Usporedba manualnog i automatiziranog testiranja web aplikacija na primjeru web stranice Ekonomskog fakulteta Sveučilišta u Zagrebu

Cerar, Tea

Sveučilište u Zagrebu

Ekonomski fakultet Zagreb

Elektroničko poslovanje u privatnom i javnom sektoru

USPOREDBA MANUALNOG I AUTOMATIZIRANOG TESTIRANJA WEB APLIKACIJA NA PRIMJERU WEB STRANICE EKONOMSKOG FAKULTETA SVEUČILIŠTA U ZAGREBU

Diplomski rad

Tea Cerar

Zagreb, rujan, 2024.

Sveučilište u Zagrebu

Ekonomski fakultet Zagreb

Elektroničko poslovanje u privatnom i javnom sektoru

USPOREDBA MANUALNOG I AUTOMATIZIRANOG TESTIRANJA WEB APLIKACIJA NA PRIMJERU WEB STRANICE EKONOMSKOG FAKULTETA SVEUČILIŠTA U ZAGREBU

COMPARISON OF MANUAL AND AUTOMATED TESTING OF WEB APPLICATIONS APPLIED TO THE FACULTY OF ECONOMICS AND BUSINESS UNIVERSITY OF ZAGREB WEBSITE

Diplomski rad

Tea Cerar, JMBAG: 0066243051

Mentor: Izv. prof. dr. sc. Nikola Vlahović

Zagreb, rujan, 2024.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Izjavljujem i svojim potpisom potvrđujem da je diplomski rad / prijava teme diplomskog rada isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu, a što pokazuju korištene bilješke i bibliografija.

Izjavljujem da nijedan dio rada / prijave teme nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog izvora te da nijedan dio rada / prijave teme ne krši bilo čija autorska prava.

Izjavljujem, također, da nijedan dio rada / prijave teme nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

_____

(vlastoručni potpis studenta)

_____

(mjesto i datum)

STATEMENT ON THE ACADEMIC INTEGRITY

I hereby declare and confirm by my signature that the final thesis is the sole result of my own work based on my research and relies on the published literature, as shown in the listed notes and bibliography.

I declare that no part of the thesis has been written in an unauthorized manner, i.e., it is not transcribed from the non-cited work, and that no part of the thesis infringes any of the copyrights.

I also declare that no part of the thesis has been used for any other work in any other higher education, scientific or educational institution.

_____

(personal signature of the student)

Sažetak

Rad analizira proces i vrste testiranja softvera s naglaskom na web aplikacije, fokusirajući se na životni ciklus web aplikacija. Cilj je pružiti jasno razumijevanje testiranja softvera, naglasak je na važnosti rane detekcije grešaka. Razmatraju se razlike između manualnog i automatiziranog testiranja, a kao primjer su prikazani manualni i Cypress automatizirani testovi provedeni na web stranici Ekonomskog fakulteta Sveučilišta u Zagrebu. Dodatno, rad pruža analizu i usporedbu dostupnih alata za automatizaciju testova i preporuke za efikasnu praksu testiranja. Automatizacija može donijeti brojne prednosti, poput ubrzanja isporuke softvera i smanjenja troškova, no zahtijeva kontinuirano održavanje. Preporuke za efikasnu praksu automatizacije testiranja uključuju korištenje kriterija pokrivenosti, te korištenje stabilnih i dosljednih selektora, izolacija testova, te pažljivo upravljanje vanjskim uslugama.

Ključne riječi: testiranje softvera, web aplikacije, automatizirano testiranje, manualno testiranje, životni ciklus softvera

Summary

The paper analyses the process and types of software testing, focusing on web applications and their lifecycle. The goal is to give a clear understanding of software testing while emphasizing the importance of early error detection. It focuses on the differences between manual and automated testing, using examples of manual and Cypress automated tests done on the website of the Faculty of Economics at the University of Zagreb. Additionally, the paper compares available test automation tools and recommends effective testing practices. Automation offers many benefits, like faster software delivery and reduced costs, but it requires ongoing maintenance. Recommendations for effective automation include using coverage criteria, stable and consistent selectors, test isolation, and careful management of external services.

Keywords: software testing, web applications, automated testing, manual testing, software life cycle.

Sadržaj:

1. UVOD

1.1. Predmet i ciljevi rada

Predmet rada je analiza procesa i vrsta testiranja softvera s naglaskom na web aplikacije. Rad obuhvaća analizu životnog ciklusa web aplikacija, te usporedbu alata za automatizaciju testova, s posebnim fokusom na praktičnu primjenu na primjeru web stranice Ekonomskog fakulteta Sveučilišta u Zagrebu. Cilj rada je pružiti jasno razumijevanje testiranja softvera, s posebnim naglaskom na važnost rane detekcije grešaka u razvoju softvera. Glavni cilj je demonstrirati razlike između manualnog i automatiziranog testiranja kroz analizu njihove primjene na web stranici Ekonomskog fakulteta. Dodatni ciljevi uključuju analizu dostupnih alata za automatizaciju testova, te pružanje preporuka za efikasnu praksu testiranja.

1.2. Izvori podataka i metode prikupljanja

Za potrebe ovog rada korišteni su različiti izvori podataka, uključujući stručnu literaturu, e-knjige, te relevantne web stranice posvećene alatima za testiranje softvera. Također, pregledani su znanstveni članci, istraživački radovi i online izvori koji se bave metodologijama testiranja softvera, uključujući manualno i automatizirano testiranje. Metode prikupljanja podataka uključivale su detaljnu analizu i usporedbu informacija iz službenih izvora, te interpretaciju podataka dobivenih iz relevantnih stručnih publikacija i online izvora.

1.3. Sadržaj i struktura rada

Rad je strukturiran u tri ključna dijela, u prvom dijelu rada obrađuje se životni ciklus web aplikacija i osnovna pojmovna određenja testiranja softvera, uključujući manualno i automatizirano testiranje. Drugi dio rada ima fokus na analizi primjene automatiziranih testova u razvoju softvera. Detaljno su razmotreni alati za „end-to-end“ automatizirane testove, pregledani neki od najpopularnijih alata za automatizaciju testiranja web aplikacija, te uspoređene prednosti i nedostaci manualnog i automatiziranog testiranja. Treći dio rada sadrži komparativnu analizu manualnog i automatiziranog testiranja na primjeru web stranice Ekonomskog fakulteta Sveučilišta u Zagrebu. Prikazani su koraci pripreme testnog koncepta za manualna testiranja, selekcija testova za automatizaciju, konkretan proces automatizacije testova pomoću alata Cypress, te preporuke za efikasnu praksu automatizacije testiranja.

2. WEB APLIKACIJE I TESTIRANJE SOFTVERA

2.1. Web aplikacije i njihov životni ciklus

Web aplikacije su softveri koji se pokreću u web pregledniku, one omogućuju pristup složenim funkcionalnostima bez instaliranja softvera na uređaje poput računala. Prednost web aplikacija je njihova pristupačnost, dostupne su putem svih web preglednika na raznim uređajima. Također, prilikom razvoja web aplikacija, nije potrebno napraviti nekoliko različitih verzija aplikacije za više platformi. Ista verzija web aplikacije radi na svim preglednicima i uređajima. (What is a web application, 2024) Kod izrade svakog softvera, pa tako i web aplikacija, cilj je razviti softver koji je primjenjiv i učinkovit za namijenjenu svrhu, te promjenjiv, ali pouzdan, razumljiv i održiv, dok istovremeno zadovoljava potrebe korisnika uz zadane troškove i raspored. (Yin i Joang, 2023) Kako bi se to postiglo, razvoj svakog softvera kreće od planiranja, planiranje prati životni ciklus softvera koji se sastoji od međusobno povezanih koraka. U literaturi postoje različite definicije tih koraka, no sve su izvedene iz općeg životnog ciklusa razvoja računalnog programa i sastoje se od šest faza. (Jakupović i Šuman, 2020)

Prva faza je analiza problema u kojoj se precizno definira neki problem, te kako bi se isti riješio računalnim programom. Opisuje se ulaz, obrada, izlaz, te korisničko sučelje, (Jakupović i Šuman, 2020) drugim riječima, potrebno je detaljno proučiti zahtjeve i potrebe koje softver mora ispuniti. Cilj ove faze je definirati što točno softver treba raditi i osigurati da su svi dijelovi razvoja jasno definirani i dokumentirani prije nego što započne stvarni razvoj softvera. Potrebno je detaljno analizirati sustav u koji će softver biti ugrađen, identificirati njegove zahtjeve, funkcionalnosti i performanse, razviti osnovnu strukturu sustava, te napisati ključnu dokumentaciju, poput zahtjeva, planova za razvoja, te planova kvalitete i testiranja. (Yin i Joang, 2023) Testiranje koje se planira u ovoj fazi je testiranje prihvaćanja, testovi obuhvaćaju potrebe korisnika, te su osmišljeni kako bi se utvrdilo zadovoljava li gotov softver njihove potrebe. (Ammann i Offutt, 2017)

Slijedeća faza razvoja softvera je dizajn koji uglavnom uključuje dva koraka, okvirni dizajn i detaljni dizajn. (Yin i Joang, 2023) Glavni zadatak dizajna je razbiti veliki problem u manje probleme, uspostaviti cjelokupnu strukturu softvera i definirati sve korake koje softver treba izvesti. (Jakupović i Šuman, 2020) U fazi dizajna uspostavlja se odnos između funkcionalnih modula prema zahtjevima softvera, definira se sučelje svakog funkcionalnog modula i dizajnira se

baza podataka. (Yin i Joang, 2023) Nakon dizajna, slijedi faza kodiranja koja se također sastoji od dva koraka, kodiranje i jedinično testiranje. (Yin i Joang, 2023) Kodiranjem se dobiva izvorni računalni kod, ono se odnosi na izradu korisničkog sučelja, pisanje softverskog koda, te pisanje sve interne dokumentacije koja objašnjava kod i svrhu korištenih naredbi u kodu. (Jakupović i Šuman, 2020) Drugi korak je jedinično testiranje kodiranih modula. Jediničnim testiranjem se provjerava dizajn i implementacija svih softverskih jedinica. (Yin i Joang, 2023)

Četvrta faza je testiranje softvera, ono je način za osiguranje kvalitete i pouzdanosti računalnog softvera. (Yin i Joang, 2023) Testiranje softvera podrazumijeva provjeru softvera, pronalaženje pogrešaka i ispravljanje pogrešaka. Smatra se da je u softveru pronađena greška, ako se softver ne ponaša prema unaprijed definiranim zahtjevima. (Jakupović i Šuman, 2020) Cilj softverskog testiranja je otkriti greške softvera u što ranijoj fazi njegovog životnog ciklusa jer se trošak pronalaženja i uklanjanja greške povećava svaki put kad se greška pronađe u kasnijoj fazi razvoja softvera. (Forgacs i Kovacs, 2024) Fazu testiranja je potrebno ponavljati sve dok se ne isprave sve pronađene greške (Jakupović i Šuman, 2020), također kada se proizvod mijenja ili nadograđuje, što može značajno utjecati na ukupne troškove. (Pezze, 2008)

Sljedeća faza je dokumentiranje softvera, međutim, i u ovoj fazi treba nastaviti s testiranjem kako bi se osiguralo da računalni program radi ispravno i da zadovoljava sve postavljene zahtjeve. Faza dokumentiranja se sastoji od pregleda već postojeće interne dokumentacije, eventualne izmjene iste, te izrade eksterne dokumentacije za korisnike. (Jakupović i Šuman, 2020) Nakon svih razvojnih aktivnosti, uključujući i izradu dokumentacije, svaki softver treba biti prihvaćen i isporučen. (Yin i Joang, 2023) Svi dokumenti koji su izrađeni tijekom životnog ciklusa softvera čine njegovu dokumentaciju. Interna dokumentacije služi kako bi olakšala programerima shvaćanje vlastitoga ili tuđega izvornoga programskog koda, dok je eksterna dokumentacija namijenjena krajnjim korisnicima, te predstavlja upute o primjeni softvera. (Jakupović i Šuman, 2020)

Posljednja i najduža faza životnog ciklusa razvoja softvera je njegovo održavanje. (Jakupović i Šuman, 2020) Održavanje softvera odnosi se na sve aktivnosti koje se provode nakon isporuke računalnog programa krajnjim korisnicima. (Yin i Joang, 2023) Aktivnosti koje se provode su sva potrebna edukacija i podrška krajnjim korisnicima, ispravljanje uočenih grešaka, te izmjene kako bi se poboljšao ili prilagodio rad softvera u promjenjivom okruženju. Unatoč provedenom testiranju softvera, krajnji korisnici tijekom korištenja često otkrivaju nove greške. (Jakupović i

Šuman, 2020) Održavanje softvera obuhvaća nekoliko vrsta. Savršeno održavanje uključuje izmjene i proširenja funkcionalnosti radi poboljšanja performansi i prilagodbe promjenjivim potrebama korisnika. (Yin i Joang, 2023)

Promjene u svrhu prilagodbe promjenama u radnom okruženju ili potrebama korisnika dio su adaptivnog održavanja. S druge strane, korektivno održavanje odnosi se na ispravljanje grešaka koje nisu otkrivene tijekom testiranja. Na kraju, preventivno održavanje ima za cilj sprječavanje budućih problema. (Yin i Joang, 2023) U trenutku kada se pronađe greška u softveru ili se primijeti potreba za poboljšanjem, životni ciklus softvera ponovno počinje sa prvom fazom. Održavanje softvera se zaustavlja tek kada prestane potreba za njime ili zastari, u toj točki održavanja potrebno je pokrenuti životni ciklus razvoja novog softvera. (Jakupović i Šuman, 2020)

2.2. Pojmovno određenje testiranja softvera

Testiranje softvera obuhvaća aktivnosti tijekom cijelog životnog ciklusa softvera, od ranih faza razvoja sve do isporuke i naknadnog razvoja, uključujući provjeru ispravnosti, validaciju funkcionalnosti i osiguranje kvalitete u svim fazama. (Pezze, 2008) Testiranje softvera ovisi o modelu izrade softvera, te služi kao potpora razvoju softvera i razvojnom timu. (Frid i Jović, 2022) Ono se kontinuirano ponavlja tijekom cijelog životnog ciklusa razvoja softvera, nakon dizajna i izvršenja testova, rezultati se analiziraju i prijavljuju razvojnom timu koji ispravlja pronađene greške. (Forgacs i Kovacs, 2024) Nakon pronalaska greška, razvojni tim ih treba locirati i otkloniti, (Frid i Jović, 2022) a proces testiranja se ponavlja dok se ne zadovolje kriteriji softvera i softver ne postigne željenu razinu kvalitete. (Forgacs i Kovacs, 2024) Testiranje je proces koji započinje gotovo u isto vrijeme kao i sami projekt izrade softvera. Potrebno je provjeriti svaki dokument i aktivnost u procesu razvoja softvera. Točnije, jesu li prikupljeni svi zahtjevi, te jesu li dobro definirani i dokumentirani, provjera dizajna softvera sa njegovim zahtjevima, te testovi funkcionalnosti softvera. (Popović, 2019)

Radno mjesto tester tradicionalno se odnosi na osobu koja testira gotov proizvod na kraju razvojnog procesa. Međutim, novija literatura opisuje testiranje kao dio verifikacije i validacije koje počinju već u ranim fazama razvoja softvera kako bi se održala kvaliteta softverskog proizvoda. U trenutku

kada se odluči izraditi softverski proizvod, trebalo bi započeti proces verifikacije i validacije. (Pezze, 2008) Provjera ispravnosti naziva se još i verifikacija, to je tehnička aktivnost gdje se utvrđuje jesu li proizvodi jedne faze ispunili zahtjeve postavljene tijekom prethodne faze. (Ammann i Offutt, 2017) Drugim riječima provjerava se usklađenosti između dva opisa, opis gotovog softvera i opis zahtjeva za taj softver, dok validacija uspoređuje opis zahtjeva, dizajna ili gotovog softvera sa stvarnim potrebama korisnika. (Pezze, 2008) Provjera validacije je znači, proces ocjenjivanja softvera kako bi se osiguralo da zadovoljava predviđene zahtjeve. (Ammann i Offutt, 2017) Specifikacija opisuje kako riješiti problem, ali to rješenje ne mora postići svoje ciljeve. S druge strane, sustav koji ispunjava specifikacije je pouzdan, radi toga je bitno napraviti provjeru specifikacija, ali i osigurava li potrebe korisnika. Kod validacije postoji mogućnost nesporazuma i neslaganja jer uvijek uključuje ljudsku prosudbu, dok bi verifikacija zahtjeva trebala biti dovoljno precizna i jednoznačna da ne može biti neslaganja. (Pezze, 2008)

Testiranje računalnih programa je vrlo promjenjivo i složeno, napredne tehnologije za razvoj mogu smanjiti učestalost nekih vrsta pogrešaka, međutim nije moguće u potpunosti eliminirati softverske greške. Programske greške su vrlo raznolike zbog čega je izazovno odabrati pravi spoj tehnika testiranja kako bi se postigla potrebna razina kvalitete uz ograničene troškove. Literatura sugerira da verifikacija softvera često premašuju polovicu ukupnih troškova razvoja i održavanja softvera, ali da je vjerojatnije da će softver koji je prošao temeljito testiranje biti pouzdaniji od softvera koji je bio samo površinski testiran. (Pezze, 2008) Osim provjere specifikacija softvera, testiranje također uključuje planiranje testiranja, analiziranje, dizajniranje i implementiranje testova, procjenu vremena testiranja, izvještavanje o napretku testiranja i rezultatima, te prijavu pronađenih grešaka razvojnom timu.

Planiranje testiranja je proces u kojem se definiraju ciljevi testiranja, testni predmeti, zadaci testiranja, pristup testiranju, što će se testirati, ljudski i drugi resursi, okruženje u kojem će se testirati, te kriteriji ulaza i izlaza koji će se koristiti u testovima. (Forgacs i Kovacs, 2024) Osim planiranja, bitan je i proces upravljanja projektom, odnosno, njegova strategija. Upravljanje projektom obuhvaća praćenje napretka aktivnosti, procjenu vremenskih rokova i budžeta, te praćenje potencijalnih rizika. (Popović, 2019) Projektna strategija testiranja regulira dizajn testova i način provođenja testiranja, strategija može biti proaktivna i reaktivna. Kod proaktivne strategije testiranje započinje što ranije u procesu razvoja. Cilj je otkriti i ispraviti greške prije nego što

proizvod bude izrađen.. Na taj se način smanjuje rizik od pojave većih problema kasnije u projektu. S druge strane, kod reaktivne strategije testiranje se odgađa do završetka razvoja računalnog programa. Tek nakon što je razvoj potpuno dovršen, pristupa se ispitivanju proizvoda kako bi se identificirale eventualne greške ili nedostaci. (Forgacs i Kovacs, 2024)

Analiza testiranja sastoji se od tri glavna koraka koja su ključna za osiguranje kvalitete softvera. Prvi korak uključuje temeljitu analizu svih relevantnih dokumenata. To podrazumijeva detaljno proučavanje poslovnih zahtjeva, sistemskih zahtjeva, specifikacija funkcionalnog dizajna, tehničkih specifikacija, korisničkih priručnika, te izvornog koda. Ovaj korak omogućuje razumijevanje kako sustav treba funkcionirati i identificira kritične točke koje treba testirati. Drugi korak usmjeren je na identificiranje testnih objekata, što obuhvaća značajke sustava i različite scenarije korištenja. U ovoj fazi definiraju se testni uvjeti, koji jasno određuju što točno treba biti testirano. Na taj način osigurava se da su obuhvaćene sve ključne funkcionalnosti sustava, kao i potencijalni slučajevi koje bi korisnici mogli susresti. (Forgacs i Kovacs, 2024).

Treći korak odnosi se na analizu rizika, njezina uloga je posložiti prioritete testiranja. Za svaki testni objekt procjenjuju se atributi rizika, uključujući važnost ispravnog funkcioniranja softvera i vjerojatnost pojave problema. Ova analiza omogućuje donošenje odluka o tome koji dijelovi sustava zahtijevaju više pažnje tijekom testiranja. Cilj cijele analize testiranja je predviđanje potencijalnih problema koji bi se mogli pojaviti u budućnosti, kako bi se minimizirali kvarovi i osigurao stabilan i pouzdan softver. (Forgacs i Kovacs, 2024).

Temeljni izvor za dizajniranje testova su funkcionalne specifikacije softvera koje opisuju željeno ponašanje softvera. (Pezze, 2008) Kod dizajna testova potrebno je provesti analizu rizika i složenosti za svaki softverski element, na kraju testiranja, potrebno je prikupiti sve troškove testiranja i ispravljanja grešaka kako bi se definirale i primijenile različite strategije za optimizaciju troškova na temelju tih podataka. (Forgacs i Kovacs, 2024). Ako se radi o automatiziranim testovima potrebno ih je implementirati u dogovoru sa razvojnim timom. U trenutku pronalaska greške u softveru, iste je potrebno prijaviti razvojnom timu. Razvojni tim ispravlja greške nakon čega testeri ponavljaju proces testiranja i cijeli proces se ponavlja dok više nema pronađenih grešaka ili greške ne utječu na funkcionalnost softvera.

Kvalitetno testiran softver zahtijeva manje održavanja nakon izlaska na tržište, manje održavanja znače manji troškovi održavanja. Također, testiranje omogućuje poboljšanje kvalitete softvera, povećava zadovoljstvo korisnika te doprinosi stabilnom i uspješnom poslovanju tvrtke. Cilj procesa kvalitete je potpunost, pravovremenost i isplativost. Potpunost procesa kvalitete odnosi se na planiranje svih aktivnosti koje su važne za otkrivanje ključnih klasa grešaka u softveru. Pravovremenost upućuje na to da su greške otkrivene najranije moguće, dok isplativost znači da se odabiru aktivnosti obzirom na cijenu i učinkovitost, uzimajući u obzir pravovremenost i potpunost. (Pezze, 2008)

Trošak cijelog projekta razvoja softvera pod utjecajem je dva faktora, trošak dizajniranja i izvođenja testova, te trošak ispravljanja grešaka. Trošak testiranja raste kako se povećava broj pronađenih grešaka. U početku, trošak testiranja raste linearno s brojem testnih slučajeva, međutim, kako se testiranje nastavlja, povećanje troškova postaje nelinearno. Razlog za to je što postaje sve teže pronaći preostale greške, pa se moraju koristiti složenije metode testiranja. Složeniji ili rizičniji dijelovi koda zahtijevaju temeljitije testiranje kako bi se osigurala prihvatljiva kvaliteta, a viši troškovi testiranja su neizbježni za optimalne ukupne troškove. (Forgacs i Kovacs, 2024).

Postoje različite razine testiranja, koje prate specifičnu aktivnost razvoja softvera, prihvatno testiranje, sustavno testiranje, integracijsko, jedinično testiranje, te regresijsko testiranje. (Ammann i Offutt, 2017) Prihvatno testiranje procjenjuje softver u odnosu na zahtjeve i potrebe korisnika, usmjereno je na donošenje odluke o tome treba li proizvod biti objavljen krajnjim korisnicima. Ono može, i trebalo bi uključivati testiranje s korisnicima gdje u stvarnim uvjetima testiraju i identificiranju potencijalne probleme prije nego što softver postane dostupan široj javnosti. Prihvatno testiranje se izvodi kada je kod dostupan, međutim, aktivnosti testiranja počinju čim su dostupni zahtjevi na temelju kojih se pišu testni slučajevi. Rano dizajniranje testova olakšava otkriti greške u softverskim zahtjevima i rano ispravljanje prije prijenosa grešaka u kasnije faze razvoja. (Pezze, 2008)

Sustavno testiranje procjenjuje softver u odnosu na njegovu arhitekturu i ukupno ponašanje. U ovoj vrsti testiranja pretpostavlja se da dijelovi softvera pojedinačno rade i postavlja se pitanje radi li sustav kao cjelina, (Ammann i Offutt, 2017) može li se pravilno povezati s drugim softverom ili hardverom u sustavu. (Yin i Joang, 2023) Testiranje sustava obuhvaća cijeli sustav i provjerava zadovoljava li sustav svoje specifikacije, ali i potrebe korisnika. Prihvatni i sistemski testni

slučajevi trebali bi se generirati prije integracijskih i jediničnih testnih slučajeva, iako se izvode nakon njih. Sustavni testovi pomažu u otkrivanju grešaka koje nisu otkrivene tijekom jediničnog i integracijskog testiranja. (Pezze, 2008)

Integracijsko testiranje procjenjuje softver u odnosu na dizajn podsustava, (Ammann i Offutt, 2017) te postoje dvije metode integracijskog testiranja. Jedna metoda je testiranje na glavnom računalu, a druga na računalo za koji je softver namijenjen. (Yin i Joang, 2023) Cilj integracijskog testiranja je provjeriti da su moduli ispravno povezani i njihovu sposobnost da zajedno rade kako je opisano u zahtjevima softvera. Ova vrsta testiranja je bitna za otkrivanje problema koji se ne pojavljuje kada se svaki modul testira samostalno, ali se može pojaviti kada se moduli kombiniraju. (Pezze, 2008) Radi toga integracija i integracijsko testiranje započinje čim su potrebne komponente prošle jedinično testiranje. (Ammann i Offutt, 2017)

Jedinično ili modularno testiranje procjenjuje softver u odnosu na implementaciju i detaljni dizajn. (Ammann i Offutt, 2017) Jedinično testiranje je vrlo važno jer provjerava svaku jedinicu, odnosno modul prije nego se on integrira s drugim dijelovima sustava, ono provjerava ponaša li se pojedinačna jedinica prema specifikacijama ili očekivanjima. Jedinično testiranje povećava kontrolu i pomaže u identificiranju grešaka u ranoj fazi razvoja,(Pezze, 2008) tek nakon što jedinično testiranje uspješno završi prelazi se na integracijsko i sustavno testiranje. (Yin i Joang, 2023)

Regresijsko testiranje ima veliku važnost u testiranju već postojećeg softvera koji je promijenjen ili nadograđen novim funkcionalnostima, cilj mu je provjeriti rade li postojeće funkcionalnosti softvera kako su radile prije promjene. (Pezze, 2008) Radi toga je bitno nakon svake promjene softvera provesti regresijsko testiranje, (Ammann i Offutt, 2017) kako bi se osiguralo da nove promjene nisu negativno utjecale na postojeću funkcionalnost, potrebno je ponovno izvršiti sve prethodne testne slučajeve. Regresijsko testiranje uključuje i odabir testnih slučajeva za ponovnu izvedbu kako bi se optimiziralo vrijeme i troškovi testiranja. (Pezze, 2008) Regresijsko testiranje je vrlo dugotrajno, pogotovo za velike komponente ili sustave koji imaju velike skupove testova regresije. (Ammann i Offutt, 2017) Svaka vrsta testiranja može koristiti različite tehnike za osiguranje ispravnosti i pouzdanosti, (Ammann i Offutt, 2017) radi čega je bitno pravilno planirati i provoditi testiranje kako bi se osigurao visokokvalitetni softverski proizvod. (Pezze, 2008)

2.3. Manualno testiranje softvera

Manualno testiranje je svaka vrsta provjere softvera u kojem osoba koja testira softver izvodi provjeru ili neki testni slučaj ručno bez upotrebe bilo kakvih automatiziranih alata. Testiranje softvera je uvijek barem djelomično ručno jer neke značajke aplikacije nije moguće testirati automatizacijom. Kada programeri provjeravaju osnovne funkcionalnosti softvera u razvoju, izvode ručne, odnosno manualne testove jer je brže od izrade testnih slučajeva za jednostavne dijelove koda. Također, kada softver ima izrađeno korisničko sučelje, ono se testira ručno kako bi se softver promatrao i provjeravao iz perspektive korisnika u stvarnom životu. Kako testiranje korisničkog sučelja uključuje kvalitativne podatke i osobno mišljenje, manualno testiranje pruža bolje i točnije rezultate. (Zaptest, 2024)

Osim testiranja osnovnih funkcionalnosti i dizajna korisničkog sučelja za koje je potrebno osobno mišljenje manualno testiranje koristi se i za penetracijsko i eksplorativno testiranje. Penetracijsko testiranje simulira napad na softver izvana kako bi se otkrilo koliko je lako neovlaštenim osobama dobiti pristup sustavu koristeći metode koje nisu dopuštene. Bitno je izvoditi penetracijsko testiranje ručno jer automatizirani alati za testiranje softvera obično slijede unaprijed definirane korake i scenarije, a ručno penetracijsko testiranje omogućava testerima da koriste svoju kreativnost i iskustvo kako bi pronašli nove, neočekivane slabosti i potencijalne sigurnosne slabosti. (Zaptest, 2024)

Eksplorativno testiranje se izvodi samo jednom ili dvaput, njegov cilj je istražiti softver za sve neočekivane značajke ili greške. (Zaptest, 2024) Ova vrsta testiranja je polustrukturirana i temelji se na iskustvu testera i sposobnosti pogađanja greške, zato ga je potrebno izvršiti ručno.(Forgacs i Kovacs, 2024) Pisanje koda za automatizirane testove koji će se pokrenuti samo jednom ili dva puta je neisplativo i oduzima mnogo više vremena nego manualno testiranje. Manualne testove izvode zaposlenici, najčešće programeri i testeri. (Zaptest, 2024)

U životnom ciklusu softvera manualni testovi se koriste za ispitivanje raznih aspekata softvera. U fazi planiranja testiranja potrebno je napisati sve testne slučajeve koje manualni tester treba izvršiti. Druga faza je izvršavanje testova, drugim riječima, prolaženje kroz napisane testne slučajeve i

bilježenje svih informacija, prolazi li test ili odstupa od opisanog testnog slučaja, te na koji način odstupa od testnog slučaja. Treća faza je analiza i pisanje testnog izvještaja gdje se opisuju rezultati testova, te informiranje programera o rezultatima i potencijalnim prilagodbama za softver. Nakon treće faze planira se ponovno testiranje u kojem se testiraju najnovije funkcionalnosti softvera i pokušavaju se ponovno dobiti greške koje su bile prisutne u prošloj verziji. (Zaptest, 2024)

Ciklus ručnog testiranje pokušava kontinuirano unaprijediti testiranje i softver koji se testira. Ovisno o razini pristupa softveru koji testeri imaju postoje tri različite vrste manualnog testiranja. Prva vrsta je testiranje bijele kutije gdje testeri imaju uvid u izvorni kod i mogu vidjeti kako pojedinačni dio koda utječe na način na koji softver radi. (Zaptest, 2024) Također, testiranje bijele kutije zahtjeva uvid u dokumentaciju za dizajn softvera i određenu razinu tehničkog znanja, radi čega testiranje bijele kutije najčešće izvode programeri (Forgacs i Kovacs, 2024) u ranim fazama razvoja softvera kako bi mogli usporediti vlastiti kod sa testnim slučajevima i pronaći potencijalne greške. (Zaptest, 2024) Ove testove mogu izvršavati i testeri koji poznaju i razumiju strukturu koda programa, ali najčešće ih izvode programeri. (Forgacs i Kovacs, 2024)

Testiranje crne kutije je suprotno od testiranja bijele kutije, testeri nemaju pristup internoj dokumentaciji ni izvornom kodu, (Zaptest, 2024) i može se izvršiti bez programerskog znanja. (Forgacs i Kovacs, 2024) Ovo testiranje se koristi u kasnijim fazama razvojnog procesa za testove prihvaćanja gdje testeri imaju pristup samo vanjskoj dokumentaciji o softveru i zahtjevima, ali ne znaju kako je softver implementiran. (Ammann i Offutt, 2017) Testovi prihvaćanja zahtijevaju perspektivu krajnjeg korisnika koji nije bio uključen u razvojni proces softvera radi čega testiranje crne kutije savršeno odgovora ovoj vrsti testiranja. (Zaptest, 2024) Testeri provjeravaju prati li softver zahtjeve i ponaša li se ispravno. (Forgacs i Kovacs, 2024)

Osim testiranja bijele i crne kutije, postoji i testiranje sive kutije koje se koristi tijekom srednjih faza razvojnog procesa. Ono je kombinacija testiranja crne i bijele kutije gdje tester ima pristup dokumentaciji i izvornom kodu, ali ograničene informacije o rukovanju podacima. (Zaptest, 2024) Drugim riječima, tester razvija testove iz dizajnerskih elemenata (Ammann i Offutt, 2017) i pokušava razumjeti kako je zamišljeno da softver funkcionira na temelju izvornog koda, dokumentacije i klikanja po sučelju, time se pokušava osigurati da krajnji korisnik može razumjeti sustav. (Zaptest, 2024)

Manualni testovi u potpunosti ovise o osobi koja piše, prati testne slučajeve i bilježi sve informacije prikupljene tokom testiranja. Glavna razlika između manualnih i automatiziranih testova je oslanjanje manualnih testova na osobu i njene sposobnosti zapažanja i poznavanja softvera. S druge strane, nakon što testni inženjer napiše kod za automatizirane testove, računalni softver, odnosno alat za automatizirane testove je odgovoran za dovršavanje testnih slučajeva, te ovisno o platformi za automatizirane testove, softver piše i izvješća za korisnike. (Zaptest, 2024)

2.4. Automatizirano testiranje softvera

Automatizirano testiranje, isto kao i manualno ima cilj provjeriti izvršava li softver zadane radnje i usporediti očekivane rezultate sa stvarnim ishodom. Međutim, da bi se to postiglo koristi se alat za automatizirane testove. (Zaptest, 2024) Automatizirani testovi nastaju tako da se ručni testovi implementiraju u alat za automatizaciju u obliku koda. (Kaul, 2023) Oni se implementiraju kako bi ubrzali proces testiranja, najčešće za veliki broj testova sa ponavljajućim zadacima (Zaptest, 2024) i za obavljanje testnih zadataka koje je teško izvršiti ručno. (Kaul,2023)

Ručno stvaranje i izvršavanje velikih količina testova nije učinkovito i teško je pronaći pogrešan rezultat unutar velikog seta testova, radi toga automatizacija ponavljajućih testova poboljšava njihovu točnost. Suprotno tome, testovi za koje je potrebna prosudba i kreativno rješenje ostaju manualni. (Pezze, 2008) Ponavljajući testovi su regresijski testovi, oni se sa svakom novom verzijom povećavaju i ručno oduzimaju jako puno vremena. Radi toga bi testovi regresije trebali biti automatizirani, (Ammann i Offutt, 2017) to bi omogućilo testnom timu da ima dovoljno vremena za analizu kvalitete i testiranje testova koje nije moguće automatizirati i zahtijevaju manualno testiranje. Automatizirani testovi dobro su usklađeni sa projektima razvoja softvera koji zahtijevaju često testiranje istih dijelova koji su već prošli početno manualno testiranje. (Kaul,2023)

Većina okvira za automatizirano testiranje napravljena je za jedinične i integracijske testove (Ammann i Offutt, 2017), međutim, glavni fokus ovog rada biti će automatizirati “End to End” testovi. Okvir za testiranje pruža strukturu i alate koji omogućuju pisanje, izvršavanje i organiziranje testova softverskih komponenti. On sadrži alat, odnosno testni upravljač, koji provodi

testove na pojedinačnim dijelovima softvera, poput funkcija, metoda, klasa ili modula, kako bi provjerio njihovu ispravnost i funkcionalnost. Testni upravljač uspoređuje stvarne rezultate s očekivanim, i generira izvještaje o uspjehu ili neuspjehu testova. Oblik testnog upravljača od kojeg se razvio JUnit je metoda main() koja se koristi za jedinične testove u Javi i postao je temelj za slične okvire u drugim programskim jezicima. Većina okvira za automatizaciju testiranja podržava evaluaciju rezultata, dijeljenje podataka među testovima, organizaciju testova u skupove, i pokretanje testova iz različitih sučelja. (Ammann, 2017)

Ako se kontinuirano primjenjuju najbolje dostupne tehnologije za testnu automatizaciju koja odgovara pojedinoj organizaciji, ona može značajno poboljšati i osigurati kvalitetu softvera. (Pezze, 2008) Iako je automatizacija testiranja sličnija programiranju ona zahtjeva nove strategije testiranja, nove procese, radne tokove, nove alate i vještine od testera, ali i programera. Izazov prijelaza sa manualnog testiranja na automatizirano je trošak održavanja testova. Nakon promjena u kodu softvera, moguće je da će se promijeniti i lokator koji se koristi u automatiziranom testu. Drugim riječima, taj test više neće raditi ispravno dok se stari lokator ne zamjeni sa novim. (Ammann i Offutt, 2017) Radi toga programeri i testeri traže stabilne lokatore za GUI objekte kada pišu automatizirane testove. (Forgacs i Kovacs, 2024)

Svaka web aplikacija ima UI objekte, to su elementi koji čine korisničko sučelje web aplikacije, također, postoji i programsko sučelje DOM koje predstavlja strukturu HTML ili XML dokumenta kao hijerarhijsko stablo. Svaki UI objekt kojemu se dodijeli vrijednost ili provjera rezultat je potrebno pronaći unutar HTML ili XML dokumenta kako bi mu se dodijelila vrijednost ili provjerio njegov rezultat. Kao lokatori za GUI objekte mogu se koristiti XPath selektori ili CSS selektori. XPath je jezik koji se koristi za navigaciju kroz element i atribute u XML dokumentu, dok su CSS selektori način odabira elementa unutar HTML-a. CSS selektori se obično koriste u kombinaciji sa programskim jezikom JavaScript ili CSS-om za stiliziranje ili manipulaciju elemenata na web stranici. (Forgacs i Kovacs, 2024)

Pri pisanju automatiziranih testova, idealno bi bilo koristiti jedinstvene selektore za svaki UI objekt. Ako se za automatizirane testove koriste CSS selektori i kod web aplikacije se promjeni, kod za testove se također mora promijeniti inače test neće raditi ili će pogrešno raditi. Rješenje za ovaj i njemu slične probleme je suradnja između programera web aplikacije i testera koji piše automatizirane testove. Selektori mogu sadržavati nekoliko elemenata, dodavanjem jedinstvenog

selektora pomoću kojeg će test uvijek pronaći točan UI objekt. Točnije, programeri moraju dodati poseban atribut u svoj kod koji će testeri automatizacije selektirati UI objekte. Dodavanje jedinstvenog atributa programeru oduzima manje od pola minute, dok testeru pronalaženje odgovarajućeg selektora i ispravljanje promijenjenih selektora oduzima puno više vremena. (Forgacs i Kovacs, 2024)

Dodavanjem jedinstvenih lokatora poboljšava se kvaliteta automatiziranog testiranja web aplikacije i smanjuju se troškovi jer automatizirani testni slučajevi ostaju stabilni i testeri ih ne moraju ispravljati nakon svake manje promjene u aplikaciji. Također, to im omogućuje da imaju dovoljno vremena automatizirati novonastale funkcionalnosti i aplikacije. (Forgacs i Kovacs, 2024) Automatizaciju testiranja potrebno je prilagoditi organizaciji i uvoditi postepeno ovisno o zrelosti organizacije i troškovima. (Pezze, 2008)

3. ANALIZA PRIMJENE AUTOMATIZIRANIH TESTOVA U RAZVOJU SOFTVERA

3.1. Alati za „End to end“ automatizirane testove

End to end testiranje, odnosno e2e testiranje je vrsta testiranja koja u kojoj se obavljaju radnje koje bi krajnji korisnik aplikacije izvršio kako bi postigao željeni rezultat. (Gelfenbuim, 2022) To je tehnika koja testira aplikacije od web preglednika sve do pozadinskog dijela aplikacije, ideja e2e testova je osigurati da aplikacija funkcionira kao cjelina. U tome je razlika između e2e testova i integracijskih testova, integracijsko testiranje provjerava kako neki dijelovi sustava reagiraju na razne događaje, dok e2e testiranje testira aplikaciju kao cjelinu. Za sve automatizirane testove potrebni su alati koji omogućavaju automatizaciju cjelokupnog procesa testiranja softverske aplikacije. (Cypress, 2024)

Alati za e2e testove se koriste kako bi simulirali stvarne korisničke interakcije s aplikacijom (Cypress, 2024), provjeravajući cijeli tijek radnje od početka do kraja. (Skadorva, 2023) Pomoću alata za 2e2 testove se prolaze i provjeravaju razne funkcionalnosti aplikacije kako bi se osiguralo da svi dijelovi aplikacije rade zajedno kao cjelina i aplikacija odgovara napisanim testnim scenarijima. (Cypress, 2024) E2E testiranje iznimno bitno za razvoj i održavanje pouzdanih aplikacija, jer omogućuje otkrivanje problema koje drugi testovi možda ne bi primijetili. Pruža provjeru cjelokupnog protoka podataka, osiguravajući dosljednost i točnost kroz sve sustave. (Skadorva, 2023)

Postoje različiti komercijalno dostupni alati za automatizirane testove, uključujući alate za jedinično testiranje, integracijsko testiranje i end-to-end testiranje. Alati za unit testiranje fokusiraju se na testiranje pojedinačnih komponenti ili funkcionalnosti koda. Takav pristup omogućava precizno pronalaženje grešaka unutar testiranih komponenti ili funkcionalnosti. S druge strane, integracijski se alati bave provjerom suradnje između tih komponenti, ispitujući kako različiti moduli međusobno komuniciraju. Kako e2e testovi uključuju cjelokupnu funkcionalnost aplikacije iz perspektive korisnika, često se koriste alati za snimanje i ponavljanje automatizacije koja koristi grafička korisnička sučelja. (Ammann, 2017) Često korišteni alati za pisanje e2e automatiziranih testova uključuju Cypress, Selenium, Appium, Cucumber, Katalon i Pupperter. Svaki od ovih alata ima svoje specifičnosti i prednosti, ovisno o potrebama projekta. Specifične mogućnosti i prednosti svakog od ovih alata, kao i njihove potencijalne primjene u različitim

kontekstima razvoja softvera, bit će detaljnije obrađene u sljedećoj cjelini. Razumijevanje ovih alata ključno je za odabir prikladnog pristupa testiranju u različitim fazama razvoja softverskih projekata.

3.2. Pregled alata za automatizaciju testova web aplikacija

Jedan od starijih alata za e2e testiranje web aplikacija je Selenium, on je počeo sa radom kada nije bilo puno alata za automatizaciju web aplikacija. Selenium je 2004. Godine predstavio novi skup alata za automatizaciju repetitivnih zadataka, radi toga danas ima veliku zajednicu i podršku velikih tvrtki. (Gelfenbuim, 2022) Selenium se prvenstveno koristi za automatizaciju web preglednika, odnosno za razvoj end-to-end testova za web aplikacije. On se sastoji od Selenium WebDriver za upravljanje preglednicima, Selenium IDE za snimanje i pregledavanje testova, te Selenium Grid za paralelno izvođenje testova na više okruženja. (Selenium, 2024).

Appium je alat za automatizaciju testova koji koristi istu komponentu za upravljanje preglednicima kao i Selenium, koristi Seleniuim WebDriver. Appium se prvenstveno koristi za automatizaciju mobilnih aplikacija na iOS i android uređajima, ali može se koristiti i za web aplikacije. (Appium, 2024) Appium i Selenium podržavaju veliki broj programskih jezika poput Java, Python i JavaScripta, dostupni su na Windowns, macOS i Linux operativnim sustavima i besplatni su za korištenje. Nedostaci Seleniuma i Appiuma su što nemaju mogućnost automatiziranog izvještavanja i potrebna je barem srednja razina poznavanja programskih jezika za pisanje testova. (Selenium, 2024)

Cucumber je alat za pisanje automatiziranih e2e testova u Gherkin jeziku koji omogućava ne tehničkim osobama da pišu specifikacije testova. (Gelfenbuim, 2022) Drugim riječima specifikacije testova se pišu koristeći prirodan, a ne programski jezik. Pisanje u prirodnom jeziku je moguće radi Gherkin sintakse koja je dizajnirana da bude razumljiva tehničkim i ne tehničkim osobama. Gherkin koristi strukturu na prirodnom jeziku za definiranje testnih scenarija kroz ključne riječi poput "Given", "When" i "Then". Koristeći tu strukturu u Cucumberu se definiraju scenariji koji opisuju korake koje softver treba izvršiti. Cucumber izvršava zadane testne scenarije, provjerava je su li ispunjeni svi kriteriji i generira izvještaj koji pokazuje uspjeh ili neuspjeh svakog

scenarija. Gherkin sintaksa je prevedena na sedamdeset jezika, uključujući i hrvatski. Na slici 1. prikazan je hrvatski ekvivalent Given, When, Then sintakse.(Cucumber, 2024)

`given`	`*`
	`Zadan`
	`Zadani`
	`Zadano`
	`Ukoliko`

`when`	`*`
	`Kada`
	`Kad`

`then`	`*` `Onda`

Slika 1 Gherkin sintaksa na hrvatskom jeziku
Izvor: (Cucumber, 2024)

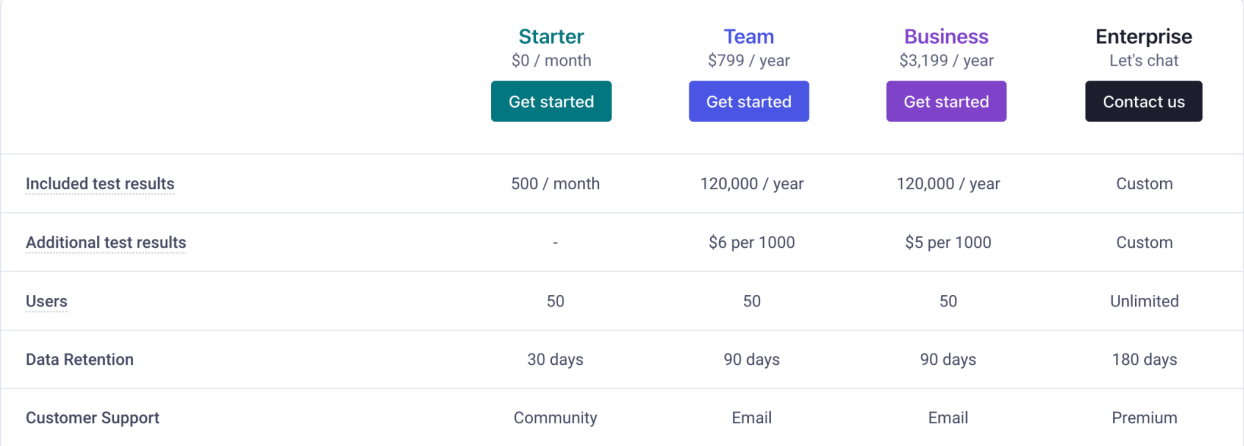
Iako su testovi napisani na prirodnom jeziku, iza tih testova stoji programska logika koja se implementira u programskom jeziku poput Jave, JavaScripta, Pythona i Rubyja. Kao i Selenium, Cucumber je dobar izbor za pisanje e2e testova za web aplikacije jer omogućava definiranje scenarija koji obuhvaćaju cjelokupni korisnički tijek kroz aplikaciju. Cucumber besplatan alat za pisanje automatiziranih testova i moguće ga je koristiti na Windowns, macOS i Linux operativnom sustavu. (Cucumber, 2024)

Slijedeći popularan alat za automatizirano testiranje e2e testova je Katalon Studio. Dostupan je na Windows, macOS i Linux operativnom sustavu, te omogućava kreiranje, izvođenje i upravljanje testovima za web aplikacije, mobilne aplikacije i API testove. Omogućava razne funkcionalnosti i integracije sa drugim alatima i besplatan je za korištenje. Katalon Studio ima mogućnost brzog stvaranja testova koristeći snimanje bez koda i drag-and-drop objekt za jednostavnije kreiranje testova. Od programskih jezika za kreiranje testova koristi Groovy i Javu. Testovi se mogu pokrenuti lokalno ili u cloud okruženjima. Nudi intuitivno pronalaženje pogrešaka, automatsko popravljanje testova i detaljne izvještaje. Kao što je već spomenuto, Katalon Studio podržava end-to-end testiranje web aplikacija, uključujući simulaciju korisničkih interakcija i testiranje na različitim preglednicima. (Katalon, 2024)

Puppeteer je alat za testiranje putem JavaScript knjižnice koja omogućuje upravljanje preglednicima Chrome ili Firefox preko DevTools protokola ili WebDriver BiDi-ja. On omogućuje automatizirano testiranje radnji u preglednicima i testiranje korisničkog sučelja, a pisanje testova isključivo je u JavaScriptu i Node.js-u. Puppeteer omogućava oponašanje korisničkog ponašanja u preglednicima što ga čini dobrim odabirom za e2e testiranje. Pisanje testova definira korake koje preglednik treba izvršiti, kako se kretati po stranici i kako komunicirati sa elementima na stranici, te provjera očekivanih rezultata. Puppeteer je također besplatan alat za automatizaciju testiranja, ali ne nudi opciju izvještaja koji testovi su prošli, a koji pali. (Pupperter, 2024)

Jedan od najpopularnijih alata za automatizirano testiranje u novije vrijeme je Cypress. On je skup moćnih JavaScript alata koji omogućavaju testiranje svih web aplikacija. (Gelfenbuim, 2022) Korisnici Cypress alata su JavaScript programeri, ali i testeri koji pišu automatizirane testove. Cypress omogućuje pisanje više vrsta testova, testovi komponenti, integracijski testovi, jedinični testovi i e2e testovi. U Cypressu se može testirati sve što se pokreće u web pregledniku, on se koristi za testiranje očekivanog ponašanja komponenti u bilo kojem web pregledniku uz trenutne vizualne povratne informacije. Cypress omogućava postavljanje, pisanje i pokretanje testova, njegova arhitektura izgrađena je da dobro radi s modernim JavaScript okvirima. Automatizirani testovi u Cypressu se isključivo pišu u JavaScriptu ili Typescriptu,(Cypress, 2024), za njih je na web stranici Cypressa dostupna bogata API dokumentaciju sa besplatnim vodičima i tečajevima. (Gelfenbuim, 2022)

Cypress omogućuje snimanje testova koji se izvode, te jednostavno otklanjanje pogrešaka pomoću developer tools alata. Kako bi testovi bolje radili, Cypress ima automatsko čekanje na izvršenje naredbi od 8 sekundi koje je moguće skratiti ili produžiti ovisno i potrebi specifičnog testa. Za razliku od velikog broja alata za e2e testiranje Cypress ne koristi WebDriver, niti istu arhitekturu kao Selenium. Od pregledanih alata za e2e testove, osim besplatne verzije, Cypress jedini nudi više modela plaćanja pogodnih za manje timove, poslovanje ili veliku korporaciju. (Cypress, 2024) Na slici 2. vidljivi su Cypress modeli plaćanja.



	Starter $0 / month Get started	Team $799 / year Get started	Business $3,199 / year Get started	Enterprise Let's chat Contact us
Included test results	500 / month	120,000 / year	120,000 / year	Custom
Additional test results	-	$6 per 1000	$5 per 1000	Custom
Users	50	50	50	Unlimited
Data Retention	30 days	90 days	90 days	180 days
Customer Support	Community	Email	Email	Premium

Slika 2 Cypress modeli plaćanja

U nastavku je pregled opisanih alata za e2e testiranje, njihov naziv, na kojem operativnom sustavu ih je moguće pokretati i koristiti, te jesu li besplatni za korištenje ili imaju plaćenu verziju. Uz svaki alat za testiranje prikazani su programski jezici u kojima je moguće pisati automatizirane testove, te nudi li alat opciju generiranja automatiziranog izvještaja nakon izvršenja napisanih testova. Iz pregleda se može zaključiti da svi alati imaju besplatnu verziju i dostupni su na Windowns, macOS i Linux operativnim sustavima. Jezik u kojem je moguće pisati testove ovisi u alatu, ali prevladava JavaScript, te više od pola alata pruža opciju generiranja automatiziranog izvještaja.

Tablica 1 Pregled alata za automatizaciju testova web aplikacija

Alati	OS	Jezici	Izvještaji	Cijena
Selenium i Appium	Windows, macOS, Linux	Java, PHP, C#, Ruby, JavaScript, Python, Perl	Ne	Besplatan
Cucumber	Windows, macOS, Linux	Gherkin	Da	Besplatan
Katalon	Windows, macOS, Linux	Java, Groovy	Da	Besplatan

Pupperter	Windows, macOS, Linux	JavaScript	Ne	Besplatan
Cypress	Windows, macOS, Linux	JavaScript	Da	Besplatna i plaćena verzija

Izvor: (Selenium, 2024), (Appium, 2024), (Cucumber, 2024), (Pupperter, 2024), (Katalon, 2024), (Cypress, 2024)

3.3. Prednosti i nedostaci manualnog i automatiziranog testiranja

Svaka vrsta testiranja ima svoje prednosti i nedostatke, a odabir vrste testiranja ovisi o potrebama tvrtke. Kako bi se odabrala vrsta testiranja postoje razni faktori koji utječu na odabir tehnike. Ako u zahtjevima za razvoj softvera nedostaju dijelovi koda, potrebno je provjeriti zahtjeve, s druge strane provjera strukture softvera može testirati samo već razvijeni dio aplikacije koji postoji u kodu. Ako postoji greška u zahtjevima softvera i u kodu, grešku će otkriti samo tehnike temeljene na iskustvu. Drugim riječima tehnike i vrsta testiranja ovise o vrsti sustava, zahtjevima korisnika, standardima, cilju testiranja, dostupnoj dokumentaciji, znanju testera, vremenu, budžetu, te prethodnom iskustvu testera u pronalaženju određenih vrsta grešaka. (Graham et al., 2019)

Ako se ispravno implementira, testiranje softvera može dugoročno smanjiti troškove softverskog proizvoda. Cilj testiranja je primijetiti probleme i greške softvera u što ranijoj fazi razvoja ciklusa softvera kako bi trošak ispravljanja bio manji (Kaul,2023) i povrat investicije što veći. (Ammann i Offutt, 2017) U svakoj novoj fazi razvoja softvera trošak pronalaženja i ispravljanja grešaka raste i postaje sve teži. Ispravak greške koja je otkrivena nakon što je softver izdan krajnjim korisnicima za korištenje uvijek je veći trošak od iste pogreške koja je pronađena u ranijoj fazi. (Kaul,2023) Glavni razlog za povećanje troška je dodatni trud i vrijeme potrebno za ispravak greške, te vrijeme i trud potrebni za pronalazak greške u softveru koji postepeno raste i postaje sve veći i kompliciraniji. (Ammann, 2017) Osim toga, greška na softveru, uz financijske gubitke može narušiti i reputaciju tvrtke i zadovoljstvo korisnika. (Kaul,2023)

Radi toga bi proces kvalitete i testiranja, trebao odabrati aktivnosti testiranja koje će se odvijati tokom životnog ciklusa softvera na način da budu što isplativiji i poboljšaju ranu vidljivost

pogrešaka. (Pezze, 2008) Pri planiranju projekta, na testiranje i kvalitetu treba gledati kao investiciju. Kako pravilno testiranje otkriva više kvarova i smanjuje troškovi održavanja, može se zaključiti da je ulaganje u testiranje ekonomična odluka. (Kaul,2023) Kako nijedna pojedinačna tehnika analize i testiranja ne postiže željene rezultate, potrebno je kombinirati manualne i automatizirane tehnike analize i testiranja. Glavni razlog tome je što razne vrste testiranja služe drugačijim svrhama. Ako tvrtka odabere sustavno testiranje, ono je usmjereno na maksimiziranje otkrivanja grešaka, ali se ne može koristiti za mjerenje pouzdanosti jer su za to potrebne statističke metode. (Pezze, 2008)

Testeri se susreću sa izazovima pri testiranju nevezano radi li se o tradicionalnim ili agilnim metodama, međutim ovisno o metodi, susreću se sa različitim izazovima. U agilnim metodama, naglasak je na kontinuiranoj provjeri sustava pomoću unaprijed definiranog testnog okvira ili korisničkih priča. U slučaju korisničkih priča, testovi često nisu dokumentirani, dok u tradicionalnom metodama zahtjevi često nisu potpuni i ažurni. Osim razvijanja testova, testeri moraju redovito ažurirati testove kako se softver razvija i testirati nove funkcionalnosti. Međutim, nije dovoljno testirati samo nove funkcionalnosti, kako bi se osigurala kvaliteta softvera potrebno je redovito testirati i stare funkcionalnosti softvera, odnosno izvršavati regresijsko testiranje. (Ammann i Offutt, 2017)

Manualni testiranje za vrijeme razvoja softverskih proizvoda može poboljšati kvalitetu softvera radi svoje fleksibilnosti i kvalitativnih informacija. Kod automatizacije testova tester kodira testni slučaj i svaki put dovršava precizan skup koraka, dok manualni tester može primijetiti postoji li greška bez potrebe za promjenom retka u kodu. Ovisno o točnoj situaciji, obije vrste testiranja su korisne, ali manualno testiranje pruža fleksibilnost i pronalaženje grešaka koje kod ne može pronaći jer nije napisan da bi primijetio neočekivane probleme. Kod manualnog testiranja tester može sam procijeniti treba li ponoviti neki test ili dodatno istražiti ako vidi da se neki od testnih slučajeva ne ponaša u potpunosti točno. (Zaptest, 2024)

Kvalitativnim informacijama testeri mogu bolje obrazložiti problem i pomoći tima programera da lakše pronađu rješenje za njega. Također, manualni testeri mogu komentirati sučelje aplikacije i prijaviti ako nešto ne izgleda praktično ili razumljivo za korisnika. Korištenjem tih informacija za unaprijeđenije softverskog proizvoda, tvrtke mogu značajno povećati kvalitetu svoje aplikacije. Manualno testiranje koristi se za testiranje upotrebljivosti aplikacije, koje pruža tvrtkama uvid u

aplikaciju i pomaže u prilagodbama koje aplikaciju čine konkurentnijom. Ovu vrstu testiranje nije moguće automatizirati jer je za njega potrebno mišljenje testera i procjena iz perspektive krajnjeg korisnika. (Zaptest, 2024)

Manualno testiranje nije ograničeno platformama i jezikom koje podržava alat za automatizirano testiranje, međutim ograničeno je vještinom i znanjem manualnih testera. Poboljšane vještine testera donose veću vrijednost tvrtki jer ručno testiranje pronalazi više grešaka i poboljšava korisničko iskustvo (Zaptest, 2024), međutim teško je pronaći iskusne testere. Osim toga, ovisno o tome kako se koristi, manualno testiranje može utjecati na porast troškova. Glavni razlog za to je što manualno testiranje može oduzeti jako puno vremena, a dok se softver testira potrebno je platiti sve prisutne, a ne samo testere. Međutim, ovaj problem se djelomično može riješiti planiranjem kada će se koji testovi izvršavati. (Zaptest, 2024)

Prirodno je da ljudi griješe, kod manualnog testiranja softvera moguće je izvršiti testove u pogrešnom redoslijedu ili krivo zabilježiti rezultate testiranja radi pogrešnog klika. Pogreške u procesu testiranja mogu uzrokovati probleme radi netočnih rezultata testova ili propuštenih grešaka u sustavu. Manualno ponavljanje testova je skuplje, naporno i zahtjevno, te s vremenom, povećava troškove testiranja softverske aplikacije. (Kaul,2023) Radi toga se manualni testeri mogu umoriti i iscrpiti, te imaju veću vjerojatnost pogrešaka od ostalih. Korištenjem automatiziranih testova uz manualne može se izbjeći iscrpljenost i osigurati redovite pauze od ekrana kako bi se upravljalo radnim opterećenjem testera. (Zaptest, 2024)

Manualni i automatizirani testovi mogu se primjenjivati u bilo kojoj fazi razvoja softvera, posebno u ranim fazama specifikacija i dizajna. Analiza zahtjeva oduzima značajnu količinu vremena, što može oduzeti vrijeme za testiranja. (Pezze, 2008) Radi toga kod primjene agilnih metoda razvoja softvera, automatizacija testova je vrlo bitna jer omogućava brzo identificiranje i ispravljanje grešaka i smanjuje potrebu za manualnim testiranjem. (Ammann, 2017) Nakon testiranja, potrebno je napisati izvještaj, koji testovi su pali, a koji prošli. Kod manualnog testiranja izvještaj ima više informacija, te opisuje zašto je neki testni slučaj pao i na kojem dijelu, dok automatizirani testovi generiraju vlastita izvješća na kraju procesa. Sva automatizirana izvješća imaju isti format, međutim kod manualnih izvještaja je puno lakše dodati sve informacije koje tester smatra da bi mogle biti koriste razvojnom timu. (Zaptest, 2024)

Implementacija automatiziranih testova pomaže produktivnosti softvera i omogućava isporuku u brzim promjenjivim ciklusima. Iako je početni trošak ulaganja u automatizirano testiranje visok, ono smanjuje dugoročne troškove. Radi visokih troškova ulaganja, tvrtke se teško odluče na automatizirane testove, međutim, u usporedbi sa manualnim testiranjem, automatizacija u testiranju softvera se pokazala troškovno učinkovitija jer uz ispravnu implementaciju i korištenje može doseći točku povrata ulaganja. Jednom napisani automatizirani testovi se mogu ponovno koristiti bez dodatnih troškova, a ako se izvršava više testova istovremeno troškovna učinkovitost je još veća. Sa tim pristupom se potrebna količina vremena i truda smanjuje, a povrat početnih troškova sve veći što se više koriste automatizirani testovi. (Kaul,2023)

Automatizacija testova oslobađa vrijeme potrebno za testne slučajeve za koje je manualno testiranje neophodno. Osim toga, automatizirani testovi se brzo izvršavaju što omogućuje brzu prijavu pronađenih grešaka i samim time, brže ispravljanje istih. Time se smanjuje vrijeme testiranja, ali i skraćuje razvojni ciklus softvera što pomaže u brzini izdavanja softvera i njegovih novih verzija. (Kaul,2023) Osim brzine testiranja, automatizacija pomaže povećati dubinu o opseg testiranja, skripte često uključuje složene testove koji se obično ne mogu obaviti ručno, a bitne su za ispravno funkcioniranje softvera. (Kaul,2023) Kako bi se prednosti automatiziranog testiranja maksimalno iskoristile potrebno je razvijati nove testove kako se razvija softver, time se osigurava da su testovi relevantni i pokrivaju sve nove promjene u softveru. Također, bitna je komunikacija između svih članova tima kako bi se osigurala usklađenost između korisničkih priča, razvoja softvera i testiranja. (Ammann i Offutt, 2017)

Kako uvođenje automatizacije testiranja zahtjeva nove strategije testiranja, nove procese, nove radne tokove, nove alata i nove vještine za programere i testere, u tom procesu se pojavljuju i određeni izazovi. (Forgacs i Kovacs, 2024) Prvi izazov je odabir alata koji odgovara programerima, testerima i projektu za koji se testovi automatiziraju. Nakon odabira alata, i procesa prilagodbe i učenja korištenja novog alata, pojavljuje se izazov sa lokatorima za GUI objekte. Ako GUI objekti nemaju jedinstvene lokatore, odnosno ID-jeve za svaki bitan GUI objekt, nakon svake promjene u kodu, potrebno je prilagoditi i lokatore u automatiziranim testovima kako bi oni i dalje ispravno radili. Takvo održavanje testova je značajan trošak koji se može izbjeći planiranjem i suradnjom između programera i testera. (Forgacs i Kovacs, 2024)

Zahtjevi za softver većinom počinju u obliku prirodnog jezika radi čega je automatizacija analize i testiranja otežana. Radi toga je bitno kombinirati manuale i automatizirane testove tokom životnog ciklusa softvera, automatizacija testova za ponavljajuće aktivnosti i manualne testove za dizajn korisničkog sučelja i kreativne zadatke. (Pezze, 2008) Kombinacija opisanih vrsta testiranja omogućava tvrtkama koje koriste agilne metoda za razvoj softvera da učinkovito testiranju softver i upravljaju troškovima. (Ammann, 2017)

3.4. Studija slučaja triju poduzeća

Poduzeća imaju razne razloge zašto koriste ili ne koriste automatizirane testove. Ovisi o strukturi i veličini poduzeća koriste različite alate za različite vrste testiranja. U nastavku se nalazi tablica 2. sa pregledom poduzeća koja su pristala pričati o procesu testiranja u svom poduzeću. Postavlja se pitanje, jesu li automatizirali testiranje u svome poduzeću, koju vrstu testova su automatizirali, te koje alate za automatizaciju koriste za koju vrstu testiranja. Također zašto su se odlučili za odabrane alate i kako automatizacija pomaže njihovom poduzeću. Radi zaštite privatnosti, poduzeća će ostati anonimna te se oslovljavati sa poduzeće A, poduzeće B i poduzeće C.

Tablica 2 Primjena alata za automatizaciju testova u praksi po poduzećima

Poduzeće	**Automatizacija**	**Alat**	**Vrste testova**
A	Da	Cypress	Integracijski i "End to end"
B	Da	Selenium	Integracijski i "End to end"
C	Da	Cucumber	"End to end"

Poduzeće A je manja hrvatska IT tvrtka koja se bavi razvojem softvera sa svega dvadesetak zaposlenika. Uz nekoliko mobilnih aplikacija, trenutni glavni fokus im je razvoj web aplikacija radi čega su prije godinu dana primijetili potrebu za automatiziranim testovima. Kako su se zahtjevi

za web aplikacije širili i preuzimali su projekte za izradu i preradu novih web aplikacija primijećena je potreba za bržim regresijskim testiranjem koje je oduzimalo jako puno vremena, Nadalje, uz nedostatak vremena imali su i nedostatak ljudskih resursa za testiranje. Kako u tom trenutku nije bilo moguće uložiti u ljudske resurse, odlučili su se za ulaganje u automatizirane testove. Prvi korak je bila analiza alata za automatizirane testove, ideja je bila odabrati alat koji je kompatibilan sa tehnologijama tvrtke i lako se integrira u postojeći sustav.

Radi razvoja web aplikacija u JavaScriptu i poznavanja alata Selenium i Cypress, provedena je analiza između ta dva alata. Na kraju je odabran alat Cypress radi jednostavnosti korištenja i prijašnjeg iskustva u korištenju od strane glavnog frontend programera za web aplikacije. Tim odabirom je započela automatizacija e2e regresijskih testova, u poduzeću A e2e testove piše testni tim uz pomoć razvojnog tima. Osim e2e testova, poduzeće A je odnedavno počelo koristiti Cypress i za integracijske testove koji su se prije pisali i izvršavali u JUnitu. Integracijske testove isključivo piše razvojni tim kako bi provjerili ispravnost pojedinih komponenti Java projekata.

Uvođenjem automatiziranih testova u poduzeću A postignut je značajan napredak u procesu testiranja. Redovitim korištenjem automatiziranih rješenja, tvrtka je ostvarila povrat ulaganja kroz bržu isporuku softvera i smanjenje troškova. Automatizacija testiranja omogućila je testerima da se više posvete manualnom testiranju, detaljnijim analizama te unapređenju i proširivanju testnih koncepata. Vrijeme koje su testeri prije trošili na ponavljajuće zadatke sada se koristi za strateški važnije aktivnosti, što doprinosi kontinuiranom poboljšanju kvalitete softverskih proizvoda i većoj efikasnosti u cjelokupnom razvojnom procesu.

Poduzeće B je velika financijska tvrtka sa vlastitim IT odjelom koje je samostalno razvilo svoje web i mobilne aplikacije. Testiranje aplikacija je donedavno bilo isključivo manualno, međutim primijećena je potreba za automatizacijom procesa testiranja radi čega je poduzeće nedavno je počelo sa procesom edukacije zaposlenih za korištenje alata Selenium. Za velike tvrtke sa velikim projektima Selenium je jako dobar odabir jer podržava više programskih jezika i platformi tako da se može koristiti za razne projekte. Testovi koje je poduzeće B krenulo pisati u Seleniumu su e2e i integracijski testovi. Uvođenjem automatizacije testiranja, poduzeće B nastoji ubrzati i unaprijediti proces testiranja, te samim time povećati povrat ulaganja.

Poduzeće C specijalizirano je za razvoj softvera, mobilnih i web aplikacija te implementaciju ERP sustava i integraciju AI rješenja. Njihov tim čine stručnjaci iz različitih područja koji surađuju kako bi pružili visokokvalitetna rješenja prilagođena specifičnim potrebama klijenata. Kako bi održali visoku kvalitetu svojih softverskih rješenja bitno je imati kvaliteta proces testiranja. Poduzeće C ima poseban tim testera koji kreiraju e2e testne slučajeve u Gherkin, "Given, When, Then" sintaksi. Također, radi jednostavnosti Gherkin sintakse poduzeće C nije moralo uložiti u edukaciju testera. Nakon što testni tim kreira testni koncept sa testnim slučajevima, razvojni tim prepisuje te testove u programski jezik Ruby kako bi se izvodili u alatu za automatizirano testiranje Cucumber.

Iako Cucumber podržava izvođenje testova napisanih korištenjem Gherkin sintakse, poduzeće C prepisuje testove u Ruby kako bi se preciznije izrazile i testirale kompleksne logike softvera. Još jedan razlog prepisivanja testova u Ruby je radi lakšeg održavanje testova kako testni sustav raste. Također, integracija sa razvojnim alatima je jednostavnija kada su testovi napisani u programskom jeziku. Poduzeće C koristi Gherkin i Cucumber jer im omogućuje suradnju između testnog i razvojnog tima. Testni tim izrađuje testne koncepte u Gherkinu kako bi omogućio razvojnom timu brzu automatizaciju e2e testova bez opsežne pripreme. Na taj način, razvojni tim može učinkovitije automatizirati kompleksne funkcionalnosti koje Gherkin ne podržava koristeći programski jezik.

Tablica 3 Ključne informacije sa intervjua sa poduzećima

Poduzeće	Približan broj zaposlenih	Proces automatizacije	Edukacija zaposlenika	Tko piše testove
A	20	U razvoju	Ne	Testeri
B	500	U razvoju	Da	Testeri
C	200	Razrađen	Ne	Razvojni tim

U Tablici 3 vidljiv je pregled ključnih informacija sa intervjua sa poduzećima. Iz tablice se primjećuje da i manja i veća poduzeća primjećuju potrebu i koristi automatizacije testiranja softvera. Poduzeće C je veće poduzeće sa dvjestotinjak zaposlenih koje se bavi razvojem softvera tako da već ima uhodan i razrađen proces automatizacije. S druge strane, također veliko poduzeće

B tek uvodi procese automatizacije testiranja softvera, ali za razliku od poduzeća C se ne bavi razvojem softvera kao primarnom aktivnošću. Poduzeće A je vrlo brzo primijetilo potrebu za automatizacijom procesa testiranja, međutim, radi nedostatka ljudskih resursa tek uvodi automatizirane testove.

Od ispitanih poduzeća, samo je najveće poduzeće, poduzeće B uložilo u edukaciju zaposlenika o procesu testiranja i automatizaciji testiranja softvera. Poduzeće C je taj izazov riješilo podjelom posla, dio posla za koji nisu potrebne dodatne vještine obavljaju testeri, dok automatizaciju obavlja razvojni tim koji već posjeduje potrebne vještine. Poduzeće A trenutno nema resurse za formalno ulaganje u edukaciju zaposlenika o testiranju softvera, ali ima proces mentorstva gdje se testeri sami uče, a razvojni tim provjerava napisani kod.

Sve više poduzeća prepoznaje vrijednost i potrebu za automatizacijom testiranja, iako to donedavno nije bila uobičajena praksa. Dok je poduzeće C jedino od ispitanih koje već godinama ima uhodan i optimiziran proces testiranja, poduzeća A i B su tek nedavno uočila važnost automatizacije i počela ulagati resurse u njen razvoj. Ovaj trend ukazuje na sve veću svijest o prednostima koje donosi automatizacija, poput bržeg otkrivanja grešaka i poboljšanja kvalitete proizvoda. Automatizacija testiranja se sve više koristi, ali prilagodba procesa i raspodjela uloga unutar timova ovisi o specifičnim potrebama i resursima svakog poduzeća.

4. KOMPARATIVNA ANALIZA MANUALNOG I AUTOMATIZIRANOG TESTIRANJA WEB APLIKACIJA NA PRIMJERU WEB STRANICE EKONOMSKOG FAKULTETA SVEUČILIŠTA U ZAGREBU

4.1. Priprema testnog koncepta za manualna testiranja

Priprema testnog koncepta za manualno testiranje bazira se na njihovim karakteristikama i značajkama koje je bitno planirati prilikom dizajna manualnih testova. (Zaptest, 2024) Njihov sadržaj je prikaz testova i opis kako će biti provedeni. Čest slučaj je da tvrtke i kupci imaju predloške testnih koncepata koje koriste za sve svoje softverske proizvode. Na internetu postoji veliki broj testnih planova i okvira za testiranje, uključujući i IEEE standard za dokumentaciju testiranja softvera i sustava. U standardu postoje dva glavna tipa testnih planova. Glavni plan koji je zapravo cjelokupni dokument za planiranje testiranja i upravljanje više razina testiranja. Glavni plan se odnosi na jedan ili više projekata unutar iste organizacije. Drugi tip je plan testiranja, on mora opisati opseg, pristup, resurse i raspored aktivnosti testiranja za svoju razinu testiranja. (Ammann i Offutt, 2017)

Manualni testni slučajevi se optimiziraju na upute koje tester mora izvršiti prije završetka testa, time se štede vrijeme i resursi jer tester izvršava više malih zadataka. Gdje god je moguće, optimalno je ograničiti veličinu testnog slučaja kako bi se dostupni resursi maksimalno iskoristili. (Zaptest, 2024) Manualni testni koncept mora biti lako razumljiv i njegovi kriteriji prolaznosti moraju biti laki za generirati kako bi se potrošilo manje vremena za kasniju analizu u procesu. (Zaptest, 2024) Prije dizajna testova potrebno je analizirati zahtjeve softvera, glavni cilj dizajna testova je pripremiti aktivnosti testiranja i verifikacije sustava. Testni koncept trebao bi biti izrađen da način da prikaže ispunjava li softver zahtjeve ili ne. Također, proces izrade testnih scenarija pomaže otkriti nejasne specifikacije zahtjeva. (Ammann, 2017)

Izrada testnog koncepta radi se u nekoliko koraka, prvi korak je analiza zahtjeva i tehnički pristup testiranju. Nakon analize, odabire se i izrađuje dizajn testova koji zadovoljava ciljeva testiranja. Kao što je već napomenuto, testni koncept mora biti lako razumljiv i pokrivati sve bitne scenarije sa jednim ili više testova. Potrebno je odabrati koliko testnih scenarija je potrebno kako bi se testirao svaki testni scenarij. (Forgacs i Kovacs, 2024) Za analizu i odabir pomaže imati strukturu testnog koncepta, kao svaki dokument, testni koncept mora imati jedinstveno ime, informacije o

autoru, datumu i opseg. Opseg opisuje što treba testirati, te može sadržavati koji dio softvera se testira. Ako postoje povezani dokumenti, potrebno ih je navesti u testni koncept. Osim toga potrebno je navesti testne klase, neki jedinstveni ID svakog testnog slučaja te opće uvjete testiranja, odnosno što se očekuje od softvera ako se ponovi određeni slijed aktivnosti. (Ammann i Offutt, 2017)

Testni koncept mora sadržavati i okruženje u kojem se test izvršava. (Ammann i Offutt, 2017) Za testiranje se u pravilu ne koristi verzija koja je u okruženju dostupnom krajnjim korisnicima, već u posebnom testnom okruženju. U testnom konceptu moraju biti navedene sve značajke koje treba testirati koristeći nazive koji su korišteni u softverskoj dokumentaciji. Za svaku stavku koja se testira potrebno je napisati kriterij prolaza i neuspjeha, drugim riječima što je očekivano ponašanje softvera. (Ammann i Offutt, 2017) To može biti očekivani izlaz ili neki drugi kriterij utvrđivanja ispravnosti izvršenja programa. (Pezze, 2008) Ako se očekivano i stvarno ponašanje ne podudaraju, test nije prošao. Testni koncept treba definirati kriterij za obustavu testiranja, ako je softver u potpunosti nedostupan ili ima grešku na ključnom dijelu softvera bolje je pričekati ispravljanje greške od strane razvojnog tima prije nastavka testiranja ili ponavljanja testiranja. (Ammann, 2017)

Kod dizajna testnih slučajeva, softver se često podijeli po funkcionalnostima kako bi se lakše testiralo ponaša li se sustav kako se očekuje. Ako je potrebno i moguće, dizajn testova uključuje i stvaranje skupa podataka za testiranje kako bi se osigurala pokrivenost svih mogućih scenarija i uvjeta. (Forgacs i Kovacs, 2024) U testnom konceptu bitno je naznačiti verziju softvera koja se testira i koji hardver ili softver je potreban za testiranje. U slučaju web aplikacija to su razni preglednici na računalima, dok su za mobilne aplikacije iOS i Android mobilni uređaji. (Pezze, 2008) U nastavku su kreirani testni slučajevi za testni koncept za manualno testiranje web stranice Ekonomskog fakulteta u Zagrebu. U naslovu testnog koncepta, bitno je navesti ime autora testa, ime testera, datum izvođenja testa i preglednik u kojem se test izvodi.

U nastavku se nalazi tablica 3 u kojoj je opisan testni slučaj za posjet web stranici Ekonomskog fakulteta. Svaki testni slučaj mora imati jedinstvenu oznaku testa, odnosno ID testa, naziv testa, korake koje je potrebno pratiti kako bi se test izvršio, očekivano ponašanje softvera ako pratimo nabrojane korake, stvarno ponašanje softvera nakon praćenja koraka i rezultat. Dodatno, može se dodati stupac zvan komentar za sva opažanja koja nisu stvarno ponašanje ili rezultat, ali su bitna za zapisati i napomenuti razvojnom timu nakon dovršenog testiranja. Za testni slučaj Posjet web

stranici EFZG-a potrebno je otvoriti preglednik, upisati URL ekonomskog fakulteta u Zagrebu u polje za pretraživanje i pritisnuti Enter. Stupac stvarno ponašanje i rezultat popunjavaju se prilikom izvršavanja testova.

Tablica 4 Testni slučaj - Posjet web stranici EFZG-a

ID	Naziv testa	Koraci	Očekivano ponašanje	Stvarno ponašanje	Rezultati
e-01	Posjet web stranici EFZG-a	1. Otvorite web preglednik 2. U pretraživač upišite https://www.efzg.unizg.hr/ 3. Pritisnite Enter	Pretraživanjem https://www.efzg.unizg.hr/ u pregledniku i pritiskom tipke Enter web stranica EFZG-a bi se trebala otvoriti		

Tablica 4 prikazuje testni slučaj za provjeru jednu od stavki navigacijske trake, O nama. Drugim riječima dođe li se kliknemo na tekst "O nama" na navigacijskoj traci na odgovarajući URL gdje se nalazi tekst sa informacijama o fakultetu. U stupcu koraci može se primijetiti preduvjet prije koraka jedan. Umjesto prepisivanja koraka koji su potrebni za doći na stranicu fakulteta, preduvjetom se označi da prije izvršavanja testa moramo ispuniti neki uvjet.

Tablica 5 Testni slučaj - Navigacijska traka O nama

ID	Naziv testa	Koraci	Očekivano ponašanje	Stvarno ponašanje	Rezultati
e-02	Navigacijska traka - O nama	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na tekst "O nama" na navigacijskoj traci	Klikom na tekst "O nama" trebala bi se otvoriti stranica sa informacijama o Ekonomskom fakultetu u Zgrebu		

Testni slučaj u tablici 4 zahtjeva i dodatnu provjeru sadržaja na stranici "O nama", pokazuje li ispravan tekst o fakultetu i nalazi li se na odgovarajućem mjestu sa odgovarajućim fontom i bojom. Tablica 5 prikazuje još nekoliko slučajeva gdje je potrebno dodatno testiranje sadržaja, a ne samo kratka provjera točnosti url-a. Prvi primjer je test promjene jezika sa hrvatskog na engleski gdje je

potrebna dodatna provjera svog teksta na stranici nakon što se jezik promjeni i provjera obavijesti. Provjera obavijesti je isključivo manualni test koji treba izvršavati nakon objave novih obavijesti. Test treba sadržavati je li obavijest uspješno objavljena, sadrži li željeni tekst i nalazi li se na odgovarajućoj lokaciji.

Tablica 6 Testni slučajevi provjere informacija

ID	Naziv testa	Koraci	Očekivano ponašanje	Stvarno ponašanje	Rezultati
e-10	Promjena jezika	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na EN u gornjem desnom kutu	Klikom na EN u gornjem desnom kutu, URL stranice bi trebao dobiti nastavak /en i sadržaj stranice bi trebao biti na engleskom jeziku		
e-11	Provjera obavijesti	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na "Prikaži" pored odlomka Obavijesti	Klikom na Prikaži trebalo bi se pojaviti 9 zadnjih obavijesti Ekonomskog fakulteta u Zagrebu, poredane od najnovije do najstarije		

Tablica 6 prikazuje jedan od testova za provjeru ikona za društvene mreže na web stranici Ekonomskog fakulteta u Zagrebu. Sadržaj testa sastoji se od odlaska na stranicu fakulteta, klika na jednu od ikona društvenih mreža. Tablica prikazuje test provjere Facebook profila, drugim riječima hoće li korisnik nakon klika na ikonu od facebook doći na Facebook profil Ekonomskog fakulteta u Zagrebu. Isti test potrebno je ponoviti za sve prikazane ikone društvenih mreža na stranici fakulteta.

Tablica 7 Testni slučaj - Društvene mreže, Facebook EFZG-a

ID	Naziv testa	Koraci	Očekivano ponašanje	Stvarno ponašanje	Rezultati

e-12	Link na Facebook	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na ikonu Facebook u gornjem desnom kutu	Klikom na Facebook ikonu trebao bi se otvoriti Facebook profil Ekonomskog fakulteta u Zagrebu		

Posljednji primjer manualnih testnih slučaja je pokušaj prijave studenta sa točnim i netočnim korisničkim imenom i lozinkom. U tablici 7 vidljiva su dva testna slučaja koji opisuju da bi upisivanje točnog korisničkog imena i lozinke trebalo rezultirati uspješnom prijavom u sustav, dok bi netočno korisničko ime ili lozinka trebalo rezultirati greškom, a prijava ne bi trebala biti moguća. Potpuni testni koncept nalazi se u dodatku 1.

Tablica 8 Testni slučaj - Prijava studenta

ID	Naziv testa	Koraci	Očekivano ponašanje	Stvarno ponašanje	Rezultati
e-17	Student login	1. Upišite slijedeću adresu u pretraživač: https://student.efzg.hr/login 2. Otvorite web stranicu 3. Upišite točno korisničko ime i lozinku 4. Kliknite na gumb "Prijava"	Upisivanjem točnog korisničkog imena i lozinke, te klikom na gumb "Prijava" korisnik bi trebao biti uspješno ulogiran, te vidjeti svoj profil sa imenom i prezimenom u gornjem desnom kutu.		
e-18	Student login greška	1. Upišite slijedeću adresu u pretraživač: https://student.efzg.hr/login 2. Otvorite web stranicu 3. Upišite netočno korisničko ime i/ili lozinku 4. Kliknite na gumb "Prijava"	Upisivanjem netočnog korisničkog imena i/ili lozinke login ne bi trebao biti uspješan i trebala bi se pojaviti greška		

4.2. Selekcija testova za automatizaciju

Prije pisanja automatiziranih testova potrebno je odabrati koji testni slučajevi će se automatizirati, a koji neće. Prilikom selekcije testova treba se uzeti u obzir da se prikupljanje, sažimanje i predstavljanje podataka može se automatizirati dok je za testiranja dizajna sučelja i tumačenje informacija na web stranici potrebno manualno testiranje. Pri odabiru testova za automatiziraju također treba uzeti u obzir koliko često će se ponavljati ti testovi i koliko dugo će se moći koristiti. (Pezze, 2008)

Na temelju tablice u dodatku 1, Testni koncept za web stranicu Ekonomskog fakulteta u Zagrebu, odabrani su slijedeći testovi za automatizaciju, posjet web stranici fakulteta i provjera navigacijske trake. Ovaj set testova odabran je za automatizaciju jer se jednostavno i brzo može izvršiti pomoću automatiziranih testova. Potrebno je napomenuti da ovi testovi provjeravaju hoće li korisnik doći na točnu stranicu klikom na određeni dio web stranice, one se provjeravaju sadržaj web stranice. Za takav tip provjere, potrebno je manualno testiranja.

Testovi za opciju pretrage pojmova na web stranici fakulteta i promjene jezika također su odabrani za automatizaciju. Test pretrage pojma na web stranici odnosi se na upisivanje pojma u tražilicu, pretraživanje i provjeru nalaze li se rezultati sa tim pojmom u dijelu za web stranice gdje se prikazuju rezultati pretraživanja. Promjena jezika sa hrvatskog na engleski, izvršit će se na sličan način kao provjera navigacijske trake, nakon promjene jezika, url stranice bi se trebao promijeniti na englesku verziju koja završava sa /en. Kako je već objašnjeno, provjera obavijesti neće biti automatizirana jer ona zahtjeva provjeru informacija. Osim toga, obavijesti se mijenjaju i s vremenom dolaze nove, što znači da bi automatizacija ovog testa zahtijevala česte promjene i prilagodbe u automatiziranim testovima koje bi oduzele više vremena nego manualna provjera.

Test za provjeru ikona društvenih mreža biti će automatizirani, provjera se sastoji od dohvaćanja ikone za svaku pojedinu društvenu mrežu i provjere je li u pozadini te ikone točan link koji vodi na ispravnu društvenu mrežu i vodi li na profil od fakulteta na toj mreži. Na kraju, testovi prijava studenta na stranicu fakulteta i greška pri prijavi studenta također su izabrani za automatizaciju. Sama prijava u neki sustav ne oduzima puno vremena, ali testiranje svakog mogućeg scenarija prijave u sustav može oduzeti dosta vremena. Na primjer, provjerom prijave sa točnim podacima potvrđuje se je li prijava moguća ako se koriste točni podaci, ali ne govori što se dogodi ako je

korisničko ime ili lozinka pogrešno. Automatizacijom je moguće pokriti sve testne slučajeve vezane za prijavu kako bi se kontinuirano moglo pratiti radi li prijava u sustav kako je zamišljeno. To je posebno važno za softvere koji nemaju vidljive funkciju bez prijave, u tom slučaju, ako prijava ne radi, ništa ne radi.

4.3. Proces automatizacije testova kroz primjer Cypress alata

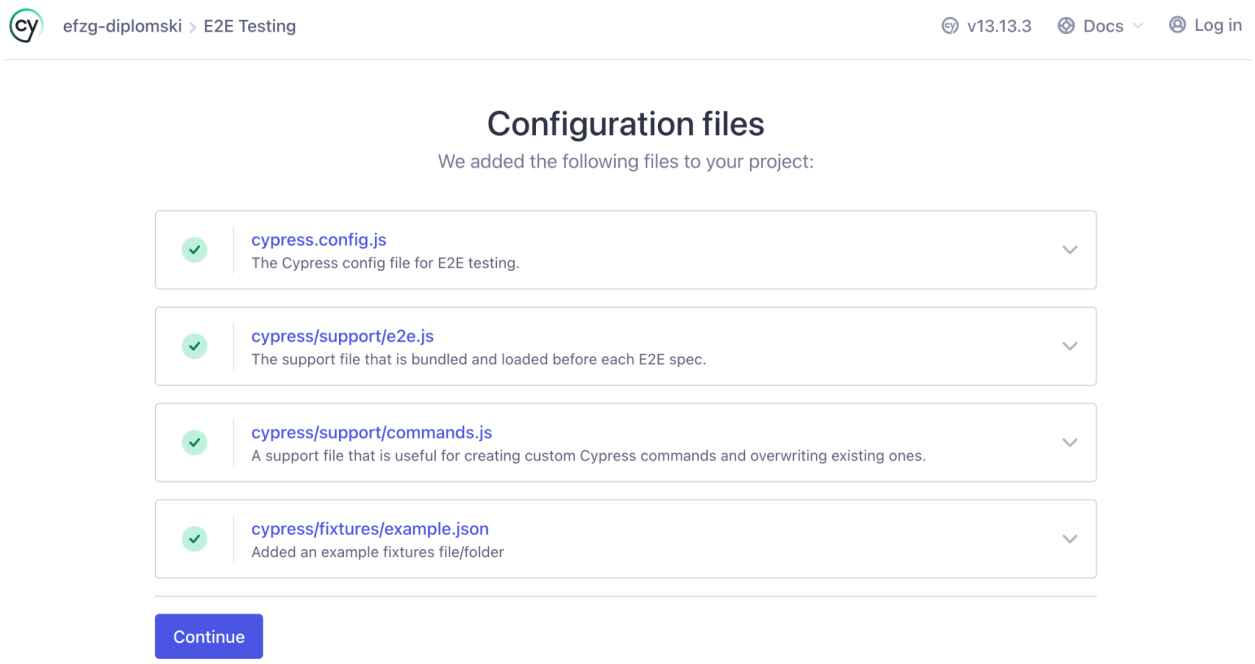
Nakon selekcije testova za automatizaciju, slijedeći korak je proces implementacije testova u Cypress alatu na primjeru web stranice Ekonomskog fakulteta u Zagrebu. Prvi korak je izrada prazne mape negdje na osobnom računalu ili laptopu, to je mapa u kojoj će se nalaziti cijeli Cypress projekt. Drugi korak je instalacija Node.js-a i naredbene linije npm-a, preporuča se instalacija koristeći sljedeću naredbenu liniju u upravitelju, nvm install -g npm. Node.js i npm su temelj za upravljanje potrebnim paketima za pisanje Cypress testova. (npm Docs, 2023)

Treći korak je instalirati i otvoriti Visual Studio Code ili neki drugi program za pisanje i uređivanje koda. U Visual Studio Codu potrebno je otvoriti kreiranu mapu "efzg-diplomski" i pokrenuti nekoliko naredba u terminalu Visual Studio Codea. Prva naredba je "npm init - y", pomoću nje se kreira package.json datoteka u JavaScript projektima koja sadrži ključne informacije o projektu i njegovim ovisnostima. Slijedeće naredbe su instalacija Cypressa u projektu pomoću naredbe "npm install cypress" i otvaranje Cypress sučelja pomoću naredbe "npx cypress open". (Cypress, 2024)

Prvim otvaranjem cypress sučelja za projekt kreirat će se Cypress podmape pod nazivom fixtures, integration, plugins i support, mapa Cypress sadrži sve datoteke povezane sa Cypressom. Mapa fixtures sprema statičke podatke koje korisnici mogu koristiti u testovima, uglavnom za simulaciju mrežnih zahtjeva. Support mapa sadrži pomoćne datoteke koje se izvršavaju prije testnih datoteka, te je odlično mjesto za spremanje naredbi koje se ponavljaju ili globalnih postavki koje utječu na cijeli projekt. (Skadorva, 2023)

Cypress Test Runner sučelje i naredba "npx cypress open" je naredba koja će se koristiti svaki put kada se žele pokrenuti u Cypress sučelju. Na slici 3 vidljivo je prvo pokretanje Cypress sučelja i kreirane datoteke u Cypress projektu. Nakon pregleda kreiranih datoteka i klika na gumb "Continue", potrebno je odabrati vrstu testova koja će se konfigurirati. (Cypress, 2024) Za potrebe

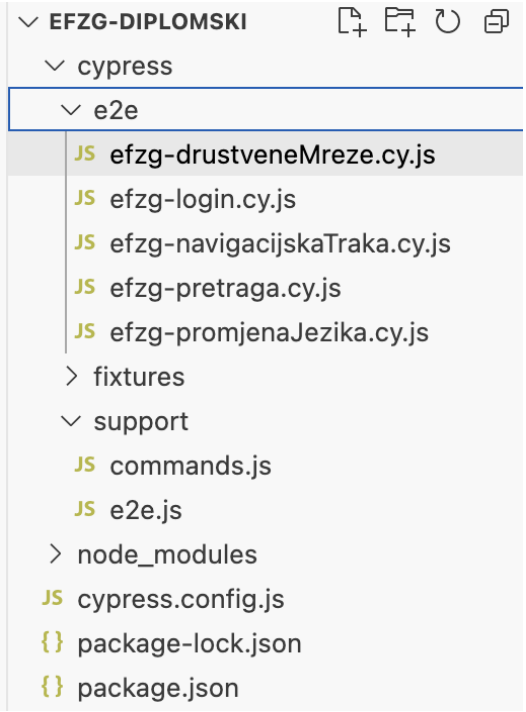
ovog projekta, odabrani su i konfigurirani E2E testovi. Nakon odabira vrste testiranja, Cypress sučelje nudi odabir preglednika u kojem će se izvršavati automatizirani testovi. Cypress nudi sve instaliranje preglednike na uređaju na kojem se pokreće. Zadnji korak prije početka pisanja testova je provjera je li e2e mapa kreirana u projektu, u slučaju da nije, potrebno ju je ručno kreirati u Cypress mapi ili Visual Studio Codeu.



Slika 3 Prvo pokretanje Cypress sučelja

Kreiranje prvog testa počinje kreiranjem nove datoteke u e2e mapi unutar Cypress projekta. Bitno je naglasiti da naziv svake test datoteke mora završavati sa cy.js ekstenzijom. Dio ekstenzije .cy označava da se radi o Cypress "End to End" testovima, dok dio .js označava da je test pisan u JavaScriptu. Mapa e2e je bitna jer je ona zadana lokacija za spremanje e2e testova tako da će Cypress prema svojim zadanim postavkama tražiti e2e testove isključivo u e2e mapi. Isto tako, za testiranje komponenti, umjesto e2e mape potrebno kliknuti na "Components" kod pokretanja sučelja i kreirati components mapu za testove komponenti. (Skadorva, 2023) Mjesto gdje se definiraju sva svojstva Cypress projekta i kako se testovi moraju ponašati nalazi se u datoteci cypress.config.js. (Skadorva, 2023) Na temelju selektiranih testova za automatizaciju u e2e mapi su kreirani slijedeći testovi efzg-društveneMreze.cy.js, efzg-login.cy.js, efzg-

navigacijskaTraka.cy.js, efzg-pretraga.cy.js i efzg-promjenaJezika.cy.js. Nabrojani testovi pregled mape u Visual Studio Code-u vidljivi su na slici 4.



Slika 4 Kreiran projekt efzg-diplomski u Visual Studio Code-u

U Cypress testovi koristi se "describe" za grupiranje srodnih testova kako bi se omogućilo organiziranje testnog seta. Unutar describe-a, koristi se "it", on predstavlja pojedinačni test koji opisuje specifični testni scenarij. (Mocha, 2024) Cypress testovi se sastoje od naredbi, upita i asertacija, odnosno provjere. Najčešća naredba je cy.get i cy.should, cy.get pretražuje programsko sučelje i pronalazi element koji odgovaraju selektoru, dok cy.should provjerava zadanu asertaciju za pronađeni selektor. Na početku testa neke web aplikacije ili web stranice koristi se naredba cy.visit kako bi se navigiralo na stranicu koja se testira. Također česta naredba je cy.contains, ona traži element na stranici koji sadrži određeni tekst. U kombinaciji sa asertacijom cy.should", contains se može koristiti i kao provjera, u tom slučaju bi naredba izgledala ovako, cy.should('contain','tekst') ili cy.should('not.contain','tekst'). (Cypress, 2024)

Test za posjet stranici fakulteta opisan je unutar testa efzg-navigacijskaTraka.cy.js. Kako je prije provjere svake stranice na navigacijskog traci potrebno posjetiti web stranicu fakulteta korištenjem funkcije "prije svakog", odnosno "before each". Na slici 5. Vidljiv je kod korišten za posjet web

stranici fakulteta prije svakog slijedećeg testa i provjera postoji li tekst na web stranici koji glasi "Sveučilište u Zagrebu Ekonomski fakultet".

```
beforeEach(() => {
  cy.visit('https://www.efzg.unizg.hr/');
  cy.contains('Sveučilište u Zagrebu Ekonomski fakultet')
  .should('exist')
});
```

Slika 5 Posjet web stranici EFZG-a i "before each" funkcija

Test jedne od stavki na navigacijskoj traci vidljiv je na slici 6. Može se primijetiti kako test počinje od klikanja po web stranici fakulteta, naredba za posjet stranici nije potrebna radi funkcije "before each". Test klikne na drugu stavku navigacijske trake ako ima tekst "Upisi" nakon čega provjeri url nove stranice, završava li sa "/upisi", ako je test ispunio sve kriterije, smatra se da je test prošao.

```
  it('should click on "Upisi" in the navigation bar and go to the "Upisi"
 subpage', () => {

   cy.get('.menu-list > :nth-child(2) > a')
     .should('have.text', 'Upisi')
     .click()

   cy.url().should('include', '/upisi');

 });
```

Slika 6 Test Navigacijske trake EFZG-a, Studiji

Slika 7 prikazuje test ispravnosti opcije pretraživanja na web stranici ekonomskog fakulteta. Pošto je ovo samo jedan test, nije bilo potrebe za before each funkcijom već test počinje posjetom stranice fakulteta. Slijedeći korak je klik na ikonu za pretraživanje, upisivanje pojma, na primjer "Upisi" i klik na gumb pod nazivom "Pretraživanje" Zadnji korak je provjera, pojavljuju li se rezultati sa ključnom riječju "Upisi" u dijelu stranice predviđenom za prikazivanje rezultata pretrage.

```
describe('Pretraga na web stranici efzg-a', () => {

      it('should be possible to search a term and get results containing the term', ()
=> {

        cy.visit('https://www.efzg.unizg.hr/')
         cy.get('.search-top')
            .click()

        cy.get('.search-input')
            .type('Upisi')

        cy.get('.search-submit')
            .should('have.text','Pretražite')
            .click()

        cy.get('.search_res')
            .should('exist')
            .should('contain.text','upisi')

     });
    });
```

Slika 7 Pretraga na web stranici EFZG-a

Test koji se vrlo često automatizira, ali je za detaljnu provjeru potrebno manualno testiranje je promjena jezika na web stranici. Na slici 8. prikazana je promjene jezika web stranice ekonomskog fakulteta sa hrvatskog na engleski. Zadani jezik web stranice je hrvatski, tako da test prvo provjerava da url zadane stranice ne završava sa "/en". Nakon toga test klikne na oznaku "EN" u gornjem desnom kutu web stranice, odmah nakon klika, url stranice bi se trebao promijeniti na englesku verziju i dobiti nastavak "/en". Kako bi test bio jasniji, u kod su dodani komentari koju pojašnjavaju očekivano ponašanje stranice što pomaže drugim testerima i razvojnom timu da tumače kod. Kako je već spomenuto, ovaj test provjerava samo je li se url promijenio na englesku verziju, ali ne provjerava je li stranica stvarno na engleskom jeziku. Za daljnju provjeru potrebno je izvršiti i manualnu provjeru teksta na stranici.

```
describe('Promjena jezika na engleski na web stranici efzg-a', () => {

   it('should be possible to change the language from croatian to english', () => {

     cy.visit('https://www.efzg.unizg.hr/')

//default language should be croatian
     cy.url().should('not.include', '/en');

     cy.get('.lang > .lang-btn')
       .should('have.text','EN')
       .click()

//en should be added to the url after clicking on the language button
     cy.url().should('include', '/en');

   });

 });
```

Slika 8 Promjena jezika web stranice EFZG-a sa hrvatskog na engleski

Set testova za društvene mreže ekonomskog fakulteta u Zagrebu sastoji se od provjere ikona na stranici fakulteta, te odvedu li korisnika da odgovarajuću društvenu mrežu. Slika 9 prikazuje test za provjeru Facebook ikone. Prvi korak je klik na ikonu Facebooka i provjera je li odgovarajući link povezan sa odgovarajućom ikonom. Na selektoru za Facebook može se primijetiti html element `<a>`, koji najčešće služi za stvaranje veza prema drugim stranicama, bez html elementa u nastavku ne bi bilo moguće dohvatiti link povezan sa Facebook ikonom. Radi toga je test izrađen tako da provjeri je li točan link povezan sa točnom ikonom.

```
describe('drustvene mreze efzg-a', () => {

   it('should open the facebook page from efzg', () => {

     cy.visit('https://www.efzg.unizg.hr/')

     cy.get('.facebook > a')
      .should('have.attr', 'href', 'https://www.facebook.com/EkonomskifakultetZagreb/')

   });
```

Slika 9 Link na Facebook stranicu EFZG-a

Zadnja dva testa koja su odabrana za automatizaciju, iz manualnog testnog koncepta, su prijava studenta na web stranicu fakulteta sa točnim i netočnim podacima. U prvom scenariju, ako student ode na stranicu za prijavu, upiše svoj točan JMBAG i lozinku od AAI identiteta, te klikne na gumb prijava trebao bi pristupiti svojem e-indeksu i zamolbama. Drugim riječima, u gornjem desnom kutu na padajućem izborniku trebalo bi pisati ime i prezime studenta, također, klikom na izbornik student bi trebao imati opcija odjave iz sustava. Opisan scenarij vidljiv je na slici 10, ako se student uspješno prijavi u sustav i piše njegovo ili njezino ime u gornjem desnom kutu test prolazi. Suprotno tome, na slici 10 prikazan je scenarij u kojem student upiše krivu lozinku. Ako student upiše krivu lozinku prijava ne bi smjela biti uspješna i trebala bi se pojaviti greška, ako su ti kriteriji ispunjeni, test se smatra uspješnim.

```
describe('efzg student login', () => {

     it('should login successfully', () => {
        cy.visit('https://student.efzg.hr/login')

       cy.get(':nth-child(1) > .input-group > .form-control')
           .type('JMBAG')

       cy.get(':nth-child(2) > .input-group > .form-control')
           .type('lozinka')

       cy.get(':nth-child(3) > .btn')
           .click()

       cy.get('.nav > .dropdown > .dropdown-toggle')
           .should('contain.text','Ime Prezime')
           .click()

       cy.get('.dropdown-menu > li > a')
           .should('have.text','Logout')
           .click()

     })
```

Slika 10 Prijava studenta na web stranicu EFZG-a

```
    it('should not login successfully', () => {
       cy.visit('https://student.efzg.hr/login')

      cy.get(':nth-child(1) > .input-group > .form-control')
          .type('JMBAG')

      cy.get(':nth-child(2) > .input-group > .form-control')
          .type('pogresnaLozinka')

      cy.get(':nth-child(3) > .btn')
          .click()

      cy.get('.panel-footer > .alert')
          .should('exist')

    })
})
```

Slika 11 Pokušaj prijave studenta na web stranicu EFZG-a sa krivim podacima

4.4. Komparativna analiza manualnih i automatiziranih testova

Na temelju manualnog testnog koncepta u dodatku 1 i Cypress testova koji se nalaze u dodatku 2, napravljena je komparativna analiza manualnog i automatiziranog testiranja. Koje su prednosti i nedostaci na stvarnom primjeru. Testni koncepti u dodacima ilustriraju prednosti i nedostatke oba pristupa kroz konkretne scenarije, kao što su provjera navigacijske trake, promjena jezika, pretraživanje sadržaja te pristup društvenim mrežama. Za manualno klikanje po web stranici fakulteta, da bi se prošli sve raspisani testovi u dodatku 1 potrebno je oko pet minuta, bez detaljne provjeru sadržaja na svakoj posječenoj stranici. Automatizirani testovi u cypressu naprave jednako detaljnu provjeru za manje od dvadeset sekundi.

Manualno testiranje prikazano u dodatku 1 uključuju detaljne korake koje tester mora fizički pratiti. Tester ručno otvara preglednik, upisuje URL i prati da li se stranica ispravno učitava, što mu pruža fleksibilnost i omogućava testerima brzu reakciju na neočekivane promjene ili scenarije. Ovisno o ponašanju aplikacije, ako primijeti potrebu za time, tester može dinamično mijenjati testne korake. Primjer toga je provjera navigacijske trake, ako tester primijeti grešku može lako proširiti ili prilagoditi test bez potrebe za pisanjem koda. Fleksibilnost koju pruža manualno testiranje je ključno za istraživačko testiranje gdje se testni scenarij razvija u hodu, te za testiranje funkcionalnosti koje zahtijevaju subjektivnu procjenu poput korisničkog sučelja i korisničkog iskustva. U primjeru iz dodatka 1, tester bi mogao uočiti greške koje utječu na korisničko iskustvo, greške u dizajnu ili nesukladnosti koje automatizirani testovi ne bi detektirali.

Za razliku od automatiziranih testova, manualno testiranje ne zahtijeva tehničku stručnost u programiranja, što omogućuje većem broju testera sudjelovanje u testiranju. Manualno testiranje je praktično kada se radi o jednostavnim web stranicama ili aplikacijama s ograničenim brojem funkcionalnosti. Na primjer, testiranje osnovnih linkova i navigacijske trake, kao što je prikazano u dodatku 1, može biti brže izvedeno manualno, bez potrebe za kompleksnim postavljanjem automatizacije. Međutim, tester može slučajno preskočiti korak ili netočno interpretirati rezultat testa. Manualno testiranje ovisi o pažnji i koncentraciji testera, što može dovesti do neujednačenih rezultata. Čak i ako tester slijedi precizne korake, kao što je opisano u Dodatku 1, postoji šansa za previd ili subjektivnu interpretaciju rezultata.

Izvještaji manualnih i automatiziranih testova značajno se razlikuju u načinu prikaza informacija i razini detalja koje pružaju. U manualnim testovima, izvještaj se obično sastoji od popunjene tablice kao što je prikazano u Dodatku 1. U manualnom izvještaju bilježi se svaki testni slučaj, koraci svakog testa, očekivano ponašanje softvera, stvarno ponašanje softvera i rezultat testa. Takav način izvještavanja zahtijeva ručni unos podataka, što povećava mogućnost ljudske pogreške i otežava brzu analizu većeg broja testnih slučajeva. S druge strane, kod automatiziranih testova korištenjem alata poput Cypressa, izvještaji se generiraju automatski, a mogu biti dodatno obogaćeni kada se testovi pokreću u "headless" načinu rada, odnosno izvođenje bez grafičkog sučelja. U ovom načinu rada moguće je dobiti detaljne izvještaje u formi koja obuhvaća sve pokrenute testove, uključujući statuse svakog testa, vrijeme izvršavanja, te vizualne prikaze kao što su snimke zaslona i videozapisi. Cypress izvještaj za testove iz dodatka 2 prikazan je na slici 12, gdje se jasno vidi pregled svih testnih slučajeva s oznakama uspješnih i neuspješnih testova. Ovi izvještaji omogućuju bržu analizu rezultata, lakše prepoznavanje problema i efikasnije praćenje napretka u odnosu na manualne izvještaje, gdje su svi podaci organizirani ručno.

(**Run Finished**)

	Spec		Tests	Passing	Failing	Pending	Skipped
✔	efzg-drustveneMreze.cy.js	00:02	5	5	–	–	–
✔	efzg-login.cy.js	00:02	2	2	–	–	–
✔	efzg-navigacijskaTraka.cy.js	00:04	7	7	–	–	–
✔	efzg-pretraga.cy.js	00:02	1	1	–	–	–
✔	efzg-promjenaJezika.cy.js	00:02	1	1	–	–	–
✔	All specs passed!	00:15	16	16	–	–	–

Slika 12 Cypress testni izvještaj

Manualni testovi su vremenski intenzivni, osobito kada se trebaju ponavljati kroz više iteracija ili kada aplikacija raste u složenosti. U velikim projektima, manualno testiranje može brzo postati neodrživo zbog opsega potrebnog rada. Ukupno trajanje testiranja svih navedenih scenarija u dodatku 1 traje oko pet minuta. U složenijim sustavima, ovo vrijeme bi se znatno povećalo. Automatizirani testovi u dodatku 2 koriste Cypress, što omogućava brzo i konzistentno izvršenje testnih scenarija, za koje je potrebno manje od dvadeset sekundi, što je značajna ušteda u odnosu

na 5 minuta potrebnih za manualno testiranje. Veliki projekti s kompleksnim funkcionalnostima mogu imati tisuće testnih scenarija koji bi ručno oduzeli jako puno vremena. Automatizirani alati omogućuju paralelno izvršavanje testova, što značajno smanjuje vrijeme potrebno za dobivanje povratnih informacija.

Automatizirani testovi se mogu pokrenuti u bilo kojem trenutku sa minimalnih ljudskim trudom, a rezultati su uvijek isti, bez obzira na vanjske faktore kao što su umor testera. Međutim, automatizirani testovi imaju druga ograničenja, poput ograničene sposobnosti za detekciju vizualnih i nepredviđenih grešaka**.** Na primjer, ako se tijekom testiranja promjene jezika sadržaj ne prevede ispravno, skripta će ipak smatrati test uspješnim ako URL sadrži "/en". Iako je Cypress kod u primjeru jednostavan radi lakšeg prikaza primjera, pisanje i održavanje automatiziranih testova za kompleksne scenarije zahtijeva tehničku stručnost i može zahtijevati dodatno vrijeme u početnoj fazi.

Iako su u analiziranim primjerima testovi relativno jednostavni, često je u praksi je obujam testiranja mnogo veći. Dok manualno testiranje u ovom primjeru traje oko pet minuta, sve te testove automatizacija može izvršiti u manje od dvadeset sekundi. Na većim testnim setovima, koji mogu sadržavati stotine ili tisuće testnih slučajeva, ušteda vremena postaje ključna. U stvarnim projektima, manualno testiranje može trajati nekoliko dana, dok bi automatizirani testovi omogućili rezultate u samo nekoliko sati ili čak minuta, što može značiti uštedu od nekoliko sati dnevno, što omogućava brže otkrivanje problema, brže povratne informacije za razvojni tim i značajno ubrzanje cijelog procesa razvoja softvera.

Provedeni testovi na web stranici Ekonomskog fakulteta u Zagrebu prošli su uspješno, što ukazuje na stabilnost i ispravnost ključnih funkcionalnosti stranice. Manualni testovi pokazali su da su osnovne navigacijske funkcije, te promjena jezika i pretraga sadržaja, u skladu s očekivanim ponašanjem, te nije uočeno odstupanje između očekivanog i stvarnog ponašanja. Automatizirani testovi, izvedeni pomoću Cypress alata, također su prošli bez grešaka, potvrđujući dosljednost u ponašanju stranice tijekom višestrukih ponavljanja testnih slučajeva. Cypress testovi su brzo potvrdili ispravnost navigacijske trake, funkcionalnost pretrage, promjene jezika, kao i rad linkova na društvene mreže i sustava prijavu studenata. Rezultati manualnih i automatiziranih testova web stranice Ekonomskog fakulteta u Zagrebu pokazali su da su sve navedene funkcionalnosti implementirane ispravno i da se ponašaju prema očekivanjima korisnika.

U konkretnim primjerima testiranja web stranice Ekonomskog fakulteta u Zagrebu, automatizirani testovi pokazuju jasnu prednost u brzini i dosljednosti, dok manualni testovi pružaju fleksibilnost i bolju sposobnost detekcije suptilnih grešaka koje možda nisu očite kroz kod, poput problema u dizajnu, korisničkom iskustvu ili neobičnih ponašanja koja se javljaju u specifičnim situacijama. Automatizirani testovi, mogu se izvršiti unutar nekoliko sekundi i ponavljati bez rizika od ljudske pogreške, što je posebno korisno kod regresijskog testiranja ili kada je potrebno često provoditi testove zbog kontinuiranih promjena na sustavu. Iako svaki pristup ima svoje prednosti i nedostatke, kombinacija oba pristupa često pruža najbolju pokrivenost i kvalitetu testiranja, posebno u složenim sustavima. U praksi, kombiniranjem manualnog i automatiziranog testiranja pokriva se veliki broj raznih testnih slučajeva čime se osigurava visoka razina kvalitete, te donosi najbolje rezultate.

4.5. Preporuke za efikasnu praksu automatizacije testiranja

Kriteriji pokrivenosti su formalni pristupi koji se koriste u testiranju softvera kako bi se odredilo koliko različitih aspekata koda ili funkcionalnosti sustava test obuhvaća. Oni definiraju pravila za odabir testnih scenarija s ciljem povećanja učinkovitosti testiranja i otkrivanja potencijalnih grešaka. Korištenje kriterija pokrivenosti u dizajnu testova donosi brojne prednosti. Tradicionalno testiranje softvera može biti skupo i vremenski zahtjevno, dok formalni kriteriji pokrivenosti pomažu u optimizaciji procesa. Cilj kriterija pokrivenosti je učinkovit odabir testnih scenarija kako bi se osigurao sveobuhvatan skup testova uz minimalno preklapanje, što znači da su razlozi za svaki test unaprijed jasni, a pokrivenost testiranja podržava regresijska testiranja. (Ammann i Offutt, 2017)

Dodatna prednost korištenja kriterija pokrivenosti je što oni definiraju jasne granice kada testiranje može biti završeno. Unaprijed definiran broja testova omogućuje preciznije planiranje troškova i vrijeme testiranja. Također, primjena testnih kriterija lako se može automatizirati, što dodatno poboljšava učinkovitost i točnost testiranja. Uvođenje novih ideja i procesa u testiranje zahtijeva dodatnu obuku za testne timove kako bi usvojili nove postupke i koncepte testiranja. Troškovi tranzicije mogu se smanjiti kroz suradnju testnog i razvojnog tima.(Ammann, 2017)

U cilju optimizacije Cypress testova, važno je slijediti nekoliko ključnih smjernica. Jedan od najvažnijih savjeta uključuje korištenje data-* atributa za selektore umjesto oslanjanja na nestabilne elemente poput tagova ili klasa. Data-* atributi su jedinstvene oznake koje su stabilne i neće se mijenjati zbog promjena u CSS-u ili JavaScriptu, čime se osigurava dosljednost selektora i olakšava održavanje testova. Osim toga, testove treba izvoditi u izolaciji kako bi se spriječila međusobna ovisnost. Svaki test trebao bi imati svoju početnu postavku i ne bi trebao biti povezan sa stanjem drugih testova. Ako se primijeti da testovi ovise jedan o drugome, preporučuje se izmjena testova i premještanje ponovljenog koda u before ili beforeEach hookove. (Cypress, 2024)

Također je važno izbjegavati testiranje dijelova aplikacije koji nisu pod kontrolom korisnika alata za automatizaciju. Kada je potrebno komunicirati s vanjskim uslugama, bolje je koristiti cy.request() umjesto cy.visit() za interakciju s API-jem treće strane, čime se smanjuje složenost i potencijalni problemi povezani s vanjskim serverima. Pri donošenju odluke između korištenja cy.contains() i data-* atributa, korisnici trebaju razmotriti hoće li promjena sadržaja elementa uzrokovati neuspjeh testa. Ako je odgovor da, treba koristiti cy.contains() za ciljanje elemenata prema njihovom sadržaju. Ako ne, data-* atributi su bolji izbor jer su stabilniji i ne ovise o promjenama u tekstualnom sadržaju elemenata. Na kraju, postavljanje baseUrl u Cypress konfiguraciji može značajno pojednostaviti testiranje. Postavljanjem baseUrl, korisnici mogu izbjeći hard-coding URL-ova, omogućujući lakše prebacivanje između različitih okruženja, kao što su razvojno i produkcijsko. Ovo poboljšava fleksibilnost i smanjuje potrebu za promjenom URL-ova unutar testova. (Cypress, 2024)

5. ZAKLJUČAK

Rad pruža sveobuhvatan uvid u usporedbu manualnog i automatiziranog testiranja softvera, s posebnim naglaskom na web aplikacije i njihov životni ciklus. Objašnjene su definicije testiranja i njegova važnost u životnom ciklusu aplikacije. Također kada i za koju vrstu testiranja se najčešće koristi manualno, a kada automatizirano testiranje. Ističe se važnost ranog planiranja procesa testiranja te testiranja zahtjeva aplikacije već od početka razvoja. Rano uključivanje testiranja omogućuje otkrivanje potencijalnih problema u ranoj fazi, kada su troškovi ispravljanja grešaka znatno niži. Planiranje testnih slučajeva paralelno s razvojem pomaže u osiguravanju da su svi funkcionalni i nefunkcionalni zahtjevi pravilno obuhvaćeni. Na taj način, testiranje postaje integralni dio razvoja, a ne samo završna faza, što doprinosi kvalitetnijem softveru i bržem vremenu isporuke.

Analiza o primjeni automatiziranih testova pokazuje kako odluka o primjeni automatiziranih testova ovisi o strukturi i potrebama poduzeća, uz ključne faktore poput dostupnih resursa i vrste projekata. Manja poduzeća preferiraju alate koji se lako integriraju u postojeći sustav, dok veće organizacije biraju fleksibilnije alate s većom pokrivenošću testova. Na tržištu postoji širok spektar alata poput Cypressa, Selenija, Appiuma, Katalon Studija, Cucumbera i Puppeteera, a svaki od njih nudi specifične prednosti ovisno o tehnološkoj strukturi i zahtjevima testiranja. Automatizacija donosi brojne prednosti, poput ubrzanja isporuke softvera i smanjenja troškova, ali zahtijeva kontinuirano održavanje kako bi se zadržala njena učinkovitost.

Usporedna analiza manualnog i automatiziranog testiranja na primjeru web stranice Ekonomskog fakulteta u zagrebu pokazala je da je, manualno testiranje, iako sporije i ovisno o pažnji testera, nudi fleksibilnost i mogućnost dinamične prilagodbe tijekom testiranja. Ono je ključno za istraživačko testiranje i detekciju suptilnih grešaka koje automatizirani alati često ne prepoznaju. Manualno testiranje zahtijeva manje tehničke stručnosti, ali je vremenski intenzivno, osobito u složenijim sustavima s velikim brojem funkcionalnosti. S druge strane, automatizirano testiranje nudi značajnu prednost u brzini i konzistentnosti. Na većim projektima s tisućama testnih slučajeva, ušteda vremena je ključna, što omogućava brže povratne informacije i ubrzanje razvojnog procesa. Iako automatizacija zahtijeva početno ulaganje u tehničku stručnost i održavanje, dugoročno donosi značajne prednosti u konzistentnosti i ponovljivosti testova.

Oba pristupa testiranja, i manualno i automatizirano imaju svoje prednosti i nedostatke, optimalan pristup često uključuje njihovu kombinaciju. U složenim sustavima, kombinacija oba pristupa omogućuje sveobuhvatnu pokrivenost testova i bolju kontrolu kvalitete, čime se osigurava kvalitetniji i brži razvoj softverskih rješenja. Bitnu ulogu ima i održavanje automatiziranih, ali i manualnih testova koje osigurava njihove dugoročne učinkovitosti. Redovito ažuriranje testnih slučajeva, usklađivanje s promjenama u aplikaciji, te praćenje rezultata testiranja ključni su za održavanje visoke kvalitete i pouzdanosti softverskih rješenja. Literatura preporuča korištenje kriterija pokrivenosti koji pomažu u identifikaciji koje testove treba provesti kako bi se maksimalno povećala učinkovitost i točnost testiranja. Oni omogućuju temeljitije pokrivanje svih mogućih scenarija i slučajeva, smanjujući rizik od previđenih grešaka i povećavajući povjerenje u rezultate testiranja. Kontinuirano unapređivanje procesa testiranja omogućuje organizacijama da odgovore na nove izazove i promjene u softverskom okruženju, čime se osigurava dugoročna stabilnost i funkcionalnost aplikacija. Pronalaženje ravnoteže između resursa, budžeta i potreba organizacije ključno je za postizanje optimalnih rezultata i osiguranje visoke kvalitete isporučenih softverskih proizvoda

Literatura

1. Ammann, P., i Offutt, J. (2017), Introduction to software testing (2nd ed.), Cambridge University Press
2. Appium (2024), https://appium.io/ (Pristupano 01. kolovoza 2024)
3. Cucumber (2024), https://cucumber.io/ (Pristupano 01. kolovoza 2024)
4. Cypress (2024), https://www.cypress.io/ (Pristupano 01. kolovoza 2024)
5. Forgacs, I., i Kovacs, A. (2024), Modern software testing techniques: A practical guide for developers and testers, Apress
6. Gelfenbuim, L. (2022), Web testing with Cypress, BPB
7. Graham, D., van Veenendaal, E., Evans, I., i Black, R. (2019), Foundations of software testing: ISTQB certification, Cengage Learning EMEA.
8. Jakupović, Š., i Šuman, A. (2020), Osnove programiranja, Veleučilište u Rijeci
9. Jović, M., i Frid, A. (2022), Procesi programskog inženjerstva, Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva
10. Kaul, N. (2023), Implementing automated software testing, Arcler Press
11. Mocha (2024), https://mochajs.org/ (Pristupano 10. kolovoza 2024)
12. npm Documentation (2024), Downloading and installing Node.js and npm. https://docs.npmjs.com/downloading-and-installing-node-js-and-npm (Pristupano 10. kolovoza 2024)
13. Pezze, M. (2008), Software testing and analysis: Process, principle, and techniques, Wiley India Pvt. Limited
14. Popović, J. (2019), Osnove softverskog inženjerstva, CET Computer Equipment and Trade
15. Puppeteer (2024), https://pptr.dev/ (Pristupano 01. kolovoza 2024)
16. Selenium (2024), https://www.selenium.dev/ (Pristupano 01. kolovoza 2024)
17. Skadorva, V. (2023), Web automation testing with Cypress, Orange Education Pvt Ltd
18. What is a web application (2024), https://aws.amazon.com/what-is/web-application/ (Pristupano 15.srpnja 2024)
19. Yin, Y., i Joang, B. (2023), Embedded software system testing, CRC Press
20. Zaptest (2024), https://www.zaptest.com/hr/ (Pristupano 03. kolovoza 2024)

Popis slika i tablica

Dodatak 1. Testni koncept za manualne testove web stranice Ekonomskog fakulteta u Zagrebu

Testni koncept - Posjet web stranici EFZG-a i navigacijska traka Preglednik: Ime autora: Ime testera: Datum:					
ID	Naziv testa	Koraci	Očekivano ponašanje	Stvarno ponašanje	Rezultati
e-01	Posjet web stranici EFZG-a	1. Otvorite web preglednik 2. U pretraživač upišite https://www.efzg.unizg.hr/ 3. Pritisnite Enter	Pretraživanjem https://www.efzg.unizg.hr/ u pregledniku i pritiskom tipke Enter web stranica EFZG-a bi se trebala otvoriti		
e-02	Navigacijska traka - O nama	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na tekst "O nama" na navigacijskoj traci	Klikom na tekst "O nama" trebala bi se otvoriti stranica sa informacijama o Ekonomskom fakultetu u Zgrebu		
e-03	Navigacijska traka - Upisi	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na tekst "Upisi" na navigacijskoj traci	Klikom na tekst "Upisi" trebala bi se otvoriti stranica sa informacijama o upisima na Ekonomski fakultet u Zagrebu		
e-04	Navigacijska traka - Studiji	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na tekst "Studiji" na navigacijskoj traci	Klikom na tekst "Studiji" trebala bi se otvoriti stranica sa informacijama o studijima koje nudi Ekonomski fakultet u Zagrebu		
e-05	Navigacijska traka - Za studente	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na tekst "Za studente" na navigacijskoj traci	Klikom na tekst "Za studente" trebala bi se otvoriti stranica sa informacija za studente Ekonomskog fakulteta u Zagrebu		
e-06	Navigacijska traka - Katedre	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na tekst "Katedre" na navigacijskoj traci	Klikom na tekst "Katedre" trebala bi se otvoriti stranica sa katedrama na Ekonomskom fakultetu u Zagrebu		

e-07	Navigacijska traka - Istraživački rad	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na tekst "Istraživački rad" na navigacijskoj traci	Klikom na tekst "Istraživački rad" trebala bi se otvoriti stranica sa informacijama o istraživačkim radovima na Ekonomskom fakultetu u Zagrebu		
e-08	Navigacijska traka - Međunarodna suradnja	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na tekst "Međunarodna suradnja" na navigacijskoj traci	Klikom na tekst "Međunarodna suradnja" trebala bi se otvoriti stranica sa informacijama o međunarodnoj suradnji Ekonomskog fakulteta u Zagrebu		
e-09	Pretraživanje	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na znak za pretraživanje 2. Upišite pojam u polje 3. Kliknite na "Pretraživanje"	Ako se upiše pojam u tražilicu i klikne na "Pretraživanje" trebali bi se pojaviti rezultati koji sadrže taj pojam		
e-10	Promjena jezika	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na EN u gornjem desnom kutu	Klikom na EN u gornjem desnom kutu, URL stranice bi trebao dobiti nastavam /en i sadržaj stranice bi trebao biti na engleskom jeziku		
e-11	Provjera obavijesti	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na "Prikaži" pored odlomka Obavijesti	Klikom na Prikaži trebale bi se pojaviti 9 zadnjih obavijesti Ekonomskog fakulteta u Zagrebu, poredane od najnovije do najstarije		
e-12	Link na Facebook	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na ikonu Facebook u gornjem desnom kutu	Klikom na Facebook ikonu trebao bi se otvoriti Facebook profil Ekonomskog fakulteta u Zagrebu		
e-13	Link na Instagram	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na ikonu Instagram u gornjem desnom kutu	Klikom na Instagram ikonu trebao bi se otvoriti Instagram profil Ekonomskog fakulteta u Zagrebu		

e-14	Link na LinkedIn	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na ikonu LinkedIn u gornjem desnom kutu	Klikom na LinkedIn ikonu trebao bi se otvoriti LinkedIn profil Ekonomskog fakulteta u Zagrebu		
e-15	Link na YouTube	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na ikonu YouTube u gornjem desnom kutu	Klikom na YouTube ikonu trebao bi se otvoriti YouTube profil Ekonomskog fakulteta u Zagrebu		
e-16	Link na Viber	Preduvjet: Na web stranici EFZG-a ste 1. Kliknite na ikonu Viber u gornjem desnom kutu	Klikom na Viber ikonu trebao bi se otvoriti Viber profil Ekonomskog fakulteta u Zagrebu		
e-17	Student login	1. Upišite slijedeću adresu u pretraživač: https://student.efzg.hr/login 2. Otvorite web stranicu 3. Upišite točno korisničko ime i lozinku 4. Kliknite na gumb “Prijava”	Upisivanjem točnog korisničkog imena i lozinke, te klikom na gumb “Prijava” korisnik bi trebao biti uspješno ulogiran, te vidjeti svoj profil sa imenom i prezimenom u gornjem desnom kutu.		
e-18	Student login greška	1. Upišite slijedeću adresu u pretraživač: https://student.efzg.hr/login 2. Otvorite web stranicu 3. Upišite netočno korisničko ime i/ili lozinku 4. Kliknite na gumb “Prijava”	Upisivanjem netočnog korisničkog imena i/ili lozinke login ne bi trebao biti uspješan i trebala bi se pojaviti greška		

Dodatak 2. Cypress kod za automatizirane testove

Test navigacijske trake web stranice Ekonomskog fakulteta u Zagrebu

```
describe('Navigacijska traka web stranice efzg-a', () => {

 beforeEach(() => {
   cy.visit('https://www.efzg.unizg.hr/')
   cy.contains('Sveučilište u Zagrebu Ekonomski fakultet')
   .should('exist')
 });

   it('should click on "O nama" in the navigation bar and go to the "O nama"
    subpage', () => {

     cy.get('.menu-list > :nth-child(1) > a')
       .should('have.text','O nama')
       .click()

     cy.url().should('include', '/o-nama');

   });

   it('should click on "Upisi" in the navigation bar and go to the "Upisi" subpage',
    () => {

     cy.get('.menu-list > :nth-child(2) > a')
       .should('have.text', 'Upisi')
       .click()

     cy.url().should('include', '/upisi');

   });

   it('should click on "Studiji" in the navigation bar and go to the "Studiji"
    subpage', () => {

     cy.get('.menu-list > :nth-child(3) > a')
       .should('have.text','Studiji')
       .click()

     cy.url().should('include', '/studiji');

   });
```

```
it('should click on "Za studente" in the navigation bar and go to the "Za studente" subpage', () => {

  cy.get('.menu-list > :nth-child(4) > a')
    .should('have.text', 'Za studente')
    .click()

  cy.url().should('include', '/za-studente');

});

it('should click on "Studiji" in the navigation bar and go to the "Katedre" subpage', () => {

  cy.get('.menu-list > :nth-child(5) > a')
    .should('have.text', 'Katedre')
    .click()

  cy.url().should('include', '/katedre');

});

it('should click on "Istrazivacki rad" in the navigation bar and go to the "Istrazivacki rad" subpage', () => {

  cy.get('.menu-list > :nth-child(6) > a')
    .should('have.text', 'Istraživački rad')
    .click()

  cy.url().should('include', '/istrazivacki-rad');

});

it('should click on "Medjunarodna suradnja" in the navigation bar and go to the "Medjunarodna suradnja" subpage', () => {

  cy.get('.menu-list > :nth-child(7) > a')
    .should('have.text','Međunarodna suradnja')
    .click()

  cy.url().should('include', '/medjunarodna-suradnja');
```

```
        });

      });
```

Test pretrage na web stranici Ekonomskog fakulteta u Zagrebu

```
describe('Pretraga na web stranici efzg-a', () => {

     it('should be possible to search a term and get results containing the term', () =>
{

       cy.visit('https://www.efzg.unizg.hr/')
        cy.get('.search-top')
           .click()

       cy.get('.search-input')
           .type('Upisi')

       cy.get('.search-submit')
           .should('have.text','Pretražite')
           .click()

       cy.get('.search_res')
           .should('exist')
           .should('contain.text','upisi')

     });
    });
```

Test promjene jezika na web stranici Ekonomskog fakulteta u Zagrebu

```
describe('Promjena jezika na engleski na web stranici efzg-a', () => {

   it('should be possible to change the language from croatian to english', () => {

     cy.visit('https://www.efzg.unizg.hr/')

//default language should be croatian
     cy.url().should('not.include', '/en');

     cy.get('.lang > .lang-btn')
       .should('have.text','EN')
```

```
        .click()

//en should be added to the url after clicking on the language button
     cy.url().should('include', '/en');

   });

 });
```

Test linkova za društvene mreže na web stranici Ekonomskog fakulteta u Zagrebu

```
describe('drustvene mreze efzg-a', () => {

   it('should open the facebook page from efzg', () => {

     cy.visit('https://www.efzg.unizg.hr/')

     cy.get('.facebook > a')
      .should('have.attr', 'href', 'https://www.facebook.com/EkonomskifakultetZagreb/')

   });


   it('should open the instagram page from efzg', () => {

       cy.visit('https://www.efzg.unizg.hr/')
        cy.get('.instagram > a')
         .should('have.attr', 'href', 'https://www.instagram.com/unizg_efzg/')

     });

     it('should open the linkedin page from efzg', () => {

       cy.visit('https://www.efzg.unizg.hr/')
        cy.get('.linkedin > a')
         .should('have.attr', 'href', 'https://www.linkedin.com/company/feb-unizg')

     });

     it('should open the youtube page from efzg', () => {

       cy.visit('https://www.efzg.unizg.hr/')
        cy.get('.youtube > a')
```

```
        .should('have.attr', 'href', 'https://www.youtube.com/channel/UC5GdZj5gL6br0-jxOfN4pog/')
      });

     it('should open the viber page from efzg', () => {

       cy.visit('https://www.efzg.unizg.hr/')
        cy.get('.viber > a')
       .should('have.attr', 'href', 'https://invite.viber.com/?g2=AQAnWLnCSRtlBkszgOLNnPl87bCjy70bvNnvtmnO%2FlVwgU6QL907U4vNQXhsevCs')
      });

 });
```

Test prijave studenta na web stranicu Ekonomskog fakulteta u Zagrebu

```
describe('efzg student login', () => {

     it('should login successfully', () => {
        cy.visit('https://student.efzg.hr/login')

       cy.get(':nth-child(1) > .input-group > .form-control')
           .type('JMBAG')

       cy.get(':nth-child(2) > .input-group > .form-control')
           .type('lozinka')

       cy.get(':nth-child(3) > .btn')
           .click()

       cy.get('.nav > .dropdown > .dropdown-toggle')
           .should('contain.text','Ime Prezime')
           .click()

       cy.get('.dropdown-menu > li > a')
           .should('have.text','Logout')
           .click()

     })

     it('should not login successfully', () => {
        cy.visit('https://student.efzg.hr/login')
```

```
    cy.get(':nth-child(1) > .input-group > .form-control')
        .type('JMBAG')

    cy.get(':nth-child(2) > .input-group > .form-control')
        .type('pogresnaLozinka')

    cy.get(':nth-child(3) > .btn')
        .click()

    cy.get('.panel-footer > .alert')
        .should('exist')

  })
})
```

Životopis

Osobne informacije

Tea Cerar
Datum rođenja:13/09/1993
Mjesto rođenja: Zagreb
Spol: žensko
Državljanstvo: hrvatsko
Email:teacerar9@gmail.com
Mobitel: +385 918919998

Obrazovanje i osposobljavanje

2022-2024 Ekonomski fakultet u Zagrebu, Specijalistički diplomski stručni studij Elektroničko poslovanje u privatnom i javnom sektoru
2017-2022 Ekonomski fakultet u Zagrebu, preddiplomski stručni studij Trgovinsko poslovanje
2016-2017 Lern Academy Pirmasens
2008-2012 Srednja škola Jastrebarsko

Jezične vještine

Engleski: Razumijevanje C2, Govor C2, Pisanje C2
Njemački Razumijevanje B2, Govor B1, Pisanje B1
Talijanski Razumijevanje A1, Govor A1, Pisanje A1

Digitalne vještine

MS Office, JIRA/Confluence, Service Desk Management Git (GitHub), Cypress, JavaScript

Radno iskustvo

2022-trenutno Axion solutions, Quality assurance
2021-2022 Assist Digital, Korisnička podrška na engleskom i njemačkom jeziku