Važnost osiguravanja kvalitete softvera manualnim i automatiziranim testiranjem

Viljevac, Josipa

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:066090

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-30**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika

VAŽNOST OSIGURAVANJA KVALITETE SOFTVERA MANUALNIM I AUTOMATIZIRANIM TESTIRANJEM

Diplomski rad

Josipa Viljevac

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 20.09.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Josipa Viljevac
Studij, smjer:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. studenta, godina upisa:	D-1088, 04.10.2017.
OIB studenta:	98256065109
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Dr. sc. Hrvoje Leventić
Sumentor iz tvrtke:	Anja Tomašić
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	Izv. prof. dr. sc. Irena Galić
Član Povjerenstva 2:	Dr. sc. Krešimir Romić
Naslov diplomskog rada:	Važnost osiguravanja kvalitete softvera manualnim i automatiziranim testiranjem
Znanstvena grana rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Ideja je predstaviti faze, metode, proces i tehnike manualnog testiranja, proces automatizacije, vrste automatiziranih testova te alate korištene pri automatizaciji. Jedan od alata koji se koristi pri automatizaciji testova jest i cypress, na kome će biti naglasak u ovom radu. U sklopu istog će biti prikazan i postupak kreiranja automatiziranih testova.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	20.09.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 28.09.2021.

Ime i prezime studenta:	Josipa Viljevac
Studij:	Diplomski sveučilišni studij Elektrotehnika, smjer Komunikacije i informatika'
Mat. br. studenta, godina upisa:	D-1088, 04.10.2017.
Turnitin podudaranje [%]:	9

Ovom izjavom izjavljujem da je rad pod nazivom**: Važnost osiguravanja kvalitete softvera manualnim i automatiziranim testiranjem**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Dr. sc. Hrvoje Leventić

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

Potpis studenta:

Sadržaj

1. UVOD

Testiranje softvera je proces kritičke analize kako bi se identificiralo i procijenilo ispunjava li razvijena aplikacija specifikacije poslovnih zahtjeva [1]. Osim što nastoji dostaviti softver koji ne sadrži greške, testiranje pomaže unaprijediti funkcionalnost i upotrebljivost aplikacije. Postoje različite vrste, metode i tehnike pri testiranju softvera, a uključuju više razina njegove verifikacije i validacije. Zadatak ovog diplomskog rada jest predstaviti faze, metode, proces i tehnike ručnog testiranja, proces automatizacije, vrste automatiziranih testova te alate korištene pri automatizaciji. Jedan od alata koji se koristi pri automatizaciji testova jest Cypress. U ovom poglavlju navedeni su razlozi zbog kojih je važno testirati softver te je navedeno nekoliko primjera softverskih propusta koji su se dogodili u nedavnoj prošlosti. U drugom poglavlju objašnjeno je što je ručno testiranje te su navedena načela testiranja, definirana od strane ISTQB organizacije, a koja poboljšavaju učinkovitost i usredotočenost testera pri testiranju. Također, navedene su i opisane razine testiranja, vrste ručnog testiranja, proces i vrste automatizacije. U trećem poglavlju navedeni su neki od alata koji se koriste pri automatizaciji testova. U četvrtom poglavlju prikazan je postupak kreiranja automatiziranih testova za *Conduit* testnu aplikaciju koja se može koristiti za vježbanje pisanja automatiziranih testova.

1.1. Važnost testiranja softvera

Postoje razlozi koji govore zašto je testiranje softvera važno i koje su glavne stvari koje bi se pri tome trebale uzeti u obzir. Potrebno je testirati softver kako bi se ukazalo na nedostatke i pogreške nastale tijekom razvojnih faza [2]. Primjerice, programeri mogu pogriješiti tijekom implementacije softvera. Razlozi za to mogu biti mnogi: nedostatak iskustva programera, nedostatak znanja o programskom jeziku, površno definirani zahtjevi od strane klijenta, pogrešna implementacija algoritma zbog složene logike ili jednostavno ljudska pogreška. Primarni cilj vlasnika proizvoda je pružanje zadovoljstva krajnjim korisnicima. Testiranjem softvera osigurava se to da korisnik organizaciju smatra pouzdanom i da se zadrži njegovo zadovoljstvo aplikacijom. Razlozi zbog kojih je potrebno odlučiti se za testiranje softvera je činjenica da je ono zaslužno za dobre povratne informacije korisnika čime se stječe ugled pouzdanog klijenta. Ako korisnik organizaciju ne smatra pouzdanom ili nije zadovoljan kvalitetom isporuke proizvoda, tada može pronaći drugu, konkurentnu organizaciju s kojom će sklopiti suradnju. Ponekad ugovori mogu uključivati i novčane kazne s obzirom na vremenski rok i kvalitetu proizvoda. U takvim slučajevima, ako se softver pravilno testira,

novčani gubici mogu biti spriječeni. Testiranjem softvera poboljšava se njegova kvaliteta, otkrivaju se problemi koji se pravovremeno mogu riješiti i time omogućiti isporuku kvalitetnog proizvoda na tržište u traženom roku. Isporuka kvalitetnih proizvoda na vrijeme gradi povjerenje korisnika u tim i organizaciju. Testiranje je potrebno za učinkovito funkcioniranje softverske aplikacije. Važno je osigurati da aplikacija ne rezultira pogreškama jer njihovo popravljanje može biti skupo u kasnijim fazama razvoja. Razvoj softvera sastoji se od mnogih faza, a ako se pogreške pronađu u ranijim fazama, trošak njihovog uklanjanja je puno manji. Zato je važno uvesti testiranje što prije u procesu razvoja softvera. Ako se greške povezane sa zahtjevima ili dizajnom otkriju u kasnoj fazi razvoja softvera, može biti vrlo skupo otkloniti ih, jer bi to moglo zahtijevati redizajn, ponovnu implementaciju i ponovno testiranje aplikacije. Korisnici nisu skloni koristiti softver koji sadrži pogreške. Ako korisnici nisu zadovoljni stabilnošću aplikacije, neće ju nastaviti koristiti. Testiranje softvera je neophodno jer se pogreške svima događaju, neke pogreške su nevažne, a neke skupe i/ili opasne. U nastavku ovog poglavlja navedeni su primjeri softverskih propusta koji su se dogodili u nedavnoj prošlosti, a koji su objavljeni u članku digitalnog časopisa *Computerworld* [4].

Hitne službe u sedam Sjedinjenih Američkih Država u travnju 2019. godine nisu bile dostupne šest sati zbog incidenta koji je zahvatio 81 pozivni centar. Oko 6000 ljudi nije moglo uspostaviti poziv prema broju 911 u Washingtonu i dijelovima šest drugih država. Savezno povjerenstvo za komunikacije (engl. *Federal Communications Commission*) je otkrilo da je za pad usluge odgovorna softverska pogreška koja se lako mogla spriječiti.

Iako najprofitabilnija tvrtka, Apple je u rujnu 2014. doživio neugodnu situaciju kada je ažuriranje novog operativnog sustava iOS 8 moralo biti povučeno samo sat vremena nakon njegova izdanja. Korisnici su se žalili na izgubljeni telefonski signal, zaustavljena ažuriranja i probleme s otključavanjem uređaja.

U periodu od 2018. do 2020. godine Nissan je morao opozvati preko milijun automobila zbog kvara softvera koji je nastao na senzornim detektorima zračnog jastuka. Ukratko, automobili koji su bili zahvaćeni softverskom pogreškom nisu mogli detektirati je li odrasla osoba sjedila na suvozačevom mjestu i kao rezultat toga zračni jastuci se nisu aktivirali. Prijavljena su dva slučaja nesreće u kojima se zračni jastuci nisu aktivirali zbog tog softverskog kvara.

Sredinom siječnja 2016. Nest „pametni“ termostat (u vlasništvu Google-a) bio je pogođen softverskom greškom koja je korisnike doslovno ostavila „u hladnom“. Ažuriranje softvera pošlo je

po zlu tako što su se baterije uređaja ugasile te onesposobile termostat za kontrolu temperature. Iz tog razloga korisnici nisu mogli zagrijati svoje domove ili dobiti toplu vodu u jednom od najhladnijih vikenda u godini do tad.

U veljači 2014. Toyota je morala opozvati gotovo 2 milijuna hibridnih vozila Prius kako bi popravila softverski kvar sa svojom upravljačkom jedinicom motora. Kvar je mogao uzrokovati pregrijavanje tranzistora, što bi automobil moglo dovesti u nesiguran način upravljanja i potencijalno uzrokovati gašenje hibridnog sustava tijekom vožnje. To je problem koji se ponavlja, isti softverski problemi doveli su do toga da je Toyota opozvala dodatnih 625000 vozila u srpnju 2015. godine.

2. METODE TESTIRANJA SOFTVERA

2.1. Ručno testiranje

Testiranje je ključan korak u razvoju proizvoda kako bi se osigurala njihova kvaliteta [5]. Važno je testirati aplikacije koje se izrađuju kako bismo se uvjerili da rade ispravno, u suprotnom, manja je vjerojatnost da će ih korisnici kupiti ili nastaviti koristiti. Ručno testiranje je proces testiranja softvera kako bi se identificirale greške u aplikaciji. Ono zahtjeva više uloženog truda, ali je neophodno kako bi se provjerila potencijalna izvedivost automatizacije. Naravno, nije moguće u potpunosti automatizirati testiranje, što znači da je ručno testiranje neophodno jer se svaka nova aplikacija mora ručno testirati. Koncepti ručnog testiranja ne zahtijevaju poznavanje ikakvih alata za testiranje [6]. Ručno testiranje je aktivnost gdje tester treba biti strpljiv, kreativan i otvorenog uma. Uz to, ono je bitan dio razvoja softvera prilagođeno korisniku jer su ljudi ti koji su uključeni u testiranje softverskih aplikacija, a i krajnji korisnici su također ljudi. Glavni cilj ručnog testiranja jest osigurati da aplikacija nema grešaka i da radi u skladu s navedenim funkcionalnim zahtjevima. U osnovi, testiranjem se provjerava kvaliteta sustava te se klijentu dostavlja proizvod koji ne sadrži greške.

Testiranje je presudan element razvoja softvera koje, kada se pravilno strukturira, predstavlja složenu aktivnost kojom se postiže maksimalna učinkovitost. Zbog te složenosti uvijek je korisno pregledati postupke i smjernice kako bi se osiguralo slijeđenje najbolje prakse. Neprofitabilna organizacija *International Software Testing Qualifications Board* (ISTQB) odgovorna je za definiranje različitih smjernica poput strukturnog ispitivanja, akreditiranja, certificiranja i slično [8]. Ova organizacija navodi sedam temeljnih načela testiranja [7]:

1. Testiranje pokazuje prisutnost pogrešaka

Testiranje softvera služi kako bismo otkrili pogreške koje se moraju ispraviti. Testiranje uvelike pomaže smanjiti broj neotkrivenih pogrešaka skrivenih u softveru, ali pronalaženje i rješavanje tih pogrešaka nije dokaz da softver više nema problema. Takav koncept timovi trebaju prihvatiti. Važno je imati na umu da, iako testiranje pokazuje prisutnost grešaka, a ne njihovu odsutnost, temeljito testiranje dat će svima osjećaj pouzdanosti u softver.

2. Iscrpno testiranje nije moguće

Koliko god je poželjno, nemoguće je testirati apsolutno sve. Međutim, jedna od vještina testiranja je procjena rizika i planiranje testova oko njih, pri čemu se mogu pokriti velika područja, istovremeno osiguravajući da se testiraju najvažnije funkcije. Uz pažljivo planiranje i procjenu, pokrivenost testovima može biti izvrsno i osnažiti potrebno povjerenje u softver, bez potrebe za testiranjem svakog retka koda.

3. Rano testiranje

Rano testiranje je ključno u životnom ciklusu softvera. Uključivanje testiranja u ranoj fazi razvoja softvera je temeljni agilni princip koji na testiranje gleda kao na aktivnost tijekom cijelog životnog ciklusa razvoja softvera, a ne kao jednu fazu u tom ciklusu. Kada tim naiđe na prepreke, rane povratne informacije jedan su od najboljih načina da se one prevladaju, a testeri su za to neophodni. U osnovi, rano testiranje može čak pomoći spriječiti pogreške.

4. Grupiranje pogrešaka

Načelo grupiranja pogrešaka govori o tome da određene komponente ili moduli softvera obično sadrže najveći broj pogrešaka ili su odgovorni za većinu problema. Upravo iz tog razloga bi se testiranje trebalo fokusirati na te komponente ili module. Ovdje se može primijeniti Paretovo načelo, a to je da 80% pogrešaka nastaje zbog 20% koda. To je osobito slučaj s velikim i složenim sustavima, ali gustoća kvarova može varirati iz različitih razloga.

5. „Paradoks pesticida“

Ovo načelo temelji se na teoriji da ako se u više navrata koriste pesticidi na usjevima, kukci će na kraju izgraditi imunitet na pesticide te će oni biti neučinkoviti. Slično je i s testiranjem, ako se koriste jedni te isti testovi, iako bi oni mogli potvrditi da softver dobro radi, na kraju neće uspjeti pronaći nove probleme. Važno je neprestano pregledavati testove i mijenjati ih ili dodavati nove testne slučajeve kako bi se izbjegao „paradoks pesticida“.

6. Testiranje ovisi o kontekstu

Metode i vrste testiranja koje se provode mogu u potpunosti ovisiti o kontekstu softvera ili sustava. Ono što se testira uvijek će utjecati na pristup daljnjem testiranju.

7. Odsutnost pogrešaka je zabluda

Ako je softver neupotrebljiv ili ne ispunjava želje korisnika, onda nije važno koliko kvarova je pronađeno i otklonjeno, on je i dalje neupotrebljiv. Bez obzira na to što sustav ima mali broj programskih pogrešaka, ako je upotrebljivost tako loša, sustav ne zadovoljava poslovne zahtjeve te je neuspješan. Važno je izvršavati testove koji su relevantni zahtjevima sustava. Također, softver bi se trebao testirati uz pomoć korisnika kako bi se prikupile povratne informacije koje se mogu koristiti za osiguravanje i poboljšanje upotrebljivosti. Valja imati na umu da iako postoji mali broj problema, to ne znači da je softver spreman za isporuku. Ispunjavanje očekivanja i zahtjeva klijenta jednako je važno kao i osiguravanje kvalitete.

Primjenom navedenih principa testiranja može se poboljšati učinkovitost i usredotočenost testera pri testiranju, a time i cjelokupna kvaliteta strategije testiranja.

2.2. Razine testiranja

Postoje četiri glavne razine testiranja: testiranje komponente, integracijsko testiranje, testiranje sustava te provjera prihvatljivosti [9].

Testiranje komponente definirano je kao razina testiranja softvera u kojoj se testiranje provodi za svaku individualnu komponentu, bez ikakve interakcije s drugim komponentama [10]. Naziva se još i testiranje jedinica, programa ili modula. Generalno, bilo koji softver sastoji se od više komponenti, a razina testiranja komponente bavi se testiranjem tih komponenti zasebno.

Nakon što su pojedinačne komponente istestirane, one se integriraju u jednu cjelinu i testiraju kao grupa, pri čemu se testira interakcija između pojedinih komponenti. Cilj ovakvog testiranja jest otkriti greške u međudjelovanju integriranih cjelina [11].

Nakon integracijskog testiranja slijedi testiranje sustava koje se vrši kako bi se procijenila cjelokupna usklađenost softvera s poslovnim, funkcionalnim te korisničkim zahtjevima. Testiranje sustava može se opisati kao slijed navedenih akcija: stvaranje testnih planova i testnih slučajeva, odabir i stvaranje testnih podataka za testiranje sustava, izvršavanje testnih slučajeva, ispravljanje programskih pogrešaka, regresijsko testiranje i ponavljanje aktivnosti testnog ciklusa [9]. Testiranje sustava uključuje funkcionalne i nefunkcionalne zahtjeve kao što su izvođenje, uporabljivost i sigurnost. Okruženje u kojem se provodi testiranje sustava treba biti što sličnije produkcijskom okruženju kako bi rizik pronalaska pogrešaka karakterističnih za takva okruženja bio što manji.

Posljednja razina jest provjera prihvatljivosti gdje se mora procijeniti je li sustav spreman za proizvodnju i upotrebu od strane krajnjih korisnika. Cilj provjere prihvatljivosti jest korisnicima uliti povjerenje u sustav. Testiranje se provodi na okruženju koje je slično produkcijskom. Postoji pet vrsti provjere prihvatljivosti [12], prva jest ona od strane korisnika gdje se provjerava je li sustav prikladan za korisnikovo upravljanje. Druga vrsta je testiranje operativne prihvatljivosti. Primarni cilj testiranja operativne prihvatljivosti jest osiguravati sigurnosne kopije i oporavak sustava, te da zadaci sigurnosnog učitavanja i migracije podataka funkcioniraju u skladu s poslovnim zahtjevima. Treća vrsta jest ugovorna provjera prihvatljivosti. Ispitivanje prihvaćanja ugovora znači da se razvijeni softver testira prema određenim kriterijima i specifikacijama koji su unaprijed definirani i dogovoreni ugovorom. Kada se projektni tim složi oko samog ugovora, definiraju se relevantni kriteriji i specifikacije za prihvaćanje istog. Četvrta vrsta ispitivanja prihvatljivosti jest ispitivanje prihvaćanja propisa koje ispituje je li softver u skladu s vladinim i pravnim propisima. Posljednja vrsta provjere prihvatljivosti su alfa i beta testiranje. Alfa testiranje se provodi kako bi se identificirali svi mogući problemi i programske pogreške prije objavljivanja konačnog proizvoda krajnjim korisnicima [13]. Alfa testiranje provode testeri koji su interni zaposlenici organizacije. Glavni cilj je identificirati zadatke koje bi korisnik mogao izvesti i testirati ih. Ova vrsta testiranja naziva se alfa samo zato što se provodi rano, pred kraj razvoja softvera i prije beta testiranja. Glavni fokus alfa testiranja je simuliranje stvarnih korisnika. Beta testiranje provode „pravi korisnici“ softverske aplikacije u „pravom okruženju“ i može se smatrati oblikom vanjske provjere prihvatljivosti. Izravna povratna informacija kupaca glavna je prednost beta testiranja. Beta verzija softvera objavljena je ograničenom broju krajnjih korisnika proizvoda radi dobivanja povratnih informacija o kvaliteti proizvoda. Beta testiranje smanjuje rizike neuspjeha proizvoda i osigurava povećanu kvalitetu provjerom valjanosti od strane kupaca.

2.3. Vrste ručnog testiranja

Vrste testiranja razlikuju se po ciljevima testiranja. Postoje četiri glavne vrste testiranja: funkcionalno, nefunkcionalno, strukturno te regresijsko i ponovno testiranje [9].

Funkcionalno testiranje može se definirati kao testiranje svih značajki i funkcionalnosti softvera kako bi se osigurali svi zahtjevi i specifikacije dane u dokumentaciji koja sadrži poslovne zahtjeve. Funkcionalno testiranje smatra se testiranjem crne kutije (engl. *Black Box testing*) jer se softver testira na temelju funkcionalnih zahtjeva. Ovdje se ne razmatra unutarnja struktura softvera, stoga testeri

uglavnom ne trebaju razumjeti kodiranje softvera. Međutim, trebaju razumjeti funkcionalne zahtjeve za koje se očekuje da će testirani sustav zadovoljiti. Funkcionalnim testiranjem provjerava se ponaša li se sustav kako je predviđeno. Ono se može provoditi na bilo kojoj razini jedinice, integracije i sustava. Funkcionalno testiranje uglavnom uključuje: identificiranje funkcionalnosti koje se trebaju testirati, pripremanje testnih podataka, dokumentiranje očekivanih rezultata, provođenje testnih slučajeva, uspoređivanje stvarnih rezultata s očekivanim rezultatima te prijavljivanje razlika između njih ako postoje.

Nefunkcionalni zahtjevi često se nazivaju kvalitetama sustava. Oni uključuju sve zahtjeve za ponašanjem koji nisu definirani u funkcionalnim zahtjevima, poput performansi i stabilnosti aplikacije. Ovi zahtjevi nisu izravno povezani s poslovanjem, ali su ključni za održavanje sustava. Funkcionalno testiranje može se provoditi na svim razinama. Primjer funkcionalnog zahtjeva za brisanjem korisnika jest takvo brisanje podataka da ono više nije vidljivo kroz sučelje. Dok nefunkcionalno brisanje korisnika može značiti „prividno brisanje“ podataka. Ako je podatak „prividno obrisan“, te se ponovno pokuša dodati taj podatak, bit će prikazana poruka da podatak koji se želi dodati već postoji.

Testiranje performansi je vrsta nefunkcionalnog testiranja koja se koristi za utvrđivanje ponašanja softvera u različitim uvjetima. Cilj je testirati stabilnost softvera u stvarnim korisničkim situacijama. Testiranje performansi može se podijeliti na četiri vrste [14]:

1. *Load testing* je postupak stavljanja sve veće količine simulirane potražnje za softverom, kako bi se provjerilo može li se softver nositi s onim za što je dizajniran.

2. *Stress testing* je postupak kojim se utvrđuje kako će softver reagirati na opterećenja koja su iznad njegova maksimuma koji može podnijeti. Cilj testiranja je namjerno preopteretiti softver dok se ne pokvari primjenom realnih i nerealnih scenarija opterećenja.

3. *Endurance testing* je testiranje izdržljivosti koje se koristi za analizu ponašanja aplikacije pod određenom količinom simuliranog opterećenja tijekom duljeg vremenskog razdoblja. Cilj je razumjeti kako će se softver ponašati u kontinuiranoj upotrebi. Testiranje izdržljivosti pomaže otkriti „curenje memorije“ (engl. *memory leak*).

4. *Spike testing* je vrsta testiranja koja se koristi za određivanje načina na koji će softver reagirati na znatno veće navale istodobnih aktivnosti korisnika ili sustava tijekom različitih vremenskih razdoblja.

U idealnom slučaju, ovakvo testiranje pomaže u shvaćanju što će se dogoditi kad se opterećenje iznenada i drastično poveća.

Strukturno testiranje naziva se i testiranje bijele kutije (engl. *White Box testing*) pri čemu testeri moraju biti upoznati s implementacijom softvera [9]. Prilikom strukturnog testiranja provjerava se kako kod u softveru funkcionira, primjerice kako funkcioniraju petlje u softveru. Strukturno testiranje je proces koji je podijeljen u faze i zahtjeva rad na testnim podacima, izvršavanju pripremljenih testnih slučajeva, konačnoj evaluaciji testnih rezultata, što u konačnici dovodi do strukturnog testnog izvještaja.

Ponovno testiranje je testiranje iste funkcionalnosti s različitim skupovima ulaznih podataka. U slučajevima kada tester pronađe grešku, developer ju ispravi, nakon čega tester ponovno testira funkcionalnost. Postupak ponovnog testiranja može se nazivati i „potvrdom“, što znači da je pogreška ispravljena. Regresijsko testiranje uključuje ponovno izvršavanje testnih slučajeva kad god nastanu nove promjene u kodu, bile to strukturne ili funkcionalne promjene, kako bi se osiguralo da pri tome nisu nastale nove pogreške. Testni slučajevi odabrani za regresijsko testiranje moraju biti ponovljivi. U većini slučajeva, regresijsko testiranje provodi se pomoću automatiziranih testova, što testerima omogućuje povećanje pokrivenosti testova tijekom regresijskog testiranja. I ponovno testiranje i regresijsko testiranje može se izvoditi na svim razinama i mogu uključivati i funkcionalne i nefunkcionalne ciljeve testiranja. U većini slučajeva regresijsko testiranje zahtjeva više truda nego ponovno testiranje, i ponovno testiranje gotovo slijedi regresijsko testiranje. Za bolje shvaćanje razlike između ponovnog i regresijskog testiranja može se razmotriti sljedeći primjer: postoji 100 testnih slučajeva koji se trebaju provesti za neki sustav. Nakon provedbe testnih slučajeva, 90 ih je bilo uspješno, a 10 neuspješno. Nakon što su pogreške otklonjene, tester prvo provodi ponovno testiranje, dakle izvršava one testne slučajeve koji su u prvoj provedbi dali neuspješne rezultate. Nakon toga se provodi regresijsko testiranje preostalih 90 slučajeva kako bi se potvrdilo da popravljanje prethodno neuspješnih 10 testova nije rezultiralo pojavom pogrešaka u već uspješno provedenim testovima.

2.4. Automatizirano testiranje

Automatizirano testiranje može se definirati kao način neprestanog izvođenja niza testova iznova bez potrebe za ručnim izvršavanjem [15]. Cilj automatiziranog testiranja je smanjiti broj testnih slučajeva koji se trebaju ručno izvoditi, a ne eliminirati ručno testiranje [16]. Automatizirano testiranje je investicija koja dugoročno štedi vrijeme i novac. Ono ne zahtijeva ljudsku intervenciju zbog čega

se testovi mogu pokrenuti i izvršavati bez nadzora. Automatizacijom testova osigurava se povećanje učinkovitosti, pokrivenost softvera testovima i brzina izvođenja testova. Nije moguće sve automatizirati, primjerice testne slučajeve čiji se zahtjevi često mijenjaju te one slučajeve koji nisu provedeni ručno barem jednom. S druge strane, testni slučajevi koji se mogu i trebaju automatizirati su oni koji se često ponavljaju, koji oduzimaju puno vremena i koje je teško ručno izvesti. Hiren Tanna navodi prednosti automatiziranog testiranja [17]:

1. Brže povratne informacije – automatizirano testiranje predstavlja olakšanu provjeru valjanosti tijekom različitih faza softverskog projekta. To poboljšava komunikaciju među programerima, dizajnerima i vlasnicima proizvoda te omogućuje otklanjanje potencijalnih kvarova neposredno nakon što se otkriju. Automatizirano testiranje osigurava veću učinkovitost razvojnog tima.

2. Ubrzani rezultati – zahvaljujući brzoj implementaciji automatiziranog testiranja, štedi se puno vremena pa čak i kod testiranja velikih i kompliciranih sustava. To omogućuje da se testiranje izvodi više puta, dajući svaki put brže rezultate s manje uloženog truda i vremena.

3. Smanjeni poslovni troškovi - početno ulaganje je visoko, ali u konačnici, automatizirano testiranje tvrtkama štedi novac. To je uglavnom zbog naglog smanjenja vremena potrebnog za pokrivanje softvera testovima. Doprinosi većoj kvaliteti rada, čime se smanjuje potreba za popravljanjem kvarova nakon izdavanja softvera pa tako i smanjenju troškova projekta.

4. Poboljšanje učinkovitosti testiranja – testiranje zauzima značajan dio cjelokupnog životnog ciklusa razvoja aplikacija. To ukazuje na to da čak i najmanje poboljšanje ukupne učinkovitosti može napraviti ogromnu razliku u ukupnom vremenskom okviru projekta. Iako vrijeme postavljanja automatiziranih testova u početku dugo traje, kasnije automatizacija testova oduzima znatno manje vremena. Automatizirani testovi mogu se izvoditi gotovo bez nadzora, ostavljajući rezultate provedenih testova.

5. Veća ukupna pokrivenost testovima – implementacijom automatiziranih testova može se izvršiti više testova koji se odnose na samu aplikaciju. Povećana pokrivenost softvera testovima omogućuje testiranje više značajki i rezultira kvalitetnijim softverima.

6. Ponovna upotrebljivost automatiziranih testova – zbog ponavljajuće prirode automatiziranih testova, uz relativno laku konfiguraciju njihovih postavki, programeri imaju priliku procijeniti

ponašanje softvera. Automatizirani testovi mogu se ponovno koristiti, stoga se mogu koristiti kroz različite pristupe.

7. Ranije otkrivanje nedostataka – dokumentiranje softverskih nedostataka postaje znatno jednostavnije za testne timove. To pomaže povećati ukupnu brzinu razvoja, istodobno osiguravajući ispravnu funkcionalnost na svim područjima. Što se ranije utvrdi kvar, to su troškovi njegovog uklanjanja manji.

8. Temeljitost u testiranju – testeri obično imaju različite pristupe testiranju, a njihova područja fokusa mogu se razlikovati zbog njihove izloženosti i stručnosti. Uključivanjem automatizacije zajamčena je usredotočenost na sva područja testiranja, čime se osigurava najbolja moguća kvaliteta.

9. Brže vrijeme izlaska na tržište – jednom kad se automatiziraju testovi, njihovo izvršavanje je brže i traje dulje od ručnog testiranja. Iz tog razloga automatizacija testova uvelike pomaže u smanjenju vremena potrebnog za stavljanje softvera na tržište tako što omogućava stalna izvješća o testnim slučajevima.

10. Sigurnost informacija – učinkovitost testiranja uvelike ovisi o kvaliteti podataka koji se koriste pri testiranju. Rješenja za automatizaciju mogu pomoći u stvaranju, manipulaciji i zaštiti testne baze podataka, što omogućuje ponovnu upotrebu podataka.

Praveen Mishra navodi četiri glavna koraka u procesu automatizacije [18]:

1. Definiranje domene automatizacije

Domena automatizacije je dio aplikacije koji treba automatizirati. Ona se može odrediti uzimajući u obzir uobičajene funkcionalnosti aplikacije koje su ključne za poslovanje, tehničku izvedivost, kompleksnost testnih slučajeva te mogućnost korištenja testnih slučajeva u različitim web preglednicima.

2. Odabir alata za testiranje

Drugi korak u procesu automatizacije jest odabir alata za testiranje. Postoji više alata za testiranje dostupnih na tržištu. Preporučuje se provesti eksperiment s određenim alatima za testiranje kako bi se pronašao odgovarajući alat koji će zadovoljiti potrebe testiranja. Odabir alata za testiranje uvelike ovisi o tehnologiji na kojoj je izrađena aplikacija koja se testira.

3. Planiranje, projektiranje i razvoj

U ovoj fazi, potrebno je kreirati strategiju i plan automatizacije. To može uključivati sljedeće detalje: definiranje odabranog alata za automatizaciju, raspored i vremenski rok pisanja i izvršavanja testnih slučajeva, stavke koje se nalaze unutar ili izvan domene automatizacije te rezultate automatiziranih testova.

4. Izvršavanje testnih slučajeva

Tijekom ove faze izvršavaju se skripte za automatizirano testiranje kojima se trebaju predati ulazni podaci kako bi testovi bili spremni za pokretanje. Nakon izvršavanja testnih skripti dostupna su detaljna izvješća testiranja.

U članku *Types Of Automation Testing And Some Misconceptions* navode se tri vrste automatizacije [15]:

1. Automatizacija temeljena na vrsti testiranja

Vrste testiranja u ovom kontekstu se dijele na funkcionalno i nefunkcionalno. Funkcionalno testiranje izvodi se kako bi se testirala poslovna logika softvera. Poslovna logika je set prilagođenih pravila ili algoritama koji upravljaju razmjenom podataka između baze podataka i korisničkog sučelja. Poslovna logika je u osnovi dio softvera koji sadrži informacije koje definiraju poslovanje [19]. Automatiziranje ovakvih testova znači pisanje skripti kojima se potvrđuje očekivana poslovna logika i funkcionalnost softvera. Nefunkcionalno testiranje odnosi se na testiranje onih zahtjeva koji nisu vezani uz poslovne zahtjeve softvera. To su zahtjevi koji se odnose na performanse, sigurnost, baze podataka itd. Ti zahtjevi mogu ostati konstantni ili se mogu prilagoditi veličini softvera.

2. Automatizacija temeljena na fazi testiranja

Prva faza jest automatizacija *unit* testova. Ovakvi testovi pokreću se tijekom razvojne faze softvera, i to od strane programera prije nego li se softver preda testeru na testiranje. Automatizirani *unit* testovi pisani su kako bi se testiralo na razini koda. Pogreške se identificiraju u funkcijama, metodama i procedurama koje programeri napišu. Druga faza je automatizacija web servisa. Sučelje za programiranje aplikacija (engl. *Application Programming Interface (API)*) omogućuje softveru komunikaciju s drugim softverskim aplikacijama. U ovoj vrsti testiranja grafičko korisničko sučelje (engl. *Graphical User Interface (GUI)*) obično nije uključeno. Ovdje se uglavnom testira funkcionalnost, usklađenost i sigurnosni problemi. Ovakve testove mogu izvršavati programeri i

testeri, a to mogu raditi prije ili nakon što se izradi korisničko sučelje aplikacije. Treća faza jest automatizacija testova temeljenih na korisničkom sučelju koji se izvode tijekom faze izvršavanja testova. Ova vrsta automatiziranog testiranja najteži je oblik automatizacije jer uključuje testiranje korisničkog sučelja aplikacije koje je podložno promjenama. No, ova vrsta testiranja najbliža je onome što će korisnici raditi s aplikacijom. S obzirom na to da će korisnici koristiti miš i tipkovnicu, automatizirani testovi oponašaju ponašanje miša i tipkovnice poput klikanja i pisanja po elementima koji su prisutni na korisničkom sučelju.

3. Automatizacija temeljena na vrsti testova

Unit testovi su oni testovi koji se pišu kako bi se testirao kod aplikacije i obično se nalaze u samom kodu. Ovakvi testovi se obično ne dotiču funkcionalnih dijelova aplikacije, ali s obzirom na to da se fokusiraju na kod, prikladno ih je automatizirati kako bi ih programer mogao pokretati kad god je to potrebno. Nakon *unit* testova mogu se automatizirati integracijski testovi. To su testovi koji služe za testiranje aplikacije nakon što se integriraju svi moduli. Vrste testova koje se najčešće automatiziraju su regresijski testovi. Regresijsko testiranje provodi se na kraju testiranja novog modula kako bi se osiguralo da implementacija novog nije utjecala na postojeće module. Iduća vrsta testova su s*moke* testovi, a odnose se na testiranje osnovnih i glavnih svojstava aplikacije. Ti testovi provode se čim se softver preda testerima kako bi se utvrdilo da osnovna i glavna svojstva aplikacije i dalje funkcioniraju kako trebaju. Automatizacija *smoke* testova je potrebna jer se ti testovi izvode više puta, točnije, pri svakoj novoj iteraciji testiranja. Automatiziranje testova korisničkog sučelja može se odnositi na funkcionalnost u aplikaciji, ili jednostavno testiranje elemenata koji se nalaze na korisničkom sučelju aplikacije.

3. ALATI ZA AUTOMATIZACIJU TESTIRANJA

Dostupni su brojni alati za automatizaciju. Pravi izbor uglavnom ovisi o aplikaciji koja se testira i o tehnologiji koju ona koristi [20]. Najbolji dostupan alat ne jamči najbolji ishod testiranja, stoga odabrani alat mora biti najprikladniji, a ne najbolji [21]. Uspjeh pri automatizaciji testova leži u prepoznavanju pravog alata za različite potrebe i zahtjeve. Postupak biranja pravog alata za automatizaciju zahtjeva vrijeme i napor, što u konačnici rezultira dugoročnom učinkovitošću automatizacije testova.

Vrste alata za automatiziranje testova generalno se dijele na javno dostupne, komercijalne i prilagođene. Javno dostupni alati su besplatni alati koji omogućuju korisnicima korištenje izvornog koda. Korisnici se mogu odlučiti za potpuno usvajanje koda ili ga modificirati u skladu sa svojim potrebama za testiranje. Komercijalni alati proizvode se u komercijalne svrhe i obično se distribuiraju putem pretplatničkih planova. Korisnici moraju kupiti licencu koja se plaća za korištenje softvera. U usporedbi s javno dostupnim alatima, komercijalni alati obično imaju više značajki od besplatnih alata i temeljitu korisničku uslugu. Neki projekti koji se testiraju zahtijevaju posebna testna okruženja gdje postupak testiranja ima posebne karakteristike. U takvim slučajevima potrebno je razmotriti razvoj prilagođenog alata jer niti jedan javno dostupan ili komercijalni alat ne zadovoljava potrebne zahtjeve testera. U nastavku ovog poglavlja navedeni su neki od alata koji se koriste za automatizaciju.

3.1. Selenium

Selenium je javno dostupan alat za automatizaciju testova, a koristi se za testiranje web aplikacija [20]. Može se koristiti u više preglednika i na različitim platformama (macOS, Windows i Linux), a testovi se mogu pisati u mnogim programskim jezicima kao što su Python, Java, C#, Scala, Groovy, Ruby, Perl i PHP. Selenium se uglavnom koristi za regresijsko testiranje. Testerima je ponuđen alat za reprodukciju koji omogućuje snimanje i prikazivanje regresijskih testova. Selenium zapravo nije jedan alat, već skup softvera koji uključuje različite alate, a to su Selenium IDE, Selenium WebDriver, Selenium RC i Selenium Grid. Svaki od ovih alata usredotočuje se na određenu razvojnu funkciju kako bi pružili cjelokupnu automatizaciju testova web aplikacija.

3.2. Katalon

Katalon je besplatni licencirani alat koji se može koristiti u više preglednika i omogućuje pokretanje automatiziranih testova za testiranje sučelja za programiranje aplikacija, web sučelja i mobilne uređaje. Uz to, ovaj alat pruža izvješća o analizi i snimanje testova. Katalon se ističe svojim višestrukim svrhama i lakoćom upotrebe, s obzirom na to da može stvarati i ponovo koristiti skripte za testiranje korisničkog sučelja bez potrebe za pisanjem koda.

3.3. Eggplant

Eggplant je razvijen kako bi testerima pružio mogućnost provođenja različitih vrsti testova. Slično Selenium-u, Eggplant nije pojedinačni alat, već je skup alata za automatizirano testiranje, a svaki alat vrši različite vrste testiranja.

3.4. JMeter

JMeter je javno dostupan alat za testiranje u potpunosti napisan Java programskim jezikom, a koji se prvenstveno koristi za funkcionalne testove i testove performansi [21]. Može se koristiti za simulaciju velikog opterećenja na poslužitelju, grupi poslužitelja, mreži ili objektu za testiranje njegove snage ili za analizu ukupnih performansi pod različitim vrstama opterećenja [23].

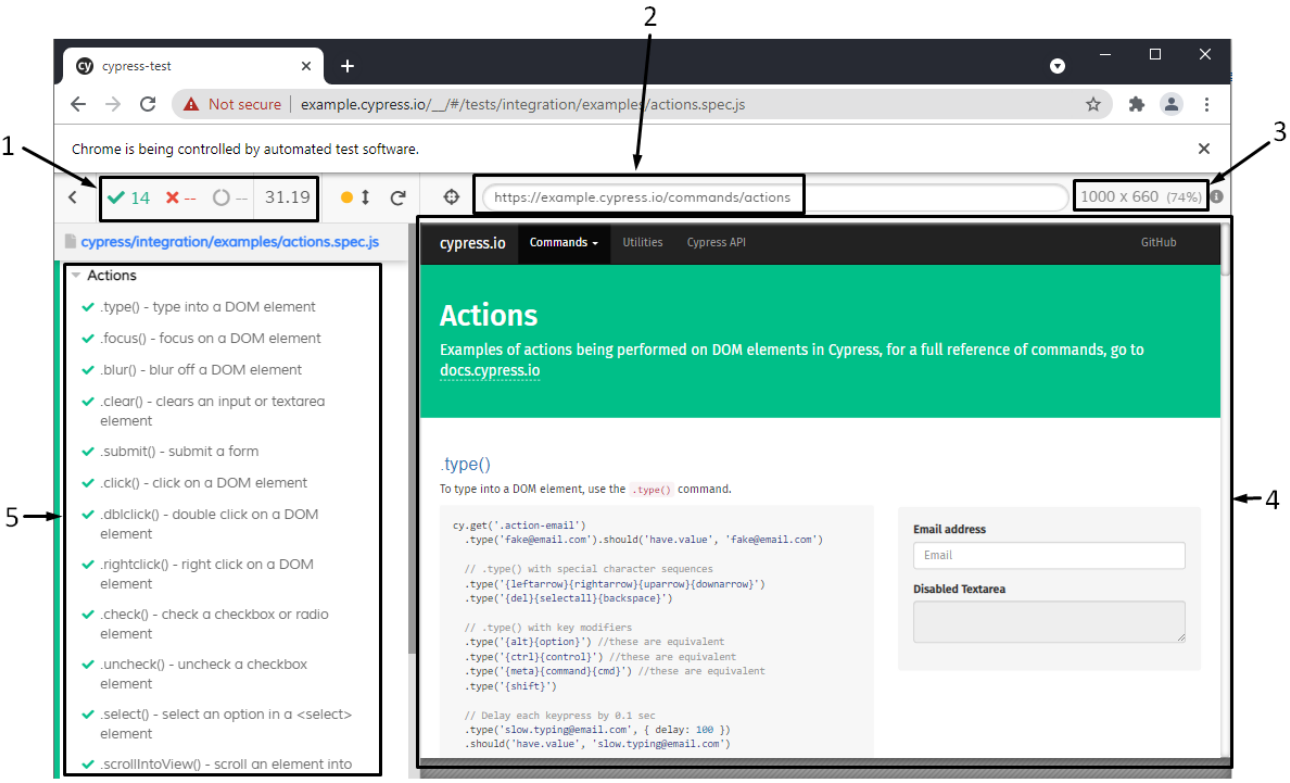
3.5. Cucumber

Cucumber je alat za automatizaciju testova koji podržava razvoj vođen ponašanjem (engl, *Behavior Driven Development (BDD)*). *BDD* je razvoj čiji se pristup sastoji od definiranja ponašanja značajki kroz primjere koji se pišu običnim tekstom. Radi toga se svima pruža jednostavan način pisanja i izvršavanja testnih slučajeva bez obzira na tehničko znanje pojedinaca [24].

3.6. Cypress

Cypress je alat za pouzdano testiranje svega što se pokreće u web pregledniku [3]. Besplatan je i javno dostupan pod MIT licencom. MIT licenca daje korisnicima dopuštenje za ponovnu upotrebu koda u bilo koju svrhu, ponekad čak i ako je kod dio vlasničkog softvera [27]. Većina alata za testiranje funkcionira tako da se naredbe izvode izvan preglednika, ali Cypress djeluje izravno unutar preglednika [26]. Drugim riječima, preglednik je taj koji izvršava napisane testove, a trenutno podržani preglednici su *Chrome, Electron, Chromium, Mozilla Firefox* i *Microsoft Edge*. Cypress ima dvije glavne komponente koje prema zadanim postavkama dolaze uz instalaciju, a to su *Test Runner*

i *Dashboard*. *Test Runner* pokreće testove u jedinstvenom interaktivnom programu koji omogućuje pregled naredbi pri njihovom izvršavanju, kao i pregled same aplikacije koja se testira. Na slici 3.1. prikazan je *Test Runner* i označene su njegove glavne komponente, a ispod slike nalazi se kratki opis tih komponenti.



Slika 3.1. *Test Runner*

1. Izbornik statusa testova: prikazuje koliko testova je uspješno prošlo, koliko ih nije uspješno prošlo, koliko ih je u tijeku i koliko vremena je bilo potrebno za izvođenje testova.

2. Pregled URL-a: prikazuje URL stranice koja se testira.

3. Veličina prozora: prikazana je veličina prozora koja se može konfigurirati kako bi se mogla testirati aplikacija na manjim rezolucijama.

4. Pregled aplikacije: prikaz aplikacije koja se testira dok se izvršavaju naredbe automatiziranih testova.

5. Zapisnik naredbi: prikazuje naredbe koje se izvršavaju dok se provode testovi nad aplikacijom.

Cypress je desktop aplikacija koja se instalira na računalo, a podržava sljedeće operacijske sustave [25]:

- macOS 10.9 i novije verzije (samo 64-bitni operacijski sustav)
- Linux Ubuntu 12.04 i novije verzije, Fedora 21 i Debian 8 (samo 64-bitni operacijski sustav)
- Windows 7 i novije verzije

Ako se za instalaciju Cypress-a koristi *npm* (engl. *Node Package Manager*), potrebno je imati instaliran Node.js i to verziju 12, 14 ili noviju. „Node.js je open source platforma na strani servera razvijena na Google Chrome JavaScript Engine (V8 Engine). Koristi se za razvoj skalabilnih i učinkovitih mrežnih aplikacija u stvarnom vremenu.“ [30]. S instalacijom Node.js-a dolazi ugrađena podrška za upravljanje paketima pomoću *npm* alata. *Npm* je upravitelj paketa za JavaScript uz pomoću kojega se lako instaliraju paketi za aplikacije [29]. Ako se za instalaciju Cypress-a ne koristi *npm*, moguće ga je izravno preuzeti. Izravnim preuzimanjem uvijek će se uzeti najnovija dostupna verzija, a moguće je preuzeti specifičnu stariju verziju Cypress-a tako da se u adresnu traku upiše URL „https://download.cypress.io/desktop/“, te nakon toga željna verzija. Primjerice za preuzimanje verzije Cypress-a 6.8.0 potrebno upisati „https://download.cypress.io/desktop/6.8.0”. No izravnim preuzimanjem Cypress-a gubi se mogućnost snimanja i spremanja testova u Cypress-ovu nadzornu ploču, jer je izravno preuzimanje namijenjeno za brzo isprobavanje Cypress-a [25].

Da bi se Cypress instalirao uz pomoć *npm*-a, potrebno je u nekom od programa koji se koriste za pisanje koda otvoriti direktorij u kojem će se nalaziti testovi, primjerice Visual Studio Code. Nakon toga potrebno je otvoriti terminal i izvršiti sljedeće tri naredbe:

1. **`npm init -y`**
2. **`npm install cypress`**
3. **`npx cypress open`**

Slike 3.2., 3.3. i 3.4. prikazuju terminal nakon izvršavanja gore navedenih naredbi. Naredbom **`npm init -y`** kreira se **`package.json`** datoteka u korijenu projekta koja osigurava to da će se Cypress instalirati u ispravnom direktoriju. U **`package.json`** datoteci nalaze se sve skripte i ovisnosti (engl. *dependencies*) potrebne za projekt. Pokretanjem naredbe **`npm install cypress`** dohvaća se najnovija verzija Cypress-a te se instalira u definiranom direktoriju. Naredbom **`npm install cypress`** kreira se direktorij **`node_modules`** te **`package-lock.json`** datoteka. Kada se naredba

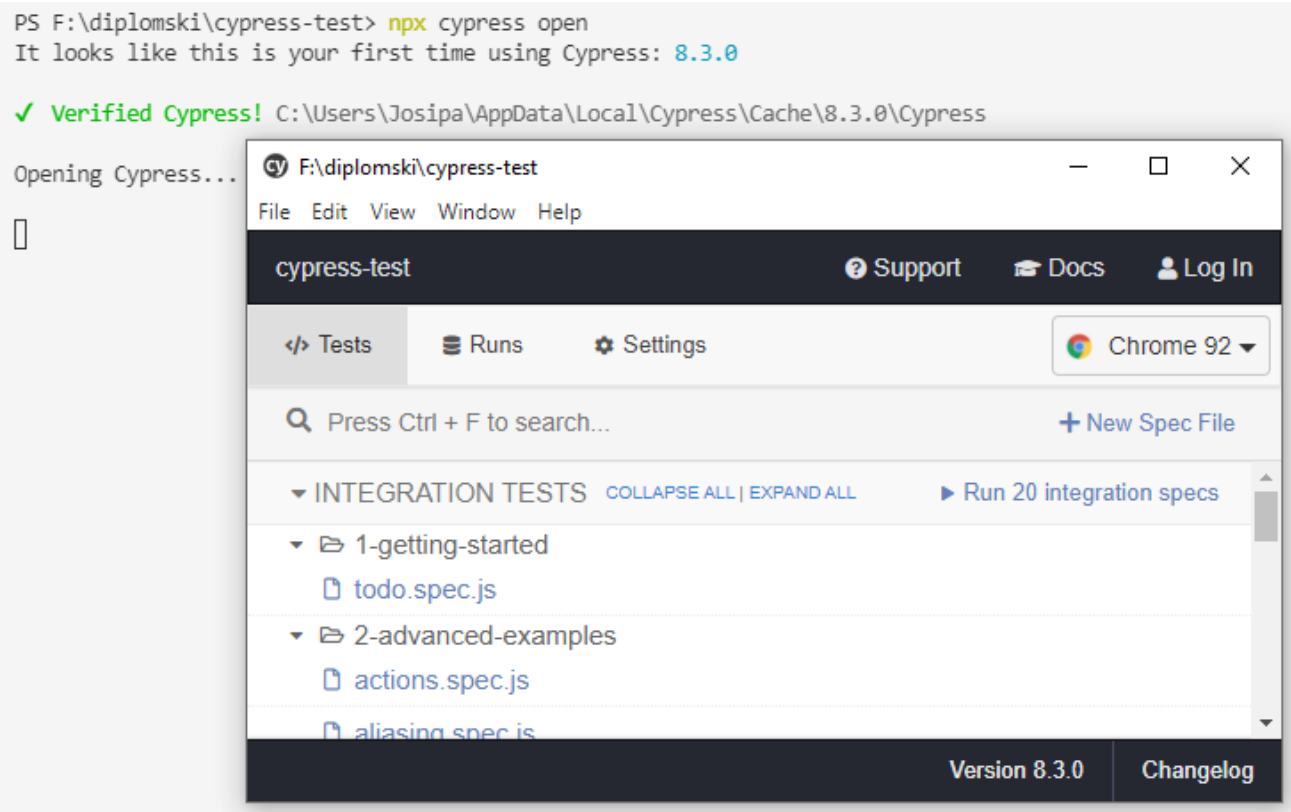
`npx cypress open` prvi put izvrši, kreiraju se **`cypress`** direktorij i **`cypress.json`** datoteka, te se otvara desktop aplikacija.

```
PS F:\diplomski\cypress-test> npm init -y
Wrote to F:\diplomski\cypress-test\package.json:

{
  "name": "cypress-test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}


PS F:\diplomski\cypress-test>
```

Slika 3.2. *Terminal nakon izvršavanja naredbe* ***`npm init -y`***

```
PS F:\diplomski\cypress-test> npm install cypress
npm WARN deprecated har-validator@5.1.5: this library is no longer supported

> cypress@8.3.0 postinstall F:\diplomski\cypress-test\node_modules\cypress
> node index.js --exec install

Installing Cypress (version: 8.3.0)

✓ Downloaded Cypress
✓ Unzipped Cypress
✓ Finished Installation C:\Users\Josipa\AppData\Local\Cypress\Cache\8.3.0

You can now open Cypress by running: node_modules\.bin\cypress open

https://on.cypress.io/installing-cypress

npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN cypress-test@1.0.0 No description
npm WARN cypress-test@1.0.0 No repository field.

+ cypress@8.3.0
added 170 packages from 181 contributors and audited 170 packages in 151.543s

25 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

PS F:\diplomski\cypress-test>
```

Slika 3.3. *Terminal nakon izvršavanja naredbe* ***`npm install cypress`***



Slika 3.4. *Terminal i aplikacija koja se otvara nakon izvršavanja naredbe* ***`npx cypress open`***

Unutar **`cypress`** direktorija nalaze se direktoriji **`fixtures`**, **`integration`**, **`plugins`** i **`support`** [28]. **`Fixtures`** direktorij služi za spremanje vanjskih dijelova statičkih podataka koji se mogu koristiti u testovima. **`Integration`** direktorij služi za spremanje datoteka u kojima se nalaze automatizirani testovi. Prilikom instalacije Cypress-a u direktoriju **`integration`** kreirani su primjeri automatiziranih testova. **`Plugins`** direktorij sadrži datoteke koje se izvršavaju prije pokretanja Cypress-a, prije nego li se web preglednik pokrene, i za vrijeme izvršavanja testova. Datoteke koje su spremljene u **`plugins`** direktoriju izvršavaju se u pozadini *Node* procesa. Tako se testovima omogućuje pristup datotečnom sustavu i ostatku operacijskog sustava pozivanjem naredbe **`cy.task()`**. **`Support`** direktorij sadrži datoteke unutar kojeg se pišu prilagođene metode za višekratnu upotrebu koje su dostupne u svim **`.spec`** datotekama.

4. AUTOMATIZACIJA TESTOVA *CONDUIT* APLIKACIJE

Conduit aplikacija je testna aplikacija društvenog mrežnog dnevnika. Na poveznici „https://github.com/gothinkster/realworld“ su dostupne razne verzije ove aplikacije za čiju implementaciju su se koristile različite tehnologije. Ta aplikacija može se koristiti za izradu automatiziranih testova, što je za potrebu izrade ovog rada i učinjeno. Funkcionalnosti koje ova aplikacija sadrži su:

- provjera autentičnosti korisnika,

- kreiranje, pregled i ažuriranje korisnika,

- kreiranje, pregled, ažuriranje i brisanje članaka,

- kreiranje, pregled i brisanje komentara,

- spremanje članaka u listu omiljenih i micanje članaka iz liste omiljenih

- praćenje autora

Stranice od kojih se sastoji *Conduit* aplikacija su:

- početna stranica,

- stranica za registraciju novog korisnika,

- stranica za prijavu postojećeg korisnika,

- stranica za ažuriranje korisničkog računa,

- stranica za kreiranje ili ažuriranje članaka,

- stranica članaka,

- stranica profila

Kako bi se mogla provesti automatizacija, potrebno je kreirati testni plan s testnim slučajevima koje je tada potrebno barem jednom izvršiti ručno. Slika 4.1. prikazuje stranicu za registraciju novog korisnika. Na toj stranici nalaze se polja za unos korisničkog imena, e-mail-a i lozinke, dugme za registraciju i „Have an account“ poveznica. Osim toga, u navigacijskoj traci nalaze se poveznice

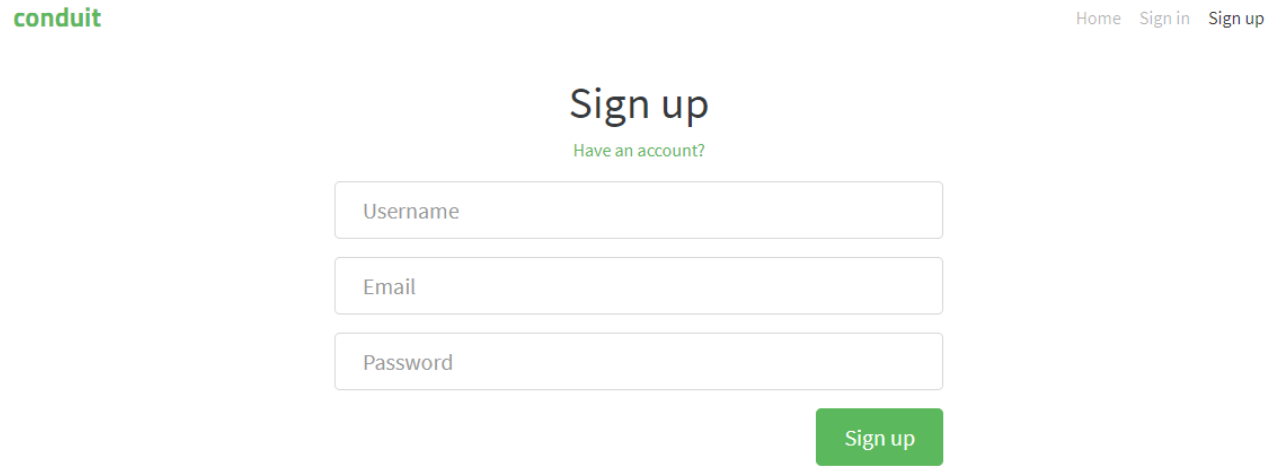
„conduit“, „Home“, „Sign in“ i „Sign up“. Imajući navedene informacije na raspolaganju, moguće je kreirati testni plan za „Sign up“ stranicu koji ima sljedeće testne slučajeve:

- može li se novi korisnik registrirati,

- je li onemogućena registracija korisnika čije korisničko ime ili e-mail adresa već postoji

- je li onemogućena registracija novog korisnika ako nije upisano korisničko ime, e-mail ili lozinka

- je li korisnik odveden na stranice na koje bi trebao biti odveden nakon što klikne na određene poveznice na stranici



Slika 4.1. *Stranica za registraciju novog korisnika*

Klase se sastoje od selektora i metoda, a na slici 4.2. prikazan je dio **`SignUp`** klase s pripadnim selektorima vezanim uz „Sign up“ stranicu. Selektori su jedinstvena svojstva po kojima se identificiraju elementi sa stranice na kojoj se provode testovi. Poželjno je za svaki element sa stranice definirati selektore unutar klase kako bi se učinkovitije kreirale metode za određene akcije koje se mogu izvršiti na stranici. Primjerice selektor **`usernameField`** se odnosi na element na stranici koji za svojstvo **`formcontrolname`** ima vrijednost **`username`**.

cypress > page-objects > SignUp > JS sign-up.js > ...

```
1   /// <reference types="cypress"/>
2
3   class SignUp {
4       usernameField = '[formcontrolname=username]';
5       emailField = '[formcontrolname=email]';
6       passwordField = '[formcontrolname=password]';
7       signUpButton = '[type=submit]';
8       navBar = '[class="navbar navbar-light"]';
```

Slika 4.2. *Dio* ***`SignUp`*** *klase s pripadnim selektorima vezanim uz „Sign up“ stranicu*

Slika 4.3. prikazuje dio **`SignUp`** klase s metodama potrebnima za automatizaciju testa u kojem se provjerava može li se registrirati novi korisnik. Kako bi se novi korisnik registrirao, potrebno je unijeti željeno korisničko ime, e-mail i lozinku te kliknuti na „Sign up“ dugme. Nakon uspješne registracije, korisnik je prijavljen u aplikaciju i njegovo korisničko ime je prikazano u navigacijskoj traci. Metode **`insertUsername`**, **`insertEmail`** i **`insertPassword`** prikazane na slici 4.3. razlikuju se samo u selektorima koje koriste, inače imaju istu funkcionalnost, a to je dohvaćanje elementa sa stranice, u ovom slučaju polja za unos, brisanje postojećeg unosa ako postoji i zapisivanje zadanog teksta u to polje. Metoda **`clickSignUpButton`** dohvaća „Sign up“ dugme te klikne na njega, a metoda **`loggedInUser`** dohvaća element iz navigacijske trake te provjerava je li taj element stvarno postoji na stranici u zadanom trenutku.

```
10      insertUsername(username){
11          cy.get(this.usernameField).clear().type(username);
12      }
13
14      insertEmail(email){
15          cy.get(this.emailField).clear().type(email);
16      }
17
18      insertPassword(password){
19          cy.get(this.passwordField).clear().type(password);
20      }
21
22      clickSignUpButton(){
23          cy.get(this.signUpButton).click();
24      }
25
26      loggedInUser(user){
27          cy.contains(this.navBar, user).should('exist');
28      }
```

Slika 4.3. *Dio* ***SignUp*** *klase s metodama koje se koriste za automatizaciju testova*

Nakon što su svi selektori definirani, i metode kreirane, potrebno je izvan klase kreirati konstantu u koju će se spremiti **SignUp** klasa koju je tada moguće izvesti. To se postiže sljedećim dvjema linijama koda:

```
const signUp = new SignUp();

export { signUp }
```

Unutar direktorija u kojem se nalazi datoteka **sign-up.js**, potrebno je dodati datoteku **index.js** te u njoj izvesti **signUp** komponentu kako bi ona bila dostupna u cijelom cypress projektu. Imajući na raspolaganju gore navedene metode, moguće je kreirati automatizirani test za registraciju novog korisnika. Prvo je potrebno kreirati **sign-up.spec.js** datoteku unutar **integration** direktorija. Kako bi se metode iz **sign-up.js** datoteke mogle koristiti u testovima koji će se nalaziti u **sign-up.spec.js** datoteci, potrebno je uvesti **signUp** klasu naredbom **import { signUp } from '../page-objects/SignUp';**. Slika 4.4. prikazuje sadržaj **sign-up.spec.js** datoteke. Funkcija **describe** prima dva parametra, naslov i funkciju unutar koje se nalaze **beforeEach** i **it** funkcije. Naredbe koje se nalaze unutar **beforeEach** funkcije izvršavaju se prije svake **it** funkcije.

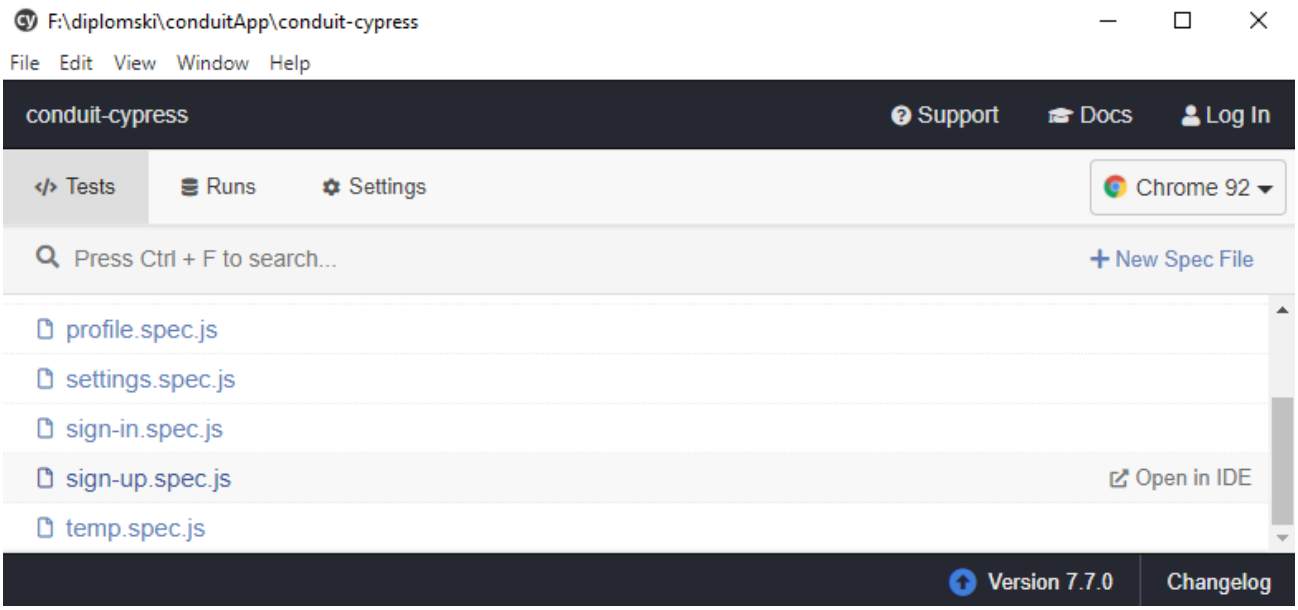
Prva naredba unutar **`beforeEach`** funkcije je **`cy.visit('http://localhost:4200/');`**, ona je zadužena za otvaranje one stranice u web pregledniku čiji URL je predan **`visit`** metodi. Naredba **`signUp.clickSignUpHeader();`** pronalazi „Sign up“ element u navigacijskoj traci te klikne na njega, što korisnika odvodi na „Sign up“ stranicu. Funkcija **`it`**, kao i **`describe`** funkcija, prima dva parametra, naslov i funkciju unutar koje se nalaze sve metode potrebne za izvršavanje testa. Unutar **`it`** funkcije prikazane na slici 4.4. nalaze se sve metode koje su potrebne za izvršavanje testa kojim se provjerava može li se registrirati novi korisnik.

cypress > integration > JS sign-up.spec.js > ...

```
1   /// <reference types="cypress" />
2
3   import { signUp } from '../page-objects/SignUp';
4
5   describe('Sign up page', () => {
6       beforeEach(() => {
7         cy.visit('http://localhost:4200/');
8         signUp.clickSignUpHeader();
9       })
10
11      it('Verify wheter new user can be registered', () => {
12        signUp.insertUsername('Josipa');
13        signUp.insertEmail('josipa@gmail.com');
14        signUp.insertPassword('josipa123');
15        signUp.clickSignUpButton();
16        signUp.loggedInUser('Josipa');
17      })
18
19  })
```

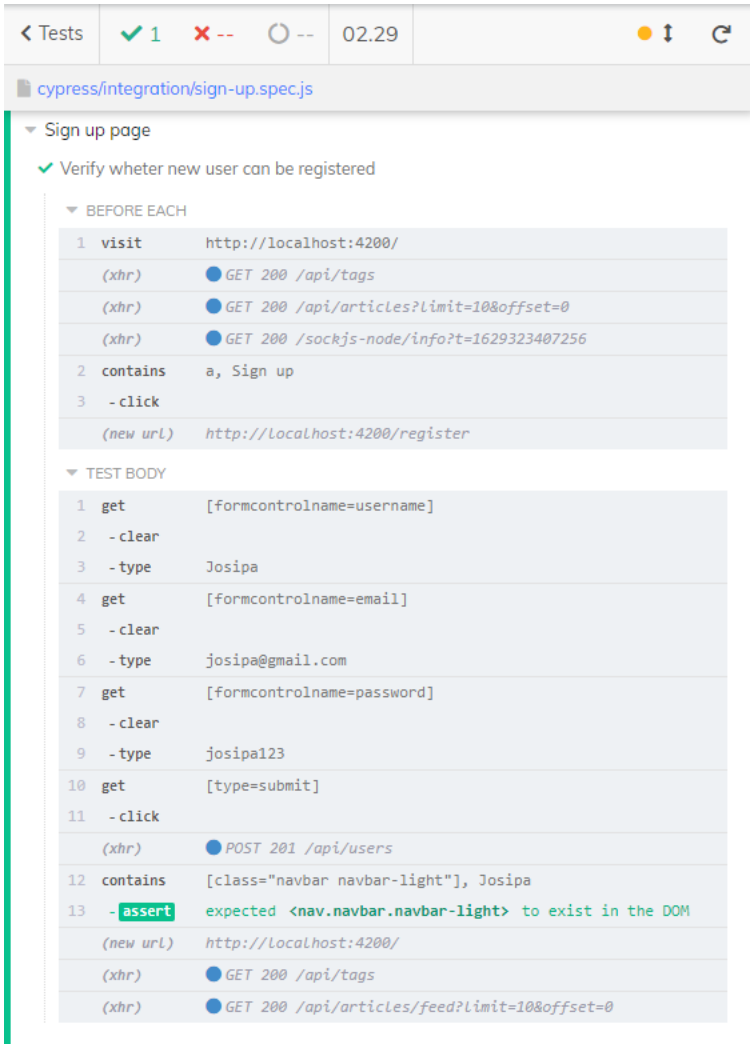
Slika 4.4. *Sadržaj **`sign-up.spec.js`** datoteke nakon automatiziranja testnog slučaja registracije novog korisnika*

Postoje dva načina pokretanja testova koji se nalaze u **`spec.js`** datotekama, jedan je preko Cypress korisničkog sučelja, a drugi je tzv. “headless” način. Da bi se test pokrenuo preko korisničkog sučelja potrebno je u terminalu Visual Studio Code-a pokrenuti naredbu **`npx cypress open`**, nakon toga odabrati **`sign-up.spec.js`**.



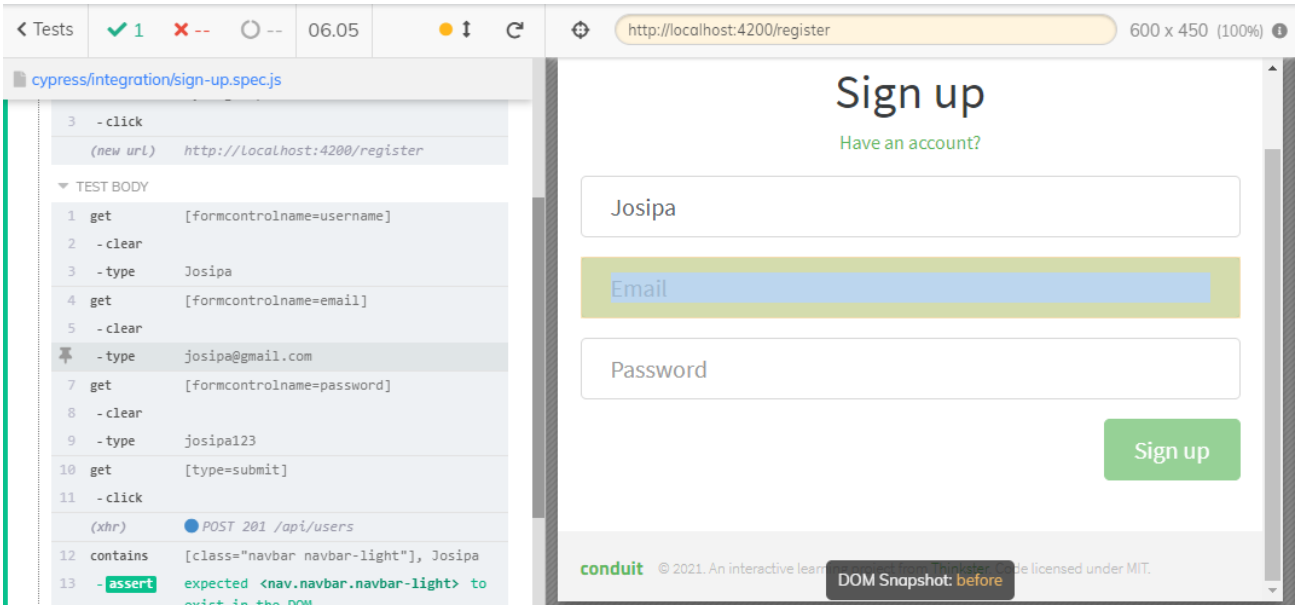
Slika 4.5. *Cypress korisničko sučelje uz pomoću kojega se pokreću testovi u Test Runner-u*

Zapisnik naredbi nakon uspješno izvršenog testa za registraciju novog korisnika prikazan je na slici 4.6..

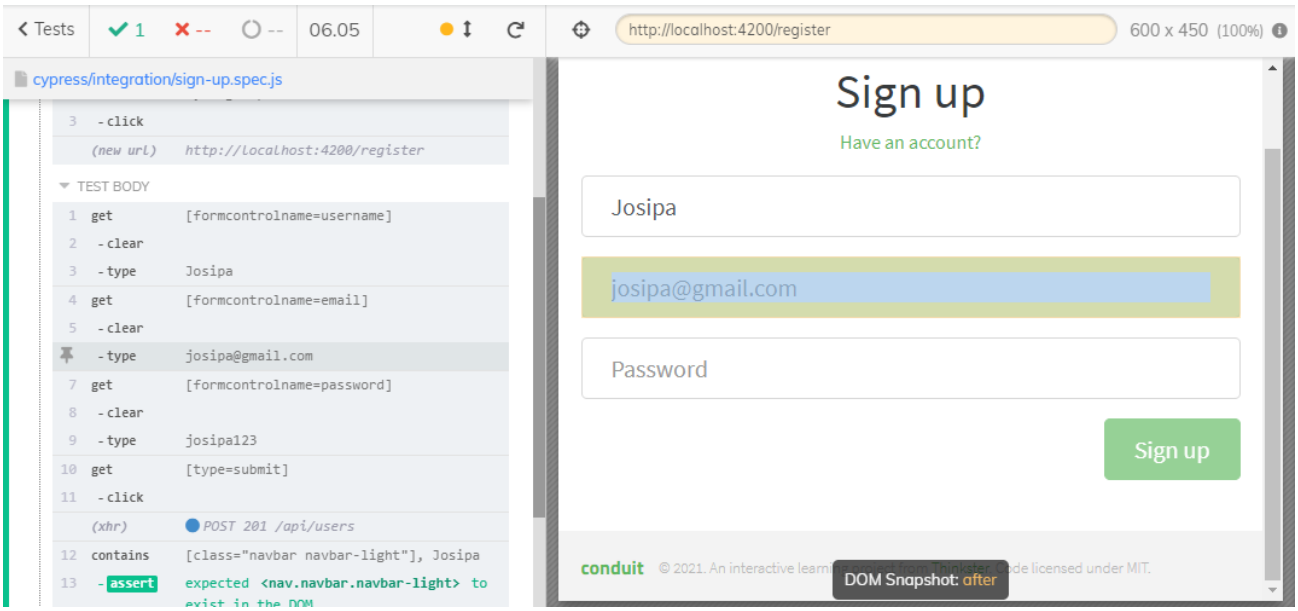


Slika 4.6. *Zapisnik naredbi nakon uspješno izvršenog testa za registraciju novog korisnika*

Zadržavanjem pokazivača nad pojedinim korakom u zapisniku naredbi, moguće je vidjeti što se točno dogodilo u pojedinom trenutku. Primjerice, zadrži li se pokazivač nad šestim korakom u zapisniku naredbi izvršenog testa, izmjenjuju se dvije slike zaslona, jedna u trenutku prije izvršavanja i druga u trenutku nakon izvršavanja naredbe **`type('josipa@gmail.com')`**. Slike zaslona navedenog primjera nalaze se na slikama 4.7. i 4.8..

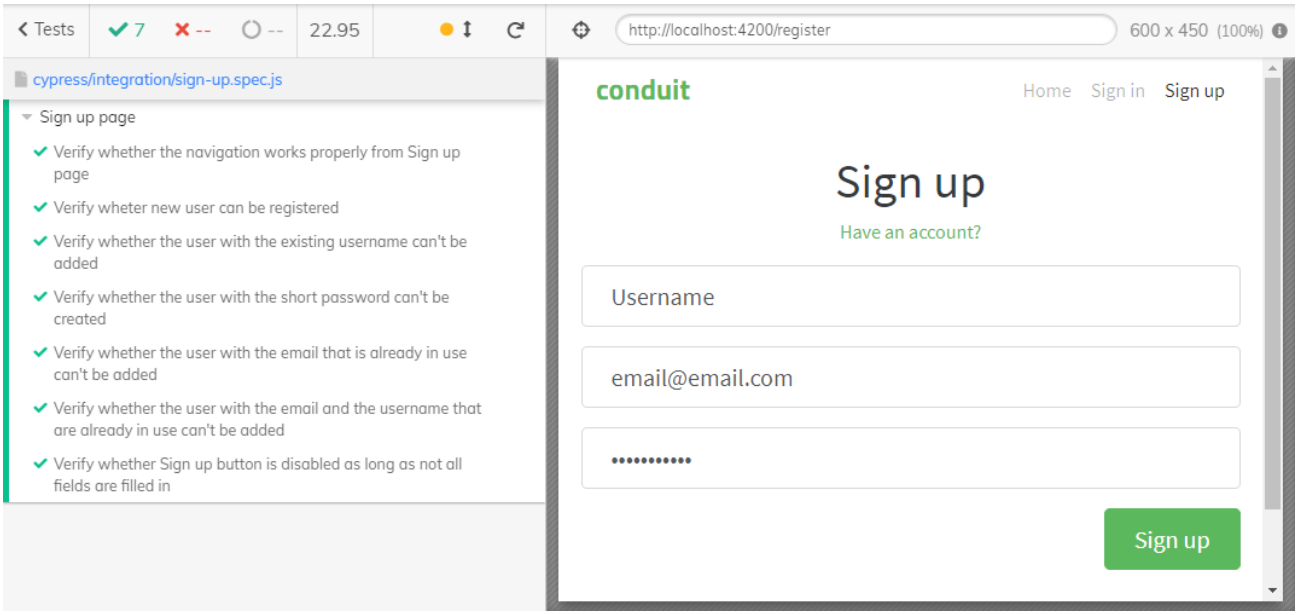


Slika 4.7. *Slika zaslona prije izvršavanja naredbe* ***`type('josipa@gmail.com')`***



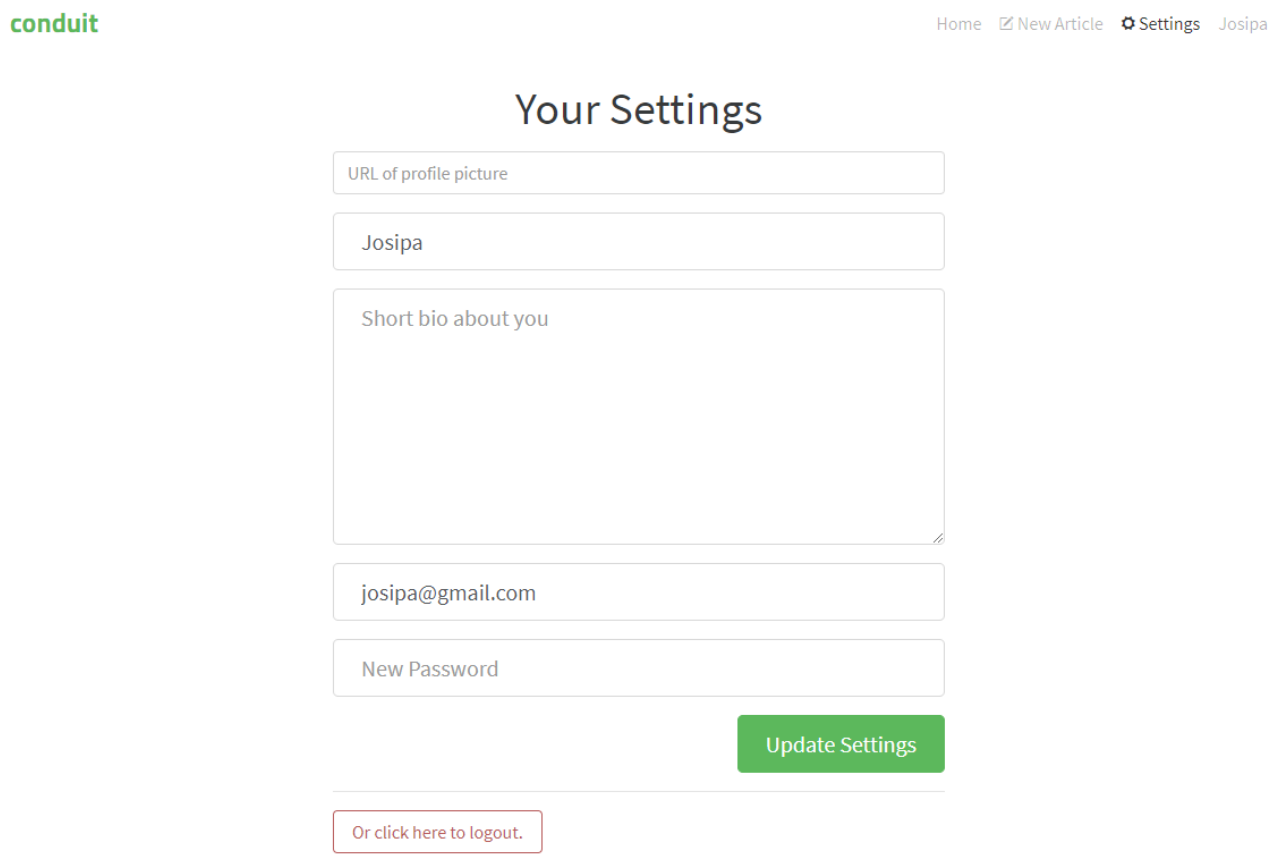
Slika 4.8. *Slika zaslona nakon izvršavanja naredbe* ***`type('josipa@gmail.com')`***

Slika 4.9. prikazuje rezultate svih testova koji su automatizirani, a vezani su uz „Sign up“ stranicu. Vidljivo je da su svi testovi uspješno izvedeni iz čega se može zaključiti da se aplikacija ponaša u skladu s očekivanim rezultatima.



Slika 4.9. *Test Runner nakon izvršavanja svih testova vezanih uz „Sign up“ stranicu*

U slučaju kada rezultat automatiziranog testa ne bude uspješan, u zapisniku naredbi moguće je pronaći točan trenutak u kojemu je test bio neuspješan, što olakšava detekciju problema pa se može zaključiti je li tester pogriješio pri automatizaciji testnog slučaja ili se aplikacija ne ponaša kako je očekivano. Primjer neuspješnog testa je testni slučaj promjene lozinke određenom korisniku. Na slici 4.10. prikazana je „Your Settings“ stranica *Conduit* aplikacije na kojoj se nalazi polje za unos nove lozinke. Ako se u to polje upiše nova lozinka, korisnik bi se pri ponovnom prijavljivanju u aplikaciju trebao moći prijaviti s tom novom lozinkom.



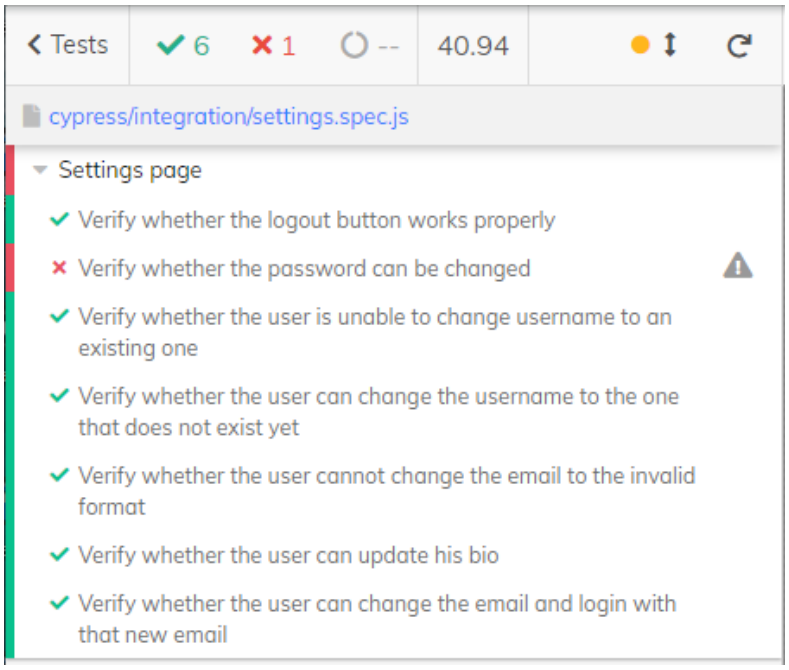
Slika 4.10. *„Your Settings" stranica*

Na slici 4.11. prikazane su metode korištene pri automatizaciji testnog slučaja promjene korisnikove lozinke. U 23. liniji koda upisuje se nova vrijednost lozinke te se u 24. liniji ta promjena sprema. Nakon toga korisnik se odjavljuje iz aplikacije te se ponovo pokušava prijaviti, ali ovog puta s novom lozinkom. Naredbom koja se nalazi u 31. liniji koda provjerava se je li korisnik stvarno prijavljen u sustav nakon upisivanja email-a i lozinke.

```
22      it('Verify whether the password can be changed', () => {
23          settings.insertPassword('changedPassword');
24          settings.clickUpdateSettings();
25          headerFooter.clickSettings();
26          settings.clickLogout();
27          headerFooter.clickSignInHeader();
28          signIn.insertEmail('josipa@gmail.com');
29          signIn.insertPassword('changedPassword');
30          signIn.clickSignInButton();
31          signUp.loggedInUser('Josipa');
32      })
```

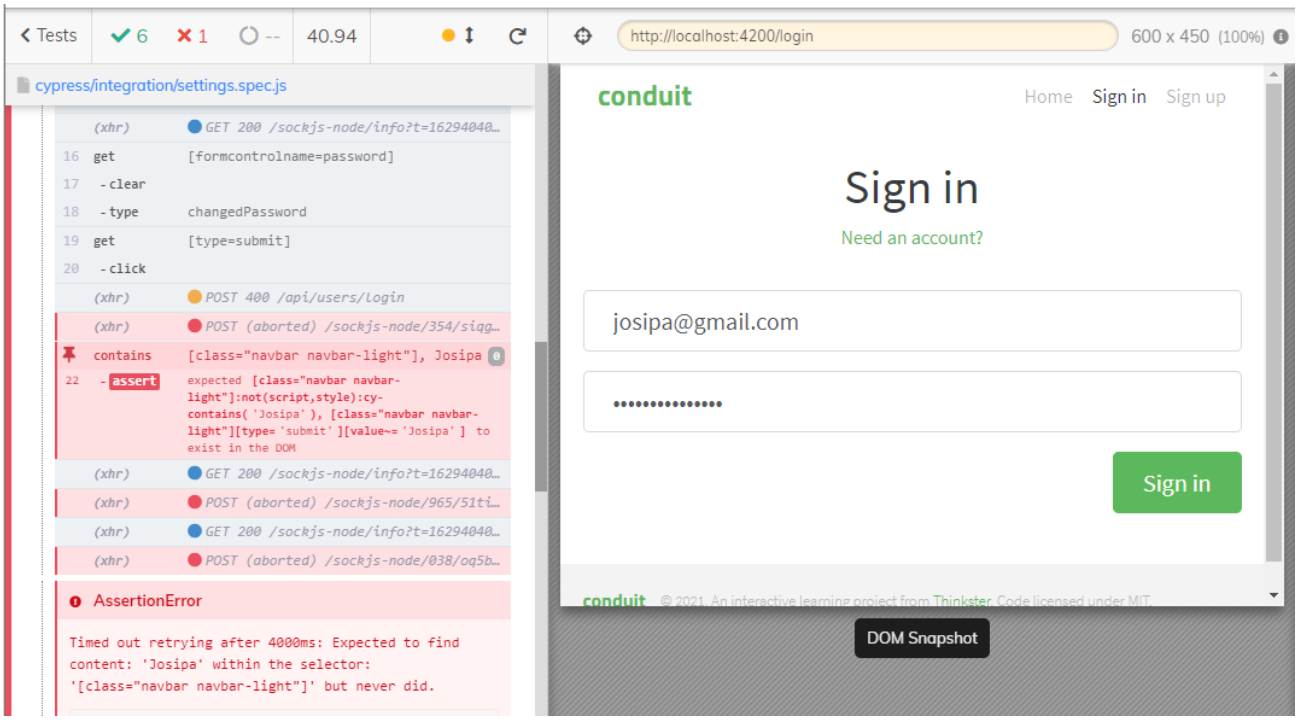
Slika 4.11. *Prikaz automatiziranog testa za provjeru promjene korisnikove lozinke*

Nakon pokretanja **`settings.spec.js`** datoteke, vidljivo je da je test sa prethodne slike neuspješno izvršen.



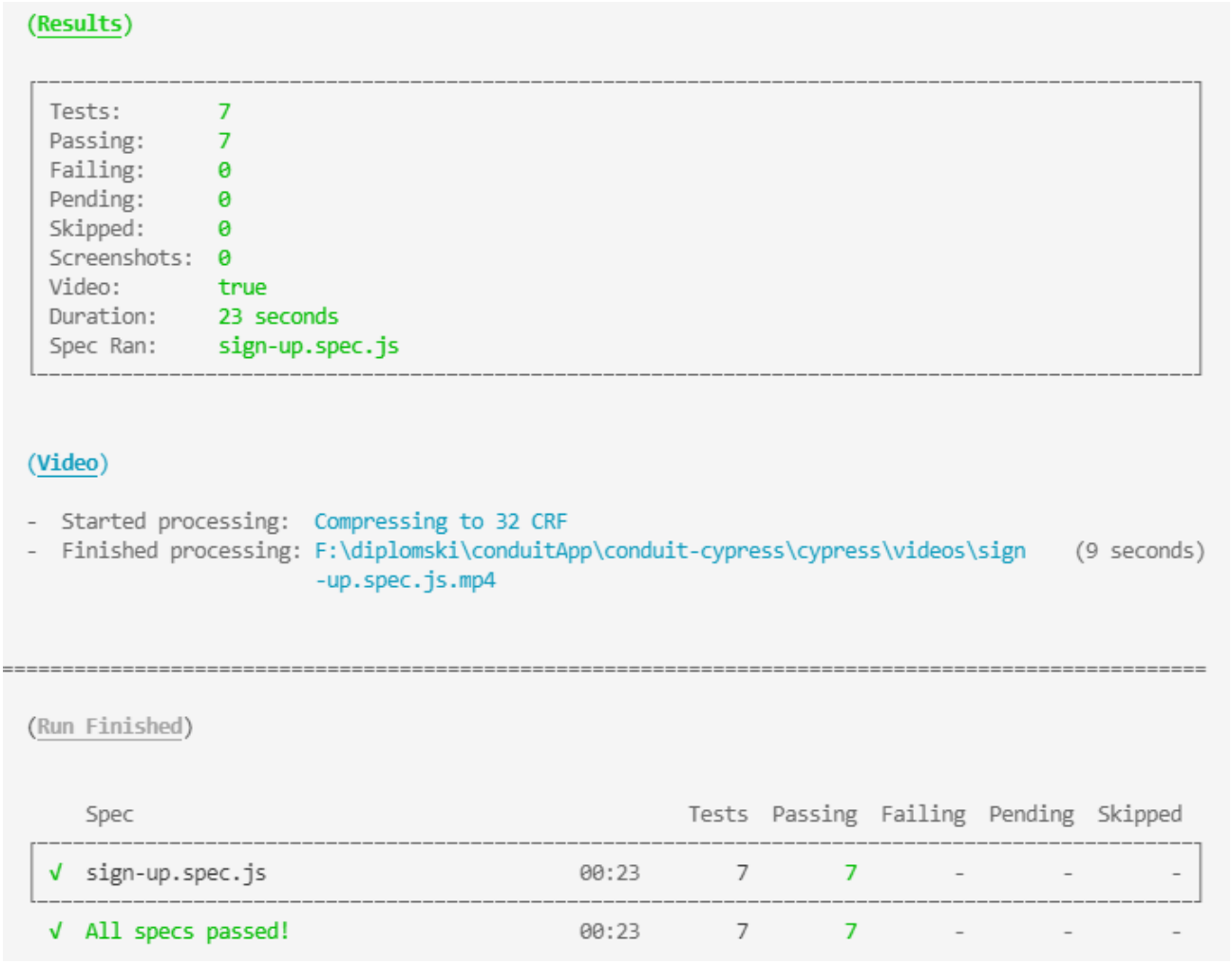
Slika 4.12. *Zapisnik naredbi nakon izvršavanja testova vezanih uz „Your Settings“ stranicu*

Iz zapisnika naredbi može se vidjeti u kojem trenutku se test nije ponašao u skladu s očekivanim rezultatom, a to je u trenutku kada se u navigacijskoj traci očekivalo pronaći korisničko ime korisnika koji se prijavio u aplikaciju.



Slika 4.13. *Zapisnik naredbi(lijevo) i slika zaslona(desno) u trenutku u kojemu je automatiziran test neuspješno izvršen*

Drugi način pokretanja testova je tzv. „headless“ način izvršavanja testova. Da bi se pokrenula **`settings.spec.js`** datoteka potrebno je izvesti naredbu **`npx cypress run --spec „cypress/integration/settings.spec.js“`** u terminalu Visual Studio Code-a. Slike 4.14. i 4.15. prikazuju rezultate „headless“ načina izvršavanja **`sign-up.spec.js`** i **`settings.spec.js`** datoteka. Pokretanjem testova u „headless“ načinu rada stvara se video zapis koji bilježi zaslon koji se inače prikazuje u *Test Runner*-u kada se testovi pokrenu preko Cypress korisničkog sučelja. Ako se neki od testova unutar **`spec.js`** datoteka neuspješno izvrše, osim video zapisa, stvara se i slika zaslona u trenutku u kojemu je test neuspješno izvršen.



Slika 4.14. *Prikaz rezultata testova iz* ***`sign-up.spec.js`*** *datoteke pokrenute „headless“ načinom rada*

```
(Results)

┌────────────────────────────────────────────────────────────────────────────────────────────┐
│ Tests:        7                                                                            │
│ Passing:      6                                                                            │
│ Failing:      1                                                                            │
│ Pending:      0                                                                            │
│ Skipped:      0                                                                            │
│ Screenshots:  1                                                                            │
│ Video:        true                                                                         │
│ Duration:     30 seconds                                                                   │
│ Spec Ran:     settings.spec.js                                                             │
└────────────────────────────────────────────────────────────────────────────────────────────┘


(Screenshots)

-  F:\diplomski\conduitApp\conduit-cypress\cypress\screenshots\settings.spec.js\Set    (1920x1080)
   tings page -- Verify whether the password can be changed (failed).png


(Video)

-  Started processing:  Compressing to 32 CRF
-  Finished processing: F:\diplomski\conduitApp\conduit-cypress\cypress\videos\sett   (10 seconds)
                        ings.spec.js.mp4

   Compression progress:  100%

================================================================================================

(Run Finished)


     Spec                                              Tests  Passing  Failing  Pending  Skipped
┌────────────────────────────────────────────────────────────────────────────────────────────┐
│ ×  settings.spec.js                       00:30        7        6        1        -        - │
└────────────────────────────────────────────────────────────────────────────────────────────┘
   ×  1 of 1 failed (100%)                  00:30        7        6        1        -        -
```

Slika 4.15. *Prikaz rezultata testova iz* ***`settings.spec.js`*** *datoteke pokrenute „headless“ načinom rada*

5. ZAKLJUČAK

Testiranje softvera je proces kritičke analize identificiranja i procjene ispunjavanja specifikacija poslovnih zahtjeva koja pomaže unaprijediti funkcionalnost i upotrebljivost aplikacija. Zadatak ovog diplomskog rada bio je naglasiti važnost osiguravanja kvalitete softvera ručnim i automatiziranim testiranjem. Vodeći se načelima testiranja predstavljenim od strane ISTQB (engl. International Software Testing Qualifications Board), a spomenutim u ovom radu, greške u softveru mogu biti pravovremeno uočene pa tako i otklonjene kako bi se klijentu dostavio softver visoke kvalitete. U radu su opisani razlozi koji govore zašto je testiranje softvera važno i koje su glavne stavke koje bi se pri tome trebale uzeti u obzir, počevši od zadovoljstva klijenata do suzbijanja opasnosti po život koje mogu nastati zbog nedostatka testiranja. Nadalje, predstavljene su faze, metode, proces i tehnike ručnog testiranja, ali i proces automatizacije, vrste automatiziranih testova te alata korištenih pri automatizaciji primarno fokusirajući se na jedan od njih – Cypress. U sklopu ovog rada prikazan je postupak instalacije Cypress-a, kreiranje automatiziranih testova za *Conduit* testnu aplikaciju, te načini na koje se automatizirani testovi u Cypress-u mogu pokretati. Automatizirano testiranje je investicija koja dugoročno štedi vrijeme i novac čiji je cilj smanjiti broj testnih slučajeva koji se trebaju izvoditi ručno, a ne eliminirati ručno testiranje.

LITERATURA

[1] S. Sameer, Why Is Software Testing and QA Important for Any Business, West Agile Labs, 2019, dostupno na: https://www.westagilelabs.com/blog/why-is-software-testing-and-qa-important-for-any-business/#:~:text=The%20importance%20of%20software%20testing%20and%20quality%20assurance%20is%20of,better%20usability%20and%20enhanced%20functionality [3.5.2021.]

[2] Try QA, Why is software testing necessary?, 2017, dostupno na: http://tryqa.com/why-is-testing-necessary/ [8.5.2021.]

[3] Cypress, Why Cypress?, 2021, dostupno na: https://docs.cypress.io/guides/overview/why-cypress#In-a-nutshell [18.9.2021.]

[4] Computerworld UK staff, Top software failures in recent history, 2020, dostupno na: https://www.computerworld.com/article/3412197/top-software-failures-in-recent-history.html [13.5.2021.]

[5] J. Lowenthal, How to Get Started Testing: Best Test Cases to Automate, 2020, Smartbear, dostupno na: https://smartbear.com/blog/how-to-get-started-testing-best-test-cases-to-auto/ [27.2.2021.]

[6] Guru99, Manual Testing Tutorial: What is, Concepts, Types & Tool, 2012, dostupno na: https://www.guru99.com/manual-testing.html [20.2.2021.]

[7] S. Prescott, The seven principles of testing, Box UK, dostupno na: https://www.boxuk.com/insight/the-seven-principles-of-testing/ [1.5.2021.]

[8] ISTQB, General FAQs, dostupno na: https://www.istqb.org/certification-path-root/why-istqb-certification/16-faq-s.html#GeneralFAQs-WhatistheISTQBInternationalSoftwareTestingQualificationsBoard [31.5.2021.]

[9] Eshna, Software Testing Certifications for your Professional Growth, Simplilearn, 2021, dostupno na: https://www.simplilearn.com/testing-throughout-the-software-life-cycle-tutorial-video [7.4.2021.]

[10] Guru99, What is Component Testing? Techniques, Example Test Cases, dostupno na: https://www.guru99.com/component-testing.html#:~:text=Component%20testing%20is%20defined%20as,viewed%20from%20an%20architecture%20perspective [8.4.2021.]

[11] Software Testing Fundamentals, Integration Testing, dostupno na: https://softwaretestingfundamentals.com/integration-testing/#:~:text=INTEGRATION%20TESTING%20is%20a%20level,ISTQB%20Definition [8.4.2021.]

[12] T. Peham, 5 Types Of User Acceptance Testing, Usersnap, dostupno na: https://usersnap.com/blog/types-user-acceptance-tests-frameworks/ [23.4.2021.]

[13] Guru99, Alpha Testing Vs Beta Testing: What's the Difference?, dostupno na: https://www.guru99.com/alpha-beta-testing-demystified.html [24.4.2021.]

[14] Smartbear, Software Testing Methodologies, dostupno na: https://smartbear.com/learn/automated-testing/software-testing-methodologies/?fbclid=IwAR1R5oEWT4j2OZoXKAuSeNHS0ysKP4cQ1OJxm7kIaXevfeHVDoItJBiFuqI [15.5.2021.]

[15] Software Testing Help, Types Of Automation Testing And Some Misconceptions, 2021, dostupno na: https://www.softwaretestinghelp.com/automation-testing-tutorial-2/ [9.5.2021.]

[16] Guru99, Automation Testing Tutorial: What is Automated Testing?, dostupno na: https://www.guru99.com/automation-testing.html [9.5.2021.]

[17] H. Tanna, Top 10 Benefits of Test Automation, DZone, 2017, dostupno na: https://dzone.com/articles/top-10-benefits-of-test-automation [4.3.2021.]

[18] P. Mishra, A Guide to Test Automation Types, Tools, and Benefits, DZone, 2019, dostupno na: https://dzone.com/articles/a-guide-to-test-automation-types-tools-and-benefits [8.5.2021.]

[19] J. Frankenfield, Business Logic, Investopedia, 2020, dostupno na: https://www.investopedia.com/terms/b/businesslogic.asp#:~:text=Business%20logic%20is%20the%20custom,constrains%20how%20a%20business%20operates [13.5.2021.]

[20] M. Berga, Top 7 Automation Testing Tools (2021), ImaginaryCloud, 2021, dostupno na: https://www.imaginarycloud.com/blog/top-automation-testing-tools/ [15.5.2021.]

[21] Katalon, How to Select the Right Automation Testing Tool, 2020, dostupno na: https://www.katalon.com/resources-center/blog/automation-testing-tool-strategy/ [16.5.2021.]

[22] Guru99, How to Select Best Automation Testing Tool & Test Management Tool, 2021, dostupno na: https://www.guru99.com/testing-automation-why-right-tools-are-necessary-for-testing-success.html [15.5.2021.]

[23] The Apache software fundation, dostupno na: https://jmeter.apache.org/ [15.5.2021.]

[24] B. Anderson, Best Automation Testing Tools for 2021, Brian, 2017, dostupno na: https://briananderson2209.medium.com/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2 [15.5.2021.]

[25] Cypress, Installing Cypress https://docs.cypress.io/guides/getting-started/installing-cypress#System-requirements (29.5.2021.)

[26] A. Khetarpal, What is Cypress: Introduction and Architecture, ToolsQA, 2020, dostupno na: https://www.toolsqa.com/cypress/what-is-cypress/ [18.3.2021.]

[27] D. Berman, What is the MIT Licence? Top 10 questions answered, 2020, dostupno na: https://snyk.io/learn/what-is-mit-license/ [23.5.2021.]

[28] A. Khetarpal, Cypress Test, ToolsQA, 2020, dostupno na: https://www.toolsqa.com/cypress/cypress-test/ [29.5.2021.]

[29] NPM: Upravljanje JavaScript paketima, 2021, dostupno na: https://hr.admininfo.info/npm-gestionar-paquetes-javascript [22.8.2021.]

[30] I. Lukić, Predavanje 5, Uvod u Node.js (materijali s predavanja)

SAŽETAK

Testiranje je neophodno pri osiguravanju kvalitete softvera. Vodeći se načelima testiranja poboljšava se učinkovitost i usredotočenost pri testiranju. Postoje četiri glavne razine testiranja: testiranje komponente, integracijsko testiranje, testiranje sustava i provjera prihvatljivosti. Vrste testiranja razlikuju se po ciljevima testiranja te se dijele na funkcionalno, nefunkcionalno, strukturno te regresijsko i ponovno testiranje. Automatizirano testiranje predstavlja neprestano izvođenje niza testova iznova bez potrebe za ručnim izvršavanjem. Cilj automatiziranog testiranja je smanjiti broj testnih slučajeva koji se trebaju ručno izvoditi, a ne eliminirati ručno testiranje. Cypress je alat za pouzdano testiranje svega što se pokreće u web pregledniku. Testni slučajevi koji se mogu i trebaju automatizirati su oni koji se često ponavljaju i koji oduzimaju puno vremena.

Ključne riječi: automatizirano testiranje, Cypress, kvaliteta softvera, ručno testiranje

ABSTRACT

The importance of software quality assurance using manual and automated testing

Testing is required in software quality assurance. Guided by the testing principles the focus and the efficiency of testing improves. Stages of manual testing include unit testing, integration testing, system testing and acceptance testing. Types of testing differ according to testing objectives and are divided into regression and retesting, functional, non-functional and structural testing. Automated testing is the continuous performance of series of tests repeatedly conducted without the need for manual execution. The goal of automated testing is not to eliminate the manual testing, but to narrow down the number of tests that need to be executed manually. Cypress is a tool for reliably testing anything that runs in web browser. Test cases that can and need to be automated are the ones that are often repeated and time-consuming.

Key words: automated testing, Cypress, software quality, manual testing

PRILOZI

Na poveznici https://github.com/jviljevac96/conduit-cypress nalazi se izvorni kod automatiziranih testova *Conduit* aplikacije.