Automatizirano testiranje web sustava

Hodić, Iva

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

AUTOMATIZIRANO TESTIRANJE WEB SUSTAVA

Diplomski rad

Iva Hodić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Iva Hodić
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1286R, 07.10.2022.
JMBAG:	0165081688
Mentor:	izv. prof. dr. sc. Josip Balen
Sumentor:	
Sumentor iz tvrtke:	Ena Filipović
Predsjednik Povjerenstva:	doc. dr. sc. Krešimir Romić
Član Povjerenstva 1:	izv. prof. dr. sc. Josip Balen
Član Povjerenstva 2:	Antonio Antunović, univ. mag. ing. comp.
Naslov diplomskog rada:	Automatizirano testiranje web sustava
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U diplomskom radu je potrebno razviti i opisati automatizirano testiranje za web sustave. Nadalje, na primjeru web sustava potrebno je usporediti i analizirati alate za automatizaciju (kao što su Cypress i Playwright). Za potrebe testiranja potrebno je razviti funkcionalnosti za testiranje, osmisliti testove/scenarije prema odabranim funkcionalnostima te provesti i dokumentirati kompletan proces testiranja. Tema rezervirana za: Iva Hodić Sumentor iz tvrtke: Ena Filipović (Span d.d.)

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	23.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	30.09.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	30.09.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 30.09.2024.

Ime i prezime Pristupnika:	Iva Hodić
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1286R, 07.10.2022.
Turnitin podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: Automatizirano testiranje web sustava**

izrađen pod vodstvom mentora izv. prof. dr. sc. Josip Balen

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

U današnjem digitalnom dobu, web sustavi su postali neizostavan dio modernog poslovanja i svakodnevnog života zbog sve veće prisutnosti digitalnih usluga i aplikacija. Kako bi se osigurala njihova pouzdanost, sigurnost i funkcionalnost, testiranje web aplikacija je postalo nezaobilazan korak u procesu razvoja softvera. Kvalitetno testiranje omogućava rano otkrivanje i ispravljanje grešaka, čime se povećava ukupna kvaliteta proizvoda i korisničko iskustvo.

Unatoč napretku u tehnologiji i razvojnim praksama, velike pogreške se i dalje događaju, uzrokujući značajne materijalne gubitke i narušavajući uspješnost poslovanja različitih tvrtki. Neke od takvih pogrešaka potkrale su se i najvećim tvrtkama, kao što su Apple i Nike. Apple je 2019. godine doživio ozbiljan problem nakon izdavanja ažuriranja za operacijski sustav. Nekoliko sati nakon puštanja novog ažuriranja, korisnici su masovno prijavljivali probleme s obavljanjem telefonskih poziva, što je prisililo Apple da hitno izda novo ažuriranje kako bi ispravio grešku. Ovaj incident je izazvao nezadovoljstvo korisnika i narušio ugled tvrtke, pokazujući koliko je kritično osigurati kvalitetno testiranje prije puštanja softvera u proizvodnju. Slično tome, Nike je iskusio ozbiljan problem tijekom prelaska na novu valutu. Prilikom konverzije cijena iz kuna u eure, dogodila se greška gdje su cijene proizvoda bile ispravno preračunate, ali je valuta ostala prikazana u kunama. Ova greška izazvala je zbunjenost među kupcima i financijske gubitke za tvrtku. Takvi incidenti naglašavaju potrebu za detaljnim i sveobuhvatnim testiranjem kako bi se izbjegle skupe i neugodne pogreške.

Zbog takvih velikih izazova, testiranje je jedan od najbitnijih procesa kod razvoja softvera. Može se podijeliti na ručno i automatizirano. Ručno testiranje je vrlo zahtjevan posao jer se testovi trebaju izvršavati nakon svake nadogradnje i izmjene koda. Na taj način se dugoročno troše vrijeme i resursi. Ponekad se testiranja provode rijetko i testovi nisu kompliciranji pa je u tim slučajevima ručno testiranje jednostavnije i isplativije.

S druge strane, kod velikih i zahtjevnih softvera, poželjno je uvesti automatizaciju testova. Automatizirano testiranje smanjuje vrijeme potrebno za testiranje u usporedbi s ručnim metodama i povećava učinkovitost testiranja. Ipak, važno je pažljivo odabrati alat za automatizaciju, jer ne odgovara svaki alat svim aplikacijama.

Cilj ovog rada je istražiti i usporediti alate za automatizirano testiranje, s naglaskom na Cypress i Playwright. Kroz detaljnu analizu i praktične primjere, rad će prikazati kako automatizirano testiranje može unaprijediti kvalitetu i pouzdanost web sustava.

2. UVOD U TESTIRANJE

U ovom poglavlju razmatra se važnost testiranja softvera kao neizostavnog dijela procesa razvoja softverskih proizvoda. Detaljno će se opisati što je testiranje softvera, uključujući njegove osnovne faze, metode i tipove testiranja. Poglavlje će obuhvatiti i razliku između ručnog i automatiziranog testiranja te prednosti i izazove svake od tih metoda.

2.1. Definicija softvera

Softver (*engl. software*) je skup uputa, podataka i računalnih programa koji se koriste za izvršavanje specifičnih zadataka koji se isporučuju korisniku. Svi softveri imaju svojstva koja se definiraju prije samog razvoja. Kvaliteta i funkcionalnost softverskog proizvoda uvelike ovise o temeljitosti i preciznosti tijekom cijelog razvojnog procesa [1]. Prema [2], greške (*engl. bug*) ili nedostatci gotovog proizvoda često su rezultat propusta tijekom same proizvodnje, ali se mogu pojaviti i u ranim fazama definiranja i dizajniranja proizvoda. Svako odstupanje od specificiranog ponašanja sustava se smatra greškom. Većina grešaka nastaje zbog ljudskih pogrešaka i pogrešaka tijekom izrade izvornog koda i njegovog dizajna. Nekoliko nedostataka također može proizaći i iz prevoditelja koji generiraju pogrešan kod. Zbog toga je izrazito bitno u samom početku biti oprezan jer nedostatak pažnje može dovesti do problema tek kasnije kada je popravak skuplji i kompliciraniji.

2.2. Testiranje softvera

U svijetu razvoja softvera, kvaliteta je metrika koja opisuje u kojoj mjeri neki softver zadovoljava specificirane zahtjeve, obavlja svoje funkcije i udovoljava željama i potrebama korisnika. Najvažnije točke koje procjenjujemo prilikom izrade softvera su validacija i verifikacija. Kroz validaciju se postavlja pitanje koliko dobro softver ispunjava stvarne potrebe korisnika. S druge strane, verifikacija postavlja pitanje gradi li se proizvod na ispravan način, u skladu sa standardima. Odgovori na ta pitanja dolaze od toga koliko principa zadovoljava softver. Najvažniji princip je funkcionalnost koja je temelj svakog softvera. Ako softver ne obavlja ono za što je namijenjen, onda svi ostali problemi postaju manje bitni. Upotrebljivost je princip koji se fokusira na korisničko iskustvo prilikom korištenja aplikacija. Sljedeći bitan princip je pouzdanost koja osigurava da korisnici mogu pouzdano koristiti softver bez prekida u radu ili neočekivanih padova. Pouzdanost uključuje i održivost koja se brine da je softver lako održavati i nadograđivati. Svi ovi principi uz kompatibilnost, sukladnost i sigurnost trebaju biti zadovoljeni testiranjem softvera kako bi se na kraju isporučio softver visokih performansi.

2.2.1. Faze testiranja softvera

Svaki razvoj softvera počinje fazom planiranja i analiza potreba u kojoj se definira što se razvija i na koji način. Nakon planiranja započinje faza dizajna sučelja te onda implementacija u kojoj programeri razvijaju softver po zadanoj specifikaciji. Dok programeri rade na implementaciji, testeri pripremaju testne scenarije kako bi se pripremili za fazu testiranja. Glavne aktivnosti obuhvaćaju planiranje, dizajn testova, izvršavanje testova, izvještavanje i rješavanje problema.

Testiranje započinje već u fazi zahtjeva i poslovne analize. U ovoj fazi, tim testera analizira funkcionalne i nefunkcionalne zahtjeve, korisničke priče (*engl. user stories*) i dizajn. Komunikacija s ostatkom tima ključna je kako bi se osigurali jasni, razumljivi i testabilni zahtjevi. Kroz ovu analizu, tim testera identificira sve moguće probleme ili nedosljednosti u zahtjevima koji bi mogli otežati kasnije faze testiranja. Ova rana uključenost testera pomaže u postizanju stabilnih i jasnih zahtjeva, što je temelj za kvalitetno testiranje.

Nakon analize zahtjeva, slijedi faza planiranja testiranja. Ova faza ima za cilj definirati opseg (*engl. scope*) testiranja, vremenski raspored i pristup testiranju. U ovom koraku, testeri određuju koje vrste testiranja će se provoditi, koje metode i alati će se koristiti, te definiraju potrebne resurse i okolinu za testiranje.

Nakon planiranja, testeri kreiraju testne slučajeve (*engl. test cases*) koji će se koristiti za provjeru funkcionalnosti softvera. Testni slučajevi detaljno opisuju korake koje je potrebno poduzeti i očekivane rezultate za svaki scenarij. Dobro osmišljeni testni slučajevi omogućuju sistematsko i sveobuhvatno testiranje svih aspekata softvera. U ovoj fazi, također se pripremaju testni podaci i određuju kriteriji za uspjeh i neuspjeh svakog testa.

Kada su testni slučajevi spremni, testeri započinju s njihovim izvršavanjem. Tijekom izvršavanja testova, pažljivo se bilježe rezultati i sva odstupanja od očekivanih rezultata se smatraju greškama. Izvršavanje testova može uključivati ručno testiranje, automatizirano testiranje ili kombinaciju, ovisno o zahtjevima projekta i dostupnim resursima. Svaka pronađena greška prijavljuje se razvojnim inženjerima koji su odgovorni za njeno ispravljanje. Nakon što razvojni tim popravi grešku, testeri ponovno izvršavaju relevantne testove kako bi se uvjerili da je problem riješen. Ovaj ciklus prijave grešaka, ispravljanja i ponovnog testiranja ponavlja se sve dok svi testovi ne budu uspješno završeni i sve greške ispravljene.

2.3. Metode testiranja softvera

Postoje dvije metode testiranja softvera prema automatizaciji. To su ručno i automatizirano testiranje. Iako se gotovo svaki testni slučaj može automatizirati, pitanje isplativosti često određuje izbor metode. Obje metode testiranja imaju svoje prednosti i nedostatke, a optimalan pristup obično uključuje kombinaciju oba. Jedna metoda testiranja nije zamjena za drugu, već se nadopunjuju kako bi se postigla sveobuhvatna kvaliteta softvera [3].

2.3.1. Ručno testiranje

Prema [3] i [4], u ručnom testiranju, tester sam izvodi testne slučajeve bez upotrebe alata za automatizaciju, ponašajući se kao krajnji korisnik. Iskustvo i znanje testera ključni su za prepoznavanje neslaganja i odstupanja od očekivanih funkcionalnosti aplikacije te bilježenje grešaka. Testeri prilagođavaju svoje pristupe i improviziraju prema potrebama aplikacije. Ručno testiranje ne zahtjeva puno vremena za pripremu, ali može biti vremenski i resursno zahtjevno za složenije aplikacije. Također omogućava testeru uočavanje grešaka iz perspektive krajnjeg korisnika.

2.3.2. Automatizirano testiranje

Prema [3] i [5], u automatiziranom testiranju tester piše skripte koje softver za automatizaciju pokreće i testira aplikaciju. Automatizirani testovi se izvode mnogo brže nego ručni, što je korisno kod velikih sustava gdje ručno testiranje može biti jako dugotrajno. Također, automatizirani testovi omogućuju veću pokrivenost testnih slučajeva smanjujući rizik od neotkrivenih grešaka. Velika prednost automatiziranog testiranja je to da se mogu ponavljati koliko god puta se to traži od testera, što je izrazito bitno kod regresijskog testiranja kada tester mora osigurati da nove promjene nisu uvele nove greške u sustav. Ipak, automatizirano testiranje ima i svoje mane, poput visokih početnih troškova i redovitog održavanja kako bi testovi ostali relevantni s promjenama u softveru.

2.4. Tipovi testiranja softvera

Tipovi testiranja imaju različite ciljeve i namijenjeni su testiranju određenih specifikacija aplikacije. Prema [6], [7] i [8] postoje dva osnovna tipa testiranja, a to su funkcionalno i nefunkcionalno testiranje.

2.4.1. Funkcionalno testiranje

Funkcije koje bi sustav trebao zadovoljavati su najčešće opisane u specifikacijama zahtjeva, slučajevima korištenja ili u funkcionalnoj specifikaciji. Funkcije sustava su ono što

sustav treba raditi, a funkcionalno testiranje je testiranje tih funkcija. Funkcionalno testiranje je tip testiranja u kojem se sustav testira prema funkcionalnim zahtjevima i specifikacijama te odgovara na pitanja „Što ovaj sustav treba raditi?“ i „Funkcionira li ovaj sustav na način kako je zamišljeno?“ [6], [7]. Postoji nekoliko podtipova funkcionalnog testiranja, a ovo su jedni od najčešćih i najvažnijih:

- testiranje komponente (*engl. Component testing*)
- testiranje dima (*engl. Smoke testing*)
- testiranje sustava (*engl. System testing*)
- regresijsko testiranje (*engl. Regression testing*)

Testiranje komponente je testiranje jednog manjeg dijela, odnosno komponente koja se može testirati neovisno o drugim komponentama. Testiranje sustava je cjelokupno testiranje svih komponenti. S obzirom da tada već postoji sučelje koje će vidjeti krajnji korisnik, testeri se stavljaju u njegovu ulogu i mogu testirati cjelokupni proces što se naziva testiranje od kraja do kraja (*engl. End To End testing*). Testiranje dima je vrlo kratko testiranje kojem je cilj provjera rade li glavne funkcionalnosti, najčešće se izvodi kako bi se provjerilo da je nova verzija aplikacije spremna za produkciju ili nakon puštanja na produkciju. Regresijsko testiranje se izvršava kada dođe do neke promjene u aplikaciji. Potrebno je provjeriti da je novi programski kod ispravno napisan i da nije prouzročio dodatne greške. Regresijsko testiranje podrazumijeva ponovno testiranje cijele aplikacije i svih njezinih funkcionalnosti [9].

2.4.2. Nefunkcionalno testiranje

Nefunkcionalno testiranje je usmjereno na provjeru nefunkcionalnih dijelova sustava, kao što su performanse, sigurnost i upotrebljivost. Ono testira kako sustav radi. Glavni podtipovi nefunkcionalnog testiranja su:

- testiranje performansi (*engl. Performance testing*)
- sigurnosno testiranje (*engl. Security testing*)
- testiranje upotrebljivosti (*engl. Usability testing*)
- testiranje pouzdanosti (*engl. Reliability testing*)

Testiranje performansi je podtip nefunkcionalnog testiranja koji se fokusira na procjenu performansi sustava ili aplikacije. Prema [10], neki od testova koje mora zadovoljiti su testiranje opterećenja, stres testiranje, testiranje skalabilnosti i testiranje odziva. Testiranje opterećenja je osiguravanje da sustav može podnijeti određen broj korisnika. Testiranje sustava

pod ekstremnim opterećenjima je stres testiranje. Testom skalabilnosti se smatra povećanje kapaciteta i performansi, a testom odziva se mjeri koliko brzo sistem odgovara na određene zahtjeve korisnika. Sigurnosno testiranje otkriva moguće prijetnje i rizike u aplikaciji i služi za sprječavanje zlonamjernih napada na sustav. Sigurnosni testovi se rade da bi se identificirale slabosti sustava koje mogu dovesti do raznih gubitaka informacija, prihoda ili ugleda tvrtke [11]. Testiranje upotrebljivosti procjenjuje koliko je aplikacija jednostavna za korištenje i koliko efikasno korisnik može postići svoj cilj. Ovo je testiranje izrazito važno da bi se osiguralo pozitivno korisničko iskustvo [12]. Testiranje pouzdanosti je proces testiranja softvera koji provjerava može li softver izvesti bez greške operacija u određenom okruženju tijekom određenog vremenskog razdoblja. Svrha testiranja pouzdanosti je osigurati da softverski proizvod nema grešaka i da je dovoljno pouzdan za svoju očekivanu svrhu [13].

2.5. Razine testiranja

Razine testiranja odnose se na različite načine testiranja koji se provode da se osigura da aplikacija radi ispravno i zadovoljava sve zahtjeve. Prema [9], [14] i [15], postoji 5 razina testiranja koje će biti opisane u nastavku, a to su: testiranje jedinice (*engl. Unit testing*), integracijsko testiranje (*engl. Integration testing*), testiranje sustava (*engl. System testing*) i testiranje prihvatljivosti (*engl. Acceptance testing*).

2.5.1. Testiranje jedinice

Testiranje jedinice, također poznato kao, testiranje modula, komponenti i testiranje programa, traži nedostatke i provjerava funkcionalnost softvera u prvoj fazi testiranja aplikacije. Testiranje jedinice obavlja se izolirano od ostatka aplikacije, ovisno o fazi životnog ciklusa razvoja i same aplikacije. Testiranje jedinice uključuje testiranje funkcionalnosti i posebnih nefunkcionalnih karakteristika kao što su ponašanje resursa, performanse ili testiranje otpornosti, kao i strukturno testiranje.

2.5.2. Integracijsko testiranje

Integracijsko testiranje slijedi nakon testiranja jedinica. Ovo je logičan slijed jer integracijsko testiranje uključuje povezivanje i testiranje više jedinica zajedno kako bi se osiguralo da one ispravno funkcioniraju u kombinaciji. Dok se u testiranju jedinica fokus stavlja na provjeru ispravnosti pojedinačnih funkcija, potprograma ili procedura u izolaciji, integracijsko testiranje usredotočuje se na provjeru sučelja i interakcija između tih jedinica. Cilj integracijskog testiranja je otkriti greške u interakcijama između različitih komponenti koje možda nisu bile vidljive tijekom testiranja pojedinačnih jedinica.

2.5.3. Sustavno testiranje

Sustavnim testiranjem se testira cijela aplikacija kao cjelina. Cilj ovog testiranja je ispuniti sve korisničke zahtjeve i postignuti željenu razinu kvalitete. Sustavno testiranje treba istražiti funkcionalne i nefunkcionalne zahtjeve sustava, kao i karakteristike kvalitete podataka. Sustavno testiranje funkcionalnih zahtjeva započinje korištenjem najprikladnijih tehnika temeljenih na specifikacijama kao što je crna kutija (*engl. Black box*) koja će biti opisana u nastavku. Testiranje se provodi u kontroliranom testnom okruženju koje je pažljivo osmišljeno da što više odgovara konačnom ciljanom okruženju. Ovakav pristup omogućava otkrivanje potencijalnih nedostataka koji bi se mogli pojaviti u stvarnom proizvodnom okruženju, čime se značajno smanjuje rizik od neotkrivenih grešaka.

2.5.4. Testiranje prihvatljivosti

Zadnja razina testiranja, koja dolazi nakon sustavnog testiranja je testiranje prihvatljivosti aplikacije. Cilj testiranja prihvatljivosti je uspostaviti povjerenje u sustav, dijelove sustava ili specifične nefunkcionalne karakteristike sustava kako bi se utvrdilo da je spreman za isporuku krajnjim korisnicima. Pronalaženje grešaka nije glavni cilj u ovom testiranju već je fokus na zadovoljavanju poslovnih potreba samog korisnika. Testiranje prihvatljivosti može procijeniti spremnost sustava za implementaciju i uporabu, iako to nije nužno posljednja razina testiranja. Na primjer, veliko integracijsko testiranje sustava može se provesti nakon testiranja prihvatljivosti sustava.

2.6. Strategije testiranja

Strategije testiranja su metode i pristupi korišteni za procjenu softvera, otkrivanje grešaka i osiguranje da softver zadovoljava specificirane zahtjeve. Prema [16], postoji nekoliko osnovnih vrsta tehnika testiranja, koje se mogu podijeliti na temelju pristupa i ciljeva testiranja, a to su: crna kutija (*engl. Black box*), bijela kutija (*engl. White box*) i testiranje na temelju iskustva (*engl. Experience – based testing*). U nastavku će biti opisani prema [17].

2.6.1. Crna kutija

Crna kutija, poznatija i kao testiranje temeljeno na specifikacijama, pristup je u kojem tester promatra aplikaciju kao "crnu kutiju". Zbog "crne kutije", cijeli je sustav skriven, pa se tester usredotočuje isključivo na funkcionalnosti, a ne na način na koji one rade. Tester nema nikakvo znanje o kodu ili unutarnjim procesima unutar aplikacije. Umjesto toga, fokusira se na testiranje aplikacije kao što bi to činio krajnji korisnik. U ovoj strategiji tester unosi određene

ulazne podatke u aplikaciju i provjerava odgovarajuće izlazne rezultate, uspoređujući ih s očekivanim ishodima.

2.6.2. Bijela kutija

Bijela kutija, odnosno testiranje strukture, analizira unutarnju strukturu, korištene podatkovne strukture, unutarnji dizajn, strukturu koda i rad same aplikacije. Za razliku od crne kutije, koje se fokusira samo na funkcionalnost, bijela kutija omogućava detaljan uvid u unutarnje dijelove aplikacije. Ova strategija testiranja promatra aplikaciju iz perspektive programera. Koristeći ovu strategiju, tester gleda na aplikaciju kao na prozirnu kutiju i koristi svoje znanje o kodu, resursima i bazi podataka aplikacije kako bi ju testirao "iznutra".

2.6.3. Testiranje na temelju iskustva

Testiranje na temelju iskustva je pristup testiranju softvera u kojem se koriste znanje, intuicija i iskustvo testera kako bi se identificirale potencijalne greške i problemi u aplikaciji. Ova metoda ne zahtijeva formalne testne slučajeve ili skripte, već se oslanja na vještine i razumijevanje testera o aplikaciji i njenom kontekstu.

3. AUTOMATIZIRANO TESTIRANJE

Automatizirano testiranje je tehnika testiranja softvera koje se izvodi korištenjem posebnih alata za automatizirano testiranje kako bi izvršio niz testnih slučajeva. Za razliku od ručnog testiranja, koje obavlja tester koji sjedi ispred računala i pažljivo izvodi testne korake, pri automatiziranom testiranju tester piše testove, priprema testne podatke i okolinu, a samo testiranje izvršava program.

Iako je ručno testiranje možda inicijalno jednostavnije i ne zahtjeva veliko tehničko znanje testera, testovi se tokom testiranja izvršavaju nekoliko puta te je za pokrivanje svih pozitivnih i negativnih testnih scenarija to vremenski zahtjevno. Osim toga, aplikacije su sve složenije i zahtjevnije pa samim time sve više raste popularnost automatizacije.

Automatizacija testiranja softvera zahtijeva značajna ulaganja novca i resursa, ali omogućuje ponavljanje istog skupa testova u uzastopnim razvojnim ciklusima bez potrebe za ljudskom intervencijom. Ovo poboljšava povrat ulaganja u automatizaciju testiranja. Cilj automatizacije nije eliminirati ručno testiranje u potpunosti, već smanjiti broj testnih slučajeva koji se moraju izvoditi ručno.

3.1. Prednosti automatiziranog testiranja

Automatizirano testiranje donosi brojne prednosti koje su ključne za suvremeni razvoj softverskih sustava. Iako postoji mnogo razloga za primjenu automatizacije u testiranju, najvažniji su svakako ušteda vremena i novca. Računalu je uvijek potrebno manje vremena da izvrši test nego čovjeku, a s kraćim vremenom izvršenja testova oslobađaju se ljudski resursi koji se mogu posvetiti drugim važnim zadacima. Na taj način, tim postaje efikasniji i profitabilniji.

Ova ušteda vremena ne pridonosi samo testerima, već i drugim članovima tima. Testovi se mogu pokretati na zahtjev, što znači da svaki projektni menadžer ili programer može pokrenuti testove i brzo dobiti izvješća s rezultatima. Ova brzina povratne informacije omogućuje brže otkrivanje i ispravljanje grešaka, čime se poboljšava kvaliteta softvera i ubrzava proces isporuke.

Osim pokretanja testova na zahtjev, testove možemo zakazati da se izvršavaju u unaprijed određeno vrijeme, bez potrebe za ljudskom intervencijom. To znači da se testovi mogu izvoditi preko noći, tijekom vikenda ili u bilo koje drugo vrijeme kada tim nije aktivan. To se postiže tako da ih se integrira u CI/CD okruženje i namjesti da se izvrše kada je potrebno. Na taj način,

maksimalno se iskorištava vrijeme dostupno za testiranje, što dodatno ubrzava isporuku softvera. Također, tako osiguravamo da je verzija aplikacije koja se isporučuje korisnicima uvijek funkcionalna i bez kritičnih grešaka [3], [5], [18].

3.2. Nedostatci automatiziranog testiranja

Prema [4], [18] i [19], automatsko testiranje ima nekoliko nedostataka i ograničenja koje je važno uzeti u obzir prilikom odlučivanja o njegovoj primjeni. Prvo, automatsko testiranje nije prikladno za sve vrste testiranja. Iako je idealno za testiranje jedinice, sigurnost, performansi i regresiju, nije dovoljno za istraživačko testiranje, testiranje korisničkog sučelja (*engl. User interface*) i provjere koje zahtijevaju ljudsko oko. U tim slučajevima je potrebna ljudska intervencija kako bi se osigurala kvaliteta i otkrile greške koje automatski testovi možda neće uočiti. Drugo, automatsko testiranje može biti vrlo skupo, posebno u početnoj fazi. Potrebno je značajno ulaganje za nabavu testnog softvera i razvoj testnih slučajeva, a troškovi održavanja također nisu niski. Ovo može predstavljati financijski izazov, posebno za manje tvrtke ili projekte s ograničenim budžetom. Treće, automatsko testiranje zahtijeva visoku razinu stručnosti. Dizajniranje automatiziranih testova zahtijeva vještine u pisanju koda, a greške u testovima mogu rezultirati lažno pozitivnim i lažno negativnim rezultatima. To može otežati otkrivanje stvarnih problema i zahtijevati dodatno vrijeme i resurse za ispravljanje pogrešaka u testovima. Četvrto, automatsko testiranje može imati ograničenja kod složenih zadataka. Dok radi savršeno s jednostavnim zadacima, može biti manje precizno kod složenih zadataka. Automatski testovi često ne uspijevaju ako postoje problemi u pozadinskom sustavu (*engl. backend*), što može dovesti do propuštanja važnih grešaka. Ova ograničenja čine automatsko testiranje manje poželjnim izborom za neke tvrtke, osobito u situacijama gdje je potrebna visoka prilagodljivost i detaljna provjera korisničkog iskustva.

3.3. Koji testni slučajevi se automatiziraju?

Automatizacija testiranja ima značajne prednosti, ali je važno odabrati koje testne slučajeve automatizirati kako bi se postigla maksimalna isplativost. Krivi odabir može rezultirati višim troškovima nego što bi se uštedjelo automatizacijom. Stoga je ključno slijediti određene kriterije prilikom odlučivanja koje testne slučajeve automatizirati.

Automatizacija se najviše isplati za testne slučajeve koji se često ponavljaju. Nema smisla automatizirati test koji će se izvršiti samo jednom, jer se troškovi inicijalne implementacije automatizacije isplate tek nakon višekratne uporabe. Izrada testnih skripti zahtijeva značajno vrijeme i trud, pa je bitno da se koristi za zadatke koji će se često ponavljati. Regresijsko

testiranje je idealno za automatizaciju jer omogućuje provjeru da nove funkcionalnosti nisu narušile postojeće funkcionalnosti. Testni slučajevi koji imaju veliku poslovnu vrijednost, odnosno koji su ključni za poslovne procese aplikacije, također trebaju biti automatizirani. Ovi testovi su kritični jer njihov neuspjeh može izazvati značajnu štetu. Automatizacijom ovih testova osiguravamo da najvažniji dijelovi sustava uvijek funkcioniraju ispravno. Također, automatski testiramo one testne slučajeve koje je teško i vremenski zahtjevno izvoditi ručno, čime štedimo vrijeme i resurse.

S druge strane, postoje testni slučajevi koje ne treba automatizirati. Testni slučajevi koje nismo barem jednom prošli ručno i istraživački kako bismo provjerili korisničko iskustvo i osnovne funkcionalnosti ne treba automatizirati jer automatski testovi ne mogu kvalitetno procijeniti korisničko iskustvo. Također, testni slučajevi čiji se zahtjevi često mijenjaju nisu pogodni za automatizaciju jer bi to zahtijevalo stalno dorađivanje koda testova, što bi povećalo troškove i smanjilo isplativost. Konačno, istraživački testovi, koji se provode prema intuiciji testera ne mogu se učinkovito automatizirati jer računalo još uvijek ne može na taj način zamijeniti čovjeka.

Automatizacija testiranja donosi brojne prednosti, ali je važno pažljivo odabrati koje testne slučajeve automatizirati kako bi se osigurala maksimalna isplativost i učinkovitost procesa testiranja.

3.4. Alati za automatizaciju

Na tržištu postoji mnogo alata za automatizaciju testiranja, ali nekoliko njih se ističe svojom popularnošću, funkcionalnošću i prilagodljivošću različitim potrebama projekata. Među najvažnijim alatima za automatizaciju su Katalon, Selenium, Cypress i Playwright. Katalon je sveobuhvatan alat za automatizaciju testiranja koji nudi podršku za web, mobilne i API testove. Jedna od njegovih glavnih prednosti je jednostavnost korištenja, što ga čini pristupačnim i za korisnike s manjim iskustvom u automatizaciji. Katalon dolazi s ugrađenim funkcionalnostima koje omogućuju brzo postavljanje i izvršavanje testova bez potrebe za značajnim prilagođavanjima [20]. Selenium je jedan od najstarijih i najčešće korištenih alata za automatizaciju web aplikacija. Pruža bogat set funkcionalnosti i podržava različite preglednike i operativne sustave, čineći ga vrlo fleksibilnim alatom za različite vrste projekata. Selenium omogućuje testiranje aplikacija u stvarnom vremenu i podržava različite programske jezike kao što su Java, C#, Python i JavaScript [21]. Cypress je noviji alat koji je stekao popularnost zbog svoje brzine, lakoće korištenja i sveobuhvatnog pristupa testiranju. Cypress je posebno

dizajniran za razvojne inženjere za web sučelja i nudi jednostavnu instalaciju i konfiguraciju. Njegova arhitektura omogućuje brzo izvršavanje testova i pruža odlične mogućnosti za testiranje modernih web aplikacija [22]. Playwright je alat razvijen od strane Microsofta koji pruža slične funkcionalnosti kao i Cypress, ali s dodatnim prednostima. Playwright se ističe svojom sposobnošću simulacije složenih korisničkih scenarija i nudi detaljnu kontrolu nad mrežnim zahtjevima i odgovorima [23]. U ovom diplomskom radu detaljnije će biti opisani alati Cypress i Playwright. Biti će prikazano kako se ovi alati koriste, koje su im glavne prednosti i kako se mogu implementirati u različite testne scenarije.

4. CYPRESS BIBLIOTEKA

Cypress je besplatan alat otvorenog izvora za automatizaciju testiranja, licenciran pod MIT (engl. *Massachusetts Institute of Technology*) licencom i napisan u JavaScriptu. Pruža moderan i kompletan okvir za testiranje od kraja do kraja kojem je cilj ujediniti testiranje web aplikacija bez obzira na korišteni programski jezik. Radi na svim platformama i podržava sljedeće web preglednike: Chrome, Electron, Mozzila Firefox i Microsoft Edge [23].

4.1. Cypress naredbe

Prema [24], Cypress nudi bogat skup naredbi koje omogućuju razne operacije tijekom testiranja web aplikacija. Ove naredbe mogu se klasificirati u nekoliko kategorija prema svojoj namjeni i funkcionalnosti: upiti (*engl. query*), tvrdnje (*engl. assertion*), akcije (*engl. actions*) i na ostale naredbe.

4.1.1. Upiti

Upiti se koriste za čitanje stanja aplikacije, omogućuju dohvaćanje elemenata objektnog modela sučelja (*engl. Document Object Model, DOM*), njegovih atributa i vrijednosti. To su ključne naredbe za provjeru elemenata koji su prisutni na stranici i za daljnju interakciju s njima. Neki od upita su:

- *get()* – dohvaća DOM elemente prema selektoru, koristi se za pronalaženje elemenata na stranici kako bi se moglo s njima dalje manipulirati
- *find()* – traži potomke unutar prethodno dohvaćenog elementa, koristi se za pronalaženje elemenata unutar drugog elementa
- *contains()* – dohvaća elemente koji sadrže određeni tekst, koristi se za pronalaženje elemenata prema njihovom tekstualnom sadržaju
- *title()* – dohvaća naslov trenutne stranice, koristi se za provjeru naslova stranice
- *url()* – dohvaća URL trenutne stranice, koristi se za provjeru URL-a stranice

4.1.2. Tvrdnje

Tvrdnje služe za provjeru određenog stanja aplikacije. One omogućuju postavljanje uvjeta koje aplikacija mora ispuniti kako bi test bio uspješan. Najčešće korištene tvrdnje u Cypressu su:

- *should()* – izvršava provjeru na dohvaćenom elementu ili vrijednosti

- *and()* – nastavlja provjeru na istom elementu ili vrijednosti nakon *should()*, koristi se za nizanje tvrdnji
- *expect()* – koristi se za provjeru vrijednosti izvan Cypress naredbi

4.1.3. Akcije

Akcije se koriste za interakciju s aplikacijom na način kako bi to radio korisnik. To uključuje klikanje, tipkanje, odabir, povlačenje i puštanje elemenata, te druge akcije koje mijenjaju stanje aplikacije. Primjeri akcija uključuju:

- *click()* – klikne na dohvaćeni element, koristi se za simulaciju korisničkog klika
- *type()* – upisuje tekst u dohvaćeni element za upis, koristi se za simulaciju unosa teksta
- *select()* – odabire opciju u dohvaćenom elementu za odabir, koristi se za simulaciju odabira iz padajućeg izbornika
- *drag()* – povlači dohvaćeni element na određenu poziciju, koristi se za simulaciju povuci-pusti (*engl. drag and drop*) akcije
- *drop()* – pušta dohvaćeni element na ciljnu poziciju, koristi se zajedno s *cy.drag()*
- *check()* – označava checkbox, koristi se za simulaciju označavanja

4.1.4. Ostale naredbe

Ova kategorija obuhvaća naredbe koje ne spadaju u prethodne tri kategorije, ali su korisne za pisanje testova. To može uključivati naredbe za dohvaćanje i postavljanje kolačića, manipulaciju vremenom, upravljanje mrežnim zahtjevima, te ostale specijalizirane funkcije. Primjeri drugih naredbi uključuju:

- *visit()* - posjećuje određeni URL (engl. *Uniform Resource Locator*), koristi se za navigaciju na određenu stranicu
- *reload()* - ponovno učitava trenutnu stranicu, koristi se za simulaciju osvježavanja stranice
- *intercept()* - presreće i manipulira mrežnim zahtjevima, koristi se za simulaciju različitih odgovora servera
- *wait()* - čeka određeno vrijeme ili dok određeni zahtjev ne bude završen, koristi se za kontrolu toka testova
- *screenshot()* - snima zaslon trenutne stranice, koristi se za dokumentiranje stanja stranice tijekom testiranja

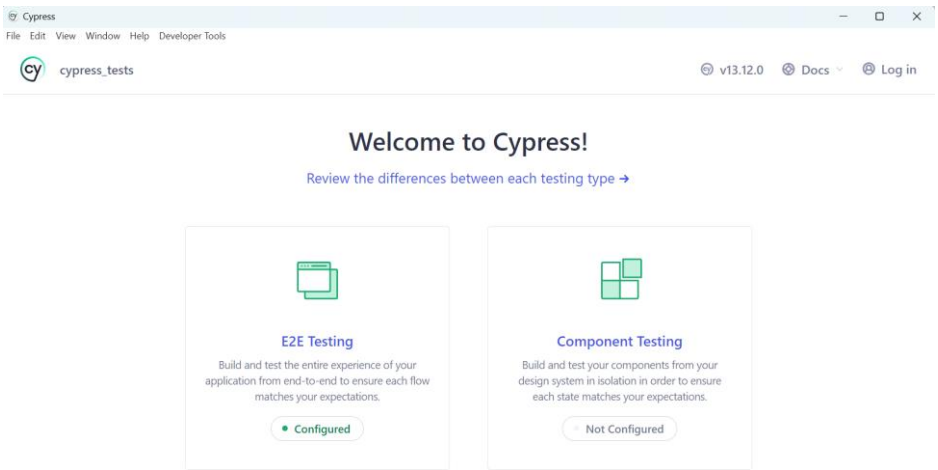
4.1.5. Mocha.js

Mocha.js je ugrađena JavaScript biblioteka koja se koristi unutar Cypressa za definiranje i organiziranje testova. Omogućava strukturiranje testova pomoću opisnih blokova, što olakšava pisanje, čitanje i održavanje testova. Neke od osnovnih naredbi koje omogućava su:

- *describe()* – koristi se za grupiranje povezanih tekstova, omogućuje organizaciju testova u logičke cjeline
- *it()* – definira pojedinačne testne slučajeve, opisuje što testira
- *before()* – izvodi se jednom prije svih testova koji se nalaze u 'describe' bloku, koristi se za postavljanje uvjeta potrebnih za testiranje
- *after()* – izvodi se jednom nakon svih testova koji se nalaze u 'describe' bloku, koristi se za čišćenje nakon testiranja
- *beforeEach()* – izvodi se prije svakog testa u 'describe' bloku, koristi se za postavljanje stanja potrebnog za svaki pojedinačni test
- *afterEach()* – izvodi se nakon svakog testa u 'describe' bloku, koristi se za vraćanje stanja nakon svakog testa

4.2. Tipovi Cypress testiranja

Prema [25], postoje dva tipa testiranja u Cypressu, a to su testiranje od kraja do kraja (E2E) i testiranje komponente. Nakon pokretanja alata, korisniku se postavlja mogućnost odabira tipa testiranja (Slika 4.1)



Slika 4.1 Početni zaslon Cypress aplikacije

4.2.1. E2E testiranje

E2E (*engl. End-to-end*) testiranje je tip testiranja aplikacije od korisničkog sučelja do pozadinskog dijela aplikacije. Ova vrsta testiranja je izvrsna za osiguranje da cijela aplikacija funkcionira kao cjelina. Cypress pokreće E2E testove na isti način na koji korisnici komuniciraju s aplikacijom koristeći pravi preglednik, posjećujući URL-ove, pregledavajući sadržaj, pritišćući na poveznice i gumbe. Testiranje na ovaj način pomaže osigurati da su testovi i korisničko iskustvo isti. Pisanje E2E testova u Cypressu obavljaju programeri koji grade aplikaciju, specijalizirani inženjeri za testiranje ili tim za osiguranje kvalitete koji je odgovoran za provjeru spremnosti aplikacije za izdavanje. Testovi se pišu u kodu pomoću API-ja koji simulira korake koje bi stvarni korisnik poduzeo. E2E testovi su izvrsni za provjeru ispravnosti rada aplikacije, od sučelja aplikacije do pozadinskog dijela. Međutim, E2E testovi mogu biti zahtjevniji za postavljanje, pokretanje i održavanje.

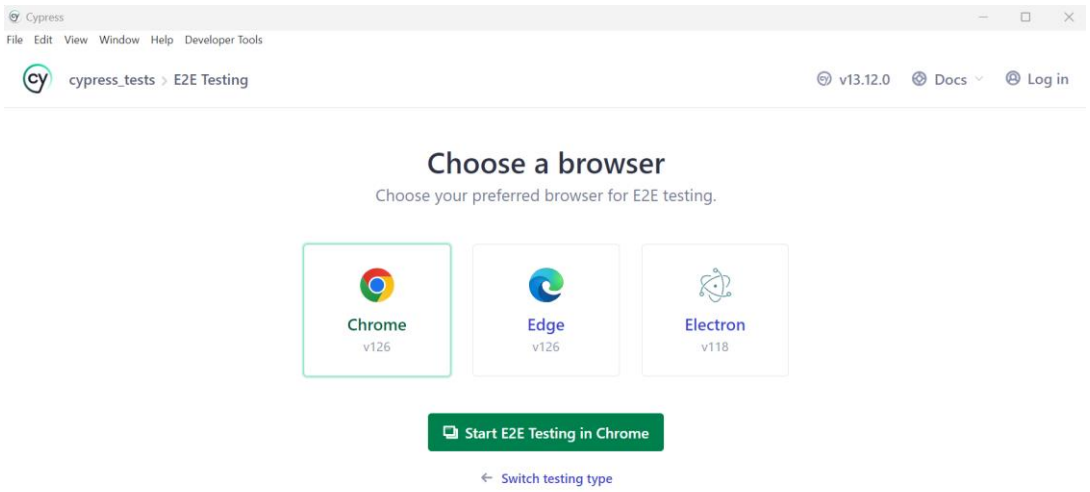
4.2.2. Testiranje komponenti

Moderni web okviri omogućuju izradu aplikacija dijeljenjem u manje jedinice logike, odnosno komponente. Te komponente mogu varirati od vrlo jednostavnih, poput gumba, do složenijih, poput obrazaca za registraciju. Komponente je često lako testirati, a upravo tu dolazi do izražaja Cypressovo testiranje komponenti. Za razliku od E2E testova, koji zahtijevaju posjećivanje URL-a za pokretanje cijele aplikacije, testovi komponenti omogućuju namještanje i samostalno testiranje pojedine komponente. To omogućuje fokusiranje isključivo na funkcionalnost komponente, bez utjecaja drugih dijelova aplikacije. Ipak, važno je napomenuti da uspješnost svih testova komponenti ne jamči ispravnost cijele aplikacije. Testovi komponenti ne osiguravaju ispravan rad svih slojeva aplikacije zajedno. Stoga, cjelovito testirana aplikacija kombinira E2E testove i testove komponenti, pri čemu svaka vrsta testova pokriva svoje specifične aspekte.

Na ovaj način, testiranje komponenti omogućuje temeljito testiranje pojedinačnih dijelova aplikacije, dok E2E testovi osiguravaju koordiniran rad svih dijelova aplikacije kao cjeline.

4.3. Cypress sučelje

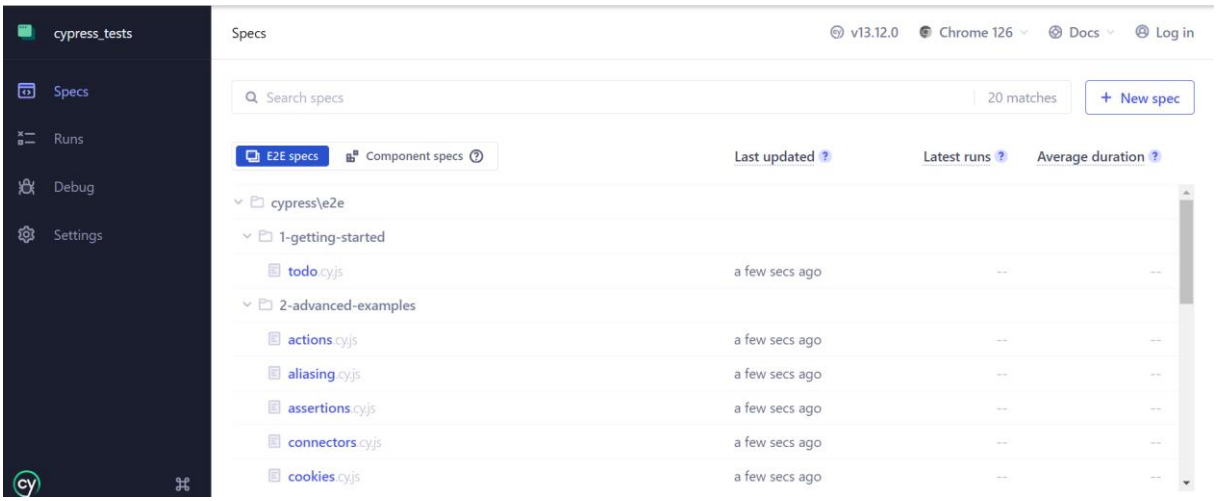
Izvršavanjem naredbe *npx cypress open* u terminalu otvara se Cypress aplikacija. Slika 4.2 prikazuje početni zaslon nakon pokretanja Cypressa i odabira tipa testiranja, gdje korisnik može odabrati odgovarajući web preglednik.



Slika 4.2 Odabir web preglednika

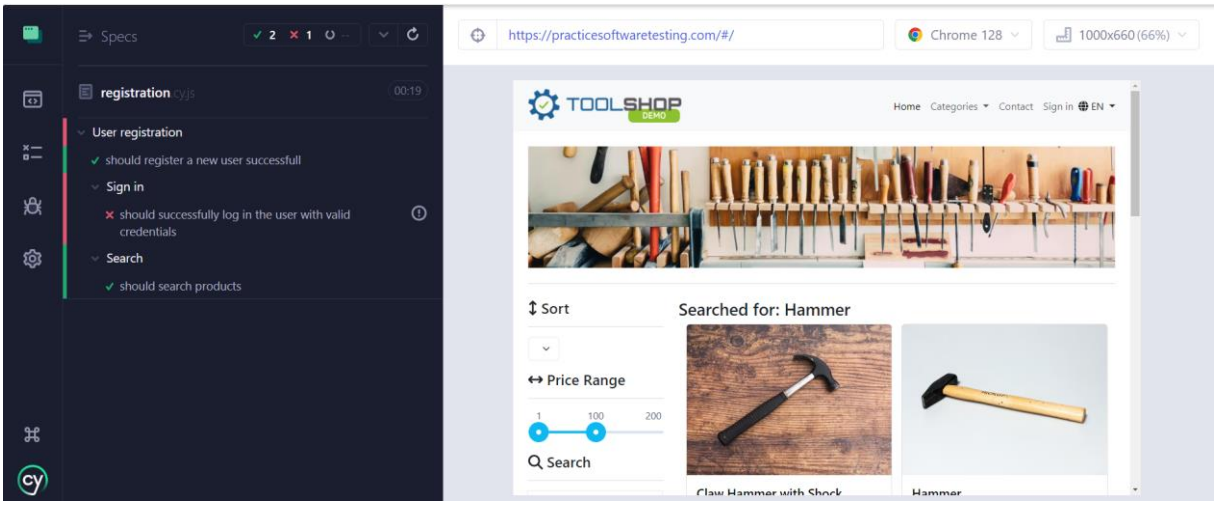
4.4. Pokretanje testova

Nakon odabira preglednika, otvara se Cypress Test Runner sučelje, odnosno sučelje za pokretanje testova. Prikazan je popis specifikacija s njihovim nazivima, lokacijama i informacijama o najnovijim izvršenjima (*engl. Specs View*) (Slika 4.3). Ovdje je moguće pokrenuti specifikacije klikom na njih, stvoriti nove te pretraživati specifikacije po imenu što je korisno za velike skupove testova.



Slika 4.3 Prikaz specifikacija

Nakon pokretanja željene specifikacije, započinje izvršavanje koje korisnik može pratiti. Nakon završenog izvršavanja s lijeve strane se nalaze rezultati izvršenih testova (Slika 4.4).



Slika 4.4 Cypress Test Runner

5. PLAYWRIGHT BIBLIOTEKA

Playwright je biblioteka za automatizirano testiranje koju je razvio Microsoft. Iako je relativno nov alat, objavljen 2020. godine, Playwright je brzo stekao popularnost zahvaljujući redovnim ažuriranjima i poboljšanjima na osnovu povratnih informacija korisnika. Ova popularnost se jasno vidi i kada se usporedi broj preuzimanja sličnih alata koji su već duže vrijeme na tržištu.

5.1. Playwright naredbe

Kao i Cypress, Playwright ima slične naredbe za izvođenje raznih akcija. Prema [26], podijelit ćemo ih na osnovne naredbe, tvrdnje i lokatore. Kako bi se bolje razumjelo kako rade pojedine naredbe, važno je objasniti način njihove upotrebe.

Page je objekt koji predstavlja stranicu preglednika kojom upravlja Playwright. Može se smatrati alatom koji omogućuje interakciju sa stranicom i izvršavanje raznih akcija, poput navigacije, unosa teksta ili klika na elemente. *Locator* je objekt koji označava određeni element na stranici, on omogućava radnje kao što su klikanje, provjera vidljivosti, dohvaćanje teksta i slično. Razlika između *page* i *locator* objekata je to što *page* predstavlja cijelu stranicu, a *locator* predstavlja specifične elemente na toj stranici.

Await je ključna riječ u JavaScriptu koja se koristi unutar *async* funkcija kako bi se pričekalo na završetak asinkrone operacije prije nego što se prijeđe na sljedeću liniju koda. To je bitno za ispravno izvršavanje Playwright skripti jer omogućuje da se svaka radnja završi prije nastavka. Također, bitno je spomenuti funkciju *expect* koja se koristi za provjeru i validaciju rezultata nakon određene radnje. Služi za definiranje očekivanja o stanju stranice ili elementima na njoj te provjeru jesu li ta očekivanja ispunjena, što je ključno za provjeru točnosti i pouzdanosti testova [27].

5.1.1. Osnovne naredbe

Osnovnim naredbama u Playwrightu se smatraju razne naredbe za pokretanje preglednika, rad sa stranicama, odabir elemenata na stranici i slično. Te naredbe su prikazane i objašnjene u nastavku.

- *page.goto()* – posjećuje određeni URL, koristi se za navigaciju na određenu stranicu
- *page.close()* – zatvara određenu stranicu
- *page.locator()* – kreira lokator za odabir elemenata
- *locator.fill()* – popunjava polja unutar forme

- *locator.check()* – označava checkbox, koristi se za odabiranje elemenata
- *locator.selectOption()* – odabire određenu opciju unutar elementa, može se koristiti za odabir opcije prema vrijednosti, tekstu ili indeksu
- *locator.click()* – simulira klik na element, koristi se za interakciju s gumbima i linkovima
- *locator.hover()* – simulira hover na element, odnosno pomiče pokazivač miša preko određenog elementa bez klika
- *locator.press()* – simulira pritiskanje tipke na tipkovnici

5.1.2. Tvrdnje

Tvrdnje u Playwrightu služe za validaciju stanja elemenata i stranice tijekom testiranja. Koriste se za provjeru očekivanog ponašanja aplikacije, poput vidljivosti elemenata, ispravnog teksta ili provjere da su određeni elementi omogućeni ili onemogućeni. Tvrdnje osiguravaju da aplikacija radi kako je predviđeno i pomažu u otkrivanju grešaka.

- *await expect(locator).toBeChecked()* – provjerava je li checkbox označen
- *await expect(locator).toBeDisabled()* – provjerava je li element onemogućen za interakciju
- *await expect(locator).toBeVisible()* – provjerava je li element vidljiv korisniku na stranici
- *await expect(locator).toHaveText()* – provjerava sadrži li element točan tekst
- *await expect(page).toHaveTitle()* – provjerava je li naslov stranice jednak očekivanom naslovu stranice

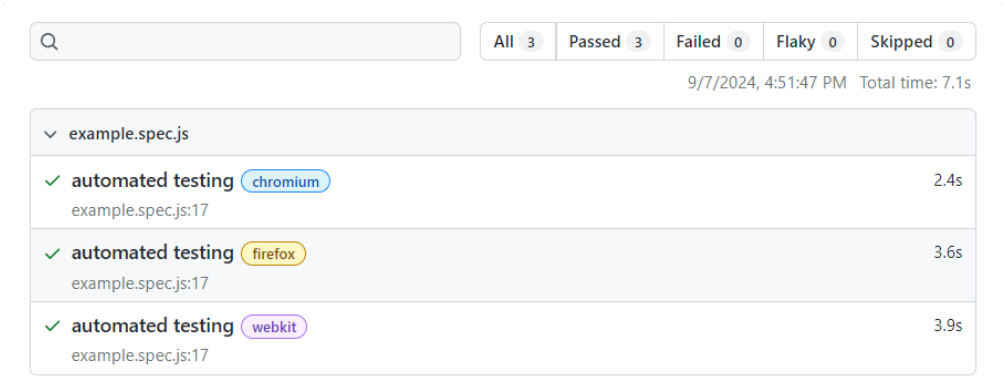
5.1.3. Lokatori

Lokatori su metode koje se koriste za pronalaženje elemenata na stranici u bilo kojem trenutku.

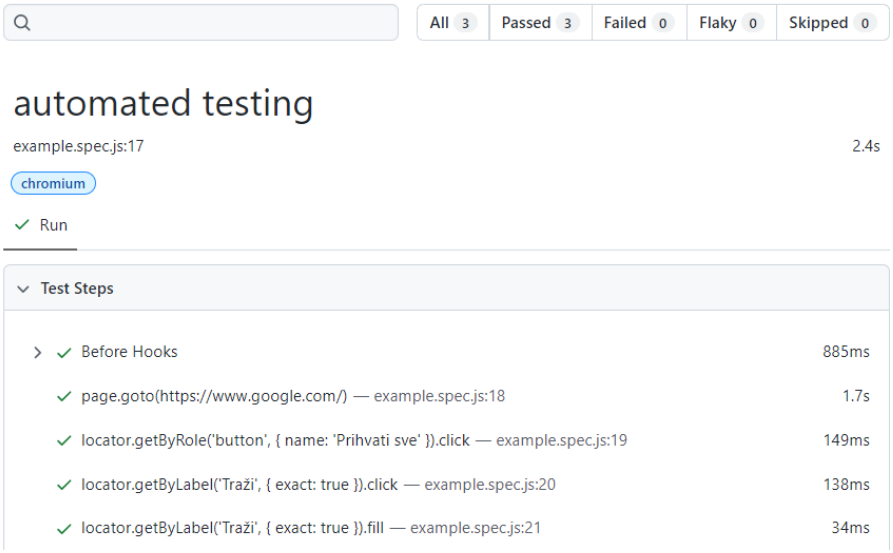
- *page.getByRole()* – pronalazi element na stranici prema njegovoj ulozi
- *page.getByText()* – pronalazi elemente prema prikazanom tekstu, korisno kada je potrebno pronaći element na temelju njegovog vidljivog sadržaja
- *page.getByLabel()* – pronalazi polja unosa na stranici pomoću njihovih oznaka
- *page.getByPlaceholder()* – pronalazi elemente na temelju pomoćnog teksta unutar polja za unos, često se koristi za polja unosa s privremenim tekstom koji opisuje što treba unijeti
- *page.getByTitle()* – pronalazi elemente na temelju atributa 'title'

5.2. Pokretanje testova

Playwright omogućava pokretanje zasebnog testa ili cijelog seta testova i to se izvodi naredbom *npx playwright test*. Testove je moguće pokretati na jednom ili više preglednika istovremeno što se definira opcijom *--project*. Na primjer, ukoliko korisnik želi pokrenuti test samo u Chromium pregledniku potrebno je izvršiti naredbu *npx playwright test --project=chromium*. U Playwrightu je predefinirano pokretanje testova bez otvaranja preglednika, odnosno u *headless* načinu, već će se samo rezultati testova vidjeti u terminalu. Ako korisnik želi pratiti izvršavanje u pregledniku, moguće je koristiti *--headed* opciju koja otvara preglednik tijekom testiranja. Nakon što su testovi pokrenuti moguće je pregledati testni izvještaj (*engl. test report*). Njih je moguće filtrirati po testovima koji su prošli, nisu uspjeli ili su preskočeni te po pregledniku. Primjer testnog izvještaja je na slici 5.1, a detalji o pojedinom testu na slici 5.2.



Slika 5.1 Testni izvještaj



Slika 5.2 Detalji pojedinog testa

5.3. Generiranje testova

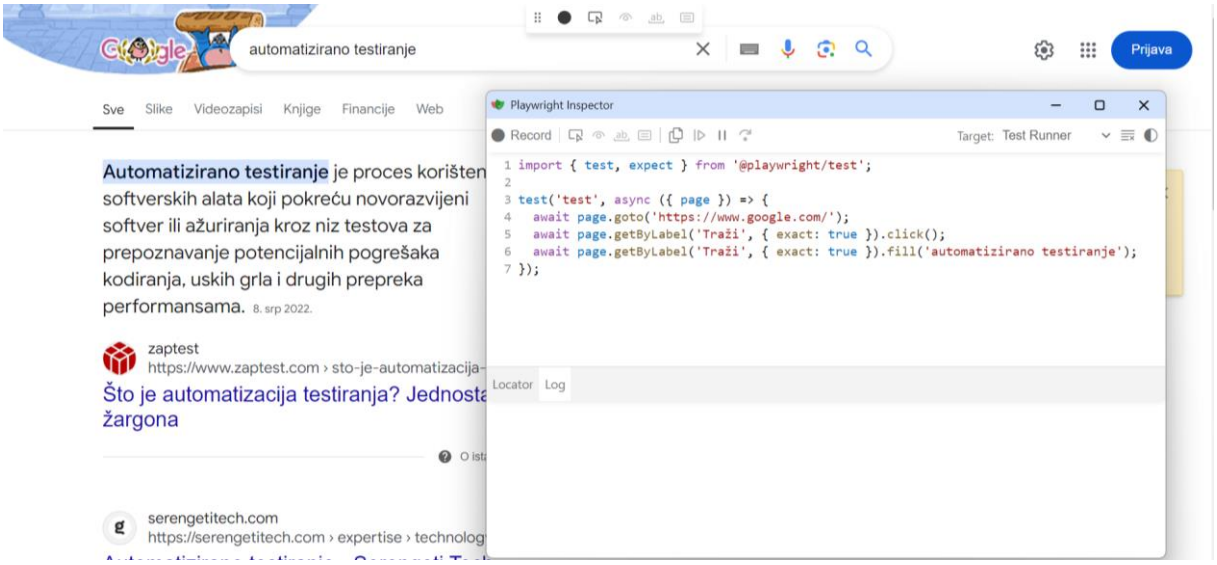
Playwright dolazi s ugrađenom funkcionalnošću generiranja testova, što omogućuje brz i jednostavan početak testiranja web aplikacija. Ovaj alat automatski otvara dva prozora: prvi prozor je preglednik u kojem korisnik može interaktivno komunicirati s web stranicom koju želi testirati, a drugi je Playwright Inspector. U tom prozoru korisnik može snimati radnje koje se izvode na web stranici, nakon čega Playwright automatski generira testove na temelju tih radnji.

Osim snimke testova, Playwright Inspector nudi i mogućnost pregledavanja i uređivanja generiranih testova, kopiranja koda, brisanja snimljenih radnji te promjene jezika testnih skripti, što omogućuje fleksibilnost ovisno o projektu. Ovaj pristup značajno ubrzava proces automatizacije testiranja jer omogućuje lako dokumentiranje korisničkih interakcija i generiranje testova bez potrebe za pisanjem koda od nule.

Alat je posebno koristan za početnike u automatiziranom testiranju, ali i za iskusnije korisnike koji žele brzo kreirati osnovne testne slučajeve te ih potom dodatno doraditi po potrebi.

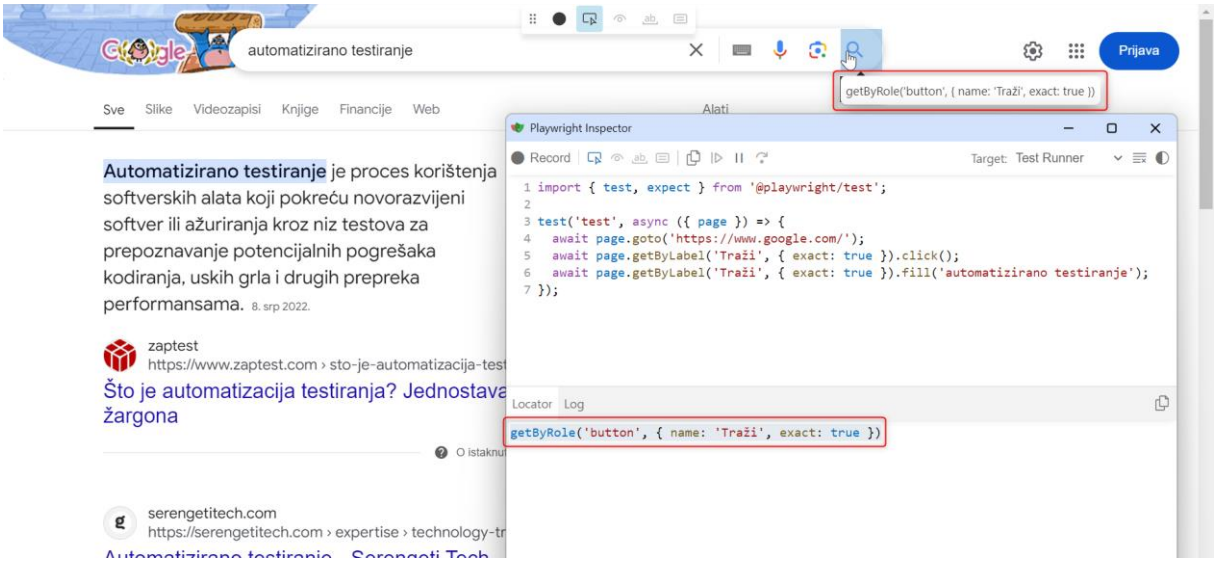
5.3.1. Codegen alat za generiranje testova

Za pokretanje generiranja testova koristi se naredba *npx playwright codegen* `<URL>`. Nije nužno odmah unijeti URL, već je moguće pokrenuti Codegen i bez njega, pa zatim naknadno unijeti željeni URL u pregledniku. Na slici 5.3 prikazana su dva prozora, kao što je ranije objašnjeno. Prozor u kojem se obavlja interakcija je Chrome preglednik, dok Playwright Inspector automatski generira kod za svaku akciju koju korisnik izvrši u pregledniku.



Slika 5.3. Codegen

Playwright Instructor također omogućuje generiranje ranije spomenutih lokatora. Nakon završetka snimanja, omogućit će se gumb 'Pick Locator'. Klikom na gumb, prelazak mišem preko svakog elementa će prikazati njegov lokator , a klik na njega će generirati kod (Slika 5.4).



Slika 5.4. Generiranje lokatora

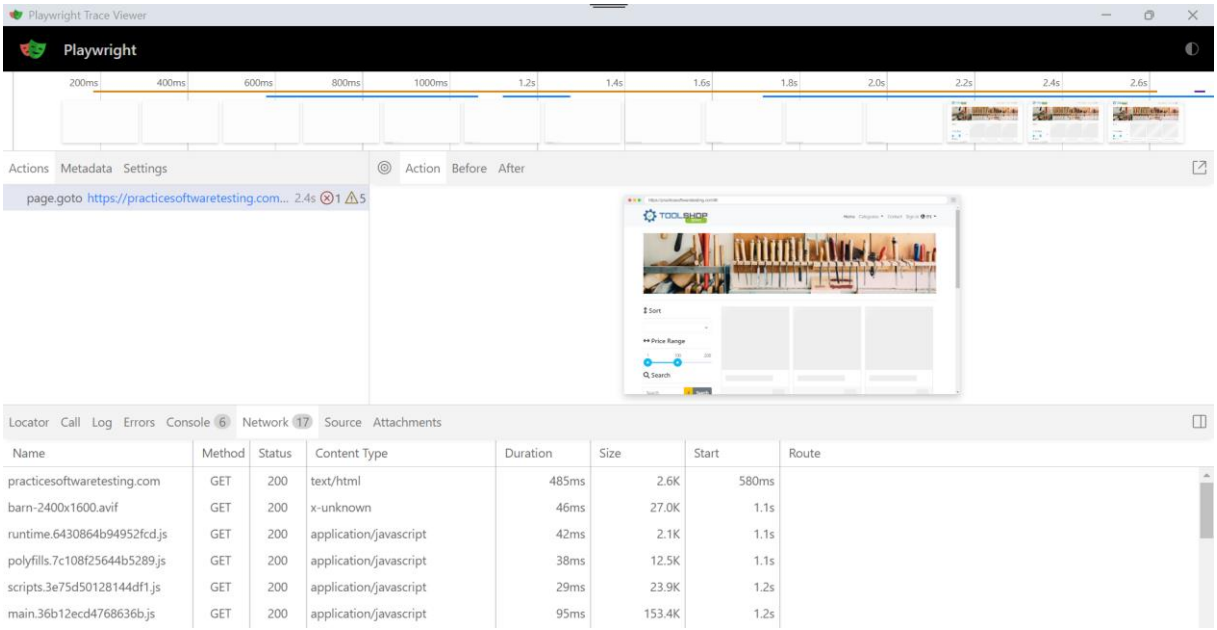
5.4. Trace Viewer programski alat

Playwright Trace Viewer je grafičko korisničko sučelje koje omogućuje analiziranje i istraživanje snimljenih testova [28]. Korisniku pruža mogućnost uvida u detaljne podatke o svemu što se dogodilo za vrijeme testa, uključujući svaki korak, snimke zaslona, poruke konzole, greške i slično. Za korištenje Trace Viewera potrebno je generirati datoteku i omogućiti praćenje na početku testa i isključiti ga na kraju što je prikazano u programskom kodu 5.1. Nakon što je datoteka generirana, potrebno je pokrenuti naredbu *npx playwright show-trace trace.zip.*

```
test('example test with trace', async ({ page, context }) => {
  // Pokreni praćenje
  await context.tracing.start( { screenshots: true, snapshots: true });

  await page.goto('https://practicesoftwaretesting.com/#/');

  // Zaustavi praćenje i spremi trace datoteku
  await context.tracing.stop({ path: 'trace.zip' });
});
```

Programski kod 5.1

Nakon pokretanja naredbe, otvara se prozor Playwright Trace Viewer koji je prikazan na slici 5.5. Prikazana je vremenska linija (*engl. timeline*) koja prikazuje vizualni pregled svih koraka koji su se dogodili tijekom testa. S lijeve strane nalaze se detalji svakog izvedenog koraka što uključuje navigaciju, klik miša, unos teksta i slično. Na dnu prozora moguće je pregledati mrežne zahtjeve, konzolu, logove, izvor i ostalo.



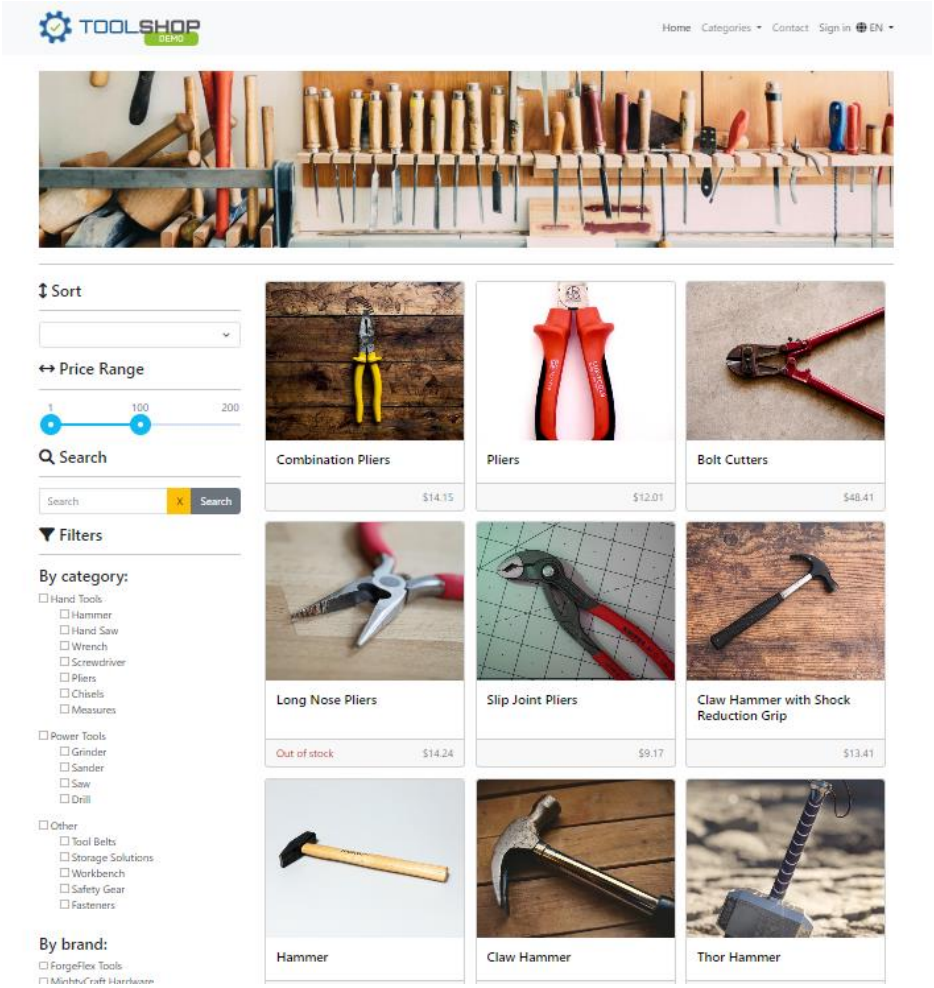
Slika 5.5 Playwright Trace Viewer prozor

6. AUTOMATIZIRANO TESTIRANJE WEB APLIKACIJE

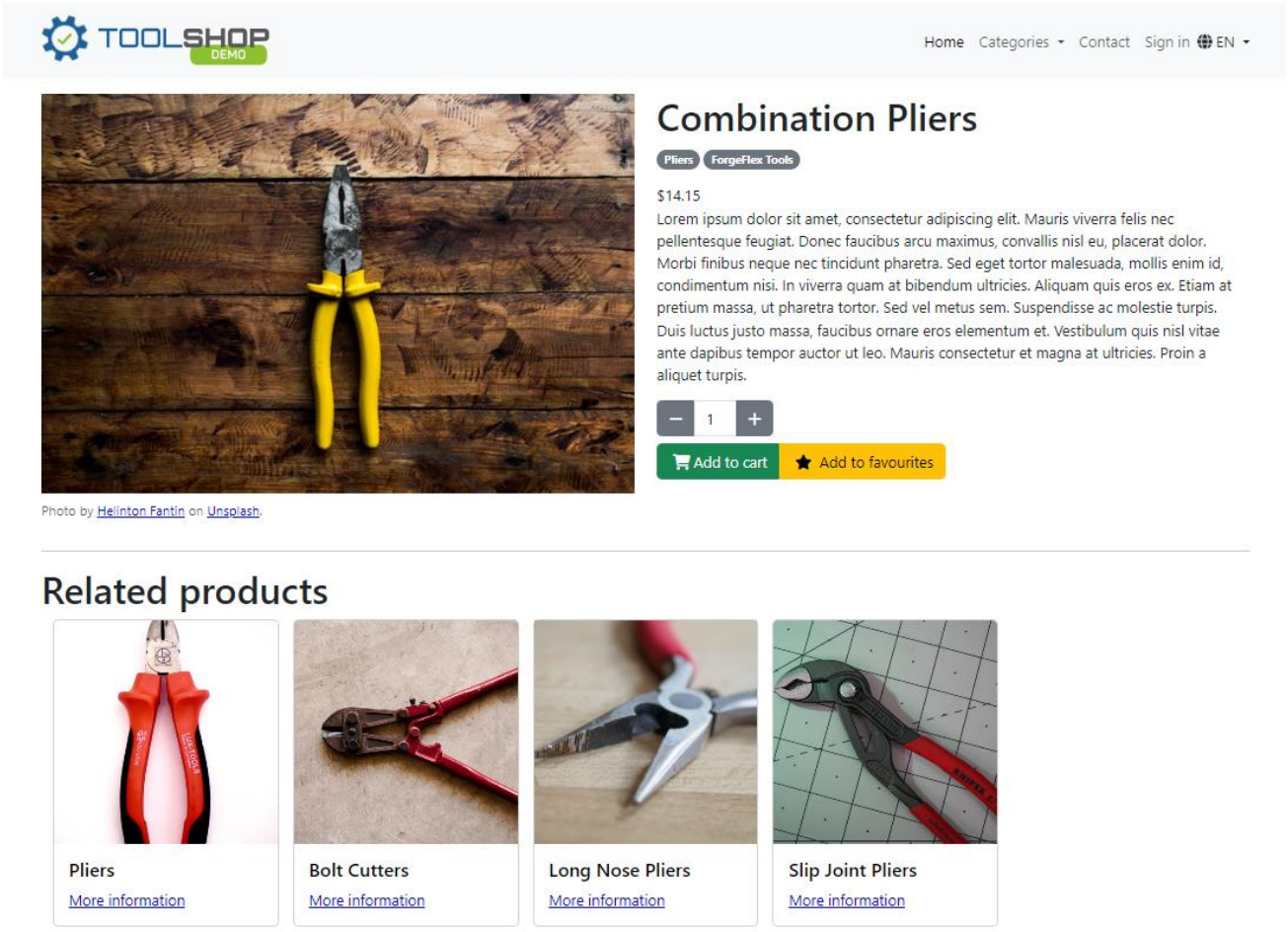
U ovom poglavlju opisat će se web aplikacija i testni plan koji je korišten za automatizaciju testova u alatima Cypress i Playwright.

6.1. Web aplikacija za prodaju alata

Web aplikacija za prodaju alata predstavlja online stranicu koja korisnicima omogućuje pregled, pretragu i kupovinu različitih vrsta alata. Cilj ove aplikacije je pružiti korisnicima jednostavno i učinkovito iskustvo kupovine, uz široku ponudu proizvoda i korisne informacije o svakom alatu. Osnovne funkcionalnosti aplikacije uključuju pregled kataloga proizvoda uz mogućnost filtriranja prema specifikacijama, cijeni i popularnosti, pretragu proizvoda korištenjem ključnih riječi i filtera, mogućnost kreiranja korisničkog računa i spremanja svih korisnih podataka poput povijesti kupovina i proizvoda u listu želja, također omogućava jednostavan proces dodavanja proizvoda u košaricu te podršku i kontakt. Početna stranica aplikacije i detaljan prikaz proizvoda prikazani su na Slici 6.1. i Slici 6.2.



Slika 6.1 Početna stranica web aplikacije (izvor: https://practicesoftwaretesting.com/#/)



Slika 6.2 Prikaz detalja proizvoda (izvor: https://practicesoftwaretesting.com/#/)

6.2. Testni plan web aplikacije za prodaju alata

Testni plan opisuje raspored testiranja testnih slučajeva potrebnih za verifikaciju projekta. Testni slučajevi će obuhvatiti različite dijelove aplikacije kako bi se osiguralo da svi funkcioniraju ispravno. Svaki testni slučaj će biti detaljno opisan i organiziran prema strukturi koja uključuje naziv, preduvjete, korake i očekivani rezultat. U Tablici 6.1 se nalaze primjeri testnih slučajeva za aplikaciju Tool Shop.

Tablica 6.1. Testni slučajevi za automatizaciju web stranice Tool Shop

Test ID	Testni slučaj	Preduvjet	#	Koraci	Očekivani rezultat
1	Registracija korisnika	Otvoriti: https://practicesoftwaretesting.com/#/.	1. 2. 3. 4.	Pritisnuti gumb „Sign in“ Pritisnuti gumb „Register your account“ Popuniti formu testnim podacima Pritisnuti gumb „Register“	Korisnik je uspješno registriran. Korisnik je preusmjeren na stranicu za prijavu.
2	Prijava korisnika u sustav	Otvoriti: https://practicesoftwaretesting.com/#/. Testni podaci za prijavu su validni.	1. 2. 3.	Pritisnuti gumb „Sign in“ Upisati testne podatke za prijavu Pritisnuti gumb „Login“	Korisnik je uspješno prijavljen. Korisniku je prikazana stranica *My account.*
3	Dodavanje proizvoda u košaricu	Otvoriti: https://practicesoftwaretesting.com/#/. Korisnik je prijavljen u sustav.	1. 2.	Pritisnuti na željeni proizvod Pritisnuti gumb „Add to cart“	Proizvod je uspješno dodan u košaricu. Korisniku se prikaže poruka da je proizvod uspješno dodan u košaricu.
4	Sortiranje proizvoda po nazivu (A-Z)	Otvoriti: https://practicesoftwaretesting.com/#/. Korisnik je prijavljen u sustav.	1. 2.	Pritisnuti na padajući izbornik ispod teksta „Sort“ Iz padajućeg izbornika odabrati „Name (A-Z)“	Proizvodi su uspješno sortirani po nazivu od A do Z.
5	Sortiranje proizvoda po cijeni (od niže prema višoj)	Otvoriti: https://practicesoftwaretesting.com/#/. Korisnik je prijavljen u sustav.	1. 2.	Pritisnuti na padajući izbornik ispod teksta „Sort“ Iz padajućeg izbornika odabrati „Price (Low-High)“	Proizvodi su uspješno sortirani po cijeni od niže prema višoj.
6	Filtriranje po kategoriji	Otvoriti: https://practicesoftwaretesting.com/#/. Korisnik je prijavljen u sustav.	1.	Pritisnuti željene kategorije	Prikazani su samo proizvodi iz odabranih kategorija.
7	Pretraživanje proizvoda	Otvoriti: https://practicesoftwaretesting.com/#/. Korisnik je prijavljen u sustav.	1. 2.	Pritisnuti na „Search“ okvir Upisati naziv željenog proizvoda	Prikazani su proizvodi koji odgovaraju unesenom nazivu.

6.3. Automatizirano testiranje web aplikacije u Cypressu

U ovom potpoglavlju bit će predstavljen programski kod za svaki testni slučaj koji je testiran pomoću Cypressa, uz detaljno objašnjenje. Cilj je prikazati kako se koriste Cypressove funkcionalnosti za automatizirano testiranje ključnih funkcionalnosti web aplikacije i objasniti osnovne metode koje se koriste prilikom testiranja.

6.3.1. Registracija korisnika

U sklopu automatizacije testiranja pomoću Cypressa, prvi implementirani testni slučaj obuhvaća provjeru funkcionalnosti registracije korisnika. Cilj ovog testa je osigurati da korisnik može uspješno dovršiti proces registracije. Testni scenarij započinje posjetom web stranici Tool Shop što se realizira korištenjem funkcije *cy.visit('https://practicesoftware testing.com/#/')*, nakon čega se uz pomoć funkcije *cy.contains('Sign in')* pronalazi gumb „Sign in“, a potom se metodom *click()* simulira klik na taj gumb, čime je korisnik preusmjeren na stranicu za prijavu. S obzirom da korisnik nema postojeći račun, potrebno je kliknuti na gumb „Register your account“, nakon čega se otvara stranica za registraciju gdje Cypress skripta automatski popunjava polja ime, prezime, email adresa i lozinka uz pomoć funkcije *cy.get()*. Nakon unosa podataka, simulira se klik na gumb za registraciju. Na kraju, test provjerava da li je korisnik uspješno preusmjeren na stranicu za prijavu (Programski kod 6.1).

```
describe('User registration', () => {
    it('should register a new user successfull', () => {
        cy.visit('https://practicesoftwaretesting.com/#/')
        cy.contains('Sign in').click()
        cy.contains('Register your account').click()
        cy.get('[data-test="first-name"]').type('Iva')
        cy.get('[data-test="last-name"]').type('Hodić')
        cy.get('[data-test="dob"]').type('2000-01-01')
        cy.get('[data-test="address"]').type('Zagrebačka 1')
        cy.get('[data-test="postcode"]').type('10000')
        cy.get('[data-test="city"]').type('Zagreb')
        cy.get('[data-test="country"]').select('Croatia')
        cy.get('[data-test="phone"]').type('0911234567')
        cy.get('[data-test="email"]').type('ivahodic@dispostable.com')
        cy.get('[data-test="password"]').type('IvaIva123!')
        cy.get([data-test="register-submit"]).click()
        cy.get('[data-test="email"]').type('ivahodic@dispostable.com')
        cy.get('[data-test="password"]').type('IvaIva123!')
        cy.get([data-test="login-submit"]).click()
}) })
```

Programski kod 6.1

6.3.2. Prijava korisnika

Drugi testni slučaj, prikazan Programskim kodom 6.2 obuhvaća provjeru funkcionalnosti prijave korisnika. Za potrebe ovog testnog slučaja potrebno je unaprijed biti registrirani korisnik i imati validne testne podatke. Testni scenarij započinje posjetom web stranici Tool Shop, nakon čega se automatski prepoznaje gumb „Sign in“ i klikom na njega se preusmjerava korisnika na stranicu za prijavu. Na stranici za prijavu popunjavaju polja za email adresu i lozinku koristeći unaprijed definirane validne podatke. Nakon popunjavanja podataka, simulira se klik na gumb „Login“, nakon čega se provjerava nalazi li se korisnik na stranici svojeg profila kao što je to definirano u testnom slučaju. Cilj ovog testa je osigurati da se korisnik može uspješno prijaviti u sustav.

```
describe('Sign in', () => {
    it('should successfully log in the user with valid credentials', () => {

        cy.visit('https://practicesoftwaretesting.com/#/')
        cy.contains('Sign in').click()

        cy.get('[data-test="email"]').type('ivahodic@dispostable.com')
        cy.get('[data-test="password"]').type('IvaIva123!')

        cy.get([data-test="login-submit"]).click()

        cy.get('h1').contains('My account')
      })
})
```

Programski kod 6.2

6.3.3. Dodavanje proizvoda u košaricu

Programski kod 6.3 provjerava funkcionalnost dodavanja određenog proizvoda u ovom slučaju 'Combination Pliers', u košaricu. Ovim testom se simulira ponašanje korisnika koji pretražuje proizvod, dodaje ga u košaricu i očekuje potvrdu da je proizvod uspješno dodan. Nakon posjeta stranici Tool Shop, odabire se proizvod prema traženom tekstu i izvršava se klik na njega. Nakon što je stranica proizvoda otvorena koristi se metoda *cy.get('[data-test="add-to-cart"]').click()* za dodavanje proizvoda u košaricu. Na kraju testnom scenarija provjerava se postoji li poruka koja potvrđuje da je proizvod dodan u košaricu.

```
 describe('Add product to cart', () => {
  it('should add Combination Pliers to the cart', () => {

      cy.visit('https://practicesoftwaretesting.com/#/')
      cy.contains('Combination Pliers').click()

      cy.get('[data-test="add-to-cart"]').click()
      cy.get('#toast-container').contains('Product added to shopping cart.')
  })
})
```

Programski kod 6.3

6.3.4. Sortiranje proizvoda po imenu (A-Z)

Testni slučaj sortiranja proizvoda po imenu abecednim redoslijedom provjerava sortira li aplikacija ispravno proizvode, što je korisnicima ključno za lakši pronalazak željenih proizvoda (Programski kod 6.4).

```
describe('Sort items by name A-Z', () => {
  it('should sort items by name from A to Z', () => {

      cy.visit('https://practicesoftwaretesting.com/#/')
      cy.get('[data-test="sort"]').select('Name (A - Z)')
  })
})
```

Programski kod 6.4

6.3.5. Sortiranje proizvoda po cijeni (od niže prema višoj)

Testni slučaj sortiranja proizvoda po cijeni se odvija na isti način kao i slučaj iznad, samo što sada odabiremo drugi način sortiranja iz padajućeg izbornika (Programski kod 6.5).

```
describe('Sort items by price', () => {
  it('should sort items by price from low to high', () => {

      cy.visit('https://practicesoftwaretesting.com/#/')
      cy.get('[data-test="sort"]').select('Price (Low - High)')
  })
})
```

Programski kod 6.5

6.3.6. Filtriranje po kategoriji

Filtriranje po kategoriji je jednostavna funkcionalnost koja korisnicima omogućava da smanje izbor proizvoda na osnovu svojih preferencija i potreba. Ta funkcionalnost se provjerava klikom na checkbox sa željenim sadržajem koji se pretražuje. U ovom testnom slučaju dohvaća

se klasa checkbox uz pomoć metode *cy.get('.checkbox')* i izvršava se klik na kategorije 'Hammer' i 'Pliers' (Programski kod 6.6).

```
describe('Search by Category', () => {
  it('should search products within a specific category', () => {

    cy.visit('https://practicesoftwaretesting.com/#/');

    cy.get('.checkbox')
      .contains('Hammer').click()
    cy.get('.checkbox')
      .contains('Pliers').click()
  });
});
```

Programski kod 6.6

6.3.7. Pretraživanje proizvoda

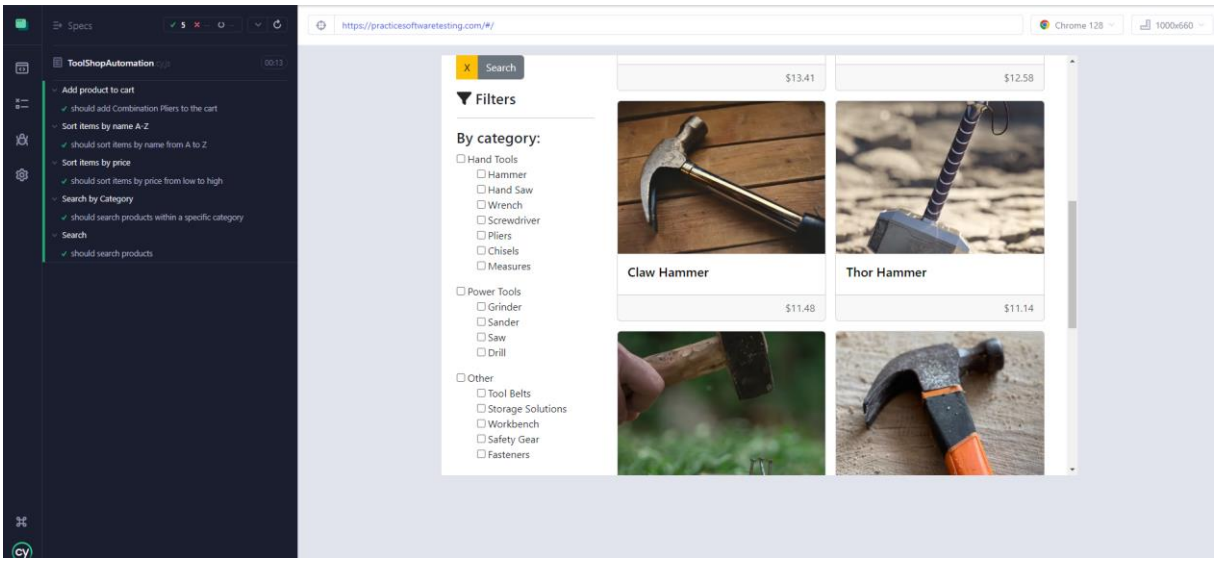
Programski kod 6.7 prikazuje testni slučaj koji provjerava funkcionalnost pretraživanja proizvoda pomoću ključne riječi. Prvo se dohvaća polje za unos ključne riječi metodom *cy.get('[data-test="search-query"]')*. Zatim se u to polje upisuje željena ključna riječ i nakon toga se simulira klik na gumb koji pokreće pretraživanje.

```
describe('Search', () => {
  it('should search products', () => {

    cy.visit('https://practicesoftwaretesting.com/#/');
    cy.get('[data-test="search-query"]').type('Hammer')

    cy.get('[data-test="search-submit"]').click()
  });
});
```

Programski kod 6.7

6.3.8. Rezultati testiranja u Cypressu

Na Slici 6.3 su prikazani rezultati testiranja unutar Cypress Test Runnera u stvarnom vremenu. Izvršeno je testiranje datoteke ToolShopAutomation.cy.js koja sadrži ranije opisane testove. Prvo je izvršena pripremna funkcija (*engl. hook*) 'before()' unutar koje je izvršena registracija i prijava. Nakon uspješne registracije i prijave izvršeni su testovi sortiranja, filtriranja, pretraživanja i dodavanja proizvoda u košaricu. Svi testovi su uspješno izvršeni.



Slika 6.3 Rezultati testiranja u Cypressu

6.4. Automatizirano testiranje web aplikacije u Playwrightu

Nakon prikazanih testnih slučajeva u Cypressu, u ovom potpoglavlju biti će prikazano kako ti isti slučajevi izgledaju kada se implementiraju koristeći Playwright. Playwright je alat za automatizaciju testiranja koji omogućava slične funkcionalnosti kao Cypress, ali sa svojim specifičnostima i prednostima. Predstavit će se odgovarajući Playwright programski kodovi za svaki testni slučaj, uz objašnjenje kako se metode u Playwrightu koriste za postizanje istih rezultata kao u Cypressu. Ovaj dio je uvod u usporedbu ova dva alata i uvid u fleksibilnost testiranja web aplikacija koristeći različite tehnologije.

6.4.1. Registracija korisnika

Ovaj testni slučaj automatizacije izrađen je pomoću Playwrighta i fokusira se na provjeru funkcionalnosti registracije korisnika na web stranici Tool Shop. Cilj ovog testa je osigurati da korisnik može uspješno završiti proces registracije.

Kao i kod automatizacije u Cypressu testni scenarij započinje posjetom web stranici Tool Shop, samo što sada koristimo metodu *page.goto('https://practicesoftwaretesting.com/#/').* Zatim se pronalazi gumb „Sign in“ korištenjem iste metode kao u Cypressu *page.click()* kako bi se simulirao klik i preusmjerilo korisnika na stranicu za prijavu. Zatim je potrebno kliknuti na gumb „Register your account“, čime se otvara stranica za registraciju. Playwright skripta tada automatski popunjava polja kao slično kao i Cypress samo što to Playwright radi uz pomoć metode *page.fill()*. Nakon unosa podataka, klikne se na gumb za registraciju. Na kraju, test

provjerava je li korisnik uspješno preusmjeren na stranicu za prijavu, što potvrđuje uspješnu registraciju (Programski kod 6.8).

```
test.describe('User Registration', () => {
  test('should register a new user successfully', async ({ page }) => {

    await page.goto('https://practicesoftwaretesting.com/#/');

    await page.click('text=Sign in');
    await page.click('text=Register your account');

    await page.fill('[data-test="first-name"]', 'Iva');
    await page.fill('[data-test="last-name"]', 'Hodić');
    await page.fill('[data-test="dob"]', '2000-01-01');
    await page.fill('[data-test="address"]', 'Zagrebačka 1');
    await page.fill('[data-test="postcode"]', '10000');
    await page.fill('[data-test="city"]', 'Zagreb');
    await page.fill('[data-test="state"]', 'Zagreb');
    await page.selectOption('[data-test="country"]', 'Croatia');
    await page.fill('[data-test="phone"]', '0911234567');
    await page.fill('[data-test="email"]', 'ivahodic@dispostable.com');
    await page.fill('[data-test="password"]', 'IvaIva123!');

    await page.click('text=Register');
    await page.waitForTimeout(2000);

    await page.fill('[data-test="email"]', 'ivahodic@dispostable.com');
    await page.fill('[data-test="password"]', 'IvaIva123!');
    await page.click('text=Login');
  });
});
```

Programski kod 6.8

6.4.2. Prijava korisnika

Kao i kod Cypressa, za uspješnu provedbu ovog testa potrebno je unaprijed definirati testne korisničke podatke koji su valjani i registrirani u sustavu. Cilj ovog testa je osigurati da sustav ispravno obrađuje korisnikove podatke za prijavu i omogućuje mu pristup njegovom profilu.

Testni scenarij započinje otvaranjem web stranice Tool Shop putem metode *page.goto(),* čime se simulira dolazak korisnika na početnu stranicu aplikacije. Playwright omogućuje identifikaciju elemenata na stranici korištenjem različitih metoda kao što su *page.click()* za klikanje na gumbe i *page.fill()* za unos podataka u tekstualna polja. U ovom testu, gumb „Sign

in“ se prepoznaje pomoću tekstualnog sadržaja, a potom se simulira klik na njega kako bi korisnik bio preusmjeren na stranicu za prijavu.

Na stranici za prijavu, test popunjava polja za email adresu i lozinku koristeći unaprijed definirane validne podatke, što se realizira pomoću funkcije *page.fill()*. Nakon popunjavanja forme, klikom na gumb „Login“ provjerava se ispravnost prijave korisnika. Validacija uspješne prijave provodi se provjerom prisutnosti elemenata specifičnih za korisnički profil, poput naslova „My account“, što se obavlja metodom *expect()* koja provjerava očekivani tekst na stranici (Programski kod 6.9).

```
test.describe('Sign in', () => {
  test('should successfully log in the user with valid credentials', async ({
page }) => {

    await page.goto('https://practicesoftwaretesting.com/#/');

    await page.click('text=Sign in');

    await page.fill('[data-test="email"]', 'ivahodic@dispostable.com');
    await page.fill('[data-test="password"]', 'IvaIva123!');
    await page.locator([data-test="login-submit"]). click();

    await expect(page.locator('h1')).toHaveText('My account');
  });
});
```

Programski kod 6.9

6.4.3. Dodavanje proizvoda u košaricu

Kao i u Cypressu, test u Playwrightu započinje posjetom stranici Tool Shop pomoću metode *page.goto()*. Nakon toga, proizvod se odabire i klikne pomoću *page.click('text= Combination Pliers')*, gdje se koristi selektor koji prepoznaje element na osnovu tekstualnog sadržaja. Za dodavanje proizvoda u košaricu koristi se *page.click('[data-test="add-to-cart"]')*, što je jednako metodi *cy.get('[data-test="add-to-cart"]').click()* u Cypressu. Na kraju, provjera je li proizvod uspješno dodan u košaricu radi se pomoću *expect(page.locator(' [data-test="add-to-cart"]).toBeVisible()*. Ova metoda locira element i provjerava da njegov tekst odgovara očekivanom, što je slično metodi *cy.get('#toast-container').contains()* u Cypressu (Programski kod 6.10).

```
test.describe('Add product to cart', () => {
  test('should add Combination Pliers to the cart', async ({ page }) => {

    await page.goto('https://practicesoftwaretesting.com/#/');

    await page.click('text=Combination Pliers');
    await page.click('[data-test="add-to-cart"]');
    await expect(page.locator('[data-test="add-to-cart"]')).toBeVisible();

  });
});
```

Programski kod 6.10

6.4.4. Sortiranje proizvoda po imenu (A-Z)

Za sortiranje proizvoda po imenu koristi se metoda *page.selectOption()*. U Playwrightu, omogućuje izbor opcije u padajućem izborniku na temelju tekstualnog opisa, u ovom slučaju 'Name (A-Z)', što je jednako metodi *cy.get('[datatest="sort"]') .select('Name (A - Z)')* u Cypressu (Programski kod 6.11).

```
test.describe('Sort items by name A-Z', () => {
  test('should sort items by name from A to Z', async ({ page }) => {

    await page.goto('https://practicesoftwaretesting.com/#/');
    await page.selectOption('[data-test="sort"]', { label: 'Name (A - Z)' });
  });
});
```

Programski kod 6.11

6.4.5. Sortiranje proizvoda po cijeni (od niže prema višoj)

Sortiranje proizvoda po cijeni izvodi se na isti način kao i prethodni slučaj, samo što se sada odabire druga opcija iz padajućeg izbornika (Programski kod 6.12).

```
test.describe('Sort items by price', () => {
  test('should sort items by price from low to high', async ({ page }) => {

    await page.goto('https://practicesoftwaretesting.com/#/');
    await page.selectOption('[datatest="sort"]', { label: 'Price (Low - High)'
 });
  });
});
```

Programski kod 6.12

6.4.6. Filtriranje po kategoriji

Za odabir kategorija 'Hammer' i 'Pliers' Playwright omogućuje prepoznavanje elemenata na temelju kombinacije klasa i tekstualnog sadržaja pomoću *:has-text()* *pseudoselktora*, što olakšava selekciju i interakciju s elementima koji imaju određeni tekst. Pseudoselektor je posebna vrsta selektora u CSSu koji omogućuje odabir elemenata na temelju njihovog stanja, položaja u dokumentu, ili specifičnih karakteristika koje se ne mogu jednostavno selektirati običnim selektorima [29]. Metoda *click()* uz *:has-text()* pseudoselektor u Playwrightu pruža preciznu kontrolu nad interakcijama s elementima, slično kao *cy.contains().click()* u Cypressu. Razlika je u boljoj podršci za kompleksnije selektore u Playwrightu (Programski kod 6.13).

```
test.describe('Search by Category', () => {
  test('should search products within a specific category', async ({ page }) =
> {
    await page.goto('https://practicesoftwaretesting.com/#/');

    await page.click('.checkbox:has-text("Hammer")');
    await page.click('.checkbox:has-text("Pliers")');
  });
});
```

Programski kod 6.13

6.4.7. Pretraživanje proizvoda

Nakon posjete stranici, koristi se *page.fill()* za unos ključne riječi u polje za pretragu i *page.click()* za pokretanje pretrage. Metoda *fill()* omogućuje unos teksta u input polja, dok click pokreće akciju, što je slično kombinaciji *cy.get().type()* i *cy.get().click()* u Cypressu (Programski kod 6.14).

```
test.describe('Search', () => {
  test('should search products', async ({ page }) => {

    await page.goto('https://practicesoftwaretesting.com/#/');

    await page.fill('[data-test="search-query"]', 'Hammer');
    await page.click('[data-test="search-submit"]');
  });
});
```

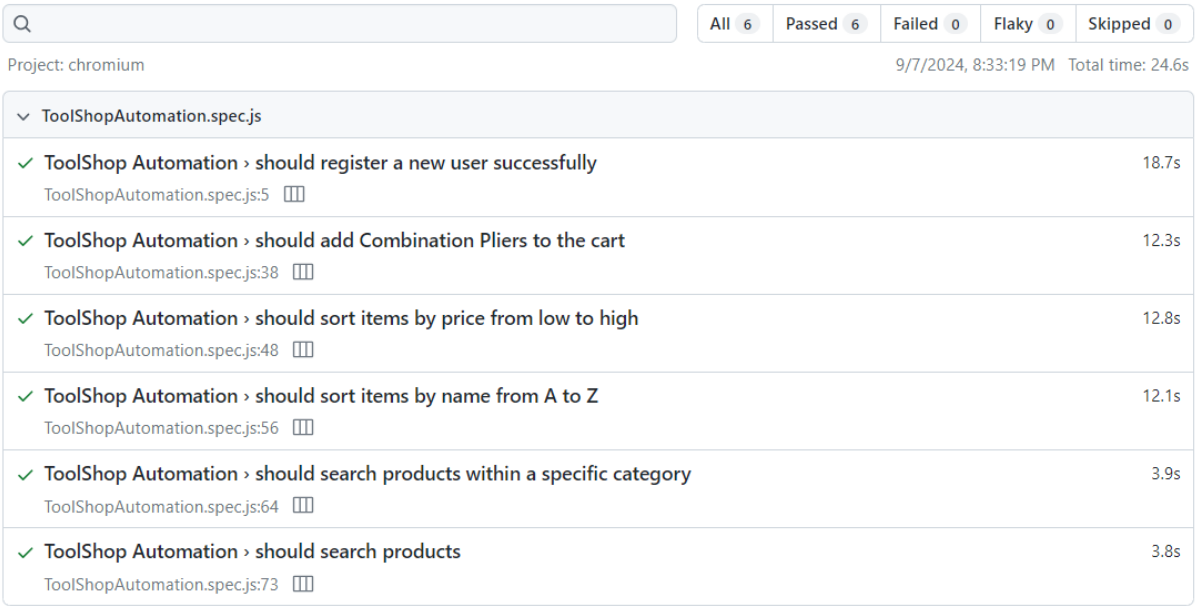
Programski kod 6.14

6.4.8. Rezultati testiranja u Playwrightu

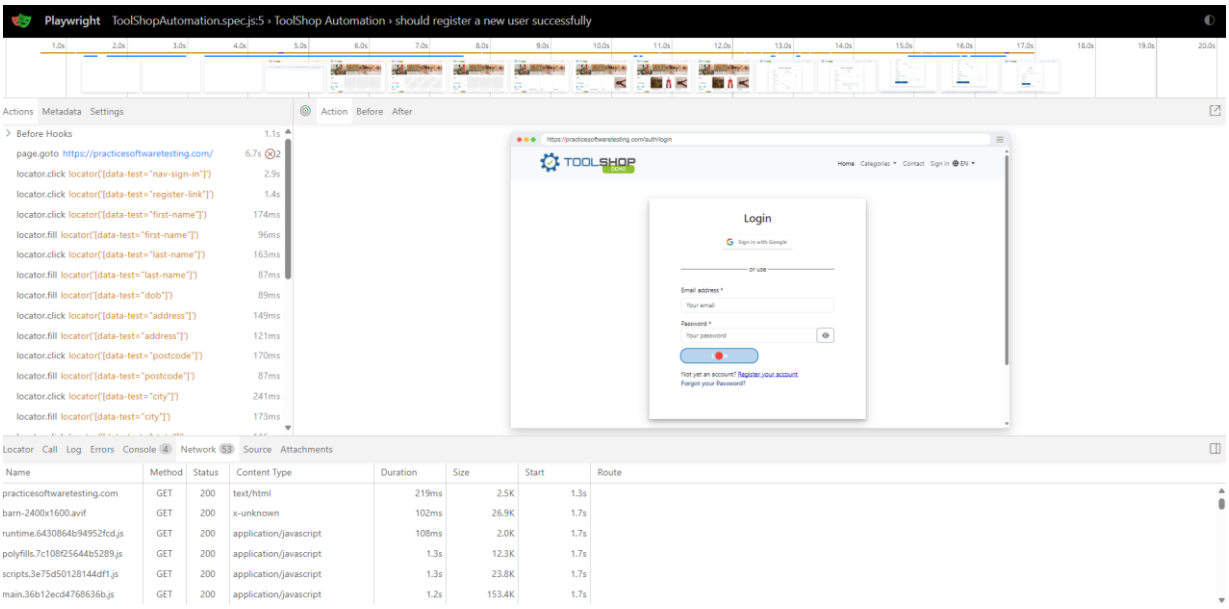
Na slikama ispod prikazani su rezultati testiranja pomoću Playwrighta. Na Slici 6.4 vidljiv je Playwright izvještaj s ukupno šest uspješno izvršenih testova unutar datoteke pod nazivom ToolShopAutomation.spec.js.. Svaki test prikazuje koliko je sekundi bilo potrebno za njegovo izvršavanje; na primjer, registracija novog korisnika trajala je 18,7 sekundi, dok je pretraga proizvoda unutar specifične kategorije trajala 3,9 sekundi. Ovaj prikaz omogućava jasan uvid u to koji su dijelovi aplikacije funkcionalni bez grešaka.

Na Slici 6.5 prikazan je detaljan zapis o testu koji provjerava uspješnost registracije u Trace Vieweru. Gornji dio slike prikazuje vremensku liniju koja dokumentira svaki korak tijekom testiranja, uključujući snimke zaslona koje prikazuju napredak kroz različite faze aplikacije, od početne login stranice do uspješne registracije. Lijevi dio slike sadrži detaljan zapis svakog koraka koji je test skripta izvela, poput otvaranja URL-a, unosa podataka u različita polja (ime, prezime, datum rođenja, adresa, poštanski broj, grad, država, telefon, e-mail) te interakcije s različitim elementima korisničkog sučelja.

Svi testovi su uspješno izvršeni.



Slika 6.4 Testni izvještaj



Slika 6.5 Detalji testa 'Registracija korisnika'

7. USPOREDBA ALATA CYPRESS I PLAYWRIGHT

Oba alata, i Cypress i Playwright su moćne moderne biblioteke za automatizirano testiranje web aplikacija. Svaki od njih ima jedinstvene karakteristika i prednosti. U ovom poglavlju će prema [30], [31] biti izdvojeni ključni elementi oba alata kako bi se bolje shvatio njihov potencijal, prednosti i ograničenja koji će pomoći u odabiru prikladnijeg alata za određen posao.

Postavljanje Cypressa je jednostavno i intuitivan je za korištenje dok postavljanje Playwrighta može biti malo složenije, ali nudi fleksibilnost u odabiru programskih jezika. Playwright pruža više mogućnosti što znači da je i više vremena potrebno da ga se nauči koristiti.

Playwright ima snažnije mogućnosti testiranja performansi, posebno pri scenarijima koji uključuju više stranica i preglednika. S druge strane, Cypress je poznatiji po svojoj brzini testiranja jednostavnijih aplikacija.

Za razliku od Playwrighta koji pokreće preglednik izvan aplikacije i podržava više otvorenih kartica te testiranje u različitim preglednicima unutar iste sesije, Cypress se integrira direktno u preglednik što pojednostavljuje rad s elementima i ubrzava testove, ali znači da nije potpuno odvojen od njega i uz to ne podržava više kartica jer koristi samo jedan preglednik. Tablica 7.1 prikazuje ključne razlike između Cypressa i Playwrighta.

Tablica 7.1 Ključne razlike Cypressa i Playwrighta

Kriterij	Cypress	Playwright
Podržani programski jezici	JavaScript	JavaScript, Java, Python, .NET #C
Podržane biblioteke	Mocha	Mocha, Jest, Jasmine
Podržani operacijski sustavi	Windows, Linux, macOS 10.9+	Windows, Linux, macOS
Podržani preglednici	Chrome, Firefox, and Edge	Chromium, Firefox, WebKit
Softver otvorenog koda	Da	Da
Arhitektura	Izvršava testove direktno unutar preglednika	Izvršava testove bez grafičkog korisničkog sučelja i koristi arhitekturu temeljenu na događajima
Podrška	Snažna	Ograničena
Podržava API testiranje	Ograničeno	Snažno

8. ZAKLJUČAK

U ovom radu su opisani važni pojmovi koji se odnose na testiranje softvera, s posebnim fokusom na automatsko testiranje. Testiranje ima ključnu ulogu u procesu razvoja softvera, jer omogućuje identifikaciju grešaka i osiguranje kvalitete softverskih proizvoda. Testiranje osigurava stabilnost, pouzdanost i funkcionalnost proizvoda bez obzira na to koliko je aplikacija velika ili složena.

Različiti tipovi testiranja softvera uključuju funkcionalno testiranje, koje se fokusira na provjeru ispravnosti aplikacije prema zahtjevima, kao i nefunkcionalno testiranje koje obuhvaća performanse, sigurnost i upotrebljivost. Dalje, provodi se testiranje na različitim razinama kao što su jedinice, integracija, sistem i prihvatljivost. Svaka razina testiranja ima svoju svrhu u prepoznavanju raznih grešaka i osiguravanju ispravnog funkcioniranja sustava.

Automatizirano testiranje ima mnogo prednosti u usporedbi sa ručnim testiranjem. To posebno dolazi do izražaja u slučajevima gdje se softver često mijenja, jer omogućava brže i efikasnije obavljanje velike količine testova. Automatizacija ima brojne prednosti kao što su smanjenje rizika od ljudskih grešaka, mogućnost ponovnog korištenja testnih skripti i ubrzanje procesa otkrivanja grešaka u ranim fazama razvoja. Također, automatizirano testiranje ima mogućnost provođenja regresijskog testiranja kako bi se osiguralo da nove promjene u kodu ne dovode do pojave grešaka u postojećim funkcionalnostima.

Rad je detaljno prikazao dva alata za automatizirano testiranje – Cypress i Playwright. Cypress je poznat zbog svoje jednostavnosti i brzine te se preporučuje za aplikacije koje koriste JavaScript. S druge strane, Playwright se izdvaja po svojoj sposobnosti testiranja složenih aplikacija koje uključuju više stranica, domena i preglednika. Svojim podržanim testiranjem u različitim preglednicima, kao i mogućnošću paralelnog izvršavanja testova na više uređaja istovremeno, ovaj alat je savršen za projekte koji zahtijevaju testiranje složenih scenarija.

Zaključno, izbor alata za automatsko testiranje ovisi o specifičnim zahtjevima projekta. Cypress je odličan izbor za jednostavne aplikacije i brze rezultate, dok Playwright pruža veće mogućnosti za složenija testiranje i scenarije koji se rade na više preglednika. Automatizirano testiranje ima mnogo prednosti, kao što su smanjenje vremena potrebnog za testiranje, povećanje pouzdanosti i osiguranje stabilnosti aplikacija tokom svih faza razvoja. U današnjem brzom razvoju, automatsko testiranje je neizostavna metoda za osiguranje kvalitete softverskih rješenja.

LITERATURA

[1] Geeks for geeks [online], dostupno na: https://www.geeksforgeeks.org/software-and-its-types/ [22. lipnja 2024.]

[2] B. Rotimi-Williams, J. T. Soncy : Software bugs: detection, analysis and fixing, Elsevier BV, Pretoria, South Africa, 2023.

[3] Softesting: Automation Testing Vs. Manual Testing: What's the Difference? [online], dostupno na: https://softesting.com/eng/automation-testing-vs-manual-testing-whats-the-difference/ [22. lipnja 2024.]

[4] G. J. Myers, C. Sandler, T. Badgett: The Art of Software Testing, John Wiley & Sons, Inc, Canada, 1979.

[5] M. Fewster, D. Graham: Software Test Automation, Addison-Wesley, Harlow, Essex, U.K.,

1999.

[6] Geeks for geeks, Differences between Functional and Non-functional Testing [online], dostupno na: https://www.geeksforgeeks.org/differences-between-functional-and-non-functional-testing/ [23. lipnja 2024.]

[7] ISTQB Foundation, Testing of Function (Functional Testing) [online], dostupno na: https://istqbfoundation.wordpress.com/2017/09/18/testing-of-function-functional-testing/ [24. lipnja 2024.]

[8] ISTQB Foundation, Non-functional Testing [online], dostupno na: https://istqbfoundation.wordpress.com/2017/09/18/non-functional-testing/ [24. lipnja 2024.]

[9] D. Graham, E. van Veenendaal, I. Evans, R. Black: Foundations of Software Testing, Thomson, Andover, UK., 2008.

[10] R. Black: Managing the Testing Process; Practical Tools and Techniques for Managing Software and Hardware Testing, 3rd edition, Wiley Publishing, Inc., New Jersey, 2009.

[11] Guru99: What is security testing? [online], dostupno na: https://www.guru99.com/hr/what-is-security-testing.html [24. lipnja 2024]

[12] Guru99: What is usability testing? [online], dostupno na: https://www.guru99.com/usability-testing-tutorial.html [24. lipnja 2024]

[13] Guru99: What is reliability testing? [online], dostupno na: https://www.guru99.com/hr/reliability-testing.html [24. lipnja 2024]

[14] R. Pham, 2.2. Test Levels: ISTQB Foundation, [online], dostupno na: https://istqbfoundation.wordpress.com/category/chapter-2-testing-throughout-the-software-lifecycle/section-2-2-test-levels-k2/ [24. lipnja 2024.]

[15] ISTQB Software Testing, ISTQB – Test levels [online], dostupno na: https://www.getsoftwareservice.com/481/ [24. lipnja 2024.]

[16] ISTQB Foundation [online], dostupno na: http://istqbfoundation.wikidot.com/4 [25. lipnja 2024.]

[17] ISTQB Glossary, Standard Glossary of Terms Used in Software Testing [online], dostupno na: https://glossary.istqb.org/en_US/home [25. lipnja 2024.]

[18] Linda G. Hayes: The Automated Testing Handbook, Software Testing Inst; 2nd edition, 2004.

[19] White test lab: How to overcome the disadvantages and limitations of automation testing [online], dostupno na: https://white-test.com/for-qa/useful-articles-for-qa/disadvantages-of-automation-testing/ [27. lipnja 2024.]

[20] Katalon Docs [online], dostupno na: https://docs.katalon.com/ [29.lipnja 2024.]

[21] Selenium [online], dostupno na: https://www.selenium.dev/ [29.lipnja 2024.]

[22] Cypress [online], dostupno na: https://www.cypress.io/ [29.lipnja 2024.]

[23] Playwright [online], dostupno na: https://playwright.dev/ [29.lipnja 2024.]

[24] Cypress [online], dostupno na: https://docs.cypress.io/guides/core-concepts/testing-types [30. lipnja 2024]

[25] Tools QA [online], dostupno na: https://www.toolsqa.com/cypress-tutorial [30. lipnja 2024]

[26] Playwright [online], dostupno na: https://playwright.dev/docs/ [2. srpnja 2024.]

[27] Playwright [online], dostupno na: https://playwright.dev/docs/writing-tests [2. srpnja 2024.]

[28] Playwright [online], dostupno na: https://playwright.dev/docs/codegen-intro [2. srpnja 2024.]

[29] W3School [online], dostupno na: https://www.w3schools.com/css/css_pseudo_classes.asp [2. srpnja 2024.]

[30] Playwright [online], dostupno na: https://playwright.dev/docs/trace-viewer-intro [3. srpnja 2024.]

[31] Browser Stack [online], dostupno na: https://www.browserstack.com/guide/playwright-vs-cypress [3.srpnja 2024.]

[32] Lambda Test [online], dostupno na: https://www.lambdatest.com/blog/cypress-vs-playwright/ [3. srpnja 2024.]

SAŽETAK

U ovom radu istražena je važnost automatiziranog testiranja softvera te je napravljena usporedba dvaju alata za automatizirano testiranje, Cypress i Playwright. Glavni cilj rada je pružiti znanje za odabir najprikladnijeg alata za automatizaciju testiranja softverskih aplikacija, uzimajući u obzir potrebe projekta i specifikaciju aplikacije. Analizirani su ključni dijelovi testiranja softvera, uključujući funkcionalno i nefunkcionalno testiranje te različite razine testiranja poput jedinica, integracije, sistema i prihvatljivosti. Automatizirano testiranje prikazano je kao nužan proces u suvremenom razvoju softvera zbog njegove sposobnosti da ubrza testiranje, smanji mogućnost ljudskih pogrešaka te omogući ponovnu upotrebu testnih skripti. Rad je rezultirao detaljnom usporedbom alata Cypress i Playwright. Cypress je ocijenjen kao izvrstan izbor za jednostavne aplikacije koje zahtijevaju brzu povratnu informaciju, dok Playwright omogućuje naprednije testiranje složenijih aplikacija, s naglaskom na testiranje u više preglednika i domena. Rad nudi smjernice za odabir odgovarajućeg alata ovisno o zahtjevima projekta, naglašavajući važnost automatiziranog testiranja u održavanju kvalitete softverskih proizvoda.

Ključne riječi: automatizirano testiranje, Cypress, Playwright, softver, testiranje

ABSTRACT

This thesis delves into the analysis and comparison of software test automation tools, with a specific focus on Cypress and Playwright. The primary objective is to determine the most appropriate tool for test automation based on project requirements and software application characteristics. The study covers essential concepts in software testing, including functional and non-functional testing, as well as different testing levels (unit, integration, system, and acceptance). It also emphasizes the benefits of automated testing, such as faster test execution, reduced human errors, and the reusability of test scripts. The research culminated in a comprehensive comparison of the tools: Cypress is found to be suitable for quick and straightforward automation, while Playwright offers advanced capabilities for testing complex applications across multiple browsers and domains. In conclusion, the study provides an overview of the criteria for tool selection based on project needs, highlighting the vital role of automated testing in ensuring software quality.

Key words: automation testing, Cypress, Playwright, software, testing

ŽIVOTOPIS

Iva Hodić rođena je 16.rujna 2000. godine u Osijeku. Odrasla je u Osijeku gdje je pohađala Osnovnu školu Frana Krste Frankopana. U istom gradu upisuje i završava Prirodoslovno-matematičku gimnaziju. Svoje obrazovanje nastavlja na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer Računarstvo na preddiplomskom sveučilišnom studiju, koji završava tri godine kasnije. Odmah nakon završetka preddiplomskog studija, na istom fakultetu upisuje diplomski studij, smjer Informacijske i podatkovne znanosti, koji trenutno pohađa.

Iva Hodić

_____