciljem provjere pravilnog funkcioniranja aplikacije kroz interakciju s elementima stranice.

4.3. Izrada testova

4.3.1. Definiranje navigacije

Većina testova započinje s navigiranjem stranice do određenog URL-a pomoću *page.goto()* metode. Navigacija omogućuje da testovi interaktivno djeluju s elementima stranice. Primjer definiranja navigacije prikazan je na Slici 9., a Playwright će u ovom slučaju pričekati da se stranica učita prije izvršavanja testova na istoj.[115]

```
await page.goto('https://demowebshop.tricentis.com/');
```

Slika 9. Primjer definiranja navigacije.

[115] Playwright; Writing tests. URL:https://playwright.dev/docs/writing-tests (2024-06-09)

Navigacija se definirala na svakom *pages* objektu za testiranje aplikacije. S obzirom da je osnovni URL postavljen u *config* datoteci, koristimo ga prilikom definiranja navigacije. Tako je za potrebe testiranja procesa registracije korištena sljedeća putanja:

```
async goto() {
  await this.page.goto('/register');
}
```

Slika 10. Navigacija na stranicu za proces registracije.[116]

Na Slici 10. metoda *page.goto()* omogućuje da testovi interaktivno djeluju s elementima stranice za proces registracije. Osim toga, korištena je i *async/await* sintaksa. TypeScript podržava korištenje asinkrone funkcije koje imaju prefiks s ključnom riječi *async* te podrazumijevaju da će se u budućnosti "nešto dogoditi", dok *await* zaustavlja izvršavanje funkcije dok se ne ispuni "obećanje" (eng. *Promise*) vraćanja asinkrone funkcije i dobije vrijednost iz tog "obećanja".[117]

Istim principom, definirana je navigacija za stranice za proces prijave korisnika, proizvoda, kategorija i upravljanje stavkama u košarici prikazano na slikama 11., 12., 13. i 14.:

```
async goto() {
  await this.page.goto('/login');
}
```

Slika 11. Navigacija na stranicu za prijavu korisnika.[118]

```
async goto() {
  await this.page.goto('/blue-jeans');
}
```

[116] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/pages/register.page.ts#L61-L63
[117] TypeScript. TypeScript 1.7. URL: https://www.typescriptlang.org/docs/handbook/release-notes/typescript-1-7.html (2024-08-20)
[118] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/pages/login.page.ts#L47-L49

Slika 12. Navigacija na stranicu za testiranje proizvoda.[119]

```
async goto() {
  await this.page.goto('/');
}
```

Slika 13. Navigacija do stranice za testiranje kategorija.[120]

```
async goto() {
  await this.page.goto('/cart', {
    waitUntil: 'load',
  });
}
```

Slika 14. Navigacija do stranice za testiranje proizvoda u košarici.[121]

Uz standardno definiranje navigacije, za navigaciju do stranice za testiranje proizvoda u košarici (Slika 14.) koristila se i metoda *waitUntil: "load"*, koja omogućava da se stranica u potpunosti učita prije nego što se krenu izvršavati ostali koraci testnih slučajeva.[122] Ista poboljšava stabilnost i točnost testova da ne "padnu" (eng. fail test) zbog isteka čekanja (eng. timeout) na učitavanju stranice.

4.3.2. Definiranje lokatora

Da bi Playwright mogao oponašati korisničke radnje, potrebno je prethodno locirati elemenate koji su potrebni za izvođenje radnji. Lokatori (eng. Locators) predstavljaju način pronalaženja

[119] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/pages/product.page.ts#L34-L36
[120] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/pages/categories.page.ts#L59-L61
[121] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/pages/shopping_cart.page.ts#L71-L74
[122] Playwright; Page. URL: https://playwright.dev/docs/api/class-page (2024-06-09)

elemenata na stranici.[123] Neki od preporučenih lokatora koji se mogu koristiti tijekom pisanja testova u Playwright-u su:

- *page.getByRole()* - za lociranje elemenata na osnovu atributa dostupnosti
- *page.getByText()* - za lociranje elemenata na osnovu teksta kojeg elementi sadrže
- *page.getByLabel()* - za lociranje elemenata na osnovu teksta povezanog oznakom (label)
- *page.getByAltText()* - najčešće se koristi za lociranje slika na osnovu teksta alternativnog opisa (alt text)
- *page.getByTitle()* - za lociranje elemenata na osnovu vrijednosti atributa title
- *page.getByTestId()* - za lociranje elemenata na osnovu vrijednosti atributa data-testid[124]

Za svaki page objekt model bilo je potrebno definirati lokatore koji jednom definirani, mogu se naknadno pozvati i upotrijebiti za bilo koji testni slučaj. U praksi, unutar svake POM datoteke deklariraju se varijable u klasi kojima se dodijeli tip podatka te mu se pridruže vrijednosti. Primjerice, kao što je prikazano na Slici 15., u *Register* POM datoteci se definirala klasa *Register*, dodane su varijable vezane za Register stranicu (npr. *registerBtn*, *genderLabel* itd.) te im se dodijelio tip podatka *Locator*.

```
export class Register {
  readonly page: Page;
  readonly registerBtn: Locator;
```

Slika 15. Dodjeljivanje klase Register i dodjeljivanje Locator tip podatka varijabli registerBtn.[125]

Prikazano na Slici 16., drugi dio odnosi se na metodu *constructor* u kojoj definiramo "lokaciju" elementa na stranici da bi ih Playwright mogao naći.

[123] Playwright; Locators. URL: https://playwright.dev/docs/locators (2024-06-09)
[124] Playwright; Locators. URL: https://playwright.dev/docs/locators (2024-06-09)
[125] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/pages/register.page.ts#L3-L30

```
constructor(page: Page) {
  this.page = page;
  this.registerBtn = page.locator('#register-button');
```

Slika 16. Pronalaženje elementa gumba za registraciju preko jedinstvenog identifikatora u constructor metodi.[126]

Istim principom dodijeljene su vrijednosti svim POM datotekama.[127]

4.3.3. Grupiranje testnih slučajeva i testne kuke (eng. Test Hooks)

Za grupiranje testnih slučajeva u Playwright-u koristi se funkcija *test.describe* koja olakšava čitljivost i preglednost testnih slučajeva.[128] Primjerice, u radu svaka test datoteka obuhvaća jedan blok opisa svih testova (pomoću *test.describe*) koje veže ista funkcionalnost.

Pojam testnih kuka podrazumijeva funkcije u Playwright-u koje se koriste za postavljanje testnog okruženja prije ili nakon izvršavanja testnih slučajeva:

- *test.beforeEach* - određeni blok koda će se izvršiti prije svakog testa u testnom okruženju,
- *test.afterEach* - određeni blok koda će se izvršiti nakon svakog testa u testnom okruženju,
- *test.beforeAll* - određeni blok koda će se izvršiti prije što su svi testovi u testnom okruženju izvršen,i
- *test.afterAll* - određeni blok koda će se izvršiti nakon što su svi testovi u testnom okruženju izvršeni.[129]

Tako su u radu testni slučajevi za provjeru registracije korisnika grupirani u jedan blok opisa koji naglašava svrhu ovih testova; provjera da registracija na Demo Web Shop aplikaciji radi kako je očekivano. Unutar bloka *test.describe*, koristi se funkcija *test.beforeEach*, što

[126] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/pages/register.page.ts#L32-L59
[127] https://github.com/mzilic98/diplomski-playwright/tree/main/pages
[128] Playwright. Playwright Test. URL: https://playwright.dev/docs/api/class-test (2024-06-09)
[129] Playwright. Playwright Test. URL: https://playwright.dev/docs/api/class-test (2024-06-09)

podrazumijeva da će prije svakog pokretanja testova u *Register* datoteci, Playwright izvršiti naredbu navigacije do stranice za registraciju (Slika 17.).

```
test.describe('Verify that registration of the Demo web shop page works as intended', () => {
  let register: Register;

  test.beforeEach(async ({ page }) => {
    register = new Register(page);
    await register.goto();
  });
```

Slika 17. Grupiranje testova za testiranje registracije.[130]

Istim principom grupirani su testni slučajevi za preostale datoteke: testovi za prijavu korisnika (Slika 18.), testiranje proizvoda (Slika 19.) i kategorija (Slika 20.).

```
test.describe('Verify that login of the Demo web shop page works as intended', () => {
  let login: Login;
  test.beforeEach(async ({ page }) => {
    login = new Login(page);
    await login.goto();
  });
```

Slika 18. Grupiranje testova za testiranje prijave korisnika.[131]

```
test.describe('Verify that the product behavior of the Demo web shop page work as intended', () => {
  let product: Product;
  test.beforeEach(async ({ page }) => {
    product = new Product(page);
    await product.goto();
  });
```

Slika 19. Grupiranje testova za testiranje proizvoda.[132]

[130] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/register.spec.ts#L5-L11
[131] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/login.spec.ts#L5-L10

[132] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/product.spec.ts#L4-L9

```
test.describe
  ('Verify that the categories of the Demo web shop page work as intended', () => {
  let categories: Categories;
  test.beforeEach(async ({ page }) => {
    categories = new Categories(page);
    await categories.goto();
  });
```

Slika 20. Grupiranje testova za testiranje kategorija.[133]

Kao što je prikazano na Slici 21., testovi košarice također su grupirani u funkciji *test.describe* i koriste funkciju *test.beforeEach*, no za razliku od prijašnjih primjera, unutar *test.beforeEach* postavljeni su dodatni koraci koji se trebaju izvršiti prije pokretanja specifičnih testnih slučajeva: navigacija na stranicu proizvoda *3rd album*, klik na gumb za dodavanje proizvoda u košaricu, čekanje jedne sekunde da se stranica potpuno učita i preusmjeravanje na stranicu košarice. Ovaj pristup omogućava uštedu vremena pisanja koda, isto početno stanje prije izvršavanja testnih slučajeva, bolju organizaciju i jednostavniju održivost koda pri testiranju funkcionalnosti košarice.

```
test.describe('Verify that the shopping cart behavior of the Demo web shop page work as intended', () => {
  let shopping_cart: Shopping_cart;
  test.beforeEach(async ({ page }) => {
    shopping_cart = new Shopping_cart(page);

    await page.goto('/album-3');
    await shopping_cart.firstItemAddToCartBtn.click();
    await page.waitForTimeout(1000);
    await shopping_cart.goto();
  });
```

Slika 21. Grupiranje testova za testiranje košarice.[134]

4.3.4. Testni slučajevi

Za kreiranje testnih slučajeva, potrebno je poznavati osnovne radnje i tvrdnje.

Osnovne radnje

[133] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/categories.spec.ts#L4-L10
[134] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/shopping_cart.spec.ts#L4-L13

Playwright omogućava velik broj radnji koje mogu oponašati korisničko ponašanje tijekom pisanja testova, a neke od najosnovnijih su:

- locator.click() - klikne na element,
- locator.check() - osigurava da je potvrdni okvir (eng. checkbox) označen,
- locator.uncheck() - poništava označeni checkbox,
- locator.hover() - pomiče miš preko elementa,
- locator.fill() - popunjava polje obrasca.[135]

Tvrdnje (eng. Assertions)

Kako bi Playwright utvrdio da stanje aplikacije odgovara očekivanom ponašanju, potrebno je uključiti testne tvrdnje u obliku funkcije očekivanja. Tvrdnjom se utvrđuje očekivano ponašanje na način da se pozove funkcija *expect(value)* i odabere podudaranje koje odražava očekivanje.[136] Neke od najosnovnijih tvrdnji su sljedeće:

- *await expect(locator).toBeVisible()* - očekuje da je element vidljiv,
- *await expect(locator).toBeHidden()* - očekuje da je element sakriven,
- *await expect(locator).toBeEnabled()* - očekuje da je element aktivan,
- *await expect(page).toHaveURL()* - očekuje da se stranica ima URL,
- *await expect(locator).toBeChecked()* - očekuje da je potvrdni okvir označen.[137]

S obzirom da je u *config* datoteci definiran vremenski rok za očekivanja, ukoliko Playwright ne pronađe podudaranje koje odražava očekivanje u vremenskom okviru od pet sekundi, test će biti neuspješan.

Testni slučajevi za proces registracije

Za testiranje procesa registracije korisnika, kreirano je šest testnih slučajeva. Cilj prvog testa je provjera ispravnosti validacije forme za registraciju (Slika 22.). Odnosno, provjera jesu li sva polja za unos informacija o korisniku obavezna. U prvom testu je postavljeno da Playwright klikne na gumb za registraciju i zatim očekuje elemente da budu vidljivi.

[135] Playwright. Writing tests. URL:https://playwright.dev/docs/writing-tests (2024-06-09)
[136] Isto.
[137] Playwright. Assertions. URL: https://playwright.dev/docs/test-assertions (2020-08-20)

```
test('TEST 1 Verify that validation of the register form works as intended',
async () => {

  await register.registerBtn.click();
  await expect(register.firstNameValidationElement).toBeVisible();
  await expect(register.lastNameValidationElement).toBeVisible();
  await expect(register.emailValidationElement).toBeVisible();
  await expect(register.passwordValidationElement).toBeVisible();
  await expect(register.confirmPasswordValidationElement).toBeVisible();
  await expect(register.registerBtn).toBeVisible();
});
```

Slika 22. Provjera ispravnosti validacije forme za registraciju.[138]

Drugi testni slučaj želi potvrditi da se korisnik može uspješno registrirati na Demo Web Shop aplikaciju ukoliko unese ispravne podatke (Slika 23.).

```
test('TEST 2 Verify that user can successfully register if valid data is given',
  async ({ page }) => {

    test.slow();
    await register.femaleCheckbox.click();
    await register.firstNameInput.fill('Marija');
    await register.lastNameInput.fill('QA');
    await generateUniqueUserEmail(page, 'mpw4');
    await expect(page).toHaveURL(
      '/registerresult/1'
    );
    await expect(register.registrationCompleted).toBeVisible();
    await expect(register.continueBtn).toBeVisible();
  }
);
```

Slika 23. Provjera uspješne registracije s ispravno unesenim podacima.[139]

Prikazano na Slici 23., u formu za registraciju ovaj test će kliknuti na potvrdni okvir da je u pitanju ženska osoba, zatim će u polje za upis imena upisati Marija i prezime QA te odgovarajuću e-mail adresu. Test će završiti nakon što prepozna da su poruka o uspješnoj registraciji i gumb *Continue* vidljivi. Uzimajući u obzir da bi ovaj test svaki sljedeći put pao (eng. fail) ukoliko je već jednom izvršen iz razloga što aplikacija dopušta samo jednom registraciju s istom e-mail adresom, kreirana je funkcija koja generira jedinstvenu e-mail adresu (Slika 24.).

[138] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/register.spec.ts#L13-L23
[139] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/register.spec.ts#L25-L39

```ts
import { expect } from '@playwright/test';
import { Register } from '../../pages/register.page';

export async function generateUniqueUserEmail(page, email: string): Promise<string> {
  let number = 1;


  while (true) {
    const checkedEmail = `${email}+${number}@gmail.com`;

    await page?.fill('#Email', checkedEmail);
    await page?.fill('#Password', 'hjrks987');
    await page?.fill('#ConfirmPassword', 'hjrks987');
    await page?.click('#register-button');

    const errorMessageElement = await page.waitForSelector('.message-error', { timeout: 2000 }).catch(() => null);
    const registrationCompletedElement = await page.waitForSelector('.result', { timeout: 2000 }).catch(() => null);

    if (errorMessageElement) {
      const errorMessage = await errorMessageElement.textContent('.message-error');
      if (errorMessage?.includes('The specified email already exists')) {
        number++;
        await page.fill('#Email', '');
      }
    } else if (registrationCompletedElement) {

      const registrationCompletedText = await registrationCompletedElement.textContent('.result');
      if (registrationCompletedText?.includes('Your registration completed')) {
        return checkedEmail;
      }
    } else {

      throw new Error('Neither error message nor registration completion message was found.');
    }

  }

}
```

Slika 24. Funkcija koja generira jedinstvenu e-mail adresu.[140]

Funkcija prikazana na Slici 24. će koristiti postavljeni e-mail *mpw4* te će dodati na tu e-mail adresu *+(broj)* kako bi pronašla e-mail adresu koja dosad nije korištena. Nakon toga će upisati lozinku *hjrks987* i kliknuti na gumb za registraciju. Ukoliko sustav izbaci poruku da e-mail adresa već postoji, funkcija omogućava ponovni pokušaj s drugim rednim brojem sve dok ne pronađe adresu koja dosad nije postojala. Nakon što se pojavi poruka da je registracija uspješna, test će uspješno završiti. Može se reći da je nedostatak ove funkcije potencijalno dugo trajanje na izvršavanje testa (potrebno je vrijeme da bi se pronašla nepostojeća e-mail adresa), stoga je poželjno s vremena na vrijeme mijenjati inicijalno postavljenu e-mail adresu.

Treći testni slučaj pokriva situaciju kada korisnik unosi nevaljanu e-mail adresu. Ovaj testni slučaj prikazan na Slici 25. koristi iste korake kao i prethodni, no ne koristi funkciju za generiranje jedinstvene e-mail adrese, nego upisuje nevažeću e-mail adresu: *invalidEmail#$%&*. Na kraju, očekuje da će poruka *Wrong email* biti vidljiva te da poruka o uspješnoj registraciji neće biti vidljiva.

[140] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/utils/helpers/generateUniqueUserEmail.ts

```
test('TEST 3 Verify email input error message if the email form is invalid',
async () => {

  await register.femaleCheckbox.click();
  await register.firstNameInput.fill('Marija');
  await register.lastNameInput.fill('QA');
  await register.emailInput.fill('invalidEmail#$%&');
  await register.passwordInput.fill('dipl987');
  await register.confirmPasswordInput.fill('dipl987');
  await register.registerBtn.click();

  await expect(register.invalidEmail).toBeVisible();
  await expect(register.registrationCompleted).toBeHidden();
});
```

Slika 25. Provjera da se korisnik ne može registrirati s nevažećom e-mail adresom.[141]

Nadalje, četvrti testni slučaj (Slika 26.) je sličan trećem, no ovim testnim slučajem se provjerava da se korisnik ne može uspješno ulogirati ukoliko koristi e-mail adresu koja je već registrirana. Koristi iste korake kao i prethodni; klik na potvrdni okvir, unošenje imena i prezimena, lozinke te klikanje gumba za registraciju, izuzev e-mail adrese koja je ovog puta ispravna, no već registrirana u aplikaciji. Na kraju, postavljeno je da se očekuje da će poruka da e-mail već postoji biti prikazana te da poruka o uspješnoj registraciji neće biti vidljiva.

```
test('TEST 4 Verify email input error message if the email form is already in use',
async () => {

  await register.femaleCheckbox.click();
  await register.firstNameInput.fill('Marija');
  await register.lastNameInput.fill('QA');
  await register.emailInput.fill('mpw+1@gmail.com');
  await register.passwordInput.fill('dipl987');
  await register.confirmPasswordInput.fill('dipl987');
  await register.registerBtn.click();

  await expect(register.emailInUse).toBeVisible();
  await expect(register.registrationCompleted).toBeHidden();
});
```

Slika 26. Provjera da se korisnik ne može registrirati s e-mail adresom koja je već u upotrebi.[142]

[141] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/register.spec.ts#L41-L54
[142] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/register.spec.ts#L56-L69

Petim testnim slučajem provjeravamo da aplikacija neće dozvoliti registraciju ukoliko je korisnik unio lozinku koja nema odgovarajući minimalan broja znakova (Slika 27.).

Test će izvršiti unos lozinke *123* te očekivati poruku da lozinka mora imati minimalno šest znakova kako bi bila prihvaćena.

```
test('TEST 5 Verify password input field for minimum characters allowed',
async () => {

  await register.passwordInput.fill('123');
  await register.confirmPasswordInput.fill('123');
  await expect(register.passwordMinimum).toBeVisible();
});
```

Slika 27. Provjera da lozinka mora imati minimalan broj znakova kako bi bila prihvaćena.[143]

Zadnji test za proces registracije također stavlja fokus na ispravnost lozinke. Odnosno, cilj ovog testa je provjera da se korisnik neće moći registrirati ukoliko su u poljima za unos lozinke i potvrde lozinke različite vrijednosti te očekuje da će se prikazati poruka da se lozinke ne podudaraju (Slika 28.).

```
test('TEST 6 Verify error message if password is not the same in both password fields',
async () => {

  await register.maleCheckbox.click();
  await register.firstNameInput.fill('Marko');
  await register.lastNameInput.fill('QA');
  await register.emailInput.fill('markopw+2@gmail.com');
  await register.passwordInput.fill('diplomski23');
  await register.confirmPasswordInput.fill('diplomski35');
  await register.registerBtn.click();

  await expect(register.passwordNotMatch).toBeVisible();
  await expect(register.registrationCompleted).toBeHidden();
});
```

Slika 28. Provjera da se prikazuje poruka o pogrešci ukoliko vrijednosti u poljima za lozinku nisu jednake.[144]

[143] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/register.spec.ts#L71-L77
[144] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/register.spec.ts#L79-L92

Testni slučajevi za prijavu korisnika

Kreirano je devet testnih slučajeva za prijavu korisnika na Demo Web Shop aplikaciju. Prvi test prikazan na Slici 29. obuhvaća provjeru ispravnosti validacije forme za prijavu korisnika. Klikom na gumb za prijavu, bez unesenih podataka, očekuje se da će poruka da je prijava neuspješna biti vidljiva.

```
test('TEST 1 Verify that validation of the login form works as intended',
async ({}) => {

  await login.loginBtn.click();
  await expect(login.logInFailedMessage).toBeVisible();
});
```

Slika 29. Provjera ispravnosti validacije forme za prijavu korisnika.[145]

Drugi testni slučaj provjerava da se korisnik može uspješno prijaviti na Demo Web Shop aplikaciju ukoliko su uneseni ispravni podaci (Slika 30.). Testni koraci su sljedeći: unijeti e-mail adresu u polje za e-mail, ispravnu lozinku i kliknuti na gumb za prijavu. Očekuje se da će poruka o neuspješnoj prijavi biti skrivena, zatim da će se na aplikaciji pojaviti e-mail adresa ulogiranog korisnika kojim potvrđujemo uspješnu prijavu te postoji dodatna provjera da je korisnik ostao na točnoj URL adresi.

```
test('TEST 2 Verify that user can successfully log in if valid data is given',
async ({ page }) => {

  await login.emailInput.fill('mpw+1@gmail.com');
  await login.passwordInput.fill('dipl987');
  await login.loginBtn.click();
  await page.pause();
  await expect(login.logInFailedMessage).toBeHidden();
  await expect(login.loggedInUserMail).toBeVisible();
  await expect(page).toHaveURL(
    '/'
  );
});
```

Slika 30. Provjera uspješne prijave s ispravnim podacima.[146]

[145] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/login.spec.ts#L12-L17
[146] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/login.spec.ts#L19-L31

Treći i četvrti testni slučajevi su slični iz razloga što oba provjeravaju scenarij da prijava neće biti uspješna ukoliko; u trećem testu (Slika 31.) unesemo netočnu lozinku ili u četvrtom testu (Slika 32.) unesemo netočan e-mail.

Postavljeno očekivanje je u oba slučaja da će biti prikazana poruka o neuspješnoj registraciji, da će biti skriven korisnički e-mail kojim bi inače potvrdili uspješnu prijavu, te dodatno očekivanje da se korisnik i dalje nalazi na *login* stranici.

```
test('TEST 3 Verify that the login will not be successful if the wrong password is entered',
async ({ page }) => {

  await login.emailInput.fill('mpw+1@gmail.com');
  await login.passwordInput.fill('diploomskiii');
  await login.loginBtn.click();
  await expect(login.logInFailedMessage).toBeVisible();
  await page.pause();
  await expect(login.loggedInUserMail).toBeHidden();
  await expect(page).not.toHaveURL(
    '/'
  );
});
```

Slika 31. Provjera neuspješne prijavi s netočnom lozinkom.[147]

```
test('TEST 4 Verify that the login will not be successful if the wrong email is entered',
async ({ page }) => {

  await login.emailInput.fill('mpw.diplomski@gmail.com');
  await login.passwordInput.fill('dipl987');
  await login.loginBtn.click();
  await expect(login.logInFailedMessage).toBeVisible();
  await expect(login.loggedInUserMail).toBeHidden();
  await expect(page).toHaveURL(
    '/login'
  );
});
```

Slika 32. Provjera neuspješne prijave s netočnom e-mail adresom.[148]

Peti testni slučaj obuhvaća provjeru da se korisnik može uspješno odjaviti iz aplikacije (Slika 33.). Kako bi se to dogodilo potrebno je prvo klasičnim koracima se prijaviti u aplikaciju: unijeti ispravnu e-mail adresu i lozinku te kliknuti na *Login* gumb. Nakon toga je postavljeno očekivanje da se korisnik ulogirao i da je e-mail adresa u aplikaciji vidljiva. Potom se očekuje

[147] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/login.spec.ts#L33-L45
[148] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/login.spec.ts#L47-L58

da će element za odjavu, koji se nalazi na navigacijskoj traci, biti aktivan te će se na njega kliknuti. Zadnjim tvrdnjama očekujemo da isti element neće biti više vidljiv te će umjesto njega u navigacijskoj traci postati vidljiv element za prijavu korisnika.

```
test('TEST 5 Verify that the Log out feature works as expected',
async ({ page }) => {

  await login.emailInput.fill('mpw+1@gmail.com');
  await login.passwordInput.fill('dipl987');
  await login.loginBtn.click();
  await expect(login.loggedInUserMail).toBeVisible();
  await expect(page).toHaveURL(
    '/'
  );
  await expect(login.logOutNavBar).toBeEnabled;
  await login.logOutNavBar.click();
  await expect(login.logOutNavBar).not.toBeVisible;
  await expect(login.logInNavBar).toBeVisible;
});
```

Slika 33. Provjera da se korisnik može uspješno odjaviti iz aplikacije.[149]

Testnim slučajevima 6, 7 i 8 provjeravamo ispravnost funkcionalnosti "Zaboravljena lozinka". U sva tri slučaja koristimo iste korake, ali s drugačijim vrijednostima i očekivanjima. Testni koraci su sljedeći: klikni na *zaboravljena lozinka*, očekuj da će se korisnik preusmjeriti na URL adresu za oporavak lozinke i da će naslov na toj URL adresi biti *Password recovery*, zatim upiši e-mail adresu za oporavak i klikni na *Recover* gumb. Međutim, ono što razlikuje ova tri testna slučajeva je što se provjeravaju različiti scenariji korištenja funkcionalnosti "Zaboravljena lozinka". U šestom testu (Slika 34.), provjerava se hoće li proces ove funkcionalnosti biti uspješan ukoliko je unesena valjana e-mail adresa. Iz tog razloga je za zadnju tvrdnju postavljeno da se očekuje prikaz poruke da je e-mail s uputama (uspješno) poslan na e-mail adresu. Nasuprot tome, u sedmom testu (Slika 35.) provjeravamo hoće li proces biti neuspješan ukoliko je unesena nevažeća e-mail adresa. Tvrdnja na kraju ovog testnog slučaja očekuje da će se prikazati poruka da je u pitanju krivi e-mail. Te u konačnici, u osmom testnom slučaju (Slika 36.), provjeravamo hoće li funkcionalnost ispravno raditi za neregistrirane korisnike; očekuje se da će se prikazati poruka da e-mail nije pronađen.

[149] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/login.spec.ts#L60-L74

```
test('TEST 6 Verify that the Forgot Password feature works as expected if valid email is given',
async ({ page }) => {

  await login.forgotPasswordLabel.click();
  await expect(page).toHaveURL(
    '/passwordrecovery'
  );
  await expect(login.passwordRecovery).toBeVisible();
  await login.recoveryEmail.fill('mzqa57@gmail.com');
  await login.recoverBtn.click();
  await expect(login.successMessage).toBeVisible();
});
```

Slika 34. Provjera funkcionalnosti "Zaboravljena lozinka" s unesenom valjanom e-mail adresom.[150]

```
test('TEST 7 Verify that the Forgot Password feature works as expected if invalid email is given',
async ({ page }) => {

  await login.forgotPasswordLabel.click();
  await expect(page).toHaveURL(
    '/passwordrecovery'
  );
  await expect(login.passwordRecovery).toBeVisible();
  await login.recoveryEmail.fill('mzqa0%&"gmail');
  await login.recoverBtn.click();
  await expect(login.wrongEmailMessage).toBeVisible();
});
```

Slika 35. Provjera funkcionalnosti "Zaboravljena lozinka" s unesenom nevažećom e-mail adresom.[151]

```
test('TEST 8 Verify that the Forgot Password feature works as expected for nonregistered user',
async ({ page }) => {

  await login.forgotPasswordLabel.click();
  await expect(page).toHaveURL(
    '/passwordrecovery'
  );
  await expect(login.passwordRecovery).toBeVisible();
  await login.recoveryEmail.fill('mzqa57+diplomski999@gmail.com');
  await login.recoverBtn.click();
  await expect(login.emailNotFoundMessage).toBeVisible();
});
```

Slika 36. Provjera funkcionalnosti "Zaboravljena lozinka" za neregistrirane korisnike.[152]

[150] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/login.spec.ts#L76-L87
[151] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/login.spec.ts#L89-L100
[152] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/login.spec.ts#L102-L113

Zadnji testni slučaj u *Login* datoteci provjerava da će klik na gumb za Registraciju na stranici za prijavu pravilno preusmjeriti korisnika na stranicu za registraciju (Slika 37.).

```
test('TEST 9 Verify that the register button on the login page redirects the user to the Register page',
async ({  page }) => {

  await login.registerBtn.click();
  await expect(page).toHaveURL(
    '/register'
  );
});
```

Slika 37. Provjera preusmjeravanja korisnika sa stranice za prijavu na stranicu za registraciju.[153]

Testni slučajevi za testiranje proizvoda

Za testiranje proizvoda kreirano je šest testnih slučajeva. Prvim testom nastoji se provjeriti da su ključne informacije o proizvodu vidljive korisniku, stoga su postavljena samo očekivanja: da su naziv proizvoda, cijena i slika vidljivi te da je proizvod dostupan (Slika 38.).

```
test('TEST 1 Validating Essential Product Information',
async () => {

  await expect(product.productTitle).toBeVisible();
  await expect(product.productPhoto).toBeVisible();
  await expect(product.availability).toContainText('In stock');
  await expect(product.productPrice).toBeVisible();
});
```

Slika 38. Provjera ključnih informacija o proizvodu.[154]

Drugi testni slučaj provjerava da korisnik može promijeniti količinu proizvoda na aplikaciji. Kao što je prikazano na Slici 39., prvotno se očekuje da su oznaka za količinu i polje za unos količine vidljivi. Zatim će se u polje za unos količine unijeti broj "3" te se nakon toga provjerava da polje za unos sadrži broj "3" što potvrđuje da je unos uspješan.

[153] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/login.spec.ts#L115-L121
[154] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/product.spec.ts#L11-L18

```
test('TEST 2 Verify that the user can change the quantity of the product',
async () => {

  await expect(product.qtyLabel).toBeVisible();
  await expect(product.qtyInput).toBeVisible();
  await product.qtyInput.fill('3');
  await expect(product.qtyInput).toHaveValue('3');
});
```

Slika 39. Provjera mijenjanja količine proizvoda.[155]

Treći testni slučaj (Slika 40.) provjerava hoće li gumb za dodavanje proizvoda u košaricu ispravno raditi na stranici proizvoda. Kako bi to provjerili, test će prvo kliknuti na *Add to cart* gumb te će pomaknuti pokazivač miša na košaricu i na kraju očekivati poruku da je proizvod uspješno dodan u košaricu.

```
test('TEST 3 Verify that the Add to cart button works as expected',
async () => {

  await product.addToCartBtn.click();
  await product.shoppingCart.hover();
  await expect(product.addToCartSucces).toBeVisible();
});
```

Slika 40. Provjera da *Add to cart* gumb dodaje proizvod u košaricu.[156]

Testni slučajevi pod brojevima 4, 5 i 6 koriste istu sintaksu za testiranje komponenata na stranici proizvoda, no testiraju se različite funkcionalnosti te stoga imaju drugačije povezane elemente i unesene vrijednosti.

Četvrti testni slučaj (Slika 41.) provjerava funkcionalnost *Email a friend* gumba na stranici proizvoda na način da klikne na gumb i očekuje da će isti korisnika preusmjeriti na stranicu za slanje e-maila prijatelju.

[155] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/product.spec.ts#L20-L27
[156] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/product.spec.ts#L29-L35

```
test('TEST 4 Verify that the Email a friend button on the product page is working correctly',
async ({ page }) => {

  await product.emailAFriendBtn.click();
  await expect(page).toHaveURL(
    '/productemailafriend/36'
  );
});
```

Slika 41. Provjera da E-mail a friend gumb na stranici proizvoda ispravno funkcionira.[157]

Slično tome, testiranje *Compare list* funkcionalnosti odnosi se na provjeru da će klik na *Add to compare list* gumb preusmjeriti korisnika na stranicu za usporedbu proizvoda (Slika 42.).

```
test('TEST 5 Verify that the Compare list button on the product page is working correctly',
async ({  page }) => {

  await product.addToCompareListBtn.click();
  await expect(page).toHaveURL(
    '/compareproducts'
  );
});
```

Slika 42. Provjera da Add to compare list gumb na stranici proizvoda ispravno funkcionira.[158]

Zadnji testni slučaj za testiranje stranice proizvoda uključuje provjeru Add your review linka. Cilj ovog testa je provjeriti da će klik na Add your review link ispravno preusmjeriti korisnika na stranicu na kojoj će korisnik moći dodati recenziju za proizvod (Slika 43.).

```
test('TEST 6 Verify that the Add your Review link on the product page is clickable',
async ({ page }) => {

  await product.addReview.click();
  await expect(page).toHaveURL(
    '/productreviews/36'
  );
});
```

Slika 43. Provjera da je Add your Review link klikabilan na stranici proizvoda.[159]

[157] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/product.spec.ts#L37-L44
[158] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/product.spec.ts#L46-L53
[159] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/product.spec.ts#L55-L62

Testni slučajevi za testiranje kategorija proizvoda

Za testiranje kategorija izrađeno je šest testnih slučajeva. Prvi testni slučaj provjerava da korisnik može pristupiti svim kategorijama proizvoda. Stoga je kreiran test 1 (Slika 44.) koji provjerava jesu li kategorije knjiga, računala, elektronike, odjeće i obuće, digitalnih proizvoda, nakita i poklon kartice aktivni odnosno klikabilni na stranici Demo Web Shop-a.

```
test('TEST 1 Verify that all categories are clickable',
async ({}) => {
  await expect(categories.books).toBeEnabled();
  await expect(categories.computers).toBeEnabled();
  await expect(categories.electronics).toBeEnabled();
  await expect(categories.apparelAndShoes).toBeEnabled();
  await expect(categories.digitalDownloads).toBeEnabled();
  await expect(categories.jewelry).toBeEnabled();
  await expect(categories.giftcards).toBeEnabled();
});
```

Slika 44. Provjera jesu li kategorije klikabilne.[160]

Drugim testnim slučajem želi se provjeriti postojanost proizvoda u kategoriji (Slika 45.). Kako bismo provjerili postojanost proizvoda, test će prvotno kliknuti na kategoriju elektronike te odabrati opciju mobilnih telefona. Test će provjeriti da će korisnik nakon toga biti preusmjeren na ispravnu URL adresu te će pričekati da se stranica učita prije nego počne prebrojavati broj proizvoda. Zadnjom tvrdnjom očekujemo da je broj proizvoda veći od nula i time utvrđujemo postojanost proizvoda u kategoriji.

```
test('TEST 2 Verify the existence of products from the category.',
async ({ page }) => {
  await categories.electronics.click();
  await categories.cellphones.click();
  await expect(page).toHaveURL(
    '/cell-phones'
  );

  await page.waitForLoadState();
  const productCount = (await page.$$('.product-item')).length;
  expect(productCount).toBeGreaterThan(0);
});
```

[160] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/categories.spec.ts#L12-L21

Slika 45. Provjera postojanosti proizvoda u kategoriji.[161]

Treći testni slučaj (Slika 46.) nastoji provjeriti da proizvodi u kategoriji Mobilni uređaji imaju prikazane slike. Za izvršavanje ovog testa također će se kliknuti na kategoriju elektronike i mobilnih uređaja. Nakon toga, dodana je tvrdnja koja očekuje da je korisnik na ispravnoj URL adresi te potom pretražuje jesu li slike proizvoda *Smartphone*, *Used phone* i *Phone Cover* vidljive u aplikaciji.

```
test('TEST 3 Verify that the products have photos on them.',
async ({ page }) => {

  await categories.electronics.click();
  await categories.cellphones.click();
  await expect(page).toHaveURL(
    '/cell-phones'
  );
  await expect(categories.smartphonePhoto).toBeVisible();
  await expect(categories.usedPhonePhoto).toBeVisible();
  await expect(categories.phoneCoverPhoto).toBeVisible();
});
```

Slika 46. Provjera da proizvodi imaju prikazane slike.[162]

Četvrti testni slučaj provjerava da svi proizvodi u kategoriji mobilnih uređaja imaju prikazanu cijenu (Slika 47.). Ponovno je potrebno kliknuti na kategoriju elektronike i mobilnih uređaja te potvrditi da se korisnik nalazi na ispravnoj URL adresi. Potom će test pričekati da se stranica potpuno učita prije nego izračuna broj proizvoda i potvrdi da postoji minimalno jedan proizvod na stranici. Nakon toga će izračunati koliko cijena postoji na stranici i utvrditi da su cijene veće od nula. Zadnja tvrdnja očekuje da je broj proizvoda jednak broju prikazanih cijena i time potvrđuje da svaki proizvod ima svoju cijenu.

[161] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/categories.spec.ts#L23-L34
[162] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/categories.spec.ts#L36-L47

```
test('TEST 4 Verify that all products have a price.',
async ({ page }) => {
  await categories.electronics.click();
  await categories.cellphones.click();
  await expect(page).toHaveURL(
    '/cell-phones'
  );

  await page.waitForLoadState();
  const productCount = (await page.$$('.product-item')).length;
  expect(productCount).toBeGreaterThan(0);

  const priceCheck = (await page.$$('.price.actual-price')).length;
  expect(priceCheck).toBeGreaterThan(0);
  expect(productCount).toBe(priceCheck);
});
```

Slika 47. Provjera da svaki proizvod u kategoriji ima prikazanu cijenu.[163]

Peti testni slučaj (Slika 48.) provjerava je li gumb *Add to Cart* (Dodaj u košaricu) klikabilan i može li se proizvod dodati u košaricu sa stranice iz kategorije digitalnih proizvoda. Test će prvo kliknuti na spomenutu kategoriju i provjeriti da li stranica na kojoj se korisnik treba nalaziti odgovara postavljenom očekivanom URL-u. Zatim će prvo provjeriti je li gumb *Add to cart* omogućen (klikabilan) te će nakon toga na njega kliknuti. Zadnja tvrdnja provjerava da li je košarica sa oznakom *Shopping cart (1)* vidljiva nakon što je proizvod dodan u košaricu i time potvrđuje da je proizvod sa kategorije uspješno dodan u košaricu.

```
test('TEST 5 Verify that a product can be added to the cart from the category page.',
  async ({ page }) => {
    await categories.digitalDownloads.click();
    await expect(page).toHaveURL(
      '/digital-downloads'
    );
    await expect(categories.addToCartBtn).toBeEnabled();

    await categories.addToCartBtn.click();
    await expect(categories.shoppingCart).toBeVisible();
  }
);
```

Slika 48. Provjera da se proizvod može dodati u košaricu sa stranice kategorija.[164]

[163] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/categories.spec.ts#L49-L64
[164] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/categories.spec.ts#L66-L77

Šesti test u ovoj datoteci provjerava da *Sort by* funkcionalnost ispravno sortira proizvode prema određenom kriteriju (Slika 49.). Za potrebe testiranja ove funkcionalnosti, izabrana je opcija sortiranja proizvoda prema cijeni, od najniže prema najvišoj. Test će kliknuti na kategoriju digitalnih proizvoda i potvrditi da se korisnik nalazi na ispravnoj URL adresi. Nakon toga će zabilježiti stanje stranice prije sortiranja, odnosno snimiti će trenutku sliku stranice (prije sortiranja). Iz padajućeg izbornika odabrati će opciju sortiranja prema cijeni: od najniže prema najvišoj te će ponovno snimiti sliku stranice (nakon sortiranja). Obje slike spremit će u mapu kategorija i one će omogućiti usporedbu slika i vizualnu provjeru ispravnosti sortiranja prema cijeni.

```
test(
  'TEST 6 Verify that the Sort By feature correctly sorts items based on the selected criteria.',
  async ({ page }) => {

    await categories.digitalDownloads.click();
    await expect(page).toHaveURL(
      '/digital-downloads'
    );

    await expect(page).toHaveScreenshot('before_sort_by.png');
    await categories.sortBy.selectOption('Price: Low to High');
    await expect(page).toHaveScreenshot('after_sort_by.png');
  }
);
```

Slika 49. Provjera da Sort By funkcionalnost ispravno sortira proizvode prema cijeni.[165]

Testni slučajevi za testiranje košarice

Zadnja datoteka u radu obuhvaća testne slučajeve za testiranje ključnih funkcionalnosti košarice, a među njima i obavljanje kupovine na Demo Web Shop aplikaciji. Ova datoteka obuhvaća osam testnih slučajeva.

Kao što smo prethodno vidjeli u radu, važno je napomenuti da je u funkciji *test.beforeEach* osigurano da se proizvod već nalazi u košarici prije svakog izvršavanja testnog slučaja za testiranje košarice.

[165] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/categories.spec.ts#L79-L92

Prvi testni slučaj obuhvaća provjeru da je proizvod vidljiv u košarici i da košarica nije prazna (Slika 50.). Dakle, prvi testni slučaj očekuje da će informativna poruka da je košarica prazna biti skrivena te da će proizvod *3rd album* koji je definiran u POM datoteci biti vidljiv.

```
test('TEST 1 Verify that the items are visible in the shopping cart',
async ({}) => {

  await expect(shopping_cart.informativeMessage).toBeHidden();
  await expect(shopping_cart.firstItemInCart).toBeVisible();
});
```

Slika 50. Provjera vidljivosti proizvoda 3rd Album u košarici.[166]

Drugi testni slučaj provjerava ima li isti proizvod *3rd Album* vidljivu cijenu dok se nalazi na stranici košarice (Slika 51.). Postavljena je tvrdnja koja očekuje da će cijena biti vidljiva.

```
test('TEST 2 Verify that the item price is visible in the shopping cart',
async ({}) => {

  await expect(shopping_cart.firstItemPrice).toBeVisible();
});
```

Slika 51. Provjera da je cijena proizvoda vidljiva u košarici.[167]

Treći testni slučaj provjerava da zbroj cijena dvaju proizvoda odgovara ukupnom iznosu u košarici (Slika 52.). S obzirom da će prije izvršavanja testa proizvod 3rd Album biti u košarici, u trećem testnom slučaju potrebno je dodati još jedan proizvod kako bi mogli zbrojiti cijene i usporediti odgovara li zbroj ukupnom iznosu u košarici. Testni koraci za izvršavanje ovog scenarija su sljedeći: simulirati navigaciju korisnika do kategorije digitalnih proizvoda, kliknuti na drugi proizvod (*Music 2*) i dodati ga u košaricu. Test će potom pričekati jednu sekundu kako bi dali vremena aplikaciji da odradi zadatak dodavanja proizvoda u košaricu te će preusmjeriti korisnika na URL adresu košarice. Tvrdnjama očekujemo da su cijene prvog i drugog proizvoda vidljive te da ukupan iznos u košarici odgovara očekivanom iznosu: *11*.

[166] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/shopping_cart.spec.ts#L15-L20
[167] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/shopping_cart.spec.ts#L22-L26

```
test('TEST 3 Verify that the sum of two items is equal to the subtotal',
async ({ page }) => {

  await page.goto(
    '/digital-downloads'
  );
  await shopping_cart.secondItem.click();
  await shopping_cart.secondItemAddtoCartBtn.click();
  await page.waitForTimeout(1000);
  await shopping_cart.goto();
  await expect(shopping_cart.firstItemPrice).toBeVisible();
  await expect(shopping_cart.secondItemPrice).toBeVisible();
  await expect(shopping_cart.subtotal).toHaveText('11.00');
});
```

Slika 52. Provjera da zbroj cijena dvaju proizvoda odgovara ukupnom iznosu u košarici.[168]

Četvrtim testnim slučajem nastoji se provjeriti da je moguće i ukloniti proizvod iz košarice (Slika 53.). Prvi korakom klikne se na potvrdni okvir proizvoda *3rd Album* iz razloga što je potrebno označiti proizvod koji se želi ukloniti te drugim korakom se izvršava klik na *Update shopping cart* gumb koji ažurira sadržaj košarice. Na kraju se očekuje da će proizvod biti skriven iz košarice, odnosno uspješno uklonjen.

```
test('TEST 4 Verify that you can remove the item from the cart',
async ({ page }) => {
  await page.pause();
  await shopping_cart.removeCheckbox.click();
  await page.pause();
  await shopping_cart.updateShoppingCartBtn.click();
  await page.pause();
  await expect(shopping_cart.firstItemInCart).toBeHidden();
});
```

Slika 53. Provjera uklanjanja proizvoda iz košarice.[169]

Peti testni slučaj provjerava funkcionalnost gumba Continue shopping (Nastavi kupovinu) sa stranice košarice (Slika 54.). Test će kliknuti na Continue shopping gumb i očekivati da je

[168] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/shopping_cart.spec.ts#L28-L41
[169] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/shopping_cart.spec.ts#L43-L51

promijenjena URL adresa odnosno da se više ne nalazi na stranici košarice već na početnoj stranici Demo Web Shop-a.

```
test('TEST 5 Verify that the Continue shopping button is working correctly',
async ({ page }) => {

  await shopping_cart.continueShoppingBtn.click();
  await expect(page).toHaveURL(
    '/'
  );
});
```

Slika 54. Provjera da Continue shopping gumb preusmjerava korisnika na početnu stranicu aplikacije.[170]

Šesti testni slučaj provjerava obvezu prihvaćanja Uvjeta korištenja prije nego što se može nastaviti s kupovinom tj. postupkom plaćanja proizvoda (Slika 55.). Test će prvo provjeriti da potvrdni okvir *Terms of service* (Uvjeti korištenja) nije označen, zatim će pokušati nastaviti kupovinu klikom na gumb *Checkout* (za nastavak plaćanja) bez klikanja na potvrdni okvir. Očekivano ponašanje je da će se na stranici prikazati upozorenje koje korisnika obavještava da mora prihvatiti uvjete korištenja prije nastavka kupovine.

```
test('TEST 6 Verify that the Terms of service checkbox is required',
async () => {

  await expect(shopping_cart.termsOfServiceCheckbox).not.toBeChecked();
  await shopping_cart.checkoutBtn.click();
  await expect(shopping_cart.termsOfServiceWarningBox).toBeVisible();
});
```

Slika 55. Provjera obveze prihvaćanja Uvjeta korištenja prije nastavka kupovine.[171]

Sedmi testni slučaj obuhvaća provjeru testiranja cjelokupnog procesa dovršavanja kupovine na Demo Web Shop aplikacije sa stranice košarice (Slika 56.). Važno je podsjetiti da je na aplikaciji moguće testirati kupovinu proizvoda i da se transakcije neće izvršavati te novac neće biti naplaćen s obzirom da je aplikacija namijenjena za testno okruženje. Za provjeru uspješnosti dovršavanja kupovine, izvršeni su sljedeći koraci: klik na potvrdni okvir *Terms of*

[170] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/shopping_cart.spec.ts#L53-L60
[171] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/shopping_cart.spec.ts#L62-L68

service, klik na gumb *Checkout* i zatim klik na *Checkout as Guest* gumb. Potom se upisuju podaci o adresi za naplatu: ime (*Marija*), prezime (*QA*), e-mail adresa (*mpw@gmail.com*).

Nakon toga se iz padajućeg izbornika odabire država (*Croatia*), te se upisuju ostali podaci: grad (*Osijek*), adresa (*Ul. Lorenza Jagera*), poštanski broj (*31000*) i broj mobilnog uređaja (*987283947*). Nakon što su ispunjeni svi unosi za popunjavanje adrese za naplatu, kliknut će se na *Continue* gumb za nastavak. U ovom trenutku opet se koristi naredba da se pauzira izvršavanje sljedećeg koraka na jednu sekundu kako bi se stranica učitala prije izvršavanja sljedećih koraka. S obzirom da sljedeći korak odabiranja *Načina plaćanja* ima već odabranu zadanu vrijednost *Cash On Delivery* (Plaćanje pouzećem), ostaviti ćemo odabranu vrijednost te samo kliknuti opet na *Continue* gumb i ponovno pričekati sekundu prije sljedećih koraka. U aplikaciji slijedi upisivanje podataka o plaćanju, a s obzirom da je prethodno odabrano plaćanje pouzećem, ovaj korak podrazumijeva isključivo klik na gumb *Continue*. Kad su upisani svi podaci, za kraj je potrebno kliknuti na gumb *Confirm* koji potvrđuje kupovinu i očekivano ponašanje je da će potom biti prikazana poruka o uspješnoj kupovini.

```
test('TEST 7 Verify the ability to complete a purchase successfully',
async ({ page }) => {

  await shopping_cart.termsOfServiceCheckbox.click();
  await shopping_cart.checkoutBtn.click();
  await shopping_cart.checkoutAsGuestBtn.click();

  await shopping_cart.firstNameInput.fill('Marija');
  await shopping_cart.lastNameInput.fill('QA');
  await shopping_cart.emailInput.fill('mpw@gmail.com');
  await shopping_cart.countryInput.selectOption('Croatia');
  await shopping_cart.cityInput.fill('Osijek');
  await shopping_cart.addresInput.fill('Ul. Lorenza Jagera');
  await shopping_cart.postalCodeInput.fill('31000');
  await shopping_cart.phoneNumberInput.fill('987283947');
  await shopping_cart.continueCheckoutBtn.click();
  await page.waitForTimeout(1000);
  await shopping_cart.continueCheckoutBtn.click();
  await page.waitForTimeout(1000);
  await shopping_cart.continueCheckoutBtn.click();
  await shopping_cart.confirmBtn.click();
  await expect(shopping_cart.successMessage).toBeVisible();

});
```

Slika 56. Provjera uspješnog dovršavanja kupovine na Demo Web Shop aplikaciji.[172]

[172] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/shopping_cart.spec.ts#L70-L93

U konačnici, posljednji test testiranja košarice provjerava da li se stranica ispravno prikazuje ukoliko je košarica prazna (Slika 57.). Test će kliknuti na potvrdni okvir za uklanjanje proizvoda i na gumb za ažuriranje sadržaja košarice. Nakon što se ti koraci izvrše, tvrdnjom očekujemo da će na stranici biti prikazana poruka da je košarica prazna.

```
test('TEST 8 Verify that the empty state of the shopping cart is displayed correctly',
async ({}) => {

  await shopping_cart.removeCheckbox.click();
  await shopping_cart.updateShoppingCartBtn.click();
  await expect(shopping_cart.informativeMessage).toBeVisible();
});
```

Slika 57. Provjera da se prazno stanje košarice ispravno prikazuje.[173]

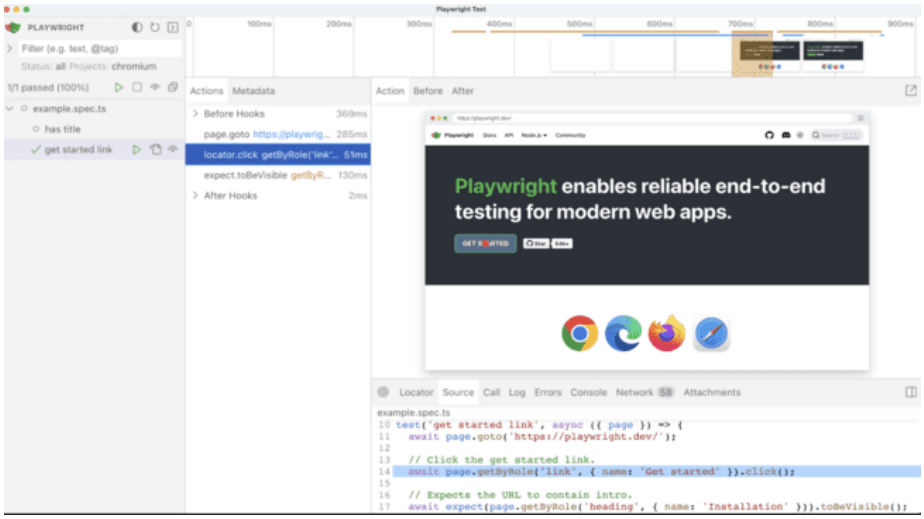
4.3.5. Pokretanje testova i prikaz izvještaja

Playwright omogućava pokretanje jednog testa, skup testova ili sve testove. Testovi se mogu pokrenuti na jednom ili više preglednika te se prema zadanim postavkama izvršavaju paralelno i kroz tzv. *headless* način rada, što znači da se tijekom izvođenja testova neće otvoriti prozor preglednika. Osim toga, testove je moguće pokrenuti i kroz tzv. *headed* način rada koji otvara preglednik tijekom izvršavanja testova omogućujući testeru da prati što se događa.[174]

Testovi se pokreću kroz terminal pomoću naredbe ***npx playwright test***.

Ukoliko želi, tester može izvoditi testove s *UI* načinom rada prikazanim na Slici 58., gdje može jednostavno prolaziti kroz svaki korak testa i vizualno provjeriti što se događa s testom prije, tijekom i poslije svakog koraka. Ovaj način rada korisničkog sučelja također dolazi s drugim značajkama poput načina rada za gledanje (eng. watch mode), birač lokatora (eng. locator picker) i sl.[175] Za pokretanje UI načina rada, potrebno je koristiti naredbu *npx playwright test --ui*.

[173] https://github.com/mzilic98/diplomski-playwright/blob/a9b994a7dbb6cdecf9de89d3476d728559447cc2/tests/shopping_cart.spec.ts#L95-L101
[174] Playwright; Running and debugging tests. URL: https://playwright.dev/docs/running-tests (2024-08-09)
[175] Isto.



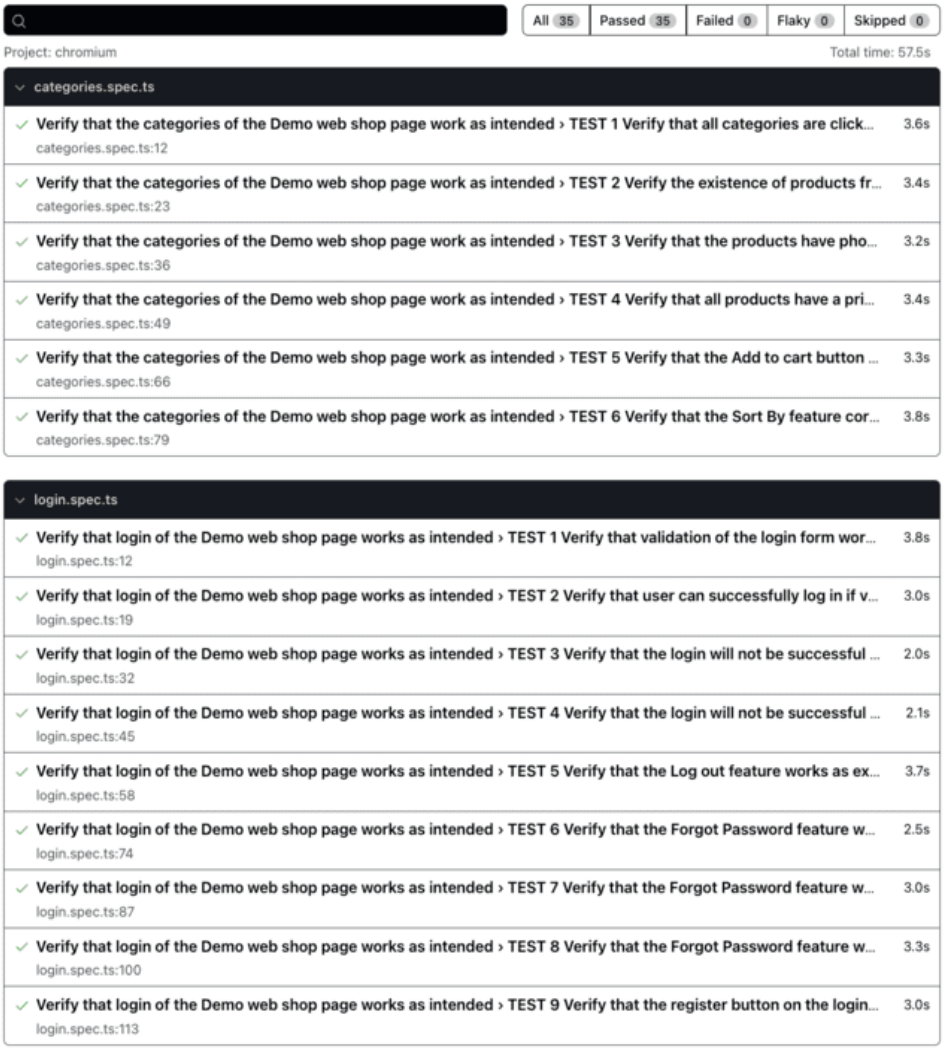
Slika 58. Pokretanje testova kroz UI način rada.[176]

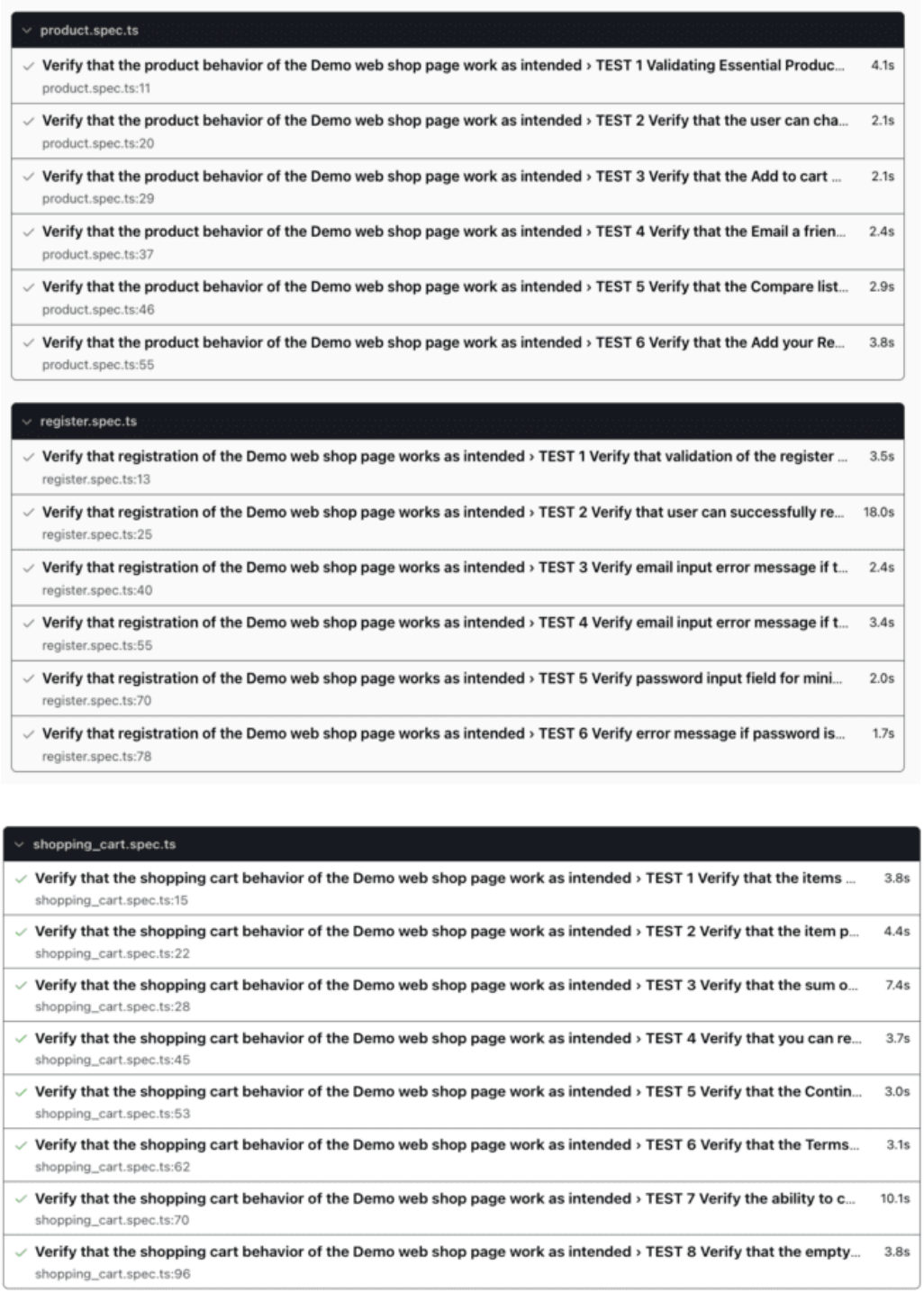
Pokretanje testova u spomenutom *headed* načinu rada, omogućuje testeru da vizualno provjeri kako Playwright komunicira s mrežnim mjestom.[177] Za ovaj način rada i pokretanja testova koristi se naredba *npx playwright test --headed*.

Za pokretanje pojedinačnog testa potrebno je kod testnog slučaja koristiti metodu *test.only* da bi Playwright prepoznao test kojeg želimo provjeriti i preskočio ostale testove.

U konačnici, za prikaz izvještaja, koristi se naredba ***npx playwright show-report***. HTML Reporter ima mogućnost prikaza potpunog izvještaja o testovima, omogućujući i filtriranje prema uspješnim, neuspješnim, preskočenim i nestabilnim testovima.[178] U izvještaju prikazanim na Slici 59. se može vidjeti da je svih 35 testnih slučajeva uspješno prošlo. Ovi rezultati ukazuju na to da je Demo Web Shop aplikacija zadovoljila kriterije prihvatljivosti odnosno da zadovoljava sve uvjete postavljene u testnim slučajevima. Testni slučajevi pokrili su ključne funkcionalnosti aplikacije i time ukazali, da iako je u pitanju testno okruženje, te funkcionalnosti ispravno rade prema predviđenim zahtjevima. Međutim, za dugoročno osiguranje kvalitete ove aplikacije, spomenute testove potrebno je redovito ažurirati te kreirati nove testne slučajeve kako bi obuhvatili nove scenarije korištenja aplikacije.

[176] Isto.
[177] Playwright. Running and debugging tests. URL: https://playwright.dev/docs/running-tests (2024-08-09)
[178] Isto.


All 35
Passed 35
Failed 0
Flaky 0
Skipped 0
Project: chromium
Total time: 57.5s
categories.spec.ts
Verify that the categories of the Demo web shop page work as intended › TEST 1 Verify that all categories are click... 3.6s
categories.spec.ts:12
Verify that the categories of the Demo web shop page work as intended › TEST 2 Verify the existence of products fr... 3.4s
categories.spec.ts:23
Verify that the categories of the Demo web shop page work as intended › TEST 3 Verify that the products have pho... 3.2s
categories.spec.ts:36
Verify that the categories of the Demo web shop page work as intended › TEST 4 Verify that all products have a pri... 3.4s
categories.spec.ts:49
Verify that the categories of the Demo web shop page work as intended › TEST 5 Verify that the Add to cart button ... 3.3s
categories.spec.ts:66
Verify that the categories of the Demo web shop page work as intended › TEST 6 Verify that the Sort By feature cor... 3.8s
categories.spec.ts:79
login.spec.ts
Verify that login of the Demo web shop page works as intended › TEST 1 Verify that validation of the login form wor... 3.8s
login.spec.ts:12
Verify that login of the Demo web shop page works as intended › TEST 2 Verify that user can successfully log in if v... 3.0s
login.spec.ts:19
Verify that login of the Demo web shop page works as intended › TEST 3 Verify that the login will not be successful ... 2.0s
login.spec.ts:32
Verify that login of the Demo web shop page works as intended › TEST 4 Verify that the login will not be successful ... 2.1s
login.spec.ts:45
Verify that login of the Demo web shop page works as intended › TEST 5 Verify that the Log out feature works as ex... 3.7s
login.spec.ts:58
Verify that login of the Demo web shop page works as intended › TEST 6 Verify that the Forgot Password feature w... 2.5s
login.spec.ts:74
Verify that login of the Demo web shop page works as intended › TEST 7 Verify that the Forgot Password feature w... 3.0s
login.spec.ts:87
Verify that login of the Demo web shop page works as intended › TEST 8 Verify that the Forgot Password feature w... 3.3s
login.spec.ts:100
Verify that login of the Demo web shop page works as intended › TEST 9 Verify that the register button on the login... 3.0s
login.spec.ts:113



Slika 59. Prikaz izvještaja svih testnih slučaja.

5. Rasprava

Za prikaz i primjenu automatiziranog testiranja u Playwright radnom okviru na mrežnom mjestu Demo Web Shop aplikacije, izabrano je 35 testova koji obuhvaćaju ključne

funkcionalnosti online trgovine, poput registracije i prijave korisnika, pregleda proizvoda, dodavanje proizvoda u košaricu i kupovine proizvoda. Ovi testovi provjeravaju da osnovne funkcionalnosti rade kako je očekivano. Unatoč visokoj pokrivenosti osnovnih funkcionalnosti, važno je naglasiti da aplikacija sadrži još dodatnih funkcionalnosti koje bi se mogle testirali, no i da obuhvaća neke pogreške. Primjerice, iz perspektive i korisnika i testera, funkcionalnost liste želja (eng. wish-list) nije intuitivna te je tijekom testiranja utrošeno značajno vrijeme u pokušaju pronalaženja načina za dodavanje proizvoda na listu želja; neuspješno. Testni slučajevi pažljivo su odabrani radi prikaza opsega, ali i dubine koje zahtjeva testiranje. Poneki testni slučajevi mogu se činiti slični, no testiraju se različiti scenariji korištenja aplikacije kako bi se omogućilo sveobuhvatnije testiranje. Prije kreiranja automatskih testova bilo je potrebno složiti test plan koji se izrađivao tijekom faze planiranja testiranja. S obzirom da je baza (osnova) za testiranje i sva dokumentacija o ponašanju aplikacije nedostupna, tijekom faze analize testiranja razmatralo se što se treba testirati iz perspektive krajnjeg korisnika, te se ručno testirala aplikacija kako bi se otkrilo ponašanje sustava. Dakle, prvenstveno je ručno istestirana aplikacija na razini testiranja sustava koja, kao što je spomenuto, omogućava testiranje sustava iz perspektive krajnjeg korisnika. Tijekom faze dizajna testova odredili su se konkretni testni slučajevi koji će se koristiti u ovom radu. Tijekom testiranja, korištena je metoda crne kutije, odnosno fokus je bio isključivo na vanjskom ponašanju sustava. Implementacija i izvođenje testova događali su se uzastopno jedan za drugim, odnosno u trenutku kad bi se jedan testni slučaj kreirao potom bi se odmah i izvršavao s ciljem provjere je li testni slučaj ispravno implementiran. Najviše izazova predstavljalo je dohvaćanje i korištenje ispravnih lokatora za mrežne elemente tijekom faze implementacije testova. U određenom broju slučajeva, mrežna aplikacija ne sadrži jedinstvene identifikatore na elementima pa je bilo izazovno odrediti ispravan lokator. Osim toga, tijekom faze izvođenja testova, testni slučajevi su često "padali" dok nije postavljen pravilan lokator i pravilna sintaksa testnog slučaja. Tijekom pisanja testova (faza implementacije testova), za svaki pojedinačan testni slučaj korišten je spomenuti *headed* način rada. Ovaj način rada omogućio je pokretanje preglednika s korisničkim sučeljem te Playwright Inspector alat koji je služio kao pomagalo u izboru lokatora, interaktivno praćenje i provjeru svih testnih koraka u testnom slučaju. Prateći Playwright dokumentaciju o postavljanju osnovnih radnji i tvrdnji, bilo je jasno odrediti koje radnje se trebaju dogoditi i kako ih implementirati u kod te su tvrdnje također bile jasno definirane. Nakon što je ustanovljeno da su svi testni slučajevi pravilno implementirani, svih 35 testova pokrenuto je u *headless* načinu rada. Tijekom faze završetka testiranja analizirali su se rezulati testiranja te je utvrđeno da aplikacija zadovoljava uvjete testiranja. Tijekom

cjelokupnog procesa od kreiranja plana do završetka testiranja, upravljanje i kontrola testiranja bila je glavna aktivnost prema kojoj se pratio napredak testiranja, te se procjenjivalo jesu li ispunjeni kriteriji za završetak testiranja. Napisani testni slučajevi dizajnirani su s namjerom kako bi provjerili funkcionalnost aplikacije prema njenom trenutnom stanju, osiguravajući da prolaze ukoliko aplikacija radi ispravno prema očekivanom ponašanju sustava, no u budućnosti, ukoliko dođe do promjene u aplikaciji, postoji mogućnost da neki od ovih testova padnu i otkriju pogrešku u aplikaciji, što ispunjava svrhu regresijskog testiranja u kontekstu automatizacije sustava.

6. Zaključak

Automatsko testiranje u Playwright radnom okviru omogućava preciznu automatizaciju testnih slučajeva koji se konstantno mogu ponovno reproducirati. Playwright kao radni okvir pokazao se vrlo jednostavnim za korištenje, s obzirom da nudi veliki spektar funkcionalnosti koje olakšavaju testiranje (poput podrške za otklanjanje pogrešaka, hvatače lokatora i sl.). Također, paralelno testiranje više mrežnih preglednika uveliko skraćuje vrijeme koje bi bilo uloženo da se ručno testirao svaki testni slučaj na svim preglednicima. Učenje o Playwright-u zahtijevalo je više vremena od pisanja testnih slučajeva, jer iako postoji dostupna dokumentacija, osobe koje se prvi put susreću s automatskim testiranjem bi se vrlo vjerojatno mogle suočiti s izazovima u snalaženju i pronalaženju informacija. Demo Web Shop mrežna aplikacija pokazala se kao testno okruženje sa čak vrlo kompleksnim i stvarnim značajkama online trgovine i mnoštvo mogućih scenarija za testiranje. Aplikacija iako ne izgleda kao moderna aplikacija, obuhvaća sve funkcionalnosti online trgovine koje mogu biti odlična podloga za učenje i edukaciju kako automatskog, tako i ručnog testiranja. Automatsko testiranje ima prednosti nad ručnim testiranjem, no automatsko testiranje ne može postojati bez ručnog testiranja. Primjerice, za svaki testni slučaj kojeg želimo automatizirati potrebno je ručno istestirati testni slučaj kako bi uvidjeli ponašanje aplikacije i mogli odrediti osnovne radnje koje se trebaju dogoditi i uspostaviti tvrdnje očekivanog ponašanja. Automatski testovi, iako jednom napisani, mogu se ponovno reproducirati nebrojeno puta, no također ih je potrebno održavati i mijenjati kako se mijenjaju zahtjevi specifikacija. Uvođenje automatizacije postaje neophodno u razvoju programskih rješenja te korištenje Playwright radne okoline za automatizirano testiranje može pridonijeti učinkovitosti, pouzdanosti i dosljednosti pri razvoju, testiranju i održavanju mrežnih mjesta.

7. Literatura

Akhtar, Hamid. What is White Box Testing? (Example, Types, & Techniques), 2023. URL: https://www.browserstack.com/guide/white-box-testing

Bose, Shreya. How to write Test Cases in Software Testing? (with Format & Example), 2024. https://www.browserstack.com/guide/how-to-write-test-cases

BrowserStack; Benefits of Automation Testing. 2023. URL: https://www.browserstack.com/guide/benefits-of-automation-testing

BrowserStack; Playwright Automation Framework: tutorial. 2023. URL: https://www.browserstack.com/guide/playwright-tutorial

Das, Sourojit. What is Black Box Testing: Types, Tools & Examples, 2023. URL: https://www.browserstack.com/guide/black-box-testing

Demo Web Shop. URL: https://demowebshop.tricentis.com/

Drumond, Claire. What is scrum and how to get started: Scrum Guide - What it is, how it works, and how to start. URL: https://atlassian.com/agile/scrum

Dwivedi, Sonal. What is a Test Log?, 2023. URL: https://www.browserstack.com/guide/what-is-test-log

Hamilton ,Thomas. What is Unit Testing?, 2024. URL: https://www.guru99.com/unit-testing-guide.html

Hamilton, Thomas. Automation Testing. URL: https://www.guru99.com/automation-testing.html

Hamilton, Thomas. Levels of Testing in Software Testing, 2024. URL: https://www.guru99.com/levels-of-testing.html

Hamilton, Thomas. Manual Testing Tutorial, 2024. URL: https://www.guru99.com/manual-testing.html

Hamilton, Thomas. Test Condition vs Test Scenario in Software Testing, 2024. URL: https://www.guru99.com/test-scenario-vs-test-condition.html

Hamilton, Thomas. What is Dynamic Testing? Types, Techniques & Example. URL: https://www.guru99.com/dynamic-testing.html

Hamilton, Thomas. What is Regression Testing? URL: https://www.guru99.com/regression-testing.html

Hamilton, Thomas. What is User Acceptance Testing (UAT)? Examples, 2024. URL: https://www.guru99.com/user-acceptance-testing.html

IEEE; Software Requirements Specifications: Building a Blueprint for Success: Navigating the Complexities of Software Requirements. URL: https://www.computer.org/resources/software-requirements-specifications

Kitakabee. Automated Regression Testing: A Detailed Guide, 2023. URL: https://www.browserstack.com/guide/automated-regression-testing

Playwright. Playwright Test. URL: https://playwright.dev/docs/api/class-test

Playwright. URL: https://playwright.dev/

Playwright; Locators. URL: https://playwright.dev/docs/locators

Playwright; Page object models. URL: https://playwright.dev/docs/pom

Playwright; Page. URL: https://playwright.dev/docs/api/class-page

Playwright; Running and debugging tests. URL: https://playwright.dev/docs/running-tests

Playwright; Writing tests. URL:https://playwright.dev/docs/writing-tests

Red Hat. What is CI/CD?, 2023. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd

Rehkopf, Max. User stories with examples and a template. URL: https://www.atlassian.com/agile/project-management/user-stories

Shadow Software News. How to Transition from Waterfall to Agile: A Step-by-Step Guide, 2024. URL: https://shadowsoftware.com/how-to-transition-from-waterfall-to-agile-a-step-by-step-guide/

Quality Assurance Handbook; Test Documentation, 2023. URL: https://infinum.com/handbook/qa/basics/test-documentation

Quality Assurance Handbook; Writing bug reports, 2023. URL: https://infinum.com/handbook/qa/basics/writing-bug-reports

Quality Assurance Handbook; Writing test cases, 2022. URL: https://infinum.com/handbook/qa/basics/writing-test-cases

Software Testing Help; How To Write A Good Bug Report? Tips and Tricks. URL: https://www.softwaretestinghelp.com/how-to-write-good-bug-report/

Spillner, Andreas; Linz, Tito. Software Testing Foundations; A Study for the Certified Tester Exam - Foundation Level & ISTQB® Compliant. 5th, revised and updated Edition. Heidelberg, Germany: dpunkt.verlag, 2021.

TypeScript. URL: https://www.typescriptlang.org/

TypeScript. TypeScript 1.7. URL: https://www.typescriptlang.org/docs/handbook/release-notes/typescript-1-7.html

Ulili, Stanley. Playwright vs Puppeteer vs Cypress vs Selenium (E2E testing), 2024. URL: https://betterstack.com/community/comparisons/playwright-cypress-puppeteer-selenium-comparison/#4-debugging-tools

Verma, Antra. End To End Testing: Tools, Types, & Best Practices, 2024. URL: https://www.browserstack.com/guide/end-to-end-testing

Visual Studio Code. URL: https://code.visualstudio.com/docs

Walker, Alyssa. How to Write A Bug Report with Examples, 2024. URL: https://www.guru99.com/how-to-write-a-bug-report.html