Cypress okruženje za automatizirano testiranje internet aplikacija

Turkalj, David

Master's thesis / Diplomski rad

2022

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:200:881205

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-08-11**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni studij

CYPRESS OKRUŽENJE ZA AUTOMATIZIRANO TESTIRANJE INTERNET APLIKACIJA

Diplomski rad

David Turkalj

Osijek, 2022. godina.

SADRŽAJ

1 UVOD 4
2 TEORIJSKE OSNOVE 5
2.1 Kvaliteta softvera 5
2.2 Povijest testiranja softvera 5
2.3 Testiranje softvera 6
2.3.1 Načela testiranja softvera 6
2.4 Testni slučaj 7
2.5 Razine testiranja softvera 7
2.5.1 Testiranje jedinice 8
2.5.2 Integracijsko testiranje 8
2.5.3 Testiranje sustava 8
2.5.4 Testiranje prihvatljivosti 9
2.5.5 Regresijsko testiranje 9
2.6 Tipovi testiranja softvera 9
2.6.1 Ručno testiranje softvera 9
2.6.2 Automatizirano testiranje 11
2.7 Testiranje od kraja do kraja 13
2.7.1 Implementacija 15
3 ALATI ZA AUTOMATIZIRANO TESTIRANJE 16
3.1 Selenium 17
3.2 Katalon Studio 18
3.3 Appium 18
3.4 TestComplete 18
3.5 Cypress 18
3.5.1 Značajke 19
3.5.2 Cypress metode 20

3.5.3 Tipovi testova 20
4 CYPRESS BIBLIOTEKA 24
4.1 Motivacija 24
4.2 Realizacija 27
4.2.1 Funkcije 29
4.3 Primjena 35
4.3.1 Postavljanje projekta 35
4.3.2 Pisanje testnih slučajeva 36
5 ZAKLJUČAK 46
6 LITERATURA 47
7 SAŽETAK 49
8 ABSTRACT 50
9 ŽIVOTOPIS 51

1 UVOD

Razvoj softvera je dugotrajan i skup proces. Tijekom razvoja, softver mora biti testiran kako bi se osiguralo ispravno funkcioniranje u stvarnom okruženju. Proces kritičke analize procjene ispunjava li softver specifikacije dizajniranih zahtjeva je testiranje softvera. Cilj ovog postupka je dostaviti softver koji ne sadrži greške.

Testiranjem softvera mogu se poboljšati karakteristike softvera. Osim softvera bez greške, dostavljen softver bolje je kvalitete, sigurniji je i efikasniji. Kako bi se postigla bolja kvaliteta i smanjio broj grešaka u softveru na minimum, koristi se više razina testiranja. Prva je testiranje jedinice, zatim slijedi integracijsko testiranje, nakon kojega ide testiranje sustava i posljednje testiranje prihvatljivosti. Svaka razina testiranja u sebi sadrži regresijsko testiranje. Postoje dva načina mogućeg testiranja, ručno testiranje i automatizirano testiranje. Svaki način ima svoje prednosti i mane, zbog toga imaju različite zadatke i probleme koje rješavaju. Ručno testiranje provode testeri prateći jednu od nekoliko metoda, koja se odabire prema potrebama softvera koji se testira. U drugu ruku, automatizirano testiranje izvodi računalni program tako što prati zadanu skriptu naredbi. Danas postoji velik izbor alata za automatizirano testiranje softvera. Izbor alata se svodi na to koji je zadatak potrebno zadovoljiti i kakvo je znanje testera. U ovom radu naglasak je na *Cypress* alat za automatizirano testiranje.

Tijekom razvoja testnih slučajeva, testeri se mogu koristiti pomoćnim alatima. Jedan primjer pomoćnog alata može biti dodatna biblioteka pomoćnih funkcija za testiranje u alatu koji je odabran. Ideja biblioteka je da pomognu testeru brže i jednostavnije pisanje testnih slučajeva. U slučaju rada u većem timu, još jedna velika prednost je mogućnost standardizacije pisanja testnih slučajeva.

Zadatak ovog diplomskog rada je predstaviti razvoj testiranja, razine i tipove, testiranje od kraja do kraja (*E2E*), alate za automatizirano testiranje, usporedbu ručnog i automatiziranog testiranje i na kraju implementaciju *Cypress* biblioteke s dodatnim funkcijama i standardiziranom strukturom. Nakon prvog uvodnog poglavlja u drugom poglavlju navedene su teorijske osnove testiranja softvera. Opisane su kvalitete softvera, principi testiranja, razine i tipovi testiranja te način testiranja koji se koristi u ovom radu. Treće poglavlje je posvećeno pregledu alata za automatizirano testiranje. Uspoređeno je pet najpopularnijih alata, te je detaljnije opisan alat koji se koristi za testiranje internet aplikacije u sklopu ovog diplomskog rada. Posljednje poglavlje opisuje razvijenu *Cypress* biblioteku. Objašnjena je motivacija, način realizacije te su prikazani primjeri korištenja na stvarnoj internet aplikaciji.

2 TEORIJSKE OSNOVE

Softver je skup računalnih programa i njima pripadne dokumentacije, koji se isporučuju korisniku. Svaki softver ima svojstva koja se definiraju u fazi razvoja i oblikovanja. Greške ili nedostatci gotovog proizvoda često su rezultat propusta tijekom same proizvodnje. Isto tako, nedostatak se može dogoditi i tijekom samog definiranja i dizajniranja proizvoda. Ti se nedostatci sprječavaju temeljitim ispitivanjem samog dizajna.

2.1 Kvaliteta softvera

Početkom osamdesetih godina dvadesetog stoljeća javljaju se mjerljivi zahtjevi za utvrđivanje karakteristika softvera. Mjerenjem zahtjeva dolazi se do kvalitete softvera. Uz mjerenje dolazi i mogućnost uspoređivanja i vrednovanja softvera, mogućnost kvantificiranja atributa i praćenje životnih ciklusa softvera. Uvođenjem mjerenja karakteristika bitno se utječe na sam razvoj softvera. Stvoreni su novi sustavi, norme i vrijednosti po kojima se softver vrednuje.

Postupnim uvođenjem normi kroz vrijeme dolazi do određenih očekivanja sljedećih kvaliteta softvera:

- **Mogućnost održavanja** – softver se može mijenjati u skladu s promjenama potreba korisnika,
- **Efikasnost** – program mora imati zadovoljavajuće performanse, odnosno treba upravljati resursima na štedljiv način,
- **Pouzdanost i sigurnost** – program ne smije izazivati ekonomske ili fizičke štete, mora se ponašati predvidljivo,
- **Upotrebljivost** – program treba imati sve funkcije koje korisnik može očekivati, mora sadržavati dokumentaciju, sučelje treba biti zadovoljavajuće i lako za korištenja,
- **Kompatibilnost** – mogućnost programa da može razmjenjivati podatke s drugim programima.

2.2 Povijest testiranja softvera

Testiranje je pojam odvojen od debugiranja 1979. godine, a uveo ga je Glenford J. Myers. William C. Hetzel i Dave Gelperin svrstavaju ciljeve i faze u testiranje:

- do 1956. godine – testiranje orijentirano debugiranju
- 1957. do 1987. godine – testiranje orijentirano na demonstraciju
- 1979. do 1982. godine – testiranje orijentirano destrukciji

- 1983. do 1987. godine – testiranje orijentirano procjeni
- 1988. do 2000. godine – testiranje orijentirano prevenciji
- od 2000. godine – uvodi se agilno testiranje, proces postaje istraživački, nije više samo validacija i verifikacija.

Vrijeme potrebno za testiranje kroz povijest se mijenjalo. Sve se više vremena provodi na postupak testiranja softvera. Analitičari procjenjuju da developer potroši 20 posto svog vremena na dizajniranje i programiranje dok je ostatak vremena utrošen na rješavanje problema i grešaka [6].

2.3 Testiranje softvera

Kada je u pitanju testiranje softvera, česte pretpostavke nisu ispravne. Tvrdnja da je testiranje softvera proces demonstriranja nepostojanja grešaka u softveru, nije ispravna. Također, tvrdnja da je svrha testiranja softvera dokazivanje da softver radi točno kako je zamišljen isto nije ispravna. Još jedna od tvrdnji je, testiranje je proces utvrđivanja da softver radi ono za što je dizajniran. To su sve ne ispravne tvrdnje. Točnija tvrdnja je, testiranje softvera je proces izvođenja programa s ciljem pronalaženja grešaka [1].

Testiranje se još može definirati kao proces verifikacije i validacije programa. Verificiranjem se utvrđuje ispunjenje zahtjeva i uvjeta koji su zadani u dokumentaciji proizvoda. Validacijom se utvrđuje pravi li se dobar proizvod, odnosno odgovara li proizvod stvarnim potrebama korisnika.

2.3.1 Načela testiranja softvera

Testiranje softvera je izuzetno zahtjevan zadatak. Postoje definirane upute za testere kako bi se taj posao olakšao. Upute moraju pratiti načela testiranja. Tijekom povijesti testiranja mogu se prepoznati loši uzorci testiranja. Za savladavanje loših uzoraka definirana su načela testiranja [2] koja su navedena u nastavku.

1. **Testiranje prikazuje prisutnost pogrešaka** – testiranje softvera može otkriti jedan ili više nedostataka u primjeni, međutim samo testiranje ne može dokazati da je aplikacija bez grešaka. Stoga je važno osmisliti testne slučajeve u kojima se može pronaći što više nedostataka.
2. **Iscrpno (potpuno) testiranje nije moguće** – osim u slučajevima kada softver ima vrlo jednostavnu logičku i podatkovnu strukture i ograničene unose. Nije moguće testirati sve moguće kombinacije podataka i scenarija. Stoga su granični slučajevi i oni koji predstavljaju najveći rizik na ispravan rad softvera, najveći prioritet za testiranje.

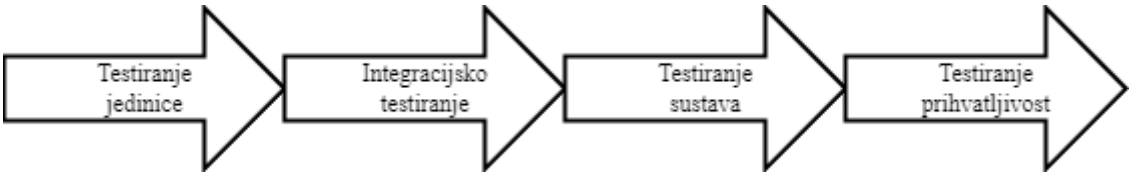
3. **Testiranje u ranoj fazi razvoja** – ako su zahtjevi i dokumentacija softvera dostupni, testiranje je moguće provoditi. Testiranje u ranoj fazi razvoja softvera je najproduktivnije i najkorisnije. U ranoj fazi greške se lakše otkrivaju i sami popravak grešaka je jeftiniji i lakši.
4. **Grupiranje grešaka** – navodi da mali broj funkcija sadrži većinu otkrivenih nedostataka. Ako se u više testnih slučajeva koriste funkcije u kojima se nalazi greška, rješavanjem greške prolaznost testova se povećava.
5. **„Paradoks pesticida“** – stalno korištenje istih pesticida protiv insekata dovodi do razvoja otpornosti insekata na te pesticide. Isto vrijedi i za testiranje softvera. Isti skup ponavljajućih testova s vremenom prestaje otkrivati nove greške. Kako bi se to eliminiralo potrebno je konstantno pregledavati i revidirati testove, dodavajući nove funkcionalnosti i promjenu same ideje testnog slučaja.
6. **Testiranje ovisi o kontekstu** – način na koji se testira internet trgovina biti će drugačiji od načina testiranja reklame na polici u trgovini. Potrebno je korištenje drugačijih metodologija, tehnika i vrsta testiranja.
7. **Nepostojeća greška (zabluda)** – temeljito testiran softver (bez grešaka) i dalje može biti neupotrebljiv. Samo otkrivanje grešaka u softveru ne koristi ako softver ne odgovara na zahtjeve korisnika. Iz toga je zaključivo da dizajn i zahtjevi softvera mogu sadržavati greške.

2.4 Testni slučaj

Testni slučaj (engl. *test case*) je skup podataka, uvjeta, procedura i očekivanih rezultata za izvođenje i ostvarivanje određenog zadanog cilja. Cilj može biti testiranje određene funkcije softvera. Testni slučaj treba biti jedinstven, testirati jednu funkcionalnost, biti izoliran od drugih testnih slučajeva, lako razumljiv i brzo izvediv. Pisanjem testnih slučajeva dolazi se do određenih prednosti. Pomažu u poboljšanju kvalitete softvera, smanjuju troškove održavanja i podrške softvera, provjeravaju zadovoljenost zahtjeva korisnika i slično.

2.5 Razine testiranja softvera

Tijekom razvoja softvera, softver prolazi kroz određene faze, skup tih faza se naziva životni ciklus razvoja softvera. Tijekom razvoja softvera, novi testni slučajevi se svrstavaju u određenu grupu testnih slučajeva. Sortiranje ovisi o načinu razvoja softvera ili razini specifičnosti testa, redoslijed razina prikazan je na Slika 2.1.



Slika 2.1 *Redoslijed razina testiranja softvera.*

2.5.1 Testiranje jedinice

Testiranje jedinice (engl. *unit test*) verificira funkcionalnost pojedinih dijelova softvera u izolaciji. To je prvi krug testiranja u kojem se fokus testiranja stavlja na pojedine funkcije softvera kako bi se utvrdilo njihovo potpuno i ispravno funkcioniranje. Glavni cilj je utvrditi točno funkcioniranje softvera kako je definirano u dizajnu softvera. Jedinica može biti funkcija, potprogram ili procedura. Najčešća metoda testiranja na ovoj razini testiranja je metoda bijele kutije (engl. *White box Testing*).

2.5.2 Integracijsko testiranje

Integracijsko testiranje slijedi nakon testiranja jedinica. Redoslijed je takav jer je ideja integracijskog testiranja nizanje više testova jedinica. Cilj ove razine testiranja je verificiranje i validacija ispravnog funkcioniranja sučelja (engl. *interface*). Sučelje je komunikacija između funkcija softvera, komponenti funkcija ili vanjskih servisa. Uz testiranje ispravnosti komunikacije, testira se i sama efikasnost softvera. Ako su pojedinačne jedinice efikasne, ne znači i da cijeli program mora biti efikasan. Veliki dio kompleksnosti programa dolazi iz komunikacije između pojedinih funkcija i to može biti ne efikasno. Kako bi se pomogla poboljšati efikasnost provodi se integracijsko testiranje.

2.5.3 Testiranje sustava

Testiranje sustava je prva razina na kojoj se testira cijeli softver kao cjelina. Cilj ovog testiranja je utvrditi zadovoljava li softver sve zahtjeve dizajna i korisnika, te zadovoljava li određenu razinu kvalitete. Testiranje sustava provode ljudi koji nisu sudjelovali u samom razvoju softvera kako bi imali čisti pogled na softver i podatke. Testiranje se izvodi u razvojnom okruženju sličnom produkcijskom razvojnom okruženju. Ovo testiranje validira i verificira softver kako bi se utvrdilo zadovoljava li sve propisane zahtjeve (tehničke, funkcionalne i poslovne).

2.5.4 Testiranje prihvatljivosti

Testiranje prihvatljivosti je posljednja razina testiranja. Često se može pronaći i naziv testiranje prihvatljivosti korisnika, a provodi se kako bi se utvrdila spremnost softvera za isporučivanje korisnicima. Test na ovoj razini može pasti ako se tijekom razvoja softvera dogodi promjena dizajna koja nije dobro definirana i time krivo implementirana. Glavni cilj je utvrditi zadovoljava li softver poslovne potrebe korisnika. Nakon prolaza, softver se može isporučiti korisnicima.

2.5.5 Regresijsko testiranje

Regresijsko testiranje nije razina testiranja, ali koristi se tijekom svake razine ovisno o potrebi. Regresijsko testiranje je selektivno testiranje cijelog softvera ili pojedinih jedinica. Obavlja se nakon promjena softvera kako bi se osiguralo da se ne dogode neželjeni efekti, točnije da se osigura da test koji je prolazio prije promjena i dalje prolazi. Regresije nastaju kao rezultat nenamjernih promjena softvera. Regresijsko testiranje ovisi razini razvoja softvera i o riziku koju promjena može uzrokovati. Promjene u kasnom razvoju softvera nose velik rizik dok je u ranim fazama taj rizik minimalan.

2.6 Tipovi testiranja softvera

Testiranje se može izvoditi na dva glavna načina, ručno i automatizirano. Svaki tip testiranja ima svoje prednosti i mane zbog kojih je nekada bolje koristi jedan ili drugi tip.

2.6.1 Ručno testiranje softvera

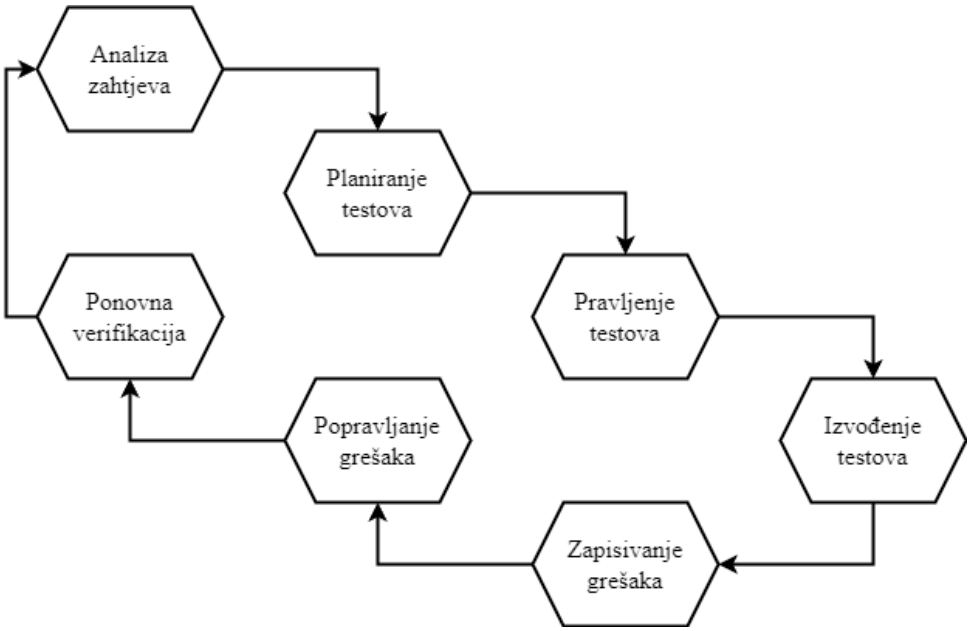
Ručno testiranje softvera se provodi tako da se prate točno definirani testni koraci. Cilj je pronaći sve greške unutar softvera i tako utvrditi radi li neki dio softvera ispravno. Postupak izvođenja ručnog testiranja se provodi praćenjem sedam koraka, koraci i njihov redoslijed su vidljivi na Slika 2.2. Test se smatra uspješnim ako je nakon izvođenja svakog koraka testnog slučaja nije pronađena greška.

Ručno testiranje provode testeri izravnom interakcijom sa softverom kojeg testiraju. Tijekom ručnog testiranja pronalaze se slučajne, ne tako očite, radnje koje korisnici mogu napraviti. Slučajne odnosno neplanirane radnje koje korisnik može koristiti su rezultat greške u softveru. Uz otkrivanje grešaka i ako je planirano automatizirano testiranje softvera, testeri mogu slagati plan automatizacije testiranja. Plan se može sastojati od: najkritičnijih procesa, procesa u kojima su pronađene greške, koraci kako ponoviti te greške i lista koraka svakog procesa koji se testira. Plan uvelike olakšava kasniju automatizaciju testiranja.

Postoje dva ključna načina kako se ručno testiranje može primijeniti, horizontalni i okomiti pristup. Svaki ima svoje prednosti i zahtjeve koje ispunjava.

Horizontalni pristup pokriva testiranje cijelog softvera. Zahtjeva točno definiran proces testiranja i okruženje u kojem se testira. Jedan proces može obuhvatiti više podsustava softvera tijekom izvođenja. Ideja ovog pristupa je testiranje softvera kao cjeline, odnosno svih spojenih servisa u cjelini, kako bi se utvrdilo ispravno funkcioniranje. Primjer je internet aplikacija koja se sastoji od: korisničkog sučelja, baze podataka i raznih vanjskih servisa..

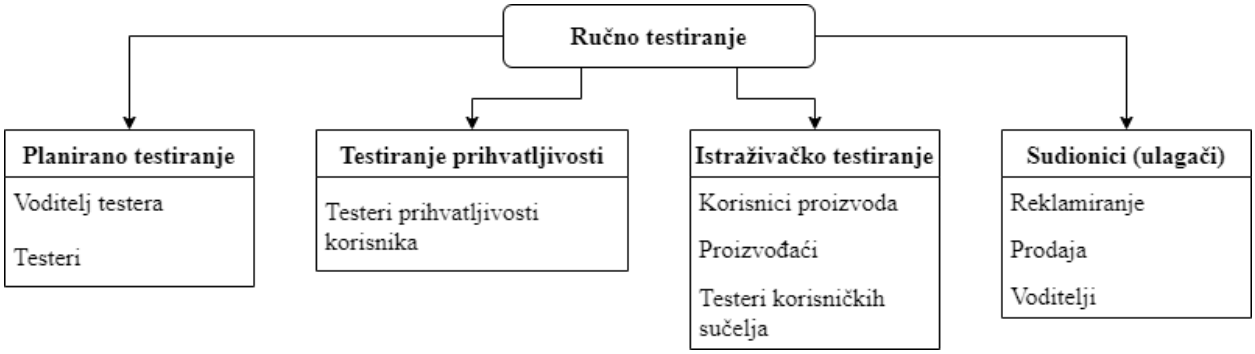
Okomiti pristup dijeli softver na više slojeva pri čemu se svaki sloj testira odvojeno. Upravo zbog testiranja u slojevima okomiti pristup prethodi horizontalnom. Provodeći testiranje okomito testiraju se pojedini procesi unutar softvera. Tako se greške mogu pronaći brže i lakše. Primjer je testiranje navigacije putem korisničkog sučelja internet aplikacije.



Slika 2.2 *Postupak ručnog testiranja.*

Kroz povijest testiranja softvera definirali su se mnogi načini testiranja. Svaki način primjenjiv je tijekom određene razine razvoja softvera. Svaki način ima drugačiji pristup testiranju, kao i ciljanu skupinu testera. Koje skupine testera odgovaraju za koji način testiranja je vidljivo na Slika 2.3. Način testiranja koji se najviše koristi je način istraživačkog testiranja. Taj način nema točno

propisane korake koji se trebaju pratiti. Ideja ovog načina je da se slučajnim korištenjem softvera pronađu situacije koje nisu predviđene tijekom dizajniranja i proizvodnje.



Slika 2.3 *Ciljane skupine testera za neke od načina ručnog testiranja.*

Tablica 2.1 *Prednosti i nedostatci ručnog testiranja.*

Prednosti	Nedostatci
Niža razina obrazovanja radne snage	Ne otkriva sve greške
Raznovrsno, lako primijeniti bilo koji način	Kompleksni proces zahtijeva iskusne testere
Blisko pravom iskustvu korisnika	Dugotrajno, svako ponavljane testa se zbraja
Lako promjenjivo, lako se rješavaju promjene tijekom razvoja softvera	Loša kvaliteta u slučaju velike količine testova
Jedino ulaganje je vrijeme	Nemoguće ponovno iskoristiti

Prednosti ručnog testiranja, vidljivi u Tablica 2.1, pokazuju da je cijena ručnog testiranja u kratkom razdoblju relativno niska, a rezultati su veoma efektivni. Potrebna razina znanja za provođenje ručnog testiranja je niža od razine koja je potrebna za automatizirano testiranje. Tijekom ručnog testiranja tester može pronaći situacije na koje se nisu definirale tijekom dizajniranja softvera.

2.6.2 Automatizirano testiranje

Kontinuiranim razvojem softvera, izvođenjem svakog testnog slučaja ručno postaje sve manje održivo i sve skuplje. Testiranje koristeći korisničko sučelje postaje skoro nemoguće. Razvojem softvera dolazi do sve većeg izbora akcija koje korisnik može odabrati. To može dovesti i do ne izvođenja nekih testnih slučajeva. Velika količina testova postaje teška za praćenje. Zbog toga je neophodno uvesti automatizaciju testova.

Automatizirano testiranje je vještina različita od ručnog testiranja. Skuplje je od ručnog testiranja. Kako bi se postigla korist od automatizacije, testni slučajevi moraju biti pažljivo izabrani prije implementiranja. Nakon odabira i implementacije, automatizirani test postaje puno ekonomičniji, jer jedini troškovi nakon toga mogu biti troškovi za kratko ponavljanje testa i održavanje.

Ekonomičnost automatiziranih testova dolazi iz njihovog ponavljanja. Što se više koriste to su isplativiji.

Koji testni slučaj automatizirati ovisi o tome koliko je kompleksan za izvođenje. Međutim, to nije jedini kriterij prema kojem treba odlučivati. Broj ponavljanja i količina korištenja tog procesa su jednako važni. Proces s velikim brojem radnji i akcija koje korisnik može izabrati, a rezultat nije kritičan korisniku, spada nisko u hitnosti automatizacije. Proces koji se često koristi je na vrhu liste za automatizaciju, bez obzira na složenost procesa.

Automatizirano testiranje se odvija pokretanjem određenih programa. Programi prate napisanu listu radnji koji definiraju jedan test. Lista radnji koja se izvršava naziva se skripta. Izvođenjem skripti dolazi se do određenih prednosti u odnosu na ručno testiranje, prednosti kao i nedostatci prikazani su u Tablica 2.2. Automatizirano testiranje zapravo imitira ručno testiranje, ali izbacuje tri ključna loša elementa ručnog testiranja, ljudsku pogrešku, nejedinstvenost izvođenja istog testa i dugotrajnost. Kako bi se uspješno izbacile mane ručnog testiranja iz automatiziranoga potrebno je imati iskusne testere.

Tablica 2.2 *Prednosti i nedostatci automatiziranog testiranja.*

Prednosti	Nedostatci
Brzina izvođenja	Brzina pisanja novih testova
Točnost i pouzdanost	Nedostatak fleksibilnosti
Ponovna iskoristivost	Teško pisanje testova za jedinstvene slučajeve
Efikasno za velike količine testova	Skupa radna snaga

Odabir načina izvođenja automatiziranog testiranja provodi se imajući na umu koji je cilj potrebno zadovoljiti. Različiti ciljevi zahtijevaju drugačije pristupe. U nastavku su opisani neki od načina i koje zahtjeve ispunjavaju.

- **Testiranje vođeno podacima** –zahtjeva da se testni podaci spremaju odvojeno od skripte. Na taj način je vrlo lako promijeniti koji se podaci koriste za testiranje. Te datoteke se spremaju u bazu podataka, lokalnu ili udaljenu, lokalno na računalo ili nešto slično. Tipovi datoteka koje se mogu koristiti su: *CSV*, tekst, *JSON, XML* i sl.
- **Testiranje opterećenjem** – koristi se kako bi se utvrdilo normalno ponašanje softvera tijekom normalnog rada i rada pod punim opterećenjem. Tako se mogu pronaći komponente koje treba optimizirati kako bi se povećale performanse.
- **Testiranje performansi** – ovaj način se koristi za testiranje brzine odziva i stabilnosti softvera.

- **Testiranje sigurnosti** –testiranje propusta u sigurnosti softvera. Sigurnost podataka korisnika je glavni zahtjev svakog korisnika koji koristi softver.
- **Regresijsko testiranje** –ponavljanje starih testova koji su prolazili kako bi se utvrdilo da nije došlo do promjena ponašanja aplikacije tijekom razvoja drugih komponenti softvera.

Za dobro definiranje automatiziranih testnih slučajeva potrebno je pratiti određene faze razvoja automatiziranih testnih slučajeva. Te faze su:

1. odabir alata za testiranje,
2. definiranje opsega automatizacije,
3. planiranje, dizajn i razvoj testova,
4. izvođenje testnih slučajeva,
5. održavanje testova.

Odabir alata za testiranje – uvelike ovisi o tehnologiji i tipu softvera koji se testira.

Definiranje opsega automatizacije – odabire se koje se komponente softvera testiraju i u kojoj mjeri. Neke od stavki za pomoć pri određivanju su: koliko je značajka važna za poslovanje, kolika je potrebna količina podataka za testiranje, je li tehnički izvedivo, sposobnost ponovnog korištenja testa, kolika je tehnička složenost i slično.

Planiranje, dizajn i razvoj testova – definiraju se strategije pisanja testnih slučajeva i plan automatizacije. Oni sadrže: odabrane alate, značajke alata, automatsku pripremu područja koje se testira, raspored izvođenja testova, isporučene rezultate.

Izvođenje testnih slučajeva - izvođenje testnih skripti te spremanje i dokumentiranje njihovih rezultata.

Održavanje testova – softver tijekom razvoja integrira nove funkcionalnosti koje mogu narušiti ispravnost postojanih testnih slučajeva. Provođenjem regresijskog testiranja moguće je otkriti gdje su nove funkcionalnosti narušile stare.

2.7 Testiranje od kraja do kraja

Testiranje od kraja do kraja (engl. *End to End testing, E2E*) [3] je način testiranja softvera kojim se utvrđuje nepostojanost grešaka na svim razinama softvera. Testira se tok podataka za sve moguće operacije koje korisnik može odraditi tijekom korištenja softvera. Ovaj način testiranja je najbliži iskustvu korisnika. Testira se od samog početka korištenja softvera pa do krajnjeg cilja. Cilj može biti bilo koja točka unutar softvera. Testni slučajevi mogu biti jednostavni ili jako

kompleksni. Primjer jednostavnog testnog slučaja je prijava na internet stranicu koristeći email i lozinku. Dok kompleksniji mogu biti više jednostavnih testnih slučajeva spojeni u jedan veći testni slučaj. Jedan od glavnih ciljeva ovog tipa testiranja je utvrditi točan, siguran i robustan tok operacija koje korisnik može odrađivati.

Testiranje od kraja do kraja koristi se praksi sve češće, a neki od razloga za to su:

- povećava pokrivenost softvera testovima jer spaja funkcionalnosti koje se putem testova jedinica i funkcijskih testove ne mogu testirati,
- utvrđuje da softver funkcionira prema zahtjevima korisnika (testni slučajevi su kreirani vodeći računa o tome kako krajnji korisnik koristi softver),
- pomaže u otkrivanju grešaka u komunikaciji između vanjskih servisa i softvera.

Ovaj način testiranja se najviše koristi u razvojnom okruženju gdje su prisutni odvojeni timovi programera, testera, menadžera i korisnika. Tako se osigurava da svaki tim ima određeni posao koji rade i samo na njega se moraju fokusirati. Programeri u tom slučaju ne moraju brinuti o utvrđivanju radi li softver kako je predviđeno prema dizajnu, nego to osiguravaju testeri. Još jedna od prednosti razdvojenih timova je drugačija perspektiva tijekom testiranja. Gledajući na softver iz stajališta korisnika, testeri mogu lako definirati ključne komponente softvera. Tako dolaze do zaključka koje komponente treba testirati intenzivnije. Tu informaciju prosljeđuju programerima kako bi se osigurao dobar nastavak razvoja. Iz tog razloga iznimno je ključna komunikacija između timova.

Izazovi i problemi koje predstavlja testiranje od kraja do kraja su slični kao i u drugim načinima testiranja. Za provođenje kvalitetnog testiranje potrebno je uložiti vrijeme, testni slučajevi koji se izvršavaju moraju biti dizajnirani tako da predstavljaju scenarij iz stvarnog svijeta. Kako bi se testni slučaj mogao dizajnirati na taj način, bitno je shvatiti ciljeve korištenja krajnjeg korisnika. Tako se dolazi do kružnog efekta. Svaki od izazova utječe na prethodni. Shvaćanje ciljeva znači veći utrošak vremena, što znači dizajniranje više detaljnijih testnih slučajeva.

Tijekom razvoja softvera, ovaj način testiranja nije dobar ako se očekuje brza povratna informacija. Programeri tijekom razvoja novih značajki softvera trebaju povratne informacije vezane uz te značajke. To nije cilj ovog načina testiranja. Cilj je testirati značajke koje će krajnji korisnici koristiti. Jedan od najtežih izazova za savladati tijekom testiranja od kraja do kraja je baš taj.

2.7.1 Implementacija

Za uspješnu implementaciju dobrih testnih slučajeva najbitnije je razumjeti kojim će se značajkama krajnji korisnik koristiti. To se može postići na način da se softver prvo testira samostalno ili uz pomoć potencijalnog korisnika. Ovaj način testiranja zahtjeva određenu pripremu pa postoji nekoliko koraka koje je dobro pratiti za dobru definiciju testnog slučaja:

- pregledati potrebne zahtjeve za ocjenjivanje rezultata testnog slučaja,
- pripremiti testno okruženje i zahtjeve,
- definirati sve potrebne procese softvera i spojenih servisa,
- opisati uloge svakog procesa softvera i spojenih servisa,
- odabrati alate za testiranje,
- definirati ulazne i izlazne podatke za svaki proces softvera.

Nakon provođenja gore definiranih koraka može se započeti s implementacijom testnih slučajeva u odabranom alatu za testiranje.

3 ALATI ZA AUTOMATIZIRANO TESTIRANJE

Alati za automatizirano testiranje su programi dizajnirani za verifikaciju i validaciju funkcionalnih i ne funkcionalnih zahtjeva pomoću skripti. Izbor pravoga alata za određeni problem nije jednostavan, ovisi o projektu na kojem se radi i sposobnostima tima koji testira softver. Najbolji alat općenito ne osigurava najbolji rezultat tijekom testiranja. Razvojem tržišta dolazi do novog zahtjeva, kvalitete u brzini (engl. *Quality at Speed*) [4]. Cilj ovog zahtjeva je dostaviti proizvod visoke kvalitete krajnjem korisniku u što kraćem vremenskom roku. Testni slučajevi koji zahtijevaju veliki broj ponavljanja istih ili sličnih akcija najveći su prioritet za automatizaciju. Regresijsko testiranje još je jedan slučaj gdje je automatizacija iznimno bitna.

Alati za automatizirano testiranje mogu se svrstati u tri glavna tipa koja su navedena u nastavku.

- Alati otvorenog koda – besplatne platforme koje omogućavaju korisnicima pristup programskom kodu alata. Korisnici se tako mogu odlučiti za cijelu platformu ili odabrati samo onaj dio koji im je potreban.
- Komercijalni alati – napravljeni su kako bi služili u komercijalne svrhe i najčešće imaju razne planove pretplata. Korisnici moraju kupiti licencu za korištenje platforme. Za razliku od besplatnih platformi, imaju dodatne značajke i temeljitu korisničku službu.
- Prilagođeni alati – postoje projekti gdje jedan alat nije dovoljan, odnosno ne može pokriti sve zahtjeve. U tom slučaju, testeri razvijaju prilagođeni alat. Rezultat je kompleksniji alat koji ispunjava točne zahtjeve projekta.

Raznolik izbor alata s različitim značajkama donosi novi problem - koji alat odabrati? Prilikom biranja alata za automatizirano testiranje potrebno je prvo ocijeniti određene kriterije. Karakteristike i značajke pet najpopularnijih alata za automatizirano testiranje mogu se vidjeti u Tablica 3.1.

1. Posjeduje li tim potrebna znanja i vještine?
2. Koliki je budžet tima?
3. Koje su potrebne značajke?
4. Koliko je kompleksno održavanje i ponovna uporaba skripti?
5. Je li alat moguće integrirati unutar *CI/CD*[1] cjevovoda?
6. Kako i gdje pronaći tehničku podršku?

[1] CI/CD (engl. *Continuous Integration/ Continuous Deployment*) – kontinuirana integracija / kontinuirana implementacija

Tablica 3.1 *Usporedba pet najpopularnijih alata za automatizirano testiranje softvera.*

Proizvod	Katalon Studio	Selenium	Appium	TestComplete	Cypress.io
Aplikacije koje je moguće testirati	Internet, API, mobilne, računalne	Internet	Mobilne	Internet, mobilne, računalne	Internet, API
Podržani OS	Windows, macOS, Linux	Windows, macOS, Linux, Solaris	Windows, macOS	Windows	Windows, macOS, Linux
Postavljanje i konfiguriranje	Jednostavno	Potrebno kodiranje	Potrebno kodiranje	Jednostavno	Potrebno kodiranje
Niska razina kodiranja i pisanje skripti	Oboje	Samo skripte	Samo skripte	Oboje	Samo skripte
Podržani jezici	Java i Groovy	Java, C#, Python, JavaScript, Ruby, PHP, Perl	Java, C#, Python, JavaScript, Ruby, PHP, Perl	JavaScript, Python, VBScript, JScript, Delphi, C++, C#	JavaScript, TypeScript
Napredni rezultati testiranja	Da	Ne	Ne	Ne	Da
Cijena	Besplatan ili komercijalan	Besplatan	Besplatan	Komercijalan	Besplatan ili komercijalan

3.1 Selenium

Selenium je alat otvorenog koda za automatizirano testiranje internet aplikacija [5]. Podržava testiranje na različitim internet preglednicima i platformama. Jedan je od najpopularnijih alata te se smatra industrijskim standardom. *Selenium* nudi veliku fleksibilnost prilikom testiranja. Testove je moguće pisati raznim jezicima i na raznim operacijskim sustavima. Kako bi se alat što bolje iskoristio, testeri moraju posjedovati napredne vještine programiranja. Uz to, potrebno je uložiti puno vremena na razvoj okvira za automatiziranje i biblioteka potrebnih za automatizaciju. Navedeni dug razvoj testnih slučajeva je glavni nedostatak *Selenium*-a.

Selenium je skup četiri programa koji zadovoljavaju različite zahtjeve testiranja. *Selenium* server jedan je od tih programa i on omogućuje pokretanje testova lokalno ili na udaljenom računalu.

3.2 Katalon Studio

Katalon Studio [5] nudi razvoj testnih slučajeva koristeći nisku razinu kodiranja te podržava pisanje skripti. Niska razina kodiranja je zapravo način razvoja softvera putem već gotovih funkcijskih blokova koji se međusobno spajaju, na engleskom je poznato kao *Low-code* ili *Drag and drop programming*. Može se koristi za pisanje automatiziranih testova za internet aplikacije i stranice, *API*, računalne i mobilne aplikacije. Kako bi olakšao pristup početnicima, alat nema posebni postupak postavljanja te nije potrebno programiranje. U tu svrhu posjeduje implementiranu mogućnost snimanja koraka i ponavljanje istih, umjesto pisanja skripti. Podržava *Java* i *Groovy* programske jezike i mogućnost jednostavne integracije u *CI/CD* cjevovod.

3.3 Appium

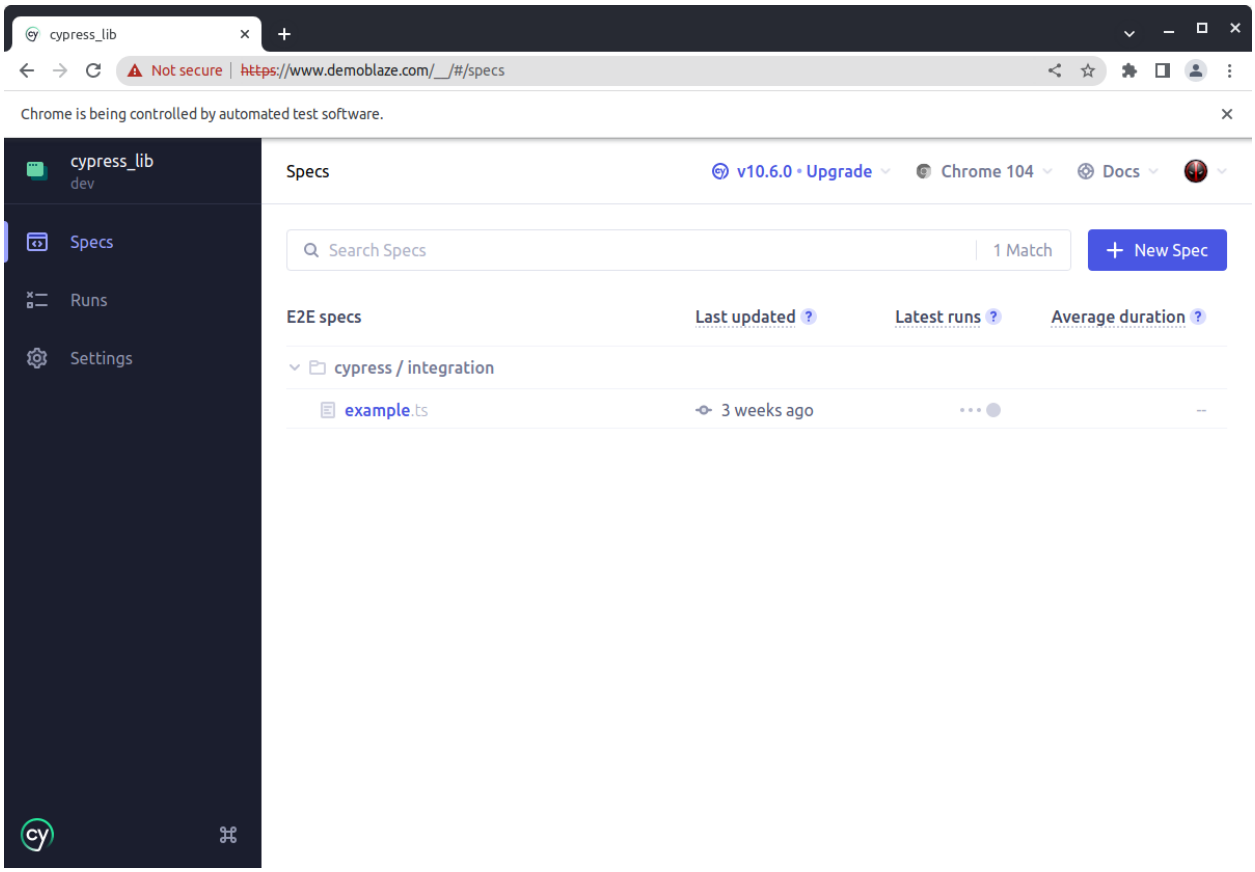
Appium [5] je alat otvorenog koda koji se koristi za pisanje automatiziranih testova mobilnih aplikacija za Android i iOS platforme. Podržava velik izbor programskih jezika za pisanje skripti. Može se koristiti na stvarnim uređajima, simulatorima i emulatorima bez ikakvih promjena napisanih skripti. Podržava integraciju s ostalim alatima za testiranje te integraciju na *CI/CD* cjevovod.

3.4 TestComplete

TestComplete [5] je komercijalni alat za automatizirano testiranje. Dostupan je na Windows platformi. Glavna značajka je prepoznavanje objekata i pisanje skripti uz pomoć umjetne inteligencije za vizualno prepoznavanje, što nije dostupno kod većine besplatnih alata.

3.5 Cypress

Cypress je alat za automatizirano testiranje bilo čega što je moguće otvoriti unutar internet preglednika. Javno je dostupan, otvorenog koda i besplatan pod licencom MIT [6]. MIT licenca dozvoljava korisnicima ponovnu uporabu koda u bilo koje svrhe. Princip rada većine alata za automatizirano testiranje je takav da se naredbe iz skripti odvijaju izvan internet preglednika, u posebnom okruženju. *Cypress* izvodi sve naredbe unutar preglednika, a trenutno podržava: *Chrome*, *Electron*, *Chromium*, *Mozzila Firefox* i *Microsoft Edge* [7]. *Cypress* se sastoji od dvije komponente, *Test Runner* i *Dashboard*.



Slika 3.1 *Cypress Test Runner*.

Test Runner, prikazan na Slika 3.1, izvodi testne slučajeve u jedinstvenom interaktivnom programu. Program je zapravo internet preglednik s dodatnim funkcionalnostima. Neke od njih su: pregled naredbi koje su izvršene ili se trenutno izvršavaju, pregled trenutnog prozora aplikacije koja se testira i pregled pohranjenih podataka o stanju aplikacije za svaku naredbu koja se izvršila.

Koristi se *JavaScript* programskim jezikom po standardnim postavkama alata. Uz to je moguće pisati testove u *TypeScript* programskom jeziku uz nekoliko malih promjena konfiguracije *Cypress*-a.

3.5.1 Značajke

Cypress posjeduje mnoge značajke dostupne i kod drugih alata, ali i nekoliko jedinstvenih. Razvoj alata je i dalje u tijeku. To znači da postoji kontinuirani razvoj novih značajki koje unaprjeđuju mogućnosti alata. Neke od značajki su navedene u nastavku [7].

- „Putovanje kroz vrijeme“ – tijekom izvođenja testova, nakon svake naredbe, *Cypress* pohranjuje cijelo stanje aplikacije u snimku (engl. *snapshot*). Pomoću toga se može provjeriti stanje aplikacije nakon svake naredbe.

- Automatsko čekanje – jedan od najvećih problema tijekom testiranja internet aplikacija je kako savladati čekanje da se podaci pojave ili da neka druga akcija završi prije izvođenja trenutne. *Cypress* automatski čeka završetak određene naredbe prije iduće naredbe. To rješava većinu asinkronih problema.
- Špijuni (engl. *spys*), stubovi (engl. *stubs*), satovi – mogućnost provjere i kontroliranja: ponašanja funkcija, odgovora servera i mjerača vremena (engl. *timer*).
- Snimke zaslona i video – mogućnost pregleda snimki zaslona u trenutku otkrivanja greške i videa izvođenja testa koji je pao.
- Brzo i jednostavno postavljanje testova – nema potrebe za instalacijom dodatnih servera ili programa.

3.5.2 Cypress metode

Cypress testerima omogućava brzo i jednostavno razvijanje novih testnih slučajeva. To je moguće zahvaljujući velikom izboru dobro definiranih metoda uz već poznate metode određenih alata. Kontinuiranim razvojem alata dolazi se do novih metoda, ali one glavne, prvotne su i dalje najkorisnije. Neke od metoda koje se koriste u ovom radu su navedene u nastavku kao i kratka objašnjenja što rade [8].

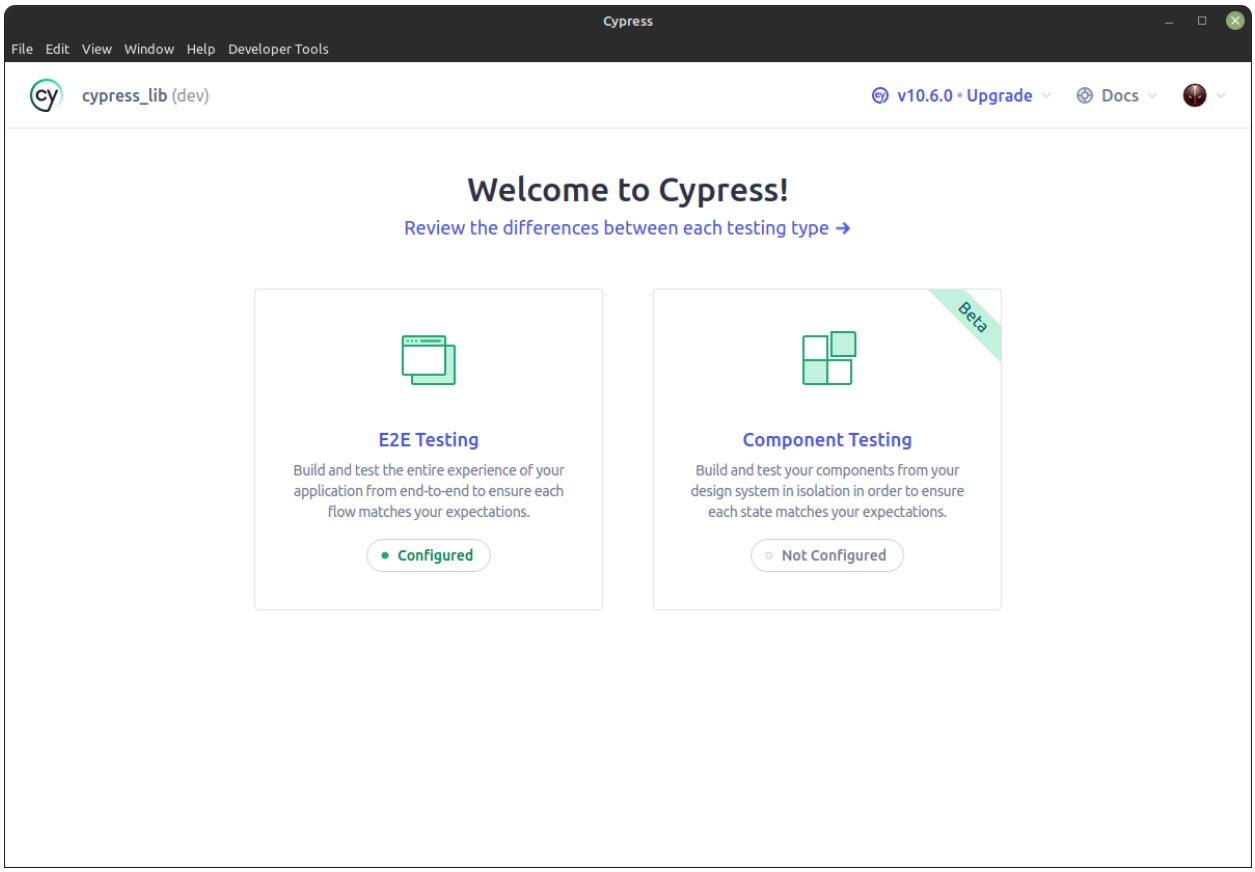
within -

visit – koristi se za posjećivanje internet stranice putem *URL*-a. Prima minimalno jedan argument, a moguće je maksimalno dva. Mogući argumenti su *url* i *options*. Povratni podatak metode je *window* objekt posjećene stranice. Ima mogućnost vezanja većine metoda, to je rezultat povezanosti metoda preko istog sučelja.

url – dohvaća trenutni *url* stranice. Prilikom pozivanja može primiti najviše jedan argument, *options*. To je argument koji dijele skoro sve metode unutar *Cypress*-a. Sadrži više opcija koje se mogu koristiti ako je potrebno posebno ponašanje metode.

3.5.3 Tipovi testova

Tijekom testiranja, velika je prednost ukoliko je moguće većinu značajki softvera testirati pomoću jednog alata. Ako je riječ o internet aplikacijama, *Cypress* nudi veliki izbor mogućih tipova testiranja. Razlog tome je što se svi testovi koji se izvode pomoću *Cypress*-a odvijaju unutar internet preglednika. Prilikom pokretanja alata pruža se mogućnost odabira tipa testiranja (Slika 3.2).



Slika 3.2 *Cypress početni prozor, izbor tipa testiranja.*

Osim za dva glavna tipa testiranja, alat je moguće koristi za testiranje sličnih procesa. Najčešći tipovi testiranja su: testiranje od kraja do kraja, testiranje komponenti, *API* testiranje. Navedeni tipovi su opisani u nastavku.

- Testiranje od kraja do kraja – u početku razvoja, *Cypress* je bio dizajniran za izvođenje *E2E* testova na bilo čemu što se može pokrenuti unutar internet preglednika. Najobičniji *E2E* test posjećuje aplikaciju u pretraživaču te obavlja akcije na korisničkom sučelju kao pravi korisnik, primjer *E2E* test slučaja prikazan je na Slika 3.3.

```
1.  it('E2E test', () => {
2.      cy.visit('https://www.demoblaze.com/');
3.      cy.contains('Sign up').click();
4.      cy.get('#sign-username').clear().type('test@test.com');
5.      cy.get('#sign-password').clear().type('test12345');
6.      cy.contains('Sign up').click();
7.  });
```

Slika 3.3 *Primjer E2E testnog slučaja.*

- Testiranje komponenti – u verziji 7.0.0 [9] dodana je podrška za testiranje pojedinih komponenti. Time je omogućeno ubacivanje komponenti podržanih razvojnih okvira, a to su trenutno: *Vue*, *React* i *Angular*, primjer prikazan na Slika 3.4.

```
1.  import Component from './components/Component';
2.
3.  it('Component test', () => {
4.      const component = [
5.          {text: "First name", id: 1},
6.          {text: "Last name", id: 2}
7.      ];
8.
9.      cy.mount(<Component component = {component}/>);
10.     cy.get('[testid="component"]').should('have.length', component.length);
11. });
```

Slika 3.4 *Primjer test slučaja komponente.*

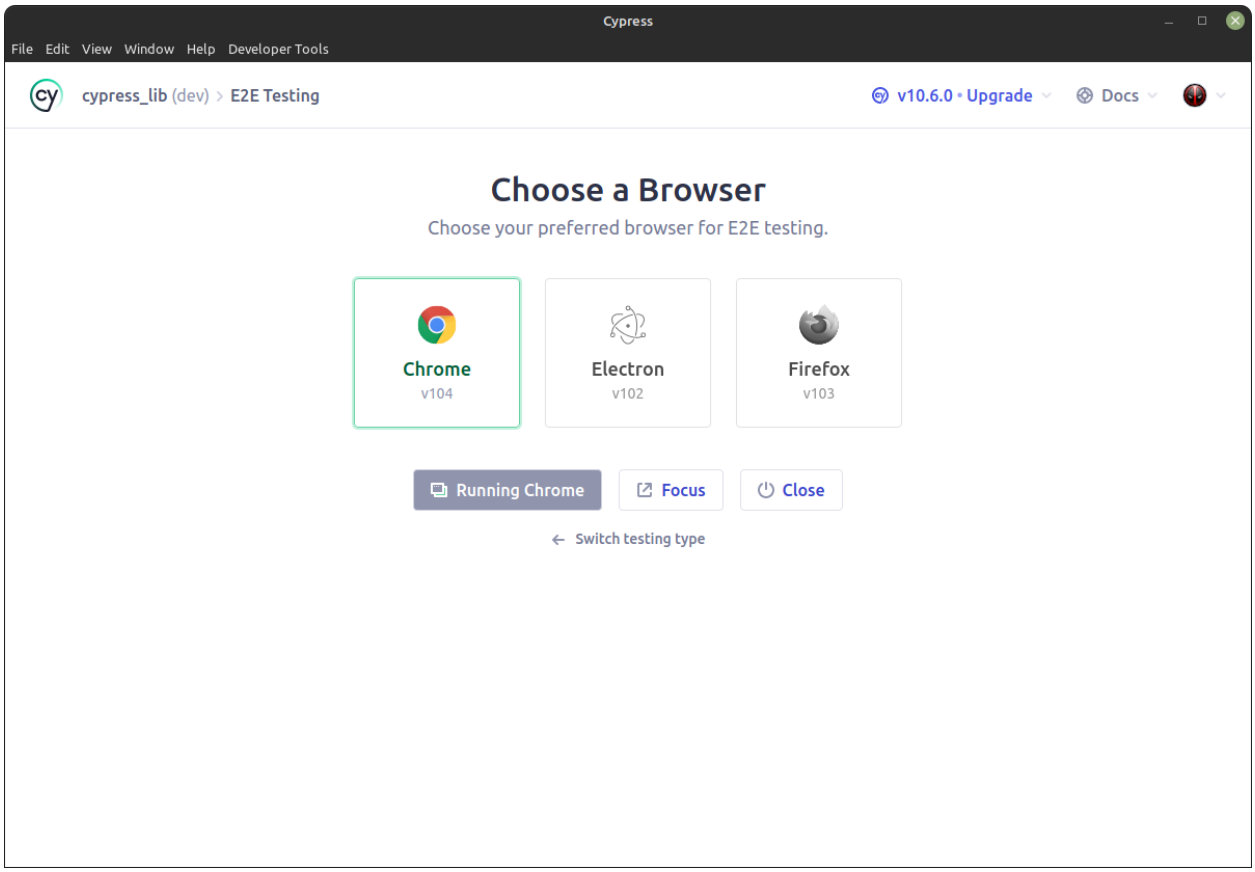
- *API* testiranje – u mogućnosti *Cypress*-a da testira bilo što što se može pokrenuti u pregledniku, spada i testiranje *API*-ja. Time *Cypress* može izvoditi proizvoljne *HTTP* pozive i tako testirati *API*. Primjer *API* testnog slučaja prikazan je na Slika 3.5.

```
1.  it('API test', () => {
2.          cy.request({
3.             url: '/login',
4.             method: 'POST',
5.             body: {
6.                 email: 'test@test.com',
7.                 password: 'test12345'
8.             },
9.          });
10.     });
```

Slika 3.5 *Primjer API testnog slučaja.*

- Ostali – uz gore navedene tipove testiranja, korištenjem dodataka *Cypress* omogućuje testiranje različitih procesa i sustava.

Nakon odabira tipa testiranja otvara se prozor gdje se nudi izbor internet preglednika u kojem se pokreće *Test Runner*-a. Predstavljeni izbor sastoji se od najpopularnijih i najbolje podržanih pretraživača (Slika 3.6).



Slika 3.6 *Izbor internet pretraživača za E2E testiranje.*

4 CYPRESS BIBLIOTEKA

U sklopu ovog diplomskog rada razvijena je *Cypress* biblioteka koja testerima olakšava korištenje *Cypress*-a. Instalacijom *Cypress*-a dobivaju se integrirane osnovne funkcije koje se mogu slagati međusobno i tako obavljati zadatak testiranja. Testeri tijekom razvoja automatiziranih testova na većim projektima, ili radom na više manjih, često dolaze do zaključka da većinom koriste slične kombinacije osnovnih funkcija kako bi odradili određeni zadatak. Prema tome, nameće se potreba za proširivanjem *Cypress* alata s dodatnim funkcijama koje će ubrzati razvoj automatiziranih testova.

4.1 Motivacija

Tijekom automatiziranog regresijskog testiranja često se pojavljuje problem da testovi ne prolaze jer se tijekom razvoja softvera, značajka promijenila i više ne funkcionira po prijašnjem dizajnu. U takvim slučajevima važna je uloga testera koji mora takve automatizirane testne slučajeve popraviti. Međutim, to nije uvijek jednostavno izvedivo. Jedan od problema ponekad može biti to što su testni slučajevi i testni podaci pisani u istu datoteku te se na više mjesta se ponavljaju iste funkcije. Funkcije koje se koriste na više mjesta, definiraju se gdje god su potrebne što stvara dodatnu konfuziju tijekom popravljanja testnih slučajeva.

Lokaliteti također mogu predstavljati problem. Ukoliko su svi podaci zapisani unutar testnih slučajeva, njihova prilagodba da funkcioniraju na drugim lokalitetima gotovo je nemoguća. Primjere navedenih problema može se vidjeti uspoređivanjem programskog koda prikazanog na Slika 4.1 i Slika 4.2. Osim razlike u jezicima, dužina linije koda isto predstavlja problem.

```
1.  it('Registration fail - English local', () => {
2.      cy.visit('https://wordpress.com/');
3.      cy.contains('Start your website').click();
4.      cy.get('h1').should('contain.text', 'Let’s get started');
5.      cy.get('#email').type('test@test.com');
6.      cy.get('#username').type('test');
7.      cy.get('#password').type('Test12345!');
8.      cy.get('.form-input-validation .is-error').should('contain.text', 'You cannot
    use that email address to signup. There are problems with them blocking some emails
    from WordPress. Please use another email provider.')
9.      cy.contains('Create your account').click();
10. });
```

Slika 4.1 *Primjer testnog slučaja za internet stranice na engleskom jeziku.*

```
1.  it('Registration fail - German local', () => {
2.      cy.visit('https://wordpress.com/de/');
3.      cy.contains('Deine Website erstellen').click();
4.      cy.get('h1').should('contain.text', 'Lass uns anfangen!');
5.      cy.get('#email').type('test@test.com');
6.      cy.get('#username').type('test');
7.      cy.get('#password').type('Test12345!');
8.      cy.get('.form-input-validation .is-error').should('contain.text', 'Du kannst
    dich mit dieser E-Mail-Adresse leider nicht registrieren, da der Anbieter manche E-
    Mails von WordPress blockiert. Verwende bitte einen anderen E-Mail-Anbieter.')
9.      cy.contains('Konto erstellen').click();
10. });
```

Slika 4.2 *Primjer testnog slučaja internet stranice na njemačkom jeziku.*

Glavni cilj pri razvoju Cypress biblioteke je uvesti standardizaciju pisanja testova. Definiranjem točnih pravila i strukture projekta može se postići veća produktivnost, ponovna uporaba napisanih funkcija i testova. Veća produktivnost je rezultat bolje komunikacije između testera.

Testni slučajevi prikazani na Slika 4.1 i Slika 4.2 prerađeni korištenjem razvijene biblioteke prikazani su na Slika 4.3 Primjer testnog slučaja uz korištenje razvijene Cypress biblioteke.. Kao što je vidljivo na Slika 4.3 Primjer testnog slučaja uz korištenje razvijene Cypress biblioteke. za testne slučajeve se koriste podatkovne datoteke koje su prikazane na Slika 4.4 JSON datoteka s podacima na engleskom jeziku.i Slika 4.5 JSON datoteka s podacima na njemačkom jeziku..

```
1.  import * as dataEN from './../fixtures/exampleEn.json';
2.  import * as dataDE from './../fixtures/exampleDe.json';
3.
4.  it('Registration fail - English local', () => {
5.      registrationFail(dataEN);
6.  });
7.  it('Registration fail - German local', () => {
8.      registrationFail(dataDE);
9.  });
10. function registrationFail(data: any) {
11.     cy.visit(data.url);
12.     cy.contains(data.registerBtn).click();
13.     cy.get('h1').should('contain.text', data.h1);
14.     cy.get('#email').type(data.test.fail.email);
15.     cy.get('#username').type(data.test.fail.username);
16.     cy.get('#password').type(data.test.fail.password);
17.     cy.get('.form-input-validation .is-error').should('contain.text',
    data.messages.failEmail);
18.     cy.contains(data.confirmBtn).click();
19. }
```

Slika 4.3 *Primjer testnog slučaja uz korištenje razvijene Cypress biblioteke.*

```json
{
  "url": "https://wordpress.com/",
  "registerBtn": "Start your website",
  "h1": "Let’s get started",
  "confirmBtn": "Create your account",

  "messages": {
    "failEmail": "You cannot use that email address to signup. There are problems with them blocking some emails from WordPress. Please use another email provider."
  },
  "form": {
    "inputFields": {
      "email": "#email",
      "username": "#username",
      "password": "#password"
    }
  },
  "test": {
    "fail": {
      "email": "test@test.com",
      "username": "test",
      "password": "Test12345!"
    }
  }
}
```

Slika 4.4 *JSON datoteka s podacima na engleskom jeziku.*

```json
{
  "url": "https://wordpress.com/de/",
  "registerBtn": "Deine Website erstellen",
  "h1": "Lass uns anfangen!",
  "confirmBtn": "Konto erstellen",

  "messages": {
    "failEmail": "Du kannst dich mit dieser E-Mail-Adresse leider nicht registrieren, da der Anbieter manche E-Mails von WordPress blockiert. Verwende bitte einen anderen E-Mail-Anbieter."
  },
  "form": {
    "inputFields": {
      "email": "#email",
      "username": "#username",
      "password": "#password"
    }
  },
  "data": {
    "fail": {
      "email": "test@test.com",
      "username": "test",
      "password": "Test12345!"
    }
  }
}
```

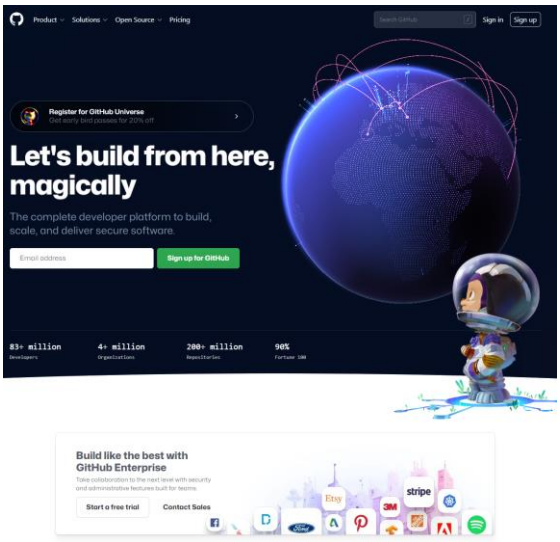
Slika 4.5 *JSON datoteka s podacima na njemačkom jeziku.*

4.2 Realizacija

Za realizaciju biblioteke potrebno je prvo definirati točne ciljeve koje je potrebno ispuniti. Za točno i precizno definiranje ciljeva potrebno je iskustvo u području automatiziranja testiranja. U nastavku je navedene okvirna lista značajki, odnosno ciljeva, koje je potrebno zadovoljiti prilikom razvoja biblioteke:

- jednostavna instalacija,
- jednostavne instrukcije,
- dostupna dokumentacija,
- proširivost,
- mogućnost korištenja na više načina,
- pouzdanost,
- omogućavanje brzog razvoja točnih i robusnih automatiziranih testova,
- definiranje jedinstvenih pravila (standarda po kojem se razvijaju testni slučajevi).

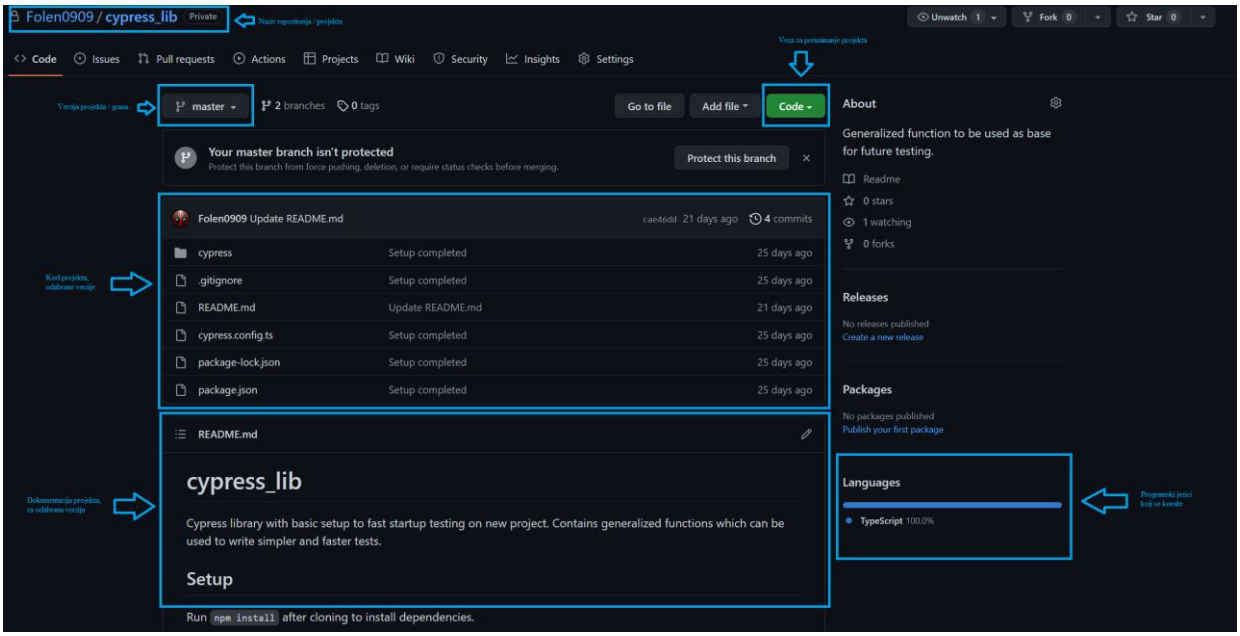
Pomoć pri ostvarivanju nekoliko ciljeva pruža alat za verzioniranje koda. Odabrani alat za verzioniranje koda je *Github* (Slika 4.6) [10]. Prednost ovog alata je njegova iznimna popularnost i pouzdanost, kao i skup značajki koje nudi. Još jedna od glavnih prednosti je ta što je besplatan za korištenje.



Slika 4.6 *Početna stranica Github-a.*

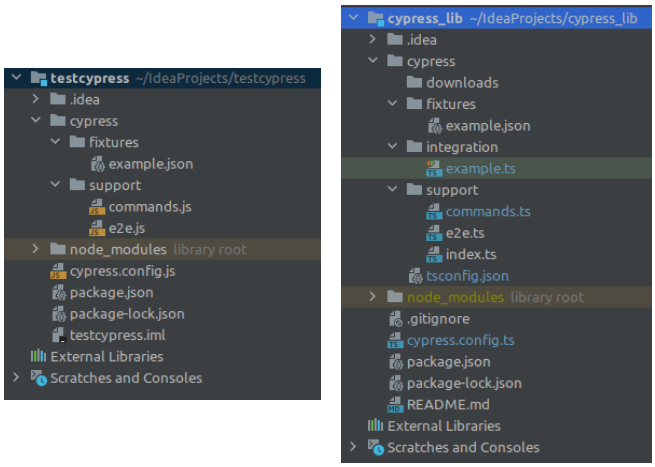
Kreiranjem novog projekta, koji predstavlja implementaciju biblioteke, i postavljanjem na Github, ostvaruje se mogućnost jednostavnog i brzog dijeljenja. Jednostavno praćenje promjena na projektu kao i mogućnost odabira točne verzije biblioteke koja se želi preuzeti. Dokumentacija

biblioteke je dostupna uz projekt. U dokumentaciji je opisano kako preuzeti biblioteku, kako ju instalirati, opisane su funkcije koje su dostupne za korištenje te je opisana struktura projekta i kako ju proširiti. Na Slika 4.7 može se vidjeti izgled biblioteke unutar alata te neke glavne dijelove i njihov opis.



Slika 4.7 *Projekt na Github-u.*

Kako bi se započeo razvoj ovog projekta, prvo je potrebno preuzimanje i instalacija osnovnog alata za automatiziranje testova, *Cypress*. Nakon instalacije i prvog pokretanja alata struktura projekta je kao na Slika 4.8. Testne datoteke su tipa *JavaScript*. Idući korak je napraviti projekt razumljivijim, što se postiže prebacivanje s *JavaScript* programskog jezika u *TypeScript*.



Slika 4.8 *Struktura projekta - JavaScript lijevo - TypeScript desno.*

TypeScript je nadogradnja na *JavaScript*, a glavna značajka koju pruža su tipovi varijabli, što uvelike pomaže prilikom čitanja i analiziranja programa koji su napisani u njemu. Za prebacivanje

projekta na *TypeScript*, potrebno je prvo instalirati *TypeScript* na računalo, u slučaju da već nije instaliran. Uz to potrebno je dodati novu datoteku unutar *cypress* mape, naziva *tsconfig.json*. U toj datoteci se definiraju pravila za točnu interpretaciju *TypeScript* koda (Slika 4.9). Kao što je vidljivo na Slika 4.8 (lijevo), cijela struktura projekta je u *JavaScript*-u. Nakon dodavanja *TypeScript* konfiguracijske datoteke program se ne može pokrenuti. Potrebne datoteke treba prebaciti u *TypeScript* tip datoteke (Slika 4.8 desno). Svaki korak detaljno je opisan unutar dokumentacije biblioteke.

```
1.  {
2.    "compilerOptions": {
3.      "target": "es5",
4.      "lib": ["es5", "dom"],
5.      "types": ["cypress", "node"],
6.      "resolveJsonModule": true
7.    },
8.    "include": ["**/*.ts", "./src", "cypress.d.ts"]
9.  }
```

Slika 4.9 *tsconfing.json datoteka.*

4.2.1 Funkcije

Glavna komponenta razvijene Cypress biblioteke su napredne funkcije koje se mogu koristiti za brži i jednostavniji razvoj novih automatiziranih testova. Analizom funkcija koje se najviše koriste tijekom testiranja raznih projekta definirano je koje su funkcije prioritet za razvijanje. Prilikom razvoja naprednih funkcija vodilo se računa o tome da budu jednostavne za razumjeti, primjenjive i otvorene poboljšanjima. U sklopu razvijene biblioteke dostupne su sljedeće funkcije, koje su detaljno objašnjene u dokumentaciji:

- *visitBaseUrl*,
- *loginByUI*,
- *loginByRequest*,
- *navigateMenu*,
- *inputData*,
- *paginationSelector*,
- *checkURL*.

Svaka funkcija je osmišljena na način da ubrza pisanje automatiziranih testova. Sve funkcije su dodane unutar *cypress* naredbi kako bi korištenje bilo što jednostavnije i korisnije. Dodavanjem

funkcija u sklop *cypress* naredbi i pružanjem odgovarajućeg povratnog tipa varijable funkcije, omogućeno je ulančavanje više naredbi. Osnovne *Cypress* naredbe većinom koriste sučelje *Chainable*. To jedinstveno sučelje omogućuje korištenje više naredbi zaredom, koristeći prethodni element kao bazu. U nastavku su prikazane i opisane sve funkcije dostupne unutar razvijene biblioteke.

visitBaseURL (Slika 4.10) – pozivom ove funkcije otvara se početna stranice aplikacije. Ne prima argumente, nego vrijednosti varijable uzima iz konfiguracijske datoteke. Povratni tip varijable je *Chainable*.

```
1.  /**
2.   * Navigate to baseUrl set in cypress.config.ts
3.   * @return Chainable<AUTWindow> same as cy.visit() so any function can be chained
    that can be chained on cy.visit()
4.   */
5.  Cypress.Commands.add("visitBaseUrl", () => {
6.      return cy.visit(Cypress.env('baseUrl'));
7.  });
```

Slika 4.10 *visitBaseURL funkcija.*

loginByUI (Slika 4.11) – skup naredbi koje unutar korisničkog sučelja pronalaze polja za unos emaila i lozinke te unose potrebne podatke. Podaci se nalaze unutar *Cypress* okoline, kao dinamičke vrijednosti koje se prosljeđuju tijekom pokretanja programa. Nakon unosa podataka prijavljuje se tako što pronalazi tipku s određenim tekstom ili identifikatorom te ju pritišće. Trenutno stanje internet preglednika sprema se unutar *Cypress* sesije. Spremanjem u sesiju ubrzava se izvođenje naknadnih testnih slučajeva. Razlog tome je što funkcija `cy.session` prvo provjera postoji li sesija spremljena pod nazivom prvog argumenta, te u slučaju da postoji učitava ju iz memorije te preskače izvođenje drugog argumenta. Drugi argument je funkcija prijave.

```
/**
 * Login user using email and password using UI elements.
 * Creates session to speed up next login.
 * @return Chainable<AUTWindow> same as cy.visit() so any function can be chained that can be chained on cy.visit()
 */
Cypress.Commands.add('loginByUI', () => {
    cy.session('CYPRESS_ENV_LOGIN', () => {
        cy.visitBaseUrl();
        cy.get(data.email).clear().type(Cypress.env('email'));
        cy.get(data.password).clear().type(Cypress.env('password'));
        cy.get(data.loginBtn).click();
    });
    return cy.visitBaseUrl();
});
```

Slika 4.11 *loginByUI funkcija.*

loginByRequest (Slika 4.12) – funkcija koja ne koristi korisničko sučelje za interakciju. Prijava korisnika se odvija slanjem posebnog HTTP zahtjeva. Tijelo zahtjeva se definira kao konstanta prije slanja zahtjeva. Tako se osigurava bolja preglednost funkcije. U tijelu zahtjeva se nalaze podaci potrebni za izvođenje zahtjeva, u ovom slučaju to su podaci za prijavu, a dohvaćaju se iz *Cypress* okoline. Kao i kod funkcije *loginByUI*, funkcija cy.session pohranjuje trenutno stanje preglednika unutar sesije kako bi se osiguralo brže izvođenje naknadnih testnih slučajeva.

```
/**
 * Login user using email and password using API request.
 * Creates session to speed up next login.
 * @return Chainable<AUTWindow> same as cy.visit() so any function can be chained that can be chained on cy.visit()
 */
Cypress.Commands.add('loginByRequest', () => {
    cy.session('CYPRESS_ENV_LOGIN', () => {
        // body to be sent in request
        cy.visitBaseUrl();
        const user = {
            email: Cypress.env('email'),
            password: Cypress.env('password')
        };
        cy.request({
            url: 'https://api.demoblaze.com/login',  //URL where login function sends requests from frontend
            method: 'POST',
            body: user
        });
    });
    return cy.visitBaseUrl();
});
```

Slika 4.12 *loginByRequest funkcija.*

navigateMenu (Slika 4.13) – koristi se za navigaciju po elementima. Željeni elementi se šalju kao argument funkcije u obliku polja nizova znakova (engl. *string*). Kako bi ispravno funkcionirala, iznimno je bitno imati ispravnu strukturu testnih podataka.

```
1.  /**
2.   * Use for navigating menu.
3.   * @param menuItemList - list of elements to navigate through
4.   * @param noDropDown (optional) - skip first element in list, default false
5.   */
6.  Cypress.Commands.add('navigateMenu', (menuItemList: string[], navigationBar?:
    string) => {
7.      const clonedList  = Object.assign([], menuItemList);
8.      if (!navigationBar) navigationBar = data.navigationBar;
9.      cy.get(navigationBar).within($nav => {
10.         clonedList.forEach(element => {
11.             cy.contains(new RegExp("^" + element + "$", "g")).first().click();
12.         });
13.     });
14. });
```

Slika 4.13 *menuNavigate funkcija.*

inputData (**Slika *4.14***) – funkcija koja prima dva argumenta, formu koju treba popuniti i podatke s kojim je popunjava. Forma i podaci moraju biti sinkronizirani u strukturi kako bi funkcija ispravno odrađivala zadatak za koji je dizajnirana. Funkcija se sastoji od pet pomoćnih funkcija: *clearFields* (**Slika *4.15***), *checkFieldData* (**Slika *4.16***), *selectFieldData* (**Slika *4.17***), *inputSuggestionFieldData* (**Slika *4.18***) i *inputFieldData* (Slika 4.19).

```
1.  /**
2.   * First clears every input inside form, then enters data to each of inputs.
3.   * @param form is list of all inputs on page that need changing.
4.   * @param data that is used after clearing inputs.
5.   */
6.  Cypress.Commands.add('inputData', (form: any, data: any) => {
7.      clearFields(form);
8.      checkFieldData(form.checkFields, data);
9.      selectFieldData(form.selectFields, data);
10.     inputSuggestionFieldData(form.suggestionFields, data);
11.     inputFieldData(form.inputFields, data);
12. });
```

Slika 4.14 *inputData funkcija.*

Brisanje vrijednosti u polju za unos izvodi se korištenjem funkcije *clearFields*. Ova funkcija se koristi kako bi se osiguralo da se u polje za unos unesu samo željeni podaci. Također, ako postoji padajući izbornik, višestruki izbor ili sličan element koji dinamički otključava ili zaključava određeno polje za unos podataka, potrebno je ukloniti zadane vrijednosti. Ova funkcija je prva u nizu poziva baš iz tog razloga.

```
function clearFields(form: any) {
    clearFieldData(form.inputFields);
    clearSuggestionFieldData(form.suggestionFields);
    uncheckFieldData(form.checkFields);
}

function clearFieldData(inputFields: any) {
    for (let key in inputFields) {
        let field = inputFields[key];
        cy.get(field).clear({force: true});
    }
}

function clearSuggestionFieldData(suggestionFields: any) {
    for (let key in suggestionFields) {
        let field = suggestionFields[key];
        cy.get(field).clear();
    }
}

function uncheckFieldData(checkFields: any) {
    for (let key in checkFields) {
        let field = checkFields[key];
        cy.get(field).uncheck();
    }
}
```

Slika 4.15 *Pomoćna funkcija clearFields i sve pripadajuće funkcije.*

Nakon brisanja vrijednosti u polju za unos te poništavanja vrijednosti svih ostalih polja, prvo se unose podaci za polja s višestrukim izborom i polja za kvačice. U slučaju da postoji izbornik koji otključava ili zaključava određeno polje za unos, potrebno ga je aktivirati čime se omogućava da funkcije koje slijede mogu ispravno odraditi naredbe.

```
function checkFieldData(checkFields: any, data: any) {
    for (let key in checkFields) {
        let field = checkFields[key];
        if(data[key] && (key in data))
            cy.get(field).click();
    }
}
```

Slika 4.16 *checkFieldData funkcija.*

Odabir iz padajućeg izbornika je sljedeći korak popunjavanja podataka, nakon kojega slijedi polje za unos podataka koji nudi izbor ovisno o unesenim podacima. Ta polja imaju posebnu funkciju zbog kašnjenja u prikazu podataka koji se pretražuju. Brzina izvođenja testova je puno veća pa i rezultati koji se dohvaćaju sa servera u vremenima ispod 10 milisekundi su prespori. Zbog toga funkcija ima implementirane presretače podataka. Pomoću njih se čeka da se svi podaci dostave i tek onda nastavi s daljnjim radom testa. Posljednji korak je upis podataka u obična polja za unos podataka.

```
1.  function selectFieldData(selectFields: any, data: any) {
2.      for (let key in selectFields) {
3.          let field = selectFields[key];
4.          if(data[key] != "" && (key in data))
5.              cy.get(field).select(data[key]);
6.      }
7.  }
```

Slika 4.17 *selectFieldData funkcija.*

```
1.  function inputSuggestionFieldData(suggestionFields, data: any) {
2.      for (let key in suggestionFields) {
3.          let field = suggestionFields[key];
4.          if (data[key] != "" && (key in data)) {
5.              cy.get(field).invoke('val', data[key]).then((element) => {
6.                  cy.get(element[0]).trigger('mousedown').then(() => {
7.                      cy.intercept('GET', '**').as(field);
8.                  });
9.                  cy.get(element[0]).trigger('mouseup').then(() => {
10.                     cy.wait(`@${field}`).its('response.statusCode')
11.                         .should('eq', 200);
12.                 });
13.             });
14.             cy.get(field)
15.                 .parent()
16.                 .within(() =>{
17.                     cy.get('.tt-suggestion').first().click();
18.                 });
19.         }
20.     }
21. }
```

Slika 4.18 *inputSuggestionData funkcija.*

```
1.  function inputFieldData(inputFields: any, data: any) {
2.      for (let key in inputFields) {
3.          let field = inputFields[key];
4.          if(data[key] && data[key].toString().includes('#date')) {
5.              let days = (data[key].split(' '))[1];
6.              if (!days) days = 0;
7.              data[key] = new Date(new Date().getTime() + (days * 24 * 60 * 60 *
    1000)).toLocaleDateString(Cypress.env('locale'));
8.          }
9.          if(data[key] != "" && (key in data)) {
10.             cy.get(field).type(data[key].toString().replace('#uniqeID',
    (Math.random() + 1).toString(36).substring(2)));
11.         }
12.     }
13. }
```

Slika 4.19 *inputFieldData funkcija.*

checkURL (**Slika *4.20***) – uspoređuje trenutni URL stranice s parametrom koji se proslijedi.

```
1.  /**
2.   * Compares current page URL with passed one.
3.   * @param urlToCompare - URL string for comparing
4.   * Last update: 11.8.2022.
5.   */
6.  Cypress.Commands.add('checkUrl', (urlToCompare: string) => {
7.      cy.url().should('eq', navigationData.baseUrl + urlToCompare);
8.  });
```

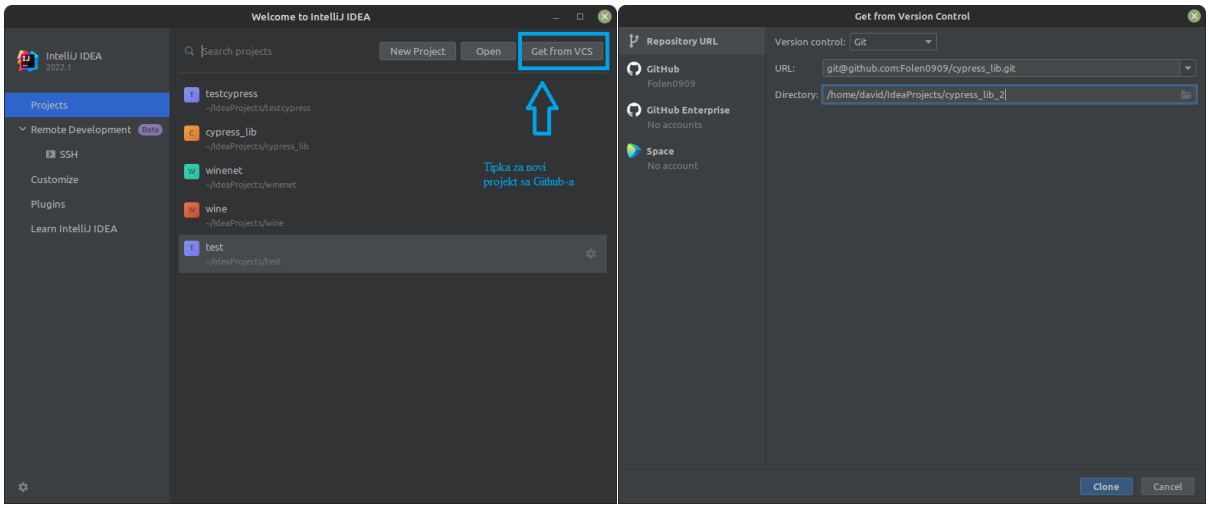
Slika 4.20 *checkURL funkcija.*

4.3 Primjena

U ovom poglavlju prikazana je primjena ranije navedenih funkcija dostupnih unutar biblioteke razvijene u sklopu ovog diplomskog rada. Svi primjeri korištenja realizirani su na stvarnom projektu.

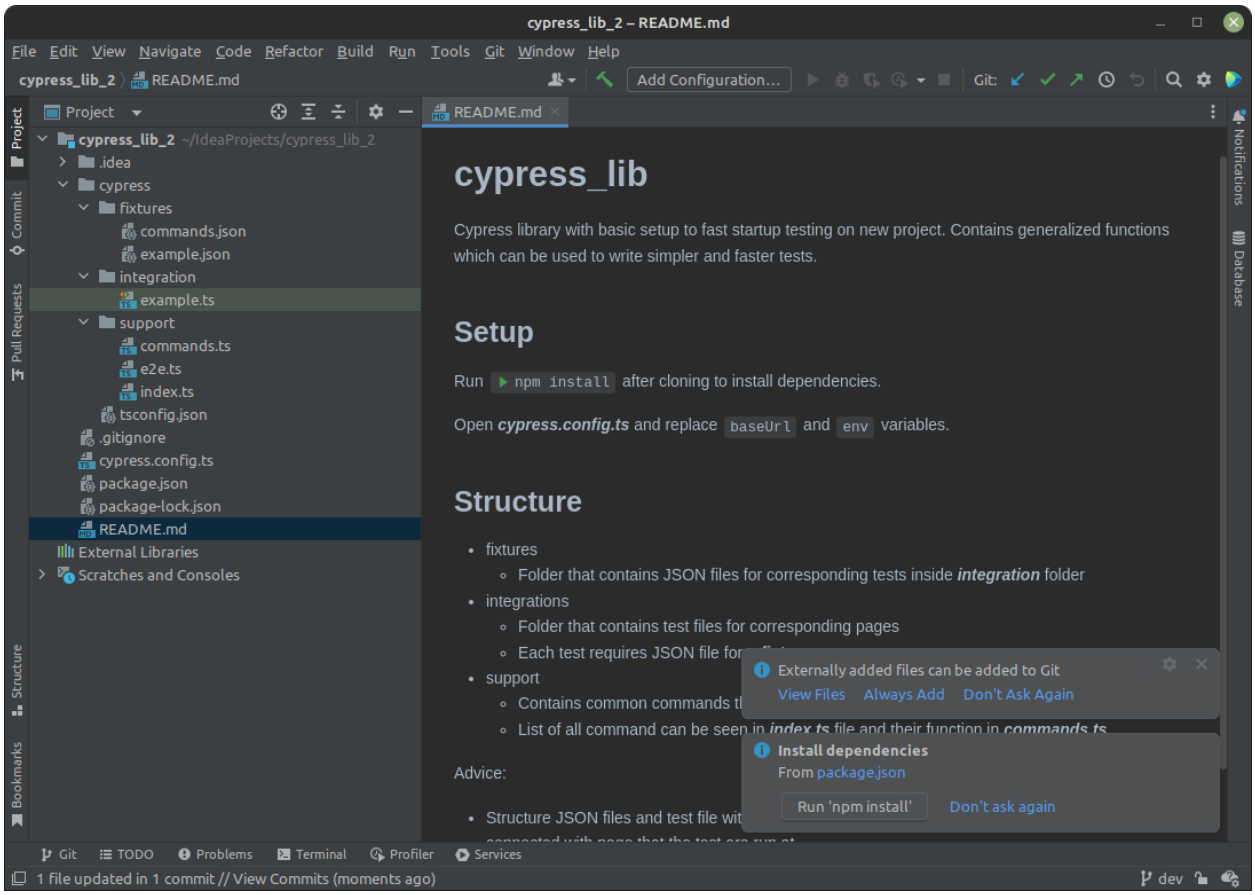
4.3.1 Postavljanje projekta

Preuzimanje projekta s *Github*-a u odabrano razvojno okruženje. U ovom slučaju koristi se *IntelliJ IDEA*. To je proizvod *Jet Brains* organizacije [11]. Postupak preuzimanja projekta s *Github*-a je prikazan na Slika 4.21.



Slika 4.21 *Novi projekt unutar IntelliJ IDEA i preuzimanje Github repozitorija.*

Nakon preuzimanja projekta koji predstavlja implementaciju biblioteke, prikazuje se početni zaslon aplikacije (Slika 4.22). Na njemu se mogu vidjeti razne informacije o projektu: struktura projekta, *README.md* datoteka otvorena u prozoru, prijedlog naredbi koje se mogu izvesti i slično.



Slika 4.22 *Početni prozor IntelliJ IDEA nakon pokretanja projekta.*

4.3.2 Pisanje testnih slučajeva

Testiranje se obavlja na stvarnom projektu te se mora poštivati sigurnost i privatnost podataka. Funkcije koje se ovdje prikazuju, kao i podaci koji će biti korišteni su modificirani. Struktura podataka ostaje ista bez obzira na projekt. To je rezultat standardizacije koja je uvedena u projekt. Prvo su odrađene funkcije koje se najviše koriste i koje su najkritičnije na projektu. Nakon odrađenog ručnog testiranja stranice i točno definiranih ciljeva automatizacije, slijedi prikupljanje podataka u JSON datoteke, unutar mape *fixtures*.

Prvo se testira prijava korisnika jer je to preduvjet za daljnje kretanje kroz aplikaciju, odnosno provođenje ostalih testova. Zbog sigurnosti podataka tijekom testiranja, podaci za prijavu korisnika (email i lozinka) su učitani u *Cypress* alat tijekom pokretanja. Skupljeni podaci se zapisuju unutar JSON datoteke (Slika 4.23) dok se testni slučajevi zapisuju unutar mape *integration* u datoteku naziva *login.ts* (Slika 4.24).

```
{
    "submitButton": "button",
    "logoutButton": "Abmelden",
    "alertDiv": ".alert",
    "successfulLoginTextContainer": ".navbar-text",
    "messages": {
        "wrongEmailError": "Die Anmeldung ist fehlgeschlagen! Der Benutzername ist ungültig.",
        "wrongPasswordError": "Die Anmeldung ist fehlgeschlagen! Das Passwort ist ungültig.",
        "successfulLoginUser": "Angemeldet als"
    },
    "form": {
        "inputFields": {
            "email": "input[name='userid']",
            "password": "input[name='password']"
        }
    },
    "testData": {
        "wrong": {
            "email": "wrong@mail.com",
            "password": "wrongPassword"
        },
        "empty": {
            "email": "",
            "password": ""
        }
    }
}
```

Slika 4.23 *Podaci za login testne slučajeve.*

```
1.  import * as loginData from '../../fixtures/login/login.json';
2.  import {inputData} from '../../support/common';
3.
4.  describe('Log in', () => {
5.
6.      const email = Cypress.env('email');
7.      const password = Cypress.env('password');
8.
9.      const submitButton = loginData.submitButton;
10.     const alertDiv = loginData.alertDiv;
11.     const successfulLoginTextContainer = loginData.successfulLoginTextContainer;
12.
13.     beforeEach(() => {
14.         cy.visitBaseUrl();
15.     });
16.
17.     it('Contains logo', () => {
18.         cy.get('.col-xs-12')
19.             .find('div')
20.             .find('img[src="/imgs/Logo_200px.jpg"]');
21.     });
22.     it('Incorrect data gives error', () => {
23.         inputData(loginData.form, loginData.testData.wrong);
24.         cy.get(submitButton).click();
25.         cy.get(alertDiv).invoke("text").should('eq',
    loginData.messages.wrongEmailError);
26.         cy.get(alertDiv).invoke("text").should('eq',
    loginData.messages.wrongPasswordError);
27.     });
28.     it('Correct email and password results in successful login', () => {
29.         inputData(loginData.form, {email, password});
30.         cy.get(submitButton).click();
31.         cy.get(successfulLoginTextContainer).should('contain.text',
    loginData.messages.successfulLoginUser);
32.     });
33. });
```

Slika 4.24 *Automatizirani testni slučajevi za prijavu korisnika, datoteka login.ts.*

Nakon testiranja funkcionalnosti prijave u aplikaciju, testira se navigacija po stranici te se uspoređuje (Slika 4.25). Za potrebe ovog testa koristi se modificirana verzija funkcije *navigateMenu* (Slika 4.26). Modifikacija se sastoji od dodavanja dijela programskog koda za dohvaćanje elementa u kojem se nalazi navigacija za cijelu stranicu.

```
1.  navigate(navigationData.menu, [])
2.
3.  function navigate(menu, items) {
4.      menu.forEach((item) => {
5.          it(item.name, () => {
6.              cy.menuNavigate(items.concat(item.items));
7.              cy.checkUrl(item.expectedUrl);
8.          });
9.          navigate(item.menu, items.concat(item.items));
10.     });
11. }
```

Slika 4.25 *Funkcija za navigaciju po svakoj stranici i provjera pripadajućeg URL-a.*

```
1.  /**
2.   * Use for navigating menu.
3.   * @param menuItemList - list of elements to navigate through
4.   * Last update 22.8.2022.
5.   */
6.  Cypress.Commands.add('navigateMenu', (menuItemList: string[]) => {
7.      const clonedList  = Object.assign([], menuItemList);
8.      cy.get('div[id^=navbar')
9.          .within($nav => {
10.         if ($nav.find('.dropdown').length == 1)
11.             cy.get(navigationData.dropDownMenu).click();
12.         else
13.             delete clonedList[0];
14.         })
15.         .then(() => {
16.             clonedList.forEach(element => {+
17.                 cy.contains(new RegExp("^" + element + "$", "g")).click();
18.         });
19.     });
20. });
```

Slika 4.26 *Modificirana funkcija navigateMenu.*

Podaci za testiranje navigacije moraju biti složeni po određenoj strukturi kako bi funkcija mogla ispravno odraditi sve implementirane korake. Funkcija je implementirana na način da se poziva rekurzivno pa stoga podaci moraju biti složeni na odgovarajući način.

```
{
  "name": "Artikel",
  "items": [
    "Artikel"
  ],
  "expectedUrl": "/menuitem-assetmgmt_otherarticle",
  "menu": [
    {
      "name": "Fremdartikel",
      "items": [
        "Fremdartikel"
      ],
      "expectedUrl": "/menuitem-assetmgmt_otherarticle_foreign",
      "menu": []
    },
    {
      "name": "Umlagerung",
      "items": [
        "Umlagerung"
      ],
      "expectedUrl": "/menuitem-assetmgmt_otherarticle_move",
      "menu": [
        {
          "name": "Artikelumlagerung",
          "items": [
            "Artikelumlagerung"
          ],
          "expectedUrl": "/menuitem-assetmgmt_otherarticle_move_article",
          "menu": []
        },
        {
          "name": "Standortumlagerung",
          "items": [
            "Standortumlagerung"
          ],
          "expectedUrl": "/menuitem-assetmgmt_otherarticle_move_location",
          "menu": [
            {
              "name": "Druck",
              "items": [
                "Druck"
              ],
              "expectedUrl": "/menuitem-
assetmgmt_otherarticle_move_location_print",
              "menu": []
            }
          ]
        }
      ]
    },
    {
      "name": "Inventurdifferenz \\+/-",
      "items": [
        "Inventurdifferenz \\+/-"
      ],
      "expectedUrl": "/menuitem-assetmgmt_otherarticle_stocktakingdifference",
      "menu": []
    }
  ]
}
```

Slika 4.27 *Skraćeni podaci korišteni za navigaciju.*

Uz navigaciju po svim elementima, razvijena je i metoda za navigaciju do određene stranice. Implementirana je tako da funkcije prolazi kroz sve elemente unutar podataka za navigaciju i pretražuje po nazivu stranice (Slika 4.28).

```
1.  function navigateToPage(name) {
2.      return _navigateToPage(navigationData.menu, name, []);
3.  }
4.  function _navigateToPage(menu, name, matched) {
5.      let match = [].concat(matched);
6.      match = match.filter((item, pos) => {match.indexOf(item) === pos});
7.      menu.forEach(item => {
8.          if (!item.menu.isEmpty) {
9.              _navigateToPage(item.menu, name, match)
10.         }
11.         item.menu.map(ele => { ele.name === name && match.push(ele)});
12.     });
13.     return match;
14. }
```

Slika 4.28 *Funkcija za navigaciju do određene stranice.*

Funkcionalnost koja se najviše testirala na projektu su CRUD[2] operacije. Zbog standardizacije svi testove su slični te prikaz jednog je dovoljan. Primjer se može vidjeti na Slika 4.29 i Slika 4.30.

[2] CRUD (engl. *Create, Read, Update, Delete*) – stvori, pročitaj, ažuriraj, obriši.

```
1.  import { signOut} from "../../support/login";
2.  import * as navigationData from "../../fixtures/navigation/navigation.json";
3.  import * as orderLineData from "../../fixtures/procurement/orderLine.json";
4.  import {clearFields, inputData} from "../../support/common";
5.  import * as returnSupplyData from "../../fixtures/procurement/returnSupply.json";
6.
7.  function inputDataReservation(testData) {
8.      if (testData.invoiceExternal != '') {
9.          clearFields(orderLineData.orderLine);
10.         inputData(orderLineData.orderLine, testData);
11.     }
12.     cy.get(orderLineData.form).contains(orderLineData.orderLine.saveButton).click()
    ;
13.     cy.get(".modal-dialog").within(() => {
14.        cy.contains("Ja").click() ;
15.     });
16.     if (testData.invoiceExternal != '') {
17.         cy.intercept('GET', '/ajax/dataTables**').as('datatable');
18.         cy.wait('@datatable');
19.     }
20. }
21. function checkErrorMessages() {
22.     cy.contains(orderLineData.messages.wrongInputGeneral).invoke('text').should('eq
    ', orderLineData.messages.wrongInputGeneral);
23. }
24. function checkCreateMessage() {
25.     cy.contains(orderLineData.messages.successfullyCreated).invoke('text').should('
    contain', orderLineData.messages.successfullyCreated);
26. }
27. function selectOrdersForInvoice(type: string, alias: string) {
28.     cy.intercept('GET', '/ajax/dataTables**').as(alias);
29.     cy.wait('@' + alias);
30.     cy.get(returnSupplyData.invoice.tbody).within(() => {
31.         cy.get(orderLineData.invoiceCheckBox).first().click();
32.         if (type == "net") {
33.             cy.get(orderLineData.actions.netInvoice).first().click();
34.         }
35.         else {
36.             cy.get(orderLineData.actions.grossInvoice).first().click();
37.         }
38.
39.     });
40. }
41. after(() => {
42.     signOut();
43. });
44. describe('Net orderLine', () => {
45.     beforeEach( () => {
46.         cy.login();
47.         cy.visit(navigationData.baseUrl + navigationData.orderLine.expectedUrl);
48.     });
49.     it('All empty fields', () => {
50.         selectOrdersForInvoice('net', 'empty');
51.         inputDataReservation(orderLineData.testData.allEmpty);
52.         checkErrorMessages();
53.     });
54.     it('Correct data', () => {
55.         selectOrdersForInvoice('net', 'correct');
56.         inputDataReservation(orderLineData.testData.correctData[0]);
57.         checkCreateMessage();
58.     });
59. });
60. describe('Gross orderLine', () => {
61.     beforeEach( () => {
62.         cy.login();
63.         cy.visit(navigationData.baseUrl + navigationData.orderLine.expectedUrl);
```

Slika 4.29 *Primjer CRUD operacija.*

```
{
  "navigationTab": "nav",
  "newBill": "Neue Rechnung",
  "allBills": "Alle Rechnungen",
  "page": "assetmgmt_selling_sell_fromorder_length'",
  "searchInput": "input[type='search']",
  "thead": "thead",
  "tbody": "tbody",
  "form": "form[id='inputform']",
  "messages": {
    "successfullyCreated": "Die Rechnung mit der Rechnungsnummer ",
    "wrongInputGeneral": "Aufgrund fehlender oder falscher Eingaben konnte nicht gespeichert werden! Überprüfen Sie Ihre Eingaben und folgen Sie den Aufforderungen zur Fehlerbehebung.",
    "wrongInvoiceDate": "Das Rechnungsdatum für einen Barverkauf muss nach dem letzten Tagesabschluss (29.05.2022) liegen."
  },
  "sorting": {
    "invoiceRecipient": "Rechnungsempfänger"
  },
  "assignments": {
    "type": "Art",
    "order": "Auftrag",
    "date": "Datum",
    "deliveryDate": "Lieferdatum",
    "isDelivered": "Geliefert?",
    "total": "Bruttosumme",
    "isInvoice": "Rechnung?"
  },
  "invoiceCheckBox": "input[name='sorder']",
  "actions": {
    "netInvoice": "a[title='Netto-Rechnung zu den gewählten Aufträgen erfassen']",
    "grossInvoice": "a[title='Brutto-Rechnung zu den gewählten Aufträgen erfassen']"
  },
  "orderLine": {
    "inputFields": {
      "invoiceExternal": "input[name='externalnumber']",
      "deliveryDate": "input[name='deliverydate']",
      "invoiceDate": "input[name='invoicedate']",
      "totalDiscountPercent": "input[name='globaldiscountperc']",
      "totalDiscountAmount": "input[name='globaldiscountamount']",
      "discountDays": "input[name='cashback1days']",
      "discountValue": "input[name='cashback1perc']",
      "paymentTerm": "input[name='duedays']",
      "paymentComment": "input[name='duedayscomment']",
      "note": "textarea[name='description']"
    },
    "checkFields": {
      "totalDiscAsPer": "input[name='globaldiscountuseperc']",
      "intraSupply": "input[name='intraeu']",
      "cashSale": "input[name='cashsale']",
      "taxable": "input[name='sparklingwinetaxed']"
    },
    "selectFields": {
      "paymentMethod": "select[name='paymentmethod']",
      "accountDetails": "select[name='bankaccount']"
    },
    "saveButton": " Speichern",
    "cancelButton": " Abbrechen",
    "preview": " Vorschau "
  },
  "testData": {
    "correctData": [
      {
        "invoiceExternal": "1",
```

Slika 4.30 *Primjer JSON datoteke za testiranje CRUD operacija.*

5 ZAKLJUČAK

Testiranje softvera dugotrajan je i skup proces. Izbor načina testiranja softvera se i dalje svodi na jedan od dva načina. Iako se vodi rasprava koji je način bolji za primijeniti, ne može se zaboraviti da je glavno pitanje tijekom izbora, koji je cilj testiranja. Oba načina imaju svoje prednosti i mane koje se ne mogu zanemariti tijekom odabira. Jedna od glavnih mana automatiziranog testiranja je što se zapravo ne može ispravno primijeniti bez ručnog testiranja. Istina je da se testiranjem softvera može poboljšati kvaliteta softvera, ali treba biti oprezan, uz iscrpno testiranje dolaze veliki troškovi. Cilj testiranja softvera je dostaviti korisniku proizvod bez greške, dok je cilj razvojnog tima što bolje napraviti zadani posao u što kraćem vremenu. Ukoliko se postigne ravnoteža između ta dva cilja, zadovoljstvo je moguće vidjeti na obje strane poslovanja.

Ručno testiranje može postati ne moguće za izvođenje. Što softver više raste i postaje kompleksniji to je posao testera sve teži i duži. Kako bi se olakšalo testiranje, uvodi se automatizirano testiranje. Za automatizirano testiranje je potreban alat. Danas postoji veliki izbor moćnih alata, s različitim značajkama, ali s istim globalnim ciljem, olakšati postupak dostavljanje softvera bez greške krajnjem korisniku. U ovom radu spominje ih se nekoliko, ali glavni naglasak se daje *Cypress* alatu za automatizirano testiranje. *Cypress* je noviji alat koji pokušava što više pojednostaviti i ubrzati razvoj automatiziranih testnih slučajeva. Ima mnogo značajki s kojima to ostvaruje. Iako je relativno nov, brzim razvojem i motiviranim timom, implementira nove značajke koje pomaže još sve lakšem i brže razvoju testova.

Cypress u sebi sadrži osnovne metode koje se mogu koristi pisanje raznih testnih slučajeva. No kada se radi na većem projektu koji se sastoji od mnoštvo sličnih funkcionalnosti, dobro je izdvojiti neke posebne skupine metoda kao dodatne funkcije. Skup tih dodatnih funkcija je biblioteka. Ovaj diplomski rad opisuje jednu takvu biblioteku. Naravno da uz prednosti bržeg i jednostavnijeg pisanja testnih slučajeva postoji i nekoliko nedostataka. Velike i kompleksne funkcije je nemoguće standardizirati. Svaki softver koji se testira uvijek ima nešto drugačije te je potrebna modifikacija funkcija kako bi funkcionirale.

Naravno tijekom razvoja biblioteke vodilo se računa o budućim unaprjeđenjima. Biblioteka je dizajnirana tako da se vrlo jednostavno može proširiti s dodatnim funkcijama. Osim ideje jednostavne integracije novih funkcija, dodatna je ideja integracija *Node.js* upravitelja paketima radi lakše integracije u nove projekte.

6 LITERATURA

[1] G. Myers, The Art of Software Testing, World Association inc., 2004.

[2] D. Kumar, »Seven Principles of software testing,« GeeksforGeeks, [Mrežno]. Available: https://www.geeksforgeeks.org/software-engineering-seven-principles-of-software-testing/. [Pokušaj pristupa 7 Rujan 2022.].

[3] J. Schmitt, »What is end-to-end testing?,« circleci.com, 5 Travanj 2022.. [Mrežno]. Available: https://circleci.com/blog/what-is-end-to-end-testing. [Pokušaj pristupa 6 Rujan 2022.].

[4] Katalon, »How to Select the Right Automation Testing Tool,« Katalon. [Mrežno]. [Pokušaj pristupa 9 Rujan 2022.].

[5] Katalon, »Top 15 List of Automation Testing Tools | Latest Update in 2022,« Katalon, 2022.. [Mrežno]. Available: https://katalon.com/resources-center/blog/automation-testing-tools. [Pokušaj pristupa 9 Rujan 2022.].

[6] Snyk, »What is the MIT License?,« Snyk, [Mrežno]. Available: https://snyk.io/learn/what-is-mit-license/. [Pokušaj pristupa Rujan 2022.].

[7] Cypress, »Why Cypress?,« Cypress, [Mrežno]. Available: https://docs.cypress.io/guides/overview/why-cypress. [Pokušaj pristupa Rujan 2022.].

[8] Cypress, »Table of Contents,« Cypress, 2022.. [Mrežno]. Available: https://docs.cypress.io/api/table-of-contents. [Pokušaj pristupa Rujan 2022.].

[9] Cypress, »Changelog,« Cypress, 30 Kolovoz 2022. [Mrežno]. Available: https://docs.cypress.io/guides/references/changelog#7-0-1. [Pokušaj pristupa Rujan 2022.].

[10] Github, »Github,« Github, [Mrežno]. Available: https://github.com/. [Pokušaj pristupa Rujan 2022.].

[11] JetBrains, »IntelliJ IDEA,« JetBrains, [Mrežno]. Available: https://www.jetbrains.com/idea/. [Pokušaj pristupa Rujan 2022.].

[12] L. Crispin i J. Gregory, Agile Testing, Addison-Wesley, 2009.

[13] M. Pezzè i M. Young, Software Testing and Analysis: Process, Principals, and Techniques, Wiley, 2008.

[14] M. Robert, Softversko inženjerstvo, 2005.

[15] L. Pearson, »The Four Levels of Software Testing,« Segue Technologies Inc, 11. Rujan 2015.. [Mrežno]. Available: https://www.seguetech.com/the-four-levels-of-software-testing/. [Pokušaj pristupa 1 Rujan 2022.].

[16] UTOR, »Manual vs Automation Testing: What to Choose For Your Project,« UTOR, 15 Siječanj 2020.. [Mrežno]. Available: https://u-tor.com/topic/manual-vs-automation. [Pokušaj pristupa 3 Rujan 2022.].

[17] G. C. Reddy, »Advantages and Disadvantages of Manual Testing,« G C Reddy Technologies, 31 Srpanj 2021.. [Mrežno]. Available: https://www.gcreddy.com/2021/07/advantages-and-disadvantages-of-manual-testing.html. [Pokušaj pristupa 6 Rujan 2022.].

[18] A. Khetarpal, »What is Cypress: Introduction and Architecture,« Tools QA, 20 Listopad 2021.. [Mrežno]. Available: https://www.toolsqa.com/cypress/what-is-cypress/. [Pokušaj pristupa Rujan 2022.].

[19] A. Khetarpal, »Cypress Test,« Tools QA, 23 Kolovoz 2021.. [Mrežno]. Available: https://www.toolsqa.com/cypress/cypress-test/. [Pokušaj pristupa Rujan 2022.].

[20] Brian, »Best Automation Testing Tools for 2022 (Latest Update),« Medium.com, 26 Listopad 2017.. [Mrežno]. Available: https://briananderson2209.medium.com/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2. [Pokušaj pristupa Rujan 2022.].

7 SAŽETAK

Testiranje softvera postupak je utvrđivanja da softver ne sadrži greške. Dvije su vrste, ručno i automatizirano. Ručno testiranje softvera razvojem dodatnih značajki softvera postaje neisplativo i dugotrajno; kako bi se skratilo vrijeme testiranja uvodi se automatizirano testiranje. Za razvoj automatiziranih testova potreban je alat. Cypress je alat za razvoj automatiziranih testova i automatizirano testiranje internet aplikacija i stranica. U cypress alatu razvijena je biblioteka s dodatnim funkcijama za brži i lakši razvoj automatiziranih testnih slučajeva. Korištenje je prikazano na primjeru internet aplikacije.

8 ABSTRACT

Cypress environment for automated testing of Internet applications

Software testing is an act of determining that software does not contain errors. There are two types of testing, manual and automated. Manual software testing becomes unprofitable and takes too long to execute while software gets new features. To shorten time needed for testing, automated testing is introduced. For developing automated test there is a need of a tool. Cypress is a tool for developing and running automated tests for Internet apps and websites. Cypress is used to develop new library of advanced function that can be used to speed up and ease up development of automated tests. Example of its use is shown on Internet app.

9 ŽIVOTOPIS

David Turkalj rođen 9. rujna 1997. godine u Vinkovcima, s prebivalištem u Rokovcima. Osnovnu školu Ivane Brlić-Mažuranić završava 2012. godine. Iste godine upisuje Tehničku školu Ruđera Boškovića Vinkovci u Vinkovcima, smjer tehničar za mehatroniku. Godine 2016. upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, završava preddiplomski sveučilišni studij računarstva te upisuje diplomski sveučilišni studij računarstva, smjer programski inženjer.

_____

Potpis autora